James F. Kurose, Keith W. Ross

Computer Networking

A *Top-Down Approach Featuring the Internet*

Second Edition

Boston San Francisco Ney York
London Toronto Sydney Tokio Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Джеймс Ф. Куроуз, Кит В. Росс

Компьютерные сети

Многоуровневая *архитектура Интернета*

2-е издание

Москва - Санкт-Петербург - Нижний Новгород - Воронеж
Ростов-на-Дону • Екатеринбург - Самара - Новосибирск
Киев - Харьков - Минск
2004

ББК 32.988.02я7
УДК 681.324(075)
К93

Куроуз Дж., Росс К.

К93 Компьютерные сети. 2-е изд. — СПб.: Питер, 2004. — 765 с: ил.

ISBN 5-8046-0093-1

Эта книга посвящена сетевым компьютерным технологиям — динамично развивающейся области вычислительной техники, и охватывает весьма широкий круг вопросов: архитектуру компьютерных сетей, протоколы, приложения, администрирование, работу с мультимедиа, а также безопасность и защиту информации. При этом полнота изложения материала сочетается с множеством наглядных аналогий и примеров, взятых из реальной жизни.

Книга будет полезна как студентам и преподавателям, специализирующимся в области сетевых компьютерных технологий, так и всем, кто интересуется компьютерными сетями.

ББК 32.988.02я7
УДК 681.324(075)

ISBN 0-201-97699-4 (англ.)
ISBN 5-8046-0093-1

Краткое содержание

Содержание

Об авторах

Джеймс Куроуз

Джим Куроуз является профессором кафедры кибернетики Массачусетского университета в Амхерсте.

Восемь раз он получал премию Outstanding Teacher Award от Национального технологического университета. Один раз эту премию ему вручил Колледж естественных наук и математики при Массачусетском университете. В 1996 году награду Outstanding Teaching Award ему присудила Северо-Восточная ассоциация аспирантур. Он был стипендиатом General Electric и Lilly Teaching. Кроме того, ему присуждалась премия IBM Faculty Development Award.

Доктор Куроуз является бывшим главным редактором IEEE Transactions on Communications и IEEE/ACM Transactions on Networking. Он принимает активное участие в программных комитетах IEEE Infocom, ACM SIGCOMM и ACM SIGMETRICS. Степень доктора философии в области кибернетики он получил в Колумбийском университете.

Кит Росс

Кит Росс — профессор кафедры мультимедийных коммуникаций института Eurecom. С 1985 по 1997 год он был профессором Университета Пенсильвании, где занимал должности на кафедре системного проектирования и в Школе бизнеса Уортона. В 1999 году он принял участие в основании Интернет-компании Wimba.com.

Доктор Росс является автором более 50 статей и двух книг. Он работал редактором пяти крупных журналов, а также включался в комитеты программирования крупных сетевых конференций, в том числе IEEE Infocom и ACM SIGCOMM. Доктор Росс был руководителем более десятка диссертаций на степень доктора философии. В сферу его исследовательских и преподавательских интересов входят мультимедийные сети, асинхронное обучение, web-кэширование, потоковые аудио- и видеоприложения, а также моделирование трафика. Степень доктора философии он получил в Университете штата Мичиган.

Предисловие

Вы держите в руках второе издание книги. Как показали два года, прошедшие после выхода первого издания, читателями этой книги стали десятки тысяч студентов и практикантов, обучающихся в сотнях колледжей и университетов. За это время мы получили множество писем о нашем издании и были приятно удивлены обилием положительных откликов.

Секрет успеха этой книги, как нам кажется, заключается в новом подходе к изучению компьютерных сетей. Для чего нужен новый подход? В последние годы в сфере компьютерных сетей произошли две революции. Первая революция обусловлена широчайшей популярностью, пришедшей к Интернету. В настоящее время любой серьезный разговор о компьютерных сетях происходит под влиянием Интернета. Вторая революция охватила сетевые службы и приложения. Это также связано с развитием Интернета, в частности web, электронной почты, потокового аудио и видео, Интернет-телефонии, систем обмена сообщениями в реальном времени, одноранговых приложений и т. д.

Новое во втором издании

Несмотря на некоторые нововведения, появившиеся во втором издании, мы сохранили наиболее важные аспекты нашего подхода к изложению материала: изучение сетей «сверху вниз», ориентация на Интернет, равное внимание как к теоретическим положениям, так и к практическим нюансам, а также доступный широкой аудитории стиль. В основном изменения, сделанные в настоящем издании, обусловлены динамичным развитием сетевых технологий в последние годы. Так, мы уделили повышенное внимание одноранговым сетям, сетям распределения ресурсов, Интернет-телефонии, сетевой безопасности и многим другим вопросам. Благодаря откликам наших читателей мы внесли некоторые изменения в первоначальный текст и ссылки, использованные в первом издании. Кроме того, в книгу включены несколько вопросов для самостоятельного решения, а также дополнительные задания (в частности, вашему вниманию будет предложена новая работа с потоковым видео по протоколам RTP и RTSP).

Требования к аудитории

Эта книга с успехом может использоваться в качестве учебника для вводного курса по компьютерным сетям. При этом она будет одинаково полезной студентам, специализирующимся в области вычислительной техники или электротехники. В отношении навыков программирования обязательным является владение хотя бы одним из языков C, C++ и Java. При этом мы особо подчеркиваем, что знание Java не обязательно, и студенты, знакомые с C и C++, будут чувствовать себя вполне комфортно при изучении глав, посвященных созданию сетевых приложений на Java. Несмотря на повышенное внимание к анализу и точности, при изложении материала вы не встретите математических концепций, которые не были бы знакомы студенту технического профиля. Создавая эту книгу, мы старались избегать сложных расчетов, теории вероятностей и стохастических процессов. Таким образом, даже студенты младших курсов без проблем смогут справиться с материалом книги. Кроме того, книга будет весьма полезна специалистам, работающим в области телекоммуникаций.

Уникальность книги

Компьютерные сети — чрезвычайно сложный и запутанный предмет, включающий в себя множество концепций, протоколов и технологий, для новичка загадочно переплетенных между собой. Для того чтобы преодолеть трудности в освоении столь сложной системы, в учебных изданиях авторы нередко опираются на многоуровневую архитектуру сети. Это позволяет студентам последовательно изучать различные концепции и протоколы и в то же время обращаться к общей картине, чтобы получить представление о связях между концепциями. Нередко структура учебных пособий основана на семиуровневой модели OSI. Как показал наш педагогический опыт, такая структура оказывается весьма эффективной, однако мы не придерживаемся ставшего традиционным подхода, заключающегося в изучении компьютерных сетей «снизу вверх». Далее мы поясним свою точку зрения.

Подход «сверху вниз»

Под изучением сетей «снизу вверх» мы понимаем последовательное изложение материала от самого низкого, физического, уровня модели OSI к прикладному уровню. Подход «сверху вниз» предполагает противоположное направление движения: от прикладного уровня к физическому. Первым несомненным достоинством такого подхода является акцент на прикладном уровне, поскольку именно прикладной уровень представляет собой базу для развития сетевых технологий. Этот тезис был выдвинут нами при подготовке первого издания книги, и время в полной мере подтвердило нашу правоту. В то же время мы столкнулись с множеством учебных изданий, в которых сетевым приложениям отводилась далеко не первостепенная роль. Их авторы практически не уделяли внимания потребностям сетевых приложений, парадигме прикладного уровня (клиент/сервер и т. п.), интерфейсам сетевых приложений и т. д.

Кроме того, наш опыт показал, что изучение сетевых приложений в начале курса компьютерных сетей значительно облегчает понимание материала студентами. Многие из них ежедневно сталкиваются с приложениями для web и электронной почты. Освоив механизм работы приложений, студенты, как правило, без труда переходят к вопросам поддержки этих приложений. Таким образом, постепенно спускаясь «вниз» компьютерной сети, студенты углубляют свои знания о принципах ее функционирования.

Третьим достоинством подхода «сверху вниз» является то, что преподаватель может на ранних этапах курса вводить студентов в процесс разработки сетевых приложений. Располагая знаниями о взаимодействии приложений и протоколов, студенты без особого труда могут начать разработку собственных приложений и протоколов для компьютерных сетей. В этой книге мы используем простые примеры программирования сокетов на языке Java, которые демонстрируют основные особенности сетевых приложений. Таким образом, с помощью этой книги студенты могут ознакомиться с понятиями прикладного программного интерфейса (API), моделей обслуживания и протокола.

Ориентация на Интернет

В большинстве существующих на сегодняшний день изданий Интернет рассматривается лишь как небольшая составная часть сетевых технологий. Мы придерживаемся другого подхода, помещая Интернет в центр внимания и придавая ему функцию «отправной точки» для изучения более фундаментальных принципов построения сетей. У некоторых читателей сразу возникнет вопрос: почему Интернет, ведь существует множество сетевых технологий, например ATM? Прежде всего, наш выбор обусловлен широкой распространенностью Интернета: в последние годы термины «сетевые технологии» и «Интернет-технологии» стали синонимами. Еще 5-10 лет назад ситуация вокруг Интернета была совершенно иной. Более того, велось множество разговоров о локальных вычислительных сетях, ATM и приложениях, напрямую взаимодействующих с ATM (без применения TCP/IP). В настоящее время ситуация такова, что почти весь трафик данных, существующий в мире, приходится на Интернет и интрасети. Пожалуй, лишь коммутируемые телефонные сети на сегодняшний день сравнимы с Интернетом по распространенности. Но и эта конкуренция оказывается весьма призрачной. Производители сетевого оборудования и операторы телефонной связи готовятся к переориентации на Интернет-технологии.

Еще одной причиной, побудившей нас сфокусировать внимание на Интернете, явился большой интерес студентов к глобальной Сети и ее протоколам. Большинство современных студентов являются ежедневными пользователями Интернета: как минимум, они используют web и электронную почту. Разумеется, многие испытывают интерес к устройству Интернета. Таким образом, преподаватель без труда может увлечь студентов Интернетом, параллельно знакомя их с фундаментальными принципами организации компьютерных сетей.

Поскольку наша книга основана на Интернет-технологиях, ее структурным стержнем является пятиуровневая архитектура, а не семиуровневая модель OSI. В Ин-

тернет-технологиях различают прикладной, транспортный, сетевой, канальный и физический уровни.

Упор на основополагающие принципы

Помимо двух ключевых фраз, отражающих суть книги (подход «сверху вниз» и Интернет), третьей ключевой фразой (точнее, словом), незримо присутствующей в структуре книги, является слово «принципы». В настоящее время сетевые технологии находятся на той стадии развития, когда можно вести речь о принципах организации различных уровней сетевых моделей. Так, например, к принципам работы транспортного уровня относятся установление и разрыв логического соединения, управление потоками данных, мультиплексирование. Принципы функционирования сетевого уровня включают маршрутизацию, то есть выбор оптимального пути передачи сообщения и передачу данных между разнородными компьютерными сетями. Главной проблемой, разрешаемой на канальном уровне, является организация множественного доступа к каналу связи. Особо стоит отметить вопросы, касающиеся безопасности компьютерных сетей, методов обеспечения конфиденциальности передаваемых данных и целостности пакетов данных; как правило, эти вопросы лежат в области криптографии. В данной книге вы найдете не только описания перечисленных проблем, но и подходы, применяющиеся для их решения.

Web-сайт

У этой книги есть расширенная и существенно более функциональная электронная версия, расположенная по адресу http://www.aw.com/kurose-ross. Этот сайт мы рекомендуем посетить всем читателям.

- *Интерактивный обучающий материал.* На сайте имеются интерактивные Java-апплеты, призванные проиллюстрировать основные сетевые концепции. Здесь же можно получить доступ к программе Traceroute, позволяющей с помощью браузера прослеживать маршрут движения пакетов через Интернет. Преподаватели могут использовать эти программы в лабораторных работах. Через сайт можно также получить прямой доступ к специальным поисковым системам, предназначенным для поиска Интернет-проектов и групп новостей, темы которых соответствуют излагаемому здесь материалу. Наконец, на сайте находятся интерактивные испытательные тесты, которые помогут студентам определить уровень своей подготовки.
- *Не менее пятисот ссылок на близкие по теме источники.* Как знает любой энтузиаст Интернета, основная (и лучшая) часть материалов об Интернете находится в самом Интернете. Мы попытались собрать максимальное количество полезных URL-адресов. Список ссылок постоянно обновляется, в него добавляются новые материалы. Сюда включены не только ссылки на документы RFC и статьи из журналов и конференций, но и ссылки на специальные образовательные сайты, в которых имеются страницы, посвященные конкретным аспектам Интернет-технологий, а также публикуются статьи из электронных изда-

ний. Преподаватели могут счесть эти материалы не только полезными, но и необходимыми.

- *Авторский материал для мультимедийных лекций.* На сайте имеются записи лекций **в** формате Real-Audio, которые авторы читают своим студентам.

Мы надеемся, что будем постоянно развивать свой сайт, публикуя на нем собственные и читательские материалы. Обновления планируются не менее чем раз в три месяца. Если у вас при посещении сайта возникнут проблемы, свяжитесь с нами по электронной почте: aw.cse@aw.com.

Педагогические аспекты

Каждый из авторов этой книги ведет профессиональную педагогическую деятельность **в** области компьютерных сетей уже приблизительно в течение 20 лет. За это время нашими выпускниками стали более 3000 инженеров. Кроме того, мы провели значительный объем исследовательских работ. Все это помогло нам хорошо понять предмет изнутри, определить место компьютерных сетей в современной вычислительной технике и оценить перспективы их развития. Должны признаться, что нам не удалось удержаться от искушения включить в книгу материал, явившийся результатом некоторых наших исследований; если вас заинтересует наша научно-исследовательская деятельность, вы можете посетить наш сайт **в** Интернете.

Итак, книга, которую вы держите в руках, посвящена современным сетевым компьютерным технологиям, протоколам и принципам, на которых основаны эти технологии и протоколы. Мы искренне верим, что изучение компьютерных сетей можно превратить в весьма увлекательный и даже забавный процесс. Именно поэтому мы постарались снабдить книгу максимальным количеством аналогий и наглядных примеров из реальной жизни.

История, принципы и практика

История развития компьютерных сетей началась в конце 60-х годов прошедшего столетия и является весьма интересной и насыщенной. В этой книге мы постарались познакомить читателя с наиболее значительными историческими событиями: изобретением пакетной передачи, развитием Интернета, появлением таких сетевых гигантов, как Cisco и 3Com, и т. д. Истории сетей посвящен специальный раздел в главе 1; более краткие исторические справки вы неоднократно можете встретить в других главах книги. Несомненно, для многих читателей освещение прогресса, происходящего в сетевых технологиях, станет хорошим стимулом для изучения предмета.

Историки говорят, что наше прошлое позволяет нам предсказывать будущее. Предсказание будущего, то есть направления развития сетевых технологий, является исключительно важным аспектом в данной области. Это еще одна причина, по которой мы сочли необходимым снабдить книгу информацией ретроспективного характера.

Как упоминалось ранее, в этой книге уделяется равное внимание как принципам, лежащим в основе сетевых технологий, так и их практическому применению. Соответствующие врезки, как и исторические справки, вы можете найти в каждой из глав.

Интервью

В нашу книгу вошло несколько интервью, взятых у видных новаторов в области компьютерных технологий. По нашему мнению, эти интервью будут весьма полезными, поскольку позволят узнать «из первых рук» мнение высокопрофессиональных специалистов. В число наших «интервьюируемых» вошли Леонард Клейнрок, Тим Бернерз-Ли, Салли Флойд, Боб Меткалфи, Хеннинг Шульцринн, Стивен Беллоуин и Джефф Кейз.

Преподавателям

Любое изменение или даже небольшое усовершенствование структуры преподаваемого курса влечет за собой массу забот по его оснащению учебными материалами. В случае, если эта книга окажется полезной для вас именно в этом контексте, мы предусмотрели дополнение к ней, состоящее из следующих частей.

- *Слайды PowerPoint.* На web-сайте вы можете найти слайды PowerPoint с иллюстративным материалом для всех восьми глав книги. Слайды содержат максимум графической информации и анимации. Преподаватели могут изменять эти слайды, приспосабливая их под собственные вкусы и потребности. Авторами немалого количества этих слайдов являются читатели первого издания, которые уже используют его в качестве учебника.
- *Лабораторные задания.* На нашем web-сайте вы также сможете найти несколько детальных описаний лабораторных работ, включающих построение многопоточного web-сервера, клиента электронной почты с графическим интерфейсом, программ передающей и принимающей сторон протокола передачи данных, Интернет-маршрутизатора.
- *Ответы на вопросы и задания.* Наконец, на нашем web-сайте вы найдете ответы на вопросы, задания и упражнения, находящиеся в конце каждой из глав.

Все указанные выше материалы находятся в разделе http://www.aw.com/kurose-ross сайта.

Порядок изложения материала

В главе 1 книги содержится общее описание сетевых компьютерных технологий, приводятся основополагающие концепции и наиболее важные термины. Таким образом, материал главы 1 является основой для понимания всех остальных глав книги. Первые пять глав следует изучать, придерживаясь последовательности изложения материала, поскольку в каждой новой главе используется уже пройденный материал. Кроме того, непоследовательное изучение приведет к нарушению провозглашенного принципа «сверху вниз», что, по нашему мнению, весьма нежелательно.

Иначе обстоит дело с заключительными тремя главами. Между ними нет строгой логической зависимости, и изучать их можно в любом порядке. Тем не менее материал этих глав связан с первыми главами книги. Для преподавателей мы рекомендуем включить в курс лишь отдельные фрагменты заключительных глав в качестве дополнения к основному курсу. Кроме того, следует иметь в виду, что глава 1, будучи простой и самодостаточной, может служить основой для краткого курса компьютерных сетей.

Последнее замечание

Мы в значительной степени поощряем усилия студентов и преподавателей, направленные на создание Java-апплетов на базе излагаемых в этой книге концепций и описываемых протоколов. При желании вы можете прислать нам ваши собственные апплеты, и, если они хорошо будут иллюстрировать материал книги, мы с удовольствием разместим их на нашем web-сайте со ссылками на авторов. Мы также будем признательны преподавателям, которые пришлют нам свои варианты вопросов для самостоятельного решения и ответов к ним. Эти вопросы также появятся на web-сайте с указанием авторов.

И, разумеется, как любые авторы, мы ждем от наших читателей отзывов о книге. Для нас было огромной пользой и удовольствием получать и читать мнения, комментарии, критику студентов и преподавателей, составивших аудиторию первого издания. Мы будем признательны, если вы направите свои отзывы авторам по адресам kurose@cs.umass.edu и ross@eurecom.fr.

Благодарности

С тех пор как мы начали работу над книгой в 1996 году, множество людей оказало нам неоценимую помощь и влияние как на стиль изложения материала, так и на его структуру и содержание. В первую очередь мы хотим выразить общую благодарность всем, кто так или иначе принимал участие в подготовке книги. Мы также очень признательны сотням читателей со всего мира, высказавшим свои суждения, замечания и советы относительно первого издания.

От издателя перевода

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на web-сайте издательства http://www.piter.com.

ГЛАВА 1 Компьютерные сети и Интернет

Компьютерные сети — это одна из самых важных и захватывающих технологий нашего времени. Количество компьютеров, подключенных к глобальной Сети Интернет, измеряется миллионами (и это число постоянно растет); таким образом, Интернет создает глобальную коммуникацию, позволяя огромному числу пользователей обмениваться информацией и задействовать вычислительные ресурсы друг друга. Кроме того, в настоящее время происходит тесная интеграция Интернета с мобильными и беспроводными технологиями, что значительно расширяет круг его функций. За те 40 лет, которые прошли с момента создания первых сетей, сетевые компьютерные технологии значительно усовершенствовались; однако и последние достижения науки говорят о том, что это лишь начало большого пути к неизведанным вершинам техники. Данная книга призвана послужить для читателей проводником в потрясающий и увлекательный мир компьютерных сетей.

Эта глава посвящена обзору компьютерных сетей и Интернета. Цель излагаемого материала — дать общее представление о том, что такое компьютерные сети, какие они бывают и для чего предназначаются. Мы также подробно рассмотрим составляющие компьютерных сетей, стараясь в то же время не потерять общее представление о них. Все, о чем будет сказано в этой главе, ляжет в основу дальнейшего повествования. Эта глава также может использоваться для разработки мини-курса по теме «компьютерные сети».

После того как вы ознакомитесь с первыми наиболее важными понятиями и терминами, мы обратимся к «периферии» компьютерных сетей. Будут рассмотрены оконечные системы и сетевые приложения, а также транспортные услуги, предоставляемые этим приложениям. Затем мы исследуем «ядро» сети — линии связи и коммутаторы, с помощью которых осуществляется передача данных, а также сети доступа и физические устройства, соединяющие оконечные системы с сетью. Мы узнаем, почему Интернет является «сетью сетей» и каким образом эти сети связаны друг с другом.

Изучив «периферию» и «ядро» компьютерной сети, мы несколько расширим предмет рассмотрения. В поле нашего зрения окажутся такие фундаментальные явле-

ния, как задержки и потери данных в сети; мы приведем несложные количественные модели для оценки полных задержек при передаче между оконечными системами, задержки обработки, передачи, распространения и ожидания. Кроме того, мы рассмотрим основополагающие принципы сетевой архитектуры, такие как уровни протоколов и модели обслуживания. Наконец, мы завершим эту главу кратким историческим обзором компьютерных сетей.

Что такое Интернет?

В этой книге мы рассматриваем специфическую компьютерную сеть, которой является Интернет, как некую основу для обсуждения компьютерных сетевых протоколов. Однако что же представляет собой Интернет? Конечно, мы были бы рады привести такое определение Интернета, которое укладывалось бы в одно предложение, легко запоминалось и использовалось бы вами всю оставшуюся жизнь, но... Увы, возможности языка не позволяют совладать с той сложностью и глобальностью, которые присущи Интернету.

Структура Интернета

Вместо краткого определения Интернета постараемся, наоборот, дать максимально подробное описание. Составлять такое описание можно двумя путями. Первый вариант заключается в рассмотрении структурных составляющих Интернета, то есть его аппаратного и программного обеспечения. Второй вариант предполагает описание Интернета в терминах сетевой инфраструктуры, предоставляющей услуги распределенным приложениям. Мы начнем с первого варианта. Обратимся к рис. 1.1.

Интернет представляет собой всемирную компьютерную сеть, то есть сеть, связывающую в единое целое миллионы вычислительных устройств, расположенных в разных уголках земного шара. Вычислительными устройствами могут быть настольные персональные компьютеры, а также так называемые серверы, хранящие и передающие информацию, представленную в виде, например, web-страниц или сообщений электронной почты. В последнее время все чаще к Интернету подключаются такие нетрадиционные оконечные системы, как устройства PDA (Personal Digital Assistant — персональный цифровой помощник), телевизоры, мобильные компьютеры, автомобили и даже тостеры. (Тостер [519] является не единственным необычным устройством, подключенным к Интернету [17].) Так или иначе, все вышеперечисленные устройства в терминологии Интернета называют *хостами,* или *оконечными системами.* По оценкам специалистов, в январе 2002 года в Интернете насчитывалось от 100 до 500 миллионов оконечных устройств, и со временем это число экспоненциально возрастает [240].

Оконечные системы связаны друг с другом *линиями связи.* Как мы увидим позже, существует большое количество различных линий связи, использующих разнообразные типы физических носителей: коаксиальные, медные, волоконно-оптические кабели, линии радиосвязи **PI** т. д. Линия связи определяет скорость передачи данных. Максимальную скорость передачи данных называют *пропускной способностью* линии и измеряют в битах в секунду.

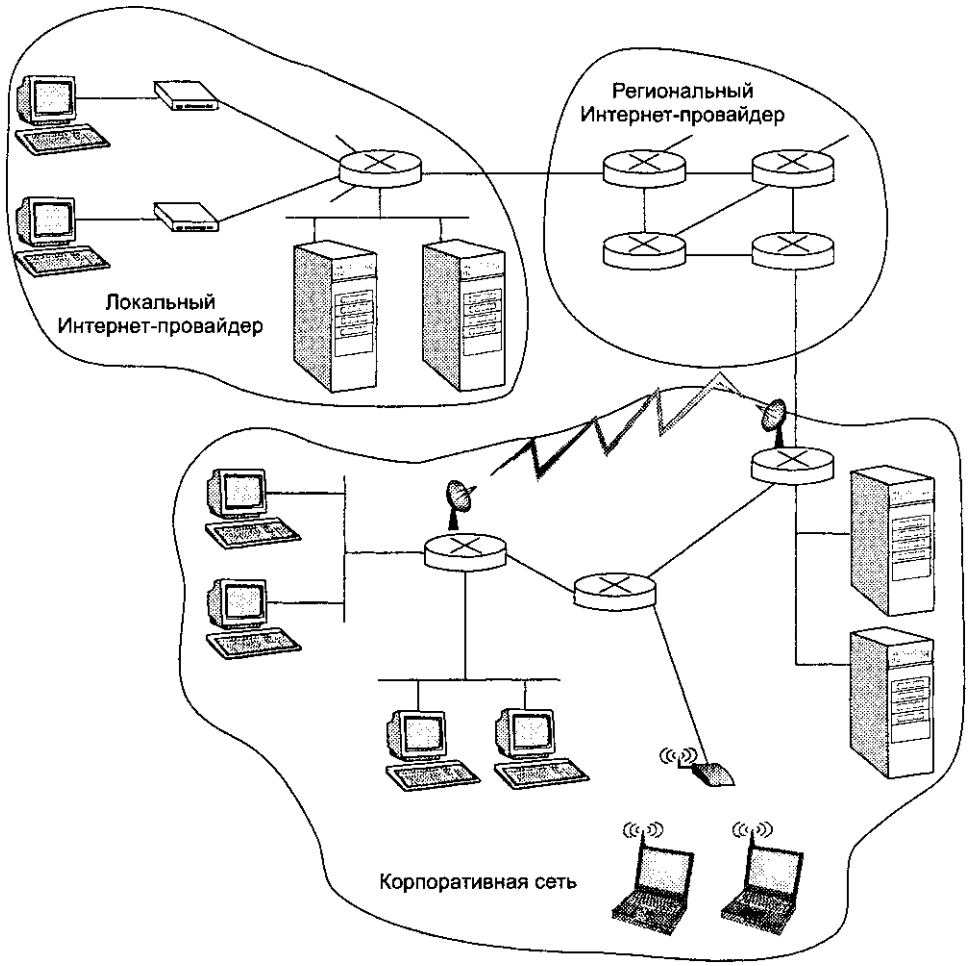


Рис. 1.1. Некоторые составные части Интернета

Оконечные системы далеко не всегда напрямую соединены между собой единственной физической линией связи. Напротив, типичной является ситуация, когда связь осуществляется с помощью множества последовательных линий, соединяемых специальными коммутирующими устройствами — *маршрутизаторами.* Маршру-

тизатор принимает порцию данных, передаваемую по одному из его входных каналов связи, а затем перенаправляет ее в один из своих выходных каналов связи. В терминологии компьютерных сетей передаваемые порции данных называют *пакетами.* Последовательность каналов связи и маршрутизаторов, через которые пакет проходит в процессе передачи, называется *маршрутом,* или *путем,* пакета в сети. Путь пакета заранее не известен и определяется непосредственно в процессе передачи. В Интернете каждой паре оконечных систем не предоставляется выделенный маршрут, а используется технология коммутации пакетов, при этом различные пары оконечных систем могут одновременно пользоваться одним и тем же маршрутом или частью маршрута. Первые сети с коммутацией пакетов, созданные в начале 70-х годов, являются «далекими предками» сегодняшнего Интернета.

Доступ оконечных систем к Интернету осуществляется при помощи *поставщиков услуг Интернета,* или *Интернет-провайдеров* (Internet Service Provider, ISP). Интернет-провайдеры подразделяются на резидентных (например, AOL или MSN), университетских (Университет Стенфорда) и корпоративных (компания Ford Motor). Интернет-провайдер предоставляет сеть маршрутизаторов и линий связи. Как правило, Интернет-провайдеры предлагают несколько способов подключения оконечных систем к Сети: коммутируемое модемное соединение на скорости 56 Кбит/с, резидентное широкополосное подключение при помощи кабельного модема или цифровой абонентской линии (Digital Subscriber Line, DSL), высокоскоростной доступ через локальную сеть (Local Area Network, LAN), а также беспроводной доступ. Кроме того, Интернет-провайдеры осуществляют прямое подключение к сети web-сайтов. Для того чтобы обеспечить связь между удаленными пользователями, а также предоставить пользователям доступ к информации, хранящейся в Интернете, местные Интернет-провайдеры подключаются к Интернет-провайдерам национального или интернационального звена, таким как UUNet и Sprint. Последние используют высокоскоростные маршрутизаторы, соединенные оптоволоконными кабелями. Каждый из Интернет-провайдеров как нижнего, так и верхнего звеньев является административной единицей, передающей данные по протоколу IP (см. далее) и придерживающейся соглашений об именах и адресах, принятых в Интернете. Более подробно вопросы, связанные с Интернет-провайдерами и их взаимодействием, будут рассмотрены в разделе «Интернет-провайдеры и магистрали Интернета» этой главы.

Оконечные системы, маршрутизаторы и другие «компоненты» Интернета используют *протоколы,* осуществляющие управление приемом и передачей информации внутри Интернета. Наиболее важными протоколами в глобальной Сети являются TCP (Transmission Control Protocol — протокол управления передачей) и IP (Internet Protocol — Интернет-протокол). Протокол IP определяет формат пакетов, передающихся между оконечными системами и маршрутизаторами. Стек основных протоколов, использующихся в Интернете, известен под названием TCP/IP. В этой главе мы начнем с сути понятия «протокол». Но помните, что это лишь начало — сетевым протоколам посвящена большая часть книги.

То, что мы обычно называем словом «Интернет», — это так называемый *открытый Интернет.* Кроме общедоступного Интернета существует также множество

закрытых (частных) компьютерных сетей, построенных по тому же принципу, что и глобальная Сеть. Как правило, частные сети предназначены для использования внутри различных фирм и организаций; они не могут обмениваться сообщениями с внешней средой, за исключением сообщений, проходящих через так называемые брандмауэры, контролирующие поток сообщений, входящих и выходящих из сети. Подобные сети объединяют под термином *интранет*. Это название созвучно имени «Интернет» и отражает тот факт, что в закрытых сетях используют такие же хосты, маршрутизаторы, каналы связи и протоколы, что и в открытом Интернете.

С точки зрения технологий и развития существование Интернета обеспечивается созданием, проверкой и внедрением *Интернет-стандартов*. Эти стандарты вырабатываются *проблемной группой разработок для Интернета* (Internet Engineering Task Force, IETF). Документы, создаваемые IETF [237], носят название RFC (Requests For Comments — предложения для обсуждения). Изначально подобные документы предназначались для разрешения архитектурных проблем, возникавших в сетях-предшественницах Интернета. Со временем ситуация сложилась так, что, формально не обладая статусом стандарта, документы RFC стали стандартами де-факто. В настоящее время эти документы составляются весьма точно и детально, описывая такие протоколы, как TCP, IP, HTTP (для web) и SMTP (для электронной почты). Существует более 3000 различных документов RFC.

Интернет с точки зрения обслуживания

Предыдущий подраздел был посвящен составным частям Интернета. Теперь мы перейдем к описанию Интернета с точки зрения обслуживания.

- Интернет позволяет *распределенным приложениям,* работающим на оконечных системах, осуществлять обмен данными друг с другом. В число таких приложений входят удаленный терминал, электронная почта, средства навигации в web, средства передачи аудио- и видеоданных, Интернет-телефония, сетевые компьютерные игры, средства одноранговой (Peer-to-Peer, P2P) обмена файлами и т. д. Следует подчеркнуть, что web — это не отдельная компьютерная сеть, а одно из множества распределенных приложений, использующих предоставляемые Интернетом службы связи.
- Интернет предоставляет своим распределенным приложениям два типа служб: надежную службу с установлением логического соединения и ненадежную службу без установления логического соединения. «В первом приближении» эти понятия означают следующее. Надежная служба с установлением логического соединения гарантирует, что передаваемые отправителем данные будут доставлены получателю полностью (то есть без потерь и искажений) и в исходном порядке. Ненадежная служба без установления логического соединения, напротив, не предоставляет никаких гарантий относительно доставки. Как правило, распределенное приложение способно поддерживать один из двух типов передачи.
- В настоящее время Интернет не дает гарантий относительно того, сколько времени понадобится для передачи данных от отправителя к адресату. И, если не

считать возможности повышения пропускной способности канала доступа к вашему Интернет-провайдеру, на сегодняшний день вы не можете потребовать в Интернете более высокого качества обслуживания (например, ограничения на длительность задержки), даже если вы готовы доплатить за это. Такую ситуацию многие (особенно американцы) находят странной. Мы подробнее рассмотрим эту проблему в главе 6.

Второе определение Интернета в терминах служб, или услуг, предоставляемых ими распределенным приложениям, является весьма важным для понимания того, что же такое глобальная Сеть. Возвращаясь к первому из двух определений, нельзя не отметить, что постоянно растущие пользовательские потребности обусловливают стремительный рост числа Интернет-приложений и их бурное развитие, что, в свою очередь, стимулирует развитие структурных компонентов сети. Поэтому необходимо помнить, что Интернет представляет собой динамически изменяющуюся *инфраструктуру,* в которой двигателем развития служат пользовательские приложения.

Итак, вы только что познакомились с двумя описаниями Интернета. Однако возможно, что эти описания не только не дали вам ответа на вопрос, что такое Интернет, но и породили множество новых вопросов. Что такое коммутация пакетов, стек протоколов TCP/IP и служба на основе соединений? Что такое маршрутизаторы? Какие виды каналов связи используются в Интернете? Что представляет собой распределенное приложение? Если сейчас мы не прояснили этого для вас, не стоит отчаиваться — чуть позже вы обязательно получите наглядное представление о каждом из перечисленных понятий.

Что такое протокол?

Теперь, когда мы имеем общее представление о том, что такое Интернет, давайте обратимся к другому центральному понятию сетевых технологий — понятию *протокола.* Что такое протокол? Для чего он нужен? Ответы на эти вопросы вы найдете ниже.

Аналогия из мира людей

Для того чтобы понять, что означает слово «протокол» в контексте компьютерных сетей, давайте рассмотрим ситуацию, далекую от вычислительной техники. Каждый человек, находясь во взаимодействии с другими людьми, всегда следует некоторым стереотипам общения. Например, ситуацию, когда один человек обращается к другому для того, чтобы узнать, который час, графически можно представить так, как это сделано на рис. 1.2. «Человеческий протокол» (обычно называемый правилами хорошего тона) гласит, что для установления контакта человеку необходимо поздороваться с собеседником и получить от него ответное приветствие. Этому начальному фрагменту общения соответствуют первые две стрелки с надписями «Привет!» на рисунке. В случае, если потенциальный собеседник не настроен на общение, он, вероятно, выдаст другой ответ, например: «Не беспокойте меня» или «Я не говорю по-русски». Тогда инициатору общения следует прекратить попытки контакта с собеседником. Возможна также ситуация, когда собесед-

ник не даст никакого ответа; вероятно, разумным решением здесь было бы повторить попытку установить контакт по прошествии некоторого времени. Таким образом, в основе «человеческого протокола» лежит следующий принцип: люди посылают определенные сообщения и предпринимают определенные действия в качестве реакции на эти сообщения и другие события (например, отсутствие ответного сообщения в течение установленного промежутка времени). Становится вполне очевидным, что протокол определяется набором входящих в него сообщений и ответных действий. Если два человека используют различные протоколы (например, у одного из них неважно с общепринятыми нормами поведения или нет ощущения времени), общение между ними становится невозможным. То же самое абсолютно справедливо и в отношении сетевых протоколов — для выполнения сетью своих функций необходимо, чтобы два (или более) устройства, обменивающихся данными, использовали один и тот же протокол.

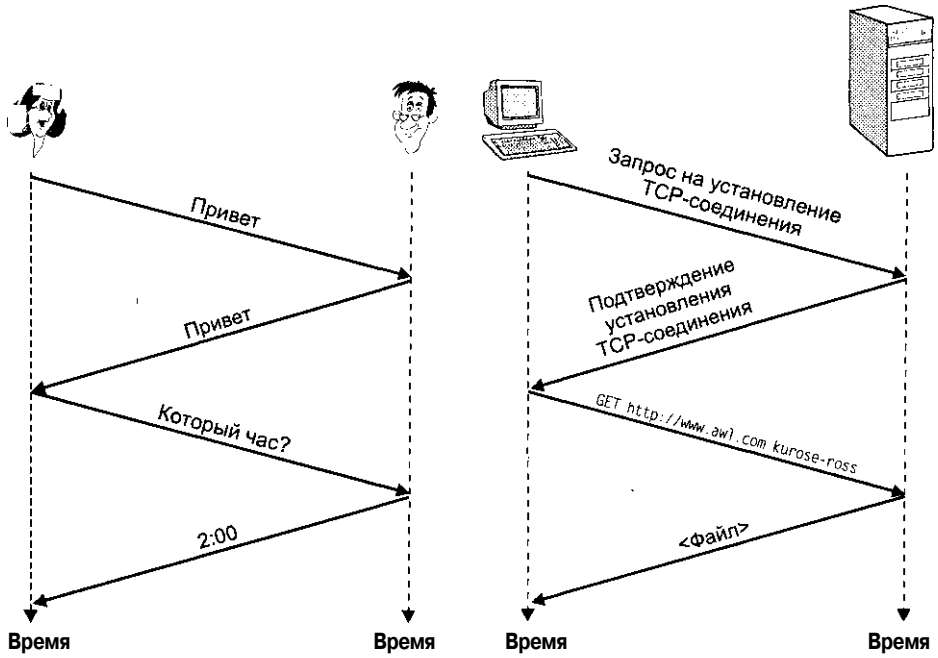


Рис. 1.2. Протоколы общения между людьми и между компьютерами

Давайте рассмотрим еще один пример аналогии «человеческих» и сетевых протоколов. Предположим, что вы находитесь на лекции в учебном заведении (пусть это будет лекция по компьютерным сетям!). Преподаватель не вполне понятно для вас излагает материал о протоколах, а затем останавливается и обращается к аудитории: «Не желает ли кто-нибудь задать вопрос?» Последняя фраза является сообщением, которое передается преподавателем и принимается всеми (кто еще не спит) студентами. Вы поднимаете руку, тем самым посылая сообщение преподавателю (хотя и в менее явной форме). Преподаватель подтверждает получение

вашего сообщения, говоря: «Да?» (передавая сообщение, поощряющее вашу любознательность — преподаватели *любят,* когда им задают вопросы). Ваше обращение к преподавателю с вопросом и его ответ также являются сообщениями, передаваемыми в процессе общения. Мы снова видим, что в основе описанного протокола вопросов и ответов лежат передача и прием сообщений, а также набор определенных договором или соглашением действий, предпринимаемых при передаче и приеме этих сообщений.

Сетевые протоколы

Основное отличие сетевого протокола от описанного выше «человеческого протокола» заключается в том, что обмен сообщениями производится не людьми, а аппаратными или программными средствами технического или программного обеспечения некоторого устройства (например, компьютерами, маршрутизаторами и т. п.). Любое движение информации в Интернете между двумя или более устройствами подчинено протоколу. Так, протоколы маршрутизаторов определяют путь пакета от отправителя к получателю; реализованные аппаратно протоколы сетевых интерфейсных карт двух физически соединенных компьютеров контролируют поток битов, передаваемых по сетевому кабелю; протоколы контроля перегрузки, используемые в оконечных системах, предназначены для контроля частоты передачи пакетов; и т. д. Интернет полностью основан на протоколах, и поэтому большая часть материала этой книги посвящена этому важнейшему в области компьютерных сетей понятию.

В качестве примера, одновременно простого и наглядно иллюстрирующего суть сетевого протокола, рассмотрим, что происходит в момент, когда вы производите запрос к web-серверу (вводите универсальный адрес ресурса в адресную строку вашего web-браузера). Графически данную ситуацию иллюстрирует правая половина рис. 1.2. Сначала ваш компьютер посылает серверу сообщение с запросом на установление соединения и ждет ответа. Сервер принимает запрос и отсылает ответное сообщение, подтверждающее установление соединения. Таким образом, зная, что теперь можно запросить web-документ, компьютер отсылает серверу имя ресурса, а сервер, в свою очередь, возвращает требуемый ресурс (web-страницу или файл) пользователю.

Рассмотрев различные примеры протоколов, демонстрирующие их две наиболее значимые составляющие — сообщения и действия, предпринимаемые при передаче и приеме сообщений, мы можем сформулировать следующее определение протокола.

> *Протокол определяет формат и очередность сообщений, которыми обмениваются два или более устройства, а также действия, выполняемые при передаче и/или приеме сообщений либо при наступлении иных событий.*

Протоколы очень широко используются как в компьютерных сетях вообще, так и в сети Интернет в частности. Для решения разных задач, связанных с передачей данных, требуются разные протоколы. Как вы убедитесь при изучении данной книги, существует масса самых разных типов протоколов, причем одни являются «пря-

молинейными» и легко поддаются пониманию, другие же, напротив, изобилуют нестандартными техническими решениями и требуют внимательного и кропотливого отношения. Без преувеличения можно сказать, что изучение компьютерных сетей основывается на понимании того, что, как и для чего выполняется протоколами.

Несколько полезных ссылок

Каждый опытный пользователь Интернета знает, что наиболее свежая и точная информация о глобальной Сети и ее протоколах содержится не в учебниках и журналах, посвященных вычислительной технике, а в самой Сети. Разумеется, Интернет является хранилищем огромного количества информации, и далеко не всегда она достаточно качественна и достоверна. Для того чтобы помочь вам «отделить зерна от плевел», мы приводим несколько ссылок на Интернет-ресурсы, посвященные сетевым технологиям, в частности, касающимся Интернета. Мы считаем, что эти ресурсы содержат весьма высококачественный и интересный материал. Иногда мы будем указывать ссылки, с помощью которых вы сможете найти в Интернете дополнительный материал к рассматриваемой теме.

- Сайт проблемной группы разработок для Интернета (IETF), http://www.ietf.org. IETF представляет собой открытое сообщество, целью которого является развитие Интернета и его архитектуры. Официально IETF возник в 1986 году по инициативе совета по архитектуре Интернета (Internet Architecture Board, IAB), http://www.iab.org. Собрания IETF проводятся три раза в год; большая часть текущей работы организации идет путем почтового обмена между рабочими группами. Управление IETF осуществляется Интернет-сообществом (Internet society), на сайте которого (http://www.isoc.org) вы можете найти массу высококачественного материала о глобальной Сети.
- Сайт web-консорциума (World Wide Web Consortium, W3C), http://www.w3.org. Web-консорциум был основан в 1994 году для разработки единых сетевых протоколов Интернета. На сайте находится великолепная подборка материалов о web-Технологиях, протоколах и стандартах.
- Сайты ассоциации компьютерной техники (Association for Computing Machinery, ACM), http://www.acm.org, и института инженеров по электротехнике и радиоэлектронике (Institute of Electrical and Electronics Engineers, IEEE), http://www.ieee.org. Эти профессиональные международные организации проводят технические конференции и издают собственные журналы, посвященные сетевым компьютерным технологиям. Наибольший интерес с точки зрения излагаемого в книге материала представляют сайты следующих подразделений внутри ACM и IEEE:
  - специальной группы по обмену данными (Special Interest Group in Data Communications, SIGCOMM) ACM, http://www.acm.org/sigcomm;
  - отдела по связи (Communications Society) группы IEEE, http://www.comsog.org;
  - компьютерного отдела (Computer Society) группы IEEE, http://www.computer.org.

и сервер взаимодействуют, передавая друг другу сообщения через Интернет. На данном уровне абстракции маршрутизаторы, каналы связи и прочие «детали», из которых состоит Интернет, являются для нас «черными ящиками», обеспечивающими передачу сообщений между клиентом и сервером. Описанную модель иллюстрирует рис. 1.3.

Иногда при взаимодействии двух частей приложения стороны «меняются ролями». Например, в популярных одноранговых приложениях для обмена файлами клиентом считается пользователь, принимающий файл, а сервером — пользователь, передающий файл, при этом направление передачи файлов в течение сеанса связи может изменяться.

ИСТОРИЧЕСКАЯ СПРАВКА

Одним из интереснейших Интернет-проектов, без сомнения, является проект SETI@home. Это научный эксперимент по поиску внеземных цивилизаций (SETI) с использованием компьютеров, подключенных к Интернету. Любой пользователь глобальной Сети может принять участие в этом проекте, бесплатно загрузив программу-клиент, предназначенную для получения и обработки данных, фиксируемых радиотелескопом. Целью проекта является обнаружение сигналов, которые могли бы исходить от внеземных цивилизаций.

В горах на севере Пуэрто-Рико расположен самый большой в мире радиотелескоп Аречибо. Его показания непрерывно фиксируются на магнитной ленте, а накопленные материалы еженедельно отсылаются в университет Беркли. Там происходит оцифровка данных с их разделением на рабочие единицы размером приблизительно 300 Кбайт. Эти единицы хранятся на сервере проекта. Для того чтобы принять участие в проекте, пользователю сначала необходимо загрузить с сайта программу-клиент и запустить ее на своем персональном компьютере. Программа установит TCP-соединение с сервером проекта, загрузит рабочую единицу и автоматически завершит соединение. Затем полученная единица будет обработана методом быстрого преобразования Фурье. В зависимости от вычислительной мощности компьютера это может занять от часа до нескольких суток. Когда программа завершит вычисления, она вновь установит соединение с сервером, чтобы передать серверу полученные результаты и загрузить новую рабочую единицу.

На сегодняшний день более 3 миллионов пользователей из 200 стран мира стали участниками этого грандиозного проекта. В среднем пользователями осуществляется более 200 триллионов операций с вещественными числами в секунду, что превышает мощность любого из существующих суперкомпьютеров. Более того, можно смело утверждать, что в проекте SETI@home задействуется лишь небольшая часть потенциала распределенных вычислений в Интернете. Если хотя бы 10 % от общего числа хостов, которое стремительно приближается к миллиардной отметке, приняли участие в распределенных вычислениях, это позволило бы осуществить не менее сотни проектов, подобных SETI@home [12].

Службы с установлением и без установления соединения

Оконечные системы используют службы Интернета для того, чтобы обмениваться данными, или, говоря точнее, сообщениями. Каналы и линии связи, маршрута-

заторы и другие элементы, составляющие структуру Интернета, определяют технологию передачи сообщений между оконечными системами. Далее мы рассмотрим виды таких технологий и их особенности.

Протокол TCP/IP предоставляет два вида служб оконечным системам: службу с установлением логического соединения и службу без установления логического соединения. При создании любого Интернет-приложения (программы обработки электронной почты, передачи файлов, web-браузера или приложения Интернет-телефонии) разработчику необходимо выбрать одну из двух указанных служб. Мы рассмотрим лишь краткие характеристики служб, а более подробное обсуждение последует в главе 3.

Служба с установлением логического соединения

Отличительной особенностью службы с установлением логического соединения является то, что клиент и сервер перед передачей данных (например, сообщений электронной почты) сначала обмениваются специальными управляющими пакетами. Эта процедура, иногда называемая *рукопожатием,* позволяет сторонам подготовиться к процессу основного обмена. Говорят, что по окончании процедуры рукопожатия *соединение* между оконечными системами является *установленным*.

Интересно отметить, что процедура рукопожатия между хостами очень напоминает протокол взаимодействия (общения) между людьми, в этом случае обмен приветствиями на рис. 1.2 представляет собой аналог человеческой процедуры рукопожатия (хотя реального рукопожатия между двумя людьми может не быть). В случае web-взаимодействия, также представленного на рис. 1.2, процедуру рукопожатия реализуют первые два сообщения, а следующая пара сообщений — сообщение GET и ответное сообщение с файлом — содержат реальные данные и пересылаются только после успешного установления соединения.

Не удивительно, если у вас сразу же возникнет вопрос о том, что означает термин «логическое соединение» и чем логическое соединение отличается от «обычного» соединения. Термин «логическое» отражает два аспекта. Во-первых, об установленном соединении известно только оконечным системам; коммутаторы (то есть маршрутизаторы) функционируют, «не зная», какие оконечные системы они обслуживают. Во-вторых, соединение представляет собой не что иное, как совокупность буферов обмена, выделенных в памяти оконечных систем, а также переменных состояния. Ни буферы, ни переменные также не содержат никакой информации о том, каким образом будет осуществляться передача пакетов.

С логическим соединением связаны несколько важных задач: надежной передачи данных, контроля потока данных и контроля перегрузки. Под *надежной передачей данных* понимается передача, в ходе которой не допускаются потери или искажения данных. Надежная передача в Интернете обеспечивается при помощи механизмов подтверждений и повторных посылок. Для того чтобы представить процесс надежной передачи данных, рассмотрим следующую ситуацию. Пусть установлено соединение между оконечными системами A и B. Каждый раз при завершении приема очередного пакета данных от системы A система B посылает подтверждение о том, что пакет был успешно принят. В случае, если система A не получает

и сервер взаимодействуют, передавая друг другу сообщения через Интернет. На данном уровне абстракции маршрутизаторы, каналы связи и прочие «детали», из которых состоит Интернет, являются для нас «черными ящиками», обеспечивающими передачу сообщений между клиентом и сервером. Описанную модель иллюстрирует рис. 1.3.

Иногда при взаимодействии двух частей приложения стороны «меняются ролями». Например, в популярных одноранговых приложениях для обмена файлами клиентом считается пользователь, принимающий файл, а сервером — пользователь, передающий файл, при этом направление передачи файлов в течение сеанса связи может изменяться.

ИСТОРИЧЕСКАЯ СПРАВКА

Одним из интереснейших Интернет-проектов, без сомнения, является проект SETI@home. Это научный эксперимент по поиску внеземных цивилизаций (SETI) с использованием компьютеров, подключенных к Интернету. Любой пользователь глобальной Сети может принять участие в этом проекте, бесплатно загрузив программу-клиент, предназначенную для получения и обработки данных, фиксируемых радиотелескопом. Целью проекта является обнаружение сигналов, которые могли бы исходить от внеземных цивилизаций.

В горах на севере Пуэрто-Рико расположен самый большой в мире радиотелескоп Аречибо. Его показания непрерывно фиксируются на магнитной ленте, а накопленные материалы еженедельно отсылаются в университет Беркли. Там происходит оцифровка данных с их разделением на рабочие единицы размером приблизительно 300 Кбайт. Эти единицы хранятся на сервере проекта. Для того чтобы принять участие в проекте, пользователю сначала необходимо загрузить с сайта программу-клиент и запустить ее на своем персональном компьютере. Программа установит TCP-соединение с сервером проекта, загрузит рабочую единицу и автоматически завершит соединение. Затем полученная единица будет обработана методом быстрого преобразования Фурье. В зависимости от вычислительной мощности компьютера это может занять от часа до нескольких суток. Когда программа завершит вычисления, она вновь установит соединение с сервером, чтобы передать серверу полученные результаты и загрузить новую рабочую единицу.

На сегодняшний день более 3 миллионов пользователей из 200 стран мира стали участниками этого грандиозного проекта. В среднем пользователями осуществляется более 200 триллионов операций с вещественными числами в секунду, что превышает мощность любого из существующих суперкомпьютеров. Более того, можно смело утверждать, что в проекте SETI@home задействуется лишь небольшая часть потенциала распределенных вычислений в Интернете. Если хотя бы 10 % от общего числа хостов, которое стремительно приближается к миллиардной отметке, приняли участие в распределенных вычислениях, это позволило бы осуществить не менее сотни проектов, подобных SETI@home [12].

Службы с установлением и без установления соединения

Оконечные системы используют службы Интернета для того, чтобы обмениваться данными, или, говоря точнее, сообщениями. Каналы и линии связи, маршрута-

заторы и другие элементы, составляющие структуру Интернета, определяют технологию передачи сообщений между оконечными системами. Далее мы рассмотрим виды таких технологий и их особенности.

Протокол TCP/IP предоставляет два вида служб оконечным системам: службу с установлением логического соединения и службу без установления логического соединения. При создании любого Интернет-приложения (программы обработки электронной почты, передачи файлов, web-браузера или приложения Интернет-телефонии) разработчику необходимо выбрать одну из двух указанных служб. Мы рассмотрим лишь краткие характеристики служб, а более подробное обсуждение последует в главе 3.

Служба с установлением логического соединения

Отличительной особенностью службы с установлением логического соединения является то, что клиент и сервер перед передачей данных (например, сообщений электронной почты) сначала обмениваются специальными управляющими пакетами. Эта процедура, иногда называемая *рукопожатием,* позволяет сторонам подготовиться к процессу основного обмена. Говорят, что по окончании процедуры рукопожатия *соединение* между оконечными системами является *установленным.*

Интересно отметить, что процедура рукопожатия между хостами очень напоминает протокол взаимодействия (общения) между людьми, в этом случае обмен приветствиями на рис. 1.2 представляет собой аналог человеческой процедуры рукопожатия (хотя реального рукопожатия между двумя людьми может не быть). В случае web-взаимодействия, также представленного на рис. 1.2, процедуру рукопожатия реализуют первые два сообщения, а следующая пара сообщений — сообщение GET и ответное сообщение с файлом — содержат реальные данные и пересылаются только после успешного установления соединения.

Не удивительно, если у вас сразу же возникнет вопрос о том, что означает термин «логическое соединение» и чем логическое соединение отличается от «обычного» соединения. Термин «логическое» отражает два аспекта. Во-первых, об установленном соединении известно только оконечным системам; коммутаторы (то есть маршрутизаторы) функционируют, «не зная», какие оконечные системы они обслуживают. Во-вторых, соединение представляет собой не что иное, как совокупность буферов обмена, выделенных в памяти оконечных систем, а также переменных состояния. Ни буферы, ни переменные также не содержат никакой информации о том, каким образом будет осуществляться передача пакетов.

С логическим соединением связаны несколько важных задач: надежной передачи данных, контроля потока данных и контроля перегрузки. Под *надежной передачей данных* понимается передача, в ходе которой не допускаются потери или искажения данных. Надежная передача в Интернете обеспечивается при помощи механизмов подтверждений и повторных посылок. Для того чтобы представить процесс надежной передачи данных, рассмотрим следующую ситуацию. Пусть установлено соединение между оконечными системами A и B. Каждый раз при завершении приема очередного пакета данных от системы A система B посылает подтверждение о том, что пакет был успешно принят. В случае, если система A не получает

подтверждения, она инициирует повторную посылку пакета. *Контроль потока данных* требуется для того, чтобы ни одна сторона не превысила установленную частоту (или скорость) передачи пакетов. Это необходимо потому, что оконечные системы могут иметь разные скорости передач и, следовательно, отсутствие контроля может привести к ошибкам. В случае вероятности таких ошибок протокол вынуждает одну из сторон снизить скорость передачи пакетов. Как мы увидим в главе 3, контроль потока данных в Интернете осуществляется путем использования буферов на принимающей и передающей сторонах. Контроль перегрузки служит для предотвращения ситуаций взаимной блокировки передающих сторон. Когда маршрутизатор перегружен, возникает угроза переполнения его буферов и потери передаваемых пакетов. В Интернете эта проблема решается путем принудительного снижения частоты передачи пакетов в периоды перегрузки. Оконечные системы «узнают» о перегрузке по отсутствию подтверждений при передаче пакетов.

Логическое соединение не обязательно подразумевает надежную передачу данных, контроль потока данных и контроль перегрузки. Вполне допустимо существование компьютерных сетей, в которых наличие логического соединения не означает необходимость решать сразу все три задачи. В самом деле, службу с установлением логического соединения использует любая сеть, в которой перед передачей данных осуществляется процедура рукопожатия [243].

В Интернете протокол, использующий службу с установлением логического соединения, имеет название TCP (Transmission Control Protocol — протокол управления передачей). Первая версия TCP была определена в документе RFC 793. Как следует из сказанного ранее, протокол TCP решает три задачи: надежной передачи данных, контроля потока данных и контроля перегрузки. Обратим внимание на то, что приложения никак не связаны с механизмами функционирования TCP; другими словами, они «не знают», каким образом протокол решает поставленные перед ним задачи. Разумеется, для нас, в отличие от сетевых приложений, принципы работы TCP представляют значительный интерес, и мы займемся их детальным рассмотрением в главе 3.

Служба без установления логического соединения

Как вы, вероятно, уже догадались, служба без установления логического соединения не использует процедуру рукопожатия: вместо нее происходит простая передача пакетов. Это, с одной стороны, позволяет значительно сэкономить время при пересылке данных. С другой стороны, выигрыш во времени происходит за счет снижения надежности передачи: передающая сторона не имеет информации о том, была ли передача пакета успешной. Более того, контроль потока данных и перегрузки в службе без установления логического соединения также не производится, что обусловливает возможность потерь данных при передаче. Протокол Интернета, использующий описанную службу, называется UDP (User Datagram Protocol — протокол пользовательских дейтаграмм) и определен в документе RFC 768.

Большая часть популярных Интернет-приложений работает по протоколу TCP, то есть задействует службу с установлением логического соединения. К этим приложениям относятся Telnet (для удаленного доступа в сеть), SMTP (для работы

с электронной почтой), FTP (для передачи файлов) и HTTP (для навигации в web). Тем не менее протокол UDP также является весьма востребованным, в особенности в развивающейся сфере мультимедиа-приложений; например, с его помощью нередко организуются аудио- и видеоконференции.

Ядро компьютерных сетей

Изучив оконечные системы и способы передачи данных между ними, давайте обратимся к «ядру» компьютерной сети. Предметом нашего рассмотрения станет взаимодействие между маршрутизаторами, то есть механизмы передачи данных от одной оконечной системы к другой. Элементы структуры сети, относящиеся к ядру, на рис. 1.4 выделены.

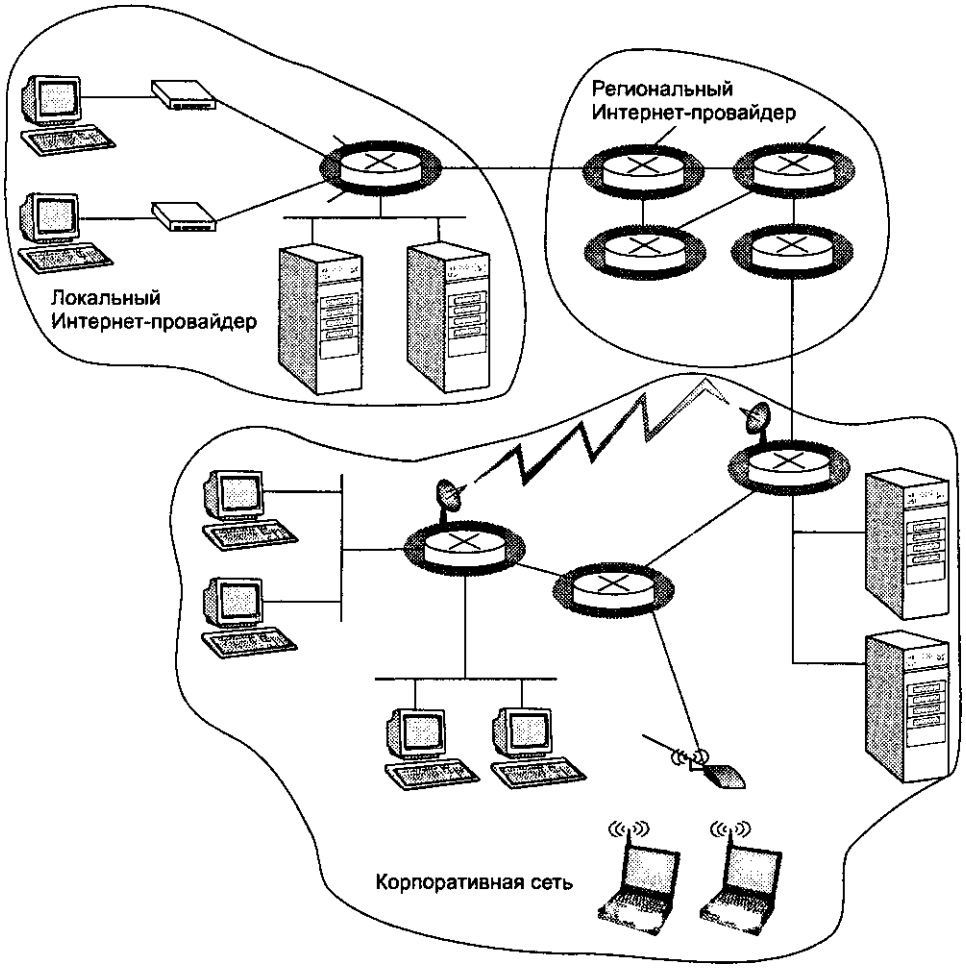


Рис. 1.4. Ядро компьютерной сети

Коммутация каналов и коммутация пакетов

Существует два фундаментальных подхода к организации ядра сети: *коммутация каналов* и *коммутация пакетов*. При коммутации каналов происходит резервирование на время сеанса связи необходимых ресурсов (буферов, диапазонов частот) на всем сетевом пути. При коммутации пакетов ресурсы запрашиваются при необходимости и выделяются по требованию. Иногда несколько сообщений могут пытаться использовать линию связи одновременно, поэтому возникает необходимость в организации очередей сообщений. Для наглядности рассмотрим следующую аналогию. Пусть имеются два ресторана, в одном из которых разрешается занимать места заранее, а в другом нет. Для посещения первого ресторана нам сначала необходимо сделать предварительный заказ мест по телефону и лишь затем отправиться туда лично. При этом мы избавлены от необходимости ждать свободного столика и можем немедленно требовать официанта. Для посещения второго ресторана мы не обязаны уведомлять его персонал заранее, однако в этом случае у нас могут возникнуть трудности с поиском свободного места.

Замечательным примером сетей с коммутацией каналов являются телефонные сети. Рассмотрим, что происходит, когда у одного абонента возникает необходимость передать информацию другому абоненту. Перед тем как начать разговор, нужно установить соединение между принимающей и передающей сторонами. В отличие от логического соединения, которое обсуждалось в предыдущем разделе, рассматриваемое соединение является «настоящим», то есть все каналы, лежащие на пути между абонентами, находятся в состоянии связи. На языке телефонии такое соединение называется *коммутацией*. При коммутации на все время соединения устанавливается постоянная частота передачи. Это возможно благодаря тому, что в телефонных сетях используется стандартная полоса частот.

Современный Интернет, напротив, является типичной сетью с коммутацией пакетов. Как правило, при передаче пакет проходит через множество каналов, однако никакого резервирования частотных полос при этом не происходит. В случае перегруженности какого-либо канала пакет будет вынужден ждать в очереди его освобождения. Таким образом, хотя с точки зрения быстродействия Интернет пытается доставлять пакеты *с максимальными усилиями,* время доставки не гарантировано.

Не каждую телекоммуникационную сеть можно однозначно отнести к сетям с коммутацией каналов или сетям с коммутацией пакетов. Так, например, сети, использующие режим асинхронной передачи (Asynchronous Transfer Mode, ATM), который будет рассмотрен в главе 5, организованы таким образом, что резервирование ресурсов в них сочетается с организацией очередей сообщений. Тем не менее разделение компьютерных сетей на сети с коммутацией каналов и коммутацией пакетов является удобной «отправной точкой» для изучения телекоммуникационных технологий.

Коммутация каналов

Главными «объектами внимания» в этой книге являются компьютерные сети, Интернет и коммутация пакетов. Тем не менее мы позволим себе небольшое отступление и кратко рассмотрим коммутируемые телефонные сети. Это поможет

вам лучше понять причины, по которым в Интернете используется коммутация пакетов, а не традиционная коммутация каналов.

На рис. 1.5 приведена типичная структура сети с коммутацией каналов. В этой сети четыре коммутатора соединены между собой линиями связи. Каждая из линий способна одновременно поддерживать *n* каналов связи. Хосты (персональные компьютеры, рабочие станции и т. п.) напрямую соединены с одним из коммутаторов. Между парами хостов устанавливается выделенное сквозное соединение (конечно, «конференционные» соединения, позволяющие общаться одновременно множеству абонентов, в подобных сетях также возможны, но мы оставим эти «экзотические» ситуации за пределами нашего рассмотрения, чтобы не усложнять общую картину). Таким образом, чтобы хост А имел возможность передавать пакеты хосту В, необходимо зарезервировать одну полосу частот на каждой из двух линий связи, соединяющих хосты А и В. Поскольку каждая линия связи способна поддерживать одновременно *n* каналов связи, ширина полосы канала связи составляет 1 /*n* часть от полосы пропускания линии связи.

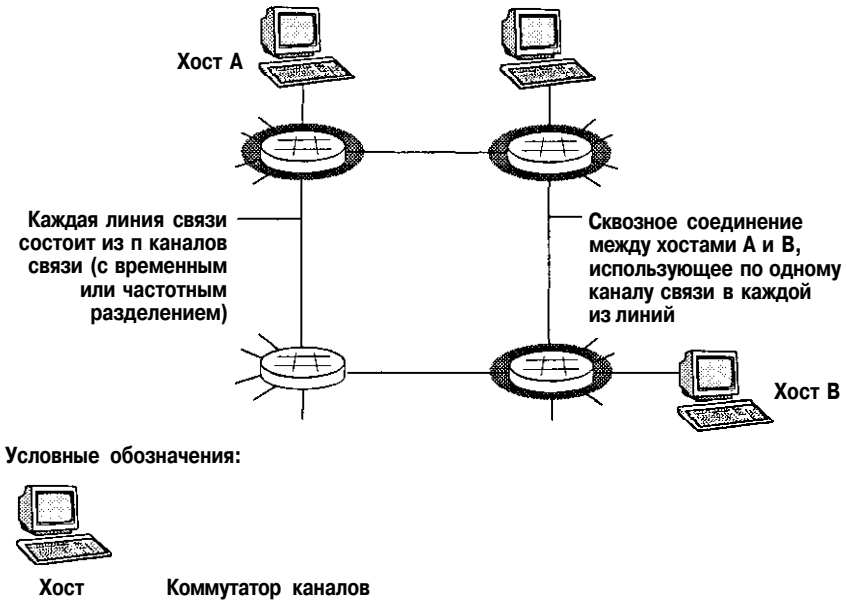


Рис. 1.5. Простая сеть с коммутацией каналов, состоящая из четырех коммутаторов и четырех линий связи

Мультиплексирование в сетях с коммутацией каналов

Каждый канал связи в линии связи организовывается при помощи *частотного* либо *временного разделения.* В первом случае каждому каналу связи отводится определенная полоса частот, которая не изменяется в течение всего сеанса связи. Например, для телефонных сетей типичной шириной полосы пропускания является 4 кГц. Радиостанции, работающие в FM-режиме, также используют принцип частотного разделения.

В настоящее время в телефонии наблюдается тенденция замены частотного разделения временным. Более того, большинство технологически развитых телефонных сетей уже использует принцип временного разделения каналов. Суть временного разделения заключается в следующем: время разбивается на равные промежутки, называемые кадрами, а каждый кадр делится на фиксированное число слотов. Выделение канала связи заключается в закреплении за парой абонентов одного временного слота в каждом кадре. Внутри этого слота происходит монопольная передача пакетов между абонентами по линии связи.

На рис. 1.6 показаны схемы функционирования линии связи, поддерживающей четыре канала, для случаев частотного и временного разделения. При частотном разделении полоса пропускания линии связи делится на 4 равные диапазона по 4 кГц. При временном разделении время делится на кадры, в каждом из которых имеются 4 слота; за каждым каналом связи закреплен один из этих слотов. Скорость передачи для временного разделения равна произведению частоты следования кадров и числа битов внутри каждого слота. Например, если частота следования кадров составляет 8000 кадров в секунду, а слот включает 8 бит, то скорость передачи по каналу связи будет составлять 64 Кбит/с.

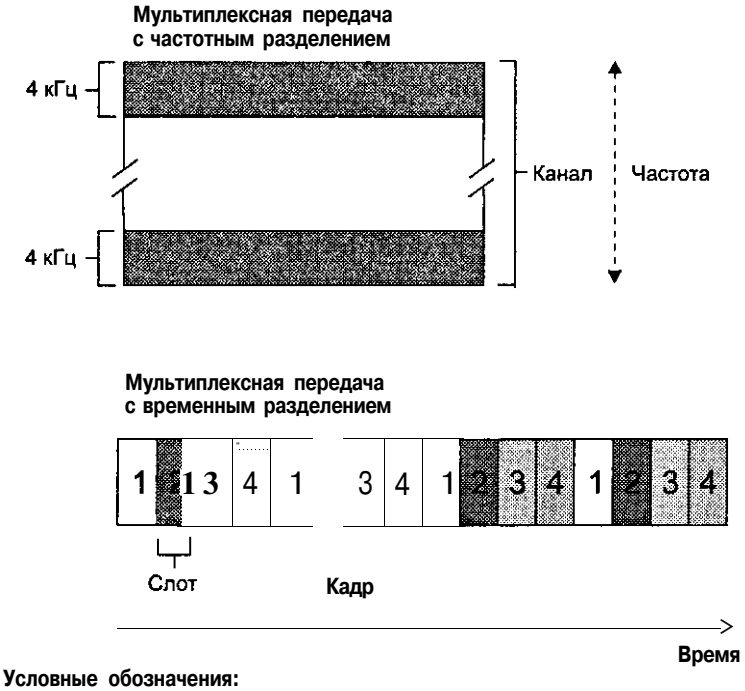


Рис. 1.6. При частотном разделении канал связи непрерывно использует свою частотную полосу, а при временном разделении — всю полосу пропускания линии связи в отведенные ему кванты времени (то есть в отведенные ему слоты)

Сторонники технологии коммутации пакетов всегда обращали внимание на серьезный недостаток сетей с коммутацией каналов, заключающийся в том, что выделенные каналы связи нельзя освободить в периоды *простоя*. Например, если во время телефонного разговора собеседники молчат (не передают информацию), то выделенный для них канал нельзя «отобрать» и использовать для других соединений. Представьте себе радиолога, которому необходим удаленный доступ к рентгеновским снимкам. Если этот доступ осуществляется при помощи коммутации каналов, то радиолог сначала устанавливает соединение, получает снимок, просматривает его, а затем запрашивает новый снимок. Все периоды времени, когда он занимается изучением снимков, являются простоями с точки зрения передачи информации по каналу связи.

Другой причиной, по которой коммутация каналов вызывает вполне обоснованную критику, является необходимость в сложном сигнальном оборудовании для управления коммутациями и выделения частотных полос каналам связи.

Перед тем как завершить разговор о сетях с коммутацией каналов, давайте рассмотрим численный пример, наглядно иллюстрирующий суть этой технологии. Пусть нам необходимо передать файл размером 640 000 бит от хоста A хосту B методом коммутации каналов. Сколько времени займет передача? Предположим, что все линии связи в сети одинаковы, используют принцип временного разделения, при этом число слотов кадра равно 24, а скорость передачи по линии составляет 1,536 Мбит/с. Предположим также, что временные затраты на установление соединения хоста A с сетью равны 0,5 с. Итак, поскольку скорость линии связи поровну «распределяется» между всеми каналами связи, скорость передачи по каналу связи составляет (1,536 Мбит/с) / 24 = 64 Кбит/с. Теперь мы можем рассчитать время передачи файла по каналу связи: 640 000 бит / (64 Кбит/с) = 10 с. Учитывая затраты хоста A на установление соединения, получаем полное время передачи файла: 10 с + 0,5 с = 10,5 с. В первом приближении время передачи не зависит от числа линий связи: его можно считать равным 10 с и в случае одной линии, и в случае 100 линий. Далее в этой главе мы рассмотрим понятие задержки распространения и избавимся от указанного предположения.

Коммутация пакетов

Как было сказано в разделе «Что такое Интернет?», для решения поставленных задач приложения обмениваются друг с другом сообщениями. Содержание и функции сообщений определяются разработчиком протокола. Так, сообщения могут выполнять контролирующую функцию («Привет!» при общении между людьми), содержать текстовую информацию (электронное письмо) или файл с изображением, звуком и т. п. В современных компьютерных сетях происходит автоматическое разбиение больших по объему сообщений на более мелкие фрагменты, называемые *пакетами*. Пакет является единицей передачи данных. При передаче пакет проходит через последовательность линий связи и коммутаторов, обычно называемых *маршрутизаторами*. Передача пакета по линии связи осуществляется монопольно, то есть с максимальной скоростью, которую способна обеспечить линия связи. Большинство маршрутизаторов используют механизм *передачи с промежуточным накоплением*. Это означает, что перед тем, как начать передачу в выход-

ную линию связи, маршрутизатору необходимо завершить процесс приема пакета в буфер. Таким образом, в маршрутизаторах возникает задержка накопления, обусловленная необходимостью ожидания окончания приема пакета. Время задержки накопления пропорционально длине пакета: если пакет длиной L бит необходимо передать в выходную линию связи, обладающую скоростью R бит/с, то время задержки накопления составит L/Rc.

Каждый маршрутизатор имеет множество входных и выходных линий связи. Каждая выходная линия связи имеет буфер, называемый *выходным буфером,* или *выходной очередью.* В выходном буфере хранятся все пакеты, предназначенные для передачи по линии связи. Буферы играют ключевую роль в механизме коммутации пакетов. Если при окончании приема пакета обнаруживается, что линия связи занята, то пакет ставится в очередь в выходном буфере. Таким образом, кроме задержки накопления в маршрутизаторах присутствует *задержка ожидания.* Задержки ожидания являются переменными величинами и зависят от загрузки линии связи. Поскольку размеры буферов ограничены, может возникнуть ситуация, когда свободного места в буфере окажется недостаточно для помещения нового пакета. В этом случае произойдет *потеря пакета* — будет утрачен либо новый пакет, либо один из пакетов, находящихся в очереди. Если вернуться к примеру с ресторанами, приведенному ранее, задержка ожидания — это время, которое вы вынуждены провести у стойки, пока не освободится место за столиком, а потеря пакета — просьба к вам со стороны официанта покинуть помещение, поскольку слишком много людей, как и вы, ожидают свободного места.

На рис. 1.7 приведена структура простой сети с коммутацией пакетов. Здесь и далее пакеты представлены в виде трехмерных брусков. Ширина бруска соответствует длине пакета. В данном примере все пакеты имеют одну и ту же длину. Предположим, что хосты А и В посылают пакеты хосту Е, при этом связь хостов А и В с первым маршрутизатором осуществляется с помощью линий связи Ethernet со скоростью 10 Мбит/с. Маршрутизатор направляет пакеты в линию связи со скоростью 1,5 Мбит/с. Если линия перегружена, пакеты ожидают ее освобождения в очереди.

Посмотрим, что происходит при одновременной передаче пакетов хостами А и В. Очевидно, что никакой синхронизации между хостами нет, и, следовательно, нельзя заранее предсказать порядок передачи пакетов. Эту особенность называют *статистическим мультиплексированием.* Статистическое мультиплексирование по сути противоположно временному разделению в технологии коммутации пакетов, когда за каждым каналом связи закреплен определенный слот в каждом временном кадре.

Теперь давайте подсчитаем время, необходимое для пересылки пакета длиной L бит между хостами при коммутации пакетов. Предположим, что число линий связи между хостами равно Q, при этом каждая линия связи обеспечивает скорость передачи R бит/с. Для простоты расчета примем, что задержки ожидания и распространения в сети отсутствуют и времени на установление соединения не требуется. Сначала происходит передача пакета по первой линии связи, время которой составляет L/R с; далее аналогичным образом пакет пересылается по Q - 1 линии связи, и общее время пересылки составляет QL/R с.

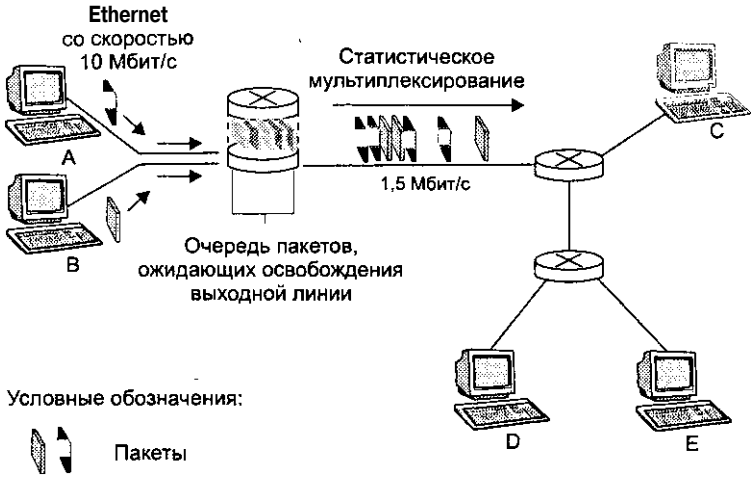


Рис. 1.7. Коммутация пакетов

Сравнение коммутации пакетов и коммутации каналов

Описав две основные технологии передачи пакетов, давайте сравним их между собой. Противники коммутации пакетов часто выдвигают тезис о том, что коммутация пакетов не позволяет организовать сетевое обслуживание в реальном времени (например, обеспечить передачу звука или видеоизображения), объясняя это непредсказуемыми задержками при передаче пакетов внутри сети. Сторонники коммутации пакетов замечают, что данная технология дает возможность более эффективно организовать разделение пропускной способности линии связи, а также является более простой, эффективной и менее дорогостоящей. Другими словами (вспоминая приведенную аналогию), люди, не любящие заказывать места в ресторанах заранее, предпочитают коммутацию пакетов.

Почему коммутация пакетов более эффективна? Рассмотрим простой пример. Предположим, что несколько пользователей совместно используют линию связи с пропускной способностью 1 Мбит/с. Каждый пользователь может находиться в состоянии активности, пересылая данные (к примеру, с постоянной скоростью 100 Кбит/с), либо в состоянии простоя, не занимая линию связи. Пусть пользователь находится в активном состоянии 10 % от общего времени. Если в сети применяется коммутация каналов, то для каждого пользователя должна быть зарезервирована скорость передачи 100 Кбит/с на все время сеанса связи, и сеть сможет одновременно обслуживать не более 10 пользователей. Теперь рассмотрим коммутацию пакетов. Если вероятность активности каждого пользователя составляет 10 % или 0,1, то вероятность наличия в сети одновременно 11 активных пользователей при условии, что общее число пользователей составляет 35, приблизительно равна 0,0004 (о способе подсчета этой величины вы узнаете в упражнении 10 — см. раздел «Резюме» в конце данной главы). Таким образом, вероятность того, что в сети находится не более 10 пользователей, составляет 0,9996. С другой стороны, последнее означает, что с вероятностью 0,9996 все пакеты будут проходить через сеть без задержек. При превышении «порога» в 10 активных пользователей в сети будут

наблюдаться перегрузки, которые приведут к появлению и росту очередей пакетов. Когда число активных пользователей упадет ниже 10, очереди станут уменьшаться. Из приведенных расчетов следует, что с вероятностью, близкой к 1, сеть с коммутацией пакетов не приведет к дополнительным ожиданиям, обслуживая при этом более чем втрое больше пользователей, чем сеть с коммутацией каналов.

На сегодняшний день коммутация пакетов и коммутация каналов являются двумя наиболее часто используемыми сетевыми технологиями, однако коммутация пакетов представляется значительно более перспективной. Об этом свидетельствует тенденция перехода к коммутации пакетов в телефонии. Более того, значительная часть дорогостоящих международных переговоров уже обеспечивается технологией коммутации пакетов.

Сегментирование сообщений

В большинстве современных сетей с коммутацией пакетов передающий хост разбивает длинные сообщения, генерируемые приложениями, на более мелкие пакеты. Эти пакеты доставляются адресату, из которых тот собирает исходные сообщения. Не удивительно, если вы уже задались вопросом: для чего нужно разбиение на пакеты? Не является ли эта работа бесполезной? Несмотря на то что подобный механизм усложняет процесс обмена как для передатчика, так и для приемника, еще «на заре» коммутации пакетов разработчики пришли к выводу о том, что преимущества разбиения на пакеты гораздо важнее недостатков. Перед тем как начать разговор об этом, необходимо ввести несколько новых терминов. Если в сети с коммутацией пакетов не производится сегментирование исходных сообщений (сообщения передаются целиком), то говорят, что сеть функционирует в режиме *коммутации сообщений.* Таким образом, коммутация сообщений является частным случаем коммутации пакетов.

На рис. 1.8 представлена схема коммутации сообщений для пути, состоящего из двух коммутаторов и трех линий связи. При коммутации сообщений каждое сообщение остается цельным на протяжении всего процесса передачи. Поскольку в коммутаторах обычно используется механизм передачи с промежуточным накоплением, возникает необходимость хранения целого сообщения для его дальнейшей передачи по сети. На рис. 1.8 брусок, представляющий сообщение, имеет больший размер, чем брусок, представляющий пакет на рис. 1.9, чтобы подчеркнуть больший по сравнению с пакетом размер сообщения.

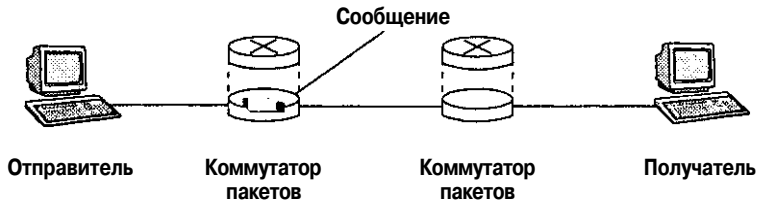


Рис. 1.8. Передача сообщения без сегментации

На рис. 1.9 показана схема передачи того же сообщения по тому же пути, однако в предположении, что сообщение разбивается на 4 пакета. Как можно видеть, рисунок соответствует моменту времени, когда первый пакет уже достиг адресата,

второй и третий пакеты находятся в процессе передачи, а последний, четвертый, пакет еще не отправлен. Механизм промежуточного накопления требует средств для временного хранения целого пакета до начала его дальнейшей передачи. Когда сообщение разбито на пакеты, говорят о *конвейерной передаче* сообщения, то есть параллельной (одновременной) передаче его фрагментов (пакетов) по различным линиям связи. Термин «конвейер» заимствован из области компьютерной архитектуры, однако вполне уместен в данной ситуации.

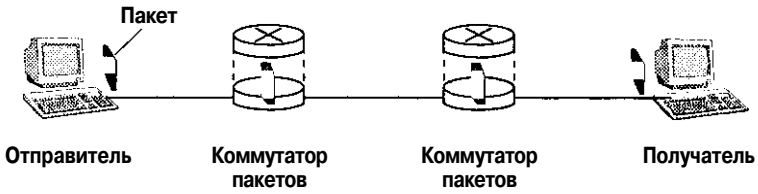
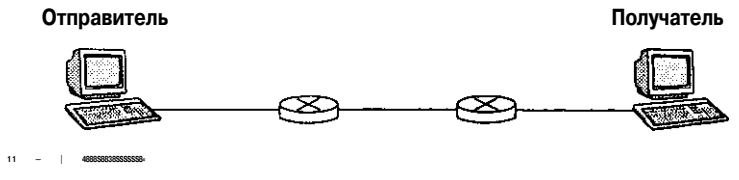


Рис. 1.9. Передача сообщения с сегментацией на пакеты

Важное преимущество разбиения на пакеты заключается в том, что время передачи сообщения, как правило, значительно сокращается по сравнению с передачей сообщения целиком. Рассмотрим это преимущество на следующем примере. Пусть необходимо передать сообщение длиной 7,5 x 10^e бит. Пусть между передатчиком и приемником находятся два коммутатора и три линии связи, каждая из которых обеспечивает скорость передачи 1,5 Мбит/с. Рассчитаем время передачи сообщения без предварительного сегментирования в предположении, что сеть не перегружена. Для передачи сообщения первому коммутатору требуется 7,5 x 10^6 бит / (1,5 Мбит/с) = 5 с. До тех пор пока все сообщение не окажется в коммутаторе, дальнейшая передача, как было сказано ранее, невозможна. Аналогично, для передачи сообщения между первым и вторым коммутаторами, а также между вторым коммутатором и приемником требуется 5 с. Таким образом, искомое время составляет 5с + 5с +5с = 15 с. Иллюстрация к задаче приведена на рис. 1.10.



Ю - ; *ШШ1*

Время, с

Условные обозначения:

ЩШШш **Сообщение**

Рис. 1.10. Расчет времени передачи сообщения без сегментации

Теперь предположим, что исходное сообщение было разбито на 5000 пакетов длиной 1500 бит. Теперь снова рассчитаем время передачи пакета, предположив отсутствие перегрузок в сети. Как показано на рис. 1.11, время передачи пакета по каждой из линий связи составляет 0,001 с. Однако сразу после того, как первый пакет будет отправлен во вторую линию связи, возникает возможность использовать освободившуюся первую линию для передачи второго пакета. Таким образом, второй пакет достигнет первого коммутатора за 0,002 с. Продолжая подсчет, мы придем к выводу, что последний пакет будет передан первому коммутатору за 5000 x 0,001 = 5 с. Таким образом, время передачи сообщения составит 5,002 с.

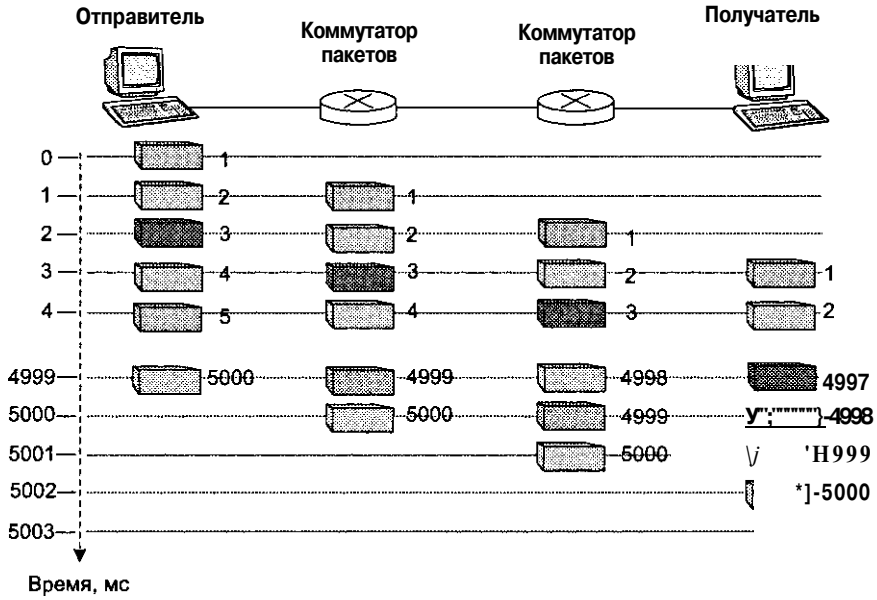


Рис. 1.11. Расчет времени передачи сообщения при его сегментации на 5000 пакетов

Как видно из примера, сегментирование позволило сократить время передачи сообщения в 3 раза! Однако почему удалось добиться такого результата? Чем механизм передачи с сегментированием отличается от механизма передачи целого сообщения? Главное отличие состоит в том, что передача сообщения целиком является по сути последовательной, а сегментированная передача — параллельной. При последовательной передаче активным является лишь один из трех узлов (передатчик или коммутатор), а при параллельной передаче все три узла активны на протяжении почти всего сеанса связи.

Сегментация сообщений обладает еще одним важным достоинством. Как мы убедимся позже, при передаче пакетов по сети иногда возникают искажения отдель-

ных битов. Как правило, при обнаружении ошибки коммутатор удаляет соответствующий пакет. В случае если передаче подлежит целое сообщение большого размера, любое искажение приведет к необходимости повторной посылки всего сообщения. Если же сообщение разбито на пакеты, то будет достаточно осуществить повторную передачу искаженного пакета.

Разумеется, сегментирование сообщений помимо достоинств имеет и недостатки. Позже мы увидим, что кроме полезных данных каждый пакет данных несет в себе массив контрольной информации. Контрольная информация, заключенная в *заголовке* сообщения, может содержать такие сведения, как адреса отправителя и получателя, а также идентификатор пакета (например, некоторое число). Поскольку при сегментировании возникает необходимость снабжения заголовком каждого пакета, объем контрольной информации, приходящейся на единицу полезных данных, по сравнению с коммутацией сообщений выше.

Перед тем как продолжать изучение нового материала, мы настоятельно рекомендуем вам обратиться к Java-апплету, посвященному сегментированию сообщений. Апплет доступен на сайте http://www.awl.com/kurose-ross. Он позволит вам самостоятельно поэкспериментировать с количественными моделями передачи сообщений. Заметим, что модели немного усложнены по сравнению с рассмотренным материалом, что позволяет учесть задержку распространения сигнала.

Передача сообщений

Существуют два основных класса компьютерных сетей с коммутацией пакетов: дейтаграммные сети и сети с виртуальным каналом. Эти два класса различаются между собой механизмами передачи пакетов внутри сети. Сети, в которых передача осуществляется на основе анализа адреса получателя, мы назовем *дейтаграммными*. Дейтаграммный способ передачи, например, характерен для Интернета. Если же в сети используется механизм передачи с виртуальным каналом, то говорят о *сети с виртуальными каналами*. К последним относятся сети, поддерживающие протокол X.25, ретрансляцию кадров, асинхронный режим передачи (ATM). Несмотря на то что разница между дейтаграммой и виртуальным каналом может показаться не столь значительной, выбор одного из двух способов передачи имеет чрезвычайно важные последствия, влияющие на механизм маршрутизации в сети. Мы убедимся в этом чуть позже.

Сети с виртуальными каналами

Виртуальный канал (Virtual Channel, VC) характеризуется тремя составляющими:

- маршрутом, по которому передаются все пакеты от отправителя к получателю;
- номерами виртуального канала, по одному номеру на каждую из линий связи, образующих маршрут;
- записями в таблицах трансляции номеров виртуального канала, имеющихся в каждом из коммутаторов на маршруте.

После того как соединение между получателем и отправителем установлено (создан виртуальный канал), отправитель может начинать пересылку пакетов с со-

ответствующими номерами виртуального канала. Поскольку у каждой линии связи имеется свой номер виртуального канала, каждый раз при прохождении пакета через коммутатор последний должен автоматически изменять для пакета значение номера виртуального канала. Новый номер виртуального канала пакет получает при помощи таблицы трансляции номеров виртуального канала.

Концепцию виртуального канала иллюстрирует рис. 1.12. Предположим, что хост A запросил виртуальный канал с хостом B, и сеть установила канал с маршрутом A-PS1-PS2-B, назначив линиям связи номера 12, 22 и 32 соответственно. Таким образом, каждый пакет, отправляющийся из хоста A, имеет номер 12, а пакеты, отправляющиеся из маршрутизаторов PS1 и PS2, — номера 22 и 32 соответственно. Номера, обозначенные рядом с линиями связи, подключенными к маршрутизатору PS1, называются *интерфейсными*.

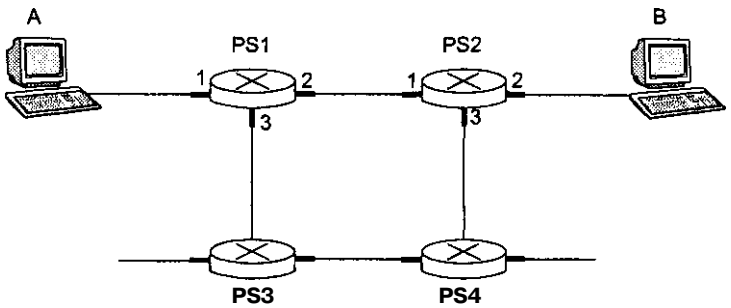


Рис. 1.12. Простая сеть с виртуальным каналом

Давайте рассмотрим, каким образом коммутатор осуществляет выбор номера для получаемых пакетов. Как было сказано, каждый коммутатор снабжен таблицей трансляции номеров виртуального канала; пример такой таблицы для коммутатора PS1 приведен ниже.

Входной интерфейс	Входной номер	Выходной интерфейс	Выходной номер
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87

При установлении виртуального канала в таблицу трансляции номеров виртуального канала коммутатора помещаются соответствующие записи, которые существуют только во время соединения и удаляются при его разрыве.

Возможно, у вас возник вопрос: зачем нужно изменять у пакета номер виртуального канала при каждой смене линии связи? Для этого имеются две причины. Первая заключается в том, что это позволяет сократить в пакете длину поля номера виртуального канала. Вторая, более важная причина кроется в упрощении механизма маршрутизации; говоря точнее, каждой линии связи ставится в соответствие

уникальный номер, не зависящий от номеров других линий. Если бы все линии связи на пути пакетов имели один и тот же номер виртуального канала, это привело бы к необходимости обработки коммутаторами значительного числа сообщений для создания общего номера виртуального канала.

Если в сети используется механизм виртуальных каналов, то коммутаторы такой сети должны располагать маршрутной информацией о каждом из текущих соединений. Другими словами, каждый раз при установлении виртуального канала в таблицу трансляции номеров виртуального канала всех коммутаторов, находящихся внутри канала, должна быть занесена необходимая информация. При разрыве соединения эта информация становится ненужной и должна автоматически удаляться. Тем не менее, независимо от того, есть ли в сети хотя бы один виртуальный канал, необходимо сохранять информацию о связи интерфейсных номеров с номерами виртуального канала на каждой линии связи. Вопрос о том, должна ли сеть хранить информацию о текущих соединениях, является критически важным и будет рассмотрен чуть позже.

Дейтаграммные сети

Дейтаграммные сети можно рассматривать как аналог обычных (не электронных) почтовых служб. Когда мы хотим отправить письмо, мы пишем на конверте почтовый адрес получателя и опускаем конверт в почтовый ящик. Почтовый адрес имеет иерархическую структуру и включает в себя, например, страну, город, улицу и номер дома. Почтовая служба обрабатывает каждое из полей в порядке иерархии, начиная с самого «общего» — страны адресата. В первую очередь, письмо передается в нужную страну, затем — в нужный город, а далее местные почтовые службы доставляют письмо непосредственно по месту назначения.

В дейтаграммной сети каждый передаваемый пакет содержит информацию об адресе получателя, который, как и почтовый адрес, имеет иерархическую структуру. Каждый раз при получении пакета коммутатор анализирует фрагмент адреса пакета и направляет пакет в соответствующую линию связи. Говоря точнее, коммутатор снабжен таблицей маршрутизации, связывающей конечные адреса или их фрагменты с линиями связи. После считывания заголовка происходит выделение адреса, который используется в качестве индекса таблицы маршрутизации. Дейтаграммную передачу можно сравнить с водителем, который ведет автомобиль, не ориентируясь по карте, а получая указания относительно дальнейшего направления движения от диспетчера.

Позже мы детально рассмотрим механизм передачи пакетов в дейтаграммных сетях. Обратите внимание на то, что дейтаграммные сети, в отличие от сетей с виртуальными каналами, не используют информацию о состоянии текущих соединений в своих коммутаторах. Фактически любая сеть, построенная на «чистой» дейтаграммной передаче, не контролирует информационные потоки внутри себя, поскольку решение о пути следования любого пакета принимается исключительно на основе адреса его назначения и не зависит от соединения между хостами. Простота дейтаграммнотр механизма дает повод для критических замечаний в адрес виртуальных каналов относительно сложности последних. Особенное усердие прилагают специалисты в области Интернет-технологий. Сторонники виртуальных

каналов парируют эти замечания тем, что их технология обеспечивает лучшее сетевое обслуживание приложений.

Не испытываете ли вы желания понаблюдать за процессом маршрутизации в Интернете? Если да, то приглашаем вас на сайт http://www.traceroute.org, который содержит весьма любопытную программу (команду) Traceroute (мы вернемся к этой программе в разделе «Задержки и потери данных в сетях с коммутацией пакетов»).

Систематика основных понятий

В предыдущих разделах мы представили несколько важных концепций, касающихся сетевых технологий, таких как коммутация каналов и пакетов, виртуальный канал, службы с установлением и без установления логического соединения. Возникает естественный вопрос: как связать воедино все эти понятия?

Сначала мы разделили все компьютерные сети на сети с коммутацией каналов и с коммутацией пакетов (рис. 1.13). Затем сети с коммутацией каналов, в свою очередь, были разделены на сети с частотным и временным мультиплексированием, а сети с коммутацией пакетов — на дейтаграммные сети и сети с виртуальными каналами. В сетях с виртуальными каналами путь пакета определяется по содержащемуся в нем виртуальному номеру линии связи, при этом в коммутаторах необходимо хранить информацию обо всех текущих соединениях. В дейтаграммных сетях передача пакета осуществляется с помощью конечного адреса и не зависит от установленного соединения.

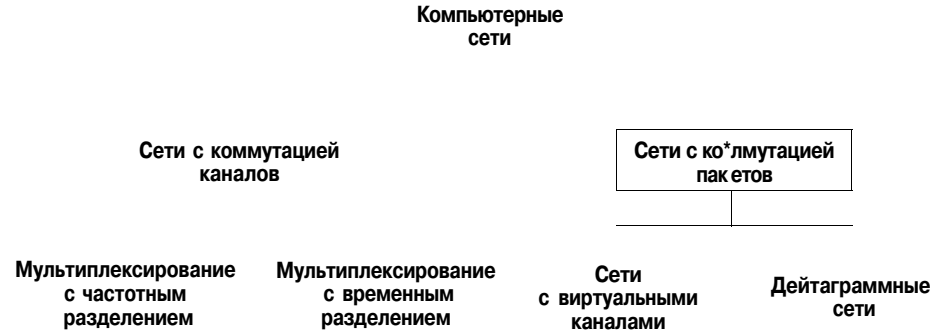


Рис. 1.13. Классификация компьютерных сетей

Следует отметить, что дейтаграммные сети по виду службы нельзя отнести ни к сетям с установлением логического соединения, ни к сетям без установления логического соединения, поскольку приложения для дейтаграммных сетей могут использовать любую из этих служб. В частности, подобную возможность предоставляет Интернет, являющийся дейтаграммной сетью. Именно поэтому существуют два принципиально различных Интернет-протокола, TCP и UDP, уже упоминавшиеся ранее в этой главе. Заметим также, что сети с виртуальным каналом всегда используют логическое соединение.

Доступ к сети и ее физическая среда

В разделах «Периферия компьютерных сетей» и «Ядро компьютерных сетей» мы определили роль оконечных систем и маршрутизаторов в процессе передачи данных. Теперь предметом обсуждения для нас станет *доступ к сети,* то есть физическая линия связи, соединяющая оконечную систему с *периферийным маршрутизатором* — первым маршрутизатором на любом пути, исходящем из оконечной системы. На рис. 1.14 представлены различные варианты подключений оконечных систем к периферийным маршрутизаторам; линии доступа выделены. Поскольку доступ к сети тесно связан с ее физической средой (волоконно-оптическими или коаксиальными кабелями, витыми парами, радиосвязью), эти две темы будут обсуждаться совместно.

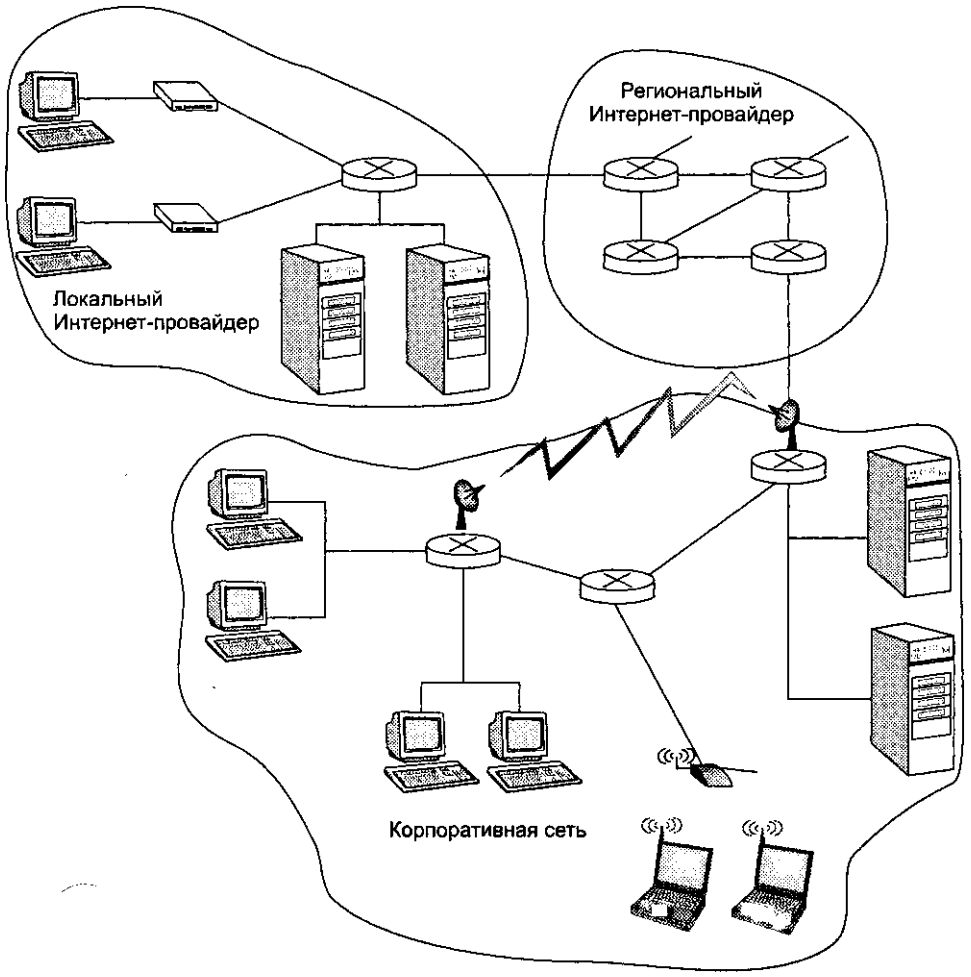


Рис. 1.14. Сети доступа

Доступ к сети

Доступ к сети можно условно классифицировать следующим образом:

- резидентный доступ используется для подключения к сети домашних оконечных систем;
- корпоративный доступ используется для подключения к сети оконечных систем, находящихся в частных и государственных организациях;
- мобильный доступ используется для подключения к сети различных портативных систем.

Заметим, что такое деление является весьма условным. Нередко на практике встречаются случаи нарушения приведенной классификации, когда компании, к примеру, используют резидентный тип доступа и т. п. Нам потребуется понятие типа доступа лишь в контексте наиболее типичных случаев.

Резидентный доступ

Резидентный доступ используется домашними оконечными системами, большинство из которых представляют собой обычные персональные компьютеры; тем не менее на сегодняшний день класс домашних систем расширяется, и к ним следует отнести web-телевизоры и некоторые другие устройства. Как правило, резидентный доступ осуществляется с помощью модема и коммутируемой телефонной линии посредством подключения к сети местного Интернет-провайдера. Модем преобразовывает цифровые сигналы домашней системы в аналоговые сигналы, которые передаются через телефонный кабель, представляющий собой медную витую пару (которая будет рассмотрена немного позже). Передаваемый сигнал принимается стороной Интернет-провайдера, где модем осуществляет обратное преобразование аналоговых сигналов в цифровые и передает их на вход маршрутизатора. Таким образом, типичный резидентный доступ к сети обеспечивается парой модемов, соединенных посредством коммутируемой телефонной линии. На сегодняшний день модемы позволяют обеспечить скорость передачи данных до 56 Кбит/с, однако из-за низкого качества телефонных линий связи, как правило, указанной скорости достичь не удается.

Многие пользователи испытывают раздражение, считая скорость передачи модемов чрезвычайно низкой. Например, трехминутная музыкальная композиция в формате MP3 стандартного качества с помощью модема обычно загружается в течение приблизительно 8 мин. Кроме того, значительное неудобство вызывает использование модемом телефонной линии, что делает невозможным совмещение телефонных разговоров с работой в Интернете. Эта проблема получила свое решение в виде новых широкополосных средств сетевого доступа, обеспечивающих более высокие скорости передачи информации, а также освобождающих телефонную линию для голосовых звонков. Существуют два основных средства широкополосного доступа: цифровые абонентские линии (Digital Subscriber Line, DSL) [133] и оптоволоконно-коаксиальные кабели (Hybrid Fiber Coaxial Cable, HFC) [59].

В январе 2001 года широкополосный доступ был мало популярен по сравнению с модемным: в Южной Корее его использовали 9,2 % пользователей, в Канаде —

4,2 %, а в США — лишь 2,2 %. В Европе распространение широкополосного доступа характеризовалось еще более скромными цифрами. Тем не менее в мире наблюдается тенденция к более широкому применению технологий DSL и HFC, причем у американских пользователей большей популярностью пользуется HFC, а пользователи из европейских и азиатских стран больше склоняются к DSL [504].

Как правило, DSL-доступ обеспечивается телефонными компаниями, иногда совместно с Интернет-провайдерами. DSL-технология во многом напоминает обычный модемный доступ по телефонному кабелю, однако позволяет за счет сокращения расстояния от пользователя до модема Интернет-провайдера значительно повысить скорость обмена. Обычно скорости обмена между сторонами асимметричны, причем скорость передачи в сторону пользователя значительно выше, чем скорость передачи в сторону Интернет-провайдера. Это обусловливает наиболее эффективное использование DSL для загрузки информации из глобальной сети. Теоретически технология DSL способна обеспечить скорость передачи более 10 Мбит/с в сторону пользователя и более 1 Мбит/с в сторону Интернет-провайдера; на практике скорости являются значительно более низкими и по результатам 2002 года лежат в промежутках от 384 Кбит/с до 1,5 Мбит/с и от 128 до 256 Кбит/с соответственно.

В технологии DSL используется частотное мультиплексирование, речь о котором шла в предыдущем разделе. Обычно диапазон частот линии связи между пользовательской системой и Интернет-провайдером делится на три неперекрывающиеся полосы:

- высокоскоростной приемный канал с диапазоном от 50 кГц до 1 МГц;
- среднескоростной передающий канал с диапазоном от 4 до 50 кГц;
- телефонный канал с диапазоном от 0 до 4 кГц.

Суммарная полоса частот приемного и передающего каналов зависит от трех факторов: расстояния между модемами пользователя и Интернет-провайдера, длины витой пары и электрических помех. Технология DSL наилучшим образом применима для коротких расстояний, где ее преимущества по скорости перед обычными модемными соединениями проявляются в наибольшей степени.

Если коммутируемые модемные соединения и DSL используют обычные телефонные линии связи, то в технологии HFC для передачи требуются линии кабельного телевидения. В традиционной кабельной системе распределительное устройство осуществляет вещание через сеть, состоящую из коаксиального кабеля и усилителей, к которой подключены сотни домов. Как показано на рис. 1.15, оптоволоконный кабель соединяет распределительное устройство с передатчиками, к которым с помощью коаксиального кабеля подключены непосредственные пользователи. Каждый передатчик способен обслуживать от 500 до 5000 абонентов.

Как и DSL, технология HFC требует наличия у пользователя специального устройства, называемого кабельным модемом. Компании, предоставляющие услуги HFC, обычно предлагают абонентам либо приобрести кабельный модем, либо взять его в аренду. Как правило, кабельный модем представляет собой внешнее устройство, подключающееся к домашнему персональному компьютеру через порт 10-BaseT Ethernet (подробный разговор о Ethernet-сети и соответствующих

стандартах пойдет в главе 5). Полоса пропускания линий связи в сетях HFC делится на два канала: передающий и принимающий (относительно пользователя). Как и в случае с DSL, принимающий канал обычно имеет более широкую полосу пропускания и, следовательно, обеспечивает большую скорость по сравнению с передающим каналом. Особенностью технологии HFC, контрастирующей с DSL, заключается в том, что скорости каналов являются ресурсом, разделяемым между всеми пользователями сети. Мы рассмотрим это подробнее чуть позже.

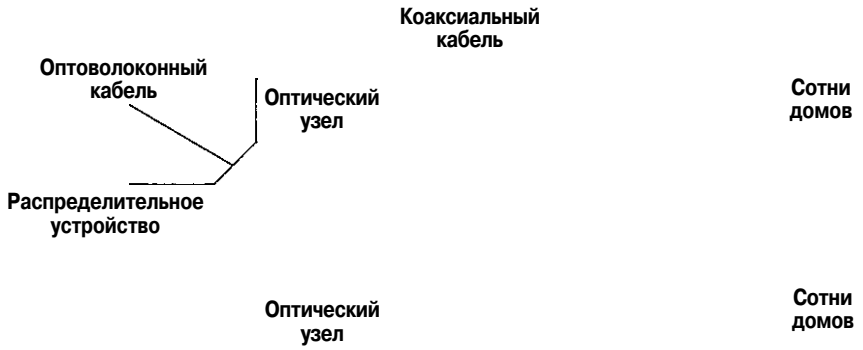


Рис. 1.15. Сеть на основе оптоволоконно-коаксиальных кабелей

Важной характеристикой HFC является широковещательность. Так, любой пакет, посылаемый распределительным устройством, передается каждому из абонентов, в то время как каждый пользовательский пакет по общему передающему каналу приходит в единое распределительное устройство. Если несколько пользователей одновременно занимаются загрузкой файлов из сети, то скорость загрузки каждого файла будет гораздо меньше скорости передачи линии связи. С другой стороны, если в сети находится небольшое число активных пользователей, путешествующих по web-страницам, то загрузка web-страниц будет происходить на скорости, близкой к максимальной, потому что вероятность отправки запросов одновременно несколькими пользователями невелика. Аналогичная ситуация наблюдается и с разделяемым передающим каналом: если несколько пользователей начнут совместную передачу своих пакетов, это приведет к столкновениям последних (разговор о столкновениях, или коллизиях, в компьютерных сетях пойдет в главе 5, где будет обсуждаться технология Ethernet). Сторонники DSL замечают, что в их технологии организуется прямое соединение между оконечными системами и Интернет-провайдерами, то есть пользователи получают выделенный, а не разделяемый ресурс. Их оппоненты, предпочитающие HFC, обращают внимание на то, что кабельные сети с ограниченным числом абонентов обеспечивают более широкую полосу пропускания. Таким образом, можно свидетельствовать, что между двумя технологиями имеет место конкуренция.

Несомненным достоинством, которым обладают обе технологии, является непрерывность обслуживания. Другими словами, пользователь может совершать обычные телефонные звонки, не разрывая соединение с сетью.

Корпоративный доступ

Как правило, в государственных и частных организациях доступ к Интернету осуществляется при помощи локальных сетей (LAN), соединяющих оконечные системы с периферийным маршрутизатором. Как мы убедимся в главе 5, существует множество технологий локальных сетей, наиболее распространенной из которых является Ethernet. Ethernet обеспечивает типичные скорости передачи 10 и 100 Мбит/с; последними достижениями явились скорости 1 и 10 Гбит/с. В Ethernet для соединения оконечных систем между собой и с периферийным маршрутизатором используется либо коаксиальный кабель, либо медная витая пара. Периферийный маршрутизатор позволяет организовать обмен информацией с внешней по отношению к локальной сети средой. Как и в сетях HFC, в Ethernet скорость передачи является ресурсом, разделяемым между пользователями. В настоящее время наблюдается движение в сторону коммутируемой Ethernet-сети, в которой сегменты на основе витой пары подключаются к «коммутатору», обеспечивающему пользователям доступ ко всей полосе пропускания одновременно. Мы рассмотрим разделяемую и коммутируемую Ethernet-сети в главе 5.

Мобильный доступ

Прорыв в области беспроводных технологий, как и возникновение глобальной Сети, привел к значительным изменениям в сфере телекоммуникаций. В 2000 году в Европе было больше владельцев мобильных телефонов, чем владельцев машин или персональных компьютеров. Специалисты прогнозируют, что портативные беспроводные устройства будут развиваться и к 2004 году вытеснят обычные персональные компьютеры в качестве основного устройства доступа в Интернет. В настоящее время существуют два основных средства беспроводногб подключения к глобальной Сети. *Беспроводные локальные сети* позволяют пользователям обмениваться данными через базовую станцию, часто называемую *точкой беспроводного доступа,* находясь на расстоянии десятков метров от нее. Как правило, базовая станция имеет подключение к Интернету с помощью кабеля и способна соединить пользователей с глобальной Сетью. В *беспроводных сетях с удаленным доступом* базовая станция управляется поставщиком телекоммуникационных услуг и обеспечивает доступ пользователей на расстоянии до десятков километров.

Беспроводные локальные сети, основанные на технологии IEEE 802.11b, известной как беспроводная Ethernet-сеть и Wi-Fi, в настоящее время получают массовое распространение в различных учебных, коммерческих, развлекательных организациях и даже в домашнем пользовании. Подобные технологии, реализованные в здании, позволяют пользователям задействовать электронную почту или заниматься путешествиям по web-страницам в любой точке этого здания. Как мы увидим в главе 5, технология IEEE 802.1 lb дает возможность получить скорость передачи данных до 11 Мбит/с.

Сейчас наблюдается расширение круга домашних пользователей, которые в сочетании с широкополосным резидентным доступом (HFC или DSL) применяют недорогие беспроводные локальные сети для создания мощных домашних компьютерных сетей. Возможная схема такой сети изображена на рис. 1.16 и используется

одним из авторов этой книги. Сеть включает в себя портативный и персональный компьютеры, базовую станцию, с которой связан портативный компьютер, кабельный модем, осуществляющий подключение персонального компьютера к Интернету, и, наконец, маршрутизатор, соединяющий базовую станцию и персональный компьютер с кабельным модемом. Описанная сеть позволяет двум членам семьи иметь широкополосный доступ в Интернет, причем один из них может при этом бродить из комнаты в комнату. Стоимость подобной сети составляет менее 500 долларов (включая стоимость кабельного модема) и постоянно падает.

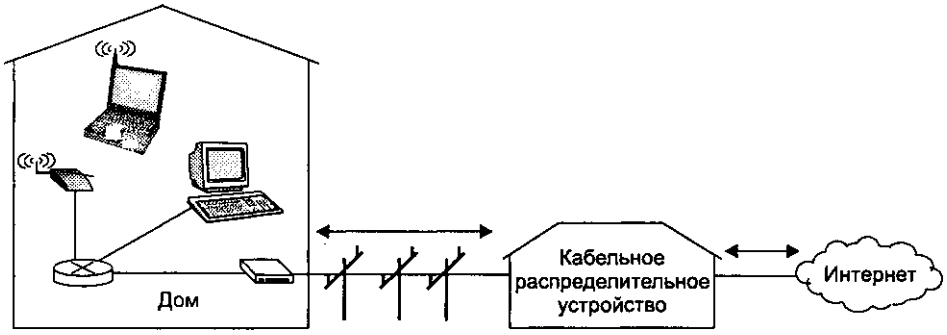


Рис. 1.16. Схема типичной домашней компьютерной сети

Получая доступ в Интернет через беспроводную локальную сеть, вы связаны необходимостью находиться на расстоянии не более нескольких десятков метров от базовой станции. Это допустимо для домашнего или корпоративного пользователя. Однако как быть, например, если вы находитесь в машине или за городом на отдыхе? Здесь на помощь приходят технологии мобильной телефонной связи, которые обеспечивают доступ к глобальной Сети на расстоянии десятков километров от базовой станции.

Технологии WAP (Wireless Access Protocol — протокол беспроводного доступа) и i-mode являются технологиями мобильной связи с Интернетом. WAP-телефоны напоминают обычные мобильные телефоны, однако имеют несколько увеличенный экран и обеспечивают низкоскоростной доступ в Интернет. Вместо языка HTML в WAP-телефонах используется язык разметки WML (WAP Markup Language), оптимизированный под низкоскоростной доступ и небольшой экран. Протокол WAP в Европе поддерживается стандартом мобильной связи GSM, использующим временное мультиплексирование. До настоящего времени популярность WAP в Европе находилась на крайне низком уровне, однако в связи с распространением технологии GPRS (General Packet Radio Service — основная служба радиотрансляции пакетов) в 2002-2003 годах ожидается рост популярности WAP. Отметим, что технология i-mode, концептуально и функционально весьма близкая WAP, имела большой успех в Японии.

Сейчас телекоммуникационные компании делают большие инвестиции в беспроводные технологии *третьего поколения* (Third Generation, 3G), которые позволят осуществлять удаленный беспроводной доступ в Интернет с коммутацией паке-

тов на скоростях не ниже 384 Кбит/с. 3в-системы обеспечат высокоскоростной доступ к web-ресурсам и интерактивному видео, а также телефонную связь с более высоким качеством звука, чем в обычных телефонных сетях. Первые 3в-системы уже находят применение в Японии. Большие инвестиции в 3в-технологию подогревают интерес и любопытство всего телекоммуникационного сообщества: оправдаются ли надежды инвесторов? Сможет ли новинка выдержать конкуренцию с IEEE 802.11? Ответ на этот вопрос даст лишь время (в [552], а также в разделе «Беспроводные каналы связи» главы 5 вы можете найти дополнительную информацию о 3С-технологии).

Физическая среда передачи

В предыдущем подразделе мы рассказали о фундаментальных технологиях доступа в Интернет. Когда мы описывали эти технологии, мы говорили о физической среде, использующейся для соединения. Так, например, в технологии HFC применяют сочетание оптоволоконных линий связи и коаксиального кабеля, в коммутируемых телефонных линиях — медную витую пару, а в беспроводных технологиях доступа — электромагнитные волны радиодиапазона. Здесь мы опишем эти, а также некоторые другие среды передачи данных в Интернете.

Для того чтобы ввести понятие физической среды передачи, рассмотрим, что происходит в процессе передачи данных. Представьте себе бит, который необходимо передать от одной оконечной системы к другой. Сначала отправляющая система передает бит первому маршрутизатору, и, спустя некоторый промежуток времени, маршрутизатор получает его. Далее первый маршрутизатор пересылает бит второму маршрутизатору, и, спустя еще один промежуток времени, второй маршрутизатор получает этот бит. Таким образом, достигая своего конечного назначения, бит проходит через последовательность передающих и принимающих пар устройств. Передача между этими парами происходит путем распространения электромагнитных волн или оптических сигналов в *физической среде.* Физическая среда может принимать весьма разнообразные формы, причем на пути следования пакета эти формы могут меняться. Примерами физических сред являются медная витая пара, коаксиальный кабель, многомодовый оптоволоконный кабель, территориальные и спутниковые радиоканалы. Физические среды можно разделить на два типа: *проводные* и *беспроводные.* Проводные среды передачи предполагают наличие твердотельного проводника и включают оптоволоконный кабель, медную витую пару и коаксиальный кабель. В беспроводной среде передача осуществляется без участия твердых проводников; этот тип среды используется в беспроводных локальных сетях и в спутниковой связи.

Перед тем как приступать к изучению характеристик различных сред передачи данных, хотелось бы сказать несколько слов об их стоимости. Фактическая стоимость физической линии связи (медного кабеля, оптоволоконного кабеля и т. д.), как правило, незначительна по сравнению со стоимостью других сетевых компонентов. Более того, стоимость работ по прокладке линий связи зачастую на порядок выше, чем стоимость самих линий. По этой причине многие заказчики компьютерных сетей предпочитают одновременно прокладывать несколько типов кабеля

(оптоволоконный, коаксиальный или витую пару) в каждом помещении здания. Даже если изначально в сети используется лишь один из проложенных кабелей, не исключена возможность, что через некоторое время возникнет потребность в другом.

Медная витая пара

Медная витая пара является самым дешевым и наиболее популярным видом кабелей. На протяжении более чем 100 лет витая пара активно используется в телефонных сетях. Можно смело утверждать, что более 99 % всех кабелей, соединяющих абонентов с телефонными коммутаторами, являются медными витыми парами. Многие могли видеть эти кабели у себя дома или на работе. Витая пара состоит из двух изолированных медных проводов толщиной 1 мм, заключенных в спиральную оболочку. Внутри оболочки провода переплетены друг с другом, чтобы снизить уровень электрических помех, возникающих между парой проводников. Обычно перед помещением пар внутрь кабеля их снабжают дополнительными защитными экранами.

Неэкранированная витая пара (Unshielded Twisted Pair, UTP), как правило, используется в офисных локальных сетях, расположенных в одном здании. Скорость передачи данных в такой среде варьируется от 10 Мбит/с до 1 Гбит/с и определяется толщиной провода и расстоянием между обменивающимися сторонами. В локальных сетях используется два типа неэкранированных витых пар: витая пара категории 3 и витая пара категории 5. Первая относится к голосовым линиям связи и характерна для офисов. Как правило, в офисах прокладывают две независимые витые пары, из которых одна используется для телефонной связи, а другая — для дополнительных телефонных соединений и локальной сети. В частности, в широко распространенной технологии Ethernet 10 Мбит/с применяют неэкранированную витую пару категории 3. Витая пара категории 5 имеет большее число витков на дюйм, а также снабжена тефлоновой изоляцией, что позволяет обеспечить более высокие скорости передачи данных. В последние годы получила распространение технология Ethernet 100 Мбит/с, в которой используется витая пара категории 5. В новых офисных компьютерных сетях, как правило, также прокладывается неэкранированная витая пара категории 5.

С появлением в 80-е годы оптоволоконных линий связи многие специалисты прогнозировали, что они со временем полностью вытеснят низкоскоростную витую пару. Однако витая пара оказалась не столь бесперспективной. Неэкранированная витая пара категории 5 позволяет получить скорость передачи данных 100 Мбит/с на расстояниях до нескольких сотен метров. На меньших расстояниях можно добиться еще большей скорости. Не удивительно, что этот тип кабеля еще долго может занимать доминирующее положение в сфере локальных офисных сетей.

Как было сказано ранее, витая пара активно используется для резидентного доступа в Интернет. С ее помощью реализуется коммутируемый модемный доступ на скоростях до 56 Кбит/с, а также DSL-доступ, позволяющий резидентным абонентам достичь скорости 6 Мбит/с.

Коаксиальный кабель

Коаксиальный кабель, как и витая пара, состоит из двух медных проводников, однако эти проводники, в отличие от витой пары, расположены не параллельно, а концентрически (коаксиально). С применением особых видов изоляции и экранирования коаксиальный кабель позволяет добиться более высоких скоростей передачи данных, чем витая пара. Коаксиальные кабели делятся на два вида: *с немодулируемой передачей* и *с модулируемой передачей*.

Коаксиальный кабель с немодулируемой передачей имеет сопротивление 50 Ом и толщину около 1 см; к его несомненным физическим достоинствам можно отнести легкость и гибкость. Этот тип кабеля часто применяется в локальных сетях наряду с неэкранированной витой парой. Внимательно рассмотрите соединение кабеля с сетевой картой вашего компьютера. Если разъем по форме совпадает с телефонным, а кабель внешне напоминает телефонный провод, то используется витая пара UTP; если же вы увидите Т-образный разъем и кабель, выходящий из обеих сторон разъема, то используется коаксиальный кабель с немодулируемой передачей. Термин «с немодулируемой передачей» означает, что битовый поток поступает в кабель без частотной модуляции. Хотя коаксиальный кабель может применяться в технологии Ethernet 10 Мбит/с, почти для всех новых реализаций Ethernet характерна неэкранированная витая пара.

Коаксиальный кабель с модулируемой передачей обладает сопротивлением 75 Ом и имеет большие толщину, вес и меньшую гибкость по сравнению с кабелем с немодулируемой передачей. Часто кабель с модулируемой передачей используют в системах кабельного телевидения. Как мы уже говорили, линии кабельного телевидения в сочетании со специальными кабельными модемами способны обеспечить связь пользователей с Интернетом на скорости 1 Мбит/с и выше. При передаче по кабелю с модулируемой передачей происходит предварительная модуляция (перенос) аналоговых сигналов в нужную полосу частот. Оба вида кабеля (с немодулируемой и с модулируемой передачей) относятся к классу *разделяемых проводных сред* передачи данных. Другими словами, информация, передаваемая или принимаемая одной системой, принимается всеми системами, подключенными к кабелю.

Оптоволоконный кабель

Оптоволоконная среда передачи представляет собой тонкий и гибкий кабель, внутри которого распространяются световые импульсы, несущие информацию о передаваемых битах. Даже простой оптоволоконный кабель способен передавать данные на огромных скоростях в десятки и даже сотни гигабит в секунду. Оптоволоконные линии не подвержены электрическим наводкам, имеют очень низкий уровень ослабления сигнала на единицу протяженности и обладают значительной устойчивостью к механическим воздействиям. Перечисленные преимущества сделали оптоволоконные линии связи весьма привлекательной технологией для передачи информации на большие расстояния, особенно для международных и межконтинентальных коммуникаций. В США многие телефонные линии связи большой протяженности реализованы с помощью оптоволоконных кабелей. Оптоволоконная среда активно используется для передачи данных в Интернете. Однако высо-

кая стоимость оптических устройств (маршрутизаторов, приемников и передатчиков) делает нецелесообразным (по экономическим причинам) применение оптоволоконных линий связи для передачи на короткие расстояния, например, в локальных офисных сетях или для резидентного домашнего доступа. Более полную информацию по оптоволоконным сетям можно получить в [187, 188,407].

Территориальные радиоканалы

Радиоканалы передают сигналы с помощью электромагнитных волн радиодиапазона. Их достоинство заключается в том, что для связи не требуется твердотельного проводника сигналов (следовательно, нет необходимости в его прокладке), то есть пользователь может быть мобильным, есть потенциал в увеличении расстояния передачи. Характеристики радиоканала зависят от среды передачи радиоволн и расстояния между оконечными системами. К факторам среды передачи относятся затухание сигнала вследствие распространения в среде, прохождения через поглощающие предметы, взаимодействия с отраженными электромагнитными волнами, а также волнами, исходящими от других источников излучения.

Территориальные радиоканалы можно разделить на каналы локального распространения сигнала, когда расстояния не превосходят сотен метров, и каналы удаленного распространения сигнала, когда расстояния превышают несколько десятков километров. Первые используются в локальных сетях, а вторые — для мобильного доступа (см. предыдущий подраздел). Более полную информацию по товарам и технологиям для территориальных радиоканалов можно получить в [129].

Спутниковые радиоканалы

Спутник связи организует взаимодействие между двумя или более наземными приемопередатчиками. Он принимает сигналы одного частотного диапазона, производит их регенерацию с помощью повторителя (см. далее), а затем передает сигналы в другом частотном диапазоне. Скорость обмена данными, обеспечиваемая спутниковыми каналами, составляет несколько гигабит в секунду. Существуют два типа спутников: *геостационарные* и *низкоорбитальные.*

Отличительной чертой геостационарных спутников является то, что они не меняют своего положения относительно заданной точки земной поверхности. Это достигается путем помещения спутника на орбиту, удаленную приблизительно на 36 000 км от земной поверхности. Значительное расстояние, которое приходится преодолевать сигналу, обусловливает большую задержку его распространения, составляющую 250 мс. Тем не менее спутниковые каналы, с помощью которых не составляет труда достичь скоростей передачи в сотни мегабит в секунду, активно используются в телефонии и Интернете.

Низкоорбитальные спутники находятся значительно ближе к земной поверхности, чем геостационарные, и вращаются вокруг нее, подобно Луне. Для постоянного покрытия определенных участков земной поверхности приходится размещать на орбите несколько спутников. В настоящее время имеется немалое число проектов низкоорбитальных систем связи [558]. В частности, в будущем предполагается использование подобных систем для передачи данных в Интернете.

Интернет-провайдеры и магистрали Интернета

Как мы видели ранее в этой главе, оконечные системы подключаются к Интернету с помощью сетей доступа. Сеть доступа может быть проводной или беспроводной локальной сетью (в компании, в университете или в государственном учреждении) или предоставляться Интернет-провайдером через обычный модем, выделенную линию или кабельный модем. Разумеется, объединение в единую сеть сетей Интернет-провайдеров и их абонентов является «каплей в море» задач, связанных с обменом информацией между сотнями миллионов пользователей по всему миру. Интернет представляет собой гигантскую *сеть сетей]* этот термин отражает ключевую особенность архитектуры Интернета.

В общедоступном Интернете сети доступа соединяются с «остальной» сетью при помощи иерархии сетей Интернет-провайдеров, изображенной на рис. 1.17. Внизу иерархии находятся сети резидентных Интернет-провайдеров, к которым обычно подключаются оконечные системы. Верхняя часть иерархии представлена сетями так называемых *Интернет-провайдеров первого звена.* С одной стороны, сети этих Интернет-провайдеров обладают типичными чертами компьютерных сетей — наличием маршрутизаторов и связей с другими сетями. С другой стороны, сети Интернет-провайдеров первого звена имеют свою специфику. Во-первых, их линии связи обычно обеспечивают скорость передачи не ниже 622 Мбит/с, а иногда 2,5-10 Гбит/с. Во-вторых, маршрутизаторы сетей Интернет-провайдеров первого звена должны функционировать с предельно высокой скоростью для того, чтобы не вызывать задержек пакетов. В-третьих, все сети Интернет-провайдеров первого звена напрямую соединены между собой. В-четвертых, к каждой сети Интернет-провайдера первого звена подключено большое количество сетей Интернет-провайдеров второго звена и прочих компьютерных сетей. Наконец, в-пятых, область сетевого охвата Интернет-провайдера первого звена является международной.

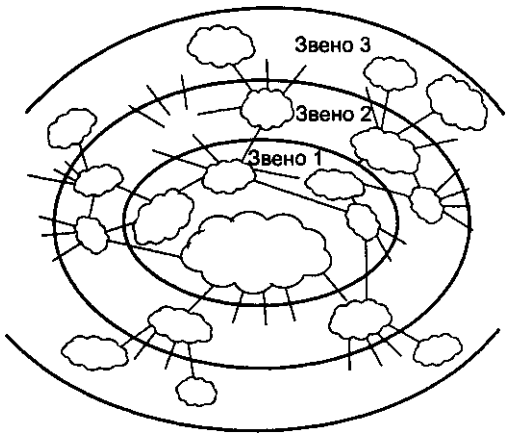


Рис. 1.17. Взаимосвязи между сетями Интернет-провайдеров

Сети Интернет-провайдеров первого звена часто называют *магистралями* Интернета. К магистральным компаниям относятся UUNet, Sprint, AT&T, Genuity, Cable and Wireless и др.

Сети Интернет-провайдеров второго звена, как правило, имеют региональную территорию охвата и подключаются к нескольким сетям Интернет-провайдеров первого звена (см. рис. 1.17). Говорят, что Интернет-провайдеры второго звена являются *потребителями услуг* Интернет-провайдеров первого звена. Крупные компании и учреждения подключают свои корпоративные сети напрямую к сетям Интернет-провайдеров второго и даже первого звеньев и считаются потребителями их услуг. Потребители оплачивают услуги своих Интернет-провайдеров, при этом стоимость обычно зависит от скорости соединений с последними. Сети Интернет-провайдеров второго звена также могут соединяться между собой и обмениваться информацией без участия Интернет-провайдеров первого звена. Ниже в иерархии расположены сети Интернет-провайдеров, которые подключаются к сетям Интернет-провайдеров второго звена (к одной или нескольким). На последней ступени иерархии находятся сети доступа. Последний аспект, несколько запутывающий предыдущую концепцию, заключается в^том, что некоторые Интернет-провайдеры первого звена одновременно могут являться Интернет-провайдерами второго звена, к сетям которых подсоединены сети крупных организаций и Интернет-провайдеров более низких звеньев. Когда сети двух Интернет-провайдеров непосредственно соединены друг с другом, говорят, что они являются *одноранговыми*.

Точки, в которых сеть Интернет-провайдера связывается с сетями других Интернет-провайдеров (расположенных выше, ниже или на одном иерархическом уровне), называются *точками присутствия* (Points of Presence, POP). Как правило, точка присутствия представляет собой одну или несколько групп маршрутизаторов сети, к которым подключены маршрутизаторы другой сети. У Интернет-провайдеров первого звена обычно имеется множество точек присутствия в различных географических регионах. Для подключения к Интернет-провайдеру более высокого звена потребитель обычно арендует высокоскоростные линии связи у какой-либо телекоммуникационной компании (являющейся «третьей стороной» в сделке) и соединяет свои маршрутизаторы с точкой присутствия Интернет-провайдера. Возможны соединения двух сетей одновременно в нескольких точках присутствия.

Помимо соединения через точки присутствия, используется также механизм соединения через точки доступа в сеть (Network Access Points, NAP), каждая из которых может принадлежать сторонней телекоммуникационной компании или магистральному Интернет-провайдеру и управляться ими. Обычно подобная схема используется при подключении сетей Интернет-провайдеров второго звена друг к другу и к сетям Интернет-провайдеров первого звена, а Интернет-провайдеры первого звена чаще предпочитают соединять свои сети между собой через точки присутствия. Поскольку в точках доступа необходимо обеспечивать коммутацию и передачу огромных объемов информации в единицу времени, их реализация весьма непроста технологически. В точках доступа в сеть часто ис-

пользуется высокоскоростная технология ATM (рассматриваемая в главе 5) с протоколом IP. Аппаратура точек доступа обычно локализована в одном здании.

Как вы, вероятно, убедились, топология Интернета является сложной и состоит из нескольких десятков сетей Интернет-провайдеров первого и второго звеньев и сотен сетей менее крупных Интернет-провайдеров регионального и локального масштаба. Последние подключаются к первым, которые, в свою очередь, соединены между собой при помощи точек присутствия либо точек доступа. Чаще всего Интернет-провайдеры соседних звеньев находятся в отношениях «поставщик-потребитель».

Заметим, что потенциально любой из вас может организовать сеть доступа и стать Интернет-провайдером. Для этого необходимо иметь лишь подключение к Интернету и соответствующее оборудование (например, маршрутизатор и модемный пул). Таким образом, расширение Интернета происходит по тому же принципу, по которому ребенок строит здание из элементов конструктора: к уже построенной части добавляются новые и новые фрагменты.

Задержки и потери данных в сетях с коммутацией пакетов

Ознакомившись с основными элементами архитектуры Интернета — приложениями, оконечными системами, протоколами передачи, маршрутизаторами и линиями связи, давайте подробнее рассмотрим процесс передачи пакета от одной оконечной системы к другой. Перемещаясь от отправителя к получателю, пакет проходит через последовательность маршрутизаторов и линий связи. Каждый узел на пути пакета вызывает различные виды задержек пакета, наиболее значимыми из которых являются *задержка узловой обработки, задержка ожидания, задержка передачи* и *задержка распространения.* Сумма всех перечисленных задержек называется *суммарной узловой задержкой* пакета. Для углубленного представления процесса коммутации пакетов в частности и работы компьютерных сетей в целом необходимо хорошо представлять природу и влияние каждого из видов задержек на передачу пакетов.

Виды задержек

Давайте рассмотрим перечисленные выше виды задержек (рис. 1.18). Путь передаваемого пакета включает фрагмент от передающего хоста до маршрутизатора B, внутри которого находится маршрутизатор A. Линия связи, соединяющая маршрутизаторы A и B, имеет выходной буфер со стороны маршрутизатора A, предназначенный для хранения пакетов, находящихся в очереди. Приняв пакет, маршрутизатор A анализирует его заголовок, чтобы определить направление дальнейшей передачи пакета, и отсылает пакет в нужную линию связи. Очевидно, что мгновенное начало дальнейшей передачи возможно лишь в том случае, когда линия связи свободна, то есть по ней не ведется передача других пакетов, а также при отсутствии очереди на передачу. Если хотя бы одно из этих двух условий не выполняется, пакет будет вынужден встать в очередь.

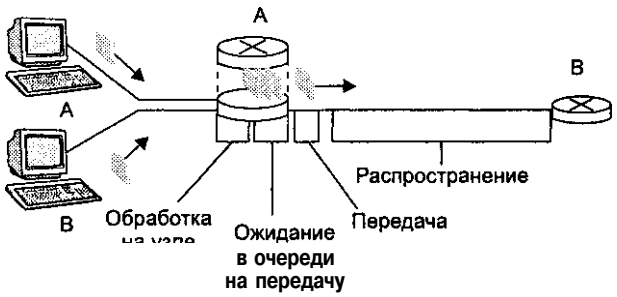


Рис. 1.18. Узловая задержка в маршрутизаторе A

Задержка узловой обработки

Время, необходимое для чтения заголовка пакета и определения дальнейшего маршрута, составляет часть *задержки узловой обработки.* На задержку обработки могут также оказывать влияние и другие факторы, например необходимость проверки искажений битов пакета при передаче. Типичным временем задержки обработки в высокоскоростных маршрутизаторах являются единицы микросекунд. После окончания обработки пакета маршрутизатор при необходимости помещает его в очередь линии связи с маршрутизатором B. Мы вернемся к принципам работы маршрутизаторов в главе 4, где рассмотрим их более детально.

Задержка ожидания

Находясь в очереди, пакет подвергается *задержке ожидания* дальнейшей передачи по линии связи к маршрутизатору B. Время задержки ожидания зависит от числа пакетов, стоящих в очереди, и может значительно варьироваться в различных маршрутизаторах на пути пакета. Если загрузка линии связи невысока, то время ожидания пакета, как правило, либо нулевое, либо незначительное, однако в случае перегруженности линии оно может многократно увеличиться. Позже мы увидим, что длина очереди на момент появления очередного пакета является функцией интенсивности и характера трафика пакетов. Как правило, задержка ожидания составляет от нескольких микросекунд до нескольких миллисекунд.

Задержка передачи

Предполагая, что пакеты обслуживаются в порядке их поступления в очередь (такая модель обслуживания является доминирующей в сетях с коммутацией пакетов), мы приходим к выводу, что наш пакет будет передан после окончания передачи всех пакетов, стоящих в очереди перед ним. Пусть L — длина пакета, а R — скорость передачи пакета по линии связи, тогда задержка передачи равна L/R. *Задержку передачи* также называют задержкой накопления (см. раздел «Ядро компьютерных сетей» в этой главе). Как и задержка ожидания, задержка распространения варьируется от нескольких микросекунд до нескольких миллисекунд.

Задержка распространения

После генерации сигнала, несущего информацию о передаваемом бите, этот сигнал распространяется по линии связи, достигая маршрутизатора B. Время, необходимое для передачи сигнала по линии связи, называется *задержкой распространения* и определяется длиной линии и физическими свойствами передающей среды (оптоволокна, меди в витой паре и т. п.). Скорость распространения сигнала лежит в пределах от 2×10^8 м/с до 3×10^8 м/с, то есть сравнима со скоростью света. Чтобы определить задержку распространения для конкретной линии связи, необходимо разделить ее длину на скорость распространения сигнала. Чаще всего время распространения составляет несколько миллисекунд.

Окончание приема последнего бита пакета маршрутизатором B заканчивает цикл передачи пакета. После этого весь описанный выше процесс повторяется для маршрутизатора B и всех остальных маршрутизаторов на пути пакета.

Сравнение задержки передачи и задержки распространения

Те, кто впервые приступает к изучению компьютерных сетей, нередко не могут уяснить разницы между задержкой передачи и задержкой распространения. Действительно, разница между этими понятиями хотя и не очевидна, но весьма важна. Задержка передачи — это суммарное время, требуемое для освобождения пакетом места в буфере и зависящее от скорости передачи по линии связи и размера пакета, но не от длины линии связи. Задержка распространения — это время, требуемое для передачи бита по линии связи, зависящее от ее длины, однако никак не зависящее ни от длины пакета, ни от скорости передачи.

Рассмотрим следующую аналогию. Представим себе шоссе, на котором через каждые 100 километров расположены пункты для сбора пошлин. Эти пункты являются аналогами маршрутизаторов, а участки шоссе — аналогами линий связи. Предположим, что автомобиль движется по шоссе («распространяется») со скоростью 100 км/ч; для простоты будем считать, что после прохождения очередного пункта автомобиль набирает требуемую скорость мгновенно. Пусть имеется колонна из 10 одинаковых автомобилей, движущихся в установленном порядке, тогда каждый автомобиль будет аналогом бита, а колонна — аналогом пакета. Допустим, что каждый пункт обслуживает («передает») одну машину за 12 с, и на шоссе нет других машин, кроме рассматриваемых. Также допустим, что первая из прибывших на пункт машин дожидается на входе, пока оставшиеся машины соберутся в колонну (подобно пакету, который сначала целиком принимается и лишь затем проходит обработку). Время, необходимое для того, чтобы «передать» колонну на очередной участок шоссе, составляет 10 машин / (5 машин/мин) = 2 мин и является аналогом задержки передачи. Время, необходимое автомобилю для прохождения участка шоссе до следующего пункта сбора пошлин, составляет 100 км / (100 км/ч) = 1 ч и является аналогом задержки распространения. Таким образом, время, проходящее между моментами сбора колонны у пунктов сбора пошлин, составляет 62 мин.

Теперь взглянем на рассмотренную аналогию чуть подробнее. Что произойдет, если время обслуживания пунктом сбора пошлин одной колонны больше, чем время,

требуемое одной машине для прохождения участка между соседними пунктами? Пусть, например, скорость движения машины составляет не 100, а 1000 км/ч, а время обслуживания одной машины составляет 1 мин. В этом случае время обслуживания колонны составит 10 мин, а время прохождения участка шоссе — всего 6 мин. Это приведет к тому, что первые машины колонны достигнут второго пункта для сбора пошлин раньше, чем последние машины будут обслужены первым пунктом. Аналогичная ситуация часто встречается в сетях с коммутацией пакетов, когда одна часть пакета находится в ожидании обработки и передачи другой части предыдущим маршрутизатором.

Если задержки обработки, ожидания, передачи и распространения обозначить соответственно через $d^{обр}$, ($1^{ожид}$, $d^{пер}$ и $d^{расп}$, то суммарная узловая задержка $d^{уз/1}$ - $d^{обр}$ +

Значения составляющих задержек могут широко варьироваться. Так, значение $d^{расп}$ может быть ничтожно малым (несколько микросекунд) для пары маршрутизаторов, расположенных в одном здании. В то же время для спутниковой линии связи $d^{расп}$ составляет порядка десятых долей секунды и значительно превышает все остальные виды задержек. Аналогичная ситуация характерна и для задержки передачи $d^{пер}$: ее значение пренебрежимо мало при высоких скоростях передачи (10 Мбит/с и выше), например в локальных сетях; в то же время для низкоскоростных модемных соединений значение $d^{пер}$ может достигать десятков и сотен миллисекунд. Единственным видом задержки, как правило, имеющим небольшие значения, является задержка обработки $d^{обр}$. Тем не менее задержка обработки оказывает существенное влияние на максимальную скорость передачи пакетов маршрутизатором.

Задержка ожидания и потеря пакетов

Наиболее сложным и интересным видом задержек, возникающих при передаче пакетов, является задержка ожидания d^{ow} Эта величина имеет настолько важное значение для сетевых технологий, что ей посвящены десятки книг и сотни научных статей [27,108,275,276,433]. Сейчас мы не будем излишне углубляться в теорию массового обслуживания и рассмотрим задержку ожидания и ее последствия лишь в общем плане.

Задержка ожидания — единственная составляющая узловой задержки, которая может иметь разные значения для разных пакетов. Так, например, если в изначально пустой буфер маршрутизатора одновременно поступит 10 пакетов, то задержка ожидания для первого пакета будет равна нулю, а для последнего пакета она окажется равной времени, необходимому для обслуживания девяти предыдущих пакетов. Таким образом, значение задержки ожидания является случайным, и для его оценки необходимо использовать статистические величины: среднее время ожидания, дисперсию времени ожидания и вероятность превышения временем ожидания заданной величины.

Каковы факторы, влияющие на величину задержки ожидания? Можно выделить три из них: частоту получения пакетов, скорость передачи выходной линии связи и закон распределения получаемых пакетов во времени. Последний характеризует,

является ли получение пакетов периодическим или апериодическим. Пусть *a* — средняя частота получения пакетов, измеряемая в пакетах в секунду, *R* — скорость передачи по выходной линии связи в битах в секунду, а *L* — длина пакета в битах. Тогда скорость получения данных маршрутизатором равна *Lxa* бит/с. Предположим, что буфер маршрутизатора является бесконечно большим, то есть позволяет организовать очередь бесконечной длины. Величина *L* x *a/R,* называемая *интенсивностью трафика,* играет определяющую роль в оценке длины очереди. Если *L* x *a/R* > 1, то средняя скорость приема информации превышает среднюю скорость ее передачи, что приводит к неограниченному росту очереди. Отсюда вытекает «золотое правило» организации трафика: обмен информацией всегда должен быть организован так, чтобы интенсивность трафика не превышала 1. Далее будем считать, что *Lxa/R<l.*

При отсутствии перегрузок можно исследовать влияние интенсивности трафика на величину задержки ожидания. Если появление новых пакетов происходит периодически, например каждые *L/R* секунд, то каждый пакет будет приходить в пустой буфер, и, следовательно, время ожидания окажется равным нулю. Если новые пакеты появляются периодически, но не поодиночке, то среднее время ожидания для них, очевидно, уже не будет нулевым. Пусть каждые *(L/R)* x *N* секунд происходит появление *N* пакетов, тогда для первого пакета время ожидания будет равно 0 с, для второго пакета — *L/R* с, а для *n-го* пакета — *(n* - 1) x *L/R* с. Среднее время ожидания для этого случая мы предлагаем вам подсчитать самостоятельно в качестве упражнения.

Оба рассмотренных примера являются скорее теоретическими, чем практическими. На практике типично появление пакетов в случайные моменты времени. Другими словами, момент появления нового пакета, а следовательно, и время между появлениями соседних пакетов, нельзя предсказать заранее. В этом случае значение интенсивности трафика не дает точного ответа на вопрос о величине задержки ожидания. Тем не менее интенсивность способна наглядно проиллюстрировать зависимость задержки ожидания от соотношения скоростей приема и передачи пакетов. Если интенсивность трафика близка к нулю, то, очевидно, вероятность непустого буфера в момент получения нового пакета мала, и поэтому среднее время ожидания также близко к нулю. Если же интенсивность имеет значение, близкое к единице, то возможны кратковременные превышения скорости приема над скоростью передачи и, как следствие, появление очередей и увеличение их длины. Чем ближе к единице значение интенсивности трафика, тем выше вероятность увеличения очереди и времени ожидания. На рис. 1.19 зависимость между интенсивностью и временем ожидания представлена графически.

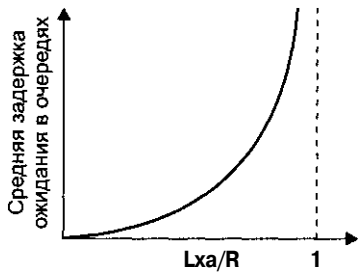


Рис. 1.19. Зависимость средней задержки ожидания от интенсивности трафика

Как видно из рисунка, в области интенсивностей, близких к 1, даже небольшой прирост интенсивности влечет за собой значительно больший прирост задержки ожидания. Этот математический эффект легко проиллюстрировать с помощью примера. Когда вы движетесь по трассе, которая до отказа заполнена машинами, даже самое незначительное снижение пропускной способности трассы (то есть увеличение интенсивности) влечет за собой длительные автомобильные «пробки».

Потеря пакетов

В рассмотренных выше примерах мы сделали допущение о том, что наш маршрутизатор способен хранить в буфере бесконечное число пакетов. Разумеется, на практике объем буферов не только конечен, но и весьма ограничен, поскольку придание маршрутизаторам способности хранить большое количество пакетов значительно повышает их стоимость. Это, в свою очередь, означает, что на практике задержка ожидания также не может быть бесконечной. Если буфер окажется заполненным до конца, то для размещения нового пакета не останется свободного места, и этот пакет будет *потерян.* С точки зрения оконечных систем это приведете тому, что пакет, переданный отправителем, не будет получен адресатом. Очевидно, что чем выше интенсивность трафика, тем выше вероятность потери пакета. Таким образом, передача через узлы сети характеризуется не только длительностями задержек, но и вероятностями потери пакетов. Как мы увидим в следующих главах, потерянный пакет подлежит повторной отправке либо сетевым приложением, либо транспортным уровнем протокола.

Общая задержка

До настоящего момента основным объектом нашего внимания являлась узловая задержка, то есть задержка, обусловленная отдельными маршрутизаторами. Теперь пришло время оценить общую задержку передачи пакета от отправителя до адресата. Для этого предположим, что на пути пакета находятся $N - 1$ маршрутизаторов, нагрузка в сети такова, что очереди отсутствуют или пренебрежимо малы, время обработки каждого маршрутизатора и отправителя равно $d^{обр}$, скорость передачи линий связи составляет R бит/с, а время распространения сигнала по линии равно $d^{пасп}$. Все узловые задержки равны между собой, а их сумма дает общую задержку

$$d_{общ} = N \times (d_{обр} + d_{пер} + d_{расчк}).$$

Мы предоставляем вам возможность самостоятельно обобщить приведенную формулу на случаи неравных узловых задержек и наличия задержек ожидания.

Задержки и маршруты в Интернете

Для того чтобы лучше понять, что представляет собой задержка в компьютерной сети, мы рекомендуем вам воспользоваться диагностической программой Traceroute. Эта программа проста и может быть использована практически в любой оконечной системе. Пользователь вводит имя хоста назначения, после чего программа

осуществляет отсылку нескольких специальных пакетов на адрес этого хоста. В процессе передачи по сети пакеты вызывают генерацию сообщений на адрес отправителя каждым встречным маршрутизатором. Каждое такое сообщение содержит адрес и имя соответствующего маршрутизатора.

Рассмотрим принцип работы программы более детально. Пусть на пути следования пакетов от отправителя до адресата находятся iV маршрутизаторов, тогда генерируемая программой серия будет состоять из N пакетов. Каждому из пакетов присваивается номер от 1 до *N*. Когда *n-й* маршрутизатор принимает *n-й* пакет, этот пакет уничтожается, а в адрес отправителя генерируется сообщение. *N-й* пакет принимается хостом назначения, уничтожается им, и последнее, *N-е* сообщение отсылается отправителю. Отправитель регистрирует время посылки каждого пакета и получения соответствующего ответного сообщения, а также адрес и имя маршрутизатора, уничтожившего пакет. Таким образом, отправитель может получить информацию о пути следования пакета в сети, а также обо всех узловых задержках. Программа повторяет описанный опыт трижды, посылая в общей сложности 3 x Nпакетов адресату. Детальное описание программы Traceroute можно найти в документе RFC 1393.

Ниже приведен результат работы программы Traceroute при пересылке пакетов от хоста eniac.seas.upenn.edu хосту diane.ibp.fr. Этот результат состоит из шести полей, первое из которых представляет собой номер пакета *n,* то есть порядковый номер маршрутизатора на маршруте, второе — имя маршрутизатора, третье — адрес маршрутизатора в форме xxx.xxx.xxx.xxx, а последние три поля — задержки для всех трех испытаний. Если в результате отправитель получил менее трех ответных сообщений (то есть при передаче наблюдались потери), рядом с номером маршрутизатора появляется знак звездочки (*).

```
 1 GW.CIS.UPENN.EDU (130.91.6.254) 3 ms 2 ms 1 ms
 2 DEFAULT7-GW.UPENN.EDU (165.123.247.8) 3 ms 1 ms 2 ms
 3 192.204.183.1 (192.204.183.1) 3 ms 4 ms 3 ms
 4 border2-hssil0.WestOrange.mci.net (204.70.66.5) 6 ms 6 ms 6 ms
 5 corel-fddi-l.WestOrange.mci.net (204.70.64.33) 7 ms 6 ms 6 ms
 6 somerouter.sprintlink.net (206.157.77.106) 16 ms 305 ms 192 ms
 7 somerouter.sprintlink.net (206.157.77.106) 20 ms 196 ms 18 ms
 8 sl-dc-6-H2/0-T3.sprintlink.net (144.228.10.33) 19 ms 18 ms 24 ms
 9 198.67.0.1 (198.67.0.1) 19 ms 24 ms 18 ms
10 gsl-dc-3-FddiO/0.gsl.net (204.59.144.197) 19 ms 18 ms 20 ms
11 * raspail-ip.eurogate.net (194.206.207.6) 133 ms 94 ms
12 raspail-ip2.eurogate.net (194.106.207.57) 93 ms 95 ms 97 ms
13 194.206.207.17 (194.206.207.17) 200 ms 94 ms 209 ms
14 stamandl.renater.ft.net (192.93.43.185) 105 ms 101 ms 105 ms
15 stlambert.rerif.ft.net (192.93.43.117) 108 ms 102 ms 95 ms
16 dantonl.rerif.ft.net (193.48.53.50) 110 ms 97 ms 91 ms
17 u-jussieu-paris.rerif.ft.net (193.48.58.122) 94 ms 96 ms 100 ms
18 r-jusren.reseau.jussieu.fr (192.44.54.126) 100 ms 94 ms 100 ms
19 r-ibp.reseau.jussieu.fr (134.157.254.250) 96 ms 100 ms 94 ms
20 masi.ibp.fr (132.227.60.23) 121 ms 100 ms 97 ms
21 * diane.ibp.fr (132.227.64.48) 105 ms 102 ms
```

Как видно из результата, путь между хостами включает 20 маршрутизаторов. Все маршрутизаторы имеют адрес, и почти все имеют имя. Так, например, марш-

рутизатор с номером 8 имеет имя sl-dc-6-H2/0-T3.sprintlink.net и адрес 144.228.10.33. Для этого маршрутизатора задержки в каждой из попыток составили 19, 18 и 24 мс соответственно. Полученные значения включают в себя все составляющие, перечисленные выше: задержки передачи, распространения, обработки и ожидания. Поскольку задержка ожидания является меняющейся во времени величиной, может оказаться, что для пакета *n,* достигнувшего маршрутизатора *n,* суммарная задержка превышает задержку, полученную для пакета *n* + 1, достигнувшего маршрутизатора *n* + 1. Обратите внимание на скачок значений задержки, наблюдающийся при переходе от маршрутизатора 10 к маршрутизатору И. Это объясняется тем, что между ними находится трансатлантическая линия связи.

Если вы желаете самостоятельно поэкспериментировать с программой Traceroute, посетите сайт http://www.traceroute.org, на котором имеется широкий выбор адресов хостов-отправителей. Вам необходимо выбрать один хост из представленного списка, а затем указать любое имя хоста, который будет использоваться в качестве адресата. Все остальные действия программа выполнит автоматически.

Уровни протоколов и модели их обслуживания

Как вы, вероятно, убедились в процессе нашего обсуждения, Интернет является чрезвычайно сложной системой. Глобальная Сеть состоит из множества самых разных компонентов: приложений, протоколов, оконечных систем, технологий их доступа к сети, маршрутизаторов, линий связи и т. д. Неудивительно, если у вас уже успел возникнуть вопрос о том, поддается ли архитектура Интернета какой-либо организации. К счастью, мы можем вселить в вас долю оптимизма, ответив положительно.

Многоуровневая структура

Перед тем как приступить к систематизации приобретенных знаний об Интернете, давайте поищем аналогию у людей. Каждый день многие из нас сталкиваются со сложными системами. Представьте себе ситуацию, когда кто-нибудь просит вас описать организацию воздушных сообщений. Как вы станете описывать эту огромную структуру, включающую в себя отделы продажи билетов, проверки багажа, обслуживающий персонал, пилотов, летную технику, диспетчерские службы и т. д.? Один из способов описать организацию — это перечислить те действия, которые вы (или сотрудники организации) совершаете при ее использовании. К примеру, вы заказываете билет, проходите багажный контроль, регистрируетесь и попадаете на борт самолета. Затем вы совершаете перелет, достигаете пункта назначения, снова регистрируетесь, получаете багаж и, если рейс был некомфортным, подаете жалобу в отдел продажи билетов. Последовательность ваших действий графически иллюстрирует рис. 1.20.

Здесь мы без труда можем увидеть аналогию с принципами функционирования компьютерных сетей. Вы путешествуете на самолете от пункта отправления до

пункта назначения, а пакет передается от хоста-отправителя к хосту-адресату. Однако более глубокая аналогия заключена в *структурах* действий. Взглянем на рис. 1.20 чуть внимательнее. Как легко видеть, оба ваших конечных действия обращены к отделу продажи билетов, второе и предпоследнее действия связаны с багажом и т. д. Структура действий является симметричной, где «осью симметрии» служит перелет. Таким образом, процесс путешествия на самолете можно представить в виде совокупности горизонтальных уровней, как показано на рис. 1.21.

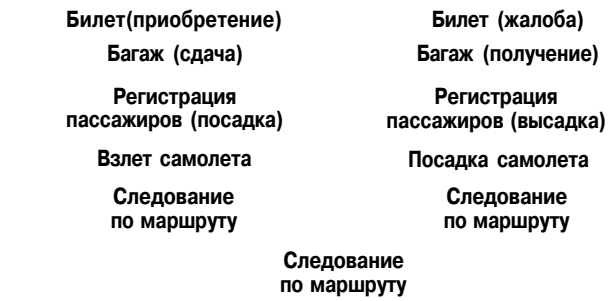


Рис. 1.20. Последовательность действий пассажира при совершении перелета

Рис. 1.21. Горизонтальная многоуровневая структура процесса перелета

Приведенная горизонтальная структура является главным предметом нашего дальнейшего рассмотрения. Нам удалось определить место каждого компонента путешествия относительно других компонентов. Например, говоря о регистрации пассажиров, мы знаем, что эта процедура происходит после проверки багажа перед посадкой на борт самолета. Заметим, что каждый выделенный уровень обладает собственной функциональностью, то есть службами, предоставляющими услуги пассажирам. Ниже билетного уровня пассажир проходит через все этапы обслуживания от получения билета до приема жалобы, ниже багажного уровня — все

этапы от проверки и сдачи багажа до его получения, и т. д. При этом на багажном уровне обслуживаются лишь те пассажиры, которые прошли обслуживание на билетном уровне. То же самое справедливо и в отношении остальных уровней. Таким образом, обслуживание на каждом из уровней производится путем выполнения функций, относящихся к этому уровню, с использованием результатов обслуживания на всех предыдущих уровнях.

Итак, многоуровневая структура позволяет детально оценивать элементы большой и сложной системы, что уже является ее значительным достоинством. Кроме того, с использованием многоуровневой структуры легче модифицировать функции системы — для этого лишь нужно внести изменения в соответствующий уровень, при этом структурно-функциональная организация системы останется прежней. Так, например, усовершенствование системы регистрации будет сведено к внутренним изменениям регистрационного уровня, что никак не отразится на его функциях и не изменит структуру в целом.

Теперь давайте вернемся к компьютерным сетям и рассмотрим организацию сетевых протоколов. Как некоторые, возможно, уже догадались, протоколы, а следовательно, и все сетевое программное и аппаратное обеспечение организованы в виде *уровней.* Каждый протокол относится к определенному уровню сетевой коммуникационной модели. Обратите внимание на то, что протоколы *распределены* между сетевыми компонентами, включающими оконечные системы и маршрутизаторы, подобно тому как функции обслуживания пассажиров распределены между аэропортами. Таким образом, каждое сетевое устройство взаимодействует с несколькими уровнями сети. Взаимодействие между компонентами на уровне *n* осуществляется с помощью специальных сообщений уровня *n,* которые называются единицами обмена (Protocol Data Unit, PDU) данного уровня. Содержание и формат подобных сообщений, а также правила их передачи определяются протоколом уровня *n.* Совокупность протоколов всех уровней коммуникационной модели называется *стеком протоколов.*

Когда уровень *n* хоста A посылает сообщение уровню *n* хоста B, внутри хоста A происходит передача сообщения уровню *n* - 1, который осуществляет обмен с уровнем *n* - 1 хоста B. Таким образом, обмен внутри сетевого уровня *n* всегда обеспечивается уровнем *n* - 1. Ключевым понятием для нас будет являться *модель обслуживания* уровня. Говорят, что уровень *n* - 1 предоставляет *услуги* уровню *n.* Например, в перечень услуг могут входить гарантированная доставка сообщений уровню *n* в течение одной секунды, безошибочная доставка сообщения и т. п.

Протоколы уровней

Концепция протоколов уровней коммуникационной модели, на первый взгляд, может показаться сложной и не вполне понятной. Смеем вас обнадежить, что картина быстро прояснится, когда мы начнем детально изучать коммуникационную модель Интернета, а пока попробуем разобраться в механизме многоуровневого обмена сообщениями между двумя хостами. Предположим, что некоторая компьютерная сеть использует четырехуровневую коммуникационную модель

(эта сеть отличается от Интернета, где коммуникационная модель состоит из пяти уровней), как показано на рис. 1.22. Таким образом, в сети существует четыре вида единиц обмена (PDU), соответствующих каждому уровню. Пусть приложение хоста А создает сообщение М, предназначенное приложению хоста В. Поскольку приложения находятся на вершине многоуровневой структуры, сообщение М относится к PDU уровня 4. Сообщение может включать в себя множество полей, подобно структуре или записи; содержимое полей интерпретируется приложениями. Обычно в PDU включается имя отправителя, тип сообщения, а также другая информация.

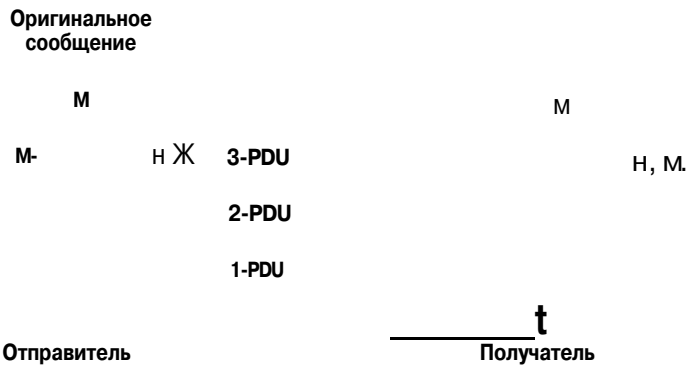


Рис. 1.22. Виды единиц обмена разных уровней в архитектуре протоколов

Сначала сообщение М передается с четвертого уровня на третий, где происходит расщепление исходного сообщения на две единицы обмена уровня 3, M^1 и M^2. При этом к сообщениям присоединяются так называемые *заголовки* третьего уровня H^3, содержащие дополнительную информацию; необходимую для обслуживания сообщения М. Аналогичным образом полученные единицы обмена «спускаются» вниз по стеку протоколов с присоединением новых заголовков на каждом уровне. Единицы обмена уровня 1 передаются по линии связи хосту В, где снова поступают в стек протоколов. Далее происходит передача полученных единиц обмена «вверх» с отсоединением заголовков, и затем единицы обмена M^1 и M^2 объединяются в исходное сообщение М, которое принимается приложением.

Приведенный пример наглядно демонстрирует концепцию службы в коммуникационной модели: каждый уровень выполняет собственные обслуживающие функции, используя результаты обслуживания более высоких уровней.

Интересно заметить, что описанным выше образом обслуживание производится не только в компьютерных сетях. Рассмотрим, например, работу почтовых служб. Когда вы намереваетесь отправить письмо, на конверте вы указываете служебную информацию, включающую адреса получателя и отправителя. В данном случае вы являетесь наивысшим уровнем почтовой модели. Опуская письмо в почтовый ящик, вы посылаете его на более низкий уровень — уровень почтового отделения. Там на конверте будет поставлен штамп отделения и, возможно, добавлена какая-либо иная информация. Эти действия аналогичны присоединению заголовка уровня.

Опуская конверт в почтовый ящик, вы задействуете обслуживающие функции (услуги) почтовой службы, которая обязана обеспечить доставку письма адресату за определенное время. Вас не интересуют технические аспекты процесса доставки, которые целиком находятся в компетенции почтовой службы. Кроме того, почтовая служба сама является многоуровневой по структуре, и каждый уровень использует обслуживающие функции уровней, «младших» по иерархии.

Для успешного взаимодействия соседних уровней необходимо четкое определение интерфейса между ними. Обычно такое взаимодействие описывается документированными стандартами, рекомендуемыми к применению разработчиками аппаратного и программного сетевого обеспечения. Это несколько ограничивает «свободу творчества» разработчиков, однако позволяет защитить структуру коммуникационной модели.

Функции уровней

В компьютерной сети каждый уровень может выполнять одну или несколько функций, перечисленных ниже.

- Контроль ошибок обеспечивает повышение надежности логического канала между смежными уровнями сетевой модели.
- Контроль потока позволяет избегать переполнения единицами обмена более медленного хоста.
- Разбиение и сборка пакетов предназначены для изменения размеров единиц обмена на разных уровнях.
- Мультиплексирование позволяет нескольким подключениям совместно использовать одно подключение более низкого уровня.
- Установка соединения есть выполнение процедуры рукопожатия между участниками обмена.

Снабжение каждого уровня коммуникационной модели собственным протоколом наряду с достоинствами имеет и несколько недостатков. Первый связан с нередко встречающейся ситуацией дублирования одних и тех же функций различными уровнями (например, контроль ошибок). Другой потенциальный недостаток заключается в том, что функциям одного уровня может понадобиться информация, хранящаяся на другом уровне (например, время). Это нарушает принцип изолированности уровней многоуровневой структуры.

Стек протоколов Интернета

Коммуникационная модель Интернета состоит из пяти уровней: физического, канального, сетевого, транспортного и прикладного. Вместо терминов «единица обмена сетевого уровня», «единица обмена канального уровня» и т. д. мы будем использовать специальные имена. Единицы обмена канального уровня мы назовем *кадрами,* единицы обмена сетевого уровня — *дейтаграммами,* единицы обмена транспортного уровня — *сегментами,* а единицы обмена прикладного уровня — *сообщениями*. Для единиц обмена физического уровня обычно не предусматривается специального имени, и мы будем придерживаться этой традиции в нашей

книге. Коммуникационная модель Интернета и единицы обмена ее уровней изображены на рис. 1.23.

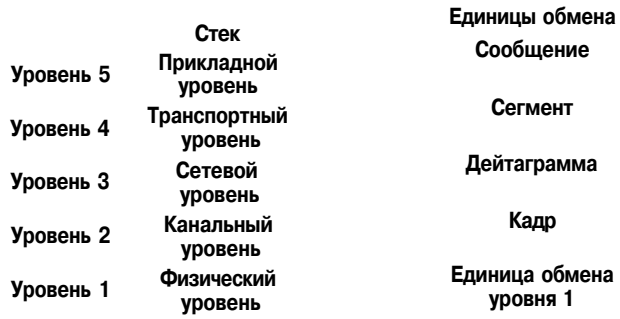


Рис. 1.23. Стек протоколов Интернета и единицы обмена различных уровней

Поддержка протоколов может быть аппаратной, программной или смешанной. Протоколы прикладного уровня, такие как HTTP и SMTP, а также протоколы транспортного уровня практически всегда поддерживаются программно. Напротив, протоколы физического и канального уровней, тесно связанные со средой передачи данных, поддерживаются аппаратно сетевой интерфейсной картой. Сетевой уровень, находящийся в центре коммуникационной модели, может поддерживаться как аппаратно, так и программно. Далее даны характеристики каждого из пяти уровней коммуникационной модели Интернета.

Прикладной уровень

Прикладной уровень, как следует из его названия, предназначен для поддержки сетевых приложений. Имеется множество протоколов прикладного уровня, из которых наиболее важными являются HTTP (для путешествий по web-страницам), SMTP (для электронной почты) и FTP (для обмена файлами). Как мы увидим в главе 2, разработка собственного протокола прикладного уровня не представляет особого труда.

Транспортный уровень

Главная функция транспортного уровня заключается в передаче сообщений прикладного уровня между клиентом и сервером. В Интернете существуют два транспортных протокола: TCP и UDP. Протокол TCP обеспечивает передачу с установлением логического соединения, то есть надежную передачу с контролем переполнения. Кроме того, TCP производит разбиение длинных сообщений на более короткие и контролирует перегрузку. Контроль перегрузки сводится к принудительному снижению скорости передачи оконечной системы при высокой загрузке сети. Протокол UDP обеспечивает передачу сообщений без установления логического соединения, то есть ненадежный вид связи, где допускаются искажения и потери данных.

Сетевой уровень

Сетевой уровень обеспечивает передачу дейтаграмм между двумя хостами и базируется на двух основных протоколах. Первый протокол определяет поля дейтаграммы

и интерпретацию их содержимого маршрутизаторами и оконечными системами. Этот протокол является единственным протоколом сетевого уровня в Интернете и имеет название IP. Вторым протоколом является один из многочисленных протоколов маршрутизации, предназначенных для определения путей дейтаграмм от отправителя до адресата. Число протоколов маршрутизации огромно. Как мы неоднократно говорили, Интернет представляет собой сеть сетей, а каждая сеть поддерживает собственный протокол маршрутизации, обычно определяемый администратором сети. Несмотря на функциональные различия между протоколом IP и протоколами маршрутизации, а также на широкое разнообразие последних, их обычно объединяют под общим именем IP, подчеркивая этим их связующую роль в организации глобальной Сети.

Протокол транспортного уровня (TCP или UDP) передает сегмент и адрес назначения протоколу IP сетевого уровня подобно тому, как вы опускаете письмо в почтовый ящик, а протокол IP сетевого уровня доставляет сегмент конечному хосту и передает его обратно транспортному уровню.

Канальный уровень

Сетевой уровень обеспечивает передачу пакета через серию маршрутизаторов между оконечными системами. Для перемещения пакета (дейтаграммы) от одного узла к другому сетевой уровень прибегает к службам канального уровня. Таким образом, основная функция канального уровня заключается в передаче дейтаграмм между узлами на маршруте.

Канальный уровень использует специальный протокол, ориентированный на используемую линию связи. Иногда протоколы канального уровни обеспечивают надежную передачу между узлами. Обратите внимание на различие надежной передачи на транспортном и канальном уровнях: протокол TCP обеспечивает надежность на всем пути следования сообщения, а протокол канального уровня — лишь между парой узлов. К протоколам канального уровня относятся Ethernet и PPP; иногда аналогичные функции несут технологии асинхронной передачи данных (ATM) и ретрансляции кадров. Поскольку путь от отправителя до адресата обычно состоит из цепочки разнородных линий связи, передача дейтаграммы может осуществляться различными канальными протоколами.

Физический уровень

Если назначением канального уровня является передача кадров между соседними узлами сети, то физический уровень обеспечивает передачу между узлами отдельных битов информации. Протоколы физического уровня также напрямую зависят от использующейся линии связи (медной витой пары, одномодового оптоволокна и т. п.). Технология Ethernet поддерживает множество протоколов физического уровня, предназначенных для поддержки витой пары, коаксиального кабеля, оптоволоконного кабеля и некоторых других видов линий. В каждой из линий связи механизмы передачи бита различны.

Просмотрев оглавление этой книги, вы можете увидеть, что последовательность изложения материала соответствует направлению от прикладного уровня к физическому. Таким образом, мы реализуем заявленный нами ранее подход «сверху вниз».

Сетевые устройства и уровни коммуникационной модели

Наиболее важными сетевыми устройствами являются оконечные системы и коммутаторы. Как мы увидим далее, существуют два типа коммутаторов — мосты и маршрутизаторы. Понятие маршрутизатора вам знакомо и употреблялось уже неоднократно. Мосты будут подробно рассмотрены в главе 5, а детальное обсуждение работы маршрутизаторов проводится в главе 4. Как и оконечные системы, мосты и маршрутизаторы поддерживают многоуровневую структуру сети, однако они обслуживают лишь нижние уровни. Как можно видеть на рис. 1.24, мосты обслуживают только физический и канальный уровни, а маршрутизаторы — физический, канальный и сетевой уровни. Это объясняется тем, что маршрутизаторы способны поддерживать протокол IP, в то время как мосты не обладают такой возможностью. Вы сможете убедиться на практике в том, что мосты распознают не IP-адреса, а лишь адреса канального уровня (например, адреса Ethernet-сети). Хосты обслуживают все пять сетевых уровней; это говорит о том, что архитектура Интернета «сваливает» большую часть своей сложности «на плечи» оконечных систем.

Рис. 1.24. Хосты, мосты, маршрутизаторы и поддерживаемые ими уровни коммуникационной модели

История компьютерных сетей и Интернета

Материал всех предыдущих разделов был посвящен основам сетевых технологий и Интернета. Его вполне достаточно, чтобы получить общее представление о том, что такое компьютерные сети, для чего они нужны, как они устроены и каковы основные принципы их функционирования. В завершение мы хотели бы рассказать об этапах истории развития компьютерных сетей и Интернета [455].

Развитие коммутации пакетов: 1961-1972

История компьютерных сетей берет свое начало в 1960-х годах, когда телефонные сети были основным средством связи в мире. Как вы помните, в телефонных сетях используется принцип коммутации каналов, при этом передача осуществляется с постоянной частотой. Стремительный рост потребности в вычислительных ресурсах, сочетающийся с высокой стоимостью ЭВМ, стал побудительной причиной объединения компьютеров в сети для обеспечения к ним удаленного совместного доступа пользователей. Сетевой трафик был неравномерным и характеризовался наличием периодов активности (например, когда один пользователь посылал команду удаленному компьютеру) и пассивности (ожидание результатов).

Три группы инженеров, находившиеся в разных частях света, независимо друг от друга начали разработку технологии коммутации пакетов, рассматривая ее как мощную и эффективную альтернативу технологии коммутации каналов. Первая научная работа на эту тему была опубликована ученым Леонардом Клейнроком [272,274], в то время еще студентом-старшекурсником. С помощью теории очередей работа Клейнрока наглядно продемонстрировала эффективность принципа коммутации пакетов в условиях неравномерной нагрузки. В 1964 году Пол Верен [23] начал эксперименты в области коммутации пакетов в защищенных военных сетях, а Дональд Дэвис и Роджер Скэнтлбери осваивали новую технологию в национальной физической лаборатории Англии.

Упомянутые разработки заложили основу современного Интернета. Однако было бы неверно сводить зарождение Интернета только к разработке технологии коммутации пакетов. В начале 1960-х годов коллеги Клейнрока, ученые Ликлайдер [124] и Роберте, стали участниками программы развития компьютерных технологий в агентстве DAPRA (Defense Advanced Research Projects Agency — агентство по защите прогрессивных исследовательских проектов). Роберте разработал схему компьютерной сети APRAnet [422], основанной на коммутации пакетов и являющейся прямым предком современного Интернета. Коммутаторы пакетов в то время назывались интерфейсными процессорами сообщений (Interface Message Processor, IMP), и контракт на их производство был заключен с компанией BBN. В 1969 году первые коммутаторы пакетов связали между собой несколько научных организаций США (рис. 1.25). Леонард Клейнрок вспоминает о первом неудачном использовании сети, когда попытка удаленного доступа вызвала полный крах системы [273].

В 1972 году в сети APRAnet насчитывалось уже 15 узлов. Тогда же Роберт Канн устроил первую публичную демонстрацию APRAnet на Международной конфе-

ренции по компьютерным коммуникациям. Был разработан первый протокол обмена информацией между оконечными системами (RFC 001), получивший название NCP (Network-Control Protocol — протокол управления сетью). Это позволило начать разработку сетевых приложений для APRAnet. Первая программа для работы с электронной почтой была создана программистом компании BBN Рэем Томлинсоном в 1972 году.

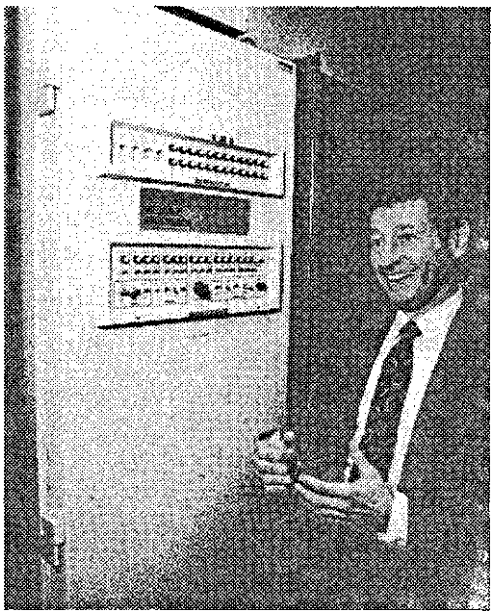


Рис. 1.25. Первый интерфейсный процессор сообщений (рядом стоит Л. Клейнрок)

Возникновение новых компьютерных сетей и Интернета: 1972-1980

Изначально APRAnet была изолированной закрытой сетью. Для установления связи с любым хостом сети было необходимо иметь подключение к интерфейсному процессору сообщений. В первой половине 1970-х годов появились еще несколько компьютерных сетей с коммутацией пакетов:

- коротковолновая сеть ALOHAnet соединила несколько университетов на Гавайских островах [5];
- коммерческая сеть Telenet компании BBN была построена по тому же принципу, что и APRAnet;
- французская компьютерная сеть Cyclades была разработана Луизом Пузи [499];
- сети с временным мультиплексированием, такие как Tymnet и GE Information Services, появились в конце 1960-х — начале!970-х годов [453];
- сеть SNA фирмы IBM создавалась в 1969-1974 годах параллельно с APRAnet [453].

Число компьютерных сетей продолжало расти. В 1973 году Роберт Меткалф разработал принципы технологии Ethernet, ориентированной на небольшие расстояния между соединяемыми компьютерами, которые позже обусловили стремительное развитие локальных сетей.

Научившись создавать новые компьютерные сети, инженеры задумались над тем, как связать несколько сетей между собой. Первые разработки в области создания сети сетей были проведены Уинтоном Серфом и Робертом Канном [60]. Именно тогда для описания создаваемой системы было применено слово «Интернет». Появилась первая версия протокола TCP, правда, в значительной степени отличающаяся от современной. Изначально в TCP была попытка объединить надежную последовательную передачу данных между оконечными системами (поддерживаемую протоколом и сегодня) и транспортные функции (обеспечиваемые современным протоколом IP). Уже первые эксперименты с протоколом TCP выявили важность не только надежной, но и ненадежной передачи данных (например, пакетной передачи голосовых сообщений). Это в конечном счете привело к появлению протокола IP и разработке протокола UDP, альтернативного TCP. Таким образом, три ключевых протокола Интернета, TCP, UDP и IP, появились уже в конце 1970-х годов.

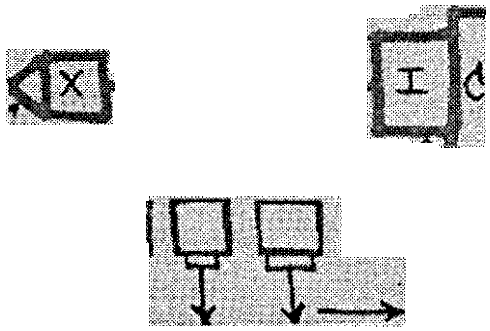


Рис. 1.26. Первоначальная концепция Интернета, намеченная рукой Меткалфа

Описанные выше разработки проводились под патронажем уже упоминавшегося агентства DAPRA. Тем не менее оно не являлось монополистом в области развития Интернет-технологий. На Гавайских островах ученым Норманном Абрамсоном был разработан проект сети ALOHAnet — беспроводной компьютерной сети с пакетной передачей данных. Протокол ALOHA [6], использовавшийся в ALOHAnet, был первым из так называемых протоколов множественного доступа, позволявшим географически распределенным пользователям совместно использовать ресурс среды передачи данных (частоту радиоволн). Разработки Абрамсона были

использованы Меткалфом и Боггсом при создании протокола Ethernet [329] для проводных широкополосных радиосетей. Схема протокола Ethernet приведена на рис. 1.26. Интересно отметить, что к созданию Ethernet Меткалфа и Боггса подтолкнула необходимость обеспечения связи компьютеров не только друг с другом, но и с удаленными разделяемыми периферийными устройствами, такими как принтеры, накопители и т. п. [375]. Таким образом, технология Ethernet, ставшая основой для множества современных локальных компьютерных сетей, насчитывает 25-летнюю историю. Трудно переоценить ее роль в решении задачи объединения компьютерных сетей. Мы еще вернемся к обсуждению Ethernet, ALOHAnet и прочих технологий локальных сетей в главе 5.

Распространение компьютерных сетей: 1980-1990

К концу 1970-х годов сеть APRAnet насчитывала уже около 200 оконечных систем. Через 10 лет число хостов в Интернете, уже объединявшем множество других компьютерных сетей, достигло 100 тысяч. Таким образом, 1980-е годы характеризуются стремительным распространением созданных ранее сетевых технологий.

В начале 80-х происходило активное объединение локальных сетей университетов в крупные региональные сети. Примерами могут служить сеть BITNET, обеспечивавшая обмен файлами и электронной почтой между университетами на северо-западе США, CSNET, объединившая исследователей в области сетевых технологий независимо от APRAnet, и др. В 1986 году была разработана сеть NSFNET, позволившая получить доступ к вычислительным ресурсам суперкомпьютеров. Начальная скорость магистрали, составившая 56 Кбит/с, к концу десятилетия выросла до 1,5 Мбит/с. Магистраль NSFNET позволила объединить между собой региональные компьютерные сети США.

В 1980-е годы APRAnet уже содержала многие из компонентов, которые составляют основу современного Интернета. 1 января 1983 года стандартный протокол NCP, предназначенный для обмена данными между хостами, был заменен стеком протоколов TCP/IP (RFC 801). С этого времени стек TCP/IP используется всеми хостами Интернета. В конце 80-х в протокол TCP были внесены значительные усовершенствования, направленные на обеспечение оконечными системами контроля переполнения. Кроме того, была разработана система доменных имен (Domain Name System, DNS), связавшая мнемонические имена Интернет-ресурсов с их 32-разрядными адресами (RFC 1034).

Параллельно с развитием APRAnet в США во Франции в начале 1980-х годов возник проект Minitel, имевший поддержку со стороны правительства Франции и поставивший перед собой амбициозную цель — связать все сети в единую компьютерную сеть. Система, разработанная Minitel, представляла собой открытую компьютерную сеть с коммутацией пакетов (протокол X.25 с поддержкой виртуального канала), состоявшую из Minitel-серверов и недорогих пользовательских терминалов со встроенными низкоскоростными модемами. Большой успех пришел к проекту Minitel после того, как французское правительство объявило о раздаче бесплатных терминалов всем желающим для домашнего пользования. Сеть Minitel содержала как бесплатные, так и платные информационные ресурсы. В зе-

ните своей популярности в середине прошлого десятилетия Minitel поддерживала более чем 20 000 видов обслуживания — от удаленных банковских операций до организации доступа к специализированным исследовательским базам данных. Пользователями сети являлись более 20 % жителей Франции, доход от ее использования составлял более миллиарда долларов в год, а обслуживающий персонал состоял из 10 000 человек. Таким образом, Франции удалось опередить США в развитии национальных сетевых технологий на целое десятилетие.

Распространение Интернета: 1990-е

В начале 1990-х годов произошел ряд событий, предвосхитивших Интернет-революцию и коммерциализацию компьютерных сетей. Сеть APRAnet, предок Интернета, постепенно прекратила свое существование. Появившиеся в 1980-е годы сети MILNET, Defense Data Network и NSFNET стали играть ведущую роль в объединении локальных сетей США, а также в международной передаче данных. В 1991 году на коммерческое использование NSFNET были наложены ограничения, а в 1995 году сеть также фактически прекратила свое существование, передав свои функции сетям коммерческих Интернет-провайдеров.

Главным событием 90-х годов, вероятно, следует считать появление Всемирной паутины (web), приведшей Интернет в миллионы домов и организаций по всему миру. Служба web также послужила платформой для разработки и внедрения сотен новых Интернет-приложений, обеспечивающих удаленные биржевые и банковские операции, работу с потоковым мультимедиа и использование огромных информационных ресурсов [540].

Автором web считается Тим Бернерс-Ли, который в 1989-1991 годах [22] развил идеи гипертекста, предложенные еще в 40-х и 60-х годах прошлого века Бушем [49] и Нельсоном [568]. Бернерс-Ли совместно со своими ассистентами создал первоначальные версии языка HTML, протокола HTTP, web-сервера и браузера. Таким образом, четыре «кита» Всемирной паутины фактически были придуманы одним человеком. Возможности первого браузера ограничивались лишь просмотром текстовых строк. К концу 1992 года количество web-серверов в мире достигло 200. Параллельно с внедрением новых web-серверов разработчики трудились над созданием пользовательского интерфейса браузеров. Одним из наиболее видных инженеров, проявивших себя в этой области, был Марк Андресен, руководивший созданием популярного браузера Mosaic. Альфа-версия браузера появилась в 1993 г., а в 1994 году Андресен совместно со своими коллегами учредил компанию Mosaic Communications, позже трансформировавшуюся в корпорацию Netscape Communications [105,402]. В 1995 году студенты университетов уже активно использовали браузер Mosaic в учебных целях. Примерно в то же время огромное количество самых разных компаний стали применять web для ведения своих дел. В 1996 г. к выпуску браузеров присоединилась компания Microsoft, что положило начало «войне браузеров» между Microsoft и Netscape. На сегодняшний день можно считать, что Microsoft выигрывает эту войну [105].

Вторая половина 1990-х годов характеризовалась небывалым прогрессом в области Интернет-технологий. Множество компаний начали разработку собственных

продуктов, связанных с глобальной Сетью. Активно развивалась служба электронной почты: появились приложения, поддерживающие адресные книги, присоединение файлов к сообщениям, «горячие» ссылки и потоковое мультимедиа. К концу десятилетия были разработаны сотни Интернет-приложений, как правило, принадлежащие к одной из следующих групп:

- приложения электронной почты, включая средства присоединения файлов к сообщениям и приложения для работы с сообщениями через web-интерфейс;
- web-приложения, включая приложения для путешествий по web-страницам и Интеренет-коммерции;
- приложения для обмена сообщениями в реальном времени, пионером которых стала программа ICQ;
- приложения для однорангового совместного доступа к файловым архивам в формате MP3, пионером которых стала ^программа Napster.

Создателями первых двух видов приложений были ученые, а последние два вида приложений, напротив, разрабатывались несколькими молодыми энтузиастами.

Период 1995-2001 годов характеризовался активным использованием Интернета для биржевых торгов. Многие компании, практически не имевшие реальных доходов, могли иметь огромный финансовый успех на подобных электронных торгах. В 2000-2001 годах многие Интернет-биржи закрылись. Тем не менее такие компании, как Microsoft, Cisco, AOL и Yahoo!, достаточно успешно продолжают вести свой бизнес в Интернете.

На протяжении 1990-х годов удалось достичь значительных успехов в области высокоскоростной маршрутизации и локальных сетей (см. главы 4 и 5 соответственно). Была разработана модель обслуживания трафика, требующего временных ограничений (например, мультимедийных приложений, описанных в главе 6). Кроме того, коммерциализация глобальной Сети актуализировала проблемы, связанные с обеспечением безопасности инфраструктуры Интернета.

Новейшие разработки

Сетевые технологии продолжают свое стремительное развитие. Постоянно появляются новые решения в разработке приложений, обеспечении безопасности, распределении ресурсов, Интернет-телефонии, высокоскоростной маршрутизации и передаче внутри локальных сетей. Мы бы хотели выделить три направления развития Интернета, которые считаем наиболее важными: широкополосный резидентный доступ, беспроводной доступ и одноранговая передача данных.

Широкополосный резидентный доступ в Интернет с использованием линий DSL и кабельных модемов (см. раздел «Доступ к сети и ее физическая среда») в настоящее время постепенно завоевывает популярность среди домашних пользователей. По прогнозам некоторых специалистов к 2005 году до 50 % резидентных подключений к Интернету будет широкополосным, что позволит разрабатывать и использовать мультимедиа-приложения, обеспечивающие высокое качество воспроизведения в реальном времени. Особую роль это, очевидно, сыграет для видеоконференций.

Беспроводной доступ в Интернет с использованием технологии i-mode чрезвычайно популярен в Японии. Устройство i-mode [4] внешне напоминает обычный мобильный телефон, однако имеет больший экран, предназначенный для вывода текстовой и графической информации. Устройство поддерживает телефонные и Интернет-соединения. Статистические данные показали, что в августе 2001 года в Японии было зарегистрировано более 200 миллионов абонентов i-mode, и их число постоянно увеличивается. Рост популярности беспроводных технологий также наблюдается в США и Европе, стимулируя операторов мобильной связи к обслуживанию низкоскоростных соединений с Интернетом. Кроме того, начало нового десятилетия ознаменовалось дальнейшим ростом беспроводных локальных сетей с предоставлением доступа к ним в кафе, гостиницах, государственных и коммерческих организациях, а также в домах частных пользователей.

Последним новшеством, о котором мы хотим упомянуть, являются одноранговые (P2P) сети. Такие сети обеспечивают каждого пользователя, подключенного к сети, совместным децентрализованным доступом к ресурсам, когда обмен данными между пользователями осуществляется путем их прямого соединения между собой. Приложение Napster стало первым приложением, успешно использовавшим P2P-доступ для обмена файлами в формате MP3. С течением времени P2P-приложения стали применяться не только для разделения MP3-файлов, но и для разделения видеофайлов, изображений, текста. Примером P2P-приложения может также служить любая программа, обеспечивающая обмен сообщениями в реальном времени (ICQ и т. п.), поскольку сообщения посылаются сторонами друг другу напрямую (обычно через TCP-соединения), без центрального сервера. Наконец, проект SETI@home, упоминавшийся в разделе «Периферия компьютерных сетей», также представляет собой реализацию одноранговых вычислений.

Резюме

В этой главе мы изучили массу важного и полезного материала. Мы рассмотрели программные и аппаратные компоненты, из которых составляются как компьютерные сети вообще, так и Интернет в частности. Мы начали с «периферии» компьютерных сетей, коснувшись оконечных систем и приложений, а также транспортных услуг, предоставляемых приложениям, запущенным на оконечных системах. Использовав распределенные сетевые приложения в качестве примера, мы познакомились с понятием протокола — ключевым понятием в области компьютерных сетей. Далее мы продолжили погружаться от периферии компь отерной сети к ее ядру и рассмотрели два фундаментальных подхода к передаче данных в телекоммуникационных сетях — коммутацию каналов и коммутацию пакетов, выделив их достоинства и недостатки. Затем в центре нашего внимания оказался самый нижний с точки зрения коммуникационной модели физический уровень передачи данных: мы рассмотрели основные среды передачи и технологии доступа к глобальной сети. Мы также познакомились со структурой Интернета, представили ее как сеть сетей и узнали о том, что иерархия сетей Интернет-провайдеров позволила глобальной Сети с легкостью включать в себя новые сегменты.

Вторая часть главы была посвящена основным аспектам функционирования компьютерных сетей. Сначала мы рассмотрели основные причины задержек и потерь пакетов в процессе передачи и создали несложные количественные модели задержек обработки, ожидания, передачи и распространения. Последние будут весьма полезны при выполнении заданий для самостоятельной работы. Далее мы ознакомились с концепциями многоуровневой коммуникационной модели и протоколами каждого из уровней, составляющих архитектурную основу компьютерных сетей. Мы завершили разговор историческим мини-экскурсом, проиллюстрировавшим основные этапы развития сетевых технологий.

Итак, в этой главе вы прочли значительный объем разнообразного материала. Если некоторые тезисы показались вам неясными и запутанными, не стоит беспокоиться об этом. Мы вернемся практически ко всем вышеизложенным идеям в следующих главах, чтобы рассмотреть их более детально. Целью этой главы является создание у вас не столько конкретного, сколько интуитивного представления о том, что такое компьютерная сеть и как она функционирует. При необходимости (которая, по нашим предположениям, может возникнуть) вы всегда сможете вернуться к материалу этой главы. Мы искренне надеемся, что ваше первое знакомство с захватывающим миром сетевых технологий прошло успешно и вы по-прежнему полны энтузиазма.

Дальнейшие планы

Отправляясь в путешествие, нам необходимо позаботиться о наличии карты, которая даст нам представление о том, какими путями можно достигнуть конечного пункта. Наше путешествие в мир компьютерных сетей также не является исключением, поэтому ниже мы приводим «карту», которая поможет вам лучше понять, по какому пути мы намерены двигаться далее.

1. Компьютерные сети и Интернет.
2. Прикладной уровень.
3. Транспортный уровень.
4. Сетевой уровень и маршрутизация.
5. Канальный уровень и локальные сети.
6. Мультимедиа в сети.
7. Безопасность в компьютерных сетях.
8. Сетевое администрирование.

Как можно видеть, главы 2-5 составляют основу книги и посвящены детальному рассмотрению четырех верхних уровней коммуникационной модели. Сначала мы рассмотрим самый верхний, прикладной уровень, а затем, как и в этой главе, начнем движение «вниз» по стеку протоколов. Мы аргументируем рациональность нашего подхода тем, что, зная функции того или иного уровня, мы сможем понять, что нужно для обслуживания этого уровня.

После завершения разговора об уровнях коммуникационной модели мы рассмотрим три темы, каждая из которых чрезвычайно важна для понимания современ-

ных компьютерных сетей. Глава 6 посвящена приложениям, предназначенным для потокового аудио и видео, Интернет-телефонии и организации видеоконференций. В главе 7 мы узнаем об основных причинах шифрования данных, о защите компьютерных сетей и о применении соответствующих технологий в Интернете. Последняя глава посвящена ключевым принципам сетевого администрирования, а также описанию предназначенных для этого Интернет-протоколов.

Вопросы и задания для самостоятельной работы

Приведенные ниже задания рекомендуются для самостоятельной проработки. При возникновении трудностей обратитесь к соответствующим разделам этой главы.

1. В чем заключается разница между хостом и оконечной системой? Перечислите известные вам типы оконечных систем. Является ли web-сервер оконечной системой?
2. Слово «протокол» часто употребляется средствами массовой информации в контексте дипломатических отношений. Приведите пример дипломатического протокола.
3. Что такое программа-клиент и программа-сервер? Пользуется ли программа-сервер службами программы-клиента?
4. Назовите и охарактеризуйте два вида услуг, которые Интернет предоставляет приложениям.
5. Говорят, что понятия «контроль потоков данных» и «контроль перегрузки» эквивалентны. Справедливо ли это для служб с установлением логического соединения? Одинаковы ли цели контроля потоков данных и контроля перегрузки?
6. Опишите в общих чертах механизм надежной передачи данных службой с установлением логического соединения.
7. В чем заключается преимущество коммутации каналов перед коммутацией пакетов? Каковы преимущества временного мультиплексирования перед частотным в сетях с коммутацией каналов?
8. Почему мультиплексирование в сетях с коммутацией пакетов называют статистическим? Сравните статистическое мультиплексирование с временным мультиплексированием.
9. Предположим, что на пути пакета находится единственный маршрутизатор. Скорости линий связи, обеспечивающих передачу от хоста-отправителя до маршрутизатора и от маршрутизатора до хоста-адресата, составляют R^{1} и R^{2} соответственно. Учитывая, что маршрутизатор осуществляет предварительное накопление пакетов, рассчитайте суммарную задержку для пакета длины I. Задержки обработки, передачи и распространения считать нулевыми.
10. Что понимается под состоянием соединения в сети с коммутацией каналов? Если установления и разрывы соединений в такой сети в среднем происходят с частотой 1000 раз в секунду, с какой частотой необходимо изменять таблицы маршрутизации коммутаторов?

11. Представьте, что вы разрабатываете стандарт для нового типа компьютерных сетей и вам необходимо решить, что следует использовать: дейтаграммы или виртуальный канал. Каковы «за» и «против» виртуального канала?
12. В чем достоинства и недостатки сегментации сообщений в сетях с коммутацией пакетов?
13. Перечислите шесть технологий доступа. Классифицируйте каждую из них с позиций резидентного, корпоративного или мобильного доступа.
14. Назовите главный признак, отличающий Интернет-провайдеров первого звена от Интернет-провайдеров второго звена.
15. Поясните разницу между точками POP и NAP.
16. Является ли полоса пропускания оптоволоконно-коаксиального кабеля (HFC) выделенным или разделяемым между пользователями ресурсом? Возможны ли столкновения пакетов в принимающем канале HFC? Обоснуйте ответ.
17. Какова скорость передачи в локальных Ethernet-сетях? Могут ли все пользователи локальной сети передавать данные с этой скоростью?
18. Какие физические среды используются в Ethernet-сетях?
19. Коммутируемые модемы, линии DSL и кабели HFC являются средствами резидентного доступа к сети. Для каждой из этих технологий приведите скорость передачи данных и поясните, является ли полоса пропускания линии выделенной или разделяемой.
20. Представим передачу пакетов от одного хоста к другому по некоторому пути. Перечислите все виды задержек, которые могут возникнуть при передаче пакета между хостами. Укажите, какие из этих задержек постоянны, а какие могут меняться.
21. Перечислите пять функций, которые может выполнять уровень коммуникационной модели. Возможно ли выполнение каких-либо из этих функций несколькими уровнями? .
22. Перечислите пять уровней коммуникационной модели, принятой в Интернете. Каковы главные функции каждого из уровней?
23. Протоколы каких уровней обслуживаются маршрутизаторами?

Упражнения

1. Разработайте и опишите протокол прикладного уровня для использования между автоматической кассовой машиной и центральным компьютером банка. Протокол должен предусматривать ввод пользователем пароля банковской карты, определение текущего баланса лицевого счета и возврат запрашиваемой суммы. Кроме того, протоколом должны обрабатываться всевозможные спорные ситуации, когда, например, запрашиваемая сумма превышает текущий баланс и т. п. Описание протокола должно содержать все используемые сообщения и наборы действий, предпринимаемых при приеме или получении сообщений. Ситуацию, когда взаимодействие между кассовой машиной и центральным компьютером проходит успешно, представьте в виде диаграммы (см. рис. 1.2). Явно укажите допущения о транспортных услугах, которые используются вашим протоколом.

Представьте себе приложение, осуществляющее пересылку данных с фиксированной скоростью (например, генерирующее Л^-разрядную порцию данных каждые k единиц времени, где k — постоянное малое значение). Пусть, будучи запущенным, это приложение функционирует в течение относительно долгого периода времени. Ответьте на перечисленные ниже вопросы, снабдив ответы краткими пояснениями.

2.1. Какой из видов сетевой организации (коммутация пакетов или коммутация каналов) будет более эффективен для указанного соединения? Почему?

2.2. Предположим, что в сети осуществляется коммутация пакетов, при этом передача данных другими пользователями происходит по тому же механизму, что был описан выше. Также предположим, что суммарный трафик не превышает пропускной способности линий связи. Необходим ли контроль перегрузок в такой сети? Почему?

Рассмотрим сеть с коммутацией каналов, подобную представленной на рис. 1.5. Пусть каждая линия связи поддерживает n каналов связи.

3.1. Каково максимальное число соединений, которое может быть установлено с сетью в каждый фиксированный момент времени?

3.2. Предположим, что все соединения в сети осуществляются между хостами в левом верхнем и правом нижнем углах. Каково максимальное число одновременно устанавливаемых соединений?

Вернемся к аналогии с колонной автомашин, приведенной в разделе «Задержки и потери данных в сетях с коммутацией пакетов». Пусть скорость «распространения» между пунктами для сбора пошлин равна 100 км/ч.

4.1. Предположим, что длина пути колонны составляет 200 км; путь начинается у входа в первый пункт для сбора пошлин и заканчивается у входа в третий пункт. Какова общая задержка в пути?

4.2. Вычислите общую задержку в пути, положив число машин в колонне равным 7, а не 10.

Пусть файл, разбитый на M пакетов длиной I, передается по маршруту, состоящему из Q линий связи. Скорость передачи каждой линии связи составляет R бит/с. Загрузка линий такова, что среднюю задержку ожидания пакетов можно положить равной нулю. Скорость распространения сигнала по линиям связи следует считать мгновенной.

5.1. Пусть в сети используется виртуальный канал. Обозначим через t^s время в секундах, необходимое для установки соединения, а через h — число битов в заголовках, добавляемых различными уровнями коммуникационной модели. Определите время, необходимое для пересылки файла между оконечными системами.

5.2. Пусть в сети используется дейтаграммный механизм передачи и модель без установления логического соединения. Предположив, что объем добавляемых заголовков составляет $2h$, определите время передачи файла между оконечными системами.

5.3. Выполните упражнение 5.2 в предположении, что в сети используется коммутация сообщений. Объем заголовков оставить неизменным.

5.4. Теперь допустим, что в сети используется коммутация каналов, а скорость передачи данных по каналу равна R бит/с. Файл не разбивается на пакеты, однако имеет тот же размер, что и в предыдущих случаях. Пусть t^s — время, необходимое для установки соединения, а h — число битов в заголовках, добавляемых к файлу уровнями коммуникационной модели. Определите время, необходимое для пересылки файла между оконечными системами.

6. Поэкспериментируйте с Java-апплетом, призванным помочь в освоении темы сегментации сообщений и размещенным на нашем web-сайте. Соответствуют ли величины задержек рассчитанным в предыдущем упражнении? Как задержки распространения сигнала влияют на суммарное время передачи при коммутации пакетов и коммутации сообщений?

7. Пусть файл размером *F6m* передается от хоста А хосту В. Между хостами имеются две линии связи, подключенные к маршрутизатору. Перегрузка в обеих линиях отсутствует. Хост А разбивает файл на сегменты размером 5 бит, добавляя 40 бит заголовка к каждому из сегментов. Скорость передач по линиям связи составляет R бит/с. Определите величину 5, при которой суммарное время передачи файла минимально. Задержкой распространения сигнала пренебречь.

8. Пусть хосты А и В соединены линией связи со скоростью передачи R бит/с. Длина линии составляет m метров, а скорость распространения сигнала — s м/с. Хост А осуществляет передачу пакета длиной L хосту В.

   8.1. Выразите время задержки распространения сигнала d^{pacn} через величины m *us.*

   8.2. Выразите время передачи пакета $d^{Iu;|)}$ через величины L и R.

   8.3. Считая задержки обработки и ожидания нулевыми, вычислите полное время передачи пакета.

   8.4. Пусть передача пакета хостом А начинается в момент времени $t = 0$. Где находится последний бит пакета в момент времени $t = d_{„}^{cp}$?

   8.5. Предположим, что $d^{pacn} > d^{IIcp}$. Где находится первый бит пакета в момент времени $t = d^{nep}$?

   8.6. Предположим, что $d^{paciI} < d^{nop}$. Где находится первый бит пакета в момент времени $t = d^{IIcp}$?

   8.7. Пусть **5** = 2,5 x 10^8, L = 100 бит, R = 28 Кбит/с. При каком значении m $d_{рас\,11} = d_{мер*}^{44}$ V

9. Хост А преобразует «на лету» аналоговые звуковые сигналы в цифровые с частотой 64 Кбит/с, а затем разбивает полученные данные на пакеты размером 48 байт. Хост А соединен с хостом В линией связи со скоростью передачи 1 Мбит/с и задержкой распространения сигнала 2 мс. Сразу после формирования пакета последний отсылается хосту В, где он преобразуется обратно в аналоговую форму. Определите время между созданием бита на передающей стороне и его декодированием на принимающей стороне.

10. Пусть несколько пользователей разделяют линию связи со скоростью передачи 1 Мбит/с. Каждому пользователю необходима скорость передачи 100 Кбит/с, а процент активности каждого пользователя составляет 10 % от времени соединения (см. п. «Сравнение коммутации пакетов и коммутации каналов» в подразделе «Коммутация каналов и коммутация пакетов» раздела «Ядро компьютерных сетей»).

    10.1. Каково количество одновременно поддерживаемых пользователей в случае сети с коммутацией каналов?

    10.2. Пусть в сети вместо коммутации каналов используется коммутация пакетов. Какова вероятность того, что в случайный момент времени заданный пользователь осуществляет передачу?

    10.3. Предположим, что в сети одновременно работают 40 пользователей. Определите вероятность того, что в случайный момент времени одновременную передачу осуществляют *n* пользователей (используйте биномиальное распределение).

    10.4. Определите вероятность того, что передачу одновременно осуществляют не менее 11 пользователей.

11. Обратимся к примеру с линией связи со скоростью 1 Мбит/с (см. п. «Сравнение коммутации пакетов и коммутации каналов» в подразделе «Коммутация каналов и коммутация пакетов» раздела «Ядро компьютерных сетей»). Пользователи осуществляют передачу со скоростью 100 Кбит/с, однако активны лишь с вероятностью $p = 0,1$. Увеличим в этом примере скорость передачи линии связи до 1 Гбит/с.

    11.1. Определите максимальное число пользователей *N,* поддерживаемых одновременно в сети с коммутацией каналов.

    11.2. Пусть в сети используется коммутация пакетов, и одновременное соединение с сетью имеют *M* пользователей. Выразите через величины *p, M* и *N* вероятность того, что *N* пользователей одновременно осуществляют передачу данных.

12. Рассмотрим очередь пакетов в маршрутизаторе. Пусть все пакеты имеют длину I, скорость передачи данных составляет *R* бит/с, а характер входящего трафика таков, что N пакетов появляются одновременно с периодом в *(L x N)/R* с. Определите среднее время задержки ожидания пакета. Учтите, что время задержки для первого пакета равно 0 с, для второго пакета — *L/R* с, для третьего пакета — 2 x *L/R* с, и т. д. Передача *N-ro* пакета полностью завершается к моменту появления очередной порции пакетов.

13. Рассмотрим очередь пакетов в маршрутизаторе. Пусть /= *L* x *a/R* — интенсивность^ трафика. Положим, что задержка ожидания равна *(IxL)/(Rx* (1 - -/))при/< 1.

    13.1. Получите формулу для общей задержки, считая ее равной сумме задержек ожидания и передачи.

    13.2. Нарисуйте график зависимости общей задержки от величины *L/R.*

14. Обратитесь к формуле общей задержки из раздела «Задержки и потери данных в сетях с коммутацией пакетов».

    14.1. Обобщите эту формулу для неравных скоростей обработки, передачи и распространения.

    14.2. Выполните упражнение 14.1 при условии, что среднее значение задержки ожидания в каждом узле составляет $d^{ожид}$.

15. Проведите эксперимент с программой Traceroute, выполнив ее для одной и той же пары хостов, расположенных на одном континенте, в разное время суток.

    15.1/ Рассчитайте среднее значение и стандартное отклонение задержек, полученных для трех опытов.

    15.2. Зафиксируйте число маршрутизаторов в опытах. Изменялось ли оно в зависимости от времени суток?

    15.3. Выделите группы маршрутизаторов, относящихся к одним и тем же Интернет-провайдерам. Это можно сделать путем сравнения адресов или имен маршрутизаторов: если в них есть идентичные фрагменты, то, вероятно, они принадлежат одному Интернет-провайдеру. Являются ли задержки при передаче между Интернет-провайдерами наиболее значительными?

    15.4. Повторите упражнения 15.1-15.3 для пары хостов, лежащих на разных континентах. Сравните полученные результаты с результатами внутриконтинентального обмена.

16. Предположим, что хосты А и В, расстояние между которыми составляет 10 000 км, соединены линией связи со скоростью передачи R = 1 Мбит/с. Скорость распространения сигнала по линии составляет 2,5 x 10^8 м/с.

    16.1. Рассчитайте значение произведения скорости передачи на задержку распространения R x $d^{расп}$.

    16.2. Пусть между хостами А и В осуществляется передача файла размером 400 000 бит. Файл передается в виде одного сообщения. Каково максимальное число битов, находящихся в процессе передачи по линии связи в определенный момент времени?

    16.3. Каков физический смысл произведения скорости передачи на задержку распространения, рассчитанного в упражнении 16.1?

    16.4. Какова «ширина» в метрах одного бита в линии связи? Длиннее ли он футбольного поля?

    16.5. Напишите формулу, связывающую ширину бита со скоростью распространения s, скоростью передачи R и длиной линии связи m.

17. Вернитесь к условию для упражнений 16.1-16.5 и представьте, что величина R является переменной. Для какого значения R ширина бита равна длине линии связи?

18. Вернитесь к условию для упражнений 16.1-16.5, однако считайте, что R = = 1 Гбит/с.

18.1. Рассчитайте значение произведения скорости передачи на задержку распространения $R \times £^{расп}$.

18.2. Пусть между хостами A и B осуществляется передача файла размером 400 000 бит. Файл передается в виде одного сообщения. Каково максимальное число битов, находящихся в процессе передачи по линии связи в определенный момент времени?

18.3. Какова ширина в метрах одного бита в линии связи?

19. Вновь обратитесь к условию для упражнений 16.1-16.5.

19.1. Рассчитайте время передачи файла, считая передачу непрерывной.

19.2. Пусть между хостами A и B осуществляется передача файла размером 400 000 бит, однако теперь он разбит на 10 пакетов размером 40 000 бит каждый. При приеме каждого пакета получатель отсылает отправителю подтверждение, временем передачи которого можно пренебречь. Передача нового пакета невозможна без получения подтверждения для предыдущего пакета. Каково время передачи файла?

19.3. Сравните результаты упражнений 19.1 и 19.2.

20. Предположим, что связь между геостационарным спутником и его базовой станцией на Земле осуществляется по радиоканалу со скоростью передачи 10 Мбит/с. Каждую минуту спутник делает цифровую фотографию земной поверхности и отсылает ее на базовую станцию. Пусть скорость распространения сигнала составляет $2{,}4 \times 10^8$ м/с.

20.1. Какова задержка распространения сигнала для данной линии связи?

20.2. Каково значение произведения скорости передачи на задержку распространения $R \times £^{рас}{}_{„}$?

20.3. Пусть x — размер графического файла, содержащего фотографию. Какова минимальная величина x, при которой передача фотографий осуществляется непрерывно?

21. Обратимся к аналогии с авиакомпанией, приведенной в разделе «Уровни протоколов и модели их обслуживания», а также к принципу добавления заголовков к единицам обмена при их передаче вниз по стеку протоколов. Существует ли аналог заголовков для процесса обслуживания пассажира авиакомпанией?

Дополнительные вопросы и задания

1. В нескольких предложениях опишите каждый из трех основных текущих проектов web-консорциума.
2. Используя беспроводную технологию локальных сетей 802.11b, разработайте компьютерную сеть для вашего дома. Приведите возможные варианты моделей сети и оцените стоимость каждой из них.
3. Что представляет собой связь компьютер-телефон? Найдите в Интернете сайты компаний, занимающихся услугами подобной связи.

4. Что представляет собой служба коротких сообщений (Short Message Service, SMS)? Популярна ли эта служба в мире, и если да, то какими причинами обусловлен успех? Возможна ли посылка коротких сообщений с web-сайта на мобильный телефон?
5. Что такое потоковое аудио? Опишите существующие программные продукты, предназначенные для работы с потоковым аудио. Найдите в Интернете сайты компаний, занимающихся разработкой таких приложений.
6. Что такое видеоконференции через Интернет? Опишите существующие программные продукты, предназначенные для организации видеоконференций. Найдите в Интернете сайты компаний, занимающихся разработкой таких приложений.
7. Что такое одноранговый (P2P) файловый обмен? Укажите названия пяти комланий, предлагающих такой обмен. Какими типами файлов предлагает обмениваться каждая из компаний?
8. Что такое обмен сообщениями в реальном времени? Существуют ли технологии, позволяющие получить доступ к системе обмена сообщениями в реальном времени через портативные устройства?
9. Назовите авторов программы ICQ, первого приложения для обмена сообщениями в реальном времени. Назовите время ее создания и примерный возраст разработчиков. Те же самые сведения укажите для создателей Napster.
10. Найдите Интернет-сайт компании, предоставляющей доступ в Интернет с помощью технологии HFC. Каковы типичные скорости передачи по кабельному модему? Являются ли эти скорости гарантированными для каждого пользователя?

И. Представьте себя разработчиком Интернет-приложения. Какой из протоколов, TCP или UDP, вы бы выбрали для своего приложения? (Мы будем подробно обсуждать этот вопрос в следующих главах, а пока предлагаем вам проявить интуицию.)

Интервью

Леонард Клейнрок является профессором вычислительной техники Калифорнийского университета в Лос-Анджелесе. Принципы коммутации пакетов, разработанные им в 1961 году, стали предвестниками глобальной Сети, а в 1969 году его компьютер стал первым Интернет-узлом. Леонард Клейнрок является основателем и председателем компании Nomadix Inc., предоставляющей широкий спектр услуг по широкополосному доступу в Интернет.

- Что подтолкнуло вас начать научную деятельность в области сетевых технологий?

  Будучи в 1959 году студентом, я заметил, что большинство моих сокурсников стали заниматься исследованиями в области теории информации и кодирования. В нашем университете работал замечательный исследователь Клод Шеннон, в область научных интересов которого входили подобные про-

блемы. К тому времени у него уже было много наработок в этой области, а нерешенные задачи представляли большую сложность и к тому же, как мне казалось, не были принципиальными. Мне хотелось начать деятельность в той области, в которой пока не было фундаментальных разработок. В процессе учебы меня окружало множество вычислительных машин, и я понял, что недалек тот час, когда потребуется организовать общение между ними. В то время еще не было эффективных средств, позволяющих это сделать, поэтому я решил разработать технологию эффективного обмена данными между компьютерами.

- Какова была ваша первая работа в области компьютерных технологий? Какую роль она сыграла для вас?

  В 1951-1957 годах я учился на бакалавра по специальности инженер-электрик. Сначала я работал техником в маленькой промышленной компании Photobell, занимавшейся электроникой, а через некоторое время получил в ней должность инженера. За это время я успел разработать цифровую технологию для линии продуктов компании. В сущности, мы занимались интеграцией цифровых технологий в систему детекции с использованием фотоэлектрических элементов. Позже наши разработки трансформировались в то, что теперь принято называть коммутаторами.

- Каковы были ваши впечатления от первого опыта передачи сообщения?

  Откровенно говоря, я был разочарован. Гораздо более приятным воспоминанием для меня служит 2 сентября 1969 года, когда интерфейсный процессор сообщений впервые соединился с моим компьютером в Лос-Анджелесе. Эту дату я считаю началом Интернета. Годом ранее я заявил, что, как только нам удастся обеспечить стабильную работу сети, мы сможем предоставить доступ к ней из наших домов и офисов так же легко, как к обычной телефонной или электрической сети. В то время я представлял себе перспективу Интернета как вездесущей и невидимой сети, доступной каждому человеку в любой точке пространства в любое время. Но я не мог и предположить, что моя 94-летняя мама будет пользователем Интернета — тем не менее это так!

- Каковы ваши прогнозы на будущее компьютерных сетей?

  Наиболее важные направления развития — мобильные технологии связи и, как следствие, устройства для их применения. Легкие, недорогие, высокотехнологичные портативные устройства приобретают все большую доступность. Мобильные технологии — это средства, позволяющие пользователям получать доступ к компьютерной сети и ее службам без территориальных ограничений. Но я думаю, что это только первый шаг в раскрытии истинного потенциала компьютерных сетей. Далее компьютерные сети в виде сенсорных датчиков, цифровых камер, микрофонов, дисплеев, накапливающих и обрабатывающих информацию устройств проникнут в то, что мы называем окружающей обстановкой: в стены, в предметы интерьера, в машины

и т. д. Например, когда я буду заходить в комнату, она будет «знать» о том, что я вошел. Я смогу общаться с окружающими предметами также, как с живыми людьми, а они будут отвечать мне, подобно серверам, у которых я запрашиваю нужную мне web-страницу.

С технологической точки зрения, на мой взгляд, произойдет распространение множества ключевых сетевых компонентов. Так, к примеру, возможно создание интеллектуальных программных систем, которые будут самостоятельно заниматься сбором информации, ее обработкой, проведением сложных аналитических операций и динамическим выполнением назначенных заданий. Большая часть сетевого трафика будет генерироваться не людьми, а различными встроенными устройствами, в частности этими интеллектуальными системами. В целом, Интернет станет во многом саморегулирующейся системой. Его ожидают огромные потоки информации, передающейся в разные концы планеты, где эта информация будет подвергаться обработке и фильтрации. Интернет способен в будущем стать гигантским «живым» организмом, способным воспринимать, думать и реагировать на действия людей.

- Кто были вашими профессиональными вдохновителями?

  Я многим обязан Клоду Шеннону, замечательному исследователю, благодаря своей потрясающей интуиции воплотившему в реальную жизнь абстрактные математические выкладки. Он был одним из членов ученого совета, принимавшего мою диссертацию.

- Можете ли вы дать совет молодым студентам, желающим стать специалистами в области компьютерных сетей?

  Интернет с его богатым потенциалом является замечательным полем для научной деятельности. Не нужно думать, что технологии сегодняшнего дня — предел совершенства. Ставьте себе новые цели и достигайте их!

ГЛАВА 2 Прикладной уровень

Приложения являются «разумным фундаментом» компьютерных сетей. Не имея приложений, выполняющих полезную работу, бессмысленно говорить о поддерживающих их протоколах. За последние 30 лет было создано множество замечательных приложений для компьютерных сетей.

- Классические текстовые приложения, появившиеся в 1980-е годы, включая текстовую электронную почту, программы организации удаленного доступа к сети, передачи файлов, обработки групп новостей и текстовые чаты.
- Web-приложения, разработанные в середине 1990-х годов.
- Мультимедиа-приложения для работы с потоковым видео, Интернет-радио, Интернет-телефонией и для организации видеоконференций.
- Появившиеся в конце 1990-х годов приложения обмена сообщениями в реальном времени и одноранговые системы совместного доступа к MP3-файлам.

В этой главе мы займемся изучением теоретических и практических аспектов сетевых приложений. Сначала мы рассмотрим ключевые концепции прикладного уровня, такие как протоколы прикладного уровня, клиенты и серверы, процессы, сокеты и интерфейсы транспортного уровня. Затем мы более детально изучим несколько протоколов прикладного уровня: HTTP (для web), SMTP и POP3 (для электронной почты), FTP (для передачи файлов) и DNS (для трансляции имен хостов в IP-адреса).

Следующим этапом для нас станет изучение разработки приложений с использованием протоколов транспортного уровня TCP и UDP. Мы рассмотрим API (Application Programming Interface — прикладной программный интерфейс) сокета и коснемся нескольких простых клиент/серверных приложений, написанных на языке Java. В частности, мы познакомимся с принципами создания web-сервера с помощью средств Java.

Заключительная часть главы будет посвящена более сложному материалу, касающемуся распределения ресурсов, web-кэширования, сетей распределения ресурсов (Content Distribution Networks, CDN), а также одноранговому совместному доступа к файлам. Основное внимание будет уделено оверлейным P2P-сетям, расположенным на прикладном уровне и являющимся «верхушкой» Интернета.

Принципы работы протоколов прикладного уровня

Несмотря на разнообразие сетевых приложений и большое число их взаимодействующих компонентов, почти всегда программное обеспечение является «ядром» приложения. Как было сказано в главе 1, программное обеспечение приложения распределяется между двумя или более оконечными системами (хостами). Так, например, web-приложения обычно состоят из двух взаимодействующих частей: браузера, находящегося на стороне пользователя, и программного обеспечения сервера. Аналогично приложение Telnet состоит из программы на локальном компьютере и программы на удаленном компьютере. Приложение, обеспечивающее проведение видеоконференций, состоит из множества программ, находящихся на всех участвующих в конференции хостах.

На языке операционных систем взаимодействие осуществляется не между программами, а между *процессами.* Процесс можно представить как программу, выполняющуюся на оконечной системе. Если процессы выполняются на одном и том же хосте, их взаимодействие обеспечивается операционной системой хоста и не связано с компьютерной сетью. Нас будет интересовать обмен данными между процессами, расположенными на разных оконечных системах (в общем случае использующих различные операционные системы). Такой обмен осуществляется с помощью *сообщений,* передаваемых через компьютерную сеть. Отправитель генерирует сообщение и посылает его в сеть, а адресат получает это сообщение, выполняет определенные действия и иногда отсылает ответное сообщение отправителю. Сетевые приложения строятся на основе протоколов прикладного уровня, которые регламентируют формат и порядок обмена сообщениями, а также процедуры, выполняемые при приеме или передаче сообщений. На рис. 2.1 приведена схема взаимодействия процессов с использованием прикладного уровня пятиуровневой коммуникационной модели.

Прикладной уровень является хорошей «отправной точкой» для изучения протоколов. Мы часто сталкиваемся с приложениями и, как правило, неплохо знакомы с ними. Это дает нам возможность лучше понять, для чего нужны протоколы прикладного уровня. В свою очередь, знание протоколов прикладного уровня позволяет «спуститься вниз» на транспортный, а затем и на другие уровни коммуникационной модели.

Протоколы прикладного уровня

Необходимо различать понятия *сетевых приложений* и *протоколов прикладного уровня.* Протоколы прикладного уровня являются частью (хотя и весьма большой) сетевых приложений. Рассмотрим два примера. Web является сетевым приложением, позволяющим пользователям получать web-документы по запросу и состоящим из множества компонентов, включая стандарт формата документов (HTML), браузеры (Netscape Navigator, Microsoft Internet Explorer и др.), web-серверы (например, Apache, Microsoft или Netscape), протоколы прикладного уровня. Протокол прикладного уровня для web носит название протокола передачи гипертекста (HyperText Transfer Protocol, HTTP) и описывает формат и порядок обмена со-

общениями между клиентом и сервером (RFC 2646). Таким образом, HTTP является лишь частью web-приложения.

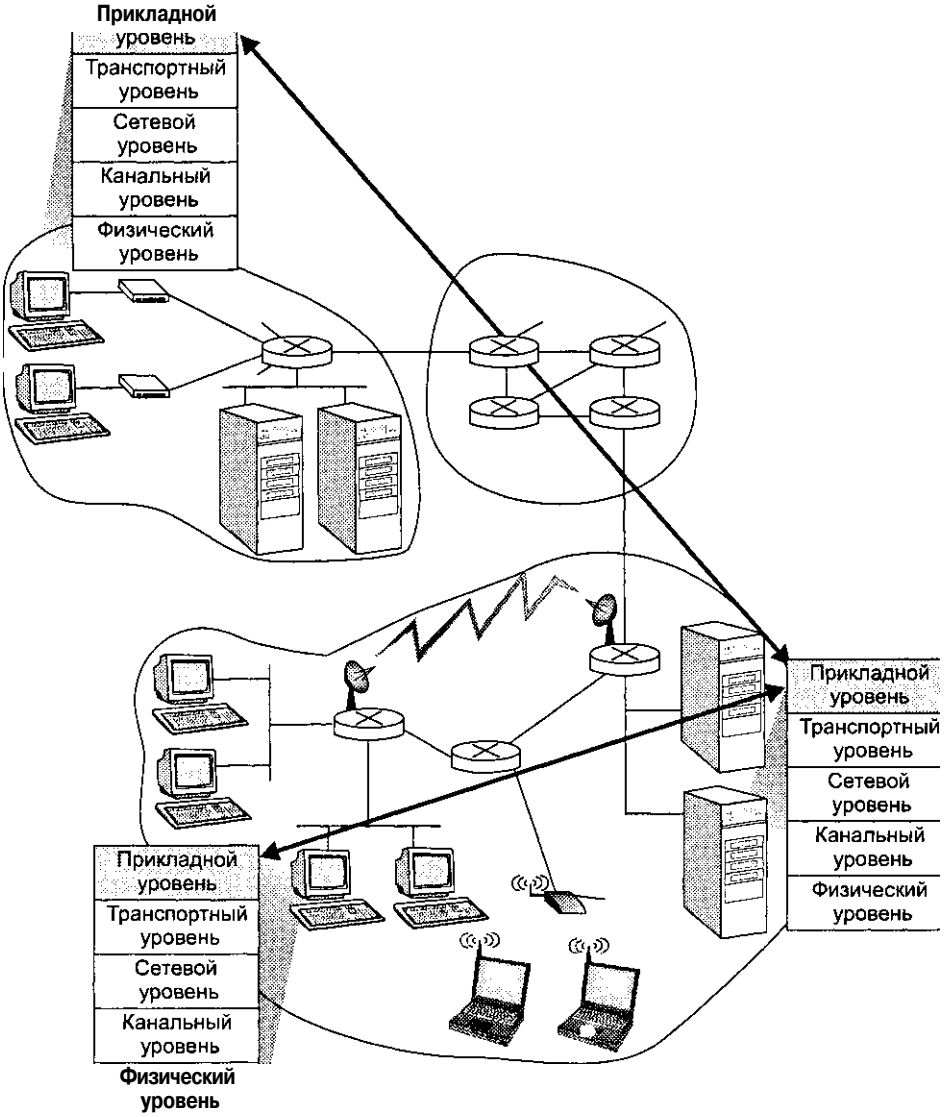


Рис. 2.1. Соединение между приложениями осуществляется на прикладном уровне коммуникационной модели

В качестве второго примера рассмотрим приложение электронной почты. Электронная почта Интернета также состоит из множества компонентов: почтовых серверов, содержащих почтовые ящики пользователей, программ для просмотра и создания электронных писем, стандартов, описывающих структуру электронных

писем, протоколов прикладного уровня, регламентирующих порядок обмена сообщениями серверов между собой и с оконечными системами пользователей, а также интерпретацию полей, из которых состоят электронные письма. Основным протоколом прикладного уровня для электронной почты является протокол простой передачи сообщений (Simple Mail Transfer Protocol, SMTP). Как мы видим, SMTP (RFC 2821) — лишь часть (хотя и достаточно большая) структуры приложений электронной почты.

Как сказано выше, протоколы прикладного уровня определяют способ обмена сообщениями между двумя процессами, выполняющимися на разных оконечных системах. Обычно протокол определяет следующие элементы:

- типы используемых сообщений, например запросы и ответы;
- синтаксис каждого из типов сообщений, описывающий поля сообщения и их разделители;
- семантику полей, то есть смысл информации, содержащейся в каждом из полей сообщения;
- правила, описывающие события, которые вызывают генерацию сообщений.

Некоторые из протоколов прикладного доступа (HTTP, SMTP и др.) являются официально документированными в RFC. Это означает, что если разработчик нового браузера будет следовать стандарту, то браузер сможет получать документы с любого web-сервера, построенного по этому же стандарту. Тем не менее существует множество протоколов прикладного уровня, которые не стандартизированы и при этом используются для поддержки коммерческих продуктов. В частности, это характерно для Интернет-телефонии.

Клиентская и серверная стороны приложения

Как показано на рис. 2.2, сетевое приложение, как правило, состоит из двух «сторон»: *клиентской* и *серверной*. Клиентская и серверная стороны находятся на разных оконечных системах и взаимодействуют путем обмена сообщениями. Так, web-браузер является клиентской стороной HTTP, в то время как программное обеспечение web-сервера представляет собой серверную сторону протокола. Роль клиентской и серверной сторон для SMTP играют соответственно передающий и принимающий почтовые серверы соответственно.

Во многих приложениях хост может играть роль как клиента, так и сервера. Представим себе сеанс Telnet, установленный между хостами A и B (как вы помните, Telnet представляет собой когда-то популярное приложение для организации удаленного доступа). Если сеанс был инициирован хостом A, то хост A будет играть роль клиента, а хост B — роль сервера. Если же инициатором сеанса был хост B, то хосты A и B поменяются ролями. В качестве другого примера представьте себе обмен файлов между двумя хостами по протоколу FTP (File Transfer Protocol — протокол передачи файлов). Во время FTP-сеанса клиент и сервер могут неоднократно меняться местами, при этом клиентом считается та сторона, которая осуществляет прием файла. Тем не менее чаще всего пользуются следующим правилом: клиентом является хост, инициирующий обмен.

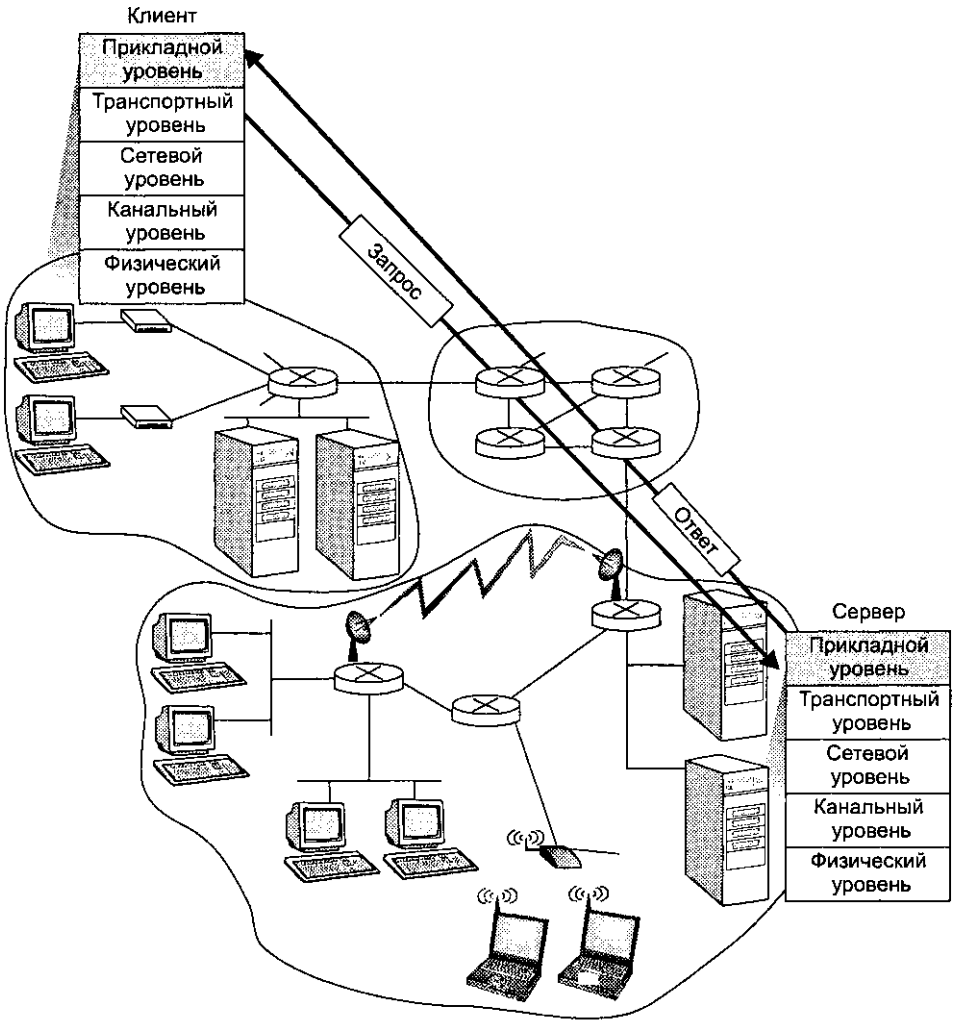


Рис. 2.2. Взаимодействие между клиентом и сервером

Взаимодействие процессов через сеть

Как было сказано выше, многие приложения состоят из двух «сторон», взаимодействующих друг с другом через компьютерную сеть. Взаимодействие осуществляется путем передачи и приема сообщений. Процесс осуществляет прием и передачу сообщений через свой *сокет*. Сокет можно сравнить с дверью: когда процессу необходимо произвести отправку сообщения, он «выталкивает» сообщение через «дверь», предполагая, что некто снаружи (службы более низких уровней) осуществит доставку сообщения до «двери» адресата. Затем сообщение попадает через «дверь» непосредственно приложению-адресату, которое осуществляет его обработку.

На рис, 2.3 изображено взаимодействие двух процессов через Интернет посредством.сокетов (хотя в представленной ситуации используется протокол TCP, с тем же успехом это мог бы быть протокол UDP). Как можно видеть, сокет представляет собой интерфейс между прикладным и транспортным уровнями хоста. Сокет также часто называют прикладным программным интерфейсом (API), осуществляющим связь приложения и компьютерной сети. Под контролем разработчика приложения практически целиком находится часть сокета, относящаяся к прикладному уровню, чего нельзя сказать о его «транспортной» части. Здесь в компетенции разработчика лишь выбор протокола транспортного уровня и установка значений нескольких параметров транспортного уровня (максимальный размер буфера, максимальный размер сегмента и др.). Приложение всегда строится с использованием единственного транспортного протокола. Мы вернемся к обсуждению сокетов в разделах «Программирование TCP-сокетов» и «Программирование UDP-сокетов» этой главы.

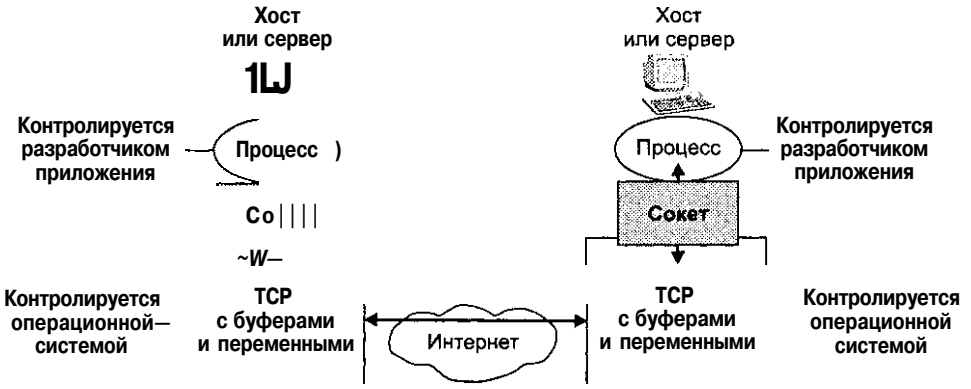


Рис. 2.3. Процессы приложения, сокеты и протокол транспортного уровня

Адресация процессов

Для успешного обмена сообщениями между процессами, выполняющимися на двух различных хостах, необходимо, чтобы они могли идентифицировать друг друга. Идентификация требует наличия следующей информации о процессе:

- имя или адрес хоста, которому принадлежит процесс;
- идентификатор процесса внутри хоста.

Сначала рассмотрим адрес хоста. В Интернет-приложениях хосты идентифицируются с помощью IP-адресов, которые будут подробно изучены нами в главе 4. Пока нам достаточно знать, что IP-адрес представляет собой 32-разрядное двоичное число, уникальное для каждого хоста сети (говоря точнее, это число уникально для каждого интерфейса, с помощью которого осуществляется подключение хоста к сети). Проблема уникальности IP-адресов является очень важной, и в главе 4 мы подробно ее обсудим.

Идентификация процесса внутри хоста производится с помощью уникального для каждого процесса хоста *номера порта*. Популярные Интернет-протоколы прикладного уровня имеют стандартизированные (говорят: *хорошо известные)* значения номеров портов. Так, процесс, использующий протокол HTTP, получает порт номер 80, а процесс, использующий протокол SMTP, — порт номер 25. Хорошо известные номера портов можно найти в документе RFC 1700 (который в настоящее время является несколько устаревшим) и на сайте http://www.iana.org (RFC 3232). Когда разработчик создает новое сетевое приложение, он должен назначить приложению собственный номер порта. Номера портов будут детально рассмотрены в главе 3.

Агенты пользователя

Перед тем как начать более детальное изучение протоколов прикладного уровня, было бы целесообразно ознакомиться с понятием *агента пользователя*. Агентом пользователя называется интерфейс между пользователем и сетевым приложением. Представим себе web. Для web роль агента пользователя играет браузер, например Netscape Navigator или Microsoft Internet Explorer. Браузеры позволяют пользователям просматривать содержимое web-страниц, осуществлять навигацию в web, заполнять формы, взаимодействовать с Java-апплетами и т. д. Кроме того, браузер является клиентской частью протокола HTTP. Таким образом, браузер — это процесс, осуществляющий обмен сообщениями через сокет и одновременно предоставляющий пользовательский интерфейс. В качестве другого примера возьмем приложение электронной почты. Здесь роль агента пользователя играет программа чтения почты, позволяющая обрабатывать электронные письма. Современные программы (Microsoft Outlook, Eudora, AOL, Netscape Communicator) обладают развитым графическим интерфейсом. Как мы увидим в разделе «Электронная почта» этой главы, перечисленные программы зачастую используют как минимум два разных протокола прикладного уровня: SMTP — для отправки электронных писем и POP3 или IMAP — для их доставки.

Службы, необходимые приложению

Вспомним, что сокет является интерфейсом между прикладным процессом и протоколом транспортного уровня. На передающей стороне сообщения через сокет оказываются на транспортном уровне, где получают возможность перемещаться внутри сети. Сетевые службы обеспечивают доставку сообщения на транспортный уровень адресата, где оно через сокет попадает в нужное приложение и обрабатывается им. Многие компьютерные сети, включая Интернет, используют более одного транспортного протокола. При разработке приложения необходимо выбрать один из транспортных протоколов, к службам которого оно будет обращаться. Как сделать выбор? Нужно изучить перечень служб, поддерживаемых каждым из протоколов, и выбрать тот, который способен обслужить ваше приложение наилучшим способом. Подобный выбор вы совершаете, решая, следует ли вам воспользоваться в путешествии поездом или самолетом. У каждого вида транспорта есть свои преимущества (например, поезд совершает остановки между конечными пунктами, а самолет тратит меньше времени в пути).

Какие службы могут понадобиться приложению? Выделяются три основных требования, предъявляемых приложениями к транспортному уровню: надежная передача данных, гарантированная скорость передачи и обеспечение доставки данных за определенное время.

Надежная передача данных

Некоторые приложения, например приложения электронной почты, обмена сообщениями в реальном времени, передачи файлов, просмотра web-документов, финансовых операций и т. д., требуют надежной передачи данных, то есть исключения вероятности потерь данных при передаче. Как правило, потери данных приводят к крайне нежелательным для пользователей последствиям (представьте обмен между банком или его клиентом!). Тем не менее существует вид приложений, *толерантных к потерям данных.* К нему относится большинство мультимедийных приложений, например аудио и видео реального времени. Небольшие потери данных в таких приложениях оборачиваются помехами (звуковые щелчки и «дергающееся» изображение), не приводящими к сбоям или серьезным потерям качества. Степень толерантности приложения к потере данных определяет максимальную долю данных, которая может быть потеряна, и, как правило, зависит от назначения приложения и использующейся схемы кодирования.

Скорость передачи

Для эффективной работы некоторым приложениям необходимо осуществлять передачу данных с определенной скоростью. Например, если приложение Интернет-телефонии кодирует аналоговые голосовые сообщения в цифровые с интенсивностью 32 Кбит/с, то для успешного функционирования необходимо обеспечить передачу данных этого приложения со скоростью 32 Кбит/с. В противном случае между фразами пользователей будут ощущаться задержки. Для избежания таких ситуаций приложение должно либо снизить интенсивность кодирования до величины, согласующейся со скоростью передачи, либо завершить свою работу. Приложения, эффективность которых зависит от скорости передачи данных, называют *чувствительными к скорости передачи данных.* На сегодняшний день многие мультимедиа-приложения являются чувствительными к скорости передачи, однако в будущем ожидается кардинальное усовершенствование систем кодирования, которые позволят приложениям адаптироваться к используемому каналу связи. Такой способностью обладают приложения электронной почты, web-приложения и приложения для передачи файлов, относящиеся к классу *эластичных* приложений. Разумеется, наличие высокоскоростного канала связи никогда не повредит работе сети; здесь весьма актуально утверждение о том, что полоса пропускания никогда не бывает слишком широкой.

Время передачи

Последнее требование приложений к транспортному уровню заключается в гарантированном времени доставки данных. Интерактивные приложения реального времени, такие как Интернет-телефония, виртуальные миры, телеконференции и многопользовательские компьютерные игры, накладывают жесткие временные

ограничения на время доставки данных (сотни миллисекунд и менее). Невыполнение временных ограничений в Интернет-телефонии приводит к длительным паузам в разговоре, а в компьютерных играх *- к задержке реакции окружения и, следовательно, к потере реалистичности. В приложениях, не относящихся к приложениям реального времени, временные ограничения на доставку данных не являются столь принципиальными, однако меньшая задержка всегда предпочтительней, чем большая.

В табл. 2.1 суммируются требования различных видов приложений к качеству передачи данных. Разумеется, не следует воспринимать эти данные как строгую классификацию; напротив, они отражают лишь тенденции, характерные для того или иного класса приложений.

Таблица 2.1. Требования приложений к качеству передачи данных

Приложение доставки	Потеря данных	Скорость передачи	Ограничение на время
Передача файлов	Недопустима	Эластичность	Нет
Электронная почта	Недопустима	Эластичность	Нет
Работа с web-документами	Недопустима	Эластичность (несколько Кбит/с)	Нет
Аудио и видео реального времени	Допустима	Аудио: несколько Кбит/с-1 Мбит/с, видео: 10Кбит/с-5 Мбит/с	Есть, сотни миллисекунд
Записанное потоковое аудио и видео	Допустима	Аналогично предыдущему	Есть, несколько секунд
Интерактивные игры	Допустима	1-10 Кбит/с	Есть, сотни миллисекунд
Обмен сообщениями в реальном времени	Недопустима	Эластичность	Есть и нет

Службы протоколов транспортного уровня

В Интернете и других сетях, использующих семейство протоколов TCP/IP, существуют два транспортных протокола: *TCP* (Transmission Control Protocol — протокол управления передачей) и *UDP* (User Datagram Protocol — протокол пользовательских дейтаграмм). При создании нового Интернет-приложения разработчику необходимо выбрать, каким из двух протоколов, TCP или UDP, будет пользоваться его продукт. Эти протоколы предлагают приложениям принципиально разные модели обслуживания.

Протокол TCP

Модель обслуживания протокола TCP опирается на установление логического соединения и надежную передачу данных. Поясним, что означают эти два термина.

- *Установление логического соединения.* Протокол TCP обеспечивает обмен управляющей информацией между клиентом и сервером до начала передачи «полезных» данных. Этот предварительный обмен, называемый процедурой рукопожатия, предназначен для подготовки обеих сторон к передаче серии пакетов. После удачного завершения процедуры рукопожатия между сокетами клиента и сервера устанавливается *TCP-соединение.* TCP-соединение является дуплексным, то есть стороны могут осуществлять передачу информации друг другу одновременно. После окончания обмена соединение должно быть автоматически разорвано. TCP-соединение называется логическим потому, что представляет собой совокупность информационных единиц (переменных). Понятие логического соединения будет детально рассмотрено в главе 3.
- *Надежная передача данных.* Взаимодействующие процессы получают гарантию, что все переданные данные будут доставлены адресату без ошибок, потерь и в правильном порядке. Выходной поток байтов передающей стороны в точности соответствует входному потоку байтов принимающей стороны.

TCP также включает механизм контроля перегрузки, который предназначен скорее для сети, нежели для обеспечения качества обслуживания взаимодействующих процессов. Если сеть перегружена, происходит автоматическое снижение скорости обмена данными. Как мы увидим в главе 3, средства контроля перегрузки стремятся поровну разделить пропускную способность сети между всеми TCP-соединениями.

Принудительное снижение скорости обмена может весьма неблагоприятно сказаться на функционировании мультимедиа-приложений, особенно связанных с реальным временем. Поскольку мультимедиа-приложения толерантны к потерям данных, служба надежной передачи является для них избыточной, и в большинстве случаев разработчики выбирают для мультимедийных целей протокол UDP.

Рассмотрев услуги, предлагаемые протоколом TCP, скажем несколько слов о том, чего он не может. Во-первых, TCP не обеспечивает гарантированную скорость передачи данных. Скорость передачи данных приложением устанавливается самим протоколом и не обязательно соответствует требованию приложения. Более того, TCP принудительно снижает скорость передачи при наличии перегрузок в сети. Во-вторых, TCP не дает никаких гарантий относительно времени доставки сообщений. Доставка большинства сообщений происходит успешно, однако может потребовать неопределенных и трудно прогнозируемых временных затрат, иногда достигающих десятков секунд или даже нескольких минут.

Протокол UDP

Протокол UDP предоставляет приложению весьма простую и бесхитростную модель обслуживания. Логическое соединение между сокетами не устанавливается, следовательно, процедура рукопожатия в протоколе отсутствует. UDP обеспечивает ненадежную передачу данных, означающую отсутствие приложения, посылающего пакет гарантии того, что этот пакет будет получен адресатом. Более того,

протокол не гарантирует, что порядок получения информации будет соответствовать порядку ее отправления.

Протокол UDP не предусматривает контроля перегрузок, и приложение может осуществлять передачу данных с той скоростью, которая ему необходима. Поскольку приложения реального времени обычно толерантны к потерям данных и в то же время требуют наличия некой минимальной скорости передачи, протокол UDP для них предпочтительнее, чем TCP. Тем не менее UDP, как и TCP, не гарантирует время доставки данных.

В табл. 2.2 собраны сведения о протоколах, использующихся в популярных Интернет-приложениях. Как можно видеть, приложения электронной почты и удаленного доступа, а также web-приложения и приложения для передачи файлов используют протокол TCP. Это обусловлено тем, что все перечисленные приложения требуют обслуживания с надежной передачей данных, гарантирующего полноту и корректность получаемой информации. В то же время приложения Интернет-телефонии используют протокол UDP, поскольку им необходимо поддерживать минимальную скорость передачи данных, что невозможно в случае протокола TCP. Приложения Интернет-телефонии толерантны к потере данных и, следовательно, не нуждаются в надежной передаче, предлагаемой протоколом TCP.

Таблица 2.2. Протоколы, использующиеся в популярных Интернет-приложениях

Приложение	Прикладной протокол	Транспортный протокол
Электронная почта	**SMTP (RFC 2821)**	**TCP**
Доступ с удаленного терминала	**Telnet (RFC 854)**	**TCP**
Web	**HTTP (RFC 2616)**	**TCP**
Передача файлов	**FTP (RFC 959)**	**TCP**
Удаленный файловый сервер	**NFS [325]**	**UDP или TCP**
Потоковое мультимедиа	**Обычно фирменный (например, Real Networks)**	**UDP или TCP**
Интернет-телефония	**Обычно фирменный (например, Dialpad)**	**Как правило, UDP**

Как уже упоминалось, ни TCP, ни UDP не гарантируют время доставки сообщений. Означает ли это, что приложения, требующие временных ограничений, не могут функционировать в Интернете? Разумеется, нет — в течение многих лет Интернет обеспечивает работу таких приложений. Секрет заключается в особом проектировании приложений, позволяющем в значительной степени «обойти» этот недостаток протоколов транспортного уровня; мы раскроем некоторые из секретов подобного проектирования в главе 6. Необходимо отметить, что «хитроумное» проектирование все же не способно принципиально решить проблему гарантированного времени доставки и в критических ситуациях не приводит к желаемому качеству обслуживания. В главе 6 также пойдет речь о разрабатываемых моделях обслуживания для Интернета, которые в будущем смогут обеспечить доставку информации за время, необходимое приложению.

Интернет-приложения, рассматриваемые в этой книге

Новые приложения для Интернета разрабатываются практически каждый день. Вместо того чтобы сводить разговор об Интернет-приложениях к их простому перечислению, мы ограничимся лишь несколькими наиболее важными и популярными приложениями: web-приложениями, приложениями для передачи файлов и электронной почтой, системами доменных имен (Domain Name System, DNS) и одноранговыми системами обмена файлами. Сначала мы рассмотрим web-приложения, поскольку используемый ими протокол HTTP весьма наглядно иллюстрирует основные принципы построения сетевых протоколов. Затем мы изучим протокол передачи файлов, сравним его с HTTP и выделим несколько дополнительных принципов. Далее предметом нашего рассмотрения станет электронная почта. Мы узнаем о том, что современные приложения электронной почты используют не один, а несколько протоколов прикладного уровня. Затем мы познакомимся с системой доменных имен, с помощью которой производится трансляция адресов в Интернете. Большинство пользователей взаимодействует с DNS не напрямую, а через другие приложения, в частности через web-приложения, а также приложения электронной почты и передачи файлов. DNS замечательно иллюстрирует организацию распределенной базы данных в Интернете. Наконец, последним приложением, которое мы рассмотрим, будет являться одноранговая система обмена файлами, часто используемая музыкальными сайтами, хранящими общедоступные коллекции MP3-файлов.

Web и HTTP

До 1990-х годов пользователями ресурсов Интернета были исследователи, ученые и студенты, которые подключались к удаленным хостам, обменивались с ними файлами, получали сообщения из групп новостей и пользовались услугами электронной почты. Несмотря на то что Интернет-приложения уже тогда обладали огромной потенциальной пользой, Интернет еще был мало распространен среди широких масс. Ситуация резко изменилась в начале 1990-х годов с появлением Всемирной паутины (World Wide Web, WWW, или web). Без преувеличения можно сказать, что именно Всемирная паутина изменила взаимодействие между людьми, выделила Интернет из множества других компьютерных сетей (Prodigy, America Online, Compuserve, Minitel) и сделала слово «Интернет» синонимом термина «компьютерная сеть».

История знает несколько примеров, когда изобретение новых коммуникационных технологий повлекло за собой значительные социальные последствия. Первым примером является изобретение телефона в 1870-е годы, позволившего общаться людям, находящимся на большом расстоянии друг от друга. Второй пример — создание широковещательных радио и телевидения в 20-30-х годах прошлого века, обеспечивших возможность передачи больших объемов информации широкой аудитории. И, как вы, вероятно, догадались, в качестве третьего исторического примера мы рассматриваем создание web. Пожалуй, главной особенностью web,

отличающей ее от других информационных технологий, является ее активация *по запросу*. Пользователи получают ту информацию, которая им нужна, и в то время, когда она им нужна. Радио и телевидение, к примеру, лишены этой возможности, поскольку зрители и слушатели не имеют контроля над информационным потоком. Кроме того, в web механизм получения информации и ее размещения предельно прост. Каждый может за небольшую плату разместить в Сети массу сведений. Гиперссылки и поисковые системы позволяют не «утонуть» в океане разнообразных web-сайтов. Высококачественная графика увеличивает наглядность представляемой информации, а формы, скрипты, апплеты, элементы ActiveX и прочие средства позволяют сделать общение пользователя с сетью интерактивным. Кроме того, все большую мощь в Интернете набирают новые формы представления аудио- и видеоинформации, а скоро пользователи получат еще более широкий спектр средств мультимедиа.

ИСТОРИЧЕСКАЯ СПРАВКА

В апреле 1994 года Марк Андрессен, ученый, ранее возглавлявший разработку браузера Mosaic, и Джим Кларк, бывший профессор Стенфордского университета и основатель компании Silicon Graphics, образовали корпорацию Netscape Communication. В состав корпорации вошли многие ученые, вместе с Андрессеном занимавшиеся созданием браузера Mosaic, и в октябре 1994 года вышла в свет бета-версия продукта Netscape Navigator 1.0. В последующие годы компания приложила множество усилий для развития нового браузера и другихтехнологий: web-серверов, коммерческих серверов, почтовых серверов, серверов новостей, прокси-серверов, программ чтения электронной почты и др. Netscape Communication по праву можно считать одной из самых прогрессивных и успешных Интернет-компаний середины 1990-х, а в августе 1995 года громкий публичный успех пришел к браузеру Netscape.

Компания Microsoft, изначально не проявлявшая значительной активности по продвижению своих интересов в Интернет, выпустила 1 -ю версию браузера Microsoft Internet Explorer в августе 1995 года. Продукт не отличался изяществом и скоростью, однако компания вложила значительные инвестиции в его развитие, и к 1997 году Microsoft и Netscape шли бок о бок в «браузерной гонке». 11 июня 1997 года Netscape выпустила версию 4.0 своего браузера, а 30 сентября вышла в свет версия 4.0 Microsoft Internet Explorer. В то время еще не сложилось устоявшегося мнения о том, какой из браузеров лучше, а компания Microsoft, обладавшая монополией на свою операционную систему Windows, набирала все большую коммерческую мощь. В 1997 году компания Netscape допустила ряд решающих просчетов: не была осознана важность создания портала на основе web-сайта компании, кроме того, было принято ошибочное решение о полном переходе браузера на Java-технологию [105]. В конечном счете, 1998 год ознаменовался для Netscape Communication снижением ее доли на рынке браузеров и других продуктов, в конце года она была приобретена компанией America Online, а Марк Андрессен и большая часть его команды покинули свое бывшее детище.

Обзор HTTP

В «сердце» web находится протокол передачи гипертекста (HTTP), являющийся протоколом прикладного уровня. Описание HTTP можно найти в RFC 1945 и RFC 2616. Протокол HTTP реализуется с помощью двух программ: клиента и сер-

вера, которые, находясь на разных оконечных системах, обмениваются HTTP-сообщениями. Порядок обмена и содержание сообщений описаны в протоколе. Перед тем как углубиться в изучение HTTP, сначала освоим терминологию, используемую в контексте web.

Каждая *web-страница,* или *документ,* состоит из *объектов.* Объект представляет собой обычный файл в формате HTML, изображение в формате JPEG или GIF, Java-апплет, аудиоклип и т. п., то есть единицу, обладающую собственным универсальным указателем ресурса (Uniform Resource Locator, URL). Как правило, web-страницы состоят из базового HTML-файла и объектов, на которые он ссылается. Так, если web-страница включает базовый HTML-файл и пять изображений, то она состоит из шести объектов. Ссылки на объекты, относящиеся к web-странице, представляют собой URL-адреса, включенные в базовый HTML-файл. URL состоит из двух частей: имени хоста сервера, на котором находится объект, и пути к объекту. Так, например, для URL www.someSchool.edu/someDepartment/picture.gif именем хоста является фрагмент www.someSchool.edu, а путем к объекту — фрагмент someDepartment/picture.gif. *Браузером* называется агент пользователя web; он отображает web-страницы, а также выполняет множество дополнительных служебных функций. Кроме того, браузеры представляют клиентскую сторону протокола HTTP. Таким образом, термины «браузер» и «клиент» в контексте web будут употребляться как эквивалентные. В число наиболее популярных браузеров входят Netscape Navigator и Microsoft Internet Explorer.

Web-сервер содержит объекты, каждый из которых идентифицируется своим URL-адресом. Кроме того, web-серверы представляют серверную сторону протокола HTTP. К наиболее популярными web-серверам следует отнести Apache и Microsoft Internet Information Server.

Протокол HTTP определяет, каким образом клиенты (например, браузеры) запрашивают web-страницы, а серверы осуществляют передачу этих страниц. Более подробный разговор о взаимодействии клиента и сервера мы проведем позднее, однако основную идею можно понять из рис. 2.4. Когда пользователь запрашивает web-страницу (например, совершает щелчок на гиперссылке), браузер посылает серверу HTTP-запрос объектов, составляющих web-страницу. Сервер получает запрос и высылает ответные сообщения, содержащие требуемые объекты. В 1997 году практически все web-браузеры и web-серверы стали поддерживать протокол HTTP версии 1.0, описанный в документе RFC 1945. В 1998 году начался переход к версии 1.1, которая была описана в документе RFC 2616. Версия 1.1 имеет обратную совместимость с версией 1.0, то есть любой сервер или браузер, использующий версию 1.1, может в полной мере взаимодействовать с браузером или сервером, поддерживающим версию 1.0.

Как HTTP 1.0, так и HTTP 1.1 используют TCP в качестве протокола транспортного уровня. HTTP-клиент сначала устанавливает TCP-соединение с сервером, а после создания соединения клиент и сервер начинают взаимодействовать с протоколом TCP через интерфейс сокетов. Как было сказано ранее, сокеты представляют собой «двери» между процессами и протоколом транспортного уровня.

Клиент посылает запросы и принимает ответы через свой интерфейс сокетов, а сервер использует интерфейс сокетов для получения запросов и их выполнения. После того как web-запрос минует сокет клиента, он оказывается «в руках» протокола TCP. Вспомним, что одной из функций протокола TCP является обеспечение надежной передачи данных; это означает, что каждый запрос, посылаемый клиентом, и каждый ответ сервера доставляются в виде, точно соответствующем отправленному. Здесь проявляется одно из достоинств многоуровневой коммуникационной модели: протоколу HTTP не нужно контролировать надежность передачи и обеспечивать повторную передачу пакетов при искажениях. Вся «черновая» работа будет проделана протоколом TCP и протоколами более низких уровней.

Рис. 2.4. Передача запросов и ответов HTTP

Необходимо отметить, что после завершения обслуживания клиентов сервер не сохраняет о них никакой информации. Если, например, какой-либо клиент сделает два запроса одного и того же ресурса подряд, сервер выполнит их, не выдав клиенту никакого оповещения о дублирующем запросе. Говорят, что протокол HTTP является *протоколом без запоминания состояния* (stateless protocol) соединения.

Постоянные и непостоянные соединения

Протокол HTTP поддерживает постоянные и непостоянные соединения (за исключением версии 1.0, которая поддерживает только непостоянные соединения). При непостоянном соединении протокол TCP получает лишь один объект, а при постоянном соединении (используемом по умолчанию в HTTP версии выше 1.0) — все объекты. Разумеется, клиенты и серверы, поддерживающие протокол HTTP 1.1, при желании можно настроить и на непостоянное соединение.

Непостоянное соединение

Рассмотрим, каким образом осуществляется передача web-страницы от сервера к клиенту в случае непостоянного HTTP-соединения. Предположим, что страница состоит из базового HTML-файла и десяти JPEG-изображений, находящихся на одном сервере. Пусть URL базового HTML-файла имеет вид **www.someSchooLedu/someDepartment/home.index.** Процесс обмена между клиентом и сервером состоит из следующих шагов.

1. HTTP-клиент инициирует TCP-соединение с сервером **www.someSchool.edu** через порт номер **80,** который по умолчанию является номером порта для HTTP.
2. HTTP-клиент посылает запрос серверу через сокет, выделенный TCP-соединению, которое было установлено на шаге **1.** Запрос включает путь к базовому HTML-файлу: **someDepartment/home.index** (чуть позже мы рассмотрим HTTP-сообщения более детально).
3. HTTP-сервер получает запрос через сокет, ассоциированный с установленным соединением, извлекает объект **someDepartment/home.index,** формирует ответ, включающий объект, и отсылает его клиенту через сокет.
4. HTTP-сервер закрывает TCP-соединение (окончательный разрыв соединения происходит после того, как сервер получает информацию об успешной передаче объекта).
5. HTTP-клиент принимает ответ сервера. TCP-соединение завершается. Клиент обрабатывает сообщение, в котором указано, что доставленный объект является базовым HTML-файлом. Клиент извлекает файл, обрабатывает его и выделяет ссылки на **10** объектов (JPEG-файлов).
6. Шаги **1-4** повторяются для каждого из **10** объектов.

После получения web-страницы браузер отображает ее на экране. Необходимо помнить, что различные браузеры могут по-разному интерпретировать одну и ту же web-страницу. Протокол HTTP никак не связан со способом визуализации web-страниц; спецификации, содержащиеся в документах RFC **1945** и RFC **2616,** описывают только метод обмена информацией между клиентом и сервером.

Описанная модель взаимодействия относится к непостоянному соединению, то есть соединению, не позволяющему осуществлять передачу нескольких объектов. Для получения web-страницы требуется многократное установление и завершение соединения (в приведенном выше примере необходимо установить **11** соединений). Каждое соединение состоит из единственных сообщения-запроса и сообщения-ответа.

Приведенная модель не дает ответа на вопрос, являются ли TCP-соединения последовательными или параллельными; другими словами, одновременно ли были получены несколько объектов. Параллелизм TCP-соединений возможен, при этом его степень (максимальное число одновременно устанавливаемых соединений) в современных браузерах конфигурируется пользователями. В большинстве случаев браузеры открывают от **5** до **10** параллельных TCP-соединений; тем не менее можно установить степень параллелизма, равную **1,** что приведет к открытию каждого нового соединения только при завершении предыдущего. Как мы убедимся

в следующей главе, параллельная передача объектов позволяет сократить время ответа сервера.

Теперь попробуем оценить величину временного интервала, проходящего с момента запроса клиентом web-страницы до окончания ее передачи. Здесь мы воспользуемся понятием *времени оборота* (Round-Trip Time, RTT), то есть времени, требующемуся пакету малой длины для передачи от клиента серверу и обратно. Время оборота включает в себя задержку распространения, ожидания и обработки (см. раздел «Задержки и потери данных в сетях с коммутацией пакетов» в главе 1). Рассмотрим, что происходит, когда пользователь совершает щелчок на гиперссылке. Как показано на рис. 2.5, браузер инициирует TCP-соединение с web-сервером, которое устанавливается после «тройного рукопожатия»: клиент посылает серверу небольшой TCP-сегмент, сервер отвечает схожим сегментом (подтверждением), и наконец, клиент посылает серверу еще один сегмент-подтверждение. Для однократного обмена сегментами требуется время, равное времени оборота. Вместе с последним сегментом рукопожатия клиент отсылает серверу свой запрос, а сервер после получения запроса высылает клиенту базовый HTML-файл. Этот фрагмент взаимодействия также вызывает задержку на время оборота. Таким образом, суммарное время ответа складывается из удвоенного времени оборота и времени передачи базового HTML-файла.

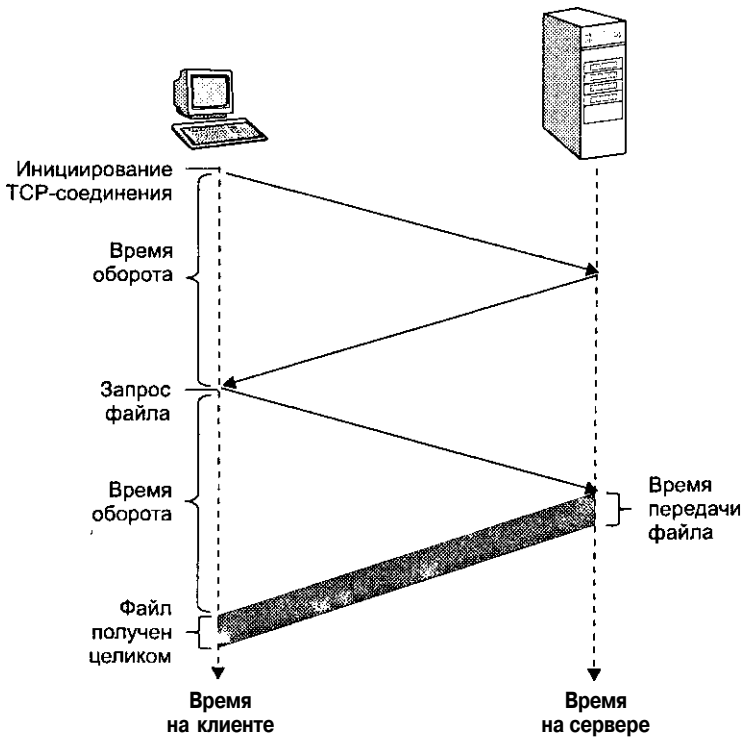


Рис. 2.5. Оценка времени ответа на запрос HTML-файла

Постоянные соединения

Непостоянные соединения обладают рядом недостатков. Прежде всего для каждого запрашиваемого объекта должно устанавливаться новое соединение. При этом необходимо учитывать, что каждое соединение требует от протокола TCP выделения буфера, а также ряда служебных переменных как на стороне клиента, так и на стороне сервера. Учитывая то, что многие web-серверы параллельно обслуживают сотни клиентов, подобная схема серьезно затрудняет процесс взаимодействия между клиентами и сервером. Кроме того, установление соединения для каждого объекта из-за времени оборота приводит к дополнительным временным затратам.

При постоянном соединении сервер не закрывает TCP-соединение после обслуживания запроса, что позволяет обслужить несколько запросов в одном соединении. Так, если в нашем примере применить механизм постоянных соединений, то вся web-страница, включающая базовый HTML-файл и 10 изображений, будет передана клиенту через одно TCP-соединение. Передача web-страниц через одно соединение возможна в случаях, если все объекты находятся на одном и том же хосте. Обычно закрытие TCP-соединения происходит в случае, когда оно не используется в течение некоторого установленного времени (интервала ожидания).

Постоянные соединения делятся на два класса: *с конвейеризацией* и *без конвейеризации*. В соединениях без конвейеризации клиент посылает серверу новый запрос после того, как завершается прием текущего объекта. Это позволяет сократить время выполнения запроса на величину времени оборота по сравнению с непостоянными соединениями. Тем не менее даже такой заметный прогресс не является пределом; соединения с конвейеризацией имеют еще меньшее время выполнения запроса. Обратите внимание на еще один недостаток соединений без конвейеризации: после окончания передачи ответа сервер простаивает, ожидая нового запроса. Это приводит к неэкономному расходованию серверных ресурсов.

По умолчанию протокол HTTP 1.1 настроен на использование постоянных соединений с конвейеризацией. В этом режиме клиент формирует запрос сразу после обнаружения ссылки на объект. Это позволяет новому запросу направляться к серверу, не дожидаясь окончания обслуживания других запросов. Аналогично, сервер, получая новые запросы, начинает их немедленное обслуживание. Очевидны достоинства конвейерного механизма: временные расходы на установление соединения сводятся к одной задержке на время оборота для всей web-страницы, а время простоя сервера значительно сокращается. Мы количественно сравним модели постоянных и непостоянных соединений в упражнениях к главам 2 и 3. Заинтересованный читатель может также обратиться к литературе [200, 359].

Формат HTTP-сообщения

Описания протокола HTTP, содержащиеся в документах RFC 1945 и RFC 2616, определяют формат сообщений, предназначенных для обмена между клиентом и сервером. В HTTP существуют два типа сообщений: запросы и ответы, которые будут рассмотрены ниже.

Сообщение-запрос

Типичное сообщение-запрос протокола HTTP выглядит следующим образом:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept-language:fr
```

Это сообщение, несмотря на свою простоту, весьма наглядно демонстрирует формат, используемый в HTTP. Как можно видеть, сообщение представляет собой совокупность вполне понятных человеку текстовых символов в кодировке ASCII. Сообщение состоит из пяти строк, каждая из которых оканчивается парой символов для перехода на новую строку (возврат каретки и перевод строки), а последняя строка — дополнительной парой указанных символов. В общем случае число строк сообщения может быть как больше, так и меньше пяти (вплоть до одной строки). Первая строка называется *строкой запроса,* а следующие строки — *строками заголовка.* Строка запроса содержит три поля: поле метода, поле URL и поле версии HTTP. Поле метода может принимать различные значения, например **GET, POST** и **HEAD.** Метод GET является наиболее часто используемым методом протокола HTTP и применяется в случаях, когда требуемый объект идентифицируется URL-адресом. Приведенное сообщение содержит URL-адрес **/somedir/page.html.** Поле версии HTTP не требует дополнительных комментариев и в нашем примере содержит запись **HTTP**/1.1.

Теперь рассмотрим строки заголовка. Строка **Host: www.someschool.edu** содержит адрес хоста, на котором находится объект. С помощью строки **Connection: close** браузер сообщает серверу о том, что не следует использовать постоянное соединение, и установленное TCP-соединение должно быть закрыто сразу после передачи требуемого объекта. Обратите внимание, что при этом браузер поддерживает версию 1.1 протокола HTTP. В строке **The User-agent:** указан агент пользователя, то есть тип браузера, сгенерировавшего запрос. В данном случае это браузер Mozilla 4.0 фирмы Netscape. Строка **User-agent:** является весьма полезной, поскольку на сервере могут храниться несколько версий одного документа, предназначенных для разных браузеров и адресуемых одним URL-адресом. Наконец, строка **Accept-language:** указывает на то, что пользователю по возможности должна быть выслана версия документа на французском языке (в случае ее наличия на сервере); в противном случае будет выслана версия документа на языке, заданном по умолчанию. Строка **Accept-language:** является одной из множества заголовочных строк согласования данных, предусмотренных протоколом HTTP.

Рассмотрев конкретный пример, обратимся теперь к общему формату запроса, представленному на рис. 2.6. Как можно видеть, пример вполне соответствует этому формату; тем не менее после строк заголовка и пустой строки формат сообщения предусматривает наличие тела сообщения. Тело сообщения остается пустым при использовании метода GET и заполняется при использовании метода POST. Метод POST применяется в случаях, когда пользователь заполняет формы, например вводит слово для поиска в поисковой системе. Заполнение форм приводит к генерации запроса, а содержимое web-страницы зависит от данных, введенных

в формы. Итак, если поле метода содержит значение POST, то в теле сообщения находятся данные, введенные в формы.

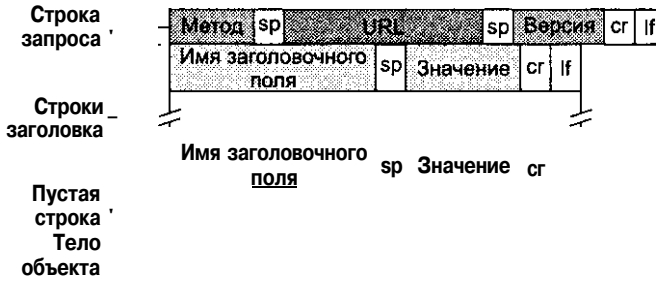


Рис. 2.6. Общий формат сообщения-запроса

Необходимо отметить, что в запросах, создаваемых с помощью форм, не всегда применяется метод POST. Напротив, HTML-формы часто используют метод GET и подставляют введенные значения в URL-адрес требуемой страницы. К примеру, если пользователь ввел в формы два значения, monkeys и bananas, то запрашиваемый с помощью метода GET URL-адрес будет иметь вид www.somesite.com/animalsearch?monkeys&bananas. Вполне вероятно, что вы нередко встречали подобные конструкции, путешествуя в web.

Метод HEAD схож с методом GET. При получении запроса с методом HEAD сервер формирует ответ, однако не осуществляет пересылку объекта. Разработчики приложений часто используют метод HEAD для отладки ошибок.

В спецификации HTTP/1.0 указаны лишь три метода: GET, POST и HEAD. Спецификация HTTP/1.1 располагает более широким набором методов, в который, кроме перечисленных выше, входят PUT и DELETE. Метод PUT часто применяется в средствах web-публикаций и позволяет поместить объект с заданным URL-адресом на web-сервер, а метод DELETE — удалить объект, расположенный на web-сервере.

Сообщение-ответ

Ниже приведен пример типичного ответа, генерируемого HTTP-сервером.

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

Рассмотрим структуру этого сообщения. Оно состоит из трех частей: *строки состояния,* шести *строк заголовка* и *тела сообщения.* Тело сообщения содержит тре-

буемый объект. Строка состояния образована из трех полей: поля версии протокола, поля кода состояния и поля соответствующей коду информации, описывающей это состояние. В данном примере строка состояния говорит о том, что сервер использует спецификацию HTTP/1.1, требуемый объект найден и осуществляется его пересылка.

Теперь обратимся к строкам заголовка. Сервер использует строку Connection: close для уведомления клиента о том, что TCP-соединение будет закрыто после того, как окончится пересылка объекта. Строка The Date: содержит дату и время создания ответа. Обратите внимание, что эта дата не относится к созданию или последнему изменению объекта, а указывает момент извлечения объекта из места его хранения и включения в тело сообщения. Строка Server: говорит о том, что сообщение было создано сервером Apache, и она аналогична строке User-agent: в сообщении-запросе. Строка Last-Modified: содержит дату и время создания или последнего изменения объекта. Как мы увидим немного позже, содержимое строки Last-Modified: крайне важно для кэширования объектов как на локальных клиентах, так и на сетевых кэш-серверах (часто называемых прокси-серверами). Строка Content-Length: содержит размер пересылаемого объекта в байтах, а строка Content-Type: указывает на то, что объект является текстом в формате HTML (обратите внимание, что тип объекта определяется содержимым строки Content-Type: и не зависит от расширения файла).

Если сервер получает запрос, в котором указана версия HTTP/1.0, постоянное соединение не будет использоваться, даже при поддержке сервером протокола HTTP/1.1. Это необходимо потому, что спецификация HTTP 1.0 не предусматривает постоянных соединений.

Рассмотрев частный случай, обратимся к общему формату ответного сообщения, представленному на рис. 2.7.

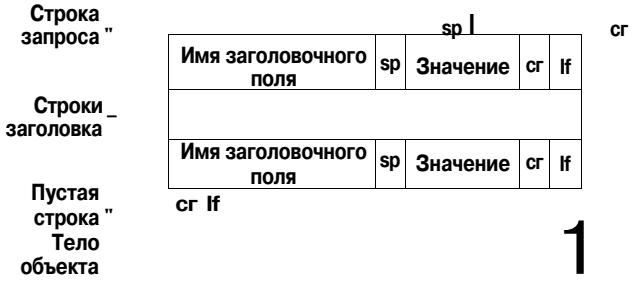


Рис. 2.7. Общий формат сообщения-ответа

Как можно убедиться, наш пример также полностью соответствует приведенному формату. Теперь скажем несколько слов о том, что означают поля кода состояния и информации о состоянии. Эти два поля взаимосвязаны и фактически отражают результат обработки запроса. Ниже приведены несколько наиболее часто встречающихся пар, содержащих код состояния и информацию об этом состоянии.

200 OK:

Запрос успешно обработан, объект получен и включен в ответ.

301 Moved Permanently:

Объект был перемещен; новый URL-адрес указан в строке ответа Location:. Программа клиента автоматически выполнит запрос по новому адресу.

400 Bad Request:

Общая ошибка, вызванная невозможностью интерпретации запроса сервером.

404 Not Found:

Запрашиваемый документ не найден на сервере.

505 HTTP Version Not Supported:

Указанная в запросе версия HTTP не поддерживается сервером.

Мы рекомендуем вам на практике ознакомиться с ответными HTTP-сообщениями. Для этого организуйте удаленный доступ к какому-либо серверу с помощью программы Telnet, а затем введите однострочный запрос объекта, находящегося на этом сервере. Например, для UNIX-машины это может выглядеть следующим образом:

```
telnet www.eurecom.fr 80
GET /-ross/index.htnil HTTP/1.0
```

После ввода второй строки следует дважды нажать клавишу ввода. В результате будет установлено TCP-соединение с портом **80** хоста **www.eurecom.fr** и отправлен запрос в виде команды **GET.** На вашем экране появится базовый HTML-файл домашней web-страницы профессора Росса. Если вам не нужно содержимое файла, достаточно заменить во второй строке слово **GET** словом **HEAD.** Теперь замените путь **/-ross/index.html** на **/-ross/banana.html** и посмотрите, какой ответ вы получите в этом случае.

В этом разделе мы изучили несколько типичных строк заголовка, которые могут использоваться в запросах и ответах протокола HTTP. В спецификациях HTTP (особенно **1.1)** содержатся описания гораздо большего количества строк заголовка, используемых браузерами, web-серверами и сетевыми кэш-серверами. Мы столкнемся с несколькими новыми строками в процессе изучения web-кэширования в конце этой главы; тем не менее за подробной информацией о HTTP/**1.1** следует обратиться к **[281,300].** Замечательным введением в проблематику web является **[562].**

Каким образом браузер определяет строки заголовка, которые нужно включить в запрос? Каким образом сервер определяет строки заголовка, которые нужно включить в ответ? Для браузера включаемые строки зависят от фирмы-разработчика, версии продукта (например, браузер, разработанный для спецификации HTTP/**1.0,** не сможет генерировать строки, характерные для спецификации HTTP/**1.1),** пользовательских настроек (например, языка), наличия на компьютере версии (возможно, устаревшей) запрашиваемого объекта. Аналогичная ситуация характерна для серверов: существуют различные программные продукты, их версии, настройки, совместно влияющие на включаемые строки заголовка.

Взаимодействие пользователя с сервером

Мы выяснили, что HTTP-сервер не запоминает информацию о состоянии соединения. Это упрощает разработку сервера и позволяет достичь значительной производительности за счет одновременного обслуживания сотен TCP-соединений. Тем не менее возможность распознавания пользователей сервером является весьма желательной. Причиной этому может служить необходимость разграничения прав доступа к информации, находящейся на сервере, либо предоставление каждому пользователю собственного набора информационных услуг. Протокол HTTP предусматривает два механизма идентификации пользователей: авторизацию и объекты cookie.

Авторизация

Вероятно, вам приходилось сталкиваться с ситуациями, когда сервер предлагал вам ввести имя пользователя и пароль для доступа к своему информационному пространству. Подобный механизм доступа называется *авторизацией*. Запрос и получение авторизации в HTTP зачастую производятся с помощью особых заголовков и кодов состояния. Рассмотрим следующий пример. Пусть клиент инициирует запрос объекта, причем объект находится на сервере, требующем авторизации.

Сначала клиент формирует обычный запрос, не содержащий специальных строк. Сервер возвращает ответ с пустым телом и кодом состояния 401 Authorization Required. В специальной строке WWW-Authenticate: заголовка содержится описание метода проведения авторизации; как правило, это описание указывает на необходимость ввести имя пользователя и пароль.

Получив подобное сообщение, клиент запрашивает имя пользователя и пароль. По окончании ввода генерируется новый запрос, содержащий строку Authorization: с введенными пользователем данными. Получив первый объект, клиент продолжает отсылать серверу имя пользователя и пароль для всех остальных запрашиваемых объектов. Обычно это происходит до тех пор, пока работа клиента не будет завершена пользователем. Таким образом, в течение всего сеанса с клиентом имя пользователя и пароль кэшируются и не запрашиваются у пользователя многократно.

Как мы увидим в главе 7, механизм HTTP-авторизации не позволяет организовать надежную защиту данных сервера от несанкционированного доступа. Там же мы изучим более сложные и безопасные схемы авторизации.

Cookie

Объекты cookie являются альтернативным авторизации средством идентификации пользователей. Описание cookie находится в документе RFC 2109. Обычно объекты cookie находят применение в Интернет-порталах (например, Yahoo!), электронной коммерции (например, Amazon) и рекламе (например, DoubleClick).

Технология cookie подразумевает наличие четырех основных компонентов:

- заголовочной cookie-строки в ответном сообщении сервера;
- заголовочной cookie-строки в запросе клиента;

- cookie-файла, находящегося на стороне клиента и обрабатываемого браузером;
- удаленной базы данных, расположенной на web-сайте.

Рассмотрим типичный пример использования объекта cookie. Предположим, пользователь применяет для доступа в web один и тот же браузер (пусть это будет Internet Explorer) и впервые оказывается на сайте, предоставляющем услуги электронной коммерции. Доступ к сайту осуществляется при помощи технологии cookie. При первом доступе сервер генерирует уникальный идентификационный номер для пользователя, создает в своей базе данных запись с индексом, равным идентификационному номеру, и отсылает клиенту пользователя ответное сообщение, включающее специальную строку Set-cookie: заголовка, содержащую идентификационный номер. Пример такой строки:

```
Set-cookie: 1678453
```

Получив ответ, браузер анализирует его и добавляет строку Set-cookie: в cookie-файл. Этот файл содержит имена хостов и соответствующие идентификационные номера серверов. Каждый раз при формировании запроса к web-сайту браузер обращается к cookie-файлу, извлекает из него нужный идентификационный номер, включает его в запрос и отсылает серверу. В каждом таком запросе содержится строка заголовка вида:

```
cookie: 1678453
```

Таким образом, сервер может собирать информацию о деятельности у себя пользователя: времени доступа, посещенных страницах и т. п. Это, в свою очередь, позволяет серверу организовать «карту покупателя» со списком сделанных во время сеанса покупок и дает возможность сразу оплатить их.

При повторных входах на сайт идентификационный номер снова будет передан серверу. Сервер, располагающий информацией о предыдущих покупках пользователя, может на ее основе составить рекомендации о новых покупках. Зачастую подобные коммерческие сайты позволяют пользователям пройти регистрацию, указав свои имя, фамилию, почтовый адрес, номер кредитной карты, адрес электронного почтового ящика и др. Введенные данные заносятся в базу данных сервера. Именно с помощью описанного механизма осуществляется «покупка одним щелчком мыши» — потребность ввода личных данных отпадает.

Итак, объекты cookie представляют собой средство идентификации пользователей. При первом сеансе доступа пользователь вводит какой-либо идентификационный параметр (например, свое имя), а сервер в ответном сообщении отсылает пользователю заголовочную cookie-строку, идентифицируя его. Кроме того, технология cookie позволяет создать подобие дополнительного «сеансового уровня» для протокола HTTP, не запоминающего информацию о соединении. К примеру, когда пользователь подключается к web-приложению электронной почты, браузер отсылает cookie-информацию серверу, позволяющую идентифицировать пользователя во время сеанса.

Несмотря на то что объекты cookie позволяют упростить процесс покупок для пользователя, они, как и приведенная выше схема авторизации, весьма ненадежны с точки зрения обеспечения конфиденциальности информации. Как мы убедились, процедура регистрации приводит к выдаче пользователем данных личного характера, ко-

торые при использовании небезопасного cookie-доступа могут стать известны другим людям. Кроме того, объекты cookie могут применяться для сбора информации о поведении пользователя на множестве web-сайтов. Web-страницы, содержащие баннерную рекламу, прибегают к cookie для получения объектов баннерной рекламы (как правило, рисунков в формате JPEG и GIF) с серверов рекламного агентства. Каждый из запросов к серверу рекламного агентства может содержать объект cookie, генерируемый сайтом рекламного агентства. Поскольку агентства обычно связаны с множеством web-страниц, это позволяет им отслеживать пути отдельных пользователей в web и анализировать собранную информацию.

Мы рекомендуем читателю обратиться к [353], где можно ознакомиться с углубленным, однако вполне доступным для понимания материалом о технологии cookie. Весьма интересные рассуждения на тему cookie имеются также в [96].

Метод GET с условием

Web-кэширование, то есть сохранение уже загруженных объектов, способно ощутимо сократить время загрузки web-страниц и сетевой трафик. Кэширование может осуществляться как клиентом (браузером пользователя), так^и промежуточным сетевым кэш-сервером. Последний будет рассмотрен в конце этой главы, а пока обратимся к кэшированию на клиенте.

Несмотря на то что кэширование способно снизить задержку на доставку объектов, необходимых для открытия документов, оно порождает новую проблему: содержимое кэшируемого объекта постоянно устаревает. Объект, находящийся на web-сервере, может быть изменен, после чего он перестает соответствовать уже загруженному объекту. К счастью, протокол HTTP располагает средством, позволяющим при необходимости обновлять кэшированные объекты; это средство представляет собой *метод GET с условием*. В HTTP существует так называемый условный GET-запрос, который использует метод GET и строку If-Modified-Since: заголовка.

Для того чтобы продемонстрировать описанный механизм в действии, рассмотрим пример. Пусть сначала браузер запрашивает некэшированный объект с web-сервера:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
```

Затем происходит передача ответного сообщения сервера с требуемым объектом:

```
HTTP/1.0 200 OK
Date: Wed, 12 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24
Content-Type: image/gif

(data data data data data ...)
```

Клиент отображает объект на экране и сохраняет его в своем локальном кэше. Кроме того, вместе с объектом сохраняется время его последнего изменения. Пусть через неделю пользователь снова соединяется с тем же сервером, при этом объект остается в кэше. Поскольку за прошедшую неделю объект на сервере мог быть изменен, браузер формирует запрос, отправляемый методом GET с условием:

```
GET /fruit/kiwi.gif HTTP/1.0
User-agent: Mozilla/4.0
If-modified-since: Mon. 22 Jun 1998 09:23:24
```

Обратите внимание на то, что содержимое строки If-modified-since: совпадает с содержимым строки Last-Modified: первого запроса. Запрос является указанием серверу осуществить пересылку объекта в случае, если с момента предыдущей пересылки последний был изменен. Если изменения не произошло, будет получен следующий ответ:

```
HTTP/1.0 304 Not Modified
Date: Wed. 19 Aug 1998 15:39:29
Server: Apache/1.3.0 (Unix)

(пустое тело)
```

Как мы видим, несмотря на отсутствие изменений объекта, сервер отсылает клиенту ответ, однако тело сообщения остается пустым. Новая посылка идентичного объекта привела бы к бесполезной трате временных и сетевых ресурсов, особенно ощутимой при большом размере объекта. Обратите внимание на то, что в ответе содержится код состояния 304 Not Modified, означающий, что клиент может использовать кэшированную версию объекта.

Область применения HTTP

Во всех приведенных примерах мы использовали объекты, относящиеся к web-страницам: базовые HTML-файлы, изображения в формате GIF, JPEG, Java-апплеты и т. д. Мы представляли протокол HTTP в контексте web для того, чтобы обеспечить наглядность и простоту наших описаний, поскольку многие читатели наверняка хорошо знакомы с web-навигацией. Однако было бы большим упущением не отметить, что протокол HTTP также используется для передачи информации другого характера.

Одним из применений протокола HTTP является передача файлов формата XML между вычислительными машинами; при этом наличие пользователя или браузера на этих машинах не обязательно. XML-файлы позволяют структурировать банковскую информацию, поэтому нередко возникает необходимость обмена такими файлами между банками. (Описание формата XML лежит за пределами темы данной книги. Мы скажем лишь, что XML-документ состоит из структурированных данных и смысловой части; в отличие от HTML, XML не имеет отношения к форматированию выводимых данных.) Кроме того, HTTP используется для передачи файлов'VoiceXML, WML (языка разметки WAP) и прочих XML-документов. А также, как мы увидим в конце главы, по протоколу HTTP организуется одноранговый файловый обмен, а в главе 6 будет рассказано о применении HTTP для передачи потокового аудио и видео.

Передача файлов по протоколу FTP

FTP-сеанс представляет собой обмен файлами, находящимися на двух хостах — локальном и удаленном. Для получения доступа к удаленному хосту пользователю необходимо ввести свои имя и пароль. После получения доступа пользователь может осуществлять передачу файлов как с удаленного хоста на локальный, так и

наоборот. Как показано на рис. 2.8, пользователь взаимодействует с FTP при помощи пользовательского агента FTP. Сначала пользователь указывает имя удаленного хоста FTP-клиенту для того, чтобы последний установил TCP-соединение с сервером, а затем вводит свои имя и пароль, пересылаемые серверу при помощи FTP-команд. После распознавания пользователя сервером начинается процесс передачи файлов в нужном направлении.

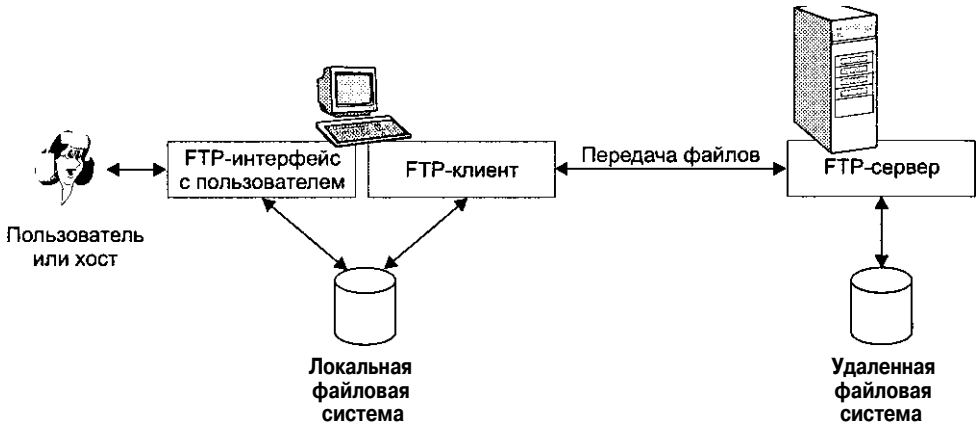


Рис. 2.8. FTP осуществляет передачу файлов между локальной и удаленной файловыми системами

HTTP и FTP являются протоколами передачи файлов и имеют много общего; например, в качестве протокола транспортного уровня они оба используют TCP. Тем не менее между HTTP и FTP существуют и принципиальные различия. Протокол FTP использует два параллельных TCP-соединения: *управляющее соединение* и *соединение данных*. Управляющее соединение служит для пересылки управляющей информации между двумя хостами: имени пользователя и пароля, команд смены текущего удаленного каталога, передачи и запроса файлов. Соединение данных предназначено для передачи самих файлов. Поскольку управляющее соединение отделено от соединения данных, говорят, что передача управляющей информации осуществляется *вне полосы* (out-of-band). В главе 6 мы познакомимся с протоколом RTSP, предназначенным для контроля передачи данных потокового мультимедиа и также использующим механизм передачи управляющей информации вне полосы. В отличие от FTP, протокол HTTP через единственное TCP-соединение осуществляет передачу и файлов, и команд (строк заголовков для запросов и ответов). Поэтому говорят, что HTTP передает свою управляющую информацию *внутри полосы* (in-band). Другим примером протокола с передачей управляющей информации внутри полосы является SMTP, характерный для приложений электронной почты. Мы рассмотрим протокол SMTP в следующем разделе. На рис. 2.9 приведена иллюстрация двух соединений протокола FTP.

FTP-сеанс начинается с установления управляющего TCP-соединения между клиентом и удаленным хостом (сервером) через порт с номером 21. По этому соедине-

нию осуществляется передача имени пользователя и пароля, а также команд смены текущего каталога и обмена файлами. Когда сервер получает команду передачи или приема файла, он устанавливает с клиентом TCP-соединение данных, затем осуществляет файловый обмен и закрывает соединение. Каждое соединение позволяет передать только один файл; таким образом, множественный обмен вызывает необходимость многократной установки соединения данных. При этом управляющее соединение остается открытым в течение всего сеанса. Учитывая введенную терминологию, соединение данных можно отнести к непостоянным соединениям.

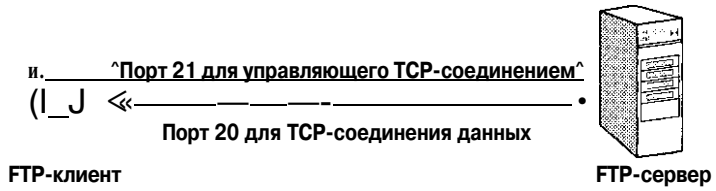


Рис. 2.9. Управляющее соединение и соединение данных

Во время FTP-сеанса серверу необходимо иметь информацию о пользователе. Как правило, управляющее соединение связано со специальной учетной записью пользователя. Кроме того, сервер должен следить за текущим каталогом, в котором работает пользователь. Необходимость затрат ресурсов на хранение информации приводит к значительному снижению числа FTP-сеансов, одновременно поддерживаемых сервером. В этом заключается недостаток протокола FTP по сравнению с HTTP: как вы помните, HTTP не запоминает состояние соединения.

Мы закончим текущий раздел кратким описанием наиболее часто используемых команд протокола FTP. Команды, посылаемые клиентом серверу, и ответы сервера передаются через управляющее соединение в кодировке ASCII для 7-разрядных символов. Таким образом, команды FTP, как и HTTP, могут быть легко прочитаны человеком. Для разделения команд используются пары символов перехода на новую строку (возврат каретки и перевод строки). Имя команды представляет собой четыре символа в верхнем регистре, за которым могут следовать параметры. Ниже перечислены несколько FTP-команд.

USER username:

Передача серверу имени пользователя.

PASS password:

Передача серверу пароля.

LIST:

Запрос к серверу на передачу имен всех файлов, находящихся в текущем каталоге. Передача списка файлов происходит с использованием непостоянного соединения данных.

RETR filename:

Запрос к серверу на передачу файла с указанным именем. Сервер в ответ должен установить соединение данных и начать передачу файла пользователю.

STOR filename:

Передача файла от пользователя в текущий каталог удаленного хоста.

Как правило, одна команда, вводимая пользователем, вызывает генерацию и передачу одной FTP-команды. Ответ сервера представляет собой трехзначное число, иногда сопровождаемое небольшим текстовым пояснением. Такой формат очень напоминает строку состояния в ответе для протокола HTTP; более того, разработчики HTTP намеренно заимствовали этот формат из FTP. Ниже приведено несколько типичных ответов протокола FTP.

331 Username OK. password required
125 Data connection already open; transfer starting
425 Can't open data connection
452 *Error* writing file

Читателям, заинтересовавшимся набором команд и ответов, использующихся в FTP, рекомендуем обратиться к документу RFC 959.

Электронная почта

Электронная почта появилась едва ли не раньше Интернета. В эпоху зарождения Интернет-технологий она была самым популярным из существовавших приложений, а за годы развития претерпела множество изменений и продолжает меняться до сих пор.

Как и обычная почта, электронная почта является асинхронным средством связи: люди посылают друг другу сообщения в любое удобное для них время без предварительной договоренности с адресатами. Преимуществами электронной почты перед обычной являются высокая скорость доставки, простота использования и низкая стоимость обслуживания. С помощью списка рассылки с адресами отправитель может разослать одно и то же письмо сотням получателей одновременно. Кроме того, современная электронная почта позволяет вместе с письмами пересылать гиперссылки, текст в формате HTML, изображения, аудио- и видеофайлы, Java-апплеты и т. д. В этом разделе мы рассмотрим протоколы прикладного уровня, составляющие основу электронной почты. Однако перед тем как углубляться в детали, взглянем на структуру почтовой службы и ее ключевые компоненты.

На рис. 2.10 представлена структура системы электронной почты. В этой структуре можно выделить три ключевых компонента: *агенты пользователя, почтовые серверы* и *протокол SMTP*. Мы рассмотрим каждый из компонентов на примере двух пользователей, Алисы и Боба, общающихся по электронной почте. Агенты пользователя позволяют читать, отвечать, пересылать, создавать и сохранять электронные письма; их часто называют программами для чтения почты, хотя мы постараемся избегать этого термина в нашей книге. Когда Алиса создает новое письмо Бобу, ее агент отсылает письмо почтовому серверу, где письмо попадает в очередь исходящих сообщений сервера. Когда Боб захочет прочитать письмо, его агент соединится с почтовым сервером и доставит письмо на персональный компьютер Боба. Во второй половине 1990-х большое распространение получили агенты с графическим интерфейсом пользователя (Graphical User Interface, GUI), позволяющие

читать и создавать мультимедийные сообщения. В настоящее время наиболее популярными агентами с интерфейсом GUI являются Eudora, Microsoft Outlook и Netscape's Messenger. Кроме того, существует множество агентов с текстовыми интерфейсами; к ним следует отнести mail, pine и elm.

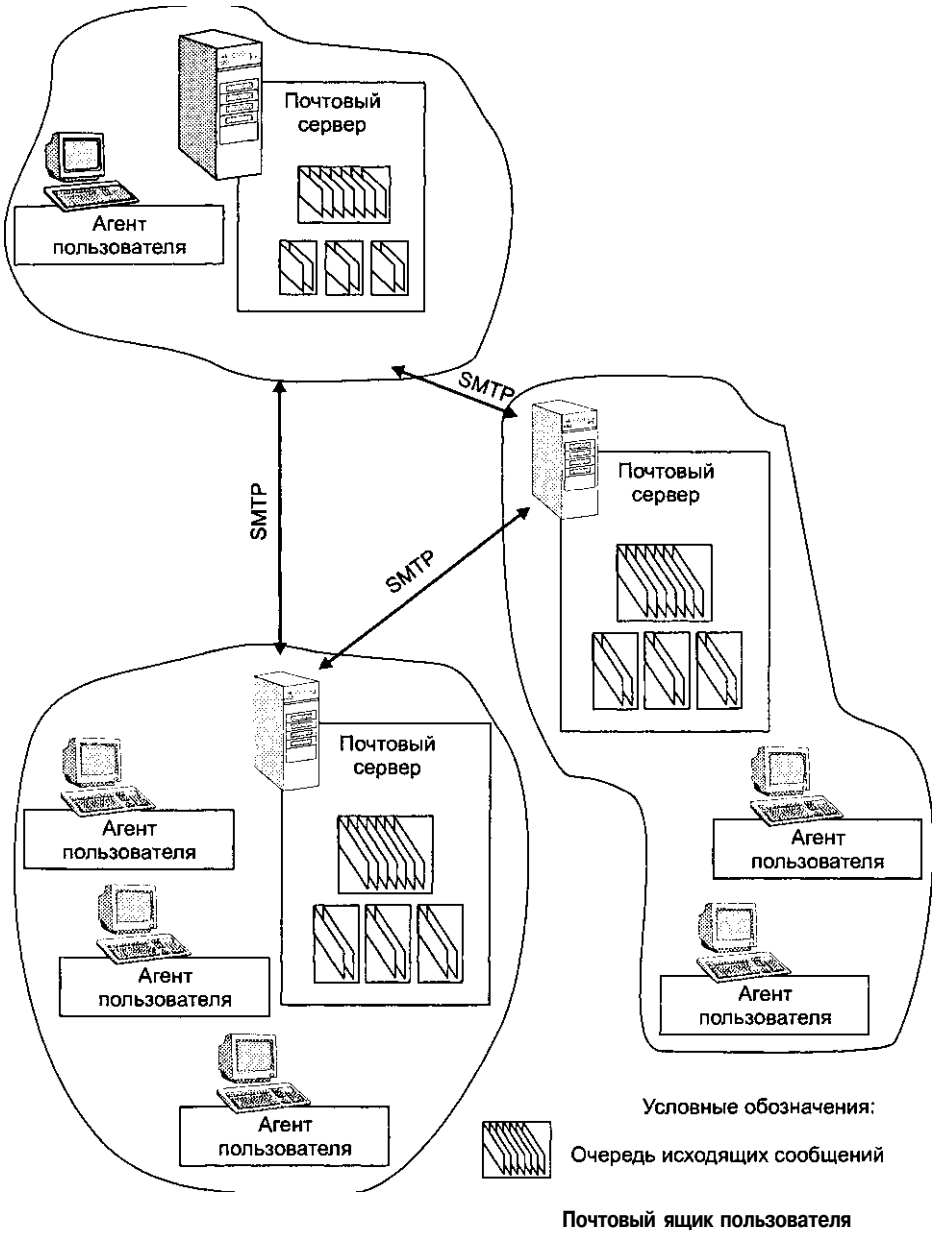
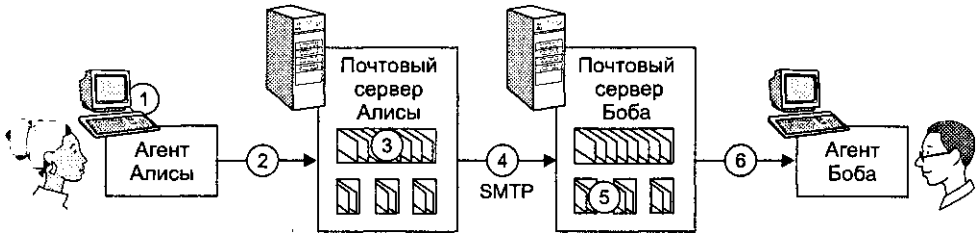


Рис. 2.10. Структура электронной почты Интернета



Условные обозначения:

Очередь сообщений **S** **Почтовый ящик пользователя**

Рис. 2.11. Алиса посылает сообщение Бобу

Теперь рассмотрим подробнее, каким образом осуществляется передача сообщения между почтовыми серверами. Любопытно отметить, что протокол SMTP по своей сути напоминает непосредственное общение между двумя людьми. Итак, сначала SMTP-клиент пытается установить TCP-соединение с портом 25 сервера; если сервер не отвечает, попытка повторяется позднее. После того как соединение установлено, клиент и сервер обмениваются рукопожатиями на прикладном уровне по аналогии с людьми, которые представляются друг другу перед тем, как начать общение. В ходе процедуры рукопожатия клиент определяет адреса почтовых ящиков отправителя и получателя сообщения. По завершении рукопожатия начинается процесс передачи сообщения от клиента к серверу. Поскольку передача осуществляется с помощью протокола TCP, гарантируется надежная доставка данных. Если в очереди клиента имеются другие сообщения, предназначенные этому же серверу, все они пересылаются последовательно через одно TCP-соединение. После передачи всех сообщений клиент закрывает соединение с сервером.

Рассмотрим пример обмена сообщениями между SMTP-клиентом (C) и SMTP-сервером (S). Хост клиента имеет имя crepes.fr, а хост сервера — имя hamburger.edu. Строки, помеченные литерой C, передаются клиентом в свой сокет в той же кодировке ASCII, в которой они приведены ниже; то же самое касается строк, помеченных литерой S и относящихся к серверу. Итак, после установления TCP-соединения обмен происходит следующим образом:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

В приведенном примере клиент послал почтовому серверу **hamburger.edu** сообщение «Do you like ketchup? How about pickles?» с почтового сервера **crepes.fr.** Клиент использовал пять различных команд: **HELO, MAIL FROM, RCPT TO, DATA** и **QUIT.** Смысл этих команд вполне понятен. Кроме того, клиент использовал символ точки, указывающий на конец сообщения. Если пару символов перехода на следующую строку (возврат каретки и перевод строки) обозначить как **CR** и **LF** соответственно, то каждое сообщение оканчивается сочетанием **CRLF.CRLF.** Сервер посылает ответы на все команды клиента; ответ включает код и (необязательно) описание на английском языке. Обратите внимание на то, что протокол SMTP поддерживает постоянные соединения: если клиенту необходимо отправить несколько сообщений подряд, все сообщения передаются через одно TCP-соединение. Передача каждого нового сообщения начинается с команды **MAIL FROM: crepes.fr,** а заканчивается одиночным символом точки. После того как все сообщения посланы, клиент генерирует команду **QUIT.**

Рекомендуем вам самостоятельно «пообщаться» с SMTP-сервером, используя программу Telnet. Для этого введите следующую строку:

```
telnet serverName 25
```

Здесь **serverName** — имя удаленного почтового сервера. При этом будет установлено TCP-соединение вашего хоста с почтовым сервером. Вероятнее всего, сервер сразу же ответит сообщением с кодом 220. После этого введите команды **HELO, MAIL FROM, RCPT TO, DATA, CRLF.CRLF** и **QUIT** в нужном порядке. Подобным образом вы можете, не прибегая к агенту пользователя, посылать сообщения электронной почты своим знакомым.

Сравнение SMTP и HTTP

Теперь настало время сравнить два важных Интернет-протокола: HTTP и SMTP. Оба они предназначены для передачи файлов между хостами, при этом HTTP организует передачу объектов между web-клиентом (который обычно представляет собой браузер) и web-сервером, а SMTP — передачу электронных сообщений между двумя почтовыми серверами. Как HTTP, так и SMTP используют постоянные соединения. Тем не менее, наряду с описанными сходствами, протоколы обладают и различиями.

Во-первых, HTTP представляет собой *протокол получения* (pull protocol), то есть некто загружает на web-сервер нужную информацию, которую пользователи с помощью протокола HTTP *получают* с сервера в удобное для себя время. Как правило, TCP-соединение устанавливается компьютером, инициирующим получение файла. SMTP, напротив, является *протоколом отправки* (push protocol), то есть передающий почтовый сервер *отправляет* файл принимающему почтовому серверу. Как правило, TCP-соединение устанавливается компьютером, инициирующим отправку файла.

Во-вторых, как уже упоминалось ранее, SMTP требует 7-разрядной кодировки ASCII для символов в заголовке и теле каждого сообщения. Если сообщение содержит символы расширенной кодировки ASCII (например, символы национальных алфавитов) или бинарные данные, требуется преобразование таких данных

в 7-разрядную кодировку ASCII. Протокол HTTP не накладывает подобных ограничений на сообщения.

В-третьих, протоколы SMTP и HTTP поддерживают разные способы обработки документов, содержащих текстовую и графическую (или мультимедийную) информацию. Как упоминалось в разделе «Web и HTTP», протокол HTTP пересылает каждый объект в отдельном ответном сообщении; SMTP, напротив, помещает все объекты в одно сообщение.

Форматы сообщений электронной почты и MIME

Когда Алиса пишет обычное электронное письмо Бобу, она может снабдить его различной дополнительной информацией: почтовым адресом Боба, своим почтовым адресом, датой создания письма. Подобная информация содержится в заголовке письма, предшествующем его телу. Заголовок представляет собой совокупность строк, которые описаны в документе RFC 822. Заголовок сообщения отделяется от тела пустой строкой **(CRLF).** RFC 822 определяет формат всех строк заголовка сообщения, а также их семантическую интерпретацию. Как и в протоколе HTTP, каждая строка заголовка содержит текст в виде символов ASCII, включающий ключевое слово и значение, разделенные знаком двоеточия. Некоторые ключевые слова являются обязательными, другие — не обязательными. Примерами обязательных ключевых слов являются **From:** и To:, а не обязательных — **Subject:.** Обратите внимание на то, что строки заголовка отличаются от SMTP-команд, рассмотренных ранее в этом разделе. Команды представляют собой часть процедуры рукопожатия, а строки заголовка — часть передаваемого сообщения.

Типичный заголовок сообщения выглядит следующим образом:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

После заголовка следует пустая строка, а за ней начинается тело сообщения в кодировке ASCII. Настоятельно рекомендуем вам самостоятельно послать почтовому серверу сообщение, содержащее несколько строк заголовка, включая строку **Subject:.** Для этого с помощью программы Telnet следует установить TCP-соединение с нужным сервером, введя следующую строку:

```
telnet serverName 25
```

MIME-расширение для кодировки, отличной от ASCII

Если приведенные выше заголовки подходят для сообщений, содержащих текст в кодировке ASCII, то их содержимого недостаточно для сообщений с аудио-, видео- и прочей информацией, формат которой не соответствует ASCII. Это требует включения в сообщение специальных заголовков, а следовательно, расширения стандарта RFC 822. Такое расширение описано в документах RFC 2045 и 2046 и носит название многоцелевых расширений почты Интернета (Multipurpose Internet Mail Extensions, MIME).

Двумя наиболее важными MIME-заголовками, предназначенными для поддержки мультимедиа, являются **Content-Type:** и **Content-Transfer-Encoding:.** Заголовок

Content-Type: позволяет агенту получателя произвести соответствующую обработку данных сообщения. Так, например, если сообщение содержит изображение в формате JPEG, агент получателя вызовет процедуру декомпрессии файлов JPEG. Для того чтобы понять смысл второго заголовка, Content-Transfer-Encoding:, вспомните о том, что все данные в кодировке, отличной от ASCII, перед передачей по протоколу SMTP должны быть преобразованы в кодировку ASCII. Заголовок Content-Transfer-Encoding: указывает адресату на то, что было произведено преобразование (кодирование) исходной кодировки символов в кодировку ASCII, а также на тип этого кодирования. Таким образом, агент получателя, распознав заголовок Content-Transfer-Encoding:, сможет произвести декодирование сообщения для приведения данных в исходную кодировку, а затем, распознав заголовок Transfer-Encoding:, обработать декодированные данные.

Рассмотрим конкретный пример. Пусть Алиса хочет переслать Бобу электронное письмо, содержащее JPEG-изображение. Для этого она вызывает свой агент, вводит адрес почтового ящика Боба, указывает тему сообщения и вставляет изображение в тело сообщения (в зависимости от используемого агента вставка файла может называться «присоединением»). После команды отправки агент генерирует MIME-сообщение, выглядящее приблизительно так:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

(base64 encoded data. . . . .

. . . . .base64 encoded data)
```

Из приведенного сообщения можно узнать о том, что агент пользователя Алисы кодировал JPEG-изображение методом base64. Этот метод стандартизирован документом RFC 2045 как способ преобразования в 7-разрядную кодировку ASCII. Другим часто используемым методом кодирования является преобразование 8-разрядных данных в кодировке ASCII (обычно символов национальных алфавитов) в 7-разрядные.

Когда Боб станет читать письмо Алисы, его агент сначала обнаружит строку Content-Transfer-Encoding: base64 заголовка и декодирует тело сообщения методом base64. Затем агент увидит строку Content-Type: image/jpeg и произведет JPEG-декомпрессию полученных данных. Строка MIME-Version: 1.0, как нетрудно догадаться, идентифицирует номер версии MIME. При отсутствии этой строки сообщение будет обработано как стандартное, соответствующее формату RFC 822/SMTP. Как правило, заголовок сообщения отделяется от тела пустой строкой.

Рассмотрим немного подробнее строку Content-Type: заголовка. Согласно спецификации MIME, указанной в RFC 2046, формат строки имеет следующий вид:

```
Content-Type: type/subtype: parameters
```

Здесь parameters — это необязательные параметры. Согласно спецификации, строка Content-Type: используется для указания типа данных, передаваемых в сообще-

нии, и состоит из имен типа и подтипа. Кроме того, в строке могут присутствовать параметры, предназначенные для уточненной информации о подтипе и не оказывающие значимого влияния на интерпретацию данных. Разумеется, для каждого подтипа определяется собственный набор параметров. Разработка MIME велась с расчетом на будущую расширяемость, и не исключено, что скоро число возможных пар типов и подтипов значительно возрастет. Для того чтобы как-то упорядочить разработку новых типов и подтипов, MIME предусматривает необходимость регистрации в IANA (Internet Assigned Numbers Authority — уполномоченная организация по назначению номеров Интернета). Регистрационный процесс описан в документе RFC 2048.

В настоящее время выделено семь основных типов данных. Для каждого типа данных существует ряд подтипов, число которых с каждым годом стремительно растет. Ниже приведены описания трех из семи типов.

- text. Этот тип указывает агенту получателя на то, что тело сообщения содержит текстовую информацию. Очень часто встречающейся парой тип/подтип является text/plain. Подтип plain означает простой текст, то есть текст не содержит команд или директив форматирования. Такой текст должен быть отображен «как есть»; никакого специального программного обеспечения для этого не требуется. Единственным условием корректности отображения текста является поддержка используемой кодировки символов. Если вы взглянете на MIME-заголовки сообщений, хранящихся в вашем почтовом ящике, то почти наверняка найдете строки следующего содержания: text/plain; charset="IS0-8859-l". Параметры указывают на название таблицы символов, использовавшейся при создании текста. Другой часто встречающейся парой является text/html. Подтип html указывает на то, что программа чтения почты должна интерпретировать теги, включенные в сообщение. Это позволяет отобразить сообщение в виде web-страницы, содержащей различные шрифты, гиперссылки, апплеты и т. д.
- image. Этот тип указывает на то, что информация, находящаяся в теле сообщения, является изображением. Двумя наиболее распространенными парами тип/подтип для изображений являются image/gif и image/jpeg. Распознавая каждый из этих подтипов, агент пользователя применяет соответствующий метод декомпрессии изображения.
- application. Это тип для всех данных, которые нельзя отнести к другим шести типам. Как правило, сюда попадают данные, предназначенные для обработки различными прикладными программами. Так, например, если включить в сообщение документ Microsoft Word, агент отправителя присвоит ему пару application/msword. При обработке сообщения агент получателя вызовет приложение Microsoft Word и передаст ему декодированные данные.

Существует еще один весьма важный MIME-тип, заслуживающий отдельного рассмотрения и называющийся multipart. Подобно тому как web-страница может включать в себя различные типы данных (текст, изображения, апплеты и т. д.), тело сообщения также может состоять из разнородной информации. Вспомните о том, что протокол HTTP посылает каждый объект в отдельном ответном сообщении; последовательность ответных сообщений может передаваться через одно или не-

сколько TCP-соединений в зависимости от типа соединения (постоянного или непостоянного). Протокол SMTP, напротив, помещает все объекты (составные части) в тело одного сообщения. В случаях, когда сообщение содержит более одного объекта (например, несколько изображений, текст и изображения и т. п.), ему присваивается пара тип/подтип multipart/mixed. Для обработки тела такого сообщения агенту пользователя необходима информация о том, где находится начало и окончание каждого из объектов, какими способами закодированы объекты и каковы типы и подтипы данных, содержащихся в объектах. Структура составного сообщения включает специальные разделительные символы, вставляемые между объектами, и заголовки Content-Type: и Content-Transfer-Encoding: перед началом данных каждого из объектов.

Для того чтобы лучше понять принцип организации составного сообщения, рассмотрим следующий пример. Пусть Алиса намеревается послать Бобу письмо, содержащее два фрагмента ASCII-текста, а также JPEG-изображение, находящееся между этими фрагментами. При помощи своего агента Алиса вводит первый текстовый фрагмент, затем присоединяет изображение и вводит второй текстовый фрагмент. После команды отправки агент генерирует сообщение приблизительно следующего содержания:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe with commentary
MIME-Version: 1.0
Content-Type: multipart/mixed; Boundary=StartOfNextPart

--StartOfNextPart
Dear Bob.
Please find a picture of an absolutely scrumptious crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data. . . . .

. . . . .base64 encoded data
--StartOfNextPart
Let me know if you would like the recipe.
```

Как можно видеть, первая строка Content-Type: заголовка указывает на последовательность символов, применяющуюся для разделения разнородных частей сообщения. Разделяющая последовательность всегда начинается с символов

Принимаемые сообщения

Ниже мы расскажем еще об одном классе строк заголовка, которые вставляются в сообщение почтовым сервером получателя. После получения сообщения с заголовками стандартов RFC 822 и MIME сервер добавляет в начало заголовка строку Received:, содержащую адреса отправителя (From), получателя (To) и время получения сообщения сервером. Для рассматриваемого примера сообщение, полученное Бобом, будет выглядеть следующим образом:

```
Received: from crepes.fr by hamburger.edu: 12 Oct 98
  15:27:39 GMT
```

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data . . . . .

. . . . .base64 encoded data
```

Практически каждый человек, хотя бы однажды использовавший электронную почту, видел строку Received: в начале сообщений (как правило, содержимое этой строки выводится при отображении сообщения на экране и при получении его печатной копии). Возможно, вы также видели сообщения, содержащие в заголовке несколько строк Received:, а иногда и более замысловатую строку Return-Path:. Это объясняется тем, что в некоторых случаях сообщение проходит через несколько SMTP-серверов. К примеру, если бы Боб отослал полученное от Алисы сообщение со своего почтового сервера hamburger.edu серверу sushi.jp, начало заголовка такого сообщения выглядело бы так:

```
Received: from hamburger.edu by sushi.jp: 3 Jul 01
  15:30:01 GMT
Received: from crepes.fr by hamburger.edu; 3 Jul 01
  15:17:39 GMT
```

Эта пара строк заголовка позволяет агенту конечного получателя проследить путь, который письмо прошло с момента его первой отправки.

Протоколы доступа к электронной почте

После того как письмо Алисы попадает на почтовый сервер Боба, оно помещается в почтовый ящик Боба. Во всех предыдущих примерах мы неявно предполагали, что Боб читает письма, входя на свой почтовый сервер и запуская программу чтения почты непосредственно на сервере. Действительно, до середины 1990-х годов такая схема доступа к электронным сообщениям была самой распространенной. В последние годы более типична ситуация, когда пользователь просматривает сообщения с помощью агента, выполняющегося на его вычислительной машине (офисном персональном компьютере, компьютере семейства Macintosh или цифровом органайзере). Это открывает пользователю доступ к набору удобных средств для работы с электронной почтой, в частности к средствам просмотра мультимедиа-сообщений и разнообразных вложений.

Если Боб имеет возможность читать электронные письма с помощью программы, выполняющейся на его домашнем компьютере, возникает резонный вопрос: почему бы не наделить его компьютер функциями почтового сервера? В этом случае почтовый сервер Алисы осуществлял бы прямое взаимодействие с компьютером Боба, и электронные сообщения попадали бы непосредственно к его агенту. Такой подход, увы, влечет за собой серьезные проблемы. Вспомним о том, что почтовый SMTP-сервер управляет почтовыми ящиками и при обмене почтой выполняет функции как серверной, так и клиентской сторон. Поэтому для

успешной работы нашей гипотетической почтовой системы компьютер Боба должен всегда оставаться включенным и иметь связь с Интернетом. Не удивительно, что такая схема совершенно непригодна для подавляющего большинства пользователей. На практике используется компромиссный вариант: пользователь просматривает электронную почту с помощью агента, находящегося на его персональном компьютере, однако прием входящих сообщений осуществляется почтовым сервером общего пользования, на котором расположен почтовый ящик пользователя. Ооычно почтовые ящики предоставляются Интернет-провайдерами.

Итак, пользователи обрабатывают электронные сообщения с помощью своих персональных компьютеров, используя почтовые серверы лишь для отправки и получения почты. Возникает вопрос: каким образом осуществляется взаимодействие между агентами пользователей и почтовыми серверами? Сначала рассмотрим, как письмо Алисы попадает на почтовый сервер Боба. Разумеется, прочитав предыдущий материал, вы можете уверенно заявить, что для этого используется протокол SMTP, и будете правы. Однако протокол SMTP описывает передачу сообщений между почтовыми серверами, а агент пользователя Алисы не имеет прямого соединения с почтовым сервером Боба. Дело обстоит так: сначала агент пользователя устанавливает SMTP-соединение с почтовым сервером Алисы и осуществляет передачу сообщения, а уже затем происходит соединение почтовых серверов Алисы и Боба, о котором шла речь ранее. Возникает вопрос: зачем использовать промежуточную передачу? Первой важной причиной является то, что агент пользователя Алисы не располагает эффективным механизмом реагирования на отсутствие ответов от почтового сервера Боба. Как вы помните, почтовый сервер Алисы предпринимает периодические попытки установления соединения с почтовым сервером Боба до тех пор, пока одна из попыток не окажется удачной. В документах RFC содержатся описания способов передачи сообщений между несколькими SMTP-серверами с помощью SMTP-команд.

Теперь остается рассмотреть, каким образом агент пользователя Боба получает сообщения, находящиеся в его почтовом ящике. Вспомним о том, что SMTP является протоколом отправки, а операция извлечения и доставки сообщений, очевидно, требует применения протокола получения. Таким образом, мы приходим к необходимости создания специального протокола получения электронной почты, находящейся в почтовом ящике сервера. Существует несколько таких протоколов, наиболее распространенными из которых являются POP3 (Post Office Protocol Version 3 — протокол почтового отделения, версия 3), IMAP (Internet Mail Access Protocol — протокол доступа к почте Интернета) и HTTP.

На рис. 2.12 представлена схема, иллюстрирующая использование различных протоколов в системе электронной почты Интернета. Как мы видим, протокол SMTP передает сообщения между почтовыми серверами отправителя и получателя, а также между агентом отправителя и почтовым сервером отправителя. От почтового сервера получателя агенту получателя сообщения доставляются по протоколу POP3.

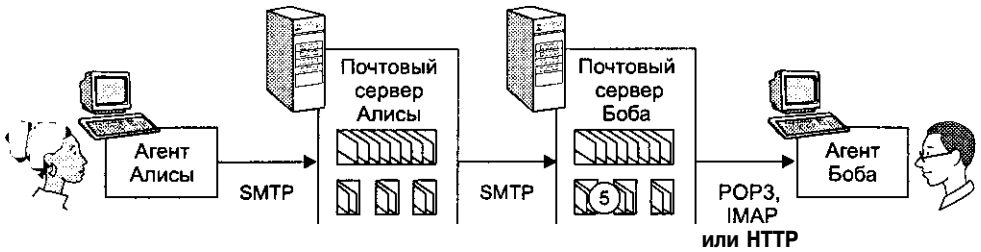


Рис. 2.12. Протоколы электронной почты и их взаимосвязь

POP3

Протокол POP3, описанный в документе RFC 1939, является одним из самых простых протоколов доступа к электронной почте. Увы, простота протокола POP3 оборачивается его весьма ограниченной функциональностью. Протокол начинает действовать после того, как агент пользователя (клиент) устанавливает TCP-соединение с портом 110 почтового сервера, и подразумевает выполнение трех основных фаз: авторизации, транзакции и обновления. Во время авторизации агент передает серверу имя пользователя и пароль для того, чтобы сервер предоставил агенту доступ к сообщениям электронной почты. В фазе транзакции пользователь получает сообщения, а также может пометить сообщения, предназначенные для удаления, и получить почтовую статистику. Наконец, фаза обновления наступает после того, как клиент посылает команду quit и закрывает POP3-сеанс. Почтовый сервер производит удаление сообщений, помеченных пользователем.

Во время POP3-транзакции агент пользователя посылает почтовому серверу команды, на каждую из которых сервер реагирует посылкой одного из двух ответных сообщений: +0K (иногда с последующей передачей данных сервера клиенту) и -ERR, указывающего на ошибку в команде клиента.

Авторизация включает в себя две возможные команды: user <имя пользователя> и pass <пароль>. Для того чтобы проиллюстрировать действие этих команд, предположим, что вы устанавливаете соединение с POP3-сервером через порт 110. Если mailServer — имя вашего почтового сервера, то процесс авторизации в программе Telnet будет выглядеть следующим образом:

```
telnet mailServer 110
+0K POP3 server ready
user bob
+0K
pass hungry
+0K user successfully logged on
```

Если какая-либо из команд будет введена неверно, сервер выдаст сообщение -ERR. Теперь обратимся к фазе транзакции. Как правило, агент пользователя, использующий протокол POP3, в зависимости от настроек может автоматически удалять или не удалять сообщения после их приема; при этом во время транзакции будут применяться различные команды. Если загруженные сообщения необходимо удалять, агент посылает серверу команды list, retr и dele. Пусть, например, в почтовом

ящике пользователя находятся два сообщения. Ниже приведен диалог клиента (C) и сервера (S) во время транзакции:

```
C: list
S: 1 498
S: 2 912
S:.
C: retr 1
S: (blah blah ...
S
S            blah)
S
C  dele 1
C  retr 2
S  (blah blah ...
S
S            blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signong off
```

Сначала агент пользователя получает от сервера список сообщений с указанием размера каждого сообщения, а затем последовательно принимает и удаляет сообщения с сервера. Во время транзакции агентом используются лишь четыре команды: list, retr, dele и quit. Синтаксис этих команд описан в документе RFC 1939. После обработки команды quit POP3-сервер переходит в фазу обновления и производит фактическое удаление переданных сообщений.

Режим удаления переданных сервером сообщений имеет важный недостаток. Предположим, Боб является мобильным пользователем и получает доступ к почтовому серверу с разных компьютеров (например, домашнего, офисного и портативного). Если каждый раз после передачи сообщений сервер будет удалять их, то часть сообщений окажется на персональном компьютере, часть — на офисном, а часть — на портативном. Таким образом, Боб будет лишен возможности одновременного доступа ко всем полученным сообщениям. Если агенты пользователя на компьютерах Боба будут настроены на загрузку сообщений без удаления, копии всех входящих сообщений останутся в почтовом ящике, что обеспечит доступ к ним с любого компьютера.

Хотя во время POP3-сеанса между почтовым сервером и агентом пользователя почтовый сервер сохраняет определенную информацию о состоянии (в основном это относится к списку сообщений, предназначенных для удаления), сохранять полную информацию о сеансе не нужно. Это в значительной степени упрощает реализацию почтового POP3-сервера.

IMAP

Если Боб использует протокол доступа к электронной почте POP3, он может создавать на своем компьютере специальные почтовые папки, в которые будут попадать загруженные с сервера сообщения. Кроме того, Боб может удалять загруженные сообщения, перемещать их между папками и производить поиск сообщений по имени отправителя или теме. Такая система хранения сообщений, реализован-

ная в виде папок на локальном компьютере, удобна для резидентного пользователя, однако вряд ли подходит в случае, если пользователь регулярно меняет вычислительные машины, с которых осуществляет доступ к электронной почте. Организация иерархии папок на почтовом сервере была бы весьма удобна для «мультикомпьютерных» пользователей. Именно по этой причине был разработан другой протокол доступа к почте — IMAP.

Протокол IMAP описан в документе RFC 2060. Он имеет много общего с протоколом POP3, однако его структура значительно сложнее; сложнее и реализация клиентской и серверной сторон IMAP.

IMAP-сервер связывает каждое сообщение с некоторой пользовательской папкой. Изначально каждое принятое сообщение попадает в папку INBOX, где пользователь может прочитать его, а затем переместить в другую папку, удалить и т. п. Для всех подобных действий протоколом IMAP предусмотрены специальные команды. Удобной функцией является возможность поиска в каждой из папок писем, удовлетворяющих заданному критерию. Обратите внимание на то, что, в отличие от POP3, IMAP-сервер сохраняет информацию о ходе IMAP-сеанса, в том числе об именах папок, о том, какие сообщения в каких папках находятся, и т. п.

Еще одним важным достоинством IMAP является наличие команд, позволяющих пользователю получать отдельные компоненты сообщений: заголовки, части составных MIME-сообщений и т. д. Эта возможность удобна при низкоскоростных соединениях между пользователем и Интернет-провайдером. Некоторые пользователи предпочитают избегать загрузки длинных сообщений, содержащих несколько объемных вложений (например, аудио- или видеоклипов), и возможность выбирать нужные фрагменты для них весьма кстати. Более подробную информацию о протоколе IMAP вы можете получить на официальном сайте IMAP [506].

Электронная почта с web-интерфейсом

Все больше и больше пользователей Интернета получают доступ к своим электронным почтовым ящикам с помощью web-браузеров. Компания Hotmail первой применила web-технологии для работы с электронной почтой в середине прошлого десятилетия; в настоящее время эта услуга предлагается практически каждым порталом Интернета, а также большинством Интернет-провайдеров. При доступе к электронной почте через web-интерфейс роль агента пользователя играет web-браузер, который взаимодействует с удаленным почтовым ящиком по протоколу HTTP. Когда Боб хочет получить новые сообщения, он подключается к своему почтовому серверу, который отсылает Бобу сообщения по протоколу HTTP (а не SMTP или IMAP). Аналогично Алиса передает новые сообщения своему почтовому серверу через браузер по протоколу HTTP. Следует обратить внимание на то, что обмен сообщениями между почтовыми серверами Алисы и Боба, как и ранее, происходит по протоколу SMTP.

Механизм доступа к почте через web-интерфейс очень удобен для тех, кто регулярно пользуется несколькими компьютерами. Для чтения и отправки сообщений пользователю необходимо лишь иметь доступ к компьютеру или цифровому органайзеру с браузером, который может находиться дома, в офисе, в отеле, в Интер-

нет-кафе. Как и в случае протокола ШАР, пользователи имеют возможность организовать иерархическую структуру папок в своем почтовом ящике; более того, для этого используются ШАР-серверы. Обычно доступ к папкам и сообщениям осуществляется с помощью скриптов, выполняющихся на HTTP-сервере; эти скрипты для доступа к ШАР-серверу используют протокол ШАР.

Служба трансляции имен Интернета

Существует много способов идентификации людей. Так, например, мы можем идентифицировать человека по его паспортным данным, номеру водительских прав, индивидуальному налоговому номеру (ИНН), номеру страхового полиса и т. д. Несмотря на то что все перечисленные способы позволяют однозначно установить личность любого человека, применяемость того или иного идентификатора зависит от ситуации. Например, в медицинских учреждениях удобнее всего применять номер страхового полиса пациентов, в отделах кадров предприятий — ИНН сотрудников, а при общении между людьми — паспортные данные.

Подобно людям, Интернет-хосты также имеют множество идентификаторов. Одним из идентификаторов *является имя хоста.* Имя хоста представляет собой мнемоничную, а следовательно, удобную для восприятия человеком запись, например **cnn.com, www.yahoo.com, gaia.cs.umass.edu, surf.eurecom.fr.** Недостатком имен хостов является то, что они не содержат информации о конкретном расположении хоста; единственным указателем на географическое местоположение может служить код страны (например, код **fr** указывает на принадлежность хоста к Франции, и т. п.). Другой недостаток имен хостов заключается в их значительной длине, приводящей к существенным затратам на обработку маршрутизаторами. По указанным причинам был введен другой идентификатор хостов — *IP-адрес.* Мы подробно разберем вопросы, касающиеся IP-адресов, в главе 4, а сейчас лишь познакомим вас с этим важным понятием. IP-адрес представляет собой совокупность четырех однобайтовых чисел и имеет жесткую иерархическую структуру. IP-адреса обычно записываются в виде четырех десятичных чисел, разделенных точками и представляющих значения каждого из байтов: 121.7.106.83. Каждое число находится в диапазоне от 0 до 255. Иерархичность IP-адресов заключается в том, что при их чтении слева направо мы получаем все более точную информацию о местонахождении хоста в Интернете (говоря точнее, мы определяем принадлежность хоста к той или иной сети, входящей в сеть сетей — Интернет). Аналогичным образом, сканируя почтовый адрес, мы последовательно уточняем местонахождение адресата.

Функции DNS

Как мы видели, существуют два принципиально разных способа идентификации хостов: с помощью имен и с помощью IP-адресов. Имя хоста удобно для людей в силу своей мнемоничности, а IP-адрес, являющийся компактной числовой величиной фиксированного размера, проще обрабатывать маршрутизаторами. Для того чтобы установить связь между этими двумя идентификаторами, используется *система доменных имен* (Domain Name System, DNS). DNS представляет собой,

с одной стороны, базу данных, распределенную между иерархически структурированными *серверами имей,* и, с другой стороны, протокол прикладного уровня, организующий взаимодействие между хостами и серверами имен для выполнения операций преобразования. Зачастую серверы имен являются UNIX-машинами, использующими программное обеспечение BIND (Berkeley Internet Name Domain — домен имен Интернета Беркли). Протоколу DNS назначен порт с номером 53, и работает DNS поверх протокола UDP транспортного уровня. На сайте http://www.awl.com/kurose-ross, посвященном этой книге, вы можете найти интерактивные ссылки на DNS-программы, осуществляющие преобразование произвольных имен хостов в IP-адреса.

Обычно DNS используется другими протоколами прикладного уровня: HTTP, SMTP и FTP для получения IP-адресов вместо вводимых пользователями имен хостов. Рассмотрим, к примеру, ситуацию, когда пользователь вводит в адресной строке браузера адрес web-страницы www.someschool.edu/index.html. Для того чтобы сформировать запрос, пользовательский хост должен сначала получить IP-адрес удаленного хоста, на котором находится ресурс, то есть www.someschool.edu. При работе протокола DNS пользовательский хост играет роль клиента. Браузер выделяет из URL-адреса страницы имя хоста и передает его клиентской стороне DNS-приложения, которая формирует и отправляет запрос DNS-серверу. DNS-сервер обрабатывает запрос и отсылает клиенту ответ, содержащий IP-адрес хоста. Затем браузер открывает TCP-соединение с HTTP-сервером, выполняющимся на хосте с полученным IP-адресом. Очевидно, что процесс получения IP-адреса не является мгновенным и вносит дополнительную задержку в суммарное время установления соединения с HTTP-сервером, которая иногда может быть весьма значительной. Как мы увидим позже, среднее время получения IP-адреса (а также объем DNS-трафика) может быть уменьшено путем кэширования IP-адреса на «ближайшем» DNS-сервере.

Помимо преобразования имен хостов в IP-адреса, DNS выполняет еще несколько важных функций, перечисленных ниже.

- *Поддержка псевдонимов серверов.* Хосты с длинными именами могут иметь один или несколько псевдонимов; например, хосту relayl.west-coast.enterprise.com можно присвоить два псевдонима — enterprise.com и www.enterprise.com. В этом случае говорят, что relayl.west-coast.enterprise.com является *каноническим именем хоста.* Обычно введение псевдонимов вызывается недостаточной мнемоничностью канонического имени. Получение канонического имени хоста по заданному псевдониму, как и IP-адреса, выполняется с помощью протокола DNS.
- *Поддержка псевдонимов почтовых серверов.* Очевидно, что мнемоничность особенно важна для адресов электронных почтовых ящиков. Например, если Боб использует почтовый ящик компании Hotmail, то его адрес будет иметь вид bob@hotmail.com, что нетрудно запомнить. На самом деле hotmail.com является лишь псевдонимом почтового сервера Боба, в то время как каноническое имя имеет гораздо более сложную структуру, например relayl.west-coast.hotmail.com. Приложение электронной почты с помощью протокола DNS по заданному псев-

дониму получает каноническое имя и IP-адрес сервера. Фактически запись типа MX (см. ниже) позволяет почтовому серверу компании и ее web-серверу иметь одинаковые псевдонимы, например enterprise.com.

- *Распределение загрузки.* В последнее время DNS все больше используется для распределения загрузки между дублирующими серверами. Популярные сайты, например cnn.com, зачастую имеют несколько копий (или реплик, или зеркал), расположенных на различных серверах с различными IP-адресами. Таким образом, в случае дублирующих серверов с одним именем хоста связывается множество IP-адресов, которые хранятся в базе данных DNS. Когда производится DNS-запрос для имени, которому сопоставлено несколько IP-адресов, в ответ включаются все IP-адреса, однако сервер может изменять порядок их перечисления. Поскольку обычно HTTP-клиент сначала формирует запрос с использованием IP-адреса, который был указан в списке первым, это позволяет распределять загрузку между дублирующими серверами. Подобный механизм применим как для web-серверов, так и для почтовых серверов, то есть несколько почтовых серверов могут иметь один и тот же псевдоним. В последнее время некоторые компании, такие как Akamai [9], стали применять DNS для «изощренных» методов распределения web-ресурсов (см. раздел «Распределение ресурсов» этой главы).

Основные спецификации DNS содержатся в документах RFC 1034 и RFC 1035; кроме того, некоторые дополнения можно найти в других документах RFC. В этой книге мы затронем лишь ключевые аспекты системы DNS ввиду ее значительной сложности. Заинтересованный читатель может обратиться к указанным документам RFC и к [3].

ПРИНЦИПЫ И ПРАКТИКА

DNS, подобно протоколам HTTP, FTP и SMTP, является протоколом прикладного уровня. Он выполняется на оконечных системах, опираясь на парадигму клиент/сервер и службы протокола транспортного уровня, передающие DNS-сообщения между оконечными системами. Тем не менее роль DNS в значительной степени отличается от роли web-протоколов, а также протоколов электронной почты и обмена файлами. Пользователь не взаимодействует с DNS напрямую: DNS осуществляет в Интернете «закулисную» функцию преобразования имен хостов в IP-адреса. Эта функция используется либо пользовательскими приложениями, либо другим сетевым программным обеспечением. В разделе «Ядро компьютерных сетей» главы 1 мы рассказали о том, что наиболее сложные проблемы Интернета сосредоточены на его «периферии». Служба DNS, выполняющая трансляцию адресов с помощью клиентов и серверов, находящихся на разбросанных по Интернету оконечных системах, только подтверждает эту философию разработчиков.

Общие принципы функционирования DNS

Ниже мы опишем основную схему функционирования DNS. Основным предметом рассмотрения для нас будет являться преобразование имени хоста в соответствующий IP-адрес.

Предположим, что некоторому приложению (web-браузеру или программе чтения электронной почты), выполняющемуся на хосте пользователя, необходимо получить IP-адрес удаленного хоста. Приложение вызывает клиентскую сторону DNS и передает ей имя нужного хоста. (Многие UNIX-машины поддерживают функцию gethostbyname(), вызываемую приложениями для получения IP-адреса. В разделе «Программирование UDP-сокетов» этой главы мы покажем, каким образом Java-приложение может обратиться к службе DNS.) Клиентская сторона, в свою очередь, создает запрос, отсылает его DNS-серверу и ждет ответа. Как правило, время ответа DNS-сервера составляет от нескольких миллисекунд до десятков секунд. Ответ представляет собой сообщение, содержащее группу из одного или нескольких IP-адресов, которые передаются DNS-клиентом вызвавшему его приложению. Таким образом, с точки зрения приложения служба DNS является «черным ящиком», на входе которого находится имя удаленного хоста, а на выходе — его IP-адрес. На самом деле этот «черный ящик» представляет собой сложную систему, состоящую из множества серверов имен, распределенных по всему земному шару, и протокола, определяющего взаимодействие между серверами имен и пользовательскими хостами.

Как устроена служба DNS? Ее можно представить себе в виде единственного центрального сервера имен, который содержит всю информацию об именах хостов и их IP-адресах. Этот сервер принимает все запросы и отсылает ответы «без посредников» напрямую каждому DNS-клиенту. К сожалению, огромное (и постоянно растущее) число Интернет-хостов не позволяет применить эту простую схему на практике. Централизованной системе присущи некоторые «врожденные» недостатки.

- *Единственная точка возможного отказа.* Сервер имен является «узким местом» с точки зрения надежности — его отказ парализует работу всего Интернета.
- *Объем трафика.* Сервер имен является «узким местом» с точки зрения загрузки, поскольку вынужден обрабатывать все запросы, поступающие с сотен миллионов хостов всего мира.
- *Удаленность централизованной базы данных.* Единственный сервер невозможно расположить на приемлемом расстоянии от всех клиентов. Например, разместив сервер в Нью-Йорке, мы обеспечим хосты Австралии гарантированными задержками, связанными с передачей сигнала по линиям связи, далеко не всегда высокоскоростным.
- *Обслуживание.* База данных сервера должна не только хранить адреса всех существующих хостов, но и постоянно обновляться с появлением новых хостов. Это влечет за собой проблемы, связанные с авторизацией и идентификацией каждого пользователя сети, пытающегося внести данные о своем хосте в центральную базу данных.

Таким образом, создание единственного сервера имен с централизованной базой данных оказывается невозможным, что обусловливает распределенную структуру DNS. DNS является замечательным примером того, как в Интернете может быть организована распределенная база данных.

Для того чтобы решить проблему хранения больших объемов информации, система DNS была спроектирована в виде совокупности многочисленных серверов имен,

рассредоточенных по всему миру и организованных в виде иерархической структуры. Ни один сервер имен не содержит информации обо всех IP-адресах хостов; эта информация распределена между множеством серверов. Можно ввести следующую укрупненную классификацию серверов имен: локальные, корневые и полномочные. Ниже описаны механизмы взаимодействия этих серверов друг с другом и с пользовательскими хостами.

- *Локальные серверы имен* имеются у каждого Интернет-провайдера (локальные серверы имен также называют серверами имен по умолчанию). Когда пользовательский хост посылает DNS-запрос, этот запрос сначала попадает на локальный сервер имен. Обычно IP-адрес локального сервера имен конфигурируется пользователем вручную, например, в операционной системе Windows для этого необходимо открыть Панель управления (Control Panel), щелкнуть на значке Сеть (Network), в списке установленных компонентов открывшегося окна выделить пункт TCP/IP и, щелкнув на кнопке Свойства (Properties), в появившемся диалоговом окне перейти на вкладку Конфигурация DNS (DNS Configuration). Обычно локальный сервер имен расположен на относительно небольшом расстоянии от пользовательского хоста. В случае, если Интернет-провайдером является государственное или коммерческое учреждение, сервер имен принадлежит локальной сети организации; для резидентных Интернет-провайдеров сервер имен обычно отделен от пользователя не более чем несколькими маршрутизаторами. Если окажется, что удаленный хост принадлежит тому же Интернет-провайдеру, локальный сервер самостоятельно проведет преобразование, и требуемый IP-адрес будет отослан пользовательскому хосту. Подобная ситуация, например, будет иметь место, когда хост surf.eurecom.fr затребует IP-адрес baie.eurecom.fr. В этом случае IP-адрес выдается локальным сервером имен института Eurecom.
- *Корневые серверы имен* являются следующей ступенью в иерархии серверов DNS. Их число в Интернете составляет немногим более десятка, и большинство из них находится в Северной Америке. Рисунок 2.13 иллюстрирует положение корневых серверов имен на географической карте мира в феврале 2002 года; кроме того, со списком корневых серверов имен можно ознакомиться в [128]. В случае, если локальный сервер имен не может самостоятельно удовлетворить запрос пользовательского хоста, он берет на себя роль клиента и передает запрос одному из корневых серверов имен. Корневой сервер обрабатывает запрос и при наличии соответствующей записи в базе данных отсылает локальному серверу ответ, содержащий IP-адрес, который, в свою очередь, передается локальным сервером пользовательскому хосту. Тем не менее корневой сервер имен не всегда может выполнить запрос из-за ограниченности своей базы данных. В случае, если хост не найден в базе данных, локальному серверу отсылается IP-адрес полномочного сервера имен, который располагает искомым IP-адресом.
- *Полномочный сервер имен* — это сервер, на котором зарегистрирован данный хост. Обычно хосты регистрируются на локальных серверах имен Интернет-провайдеров (на самом деле в целях обеспечения надежности хосты регистрируются не менее чем на двух полномочных серверах). Полномочный сервер содержит

информацию о связи имени хоста с его IP-адресом. Когда корневой сервер посылает запрос полномочному серверу, последний отсылает ответ, содержащий требуемый IP-адрес. Далее корневой сервер отсылает полученный ответ локальному серверу, а локальный сервер — хосту пользователя. Таким образом, многие серверы имен одновременно являются локальными и полномочными.

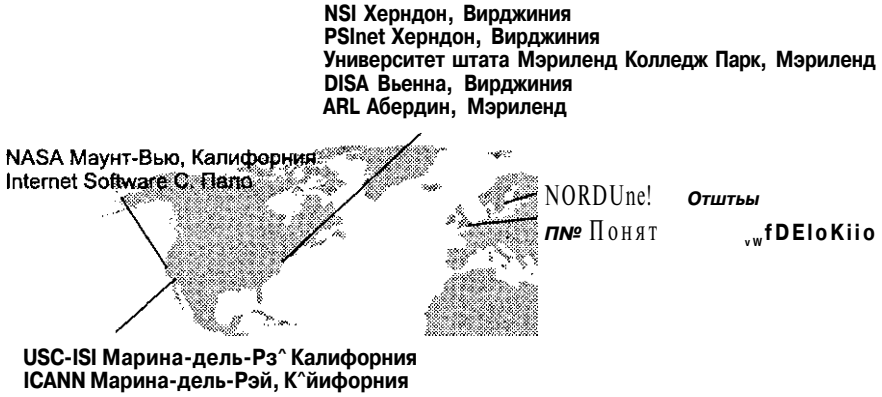


Рис. 2.13. Корневые серверы DNS в начале 2002 года (имя, организация, расположение)

Рассмотрим простой пример. Предположим, что хост surf.eurecom.fr создал запрос IP-адреса gaia.cs.umass.edu. Пусть локальный сервер имен института Eurecom имеет имя dns.eurecom.fr, а полномочный сервер имен хоста gaia.cs.umass.edu — имя dns.umass.edu. Как показано на рис. 2.14, хост surf.eurecom.fr сначала отправляет запрос своему локальному серверу имен dns.eurecom.fr, в котором содержится имя для преобразования в IP-адрес (gaia.cs.umass.edu). Локальный сервер имен передает запрос корневому серверу, который, в свою очередь, перенаправляет его серверу, являющемуся полномочным для всех хостов домена umass.edu, например dns.umass.edu. Сервер обрабатывает запрос, создает ответ с IP-адресом хоста gaia.cs.umass.edu и отсылает его хосту surf.eurecom.fr через корневой и локальный серверы имен. Обратите внимание на то, что в этом примере процедура получения IP-адреса требует передачи шести DNS-сообщений: трех запросов и трех ответов.

Выше было сделано допущение о том, что корневой сервер имен располагает IP-адресами полномочных серверов для любого хоста. На самом деле это допущение неверно. Корневой сервер «знает» адрес лишь промежуточного сервера имен, который содержит IP-адрес полномочного сервера хоста. Для того чтобы проиллюстрировать это, снова обратимся к примеру с хостом surf.eurecom.fr, желающим получить IP-адрес хоста gaia.cs.umass.edu. Предположим, что локальный сервер имен университета Массачусетс имеет имя dns.umass.edu. Допустим также, что каждое подразделение университета располагает собственным локальным сервером имен,

являющимся полномочным для всех хостов соответствующего подразделения. Как показано на рис. 2.15, когда корневой сервер имен получает запрос с именем, оканчивающимся на umass.edu, он перенаправляет запрос серверу имен dns.umass.edu. Далее сервер dns.umass.edu перенаправляет все запросы с именами, оканчивающимися на cs.umass.edu, локальному серверу имен dns.cs.umass.edu, являющемуся полномочным для всех хостов cs.umass.edu. Искомый IP-адрес передается сервером dns.cs.umass.edu хосту surf.eurecom.fr последовательно через серверы dns.umass.edu, корневой сервер имен и локальный сервер имен dns.eurecom.fr. Таким образом, получение IP-адреса в этом примере потребовало посылки восьми DNS-сообщений. На практике встречаются ситуации, когда между корневым и полномочным серверами имен находятся два или более промежуточных сервера, что еще увеличивает количество DNS-сообщений в цикле получения IP-адреса.

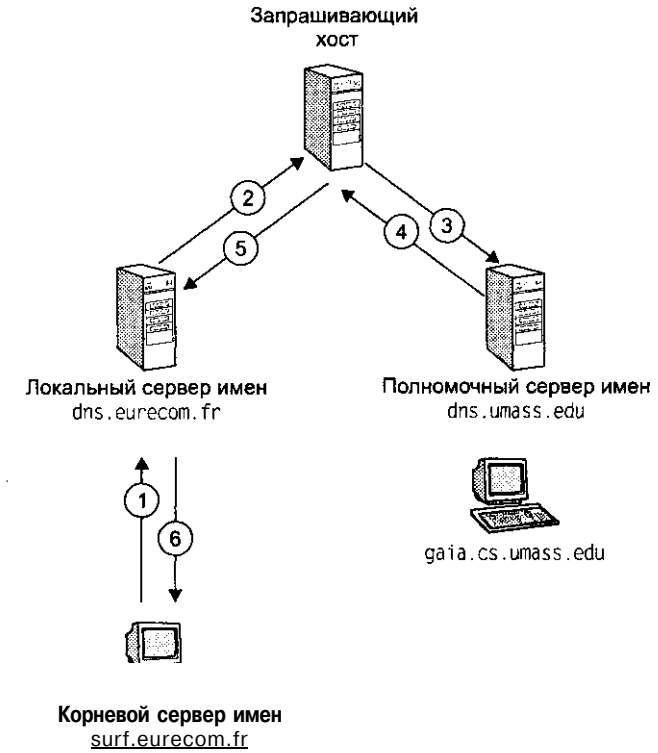


Рис. 2.14. Рекурсивные запросы при получении IP-адреса gaia.cs.umass.edu

Все приведенные выше примеры строились на основе допущения о том, что все DNS-запросы являются *рекурсивными*. Это означает, что, когда хост или сервер имен A обращается к серверу имен B, последний предпринимает необходимые для получения требуемого IP-адреса действия от имени A и передает его A. Протокол DNS предусматривает также *итеративные* запросы. Итеративные запросы отличаются от рекурсивных тем, что в случае отсутствия искомого IP-адреса сервер

имен B возвращает A IP-адрес следующего сервера имен в цепочке, к которому A должен обратиться самостоятельно.

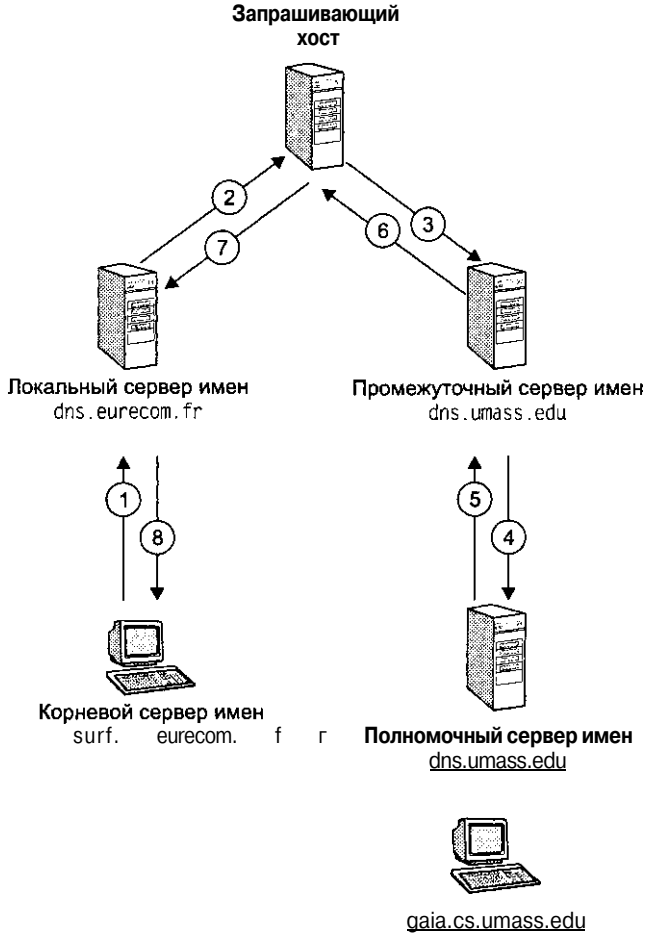


Рис. 2.15. Рекурсивные запросы к промежуточному серверу имен, расположенному между корневым и полномочными серверами

Последовательность запросов, необходимых для получения IP-адреса хоста, может содержать одновременно и рекурсивные, и итеративные запросы. Пример такой смешанной цепи представлен на рис. 2.16. На практике часто встречаются ситуации, когда все запросы являются рекурсивными за исключением единственного итеративного запроса локального сервера имен к корневому. Это объясняется тем, что корневые серверы вынуждены обрабатывать значительное число запросов, а итеративный механизм снижает трудоемкость обработки запроса сервером. На сайте http://www.awl.com/kurose-ross вы можете найти Java-апплет, позволяющий экспериментировать с различными комбинациями рекурсивных и итеративных запросов.

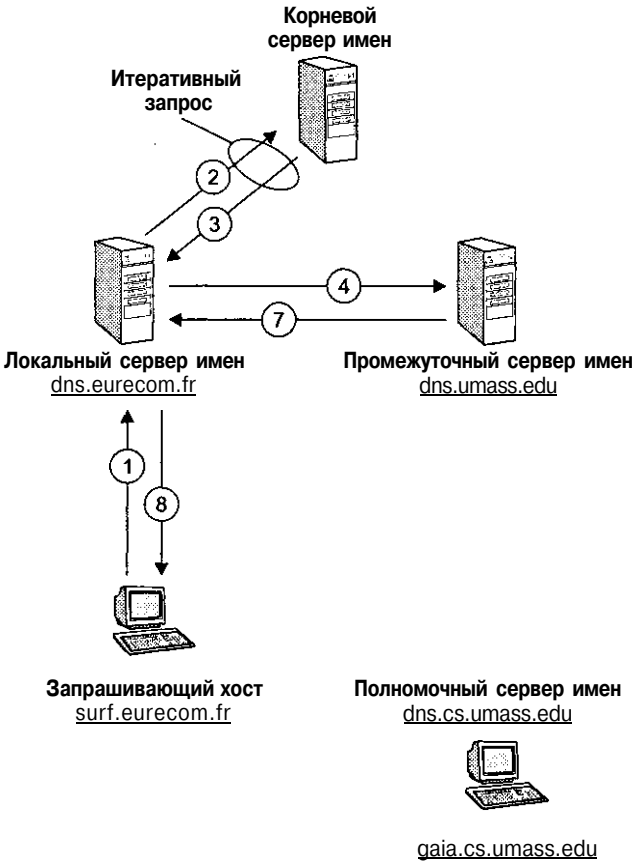


Рис. 2.16. Цепочка, включающая рекурсивные и итеративные запросы

Теперь настало время рассмотреть такую важную составляющую DNS, как *кэширование*. Механизм кэширования активно используется DNS для того, чтобы сократить число запросов и время получения IP-адресов хостами. Идея кэширования проста: когда сервер имен получает ответ, содержащий IP-адрес хоста, он сохраняет пару «имя/1P-адрес хоста» в специально отведенной области памяти (дисковой или оперативной). В случае, если этот сервер имен вновь получит запрос с тем же именем хоста, он сможет извлечь соответствующую пару из кэш-памяти и сгенерировать ответ, даже не являясь полномочным сервером хоста. Чтобы не хранить большие объемы невостребованной информации, записи остаются в кэш-памяти некоторый ограниченный промежуток времени (обычно 48 часов). Рассмотрим следующий пример. Пусть хост surf.eurecom.fr создал запрос на получение IP-адреса хоста cnn.com. Несколько часов спустя другой хост института Eurecom, например baie.eurecom.fr, также создал DNS-запрос хосту cnn.com. Механизм кэширования обеспечивает появление IP-адреса cnn.com на локальном сервере имен Eurecom после выполнения запроса хоста surf.eurecom.fr; таким образом, запрос хоста baie.eurecom.fr будет удовлетворен локальным сер-

вером без дальнейшей передачи. Кэширование поддерживается всеми серверами имен.

DNS-записи

Серверы имен, в совокупности образующие базу данных DNS, хранят *ресурсные записи* (Resource Records, RR), связывающие имена хостов с их IP-адресами. Каждый DNS-ответ содержит одну или более ресурсных записей. Ниже мы рассмотрим несколько вопросов, касающихся записей и сообщений DNS; более детальное описание можно найти в [3], а также в документах RFC (RFC 1034, RFC 1035).

RR-запись состоит из четырех полей:

(Name. Value, Type, TTL)

Поле **TTL** определяет время жизни записи в кэш-памяти; говоря точнее, оно содержит время удаления записи из кэш-памяти. В примерах, приводимых ниже, мы будем опускать поле **TTL.** Значения полей **Name** и **Value** зависят от поля Type.

- Если Type = **A,** то поле **Name** содержит имя хоста, **а Value** — IP-адрес. Таким образом, тип **A** в дополнение к IP-адресу включает стандартное имя хоста. Примером записи типа **A** является **(relayl.bar.foo.com, 145.37.93.126, A).**
- Если Type = **NS,** то поле **Name** содержит домен (например, **foo.com**), **а Value** — имя хоста или полномочного сервера, располагающего информацией о том, на каком сервере имен хранятся IP-адреса хостов домена. Эта запись используется для дальнейшей передачи запроса по цепочке серверов имен. Примером записи типа **NS** является **(foo.com, dns.foo.com, NS).**
- Если Type = **CNAME,** то **Value** является каноническим именем хоста, **а Name** — псевдонимом хоста. Этот тип записи позволяет получить каноническое имя хоста по известному псевдониму. Примером записи типа **CNAME** является **(foo.com, relayl.bar.foo.com, CNAME).**
- Если Type = **MX,** то **Value** является каноническим именем почтового сервера с псевдонимом **Name.** Примером записи типа **MX** является **(foo.com,** mai4**.bar.foo.com, MX).** Записи типа **MX** позволяют использовать простые псевдонимы вместо длинных имен хостов почтовых серверов. Обратите внимание на то, что при помощи записей типа **MX** компании могут задействовать один и тот же псевдоним для своего почтового сервера и других серверов (например, web-сервера). Для получения канонического имени почтового сервера DNS-клиент создает запрос на запись типа **MX,** а для получения канонического имени web-серверов и прочих серверов — запрос на запись типа **CNAME.**

Если сервер имен является полномочным для хоста, то для этого хоста он будет содержать запись типа **A** (тем не менее, если сервер имен не является полномочным, он может содержать запись типа **A** для данного хоста в кэш-памяти); в противном случае сервер имен содержит запись типа **NS** с информацией о домене, включающем имя хоста, а также запись типа **A** с IP-адресом сервера имен, указанного в поле **Value** записи типа **NS.** Предположим, к примеру, что корневой сервер имен не является полномочным для хоста **gaia.cs.umass.edu. В** этом случае корневой сервер будет содержать запись о домене, включающем хост **cs.umass.edu (umass.edu.**

dns.umass.edu, NS), и запись, связывающую имя сервера имен dns.umass.edu с его IP-адресом (dns.umass.edu, 128.119.40.111, A).

DNS-сообщения

В этом разделе мы столкнулись с понятиями DNS-запроса и DNS-ответа. Запрос и ответ представляют собой единственные два типа сообщений, используемые протоколом DNS. Форматы этих сообщений совпадают и представлены на рис. 2.17. Ниже приведено описание структуры DNS-сообщения.

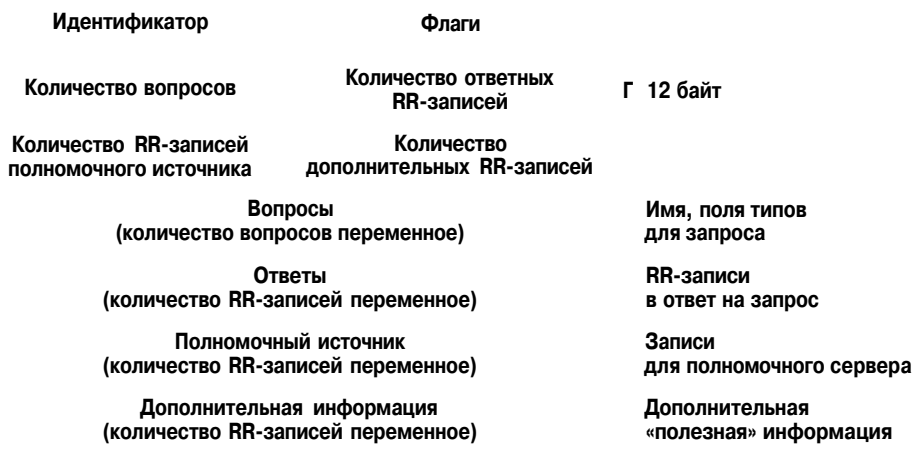


Рис. 2.17. Формат DNS-сообщения

- Первые 12 байт составляют *заголовочную секцию,* состоящую из нескольких полей. Первое поле представляет собой 16-разрядное число, идентифицирующее запрос. Идентификатор запроса копируется в ответное сообщение, что позволяет клиенту сопоставлять ответы с запросами. Поле флагов включает одноразрядный флаг «запрос/ответ» и определяет, является ли сообщение запросом (0) или ответом (1). Одноразрядный флаг полномочности устанавливается для ответного сообщения в случае, если сервер имен является полномочным для запрашиваемого имени. Одноразрядный флаг предпочтительности рекурсии устанавливается в случае, когда клиенту (хосту или серверу имен) необходимо дать указание серверу имен применить рекурсивный запрос при отсутствии IP-адреса. Одноразрядный флаг доступности рекурсии устанавливается в ответном сообщении, если сервер имен поддерживает рекурсивный механизм запросов. В заголовке также содержатся четыре секции для «типов» данных.
- *Секция вопросов* содержит информацию о запросе и включает поле имени, в котором указано имя запрашиваемого хоста, и поле типа, определяющее содержание ответа, например адрес хоста (тип A) или имя почтового сервера (тип MX).

- *Секция ответов* присутствует в ответных сообщениях и содержит требуемые ресурсные записи. Каждая RR-запись включает поля Type с одним из значений A, NS, CNAME или MX, Value и TTL. Поскольку имени хоста может быть сопоставлено несколько IP-адресов (например, из-за дублирования web-серверов, упоминавшегося в этой главе), секция ответов также может содержать несколько записей.
- *Секция полномочности* включает в себя записи о других полномочных серверах.
- *Дополнительная секция* содержит прочие «полезные» записи. Например, в поле ответов для запроса на запись типа MX может находиться запись, хранящая каноническое имя почтового сервера, а в дополнительную секцию помещена запись типа A с IP-адресом почтового сервера.

Приведенные выше сведения в основном касались извлечения записей из базы данных DNS. Однако открытым пока остается вопрос о том, как в базу заносятся новые данные. До последнего времени обновление базы данных производилось статически, например путем ручного ввода данных в файл конфигурации системным администратором. Недавно в протокол DNS был добавлен параметр UPDATE, позволяющий с помощью DNS-сообщений добавлять и удалять записи базы данных. Описание параметра UPDATE можно найти в RFC 2136.

В [239] можно найти множество ресурсов для BIND, популярного сервера имен для UNIX-машин.

Программирование TCP-сокетов

Этот и следующий разделы посвящены знакомству с вопросами разработки приложений для Интернета. Как упоминалось в разделе «Принципы работы протоколов прикладного уровня», ядро сетевого приложения состоит из двух программ — клиента и сервера. Когда эти программы запускаются, создаются клиентский и серверный процессы, которые взаимодействуют друг с другом, обмениваясь сообщениями через сокеты. При создании сетевого приложения главной задачей разработчика является написание программного кода для клиентской и серверной частей приложения.

Существуют два вида приложений с архитектурой клиент/сервер. К первому виду относятся приложения, поддерживающие стандартный протокол, описанный в документах RFC. В этом случае как клиентская, так и серверная части приложения должны соответствовать всем описаниям, приведенным в RFC. Например, если приложение использует протокол FTP, то клиентская и серверная программы приложения должны быть построены в соответствии с требованиями документа RFC 959, описывающего FTP-приложения. Следование стандартам позволяет независимым разработчикам подключаться к созданию различных частей одного и того же приложения. Так, например, браузер Netscape способен успешно взаимодействовать с web-сервером Apache, а FTP-клиент, предназначенный для домашнего компьютера, — с FTP-сервером на платформе UNIX. Для приложений, разработанных по стандарту RFC, следует применять порт с номером, соответствующим

используемому протоколу (мы кратко рассказали о номерах портов в разделе «Принципы работы протоколов прикладного уровня», более подробно они рассматриваются в главе 3).

Второй вид приложений с архитектурой клиент/сервер — нестандартные приложения. В таких приложениях поддержка клиентом и сервером каких-либо соглашений, принятых в RFC, не обязательна. Разработчик в лице одного или нескольких человек создает клиентскую и серверную части приложения, полностью контролируя процесс написания кода. При этом другие разработчики не смогут создавать программы, взаимодействующие с нестандартным приложением. Для нестандартных приложений необходимо использовать порты с номерами, отличающимися от хорошо известных, указанных в RFC.

В этом и следующем разделах мы рассмотрим ключевые принципы разработки нестандартных приложений с архитектурой клиент/сервер. Одно из первых решений, которое вынужден принимать разработчик, касается выбора протокола транспортного уровня (TCP или UDP) для своего приложения. Как вы помните, протокол TCP предполагает установление логического соединения между хостами и обеспечивает надежную передачу данных. Протокол UDP, напротив, не устанавливает логического соединения, и передача пакетов осуществляется без гарантии их доставки адресату.

В этом разделе мы создадим простую клиентскую часть приложения, поддерживающую протокол TCP, а в следующем разделе — протокол UDP. Обе программы написаны на языке Java. Необходимо отметить, что эти же программы можно было с равным успехом писать на языке C или C++, однако наш выбор обусловлен несколькими существенными причинами. Во-первых, Java-код проще, понятнее (что особенно важно для новичков) и состоит из меньшего числа строк, чем код на C/C++. Во-вторых, программирование приложений клиент/сервер на языке Java в последние годы получило большую популярность, и не исключено, что в ближайшие годы станет общепринятым. Не стоит отчаиваться, если вы не знакомы с языком Java: чтение кода не станет для вас проблемой, если вы обладаете навыками программирования на любом другом языке.

Читателям, интересующимся аспектами программирования для архитектуры клиент/сервер на языке C, рекомендуем обратиться к дополнительным источникам информации [171, 284,487].

Взаимодействие процессов при помощи TCP-сокетов

Как мы отмечали в разделе «Принципы работы протоколов прикладного уровня», процессы, выполняющиеся на разных вычислительных машинах, взаимодействуют друг с другом при помощи сообщений, посылаемых через сокеты. Мы представили процессы в виде домов, а сокеты — в виде дверей. Как показано на рис. 2.18, сокет является дверью между процессом и протоколом TCP. Разработчик приложения полностью контролирует ту часть сокета, которая относится к прикладному уровню, почти не имея возможности воздействовать на «транспортную» часть

(за исключением задания нескольких параметров протокола TCP, таких как максимальный размер буфера и сегмента).

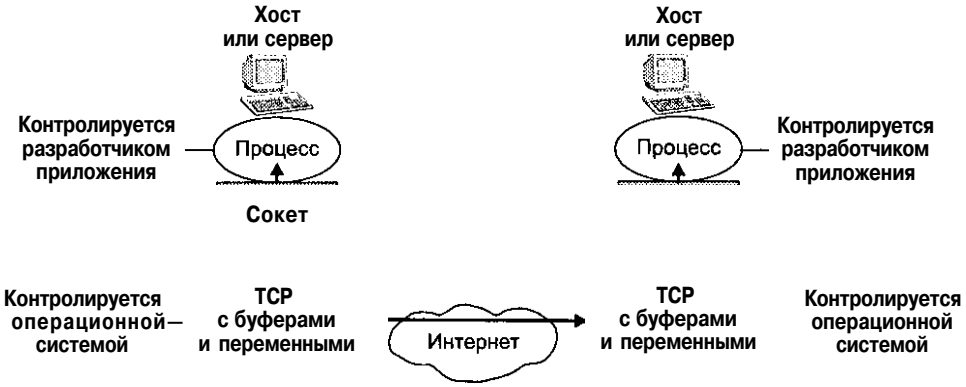


Рис. 2.18. Взаимодействие процессов при помощи TCP-сокетов

Теперь рассмотрим процесс взаимодействия клиентской и серверной программ более подробно. В функции клиента входит инициирование соединения с сервером, а сервер должен быть готовым к установлению соединения. Это означает, что, во-первых, программа-сервер должна быть запущена раньше, чем клиент сделает попытку установить соединение, и, во-вторых, что сервер должен располагать сокетом, с помощью которого устанавливается соединение.

Когда серверный процесс запущен, клиент может инициировать установку TCP-соединения с сервером. Первым действием клиентской программы является создание сокета, при этом программа указывает адрес серверного процесса, состоящий из IP-адреса и номера порта процесса. После создания сокета клиентская сторона протокола TCP осуществляет процедуру тройного рукопожатия с сервером, оканчивающуюся установлением соединения. Заметим, что процедура рукопожатия никак не сказывается на работе приложения.

В ходе тройного рукопожатия клиентский процесс стучит во входную дверь серверного процесса. Когда сервер слышит стук, он создает новую дверь (то есть новый сокет), относящуюся к текущему клиенту. В примере, который последует ниже, входной дверью является объект ServerSocket с именем welcomeSocket. Когда клиент стучит в эту дверь, вызывается метод accept() объекта welcomeSocket, создающий новую дверь для клиента. По окончании процедуры рукопожатия устанавливается TCP-соединение между сокетом клиента и новым сокетом сервера, который называют *сокетом соединения*.

С точки зрения приложения TCP-соединение является прямым виртуальным каналом между сокетами соединения клиента и сервера. Клиент может осуществлять передачу любых байтов через свой сокет, при этом протокол TCP гарантирует, что сервер получит эти байты через свой сокет без искажений и в том же порядке, в каком они были переданы. Подобно тому как люди могут входить и выходить

через одни и те же двери, клиент и сервер способны с помощью сокетов осуществлять прием и передачу информации. Рисунок 2.19 иллюстрирует сказанное.

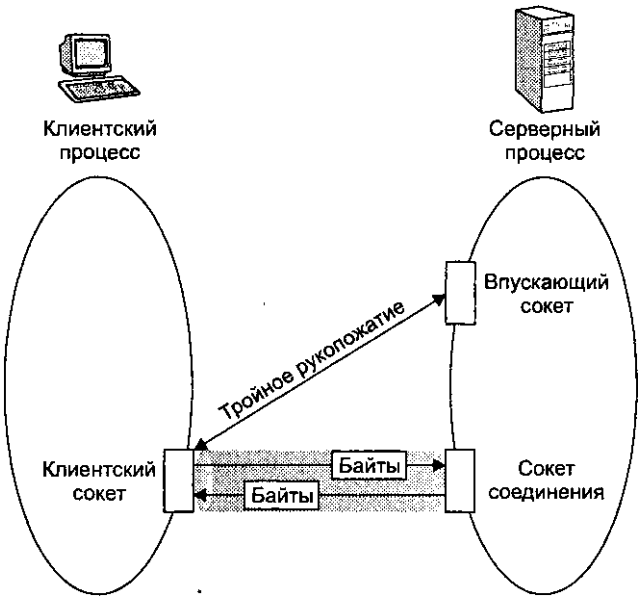


Рис. 2.19. Клиентский сокет, впускающий сокет и сокет соединения

Поскольку сокеты играют центральную роль в работе приложений клиент/сервер, разработку таких приложений часто называют программированием сокетов. Перед тем как привести пример первого приложения клиент/сервер, мы бы хотели рассмотреть понятие *потока данных.* Под потоком данных понимается последовательность символов, передающаяся между двумя процессами. По отношению к процессу выделяют *входной* и *выходной* потоки. Входной поток процесса связывается с некоторым входным устройством (клавиатурой, сокетом и т. д.), а выходной поток — с некоторым выходным устройством (монитором, сокетом и т. д.).

Пример приложения клиент/сервер на языке Java

С помощью следующего приложения мы осветим вопросы программирования сокетов как для протокола TCP, так и для протокола UDP. Приложение функционирует следующим образом.

1. Клиент считывает со стандартного устройства ввода (клавиатуры) строку символов и посылает эту строку серверу через свой сокет.
2. Сервер принимает строку через свой сокет.
3. Сервер переводит все символы строки в верхний регистр.
4. Сервер отсылает модифицированную строку клиенту.
5. Клиент получает строку и печатает ее с помощью стандартного устройства вывода (монитора).

Мы начнем с рассмотрения приложения, в котором взаимодействие клиента и сервера осуществляется через логическое соединение по протоколу TCP (рис. 2.20).

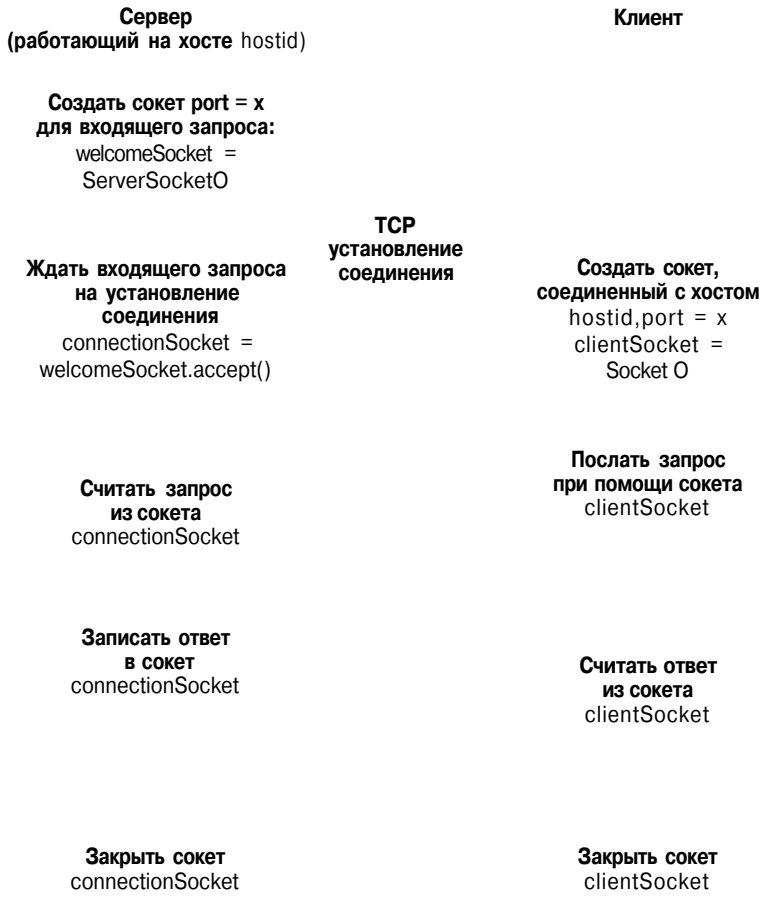


Рис. 2.20. Приложение клиент/сервер, использующее транспортные службы с установлением логического соединения

Ниже мы приведем тексты программ для клиентской и серверной сторон приложения и снабдим пояснениями каждую строку кода. Программа-клиент названа TCPClient.java, программа-сервер — TCPServer.java. Отметим, что приведенные программы лишь иллюстрируют ключевые фрагменты приложения и не претендуют на то, чтобы называться «хорошими» с точки зрения практического применения. Для улучшения программ приведенные тексты следовало бы снабдить несколькими дополнительными строками.

После компиляции обеих программ (каждой на своем компьютере) первой запускается программа-сервер, поскольку для установления соединения серверный процесс должен ожидать запроса со стороны клиентского процесса. Клиентский про-

цесс создается после запуска программы-клиента, и в его функцию входит инициирование TCP-соединения с сервером. После того как соединение установлено, пользователь, находящийся на клиентской стороне, может вводить строки символов и получать их обратно в верхнем регистре.

Ниже приведен текст программы TCPClient.java.

```
* import java.io.*;
  import java.net.*;
  class TCPClient {
    public static void main(String argv[]) throws Exception
    {
     String sentence;
     String modifiedSentence;
     BufferedReader inFromUser =
       new BufferedReader(
           new InputStreamReader(System.in));
     Socket clientSocket = new Socket ("hostname". 6789);
     DataOutputStream outToServer =
       new DataOutputStream(
           cli entSocket.getOutputStream());
     BufferedReader inFromServer =
       new BufferedReader(new InputStreamReader(
           cli entSoket.getInputStream()));
     sentence = inFromUser.readLine();
     outToServer.writeBytes(sentence + '\n');
     modifiedSentence = inFromServer.readLine();
     System.out.print!n("FROM SERVER: " + modifiedSentence);
     clientSocket.close();
    }
  }
```

Как показано на рис. 2.21, программа TCPClient создает три потока данных и один сокет.

Сокет имеет название clientSocket. Поток inFromUser является входным потоком данных программы и связан со стандартным устройством ввода (клавиатурой). Все символы, вводимые с клавиатуры пользователем, попадают в поток inFromUser. Другой входной поток данных программы, inFromServer, связан с сокетом и состоит из символов, передаваемых серверной стороной приложения. Выходной поток данных программы TCPClient имеет название outToServer и также связан с сокетом. Этот поток содержит символы, передающиеся серверу для обработки.

Теперь рассмотрим приведенный код подробнее. Первые две строки содержат имена двух Java-пакетов, ja va л'о и java. net:

```
import java.io.*;
import java.net.*;
```

Пакет java.io содержит классы входных и выходных потоков данных. Обычно этими классами являются BufferedReader и DataOutputStream, которые и были использованы в программе. Пакет java.net хранит классы, поддерживающие работу с компьютерной сетью (Socket и ServerSocket). Объект clientSocket программы порожден классом Socket.

```
class TCPClient {
  public static void main(String argv[]) throws Exception
    {.....}
}
```

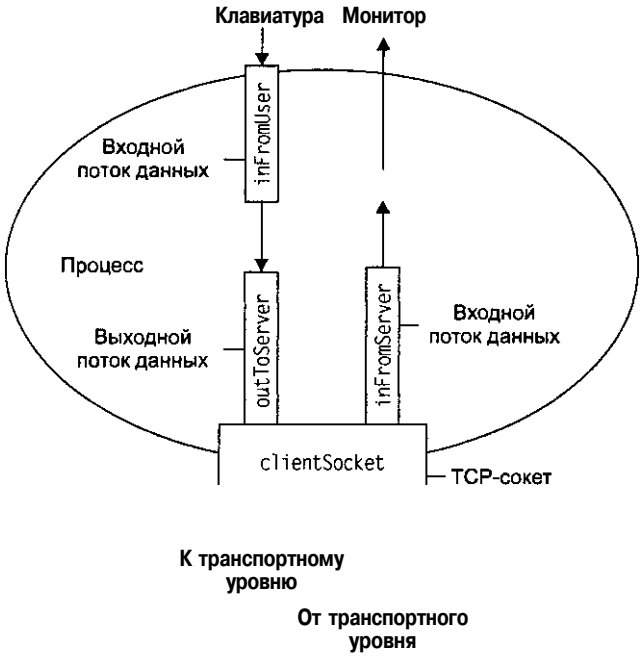


Рис. 2.21. Программа TCPClient организует три потока данных для символов

Конструкции, аналогичные приведенной выше, являются стандартными в практике программирования на Java. Первая строка представляет собой начало блока определения класса. В ней присутствует ключевое слово class и имя определяемого класса TCPClient. Класс может содержать переменные и методы; в объявлении класса они заключены в фигурные скобки и фактически составляют блок определения класса. В классе TCPClient нет переменных, он содержит единственный метод с именем main(). Методы похожи на процедуры и функции языков, подобных C; метод main() также играет роль, схожую с функцией main() в C или C++. Когда интерпретатор Java исполняет приложение (будучи вызванным управляющим классом приложения), он обращается к методу main() этого класса. Метод mainQ вызывает все остальные методы, используемые в приложении. Если в данный момент вы впервые сталкиваетесь с Java-приложением, можете не обращать внимания на ключевые слова public, static, void, main и throws Exception.

```
String sentence;
String modifiedSentence;
```

Приведенные две строки являются объявлениями объектов типа String. Объект sentence предназначен для хранения строки, вводимой пользователем и передаваемой серверу. Объект modifiedSentence содержит строку, принимаемую от сервера и выводимую на стандартное устройство вывода.

```
BufferedReader inFromUser =
  new BufferedReader(new InputStreamReaderCSystem.in));
```

Здесь происходит создание потокового объекта in From User типа BufferedReader. Инициализация потока данных производится объектом System.in, связывающим поток со стандартным устройством ввода. После выполнения этой команды клиенту начинают передаваться символы, вводимые пользователем с клавиатуры.

```
Socket clientSocket = new Socket ("hostname", 6789);
```

В приведенной строке происходит создание объекта clientSocket типа Socket. Кроме того, при этом инициируется TCP-соединение между клиентом и сервером. Слово hostname необходимо заменить именем хоста, сохранив кавычки (например, "fling.seas.upenn.edu"). Перед началом установки TCP-соединения клиент производит DNS-запрос IP-адреса хоста. Значение 6789 — это номер порта; вы можете выбрать другое число, однако необходимо помнить, что клиентская и серверная стороны должны использовать один и тот же номер порта. Как упоминалось ранее, IP-адрес хоста в совокупности с номером порта приложения идентифицирует серверный процесс.

```
DataOutputStream outToServer =
  new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer =
  new BufferedReader(new InputStreamReader(clientSoket.getInputstream()));
```

Здесь происходит создание потоковых объектов, связанных с сокетом. Поток outToServer обеспечивает вывод программы через сокет, а поток inFromServer предназначен для приема данных от серверной стороны (см. рис. 2.21).

```
sentence = inFromUser. readLineO;
```

Эта строка помещает последовательность символов, вводимых пользователем, в переменную sentence. Ввод оканчивается нажатием пользователем клавиши Enter. Как видим, для помещения символов в переменную используется потоковый объект inFromUser.

```
outToServer.writeBytes(sentence + '\n');
```

Здесь строка sentence, снабженная символом возврата каретки, помещается в выходной поток outToServer. После этого она будет передана через сокет в канал, соединяющий клиента с сервером.

```
modifiedSentence = inFromServer. readLineO;
```

Ответ сервера принимается во входной поток inFromServer, откуда принятая строка копируется в переменную modifiedSentence. Помещение символов в строку modifiedSentence продолжается до тех пор, пока не будет получен символ возврата каретки.

```
System.out.println("FROM SERVER: " + modifiedSentence);
```

Здесь происходит вывод принятой от сервера строки на монитор,

```
clientSocket.close();
```

Эта строка закрывает сокет, а следовательно, и TCP-соединение между клиентом и сервером. Как будет показано в главе 3, исполнение этой команды ведет к отправке TCP-клиентом сообщения TCP-серверу.

Ниже приведен текст программы TCPServer.java.

```
import java.io.*;
import java.net.*;
```

```
class TCPServer {
  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;
     ServerSocket welcomeSocket = new ServerSocket (6789);
     while (true) {
      Socket connectionSocket = welcomeSocket.
        acceptO;
      BufferedReader inFromClient =
        new BufferedReader(new InputStreamReaderC
         connectionSocket.getInputStream()));
      DataOutputStream outToClient =
        New DataOutputStreamC
          connecti onSocket.getOutputStream());
      clientSentence = inFromClient. readLineO;
      capitalizedSentence =
          clientSentence.toUpperCaseO + '\n';
      outToClient.writeBytes(capitalizedSentence);
     }
    }
}
```

Программа TCPServer имеет много общего с программой TCPClient; рассмотрим ее подробнее, опустив строки, общие с программой TCPClient и описанные выше.

```
ServerSocket welcomeSocket = new ServerSocket (6789);
```

Здесь происходит создание объекта welcomeSocket типа ServerSocket. Объект welcomeSocket представляет собой впускающий сокет, с помощью которого клиент устанавливает первоначальный контакт с сервером. Для этого сокета используется порт с номером 6789, совпадающим с номером порта сокета клиента (мы раскроем причины совпадения номеров портов сокетов клиента и сервера в главе 3). Протокол TCP устанавливает прямой виртуальный канал между сокетом clientSocket на клиентской стороне и connectionSocket на серверной стороне, после чего клиент и сервер могут свободно осуществлять обмен информацией. После создания сокета connectionSocket сервер может вновь использовать сокет welcomeSocket для инициирования других соединений с клиентами (приведенная программа не обрабатывает новых соединений, однако ее можно модифицировать соответствующим образом).

Далее программа создает несколько потоковых объектов, аналогичных clientSocket.

```
capitalizedSentence = clientSentence.toUpperCaseO + '\n';
```

Эта команда основная в приложении. Она выполняет получение строки, передаваемой клиентом, перевод строки в верхний регистр с помощью метода tollpperCase() и добавление в конец символа возврата каретки. Все остальные команды программы являются периферийными и не касаются взаимодействия сервера с клиентом.

Для того чтобы протестировать совместную работу наших программ, следует поместить их на разные хосты и указать в программе TCPClient имя хоста-сервера. Затем необходимо запустить программу TCPServer, чтобы создать серверный процесс. Серверный процесс будет находиться в состоянии ожидания до тех пор, пока клиентский процесс не инициирует TCP-соединение; для этого, в свою очередь, нужно запустить программу TCPClient. Наконец, чтобы проверить наше приложение

в действии, следует ввести на стороне клиента строку, оканчивающуюся символом возврата каретки (он вводится нажатием клавиши Enter).

Используя приведенные тексты программ, вы можете создать собственное приложение клиент/сервер. Например, вы можете изменить работу сервера таким образом, что вместо перевода строки в верхний регистр он будет подсчитывать количество букв «s» в ней.

Программирование UDP-сокетов

Как было показано в предыдущем разделе, с точки зрения процессов взаимодействие по протоколу TCP выполняется через канал, который существует до тех пор, пока один из процессов не закроет его. Передача процессом информации сводится к тому, что процесс «сбрасывает» байты непосредственно в канал, при этом нет необходимости снабжать байты адресом назначения, поскольку канал логически связан с адресатом. Кроме того, передача по каналу является надежной, то есть принимаемая последовательность байтов в точности соответствует передаваемой последовательности.

Протокол UDP, как и TCP, реализует взаимодействие между двумя процессами, выполняющимися на разных хостах; тем не менее UDP-взаимодействие принципиально отличается от TCP-взаимодействия. Во-первых, логическое соединение между хостами отсутствует, поскольку не существует процедуры рукопожатия, в ходе которой устанавливался бы канал. Во-вторых, любая передаваемая последовательность байтов должна снабжаться адресом назначения, представляющим собой совокупность IP-адреса хоста и номера порта процесса. IP-адрес, номер порта и последовательность информационных байтов мы будем называть пакетом. Передачу по протоколу U DP можно сравнить с поездкой на такси: первым делом нужно сообщить водителю адрес дома, к которому мы хотим добраться.

После того как пакет создан, процесс отправляет его адресату через свой сокет. Транспортный уровень (протокол UDP) предпринимает действия для доставки пакета по адресу назначения, однако не дает относительно доставки никаких гарантий. Другими словами, протокол UDP обеспечивает ненадежную передачу данных между процессами.

В этом разделе в качестве примера мы модифицируем предыдущее приложение для протокола UDP. Как вы увидите, коды программ для протоколов TCP и UDP значительно различаются. Во-первых, из-за отсутствия процедуры рукопожатия нет необходимости во впускающем сокете; во-вторых, входные и выходные потоки данных не связаны с сокетами; в-третьих, при пересылке каждой группы байтов указывается IP-адрес хоста-получателя и номер порта сервера; в-четвертых, принимающей стороне приходится выделять информационные байты из принятых пакетов. Напомним порядок действий, выполняемых нашим приложением.

1. Клиент считывает со стандартного устройства ввода (клавиатуры) строку символов и посылает эту строку серверу через свой сокет.
2. Сервер принимает строку через свой сокет.

3. Сервер переводит все символы строки в верхний регистр.
4. Сервер отсылает модифицированную строку клиенту.
5. Клиент получает строку и печатает ее с помощью стандартного устройства вывода (монитора).

На рис. 2.22 приведена схема взаимодействия клиента и сервера при использовании протокола UDP.

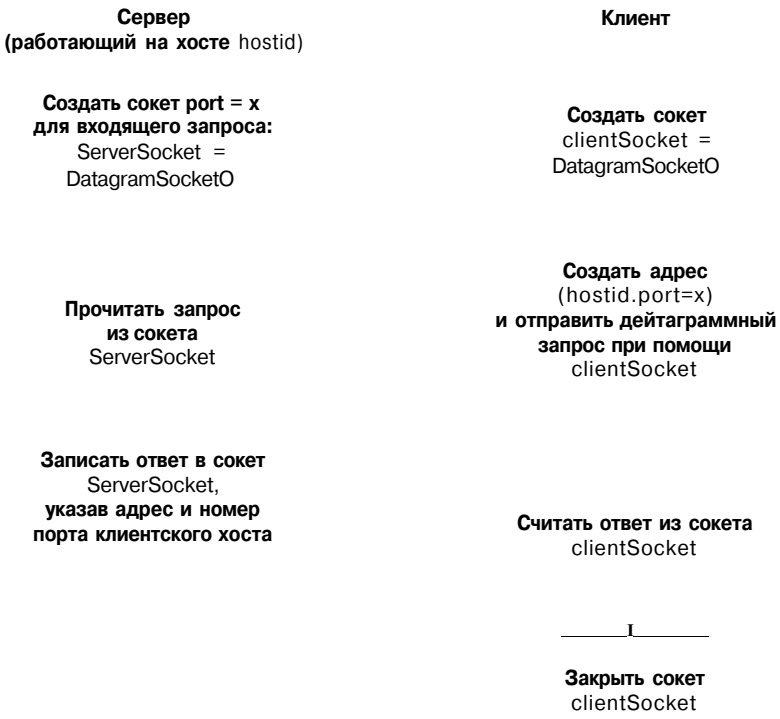


Рис. 2.22. Приложение клиент/сервер, использующее транспортные службы без установления логического соединения

Ниже следует текст программы UDPClient.java.

```
import java.io.*;
import java.net.*;
class UDPClient {
  public static void main(String args[]) throws Exception
  {
   BufferedReader inFromUser =
     new BufferedReader(new InputStreamReader
          (System.in));
   DatagramSocket clientSocket = new DatagramSocketO;
   InetAddress IPAddress =
          InetAddress.getByName("hostname");
   byte[] sendData = new byte[1024];
```

```
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readl_ine();
        sendData = sentence.getBytesO;
        DatagramPacket sendPacket =
          new DatagramPacket(sendData. sendData.length.
                IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket -
          new DatagramPacket(receiveData.
                receiveData.length);
        cl i entSocket.recei ve(recei vePacket);
        String modifiedSentence =
          new String(receivePacket.getDataO);
        System.out.print!n("FROM SERVER;" + modifiedSentence);
        cl i entSocket. closeO;
      }
    }
```

Как показано на рис. 2.23, программа UDPClient создает один поток данных и один сокет. Сокет имеет тип DatagramSocket и имя clientSocket. Обратите внимание, что при использовании протоколов UDP и TCP типы сокетов различаются: в случае TCP обычно применяют тип Socket. Поток inFromUser является входным для программы и связан со стандартным устройством ввода (клавиатурой). Когда пользователь вводит символы, они попадают в программу через входной поток данных. Аналогичным образом входной поток был определен в программе TCPClient. Следует отметить одну важную деталь: при использовании протокола UDP ни один из потоков (ни входной, ни выходной) не связан с сокетом. Отправка байтов осуществляется в пакетах с помощью объекта типа DatagramSocket.

Теперь более подробно рассмотрим строки кода программы UDPClient.

```
DatagramSocket clientSocket = new DatagramSocketO;
```

Эта команда создает объект clientSocket типа DatagramSocket, однако в отличие от программы TCPClient не инициирует установление TCP-соединения. По этой причине конструктор DatagramSocketO не принимает в качестве аргументов имя хоста сервера и номер порта. Таким образом, действие приведенной строки сводится лишь к созданию сокета клиента.

```
InetAddress IPAddress = InetAddress.getByName("hostname");
```

Для того чтобы иметь возможность передавать удаленному процессу информацию, необходимо располагать его адресом. Как мы знаем, частью адреса процесса является IP-адрес хоста, на котором он выполняется. Данная строка создает DNS-запрос IP-адреса хоста с именем hostname (как и в программе TCPClient, слово hostname необходимо заменить реальным именем хоста). Фактически работа с DNS-запросом осуществляется методом getByName(), который принимает в качестве аргумента имя хоста и возвращает полученный IP-адрес. IP-адрес помещается в объект IPAddress типа InetAddress.

```
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
```

Байтовые массивы sendData и receiveData предназначены для хранения передаваемой и принимаемой информации соответственно.

```
sendData = sentence.getBytesO;
```

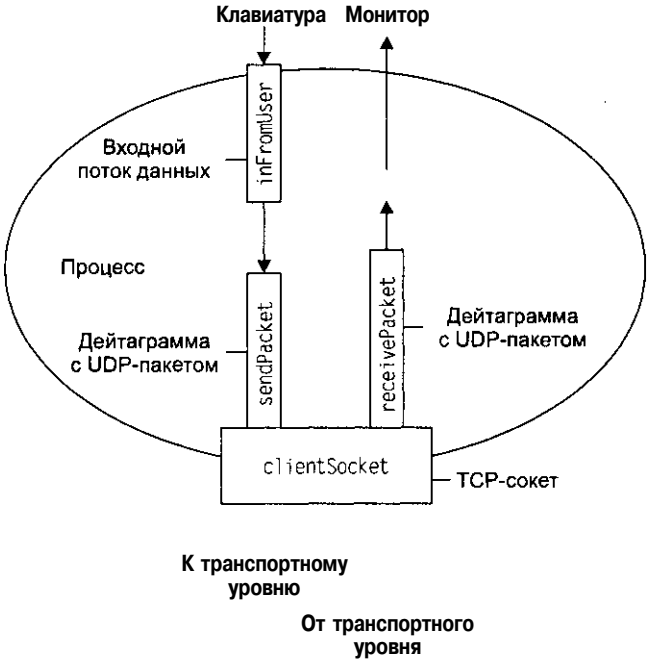


Рис. 2.23. В программе UDPClient имеется единственный поток данных; сокет принимает и передает пакеты процессу

Строка sentence с помощью метода getBytes() преобразуется в массив байтов, который копируется в переменную sendData.

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress. 9876);
```

Здесь происходит создание пакета sendPacket, предназначенного для отправки серверу через сокет. В пакет входят данные, содержащиеся в переменной sendData, размер этих данных, IP-адрес хоста назначения и номер порта приложения (в данном случае мы выбрали номер 9876). Обратите внимание на то, что пакет имеет тип DatagramPacket.

```
clientSocket.send(sendPacket);
```

Метод send() объекта clientSocket отправляет пакет sendPacket (как упоминалось ранее, clientSocket представляет собой клиентский сокет приложения). Обратимся к механизмам передачи данных, осуществляемым протоколами TCP и UDP. В случае TCP строка символов включается в выходной поток, имеющий логическую связь с сервером. В случае UDP нам необходимо создать пакет, содержащий адрес сервера. После передачи пакета клиент ожидает приема пакета от сервера.

```
DatagramPacket receivePacket =
  new DatagramPacket(receiveData. receiveData.1ength);
```

Здесь клиент создает переменную для хранения пакета receive Packet типа Datagram Packet.

```
cli entSocket.recei ve(receivePacket):
```

Клиент находится в состоянии ожидания до тех пор, пока не будет начат прием пакета от сервера; принимаемые символы помещаются в переменную receivePacket.

```
String modifiedSentence =
  new String(recei vePacket.getData());
```

В этой строке происходит извлечение данных из объекта receivePacket и преобразование типа из массива байтов в строку с последующим присваиванием переменной modifiedSentence.

```
System.out.println("FROM SERVER:" + modifiedSentence);
```

Эта строка, присутствовавшая ранее в программе TCPClient, печатает на экране клиента строку modifiedSentence.

```
clientSocket.closeO;
```

Вызов метода close() объекта clientSocket приводит к закрытию сокета. Поскольку протокол UDP не использует логическое соединение, при закрытии сокета клиент не посылает серверу сообщений транспортного уровня (как это было в случае TCP).

Теперь ознакомимся с текстом программы UDPServer.java, представляющей серверную часть нашего приложения.

```
import java.io.*;
import java.net.*:
class UDPServer {
  public static void main(String args[]) throws Exception
    {
    DatagramSocket ServerSocket = new DatagramSocket(9876);
    byte[] receiveData = new byte[1024];
    byte[] sendData - new byte[1024]:
   while(true)
      {
        DatagramPacket receivePacket =
            new DatagramPacket(receiveData. receiveData.length);
        ServerSocket.receive(receivePacket);
        String sentence = new String(receivePacket.getDataO);
        InetAddress IPAddress = receivePacket.getAddress();
        int port = recei vePacket. get Port 0;
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capital izedSentence.getBytesO;
        DatagramPacket sendPacket =
        new DatagramPacket(sendData. sendData.length. IPAddress. port);
        ServerSocket.send(sendPacket);
    }
  }
}
```

Как показано на рис. 2.24, программа UDPServer создает единственный сокет с именем ServerSocket типа DatagramSocket; этот же тип использовался нами в программе-клиенте приложения. Аналогично с сокетом не связан ни один из потоков данных.

Рассмотрим некоторые из строк приведенного кода.

```
DatagramSocket ServerSocket = new DatagramSocket(9876);
```

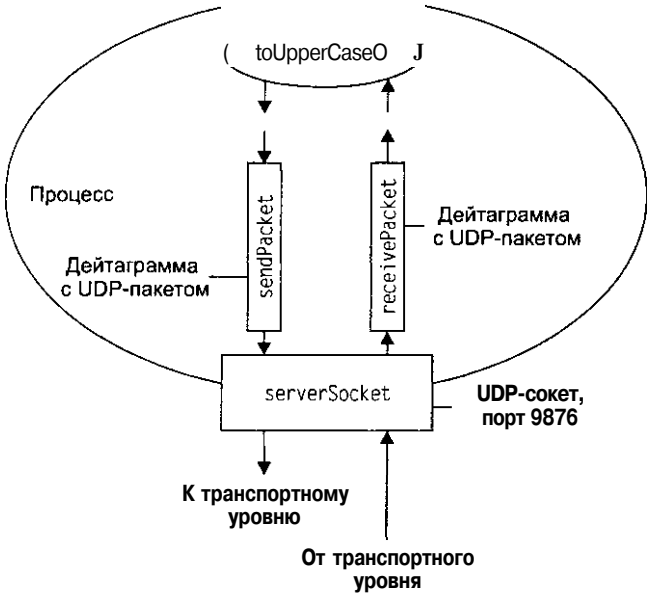


Рис. 2.24. Программа UDPServer не имеет потоков; сокет принимает и передает пакеты процессу

Эта строка создает объект ServerSocket типа DatagramSocket с номером порта 9876. Передача и прием всех данных сервером осуществляется при помощи этого объекта. Поскольку протокол UDP не предполагает установления логического соединения, у нас нет необходимости поддерживать отдельный сокет для каждого клиента и специальный впускающий сокет для рукопожатий с новыми клиентами, как для протокола TCP. В случае нескольких клиентов все они будут осуществлять обмен данными с сервером через единственный сокет ServerSocket.

```
String sentence = new String(receivePacket.getDataO);
InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();
```

Вышеприведенные строки осуществляют распаковку пакета, переданного клиентом. Первая команда извлекает из пакета данные и помещает их в строковый объект sentence (аналогичная команда есть в программе UDPClient). Следующая команда извлекает IP-адрес, а последняя, третья команда — номер порта сервера (в общем случае номер порта отличается от 9876). Мы подробно рассмотрим номера портов в следующей главе. Извлечение IP-адреса и номера порта обусловлено необходимостью идентификации клиента для передачи модифицированной строки.

Мы полностью рассмотрели пару программ, составляющих UDP-приложение. Для того чтобы протестировать приложение, необходимо поместить программы на разные хосты, а затем скомпилировать и запустить их. Обратите внимание, что в данном случае клиентскую часть приложения можно запустить раньше, чем серверную. Это объясняется тем, что после запуска клиент не предпринимает попыток

установить соединение с сервером. Запустив программу-клиент, вы можете начать ввод символов строки, предназначенной для удаленной обработки.

Разработка простого web-сервера

Теперь, когда мы ознакомились с некоторыми деталями протокола HTTP и создали на языке Java два приложения архитектуры клиент/сервер, постараемся связать полученные знания и разработать web-сервер средствами языка Java. Как вы убедитесь чуть позже, эта задача не представляет сложности.

Сервер, который мы собираемся построить, будет выполнять следующие функции.

1. Прием и обработка единственного HTTP-запроса.
2. Извлечение требуемого файла с помощью файловой системы сервера.
3. Создание ответного HTTP-сообщения, включающего строки заголовка и требуемый файл.
4. Передача ответного сообщения непосредственно клиенту.

Программный код, который будет представлен ниже, иллюстрирует лишь основные аспекты работы web-сервера и лишен многих деталей (например, в нем отсутствует обработка исключений); мы сделали его максимально кратким, чтобы вы могли сконцентрировать внимание на ключевых составляющих сервера. Кроме того, программа WebServer.java написана с допущением, что запрашиваемый файл имеется в файловой системе сервера. Ее текст приведен ниже.

```
import java.io.*;
import Java.net.*;
import java.uti1.*;
class Webserver {
   public static void main(String argv[]) throws Exception
   {
     String requestMessageLine;
     String fileName;
     ServerSocket listenSocket = new ServerSocket(6789);
     Socket connecti onSocket = listenSocket.acceptO:
     BufferedReader inFromClient =
        new BufferedReader(
          new InputStreamReader(connectionSocket.getInputStream()));
     DataOutputStream outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());
     requestMessageLine = inFromClient. readLineO;
     StringTokenizer tokenizedLine =
       new StringTokenizer(requestMessageLine);
     if   (tokenizedLine.nextToken().equals("GET")){
       fileName = tokenizedLine.nextToken();
       if (fileName.startsWithCV") == true )
            fileName = fileName.substring(l);
       File file = new File(fileName);
       int numOfBytes = (int) file.length();
       FileInputStream inFile = new FileInputStream (fileName);
       byte[] fileInBytes = new byte[numOfBytes]:
       inFile.read(fileInBytes):
```

но осуществить. Для создания запроса используйте браузер, однако помните, что сервер использует нестандартный номер порта 6789, который должен быть указан в запросе. Например, если сервер находится на хосте с именем somehost.somewhere.edu, а запрашиваемый файл имеет имя somefile.html, то в адресной строке браузера следует ввести следующий запрос:

http://somehost.somewhere.edu:6789/somefi1e.html

Распределение ресурсов

Web-ресурсы очень богаты и продолжают непрерывно пополняться. Это web-страницы (содержащие текст, изображения, Java-апплеты, фреймы и т. д.), музыкальные файлы в формате MP3, записанное потоковое аудио и видео, виртуальные миры. Ресурсы распределены между огромным количеством серверов, разбросанных по всему миру, и доступны миллионам пользователей.

Протокол HTTP является средством, позволяющим любому пользователю получить любой объект независимо от того, сколькими тысячами километров измеряется расстояние между хостом пользователя и удаленным сервером и сколько Интернет-провайдеров находится на пути запроса. Тем не менее время доступа к web-ресурсам иногда бывает весьма значительным. Ниже перечислены некоторые причины такой ситуации.

- На пути объекта к хосту пользователя имеются низкоскоростные линии связи, что приводит к значительным задержкам передачи. Как было показано в главе 1, задержка передачи для линии связи может быть рассчитана как L/R, где L представляет объем объекта, а R — скорость передачи по линии связи.
- На пути объекта находится хотя бы один перегруженный узел, в котором велико значение задержки ожидания и имеет место потеря пакетов (см. раздел «Задержки и потери данных в сетях с коммутацией пакетов» в главе 1). Перегрузки (приближение коэффициента интенсивности трафика к 1) могут происходить даже в тех случаях, когда входы узла представляют собой высокоскоростные линии связи.
- Web-сервер, которому адресован запрос, перегружен, и время ожидания обслуживания запроса может быть весьма значительным.

Для решения проблемы задержек используется нехитрый прием: один и тот же ресурс располагают на нескольких серверах, и запрос переадресуется «наилучшему» серверу. Для web-страницы или MP3-файла «наилучшим» будет тот сервер, время выполнения запроса которым минимально. Зачастую такой сервер принадлежит наиболее близкому к пользовательскому хосту Интернет-провайдеру.

Распределение ресурсов подразумевает механизмы дублирования ресурсов, а также способы определения хостами серверов, наиболее подходящих для выполнения запросов. Во второй половине 1990-х годов средства распределения ресурсов получили широкое распространение; в настоящее время они активно применяются, особенно в сфере аудио- и видеоинформации. Существуют несколько крупных компаний, занимающихся распределением ресурсов. Компании Cisco, Lucent, Inktomi и CacheFlow разрабатывают соответствующее аппаратное и программное обеспечение, а Akamai, Digital Island и AT&T реализуют услуги распределения ресур-

сов компаниям-поставщикам ресурсов, таким как Yahoo! и CNN. Распределение ресурсов является полем для активных исследований как с научной, так и с промышленной точек зрения.

За прошедшие годы инженеры и исследователи предложили множество решений, касающихся распределения ресурсов. Эти решения можно приближенно разделить на три группы: web-кэширование, сети распределения ресурсов (Content Distribution Networks, CDN) и одноранговое разделение файлов. Ниже мы рассмотрим каждую из технологий, однако сначала немного уточним терминологию. *Поставщиком ресурсов* мы будем считать любое лицо, организацию или компанию, которые располагают ресурсом, доступным для пользователей Интернета. Под *сервером-источником* объекта будет подразумеваться сервер, на котором первоначально находился объект и где всегда можно найти копию этого объекта.

Web-кэширование

Web-кэш, часто называемый *прокси-сервером,* представляет собой сеть, которая выполняет HTTP-запросы от имени сервера-источника. Web-кэш имеет собственное дисковое устройство хранения информации, содержащее ранее запрашивавшиеся копии объектов. Как показано на рис. 2.25, браузер пользователя можно настроить таким образом, чтобы все создаваемые HTTP-запросы сначала направлялись в web-кэш (данная процедура в браузерах Microsoft и Netscape выполняется очень просто).

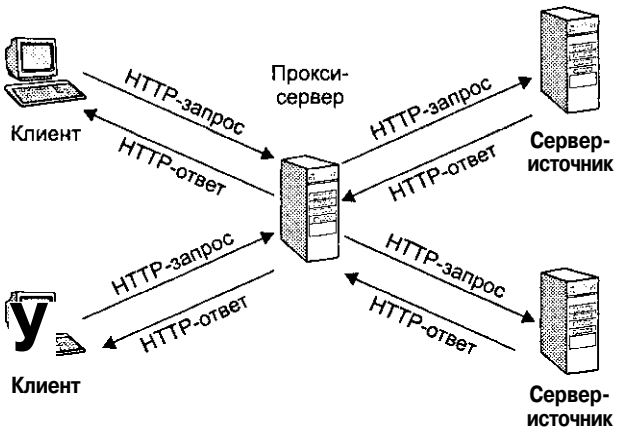


Рис. 2.25. Клиенты, запрашивающие объекты в web-кэше

После того как браузер настроен указанным образом, любой запрашиваемый объект сначала ищется в web-кэше. В качестве примера предположим, что браузер запрашивает объект http://www.someschool.edu/campus.gif. Алгоритм использования кэш-сервера выглядит следующим образом.

1. Браузер устанавливает TCP-соединение с кэш-сервером и посылает ему запрос объекта.
2. Кэш-сервер проверяет наличие локальной копии требуемого объекта и в случае ее обнаружения формирует ответное сообщение и отсылает объект браузеру.

3. Если локальная копия отсутствует, кэш-сервер устанавливает TCP-соединение с сервером-источником http://www.someschool.edu и посылает ему запрос на получение объекта. Сервер-источник обрабатывает запрос и отсылает требуемый объект кэш-серверу.
4. После получения объекта кэш-сервер сохраняет его копию на локальном накопителе информации и передает объект браузеру через открытое ранее TCP-соединение.

Обратите внимание на то, что web-кэш может играть роль как клиента, так и сервера: он является сервером при взаимодействии с браузерами и клиентом при взаимодействии с серверами-источниками.

Обычно кэш-серверы арендуются и устанавливаются Интернет-провайдерами. Например, университет может создать кэш-сервер в своей локальной сети и выполнить конфигурирование всех браузеров так, чтобы они обращались к кэш-серверу.

Web-кэширование является формой распределения ресурсов, поскольку дублирует объекты серверов-источников и организует доступ пользователей к локальным копиям объектов. Обратите внимание на то, что поставщик ресурсов никак не влияет на процесс дублирования; напротив, дублирование зависит лишь от запросов пользователей.

Кэширование получило широкое распространение в Интернете по трем причинам. Первая заключается в том, что кэш-серверы способны значительно сократить время выполнения запроса пользователя, в особенности если скорость передачи между пользователем и кэш-сервером превышает скорость передачи между пользователем и сервером-источником. Зачастую для соединения пользователя с кэш-сервером используются высокоскоростные линии связи, поэтому при наличии на кэш-сервере требуемого объекта его доставка пользователю происходит за очень короткое время. Вторая причина популярности механизма кэширования состоит в том, что он способен значительно снизить трафик между локальными сетями и Интернетом. Это позволяет, в свою очередь, сократить расходы на дорогостоящие линии связи, соединяющие локальные сети с Интернетом. Кроме того, значительное сокращение трафика при кэшировании происходит и в Интернете в целом, приводя к лучшему качеству обслуживания приложений всех пользователей глобальной Сети. Наконец, третья причина успеха кэширования заключается в том, что оно позволяет с большой скоростью распространять ресурсы среди пользователей. Даже в случае, если поставщик использует недорогое низкоскоростное сетевое оборудование, наиболее популярные ресурсы в скором времени окажутся в web-кэшах, и, следовательно, пользователи смогут загружать их с приемлемым качеством обслуживания.

Чтобы лучше оценить достоинства кэширования, обратимся к примеру. На рис. 2.26 изображены две компьютерные сети: локальная высокоскоростная университетская сеть и Интернет. Маршрутизаторы локальной сети и Интернета соединены между собой линией связи, обеспечивающей скорость передачи 1,5 Мбит/с. Серверы-источники имеют прямое соединение с Интернетом, однако находятся в разных частях земного шара. Предположим, что средний размер объектов равен

100 000 бит, а средняя частота запросов браузеров университетских компьютеров к web-серверам-источникам составляет 15 раз в секунду. Будем считать, что размер HTTP-запросов пренебрежимо мал и не создает дополнительного трафика на линии связи между локальной сетью и Интернетом. Предположим также, что среднее время, проходящее с момента передачи запроса (в виде одной IP-дейтаграммы) первым маршрутизатором Интернета, показанным на рисунке, до получения им ответа (как правило, в виде нескольких дейтаграмм), составляет 2 с. Эту величину мы неформально назовем «задержкой Интернета».

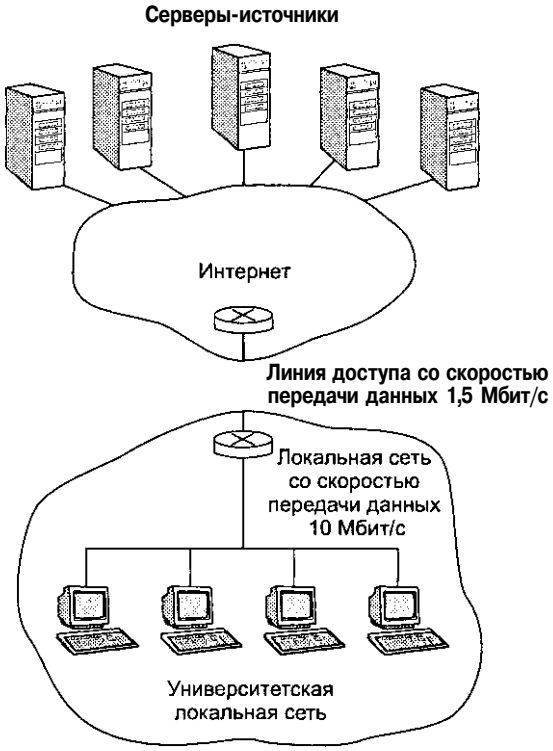


Рис. 2.26. Соединение между университетской сетью и Интернетом

Полное время ответа, то есть время, проходящее с момента формирования запроса браузером до завершения приема требуемого объекта, складывается из трех составляющих: задержки локальной сети, времени доступа (задержки между двумя указанными выше маршрутизаторами) и задержки Интернета. Проведем грубую оценку времени ответа. Интенсивность трафика локальной сети (см. раздел «Задержки и потери данных в сетях с коммутацией пакетов» в главе 1) составляет:

(15 запросов в секунду) x (100 Кбит/запрос) / (10 Мбит/с) = 0,15,

а интенсивность трафика на линии доступа равна:

(15 запросов в секунду) x (100 Кбит/запрос) / (1,5 Мбит/с) = 1.

Интенсивность трафика в локальной сети, составляющая 0,15, способна привести к задержкам, не превышающим десятков миллисекунд. С другой стороны, единичная интенсивность на линии доступа может привести к неограниченному росту задержек. Таким образом, среднее время ответа в нашем примере может легко достигать нескольких минут и более, что, очевидно, неприемлемо с точки зрения качества обслуживания пользователей.

Одним из возможных решений проблемы является увеличение пропускной способности линии доступа, например до 10 Мбит/с. Это приведет к снижению интенсивности трафика на линии до 0,15, тем самым уменьшив задержку доступа до пренебрежимо малых значений. В этом случае время ответа будет приблизительно равно задержке Интернета и составит 2 с. К сожалению, подобное решение требует немалых финансовых затрат.

Теперь рассмотрим, что произойдет, если вместо увеличения скорости передачи по линии доступа мы установим кэш-сервер в университетской сети. Это решение иллюстрирует рис. 2.27.

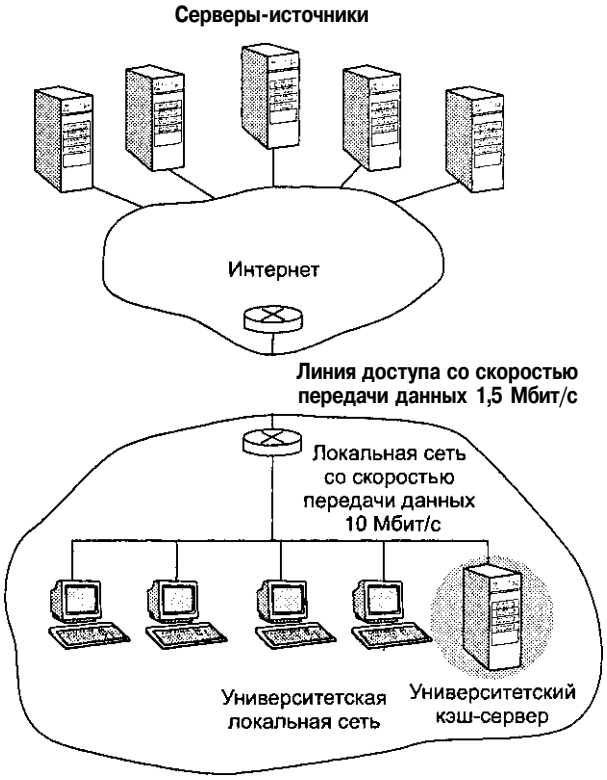


Рис. 2.27. Добавление кэш-сервера в университетскую сеть

На практике вероятность того, что кэш способен удовлетворить запрос пользователя, лежит в интервале от 0,2 до 0,7; мы положим эту вероятность равной 0,4.

Поскольку пользователи соединены с кэш-сервером высокоскоростными линиями связи локальной сети, время ответа для запросов, выполняемых кэш-сервером, очень мало; мы будем считать его равным 10 миллисекундам. Указанное время ответа соответствует значению в 40 % от общего числа запросов; оставшиеся 60 % запросов будут вынуждены обслуживаться серверами-источниками. Тем не менее это позволяет снизить коэффициент интенсивности трафика на линии доступа с 1 до 0,6 и решает проблему перегрузки. Как правило, коэффициент интенсивности ниже 0,8 для линии связи с пропускной способностью 1,5 Мбит/с приводит к незначительным задержкам, не превышающим десятков миллисекунд. Учитывая двухсекундное значение задержки Интернета, мы можем пренебречь временем доступа, и среднее время ответа составит:

$$0{,}4 \times (0{,}01 \text{ с}) + 0{,}6 \times (2{,}01 \text{ с}) = 1{,}21 \text{ с}.$$

Таким образом, применение кэш-сервера дает лучшие результаты, чем увеличение пропускной способности линии доступа, и не требует замены сетевого оборудования. Разумеется, аренда и установка кэш-сервера не является бесплатной, однако расходы университета в случае замены линии доступа были бы значительно выше. Отметим, что для создания web-кэша вполне достаточно недорогого персонального компьютера и, кроме того, для кэш-серверов существует бесплатное программное обеспечение.

Совместное кэширование

Несколько территориально распределенных кэш-серверов могут объединяться и функционировать совместно. Например, локальный кэш-сервер организации можно настроить таким образом, чтобы он в случае необходимости перенаправлял запросы кэш-серверу магистрального Интернет-провайдера. Это обеспечит двухступенчатый кэш-поиск, причем обращение к серверу-источнику будет производиться только в том случае, если требуемоый объект не обнаружится ни на одном из кэш-серверов. Если объект будет найден на магистральном кэш-сервере, последний передаст его пользователю через локальный кэш-сервер, который сохранит копию объекта на своем дисковом пространстве. Достоинством кэша высокого уровня является большее количество пользователей, а следовательно, большее количество хранящихся в кэше объектов и соответствующее повышение скорости их обнаружения.

Примером системы совместного кэширования является служба JWCS (Janet Web Caching Service — служба web-кэширования Жанет), разработанная для научных организаций Великобритании [510]. Более 200 локальных кэш-серверов научных организаций направляют свои запросы национальному кэш-серверу, обрабатывающему около 100 миллионов транзакций в день. Взаимодействие внутри системы осуществляется при помощи протоколов HTTP и ICP (Internet Caching Protocol — протокол Интернет-кэширования). ICP представляет собой протокол прикладного уровня, описанный в документе RFC 2186 и позволяющий кэш-серверу обратиться к другому кэш-серверу для быстрого поиска требуемого объекта. При наличии искомого объекта осуществляется его передача по протоколу HTTP. Протокол ICP активно используется в системах совместного кэширования и поддерживает-

ся бесплатно распространяемым популярным приложением для кэш-серверов Squid [478]. Заинтересовавшемуся читателю рекомендуем обратиться к дополнительным источникам информации [300, 431, 553] и документу RFC 2186.

Альтернативным вариантом систем совместного кэширования являются кластеры кэшей, как правило, расположенные внутри одной локальной сети. Объединение отдельных кэшей в кластеры обусловливается тем, что первые не всегда имеют накопители информации достаточного объема или не способны обеспечить обработку трафика. Однако, решая проблему недостаточной мощности отдельных кэш-серверов, кластерная система порождает проблему выбора: какому из кэшей кластера следует направлять запрос объекта. Эта проблема решается путем маршрутизации с использованием хеш-функций (если вы сталкиваетесь с этими вопросами впервые, можете ознакомиться с ними более подробно в главе 7). В наиболее простой форме подобной маршрутизации браузер выполняет хэширование URL-адреса запрашиваемого объекта и на основе полученного результата формирует запрос с адресом соответствующего кэш-сервера. Поскольку все браузеры используют одну и ту же хэш-функцию, объект никогда не будет содержаться более чем в одном кэше кластера, и при наличии объекта в кластере браузер сможет однозначно определить, в каком именно кэше находится объект. Маршрутизация с помощью хэш-функций лежит в основе протокола маршрутизации кэш-массива (Cache Array Routing Protocol, CARP), используемого в поддерживающих кэширование продуктах компаний Microsoft и Netscape. Заинтересовавшемуся читателю рекомендуем обратиться к дополнительным источникам информации [300,432, 553].

Сети распределения ресурсов

Интернет-провайдеры арендуют и устанавливают кэш-серверы, чтобы повысить качество обслуживания своих пользователей. Как было показано в подразделе «Web-кэширование» данного раздела, применение кэш-серверов способно значительно сократить время доставки наиболее востребованных ресурсов пользователям.

В конце 1990-х годов широкое распространение получила еще одна технология распределения ресурсов — технология CDN (Content Distribution Network — сети распределения ресурсов). В CDN используется иная «бизнес-модель», нежели в web-кэшировании. Если при web-кэшировании потребителями услуг являются Интернет-провайдеры, то в сетях распределения ресурсов на их месте находятся поставщики ресурсов. Тем не менее сети распределения ресурсов преследуют ту же цель — сократить время доставки ресурсов. Обычно CDN-компания функционирует по следующему плану.

1. Компания устанавливает множество (порядка сотен) CDN-серверов, распределенных в Интернете. Как правило, CDN-серверы располагаются в *центрах Интернет-хостинга,* принадлежащих сторонним компаниям (наподобие Exodus и Worldcom) и представляющих собой здания с большим числом хостов внутри. Обычно центры Интернет-хостинга подключены к Интернет-провайдерам нижнего звена и расположены вблизи их сетей доступа.

2. Компания копирует ресурсы своих потребителей на CDN-серверы. Когда потребитель обновляет свои ресурсы, CDN-компания заменяет старое содержимое серверов новым.
3. Компания обеспечивает такое обслуживание, при котором CDN-сервер осуществляет обработку запроса за минимальное время. Для этого выбирается либо такой CDN-сервер, который территориально наиболее близок к пользователю, либо такой, на пути к которому нет перегруженных узлов.

Любопытно отметить, что в создание CDN вовлекаются сразу несколько независимых компаний. Поставщик ресурсов (например, Yahoo!) использует услуги CDN-компании (например, Akamai). В свою очередь, CDN-компания покупает CDN-серверы у их производителей (Inktomi, IBM и др.) и устанавливает эти серверы в центрах Интернет-хостинга, принадлежащих компаниям, предоставляющим услуги Интернет-хостинга (например, Exodus). Таким образом, в создании CDN участвуют как минимум четыре компании, не считая Интернет-провайдеров!

На рис. 2.28 представлена схема взаимодействия между поставщиком ресурсов и CDN-компанией. Сначала поставщик ресурсов определяет объекты, которые он хотел бы сделать распределенными с помощью CDN (остальные объекты могут распространяться без участия CDN). Нужные объекты отсылаются узлу распределения CDN, который, в свою очередь, доставляет их на все CDN-серверы. Специально для этой цели CDN-компании иногда приобретают частные компьютерные сети. В случае, если поставщик ресурсов желает произвести обновление своих объектов, он отсылает их последние версии узлу распределения, откуда они попадают на все CDN-серверы, заменяя устаревшие объекты. Обратите внимание на то, что каждый CDN-сервер содержит объекты, принадлежащие множеству поставщиков ресурсов.

При знакомстве с технологией CDN возникает интересный вопрос. Когда браузер на хосте пользователя формирует запрос на получение объекта (идентифицируемого своим URL-адресом), каким образом браузер определяет, кому отправить этот запрос — серверу-источнику или одному из CDN-серверов? Для решения этого вопроса в CDN используется механизм DNS-перенаправления, указывающий браузерам адрес нужного сервера. Заинтересовавшемуся читателю рекомендуем обратиться к дополнительным источникам информации [256].

Рассмотрим следующий пример. Пусть поставщик ресурсов имеет имя хоста www.foo.com, а CDN-компания — cdn.com. Предположим, что поставщик ресурсов желает распределить свои GIF-файлы при помощи CDN, при этом все остальные файлы, включая базовые HTML-страницы, будут распространяться им (поставщиком) самостоятельно. Для этого поставщик изменяет все свои объекты таким образом, что ссылки на GIF-файлы предваряются фрагментом http://www.cdn.com. Например, ссылка http://www.foo.com/sports/ruth.gif после изменения приобретает вид http://www.cdn.com/www.foo.com/sports/ruth.gif. Когда браузер формирует запрос объекта ruth.gif, происходит следующее.

1. Браузер передает запрос на получение базового HTML-объекта серверу-источнику www.foo.com, который выполняет запрос и отсылает браузеру требуемый файл. Браузер производит обработку HTML-страницы и находит в ней ссылку на объект http://www.cdn.com/www.foo.com/sports/ruth.gif.

2. Браузер осуществляет DNS-поиск хоста www.cdn.com, при этом запрос передается от корневого сервера имен полномочному серверу имен хоста www.cdn.com. Последний извлекает IP-адрес хоста, сделавшего запрос, и на основе специальной внутренней карты Интернета возвращает IP-адрес CDN-сервера, который более других подходит для обслуживания запроса.
3. Ответ с IP-адресом CDN-сервера возвращается браузеру, который перенаправляет свой запрос с использованием полученного IP-адреса. Далее запрос обрабатывается и обслуживается CDN-сервером. Любой последующий запрос, адресованный хосту www.cdn.com, будет автоматически обслуживаться этим же CDN-сервером, поскольку IP-адрес www.cdn.com окажется в DNS-кэше либо на стороне клиента, либо на стороне локального сервера имен.

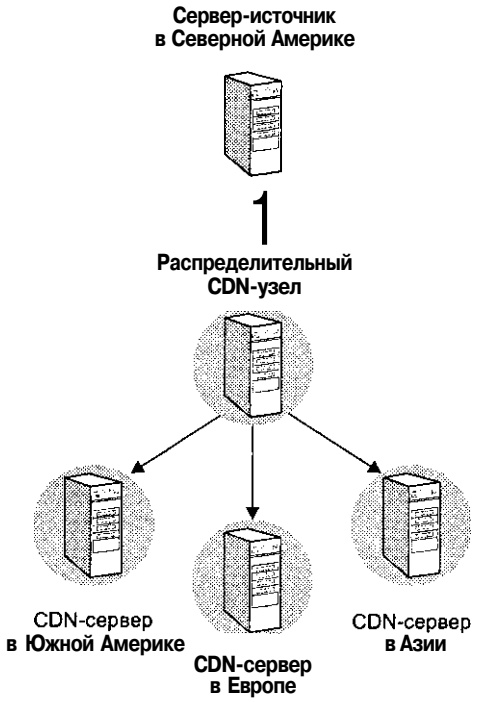


Рис. 2.28. CDN-компания размещает объекты поставщиков ресурсов на CDN-серверах

На рис. 2.29 приведена иллюстрация этапов выполнения запроса на получение документа с помощью CDN. Сначала браузер получает базовый HTML-файл с сервера-источника, затем обращается к полномочному серверу имен, который определяет IP-адрес «наилучшего» CDN-сервера, и, наконец, запрашивает распределенные объекты у CDN-сервера. Обратите внимание на то, что приведенная схема не требует внесения изменений в протоколы HTTP и DNS или применения каких-либо дополнительных средств.

Пожалуй, единственным оставшимся вопросом, который следует рассмотреть детально, является методика определения «наилучшего» CDN-сервера для хоста,

формирующего запрос. Несмотря на то что каждая CDN-компания располагает собственным алгоритмом решения этой задачи, нетрудно выделить общую идею подобных решений. Каждому Интернет-провайдеру нижнего звена (то есть группе оконечных пользователей, подключенных к сети такого провайдера) компанией ставится в соответствие CDN-сервер, который определяется на основе анализа таблиц маршрутизации (точнее, BGP-таблиц, рассматриваемых в главе 4), времени кругового оборота и прочих данных о передаче между различными сетями доступа. Подробности вы можете найти в [535]. Оценив приближенные значения времени ответа для каждой сети доступа, можно выбрать сеть с минимальным временем. Этой сети и назначается определенный CDN-сервер, а соответствующая информация заносится в полномочный DNS-сервер.

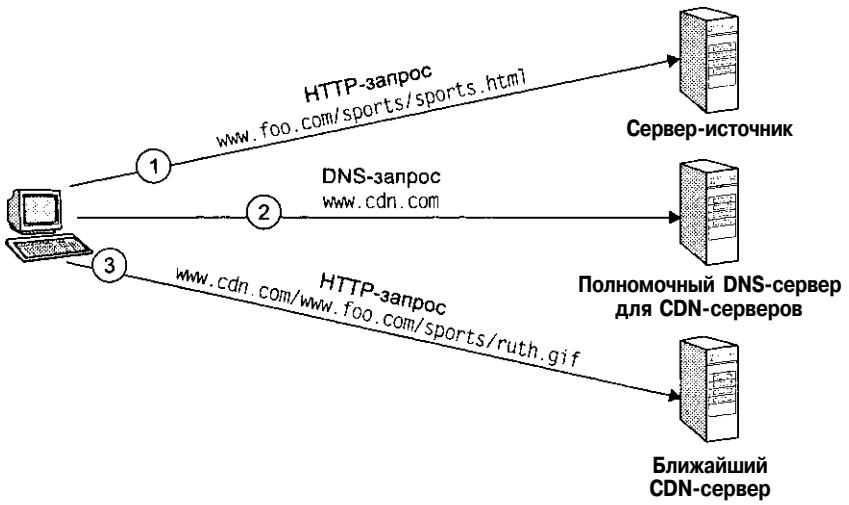


Рис. 2.29. CDN использует DNS для перенаправления запросов ближайшим CDN-серверам

Хотя наше рассмотрение CDN лежит в контексте web-ресурсов, сети CDN весьма активно используются, например, для передачи потоковых аудио- и видеоресурсов. Для этого применяются специальные протоколы (RTSP и др.), поддерживаемые большинством CDN-серверов. Мы рассмотрим соответствующие вопросы в главе 6.

Одноранговое разделение файлов

Из предыдущих разделов мы знаем, что пользователь может получать объекты с web-серверов-источников, принадлежащих поставщикам ресурсов, с прокси-серверов, арендуемых Интернет-провайдерами, а также с CDN-серверов, управляемых CDN-компаниями. Тем не менее технологии распределения ресурсов этим не исчерпываются. Оказывается, обычные оконечные системы могут обмениваться объектами непосредственно друг с другом! Такому обмену, называемому одноранговым (Peer-to-Peer, P2P) разделением файлов, посвящено немало информационных ресурсов [141, 167, 184, 343]. Мы рассмотрим вопросы, связанные с соедине-

нием и передачей информации в контексте одноранговой разделения файлов. Разумеется, технология P2P имеет множество других важных аспектов применения, относящихся к безопасности, конфиденциальности, анонимности и защите авторских прав; мы не будем рассматривать их в нашей книге, однако заинтересованный читатель может ознакомиться с этими (и не только) аспектами применения технологии P2P в [167, 373].

Перед тем как «проникнуть» внутрь одноранговой системы, посмотрим, каким образом пара пользователей может ее применить. Для наглядности вспомним наших персонажей. Пусть Алиса использует P2P-систему для загрузки MP3-файлов с помощью программного обеспечения, выполняющегося на ее портативном персональном компьютере, который имеет резидентный доступ в Интернет через модем. Алиса пользуется связью с Интернетом в течение нескольких часов в день; в остальное время телефонная линия свободна для переговоров. Компьютер Алисы не имеет имени хоста и каждый раз при подключении к сети ему назначается новый IP-адрес.

Предположим, Алиса подключена к Интернету, ее одноранговое приложение запущено, и с его помощью она осуществляет поиск MP3-файла с композицией «Hey Jude» группы «The Beatles». Через некоторое время после ввода команды поиска приложение выводит на экран список систем, подключенных к Интернету и содержащих искомую композицию. Как правило, одноранговая система представляет собой обычный персональный компьютер. Обычно P2P-приложения вместе со списком систем выводят дополнительную информацию о каждой системе, например скорость обмена информацией и приблизительное время передачи файла. Алиса выбирает одну систему из списка, к примеру компьютер Боба, и отправляет ему запрос на получение файла. Между компьютерами Алисы и Боба устанавливается прямое TCP-соединение и осуществляется передача требуемого файла. В случае разрыва соединения компьютера Боба с Интернетом P2P-приложение Алисы может предпринять автоматическую попытку загрузки оставшегося фрагмента файла с другой одноранговой системы. Во время загрузки файла с композицией «Hey Jude» компьютер Алисы может обслуживать запрос другой системы, передавая ему, к примеру, файл с песней «Angie» группы «Rolling Stones». Таким образом, каждая одноранговая система одновременно является поставщиком и потребителем ресурсов.

Очевидно важное достоинство P2P-систем: все объекты передаются между хостами без участия «третьей» стороны в лице серверов-источников, кэш-серверов и CDN-серверов. Таким образом, миллионы пользователей могут напрямую использовать ресурсы друг друга (пропускную способность, дисковое пространство и вычислительные мощности) для распределения ресурсов.

Несмотря на то что файловый обмен в P2P-системе происходит децентрализованно, следует помнить о том, что фактически при этом используется уже знакомая нам парадигма клиент/сервер. Действительно, систему, формирующую запрос на получение файла, следует отнести к клиентской стороне, а систему, передающую файл, — к серверной. Передача файла осуществляется при помощи протокола, предназначенного для файлового обмена. Для того чтобы одноранговая система была

способна осуществлять генерацию и выполнение запросов, необходимо, чтобы она поддерживала как клиентскую, так и серверную части протокола. Рассмотрим, что происходит в процессе обмена при использовании протокола HTTP, популярного в P2P-приложениях. Вернемся к описанному выше примеру, где Алиса загружает файл с композицией «Hey Jude» с компьютера Боба. Одноранговое приложение на стороне Алисы формирует HTTP-запрос на получение соответствующего файла в адрес хоста Боба, а приложение Боба обрабатывает запрос и формирует ответ, содержащий требуемый файл. Необходимо отметить, что компьютер Алисы в P2P-системе является одновременно и клиентом, и *временным сервером*. Он играет роль клиента при передаче запросов и приеме ответов и роль сервера при обработке и обслуживании запросов. Термин «временный сервер» отражает тот факт, что соединение компьютера Алисы с Интернетом не является постоянным, IP-адрес компьютера изменяется с каждым новым подключением к сети, и прием запросов осуществляется только при наличии соединения с Интернетом.

Мы описали наиболее простую область применения P2P-системы — передачу файлов между одноранговыми системами. А вопрос о том, каким образом приложение осуществляет поиск нужного файла, остался открытым. Обычно в каждый момент времени существует множество подключенных к P2P-системе пользователей, каждый из которых обладает общедоступными ресурсами, включая MP3-файлы, изображения, видеоклипы и программы. Для того чтобы составить список пользователей, располагающих требуемым объектом, приложению необходимо получить IP-адреса всех систем, подключенных к сети. Поскольку общее число систем весьма велико, новые отключения и подключения к сети происходят очень часто, что делает разрешение задачи непростым. Ниже мы опишем три варианта архитектуры, позволяющих осуществлять поиск ресурсов и применяющихся в различных P2P-системах. Заинтересованный читатель также может обратиться к дополнительным источникам информации [410, 434, 489].

Централизованный каталог

Одним из наиболее прямолинейных решений проблемы поиска ресурсов является создание централизованного каталога. Подобным образом поступила компания Napster — первая коммерческая компания, реализовавшая широкомасштабную P2P-систему обмена MP3-файлами. При таком способе решения специально для осуществления поиска создается сервер или объединение серверов. Как показано на рис. 2.30, при запуске однорангового приложения оно связывается с централизованным каталогом и сообщает свой IP-адрес, а также список файлов, выделяемых в совместное использование. Таким образом, централизованный каталог представляет собой динамическую базу данных, с помощью которой все системы могут получать наиболее важную информацию друг о друге: IP-адреса и списки доступных файлов. Если активный пользователь добавляет или удаляет какой-либо объект, соответствующее изменение сразу же вносится в базу данных.

Для того чтобы данные базы не устаревали, необходимо следить за моментами отключения пользователей. Отключение может произойти вследствие завершения работы однорангового приложения либо при разрыве соединения пользователя с Интернетом. Один из способов слежения за активностью пользователей состоит

в периодической генерации сообщений центральным сервером в адрес каждой из систем и проверке наличия ответов от них. Другой способ заключается в установлении постоянного TCP-соединения между каждой системой и сервером; в этом случае разрыв TCP-соединения будет сигнализировать об отключении системы. При любом способе слежения отключение пользователя влечет за собой удаление его IP-адреса из базы данных центрального сервера.

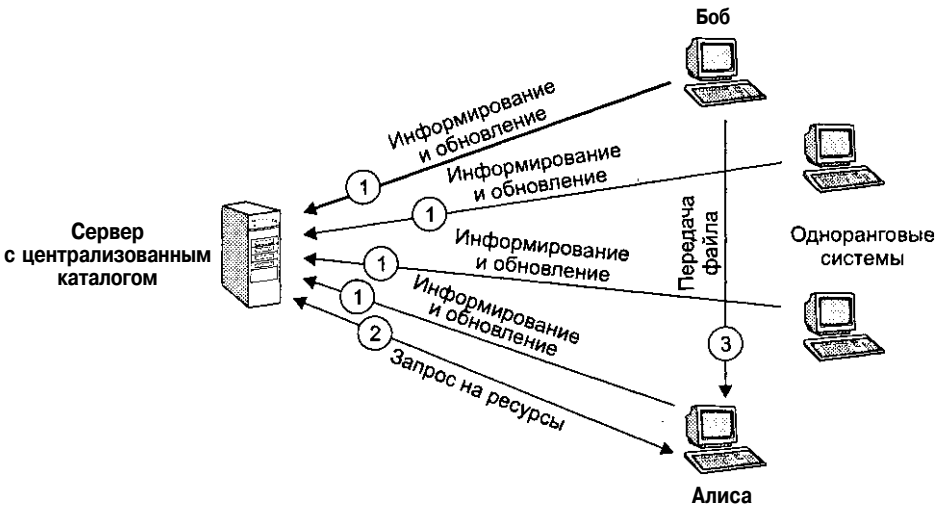


Рис. 2.30. Одноранговая система с централизованным каталогом

Использование централизованного каталога является простым с концептуальной точки зрения, однако имеет три очевидных недостатка.

- *Одна точка возможного отказа.* Центральный сервер является «узким местом» с точки зрения надежности, поскольку сбой в его работе влечет за собой остановку всей системы. Даже при наличии резервных серверов, как правило, не удается сохранить постоянную работоспособность системы.
- *Возможность перегрузки.* В больших одноранговых системах центральный сервер является «узким местом» с точки зрения нагрузки, поскольку вынужден хранить огромный объем информации о текущих активных пользователях и обрабатывать тысячи запросов в секунду. В 2000 году компания Napster испытала немало неприятностей, связанных с недостаточной мощи остью своего центрального сервера, когда популярность их одноранговой системы резко возросла вместе с числом пользователей и объемом трафика.
- *Защита авторских прав.* Хотя вопросы защиты авторских прав не связаны с темой нашей книги, отметим, что звукозаписывающая индустрия, мягко говоря, обеспокоена тем фактом, что музыкальная продукция, защищенная авторскими правами, в цифровом виде может бесплатно и практически беспрепятственно распространяться среди пользователей Интернета. Если компания располагает централизованным сервером, то суд может обязать компанию закрыть этот

сервер. Подобное решение гораздо труднее осуществить в отношении систем с более распределенной архитектурой.

Подводя итог, можно сказать, что указанные недостатки проистекают из того факта, что одноранговые системы децентрализованы лишь отчасти. Хотя обмен файлами между одноранговыми системами является децентрализованным, местоположение ресурсов жестко централизовано, что и является источником проблем.

Децентрализованный каталог

Для того чтобы избежать проблем, порождаемых централизацией, естественно попытаться распределить ресурсы центрального каталога между одноранговыми системами. Подобный подход был использован в системе KaZaA/FastTrack [141], завоевавшей широкую популярность в 2001-2002 годах.

Как показано на рис. 2.31, часть одноранговых систем выделяется, и они становятся *лидерами групп*. Все остальные системы входят в состав одной из групп и связаны с ее лидером. Получив IP-адрес лидера своей группы, каждый пользователь отсылает ему список доступных файлов; таким образом, лидер группы хранит базу данных, связывающую IP-адреса членов группы с именами файлов. В такой схеме исходная система фактически разбивается на множество связанных между собой одноранговых подсистем, в которых лидер группы играет роль центрального сервера. Необходимо отметить, что лидер группы отличается от ее «рядовых» членов лишь тем, что располагает базой данных о ресурсах группы; было бы ошибочно думать, что он представляет собой выделенный сервер.

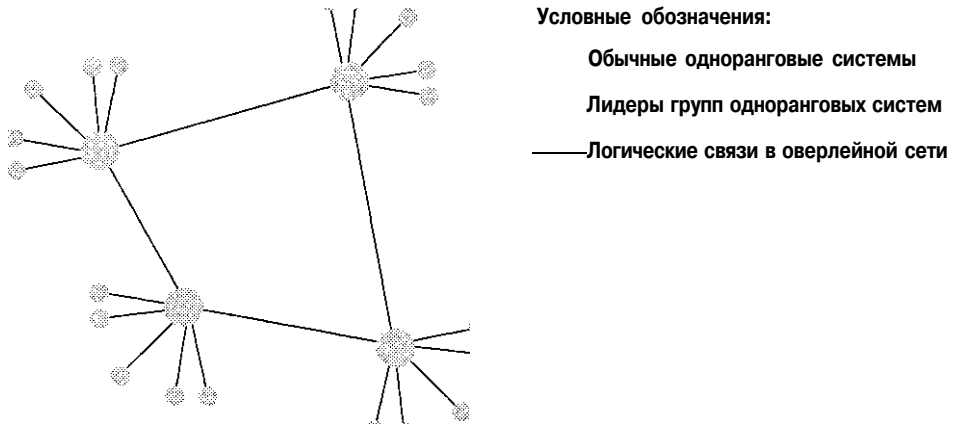


Рис. 2.31. Иерархическая оверлейная одноранговая сеть с децентрализованным каталогом

Когда некоторый пользователь хочет получить файл, он посылает запрос лидеру своей группы. Лидер группы осуществляет поиск сначала в собственной базе данных, а затем в базах данных лидеров других групп и отсылает результаты запрашивающей системе. Таким образом, ответ содержит IP-адреса множества пользователей, находящихся в разных группах системы.

Одноранговые системы и их взаимосвязи образуют абстрактную логическую структуру, называемую *оверлейной сетью,* представленную на рис. 2.31 в виде графа, где системы являются вершинами графа, а связи между системами — ребрами. Все вершины, соответствующие членам каждой группы, соединены ребрами со своим лидером; лидеры, напрямую направляющие друг другу запросы, также связаны между собой ребрами. Отметим, что ребрам не обязательно соответствуют физические линии связи; напротив, связи в оверлейной сети носят логический характер и отражают направления передачи запросов. Даже в случае, если лидер группы и какой-либо ее член подключены к сетям разных Интернет-провайдеров, в оверлейной сети они всегда соединены ребром и являются *соседями.*

Не удивительно, если наши последние рассуждения вызвали у вас целый ряд вопросов о том, как устроена оверлейная сеть и каким образом она управляется. Например, неясны механизмы назначения лидеров групп и закрепления за ними новых пользователей. Поскольку процесс подключения и отключения пользователей непрерывен и интенсивен, оверлейная сеть постоянно меняется. Для принятия в сеть нового пользователя необходимо установить соединение последнего хотя бы с одним уже подключенным к сети пользователем. Очевидно, что для подключения потенциальный пользователь должен получить IP-адрес своего будущего лидера группы. Это делается при помощи одного или нескольких специальных *узлов начальной загрузки.* Будущий одноранговый пользователь сначала устанавливает соединение с узлом начальной загрузки, который сообщает ему IP-адрес лидера группы, а затем с помощью полученного IP-адреса устанавливает соединение с лидером группы. Иногда узел начальной загрузки может сразу присвоить новой системе статус лидера группы. В этом случае система получает IP-адреса других (возможно, всех) лидеров групп системы. Поскольку узлы начальной загрузки являются непрерывно функционирующими серверами, их IP-адреса доступны по протоколу DNS.

Главным достоинством системы с децентрализованным каталогом является отсутствие в ней выделенного сервера для базы данных каталога: база данных распределена между лидерами групп. Поскольку каждый лидер хранит информацию только о членах собственной группы, размер его базы данных относительно невелик. Кроме того, подобную систему в силу низкой степени ее централизации трудно закрыть путем судебного решения (например, за нарушение авторских прав).

Разумеется, наряду с достоинствами предложенная модель имеет и несколько недостатков. Во-первых, для функционирования оверлейной сети необходим сложный протокол взаимодействия ее пользователей. Представим себе ситуацию, когда лидер группы отключается от сети; в этом случае требуется оперативно установить факт отключения и присоединить всех членов группы к другим лидерам. Решение только этой задачи уже представляет немалую трудность. Во-вторых, несмотря на относительно низкую степень централизации, лидеры групп не являются истинно одноранговыми (то есть равноправными) по отношению к остальным членам групп. Полностью децентрализованная система предполагает равенство всех ее составных частей. Лидеры групп вынуждены выполнять относительно более широкий круг обязанностей, тем самым становясь потенциальным «узким местом» с точки зрения загрузки. Кроме того, в системе присутствует непрерывно функци-

онирующий сервер, выполняющий функции узла начальной загрузки. Несмотря на то, что функции узла начальной загрузки не представляют сложности (присоединение новых систем к лидерам групп, назначение лидеров групп и поддержка оверлейной сети), в каждый момент времени необходимо присутствие хотя бы одного такого узла.

Потокзапросов

В бесплатном приложении для совместного использования файлов Gnutella [184] задействована полностью распределенная модель распределения ресурсов. В этой модели одноранговые системы самоорганизуются в оверлейную сеть. В отличие от оверлейной сети, описанной выше, сеть приложения Gnutella имеет «плоскую», неструктурированную топологию. Все пользователи равны между собой, поскольку иерархическая структура в сети отсутствует. Рисунок 2.32 иллюстрирует графическое представление описываемой топологии. Как и в предыдущей модели, присоединение новой системы происходит с помощью узла начальной загрузки, выдающего IP-адреса одного или нескольких существующих пользователей сети. Каждая система располагает сведениями только о своих соседях, то есть системах, с которыми у нее имеется логическая связь. Для поддержки подобной оверлейной сети требуется сложный протокол, учитывающий постоянные динамические изменения структуры сети, связанные с подключениями и отключениями пользователей.

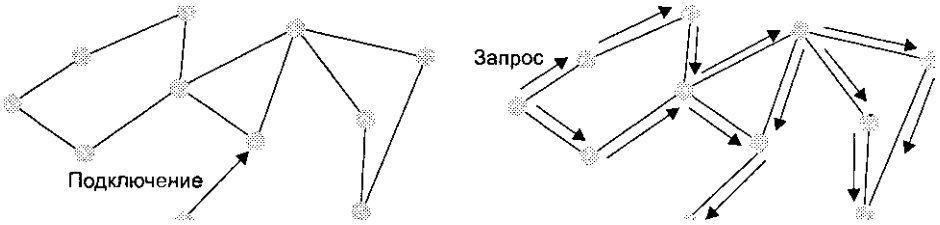


Рис. 2.32. Включение нового пользователя в оверлейную сеть (а) и поток запросов в оверлейной сети (б)

Для поиска объектов в приложении Gnutella не используются каталоги (ни централизованные, ни децентрализованные); вместо них применяется поток запросов, суть которого заключается в следующем. Когда некоторый пользователь, например Алиса, желает получить объект, приложение направляет запросы на получение этого объекта соседним системам. Последние, в свою очередь, направляют запросы своим соседям (кроме Алисы), и эта процедура продолжается до тех пор, пока запросы не будут переданы всем системам сети. При наличии объекта система отправляет соответствующий ответ системе, отправившей запрос.

Информационная модель, используемая приложением Gnutella, обладает рядом несомненных достоинств. Прежде всего, следует отметить реальную одноранго-

вость (равноправие) всех систем, приводящую к равномерному распределению обязанностей между ними. Кроме того, ни одна из систем не хранит информацию, связывающую ресурсы с IP-адресами. Отсутствие базы данных (централизованной или распределенной) значительно упрощает модель системы.

Вместе с тем приложение Gnutella часто критикуется за невозможность масштабирования. При использовании модели в том виде, в котором мы ее описали, любой запрос распространяется по всей сети. Это приводит к значительному трафику запросов, которые должны обрабатываться каждой системой. Разработчики Gnutella предусмотрели решение этой проблемы, ограничив область распространения запросов определенным числом систем. Запрос включает в себя специальное поле, содержащее счетчик узлов, через которые прошел запрос. При создании запроса значение счетчика устанавливается равным области распространения и при прохождении каждой системой декрементируется. Если система получает запрос с нулевым значением счетчика, дальнейшая пересылка запроса прекращается. Такой подход позволяет сократить объем трафика запросов, однако повышает вероятность того, что список пользователей, располагающих требуемым объектом, окажется неполным.

Как и в схеме с децентрализованным каталогом, система Gnutella вынуждена поддерживать оверлейную сеть. Несмотря на то что данная задача является весьма непростой, на практике ее решение вполне достижимо. Для присоединения к сети новых систем используется один или несколько узлов начальной загрузки. На сегодняшний день задача построения одноранговой сети без узлов начальной загрузки не решена.

Как и упомянутые выше системы Napster и KaZaA/FastTrack, Gnutella пережила стремительный взлет популярности, приобретя миллионы новых пользователей за несколько месяцев функционирования. Бурное развитие одноранговых приложений, включая системы обмена сообщениями в реальном времени и совместного использования файлов, в очередной раз показало исключительную универсальность структуры Интернета, позволившую успешно функционировать на протяжении 25 лет после создания, поддерживая множество самых разных приложений. Разумеется, функционирование этих приложений невозможно без специальных средств обслуживания, куда входят системы передачи дейтаграмм с установлением и без установления соединения, системы адресации и именования (DNS), интерфейс сокетов и др. Поскольку приложения находятся на самом верхнем уровне стека протоколов Интернета, их разработка означает разработку нового программного обеспечения, предназначенного для взаимодействия оконечных систем с использованием парадигмы клиент/сервер.

Отметим, что далеко не все разработки в сфере Интернет-приложений являются столь же успешными, как системы обмена сообщениями в реальном времени и совместного использования файлов. Так, например, приложения, предназначенные для работы с потоковым мультимедиа и организации видеоконференций, до сих пор далеки от популярности. Вероятно, это можно объяснить недостаточной сетевой поддержкой (мы обсудим существующие средства поддержки в главе 6) либо негативным влиянием социально-экономических факторов.

Резюме

В этой главе мы рассмотрели концептуальные и практические аспекты, касающиеся сетевых приложений. Мы ознакомились с важнейшей парадигмой клиент/сервер и ее реализацией в различных Интернет-протоколах: HTTP, FTP, SMTP, POP3 и DNS. Кроме протоколов нами были рассмотрены принципы функционирования ряда ключевых приложений (web-браузеров, электронной почты, приложений обмена файлами, системы доменных имен). Мы ознакомились с понятием сокета и увидели, каким образом интерфейс сокета используется для построения приложений. Мы рассмотрели применение сокетов для обоих протоколов транспортного уровня (TCP и UDP), а затем разработали несложный web-сервер с поддержкой сокетов. Последний вопрос, который был нами рассмотрен, касался различных технологий распределения ресурсов: кэширования, сетей распределения ресурсов (CDN) и одноранговых приложений. Таким образом, наш первый шаг от «границы» сети к ее «ядру» можно считать сделанным.

В самом начале книги мы дали весьма расплывчатое и непонятное определение протокола; мы сказали, что протоколом называется совокупность соглашений, описывающих формат сообщений, предназначенных для передачи между двумя или более устройствами, а также процедур, выполняемых при передаче и/или приеме сообщений либо наступлении иных событий. Материал, изложенный в этой главе, во многом прояснил смысл данного нами определения. Протокол является ключевой концепцией в сфере компьютерных сетей; мы надеемся, что рассмотренные протоколы, такие как HTTP, FTP, SMTP, POP3 и DNS, сформировали у вас ясное представление о том, что представляет собой это понятие.

В разделе «Принципы работы протоколов прикладного уровня» мы описали модели обслуживания протоколов транспортного уровня TCP и UDP и ознакомились с практическим применением этих моделей в разделах «Программирование TCP-сокетов», «Программирование UDP-сокетов» и «Разработка простого web-сервера». Как вы, вероятно, заметили, при этом мы практически не коснулись принципов функционирования TCP и UDP. В следующей главе мы детально рассмотрим эти принципы, сделав еще один шаг к «ядру» компьютерных сетей.

Ознакомившись со структурой сетевых приложений и их протоколами, мы готовы перейти к изучению следующего уровня коммуникационной модели Интернета — транспортного уровня.

Вопросы и задания для самостоятельной работы

Приведенные ниже задания рекомендуются для самостоятельной проработки. При возникновении трудностей обратитесь к соответствующим разделам этой главы.

1. Перечислите пять наиболее популярных Интернет-приложений и укажите протоколы прикладного уровня, которые они используют.
2. Каким образом при взаимодействии двух хостов вы определяете, какая из сторон клиентская, а какая серверная?
3. Какая информация используется процессом для идентификации удаленного процесса?

4. Перечислите несколько агентов пользователя, используемых вами в повседневной работе за компьютером.
5. Обратившись к табл. 2.1 (см. раздел «Принципы работы протоколов прикладного уровня»), можно увидеть, что ни одно из перечисленных в ней приложений не требует, чтобы отсутствовали потери данных и одновременно выполнялись ограничения на время доставки. Можете ли вы представить себе приложение, которое выдвигало бы подобные требования?
6. Что означает термин «протокол рукопожатия»?
7. Почему протоколы прикладного уровня HTTP, FTP, SMTP, POP3 и IMAP строятся на основе протокола транспортного уровня TCP, а не UDP?
8. Предположим, что для каждого пользователя на web-сайте, занимающемся электронной коммерцией, должна существовать запись, в которой будут регистрироваться его покупки. Опишите, каким образом эта задача решается с помощью механизма аутентификации HTTP, а также с помощью объектов cookie.
9. В чем заключается разница между постоянными соединениями с конвейеризацией и постоянными соединениями без конвейеризации? Какое из соединений используется в протоколе HTTP/1.1?
10. Установите соединение с удаленным web-сервером при помощи программы Telnet и сформируйте запрос, состоящий из нескольких строк. Включите в заголовок запроса строку If-modified-since:, чтобы вызвать генерацию ответа с кодом состояния 304 Not Modified.
11. Почему говорят, что протокол FTP осуществляет передачу информации вне полосы (out-of-band)?
12. Предположим, что Алиса, получающая доступ к электронной почте через web-интерфейс, отправляет письмо Бобу, использующему протокол POP3. Объясните, каким образом письмо Алисы попадает на хост Боба. Перечислите протоколы прикладного уровня, применяющиеся в процессе доставки.
13. Предположим, что вы создаете электронное письмо, не содержащее иной информации, кроме единственного вложения в формате Microsoft Excel. Каково будет содержание строк заголовка сообщения (включая строки MIME-расширения)?
14. Просмотрите заголовок одного из полученных вами электронных сообщений. Сколько строк Received: в нем содержится? Проанализируйте содержимое каждой строки заголовка.
15. В чем различие с точки зрения пользователя между загрузкой сообщений с их удалением и без их удаления в протоколе POP3?
16. Измените рис. 2.16 для случая, когда все запросы локального сервера имен являются итеративными.
17. Каждый Интернет-хост имеет как минимум один локальный сервер имен и один полномочный сервер имен. Какую роль в системе DNS играет каждый из этих серверов?

18. Возможно ли наличие одного и того же псевдонима (например, foo.com) для имен web-сервера и почтового сервера организации? Какой тип будет иметь запись ресурсов, содержащая имя почтового сервера?
19. С использованием команды nslookup найдите web-сервер, имеющий несколько IP-адресов. Определите, один или несколько адресов имеет web-сервер вашей компании/университета.
20. UDP-сервер, описанный в разделе «Программирование UDP-сокетов», требует единственного сокета, в то время как TCP-сервером, описанным в разделе «Программирование TCP-сокетов», используются два сокета. Объясните причину разного числа сокетов. Каковым было бы число TCP-сокетов сервера, если бы он поддерживал *n* одновременных соединений с разными клиентами?
21. Почему для TCP-приложения с архитектурой клиент/сервер, описанного в разделе «Программирование TCP-сокетов», запуск программы-сервера требуется производить раньше запуска программы-клиента? Почему подобное ограничение отсутствует в UDP-приложении?
22. Объясните, почему web-кэширование способно сократить время доставки требуемого объекта. Будет ли сокращение времени доставки иметь место для всех объектов, запрашиваемых пользователями? Объясните ответ.
23. В подразделе «Метод GET с условием» раздела «Web и HTTP» был описан механизм проверки на необходимость обновления объектов, содержащихся в локальном кэше пользователя. Может ли этот механизм быть применен для сетевых кэшей? Поясните ответ.
24. Какова роль DNS в сетях распределения ресурсов?
25. Что представляет собой оверлейная сеть в одноранговых системах? Включает ли она в себя маршрутизаторы? Что означают ребра оверлейной сети? Каков механизм создания и поддержки оверлейной сети в системе Gnutella?
26. Найдите три компании, предоставляющие услуги одноранговой службы. Какой тип ресурсов распределяется с помощью этих компаний? Каковы механизмы поиска ресурсов в системах каждой из компаний?

Упражнения

1. Верно или неверно следующее утверждение.
   1.1. Предположим, что пользователь осуществляет загрузку web-страницы, содержащей текст и два изображения. В процессе загрузки клиент сформирует один запрос и получит три ответных сообщения.
   1.2. Две отдельные web-страницы (например, www.rrnt.edu/research.html и www.mit.edu/students.html) могут быть доставлены через одно постоянное соединение.
   1.3. При непостоянном соединении между браузером и сервером-источником можно передать два HTTP-запроса в одном TCP-сегменте.
   1.4. Строка Date: заголовка ответного HTTP-сообщения содержит информацию о дате последнего изменения объекта.

2. Ознакомьтесь с содержанием документа RFC 959, описывающего протокол FTP. Перечислите все команды клиента, указанные в документе.
3. Ознакомьтесь с содержанием документа RFC 1700. Каковы номера портов, используемые протоколами SFTP (Simple File Transfer Protocol — простой протокол передачи файлов) и NNTP (Network News Transfer Protocol — протокол передачи сетевых новостей)?
4. Предположим, что некоторый HTTP-клиент желает загрузить web-документ, зная его URL-адрес и не зная IP-адреса. Web-документ содержит один встроенный рисунок в формате GIF, находящийся на том же сервере, что и сам документ. Какие протоколы прикладного и транспортного уровней потребуются в процессе доставки?
5. Ознакомьтесь со спецификацией HTTP/1.1 (документ RFC 2616) и ответьте на следующие вопросы.
   5.1. Каков механизм уведомления клиентом и сервером друг друга о закрытии постоянного соединения? Могут ли сигналы о закрытии соединения формироваться клиентом? Формироваться сервером? Формироваться обеими сторонами?
   5.2. Какие функции шифрования данных выполняет протокол HTTP?
6. Представьте, что в своем web-браузере вы щелкаете на ссылке, чтобы открыть web-страницу. Предположим, что IP-адрес страницы отсутствует в локальном кэше вашего компьютера и, таким образом, для доставки страницы сначала необходимо сделать DNS-запрос. Пусть в процессе поиска нужного сервера имен было сформировано *n* запросов к различным DNS-серверам, время ответа для которых составило RTT^{t}, $RTT_{„}$ соответственно. Предположим, что запрашиваемая web-страница состоит из единственного объекта, содержащего небольшой объем текстовой информации. Пусть $RTT^{()}$ — время оборота между вашим хостом и сервером, содержащим web-страницу. Считая время передачи объекта пренебрежимо малым, оцените время, начиная от щелчка на ссылке и заканчивая получением запрошенного объекта.
7. Вернемся к предыдущей проблеме и предположим, что HTML-файл содержит ссылки на три объекта, находящихся на том же сервере и имеющих достаточно малый размер, чтобы временем их пересылки можно было пренебречь. Оцените время выполнения запроса для случаев непостоянных TCP-соединений без параллелизма, непостоянных соединений с параллелизмом и постоянных соединений с конвейеризацией.
8. В протоколе HTTP предусмотрены методы GET и POST. Существуют ли другие методы выполнения запросов в версии HTTP/1.0? Если да, то для чего они используются? Дайте ответы на эти же вопросы для версии HTTP/1.1.
9. Напишите простую программу с использованием протокола TCP, принимающую строки символов от клиента и печатающую эти строки на стандартном устройстве вывода сервера (можете воспользоваться готовыми фрагментами программы TCPServer.java). Скомпилируйте вашу программу и запустите ее. На любом другом компьютере, имеющем браузер, укажите в качестве прокси-сер-

вера компьютер, где выполняется ваша программа; при этом проследите за правильностью задания номера порта. После этого запросы браузера будут отображаться на экране вашего компьютера. Проследите, использует ли браузер метод GET с условием при генерации запросов на получение объектов, находящихся в локальном кэше.

10. Ознакомьтесь с документом RFC 1939, описывающим протокол POP3. Для чего предназначена команда UIDL?

11. Установите и скомпилируйте пару программ TCPClient и UDPClient на одном хосте, и пару программ TCPServer и UDPServer на другом хосте. Ответьте на следующие вопросы.

    11.1. Что произойдет, если программа TCPClient будет запущена раньше, чем программа TCPServer? Почему?

    11.2. Что произойдет, если программа UDPClient будет запущена раньше, чем программа UDPServer? Почему?

    11.3. Что произойдет, если номера портов клиента и сервера не будут согласованы?

12. Модифицируйте программу TCPServer.java так, чтобы она могла принимать множество запросов; для этого вам придется использовать потоки выполнения (threads).

13. Обратимся к рис. 2.26, на котором изображена университетская локальная сеть, подключенная к Интернету, и примеру, иллюстрацией к которому служит рисунок. Предположим, что средний размер запрашиваемых объектов составляет 900 000 бит, частота запросов равна 1,5 запроса в секунду, а время с момента передачи запроса первым маршрутизатором Интернета до получения им ответа («задержка Интернета») составляет в среднем 2 с. Время ответа будем считать равным сумме среднего времени доступа (задержки между маршрутизаторами локальной сети и Интернета) и задержки Интернета. Среднее время доступа равно A/(1 - A(3), где A — среднее время передачи объекта по линии доступа, а P — частота поступления объектов в линию.

    13.1. Найдите среднее время ответа.

    13.2. Пусть в локальной сети установлен кэш, вероятность удовлетворения запроса которым составляет 0,4. Найдите среднее время ответа.

14. Здесь мы коснемся вопроса хэширования URL-адресов при совместном кэшировании (см. подраздел «Web-кэширование» в разделе «Распределение ресурсов»). Предположим, что организация имеет три кэш-сервера, cache-0, cache-1 и cache-2. Пусть система хэширования функционирует в соответствии с пятью условиями: используется только фрагмент URL-адреса, содержащий имя хоста; все символы, не входящие в алфавит, игнорируются; каждому символу алфавита ставится в соответствие число, указывающее на его порядковый номер в алфавите; хэш-значение URL-адреса рассчитывается как сумма по модулю 3 всех чисел, соответствующих символам, входящим в URL; если сумма оказывается равной 0,1 и 2, то запрос направляется кэшу cache-0, cache-1 и cache-2 соответственно. Например, для URL-адреса http://aaa.bbb.com хэш-

значение равно (1 + 1 + 1 + 2 + 2 + 2 + 3 + 15 + 13) mod 3 = 1. Таким образом, запрос будет направлен кэшу cache-1.

14.1. Определите, какому кэшу будут направлены запросы к следующим URL-адресам: www.eurecom.fr; www.upenn.edu; www.wimba.com; www.w3c.org; www.ietf.org; www.awl.com.

14.2. Предположим, что в начале рабочего дня все три кэша пусты. За утренние часы пользователи произвели запросы к трем разным URL-адресам: www.upenn.edu/wharton.html, www.upenn.edu/engineering.html и www.upenn.edu/nursing.html. Днем поступили 15 запросов, по 5 запросов к каждому из указанных URL-адресов. Какие серверы (серверы-источники и кэш-серверы) выполняли 18 запросов в течение дня?

14.3. Продолжая предыдущий пример, предположим, что на следующий день в сеть был добавлен четвертый кэш-сервер, cache-3, и алгоритм хэширования был изменен так, что хэш-значение стало вычисляться по модулю 4 вместо модуля 3. Что произойдет при формировании запросов к трем указанным URL-адресам?

15. Обратимся к принципам функционирования CDN и DNS. Вспомним о том, что при создании DNS-запроса последний обычно отсылается локальному серверу имен, в кэше которого сохраняется результат выполнения запроса. Продолжая пример, приведенный в подразделе «Сети распределения ресурсов» раздела «Распределение ресурсов», предположим, что после того, как один пользователь создал запрос на получение спортивной страницы с сайта www.foo.com, другой пользователь запросил финансовую страницу с этого же сайта через тот же локальный сервер имен. Финансовая страница содержит два изображения в формате GIF. Будет ли использован сервер-источник www.foo.com для выполнения запроса второго пользователя? Какой из серверов имен, корневой или полномочный, выдаст IP-адрес для второго запроса?

16. Рассмотрим следующий альтернативный вариант организации сети CDN. Пусть поставщик ресурсов foo.com использует услуги CDN-компании cdn.com для распределения всех объектов, включая базовые HTML-страницы. С этой целью компанией cdn.com по заказу foo.com создается и поддерживается полномочный сервер имен для foo.com. Будут ли какие-либо запросы приниматься сервером-источником для foo.com при такой организации CDN? Обоснуйте ответ. Назовите достоинства и недостатки приведенного способа построения сети CDN.

17. Предположим, что вы загружаете MP3-файл при помощи однорангового приложения. «Узким местом» в Интернете для вас является ваша собственная линия доступа, представляющая собой дуплексную линию связи с пропускной способностью 128 Кбит/с. В некоторый момент времени 10 других пользователей одновременно начинают загрузку файлов с вашего компьютера. Предположим, что ваш компьютер обладает достаточной мощностью, и процессы взаимодействия с пользователями не вызывают перегрузок и сбоев. Будет ли обслуживание вашим компьютером 10 пользователей приводить к снижению скорости загрузки файла? Поясните ответ.

18. Рассмотрим поток запросов, упоминавшийся в разделе «Распределение ресурсов». Предположим, что каждый пользователь имеет не более *N* соседей, а область распространения запросов равна *K* узлов. Каково максимальное число запросов, которое может быть сгенерировано в сети в результате появления запроса от какого-либо пользователя?

19. В программе UDPClient.java имеется такая строка:

    ```
    DatagramSocket clientSocket = new DatagramSocketO;
    ```

    Предположим, что эта строка заменена следующей:

    ```
    DatagramSocket clientSocket = new DatagramSocket(5432);
    ```

    Возникнет ли при этом необходимость внесения изменений в программу UDPServer.java? Каков номер порта, использующийся в программах UDPClient и UDPServer до и после изменения?

Дополнительные вопросы и задания

1. Web-сайты (в особенности предназначенные для электронной коммерции) часто имеют встроенные базы данных. Каким образом HTTP-серверы получают доступ к этим базам данных?
2. Что представляет собой динамический язык HTML? Приведите примеры web-сайтов, использующих динамический язык HTML.
3. Что представляют собой серверные языки скриптов? Для чего они предназначены?
4. Каким образом можно настроить локальное кэширование вашего браузера? Перечислите соответствующие параметры.
5. Можете ли вы сконфигурировать свой браузер для поддержки нескольких одновременных соединений с web-сайтом? В чем заключаются достоинства и недостатки большого числа одновременных TCP-соединений?
6. В разделе «Электронная почта» было показано, что служба электронной почты зачастую реализуется с использованием одновременно HTTP-сервера и IMAP-сервера. Каким образом происходит взаимодействие серверов?
7. Как мы видели, сокеты протокола TCP воспринимают передаваемые данные как поток байтов, а сокеты UDP, напротив, способны различать границы сообщений. Назовите одно достоинство и один недостаток байт-ориентированного интерфейса по сравнению с интерфейсом, обеспечивающим передачу сообщений в виде, формируемом приложением.
8. Предположим, что вы располагаете астрологической базой данных и хотите сделать ее доступной для приложений, использующих протоколы HTTP и WAP, а также для устройств, преобразующих текст в речь. Каким образом можно достичь поставленной цели?
9. Что такое ICQ? Опишите, что предпринимается для стандартизации ICQ.
10. Каким образом функционируют списки собеседников систем обмена сообщениями в реальном времени?

11. Посетите web-сайт компании Akamai и опишите технические аспекты ее продукции.
12. Что представляет собой web-сервер Apache? Какова его стоимость? Какие основные функции он выполняет?
13. Представьте себе, что организации, занимающиеся стандартизацией в web, приняли решение изменить соглашение об адресации объектов, введя систему уникальных имен, не зависящих от расположения объектов (URN вместо URL). Выскажите свою гипотезу о возможных последствиях такого изменения.
14. Возможно ли построение однорангового приложения без узлов начальной загрузки? Если да, то каким образом?

Задания по программированию

Задание 1

Целью данного задания является разработка многопоточного web-сервера на языке Java, способного параллельно обслуживать несколько запросов. Сервер будет построен на основе протокола HTTP/1.0, описанного в документе RFC 1945.

Вспомним, что согласно спецификации HTTP/1.0 обслуживание каждого запроса осуществляется с помощью отдельного TCP-соединения. В свою очередь, каждое соединение управляется отдельным *потоком выполнения* (thread) программы. Кроме потоков, соответствующих каждому TCP-соединению, приложение предусматривает наличие главного потока, «прослушивающего» клиентов, желающих установить соединение. Для упрощения задачи мы разделим процесс написания кода на две части. Первая часть предусматривает создание многопоточного сервера, выводящего на экран содержимое принимаемых им запросов. Когда полученная программа будет функционировать корректно, добавим к ней код, генерирующий ответные сообщения.

Тестирование вашей программы можно проводить с помощью web-браузера. При этом следует помнить о том, что программа использует нестандартный порт, поэтому при создании запросов в URL-адресах необходимо явно указывать номер порта. К примеру, если ваш хост имеет имя host.someschool.edu, номер используемого порта 6789, а запрашиваемым объектом является файл index.html, то в адресную строку браузера следует ввести следующее значение:

http://host.someschool.edu:6789/i ndex.html

В случае обнаружения ошибки сервер должен генерировать ответное сообщение с указанием типа ошибки, чтобы соответствующая информация появилась в браузере. Детальное описание задания, а также полезные фрагменты кода вы можете найти на web-сайте http://www.awl.com/kurose-ross.

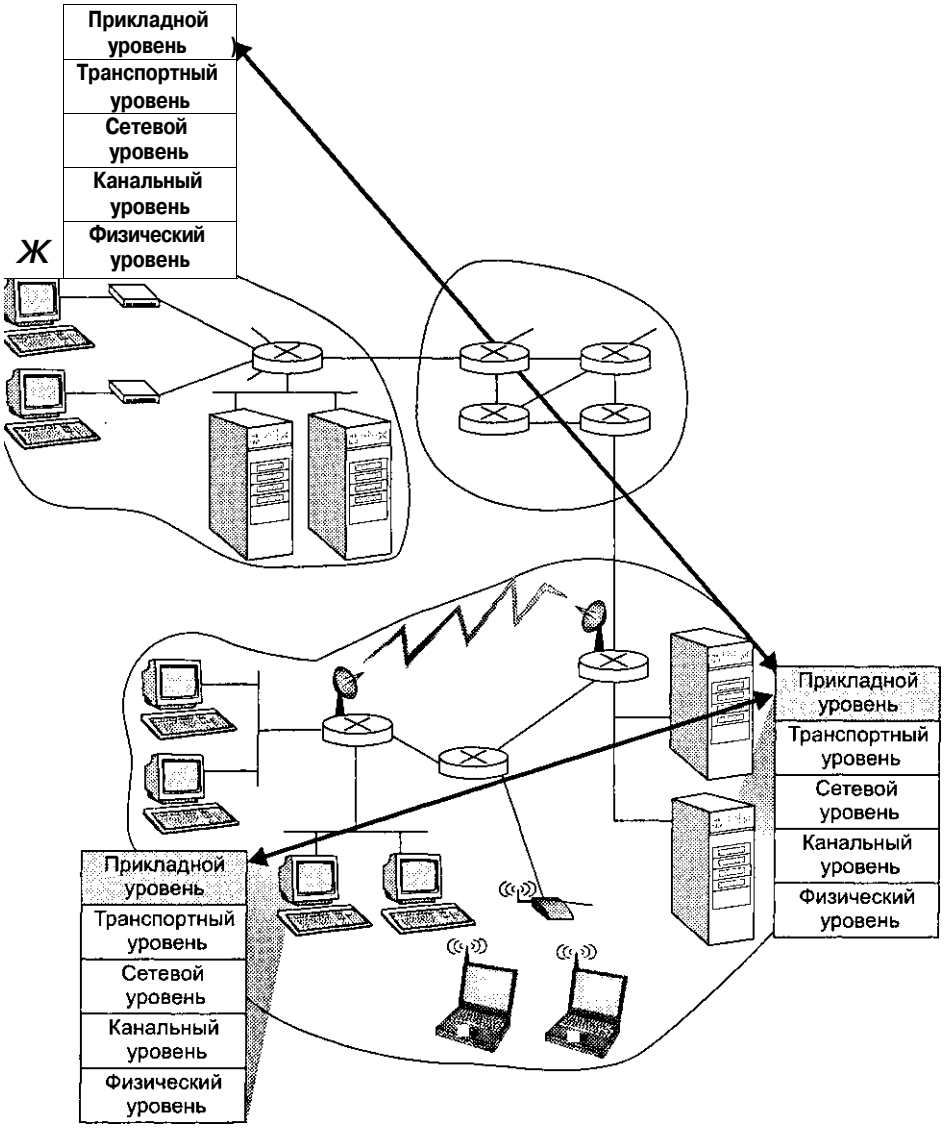
Задание 2

В этом задании требуется создать почтовый агент пользователя на языке Java, удовлетворяющий следующим требованиям.

- Графический интерфейс, позволяющий отображать поля адресов отправителя и получателя, тему письма, а также текст письма.

- Прямое TCP-соединение с почтовым сервером получателя. В этом случае сообщения не будут передаваться через почтовый сервер отправителя, как это происходит в подавляющем большинстве приложений.
- Передача и прием SMTP-команд и данных для доставки сообщения почтовому серверу получателя.

Возможный вид интерфейса приведен ниже.



Агент пользователя должен быть разработан таким образом, чтобы в течение одного соединения осуществлялась передача только одного сообщения. Кроме того,

предполагается, что доменная часть адреса получателя является именем SMTP-сервера, принимающего сообщения данного получателя (программа не осуществляет поиск DNS-записи типа MX, поэтому пользователю необходимо вводить фактическое имя почтового сервера).

Детальное описание задания, а также полезные фрагменты кода вы можете найти на web-сайте **http://www.awl.com/kurose-ross**.

Интервью

Тим Бернерс-Ли — директор web-консорциума (W3C) и главный научный сотрудник лаборатории вычислительной техники Института Массачусетса. В 1989 году, работая в лаборатории физики Европейского отдела CERN, он создал гипермедийное средство доступа к информации, получившее название World Wide Web. Через год после создания web Тим Бернерс-Ли разработал первые клиентскую и серверную программы для web. Тим Бернерс-Ли имеет диплом физика Оксфордского университета, который он окончил в 1976 году.

- Вашей первой специальностью является физика. Какое отношение физика имеет к компьютерным сетям?

  Особенность физики заключается в том, что, изучая ее законы в малом масштабе, вам необходимо представлять, каковы будут их глобальные проявления. Аналогично, создавая законы функционирования web, мне было необходимо задумываться над тем, чтобы в масштабе глобальной сети они были работоспособны. Мой подход основывался на анализе и синтезе, хотя эти два понятия весьма близки друг другу.

- Что повлияло на ваше решение изменить специализацию?

  В то время, когда я закончил обучение по специальности физика, наиболее интересной мне представлялась работа в компаниях, занимавшихся исследованиями в области телекоммуникаций. Появились первые микропроцессоры, которые стали стремительно вытеснять аппаратную логику. Это подавало большие надежды.

- Что, на ваш взгляд, является самым трудным в вашей работе?

  Пожалуй, урегулирование противоречий между группами разработчиков, работающих над одним проектом, но имеющих разногласия в инженерных вопросах. Эта проблема хорошо известна любому руководителю проекта. Поиск консенсуса всегда требует усилий.

- Каким вы представляете себе будущее Интернета?

  Прогноз, изложенный в моей книге, состоит из двух частей. Первая часть говорит о том, что web станет гораздо более мощным средством сотрудничества между людьми, чем сейчас. Информационное пространство представляется мне тем, к чему каждый человек может получить простой и быстрый доступ, и не только для того, чтобы использовать информацию, а для того, чтобы самому создавать ее. Взаимодействие между людьми при помо-

щи электронных средств и общего информационного хранилища должно стать настолько же простым и естественным, как личное общение. Вторая часть моего прогноза говорит о том, что расширение возможностей взаимодействия коснется и вычислительных машин. Компьютеры будут способны анализировать web-ресурсы, гиперссылки и транзакции пользователей. «Семантическая паутина», которая обеспечит возможность такого анализа, пока еще не создана, однако с ее появлением в торговле, в документообороте и во многих других сферах деятельности взаимодействующие компьютеры постепенно заменят людей, предоставив их своему вдохновению и интуиции. Для того чтобы это стало реальностью, необходим технический прогресс и наличие соответствующей законодательной базы. Я думаю, что Всемирная паутина будущего способна гармонично соединить прихоти людей с рациональностью компьютерной логики.

- Кого бы вы назвали своими профессиональными вдохновителями?

  Во-первых, своих родителей, которые занимались компьютерами со времени их возникновения и привили мне интерес к предмету. Своими учителями я считаю Майка Сенделл и Пегги Риммер, с которыми я работал в CERN. Кроме того, я испытываю большое уважение к ученым Ванневару Бушу, Дугу Энглберту и Теду Нельсону, которые в свое время делали аналогичные прогнозы, однако не имели возможностей осуществить их.

- Целью web-консорциума является «полное раскрытие потенциала web». В чем, по вашему мнению, заключается потенциал web? Каковы пути его раскрытия?

  Мы пытаемся направить развитие web в двух упомянутых мной направлениях. Наши усилия направлены на то, чтобы сделать Всемирную паутину максимально универсальной и доступной каждому человеку в любой точке пространства, с помощью любого браузера и устройства.

ГЛАВА 3 Транспортный уровень

Транспортный уровень, расположенный между прикладным и сетевым уровнями коммуникационной модели, играет ключевую роль в архитектуре Интернета. Особая важность транспортного уровня состоит в том, что он предоставляет услуги непосредственно прикладным процессам, выполняющимся на оконечных системах. Мы продолжим придерживаться установленного ранее подхода, заключающегося в поэтапном изучении базовых принципов организации транспортного уровня и их практического применения в протоколах. Как обычно, наибольшее внимание будет уделено Интернету, а следовательно, протоколам TCP и UDP.

В этой главе мы рассмотрим одну из наиболее фундаментальных проблем компьютерных сетей — надежную передачу по линии связи, допускающей искажения и потери данных. Мы опишем несколько постепенно усложняющихся (и от этого становящихся более реалистичными) ситуаций, отражающих данную проблему, и познакомимся с технологиями, направленными на ее решение. Затем мы узнаем о реализации этих технологий в протоколе TCP.

В конце главы мы рассмотрим другую фундаментальную проблему, касающуюся управления скоростью передачи данных оконечными системами во избежание перегрузок в сети. Мы раскроем причины перегрузок и их последствия, а также познакомимся с основными технологиями борьбы с перегрузками. Как и в предыдущем случае, мы закончим применением описанных технологий в протоколе TCP.

Службы транспортного уровня

В двух предыдущих главах мы коснулись роли транспортного уровня и услуг, предоставляемых его службами прикладному уровню коммуникационной модели. Подведем итог.

Протокол транспортного уровня обеспечивает *логическое соединение* между прикладными процессами, выполняющимися на разных хостах. Логическое соединение с точки зрения приложений выглядит как канал, непосредственно соединяющий процессы, хотя реальная связь между процессами может осуществляться с помощью

длинной цепи маршрутизаторов и разнообразных линий связи. С помощью логического соединения независимо от его физической инфраструктуры процессы могут осуществлять обмен данными. Рисунок 3.1 иллюстрирует эту ситуацию.

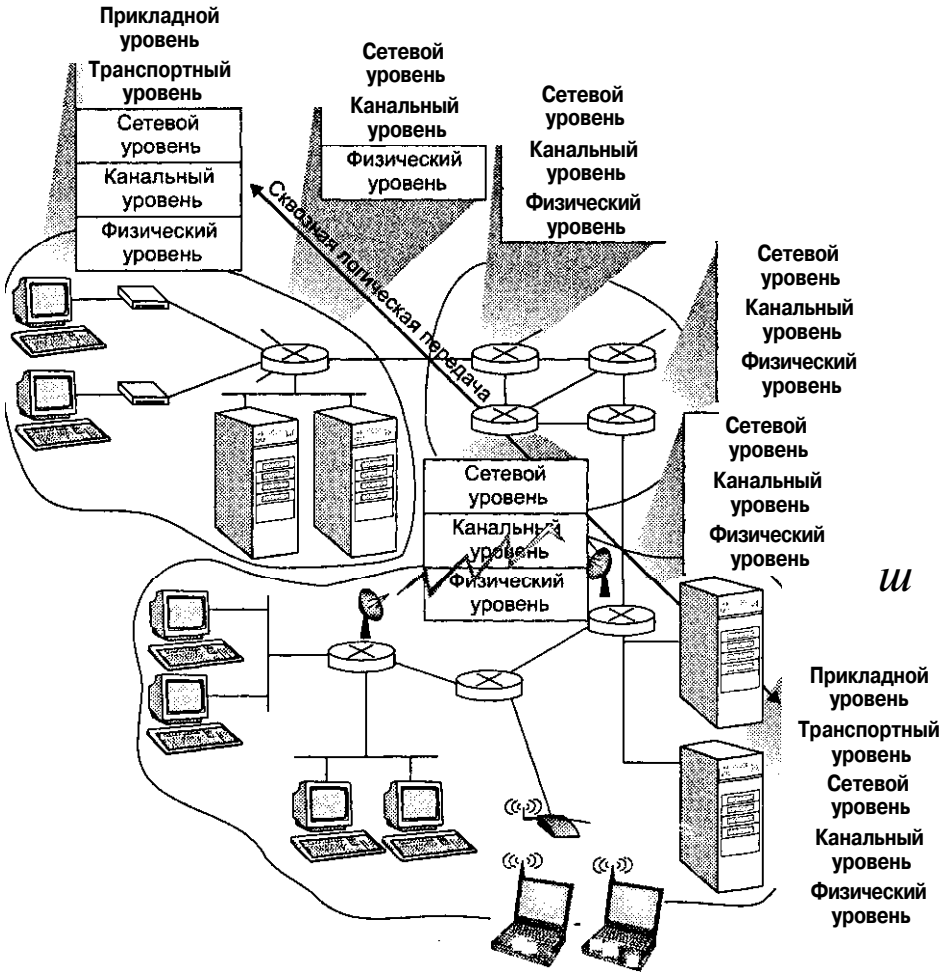


Рис. 3.1. Транспортный уровень обеспечивает логическое, а не физическое взаимодействие между прикладными процессами

Как следует из рисунка, протоколы транспортного уровня поддерживаются оконечными системами, однако не поддерживаются маршрутизаторами. Маршрутизаторы обрабатывают сообщения сетевого уровня и не оказывают влияния на сообщения транспортного уровня. На передающей стороне транспортный протокол преобразует сообщения приложения в сообщения транспортного уровня. Это преобразование иногда подразумевает разбиение сообщения, полученного от приложения, на несколько фрагментов с добавлением к каждому фрагменту заголовка

транспортного уровня. Затем транспортный уровень передает свои сообщения сетевому уровню. На приемной стороне транспортный уровень получает сообщения сетевого уровня, преобразует их в сообщения транспортного уровня, отбрасывает заголовки, производит необходимую обработку (например, сборку) и передает результат приложению.

Существует несколько протоколов транспортного уровня. Например, в Интернете протоколами транспортного уровня являются TCP и UDP. Каждый из протоколов обладает собственным набором служб, предоставляющих услуги приложениям.

Взаимодействие между транспортным и сетевым уровнями

Как вы знаете, «соседом снизу» транспортного уровня в стеке протоколов является сетевой уровень. В то время как протоколы транспортного уровня обеспечивают логическое соединение между процессами, протоколы сетевого уровня поддерживают логическое соединение между оконечными системами. Данное различие весьма тонкое, однако от этого оно не менее значимо. Мы поясним его на следующем примере.

Представьте себе две семьи, одна из которых находится в Санкт-Петербурге, а другая во Владивостоке. В каждой семье живут 12 детей, при этом дети, живущие в Санкт-Петербурге, являются двоюродными братьями и сестрами детей, живущих во Владивостоке. Каждый из детей раз в неделю пишет письма всем своим 12 родственникам. Таким образом, каждую неделю оба дома отсылают друг другу 144 письма. В обеих семьях есть старшие дети (Анна в Санкт-Петербурге и Роберт во Владивостоке), которые отправляют и получают письма для всей семьи. Анна и Роберт каждую неделю собирают письма детей и передают их почтальону, а в день, когда приходит почта, получают ее и раздают своим братьям и сестрам.

В этом примере почтовая служба обеспечивает логическое соединение между двумя домами: она осуществляет доставку писем от одного дома к другому (а не от одного ребенка к другому). В то же время Анна и Роберт создают логическое соединение между своими братьями и сестрами, переписывающимися друг с другом. Обратите внимание, что с точки зрения детей Анна и Роберт представляют собой почтовую службу, несмотря на то что они на самом деле являются лишь конечными звеньями в процессе передачи писем. Этот пример очень наглядно иллюстрирует отношения между транспортным и сетевым уровнями:

- сообщения от приложений — это письма;
- процессы — это братья и сестры;
- оконечные системы — это дома;
- протокол транспортного уровня — это Анна и Роберт;
- протокол сетевого уровня — это почтовая служба (включая почтальона).

Продолжая данную аналогию, заметим, что Анна и Роберт выполняют свои обязанности, не выходя за пределы домов: к примеру, им не нужно искать нужные

письма в почтовом отделении или ходить туда, чтобы отправить письма. Протоколы транспортного уровня также «не выходят за пределы» оконечных систем. Их задачей является организация обмена сообщениями между прикладным и сетевым уровнями, при этом они не имеют никакого отношения к процессу передачи информации в «ядре» сети. Кроме того, как показано на рис. 3.1, работа маршрутизаторов также не зависит от информации, содержащейся в сообщениях транспортного уровня.

Пусть у Анны и Роберта имеются «заместители» — Марина и Александр, которые выполняют их обязанности, когда Анна и Роберт уезжают на каникулы. Поскольку Марина и Александр не являются старшими детьми в семьях, они не всегда выполняют поручения вовремя и, более того, иногда теряют письма. Таким образом, услуги, предоставляемые Мариной и Александром, отличаются от услуг, предоставляемых Анной и Робертом. Аналогично, в компьютерных сетях могут существовать несколько протоколов транспортного уровня, каждый из которых обладает собственными службами, предоставляющими услуги приложениям.

Услуги Анны и Роберта в значительной степени зависят от услуг, предоставляемых почтовой службой. Например, если почтовая служба не обеспечивает доставку писем между домами в течение определенного срока (например, трех дней), то Анна и Роберт не могут дать своим младшим братьям и сестрам никаких гарантий по поводу времени получения адресованных им писем. Аналогично службы сетевого уровня накладывают ограничения на возможности служб транспортного уровня. Если протокол сетевого уровня не гарантирует величину задержки или скорости передачи данных между хостами, протокол транспортного уровня не может обеспечить соответствующую задержку или скорость передачи данных между процессами.

Тем не менее существуют услуги, которые могут предоставляться службами транспортного уровня без участия служб сетевого уровня. Как мы увидим далее в этой главе, протоколы транспортного уровня могут выполнять надежную передачу данных даже при условии, что используемый протокол сетевого уровня допускает искажения, дублирования и потери пакетов. Другим примером (который детально будет рассмотрен в главе 7, посвященной безопасности компьютерных сетей) является шифрование данных на транспортном уровне. Шифрование обеспечивает конфиденциальность передаваемой информации независимо от того, гарантируется она протоколом сетевого уровня или нет.

Транспортный уровень в Интернете

Как уже неоднократно отмечалось, в Интернете (а точнее, в любой компьютерной сети, поддерживающей протокол TCP/IP) существуют два протокола транспортного уровня. Протокол *UDP* (User Datagram Protocol — протокол пользовательских дейтаграмм) предоставляет приложениям службу ненадежной передачи данных без установления логического соединения. Протокол *TCP* (Transmission Control Protocol — протокол управления передачей), напротив, предоставляет службу надежной передачи данных с установлением логического соединения. Создавая новое приложение, разработчик должен выбрать один из двух протоколов транспортного уровня для своего продукта. Как мы убедились в разделах «Программирование

TCP-сокетов» и «Программирование UDP-сокетов» главы 2, выбирать нужно при создании сокетов.

Для упрощения терминологии в контексте Интернета мы будем называть единицы обмена (PDU) транспортного уровня *сегментами*. Заметим, что в публикациях, посвященных Интернету (например, в документах RFC), сегментами, как правило, называют единицы обмена протокола TCP, при этом термин «дейтаграмма» используется для обозначения единиц обмена протокола UDP. В то же время в этих публикациях дейтаграммами называют единицы обмена сетевого протокола! Поскольку наша книга является введением в курс компьютерных сетей, мы сочли более наглядным употребление термина «сегмент» для обозначения единиц обмена обоих протоколов транспортного уровня, TCP и UDP, и термина «дейтаграмма» для обозначения единиц обмена сетевого уровня.

Перед тем как продолжить разговор о протоколах TCP и UDP, необходимо сказать несколько слов о сетевом уровне Интернета (подробно рассматриваемом в главе 4). В Интернете на сетевом уровне используется единственный протокол IP (Internet Protocol — Интернет-протокол). Протокол IP обеспечивает логическое соединение между хостами и предоставляет транспортному уровню услуги с доставкой «по возможности», или «с максимальными усилиями». Это означает, что IP пытается осуществить успешную доставку сегментов от отправителя до получателя, однако не дает никаких гарантий. Отсутствие гарантий распространяется не только на сам факт доставки сегментов, но и на сохранение порядка следования сегментов, а также на отсутствие искажений в доставленной информации. Говорят, что протокол IP предоставляет *ненадежную службу*. Каждый хост сети имеет не менее одного адреса сетевого уровня, часто называемого IP-адресом. В главе 4 мы подробно рассмотрим вопросы, относящиеся к IP-адресам, а здесь нам будет достаточно знать лишь об их существовании.

Кратко рассмотрев модель обслуживания протокола IP, займемся обобщением наших знаний о службах протоколов UDP и TCP. Основной задачей UDP и TCP является обеспечение обмена данными между процессами, выполняющимися на оконечных системах, при помощи службы обмена данными между оконечными системами, предоставляемой протоколом сетевого уровня. Такое «продолжение» соединения между оконечными системами до уровня процессов называется *мультиплексированием* и *демультиплексированием на транспортном уровне* и рассматривается в следующем разделе. Протоколы UDP и TCP также обеспечивают отсутствие искажений данных при передаче, включая в свои заголовки поля обнаружения ошибок. Заметим, что протокол UDP предоставляет минимальный набор служб транспортного уровня: службы обмена данными между процессами и контроля ошибок. Как и протокол IP, UDP обеспечивает ненадежную передачу данных, не предоставляя гарантий доставки и отсутствия ошибок. Подробное описание UDP будет приведено в разделе «Протокол UDP — передача без установления соединения» этой главы.

Протокол TCP обладает более широким набором служб по сравнению с UDP. Прежде всего TCP обеспечивает *надежную передачу данных*. При помощи средств контроля переполнения, порядковых номеров, квитанций и таймеров (которые рассматриваются позже) TCP гарантирует, что вся переданная информация будет по-

лучена в правильном порядке и без искажений. Таким образом, протокол TCP, используя службу ненадежной передачи данных между оконечными системами протокола IP, обеспечивает надежную передачу данных между процессами. *Контроль перегрузки* является одной из функций TCP, которую трудно отнести к услуге, предоставляемой приложению; скорее, контроль перегрузки помогает повысить качество обслуживания всех пользователей сети. Цель контролирования перегрузки — предотвращение слишком интенсивного трафика между парами оконечных систем, вызывающего перегрузку линий связи и маршрутизаторов. Фактически действие механизма контроля перегрузки заключается в разделении пропускной способности линии связи поровну между всеми TCP-соединениями. В свою очередь, такое разделение обеспечивается регулированием скорости передачи каждой оконечной системой. Протокол UDP не контролирует трафик, а следовательно, приложение, использующее UDP, может осуществлять передачу данных с любой скоростью в течение сколь угодно долгого времени.

Не удивительно, что протокол TCP, обеспечивающий надежную передачу данных и контролирование перегрузки, является весьма сложным. Принципы надежной передачи данных и контроля перегрузки изложены в нескольких разделах этой главы; кроме того, отдельный раздел посвящен самому протоколу TCP. Придерживаясь нашего подхода к изложению материала, мы сначала будем рассматривать общие задачи, а затем переходить к их практическому решению в протоколе TCP. Однако перед тем как начать детальное изучение TCP, обратимся к упомянутым выше операциям мультиплексирования и демультиплексирования на транспортном уровне.

Мультиплексирование и демультиплексирование

В этом разделе мы рассмотрим операции мультиплексирования и демультиплексирования на транспортном уровне, «продолжающие» соединение между оконечными системами до уровня соединения между процессами. Для того чтобы конкретизировать обсуждение, мы будем рассматривать службу мультиплексирования и демультиплексирования на транспортном уровне в контексте Интернета. Тем не менее эта служба необходима во всех компьютерных сетях.

Сетевой уровень принимающей оконечной системы передает полученные сегменты транспортному уровню, а транспортный уровень передает данные, содержащиеся в сегментах, процессам, которым они предназначены. Рассмотрим простой пример. Предположим, что вы находитесь за своим персональным компьютером и загружаете web-страницы; при этом на компьютере одновременно открыты два Telnet-сеанса и один FTP-сеанс. Таким образом, данные, поступающие из Интернета, могут быть адресованы одному из четырех процессов: FTP-процессу, HTTP-процессу, одному из двух Telnet-процессов. Протокол транспортного уровня должен определить, какому процессу предназначается каждый принятый сегмент Данных. Рассмотрим, каким образом это происходит.

Вернемся к разделам «Программирование TCP-сокетов» и «Программирование UDP-сокетов» в главе 2. Как мы выяснили, процесс, представляющий собой часть

приложения, имеет собственный *сокет,* или «дверь», через которую осуществляется обмен данными с другими процессами. Таким образом, как показано на рис. 3.2, транспортный уровень фактически передает данные не процессу, а сокету. Поскольку принимающий хост может иметь несколько сокетов одновременно, каждый сокет имеет уникальный идентификатор. Формат идентификатора, как мы увидим позже, зависит от того, к какому протоколу принадлежит сокет, к TCP или к UDP.

Условные обозначения:

(^) Процесс И **Сокет**

Рис. 3.2. Мультиплексирование и демультиплексирование на транспортном уровне

Теперь рассмотрим, каким образом принятые сегменты транспортного уровня направляются в соответствующие сокеты. Для этой цели каждый сегмент содержит набор специальных полей. Транспортный уровень принимающего хоста анализирует содержимое этих полей, идентифицирует сокет, которому предназначен сегмент, и передает ему данные сегмента. Процедура вручения данных сокету носит название *демультиплексирования.* Сбор фрагментов данных, поступающих на транспортный уровень хоста-отправителя из различных сокетов, создание сегментов путем присоединения заголовка (который используется при демультиплексировании) к каждому фрагменту и передача сегментов сетевому уровню называется *мультиплексированием.* Транспортный уровень среднего хоста на рисунке должен демультиплексировать сегменты, поступающие от сетевого уровня в адрес процессов Pj и P^2; это достигается путем направления данных, содержащихся в сегментах, в сокеты соответствующих процессов. Кроме того, транспортный уровень среднего хоста собирает данные, поступающие из сокетов процессов P^t и P^2, формирует сегменты и передает их сетевому уровню.

Для того чтобы наглядно представить процесс демультиплексирования, обратимся к примеру с двумя семьями, приведенному в предыдущем разделе. Каждый ребенок идентифицируется своим именем. Получая от почтальона новые письма, Роберт

выполняет операцию демультиплексирования, читая имя получателя и вручая письма своим братьям и сестрам. Анна выполняет операцию мультиплексирования, собирая письма от членов своей семьи и передавая их почтальону.

Узнав о роли мультиплексирования и демультиплексирования на транспортном уровне, обратимся к их практической реализации на оконечных системах. Как мы знаем, мультиплексирование на транспортном уровне требует наличия у сокетов уникальных идентификаторов, а у сегментов — специальных полей, содержащих номера сокетов, которым они предназначены. Как показано на рис. 3.3, сегменты транспортного уровня содержат *поле номера порта отправителя* и *поле номера порта получателя* (в сегментах протоколов UDP и TCP имеются также другие поля, о которых будет рассказано в следующих разделах этой главы). Номер порта представляет собой 16-разрядное число, принимающее значения от 0 до 65 535. Номера в диапазоне от 0 до 1023 предназначены для использования в популярных протоколах прикладного уровня (таких, как HTTP и FTP, имеющих порты с номерами 80 и 21 соответственно). Список зарезервированных номеров портов приведен в документе RFC 1700, а его регулярно обновляемая версия находится в Интернете по адресу http://www.iana.org (RFC 3232). При разработке новых приложений (примеры которых приведены в разделах «Программирование TCP-сокетов», «Программирование UDP-сокетов» и «Разработка простого web-сервера» главы 2) необходимо присваивать им собственные номера портов.

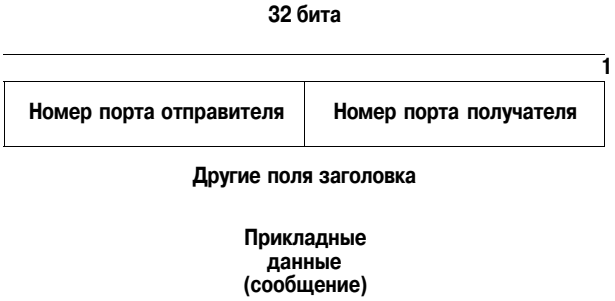


Рис. 3.3. Поля нЬмеров портов отправителя и получателя в сегменте транспортного уровня

Теперь становится ясен один из вариантов организации службы демультиплексирования: каждому сокету сопоставляется номер порта, а сегмент направляется тому сокету, чей номер совпадает со значением, содержащимся в поле номера порта получателя. Затем данные сегмента передаются через сокет соответствующему процессу. Как мы увидим, подобная схема характерна для протокола UDP; процесс мультиплексирования/демультиплексирования в протоколе TCP несколько сложнее.

Мультиплексирование и демультиплексирование без установления логического соединения

Вспомним о том, что в программе на языке Java, выполняющейся на оконечной системе, создание UDP-сокета производится командой

```
DatagramSocket mySocket = new DatagramSocketO:
```

При выполнении этой команды транспортный уровень автоматически связывает номер порта с создаваемым сокетом. Номером порта является любое число от 1024 до 65 535, не используемое в текущий момент другим UDP-портом. Номер порта может быть задан явно:

```
DatagramSocket mySocket = new DatagramSocket(19157);
```

В этом случае UDP-сокету будет сопоставлен номер порта 19 157. Явное указание номера порта необходимо в случаях, когда разработчик создает приложение, поддерживаемое каким-либо популярным протоколом, занимающим один из «хорошо известных» номеров порта. Обычно на клиентской стороне приложения транспортный уровень связывает сокет с номером порта автоматически, в то время как на серверной стороне требуется явное указание последнего.

После того как номера портов UDP-сокетов определены, мы можем точно описать процедуру мультиплексирования и демультиплексирования протокола UDP. Предположим, что процесс с номером сокета 19 157, выполняющийся на хосте А, пытается передать массив данных процессу с номером сокета 46 428, выполняющемуся на хосте В. Транспортный уровень хоста А создает сегмент, включающий данные процесса, номер порта отправителя (19 157), номер порта получателя (46 428), а также две другие величины, которые сейчас не представляют для нас интереса и будут рассмотрены позже. Сформированный сегмент передается сетевому уровню, который создает IP-дейтаграмму и «по-возможности» доставляет ее хосту В. В случае успешного получения сегмента хостом-получателем последний анализирует значение поля номера порта получателя и направляет сегмент сокету с номером 46 428. Обратите внимание, что на хосте В могут одновременно выполняться несколько процессов, каждый из которых обладает сокетом с уникальным номером порта. Каждый получаемый сегмент анализируется и направляется тому сокету, номер порта которого совпадает со значением поля номера порта получателя сегмента.

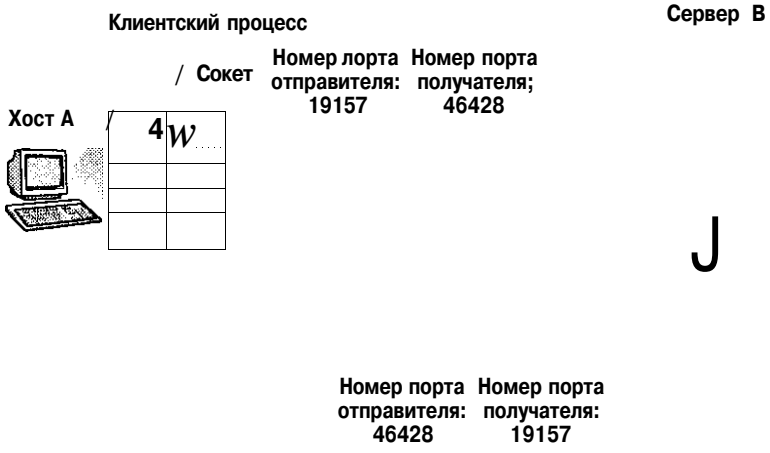


Рис. 3.4. При передаче в обратном направлении номера портов отправителя и получателя меняются местами

Отметим, что любой UDP-сокет однозначно идентифицируется совокупностью IP-адреса хоста назначения и номера порта. Таким образом, если два UDP-сегмента имеют разные IP-адреса и/или номера портов отправителя, но одинаковые IP-адреса и номера портов получателя, оба сегмента будут переданы одному и тому же процессу.

Не удивительно, если у вас сразу возник вопрос: для чего нужен номер порта отправителя? Как показано на рис. 3.4, номер порта отправителя используется как часть «обратного адреса». Если у получателя возникнет необходимость в отправке ответа, он скопирует значение поля номера порта отправителя полученного сегмента в поле номера порта получателя ответного сегмента (мы назвали номер порта частью «обратного адреса» потому, что полный адрес также включает IP-адрес отправителя). В качестве примера обратимся к серверной программе UDPServer.java, созданной в разделе «Разработка простого web-сервера» главы 2. В ней используется специальный метод для извлечения номера порта отправителя из принятого сегмента; извлеченный номер порта служит в качестве номера порта получателя для создаваемого ответного сегмента.

Мультиплексирование и демультиплексирование с установлением логического соединения

Для того чтобы понять суть демультиплексирования в протоколе TCP, необходимо сначала подробнее рассмотреть TCP-сокеты и процесс установления TCP-соединения. Отличие TCP-сокета от UDP-сокета заключается в том, что TCP-сокет идентифицируется при помощи не двух, а четырех составляющих: IP-адреса отправителя, номера порта отправителя, IP-адреса получателя и номера порта получателя. Все четыре компонента используются хостом-получателем в процессе демультиплексирования (направления в нужный сокет) получаемых сегментов. В отличие от протокола TCP, сегменты с разными IP-адресами или номерами порта отправителя будут переданы разным сокетам даже при совпадении IP-адресов и номеров портов получателя. Обратимся к программам, приведенным в разделе «Программирование TCP-сокетов» главы 2. Эти программы обладают следующими особенностями.

- Программа-сервер использует «впускающий» сокет с номером порта 6789, принимающий запросы от клиентов на установление TCP-соединения.
- Программа-клиент генерирует запрос на установление соединения командой

  ```
  Socket clientSocket = new SocketC'serverHostName". 6789);
  ```

- Запрос на установление логического соединения представляет собой не что иное, как TCP-сегмент с номером порта 6789 и специальным битовым набором в заголовке (более подробное описание будет приведено в разделе «Протокол TCP — передача с установлением соединения» этой главы). Кроме того, сегмент также включает номер порта отправителя, определяемый программой-клиентом. Команда, приведенная выше, создает TCP-сокет клиентского процесса, через который клиент осуществляет обмен информацией с сервером.
- Получив запрос на установление соединения, серверный процесс создает сокет соединения командой

  ```
  Socket connectionSocket = welcomeSocket.acceptO;
  ```

- При обработке сегмента с запросом на установление логического соединения сервер использует четыре параметра: номер порта отправителя, IP-адрес отправителя, номер порта *получателя* и собственный IP-адрес. Создаваемый сокет соединения идентифицируется совокупностью этих четырех параметров. Все сегменты, содержащие в соответствующих полях значения, совпадающие с параметрами сокета соединения, направляются (демультиплексируются) в этот сокет. После того как TCP-соединение установлено, клиент и сервер могут начинать обмен данными.

Сервер на хосте может одновременно поддерживать множество TCP-сокетов, каждый из которых связан с процессом и идентифицируется четырьмя приведенными выше параметрами. Когда очередной сегмент принимается сервером, при его демультиплексировании используются четыре поля, соответствующие параметрам, идентифицирующим сокет.

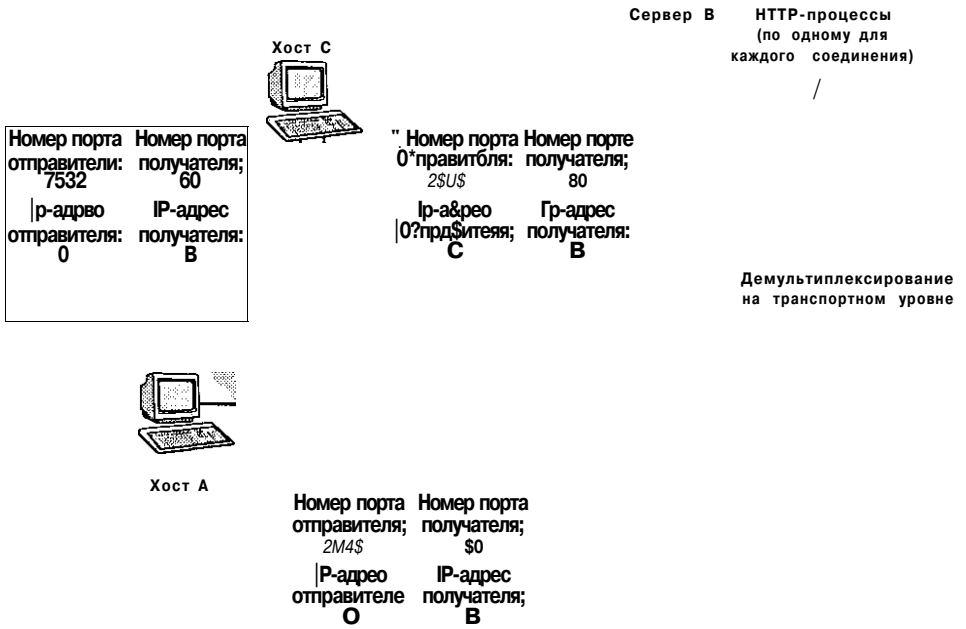


Рис. 3.5. Два клиента используют порт с номером 80 для взаимодействия с web-сервером

Рисунок 3.5 иллюстрирует ситуацию открытия трех HTTP-сеансов с сервером В: двух сеансов клиентом С и одного клиентом А. Все три хоста, А, В и, С, имеют уникальные IP-адреса. Для двух HTTP-соединений хост С использует номера портов 26 145 и 7532. Поскольку хост А выбирает номера портов независимо от хоста С, возможна ситуация, когда для своего HTTP-соединения с сервером В хост А также использует порт с номером 26 145. Тем не менее процесс демультиплекси-

рования будет выполняться сервером корректно, поскольку IP-адреса отправителей сегментов для хостов A и C различны.

Web-серверы и TCP

Перед тем как завершить разговор о мультиплексировании и демультиплексировании, необходимо сказать несколько слов о web-серверах и об использовании ими номеров портов. Предположим, что на хосте выполняется web-сервер (например, Apache) с портом номер 80. Когда клиенты (к примеру, браузеры) формируют сегменты для передачи серверу, во всех сегментах номер порта получателя получает значение 80. Как упоминалось выше, сервер различает подобные сегменты по IP-адресам и номерам портов отправителей.

Как правило, для каждого нового соединения с клиентом сервер либо порождает новый процесс, либо создает поток выполнения в рамках существующего процесса. Сервер, представленный на рис. 3.5, относится к первому из типов. Как видно, каждый из процессов имеет собственный сокет соединения, с помощью которого сервер обменивается данными с клиентом. Необходимо отметить, что сокеты соединения и процессы не всегда состоят в отношениях «один к одному». Современные высокопроизводительные web-серверы зачастую используют единственный процесс, создающий потоки выполнения для соединений с клиентами, при этом каждый поток располагает собственным сокетом (поток выполнения можно рассматривать как упрощенный процесс в рамках «нормального» процесса). В первом задании по программированию главы 2 требовалось создать сервер, который функционирует именно таким образом. Подобные серверы позволяют одному процессу задействовать несколько сокетов соединения с разными идентификаторами.

Если клиент и сервер используют протокол HTTP с постоянными TCP-соединениями, то они могут многократно обмениваться сообщениями через один и тот же сокет сервера. В противном случае для каждой пары запрос/ответ устанавливается новое TCP-соединение, которое разрывается по окончании процесса передачи ответа. Такой механизм привносит дополнительную нагрузку на сервер, ухудшая качество обслуживания (несмотря на то, что некоторые операционные системы имеют средства борьбы с этой проблемой). Читателям, интересующимся вопросами поддержки постоянных и непостоянных HTTP-соединений операционными системами, рекомендуем обратиться к дополнительным источникам информации [359].

Ознакомившись с мультиплексированием и демультиплексированием на транспортном уровне, перейдем к подробному рассмотрению UDP — протокола транспортного уровня, используемого в Интернете. Как вскоре станет ясно, службы протокола UDP помимо мультиплексирования и демультиплексирования предоставляют приложениям некоторые дополнительные услуги.

Протокол UDP — передача без установления соединения

В этом разделе мы детально рассмотрим функции и принципы работы протокола UDP. При необходимости мы рекомендуем вам вернуться к разделу «Принципы

работы протоколов прикладного уровня» в главе 2, где приведен обзор модели обслуживания UDP, а также к разделу «Программирование UDP-сокетов», в котором приведен пример приложения, использующего протокол UDP.

Представьте себе, что вам необходимо разработать максимально простой, без лишних функций, протокол транспортного уровня. Как бы вы стали решать эту задачу? Наиболее простым мыслимым способом является создание такого протокола, который не производит никаких действий с данными. На передающей стороне сообщения приложений без изменений передаются сетевому уровню, а на приемной стороне выполняется передача сообщений от сетевого уровня прикладному. Ясно, что такой протокол не является жизнеспособным: из предыдущего раздела следует, что протоколу как минимум необходимо выполнять операции мультиплексирования и демультиплексирования, обеспечивающие корректный обмен данными между сетевым уровнем и прикладными процессами.

Протокол UDP, описанный в документе RFC 768, выполняет минимум действий, необходимых для протокола транспортного уровня. Фактически функции UDP сводятся к операциям мультиплексирования и демультиплексирования, а также несложной проверке наличия ошибок в данных. Таким образом, при использовании UDP приложение почти напрямую взаимодействует с протоколом сетевого уровня IP. UDP получает сообщения от прикладного уровня, добавляет к ним поля номеров портов отправителя и получателя для демультиплексирования приемной стороной, а также два других специальных поля и передает полученный сегмент сетевому уровню. Сетевой уровень заключает сегмент в дейтаграмму и «по возможности» передает ее хосту назначения. Если последний успешно получает сегмент, протокол UDP с помощью поля номера порта получателя направляет данные сегмента нужному процессу. Обратите внимание на то, что протокол UDP не предусматривает процедуры рукопожатия перед началом передачи сегментов. Поэтому говорят, что UDP осуществляет передачу данных без установления соединения.

Примером протокола прикладного уровня, использующего службы протокола UDP, является DNS. Когда DNS-приложение генерирует запрос, оно создает DNS-сообщение и передает его протоколу UDP. Не выполняя рукопожатия с хостом назначения, UDP добавляет к сообщению заголовочные поля и передает его сетевому уровню. Сетевой уровень заключает сообщение в дейтаграмму и передает ее серверу имен. DNS-приложение, создавшее запрос, ожидает ответа и в случае, если ответ не приходит (возможно, по причине потери запроса или ответа), генерирует запрос другому серверу имен либо информирует вызывающее приложение о том, что получение IP-адреса невозможно.

После приведенных выше рассуждений вполне уместным становится вопрос: есть ли у протокола UDP такие преимущества перед TCP, которые могут заставить разработчика создавать свое приложение с поддержкой UDP, а не TCP? Не удивительно, что ответ на этот вопрос положителен, и ниже перечислены четыре главные достоинства протокола UDP.

- *Отсутствие процедуры установления соединения.* Как мы увидим позднее, протокол TCP перед началом передачи данных требует тройного рукопожатия. Протокол UDP освобожден от подобной «формальности» и поэтому не вносит

дополнительную задержку в процесс передачи. Это является главной причиной для применения UDP прикладным протоколом DNS. Если бы DNS использовал службы TCP, процесс доставки IP-адресов был бы значительно более медленным. Протокол HTTP, напротив, задействует TCP, поскольку при доставке web-страниц, содержащих текстовую информацию, важно отсутствие искажений в передаваемых данных. Стоит заметить, что необходимость в установлении TCP-соединений вносит немалый вклад в задержку доставки необходимых объектов.

- *Отсутствие информации о состоянии соединения.* Протокол TCP поддерживает информацию о состоянии TCP-соединения, что вызывает необходимость в наличии буферов для промежуточного хранения информации о приеме и передаче, параметров контроля перегрузки, порядковых номеров и номеров квитанций. Как мы увидим в разделе «Протокол TCP — передача с установлением соединения», информация о состоянии соединения необходима для реализации протоколом TCP служб надежной передачи данных и контроля перегрузки. UDP не поддерживает информацию о состоянии соединения, а следовательно, не требует учета перечисленных параметров. Это позволяет UDP-серверу одновременно обслуживать гораздо больше клиентов, чем TCP-серверу.
- *Небольшой размер заголовка.* Заголовок UDP-сегмента имеет длину всего лишь 8 байт, в то время как длина заголовка TCP составляет 20 байт.
- *Улучшенный механизм управления передачей данных приложением.* При использовании протокола UDP данные, поступающие от приложения, сразу же упаковываются в сегмент и передаются сетевому уровню. Протокол TCP, располагая средствами контроля перегрузки, может приостановить процесс передачи сегмента вниз по стеку протоколов в случае, если на пути между отправителем и получателем имеются перегруженные линии связи. Кроме того, пересылка сегмента осуществляется протоколом до тех пор, пока не будет получено подтверждение о том, что адресат получил его (при этом затрачиваемое на пересылку время не учитывается). Поскольку приложения, работающие в реальном времени, обычно налагают ограничения на минимальную скорость передачи данных, не допускают значительных задержек сегментов, но в то же время толерантны к потере данных, службы протокола TCP малопригодны для таких приложений. Напротив, службы протокола UDP значительно лучше отвечают требованиям приложений реального времени, которые используют UDP в сочетании с собственными средствами обмена данными между процессами.

В таблице 3.1 представлены сведения о протоколах прикладного и транспортного уровней, используемых популярными Интернет-приложениями. Как и следовало ожидать, приложения электронной почты, web-приложения и приложения для передачи файлов строятся на основе протокола TCP, поскольку им необходима надежная передача данных. В сферу применения UDP входит протокол RIP, предназначенный для обновления информации таблиц маршрутизации (см. главу 4), поскольку обновления, когда потерянная информация заменяется новой, носят периодический характер (сообщения генерируются приблизительно каждые 5 мин). UDP также используется для поддержки протокола сетевого администрирования

SNMP (см. главу 8); выбор UDP в этом случае объясняется тем, что администрирующие приложения должны нормально функционировать при отклонениях от нормы в работе сети, то есть в ситуациях, когда надежная передача данных и контроль перегрузки могут серьезно ухудшить качество обслуживания. Протоколом UDP также поддерживается протокол DNS, поскольку UDP позволяет избежать дополнительных задержек на установление соединения. Кроме того, UDP используется почти всеми мультимедиа-приложениями: Интернет-телефонией, видеоконференциями в реальном времени, а также потоковыми аудио и видео. Мультимедиа-приложения более детально рассматриваются в главе 6, здесь же только заметим, что эти приложения допускают потери небольшой доли пакетов, поэтому надежная передача данных не является для них необходимой. Кроме того, из-за затрат на контролирование перегрузки протоколом TCP значительно ухудшается качество функционирования приложений реального времени (видеоконференций и Интернет-телефонии). Эти причины обусловливают выбор протокола UDP для поддержки перечисленных приложений.

Таблица 3.1. Протоколы прикладного и транспортного уровней, используемые Интернет-приложениями

Приложение	Прикладной протокол	Транспортный протокол
Электронная почта	SMTP	TCP
Доступ с удаленного терминала	Telnet	TCP
Web	HTTP	TCP
Передача файлов	FTP	TCP
Удаленный файловый сервер	NFS	UDP или TCP
Потоковое мультимедиа	Нестандартный	UDP или TCP
Интернет-телефония	Нестандартный	Как правило, UDP
Сетевое администрирование	SNMP	Как правило, UDP
Протокол маршрутизации	RIP	Как правило, UDP
Трансляция имен	DNS	Как правило, UDP

Несмотря на широкое применение протокола UDP для мультимедиа-приложений, вопрос о его рациональности, как минимум, остается открытым. Как известно, в UDP не предусмотрен контроль перегрузки. Контролирование перегрузки позволяет предотвратить возникновение такого состояния сети, когда большая часть ее работы выполняется «вхолостую». Например, если все пользователи Интернета одновременно станут просматривать потоковое видео с высоким качеством изображения, то процент потерянных вследствие перегрузки пакетов окажется таким большим, что в результате никто ничего не увидит. Таким образом, отсутствие механизмов контроля перегрузки потенциально представляет собой весьма серьезную проблему [151]. Многие исследователи предлагают ввести механизмы, принуждающие все источники, передающие информацию (в том числе и приложения, поддерживаемые протоколом UDP), выполнять адаптивный контроль перегрузки [156, 305].

Перед тем как перейти к изучению структуры UDP-сегмента, отметим, что надежная передача данных приложения возможна даже при использовании UDP. Это достигается путем включения механизмов обеспечения надежной передачи (например, квитирования и повторных передач, которые мы рассмотрим чуть позже) в само приложение. На практике данная задача часто оказывается весьма нетривиальной, и для создания корректно функционирующего приложения может потребоваться много времени. Вместе с тем обеспечение надежной передачи данных приложением позволяет «убить двух зайцев», удовлетворяя требование к обслуживанию процесса и предоставляя возможность преодолеть недостатки протокола TCP (например, более низкую скорость передачи). Подобным образом строится большинство современных потоковых приложений [418].

Структура UDP-сегмента

Структура UDP-сегмента, представленная на рис. 3.6, описана в RFC 768. Данные приложения размещаются в поле данных сегмента; например, в поле данных может быть размещено DNS-сообщение (запрос или ответ) или сэмпл потокового аудио. Заголовок UDP-сегмента состоит из четырех двухбайтовых полей. Номера портов отправителя и *получателя* позволяют хосту назначения направить данные сегмента нужному сокету (другими словами, осуществить процедуру демультиплексирования). Контрольная сумма предназначена для проверки ошибок в полученных данных. На самом деле вычисление контрольной суммы производится также для некоторых полей IP-заголовка, но мы не будем акцентировать внимание на деталях, которые сейчас не представляют для нас важности (вычисление контрольной суммы рассматривается отдельно). Основные подходы к обнаружению ошибок изложены в главе 5. Поле длины указывает на размер UDP-сегмента в байтах, включая заголовок.

Рис. 3.6. Структура UDP-сегмента

Контрольная сумма UDP-сегмента

Как было показано ранее, контрольная сумма UDP-сегмента предназначена для обнаружения ошибок, то есть определения, были ли какие-либо биты сегмента искажены в процессе передачи (например, в результате помех на линии связи или промежуточного хранения в маршрутизаторе). Протокол UDP на передающей стороне вычисляет

дополнение до 1 суммы всех 16-разрядных слов сегмента, игнорируя происходящие при суммировании переполнения. Результат вычисления заносится в поле контрольной суммы сегмента. Материалы об эффективном вычислении контрольной суммы можно найти в RFC 1071, а для ознакомления со способами практической реализации вычислений рекомендуем обратиться к дополнительным источникам информации [491, 492]. В качестве примера рассмотрим три 16-разрядных слова:

0110011001100110

0101010101010101

0000111100001111

Сумма первых двух слов равна:

0110011001100110

0101010101010101

1011101110111011

Сложение полученной суммы с третьим словом дает результат:

1011101110111011

0000111100001111

1100101011001010

Дополнение до 1 вычисляется путем замены всех нулей суммы единицами и наоборот. Дополнением до 1 слова 1100101011001010 является слово 0011010100110101, которое и будет записано в поле контрольной суммы. На приемной стороне производится суммирование всех слов сегмента, включая поле контрольной суммы. Если при передаче не произошло искажения ни одного из битов, результатом суммирования является 1111111111111111. Присутствие хотя бы одного нулевого бита в сумме свидетельствует о наличии ошибок в данных.

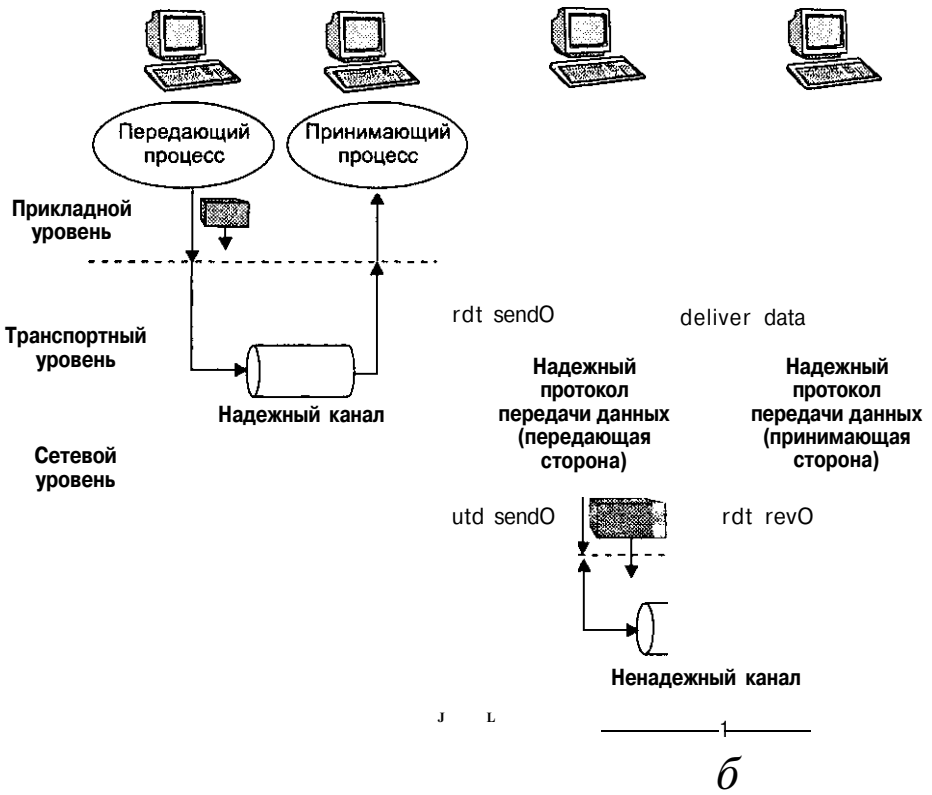
Возможно, вызывает удивление, что протокол UDP в первую очередь осуществляет проверку данных на наличие ошибок, поскольку известно, что эту функцию обычно осуществляют протоколы канального уровня (например, популярный протокол Ethernet). Причина заключается в том, что на самом деле не все протоколы физического уровня выполняют такую проверку, и, следовательно, существует вероятность, что при передаче искажение данных будет не обнаружено. Поскольку протокол IP может работать в сочетании практически с любым протоколом канального уровня, функция обнаружения ошибок на транспортном уровне необходима для повышения надежности передачи. Заметим, что протокол UDP способен лишь обнаруживать ошибки, однако не располагает средствами их исправления. Некоторые реализации UDP удаляют искаженный сегмент, а некоторые передают его прикладному уровню с соответствующим предупреждением.

На этом мы завершим рассказ о протоколе UDP. Далее в центре нашего внимания окажется протокол TCP, который предоставляет приложениям набор услуг, не поддерживаемых протоколом UDP. Заметим, что структура TCP при этом значительно сложнее. Перед тем как начать непосредственное знакомство с TCP, мы немного вернемся назад и вспомним основные принципы надежной передачи данных.

Принципы надежной передачи данных

В этом разделе мы рассмотрим общие принципы надежной передачи данных. Проблема надежной передачи данных является одной из центральных для компьютерных сетей и проявляется не только на транспортном, но также на сетевом и прикладном уровнях. Большинство принципов, изложенных ниже, реализованы в протоколе TCP; мы убедимся в этом в следующем разделе.

На рис. 3.7 приведена схема надежной передачи данных. Служба надежной передачи данных обслуживает канал, по которому осуществляется надежная передача сообщений верхних уровней коммуникационной модели. При надежной передаче не происходит искажений битов, то есть изменений их значений с 0 на 1 или наоборот; кроме того, данные доставляются в том порядке, в котором они были отправлены. Именно такая модель обслуживания используется протоколом TCP.



Условные обозначения:

WrR **Данные** *ЩШЯ* **Пакет**

Рис. 3.7. Надежная передача данных: а — модель обслуживания; *б* — реализация модели обслуживания

Главная проблема обеспечения надежной передачи данных протоколом транспортного уровня заключается в том, что протокол более низкого уровня может не поддерживать надежную передачу. Подобная ситуация характерна для протокола TCP, который использует службу ненадежной передачи протокола сетевого уровня IP. Тем не менее эта проблема может касаться не только транспортного, но сетевого и даже канального уровней: они используют службы передачи данных сетей и отдельных линий связи, которые очевидно не являются надежными.

В этом разделе мы займемся созданием клиентской и серверной сторон протокола надежной передачи данных, постепенно усложняя модель канала передачи. Интерфейс будущего протокола приведен на рис. 3.7, *б*. Передающая сторона вызывается с расположенного выше уровня методом rdt_send(). Здесь rtd *является* аббревиатурой от reliable data transfer (надежная передача данных), а _send указывает на передающую сторону протокола (вообще говоря, первым шагом в создании протокола является выбор для него подходящего имени). На принимающей стороне при появлении нового пакета выполняется метод rdt_rcv(), а метод deliver_data() передает данные прикладному уровню. Для обозначения единицы обмена мы далее будем использовать термин «пакет» вместо ставшего привычным термина «сегмент». Мы выходим за пределы Интернета и рассматриваем компьютерные сети в целом, поэтому использование общей терминологии в данном случае более предпочтительно.

В этом разделе мы ограничимся лишь *однонаправленной,* или *полудуплексной,* передачей данных, то есть передачей от источника к приемнику. *Двунаправленная,* или *дуплексная,* передача концептуально не сложнее, однако делает описание весьма утомительным. Несмотря на свое название, однонаправленная передача предусматривает движение пакетов в обоих направлениях, как и показано на рис. 3.7. Мы убедимся в том, что принимающей и передающей сторонам необходимо, кроме данных, обмениваться также различной управляющей информацией. Передача пакетов сторонами протокола друг другу будет производиться с помощью метода udt_send(), где udt означает unreliable data transfer (ненадежная передача данных).

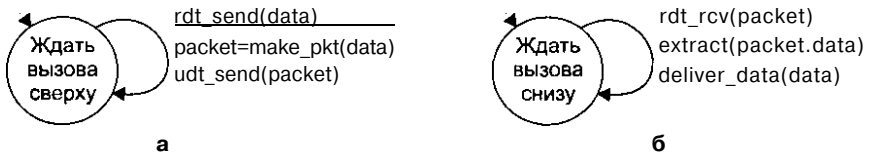
Создание протокола надежной передачи данных

Ниже мы создадим несколько протоколов, каждый из которых будет лучше предыдущего. В конечном счете, мы достигнем поставленной цели и получим протокол, действительно обеспечивающий надежную передачу данных.

Надежная передача данных по абсолютно надежному каналу

Сначала рассмотрим простейший случай, когда канал, по которому передаются данные, является абсолютно надежным. Протокол rdt 1.0, обеспечивающий передачу по такому каналу, также является тривиальным. На рис. 3.8 приведена схема конечных автоматов, описывающая этот протокол. Фрагмент *а* соответствует передающей стороне, фрагмент *б* — принимающей. При описании протокола важным является наличие *двух* моделей конечных автоматов, соответствующих каждой из его сторон. В данном случае оба автомата имеют единственное состояние. Стрел-

ки на схемах обозначают переходы между состояниями автомата; поскольку автоматы имеют единственное состояние, переход возможен только в то же состояние. Ниже мы разработаем более сложные модели. Событие, вызывающее переход, указывается над горизонтальной чертой возле стрелки, а действие, предпринимаемое при наступлении этого события, — под горизонтальной чертой. Отсутствие события или ответного действия явно обозначается символом «?». Пунктирная стрелка указывает на начальное состояние автомата. В более сложных моделях, имеющих несколько состояний, определение начального состояния является важным.



а б

Рис. 3.8. Протокол rdt 1.0 — передача по абсолютно надежному каналу связи: а — передающая сторона; б — принимающая сторона

Передающая сторона протокола принимает данные от верхнего уровня при помощи метода rdt_send(data), методом make_pkt(data) создает пакет с данными и отправляет созданный пакет в канал. На практике событие rdt_send(data) является результатом вызова процедуры rdt_send(data) прикладной программой.

На приемной стороне получение пакета rdt от протокола сетевого уровня происходит при наступлении события rdt_rcv(packet). Далее данные пакета извлекаются при помощи метода extract(packet, data), а метод deLiver_data(data) передает данные приложению. На практике событие rdt_rcv(packet) является результатом вызова процедуры rdt_rcv() протоколом нижнего уровня.

В нашем простом протоколе пакет соответствует единице обмена; кроме того, все пакеты движутся в направлении от передающей стороны к принимающей, поскольку отсутствие ошибок избавляет протокол от необходимости поддерживать обратную связь. Обратите внимание на то, что скорость приема данных совпадает со скоростью их передачи. Это позволяет не думать над разработкой механизма снижения скорости передачи по требованию принимающей стороны протокола.

Надежная передача данных по каналу, допускающему искажение битов

Рассмотрим более реалистичную модель канала, допускающую искажение некоторых битов при передаче. Обычно искажения имеют место из-за влияния физических факторов, проявляющихся в процессе передачи или промежуточного хранения пакетов, а также при распространении сигнала в физической среде. При построении новой модели мы по-прежнему будем использовать допущение о том, что все передаваемые пакеты достигают получателя и их порядок при этом не нарушается.

Перед тем как начать разработку протокола, обеспечивающего надежную передачу данных по описанному каналу, представим себе следующую аналогию. Предпо-

ложим, что вы диктуете вашему знакомому длинное сообщение по телефону. Каждый раз после того, как продиктованное предложение принято и записано на бумагу, ваш знакомый говорит: «Да!» Если знакомый вас не понял, он просит повторить предложение еще раз. Этот протокол передачи голосовых сообщений содержит *положительные* («Да!») и *отрицательные* («Повтори это еще раз!») *квитанции*. Квитанции служат для того, чтобы принимающая сторона могла уведомлять передающую сторону о том, содержатся ли искажения в каждом из принятых предложений. В сфере компьютерных сетей протоколы надежной передачи данных, обладающие подобным механизмом многократного повторения передачи, называются протоколами с *автоматическим запросом повторной передачи* (Automatic Repeat reQuest, ARQ).

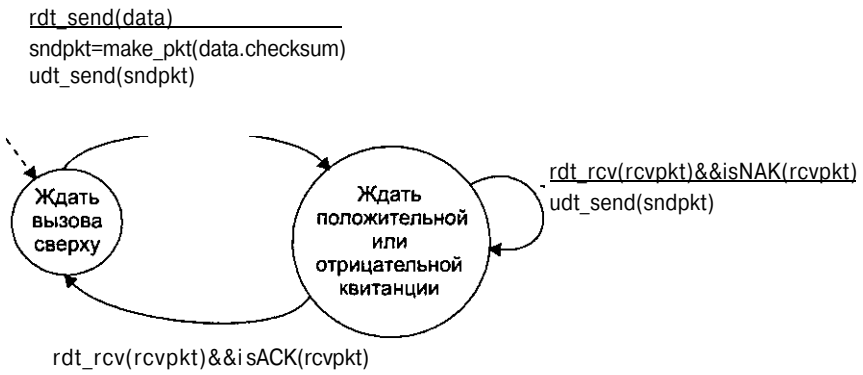
Для разрешения проблем искажения битов в ARQ-протоколах используются три дополнительных механизма.

- *Обнаружение ошибок.* Как следует из названия, данный механизм позволяет определять наличие искаженных битов в принятых данных. В предыдущем разделе было показано, что протокол UDP использует для этой цели значение контрольной суммы. Подробное рассмотрение методов обнаружения и исправления ошибок приводится в главе 5; сейчас нам необходимо знать лишь о том, что эти методы основаны на передаче специальных дополнительных битов, не входящих в информационную часть пакета. Эти биты будут помещены в поле контрольной суммы протокола rdt 2.0.
- *Обратная связь с передающей стороной.* Поскольку приемная и передающая стороны находятся на разных оконечных системах, возможно, разделенных тысячами километров, единственный способ уведомить передающую сторону о результате передачи пакетов заключается в организации обратной связи, исходящей от принимающей стороны. Обратная связь, как и в примере с телефонным сообщением, заключается в посылке положительных (ACK) или отрицательных (NAK) квитанций. Минимальная длина квитанции составляет 1 бит, поскольку двух различных значений (0 и 1) достаточно, чтобы указать исход передачи.
- *Повторная передача.* Пакет, при передаче которого были зафиксированы ошибки, подлежит повторной передаче передающей стороной.

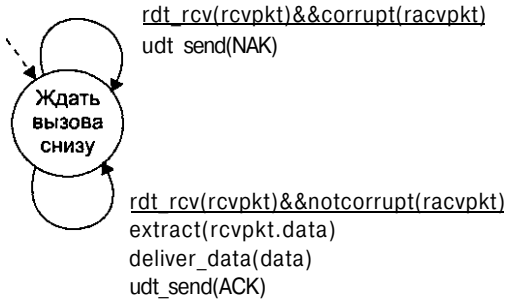
На рис. 3.9 изображены схемы конечных автоматов для протокола rdt 2.0, осуществляющего обнаружение ошибок, а также передачу положительных и отрицательных квитанций.

Автомат передающей стороны имеет два состояния. Состояние *а* соответствует ожиданию передачи данных от верхнего уровня. При наступлении события rdt_send(data) передающая сторона создает пакет sndpkt, включающий данные и поле контрольной суммы (например, вычисляемой по аналогии с протоколом UDP, как описано в предыдущем разделе), и отсылает его приемной стороне методом udt_send(sndpkt). Состояние *б* соответствует ожиданию квитанции от приемной стороны. В случае получения квитанции ACK, то есть наступления события rdt_rcv(rcvpkt) && isACK (rcvpkt), передающая сторона переходит в состояние ожидания данных от верхнего уровня. Если квитанция оказывается отрицательной (NAK), происходит повторная переда-

ча пакета и ожидание ее результата (квитанции). Важной особенностью передающей стороны является то, что, находясь в состоянии ожидания квитанции, она не может принимать данные от верхнего уровня; прием новой порции данных возможен только после получения положительной квитанции для текущего пакета. Таким образом, безошибочная передача предыдущего пакета является необходимым условием для начала передачи следующего пакета. Протоколы, функционирующие подобным образом, называют *протоколами с ожиданием подтверждений*.



а



б

Рис. 3.9. Протокол rdt 2.0 — передача данных, допускающая искажения битов: а — передающая сторона; *б* — принимающая сторона

Принимающая сторона **rdt** 2.0 по-прежнему описывается единственным состоянием. При получении пакета генерируется квитанция ACK или NAK в зависимости от того, содержит принятый пакет ошибки или нет. Приему пакета, содержащего ошибки, соответствует событие **rdt_rcv(rcvpkt) && corrupt(rcvpkt).**

Со стороны протокол **rdt** 2.0 может выглядеть работоспособным, однако ему присущ «врожденный» порок, заключающийся в незащищенности квитанций от воз-

можных искажений (перед тем как двигаться дальше, обдумайте возможные пути решения этой проблемы). Этот порок, к сожалению, гораздо серьезней, чем кажется на первый взгляд. Для его устранения нам необходимо, как минимум, включить в квитанцию поле контрольной суммы. Нетривиальным также является решение вопроса о том, каким образом протокол должен действовать при наличии ошибок в квитанциях, поскольку принимающая сторона в этом случае не получает никакой информации о результатах передачи последнего пакета.

Ниже приведены три возможных варианта организации работы протокола с квитанциями, содержащими ошибки.

- Вернемся к примеру с телефонным сообщением и представим себе, как два человека могут решить подобную проблему. Если передающий абонент не расслышал фразу «Да!» или «Повтори это еще раз!», он может обратиться к принимающему абоненту с фразой «Что ты сказал?», являющейся новым типом пакета в протоколе. С другой стороны, как поступить, если и она не будет расслышана? Не понимая, является ли эта фраза новым предложением, принимающий, вероятно, ответит фразой «Что ты сказал?»; в свою очередь, она также может быть не расслышана. Очевидно, что описанный способ решения проблемы является весьма громоздким и ненадежным.
- Можно добавить в квитанции некоторое количество контрольных битов, достаточное не только для обнаружения, но и для исправления ошибок. Этот способ решает поставленную проблему в случае, если канал только искажает данные, но не теряет их.
- Можно выполнить обычную повторную передачу пакета, приравняв поврежденные квитанции к отрицательным. Такой подход приводит к *дублированию пакетов*. Основная проблема с дублированием пакетов заключается в том, что принимающая сторона не может определить, положительная или отрицательная квитанция на получение предыдущего пакета была отправлена ей в ответ. Следовательно, она также не может определить, является ли последний пакет новым или имела место повторная передача.

Нехитрое решение приведенной задачи, используемое во многих современных протоколах, включая TCP, состоит в добавлении в пакет данных нового поля, содержащего *порядковый номер* пакета. Порядковый номер формируется передающей стороной, осуществляющей подсчет передаваемых пакетов. Для того чтобы определить, является ли последний принятый пакет новым, принимающей стороне достаточно лишь проанализировать значение порядкового номера. В случае нашего простого протокола роль порядкового номера может играть единственный бит, сохраняющий значение для повторно посылаемого пакета и изменяющий значение на противоположное при передаче нового пакета. Поскольку мы создаем протокол в предположении о том, что потеря данных в канале невозможна, включать в квитанции порядковый номер пакета, к которому они относятся, нет необходимости. Передающая сторона всегда получает квитанцию, соответствующую последнему переданному пакету.

На рис. 3.10 и 3.11 приведены схемы конечных автоматов для протокола rdt 2.1, являющегося исправленной версией rdt 2.0. Оба автомата теперь имеют вдвое больше

состояний по сравнению с протоколом rdt 2.0. Это объясняется тем, что необходимо различать состояния протокола, соответствующие двум возможным порядковым номерам (0 и 1) принимаемого/передаваемого пакета. Обратите внимание на то, что действия при приеме/передаче пакета с порядковым номером 0 являются зеркальным отображением действий при приеме/передаче пакета с порядковым номером 1; единственное различие заключается в обработке порядкового номера.

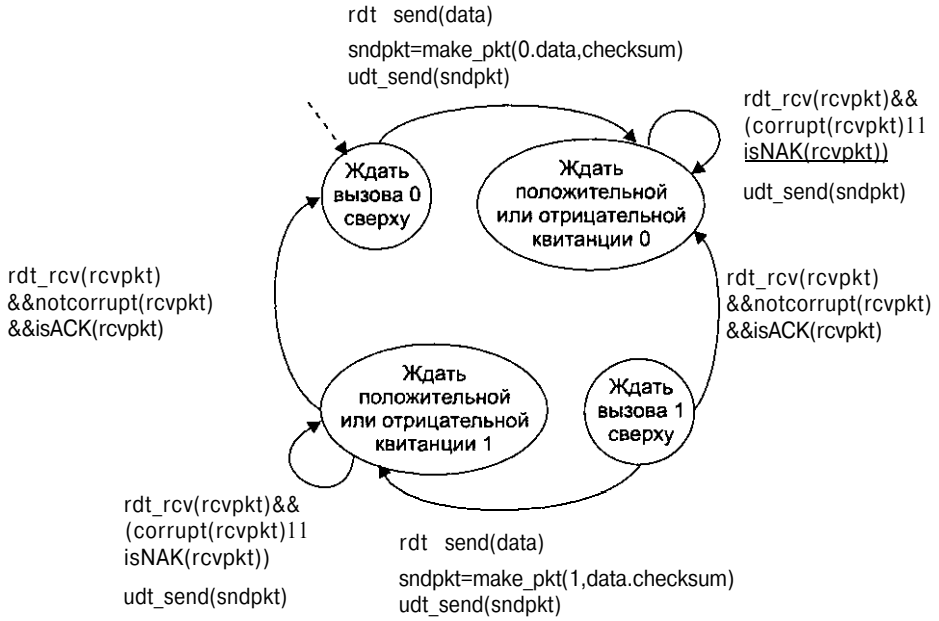


Рис. 3.10. Передающая сторона протокола rdt 2.1

В протоколе rdt 2.1 используются оба вида квитанций (положительные и отрицательные). В случае, когда порядковый номер принятого пакета отличается от ожидаемого, генерируется ACK; если пакет содержит искаженные данные, то генерируется NAK. Мы можем добиться эффекта NAK, если перешлем ACK для последнего принятого пакета. Если передающая сторона получит две положительные квитанции для одного и того же пакета (то есть произойдет удвоение положительных квитанций), то это будет указывать на наличие ошибок в пакете, следующем за тем, для которого были получены сдвоенные квитанции. Модель протокола rdt 2.2, осуществляющего надежную передачу по каналу, допускающему искажения битов без отрицательного квитирования, приведена на рис. 3.12 и 3.13. Единственное различие между протоколами rdt 2.1 и rdt 2.2 заключается в том, что в rdt 2.2 принимающая сторона должна включать в ACK порядковый номер пакета (это достигается путем передачи аргументов ACK,0 и ACK,1 процедуре make_pkt(); см. схему автомата принимающей стороны), а передающей стороне, в свою очередь, необходимо анализировать порядковый номер пакета, включенный в ACK (путем включения дополнительного аргумента 0 или 1 в процедуру isACK(); см. схему автомата передающей стороны).

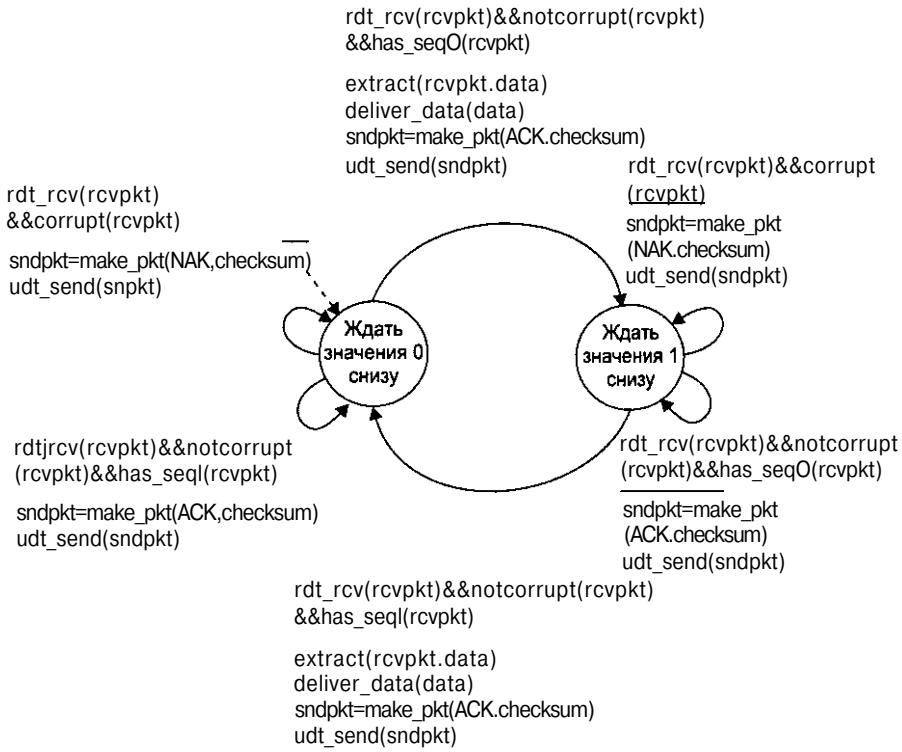


Рис. 3.11. Принимающая сторона протокола rdt 2.1

Надежная передача данных по каналу, допускающему искажение битов и потерю пакетов

Теперь предположим, что нам необходимо обеспечить передачу данных по каналу, в котором возможны не только искажения, но и потери пакетов (такая ситуация вполне типична для современных компьютерных сетей, включая Интернет). При разработке протокола нам придется решить две дополнительные задачи: найти способ определения факта потери пакета и указать действия, предпринимаемые в этом случае. Последняя задача решается с помощью контрольных сумм, порядковых номеров, квитанций и повторных посылок — механизмов, реализованных ранее в протоколе rdt 2.2. Для решения первой задачи нам потребуется применение новых механизмов.

Существует множество подходов к решению проблемы потери пакетов (некоторые из которых будут рассмотрены как дополнительные в упражнениях, приведенных в конце главы). Сейчас нас будут интересовать определение факта потери пакетов, а также методы доставки потерянных пакетов принимающей стороне. Предположим, что при передаче пакета происходит потеря либо самого пакета, либо квитанции, сгенерированной для этого пакета принимающей стороной. В обоих случаях квитанция не будет получена передающей стороной. Передающая сторона может продолжать ожидание в течение какого-либо промежутка времени, по окончании которого посчитает пакет потерянным и выполнит его повторную передачу.

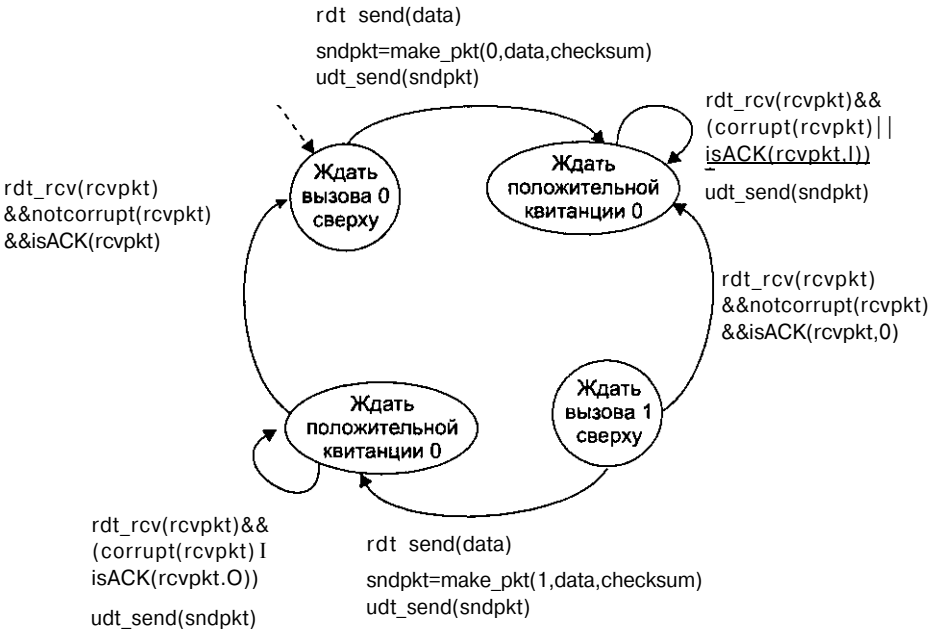


Рис. 3.12. Передающая сторона протокола rdt 2.2

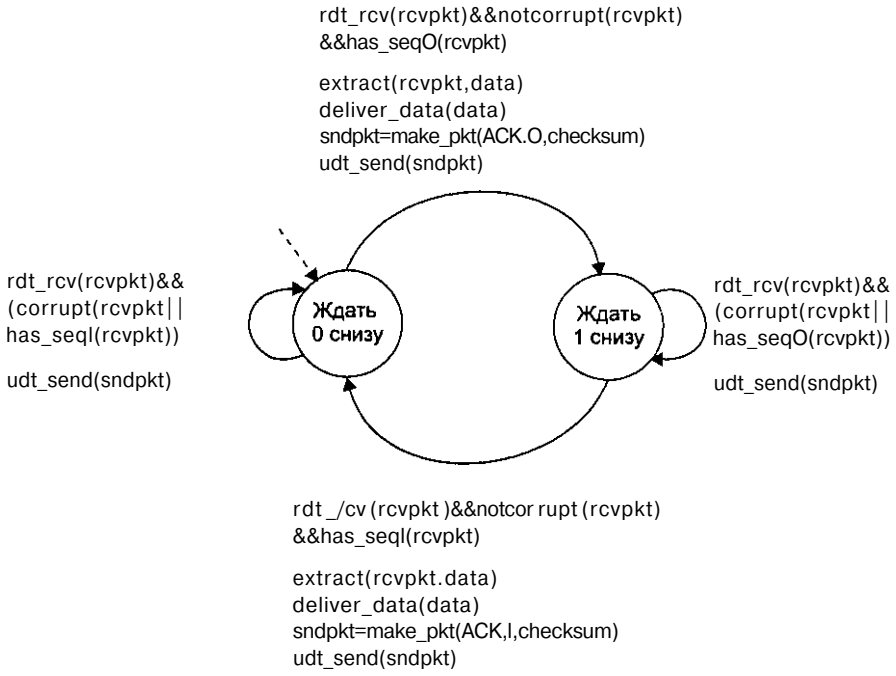


Рис. 3.13. Принимающая сторона протокола rdt 2.2

Возникает вопрос о том, насколько долгим должно быть ожидание. Очевидно, что время ожидания складывается из времени оборота между передающей и принимающей сторонами (возможно, включая промежуточную буферизацию в маршрутизаторах) и времени обработки пакета принимающей стороной. В большинстве компьютерных сетей оценка максимума времени ожидания, особенно с высокой точностью, весьма трудоемка. Кроме того, желательно разрешить проблему передачи потерянного пакета за короткое время, а ожидание в течение максимального времени получения квитанции оказывается слишком долгим. На практике интервал ожидания делают более коротким, выбирая его из предположения, что вероятность потери пакета весьма велика (хотя и не абсолютна). По истечении этого интервала происходит повторная передача пакета. Поскольку существует ненулевая вероятность получения квитанции для пакета после его повторной передачи, становится возможным *дублирование пакетов*. Эта проблема уже решена нами в протоколе rdt 2.2 с помощью порядковых номеров.

Приведенный механизм является панацеей для передающей стороны: ей не нужно знать о том, потерян ли пакет, потеряна ли его квитанция или ничего не потеряно, но квитанция пришла с задержкой. Во всех трех случаях производится одно и то же действие: повторная передача пакета. Для контролирования времени в данном механизме используется *таймер отсчета,* который позволяет определить окончание интервала ожидания. Передающей стороне необходимо запускать таймер каждый раз при передаче пакета (как при первой, так и при повторной), обрабатывать прерывания от таймера и останавливать его.

Вероятность потерь пакетов и квитанций, а также дублирования пакетов усложняет процесс обработки квитанций передающей стороной. Например, необходимо выяснять, относится получаемая квитанция к последнему пакету или к одному из ранее переданных. Для решения этой проблемы в квитанцию включается специальное *поле квитанции*. При генерации квитанции приемная сторона записывает в поле квитанции порядковый номер соответствующего пакета. Таким образом, передающая сторона может определить, к какому пакету относится принятая квитанция.

На рис. 3.14 приведена схема конечных автоматов протокола rdt 3.0, осуществляющего надежную передачу данных по каналу, допускающему искажение и потерю пакетов, а на рис. 3.15 — схема функционирования протокола для четырех возможных случаев: передача происходит без потерь и задержек; происходит потеря пакета; происходит потеря квитанции для пакета; истекает интервал ожидания квитанции. Для каждого случая время показано пунктирной стрелкой, направленной сверху вниз. Обратите внимание на то, что момент получения пакета принимающей стороной наступает позже момента передачи пакета передающей стороной из-за задержек в распространении сигнала в сети. На рис. 3.15, *б, в* и *г* квадратные скобки обозначают моменты запуска и переполнения таймера. Некоторые тонкости функционирования протокола рассмотрены в упражнениях в конце этой главы. Поскольку порядковые номера пакетов представляют собой меняющиеся значения 0 и 1, протокол rdt 3.0 называется *протоколом с чередованием битов*.

Итак, мы ознакомились с ключевыми понятиями протоколов надежной передачи данных: контрольными суммами, порядковыми номерами пакетов, таймерами и по-

ложительными и отрицательными квитанциями. Созданный нами протокол оказывается вполне работоспособным!

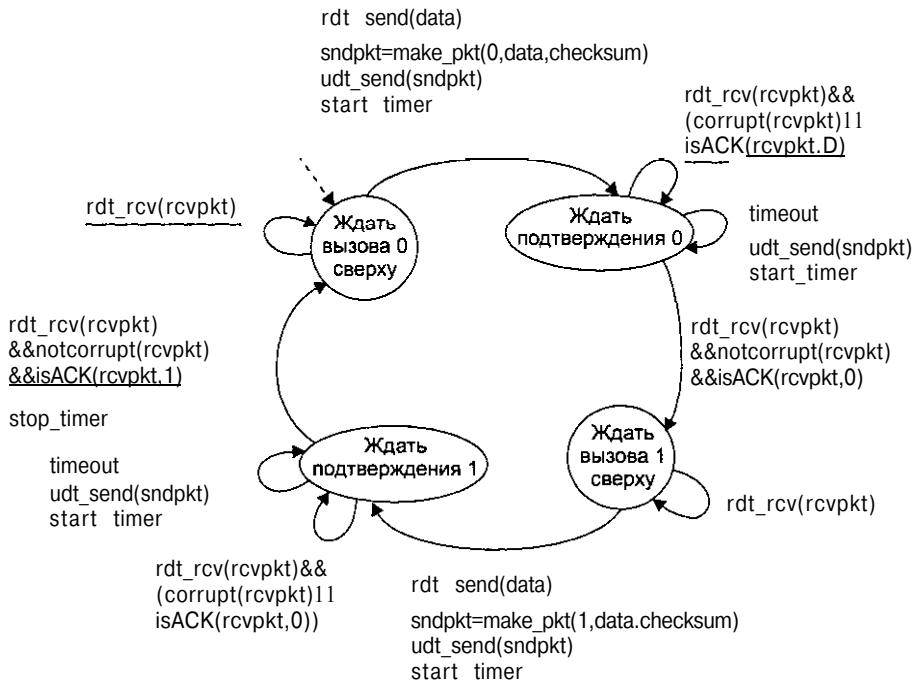


Рис. 3.14. Передающая сторона протокола rdt 3.0

Протоколы надежной передачи данных с конвейеризацией

Протокол rdt 3.0 является корректным с точки зрения функционирования, однако вряд ли нашлось бы много пользователей, которых бы устроило качество обслуживания этого протокола, особенно в современных высокоскоростных компьютерных сетях. Главной проблемой протокола rdt 3.0 является то, что он относится к протоколам с ожиданием подтверждений.

Для того чтобы лучше понять последствия ожидания, представим себе следующую ситуацию. Пусть имеется пара хостов, как показано на рис. 3.16, один из которых находится в Санкт-Петербурге, а другой в Новосибирске.

Время оборота между хостами при условии, что распространение сигнала происходит со скоростью света, составляет приблизительно 30 мс. Предположим, что хосты соединены между собой каналом связи со скоростью передачи $R = 1$ Гбит/с = $= 10^9$ бит/с. Если размер пакета, включая заголовок и данные, $L =$ 1000 байт = = 8000 бит, то время фактической передачи пакета по каналу:

$$*_{1Icp} = L/R = (8000 \text{ бит/пакет}) / (10^9 \text{ бит/с}) = 8 \text{ мкс}.$$

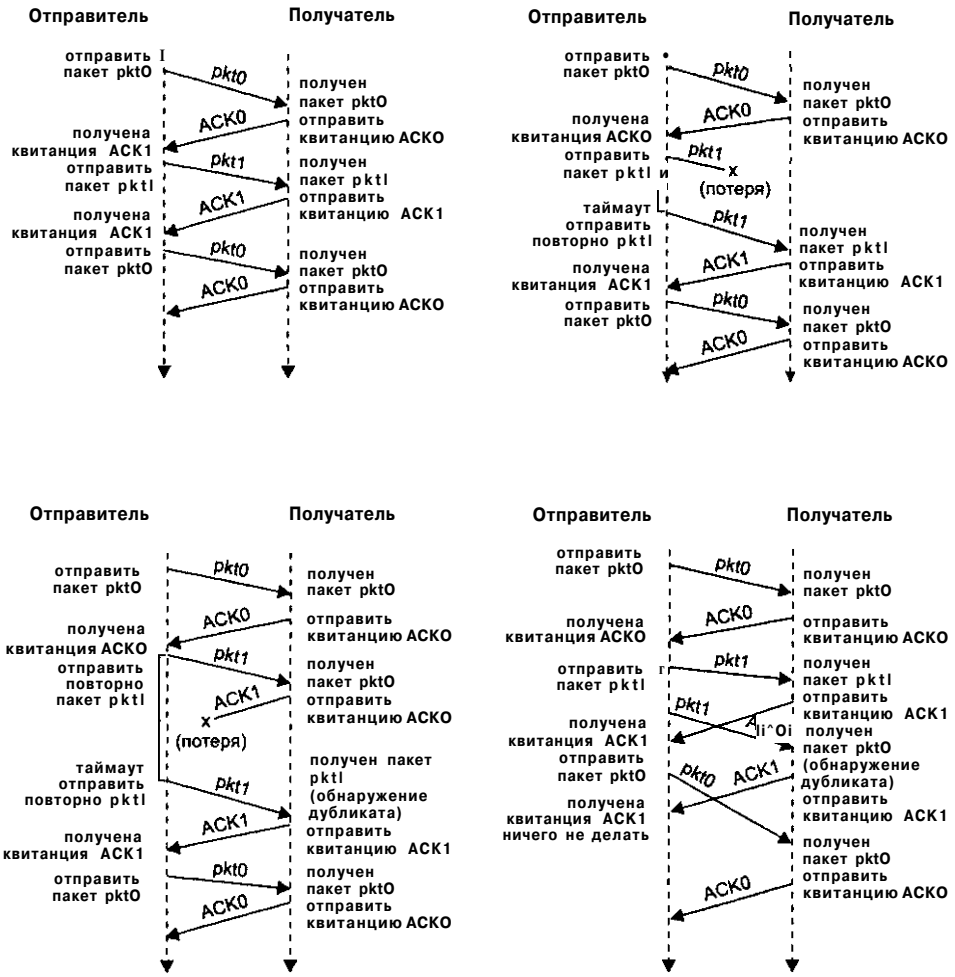
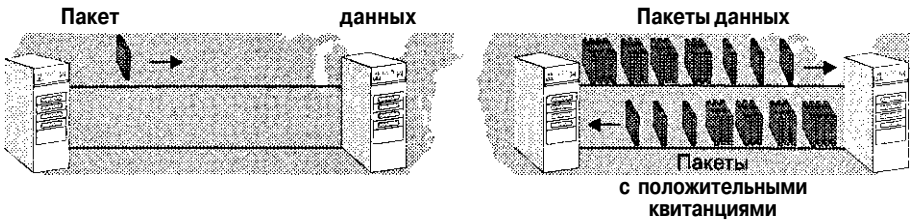


Рис. 3.15. Схема работы протокола rdt 3.0: а — потери пакетов отсутствуют; б — потеря пакета; *в* — потеря подтверждения; г — истек интервал ожидания

Обратимся к рис. 3.17, *а*. Если с помощью нашего протокола передающая сторона начнет передачу пакетов в момент времени $t = 0$, то при $t = L/R = 8$ мкс последний бит окажется в канале. Распространение пакета в канале займет 15 мс; таким образом, последний бит пакета достигнет принимающей стороны в момент времени $t = \text{RTT}/2 + L/R = 15{,}008$ мс. Считая размеры квитанций достаточно малыми для того, чтобы временем их передачи можно было пренебречь, и предполагая, что начало отправки квитанции совпадает с окончанием приема пакета, получаем, что квитанция достигает передающей стороны в момент времени $t = \text{RTT} + L/R = 30{,}008$ мс. С этого момента передающая сторона может начинать передачу следующего пакета. Таким образом, оказывается, что в каждом периоде длительно-

стью 30,008 мс передача пакета занимает лишь 0,008 мс. Если ввести коэффициент *полезности* как долю времени, в течение которого ведется передача пакетов, то полезность передающей стороны

$$u_{перед} = (L/R) / (RTT + L/R) = 0{,}008/30{,}008 = 0{,}00027.$$



а *б*

Рис. 3.16. Сравнение протокола с ожиданием подтверждения и протокола с конвейеризацией: *а* — работа протокола с ожиданием подтверждений; б — работа протокола с конвейеризацией

Другими словами, передающая сторона активна в течение лишь 0,0027 % общего времени. Кроме того, наши расчеты говорят о том, что передача данных ведется со скоростью 1000 байт за 30,008 мс, что составляет 267 Кбит/с — величину, в тысячи раз меньшую скорости, обеспечиваемой каналом связи. Представьте, каким разочарованием для администратора сети было бы закупить дорогостоящую линию связи и получить столь низкое качество обслуживания! Этот пример наглядно демонстрирует, каким образом сетевые протоколы могут «тормозить» передачу, обеспечиваемую аппаратно. Обратите внимание на то, что мы не учли время, затрачиваемое на обработку данных протоколами более низких уровней передающей и принимающей сторон, а также задержки обработки и ожидания, которые могли иметь место при передаче пакетов. Включение этих факторов в модель показало бы еще большую несостоятельность нашего протокола с точки зрения полезности.

Устранение описанного недостатка производится весьма несложным способом: вместо отправки каждого нового пакета после получения квитанции для предыдущего осуществляется передача группы пакетов без ожидания квитанций, как показано на рис. 3.16, *б*. На рис. 3.17, б видно, что передача трех пакетов подряд приводит к трехкратному повышению полезности по сравнению с нашим протоколом. Поскольку пакеты, передающиеся группами, удобно изображать в виде конвейера, данный способ передачи получил название передачи с *конвейеризацией*. Применение конвейеризации в протоколах надежной передачи данных приводит к следующим последствиям.

- Увеличивается диапазон порядковых номеров, поскольку все передаваемые пакеты (за исключением повторных передач) должны быть однозначно идентифицируемы.

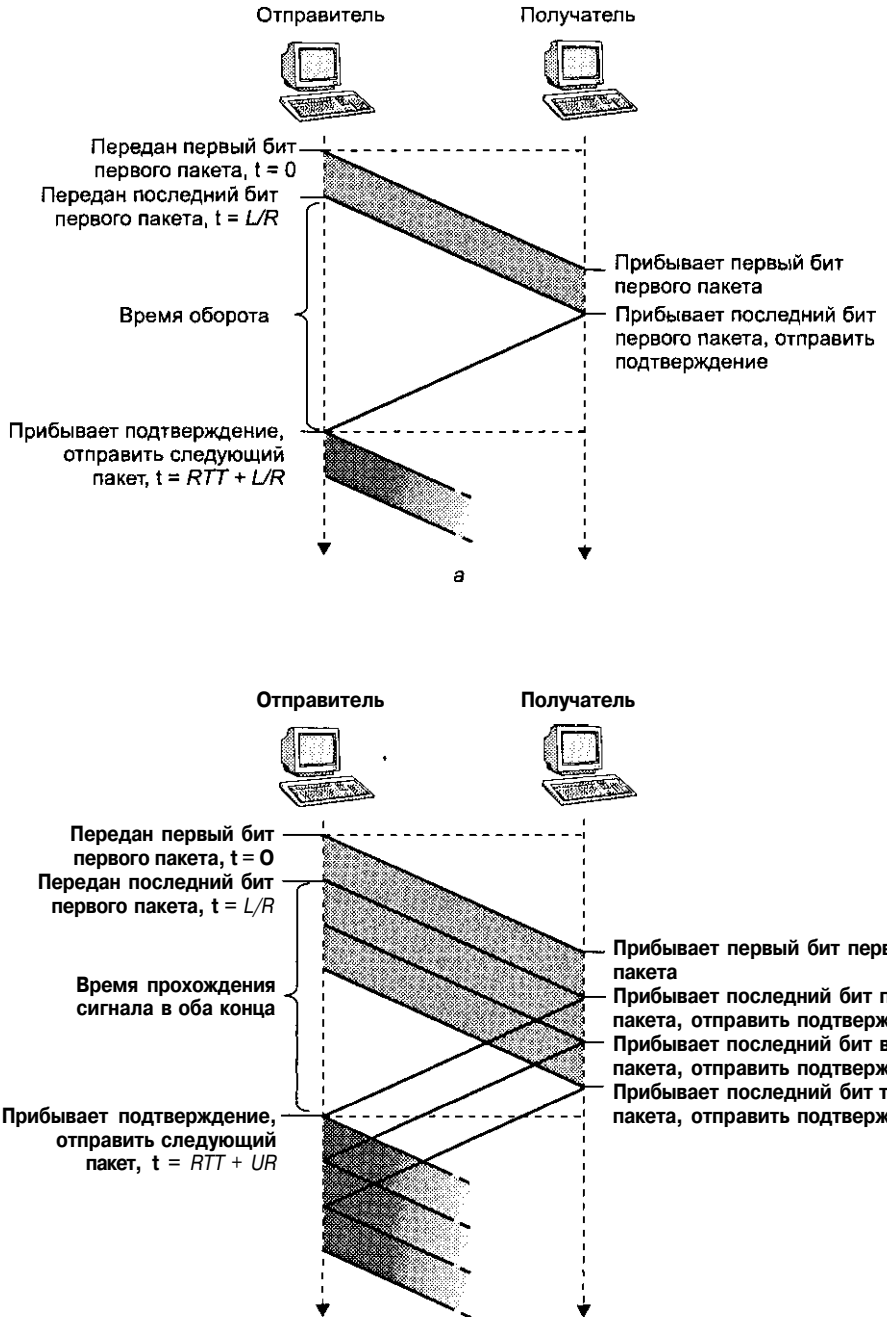


Рис. 3.17. Передача с ожиданием подтверждения и конвейеризаци.ей: а — работа протокола с ожиданием подтверждений; *б* — работа протокола с конвейеризацией

- Появляется необходимость в увеличении буферов на передающей и принимающей сторонах. Так, размер буфера передающей стороны должен быть достаточным для того, чтобы хранить все переданные пакеты, для которых ожидается получение квитанций. На принимающей стороне может возникнуть необходимость в буферизации успешно принятых пакетов, о чем будет сказано далее.

Диапазон порядковых номеров и требования к размерам буферов зависят от действий, предпринимаемых протоколом в ответ на искажение, потерю и задержку пакета. В случае конвейеризации существуют два метода исправления ошибок: возвращение на N пакетов назад и выборочное повторение.

Возвращение на N пакетов назад

В протоколах, использующих метод *возвращения на N шагов назад* (Go-Back-N, GBN), передающая сторона может посылать последовательности пакетов, не ожидая квитанций, при этом максимальная длина последовательности ограничена некоторым числом *N*. На рис. 3.18 представлена схема определения диапазона порядковых номеров. Если base — порядковый номер самого «старого» неподтвержденного пакета, а nextseqnum — наименьший из «свободных» порядковых номеров (то есть номер, который будет присвоен следующему посылаемому пакету), то полный диапазон порядковых номеров можно разделить на четыре части. Номера в диапазоне от 0 до base-1 назначены переданным ранее пакетам, для которых уже получены квитанции. Номера от base до nextseqnum-1 также относятся к переданным пакетам, однако для этих пакетов квитанции еще не получены. Номера от nextseqnum до base+N-1 могут быть назначены пакетам, которые необходимо сформировать и отправить сразу при поступлении новых данных от прикладного уровня. Номера, превышающие значение base+N, нельзя использовать до тех пор, пока не будет получена квитанция для текущего пакета, то есть пакета с порядковым номером base.

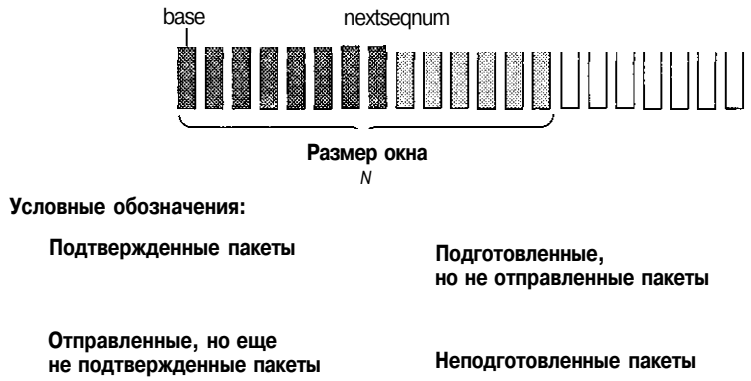


Рис. 3.18. Диапазон порядковых номеров с точки зрения передающей стороны

Как показано на рисунке, диапазон порядковых номеров для пакетов, которые могут быть переданы без ожидания квитанций, можно рассматривать в виде «окна» раз-

мером *N,* лежащего внутри множества порядковых номеров. В процессе выполнения протокола это окно «движется» в сторону увеличения порядковых номеров. Такое представление определило терминологию, в соответствии с которой значение *N* называют *размером окна,* а протокол GBN — *протоколом скользящего окна.* Вероятно, у вас возникает вопрос: для чего нужно ограничивать число пакетов, передаваемых без ожидания подтверждения, размером окна ЛГ? В следующем разделе этой главы мы увидим, что одной из причин является необходимость контролировать передаваемый поток данных. Другая причина будет раскрыта в разделе «Контроль перегрузок в TCP», где мы займемся изучением механизма контроля перегрузки, реализованного в протоколе TCP.

На практике порядковый номер пакета хранится в одном из полей заголовка, имеющем фиксированную длину. Если k — число битов этого поля, то диапазоном порядковых номеров *является* $[0, 2^k - 1]$. Поскольку множество порядковых номеров конечно, все арифметические операции с ними производятся по модулю 2^k (другими словами, множество порядковых номеров можно представить в виде кольца, в котором числом, следующим за $2^k - 1$, является 0). Возвращаясь к протоколу rdt 3.0, отметим, что разрядность его порядковых номеров равна 1, а диапазон — [0,1]. В конце этой главы перечислены несколько проблем, порождаемых ограниченностью множества порядковых номеров. Как мы увидим позже, в протоколе TCP порядковые номера имеют разрядность 32 и предназначены для подсчета байтов (вместо пакетов) в байтовом потоке.

На рис. 3.19 и 3.20 представлены расширенные модели конечных автоматов, описывающие передающую и принимающую стороны GBN-протокола с механизмом квитирования без использования отрицательных квитанций. Мы назвали эти модели расширенными, поскольку они включают в себя переменные величины base и nextseqnum (напоминающие переменные в языках программирования), а также различные операции и другие действия с участием этих переменных. Более того, спецификация расширенной модели конечных автоматов очень напоминает спецификацию языка программирования. В [32] вы можете найти замечательный обзор средств описания протоколов с помощью программно-ориентированных механизмов, в частности расширенных моделей конечных автоматов.

Передающая сторона GBN-протокола должна реагировать на три вида событий.

- *Вызов протоколом более высокого уровня.* Когда «сверху» вызывается функция rdt_send(), передающая сторона сначала производит проверку степени заполнения окна (то есть наличия *N* посланных сообщений, ожидающих получения квитанций). Если окно оказывается незаполненным, новый пакет формируется и передается, а значения переменных обновляются. В противном случае передающая сторона возвращает данные верхнему уровню, что является неявным указанием на то, что окно заполнено. Обычно верхний уровень предпринимает повторную попытку передачи данных через некоторое время. На практике передающая сторона чаще всего располагает либо буфером, в котором данные хранятся до появления возможности их пересылки, либо механизмом синхронизации (например, семафором или флагом), позволяющим вызывать функцию rdt_send() только при наличии незаполненного окна.

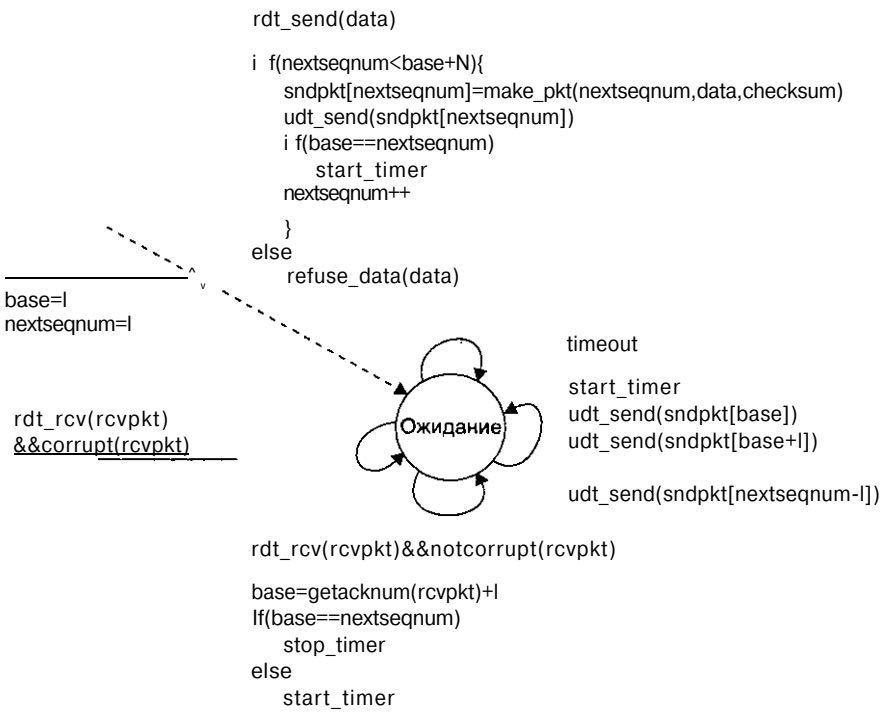


Рис. 3.19. Расширенная модель конечных автоматов передающей стороны GBN-протокола

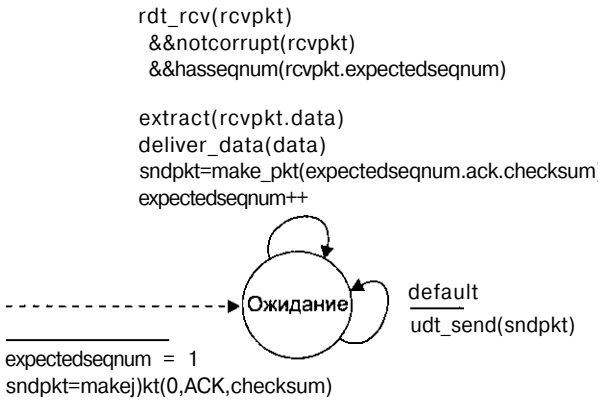


Рис. 3.20. Расширенная модель конечных автоматов принимающей стороны GBN-протокола

- *Получение подтверждения.* В нашем GBN-протоколе для пакета с порядковым номером *n* выдается *общая квитанция,* указывающая на то, что все пакеты с порядковыми номерами, предшествующими n^y успешно приняты. Мы вернемся к механизму квитирования при рассмотрении принимающей стороны GBN-протокола.

- *Истечение интервала ожидания.* Своим названием GBN-протокол обязан механизму реагирования на потери и задержки данных. Как и в случае протокола с ожиданием подтверждения, для определения фактов потерь и задержек пакетов и квитанций GBN-протокол использует таймер. В случае истечения интервала ожидания передающая сторона осуществляет повторную передачу всех посланных неподтвержденных пакетов. В нашем примере (см. рис. 3.19) передающая сторона использует единственный таймер, который отсчитывает время от момента передачи самого «старого» из пакетов, для которого не получено подтверждение. Если подтверждение получено, но при этом имеются переданные неподтвержденные пакеты, происходит сброс таймера. Отсутствие неподтвержденных пакетов приводит к остановке таймера.

Действия, предпринимаемые принимающей стороной GBN-протокола, также не отличаются сложностью. Если доставка пакета с порядковым номером *n* произошла корректно и без нарушения очередности (то есть последние данные, переданные верхнему уровню, относятся к пакету с порядковым номером *n*- 1), то принимающая сторона генерирует подтверждение и передает новые данные верхнему уровню. В противном случае принятый пакет игнорируется, а подтверждение генерируется для последнего корректно обработанного пакета. Поскольку пакеты передаются верхнему уровню по одному, передача пакета *k* означает, что все пакеты с порядковыми номерами, меньшими *k,* уже переданы верхнему уровню. Таким образом, применение механизма группового квитирования в GBN-протоколах является весьма естественным.

Наш GBN-протокол игнорирует пакеты, полученные с нарушением порядка следования. Несмотря на то что удаление пакета, не содержащего ошибок (а лишь нарушившего порядок следования данных), кажется нерациональным и неэкономным, существуют причины, обусловливающие это удаление. Мы знаем о том, что приемная сторона должна обеспечивать передачу данных верхнему уровню в правильном порядке. Предположим, что вместо пакета с порядковым номером *n* приемная сторона получает пакет с номером *n* + 1. Поскольку мгновенная доставка такого пакета верхнему уровню невозможна, необходимо организовать промежуточное хранение пакета *n* + 1 до момента, когда будет принят пакет *n,* а его данные переданы верхнему уровню. В случае потери пакета *n* в соответствии с GBN-протоколом организуется повторная передача как пакета *n,* так и пакета *n* + 1. Таким образом, принимающая сторона может не сохранять пакет *n* + 1. Кроме того, игнорирование пакетов, полученных с нарушением порядка следования, позволяет упростить буферизацию на принимающей стороне. Из сказанного следует, что передающая сторона оперирует тремя величинами: верхней и нижней границами окна, а также наименьшим свободным порядковым номером. Для принимающей стороны необходим лишь порядковый номер следующего по порядку пакета. Эта величина хранится в переменной expectedseqnum, присутствующей на схеме, показанной на рис. 3.20. Необходимо признать, что механизм удаления успешно принятых пакетов тоже «небезгрешен»: при повторной передаче существует вероятность новой потери пакета, что, в свою очередь, приводит к необходимости дополнительных повторных передач.

На рис. 3.21 показана схема функционирования GBN-протокола, в котором размер окна составляет 4 пакета. Осуществив передачу пакетов с порядковыми номерами от 0 до 3, передающая сторона должна получить подтверждение хотя бы для одного из них. Как только подтверждения будут получены (например, ACKO и ACK1), окно переместится «вперед», и передающая сторона сможет послать 2 новых пакета (pkt4 и pkt5). Когда принимающая сторона обнаруживает потерю пакета 2, пакеты 3, 4 и 5 считаются нарушившими порядок следования данных и игнорируются.

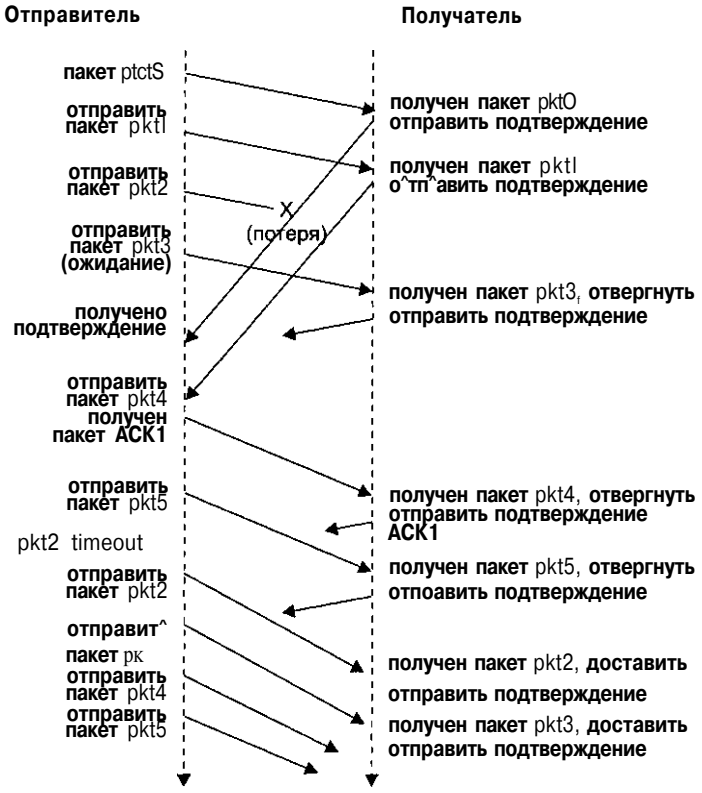


Рис. 3.21. Функционирование GBN-протокола

Отметим, что реализация GBN-протоколов в стеке протоколов, как правило, имеет структуру, схожую со структурой конечного автомата, представленного на рис. 3.19. Действия, выполняемые протоколом при наступлении различных событий, оформлены в виде процедур. В подобных *событийно-управляемых программах* выполнение процедур происходит либо при их вызове другими процедурами стека протоколов, либо при наступлении прерывания. На передающей стороне событиями, управляющими программой, являются вызов функции rdt_send() протоколом верхнего уровня, прерывание от таймера и вызов функции rdt_rcv() протоколом нижнего уровня при получении пакета.

Созданный нами GBN-протокол поддерживает почти все методы, использующиеся в протоколе надежной передачи данных TCP, который будет рассмотрен в следующем разделе. Мы увидим, что и TCP поддерживает порядковые номера, общие квитанции, контрольные суммы, интервалы ожидания и повторную передачу пакетов.

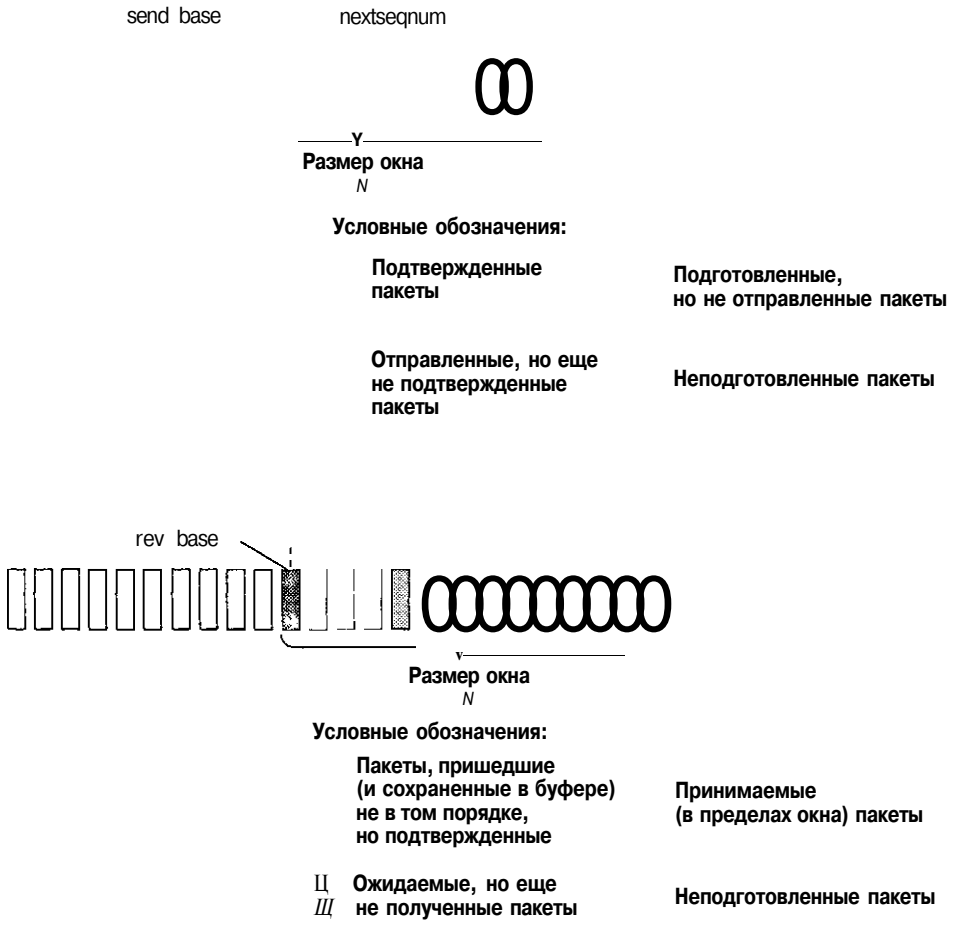
Выборочное повторение

Как показано на рис. 3.16, GBN-протокол, в отличие от протоколов с ожиданием подтверждения, позволяет передающей стороне «заполнять конвейер» пакетами, повышая чрезвычайно низкую загрузку линии связи. Тем не менее возможны ситуации, в которых эффективность GBN-протокола также оказывается весьма невысокой. Например, если размер окна и произведение пропускной способности на задержку распространения велики, конвейер может содержать слишком много пакетов. Наличие искажения хотя бы в одном из пакетов приводит к необходимости повторной передачи значительных объемов уже однажды переданных (причем без ошибок) данных. Чем выше вероятность искажений при передаче, тем большая часть издержек на передачу данных оказывается бесполезной. Возвращаясь к примеру с диктовкой сообщений, можно представить рассматриваемую ситуацию следующим образом: если «размер окна» составляет 1000 слов, то любое искаженное слово приводит к необходимости повторения всех 1000 слов «окна». Понятно, что такое «общение» отнимет у собеседников слишком много времени.

Как ясно из названия, метод *выборочного повторения* (Selective Repeat, SR) направлен на снижение количества повторных передач путем отправки только тех пакетов, которые были зафиксированы принимающей стороной как потерянные или искаженные. Это требует введения *отдельных* квитанций для каждого успешно принятого пакета. Как и в GBN-протоколе, число пакетов, находящихся в конвейере, ограничивается размером окна *N,* однако передающая сторона может получать квитанции для некоторых пакетов окна. На рис. 3.22 представлена схема определения диапазона порядковых номеров, а представленная ниже последовательность действий иллюстрирует работу передающей стороны SR-протокола.

1. *Получены данные от верхнего уровня.* Передающая сторона анализирует значение первого свободного порядкового номера. Если он принадлежит окну, то формируется пакет, содержащий переданные данные, и отсылается принимающей стороне. В противном случае передача данных откладывается, при этом данные либо помещаются в буфер, либо возвращаются верхнему уровню, как и в GBN-протоколах.
2. *Истек интервал ожидания.* Для обнаружения потерь пакетов снова приходится прибегать к использованию таймера. Каждый пакет должен быть снабжен собственным логическим таймером, поскольку необходимо, чтобы в интервале ожидания передавался только один пакет. Для организации множества логических таймеров достаточно иметь единственный аппаратный таймер [531].
3. *Получено подтверждение.* Передающая сторона помечает соответствующий пакет как принятый (при условии, что он принадлежит окну). Если оказывается, что порядковый номер пакета совпадает со значением send_base, база окна

сдвигается вперед к неподтвержденному пакету с наименьшим порядковым номером. Если после сдвига окна обнаруживаются пакеты, попавшие внутрь окна и еще не переданные, осуществляется их передача.



б

Рис. 3.22. Диапазон порядковых номеров с точки зрения передающей и принимающей сторон SR-протокола

Принимающая сторона выдает квитанцию каждому принятому правильному (не содержащему ошибок) пакету, независимо от того, нарушает он порядок следования или нет. Пакеты, нарушающие порядок следования, хранятся в буфере до тех пор, пока не будут получены предшествующие пакеты; после этого данные всех принятых пакетов доставляются верхнему уровню. Ниже перечислены действия, выполняемые принимающей стороной SR-протокола, а рис. 3.23 иллюстрирует реакцию принимающей стороны на ситуацию, когда при передаче происходит по-

теря пакета. Обратите внимание на то, что принимающая сторона помещает пакеты 3, 4 и 5 в буфер, где они хранятся до тех пор, пока не будет получен пакет 2; после этого содержимое пакетов 2, 3, 4 и 5 доставляется верхнему уровню.

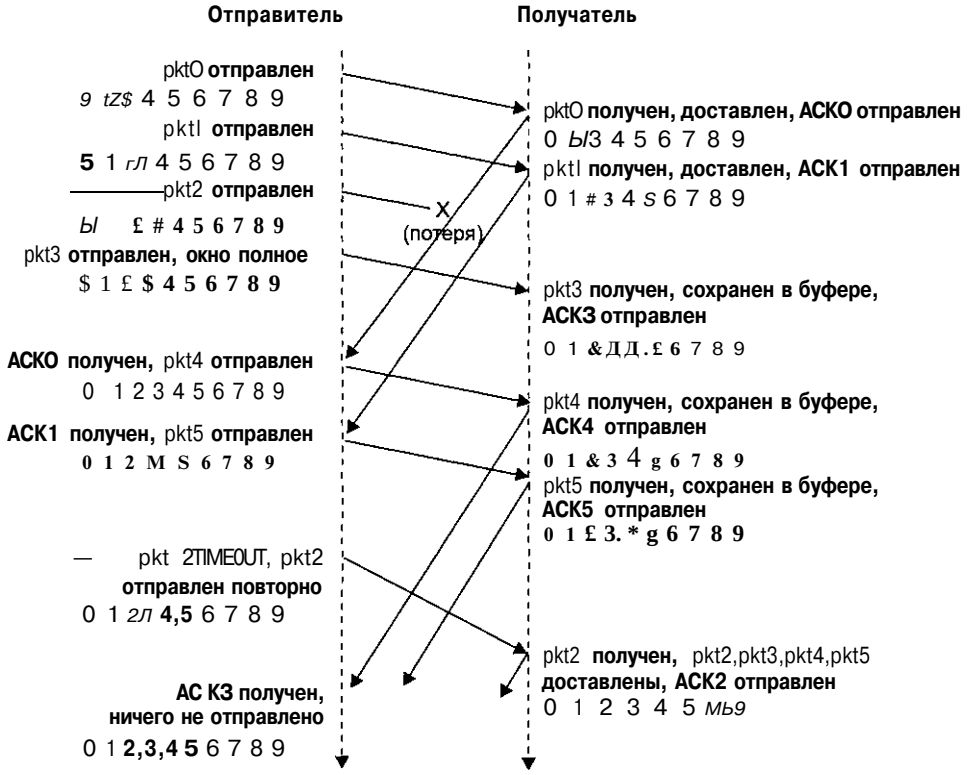


Рис. 3.23. Функционирование SR-протокола

1. *Успешно принят пакет с номером, лежащим в диапазоне [rcv_base,rcv_base* + N - 1]. В этом случае принятый пакет принадлежит окну, и принимающая сторона генерирует подтверждение для этого пакета. Если прием пакета осуществляется впервые, он заносится в буфер. В случае совпадения порядкового номера пакета с базой окна (rcv_base на рис. 3.22) этот пакет вместе со всеми пакетами, хранящимися в буфере, образует последовательность с наименьшим номером rcv_base. Данные, извлеченные из последовательности, передаются верхнему уровню; затем происходит сдвиг окна передающей стороны на длину последовательности. Подобная ситуация изображена на рис. 3.23: при получении пакета 2 верхнему уровню передаются данные пакетов 2, 3, 4 и 5.
2. *Принят пакет с номером, лежащим в диапазоне* [rev_base - N,rcv_base - 1]. Несмотря на то что квитанция для этого пакета уже была послана передающей стороне ранее, в этом случае предусматривается повторное квитирование пакета.
3. *Иначе пакет игнорируется.*

Действие 2 представленной процедуры демонстрирует важную особенность SR-протокола: принимающая сторона повторно квитирует пакеты с порядковыми номерами, лежащими ниже текущего базового номера окна. Для чего нужно повторное квитирование? Обратимся к рис. 3.22. При указанных множествах порядковых номеров принимающей и передающей сторон отсутствие подтверждения для пакета send_base приведет к его повторной передаче, несмотря на его успешный прием (очевидный для нас, но не для передающей стороны). Если подтверждение не будет сгенерировано, окно передающей стороны не сможет переместиться вперед! Эта ситуация указывает на важное свойство SR-протоколов (и не только): принимающая и передающая стороны не всегда имеют одинаковое представление о том, какие данные переданы корректно, а какие — нет. В случае SR-протоколов различие состоит в несовпадении окон обменивающихся сторон.

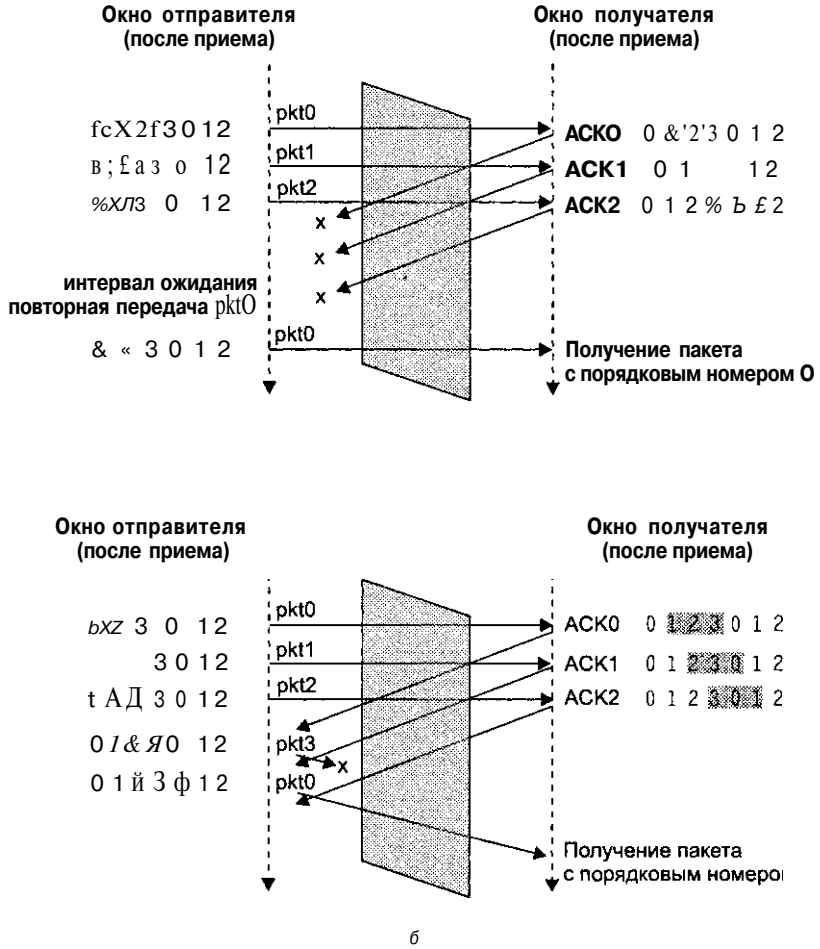
Отсутствие синхронизации между окнами передающей и принимающей сторон имеет важные последствия, когда мы сталкиваемся с ограниченностью диапазона порядковых номеров. Представим себе, что у нас имеется 4 порядковых номера (0, 1, 2 и 3), при этом размер окна равен 3. Пусть пакеты с порядковыми номерами 0, 1 и 2 были успешно получены и квитированы принимающей стороной. В этом случае окно принимающей стороны должно переместиться на 3 пакета вперед, то есть охватить пакеты с номерами 3, 0 и 1. Далее рассмотрим две ситуации. Первая ситуация, представленная на рис. 3.24, я, заключается в потере квитанции для переданных пакетов, что приводит к необходимости их повторной передачи. Таким образом, следующий пакет, получаемый принимающей стороной, будет иметь порядковый номер 0 и являться копией пакета, переданного ранее.

Во второй ситуации, представленной на рис. 3.24, б, квитанция для трех переданных пакетов успешно доставляется передающей стороне. Передающая сторона сдвигает свое окно на 3 позиции и осуществляет передачу пакетов с порядковыми номерами 3, 0 и 1; при этом пакет с номером 3 теряется, а пакет 0 доставляется на принимающую сторону. Заметим, что в пакете с номером 0 содержатся новые, не переданные ранее данные.

Теперь рассмотрим ту же ситуацию с точки зрения принимающей стороны. Действия, выполняемые передающей стороной, скрыты от нее; принимающая сторона способна лишь следить за последовательностями получаемых пакетов и генерируемых квитанций. Подобная ограниченность приводит к тому, что обе описанные выше ситуации воспринимаются принимающей стороной как *одинаковые.* Принимающая сторона не может отличить исходную передачу первого пакета от его повторной передачи. Очевидно, что протокол, размер окна которого на единицу меньше диапазона порядковых номеров, не является работоспособным. Однако насколько малым должен быть размер окна? В одном из упражнений, приведенных в конце этой главы, вам предлагается самостоятельно доказать, что размер окна SR-протокола не должен превосходить половины диапазона порядковых номеров.

Итак, мы завершаем разговор о протоколах надежной передачи данных. Мы рассмотрели значительный объем материала и ознакомились с множеством механиз-

мов, используемых в этих протоколах, — итог накопленным знаниям подводит табл. 3.2. Просмотрев этот раздел сначала, вы сможете увидеть, как различные механизмы постепенно включались в создаваемый протокол передачи данных, делая его все более и более реалистичным и улучшая качество его функционирования.



б

Рис. 3.24. Слишком большие окна на принимающей стороне SR-протокола: новый пакет или повторная передача?

Напоследок особо отметим еще одно допущение, лежащее в основе рассмотренной модели канала передачи данных. Оно состоит в том, что во время передачи пакетов не может произойти нарушение порядка их следования. В случае, если канал представляет собой физическую линию связи, подобное допущение выглядит естественным, однако при передаче по разветвленной компьютерной сети изменение порядка следования пакетов вполне возможно. Одно из прояв-

лений такого изменения заключается в том, что пакеты и квитанции с порядковым номером *x* могут появиться на принимающей и передающей сторонах в те моменты времени, когда номер *x* уже не принадлежит текущему окну. Фактически процесс передачи по каналу в этом случае можно представить как буферизацию пакетов и отбрасывание их из буфера в случайные моменты времени. Поскольку одни и те же порядковые номера могут неоднократно использоваться при передаче, необходимо следить за возможным дублированием пакетов. Механизмы слежения обеспечивают уверенность передающей стороны в том, что при использовании любого из порядковых номеров в сети не будет ни одного пакета с таким же порядковым номером. Обычно такая уверенность достигается путем введения понятия *максимального времени жизни* пакета. Например, согласно документу RFC 1323, в протоколе TCP максимальное время жизни пакета для высокоскоростных сетей приблизительно равно 3 мин. В [497] описан механизм, полностью решающий проблему изменения порядка следования пакетов.

Таблица 3.2. Механизмы, обеспечивающие надежную передачу данных и их использование

Механизм	Применение, комментарии
Контрольная сумма	Обнаружение искажений битов в принятом пакете
Таймер	Отсчет интервала ожидания и указание на его истечение. Последнее означает, что с высокой степенью вероятности пакет или его квитанция потеряны при передаче. В случае, если пакет доставляется с задержкой, но не теряется (преждевременное истечение интервала ожидания), либо происходит потеря квитанции, повторная передача приводит к дублированию пакета на принимающей стороне
Порядковые номера	Последовательная нумерация пакетов, посылаемых передающей стороной. «Пробелы» в номерах получаемых пакетов позволяют сделать заключение о потерях данных. Одинаковые порядковые номера пакетов означают, что пакеты дублируют друг друга
Положительная квитанция (подтверждение)	Генерируется принимающей стороной и указывает передающей стороне на то, что соответствующий пакет или группа пакетов успешно приняты. Обычно подтверждение содержит порядковые номера успешно принятых пакетов. В зависимости от протокола различают индивидуальные и групповые подтверждения
Отрицательная квитанция	Генерируется принимающей стороной и указывает передающей стороне на то, что соответствующий пакет не был успешно принят. Обычно отрицательная квитанция содержит порядковый номер пакета, при передаче которого произошли ошибки
Окно,конвейер	Ограничивают диапазон порядковых номеров, которые могут использоваться для передачи пакетов. Групповая передача и квитирование позволяют значительно увеличить пропускную способность протоколов по сравнению с режимом ожидания подтверждений. Как мы увидим, размер окна может быть рассчитан на основе возможностей приема и буферизации принимающей стороны, а также уровня загрузки сети

Протокол TCP — передача с установлением соединения

Теперь, располагая знаниями об основополагающих принципах надежной передачи данных, мы можем перейти к рассмотрению протокола транспортного уровня TCP, обеспечивающего надежную передачу данных с установлением соединения и популярного в Интернете. Как мы увидим, в протоколе TCP используются многие из методов обеспечения надежной передачи данных, описанных в предыдущем разделе: обнаружение ошибок, повторные передачи пакетов, общие квитанции, таймеры и поля порядковых номеров в заголовках пакетов и квитанциях. Описание TCP содержится в документах RFC 793, RFC 1122, RFC 1323, RFC 2018 и RFC 2581.

TCP-соединение

Говорят, что протокол TCP осуществляет передачу с установлением логического соединения, поскольку перед началом обмена данными два процесса осуществляют «рукопожатие» — процедуру, заключающуюся в передаче друг другу специальных сегментов, предназначенных для определения параметров обмена данными. Частью процедуры установления TCP-соединения является инициализация переменных состояния (многие из которых будут рассмотрены в этом разделе и разделе «Контроль перегрузок в TCP»), связанных с TCP-соединением.

TCP-соединение не является «соединением» в том смысле, в каком этот термин употребляется в коммутируемых сетях. Оно также не является виртуальным каналом (см. главу 1), поскольку наблюдение за его состоянием ведется обеими сторонами. Поскольку протокол TCP выполняется на оконечных системах и не выполняется на промежуточных сетевых устройствах (маршрутизаторах и мостах), последние не отслеживают состояния TCP-соединения. Более того, маршрутизаторы не «знают» о существовании TCP-соединения: их работа выполняется на уровне дейтаграмм.

TCP-соединение обеспечивает *дуплексную* передачу данных. Если на двух хостах выполняются соответственно процессы A и B, то данные могут одновременно передаваться как от процесса A к процессу B, так и от процесса B к процессу A. TCP-соединение также называют соединением *точка-точка,* то есть соединением между единственным приемником и единственным передатчиком. Групповая рассылка, описанная в главе 4, позволяет передавать данные сразу множеству адресатов с помощью одной операции, однако такая рассылка не поддерживается протоколом TCP.

Теперь давайте подробнее рассмотрим процесс установления TCP-соединения. Предположим, что процесс, выполняющийся на одном хосте, желает установить соединение с процессом, выполняющимся на другом хосте. Вспомним о том, что процесс, инициирующий соединение, мы договорились называть клиентским, а процесс, с которым устанавливается соединение, — серверным. Сначала клиентский процесс сообщает транспортному уровню своего хоста о том, что необходимо установить соединение с серверным процессом. Обратившись к разделу «Программиро-

вание TCP-сокетов» в главе 2, можно увидеть, что программа-клиент на языке Java делает это при помощи следующей команды:

```
Socket clientSocket = new Socket("hostname", portNumber);
```

Здесь hostname — имя сервера, а portNumber — номер порта, идентифицирующий процесс внутри сервера. Затем транспортный уровень клиента начинает установление TCP-соединения с транспортным уровнем сервера. Эту процедуру мы детально рассмотрим в конце этого раздела, а пока нам достаточно знать, что клиент сначала посылает серверу специальный TCP-сегмент, сервер отвечает клиенту другим специальным сегментом, и, наконец, клиент посылает серверу третий специальный сегмент. Первые два сегмента не содержат данных прикладного уровня; третий сегмент может содержать их. Поскольку обмен сегментами входит в процедуру установления соединения, последнюю часто называют *тройным рукопожатием*.

После того как TCP-соединение установлено, прикладные процессы могут начинать обмен данными. Передача данных от клиента к серверу происходит следующим образом: клиент направляет поток своих данных в сокет («дверь» процесса), как описано в разделе «Программирование TCP-сокетов» главы 2. Через сокет данные попадают в протокол TCP, выполняющийся на стороне клиента. Как показано на рис. 3.25, TCP направляет эти данные в буфер передачи — один из буферов, создаваемых при выполнении тройного рукопожатия. Время от времени TCP извлекает данные из буфера передачи. Интересной особенностью спецификации TCP (RFC 793) является свобода в выборе моментов отправки данных, находящихся в буфере. Согласно спецификации, протокол TCP должен «передать эти данные в виде сегментов в любой подходящий для этого момент времени». Максимальный объем данных, который может быть извлечен из буфера и помещен в сегмент, ограничивается *максимальным размером сегмента* (Maximum Segment Size, MSS). Максимальный размер сегмента зависит от реализации протокола TCP (определяется операционной системой) и может быть сконфигурирован; наиболее часто используются значения 1500,536 и 512 байт (размер сегмента часто устанавливается так, чтобы избежать IP-фрагментации, которая будет рассмотрена позже в этой главе). Обратите внимание на то, что максимальный размер относится к данным приложения, содержащимся в сегменте, а не к сегменту вместе с заголовком (такая терминология является несколько запутанной, однако мы вынуждены придерживаться этой терминологии ввиду ее распространенности).

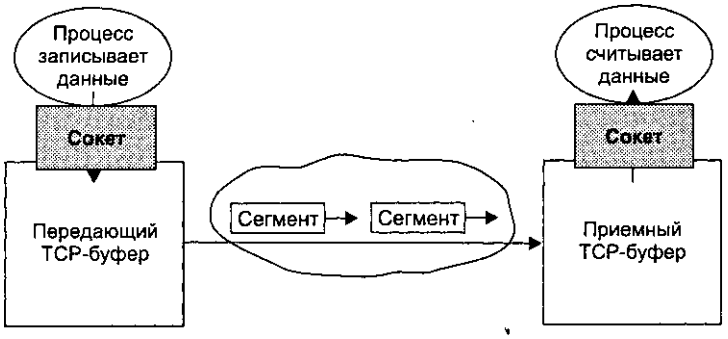


Рис. 3.25. Передающий и приемный буферы протокола TCP

Протокол TCP добавляет заголовок к каждому фрагменту данных, формируя *TCP-сегменты*. Сегменты передаются сетевому уровню, где заключаются в IP-дейтаграммы. Дейтаграммы пересылаются по сети и принимаются получателем. Когда сегмент оказывается на транспортном уровне, протокол TCP помещает данные сегмента в приемный буфер, как и показано на рисунке. Затем приложение считывает поток данных из буфера. Приемный и передающий буферы имеются на обеих сторонах соединения (мы рекомендуем читателю обратиться к апплету, который находится на нашем сайте http://www.awl.com/kurose-ross и позволяет просмотреть анимацию, иллюстрирующую работу приемного и передающего буферов).

Итак, TCP-соединение состоит из буферов и переменных на передающей и принимающей сторонах, а также сокетного соединения между сторонами. При этом для соединения не выделяется никаких буферов или переменных на промежуточных сетевых устройствах (маршрутизаторах, мостах и повторителях).

ИСТОРИЧЕСКАЯ СПРАВКА

В начале 1970-х годов стали получать распространение сети с коммутацией пакетов. Сеть APRAnet, предок Интернета, была одной из таких сетей. В каждой сети использовался собственный протокол. Двое исследователей, Уинтон Серф и Роберт Канн, понимая важность взаимодействия компьютерных сетей между собой, разработали межсетевой протокол TCP/IP (Transmission Control Protocol/Internet Protocol —- протокол управления передачей/Интернет-протокол). Несмотря на то что изначально TCP/IP представлял собой единую сущность, позже он был разделен на две части, TCP и IP, каждая из которых функционировала независимо. В мае 1974 года Серфом и Канном были выпущены публикации в IEEE Transaction on Communication Technology.

Протокол TCP/IP, составляющий основу современного Интернета, был разработан до появления персональных компьютеров и рабочих станций, технологий локальных сетей (Ethernet и др.), web, потокового аудио и чатов. Серф и Канн осознали необходимость в создании сетевого протокола, который бы обеспечивал, с одной стороны, мощную поддержку будущих сетевых приложений и, с другой стороны, взаимодействие протоколов более низких и более высоких уровней.

Структура TCP-сегмента

Рассмотрев понятие TCP-соединения, давайте обратимся к структуре TCP-сегмента. TCP-сегмент состоит из поля данных и нескольких полей заголовка. Поле данных содержит фрагмент данных, передаваемых между процессами. Как было показано ранее, размер поля данных ограничивается величиной MSS. Когда протокол осуществляет передачу большого файла (например, изображения, являющегося частью web-страницы), он, как правило, разбивает данные на фрагменты размером MSS (кроме последнего фрагмента, который обычно имеет меньший размер). Интерактивные приложения, напротив, часто обмениваются данными, объем которых значительно меньше MSS. Например, приложения удаленного доступа к сети, подобные Telnet, могут передать транспортному уровню 1 байт данных. Поскольку обычно длина заголовка TCP-сегмента составляет 20 байт (на 12 байт больше, чем в UDP), полный размер сегмента в этом случае равен 21 байт.

Рис. 3.26. Структура TCP-сегмента

На рис. 3.26 представлена структура TCP-сегмента. Как и в протоколе UDP, заголовок включает номера портов отправителя и получателя, предназначенные для процедур мультиплексирования и демультиплексирования данных, а также поле контрольной суммы. Кроме того, в состав TCP-сегмента входят еще некоторые поля.

- 32-разрядные поля *порядкового номера* и *номера подтверждения* необходимы для надежной передачи данных, о чем будет сказано ниже.
- 16-разрядное *окно приема* используется для управления потоком данных. Мы увидим, что оно содержит количество байтов, которое способна принять принимающая сторона.
- 4-разрядное поле *длины заголовка* определяет длину TCP-заголовка в 32-разрядных словах. TCP-заголовок может иметь переменную длину благодаря полю параметров, описанному ниже (как правило, поле параметров является пустым; это означает, что длина заголовка составляет 20 байт).
- Необязательное *поле параметров* используется в случаях, когда передающая и принимающая стороны «договариваются» о максимальном размере сегмента, либо для масштабирования окна в высокоскоростных сетях. Также в этом поле определяется параметр временных меток. Дополнительную информацию вы можете найти в документах RFC 854 и RFC 1323.
- Поле *флагов* состоит из б бит. *Бит подтверждения* (ACK) указывает на то, что значение, содержащееся в квитанции, является корректным. Биты *RST, SYN* и *FIN* используются для установки и завершения соединения (они рассматриваются в конце этого раздела). Установленный бит *PSH* указывает на то, что данные сегмента должны быть переданы верхнему уровню принимающей стороны немедленно. Наконец, бит *URG* показывает, что в сегменте нахо-

дятся данные, помещенные верхним уровнем как «срочные». Расположение последнего байта срочных данных указано в 16-разрядном поле *указателя срочных данных.* На принимающей стороне протокол TCP должен уведомить верхний уровень о наличии срочных данных в сегменте и передать ему указатель на конец этих данных. (На практике флаги PSH, URG и поле указателя срочных данных не используются. Мы упомянули о них лишь для полноты описания.)

Порядковые номера и номера подтверждения

Поля порядкового номера и номера подтверждения являются наиболее важными в заголовке TCP-сегмента, поскольку играют ключевую роль в функционировании службы надежной передачи данных. Однако перед тем, как рассматривать роль этих полей в механизме надежной передачи, обратимся к величинам, которые протокол TCP помещает в эти поля.

Протокол TCP рассматривает данные как неструктурированный упорядоченный поток байтов. Такой подход проявляется в том, что TCP назначает порядковые номера не сегментам, а каждому передаваемому байту. Исходя из этого, *порядковый номер сегмента* определяется как порядковый номер первого байта этого сегмента. Рассмотрим следующий пример. Пусть хост A желает переслать поток данных хосту B через TCP-соединение. Протокол TCP на передающей стороне неявно нумерует каждый байт потока. Пусть размер передаваемого файла составляет 500 000 байт, величина MSS равна 1000 байт, и первый байт потока имеет порядковый номер 0. Как показано на рис. 3.27, TCP разбивает поток данных на 500 сегментов. Первому сегменту присваивается порядковый номер 0, второму сегменту — номер 1000, третьему сегменту — номер 2000, и т. д. Порядковые номера заносятся в поля порядковых номеров каждого TCP-сегмента.

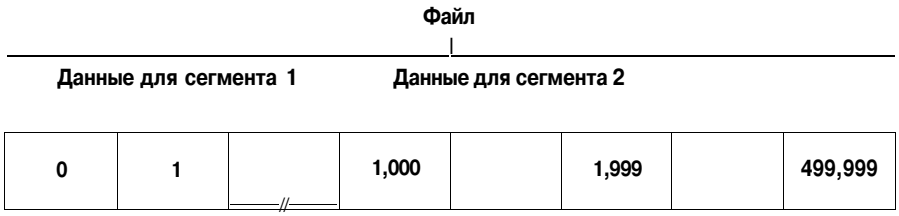


Рис. 3.27. Разбиение данных файла на TCP-сегменты

Теперь рассмотрим номера подтверждения. Вспомним о том, что протокол TCP обеспечивает дуплексную передачу данных, то есть через единственное TCP-соединение данные между хостами A и B могут передаваться одновременно в обе стороны. Каждый сегмент, исходящий из хоста B, содержит порядковый номер данных, передающихся от хоста B к хосту A. *Номер подтверждения,* который хост A помещает в свой сегмент, — это порядковый номер следующего байта, ожидаемого хостом A от хоста B. Рассмотрим следующий пример. Предположим, что хост A получил все байты с номерами от 0 до 535, посланные хостом B, и формирует сегмент для передачи хосту B. Хост A ожидает, что следующие байты, посылаемые

хостом B, будут иметь номера, начинающиеся с 536, и помещает число 536 в поле номера подтверждения своего сегмента.

Рассмотрим другую ситуацию. Пусть хост A получил от хоста B два сегмента, в первом из которых содержатся байты с номерами от 0 до 535, а во втором — байты с номерами от 900 до 1000. Это означает, что по какой-либо причине байты с номерами от 536 до 899 не были получены хостом A. В этом случае хост A ожидает появления отсутствующих байтов и в поле номера подтверждения своего сегмента помещает число 536. Поскольку TCP квитирует принятые данные до первого отсутствующего байта, говорят, что он поддерживает *общее квитирование*.

Последний пример демонстрирует очень важный аспект функционирования протокола TCP. Третий сегмент (содержащий байты 900-1000) принят хостом A раньше, чем второй (содержащий байты 536-899), то есть с нарушением порядка следования данных. Возникает вопрос: как протокол TCP реагирует на нарушение порядка? Оказывается, что спецификация протокола предоставляет программистам, реализующим TCP, полную свободу в разрешении этого вопроса. Тем не менее выделяются два основных подхода: принимающая сторона может либо немедленно проигнорировать сегменты, нарушающие порядок следования данных, либо сохранять принятые сегменты до тех пор, пока недостающие данные не будут получены. Первый подход позволяет упростить программирование, в то время как второй повышает эффективность использования линий связи.

На рис. 3.27 первым порядковым номером является 0, однако на практике стороны протокола TCP выбирают начальный номер случайным образом. Это объясняется необходимостью минимизировать вероятность нахождения в сети сегмента, который сформирован другим (уже закрытым) TCP-соединением между этими же двумя хостами, но который может быть по ошибке отнесен к существующему TCP-соединению. Заметим, что существующее соединение может использовать те же номера портов, что и предыдущее [497].

Telnet — пример на тему порядковых номеров и номеров подтверждения

Протокол Telnet, описанный в документе RFC 854, является популярным протоколом прикладного уровня, применяемым для удаленного доступа к сети. В нем используются службы протокола TCP, и он может выполняться на любой паре хостов. В отличие от приложений, рассмотренных в главе 2 и передающих большие объемы данных, Telnet является интерактивным приложением. Ниже мы рассмотрим пример обмена данными по протоколу Telnet, поскольку он очень наглядно иллюстрирует механизм использования порядковых номеров и номеров подтверждения протокола TCP.

Предположим, что хост A инициирует Telnet-сеанс с хостом B. Это означает, что A будет играть роль клиента, а B — сервера. Каждый символ, введенный пользователем клиентской стороны, передается удаленному хосту; последний отсылает обратно копии символов, которые отображаются на экране пользователя. Это «обратное эхо» позволяет убедиться в том, что вводимые символы успешно принимаются сервером. Таким образом, введенные символы проходят двойной путь между хостами перед тем, как отобразиться на экране.

Предположим, что пользователь вводит единственный символ «C», и посмотрим, какими TCP-сегментами обмениваются клиент и сервер в этом случае. На рис. 3.28 показано, что начальными порядковыми номерами клиента и сервера являются 42 и 79 соответственно. Как вы помните, порядковый номер TCP-сегмента определяется как порядковый номер первого байта его поля данных. Таким образом, первый переданный клиентом сегмент будет иметь номер 42, а первый сегмент, переданный сервером, — 79. Далее, вспомним о том, что номер подтверждения — это порядковый номер байта данных, которого ожидает хост. После того как TCP-соединение установлено, клиент будет ожидать получения байта с номером 79, а сервер — байта с номером 42.

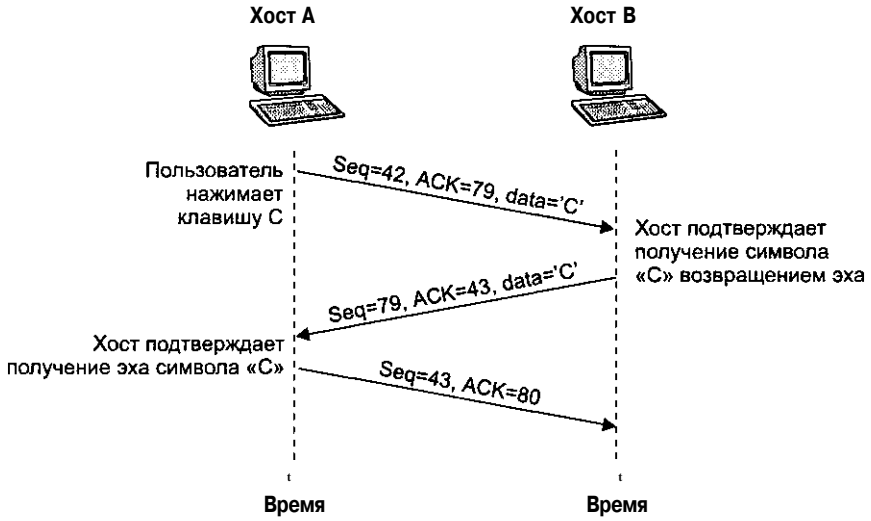


Рис. 3.28. Порядковые номера и номера подтверждений в простом Telnet-приложении, работающем поверх протокола TCP

Как видно из рисунка, в рассматриваемом нами случае между хостами передаются три сегмента. Первый сегмент посылается клиентом и содержит в своем поле данных ASCII-код символа «C», а в поле порядкового номера — число 42. Поскольку к моменту отправки сегмента клиентом не получено никаких данных от сервера, поле номера подтверждения будет содержать число 79.

Второй сегмент передается от сервера к клиенту и содержит две информационные составляющие. Во-первых, он квитирует данные, полученные от клиента: число 43 в поле номера подтверждения говорит о том, что байты с номерами до 42 приняты сервером успешно и ожидается байт с номером 43. Во-вторых, сегмент возвращает символ «C» клиенту, поэтому ASCII-код этого символа заносится в поле данных. Порядковый номер сегмента равен 79, поскольку он является первым сегментом, посылаемым клиентом. Обратите внимание на то, что квитанции для данных, передаваемых от клиента к серверу, отправляются вместе с данными, передаваемыми от сервера к клиенту. Подобные квитанции называют *вложенными*.

Наконец, третий сегмент вновь передается от клиента к серверу. Он необходим лишь для квитирования данных, посланных сервером в предыдущем сегменте (ASCII-кода символа «С»). Поле данных третьего сегмента оказывается пустым (квитанция не является «вложенной» в том смысле, что она не передается вместе с данными). Поскольку байт с номером 79 был успешно принят клиентом, последний ожидает появления байта с номером 80 и записывает число 80 в иоле номера подтверждения. Кроме того, сегмент имеет порядковый номер 43. Наличие порядкового номера на первый взгляд кажется лишним (сегмент не несет никаких данных), однако, будучи обязательной частью сегмента, поле порядкового номера всегда должно содержать какое-либо значение.

Время оборота и интервал ожидания

Как мы увидим далее, протокол TCP (как и протокол **rdt,** созданный нами в предыдущем разделе) использует интервалы ожидания и повторные передачи для решения проблемы потерянных сегментов. Несмотря на концептуальную простоту, при реализации подобного механизма в конкретных протоколах (например, в TCP) приходится учитывать множество нюансов. Например, нужно решать вопрос определения длительности интервала ожидания. Очевидно, что интервал ожидания должен быть больше времени оборота, равного времени получения квитанции передающей стороной, в противном случае неизбежны бесполезные повторные передачи. Однако во сколько раз больше? Как оценить время оборота? Нужно ли связывать таймеры со всеми неподтвержденными сегментами? Все эти вопросы требуют ответов! В данном разделе мы использовали материалы [245] и рекомендации IETF по управлению TCP-таймерами (RFC 2988).

Оценка времени оборота

Первый вопрос, который мы рассмотрим в контексте протокола TCP, — это вопрос об оценке времени оборота между передающей и принимающей сторонами. Под *выборочным временем оборота* (значение **SampleRTT)** будем понимать время, проходящее с момента передачи сегмента протоколу сетевого уровня (протоколу IP) передающей стороны до получения квитанции для этого сегмента. Вместо того чтобы измерять каждое значение **SampleRTT,** TCP делает измерение лишь для одного из переданных, но не квитированных сегментов. Таким образом, оказывается, что с периодичностью приблизительно в одно время оборота значение **SampleRTT** обновляется. Кроме того, **SampleRTT** никогда не измеряется для сегментов, передаваемых повторно (одно из упражнений, приведенных в конце этой главы, предлагает вам объяснить, почему).

Вследствие возможных перегрузок в маршрутизаторах на пути сегментов, а также изменения загрузок оконечных систем значение **SampleRTT** постоянно меняется. Это означает, что в любой момент времени оно может значительно отклониться от типичного значения. Для получения типичного значения необходимо некоторым способом усреднить величину **SampleRTT.** Для этого в протоколе TCP вводится величина **EstimatedRTT,** вычисляемая вместе с каждым новым значением **SampleRTT** по формуле

$$EstimatedRTT = (1 - a) \times EstimatedRTT + a \times SampleRTT.$$

Подобная запись похожа на оператор языка программирования: новое значение EstimatedRTT вычисляется с использованием старого значения и значения SampleRTT. Величину а рекомендуется (RFC 2988) принимать равной 0,125 (то есть 1/8); при этом формула принимает вид

$$\text{EstimatedRTT} = 0{,}875 \times \text{EstimatedRTT} + 0{,}125 \times \text{SampleRTT}.$$

Таким образом, EstimatedRH является весовым средним значением SampleRH. В упражнениях, приведенных в конце главы, демонстрируется, что при использовании данной формулы наибольший вес имеют последние измерения значения SampleRTT. Это придает величине EstimatedRTT актуальность, поскольку она в большей степени отражает текущую ситуацию в сети, чем ее предыдущие состояния. В статистике подобные величины *называют экспоненциальным весовым скользящим средним*. Термин «экспоненциальное» означает, что вес каждого значения SampleRTT экспоненциально убывает с появлением новых значений. В упражнениях, приведенных в конце главы, вам будет предложено извлечь экспоненциальный член из формулы EstimatedRTT.

На рис. 3.29 показаны значения SampleRTT и EstimatedRTT для TCP-соединения между хостами gaia.cs.umass.edu в штате Массачусетс и fantasia.eurecom.fr на юге Франции при а = 0,125. Как видно, график EstimatedRTT значительно сглажен по сравнению с графиком SampleRTT.

Рис. 3.29. Выборочные значения времени оборота и их весовые средние значения

Кроме усредненного значения времени оборота полезно располагать мерой его изменчивости. В RFC 2988 описывается величина DevRTT как приближенное отклонение SampleRTT от EfstimatedRTT:

$$DevRTT = (1 - P) \times DevRTT + (3 \times |SampleRTT - EstimatedRTT|.$$

Обратите внимание на то, что **DevRTT** представляет собой экспоненциальное весовое скользящее среднее разности между **SampleRTT** и **EstimatedRTT.** Чем меньше разброс значений **SampleRTT,** тем меньшей является величина **DevRTT.** Коэффициент P рекомендуется считать равным 0,25.

Определение и управление величиной интервала ожидания

Как определить величину временного интервала ожидания на основе значений **EstimatedRTT** и **DevRTT?** Очевидно, что интервал ожидания должен быть не меньше **EstimatedRTT,** поскольку в противном случае это приведет к лишним повторным передачам. Вместе с тем интервал ожидания не должен значительно превосходить значение **EstimatedRTT:** чем больше времени требуется на обнаружение факта потери пакета, тем большие задержки при передаче данных испытывает приложение. Таким образом, для задания интервала ожидания наиболее предпочтительно увеличить значение **EstimatedRTT** на некоторую переменную величину, зависящую от разброса значений **SampleRTT.** Именно такой величиной является **DevRTT.** В протоколе TCP используется следующая формула для вычисления интервала ожидания:

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT.$$

Надежная передача данных

Как мы упоминали ранее, протокол сетевого уровня IP предоставляет транспортному уровню службу ненадежной передачи данных. IP не дает гарантий относительно доставки дейтаграмм, сохранения порядка их следования и корректности информации. При перегрузке маршрутизаторов дейтаграммы могут быть потеряны, порядок их получения может отличаться от порядка отправки, и, кроме того, допускаются искажения битов (изменения значений с 0 на 1 и наоборот). Поскольку дейтаграммы являются средством передачи сегментов транспортного уровня, последний сталкивается с перечисленными проблемами.

Тем не менее протокол TCP способен обеспечить надежную передачу данных, используя службы ненадежной передачи протокола IP. TCP гарантирует, что данные, которые процесс считывает из своего TCP-буфера, не содержат искажений, пропусков и дублирований, а также расположены в правильном порядке. Другими словами, входной поток принимающей стороны в точности совпадает с выходным потоком передающей стороны. Ниже мы увидим, каким образом TCP обеспечивает подобные гарантии. Оказывается, в основе его функционирования лежат многие из принципов, описанных в предыдущем разделе этой главы.

Простым и удобным механизмом, которым мы пользовались при построении собственного протокола надежной передачи данных, является связывание каждого переданного неподтвержденного сегмента с индивидуальным таймером. На практике управление таймерами оказывается далеко не столь тривиальным, и в RFC 2988 рекомендуется использовать *единственный* таймер для всех сег-

ментов, которые должны быть квитированы. Эта рекомендация учитывается и в протоколе TCP.

Изучение механизма надежной передачи данных в TCP будет происходить в два этапа. Сначала мы рассмотрим упрощенную модель передающей стороны TCP, в которой интервалы ожидания являются единственным средством обнаружения потерь пакетов. Затем мы усложним модель, введя в нее механизм дублирования подтверждений. При этом мы будем опираться на предположение о том, что передача данных ведется в одном направлении, от хоста A к хосту B, и объем передаваемых данных значителен (например, хост A посылает файл большого размера).

Упрощенная программа на псевдоязыке для передающей стороны приведена ниже. В ней обрабатываются три события, связанные с передачей данных: получение новых данных от верхнего уровня, истечение интервала ожидания и получение подтверждения (ACK). При наступлении первого из событий данные заключаются в сегмент, который затем передается протоколу IP. Каждый сегмент содержит порядковый номер, равный номеру первого байта данных в нем. Кроме того, если таймер не был запущен ранее для какого-либо сегмента, в момент передачи текущего сегмента протоколу IP происходит его запуск (представьте себе, что таймер связан с самым «старым» неподтвержденным сегментом). Интервал ожидания TimeoutInterval, устанавливаемый для таймера, вычисляется по формуле, приведенной выше.

```
/* Предполагается, что передающая сторона не ограничивает поток данных и не контролирует
перегрузки; объем данных, передаваемый верхним уровнем, меньше величины MSS; передача
ведется в одном направлении. */
NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber

Цикл (бесконечно) {
  switch(co6biTne)

    событие: получены данные от верхнего уровня
      создать TCP-сегмент с порядковым номером NextSeqNum
      if (таймер не запущен)
        запустить таймер
      передать сегмент протоколу IP
      NextSeqNum = NextSeqNum + length(данные)
      break:

    событие: истек интервал ожидания
      снова послать неподтвержденный сегмент с наименьшим порядковым номером
      запустить таймер
      break:

    событие: получена квитанция со значением поля подтверждения y
      if (y > SendBase) {
        Sendbase = y
        if (есть хотя бы 1 неподтвержденный сегмент)
          запустить таймер
      }
      break;
} /* конец тела бесконечного цикла*/
```

При наступлении второго события (истечения интервала ожидания) TCP производит повторную передачу сегмента, для которого истек интервал ожидания, а затем перезапускает таймер.

Наконец, последним событием, на которое реагирует протокол TCP, является получение квитанции (то есть сегмента, содержащего корректное значение в поле подтверждения). TCP сравнивает значение поля подтверждения с переменной **SendBase,** хранящей значение порядкового номера самого старого из неподтвержденных байтов (значение **SendBase - 1**, таким образом, является порядковым номером последнего байта, принятого принимающей стороной без ошибок и в правильном порядке). Как упоминалось ранее, в протоколе TCP используются общие квитанции, то есть квитанция с номером подтверждения **y** свидетельствует о том, что все байты с номерами, меньшими **y,** успешно приняты. Если оказывается, что y > **SendBase,** это означает, что пришедшая квитанция подтверждает получение одного или нескольких сегментов, не квитированных ранее; значение **SendBase** обновляется, и для этих сегментов происходргг перезапуск таймера.

ПРИНЦИПЫ И ПРАКТИКА

Протокол TCP обеспечивает надежную передачу данных, используя положительные квитанции (подтверждения) и таймеры почти также, как было описано в предыдущем разделе. TCP квитирует корректно принятые данные и осуществляет повторную передачу в случаях, если сегменты или их квитанции были искажены или потеряны. Последние версии TCP также обладают неявным отрицательным квитированием: существует механизм ускоренной повторной передачи, который действует следующим образом. Получение трех одинаковых квитанций для сегмента является признаком потери следующего за ним сегмента и вызывает повторную передачу последнего до истечения интервала ожидания. Для обнаружения потерь и дублирований сегментов в TCP используются порядковые номера. Как и наш протокол rdt 3.0, протокол TCP не способен различать потери, задержки и искажения сегментов и их квитанций. Во всех случаях передающая сторона реагирует на это одним и тем же действием — повторной передачей сегмента.

В протоколе TCP используется механизм конвейеризации, то есть передаются несколько сегментов без ожидания квитанции для каждого из них. Мы убедились в том, что конвейеризация может значительно повысить коэффициент полезности, если отношение времени передачи одного сегмента ко времени оборота мало. Число сегментов, которые могут быть переданы без ожидания квитанций, определяется механизмами контроля потоков данных и перегрузки. Мы рассмотрим механизм контроля потоков в конце этого раздела; контроль перегрузки будет обсуждаться в разделе «Контроль перегрузок в TCP». Сейчас нам достаточно просто знать о том, что в протоколе TCP используется конвейеризация.

Несколько интересных сценариев

Итак, мы упрощенно описали механизм, с помощью которого TCP осуществляет надежную передачу данных. Тем не менее оказывается, что даже такая простая модель не лишена некоторых нюансов. Сейчас мы рассмотрим несколько ситуаций и соответствующих режимов работы протокола TCP. Первую ситуацию

иллюстрирует рис. 3.30, где хост А посылает один сегмент хосту В. Предположим, что сегмент имеет порядковый номер 92 и содержит 8 байт данных. После передачи этого сегмента хост А ожидает от хоста В сегмент с номером подтверждения 100. Положим, что сегмент, посланный хостом А, успешно получен, а сегмент хоста В оказывается потерянным. В этом случае происходит истечение интервала ожидания, и хост А отправляет свой сегмент повторно. Хост В получает этот сегмент и по его порядковому номеру выясняет, что такой сегмент им уже принят. Это приводит к удалению повторно переданного сегмента.

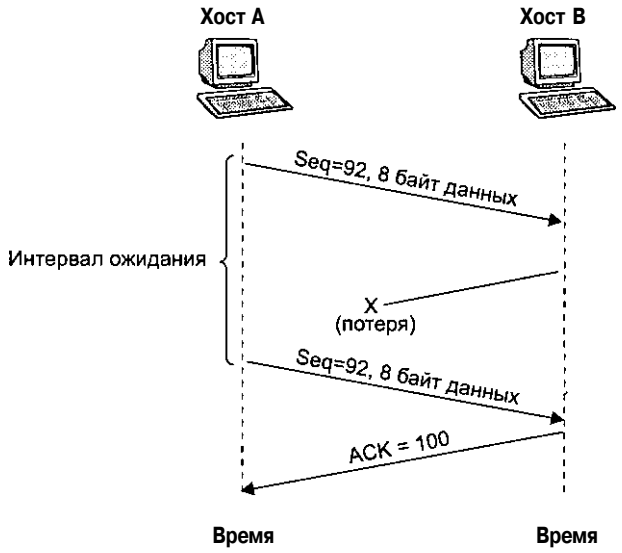


Рис. 3.30. Повторная передача, вызванная потерей квитанции

Вторая ситуация представлена на рис. 3.31. Хост А передает подряд два сегмента, первый из которых имеет порядковый номер 92 и содержит 8 байт данных, а второй сегмент имеет порядковый номер 100 и содержит 20 байт данных. Предположим, что оба сегмента успешно принимаются хостом В, который генерирует две квитанции с номерами подтверждения 100 и 120 соответственно. Пусть ни одна из квитанций не достигает хоста А до истечения интервала ожидания. В этом случае хост А повторно пересылает сегмент с номером 92 и перезапускает таймер. Поскольку хост А получает квитанцию для второго сегмента до нового истечения интервала ожидания, повторной передачи второго сегмента не происходит.

Третья ситуация приведена на рис. 3.32 и отличается от второй тем, что квитанция для первого из сегментов теряется, а вторая квитанция (с номером подтверждения 120) достигает хоста А до истечения интервала ожидания. Получение этой квитанции хостом А сигнализирует о том, что все байты до 119 включительно успешно получены хостом В; таким образом, хосту А не требуется повторно пересылать ни один из двух сегментов.

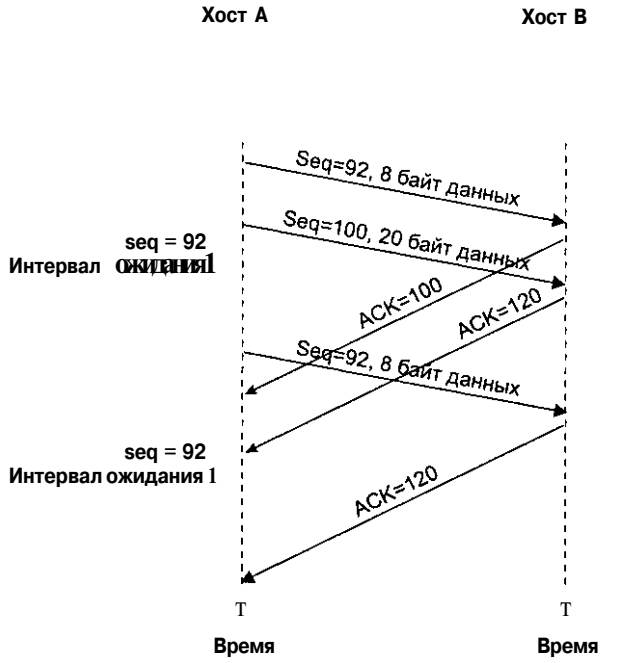


Рис. 3.31. Повторная передача сегмента с номером 100 не производится

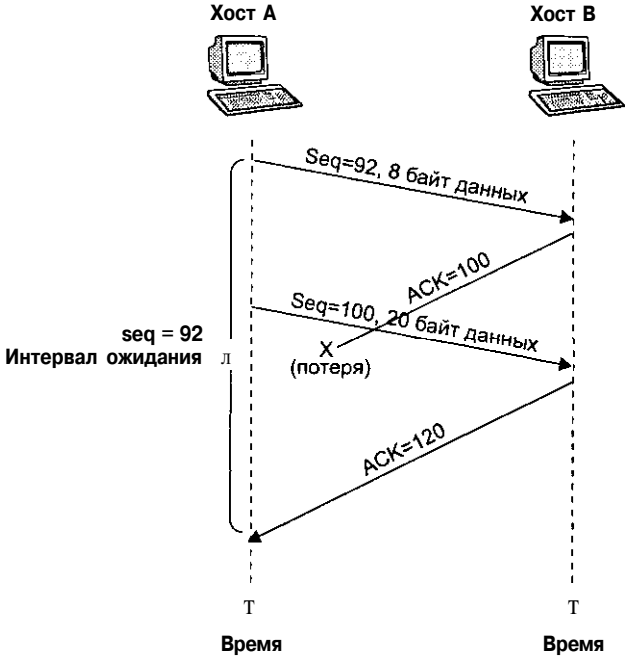


Рис. 3.32. Общая квитанция позволяет избежать повторной передачи первого сегмента

Удвоение интервала ожидания

Здесь мы рассмотрим несколько модификаций предыдущей модели, присутствующих в большинстве реализаций протокола TCP. Первая модификация заключается в изменении длительности интервала ожидания по его истечении. Мы знаем, что по истечении интервала ожидания TCP осуществляет повторную передачу неподтвержденного сегмента с наименьшим порядковым номером. При этом оказывается, что вместо расчета нового интервала ожидания с использованием значений EstimatedRTT и DevRTT (см. подраздел «Время оборота и интервал ожидания» данного раздела) TCP удваивает текущее значение интервала ожидания. Пусть, например, при первом переполнении таймера было установлено время ожидания 0,75 с. Первая повторная передача сегмента будет происходить с таймером, установленным на 1,5 с. Если интервал ожидания вновь истечет, таймер будет установлен на 3 с, и т. д. Таким образом, увеличение интервала ожидания происходит экспоненциально при каждой новой повторной передаче. Однако, если запуск таймера происходит при наступлении одного из двух других событий (получения данных от верхнего уровня или квитанции), длительность интервала ожидания рассчитывается обычным способом — при помощи величин EstimatedRTT и DevRTT.

Такая модификация является «мягкой» формой контролирования перегрузки (другие механизмы контроля перегрузки будут рассмотрены в разделе «Контроль перегрузок в TCP»). Истечение интервала ожидания, как правило, свидетельствует о наличии перегрузок в сети, то есть скоплении большого числа пакетов на одном или нескольких маршрутизаторах, что приводит к значительным задержкам и потерям данных. Если при этом стороны продолжают непрерывно передавать пакеты, перегрузка может стать еще более значительной. Увеличение интервалов ожидания приводит к тому, что повторные передачи осуществляются через возрастающие промежутки времени. Подобный механизм используется в технологии Ethernet; мы убедимся в этом в главе 5.

Ускоренная повторная передача

Одним из недостатков механизма повторной передачи с интервалами ожидания является то, что интервалы ожидания часто оказываются относительно долгими. При потере пакета передающая сторона вынуждена ждать истечения интервала ожидания для того, чтобы осуществить повторную передачу, тем самым увеличивая общую задержку. Здесь на помощь приходит механизм *дублирования подтверждений,* позволяющий передающей стороне обнаруживать потери пакетов до истечения интервала ожидания. Дублирующее подтверждение — это копия положительной квитанции предыдущего сегмента, отправляемая в ответ на получение следующего сегмента, если порядковый номер последнего превышает ожидаемый. Чтобы понять реакцию передающей стороны на появление дублирующих подтверждений, сначала необходимо выяснить причину, побуждающую к использованию такого механизма. В табл. 3.3 приведены основные правила генерации квитанций принимающей стороной TCP. При получении сегмента, порядковый номер которого превышает ожидаемый, протокол регистрирует наличие недостающего сегмента. Появление недостающих сегментов может быть обусловлено по-

терей сегментов или нарушением порядка их следования при передаче по сети. Поскольку в TCP не поддерживается отрицательное квитирование, принимающая сторона не может явно указать на необходимость повторной передачи недостающих данных. Вместо этого она повторно отсылает подтверждение для *последнего успешно принятого* байта (обратите внимание на то, что таблица иллюстрирует ситуацию, когда принимающая сторона не игнорирует сегменты, нарушающие очередность).

Таблица 3.3. Основные правила генерации квитанций принимающей стороной TCP (RFC 1122, RFC 2581)

Событие	Действие протокола TCP на принимающей стороне
Получение сегмента с правильным порядковым номером. На данные со всеми ожидаемыми порядковыми номерами уже получены подтверждения	Задержка в отправке подтверждения. Ожидание в течение 500 мс следующего по порядку сегмента. Если следующий по порядку сегмент не получен в течение этого интервала, отправка подтверждения
Получение сегмента с правильным порядковым номером. Один из уже полученных сегментов ожидает подтверждения	Немедленная отправка общей квитанции, подтверждающей получение обоих сегментов
Получение сегмента с порядковым номером, превышающим ожидаемый. Фиксируется промежуток в принимаемых данных	Немедленная отправка дублирующего подтверждения, показывающего порядковый номер следующего ожидаемого байта (которым является начальный байт промежутка)
Получение сегмента, который частично или полностью заполняет возникший промежуток в принимаемых данных	Немедленная отправка подтверждения в получении сегмента, начавшего заполнение промежутка в данных

Поскольку передающая сторона зачастую передает несколько сегментов подряд, потеря одного из них вызывает генерацию множества дублирующих подтверждений. Если на передачу одного и того же сегмента приходит три дублирующих подтверждения, передающая сторона воспринимает это как указание на потерю следующего за ним сегмента (в упражнениях, приведенных в конце главы, затрагивается вопрос о том, почему передающая сторона ожидает три дублирующих подтверждения, а не одно). Протокол TCP в этом случае осуществляет *ускоренную повторную передачу* пропущенного сегмента, не дожидаясь истечения интервала ожидания (RFC 2581). Для включения в TCP механизма ускоренной повторной передачи код события «получена квитанция со значением поля подтверждения y» в предыдущем фрагменте программы на псевдоязыке необходимо заменить следующим:

```
событие: получена квитанция со значением поля подтверждения y
      if (y > SendBase) {
        Sendbase=y
        if (есть хотя бы 1 неподтвержденный сегмент)
          запустить таймер
        }
      else /* дублирующее подтверждение для уже квитированного сегмента */
        увеличить на 1 число дублирующих подтверждений сегмента y
```

```
if (число дублирующих ACK для y == 3) {
  /* ускоренная повторная передача */
  повторно передать сегмент с порядковым номером y
  }
break;
```

Как мы говорили ранее, при реализации механизма интервалов ожидания и повторных передач в протоколах (например, TCP) возникает множество нюансов. Процедуры, приведенные выше, являются результатом пятнадцати лет экспериментирования с TCP-таймерами и в полной мере подтверждают это!

Возвращение на N шагов назад или выборочное повторение

Напоследок мы зададимся вопросом, к каким протоколам относится TCP: к GBN- или к SR-протоколам? Как мы знаем, в TCP используется общее квитирование, и для неискаженных сегментов, полученных с нарушением порядка следования, не формируются отдельные квитанции. Таким образом, передающей стороне TCP необходимо хранить лишь наименьший порядковый номер отправленного неподтвержденного байта SendBase и порядковый номер следующего передаваемого байта NextSeqNum (см. рис. 3.18 и приведенный в начале этого подраздела фрагмент программы на псевдоязыке). Это создает видимость сходства TCP с GBN-протоколами; тем не менее между ними существуют важные различия. Первое различие заключается в том, что в реализациях TCP, как правило, предусмотрена буферизация пакетов, полученных с нарушением порядка следования [484]. Далее, представим себе, что происходит, когда передающая сторона посылает последовательность сегментов 1, 2 , A ^ и все сегменты успешно принимаются приемной стороной в правильном порядке. Пусть квитанция для одного из сегментов с номером *n* < Утеряется, но оставшиеся *N- 1* квитанций достигают передающей стороны до истечения интервала ожидания. В этом случае GBN-протокол осуществил бы повторную передачу не только пакета *n,* но и всех последующих пакетов *(n* + 1, я + 2 , *N)* . TCP, напротив, в худшем случае передает единственный сегмент *n;* более того, если квитанция для сегмента *n* + 1 принимается передающей стороной до истечения интервала ожидания, повторная передача сегмента *n* вообще не производится.

В предлагаемой модификации протокола TCP используется механизм *выборочного подтверждения* (RFC 2581), позволяющий принимающей стороне индивидуально квитировать сегменты, нарушающие порядок следования, а не выдавать общую квитанцию для последнего корректно принятого сегмента. Выборочное подтверждение в совокупности с выборочной повторной передачей, исключающей повторную передачу сегментов, для которых получены общие квитанции, придает протоколу TCP значительное сходство с SR-протоколами. Таким образом, механизм надежной передачи данных TCP следует рассматривать как сочетание подходов GBN и SR.

Контроль потока

Как говорилось ранее, на обоих хостах, между которыми установлено TCP-соединение, имеются приемные буферы. Протокол TCP помещает в приемные буферы

байты, принятые без искажений и в правильном порядке. Приложение, связанное с TCP-соединением, считывает данные из приемного буфера в произвольные моменты времени, не зависящие от поступления новых данных. К примеру, приложение может быть занято выполнением трудоемких операций и обращаться к буферу со значительными задержками. Если частота поступления данных в буфер превышает частоту их считывания приложением, то через некоторый интервал времени возникает угроза переполнения буфера.

Для того чтобы избежать переполнения приемного буфера, протокол TCP предоставляет приложениям *службу контроля потока,* призванную контролировать, чтобы частота передачи данных соответствовала частоте их считывания принимающим приложением. Как уже упоминалось, скорость передачи данных может быть принудительно снижена при наличии перегрузок в сети; механизм, реализующий эту операцию, называется механизмом *контроля перегрузок* и детально рассматривается в разделах «Принципы контролирования перегрузки» и «Контроль перегрузок в TCP». Несмотря на то что контроль потока и контроль перегрузок решают одну и ту же задачу (снижение скорости передачи), причины, вызывающие их выполнение, не имеют ничего общего. К сожалению, авторы многих книг употребляют эти два термина как синонимы, оставляя сообразительным читателям возможность самим «почувствовать разницу». Ниже мы увидим, каким образом TCP осуществляет контроль потока. Для того чтобы сосредоточить максимум внимания на концепции механизма, будем считать, что рассматриваемая реализация TCP игнорирует сегменты, нарушающие порядок следования данных.

Протокол TCP обеспечивает контроль потока с помощью переменной, называемой *окном приема.* Говоря неформальным языком, окно приема используется для оповещения передающей стороны об объеме свободного места в буфере принимающей стороны. Поскольку протокол TCP осуществляет дуплексную передачу данных, окно приема поддерживается обеими сторонами соединения. Рассмотрим понятие окна приема на примере передачи файла. Пусть хост A пересылает файл большого размера хосту B при помощи TCP-соединения. Хост B выделяет буфер для этого соединения, а процесс, которому адресован файл, периодически считывает данные из буфера. Размер буфера хранится в переменной RcvBuffer. Определим следующие переменные

- LastByteRead — номер последнего байта потока данных, считанного из буфера прикладным процессом хоста B;
- LastByteRcvd — номер последнего байта потока данных, полученного от передающей стороны и помещенного в буфер.

Чтобы входной буфер не мог переполниться, необходимо соблюдать неравенство

$$\text{LastByteRcvd} - \text{LastByteRead} < \text{RcvBuffer}.$$

Значение окна приема RcvWindow равно объему свободного места в буфере:

$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}].$$

Поскольку объем свободного места в буфере не постоянен, значение переменной RcvWindow также динамически меняется. Сказанное иллюстрирует рис. 3.33.

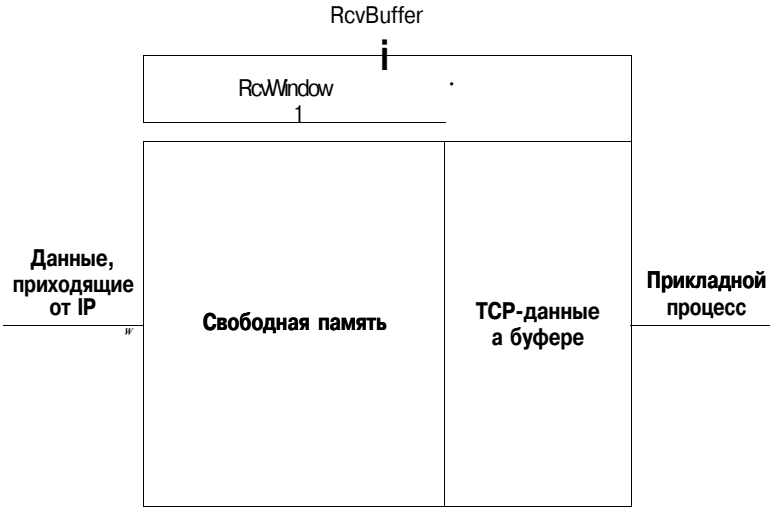


Рис. 3.33. Окно приема (RcvWindow) и приемный буфер (RcvBuffer)

Каким образом соединение использует переменную **RcvWindow** для реализации службы контроля потока? Хост В «говорит» хосту А о том, сколько свободного места имеется в его приемном буфере, помещая значение переменной **RcvWindow** в поле окна приема каждого сегмента, отправляемого хосту А. Начальное значение **RcvWindow** равно **RcvBuffer.** Для того чтобы этот механизм работал, необходимо поддерживать на хосте В несколько переменных, характеризующих данное TCP-соединение.

Хост А поддерживает две переменные, **LastByteSent** и **LastByteAcked,** хранящие номера последнего отправленного и последнего квитированного байтов соответственно. Разность **LastByteSent - LastByteAcked** представляет размер неподтвержденных данных, переданных от хоста А хосту В. Поддерживая значение этой разности меньшим, чем **RcvWindow,** хост А может гарантировать, что приемный буфер хоста В не переполнится. Таким образом, на протяжении жизни соединения хост А должен соблюдать неравенство:

$$\text{LastByteSent} - \text{LastByteAcked} < \text{RcvWindow}.$$

Приведенной схеме присуща небольшая проблема. Предположим, что приемный буфер хоста В оказался заполненным, то есть значение **RcvWindow** стало равно 0. Возможно, что в этот момент хост В не имеет данных для передачи хосту А. Обратите внимание на то, что считывание приложением данных из буфера, приводящее к изменению значения **RcvWindow,** не влечет за собой генерацию пакета с новым значением **RcvWindow.** Напротив, TCP отсылает **RcvWindow** хосту А только вместе с данными или квитанциями. Таким образом, хост А своевременно не получит сведений об освободившемся пространстве буфера и не сможет осуществлять дальнейшую передачу данных. Чтобы решить эту проблему, спецификация TCP предусматривает периодическую генерацию хостом А сегментов, содержащих один байт данных, если окно приема хоста В имеет

нулевое значение. Эти сегменты квитируются принимающей стороной, и при освобождении места в буфере ненулевое значение RcvWindow отправляется вместе с одной из квитанций. ,

Функционирование окна приема протокола TCP наглядно проиллюстрировано с помощью Java-апплета, находящегося на нашем сайте http://www.awl.com/kurose-ross.

Отметим, что протокол UDP, в отличие от TCP, не обеспечивает контроля потока. Представьте себе описанную выше ситуацию, предположив, что вместо протокола TCP приложение использует UDP. Обычно протокол UDP помещает принятые данные в буфер, из которого они извлекаются прикладным процессом через сокет. Если считывание данных из буфера происходит медленнее, чем поступление новых данных, то буфер переполняется и происходит потеря части сегментов.

Управление TCP-соединением

В этом подразделе мы рассмотрим вопросы установления и разрыва TCP-соединения. Несмотря tfa то что эта тема может показаться не столь увлекательной, как тема надежной передачи данных или контроля потока, она является весьма важной, поскольку процедура установления соединения способна в значительной степени увеличить время ожидания (например, при навигации в web). Итак, мы изучаем вопрос установления TCP-соединения. Пусть процесс, выполняющийся на одном хосте (клиент), желает инициировать соединение с процессом, выполняющимся на другом хосте (с сервером). Сначала клиентское приложение уведомляет TCP-клиент о том, что необходимо установить TCP-соединение с сервером. TCP-клиент инициирует TCP-соединение следующим образом.

1. Клиентская сторона TCP отсылает серверной стороне специальный сегмент, не содержащий данных. Флаг SYN, находящийся в заголовке этого сегмента (см. рис. 3.26), установлен в 1, поэтому данный сегмент часто называют *SYN-сегментом*. Клиентская сторона устанавливает начальный порядковый номер (client_isn) и помещает его в поле порядкового номера SYN-сегмента. SYN-сегмент заключается в IP-дейтаграмму и отправляется серверу.
2. Когда IP-дейтаграмма с SYN-сегментом достигает хоста сервера (если не происходит ее потери), сервер извлекает из нее SYN-сегмент, создает буфер и переменные для соединения, а затем отправляет клиенту сегмент, уведомляющий о выделении TCP-соединения. Этот сегмент также не содержит прикладных данных, однако его заголовок несет важные сведения. Во-первых, флаг SYN, как и в предыдущем сегменте, установлен в 1. Во-вторых, в поле подтверждения содержится число client_isn + 1. Наконец, в поле порядкового номера сервер указывает свой начальный порядковый номер server_isn. Если бы хосты могли общаться при помощи слов, то содержимое второго сегмента, вероятно, выглядело бы следующим образом: «Я получил Ваш SYN-сегмент с просьбой установить с Вами TCP-соединение с начальным порядковым номером client_isn. Я согласен удовлетворить эту просьбу. Мой начальный порядковый номер server_isn». Иногда второй сегмент называют *SYNACK-сегментом*.

3. Получив SYNACK-сегмент, клиент выделяет память для буфера и переменных TCP-соединения и отсылает серверу сегмент, подтверждающий получение SYNACK-сегмента — в поле подтверждения помещается число serveMsn + 1. Поскольку соединение уже установлено, флаг SYN сбрасывается в 0.

После выполнения перечисленных шагов клиент и сервер готовы к обмену данными друг с другом. Во всех последующих сегментах значение флага SYN равно 0. Процесс установления TCP-соединения иллюстрирует рис. 3.34. Поскольку в этом процессе клиент и сервер обмениваются тремя сегментами, процедуру установления соединения часто называют *тройным рукопожатием*. Некоторые аспекты тройного рукопожатия затронуты в упражнениях, приведенных в конце главы. (Для чего нужны начальные порядковые номера? Каковы причины использования тройного рукопожатия вместо двойного?) Интересно отметить, что скалолаз и тот, кто его страхует, перед началом восхождения обмениваются тройным рукопожатием, чтобы удостовериться в обоюдной готовности.

Рис. 3.34. Обмен сегментами при тройном рукопожатии в протоколе TCP

Процедура закрытия TCP-соединения подразумевает освобождение памяти, выделенной для буферов и переменных, и может происходить по инициативе любой из сторон. Так, рис. 3.35 иллюстрирует закрытие TCP-соединения клиентской стороной. Клиентский процесс генерирует команду закрытия соединения, которая приводит к отправке TCP-клиентом специального сегмента. В заголовке этого сегмента флаг FIN установлен в 1. Получив данный сегмент, сервер подтверждает это. Затем сервер отсылает клиенту завершающий сегмент, в котором бит FIN также установлен в 1; в свою очередь, получение этого сегмента подтверждается клиентом. После этого все ресурсы соединения на обеих сторонах освобождаются.

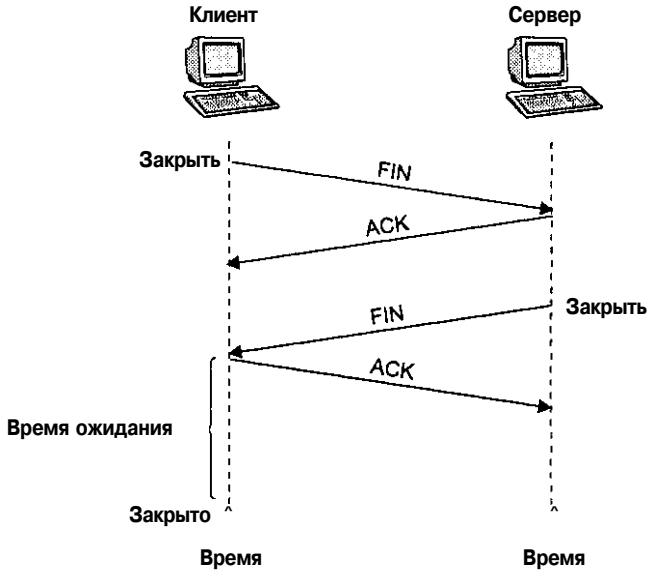


Рис. 3.35. Закрытие TCP-соединения

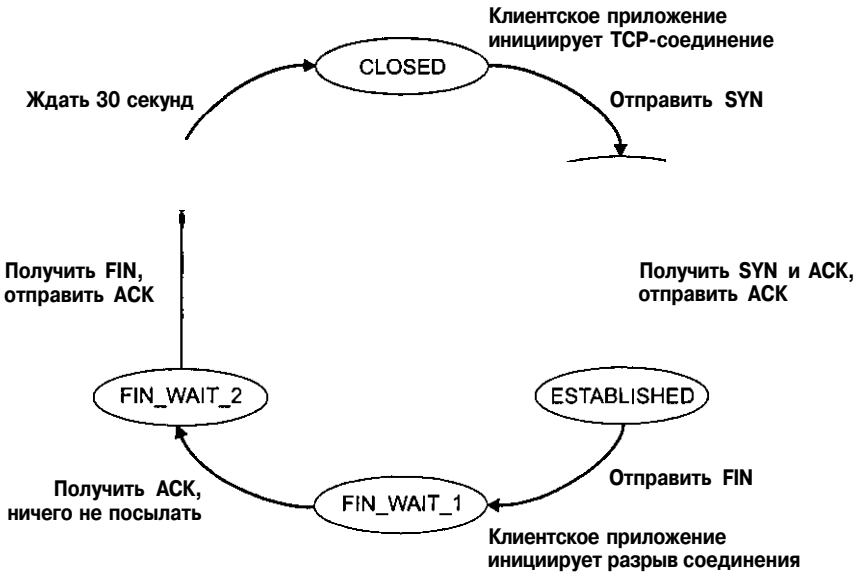


Рис. 3.36. Типичная последовательность состояний клиента TCP

На протяжении жизни TCP-соединения каждая из сторон проходит через серию изменяющихся *TCP-состояний.* На рис. 3.36 приведена типичная последовательность TCP-состояний клиентской стороны. Первым состоянием клиента является состояние CLOSED; в этом состоянии происходит инициирование TCP-соединения клиентским приложением, заключающееся в создании соке-

та (см. примеры Java-программ в главе 2). Клиентская сторона TCP посылает серверной стороне SYN-сегмент и переходит в состояние SYN_SENT. В этом состоянии она ожидает ответного SYNACK-сегмента от сервера, в котором бит SYN установлен в 1. Получив SYNACK-сегмент, клиент входит в состояние ESTABLISHED, в котором он может принимать и отправлять сегменты, содержащие данные прикладного уровня.

Предположим, что закрытие соединения инициируется клиентской стороной (заметим, что сервер также может закрыть соединение). Клиент отправляет TCP-сегмент с битом FIN, установленным в 1, и входит в состояние FIN_WAIT__1. В этом состоянии клиентская сторона ожидает подтверждения (ACK) для переданного сегмента. Получив подтверждение, клиент переходит в состояние FIN_WAIT_2, где ожидает получения от сервера завершающего сегмента с битом FIN, установленным в 1. Получив сегмент, клиент квитирует его и входит в состояние TIME_WAIT. Это состояние предусматривает повторную передачу подтверждения для завершающего сегмента в случае возможной потери этого подтверждения. Длительность нахождения клиента в состоянии TIME_WAIT зависит от реализации протокола, однако наиболее типичными значениями являются 30 с, 1 мин и 2 мин. После выхода из состояния TIME_WAIT происходит формальное закрытие TCP-соединения, при котором освобождаются все его ресурсы, включая номера портов.

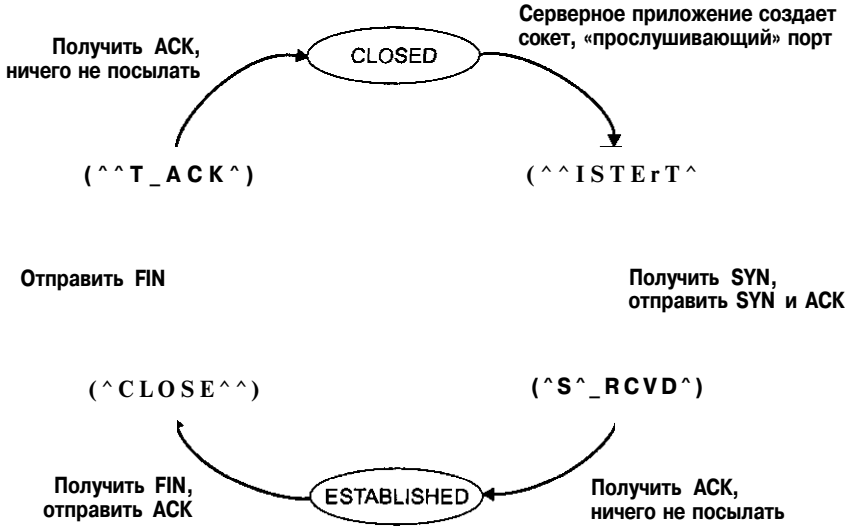


Рис. 3.37. Типичная последовательность состояний TCP-сервера

На рис. 3.37 представлена типичная последовательность состояний серверной стороны TCP-соединения для случая, когда закрытие соединения происходит по инициативе клиентской стороны. Переходы из одного состояния в другое понятны, и мы не будем останавливаться на их описании. Обратите внимание на то, что две приведенные диаграммы переходов соответствуют случаям, когда уста-

новление и закрытие TCP-соединения происходит «в штатном режиме». Мы не рассматривали такие нетипичные ситуации, как, например, одновременные попытки закрыть соединение, предпринимаемые обеими сторонами. Если вас интересует описание подобных ситуаций и прочие нюансы, касающиеся протокола TCP, рекомендуем вам обратиться к дополнительным информационным ресурсам [484].

Итак, мы завершаем разговор о контроле потока и обнаружении ошибок в протоколе TCP. Другим важным вопросом, который нам необходимо изучить, является контроль перегрузки. В следующем разделе мы рассмотрим общие принципы контролирования перегрузки, а затем вновь вернемся к протоколу TCP для того, чтобы увидеть, каким образом изученные принципы применяются на практике.

Принципы контролирования перегрузки

В предыдущих разделах мы рассмотрели общие принципы надежной передачи данных при существовании вероятности потерь пакетов, а также применение этих принципов в протоколе TCP. Как было показано, потери пакетов чаще всего обусловлены переполнением буферов маршрутизаторов при перегрузках в сети. Повторная передача пакетов является признаком перегрузки, однако не устраняет ее причину, которая заключается в том, что передающие стороны посылают слишком большое число пакетов одновременно. Для устранения перегрузок необходимы механизмы, воздействующие на их причину, то есть принудительно снижающие частоту передачи пакетов источниками.

В этом разделе мы рассмотрим общие вопросы, касающиеся проблемы перегрузок. Мы узнаем, в чем заключается вредное воздействие перегрузок, как они влияют на качество обслуживания прикладных процессов, как избегать перегрузок и как реагировать на их появление. Фундаментальный подход к изучению проблемы перегрузок необходим потому, что проблема перегрузок является одной из самых важных в компьютерных сетях. Мы завершим разговор рассмотрением механизмов контроля перегрузки, реализованных в модели обслуживания ABR (Available Bit Rate — доступная битовая скорость) сетей ATM (Asynchronous Transfer Mode — режим асинхронной передачи), а следующий раздел посвятим алгоритму контроля перегрузки протокола TCP.

Причины и следствия перегрузки

Мы начнем изучение вопросов контроля перегрузки с трех ситуаций возникновения перегрузок в сети в порядке возрастания сложности. Для каждой ситуации мы выявим причину и укажем ее негативные последствия (невозможность полного использования ресурсов, ухудшение качества обслуживания). Сейчас мы не станем уделять внимание способам возможного реагирования на перегрузки или избежания их появления; нас будет интересовать, что происходит в сети при увеличении частоты передачи пакетов источниками, приводящем к перегрузкам.

Два источника и маршрутизатор с буферами неограниченной емкости

Мы начнем с рассмотрения самой простой из возможных ситуаций, приводящих к перегрузке в сети. Два хоста (A и B) поддерживают соединения, использующие общий маршрутизатор, как показано на рис. 3.38.

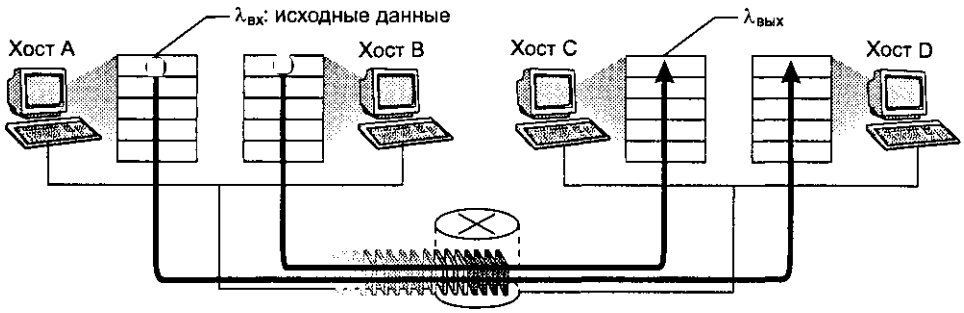


Рис. 3.38. Ситуация 1: два соединения совместно используют маршрутизатор с буферами неограниченной емкости

Предположим, что приложение хоста A осуществляет передачу данных (например, отправляя их транспортному уровню через сокет) со средней скоростью $Л^{вх}$ байт/с. Каждая единица обмена поступает в сокет только один раз. Будем считать, что протокол транспортного уровня максимально примитивен: его функции заключаются лишь в преобразовании данных в сегменты и не имеют механизмов защиты от ошибок (например, путем повторной передачи), контроля потока и предотвращения перегрузки. Пренебрегая увеличением объема передаваемых данных, обусловленных добавлением заголовков транспортного и других уровней, можно считать скорость передачи данных хостом A равной ^{вх} байт/с. Передача данных хостом B осуществляется аналогичным образом, и мы также примем ее скорость равной ^{вх} байт/с. Пакеты, посылаемые хостами A и B, через маршрутизатор поступают в общую линию связи с пропускной способностью *R*. Маршрутизатор располагает буферами, предназначенными для промежуточного хранения пакетов в случаях, когда частота приема пакетов превышает частоту их передачи. В этом примере мы будем считать объем буферного пространства бесконечным.

На рис. 3.39 представлены графики функционирования соединения со стороны хоста A. График слева изображает зависимость *производительности соединения* (числа байтов в единицу времени, получаемых принимающей стороной) от скорости передачи данных передающей стороной. При скоростях передачи, лежащих в диапазоне от 0 до *R/2,* производительность соединения равна скорости передачи: все данные, посылаемые в сеть передающей стороной, достигают хоста назначения. Если скорость передачи превышает значение *R/2,* производительность соединения остается равной *R/2:* часть пакетов попросту теряется из-за

невозможности линии связи осуществить их передачу. Величина *R/2* является максимальным значением производительности для данного случая, и какими бы ни были скорости передачи хостов A и B, ни одному из них не удастся преодолеть этот максимум.

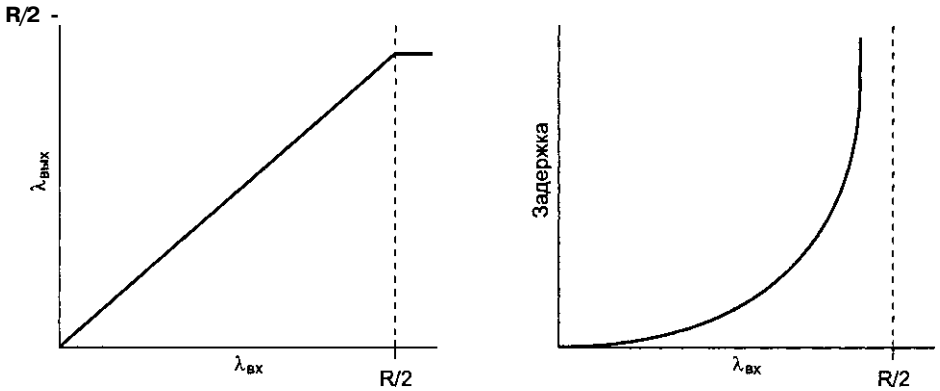


Рис. 3.39. Ситуация 1: зависимости производительности и задержки от скорости передачи данных

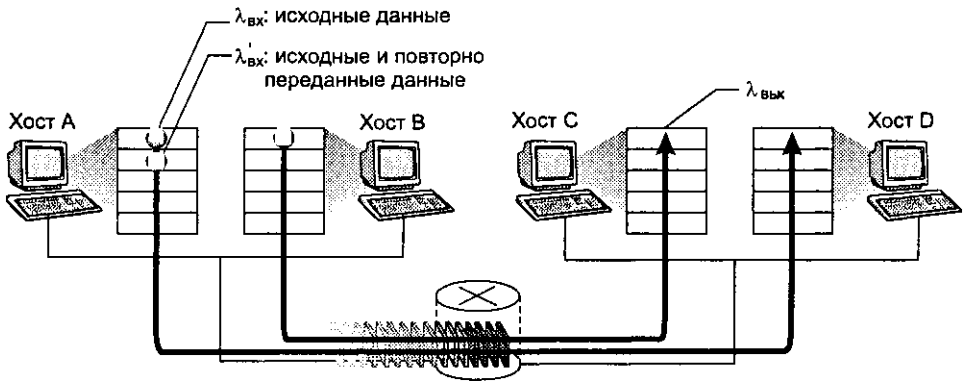
С одной стороны, достижение максимальной производительности каждым соединением можно считать хорошим результатом, поскольку в этом случае ресурсы линии связи используются полностью. С другой стороны, график зависимости задержки от скорости передачи свидетельствует о том, что при производительности, близкой к максимальной (значения $\lambda_{вх}$ близки к *R/2),* задержки передачи данных неограниченно растут (в предположении, что передающие стороны не изменяют скорость передачи в течение бесконечного промежутка времени). Таким образом, оказывается, что высокая производительность негативно воздействует на время передачи данных. Даже в такой чрезвычайно простой и идеализированной ситуации мы столкнулись с вредным воздействием перегрузки в сети на качество обслуживания: *чем ближе скорость передачи данных к пропускной способности линии связи, тем большими становятся задержки ожидания пакетов.*

Два источника и маршрутизатор с буферами ограниченной емкости

Давайте внесем в предыдущую ситуацию несколько изменений (рис. 3.40). Во-первых, будем считать объем буферного пространства маршрутизатора конечным. Это сделает ситуацию более реалистичной, поскольку пакеты, достигающие маршрутизатора с заполненным буфером, будут теряться. Во-вторых, мы предположим, что каждое из соединений является надежным, то есть транспортный уровень осуществляет повторную передачу каждого потерянного пакета. Мы снова обозначим через $\lambda_{вх}$ скорость передачи данных приложения через сокет, а через $\lambda'_{вх}$ — скорость передачи транспортным уровнем сегментов, включающих

как новые, так и повторно передаваемые данные. Величину ^'вх часто называют *прилагаемой нагрузкой.*

Теперь качество обслуживания в значительной степени зависит от механизма повторной передачи. Можно представить невыполнимый на практике вариант, когда хост А некоторым образом определяет, заполнен ли буфер маршрутизатора, и осуществляет передачу только в том случае, если в буфере имеется достаточно свободного места. Такой механизм позволяет полностью избежать потерь, поэтому значения ^вх и X'вх окажутся равными, и через соединение пройдет ^вх данных. На рис. 3.41, *а* данному случаю соответствует верхняя из двух прямых. С точки зрения производительности приведенный механизм является идеальным — все передаваемые пакеты достигают хоста назначения. Обратите внимание на то, что максимальная скорость, с которой хост может передавать данные, не превышает *R/2,* поскольку потери пакетов не допускаются.



Общие буферы неограниченной емкости для выходных линий

Рис. 3.40. Ситуация 2: два хоста и маршрутизатор с буферами конечного объема

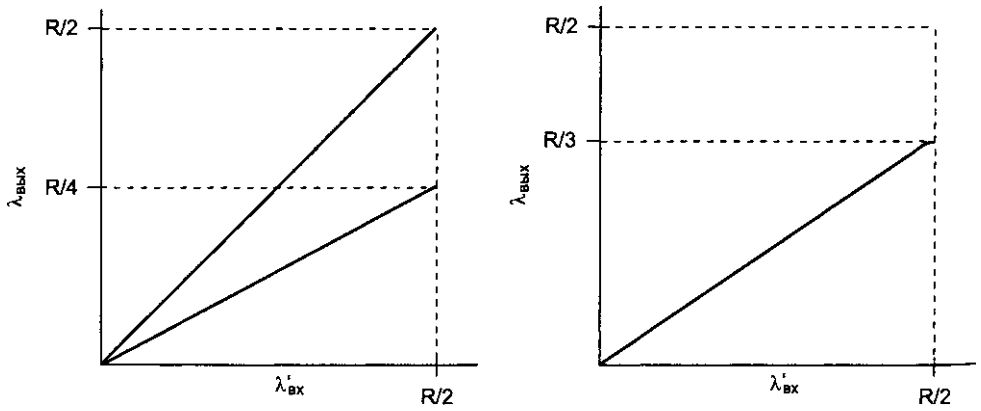


Рис. 3.41. Ситуация 2: производительность

Теперь представим себе немного более реалистичный механизм, осуществляющий повторную передачу только тогда, когда факт потери пакета достоверно установлен. (Разумеется, это допущение тоже идеализирует ситуацию. Тем не менее можно сделать интервал ожидания квитанции настолько большим, что его истечение с высокой вероятностью будет свидетельствовать о потере пакета.) График, характеризующий качество обслуживания для такого механизма, приведен на рис. 3.41, *б*. Представьте, что происходит, когда значение $\lambda'_{вх}$ (скорость новых и повторных передач) равно *R/2*. Согласно графику, производительность в этом случае равна i?/3. Таким образом, суммарная скорость 0,5/? делится на две составляющие: одна, равная 0,333/?, характеризует передачу новых данных, а другая, равная 0,166/?, относится к повторным передачам. Здесь мы наблюдаем еще одно негативное влияние перегрузки на качество обслуживания: *потери пакетов, обусловленные переполнением буферов маршрутизатора, вынуждают передающую сторону делать повторные передачи*.

Наконец, рассмотрим случай, когда задержки пакета могут вызывать получение квитанции передающей стороной после истечения интервала ожидания (пакет при этом не теряется). Это приводит к дублированию пакетов на принимающей стороне и, следовательно, к удалению копий. Таким образом, при передаче копии успешно принятого пакета маршрутизатор не совершает полезной работы; вместо повторной передачи он мог бы использовать ресурс линии связи для передачи пакета с новыми данными. Мы сталкиваемся с еще одним негативным влиянием перегрузок: *повторные передачи в сочетании со значительными задержками ожидания приводят к бесполезной трате значительной доли пропускной способности линий связи*. Нижняя прямая на рис. 3.41, *а* иллюстрирует производительность передачи, когда каждый пакет посылается в среднем 2 раза. Как видно из графика, максимальная производительность такой передачи составляет приблизительно *R/A*.

Четыре источника, маршрутизаторы с буферами ограниченной емкости и пути с несколькими маршрутизаторами

Последнюю ситуацию иллюстрирует рис. 3.42. Четыре хоста обмениваются пакетами, при этом пути пакетов накладываются друг на друга, и каждый путь проходит через два маршрутизатора. Как и ранее, мы будем предполагать, что хосты используют надежную передачу данных, осуществляемую при помощи механизмов интервалов ожидания и повторных передач, скорость передачи всех хостов одинакова и составляет $X_{вх}$, а пропускные способности линий связи равны *R бит/с*.

Рассмотрим соединение между хостами A и C, проходящее через маршрутизаторы R1 и R2. Соединение A-C имеет общий маршрутизатор R1 с соединением D-B и общий маршрутизатор R2 с соединением B-D. Для небольших значений $X_{вх}$ вероятность перегрузки мала (как в первых двух ситуациях), и производительность соединения приблизительно соответствует прилагаемой нагрузке. При увеличении $\lambda_{вх}$ в области низких значений значение $X_{вых}$ также увеличивается, поскольку перегрузки отсутствуют, и, следовательно, прирост получаемых данных равен приросту передаваемых данных.

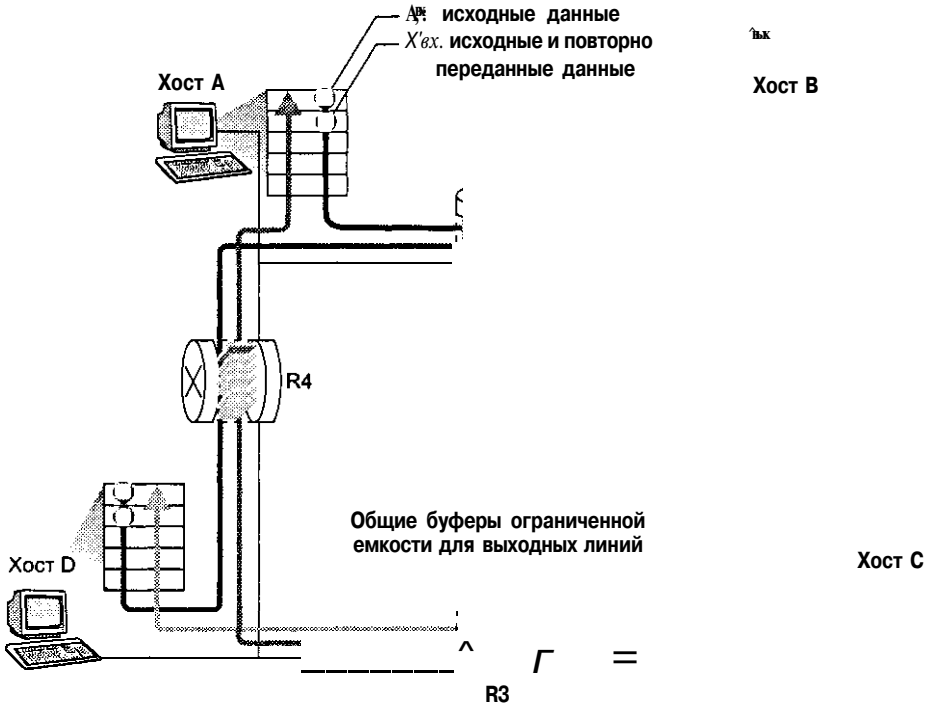


Рис. 3.42. Ситуация 3: четыре источника, маршрутизаторы с буферами ограниченной емкости и пути с несколькими маршрутизаторами

Теперь обратимся к области больших значений $Л^{вх}$ (а значит, и $X\backslash^{x}$). Данные соединения A-C поступают в маршрутизатор R2 из маршрутизатора R1 со скоростью, не превышающей *R,* при любом значении $X^{вх}$. Если значение $\backslash'^{вх}$ окажется большим для всех соединений (включая B-D), то скорость поступления пакетов соединения B-D в маршрутизатор R2 значительно превысит скорость поступления пакетов соединения A-C. Поскольку соединения A-C и B-D используют одно и то же ограниченное буферное пространство маршрутизатора R2, с ростом в последнем числа пакетов соединения B-D (при увеличении прилагаемой загрузки), число пакетов соединения A-C, не теряемых при передаче, уменьшается. В пределе, когда прилагаемая загрузка соединения B-D стремится к бесконечности, производительность соединения A-C становится равной нулю. Это, в свою очередь, приводит к тому, что производительность соединения A-C стремится к нулю в случае высокоинтенсивного трафика. Перечисленные выше соображения позволяют построить кривую зависимости производительности соединения от прилагаемой загрузки, представленную на рис. 3.43.

Причиной падения производительности при высоких прилагаемых загрузках является большое количество бесполезной работы, совершаемой при передаче. Например, в описанной выше ситуации потеря пакета во втором маршрутизаторе приводит к бесполезной работе первого маршрутизатора. Производительность передачи была бы выше, если бы первый маршрутизатор удалил этот пакет сразу

после его получения, а освободившиеся ресурсы использовал для передачи другого пакета, который впоследствии достигнет хоста назначения (в этом смысле было бы хорошо обеспечить маршрутизатору возможность выбирать для дальнейшей передачи те пакеты, которые ранее прошли через наибольшее число маршрутизаторов). Итак, мы наблюдаем еще одно негативное воздействие перегрузок: *при потере пакета вследствие переполнения буфера одного из маршрутизаторов на его пути оказывается бесполезной работа всех маршрутизаторов, через которые прошел этот пакет.*

RJ2 -

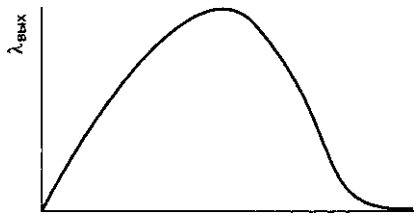


Рис. 3.43. Ситуация 3: производительность

Подходы к контролированию перегрузки

В следующем разделе мы детально рассмотрим механизм контроля перегрузки в протоколе TCP, а сейчас займемся изучением двух общих подходов к этому вопросу, наиболее часто встречающихся на практике.

С самой общей точки зрения можно разделить механизмы контроля перегрузки на те, которые реализуются с помощью сетевого уровня, и те, которые целиком реализуются транспортным уровнем.

- *Контроль перегрузки оконечными системами.* В этом случае контроль перегрузки полностью осуществляется транспортным уровнем. В функции транспортного уровня входит установление факта перегрузки в сети по характерным признакам «поведения» сети (задержки и потери пакетов). В следующем разделе мы увидим, что протоколу TCP необходимо использовать этот механизм контроля перегрузки, поскольку протокол сетевого уровня IP не предоставляет оконечным системам никакой информации относительно перегрузок в сети. Сигналом о наличии перегрузки для TCP служит потеря сегмента (истечение интервала ожидания или получение тройного подтверждения), реакцией на которую является уменьшение размера окна приема. Мы увидим, что в предлагаемых новых проектах TCP признаком перегрузок является увелрхчение времени оборота.
- *Контроль перегрузок с поддержкой на сетевом уровне.* В этом механизме предполагается наличие явной обратной связи, с помощью которой сетевые устройства

(маршрутизаторы) информируют оконечные системы о нагрузках в сети. В простейшем случае обратная связь может представлять собой передачу бита, свидетельствующего о наличии или отсутствии перегрузки в маршрутизаторе. Подобный подход был реализован в архитектурах IBM SNA [452] и DEC DECnet [248, 405], в последнее время предлагается для сетей TCP/IP [155,404] и применяется в механизме контроля перегрузок ABR сетей ATM (о чем будет рассказано далее). Также возможна и более сложная обратная связь: например, одна из форм ABR-контроля позволяет маршрутизатору указывать оконечной системе скорость передачи в выходной линии связи, которую он способен обеспечить.

При контроле перегрузок с сетевой поддержкой информация о наличии перегрузки передается из сети оконечной системе одним из двух путей, как показано на рис. 3.44. Обратная связь может напрямую связывать маршрутизатор и передающую оконечную систему, либо может использоваться специальное поле в передаваемых пакетах: при возникновении перегрузки маршрутизатор делает соответствующую метку в этом поле и передает пакет далее в направлении принимающей стороны. Получив помеченный пакет, принимающая сторона отсылает передающей стороне уведомление о перегрузке. Обратите внимание на то, что в этой схеме передающая сторона «узнает» о перегрузке не позднее чем по истечении времени оборота.

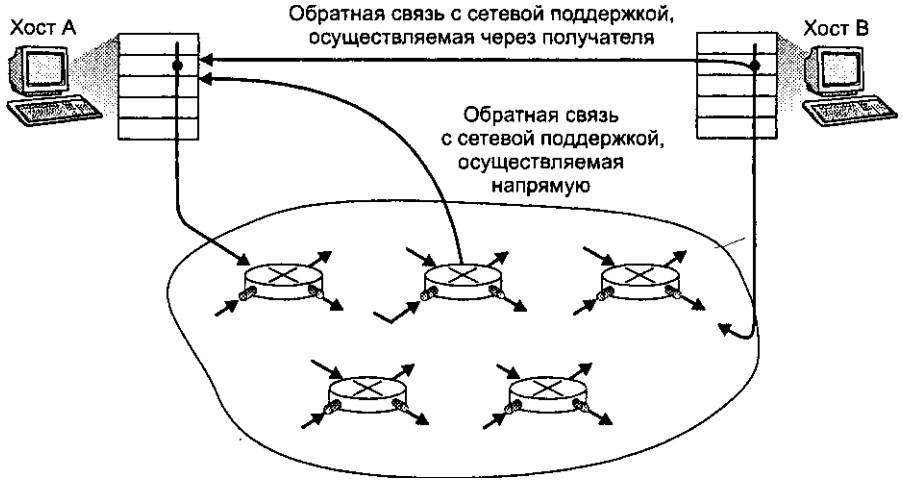


Рис. 3.44. Два возможных варианта обратной связи при контроле перегрузок с сетевой поддержкой

Контроль перегрузок в службе ABR сетей ATM

В разделе «Контроль перегрузок в TCP» мы подробно рассмотрим механизмы контроля перегрузки оконечными системами на примере протокола TCP, а сейчас остановимся на службе ABR сетей ATM, в которой реализован другой вид контроля перегрузок с сетевой поддержкой. Служба ABR была разработана для «эластич-

ной» передачи данных в стиле протокола TCP. При низких нагрузках в сети служба ABR должна была улучшать качество обслуживания, используя свободные ресурсы, а в условиях перегрузки снижать скорости передачи до заранее установленного минимума. Детальное описание механизмов контроля перегрузок в службе ABR сетей ATM и управления трафиком в сетях ATM можно найти в [250].

На рис. 3.45 представлена схема ABR-контроля перегрузки в сетях ATM. Ниже мы будем придерживаться терминологии, принятой в ATM, и употреблять вместо терминов «маршрутизатор» и «пакет» термины «коммутатор» и «ячейка» соответственно. При использовании службы ABR ячейки передаются от отправителя к получателю через последовательность коммутаторов. Среди ячеек с данными содержатся служебные ячейки *управления ресурсами* (Resource Management, RM). RM-ячейки используются для обмена информацией о перегрузках между хостами и коммутаторами. Получатель отсылает RM-ячейки обратно отправителю, иногда внося изменения в их содержимое. Коммутатор может самостоятельно генерировать RM-ячейки и отправлять их нужным хостам. Таким образом, с помощью RM-ячеек возможна организация как прямой обратной связи, так и обратной связи «через получателя».

Рис. 3.45. Основа механизма ABR-контроля перегрузок в сетях ATM

Механизм ABR-контроля перегрузок в сетях ATM основан на величине скорости передачи: передающая сторона явно рассчитывает максимальную скорость, с которой может вестись передача, и при необходимости регулирует ее. У ABR есть три возможности оповестить через коммутатор получателя о наличии перегрузок.

- *Бит EFCI.* Каждая ячейка с данными содержит бит EFCI (Explicit Forward Congestion Indication — явный признак перегрузки). При возникновении перегрузки коммутатор устанавливает этот бит в 1. Хосты назначения проверяют значение бита EFCI всех принимаемых ячеек. Если при получении RM-ячейки хост обнаруживает 1 в бите EFCI последней принятой ячейки с данными, то бит CI (Congestion Indication — признак перегрузки) RM-ячейки также уста-

навливается в 1, а RM-ячейка отсылается передающей стороне. Таким образом, при помощи бита EFCI ячеек с данными и бита CI RM-ячеек осуществляется обратная связь коммутаторов с хостом-отправителем.

- *Биты CIuNI.* Как было показано ранее, RM-ячейки, передаваемые от отправителя получателю, распределены среди множества ячеек с данными. Частота передачи RM-ячеек настраивается; по умолчанию RM-ячейки пересылаются через каждые 32 ячейки с данными. В состав RM-ячеек входят уже знакомый вам бит CI, а также бит N1 (No Increase — запрет увеличения), значения которых могут быть изменены коммутаторами. Обычно 6**HTNI** устанавливается коммутатором в 1 при умеренной перегрузке, а бит CI — при значительной. Получив RM-ячейку, получатель отсылает ее обратно отправителю, при этом значение бита N1 всегда остается прежним, а значение бита CI может быть установлено в 1 в результате проверки бита EFCI.
- *Поле ER.* RM-ячейки содержат двухбайтовое поле ER (Explicit Rate — явная частота). При возникновении перегрузки коммутатор может снизить значение, содержащееся в этом поле. Таким образом, при достижении получателя RM-ячейка будет содержать в поле ER наименьшую из частот передачи, обеспечиваемую коммутаторами пути.

ABR-источники в сетях ATM регулируют скорость передачи данных как функцию значений CI, N1 и ER получаемых RM-ячеек. Правила вычисления скорости передачи весьма сложны и не представляют для нас большой важности; заинтересованный читатель может обратиться к дополнительным источникам информации [250].

Контроль перегрузок в TCP

В этом разделе мы вновь возвращаемся к протоколу TCP. Как вы знаете (см. раздел «Протокол TCP — передача с установлением соединения»), TCP предоставляет службу надежной передачи данных прикладным процессам, выполняющимся на разных хостах. Другим ключевым компонентом протокола TCP является механизм контроля перегрузок. В предыдущем разделе мы отметили, что TCP осуществляет контроль перегрузок самостоятельно, поскольку протокол сетевого уровня IP не имеет механизмов поддержания явной обратной связи между сетевыми устройствами и оконечными системами, которые позволили бы информировать отправителей о наличии перегрузок в сети.

Подход, применяемый в TCP, заключается в ограничении скорости передачи источников в зависимости от наблюдаемой перегрузки. Если перегрузки в сети не наблюдаются, источник может увеличивать скорость передачи; в противном случае скорость передачи принудительно снижается. При подобном подходе возникают три вопроса. Во-первых, каким образом ограничить скорость передачи данных источником? Во-вторых, как источник может определить наличие перегрузки на пути между ним и приемником? В-третьих, каков алгоритм изменения скорости передачи в зависимости от перегрузки? Мы рассмотрим эти вопросы на примере алгоритма Reno, использующегося в современных операционных системах

для контроля перегрузки TCP-соединений [367]. Для определенности будем считать, что источник осуществляет передачу большого файла данных.

Сначала рассмотрим, каким образом в TCP удается ограничивать скорость передачи данных. Как мы узнали из раздела «Протокол TCP — передача с установлением соединения», каждая сторона TCP-соединения состоит из буферов передачи и приема, а также набора переменных (LasttByteRead, RcvWindow и т. д.). Механизм контроля перегрузки использует дополнительную переменную CongWin, называемую *окном перегрузки*. Окно перегрузки влияет на скорость передачи данных в сеть следующим образом: объем неподтвержденных данных, которые может передать источник, не превышает минимального из значений CongWin и RcvWindow, то есть выполняется неравенство

$$\text{LastByteSent} - \text{LastByteAcked} < \min\{\text{CongWin}, \text{RcvWindow}\}.$$

Для того чтобы наглядно представить разницу между контролем перегрузок и контролем потока, предположим, что объем приемного буфера TCP настолько велик, что ограничение, накладываемое окном приема, можно считать несущественным. Таким образом, количество неподтвержденных данных определяется только переменной CongWin.

Ограничение объема неподтвержденных данных косвенно влияет на скорость передачи источника. Представьте себе соединение, в котором отсутствуют потери и задержки пакетов. Если в начале времени оборота источнику разрешается передача CongWin байтов, а по истечении времени оборота он получает квитанции для переданных данных, то скорость его передачи равна отношению CongWin ко времени оборота. Таким образом, меняя значение CongWin, протокол TCP может регулировать скорость передачи данных через соединение.

Теперь рассмотрим, каким образом TCP определяет наличие перегрузок на пути соединения. Введем событие «потеря пакета», наступающее по истечении интервала ожидания или при получении тройного подтверждения от принимающей стороны. При значительных перегрузках в сети буферы одного или нескольких маршрутизаторов переполняются, что приводит к потере дейтаграммы. Потеря дейтаграммы обнаруживается источником при наступлении события «потеря пакета», то есть по истечении интервала ожидания либо при получении трех дублирующих квитанций. Потеря пакета является признаком перегрузок в сети.

Итак, мы приступаем к рассмотрению алгоритма, используемого передающей стороной TCP для регулирование скорости передачи в зависимости от наличия перегрузок в сети. Этот алгоритм обычно называют *алгоритмом контроля перегрузки протокола TCP*. В нем выделяют три основные составляющие: аддитивное увеличение вместе с мультипликативным уменьшением, медленный старт, реакция на истечение интервала ожидания.

Аддитивное увеличение и мультипликативное уменьшение

Основной идеей механизма контроля перегрузки протокола TCP является снижение скорости передачи источника путем уменьшения размера его окна перегрузок при потере пакета. Вполне вероятно, что во всех TCP-соединениях, обслужива-

емых перегруженным маршрутизатором, наблюдаются потери пакетов, что приводит к одновременному уменьшению окон перегрузок всеми этими соединениями. Конечный эффект заключается в снижении трафика, проходящего через перегруженный маршрутизатор и, как следствие, в ослаблении перегрузки. Однако все еще остается открытым вопрос о том, насколько существенным должно быть снижение скорости передачи при потере пакета. В протоколе TCP используется так называемое «мультипликативное уменьшение», означающее двойное уменьшение размера окна перегрузки при потере пакета. Если потеря пакета произошла при значении CongWin, равном 20 Кбайт, то последнее будет «урезано» до 10 Кбайт. В случае потери еще одного пакета значение CongWin станет равным 5 Кбайт. Уменьшение размера окна перегрузки может происходить многократно, однако значения CongWin, меньшие максимального размера сегмента (MSS), не допускаются. Заметим, что приведенное выше описание является упрощенным, и изменение размера окна перегрузки на самом деле представляет собой несколько более сложный процесс. Как мы увидим чуть позже, при потере пакета значение CongWin сначала становится равным MSS и лишь затем достигает половины исходного значения.

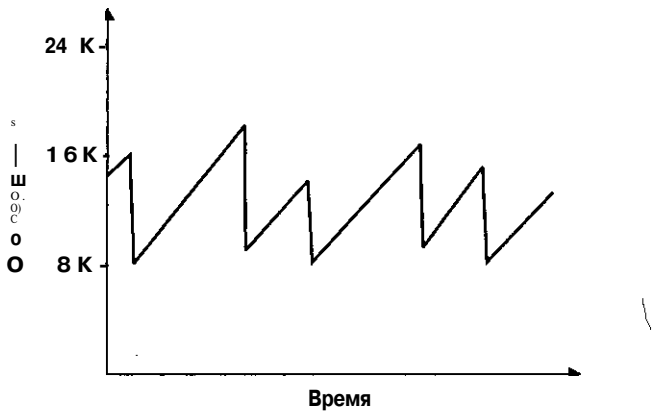


Рис. 3.46. Контроль перегрузки с аддитивным увеличением и мультипликативным уменьшением

Вопрос снижения скорости передачи протоколом TCP при наличии перегрузок вполне естественно продолжить рассмотрением механизма увеличения скорости передачи при отсутствии перегрузок. Отсутствие перегрузок, или потерь пакетов, указывает на вероятность присутствия на линии связи неиспользуемых ресурсов и является побудительным мотивом для увеличения скорости передачи источника. Нарастание скорости происходит медленно и плавно; протокол TCP «прощупывает» свободные ресурсы на пути соединения. Каждый раз при получении квитанции значение CongWin немного увеличивается; не вдаваясь в детали алгоритма, можно сказать, что TCP каждый раз увеличивает значение CongWin на величину MSS, если за время оборота не наблюдалось потерь пакетов. Таким образом, TCP осуществляет аддитивное увеличение скорости передачи при отсутствии перегру-

зок в сети и мультипликативное уменьшение скорости передачи при наличии перегрузок. Алгоритм контроля перегрузок протокола TCP часто называют алгоритмом *AIMD* (Additive-Increase, Multiplicative-Decrease — аддитивное увеличение и мультипликативное уменьшение). Фаза линейного нарастания потока данных в протоколе контроля перегрузки называется *уходом от перегрузки.* Величина CongWin циклически проходит через стадии линейного нарастания и резкого спада до половины текущего значения при потере пакета. График CongWin для TCP-соединений со значительным временем жизни имеет вид, напоминающий зубья пилы, и представлен на рис. 3.46.

Медленный старт

При установлении TCP-соединения начальным значением переменной CongWin является величина MSS; следовательно, начальная скорость передачи источника составляет MSS/RTT, где RTT — время оборота для соединения. Например, если MSS = 500 байт, а RTT = 200 мс, то начальная скорость передачи соединения равна приблизительно 20 Кбайт. Поскольку максимально возможная скорость передачи значительно превосходит величину MSS/RTT, линейное увеличение начальной скорости нерационально, так как этот процесс тянется слишком долго. Для решения проблемы на начальном этапе вместо линейного увеличения используется экспоненциальное увеличение, то есть значение CongWin возрастает вдвое после каждого истечения времени оборота. Экспоненциальный рост продолжается до первой потери пакета, после чего значение CongWin уменьшается вдвое и в дальнейшем увеличивается по линейному закону. Итак, в первой фазе, называемой *медленным стартом,* источник начинает передачу с низкой скоростью, которая растет по экспоненциальному закону. Подобный рост обеспечивается следующим образом. Сначала источник отсылает первый сегмент; если не происходит его потери, то при получении квитанции значение CongWin увеличивается на величину MSS. Это позволяет передать во втором периоде не один, а два сегмента максимального размера; при получении квитанций для каждого из этих сегментов значение CongWin вновь увеличивается на MSS. Таким образом, на третьем «проходе» источник может осуществить передачу не двух, а четырех сегментов, и т. д. Каждая получаемая квитанция увеличивает CongWin на значение MSS. Эта процедура продолжается до первой потери сегмента. Таким образом, в фазе медленного старта происходит быстрое нарастание скорости передачи за счет изменения ширины окна перегрузок.

Реакция на истечение интервала ожидания

Согласно предыдущему описанию, размер окна перегрузок протокола TCP, будучи изначально равным величине MSS, экспоненциально растет до первой потери пакета, после чего в силу вступает алгоритм аддитивного увеличения и мультипликативного уменьшения. Эта картина уже весьма близка к реальной, однако и она останется неполной, если мы не раскроем еще одну деталь механизма контроля перегрузок TCP: оказывается, по истечении интервала ожидания протоколом предпринимаются иные действия, нежели при получении трех дублирующих квитанций. В последнем случае поведение TCP соответствует описанному выше: раз-

мер окна перегрузки уменьшается вдвое, а затем начинает линейно расти. Однако по истечении интервала ожидания передающая сторона TCP входит в состояние медленного старта, уменьшая размер окна перегрузки до величины MSS и далее позволяя ему увеличиваться по экспоненциальному закону до тех пор, пока он не достигнет половины своего значения до истечения интервала ожидания. После этого рост значения **CongWin** вновь становится линейным.

Для реализации этого усложненного механизма TCP поддерживает специальную переменную **Threshold** (пороговое значение), определяющую размер окна перегрузки, при котором происходит окончание фазы медленного старта и начинается фаза ухода от перегрузки. Обычно по умолчанию для переменной **Threshold** используется большое значение (чаще всего 65 Кбайт [484]), чтобы минимизировать влияние этой переменной в начале передачи. При потере пакета значение **Threshold** становится равным половине текущего значения **CongWin.** Например, если перед потерей пакета размер окна перегрузки составлял 20 Кбайт, то переменная **Threshold** окажется равной 10 Кбайт и сохранит это значение до следующей потери пакета.

Зная о существовании переменной **Threshold,** мы можем уточнить описание механизма управления значением переменной **CongWin** после истечения интервала ожидания. Как показано выше, передающая сторона TCP входит в состояние медленного старта. Величина **CongWin** экспоненциально увеличивается до тех пор, пока не достигнет значения **Threshold.** После этого TCP вступает в фазу ухода от перегрузки, где **CongWin** увеличивается линейно.

Резюмируя изложенные выше идеи, алгоритм контроля перегрузки протокола TCP можно представить следующим образом.

- Когда размер окна перегрузки не превышает пороговую величину **(Threshold),** передающая сторона находится в фазе медленного старта, и размер окна перегрузки растет экспоненциально.
- Когда размер окна перегрузки превышает пороговую величину, Передающая сторона находится в фазе ухода от перегрузки, и размер окна перегрузки растет линейно.
- При получении трех дублирующих квитанций пороговая величина устанавливается равной половине текущего значения размера окна перегрузки, а размер окна перегрузки становится равным пороговому значению.
- По истечении интервала ожидания пороговая величина устанавливается равной половине текущего значения размера окна перегрузки, а размер окна перегрузки становится равным величине MSS.

Вызывает интерес вопрос о причине разных моделей поведения протокола TCP при истечении интервала ожидания и при получении трех дублирующих квитанций. Другими словами, чем обусловлена «осторожность» TCP, заставляющая его снижать размер окна перегрузки до минимального значения по истечении интервала ожидания, и почему получение трех дублирующих квитанций вызывает «урезание» окна перегрузки лишь вдвое? Ранняя версия TCP, *TCP Такое,* в обоих случаях устанавливала размер окна перегрузки равным величине MSS и входила в фазу медленного старта. Более поздняя версия, *TCP Reno,* предусматривает выход из

фазы медленного старта при получении трех дублирующих квитанций. Логика этой модификации заключается в том, что наличие трех квитанций указывает не только на потерю одного сегмента, но и на успешную доставку трех следующих за ним сегментов. Следовательно, в отличие от ситуации с истечением интервала ожидания, сеть способна успешно доставлять, по крайней мере, часть передаваемых сегментов. «Досрочный» выход из фазы медленного старта при получении трех дублирующих квитанций называется *ускоренным восстановлением*.

На рис. 3.47 показаны графики изменения размера окна перегрузки для версий Reno и Tahoe протокола TCP. В рассматриваемом случае начальное пороговое значение составляет 8 x MSS. Размер окна перегрузки экспоненциально растет и достигает порогового значения в четвертой фазе передачи. Затем размер окна увеличивается линейно до тех пор, пока передающая сторона не получит трех дублирующих квитанций в конце восьмой фазы; при этом размер окна составляет 12 x MSS. Пороговое значение устанавливается равным 0,5 x 12 x MSS = 6 x MSS. В протоколе TCP Reno новый размер окна также становится равным 6 x MSS и далее увеличивается по линейному закону. TCP Tahoe, напротив, устанавливает размер окна равным 1 x MSS и экспоненциально наращивает его до тех пор, пока он не превысит порогового значения [10, 245,484].

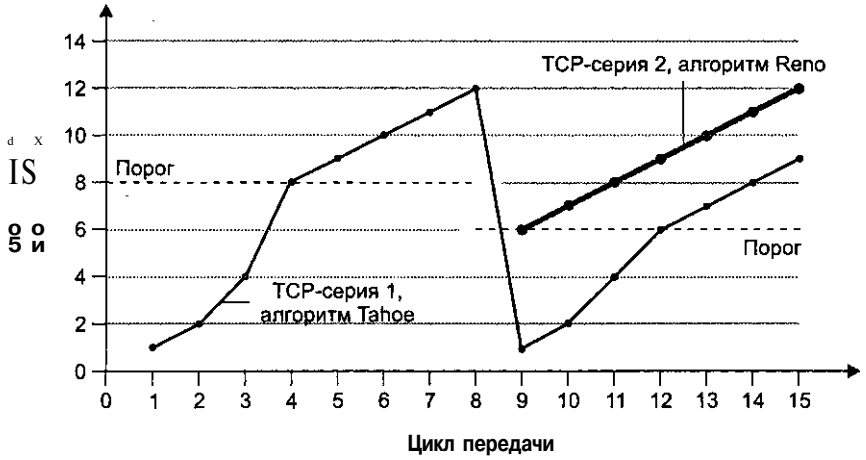


Рис. 3.47. Окна перегрузки в протоколах TCP Tahoe и TCP Reno

Как было показано ранее, в большинстве современных реализаций TCP используется алгоритм Reno. Существует множество модификаций этого алгоритма (RFC 2018, RFC 2582). Так, например, в алгоритме Вегаса [8, 43] предпринимается попытка избежать перегрузок и одновременно обеспечить высокую скорость передачи. Основными идеями алгоритма Вегаса являются обнаружение перегрузок в маршрутизаторах на пути соединения *до* появления потерь пакетов и линейное снижение скорости передачи при перегрузках. Прогнозирование перегрузок осуществляется путем анализа времени оборота: чем больше это время, тем более значительной является перегрузка в маршрутизаторах. Читателям, интересующимся последни-

ми разработками в области контроля перегрузок TCP, рекомендуем обратиться к дополнительным источникам информации [152].

Необходимо отметить, что контроль перегрузок протокола TCP эволюционировал много лет и процесс его развития не завершен. Механизмы, использовавшиеся в период, когда большинство TCP-соединений поддерживало протоколы SMTP, FTP и Telnet, являются не столь эффективными в современном Интернете, где доминирующее положение занимает HTTP. Службы, которые будут созданы в будущем, также потребуют новых механизмов контроля перегрузки.

Макроскопическая производительность TCP-соединения

Мы знаем, что регулирование скорости передачи протоколом TCP приводит к графику, напоминающему зубья пилы (см. рис. 3.46). Каково среднее значение скорости передачи TCP-соединения с длительным временем жизни? Мы попытаемся ответить на этот вопрос, исключив из рассмотрения фазы медленного старта (как правило, они являются очень короткими, поскольку передающая сторона экспоненциально наращивает скорость передачи). Скорость передачи во время оборота определяется текущим размером окна перегрузки и длительностью интервала. Ранее мы упоминали, что если размер окна составляет *w* байтов, а время оборота равно величине RTT, то скорость передачи источника определяется как *w/KYT*. TCP увеличивает скорость передачи на величину MSS каждый раз по истечении времени оборота до тех пор, пока не произойдет потеря пакета. Пусть *W* — размер окна в момент обнаружения факта потери пакета; учитывая, что значения *Wu* RTT почти не меняются для конкретного соединения, скорость передачи источника находится в интервале от *W/(2* x RTT) до W/RTT.

Приведенные соображения значительно упрощают макроскопическую модель устойчивого TCP-соединения. Потеря пакета происходит, когда скорость передачи приближается к W/RTT; затем скорость передачи снижается вдвое и начинает линейно увеличиваться на величину MSS/RTT каждый раз по истечени^ времени оборота до тех пор, пока вновь не приблизится к W/RTT. Этот процесс повторяется циклически. Поскольку производительность (то есть скорость передачи) TCP-соединения линейно растет между двумя граничными значениями, можно сделать вывод о том, что средняя производительность соединения составляет (0,75 x *W)/* RTT.

Используя такую идеализированную модель изменения производительности TCP-соединения, мы можем вывести интересное соотношение, связывающее частоту потерь пакетов и доступную скорость передачи [305]. Этому вопросу посвящено одно из заданий в конце главы. Существует также и более сложная эмпирическая модель, описывающая приведенные данные [366].

Выравнивание скоростей передачи

Предположим, что *K* TCP-соединений между разными хостами используют одну и ту же проблемную линию связи с пропускной способностью *R* бит/с (говоря «проблемная линия», мы имеем в виду, что в каждом соединении все остальные линии связи не перегружены и скорость передачи по ним значительно ближе к их про-

пускной способности, чем по проблемной линии). Предположим, что по каждому из соединений передается файл большого размера и в проблемной линии связи отсутствует UDP-трафик. Говорят, что механизм контроля перегрузки обеспечивает *выравнивание скоростей,* если средняя скорость передачи каждого соединения составляет приблизительно *R/K;* другими словами, пропускная способность линии связи делится между всеми использующими ее соединениями поровну.

Так как TCP-соединения могут устанавливаться в разные моменты времени и тем самым иметь неравные размеры окон перегрузки, возникает вопрос: обеспечивает ли выравнивание скоростей алгоритм аддитивного увеличения и мультипликативного уменьшения протокола TCP? В [73] наглядно показано, каким образом TCP обеспечивает выравнивание скоростей передачи для всех соединений, использующих проблемную линию связи.

Рассмотрим простейший случай, когда два TCP-соединения используют общую линию связи с пропускной способностью *R,* как показано на рис. 3.48. Предположим, что оба соединения имеют одни и те же значения MSS и RTT (то есть если размеры окон перегрузки этих соединений равны, то скорости передачи у них также одинаковы), что передается большой объем данных и что при этом общая линия связи не используется другими TCP- или UDP-соединениями. Кроме того, мы исключаем из рассмотрения фазу медленного старта, считая, что оба соединения постоянно находятся в фазе ухода от перегрузки (работает алгоритм аддитивного увеличения и мультипликативного уменьшения).

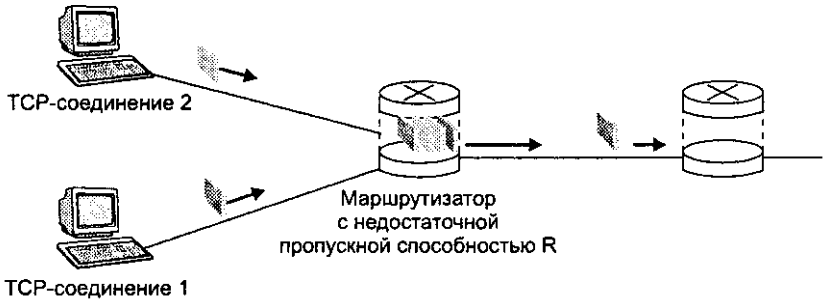


Рис. 3.48. Два TCP-соединения, использующие общую проблемную линию связи

На рис. 3.49 представлены графики соотношений пропускной способности двух TCP-соединений. Если оба соединения работают с одной и той же скоростью передачи, то соответствующая точка графика будет лежать на прямой, составляющей угол 45° с осями координат (линия равных скоростей). С точки зрения полноты использования ресурсов линии связи идеальной является ситуация, когда суммарная скорость передачи двух соединений составляет *R* (разумеется, нет смысла выравнивать скорости передачи для каждого соединения, если в результате они окажутся нулевыми!). Таким образом, цель алгоритма контроля перегрузки — добиться, чтобы на графике пропускная способность каждого из соединений оказалась как можно ближе к точке пересечения линии равных скоростей с линией максимального использования ресурсов.

Рис. 3.49. Производительность TCP-соединений 1 и 2

Предположим, что размеры окон перегрузки двух соединений таковы, что в заданный момент времени соотношение значений пропускной способности соединений 1 и 2 изображается точкой A. Поскольку суммарная скорость передачи оказывается меньше *R,* потерь пакетов не наблюдается, и, следовательно, оба соединения в соответствии с алгоритмом ухода от перегрузки увеличивают скорости передачи на величину MSS каждый раз по истечении времени оборота. Таким образом, соотношение значений пропускной способности соединений движется от точки A вдоль линии равных скоростей. В определенный момент времени увеличение скоростей передачи приводит к тому, что суммарная скорость превышает *R* и происходит потеря пакета. Пусть в этот момент времени значение пропускной способности соединений изображается точкой B. Оба соединения снижают размер окна перегрузки вдвое, в результате значение пропускной способности перемещается в точку C, лежащую на середине вектора, соединяющего точку B с на/чалом координат. Поскольку в точке C суммарная скорость передачи оказывается меньше *R,* соединения вновь начинают увеличивать свои скорости передачи, перемещая точку значения пропускной способности из положения C вдоль линии равных скоростей. Потеря пакета в точке D ведет к тому, что оба соединения снова уменьшают окна перегрузки вдвое, и т. д. Обратите внимание на то, что точка, изображающая значение пропускной способности двух соединений, в конечном счете приходит в движение вдоль линии равных скоростей, причем подобное поведение наблюдается независимо от того, где ее исходное положение. Хотя в основе описанной ситуации лежат некоторые идеализирующие ее допущения, она достаточно наглядно демонстрирует механизм выравнивания пропускной способности линии между TCP-соединениями.

Мы предположили, что проблемная линия связи не используется другими соединениями, кроме TCP, время оборота для всех соединений одинаково и между каждой парой хостов установлено единственное TCP-соединение. На практике

подобные допущения не выполняются, и приложения клиент/сервер зачастую используют неравные доли пропускной способности линии связи. В частности, сеансы с меньшими значениями времени оборота способны захватывать большую часть свободных ресурсов вследствие более быстрого наращивания размеров окна перегрузки по сравнению с другими соединениями [288].

Выравнивание скоростей и UDP

Выше мы показали, каким образом механизм контроля перегрузки TCP управляет скоростью передачи источника, изменяя размер окна перегрузки. Контролирование перегрузки заставляет многие мультимедиа-приложения (Интернет-телефонию и видеоконференции) отказываться от протокола TCP: снижение скорости передачи для них нежелательно даже при значительных перегрузках в сети. Вместо TCP эти приложения пользуются службами протокола UDP, не имеющего собственного механизма контроля перегрузки. Протокол UDP позволяет приложениям передавать данные с любой нужной им скоростью, не ограничивая их равными долями пропускной способности и допуская потери пакетов при наступлении перегрузок. С позиций TCP мультимедиа-приложения не обеспечивают выравнивание скоростей, поскольку не взаимодействуют с другими соединениями и не регулируют свои скорости передачи. Одной из главных задач, стоящих перед современными исследователями в области Интернет-технологий, является создание механизмов контроля перегрузки, ограждающих пропускную способность Интернета от «губительного» воздействия UDP-соединений [151, 156].

Выравнивание скоростей и параллельные TCP-соединения

Даже если механизм, заставляющий UDP-соединения выравнивать пропускные способности линий связи, будет создан, он не решит существующую проблему до конца. Это объясняется тем, что приложениям невозможно запретить использовать параллельные TCP-соединения. К примеру, параллельные TCP-соединения применяются web-браузерами для одновременной доставки нескольких объектов web-страницы (число соединений, одновременно поддерживаемых браузером, обычно задается при его настройке). Увеличение числа соединений, устанавливаемых приложением, увеличивает долю пропускной способности, «захватываемую» для передачи данных этого приложения. Представьте линию связи с пропускной способностью *R,* обслуживающую одновременно 9 приложений клиент/сервер, каждое из которых устанавливает одно TCP-соединение. В случае появления в линии связи еще одного соединения все соединения получат возможность передавать данные со скоростью приблизительно *R/10.* Однако если эта линия станет использоваться приложением, установившим И параллельных TCP-соединений, то окажется, что оно будет передавать свои данные со скоростью, превышающей *R/21* Поскольку web-трафик составляет значительную часть общего трафика Интернета, подобные ситуации, к сожалению, нередки.

Модель задержек протокола TCP

Мы завершаем эту главу рассмотрением нескольких простых моделей, позволяющих вычислить время, необходимое TCP для передачи объекта (рисунка, текстового файла или музыкального произведения в формате MP3). Для каждого объекта мы введем понятие задержки — промежутка времени, проходящего с момента инициирования TCP-соединения клиентской стороной до полного завершения процесса приема объекта получателем. В моделях, которые будут описаны ниже, учитываются ключевые составляющие задержки: процедура рукопожатия, медленный старт, время передачи объекта.

Наш простой анализ основывается на предположении, что сеть не перегружена, то есть TCP-соединение, с помощью которого осуществляется передача объекта, не конкурирует за ресурсы линий связи с другими TCP- или UDP-соединениями. Кроме того, чтобы не описывать множество деталей, мы будем считать, что передающая и принимающая стороны соединены единственной линией связи, как показано на рис. 3.50. (Данная линия связи является единственной проблемной линией на пути соединения. В упражнениях, приведенных в конце главы, рассматривается вопрос обобщения ситуации для пути, состоящего из нескольких линий связи.) В моделях также используются следующие допущения:

- объем данных, пересылаемых источником, ограничивается только размером окна перегрузки (объем входных буферов достаточно велик);
- пакеты не теряются и не искажаются, то есть необходимости в повторных передачах нет;
- размеры заголовков протоколов TCP, IP и др. пренебрежимо малы и не учитываются при построении моделей;
- передаваемый объект (файл) состоит из целого числа сегментов максимального размера (MSS);
- время передачи учитывается только для сегментов MSS, а для^зйпросов, квитанций и сегментов, посылаемых при установлении TCP-соединения, время передачи считается пренебрежимо малым;
- начальное пороговое значение в механизме контроля перегрузок TCP настолько велико, что размер окна перегрузки никогда не достигает его.

Для создания моделей мы вводим следующие обозначения:

- O бит — размер передаваемого объекта;
- S бит — максимальный размер сегмента (MSS);
- R бит/с — пропускная способность линии связи, соединяющей источник и приемник.

Перед тем как перейти к написанию формул, давайте оценим величину задержки объекта, исключив из рассмотрения окно перегрузки, то есть позволив серверу передать объект целиком. Заметим, что инициирование TCP-соединения занимает все время оборота; на втором обороте клиент отсылает серверу запрос на получение объекта, вложенный в последний сегмент тройного рукопожатия. Таким

образом, прием объекта клиентом начинается по истечении удвоенного времени оборота. Время приема составляет *O/R;* следовательно, суммарная задержка равна 2 x RTT + *O/R.* Мы составили выражение, описывающее нижнюю границу суммарной задержки; очевидно, что процедура медленного старта, присутствующая в протоколе TCP, увеличивает время передачи объекта.

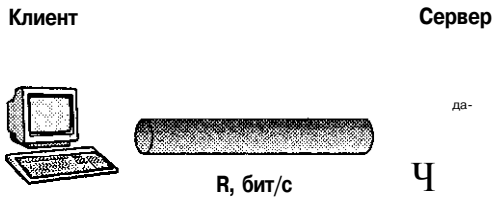


Рис. 3.50. Простая сеть, напрямую связывающая клиент с сервером

Статическое окно перегрузки

Несмотря на то что на практике размер окна перегрузки протокола TCP постоянно меняется, для наглядности удобно сначала представить окно перегрузки статическим. Пусть *W* — положительное целое число, равное размеру статического окна; это означает, что сервер, не ожидая подтверждений, может передать не более *W* сегментов. Получив запрос, сервер сразу отсылает клиенту W сегментов. Далее сервер пересылает каждый новый сегмент после получения одной квитанции от клиента; процесс продолжается до завершения передачи объекта. Возможны два случая.

- *WS/R* > RTT + *S/R.* Это соотношение означает, что сервер получает квитанцию для первого сегмента первого окна до завершения передачи сегментов первого окна.
- *WS/R* < RTT + *S/R.* Это соотношение означает, что сервер заканчивает передачу сегментов окна раньше, чем получает квитанцию для первого сегмента окна.

Рассмотрим первый из случаев (рис. 3.51). Размер окна положен равным четырем сегментам: *W*= 4. Инициирование TCP-соединения занимает все время оборота; на втором обороте клиент отсылает серверу запрос на получение объекта, вложенный в последний сегмент тройного рукопожатия. Таким образом, прием объекта клиентом начинается по истечении удвоенного времени. Новые сегменты поступают к клиенту с периодичностью в *S/R* с и подтверждаются им. Поскольку сервер получает первое подтверждение до завершения передачи сегментов окна, он имеет возможность передавать сегменты следующего окна сразу по завершении передачи сегментов предыдущего окна. Все подтверждения поступают на сервер с периодичностью *S/R* с, поэтому сервер передает новые сегменты без простоев в ожидании подтверждений. Таким образом, если передача объекта начата со скоростью *R,* то такая скорость сохраняется на протяжении всего процесса передачи, а следовательно, задержка составляет

$$2RTT + O/R.$$

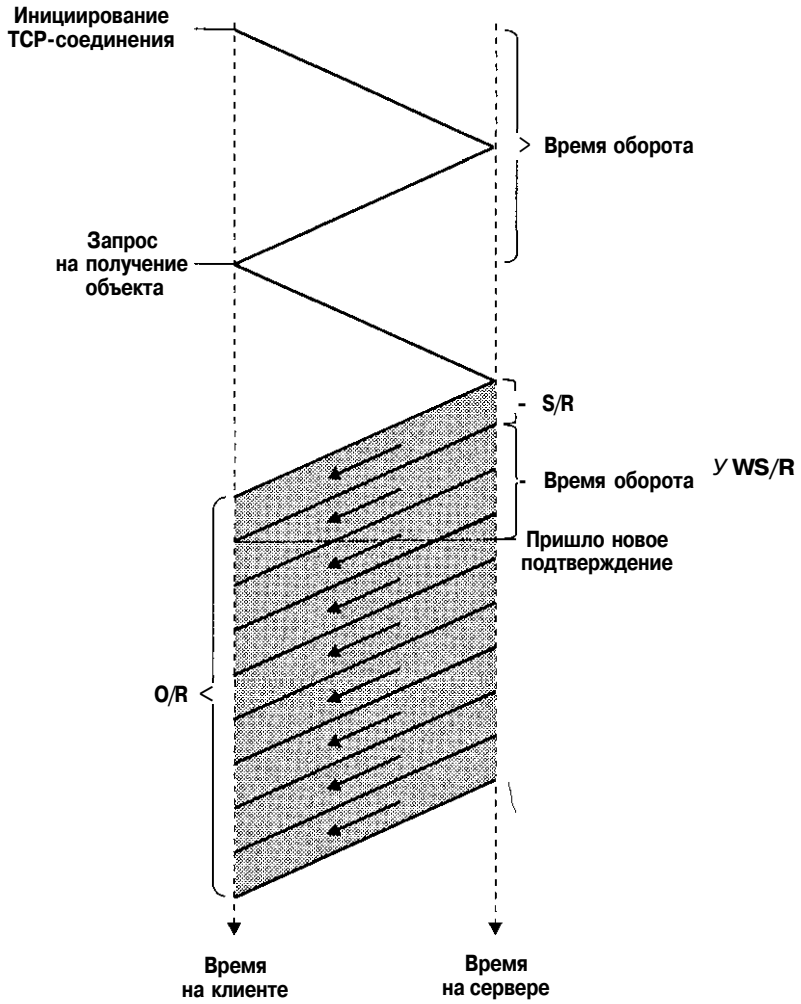


Рис. 3.51. Схема для случая WS/R > RTT + S/R

Теперь рассмотрим второй случай (рис. 3.52). Предположим, что размер окна Нравен 2 сегментам. Как и ранее, прием объекта клиентом начинается по истечении двойного времени оборота. Новые сегменты поступают к клиенту с периодичностью в *S/R* с, однако передача сегментов окна завершается до того, как подтверждение для первого сегмента оказывается на сервере. Поскольку для продолжения передачи сервер должен получить подтверждение для первого сегмента окна, он вынужден провести некоторое время в его ожидании. Подтверждения поступают на сервер с пе-

риодичностью в *S/R* с, и при получении каждого подтверждения сервер отсылает клиенту очередной сегмент. Таким образом, сервер всегда находится в одном из двух состояний: передачи ^сегментов или простоя; суммарная задержка, следовательно, складывается из величины 2RTT, времени передачи объекта и времени простоя сервера. Для определения последнего введем величину *K,* равную числу окон, необходимых для передачи объекта. *K = O/WS;* если результат деления оказывается не целым, он округляется в большую сторону до ближайшего целого. В процессе передачи сервер входит в состояние простоя *K* - 1 раз, при этом время» простоя составляет RTT - (*W- 1)S/R.* Итак, в рассматриваемом случае задержка равна

$$2RTT + O/R + (K - 1) \times [S/R + RTT - WS/R\}.$$

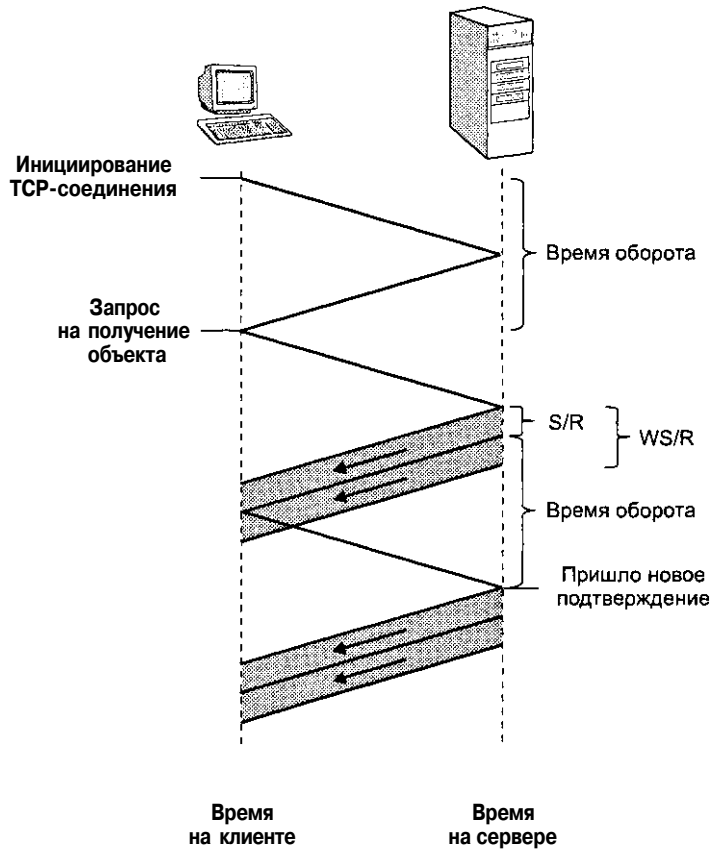


Рис. 3.52. Схема для случая WS/R < RTT + S/R

Объединяя два случая, получаем следующую задержку:

$$2RTT + O/R + (K - 1) \times [S/R + RTT - WS/R]\backslash$$

где $[x]^{>=} = \max(x,0)$. Обратите внимание, что задержка складывается из трех составляющих: 2 x RTT, соответствующей установлению соединения, *O/R,* равной вре-

мени передачи объекта, и $(K - 1)\ [S/R + \text{RTT} - WS/R]^{+}$, представляющей время простоя сервера.

На этом мы завершаем рассмотрение статического окна перегрузки. Анализ динамически изменяющегося окна является более сложным, однако имеет много общего с анализом статического окна.

Динамическое окно перегрузки

Изменим предыдущую модель, позволив окну перегрузки изменять свой размер. Как мы говорили ранее, в начале передачи сервер устанавливает размер окна перегрузки равным величине MSS и отсылает один сегмент клиенту. Получив квитанцию, сервер увеличивает размер окна перегрузки до величины 2MSS и посылает 2 сегмента с интервалом в S/R с. Получив еще 2 квитанции, сервер увеличивает размер окна перегрузки до величины 4MSS и отсылает клиенту 4 сегмента также с интервалом в S/R с. Таким образом, каждый раз по истечении времени оборота окно перегрузки увеличивается вдвое, как показывает временная диаграмма на рис. 3.53.

Обратите внимание, что O/S представляет собой число сегментов объекта; на приведенной выше диаграмме $O/S = 15$. Далее, рассмотрим закон изменения размеров окна перегрузки. Первое окно содержит 1 сегмент, второе окно — 2 сегмента, третье окно — 4 сегмента, и т. д. Обобщая закономерность, получаем, что k-е окно содержит 2^{k-1} сегментов. Пусть K — число окон, которые «покрывают» объект; на предыдущей диаграмме $K = 4$. Отношение между величинами K и O/S можно выразить следующим образом:

$$K = \min\left\{k : 2^0 + 2^1 + \ldots + 2^{k-1} \geq \frac{O}{S}\right\} =$$

$$= \min\left\{k : 2^k - 1 \geq \frac{O}{S}\right\} =$$

$$= \min\left\{k : k \geq \log_2\left(\frac{O}{S} + 1\right)\right\} =$$

$$= \text{bg}2 \quad - + 1$$

После передачи данных окна возможны простои сервера, обусловленные ожиданием получения квитанций. На рис. 3.53 показано, что сервер простаивает после передачи первого и второго окон, а третье окно не вызывает простоя. Подсчитаем суммарное время простоя после передачи k-mo окна. Время, проходящее с момента начала передачи k-mo окна до получения подтверждения для его первого сегмента, составляет $S/R + \text{RTT}$; время передачи k-mo окна равно $\{S/R)\ 2^{M}$. Таким образом, время простоя, равное разности указанных величин, составляет

$$[S/R + \text{RTT} - \mathbf{2^{w}}\ (S/R)]\backslash$$

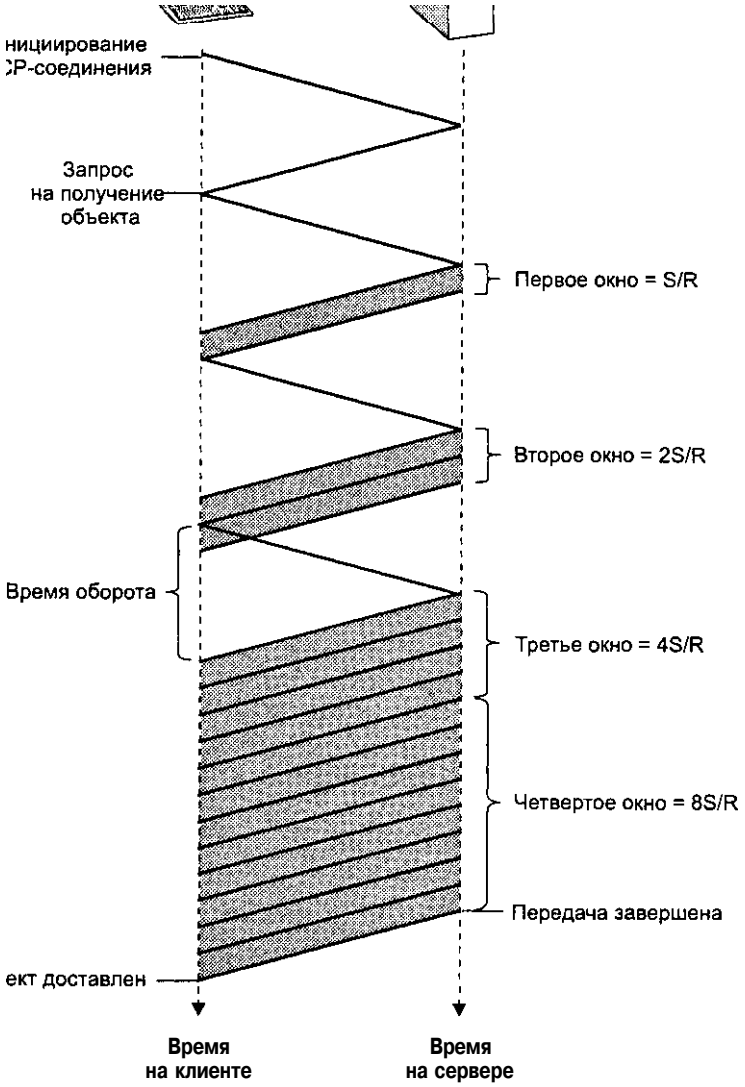


Рис. 3.53. Временная диаграмма медленного старта TCP

Простой сервера возможен после передачи любого из первых $K - 1$ сегментов, поскольку K-й сегмент является последним. Теперь мы можем подсчитать полную задержку передачи файла. В нее входят три составляющие: 2RTT для установления TCP-соединения и запроса файла, O/K для передачи объекта и сумма всех простоев сервера. Таким образом, задержка составляет

$$2Д7Т + \hat{} +] Г \qquad R \qquad R$$

Сравнив полученную формулу с формулой задержки для статического окна перегрузки, можно увидеть, что единственным отличием первой от второй является член $2^{k}\backslash S/R)$ вместо WS/R. Для упрощения выражения введем величину Q, равную числу простоев сервера в предположении, что передаваемый объект состоит из бесконечного числа сегментов. Проводя преобразования, аналогичные тем, которые были выполнены для величины K, получим

$$Q = \log_2 \left(1 + \frac{RTT}{S/R}\right) + 1.$$

Фактическое число простоев сервера при передаче составляет $P = mm\{Q, K - 1\}$. На рис. 3.53 $P = Q = 2$. С учетом принятых обозначений формула задержки преобразуется к следующему виду (см. упражнения, приведенные в конце главы):

$$2RTT + \frac{O}{R} + P\left(RTT + \frac{S}{R}\right)$$

Таким образом, для вычисления задержки необходимо получить значения K и Q, присвоить величине P минимальное из значений Q и $K - 1$ и подставить его в приведенную формулу.

Сравним задержку протокола TCP с задержкой, которая имела бы место при отсутствии механизма контроля перегрузки (то есть при отсутствии ограничений, налагаемых окном перегрузки). Как мы определили ранее, минимальное значение задержки в случае контроля перегрузки составляет $2RTT + O/R$. Нетрудно показать, что

$$\frac{\text{Задержка}}{\text{Минимальная задержка}} < 1 + [(0/Я)/Я7Т] + 2$$

Из приведенной формулы видно, что фаза медленного старта не вносит ощутимого вклада в задержку, если $RTT \ll O/R$, то есть при длительности передачи, значительно превышающей время оборота.

Рассмотрим несколько количественных примеров. Положим, что величина 5 составляет 536 байт — значение, обычно применяемое в TCP по умолчанию. Время оборота возьмем равным 100 мс, что не является редкостью для континентальных или межконтинентальных линий связи, испытывающих перегрузку средней силы. Для начала рассмотрим передачу объекта размером $O = 100$ Кбайт. Число окон K, «покрывающих» объект, равно 8. Таблица 3.4 демонстрирует влияние медленного старта на задержку для нескольких значений скорости передачи.

Как видно из таблицы, для объекта большого размера медленный старт вносит значительную задержку только в случае высокой скорости передачи. Если скорость передачи невысока, то подтверждения поступают относительно скоро,

и TCP за короткое время достигает максимальной скорости передачи. Например, если R = 100 Кбайт/с, число простоев составляет P= 2, в то время как количество окон K равно 8. Это означает, что сервер простаивает лишь после передачи первых двух из восьми окон. Вместе с тем если R = 10 Мбит/с, то простои наблюдаются после передачи каждого пакета и значительно увеличивают общую задержку.

Таблица 3.4. Влияние медленного старта на задержку (RTT= 100 мс, 0= 100 Кбайт, K = 8)

R	O/R	P	Минимальная задержка	Задержка при медленном старте
28 Кбит/с	28,6 с	1	28,8 с	28,9 с
100 Кбит/с	8с	2	8,2 с	8,4 с
1 Мбит/с	800 мс	5	1 с	1,5с
10 Мбит/с	80 мс	7	0,28 с	0,98 с

Теперь рассмотрим передачу небольшого объекта размером 0 = 5 Кбайт. Число окон, покрывающих объект, составляет K = 4. Таблица 3.5 демонстрирует влияние медленного старта на задержку для нескольких скоростей передачи.

Таблица 3.5. Влияние медленного старта на задержку (RTT= 100 мс, 0 = 5 Кбайт, K = 4)

R	O/R	P	Минимальная задержка	Задержка при медленном старте
28 Кбит/с	1,43 с	1	1,63 с	1,73 с
100 Кбит/с	0,4 с	2	0,6 с	0,76 с
1 Мбит/с	40 мс	3	0,24 с	0,52 с
10 Мбит/с	4 мс	3	0,20 с	0,50 с

Как и в предыдущем случае, медленный старт значительно повлиял на задержку только для больших скоростей передачи. Например, когда R = 1 Мбит/с, сервер простаивает после передачи каждого окна, что обусловливает значение задержки, почти вдвое превосходящее минимальное.

С возрастанием величины RTT влияние медленного старта для небольших объектов становится заметным и при небольших скоростях передачи. Таблица 3.6 демонстрирует влияние медленного старта на задержку при RTT = 1 с и O =5 Кбайт (K = 4).

Таблица 3.6. Влияние медленного старта на задержку (RTT = 1 с, O = 5 Кбайт, K = 4)

R	O/R	P	Минимальная задержка	Задержка при медленном старте
28 Кбит/с	1,43 с	3	3,4 с	5,8 с
100 Кбит/с	0,4 с	3	2,4 с	5,2 с
1 Мбит/с	40 мс	3	2,0с	5,0 с
10 Мбит/с	4 мс	3	2,0 с	5,0 с

Суммируя полученные результаты, можно сделать вывод о том, что медленный старт значительно увеличивает общую задержку в случаях, когда размер объекта относительно мал, а время оборота относительно велико. К сожалению, в web подобные ситуации не являются редкостью.

Расчет времени ответа для протокола HTTP

В качестве примера практического применения методов анализа задержки рассчитаем время ответа для web-страницы, передаваемой по протоколу HTTP, не поддерживающему постоянные соединения. Предположим, что страница состоит из базового HTML-файла и *M*рисунков. Для простоты выкладок примем, что размеры всех $M + 1$ объектов одинаковы и составляют O бит.

В случае непостоянного HTTP-соединения объекты пересылаются последовательно, один за другим. Таким образом, время ответа равно сумме задержек для всех объектов:

$$(M+1)\left\{2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - \left(2^P - 1\right)\frac{S}{R}\right\}.$$

Обратите внимание на то, что указанная формула имеет следующую структуру: время ответа = *(M+1)0/R* + 2 (M+1) RTT + суммарная задержка медленного старта для M + 1 объектов. Очевидно, что если число объектов web-страницы велико и время оборота значительно, то производительность непостоянного HTTP-соединения окажется небольшой. В упражнениях, приведенных в конце главы, предлагается рассчитать время ответа для других схем передачи протокола HTTP. Читателю, заинтересованному подобным анализом, рекомендуем обратиться к дополнительным источникам информации [53, 200].

Резюме

Мы начали эту главу с рассмотрения служб, которые транспортный уровень может предоставить сетевым приложениям. С одной стороны, протоколы транспортного уровня могут быть очень простыми и выполнять только функции мультиплексирования и демультиплексирования. Примером служит протокол UDP, обладающий минимальным набором служб. С другой стороны, протоколы транспортного уровня обеспечивают приложениям различные гарантии: надежную передачу данных, ограниченное время доставки данных и минимальную скорость передачи. Стоит отметить, что услуги, предоставляемые протоколами транспортного уровня, в значительной степени зависят от модели обслуживания используемого протокола сетевого уровня. Если сетевой уровень не обеспечивает доставку данных за определенное время или с заданной минимальной скоростью, транспортный уровень не может своими средствами предоставить приложениям соответствующие гарантии.

В разделе «Принципы надежной передачи данных» мы узнали о том, что транспортный протокол способен обеспечить надежную передачу данных даже в том

случае, если протокол сетевого уровня не предоставляет такой услуги. Мы убедились, что данная задача решается путем совместного использования механизмов квитирования и повторной передачи, таймеров и порядковых номеров.

Несмотря на то что мы рассмотрели надежную передачу данных в главе, посвященной транспортному уровню, следует понимать, что она может с равным успехом осуществляться протоколами сетевого, канального, физического и прикладного уровней. Любой из этих уровней допускает применение квитанций, таймеров, повторных передач и порядковых номеров. В течение многих лет инженеры и ученые разрабатывали и внедряли протоколы надежной передачи данных на всех уровнях коммуникационной модели (значительная часть этих протоколов не выдержала испытания временем и вышла из употребления).

Раздел «Протокол TCP — передача с установлением соединения» был посвящен детальному рассмотрению TCP — протокола транспортного уровня Интернета, использующего логическое соединение и обеспечивающего надежную передачу данных. Мы увидели, что протокол TCP очень сложен и поддерживает механизмы управления соединением, контроля потока, оценки времени оборота и надежной передачи. На самом деле протокол TCP является еще более сложным, чем мы его представили. Тем не менее сложность TCP полностью скрыта от приложений. Если клиентскому процессу необходим надежный обмен данными с сервером, он просто создает TCP-сокет и помещает в него нужные данные.

В разделе «Принципы контролирования перегрузки» мы рассмотрели общие положения, касающиеся контроля перегрузки, а раздел «Контроль перегрузок в TCP» был посвящен механизму контроля перегрузок протокола TCP. Мы узнали, что контролирование перегрузки является необходимым условием производительного функционирования компьютерной сети. Наличие соответствующих механизмов позволяет избежать состояний, в которых происходит полная потеря передаваемых данных либо их значительной части. В разделе «Контроль перегрузок в TCP» мы показали, что протокол TCP предоставляет механизм контроля перегрузки, который позволяет оконечным системам аддитивно увеличивать скорость передачи при отсутствии перегрузок на пути соединения и мультипликативно уменьшать ее в случае потери пакета. Кроме того, этот механизм стремится выровнять скорости передачи всех соединений, совместно использующих перегруженную линию связи. Мы провели небольшое исследование влияния процедур установления соединения и медленного старта на общую задержку. Рассмотрев несколько важных ситуаций, мы показали, что это влияние оказывается весьма существенным. Снова отметим, что процесс совершенствования механизма контроля перегрузки TCP далек от завершения и в будущем ожидается интенсивная исследовательская работа в этой области.

В главе 1 мы разделили компьютерную сеть на «ядро» и «периферию». В «периферию» мы включили все компоненты сети, относящиеся к оконечным системам. Теперь, изучив прикладной и транспортный уровни, мы завершаем разговор о «периферии» компьютерной сети и переходим к рассмотрению «ядра». Глава 4 будет посвящена сетевому уровню, а глава 5 — канальному.

Вопросы и задания для самостоятельной работы

Приведенные ниже задания рекомендуются для самостоятельной проработки. При возникновении трудностей обратитесь к соответствующим разделам этой главы.

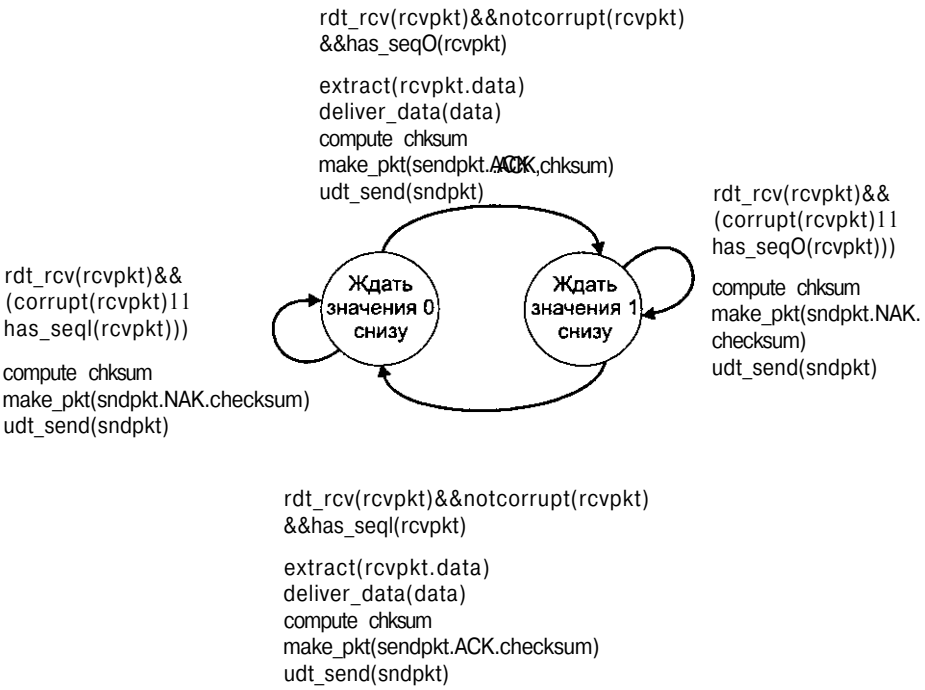
1. Пусть между хостами A и B имеется TCP-соединение, при этом сегменты, передающиеся хостом A, имеют номер порта отправителя *x* и номер порта получателя *y*. Укажите номер порта отправителя и номер порта получателя для сегментов, передаваемых хостом B.
2. Назовите причины, по которым разработчики приложений для поддержки своих продуктов могут предпочесть протокол UDP протоколу TCP.
3. Возможна ли надежная передача данных приложением при использовании протокола UDP? Если да, то каким образом она реализована?
4. Верны или нет следующие утверждения?
   - 4.1. Хост A передает хосту B большой файл данных через TCP-соединение. Предположим, что хост B не имеет данных для передачи хосту A. Хост B не будет квитировать сегменты, поступающие от хоста A, поскольку он не сможет вложить квитанции в исходящие сегменты.
   - 4.2. Размер окна приема протокола TCP никогда не изменяется в течение соединения.
   - 4.3. Хост A передает хосту B большой файл данных через TCP-соединение. Число неподтвержденных байтов, посылаемых хостом A, не превышает размера приемного буфера.
   - 4.4. Хост A передает хосту B большой файл данных через TCP-соединение. Если *m* — порядковый номер некоторого передаваемого сегмента, то следующий сегмент всегда имеет порядковый номер *m* + 1.
   - 4.5. Заголовок TCP-сегмента имеет поле размера окна приема.
   - 4.6. Предположим, что последнее значение переменной SampleRTT TCP-соединения равно 1 с. Это означает, что текущее значение TimeoutInterval составляет не менее 1 с.
   - 4.7. Пусть хост A пересылает хосту B единственный сегмент, имеющий порядковый номер 38 и содержащий 4 байта данных. Номер подтверждения этого сегмента обязательно равен 42.
5. Предположим, что хост A через TCP-соединение осуществляет передачу хосту B двух сегментов, следующих один за другим. Первый сегмент имеет порядковый номер 90, а второй — номер HO.
   - 5.1. Каков объем данных, содержащихся в первом сегменте?
   - 5.2. Предположим, что первый сегмент теряется, однако второй сегмент успешно доставляется хосту B. Каков будет номер подтверждения, посылаемый хостом B?
6. Вернемся к примеру с протоколом Telnet, рассмотренному в разделе «Протокол TCP — передача с установлением соединения». Пусть через несколько секунд после нажатия клавиши C пользователь вводит символ R. Каким будет число

посылаемых сегментов после завершения ввода? Каковы значения полей порядкового номера и номера подтверждения этих сегментов?

7. Предположим, два TCP-соединения используют ресурсы проблемной линии связи с пропускной способностью *R* бит/с. По обоим соединениям передаются файлы большого размера в одном и том же направлении; передача файлов начинается в один и тот же момент времени. Какова скорость передачи, обеспечиваемая протоколом TCP для каждого из соединений?
8. Верно или нет, что в механизме контроля перегрузки TCP по истечении интервала ожидания происходит снижение порогового значения вдвое?

Упражнения

1. Клиент A устанавливает Telnet-сеанс с сервером B. Приблизительно в это же время клиент B устанавливает Telnet-сеанс с сервером S.
   1.1. Укажите возможные номера портов отправителя и получателя для сегментов, передаваемых от A к S.
   1.2. Укажите возможные номера портов отправителя и получателя для сегментов, передаваемых от B к S.
   1.3. Укажите возможные номера портов отправителя и получателя для сегментов, передаваемых от S к A.
   1.4. Укажите возможные номера портов отправителя и получателя для сегментов, передаваемых от S к B.
   1.5. Если хосты A и B различны, могут ли быть одинаковыми номера портов отправителя для сегментов, передаваемых от A к S, и сегментов, передаваемых от B к S?
   1.6. Дайте ответ на предыдущий вопрос для случая, если A и B — это один хост.
2. Вернитесь к рис. 3.5. Каковы номера портов отправителя и получателя сегментов, передаваемых от сервера к клиенту? Какие IP-адреса содержатся в дейтаграммах, включающих эти сегменты?
3. Протоколы TCP и UDP используют операцию дополнения до 1 при формировании контрольной суммы. Пусть имеются три байта: 01010101, 01110000 и 01001100. Чему равно дополнение до 1 суммы этих байтов? (Хотя при подсчете контрольной суммы протоколы UDP и TCP используют 16-разрядные слова, в данном случае мы будем оперировать 8-разрядными байтами.) Почему UDP дополняет обычную сумму до 1? Как с помощью этой операции принимающая сторона обнаруживает ошибки? Возможно ли, что искажение одного бита не будет обнаружено? Будет ли обнаружено искажение двух битов?
4. Вспомните о тех доводах, которые приводились в пользу совершенствования разработанного нами протокола rdt 2.1 (см. рис. 3.11). Покажите, что действия принимающей стороны могут заблокировать соединение, введя клиент и сервер в состояние ожидания события, которое никогда не сможет произойти.



5. В протоколе **rdt** 3.0 подтверждения, передаваемые от приемника к источнику, не имеют порядковых номеров (хотя они содержат поле подтверждения с порядковым номером сегмента, к которому относится подтверждение). Почему нет необходимости снабжать подтверждения порядковыми номерами?
6. Изобразите конечный автомат, описывающий принимающую сторону протокола **rdt** 3.0.
7. Изобразите последовательность действий протокола **rdt** 3.0 для случая, когда пакеты и подтверждения содержат искажения. Ваша схема должна напоминать ту, которая приведена на рис. 3.15.
8. Представьте себе канал связи, допускающий потери пакетов, однако имеющий известное максимальное время задержки. Измените протокол **rdt** 2.1, включив в него механизмы интервалов ожидания и повторной передачи. Дайте неформальное пояснение, почему ваш протокол будет корректно работать с данным каналом.
9. Передающая сторона протокола **rdt** 3.0 игнорирует (не предпринимает никаких действий) при наличии искажений в данных или при неверном значении порядкового номера. Представьте, что в этом случае протокол осуществляет повторную передачу текущего пакета. Будет ли протокол работоспособным? Сначала рассмотрите ситуацию, в которой присутствуют только искажения данных, то есть потерь данных не происходит, но возможно истечение интервала ожидания. Обдумайте, сколько раз происходит передача *п-то* пакета при значении *п,* стремящемся к бесконечности.

10. Рассмотрим протокол с чередованием битов (называемый также протоколом с ожиданием подтверждения). Нарисуйте диаграмму, демонстрирующую, что в случае, если соединение между источником и приемником допускает изменение порядка следования сообщений, работа протокола будет некорректной (обдумайте, что означает в данном контексте термин «некорректная»). Расположите передающую и принимающую стороны соответственно слева и справа, ось времени направьте вниз и введите в диаграмму передачу данных (D) и квитанций (A), указывая порядковые номера сегментов.
11. Рассмотрим протокол надежной передачи данных, использующий только отрицательные квитанции. Предположим, что передача данных источником ведется нерегулярно. Будет ли такой протокол функционировать лучше в сравнении с протоколом, использующим положительные квитанции? Почему? Предположим, что источник посылает значительный объем данных, при передаче которого наблюдаются потери. Какой из протоколов будет лучше работать в этом случае? Почему?
12. Вернемся к примеру передачи данных между Санкт-Петербургом и Новосибирском (см. рис. 3.16). Насколько большим следует сделать размер окна для того, чтобы коэффициент использования соединения составил не менее 90 %?
13. Разработайте протокол надежной передачи данных с конвейеризацией, в~котором используются только отрицательные квитанции. Насколько быстрым будет отклик протокола на потерю пакетов, если скорость получения данных источником мала? Если скорость получения данных источником велика?
14. В SR-протоколе, рассмотренном в разделе «Принципы надежной передачи данных», передающая сторона отсылает сообщение, находящееся в окне, максимально скоро (не ожидая получения квитанции). Предположим, что нам необходимо разработать SR-протокол, который передает сообщения парами (каждая новая пара отсылается в случае, если передача обоих сообщений предыдущей пары оказывается успешной).

    Предположим, что канал связи допускает потерю сообщений, однако переупорядочивание и искажение сообщений невозможно. Разработайте протокол надежной однонаправленной передачи с контролем ошибок. Опишите передающую и принимающую стороны с помощью конечных автоматов. Опишите формат пакетов, генерируемых каждой из сторон. В случае использования новых процедур (не входящих в число тех, которые были описаны в разделе «Принципы надежной передачи данных», то есть процедур udt_send(), start_timer(), rdt_rcv() и т. д.) точно опишите действия каждой из них. Приведите пример (в виде временной диаграммы), иллюстрирующий поведение протокола при потере пакета.
15. Представьте ситуацию, в которой хосту A необходимо осуществить одновременную передачу сообщений хостам B и C. Хост A соединен с хостами B и C с помощью широковещательного канала — пакет, передаваемый хостом A, доставляется как хосту B, так и хосту C. Предположим, что канал допускает независимую потерю и искажение сообщений (то есть возможно успешное получение сообщения хостом B, однако при этом хост C может либо не получить

26.1. Укажите интервалы времени, соответствующие фазе медленного старта.

26.2. Укажите интервалы времени, соответствующие фазе ухода от перегрузки.

26.3. Каким из способов (истечение интервала ожидания или получение трех дублирующих подтверждений) была обнаружена потеря пакета после 16-го цикла передачи?

26.4. Каким из способов была обнаружена потеря пакета после 22-го цикла передачи?

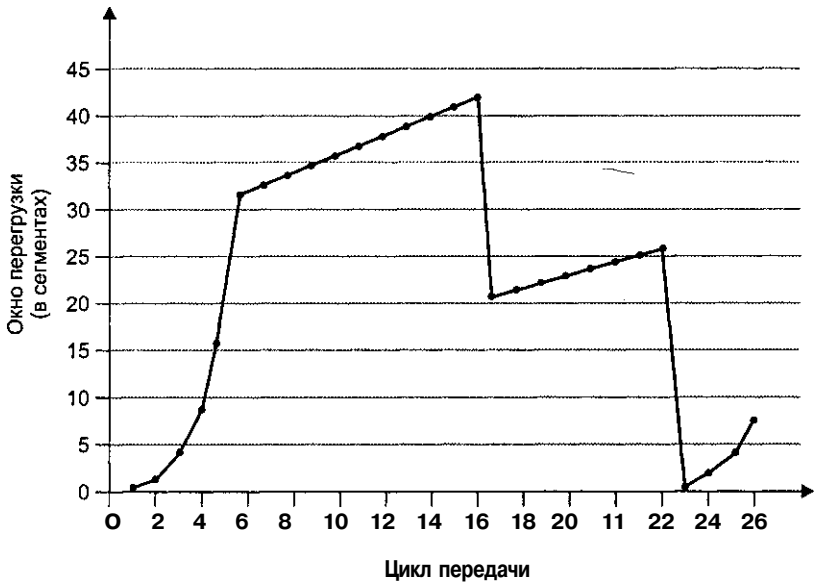
26.5. Каково начальное значение переменной **Threshold?**

26.6. Каково значение переменной **Threshold** после 18-го цикла передачи?

26.7. Каково значение переменной **Threshold** после 24-го цикла передачи?

26.8. В каком периоде осуществляется передача 70-го сегмента?

26.9. Предположим, что после 26-го цикла передачи обнаружилась потеря пакета, о чем передающая сторона была уведомлена с помощью трех дублирующих подтверждений. Укажите новый размер окна и пороговое значение.



27. Обратимся к рис. 3.49, иллюстрирующему оптимальное соотношение значений пропускной способности при использовании алгоритма аддитивного увеличения и мультипликативного уменьшения. Предположим теперь, что TCP уменьшает размер окна не мультипликативно, а аддитивно. Будет ли полученный алгоритм аддитивного увеличения и аддитивного уменьшения обеспечивать равные скорости передач? Поясните ответ с помощью диаграммы, аналогичной приведенной на рис. 3.49.

28. В подразделе «Контроль потока» раздела «Протокол TCP — передача с установлением соединения» мы рассмотрели механизм удвоения интервала ожидания по его истечении. Почему в протоколе TCP кроме этого механизма также используется окно приема?

29. Пусть хост А передает хосту В файл большого объема через TCP-соединение. В этом соединении не происходит потерь пакетов, и таймеры никогда не переполняются. Линия связи, соединяющая хост А с Интернетом, имеет пропускную способность ***R*** бит/с. Предположим, что процесс, выполняющийся на хосте А, способен передавать данные в сокет со скоростью *S* бит/с, где 5 = 10***R.*** Также предположим, что приемный буфер TCP достаточно велик для хранения всего файла, а буфер передачи способен хранить лишь 1 % от объема файла. Какой механизм (контроль потока, контроль перегрузки или другой) не позволит процессу передавать данные со скоростью 5?

30. Вернемся к идеализированной макроскопической модели устойчивого TCP-соединения. В конце каждого периода времени, в котором скорость передачи изменяется от W/2RTT до W/RTT, происходит потеря одного пакета.

    30.1. Покажите, что частота потерь

$$\frac{o}{-W}\ \frac{2}{8}\ \frac{o}{+-W}\ \frac{}{4}$$

    30.2. Используя полученный результат, покажите, что средняя скорость передачи

$$\_\ \frac{1,22xM55}{RTTy/l}$$

31. Объект размером О = 100 Кбайт передается от сервера клиенту. Пусть 5 = = 536 байт, а MSS =100 мс. Предположим, что в протоколе используется статическое окно размера **Wcm** подраздел «Модель задержек протокола TCP» в разделе «Контроль перегрузок в TCP»).

    31.1. Определите минимальную величину задержки при скорости передачи, равной 28 Кбит/с. Укажите минимальный размер окна, обеспечивающий такую задержку.

    31.2. Выполните первое задание для скорости передачи, равной 100 Кбит/с.

    31.3. Выполните первое задание для скорости передачи, равной 1 Мбит/с.

    31.4. Выполните первое задание для скорости передачи, равной 10 Мбит/с.

32. Предположим, что в фазе медленного старта протокол TCP увеличивает размер окна перегрузки не на один, а на два размера сегмента с получением каждой новой квитанции. Таким образом, первое окно включает один сегмент, второе окно — три сегмента, третье окно — девять сегментов, и т. д. Выполните следующие задания, используя выкладки, сделанные в подразделе «Модель задержек протокола TCP» раздела «Контроль перегрузок в TCP».

32.1. Выразите K через O и S.

32.2. Выразите Q через RTT, S и 7?.

32.3. Выразите задержку через P = min(X - 1, Q), O, i? и RTT.

33. Пусть RTT = 1 с и O = 100 Кбайт. Постойте таблицу, сходную с табл. 3.4-3.6 в которых сравнивается минимальная задержка $(O/R + 2RTT)$ с задержкой при медленном старте для скоростей передачи R = 28 Кбит/с, 100 Кбит/с, 1 Мбит/с 10 Мбит/с.

34. Верны или нет следующие утверждения.

34.1. Если web-страница состоит из единственного объекта, то и постоянные, и непостоянные соединения обеспечат одинаковое время ответа.

34.2. Пусть сервер пересылает браузеру объект размером O через TCP-соединение. Если O > S, где 5 — максимальный размер сегмента, то простой сервера в процессе передачи будет иметь место хотя бы один раз.

34.3. Пусть web-страница состоит из 10 объектов, каждый из которых имеет размер O бит. В случае постоянного HTTP-соединения та составляющая во времени ответа, которая относится ко времени кругового оборота, равна 20RTT.

34.4. Пусть web-страница состоит из 10 объектов, каждый из которых имеет размер O бит. В случае непостоянного HTTP-соединения та составляющая во времени ответа, которая относится ко времени кругового оборота, равна 12RTT.

35. Предлагаем вам сделать несколько дополнительных выкладок, связанных с выводом формулы задержки в подразделе «Модель задержек протокола TCP» раздела «Контроль перегрузок в TCP».

35.1. Докажите равенство:

$$Q = \log_2\left(1 + \frac{RTT}{S/R}\right) + 1.$$

35.2. Используя соотношение $\sum_{k=0}^{P-1} 2^k = 2^P - 1$, получите формулу задержки:

$$2RTT + \frac{O}{R} + P\left(RTT + \frac{S}{R}\right) - (2^P - 1)S$$

36. Анализ динамических окон в подразделе «Модель задержек протокола TCP» раздела «Контроль перегрузок в TCP» построен на предположении, что между клиентом и сервером существует единственная линия связи. Выполните аналогичный анализ для случая T линий связи между клиентом и сервером, допустив, что в сети отсутствуют перегрузки, то есть задержки ожидания пакетов равны нулю (тем не менее задержки обработки следует учитывать). Определение времени оборота происходит таким же образом, как описано в ука-

занном разделе (время, проходящее с момента отправки сервером первого сегмента до получения подтверждения, составляет *TS/R* + RTT).

37. Вернемся к вычислению времени ответа для web-страницы, приведенному в конце подраздела «Модель задержек протокола TCP» раздела «Контроль перегрузок в TCP». Выразите долю времени ответа, обусловленную фазой медленного старта TCP, для непостоянных HTTP-соединений.

Дополнительные вопросы и задания

1. Рассмотрим передачу потокового аудио. Каким протоколом (UDP или TCP) следует воспользоваться для поддержки соответствующих приложений? Какой протокол использует система RealNetworks? Почему?
2. Ответьте на предыдущий вопрос для продуктов потокового мультимедиа Microsoft.
3. В разделе «Контроль перегрузок в TCP» мы упомянули о том, что приложение может «нечестно» использовать линии связи при помощи параллельных TCP-соединений. Каковы способы организации «честного» (всем поровну) распределения ресурсов Интернета между приложениями?
4. Ознакомьтесь с исследовательскими публикациями о «дружественном протоколе TCP». Прочитайте интервью Салли Флойд в конце этой главы. Напишите небольшое резюме о дружественном протоколе TCP.

Интервью

Салли Флойд является научным исследователем Центра Интернет-исследований AT&T ICSI (ACIRI) — института, занимающегося проблемами компьютерных сетей и Интернета. Она приобрела известность благодаря участию в разработке протокола IP, в частности в разработке механизмов надежной групповой рассылки, контроля перегрузки протокола TCP, планирования пакетов (RED) и анализа протоколов. Она получила диплом социолога, а затем инженера вычислительной техники.

- Почему вы решили заняться вычислительной техникой?

  Получив диплом социолога, я должна была решить, каким образом обеспечить свое существование. Я завершила двухлетний курс обучения электронике в местном колледже и следующие 10 лет посвятила работе в сфере вычислительной техники, 8 из которых я разрабатывала компьютерные системы для высокоскоростных поездов. Затем я решила заняться изучением теории и поступила на отделение вычислительной техники университета Беркли.

- Что привело вас к решению специализироваться в области вычислительной техники?

  Обучаясь в университете, я интересовалась теорией вычислительной техники. Сначала я занималась вероятностным анализом алгоритмов, а затем теорией вычислений. Раз в месяц я работала в лаборатории LBL, и мой офис

располагался на другом конце коридора от офиса Ванна Якобсона, который / в то время работал над проблемами контроля перегрузки протокола TCP. Ван предложил мне работу на летнее время, заключавшуюся в анализе алгоритмов, направленных на решение проблемы нежелательной синхронизации периодических сообщений маршрутизации. Меня заинтересовало это предложение, и я приняла его.

После защиты диплома Ван выразил желание продолжить сотрудничество, предложив постоянную работу. В мои планы не входило посвятить компьютерным сетям следующие 10 лет своей жизни, однако сетевые исследования привлекали меня больше, чем теория вычислительной техники. Я предпочитаю практическое поле деятельности, поскольку результаты моей работы здесь более ощутимы.

- В чем заключалась ваша первая работа в области вычислительной техники и каковы были ее последствия для вас?

  С 1975 по 1982 год я работала над системой BART (Bay Area Rapid Transit), занимаясь компьютерами для поездов. Будучи техником в начальный период, я поддерживала и ремонтировала различные распределенные компьютерные системы, использовавшиеся в системе BART.

  Управление движением поездов осуществлялось при помощи централизованной компьютерной системы, распределенной микрокомпьютерной системы, системы DEC-компьютеров, отвечавшей за отображение станций назначения и рекламы, и системы M'odcomp-компьютеров, собиравших информацию от билетных касс. Последние годы моей работы над системой BART были посвящены проекту BART/LBL, целью которого была заменаустаревшей компьютерной системы управления движением поездов.

- В чем заключается самая сложная часть вашей работы?

  Самая сложная часть работы для меня — собственно исследование. В настоящее время это разработка и изучение нового механизма контроля перегрузки оконечными системами. Новый механизм не предназначен для замены существующего в протоколе TCP; он лишь поможет улучшить качество обслуживания при индивидуальной передаче адаптивного трафика реального времени, для которого крайне нежелательны снижения скорости передачи вдвое при каждой потере пакета. Контроль перегрузки также интересен в качестве потенциальной основы механизма, который может быть использован при групповой передаче. Более подробную информацию вы можете найти на web-странице http://www.psc.edu/networking/tcp_friendly.html.

- Каким вы представляете будущее компьютерных сетей и Интернета?

  Возможно, перегрузки в Интернете станут менее жесткими, когда вступят в действие ценовые факторы и рост максимальных скоростей передачи будет опережать запросы. При этом я все же не исключаю, что в среднесрочном периоде может произойти своеобразный коллапс Интернета, вызванный мощными перегрузками.

Будущее Интернета или его архитектуры не представляется мне ясным. Слишком большое число факторов может вызывать быстрые изменения, поэтому трудно не только говорить о перспективах Интернета и его архитектуры, но и оценивать, насколько успешным будет преодоление трудностей, неизбежно возникающих в процессе их развития.

- Кого бы вы назвали своими профессиональными вдохновителями?

  Ричарда Карпа, который консультировал меня в процессе написания диплома в университете и учил проведению исследований, а также Ванна Якобсона, моего руководителя группы в LBL, повлиявшего на мой интерес к компьютерным сетям и предоставившего немало ценной информации об инфраструктуре Интернета. Авторитетом для меня является Дэйв Кларк с его глубоким пониманием архитектуры Интернета и значительной роли в ее развитии исследований и публикаций. Дебора Эстрин вдохновила меня своей сосредоточенностью, работоспособностью и умением принимать обоснованные решения о дальнейших путях своих исследований.

  Присутствие коллег, к которым я испытываю уважение и симпатию, является одной из причин, благодаря которым моя десятилетняя работа в сфере исследований сетевых технологий доставляет мне большое удовольствие. Эти люди обладают живым умом ц упорством в работе, имеют непосредственное отношение к развитию Интернета, приятны в общении и за кружкой пива, и в ходе профессиональных споров.

ГЛАВА 4 Сетевой уровень и маршрутизация

В предыдущей главе мы говорили о том, что транспортный уровень предоставляет службы связи между двумя процессами, работающими на двух разных хостах. Для предоставления служб связи между процессами транспортный уровень опирается на службы связи между хостами, предоставляемые сетевым уровнем. Мы узнали, что транспортный уровень пользуется службой сетевого уровня, не имея информации о том, как эта служба реализована.

Эта глава посвящена как раз тому, как сетевой уровень реализует службы связи между хостами. Мы увидим, что, в отличие от транспортного уровня, «кусок» сетевого уровня присутствует *в каждом хосте и маршрутизаторе сети.* Благодаря этому протоколы сетевого уровня являются одш^ми из самых сложных (и поэтому одними из самых интересных!).

Мы начнем наше обсуждение с изучения всевозможных служб, которые могут предоставляться сетевым уровнем. Затем мы рассмотрим теоретическое обоснование сетевого уровня — алгоритмы маршрутизации, определяющие пути, по которым движутся пакеты через сеть от отправителя к получателю. Далее мы поговорим об Интернет-протоколе (IP) и маршрутизации в Интернете, в том числе о формате IP-дейтаграммы, адресации и тесной взаимосвязи между адресацией и маршрутизацией. Мы также заглянем «внутрь» маршрутизатора — рассмотрим архитектуру и организацию его аппаратного обеспечения. Мы завершим главу подробным обсуждением важных специальных вопросов — протокола IP версии 6, групповой маршрутизации и мобильных хостов.

Модели сетевого обслуживания

На рис. 4.1 изображена схема простой сети с двумя хостами, H1 и H2, и несколькими маршрутизаторами на пути от хоста H1 до хоста H2. Пусть хост H1 посылает информацию хосту H2. Рассмотрим роль сетевого уровня на этих хостах и промежуточных маршрутизаторах. Сетевой уровень хоста H1 принимает сегменты от транспортного уровня хоста H1, инкапсулирует каждый сегмент в дейтаграмму (единицу обмена сетевого уровня), после чего отправляет дейтаграммы в путь к их

адресату; то есть он посылает дейтаграммы своему ближайшему маршрутизатору R1. На принимающем хосте H2 сетевой уровень получает дейтаграммы от своего ближайшего маршрутизатора (в данном случае R2), извлекает сегменты транспортного уровня и доставляет их транспортному уровню хоста H2. Основная задача маршрутизаторов заключается в «продвижении» дейтаграмм из входных линий связи в выходные линии. Обратите внимание, что на рис. 4.1 маршрутизаторы показаны с сокращенным стеком протоколов, то есть без уровней выше сетевого, потому что на маршрутизаторах не работают протоколы прикладного и транспортного уровней (исключая задачи контроля), рассмотренные нами в главах 2 и 3.

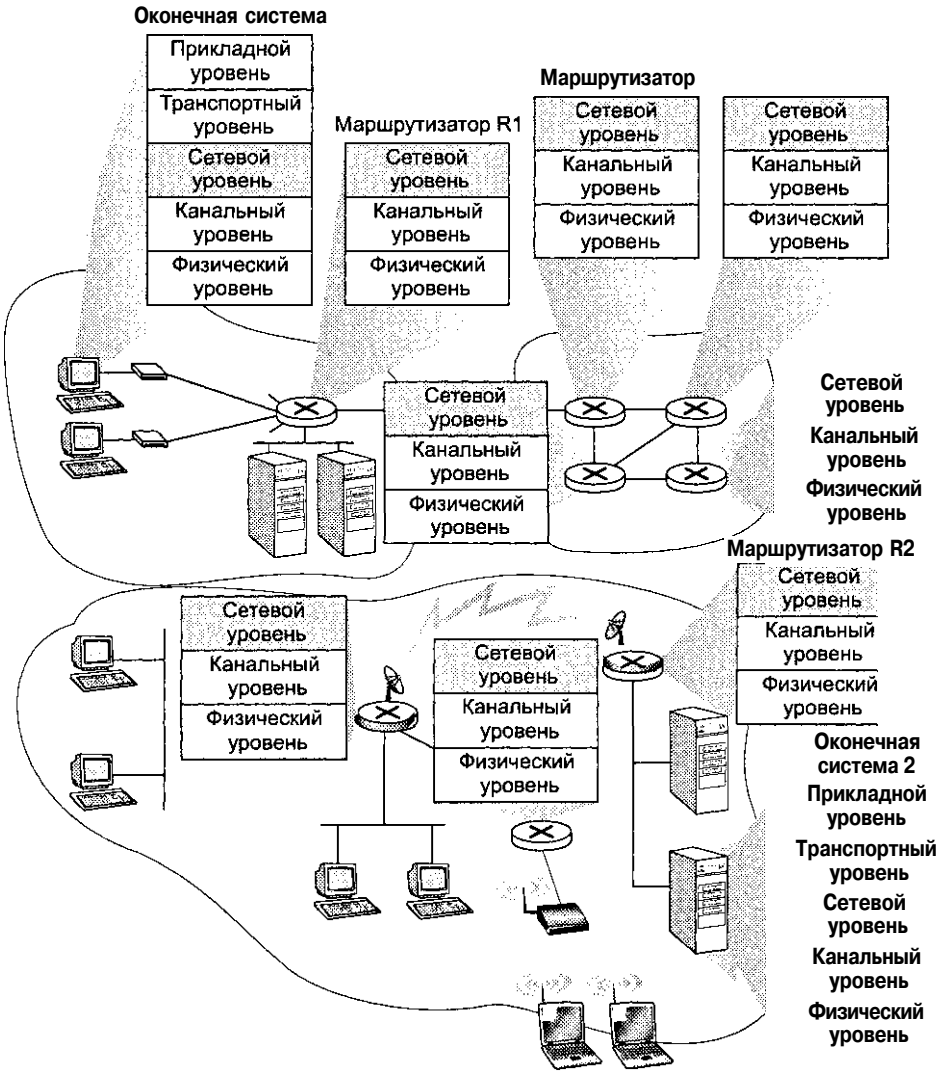


Рис. 4.1. Сетевой уровень

Таким образом, роль сетевого уровня обманчиво проста — перемещение пакетов от передающего хоста к принимающему. Для этого можно выделить три важные функции сетевого уровня.

- *Определение пути.* Сетевой уровень должен определить маршрут, или путь, по которому следуют пакеты от отправителя к получателю. Алгоритмы, рассчитывающие эти маршруты, называются *алгоритмами маршрутизации.* Алгоритм маршрутизации определяет, например, путь, по которому пакеты движутся от хоста H1 к хосту H2. Большая часть этой главы посвящена алгоритмам маршрутизации. В разделе «Основы маршрутизации» мы изучим теорию алгоритмов маршрутизации, уделив особое внимание двум наиболее распространенным классам алгоритмов маршрутизации: маршрутизации с учетом состояния линии и дистанционно-векторной маршрутизации. Мы увидим, что сложность алгоритмов маршрутизации значительно возрастает с увеличением количества маршрутизаторов в сети. Это служит мотивом для использования иерархической маршрутизации, которую мы обсудим в разделе «Иерархическая маршрутизация».
- *Продвижение данных.* Когда пакет прибывает на вход маршрутизатора, маршрутизатор должен переместить его на соответствующую выходную линию. Например, пакет, прибывающий с хоста H1 на маршрутизатор R2, должен быть передан следующему маршрутизатору на пути к хосту H2. В разделе «Устройство маршрутизатора» мы обсудим устройство маршрутизатора и поговорим о коммутации (продвижении) пакета с входной линии в выходную.
- *Установка соединения.* Как было показано при обсуждении протокола TCP, прежде чем начинать продвижение данных от отправителя к получателю, требовалось выполнить процедуру тройного рукопожатия. Это давало отправителю и получателю возможность настроить необходимые параметры состояния соединения (например, задать порядковый номер и начальный размер окна управления потоком). Аналогичным образом, некоторые архитектуры сетевых уровней (например, ATM) требуют, чтобы маршрутизаторы вдоль выбранного пути от отправителя до получателя обменялись рукопожатиями друг с другом, чтобы настроить параметры, прежде пакеты сетевого уровня с данными начнут свое движение. На сетевом уровне этот процесс называют *установкой соединения* (call setup). Сетевой уровень архитектуры Интернета не выполняет установки соединения.

Термины «маршрутизация» и «продвижение данных» многие авторы часто путают и используют как синонимы. Мы постараемся применять эту терминологию более точно. «Маршрутизацией» мы будем называть глобальный (охватывающий всю сеть) процесс определения всего пути, который проходит дейтаграмма от отправителя до получателя. «Продвижением данных» мы станем называть локальные действия конкретного маршрутизатора по перемещению дейтаграммы из интерфейса входной линии связи в интерфейс выходной линии. Используя аналогию с вождением автомобиля, маршрутизацию можно сравнить с процессом составления карты и прокладки маршрута от отправителя до получателя в виде последовательности перекрестков. Мы увидим, что подобно тому, как для прокладки марш-

рутов могут потребоваться несколько карт (например, городская карта для езды по городу, карта области для езды за пределами города и т. д.), в определении пути от отправителя до получателя могут участвовать несколько протоколов. Продолжая нашу аналогию с вождением автомобиля, продвижение данных соответствует процессу преодоления единственного перекрестка — автомобиль подъезжает к перекрестку, водитель определяет направление до следующего перекрестка на своем пути к конечной цели, а затем выбирает соответствующую дорогу к этому перекрестку.

Прежде чем углубиться в детали теории и реализации сетевого уровня, рассмотрим различные типы служб, предоставляемые сетевым уровнем.

Понятие модели сетевого обслуживания

Когда транспортный уровень передающего хоста посылает пакет в сеть (то есть передает его сетевому уровню того же хоста), может ли транспортный уровень положиться на сетевой уровень в деле доставки пакета получателю? Когда посылается большое количество пакетов, будут ли они доставлены транспортному уровню в том же порядке, в котором были отправлены? Сохранятся ли длительности временных интервалов между двумя последовательными пакетами? Будет ли сеть предоставлять обратную связь, извещая о перегрузке? Каковы абстрактные свойства канала, соединяющего транспортные уровни передающего и принимающего хостов? Ответы на эти вопросы определяются моделью обслуживания, предоставляемого сетевым уровнем. *Модель сетевого обслуживания* определяет характеристики сквозного транспорта данных между двумя периферийными устройствами сети, то есть между передающей и получающей оконечными системами.

Возможно наиболее важной абстракцией, предоставляемой сетевым уровнем более высоким уровням, является *виртуальный канал* (Virtual Channel, VC). В главе 1 говорилось, что пакетная сеть с виртуальными каналами ведет себя во многом подобно телефонной сети, в которой вместо «виртуальных каналов» используются «реальные каналы». «Жизнь» виртуального канала состоит из трех фаз.

1. *Установка виртуального канала.* Во время этой фазы отправитель связывается с сетевым уровнем, указывает адрес получателя и ждет, пока сеть установит виртуальный канал. Сетевой уровень определяет путь от отправителя до получателя, то есть последовательность линий связи и пакетных коммутаторов, через которые будут проходить все пакеты данного виртуального канала. Как было показано в главе 1, этот процесс обычно подразумевает обновление таблиц в каждом пакетном коммутаторе вдоль пути виртуального канала. Во время установки виртуального канала сетевой уровень может также зарезервировать ресурсы (например, пропускную способность) вдоль пути виртуального канала.
2. *Передача данных.* Как только виртуальный канал установлен, данные могут начать перемещение по виртуальному каналу.
3. *Разрыв виртуального канала.* Эта процедура начинается, когда отправитель (или получатель) информирует сетевой уровень о своем желании разорвать виртуальный канал. Затем, как правило, сетевой уровень информирует оконечную

систему на другой стороне сети о разрыве соединения и обновляет таблицы в каждом пакетном коммутаторе пути, показывая, что виртуального канала более не существует.

Между установкой виртуального канала на сетевом уровне и установкой соединения на транспортном уровне (например, тройным рукопожатием протокола TCP, которое мы обсуждали в главе 3) есть едва различимая, но важная разница. В установку соединения на транспортном уровне вовлечены только две оконечные системы. Две оконечные системы договариваются об обмене данными и совместно определяют параметры (например, начальные порядковые номера и размер окна управления потоком) соединения транспортного уровня, прежде чем данные начнут перемещаться по соединению транспортного уровня. Хотя две оконечные системы «знают» о соединении транспортного уровня, коммутаторам сети ничего о нем не известно. Вместе с тем в случае сетевого уровня, основанного на виртуальных каналах, *пакетные коммутаторы вдоль пути между двумя оконечными системами вовлекаются в установку виртуального канала, и таким образом каждый коммутатор знает все обо всех виртуальных каналах, проходящих через него.*

Сообщения, которые оконечные системы посылают в сеть, чтобы информировать о начале процедуры разрыва виртуального канала, а также сообщения, которыми обмениваются коммутаторы для установки виртуального канала (то есть для изменения таблиц маршрутизации коммутаторов), называются *сигнальными сообщениями,* а протоколы, используемые для обмена этими сообщениями, часто называют *сигнальными протоколами.* Процесс установки виртуального канала схематично показан на рис. 4.2. Три технологии, в которых применяются виртуальные каналы (ATM, Frame Relay и X.25), будут описаны в главе 5.

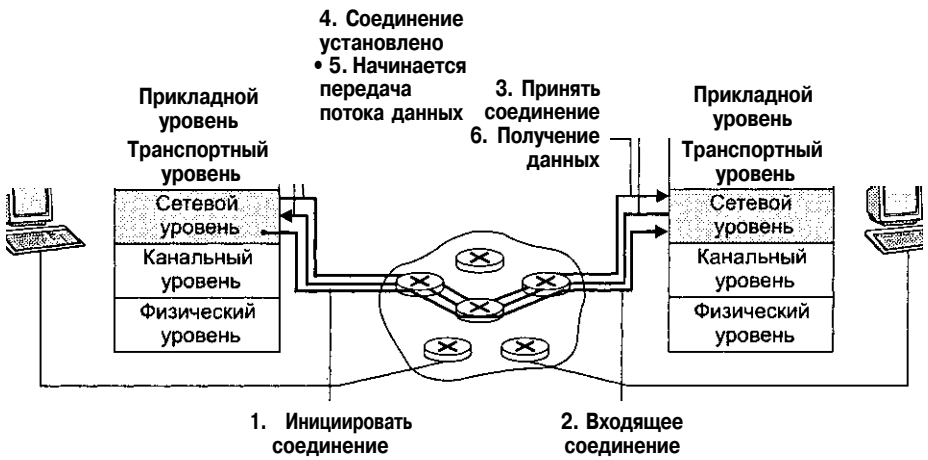


Рис. 4.2. Модель обслуживания на основе виртуальных каналов

При использовании *дейтаграммного сетевого уровня* каждый раз, когда оконечная система хочет послать пакет, она указывает в нем адрес получающей оконечной системы, а затем передает этот пакет в сеть. Как показано на рис. 4.3, эта процедура

выполняется без предварительной установки виртуального канала. Коммутаторы пакетов в дейтаграммной сети (называемые в Интернете «маршрутизаторами») не содержат информации о состоянии виртуальных каналов (так как виртуальных каналов нет!). Вместо этого коммутаторы пакетов продвигают пакет по направлению к адресату, изучая адрес получателя пакета. При этом они ищут нужную им для этого информацию в своей *таблице продвижения данных,* используя адрес получателя в качестве индекса. (Как упоминалось в главе 1, маршрутизация дейтаграмм напоминает процесс выбора маршрута для обычной почтовой открытки.) Поскольку таблицы продвижения данных могут быть изменены в любое время, пакеты, относящиеся к одной серии пакетов, посланных одной оконечной системой другой оконечной системе, могут следовать по разным маршрутам и прибыть к получателю не в том порядке, в котором были посланы. В [371] содержится интересное исследование о нарушении порядка прибытия пакетов в реальных сетях, а также другие феномены Интернета.

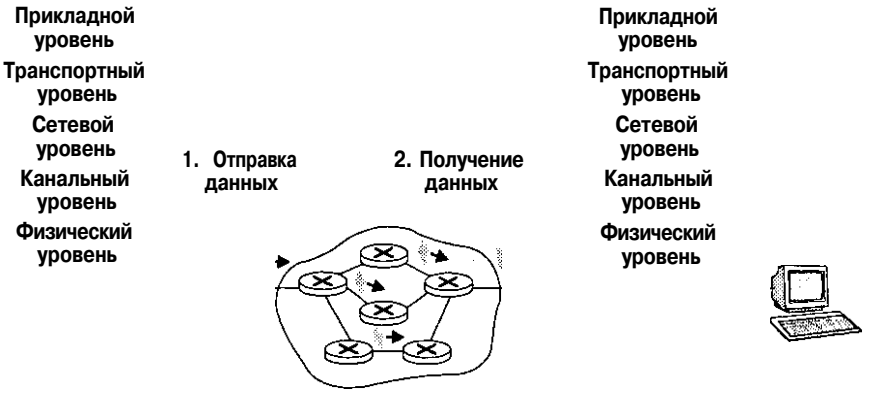


Рис. 4.3. Модель обслуживания на основе дейтаграмм

Службу виртуальных каналов также называют *сетевой службой с у становлением соединения,* а дейтаграммную службу — *сетевой службой без установления соединения.* В самом деле, служба виртуальных каналов представляет собой разновидность службы с установлением соединения, так как ее использование подразумевает установление и разрыв соединения, а также поддержку информации о состоянии соединения в коммутаторах пакетов. Дейтаграммная служба является разновидностью службы без установления соединения, так как в ней соединения не используются. Эта терминология обладает своими достоинствами и недостатками, и в литературе, посвященной компьютерным сетям, часто применяется и те и другие термины. В этой книге мы решили для сетевого уровня использовать понятия «службы виртуальных каналов» и «дейтаграммной службы», а термины «служба с установлением соединения» и «служба без установления соединения» оставить для транспортного уровня. Мы полагаем, что такое разграничение терминологии поможет читателю различать службы, функционирующие на этих двух уровнях.

Современная архитектура Интернета предоставляет единственную модель обслуживания с использованием дейтаграмм, также называемую *обслуживанием по остаточному принципу* (best-effort service). Если взглянуть на табл. 4.1, может показаться, что обслуживание по остаточному принципу — это просто эвфемизм, означающий «вообще никакого обслуживания». При обслуживании по остаточному принципу не дается гарантий сохранения временных интервалов между пакетами, не дается гарантий доставки пакетов с сохранением исходного порядка следования и даже не предоставляются гарантии доставки переданных пакетов. Такому определению службы будет удовлетворять даже сеть, которая вообще не доставляет никаких пакетов (кстати, в часы перегрузки Интернет напоминает именно такую сеть). Тем не менее, как мы скоро увидим, для использования подобной минималистской модели обслуживания имеются вполне разумные причины. Характерное для сегодняшнего Интернеты обслуживание по остаточному принципу расширяется, предлагая так называемое интегрированное обслуживание и дифференцированное обслуживание. Эти все еще развивающиеся модели обслуживания мы обсудим в главе 6.

В других сетевых архитектурах бытли определены и реализованы модели предоставления услуг, выходящие за рамки обслуживания по остаточному принципу. Например, сетевая архитектура ATM [19] предоставляет несколько моделей обслуживания, таким образом, различные соединения в одной и той же ATM-сети могут предоставлять различные классы обслуживания. Ниже представлены две наиболее важные модели обслуживания ATM.

- *CBR* (Constant Bit Rate — постоянная битовая скорость). Это была первая стандартизированная модель обслуживания ATM, отразившая интерес телефонных компаний к технологии ATM. Служба CBR годится для передачи трафика реального времени с постоянной скоростью, например аудио и видео. Цель службы CBR проста — создать у отправителя и получателя впечатление, что их соединяет выделенная линия связи. Служба CBR гарантирует, что определенные параметры, такие как суммарная задержка доставки ATM-пакетов (в терминологии ATM называемых ячейками), изменчивость этой суммарной задержки (часто называемая «джиттером») и доля потерянных или доставленных с опозданием ячеек, не будут превышать указанных величин. Об этих величинах передающий хост и ATM-сеть договариваются во время установки CBR-соединения.
- *ABR* (Available Bit Rate — доступная битовая скорость). По сравнению со службами Интернета эта служба предлагает несколько лучшее качество обслуживания. Как и в модели обслуживания Интернета, в модели обслуживания с доступной битовой скоростью ячейки могут теряться. Однако в отличие от Интернета порядок следования ячеек гарантируется. К тому же для соединения, использующего службу ABR, гарантируется определенная минимальная скорость передачи ячеек (Minimal Cell Rate, MCR). Кроме того, если в сети имеется достаточно свободных ресурсов, отправитель может успешно передавать данные с большей, чем MCR, скоростью. Помимо этого, как было показано в подразделе «Контроль перегрузок в службе ABR сетей ATM» раздела «Принципы контролирования перегрузки», служба ABR может предоставлять отправителю обратную

связь (при помощи специального бита индикации перегрузки или явного указания скорости передачи), что позволяет отправителю выбирать оптимальную скорость передачи в пределах от MCR до разрешенной пиковой скорости ячеек (Peak Cell Rate, PCR) в соответствии с текущей ситуацией.

Таблица 4.1. Модели обслуживания Интернета, CBR и ABR

Сетевая архитектура	Модель обслуживания	Гарантия пропускной способности	Гарантия отсутствия потерь	Сохранение порядка следования пакетов	Гарантия сроков доставки	Индикация перегрузки
Интернет	Остаточный принцип	Нет	Нет	Нет	Нет	Нет
ATM	CBR	Гарантированная постоянная скорость	Да	Да	Да	Перегрузки не будет
ATM	ABR	Гарантированная минимальная скорость	Нет	Да	Нет	Да

Помимо служб CBR и ABR в стандарте ATM-сети специфицированы такие службы, как VBR (Variable Bit Rate — переменная битовая скорость) и UBR (Unspecified Bit Rate — неуказанная битовая скорость). Информацию об этих службах см. в [20, 174].

Происхождение дейтаграммной службы и службы виртуальных каналов

Эволюция сетевых служб отражает их происхождение. Идея виртуального канала как центрального организационного принципа уходит своими корнями в мир телефонии, в котором используются «реальные» электрические цепи. Сеть виртуальных каналов значительно сложнее дейтаграммной сети, так как в ней требуется установка соединения, а маршрутизаторы сети хранят информацию о состоянии соединения. Эти свойства также представляют собой «родимые пятна» телефонных сетей. Телефонным сетям присуще сложное внутреннее строение, так как еще недавно они соединяли примитивные оконечные устройства, такие как телефоны с дисковым набором номера (для юных читателей, не знакомых с подобной техникой, поясняем, что это аналоговые телефоны, у которых вместо кнопок использовался вращающийся диск, соединенный с механическим прерывателем).

Дейтаграммная модель обслуживания Интернета, напротив, «родилась» из потребности объединения компьютеров. Несмотря на большую сложность оконечных систем, архитекторы Интернета в этом случае решили использовать как можно более простую модель обслуживания сетевого уровня. Как уже упоминалось в главах 2 и 3, другие функции (например, доставка данных с сохранением порядка еле-

дования, надежная доставка, борьба с перегрузкой и разрешение имен DNS) реализуется на более высоком уровне в оконечных системах. Такой подход, полностью противоположный модели телефонной сети, обладает интересными особенностями.

- Получившаяся в результате модель сетевого обслуживания в Интернете, предоставляющая минимальные гарантии (никаких гарантий обслуживания!) и поэтому предъявляющая минимальные требования к сетевому уровню, облегчает задачу *объединения сетей,* использующих различные технологии канального уровня (например, спутниковые сети, Ethernet, оптоволоконные кабели или радиосети) и обладающих сильно различающимися скоростями передачи данных, а также характеристиками потерь данных. Подробно вопрос объединения IP-сетей будет рассматриваться в разделе «Интернет-протокол».
- Как мы видели в главе 2, такие приложения, как электронная почта, web и даже DNS, реализуются на хостах (серверах), то есть на периферии сети. Возможность добавления новых служб простым присоединением хоста к сети и определением нового протокола прикладного уровня (например, HTTP) позволило включить в Интернет новые службы (например, web) за замечательно короткое время.

Однако, как будет показано в главе 6, в Интернет-сообществе протекают довольно жаркие споры по поводу того, как должна развиваться архитектура сетевого уровня Интернета, чтобы обслуживать службы реального времени, например мультимедиа. Интересное сравнение ориентированной на виртуальные каналы сетевой архитектуры ATM и предлагаемой архитектуры Интернета следующего поколения содержится в [104].

Основы маршрутизации

Для того чтобы переместить пакеты от хоста-отправителя к хосту-получателю, сетевой уровень должен определить *путь,* или *маршрут,* следов^ия пакетов. Независимо от того, какую службу предоставляет сетевой уровень, деитаграммную службу (в этом случае различные пакеты данной пары отправитель-получатель могут двигаться по разным маршрутам) или службу виртуальных каналов (в этом случае все пакеты, передаваемые данным отправителем данному получателю, будут перемещаться по одному и тому же пути), сетевой уровень должен определить путь продвижения пакета. Этим занимается *протокол маршрутизации* сетевого уровня.

Как правило, хост напрямую подключен к одному из маршрутизаторов, так называемому *маршрутизатору по умолчанию,* или *маршрутизатору первого ретрансляционного участка.* Когда хост передает пакет, этот пакет попадает на маршрутизатор по умолчанию. Мы будем называть маршрутизатор по умолчанию хоста-источника *маршрутизатором-источником,* а маршрутизатор хоста-приемника по умолчанию *маршрутизатором-приемником.* Задача выбора пути пакета от хоста-источника к хосту-приемнику, очевидно, сводится к задаче выбора пути пакета от маршрутизатора-источника к маршрутизатору-приемнику. Этой задаче и посвящается данный раздел книги.

Сердцевиной любого протокола маршрутизации является алгоритм, определяющий путь пакета от маршрутизатора-источника к маршрутизатору-приемнику *(алго-*

ритм маршрутизации). Задача алгоритма маршрутизации проста: для заданного множества маршрутизаторов и линий, соединяющих маршрутизаторы, алгоритм маршрутизации находит «оптимальный» путь от маршрутизатора-источника к маршрутизатору-приемнику. Как правило, «оптимальный» означает путь с «минимальной стоимостью». Мы увидим, однако, что на практике в игру часто вступают такие стратегические соображения, как вопросы безопасности (например, такое требование, как «маршрутизатор *X,* принадлежащий организации *Y,* не должен переправлять пакеты, исходящие из сети, принадлежащий организации Z»), усложняя концептуально простые и элегантные алгоритмы, на теории которых покоится практика маршрутизации в современных сетях.

Для формулирования алгоритмов маршрутизации сеть рассматривается как граф (рис. 4.4). Графы, соответствующие некоторым реальным сетям, приводятся в [127]. Обсуждение вопроса, насколько хорошо различные основанные на теории графов модели отражают реальный Интернет, содержится в [564]. Узлы графа представляют маршрутизаторы — точки, в которых принимаются решения о продвижении пакетов, — а линии (в соответствии с терминологией теории графов называемые «ребрами»), соединяющие эти узлы, представляют физические линии между маршрутизаторами. Каждой линии связи соответствует некоторое значение, представляющее «стоимость» пересылки пакета по этой линии. Стоимость может зависеть от физической длины линии (например, стоимость передачи кадра по трансокеанскому кабелю может быть выше, чем по короткому кабелю, проложенному по суше), скорости передачи данных по линии или финансовой стоимости линии. Для наших текущих целей мы просто примем стоимости линий как данное и не станем беспокоиться о том, как они определяются.

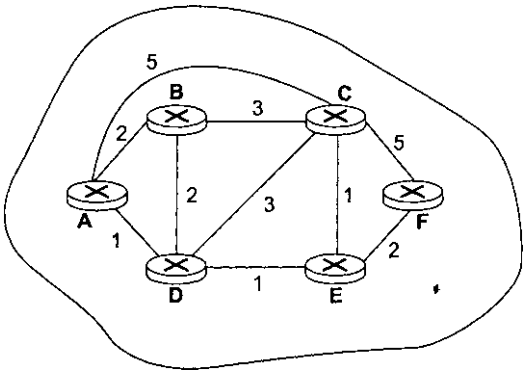


Рис. 4.4. Абстрактная модель сети

При рассмотрении сети в виде графа для решения задачи определения пути от отправителя к получателю с минимальной стоимостью необходимо найти последовательность линий такую, что:

- первая линия пути соединена с источником;
- последняя линия пути соединена с адресатом;
- для всех *i* линии с номерами *г* и *г* - 1 соединены с одним и тем же узлом;

- для *пути с минимальной стоимостью* сумма стоимостей всех линий пути является минимальной по всем возможным путям между отправителем и получателем (обратите внимание, что если все линии имеют одинаковую стоимость, тогда путь с минимальной стоимостью представляет собой также *кратчайший путь,* то есть путь, состоящий из минимального количества линий между отправителем и получателем).

Например, на рис. 4.4 путь с минимальной стоимостью между узлами *A* (отправителем) и *C* (получателем) представляет собой маршрут *ADEC*. (Как мы увидим, путь проще обозначать узлами, входящими в путь, а не ребрами, образующими путь.)

В качестве простого упражнения попытайтесь найти путь с минимальной стоимостью от узла *A* до узла *F.* Скорее всего, вы будете искать путь от узла *A* до узла *F,* изучая рис. 4.4 и пробуя несколько маршрутов от узла Л до узла *F,* каким-то образом определяя, что стоимость выбранного вами пути минимальна из всех возможных путей (вряд ли вам удастся проверить все 17 возможных путей между узлами *A* и *F).* Подобные вычисления представляют собой пример централизованного алгоритма маршрутизации — алгоритм маршрутизации рассчитывался в одном месте (вашем мозгу) при наличии полной информации о сети. Обобщая, скажем, что все алгоритмы маршрутизации можно разбить на два класса: глобальные и децентрализованные.

- *Глобальный алгоритм маршрутизации* находит путь с наименьшей стоимостью от отправителя до получателя с помощью полной информации о сети. Таким образом, в качестве входных данных алгоритм использует информацию о том, какой узел с каким соединен напрямую линией связи, и о стоимости всех линий. Прежде чем начать сами вычисления, данный алгоритм должен каким-то образом получить эту информацию. Сами вычисления могут производиться на каком-либо одном компьютере (централизованный глобальный алгоритм маршрутизации) или тиражироваться в разных местах. Однако ключевой особенностью здесь является то, что глобальный алгоритм обладает *полной* информацией о топологии сети и стоимости линий. На практике алгоритмы, обладающие глобальной информацией о состоянии сети, часто называют *алгоритмами, основанными на состояниях линий,* так как алгоритм должен знать стоимость каждой линии в сети. Мы рассмотрим глобальный алгоритм, основанный на состояниях линий, в подразделе «Алгоритм маршрутизации, основанный на состояниях линий» раздела «Основы маршрутизации».
- В *децентрализованном алгоритме маршрутизации* вычисление пути с наименьшей стоимостью выполняется итерационным распределенным образом. Ни один узел не обладает полной информацией о стоимости всех линий сети. Изначально каждому узлу известна только стоимость напрямую присоединенных к нему линий. Затем, путем итерационных вычислений и обмена информацией с соседними узлами (то есть узлами, находящимися на противоположных концах напрямую присоединенных к нему линий) узел постепенно определяет путь с наименьшей стоимостью до получателя или до группы получателей. Мы рассмотрим децентрализованный алгоритм маршрутизации, известный как *дис-*

танционно-векторный алгоритм в подразделе «Алгоритм дистанционно-векторной маршрутизации» данного раздела. Он называется дистанционно-векторным алгоритмом, потому что узлу никогда не известен весь маршрут от отправителя до получателя. Вместо этого он знает соседа, которому должен переправить пакет, чтобы достичь получателя по пути с наименьшей стоимостью, а также стоимость этого пути.

Кроме того, все алгоритмы маршрутизации можно разделить на *статические* и *динамические*. В статическом алгоритме маршрутизации маршруты изменяются со временем очень медленно, часто в результате вмешательства человека (например, администратор сети может вручную отредактировать таблицу продвижения данных маршрутизатора). Динамический алгоритм может либо запускаться периодически, либо в ответ на изменения топологии или стоимости линий. Хотя динамические алгоритмы обладают большей чувствительностью к изменениям в сети, они также более восприимчивы к таким проблемам, как петли в маршрутах и осцилляция. Данные проблемы будут рассматриваться в подразделе «Алгоритм дистанционно-векторной маршрутизации» данного раздела.

Третий способ классификации алгоритмов маршрутизации определяется по тому, чувствителен ли алгоритм к перегрузке. В *чувствительном к перегрузке алгоритме* стоимости линий динамически изменяются, отражая текущий уровень перегрузки в соответствующей линии. Если с временно перегруженной линией ассоциируется высокая стоимость, алгоритм маршрутизации будет стараться выбирать маршруты в обход перегруженной линии. Первые алгоритмы сети ARPAnet были чувствительными к перегрузке [326], однако попытки использования чувствительных к перегрузке алгоритмов натолкнулись на ряд проблем [220]. Сегодня в Интернете применяются алгоритмы, нечувствительные к перегрузке (такие как RIP, OSPF и BGP), так как стоимость линии, как правило, не отражает ее текущего (или недавнего) уровня перегрузки.

В Интернете, как правило, используются только два типа алгоритмов маршрутизации: динамический глобальный алгоритм, основанный на состояниях линий, и динамический децентрализованный дистанционно-векторный алгоритм. Мы рассмотрим эти алгоритмы соответственно в подразделах «Алгоритм маршрутизации, основанный на состояниях линий» и «Алгоритм дистанционно-векторной маршрутизации», а другие алгоритмы маршрутизации — в подразделе «Другие алгоритмы маршрутизации» данного раздела.

Алгоритм маршрутизации, основанный на состоянии линий

Как уже отмечалось, алгоритм, основанный на состоянии линий (Line State, LS), знает стоимость всех линий, то есть все эти данные можно подать на вход LS-алгоритма. На практике, чтобы собрать эту информацию, каждый узел путем *широковещательной рассылки* отправляет свой идентификатор и стоимости всех присоединенных к нему линий *всем* остальным маршрутизаторам сети. Чтобы выполнить эту операцию [384], узлам не нужно знать идентификаторы всех остальных узлов сети. Узел должен лишь знать идентификаторы своих ближайших соседей, а так-

же стоимости линий, соединяющих его с ними. О топологии остальной сети ему станет известно, когда он получит широковещательные пакеты с информацией о состоянии линий от других узлов. (В главе 5 мы узнаем, как маршрутизатор узнает идентификаторы своих ближайших соседей.) В результате всех этих широковещательных рассылок все узлы сети получают идентичное и полное представление о сети. Затем каждый узел может запустить алгоритм, основанный на состояниях линий, и вычислить пути к каждому из узлов.

Ниже мы рассмотрим пример алгоритма, основанный на состояниях линий, на примере алгоритма Дейкстры, названного по имени его создателя. Близко связан с ним алгоритм Прима (общее описание алгоритмов графов см. в [97]). Алгоритм Дейкстры вычисляет путь с наименьшей стоимостью от одного узла (источника, который мы будем называть узлом Л) до всех остальных узлов сети. Алгоритм Дейкстры является итерационным и после *k* итераций он определяет пути с наименьшей стоимостью до *k* узлов-адресатов, и среди путей с наименьшей стоимостью до всех узлов-адресатов у этих *k* путей будут *k* наименьших стоимостей. Определим систему обозначений:

- *c(i,j)* — стоимость линии от узла *г* до узла/ Если узлы *г nj* не соединены напрямую, тогда *c(ij)* = ° о . Чтобы упростить нашу систему обозначений и примеры, мы будем предполагать, что *c(ij) = c(j, i).* Однако следует отметить, что данный алгоритм будет работать и в случае, когда стоимости *c(i,j)* и *c(j, i)* не равны.
- *D(v)* — стоимость пути от узла-источника до узла-адресата *v,* у которого на данный момент (на текущей итерации алгоритма) стоимость минимальна.
- *p(v)* — предыдущий узел (соседний с узлом *v)* на текущем пути с наименьшей стоимостью от источника до узла *v.*
- *N* — множество узлов, для которых на данной итерации известны пути с наименьшей стоимостью.

Алгоритм, основанный на состоянии линий, состоит из этапа инициализации, за которым следует цикл. Количество итераций этого цикла равно числу узлов в сети. По завершении алгоритм вычислит кратчайший путь от узла-источника (Л) до всех остальных узлов сети.

Листинг 4.1. Алгоритм, основанный на состоянии линий для узла-источника А

```
1 инициализация:
2    N = {A}
3    для всех узлов v
4       если v смежный с A
5          тогда D(v) = c(A.v)
6          иначе D(v) = °o
7
8 цикл
9     найти w, не входящий в N. такой что D(w) минимальна
10    добавить w к N
11    обновить D(v) для всех v. смежных с w и не входящих в N:
12       D(v) = min(D(v). D(w) + c(w.v))
13 /* новая стоимость пути к v равна старой стоимости к v или стоимости
14 известного кратчайшего пути к w плюс стоимость пути от w к v */
15 по всем узлам в N
```

В качестве примера рассмотрим сеть, показанную на рис. 4.4, и найдем пути с минимальной стоимостью от узла *A* до всех возможных адресатов. В табл. 4.2 показаны поэтапные результаты работы алгоритма. В каждой строке таблицы приведены значения переменных алгоритма в конце очередной итерации.

Таблица 4.2. Результаты работы алгоритма, основанного на состоянии линий, для сети с рис. 4.4

Шаг	N	D(B),p(B)	D(C), p(C)	D(D), p(D)	D(E), p(E)	D(F), p(F)
0	A	2, A	5, A	1, A	oo	oo
1	AD	2, A	4, D		2, D	oo
2	ADE	2, A	3,E			4,E
3	ADEB		3,E			4.E
4	ADEBC					4,E
5	ADEBCF					

Рассмотрим подробно несколько первых шагов.

1. *На шаге инициализации* текущие наименьшие стоимости для путей от узла *A* до напрямую соединенных с ним соседних узлов *B, C и D* равны 2, 5 и 1 соответственно. Обратите, в частности, внимание, что стоимость пути до узла *C* установлена равной 5 (хотя вскоре мы увидим, что существует путь с меньшей стоимостью), так как этой величине равна стоимость прямой (один ретрансляционный участок) линии от узла *A* до узла *C*. Стоимости путей до узлов *Ей F*устанавливаются равными бесконечности, так как эти узлы не связаны напрямую с узлом *A*.
2. *На первой итерации* из узлов, не входящих в множество *N,* мы выбираем узел с наименьшим по стоимости путем от узла *A.* Это узел *D* со стоимостью пути 1. Таким образом, узел *D* добавляется к множеству *N* Затем выполняется строка 12 алгоритма, чтобы вычислить новое значение *D(v)* для всех узлов *v* и получить результаты, показанные во второй строке (шаг 1) табл. 4.2. Стоимость пути к узлу *B* не изменилась. Стоимость пути к узлу *C* (которая вначале была равна 5) через узел *D* оказалась равной 4. Таким образом, выбирается этот путь, имеющий наименьшую стоимость, а для узла *C* указывается, что путь с наименьшей стоимостью к нему проходит через узел *D.* Аналогично вычисляется стоимость пути до узла *E* (через узел D), равная 2, что отражается в таблице.
3. *На второй итерации* следующими узлами с путями минимальной стоимости (равной 2) оказываются узлы *B* и *E.* Мы произвольным образом выбираем из двух равноудаленных узлов один (узел *E)* и добавляем его в набор *N* Теперь множество содержит узлы *A,DhE.* Стоимости путей до остальных узлов, еще не вошедших в множество *N,* то есть узлов B, Си *F,* вычисляются заново в строке 12 алгоритма. Результаты этих вычислений показаны в строке 3 табл. 4.2.
4. И т. д.

Когда LS-алгоритм завершает свою работу, для каждого узла мы узнаем узел-предшественник, через который проходит путь с наименьшей стоимостью к данному

узлу. Для каждого узла-предшественника мы также знаем узел-предшественник и т. **д".** Таким образом, мы можем по этой информации построить полный путь от источника до любого адресата, а также сформировать таблицу маршрутизации для узла-источника.

Какова вычислительная сложность этого алгоритма? То есть сколько потребуется произвести вычислений в худшем случае, чтобы найти пути с наименьшей стоимостью от источника до всех *n* адресатов? На первой итерации мы должны просмотреть все *n* узлов, чтобы найти узел *w,* не принадлежащий множеству *N,* стоимость пути до которого минимальна. На второй итерации мы должны просмотреть гг — 1 узлов, чтобы найти узел с путем наименьшей стоимости. На третьей итерации нам придется просмотреть *n* - 2 узлов, и т. д. Всего нам придется просмотреть *n(n* + 1)/2 узлов. Таким образом, количество вычислений в алгоритме, основанном на состоянии линий, растет пропорционально квадрату узлов сети: $0(n^2)$. Существует и более сложный вариант этого алгоритма, в котором используется структура данных, называемая «кучей» (heap). Данному алгоритму для нахождения минимума в строке 9 требуется время, пропорциональное логарифму от числа рассматриваемых узлов, что снижает суммарное время выполнения алгоритма.

Прежде чем завершить наше обсуждение LS-алгоритма, рассмотрим патологическую ситуацию, которая может возникнуть. На рис. 4.5 показана схема простой сети, в которой стоимость линии пропорциональна текущей нагрузке на линию, например, оцениваемой по испытываемой задержке. В данном примере стоимости линий несимметричны, то есть стоимость линии с(Л, *В)* равна *с(В,* Л) только в том случае, если нагрузка на линию *ЛВ* одинакова в обоих направлениях. В данном примере узел *D* передает единичную порцию трафика узлу Л, узел *В* также посылает единичную порцию трафика узлу Л, а узел С посылает узлу Л порцию трафика объема *е.* Начальная схема маршрутизации, в кбторой стоимости линий соответствуют количеству трафика, показана на рис. 4.5, *а.*

Когда в очередной раз запускается LS-алгоритм, узел С определяет (на основании стоимостей линий, показанных на рис. 4.5, *а),* что путь к узлу Л по часовой стрелке обладает стоимостью 1, тогда как стоимость пути к этому же узлу против часовой стрелки (которым он пользовался до сих пбр) равна 1 + *е.* Таким образом, путь наименьшей стоимости от узла *С* к узлу Л теперь имеет направление по часовой стрелке. Аналогично, узел *В* определяет, что его новый путь к узлу Л с наименьшей стоимостью также направлен по часовой стрелке, в результате чего вычисляются новые стоимости путей, показанные на рис. 4.5, *б.* При следующем запуске LS-алгоритма узлы В, *С* и *D* обнаруживают путь нулевой стоимости к узлу Л в направлении против часовой стрелки, поэтому все узлы направляют свой трафик против часовой стрелки. В следующий раз эти же узлы обнаруживают, что дешевле направлять трафик по часовой стрелке, так как в этом направлении никакого трафика нет.

Что можно предпринять, чтобы предотвратить подобный колебательный процесс? Осцилляция может возникнуть в любом алгоритме (не только в LS-алгоритме), использующем уровень загруженности линий или задержку в линиях в качестве

параметра определения стоимости линий. Один из методов решения данной проблемы заключается в том, чтобы стоимость линий не зависела от текущего объема трафика, проходящего по ним. Однако такое решение является неприемлемым, так как наша цель состоит в том, чтобы алгоритм мог пускать потоки данных в обход сильно загруженных линий (например, с высокой задержкой). Другое возможное решение могло бы гарантировать, что не все маршрутизаторы запустят LS-алгоритм в одно и то же время. Это решение кажется более разумным, так как есть надежда, что даже если маршрутизаторы запускают LS-алгоритм с одной и той же периодичностью, исполняемые экземпляры алгоритма не будут одинаковыми на каждом узле. Исследователи недавно отметили, что маршрутизаторы Интернета могут самосинхронизироваться [159]. То есть даже если изначально они выполняют один и тот же алгоритм с одним и тем же периодом, но с разной фазой, со временем алгоритмы на разных узлах синхронизируются и остаются синхронными. Один из способов избежать синхронизации состоит в том, чтобы намеренно сделать случайными интервалы времени между запусками алгоритма на каждом узле.

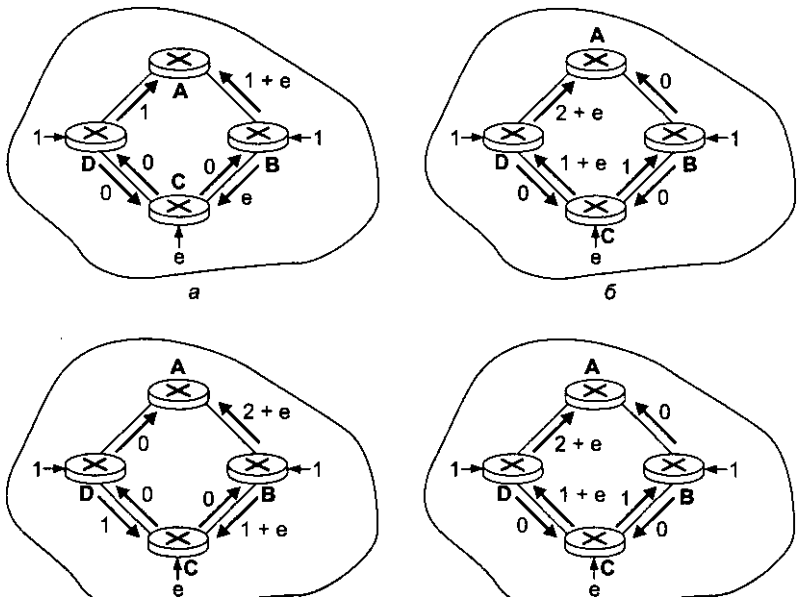


Рис. 4.5. Осцилляция при LS-маршрутизации: начальные маршруты (а); узлы B и C находят лучшие маршруты к узлу A по часовой стрелке (б); узлы B, C и D находят лучшие маршруты к узлу A против часовой стрелки (в); узлы B, C и D находят лучшие маршруты к узлу A по часовой стрелке (г)

Итак, мы обсудили алгоритм маршрутизации, основанный на состоянии линий. Рассмотрим теперь другой важный алгоритм маршрутизации, применяемый сегодня на практике, — алгоритм дистанционно-векторной маршрутизации.

Алгоритм дистанционно-векторной маршрутизации

В отличие от алгоритма, основанного на состоянии линий и использующего глобальную информацию, *дистанционно-векторный* (Distance Vector, DV) алгоритм является *распределенным, итерационным* и *асинхронным*. Он является распределенным, так как каждый узел получает порцию информации от одного или нескольких напрямую соединенных с ним соседей, выполняет вычисления, а затем может разослать результаты своих вычислений своим соседям. Он является итерационным, так как этот процесс продолжается до тех пор, пока соседние узлы не перестанут обмениваться информацией. (Интересно отметить, что, как мы увидим, этот алгоритм сам завершает свою работу — он не получает «сигнала», требующего остановить работу; он просто останавливается.) Алгоритм является асинхронным, так как он не требует, чтобы все узлы работали в жесткой взаимосвязи друг с другом. Как мы увидим, асинхронный, итерационный, самоостанавливающийся, распределенный алгоритм значительно интереснее централизованного!

Главной структурой данных в дистанционно-векторном алгоритме является *таблица расстояний,* содержащаяся на каждом узле. В каждой таблице расстояний есть по строке для каждого адресата в сети и по столбцу для каждого ближайшего соседа. Рассмотрим узел *X,* который заинтересован в маршрутизации к адресату *Y* через ближайшего соседа Z Запись в таблице расстояний $D^{X}(Y,Z)$ узла *X* представляет собой сумму стоимостей прямой линии между узлами *X* и Z, то есть *c(X,Z)* и известного на данный момент соседу Z пути наименьшей стоимости до узла У:

$$D^{X}(Y,Z) = c(X,Z) + mmJD^{z}(Y,w)\}.$$

Второе слагаемое, минимальное значение стоимости пути, определяется по всем ближайшим соседям узла Z (включая, как мы скоро увидим, узел *X).*

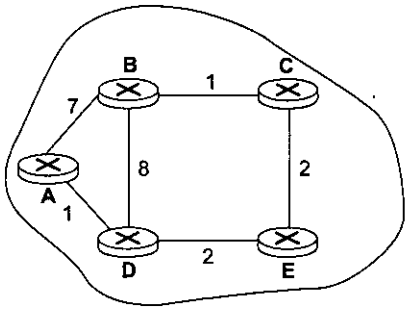
Дистанционно-векторный алгоритм предполагает определенную форму общения между соседями — каждый узел должен знать наименьшую стоимость каждого пути от каждого из его соседей до каждого адресата. Таким образом, всегда, когда узел вычисляет новую минимальную стоимость до какого-либо узла, он должен сообщить об этом своим соседям.

Прежде чем рассматривать дистанционно-векторный алгоритм, рассмотрим пример, который поможет нам понять смысл записей в таблице расстояний. Соответствующие топология сети и таблица расстояний для узла *E* показаны на рис. 4.6. Эта таблица расстояний получена узлом *E* после схождения алгоритма.

- Обратите внимание на столбец для адресата Л. Очевидно, путь с наименьшей стоимостью до узла Л идет по прямой линии, соединяющей узлы £ и Л. Стоимость такого пути равна 1. Таким образом, $D^{E}(A,A) = 1$.
- Рассмотрим теперь значение $D^{E}(A,D)$ — стоимость пути от узла £ до узла Л, проходящего через узел *D.* В этом случае запись в таблице расстояний представляет собой стоимость пути от узла *E* до узла *D* (2) плюс минимальную стоимость пути узла *D* до узла Л. Обратите внимание, что минимальная стоимость пути узла *D* до узла Л равна 3. Это путь, проходящий снова через узел *E*\ Тем не

менее мы отмечаем, что минимальная стоимость пути от узла *E* до узла Л, проходящего через узел Д равна 5. У нас, однако, остается подозрение,'что этот путь «зацикливается», проходя дважды через узел *E,* и может стать источником проблем в дальнейшем.

- Аналогично, мы можем определить, что стоимость пути от узла £до узла Л, проходящего через узел *B,* равна 14. Обратите внимание, что стоимость этого пути равна именно 14, а не 15. (Почему?)



Стоимость пути до адресата через

Адресат	$D^E()$	A	B	D
A	ф	14	5	
B	7	8	®	
C	6	9	**0**	
D	4	11	(2)	

Рис. 4.6. Таблица расстояний для узла-источника E

Кружками в таблице расстояний отмечены минимальные значения стоимости пути к данному адресату. Столбец с отмеченным кружком значением указывает следующий узел на пути с наименьшей стоимостью. Таким образом, из таблицы расстояний узла несложно построить *таблицу продвижения данных* (таблицу маршрутизации), в которой указывается, по какой линии следует посылать пакет каждому адресату.

Листинг 4.2. Дистанционно-векторный алгоритм (на каждом узле X)

```
1 инициализация:
2    для всех смежных узлов v:
3        D^X(*,v) = oo  /* оператор * означает "для всех рядов" */
4        D^X(v,v) = c(X.v)
5    для всех адресатов, y
6        послать minJKy.w) каждому соседу /* w по всем соседям узла X */
7
8 цикл
9    ждать (пока не изменится стоимость линии связи с соседом V
10           или пока не будет получено обновление от соседа V)
11
12   если (стоимость c(X.V) изменилась на d)
13      /* изменить стоимость путей ко всем адрсатам через соседа v на d */
14      /* величина d может быть положительной или отрицательной */
15      для всех адресатов y: D^X(y.V) = D^X(y.V) + d
16
17   иначе если (получено обновление от V, адресат Y)
18              /* изменился кратчайший путь от V до некоторого Y */
19              /* Узел V послал свое новое значение min_w D^V(Y.w) */
20              /* назовем это новое полученное значение "newval" */
21              для одного адресата y: D^X(Y.V) = c(X.V) + newval          продолжение &
```

Листинг 4.2 *(окончание)*

```
22
23   если мы получаем новое значение min_w D^X(Y,w) для любого адресата Y
24        послать новое значение min_w D^X(Y.w) всем соседям
25
26 конец цикла
```

При обсуждении записей таблицы расстояний узла *E* мы рассмотрели сеть, для которой нам были известны стоимости всех линий. Дистанционно-векторный алгоритм, который мы сейчас рассмотрим, является *децентрализованным* и не пользуется подобной глобальной информацией. В самом деле, единственной информацией, которой обладает узел, являются стоимости линий, связывающих его с ближайшими соседями, а также сведения, получаемые им от этих ближайших соседей. Дистанционно-векторный алгоритм, который мы сейчас рассмотрим, также называют алгоритмом Беллмана-Форда, по именам его изобретателей. Он применяется на практике во многих протоколах маршрутизации, включая протоколы Интернета RIP и BGP, а также ISO IDRP, Novell IPX и оригинальный протокол сети ARPAnet.

Ключевыми являются строки 15 и 21 алгоритма, в которых узел обновляет свои записи в таблице расстояний либо в ответ на изменение стоимости присоединенной к узлу линии, либо в ответ на получение сообщения об обновлении от соседнего узла. Другим ключевым шагом является строка 24, в которой узел посылает обновленные данные своим соседям, если путь наименьшей стоимости до некоторого адресата изменился.

Рисунок 4.7 иллюстрирует работу дистанционно-векторного алгоритма для простой сети из трех узлов, изображенной в верхней части рисунка. Алгоритм функционирует синхронно, то есть все узлы одновременно получают сообщения от своих соседей, .вычисляют новые значения записей таблицы расстояний и информируют своих соседей о любых изменениях в их путях с наименьшей стоимостью. Когда вы изучите этот пример, вам придется поверить, что данный алгоритм столь же корректно работает и в асинхронном режиме, когда производимые узлами вычисления и обмен данными между узлами происходят в произвольные моменты времени.
Обведенные кружками записи в таблицах расстояний на рисунке показывают текущие минимальные значения стоимости пути до адресата. Запись, обведенная двойным кружком, означает, что была вычислена новая минимальная стоимость (в строке 4, 15 или 21 дистанционно-векторного алгоритма). Те случаи, когда соседям посылается сообщение об изменении стоимости (строка 24 дистанционно-векторного алгоритма), отмечены на рисунке стрелками. В самом левом столбце на рисунке показаны записи таблиц расстояний для узлов *X, Y* и Z после первого шага инициализации.

Рассмотрим теперь, как узел *X* вычисляет таблицу расстояний, показанную в средней колонке, после получения обновлений от узлов Ки Z. Получив обновления от узлов *Yn* Z, узел *X* выполняет строку 21 дистанционно-векторного алгоритма:

$$D^X(Y]Z) = c(X,Z) + \min, D^7(Y,w) = 7 + 1 = 8,$$
$$D^X(Z,Y) = \quad Y) + \min, D^Y(Z,w) = 2 + 1 = 3.$$

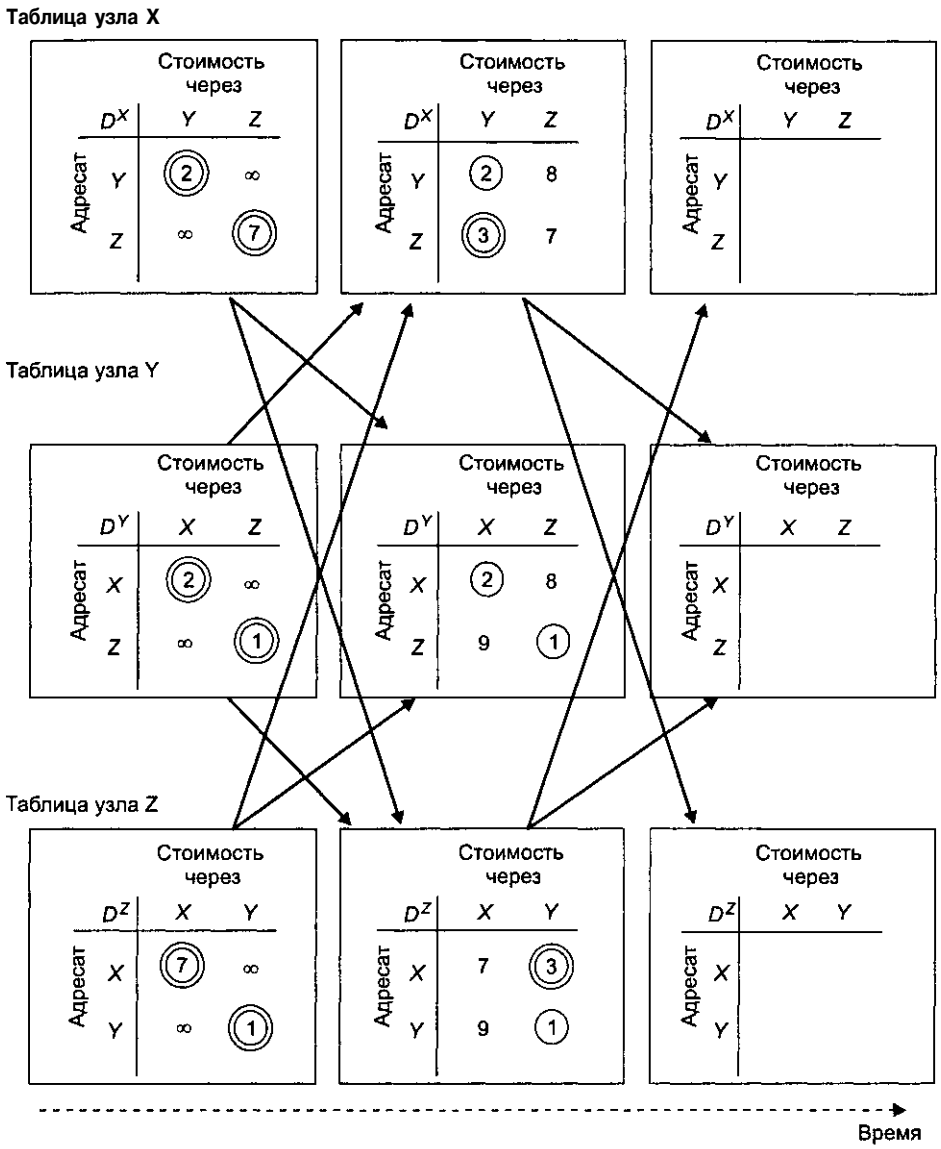


Рис. 4.7. Пример дистанционно-векторного алгоритма

Важно отметить, что узел *X* узнает о значениях min, $D^Z(Y,w)$ и min, $D^Y(Z,w)$ только потому, что узлы У и Z посылают ему эти значения (их узел *X* получает в строке 10 дистанционно-векторного алгоритма). В качестве упражнения проверьте таблицы расстояний, вычисленные узлами Уи Z**b** средней колонке (см. рис. 4.7).

Значение $D^X(Z,Y) = 3$ во второй таблице расстояний узла *X* означает, что наименьшая стоимость пути от узла *X* до узла Z изменилась с 7 до 3. Таким образом, узел *X* посылает новое значение стоимости пути до узла Z узлам У и Z, информируя их об этом. Обратите внимание, что узлу Хне нужно посылать обновления узлам Уи Z о стоимости пути к узлу У, так как она не изменилась. Также обратите внимание, что хотя в результате новых расчетов узел У получает новые значения элементов таблицы расстояний, значения минимальных стоимостей путей к узлам *X* и Z не изменяются. Поэтому узел Уне посылает обновлений узлам *X* и Z.

Процесс получения от соседей новой информации о стоимости путей, вычисления новых табличных значений и извещения своих соседей об изменениях в стоимости путей продолжается до тех пор, пока стоимости не перестанут изменяться. С этого момента алгоритм становится статическим, поскольку сообщения с новыми значениями более не посылаются и значения в таблице не пересчитываются; то есть все узлы переходят в состояние ожидания в строке 9. Алгоритм остается в статическом состоянии до тех пор, пока стоимость какой-либо линии не изменится.

Дистанционно-векторный алгоритм при изменении стоимостей и неисправностях линий

Когда узел, на котором работает дистанционно-векторный алгоритм, обнаруживает изменение стоимости линии, направляющейся от него к соседу (строка 12), он обновляет свою таблицу расстояний (строка 15) и, если наименьшая стоимость пути изменяется, он извещает об этом своих соседей (строки 23 и 24). Эту ситуацию иллюстрирует рис. 4.8. В данном примере стоимость линии от узла *X* ***r q*** узла У изменяется с 4 до 1. Здесь мы рассматриваем только записи таблиц расстояний узлов У и Z, содержащие стоимости путей до узла *X*.

1. В момент времени t^0 узел У замечает изменение стоимости линии (с 4 до 1) и информирует об этом своих соседей, так как благодаря изменению стоимости линии меняются минимальные стоимости путей.
2. В момент времени t^1 узел Z получает сообщение от узла У и обновляет свою таблицу. Затем он вычисляет новое значение минимальной стоимости маршрута до узла *X* (она уменьшилась с 5 до 2) и извещает об этом своих соседей.
3. В момент времени t^2 узел У получает сообщение от узла Z и обновляет свою таблицу. Его значения минимальных стоимостей не изменились (хотя стоимость пути до узла *X* через узел Z изменилась), поэтому узел Уне посылает сообщений своим соседям. Алгоритм переходит в стационарное состояние.

В примере на рис. 4.8 дистанционно-векторному алгоритму требуется только две итерации, чтобы перейти в стационарное состояние. «Хорошие новости» об изменившейся стоимости линии между узлами *X* и У быстро распространилась по сети.

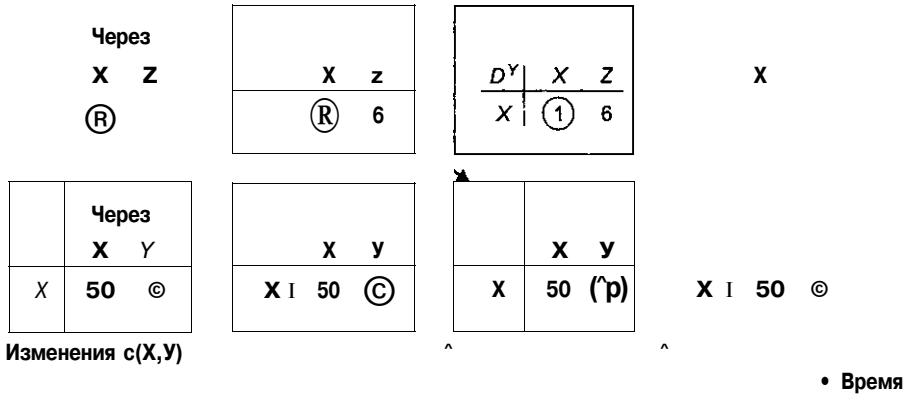


Рис. 4.8. Изменение стоимости линии: добрые вести распространяются быстро

Рассмотрим теперь, что произойдет в случае *увеличения* стоимости линии. Предположим, что стоимость линии между узлами *X* и У возросла с 4 до 60, как показано на рис. 4.9.

1. В момент времени t^0 узел У замечает изменение стоимости линии (с 4 до 60). Узел У вычисляет, что новая наименьшая стоимость пути до узла *X* равна 6. Этот путь проходит через узел Z. Поскольку мы можем видеть всю сеть сразу (глобально), нам понятно, что это новое значение *неверно*. Но узлу У известно лишь то, что стоимость его прямого соединения с узлом *X* равна 60 и что узел Z в последний раз сообщил узлу У, что у узла Zесть путь к узлу *X* стоимостью 5. Узел У решает, что теперь ему дешевле посылать пакеты узлу *X* через узел Z. Таким образом, в момент времени t^x у нас образовалась *маршрутная петля* — узел У направляет пакеты для узла *X* через узел Z, а узел Z направляет пакеты для узла *X* через узел У. Маршрутная петля подобна черной дыре — пакет, предназначенный узлу *X,* будет «отфутболиваться» узлами У и Z друг другу до тех пор, пока таблицы маршрутизации на этих узлах не изменятся.
2. Так как узел У вычислил новый путь наименьшей стоимости до узла *X,* он информирует об этом узел Z в момент времени t^1.
3. Спустя некоторое время узел Z получает пакет с новой наименьшей стоимостью пути от узла У до узла *X* (узел У сообщил узлу Z, что новое значение наименьшей стоимости пути от узла У до узла *X* равно 6). Узел Z знает, что он может добраться до узла У по линии стоимости 1, и вычисляет новое значение стоимости маршрута до узла *X* (все также через узел У), равное 7. Поскольку

наименьшая стоимость пути от узла Z до узла *X* увеличилась, узел Z извещает об этом узел *Yb* момент времени t^2.

4. Аналогичным образом узел *Y* обновляет свою таблицу и информирует узел *Z* о том, что новое значение минимальной стоимости равно 8. Затем узел Z обновляет свою таблицу и информирует узел Fo том, что новое значение минимальной стоимости равно 9, и т. д.

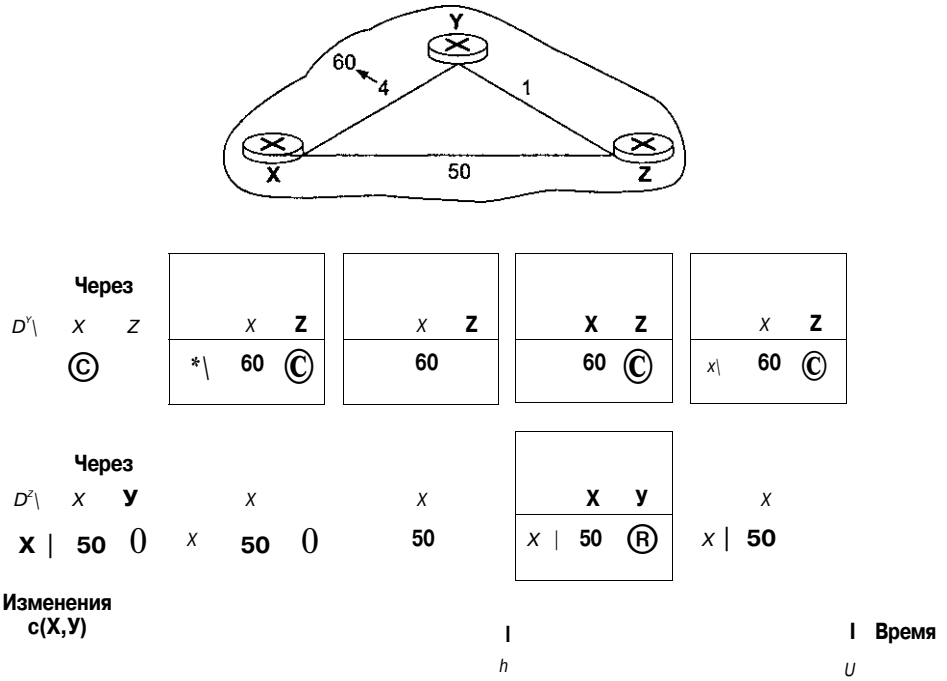


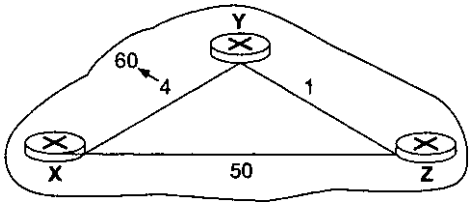
Рис. 4.9. Изменение стоимости линии: дурные вести распространяются медленно и приводят к образованию маршрутных петель

Как долго будет продолжаться этот процесс? Вы можете убедиться в том, что маршрутная петля будет сохраняться в течение 44 итераций (необходимых узлам 7и Zдля обмена сообщениями) — пока, наконец, узел Z не определит, что стоимость пути до узла *X* через узел *Y*больше 50. В этот момент узел Z (наконец-то!) выяснит, что путь наименьшей стоимости к узлу *X* пролегает по прямой линии с узлом *X*. Таким образом, «дурным вестям» об увеличении стоимости линии между узлами *Xu* Гдля распространения по сети потребовалось очень много времени! Что бы произошло, если бы стоимость линии с(*Y,X)* увеличилась с 4 до 10 000, а стоимость линии с(Z,X) была бы равна 9999? Подобная проблема иногда называется проблемой «счета до бесконечности».

Дистанционно-векторный алгоритм и обратная коррекция

Обсуждавшегося выше сценария со счетом до бесконечности (см. рис. 4.9) можно избежать, если использовать метод, называемый *обратной коррекцией,* или *«от-*

равлением» обратного пути (poisoned reverse). Идея этого метода проста — если узел Z направляет пакеты узлу *X* через узел У, тогда узел Z объявит узлу *Y,* что его (узла Z) расстояние до узла *X* равно бесконечности. Узел Z будет продолжать говорить узлу *Y* эту «маленькую ложь» до тех пор, пока узел Z направляет пакеты узлу *X* через узел *Y.* Поскольку узел *Y* полагает, что у узла Z нет пути к узлу X, узел Уникогда не станет пытаться посылать пакеты узлу *X* через узел Z, пока узел Z продолжает посылать пакеты узлу *X* через узел *Y* (и лгать о том, что у него нет пути к узлу *X).*



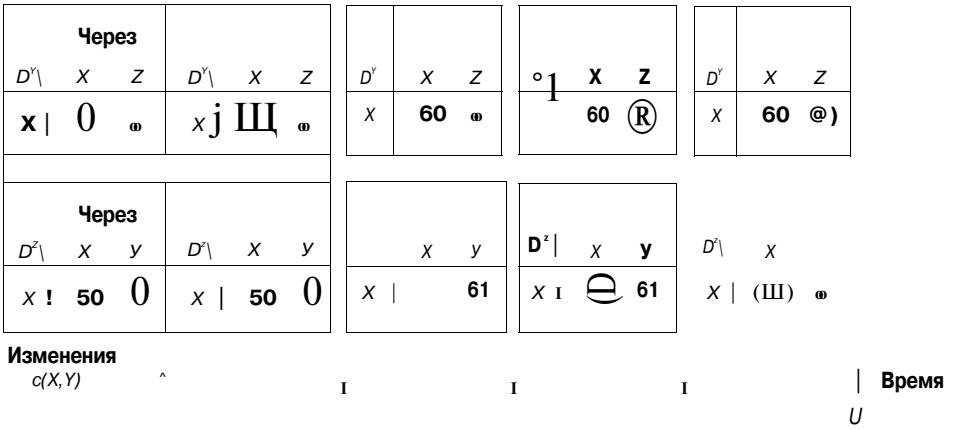


Рис. 4.10. Обратная коррекция

На рис. 4.10 показано, как метод обратной коррекции решает проблему маршрутных петель, с которой мы столкнулись на рис. 4.9. В результате обратной коррекции в таблице расстояний узла *Y* в записи стоимости пути до узла *X* через узел Z указана бесконечность (так как узел Z сообщил узлу *Y,* что расстояние от узла Z до узла *X* равно бесконечности). Когда в момент времени t^0 стоимость линии *XY* увеличивается с 4 до 60, узел У обновляет свою таблицу, но продолжает направлять пакеты узлу *X* напрямую, несмотря на увеличившуюся стоимость маршрута, а также информирует узел Z об изменении стоимости. Получив это извещение в момент времени t^u узел Z немедленно изменяет свой маршрут к узлу *X* на прямую линию *ZX* стоимости 50. Поскольку маршрут к узлу *X* изменился, и этот новый маршрут более не проходит через узел F, узел Z сообщает узлу *Y* о своем новом маршруте в момент времени t^2. Получив эту информацию, узел *Y* обновляет свою таблицу расстояний, изменяя свой маршрут к узлу *X.* Теперь этот

маршрут стоимости 51 проходит через узел Z. Кроме того, поскольку через узел *Z* теперь проходит путь наименьшей стоимости от узла Fk узлу X, узел *Y* «отравляет» обратный путь от узла Z до узла *X,* сообщая в момент времени t^3 узлу Z, что стоимость пути от узла 7до узла *X* равна бесконечности. Этот алгоритм успокаивается в момент времени £„ со значениями таблицы расстояний для адресата *X)* показанными в правом столбце.

Позволяет ли метод «отравления» обратного пути решить проблему счета до бесконечности в общем случае? Нет. Вы можете убедиться сами, что петли, состоящие из трех и более узлов (а не просто из двух смежных узлов, какие мы видели на рис. 4.10), метод обратной коррекции распознать не сможет.

Сравнение алгоритмов маршрутизации

Перед рассмотрением других алгоритмов маршрутизации дадим краткое сравнение некоторых характеристик алгоритма, основанного на состоянии линий, и дистанционно-векторного алгоритма.

- *Сложность сообщений.* Как мы видели, алгоритм, основанный на состоянии линий, требует от каждого узла знания стоимости каждой линии сети. Для этого необходимо отправить *0(nE)* сообщений, где *n* представляет собой количество узлов сети, а *E* — число линий. Кроме того, каждый раз, когда стоимость линии изменяется, об этом следует известить *все* узлы. Дистанционно-векторный алгоритм требует обмена сообщениями только между напрямую соединенными узлами на каждой итерации. Как было показано, время, необходимое для схождения алгоритма, может зависеть от многих факторов. Когда изменяется стоимость линии, дистанционно-векторный алгоритм распространяет результаты только в том случае, если это изменение приводит к изменению пути с наименьшей стоимостью для одного из узлов, присоединенного к этой линии.
- *Скорость схождения.* Как мы видели, количество вычислений в нашей реализации алгоритма, основанного на состоянии линий, растет пропорционально квадрату узлов сети, требуя передачи *0(nE)* сообщений. Дистанционно-векторный алгоритм может сходиться медленно (в зависимости от относительной стоимости путей, как было показано в примере на рис. 4.10), и во время схождения могут образовываться маршрутные петли. Кроме того, дистанционно-векторный алгоритм страдает от «приступов» счета до бесконечности.
- *Живучесть.* Что может случиться, если маршрутизатор выйдет из строя, «сойдет с ума» или объявит забастовку? В алгоритме маршрутизации, основанном на состоянии линий, маршрутизатор может передать всем остальным маршрутизаторам неверные сведения о стоимости одной из присоединенных к нему линий. Узел может также повредить или потерять один из широковещательных пакетов LS-алгоритма, который он получил. Но узел рассчитывает только собственную таблицу продвижения данных. Остальные узлы сами вычисляют свои таблицы. Это означает, что в алгоритме, основанном на состоянии линий, расчеты маршрутов выполняются в значительной степени

раздельно, что предоставляет определенную степень живучести. В случае дистанционно-векторного алгоритма узел может передать другим узлам неверно сосчитанные им значения минимальной стоимости путей. (Такие ситуации встречаются на практике. В 1997 году неисправный маршрутизатор, принадлежащий небольшой компании, занимающейся предоставлением доступа в Интернет, снабжал маршрутизаторы национальной магистрали ошибочной информацией о маршрутах. Это привело к тому, что другие маршрутизаторы завалили трафиком неисправный маршрутизатор, в результате большие фрагменты Интернета в течение нескольких часов были отрезаны [355].) Обратите внимание, что в дистанционно-векторном алгоритме на каждой итерации результаты вычислений узла непосредственно передаются соседнему узлу, а затем на следующей итерации они попадают к соседу соседа и т. д. Таким образом, в дистанционно-векторном алгоритме некорректно вычисленные данные могут распространиться по всей сети.

В заключение скажем, что ни один алгоритм нельзя считать «победителем». Как мы увидим в разделе «Маршрутизация в Интернете», в Интернете применяются оба эти алгоритма.

Другие алгоритмы маршрутизации

Рассмотренные нами дистанционно-векторный алгоритм и алгоритм, основанный на состоянии линий, представляют собой не просто популярные алгоритмы маршрутизации. По сути, только эти два алгоритма и применяются на практике. Тем не менее за последние 30 лет исследователями было предложено множество других алгоритмов маршрутизации, варьирующихся от крайне простых до очень сложных. Один из простейших предложенных алгоритмов назывался *маршрутизацией «горячей картофелины»*. Своим названием алгоритм обязан поведению маршрутизаторов — каждый маршрутизатор пытается как можно быстрее избавиться (переслать дальше) от пакета данных, передавая его по *любой* неперегруженной выходной линии, не обращая внимания на то, куда направляется эта линия. В основе другого широкого класса алгоритмов маршрутизации лежит точка зрения на пакетный трафик, как на потоки данных между отправителями и получателями. При таком подходе проблема выбора маршрута может быть сформулирована математически как задача оптимизации при ограничениях, известная как задача сетевых потоков [27]. Еще одно множество алгоритмов маршрутизации, о которых следует сказать здесь несколько слов, обязаны своим происхождением телефонным сетям. Эти *алгоритмы маршрутизации коммутируемых каналов* представляют интерес для сетей с коммутацией пакетов в ситуациях резервирования ресурсов линии для каждого соединения, таких как пропускная способность части линии связи или объем буферов. И хотя формулировка задачи маршрутизации значительно отличается от ее формулировки для случая определения маршрутов наименьшей стоимости, у подобных алгоритмов маршрутизации есть довольно много общего, по крайне мере в том, что касается алгоритмов определения маршрутов. Более подробную информацию об исследованиях в области маршрутизации коммутируемых каналов можно найти в [16, 181, 433].

Иерархическая маршрутизация

В предыдущем разделе мы рассматривали сеть просто как множество соединенных друг с другом маршрутизаторов. Один маршрутизатор ничем не отличался от других маршрутизаторов в том смысле, что на всех маршрутизаторах работал один и тот же алгоритм маршрутизации для расчета маршрутов через всю сеть. Однако данная модель с однородным набором маршрутизаторов является упрощением и не применяется на практике по двум причинам.

- *Масштабирование.* Когда количество маршрутизаторов становится очень большим, накладные расходы на вычисление, хранение данных и обмен данными о маршрутах (такими как обновление состояния линий или изменения путей наименьшей стоимости) между маршрутизаторами становятся неприемлемыми. Сегодняшний Интернет состоит из сотен миллионов хостов. Хранение информации о маршрутах на каждом из этих хостов потребовало бы памяти огромных размеров. Накладные расходы на широковещательную рассылку пакетов с информацией о состоянии линий между всеми маршрутизаторами Интернета просто не оставили бы пропускной способности для пакетов с данными! А дистанционно-векторный алгоритм при таком огромном количестве узлов сети никогда бы не достиг схождения! Очевидно, необходимо было что-то предпринять, чтобы упростить задачу вычисления маршрутов в такой огромной сети, как Интернет.
- *Административная автономия.* Хотя инженеры часто игнорируют такие вопросы, как пожелания компаний управлять маршрутизаторами, как им хочется (например, использовать алгоритм маршрутизации по своему выбору), или скрыть внутреннюю организацию сети от внешних наблюдателей, важность подобных вопросов может быть высокой. В идеальном случае организация должна иметь возможность управлять своей сетью так, как ей заблагорассудится, не теряя при этом возможности соединения своей сети с «внешним» миром.

Обе эти задачи могут быть решены путем объединения маршрутизаторов в отдельные области, называемые *автономными системами* (Autonomous System, AS). Маршрутизаторы в пределах одной автономной системы используют один и тот же алгоритм маршрутизации (например, дистанционно-векторный алгоритм или алгоритм, основанный на состоянии линий) и обладают информацией обо всех маршрутизаторах своей автономной системы, как это было в рассмотренном ранее идеальном случае. Алгоритм маршрутизации, используемый внутри автономной системы, называется *протоколом внутренней маршрутизации.* Разумеется, необходимо соединить автономные системы друг с другом, и поэтому один или несколько маршрутизаторов в автономной системе будет отвечать за пересылку пакетов за пределы автономной системы. Эти маршрутизаторы называются *шлюзовыми маршрутизаторами,* или просто *шлюзами.* Чтобы шлюзовые маршрутизаторы могли направлять пакеты из одной автономной системы в другую (возможно, пересекая множество других автономных систем), шлюзы должны знать, как определить маршруты друг к другу. Алгоритм маршрутизации, в котором для определения

маршрутов между различными автономными системами применяют шлюзы, называется *протоколом внешней маршрутизации*.

Таким образом, задачи масштабирования и административного управления решаются путем выделения автономных систем. В пределах автономной системы все маршрутизаторы используют один и тот же протокол внутренней маршрутизации. Для определения маршрутов между автономными системами на специальных шлюзовых маршрутизаторах используется протокол внешней маршрутизации. Задача масштабирования решается благодаря тому, что внутренний маршрутизатор автономной системы должен знать только о маршрутизаторах его автономной системы, а также о шлюзовых маршрутизаторах его автономной системы. Задача административного управления решается благодаря тому, что организация может использовать любой протокол внутренней маршрутизации, какой она пожелает, при условии, что этот протокол поддерживается шлюзами автономной системы, обеспечивающими соединение данной автономной системы с другими автономными системами.

Данный сценарий иллюстрирует рис. 4.11. Здесь изображены три автономные системы, A, B и C. В автономной системе A имеется четыре маршрутизатора, A.a, A.b, A.c и A.d, на которых работает протокол внутренней маршрутизации, используемый в пределах автономной системы A. Эти четыре маршрутизатора обладают полной информацией о маршрутах внутри автономной системы A. Аналогично в автономных системах B и C есть три и два маршрутизатора соответственно. Обратите внимание, что протоколы внутренней маршрутизации, применяемые в автономных системах A, B и C, могут быть разными. Шлюзовыми маршрутизаторами являются A.a, A.c, B.a и C.b. Помимо протоколов внутренней маршрутизации на этих четырех маршрутизаторах работает протокол внешней маршрутизации. Топология протокола внешней маршрутизации показана на верхнем уровне сплошными линиями. Обратите внимание, что линия связи высокого уровня может быть как физической линией (это, например, линия, соединяющая маршрутизаторы A.a и B.a), так и логической линией (соединяет маршрутизаторы A.c и A.a). На рисунке также показано, что на шлюзовом маршрутизаторе A.c должен работать как протокол внутренней маршрутизации для связи с соседями по автономной системе A.a и A.d, так и протокол внешней маршрутизации для связи со шлюзовым маршрутизатором B.a. Обратите внимание, что записи в таблице продвижения данных маршрутизатора A.c сформированы обоими протоколами (и внутренней, и внешней маршрутизации).

Предположим, хосту H1, соединенному с маршрутизатором A.d, нужно послать пакет хосту H2 в автономной системе B (рис. 4.12). Предполагая, что в таблице продвижения данных маршрутизатора A.d указано, что за отправку пакетов за пределы автономной системы отвечает маршрутизатор A.c, маршрутизатор A.d посылает первый пакет маршрутизатору A.c при помощи протокола внутренней маршрутизации. Важно отметить, что маршрутизатору A.d не известна внутренняя структура автономных систем B и C, более того, он даже не должен знать топологию, соединяющую автономные системы A, B и C. Маршрутизатор A.c получает пакет и видит, что он предназначается хосту, находящемуся за пределами автономной системы A. В таблице продвижения данных маршрутизатора A.c для про-

токола внешней маршрутизации указано, что пакет, направляющийся в автономную систему B, должен быть передан маршрутизатору B.a. Когда пакет прибывает на маршрутизатор B.a, протокол внешней маршрутизации видит, что пакет предназначается автономной системе B. Поэтому пакет «передается» протоколу внутренней маршрутизации, который направляет пакет его конечному адресату, хосту H2. Для этого протокол внутренней маршрутизации автономной системы B создает записи в таблице продвижения данных маршрутизатора B.a, чтобы указать следующий ретрансляционный участок на пути к хосту H2. На рисунке пунктиром на нижнем уровне показаны фрагменты пути, рассчитанные с помощью протоколов внутренней маршрутизации автономных систем A и B, а часть пути, пройденного при помощи протокола внешней маршрутизации, изображена сплошной линией на верхнем уровне.

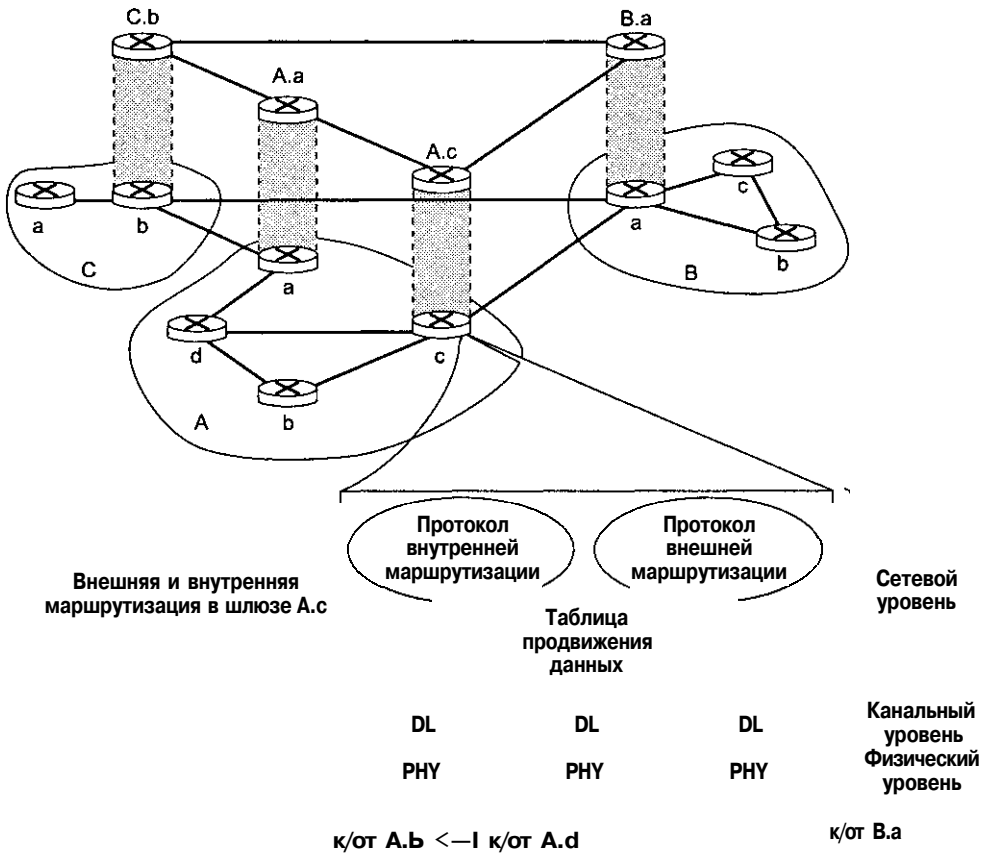


Рис. 4.11. Внутренняя и внешняя маршрутизация

В разделе «Маршрутизация в Интернете» мы изучим два протокола внутренней маршрутизации (RIP и OSPF), а также протокол внешней маршрутизации (BGP), завершив рассмотрение темы иерархической маршрутизации (эти протоколы применяются сегодня в Интернете).

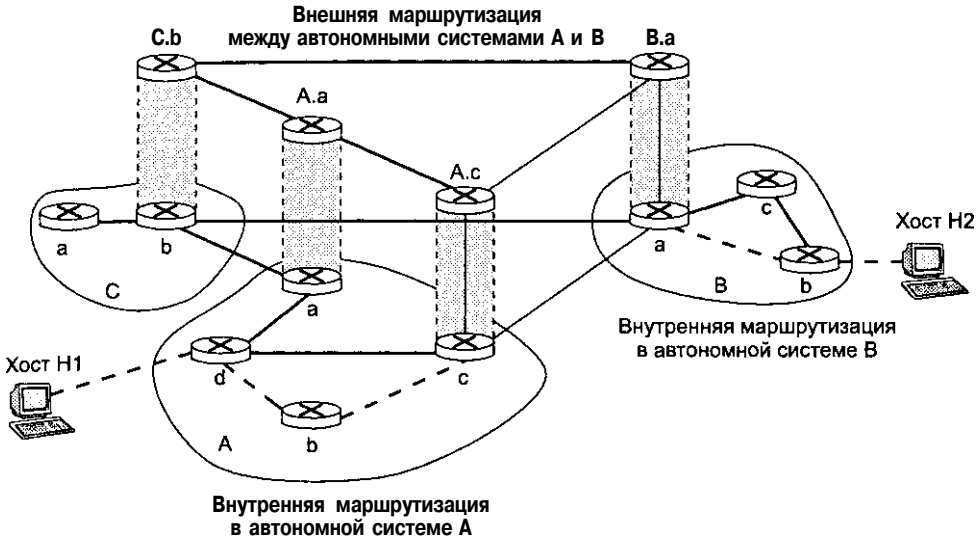


Рис. 4.12. Маршрут от A.d до B.b: внутренняя и внешняя маршрутизация

Интернет-протокол

До сих пор в этой главе мы обсуждали основополагающие принципы сетевого уровня вне контекста какой-либо конкретной сетевой архитектуры. Мы рассмотрели различные модели обслуживания сетевого уровня, популярные алгоритмы маршрутизации для определения маршрута от отправителя к получателю, а также использование иерархической структуры для решения задачи масштабирования. В этом разделе мы обратим наше внимание на сетевой уровень Интернета, часто называемый IP-уровнем (по названию протокола IP). Однако мы увидим, что сам протокол IP представляет собой лишь часть (хотя и очень важную) сетевого уровня Интернета.

Как показано на рис. 4.13, сетевой уровень Интернета состоит из трех основных компонентов.

- Первый компонент представляет собой протокол сетевого уровня, определяющий адресацию сетевого уровня, формат полей дейтаграммы и действия над дейтаграммой, предпринимаемые маршрутизаторами и оконечными системами на основании значений этих полей. Сетевой протокол в Интернете называется Интернет-протоколом, а чаще *протоколом IP* (Internet Protocol). Сегодня используются две версии протокола IP. В этом разделе мы обсудим широко распространенную версию 4, обычно называемую IPv4 (RFC 791). В разделе «Протокол IPv6» мы изучим версию 6 (RFC 2373, RFC 2460), которая должна в будущем заменить IPv4.
- Второй главной составляющей сетевого уровня является компонент определения пути. Он выбирает маршрут, по которому следует дейтаграмма от отправителя к получателю. Как было показано в разделе «Основы маршрутизации», протоколы маршрутизации вычисляют таблицы продвижения данных, используемые для маршрутизации пакетов в сети. Мы обсудим компонент определения маршрута в Интернете в разделе «Маршрутизация в Интернете».

- Наконец, третий компонент сетевого уровня должен уметь сообщать об ошибках в дейтаграммах и отвечать на запросы определенной информации сетевого уровня. Эту задачу в Интернете решает протокол ICMP (Internet Control Message Protocol — протокол управляющих сообщений Интернета), который мы рассмотрим в конце этого раздела.

Рис. 4.13. Сетевой уровень Интернета

Адресация в протоколе IPv4

Начнем наше изучение протокола IPv4 с вопроса адресации. Хотя этот вопрос может показаться довольно простым и, возможно, скучным, связь адресации и маршрутизации на сетевом уровне является одновременно очень важной и трудноуловимой. Превосходное описание адресации в протоколе IPv4 можно найти в [457,488].

Однако прежде, чем перейти к обсуждению IP-адресации, мы должны сказать несколько слов о том, как хосты и маршрутизаторы объединяются в сети. Как правило, хост соединен с сетью единственной линией. Когда протоколу IP хоста необходимо послать дейтаграмму, он пользуется этой линией. Между физической линией и хостом располагается *интерфейс.* Маршрутизатор принципиально отличается от хоста. Поскольку работа маршрутизатора заключается в получении дейтаграммы по «входной» линии и отправке ее по одной из «выходных» линий, маршрутизатор должен быть присоединен к двум или более линиям. Границы между маршрутизатором и его линиями также называются интерфейсами. Таким образом, у маршрутизатора

несколько интерфейсов, по одному для каждой линии. Поскольку отправлять и принимать IP-дейтаграммы может каждый хост и каждый маршрутизатор, протокол IP требует, чтобы у интерфейса каждого хоста и у интерфейса каждого маршрутизатора был собственный IP-адрес. Таким образом, IP-адрес технически ассоциируется с интерфейсом, а не хостом или маршрутизатором, которому принадлежит интерфейс.

Каждый IP-адрес представляет собой 32-разрядное число (четыре байта), поэтому всего может быть 2^{32} IP-адресов. Как правило, эти адреса изображаются в виде четырех десятичных чисел, разделенных точками. Каждое десятичное число соответствует одному байту адреса, например 193.32.216.9. В двоичном виде этот же адрес будет выглядеть так:

11000001 00100000 11011000 00001001

У интерфейсов всех хостов и маршрутизаторов в Интернете должны быть уникальные IP-адреса. Поэтому эти адреса не могут выбираться произвольным образом. Часть IP-адреса интерфейса определяется «сетью», с которой он соединен. В данном контексте термин «сеть» не имеет отношения к общей инфраструктуре хостов, маршрутизаторов и линий связи, образующих сеть. Как мы скоро увидим, данный термин имеет очень точное значение, тесно связанное с IP-адресацией.

На рис. 4.14 показаны примеры IP-адресов и интерфейсов. На этом рисунке один маршрутизатор (с тремя интерфейсами) используется для объединения семи хостов. Обратите внимание на IP-адреса, назначенные интерфейсам хостов и маршрутизатора. IP-адреса трех хостов в левой части рисунка, а также IP-адрес интерфейса маршрутизатора, к которому они присоединены, имеют вид 223.1.1.xxx. То есть левые 24 бита их IP-адресов одинаковые. Кроме того, они соединены друг с другом единой физической линией (в данном случае широковещательной линией связи, например Ethernet-кабелем, к которому они все физически присоединены), без промежуточных маршрутизаторов. В терминологии IP интерфейсы этих трех хостов и левого верхнего интерфейса маршрутизатора образуют *IP-сеть,* или просто *сеть.* Общие 24 бита адреса формируют сетевую часть их IP-адреса. Оставшиеся восемь разрядов представляют собой «хостовую» часть IP-адреса. (Мы будем называть хостовую часть IP-адреса «интерфейсной частью адреса», так как IP-адрес соответствует не хосту, а интерфейсу; тем не менее терминология «хостовая часть адреса» часто используется на практике.) У самой сети также есть адрес: 223.1.1.0/24, где нотация /24, иногда называемая *сетевой маской,* означает, что самые левые 24 бита 32-разрядного числа определяют сетевой адрес. Эти самые левые разряды часто называют *сетевым префиксом.* Таким образом, сеть 223.1.1.0/24 состоит из трех хостовых интерфейсов (223.1.1.1, 223.1.1.2 и 223.1.1.3) и одного интерфейса маршрутизатора (223.1.1.4). Любые дополнительные хосты, присоединенные к сети 223.1.1.0/24, должны иметь адреса вида 223.1.l.xxx. На рис. 4.14 изображены еще две сети: 223.1.2.0/24 и 223.1.3.0/24. Все три сети показаны на рис. 4.15.

Принятое в протоколе IP определение «сети» не ограничивается Ethernet-сегментами, соединяющими несколько хостов с интерфейсом маршрутизатора. Рассмотрим рис. 4.16, на котором показаны три маршрутизатора, соединенные друг с другом линиями «точка-точка». У каждого маршрутизатора есть три интерфейса, два из которых соединяют маршрутизаторы друг с другом по линиям «точка-точка» и

один выделен для широковещательной линии, напрямую соединяющей маршрутизатор с парой хостов. Здесь мы видим также три сети, 223.1.1.0/24, 223.1.2.0/24 и 223.1.3.0/24, аналогичные сетям на рис. 4.14. Однако обратите внимание, что в этом примере есть еще три сети: 223.1.9.0/24, для интерфейсов, соединяющих маршрутизаторы R1 и R2; 223.1.8.0/24, для интерфейсов, соединяющих маршрутизаторы R2 и R3; и 223.1.7.0/24, для интерфейсов, соединяющих маршрутизаторы R1 и R3.

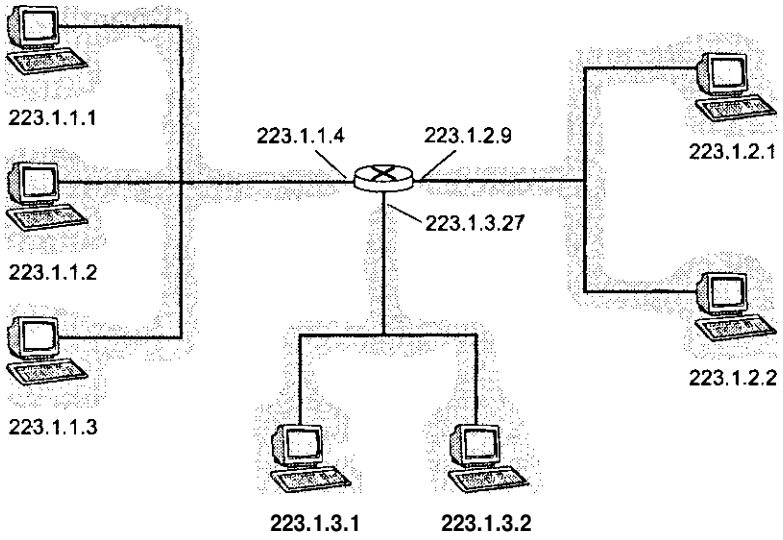


Рис. 4.14. Адреса интерфейсов

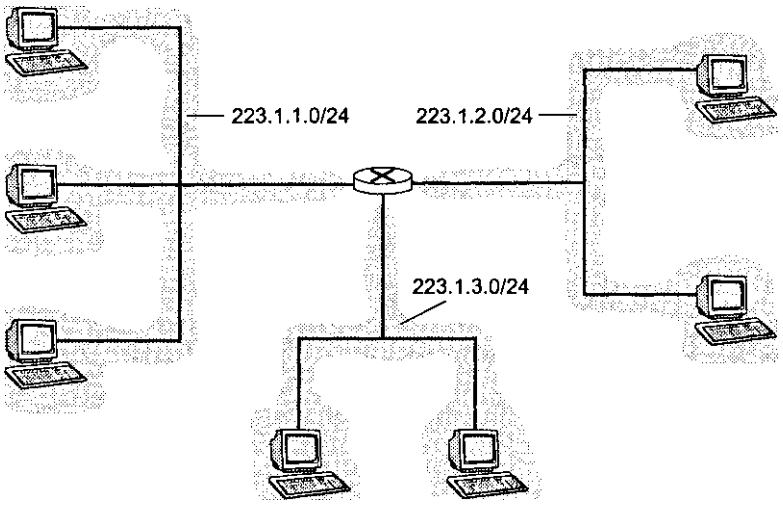


Рис. 4.15. Сетевые адреса

Для общего случая сложной системы из маршрутизаторов и хостов можно распознавать IP-сети следующим способом. Отсоединим каждый интерфейс от его хоста или маршрутизатора. Таким образом, мы получим островки изолированных сетей, границы которых состоят из интерфейсов. Будем называть каждую такую изолированную структуру *сетью.* Применив подобную процедуру к системе, показанной на рис. 4.16, мы получим шесть островков, или сетей. Сегодняшний Интернет состоит из миллионов подобных сетей. Понятия сети и сетевого адреса очень важны. Они играют центральную роль в архитектуре маршрутизации Интернета.

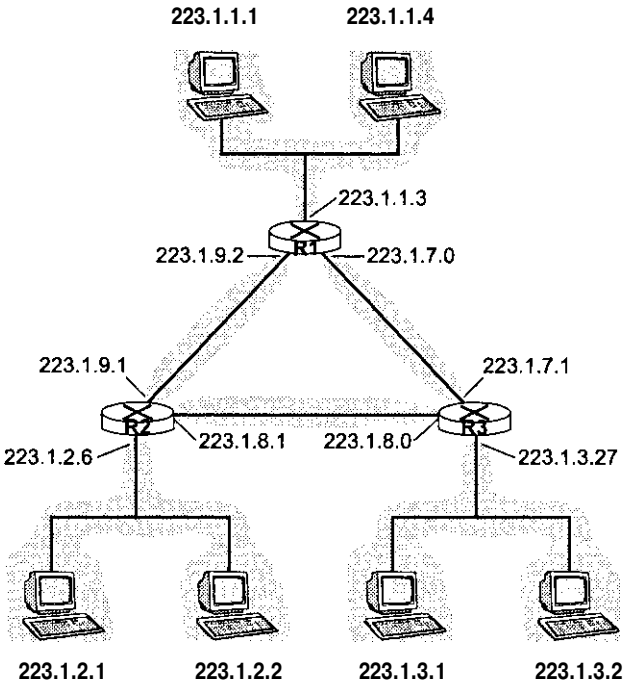


Рис. 4.16. Три маршрутизатора, соединяющие шесть хостов

Теперь, когда мы определили понятие сети, можно переходить к более подробному обсуждению IP-адресации. В оригинальной архитектуре адресации Интернета определены четыре класса адресов, как показано на рис. 4.17. Пятый класс адресов, начинающихся с цифр 11110, зарезервирован на будущее. В адресах класса A первые 8 бит идентифицируют сеть, а последние 24 бита обозначают интерфейс в этой сети. Таким образом, в классе A может существовать до 2^7 сетей (первый из восьми битов имеет фиксированное нулевое значение), в каждой из которых может быть до 2^{24} интерфейсов. Адресное пространство класса B позволяет создать до 2^{14} сетей, в каждой из которых может быть до 2^{16} интерфейсов. В адресе класса C первые 24 бита используются для идентификации сети, и только 8 бит остаются для идентификатора интерфейса. Адреса класса D зарезервированы для так называемых групповых адресов. Мы обсудим адреса класса D в разделе «Групповая маршрутизация».

Рис. 4.17. Форматы адресов протокола IPv4

Показанные на рисунке четыре класса адресов более не являются частью архитектуры IP-адресации. Требование, чтобы сетевая часть IP-адреса занимала ровно один, два или три байта, оказалось серьезным препятствием на пути быстро растущего числа организаций с сетями небольшого и среднего размера. Сеть класса C (/24) может содержать не более 2^8 - 2 = 254 хостов (два из 2^8 = 256 адресов зарезервированы для специального использования), чего слишком мало для большинства организаций. Однако сеть класса B (1/16), поддерживающая до 65 534 хостов, слишком велика для небольших компаний. В классической схеме адресации организация со, скажем, 2000 хостами, как правило, приобретала сетевой адрес класса B (/16). В результате адресное пространство класса B использовалось неэффективно, а количество свободных адресов сетей быстро сокращалось. Так, из 65 534 адресов организация с 2000 хостами задействовала лишь 2000, в результате более 63 000 адресов оставались бесполезными, так как не могли использоваться другими организациями.

В 1993 году группа IETF приняла стандарт *CIDR* (Classless Inter-Domain Routing — бесклассовая внутридоменная маршрутизация). Данный стандарт (RFC 1519) позволяет использовать сетевую часть адреса любой длины, не обязательно кратной 8 бит. Сетевой адрес стандарта CIDR записывается четырьмя десятичными числами, разделенными точками, например *a.b.c.d/x,* где *x* указывает количество разрядов в сетевой части адреса. В нашем примере организации с 2000 хостами может быть выделен блок из 2048 адресов вида *a.b.c.d/21,* а около 63 000 адресов в этом случае могут быть предоставлены другим организациям. В данном случае первые 21 бит указывают адрес сети организации, и у всех хостов данной организации эти разряды IP-адресов должны быть одинаковыми. Остальные 11 разрядов идентифицируют хост в организации. На практике организация может также разбить и эти 11 младших разрядов адреса, выделяя тем самым в своей сети *подсети* (RFC 950). Теперь, когда мы подробно изучили IP-адресацию, самое время задать себе вопрос, каким образом хост или сеть получают свои IP-адреса. Сначала обсудим, как сеть получает блок адресов для своих устройств, а затем поговорим о том, как адрес назначается устройству (например, хосту).

Получение сетевого адреса

Чтобы получить блок IP-адресов для корпоративной сети, администратор сети может сначала связаться со своим Интернет-провайдером, который может выделить адреса из большего блока ранее выделенных ему адресов. Например, Интернет-провайдеру может быть выделен блок адресов 200.23.16.0/20. Интернет-провайдер, в свою очередь, может разделить этот блок адресов на восемь равных по размеру меньших блоков и распределить эти блоки между восемью организациями, как показано ниже (для наглядности мы подчеркнули сетевую часть этих адресов).

Блок адресов провайдера	**200.23.16.0/20**	**11001000 00010111 0001**0000 00000000
Организация 1	**200.23.16.0/23**	**11001000 00010111 00010000** 00000000
Организация 2	**200.23.18.0/23**	**11001000 00010111 00010010** 00000000
Организация 3	**200.23.20.0/23**	**11001000 00010111 00010100** 00000000
Организация 7	**200.23.30.0/23**	**11001000 00010111 00011110** 00000000

ПРИНЦИПЫ И ПРАКТИКА

Данный пример Интернет-провайдера, предоставляющего доступ к Интернету для восьми организаций, также служит прекрасной иллюстрацией того, как стандарт CIDR помогает тщательно подбирать блоки сетевых адресов и упрощает иерархическую маршрутизацию. Пусть, как показано на рис. 4.18, данный Интернет-провайдер (назовем его «Ночные полеты») объявляет всем о том, что все дейтаграммы, первые 20 бит адреса которых совпадают с 200.23.16.0/20, следует посылать ему. Остальному миру ни к чему знать, что в блоке адресов 200.23.16.0/20 на самом деле скрываются восемь организаций, у каждой из которых есть собственная сеть. Эту способность задействовать один сетевой префикс для нескольких сетей часто называют *маршрутной агрегацией.*

Маршрутная агрегация особенно хороша тогда, когда адреса поблочно выделяются Интернет-провайдерам, которые, в свою очередь, выделяют их организациям-клиентам. Но что произойдет, если адреса не будут выделяться таким иерархическим образом? Например, если организацию 1 перестанет удовлетворять качество услуг, предоставляемых Интернет-провайдером «Ночные полеты», и она решит перейти к новому Интернет-провайдеру с названием, например, «Голубая луна»? Как показано на рис. 4.18, Интернет-провайдеру «Голубая луна» принадлежит блок адресов 199.31.0.0/16, но IP-адреса организации 1 оказываются за пределами этого блока. Что делать в этой ситуации? Разумеется, организация 1 может запомнить адреса всех своих хостов и маршрутизаторов в пределах адресного блока Интернет-провайдера «Голубая луна». Но такое решение потребует больших затрат, кроме того, организация 1 может в будущем еще раз сменить Интернет-провайдера. Как правило, принимается решение сохранить за организацией 1 блок адресов 200.23.18.0/23. В этом случае, как показано на рис. 4.19, Интернет-провайдер «Ночные полеты» продолжит поддерживать блок адресов 200.23.16.0/20, а Интернет-провайдер «Голубая луна» — блок адресов 199.31.0.0/16. Однако теперь Интернет-провайдер «Голубая луна» также будет поддерживать блок адресов 200.23.18.0/23 для организации 1. Когда другие маршрутизаторы в Интернете увидят блоки адресов 200.23.16.0/20 (от Интернет-провайдера «Ночные полеты») и 200.23.18.0/23 (от Интернет-провайдера «Голубая луна») и захотят направлять пакеты по адресам в блоке 200.23.18.0/23, они воспользуются правилом *соответствия самому длинному префиксу* **и направят пакеты Интернет-провайдеру «Голубая луна», так как он заявил о самом длинном (наиболее конкретном) адресном префиксе, совпадающем с адресом получателя.**

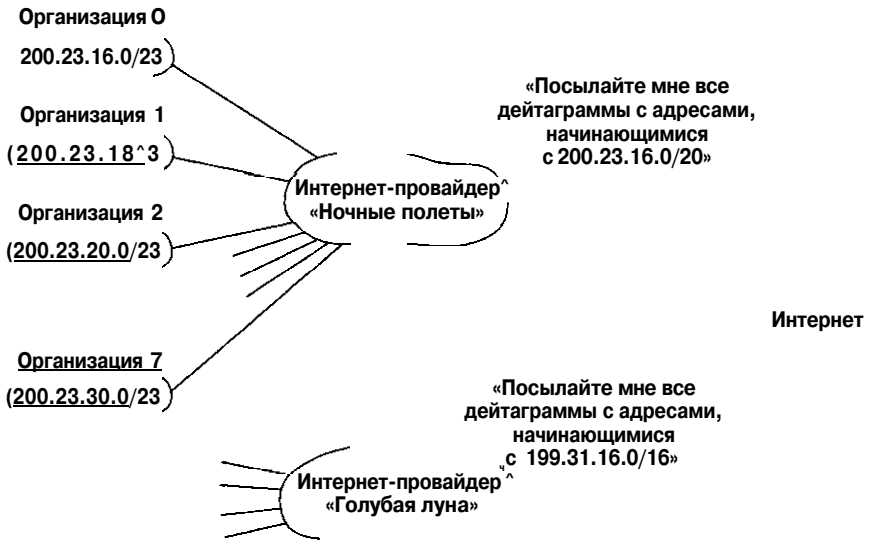


Рис. 4.18. Иерархическая адресация и маршрутная агрегация

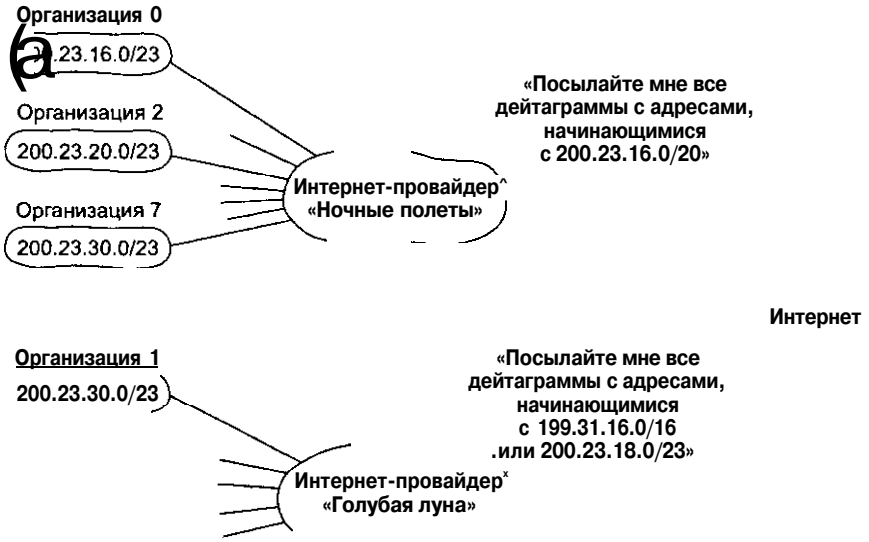


Рис. 4.19. У Интернет-провайдера «Голубая луна» есть наиболее точный маршрут к организации 1

Интернет-провайдер — не единственный источник блоков IP-адресов. Очевидно, сами Интернет-провайдеры тоже должны где-то получать IP-адреса. Существует ли организация, в глобальном масштабе отвечающая за управление пространством IP-адре-

сов и выделение блоков адресов Интернет-провайдерам и другим организациям? Разумеется, такая организация есть [508]! ICANN (Internet Corporation for Assigned Names and Numbers — организация по назначению адресов и имен в Интернете) управляет IP-адресами на основе набора правил, опубликованных в RFC 2050. Роль некоммерческой организации ICANN заключается не только в предоставлении IP-адресов, но также в управлении корневыми DNS-серверами [348]. Она также занимается весьма спорной работой по выделению имен доменам и разрешению связанных с этим конфликтов. ICANN предоставляет адреса региональным Интернет-регистратурам, таким как ARIN, RIPE и APNIC, управляющим предоставлением адресов в своих регионах.

Получение адреса хоста

Как только организация получает блок адресов от своего Интернет-провайдера, она может назначать индивидуальные IP-адреса интерфейсам своих хостов и маршрутизаторов. Для интерфейсов маршрутизаторов системный администратор вручную задает IP-адреса (часто удаленным образом с помощью сетевой управляющей программы). Хосту IP-адрес может быть выделен двумя способами.

- *Ручное конфигурирование.* Системный администратор вручную указывает IP-адрес хоста (как правило, в файле).
- *Использование протокола DHCP* (RFC 2131). Протокол DHCP (Dynamic Host Configuration Protocol — протокол динамической конфигурации хоста) позволяет хосту автоматически получить IP-адрес, а также дополнительную информацию, такую как адрес ближайшего маршрутизатора и адрес DNS-сервера.

Благодаря своей способности автоматизировать настройку сетевых параметров соединения хоста и сети протокол DHCP часто называют *самонастраивающимся* (plug-and-play) *протоколом*. Эта способность делает его очень привлекательным для сетевых администраторов, которым в противном случае пришлось бы выполнять всю эту работу вручную! Протокол DHCP также широко применяется в региональных сетях доступа к Интернету и беспроводных локальных сетях, в которых хосты редко подключаются к сети и редко от нее отключаются.

Сетевой администратор может сконфигурировать протокол DHCP таким образом, чтобы у определенных хостов IP-адреса были постоянными, то есть при каждом подключении к сети хосту будет выделяться один и тот же адрес. Однако у многих организаций и локальных Интернет-провайдеров недостаточно IP-адресов для всех их хостов. В этом случае протокол DHCP позволяет назначать каждому соединившемуся хосту *временный IP-адрес*. В качестве примера рассмотрим регионального Интернет-провайдера, у которого 2000 клиентов, но одновременно к Интернету подключается не более 400. Для поддержки всех 2000 клиентов Интернет-провайдеру не нужен блок из 2000 адресов. Используя DHCP-сервер, динамически выделяющий адреса, Интернет-провайдер может обойтись блоком из 512 адресов (например, блоком 200.23.30.0/23). Когда хосты подключаются к сети или отключаются от сети, DHCP-сервер обновляет свой список доступных IP-адресов. Каждый раз, когда хост подключается к Интернету, DHCP-сервер выделяет ему произвольный адрес из текущего пула доступных адресов. Каждый раз, когда хост отсоединяется от Интернета, его адрес возвращается в пул.

Еще одна важная причина, по которой протокол DHCP получил широкое распространение, связана с мобильными компьютерами. Рассмотрим, например, студента, переносящего свой лэптоп из комнаты в общежитии в библиотеку или в учебный класс. Скорее всего, в каждом новом помещении студент будет подключаться к новой (в смысле адресации; см. рис. 4.15) сети, и, таким образом, ему каждый раз будет требоваться новый IP-адрес. Протокол DHCP идеально подходит для этой ситуации, так как многие пользователи подключаются к Интернету и отключаются от Интернета в разных местах, и адреса нужны им только на ограниченное время. Мы рассмотрим протокол DHCP подробнее в конце этого раздела.

Адресация, маршрутизация и продвижение дейтаграмм

Теперь, определив понятия интерфейсов и сетей, а также получив базовое представление об IP-адресации, поговорим о том, как хосты и маршрутизаторы переносят IP-дейтаграмму от отправителя к получателю. Высокоуровневый взгляд на IP-дейтаграмму иллюстрирует рис. 4.20. Каждая IP-дейтаграмма содержит поля адресов отправителя и получателя. Хост-отправитель заполняет поле адреса отправителя собственным 32-разрядным IP-адресом. Поле адреса получателя он заполняет 32-разрядным IP-адресом хоста-получателя, которому предназначается дейтаграмма. Поле данных дейтаграммы, как правило, заполняется TCP- или UDP-сегментом. Остальные поля IP-дейтаграммы мы обсудим чуть позже.

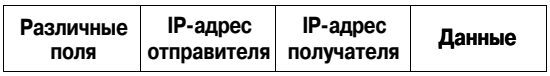


Рис. 4.20. Ключевые поля IP-дейтаграммы

Как после создания IP-дейтаграммы хостом-отправителем сетевой уровень перемещает ее хосту-получателю? Ответ на этот вопрос зависит от того, располагаются оба хоста в одной и той же сети или в разных сетях (здесь под сетью понимается IP-сеть, как было описано ранее). Рассмотрим этот вопрос на примере сети на рис. 4.21. Пусть сначала хост *A* хочет послать IP-дейтаграмму хосту J5, находящемуся в той же сети 223.1.1.0/24, что и хост Л. Это делается следующим образом. Сначала протокол IP хоста *A* заглядывает в свою внутреннюю IP-таблицу продвижения данных, также показанную на рисунке, и находит в ней сетевой адрес 223.1.1.0/24, совпадающий со старшими разрядами IP-адреса хоста *B*. В таблице продвижения данных указывается, что количество ретрансляционных участков до сети 233.1.1.0 равно 1; это означает, что хост *B* расположен в той же самой сети, что и хост *A*. Таким образом, хост *A* узнает, что дейтаграмму хосту *B* можно послать прямо по выходному интерфейсу хоста *A* безо всяких промежуточных маршрутизаторов. Тогда хост *A* передает IP-дейтаграмму протоколу канального уровня, который передает ее хосту *B*. (Как канальный уровень пересылает дейтаграмму от одного интерфейса к другому, мы обсудим в главе 5.)

Рассмотрим теперь более интересный случай, когда хосту *A* нужно передать дейтаграмму другому хосту, например хосту *E,* находящемуся в другой сети. Хост *A*

также обращается к своей таблице продвижения данных и находит в ней адрес сети 223.1.2.0/24, совпадающий со старшими разрядами IP-адреса хоста *E*. Поскольку количество ретрансляционных участков до сети 233.1.2.0/24 равно 2, хост Л понимает, что получатель располагается в другой сети, и поэтому дейтаграмму следует посылать промежуточному маршрутизатору. Из той же таблицы хост Л узнает, что предназначаемые хосту £ дейтаграммы следует направлять интерфейсу маршрутизатора с адресом 223.1.1.4, с которым интерфейс хоста Л соединен напрямую. Затем протокол IP хоста Л передает дейтаграмму канальному уровню, сообщая ему, что дейтаграмму следует отправить по IP-адресу 223.1.1.4. Здесь важно отметить, что, хотя дейтаграмма посылается (с помощью канального уровня) интерфейсу маршрутизатора, поле адреса получателя дейтаграммы, по-прежнему содержит адрес конечного получателя (хоста £), а не адрес интерфейса промежуточного маршрутизатора.

Таблица продвижения данных хоста А

Сеть получателя	Следующий маршрутизатор	Число ретрансляционных участков
223.1.1.0/24		1
223.1.2.0/24	223.1.1.4	2
223.1.3.0/24	223.1.1.4	2

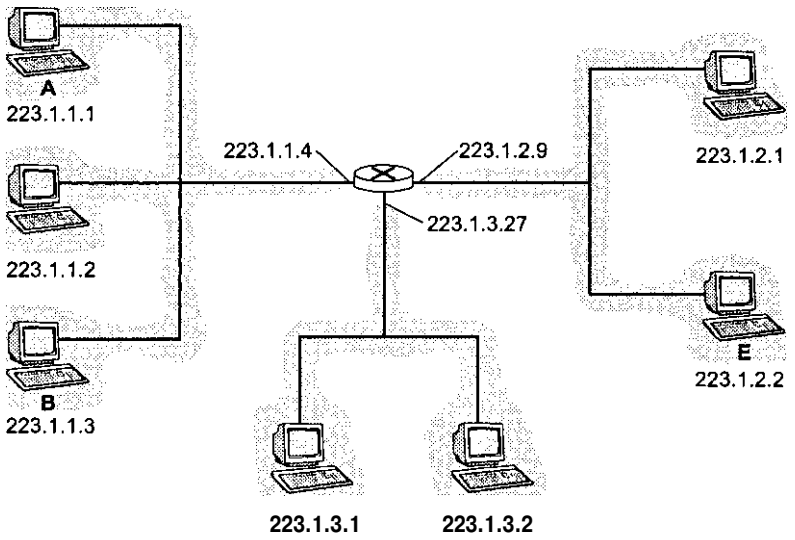


Рис. 4.21. Продвижение данных хоста А

Итак, дейтаграмма попадает на маршрутизатор, и теперь маршрутизатор должен переправить дейтаграмму ее конечному получателю. Как показано на рис. 4.22, маршрутизатор сверяется с собственной таблицей продвижения данных и находит в ней

адрес сети 223.1.2.0/24, совпадающий с начальными битами IP-адреса хоста *E*. В этой таблице также указывается, что дейтаграмму следует направлять интерфейсу маршрутизатора 223.1.2.9. Поскольку количество ретрансляционных участков до получателя равно 1, маршрутизатор понимает, что получающий хост £ располагается в той же самой сети, что и его собственный интерфейс 223.1.2.9. Таким образом, маршрутизатор передает дейтаграмму этому интерфейсу, который пересылает ее хосту *E*.

Таблица продвижения данных хоста A

Сеть получателя	Следующий маршрутизатор	Число ретрансляционных участков	Интерфейс
223.1.1.0/24	—	1	223.1.1.4
223.1.2.0/24	—	1	223.1.2.9
223.1.3.0/24	—	1	223.1.3.27

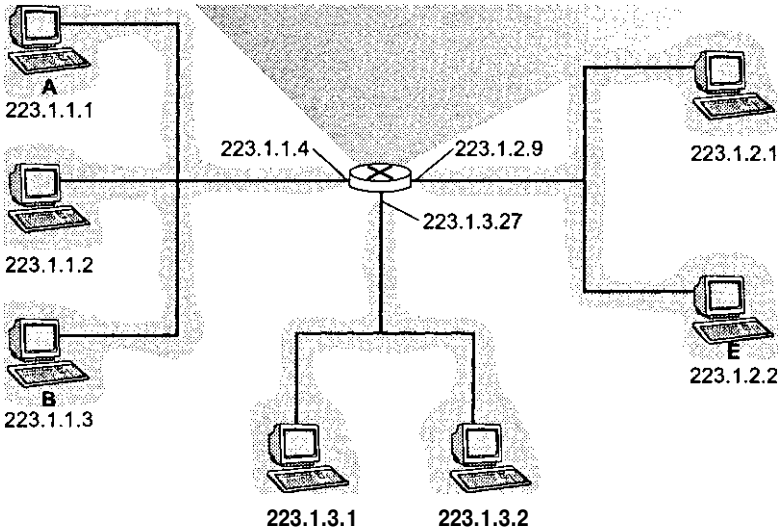


Рис. 4.22. Продвижение данных маршрутизатора

Обратите внимание, что все записи в столбце, предназначенном для указания адресов следующих маршрутизаторов, пусты, так как каждая сеть (223.1.1.0/24, 223.1.2.0/24 и 223.1.3.0/24) напрямую соединена с маршрутизатором. В этом случае нет необходимости направлять дейтаграммы промежуточным маршрутизаторам. Однако если бы хосты *AuE* разделяли два маршрутизатора, тогда в таблице продвижения данных первого маршрутизатора на пути от хоста *A* до хоста *E* в соответствующей строке указывалось бы два ретрансляционных участка до адресата, а также IP-адрес второго маршрутизатора вдоль маршрута. Тогда первый маршрутизатор переправлял бы дейтаграмму второму маршрутизатору при помощи протокола канального уровня, соединяющего два маршрутизатора. Второй марш-

рутизатор направлял бы дейтаграмму хосту-получателю с помощью протокола канального уровня, соединяющего второй маршрутизатор и хост-получатель.

Возможно, вы помните из главы 1, что маршрутизация дейтаграммы в Интернете напоминает водителя автомобиля, спрашивающего дорогу на каждом перекрестке. Теперь должно быть ясно, почему эта аналогия соответствует маршрутизации в Интернете. Путешествуя от отправителя до получателя, дейтаграмма проходит через целую последовательность маршрутизаторов. На каждом маршрутизаторе в этой последовательности она останавливается и «спрашивает» у маршрутизатора дорогу до конечного получателя. Если маршрутизатор не находится в той же сети, что и конечный получатель, таблица продвижения данных как бы говорит дейтаграмме: «Я не знаю точно, как добраться до конечного получателя, но я знаю, что до него можно добраться по этой линии (аналог дороги), соединенной с одним из моих интерфейсов». Затем дейтаграмма отправляется по указанной линии, прибывает на следующий маршрутизатор и снова спрашивает дорогу.

Таким образом, мы видим, что таблицы продвижения данных на маршрутизаторах играют центральную роль в маршрутизации дейтаграмм через Интернет. Но как эти таблицы конфигурируются и управляются в большой сети (такой как Интернет) с большим количеством маршрутов между отправителями и получателями? Очевидно, таблицы продвижения данных должны конфигурироваться так, чтобы дейтаграммы следовали по оптимальным маршрутам от отправителя к получателю. Как вы, возможно, уже догадались, для конфигурирования и поддержки таблиц продвижения данных служат алгоритмы маршрутизации, которые мы рассматривали в разделе «Основы маршрутизации». Алгоритмы маршрутизации, применяемые в Интернете, мы обсудим в разделе «Маршрутизация в Интернете». Но прежде чем перейти к обсуждению вопроса реализации алгоритмов маршрутизации в Интернете, поговорим о пяти наиболее важных составляющих успешного функционирования протокола IP. Это, во-первых, формат дейтаграмм; во-вторых, фрагментация IP-дейтаграмм; в-третьих и в-четвертых, уже упоминавшиеся протоколы ICMP и DHCP; и наконец в-пятых, трансляция сетевых адресов (NAT). Рассмотрим эти составляющие поочередно.

Формат дейтаграммы

Формат дейтаграммы протокола IPv4 показан на рис. 4.23. Ниже перечислены ключевые поля IPv4-дейтаграммы.

- *Версия.* Четыре бита в этом поле определяют номер версии протокола IP. По этому номеру маршрутизатор может определить, как интерпретировать остальные поля IP-дейтаграммы. В различных версиях протокола IP применяются различные форматы IP-дейтаграмм. На рисунке показан формат дейтаграммы текущей версии протокола IP (IPv4). Формат дейтаграммы новой версии протокола IP (IPv6) обсуждается в разделе «Протокол IPv6».
- *Длина заголовка.* Поскольку IPv4-дейтаграмма может содержать разное количество необязательных полей параметров (включаемых в заголовок IPv4-дейтаграммы), эти четыре бита необходимы для того, чтобы определить, где заканчивается заголовок и начинаются данные. В большинстве IP-дейтаграмм не содержатся поля параметров, поэтому обычно заголовок IP-дейтаграммы 20-разрядный.

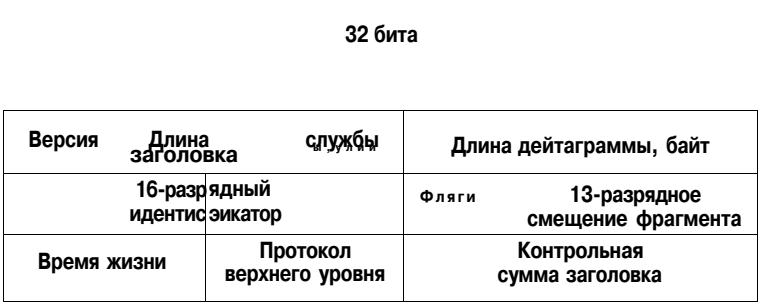


Рис. 4.23. Формат ^4-дейтаграммы

- *Тип службы.* Поле типа службы (Type Of Service, TOS) было включено в заголовок IPv4-дейтаграммы, чтобы была возможность разделять IP-дейтаграммы на типы (например, выделять дейтаграммы, для которых требуется низкая задержка, или высокая пропускная способность, или высокая надежность). Так, может оказаться полезным отличать дейтаграммы реального времени (например, используемые в IP-телефонии) от прочего трафика (например, FTP). В последние годы один из основных производителей маршрутизаторов (Cisco) стал интерпретировать первые три бита поля TOS как идентификатор уровня услуг, предоставляемых маршрутизатором. Предоставляемый уровень услуг определяется администратором маршрутизатора. Дифференцированное обслуживание будет более подробно обсуждаться в главе 6.
- *Длина дейтаграммы.* Это полная длина IP-дейтаграммы (заголовок плюс данные) в байтах. Поскольку размер этого поля равен 16 бит, теоретически максимальный размер IP-дейтаграммы может составлять 65 535 байт. Однако размер дейтаграмм редко превосходит 1500 байт и обычно ограничивается значением 576 байт.
- *Идентификатор, флаги, смещение фрагмента.* Эти три поля имеют отношение к так называемой IP-фрагментации. Этот вопрос мы подробно рассмотрим чуть позже. Интересно отметить, что новая версия протокола IP (IPv6) запрещает фрагментацию в маршрутизаторах.
- *Время жизни.* Поле времени жизни (Time To Live, TTL) позволяет гарантировать, что дейтаграммы не будут вечно циркулировать в сети (например, из-за существующей в течение долгого времени маршрутной петли). Значение этого поля уменьшается на единицу на каждом маршрутизаторе. Когда значение поля TTL достигает нуля, маршрутизатор отбрасывает дейтаграмму.
- *Протокол.* Это поле используется только тогда, когда IP-дейтаграмма достигает конечного адресата. Значение поля определяет протокол транспортного уровня,

которому следует передать данные из IP-дейтаграммы. Например, значение 6 означает, что порция данных должна быть передана протоколу TCP, а значение 17 — протоколу UDP. Список всех возможных номеров имеется в RFC 1700, RFC 3232. Обратите внимание, что роль номера протокола в IP-дейтаграмме полностью аналогична роли номера порта в сегменте транспортного уровня. Номер протокола представляет собой «клей», связывающий вместе сетевой и транспортный уровни, тогда как номер порта связывает транспортный уровень с прикладным. В главе 5 мы увидим, что в кадре канального уровня также есть специальное поле, связывающее канальный уровень с сетевым уровнем.

- *Контрольная сумма заголовка.* Контрольная сумма заголовка помогает маршрутизатору обнаруживать ошибки в полученных IP-дейтаграммах. Контрольная сумма заголовка вычисляется путем суммирования всех двухбайтовых слов заголовка в дополнительном коде. Как уже упоминалось в разделе «Протокол UDP — передача без установления соединения» главы 3, дополнение до 1 этой суммы хранится в поле контрольной суммы. Маршрутизатор вычисляет контрольную сумму заголовка для каждой полученной дейтаграммы и таким образом проверяет ошибки в заголовке. Как правило, маршрутизаторы отбрасывают дейтаграммы, в которых обнаруживают ошибки. Обратите внимание, что контрольную сумму нужно вычислять заново и снова сохранять в поле заголовка на каждом маршрутизаторе, так как на единицу уменьшается поле времени жизни, могут также измениться поля параметров. Интересное описание быстрых алгоритмов для вычисления контрольной суммы заголовка IP-дейтаграммы содержится в RFC 1071. В этом месте читатели часто задают вопрос, зачем TCP/IP выполняет проверку контрольной суммы на обоих уровнях (на сетевом и на транспортном)? Тому есть несколько причин. Во-первых, на IP-уровне вычисляется только контрольная сумма IP-заголовка, тогда как на транспортном уровне вычисляется сумма всего TCP- или UDP-сегмента. Во-вторых, протоколы TCP/UDP и IP, вообще говоря, не обязаны принадлежать одному и тому же стеку протоколов. В принципе, протокол TCP может работать поверх другого протокола (например, ATM), а протокол IP может переносить данные, которые не будут передаваться протоколу TCP или UDP.
- *IP-адреса отправителя и получателя.* Эти поля содержат 32-разрядные IP-адреса отправителя и конечного получателя IP-дейтаграммы. Мы подробно обсуждали IP-адресацию в подразделе «Адресация в протоколе IPv4» данного раздела. Однако нам следует упомянуть еще один тип IP-адреса, а именно широковещательный IP-адрес 255.255.255.255. Когда хост передает дейтаграмму с адресом получателя 255.255.255.255, сообщение доставляется всем хостам той же сети. Кроме того, маршрутизаторы могут переправить широковещательное сообщение в соседние IP-сети (хотя, как правило, они этого не делают). При описании протокола DHCP будет приведен пример использования широковещательного IP-пакета.
- *Параметры.* Поле параметров позволяет расширить IP-заголовок. Параметры заголовка представляют собой редко используемые необязательные поля IP-дейтаграммы. Поэтому было решено не включать их в заголовок каждой дейтаграммы и таким образом снизить накладные расходы. Однако само существование необязательных полей заголовка усложняет его обработку, так как заголовки

дейтаграмм могут иметь различную длину, и, чтобы определить, где заканчивается заголовок и начинается поле данных, необходимо дополнительное поле длины заголовка. Кроме того, поскольку для одних дейтаграмм требуется обработка параметров, а для других нет, время обработки IP-дейтаграммы на маршрутизаторе может варьироваться в широких пределах. Эти соображения имеют особую важность для высокопроизводительных маршрутизаторов и хостов. Среди причин, по которым в протоколе IPv6 отказались от необязательных полей заголовка, была и эта.

- *Данные {полезная нагрузка).* Наконец, мы добрались до последнего, самого важного поля, ради которого и существует дейтаграмма! В большинстве случаев поле данных IP-дейтаграммы содержит сегмент транспортного уровня (TCP или UDP), который необходимо доставить адресату. Однако поле данных может содержать и другие типы данных, например сообщения протокола ICMP (который будет обсуждаться в подразделе «Протокол 1CMP» данного раздела).

Обратите внимание, что IP-дейтаграмма содержит 20-разрядный заголовок (без дополнительных полей). Если дейтаграмма содержит TCP-сегмент, тогда в каждой (не фрагментированной) дейтаграмме помимо сообщения прикладного уровня содержится 40 байт заголовков (20 байт IP-заголовка и 20 байт TCP-заголовка).

Фрагментация IP-дейтаграмм

В главе 5 мы увидим, что у различных протоколов канального уровня может быть разный, максимальный размер переносимых пакетов. Некоторые протоколы могут переносить «большие» пакеты, тогда как другие допускают перенос только «маленьких» пакетов. Например, Ethernet-пакеты могут содержать не более 1500 байт данных, тогда как многие протоколы глобальных линий способны переносить пакеты размером не более 576 байт. Максимальное количество данных, которое может переносить пакет канального уровня, называют *максимальной единицей передачи* (Maximum Transfer Unit, MTU). Поскольку каждая IP-дейтаграмма для передачи от одного маршрутизатора к другому инкапсулируется в пакет канального уровня, максимальный размер поля данных протокола канального уровня накладывает жесткое ограничение на длину IP-дейтаграммы. Само по себе жесткое ограничение на размер IP-дейтаграммы не представляет проблемы. Проблема заключается в том, что в каждой линии связи на пути от отправителя до получателя могут использоваться разные протоколы канального уровня, и у каждого из этих протоколов может быть свой, отличный от других, максимальный размер поля данных.

Чтобы лучше разобраться в этой проблеме, представьте себе маршрутизатор, соединяющий несколько линий, в каждой из которых применяется свой, отличный от других протокол канального уровня со своим максимальным размером поля данных. Предположим, маршрутизатор получает IP-дейтаграмму по одной линии и заглядывает в свою таблицу продвижения данных, чтобы определить исходящую линию для этой дейтаграммы. Предположим также, что максимальный размер поля данных в исходящей линии меньше длины IP-дейтаграммы. Впору запаниковать, поскольку нужно сжимать слишком большой IP-пакет так, чтобы он поместился в поле полезной нагрузки пакета канального уровня. Решение этой проблемы состоит в разбиении со-

держащихся в IP-дейтаграмме данных на несколько IP-дейтаграмм меньшего размера. Каждую из этих IP-дейтаграмм меньшего размера называют *фрагментом*. Прежде чем фрагменты достигнут транспортного уровня адресата, из них необходимо снова собрать исходную дейтаграмму. Действительно, протоколы TCP и UDP ожидают получить от сетевого уровня полный, не фрагментированный пакет. Разработчики протокола IPv4 понимали, что повторная сборка (и, возможно, повторная фрагментация) дейтаграмм на маршрутизаторах значительно усложнит протокол и снизит производительность маршрутизаторов. (Если бы вы были маршрутизатором, захотели бы вы сверх всех ваших обязанностей заниматься еще и повторной сборкой фрагментированных дейтаграмм?) Придерживаясь принципа сохранения простоты сетевого уровня, разработчики IPv4 решили оставить задачу повторной сборки фрагментированных дейтаграмм оконечным системам.

Когда хост-адресат получает серию дейтаграмм, он должен определить, являются ли данные дейтаграммы фрагментами некой исходной дейтаграммы большего размера. Если он выясняет, что некие дейтаграммы представляют собой фрагменты, ему нужно также идентифицировать последний фрагмент дейтаграммы, чтобы можно было собрать эти фрагменты вместе в оригинальную дейтаграмму и выяснить, как это делается. Чтобы хост-получатель мог осуществлять повторную сборку дейтаграмм, разработчики IPv4 поместили в дейтаграмму поля *идентификации, флага* и *фрагментации*. Когда дейтаграмма создается, хост-отправитель маркирует ее номером-идентификатором, а также помещает в нее адреса отправителя и получателя. Хост-отправитель увеличивает на единицу идентификационный номер для каждой следующей посылаемой дейтаграммы. Когда маршрутизатору необходимо фрагментировать дейтаграмму, каждый получающийся фрагмент помечается адресом отправителя, адресом получателя и идентификационным номером оригинальной дейтаграммы. Когда хост-адресат получает серию дейтаграмм от одного и того же передающего хоста, он изучает идентификационные номера дейтаграмм, чтобы определить, являются ли данные дейтаграммы фрагментами дейтаграммы большего размера. Поскольку протокол IP предоставляет ненадежную службу, один или несколько фрагментов могут не достичь адресата. Чтобы хост-адресат мог быть абсолютно уверен в том, что получил последний фрагмент оригинальной дейтаграммы, бит флага в последнем фрагменте устанавливается в 0, тогда как во всех остальных фрагментах он устанавливается в 1. Кроме того, чтобы хост-адресат мог определить, не был ли потерян какой-либо из фрагментов (а также иметь возможность собрать фрагменты оригинальной дейтаграммы в правильном порядке), в каждом фрагменте имеется поле смещения.

На рис. 4.24 изображен пример. Дейтаграмма из 4000 байт (20 байт IP-заголовка и 3980 байт полезной нагрузки) прибывает на маршрутизатор и должна быть переправлена далее по линии с максимальным размером поля данных в 1500 байт. Это означает, что 3980 байт оригинальной дейтаграммы должны быть распределены между тремя отдельными фрагментами (каждый из которых также представляет собой IP-дейтаграмму). Предположим, что оригинальная дейтаграмма маркирована идентификационным номером 777. Характеристики трех фрагментов показаны в табл. 4.3.

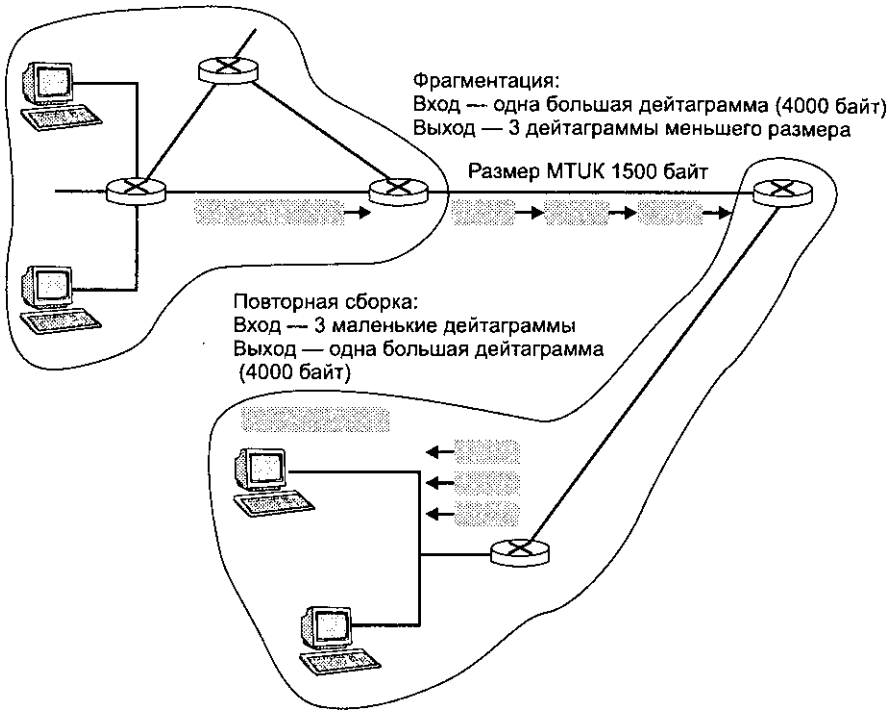


Рис. 4.24. Фрагментация и повторная сборка IP-дейтаграммы

Таблица 4.3. IP-фрагменты

Фрагмент	Байты	ID	Смещение	Флаг
1	1480	777	0	1
2	1480	777	1480	1
3	1020 = 3980- 1480- 1480	777	2960	0

Полезная нагрузка дейтаграммы передается транспортному уровню получателя только после того, как IP-уровень полностью восстановит оригинальную дейтаграмму. Если один или несколько фрагментов не сумеют достичь адресата, вся дейтаграмма отбрасывается и не передается транспортному уровню. Но, как было показано в предыдущей главе, если в качестве транспортного уровня используется протокол TCP, тогда восстановлением после потери фрагмента занимается протокол TCP, повторяя передачу всех данных оригинальной дейтаграммы.

Фрагментация и повторная сборка накладывают дополнительное бремя на Интернет-маршрутизаторы (фрагментация) и на хосты-адресаты (повторная сборка). Поэтому желательно свести фрагментацию к минимуму. Для этого часто ограничиваются размеры TCP- и UDP-сегментов, что снижает вероятность фрагментации. Поскольку все протоколы передачи данных, поддерживаемые протоколом IP, должны обеспечивать транспортировку пакетов данных размером, по меньшей мере, 576 байт, фрагментации можно полностью избежать, если использовать макси-

мальный размер сегмента (MSS), равный 536 байт, что вместе с двумя 20-разрядными IP- и TCP-заголовками составит 576 байт. По этой причине размер большинства TCP-сегментов для передачи данных больших объемов (например, HTTP-данных) находится в пределах от 512 до 536 байт (возможно, путешествуя в web, вы замечали, что данные поступают блоками примерно по 500 байт).

С web-сайта, посвященного этой книге (http://www.awl.com/kurose-ross), можно загрузить Java-апплет, автоматически генерирующий фрагменты. Вы задаете размер исходной дейтаграммы, размер максимальной единицы передачи (MTU) и идентификатор исходной дейтаграммы.

Протокол ICMP

Теперь мы рассмотрим протокол управляющих сообщений Интернета (Internet Control Message Protocol, ICMP), используемый хостами, маршрутизаторами и шлюзами для обмена информацией сетевого уровня друг с другом. Спецификация протокола ICMP содержится в RFC 792. Наиболее типичное использование протокола ICMP заключается в передаче сообщений об ошибках. Например, при работе с Telnet-, FTP- или HTTP-приложениями вы можете встретить сообщение об ошибке, например Destination network unreachable (сеть назначения недоступна). Это сообщение формирует протокол ICMP, если IP-маршрутизатор не может найти маршрут к хосту, указанному в вашем приложении. Маршрутизатор создает и отправляет вашему хосту ICMP-сообщение об ошибке. Ваш хост получает ICMP-сообщение и возвращает пытающейся связаться с удаленным хостом TCP-программе код ошибки. Протокол TCP, в свою очередь, возвращает код ошибки вашему приложению.

Протокол ICMP часто рассматривается как часть протокола IP, однако с точки зрения архитектуры он работает поверх протокола IP, так как ICMP-сообщения переносятся внутри IP-пакетов. То есть ICMP-сообщения переносятся как полезная нагрузка IP-дейтаграмм. Аналогично, когда хост получает IP-пакет, в котором протокол ICMP указан в качестве протокола более высокого уровня, хост передает пакет протоколу ICMP, так же как он передает пакет протоколам TCP и UDP.

У ICMP-сообщений есть поле типа и кодовое поле. Кроме того, ICMP-сообщения содержат первые 8 байт IP-дейтаграммы, вызвавшей генерацию ICMP-сообщения (так, чтобы отправитель мог определить, который пакет вызвал ошибку). Некоторые типы ICMP-сообщений показаны в табл. 4.4. Обратите внимание, что ICMP-сообщения используются не только для сигнализации об ошибках. Хорошо известная команда ping посылает указанному хосту ICMP-сообщение типа 8 с кодом 0. Хост-адресат, получив запрос эха, посылает обратно ICMP-сообщение типа 0 с кодом 0. Еще одним интересным ICMP-сообщением является сообщение гашения источника. Это сообщение редко используется на практике. Изначально оно предназначалось для борьбы с перегрузкой — испытывающий перегрузку маршрутизатор отправлял это ICMP-сообщение передающему хосту, чтобы заставить его снизить скорость передачи. Как было показано в главе 3, у протокола TCP есть собственный механизм борьбы с перегрузками, действующий на транспортном Уровне и не требующий обратной связи (как ICMP-сообщения гашения источника) от сетевого уровня.

Таблица 4.4. Типы ICMP-сообщений

Тип ICMP-сообщения	Код	Описание
0	0	Эхо-ответ (на команду ping)
3	0	Сеть-адресат недоступна
3	1	Хост-адресат недоступен
3	2	Протокол адресата недоступен
3	3	Порт-адресат недоступен
3	6	Сеть-адресат неизвестна
3	7	Хост-адресат неизвестен
4	0	Гашение источника (борьба с перегрузкой)
8	0	Запрос эха
8	0	Объявление маршрутизатора
10	0	Обнаружение маршрутизатора
11	0	Время жизни истекло
12	0	Неверный IP-заголовок

В главе 1 мы познакомились с программой (командой) Traceroute, позволяющей проследить маршрут от одного хоста до другого. Интересно отметить, что программа Traceroute реализована при помощи ICMP-сообщений. Чтобы определить имена и адреса маршрутизаторов между отправителем и получателем, программа Traceroute на хосте-отправителе посылает хосту-адресату серию обычных IP-дейтаграмм. У первой из этих дейтаграмм значение поля времени жизни равно 1, у второй оно равно 2, у третьей — 3, и т. д. Кроме того, для каждой из этих дейтаграмм отправитель запускает таймер. Когда *n-я* дейтаграмма прибывает на *n-и* маршрутизатор, *n-и* маршрутизатор видит, что время жизни этой дейтаграммы только что истекло. В соответствии с правилами протокола IP, маршрутизатор отбрасывает эту дейтаграмму и посылает источнику предупреждающее ICMP-сообщение (тип 11 код 0). Это сообщение содержит имя маршрутизатора и его IP-адрес. Когда это ICMP-сообщение прибывает к отправителю, тот по значению таймера узнает время оборота пакета, а также (из ICMP-сообщения) имя и IP-адрес *n-mo* маршрутизатора. Теперь, когда вы понимаете, как работает программа Traceroute, возможно, вам захочется с ней поэкспериментировать.

Протокол DHCP

Ранее в этом разделе говорилось, что протокол DHCP (Dynamic Host Configuration Protocol — протокол динамической конфигурации хоста) часто используется для назначения хостам IP-адресов в динамическом режиме. Мы кратко упомянули службы, предоставляемые хосту протоколом DHCP. В данном подразделе мы несколько подробнее поговорим о том, как протокол DHCP предоставляет свои службы.

Протокол DHCP представляет собой протокол для связи клиента с сервером. Клиентом, как правило, выступает только что подключившийся к сети новый хост,

желающий получить информацию о конфигурации сети и собственный IP-адрес. В простейшем случае в каждой сети (имеется в виду IP-сеть, см. рис. 4.15) есть свой DHCP-сервер. Если в сети нет сервера, необходим промежуточный DHCP-агент (как правило, маршрутизатор), которому известен адрес DHCP-сервера для этой сети. На рис. 4.25 показан DHCP-сервер, соединенный с сетью 233.1.2/24, и маршрутизатор, выступающий в роли промежуточного DHCP-агента для новых клиентов, присоединяющихся к сетям 223.1.1/24 и 223.1.3/24.

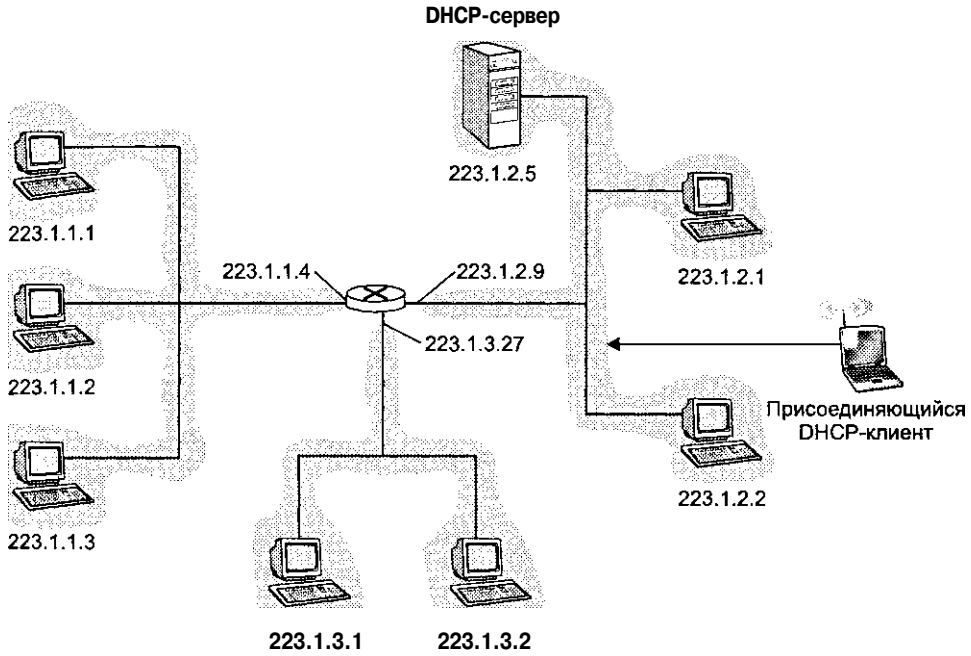


Рис. 4.25. Сценарий взаимодействия DHCP-клиента и DHCP-сервера

При появлении в сети нового клиента протокол DHCP запускает процесс из четырех этапов.

1. *Обнаружение DHCP-сервера.* Первым делом только что подключившийся хост должен найти DHCP-сервер, с которым можно взаимодействовать. Это делается при помощи сообщения о *поиске DHCP-сереера*[у] которое клиент посылает UDP-дейтаграмме через порт 67. Но кому он должен посылать эту дейтаграмму? Хосту не известен даже IP-адрес сети, к которой он подключился, а уж тем более адрес DHCP-сервера, обслуживающего эту сеть. Поэтому DHCP-клиент в поле адреса получателя дейтаграммы указывает широковещательный адрес 255.255.255.255, а себя (отправителя) обозначает как 0.0.0.0. Сообщение о поиске DHCP-сервера будет получено всеми машинами сети, включая все DHCP-серверы (и/или промежуточные агенты). В этом сообщении содержится идентификатор транзакции, позволяющий соотнести последующие ответы с запросом обнаружения DHCP-сервера.

2. *Предложение DHCP-сервера.* Получив сообщение о поиске DHCP-сервера DHCP-сервер отвечает клиенту сообщением с *DHCP-предложением.* Поскольку в сети могут находиться несколько DHCP-серверов, может случиться, что клиенту придется выбирать из нескольких предложений. Каждое DHCP-предложение содержит идентификатор транзакции полученного сообщения о поиске DHCP-сервера, предлагаемый IP-адрес клиента, маску сети и срок действия IP-адреса, называемый обычно *сроком аренды IP-адреса.* Как правило, сервер предоставляет новому хосту IP-адрес на срок от нескольких часов до нескольких дней [132]. Затем прибывшему клиенту посылается кадр канального уровня с IP-дейтаграммой, в которой содержится UDP-сегмент с DHCP-предложением (прекрасная иллюстрация многоуровневой иерархии протоколов). Как кадр канального уровня посылается клиенту, мы рассмотрим в главе 5, посвященной канальному уровню.
3. *DHCP-запрос.* Новый клиент выбирает одно из полученных им предложений от серверов и отвечает на выбранное им предложение DHCP-запросом, повторяя в нем конфигурационные параметры.
4. *DHCP-подтверждение.* Сервер отвечает на DHCP-запрос DHCP-подтверждением, подтверждая запрашиваемые параметры.

Как только клиент получает DHCP-подтверждение, диалог клиента и сервера завершается, и клиент может использовать выделенный ему протоколом DHCP IP-адрес в течение срока аренды. Поскольку клиенту адрес может требоваться дольше, протокол DHCP также предоставляет механизм для продления срока аренды.

Простой пример диалога клиента и сервера для конфигурации сети, изображенной на рис. 4.25, показан на рис. 4.26.

Значение протокола DHCP для поддержки функции самонастройки должно быть очевидно. Представим себе студента, переходящего со своим лэптопом из класса в библиотеку, а оттуда — в свою комнату в общежитии. В каждом помещении он подключается к новой сети и таким образом каждый раз получает новый IP-адрес. Трудно себе представить ситуацию, в которой при каждом подключении лэптопа настройка его IP-адреса и других параметров выполнялась бы вручную системным администратором. Кроме того, мало кто из студентов (кроме посещающих занятия по компьютерным сетям) обладает достаточной квалификацией для подобной настройки. Однако с точки зрения мобильности у протокола DHCP есть несколько недостатков. Поскольку каждый раз, когда узел соединяется с новой сетью, ему предоставляется протоколом DHCP новый сетевой адрес, невозможно поддерживать соединение с удаленным приложением в то время, когда мобильный узел перемещается между сетями. В разделе «Мобильность и сетевой уровень» мы обсудим мобильный протокол IP — недавно разработанное расширение протокола IP, позволяющее мобильным хостам при перемещении из одной сети в другую использовать один и тот же постоянный адрес.

Дополнительные сведения о протоколе DHCP можно найти в [132, 227]. Свободно распространяемые исходные тексты эталонной реализации протокола DHCP можно получить на web-сайте Internet Software Consortium [240].

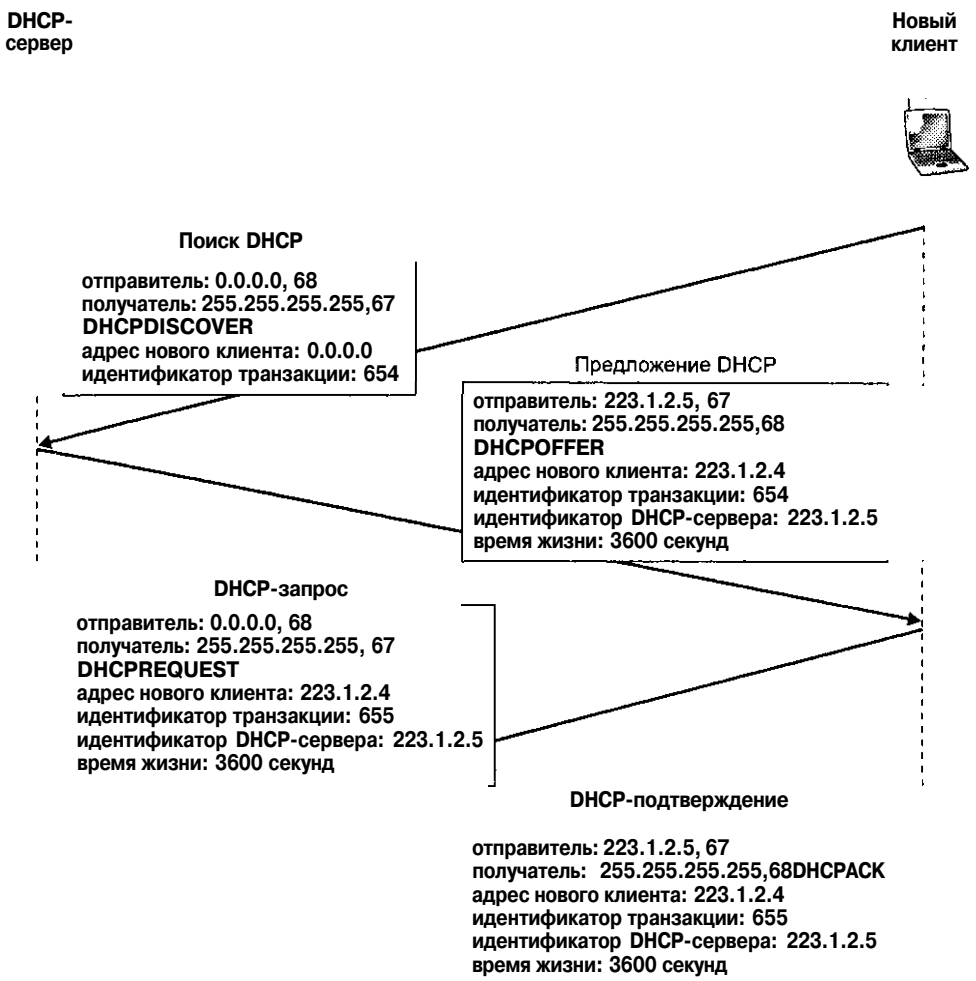


Рис. 4.26. Диалог DHCP-клиента и DHCP-сервера

Трансляторы сетевых адресов

Итак, теперь мы знаем, что у каждого устройства, поддерживающего протокол IP, должен быть IP-адрес. С распространением небольших и домашних офисов это бы означало, что каждый раз, когда в таком офисе возникает необходимость установить локальную сеть или соединить друг с другом несколько машин, поставщику услуг Интернета пришлось бы выделять блок адресов для всех машин маленького

офиса. В случае роста сети (например, у детей в доме помимо настольных персональных компьютеров появились карманные компьютеры, телефоны с поддержкой протокола IP или сетевые игровые приставки) такому офису нужно было бы выделить блок адресов большего размера. А что, если Интернет-провайдер уже распределил все соседние участки адресного пространства? И что нужно знать об управлении IP-адресами типичному домовладельцу? К счастью, существует более простой метод выделения адресов, нашедший широкое применение в подобных ситуациях. Этим методом является (RFC 2663, RFC 3022) так называемая *трансляция сетевых адресов* (Network Address Translation, NAT).

Рисунок 4.27 иллюстрирует работу маршрутизатора, поддерживающего трансляцию сетевых адресов (NAT-маршрутизатора). У этого маршрутизатора есть интерфейс, представляющий собой часть домашней сети (на правой стороне рисунка). Адресация в пределах домашней сети полностью аналогична тому, что мы уже видели — у всех четырех интерфейсов домашней сети один и тот же сетевой адрес 10.0.0/24. От обсуждавшейся выше ситуации данный пример отличается взаимоотношением между домашним маршрутизатором и Интернет-провайдером. На NAT-маршрутизаторе не работает протокол внутренней маршрутизации для связи с маршрутизатором Интернет-провайдера. В самом деле, для внешнего мира NAT-маршрутизатор выглядит как единое устройство с одним IP-адресом — весь трафик, исходящий из домашнего маршрутизатора в Интернет, помечается IP-адресом отправителя 138.76.29.7, а весь прибывающий из Интернета трафик должен иметь IP-адрес получателя 138.76.29.7. Таким образом, NAT-маршрутизатор скрывает от внешнего мира детали домашней сети. (Обычно маршрутизатор получает свой адрес у DHCP-сервера Интернет-провайдера и сам выступает в роли DHCP-сервера для компьютеров своей домашней сети.)

Таблица трансляции сетевых адресов	
Сторона глобальной сети	**Сторона локальной сети**
138.76.29.7, 5001	10.00. 00.1,3345

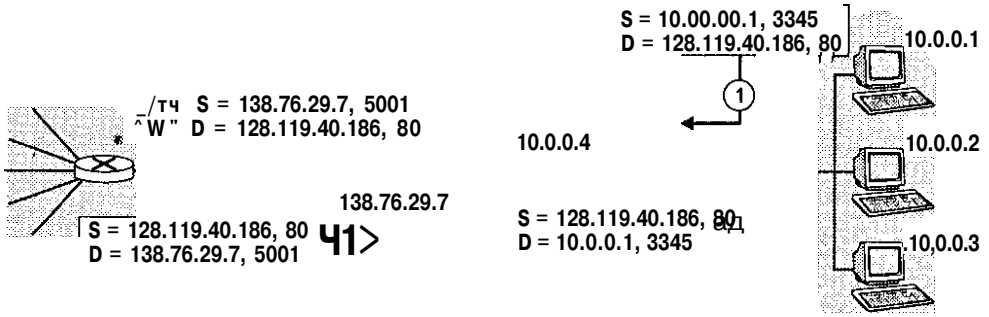


Рис. 4.27. Трансляция сетевых адресов

Теперь вам, вероятно, непонятно, как маршрутизатор определяет, кому предназначается прибывшая из Интернета дейтаграмма, если все дейтаграммы маркируются одним и тем же IP-адресом получателя. Дело в том, что маршрутизатор различает прибывающие дейтаграммы по номерам портов, которые с помощью таблицы трансляции адресов (NAT-таблицы) преобразуются во внутренние адреса и номера портов.

Вернемся к примеру на рисунке. Предположим, пользователь, сидящий за хостом 10.0.0.1, запрашивает web-страницу с некоторого web-сервера (порт 80) с IP-адресом 128.119.40.186. Хост выбирает себе произвольный номер порта отправителя 3345 и отправляет дейтаграмму в локальную сеть. NAT-маршрутизатор получает дейтаграмму, генерирует новый номер порта 5001, которым заменяет оригинальный номер порта, а IP-адрес отправителя заменяет своим IP-адресом 138.76.29.7. Генерируя новый номер порта, NAT-маршрутизатор может выбирать любое число, которого нет в NAT-таблице. (Обратите внимание, что поле номера порта состоит из 16 бит, поэтому протокол NAT может поддерживать более 60 000 одновременных соединений с единственным IP-адресом маршрутизатора!) Web-сервер, не имея представления о том, что полученная им дейтаграмма с HTTP-запросом была обработана NAT-маршрутизатором, отвечает дейтаграммой, в поле адреса получателя которой указан IP-адрес NAT-маршрутизатора, а в поле номера порта— порт 5001. Получив эту дейтаграмму, NAT-маршрутизатор по указанным в дейтаграмме IP-адресу получателя и номеру порта находит в NAT-таблице соответствующий IP-адрес (10.0.0.1) и номер порта (3345) браузера домашней сети. Затем маршрутизатор заменяет оба этих поля найденными в таблице и переправляет дейтаграмму в домашнюю сеть.

В последние годы трансляция сетевых адресов получила широкое распространение. Но мы должны сказать, что многие члены IETF громко возражают против этого метода. Во-первых, заявляют они, номера портов предназначались для адресации процессов, а не хостов. Это нарушение в самом деле может стать причиной проблем для серверов, обслуживающих домашние сети, так как (см. главу 2) сервер ждет входящие обращения к так называемым *хорошо известным номерам портов*. Во-вторых, говорят они, маршрутизаторы должны обрабатывать пакеты только до уровня 3. В-третьих, протокол NAT противоречит так называемому «сквозному аргументу», заключающемся в том, что хосты должны взаимодействовать друг с другом напрямую, без вмешательства узлов, изменяющих IP-адреса и номера портов. И в-четвертых, для решения проблемы нехватки IP-адресов мы должны использовать протокол IPv6 (см. раздел «Протокол IPv6»), а не применять всяческие временные «жучки» и «затычки» вроде трансляции сетевых адресов. Тем не менее, нравится вам это или нет, трансляция сетевых адресов уже стала важной составляющей Интернета.

Наше обсуждение трансляции сетевых адресов было вынужденно кратким. У этого метода есть еще множество других важных аспектов, таких как статическая и динамическая NAT-трансляция, а также влияние NAT на протоколы более высокого уровня и на безопасность. Дополнительные сведения, а также аргументы за и против NAT см. в [78, 385].

Маршрутизация в Интернете

Теперь, когда мы изучили вопросы адресации в Интернете и протокол IP, обсудим протоколы маршрутизации в Интернете, работа которых заключается в определении маршрута следования дейтаграммы от отправителя к получателю. Мы увидим, что в протоколах маршрутизации в Интернете реализованы многие принципы, с которыми мы познакомились раньше в этой главе. Алгоритм маршрутизации, основанный на состоянии линий, и дистанционно-векторный алгоритм, которые обсуждались в разделе «Основы маршрутизации», а также упомянутые в разделе «Иерархическая маршрутизация» автономные системы занимают центральное место в сегодняшнем Интернете.

Как рассказывалось в разделе «Иерархическая маршрутизация», множество маршрутизаторов, находящихся под общим административным и техническим управлением и использующих один и тот же протокол маршрутизации, называется автономной системой. Каждая автономная система, в свою очередь, как правило, состоит из нескольких сетей (здесь под сетью понимается IP-сеть, как было описано в разделе «Интернет-протокол»). Существует два класса протоколов маршрутизации. Первый класс протоколов, используемых для выбора маршрутов дейтаграмм в пределах одной автономной системы и называемых протоколами внутренней маршрутизации, мы рассмотрим в подразделе «Протоколы внутренней маршрутизации». Второй класс протоколов, используемых для выбора маршрутов дейтаграмм между автономными системами и называемых протоколами внешней маршрутизации, мы рассмотрим в подразделе «Протоколы внешней маршрутизации».

Протоколы внутренней маршрутизации

Протоколы внутренней маршрутизации используются для определения маршрутов внутри автономной системы. Эти протоколы также называют внутренними шлюзовыми протоколами (Interior Gateway Protocol, IGP). Исторически в качестве протоколов внутренней маршрутизации в Интернете широко применяются два протокола: RIP и OSPF.

Протокол RIP

Протокол RIP (Routing Information Protocol — протокол маршрутной информации) был одним из первых протоколов внутренней маршрутизации, применявшихся в Интернете; он и в наши дни по-прежнему популярен. Своим происхождением и названием он обязан архитектуре XNS (Xerox Network Systems). Широкое распространение протокола RIP было во многом вызвано тем, что он был включен в версию 1982 года операционной системы Berkeley UNIX, поддерживающей стек протоколов TCP/IP. Протокол RIP версии 1 определен в RFC 1058, обратно совместимая версия 2 этого протокола определена в RFC 2453.

Протокол RIP работает по дистанционно-векторному алгоритму и очень напоминает идеализированный протокол, рассматривавшийся нами в подразделе «Алгоритм дистанционно-векторной маршрутизации» раздела «Основы маршрутизации». Версия протокола RIP, специфицированная в RFC 1058, в качестве единиц

измерения стоимости маршрутов использует количество ретрансляционных участков, то есть стоимость каждой линии считается равной 1. Максимальная стоимость пути ограничена значением 15, таким образом, диаметр автономной системы, поддерживаемой протоколом RIP, не может превышать 15 ретрансляционных участков. Вспомним также, что в дистанционно-векторных протоколах соседние маршрутизаторы обмениваются друг с другом информацией о маршрутах. В протоколе RIP обмен новыми сведениями между соседними маршрутизаторами происходит приблизительно через каждые 30 с, для чего используются так называемые *ответные RIP-сообщения* (RIP response messages). Ответное RIP-сообщение, посылаемое маршрутизатором или хостом, содержит список, в котором указаны до 25 сетей-адресатов в пределах автономной системы, а также расстояния до каждой из этих сетей от отправителя. Ответные RIP-сообщения также иногда называют RIP-объявлениями.

Рассмотрим работу RIP-объявлений на простом примере. На рис. 4.28 показан фрагмент автономной системы. Линии, соединяющие маршрутизаторы, обозначают сети. Для удобства мы будем рассматривать только несколько выделенных маршрутизаторов (A, B, C и D) и сетей (w, x, y и z). Пунктирные линии означают, что автономная система не ограничивается помеченными маршрутизаторами и сетями, а распространяется дальше.

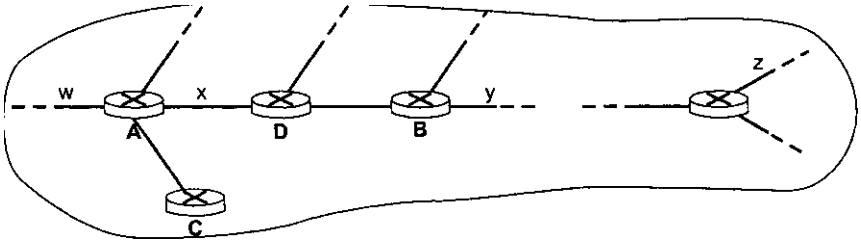


Рис. 4.28. Фрагмент автономной системы

Предположим теперь, что таблица продвижения данных маршрутизатора D выглядит так, как показано в табл. 4.5. Обратите внимание, что в этой таблице три столбца. В первом столбце указана сеть-адресат, во втором — идентификатор следующего маршрутизатора на кратчайшем пути к сети-адресату, в третьем — количество ретрансляционных участков (хопов) до сети-адресата на кратчайшем пути, то есть количество сетей, отделяющих отправителя от получателя, включая сеть-адресат. Видно, что для отправки дейтаграммы от маршрутизатора D в сеть w дейтаграмму сначала нужно переправить соседнему маршрутизатору A. Кроме того, сеть-адресат w находится на расстоянии двух ретрансляционных участков по самому кратчайшему пути. Аналогично, до сети z семь ретрансляционных участков через маршрутизатор B. В принципе, в таблице продвижения данных содержится по одной строке для каждой сети в автономной системе, хотя версия 2 протокола RIP допускает агрегацию маршрутов к сетям при помощи метода, схожего с теми, которые мы рассматривали в подразделе «Адресация в протоколе IPv4» раздела «Интернет-протокол». В таблице продвижения данных также содержится, по мень-

шей мере, по одной строке для пути к каждой сети за пределами автономной системы. Таким образом, табл. 4.5 и последующие таблицы продвижения данных, представленные в этом подразделе, являются полными только отчасти.

Таблица 4.5. Таблица продвижения данных маршрутизатора D до получения им объявления маршрутизатора A

Сеть-адресат	Следующий маршрутизатор	Количество хопов до адресата
w	A	2
y	B	2
z	B	7
x	—	1

Теперь предположим, что 30 с спустя маршрутизатор D получает от маршрутизатора A объявление, показанное в табл. 4.6. Обратите внимание, что это объявление представляет собой не что иное, как информацию из таблицы продвижения данных маршрутизатора A! Эта информация указывает, в частности, что сеть z находится на расстоянии всего четырех ретрансляционных участков от маршрутизатора A. Получив это объявление, маршрутизатор D узнает, что теперь появился путь от маршрутизатора A до сети z, который короче, чем путь через маршрутизатор B. Таким образом, маршрутизатор D обновляет свою таблицу продвижения данных, чтобы учесть более короткий кратчайший путь, как показано в табл. 4.7. Как же так, возможно, спросите вы, кратчайший путь к сети z стал еще короче? Возможно, децентрализованный дистанционно-векторный алгоритм все еще находился в процессе схождения (см. раздел «Основы маршрутизации») или в него были добавлены новые линии и/или маршрутизаторы, в результате чего в сети появились новые кратчайшие маршруты.

Таблица 4.6. Объявление маршрутизатора A

Сеть-адресат	Следующий маршрутизатор	Количество хопов до адресата
z	C	4
w	—	1
x	-	1

Таблица 4.7. Таблица продвижения данных маршрутизатора D после получения им объявления маршрутизатора A

Сеть-адресат	Следующий маршрутизатор	Количество хопов до адресата
w	A	2
y	B	2
z	A	5

Рассмотрим теперь некоторые аспекты реализации протокола RIP. Вспомним, что использующие этот протокол маршрутизаторы обмениваются объявлениями приблизительно раз в 30 с. Если маршрутизатор не получает пакетов от своего соседа в течение 180 с, он решает, что данный сосед более недоступен. Это может означать, что сосед вышел из строя или выключен либо вышла из строя связывавшая их линия связи. Когда такое происходит, протокол RIP изменяет локальную таблицу продвижения данных, а затем распространяет эту информацию, рассылая объявления соседним маршрутизаторам (тем, которые все еще доступны). Маршрутизатор может также запросить у соседа информацию о стоимости маршрута от него до заданного адресата при помощи RIP-запроса. Маршрутизаторы пересылают друг другу RIP-запросы и RIP-ответы в UDP-пакетах через порт номер 520. UDP-пакет переносится между маршрутизаторами в стандартном IP-пакете. Тот факт, что протокол RIP пользуется протоколом транспортного уровня (UDP) поверх протокола сетевого уровня (IP), может показаться излишне сложным (так оно и есть!). Чтобы прояснить данный вопрос, необходимо несколько углубиться в реализацию протокола RIP.

На рис. 4.29 изображена схема типичной реализации протокола RIP в операционной системе UNIX, например на рабочей станции, играющей роль маршрутизатора. Протокол RIP исполняет процесс, называемый *routed;* он обрабатывает информацию о маршрутах и обменивается сообщениями с такими же процессами, работающими на соседних маршрутизаторах. Поскольку протокол RIP реализован как процесс прикладного уровня (хотя и весьма специфический, например, он способен управлять таблицами продвижения данных, находящихся в ядре операционной системы UNIX), он может отправлять и получать сообщения через стандартный сокет и использовать стандартный транспортный протокол. Таким образом, протокол RIP представляет собой протокол прикладного уровня (см. главу 2), работающий поверх протокола UDP.

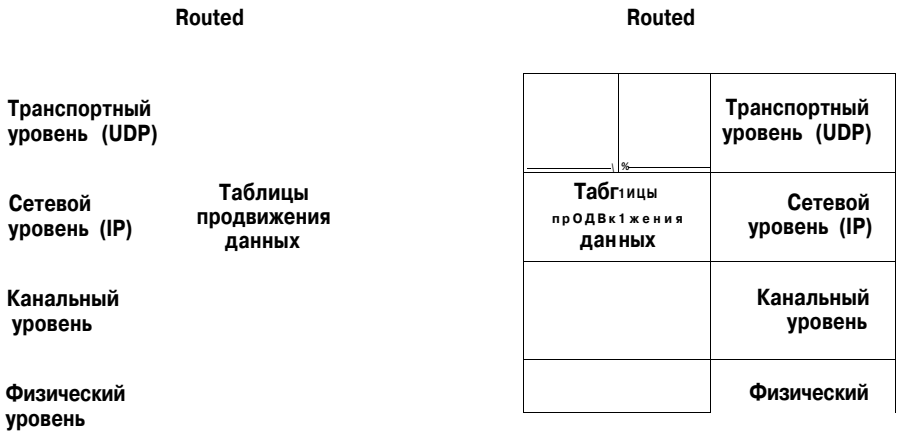


Рис. 4.29. Реализация протокола RIP в виде демона «routed»

Наконец, рассмотрим реальную таблицу продвижения данных протокола RIP (табл. 4.8), взятую с маршрутизатора giroflee.eurecom.fr, работающего под управлением операционной системы UNIX. Если в операционной системе UNIX выполнить команду netstat -rn, вы можете увидеть таблицу продвижения данных (в действительности называемую в документации netstat «таблицей маршрутизации») для данного хоста или маршрутизатора.

Таблица 4.8. Таблица маршрутизации протокола RIP с маршрутизатора giroflee.eurecom.fr

Адресат	Шлюз	Флаги	Ref	Использование	Интерфейс
127.0.0.1	127.0.0.1	UH	0	26492	loO
192.168.2.	192.168.2.5	и	2	13	faO
193.55.114.	193.55.114.6	и	∞	58503	leO
192.168.3.	192.168.3.5	и	2	25	gaaO
224.0.0.0	193.55.114.6	и	∞	0	leO
По умолчанию	193.55.114.129	UG	0	143454	

Маршрутизатор giroflee соединен с тремя сетями. Вторая, третья и четвертая строки таблицы сообщают нам, что эти три сети присоединены к маршрутизатору giroflee через сетевые интерфейсы faO, LeO и gaaO. Адреса этих интерфейсов равны 192.168.2.5, 193.55.114.6 и 192.168.3.5 соответственно. Для передачи пакета любому хосту, принадлежащему одной из этих трех сетей, маршрутизатор giroflee просто посылает IP-дейтаграмму соответствующему интерфейсу. Особый интерес для нас представляет *маршрут по умолчанию*. Любая IP-дейтаграмма, не предназначающаяся одной из сетей, явно перечисленных в таблице продвижения данных, будет переправляться маршрутизатору с IP-адресом 193.55.114.129. Дейтаграммы посылаются этому маршрутизатору через сетевой интерфейс по умолчанию. Первая запись в таблице продвижения данных представляет собой так называемый кольцевой интерфейс. Когда протокол IP посылает дейтаграмму по кольцевому интерфейсу, пакет просто возвращается протоколу IP. Это бывает полезно при отладке. Адрес 224.0.0.0 является специальным групповым (класса D) IP-адресом. Мы рассмотрим групповую рассылку протокола IP в разделе «Групповая маршрутизация».

Протокол OSPF

Как и RIP, протокол OSPF (Open Shortest Path First — открытый протокол выбора кратчайшего маршрута) используется для маршрутизации внутри автономной системы. Слово «Ореп» в названии протокола означает, что спецификация протокола маршрутизации свободно распространяется (в отличие от, к примеру, спецификации протокола EIGRP корпорации Cisco). Последняя (вторая) версия протокола OSPF определена в RFC 2328.

Протокол OSPF считается преемником протокола RIP и обладает рядом дополнительных функций. Однако по своей сути протокол OSPF представляет собой протокол, основанный на учете состоянии линий и использующий метод лавинной рассылки для распространения информации о состоянии линий, а также алгоритм

определения пути наименьшей стоимости Дейкстры. Маршрутизатор, работающий по протоколу OSPF, формирует полную топологическую карту (направленный граф) всей автономной системы. Затем маршрутизатор локально запускает алгоритм определения кратчайшего пути Дейкстры, чтобы найти дерево кратчайших путей ко всем сетям автономной системы. Далее из этого дерева кратчайших путей формируется таблица продвижения данных маршрутизатора. Стоимости линий настраиваются сетевым администратором. Администратор может установить стоимости всех линий равными 1, в результате путь наименьшей стоимости совпадет с кратчайшим путем, или установить весовой коэффициент каждой линии обратно пропорциональным пропускной способности линии, чтобы маршрутизаторы старались избегать линий с низкой пропускной способностью. Протокол OSPF не занимается определением стоимости линий (это работа сетевого администратора), а лишь предоставляет механизмы (протокол) определения пути наименьшей стоимости для заданного набора стоимостей линий.

Маршрутизатор, работающий по протоколу OSPF, путем широковещательной рассылки переправляет информацию о маршрутах всем маршрутизаторам автономной системы, а не только соседним. Прекрасное описание алгоритмов широковещательной рассылки, основанных на учете состояния линий, содержится в [384]. Маршрутизатор рассылает всем информацию о состоянии линий при каждом изменении состояния какой-либо из линий (например, при изменении стоимости или включении/отключении). Он также рассылает информацию о состоянии линий периодически (по меньшей мере, раз в 30 мин), даже если состояние линии не изменилось. В RFC 2328 отмечается, что «эти периодические объявления о состояниях линий увеличивают устойчивость алгоритма, основанного на состоянии линий». Объявления протокола OSPF содержатся в OSPF-сообщениях, напрямую переносимых IP-дейтаграммами, в поле протокола верхнего уровня которых протокол OSPF обозначается кодом 89. Таким образом, протокол OSPF должен сам заниматься такими вопросами, как надежность передачи сообщений и широковещательная рассылка информации о состоянии линий. Протокол OSPF также проверяет работоспособность линий (при помощи сообщения HELLO, посылаемого соседу) и позволяет маршрутизатору OSPF получать информацию из базы данных соседнего маршрутизатора о состоянии линий всей сети.

Ниже перечислены достоинства протокола OSPF.

- *Безопасность.* Весь обмен данными между OSPF-маршрутизаторами (например, новые сведения о состоянии линий) проходит процедуру аутентификации. Это означает, что только доверенные маршрутизаторы могут принимать участие в работе протокола OSPF в пределах домена, не позволяя, таким образом, злоумышленникам (или студентам, применяющим полученные знания для развлечений) вставлять в таблицы маршрутизации некорректную информацию.
- *Несколько путей с одинаковой стоимостью.* Когда у нескольких маршрутов к одному адресату оказывается одинаковая стоимость, протокол OSPF позволяет использовать все (нет проблемы выбора одного из этих маршрутов).
- *Интегрированная поддержка многоадресной и одноадресной маршрутизации.* Протокол MOSPF (Multicast OSPF — протокол OSPF для групповой рассыл-

напрямую присоединены. Глобальная информация о маршрутах к удаленным адресатам распространяется от одной автономной системы к другой путем обмена данными между напрямую соединенными друг с другом равноправными BGP-узлами. Следует также отметить, что протокол BGP направляет пакеты сетям-адресатам (имеются в виду IP-сети), а не конкретным маршрутизаторам или хостам. Как только дейтаграмма достигает сети-адресата, ее пересылкой к конечному получателю начинает заниматься протокол внутренней маршрутизации. Протокол BGP распознает автономные системы (RFC 1930) по глобально уникальным *номерам автономных систем* (Autonomous System Number, ASN). На практике не у каждой автономной системы есть номер. В частности, так называемая тупиковая автономная система (автономная система-заглушка), переносящая только трафик, для которого он является отправителем или получателем, как правило, не имеет номера. В нашем обсуждении мы опустим эти детали. Номера автономных систем, как и IP-адреса, назначаются региональными организациями ICANN [508].

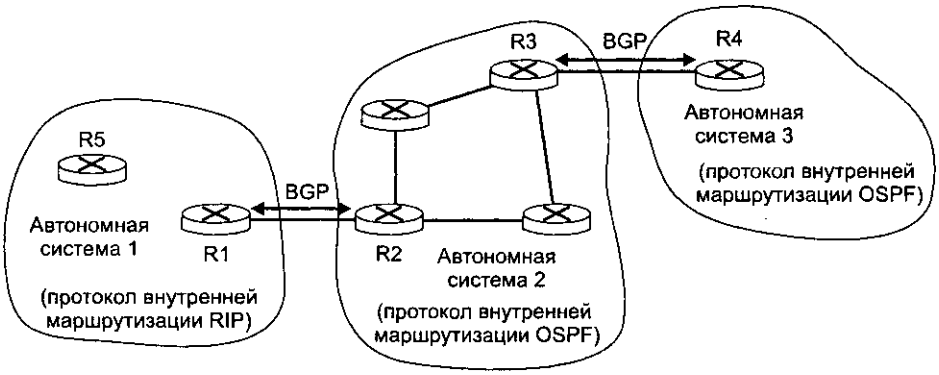


Рис. 4.31. Использование протокола BGP для междоменной маршрутизации

В основе протокола BGP лежат объявления о маршрутах. Объявление о маршрутах посылается одним равноправным BGP-узлом другому равноправному BGP-узлу по соединению «точка-точка». Объявление состоит из адреса сети (например, 128.119.40/24) и набора атрибутов, ассоциированных с маршрутом к этой сети. Двумя наиболее важными атрибутами являются атрибут маршрута (явный список всех автономных систем на пути к указанной сети-адресату) и идентификатор следующего маршрутизатора на пути к указанной сети-адресату. Полное описание атрибутов маршрута см. в [189, 195, 488].

Работа протокола BGP в основном состоит из трех действий с объявлениями.

1. *Получение и фильтрация объявлений о маршрутах от напрямую присоединенных соседей.* BGP-маршрутизатор получает объявления о маршрутах от равноправного BGP-узла. Мы можем рассматривать полученное BGP-объявление о маршруте как обещание. Равноправный BGP-узел, посылающий объявление о маршруте к некоторой автономной системе, обещает, что, если соседняя автономная система пошлет ему дейтаграмму, адресованную указанной автономной системе, тогда он сможет переправить эту дейтаграмму далее по пути к адре-

сату. BGP-маршрутизатор может также отфильтровывать (отбрасывать) полученные объявления о маршрутах. Например, BGP-маршрутизатор будет игнорировать объявления, содержащие номер его собственной автономной системы, указанный в качестве автономной системы-получателя, так как использование такого маршрута привело бы к появлению маршрутной петли. Поскольку маршрут указывается полностью, сетевой администратор может в значительной степени контролировать маршруты дейтаграмм его сети. Например, в автономной системе Hatfield.net можно реализовать политику, заключающуюся в том, чтобы трафик из автономной системы Hatfield.net не попадал в автономную систему McCoy.net (семейства Hatfield и McCoy, живущие в США, представляют собой два знаменитых враждующих клана).

2. *Выбор маршрута.* BGP-маршрутизатор может получить несколько объявлений о маршрутах к одной и той же автономной системе-адресату. В этом случае он должен выбрать из объявленных маршрутов один. Данные об автономной системе-адресате и следующем ретрансляционном участке для выбранного маршрута должны быть помещены в таблицы продвижения данных (как, например, было показано на рис. 4.11). BGP-маршрутизатору могут быть известны несколько маршрутов к одному адресату, но в таблице продвижения данных, как правило, указывается лишь один маршрутизатор по направлению к данному адресату.

   Но как BGP-маршрутизатор выбирает из нескольких маршрутов один? В протоколе BGP проводится четкое разграничение между *механизмом маршрутизации* и *политикой маршрутизации.* В частности, протокол BGP *не* определяет, как автономная система должна выбирать один маршрут из нескольких предложенных. Это *политическое* решение, оставляемое на усмотрение сетевого администратора автономной системы. Администратор сети может указать так называемые локальные предпочтения, например, что при наличии выбора маршрутизация через соседнюю автономную систему A всегда предпочтительнее маршрутизации через соседнюю автономную систему B. В отсутствие локальных предпочтений, как правило, выбирается кратчайший маршрут, то есть маршрут, состоящий из наименьшего количества автономных систем. Обсуждение алгоритмов выбора оптимального маршрута содержится в [77].

3. *Передача объявлений о маршрутах своим соседям.* BGP-маршрутизатор получает объявления о маршрутах от своих соседей, а также передает объявления о маршрутах своим соседям. Протокол BGP предоставляет механизм для таких объявлений, но не политику. В результате администратор сети получает значительный уровень контроля над трафиком, поступающим в его сеть. Применительно к нашему примеру с семействами Hatfield и McCoy, клан Hatfield легко может запретить трафику клана McCoy проходить через свою сеть. Например, если сеть клана McCoy расположена по соседству с сетью клана Hatfield, клан Hatfield может просто не информировать о маршрутах к сети клана McCoy, проходящих через сеть клана Hatfield. Однако эффективность ограничения трафика путем контролирования объявлений автономной системы о маршрутах может быть не полной. Например, если сеть Петровых располагается между сетями семейств Hatfield и McCoy, и клан Hatfield информируют о маршрутах

к сети Петровых, проходящих через сеть клана Hatfield, тогда клан Hatfield не может запретить (с помощью протокола BGP) Петровым сообщить об этих маршрутах клану McCoy.

Проиллюстрируем некоторые из основных концепций информирования о маршрутах несколько более реалистическим примером, чем пример с семействами Hatfield и McCoy. На рис. 4.32 показаны шесть соединенных друг с другом автономных систем: A, B, C, W, X и Y. Необходимо отметить, что A, B, C, W, X и Y — это *сети,* а не маршрутизаторы. Пусть автономные системы W, X и Y представляют собой тупиковые сети, а автономные системы A, B и C являются сетями магистральных Интернет-провайдеров. Весь попадающий в *тупиковую сеть* трафик должен предназначаться этой сети, а весь покидающий тупиковую сеть трафик должен генерироваться узлами этой сети. Если взглянуть на рисунок, должно быть очевидно, что сети W и Y являются тупиковыми, а сеть X называют «многодомовой» тупиковой сетью, так как она соединяется с остальными сетями через двух разных Интернет-провайдеров (ситуация, становящаяся все более популярной в последнее время). Однако, как и сети W и Y, сеть X должна быть источником/адресатом всего исходящего/входящего трафика сети X. Но как реализовать и гарантировать такое поведение? Как сеть X может не пропускать трафик между сетями B и C? Этого легко добиться, контролируя способ, которым рекламируются BGP-маршруты. В частности, сеть X будет функционировать как тупиковая, если она объявит (своим соседям, сетям B и C), что у нее нет путей к другим адресатам, кроме самой сети X. То есть, даже если сети X известен маршрут до сети Y, например XCY, она *не* будет сообщать этот маршрут сети B. Поскольку сеть B не знает, что у сети X есть маршрут до сети Y, сеть B никогда не станет направлять трафик, предназначенный сети Y (или C) через сеть X. Этот простой пример иллюстрирует механизм выборочного информирования о маршрутах для реализации связанных с маршрутизацией отношений между клиентом и поставщиком услуг.

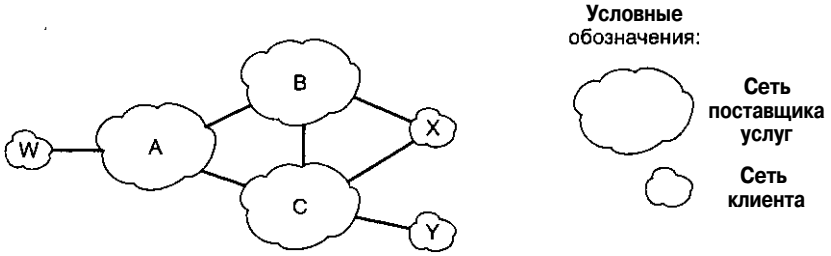


Рис. 4.32. Простой сценарий работы протокола BGP

Рассмотрим теперь сеть поставщика услуг, например автономную систему B. Предположим, сеть B узнала (от сети A), что у сети A есть путь AW к сети W. В результате сеть B может добавить в свою маршрутную базу данных маршрут BAW. Естественно, сеть B захочет сообщить о маршруте BAW своему клиенту X, чтобы клиент X знал, что он может направлять данные сети W через сеть B. Но должна ли сеть B рекламировать маршрут B AW сети C? Если она это сделает, тогда сеть C сможет направлять трафик в сеть W по маршруту CBAW. Если все сети A, B и C

представляют собой магистральные сети Интернет-провайдеров, тогда сеть B может решить, что ей нет резона взваливать на себя бремя переноса трафика между сетями A и C. Поскольку владельцы сетей A и C взимают со своих клиентов плату за продвижение данных, будет только справедливо, если они для пересылки данных друг другу воспользуются прямым соединением между ними. Сегодня нет официальных «стандартов», определяющих, как Интернет-провайдеры должны поступать с трафиком друг друга. Однако на практике Интернет-провайдеры придерживаются простого правила, состоящего в том, что любой трафик, пересекающий магистральную сеть Интернет-провайдера, должен либо генерироваться, либо приниматься в сети, являющейся клиентом этого Интернет-провайдера. В противном случае подобный трафик будет восприниматься Интернет-провайдером как «безбилетный пассажир». Индивидуальные соглашения, регулирующие подобные спорные вопросы, как правило, заключаются парами Интернет-провайдеров и зачастую являются конфиденциальными. Интересное обсуждение подобных соглашений содержится в [225].

Обсудив в общих чертах некоторые политические вопросы, связанные с протоколом BGP, мы можем вернуться к внутренним механизмам протокола BGP. Равноправные BGP-узлы обмениваются данными при помощи протокола TCP через порт 179. Таким образом, протокол TCP обеспечивает надежный обмен сообщениями между двумя равноправными BGP-узлами с контролем перегрузки. Для сравнения вспомните, как общаются равноправные RIP-узлы, например программы *routed на* рис. 4.29, обменивающиеся данными при помощи ненадежного транспортного протокола UDP, а также, что для обмена OSPF-сообщениями используется собственный протокол OSPF. В протоколе BGP определены четыре типа сообщений: OPEN, UPDATE, KEEPALIVE и NOTIFICATION.

- *OPEN* (открытие). Когда BGP-маршрутизатор желает установить контакт с равноправным BGP-узлом (например, после того как сам маршрутизатор был загружен или была включена линия связи), сообщение OPEN пересылается равноправному BGP-узлу. Сообщение OPEN позволяет BGP-маршрутизатору идентифицировать и аутентифицировать себя, а также синхронизироваться. Если получивший сообщение OPEN равноправный BGP-узел считает это для себя приемлемым, он отвечает на него сообщением KEEPALIVE.
- *UPDATE* (обновление). BGP-маршрутизатор использует сообщение UPDATE, чтобы объявить равноправному BGP-узлу о маршруте к определенному адресату. Сообщение UPDATE может также использоваться для того, чтобы аннулировать объявленный ранее маршрут (то есть сообщить равноправному BGP-узлу, что объявленный ранее маршрут более не действителен). BGP-маршрут считается действительным до тех пор, пока он не будет явно аннулирован.
- *KEEPALIVE* (дежурное сообщение). Это BGP-сообщение позволяет известить равноправный BGP-узел о том, что отправитель сообщения продолжает нормальную деятельность и у него нет другой информации для передачи. Это сообщение также служит подтверждением полученного сообщения OPEN.
- *NOTIFICATION* (уведомление). Это BGP-сообщение используется для информирования равноправного BGP-узла об обнаруженной ошибке (например, в пе-

реданном ранее BGP-сообщении) или о том, что отправитель собирается завершить BGP-сеанс.

ПРИНЦИПЫ И ПРАКТИКА

Изучив детали конкретных протоколов для внутренней и внешней маршрутизации, реализованных в сегодняшнем Интернете, закончим наше изучение, возможно, одним из наиболее фундаментальных вопросов, касающихся протоколов маршрутизации: *зачем нужны разные протоколы для внутренней и внешней маршрутизации?* **Ответ на этот вопрос кроется в различном назначении маршрутизации внутри автономных систем и между автономными системами.**

- *Политика.* **Среди автономных систем политические вопросы доминируют. Может быть важным, чтобы трафик, сгенерированный в некоторой автономной системе, не мог проходить через другую конкретную автономную систему. Аналогично, конкретная автономная система может пожелать контролировать транзитный трафик, переносимый между другими автономными системами. Мы видели, что протокол BGP передает атрибуты маршрутов и предоставляет возможность контролируемого распределения информации о маршрутах, что позволяет принимать политические решения о выборе маршрутов.**
- *Масштабирование.* **Способность алгоритма маршрутизации и его структур данных к масштабированию в целях поддержания большого количества сетей является ключевой для внешней маршрутизации. В пределах одной автономной системы значимость масштабирования не так велика, поскольку, если какой-либо административный домен становится слишком большим, его всегда можно разбить на две автономные системы поменьше и соединить их с помощью механизмов внешней маршрутизации. (Вспомним также, что протокол OSPF позволяет разделять автономную систему на отдельные области, создавая таким образом иерархическую структуру.)**
- *Производительность.* **Поскольку политические вопросы играют во внешней маршрутизации столь важную роль, качество обслуживания в используемых маршрутах (например, производительность) часто оказывается на втором плане (то есть более длинному или более дорогому маршруту, удовлетворяющему определенным политическим критериям, может быть оказано предпочтение по сравнению с более коротким маршрутом, который не удовлетворяет этим критериям). Действительно, мы видели, что при маршрутизации между автономными системами стоимость даже не упоминается (не считая количества ретрансляционных участков). Однако в пределах одной автономной системы подобные политические соображения не столь важны, что позволяет при выборе маршрута больше внимания уделить производительности.**

До сих пор в нашем обсуждении мы уделяли внимание вопросам использования протокола BGP для выбора маршрута между маршрутизаторами разных автономных систем, то есть так называемому *протоколу E-BGP* (External BGP — внешний протокол BGP). Однако существует и другая версия протокола BGP, называемая *протоколом I-BGP* (Internal BGP — внутренний протокол BGP). Протокол I-BGP применяется для доставки находящимся в той же автономной системе маршрутизаторам информации о маршрутах, касающейся адресатов, расположенных за пределами автономной системы. Чтобы понять, зачем нужен протокол I-BGP, вернемся к рис. 4.31. Как может маршрутизатор R5 получить информацию о маршрутах к удаленным автономным системам, например, AS2 и AS3? Как уже было показано, протокол внутренней маршрутизации, например RIP, позволяет указать марш-

рут по умолчанию. По этому маршруту отправляются дейтаграммы, адресат которых не указан явно в таблице продвижения данных. Другая возможность состоит в использовании протокола I-BGP для распределения внутри автономной системы информации о маршрутах, касающейся адресатов, расположенных за пределами автономной системы. В протоколах I-BGP и E-BGP применяются одинаковые форматы сообщений и форматы атрибутов путей, но между этими двумя протоколами имеется несколько существенных различий. Все I-BGP-маршрутизаторы в пределах одной автономной системы логически связаны друг с другом. То есть все I-BGP-маршрутизаторы в пределах одной автономной системы считаются соседями друг друга. Кроме того, I-BGP-маршрутизаторы ограничены в способах информирования о маршрутах, о которых они узнают от других I-BGP-маршрутизаторов (в отличие от E-BGP-маршрутизаторов). Обсуждение протокола I-BGP содержится в [189,195,488].

Как уже отмечалось, протокол BGP является стандартом де-факто для маршрутизации внутри автономных систем в Интернете. Например, протокол BGP используется в основных точках доступа в сеть (Network Access Point, NAP), через которые крупнейшие Интернет-провайдеры связываются друг с другом и обмениваются трафиком. Содержимое самой «свежей» (хотя и устаревшей менее чем на два часа) таблицы маршрутизации протокола BGP (гигантской!) на одной из основных точек доступа в сеть США можно найти в [223]. Статистика размеров и характеристик таблиц маршрутизации протокола BGP представлена в [178].

На этом мы завершаем наш краткий обзор протокола BGP. Несмотря на свою сложность этот протокол играет центральную роль в Интернете. Вы можете узнать больше о протоколе BGP, если заглянете в [189, 195, 220, 285, 488].

Устройство маршрутизатора

До сих пор в этой главе мы рассматривали модели обслуживания сетевого уровня, алгоритмы маршрутизации и реализующие их протоколы. Однако эти вопросы представляют собой лишь часть (хотя и важную) того, что происходит на сетевом уровне. Мы еще не изучали коммутирующую функцию маршрутизатора — процесс передачи дейтаграмм с входных линий маршрутизатора на его выходные линии. Ограничившись при знакомстве с сетевым уровнем только вопросами управления и обслуживания, мы уподобляемся нерадивым акционерам, которые при изучении компании считают достаточным встретиться с ее управленческим персоналом (он, хотя и руководит компанией, как правило, выполняет очень мало фактической работы!) и посетить ее рекламный отдел («только наш продукт предоставит вам эту замечательную услугу!»). Чтобы полнее представлять себе, чем в действительности занимается компания, необходимо говорить с непосредственными исполнителями. На сетевом уровне фактическая работа (то есть то, ради чего существует сетевой уровень) заключается в продвижении дейтаграмм, а ее исполнителями являются маршрутизаторы. Ключевой составляющей продвижения дейтаграмм является их передача с входной линии маршрутизатора на его выходную линию. Обсуждению этого процесса и посвящен данный раздел. Наше обсужде-

ние будет вынужденно кратким, так как для подробного изучения устройства маршрутизатора потребовался бы отдельный курс. Соответственно, мы постараемся предоставить необходимые ссылки на материал, в котором этот вопрос обсуждается более подробно.

Условная схема маршрутизатора показана на рис. 4.33. Маршрутизатор состоит из четырех компонентов.

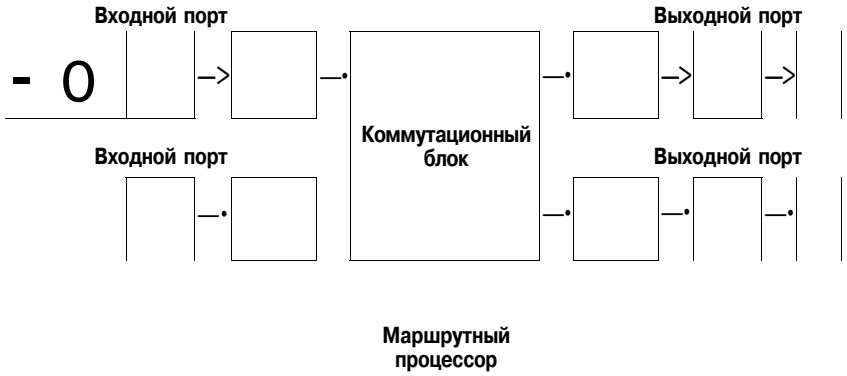


Рис. 4.33. Архитектура маршрутизатора

- *Входные порты.* Входной порт выполняет несколько функций. Он выполняет функции физического уровня (самый левый прямоугольник входного порта и самый правый прямоугольник выходного порта на рис. 4.33), завершая входную физическую линию маршрутизатора. Он также осуществляет функции канального уровня (средние прямоугольники входного и выходного портов), необходимые для взаимодействия с функциями канального уровня (см. главу 5) на другой стороне линии связи. Еще он выполняет функции поиска и продвижения данных (самый правый прямоугольник входного порта и самый левый прямоугольник выходного порта), так что пакет, переправленный в коммутационный блок маршрутизатора, на выходе из него появляется из того порта, из которого следует. Управляющие пакеты (например, пакеты, содержащие информацию протокола RIP, OSPF или BGP) продвигаются из входного порта в маршрутный процессор. На практике несколько портов часто объединяют на одной канальной карте маршрутизатора.
- *Коммутационный блок.* Коммутационный блок соединяет входные порты маршрутизатора с его выходными портами. Коммутационный блок целиком располагается внутри маршрутизатора — сеть внутри сетевого маршрутизатора!
- *Выходные порты.* Выходной порт хранит пакеты, переправленные ему через коммутационный блок, а затем передает пакеты по выходной линии. Таким образом, выходной порт осуществляет функции физического и канального уровней, обратные функциям входного порта. В случае двунаправленной линии связи (то есть когда линия передает данные в оба направления) выходной порт

линии связи, как правило, составляет пару с входным портом этой линии, располагаясь на той же самой карте канала.

- *Маршрутный процессор.* Маршрутный процессор выполняет функции протоколов маршрутизации (например, тех, которые мы изучали в разделе «Маршрутизация в Интернете»), обрабатывает информацию о маршрутах, а также выполняет функции управления сетью (см. главу 8) в маршрутизаторе. Поскольку эта тема будет рассматриваться в главе 8, мы не станем обсуждать ее здесь.

В следующих подразделах мы более подробно рассмотрим входные и выходные порты, а также блок коммутации. Обсуждение архитектур маршрутизаторов имеется в [178, 324, 370, 524]. В [323] дается обзор современных архитектур маршрутизаторов на примере маршрутизатора Cisco 12000.

Входные порты

Более детальная, чем на рис. 4.35, функциональная схема входного порта приведена на рис. 4.34. Как уже упоминалось, блок завершения физической линии входного порта маршрутизатора и блок обработки канального уровня реализуют физический и канальный уровни входной линии маршрутизатора. Блок поиска/продвижения данных входного порта является центральным для системы коммутации маршрутизатора. Во многих маршрутизаторах именно здесь маршрутизатор определяет выходной порт, которому будет передан принятый пакет через коммутационный блок. Выбор выходного порта осуществляется при помощи информации, содержащейся в таблице продвижения данных. Хотя таблица продвижения данных вычисляется маршрутным процессором, локальнная копия таблицы продвижения данных, как правило, сохраняется на каждом входном порту и, при необходимости обновляется маршрутным процессором. Наличие локальных копий таблицы продвижения данных позволяет принимать решения о коммутации локально на каждом входном порту, не занимая централизованный маршрутный процессор. Подобная централизованная коммутация позволяет избежать заторов на входе в маршрутизатор.

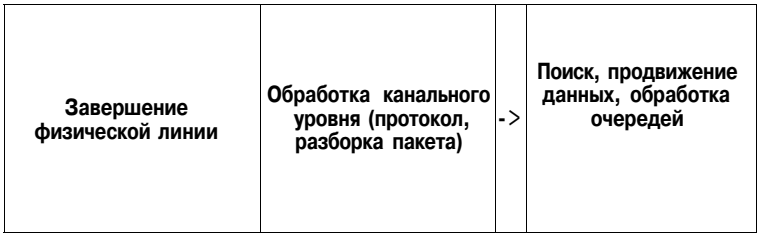


Рис. 4.34. Обработка на входном порту

На маршрутизаторах с ограниченными мощностями процессоров входного порта входной порт может просто переправлять пакет централизованному маршрутному процессору, чтобы тот сам осуществлял поиск в таблице продвижения данных и переправлял пакет в соответствующий выходной порт. Такой подход предпринимается, когда рабочая станция или сервер выполняет функции маршрутизатора.

В данном случае роль маршрутного процессора исполняет центральный процессор рабочей станции, а входной порт представляет собой просто сетевую интерфейсную карту (например, Ethernet-карту).

ИСТОРИЧЕСКАЯ СПРАВКА

На момент написания этой книги (февраль 2002) в компании Cisco работают более 30 000 сотрудников, а капитализация составляет 150 миллиардов долларов. Сегодня эта компания доминирует в производстве Интернет-маршрутизаторов и занимает неплохие позиции в производстве Интернет-телефонии, где она в последние годы конкурирует с такими «гигантами» разработки телефонного оборудования, как Lucent, Alcatel, Northern Telecom и Siemens. Как этой компании удалось занять столь высокое место на рынке изготовителей сетевого и телефонного оборудования? Все началось в 1984 году (всего 18 лет назад) в общежитии Silicon Valley.

Лен Босак и его жена Санди Лернер работали в университете Станфорда, когда у них возникла идея создавать и продавать Интернет-маршрутизаторы научно-исследовательским институтам и университетам. Санди Лернер придумала название компании Cisco (сокращение от Сан-Франциско), кроме того, она разработала логотип компании. Изначально штаб-квартира корпорации располагалась в их комнате в общежитии, а проект финансировался из их собственных сбережений и доходов от работы по совместительству в качестве консультантов. К концу 1986 г. доходы компании Cisco достигли 250 000 долларов в месяц — неплохо для бизнеса, изначально финансировавшегося кредитными картами и не имевшего капитала. К концу 1987 года корпорации Cisco, наконец, удалось привлечь в свой бизнес 2 миллиона долларов от Sequoia Capital в обмен на одну треть акций. За несколько следующих лет корпорация Cisco продолжала расти и захватывать все больший сектор рынка. В то же время отношения между супругами Босак/Лернер и менеджерами Cisco испортились. В 1990 году корпорация Cisco стала открытым акционерным обществом, и в том же году Лернер и Босак ушли из компании.

При наличии таблицы продвижения данных поиск представляет собой относительно простую задачу — мы просто просматриваем таблицу продвижения данных, ища запись, которая лучше всего соответствует сетевому адресу получателя пакета. Если такую запись в таблице найти не удается, для передачи пакета выбирается маршрут по умолчанию. (В подразделе «Адресация в протоколе IPv4» раздела «Интернет-протокол» отмечалось, что лучшим соответствием адресу получателя пакета считается табличная запись с самым длинным сетевым префиксом, совпадающим с адресом получателя пакета.) На практике, однако, все не так просто. Возможно, наиболее важный усложняющий фактор состоит в том, что магистральные маршрутизаторы должны работать на высоких скоростях, выполняя миллионы операций поиска в секунду. В самом деле, желательно, чтобы входной порт мог работать на *скорости линии,* то есть операция поиска должна выполняться быстрее операции приема пакета во входной порт. В этом случае обработка полученного пакета может быть выполнена прежде, чем завершится операция получения следующего пакета. Чтобы получить представление о необходимой производительности операции поиска, рассмотрим так называемую линию OC48, передающую данные на скорости 2,5 Гбит/с. При длине пакетов в 256 байт входной порт должен успевать выполнять приблизительно миллион операций поиска в секунду.

Поскольку скорости передачи данных в современных линиях связи очень высокие, линейный поиск в таблице продвижения данных просто невозможен. Более разумный подход заключается в хранении таблицы продвижения данных в виде дерева, каждый уровень которого соответствует одному двоичному разряду адреса получателя. Поиск адреса начинается с вершины дерева. Если первый бит адреса равен нулю, тогда дальнейший поиск ведется по левому поддереву; в противном случае адрес получателя должен находиться в правом поддереве. На каждом шаге просматривается один разряд адреса получателя и выбирается одна ветвь дерева из двух. Таким образом, всю таблицу продвижения данных можно просмотреть за N шагов, где N — количество двоичных разрядов в адресе. (Такой поиск называется двоичным поиском в адресном пространстве размера 2^.) Однако даже этот метод можно усовершенствовать, о чем вы можете прочитать в [479], а общее описание алгоритмов классификации пакетов содержится в [191].

Но даже при *N=32* (например, для 32-разрядного IP-адреса) скорость просмотра таблицы методом двоичного поиска недостаточно высока для маршрутизации в современных магистралях. Например, если на каждом шаге работы алгоритма требуется одно обращение к памяти, то при памяти со временем доступа 40 не уровня одного миллиона операций в секунду достичь не удастся. Для увеличения скорости поиска применяются несколько приемов. Один из таких приемов заключается в использовании ассоциативной памяти (Content Addressable Memory, CAM). В маршрутизаторах Cisco 8500 [81] каждый порт оснащен CAM-памятью объемом 64 Кбайт. Другой метод увеличения скорости поиска состоит в том, что недавно полученные записи таблицы продвижения данных хранятся в кэше [145]. В данном методе важен размер кэша. Измерения, результаты которых приведены в [517], показали, что даже для линии OC-3 магистральный маршрутизатор должен обрабатывать в минуту приблизительно 250 ООО пар транзакций отправитель-получатель. В последние годы были предложены еще более быстрые структуры данных, позволяющие находить записи в таблице за log(JV) шагов [545], а также новые методы сжатия таблиц продвижения данных [46]. Аппаратный метод оптимизации поиска в таблице, использующий тот факт, что в адресе, как правило, ищутся 24 или меньше значимых разрядов, обсуждается в [190].

Как только алгоритм поиска определяет выходной порт для пакета, этот пакет может быть передан в коммутационный блок. Однако, какТбудет показано далее, пакет может быть временно *заблокирован* на входе в коммутационный блок (так как коммутационный блок может быть занят другим пакетом). Таким образом, блокированный пакет необходимо поставить в очередь на входном порте, чтобы он мог пройти через коммутационный блок позднее. Более подробно мы рассмотрим вопросы блокировки, обработки очередей и планирования пакетов[1] в маршрутизаторе (как на входных, так и на выходных портах) в подразделе «Очереди» данного раздела.

Принятие решения о том, какой из находящихся в очереди пакетов будет первым отправлен дальше по маршруту следования. — *Примеч. пер.*

Коммутационный блок

Коммутационный блок располагается в самом сердце маршрутизатора. Именно через коммутационный блок пакеты перемещаются от входного порта к выходному порту. Коммутационный блок может быть реализован несколькими способами, как показано на рис. 4.35.

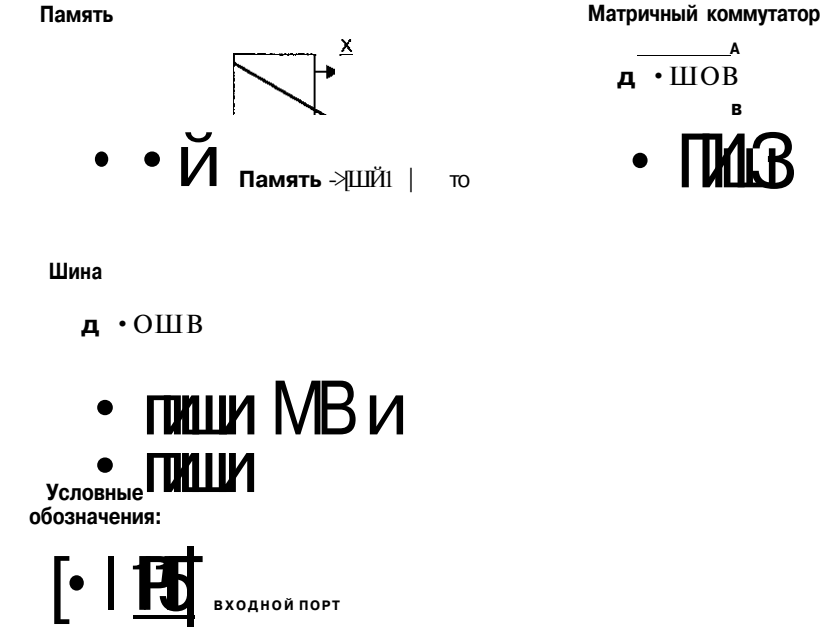


Рис. 4.35. Три метода коммутации

- *Коммутация с использованием памяти.* Простейшие ранние маршрутизаторы часто представляли собой традиционные компьютеры, в которых коммутация между входными и выходными портами осуществлялась под непосредственным управлением центрального процессора (игавшего роль маршрутного процессора). Входные и выходные порты функционировали как традиционные устройства ввода-вывода в операционной системе. Получив пакет, входной порт сначала сигнализирует маршрутному процессору прерыванием. Затем пакет копируется из входного порта в память процессора. После этого маршрутный процессор извлекает из заголовка пакета адрес получателя, ищет соответствующий выходной порт в таблице продвижения данных и копирует пакет в буфер выходного порта. Обратите внимание, если пропускная способность памяти позволяет записать или прочитать *В* пакетов в секунду, тогда суммарная пропускная способность коммутатора (полная скорость, с которой пакеты переносятся от входных портов к выходным портам) не может превышать *В/2.*

Многие современные маршрутизаторы также используют память для коммутации пакетов. Однако их главное отличие от ранних маршрутизаторов заклю-

чается в том, что поиск адреса получателя и хранение (коммутация) пакетов в соответствующей области памяти выполняется процессорами на карте входной линии. Маршрутизаторы, коммутирующие пакеты через память, во многом напоминают мультипроцессоры с коллективным доступом к памяти. К таким маршрутизаторам относятся коммутаторы серии Catalyst 8500 компании Cisco [81] и маршрутизаторы Bay Network Accelar 1200.

- *Коммутация с использованием шины.* При таком подходе входные порты переносят пакеты напрямую в выходные порты по общей шине без вмешательства маршрутного процессора (обратите внимание, что в случае коммутации через память пакет также должен пересечь системную шину, соединяющуюся с памятью). Хотя маршрутный процессор не участвует в переносе пакета по шине, поскольку шина используется коллективно, в каждый момент времени по шине может двигаться только один пакет. Пакет, поступивший на входной порт в тот момент, когда шина занята переносом другого пакета, блокируется и устанавливается в очередь входного порта. Поскольку каждый пакет должен пересечь единственную шину, пропускная способность подобного маршрутизатора ограничивается скоростью шины.

  Учитывая, что пропускная способность современной шины может превышать гигабит в секунду, метод коммутации через шину часто оказывается достаточным для маршрутизаторов, работающих в корпоративных сетях. Основанный на использовании шины метод коммутации принят в ряде современных маршрутизаторов, включая Cisco 1900 [79], коммутирующих пакеты по шине Packet Exchange Bus с пропускной способностью 1 Гбит/с. Система CoreBuilder компании 3Com [258] соединяет порты, располагающиеся в различных коммутирующих модулях при помощи шины PacketChannel с пропускной способностью 2 Гбит/с.

- *Коммутация с использованием соединительной сети.* Один из способов преодолеть ограничения пропускной способности одной общей шины заключается в использовании более сложной соединительной сети, подобной сети, связывающей процессоры в мультипроцессорных системах. Матричный коммутатор представляет собой соединительную сеть, состоящую из 2Л/Г шин, соединяющих *N* входных портов с JV выходными портами, как показано на рис. 4.35. Прибывающий на входной порт пакет перемещается по горизонтальной шине до пересечения с вертикальной шиной, ведущей к нужному выходному порту. Если ведущая к выходному порту вертикальная шина свободна, пакет переносится в выходной порт. Если в данный момент вертикальная шина уже используется для передачи пакета в тот же самый выходной порт из другого входного порта, тогда пакет блокируется и ставится в очередь входного порта.

  В качестве соединительных сетей между входными и выходными портами также предлагалось использовать коммутационные блоки Delta и Omega. Обзор архитектур коммутаторов приводится в [520]. В коммутаторах семейства Cisco 12000 [86] применяется соединительная сеть, обеспечивающая пропускную способность до 60 Гбит/с. одна из современных тенденций в устройстве соединительной сети [268] заключается в том, что IP-дейтаграмма переменной длины фрагментируется на ячейки фиксированной длины, которые затем марки-

руются и коммутируются через соединительную сеть. На выходном порту из этих ячеек восстанавливается исходная дейтаграмма. Ячейки фиксированной длины и внутренняя маркировка существенно упрощают и ускоряют коммутацию пакета через соединительную сеть.

Выходные порты

Схема обработки данных в выходном порту, представленная на рис. 4.36, показывает, что хранящиеся в памяти выходного порта дейтаграммы передаются по выходной линии. Среди функций выходного порта можно выделить функции протоколов канального и физического уровней, взаимодействующих со своими аналогами во входном порту на другом конце линии связи, как уже рассказывалось в подразделе «Входные порты» данного раздела. Также в выходном порту необходимы функции обработки очередей и управления буферами в тех случаях, когда коммутационный блок доставляет пакеты выходному порту со скоростью, превосходящей скорость передачи данных в выходной линии.

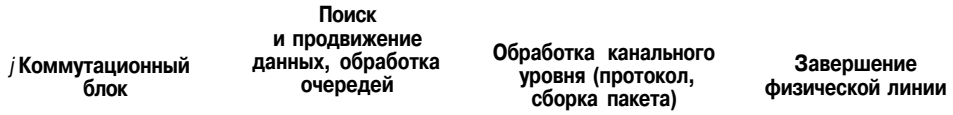


Рис. 4.36. Обработка данных в выходном порту

Очереди

Глядя на конфигурацию, изображенную на рис. 4.35, и учитывая функциональность входного и выходного портов, очевидно, что очереди пакетов могут образовываться как на входных, так и на выходных портах. Необходимо рассмотреть эти очереди несколько подробнее, поскольку при увеличении их размеров буферное пространство маршрутизатора, в конце концов, исчерпывается, и в результате маршрутизатор начинает *терять пакеты*. Ранее уже вскользь упоминалось, что пакеты «теряются в сети» или «отбрасываются маршрутизатором». Это происходит именно в этих очередях. Точное место, в котором теряется пакет (очередь входного порта или очередь выходного порта), зависит от интенсивности трафика, относительной пропускной способности коммутационного блока и скорости передачи данных в линии связи, что будет показано далее.

Предположим, что скорости передачи данных во входной и выходной линиях одинаковы и у маршрутизатора n входных и n выходных портов. Если пропускная способность коммутационного блока, по меньшей мере, в n раз превосходит скорость передачи данных в линии, тогда во входных портах очереди возникнуть не могут. Даже в худшем случае, когда по всем n входных линиям будут поступать пакеты, коммутатор сможет переправлять все n пакетов из входных портов в выходные пор-

ты за время, необходимое для того, чтобы каждый из *n* входных портов одновременно принял по одному пакету. Но что происходит на выходных портах? Будем продолжать предполагать, что пропускная способность коммутационного блока, по меньшей мере, в *n* раз превосходит скорость передачи данных в линии. В худшем случае все пакеты, принятые на *n* входных портах, направляются в один и тот же выходной порт. В этом случае за время приема/передачи одного пакета на этот выходной порт поступят сразу *n* пакетов. Поскольку за этот интервал времени выходной порт может передать только один пакет, из оставшихся *n* - 1 пакетов образуется очередь. За следующий интервал времени на выходной порт могут поступить еще *n* пакетов, которые добавятся к уже имеющейся очереди, и т. д. Наконец, число пакетов в очереди может вырасти настолько, что у маршрутизатора закончится свободная память для их размещения, после чего ему придется отбрасывать некоторые пакеты.

Образование очереди на выходном порту иллюстрирует рис. 4.37. В момент времени *t* на каждый входной порт прибывает по одному пакету. Все пакеты направлены в один и тот же (самый верхний) выходной порт. Предполагая, что скорости передачи данных во всех линиях одинаковы, пропускная способность коммутатора в три раза превосходит скорость передачи данных в линии. Спустя время, требующееся для передачи или приема одного пакета, все три принятых пакета оказываются в выходном порту, где ставятся в очередь на передачу. Еще через такое же время один из этих трех пакетов передается по выходящей линии. В нашем примере в этот момент времени на маршрутизатор поступают два новых пакета. Один из этих пакетов направляется в самый верхний выходной порт.

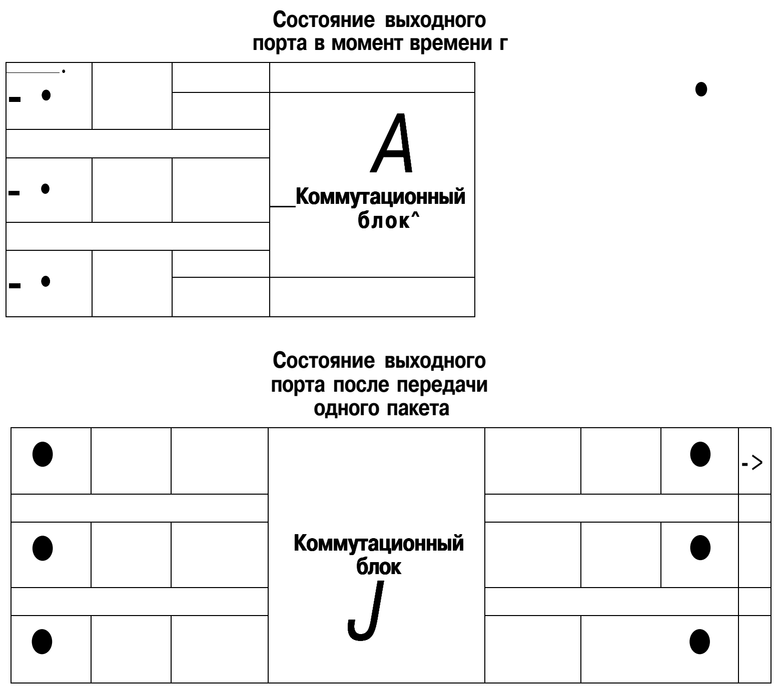


Рис. 4.37. Образование очередей в выходном порту

В результате *планировщик пакетов* выходного порта должен выбрать из стоящих в очереди пакетов один для передачи в линию. Этот выбор может осуществляться на основе простой дисциплины очередей, например, алгоритма FCFS (First Соте, First Served — первым пришел, первым обслужен), или на основе более сложной дисциплины планирования, например, алгоритма WFQ (Weighted Fair Queuing — взвешенная справедливая организация очередей). Планирование пакетов играет ключевую роль в предоставлении *гарантий качества обслуживания*. Этот вопрос мы подробно рассмотрим в главе 6. Обсуждение применяемых в современных маршрутизаторах дисциплин очередей в выходных портах имеется в [85].

Аналогично, если не хватает памяти для буферизации входящего пакета, необходимо принять решение о том, отбросить ли новый пакет (такая политика иногда называется «обрубанием хвостов») или удалить из буфера один или несколько уже стоящих в очереди пакетов, чтобы освободить место для нового пакета. В некоторых случаях может быть выгодно отбросить пакет (или пометить заголовок пакета), прежде чем буфер заполнится, чтобы тем самым послать сигнал отправителю. Было предложено и проанализировано множество алгоритмов отбрасывания и пометки пакетов, которые совместно получили название алгоритмов *активного управления очередями* (Active Queuing Management, AQM) [286]. Одним из наиболее глубоко изученных и распространенных алгоритмов AQM является алгоритм RED (Random Early Detection — случайное раннее обнаружение). В алгоритме RED для длины выходной очереди вычисляется взвешенное среднее значение. Если при поступлении пакета средняя длина очереди оказывается меньше минимальной границы min^, пакет ставится в очередь. Если, напротив, в момент прибытия пакета очередь полна или средняя длина очереди оказывается больше максимальной границы max^, пакет маркируется или отбрасывается. Наконец, если при поступлении пакета средняя длина очереди находится в интервале [тт(Л, тах,Л], пакет маркируется или отбрасывается с вероятностью, представляющей собой функцию средней длины и параметров mi% и max^. Было предложено несколько вероятностных функций и реализовано несколько версий алгоритма RED. Обзоры результатов этих исследований, а также ссылки на дополнительные источники информации приводятся в [74,154].

Если пропускная способность коммутационного блока недостаточно высока (по сравнению со скоростью передачи данных во входных линиях), чтобы переправить все прибывшие пакеты без задержки, тогда очереди могут возникать и на входных портах. Чтобы проиллюстрировать важные последствия образования подобных очередей, рассмотрим коммутационный блок, предположив, что, во-первых, скорости передачи данных на всех линиях одинаковы, во-вторых, что один пакет может быть переправлен из любого входного порта в любой выходной порт за время, необходимое для приема пакета по входной линии, и в-третьих, что пакеты перемещаются из каждой входной очереди в выходную очередь в порядке их поступления во входную очередь (в соответствии с алгоритмом FCFS). До тех пор пока пакеты направляются в разные выходные порты, параллельно могут переправляться несколько пакетов. Однако если два пакета в начале двух входных очередей направляются в одну и ту же выходную очередь, тогда один из пакетов блокируется и вынужден ждать во входной очереди, так как коммутационный блок может переносить блоки в определенный выходной порт только по одному за раз.

На рис. 4.38 показан пример, в котором два пакета (темных), находящиеся в начале своих входных очередей, направляются в один и тот же правый верхний выходной порт. Предположим, что коммутационный блок решает переправить пакет из начала верхней левой очереди. В этом случае темному пакету из левой нижней очереди придется подождать. Но подождать придется также светлому пакету, стоящему в очереди позади пакета в левой нижней очереди, хотя за средний правый выходной порт (пункт назначения светлого пакета) конкуренции нет. Это явление называется «блокированием головы очереди» — стоящий во входной очереди пакет должен ждать, хотя его выходной порт свободен, так как он блокирован другим пакетом в начале очереди. В [259] показано, что при определенных допущениях, как только частота поступления пакетов по входным линиям достигнет 58 % от их пропускной способности, из-за подобного блокирования входная очередь начинает неограниченно расти (это означает, что рано или поздно маршрутизатор начнет терять пакеты). Несколько решений проблемы блокирования головы очереди обсуждаются в [323].

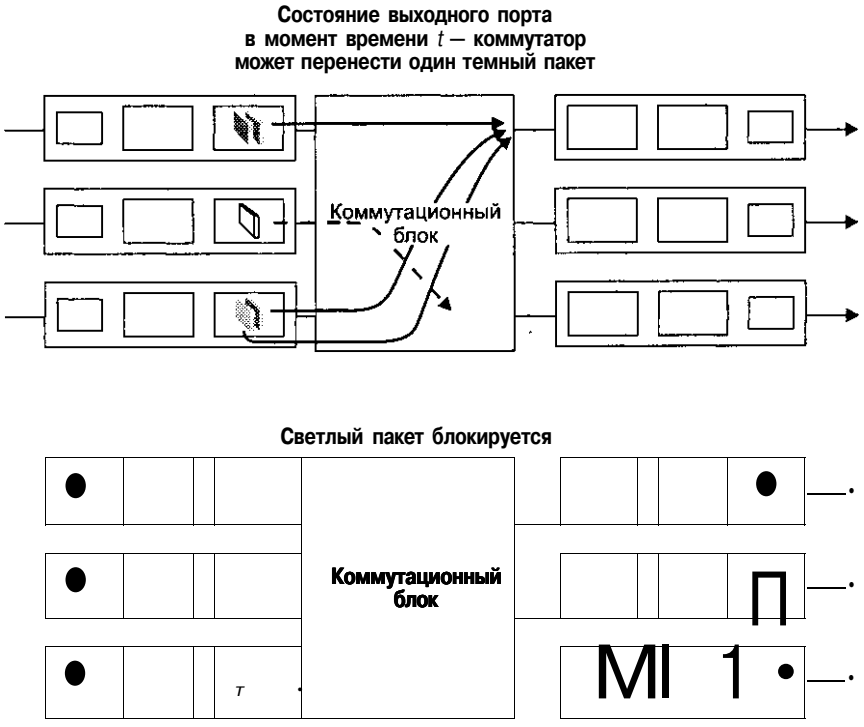


Рис. 4.38. Блокирование головы очереди на входном порту

Протокол IPv6

В начале 90-х годов группа IETF начала работы над очередной версией протокола IP. В первую очередь, необходимость подобных работ была вызвана осознанием того факта, что 32-разрядное адресное пространство протокола IP исчерпывается, а новые сети и IP-узлы присоединяются к Интернету с захватывающей дух скоростью. В ответ на потребность в большем адресном пространстве была разработана версия IPv6 протокола IP. При этом разработчики воспользовались «удобным случаем», чтобы, используя накопленный опыт эксплуатации протокола IPv4, в чем-то его исправить и дополнить.

Вопрос, когда наступит время полного исчерпания адресного пространства протокола IPv4 (и, следовательно, новые сети будет невозможно присоединить к Интернету), дебатировался довольно долго. Основываясь на современных тенденциях в области распределения адресного пространства, два лидера рабочей группы, созданной для оценки этого времени, решили, что свободное адресное пространство закончится в 2008 и 2018 годах [475]. В 1996 году Американская служба регистрации номеров Интернета (American Registry for Internet Numbers, ARIN) сообщила, что ею были выделены все IP-адреса класса A, 62 % адресов класса B и 37 % адресов класса C [15]. Хотя эти оценки и предоставляли протоколу IPv4 некоторый запас времени, на разработку и развертывание так называемого протокола IP «следующего поколения» также требовалось время, поэтому были начаты работы над новым протоколом [41, 42]. Результатом этих усилий явилась спецификация RFC 2460 протокола IP версии 6 (IPv6), который мы рассмотрим ниже. (Часто задают вопрос, что случилось с IPv5. Изначально предполагалось, что протоколом IPv5 станет протокол ST-2, но впоследствии от протокола ST-2 отказались в пользу протокола RSVP, который мы обсудим в главе 6.)
Протоколу IPv6 посвящены такие замечательные источники информации, как [211,220].

Формат дейтаграммы протокола IPv6

Формат дейтаграммы протокола IPv6 показан на рис. 4.39. По новому формату можно судить о наиболее существенных изменениях в протоколе IP.

- *Расширенные возможности адресации.* В дейтаграмме протокола IPv6 размер IP-адреса увеличен с 32 до 128 бит. Это гарантирует, что адресного пространства будет хватать всем и всегда. Теперь можно дать IP-адрес каждой песчинке на планете. В дополнение к индивидуальным и групповым адресам в протоколе IPv6 появился новый тип адресов, называемых *адресами свободной рассылки* (anycast addresses), позволяющих пересылать дейтаграмму любому члену группы хостов. (Это может служить, например, для передачи сообщения HTTP GET ближайшему из нескольких зеркальных сайтов, содержащих данный документ.)
- *Упрощенный 40-разрядный заголовок.* Несколько полей протокола IPv4 были опущены или сделаны необязательными, о чем будет сказано далее. Получившийся в результате 40-разрядный заголовок фиксированной длины обеспечи-

вает ускоренную обработку IP-дейтаграммы. Новый способ кодирования необязательных полей обеспечивает их более гибкую обработку.

Метка потока и приоритет. Определение потока в протоколе IPv6 довольно расплывчато. В RFC 1752 и RFC 2460 утверждается, что поле метки потока позволяет «маркировать пакеты, для которых отправителю требуется специальная обработка, например, обслуживание с отличным от предоставляемого по умолчанию уровнем качества или обслуживание в реальном времени». С одной стороны, обрабатываться в качестве потока могут, например, транслируемые аудио- или видеоданные, трафик высокоприоритетного пользователя (например, платящего за более качественное обслуживание своего трафика). С другой стороны, традиционные приложения, такие как приложения передачи файлов и электронной почты, в качестве потока могут не обрабатываться. Ясно лишь то, что разработчики протокола IPv6 предвидят необходимость в дифференцировании потоков, несмотря на то что точное понятие потока еще не определено. В заголовке IPv6 также есть восьмиразрядное поле класса трафика. Подобно полю TOS (Type Of Service — тип службы) протокола IPv4 это поле может использоваться для предоставления приоритета определенным пакетам потока, а также для предоставления приоритета дейтаграммам определенных приложений (например, ICMP-пакетам) по сравнению с дейтаграммами других приложений (например, пакетам с сетевыми новостями).

Рис. 4.39. Формат дейтаграммы протокола IPv6

Как упоминалось выше, формат IPv6 ^**eftTarpaMM** проще формата IPv4-дейтаграмм (см. рис. 4.23 и 4.39). Ниже перечислены поля IPve-дейтаграммы.

- *Версия.* Это 4-разрядное поле идентифицирует номер версии протокола IP и для протокола IPv6 содержит значение 6. Обратите внимание, если поместить в это поле значение 4, то мы не получим корректную дейтаграмму протокола IPv4 (если бы это было так, жизнь была бы намного проще).
- *Класс трафика.* Это 8-разрядное поле отчасти напоминает поле TOS протокола IPv4.
- *Метка потока.* Как упоминалось выше, это 20-разрядное поле используется для идентификации «потока» дейтаграмм.

- *Длина полезной нагрузки.* Это 16-разрядное поле обрабатывается как целое число без знака и содержит количество байтов в 1Ру6-дейтаграмме, следующих за 40-разрядным заголовком дейтаграммы.
- *Следующий заголовок.* Это поле идентифицирует протокол, которому доставляется содержимое (поле данных) дейтаграммы (например, TCP или UDP). Для этого поля используются те же значения, что и для поля протокола в IPv4-заголовке.
- *Ограничение на число ретрансляционных участков.* Содержимое этого поля уменьшается на единицу на каждом маршрутизаторе, через который проходит дейтаграмма. Когда содержимое этого поля достигает нуля, дейтаграмма уничтожается.
- *Адреса отправителя и получателя.* Различные форматы 128-разрядных IPv6-адресов описаны в RFC 2373.
- *Данные.* Это полезная нагрузка 1Ру6-дейтаграммы. Когда дейтаграмма достигает пункта назначения, полезная нагрузка извлекается из нее и передается протоколу, указанному в поле следующего заголовка.

Итак, мы перечислили поля, включенные в 1Ру6-дейтаграмму. Сравнивая формат 1Ру6-дейтаграммы, показанной на рис. 4.39, с форматом дейтаграммы протокола IPv4 на рис. 4.23, можно заметить, что некоторые поля 1Ру4-дейтаграммы не включены в 1Ру6-дейтаграмму.

- *Фрагментация/повторная сборка.* Протокол IPv6 не позволяет производить на маршрутизаторах фрагментацию и повторную сборку. Если принятая маршрутизатором 1Ру6-дейтаграмма слишком велика для передачи по выходной линии, маршрутизатор просто отбрасывает такую дейтаграмму и посылает обратно отправителю (см. ниже) ICMP-сообщение об ошибке (пакет слишком велик). Отправитель может послать данные еще раз, используя IP-дейтаграммы меньшего размера. Такие операции, как фрагментация и повторная сборка дейтаграмм, требуют много времени. Избавление маршрутизаторов от необходимости выполнять подобные действия существенно ускоряет работу протокола IP.
- *Контрольная сумма заголовка.* Поскольку протоколы транспортного уровня (например, TCP и UDP), а также протоколы канального уровня (например, Ethernet) в архитектуре протоколов Интернета считают контрольную сумму передаваемых данных, разработчики протокола IP посчитали, что реализация той же функции на сетевом уровне будет излишней. Опять же, центральной задачей была быстрая обработка IP-пакетов. Как было показано в подразделе «Адресация в протоколе IPv4» раздела «Интернет-протокол», поскольку IPv4-заголовок содержит поле времени жизни (TTL), аналогичное полю ограничения на число ретрансляционных участков в протоколе IPv6, контрольную сумму IPv4-заголовка приходилось пересчитывать на каждом маршрутизаторе. Как и фрагментация, и повторная сборка, эта операция также отнимает много времени.
- *Параметры.* Поле параметров более не входит в стандартный IP-заголовок. Однако от него не отказались полностью. Вместо этого поле необязательных

параметров может быть одним из «следующих заголовков» (наравне с TCP- или UDP-заголовками), на которые может ссылаться 1Ру6-заголовок. В результате удаления этого поля длина IP-заголовка стала фиксированной (40 байт).

Новый протокол ICMP для протокола IPv6

Как рассказывалось в разделе «Интернет-протокол», протокол ICMP используется IP-узлами для сообщения о произошедших ошибках, а также для предоставления ограниченной информации (например, эхо-ответа на запрос ping) оконечной системе. Для протокола IPv6 была разработана новая версия протокола ICMP, определенная в RFC 2463. В дополнение к реорганизации существующих ICMP-типов и кодов в ICMPv6 также были добавлены новые типы и коды, призванные обеспечить функциональность протокола IPv6. К ним относятся, например, такие сообщения об ошибках, как «пакет слишком велик» и «нераспознаваемые IPv6-параметры». Кроме того, в протоколе ICMP для IPv6 реализована функциональность протокола IGMP, который мы рассмотрим в разделе «Групповая маршрутизация». Протокол IGMP используется для управления присоединением хоста к так называемой группе рассылки и выходом хоста из этой группы.

Переход с IPv4 на IPv6

Теперь, когда мы обсудили технические детали протокола IPv6, рассмотрим довольно практический вопрос: как перевести на IPv6 Интернет, функционирующий по протоколу IPv4? Проблема заключается в том, что, хотя новые 1Ру6-системы можно сделать обратно совместимыми, то есть реализовать в них поддержку дейтаграмм старого формата, уже работающие IPv4-системы не смогут обрабатывать IPve-дейтаграммы. Решений может быть несколько.

Одним из решений могло бы стать объявление о некой дате перехода с IPv4 на IPv6. Последняя подобная глобальная смена технологий (переход с протокола NCP на TCP для предоставления надежной транспортной службы) произошла почти 20 лет назад (RFC 801). Однако уже довольно давно, еще когда Интернет был крошечный и управлялся небольшим количеством «волшебников», стало ясно, что подобный «день икс» невозможен. Сегодня такая операция затронула бы сотни миллионов машин и миллионы сетевых администраторов и пользователей. В RFC 2893 описываются два подхода (которые можно использовать вместе или по отдельности) постепенного ввода IPv6-**хостов** в «мир» IPv4 (с долгосрочной целью, разумеется, полного перехода 1Ру4-узлов на протокол IPv6).

Вероятно, наиболее простой способ внедрить в Интернет узлы, поддерживающие протокол IPv6, представляет собой метод *двойного стека,* при котором IPve-узлы обладают полной поддержкой протокола IPv4. Некоторые узлы, называемые в RFC 28931Ру6/1Ру4-узлами, обладают способностью принимать и посылать как 1Ру4-дейтаграммы, так и IPve-дейтаграммы. Взаимодействуя с 1Ру4-узлом, IPv6/1Ру4-узел может использовать 1Ру4-дейтаграммы; взаимодействуя с IPve-узлом, он может использовать IPve-дейтаграммы. У IPv6/IPv4-узла должен быть как IPv4-адрес, так и IPve-адрес. Кроме того, они должны уметь определять, поддерживает

ли другой узел протокол IPv6 или только IPv4. Эта проблема может быть разрешена с помощью DNS-сервера (см. главу 2), возвращающего IPve-адрес узла, если данный узел поддерживает протокол IPv6, или 1Py4-адрес узла в противном случае. Разумеется, если узел, выдающий DNS-запрос, сам поддерживает только протокол IPv4, DNS-сервер возвращает только 1Py4-адрес.

При таком подходе, если либо отправитель, либо получатель поддерживает только протокол IPv4, должна быть использована 1Py4-дейтаграмма. В результате возможна ситуация, когда два узла, поддерживающие протокол IPv6, обмениваются дейтаграммами в формате IPv4. Эту ситуацию иллюстрирует рис. 4.40. Предположим, что узел A, поддерживающий протокол IPv6, хочет переслать IP-дейтаграмму узлу F, который также обеспечивает поддержку протокола IPv6. То есть узлы A и B могут обмениваться дейтаграммами формата IPv6, но чтобы переслать дейтаграмму узлу C, узел B должен создать 1Py4-дейтаграмму. Разумеется, поле данных IPve-дейтаграммы копируется в поле данных 1Py4-дейтаграммы, также выполняется соответствующее преобразование адресов. Однако некоторые поля протокола IPv6 (например, поле идентификатора потока) не имеют соответствий в 1Py4-дейтаграммах. В результате при подобном преобразовании хранящаяся в этих полях информация теряется. Таким образом, хотя узлы E и F могут обмениваться дейтаграммами формата IPv6, переданная узлу E узлом D IPv4-дейтаграмма не содержит всех полей, которые были в оригинальной дейтаграмме, полученной от узла A.

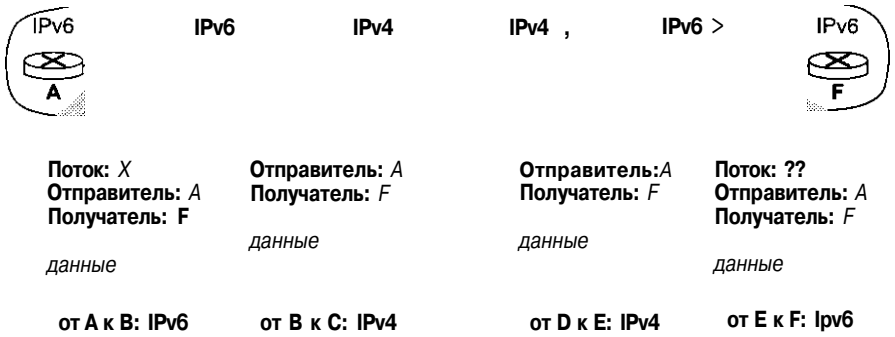


Рис. 4.40. Метод двойного стека

Альтернативный метод, также обсуждаемый в RFC 2893, называется *туннелированием.* Путем туннелирования можно решить упомянутую выше проблему, то есть узел E сможет принять отправленную узлом A IPve-дейтаграмму в исходном виде. В основе туннелирования лежит следующая идея. Предположим, два IPve-узла (например, узлы B и E на рис. 4.40) хотят обменяться IPve-дейтаграммами, но их разделяют 1Py4-маршрутизаторы. Промежуточные 1Py4-маршрутизаторы мы будем называть *туннелем,* как показано на рис. 4.41. Метод туннелирования заключается в том, что передающий IPve-узел (например, узел B) помещает IPve-дейтаграмму целиком в поле данных дейтаграммы формата IPv4. Затем эта IPv4-дейтаграмма, адресованная получающему IPve-узлу на другой стороне туннеля (например,

узлу E), передается первому узлу туннеля (например, узлу C). Промежуточные IPv4-маршрутизаторы передают эту дейтаграмму друг другу, как обычную дейтаграмму, не зная о том, что она содержит полную дейтаграмму формата IPv6. Наконец, принимающий IPv6-узел на другой стороне туннеля получает IPv4-дейтаграмму, определяет, что она содержит дейтаграмму формата IPv6, извлекает IPv6-дейтаграмму и переправляет ее дальше, как если бы он получил ее по сети от непосредственного соседа.

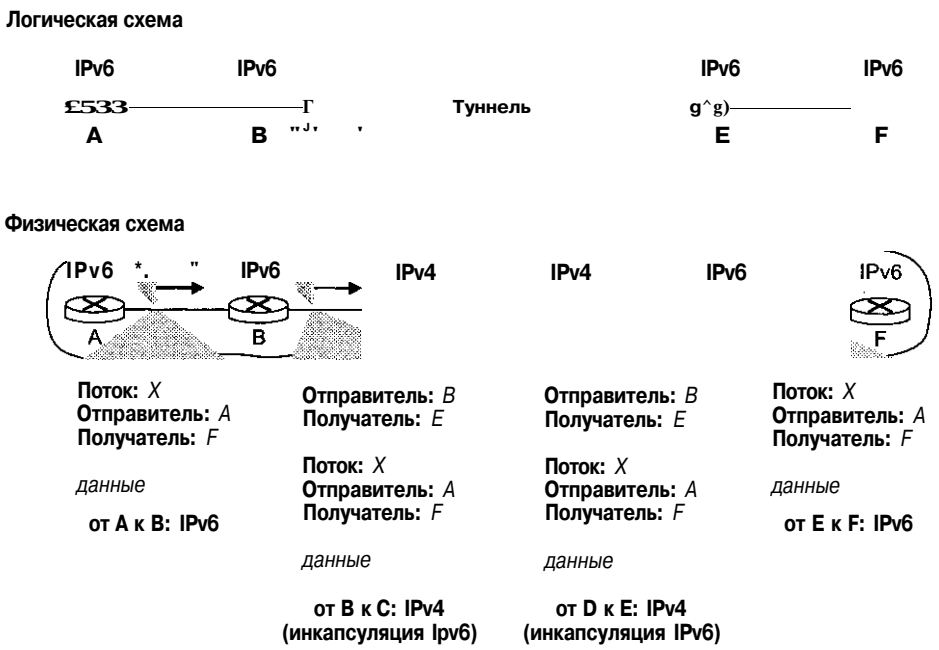


Рис. 4.41. Туннелирование

В завершение этого раздела отметим, что процесс перехода на протокол IPv6 пока 4to продвигается не слишком быстрыми темпами [294]. Вспомним, что основным мотивом разработки протокола IPv6 было истощение адресного пространства протокола IPv4. Однако, как отмечалось в разделе «Интернет-протокол», протоколу IPv4 удалось значительно «продлить жизнь» с помощью таких методов, как бесклассовая внутридоменная маршрутизация (позволяет использовать сетевую часть адреса любой длины), динамическое выделение адреса протоколом DHCP, трансляция сетевых адресов (позволяет к 32 бит IP-адресов добавить 16 бит номера порта). Однако возможно, что широкое распространение мобильных телефонов и других портативных устройств может привести к тому, что адресного пространства протокола IPv4 снова станет недостаточно. Компания Ericsson планирует выпуск телефонов с поддержкой протокола IPv6 в 2002 г., а Европейская программа партнерства третьего поколения (Third Generation Partnership Program, 3GPP) специфицировала протокол IPv6 [516] как стандартную схему адресации для мобильных мультимедийных устройств. Хотя за пер-

вые семь лет своего существования протокол IPv6 не получил широкого распространения, без всякого сомнения, у него большие перспективы. Подобно тому как потребовалось несколько десятков лет, чтобы сформировалась современная система телефонных номеров (она используется уже почти полвека и, похоже, уходить не собирается), для широкого распространения протокола IPv6 может потребоваться некоторое время, но затем он будет применяться в течение долгого срока. Брайан Карпентер, бывший председатель совета по архитектуре Интернета (Internet Architecture Board, IAB) [234], а также автор нескольких документов RFC, относящихся к протоколу IPv6, говорит: «Я всегда рассматривал это как 15-летний процесс, начавшийся в 1995 году» [294]. Если верить Крапентеру, мы еще лишь на полпути!

Один важный урок, который мы должны усвоить из знакомства с протоколом IPv6, состоит в том, что сменить сетевой протокол крайне сложно. С начала 90-х годов множество новых сетевых протоколов провозглашались следующим революционным прорывом в Интернете, но большая часть их просуществовала очень недолго. К этим протоколам относятся IPv6, протоколы групповой рассылки (см. раздел «Групповая маршрутизация»), а также протоколы резервирования ресурсов (см. главу 6). В самом деле, введение нового протокола в сетевой уровень подобно замене фундамента у здания — это сложно сделать, не развалив весь дом или, по меньшей мере, не переселив временно его жильцов. С другой стороны, Интернет был свидетелем быстрого развертывания новых протоколов прикладного уровня. Классическими примерами являются web, передача сообщений в реальном времени, коллективное использование файлов одноранговыми (равноправными) узлами. К другим примерам относятся передача потокового аудио и видео, чат. Внедрение новых протоколов прикладного уровня подобно перекраске здания — это относительно легко сделать, и, если вы выберете привлекательный цвет, соседи быстро сделают то же самое. Итак, в будущем мы можем ожидать изменений на сетевом уровне стека протоколов Интернета, но эти изменения, скорее всего, будут происходить значительно медленнее изменений на прикладном уровне.

Групповая маршрутизация

Рассмотренные нами протоколы транспортного и сетевого уровней обеспечивают доставку пакетов от одного отправителя одному получателю, поэтому такие протоколы часто называют *протоколами выборочной рассылки* (unicast protocols).

Для ряда новых сетевых приложений требуется доставка пакетов от одного или нескольких отправителей *группе получателей.* Сюда относятся приложения для переноса больших объемов данных (например, рассылка разработчиком программного обеспечения пакета обновлений своим пользователям), приложения для передачи потокового аудио или видео, приложения, использующие распределенные данные (например, доски объявлений или телеконференции), приложения для периодической рассылки новых данных с биржи, приложения для

обновления web-кэша, интерактивные сетевые игры. Для каждого из таких приложений было бы крайне полезно использовать *групповую рассылку* (multicasting): передачу пакета от одного отправителя нескольким получателям за одну операцию.

В этом разделе мы рассмотрим аспекты сетевого уровня групповой рассылки. Мы увидим, что, как и в случае выборочной рассылки, на сетевом уровне алгоритмы маршрутизации играют центральную роль. Однако мы также увидим, что в отличие от выборочной рассылки групповая рассылка в Интернете относится к службам, требующим установки соединения — маршрутизаторы, управляющие групповыми пакетами, должны обменяться информацией о состоянии группового соединения. Для этого требуется комбинация сигнальных протоколов и протоколов маршрутизации, обеспечивающих установку, поддержание и разрыв соединения на маршрутизаторах.

Групповая рассылка в Интернете и группы рассылки

С точки зрения сети групповая рассылка представляет собой одну операцию передачи, в результате которой копии переданных данных доставляются группе получателей. Групповая рассылка может быть реализована несколькими способами.

- *Выборочная рассылка от одного отправителя всем получателям группы.* Отправитель устанавливает обычные одноадресные транспортные соединения с каждым из получателей. Передаваемая транспортному уровню отправителя единица обмена прикладного уровня дублируется самим отправителем и передается через каждое соединение. При таком подходе для групповой рассылки нижележащий сетевой уровень используется обычным путем (так же, как для выборочной рассылки) , и явная поддержка на нем групповой рассылки не требуется [498]. Данный подход иллюстрирует рис. 4.42, *а,* на котором сетевые маршрутизаторы, не принимающие участия в рассылке, показаны светлыми. Чтобы доставить данные трем получателям, отправитель использует три *отдельных* одноадресных соединения.
- *Групповая рассылка на прикладном уровне.* Во втором методе также используется выборочная рассылка, но в дублирование и продвижение данных вовлекаются получатели. В отличие от предыдущего случая, когда отправитель сам пересылал копии данных всем получателям, в данном случае отправитель рассылает копии только нескольким (или одному) из них, а те затем сами создают копии и переправляют их другим получателям. Последние также могут создать копии и разослать их дополнительным получателям и т. д. Для реализации данной схемы необходимо создать и поддерживать инфраструктуру распределения на прикладном уровне [75,374]. Как показано на рис. 4.42, б, единственная дейтаграмма методом выборочной рассылки посылается отправителем получателю, который делает две копии и посылает их остальным двум получателям тем же методом.

- *Явная групповая рассылка.* Третья возможность заключается в предоставлении явной поддержки групповой рассылки на сетевом уровне. При таком подходе передающий хост отправляет всего *одну* дейтаграмму. Эта дейтаграмма (или ее копия) дублируется сетевым *маршрутизатором,* и копии отправляются по нужным исходящим линиям. Данный подход иллюстрирует рис. 4.42, в, на котором маршрутизаторы, поддерживающие групповую рассылку, показаны темными. Здесь отправитель передает всего одну дейтаграмму, которая затем дублируется маршрутизатором. Одну из копий маршрутизатор посылает самому верхнему получателю, а вторая направляется правому маршрутизатору, который посылает ее по локальной сети Ethernet с широковещательным адресом, в результате эту копию получают оба получателя.

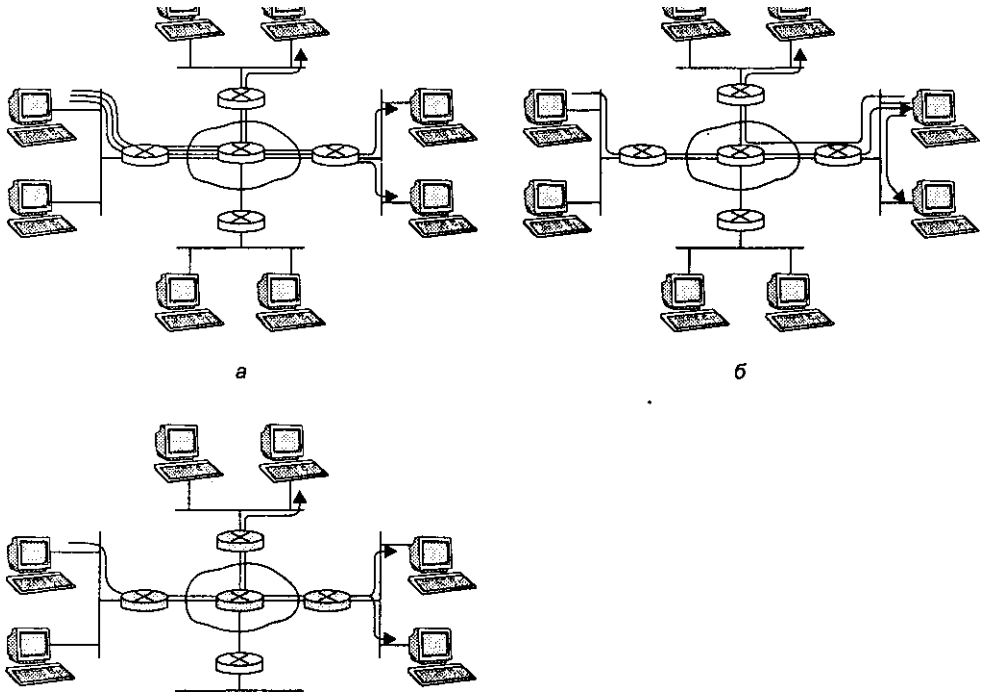


Рис. 4.42. Три способа групповой рассылки

Очевидно, что в случае явной групповой рассылки пропускная способность сети используется наиболее эффективно, так как по каждой линии пересылается всего одна копия дейтаграммы. С другой стороны, для реализации данного метода требуется существенная поддержка со стороны сетевого уровня. Аналогично, реализация групповой рассылки на прикладном уровне может быть более эффектов-

ной, чем первый вариант, при котором отправитель сам пересылал копии данных всем получателем путем выборочной рассылки. Но и для этого метода также необходима установка и поддержка специальной инфраструктуры на прикладном уровне. В оставшейся части этого раздела мы рассмотрим механизмы поддержки групповой рассылки на сетевом уровне, так как такая поддержка требует решения ряда интересных проблем, некоторые из которых характерны и для групповой рассылки на прикладном уровне.

При групповой рассылке мы сразу же сталкиваемся с двумя проблемами: как идентифицировать получателей многоадресной дейтаграммы и как адресовать эту дейтаграмму.

В случае выборочной рассылки IP-адрес получателя содержится в каждой одноадресной IP-дейтаграмме и означает единственного получателя. Но в случае групповой рассылки имеется несколько получателей. Есть ли смысл помещать в каждую групповую дейтаграмму IP-адреса всех получателей, входящих в группу? Хотя такой подход может работать для небольшого количества получателей, он плохо подходит для случая сотен и тысяч получателей. Объем адресной информации просто займет все место в дейтаграмме, вытеснив из нее всю полезную информацию. Кроме того, чтобы явно указать всех получателей, отправитель должен знать идентификаторы и адреса всех получателей. Далее будет показано, что в некоторых ситуациях такое требование может быть нежелательным.

По вышеуказанным причинам в архитектуре Интернета (а также в архитектуре ATM-сети) групповая дейтаграмма адресуется *косвенно.* То есть для группы получателей используется один идентификатор, а копия дейтаграммы, адресованной группе получателей при помощи этого единственного идентификатора, доставляется всем членам этой группы. В Интернете единый идентификатор, соответствующий группе получателей, представляет собой групповой адрес класса D (см. раздел «Интернет-протокол»). Группа получателей, ассоциированная с адресом класса D, называется *группой рассылки.* На рис. 4.43 четыре хоста (показаны темными) ассоциированы с группой рассылки с IP-адресом 226.17.30.197. Эти хосты будут получать все дейтаграммы, направляемые по данному адресу. Сложность заключается в том, что у каждого хоста имеется уникальный IP-адрес, полностью независимый от адреса группы рассылки, членом которой он является.

Хотя идея группы рассылки проста, она поднимает ряд вопросов. Как создается группа и как она прекращает свое существование? Каким образом выбирается групповой адрес? Как к группе добавляются новые хосты (в качестве отправителей или получателей)? Может ли кто угодно присоединиться к группе (и передавать или принимать данные, посылаемые этой группе) или членство в группе ограничивается и, если да, то кем? Известны ли членам группы идентификаторы других членов группы? Как сетевые маршрутизаторы взаимодействуют друг с другом, чтобы доставить групповую дейтаграмму всем членам группы? В Интернете на все эти вопросы отвечает протокол IGMP (RFC 2236). Мы рассмотрим протокол IGMP, а затем вернемся к вопросам.

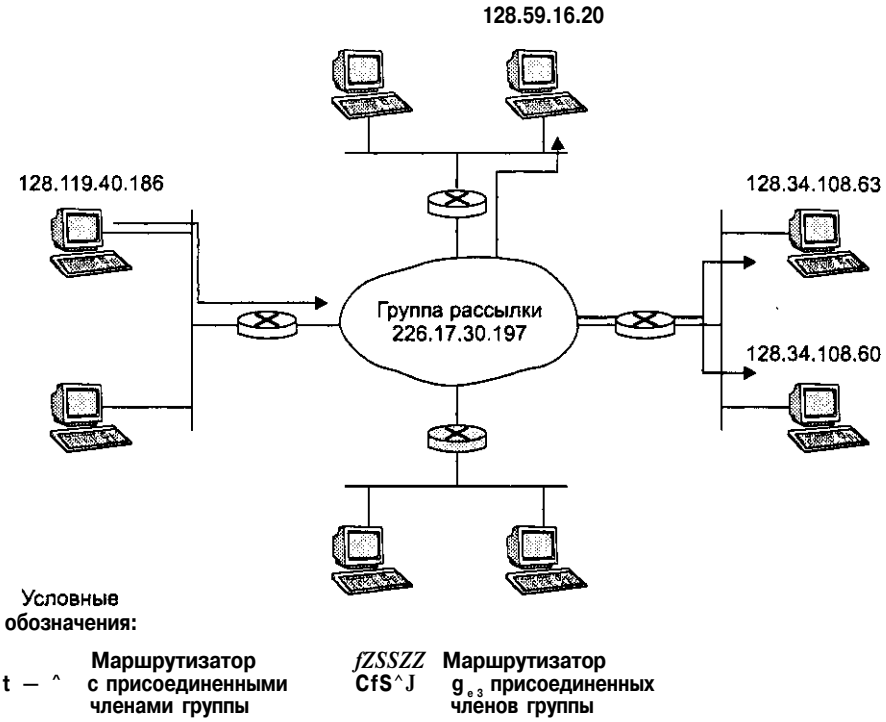


Рис. 4.43. Адресованная в группу рассылки дейтаграмма доставляется всем членам группы

Протокол IGMP

Протокол IGMP (Internet Group Management Protocol — межсетевой протокол управления группами) версии 2, определенный в RFC 2236, работает между хостом и соединенным с ним напрямую маршрутизатором (этот маршрутизатор можно рассматривать как первый маршрутизатор на пути следования входящих дейтаграмм или последний маршрутизатор на пути следования исходящих дейтаграмм). На рис. 4.44 изображены три групповых маршрутизатора, каждый из которых соединен с парой хостов через локальный интерфейс. В данном примере локальный интерфейс связан с локальной сетью, и, как правило, несколько хостов локальной сети являются членами той или иной группы рассылки.

Протокол IGMP предоставляет хосту средство информирования соединенного с ним маршрутизатора о том, что работающее на хосте приложение желает присоединиться к определенной группе рассылки. Учитывая ограниченность сферы действия протокола IGMP хостом и соединенным с ним маршрутизатором, для координирования групповых маршрутизаторов (включая присоединенные маршрутизаторы) в Интернете, очевидно, необходимы другие протоколы. Задачу координирования групповых маршрутизаторов решают *протоколы групповой маршрутизации сетевого уровня,* такие как PIM, DVMRP и MOSPF, которые мы рассмотрим в подразделах «Общий случай групповой маршрутизации» и «Труп-

повая маршрутизация в Интернете» данного раздела. Таким образом, групповая рассылка на сетевом уровне в Интернете состоит из двух взаимодополняющих компонентов: протокола IGMP и протоколов групповой маршрутизации.

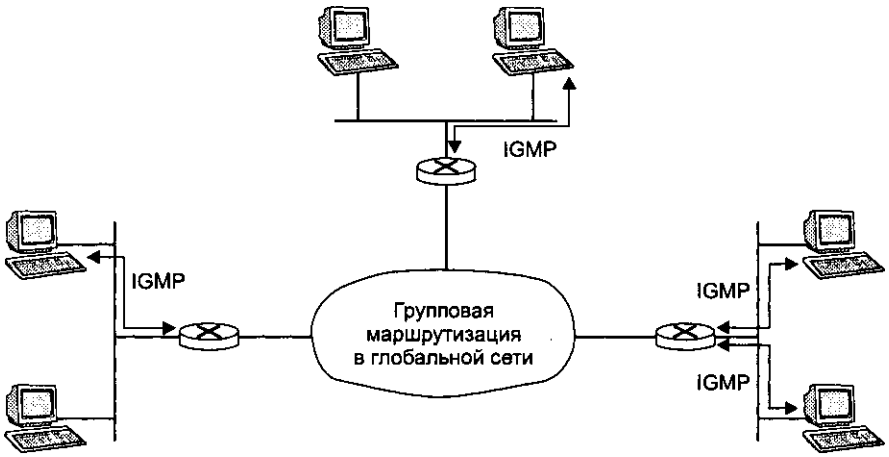


Рис. 4.44. Две составляющие групповой рассылки сетевого уровня: протокол IGMP и протоколы групповой маршрутизации

Хотя протокол IGMP называют «протоколом членства в группах», этот термин может ввести в заблуждение, так как протокол IGMP действует локально, между хостом и соединенным с ним маршрутизатором. Несмотря на свое название, протокол IGMP не работает на всех хостах, входящих в группу рассылки. В действительности, протокола сетевого уровня, управляющего членством в группах рассылки и функционирующего на всех Интернет-хостах группы, *не существует*. Например, не существует протокола сетевого уровня, позволяющего хосту определить идентификаторы всех остальных хостов, присоединившихся к группе.

В протоколе IGMP версии 2 (RFC 2236) используются только три типа сообщений, приведенные в табл. 4.9. Общее сообщение *membership_query* (запрос о членстве) посылается маршрутизатором всем хостам, присоединенным к его интерфейсу (например, всем хостам локальной сети), чтобы узнать обо всех группах рассылки, членами которых стали хосты данного интерфейса. С помощью специального сообщения *membership_query* маршрутизатор может также определить, вступил ли какой-либо хост, присоединенный к одному из его интерфейсов, в определенную группу рассылки. Этот специальный запрос включает адрес группы, помещаемый в специально отведенное для него поле (см. далее).

Хосты отвечают на сообщение *membership_query* IGMP-сообщением *membership jreport,* как показано на рис. 4.45. Хост может также генерировать сообщения *membership jreport,* не ожидая сообщения *membershipjquery* от маршрутизатора, когда приложение впервые присоединяется к группе рассылки. Сообщения *membership jreport* получают маршрутизаторы, а также все хосты, присоединенные к тому же интерфейсу маршрутизатора (например, в случае локальной сети). Каждое сообщение

membership jreport содержит групповой адрес той группы, в которую вступил отвечающий хост. Обратите внимание, что маршрутизатору все равно, *который* из хостов присоединился к данной группе рассылки или даже *сколько* хостов из данной локальной сети присоединилось к определенной группе. (В любом случае маршрутизатор выполняет ту же самую работу — он должен поддерживать протокол групповой маршрутизации вместе с другими маршрутизаторами, обеспечивая получение многоадресных дейтаграмм соответствующими группами рассылки.) Поскольку маршрутизатор беспокоится лишь о том, принадлежит ли любой из присоединенных к нему хостов к той или иной группе рассылки, в идеальном случае хотелось бы получать не более чем по одному уведомлению о принадлежности к группе (зачем тратить время на обработку идентичных сообщений от множества хостов?). Для этого в протоколе IGMP есть специальный механизм, предназначенный для снижения количества сообщений *membershipjreport* в том случае, когда несколько присоединенных хостов относятся к одной группе рассылки.

Таблица 4.9. Типы сообщений протокола IGMP v2

Тип сообщения	Отправитель	Назначение
membership_query (общий запрос)	Маршрутизатор	Запрос о группах рассылки, в которые входят присоединенные хосты
membership_query (конкретный запрос)	Маршрутизатор	Запрос о том, есть ли среди присоединенных хостов члены указанной группы
membership_report	Хост	Уведомление хоста, желающего присоединиться к определенной группе рассылки
leave_group	Хост	Уведомление хоста, желающего покинуть определенную группу рассылки

Рис. 4.45. Запрос и ответ протокола IGMP о членстве в группе

В частности, каждое посланное маршрутизатором сообщение *membershipjquery* также содержит поле максимального времени отклика (рис. 4.46). Получив сообщение *membership jquery,* хост выжидает в течение случайного периода времени в диапазоне от нуля до максимального времени отклика, прежде чем ответить сообщением *membership j-eport.* Если за время ожидания хост заметит, что сообщение

membership jreport послал какой-либо другой хост, входящий в данную группу рассылки, он воздерживается от передачи, так как понимает, что маршрутизатор теперь знает о существовании среди его хостов членов данной группы. Такая форма *подавления отклика* представляет собой разновидность оптимизации производительности — она позволяет хостам избежать передачи излишних сообщений *membershipj-eport.* Аналогичные механизмы подавления отклика используются в ряде протоколов Интернета, включая надежные транспортные протоколы групповой рассылки [160].

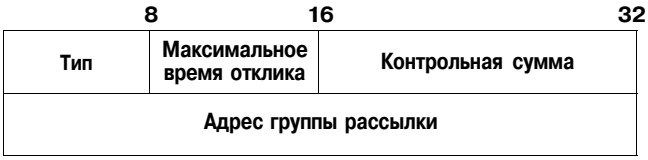


Рис. 4.46. Формат IGMP-сообщения

Четвертый, и последний тип IGMP-сообщений — это сообщение *leave_group.* Интересно отметить, что это сообщение не является обязательным! Но как тогда маршрутизатор определяет, что в данной локальной сети не осталось хостов, входящих в определенную группу? Ответ на этот вопрос дает IGMP-сообщение *membershipjquery.* Маршрутизатор приходит к выводу, что в данную группу рассылки более не входят присоединенные к нему хосты, если ни один хост не отвечает на его сообщение *membership jquery* с конкретным групповым адресом. Интернет-протоколы, в которых по истечении некоторого интервала времени информация об адресах удаляется, иногда называют протоколами с *неустойчивым состоянием* (soft state). Таким протоколом является протокол IGMP, в котором информация о наличии членов определенной группы рассылки среди хостов локальной сети удаляется по истечении заданного интервала времени (в данном случае этот интервал задает периодически посылаемое маршрутизатором сообщение *membership jquery),* если оно не обновляется явно (при помощи посылаемого хостом сообщения *membershipj-eport).* Утверждается, что протоколами с неустойчивым состоянием проще управлять, нежели протоколами с устойчивым состоянием. Для последних нужны не только механизмы явного добавления или удаления состояний (то есть информации о членстве в группах), но также какие-то средства восстановления в ситуации, когда один из этих механизмов преждевременно завершит работу или вообще выйдет из строя [463]. Прекрасное обсуждение вопросов неустойчивого состояния имеется в [406].

Подобно ICMP-сообщениям, сообщения протокола IGMP (см. рис. 4.46) переносятся (инкапсулируются) в IP-дейтаграммах.

Познакомившись с протоколом, предназначенным для присоединения к группам рассылки и для выхода из них, мы лучше можем представить себе применяемую сегодня в Интернете модель обслуживания, основанную на групповой рассылке [112,114]. В данной модели любой хост может присоединиться к группе рассылки на сетевом уровне. Хост просто посылает соединенному с ним маршрутизатору IGMP-сообщение *membership jreport.* Этот маршрутизатор, работающий в Интер-

нете совместно с другими маршрутизаторами, вскоре начинает доставлять групповые дейтаграммы пославшему уведомление хосту. Таким образом, присоединением к группе управляет получатель. Отправитель добавлением получателей к группе рассылки не занимается. Он даже не может контролировать состав группы получателей. Аналогично, получатель не может контролировать тех, кто посылает дейтаграммы в группу рассылки. Очередность прибытия посылаемых разными хостами дейтаграмм у разных получателей может быть разной. Отправитель-злоумышленник может легко вставлять свои дейтаграммы в поток дейтаграмм групповой рассылки. Даже законопослушные отправители могут случайно выбрать два одинаковых адреса при создании групп рассылки, поскольку координация на сетевом уровне отсутствует. С точки зрения приложения, использующего групповую рассылку, это приведет к тому, что оно «вперемешку» будет принимать дейтаграммы сразу из двух потоков вместо одного.

Перечисленные проблемы могут показаться непреодолимым препятствием для разработки прикладных программ, использующих групповую рассылку. Однако все не так плохо. Несмотря на то что сетевой уровень не обеспечивает фильтрации, упорядочивания или конфиденциальности групповых дейтаграмм, все эти механизмы могут быть реализованы на прикладном уровне. Кроме того, в настоящее время ведутся работы по включению некоторых из этих функций в сетевой уровень [51]. Применяемая сегодня в Интернете модель обслуживания, основанная на групповой рассылке, во многом отражает ту же философию, что и модель обслуживания выборочной рассылки — предельно простой сетевой уровень с дополнительными службами, предоставляемыми протоколами более высокого уровня на хостах. Такая философия в случае выборочной рассылки была, несомненно, успешной. Вопрос о том, может ли подобный подход с максимально упрощенным сетевым уровнем быть таким же успешным для службы групповой рассылки, до сих пор остается открытым. Альтернативная модель обслуживания, основанная на групповой рассылке, представлена в [215]. Интересное обсуждение службы групповой рассылки сегодняшнего Интернета и вопросов ее развертывания содержится в [126].

Общий случай групповой маршрутизации

В предыдущем подразделе мы познакомились с тем, как работает протокол IGMP на периферии сети, между маршрутизатором и соединенным с ним хостом, позволяя маршрутизатору определить, какой групповой трафик он должен получать для своих хостов. Теперь мы можем перейти к рассмотрению самих групповых маршрутизаторов: как они должны выбирать маршруты для пакетов, пересылаемых друг другу, чтобы гарантировать, что каждый маршрутизатор получит предназначенный ему групповой трафик?

Рисунок 4.47 призван проиллюстрировать *проблему групповой маршрутизации*. Рассмотрим группу рассылки в предположении, что любой маршрутизатор, у которого есть присоединившийся к группе хост, может как посылать, так и принимать трафик, адресованный этой группе. Хосты, являющиеся членами группы рассылки, показаны на рисунке темными. Непосредственно соединенные с ними

маршрутизаторы также показаны темными. Как видно из рисунка, принимать групповой трафик нужно не всем маршрутизаторам, а только тем, чьи хосты являются членами группы рассылки, то есть маршрутизаторам A, B, E и F. Маршрутизаторам C и D не нужен групповой трафик, так как присоединенные к маршрутизатору D хосты не являются членами группы рассылки, а у маршрутизатора C вообще нет непосредственно присоединенных хостов.

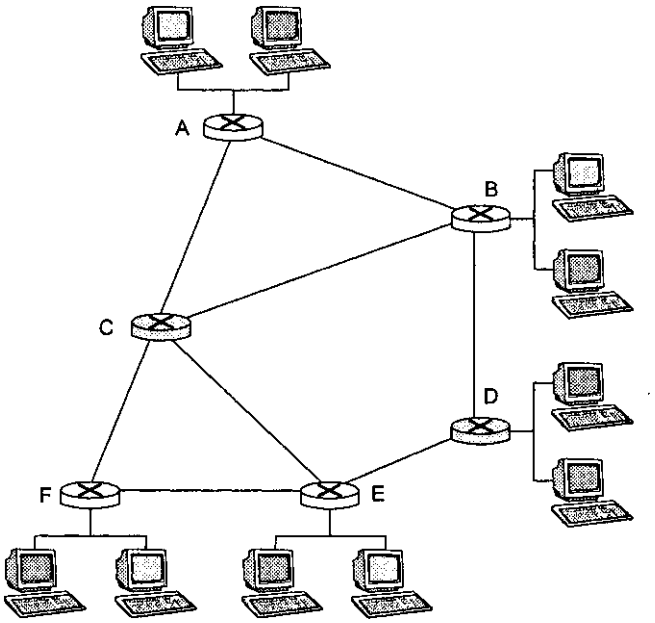


Рис. 4.47. Иллюстрация проблемы групповой маршрутизации

Цель групповой маршрутизации заключается в том, чтобы построить дерево, связывающее все маршрутизаторы, присоединенные хосты которых относятся к данной группе рассылки. При этом групповые пакеты будут направляться по этому дереву от отправителя ко всем хостам, входящим в дерево группы рассылки. Разумеется, дерево может содержать маршрутизаторы, не имеющие хостов, относящихся к данной группе рассылки (например, как видно из рисунка, невозможно связать маршрутизаторы A, B, E и F в дерево, не включив в него маршрутизатор C или D).

На практике для построения дерева групповой маршрутизации применяются два подхода, отличающиеся тем, используется ли общее дерево для нескольких отправителей или для каждого отправителя создается специальное дерево.

Групповая маршрутизация с общим деревом

Рассмотрим сначала случай, в котором все посланные в группу рассылки пакеты направляются по одному и тому же общему дереву группы независимо от отправителя. В этом случае проблема групповой маршрутизации кажется довольно про-

стой: нужно построить дерево, связывающее все маршрутизаторы сети, присоединенные хосты которых являются членами данной группы рассылки. На рис. 4.48 (слева) одно из возможных деревьев группы показано жирными линиями. Обратите внимание, что в это дерево входят маршрутизаторы, присоединенные хосты которых являются членами данной группы рассылки (то есть маршрутизаторы A, B, E и F), а также маршрутизаторы, у которых нет хостов, принадлежащих к данной группе рассылки. В идеальном случае можно также задаться дополнительной целью минимизации стоимости дерева. Если каждой линии сети назначена определенная стоимость, как это делалось в случае одноадресной маршрутизации (см. раздел «Основы маршрутизации»), тогда оптимальным деревом для групповой маршрутизации будет дерево с минимальной суммой стоимостей входящих в него линий. На рис. 4.49 оптимальное дерево групповой рассылки (с суммарной стоимостью линий, равной 7) изображено жирными линцями.

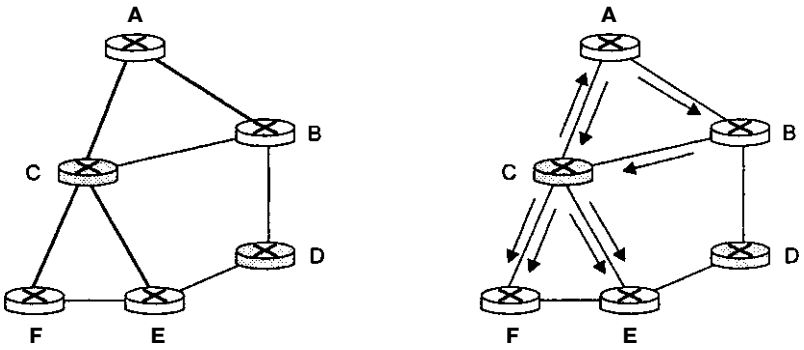


Рис. 4.48. Единое общее дерево (слева) и два отдельных дерева для двух отправителей (справа)

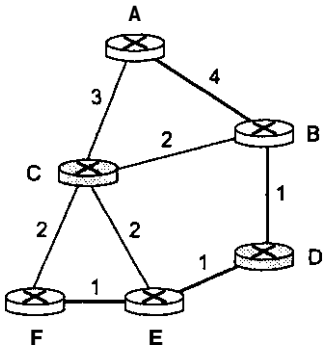


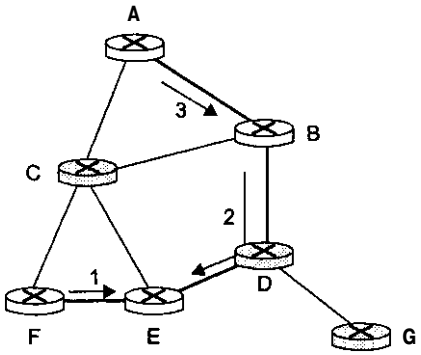
Рис. 4.49. Дерево наименьшей стоимости для группы

Задача нахождения дерева наименьшей стоимости называется задачей дерева Штайнера [194]. Доказано, что решение этой проблемы является NP-полным [175], но приведенный в [279] приближенный алгоритм позволяет найти решение, отличающееся от оптимального на постоянный множитель. Другие исследования пока-

зали, что в общем случае приближенные алгоритмы нахождения дерева Штайнера хорошо зарекомендовали себя на практике [547,548, 551].

Несмотря на то что для решения задачи Штайнера существуют хорошие эвристические приближенные методы, интересно отметить, что ни один из применяемых в Интернете алгоритмов групповой маршрутизации не основан на данных методах. Почему? Одна из причин заключается в том, что для нахождения дерева наименьшей стоимости при каждом изменении стоимости линии алгоритм необходимо запускать заново. Кроме того, важную роль в принятии решения о пригодности алгоритма групповой маршрутизации играют другие соображения, такие как возможность использования информации о маршрутах и таблиц продвижения данных, уже рассчитанных для одноадресной маршрутизации. В конце концов, производительность (и оптимальность) — это не все, что нужно учитывать.

Альтернативный подход определения общего многоадресного дерева, используемый в Интернете в нескольких алгоритмах групповой маршрутизации, основан на определении центрального узла (также называемого *точкой встречи,* или *ядром)* общего дерева групповой маршрутизации. Сначала для группы рассылки определяется центральный узел. Затем маршрутизаторы, хосты которых принадлежат к группе рассылки, посылают (путем обычной выборочной рассылки) центральному узлу сообщения с запросом о намерении присоединиться к дереву. Эти сообщения принимаются либо центральным узлом дерева, либо маршрутизатором, уже принадлежащим дереву. В любом случае путь, который прошло данное сообщение, определяет ветвь дерева маршрутизации между оконечным маршрутизатором, пославшим сообщение, и центральным узлом. Этот путь можно рассматривать как новую ветвь дерева.



Условные обозначения:

Путь и порядок, в котором генерируются сообщения

Рис. 4.50. Формирование дерева с центральным узлом

Создание дерева групповой маршрутизации с центральным узлом иллюстрирует рис. 4.50. Пусть маршрутизатор E выбран в качестве центрального узла дерева.

Сначала к группе рассылки присоединяется узел F, посылая маршрутизатору E соответствующее сообщение. Линия EF становится начальным деревом группы. Затем к дереву присоединяется узел B, посылая соответствующее сообщение маршрутизатору E. Предположим, групповой маршрут от маршрутизатора B до маршрутизатора E проходит через узел D. В этом случае сообщение маршрутизатора B проходит по маршруту BDE, в результате этот маршрут добавляется к дереву. Наконец, к группе рассылки присоединяется узел A, посылая сообщение с запросом об этом маршрутизатору E. Предположим, что одноадресный маршрут от маршрутизатора A до маршрутизатора E проходит через узел B. Поскольку узел B уже присоединился к дереву групповой рассылки, сразу по прибытии сообщения от узла A на узел B линия AB добавляется к дереву.

Ключевым вопросом групповой маршрутизации с центральным узлом является выбор центрального узла. Алгоритмы выбора центрального узла обсуждаются в [138, 500, 547].

Групповая маршрутизация с деревом у каждого отправителя

В рассмотренных нами алгоритмах создается общее для группы дерево, используемое для маршрутизации пакетов от всех отправителей. Второй большой класс алгоритмов образуют алгоритмы групповой маршрутизации, в которых дерево групповой маршрутизации строится для *каждого* отправителя группы рассылки.

Мы уже познакомились с алгоритмом Дейкстры, основанным на учете состояния линий (см. подраздел «Алгоритм маршрутизации, основанный на состоянии линий» в разделе «Основы маршрутизации»). Этот алгоритм выполняет поиск одноадресных маршрутов с наименьшей стоимостью от одного отправителя до всех получателей. Объединение всех этих маршрутов можно рассматривать как *дерево выборочной маршрутизации наименьшей стоимости* (или кратчайшее дерево выборочной маршрутизации, если стоимости всех линий одинаковы). Возможно, вам захочется убедиться в том, что дерево маршрутов наименьшей стоимости не совпадает с деревом, суммарная стоимость всех маршрутов которого минимальна (решение задачи Штайнера). Обратите внимание, что для групповой маршрутизации с использованием путей наименьшей стоимости требуется, чтобы каждому маршрутизатору было известно состояние каждой линии связи в сети. Только это позволяет вычислить дерево маршрутов наименьшей стоимости от одного отправителя до всех получателей. Более простым алгоритмом групповой маршрутизации, для которого требуется гораздо меньше информации о состоянии линий, является алгоритм *RPF(* Reverse Path Forwarding — продвижение данных по обратным маршрутам).

Лежащая в основе алгоритма RPF идея проста, но элегантна. Когда маршрутизатор получает групповой пакет с определенным адресом отправителя, он передает этот пакет по всем исходящим линиям (кроме, естественно, той, по которой пришел пакет) только в том случае, если пакет прибыл по линии, входящей в кратчайший путь к отправителю. В противном случае маршрутизатор просто отбрасывает полученный пакет, поскольку обязательно получит (либо уже получил) копию

этого пакета по линии, входящей в кратчайший путь к отправителю. Обратите внимание, что алгоритм продвижения данных по обратным маршрутам не требует, чтобы маршрутизатор знал весь кратчайший путь от себя до отправителя. Ему достаточно знать только направление (одну линию).

Работу алгоритма RPF иллюстрирует рис. 4.51. Жирными линиями показаны маршруты с наименьшей стоимостью от получателей до отправителя (узла A). Сначала маршрутизатор A рассылает копии полученного от хоста S пакета маршрутизаторам C и B. Затем маршрутизатор B переправляет полученный от маршрутизатора A (так как маршрутизатор A находится на пути наименьшей стоимости до маршрутизатора A) пакет маршрутизаторам C и D и игнорирует (отбрасывает, не переправляя) все пакеты от хоста S, полученные им от любых других маршрутизаторов (например, от маршрутизатора C или D).

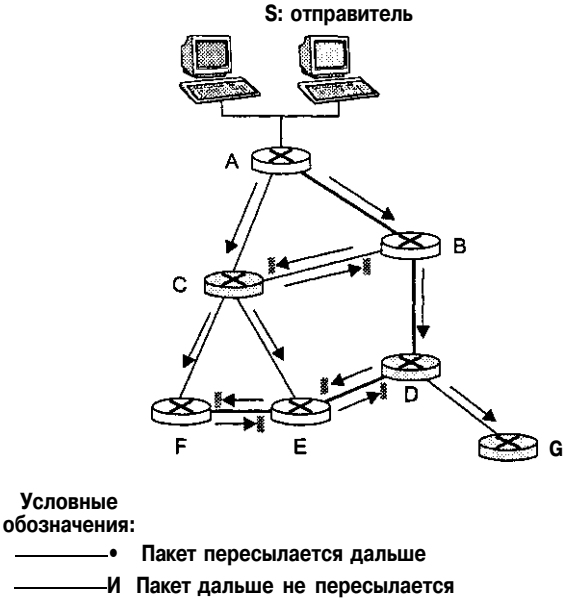


Рис. 4.51. Алгоритм продвижения данных по обратным маршрутам

Рассмотрим теперь маршрутизатор C, который отправленные хостом S пакеты получает как напрямую от маршрутизатора A, так и от маршрутизатора B. Поскольку маршрутизатор B не находится на кратчайшем пути от узла C до узла A, маршрутизатор C игнорирует все отправленные хостом S пакеты, полученные им от маршрутизатора B. С другой стороны, когда маршрутизатор C получает отправленный хостом S пакет напрямую от маршрутизатора A, он переправляет его маршрутизаторам B, E и F.

Алгоритм RPF работает очень остроумно. Но посмотрим, что происходит на маршрутизаторе D. Он переправляет пакеты маршрутизатору G, хотя у маршрутизатора G нет хостов, относящихся к группе рассылки. Такое поведение алгоритма не так уж страшно, когда у маршрутизатора D ниже по дереву располагается всего

один маршрутизатор (G), но представьте себе, что произойдет, если таких маршрутизаторов несколько тысяч! Каждый из этих тысяч маршрутизаторов получил бы ненужные ему групповые пакеты. (Этот сценарий вовсе не так фантастичен, как это может показаться. Первая глобальная сеть групповой рассылки Mbone [58, 302] вначале страдала именно таким недостатком!)

Решение проблемы получения ненужных групповых пакетов в алгоритме RPF называют *отсечением*. Маршрутизатор, получающий групповые пакеты, у которого нет соединенных с ним напрямую хостов, принадлежащих этой группе, посылает отсекающее сообщение маршрутизатору «выше по течению». Если маршрутизатор получает отсекающие сообщения от всех своих маршрутизаторов «ниже по течению», тогда он также может послать отсекающее сообщение маршрутизатору «выше по течению».

Групповая маршрутизация в Интернете

Изучив теоретические аспекты применения алгоритмов групповой маршрутизации, поговорим о том, как эти алгоритмы реализуются на практике в сегодняшнем Интернете. Мы обсудим два стандартных алгоритма групповой маршрутизации, DVMRP и PIM.

Протокол DVMRP

Первым протоколом групповой маршрутизации, получившим широкое распространение в Интернете (RFC 1075), был протокол DVMRP (Distance Vector Multicast Routing Protocol — дистанционно-векторный протокол групповой маршрутизации). В протоколе DVMRP используются деревья с вершиной в источнике с продвижением данных по обратному маршруту и отсечениями. Протокол DVMRP основан на дистанционно-векторном алгоритме (см. раздел «Основы маршрутизации»), в котором каждый маршрутизатор находит исходящую линию (следующий ретрансляционный участок) на кратчайшем обратном маршруте к каждому возможному отправителю. Затем эта информация используется в алгоритме RPF (см. предыдущий подраздел). На сайте ftp://parcftp.xerox.com/pub/net-research/ipmulti можно найти свободно распространяемое программное обеспечение для протокола DVMRP.

Помимо нахождения следующего ретрансляционного участка протокол DVMRP также рассчитывает список зависимых маршрутизаторов «ниже по течению» для решения задачи отсечения. Когда маршрутизатор получает отсекающие сообщения от всех своих зависимых для данной группы маршрутизаторов, он посылает отсекающее сообщение маршрутизатору «выше по течению», от которого получает трафик групповой рассылки для группы. Отсекающее сообщение протокола DVMRP содержит поле времени действия отсечения, указывающее, как долго отсеченная ветвь останется отсеченной, прежде чем будет автоматически восстановлена (по умолчанию два часа). Чтобы добавить отсеченную ранее ветвь к дереву, маршрутизатор посылает своему соседу выше сообщение протокола DVMRP о присоединении ветви.

Прежде чем перейти к рассмотрению других алгоритмов групповой маршрутизации, поговорим о том, как групповая маршрутизация реализуется в Интернете. a

Проблема заключается в том, что только небольшая часть маршрутизаторов Интернета поддерживают групповую маршрутизацию. Если один маршрутизатор поддерживает групповую маршрутизацию, а его непосредственные соседи ее не поддерживают, не потеряется ли этот островок групповой маршрутизации в океане одноадресных маршрутизаторов? Со всей решительностью заявляем, что нет! Для создания виртуальной сети групповых маршрутизаторов поверх физической сети, в которой перемешаны маршрутизаторы для выборочной и групповой рассылки, можно использовать туннелирование, обсуждавшееся нами в контексте версии 6 протокола IP (см. раздел «Протокол IPv6»). Именно этот метод применяется в магистрали MBone Интернета.

Схематически туннели для групповой рассылки показаны на рис. 4.52. Пусть групповой маршрутизатор A желает переслать групповую дейтаграмму групповому маршрутизатору B. Предположим, что маршрутизаторы A и B не имеют непосредственного физического соединения друг с другом, а промежуточные маршрутизаторы не поддерживают групповой маршрутизации. В этом случае маршрутизатор A инкапсулирует (RFC 2003) групповую дейтаграмму в стандартную одноадресную дейтаграмму. Таким образом, вся групповая дейтаграмма целиком (включая поля адресов отправителя и группы получателей) переносится в одноадресной IP-дейтаграмме как полезная нагрузка. В поле адреса получателя одноадресной дейтаграммы указывается адрес маршрутизатора B, после чего эта дейтаграмма пересылается маршрутизатору B. Одноадресные маршрутизаторы между маршрутизаторами A и B послушно переправляют дейтаграмму маршрутизатору B, не догадываясь о том, что в ней содержится групповая дейтаграмма. Получив одноадресную дейтаграмму, маршрутизатор B извлекает из нее групповую. Затем маршрутизатор B может переслать эту групповую дейтаграмму одному из присоединенных к нему хостов, переправить ее напрямую соединенному с ним маршрутизатору, поддерживающему групповую маршрутизацию, или переправить ее соседнему групповому маршрутизатору при помощи другого туннеля.

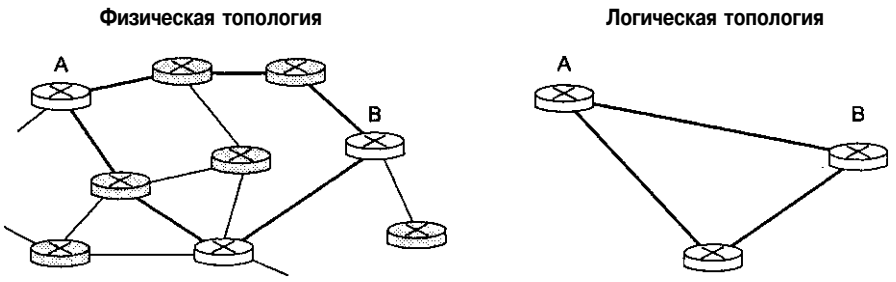


Рис. 4.52. Туннели для групповой рассылки

Протокол PIM

Протокол PIM (Protocol Independent Multicast — независимая от протокола групповая рассылка) предлагает два разных сценария групповой рассылки [115, 137, 139]. Так называемый *плотный режим* работы протокола рассчитан на ситуацию,

когда члены группы рассылки располагаются плотно, то есть большая часть маршрутизаторов некоторой области задействована в групповой рассылке дейтаграмм.

Разреженный режим работы протокола предназначен для случая, когда маршрутизаторов, хосты которых являются членами группы, относительно немного и эти маршрутизаторы разбросаны далеко друг от друга.

Разработчики протокола PIM отметили несколько последствий применения каждого из режимов работы протокола. Поскольку в плотном режиме в групповую рассылку вовлекается большая часть маршрутизаторов, разумно положить, что это должны делать все маршрутизаторы. Для такой ситуации хорошо подходит применяемый в протоколе RPF метод лавинной маршрутизации, когда групповые дейтаграммы пересылаются всем групповым маршрутизаторам (кроме тех, кто явным образом отсечет себя от дерева). В разреженном режиме, напротив, маршрутизаторов, которые должны участвовать в групповой рассылке, немного, и они удалены друг от друга. В этом случае представляется значительно менее привлекательным применяемый в протоколе RPF метод, постоянно заставляющий маршрутизатор посылать отсекающие сообщения, чтобы избежать получения ненужного ему группового трафика. В разреженном режиме протокол исходит из предположения, что маршрутизатор по умолчанию не участвует в групповой рассылке. То есть маршрутизатор не должен выполнять никаких действий, если только не хочет присоединиться к группе рассылки. Такие рассуждения предполагают подход с выбором центрального узла, которому маршрутизаторы явным образом посылают сообщения с запросом о присоединении к той или иной группе рассылки. Остальные маршрутизаторы никак не участвуют в групповой рассылке. Таким образом, разреженный режим работы протокола может рассматриваться как управляемый получателем (ничего не случится до тех пор, пока получатель явно не заявит о желании вступить в группу), а плотный режим работы протокола — как управляемый отправителем (отправитель рассылает дейтаграммы всем, если только маршрутизаторы явно не заявят о своем нежелании их получать).

В плотном режиме работы протокол PIM реализует метод лавинной маршрутизации по обратным маршрутам с отсечениями, то есть, по сути, напоминает протокол DVMRP. Вспомним, что протокол PIM не зависит от нижележащего протокола выборочной (одноадресной) рассылки. Другими словами, протокол PIM может взаимодействовать с любым протоколом одноадресной маршрутизации. Поскольку протокол PIM не предъявляет никаких требований к нижележащему протоколу одноадресной маршрутизации, используемый в нем алгоритм продвижения данных по обратному маршруту несколько проще, чем в протоколе DVMRP, но он также и менее эффективный.

В разреженном режиме работы протокола PIM используется метод центрального узла (RFC 2189, RFC 2201), аналогичный применявшемуся в более раннем протоколе групповой маршрутизации CBT (Core-Based Tree — дерево с вершиной в ядре). В протоколе PIM маршрутизаторы посылают центральному маршрутизатору, называемому точкой встречи, сообщение JOIN (присоединиться), чтобы присо-

единиться к дереву. Как и в протоколе CBT, промежуточные маршрутизаторы переходят в состояние групповой рассылки и переправляют сообщение JOIN точке встречи. В отличие от протокола CBT, в протоколе PIM в ответ на сообщение JOIN не посылается подтверждение. Сообщения JOIN периодически отправляются «вверх по течению» для обновления состояния дерева маршрутов протокола PIM. Еще одна особенность протокола PIM заключается в способности переключаться с общего для группы дерева на дерево конкретного отправителя после присоединения к точке встречи. Это может оказаться полезным, так как при использовании нескольких деревьев, построенных для конкретных отправителей, концентрация трафика уменьшается (см. упражнения в конце главы).

В разреженном режиме работы протокола PIM маршрутизатор, получивший дейтаграмму от одного из присоединенных к нему хостов, пересылает ее точке встречи. Затем точка встречи рассылает эту дейтаграмму по общему для группы дереву, используя групповую маршрутизацию. Если к дереву не присоединился ни один из маршрутизаторов (то есть у данного источника групповой рассылки нет получателей), отправитель уведомляется точкой встречи о том, что он должен прекратить посылать ей дейтаграммы.

Протокол PIM реализован на многих маршрутизаторах, а также развернут в сети UUNet как часть проекта потоковой доставки мультимедиа [528]. Другим реализованным протоколом групповой маршрутизации является протокол MOSPF (Multicast OSPF — протокол OSPF для групповой рассылки). Протокол MOSPF (RFC 1584) работает в автономных системах, использующих протокол OSPF для выборочной рассылки (см. раздел «Маршрутизация в Интернете»), и представляет собой расширение протокола OSPF. В этом протоколе маршрутизаторы добавляют к распространяемым путем широковещательной рассылки уведомлениям о состоянии линий информацию о членстве своих хостов в группах.

Внешняя групповая маршрутизация

Выше неявно предполагалось, что все маршрутизаторы используют один и тот же протокол групповой маршрутизации. Как было показано при обсуждении одноадресной маршрутизации, такая ситуация верна в пределах одной автономной системы. Однако в разных автономных системах могут работать разные протоколы групповой маршрутизации. Для основных протоколов групповой маршрутизации Интернета были разработаны правила взаимодействия (RFC 2715). (Эти правила исключительно беспорядочны, что вызвано принципиально различными подходами к групповой маршрутизации в протоколах, работающих в плотном и разреженном режимах.) Однако до сих пор отсутствует протокол внешней групповой маршрутизации, который позволял бы определять маршруты групповых дейтаграмм между автономными системами.

Протоколом внешней групповой маршрутизации де-факто стал протокол DVMRP. Однако в качестве протокола плотного режима он не очень хорошо подходит для наблюдающейся сегодня ситуации в магистрали MBone Интернета, где участвующие в групповой рассылке маршрутизаторы разбросаны довольно далеко друг от

друга. Разработка протокола внешней групповой маршрутизации представляет собой область активных исследований, осуществляемых рабочей группой *IDMR* (InterDomain Multicast Routing — междоменная групповая маршрутизация), входящей в группу IETF [228]. Обсуждение внешней групповой маршрутизации, комментарии к модели обслуживания, основанной на групповой рассылке, вопросы развертывания и прогнозы на будущее можно найти в [126].

Мобильность и сетевой уровень

В последнее время люди все активнее перемещаются в пространстве и все больше применяют лэптопы, палмтопы, устройства PDA и мобильные телефоны. Это означает, что все большее количество сетевых устройств оказывается в движении. До сих пор в нашем обсуждении сетевого уровня неявно подразумевалась статическая сетевая инфраструктура, в которой точка подключения хоста к сети (то есть маршрутизатор, соединенный с ним напрямую) не изменяется во времени. В этом разделе мы рассмотрим мобильные узлы, которые могут изменять точку подключения. Мы увидим, что для поддержания мобильности требуется внести ряд существенных дополнений в архитектуру сетевого уровня.

Учет мобильности в структуре сетевого уровня

Поскольку понятие *мобильности* получило так много значений как в компьютерном, так и в телефонном мире, рассмотрим сначала несколько аспектов мобильности.

- *Насколько мобилен пользователь с точки зрения сетевого уровня?* Физически мобильный пользователь предъявляет к сетевому уровню значительно различающиеся наборы требований в зависимости от того, как он перемещается между точками подключения к сети. С одной стороны, пользователь может переносить по зданию лэптоп со встроенной интерфейсной картой беспроводного сетевого доступа. Если при этом постоянно задействован один и тот же беспроводной канал, то пользователь не является мобильным с точки зрения сетевого уровня независимо от физического расположения пользователя в пространстве. С другой стороны, рассмотрим пользователя, который едет по автостраде на своем автомобиле со скоростью 150 км/ч и, проезжая через многочисленные беспроводные сети доступа, желает иметь непрерываемое соединение с удаленным приложением. Такой пользователь, определенно, мобилен! Между этими крайностями находится пользователь, переносящий свой лэптоп из одного помещения (например, офиса или спальни) в другое (например, в кафе, класс или другое офисное здание) и желающий подключиться к сети на новом месте. Этот пользователь также мобилен (хотя и в меньшей степени, чем водитель), но ему не нужно поддерживать соединение в то время, когда он перемещается между двумя точками доступа к сети. Этот спектр мобильности пользователей иллюстрирует рис. 4.53.
- *Насколько важна мобильность сетевого адреса ?* В случае мобильного телефона ваш телефонный номер (по сути, сетевой адрес вашего телефона) остается не-

изменным, в то время как вы перемещаетесь из сети одного оператора сотовой связи в сеть другого. Должен ли лэптоп сохранять свой сетевой IP-адрес при перемещении из одной IP-сети в другую?

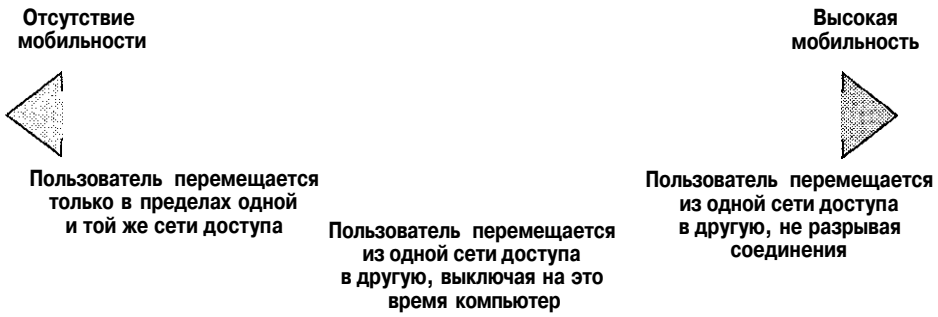


Рис. 4.53. Различные степени мобильности с точки зрения сетевого уровня

Ответ на этот вопрос в значительной мере будет зависеть от используемого приложения. С одной стороны, для водителя, желающего поддерживать постоянное TCP-соединение с удаленным приложением и в то же время продолжающего двигаться по автостраде, было бы удобно иметь постоянный IP-адрес. Как показано в предыдущей главе, Интернет-приложению нужно знать IP-адрес и номер порта удаленного приложения, с которым оно обменивается данными. Если мобильный хост способен сохранять свой IP-адрес во время движения, его мобильность оказывается скрытой от прикладного уровня. Достоинство подобной прозрачности неоспоримо — приложению не нужно заботиться о возможном изменении IP-адреса, поэтому одно и то же приложение может работать как на мобильных, так и на стационарных хостах. В подразделе «Мобильный протокол IP» данного раздела мы увидим, что прозрачность обеспечивает мобильный протокол IP, позволяющий мобильному узлу сохранять постоянный IP-адрес во время перемещения из одной сети в другую.

С другой стороны, мобильному пользователю с меньшими запросами может быть достаточно просто выключить свой лэптоп в офисе, принести его домой, включить питание и подключиться к сети. Если на лэптопе работает в основном клиентская часть приложений клиент-сервер (например, пользователь отправляет и получает электронную почту, путешествует в web, устанавливает Telnet-соединение с удаленным хостом), IP-адрес лэптопа не так уж важен. Такого пользователя прекрасно устроит временный IP-адрес, выделенный ему Интернет-провайдером при подключении из дома. В разделе «Интернет-протокол» было показано, что подобную функциональность предоставляет протокол DHCP (Dynamic Host Configuration Protocol — протокол динамической конфигурации хоста).

- *Какая существует кабельная инфраструктура, поддерживающая мобильные хосты?* Во всех наших обсуждавшихся выше сценариях неявно предполагалось, что существует фиксированная инфраструктура, к которой может подключиться

мобильный пользователь, например домашняя сеть Интернет-провайдера, сеть беспроводного доступа в офисе или сети беспроводного доступа, тянущиеся вдоль автострады. А что если такой инфраструктуры не существует? Если два пользователя находятся недалеко друг от друга, могут ли они установить сетевое соединение при отсутствии сетевой инфраструктуры? Подобную возможность предоставляют так называемые спецсети (ad hoc networks). Это быстро развивающаяся область исследований мобильных сетей, и ее обсуждение выходит за рамки темы этой книги. Подробное описание можно найти в [229,379].

Как уже упоминалось, методы выделения динамических адресов, вроде применяющихся в протоколе DHCP (см. раздел «Интернет-протокол»), обеспечивают только ограниченную мобильность. Мобильный узел может соединяться с сетью в разных точках доступа, но не может использовать сетевые приложения во время перемещения от одной точки доступа к другой. Рассмотрим теперь случай, когда узел желает иметь постоянный адрес и не разрывать свои сетевые соединения, перемещаясь в то же время из одной сети в другую. Сначала мы обсудим принципы и методы, обеспечивающие подобную мобильность, а затем познакомимся с реализацией этих принципов в стандарте мобильного протокола IP.

Управление мобильной связью

Чтобы проиллюстрировать некоторые аспекты предоставления мобильному пользователю возможности поддерживать непрерывные во времени соединения при перемещении между сетями, рассмотрим аналогию из мира людей. Молодой человек, покидающий родительский дом, становится мобильным. Он часто меняет адреса, проживая в общежитиях или снимая квартиры. Если старый знакомый хочет с ним связаться, как ему узнать новый адрес своего мобильного друга? Проще всего связаться с родителями друга, так как мобильная молодежь часто информирует своих родителей о своем новом месте жительства (хотя бы для того, чтобы родители могли прислать денег на аренду жилья!). Таким образом, родительский дом с постоянным адресом становится тем местом, куда можно обратиться на первом этапе установки контакта с мобильным юнцом. После этого общение старых друзей может быть непосредственным (например, друг узнает адрес/телефон своего мобильного приятеля, чтобы писать/звонить ему напрямую) или косвенным (например, предназначенные мобильному другу письма попадают сначала на адрес его родителей и только потом ему).

В контексте сетей постоянный «дом» мобильного узла (например, лэптопа) называется *домашней сетью,* а устройство в домашней сети, выполняющее обсуждающиеся ниже функции управления мобильной связью от имени мобильного узла, называется *домашним,* или *внутренним, агентом.* Текущая сеть, в которой находится мобильный узел, называется *внешней,* или *посещаемой, сетью,* а устройство во внешней сети, помогающее мобильному узлу справиться с управлением мобильными функциями (см. далее), называется *внешним агентом.* Для мобильных профессионалов их домашней сетью может быть сеть их компании, а посещаемой сетью — сеть посещаемых ими коллег. Узел, желающий общаться с мобильным хостом, называют *корреспондентом.* Проиллюстрировать эти, а также рассматри-

ваемые далее концепции адресации призван рис. 4.54. Агенты на рисунке показаны в виде маршрутизаторов (это могут быть процессы, выполняющиеся на маршрутизаторах), но они также могут выполняться на хостах или на серверах сети.

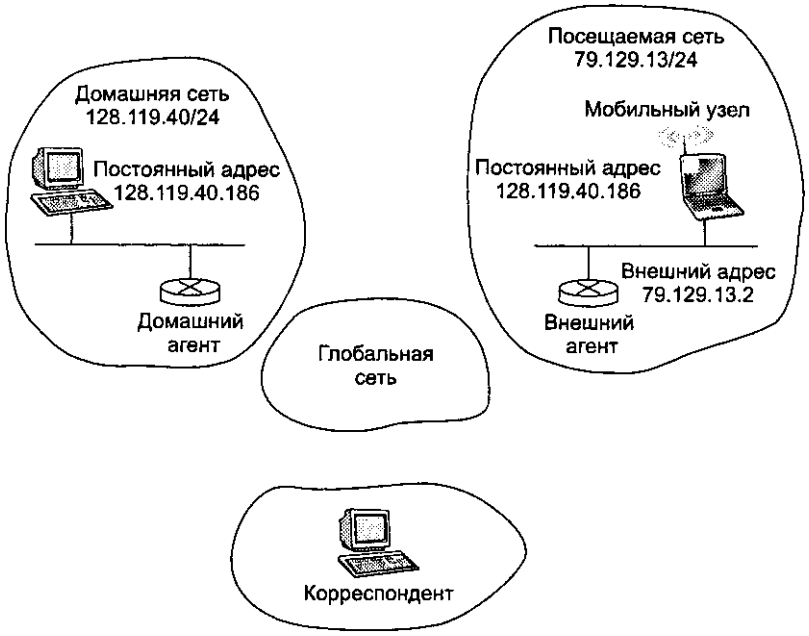


Рис. 4.54. Основные компоненты мобильной сетевой архитектуры

Адресация

Как отмечалось ранее, для скрытия от сетевых приложений факта мобильности пользователя было бы желательно, чтобы мобильный узел мог сохранять свой IP-адрес во время перемещения из одной сети в другую. Когда мобильный узел находится во внешней сети, весь адресованный постоянному адресу узла трафик должен быть направлен во внешнюю сеть. Как этого достичь? В одном из способов внешняя сеть объявляет всем остальным сетям о том, что мобильный узел находится в ней. Это делается путем обычного обмена внутридоменной и междоменной информацией о маршрутах, что требует незначительных изменений существующей инфраструктуры маршрутизации. Внешняя сеть может просто объявить своим соседям о наличии специфического маршрута к постоянному адресу мобильного узла, то есть проинформировать другие сети о том, что у нее имеется корректный путь для дейтаграмм, направляемых по постоянному адресу мобильного узла (см. подраздел «Адресация в протоколе IPv4» в разделе «Интернет-протокол»). Затем эти соседи распространяют по сети информацию о маршрутах как часть обычной процедуры обновления информации в таблицах продвижения данных. Когда мобильный узел покидает внешнюю сеть и присоединяется к другой сети, новая посещаемая сеть объявляет о новом специфическом маршруте к мобильному узлу, а старая внешняя сеть удаляет информацию о маршруте к мобильному узлу.

Такой подход позволяет решить две проблемы сразу, причем для этого не требуется вносить значительных изменений в инфраструктуру сетевого уровня. Другим сетям известно расположение мобильного узла, поэтому дейтаграммы, направляемые мобильному узлу, переправляются во внешнюю сеть. Однако значительный недостаток этого метода связан с масштабируемостью. Если предположить, что за управление мобильной связью должны отвечать сетевые маршрутизаторы, тогда маршрутизаторам придется содержать таблицы продвижения данных для, возможно, миллионов мобильных узлов.

Альтернативный подход (применяемый на практике) состоит в том, что средства обеспечения мобильности перемещаются из ядра сети на ее периферию — знакомый сюжет в нашем подходе к изучению архитектуры Интернета. Естественный способ реализации заключается в использовании домашней сети мобильного узла. Подобно тому как родители мобильного тинэйджера отслеживают его положение в пространстве, домашний агент в домашней сети мобильного узла может проконтролировать, в какой внешней сети находится мобильный хост в данный момент. Разумеется, необходим протокол взаимодействия между мобильным узлом (или внешним агентом, представляющим мобильный узел) и домашним агентом для обновления информации о положении мобильного узла.

Рассмотрим теперь внешний агент более подробно. В соответствии с концептуально простейшим подходом, который иллюстрирует рис. 4.54, внешние агенты должны располагаться на периферийных маршрутизаторах внешних сетей. Одна из задач внешнего агента состоит в создании *внешнего адреса* мобильного узла, сетевая часть которого совпадает с сетевым адресом внешней сети. Таким образом, у мобильного узла оказывается два адреса: *постоянный адрес* (аналогичный адресу родителей мобильного юноши) и *внешний адрес* (аналогичный адресу, по которому мобильный юноша временно проживает). В примере на рисунке постоянный адрес мобильного узла равен 128.119.40.186, а на время посещения сети 79.129.13/24 мобильному узлу выделяется внешний адрес 79.129.13.2. Вторая задача внешнего агента заключается в информировании домашнего агента о том, что мобильный узел находится в его (внешнего агента) сети и что ему (узлу) выделен определенный внешний адрес. Мы вскоре увидим, что внешний адрес требуется для перенаправления дейтаграмм мобильному узлу через его внешний агент.

Несмотря на то что мы разделили функции мобильного узла и внешнего агента, следует отметить, что мобильный узел сам может выполнять функции внешнего агента. Например, мобильный узел может сам получить временный внешний адрес в посещаемой им сети (например, при помощи протокола DHCP) и сам же сообщить этот адрес своему домашнему агенту.

Итак, мы выяснили, как мобильный узел получает внешний адрес и как домашний агент может узнать этот адрес. Но это решает лишь часть проблемы. Как адресовать и переправлять дейтаграммы мобильному узлу? Поскольку положение мобильного узла известно только домашнему агенту (и не известно маршрутизаторам глобальной сети), недостаточно просто указать в дейтаграмме постоянный адрес мобильного узла и передать дейтаграмму инфраструктуре сетевого уровня.

Необходимо предпринять еще какие-то действия. Для решения проблемы применяются два различных подхода, которые мы будем называть косвенной и непосредственной маршрутизацией.

Косвенная маршрутизация

Рассмотрим сначала корреспондента, желающего послать дейтаграмму мобильному узлу. В случае косвенной маршрутизации корреспондент просто указывает в поле адреса дейтаграммы постоянный адрес мобильного узла и отправляет дейтаграмму в сеть, не задумываясь о том, где реально находится мобильный узел. Таким образом, мобильность в данном случае оказывается абсолютно прозрачной для корреспондента. Дейтаграммы сначала направляются, как обычно, в домашнюю сеть мобильного узла (шаг 1 на рис. 4.55).

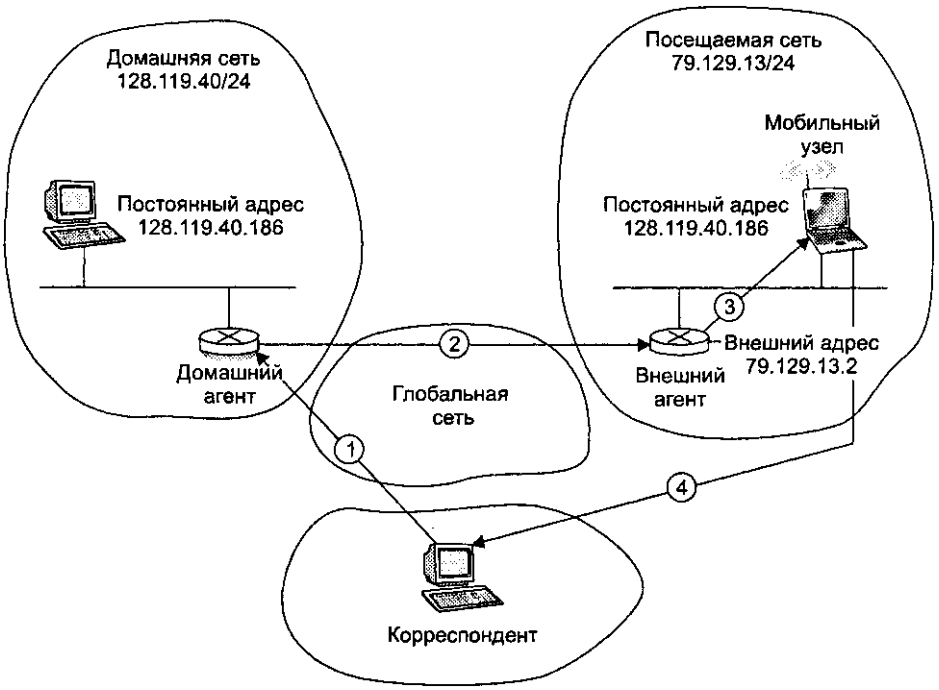


Рис. 4.55. Косвенное продвижение данных к мобильному узлу

Обратим теперь наше внимание на домашний агент. Помимо взаимодействия с внешним агентом для отслеживания внешнего адреса мобильного узла у домашнего агента есть еще одна очень важная функция. Его вторая задача заключается в выявлении тех прибывающих дейтаграмм, которые адресованы узлам, находящимся вдали от дома. Домашний агент перехватывает эти дейтаграммы, а затем в два этапа переправляет их мобильному узлу. Сначала дейтаграмма посылается внешнему агенту на внешний адрес узла (шаг 2), а затем внешний агент передает ее мобильному узлу (шаг 3).

Полезно рассмотреть процесс изменения маршрута следования дейтаграммы более детально. С одной стороны, домашний агент должен адресовать дейтаграмму, используя внешний адрес мобильного узла так, чтобы сетевой уровень мог направить дейтаграмму во внешнюю сеть. С другой стороны, желательно оставить саму дейтаграмму неизмененной, потому что получающее дейтаграмму приложение не должно знать о том, что та «добиралась» до приложения через домашний агент. Обеих целей можно достичь, если домашний агент инкапсулирует оригинальную дейтаграмму корреспондента целиком в новой дейтаграмме (большего размера). В поле адреса этой новой дейтаграммы указывается внешний адрес мобильного узла. По этому адресу она и доставляется. Внешний агент, которому «принадлежит» внешний адрес, получает дейтаграмму, извлекает из нее оригинальную дейтаграмму корреспондента и переправляет ее мобильному узлу (шаг 3). На рис. 4.56 показаны оригинальная дейтаграмма, отправленная в домашнюю сеть, инкапсулированная дейтаграмма, посланная внешнему агенту, и снова оригинальная дейтаграмма, доставленная мобильному узлу. Внимательный читатель может заметить, что описанные здесь процедуры инкапсуляция и извлечения инкапсулированной дейтаграммы идентичны туннелированию, обсуждавшемуся ранее в этой главе в контексте групповой рассылки и протокола IPv6.

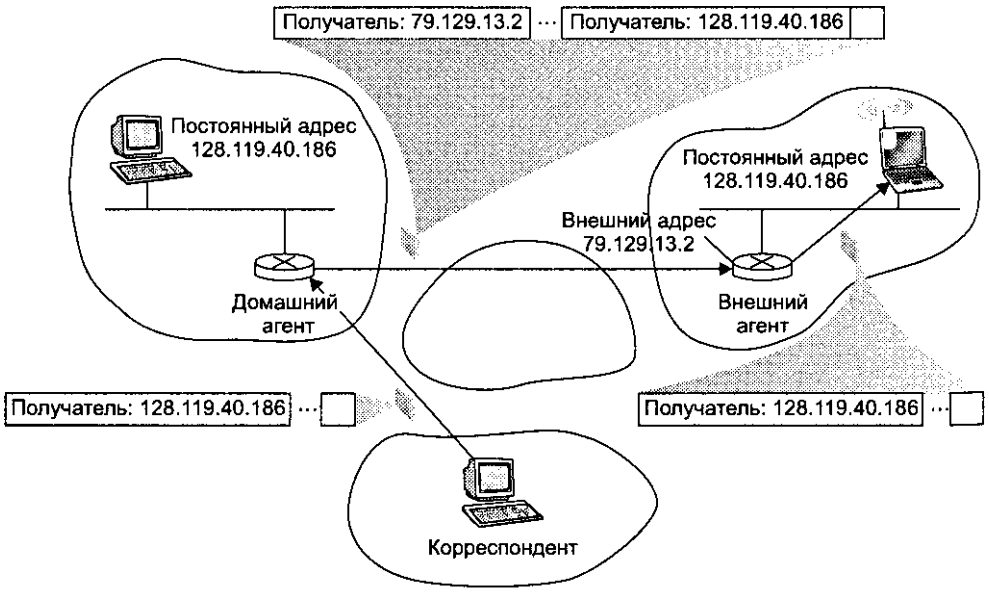


Рис. 4.56. Инкапсуляция и извлечение инкапсулированной дейтаграммы

Поговорим теперь о том, как мобильный узел посылает дейтаграммы корреспонденту. Это совсем просто, так как мобильный узел может адресовать дейтаграмму корреспонденту *напрямую* (используя собственный постоянный адрес и адрес корреспондента в качестве адресов отправителя и получателя). Поскольку адрес корреспондента известен мобильному узлу, нет необходимости направлять дейтаграмму обратно через домашний агент. На рис. 4.55 это действие показано как шаг 4.

Подведем итоги нашего обсуждения косвенной маршрутизации, перечислив новые функции сетевого уровня, необходимые для поддержания мобильности.

- *Протокол между мобильным узлом и внешним агентом* позволяет мобильному узлу зарегистрироваться у внешнего агента при установке соединения с внешней сетью. Аналогично, мобильный узел отменяет регистрацию, покидая посещаемую сеть.
- *Регистрационный протокол между внешним и домашним агентами* позволяет внешнему агенту зарегистрировать внешний адрес мобильного хоста у домашнего агента. Когда мобильный узел покидает сеть, внешний агент не должен явно отменять регистрацию внешнего адреса, так как об этом позаботится внешний агент новой посещаемой мобильным узлом сети во время следующей регистрации.
- *Протокол инкапсуляции дейтаграмм домашнего агента* обеспечивает инкапсуляцию и пересылку оригинальной дейтаграммы корреспондента мобильному узлу.
- *Протокол извлечения инкапсулированных дейтаграмм внешнего агента* обеспечивает выделение оригинальной дейтаграммы корреспондента из дейтаграммы домашнего агента и передачу ее мобильному узлу.

Итак, мы поговорили обо всем, что требуется мобильному узлу для поддержания непрерывного соединения при перемещении из одной сети в другую, то есть о внешних агентах, домашнем агенте и косвенной маршрутизации. Посмотрим теперь, как все это работает вместе. Пусть мобильный узел соединен с внешней сетью A, и его внешний адрес в сети A зарегистрирован у домашнего агента. Пусть также мобильный узел принимает дейтаграммы, направляемые косвенным образом через его домашний агент. Затем мобильный узел перемещается в сеть B, в которой регистрируется у внешнего агента. Внешний агент сети B сообщает домашнему агенту мобильного узла новый внешний адрес мобильного узла. С этого момента домашний агент будет направлять дейтаграммы в сеть B. С точки зрения корреспондента, посылающего дейтаграммы мобильному узлу, мобильность остается прозрачной — дейтаграммы переправляются через один и тот же домашний агент как до, так и после перемещения мобильного узла из одной сети в другую. С точки зрения домашнего агента поток дейтаграмм является непрерывным — сначала поступающие дейтаграммы направлялись во внешнюю сеть A, а после смены внешнего адреса мобильного узла они направляются во внешнюю сеть B. Но заметит ли мобильный узел прерывание потока дейтаграмм при переходе из одной сети в другую? Если время между отключением узла от сети A и подключением его к сети B окажется небольшим, количество потерянных дейтаграмм также будет невелико. Как упоминалось в главе 3, сквозные соединения допускают потерю определенного количества дейтаграмм, например во время сетевой перегрузки. Таким образом, потеря нескольких дейтаграмм при перемещении узла из одной сети в другую не является катастрофой. Если требуется связь без потерь данных, это может быть решено на более высоком уровне (например, транспортном) путем повторной передачи потерянных дейтаграмм независимо от того, чем были вызваны потери, перегрузкой в сети или мобильностью пользователя.

Метод косвенной маршрутизации используется в стандарте мобильного протокола IP (RFC 3220), который будет обсуждаться в подразделе «Мобильный протокол 1P».

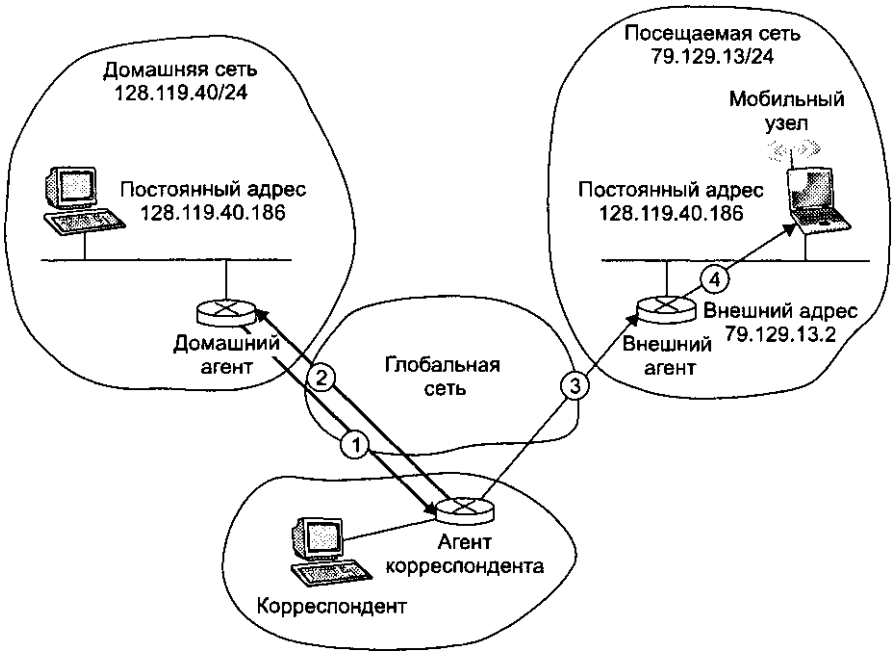
Непосредственная маршрутизация

Недостатком проиллюстрированного на рис. 4.55 метода косвенной маршрутизации является низкая эффективность. Данная проблема получила название *проблемы треугольной маршрутизации* — дейтаграммы, адресованные мобильному узлу, должны сначала переправляться домашнему агенту, а уже затем в посещаемую мобильным узлом сеть, даже при наличии более эффективного маршрута между корреспондентом и мобильным узлом. В худшем случае представим себе мобильного пользователя, зашедшего к своему коллеге. Хотя коллеги сидят рядом друг с другом, при обмене данными по сети дейтаграммы от корреспондента (в данном случае коллеги пользователя) направляются домашнему агенту мобильного пользователя и только затем возвращаются обратно в посещаемую сеть!

Данная проблема решается путем *непосредственной маршрутизации,* которая несколько сложнее косвенной маршрутизации. При непосредственной маршрутизации *агент корреспондента* в сети корреспондента сначала узнает внешний адрес мобильного узла. (Корреспондент может выполнять функцию агента и самостоятельно, так же как мобильный узел может выполнять функцию внешнего агента.) Эти действия показаны как шаги 1 и 2 на рис. 4.57. Затем агент корреспондента туннелирует дейтаграммы напрямую по внешнему адресу мобильного узла (шаги 3 и 4) аналогично тому, как их туннелировал домашний агент.

Хотя метод непосредственной маршрутизации позволяет решить проблему треугольной маршрутизации, он несколько усложняет процедуру. Для того чтобы агент корреспондента мог узнать внешний адрес мобильного узла (шаги 1 и 2), требуется специальный протокол. Другое усложнение метода имеет место при перемещении мобильного узла из одной сети в другую. Возможное решение заключается в создании нового протокола для уведомления корреспондента об изменении внешнего адреса. Альтернативное решение состоит в том, что новый внешний агент сообщает новый внешний адрес мобильного узла прежнему внешнему агенту. Внешний агент, получив инкапсулированную дейтаграмму для покинувшего сеть мобильного узла, переправляет ее по новому внешнему адресу. Если мобильный узел перемещается по нескольким сетям, образуется целая цепочка внешних агентов, передающих друг другу дейтаграммы. Таким образом, хотя последний метод позволяет избавиться от уведомления корреспондента об изменении внешнего адреса мобильного узла, он требует дополнительной координации между внешними агентами.

Метод непосредственной маршрутизации применяется для маршрутизации телефонных звонков в некоторых стандартах сотовой телефонии, например в системе GSM [297]. В настоящее время рассматривается возможность расширения мобильного стандарта протокола IP для поддержания непосредственной маршрутизации [382].



Условные обозначения:

—• **Управляющие сообщения**
—• **Поток данных**

Рис. 4.57. Непосредственное продвижение данных к мобильному узлу

Мобильный протокол IP

Архитектура Интернета и протоколы поддержания мобильности, называемые все вместе *мобильным протоколом IP,* были изначально определены в RFC 3220. Мобильный протокол IP представляет собой гибкий стандарт, поддерживающий различные режимы работы, например, он может функционировать как с внешним агентом, так и без него. Кроме того, мобильный протокол IP предоставляет агентам и мобильным узлам различные способы обнаружения друг друга, поддерживает использование единого внешнего адреса или нескольких внешних адресов, а также различные формы инкапсуляции. Таким образом, мобильный протокол IP представляет собой сложный стандарт, для подробного описания которого потребовалась бы отдельная книга. Здесь же наша скромная цель ограничивается предоставлением обзора наиболее важных аспектов организации и применения мобильного протокола IP.

Мобильная IP-архитектура подразумевает использование множества средств и методов, которые мы уже обсуждали выше, включая домашние и внешние агенты, внешние адреса, инкапсуляцию дейтаграмм и их извлечение. Текущий стандарт (RFC 3220) специфицирует три аспекта косвенной маршрутизации данных для мобильного узла.

- *Обнаружение агента.* Стандарт мобильного протокола IP определяет протоколы, используемые домашним и внешним агентами для объявления мобильным узлам о своих службах, а также протоколы, позволяющие мобильным узлам запрашивать эти службы у домашнего или внешнего агента.
- *Регистрация у домашнего агента.* Стандарт мобильного протокола IP определяет протоколы, используемые мобильным узлом и/или внешним агентом для регистрации и отмены регистрации внешних адресов у домашнего агента мобильного узла.
- *Косвенная маршрутизация дейтаграмм.* Стандарт мобильного протокола IP также определяет способ, которым дейтаграммы направляются мобильным узлам домашним агентом, включая правила продвижения дейтаграмм, правила обработки ошибок и несколько форм инкапсуляции (RFC 2003, RFC 2004).

В стандарте мобильного протокола IP отчетливо заметно внимание, уделяемое разработчиками вопросам безопасности. Например, необходима аутентификация мобильного узла, чтобы гарантировать, что злоумышленник не зарегистрирует у домашнего агента поддельный внешний адрес, в результате чего дейтаграммы, адресованные мобильному узлу, могут достаться злоумышленнику. Безопасность в стандарте мобильного протокола IP достигается при помощи различных механизмов, которые мы изучим в главе 7.

Обнаружение агента

Прибывающий в новую сеть мобильный IP-узел, либо подключаясь к новой внешней сети, либо возвращаясь в домашнюю сеть, должен узнать идентификатор внешнего или домашнего агента в этой сети. Именно обнаружив новый внешний агент с новым сетевым адресом, сетевой уровень мобильного узла понимает, что мобильный узел попал в новую внешнюю сеть. Этот процесс называется *обнаружением агента.* Обнаружение агента может осуществляться двумя способами: с помощью объявлений от агента или с помощью запроса мобильного узла об агенте.

В случае *объявлений от агента* внешний или домашний агент рекламирует свои службы при помощи расширения существующего протокола обнаружения маршрутизатора ICMP (RFC 1256). Агент периодически рассылает по всем линиям, к которым он подключен, широковещательные ICMP-сообщения с кодом 9 (обнаружение маршрутизатора) в поле типа сообщения. Это сообщение содержит IP-адрес маршрутизатора (то есть агента), что позволяет мобильному узлу узнать IP-адрес агента. В сообщении также содержится дополнительная информация, необходимая мобильному узлу и называемая рекламным расширением мобильного агента. Наиболее важные поля этого расширения перечислены ниже.

- *Бит домашнего агента (H)* указывает, что агент является домашним агентом для сети, в которой он находится.
- *Бит внешнего агента (F)* указывает, что агент является внешним агентом для сети, в которой он находится.
- *Биттребованиярегистрации (R)* указывает, что мобильный пользователь в данной сети обязан зарегистрироваться у внешнего агента. В частности, мобиль-

ный пользователь не может самостоятельно (не зарегистрировавшись у внешнего агента) выбрать себе внешний адрес (например, при помощи протокола DHCP) и взять на себя функции внешнего агента.

- *Биты инкапсуляции (Mи G)* указывают, будет ли использоваться иная форма инкапсуляции (отличная от варианта, когда IP-дейтаграмма располагается внутри 1P-дейтаграммы).
- *Поля внешнего адреса (COA).* Список из одного или нескольких внешних адресов предоставляется внешним агентом. В нашем примере (см. далее) внешний адрес ассоциируется с внешним агентом, который будет получать дейтаграммы, посылаемые по внешнему адресу, а затем переправлять их соответствующему мобильному узлу. Мобильный узел должен выбрать из этих адресов один в качестве своего внешнего адреса и зарегистрировать его у своего домашнего агента.

На рис. 4.58 показаны некоторые ключевые поля рекламного объявления агента.

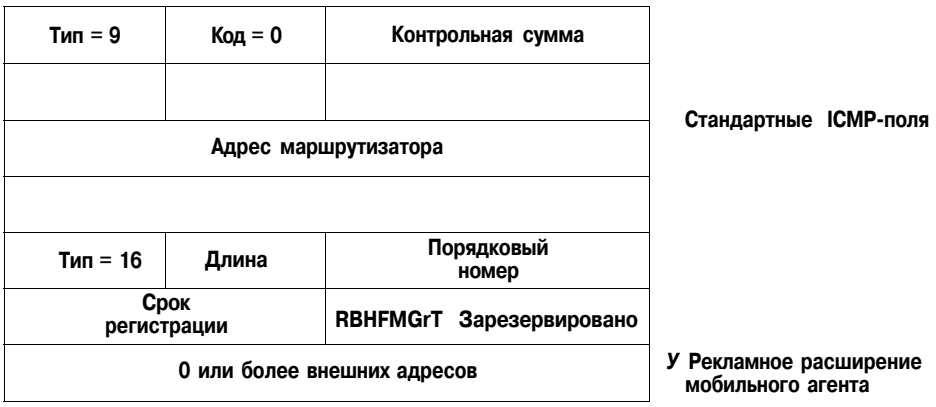


Рис. 4.58. ICMP-сообщение обнаружения маршрутизатора с рекламным расширением мобильного агента

В случае запроса мобильного узла об агенте мобильный узел, не дожидаясь получения рекламного объявления от агента, сам передает широковещательное сообщение с запросом об агенте, представляющее собой обычное ICMP-сообщение типа 10. Получив запрос, агент пересылает рекламное объявление напрямую мобильному узлу, последующие действия которого ничем не отличаются от случая получения им обычного широковещательного рекламного объявления от агента.

Регистрация у домашнего агента

Как только мобильный узел получает внешний адрес, он должен зарегистрировать этот адрес у своего домашнего агента. Это можно сделать через внешний

агент или напрямую. Мы рассмотрим первый вариант. Процедура состоит из четырех шагов.

1. Получив рекламное объявление от внешнего агента, мобильный узел посылает регистрационное сообщение внешнему агенту. Это сообщение переносится BUDP-дейтаграмме и адресуется в порт 434. В регистрационном сообщении указывается внешний адрес (COA), о котором объявил внешний агент, адрес домашнего агента (HA), постоянный адрес мобильного узла (MA), требуемый срок регистрации (в секундах) и 64-разрядный идентификатор регистрации. Если в течение указанного срока регистрация у домашнего агента не возобновляется, она считается недействительной. Идентификатор регистрации действует как порядковый номер и служит для сопоставления регистрационного запроса с ответом на этот запрос, о чем будет сказано ниже.
2. Внешний агент получает регистрационное сообщение и записывает постоянный IP-адрес мобильного узла. Теперь внешний агент знает, что он должен искать дейтаграммы, содержащие инкапсулированные дейтаграммы, адрес получателя которых совпадает с постоянным адресом мобильного узла. Затем внешний агент посылает регистрационное сообщение (также в UDP-дейтаграмме) в порт 434 домашнего агента. В сообщении содержатся временный адрес, адрес домашнего агента, постоянный адрес мобильного узла, запрашиваемый формат инкапсуляции, запрашиваемый срок регистрации и идентификатор регистрации.
3. Домашний агент получает запрос о регистрации и проверяет его подлинность и корректность. Домашний агент привязывает постоянный IP-адрес мобильного узла к внешнему адресу. Впоследствии поступающие к домашнему агенту дейтаграммы, предназначаемые мобильному узлу, будут инкапсулироваться и туннелироваться по внешнему адресу. Домашний агент посылает ответ на регистрационный запрос, содержащий адрес домашнего агента, постоянный адрес мобильного узла, фактический срок регистрации и идентификатор регистрации.
4. Внешний агент получает ответ на запрос о регистрации, а затем направляет его мобильному узлу.

К этому моменту процедура регистрации завершается, и мобильный узел теперь может получать дейтаграммы, посылаемые по его постоянному адресу. Описанные шаги показаны на рис. 4.59. Обратите внимание, что домашний агент указывает меньший срок регистрации, чем срок, запрашиваемый мобильным узлом.

Внешнему узлу не нужно явно отменять регистрацию внешнего адреса, если мобильный узел покидает его сеть. Это происходит автоматически, когда мобильный узел перемещается в новую сеть (либо другую внешнюю сеть) и регистрирует новый внешний адрес.

Стандарт мобильного протокола IP позволяет реализовывать множество дополнительных сценариев, помимо описанных выше. Если вас заинтересовала эта тема, дополнительную информацию можно получить в [380, 382].

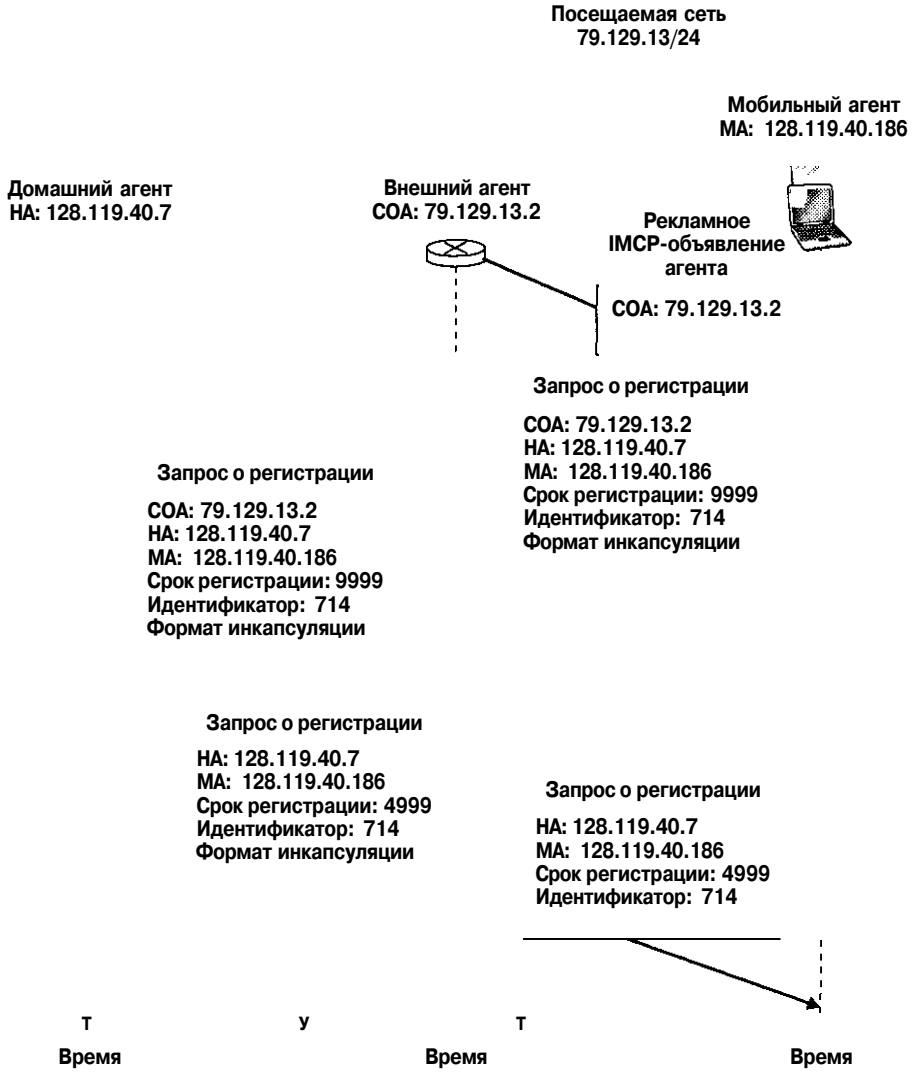


Рис. 4.59. Рекламное объявление агента и IP-регистрация мобильного агента

Резюме

В этой главе мы начали наше знакомство с ядром сети. Мы узнали, что на сетевом уровне работают абсолютно все хосты и маршрутизаторы сети. Поэтому сетевые протоколы являются одними из самых сложных в стеке протоколов.

Мы узнали, что одна из самых сложных задач сетевого уровня состоит в маршрутизации дейтаграмм через сеть, состоящую из миллионов хостов и маршрутизаторов. Эта масштабная задача решается путем разбиения больших сетей на независимые административные домены, называемые автономными системами. Каждая

автономная система независимо выбирает маршруты для проходящих через нее дейтаграмм, подобно тому как каждое государство независимо от других занимается доставкой почты в пределах своей территории. Для маршрутизации внутри автономных систем в Интернете сегодня применяются два популярных протокола внутренней маршрутизации: RIP и OSPF. Для перемещения пакетов между автономными системами необходим протокол внешней маршрутизации. На сегодня наибольшее распространение получил протокол внешней маршрутизации BGP4. Осуществление маршрутизации на двух уровнях (внутри каждой автономной системы и между автономными системами) называется иерархической маршрутизацией. Задача масштабирования в большой степени решается благодаря иерархической организации инфраструктуры маршрутизации. При разработке протоколов, особенно протоколов сетевого уровня, мы должны всегда помнить этот общий принцип: проблемы масштабирования часто можно решить путем их иерархической организации. Интересно отметить, что этот принцип на протяжении веков и тысячелетий применялся во многих других дисциплинах, в частности в коммерческих, государственных, религиозных и военных организациях.

В этой главе мы также познакомились с другой проблемой масштабирования. В больших компьютерных сетях маршрутизатору может потребоваться одновременно обрабатывать миллионы потоков пакетов между различными парами отправитель-получатель. За годы работы разработчики сетей пришли к важному выводу: чтобы маршрутизатор мог обрабатывать такое большое количество потоков, функции маршрутизатора должна быть максимально простыми. Для упрощения работы маршрутизатора доступны разные методы, включая использование вместо виртуальных каналов дейтаграмм на сетевом уровне, применение упрощенного заголовка фиксированной длины (как в протоколе IPv6), устранения фрагментации (что также сделано в IPv6), предоставление минимального обслуживания (обслуживание по остаточному принципу). Возможно, наиболее важный принцип здесь заключается в том, чтобы не отслеживать отдельные потоки, а принимать решения о выборе маршрутов исключительно на базе иерархически структурированных адресов получателей, указанных в пакетах. Интересно отметить, что почтовая служба применяла подобный принцип на протяжении многих лет.

В этой главе мы также рассмотрели основополагающие принципы алгоритмов маршрутизации. Мы узнали, что разработчики алгоритмов маршрутизации используют абстракцию компьютерной сети, представляя ее в виде графа с узлами и ребрами (линиями связи между узлами). Это позволяет задействовать хорошо развитую теорию графов и алгоритмы нахождения кратчайших маршрутов, разработанные за последние 40 лет. Мы увидели, что существует два основных подхода — централизованный подход, при котором каждый узел получает полную карту сети и независимо от других узлов применяет алгоритм определения кратчайшего маршрута, и децентрализованный подход, при котором отдельные узлы обладают только частичным представлением обо всей сети, однако, работая вместе, узлы доставляют пакеты по кратчайшим маршрутам. На протяжении многих лет алгоритмы маршрутизации в компьютерных сетях представляли собой область активных исследований, и эти исследования продолжаются в наши дни.

В конце главы мы рассмотрели три дополнительные темы, отражающие современные тенденции в компьютерных сетях и Интернете. Первой такой темой стал протокол IPv6, упрощающий сетевой уровень и решающий проблему адресного пространства протокола IPv4. Вторая тема связана с групповой маршрутизацией. Групповая маршрутизация позволяет значительно экономить пропускную способность сетей, а также ресурсов маршрутизаторов и серверов. Наконец, мы рассмотрели тему поддержки мобильных узлов на сетевом уровне. Здесь были представлены различные методы решения проблемы мобильности, а также специфический метод решения, реализованный в стандарте мобильного протокола IP. Будет интересно наблюдать, как в ближайшее десятилетие пойдет процесс развертывания протокола IPv6, протоколов групповой маршрутизации и мобильного протокола IP.

Итак, мы завершили наше изучение сетевого уровня и теперь готовы спуститься еще на один уровень ниже по стеку протоколов, а именно перейти к канальному уровню, также называемому уровнем передачи данных. Как и сетевой уровень, канальный уровень представляет собой часть ядра сети. Однако в следующей главе мы увидим, что канальный уровень решает значительно более узкую задачу — задачу перемещения пакетов между узлами, присоединенными к одной и той же линии или локальной сети. Хотя эта задача на первый взгляд может показаться тривиальной по сравнению с задачами сетевого уровня, мы увидим, что канальный уровень предлагает ряд важных и интересных вопросов, способных надолго занять ваше внимание.

Вопросы и задания для самостоятельной работы

1. Какие две основные функции выполняет сетевой уровень, основанный на дейтаграммах? Какие дополнительные функции выполняет сетевой уровень, основанный на виртуальных каналах?
2. Перечислите и опишите модели сетевого обслуживания ATM.
3. Сравните алгоритмы, основанные на состоянии линий, и дистанционно-векторные алгоритмы.
4. Каким образом иерархическая организация Интернета помогает решать задачи масштабирования, в частности поддерживать миллионы пользователей?
5. Должен ли в каждой автономной системе использоваться один и тот же алгоритм внутренней маршрутизации? Почему да или почему нет?
6. Как выглядит в двоичном виде IP-адрес 223.1.3.27?
7. Рассмотрите локальную сеть, к которой подключены десять интерфейсов хостов и три интерфейса маршрутизаторов. Пусть в локальной сети используются адреса класса C. Сколько первых битов IP-адресов тринадцати интерфейсов будут идентичными?
8. Представьте себе маршрутизатор с тремя интерфейсами. Пусть для всех трех интерфейсов используются адреса класса C. Обязательно ли первые 8 бит IP-адресов будут совпадать?

9. Пусть между хостом-отправителем и хостом-получателем имеются три маршрутизатора. Сколько интерфейсов преодолеет IP-сегмент, посланный хостом-отправителем хосту-получателю, если игнорировать фрагментацию? Сколько таблиц продвижения данных будут участвовать в перемещении дейтаграммы?
10. Предположим, приложение генерирует блоки данных по 40 байт через каждые 20 мс. Каждый блок данных помещается в TCP-сегмент, а затем в IP-дейтаграмму. Какой процент каждой дейтаграммы составят накладные расходы, а какой — данные приложения?
11. Рассмотрите отправку 3000-разрядной дейтаграммы по линии, максимальная единица передачи (MTU) которой равна 500 байт. Предположим, оригинальная дейтаграмма маркируется идентификатором 422. Сколько фрагментов будет создано? Опишите их характеристики.
12. Вернитесь к рис. 4.28. Начиная с оригинальной таблицы маршрутизатора D, предположим, что маршрутизатор D получает от маршрутизатора A уведомление (рекламное объявление), представленное в табл. 4.10. Изменится ли таблица маршрутизатора D? Если да, то каким образом?

Таблица 4.10. Типы сообщений протокола IGMP v2

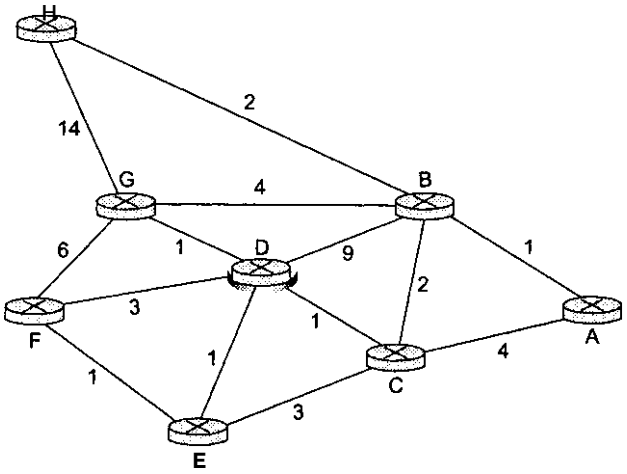
Сеть-адресат	Следующий маршрутизатор	Количество хопов до адресата
z	C	10
w	—	1
x	—	1

13. Сравните рекламные объявления, применяемые в протоколах RIP и OSPF.
14. Вставьте пропущенное слово: RIP-объявления, как правило, содержат количество ретрансляционных участков до различных получателей. В отличие от них, объявления протокола BGP информируют о_____до различных адресатов.
15. Почему в Интернете используются разные протоколы внутренней и внешней маршрутизации?
16. Опишите три типа коммутаторов, широко применяемых в качестве пакетных.
17. Зачем нужны буферы в выходных портах коммутатора? Зачем нужны буферы во входных портах коммутатора?
18. Сравните поля заголовков протоколов IPv4 и IPv6. Много ли у них общих полей?
19. Было сказано, что при туннелировании дейтаграмм протокола IPv6 через IPv4-маршрутизаторы протокол IPv6 обращается с IPv4-туннелями, как с протоколами канального уровня. Вы согласны с таким утверждением? Почему да или почему нет?
20. В чем принципиальная разница в реализации групповой рассылки при помощи большого числа выборочных (одноадресных) рассылок и при помощи группы рассылки, опирающейся на поддержку сети (маршрутизаторов).

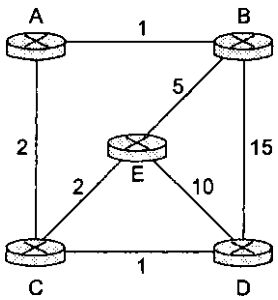
21. Ответьте, правильно ли следующее утверждение. Когда хост присоединяется к группе рассылки, он должен изменить свой IP-адрес на адрес этой группы рассылки.
22. Какие роли играют межсетевой протокол управления группами и глобальный протокол групповой маршрутизации?
23. В чем применительно к групповой маршрутизации разница между общим для группы деревом и деревом с вершиной в источнике?
24. Ответьте, правильны ли следующие утверждения.
    24.1. При продвижении данных по обратному пути узел получит несколько копий одного и того же пакета.
    24.2. При продвижении данных по обратному пути узел может отправить несколько копий одного и того же пакета по одной и той же линии.
25. К какому из двух алгоритмов групповой маршрутизации (основанном на дереве с основанием у отправителя и основанном на дереве, общем для группы) относятся следующие алгоритмы: DVMRP, MOSPF, CBT, алгоритм разреженного режима протокола PIM, алгоритм плотного режима протокола PIM.

Упражнения

1. Рассмотрим некоторые аргументы в пользу архитектуры, ориентированной на установление соединения, и архитектуры, не требующей установления соединения.
    1.1. Предположим, что на сетевом уровне маршрутизаторы подвержены «стрессу», в результате которого они довольно часто выходят из строя. Какие действия следует предпринимать на верхнем уровне при выходе маршрутизатора из строя? Является ли это аргументом в пользу окружения, ориентированного на установление соединения, или окружения, не требующего установления соединений?
    1.2. Предположим, что для предоставления гарантированного уровня производительности (например, задержки) на маршруте от отправителя до получателя сеть требует, чтобы отправитель объявлял пиковую скорость своего трафика. Если объявленная пиковая скорость трафика и существующие объявленные скорости трафика таковы, что сеть не может удовлетворить требования отправителя, отправитель не получит доступа к сети. При использовании какой парадигмы легче реализовать данный принцип: ориентированной на установление соединения или не требующей установления соединения?
2. Перечислите маршруты от узлов A и F (см. рис. 4.4), не содержащие петель.
3. Рассмотрим следующую сеть. С помощью алгоритма поиска кратчайшего пути Дейкстры рассчитайте кратчайший маршрут от узла F до всех остальных узлов сети на основе указанных стоимостей линий. Покажите, как работает алгоритм, построив таблицу, аналогичную табл. 4.2.



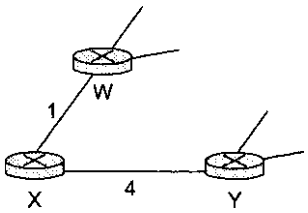
4. Рассмотрите сеть из упражнения 3. С помощью алгоритма пути Дейкстры и составленной таблицы найдите кратчайший маршрут.
   4.1. От узла A до всех узлов сети.
   4.2. От узла B до всех узлов сети.
   4.3. От узла C до всех узлов сети.
   4.4. От узла D до всех узлов сети.
   4.5. От узла E до всех узлов сети.
   4.6. От узла F до всех узлов сети.
   4.7. От узла G до всех узлов сети.
   4.8. От узла H до всех узлов сети.
5. Обратите внимание на показанную ниже сеть. Пусть каждому узлу изначально известны стоимости линий до каждого соседа. Рассмотрите дистанционно-векторный алгоритм и сформируйте таблицу расстояний для узла E.



6. Рассмотрите общую (то есть не относящуюся к показанной выше сети) топологию и синхронную версию дистанционно-векторного алгоритма. Пусть на каждой итерации узел обменивается со своими соседями сведениями о мини-

мальных стоимостях. Предположим, что в начале работы алгоритма каждому узлу известны только стоимости его непосредственных соседей. Какое максимальное количество итераций необходимо для схождения алгоритма? Аргументируйте свой ответ.

7. Рассмотрим показанный ниже фрагмент сети. У узла X есть только два соседа, W и Y. У узла W есть маршрут минимальной стоимости до узла A стоимости 6. Полные маршруты от узла W и от узла Y до узла A (а также между узлами W и Y) неизвестны. Стоимости всех линий сети представляют собой положительные целые числа.



   7.1. Покажите таблицу (ряд) расстояний узла X для адресатов W, Y и A.

   7.2. Приведите пример изменения стоимости линии — либо c(X, W), либо c(X, Y) — такого, что узел X в результате выполнения строк 15 и 24 дистанционно-векторного алгоритма проинформирует своих соседей о новом пути наименьшей стоимости до узла A.

   7.3. Приведите пример изменения стоимости линии — либо c(X, W), либо c(X, Y) — такого, что узел X в результате выполнения строк 15 и 24 дистанционно-векторного алгоритма не проинформирует своих соседей о новом пути наименьшей стоимости до узла A.

8. Рассчитайте таблицы расстояний для узлов X, Y и Z, показанных в самой правой колонке на рис. 4.7. Какие узлы будут посылать обновленные значения своим соседям после вычисления таблицы?

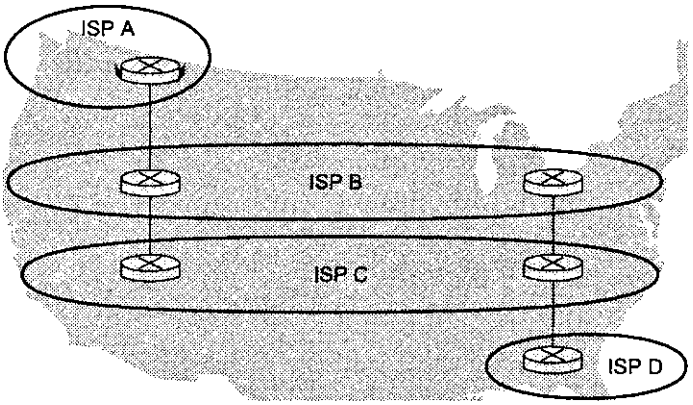
9. Рассмотрим сеть из трех узлов, показанную на рис. 4.7. Пусть стоимости линий этой сети равны: c(X, Y) = 5, c(Y, Z) = 6, c(Z, X) = 2. Составьте таблицы расстояний после этапа инициализации, а также на каждой итерации синхронной версии дистанционно-векторного алгоритма (как мы это делали ранее при обсуждении рис. 4.7).

10. Начертите диаграмму конечных состояний для стороны клиента протокола DHCP, обсуждавшегося в разделе «Интернет-протокол». Вспомним, что протокол DHCP использует протокол UDP, предоставляющий ненадежную транспортную службу для DHCP-сообщений. Убедитесь, что ваша машина конечных состояний показывает, как интервалы ожидания и повторные передачи обеспечивают надежность.

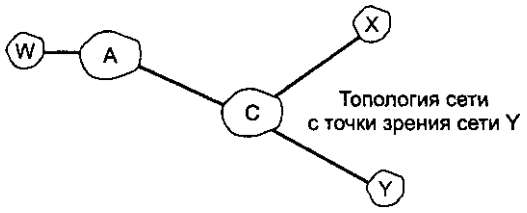
11. Опишите, как протокол BGP обнаруживает петли в маршрутах.

12. Рассмотрите показанную ниже сеть. Интернет-провайдер B предоставляет доступ к национальной магистрали региональному провайдеру A. Интернет-

провайдер C предоставляет доступ к национальной магистрали региональному провайдеру D. Интернет-провайдеры B и C соединяются друг с другом в двух точках с помощью протокола BGP. Рассмотрим трафик, идущий от A к D. Интернет-провайдер B предпочел бы передать этот трафик Интернет-провайдеру C на Западном побережье (чтобы через весь континент трафик нес Интернет-провайдер C), тогда как Интернет-провайдер C хотел бы получить этот трафик от Интернет-провайдера B на Восточном побережье (чтобы бремя транспортировки данных через континент легло на плечи Интернет-провайдера B). Какой механизм протокола BGP может использовать Интернет-провайдер C, чтобы Интернет-провайдер B передавал ему данные, посылаемые от A к D на Восточном побережье? Для ответа на этот вопрос вам придется порыться в спецификации протокола BGP.



13. Рассмотрим информацию, попадающую в тупиковые сети W, X и Y на рис. 4.32. Какова топология сети с точки зрения сетей W и X, основанная на доступной им информации? Аргументируйте свой ответ. Топология сети с точки зрения сети Y показана ниже.



14. Рассмотрите сеть с восемью узлами (помеченными от A до H) в упражнении 3. Покажите дерево минимальной стоимости с вершиной в узле A, включающее (в качестве оконечных хостов) узлы C, D, E и G. Аргументируйте, почему ваше дерево является деревом наименьшей стоимости.

15. В разделе «Групповая маршрутизация» было показано, что не существует сетевого протокола, который мог бы использоваться для идентификации хостов, входящих в группу рассылки. Как приложение, использующее групповую рассылку, учитывая это, может узнать идентификаторы хостов, входящих в группу рассылки?

16. Рассмотрим два основных метода групповой рассылки: одноадресную эмуляцию групповой рассылки и собственно групповую рассылку на сетевом уровне. Рассмотрим случай одного отправителя и 32 получателей. Пусть отправитель соединен с получателями двоичным деревом маршрутизаторов. Какова для этой топологии стоимость пересылки группового пакета в случае одноадресной эмуляции и групповой рассылки на сетевом уровне? При каждой пересылке пакета (или копии пакета) по линии расходуется одна единица стоимости. При какой топологии сети, соединяющей отправителя, получателей и маршрутизаторы, стоимости одноадресной эмуляции и реализованной на сетевом уровне групповой рассылки будут максимально отличаться? Количество маршрутизаторов в сети может быть любым.

17. Разработайте (дайте описание на псевдокоде) протокол прикладного уровня, управляющий адресами всех хостов в группе рассылки. Идентифицируйте сетевую службу (выборочная рассылка или групповая рассылка), которой пользуется ваш протокол, а также укажите, как ваш протокол пересылает сообщения (внутри полосы или вне полосы относительно потока данных приложения, использующего групповую рассылку) и почему.

18. Рассмотрим топологию сети, показанной на рис. 4.49. Пусть стоимость линии от узла B до узла D изменилась с 1 до 10. Найдите дерево Штайнера, соединяющее все скрытые маршрутизаторы. (Вам не нужно писать программу решения проблемы Штайнера. Вместо этого вы должны построить дерево минимальной стоимости, исследовав сеть и убедившись, что оно является деревом минимальной стоимости.) Как бы вы доказали (в действительности от вас этого не требуется), что дерево на самом деле представляет собой дерево минимальной стоимости?

19. Вернемся к теме маршрутизации с использованием центрального узла. Рассмотрите топологию сети, показанной на рис. 4.49. Пусть в качестве центрального узла выбран узел C. Предполагая, что каждый присоединенный узел в группе рассылки использует свой маршрут наименьшей стоимости до узла C, начертите получающееся в результате дерево групповой маршрутизации. Является ли получившееся в результате дерево деревом Штайнера минимальной стоимости? Аргументируйте свой ответ.

20. Вернемся к теме групповой маршрутизации по одноадресным маршрутам минимальной стоимости. Рассмотрим рис. 4.49. Пусть отправителем является узел E. Рассчитайте дерево групповой маршрутизации, состоящее из одноадресных маршрутов минимальной стоимости от узла E до групповых маршрутизаторов A, B и F.

21. Вернемся к теме продвижения данных по обратным маршрутам. Рассмотрим топологию и стоимости линий сети, показанной на рис. 4.49. Пусть узел E яв-

ляется источником групповой рассылки. Стрелками (как на рис. 4.51) покажите линии, по которым пакеты переправляются при помощи протокола RPF, а также линии, по которым пакеты не будут переправляться.

22. Пусть стоимость передачи группового пакета по линии совершенно не зависит от стоимости передачи по этой же линии одноадресного пакета. Будет ли метод продвижения данных по обратным маршрутам продолжать работать в этом случае? Аргументируйте свой ответ.

23. Вернемся к теме трафика в деревьях с центральными узлами. Рассмотрим простую топологию сети, показанной на рис. 4.49. Пусть каждый групповой маршрутизатор получает от присоединенного к нему хоста за одну единицу времени одну единицу трафика. Этот трафик нужно переправить остальным трем групповым маршрутизаторам. Пусть в качестве центрального узла выбран узел C (см. упражнение 19). Постройте дерево маршрутизации и вычислите скорость трафика по каждой линии сети (рассчитайте суммарный трафик в каждой линии независимо от направления потока данных). Затем предположите, что с помощью протокола RPF для маршрутизаторов A, B, E и F строятся четыре дерева с вершиной у отправителя. Повторите расчет трафика для всех линий во втором сценарии. В каком случае трафик будет сильнее?

24. Пусть в сети есть G групп рассылки и пусть в каждую из них входит S членов (хостов), каждый из которых может быть отправителем. Под управлением протокола DVMPR каждый маршрутизатор должен управлять S порциями информации о маршрутах (исходящих линий по кратчайшим обратным путям к отправителю для каждого из S отправителей) для каждой группы. Таким образом, в худшем случае каждый маршрутизатор должен обрабатывать до S x G порций информации о маршрутах. С каким количеством информации придется иметь дело маршрутизатору в худшем случае при использовании протокола MOSPF, а также протокола PIM в обоих его режимах работы (плотном и разреженном)? Аргументируйте свой ответ.

25. Каков размер адресного пространства групповой рассылки? Предположим, что две различные группы рассылки одновременно выбирают групповые адреса случайным образом. Какова вероятность того, что они выберут один и тот же адрес? Предположим теперь, что одновременно случайным выбором адресов занимаются 1000 групп. С какой вероятностью их интересы пересекутся?

26. Вспомним, что при обсуждении применяемого при групповой рассылке туннелирования говорилось, что групповая IP-дейтаграмма переносится внутри одноадресной IP-дейтаграммы. Как IP-маршрутизатор в конце туннеля узнает, что в одноадресной IP-дейтаграмме содержится многоадресная IP-дейтаграмма?

27. В подразделе «Мобильный протокол IP» раздела «Мобильность и сетевой уровень» одно из предложенных решений поддержки мобильных IP-узлов во время их перемещения по внешним сетям заключалось в том, что внешняя сеть объявляла маршрут к мобильному пользователю и распространяла эту информацию по сети при помощи существующей инфраструктуры маршрутизации. Мы определили масштабирование как одну из проблем подобного подхода.

Предположим, что когда мобильный пользователь перемещается из одной сети в другую, новая внешняя сеть объявляет о своем маршруте к мобильному пользователю, а прежняя внешняя сеть удаляет свой маршрут. Рассмотрим, как информация о маршрутах распространяется по сети в случае дистанционно-векторного алгоритма (например, для междоменной маршрутизации в сетях, охватывающих всю планету).

27.1. Смогут ли другие маршрутизаторы направлять дейтаграммы непосредственно в новую внешнюю сеть, как только внешняя сеть начнет рекламировать свой маршрут?

27.2. Может ли случиться так, что разные маршрутизаторы будут считать, что мобильный пользователь находится в разных сетях?

27.3. Сколько потребуется времени, чтобы другие маршрутизаторы сети в конце концов узнали маршрут к мобильным пользователям?

28. Предположим, что корреспондент на рис. 4.54 является мобильным. Нарисуйте дополнительную инфраструктуру сетевого уровня, необходимую для маршрутизации дейтаграммы, посылаемой оригинальным мобильным пользователем корреспонденту (ныне мобильному). Покажите структуру дейтаграмм, посылаемых оригинальным мобильным пользователем корреспонденту (мобильному), как это сделано на рис. 4.55.

29. Какой эффект оказывает мобильность в стандарте мобильного протокола IP на сквозные задержки при передаче дейтаграмм от отправителя к получателю?

30. Рассмотрите пример цепи, обсуждавшийся в подразделе «Управление мобильной связью» раздела «Мобильность и сетевой уровень». Предположим, мобильный пользователь посещает внешние сети A, B и C, а корреспондент начинает установку соединения с мобильным пользователем, когда тот находится во внешней сети A. Перечислите последовательность сообщений между внешними агентами, а также между внешними агентами и домашним агентом во время перемещения мобильного узла от сети A в сеть B и в сеть C. Затем предположите, что цепь не образуется, и корреспондент (так же как и домашний агент) должен явно уведомляться об изменениях внешнего адреса мобильного хоста. Перечислите последовательность сообщений, которые понадобятся при таком сценарии.

31. Рассмотрите два мобильных узла во внешней сети с внешним агентом. Могут ли два мобильных узла использовать один и тот же внешний адрес? Аргументируйте свой ответ.

Дополнительные вопросы и задания

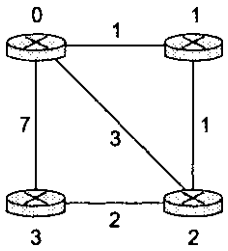
1. Предположим, автономные системы X и Z не имеют прямого соединения, а соединяются через автономную систему Y. Предположим также, что у автономной системы X есть соглашение об одноранговом информационном обмене с автономной системой Y, а у автономной системы Y есть соглашение об одноранговом информационном обмене с автономной системой Z. Наконец, предположим, что автономная система Z хочет переносить весь трафик авто-

номной системы Y, но не желает переносить трафик автономной системы X. Позволяет ли протокол BGP реализовать данную политику?

2. В разделе «Протокол IPv6» было показано, что развертывание протокола IPv6 идет медленными темпами. Почему? Что нужно для ускорения этого процесса?
3. В подразделе «Групповая рассылка в Интернете и группы рассылки» раздела «Групповая маршрутизация» было показано, что групповая рассылка может быть эмулирована путем установки индивидуальных соединений между отправителем и каждым из получателей. Каковы недостатки этого подхода по сравнению с методом, основанным на поддержке групповой рассылки на сетевом уровне? Какими преимуществами обладает данный метод?
4. В разделе «Групповая маршрутизация» упоминались приложения, использующие групповую рассылку. Какие из этих приложений хорошо подходят к минималистской модели обслуживания Интернета, основанной на групповой рассылке? Почему? Какие приложения не очень хорошо подходят к этой модели обслуживания?
5. Применительно к механизму неустойчивого состояния протокола CBT, используемого для управления деревом, почему, как вы думаете, требуется отдельное сообщение FLUSH_TREE? Что произойдет, если сообщение FLUSH_TREE потеряется?

Задание по программированию

В этом задании вам предлагается написать «распределенный» набор процедур, реализующих распределенный асинхронный алгоритм дистанционно-векторной маршрутизации для сети, изображенной ниже.



Вы должны написать для узла 0 следующие процедуры, которые будут «выполняться» асинхронно в эмулированном окружении, предоставляемом для данного задания.

- rtinitO(). Эта процедура будет вызываться один раз в начале эмуляции. У нее нет аргументов. Она должна инициализировать вашу таблицу расстояний в узле 0, помещая в нее значения стоимостей 1,3 и 7 для узлов 1,2 и 3 соответственно. В показанной на рисунке сети все линии являются двунаправленными, и стоимости линий идентичны в обоих направлениях. После инициализации таб-

лицы расстояний, а также других структур данных, необходимых процедурам узла 0, она должна посылать соседям узла 0 (то есть узлам 1, 2 и 3) значения стоимостей маршрутов наименьшей стоимости до всех остальных узлов сети. Эта информация посылается соседним узлам в пакете обновлений при вызове процедуры tolayer2(). Форматы этой процедуры и пакета обновления описаны в полном задании на сайте, адрес которого указан ниже.

- rtupdateO(struct rtpkt *rcvdpkt). Эта процедура вызывается, когда узел 0 получает пакет с информацией о маршрутах, отправленный одним из своих соседей. Параметр *rcvdpkt представляет собой указатель на полученный пакет. Процедура rtupdateO() является «сердцем» дистанционно-векторного алгоритма. Получаемый ею от некоего узла *г* пакет обновлений содержит текущие значения стоимостей кратчайших маршрутов от узла *г* до всех остальных узлов сети. С помощью этой информации процедура rtupdateO() обновляет свою таблицу расстояний (как требует дистанционно-векторный алгоритм). Если в результате обновления изменяется стоимость маршрута от узла 0 до какого-либо узла сети, узел 0 информирует об этом своих непосредственных соседей, посылая им пакет. Вспомним, что в дистанционно-векторном алгоритме пакетами с информацией о маршрутах обмениваются только соседние узлы. Таким образом, узлы 1 и 2 будут обмениваться пакетами, а узлы 1 и 3 — нет.

Аналогичные процедуры определены для узлов 1, 2 и 3. Таким образом, всего вы напишете восемь процедур: rtimtO(), rtinitl(), rtinit2(), rtinit3(), rtupdateO(), rtupdatel(), rtupdate2() и rtupdate3(). Вместе эти процедуры реализуют распределенные асинхронные вычисления таблиц расстояний для топологии и значений стоимостей, показанных на рисунке.

Дополнительные детали программного задания, а также программу на языке С, которая вам понадобится для эмуляции аппаратно-программной среды, можно найти на сайте http://www.awl.com/kurose-ross. Там также имеется версия этого задания на языке Java.

Интервью

Винтон Г. Серф является старшим вице-президентом отдела архитектуры и технологии компании WorldCom. Он широко известен как один из разработчиков протоколов TCP/IP и архитектуры Интернета. В качестве вице-президента компании MCI Digital Information Services с 1982 по 1986 год он руководил разработкой проекта MCI Mail, первой коммерческой службы электронной почты, которая должна была соединяться с Интернетом. За время своей работы в управлении перспективных исследовательских программ DARPA (Defense Advanced Research Projects Agency) при Министерстве обороны США с 1976 по 1982 год он играл ключевую роль в руководстве разработками для Интернета и связанными с этим технологиями обработки пакетов и безопасности. Винтон Г. Серф получил степень бакалавра наук в университете Станфорда, а степень доктора философии в области кибернетики в Калифорнийском университете в Лос-Анджелесе.

- Почему вы решили специализироваться в области компьютерных сетей?

  В конце 60-х я работал программистом в Калифорнийском университете в Лос-Анджелесе. Моя работа была поддержана у правлением перспективных исследовательских программ DARPA при Министерстве обороны США (тогда оно называлось ARPA). Я состоял в штате в лаборатории профессора Леонарда Кляйнрока в центре сетевых измерений недавно созданной сети ARPANET. Первый узел сети ARPANET был установлен в Калифорнийском университете в Лос-Анджелесе 1 сентября 1969 года. Я отвечал за программирование компьютера, использовавшегося для определения производительности ARPANET и для передачи этой информации, которая сравнивалась с математическими моделями и прогнозами по производительности сети.

  Я и еще несколько аспирантов отвечали за разработку так называемых «протоколов уровня хостов» сети ARPANET — процедур и форматов, обеспечивающих взаимодействие в сети компьютеров различных типов. Это было захватывающее изучение нового (для меня) мира распределенных вычислений и средств связи.

- Предполагали ли вы, разрабатывая протокол IP, что он получит сегодня такое широкое распространение?

  Когда мы с Бобом Каном работали над этим протоколом в 1973 году, я думаю, в первую очередь, нас интересовал центральный вопрос: как заставить разнородные сети взаимодействовать друг с другом,, притом что сами сети мы изменить не можем.. Мы надеялись, что сможем найти способ, позволяющий прозрачным образом соединять произвольное множество сетей, так чтобы хосты могли общаться друг с другом без каких-либо дополнительных преобразований данных. Я полагаю, мы понимали, что имеем дело с мощной и расширяемой технологией, но я сомневаюсь, что мы четко представляли себе, как будет выглядеть мир, в котором 100 ООО ООО компьютеров соединяются через Интернет.

- Каким теперь вам представляется будущее сетей и Интернета? Какие основные препятствия предвидите вы на пути их развития?

  Я полагаю, что и сам Интернет, и сети вообще продолжат свой рост. Уже сейчас ясно, что в будущем появятся миллиарды Интернет-устройств, включая сотовые телефоны, холодильники, карманные компьютеры, домашние серверы, телевизоры, лэптопы, обычные серверы и т. д. К основным сложностям относятся поддержка мобильности, время жизни батарей, емкость линий доступа к сети, а также возможность неограниченного увеличения оптического ядра сети. Проектирование межпланетного расширения Интернета представляет собой проект, в котором я занят в лаборатории реактивного движения. Нам предстоит перейти с протокола IPv4 (с 32-разрядными адресами) на протокол IPv6 (128 бит). Дел предстоит немало!

- Кто помог вам достичь успеха в вашей профессиональной деятельности?

  Мой коллега Боб Кан; руководитель моего диссертационного проекта Джеральд Эстрин; мой лучший друг Стив Крокер (мы познакомились в средней школе, ионе 1960 году ввел меня в мир компьютеров!); а также тысячи инженеров, продолжающих развивать Интернет сегодня.

- Что вы можете посоветовать студентам, начинающим изучение сетей и Интернета?

  Не ограничивайте свою фантазию рамками существующих систем — думайте о том, что еще только может стать возможным, а затем взваливайте на себя бремя превращения сказки в быль. Полдюжины моих коллег ияв лаборатории реактивного движения работаем над проектом межпланетного расширения земного Интернета. Реализация этой идеи может занять десятки лет, а как же иначе?

  «Зачем нам нужны небеса, если их не достать рукой?»

ГЛАВА 5 Канальный уровень и локальные сети

В предыдущей главе мы говорили о том, что канальный уровень поддерживает связь между двумя хостами. Как показано на рис. 5.1, эту связь обеспечивают линии связи, начинающиеся от хоста-источника, проходящие через несколько маршрутизаторов и заканчивающиеся у хоста-получателя. В этой главе мы продолжим продвигаться вниз по стеку протоколов и перейдем от сетевого уровня к канальному, или, как его еще называют, уровню передачи данных. Нас будут интересовать следующие вопросы. Как передаются пакеты по образующим канал индивидуальным линиям связи? Как дейтаграммы сетевого уровня инкапсулируются в кадры канального уровня для передачи по отдельной линии связи? Могут ли протоколы канального уровня обеспечить надежную передачу данных от одного маршрутизатора до другого? Могут ли в различных линиях связи одного канала связи использоваться разные протоколы канального уровня? В этой главе мы ответим на эти, а также другие важные вопросы.

Во время обсуждения канального уровня мы обнаружим, что на канальном уровне существуют два принципиально разных типа каналов. Первый тип — это широковещательные каналы, распространенные в локальных, беспроводных локальных и спутниковых сетях, а также в гибридных оптоволоконных сетях. В широковещательном канале несколько хостов присоединены к одному и тому же каналу связи, а для того чтобы координировать передачу и предотвращать коллизии, требуется протокол доступа к несущей. Второй тип канала — это обычная двухточечная линия связи, соединяющая, например, два маршрутизатора или модем и маршрутизатор Интернет-провайдера. Управление доступом к несущей в подобном двухточечном соединении является тривиальным, тем не менее такие вопросы, как формирование кадра, надежная передача данных, обнаружение ошибок и управление потоком, сохраняют свою актуальность.

В этой главе мы также ознакомимся с несколькими важными технологиями канального уровня. Мы подробно рассмотрим стандарт Ethernet, представляющий собой на сегодняшний день наиболее распространенную технологию локальных

сетей. Кроме того, мы обсудим беспроводные локальные сети, уделив особое внимание популярному стандарту IEEE 802.11, и мы ознакомимся с протоколом PPP, применяемым для соединения хостов с сетью при помощи модема, а также с протоколом ATM, часто используемым Интернет-провайдерами верхнего звена.

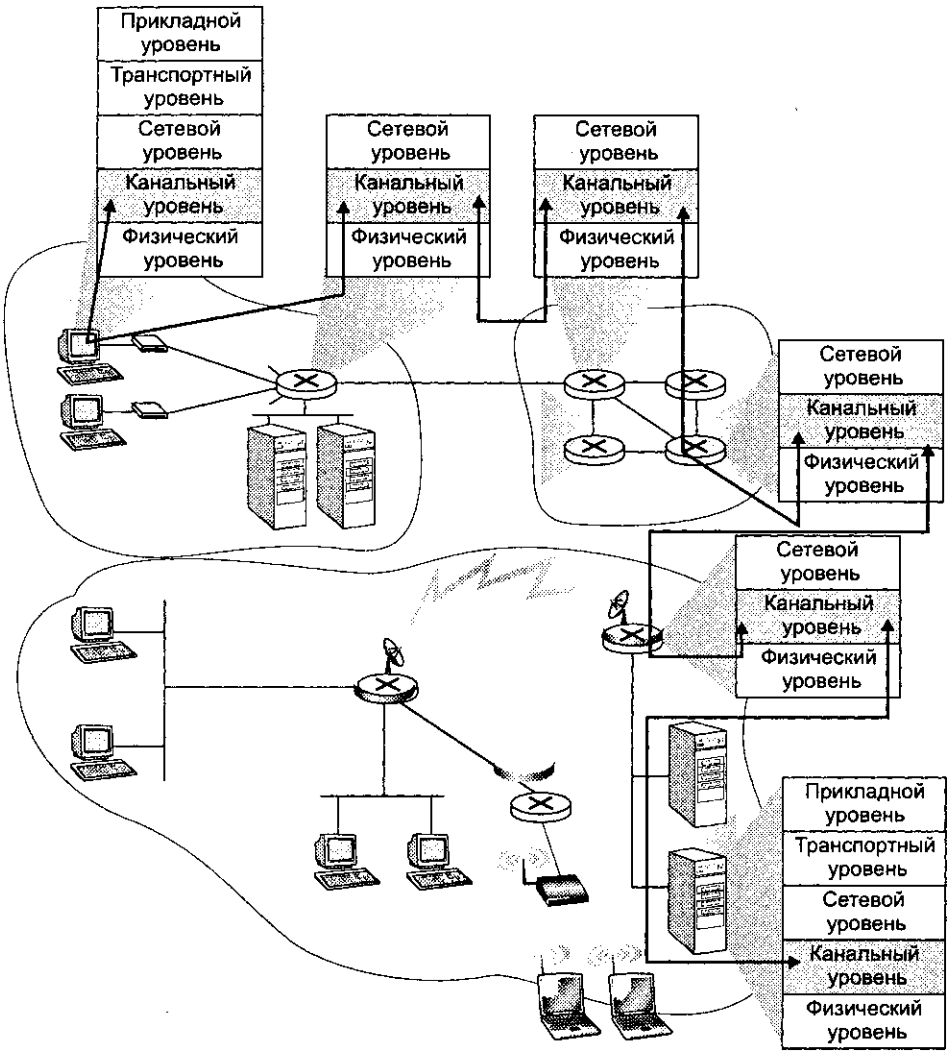


Рис. 5.1. Канальный уровень

Введение и терминология

Начнем изучение канального уровня с терминологии. В данной главе мы будем называть хосты и маршрутизаторы просто *узлами,* поскольку, как мы скоро уви-

дим, на канальном уровне не так уж и важно, чем является узел, маршрутизатором или хостом. Соединяющие соседние узлы каналы связи мы будем называть *линиями связи*. Таким образом, при перемещении от отправителя к получателю дейтаграмма должна пройти по каждой *отдельной линии связи,* входящей в путь связи. На каждой линии связи передающий узел упаковывает дейтаграмму в кадр канального уровня и передает этот кадр в линию, а получающий узел принимает кадр и извлекает из него дейтаграмму.

Службы канального уровня

Протокол канального уровня используется для перемещения дейтаграммы по индивидуальной линии связи. Этот протокол определяет формат пакетов, которыми обмениваются узлы на концах линии связи, а также действия, предпринимаемые этими узлами при отправке и получении пакетов. Как упоминалось в главе 1, эти пакеты, представляющие собой единицы обмена (PDU) протокола канального уровня, называют *кадрами*. Как правило, каждый кадр канального уровня содержит одну дейтаграмму сетевого уровня. Мы вскоре увидим, что протокол канального уровня при отправке и приеме кадров обеспечивает обнаружение ошибок, повторную передачу, управление потоком и произвольный доступ. В качестве примеров технологий канального уровня можно назвать Ethernet, 802.11, беспроводные локальные сети, маркерное кольцо и протокол PPP. Кроме того, во многих ситуациях протокол ATM также можно рассматривать как технологию канального уровня. Мы подробно обсудим эти технологии во второй половине этой главы.

В то время как сетевой уровень занимается перемещением сегментов транспортного уровня по всей длине пути от хоста-отправителя до хоста-получателя, протокол канального уровня выполняет работу по перемещению дейтаграмм сетевого уровня по отдельным линиям связи на этом пути. Для канального уровня важно, что передачей дейтаграммы по первой линии может управлять, например, протокол Ethernet, по последней линии — протокол PPP, а по всем промежуточным линиям — протокол Frame Relay. Следует отметить, что службы, поддерживаемые различными протоколами канального уровня, также могут различаться. Например, протокол канального уровня может обеспечивать или не обеспечивать надежную доставку. Таким образом, протокол сетевого уровня должен суметь выполнить свою задачу по сквозной доставке дейтаграмм несмотря на «разношерстный» набор служб канального уровня на отдельных линиях связи.

Чтобы лучше понимать, как работает канальный уровень и как он взаимодействует с сетевым уровнем, рассмотрим аналогию с транспортом. Представьте себе агента бюро путешествий, планирующего тур из Принстона, штат Нью-Джерси, в Лозанну, Швейцария. Предположим, тур-агент решает, что туристам удобнее всего отправиться из Принстона в аэропорт Кеннеди на лимузине, откуда на самолете перелететь в Женеву и, наконец, добраться до Лозанны на поезде. Тур-оператор делает три заказа, после чего за доставку туриста из Принстона в аэропорт отвечает Принстонская лимузинная компания; за доставку туриста из аэропорта JFK в Женеву ответственность несет авиакомпания; а за доставку туриста из Женевы в Лозанну — Швейцарская железнодорожная служба. Каждый из трех сегментов маршру-

та «напрямую» связывает два «соседних» узла. Обратите внимание, что тремя сегментами маршрута управляют три разные компании, и на этих сегментах используются совершенно разные виды транспорта (лимузин, самолет и поезд). Несмотря на это, каждый сегмент предоставляет основную службу, обеспечивающую перемещение пассажиров в соседний узел. В данной аналогии турист соответствует дейтаграмме, каждый сегмент — линии связи, вид транспорта — протоколу канального уровня, а тур-агент, планирующий весь маршрут, — протоколу маршрутизации.

Хотя основная услуга любого канального уровня заключается в «перемещении» дейтаграммы между двумя узлами по одной линии связи, полный перечень услуг зависит от конкретного протокола канального уровня, используемого на данной линии связи.

- *Формирование кадра.* Почти все протоколы канального уровня помещают каждую дейтаграмму сетевого уровня в кадр канального уровня, перед тем как отправить ее в линию связи. Кадр состоит из поля данных, в которое помещается дейтаграмма сетевого уровня, и нескольких полей заголовка (кадр также может включать поля концевика; но мы будем те и другие называть полями заголовка). Структура кадра специфицируется протоколом передачи данных. Во время обсуждения конкретных протоколов канального уровня во второй половине этой главы мы рассмотрим несколько различных форматов кадров. В разделе «Адресация в локальных сетях и протокол ARP» будет показано, что заголовки кадров также часто включают поля для так называемого *физического адреса* узла, не имеющего ничего общего с адресом узла сетевого уровня (например, IP-адресом).
- *Доступ к линии связи.* Протокол MAC (Media Access Control — управление доступом к носителю) определяет правила передачи кадра в линию. Для двухточечных линий с единственным отправителем на одном конце и единственным получателем на другом конце линии протокол MAC очень прост (или вообще отсутствует) — отправитель может передать кадр в любой момент, когда линия свободна. Более интересный случай представляет конфигурация, в которой несколько узлов совместно используют один широковещательный канал. В этом случае возникает так называемая проблема коллективного доступа, и протокол MAC призван координировать передачу кадров многих узлов. Подробно данный вопрос будет рассмотрен в разделе «Протоколы коллективного доступа».
- *Надежная доставка.* Когда протокол канального уровня предоставляет услугу по надежной доставке, он гарантирует перемещение каждой дейтаграммы сетевого уровня по линии связи без ошибок. Вспомним, что некоторые протоколы транспортного уровня (как, например, TCP) также обеспечивают надежную доставку. Аналогично службе надежной доставки транспортного уровня, служба надежной доставки канального уровня поддерживается с помощью механизмов подтверждений и повторных передач (см. раздел «Принципы надежной передачи данных»). Служба надежной доставки транспортного уровня часто обслуживает линии связи с высокой вероятностью ошибок, характерной, например, для беспроводных линий связи. Таким образом, на канальном уровне

ошибки исправляются локально — на той линии связи, на которой они возникают, что позволяет отказаться от повторной передачи данных протоколом транспортного или прикладного уровня. Однако в линиях с низкой вероятностью ошибок надежная доставка на канальном уровне может оказаться излишней. К таким линиям относятся волоконно-оптические и экранированные кабели, а также различные категории линий типа «витая пара». Поэтому многие протоколы для кабельных линий не предоставляют услуги по надежной доставке.

- *Управление потоком.* Узлы на каждой стороне линии связи обладают буферами для хранения кадров ограниченного размера. Это порождает потенциальную проблему, так как кадры могут поступать на получающий узел быстрее, чем этот узел способен их обрабатывать. Без управления потоком буфер получателя может переполниться, а кадры будут потеряны. Аналогично транспортному уровню протокол канального уровня может обеспечить управление потоком с целью предотвращения ситуации, когда передающий узел на одной стороне линии связи заваливает пакетами принимающий узел на другой стороне линии.
- *Обнаружение ошибок.* Принимающий узел может неверно посчитать, что значение бита в кадре равно нулю, в то время как передавалась единица, и наоборот. Подобные битовые ошибки вызываются ослаблением сигнала и электромагнитными помехами. Поскольку нет смысла передавать дальше дейтаграмму, содержащую ошибки, многие протоколы канального уровня предоставляют услугу по обнаружению ошибок в кадре. Для этого передающий узел добавляет к кадру биты обнаружения ошибок (контрольную сумму), а получающий узел выполняет проверку контрольной суммы. Служба обнаружения ошибок очень распространена среди протоколов канального уровня. Как было показано в главах 3 и 4, транспортный и сетевой уровни в Интернете также предоставляют ограниченную услугу по обнаружению ошибок. На канальном уровне обнаружение ошибок сложнее и, как правило, реализуется аппаратно.
- *Исправление ошибок.* Исправление ошибок выполняет расширенная служба обнаружения ошибок. Такая служба способна не только обнаружить ошибку в кадре, но также определить, в каком именно разряде она произошла, и таким образом исправить некоторые ошибки. Услуга по исправлению ошибок предоставляется некоторыми протоколами канального уровня (например, ATM), но, как правило, не для всего пакета, а только для его заголовка. Эта тема будет обсуждаться в разделе «Обнаружение и исправление ошибок».
- *Дуплексная и полудуплексная передача.* При дуплексной передаче оба узла могут передавать друг другу пакеты одновременно. При полудуплексной передаче оба узла тоже могут передавать друг другу пакеты, но только поочередно.

Как упоминалось выше, многие услуги, предоставляемые службами канального уровня, аналогичны услугам, предоставляемым на транспортном уровне. Например, надежная доставка может предоставляться как канальным, так и транспортным уровнями. Хотя службы надежной доставки двух уровней имеют много общего (см. раздел «Принципы надежной передачи данных» в главе 3), они не одинаковы. Надежный транспортный протокол обеспечивает *сквозную* (через все линии свя-

зи) надежную доставку между двумя процессами. Надежный протокол канального уровня обеспечивает надежную доставку только между двумя узлами (то есть между узлами, соединенными одной линией связи). Аналогично протоколы канального и транспортного уровней могут предоставлять услугу по управлению потоком. Однако на транспортном уровне управление сквозное, тогда как протокол канального уровня обеспечивает управление потоком только на каждой двухточечной линии.

Адаптеры

В каждой линии связи протокол канального уровня обычно реализован в *адаптере*. Адаптер представляет собой плату, или карту, PCMCIA (Personal Computer Memory Card International Association — Международная ассоциация производителей плат памяти для персональных компьютеров IBM PC), на которой, как правило, установлены микросхемы памяти и DSP (Digital Signal Processor — цифровой обработчик сигналов), а также интерфейсы шины хоста и линии связи. Адаптеры также часто называют *сетевыми интерфейсными картами* (Network Interface Card, NIC). Как показано на рис. 5.2, сетевой уровень передающего узла (то есть хоста или маршрутизатора) передает дейтаграмму сетевого уровня адаптеру, управляющему передающей стороной линии связи. Адаптер помещает дейтаграмму в кадр, а затем передает кадр в канал связи. На другой стороне линии связи принимающий адаптер получает кадр целиком, извлекает из него дейтаграмму сетевого уровня и передает ее сетевому уровню. Если протокол канального уровня обеспечивает обнаружение ошибок, тогда передающий адаптер считает контрольную сумму кадра и помещает ее в кадр перед отправкой, а принимающий адаптер проверяет контрольную сумму полученного кадра. Если протокол канального уровня обеспечивает надежную доставку, тогда в адаптерах полностью реализуются механизмы надежной доставки (например, порядковые номера, таймеры и подтверждения). Если протокол канального уровня предоставляет возможность произвольного доступа (см. раздел «Протоколы коллективного доступа»), тогда этот протокол также целиком реализуется в адаптерах.

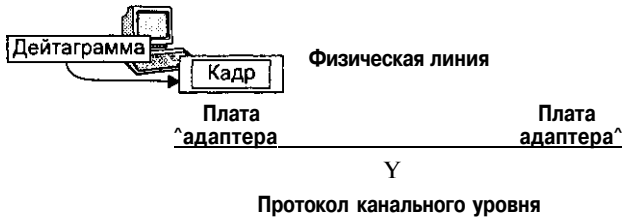


Рис. 5.2. Протокол канального уровня реализован в адаптерах на двух концах линии связи

Адаптер представляет собой полуавтономное устройство. Например, адаптер может принять кадр, определить факт наличия ошибок в кадре, а также отбросить кадр, не уведомив свой «родительский» узел. Получив кадр, адаптер прерывает работу своего родительского узла, только если он хочет передать ему дейтаграмму сетевого уровня. Аналогично, когда узел передает дейтаграмму вниз по стеку про-

токолов адаптеру, узел полностью делегирует адаптеру задачу передачи дейтаграммы по линии связи. Адаптер является полуавтономным, а не полностью автономным устройством. Хотя на рис. 5.3 адаптер изображен в виде отдельного блока, как правило, адаптер располагается в одном корпусе с остальными элементами узла, пользуется с ними общими источником питания и шинами и, наконец, управляется узлом.

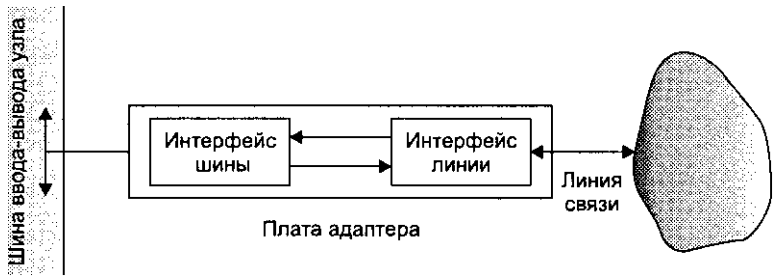


Рис. 5.3. Адаптер представляет собой полуавтономное устройство

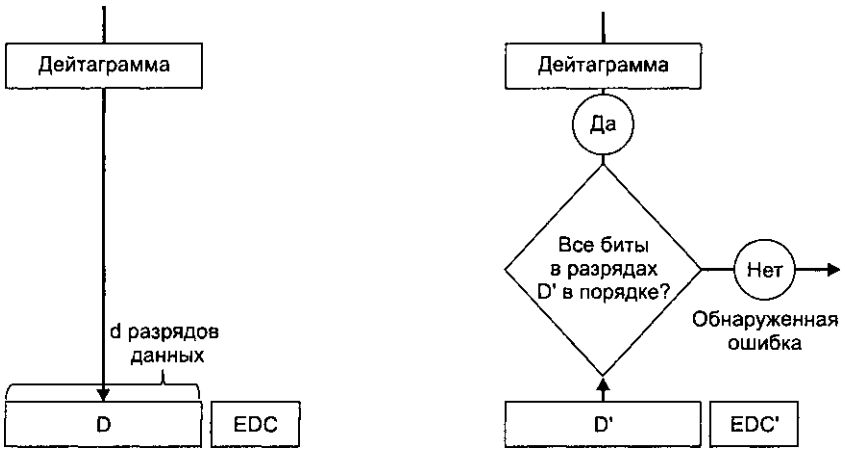
Как показано на рисунке, основными компонентами адаптера являются интерфейсы шины и линии. Интерфейс шины отвечает за общение с родительским узлом. Он переносит данные и управляющую информацию между адаптером и родительским узлом. Интерфейс линии отвечает за реализацию протокола канального уровня. Помимо формирования кадров и извлечения из кадров дейтаграмм, он может предоставлять услуги по обнаружению ошибок, произвольному доступу и другие услуги канального уровня. Он также включает цепи передачи и приема. Такие популярные технологии, как Ethernet, реализованы в интерфейсе линии при помощи набора микросхем, которые можно приобрести на рынке. По этой причине Ethernet-адаптеры невероятно дешевы — часто меньше 30 долларов за адаптер, поддерживающий скорости передачи 10 Мбит/с и 100 Мбит/с. Дополнительную информацию об адаптерах для сетей Ethernet, работающих со скоростями 10, 100 и 1000 Мбит/с, а также о ATM-сетях, работающих на скорости 155 Мбит/с, можно узнать, посетив web-сайт компании 3Com (http://www.3com.com/products/nics.html).

Обнаружение и исправление ошибок

В предыдущем разделе мы отмечали, что на канальном уровне часто предоставляются услуги по обнаружению и исправлению ошибок в отдельных разрядах кадра, передаваемого между двумя физически соединенными узлами. Как было показано в главе 3, аналогичные услуги часто предоставляются также на транспортном уровне. В данном разделе мы рассмотрим некоторые простейшие методы обнаружения и исправления однобитовых ошибок. Данному вопросу посвящены целые книги (см., например, [27, 454]), а здесь мы можем дать лишь краткое введение

в тему. Цель нашего обсуждения — сформировать представление о возможностях этих методов, а также показать, как работают наиболее простые из них и как они практически используются для обнаружения и исправления ошибок на канальном уровне.

На рис. 5.4 представлена схема движения данных, иллюстрирующая помехоустойчивое кодирование. На передающем узле к данным *D* добавляются разряды *EDC* (Error Detection and Correction — обнаружение и исправление ошибок), позволяющие обнаруживать и исправлять ошибки. Как правило, таким образом защищается не только дейтаграмма, передаваемая сетевым уровнем для транспортировки по линии связи, но также и адресная информация канального уровня, включая порядковые номера и другие поля заголовка кадра канального уровня. В кадре канального уровня принимающему узлу передаются как данные *D,* так и биты обнаружения и исправления ошибок *EDC*. Принимающий узел получает разряды *D'* и *EDC* соответственно. Обратите внимание, что значения *D'YL EDC* могут отличаться от исходных значений *D* и *EDC* вследствие ошибок при передаче.



Канал связи

Рис. 5.4. Сценарий обнаружения и исправления ошибок

На основании принятых полей *D* 'и *ED C* получатель должен определить, совпадают ли разряды *D 'c* разрядами *D.* Обратите внимание, что вопрос заключается в том, обнаружена ли ошибка, а не в том, произошла ли ошибка! Методы обнаружения и исправления ошибок иногда, *но не всегда,* позволяют получателю обнаруживать, что произошла ошибка в одном или нескольких разрядах кадра. Другими словами, Даже при помехоустойчивом кодировании, то есть при добавлении к данным избыточной информации (контрольной суммы), остается вероятность, что *ошибка не будет обнаружена,* а получатель не получит информации об искажении по-

лученных данных во время их прохождения по каналу связи. В результате канальный уровень получателя может передать сетевому уровню поврежденную дейтаграмму или не заметить повреждения какого-либо другого поля в заголовке кадра. Поэтому следует выбирать такую схему определения ошибок, при которой вероятность подобных событий мала. Как правило, чем изощреннее методы обнаружения и исправления ошибок (снижающие вероятность появления необнаруженных ошибок), тем больше накладные расходы — требуется больше операций для вычисления контрольной суммы и больше времени для передачи дополнительной информации.

Мы рассмотрим три метода обнаружения ошибок в переданных данных: контроль четности (чтобы проиллюстрировать основные идеи, лежащие в основе методов обнаружения и исправления ошибок), вычисление контрольных сумм (этот метод больше характерен для транспортного уровня) и применение циклического избыточного кода (как правило, используется в адаптерах канального уровня).

Контроль четности

Возможно, простейшая форма обнаружения ошибок заключается в использовании одного *бита четности*. Предположим, что на рис. 5.4 передаваемые данные *D* имеют длину *d* разрядов. При проверке на четность отправитель просто добавляет к данным один бит, значение которого вычисляется как сумма всех *d* разрядов данных по модулю 2. В этом случае количество единиц в получающемся в результате числе всегда будет четным. Применяются также схемы, в которых контрольный бит инвертируется, в результате чего количество единиц в получающемся в результате числе всегда будет нечетным. На рис. 5.5 изображена схема проверки на четность, а единственный бит четности хранится в отдельном поле.

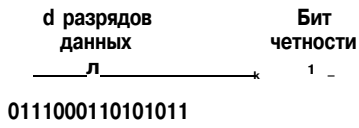


Рис. 5.5. Контроль четности

Действия, выполняемые получателем при использовании такой схемы, также очень просты. Получатель должен всего лишь сосчитать количество единиц в полученных $d + 1$ разрядах. Если при проверке на четность получатель обнаруживает, что в принятых им данных нечетное количество единичных разрядов, он понимает, что произошла ошибка, по меньшей мере, в одном разряде. В общем случае это означает, что в полученных данных инвертировано нечетное количество разрядов (произошла ошибка нечетной кратности).

Что произойдет, если в полученном пакете данных произойдет четное количество однобитовых ошибок? В этом случае получатель не сможет обнаружить ошибку. Если вероятность ошибки в одном разряде мала и можно предположить, что ошибки в отдельных разрядах возникают независимо друг от друга, тогда вероятность нескольких ошибок в одном пакете крайне мала. В таком случае единственного бита

четности может быть достаточно. Однако практические наблюдения показали, что в действительности ошибки не являются независимыми, а часто группируются в пакеты ошибок. В случае пакетных ошибок вероятность того, что получатель не обнаружит ошибку в пакете, может приблизиться к величине 50 % [476]. Очевидно, в такой ситуации требуется более надежная схема обнаружения ошибок! Но прежде чем перейти к изучению схем обнаружения ошибок, применяемых на практике, рассмотрим простую схему, которая обобщает предыдущую схему одноразрядного контроля четности и помогает понять принцип работы методов исправления ошибок.

На рис. 5.6 показано двухмерное обобщение одноразрядной схемы проверки на четность. В данной схеме *d* разрядов пакета данных разделяются на *г* строк и *j* столбцов, образуя прямоугольную матрицу. Значение четности вычисляется для каждой строки и каждого столбца. Получающиеся в результате *i* +/ +1 битов четности образуют разряды обнаружения ошибок кадра канального уровня.

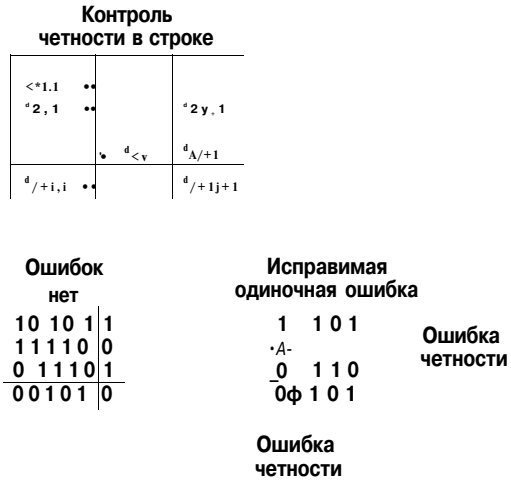


Рис. 5.6. Двухмерный контроль четности

Предположим теперь, что в исходном блоке данных из *d* разрядов происходит однократная ошибка. В такой двухмерной схеме контроля четности об ошибке будут одновременно сигнализировать контрольные разряды строки и столбца. Таким образом, получатель сможет не только обнаружить сам факт ошибки, но и по номерам строки и столбца найти поврежденный бит данных и *исправить* его! На рисунке показан пример, в котором поврежден бит в позиции (2, 2) — он изменил свое значение с 1 на 0. Такую одиночную ошибку получатель может не только обнаружить, но и исправить. Хотя нас, в первую очередь, интересует обнаружение и исправление ошибок в исходных *d* разрядах, данная схема позволяет также обнаруживать и исправлять одиночные ошибки в самих битах четности. Кроме того, данная двухмерная схема контроля четности позволяет обнаруживать (но не исправлять!) любые комбинации из двух одиночных ошибок (то есть двойные ошибки) в пакете. Другие свойства двухмерной схемы рассматриваются в упражнениях в конце этой главы.

Способность приемника обнаруживать и исправлять ошибки иногда называют *прямым исправлением ошибок* (Forward Error Correction, FEC). Подобные приемы широко применяются в устройствах хранения и воспроизведения звука, например на лазерных компакт-дисках. В сетях методы обнаружения и исправления ошибок могут использоваться сами по себе, а также в сочетании с автоматическими запросами на повторную передачу, которые мы рассматривали в главе 3. Методы обнаружения и исправления ошибок очень полезны, так как позволяют снизить необходимое количество повторных передач. Кроме того (что, возможно, даже важнее), эти методы позволяют получателю немедленно исправлять ошибки. Таким образом, получатель данных может не ждать, пока отправитель получит его сообщение об ошибке и вышлет пакет еще раз, что может быть существенным преимуществом в сетевых приложениях реального времени [441]. Недавние исследования, касающиеся использования методов обнаружения и исправления ошибок, представлены в [28, 50,361,462].

Вычисление контрольной суммы

Методы вычисления контрольной суммы обрабатывают *d* разрядов данных (см. рис. 5.4) как последовательность ^-разрядных целых чисел. Наиболее простой метод заключается в простом суммировании этих ^-разрядных целых чисел и использовании полученной суммы в качестве битов определения ошибок. На этом методе основан алгоритм вычисления контрольной суммы, принятый в Интернете, — байты данных группируются в 16-разрядные целые числа и суммируются. Затем от суммы берется обратное значение (дополнение до 1), которое и помещается в заголовок сегмента. Как уже отмечалось в разделе «Протокол UDP — передача без установления соединения» главы 3, получатель проверяет контрольную сумму, вычисляя дополнение до 1 от суммы полученных данных (включая контрольную сумму), и сравнивает результат с числом, все разряды которого равны 1. Если хотя бы один из разрядов результата равен 0, это означает, что произошла ошибка. Принятый в Интернете алгоритм вычисления контрольной суммы и его реализация подробно описываются в RFC 1071. В протоколах TCP и UDP контрольная сумма вычисляется по всем полям (включая поля заголовка и данных). В других протоколах, например XTP [493], вычисляются две контрольной суммы, одна для заголовка, а другая для всего пакета.

Метод вычисления контрольной суммы для пакетов требует относительно небольших накладных расходов. Например, в протоколах TCP и UDP для контрольной суммы используются всего 16 бит. Однако подобные методы предоставляют относительно слабую защиту от ошибок по сравнению с обсуждаемым далее методом контроля с помощью *циклического избыточного кода,* который часто используется на канальном уровне. Разумеется, возникает вопрос: почему на транспортном уровне применяют контрольные суммы, а на канальном уровне — циклический избыточный код? Вспомним, что транспортный уровень, как правило, реализуется на хосте программно как часть операционной системы хоста. Поскольку обнаружение ошибок на транспортном уровне реализовано программно, важно, чтобы схема обнаружения ошибок была простой. В то же время обнаружение ошибок на

канальном уровне реализуется аппаратно в адаптерах, способных быстро выполнять более сложные операции по вычислению циклического избыточного кода. Таким образом, основная причина использования контрольных сумм на транспортном уровне и более сложного метода вычисления циклического избыточного кода на канальном уровне состоит в том, что программно проще реализовать вычисление суммы.

В [313] описываются улучшенные взвешенные коды контрольных сумм, пригодные для высокоскоростной программной реализации, а в [144] обсуждаются быстрые методы программной реализации для вычисления не только взвешенных кодов контрольных сумм, но и циклических (см. ниже), а также других кодов.

Циклический избыточный код

Широко применяемый в современных компьютерных сетях метод обнаружения ошибок основан на *контроле при помощи циклического избыточного кода* (Cyclic Redundancy Check, CRC). Циклические избыточные коды также называют полиномиальными кодами, так как при их вычислении битовая строка рассматривается как многочлен (полином), коэффициенты которого равны 0 или 1, и операции с этой битовой строкой можно интерпретировать как операции деления и умножения многочленов.

Циклические коды работают следующим образом. Рассмотрим фрагмент данных Д состоящий из *d* разрядов, которые передающий узел хочет отправить принимающему узлу. Отправитель и получатель должны договориться о последовательности из r+ 1 бит, называемой образующим многочленом (или генератором), который мы будем обозначать *G*. Старший (самый левый) бит образующего многочлена *G* должен быть равен 1. Ключевую идею циклических избыточных кодов иллюстрирует рис. 5.7. Для заданного фрагмента данных *D* отправитель формирует г дополнительных разрядов *R,* которые он добавляет к данным Д так что получающееся в результате число, состоящее из *d* + *г* бит, делится по модулю 2 на образующий многочлен (число) *G* без остатка. Таким образом, процесс проверки данных на наличие ошибки относительно прост. Получатель делит полученные *d* + *г* бит на образующий многочлен *G*. Если остаток от деления не равен нулю, это означает, что данные повреждены. В противном случае данные считаются верными и принимаются.

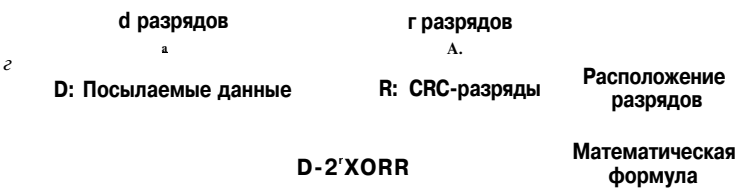


Рис. 5.7. CRC-коды

Все операции по вычислению CRC-кода производятся в арифметике по модулю 2 без переносов в соседние разряды. Это означает, что операции сложения и вычита-

ния идентичны друг другу и эквивалентны поразрядной операции исключающего ИЛИ (exclusive OR, XOR). Например:

1011 XOR 0101 = 1110

1001XOR 1101 = 0100

Также мы получим:

1011-0101 - 1110

1001 -1101 = 0100

Операции умножения и деления аналогичны соответствующим операциям в обычной двоичной арифметике с той разницей, что любая требующаяся при этом операция сложения или вычитания выполняется без переносов в соседний разряд. Как и в обычной арифметике, умножение на 2* означает сдвиг числа на *k* разрядов влево. Таким образом, при заданных значениях *D* и *R* величина $D \cdot 2^r$ XOR *R* соответствует последовательности из d+ r бит, показанной на рис. 5.7. В нашем дальнейшем обсуждении мы будем использовать именно эту алгебраическую форму для обозначения последовательности из *d + г* бит.

Вернемся теперь к главному вопросу: как отправитель вычисляет *R?* Вспомним, что нам требуется найти такое значение *R,* чтобы для некоторого *n* выполнялось следующее равенство:

$$D-2>XORR = nG.$$

То есть требуется выбрать такое значение *R,* чтобы $D \cdot 2^k$ XOR *R* делилось на *G* без остатка. Если прибавить к каждой части уравнения значение *R* по модулю 2, то мы получим

$$D-2^r = nGXOR\ R.$$

Отсюда следует, что если мы разделим *D* • 2' на *G,* то значение остатка будет равно *R*. Таким образом, мы можем вычислить *R* как

$$R = остаток\frac{}{G}.$$

На рис. 5.8 показан пример вычисления *R* для *D* = 101110, *d* = 6, *G* = 1001 и *г* = 3. В этом случае отправитель передает следующие 9 бит: 1011100И. Вы можете сами проверить правильность расчетов, а также убедиться, что

£•2'-101011 -2'=GXOR#.

Для 8-, 12-, 16- и 32-разрядных образующих многочленов определены международные стандарты. Восьмиразрядный CRC-код используется для защиты 5-байтового заголовка ATM-ячеек. В 32-разрядном стандарте CRC-32, принятом в ряде IEEE-протоколов канального уровня, используется образующий многочлен вида

$$G_{CRC-32} = 100000100110000010001110110110111.$$

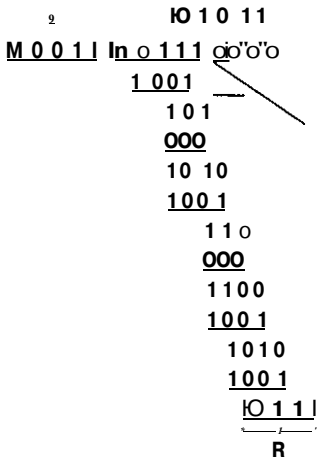


Рис. 5.8. Пример вычисления CRC-кода

Каждый стандарт CRC-кода способен обнаруживать ошибочные пакеты длиной не более r разрядов. Кроме того, при соответствующих допущениях ошибочный пакет длиной более чем r разрядов будет обнаружен с вероятностью $1 - 0{,}5^r$. Помимо этого каждый стандарт CRC-кода может обнаруживать ошибки нечетной кратности. Теория CRC-кодов и еще более мощных кодов выходит за рамки темы данной книги. Прекрасное введение в эту тему можно найти в [454].

Протоколы коллективного доступа

В начале главы мы отметили, что существуют два типа сетевых каналов: двухточечные и широковещательные. Двухточечная линия связи состоит из передатчика на одном конце линии и приемника на другом конце. Для двухточечных линий связи разработано множество протоколов канального уровня. Ниже в этой главе будет рассказано о двух таких протоколах: PPP (Point-to-Point Protocol — протокол передачи от точки к точке) и HDLC (High-level Data Link Control — высокоуровневый протокол управления каналом). Второй тип канала, *широковещательный канал,* может иметь несколько передающих и принимающих узлов, присоединенных к одному и тому же совместно используемому широковещательному каналу. Термин «широковещание» используется здесь потому, что, когда один из узлов передает кадр, этот кадр принимается всеми остальными узлами, присоединенными к каналу. Примерами применения широковещательной технологии канального уровня являются Ethernet-сети и беспроводные локальные сети. В данном разделе мы сделаем небольшое отступление от темы протоколов канального уровня и сначала рассмотрим крайне важную для канального уровня проблему, заключающуюся в координации доступа множества передающих и принимающих узлов к общему широковещательному каналу, — так называемую *проблему коллективного доступа.* Широковещательные каналы часто применяют в локальных сетях, то есть в сетях, географически сконцентрированных в одном здании (или комплексе зда-

ний, принадлежащих одной организации, например университету или компании). В конце этого раздела мы познакомимся с тем, как используются каналы коллективного доступа в локальных сетях.

Нам всем известно понятие широковещательной рассылки, так к № эта технология передачи данных используется в телевидении с момента его изобд>етения. Но в традиционном телевидении широковещательная рассылка является односторонней, так как там один фиксированный узел передает информацию множеству получающих узлов. Между тем узлы компьютерной широковещательн ой сети могут как принимать данные, так и передавать их. Возможно, более близкой к компьютерной широковещательной сети аналогией является вечеринка с коктейлями, где множество людей собираются в большой комнате (средой широковещательного канала при этом является воздух), чтобы поговорить и послушать. Вторая хорошая аналогия, хорошо знакомая многим читателям, — классная комната, в которой преподаватель и студенты совместно используют один общий широковещательный носитель. Центральная проблема обоих сценариев заключается в том, чтобы решить, кто и когда получает право говорить (передавать по каналу). Для себя люди разработали сложный набор правил коллективного использования канала:

- «дайте возможность поговорить каждому»;
- «не говорите, пока с вами не заговорят»;
- «не монополизируйте беседу»;
- «если у вас есть вопрос, поднимите руку»;
- «не прерывайте говорящего громким храпом».

Обмен данными в компьютерных сетях также управляется наборами правил, составляющих так называемые *протоколы коллективного доступа*. Как показано на рис. 5.9, протоколы коллективного доступа применяются в сетях самых разных конфигураций, включая кабельные и беспроводные локальные сети, а также спутниковые сети. Хотя технически каждый узел получает доступ к широковещательному каналу через адаптер, в данном разделе и передающее, и принимающее устройства мы будем называть *узлом*. На практике по одному широковещательному каналу могут обмениваться данными сотни или даже тысячи узлов.

Поскольку передавать кадры могут все узлы, возможна ситуация, когда одновременно начнут передачу несколько узлов. Когда такое происходит, каждый из узлов одновременно получает несколько кадров, то есть на принимающих узлах имеет место *коллизия* переданных кадров. Как правило, в случае коллизии принимающие узлы не могут корректно обрабатывать принятые кадры, так как сигналы таких кадров накладываются друг на друга. Таким образом, все вовлеченные в коллизию кадры теряются, а все время, пока они передавались, оказывается потраченным впустую. Очевидно, что наличие множества узлов, требующих частой передачи данных, означает высокую вероятность коллизий и низкий коэффициент использования канала.

Чтобы гарантировать высокую производительность канала при большом количестве активных узлов, необходимо каким-то образом координировать передачу ими кадров. За эту координацию отвечает протокол коллективного доступа. За послед-

ние 30 лет по теме протоколов коллективного доступа были написаны тысячи статей и защищены сотни докторских диссертаций. Всесторонний обзор этой работы можно найти в [424]. Более того, активные исследования протоколов коллективного доступа продолжаются до сих пор, что вызвано появлением новых типов линий связи, например новых беспроводных каналов.

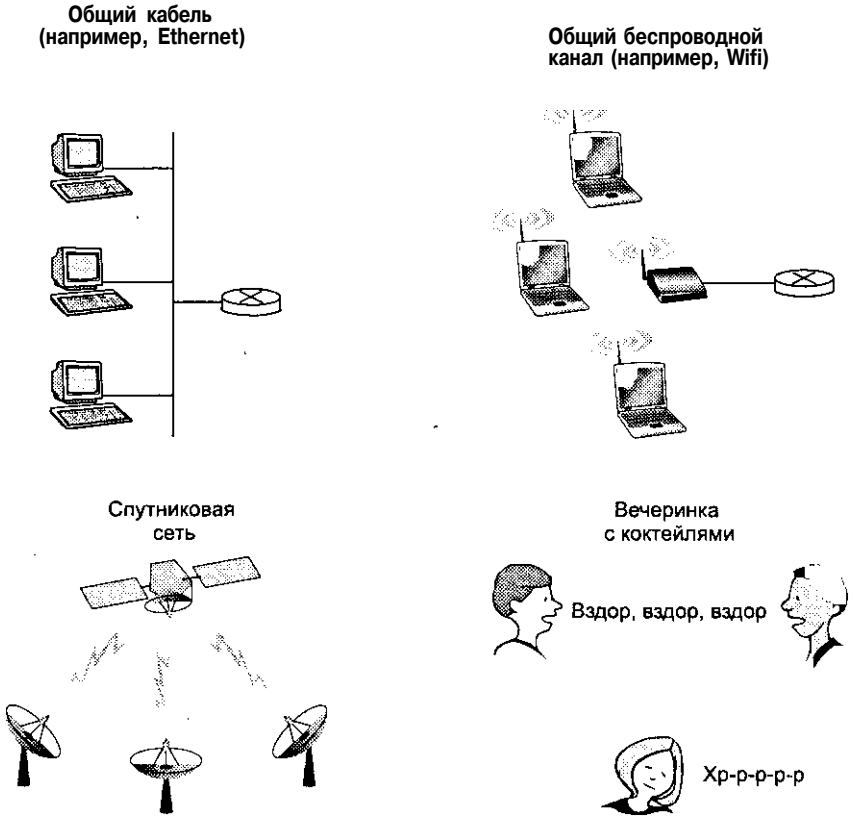


Рис. 5.9. Примеры каналов коллективного доступа

За годы с помощью широкого спектра технологий канального уровня были реализованы десятки протоколов коллективного доступа. Тем не менее практически любой из этих протоколов мы можем отнести к одной из трех категорий: *протоколы разделения канала, протоколы произвольного доступа* и *протоколы последовательного доступа.* Мы обсудим эти категории протоколов коллективного доступа в следующих трех подразделах.

В заключение этого обзора отметим, что в идеальном случае протокол коллективного доступа для широковещательного канала со скоростью передачи данных R бит/с должен обладать следующими характеристиками.

- Когда данные для передачи есть только у одного узла, этот узел обладает пропускной способностью в R бит/с.

- Когда данные для передачи есть у M узлов, каждый из этих узлов обладает пропускной способностью в R/M бит/с. Это не означает, что каждый из M узлов в каждый момент времени может передавать данные со скоростью R/M бит/с, — это средняя скорость передачи данных каждого из узлов.
- Протокол является децентрализованным, то есть не существует управляющих узлов, выход из строя которых может остановить работу всей сети.
- Протокол прост и дешев в реализации.

Протоколы разделения канала

TDM- и FDM-мультиплексирование

Вспомним (см. раздел «Ядро компьютерных сетей» в главе 1) наше обсуждение двух методов разделения пропускной способности широковещательного канала между узлами: мультиплексирования с временным разделением (Time-Division Multiplexing, TDM) и мультиплексирования с частотным разделением (Frequent-Division Multiplexing, FDM). Предположим для примера, что канал поддерживает N *узлов* и скорость передачи данных в канале равна R бит/с. При временном разделении канала время делится на интервалы, называемые *кадрами,* каждый из которых делится на N элементарных интервалов времени, называемых *слотами.* Затем каждому из N узлов назначается один временной слот. Когда у узла есть кадр для отправки, он передает биты этого кадра в течение назначенного ему элементарного интервала времени. Как правило, длина слота выбирается таким образом, чтобы за этот интервал можно было передать один кадр. На рис. 5.10 показан простой пример мультиплексирования с временным разделением для канала с четырьмя узлами. Если вернуться к нашей аналогии с вечеринкой, то при такой схеме каждый участник вечеринки сможет говорить в течение некоторого фиксированного интервала времени, после чего такое же право получает другой участник, и т. д. После того как возможность поговорить получат все участники вечеринки, эта возможность снова переходит первому участнику, и все повторяется.

Привлекательность временного разделения канала заключается в том, что такая схема полностью устраняет коллизии и обладает идеальной справедливостью: каждый узел получает выделенную скорость передачи данных, равную R/N бит/с в течение каждого временного кадра. Однако у данной схемы есть два существенных недостатка. Во-первых, каждый узел ограничен средней скоростью передачи данных в R/N бит/с даже в том случае, когда этот узел единственный, кому нужно передавать данные в этот момент. Во-вторых, при передаче узел всегда должен ждать своей очереди, даже когда кроме него никто не передает данные. Таким образом, очевидно, что временное разделение канала плохо подходит для протокола коллективного доступа.

Метод мультиплексирования с частотным разделением делит канал с пропускной способностью R бит/с на частотные диапазоны с полосой пропускания R/N бит/с, при эгом каждому узлу выделяется собственный частотный диапазон. Таким образом, при методе частотного разделения из одного канала с пропускной способностью R бит/с создается N каналов с пропускной способностью R/N бит/с. Мульти-

плексирование с частотным разделением канала обладает теми же преимуществами и недостатками, что и с временным. Устраняются коллизии и обеспечивается справедливое распределение пропускной способности между узлами, но пропускная способность каждого узла ограничена значением R/N бит/с, независимо от текущей загруженности канала.

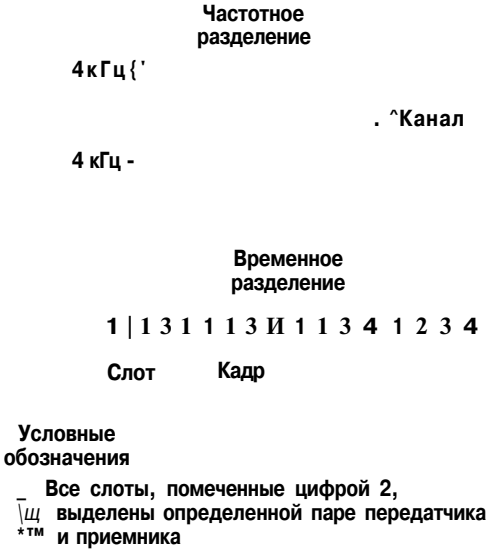


Рис. 5.10. Примеры мультиплексирования с временным и частотным разделением для канала с четырьмя узлами

Протокол CDMA

Еще один метод коллективного использования общего канала, который мы рассмотрим, предлагает протокол *CDMA* (Code Division Multiple Access — множественный доступ с кодовым разделением). В отличие от схем мультиплексирования с частотным и временным разделением канала, предоставляющих узлам частотные диапазоны или временные интервалы, протокол CDMA назначает каждому узлу собственный *код.* Затем каждый узел использует этот уникальный код для кодирования передаваемых им данных. Как мы увидим, протокол CDMA позволяет нескольким узлам передавать данные *одновременно,* при этом получатели могут корректно принимать эти данные (при условии, что получателю известен код передатчика). Протокол CDMA в течение некоторого времени использовался в военных системах связи (благодаря своей устойчивости к попыткам подавления сигнала), а в настоящее время получает все более широкое распространение в гражданских беспроводных средствах связи коллективного доступа.

В протоколе CDMA при передаче каждый бит кодируется, для чего он умножается на некий сигнал (код), изменяющийся с частотой, в несколько раз превосходящей исходную скорость передачи данных. Рисунок 5.11 иллюстрирует простой идеализированный сценарий кодирования/декодирования данных протоколом

CDMA. Предположим, что частота, с которой исходные биты попадают в кодирующее устройство CDMA, определяет длительность временного интервала, то есть для передачи каждого бита данных требуется один однобитовый временной слот. Пусть d^t — значение бита данных для f-го битового слота. Нам будет удобнее представлять нулевой бит данных как - 1 . Каждый битовый слот разделяется на M мини-слотов. На рисунке $M = 8$, хотя на практике применяют гораздо большие значения. Используемый передатчиком CDMA-код состоит из последовательности M значений $c^m, m = 1, M$, каждое из которых может равняться +1 или - 1 . В данном примере этот код равен (1, 1, 1, - 1 , 1, - 1 , - 1 , - 1).

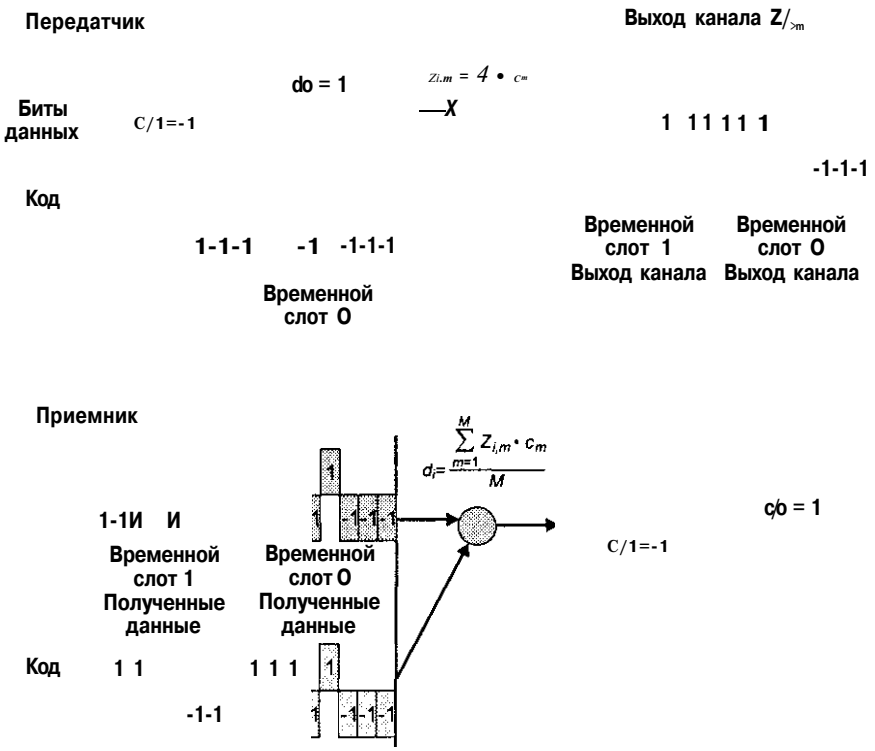


Рис. 5.11. Простой пример работы протокола CDMA

Чтобы проиллюстрировать, как работает протокол CDMA, рассмотрим /-й бит данных *dj*. Для m-го мини-слота передачи бита $d^{\prime}$ выход кодирующего устройства Z^{ittn} будет равен произведению величины $d^{\prime}$ на m-й бит кода CDMA c^m:

$$Z_{i,m} = d_i \cdot c_m \qquad (5.1)$$

В простом случае отсутствия передатчиков, кадры которых накладываются друг на друга, получатель получит кодированные биты $Z^{z>}$, по которым сможет определить значение исходного бита данных по формуле

$$d_i = \frac{1}{M} \sum_{m=1}^{M} Z_{i,m} \cdot c_m. \qquad (5.2)$$

Возможно, читателю захочется подробнее изучить пример на рис. 5.11, чтобы убедиться, что исходные биты данных корректно восстанавливаются приемником с помощью формулы 5.2.

Однако на практике все бывает далеко не так идеально, и протоколу CDMA приходится работать в ситуации нескольких одновременно работающих передатчиков, использующих разные коды. Но как удается CDMA-приемнику корректно распознать оригинальные биты данных, если на них накладываются биты, переданные другими передатчиками? Протокол CDMA работает в предположении об аддитивности интерферирующих битовых сигналов. Это означает, что, если, например, за один и тот же мини-слот три передатчика передают значение 1 а затем четвертый передатчик передает - 1 , тогда все приемники получат значение 2 (1 + 1 + 1 - 1 = 2). В присутствии нескольких передатчиков передатчик **5** вычисляет свои передаваемые данные $Z\backslash^m$ тем же способом (см. формулу 5.1). Однако теперь значение, полученное приемником за *m-и* мини-слот, представляет собой *сумму* битов, переданных всеми N передатчиками за этот интервал времени:

$$Z^* \quad = \mathbf{y}\ 7^s$$

Замечательно, что при соответствующем выборе кодов каждый получатель сможет извлечь передаваемые ему данные из суммарного сигнала способом, который используется в формуле 5.2:

$$d_i = \frac{\mathbf{j}}{M} \sum_{m=1}^{M} Z_{i,m}^{*} \cdot c_m. \qquad (5.3)$$

Пример работы протокола CDMA с двумя передатчиками показан на рис. 5.12. Верхний передатчик использует код (1, 1, 1,-1, 1,-1,-1,-1), а нижний передатчик — код (1,-1, 1, 1, 1,-1, 1, 1). В данном примере получатель восстанавливает оригинальную последовательность битов, посланных верхним передатчиком. Обратите внимание, что получатель может извлечь из сигнала данные, передаваемые передатчиком 1, хотя на сигнал передатчика 1 налагается сигнал передатчика 2.

Если вернуться к нашей аналогии с вечеринкой, то протокол CDMA напоминает ситуацию, в которой участники вечеринки разговаривают на нескольких языках. В этом случае людям несложно фиксировать внимание на понятном им языке и отфильтровывать остальную речь. Таким образом, протокол CDMA разделяет кодовое пространство (в отличие от протоколов мультиплексирования с временным или частотным разделением) и предоставляет каждому узлу определенную долю кодового пространства.

К сожалению, мы не можем здесь подробнее обсудить протокол CDMA. Отметим лишь, что при его практическом применении возникает множество сложных проблем. Во-первых, чтобы приемник мог извлечь сигнал конкретного передатчика,

следует тщательно выбирать CDMA-коды. Во-вторых, при нашем обсуждении протокола CDMA предполагалось, что мощности принимаемых приемником сигналов от разных передатчиков одинаковы, чего крайне трудно добиться на практике. Этим и другим вопросам, касающимся протокола CDMA, посвящено довольно много литературы [386, 536].

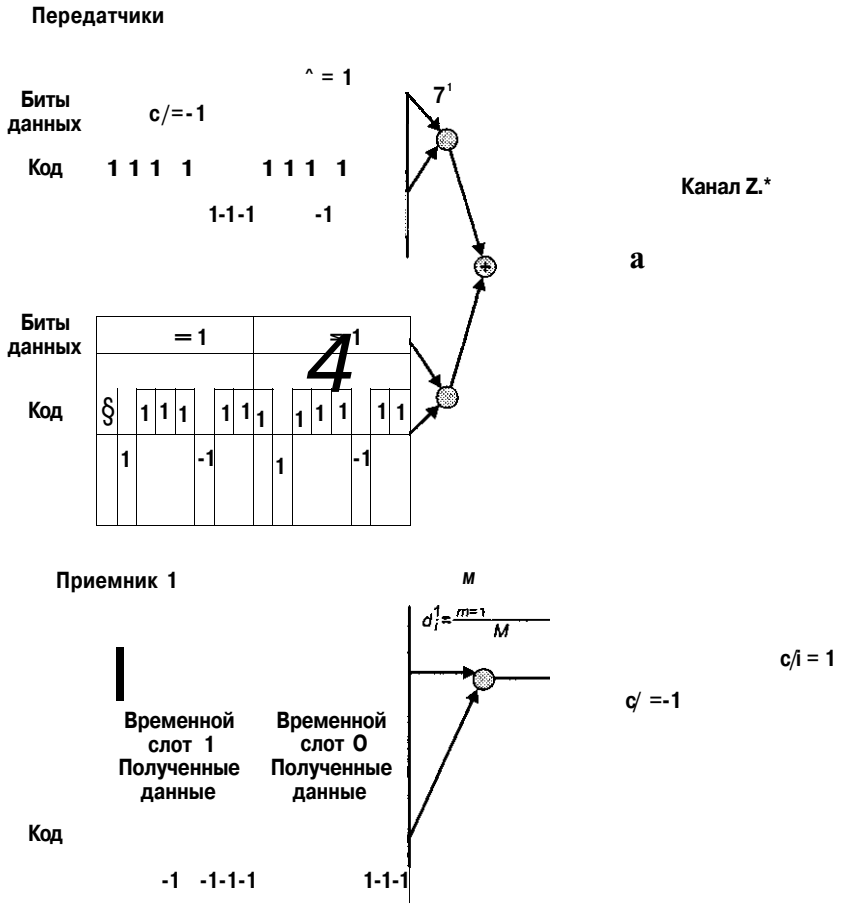


Рис. 5.12. Пример работы протокола CDMA с двумя передатчиками

Протоколы произвольного доступа

Второй широкий класс протоколов коллективного доступа составляют так называемые протоколы произвольного доступа. В протоколе произвольного доступа передающий узел всегда передает данные в канал с максимальной скоростью, то есть *R* бит/с. Когда возникает коллизия, каждый вовлеченный в нее узел передает

свой кадр повторно до тех пор, пока ему не удастся пройти по каналу без коллизий. Однако, испытав коллизию, узел, как правило, не повторяет передачу тут же, а *выжидает в течение случайного интервала времени.* Благодаря разной длительности случайных интервалов времени существует ненулевая вероятность того, что интервал, выбранный одним из узлов, окажется меньше, чем у других вовлеченных в коллизию узлов, и он успеет «пропихнуть» свой кадр в канал без коллизии.

В литературе описаны десятки, если не сотни, протоколов произвольного доступа [27,424]. В данном разделе мы опишем некоторые из наиболее популярных, включая протоколы ALOHA [5, 6] и CSMA [271]. Затем в разделе «Ethernet» мы обсудим детали популярной Ethernet-сети [329], использующей протокол CSMA.

Дискретный протокол ALOHA

Начнем наше изучение протоколов произвольного доступа с одного из наиболее простых протоколов, так называемого дискретного протокола ALOHA. В нашем описании дискретной системы ALOHA мы будем предполагать следующее:

- все кадры состоят ровно из L бит;
- время разделено на интервалы времени (слоты) длительностью L/R секунд (это время, за которое передается один кадр);
- узлы начинают передачу кадров только в момент начала очередного слота;
- узлы синхронизируются так, что каждый узел знает, когда начинается слот;
- если в течение данного временного слота сталкиваются несколько кадров, тогда все узлы обнаруживают факт столкновения, прежде чем закончится данный слот.

Работа дискретного протокола ALOHA на каждом узле проста. Когда у узла есть новый кадр для передачи, он ждет, пока не начнется новый временной слот, после чего весь кадр передается в течение одного временного слота. Если передача проходит без коллизии, повторная передача кадра не требуется (узел может подготовить к передаче новый кадр). В случае коллизии узел обнаруживает факт коллизии, прежде чем заканчивается данный слот. Затем при наступлении каждого последующего слота с вероятностьюp узел передает кадр повторно до тех пор, пока кадр не будет передан без коллизии.

Под повторной передачей кадра с вероятностью p мы подразумеваем, что узел как бы подбрасывает монетку. При этом кадр передается повторно, только если выпадает решка, что происходит с вероятностью p. При выпадении орла, что происходит с вероятностью $(1 - p)$, узел пропускает данный временной интервал и подбрасывает монетку еще раз. Все узлы, вовлеченные в коллизию, подбрасывают свои монетки независимо друг от друга.

Может показаться, что дискретный протокол ALOHA обладает рядом достоинств. В отличие от протоколов мультиплексирования с разделением канала, дискретный протокол ALOHA позволяет единственному активному в сети узлу передавать кадры без перерыва с максимальной скоростью R (узел называется активным, если у него есть кадр для передачи). Дискретный протокол ALOHA является в большой степени децентрализованным, так как каждый узел сам обнаруживает факт

коллизии и независимо от других узлов принимает решение о времени повторной передачи. (Однако для использования дискретного протокола ALOHA требуется синхронизация узлов. Далее мы кратко обсудим непрерывную версию протокола ALOHA, а также протоколы CSMA, не требующие подобной синхронизации и, таким образом, являющиеся полностью децентрализованными.) Кроме того, ALOHA представляет собой чрезвычайно простой протокол.

Дискретный протокол ALOHA хорошо работает в тех ситуациях, когда имеется только один активный узел, но какова его эффективность при наличии нескольких активных узлов? Эффективность дискретного протокола ALOHA снижается благодаря двум факторам. Во-первых, как показано на рис. 5.13, когда в сети несколько активных узлов, определенная доля слотов тратится впустую из-за коллизий. (Как показано на рисунке, в первом слоте в коллизию вовлекаются три узла. Затем узлу 2 удается передать кадр в четвертом слоте, узлу 1 — в восьмом слоте, а узлу 3 — в девятом.) Во-вторых, другая доля слотов тратится напрасно, когда все активные узлы одновременно отказываются от передачи. Дискретный протокол ALOHA работает продуктивно только в те слоты, когда передавать требуется ровно одному узлу. Слот, на протяжении которого передает только один узел, называется *успешным слотом*. Эффективность дискретного протокола коллективного доступа определяется долей успешных слотов в ситуации большого количества активных узлов, у каждого из которых всегда есть большое количество кадров для передачи. Обратите внимание, что, если вообще не использовать протокола коллективного доступа и после коллизии немедленно повторять передачу каждым из узлов, эффективность сети была бы равна нулю. Дискретный протокол ALOHA очевидно увеличивает эффективность сети, но насколько?

Рис. 5.13. Пример работы дискретного протокола ALOHA

Попытаемся определить максимальную эффективность дискретного протокола ALOHA. Чтобы упростить наши вычисления, немного изменим протокол, предположив, что каждый узел с вероятностью p пытается передать кадр с наступлением

каждого нового слота. То есть мы предполагаем, что у каждого узла всегда есть кадр для передачи и узел всегда с вероятностью р пытается передать кадр независимо от того, новый это кадр или передаваемый повторно. Пусть в сети будет *N* узлов. В этом случае слот оказывается успешным, если один из узлов передает, а *N* - 1 узлов воздерживаются от передачи. Вероятность того, что некий конкретный узел будет передавать, равна р. Вероятность того, что остальные *N*- 1 узлов не будут передавать, равна $(1 - p)^{N-1}$. Таким образом, вероятность того, что данному узлу удастся успешно передать кадр, равна $p(1 - p)^{N-1}$. Поскольку существует *N* узлов, вероятность того, что повезет одному (любому) из них, равна $Np(i - p)^{N-K}$

Таким образом, при наличии N активных узлов эффективность дискретного протокола ALOHA равна $Np(l - p)^{N-1}$. Чтобы определить максимальную эффективность протокола при iV активных узлах, нам нужно найти такое значение вероятности р*, при котором данное выражение достигает максимума (см. упражнения в конце этой главы, относящиеся к данной теме). А чтобы получить максимальную эффективность протокола для большого количества активных узлов, мы можем найти предел значения $Np*(l - p*)^{N-1}$ для значения *N*> стремящегося к бесконечности (опять же, см. упражнения в конце главы). Выполнив все эти вычисления, мы обнаружим, что максимальная эффективность протокола составляет *1/e* ~ 0,36788. Таким образом, когда у большого количества узлов имеется много кадров для передачи, то (в лучшем случае) только окбло 37 % слотов канал будет работать с пользой. То есть эффективная пропускная способность канала равна не *R* бит/с, а всего лишь 0,37 *R* бит/с! Оказывается, что около 37 % времени канал будет простаивать и еще около 26 % времени тратить на обработку коллизий. Представьте себе несчастного сетевого администратора, приобретшего дискретную систему ALOHA с пропускной способностью 100 Мбит/с и собирающегося использовать ее для обслуживания трафика между большим количеством пользователей с суммарной пропускной способностью около 80 Мбит/с! Несмотря на то что мгновенная пропускная способность канала равна 100 Мбит/с, его успешная пропускная способность составит менее 37 Мбит/с.

Чистый протокол ALOHA

Дискретный протокол ALOHA требует, чтобы все узлы синхронизировали время начала передачи кадров. Собственно, первый протокол ALOHA **[5]** не был дискретным, представляя собой полностью децентрализованный протокол. В так называемом чистом протоколе ALOHA, когда прибывает первый кадр (то есть дейтаграмма сетевого уровня передается на более низкий уровень передающего узла), узел немедленно передает весь кадр целиком в широковещательный канал. Если переданный кадр сталкивается с одним или несколькими другими кадрами, с вероятностью р узел немедленно передает кадр повторно. В противном случае узел выжидает в течение времени, необходимого для передачи одного кадра, после чего опять с вероятностью р передает кадр либо пережидает еще один интервал времени.

Чтобы определить максимальную эффективность чистого протокола ALOHA, сконцентрируем наше внимание на отдельном узле. Мы будем использовать те же допущения, что и в случае дискретного протокола ALOHA, и примем за единицу времени интервал (слот), требующийся для передачи одного кадра. В любой момент

времени вероятность того, что узел передает кадр, равна p. Предположим, передача этого кадра началась в момент времени t^0. Как видно из рис. 5.14, чтобы этот кадр был передан успешно, никакой другой узел не должен начать свою передачу во временном интервале $[t^0 - 1, £_0]$, так как иначе такая передача совпадет по времени с началом передачи нашего узла. Вероятность того, что остальные узлы не начнут передачу в течение этого интервала времени, равна $p(1-p)^{N-1}$. Аналогично, никакой другой узел не должен начать свою передачу, пока передает наш узел, так как такая передача также приведет к коллизии, но уже с концом нашего кадра. Вероятность этого события также равна $p(1-p)^{Nx}$. Таким образом, вероятность успешной передачи кадра данным узлом равна $p(1-p)^{2(ДМ)}$. При стремлении количества узлов к бесконечности максимальная эффективность чистого протокола ALOHA будет равна всего лишь 1/(2e), то есть половине от максимальной эффективности дискретного протокола ALOHA. Такова плата за полную децентрализацию.

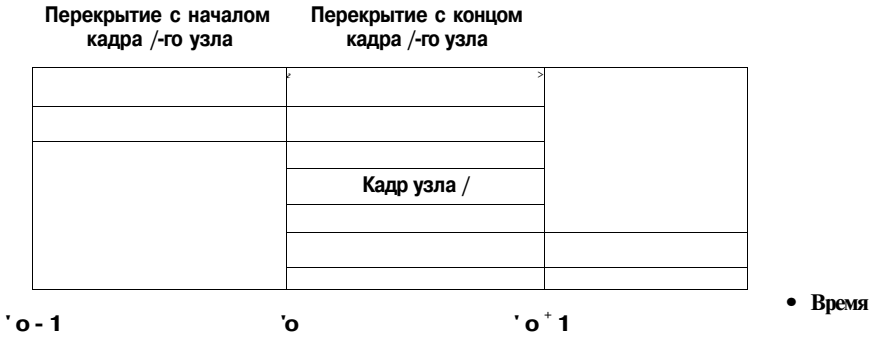


Рис. 5.14. Накладывающиеся передачи в чистой системе ALOHA

ИСТОРИЧЕСКАЯ СПРАВКА

Доктор философии Норм Абрамсон любил серфинг и интересовался коммутацией пакетов. Это сочетание увлечений привело его в 1969 году в Гавайский университет. Гавайи представляют собой множество гористых островков, на которых трудно установить традиционную локальную сеть. В свободное от серфинга время Абрамсон размышлял о том, как разработать сеть с коммутацией пакетов, передаваемых по радио. У спроектированной им сети был один центральный хост и несколько второстепенных узлов, разбросанных по Гавайским островам. У сети было два канала, для каждого из которых использовался свой частотный диапазон. По нисходящему широковещательному каналу пакеты рассылались от центрального хоста остальным узлам. По восходящему каналу остальные узлы посылали пакеты центральному хосту. Помимо информационных пакетов центральный хост также посылал подтверждения для каждого успешно принятого пакета.

Поскольку второстепенные узлы передавали пакеты децентрализованно, в восходящем канале неизбежно возникали коллизии. Это наблюдение натолкнуло Абрамсона на идею протокола ALOHA (чистого), описанного в этой главе. В 1970 году Абрамсон при финансовой поддержке управления ARPA соединил сеть ALOHAnet с сетью ARPAnet. Эта работа не только привела к рождению первой беспроводной сети с коммутацией пакетов, но еще и вдохновила Боба Меткалфа на создание на основе ALOHA протокола CSMA/CD и локальной Ethernet-сети.

Протокол CSMA

В обоих вариантах протокола ALOHA, дискретном и чистом, узел принимает решение о передаче кадра независимо от активности остальных узлов, присоединенных к широковещательному каналу. В частности, узел не обращает внимания на то, ведется ли в данный момент передача другими узлами, и не прекращает передачу в случае коллизий. Если вспомнить нашу аналогию с вечеринкой, протоколы ALOHA подобны невоспитанным собеседникам, прерывающим чужой разговор и продолжающим говорить, несмотря на то что в разговор вступили другие участники вечеринки. У людей также есть свои протоколы, позволяющие им не только вести себя более цивилизованно, но и тратить меньше времени на «коллизии» друг с другом и, таким образом, повышать «производительность» беседы. В частности, существуют два важных правила вежливого разговора.

- *Слушайте, прежде чем говорить.* Если кто-то уже говорит, подождите, пока он не закончит. В мире компьютерных сетей это правило называется *контролем несущей* и заключается в том, что узел прослушивает канал перед тем, как начать передачу. Если по каналу передается кадр, узел выжидает («отступает») в течение случайного периода времени, после чего снова опрашивает канал. Если канал оказывается свободным, узел начинает передачу кадра. В противном случае узел ждет в течение еще одного случайного интервала времени и повторяет весь процесс.
- *Если кто-то начал говорить, прекращайте разговор.* В мире компьютерных сетей это правило называется *обнаружением коллизий*. Оно заключается в том, что во время передачи узел прослушивает канал. Если он обнаруживает, что другой узел в этот момент времени тоже ведет передачу, он прекращает свою передачу и с помощью протокола вычисляет время своей следующей попытки передачи.

Эти два правила реализованы в семействе протоколов *CSMA* (Carrier Sense Multiple Access — множественный доступ с контролем несущей) и *CSMA/CD* (CSMA with Collision Detection — множественный доступ с контролем несущей и обнаружением коллизий). Было разработано множество вариантов протоколов CSMA и CSMA/CD, с деталями которых можно ознакомиться в [271, 289, 329, 424]. Кроме того, протокол CSMA/CD, применяемый в Ethernet-сети, мы подробно обсудим в разделе «Ethernet», а пока рассмотрим некоторые фундаментальные характеристики протоколов CSMA и CSMA/CD.

Первый вопрос о протоколе CSMA, который может возникнуть, состоит в том, как вообще могут возникать коллизии, если все узлы прослушивают несущую? В самом деле, ведь узел воздерживается от передачи, если он замечает, что канал занят. Ответ на этот вопрос лучше всего проиллюстрировать с помощью пространственно-временной диаграммы [336]. На рис. 5.15 показана пространственно-временная диаграмма работы четырех узлов (A, B, C, D), присоединенных к линейной широковещательной шине. На горизонтальной оси фиксируется положение каждого узла в пространстве, вдоль вертикальной оси изменяется время.

В момент времени t^0 узел B опрашивает канал и приходит к выводу, что канал свободен, так как никакой другой узел в этот момент не ведет передачу. Поэтому узел B начинает передачу, и передаваемый им сигнал распространяется по широ-

ковещательному носителю в обоих направлениях со скоростью, близкой к скорости света, но конечной. В момент времени t^{l} ($t^{l} > t^{0}$) у узла D появляется кадр для передачи. Хотя узел B в этот момент уже передает данные, передаваемый им сигнал еще не достиг узла D, таким образом, узел D полагает, что канал свободен. Поэтому в соответствии с протоколом CSMA узел D начинает передачу своего кадра. Немного позднее сигнал, передаваемый узлом B, смешивается с сигналом узла D — возникает коллизия. Из рисунка видно, что производительность широковещательного канала в значительной степени зависит от *времени прохождения сигнала* по каналу от одного узла до другого. Чем больше это время, тем выше вероятность коллизий, вызванных тем, что сигнал уже начавшего передачу узла не успел достичь другого узла, готового к передаче.

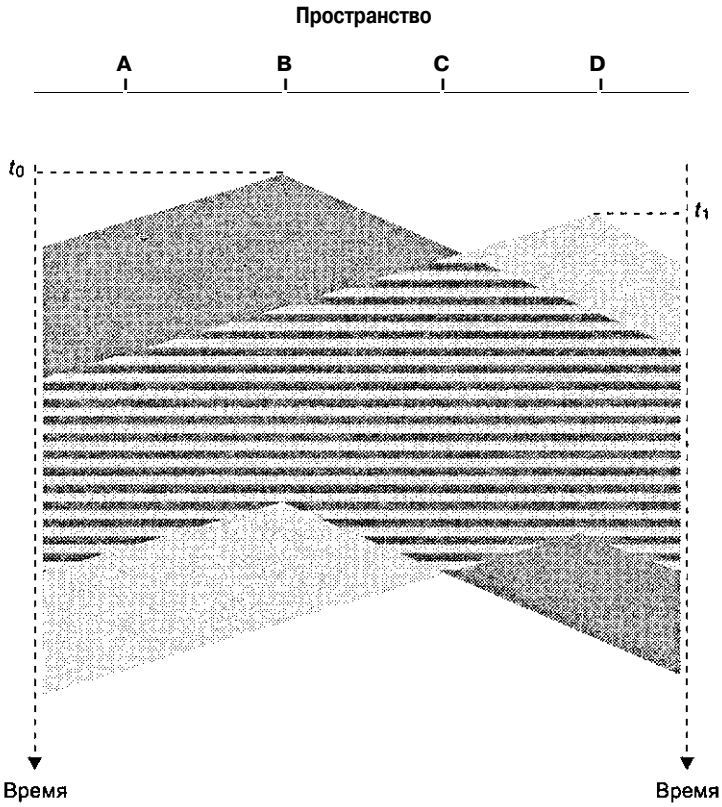


Рис. 5.15. Пространственно-временная диаграмма коллизий в канале с протоколом CSMA

Изображенные на рис. 5.15 узлы не обнаруживают коллизию. Оба узла (B и D) продолжают передавать свои кадры целиком и после коллизии. При использовании протокола с обнаружением коллизий узел прекращает передачу, как только он обнаруживает коллизию. На рис. 5.16 показан тот же сценарий, что и на рис. 5.15, но в этом случае узлы прекращают передачу, обнаружив коллизию. Очевидно, добавление способности обнаружения коллизий повышает производительность про-

токола, так как продолжение передачи кадров в случае коллизии не имеет смысла и приводит лишь к дополнительным потерям времени. Протокол Ethernet, который будет рассматриваться в разделе «Ethernet», представляет собой протокол CSMA с обнаружением коллизий.

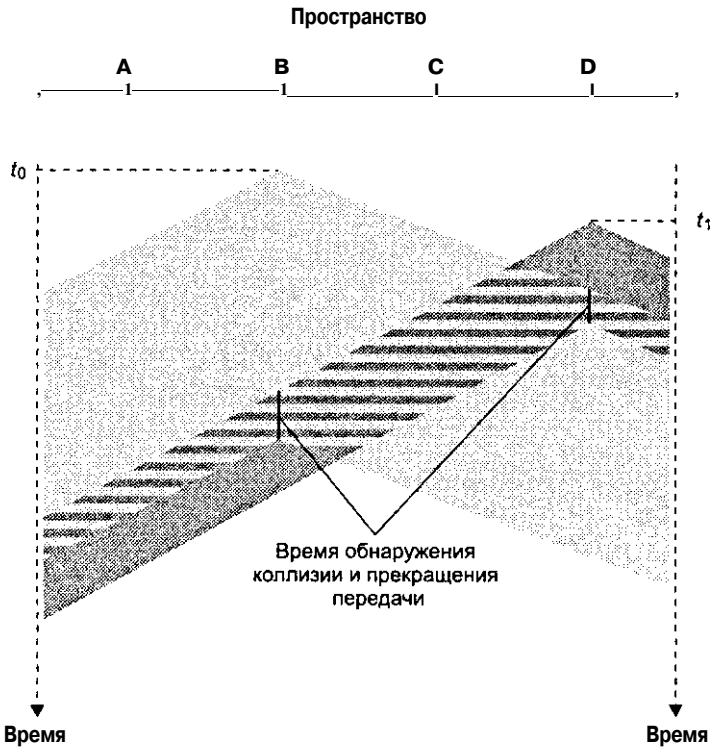


Рис. 5.16. Протокол CSMA с обнаружением коллизий

Протоколы последовательного доступа

Как уже упоминалось, двумя желательными свойствами протокола коллективного доступа являются, во-первых, возможность единственного активного узла передавать свои данные с максимальной пропускной способностью канала R бит/с, во-вторых, возможность для каждого из M активных узлов передавать свои данные со скоростью R/M бит/с. Протоколы ALOHA и CSMA удовлетворяют первому требованию, но не удовлетворяют второму. Это подвигло исследователей на создание нового класса протоколов — *протоколов последовательного доступа*. Как и в случае с протоколами произвольного доступа, существуют десятки протоколов последовательного доступа, и у каждого есть множество вариантов. Здесь мы рассмотрим два наиболее важных протокола последовательного доступа. Первый из них — *протокол опроса*. При использовании протокола опроса один из узлов должен быть назначен главным (управляющим) узлом. Главный узел поочередно *опрашивает* все узлы. Например, сначала главный узел посылает сообщение узлу 1,

сообщая ему, что он может передать некоторое максимальное количество кадров. После того как узел 1 передает несколько кадров, главный узел разрешает передать некоторое количество кадров узлу 2. (Главный узел может определить момент завершения передачи очередным узлом по отсутствию сигнала в канале.) Данная процедура продолжается бесконечно, при этом главный узел в цикле опрашивает все узлы.

Протокол опроса устраняет коллизии и пустые слоты, от которых страдают протоколы произвольного доступа. Таким образом, эффективность протокола опроса значительно выше. Однако у протокола опроса есть несколько недостатков. Первый недостаток заключается в том, что определенное время тратится протоколом опроса на саму процедуру опроса, то есть на выдачу узлу разрешения на передачу. Например, если только один узел является активным, тогда он не сможет передавать со средней скоростью, равной полной пропускной способности канала, так как после передачи активным узлом разрешенного количества кадров главный узел будет опрашивать остальные узлы в каждом цикле. Второй недостаток является даже более серьезным. Он заключается в том, что при выходе из строя главного узла вся деятельность канала прекращается.

Второй протокол последовательного доступа — *протокол с передачей маркера*. В этом протоколе главного узла не существует. Все узлы, присоединенные к широковещательному каналу, обмениваются небольшим специальным кадром, называемым *маркером*. Порядок обмена маркера фиксирован. Например, узел 1 всегда посылает маркер узлу 2, а узел 2 всегда посылает маркер узлу 3 и т. д.; а узел *N* всегда посылает маркер узлу 1. Получив маркер, узел удерживает его, только если у него есть данные для передачи; в противном случае он немедленно передает маркер следующему узлу. Если к моменту получения маркера у узла есть кадры для передачи, он передает некое максимальное количество кадров, после чего передает маркер следующему узлу. Передача маркера осуществляется децентрализованно и обладает высокой эффективностью. Но проблемы могут возникнуть и в данной схеме. Например, выход из строя одного узла может вывести из строя весь канал, а если какой-либо узел забудет передать маркер, потребуется специальная процедура вывода канала из тупиковой ситуации. За многие годы было разработано множество протоколов с передачей маркера, в каждом из которых приходилось решать эти и другие неприятные вопросы.

Локальные сети

Протоколы коллективного доступа используются в самых разных типах широковещательных каналов, а также в спутниковых и беспроводных каналах, узлы которых ведут передачу в радиодиапазоне, и в восходящих каналах при кабельном доступе к Интернету (см. раздел «Интернет-провайдеры и магистрали Интернета» в главе 1). Кроме того, протоколы коллективного доступа очень популярны в локальных сетях.

Как уже упоминалось, локальная сеть представляет собой компьютерную сеть, географически сконцентрированную в одном здании или в комплексе зданий. Когда пользователь выходит в Интернет из здания университета или корпорации, доступ

почти всегда осуществляется через локальную сеть. При подобном типе доступа к Интернету хост пользователя представляет собой узел локальной сети, а локальная сеть предоставляет доступ к Интернету через маршрутизатор, как показано на рис. 5.17. Здесь локальная сеть изображается как одна линия связи (один канал) между каждым пользовательским хостом и маршрутизатором. В линии связи используется протокол канального уровня (уровня передачи данных), частью которого является протокол коллективного доступа. Скорость передачи R в большинстве локальных сетей очень высока. Уже в начале 80-х годов популярными были локальные сети, работавшие на скорости 10 Мбит/с; сегодня широкое распространение получили сети со скоростями в 100 Мбит/с, но также существуют сети, работающие на скорости в 1 Гбит/с.

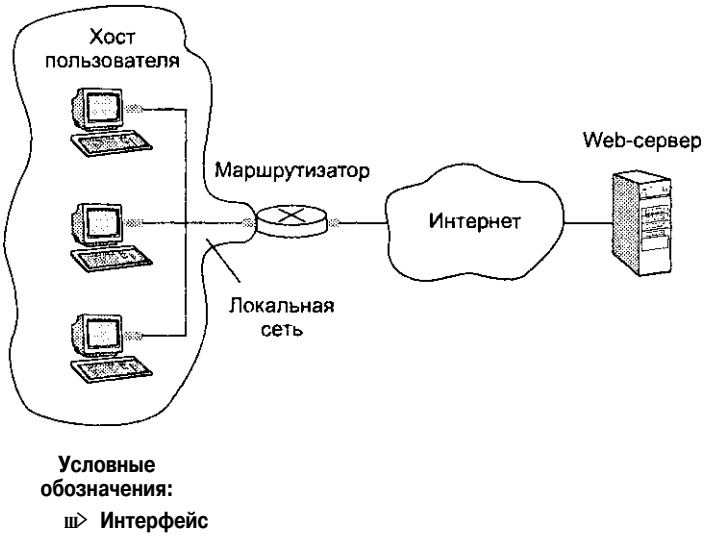


Рис. 5.17. Локальная сеть, соединенная с Интернетом через маршрутизатор

В 80-е годы и в начале 90-х популярными были два класса технологий локальных сетей. Первый класс технологий состоял из локальных сетей Ethernet (также известных по номеру стандарта IEEE 802.3 [477]), в которых применяется протокол произвольного доступа. Второй класс локальных сетей основан на протоколах передачи маркера, к которым относятся сети типа Token ring (маркерное кольцо), также известные как сети стандарта IEEE 802.5, и протоколе FDDI (Fiber Distributed Data Interface — распределенный интерфейс передачи данных по волоконно-оптическим каналам) [249]. Поскольку подробно технология Ethernet будет рассматриваться в одноименном разделе, здесь мы сконцентрируемся на локальных сетях с передачей маркера. Мы намеренно сократим обсуждение технологии передачи маркера, так как эти сети не выдержали конкуренции с технологией Ethernet и в настоящее время значительно уступают им по популярности. Тем не менее следует сказать несколько слов о сетях типа Token ring, чтобы на примере продемонстрировать технологию передачи маркера, а также дать представление об исторической перспективе.

В сетях типа Token ring *N* узлов локальной сети (хосты и маршрутизаторы) соединены в кольцо прямыми линиями связи. Порядок передачи маркера определяется топологией маркерного кольца. Узел получает маркер, передает кадр, после чего посылает маркер дальше. Таким образом, маркер обходит все кольцо, благодаря чему создается виртуальный широковещательный канал. Узел, передающий кадр, отвечает за удаление кадра из кольца. Протокол FDDI-был разработан для крупных локальных сетей с линиями большой протяженности, а также для так называемых *региональных сетей* (Metropolitan Area Network, MAN). В более крупных локальных сетях (распространенных на несколько километров) позволять кадру возвращаться к пославшему его узлу, после того как кадр достиг узла, которому он предназначался, неэффективно. В протоколе FDDI кадр из кольца удаляет узел-получатель. (Строго говоря, FDDI не является широковещательным каналом в чистом виде, так как передаваемый кадр получает не каждый узел.)

Адресация в локальных сетях и протокол ARP

Как было показано в предыдущем разделе, узлы локальной сети (Local Area Network, LAN) посылают друг другу кадры по широковещательному каналу. Это означает, что кадр, переданный одним узлом локальной сети, принимается всеми остальными узлами этой локальной сети. Но, как правило, узлу локальной сети нужно передать кадр не *всем* узлам сети, а одному *определенному* узлу. Чтобы предоставить ему такую возможность, у узлов локальной сети должны быть адреса, и адрес получателя должен указываться в поле кадра канального уровня. В этом случае, получив кадр, узел может определить, предназначается этот кадр ему или какому-то другому узлу локальной сети.

- Если адрес получателя в кадре совпадает с адресом получившего этот кадр узла, тогда узел извлекает из кадра канального уровня дейтаграмму сетевого уровня и передает ее вверх по стеку протоколов.
- Если адрес получателя в кадре не совпадает с адресом получившего этот кадр узла, тогда узел просто отбрасывает этот кадр.

Адресация в локальных сетях

В действительности адрес в локальной сети есть не у узла, а у сетевого адаптера. Это иллюстрирует рис. 5.18. *Адрес в локальной сети,* или *LAN-адрес,* также называют *физическим адресом, Ethernet-адресом* или *MAC-адресом* (Media Access Control — управление доступом к носителю). В большинстве локальных сетей (включая Ethernet-сети и сети с передачей маркера) LAN-адрес представляет собой 6-байтовое число, что позволяет использовать 2^{48} возможных адресов. Как показано на рис. 5.18, эти 6-байтовые адреса, как правило, изображаются в шестнадцатеричном виде, при этом каждый байт адреса записывается как пара шестнадцатеричных цифр. Адрес адаптера в локальной сети является постоянным. Этот адрес прошивается в постоянной памяти адаптера при его изготовлении.

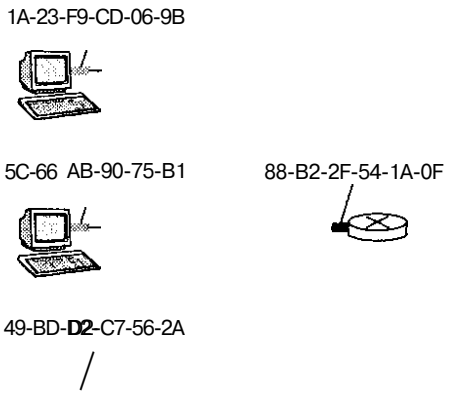


Рис. 5.18. У каждого адаптера, соединенного с локальной сетью, есть уникальный адрес

Одно интересное свойство LAN-адресов заключается в том, что не может существовать двух адаптеров с одинаковыми адресами. Это может показаться удивительным, так как сетевые адаптеры производятся в разных странах, разными производителями. Как может компания, производящая адаптеры в Тайване, гарантировать, что адреса ее адаптеров будут отличаться от адресов адаптеров, производимых другой компанией в Бельгии? Дело в том, что физическим адресным пространством управляет институт IEEE (Institute of Electrical and Electronics Engineers — Институт инженеров по электротехнике и электронике). В частности, когда компания хочет выпускать адаптеры, она приобретает блок адресного пространства, состоящий из 2^{24} адресов. IEEE выделяет блок из 2^{24} адресов, фиксируя старшие 24 бита физического адреса и позволяя компании создавать уникальные комбинации из младших 24 разрядов для каждого адаптера.

Таким образом, LAN-адреса адаптеров образуют плоскую (в отличие от иерархической) структуру и не изменяются при перемещении адаптеров. У мобильного компьютера с Ethernet-картой всегда один и тот же LAN-адрес независимо от того, где находится этот компьютер. Вспомним, что IP-адреса, напротив, образуют иерархическую структуру (то есть делятся на сетевую и хостовую части), и при перемещении хоста IP-адрес узла должен быть изменен. Таким образом, LAN-адрес адаптера аналогичен номеру карточки социального обеспечения (эти номера также образуют плоскую адресную структуру и не изменяются при перемещении их владельца). IP-адрес можно сравнить с почтовым адресом человека, являющимся иерархическим и изменяющимся при смене места жительства владельца.

Как упоминалось в начале этого раздела, когда адаптер хочет переслать кадр другому адаптеру в той же локальной сети, передающий адаптер помещает в кадр адрес получателя в локальной сети. Получив кадр, принимающий адаптер извлекает из него дейтаграмму и передает ее вверх по стеку протоколов. Все остальные адаптеры этой локальной сети также получают этот кадр, но они отбрасывают его, не передавая дейтаграммы сетевого уровня вверх по стеку протоколов. Таким образом, эти адаптеры не прерывают свой родительский узел, получив кадр, предна-

значенный другому узлу. Однако иногда передающий узел бывает заинтересован в том, чтобы все адаптеры в локальной сети приняли и обработали кадр, который он посылает. В этом случае передающий адаптер помещает в поле адреса получателя специальный *широковещательный адрес*. В локальных сетях, использующих 6-байтовые адреса (как, например, Ethernet или локальные сети с передачей маркера), широковещательный адрес представляет собой строку из 48 двоичных единиц (то есть FF-FF-FF-FF-FF-FF в шестнадцатеричной нотации).

ПРИНЦИПЫ И ПРАКТИКА

Есть несколько причин, по которым узлам помимо адресов сетевого уровня выделяются LAN-адреса. Во-первых, локальные сети разрабатываются для работы с произвольными протоколами сетевого уровня, а не только с протоколом IP и Интернетом. Если бы вместо «нейтральных» LAN-адресов адаптерам назначались IP-адреса, адаптерам было бы трудно поддерживать другие протоколы сетевого уровня (например, IPX или DECnet). Во-вторых, если бы адаптеры должны были использовать IP-адреса вместо LAN-адресов, тогда адрес сетевого уровня пришлось бы хранить в оперативной памяти адаптера и перенастраивать его при каждом перемещении адаптера или при включении питания. Правда, можно было бы вообще не использовать адреса в адаптерах и передавать данные (как правило, IP-дейтаграммы) каждого полученного адаптером кадра родительскому узлу. После этого родительский узел мог бы проверять соответствие адреса сетевого уровня. Недостаток такого варианта состоит в том, что в этом случае родительский узел прерывался бы каждым кадром, посланным по локальной сети, включая кадры, предназначающиеся другим узлам той же широковещательной сети. Таким образом, для большей независимости уровней в стеке протоколов уровни должны обладать собственными адресными схемами. Мы уже ознакомились с тремя типами адресов: именами хостов на прикладном уровне, IP-адресами на сетевом уровне и LAN-адресами на канальном уровне.

Протокол ARP

При передаче дейтаграмм одновременно используются адреса сетевого уровня (например, IP-адреса Интернета) и адреса канального уровня (то есть LAN-адреса), поэтому возникает необходимость в преобразовании одних адресов в другие. В Интернете эту работу выполняет *протокол ARP* (Address Resolution Protocol — протокол разрешения адресов). У каждого хоста, подключенного к Интернету, и маршрутизатора, соединенного с локальной сетью, есть *ARP-модуль* (RFC 826).

Передача дейтаграммы узлу в пределах локальной сети

Чтобы понять, зачем нужен протокол ARP, рассмотрим сеть, изображенную на рис. 5.19. В этом простом примере у каждого узла есть IP-адрес, а у адаптера каждого узла есть LAN-адрес. IP-адреса, как обычно, представляются в виде четырех десятичных чисел, а LAN-адреса показаны в шестнадцатеричном виде. Теперь предположим, что узел с IP-адресом 222.222.222.220 хочет послать IP-дейтаграмму узлу 222.222.222.222. Чтобы выполнить эту задачу, передающий узел должен передать адаптеру не только IP-дейтаграмму, но также LAN-адрес узла 222.222.222.222. Получив IP-дейтаграмму и LAN-адрес узла, адаптер передающего узла формирует кадр канального уровня, содержащий LAN-адрес принимающего узла, и передает

кадр в локальную сеть. Но каким образом передающий узел определяет LAN-адрес узла с IP-адресом 222.222.222.222? Он делает это с помощью модуля ARP, передавая ему IP-адрес 222.222.222.222. На это ARP-модуль отвечает соответствующим LAN-адресом узла, то есть адресом 49-BD-D2-C7-56-2A.

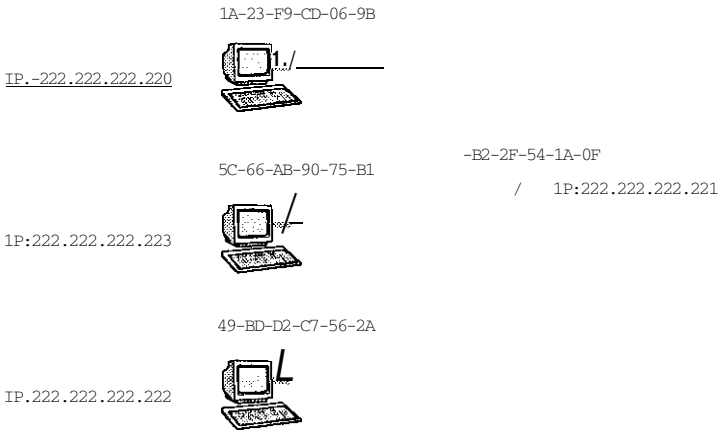


Рис. 5.19. У каждого узла локальной сети есть IP-адрес, а у каждого адаптера узла есть LAN-адрес

Итак, ARP-модуль преобразует IP-адрес в LAN-адрес узла. Во многом это аналогично системе DNS (см. раздел «Служба трансляции имен Интернета» в главе 2), преобразующей имена хостов в IP-адреса. Однако важное различие между этими двумя схемами преобразования адресов заключается в том, что DNS преобразует имена хостов в IP-адреса во всем Интернете, тогда как протокол ARP занимается только IP-адресами в пределах одной локальной сети. Если бы узел в Калифорнии попытался узнать LAN-адрес для IP-адреса узла в Миссисипи, протокол ARP вернул бы ошибку.

Теперь, выяснив, что делает протокол ARP, посмотрим, как он это делает. У ARP-модуля каждого узла есть оперативная память, в которой хранится ARP-таблица. В этой таблице прописаны IP-адреса хостов локальной сети и соответствующие им LAN-адреса. Ниже показан пример ARP-таблицы для узла 222.222.222.220 (табл. 5.1). Для каждой пары адресов в таблице также содержится поле времени жизни (Time To Live, TTL), в котором указывается, когда данная запись будет удалена из таблицы. Обратите внимание, что в таблице не обязательно содержатся записи для всех узлов локальной сети. Например, записи для одних узлов могут быть удалены, так как время их жизни истекло, тогда как записи для других узлов вообще могут никогда не попасть в эту таблицу. Типичное значение времени жизни — 20 мин с момента помещения записи в ARP-таблицу.

Теперь предположим, что узел 222.222.222.220 хочет отправить дейтаграмму другому узлу той же локальной сети. Для этого передающему узлу нужно по IP-адресу получающего узла узнать его LAN-адрес. Эта задача несложная, если в ARP-таблице передающего узла содержится запись для узла-получателя. Но что делать,

если такой записи в ARP-таблице нет? Например, пусть узел 222.222.222.220 желает переслать дейтаграмму узлу 222.222.222.222. В этом случае передающий узел определяет нужный ему адрес при помощи протокола ARP. Сначала передающий узел формирует специальный *ARP-пакет*. В ARP-пакете содержится несколько полей, среди которых есть IP-адреса и LAN-адреса передающего и принимающего узлов. Для обоих ARP-пакетов (запроса и ответа) используется один и тот же формат. Цель ARP-пакета с запросом состоит в том, чтобы опросить все остальные узлы локальной сети и определить LAN-адрес, соответствующий интересующему нас IP-адресу.

Таблица 5.1. Пример ARP-таблицы для узла с LAN-адресом 222.222.222.220

IP-адрес	LAN-адрес	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Итак, в нашем примере узел 222.222.222.220 передает ARP-пакет с запросом своему адаптеру вместе с указанием переслать этот пакет по широковещательному LAN-адресу FF-FF-FF-FF-FF-FF. Адаптер помещает ARP-пакет в кадр канального уровня, указывает широковещательный адрес в поле адреса получателя и передает кадр в локальную сеть. Если вспомнить нашу аналогию с номерами карточек социального страхования и почтовыми адресами, то можно заметить, что ARP-запрос аналогичен человеку, выкрикивающему в переполненной комнате общежития некоторой компании (например, АпуСогр): «Какой номер карточки социального страхования у человека, проживающего в комнате 112 общежития 13 компании АпуСогр, Пало-Альто, Калифорния?» Кадр с ARP-запросом принимается всеми остальными адаптерами локальной сети, и (поскольку в запросе использовался широковещательный адрес) каждый адаптер передает содержащийся в кадре ARP-пакет своему узлу. Каждый узел проверяет, совпадает ли его IP-адрес с указанным IP-адресом получателя в ARP-пакете. Узел, IP-адрес которого совпадает с указанным в пакете, посылает запрашивающему узлу ответный ARP-пакет с указанным в нем соответствующим LAN-адресом. После этого запрашивающий узел 222.222.222.220 может обновить свою ARP-таблицу и отправить 1P-дейтаграмму.

Следует сделать два интересных замечания о протоколе ARP. Во-первых, ARP-запрос посылается в широковещательном кадре, а ответ передается в стандартном кадре. Прежде чем продолжить чтение, подумайте, почему. Во-вторых, в протоколе ARP реализован принцип самонастройки (plug-and-play), так как ARP-таблица узла формируется автоматически — она не должна настраиваться системным администратором. И если узел отсоединяется от локальной сети, соответствующая ему запись по истечении времени жизни удаляется из таблицы.

Передача дейтаграммы узлу за пределы локальной сети

Теперь должно быть ясно, как работает протокол ARP, когда узел хочет послать дейтаграмму другому узлу *той же самой* локальной сети (то есть той же IP-сети,

согласно терминологии главы 4). Но теперь мы рассмотрим более сложную ситуацию, в которой узел локальной сети хочет послать дейтаграмму сетевого уровня узлу, находящемуся *за пределами* локальной сети (то есть в другую IP-сеть). Обсудим этот вопрос на примере сети, состоящей из двух локальных сетей, соединенных маршрутизатором (рис. 5.20).

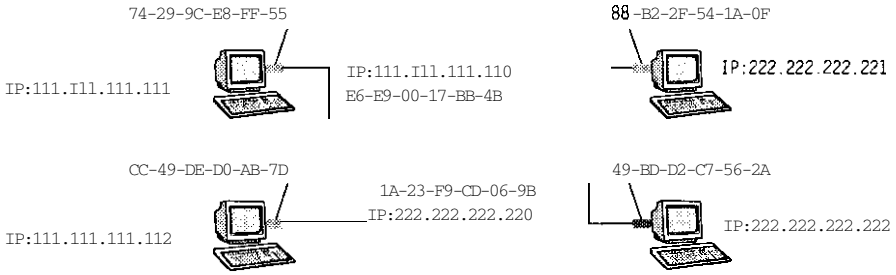


Рис. 5.20. Две локальные сети, соединенные маршрутизатором

Во-первых, обратите внимание, что существуют два типа узлов: хосты и маршрутизаторы. У каждого хоста есть ровно один IP-адрес и один адаптер. Но, как было показано в разделе «Интернет-протокол» главы 4, у маршрутизатора есть по одному IP-адресу для *каждого* интерфейса. У каждого интерфейса маршрутизатора есть собственный ARP-модуль и собственный адаптер. У изображенного на рисунке маршрутизатора два интерфейса, поэтому у него два IP-адреса, два ARP-модуля и два адаптера. Разумеется, у каждого адаптера есть свой LAN-адрес.

Обратите также внимание, что адреса всех интерфейсов, соединенных с локальной сетью 1, имеют вид 111.111.111.xxx, а адреса всех интерфейсов, соединенных с локальной сетью 2, имеют вид 222.222.222.xxx. В данном примере первые три байта IP-адреса обозначают «сеть», а последний байт указывает на конкретный интерфейс в сети. В нотации CIDR, использовавшейся в главе 4, локальная сеть 1 имеет сетевой адрес 111.111.111.000/24, а локальная сеть 2 — сетевой адрес 222.222.222.000/24.

Теперь предположим, что хост 111.111.111.111 хочет переслать IP-дейтаграмму хосту 222.222.222.222. Передающий хост, как обычно, отправляет дейтаграмму своему адаптеру. Но при этом передающий хост должен указать адаптеру LAN-адрес. Какой LAN-адрес следует использовать? Кое-кто может подумать, что это должен быть локальный адрес адаптера хоста 222.222.222.222, то есть 49-BD-D2-C7-56-2A. Однако это неверно. Если передающий адаптер будет использовать этот LAN-адрес, тогда ни один из адаптеров локальной сети 1 не станет передавать переданную IP-дейтаграмму своему сетевому уровню, так как адрес получателя в кадре не совпадет с LAN-адресом ни одного адаптера локальной сети 1. Переданная дейтаграмма просто умрет и попадет в свой дейтаграммный рай.

Если внимательно посмотреть на рис. 5.20, то можно заметить, что для передачи дейтаграммы в локальную сеть 2 ее нужно направлять интерфейсу марш-

рутизатора по адресу 111.111.111.110. Как показано в разделе 4.4, в таблице продвижения данных (таблице маршрутизации) хоста 111.111.111.111 должно быть указано, что дейтаграммы, предназначающиеся хосту 222.222.222.222, следует посылать интерфейсу маршрутизатора по адресу 111.111.111.110. Таким образом, в поле локального адреса получателя кадра следует указывать адрес адаптера интерфейса маршрутизатора 111.111.111.110, то есть E6-E9-00-17-BB-4B. Как передающему хосту узнать локальный адрес узла 111.111.111.110? С помощью протокола ARP, разумеется! Как только передающий адаптер получает этот локальный адрес, он создает кадр и посылает его в локальную сеть 1. Адаптер маршрутизатора в локальной сети 1 видит, что данный кадр канального уровня адресован ему, и поэтому передает кадр сетевому уровню маршрутизатора. Ура! IP-дейтаграмма была успешно перемещена от хоста-отправителя на маршрутизатор! Но мы еще не закончили. Нам нужно еще переслать дейтаграмму от маршрутизатора получателю. Теперь маршрутизатор должен определить, по какому интерфейсу следует переслать дейтаграмму. Как показано в разделе «Интернет-протокол» главы 4, выбор делается при помощи таблицы маршрутизации, хранящейся на маршрутизаторе. В этой таблице маршрутизатор находит запись, по которой определяет, что дейтаграмму следует отправить через интерфейс 222.222.222.220. Этот интерфейс передает дейтаграмму своему адаптеру, который помещает дейтаграмму в новый кадр и посылает этот кадр в локальную сеть 2. Теперь уже LAN-адрес кадра указывает на его конечного получателя. И как же маршрутизатор узнает его LAN-адрес? При помощи протокола ARP, конечно!

Протокол ARP для Ethernet-сети определен в RFC 826. Прекрасное введение в тему ARP имеется в RFC 1180. Мы вернемся к протоколу ARP в упражнениях, представленных в конце главы.

Ethernet

Технология Ethernet очень популярна на рынке локальных сетей. В 80-е годы и в начале 90-х технологии Ethernet приходилось конкурировать с множеством альтернативных технологий локальных сетей, включая сети Token ring, FDDI и ATM. Некоторым из этих технологий удалось на несколько лет захватить часть рынка. Но технология Ethernet, разработанная в середине 70-х, продолжает свой рост и развитие до сих пор, а в последние годы она прочно заняла доминирующее положение на рынке. Сегодня Ethernet представляет собой превалирующую технологию локальных сетей, и, похоже, такая ситуация сохранится в обозримом будущем. Значение Ethernet в локальных сетях так же велико, как и значение Интернета в глобальных сетях.

Успеху технологии Ethernet способствовало множество причин. Во-первых, Ethernet была первой локальной сетью, получившей широкое распространение. Благодаря этому сетевые администраторы очень близко познакомились с технологией Ethernet и уже не хотели переходить на другие технологии локальных сетей, когда те появились на рынке. Во-вторых, такие технологии, как Token ring, FDDI

и ATM, были более сложными и дорогими, чем Ethernet, что еще больше препятствовало переходу на них. В-третьих, наиболее сильным стимулом перехода на другие технологии локальных сетей (например, FDDI или ATM), как правило, была более высокая скорость передачи данных. Однако Ethernet всегда отвечала ударом на удар, и каждая новая версия Ethernet не уступала конкурентам, а то и превосходила их по данному параметру. В начале 90-х была представлена коммутируемая Ethernet-сеть, что позволило снова повысить эффективную скорость передачи данных. Наконец, благодаря популярности Ethernet аппаратура Ethernet (адаптеры, хабы и коммутаторы) стала производиться массовым тиражом, в результате чего цены на нее упали до невероятно низкого уровня. Причиной столь низких цен также является тот факт, что протокол коллективного доступа, применяемый в сетях Ethernet, полностью децентрализован, что обусловливает простоту аппаратуры.

Оригинальная локальная Ethernet-сеть была придумана Бобом Меткалфом и Дэвидом Боггсом в середине 70-х. Показанная на рис. 5.21 схема привела к появлению стандарта 10Base5 Ethernet, в который входил интерфейсный кабель, соединявший адаптер Ethernet (то есть интерфейс) с внешним приемопередатчиком. Ethernet-сети посвящен превосходный web-сайт, созданный Чарльзом Сперджоном (http://www.ethermanage.com/ethernet/ethernet.html).

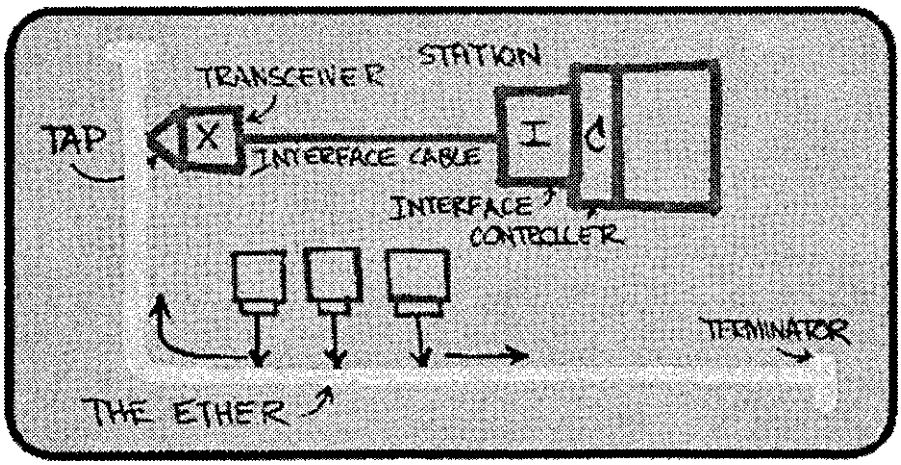


Рис. 5.21. Оригинальная схема Меткалфа

Основы технологии Ethernet

Сегодня Ethernet-сеть существует во множестве видов и форм. Топологией локальной Ethernet-сети может быть шина или звезда. В качестве физического носителя в локальных сетях Ethernet применяются коаксиальный кабель, медная витая пара, оптоволокно. Кроме того, стандартами локальной Ethernet-сети поддерживается передача данных на скоростях 10 Мбит/с, 100 Мбит/с, 1 Гбит/с

и даже 10 Гбит/с. Однако несмотря на это многообразие все технологии Ethernet обладают некоторыми общими важными характеристиками. Прежде чем перейти к изучению других технологий, рассмотрим сначала эти общие характеристики.

Структура Ethernet-кадра

Итак, что общего у всех технологий Ethernet, имеющихся сегодня на рынке? Что объединяет их под одним именем? Во-первых, и прежде всего, это структура Ethernet-кадра. Во всех технологиях Ethernet независимо от используемого носителя и скорости передачи данных одна и та же структура кадра (рис. 5.22).

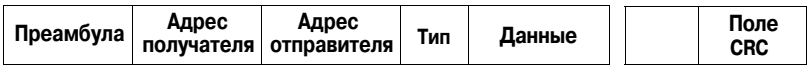


Рис. 5.22. Структура Ethernet-кадра

Разобравшись в структуре кадра, мы довольно много узнаем о технологии Ethernet. Чтобы сделать наше обсуждение более предметным, рассмотрим передачу IP-дейтаграммы от одного хоста другому в ситуации, когда оба хоста находятся в одной и той же локальной Ethernet-сети. (Отметим, однако, что Ethernet может переносить и другие пакеты сетевого уровня.) Пусть физический адрес передающего адаптера A равен AA-AA-AA-AA-AA-AA, а физический адрес принимающего адаптера, адаптера B — BB-BB-BB-BB-BB-BB. Передающий адаптер помещает IP-дейтаграмму в Ethernet-кадр и передает кадр физическому уровню. Принимающий адаптер получает кадр от физического уровня, извлекает IP-дейтаграмму и передает ее сетевому уровню. Рассмотрим в этом контексте шесть полей Ethernet-кадра.

- *Поле данных* (от 46 до 1500 байт). Это поле содержит IP-дейтаграмму. Максимальная единица передачи (Maximal Transfer Unit, MTU) в Ethernet-сети составляет 1500 байт. Это означает, что если размер IP-дейтаграммы превышает 1500 байт, тогда хост должен разбить ее на отдельные фрагменты (см. подраздел «Фрагментация IP-дейтаграмм» в разделе «Интернет-протокол» главы 4). Минимальный размер поля данных равен 46 байт. Это означает, что если размер IP-дейтаграммы меньше 46 байт, то данные, помещаемые в это поле, дополняются байтами-заполнителями. При этом сетевой уровень получает дейтаграмму от канального уровня с этими дополнительными байтами и отсекает все лишнее сам, ориентируясь на поле длины в заголовке IP-дейтаграммы.
- *Адрес получателя* (6 байт). Это поле содержит LAN-адрес принимающего адаптера, а именно BB-BB-BB-BB-BB-BB. Получив Ethernet-кадр с адресом получателя, *отличным* от собственного физического адреса или широковещательного адреса локальной сети, адаптер B отбрасывает кадр. В противном случае он передает содержимое поля данных сетевому уровню.
- *Адрес отправителя* (6 байт). Это поле содержит LAN-адрес адаптера, передающего кадр в локальную сеть, а именно AA-AA-AA-AA-AA-AA.

- *Поле типа* (2 байта). Поле типа позволяет локальной Ethernet-сети «мультиплексировать» протоколы сетевого уровня. Чтобы понять, что это означает, вспомним, что хосты могут помимо протокола IP использовать и другие протоколы. В самом деле, любой хост может поддерживать несколько протоколов сетевого уровня — разные протоколы для разных приложений. По этой причине, получив Ethernet-кадр, адаптер В должен знать, какому протоколу сетевого уровня он должен передать (то есть демультиплексировать) содержимое поля данных. Каждому сетевому протоколу (например, IP, Novell IPX или AppleTalk) присвоен зафиксированный в стандарте номер. Обратите внимание, что поле типа аналогично полю протокола в дейтаграмме сетевого уровня и полю номера порта сегмента транспортного уровня. Все эти поля служат для связи протокола одного уровня с протоколом уровнем выше.
- *CRC* (4 байта). Как было показано в подразделе «Циклический избыточный код» раздела «Обнаружение и исправление ошибок», назначение поля CRC заключается в том, чтобы получающий адаптер (адаптер В) мог определить, не исказился ли кадр при передаче, то есть обнаружить ошибки в кадре. Искажение битов в кадре может быть вызвано ослаблением сигнала в канале, скачками напряжения, наводками в кабелях и интерфейсных платах. Обнаружение ошибок осуществляется следующим образом. Формируя Ethernet-кадр, хост А вычисляет значение поля CRC, получая его из значений остальных разрядов кадра (кроме преамбулы). Получив кадр, хост В вычисляет CRC по тому же методу и сравнивает, совпадает ли полученный результат с содержимым поля CRC в кадре. Эта операция, выполняемая принимающим хостом, называется контролем с помощью циклического избыточного кода, или просто проверкой контрольной суммы. Если контрольная сумма не совпадает со значением поля CRC кадра, тогда хост В понимает, что кадр поврежден.
- *Преамбула* (8 байт). Ethernet-кадр начинается с 8-байтового поля преамбулы. В каждый из первых 7 байт преамбулы записывается значение 10101010, а в последний байт — значение 10101011. Первые 7 байт должны «разбудить» принимающие адаптеры и помочь им синхронизировать свои таймеры с часами отправителя. Как уже отмечалось, адаптер А должен передать кадр со скоростью 10 Мбит/с, 100 Мбит/с или 1 Гбит/с в зависимости от типа локальной Ethernet-сети. Однако поскольку ничего не бывает абсолютно точным в реальном мире, скорость передачи всегда будет несколько отличаться от номинала. Величина этого отклонения скорости другим адаптерам локальной сети заранее не известна. Таким образом, первые 62 бита преамбулы, представляющие собой чередующиеся нули и единицы, позволяют приемнику с достаточной точностью настроиться на скорость передатчика, а последние два разряда (две единицы подряд) сообщают адаптеру В, что преамбула закончилась и следом идет уже первый информационный байт поля кадра. Адаптер В понимает, что следующие 6 байт содержат адрес получателя. Конец кадра адаптер может распознать просто по отсутствию сигнала в линии.

ИСТОРИЧЕСКАЯ СПРАВКА

Готовясь к получению степени доктора философии в Гарварде в начале 70-х годов, Боб Меткалф работал над проектом ARPAnet в Массачусетсом технологическом институте (Massachusetts Institute of Technologies, MIT). Во время обучения он познакомился с работами Абрамсона над сетью ALOHA и протоколами произвольного доступа. Получив степень незадолго до начала работы в исследовательском центре Xerox PARC корпорации Xerox в Пало-Альто, он посетил Абрамсона и его коллег в Гавайском университете, где в течение трех месяцев изучал сеть ALOHAnet. В центре Xerox PARC Меткалф занялся компьютерами Alto, во многом послужившими прототипами первых персональных компьютеров 80-х годов. Меткалф осознал необходимость разработки недорогого способа соединения подобных компьютеров в сеть. Таким образом, вооружившись знаниями о сетях ARPAnet и ALOHAnet, а также о протоколах произвольного доступа, Меткалф вместе со своим коллегой Дэвидом Боггсом спроектировал Ethernet-сеть.

Оригинальная Ethernet-сеть, разработанная Меткалфом и Боггсом, работала на скорости 2,94 Мбит/с и соединяла до 256 хостов на расстоянии до одной мили. Меткалфу и Боггсу удалось уговорить большую часть исследователей Xerox PARC принять участие в тестировании этой сети. Затем Меткалф сумел объединить корпорации Xerox, Digital и Intel в целях разработки стандарта Ethernet со скоростью 10 Мбит/с, ратифицированного институтом IEEE. Однако компания Xerox не проявила большого интереса к коммерциализации технологии Ethernet, поэтому в 1979 году Меткалф основал собственную компанию, 3Com, которая занялась разработкой и коммерциализацией сетевых технологий, включая технологию Ethernet. В частности, в начале 80-х годов компания 3Com подготовила и выпустила на рынок Ethernet-карты для очень популярных персональных компьютеров IBM PC. В 1990 году Меткалф покинул компанию 3Com. В это время в компании работали 2000 сотрудников, а доходы составляли 400 миллионов долларов. На начало января 2002 года количество сотрудников, работавших в компании 3Com, превышало 8000.

Ненадежная служба без установления соединения

Все технологии Ethernet предоставляют сетевому уровню *службу, не требующую установки соединения*. Таким образом, когда адаптер A хочет послать дейтаграмму адаптеру B, он помещает ее в Ethernet-кадр и передает этот кадр в локальную сеть, даже не обменявшись «рукопожатием» с адаптером B. Подобная не требующая соединения служба на канальном уровне (уровень 2) аналогична дейтаграммной службе протокола IP (уровень 3) и службе протокола UDP (уровень 4).

Даже если передаваемому кадру удается избежать коллизий, полученный кадр может содержать ошибки в отдельных битах, вызванные шумом в канале связи. Все разновидности технологий Ethernet предоставляют сетевому уровню *ненадежную службу*. В частности, когда адаптер B получает кадр от адаптера A, он выполняет CRC-контроль кадра, но не передает в ответ подтверждения, что кадр успешно прошел проверку (также не передается и отрицательная квитанция, если контрольная сумма не совпала со значением поля CRC). То есть у адаптера A нет ни малейшего представления о том, прошел переданный им кадр CRC-контроль или нет. Когда кадр не проходит такой проверки, принимающий адаптер просто отбрасывает кадр. Благодаря отсутствию на канальном уровне службы надежной доставки технология Ethernet остается простой и дешевой. Но это также означает, что в потоке дейтаграмм, переданных сетевому уровню, могут быть пропуски.

Если в потоке данных возникают пропуски, потому что протокол Ethernet теряет кадры, видит ли эти пропуски приложение на хосте B? Как отмечалось в главе 3, это полностью зависит от используемого приложением протокола (UDP или TCP). Если это протокол UDP, тогда приложению самому придется заниматься проблемой потерянных кадров. Если же это протокол TCP, то проблемой пропущенных кадров будет заниматься протокол TCP, с помощью механизма подтверждений и интервалов ожидания заставляя хост A передавать пропущенные данные повторно. Обратите внимание, когда протокол TCP передает данные повторно, эти данные снова продвигаются тем же Ethernet-адаптером. Так что в этом смысле Ethernet может повторять передачу данных. Однако следует помнить, что Ethernet-сеть не догадывается о том, что передает данные повторно, наивно полагая, что получает новенькую дейтаграмму с новыми данными.

Немодулированная передача и Манчестерское кодирование

В Ethernet-сети применяется немодулированная передача, то есть адаптер посылает цифровой сигнал прямо в широковещательный канал. Интерфейсная карта не сдвигает сигнал в другой частотный диапазон, как это делается, например, в ADSL-модемах и кабельных модемных системах. Кроме того, в Ethernet используется Манчестерское кодирование (рис. 5.23), когда каждый бит кодируется изменением уровня сигнала: единичный бит — спадом, нулевой, наоборот, — скачком. Такое кодирование позволяет приемнику синхронизироваться с передатчиком на каждом бите. Манчестерское кодирование скорее относится к физическому уровню, чем к канальному, но мы решили кратко упомянуть о нем в данной главе, так как оно широко применяется в сетях Ethernet.

Рис. 5.23. Манчестерское кодирование

Протокол CSMA/CD

Узлы в локальной Ethernet-сети соединены широковещательным каналом, так что кадр, переданный одним адаптером, принимается всеми адаптерами локальной сети. В разделе «Протоколы коллективного доступа» упоминался применяемый в Ether-

net алгоритм коллективного доступа CSMA/CD (Carrier Sense Multiple Access with Collision Detection — множественный доступ с контролем несущей и обнаружением коллизий). Вспомним некоторые его свойства.

- Адаптер может начинать передачу в любое время, то есть время не делится на слоты.
- Адаптер никогда не начинает передачу, если он слышит передачу другого адаптера, то есть выполняется контроль несущей.
- Адаптер прерывает передачу, как только он обнаруживает, что другой адаптер также выполняет передачу, то есть имеет место обнаружение коллизий.
- Прежде чем пытаться повторить передачу после обнаружения коллизии, адаптер выжидает в течение интервала времени случайной длительности. Как правило, по сравнению со временем, необходимым для передачи кадра, этот интервал небольшой.

Благодаря этим свойствам производительность протокола CSMA/CD значительно превышает производительность дискретного протокола ALOHA. В самом деле, если максимальная задержка распространения сигнала между станциями невелика, эффективность протокола CSMA/CD может достигать 100 %. Однако обратите внимание, что адаптеру приходится решать две важные задачи: прослушивать передачу другого адаптера (контролировать несущую) и обнаруживать коллизии. Эти две задачи адаптеры выполняют, измеряя уровни напряжения на линии перед началом и во время передачи.

Протокол CSMA/CD работает на каждом адаптере независимо от других адаптеров локальной Ethernet-сети.

1. Адаптер принимает от своего родительского узла единицу обмена (PDU) сетевого уровня, формирует Ethernet-кадр и помещает его в буфер адаптера.
2. Если адаптер определяет, что канал свободен (то есть в нем не наблюдается сигнала), он начинает передавать кадр. Если же канал занят, адаптер ждет, пока сигнал в линии не прекратится (плюс еще время, необходимое для передачи 96 бит), после чего начинает передачу кадра.
3. Передавая кадр, адаптер отслеживает наличие напряжения от сигнала других адаптеров. Если адаптер успевает передать весь кадр, не обнаружив сигнала других адаптеров, передача кадра считается успешной.
4. Если адаптер обнаруживает сигнал других адаптеров во время собственной передачи, он прекращает передачу кадра и передает 48-разрядный сигнал коллизии (jam signal).
5. После этого адаптер входит в фазу *экспоненциального отката*. В частности, после *n* неудачных попыток повторить передачу одного и того же кадра адаптер выбирает значение *K* псевдослучайным образом из множества {0, 1, 2,..., 2^m - 1}, где *m* := т т (я , 10). Затем адаптер выжидает в течение интервала времени, длительность которого равна *Kx* 512 длительностей передачи одного бита, после чего возвращается к шагу 2.

Следует сделать несколько замечаний. Назначение сигнала коллизии — информировать все передающие адаптеры о том, что имела место коллизия. Рассмотрим

следующий пример. Предположим, адаптер A начинает передавать кадр, и перед тем, как сигнал адаптера A достигнет адаптера B, адаптер B также начинает передачу. Поэтому адаптер B успевает передать всего несколько битов, после чего прерывает передачу. Разумеется, эти несколько битов достигают адаптера A, но скачка напряжения от них может оказаться недостаточно для того, чтобы адаптер A обнаружил коллизию. Чтобы адаптер A обнаружил коллизию наверняка (и также мог прервать свою передачу), адаптер B и передает 48-разрядный сигнал коллизии.

Теперь рассмотрим алгоритм экспоненциального отката. Прежде всего здесь следует обратить внимание на то, что время передачи одного бита очень мало; в Ethernet-сети со скоростью передачи данных 10 Мбит/с оно составляет всего лишь 0,1 мкс. Предположим, что адаптер пытается передать кадр в первый раз и обнаруживает коллизию. В этом случае с равной вероятностью 0,5 адаптер выбирает число $K = 0$ или $K = 1$. Если адаптер выбирает $K = 0$, тогда он, передав сигнал коллизии, немедленно переходит к шагу 2 алгоритма. Если же адаптер выбирает $K = 1$, тогда он, прежде чем перейти к шагу 2, ждет в течение 51,2 мкс. После второй коллизии подряд адаптер с равной вероятностью выбирает число K из набора {0, 1, 2, 3}. Испытав коллизию три раза подряд, адаптер выбирает число K из набора {0,1,2,3, 4, 5, 6, 7}. После десяти и более коллизий подряд число K выбирается из набора {0, 1, 2,..., 1023}. Таким образом, ряд чисел, из которого выбирается число K, растет экспоненциально с каждой следующей коллизией, пока количество попыток не достигнет 10. Поэтому алгоритм отката в протоколе Ethernet называется «экспоненциальным».

Стандарт Ethernet накладывает ограничения на расстояние между любой парой узлов. Эти ограничения гарантируют, что если адаптер A выберет меньшее значение K, чем остальные адаптеры, то он успеет передать кадр без коллизии. Более подробно мы рассмотрим это свойство в упражнениях, приведенных в конце главы.

Почему используется экспоненциальный откат? Почему, например, не выбирать число K из набора {0, 1, 2, 3, 4, 5, 6, 7}? Причина в том, что после первой коллизии адаптер не знает, сколько адаптеров вовлечено в коллизию. Если число таких адаптеров невелико, имеет смысл выбирать число K из небольшого набора чисел. С другой стороны, если в коллизию вовлечено много адаптеров, лучше выбирать число K из большего набора (почему?). Увеличивая набор чисел после каждой коллизии, адаптер постепенно адаптируется к ситуации.

Также обратите внимание, что адаптер инициирует запуск алгоритма CSMA/CD при передаче каждого нового кадра. При этом, в частности, не учитывается, были или нет перед этим коллизии. Таким образом, возможно, что пока несколько адаптеров выжидают в состоянии экспоненциального отката, адаптер сумеет «протолкнуть» свой новый кадр.

Когда кадр для передачи есть только у одного узла, он может передавать данные с максимальной скоростью, предоставляемой технологией Ethernet (например, 10 Мбит/с, 100 Мбит/с или 1 Гбит/с). Однако если кадры одновременно хотят передавать несколько узлов, эффективная скорость передачи данных по каналу может быть значительно ниже. Определим *эффективность Ethernet* как оцененную

за длительный период долю времени, в течение которого кадры передаются по каналу без коллизий при большом количестве активных узлов, у каждого из которых есть много кадров для передачи. Чтобы найти аналитическое приближение эффективности Ethernet-сети, обозначим через $t_{распр}$ максимальное время, необходимое для того, чтобы сигнал распространился между двумя самыми удаленными друг от друга адаптерами. Пусть $t_{Кадр}$ обозначает время передачи Ethernet-кадра максимального размера (в локальной Ethernet-сети со скоростью передачи 10 Мбит/с это составляет приблизительно 1,2 мс). Вывод формулы эффективности локальной Ethernet-сети выходит за рамки темы этой книги [27, 289], а здесь мы просто ее приведем:

$$\text{Эффективность} = \frac{1}{1 + 5t_{распр} / t_{Кадр}}.$$

Из этой формулы видно, что при приближении $t_{распр}$ к нулю эффективность Ethernet стремится к 1. Это показывает, что при нулевой задержке распространения вовлеченные в коллизию узлы будут немедленно прекращать передачу. Аналогично, если время передачи Ethernet-кадра максимального размера ($t_{Кадр}$) станет очень большим, эффективность приближается к 1. Это интуитивно понятно, так как, захватив канал, адаптер, передающий кадр, будет удерживать его в течение долгого времени, и таким образом большую часть времени канал будет занят полезным делом.

Технологии Ethernet

На сегодняшний день наиболее распространенными технологиями Ethernet, являются технологии 10Base2, lOBaseT, 100BaseT и Gigabit Ethernet. Стандарт 10Base2 предусматривает использование коаксиального кабеля в сети с топологией общей шины и скорость передачи данных 10 Мбит/с. Стандарт lOBaseT предусматривает использование медной витой пары в сети с топологией звезды и скорость передачи данных 10 Мбит/с. Стандарт 100BaseT предусматривает использование, как правило, медной витой пары в сети с топологией звезды и скорость передачи данных 100 Мбит/с. Стандарт Gigabit Ethernet (гигабитная Ethernet-сеть) предусматривает использование оптоволоконного кабеля или медной витой пары и скорость передачи данных 1 Гбит/с. Эти технологии Ethernet стандартизированы рабочей группой IEEE 802.3, поэтому локальную Ethernet-сеть часто называют локальной сетью 802.3.

Прежде чем перейти к обсуждению конкретных технологий Ethernet, мы должны поговорить о *повторителях,* широко применяемых в локальных сетях, а также в дальних линиях связи глобальных сетей. Повторитель представляет собой устройство физического уровня, работающее не с кадрами, а с отдельными битами. У повторителя два или более интерфейсов. Когда представляющий ноль или единицу бит прибывает на один из интерфейсов, повторитель просто воссоздает этот бит (восстанавливая форму и уровень сигнала) и передает его на все остальные свои интерфейсы. Повторители широко применяются в локальных сетях, позволяя расширить их географические размеры. Важно помнить, что в сетях Ethernet

повторители не контролируют несущую и не выполняют других функций протокола CSMA/CD. Повторитель просто воспроизводит принятый бит на всех своих выходных интерфейсах, даже если на каких-то из них уже есть сигнальное напряжение.

10Base2 Ethernet

В 90-х годах стандарт 10Base2 представлял собой популярную технологию Ethernet. Сегодня эта технология уже считается устаревшей, хотя все еще используется. Если вы взглянете, как соединены в сеть ваши компьютеры (на работе или в школе), возможно, вы обнаружите соединения стандарта 10Base2. Число 10 в слове «10Base2» означает 10 Мбит/с, а цифра «2» — 200 м, то есть максимальное расстояние между двумя узлами без повторителя между ними. (В действительности это расстояние составляет 185 м.) Схематично Ethernet-сеть стандарта 10Base2 изображена на рис. 5.24.

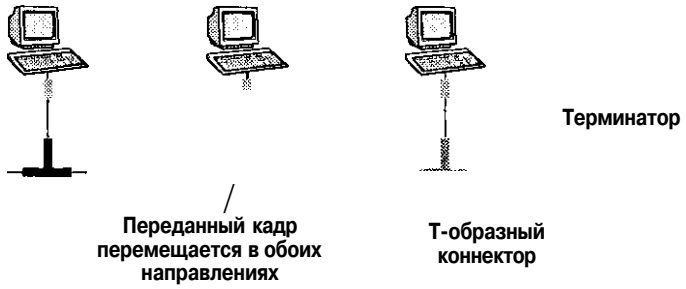


Рис. 5.24. 10Base2 Ethernet

Как видно из рисунка, в стандарте 10Base2 используется топология общей шины, то есть узлы соединены (через адаптеры) линейным образом. В качестве физического носителя для соединения узлов используется *тонкий коаксиальный кабель,* напоминающий кабель для кабельного телевидения, но несколько тоньше и легче. Когда адаптер передает кадр, этот кадр проходит через Т-образный коннектор, который покидают уже две копии кадра, при этом каждая копия направляется в своем направлении. Двигаясь по кабелю вплоть до терминатора, кадры оставляют копии на каждом узле, мимо которого они проходят. (Точнее, часть энергии бита, проходящего мимо узла, остается в адаптере.) Наконец, кадр достигает терминатора, который полностью поглощает кадр. Обратите внимание, когда один адаптер передает кадр, этот кадр принимается всеми остальными адаптерами локальной сети. Таким образом, 10Base2 действительно представляет собой широковещательную технологию.

Предположим, что вам нужно соединить дюжину персональных компьютеров в вашем офисе при помощи сети Ethernet стандарта 10Base2. Для этого вам нужно купить 12 Ethernet-карт с тонкими Ethernet-портами; 12 Т-образных коннекторов, представляющих собой маленькие металлические устройства, присоединяемые к адаптерам (меньше доллара за штуку); около дюжины кусков тонкого коаксиального кабеля, от 5 до 20 м каждый; и пару терминаторов, помещаемых на двух

Ethernet со скоростями передачи 1 и 10 Гбит/с

Локальная Ethernet-сеть со скоростью передачи данных 1 Гбит/с (Gigabit Ethernet) представляет собой расширение популярных стандартов lOBaseT и 100BaseT. Обеспечивая такую высокую скорость передачи необработанных данных, стандарт Gigabit Ethernet полностью совместим со всем оборудованием Ethernet, установленным за последние годы. Ниже перечислены некоторые характеристики стандарта Gigabit Ethernet (он имеет маркировку IEEE 802.3z).

- Используется стандартный для Ethernet формат кадра (см. рис. 5.22) и обеспечивается обратная совместимость с технологиями lOBaseT и 100BaseT. Это позволяет легко интегрировать сеть Gigabit Ethernet в существующую сеть с уже установленным оборудованием Ethernet.
- Допускается использовать как двухточечные линии, так и разделяемые широковещательные каналы. В двухточечных линиях применяются коммутаторы (см. раздел «Хабы, мосты, коммутаторы»), в широковещательных каналах — упоминавшиеся ранее хабы. В стандарте Gigabit Ethernet хабы называют «буферизированными распределителями».
- В разделяемых широковещательных каналах используется протокол CSMA/CD. Чтобы эффективность была приемлемой, максимальное расстояние между узлами должно быть строго ограничено.
- В двухточечных каналах допускаются дуплексные операции на скорости 1000 Мбит/с в обоих направлениях.

Как и в стандартах lOBaseT и 100BaseT, в стандарте Gigabit Ethernet применяется топология звезды с хабом или коммутатором в центре (коммутаторы Ethernet будут обсуждаться в разделе «Хабы, мосты, коммутаторы»). Сеть Gigabit Ethernet часто используется в качестве магистрали для соединения нескольких локальных сетей Ethernet стандартов lOBaseT и 100BaseT. Изначально для стандарта Gigabit Ethernet в качестве носителя предполагался оптоволоконный кабель, но теперь допускается и неэкранированная витая пара категории 5.

Десятигигабитная разновидность стандарта Ethernet явилась следующим шагом в развитии технологии Ethernet. Кроме того, 10-гигабитная Ethernet-сеть стандарта 802.3ae расширяет технологию двухточечных соединений для глобальных сетей. Много полезной информации о сетях Ethernet, работающих на скоростях 1 Гбит/с и 10 Гбит/с, содержится на web-сайте http://www.ethermanage.com/ethernet/ethernet.htmL

Хабы, мосты, коммутаторы

Многие организации, включая корпорации, университеты, школы, состоят из различных подразделений, у каждого из которых, как правило, есть собственная локальная Ethernet-сеть. Разумеется, этим организациям необходимо иметь возможность объединять свои отдельные локальные сети в единую сеть. В данном разделе мы рассмотрим три разных устройства, предназначенные для соединения локальных сетей, — это хабы, мосты и коммутаторы. Сегодня эти три устройства получили широкое распространение.

Хабы

Простейший способ объединения локальных сетей заключается в использовании *хаба.* Получив бит по одному интерфейсу, хаб просто передает этот бит по всем остальным интерфейсам. Поскольку хабы оперируют не кадрами, а битами, они представляют собой устройства физического уровня. Хабы являются ничем иным, как повторителями, которым добавлены некоторые функции по управлению сетью (см. главу 8).

На рис. 5.26 показан пример объединения в единую сеть трех локальных сетей, относящихся к трем факультетам университета. У каждого из трех факультетов есть локальная Ethernet-сеть стандарта lOBaseT, предоставляющая сетевой доступ преподавателям, сотрудникам и студентам факультета. Каждый хост факультета соединен с хабом факультета двухточечной линией. Четвертый хаб, называемый магистральным, обеспечивает объединение локальных сетей трех факультетов. Такая схема называется *схемой многозвенных хабов,* так как хабы организованы в иерархическую структуру. Можно создать подобную иерархическую структуру с количеством звеньев более двух, например, первое звено — для кафедр, второе — для факультетов и, наконец, третье — для университета в целом. Многозвенную структуру также можно создать из локальных сетей Ethernet стандарта lOBaseT (с топологией общей шины) с повторителями.

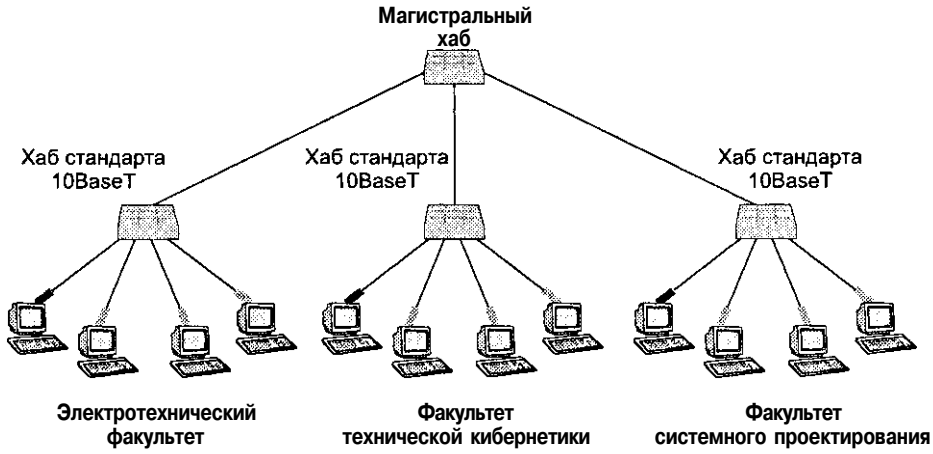


Рис. 5.26. Три локальные сети, соединенные хабом

В многозвенной схеме мы будем называть всю объединенную сеть локальной седъю, а каждую отдельную локальную сеть факультета — *сегментом локальной сети.* Важно отметить, что все сегменты локальной сети, представленные на рисунке, принадлежат одному и тому же *домену коллизий.* Это означает, что каждый раз, когда два или более узлов локальной сети одновременно начнут передачу, эта передача приведет к коллизии, и все передающие узлы войдут в фазу экспоненциального отката.

Объединение локальных сетей факультетов при помощи магистрального хаба имеет много преимуществ. Во-первых, такая схема позволяет соединить хосты различ-

ных факультетов. Во-вторых, она позволяет увеличить максимальное расстояние между отдельными узлами сети. Например, в стандарте lOBaseT максимальное расстояние между узлом и его хабом составляет 100 м. Таким образом, при наличии одного хаба максимальное расстояние между двумя узлами равно 200 м. Если же соединить хабы через хаб более высокого звена, это расстояние может быть увеличено, так как между соединенными напрямую хабами при использовании витой пары также может быть до 100 метров (и более в случае оптоволоконного кабеля). Третье преимущество заключается в том, что многозвенная структура обеспечивает определенную степень «непотопляемости»: если один из хабов факультетского звена выйдет из строя, магистральный хаб может обнаружить неисправность и отключить неисправный хаб от локальной сети. Таким образом, сети остальных факультетов смогут продолжить работу даже во время ремонта поврежденного хаба.

Хотя магистральный хаб является полезным соединительным устройством, у него есть три серьезных недостатка. Первый и, возможно, наиболее существенный недостаток состоит в том, что при объединении локальных сетей факультетов через хаб (или повторитель) независимые домены коллизий факультетов также объединяются в один большой общий домен. Рассмотрим этот вопрос на примере рис. **5**.26. До объединения локальных сетей трех факультетов максимальная пропускная способность каждой сети составляла 10 Мбит/с, таким образом, максимальная суммарная пропускная способность трех сетей была равна 30 Мбит/с. Однако как только эти три локальные сети были объединены при помощи хаба, все хосты трех факультетов оказались в одном общем домене коллизий, и максимальная суммарная пропускная способность уменьшилась до 10 Мбит/с.

Второй недостаток заключается в том, что если разные факультеты используют различные технологии Ethernet, то подобное объединение локальных сетей через хаб может оказаться невозможным. Например, если на одном факультете установлена сеть lOBaseT, а на остальных факультетах — сети стандарта 100BaseT, тогда объединить сети всех факультетов без буферизации кадров в точке соединения невозможно. Поскольку хабы представляют собой повторители и не буферизируют кадры, они не могут соединять сегменты локальных сетей, работающих на разных скоростях.

Третий недостаток состоит в том, что в каждой технологии Ethernet (10Base2, lOBaseT, 100BaseT и т. д.) имеются свои ограничения на максимальное допустимое количество узлов в домене коллизий, максимальное расстояние между двумя хостами домена коллизий и на максимальное допустимое число звеньев в многозвенной схеме. В результате эти ограничения накладываются и на суммарное количество хостов в многозвенной локальной сети, и на ее географические размеры.

Мосты

В отличие от хабов, представляющих собой устройства физического уровня, мо^сты работают с Ethernet-кадрами и таким образом являются устройствами второго уровня. В самом деле, *мосты* (а также их ближайшие родственники, Ethernet-коммутаторы) представляют собой коммутаторы пакетов, переправляющие и

фильтрующие кадры при помощи LAN-адресов получателей. Когда кадр попадает в интерфейс моста, мост не копирует кадр на все остальные интерфейсы, как хаб, а проверяет адрес получателя второго уровня, содержащийся в кадре, и пытается переправить кадр по интерфейсу, ведущему к получателю.

Вариант соединения локальных сетей трех факультетов с помощью моста показан на рис. 5.27. Три числа рядом с мостом обозначают номера интерфейсов моста. Мы снова будем называть всю объединенную сеть локальной сетью, а локальную сеть каждого факультета — сегментом локальной сети. Однако в отличие от многозвенной схемы, показанной на рис. 5.26, в данном случае каждый сегмент локальной сети представляет собой изолированный домен коллизий.

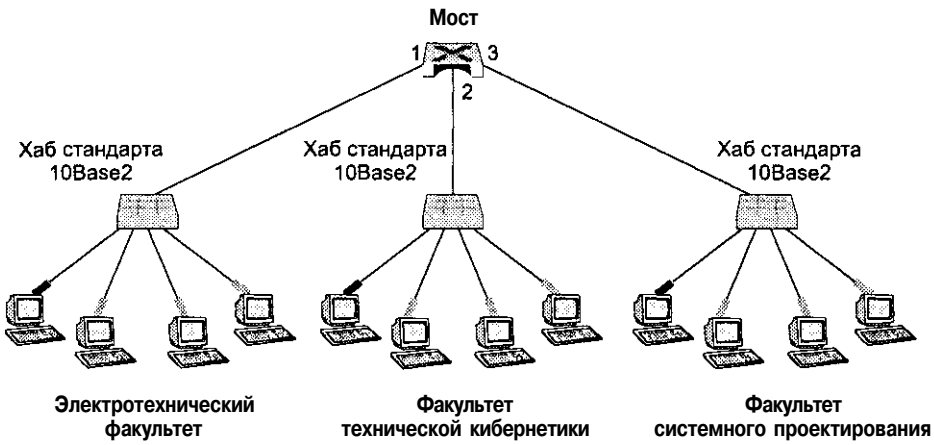


Рис. 5.27. Три локальные сети, соединенные мостом

С помощью мостов можно решить множество проблем, возникающих при использовании хабов. Во-первых, мосты обеспечивают связь между разными факультетами, сохраняя при этом домены коллизий каждого факультета изолированными. Во-вторых, при помощи мостов можно объединять технологически разнотипные локальные сети, например Ethernet-сети со скоростями 10 Мбит/с и 100 Мбит/с. В-третьих, в случае мостов исчезает ограничение на размеры объединенной сети. Теоретически мосты позволяют создать локальную сеть, покрывающую всю планету.

Продвижение данных и фильтрация

Фильтрацией называют способность мостов определять, следует ли передать кадр определенному интерфейсу или его можно просто отбросить. *Продвижением данных* называют способность мостов определять, какому из интерфейсов следует направить кадр. Фильтрация и продвижение кадров осуществляются при помощи *таблицы моста*. В таблице моста содержатся записи для некоторых (не обязательно всех) узлов локальной сети. Каждая запись таблицы моста содержит, во-первых, LAN-адрес узла, во-вторых, номер интерфейса моста, ведущего к этому узлу, и, в-третьих, время включения этой информации в таблицу. Пример таблицы моста для локальной сети, представленной на рис. 5.27, дан в табл. 5.2.

Таблица 5.2. Фрагмент таблицы моста для локальной сети

LAN-адрес узла	Номер интерфейса моста	Время
62-FE-F7-11-89-A3	1	9:22
7C-BA-B2-B4-91-10	3	9:36

Хотя представленное здесь описание механизмов продвижения кадра может напомнить описание продвижения дейтаграмм, данное в главе 4, мы вскоре обнаружим существенные различия. Следует отметить, что используемые мостами адреса являются локальными адресами (LAN-адресами), а не адресами сетевого уровня (например, IP-адресами). Мы также вскоре увидим, что процесс формирования таблицы моста значительно отличается от построения таблицы продвижения данных.

Чтобы понять механизмы фильтрации и продвижения, предположим, что на мост поступает кадр с адресом получателя DD-DD-DD-DD-DD-DD по интерфейсу *x*. Мост использует LAN-адрес DD-DD-DD-DD-DD-DD в качестве индекса и находит соответствующий ему интерфейс *y*. (Что произойдет, если адреса DD-DD-DD-DD-DD-DD не окажется в таблице, мы рассмотрим чуть позже.)

- Если номер *x* равен номеру г/, тогда кадр поступил из сегмента локальной сети, содержащей адаптер DD-DD-DD-DD-DD-DD. В этом случае нет необходимости переправлять кадр какому-либо другому интерфейсу, и мост просто отбрасывает (фильтрует) кадр.
- Если номер *x* не равен номеру г/, тогда кадр должен быть переправлен в сегмент локальной сети, присоединенный к интерфейсу *y*. В этом случае мост выполняет функцию продвижения данных, помещая кадр в выходной буфер, соответствующий интерфейсу *y*.

Эти простые правила позволяют мосту сохранять раздельные домены коллизий для каждого из сегментов локальной сети, присоединенных к его интерфейсам. Таким образом, два множества узлов могут одновременно обмениваться информацией, не мешая друг другу.

Рассмотрим работу этих правил на примере сети, показанной на рис. 5.27, и ее таблицы моста (см. табл. 5.2). Предположим, что кадр с адресом получателя 62-FE-F7-11-89-A3 прибывает на мост через интерфейс 1. Мост сверяется с таблицей и определяет, что получатель находится в сегменте локальной сети, соединенном с интерфейсом 1 (то есть в локальной сети электротехнического факультета). Это означает, что данный кадр уже получили все хосты этой локальной сети, поэтому мост отфильтровывает (то есть отбрасывает) этот кадр. Теперь предположим, что кадр с тем же адресом получателя прибывает на мост по интерфейсу 2. Мост снова заглядывает в таблицу и видит, что адресат доступен через интерфейс 1. Поэтому мост переправляет кадр в выходной буфер интерфейса 1. Из этого примера должно быть ясно, что пока таблица моста содержит полную и точную информацию, мост обеспечивает хостам различных факультетов возможность обмениваться данными, в то же время сохраняя для каждого изолированные домены коллизий.

Вспомним, что, продвигая кадр в линию, хаб (повторитель) просто пересылает в линию биты, не прослушивая линию, чтобы проверить, свободна она или занята. В отличие от хаба мост, переправляющий кадр в линию, работает по алгоритму CSMA/CD (см. раздел «Протоколы коллективного доступа»). В частности, мост воздерживается от передачи, если обнаруживает, что ведет передачу какой-либо другой узел в данном сегменте локальной сети. Так же как и любой другой узел, в случае коллизии мост выполняет процедуру экспоненциального отката. Таким образом, поведение интерфейсов моста во многом подобно поведению адаптеров узлов. Но, строго говоря, интерфейсы моста не являются адаптерами, так как ни они, ни сам мост не имеют LAN-адресов. Вспомним, что адаптер узла всегда помещает в поле адреса отправителя каждого отправляемого им кадра свой LAN-адрес. Это справедливо как для адаптеров хостов, так и для адаптеров маршрутизаторов. Однако мост не меняет адреса отправителя кадра.

Важным свойством мостов является то, что они могут объединять сегменты локальных сетей Ethernet, использующие разные технологии. Например, если бы на рис. 5.27 электротехнический факультет был оснащен локальной сетью Ethernet стандарта 10Base2, на факультете технической кибернетики была установлена сеть 100BaseT, а на факультете системного проектирования — сеть lOBaseT, тогда для объединения этих трех локальных сетей тоже можно было бы использовать мост. Мост также позволяет соединить старую и медленную локальную сеть с новой и быстрой линией связи. Как уже упоминалось, хабы не обладают подобной способностью объединять сети с различными скоростями передачи данных.

При использовании мостов в качестве соединительных устройств исчезает теоретическое ограничение на географические размеры локальной сети. Теоретически, соединяя хабы попарно мостами, можно построить локальную сеть линейной топологии, покрывающую всю планету. При таком строении сети у каждого хаба будет свой собственный домен коллизий, и не будет ограничения на протяженность локальной сети. Однако, как мы скоро увидим, создание очень больших сетей при помощи одних только мостов в качестве соединительных устройств нежелательно. Большие сети должны содержать также и маршрутизаторы.

Самообучение

Мост обладает замечательным свойством, состоящим в том, что его таблица формируется автоматически, динамически и автономно — без вмешательства сетевого администратора или протокола конфигурирования. Другими словами, мосты обладают способностью *самообучения,* которая реализуется следующим образом.

1. Изначально таблица моста пуста.
2. Когда по одному из интерфейсов прибывает кадр, а адреса получателя нет в таблице, мост посылает копии кадра по *всем* остальным интерфейсам. (На каждом из этих интерфейсов при передаче кадров в локальные сети используется протокол CSMA/CD.)
3. Для каждого полученного кадра мост сохраняет в своей таблице, во-первых, LAN-адрес, содержащийся в *поле адреса отправителя кадра,* во-вторых, номер интерфейса, по которому прибыл кадр, в-третьих, время получения кадра. Таким

образом, мост записывает в своей таблице сведения о том, в каком сегменте локальной сети располагается узел, отправивший кадр. Если каждый узел локальной сети передаст по кадру, тогда в таблице моста окажутся LAN-адреса всех узлов.

4. Когда по одному из интерфейсов прибывает кадр, адрес получателя которого есть в таблице, мост направляет кадр соответствующему интерфейсу.
5. Если за определенный период времени *{время старения)* мост не получает кадров с некоторым адресом, этот адрес удаляется из таблицы. Таким образом, если один персональный компьютер в сети заменяется другим (с другим адаптером), локальный адрес первого персонального компьютера, в конце концов, будет удален из таблицы моста.

Рассмотрим способность к самообучению на примере сети, показанной на рис. 5.27, и соответствующей ей табл. 5.2. Пусть в момент времени 9:39 кадр с адресом источника 01-12-23-34-45-56 прибывает по интерфейсу 2. Предположим, что этот адрес отсутствует в таблице. В этом случае мост добавляет к таблице новую запись (табл. 5.3).

Таблица 5.3. Фрагмент таблицы моста, когда мост узнает об адаптере 01-12-23-34-45-56

LAN-адрес узла	Номер интерфейса моста	Время
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36

Продолжая тот же пример, предположим, что время старения для данного моста равно 60 мин, и с 9:32 до 10:32 не приходит ни одного кадра с адресом источника 62-FE-F7-11-89-A3. В этом случае в 10:32 мост удаляет запись с этим адресом из таблицы.

Мосты представляют собой *самонастраивающиеся устройства,* так как не требуют вмешательства со стороны сетевого администратора или пользователя. Сетевой администратор, который хочет установить мост, должен всего лишь присоединить сегменты локальной сети к интерфейсам моста. Ему не нужно настраивать таблицы моста во время установки моста или после удаления хоста из одного из сегментов локальной сети. Поскольку мосты представляют собой самонастраивающиеся устройства, их также называют *прозрачными мостами.*

Связующее дерево

Один из недостатков чисто иерархической схемы соединенных сегментов локальной сети заключается в том, что при выходе из строя хаба или моста, близкого к вершине в иерархической структуре, большие фрагменты локальной сети оказываются отсоединенными. По этой причине желательно создавать сети, сегменты которых соединены несколькими маршрутами. Пример такой сети показан на рис. 5.28.

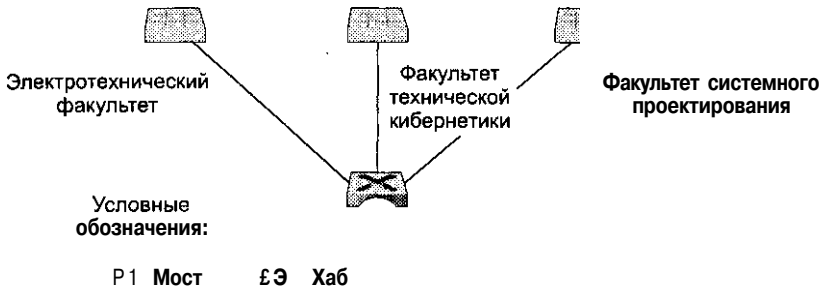


Рис. 5.28. Объединенные сегменты локальной сети с избыточными путями

Наличие большого количества избыточных путей между отдельными сегментами локальной сети (например, сетями факультетов) может значительно повысить устойчивость к неисправностям. Но, к сожалению, у сети с несколькими путями есть серьезный побочный эффект: если не предпринимать специальных мер, в объединенной сети кадры могут «зацикливаться» и «размножаться» [384]. Чтобы увидеть, как это происходит, предположим, что таблицы в мостах, показанных на рис. 5.28, пусты, а один из хостов электротехнического факультета посылает кадр хосту факультета технической кибернетики. Когда кадр прибывает на хаб электротехнического факультета, хаб создает две копии кадра, по одной для каждого моста, и посылает их мостам. Каждый мост, получив кадр, также создает две копии, одну из которых посылает хабу факультета кибернетики, другую — хабу факультета системного проектирования. Теперь в локальной сети оказываются уже четыре идентичных кадра. Это размножение кадров может продолжаться бесконечно, так как мосты не знают, где располагается хост-адресат. (Вспомним, что LAN-адрес хоста не появится в таблице, пока этот хост сам не пошлет кому-нибудь кадр.) В результате количество копий кадра растет в геометрической прогрессии, что приводит к выходу из строя всей сети.

Чтобы предотвратить зацикливание и размножение кадров, в мостах применяется *протокол связующего дерева* [383]. Мосты обмениваются друг с другом по локальной сети данными, чтобы найти связующее дерево, то есть подмножество оригинального графа сети, не имеющее замкнутых контуров (петель). Найдя связующее дерево, мосты виртуально отсоединяют (не используют) интерфейсы, не входящие в связующее дерево. Например, в сети на рис. 5.28 связующее дерево может быть создано, если верхний мост отсоединит свой интерфейс от хаба электротехнического факультета, а нижний — от хаба факультета системного проектирования. После этой операции кадры перестанут зацикливаться и размножаться. Если вдруг когда-либо один из каналов связующего дерева выйдет из строя, мосты могут автоматически восстановить коммутацию отключенных интерфейсов, снова запустить алгоритм связующего дерева и определить новый набор интерфейсов, которые следует виртуально отсоединить.

Мосты и маршрутизаторы

Как было показано в главе 4, маршрутизаторы представляют собой коммутаторы пакетов с промежуточным хранением, осуществляющие продвижение пакетов при помощи адресов сетевого уровня. Хотя мост также является коммутатором пакетов с промежуточным хранением, он принципиально отличается от маршрутизатора тем, что для продвижения пакетов он использует LAN-адреса. Таким образом, маршрутизатор представляет собой коммутатор пакетов третьего уровня, а мост — коммутатор пакетов второго уровня.

Несмотря на фундаментальное различие мостов и маршрутизаторов, сетевым администраторам часто приходится выбирать между ними при установке соединительного устройства. Например, в сети, представленной на рис. 5.27, сетевой администратор вполне мог бы вместо моста установить маршрутизатор. В самом деле, маршрутизатор также обеспечил бы независимость трех доменов коллизий, в то же время предоставляя возможность обмена данными между сетями факультетов. Итак, каковы достоинства и недостатки мостов ^маршрутизаторов?

Сначала рассмотрим достоинства и недостатки мостов. Как уже упоминалось, мосты представляют собой самонастраивающиеся устройства, что всегда высоко ценится «по уши» загруженными работой сетевыми администраторами. Кроме того, мосты могут поддерживать относительно высокие скорости обработки (фильтрации и продвижения) пакетов. Как видно из схемы, показанной на рис. 5.29, мосты должны обрабатывать пакеты только до уровня 2, тогда как маршрутизаторам приходится просматривать их до уровня 3. С другой стороны, протокол связующего дерева ограничивает эффективную топологию соединенной мостами сети только самим связующим деревом. Это означает, что все кадры должны передаваться только по связующему дереву, даже если существуют более короткие (но виртуально отключенные) маршруты от отправителя до получателя. Кроме того, применение алгоритма связующего дерева приводит к тому, что весь трафик концентрируется на линиях, входящих в связующее дерево, когда его можно было бы распределить между всеми линиями связи, имеющимися в сети. Более того, мосты не обеспечивают защиты от так называемого «широковещательного шторма» — ситуации, когда один из хостов «сходит с ума» и начинает передавать нескончаемый поток широковещательных Ethernet-кадров. В этом случае мосты будут продолжать рассылать по сети все эти кадры, что, в конце концов, приведет сеть в совершенно неработоспособное состояние.

Теперь рассмотрим достоинства и недостатки маршрутизаторов. Поскольку сетевое адресное пространство иерархическое (в отличие от плоского адресного пространства локальной сети), пакеты в нормальных условиях не зацикливаются, даже при наличии в сети избыточных путей. (Хотя при неверно сконфигурированных таблицах зацикливание возможно, для ограничения зацикливания, как было показано в главе 4, протокол IP использует специальное поле в заголовке дейтаграммы.) Таким образом, маршруты пакетов не ограничены связующим деревом, и пакеты могут передвигаться от отправителя к получателю по оптимальным маршрутам. Отсутствие подобного ограничения позволяет произвольно развивать топологию Интернета, например, связывая Европу и Северную Америку несколькими ак-

тивными линиями. Другая особенность маршрутизаторов заключается в том, что они, подобно брандмауэрам, предоставляют защиту от широковещательных штормов уровня 2. Однако у маршрутизаторов есть один очень существенный недостаток — они не являются самонастраивающимися устройствами; для работы им и соединенным с ними хостам необходимо знать IP-адреса. Кроме того, обработка каждого пакета на маршрутизаторе часто требует больше времени, чем на мосту, так как им приходится обрабатывать поля заголовков вплоть до 3-го уровня.

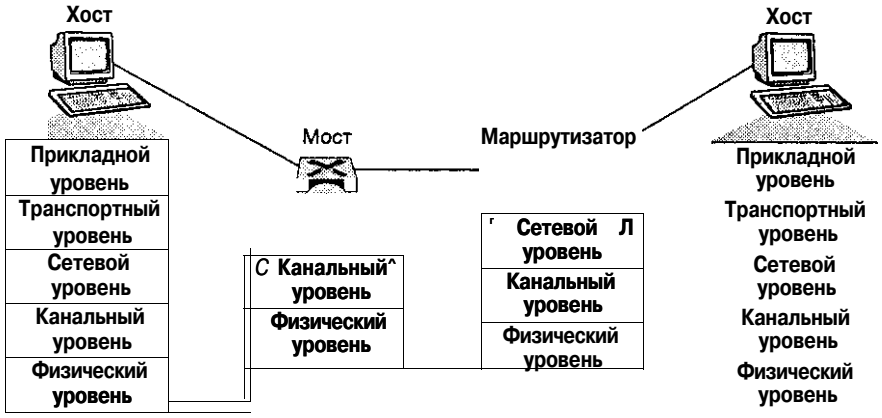


Рис. 5.29. Обработка пакетов мостами, маршрутизаторами и хостами

Итак, учитывая достоинства и недостатки мостов и маршрутизаторов, какие соединительные устройства следует использовать для объединения локальных сетей организации? Как правило, небольшие сети, включающие несколько сот хостов, состоят из небольшого количества сегментов. Для таких небольших сетей достаточно мостов, так как они локализуют трафик и увеличивают суммарную пропускную способность, не требуя настройки IP-адресов. Однако сети больших размеров, состоящие из тысяч хостов, как правило, помимо мостов должны содержать и маршрутизаторы. Маршрутизаторы обеспечивают более устойчивую изоляцию трафика, сдерживают широковещательные штормы, а также более эффективно используют маршруты между хостами в сети.

Соединение локальных сетей через магистраль

Рассмотрим еще раз проблему соединения локальных Ethernet-сетей трех факультетов с помощью мостов (см. рис. 5.27). Альтернативный подход показан на рис. 5.30. В данной схеме используются два моста, у каждого из которых по два интерфейса. Один мост соединяет электротехнический факультет с факультетом технической кибернетики, а второй — факультет технической кибернетики с факультетом системного проектирования. Хотя мосты с двумя интерфейсами очень популярны, что вызвано их низкой стоимостью и простотой, использование подобной схемы *не рекомендуется* по двум причинам. Во-первых, если хаб факультета технической кибернетики выйдет из строя, тогда хосты электротехнического факультета не смогут общаться с хостами факультета системного проектирования. Но что еще

отключаются. Например, на рис. 5.31 хост A может переслать файл хосту A', в то время как хост B посылает файл хосту B', а хост C посылает файл хосту C. Если у каждого хоста есть 10-мегабитная сетевая карта, тогда суммарная пропускная способность при такой одновременной передаче файлов равна 30 Мбит/с. Если же, например, у хостов A и A' 100-мегабитные адаптеры, а у остальных хостов 10-мегабитные, тогда суммарная пропускная способность при одновременной передаче файлов составит 120 Мбит/с.

На рис. 5.32 показан пример конфигурации сети для института, состоящего из нескольких факультетов, в которой используется комбинация из хабов, Ethernet-коммутаторов и маршрутизаторов. В данном примере у каждого факультета есть свой 10-мегабитный сегмент Ethernet со своим хабом. Поскольку хаб каждого факультета соединен с коммутатором, весь трафик внутри факультета не выходит за пределы данного сегмента сети (при условии, что таблицы продвижения данных в Ethernet-коммутаторе заполнены). Почтовый сервер и web-сервер соединены с коммутатором выделенными 100-мегабитными линиями. Наконец, маршрутизатор, соединяющий локальную сеть с Интернетом, также соединен с коммутатором выделенной 100-мегабитной линией. Обратите внимание, что у используемого в данной конфигурации коммутатора должно быть, по меньшей мере, три 10-мегабитных интерфейса и три 100-мегабитных интерфейса.

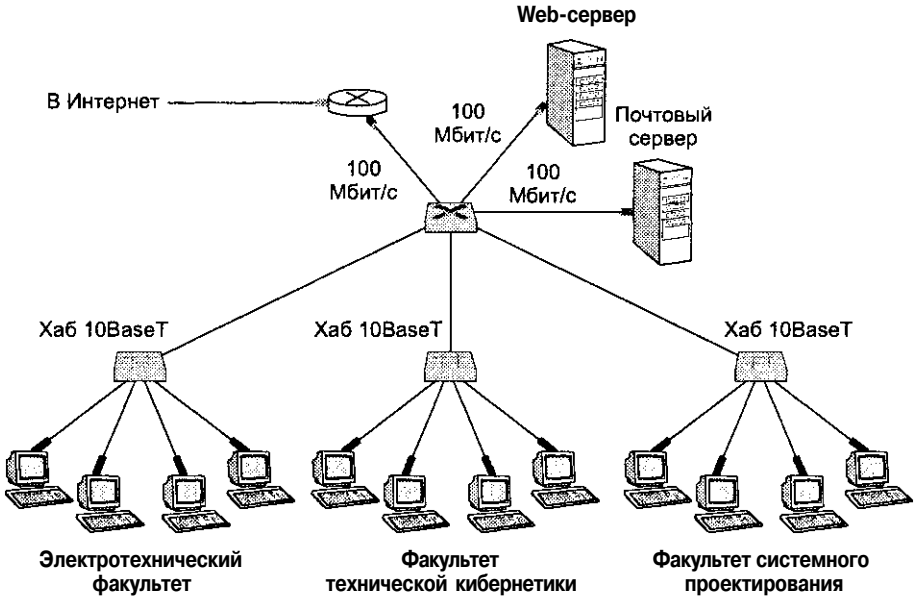


Рис. 5.32. Пример конфигурации сети

В подразделе «Мосты» данного раздела мы упомянули, что сетевой администратор часто сталкивается с необходимостью выбора между мостами и маршрутизаторами. С появлением коммутаторов задача сетевого администратора еще более усложнилась. Но, поскольку функционально коммутаторы близки к мостам, выбор между

коммутаторами и маршрутизаторами не сложнее выбора между мостами и маршрутизаторами. В частности, в сети, построенной исключительно на коммутаторах, широковещательный шторм охватывает сразу всю сеть. Именно по этой причине рекомендуется включать маршрутизаторы в корпоративные или университетские сети.

Помимо большого количества интерфейсов, средств поддержки разных типов физических носителей и разных скоростей передачи даннщх, а также массы соблазнительных функций управления сетью, производители Ethernet-коммутаторов часто любят говорить о том, что в их коммутаторах используется не обычная коммутация пакетов с промежуточным хранением, применяемая в маршрутизаторах и мостах, а *сквозная коммутация.* Отличие сквозной коммутации от коммутации с промежуточным хранением едва уловимо. Чтобы понять, в чем же разница, рассмотрим пакет, проходящий через коммутатор пакетов (то есть маршрутизатор, мост или Ethernet-коммутатор). Пакет поступает на коммутатор через входную линию и покидает его по выходной линии. В момент прибытия пакета в коммутаторе могут находиться другие пакеты, стоящие в очереди (в буфере) на выходной линии. Когда выходной буфер не пуст, нет абсолютно никакой разницы между коммутацией с промежуточным хранением и сквозной коммутацией. Эти два метода коммутации различаются только тогда, когда выходной буфер пуст.

Как было показано в главе 1, когда пакет проходит через коммутатор пакетов с промежуточным хранением, он сначала принимается целиком и помещается в буфер, и лишь после этого коммутатор начинает передачу по выходной линии. Однако в случае, когда выходной буфер пуст, это накопление данных в буфере всего лишь создает лишнюю задержку, вносящую свой вклад в суммарную сквозную задержку (см. раздел «Задержки и потери данных в сетях с коммутацией пакетов» в главе 1). Верхняя граница этой задержки равна L/R, где L представляет собой длину пакета, а R — скорость передачи данных по входной линии. Обратите внимание, что задержка промежуточного хранения появляется только в том случае, если выходной буфер коммутатора освобождается прежде, чем весь пакет прибывает на коммутатор.

При сквозной коммутации, если буфер освобождается прежде, чем прибудет весь пакет, коммутатор может начать передачу начала пакета, пока оставшаяся часть пакета продолжает прибывать. Разумеется, прежде чем начать передачу пакета по выходной линии на коммутатор, должна прибыть та часть пакета, в которой содержится адрес получателя (эта небольшая задержка неизбежна для всех типов коммутации, так как коммутатор должен выбрать соответствующую выходную линию). Таким образом, при сквозной коммутации коммутатор не должен ждать, пока прибудет весь пакет, а может начинать передачу, как только освобождается выходная линия. Если выходная линия представляет собой сеть коллективного доступа, совместно используемую несколькими хостами (например, выходная линия соединяется с хабом), то коммутатор должен также прослушивать выходную линию, дожидаясь, когда она освободится.

Чтобы яснее представить себе различие между сквозной коммутацией и.коммутацией с промежуточным хранением, вспомним аналогию с группой движущихся по шоссе машин, приводившуюся в разделе «Задержки и потери данных в сетях с коммутацией пакетов» главы 1. В данной аналогии рассматривается шоссе с установленными вдоль него пунктами сбора проездных пошлин, в каждом из которых нахо-

дится один сотрудник. По шоссе движется группа из десяти машин, причем скорость каждой одинакова и постоянна. Других автомашин на шоссе нет. Обслуживание каждой машины на каждом пункте сбора пошлин занимает одинаковое время, поэтому машины отъезжают от пункта сбора пошлин, разделенные равными интервалами. Как и раньше, мы можем представить, что группа машин — это пакет, каждая машина — это бит, а скорость обслуживания на пункте сбора пошлин соответствует скорости передачи данных в канале. Рассмотрим теперь, как идет обслуживание машин на пункте сбора пошлин. Если каждая машина обслуживается сразу по прибытии, то в этом случае мы имеем аналогию со сквозной коммутацией. Если же, напротив, каждая машина должна ждать, пока соберутся все остальные машины, тогда данный пункт сбора пошлин осуществляет «промежуточное хранение». Очевидно, во втором случае группа в каждом пункте будет задерживаться дольше, чем в первом.

Сквозные коммутаторы позволяют снизить сквозную задержку пакетов, но насколько? Как уже отмечалось, максимальная задержка промежуточного хранения равна L/R, где L представляет собой длину пакета, а R — скорость передачи данных по входной линии. Максимальная задержка приблизительно равна 1,2 мс для 10-мегабитной Ethernet-сети и 0,12 мс для локальной Ethernet-сети со скоростью передачи данных 100 Мбит/с (учитывая максимальный размер пакета). Таким образом, сквозной коммутатор снижает задержку всего лишь на интервал времени от 0,12 до 1,2 мс, причем только в том случае, когда выходная линия загружена незначительно. Насколько существенна эта задержка? Вероятно, для большинства приложений она не очень существенна, поэтому, прежде чем инвестировать средства в сквозную коммутацию, вам следует крепко подумать,* стоит ли ради этого продавать свой дом.

Итак, в этом разделе мы узнали, что для соединения сегментов локальной сети могут использоваться хабы, мосты, маршрутизаторы и коммутаторы. В табл. 5.4 приводятся характеристики каждого из этих соединительных устройств. Дополнительную информацию о различных технологиях объединения сетей можно найти на сайте компании Cisco (http://wwwxiscoxom/warp/public/cc/pd/si/index.shtml).

Таблица 5.4. Сравнительные характеристики популярных соединительных устройств

Характеристика	Хабы	Мосты	Маршрутизаторы	Ethernet-коммутаторы
Изоляция трафика	Нет	Да	Да	Да
Самонастройка	Да	Да	Нет	Да
Оптимальная маршрутизация	Нет	Нет	Да	Нет
Сквозная коммутация	Да	Нет	Нет	Да

Беспроводные каналы связи

Одним из наиболее перспективных направлений в мире компьютерных сетей являются беспроводные оконечные системы. К этим мобильным оконечным системам относятся переносные персональные компьютеры, объединяемые в беспроводные локальные сети, а также карманные компьютеры, предоставляющие соединение

с Интернетом по сотовой связи. Через несколько лет ожидается также появление других мобильных устройств, соединенных с Интернетом. Ряды этих устройств могут пополнить фото- и видеокамеры, автомобили, домашние животные, охранные системы, кухонные комбайны и бытовые приборы. Когда-нибудь мобильные устройства, соединенные с Интернетом, распространятся повсеместно, и их можно будет встретить везде: на стенах, в автомобилях, в спальнях, в наших карманах и даже на наших телах [222].

В этом разделе мы познакомимся с двумя наиболее важными стандартами беспроводных сетей. Сначала мы обсудим популярный стандарт беспроводных сетей IEEE 802.1 lb, а затем кратко рассмотрим новый стандарт Bluetooth, позволяющий обмениваться данными устройствам, находящимся в пределах прямой видимости.

Каждое устройство беспроводного сетевого доступа можно классифицировать по трем характеристикам: мощности, дальности и скорости передачи данных. Стандарт Bluetooth представляет собой низкоскоростную технологию «замены кабеля», использующую сигнал малой мощности, действующий на коротком расстоянии, тогда как стандарт 802.11 описывает технологию сигнала большей мощности, обеспечивающей доступ на более высокой скорости на среднее расстояние.

Беспроводные локальные сети стандарта IEEE 802.11b

Сегодня наблюдается быстрый рост беспроводных локальных сетей на университетских факультетах, в офисах компаний, в кафе, в больницах, в домах. Например, пользователи устанавливают недорогие беспроводные локальные сети у себя дома, что обеспечивает домашним возможность одновременного доступа в Интернет из разных комнат. На сегодняшний день разработано несколько стандартов и технологий беспроводных локальных сетей. Но самым популярным из них является стандарт IEEE 802.11b (также известный как беспроводной стандарт Ethernet, или стандарт Wi-Fi), использующий радиосвязь на частоте около 2,4 ГГц и предоставляющий Ethernet-доступ на скорости 11 Мбит/с [102, 202, 226].

Стандарт IEEE 802.1 lb определяет физический уровень и уровень управления доступом к носителю (MAC) для беспроводных локальных сетей. На физическом уровне используется технология DSSS (Direct Sequence Spread Spectrum), когда каждый бит перекодируется в определенную последовательность битов, называемую *чип-кодом*. Эта техника аналогична той, которая применяется в протоколе CDMA (Code Division Multiple Access — множественный доступ с кодовым разделением) с той разницей, что всеми мобильными хостами (и базовыми станциями) используется один и тот же чип-код. По этой причине технология DSSS не является протоколом коллективного доступа, а представляет собой физический механизм, распределяющий энергию сигнала по более широкому диапазону частот, тем самым улучшая способность приемника восстанавливать переданные исходные биты.

Стандарт IEEE 802.11b принадлежит к семейству протоколов беспроводных локальных сетей, совместно называемых IEEE 802.11. В это семейство также входит стандарт IEEE 802.1 la, работающий в диапазоне частот от 5 ГГц до 6 ГГц, в кото-

ром вместо технологии DSSS используется технология OFDM (Orthogonal Frequency Division Multiplexing — ортогональное мультиплексирование с частотным разделением) и обеспечивается скорость передачи данных до 54 Мбит/с. Также входящий в данную группу стандартов стандарт IEEE 802.1 lg, как и 802.1 lb, работает на частоте 2,4 ГГц и обеспечивает скорость передачи данных 54 Мбит/с (как и 802.11a). Вся группа стандартов IEEE 802.11 обладает одинаковой архитектурой и использует один и тот же протокол MAC.

Архитектура локальных сетей стандарта 802.11

На рис. 5.33 изображены основные компоненты архитектуры локальных сетей 802.11. Основным строительным блоком этой архитектуры является *сота,* на языке стандарта 802.11 также называемая базовым служебным набором (Basic Service Set, BSS). Как правило, сота содержит одну или несколько беспроводных станций, а также центральную *базовую станцию,* в стандарте 802.11 также называемую *точкой доступа* (Access Point, AP). Беспроводные станции, которые могут быть стационарными или мобильными, обмениваются данными с базовой станцией по протоколу MAC стандарта IEEE 802.11. Несколько базовых станций могут быть соединены вместе (например, с помощью обычного кабельного Ethernet-соединения или другого беспроводного канала), в результате чего образуется так называемая *распределительная система* (Distribution System, DS). Для протоколов более высокого уровня (например, IP) распределительная система выглядит как единая сеть стандарта 802, во многом подобно тому, как протоколы более высокого уровня воспринимают соединенную мостами кабельную локальную Ethernet-сеть 802.3 как единую сеть стандарта 802.

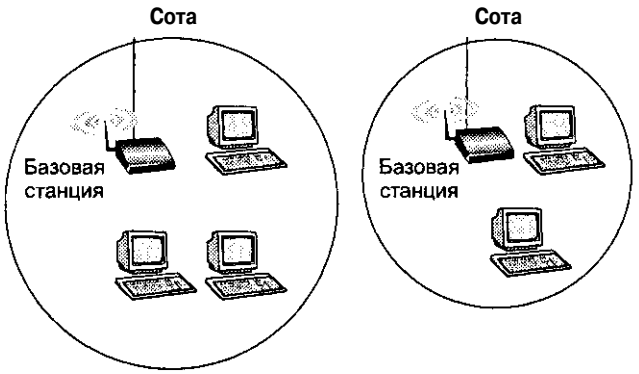


Рис. 5.33. Архитектура локальной сети IEEE 802.11

На рис. 5.34 показано, как могут группироваться станции IEEE 802.11, образуя так называемую спецсеть (ad hoc network) — сеть без центрального управления и без связи с «внешним миром». В этом случае сеть образуется «на лету», просто потому, что несколько мобильных устройств случайно оказались в одном месте и у них появилась необходимость в обмене информацией, а установленной сетевой инфра-

структуры (например, соты с базовой станцией 802.11) в данном месте нет. Спецсеть может быть сформирована несколькими собравшимися вместе людьми с лэптопами (например, в конференц-зале, поезде или автомобиле), когда им нужно обменяться данными в отсутствие централизованной точки доступа. Огромный интерес к подобным сетям объясняется все продолжающимся ростом популярности мобильных устройств. В группе IETF деятельностью, относящейся к спецсетям [229], занимается рабочая группа MANet (Mobile Ad hoc NETworks — мобильные спецсети).

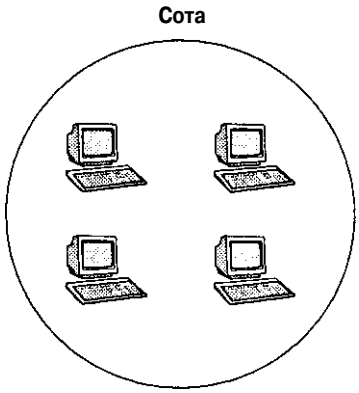


Рис. 5.34. Спецсеть стандарта IEEE 802.11

Протокол доступа к носителю стандарта 802.11

Так же как и в кабельной Ethernet-сети стандарта 802.3, станции беспроводной локальной сети стандарта IEEE 802.11 должны координировать свой доступ и использовать коллективный канал связи (в данном случае радиочастоту). Этим, опять же, занимается протокол управления доступом к носителю (MAC). В сетях стандарта IEEE 802.11 в качестве протокола MAC используется протокол *CSMA/CA* (CSMA with Collision Avoidance — множественный доступ с контролем несущей и предотвращением коллизий). Как было показано в разделе «Ethernet», протокол CSMA сначала прослушивает канал, чтобы определить, не занят ли канал другой станцией, передающей кадр. В спецификации стандарта 802.11 указывается, что физический уровень наблюдает за уровнем радиосигнала, чтобы определить, не ведется ли в данный момент передача другой станцией, и предоставляет эту информацию протоколу MAC. Если канал оказывается свободным в течение времени, большего чем распределенный интервал между кадрами (Distributed Inter Frame Space, DIFS), станция получает разрешение на передачу. Как и в любом протоколе произвольного доступа, кадр будет успешно получен принимающей станцией, если другие станции не начнут передачу и их сигнал не наложится на сигнал, передаваемый данной станцией.

Корректно и полностью получив кадр, получающая станция ждет в течение короткого периода времени SIFS (Short Inter Frame Space — короткий интервал между кадрами), после чего посылает отправителю кадр явного подтверждения. Благодаря

подтверждению отправитель узнает, что получатель получил предназначавшийся ему кадр без повреждений. Это подтверждение необходимо, так как, в отличие от кабельной Ethernet-сети, беспроводной отправитель не может самостоятельно определить, был ли переданный кадр принят успешно (без коллизии с другим кадром). Передача кадра и последующее подтверждение схематично показаны на рис. 5.35.

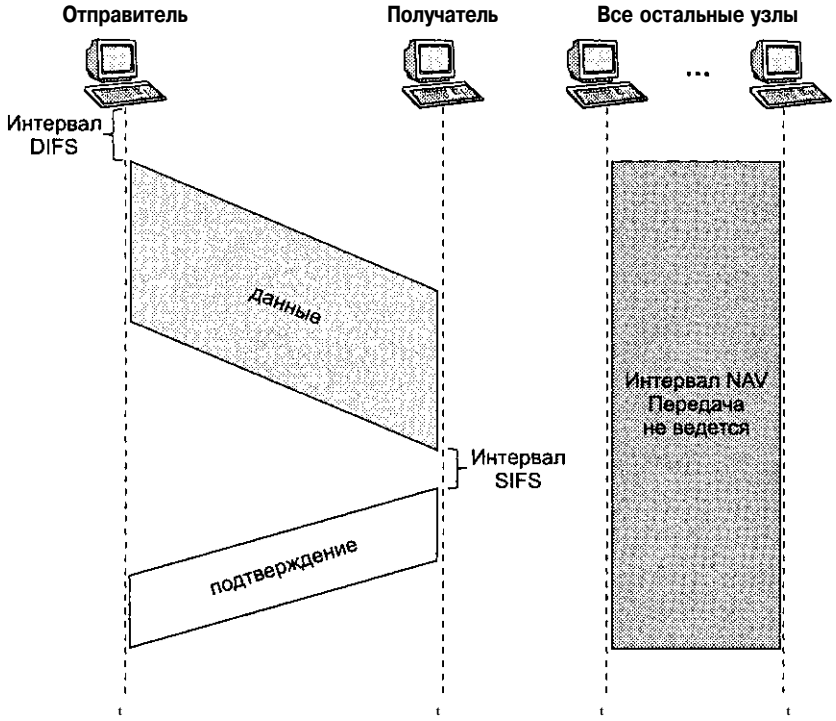


Рис. 5.35. Передача данных и подтверждение в сети IEEE 802.11

На рисунке показана ситуация, в которой отправитель сначала прослушивает канал, убеждаясь, что канал свободен. Что произойдет, если отправитель обнаружит, что канал занят? В этом случае станция выполняет процедуру отката, аналогичную той, что используется в Ethernet-сети. Станция, обнаруживающая, что канал занят, не пытается занимать его до тех пор, пока он не освободится. Как только станция замечает, что канал свободен в течение интервала времени DIFS, она рассчитывает дополнительный интервал времени случайной длительности и ждет в течение этого интервала, после чего, наконец, передает кадр. Как и в Ethernet-сети, таймер случайной длительности позволяет избежать ситуации, при которой несколько станций одновременно начинают передачу (что приводит к коллизии) по истечении интервала DIFS. Как и в Ethernet-сети, максимальная длительность случайного интервала времени удваивается при каждой новой коллизии.

Как уже отмечалось, в отличие от протокола 802.3 Ethernet, беспроводной протокол MAC стандарта 802.11 не обнаруживает коллизии. Это вызвано двумя причинами.

- Для обнаружения коллизий необходима способность одновременно передавать и принимать сигнал. Это позволяет определить, не накладывается ли передача какой-либо другой станции на собственную передачу. В случае радиосвязи реализация подобной способности требует дополнительных затрат.
- Даже если оснастить адаптеры средствами обнаружения коллизий, возможна ситуация, когда коллизия (наложение сигналов) будет выявлена только на приемнике, но ее не будет на передатчике.

Возможность последней ситуации обусловливается особенностями радиоканала. Предположим, что станция A передает данные станции B. Допустим, что станция C также передает данные станции B. Поскольку в качестве носителя в данной ситуации используется радиосвязь, возможна так называемая *проблема скрытого терминала,* когда из-за каких-либо физических препятствий (например, горы) станции A и C не слышат друг друга, хотя при этом передачи обеих станций принимаются станцией B. Пример показан на рис. 5.36, *а*. Кроме того, подобная ситуация может быть вызвана ослаблением силы сигнала с расстоянием. На рис. 5.36, *б* станции A и C расположены так, что силы их сигналов недостаточно, чтобы обнаруживать друг друга, но в то же время достаточно для коллизии их сигналов на станции B.

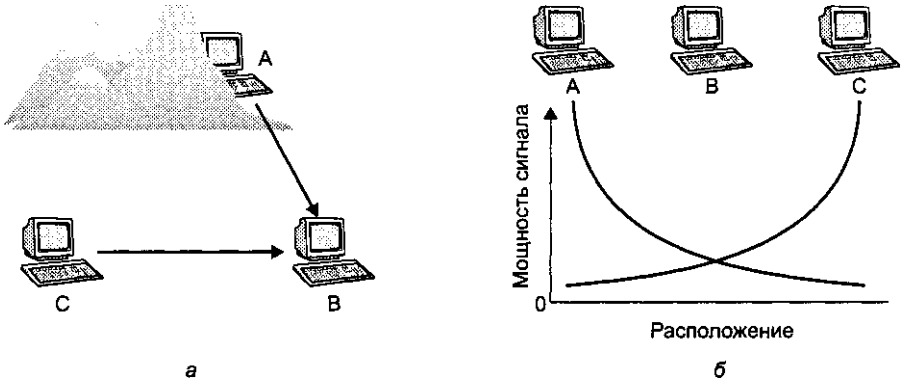


Рис. 5.36. Проблема скрытого терминала (а); проблема затухания сигнала *(б)*

Учитывая трудности с обнаружением коллизий в беспроводной связи, разработчики стандарта IEEE 802.11 создали протокол доступа, основное назначение которого состоит не в том, чтобы обнаруживать коллизии (CSMA/CD), а в том, чтобы их предотвращать (CSMA/CA). Во-первых, кадр IEEE 802.11 содержит поле длительности, в котором передающая станция явно указывает, как долго данный кадр будет занимать канал. Это значение позволяет другим станциям определить минимальное время, в течение которого они должны воздерживаться от передачи. В спецификации стандарта этот интервал времени назван вектором сетевого дос-T?yria (Network Allocation Vector, NAV).

Протокол IEEE 802.11 также может использовать управляющий кадр RTS (Request To Send — готовность к передаче) и короткий кадр CTS (Clear To Send —

готовность к приему) для резервирования доступа к каналу. Когда отправитель хочет послать кадр, он может сначала передать получателю кадр RTS, в котором указывается длительность пакета данных (DATA) и пакета подтверждения (ACK). Приняв кадр RTS, получатель отвечает кадром CTS, таким образом давая отправителю явное разрешение на передачу. Все остальные станции, получив кадры RTS и CTS, узнают о готовящейся передаче и таким образом могут избежать коллизии с ней. Кадры RTS, CTS, а также кадры данных и подтверждения показаны на рис. 5.37. Отправитель в соответствии с протоколом IEEE 802.11 может сначала передать управляющий кадр RTS и получить управляющий кадр CTS, как показано на рис. 5.37, или сразу начать передавать данные (см. рис. 5.35).

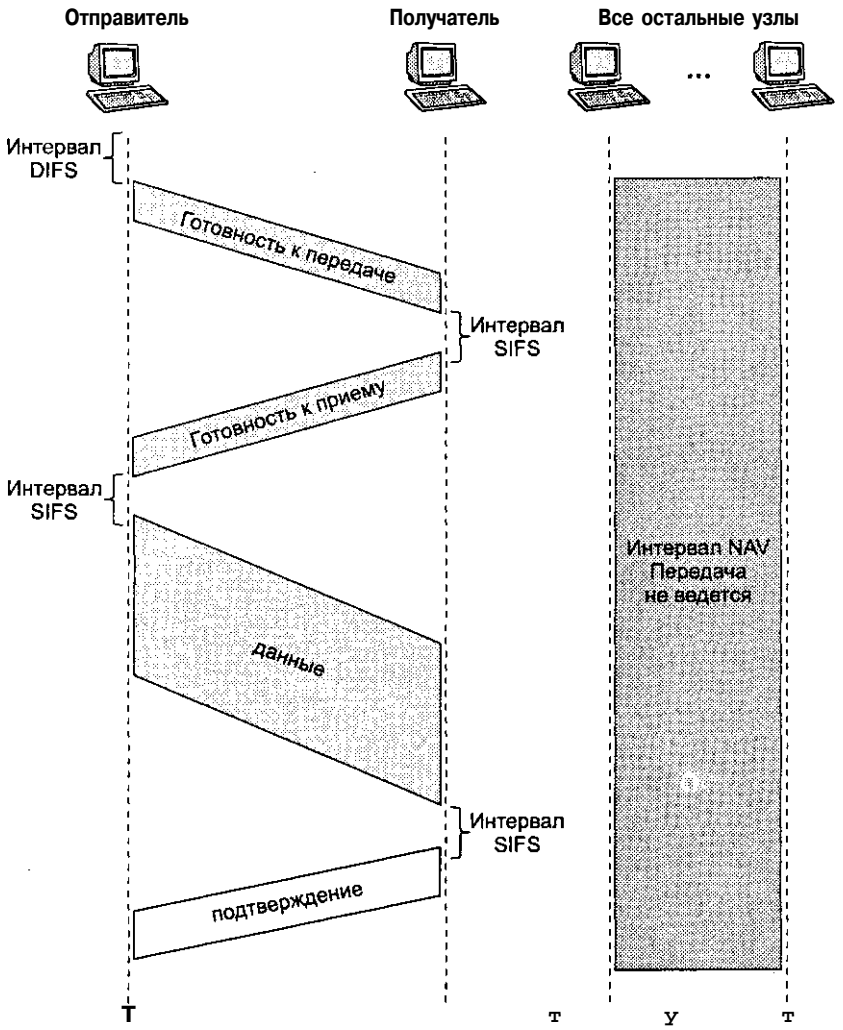


Рис. 5.37. Предотвращение коллизий при помощи кадров RTS и CTS

Использование кадров RTS и CTS позволяет избежать коллизий по двум причинам.

- Переданный получателем кадр CTS будет получен всеми станциями в окружении получателя. Таким образом, кадр CTS помогает избежать как проблемы скрытого терминала, так и проблемы затухания сигнала.
- Поскольку длительность кадров RTS и CTS невелика, коллизия с подобным кадром также должна занимать меньше времени. Обратите внимание, что при успешной передаче кадров RTS и CTS передача последующих кадров данных и подтверждения должна пройти без коллизий.

В приведенном выше описании мы осветили только некоторые ключевые аспекты протокола 802.11. Дополнительную информацию об этом протоколе, касающуюся временной синхронизации, управления питанием, а также присоединения к сети и отсоединения **от** сети, можно узнать в спецификации стандарта IEEE 802.11 [102, 202, 226].

ИСТОРИЧЕСКАЯ СПРАВКА

В последние годы все большую популярность приобретают системы сотовой телефонной связи стандарта 3G, обеспечивающие получение данных на скорости 2 Мбит/с и передачу данных на скорости 384 Кбит/с. Системы 3G работают в лицензированных радиодиапазонах (1885-2025 МГц и 2100-2200 МГц), причем некоторые операторы платят государству по 2000 долларов за каждого абонента. Системы 3G предоставляют пользователям доступ в Интернет с мобильных компьютеров, когда пользователи находятся вне дома. Например, с помощью технологии 3G пользователь может получить доступ к карте дорог во время вождения автомобиля или к информации о репертуаре кинотеатров, загорая на пляже. Тем не менее многие эксперты сегодня задаются вопросом о том, будет ли технология 3G успешна, учитывая ее стоимость, а также конкуренцию со стороны беспроводных локальных сетей [552]. В частности, эксперты приводят следующие аргументы:

- ***Рождающаяся сегодня инфраструктура локальных сетей скоро станет практически повсеместной.*** **Беспроводные локальные сети стандарта IEEE 802.11b, работающие на скорости 11 Мбит/с, получат широкое распространение в частных домах и общественных местах, включая супермаркеты, аэропорты, больницы, офисные здания и территории университетов. Скоро почти все новые мобильные компьютеры будут оснащаться сетевыми картами 802.11. Более того, в новых Интернет-приложениях, таких как фото- и видеокамеры, также будут применяться небольшие адаптеры беспроводных локальных сетей с низким энергопотреблением.**
- ***Основная доля беспроводного трафика данных начинается и заканчивается локально.*** **Учитывая, что беспроводной трафик, начинающийся и заканчивающийся в супермаркетах, офисных зданиях и т. д., переносится недорогими беспроводными локальными сетями, для дорогих систем 3G остается относительно немного трафика.**
- ***Беспроводные базовые станции локальных сетей также могут поддерживать стандарты GSM/GPRS.*** **Это позволяет предоставить недорогой доступ для мобильных телефонов, а также для беспроводных локальных сетей. Таким образом, сотовый телефон без сетевой карты даст возможность получить доступ в Интернет без помощи операторов системы 3G.**

Разумеется, многие другие эксперты полагают, что систему 3G ждет не просто успех, а что эта система кардинально изменит привычные нам методы работы и весь наш образ жизни. Битва началась.

Bluetooth

Технология Bluetooth может применяться в широком спектре приложений, но наиболее интересным ее приложением является предоставление удобного средства беспроводной связи для электронных устройств, таких как мобильные телефоны, лэптопы, карманные и настольные компьютеры, цифровые фото- и видеокамеры, факсы, принтеры, клавиатуры, мыши и джойстики, системы домашней сигнализации, часы и кофеварки. В 90-х годах в большинстве случаев беспроводные соединения между мобильными телефонами, ноутбуками и карманными компьютерами осуществлялись при помощи действующего в пределах зоны прямой видимости инфракрасного сигнала. Поскольку в технологии Bluetooth используется радиосигнал, устройства более не должны находиться в пределах прямой видимости. Кроме того, помимо двухточечных'соединений технология Bluetooth поддерживает многоточечные соединения.

Встроенные в мобильные устройства приемопередатчики Bluetooth работают в нелицензированном радиодиапазоне 2,45 ГГц, предоставляя передачу данных на скорости до 721 Кбит/с, а также три голосовых канала по 64 Кбит/с. Расстояние, на котором могут работать поддерживающие технологию Bluetooth устройства, зависит от мощности этих устройств и составляет от 10 до 100 метров. Чтобы минимизировать возможность взаимного наложения сигналов от других устройств, после приема или передачи пакета частота передачи меняется. Помимо изменения частоты передачи технология Bluetooth обеспечивает прямое исправление ошибок (Forward Error Correction, FEC) и автоматический запрос повторной передачи (Automatic Repeat reQuest, ARQ), при этом для каждого пакета данных вычисляется значение CRC, а полученные с ошибками пакеты передаются повторно (см. главу 3).

У каждого устройства, поддерживающего технологию Bluetooth, есть уникальный 12-разрядный адрес. Чтобы устройство А могло связаться с устройством В, оно должно знать адрес устройства В. Технологией Bluetooth также поддерживается аутентификация устройств и шифрование сообщений.

В набор протоколов Bluetooth входят несколько протоколов.

- *Узкополосный протокол* обеспечивает физическое беспроводное соединение между устройствами. При соединении от двух до семи Bluetooth-устройств образуется небольшая сеть, называемая *piconet.*
- *Протокол управления каналом* отвечает за установку соединения, называемую «рукопожатием», во время которой два устройства обмениваются необходимой информацией.
- *Протокол L2CAP* обеспечивает адаптацию протоколов более высокого уровня к передаче по узкополосному каналу.

Читателям, желающим ознакомиться с технологией Bluetooth, советуем почитать дополнительную литературу [202].

Протокол PPP

До сих пор мы в основном рассматривали протоколы для широковещательных каналов. В данном разделе мы обсудим протокол *PPP* (Point-to-Point Protocol — протокол

передачи от точки к точке) — протокол канального уровня для двухточечных соединений. Поскольку на модемных телефонных линиях, соединяющих компьютеры с Интернет-провайдерами, как правило, применяется протокол PPP, этот протокол, несомненно, является сегодня одним из наиболее популярных протоколов канального уровня. Другим подобным протоколом является протокол HDLC (High-level Data Link Control — высокоуровневый протокол управления каналом) [476], однако здесь мы ограничимся рассмотрением более простого протокола PPP, что позволит нам изучить наиболее важные свойства двухточечных протоколов передачи данных.

Как следует из его названия (протокол передачи от точки к точке), PPP представляет собой протокол канального уровня, работающий на *двухточечных линиях связи,* то есть линиях, напрямую соединяющих два узла (RFC 1661, RFC 2153). Это может быть последовательная телефонная линия (например, соединение по модему со скоростью 56 Кбит/с), линия SONET/SDH, соединение стандарта X.25 или канал ISDN. Как уже отмечалось выше, именно протокол PPP используется для соединения пользователя с Интернет-провайдером по телефонной линии.

Прежде чем углубиться в детали протокола PPP, рассмотрим оригинальные требования, заложенные рабочей группой IETF в идею протокола PPP (RFC 1547).

- *Формирование кадра.* В соответствии с протоколом PPP отправитель данных должен обеспечить инкапсуляцию пакета сетевого уровня в кадре канального уровня таким образом, чтобы получатель мог определить начало и конец как кадра канального уровня, так и пакета сетевого уровня, содержащегося в кадре.
- *Прозрачность.* Протокол PPP не должен накладывать ограничения на то, как выглядят данные на сетевом уровне (данные и заголовки). Таким образом, например, протокол PPP не должен запрещать передачу каких бы то ни было последовательностей битов в пакете сетевого уровня. Мы скоро вернемся к этому вопросу при обсуждении байтовой подстановки.
- *Поддержка различных протоколов сетевого уровня.* Протокол PPP должен обеспечивать одновременную работу различных протоколов сетевого уровня (например, IP и DECnet) в одном и том же физическом канале. Подобно тому как протокол IP обеспечивает мультиплексирование различных протоколов транспортного уровня (например, TCP и UDP) в одном сквозном соединении, протокол PPP должен обеспечивать мультиплексирование различных протоколов сетевого уровня в одном двухточечном соединении. Это требование означает, что протоколу PPP потребуется, по меньшей мере, поле типа протокола или какой-либо подобный механизм, позволяющий принимающей стороне протокола PPP демультиплексировать полученный кадр и передать его соответствующему протоколу сетевого уровня.
- *Поддержка различных типов линий.* Помимо поддержки различных протоколов более высокого уровня протокол PPP должен также обеспечивать работу на различных типах линий связи, включая последовательные (передающие по каналу в данном направлении по одному биту) и параллельные (передающие биты параллельно), синхронные (передающие сигналы таймера вместе с данными) и асинхронные, низкоскоростные и высокоскоростные, электрические и оптические линии.

- *Обнаружение ошибок.* PPP-приемник должен обнаруживать битовые ошибки в принимаемом кадре.
- *Живучесть соединения.* Протокол PPP должен уметь обнаруживать неисправность на канальном уровне (например, невозможность передачи данных по каналу) и сигнализировать об этой неисправности сетевому уровню.
- *Согласование адресов сетевого уровня.* Протокол PPP должен предоставлять механизм, позволяющий обменивающимся данными сетевым уровням (например, IP) узнавать или настраивать адреса сетевого уровня друг друга.
- *Простота.* Помимо перечисленных выше требований протокол PPP должен удовлетворять ряду дополнительных требований, самым важным из которых является «простота». В RFC 1547 говорится: «девизом двухточечного протокола должна быть простота». Трудная задача, учитывая все остальные требования к протоколу PPP! Сегодня различные аспекты этого «простого» протокола описываются более чем 50 документами RFC.

Хотя может показаться, что к протоколу PPP предъявляется очень много требований, в действительности ситуация могла быть еще сложнее. В спецификации протокола PPP также явно указываются функции, которые протокол PPP *не* должен реализовывать.

- *Исправление ошибок.* Протокол PPP должен обнаруживать ошибки, но *не должен* исправлять их.
- *Управление потоком.* Предполагается, что PPP-приемник способен принимать кадры на полной скорости, с которой способен их переносить физический носитель. Если более высокий уровень не может принимать кадры на данной максимальной скорости, то проблема должна решаться на более высоком уровне путем отбрасывания пакетов, которые получатель не успевает обработать, или путем снижения скорости передачи данных отправителем. Таким образом, скорость передачи данных снижается не протоколом PPP, а протоколом более высокого уровня, который «спускает» пакеты протоколу PPP.
- *Порядок доставки кадров.* Протокол PPP *не обязан* доставлять кадры получателю в том же порядке, в котором они были отправлены. Интересно отметить, хотя подобное отсутствие требования доставки кадров с сохранением порядка следования соответствует модели обслуживания протокола IP (который также может доставлять пакеты конечному получателю в произвольном порядке), другие протоколы сетевого уровня, работающие поверх протокола PPP, обязаны доставлять пакеты с сохранением порядка их следования.
- *Многоточечные линии.* Протокол PPP должен работать только на линиях с одним передатчиком и одним приемником. Другие протоколы (например, HDLC) могут поддерживать линии с несколькими приемниками (как, например, в сетях Ethernet) на одной линии связи.

Формат кадра протокола PPP

На рис. 5.38 показан кадр протокола PPP, формат которого близок формату HDLC-кадров.

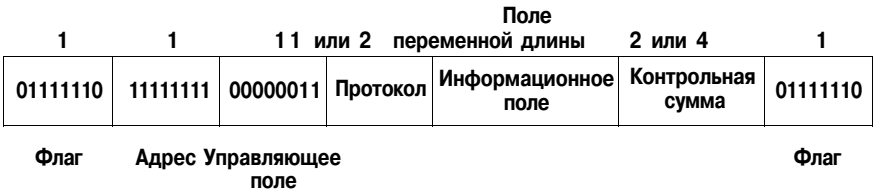


Рис. 5.38. Формат кадра протокола PPP

Ниже перечислены поля кадра протокола PPP.

- *Поле флага.* Каждый PPP-кадр начинается и заканчивается однобайтовым полем флага, в котором содержится значение 01111110.
- *Поле адреса.* Единственное возможное значение этого поля — 11111111.
- *Управляющее поле.* Единственное возможное значение этого поля — 00000011. Поскольку поле адреса и управляющее поле могут содержать только фиксированные значения, непонятно, зачем вообще нужны эти поля. В спецификации протокола PPP утверждается (RFC 1662), что «эти поля могут быть определены позднее». Поскольку у них фиксированные значения, протокол PPP разрешает отправителю вообще не посылать адресный и управляющий байты, что позволяет экономить два байта в каждом кадре.
- *Протокол.* Поле протокола сообщает PPP-получателю, какому протоколу более высокого уровня принадлежат инкапсулированные данные (то есть содержимое информационного поля PPP-кадра). Получив PPP-кадр, PPP-приемник проверяет корректность кадра, после чего передает содержащиеся в информационном поле данные соответствующему протоколу. В документах RFC 1700 и RFC 3232 определены 16-разрядные коды, используемые протоколом PPP. Нас будет интересовать протокол IP (то есть ситуация, когда инкапсулированные в PPP-кадре данные представляют собой IP-дейтаграмму), которому соответствует шестнадцатеричное значение 21 в этом поле. Кроме того, с протоколом PPP могут использоваться такие сетевые протоколы, как AppleTalk (29), DECnet (27), протокол управления каналом PPP (C021) и протокол IPCP (8021). Протокол IPCP (IP Control Protocol — протокол управления протоколом IP) вызывается протоколом PPP при первой активизации линии для конфигурирования соединения IP-уровня между двумя поддерживающими протокол IP устройствами.
- *Информационное поле.* Это поле содержит инкапсулированный пакет (данные), посылаемый протоколом более высокого уровня (например, IP) по PPP-линии. По умолчанию максимальная длина информационного поля составляет 1500 байт, хотя, как будет сказано ниже, эта величина может быть изменена во время первого конфигурирования линии.
- *Контрольная сумма.* Поле контрольной суммы используется для обнаружения ошибок в передаваемом кадре. В качестве контрольной суммы используется либо 2-байтовый, либо 4-байтовый циклический избыточный код (CRC) стандарта HDLC.

Прежде чем завершить наше обсуждение формата кадра протокола PPP, рассмотрим проблему поля флага, призванного обозначать начало и конец кадра. Что произойдет, если последовательность битов в этом поле встретится где-либо еще в пакете? Например, что случится, если стандартное для поля флага значение 01111110 встретится в информационном поле? Может ли получатель по ошибке принять этот байт за конец кадра?

Один из способов решения проблемы заключается в том, чтобы запретить протоколам более высокого уровня передавать данные, содержащие данную последовательность битов. Однако такой подход противоречит требованию прозрачности протокола PPP. Альтернативное решение, используемое в протоколе PPP, а также во многих других протоколах, представляет собой технический прием, называемый *байтовой подстановкой* (byte stuffing).

Протоколом PPP определяется специальный управляющий префиксный байт 01111101. Если где-либо в кадре помимо поля флага встречается флаговая последовательность битов 01111110, протокол PPP предваряет этот байт управляющим префиксным байтом. Таким образом, вставляя управляющий байт перед байтом 01111110, протокол PPP указывает, что следующий байт 01111110 не является флагом, а представляет собой данные. Получатель, встречая байт 01111101, за которым следует 01111110, удаляет управляющий префиксный байт, восстанавливая исходные данные. Аналогично, если сам управляющий префиксный байт встречается в данных, он также предваряется управляющим префиксным байтом. Таким образом, когда получатель встречает одиночный управляющий префиксный байт, он понимает, что следом в потоке данных идет флаговая последовательность. Пара управляющих префиксных байтов подряд означает, что подобный байт идет следом в потоке данных. Пример байтовой подстановки показан на рис. 5.39. В действительности протокол PPP инвертирует в предваряемом префиксом байте 5-й бит (выполняет с ним операцию XOR с шестнадцатеричным числом 20), но мы для простоты опустили эту деталь.

Рис. 5.39. Байтовая подстановка

Протоколы управления каналом и сетью

До сих пор мы рассматривали формирование кадров из данных, передаваемых с помощью протокола PPP по двухточечному соединению. Но как инициали-

зируется линия, когда на одном конце линии в сеть включается хост или маршрутизатор? Инициализацией, поддержкой, сообщением об ошибках и отключением PPP-линии занимается протокол LCP (Link Control Protocol — протокол управления каналом), а также семейство протоколов управления сетью протокола PPP.

Прежде чем передавать какие-либо данные по PPP-линии, две одноранговые сущности (на каждом конце PPP-линии) должны проделать существенную часть работы по настройке линии, что во многом напоминает тройное рукопожатие отправителя и получателя в протоколе TCP (см. раздел «Протокол TCP — передача с установлением соединения» в главе 3), когда перед передачей TCP-сегментов устанавливаются параметры TCP-соединения. На рис. 5.40 показана диаграмма состояний для протокола LCP, осуществляющего конфигурирование, поддержку и разрыв PPP-соединения.

Рис. 5.40. Диаграмма состояний протокола LCP

PPP-линия начинает и заканчивает работу в пассивном состоянии. Когда какое-либо событие, например обнаружение сигнала в линии или вмешательство сетевого администратора, указывает на готовность физического уровня, протокол PPP переходит в состояние установки соединения. В этом состоянии один из концов линии посылает свои настроечные параметры в кадре запроса на конфигурирование протокола LCP (PPP-кадре, в поле протокола которого установлен код протокола LCP, а параметры настройки помещены в информационное поле). Другая сторона линии отвечает кадром положительной квитанции (все параметры принимаются), кадром отрицательной квитанции (все параметры понятны, но не принимаются) или кадром отказа в конфигурировании (параметры не распознаны или неприемлемы). Параметры конфигурации протокола LCP включают максимальный размер кадра, спецификацию протокола аутентификации (если она используется), а также возможность пропуска адресного и контрольного полей в PPP-кадрах.

После того как соединение установлено, процедура аутентификации выполнена и обменивающиеся данными стороны договорились о параметрах соединения, обе стороны соединения обмениваются друг с другом пакетами управления сетью, специфичными для конкретного сетевого протокола. Если поверх протокола PPP работает сетевой протокол IP, для настройки IP-модулей на каждом конце PPP-соединения применяется протокол IPCP (RFC 1332). Данные протокола IPCP переносятся в PPP-кадре (при этом в поле протокола устанавливается значение 8021), так же как и данные протокола LCP. Протокол IPCP позволяет двум IP-модулям узнать или настроить IP-адреса друг друга, а также договориться о том, будут ли IP-дейтаграммы передаваться в сжатом виде. Аналогичные протоколы управления сетью определены для других протоколов сетевого уровня, таких как DECnet (RFC 1762) и AppleTalk (RFC 1378). Как только сетевой уровень настроен, протокол PPP может начать передачу дейтаграмм сетевого уровня — линия находится в активном (открытом) состоянии, и по ней начинают перемещаться данные. Для проверки состояния линии две PPP-сущности могут обменяться кадрами эхо-запроса и эхо-ответа.

PPP-линия остается в настроенном и активном состоянии до тех пор, пока одна из сторон не отправит LCP-кадр с запросом о разрыве соединения. Если одна из сторон PPP-линии отправляет подобный LCP-кадр, а другая отвечает ей LCP-кадром подтверждения разъединения, линия переходит в пассивное состояние.

Итак, протокол PPP представляет собой протокол канального уровня, с помощью которого две сущности канального уровня обмениваются PPP-кадрами с дейтаграммами сетевого уровня. Основные составляющие протокола PPP перечислены ниже.

- *Формирование кадров.* Метод инкапсуляции данных в PPP-кадре, определения начала и конца кадра, а также обнаружения ошибок в кадре.
- *Протокол управления каналом.* Протокол для инициализации, поддержки и разрыва PPP-соединения.
- *Протоколы управления сетью.* Семейство протоколов по одному для каждого сетевого протокола, позволяющих модулям сетевого уровня настраивать друг друга, прежде чем по PPP-линии начнут передаваться дейтаграммы сетевого уровня.

Технология ATM

Стандарты для технологии ATM были разработаны в середине 80-х годов. Многие читатели этой книги, скорее всего, слишком молоды, чтобы помнить те времена. В те годы доминировали два типа сетей: телефонные сети, изначально использовавшиеся (и использующиеся до сих пор) для передачи голосового сигнала реального времени, и информационные сети, изначально применявшиеся для передачи текстовых файлов, поддержки удаленных терминалов и обслуживания электронной почты. Кроме того, в те времена существовали частные сети, обеспечивающие возможность проведения видеоконференций. Интернет уже суще-

ствовал, но мало кто думал об использовании его для передачи телефонных разговоров, а о технологии web тогда еще никто не слышал. Поэтому неудивительно, что в те времена была спроектирована сетевая технология, способная передавать как аудио и видео в реальном времени, так и текст, электронную почту и изображения. Данная технология получила название *ATM* (Asynchronous Transfer Mode — режим асинхронной передачи). Стандарты для широкополосных цифровых сетей разрабатывались двумя организациями: ATM-форумом [503] и Международным союзом телекоммуникаций (International Telecommunications Union, ITU) [509].

Стандартами ATM предусматривается коммутация пакетов с использованием виртуальных каналов (также иногда называемых виртуальными цепями) и определяются интерфейсы приложений с ATM. Таким образом, технология ATM предоставляет законченное сетевое решение для распределенных приложений. Параллельно с разработкой стандартов ATM крупные компании во всем мире вкладывают значительные средства в исследовательские работы и проектирование в области ATM. В результате этих инвестиций появилось множество высокопроизводительных технологий ATM, включая ATM-коммутаторы, работающие на скоростях в несколько терабит в секунду. В последние годы технология ATM весьма агрессивно заявляет о себе на рынке как в телефонных сетях, так и на магистралях Интернета.

Однако в локальных сетях успехи технологии ATM не столь значительны. И пока не ясно, будет ли эта технология существенно представлена в мире настольных персональных компьютеров. В самом деле, пока ATM создавала свои комитеты по стандартам и исследовательские лаборатории в конце 80-х — начале 90-х, Интернет и протоколы TCP/IP уже успешно работали и развивались.

- Набор протоколов TCP/IP интегрирован в большинство наиболее популярных операционных систем.
- Компании начали заниматься коммерцией в Интернете.
- Доступ в Интернет стал дешевле.
- Для TCP/IP-сетей было разработано множество замечательных настольных приложений, включая web, Интернет-телефонию, а также интерактивное видео. В настоящее время тысячи компаний разрабатывают новые приложения и службы для Интернета.

Сегодня мы живем в мире, в котором большинство сетевых приложений рассчитано на работу только с протоколами TCP/IP. Вместе с тем ATM-коммутаторы способны пропускать данные на очень высоких скоростях, и, следовательно, они находят применение в магистральных сетях Интернета, где необходимость в передаче данных с высокой скоростью особенно остра. Когда технология ATM применяется в магистралях Интернета, протоколы TCP/IP работают поверх протокола ATM и рассматривают ATM-сеть, которая может охватывать целый континент, как одну большую сеть канального уровня. Другими словами, хотя технология ATM не стала популярной «на ниве» соединения процессов или даже отдельных компьютеров, она заняла свою нишу на канальном уровне в Интернет-магистралях (данную

конфигурацию, называемую IP поверх ATM, мы обсудим в подразделе «1P поверх ATM» данного раздела). Именно поэтому мы включили наше обсуждение технологии ATM в главу, посвященную канальному уровню, а не в предыдущую главу о сетевом уровне.

Основные характеристики ATM

Ниже перечислены основные характеристики ATM.

- Стандарт ATM определяет полный набор протоколов обмена данными от прикладного до физического уровня.
- Модели обслуживания ATM включают обслуживание с постоянной битовой скоростью (Constant Bit Rate, CBR), с переменной битовой скоростью (Variable Bit Rate, VBR), с доступной битовой скоростью (Available Bit Rate, ABR) и с неуказанной битовой скоростью (Unspecified Bit Rate, UBR). Некоторые из этих моделей уже обсуждались нами в разделе «Модели сетевого обслуживания» главы 4.
- В ATM применяется коммутация пакетов фиксированной длины 53 байта. В терминах ATM эти пакеты называются *ячейками.* Каждая ячейка состоит из 5-байтового заголовка и 48-байтовой «полезной нагрузки». Фиксированная длина ячеек и простота заголовков упрощают высокоскоростную коммутацию ATM-ячеек.
- В ATM-сетях используются *виртуальные каналы.* Номер виртуального канала, называемый *идентификатором виртуального канала* (Virtual Channel Identifier, VCI), помещается в специальное поле заголовка ATM-ячейки. Идентификаторы VCI используются коммутаторами для направления ATM-ячеек их адресатам.
- Технология ATM не предоставляет повторной передачи ячеек на канальном уровне. Если коммутатор обнаруживает ошибку в заголовке ATM-ячейки, он пытается исправить ее при помощи помехоустойчивых кодов. Если исправить ошибку не удается, коммутатор не запрашивает повторную передачу у предыдущего коммутатора, а просто отбрасывает ячейку.
- Технология ATM обеспечивает борьбу с перегрузкой только при обслуживании класса ABR (см. раздел «Модели сетевого обслуживания» в главе 4). Метод борьбы с перегрузкой, применяемый в службе ABR, рассматривался в разделе «Принципы контролирования перегрузки» главы 3, где было показано, что этот метод принадлежит к общему классу методов борьбы с перегрузкой, использующих возможности сети. ATM-коммутаторы предоставляют обратную связь передающим оконечным системам, помогая им регулировать скорость передачи данных во время сетевых перегрузок.
- ATM-сети могут работать поверх практически любого физического уровня. Эта технология часто применяется поверх оптоволоконных кабелей, использующих стандарт SONET со скоростями передачи данных 155,52 Мбит/с, 622 Мбит/с и выше.

Как показано на рис. 5.41, стек протоколов ATM состоит из трех уровней: физического уровня ATM, уровня ATM и уровня адаптации ATM (ATM Adaptation Level, AAL).

- *Физический уровень A TM* имеет дело с напряжениями, синхронизацией битов и формированием кадров, передаваемых по физическому носителю.
- *Уровень A TM* представляет собой ядро стандарта ATM. Он определяет структуру ATM-ячейки.
- *Уровень адаптации ATM* в определенной степени соответствует транспортному уровню стека протоколов Интернета. Разработано несколько типов протокола AAL для поддержки различных типов служб.

Рис. 5.41. Три уровня ATM

На сегодня ATM в основном применяется как технология канального уровня в локализованных областях Интернета. Для поддержки интерфейса с набором протоколов TCP/IP была разработана специальная разновидность протокола AAL, AAL5. Протокол AAL5 готовит IP-дейтаграммы для передачи их по ATM-линиям, а также вновь собирает IP-дейтаграммы из ATM-ячеек. На рис. 5.42 изображен стек протоколов для области Интернета, в которой используется технология ATM. Обратите внимание, что в данной конфигурации три уровня ATM «втиснуты» в два нижних уровня стека протоколов Интернета. В частности, сетевой уровень Интернета рассматривает ATM как протокол канального уровня. Прекрасный учебный материал по теме ATM, в котором отражены исходные цели, ставившиеся при разработке этой технологии, имеется в [295].

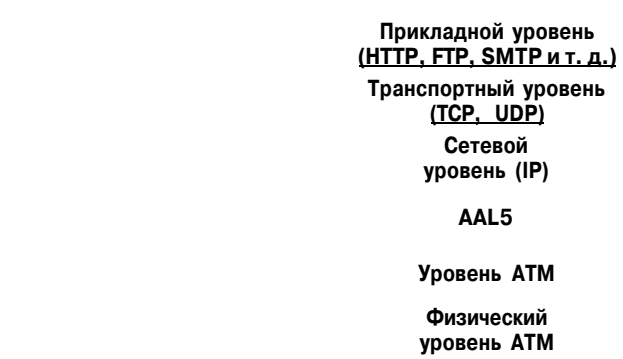


Рис. 5.42. Протоколы Интернета поверх протоколов ATM

Физический уровень ATM

Физический уровень занимается передачей ATM-ячейки по одной физической линии. Как показано в табл. 5.5, физический уровень состоит из двух подуровней: подуровня PDM (Physical Medium Dependent — подуровень, зависимый от физического носителя) и подуровня TC (Transmission Convergence — подуровень конвергенции передачи).

Таблица 5.5. Два подуровня физического уровня и их сферы ответственности

Подуровень	Сферы ответственности
Подуровень TC	**Вставка холостых (не несущих полезной нагрузки) ячеек, определение границ ячейки, адаптация передачи кадра**
Подуровень PDM	**Физический носитель, напряжение сигнала и синхронизация, структура кадра**

Подуровень PMD

Подуровень PMD располагается в самом низу стека протоколов ATM. Как видно из его названия, подуровень PMD определяется физическим носителем линии связи. В частности, спецификация подуровня PMD различается для разных физических носителей (оптического волокна, медного кабеля и т. д.). Подуровень PMD отвечает за передачу и прием отдельных битов. Существуют два класса подуровней PMD: подуровни PMD, имеющие структуру кадров передачи (например, T1, T3, SONET или SDH), и подуровни PMD, не имеющие такой структуры. Если у подуровня PMD есть структура кадров передачи, тогда он отвечает за формирование кадров и определение их границ. (Употребляемый в данном разделе термин «кадр» не следует путать с кадром канального уровня, упоминавшимся в начале этой главы. Кадр передачи представляет собой механизм физического уровня, вроде мультиплексирования с временным разделением, позволяющий организовать передачу битов по линии.) Подуровень PDM имеет дело с битами и не «видит» ячеек. Ниже перечислены некоторые возможные варианты подуровней PDM.

- SONET/SDH (Synchronous Optical Network/Synchronous Digital Hierarchy — синхронная оптическая сеть/синхронная цифровая иерархия) поверх одномодового оптоволоконного кабеля. Подобно линиям T1 и T3, технологии SONET и SDH обладают структурой кадров, позволяющей выполнять синхронизацию битов между передатчиком и приемником. Некоторые скорости стандартизированы, например:
  - OC-1 -51,84 Мбит/с;
  - OC-3 - 155,52 Мбит/с;
  - OC-12 -622,08 Мбит/с.
- T1/T3-кадры поверх оптоволоконного кабеля, микроволн (волн сантиметрового диапазона) и медного кабеля.
- Вариант, основанный на ячейках, а не на кадрах передачи. В этом случае таймер в приемнике настраивается по сигналу.

Подуровень TC

Уровень ATM специфицируется независимо от физического уровня. Он не имеет представления о технологии SONET, линиях T1 и физическом носителе. Таким образом, возникает потребность в подуровне, имеющемся на передающей стороне линии для получения ATM-ячеек от уровня ATM и подготовки их к передаче по физическому носителю и на приемной стороне линии для группирования поступающих от физического уровня битов в ячейки и передачи их уровню ATM. Этим занимается подуровень конвергенции передачи (TC), располагающийся над подуровнем PMD и под уровнем ATM. Следует отметить, что подуровень TC также зависит от физического носителя — если мы поменяем физический носитель или лежащую в его основе структуру кадра передачи, тогда нам придется также сменить и подуровень TC.

На передающей стороне подуровень TC помещает ATM-ячейки в структуру кадра подуровня PDM. На приемной стороне он извлекает ATM-ячейки из структуры кадра подуровня PDM, а также проверяет контрольную сумму заголовка. Более детально подуровень TC выполняет следующие задачи.

- На передающей стороне подуровень TC формирует для каждой передаваемой ATM-ячейки байт контрольной суммы заголовка. На приемной стороне подуровень TC при помощи байта контрольной суммы заголовка исправляет все однобитовые ошибки в заголовке, тем самым снижая вероятность некорректной маршрутизации ячеек. Контрольная сумма заголовка вычисляется по первым 32 битам заголовка, для чего используется восьмибитовая техника полиномиальных кодов, обсуждавшаяся в подразделе «Циклический избыточный код» раздела «Обнаружение и исправление ошибок».
- На приемной стороне подуровень TC определяет границы между ячейками. Если используется подуровень PMD, основанный не на кадрах, а на ячейках, тогда определение границ ячеек, как правило, выполняется путем проверки контрольной суммы заголовка на всей непрерывной последовательности из 40 бит (то есть пяти байтов). Если контрольная сумма совпадает, подуровень TC решает, что начало ячейки найдено. Если контрольная сумма совпадает у четырех ячеек, считается, что синхронизация установлена, и последующие ячейки передаются уровню ATM.
- Если используется подуровень PMD, основанный не на кадрах, а на ячейках, тогда, не получив вовремя ячейки от уровня ATM, подуровень сам посылает холостую ячейку, формируя непрерывный поток ячеек. Получающий подуровень TC не передает холостые ячейки уровню ATM. Холостые ячейки особым образом помечаются в поле типа полезной нагрузки заголовка.

Уровень ATM

При работе протокола IP поверх ATM ATM-ячейки играют роль кадра канального уровня. Структура ATM-ячейки и значение каждого поля ячейки определяются уровнем ATM. Первые 5 байт ячейки составляют заголовок. Остальные 48 байт несут полезную нагрузку. Структура заголовка ATM-ячейки показана на рис. 5.43.

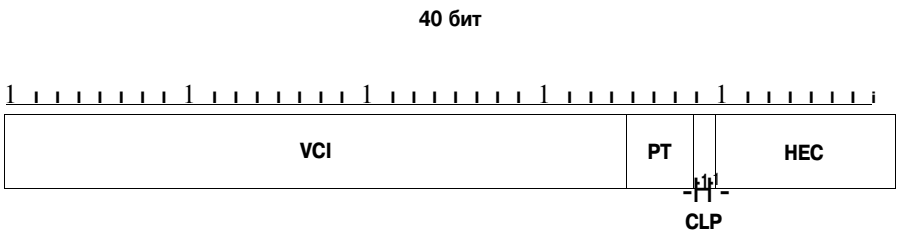


Рис. 5.43. Формат заголовка ATM-ячейки

Ниже перечислены поля заголовка ATM-ячейки.

- *VCI* (Virtual Channel Identifier — идентификатор виртуального канала). Определяет виртуальный канал, которому принадлежит ячейка. Как и в большинстве сетевых технологий, использующих виртуальные каналы, идентификатор виртуального канала ячейки транслируется от линии к линии (см. раздел «Ядро компьютерных сетей» в главе 1).
- *PT* (Payload Type — тип полезной нагрузки). Указывает тип полезной нагрузки, содержащейся в ячейке. Определены несколько типов полезной нагрузки для данных, еще несколько типов полезной нагрузки для управляющих ячеек, а также тип полезной нагрузки для холостых ячеек. (Вспомним, что холостые ячейки иногда бывают нужны физическому уровню для синхронизации.)
- *CLP* (Cell-Loss Priority — приоритет отбрасывания ячеек). Бит CLP может устанавливаться источником для разграничения высокоприоритетного и низкоприоритетного трафиков. Если возникает затор и ATM-коммутатору приходится отбрасывать ячейки, при помощи этого бита коммутатор может отфильтровывать ячейки низкоприоритетного трафика.
- *HEC* (Header Error Control — контрольная сумма заголовка). Восемь избыточных битов, позволяющих обнаруживать и исправлять некоторые ошибки в заголовке ячейки.

Виртуальные каналы

Прежде чем источник сможет начать передачу ячеек адресату, ATM-сеть должна установить *виртуальный канал* (VC) от источника до адресата. Понятие виртуального канала уже обсуждалось в разделе «Ядро компьютерных сетей» главы 1. Виртуальный канал образуется из линий связи, расположенных между отправителем и получателем. Каждая из линий, входящих в виртуальный канал, идентифицируется собственным номером виртуального канала. При установке или разрыве виртуального канала таблицы трансляции номеров виртуального канала должны обновляться (см. главу 1). Как уже отмечалось ранее, в ATM-магистралях Интернета часто применяются постоянные виртуальные каналы, что позволяет избавиться от необходимости устанавливать и разрывать виртуальный канал в динамическом режиме.

Уровень адаптации ATM

Назначение уровня AAL (ATM Adaptation Level — уровень адаптации ATM) заключается в том, чтобы позволить существующим протоколам (например, IP) и

приложениям (например, потоковое видео с постоянной скоростью передачи данных) работать поверх ATM-сети. Как показано на рис. 5.44, уровень AAL реализуется только на оконечных точках ATM-сети. Данной оконечной точкой может быть хост (если ATM обеспечивает сквозное соединение от хоста к хосту) или IP-маршрутизатор (если ATM используется для соединения двух 1P-маршрутизаторов). В этом случае уровень AAL аналогичен транспортному уровню в стеке протоколов Интернета.

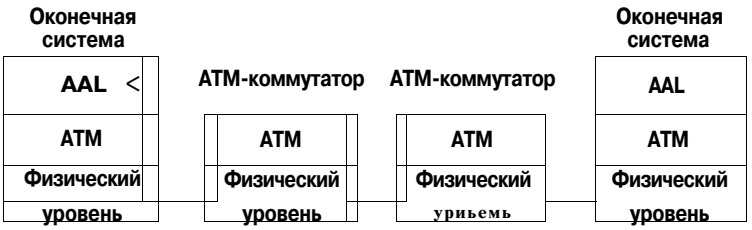


Рис. 5.44. Уровень AAL реализуется только на конечных точках ATM-сети

У уровня AAL есть собственные поля в заголовке. Как видно из рис. 5.45, эти поля занимают небольшую часть поля полезной нагрузки ATM-ячейки.

Рис. 5.45. AAL-поля в поле полезной нагрузки ATM

Несколько версий уровня AAL были стандартизированы международным союзом телекоммуникаций (ITU) и ATM-форумом. Ниже приводятся некоторые наиболее важные варианты уровня AAL и поддерживаемые ими модели обслуживания ATM:

- AAL 1 — для обслуживания с постоянной битовой скоростью (CBR) и эмуляции цепи;
- AAL 2 — для обслуживания с переменной битовой скоростью (VBR);
- AAL 5 — для передачи данных (например, 1P-дейтаграмм).

Структура AAL

Уровень AAL состоит из двух подуровней: *SAR* (Segmentation And Reassembly — сегментация и повторная сборка) и *CS* (Convergence Sublayer — подуровень конвергенции). Как показано на рис. 5.46, подуровень SAR располагается сразу над уровнем ATM, а подуровень CS — между пользовательским приложением и подуровнем SAR. Данные более высоких уровней (например, IP-дейтаграмма) сначала

дает ATM-сеть, выходным маршрутизатором. Входной маршрутизатор выполняет следующие действия.

1. Маршрутизатор изучает адрес получателя дейтаграммы.
2. С помощью таблицы маршрутизации входной маршрутизатор определяет IP-адрес выходного маршрутизатора (то есть следующего маршрутизатора на пути дейтаграммы).
3. Чтобы доставить дейтаграмму к выходному маршрутизатору, входной маршрутизатор рассматривает ATM как еще один протокол канального уровня. Чтобы передать дейтаграмму следующему маршрутизатору, необходимо определить его физический адрес. Как отмечалось в разделе «Адресация в локальных сетях и протокол ARP», определение физического адреса осуществляется при помощи протокола ARP. В частности, входной маршрутизатор по IP-адресу выходного маршрутизатора находит в ARP-таблице его ATM-адрес.
4. Протокол IP входного маршрутизатора передает дейтаграмму канальному уровню (то есть ATM) вместе с ATM-адресом выходного маршрутизатора.

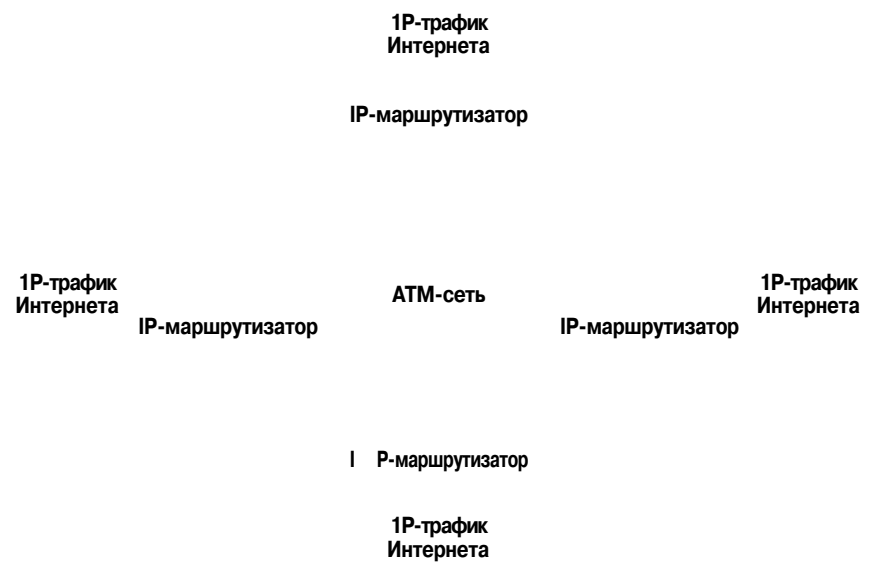


Рис. 5.49. ATM-сеть в центре магистрали Интернета

После того как выполнятся эти четыре действия, ответственность по перемещению дейтаграммы передается от протокола IP протоколу ATM. Протокол ATM должен переместить дейтаграмму узлу с ATM-адресом, полученным на шаге 3. Эта задача распадается на две подзадачи.

- Нахождение идентификатора виртуального канала, ведущего к получающему узлу ATM-сети.
- На передающей стороне виртуального канала (то есть на входном маршрутизаторе) дейтаграмму необходимо разбить на ячейки, а на приемной стороне вир-

туального канала (то есть на выходном маршрутизаторе) ее следует вновь собрать.

Первая из этих задач проста. Интерфейс на передающей стороне поддерживает таблицу соответствий ATM-адресов и идентификаторов виртуального канала. Поскольку мы предполагаем, что виртуальный канал постоянный, эта таблица не изменяется со временем и не может устареть. (Если бы виртуальный канал не был постоянным, тогда для динамической установки и разрыва виртуального канала потребовался бы сигнальный протокол ATM.) Вторая задача заслуживает более пристального рассмотрения. Один из методов заключается в том, чтобы использовать IP-фрагментацию, обсуждавшуюся в подразделе «Фрагментация IP-дейтаграмм» раздела «Интернет-протокол» главы 4. При этом передающий маршрутизатор должен сначала разбить оригинальную дейтаграмму на отдельные фрагменты размером не более 48 байт, чтобы каждый фрагмент поместился в поле полезной нагрузки ATM-ячейки. Но у данного метода есть один существенный недостаток: каждая IP-дейтаграмма, как правило, содержит 20-байтовый заголовок, так что в ATM-ячейке, переносящей фрагмент, будет 25 байт накладных расходов и только 28 байт полезной информации. Для предоставления более эффективного способа фрагментации и повторной сборки дейтаграммы используется уровень AAL5.

Вспомним, что протокол IP входного маршрутизатора передает дейтаграмму уровню ATM вместе с ATM-адресом выходного маршрутизатора. ATM-протокол на входном маршрутизаторе по ATM-адресу получателя находит идентификатор ведущего к нему виртуального канала. Из IP-дейтаграммы уровень AAL5 формирует ATM-ячейки по следующей схеме.

1. Дейтаграмма инкапсулируется в единицу обмена подуровня CPCS, формат которой показан на рис. 5.47.
2. Единица обмена подуровня CPCS разбивается на 48-байтовые фрагменты. Каждый фрагмент помещается в поле полезной нагрузки ATM-ячейки.
3. Во всех ячейках, кроме последней, третий бит поля PT устанавливается на 0. В последней ячейке он устанавливается в 1.

Затем уровень AAL5 передает ячейки уровню ATM. Уровень ATM устанавливает значения поля идентификатора виртуального канала и бит приоритета отбрасывания ячеек (CLP), после чего передает каждую ячейку подуровню TC. Для каждой ячейки подуровень TC вычисляет контрольную сумму заголовка и помещает ее в поле HEC. Далее подуровень TC передает ячейку подуровню PMD.

После этого ATM-сеть перемещает каждую ячейку по сети к получателю, используя ATM-адрес. На каждом ATM-коммутаторе между ATM-отправителем и ATM-получателем ATM-ячейка обрабатывается физическим уровнем ATM и уровнем ATM, но не уровнем AAL. Как правило, на каждом коммутаторе транслируется идентификатор виртуального канала (см. раздел «Ядро компьютерных сетей» в главе 1), после чего контрольная сумма заголовка вычисляется заново. Прибывающие к ATM-получателю ATM-ячейки направляются в AAL-буфер, выделенный для определенного виртуального канала. Затем с помощью бита **AAMndicate,** указывающего на последнюю ячейку (см. рис. 5.48), восстанавливается единица об-

мена подуровня CPCS. Наконец, IP-дейтаграмма извлекается из единицы обмена подуровня CPCS и передается вверх по стеку протоколов уровню IP.

Frame Relay

В данном разделе мы обсудим две сквозные технологии глобальных сетей (Wide-Area Network, WAN), а именно стандарты X.25 и Frame Relay (ретрансляция кадров). Появившийся в начале 80-х годов и пользовавшийся популярностью в Европе вплоть до середины 90-х, стандарт X.25 представляет собой первую публичную технологию коммутации пакетов. Технология Frame Relay, преемница стандарта X.25, является другой публичной технологией коммутации пакетов, популярной в Северной Америке на протяжении 90-х годов.

Как было сказано, X.25 и Frame Relay представляют собой сквозные технологии глобальных сетей, поэтому, возможно, вам непонятно, почему они обсуждаются в главе, посвященной канальному уровню. Причина та же, что и для технологии ATM, — все упомянутые технологии сегодня часто применяются для передачи IP-дейтаграмм от одного IP-маршрутизатора к другому. Таким образом, с точки зрения протокола IP (также представляющего собой сквозную технологию глобальных сетей), X.25, Frame Relay и ATM — это технологии канального уровня. Поскольку протоколу IP в данной книге уделяется особое внимание, мы поместили описание технологий X.25, Frame Relay и ATM туда, где с точки зрения протокола IP (и большинства фанатиков Интернета) оно и должно располагаться, а именно в описание канального уровня.

Сейчас X.25-сети уже почти полностью исчезли. Они разрабатывались почти 20 лет назад для уровня технологий, в корне отличного от технологий сегодняшних кабельных сетей. Сети ретрансляции кадров были очень привлекательными для корпоративных клиентов, но теперь им приходится выдерживать жестокую конкуренцию со стороны «чистых» IP-решений. В самом деле, уже в середине первого десятилетия нового века технология Frame Relay может значительно уступить свои позиции. Но, несмотря на то что X.25-сети уже практически не применяются, а технология Frame Relay через несколько лет также может исчезнуть, мы решили обсудить их в этой книге в связи с их огромным историческим значением.

Исторический контекст

Набор протоколов X.25 был разработан в конце 70-х годов. Чтобы понять цели данной разработки, нам необходимо познакомиться с уровнем технологий этой «древней» эпохи. Хотя в то время персональный компьютер Apple II произвел фурор [523], персональные компьютеры и рабочие станции не были так широко распространены и не имели столь мощной сетевой поддержки, как теперь. Большая часть пользователей сидели за недорогими «неинтеллектуальными» терминалами, предоставляющими доступ по компьютерным сетям к удаленным мэйнфреймам. Эти терминалы обладали минимальными «умственными» способностями и способностями хранить данные (у них не было дисков); тем, что появлялось на

их экранах, полностью управлял мэйнфрейм на другом конце сети. Для поддержки неинтеллектуальных терминалов разработчики стандарта X.25 решили «наделить разумом» сеть. Как мы теперь знаем, эта философия диаметрально противоположна философии Интернета, в которой все сложные вопросы предлагается решать в оконечных системах, а к службам сетевого уровня предъявляются минимальные требования.

Один из способов наделения разумом X.25-сетей заключался в использовании виртуальных каналов. Как отмечалось в главе 1, для поддержки информации о состоянии сетям виртуальных каналов требуются коммутаторы пакетов. В частности, коммутатор должен содержать таблицу, позволяющую преобразовать номер виртуального канала входного интерфейса в номер виртуального канала выходного интерфейса. Кроме того, для установки и разрыва виртуальных каналов необходимы сложные сигнальные протоколы. Как было показано в главе 4, протокол IP не требует установки соединений и, таким образом, не использует виртуальные каналы. Когда узлу нужно отправить в сеть IP-дейтаграмму, он просто указывает в дейтаграмме адрес получателя и передает ее в сеть, не ожидая, что сеть установит виртуальный канал с получателем.

Другая важная часть технологического контекста конца 70-х и начала 80-х годов касается физических линий. В те дни почти все кабельные соединения представляли собой медные кабели с высоким уровнем шума и, соответственно, высокой вероятностью ошибок. Передача данных по оптоволоконным кабелям в те времена находилась в зачаточном состоянии. Вероятность появления ошибок в медных кабелях большой протяженности была на несколько порядков выше, чем в современных оптоволоконных. Эта высокая вероятность появления ошибок послужила причиной разработки протокола X.25, обеспечивающего восстановление после ошибок на отдельных ретрансляционных участках. В частности, когда X.25-коммутатор посылает пакет, он сохраняет в своем буфере копию посланного пакета до тех пор, пока следующий коммутатор, получив этот пакет, не ответит подтверждением. Таким образом, каждый коммутатор, получив пакет, выполняет проверку контрольной суммы и, если пакет не содержит ошибок, посылает подтверждение предыдущему коммутатору. Использование механизма исправления ошибок на каждом ретрансляционном участке значительно снижает скорость передачи данных в линии, но такой подход соответствовал уровню развития технологий той эпохи — высокому уровню ошибок в линиях и неинтеллектуальным терминалам. Технология X.25 также предполагала использовать механизмы управления потоком на уровне отдельных ретрансляционных участков. Протокол TCP, напротив, занимается исправлением ошибок и управлением потоком на сквозном уровне (то есть на уровне всего соединения между двумя приложениями на оконечных станциях) и, таким образом, не требует решать эти задачи на канальном уровне.

Сети ретрансляции кадров

Технология Frame Relay, разработанная в конце 80-х годов и получившая широкое применение в 90-х, во многом представляет собой второе поколение стандарта

X.25. Как и в стандарте X.25, в технологии Frame Relay используются виртуальные каналы. Однако, поскольку оптоволоконные системы 90-х годов обладали значительно более низкой вероятностью появления ошибок по сравнению с основанными на медных проводах системами 80-х, технология Frame Relay предназначалась для существенно более низкого уровня ошибок. В случае обнаружения ошибки в пакете коммутатор Frame Relay может только отбросить его. В результате удается получить сети с меньшими накладными расходами и более высокими скоростями передачи данных, чем в X.25-сетях, но при этом требуются более интеллектуальные оконечные системы для поддержания целостности передаваемых данных. В большинстве случаев сеть ретрансляции кадров принадлежит общественному Интернет-провайдеру (как, например, AT&T или British Telecom), а ее использование корпоративными клиентами оговаривается в договорах на несколько лет вперед. Сегодня технология Frame Relay позволяет локальным сетям различных корпораций обмениваться данными на довольно высоких скоростях. Как показано на рис. 5.50, сеть ретрансляции кадров часто соединяет эти локальные сети через IP-маршрутизаторы, при этом в каждой отдельной локальной сети устанавливается отдельный IP-маршрутизатор. Технология Frame Relay предоставляет альтернативу Интернету при необходимости обмениваться данными между отдельными локальными сетями внутри корпораций. Выбор может обусловливаться соображениями надежности и безопасности.

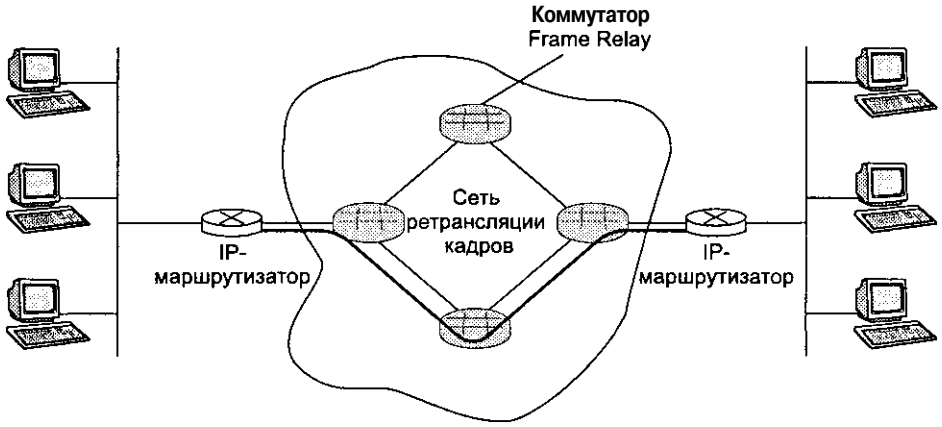


Рис. 5.50. Сеть ретрансляции кадров, соединяющая две Ethernet-сети

В сетях ретрансляции кадров могут использоваться как *коммутируемые* (Switched Virtual Circuit, SVC), так и *постоянные виртуальные каналы* (Permanent Virtual Circuit, PVC). Для соединения маршрутизаторов часто устанавливаются постоянные виртуальные каналы между каждой парой маршрутизаторов. Чтобы соединить JV маршрутизаторов, необходимо установить $N(N - 1)/2$ постоянных виртуальных каналов. В нашем последующем обсуждении мы будем предполагать, что в сети ретрансляции кадров используются постоянные виртуальные каналы (что является более общим случаем).

Передача IP-дейтаграмм между Ethernet-сетями через сеть ретрансляции кадров

Рассмотрим передачу IP-дейтаграммы между двумя Ethernet-сетями, соединенными сетью ретрансляции кадров на примере сети, изображенной на рис. 5.50. Когда Ethernet-кадр прибывает на маршрутизатор источника, сетевая Ethernet-карта маршрутизатора извлекает из него IP-дейтаграмму и передает ее сетевому уровню. Сетевой уровень передает IP-дейтаграмму сетевой карте Frame Relay. Эта карта помещает IP-дейтаграмму в кадр Frame Relay, как показано на рис. 5.51. Она также вычисляет контрольную сумму (2 байта) и помещает ее в поле CRC. Поле канального уровня (2 байта) включает 10-разрядное поле номера виртуального канала. Интерфейсная карта получает номер виртуального канала из таблицы соответствий номеров IP-сетей номерам виртуальных каналов. Затем интерфейсная карта передает пакет.

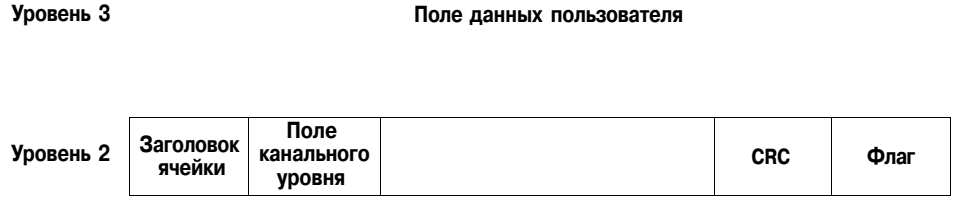


Рис. 5.51. Инкапсуляция данных пользователя (например, IP-дейтаграммы) в кадр Frame Relay

Интерфейсная карта передает пакет Frame Relay ближайшему коммутатору Frame Relay, которым владеет поставщик услуг сети ретрансляции кадров. Коммутатор изучает поле CRC. Если кадр содержит ошибку, коммутатор отбрасывает его (в отличие от X.25-сетей сети ретрансляции кадров не занимаются повторной передачей пакета на уровне ретрансляционных участков). В противном случае коммутатор использует номер виртуального канала в кадре для маршрутизации кадра к следующему коммутатору или к получающему маршрутизатору. Получающий маршрутизатор удаляет поля, относящиеся к сети ретрансляции кадров, а затем доставляет дейтаграмму по локальной Ethernet-сети получающему хосту. Если TCP-сегменты теряются или прибывают не в том порядке, тогда эту проблему в обменивающихся данными хостах решает протокол TCP.

Согласованная скорость передачи информации

В технологии Frame Relay используется новый механизм, называемый *согласованной скоростью передачи информации* (Committed Information Rate, CIR). Каждый виртуальный канал Frame Relay поддерживает этот механизм. Мы дадим более точное определение CIR далее, а пока можно считать, что согласованная скорость передачи информации представляет собой обязательство со стороны сети ретрансляции кадров предоставить виртуальному каналу указанную службой CIR скорость передачи данных. Служба CIR, появившаяся в сетях ретрансляции кадров

в начале 90-х годов, во многом явилась предтечей дифференцированных служб Интернета (см. главу 6). Как мы вскоре увидим, в сетях ретрансляции кадров служба CIR функционирует путем пометки пакетов.

В сетях ретрансляции кадров для пакетов используется два уровня приоритетов. Пометка пакетов осуществляется при помощи специального бита DE (Discard Eligibility — отмена пригодности) в заголовке пакета: 0 означает высокий уровень приоритета, 1 — низкий уровень. Сеть ретрансляции кадров старается доставить кадры с высоким приоритетом при любых обстоятельствах, но в то же время сеть может отбросить низкоприоритетный кадр в случае перегрузки. При особенно тяжелых условиях сеть может отбрасывать даже высокоприоритетные кадры. Перегрузка, как правило, измеряется состоянием выходных буферов в коммутаторах Frame Relay. Когда выходной буфер коммутатора Frame Relay близок к переполнению, коммутатор в первую очередь отбрасывает низкоприоритетные кадры, то есть те кадры в буфере, у которых бит DE установлен в 1.

Итак, теперь нам ясно, какие действия предпринимает коммутатор Frame Relay с маркированными пакетами, но мы еще ничего не сказали о том, как пакеты маркируются. Чтобы рассказать об этом, нам необходимо познакомиться с некоторыми терминами Frame Relay, что мы и сделаем с помощью рис. 5.50. *Скоростью доступа* называют скорость передачи на линии от маршрутизатора-источника до «крайнего» коммутатора Frame Relay. Как правило, эта скорость составляет 64 Кбит/с или больше (вплоть до 1,544 Мбит/с), но обязательно должна быть кратной 64 Кбит/с. Пусть скорость доступа равна R Кбит/с. Маркировкой пакетов, прибывающих от маршрутизатора-источника, занимается крайний коммутатор. Для этого коммутатор проверяет, укладываются ли значения времени прибытия кадров в фиксированный интервал, называемый *интервалом измерения* и обозначаемый символами T^c. Большинство поставщиков услуг в сетях ретрансляции кадров используют интервал измерения в диапазоне от 100 мс до 1 с.

Теперь мы можем описать скорость CIR более точно. Каждому виртуальному каналу, исходящему из маршрутизатора-источника (их может быть много, например они могут направляться в различные локальные сети), назначается *согласованная скорость передачи информации* (CIR), измеряемая в битах в секунду. Согласованная скорость передачи информации не может превышать скорость доступа R. Клиенты платят за определенный уровень CIR; чем выше CIR, тем больше клиент платит поставщику услуг в сети ретрансляции кадров. Если виртуальный канал генерирует пакеты с меньшей, чем CIR, скоростью, все пакеты этого виртуального канала помечаются как высокоприоритетные. Однако если скорость, с которой виртуальный канал генерирует пакеты, превышает CIR, тогда пакеты помечаются как низкоприоритетные. В частности, в течение каждого интервала измерения T^c для первых $CIR \times T^c$ бит, посланных виртуальным каналом, крайний коммутатор помечает соответствующие пакеты как высокоприоритетные ($DE = 0$). Все прочие пакеты, переданные за тот же временной интервал, коммутатор помечает как низкоприоритетные ($DE = 1$).

Рассмотрим работу крайнего коммутатора на примере. Пусть поставщик услуг в сети ретрансляции кадров использует интервал измерения $T^c = 500$ мс. Предполо-

жим, что скорость доступа к линии равна R = 64 Кбит/с, а скорость CIR, назначаемая некоему виртуальному каналу, равна 32 Кбит/с. Для простоты предположим, что каждый пакет Frame Relay состоит ровно из L = 4000 бит. Это означает, что каждые 500 мс виртуальный канал может посылать CIR x T^c/L = 4 высокоприоритетных пакета. Все дополнительные пакеты, переданные за тот же временной интервал, помечаются как низкоприоритетные. Обратите внимание, что за каждый интервал в 500 мс виртуальный канал может послать до четырех низкоприоритетных пакетов (сверх четырех высокоприоритетных пакетов). Поскольку задача сети ретрансляции кадров заключается в том, чтобы доставить все высокоприоритетные пакеты получающему узлу Frame Relay, виртуальный канал, по существу, гарантирует пропускную способность в 32 Кбит/с. В то же время сеть ретрансляции кадров не дает никаких гарантий насчет сквозных задержек как для высокоприоритетных, так и для низкоприоритетных пакетов.

При увеличении длительности интервала измерения T^c растет вероятность неравномерной доставки высокоприоритетных пакетов, переданных маршрутизатором-источником. В предыдущем примере, если T^c = 0,5 с, то маршрутизатор может передать до четырех высокоприоритетных пакетов подряд (без пауз между ними). При T^c = 1 с маршрутизатор может передать до восьми высокоприоритетных пакетов подряд. При меньших значениях интервала измерения T^c сглаживаются неравномерности потока высокоприоритетных пакетов, но большие значения интервала измерения увеличивают гибкость виртуального канала. В любом случае при любом значении Гс, средняя скорость передачи высокоприоритетных данных за большой интервал времени не может превысить значение CIR для данного виртуального канала.

Нам необходимо помнить, что от маршрутизатора-источника может исходить множество постоянных виртуальных каналов, проходящих через одну и ту же линию доступа. Интересно отметить, что сумма значений CIR для всех этих виртуальных каналов может превышать скорость доступа R. Это свойство называют *избыточным бронированием*. Таким образом, линия может передавать высокоприоритетные пакеты со скоростью, превышающей CIR (хотя каждый отдельный виртуальный канал передает пакеты со скоростью, не превышающей CIR).

Мы завершим этот раздел ссылкой на сайт форума Frame Relay, содержащий множество документов по данной теме (http://www.frforum.com). Превосходный ознакомительный курс, посвященный технологии Frame Relay, можно найти на web-сайте Hill Associates (http://www.hill.com). Кроме того, Уолтер Горальски написал книгу о технологии Frame Relay [186]. Эта книга написана понятным языком и в то же время достаточно подробна.

Резюме

В данной главе мы рассмотрели канальный уровень (уровень передачи данных) — его службы и принципы работы, а также ряд важных протоколов, использующих эти принципы.

Как было показано, основная задача канального уровня заключается в перемещении дейтаграммы сетевого уровня от одного узла (маршрутизатора или хоста)

к смежному узлу. Все канальные протоколы инкапсулируют дейтаграмму сетевого уровня в кадр канального уровня, который передают по линии связи смежному узлу. Кроме того, различные канальные протоколы предоставляют сильно различающиеся услуги по доступу к линии, доставке (надежность, обнаружение/исправление ошибок), управлению потоком и передаче (например, дуплекс или полудуплекс). Такие различия отчасти вызваны тем разнообразием типов линий связи, на которых должны работать канальные протоколы. На простой двухточечной линии присутствуют единственный передатчик и единственный приемник, обменивающиеся данными по одному «проводу». Линия коллективного доступа совместно используется многими отправителями и получателями. Соответственно, канальный протокол для канала коллективного доступа включает собственный протокол коллективного доступа к линии. В случае ATM, X.25 и Frame Relay мы видели, что «линия», связывающая два смежных (смежных в IP-смысле, то есть представляющих собой IP-маршрутизаторы следующего ретрансляционного участка в определенном направлении) узла, сама может быть *сетью*. В некотором смысле идея сети, рассматриваемой как «линия», не должна казаться странной. Например, телефонная «линия», соединяющая модем домашнего компьютера с модемом маршрутизатора, по сути, представляет собой путь через сложную и запутанную *сеть*.

Среди принципов, лежащих в основе обмена данными по каналу, мы рассмотрели методы обнаружения и исправления ошибок, протоколы коллективного доступа, адресацию канального уровня и построение расширенных локальных сетей при помощи хабов (концентраторов), мостов и коммутаторов. Мы познакомились с основами помехоустойчивого кодирования, то есть использования избыточной информации для обнаружения (а в некоторых случаях даже исправления) ошибок, заключающихся в искажении информации при передаче кадра по линии связи. Мы рассмотрели простые схемы проверки четности и подсчета контрольных сумм, а также более устойчивые методы проверки с помощью циклических избыточных кодов (CRC). Затем мы обсудили протоколы коллективного доступа. Мы выделили и изучили три основных метода координации доступа к широковещательному каналу: методы разделения канала (временного, частотного и кодового), методы произвольного доступа (протоколы ALOHA и CSMA), а также протоколы последовательного доступа (опрос и передача маркера). При наличии нескольких узлов, совместно использующих общий широковещательный канал, каждый узел должен иметь уникальный адрес на канальном уровне. Физические адреса в корне отличаются от адресов сетевого уровня, и для преобразования одних адресов в другие требуется специальный протокол (в случае Интернета это протокол ARP). Далее мы познакомились с тем, как узлы, совместно использующие общий широковещательный канал, образуют локальную сеть и как можно соединить несколько локальных сетей в локальные сети большего размера — и все это без маршрутизации на сетевом уровне.

Наконец, мы подробно обсудили ряд протоколов канального уровня: Ethernet, беспроводной протокол IEEE 802.11, а также протокол PPP. Как отмечалось в разделах «Технология ATM» и «Frame Relay», для соединения двух маршрутизаторов сетевого уровня могут также применяться технологии ATM, X.25 и Frame Relay.

Например, в случае работы протокола IP поверх ATM два смежных IP-маршрутизатора можно соединить друг с другом через ATM-сеть по виртуальному каналу. В таких ситуациях сеть, основанная на одной сетевой архитектуре (например, ATM или Frame Relay), может играть роль логической линии связи между двумя соседними узлами (например, IP-маршрутизаторами) в другой сетевой архитектуре.

Итак, на канальном уровне (уровне передачи данных) мы завершаем наше путешествие вниз по стеку протоколов! Разумеется, под канальным уровнем располагается физический уровень, но, вероятно, детали физического уровня лучше оставить для другого курса (например, теории связи, а не для курса компьютерных сетей). Тем не менее в этой главе мы затронули некоторые аспекты функционирования физического уровня (например, манчестерское кодирование в разделе «Ethernet», ослабление сигнала в разделе «Беспроводные каналы связи»), а также в главе 1 (обсуждение физических носителей в разделе «Доступ к сети и ее физическая среда»).

Итак, хотя наше путешествие вниз по стеку протоколов закончилось, изучение компьютерных сетей все еще продолжается. В следующих трех главах мы рассмотрим вопросы использования сетей для передачи мультимедиа, а также вопросы сетевой безопасности и управления сетью. Эти три темы не относятся к какому-либо одному уровню; в самом деле, в каждой из этих тем затрагиваются несколько уровней. Для понимания этих вопросов (в некоторых книгах по сетям называемых «специальными») требуется твердое знание основ всех уровней стека протоколов.

Вопросы и задания для самостоятельной работы

1. Если бы все линии связи Интернета обеспечивали надежную доставку, была бы в этом случае служба надежной доставки протокола TCP совершенно лишней? Почему да или почему нет?
2. Какие службы могут предоставляться протоколом канального уровня сетевому уровню? У каких из этих служб канального уровня есть соответствующие аналоги в протоколе IP? В TCP?
3. Предположим, в пакете содержатся данные с последовательностью битов 1010101010101011, и используется схема с битом четности. Каким будет значение поля контрольной суммы?
4. Предположим, два узла одновременно начинают передачу пакета длиной L по широковещательному каналу со скоростью R. Обозначим задержку распространения между двумя узлами как $^{\wedge}\mathrm{tp}$. Произойдет ли коллизия, если $t^{распр} < L/R$? Почему да или почему нет?
5. В разделе «Протоколы коллективного доступа» были перечислены четыре желательные характеристики широковещательного канала. Какими из этих характеристик обладает дискретная сеть ALOHA? Какими из этих характеристик обладают сети с передачей маркера?
6. Какие аналогии из вечеринки с коктейлями можно привести для протоколов опроса и передачи маркера?

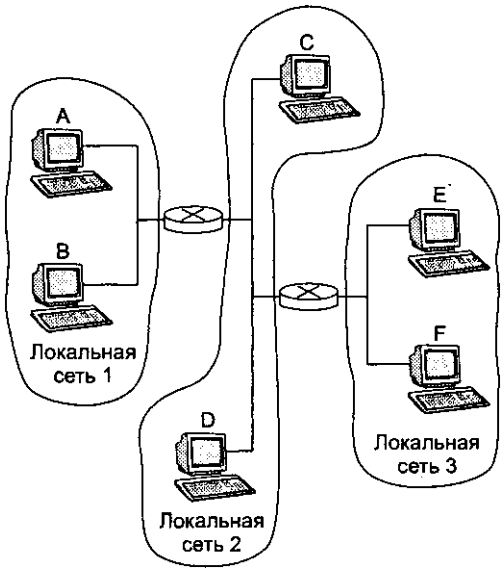
7. Почему протокол маркерного кольца неэффективен при очень большом периметре локальной сети?
8. Насколько велико адресное пространство локальной сети? Адресное пространство протокола IPv4? Адресное пространство протокола IPv6?
9. Пусть узлы A, B и C соединены с одной и той же широковещательной локальной сетью (с помощью адаптеров). Если узел A передает узлу B тысячи IP-дейтаграмм, упаковывая каждую дейтаграмму в кадр, направляемый по локальному адресу узла B, будет ли адаптер узла C обрабатывать эти кадры? Если да, будет ли адаптер узла C передавать эти кадры узлу C (то есть родительскому узлу адаптера)? Что изменится, если узел A будет передавать кадры по широковещательному адресу локальной сети?
10. Почему ARP-запрос посылается в широковещательном кадре? Почему ARP-ответ посылается в кадре с определенным LAN-адресом получателя?
11. У изображенного на рис. 5.20 маршрутизатора два ARP-модуля, у каждого из которых есть собственная ARP-таблица. Может ли один и тот же LAN-адрес присутствовать в обеих таблицах?
12. Сравните структуру кадров в стандартах lOBaseT, 100BaseT, 802.1 lb и Gigabit Ethernet. В чем они различаются?
13. Пусть 10-мегабитный адаптер передает в канал бесконечный поток единиц, используя манчестерское кодирование. Какова будет частота изменения сигнала, выходящего из адаптера?
14. Чему равна вероятность того, что узел выберет значение $K = 4$ после пятой коллизии? Какой задержке в секундах соответствует $K = 4$ в Ethernet-сети со скоростью передачи данных 10 Мбит/с?
15. В спецификации IEEE 802.11 длительность интервала SIFS должна быть меньше длительности интервала DIFS. Почему?
16. Предположим, что кадры RTS и CTS сети IEEE 802.11 были бы той же длины, что и стандартные кадры DATA и ACK. Даст ли в этом случае использование кадров RTS и CTS какое-либо преимущество? Почему?
17. Различает ли подуровень TC разные виртуальные каналы на передающей и принимающей сторонах?
18. Почему для передающего подуровня TC важно обеспечивать непрерывный поток ячеек, если подуровень PMD основан на ячейках, а не на кадрах?
19. Заполняет ли передающий подуровень TC какие-либо поля в заголовке ATM-ячейки? Какие?

Упражнения

1. Предположим, в пакете содержатся данные с последовательностью битов 1010101010101011 и используется схема с битом четности. Каким будет значение поля контрольной суммы при применении двухмерной схемы четности? Предполагается, что длина поля контрольной суммы минимальна.

2. Покажите (ваш пример должен отличаться от показанного на рис. 5.6), что двухмерная схема контроля четности способна обнаруживать и исправлять однобитовые ошибки. Приведите пример ошибки в двух битах, которую такая схема способна обнаружить, но не исправить.
3. Предположим, что в пакете *(D* на рис. 5.4) содержатся 10 байт, представляющие собой 8-разрядное двоичное представление целых чисел без знака от 0 до 9. Вычислите контрольную сумму для этих данных с помощью используемого в Интернете метода.
4. Рассмотрим 4-разрядный образующий многочлен G, показанный на рис. 5.8, и предположим, что поле данных *D* имеет значение 10101010. Каким будетзначение *R?*
5. Рассмотрим пример схемы CDMA с одним передатчиком на рис. 5.11. Каким будет выход передатчика (для двух показанных битов), если CDMA-код передатчика равен (1, - 1 , 1, - 1 , 1, - 1 , 1, -1)?
6. Рассмотрим второй передатчик на рис 5.12. Каким будет выход передатчика (прежде чем он сложится с сигналом первого передатчика) Zf_m *?*
7. Пусть приемнику на рис. 5.12 нужно получить данные, посланные вторым передатчиком. Покажите, что приемник действительно способен восстановить данные второго передатчика из суммарного канального сигнала при помощи кода второго передатчика.
8. В разделе «Протоколы коллективного доступа» мы в общих чертах обрисовали формулу эффективности дискретной системы ALOHA. В данной задаче мы завершим вывод формулы.
   8.1. Вспомним, что при наличии JVактивных узлов эффективность дискретной системы ALOHA равна $Np(t - p)^{N}$~*K* Найдите такое значение*p,* при котором данное выражение будет максимально.
   8.2. Используя значение;? из предыдущего упражнения, найдите эффективность дискретной системы ALOHA, устремив *NK* бесконечности. Вспомните, при значении *N,* стремящемся к бесконечности, $(1 - 1/N)^{N}$ стремится к 1/e.
9. Продемонстрируйте, что эффективность чистой системы ALOHA равна 1/(2e). Это упражнение легко выполнить, если вы выполнили предыдущее.
10. Постройте график зависимости эффективности дискретной и чистой систем ALOHA как функции отр при JV= 100.
11. Рассмотрим широковещательный канал с *N* узлами и скоростью передачи данных *R* бит/с. Предположим, что для коллективного доступа широковещательным каналом используется опрос (с дополнительным опрашивающим узлом). Предположим, что интервал времени с момента завершения передачи узлом до того, как следующий узел получит право передачи (то есть задержка опроса), равен $t^{опроса}$. Допустим, за период опроса узлу разрешается передать максимум Q бит. Чему равна максимальная пропускная способность широковещательного канала?

12. Рассмотрим три локальные сети, соединенные при помощи двух маршрутизаторов, как показано на схеме ниже.



12.1. Изобразите на схеме адаптеры.

12.2. Присвойте всем интерфейсам IP-адреса. Для локальной сети 1 используйте адреса вида 111.111.111.xxx; для локальной сети 2 — адреса вида 222.222.222.xxx; для локальной сети 3 — адреса вида 333.333.333.xxx.

12.3. Всем адаптерам назначьте LAN-адреса.

12.4. Рассмотрите передачу IP-дейтаграммы от хоста A хосту F. Предполагается, что все ARP-таблицы содержат всю необходимую информацию. Пронумеруйте этапы, как это было сделано для примера с одним маршрутизатором в подразделе «Протокол ARP» раздела «Адресация в локальных сетях и протокол ARP».

12.5. Вернитесь к предыдущему упражнению, предполагая, что ARP-таблица передающего хоста пуста (а остальные ARP-таблицы содержат всю необходимую информацию).

13. Как отмечалось при описании протокола CSMA/CD, после коллизии адаптер выжидает в течение ***Kx*** 512 длительностей передачи одного бита, где число ***K*** выбирается случайным образом. Сколько времени будет ждать адаптер при iC= 100, прежде чем перейти к шагу 2 алгоритма CSMA/CD (см. подраздел «Протокол CSMA/CD» в разделе «Ethernet») в Ethernet-сети со скоростью передачи данных 10 Мбит/с? 100 Мбит/с?

14. Предположим, узлы A и B подключены к одному и тому же сегменту 10-мегабитной Ethernet-сети, и время распространения сигнала между этими двумя узлами равно 225 длительностям передачи одного бита. Пусть узел A начинает передачу

кадра, а прежде, чем он закончит передачу, узел B также начинает передавать свой кадр. Может ли узел A закончить передачу прежде, чем он обнаружит передачу узла B? Почему да или почему нет? Если да, тогда узел A неверно полагает, что его кадр был успешно передан, избежав коллизии. (Чтобы было проще, пусть узел A начинает передачу кадра в момент времени $t = 0$. В худшем случае узел A передаст кадр минимального размера 512 + 64 бит. Поэтому узел A закончит передачу в момент времени $t^{mHeUi} = 512 + 64$ длительностей передачи одного бита. Таким образом, ответ будет отрицательным, если сигнал узла B достигнет узла A до момента времени t^{KOneUi}. Когда сигнал узла B достигнет узла A в худшем случае?)

15. Предположим, узлы A и B подключены к одному и тому же сегменту 10-мегабитной Ethernet-сети, и время распространения сигнала между этими двумя узлами равно 225 длительностям передачи одного бита. Пусть узлы A и B одновременно начинают передачу своих кадров, происходит коллизия, и узлы A и B выбирают различные значения параметра K для алгоритма CSMA/CD. Предполагая, что остальные узлы неактивны, может ли повторно произойти коллизия при передаче кадров узлами A и B? Для выполнения данного упражнения нам достаточно рассмотреть следующий пример. Пусть узлы A и B начинают передачу своих кадров в момент времени $t = 0$. Они обнаруживают коллизию в момент времени $t = 225$ длительностей передачи одного бита. Оба узла заканчивают передачу сигнала коллизии в момент времени $t = 225 + 48 = 273$ длительности передачи одного бита. Пусть $K^A = 0$, а $K^B = 1$. На какой момент времени планирует свою повторную передачу узел B? В какой момент времени начнет передачу узел A? (В соответствии с протоколом после возвращения к шагу 2 узлы должны ждать, пока канал освободится — см. подраздел «Протокол CSMA/CD» в разделе «Ethernet».) В какой момент времени сигнал узла A достигнет узла B? Воздержится ли узел B от передачи в запланированное время?

16. Рассмотрим Ethernet-сеть 100BaseT. Каким должно быть максимальное расстояние между узлом и хабом для эффективности 0,50? Предполагается, что длина кадра составляет 64 байт, а повторители в сети отсутствуют. Гарантирует ли также данное максимальное расстояние обнаружение передачи, ведущейся другим узлом во время собственной передачи? Почему да или почему нет? Как это максимальное расстояние соотносится с фактическим стандартом для сетей 100BaseT?

17. В этом упражнении вам предлагается вывести формулу эффективности протокола коллективного доступа, схожего с протоколом CSMA/CD. В этом протоколе время дискретное и все адаптеры работают синхронно. Однако в отличие от дискретной системы ALOHA длительность элементарного интервала времени, называемого «слотом», много меньше времени передачи одного кадра. Пусть длительность слота равна 5. Предположим, что все кадры имеют постоянную длину $L = kRS^y$ где R представляет собой скорость передачи данных в канале, а k — большое целое число. Пусть в сети N узлов, у каждого из которых есть бесконечное количество кадров для передачи. Также предполагается, что время распространения сигнала $t^{pacnp} < 5$, так что все узлы могут обнаружить коллизию прежде, чем закончится очередной слот. Протокол работает следующим образом.

- Если в течение данного слота ни один из узлов не владеет каналом, все узлы состязаются за канал; то есть в момент начала данного слота каждый узел передает кадр с вероятностью p. Если передачу начинает ровно один узел, этот узел захватывает канал на последующие k - 1 слотов и передает кадр целиком.
- Если какой-либо узел владеет каналом, все остальные узлы воздерживаются от передачи до тех пор, пока данный узел не закончит передачу кадра. Как только узел заканчивает передачу кадра, все узлы начинают борьбу за канал.

Обратите внимание, что канал попеременно находится в одном из двух состояний: в «продуктивном состоянии», длящемся ровно k слотов, и в «непродуктивном состоянии», длительность которого случайна. Очевидно, что коэффициент использования канала равен $k/(k + x)$, где x — ожидаемое количество последовательных непродуктивных слотов.

17.1. Определите эффективность этого протокола для фиксированных значений N и p.

17.2. Для фиксированного значения N определите значение p, при котором эффективность является максимальной.

17.3. Используя значение p (являющееся функцией N), полученное в предыдущем упражнении, определите эффективность при значении N, стремящемся к бесконечности.

17.4. Покажите, что эффективность приближается к 1 при увеличении размера пакета.

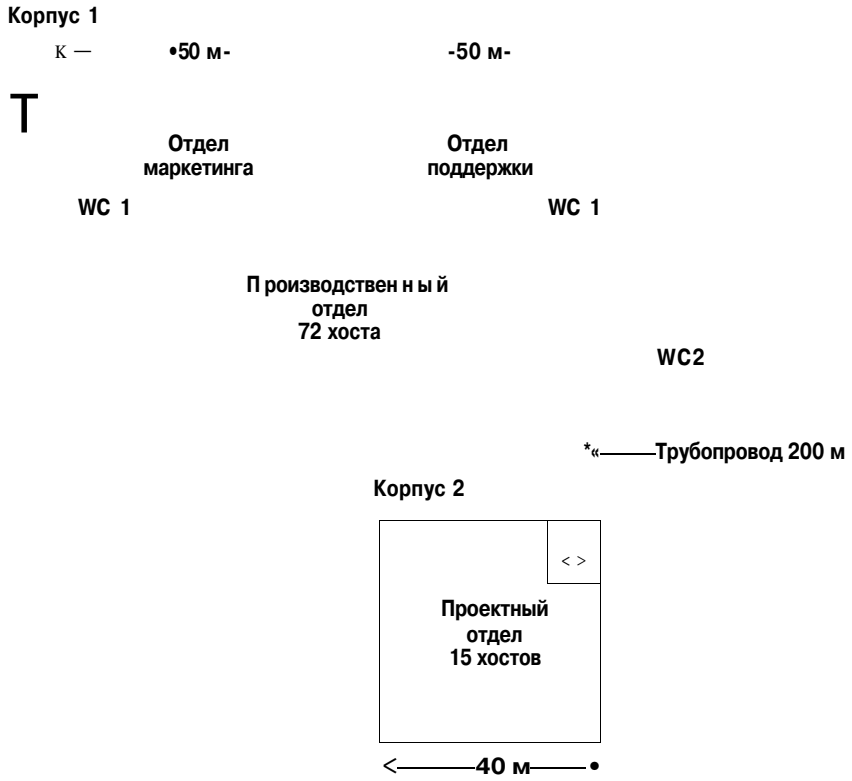
18. Пусть два узла, A и B, присоединены к противоположным концам 900-метрового кабеля и каждому из них необходимо послать друг другу по одному кадру в 1000 бит (включая заголовки и преамбулы). Оба узла пытаются выполнить передачу в момент времени $t = 0$. Предположим, что между узлами A и B располагаются четыре повторителя, каждый из которых вносит 20-разрядную задержку. Пусть скорость передачи данных равна 10 Мбит/с и используется протокол CSMA/CD с интервалами отката, кратными 512 длительностей передачи одного бита. После первой коллизии узел A выбирает $K = 0$, а узел B — $K = 1$. Игнорируйте сигнал коллизии.

18.1. Чему равна задержка распространения сигнала в одну сторону (включая задержки в повторителях) между узлом A и узлом B в секундах? Предполагается, что скорость распространения сигнала составляет 2×10^8 м/с.

18.2. В какой момент времени (в секундах) пакет узла A будет полностью доставлен узлу B?

18.3. Предположим, что пакет для передачи есть только у узла A, а повторители заменены мостами. Предположим также, что на каждом мосту помимо задержки промежуточного хранения имеет место задержка обработки, равная 20 интервалам передачи одного бита. За какое время (в секундах) пакет узла A будет доставлен узлу B?

19. Вам предстоит спроектировать локальную сеть для организации, располагающейся в двух зданиях (схема показана на рисунке ниже).



Условные обозначения:

WC — коммутационный шкаф

Вы можете использовать следующее оборудование:

- тонкий коаксиальный кабель (1 доллар за метр);
- неэкранированная витая пара (1 доллар за метр);
- пара оптоволоконных кабелей (2 доллара за метр);
- сетевой адаптер с портом для тонкого коаксиального кабеля (70 долларов);
- сетевой адаптер с UTP-портом (70 долларов);
- двухпортовый повторитель (800 долларов);
- многопортовый повторитель (8 портов для тонкого коаксиального кабеля — 1500 долларов);
- многопортовый повторитель (6 оптических портов — 2000 долларов);
- двухпортовый мост (любая комбинация портов для коаксиальных кабелей, неэкранированных витых пар и оптоволоконных кабелей — 2200 долларов);
- хаб на 36 портов для неэкранированных витых пар (4000 долларов);

- хаб на 6 оптических портов и 24 порта для неэкранированных витых пар (6000 долларов);
- файловый сервер на основе процессора Pentium с сетевой операционной системой (максимум на 30 пользователей — 9000 долларов).

Проект должен удовлетворять следующим требованиям:

- каждый отдел компании должен иметь доступ ко всем ресурсам всех остальных отделов;
- трафик, создаваемый сотрудниками одного отдела, не должен влиять на локальные сети других отделов, кроме случаев обращения к ресурсам локальных сетей других отделов;
- файловый сервер может поддерживать только 30 пользователей;
- файловые серверы не могут совместно использоваться несколькими отделами;
- все повторители, мосты и хабы должны располагаться в коммутационных шкафах.

19.1. Вы должны использовать тонкий коаксиальный кабель (не витую пару) и, в случае необходимости, волоконную оптику. Составьте схему вашего проекта. Также составьте список используемого оборудования (с указанием численных параметров) и определите общую стоимость локальной сети.

19.2. Повторите предыдущее упражнение, но для витой пары и, если необходимо, волоконной оптики.

20. Пусть виртуальный канал Frame Relay генерирует пакеты фиксированной длины L. Пусть JR, T^c и CIR обозначают скорость доступа, интервал измерения и согласованную скорость передачи информации соответственно.

20.1. Определите в виде функции этих переменных количество высокоприоритетных пакетов, которые данный виртуальный канал может послать за время измерения.

20.2. Определите в виде функции этих переменных количество низкоприоритетных пакетов, которые данный виртуальный канал может послать за время измерения.

Для предыдущего упражнения считайте, что в течение каждого интервала измерения виртуальный канал сначала генерирует максимальное разрешенное количество высокоприоритетных пакетов и только затем начинает генерировать низкоприоритетные пакеты.

21. Пусть Ethernet-сеть, являющаяся источником на схеме, изображенной на рис. 5.50, содержит web-сервер, загруженный обработкой запросов, поступающих от клиентов, находящихся в другой Ethernet-сети. Каждый HTTP-запрос переносится в одной или нескольких IP-дейтаграммах. Когда IP-дейтаграмма прибывает на интерфейс Frame Relay, она инкапсулируется в кадр Frame Relay. Предположим, что каждый web-объект состоит из 5 бит, а размер каждого пакета Frame Relay равен I. Пусть web-сервер начинает обслуживание одного объекта в начале каждого интервала измерения. Игнорируя для пакетов все накладные

расходы на прикладном, транспортном, сетевом (IP) и канальном (Frame Relay) уровнях, определите максимальный размер web-объекта (в виде функции параметров T^c, CIR и I), при котором каждый объект целиком переносится высокоприоритетными пакетами Frame Relay.

22. Как отмечалось в этой главе, в ATM-сетях используются 53-байтовые пакеты, состоящие из заголовка в 5 байт и полезной нагрузки в 48 байт. 53 байт — необычно мало для пакета фиксированного размера. В большинстве сетевых протоколов (IP, Ethernet, Frame Relay и т. д.) используются пакеты, средние размеры которых значительно больше. Один из недостатков пакетов небольшого размера состоит в том, что значительная доля пропускной способности канала занимается избыточными байтами. В случае ATM почти 10 % пропускной способности «тратится» на ATM-заголовок. В этом упражнении мы попробуем разобраться, почему был выбран такой маленький размер пакета. Предположим, что ATM-ячейка состоит из P байт (не обязательно 48) и 5 байт заголовка.

    22.1. Рассмотрим передачу оцифрованного голосового сигнала напрямую по ATM-сети. Предположим, источник генерирует сигнал с постоянной скоростью 64 Кбит/с. Пусть каждая передаваемая ячейка заполняется данными полностью. Время, необходимое для заполнения ячейки, представляет собой *задержку пакетирования*. Определите задержку пакетирования (в миллисекундах) как функцию от размера пакета L.

    22.2. Задержка пакетирования, превышающая 20 мс, может вызвать заметное и неприятное эхо. Определите задержку пакетирования для размера пакета L = 1,500 байт (что грубо соответствует максимальному размеру пакета в Ethernet) и для L = 48 байт (размер ATM-ячейки).

    22.3. Вычислите задержку промежуточного хранения на одном ATM-коммутаторе для канала с пропускной способностью R = 155 Мбит/с (популярная в ATM-сетях скорость передачи данных), размером пакета L = 1,500 байт и L = 48 байт.

    22.4. Прокомментируйте преимущества от использования ячеек малого размера.

Дополнительные вопросы и задания

Вам предлагается «побродить» по web и ответить на перечисленные ниже вопросы.

1. Каков приблизительный диапазон цен на адаптеры 10/100 Мбит/с? На адаптеры Gigabit Ethernet? Адаптеры 802.11b? Сравните эти цены с ценами телефонных модемов 56 Кбит/с и ценами ADSL-модемов.
2. Цена хаба или коммутатора часто зависит от количества интерфейсов (в терминах локальных сетей также называемых портами). Каков приблизительный диапазон цен одного интерфейса для хаба на 10 Мбит/с? Для хаба на 100 Мбит/с? Для коммутатора, состоящего только из 10-мегабитных интерфейсов? Для коммутатора, состоящего только из 100-мегабитных интерфейсов?
3. Предположим, вам нужно построить домашнюю сеть, включающую маршрутизатор с выходом в Интернет, беспроводную Ethernet-сеть и пять беспроводных

Ethernet-адаптеров. Определите, какие устройства вам необходимы для этого, и сосчитайте их суммарную стоимость.

4. Перечислите пять продуктов на сегодняшнем рынке, предоставляющих интерфейс Bluetooth.
5. Многие функции адаптера могут выполняться программно центральным процессором узла. Каковы преимущества и недостатки перемещения этих функций от адаптера к узлу?
6. Найдите номера протоколов, применяемых в Ethernet-кадре для IP-дейтаграммы и для ARP-пакета.

Интервью

Роберт М. Меткалф является директором и вице-президентом группы Technology of the International Data Group в Бостоне. Он также ведет еженедельную колонку в Info World. Во время работы в исследовательском центре Palo Alto в 1973 году он разработал идею стандарта локальной Ethernet-сети. В 1979 году он основал корпорацию 3Com и глобальную компанию Fortune-500. Роберт М. Меткалф получил степень бакалавра в области электротехники и управления в Массачусетском технологическом институте и степени магистра наук и доктора философии (в области прикладной математики и технической кибернетики соответственно) в Гарварде.

- Почему вы решили специализироваться в области компьютерных сетей?

  В 1969 году я был новоиспеченным аспирантом в области технической кибернетики, учился в Гарварде и собирался вскоре стать одним из сотрудников исследовательской лаборатории Массачусетского технологического института. В то время главным спонсором исследований в компьютерной области было агентство ARPA (Advanced Research Projects Agency — агентство перспективного планирования исследовательских работ). Новым интересным проектом, финансировавшимся агентством ARPA, была компьютерная сеть ARPAnet, представлявшая собой первую версию Интернета. Итак, я стал работать над проектом ARPAnet. Я разрабатывал аппаратное и программное обеспечение, а также протоколы. Все это привело к созданию Ethernet, Интернета и компании 3Com.

- Какова была ваша первая работа в компьютерной отрасли?

  В 1965 году, начиная свой второй год обучения в Массачусетском технологическом институте, в лабораториях Рэйтиона Уэйлэнда я начал работать над тем, что стало моим настоящим призванием. Первой программой, за которую мне заплатили, была подпрограмма, преобразующая 5-разрядные символы Бодо в 6-разрядные компьютерные коды. Я был занят полную рабочую смену в Массачусетском технологическом институте, как правило, с полуночи до 8 утра. Я не пропускал занятий. Я участвовал в спортивных университетских состязаниях. Я был социальным председателем студенческого братства. Когда же я спал ?

- На что похож ваш обычный день?

  Обычных дней не бывает. Я провожу очень много времени за чтением, в разговорах по телефону, за своим компьютером Macintosh, отправляя электронную почту, а также плутая по Интернету. Кроме того, я путешествую и просиживаю в конференц-залах либо слушая презентации, либо сам их проводя. В последнее время я предпочитаю больше времени проводить с семьей в моем доме в штате Мэн.

- Что самое сложное в вашей работе?

  Я специалист в области технологий. Я пишу статьи, читаю лекции и организую конференции. Самое сложное при этом — определить, кто врет.

- Кто помог вам достичь успеха в вашей профессиональной деятельности?

  Дж. С. Р. Ликлидер, Майкл Дертузос, Никлас Негропонтэ и Пол Грэй из Массачусетского технологического института (MIT). Боб Нойс из корпорации Intel. Стив Джобе из компании Apple. Ларри Эллисон из Oracle. Корпорация Hewlett & Packard. Том Уотсон из корпорации IBM. Грейс Мюррэй Хоппер из военно-морского флота США. Билл Гейтс (Microsoft). БатлерЛампсон (Xerox Pare, Digital, Microsoft). Алан Кэй (Xerox, Disney). Дэвид Лиддл (Xerox, Metaphor, Interval). Гордон Белл (DEC, NSF, Microsoft). Компании Bell, Edison, Shockley и многие другие.

- Какое влияние оказала технология на процесс обучения? Какое влияние, по вашему мнению, она окажет в будущем?

  Вместо того чтобы так беспокоиться об использовании Интернета в школах, что, в общем-то, неплохо, нам следует больше думать об использовании Интернета вместо школ.

- Вы преподавали в университете Стэнфорда в течение восьми лет. В чем заключается особенность вашего метода преподавания технологий?

  Преподавание и обучение неразделимы. Вот почему наши исследовательские университеты, такие как Массачусетский технологический институт и университет Стэнфорда, работают столь успешно.

- Расскажите немного о ваших салонах в Бостоне. Какую цель они преследуют? Чего удалось достичь с их помощью?

  Мои салоны в Бостоне предназначены, главным образом, для людей, которых я люблю, — предпринимателей, а также тех, кто их окружает и поддерживает. Я пытаюсь внести свой вклад в дело поддержки предпринимательства в Бостоне. Салоны имеют успех, так как я тщательно отбираю тех, кого приглашаю, — посетители рады встрече друг с другом. Кроме того, я не поощряю речи. Только сплетни.

ГЛАВА 6 Мультимедиа в компьютерных сетях

Итак, мы закончили наше путешествие вниз по стеку протоколов и теперь знакомы с основами теории и практики компьютерных сетей. Знание этих основ поможет нам в этой главе, в которой мы обсудим сетевые мультимедийные приложения, работа которых затрагивает сразу несколько уровней.

В последние несколько лет мы стали свидетелями экспоненциального роста в области развития и развертывания сетевых приложений, передающих и получающих аудио- и видеоданные по Интернету. Новые сетевые мультимедийные приложения (также называемые приложениями с непрерывными потоками данных) — видео, IP-телефония, Интернет-радио, телеконференции, интерактивные игры, виртуальные миры, дистанционное обучение и многое другое — появляются, похоже, чуть ли не каждый день. Требования к сетевым службам у этих приложений существенно отличаются от требований эластичных приложений (таких как электронная почта, web, удаленный терминал, совместное использование файлов), которые рассматривались в главе 2. В частности, многие мультимедийные приложения очень чувствительны к длительности сквозной задержки, а также к изменению задержки, но допускают потери некоторого количества данных. В первой половине этой главы мы обсудим, как должны быть устроены мультимедийные приложения, чтобы максимально использовать предоставляемое Интернетом обслуживание по остаточному принципу, не дающее никаких гарантий относительно величины сквозной задержки. Во второй половине главы мы познакомимся с теми действиями по расширению архитектуры Интернета, которые предпринимаются сегодня, чтобы обеспечить явную поддержку служб, требующихся мультимедийным приложениям.

Сетевые мультимедийные приложения

В главе 2, обсуждая требования приложений к сетевым службам, мы выделили несколько показателей, по которым можно классифицировать эти требования. Два из этих показателей — некоторые аспекты синхронности и устойчивость к потере дан-

ных — особенно важны для сетевых мультимедийных приложений. Аспекты, относящиеся к синхронности, важны, потому что многие мультимедийные приложения очень *чувствительны к задержке*. Мы вскоре увидим, что в таких приложениях пакеты, запаздывающие более чем на несколько сот миллисекунд, практически бесполезны. Вместе с тем сетевые мультимедийные приложения, по большей части, *допускают потерю части данных* — эти случайные потери пакетов вызывают незначительные сбои при воспроизведении аудио- и видеоданных и часто могут быть частично или полностью замаскированы. Эти характеристики мультимедийных приложений отчетливо контрастируют с характеристиками эластичных приложений, таких как web, электронная почта, FTP и Telnet. Для эластичных приложений долгие задержки не наносят существенного ущерба, они неприятны только пользователю, зато исключительную важность представляют полнота и целостность передаваемых данных.

Примеры мультимедийных приложений

Через Интернет передают свои данные множество мультимедийных приложений. В этом подразделе мы рассмотрим три класса мультимедийных приложений: записанное потоковое аудио и видео, потоковое аудио и видео реального времени, а также интерактивное аудио и видео реального времени.

Мы не будем рассматривать приложения, сначала загружающие мультимедийные данные (например, MP3-файл) из сети, а уже затем воспроизводящие их. Это обычные эластичные приложения, выполняющие передачу файлов и не предъявляющие особых требований к задержкам. Системы передачи (HTTP и TCP) и одноранговые системы совместного использования файлов обсуждались в главе 2.

Записанное потоковое аудио и видео

К этому классу приложений относятся приложения, в которых клиент отправляет запрос на просмотр хранящегося на сервере аудио- или видеофайла, который передается ему, как правило, в сжатом виде и тут же воспроизводится. В этих файлах могут содержаться, например, аудиозаписи и видеозаписи лекций (вы можете посетить web-сайт этой книги и попробовать прослушать такую лекцию!), музыка, архивы знаменитых радиопередач, исторические записи, полнометражные фильмы, записи телевизионных шоу, документальные фильмы, видеоархивы исторических событий, мультфильмы или музыкальные видеоклипы. Этот класс приложений обладает тремя ключевыми характеристиками.

- *Сохранение мультимедийных данных на запоминающем устройстве.* Аудио- или видеоданные заранее записываются и сохраняются на сервере в виде файлов. В результате пользователь может остановить, перемотать фильм вперед или назад, а также начать воспроизведение с начала любой части фильма. Время отклика системы на подобные команды пользователя не должно превышать десяти секунд.
- *Потоковое воспроизведение.* В приложениях записанного потокового аудио/видео клиент может воспроизводить аудио- и видеоданные уже через несколько секунд после того, как данные начнут поступать с сервера. Это означает, что клиент воспроизводит одну часть файла, в то же время принимая по сети следующие его части. Такой метод воспроизведения, называемый потоковым, по-

зволяет не ждать загрузки всего файла (для чего может потребоваться довольно много времени) перед его воспроизведением. Существует множество потоковых мультимедийных продуктов, включая RealPlayer компании RealNetworks **(http://www.realnetworks.com)**, QuickTime компании Apple **(http://www.apple.com/quicktime)** и Windows Media Player производства корпорации Microsoft **(http://www.microsoft.com/windows/windowsmedia/)**.

- *Непрерывное воспроизведение.* Начавшись, воспроизведение мультимедиа должно продолжаться столько времени, сколько длится оригинальная запись. Это требование накладывает строгие ограничения на значение задержки в доставке данных. Данные с сервера должны доставляться вовремя. Хотя приложения записанного потокового мультимедиа предъявляют довольно высокие требования к службе доставки данных, накладываемые ими ограничения на сквозную задержку не столь строги, как для интерактивных приложений реального времени, например Интернет-телефонии или видеоконференций (см. далее).

ИСТОРИЧЕСКАЯ СПРАВКА

Компания RealNetworks, пионер в области аудио- и видеопродуктов, первой занялась вопросом распространения аудиоданных в Интернете. Эта компания появилась под именем Progressive Networks в 1995 году. Ее первый продукт — система RealAudio — включал аудиокодировщик, аудиосервер и аудиопроигрыватель. Система RealAudio позволяла клиентам искать, выбирать и воспроизводить заказанные в сети аудиоданные. Пользоваться этой системой было также легко, как стандартным кассетным магнитофоном. Система RealAudio быстро стала популярной у поставщиков развлекательных и информационных передач в сети. В начале 1997 года компания RealNetworks расширила линейку своих продуктов, включив также и видеопроигрыватели. В настоящее время продукты компании RealNetworks работают по протоколам RTP и RTSP.

Последние несколько лет компании RealNetworks сильную конкуренцию стала составлять корпорация Microsoft (владеющая частью акций RealNetworks). В 1997 году корпорация Microsoft выпустила на рынок собственные мультимедийные потоковые продукты, тем самым создав почву для новой войны медиа-проигрывателей, аналогичной войне браузеров между компаниями Netscape и Microsoft. Интересно отметить, что компании RealNetworks и Microsoft используют для своих мультимедийных проигрывателей несовместимые форматы данных. Таким образом, это не только война проигрывателей, но и борьба форматов за право стать стандартом в Интернете.

Потоковое аудио и видео реального времени

Этот класс приложений схож с традиционной трансляцией радио- и телепередач с той лишь разницей, что передача ведется не в эфире и не по специальному кабелю, а через Интернет. Эти приложения позволяют пользователю получать теле- или радиопрограммы из любого уголка мира. (Например, один из авторов этой книги часто слушает свои любимые Филадельфийские радиостанции у себя дома во Франции. Второй автор, живя в течение года во Франции, регулярно слушал прямые репортажи о матчах своей любимой университетской баскетбольной команды.)

Поскольку «живое» потоковое аудио и видео не сохраняется на запоминающих устройствах, клиент не может осуществлять перемотку вперед. Однако некоторые приложения позволяют пользователю локально сохранять полученные данные и

выполнять такие действия, как остановка и перемотка назад. Довольно часто у подобных радио- или телепередач бывает весьма широкая аудитория. Доставка данных сразу многим клиентам, принимающим в этот момент передачу одной и той же станции, может эффективно осуществляться путем групповой IP-маршрутизации (см. раздел «Групповая маршрутизация» в главе 4). Однако на момент написания этой книги доставка мультимедийных данных чаще выполняется путем выборочной рассылки нескольких отдельных потоков. Как и в случае записанного потокового мультимедиа, здесь требуется непрерывное воспроизведение, хотя временные ограничения не столь строги, как для интерактивных приложений реального времени. Допустимыми считаются задержки в десятки секунд от момента запроса пользователя до начала воспроизведения.

ИСТОРИЧЕСКАЯ СПРАВКА

Учитывая всемирную распространенность телефонов, уже в конце 80-х годов многие мечтатели предсказывали, что следующим популярным Интернет-приложением будет какая-нибудь разновидность голосовой связи. Эти предсказания сопровождались исследованиями в области Интернет-телефонии и работами над созданием соответствующего программного обеспечения. Так, например, исследователи разработали прототипы Интернет-телефона в 80-е годы, за годы до того, как стала популярной Всемирная паутина. На протяжении всех 80-х годов многие компании-первопроходцы создавали программные продукты, обеспечивающие телефонную связь между двумя персональными компьютерами по Интернету. Однако ни одна из этих программ не стала популярной у большинства пользователей Интернета (несмотря на то что некоторые из них были совместимы с популярными браузерами). И только в 1999 году популярность голосовой связи через Интернет стала расти.

Фактически массовое использование телефонных приложений для персональных компьютеров началось в конце 90-х годов. Эти приложения позволяют Интернет-пользователю, к компьютеру которого подключен микрофон, задействовать свое Интернет-соединение как обычный телефон и звонить на другие обычные телефоны. Двумя компаниями, Net2Phone (http://www.net2phone.com/) и Dialpad (http://www.dialpad. com), была разработана служба телефонной связи через Интернет. С помощью подобных служб пользователь получает возможность звонить по телефону через Интернет со своего персонального компьютера, что для междугородной и международной связи оказывается в несколько раз дешевле, чем по обычному телефону, поэтому они быстро завоевали популярность среди публики, любящей поговорить, но ограниченной в средствах. Кроме того, существует несколько компаний, предоставляющих подобные услуги предприятиям. Как будет показано далее, при использовании данного метода оцифрованные голосовые данные направляются через Интернет от персонального компьютера к шлюзу, а по телефонным сетям с коммутацией каналов от шлюза до телефона (стационарного или мобильного). Для связи по Интернету между персональным компьютером и шлюзом часто используются несколько популярных протоколов, включая RTP, SIP и H.323. Все они обсуждаются в этой главе. Второй класс приложений составляют программы, позволяющие пользователям с помощью обычных телефонов звонить по межгороду через Интернет по более низким тарифам, чем при традиционной телефонной связи. Именно такая связь используется при звонках по различным телефонным картам. Третий класс приложений составляют программы, обеспечивающие асинхронную передачу голосовых данных через Интернет (http://www.wimba.com). Помимо служб, многие производители сетевого оборудования, включая Cisco, Lucent и Alcatel, предоставляют телефонным операторам и деловым клиентам законченные линейки продуктов для передачи голосовых данных по IP-сетям (http://www.von.com).

Интерактивное аудио и видео реального времени

Этот класс приложений дает возможность пользователям общаться друг с другом в режиме реального времени. Службу интерактивного аудио реального времени с передачей данных через Интернет часто называют *Интернет-телефонией,* поскольку с точки зрения пользователя она напоминает традиционную телефонную службу с коммутацией каналов. Интернет-телефония может использоваться для локальной и междугородной телефонной связи по очень низкой цене. Кроме того, Интернет-телефония позволяет упростить развертывание новых служб, которые плохо поддерживаются традиционными сетями с коммутацией каналов, таких как службы интеграции телефонии в web, видеоконференции, службы каталогов, службы фильтрации абонентов и т. д. На сегодняшний день созданы сотни программ поддержки Интернет-телефонии (http://www.von.com). Например, пользователи программы Instant Messenger компании Microsoft могут звонить с персонального компьютера на обычный телефон или с одного персонального компьютера на другой. Интерактивная видеосвязь в реальном времени (видеоконференции) позволяет пользователям не только слышать, но и видеть друг друга. Сегодня на рынке предлагается множество программных продуктов, обеспечивающих интерактивную видеосвязь через Интернет в реальном времени, включая программу NetMeeting корпорации Microsoft. Обратите внимание, что в интерактивных аудио- и видеоприложениях реального времени пользователь может двигаться и говорить. Для подобных приложений задержка в доставке данных не должна превышать нескольких десятых долей секунды. При передаче голоса задержки менее 150 мс не воспринимаются слушателем, задержки в пределах от 150 мс до 400 мс считаются приемлемыми, а задержки, превышающие 400 мс, могут восприниматься как существенные искажения и вести к неразборчивости речи.

Проблемы мультимедиа в сегодняшнем Интернете

Вспомним, что развернутый сегодня в Интернете протокол IP предоставляет всем переносимым им дейтаграммам *обслуживание по остаточному принципу*. Другими словами, Интернет прилагает максимум усилий по перемещению каждой дейтаграммы от отправителя к получателю за минимальное время, но не предоставляет никаких гарантий относительно величины сквозной задержки для отдельных пакетов. Также никаких гарантий не предоставляется относительно изменений величины задержки доставки пакета в потоке пакетов. Поскольку протоколы TCP и UDP работают поверх протокола IP, эти протоколы также не могут предоставить использующим их приложениям каких бы то ни было гарантий относительно задержки. Поскольку в Интернете нет службы, предпринимающей специальные усилия по доставке пакетов в жесткие сроки, разработка успешно работающих мультимедийных сетевых приложений для Интернета представляет собой крайне сложную задачу. На сегодняшний день мультимедийные приложения в Интернете достигли значительного, но ограниченного успеха. Так, в приложениях для воспроизведения записанного потокового аудио и видео задержка, как правило, может составлять от 5 до 10 с. Однако в периоды пиковой нагрузки производительность таких приложений может стать неудовлетворительной, особенно если линии связи, по которым

передаются мультимедийные данные (например, кабели, проложенные по дну океанов), оказываются перегруженными.

Успех интерактивных мультимедийных приложений в Интернете, таких как Интернет-телефония и видеоконференции, оказался не столь большим, как у записанного потокового видео. Разумеется, интерактивные аудио- и видеоприложения налагают более строгие ограничения на задержку доставки пакетов и на джиттер. *Джиттером* называют изменение задержки в доставке отдельных пакетов из одного потока. Аудио- и видеоприложения реального времени могут хорошо работать в сетях, в которых достаточно ресурсов пропускной способности, а следовательно, задержка и джиттер минимальны. Но как только поток пакетов аудио- или видеоданных реального времени попадает в умеренно загруженную линию, качество может снизиться до неприемлемого.

Устройство мультимедийных приложений, без сомнения, было бы более простым, если бы в Интернете существовали службы первого и второго класса, причем количество пакетов первого класса было бы ограничено. Такие пакеты первого класса получали бы приоритетное обслуживание в очередях маршрутизатора. Подобного «первоклассного» обслуживания было бы достаточно для чувствительных к задержке приложений. Однако на сегодняшний день в Интернете применяется равноправный подход к планированию очередей на маршрутизаторах. Все пакеты получают равное обслуживание. Никакие пакеты, включая чувствительные к задержке аудио- и видеопакеты, не получают специальных приоритетов в очередях маршрутизаторов. Независимо от того, сколько у вас денег, вы должны встать в конец очереди и ждать! Во второй половине этой главы мы обсудим предлагаемые архитектуры, предназначенные для устранения этого ограничения.

Таким образом, пока что нам приходится смириться с обслуживанием по остаточному принципу. Но и при этом ограничении мы можем применить несколько методов улучшения потребительских свойств мультимедийных сетевых приложений. Например, мы можем посылать аудио- или видеоданные в UDP-дейтаграммах и таким образом обойти низкую пропускную способность протокола TCP, когда тот входит в фазу медленного старта. Мы можем отложить воспроизведение данных у получателя на 100 мс и более, чтобы снизить эффект вызванного сетью джиттера. Мы можем при передаче каждого пакета поставить в нем метку времени, ^тобы получатель знал, когда его следует воспроизводить. В приложениях для воспроизведения записанного потокового аудио и видео при наличии достаточной пропускной способности можно передавать данные *с* упреждением и сохранять их у получателя. Можно даже посылать избыточную информацию, чтобы компенсировать эффект потери пакетов в сети. Многие из этих методов мы изучим в оставшейся части первой половины этой главы.

Развитие Интернета в направлении поддержки мультимедиа

Сегодня ведутся яростные дебаты о том, как должен развиваться Интернет, чтобы лучше приноровиться к мультимедийному трафику с его строгими временными ограничениями. На одном конце находятся исследователи, утверждающие, что

в Интернете должны быть сделаны радикальные изменения, что позволило бы приложениям явным образом резервировать сквозную пропускную способность. Эти исследователи полагают, что пользователь, желающий, например, позвонить по телефону с хоста A на хост B, должен иметь возможность явно зарезервировать пропускную способность в каждой линии на всем маршруте. Но для реализации подобного подхода требуется сделать очень много. Во-первых, нужен протокол, резервирующий от имени приложений пропускную способность на маршруте от отправителя до получателя. Во-вторых, необходимо модифицировать политику планирования очередей на маршрутизаторах, чтобы те поддерживали резервирование пропускной способности. В соответствии с новой политикой не все пакеты должны обрабатываться одинаково. Напротив, пакеты тех потоков, для которых зарезервирована большая пропускная способность, должны обслуживаться в первую очередь. В-третьих, чтобы маршрутизаторы могли поддержать зарезервированную пропускную способность, приложения должны заранее передать в сеть описание своего будущего трафика. В результате сеть сможет следить за тем, чтобы отправитель не превышал установленных параметров трафика. Наконец, у сети должна быть возможность определить, обладает ли она достаточными ресурсами для удовлетворения нового запроса. Для удовлетворения всех этих запросов требуется новое и сложное программное обеспечение на хостах и маршрутизаторах, а также новые типы служб. Мы рассмотрим эти механизмы подробнее в разделе «Интегрированное обслуживание», когда будем изучать соответствующую модель обслуживания.

Другую крайнюю точку зрения представляют исследователи, утверждающие, что нет необходимости производить какие бы то ни было фундаментальные изменения в предоставляемых на сегодняшний день Интернетом услугах, а также в протоколах Интернета. Вместо этого они отстаивают политику невмешательства.

- По мере увеличения требований Интернет-провайдеры (как верхнего, так и нижнего звеньев) будут наращивать свои сети, чтобы удовлетворить эти новые требования. В частности, Интернет-провайдеры будут увеличивать пропускную способность линий и мощность коммутаторов, чтобы обеспечить приемлемые значения задержки и процент потери пакетов в своих сетях. Таким образом, Интернет-провайдеры будут предоставлять своим клиентам лучшее обслуживание, увеличивая свои доходы за счет роста числа клиентов и повышения стоимости услуг. Как отмечалось в главе 2, Интернет-провайдеры могут также установить кэши в своих сетях, перенося с их помощью ресурсы сетей (web-страницы, записанное потоковое аудио и видео) ближе к пользователям, тем самым снижая трафик Интернет-провайдеров более высокого звена.
- Описанные в главе 2 сети распределения ресурсов (Content Distribution Network, CDN) реплицируют хранящиеся в сети ресурсы и перемещают их на периферию Интернета. Принимая во внимание то, что большая часть трафика Интернета представляет собой записанные ресурсы (web-страницы, MP3- и видеофайлы), сети CDN могут значительно облегчить нагрузку на сети и интерфейсы между Интернет-провайдерами. Более того, сети CDN могут предоставлять поставщикам ресурсов дифференцированное обслуживание: те постав-

щики ресурсов, которые заплатят за услуги сетей CDN, доставят свои ресурсы быстрее и эффективнее.

- Для обработки живого потокового трафика (например, сообщений о спортивных событиях), одновременно посылаемого миллионам пользователей, могут быть развернуты *групповые оверлейные сети*. Групповая оверлейная сеть состоит из серверов, разбросанных по сети (и, возможно, по всему Интернету). Эти серверы и логические линии между ними образуют вместе оверлейную сеть, предназначенную для групповой рассылки данных (см. раздел «Групповая маршрутизация» в главе 4) от отправителя к миллионам пользователей. В отличие от группового протокола IP, в котором функции групповой рассылки выполняют маршрутизаторы на IP-уровне, оверлейные сети осуществляют групповую рассылку на прикладном уровне. Например, хост-источник может отправить поток трем или более оверлейным серверам; этот процесс продолжается, создавая дерево распределения поверх IP-сети с ее маршрутизаторами и хостами. Путем групповой рассылки популярного трафика по оверлейным сетям можно снизить суммарную нагрузку на Интернет еще больше.

Между этими двумя крайними точками зрения располагаются сторонники так называемого дифференцированного обслуживания. Эти исследователи предлагают произвести в Интернете относительно небольшие изменения на сетевом и транспортном уровнях, а также добавить на периферию сети (то есть в интерфейс между пользователем и Интернет-провайдером) простые схемы ценообразования и мониторинга. Идея заключается в том, чтобы использовать небольшое количество классов трафика (возможно, будет достаточно двух классов), отмечать класс трафика в каждой дейтаграмме и предоставлять дейтаграммам различных классов разные уровни обслуживания в очередях маршрутизаторов. Разумеется, обслуживание более высокого класса будет стоить пользователю дороже. Дифференцированное обслуживание мы обсудим в одноименном разделе.

Сжатие аудио- и видеоданных

Прежде чем передавать аудио- или видеоданные по сети, они должны быть оцифрованы и сжаты. Необходимость оцифровки очевидна: компьютерные сети способны передавать только биты, поэтому вся передаваемая по сетям информация должна быть представлена в виде последовательности битов. Сжатие важно, потому что в несжатом виде аудио- и видеоданные занимают очень много места на носителях данных и при передаче по сети требуют большой пропускной способности. Специальные методы сжатия аудио- и видеоданных, учитывающие особенности восприятия их человеком, позволяют снизить объемы передаваемых данных во много раз. Например, несжатое изображение, состоящее из одного миллиона (1024 x 1024) пикселов по 24 бита каждый (по 8 бит для каждой из трех составляющих цвета, то есть для красной, синей и зеленой) занимает 3 Мбайт. Чтобы передать такой файл по линии с пропускной способностью 64 Кбит/с, потребуется 7 мин. Если же сжать это изображение хотя бы в 10 раз, оно будет занимать уже 300 Кбайт, и время, требуемое для его передачи, также сократится в 10 раз.

Исследователи активно занимались вопросами сжатия аудио- и видеоданных более 50 лет, и сегодня существуют буквально сотни популярных методов и стандартов сжатия как аудио-, так и видеоданных. Многие университеты посвящают методам сжатия аудио- и видеоданных целые курсы и даже часто предоставляют для них отдельные курсы — курс для сжатия аудиоданных и курс для сжатия видеоданных. Поэтому мы предоставим здесь краткий ознакомительный материал по этой теме.

Сжатие аудиоданных

Постоянно изменяющийся аналоговый аудиосигнал (это может быть речь или музыка), как правило, преобразуется в цифровую форму следующим образом.

1. Аналоговый сигнал сэмплируется (дискретизируется по времени) с некоторой фиксированной частотой, например 8000 отсчетов в секунду. Величина каждого отсчета представляет собой произвольное вещественное число.
2. Затем значение каждого отсчета «округляется», то есть преобразуется в некоторую величину, принадлежащую конечному множеству величин. Эта операция называется «квантованием». Количество конечных значений, то есть размер множества, как правило, представляет собой степень числа два, например 256.
3. Каждая квантованная величина представляется в виде фиксированного числа битов. Например, при использовании шкалы из 256 квантованных величин каждое значение (отсчет сигнала) может быть представлено одним байтом. Затем все двоичные представления сигнала располагаются друг за другом (конкатенируются) в один общий файл, в результате мы получаем цифровое представление сигнала.

Например, если аналоговый радиосигнал дискретизируется с частотой 8000 отсчетов в секунду и каждый отсчет квантуется и представляется в виде 8 бит, тогда получающийся в результате цифровой сигнал будет иметь скорость в 64 000 бит/с. Затем этот цифровой сигнал может быть преобразован в аналоговую форму (то есть декодирован) для воспроизведения. Хотя, как правило, декодированный сигнал отличается от оригинального аудиосигнала, цифровое представление сигнала можно сделать сколь угодно близким к оригиналу, увеличивая частоту дискретизации и уменьшая шаг квантования. Таким образом, при оцифровке сигнала обязательно возникает вопрос выбора между точностью его цифрового представления и занимаемым этими цифровыми данными местом, а также пропускной способностью, требуемой для их передачи.

Описанный нами здесь метод кодирования называется *кодово-импульсной модуляцией* (Pulse Code Modulation, PCM). При помощи этого метода часто кодируется речь для передачи телефонных разговоров по цифровым каналам связи. В аудиокомпакт-дисках также используется кодово-импульсная модуляция с частотой дискретизации 44 100 Гц и разрешением 16 бит на отсчет, что соответствует скорости передачи данных в 705,6 Кбит/с для монофонического сигнала и 1,411 Мбит/с для стереофонического.

Скорость передачи данных в 1,411 Мбит/с для стереофонической музыки превышает большинство стандартных скоростей доступа в Интернет, и даже скорость

64 Кбит/с для передачи речи превосходит скорость доступа в Интернет через телефонный модем. Чтобы снизить скорость передачи потока данных, применяются методы сжатия аудиоданных. Среди популярных методов сжатия оцифрованной речи можно отметить *GSM(* 13 Кбит/с), *G.729* (8 Кбит/с) и *G.7233* (6,4 Кбит/с и 5,3 Кбит/с), а также большое количество нестандартных методов, включая используемые компанией RealNetworks. Популярным методом сжатия стереофонической музыки высокого качества является стандарт *MPEG 1 layer 3,* чаще называемый *MP3.* Этот стандарт позволяет сжимать аудиоданные с различной степенью. Популярными являются форматы 96 Кбит/с, 128 Кбит/с и 160 Кбит/с, из которых последний позволяет сжимать стереозвук с весьма незначительным ухудшением качества восприятия. Если разбить файл формата MP3 на отдельные фрагменты, каждый фрагмент можно воспроизводить по отдельности. Не имеющий заголовков формат MP3 позволяет передавать файлы в виде потока через Интернет (при условии, что скорость воспроизведения адекватна скорости передачи данных через соединения Интернета). MP3 представляет собой сложный стандарт сжатия, в котором используется психоакустическое маскирование, снижение избыточности и битовая буферизация.

Сжатие видеоданных

Видео представляет собой последовательность изображений (кадров), отображаемых с постоянной частотой (как правило, от 24 до 30 кадров в секунду). Несжатый оцифрованный кадр состоит из массива пикселов, каждый из которых кодируется в виде последовательности битов, представляющих уровень яркости и цвет. Методы сжатия видеоданных основаны на двух типах избыточности: временной и пространственной. Пространственная избыточность представляет собой избыточность в пределах одного кадра. Например, изображение может содержать много пикселов одного цвета. Временная избыточность представляет собой повторение многих фрагментов изображения в соседних кадрах. Например, если два соседних кадра полностью идентичны, нет смысла кодировать оба кадра.

Наиболее популярными для сжатия видеоданных являются методы, описанные в группе стандартов MPEG. К этим стандартам относится стандарт *MPEG 1* для видеофильмов с уровнем качества компакт-дисков (1,5 Мбит/с), стандарт *MPEG 2* для высококачественных DVD-дисков (3-6 Мбит/с), а также стандарт *MPEG 4* для объектно-ориентированного сжатия видеоданных. В стандарте MPEG для сжатия отдельного кадра используется стандарт JPEG. Кроме того, в Интернете очень популярны стандарты сжатия *H.261.* Кроме этих стандартов также существуют многочисленные нестандартные методы, включая QuickTime компании Apple, а также алгоритмы кодирования компании Real Networks.

Если вы хотите узнать больше о методах сжатия аудио- и видеоданных, обратитесь к дополнительным источникам информации [409, 474]. Теме передачи мультимедиа по сети посвящается [103].

Записанное потоковое аудио и видео

В последние годы потоковое аудио и видео стало довольно популярным, а его трафик занял существенную долю пропускной способности сетей. Эта тенденция

продолжается, и тому есть ряд причин. Во-первых, стоимость хранения данных на дисках продолжает быстро снижаться, в результате для хранения фильмов выделяется все больше места. Сегодня сервер можно оборудовать дисковым пространством объемом в несколько терабайт и хранить на нем несколько тысяч фильмов в формате MPEG-2. Во-вторых, распространению видеоданных по сети способствуют такие усовершенствования, как высокоскоростной доступ конечных пользователей (то есть кабельные модемы и цифровые абонентские линии ADSL, обсуждавшиеся в главе 1), методы распределения ресурсов, такие как кэширование и CDN (см. главу 2), а также рассматриваемые в этой главе новые Интернет-протоколы, поддерживающие разные уровни качества обслуживания. В-третьих, наблюдается большой отложенный спрос на высококачественное потоковое видео, которое объединяет в себе две популярные технологии — телевидение и web.

В случае записанного потокового аудио и видео клиентам по их запросам через сеть доставляются сжатые аудио- и видеофайлы, хранящиеся на сервере. Как мы скоро увидим, этими серверами могут быть «обычные» web-серверы или серверы, специально приспособленные для подобных приложений. По запросу клиента сервер направляет в сокет аудио- или видеофайл. На практике используются как TCP-, так и UDP-соединения. Прежде чем отправить аудио- или видеофайл в сеть, файл сегментируется, сегменты, как правило, снабжаются специальными заголовками, соответствующими аудио- или видеотрафику. Для инкапсуляции таких сегментов используется обсуждаемый в разделе «Протоколы для интерактивных приложений реального времени» протокол *RTP* (Real-Time Protocol — протокол реального времени), представляющий собой открытый стандарт. После того как клиент получает первые же фрагменты запрошенного им аудио- или видеофайла, начинается его воспроизведение (обычно с задержкой в несколько секунд). Большинство современных программных продуктов также предоставляют пользователю возможность управлять получением/воспроизведением файла, например останавливать воспроизведение и перематывать фильм вперед или назад. Для реализации такой возможности требуется еще один протокол между клиентом и сервером, которым и является протокол *RTSP* (Real-Time Streaming Protocol — потоковый протокол реального времени), также представляющий собой открытый стандарт (мы вернемся к нему в конце этого раздела).

Пользователи часто запрашивают потоковое аудио или видео при помощи своего web-браузера. Но поскольку средства воспроизведения аудио и видео не интегрированы напрямую в сегодняшние web-браузеры, требуется отдельное *вспомогательное приложение*. Наиболее популярными приложениями такого рода, которые часто называют мультимедийными проигрывателями (media players), являются проигрыватели RealPlayer компании RealNetworks и Media Player компании Microsoft. Мультимедийный проигрыватель выполняет несколько функций.

- *Распаковка*. Аудио- и видеоданные почти всегда хранятся и передаются в сжатом виде, что делается для экономии дискового пространства и пропускной способности сети. Мультимедийный проигрыватель должен «на лету» распаковывать получаемые им данные.
- *Борьба с джиттером*. Джиттер представляет собой изменение задержки в доставке от отправителя до получателя пакетов, относящихся к одному и тому же

потоку. Поскольку аудио- или видеоданные должны воспроизводиться с той же скоростью, с которой они были записаны, получатель должен буферизовать принимаемые пакеты на короткий период времени, чтобы компенсировать неравномерность в поступлении пакетов. Более подробно мы рассмотрим этот вопрос в разделе «Интернет-телефония».

- *Исправление ошибок.* В связи с непредсказуемыми заторами в Интернете часть пакетов потока может быть потеряна. Если потерянных пакетов становится очень много, потребительские свойства воспроизводимого файла могут стать неприемлемыми. Поэтому многие потоковые приложения пытаются исправить ошибки, вызванные потерями пакетов, заранее передавая избыточные пакеты, передавая пакеты повторно по запросу клиента или маскируя потерянные данные путем интерполяции.

Как правило, у мультимедийного проигрывателя есть графический интерфейс с управляющими кнопками. Именно с этим интерфейсом взаимодействует пользователь. Обычно интерфейс пользователя включает ползунок управления громкостью, кнопку остановки и воспроизведения, ползунки для перемещения по аудио- и видеопотоку вперед и назад и т. д.

Для встраивания мультимедийного проигрывателя в окно web-браузера могут использоваться подключаемые модули. В этом случае браузер резервирует место на текущей web-странице, а мультимедийный проигрыватель управляет экранным пространством. Но независимо от того, отображается мультимедийный проигрыватель в отдельном окне или в окне браузера, он представляет собой программу, которая работает независимо от браузера.

Доступ к аудио- и видеоданным через web-сервер

Аудио- или видеоданные могут храниться либо на web-сервере, доставляющем их своим клиентам по протоколу HTTP, или на специальном потоковом сервере, использующем для доставки данных другие протоколы, которые могут быть как стандартными, так и нестандартными. В этом подразделе мы рассмотрим доставку аудио- или видеоданных с web-сервера, а в следующем — с потокового сервера.

Для начала познакомимся с доставкой потокового аудио. Когда аудиофайл располагается на web-сервере, этот файл представляет собой обычный объект в файловой системе сервера, такой же как HTML- или JPEG-файл. Когда пользователь желает послушать этот аудиофайл, хост пользователя устанавливает с web-сервером TCP-соединение и посылает HTTP-запрос на получение этого объекта (см. раздел «Web и HTTP» в главе 2). Получив запрос, web-сервер инкапсулирует аудиофайл в ответное HTTP-сообщение и посылает его обратно по TCP-соединению. Случай с видеоданными может быть несколько более сложным, так как звук и изображение для видеофильма могут храниться в отдельных файлах, то есть представлять собой два отдельных объекта в файловой системе web-сервера. В этом случае на сервер посылаются два отдельных HTTP-запроса (по двум TCP-соединениям в случае протокола HTTP/1.0), и файлы со звуком и с изображением поступают

к клиенту параллельно. Синхронизацией этих двух потоков занимается клиент. Звук и изображение могут также храниться в одном и том же файле чередующимися фрагментами. В этом случае клиенту нужен только один объект. Для простоты в нашем обсуждении мы будем считать, что звук и изображение содержатся в одном файле.

Упрощенная архитектура потокового аудио и видео показана на рис. 6.1.

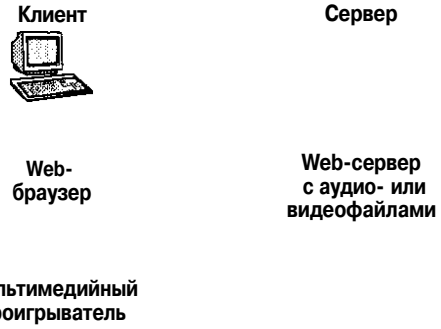


Рис. 6.1. Упрощенная реализация потокового мультимедиа

Такая архитектура функционирует следующим образом.

1. Браузер устанавливает TCP-соединение с web-сервером и запрашивает в HTTP-сообщении аудио- или видеофайл.
2. Web-сервер посылает браузеру запрашиваемый файл в ответном HTTP-сообщении.
3. В строке с информацией о типе содержимого, имеющейся в заголовке ответного HTTP-сообщения, указывается тип кодирования аудио- или видеоданных. Браузер исследует эту информацию, запускает соответствующий мультимедийный проигрыватель и передает ему файл для воспроизведения.
4. Мультимедийный проигрыватель начинает воспроизведение файла.

Хотя такой метод очень прост, у него имеется серьезный недостаток: мультимедийный проигрыватель (то есть вспомогательное приложение) должен взаимодействовать с сервером через web-браузер в качестве посредника. Это может привести к ряду проблем. В частности, если web-браузер является посредником, то прежде, чем передавать файл вспомогательному приложению, браузер должен получить этот файл целиком. В результате задержка между запросом пользователя и началом воспроизведения для аудио- или видеоклипов средней длины будет, скорее всего, неприемлемой. По этой причине, как правило, сервер посылает аудио- или видеофайл напрямую процессу мультимедийного проигрывателя. Другими словами, между серверным процессом и процессом мультимедийного проигрывателя устанавливается прямое соединение через сокет. Как показано на рис. 6.2, обычно такое соединение обеспечивает метафайл, то есть файл, предоставляющий информацию (например, URL-адрес или метод кодирования) о передаваемом аудио- или видеофайле.

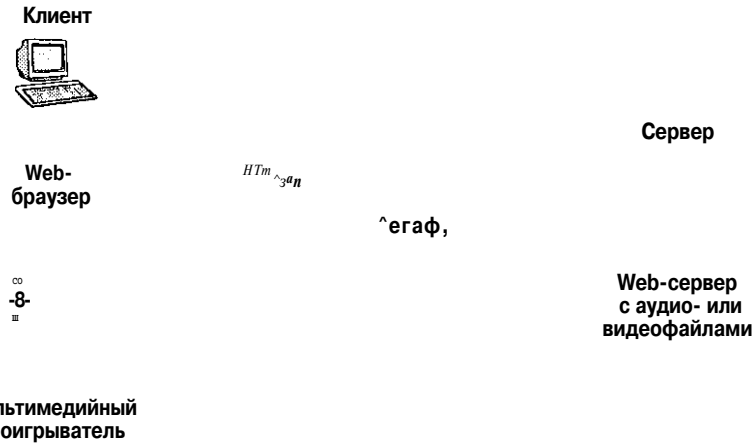


Рис. 6.2. Web-сервер посылает файл напрямую мультимедийному проигрывателю

Прямое TCP-соединение между сервером и мультимедийным проигрывателем устанавливается следующим образом.

1. Пользователь щелкает мышью на гиперссылке аудио- или видеофайла.
2. Гиперссылка указывает не напрямую на аудио- или видеофайл, а на метафайл. В метафайле содержится URL-адрес самого аудио- или видеофайла. В ответное HTTP-сообщение помещается метафайл и строка заголовка, указывающая на тип запрашиваемого мультимедийного файла.
3. Браузер исследует строку заголовка с информацией о типе мультимедийного файла в ответном сообщении, запускает соответствующий мультимедийный проигрыватель и передает ему все тело (то есть метафайл) ответного HTTP-сообщения.
4. Мультимедийный проигрыватель устанавливает TCP-соединение напрямую с HTTP-сервером, после чего посылает по этому соединению серверу HTTP-запрос на аудио- или видеофайл.
5. Аудио- или видеофайл посылается в ответном HTTP-соединении мультимедийному проигрывателю.

Необходимость промежуточного этапа пересылки метафайла должна быть ясна. Зная тип содержимого файла, браузер может запустить соответствующий мультимедийный проигрыватель, после чего тот связывается с сервером напрямую.

Итак, мы рассмотрели, как с помощью метафайла установить прямое соединение мультимедийного проигрывателя с web-сервером, на котором хранится аудио- или видеофайл. Тем не менее многие компании, производящие программное обеспечение для потокового аудио и видео, не рекомендуют такую архитектуру. Объясняется это тем, что в ней мультимедийный проигрыватель общается с сервером по протоколу HTTP, а следовательно, по протоколу TCP. Протокол HTTP обычно считается недостаточно развитым для успешного взаимодействия пользователя с сервером. В частности, с по-

мощью протокола HTTP трудно реализовать отправку на сервер команд управления потоком, таких как пауза, воспроизведение, перемотка и т. д.

Передача мультимедиа с потокового сервера

Проблема недостатков протоколов HTTP и TCP может быть решена, если хранить аудио- и видеоданные на выделенном потоковом сервере, с которого и посылать их мультимедийному проигрывателю. Этот потоковый сервер может быть фирменным (например, такими серверами торгуют компании RealNetworks и Microsoft) или стандартным. С потокового сервера аудио- и видеоданные могут посылаться по протоколу UDP (а не TCP), поверх которого могут работать протоколы прикладного уровня, лучше чем HTTP приспособленные к управлению потоками аудио- и видеоданных.

Для этой архитектуры требуются два сервера, как показано на рис. 6.3. Один (HTTP-сервер) поставляет клиентам web-страницы (включая метафайлы), второй (потоковый сервер) — аудио- и видеофайлы. Два сервера могут работать на одной и той же оконечной системе или на двух разных оконечных системах. Эта архитектура функционирует так же, как предыдущая. Однако теперь мультимедийный проигрыватель запрашивает файл у потокового сервера, а не у web-сервера, поэтому теперь мультимедийный проигрыватель может взаимодействовать с потоковым сервером с помощью собственных протоколов. Эти протоколы способны обеспечить полноценное взаимодействие пользователя с сервером для управления аудио- или видеопотоком.

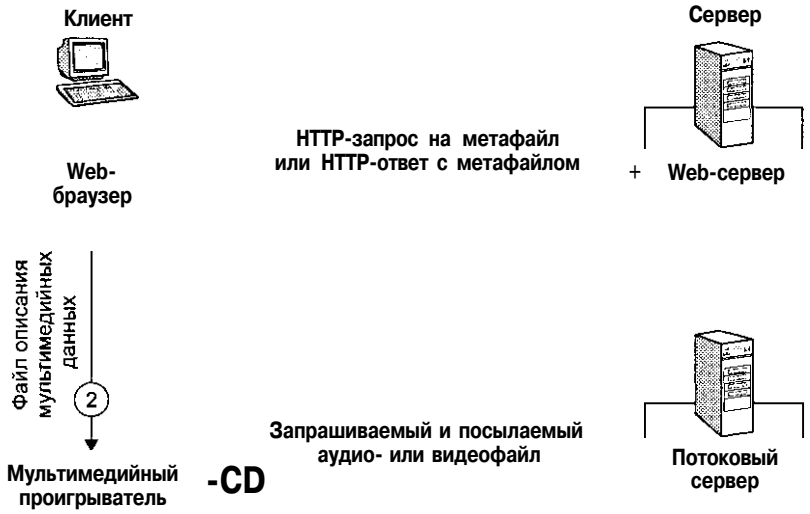


Рис. 6.3. Использование выделенного потокового сервера

В показанной на рисунке архитектуре могут применяться разные варианты доставки аудио- и видеоданных с потокового сервера мультимедийному проигрывателю. Некоторые варианты перечислены ниже.

- Аудио- или видеоданные посылаются по протоколу UDP с постоянной скоростью, равной скорости их отправки получателем. Например, если аудиоданные сжаты по стандарту GSM для передачи со скоростью 13 Кбит/с, тогда сервер передает сжатый аудиофайл со скоростью 13 Кбит/с. Как только клиент получает сжатые аудио- или видеоданные из сети, он распаковывает их и воспроизводит.
- В отличие от первого варианта, мультимедийный проигрыватель откладывает воспроизведение аудио- или видеоданных на время от двух до пяти секунд, чтобы устранить джиттер, вызванный неравномерной доставкой пакетов по сети. Клиент решает эту задачу, помещая полученные им сжатые данные в буфер (рис. 6.4). Создав запас мультимедийных данных, достаточный для нескольких секунд воспроизведения, клиент начинает воспроизводить данные, извлекая их из буфера. В данном, как и в предыдущем, случае скорость заполнения буфера *x(t)* равна скорости его опустошения *d* с той разницей, что в случае потери пакета скорость заполнения буфера *x(t)* временно оказывается меньше скорости опустошения *d*.

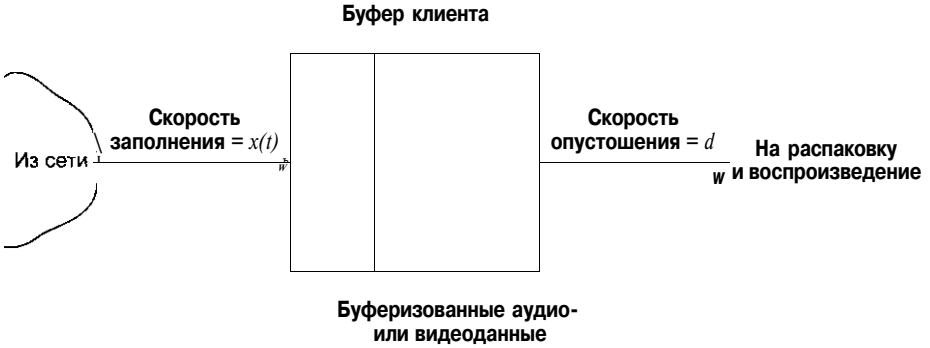


Рис. 6.4. Буфер клиента заполняется и опустошается с разными скоростями

- Мультимедийные данные посылаются по протоколу TCP. Сервер передает файл в TCP-сокет с максимальной скоростью. Клиент (то есть мультимедийный проигрыватель) считывает файл из TCP-сокета с максимальной скоростью и помещает сжатые данные в свой буфер. После задержки в несколько секунд мультимедийный проигрыватель считывает данные из своего буфера со скоростью *d* и направляет считанные данные на распаковку и воспроизведение. Поскольку протокол TCP повторяет передачу потерянных пакетов, он может обеспечить лучшее качество звука, чем протокол UDP. Вместе с тем теперь скорость заполнения буфера *x(t)* оказывается переменной, что вызвано механизмами борьбы с перегрузкой и управления потоком, поддерживаемыми протоколом TCP. Действительно, потеряв пакет, система борьбы с перегрузкой протокола TCP может надолго снизить мгновенную скорость передачи данных до уровня ниже *d*. Это может привести к опустошению буфера клиента и вызвать нежелательные паузы в воспроизведении аудио- и видеопотока.

В последнем вариаР1те режим заполнения буфера клиента (который не следует путать с приемным буфером протокола TCP) в значительной степени зависит от

его размера. Если этот буфер достаточно велик, чтобы вместить весь мультимедийный файл, тогда протокол TCP может использовать всю доступную ему пропускную способность, так что скорость заполнения буфера $x(t)$ может стать много больше скорости его опустошения d. Если такая разница в скоростях будет поддерживаться достаточно долго, последующее «голодание» клиента маловероятно. Если же, с другой стороны, буфер клиента мал, тогда скорость заполнения буфера $x(t)$ не должна сильно отличаться от скорости опустошения d. В этом случае риск голодания клиента значительно выше.

Протокол RTSP

Многие потребители мультимедиа в Интернете (в частности, те, что выросли, не расставаясь с пультом дистанционного управления телевизором) хотели бы управлять процессом воспроизведения мультимедийных данных, останавливая и вновь начиная воспроизведение, перематывая фильм вперед и назад, и т. д. Эти возможности сродни тем, которыми обладает пользователь, просматривающий фильм на обычном проигрывателе DVD-дисков или слушающий музыку на проигрывателе компакт-дисков. Чтобы пользователь мог управлять процессом воспроизведения, мультимедийный проигрыватель должен обмениваться с сервером управляющей информацией по специальному протоколу. Таким протоколом является протокол RTSP (Real-Time Streaming Protocol — потоковый протокол реального времени), определенный в RFC 2326.

Прежде чем перейти к деталям протокола RTSP, скажем, чего протокол RTSP не делает.

- Протокол RTSP не выясняет схему сжатия аудио- и видеоданных.
- Протокол RTSP не определяет метод инкапсуляции аудио- и видеоданных в пакетах для передачи их по сети. Инкапсуляцией потоковых мультимедийных данных может заниматься протокол RTP (см. раздел «Протоколы для интерактивных приложений реального времени») или какой-нибудь нестандартный (фирменный) протокол. То есть протокол RTSP используется аудио- и видеосерверами, а также проигрывателями для передачи друг другу управляющей информации, но сам мультимедийный поток может инкапсулироваться в RTP-пакеты или в пакеты какого-либо нестандартного формата.
- Протокол RTSP не ограничивает средств транспортировки потоковых мультимедийных данных, которые могут передаваться по протоколу UDP или TCP.
- Протокол RTSP не ограничивает способов буферизации аудио- и видеоданных мультимедийным проигрывателем. Последний может начать воспроизводить получаемые данные сразу, отложить воспроизведение на несколько секунд или прежде загрузить данные целиком.

Итак, если протокол RTSP не выполняет перечисленных выше функций, что же он делает? RTSP позволяет мультимедийному проигрывателю управлять передачей потока мультимедийных данных. Как уже отмечалось ранее, в функции управления входят приостановка и продолжение воспроизведения, быстрая перемотка вперед и назад, а также произвольное перемещение места начала воспроизведе-

ния. RTSP является *внеполосным протоколом*. В частности, RTSP-сообщения посылаются отдельно от мультимедийного потока, структура пакетов которого не определяется протоколом RTSP. Для RTSP-сообщений используется отдельный порт с номером 544. Спецификация протокола RTSP (RFC 2326) позволяет посылать RTSP-сообщения при помощи как протокола TCP, так и протокола UDP.

В разделе «Передача файлов по протоколу FTP» главы 2 отмечалось, что в протоколе FTP передача сигнальной информации также идет вне полосы. В частности, протокол FTP использует две пары сокетов на клиенте и на сервере, у каждой из которых свои номера портов: одна пара сокетов поддерживает TCP-соединение для передачи управляющей информации, а сам файл передается по другому TCP-соединению. RTSP-канал во многом напоминает управляющий канал протокола FTP.

Рассмотрим теперь простой пример использования протокола RTSP, который иллюстрирует рис. 6.5. Сначала web-браузер запрашивает у web-сервера файл описания мультимедийных данных (метафайл). В этом файле могут содержаться ссылки на несколько мультимедийных файлов, а также на директивы для их синхронизации; каждая ссылка на мультимедийный файл начинается с URL-метода rtsp://. Ниже приведен пример файла описания мультимедийных данных из [447]. В данном примере аудио- и видеопотоки воспроизводятся параллельно с последующей синхронизацией (как часть одной «группы»). Для аудиопотока мультимедийный проигрыватель может выбирать («переключаться») между двумя аудиозаписями низкого и высокого качества. (Формат этого файла похож на формат SMIL [540], который используется многими потоковыми продуктами для определения синхронизированных мультимедийных потоков.)

```
<title>Twister</title>
<session>
   <group language=en lipsync>
      <switch>
         <track type=audio
                e-'TCMU/eOOO/l"
                src="rtsp://audio.example.com/twister/audio.en/1 ofi">
         <track type=audio
                e='DVI4/16000/2" pt="90 DVI4/8000/1"
                src="rtsp://audio.example.com/twister/audio.en/hi fi">
            </switch>
         <track type="video/jpeg"
                src="rtsp://video.example.com/twister/video">
   </group>
</session>
```

Web-сервер инкапсулирует файл описания мультимедийных данных в ответном HTTP-сообщении, которое отправляет браузеру. Получив ответное HTTP-сообщение, браузер запускает мультимедийный проигрыватель (то есть вспомогательное приложение) в соответствии с полем типа содержимого этого сообщения. С помощью URL-метода **rtsp://,** как показано в приведенном выше примере, в файл описания мультимедийных данных включаются ссылки на мультимедийные потоки. На рисунке показано, что мультимедийный проигрыватель и сервер посылают друг другу серии RTSP-сообщений. Проигрыватель посылает RTSP-запрос **SETUP** (установка соединения), а сервер отвечает RTSP-сообщением **OK.** Затем про-

игрыватель посылает RTSP-запрос PLAY (воспроизвести), например, требуя воспроизведения низкокачественного аудиофайла, и сервер снова отвечает RTSP-сообщением OK. С этого момента потоковый сервер начинает загружать низкокачественный аудиофайл в свой внутриполосный канал. Спустя некоторое время мультимедийный проигрыватель посылает RTSP-запрос PAUSE (приостановить воспроизведение), и сервер отвечает RTSP-сообщением OK. Наконец, когда пользователь завершает работу мультимедийного проигрывателя, проигрыватель посылает серверу RTSP-запрос **TEARDOWN** (разорвать соединении), на что сервер отвечает RTSP-сообщением OK.

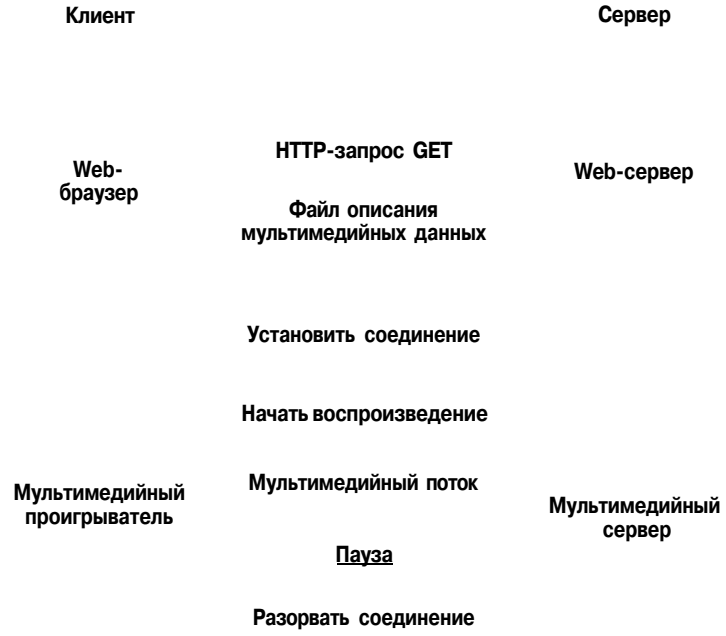


Рис. 6.5. Взаимодействие между клиентом и сервером по протоколу RTSP

Рассмотрим теперь примеры некоторых RTSP-сообщений. Ниже приводится упрощенный пример RTSP-сеанса между клиентом (C:) и сервером (S:).

```
C: SETUP rtsp://audio.example.com/twister/audio RTSP/1.0
   Cseq: 1
   Transport: rtp/udp; compression: port=3056: mode=PLAY
S: RTSP/1.0 200 OK
   Cseq: 1
   Session: 4231
C: PLAYrtsp://audio.example.com/twister/audio.en/lofi
   RTSP/1.0
   Range: npt=0-
```

```
    Cseq: 2
    Session: 4231
S: RTSP/1.0 200 OK
    Cseq: 2
    Session: 4231
C: PAUSE rtsp://audio.example.com/twister/audio.en/lofi
    RTSP/1.0
    Range: npt=37
    Cseq: 3
    Session: 4231
S: RTSP/1.0 200 OK
    Cseq: 3
    Session: 4231
C: TEARDOWN rtsp://audio.example.com/twister/audio.en/
    lofi RTSP/1.0
    Cseq: 4
    Session: 4231
S: RTSP/1.0 200 OK
    Cseq: 4
    Session: 4231
```

Обратите внимание на сходство между протоколами HTTP и RTSP. Все запросы и ответы посылаются в текстовом ASCII-коде. Клиент использует стандартные методы **(SETUP, PLAY, PAUSE** и т. д.), а сервер отвечает на запросы клиента стандартными кодами ответов. Однако одно важное отличие протокола RTSP заключается в том, что RTSP-сервер отслеживает состояние клиента в каждом RTSP-сеансе. Например, сервер помнит, находится ли клиент в состоянии инициализации, воспроизведения или паузы (см. задание по программированию к этой главе). Номера сеанса и порядковые номера, являющиеся частью каждого RTSP-запроса и каждого RTSP-ответа, помогают серверу следить за состоянием сеанса. Номер сеанса неизменен на протяжении всего сеанса. Порядковый номер сообщения увеличивается на единицу клиентом для каждого нового сообщения. Сервер возвращает в ответах номер сеанса и порядковый номер.

Как показано в примере, клиент начинает сеанс запросом **SETUP,** передавая серверу URL-адрес нужного ему файла, а также номер версии протокола RTSP. Сообщение **SETU P** содержит номер порта клиента, которому следует посылать мультимедийные данные. Сообщение **SETUP** также указывает, что эти мультимедийные данные должны посылаться с помощью протокола RTP (он будет обсуждаться в разделе «Протоколы для интерактивных приложений реального времени») поверх протокола UDP. Обратите внимание, что в этом примере проигрыватель решает воспроизводить мультимедийные данные не полностью, а только их низкокачественную составляющую.

В действительности протокол RTSP может выполнять много больше, чем здесь описано. В частности, протокол RTSP позволяет клиенту направлять серверу свой поток мультимедийных данных (например, для записи). Протокол RTSP принят компанией RealNetworks, одним из лидеров в области производства программного обеспечения для приложений потокового аудио и видео. Протоколу RTSP посвящена web-страница, созданная Хеннингом Шульцринне (http://www.cs.columbia.edu/~hgs/rtsp).

В конце этой главы вы найдете задание по программированию, в котором предлагается разработать потоковую видеосистему (и сервер, и клиент), использующую протокол RTSP. Программа должна создавать и посылать клиенту RTSP-сообщения. К заданию прилагается черновик программы RTSP-сервера, осуществляющего грамматический разбор RTSP-сообщений и формирующего соответствующие ответы. Читателям, желающим лучше понять работу протокол RTSP, советуем выполнить это интересное задание.

Интернет-телефония

Применяемый в Интернете протокол IP сетевого уровня предоставляет обслуживание по остаточному принципу, то есть прилагается максимум усилий по доставке каждой дейтаграммы от отправителя к получателю как можно быстрее. Однако не дается никаких гарантий относительно таких параметров доставки, как длительность задержки, джиттер и прбцент потерянных пакетов. Отсутствие гарантий относительно величины задержки и джиттера существенно усложняет проектирование мультимедийных приложений реального времени, таких как Интернет-телефония и видеоконференции, которые исключительно чувствительны к длительности задержки, джиттеру и проценту потерянных пакетов. Тем не менее разработчики этих приложений могут использовать некоторые механизмы, способные поддерживать высокое качество звука и изображения до тех пор, пока задержка, джиттер и количество потерь не превосходят определенных пределов. В этом разделе мы рассмотрим некоторые из этих механизмов. Чтобы сделать наше обсуждение более предметным, мы будем рассматривать эти механизмы в контексте *Интернет-телефонии.* Ситуация с приложениями для проведения видеоконференций практически аналогична [35].

Пользователь, говорящий по телефону в нашем Интернет-приложении, генерирует аудиосигнал, состоящий из чередующихся периодов речи и молчания. Чтобы сэкономить пропускную способность, Интернет-приложение формирует пакеты только в течение периодов речи. В это время отправитель генерирует данные со скоростью 8000 байт/с и раз в 20 мс собирает эти данные в пакет. Таким образом, количество байтов данных в пакете равно (20 мс) x (8000 байт/с) = 160 байт. К каждому пакету присоединяется специальный заголовок, содержимое которого будет обсуждаться далее. Путем обращения к интерфейсу сокета данные вместе с заголовком инкапсулируются в UDP-сегмент. Таким образом, во время разговора UDP-сегмент посылается через каждые 20 мс.

Если каждый пакет добирается до адресата с небольшой постоянной задержкой, тогда адресат будет получать пакеты периодически через каждые 20 мс. В таких идеальных условиях получатель может просто воспроизводить пакеты, как только они поступают. Но, к сожалению, некоторые пакеты могут быть потеряны, а у большинства пакетов, добравшихся до получателя, будут разные задержки даже при незначительной перегрузке на линиях Интернета. По этой причине получатель должен решать задачу о времени воспроизведения пакета, а также каким-то образом реагировать на потерю пакета.

Недостатки обслуживания по остаточному принципу

Как уже отмечалось, обслуживание, основанное на остаточном принципе, может привести к потере пакетов, длительным задержкам доставки и джиттеру. Рассмотрим эти вопросы более подробно.

Потеря пакетов

Рассмотрим один из UDP-сегментов, генерируемых нашим телефонным приложением. UDP-сегмент инкапсулирован в IP-дейтаграмму. Перемещаясь по сети, дейтаграмма проходит через буферы (то есть очереди) маршрутизаторов. Может случиться, что один или несколько буферов на пути следования IP-дейтаграммы от отправителя к получателю окажутся переполненными и не смогут принять прибывшую IP-дейтаграмму. В этом случае IP-дейтаграмма отбрасывается маршрутизатором и не доходит до получателя.

Потерь пакетов можно избежать, если вместо UDP посылать их по протоколу TCP. Вспомним, что протокол TCP повторяет передачу пакетов, не достигших получателя. Однако для приложений интерактивного аудио и видео реального времени, таких как Интернет-телефония, механизмы повторной передачи обычно неприемлемы, так как они увеличивают сквозную задержку [34]. Более того, алгоритм борьбы с перегрузкой протокола TCP после потери пакета снижает скорость передачи данных, а это может серьезно ухудшить качество воспроизводимых данных у получателя и даже привести к неразборчивости речи. По этой причине почти во всех существующих сегодня реализациях Интернет-телефонии используется протокол UDP, а передача потерянных пакетов не повторяется.

Потеря пакетов не обязательно является катастрофой, как могут подумать некоторые читатели. В самом деле, в зависимости от применяемых методов кодирования и передачи звукового сигнала, а также методов маскирования потерь на приемной стороне допустимыми могут считаться потери от 1 до 20 % пакетов. Например, для маскирования потери пакета может использоваться метод FEC (Forward Error Correction — прямое исправление ошибок). Далее будет показано, что при использовании метода FEC вместе с оригинальной информацией передаются избыточные данные, что позволяет восстановить на приемной стороне некоторую часть потерянных данных. Тем не менее при серьезной перегрузке в одной или нескольких линиях связи между отправителем и получателем и значительных потерях пакетов (более 10-20 %) уже ничего не поможет достичь приемлемого уровня качества звука у получателя. Очевидно, обслуживание по остаточному принципу имеет некоторые ограничения.

Сквозная задержка

Сквозная задержка представляет собой сумму задержек передачи, обработки и ожидания пакета в очередях на маршрутизаторах, распространения сигнала в линиях и обработки на оконечных системах. Для таких интерактивных аудиоприложений, как Интернет-телефония, сквозные задержки, не превышающие 150 мс, не воспринимаются на слух; задержки в интервале от 150 мс до 400 мс считаются приемлемыми, хотя и заметны; а задержки, превышающие 400 мс, способны существен-

но затруднить общение пользователей. Принимающая сторона в приложениях Интернет-телефонии, как правило, игнорирует все пакеты, задержавшиеся в пути дольше определенного порогового значения, например более 400 мс. Такие пакеты можно считать потерянными.

Джиттер

Основной составляющей сквозной задержки является случайная задержка ожидания в очередях на маршрутизаторах. Благодаря этим изменяющимся задержкам время между передачей и получением пакета для разных пакетов может быть разным. Это явление называется *джиттером*[1].

Рассмотрим для примера два пакета, передаваемых поочередно нашим приложением. Отправитель посылает второй пакет через 20 мс после первого. Но на приемной стороне интервал между моментами получения этих двух пакетов может быть больше 20 мс. Почему? Предположим, первый пакет прибывает в почти пустую очередь на маршрутизаторе, но незадолго до прибытия на этот маршрутизатор второго пакета в эту же очередь поступает большое количество пакетов от других источников. Поскольку задержка первого пакета на маршрутизаторе будет значительно меньше задержки второго пакета, интервал между моментами их прибытия к конечному получателю окажется больше 20 мс. Этот интервал может также стать меньше 20 мс. Почему? Пусть первый пакет, попав на маршрутизатор, ставится в конец длинной очереди пакетов, но после этого вплоть до прибытия на тот же маршрутизатор второго пакета никакие другие пакеты в очередь не поступают. В этом случае наши два пакета оказываются в очереди друг за другом. Если время, необходимое маршрутизатору для передачи одного пакета, меньше 20 мс, тогда интервал между моментами их прибытия к конечному получателю будет меньше 20 мс.

Эта ситуация аналогична движению автомобильного транспорта по дорогам. Предположим, вы и ваш друг едете каждый в своем автомобиле из Сан-Диего в Финикс. Пусть стиль вождения у вас и вашего друга примерно одинаковый, и вы оба едете со скоростью 100 км/ч при условии, что трафик это позволяет. Наконец, предположим, ваш друг выезжает на час раньше вас. В этом случае вы можете приехать в Финикс не точно через час после вашего друга, а чуть раньше или чуть позже в зависимости от ситуации на дорогах.

Если получатель будет игнорировать джиттер и попытается воспроизводить пакеты сразу же по их получении, качество воспроизведения может быть очень низким даже при незначительной загруженности Интернета. К счастью, с джиттером можно бороться при помощи *порядковых номеров, меток времени* и *задержки воспроизведения,* о чем будет рассказано далее.

Борьба с джиттером у получателя в аудиоприложениях

При использовании аудиоприложений, таких как Интернет-телефония и записанное потоковое аудио, в случае джиттера получатель должен попытаться обеспе-

[1] **Буквально, это слово означает «дрожание». — *Примеч. пер.***

чить синхронизированное воспроизведение пакетов с аудиоданными. Как правило, синхронизация осуществляется с помощью трех механизмов.

- *Добавление к каждому пакету порядкового номера.* Отправитель увеличивает порядковый номер каждого следующего пакета на единицу.
- *Снабжение каждого пакета меткой времени.* Отправитель помечает каждый пакет временем его создания.
- *Задержка воспроизведения у получателя.* Задержка воспроизведения полученных пакетов с аудиоданными должна быть достаточно большой, чтобы большинство пакетов успело прибыть к получателю. Эта задержка может быть фиксированной на протяжении всего сеанса связи или изменяться, приноравливаясь к текущей ситуации. Пакеты, не успевающие прибыть в срок, считаются потерянными. Чтобы попытаться скрыть потерю пакета, получатель может использовать какую-либо форму интерполяции.

Теперь мы поговорим о том, как эти три механизма могут ослабить или даже устранить эффект джиттера. Рассмотрим две стратегии, подразумевающие использование фиксированной и адаптивной задержек воспроизведения.

Фиксированная задержка воспроизведения

В случае фиксированной задержки воспроизведения получатель пытается воспроизводить каждый пакет ровно через q мс после того, как этот пакет был сформирован. Таким образом, если пакет содержит метку времени со значением £, получатель воспроизводит его в момент времени $t + q$ при условии, что пакет прибыл вовремя. Пакеты, прибывающие позже запланированного времени воспроизведения, отбрасываются и считаются потерянными.

Каким следует выбирать значение задержки q? Интернет-телефония допускает задержки до 400 мс, хотя при меньших значениях q можно достичь лучших результатов в плане интерактивности. Вместе с тем если выбрать значение q значительно меньшим, чем 400 мс, тогда из-за джиттера в сети многие пакеты могут не успеть прибыть вовремя. Таким образом, если для данного соединения типичны большие флуктуации сквозной задержки, лучше использовать большие значения q. Если же величина и изменения задержки невелики, то лучше использовать небольшие значения q, возможно, меньше 150 мс.

Проблему выбора между задержкой воспроизведения и потерями пакетов иллюстрирует рис. 6.6, на котором показаны моменты времени создания и воспроизведения пакетов одного потока. Рассматриваются два различных значения начальной задержки воспроизведения. Отправитель генерирует пакеты с равными интервалами, скажем, каждые 20 мс (самая левая лестница). Первый пакет достигает получателя в момент времени г. Как видно на рисунке, пакеты прибывают через неравные промежутки времени, что вызвано сетевым джиттером.
В первом случае выбирается значение задержки p - $г$, и четвертый пакет не успевает прибыть вовремя, поэтому получатель считает его потерянным. Во втором случае значение фиксированной задержки воспроизведения устанавливается равным p - г, и все пакеты успевают достичь получателя в срок, поэтому потерь нет.

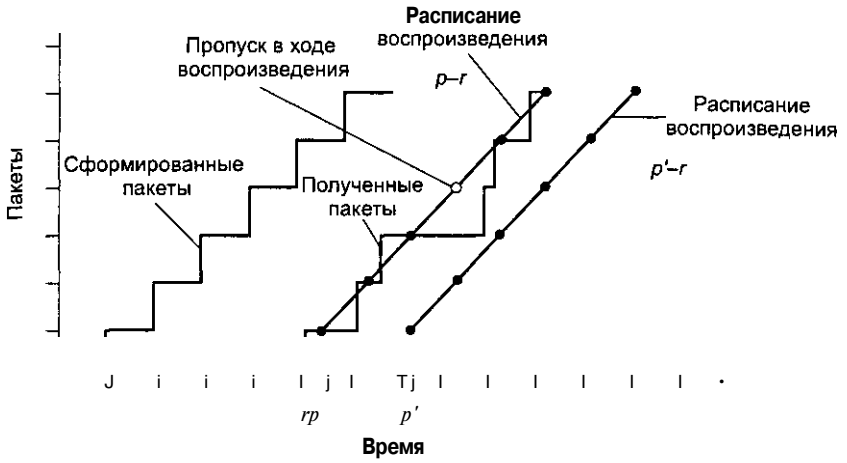


Рис. 6.6. Потеря пакетов для разных значений фиксированной задержки воспроизведения

Адаптивная задержка воспроизведения

Приведенный выше пример демонстрирует важность правильного выбора величины задержки воспроизведения. При выборе больших начальных значений задержки воспроизведения большинство пакетов успеют достичь получателя вовремя, и, следовательно, количество потерянных пакетов будет пренебрежимо мало. Однако для интерактивных служб, какой является Интернет-телефония, длительные задержки могут стать раздражающими или даже неприемлемыми. В идеальном случае было бы желательно минимизировать задержку воспроизведения при условии, что доля потерянных пакетов не будет превышать нескольких процентов.

Естественный способ решения проблемы заключается в том, чтобы оценивать сетевую задержку и ее изменчивость (дисперсию), а затем устанавливать задержку воспроизведения в начале каждого фрагмента разговора (фразы). Адаптивная настройка задержки воспроизведения в начале каждого фрагмента разговора приведет к увеличению и уменьшению пауз между фрагментами разговора, но если эти изменения длительности пауз будут не слишком большими, они окажутся незаметными.

Опишем общий алгоритм, который может использовать получатель для подстройки величины задержки воспроизведения [408]. Введем следующие обозначения:

- t^x — метка времени формирования i-го пакета отправителем;
- г, — время получения i-го пакета;
- p^i — время воспроизведения i-го пакета.

Сквозная задержка г-го пакета равна $r^i - t^i$. Из-за сетевого джиттера эта величина может изменяться от пакета к пакету. Пусть d^t означает оценочную величину *средней* сетевой задержки после получения г-го пакета. Эта оценка образуется из значений меток времени:

где *u* представляет собой фиксированную константу (например, *u* = 0,01). Таким образом, d^i — это сглаженное среднее значение наблюдаемых сетевых задержек $r^x - t^{()}$ /7 - t^i. В усредненной оценке больший вес имеют недавние наблюдения. Сходная форма оценки используется в протоколе TCP для расчета времени оборота пакетов (см. главу 3). Пусть v^i означает оценку среднего отклонения задержки от среднего значения. Эта оценка также формируется из значений меток времени:

$$v^i = \{1-ü)\ v^i\_\{ + u\ \backslash r^i - t^t - d^t\backslash.$$

Значения d^t и $v\text{-}^t$ вычисляются для каждого прибывшего пакета, хотя они используются только для определения момента воспроизведения первого пакета серии.

Вычислив эти величины, для воспроизведения пакетов получатель использует следующий алгоритм. Если i-й пакет является первым пакетом в серии, то время его воспроизведения вычисляется так:

$$Pi = t^i + d^t + Kv^{t1}$$

где *K* представляет собой положительную постоянную величину (например, *K* = 4). Цель введения величины Kv^i заключается в том, чтобы отдалить момент воспроизведения на достаточно долгий срок, тогда только небольшая доля прибывающих пакетов из данной серии окажется потерянной из-за слишком позднего прибытия. Точка воспроизведения для любого последующего пакета в серии вычисляется как смещение от момента времени воспроизведения первого пакета серии. В частности, пусть q^t будет интервалом времени между моментом создания первого пакета серии и моментом его воспроизведения:

Если *j-ü* пакет также принадлежит этой серии, он воспроизводится в момент времени

$$Pj^{=}\ h + q,$$

Описанный только что алгоритм прекрасно работает при условии, что получатель может выявить первый пакет в серии. В отсутствие потерь пакетов это можно сделать, сравнивая метки времени i-го и (г - 1)-го пакета. В самом деле, если t^i - £м > 20 мс, тогда получатель понимает, что i-й пакет начинает новую серию (фразу). Но предположим, что один из пакетов потерялся. В этом случае метки времени двух соседних пакетов будут различаться более чем на 20 мс, несмотря на то что оба пакета принадлежат одной серии. В этой ситуации на помощь приходят порядковые номера пакетов. С их помощью получатель может определить, что означает большая разница в метках времени, новую фразу или потерю пакетов.

Восстановление после потери пакета

Итак, мы узнали, как приложение для Интернет-телефонии может справиться с джиттером пакетов. Теперь мы обсудим некоторые схемы, в которых делается попытка сохранить приемлемое качество звука при потере пакетов. Такие схемы называют *схемами восстановления после потерь.* Определим сначала потерю паке-

та в широком смысле. Пакет считается потерянным, если он вообще не достигает получателя или прибывает позже того момента, когда получатель планирует его воспроизведение. Для обсуждения схем восстановления после потерь мы снова воспользуемся примером из Интернет-телефонии.

Как уже отмечалось в начале этого раздела, повторная передача пакета в интерактивных приложениях реального времени, таких как Интернет-телефония, неприемлема. В самом деле, передавать повторно пакет, не успевший прибыть в срок при первой попытке, абсолютно бессмысленно. А повторная передача пакета, потерянного из-за того, что была переполнена очередь маршрутизатора, не может быть выполнена достаточно быстро. По этим соображениям в приложениях для Интернет-телефонии часто используется какая-либо разновидность метода упреждения потерь. Такими схемами являются уже упоминавшаяся схема *прямого исправления ошибок* (FEC) и схема *чередования*.

Прямое исправление ошибок

Основная идея метода прямого исправления ошибок состоит в том, чтобы добавить к оригинальному потоку пакетов избыточную информацию. За счет увеличения скорости передачи потоковых аудиоданных избыточная информация может использоваться для приблизительного или точного восстановления потерянных пакетов. Рассмотрим два механизма прямого исправления ошибок [34,381]. Первый метод заключается в передаче избыточного закодированного пакета через каждые n пакетов. Избыточный пакет получается как сложение по модулю $2n$ оригинальных пакетов [462]. Таким образом, если любой пакет из группы размером $n + 1$ пакетов теряется, получатель может полностью восстановить его по значениям остальных пакетов. Однако, если теряются два пакета из группы, восстановление оказывается невозможным. При небольшом размере группы и умеренной частоте потерь можно восстановить большую часть потерянных пакетов. Однако чем меньше размер группы, тем выше относительное увеличение скорости передачи потоковых данных (коэффициент увеличения скорости передачи данных равен $1/n$). Например, при $n = 3$ скорость передачи данных увеличивается на 33 %. Кроме того, подобная простая схема приводит к увеличению задержки воспроизведения, так как получатель должен ждать всю группу пакетов, прежде чем он сможет начать воспроизведение. Дополнительные подробности о применении метода прямого исправления ошибок при передаче мультимедиа можно найти в RFC 2733.

Второй механизм прямого исправления ошибок заключается в том, что в качестве избыточной информации посылается аудиопоток пониженного разрешения. Например, отправитель может сформировать номинальный аудиопоток и соответствующий ему поток с меньшей частотой следования двоичных разрядов (номинальный поток может иметь формат PCM с частотой следования двоичных разрядов 64 Кбит/с, а поток пониженного качества может быть в кодировке GSM со скоростью передачи данных 13 Кбит/с). Поток пониженной частоты следования двоичных разрядов называется избыточным. Как показано на рис. 6.7, отправитель формирует n-й пакет из n-го блока данных оригинального потока, добавляя к нему $(n - 1)$-й блок избыточного потока. Таким образом, когда теряется пакет, получатель может скрыть его потерю, воспроизведя блок данных с пониженной*

частотой следования двоичных разрядов, взятый из следующего пакета. Конечно, качество звука при этом будет хуже, однако на слух такое очень кратковременное снижение качества окажется гораздо менее заметным, чем полное отсутствие звука. Обратите внимание, что в данной схеме получатель должен принять всего два пакета, прежде чем сможет начать воспроизведение, поэтому увеличение задержки воспроизведения здесь невелико. Более того, если избыточный поток кодируется со значительно меньшей частотой следования двоичных разрядов, требуемое увеличение скорости передачи данных тоже будет невелико.

Рис. 6.7. Использование мультимедийного потока пониженного качества в роли избыточной информации

Чтобы справиться с потерей двух соседних пакетов, может использоваться простая комбинация. Вместо того чтобы добавлять к *n-му* блоку данных оригинального потока только один *(n* - 1)-й блок из избыточного потока, отправитель может добавить *(n* - 1)-й и *(n* - 2)-й блоки из избыточного потока. Если и этого окажется недостаточно, можно добавить также и *(n* - 3)-й блок из избыточного потока и т. д. Таким образом, можно добиться приемлемого качества даже на линиях с высокой вероятностью потерь. Однако при добавлении дополнительных блоков увеличиваются суммарная скорость передачи данных и задержка воспроизведения.

Метод прямого исправления ошибок применяется в двух таких хорошо документированных приложениях для Интернет-телефонии, как Free Phone (http://www-sop.inria.fr/rodeo/fphone/) и RAT (http://www-mice.cs.ucLac.uk/multimedia/software/rat/). Дополнительную информацию по этой теме см. в [428].

Чередование

В качестве альтернативы методу передачи избыточной информации в приложении для Интернет-телефонии может использоваться чередование аудиоданных. Как показано на рис. 6.8, отправитель изменяет порядок следования блоков аудиоданных перед передачей таким образом, что бывшие соседними блоки данных ока-

RTP

В предыдущем разделе мы говорили о том, что передающая сторона мультимедийного приложения, прежде чем передать блоки аудио и видеоданных транспортному уровню, добавляет к ним заголовки. Эти заголовки содержат порядковые номера и метки времени. Поскольку использовать порядковые номера и метки времени способна большая часть мультимедийных приложений, было бы удобно стандартизировать структуру пакета с включением полей для аудио- или видеоданных, порядкового номера и метки времени, а также других полезных полей. Таким стандартом является RTP, определенный в RFC 1889. Стандарт RTP может использоваться для передачи по сети таких популярных мультимедийных форматов, как PCM, GSM и MP3 для аудио, и MPEG и H.263 для видео. Он также может использоваться для передачи аудио- и видеоданных нестандартных форматов. Сегодня стандарт RTP получил применение в сотнях продуктов и исследовательских прототипах. Он также дополняет другие важные интерактивные протоколы реального времени, включая SIP и H.323.

В этом разделе мы познакомимся со стандартом RTP и часто сопровождающим его протоколом RTCP. Читателям также предлагается посетить посвященный протоколу RTP web-сайт Хеннинга Шульцринна (http://www.cs.columbia.edu/~hgs/rtp), на котором представлено огромное количество информации по этой теме. Кроме того, читатели могут посетить сайт Free Phone (http://freenet.sourceforge.net/), посвященный приложениям для Интернет-телефонии, использующим протокол RTP.

Основные сведения

Протокол RTP, как правило, работает поверх протокола UDP. Передающая сторона инкапсулирует блок мультимедийных данных в RTP-пакет, затем помещает RTP-пакет в UDP-сегмент, который передает протоколу IP. Получающая сторона извлекает из UDP-сегмента RTP-пакет, а затем извлекает из RTP-пакета блок мультимедийных данных, который передает мультимедийному проигрывателю для декодирования и воспроизведения.

В качестве примера рассмотрим, как с помощью протокола RTP передавать аудиоданные. Предположим, что источник кодирует речь методом кодово-импульсной модуляции (PCM) с частотой следования битов 64 Кбит/с, то есть данные дискретизированы, квантованы и оцифрованы. Пусть приложение собирает данные в блоки по 20 мс, тр есть по 160 байт в блоке. Передающая сторона предваряет каждый блок *RTP-заголовком,* в который включаются тип кодирования аудиоданных, порядковый номер и метка времени. Как правило, RTP-заголовок состоит из 12 байт. Блок аудиоданных вместе с RTP-заголовком образует *RTP-пакет*. Затем RTP-пакет посылается по интерфейсу UDP-сокета. На приемной стороне приложение принимает RTP-пакет от интерфейса своего сокета, извлекает блок аудиоданных из RTP-пакета и с помощью полей заголовка RTP-пакета декодирует и воспроизводит аудиоданные.

Если для обозначения типа полезной нагрузки, порядковых номеров и меток времени приложение использует стандарт RTP вместо одной из фирменных схем, то ему легче взаимодействовать с другими сетевыми мультимедийными прило-

жениями. Например, если две компании разрабатывают программное обеспечение для Интернет-телефонии и обе эти компании используют в своих продуктах стандарт RTP, то есть надежда, что пользователи программ этих двух компаний смогут общаться друг с другом. В подразделе «Протокол SIP» данного раздела мы увидим, что протокол RTP часто применяется вместе с телефонными стандартами Интернета.

Следует подчеркнуть, что сам протокол RTP не предоставляет механизма, гарантирующего своевременную доставку данных или каких-либо иных гарантий качества обслуживания. Он даже не гарантирует саму доставку пакетов и сохранение порядка их следования. Действительно, протокол RTP реализуется только на оконечных системах, а маршрутизаторы не отличат IP-дейтаграмм, переносящих RTP-пакеты, от других 1P-дейтаграмм.

Протокол RTP позволяет каждому источнику (например, камере или микрофону) формировать собственный RTP-поток пакетов. Например, для видеоконференции между двумя пользователями могут быть открыты четыре RTP-потока: два потока для передачи звука (по одному в каждом направлении) и два потока для передачи изображения (также по одному в каждом направлении). Однако многие популярные методы кодирования, включая стандарты MPEG 1 и MPEG 2, в процессе кодирования объединяют аудио и видео в один поток. В этих случаях для каждого направления создается только один RTP-поток.

RTP-пакеты могут использоваться не только в приложениях для выборочной рассылки, но и в приложениях для групповой рассылки по схеме с одним отправителем и множеством получателей или с множеством отправителей и множеством получателей. В случае сеанса групповой рассылки в схеме с множеством отправителей все они, как правило, передают свои RTP-потоки в одну и ту же группу рассылки. Связанные друг с другом групповые RTP-потоки (например, генерируемые несколькими отправителями аудио- и видеопотоки в видеоконференции) относятся к одному *RTP-сеансу*.

Поля заголовка RTP-пакета

Четырьмя основными полями RTP-пакета являются поля типа полезной нагрузки, порядкового номера, метки времени и идентификатора синхронизации источника (рис. 6.9).

Рис. 6.9. Поля заголовка RTP-пакета

Поле типа полезной нагрузки RTP-пакета состоит из семи битов. В аудиопотоках это поле индицирует метод кодирования аудиоданных (например, кодово-импульсной модуляции, адаптивной дельта-модуляции, линейного предикативного кодирования). Если отправитель решает изменить метод кодирования посреди сеанса, он может информировать об этом получателя при помощи поля типа полезной нагрузки.

RTP-заголовков, включая поле порядкового номера и поле метки времени. Для каждого созданного вами RTP-пакета вам придется указать соответствующие порядковый номер и метку времени. Ваша программа должна будет явно выполнять эти действия на передающей стороне приложения. Как показано на рис. 6.10, программным интерфейсом вашего приложения с сетью будет стандартный прикладной программный интерфейс (API) UDP-сокета.

Рис. 6.10. RTP является частью приложения и располагается над UDP-сокетом

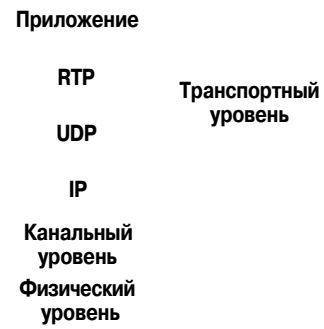


Рис. 6.11. RTP может рассматриваться как подуровень транспортного уровня

Альтернативный подход (его нет в задании по программированию) заключается в использовании для реализации RTP-операций Java-класса (или RTP-библиотеки при программировании на языке C). При таком подходе, как показано на рис. 6.11, с точки зрения разработчика приложения RTP является частью транспортного уровня, при этом программный интерфейс протоколов RTP и UDP располагается между приложением (прикладным уровнем) и транспортным уровнем. Не вдаваясь в мелкие детали (так как они зависят от класса или библиотеки), скажем, что при передаче блока мультимедийных данных в API передающая сторона приложения должна предоставить интерфейс с самим блоком данных (тип полезной нагрузки, SSRC и метку времени, а также номер порта и IP-адрес получателя). Полная реализация протокола RTP содержится в Java Media Framework OMF).

Протокол RTCP

В RFC 1889 определяется протокол RTCP (RTP Control Protocol — управляющий протокол RTP), который сетевые мультимедийные приложения могут использовать вместе с протоколом RTP. Как показано на рис. 6.12, RTCP-пакеты путем групповой IP-рассылки передаются каждым участником RTP-сеанса всем остальным участникам. Как правило, для RTP-сеанса имеется единый групповой адрес, который используют все RTP- и RTCP-пакеты, относящиеся к этому сеансу. RTP- и RTCP-пакеты отличаются друг от друга номерами портов. (Номер порта RTCP получается из номера порта RTP добавлением к нему единицы.)

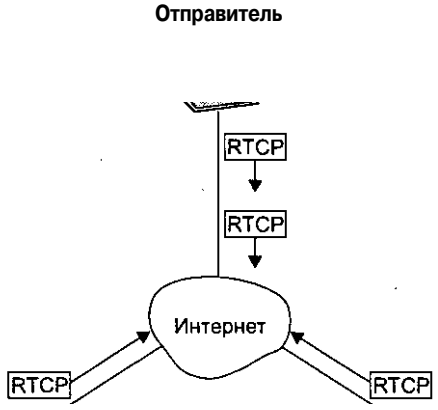


Рис. 6.12. Отправители и получатели обмениваются RTCP-сообщениями

RTCP-пакеты не содержат аудио- или видеоданных. Они содержат сообщения отправителя или получателя с полезной для приложения статистикой и рассылаются периодически. В статистику включается количество отправленных и потерянных пакетов, а также величина джиттера. В спецификации RTP (RFC 1889) не указывается, что должно делать приложение с этой информацией; все оставлено на усмотрение разработчика. Например, отправители могут, опираясь на нее, изменять скорость передачи данных, реализуя, таким образом, обратную связь. Кроме того, подобная обратная связь может быть полезна для диагностики. Например, получатели с ее помощью могут определить, являются ли проблемы локальными, региональными или глобальными.

Типы RTCP-пакетов

Для каждого RTP-потока, принимаемого получателем в рамках одного сеанса, получатель формирует отчет о приеме. Эти отчеты складываются получателем в один

RTCP-пакет, который затем переправляется по дереву групповой рассылки, связывающему всех участников сеанса. Отчет о приеме включает несколько полей, наиболее важные из которых перечислены ниже.

- *Идентификатор синхронизации источника* (SSRC) RTP-потока, для которого генерируется отчет о приеме.
- *Доля потерянных пакетов RTP-потока.* Каждый получатель вычисляет количество потерянных RTP-пакетов и делит это число на общее количество посланных в потоке RTP-пакетов. Если отправитель постоянно получает отчеты о приеме, в которых указывается, что получатели принимают лишь малую долю переданных им пакетов, он может перейти на более низкую скорость кодирования данных, рассчитывая тем самым снизить перегрузку в сети и повысить процент успешно принимаемых пакетов.
- *Последний порядковый номер, полученный в потоке RTP-пакетов.*
- *Величина джиттера,* представляющая собой оценку изменчивости интервала между получениями соседних пакетов в RTP-потоке.

Для каждого передаваемого RTP-потока отправитель создает и передает RTCP-пакеты с отчетами отправителя. В эти пакеты помещается информация о RTP-потоке, включая:

- идентификатор синхронизации источника (SSRC) RTP-потока;
- метку времени и реальное время последнего сформированного RTP-пакета в потоке;
- количество пакетов, переданных в поток;
- количество байтов, переданных в поток.

Отчеты отправителя могут использоваться для синхронизации различных мультимедийных потоков одного RTP-сеанса. Например, представьте себе видеоконференцию, в которой каждый отправитель генерирует по два независимых RTP-потока, один для изображения и второй для звука. Метки времени в этих RTP-пакетах привязаны к таймерам дискретизации видео и аудио, и не имеют отношения к таймеру реального времени. Каждый RTCP-отчет отправителя содержит метку времени последнего RTP-пакета и значение реального времени создания пакета. Таким образом, отчет отправителя связывает время дискретизации с реальным временем. Получатели могут использовать эту информацию для синхронизации звука и изображения при воспроизведении.

Для каждого передаваемого RTP-потока отправитель также создает и передает пакеты описания источника. В этих пакетах содержится информация об источнике, например адрес электронной почты, имя, идентификатор приложения, породившего поток. В них также включается идентификатор SSRC потока. Эти пакеты позволяют по идентификатору SSRC определить имя пользователя или хоста, являющегося источником потока.

Отчеты о приеме, отчеты отправителя и описания источника могут объединяться в одном пакете. Затем такой пакет инкапсулируется в UDP-сегмент и пересылается по дереву групповой рассылки.

Проблема масштабирования протокола RTCP

Внимательный читатель, вероятно, уже заметил, что у протокола RTCP есть потенциальная проблема масштабирования. Рассмотрим, например, RTP-сеанс, в котором взаимодействуют один отправитель и множество получателей. Если каждый получатель периодически генерирует RTCP-пакеты, тогда суммарная скорость передачи данных RTCP-пакетов может значительно превысить скорость передачи данных RTP-пакетов, посылаемых отправителем. Обратите внимание, что объем RTP-трафика, направляемого в дерево групповой рассылки, не изменяется при увеличении числа получателей. В то же время объем RTCP-трафика с увеличением числа получателей линейно растет. Для решения этой проблемы масштабирования протокол RTCP снижает частоту, с которой участник посылает RTCP-пакеты в дерево групповой рассылки, при росте числа участников сеанса. Поскольку каждый участник сеанса пересылает управляющие пакеты всем остальным, каждый участник способен оценить число участников сеанса [169].

Протокол RTCP пытается ограничить свой трафик значением в 5 % от суммарного трафика сеанса. Пусть, например, имеется один отправитель, посылающий видеоданные со скоростью 2 Мбит/с. В этом случае протокол RTCP пытается ограничить свой трафик значением в 5 % от 2 Мбит/с, то есть 100 Кбит/с. Делает он это следующим образом. Протокол предоставляет 75 % от этой пропускной способности, то есть 75 Кбит/с, получателям. Остальные 25 % отдаются отправителю. Выделенные получателям 75 Кбит/с равномерно распределяются между получателями. Таким образом, если есть *R* получателей, тогда каждый из них получает возможность посылать RTCP-сообщения со скоростью *75/R* Кбит/с, а отправитель получает возможность посылать RTCP-сообщения со скоростью 25 Кбит/с. Участник (отправитель или получатель) определяет период передачи RTCP-пакета, динамически вычисляя средний (по всему сеансу) размер RTCP-пакета и деля его на величину выделенной скорости. Таким образом, период передачи RTCP-пакетов отправителя

$$T = \frac{\text{число отправителей}}{0{,}25 \cdot 0{,}05 \cdot \text{трафик сеанса}} (\text{средний размер RTCP-пакета}).$$

А для получателя период передачи RTCP-пакетов

$$T = \frac{\text{число получателей}}{0{,}75 \cdot 0{,}05 \cdot \text{трафик сеанса}} (\text{средний размер RTCP-пакета}).$$

Протокол SIP

Представьте себе, что, когда вы работаете на персональном компьютере, адресованные вам телефонные звонки прибывают на ваш персональный компьютер по Интернету. Когда вы встаете и идете куда-нибудь, адресованные вам телефонные звонки автоматически перенаправляются на ваш карманный компьютер. А когда вы ведете автомобиль, адресованные вам телефонные звонки автоматически пере-

правляются некоему Интернет-устройству в вашем автомобиле. Участвуя в телеконференции, вы можете обратиться к электронной записной книжке и пригласить других абонентов принять участие в конференции. Другие участники могут сидеть за своими персональными компьютерами, прогуливаться со своими карманными компьютерами или ехать в своих автомобилях — не важно, где они находятся, ваше приглашение прозрачным образом переправляется им. А когда вы открываете в браузере чью-либо личную web-страницу, на ней вы видите ссылку «Позвони мне», при щелчке на которой между вашим персональным компьютером и владельцем web-страницы (где бы он ни находился) устанавливается телефонный сеанс.

В описанном виртуальном мире более не нужна телефонная сеть с коммутацией каналов. Вместо этого все звонки проходят по Интернету — от начала до конца. В том же самом мире компании более не пользуются собственными телефонными системами PBX (Private Branch eXchange), то есть локальными коммутаторами для обработки внутренних телефонных звонков. Вместо этого внутренний телефонный трафик компании перемещается по высокоскоростным локальным сетям компании.

Все это, возможно, выглядит фантастично. И, разумеется, сегодняшние сети с коммутацией каналов, а также телефонные системы, используемые внутри отдельной компании, в ближайшем будущем полностью не исчезнут [251]. Тем не менее уже сейчас существуют протоколы и программное обеспечение, способные превратить эту мечту в реальность. Среди многообещающих протоколов в этой области можно отметить протокол SIP (Session Initiation Protocol — протокол инициирования сеанса), определенный в RFC 3261.

- Протокол SIP предоставляет механизмы для установки телефонных соединений по IP-сети. Он позволяет звонящему уведомить своего абонента о том, что хочет установить соединение, а обоим абонентам договориться о методах кодирования мультимедийных данных, а также прервать соединение.
- Протокол SIP предоставляет механизмы, позволяющие звонящему определить текущий IP-адрес абонента. У пользователей нет фиксированных IP-адресов, так как адреса могут выделяться им динамически (при помощи протокола DHCP), или у них может быть несколько IP-устройств, каждое с собственным IP-адресом.
- Протокол SIP предоставляет механизм для управления соединением, например, он позволяет во время сеанса добавлять новые мультимедийные потоки, изменять метод кодирования данных, добавлять новых участников и т. п.

Установка соединения с известным IP-адресом

Чтобы понять, как работает протокол SIP, лучше всего рассмотреть конкретный пример. В нашем примере Алиса сидит за своим персональным компьютером и хочет позвонить Бобу, также работающему на своем персональном компьютере. Персональные компьютеры Алисы и Боба оборудованы поддерживающим протокол SIP программным обеспечением, позволяющим звонить по телефону и принимать телефонные звонки. В этом примере мы будем предполагать, что Алиса

знает IP-адрес персонального компьютера Боба. Процесс установки соединения иллюстрирует рис. 6.13.

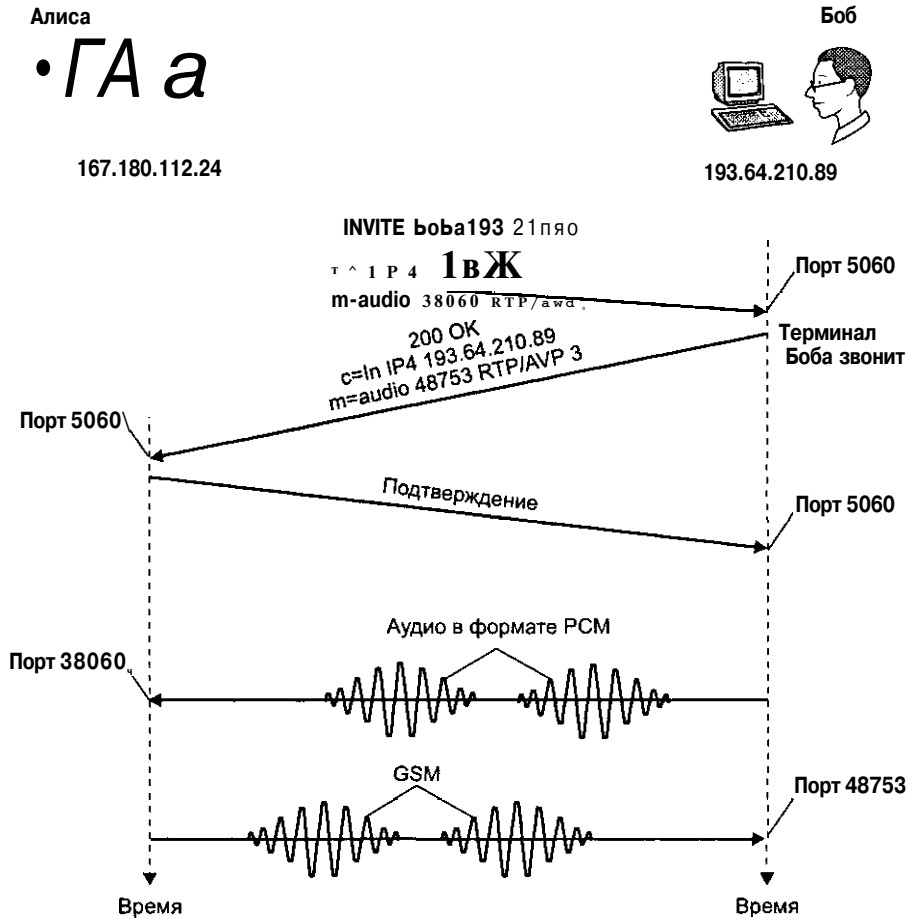


Рис. 6.13. Установка соединения

Как показано на рисунке, SIP-сеанс начинается с того, что Алиса отправляет Бобу сообщение INVITE (пригласить), напоминающее HTTP-запрос. Это сообщение посылается по протоколу UDP в порт 5060, зарезервированный за протоколом SIP. (SIP-сообщения могут также посылаться по протоколу TCP.) Сообщение INVITE содержит идентификатор Боба (bob@193.64.210.89), текущий IP-адрес Алисы, указание на то, что Алиса хочет получать аудиоданные, которые должны кодироваться в формате PCM и инкапсулироваться в RTP-пакеты, а также номер порта (38060) для приема ответных RTP-пакетов. Получив сообщение INVITE Алисы, Боб посылает ответное SIP-сообщение, напоминающее ответное сообщение протокола HTTP. Это ответное SIP-сообщение также посылается в SIP-порт 5060. Ответ Боба содержит код 200 0K, а также его IP-адрес и номер порта, кодировку, которую

он предпочитает, и прочие параметры. Обратите внимание, что в данном примере Алиса и Боб собираются использовать разные методы кодирования аудиоданных: Боб просит Алису кодировать свои аудиоданные в формате GSM, а Алиса просит Боба кодировать передаваемы им аудиоданные в формате PCM. Получив ответ Боба, Алиса отправляет Бобу SIP-подтверждение. Обменявшись перечисленными SIP-сообщениями, Алиса и Боб могут начать разговор. (Для простоты на рисунке показано, что Алиса говорит после Боба, но в действительности они могут говорить одновременно.) Боб кодирует и пакетирует аудиоданные так, как его об этом просит Алиса, и отправляет аудиопакеты в порт 38060 по IP-адресу 167.180.112.24. Алиса также кодирует и пакетирует аудиоданные так, как ее об этом просит Боб, и отправляет аудиопакеты в порт 48753 по IP-адресу 193.64.210.89.

Из этого простого примера мы узнали несколько ключевых характеристик протокола SIP. Во-первых, протокол SIP работает вне полосы: SIP-сообщения посылаются и принимаются через собственные сокеты, не имеющие отношения к сокетам передачи и приема мультимедийных данных. Во-вторых, сами SIP-сообщения представляют собой текст в кодировке ASCII и напоминают HTTP-сообщения. В-третьих, протокол SIP требует, чтобы на все сообщения посылались подтверждения, так что он может работать поверх протокола UDP или TCP.

Рассмотрим, что произойдет, если у Боба нет кодека PCM для кодирования аудиоданных. В этом случае вместо кода 200 0K Боб, скорее всего, ответит кодом 600 Not Acceptable (неприемлемо) и перечислит в своем сообщении все имеющиеся у него кодеки. Затем Алиса может выбрать любой кодек из списка и послать новое сообщение INVITE, на этот раз предлагая использовать выбранный кодек. Боб также может просто отвергнуть запрос, послав один из возможных отрицательных ответов. (Существует множество подобных кодов, включая «я занят», «меня нет на месте», «необходимо заплатить» и «запрещено».)

Адреса протокола SIP

В предыдущем примере SIP-адрес Боба был sip:bob@193.64.210.89. Однако можно ожидать, что скоро многие SIP-адреса (или даже большая часть SIP-адресов) будут напоминать адреса электронной почты. Например, адрес Боба мог бы быть таким: sip:bob@domain.com. Когда SIP-устройство Алисы посылает сообщение INVITE, в него помещается этот адрес, напоминающий адрес электронной почты. Затем инфраструктура протокола SIP направляет сообщение IP-устройству, которым в данный момент пользуется Боб (мы обсудим это далее). Другими возможными формами SIP-адресов может быть телефонный номер Боба или просто комбинация его имени и фамилии при условии их уникальности.

Интересная особенность SIP-адресов заключается в том, что они могут включаться в web-страницы подобно адресам электронной почты. Например, пусть у Боба есть личная web-страница, и он хочет предоставить посещающим эту страницу пользователям возможность позвонить ему по телефону. Он может просто включить в страницу SIP-адрес sip:bob@domain.com. Когда посетитель щелкает мышью на этом адресе, запускается SIP-приложение на устройстве посетителя и Бобу отправляется сообщение INVITE.

Сообщения протокола SIP

В этом кратком введении мы не станем описывать все типы сообщений и заголовков протокола SIP. Вместо этого мы рассмотрим SIP-сообщение INVITE, а также некоторые общие строки заголовка. Пусть Алиса хочет начать телефонный разговор с Бобом по протоколу IP, но на этот раз Алиса знает только SIP-адрес Боба bob@domain.com, а IP-адрес устройства, которым в данный момент пользуется Боб, ей не известен. В этом случае ее сообщение может выглядеть следующим образом:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c-IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

Подобно HTTP-запросу, строка INVITE содержит версию протокола SIP. Каждый раз, когда SIP-сообщение проходит через SIP-устройство (включая устройство, генерирующее сообщение), оно добавляет заголовок Via, в котором указывается IP-адрес устройства. (Мы вскоре увидим, что типичное сообщение INVITE проходит через многие SIP-устройства, прежде чем достигнет SIP-устройства абонента.) Подобно сообщению электронной почты, SIP-сообщение включает строку заголовка From (от кого) и строку заголовка To (кому), а также строку заголовка Call-ID (идентификатор звонка), однозначно идентифицирующую звонок (подобно идентификатору сообщения электронной почты), и строку заголовка Content-Type, определяющую формат, используемый для описания содержимого SIP-сообщения. Строка заголовка Content-Length представляет длину содержимого сообщения в байтах. Наконец, в отделенном пустой строкой (символами возврата каретки и перевода строки) фрагменте передается содержимое сообщения. В этом случае содержимое представляет собой информацию об IP-адресе Алисы и о том, как Алиса хочет получать аудиоданные.

Трансляция имен и расположение пользователя

В примере на рис. 6.13 мы предполагали, что SIP-устройству Алисы был известен IP-адрес устройства, которым пользовался Боб. Однако такое предположение далеко от реальности не только потому, что IP-адреса часто назначаются динамически протоколом DHCP, но также потому, что у Боба может быть несколько IP-адресов (например, для разных устройств дома, на работе, в автомобиле). Таким образом, предположим, что Алисе известен только адрес электронной почты Боба bob@domain.com и тот же адрес используется для телефонных звонков по протоколу SIP. В этом случае Алиса должна получить IP-адрес устройства, которым в данный момент пользуется пользователь bob@domain.com. Чтобы узнать его, Алиса формирует сообщение, начинающееся со строки INVITE sip:bob@domain.com SIP/2.0, и посылает это сообщение прокси-серверу протокола SIP. Прокси-сервер отвечает SIP-сообщением с IP-адресом устройства, который в данный момент использует

В частности, на этом сайте вы найдете свободно распространяемое в исходных текстах программное обеспечение для SIP-клиентов и SIP-серверов **(http://www.cs.columbia.edu/IRT/software).**

Стандарт H.323

Популярный стандарт H.323, предназначенный для проведения аудио- и видеоконференций в реальном времени между оконечными системами по Интернету, представляет собой альтернативу стандарту SIP. Как показано на рис. 6.15, стандарт описывает, как присоединенные к Интернету оконечные системы взаимодействуют с телефонами, соединенными обычной телефонной сетью с коммутацией каналов. (Стандарт SIP также обеспечивает такую возможность, хотя мы это не обсуждали.) Привратник представляет собой устройство, аналогичное SIP-регистратору.

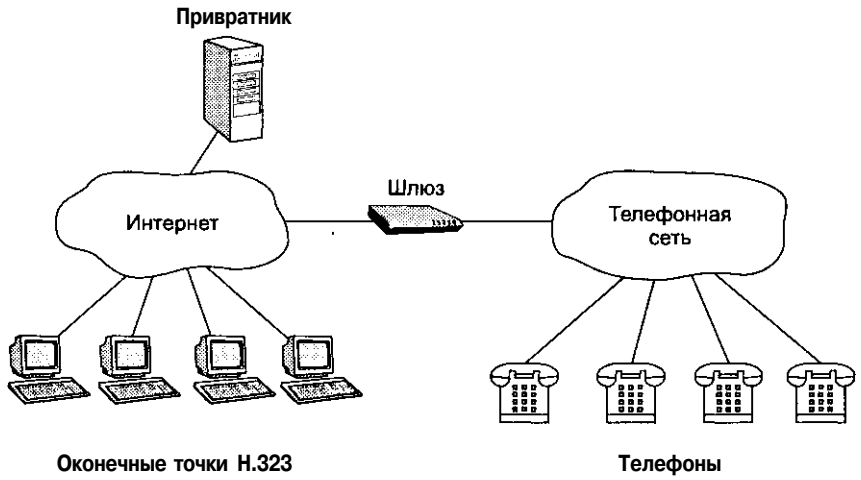


Рис. 6.15. Оконечные системы стандарта H.323

Ниже перечислены спецификации, которые стандарт H.323 объединяет в одном документе.

- Спецификация, описывающая, как оконечные точки договариваются о методе кодирования аудио- или видеоданных. Поскольку стандарт H.323 поддерживает множество методов кодирования аудио- и видеоданных, нужен протокол, чтобы оконечные точки могли договориться о выборе определенного метода.
- Спецификация, описывающая, как аудио- и видеоданные инкапсулируются и посылаются по сети. В частности, стандартом H.323 предписывается использовать для этой цели протокол RTP.
- Спецификация, описывающая, как оконечные точки обмениваются информацией со своими привратниками.
- Спецификация, описывающая, как Интернет-телефоны взаимодействуют через шлюзы с «нормальными» телефонами, связанными обычной телефонной сетью с коммутацией каналов.

Каждая оконечная точка стандарта H.323 *должна* поддерживать стандарт сжатия речи G.711. Стандарт G.711 использует для формирования цифрового сигнала кодово-импульсную модуляцию (PCM) со скоростью передачи данных 56 Кбит/с или 64 Кбит/с. Хотя в соответствии со стандартом H.323 оконечные точки должны поддерживать цифровую передачу голоса (по стандарту G.711), еще они могут поддерживать передачу видеоданных. Поскольку поддержка воспроизведения и передачи видеоданных не является обязательной, производители терминалов могут продавать упрощенные голосовые терминалы, а также более сложные терминалы, поддерживающие воспроизведение и передачу как аудио-, так и видеоданных. Хотя поддержка видео в оконечных точках не обязательна, если такая точка обеспечивает воспроизведение и передачу видеоданных, она должна поддерживать (как минимум) видеостандарт QCIF H.261 (176 x 144 точек).

В дополнение к описанным выше стандартам и протоколам стандарт H.323 предписывает использовать управляющий протокол H.245 сигнального канала Q.931, а также протокол RAS для регистрации у привратника.

В заключение данного раздела отметим некоторые наиболее существенные различия между стандартами H.323 и SIP.

- Стандарт H.323 представляет собой законченный набор протоколов для проведения мультимедийных конференций. Эти протоколы обеспечивают сигнализацию, регистрацию, управление допуском, транспортировку и кодирование. Стандарт SIP, который описывает только инициирование сеанса и управление им, напротив, представляет единственный протокол. Стандарт SIP работает с протоколом RTP, но не обязывает его использовать. Стандарт SIP работает с речевыми кодеками G.711 и видеокодеками QCIF H.261, но также не обязывает их использовать. Стандарт SIP вполне может функционировать в сочетании с другими протоколами и другими службами.
- Стандарт H.323 разработан международным союзом телекоммуникаций (ITU), а стандарт SIP — проблемной группой разработок для Интернета (IETF), поэтому в нем используются многие концепции, применяющиеся в web, DNS и электронной почте Интернета.
- Стандарт H.323 является всеобъемлющим, поэтому он большой и сложный. В основе стандарта SIP — простота.

Превосходное обсуждение стандартов H.323 и SIP, а также передачи речи по протоколу IP вообще содержится в [207].

За пределами остаточного принципа

В предыдущих разделах мы обсудили вопросы использования в сегодняшних мультимедийных Интернет-приложениях порядковых номеров, меток времени, системы прямого исправления ошибок (FEC), протокола RTP и стандарта H.323. Но достаточно ли этого для поддержания надежных и устойчивых мультимедийных приложений, например IP-телефонии? Прежде чем ответить, вспомним, что сегодняшний Интернет предоставляет всем своим приложениям обслуживание по остаточному принципу, то есть не обеспечивает никаких гарантий качества обслужи-

вания. Приложение будет получать тот уровень производительности (например, значение сквозной задержки и процент потерь), который сеть способна предоставить в данный момент. Вспомним также, что сегодняшний Интернет не позволяет чувствительным к задержке мультимедийным приложениям запрашивать какое-либо специальное обслуживание. Все пакеты обрабатываются на маршрутизаторах одинаково, включая чувствительные к задержке аудио- и видеопакеты. В такой ситуации для того, чтобы качество передаваемой по протоколу IP телефонной связи ухудшилось, в сети достаточно достичь определенного уровня трафика (то есть перегрузки), увеличивающего задержку и долю потерь.

В этом разделе мы обсудим новые архитектурные компоненты, которые могут быть включены в архитектуру Интернета. Эти компоненты защищают приложения от перегрузок и таким образом делают реальным создание высококачественных сетевых мультимедийных приложений. Многие вопросы, которые мы будем рассматривать в этом и в оставшихся разделах этой главы, в последние годы в рамках группы IETF активно обсуждались рабочими группами Diffserv (дифференцированное обслуживание), Intserv (интегрированное обслуживание) и RSVP.

На рис. 6.16 показан простой сценарий, который мы будем использовать для иллюстрации наиболее важных архитектурных компонентов, предложенных для предоставления необходимого мультимедийным приложениям качества обслуживания. Предположим, два потока пакетов формируются хостами H1 и H2 в одной локальной сети и направляются на хосты H3 и H4 в другой локальной сети. Маршрутизаторы двух локальных сетей соединены каналом связи с пропускной способностью 1,5 Мбит/с. Пусть скорости передачи данных в локальных сетях значительно выше, чем 1,5 Мбит/с. Обратите внимание на выходную очередь маршрутизатора R1. Именно в этой очереди возникнут задержки и потери пакетов, если совокупная скорость передачи данных на хостах H1 и H2 превысит 1,5 Мбит/с. Рассмотрим теперь несколько сценариев, каждый из которых позволит нам понять важные принципы предоставления гарантий качества обслуживания мультимедийным приложениям.

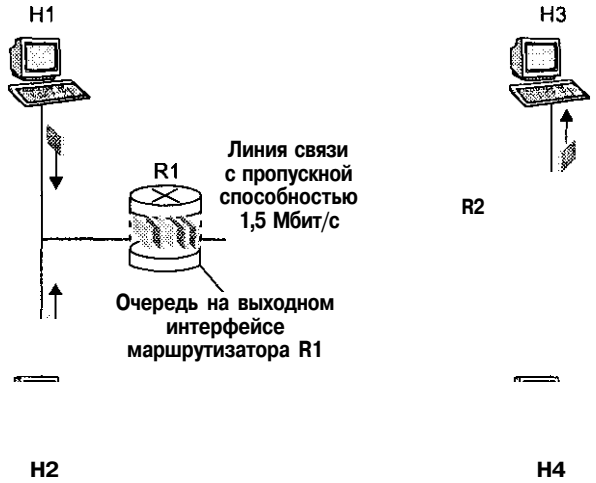


Рис. 6.16. Простая сеть с двумя приложениями

Сценарий 1 — мегабитное аудиоприложение и FTP-приложение

Сценарий 1 иллюстрирует рис. 6.17. Здесь мегабитное аудиоприложение (например, аудиосоединение с качеством компакт-диска) делит линию связи с пропускной способностью 1,5 Мбит/с между маршрутизаторами R1 и R2 с FTP-приложением, выполняющим передачу файла между хостами H2 и H4. В предоставляющем обслуживание по остаточному принципу Интернете аудиопакеты и FTP-пакеты перемешиваются в выходной очереди маршрутизатора R1 и (как правило) передаются в порядке FIFO (First In First Out — первым прибыл, первым обслужен). В этом сценарии при отправке пакетов FTP-источником очередь может заполниться, в результате аудиопакеты будут задерживаться или даже теряться из-за переполнения буфера на маршрутизаторе R1. Как решить проблему? Учитывая, что FTP-приложение не связано временными ограничениями, возможно, маршрутизатор R1 должен предоставить строгий приоритет аудиопакетам. В этом случае аудиопакет в выходном буфере маршрутизатора R1 всегда должен передаваться прежде любого FTP-пакета. Линия связи от маршрутизатора R1 к маршрутизатору R2 будет выглядеть для аудиотрафика как выделенная линия с пропускной способностью 1,5 Мбит/с, а FTP-трафик сможет использовать эту линию, только когда в очереди нет пакетов аудиотрафика.

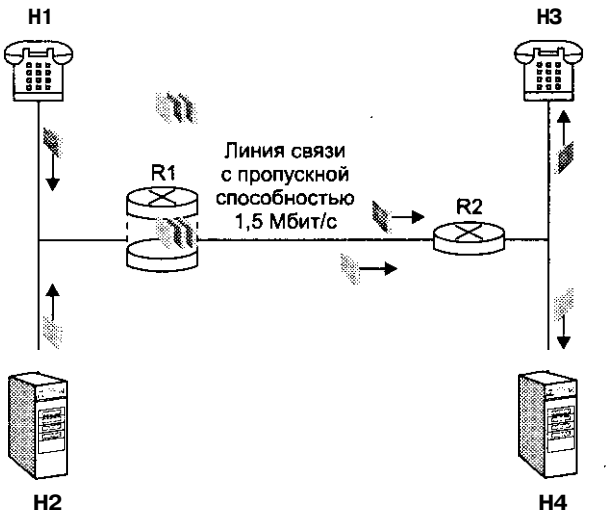


Рис. 6.17. Конкурирующие аудио- и FTP-приложения

Чтобы маршрутизатор R1 мог отличить аудиопакеты от FTP-пакетов, каждый пакет должен быть соответствующим образом промаркирован. Как уже отмечалось в разделе «Протокол IPv6» главы 4, именно для этого изначально предназначалось поле ToS (Type of Service — тип службы) в заголовке пакета IPv4. Таким образом, первый очевидный принцип предоставления гарантий качества обслуживания должен звучать следующим образом.

Принцип 1. *Маркировка пакетов позволяет маршрутизатору различать пакеты, относящиеся к разным классам трафика.*

Сценарий 2 — мегабитное аудиоприложение и высокоприоритетное FTP-приложение

Наш второй сценарий немного отличается от сценария 1. Предположим, что пользователь FTP-приложения заплатил своему Интернет-провайдеру за высококачественный (дорогой) доступ к Интернету, тогда как пользователь аудиоприложения заплатил за Интернет-услуги значительно меньше. Должен ли в этом случае предоставляться приоритет аудиопакетам перед FTP-пакетами? Можно утверждать, что нет. В этом случае более разумным было бы различать пакеты по IP-адресам отправителей. Таким образом, маршрутизатор должен классифицировать пакеты на основании некоторого критерия. В результате мы должны внести в принцип 1 небольшие изменения.

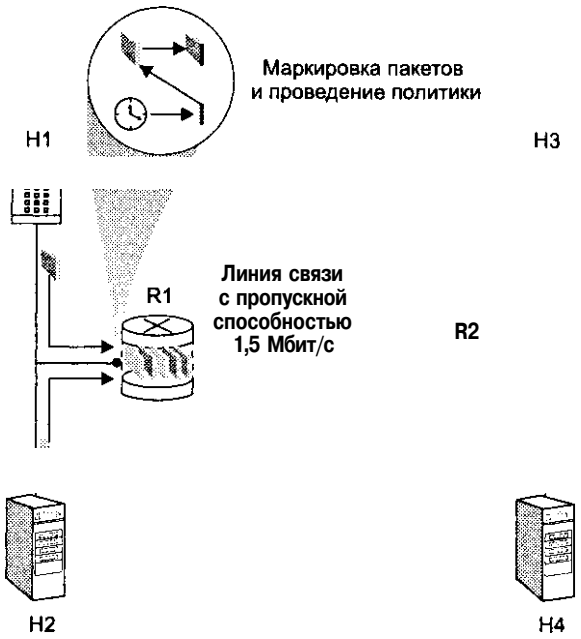
Принцип 1 *(модифицированный). Классификация пакетов позволяет маршрутизатору различать пакеты, относящиеся к разным классам трафика.*

Чтобы различать пакеты, можно использовать явную маркировку. Однако только маркировка не гарантирует, что помеченному пакету будет предоставлен соответствующий уровень обслуживания. Маркировка пакетов представляет собой всего лишь один механизм, помогающий различать пакеты. На основании маркировки, а также других факторов маршрутизатор принимает решение о способе обработки пакета. Это решение зависит от *политики,* которую проводит маршрутизатор.

Сценарий 3 — некачественное аудиоприложение и FTP-приложение

Предположим теперь, что маршрутизатор каким-либо образом узнает (о том, как именно он узнает, мы расскажем в следующих подразделах), что он должен предоставить приоритет аудиоприложению, посылающему поток пакетов со скоростью 1 Мбит/с. Поскольку пропускная способность выходной линии равна 1,5 Мбит/с, FTP-пакеты в среднем получают пропускную способность 0,5 Мбит/с, хотя и имеют более низкий приоритет. Но что произойдет, если аудиоприложение начнет посылать пакеты со скоростью 1,5 Мбит/с или даже еще быстрее (либо злонамеренно, либо из-за ошибки)? В этом случае FTP-приложение не сможет передавать свои пакеты по линии, связывающей маршрутизаторы R1 и R2, так как его пакетам не останется пропускной способности и они не получат никакого обслуживания. Похожие проблемы могут возникнуть, если несколько приложений (например, несколько телефонных звонков) с одинаковыми приоритетами пользуются одной общей линией связи. В этом случае один «невоспитанный» поток может радикально снизить производительность остальных потоков. В идеальном случае желательно в определенной степени *изолировать* потоки друг от друга, чтобы защитить от некорректно работающих потоков. В этом состоит второй основополагающий принцип предоставления гарантий качества обслуживания.

Принцип 2. *Желательно обеспечить определенную степень изоляции потоков друг от друга, так чтобы некачественный поток не мог помешать работе других потоков.*



Условные обозначения:

Измерение и управление

|| **Маркировка**

Рис. 6.18. Управление потоками аудио- и FTP-приложений

В следующих подразделах мы исследуем некоторые специфические механизмы обеспечения подобной изоляции. Для этого могут использоваться два общих подхода. Во-первых, к потокам можно применить «политические» меры, как показано на рис. 6.18. Если поток трафика должен удовлетворять определенным критериям (например, пиковая скорость аудиопотока не должна превышать 1 Мбит/с), то, чтобы гарантировать соблюдение этого правила, может применяться специальный механизм проведения политики в жизнь. Если приложение, к которому применяется политика, ведет себя некорректно, данный механизм предпринимает определенные действия (например, отбрасывает или задерживает пакеты, нарушающие заданный критерий), чтобы попадающий в сеть трафик удовлетворял критерию. Возможно, наиболее распространенным механизмом проведения политики является алгоритм «дырявого ведра», который мы рассмотрим в следующем разделе. На рис. 6.18 механизм классификации и маркировки пакетов (принцип 1) и механизм проведения политики

(принцип 2) реализованы на периферии сети (на оконечной системе или на периферийном маршрутизаторе).

Альтернативный метод обеспечения изоляции потоков трафика заключается в том, чтобы механизм планирования пакетов канального уровня явно выделял фиксированные порции пропускной способности каждому прикладному потоку. Например, аудиопотоку на маршрутизаторе R1 может быть выделена пропускная способность 1 Мбит/с, а FTP-поток может получить 0,5 Мбит/с. В этом случае аудиопоток и FTP-поток в одном и том же канале получат по логической выделенной линии каждый с пропускной способностью 1 Мбит/с и 0,5 Мбит/с соответственно, как показано на рис. 6.19.

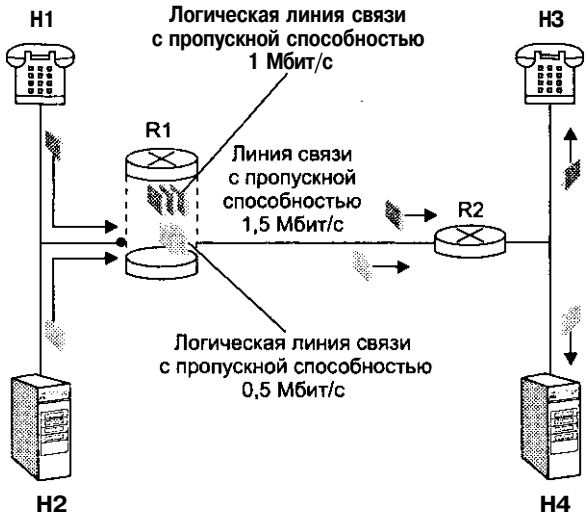


Рис. 6.19. Логическая изоляция потоков аудио- и FTP-приложений

При строгом распределении пропускной способности на уровне передачи данных поток может использовать только ту часть пропускной способности, которая ему предоставлена. В частности, поток не может расходовать пропускную способность других приложений. Например, если аудиопоток временно не передает данные (например, абонент замолкает, и аудиоприложение перестает генерировать пакеты аудиоданных), FTP-поток все равно не сможет передать по линии, связывающей маршрутизаторы R1 и R2, более 0,5 Мбит/с своих данных, хотя выделенная аудиоприложению логическая линия с пропускной способностью 1 Мбит/с в данный момент свободна. То есть желательно использовать пропускную способность по возможности эффективно, позволяя одному потоку в любой момент времени задействовать свободную часть пропускной способности другого потока. В этом состоит третий принцип предоставления качества обслуживания.

> ***Принцип 3.*** *Обеспечивая изоляцию потоков друг от друга, желательно максимально эффективно использовать ресурсы (например, пропускную способность линий связи и буферы).*

Сценарий 4 — два мегабитных аудиоприложения на одной линии

В нашем последнем сценарии два аудиоприложения передают свои пакеты со скоростью 1 Мбит/с по одной линии с пропускной способностью 1,5 Мбит/с (рис. 6.20). В этом случае суммарная скорость передачи данных в двух потоках (2 Мбит/с) превышает пропускную способность линии. Даже классификация и маркировка (принцип 1), изоляция потоков (принцип 2) и совместное использование свободной пропускной способности (принцип 3), которой в данной ситуации нет, не помогут. Линия просто не обладает пропускной способностью, необходимой для удовлетворения потребностей обоих приложений. Если разделить ресурсы (пропускную способность) между приложениями поровну, то каждое приложение получит лишь 0,75 Мбит/с и будет терять по 25 % передаваемых пакетов. В результате качество связи окажется настолько низким, что пользоваться приложениями будет просто невозможно, и передача остальных 75 % пакетов потеряет смысл.

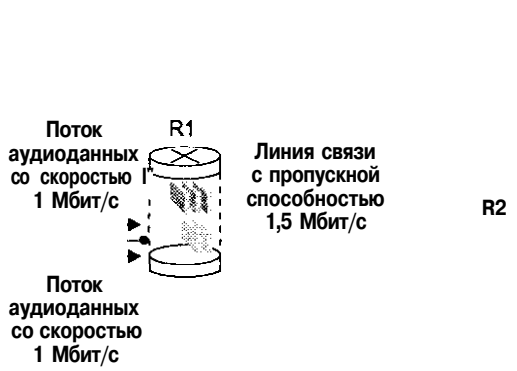


Рис. 6.20. Конкурирующие аудиопотоки перегружают линию

Если потоку, чтобы считаться «полезным», необходим некоторый минимальный уровень обслуживания, тогда сеть должна либо позволить ему передаваться, либо его *блокировать*. Примером подобной сети является телефонная сеть — если требуемые ресурсы (линии связи в случае телефонной сети) не могут быть выделены соединению, соединение блокируется (не допускается в сеть), а пользователю возвращается сигнал «занято». В нашем примере нет смысла предоставлять потоку доступ к сети, если он не получит достаточного уровня качества обслуживания, чтобы считаться «полезным».

Очевидно, чтобы предоставить потоку гарантированный уровень обслуживания, поток должен объявить о своих потребностях. Процедура объявления потоком своих требований к уровню обслуживания и принятия сетью решения о предоставлении потоку доступа в сеть называется *допуском* (admission). Необходимость в про-

цедуре допуска и составляет четвертый принцип предоставления гарантий качества обслуживания.

> **Принцип 4.** *Необходима процедура допуска, в которой потоки объявляют о своих требованиях к качеству обслуживания, а затем либо получают доступ в сеть (с требуемым уровнем качества обслуживания), либо блокируются (если требуемый уровень качества обслуживания не может быть предоставлен сетью).*

В приведенном выше обсуждении мы определили четыре основных принципа предоставления гарантий качества обслуживания мультимедийным приложениям. В следующем разделе мы рассмотрим различные механизмы реализации этих принципов. Затем мы изучим предлагаемые в Интернете модели обслуживания с предоставлением гарантий качества.

Механизмы проведения политики и планирования

В предыдущем разделе мы определили важные основополагающие принципы предоставления сетевым мультимедийным приложениям гарантий качества обслуживания; теперь мы рассмотрим, что для этого нужно.

Механизмы планирования

В разделах «Задержки и потери данных в сетях с коммутацией пакетов» главы 1 и «Устройство маршрутизатора» главы 4 отмечалось, что пакеты, относящиеся к разным сетевым потокам, мультиплексируются и ставятся в очередь на передачу в выходных буферах. Алгоритм выбора стоящего в очереди пакета для передачи его по линии называется *дисциплиной очередей*. В предыдущем разделе было показано, что дисциплина очередей играет важную роль при предоставлении гарантий качества обслуживания. Рассмотрим теперь некоторые наиболее важные дисциплины очередей более детально.

Дисциплина FIFO

На рис. 6.21 представлена абстрактная модель дисциплины очередей FIFO (First In First Out — первым прибыл, первым обслужен). Пакеты, прибывающие в выходной буфер линии, ставятся в очередь на передачу, если линия занята передачей другого пакета. Если буферного пространства для размещения пакета недостаточно, один из пакетов (вновь прибывший или уже находящийся в буфере) должен быть удален из буфера. Выбором такого пакета занимается алгоритм, называемый *политикой отбрасывания пакета*. В нашем последующем обсуждении мы будем игнорировать отбрасывание пакетов (тем не менее не забывайте о такой возможности — см. подраздел «Очереди» в разделе «Устройство маршрутизатора» главы 4). Когда завершается передача пакета в исходящую линию (то есть пакет получает обслуживание), он удаляется из очереди.

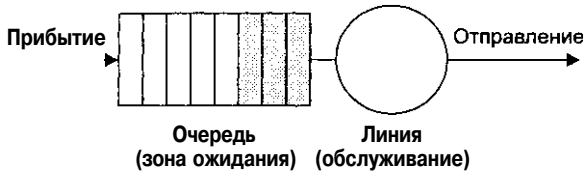


Рис. 6.21. Дисциплина очередей FIFO

В соответствии с дисциплиной очередей FIFO пакеты выбираются для передачи их по линии в том же порядке, в котором они поступили в очередь. Мы все встречались с этой дисциплиной очередей на автобусных остановках (особенно в Англии, где искусство стояния в очередях, кажется, доведено до совершенства) и на других предприятиях сферы обслуживания, где прибывающие клиенты встают в хвост очереди и обслуживаются в порядке прибытия в очередь.

На рис. 6.22 показан пример очереди FIFO в действии. Прибытия пакетов показаны стрелками в верхней части рисунка. Номера определяют порядок, в котором прибыли пакеты. Покидающие очередь пакеты расположены в нижней части рисунка. Периоды времени обслуживания (передачи) пакетов имеют вид прямоугольников (в средней части рисунка). Благодаря дисциплине FIFO пакеты покидают очередь в том же порядке, в котором они в нее попали. Обратите внимание, после того как очередь покидает пакет 4, линия остается свободной (так как пакеты 1-4 уже переданы по линии и покинули очередь) до тех пор, пока не прибудет пакет 5.

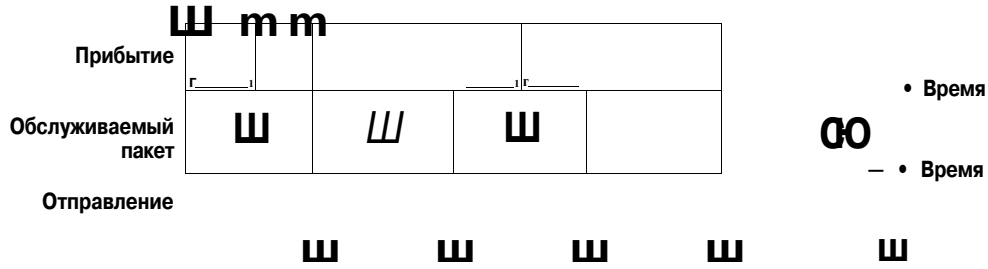


Рис. 6.22. Очередь FIFO в действии

Приоритетная очередь

При использовании приоритетной дисциплины очередей прибывающие пакеты разбиваются на несколько классов приоритетов, как показано на рис. 6.23. В предыдущем разделе отмечалось, что класс приоритетов может зависеть от явной маркировки в заголовке пакета (например, от значения поля типа службы в пакете протокола IPv4), IP-адреса отправителя или получателя, номера порта получателя и других параметров. У каждого класса приоритетов есть своя очередь. При выборе пакета для передачи алгоритм приоритетной дисциплины очередей выбирает пакет с наивысшим приоритетом. Выбор пакета одного класса приоритета, как правило, осуществляется в соответствии с дисциплиной FIFO.

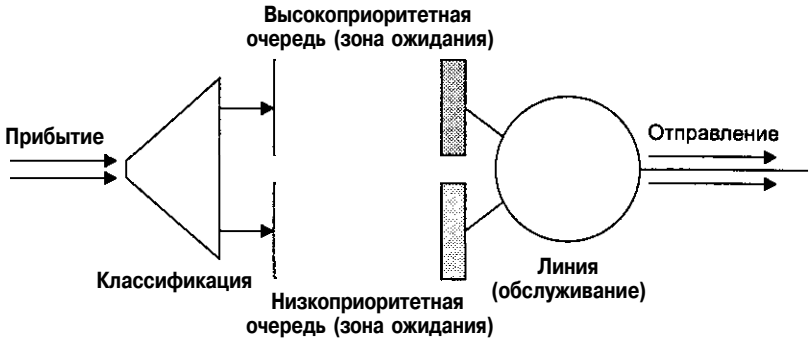


Рис. 6.23. Модель приоритетной очереди

Рисунок 6.24 иллюстрирует работу приоритетной очереди с двумя классами приоритетов. Пакеты 1, 3 и 4 относятся к высокоприоритетному классу, а пакеты 2 и 5 — к низкоприоритетному. Пакет 1 прибывает и, поскольку линия свободна, сразу же передается в линию. Во время передачи первого пакета прибывают пакеты 2 и 3, которые попадают в низкоприоритетную и высокоприоритетную очереди соответственно. После передачи пакета 1 для передачи выбирается пакет 3 (высокоприоритетный). Пакет 2, как низкоприоритетный, должен ждать, хотя он и прибыл раньше пакета 3. После пакета 3 передается пакет 2. Во время передачи низкоприоритетного пакета 2 прибывает пакет 4 (высокоприоритетный). При использовании так называемой дисциплины очереди без прерывания обслуживания передача пакета не прерывается. В этом случае пакет 4 устанавливается в очередь и его передача начинается, когда завершается передача пакета 2.

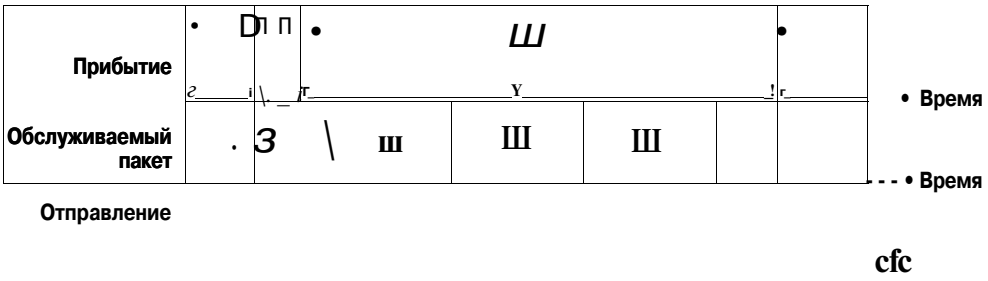


Рис. 6.24. Работа приоритетной очереди

Циклическая дисциплина очередей и дисциплина очередей WFQ

При использовании циклической дисциплины очередей пакеты так же сортируются по классам, как и в случае приоритетной дисциплины очередей, однако в первом случае обслуживание поочередно предоставляется пакетам различных классов. В простейшей форме циклической дисциплины очередей сначала передается пакет класса 1, затем пакет класса 2, затем опять пакет класса 1 и т. д. Если паке-

тов определенного класса нет, чтобы линия не простаивала, происходит немедленное переключение на следующий класс пакетов. Подобные дисциплины очередей, не допускающие простоя линии при наличии пакетов в очереди, называют *дисциплинами очередей у поддерживающими рабочий режим*.

Рисунок 6.25 иллюстрирует работу циклической очереди с двумя классами. В данном примере пакеты 1, 2 и 4 принадлежат классу 1, а пакеты 3 и 5 — классу 2. Передача пакета 1 начинается сразу, как только пакет прибывает в выходную очередь. Пока идет передача пакета 1, прибывают пакеты 2 и 3 и становятся в очередь. После передачи пакета 1 планировщик ищет пакет класса 2 и поэтому передает пакет 3. Затем планировщик ищет пакет класса 1 и поэтому передает пакет 2. После передачи пакета 2 в очереди остается только пакет 4, который и передается сразу после пакета 2.

В архитектурах, предоставляющих гарантии качества обслуживания, нашла применение обобщенная абстракция циклической очереди, называемая *взвешенной справедливой организацией очередей* (Weighted Fair Queuing, WFQ) [117, 368]. Дисциплину WFQ иллюстрирует рис. 6.26. Прибывающие пакеты классифицируются и ставятся в очередь, соответствующую их классу. Как и при циклической дисциплине очередей, WFQ-планировщик обслуживает классы по очереди: сначала обслуживается класс 1, затем класс 2, потом класс 3, а затем (предполагая, что всего классов три) все повторяется снова. Дисциплина WFQ также является дисциплиной очередей, поддерживающей рабочий режим, поэтому, обнаружив, что очередь какого-либо класса пуста, она сразу же переключается на следующий класс.

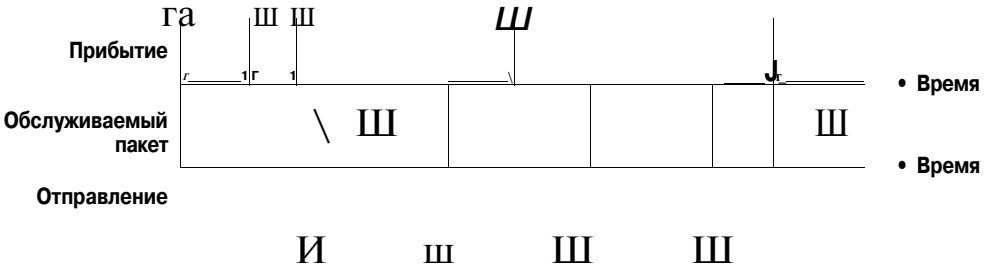


Рис. 6.25. Работа циклической очереди с двумя классами

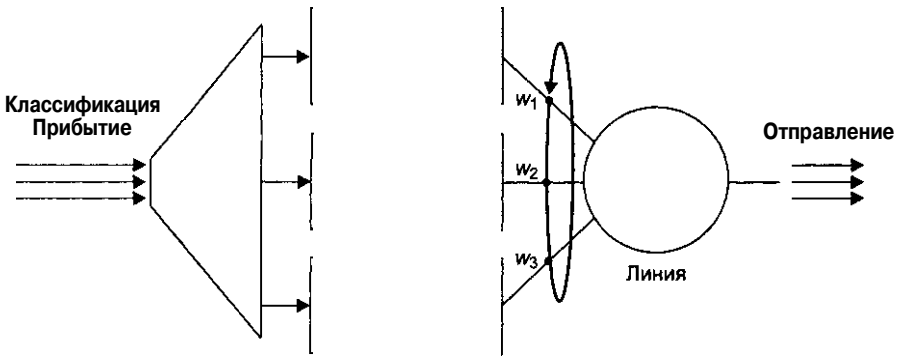


Рис. 6.26. Дисциплина очередей WFQ

От циклической дисциплины очередей дисциплина WFQ отличается тем, что каждый класс получает разный объем услуг. Каждому классу *г* назначается весовой коэффициент w^{i}. В дисциплине очередей WFQ в течение любого интервала времени, когда у класса *г* есть пакеты для передачи, классу *i* будет предоставлена гарантированная доля обслуживания, равная **2^),** где сумма в делителе вычисляется по всем классам, у которых также есть пакеты для передачи. В худшем случае, когда в очереди есть пакеты всех классов, классу *i* будет предоставлена гарантированная доля, равная $w^{i}/(\ ^wj)$ от суммарной пропускной способности. Таким образом, для линии с пропускной способностью *R* классу *г* будет предоставлена, по меньшей мере, пропускная способность R^{x} *Щ/(^j^wj)*- Мы привели здесь идеализированное описание дисциплины очередей WFQ, не приняв во внимание тот факт, что пакеты представляют собой неделимые блоки данных и нельзя прерывать передачу одного пакета, чтобы начать передачу другого. Данный вопрос обсуждается в [117, 368]. Как будет показано в следующих разделах, дисциплина очередей WFQ играет центральную роль в архитектурах, предоставляющих гарантии качества обслуживания. Дисциплина очередей WFQ также применяется в современных маршрутизаторах [76]. (Таким образом, гарантии качества обслуживания могут предоставляться внутренним потокам в интрасетях, маршрутизаторы которых поддерживают дисциплину очередей WFQ.)

Политика дырявого ведра

В разделе «За пределами остаточного принципа» мы сказали, что проведение политики заключается в регулировании скорости, с которой потоку разрешается посылать пакеты в сеть. Политика — это краеугольный камень любой архитектуры качества обслуживания. Но какие аспекты скорости должны регулироваться? Мы можем идентифицировать три важных критерия регулирования, каждый из которых отличается от других масштабом времени.

- *Средняя скорость.* Сеть может ограничить долгосрочную среднюю скорость (количество пакетов за интервал времени), с которой поток пакетов направляется в сеть. Ключевым вопросом здесь является интервал времени усреднения скорости. Поток, средняя скорость которого ограничена величиной 100 пакетов в секунду, оказывается в более строгих рамках, чем поток, средняя скорость которого равна 6000 пакетов в минуту, хотя может показаться, что их средние скорости одинаковы. Например, во втором случае поток может передать в течение одной секунды даже 1000 пакетов (при условии, что в течение минутного интервала количество отправленных пакетов не превысит 6000), тогда как первому потоку сеть этого не разрешает.
- *Пиковая скорость.* В то время как средняя скорость определяется на относительно долгом интервале времени, пиковая скорость представляет собой максимальное количество пакетов, которое может быть отправлено в сеть за более короткий период времени. В приведенном выше примере сеть может ограничить среднюю скорость значением 6000 пакетов в минуту, а пиковую скорость — значением 1500 пакетов в секунду.

- *Размер импульса.* Сеть может также пожелать установить ограничение на максимальное число пакетов («импульс» пакетов), которые могут быть переданы в сеть за крайне короткий интервал времени.

Алгоритм дырявого ведра представляет собой абстракцию, которую можно использовать для описания этих ограничений. Как показано на рис. 6.27, дырявое ведро способно вместить ***b*** маркеров. Каждую секунду в ведро поступают ***r*** новых маркеров. (Для простоты мы примем за единицу времени одну секунду.) Если ведро целиком не заполнено, в него могут добавляться новые маркеры. В противном случае новые маркеры игнорируются.

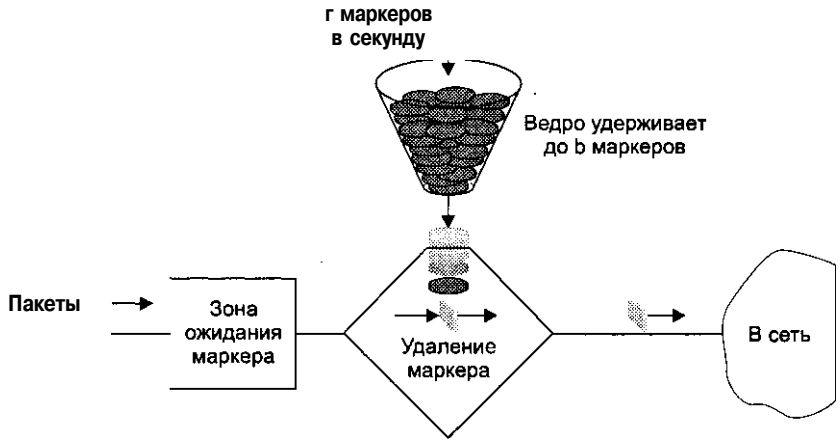


Рис. 6.27. Регулирование потока при помощи дырявого ведра

Посмотрим теперь, как дырявое ведро может использоваться для регулирования потока пакетов. Пусть для того, чтобы переслать пакет в сеть, из ведра должен быть сначала удален маркер. Если ведро пустое, пакет должен ждать маркера. (Альтернативный подход заключается в отбрасывании пакета, для которого нет маркера, но мы не станем рассматривать такой вариант.) Посмотрим теперь, как такое поведение будет регулировать поток пакетов. Поскольку в ведре может быть не более ***b*** маркеров, максимальный размер импульса пакетов также будет равен ***b*** пакетов. Кроме того, поскольку скорость генерирования маркеров равна r, максимальное количество пакетов, которые могут быть переданы в сеть в течение любого интервала времени ***t,*** равно ***rt + b.*** Таким образом, скорость генерации маркеров r служит ограничителем долгосрочной средней скорости, с которой пакеты могут поступать в сеть. Дырявое ведро (два последовательно соединенных ведра) также может использоваться для ограничения пиковой скорости (см. упражнения в конце этой главы).

В разделах «Интегрированное обслуживание» и «Дифференцированное обслуживание» мы рассмотрим интегрированное и дифференцированное обслуживание, являющееся, по сути, методами предоставления гарантий качества обслуживания в Интернете. Как мы увидим, алгоритм дырявого ведра и дисциплина очередей

WFQ могут играть в этом важную роль. Рассмотрим выход маршрутизатора, работающего в соответствии с дисциплиной очередей WFQ, при мультиплексировании *n* потоков, каждый из которых регулируется алгоритмом дырявого ведра с параметрами *Ь{* и г„ *г* = 1,..., *n*. Здесь мы будем использовать термин «поток» в широком смысле для обозначения набора пакетов, не отличаемых друг от друга планировщиком. На практике поток может состоять из трафика, относящегося к одному сквозному соединению (например, в случае интегрированного обслуживания) или к множеству таких соединений (как в случае дифференцированного обслуживания). Мультиплексирование потоков, регулируемых алгоритмами дырявого ведра, иллюстрирует рис. 6.28.

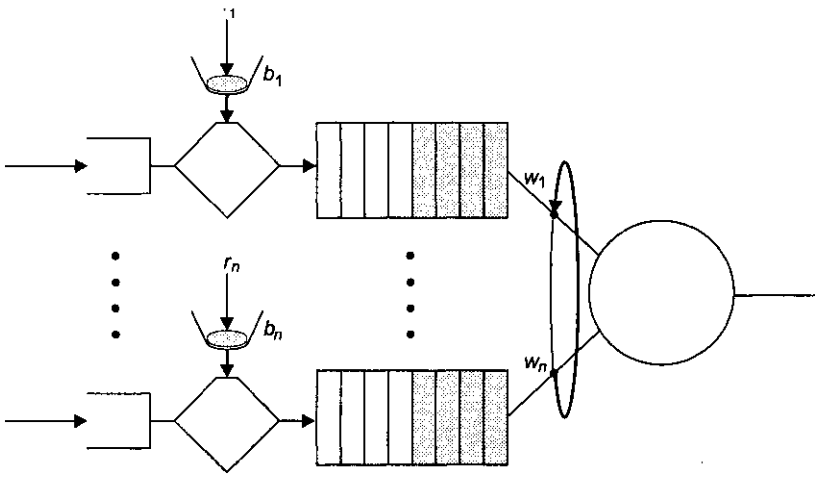


Рис. 6.28. Мультиплексирование потоков

При обсуждении дисциплины очередей WFQ отмечалось, что каждому потоку *г* гарантируется доля пропускной способности, равная, по меньшей мере, *RwJ{ ^щ),* где *R* представляет собой скорость передачи данных по линии в пакетах в секунду. Чему в этом случае равна максимальная задержка пакета, ожидающего обслуживания в очереди с дисциплиной WFQ (после того как пакет пройдет через дырявое ведро)? Рассмотрим поток 1. Предположим, ведро потока 1 изначально полно маркеров. Затем в него прибывает импульс из *Ь{* пакетов. Эти пакеты (без ожидания) удаляют все маркеры из ведра и помещаются в зону ожидания WFQ для потока 1. Поскольку эти *Ь{* пакетов обслуживаются на скорости, по меньшей мере, равной *Rw{/{* пакетов в секунду, последние из этих пакетов будут задержаны на максимальное время

$$d_{u\,max} = \frac{2}{\text{п}\quad N^*}$$

$$R\text{-}W_{(}/2,Wj$$

Смысл этой формулы в том, что при наличии в очереди *Ь{* пакетов, обслуживаемых с минимальной скоростью *RwJ(^щ)* пакетов в секунду, время, необходи-

мое для того, чтобы передать все эти пакеты полностью, не может превышать bj R x $Wj/(\hat{}\ W\ j\ )$. В упражнениях, находящихся в конце этой главы, вам предлагается доказать, что пока i \ < R x з н а ч е н и е d^{max} действительно представляет собой максимальную задержку в очереди WFQ любого пакета из потока 1.

Интегрированное обслуживание

В предыдущих разделах мы рассмотрели принципы и механизмы, используемые для предоставления гарантий качества обслуживания в Интернете. В этом разделе мы поговорим о том, как эти идеи реализуются в специальной архитектуре предоставления гарантий качества обслуживания в Интернете — так называемой, архитектуре *интегрированного обслуживания* (Integrated Services, Intserv). Интегрированное обслуживание представляет собой структуру, разработанную группой IETF и служащую для предоставления гарантий качества обслуживания индивидуальным прикладным сеансам. Сердцевиной архитектуры интегрированного обслуживания являются два ключевых компонента.

- *Зарезервированные ресурсы.* Маршрутизатор должен знать, какое количество его ресурсов (буферов, пропускной способности) уже зарезервировано для текущих сеансов.
- *Установка соединения.* Сеанс, которому требуются гарантии качества обслуживания, должен сначала зарезервировать достаточное количество ресурсов на каждом сетевом маршрутизаторе вдоль пути от отправителя к получателю, чтобы гарантировать требуемый уровень обслуживания. В этом процессе установки соединения должны принимать участие все маршрутизаторы вдоль маршрута. Каждый маршрутизатор должен определить, какие ресурсы требуются для устанавливаемого сеанса, учитывая ресурсы, уже занятые другими текущими сеансами. Кроме того, маршрутизатор должен определить, достаточно ли у него ресурсов для того, чтобы удовлетворить требования нового сеанса, не нарушив гарантии, предоставленные уже открытым сеансам.

Процесс установки соединения иллюстрирует рис. 6.29. Рассмотрим этапы этого процесса более подробно.

1. *Описание трафика и спецификация требуемого качества обслуживания.* Чтобы маршрутизатор мог определить, достаточно ли у него ресурсов для удовлетворения требований качества обслуживания нового сеанса, этот сеанс должен сначала объявить свои требования, а также описать трафик, который он будет посылать в сеть и для которого он запрашивает гарантии качества обслуживания. В архитектуре интегрированного обслуживания требуемый уровень качества описывается в так называемой спецификации Rspec (R означает Reservation — резервирование), а трафик, который будет посылаться в сеть (или приниматься из сети), описывается в спецификации Tspec (T означает Traffic — трафик). Конкретная форма спецификаций Rspec и Tspec может варьироваться в зависимости от запрашиваемого обслуживания, о чем будет рассказано далее. Спецификации Rspec и Tspec определены в RFC 2210 и RFC 2215.

2. *Выполнение сигнальной процедуры перед установкой соединения.* Спецификации Rspec и Tspec переправляются всем тем маршрутизаторам, на которых должны резервироваться ресурсы для сеанса. В настоящий момент для этой цели в Интернете используется сигнальный протокол RSVP, который будет подробно освещаться в следующем разделе. Применение протокола RSVP в архитектуре интегрированного обслуживания описано в RFC 2210.
3. *Поочередная установка соединения.* Как только маршрутизатор получает спецификации Rspec и Tspec для сеанса, запрашивающего гарантии качества обслуживания, он определяет, сможет ли удовлетворить эти запросы. Принимаемое маршрутизатором решение зависит от спецификации трафика, запрашиваемого типа службы и объема ресурсов, уже предоставленных другим сеансам. Процесс поочередной установки соединения иллюстрирует рис. 6.30.

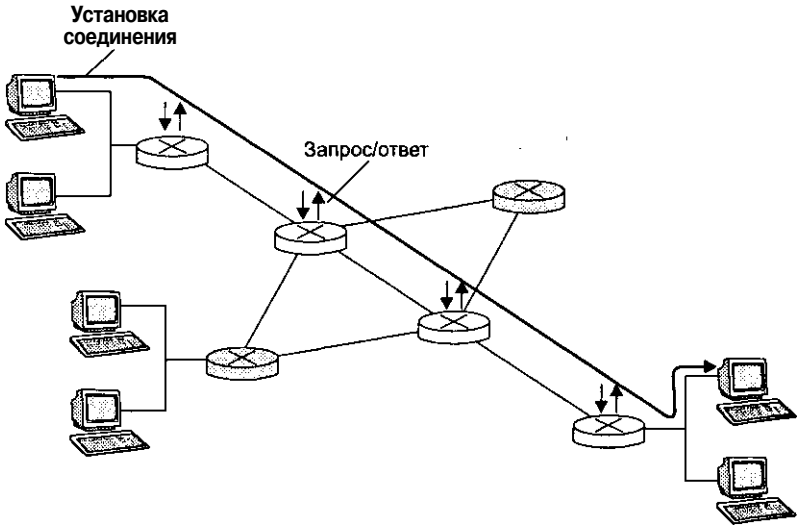


Рис. 6.29. Общий процесс установки соединения

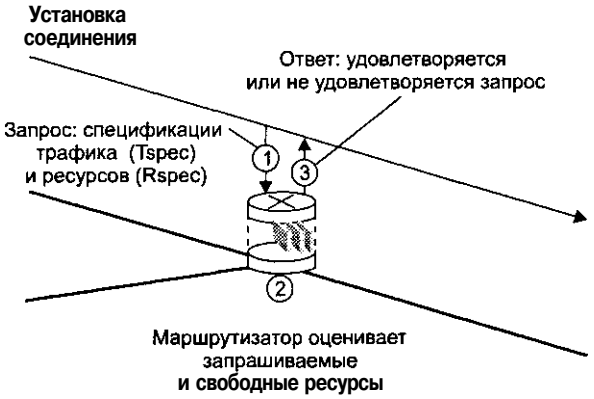


Рис. 6.30. Установка соединения на одном из маршрутизаторов

В архитектуре интегрированного обслуживания определяются два основных класса обслуживания: гарантированное обслуживание и обслуживание с контролируемой нагрузкой. Вскоре мы увидим, что каждый из этих вариантов обслуживания предоставляет совершенно разные гарантии качества.

Гарантированное качество обслуживания

Спецификация гарантированного обслуживания, определенная в RFC 2212, обеспечивает жесткие (математически проверяемые) границы задержки на ожидание пакетов в очереди маршрутизатора. Хотя детали гарантированного обслуживания довольно сложны, основная идея проста. В первом приближении описание трафика источника задается параметрами (г, *Ь)* алгоритма дырявого ведра (см. раздел «Механизмы проведения политики и планирования»), а запрашиваемое обслуживание характеризуется скоростью *R,* с которой могут передаваться пакеты. По существу, сеанс, запрашивающий гарантированное обслуживание, требует, чтобы биты в его пакете были гарантированно переданы со скоростью *R* бит/с. Учитывая, что трафик описывается при помощи алгоритма дырявого ведра, и запрашивается гарантированная скорость передачи данных *R* бит/с, можно ограничить максимальную задержку ожидания в очереди на маршрутизаторе. Вспомним, что при описании трафика с помощью алгоритма дырявого ведра объем трафика (в битах), генерируемый за любой произвольный интервал времени *t,* ограничен сверху величиной $rt + b$. Также вспомним, что, когда источник дырявого ведра направляется в очередь, гарантирующую, что поступающий в нее трафик будет обслужен со скоростью не ниже *R* бит/с, то максимальная задержка любого пакета не будет превышать b/R при условии, что *R* больше, чем г. Фактическое граничное значение задержки в определении гарантированного обслуживания несколько сложнее, что вызвано эффектами пакетирования (граничное значение вида b/R предполагает, что данные представляют собой непрерывный поток, а не дискретные пакеты) и тем фактом, что процесс поступления трафика зависит от ограничения пиковой скорости на входной линии (граничное значение вида b/R предполагает, что импульс из *b* бит может прибыть мгновенно), а также возможными дополнительными изменениями времени передачи пакета.

Обслуживание с контролируемой нагрузкой

Сеанс, запросивший обслуживание с контролируемой нагрузкой, получает «качество обслуживания, приближенное к тому, которое поток получал бы от ненагруженного сетевого элемента» (RFC 2211). Другими словами, сеанс может рассчитывать, что «очень высокий процент» его пакетов успешно пройдет через маршрутизатор и не будет отброшен, а задержка на маршрутизаторе окажется близкой к нулю. Интересно отметить, что обслуживание с контролируемой нагрузкой не дает численных гарантий относительно производительности — оно не указывает, что означает «очень высокий процент» пакетов, а также какое качество обслуживания можно считать близким к «получаемому потоком от ненагруженного сетевого элемента».

Обслуживание с контролируемой нагрузкой предназначено для мультимедийных приложений реального времени, разрабатывавшихся для сегодняшнего Интернета.

Как мы видели, эти приложения прекрасно работают в ненагруженной сети, но быстро теряют производительность при возрастании нагрузки.

Протокол RSVP

Как мы выяснили в разделе «Интегрированное обслуживание», чтобы сеть могла предоставлять гарантии качества обслуживания, нужен сигнальный протокол, позволяющий работающим на хостах приложениям резервировать ресурсы в Интернете. Таким протоколом является *RSVP* (ReSerVation Protocol — протокол резервирования) [40,565].

Когда идет речь о *ресурсах* в контексте Интернета, как правило, имеется в виду пропускная способность линий и буферы маршрутизаторов. Однако чтобы наша дискуссия была предметной, будем считать, что *ресурс* является синонимом *пропускной способности,* а протокол RSVP представляет собой протокол резервирования пропускной способности.

Сущность протокола RSVP

Протокол RSVP позволяет приложениям резервировать пропускную способность для своих потоков данных. Этим протоколом от имени потока данных приложения пользуется хост, запрашивая у сети определенный объем пропускной способности. Протокол RSVP также используется маршрутизаторами для пересылки запросов о резервировании ресурсов. Чтобы протокол RSVP мог работать, соответствующее программное обеспечение должно быть установлено у отправителей, получателей и маршрутизаторов. Протокол RSVP обладает двумя основными свойствами.

- Он обеспечивает *резервирование пропускной способности в деревьях групповой рассылки* (выборочная рассылка обрабатывается как вырожденный случай групповой рассылки).
- Он *ориентирован на получателя',* то есть резервирование ресурсов инициируется и управляется получателем потока данных.

Эти два свойства иллюстрирует рис. 6.31. На рисунке показано дерево групповой рассылки и данные, передаваемые от вершины дерева хостам. Хотя потоки данных начинаются у отправителя, запросы на резервирование отправляются получателями. Когда маршрутизатор переправляет запросы на резервирование вверх по дереву отправителю потока данных, он может объединять эти запросы в общие сообщения.

Прежде чем перейти к более детальному обсуждению протокола RSVP, нам необходимо еще раз рассмотреть понятие *сеанса.* Как и в протоколе RTP, в сеансе может объединяться несколько потоков данных. Каждый отправитель в сеансе представляет собой источник одного или нескольких потоков данных; например, отправитель может быть источником потока видеоданных и потока аудиоданных. У каждого потока данных в сеансе должен быть один и тот же групповой адрес. Чтобы наше обсуждение было более предметным, будем предполагать, что маршрутизаторы

и хосты идентифицируют сеанс, которому принадлежат пакеты, по групповому адресу пакета. (В действительности спецификация протокола RSVP позволяет применять для идентификации сеанса более обобщенные методы.) Внутри сеанса также необходимо идентифицировать поток данных, к которому принадлежит пакет. Это может быть сделано, например, при помощи поля идентификатора потока в заголовке пакета формата IPv6.

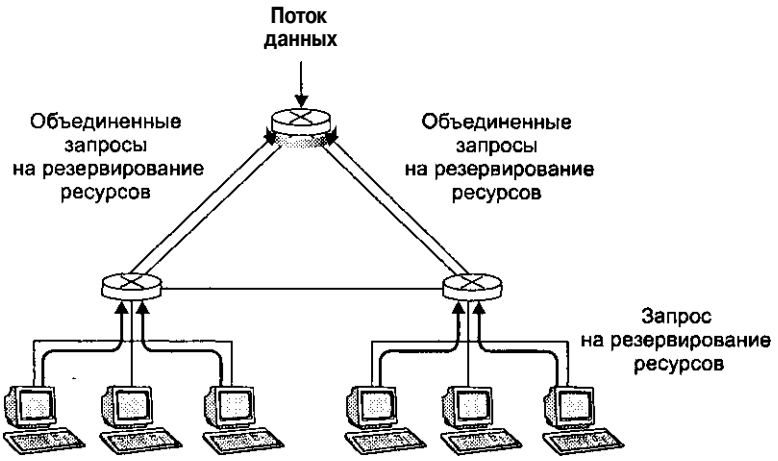


Рис. 6.31. Протокол RSVP: групповая рассылка и ориентация на получателя

Что не определяет стандарт RSVP

Обратите внимание, что стандарт RSVP (RFC 2205) не определяет, каким образом сеть предоставляет потокам данных зарезервированную пропускную способность. Это всего лишь протокол, позволяющий приложениям резервировать пропускную способность линий связи. Как только пропускная способность зарезервирована, предоставлением этой пропускной способности потокам данных занимаются маршрутизаторы. Как правило, эта задача решается с помощью механизмов планирования (приоритетного планирования, взвешенной справедливой организации очередей и т. д.), обсуждавшихся в разделе «Механизмы проведения политики и планирования».

Также важно понимать, что протокол RSVP не является протоколом маршрутизации — он не занимается определением линий, на которых должна быть зарезервирована пропускная способность. Этим занимается протокол маршрутизации (выборочной или групповой). После того как протокол маршрутизации выбрал линии для передачи новых потоков, протокол RSVP может зарезервировать пропускную способность в линиях вдоль этих маршрутов. (Вскоре мы увидим, что при изменении этих маршрутов протокол RSVP производит резервирование ресурсов заново.) Когда ресурсы зарезервированы, планировщики пакетов на маршрутизаторах должны предоставить потокам данных зарезервированную пропускную способность. Таким образом, протокол RSVP представляет собой всего лишь часть — хотя и важную — механизма предоставления гарантий качества обслуживания.

Протокол RSVP иногда называют *сигнальным протоколом*. При этом имеется в виду, что RSVP позволяет хостам запрашивать для потоков данных ресурсы и возвращать зарезервированные ресурсы в сеть. Термин «сигнальный» обязан своим происхождением телефонным компаниям.

Неоднородность получателей

Рассмотрим сеть, в которой одни получатели могут получать потоки со скоростью передачи данных 28,8 Кбит/с, другие — со скоростью 128 Кбит/с, а третьи — со скоростью 10 Мбит/с или более. Эта неоднородность получателей ставит интересную проблему. Если отправитель пересылает видеоданные группе разнородных получателей, должен ли он генерировать видеопоток низкого качества для скорости 28,8 Кбит/с, видеопоток среднего качества для скорости 128 Кбит/с или только видеопоток высокого качества для скорости 10 Мбит/с? Если видеоданные кодируются только на скорости 10 Мбит/с, тогда его смогут принимать лишь пользователи с 10-мегабитным доступом к сети. С другой стороны, если кодировать видео для передачи со скоростью 28,8 Кбит/с, тогда обладатели 10-мегабитного доступа будут видеть изображение очень низкого качества, хотя могли бы получать нечто существенно лучшее.

Для разрешения этой дилеммы часто предполагается, что видео- и аудиоданные кодируются по уровням. Например, видеоданные могут кодироваться по двум уровням: базовому и дополнительному. Базовый уровень может передаваться со скоростью 20 Кбит/с, тогда как дополнительный уровень может иметь скорость передачи данных, равную 100 Кбит/с. В этом случае получатель с 28-килобитным модемом сможет принимать 20-килобитный поток, тогда как обладатели 128-килобитного доступа смогут принимать оба уровня и строить из них высококачественное изображение.

Обратите внимание, что отправитель не должен знать скоростей доступа всех получателей. Ему нужно лишь знать максимальную скорость всех получателей. Отправитель кодирует видео- и аудиоданные по нескольким уровням и посылает их по дереву групповой рассылки. Получатели выбирают уровни, соответствующие их скорости доступа. Чтобы не расходовать пропускную способность линий впустую, разнородные получатели должны сообщать сети о поддерживаемых ими скоростях. Как мы увидим, в протоколе RSVP уделяется первостепенное значение вопросу резервирования ресурсов для разнородных получателей.

Несколько простых примеров

Сначала опишем протокол RSVP в контексте конкретного примера групповой рассылки от одного отправителя нескольким получателям. Предположим, имеется источник, передающий в Интернет видеоизображение какого-либо спортивного состязания. Этому сеансу назначен групповой адрес, и источник маркирует все свои пакеты этим групповым адресом. Также предположим, что использующийся сетью протокол групповой маршрутизации сформировал дерево групповой рассылки от отправителя к четырем получателям, как показано на рис. 6.32. Рядом с каждым получателем на рисунке отмечена скорость, с которой он может прини-

мать данные. Предположим также, что видеоданные передаются в виде отдельных уровней с учетом неоднородности группы получателей.

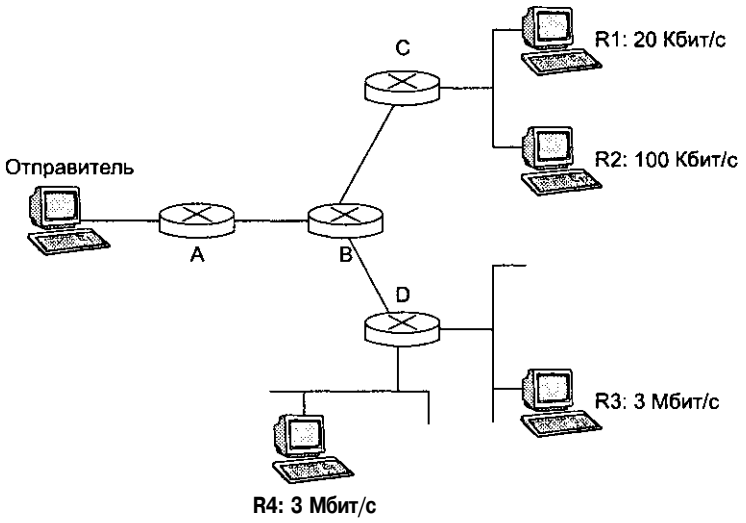


Рис. 6.32. Пример работы протокола RSVP

В первом приближении протокол RSVP работает следующим образом. Каждый получатель посылает вверх по дереву групповой рассылки *запрос на резервирование ресурсов*. В этом запросе указывается скорость, с которой получатель хотел бы принимать данные от источника. Когда запрос попадает на маршрутизатор, маршрутизатор настраивает свой планировщик пакетов, приспосабливаясь к новому потоку. Затем он пересылает запрос дальше вверх по дереву. Пропускная способность, резервируемая вверх по дереву, зависит от полученных маршрутизатором запросов на резервирование ресурсов от маршрутизаторов и получателей, расположенных ниже по дереву. В примере на рисунке получатели R1, R2, R3 и R4 резервируют 20 Кбит/с, 100 Кбит/с, 3 Мбит/с и 3 Мбит/с соответственно. Таким образом, получатели, соединенные с маршрутизатором D, запрашивают максимальную скорость 3 Мбит/с. Маршрутизатор D посылает маршрутизатору B запрос на резервирование пропускной способности 3 Мбит/с по линии между двумя маршрутизаторами. Обратите внимание, что резервируются только 3 Мбит/с, а не 3 + 3 = 6 Мбит/с, так как получатели R3 и R4 смотрят одну и ту же спортивную передачу, и, следовательно, их запросы на резервирование ресурсов могут быть объединены. Аналогично, маршрутизатор C просит маршрутизатор B зарезервировать ресурсы в 100 Кбит/с на линии, связывающей маршрутизаторы C и B. Схема многоуровневого кодирования гарантирует, что 20-килобитный поток для получателя R1 будет включен в 100-килобитный поток. Как только маршрутизатор B получает запросы на резервирование ресурсов от маршрутизаторов, расположенных ниже по дереву, и передает их своим планировщикам, он отправляет вверх расположенному выше маршрутизатору A но-

вый запрос на резервирование ресурсов. В нем запрашивается пропускная способность в 3 Мбит/с на линии, соединяющей маршрутизаторы A и B, то есть максимальная из запрошенных скоростей.

По этому первому примеру видно, что протокол RSVP *ориентирован на получателя*. То есть резервирование ресурсов инициируется и управляется получателем потока данных. Обратите внимание, что каждый маршрутизатор получает запросы на резервирование от всех ветвей дерева групповой рассылки и отправляет вверх по дереву только один запрос.

В следующем примере предположим, что четыре пользователя участвуют в видеоконференции (рис. 6.33). У каждого пользователя открыто по три окна, в каждом из которых отображается по собеседнику. Пусть применяемый в сети протокол маршрутизации сформировал дерево групповой рассылки между четырьмя хостами. Наконец, предположим, что каждый пользователь хочет получать от каждого из своих трех собеседников поток данных со скоростью 3 Мбит/с. Таким образом, для каждой линии этого дерева групповой рассылки протокол RSVP резервирует по 9 Мбит/с в одном направлении и 3 Мбит/с в другом направлении. Обратите внимание, что протокол RSVP в данном примере не объединяет запросы на резервирование, так как каждый пользователь хочет получать три разных потока.

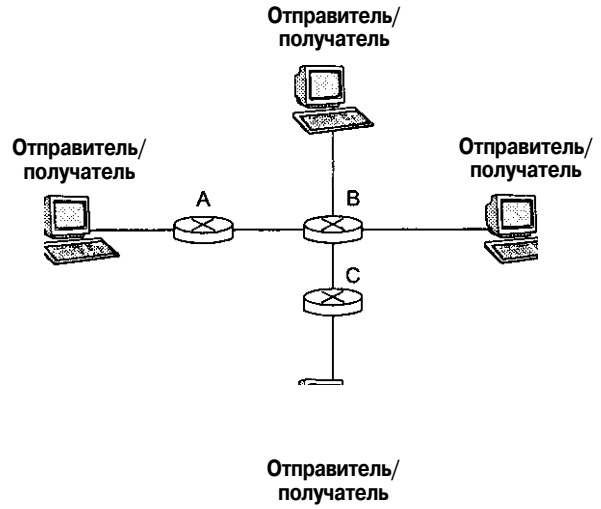


Рис. 6.33. Пример использования протокола RSVP при проведении видеоконференции

Теперь рассмотрим аудиоконференцию между теми же четырьмя пользователями с тем же самым деревом групповой рассылки. Пусть для отдельного аудиопотока требуется ***b*** бит/с. Поскольку в аудиоконференции редко бывает, чтобы одновременно говорили более двух участников, в данном случае нет необходимости резервировать 36 бит/с, достаточно зарезервировать *2b* бит/с. Таким образом, в этом последнем примере мы можем сэкономить пропускную способность, объединив запросы на резервирование.

ПРИНЦИПЫ И ПРАКТИКА

Резервирование на маршрутизаторах и хостах поддерживается в форме так называемого неустойчивого состояния. Под этим подразумевается, что с каждым запросом на резервирование пропускной способности на маршрутизаторе ассоциирован таймер. Когда указанный интервал времени резервирования истекает, резервирование отменяется и ресурсы высвобождаются. Если получатель потока данных желает поддерживать резервирование, он должен периодически продлевать резервирование, посылая запросы на резервирование. Этот принцип неустойчивого состояния применяется и в других сетевых протоколах. Как было показано в главе 5, в таблицах маршрутизации на прозрачных мостах записи обновляются пакетами данных, поступающими на мост. Необновляемые записи со временем устаревают и удаляются из таблицы. Если же протокол, напротив, требует явных действий для изменения состояния, тогда говорят, что состояние такого протокола устойчиво. Устойчивое состояние характерно для сетей виртуальных каналов, в которых, чтобы установить или разорвать виртуальный канал, необходимо предпринять явные действия по модификации таблиц виртуальных каналов на коммутирующих узлах.

Подобно тому как менеджер ресторана не должен принимать заказы на столики, если их становится больше, чем столиков в ресторане, суммарная пропускная способность, зарезервированная маршрутизатором для определенной линии, не должна превышать физическую пропускную способность линии. Таким образом, получая новый запрос на резервирование ресурсов, маршрутизатор должен сначала решить, смогут ли его исходящие линии предоставить заказанные ресурсы. Эта *проверка на допуск* в сеть проводится при получении каждого запроса на резервирование ресурсов. Если проверка на допуск дает отрицательный результат, маршрутизатор отвергает запрос и возвращает соответствующим получателям сообщение об ошибке.

Протокол RSVP не определяет процедуру проверки на допуск, но предполагает, что маршрутизатор ее выполняет.

Дифференцированное обслуживание

В предыдущем разделе мы говорили о том, как с помощью протокола RSVP резервировать ресурсы на маршрутизаторах сети *для отдельных потоков.* Возможность запросить и зарезервировать ресурсы для каждого потока в отдельности позволяет системе интегрированного обслуживания, в свою очередь, предоставлять отдельным потокам гарантии качества обслуживания. Однако по мере продвижения работ над системой интегрированного обслуживания и протоколом RSVP разработчики начали открывать для себя новые проблемы, связанные с моделью интегрированного обслуживания и резервированием ресурсов для отдельных потоков [567].

- *Масштабируемость.* При резервировании ресурсов для отдельных потоков с помощью протокола RSVP предполагается, что маршрутизатор должен обрабатывать запросы на резервирование и следить за состоянием каждого потока, проходящего через маршрутизатор. Таким образом, обработка запросов на резервирование ресурсов для отдельных потоков в больших сетях может составить значительные накладные расходы.

- *Модели гибкого обслуживания.* В архитектуре интегрированного обслуживания выделяется несколько заранее определенных классов обслуживания. Этот фиксированный набор классов не позволяет достаточно точно качественно разграничить разные уровни обслуживания (например, «обслуживание класса А предпочтительнее обслуживания класса В»). Такое качественное разграничение больше соответствует нашему интуитивному пониманию разных уровней обслуживания (например, места первого класса в самолетах лучше места второго класса, а платиновая кредитная карта лучше золотой, которая, в свою очередь, лучше обычной).

Эти соображения привели к так называемому дифференцированному обслуживанию [505], которым в рамках IETF занимается рабочая группа Diffserv (Differentiated Services — дифференцированное обслуживание). Эта группа разрабатывает архитектуру, призванную дать *масштабируемый* и *гибкий* способ различать уровни обслуживания — то есть уметь обслуживать разные «классы» трафика в Интернете по-разному. Необходимость в *масштабируемости* проистекает из того факта, что через магистральный маршрутизатор Интернета могут одновременно проходить сотни и тысячи потоков данных. Далее будет показано, что для дифференцированного обслуживания в ядре сети достаточно реализовать несколько простых функций, тогда как более сложные функции управления реализуются на периферии сети. Потребность в *гибкости* означает, что со временем могут появиться новые классы обслуживания, а существующие классы устареют. Архитектура дифференцированного обслуживания является гибкой в том смысле, что она не определяет конкретные классы обслуживания (как, например, в архитектуре интегрированного обслуживания). Вместо этого архитектура дифференцированного обслуживания предоставляет функциональные компоненты, то есть фрагменты сетевой архитектуры, из которых могут быть построены соответствующие службы. Рассмотрим эти компоненты более подробно.

Простой сценарий дифференцированного обслуживания

Рассмотрим простой пример сети, показанной на рис. 6.34. Мы опишем только один из возможных сценариев дифференцированного обслуживания. Возможны и другие сценарии (RFC 2475). Здесь наша задача заключается в том, чтобы показать ключевые аспекты дифференцированного обслуживания, а не описать эту архитектуру во всех деталях.

Архитектуру дифференцированного обслуживания составляют две группы функций.

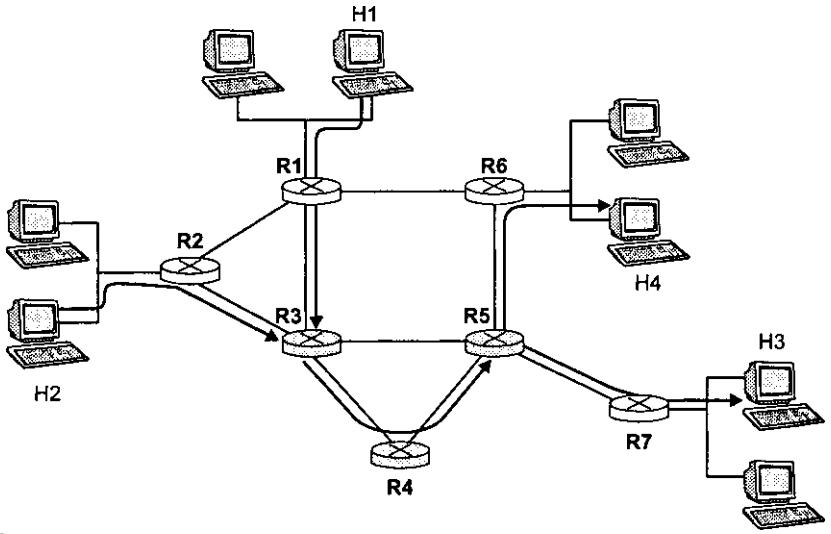
- *Периферийные функции — классификация пакетов и согласование трафика.* На «входе» сети (то есть либо на генерирующем трафик хосте, поддерживающем дифференцированное обслуживание, либо на первом маршрутизаторе, через который этот трафик проходит и который поддерживает дифференцированное обслуживание) поступающие пакеты маркируются, а именно в поле DS (Differentiated Services — дифференцированное обслуживание) заносится некоторое значение. Например, на рис. 6.34 пакеты, посылаемые от хоста H1 хосту H3,

могут быть промаркированы на маршрутизаторе R1, а пакеты, посылаемые от хоста H2 хосту H4, — на маршрутизаторе R2. Маркировка идентифицирует класс трафика, к которому он относится. При этом разные классы трафика получают разное обслуживание от ядра сети. В определении дифференцированного обслуживания (RFC 2475) вместо класса трафика используется понятие *совокупности поведения*. После того как пакет промаркирован, он может быть переправлен в сеть немедленно, временно задержан или отброшен. Далее будет показано, что на маркировку пакета, а также на то, насколько быстро он будет отправлен в сеть, могут влиять многие факторы.

- *Функции ядра — продвижение данных.* Когда пакет с определенной маркировкой в поле DS прибывает на поддерживающий дифференцированное обслуживание маршрутизатор, пакет переправляется по следующей линии в соответствии с так называемым *поведением на ретрансляционном участке,* ассоциированным с каждым классом пакетов. Поведение на ретрансляционном участке влияет на то, как распределяются буферы маршрутизатора и пропускная способность линий среди конкурирующих классов трафика. Центральный принцип архитектуры дифференцированного обслуживания состоит в том, что поведение на ретрансляционном участке основывается только на *маркировке* пакетов, то есть на классе трафика, к которому относится данный пакет. Таким образом, если пакеты, посланные от хоста HI хосту H3 (см. рис. 6.34), получают ту же маркировку, что и пакеты, посланные от хоста H2 хосту H4, тогда сетевые маршрутизаторы обрабатывают все эти пакеты одинаково, не различая пакеты, посланными хостом HI и хостом H2. Например, маршрутизатор R1 не станет различать пакеты, переправляя их маршрутизатору R4. Таким образом, архитектура дифференцированного обслуживания избавляет маршрутизаторы от необходимости хранить информацию о состоянии для отдельных пар отправителей и получателей, что очень важно с точки зрения масштабируемости, обсуждавшейся в начале этого раздела.

Здесь, может быть, полезно рассмотреть следующую аналогию. На многих масштабных социальных мероприятиях (например, в больших танцевальных клубах, на дискотеках, концертах или на футбольных матчах) посетители получают пропуска разного типа. Существуют VIP-пропуска для VIP-персон (VIP — Very Important People, то есть очень важные люди), пропуска для лиц, достигших возраста 21 года (например, если на приеме подаются алкогольные напитки), пропуска за кулисы на концертах, специальные пропуска для прессы, наконец, обычные пропуска для Обычных Людей. Эти пропуска, как правило, распределяются перед мероприятием, то есть на «периферии» мероприятия. Именно здесь на периферии выполняются такие сложные вычислительные операции, как оплата за вход, проверка соответствия типа приглашения, а также соответствия приглашения удостоверению личности. Кроме того, количество лиц, которым разрешается прийти на мероприятие, может быть ограничено. При подобном ограничении людям, возможно, придется ждать, прежде чем они смогут попасть на мероприятие. Наконец, когда посетитель попадает на представление, он получает дифференцированное обслуживание в зависимости от типа своего пропуска — VIP-персонам подаются бесплатные напитки, лучшие столы, бесплатная еда, им разрешается входить в опре-

деленные помещения, кроме того, они обслуживаются с особыми церемониями. Обычному посетителю, напротив, не разрешается заходить в определенные помещения, он должен платить за выпивку и довольствоваться обычным обслуживанием. В обоих случаях обслуживание зависит исключительно от типа пропуска, в то же время все клиенты одного класса обслуживаются одинаково.



Условные обозначения:

R2
Периферийный маршрутизатор

R2

£3^5 Центральный маршрутизатор

Рис. 6.34. Простой пример дифференцированного обслуживания

Классификация и согласование трафика

В архитектуре дифференцированного обслуживания маркировка пакета помещается в поле DS заголовка пакета протокола IPv4 или IPv6. Определение поля DS должно заменить более раннее определение поля типа службы (TOS) протокола IPv4 и поля класса трафика протокола IPv6 (см. главу 4). Структура этого восьмиразрядного поля показана на рис. 6.35.

Рис. 6.35. Структура поля DS в заголовке IPv4- и 1Ру6-пакетов

Шестиразрядное подполе DSCP (Differentiated Services Code Point) определяет поведение на ретрансляционном участке (см. одноименный подраздел данного

раздела), то есть то, как пакет будет обслуживаться в сети. Название двухразрядного подполя CU (Currently Unused — временно неиспользуемое) говорит само за себя. На значение половины поля DSCP наложены ограничения для обеспечения обратной совместимости с полем TOS протокола IPv4 (RFC 2474). Отметим лишь, что маркировка пакета помещается в поле DS.

Как уже отмечалось, маркировка пакета осуществляется в поле DS на периферии сети. Это может случиться либо на поддерживающем дифференцированное обслуживание хосте, либо в первой точке, где пакет встретит поддерживающий дифференцированное обслуживание маршрутизатор. В нашем обсуждении мы будем предполагать, что маркировка производится на периферийном маршрутизаторе, напрямую соединенным с отправителем (см. рис. 6.34).

Рисунок 6.36 иллюстрирует логическую точку зрения на функции классификации и маркировки на периферийном маршрутизаторе. Сначала прибывающие на периферийный маршрутизатор пакеты классифицируются. При этом пакеты выбираются на основе значений одного или нескольких полей заголовка (например, адреса отправителя, адреса получателя, порта отправителя, порта получателя и идентификатора протокола) и соответствующим образом маркируются. Затем пакеты переправляются по своему маршруту к получателю. На каждом последующем маршрутизаторе, поддерживающем дифференцированное обслуживание, маркированные пакеты получают обслуживание, соответствующее их маркировке. Даже такая простая схема маркировки подходит для разных классов обслуживания в Интернете. Например, все пакеты, поступающие с определенного набора IP-адресов (например, тех IP-адресов, владельцы которых заплатили своим Интернет-провайдерам за приоритетное обслуживание), могут соответствующим образом помечаться на маршрутизаторах своих Интернет-провайдеров, а затем получать специальное обслуживание (например, более высокий приоритет в очередях) на последующих маршрутизаторах, поддерживающих дифференцированное обслуживание. Рабочая группа Diffserv не определяет, по каким правилам классифицирующее пакеты устройство должно выполнять эту классификацию. Это может делаться вручную, то есть администратор сети может загружать в периферийные маршрутизаторы таблицы адресов тех отправителей, пакеты которых должны быть соответствующим образом помечены, или автоматически при помощи специального сигнального протокола, который еще предстоит разработать.

Рис. 6.36. Простая классификация и маркировка пакетов

В примере на рисунке все пакеты, заголовки которых удовлетворяют определенным условиям, получают одинаковую маркировку независимо от скорости поступления пакетов. В некоторых ситуациях может потребоваться ограничить скорость, с которой пакеты с определенной маркировкой поступают в сеть. Например, конечный пользователь может заключить со своим Интернет-провайдером договор

на получение высокоприоритетного обслуживания, но в то же время согласиться на ограничение максимальной скорости, с которой он может посылать пакеты в сеть. Таким образом, конечный пользователь соглашается с тем, что его скорость передачи пакетов будет соответствовать некоторому объявленному *профилю трафика*. В профиле трафика могут быть прописаны значения пиковой скорости и неравномерности потока пакетов (см. раздел «Механизмы проведения политики и планирования»). До тех пор пока пользователь посылает пакеты в сеть в соответствии с оговоренным профилем трафика, его пакеты получают приоритетную маркировку. Пакеты, нарушающие профиль трафика, могут маркироваться по-другому (например, задерживаться таким образом, чтобы соблюсти ограничение на максимальную скорость) или отбрасываться. Роль измеряющей функции (рис. 6.37) заключается в сравнении входящего потока пакетов с оговоренным профилем трафика и в том, чтобы определить, удовлетворяет ли пакет оговоренному профилю. Решение, принимаемое периферийным маршрутизатором (маркировать, задержать или отбросить пакет), не определяется архитектурой дифференцированного обслуживания. Архитектура дифференцированного обслуживания всего лишь предоставляет механизмы для маркировки, приостановки или отбрасывания пакетов. Она не предписывает какую-либо конкретную политику маркировки и согласования (приостановить или отбросить). Нам же остается только надеяться, что архитектурные компоненты дифференцированного обслуживания обладают достаточной гибкостью для поддержки широкого и постоянно расширяющегося спектра услуг, предоставляемых конечным пользователям. Обсуждение политики дифференцированного обслуживания содержится в [403].

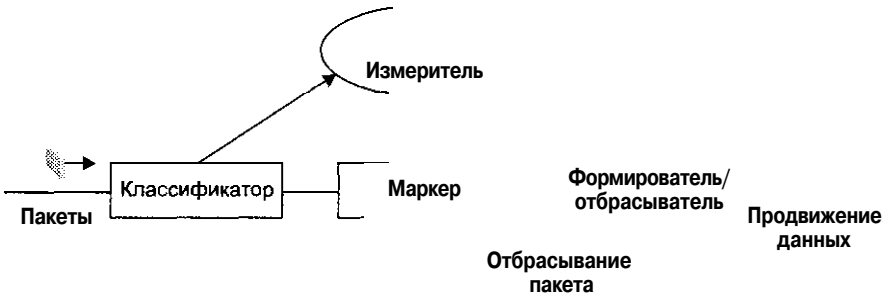


Рис. 6.37. Взгляд на классификацию и маркировку пакетов с логической точки зрения

Поведение на ретрансляционном участке

До сих пор мы фокусировали наше внимание на периферийных функциях архитектуры дифференцированного обслуживания. Второй ключевой группой функций являются функции *поведения на ретрансляционном участке* (Per-Hop Behavior, PHB), реализуемые маршрутизаторами, поддерживающими дифференцированное обслуживание. Поведение на ретрансляционном участке довольно путано и определяется как «описание наблюдаемого внешним наблюдателем поведения поддерживающего дифференцированное обслуживание узла относительно продвижения

данных определенного агрегата дифференцированного обслуживания» (RFC 2475). Если заглянуть в это определение немного глубже, мы увидим несколько важных соображений.

- Результатом реализации поведения на ретрансляционном участке может стать то, что разные классы трафика получат разную производительность (то есть разные с точки зрения внешнего наблюдателя образцы поведения по продвижению данных).
- В то время как поведение на ретрансляционном участке определяет различия в производительности (поведении) между классами, оно не предписывает применения какого-либо конкретного механизма для достижения данного поведения. До тех пор пока соблюдаются наблюдаемые критерии производительности, может использоваться любой механизм реализации и любая политика распределения ресурсов. Например, для поведения на ретрансляционном участке не важно, какая дисциплина организации очередей задействована (приоритетная организация очередей, взвешенная справедливая организация очередей или организация очередей FIFO).
- Различия в производительности должны быть наблюдаемыми, то есть измеряемыми.

Примером простого поведения на ретрансляционном участке может служить некий механизм, гарантирующий, что определенный класс маркированных пакетов получит, по меньшей мере, *x* процентов пропускной способности исходящей линии связи в течение некоторого интервала времени. Другой вариант поведения на ретрансляционном участке может заключаться в том, чтобы один класс трафика всегда получал приоритет над другим классом трафика, то есть если в очереди маршрутизатора одновременно находятся высокоприоритетный и низкоприоритетный пакеты, высокоприоритетный пакет всегда должен отправляться первым. Обратите внимание, что в то время, как приоритетная дисциплина очередей может быть естественным выбором для второго варианта поведения на ретрансляционном участке, для реализации требуемого наблюдаемого поведения годится любая дисциплина очередей.

Сегодня в рабочей группе Diffserv активно обсуждаются два варианта поведения на ретрансляционном участке: срочное продвижение данных (RFC 3246) и гарантированное продвижение данных (RFC 2597).

- В случае *срочного продвижения данных* (Expedited Forwarding, EF) гарантируется минимальная скорость отправки в сеть пакетов определенного класса. То есть в течение любого интервала времени классу трафика будет гарантировано получение достаточной пропускной способности для того, чтобы скорость трафика была не ниже определенного минимального уровня. Обратите внимание, что поведение на ретрансляционном участке, заключающееся в срочном продвижении данных, предполагает некую форму изоляции классов трафика, так как эта гарантия предоставляется *независимо* от интенсивности поступающего на маршрутизатор трафика других классов. Таким образом, даже если другие классы трафика перегружают ресурсы маршрутизатора и линий связи, для данного класса все равно будет выделено достаточно ресурсов, чтобы гарантиро-

вать указанную минимальную скорость трафика. Таким образом, при срочном продвижении данных классу трафика виртуально предоставляется линия с минимальной гарантированной пропускной способностью.

- При *гарантированном продвижении данных* (Assured Forwarding, AF) трафик делится на четыре класса, каждому из которых гарантируется определенный минимум пропускной способности и буферного пространства. В каждом классе пакеты разделяются на три категории, в соответствии с которыми они отбрасываются маршрутизатором в случае перегрузки (RFC 2597). Изменяя объем ресурсов, выделяемых каждому классу, Интернет-провайдер может предоставлять различным классам трафика разные уровни производительности.

Гарантированное продвижение данных позволяет предоставить оконечным системам разные уровни обслуживания, например (почти как в случае кредитных карт), золотого, серебряного и бронзового классов обслуживания. Но что для этого требуется? Если золотой класс обслуживания должен быть «лучше» (и, соответственно, дороже!) серебряного класса, тогда Интернет-провайдер должен гарантировать, что золотые пакеты меньше задерживаются и/или обладают меньшим процентом потерь, нежели серебряные. Вспомним, однако, что минимальная пропускная способность и буферное пространство выделяются *каждому* классу. Что произойдет, если золотому классу будет выделено *x* процентов пропускной способности линии, а серебряному классу — *x/2* процентов пропускной способности, однако интенсивность золотого трафика превысит интенсивность серебряного трафика в 100 раз? В этом случае, весьма вероятно, серебряные пакеты получат лучшую производительность, чем золотые! (В результате клиенты, заплатившие за дешевые услуги, окажутся в выигрыше по сравнению с клиентами, купившими более дорогие услуги!) Таким образом, очевидно, что одного только правильного поведения на ретрансляционном участке для дифференцированного обслуживания недостаточно. В данном примере для определения долей ресурсов, предоставляемых каждому классу трафика, необходимо учитывать требования всех классов трафика.

Недостатки дифференцированного обслуживания

За последние 20 лет предпринималось множество неудачных попыток гарантировать качество обслуживания в сетях с коммутацией пакетов. В основном все эти попытки проваливались по экономическим и законодательным, а не по техническим причинам. К этим попыткам можно отнести разработку сети ATM, а также сети TCP/IP с протоколом RSVP. Система дифференцированного обслуживания представляет собой еще одну попытку гарантировать качество обслуживания в Интернете. Есть ли у этой попытки шансы на хотя бы частичный успех? Рассмотрим недостатки системы дифференцированного обслуживания.

В приведенном выше обсуждении предполагалось, что архитектура дифференцированного обслуживания развертывается в одном административном домене. Более типичным является случай, при котором обслуживание должны предоставлять сразу несколько Интернет-провайдеров, расположенных между оконечными системами. Для дифференцированного обслуживания сквозных соединений Интер-

нет-провайдеры должны не просто его предоставить, а еще и сотрудничать друг с другом. Однако некоторые Интернет-провайдеры могут просто не захотеть качественно обслуживать клиентов, поскольку те покупали дополнительные услуги совсем у других провайдеров.

Другая проблема, связанная с реализацией дифференцированного обслуживания, заключается в необходимости осуществлять политику и аутентификацию при предоставлении такого обслуживания, чтобы предотвратить мошенничество, что может оказаться сложно и дорого. Кроме того, обслуживание разных классов должно также по-разному оплачиваться пользователями, причем, скорее всего, оплачиваться должен объем предоставляемых услуг, а не помесячный доступ, как практикуется сегодня большинством Интернет-провайдеров. Наконец, некоторые исследователи считают, что, даже если реализовать дифференцированное обслуживание, в большинстве случаев не будет заметной разницы между различными классами обслуживания. В самом деле, сегодня сквозная задержка, как правило, определяется скоростью доступа и количеством ретрансляционных участков, а не ожиданием в очередях на маршрутизаторах.

Дополнительные сведения о дифференцированном обслуживании можно получить в [269].

Резюме

Развитие систем передачи мультимедийных данных по сетям представляет собой, возможно, наиболее захватывающий процесс в сегодняшнем Интернете. Люди во всем мире проводят все меньше времени у своих радиоприемников и телевизоров, и все больше принимают радио- и телевизионные передачи по Интернету (как «в живую», так и в записи). По мере распространения высокоскоростного доступа среди клиентов эта тенденция будет продолжаться — домоседы всего мира начнут получать свои любимые видеопрограммы по Интернету, а не по традиционным широковещательным распределительным каналам. Помимо распространения аудио- и видеоданных Интернет также используется для транспортировки телефонных звонков. В самом деле, лет через десять во многих странах традиционная телефонная сеть с коммутацией каналов может устареть — ее полностью вытеснит Интернет. Интернет будет предоставлять не только более дешевые телефонные услуги, но и множество других ^полезных услуг, таких как видеоконференции, доступ к службам каталогов в режиме подключения (on-line), передача голосовых сообщений, интеграция в web.

В разделе «Сетевые мультимедийные приложения» была приведена классификация мультимедийных приложений по трем категориям: записанное потоковое аудио и видео, «живое» (в реальном времени) потоковое аудио и видео, а также интерактивное аудио и видео реального времени. Мы отметили, что мультимедийные приложения чувствительны к задержке, зато допускают определенную долю потерь пакетов, то есть обладают характеристиками, в корне отличающими их от приложений, передающих статические данные, которые допускают задержку, но не переносят потерь и искажений данных. Мы также обсудили некоторые препятствия,

которые сегодняшнее обслуживание по остаточному принципу ставит перед мультимедийными приложениями. Мы рассмотрели некоторые предложения по преодолению этих препятствий, включая простое усовершенствование существующей сетевой инфраструктуры (увеличение пропускной способности, сетевое кэширование, увеличение числа CDN-узлов, групповую рассылку), добавление к Интернету новых функций, позволяющих приложениям резервировать ресурсы (и обеспечивающих поддержку сетью этого резервирования), и, наконец, введение классов дифференцированного обслуживания.

Мы представили архитектуры и механизмы для передачи мультимедийных данных по сети, предоставляющие обслуживание по остаточному принципу. В разделе «Записанное потоковое аудио и видео» мы рассмотрели несколько подобных архитектур. Мы обсудили, как пользователь может управлять воспроизведением, и дали базовые сведения о протоколе RTSP, обеспечивающем взаимодействие клиента с сервером в потоковых приложениях. В разделе «Интернет-телефония» мы говорили о том, как разрабатывать интерактивные приложения реального времени, предназначенные для работы в сети, которая предоставляет обслуживание по остаточному принципу. Было показано, как буферизация на клиенте, введение порядковых номеров пакетов и меток времени могут существенно снизить эффект сетевого джиттера. В разделе «Протоколы для интерактивных приложений реального времени» мы изучили соответствующие протоколы, включая RTP, SIP и H.323.

Мы рассмотрели имеющиеся возможности по развитию Интернета в плане предоставления своим приложениям гарантированного качества обслуживания. В разделе «За пределами остаточного принципа» мы описали несколько принципов предоставления гарантий качества обслуживания мультимедийным приложениям. Сюда относят маркировку и классификацию пакетов, изоляцию потоков пакетов, эффективное использование ресурсов и допуск на установку соединения. В разделе «Механизмы проведения политики и планирования» обсуждались разнообразные политики и механизмы планирования, призванные реализовать основы сетевой архитектуры поддержания качества обслуживания. В качестве примеров политик были рассмотрены приоритетное планирование, циклическое планирование и взвешенная справедливая организация очередей. Затем мы познакомились с алгоритмом дырявого ведра, применяемым для проведения политики.

Далее было показано, как описанные принципы и механизмы привели к разработке новых стандартов предоставления гарантий качества обслуживания в Интернете. Первым классом подобных стандартов является так называемый стандарт интегрированного обслуживания (см. раздел «Интегрированное обслуживание»), описывающий два варианта обслуживания: с гарантированным качеством и с контролируемой нагрузкой. Для резервирования пропускной способности и буферного пространства архитектуре интегрированного обслуживания необходим сигнальный протокол. В разделе «Протокол RSVP» мы рассмотрели сигнальный протокол резервирования ресурсов. Мы отметили, что один из недостатков архитектуры интегрированного обслуживания заключается в том, что маршрутизаторы должны отслеживать состояние каждого проходящего через них потока, поэтому при увеличении размеров сети возникают проблемы. В последнем разделе главы (см. раздел «Дифференцированное обслуживание») мы рассмотрели архитектуру

дифференцированного обслуживания, которая не требует от маршрутизаторов отслеживать состояния потоков. Вместо этого в данном подходе пакеты разбиваются на несколько классов, которым маршрутизаторы предоставляют дифференцированное обслуживание на уровне ретрансляционных участков.

Итак, мы завершили изучение вопроса передачи мультимедиа по сети. Теперь настало время перейти к другому интересному вопросу — вопросу сетевой безопасности. Достижения последних лет способны переместить основные средства аудио- и видеовещания в Интернет. Как мы увидим в следующей главе, недавние разработки в области сетевой безопасности могут также способствовать перемещению в Интернет большей части экономических транзакций.

Вопросы и задания для самостоятельной работы

1. Что подразумевается под интерактивностью записанного потокового аудио и видео? Что подразумевается под интерактивностью интерактивного аудио и видео реального времени?
2. В этой главе упоминались три «лагеря» сторонников различных вариантов усовершенствования Интернета в плане поддержки мультимедийных приложений. Кратко опишите точки зрения каждого лагеря. К которому лагерю принадлежите вы?
3. На рис. 6.1,6.2 и 6.3 представлены три схемы передачи записанных потоковых мультимедийных данных. Каковы преимущества и недостатки каждой схемы?
4. Каковы различия между сквозной задержкой и джиттером пакетов? Каковы причины джиттера пакетов?
5. Почему пакет, опоздавший к планируемому времени воспроизведения, считается потерянным?
6. В разделе «Интернет-телефония» обсуждались две схемы прямого исправления ошибок (FEC). Опишите их кратко. В обеих схемах за счет роста накладных расходов увеличивается скорость передачи данных в потоке. Увеличивается ли скорость передачи данных в случае чередования?
7. Каким образом получатель идентифицирует различные RTP-потоки в разных сеансах? Как идентифицируются различные потоки в одном и том же сеансе? Каким образом различаются RTP-пакеты и RTPC-пакеты (в одном и том же сеансе)?
8. В разделе «Протоколы для интерактивных приложений реального времени» описываются три типа RTPC-пакетов. Кратко представьте информацию, содержащуюся в каждом из этих типов пакетов.
9. В разделе «Механизмы проведения политики и планирования» мы обсуждали приоритетную организацию очередей без прерывания обслуживания. Какой была бы приоритетная организация очередей с прерыванием обслуживания? Имеет ли подобная организация очередей смысл для компьютерных сетей?
10. Приведите пример дисциплины планирования, *не* поддерживающей рабочий режим.

11. Какими недостатками обладает модель интегрированного обслуживания и резервирования ресурсов для отдельных потоков?

Упражнения

1. Найдите в Интернете два продукта для передачи сохраненного потокового аудио и видео. Для каждого продукта ответьте на перечисленные ниже вопросы.
   1.1. Используются ли метафайлы?
   1.2. Какой протокол используется для транспортировки аудио- и видеоданных, UDP или TCP?
   1.3. Используется ли протокол RTP?
   1.4. Используется ли протокол RTSP?
2. Напишите поэму, короткий рассказ, сочинение или еще что-нибудь не слишком длинное на тему «Как я провел лето у бабушки». Прочитайте свое произведение вслух и запишите прочитанное в аудиофайл. Преобразуйте вашу запись в один из аудиоформатов RealNetworks или Microsoft при помощи одного из свободно распространяемых кодировщиков. Загрузите свой файл на сервер, на котором расположена ваша домашняя web-страница. Также загрузите на сервер соответствующий метафайл. Наконец, создайте на вашей web-странице ссылку на этот метафайл.
3. Рассмотрим буфер клиента, показанный на рис. 6.4. Пусть в потоковой системе используется третий вариант, то есть сервер пересылает мультимедийные данные в сокет с максимальной скоростью. Предположим, что большую часть времени доступная пропускная способность TCP » *d*. Кроме того, предположим, что буфер клиента может вмещать только около одной трети мультимедийных данных. Опишите, как значение *x(t)* и содержимое буфера клиента будут развиваться со временем.
4. Являются приемный TCP-буфер и клиентский буфер мультимедийного проигрывателя одним и тем же? Если нет, то как они взаимодействуют?
5. Пусть в примере Интернет-телефонии (см. раздел «Интернет-телефония») *h* — это общее количеством байтов заголовка, добавленных к блоку данных, включая UDP- и IP-заголовки.
   5.1. Предполагая, что IP-дейтаграмма передается каждые 20 мс, определите скорость передачи данных в битах в секунду для дейтаграмм, генерируемых одной стороной приложения.
   5.2. Чему, как правило, равно значение *h* при использовании протокола RTP?
6. Рассмотрим процедуру, описанную в разделе «Интернет-телефония» для оценки средней задержки d^t. Пусть $u = 0{,}1$. Пусть $r^i - t^i$ — это последнее значение задержки, r^2-t^2 — предыдущее значение задержки, и т. д.
   6.1. Пусть для данного аудиоприложения прибыло четыре пакета с задержками $r^4 - £_4$, $r_3 - £_3$, $r_2 - t_2$ и $r^i - t^x$. Чему равна в этом случае оценка задержки *d?*
   6.2. Обобщите вашу формулу для *n* пакетов.

6.3. Пусть для формулы из предыдущего упражнения n стремится к бесконечности. Приведите формулу для этого случая. Почему эта формула усреднения называется экспоненциальным скользящим средним?

7. Повторите упражнения 6.1 и 6.2 для оценки средней дисперсии задержки.

8. В примере Интернет-телефонии в одноименном разделе мы определили процедуру оценки задержки (экспоненциальное скользящее среднее). В этом упражнении мы рассмотрим другую процедуру. Пусть t_i — метка времени i-го полученного пакета, r_i — время получения i-го пакета, d_n — наша оценка средней задержки после получения n-го пакета. После получения первого пакета установим оценку задержки $d_1 = r_1 - t_1$.

   8.1. Пусть мы захотели использовать следующую формулу оценки:

   $$d_n = (r_1 - t_1 + r_2 - t_2 + \dots + r_n - t_n)/n.$$

   Напишите эту формулу в рекурсивном виде, то есть определите d_n через d_{n-1}, r_n и t_n.

   8.2. Чем формула оценки средней задержки, принятая в Интернет-телефонии, лучше формулы из предыдущего упражнения?

9. Сравните описанную в разделе «Интернет-телефония» процедуру оценки средней задержки с процедурой оценки времени оборота из раздела «За пределами остаточного принципа». Что общего у этих процедур? В чем их различие?

10. Рассмотрите описанную в разделе «Интернет-телефония» стратегию адаптивного воспроизведения.

    10.1. Как могут метки времени двух последовательных пакетов, принадлежащих к одной и той же «фразе», различаться более чем на 20 мс?

    10.2. Как с помощью порядковых номеров получатель может определить, является ли пакет первым в новой «фразе»?

11. Вспомним описанные в разделе «Интернет-телефония» две схемы прямого исправления ошибок для Интернет-телефонии. Предположим, первая схема для каждой группы из четырех блоков данных генерирует избыточный блок. Пусть во второй схеме используется низкоскоростное кодирование, для которого отводится 25 % от суммарной скорости передачи данных номинального потока.

    11.1. Сколько потребуется дополнительной пропускной способности для каждой схемы? На сколько увеличится задержка воспроизведения для каждой схемы?

    11.2. Что произойдет при использовании обеих схем, если потеряется первый пакет каждой группы из пяти пакетов? При использовании какой схемы будет наблюдаться лучшее качество звука?

    11.3. Что произойдет при использовании обеих схем, если потеряется первый пакет каждой группы из двух пакетов? При использовании какой схемы будет наблюдаться лучшее качество звука?

12. Как вычисляется джиттер в отчете о приеме протокола RTCP? (Рекомендуем прочитать посвященный этому протоколу документ RFC.)

12.1. Пусть в Интернет посылаются две IP-дейтаграммы, в каждой из которых содержится отдельный UDP-сегмент. В первой IP-дейтаграмме указан IP-адрес отправителя A1, IP-адрес получателя B, порт отправителя P1 и порт получателя T. Во второй IP-дейтаграмме указан IP-адрес отправителя A2, IP-адрес получателя B, порт отправителя P2 и порт получателя T. Пусть адрес A1 отличается от адреса A2, а порт отправителя P1 отличается от порта отправителя P2. Пусть обе дейтаграммы достигают пункта назначения. Будут ли две UDP-дейтаграммы приняты одним и тем же сокетом? Аргументируйте свой ответ.

12.2. Пусть Алиса, Боб и Клэр хотят провести аудиоконференцию при помощи протоколов SIP и RTP. Достаточно ли Алисе всего одного UDP-сокета для передачи и приема RTP-пакетов от Боба и Клэр (помимо сокета для SIP-сообщений)? Если да, тогда как SIP-клиент Алисы различит RTP-пакеты, полученные от Боба и от Клэр?

13. Рассмотрим RTP-сеанс, в котором принимают участие четыре пользователя, получающих и принимающих RTP-пакеты по одному и тому же групповому адресу. Каждый пользователь посылает видеоданные со скоростью 100 Кбит/с.

13.1. До какой скорости будет ограничивать свой трафик протокол RTCP?

13.2. Какая пропускная способность для протокола RTCP будет выделена отдельному получателю?

13.3. Какая пропускная способность для протокола RTCP будет выделена отдельному отправителю?

13.4. Что общего между протоколами RTSP и HTTP? Есть ли у протокола RTSP методы? Может ли протокол HTTP использоваться для запроса потока?

13.5. Чем протокол RTSP отличается от протокола HTTP? Например, является ли протокол HTTP внутриполосным или внеполосным? Хранит ли протокол RTSP информацию о состоянии клиента (например, о применении функций приостановки и возобновления воспроизведения)?

14. Какие существуют продукты корпорации Microsoft для проведения аудио- и видеоконференций реального времени? Используются ли в этих продуктах протоколы, обсуждавшиеся в этой главе (например, RTP или RTSP)?

15. Истинно или ложно следующее утверждение.

15.1. Если сохраненные видеоданные передаются мультимедийному проигрывателю напрямую с web-сервера, то приложение использует в качестве транспортного протокола протокол TCP.

15.2. Когда применяется протокол RTP, отправитель может изменять кодировку данных в ходе сеанса.

15.3. Все приложения, работающие по протоколу RTP, должны использовать порт 87.

15.4. Пусть в RTP-сеансе для каждого отправителя используются отдельные потоки для аудио и видео. В этом случае у каждой пары аудио- и видеопотоков один и тот же идентификатор SSRC.

15.5. При дифференцированном обслуживании поведение на ретрансляционном участке определяет различия в производительности между классами, но не предписывает использовать конкретный механизм достижения этой производительности.

15.6. Пусть Алиса хочет установить SIP-сеанс с Бобом. В свое сообщение INVITE она включает строку: m=audio 48753 RTP/AVP 3 (AVP 3 обозначает аудиоданные в формате GSM). Таким образом Алиса указывает, что она хочет передавать аудиоданные в формате GSM.

15.7. В предыдущем упражнении Алиса указывает своим сообщением INVITE, что она хочет посылать аудиоданные в порт 48753.

15.8. SIP-сообщения, как правило, пересылаются между SIP-объектами через порт протокола SIP, выделенный по умолчанию.

15.9. Для поддержки регистрации SIP-клиенты должны периодически посылать сообщения REGISTER.

15.10. Протокол SIP требует, чтобы все SIP-клиенты поддерживали кодировку аудиоданных G.711.

16. Пусть к буферу применяется политика планирования WFQ, поддерживающая три класса. Пусть веса этих классов равны 0,5,0,25 и 0,25.

16.1. Предположим, в буфере содержится большое количество пакетов каждого класса. В каком порядке должны обслуживаться эти три класса, чтобы получить указанные выше весовые коэффициенты WFQ? (При циклическом планировании мы бы имели последовательность 123123123...)

16.2. Пусть в буфере содержится большое количество пакетов классов 1 и 2, а пакетов класса 3 нет. В каком порядке должны обслуживаться эти три класса, чтобы получить указанные выше весовые коэффициенты WFQ?

17. Рассмотрим блок проведения политики дырявого ведра (см. раздел «Механизмы проведения политики и планирования»), регулирующий среднюю скорость и размер импульса потока пакетов. Теперь мы хотели бы также регулировать пиковую скорость *p*. Покажите, как можно направить выход блока проведения политики на вход другого аналогичного блока, так чтобы два последовательно соединенных «дырявых ведра» регулировали среднюю скорость, пиковую скорость и размер импульса потока пакетов. Укажите размер второго блока и скорость генерации маркеров во втором блоке.

18. Говорят, что поток соответствует спецификации дырявого ведра (г, *Ь)* с размером импульса *Ъ* и средней скоростью г, если количество пакетов, поступающих на вход «дырявого ведра», меньше, чем *rt* + **й,** для любого интервала времени *t* Может ли возникнуть ситуация, при которой потоку, соответствующему спецификации дырявого ведра (г, **й),** придется ждать поступления маркера в «дырявое ведро» с параметрами г и *Ь?* Аргументируйте свой ответ.

19. Покажите, что до тех пор, пока $r^{\prime} < R$ $w^{x}/\{$ **X",)>** значение $d^{m}.^{AX}$ действительно представляет собой максимальное значение задержки, которое будет когда-либо испытывать пакет потока 1 в очереди WFQ.

Дополнительные вопросы и задания

1. Как может хост использовать поступающую в ответ информацию протокола RTCP для определения, являются ли проблемы локальными, региональными или глобальными?
2. Как вы думаете, какой транспортный протокол лучше использовать для передачи записанного потокового аудио и видео, TCP или UDP?
3. Напишите доклад о SIP-продуктах компании Cisco.
4. Может ли проблема предоставления гарантий быть решена простым выделением приложению достаточной пропускной способности, то есть увеличением пропускной способности линий, чтобы ее всем всегда хватало?
5. В последнее время на рынке появилась интересная тенденция по использованию Интернет-телефонии и корпоративных высокоскоростных локальных сетей для замены традиционной внутренней телефонной сети компаний. Напишите отчет объемом на одну страницу по этой теме. Осветите в вашем отчете следующие вопросы.
   5.1. Что представляет собой традиционная внутренняя телефонная сеть? Кто ею пользуется?
   5.2. Рассмотрите телефонный разговор между сотрудником компании и пользователем, находящимся за пределами компании, с помощью традиционной телефонной сети. Какие технологии необходимы для установки интерфейса между локальной сетью и традиционной телефонной сетью?
   5.3. Что еще нужно помимо программного обеспечения и интерфейса, чтобы заменить традиционную внутреннюю телефонную сеть?
6. Рассмотрим четыре «краеугольных камня» системы предоставления гарантий качества обслуживания (см. раздел «За пределами остаточного принципа»). Опишите ситуации, если такие возможны, когда от каждого из этих «краеугольных камней» можно избавиться.

Задание по программированию

В этом задании вам предлагается реализовать потоковый видеосервер и соответствующий ему клиент. Для контролирования действий сервера клиент должен использовать протокол RTSP. Для пакетирования видеоданных и транспортировки их в UDP-дейтаграммах сервер должен использовать протокол RTP.

Вам предоставляется программа на языке Java, частично реализующая протоколы RTSP и RTP на клиенте и на сервере. Ваша задача заключается в доработке программ клиента и сервера. Когда вы закончите эту часть задания, вам предстоит написать распределенное приложение, алгоритм функционирования которого описан ниже.

1. Клиент посылает серверу RTSP-команды SETUP (установка соединения), PLAY (воспроизведение), PAUSE (приостановка) и TEARDOWN (разрыв соединения), а сервер выполняет эти команды.

2. В процессе воспроизведения сервер поочередно упаковывает хранящиеся у него JPEG-кадры в RTP-пакеты и посылает их в UDP-сокет.
3. Клиент получает RTP-пакеты, извлекает из них JPEG-кадры, распаковывает их и воспроизводит на мониторе клиента.

Предоставляемая вам программа реализует протокол RTSP на сервере и RTP-распаковку на клиенте. Программа также занимается воспроизведением передавае; мых видеоданных. Вам потребуется реализовать протокол RTSP на клиенте и протокол RTP на сервере.

Это задание позволит значительно улучшить понимание студентами протоколов RTP и RTSP, а также потокового видео. Мы настоятельно рекомендуем его выполнить. В этом задании также предлагается решить ряд дополнительных задач, включая реализацию RTSP-команды **DESCRIBE** как на клиенте, так и на сервере. Полные детали задания, а также важные фрагменты программ на языке Java вы найдете на web-сайте http://www.awl.com/kurose-ross.

Интервью

Хеннинг Шульцринне *является* адъюнкт-профессором и возглавляет лабораторию Интернет-приложений реального времени в Колумбийском университете. Он является соавтором протоколов RTSP, RTP и SIP — ключевых протоколов передачи аудио- и видеоданных через Интернет. Хеннинг Шульцринне получил степень бакалавра наук в Дармштадском университете в Германии, степень магистра наук по специальности электротехника и компьютерное проектирование в Университете Цинциннати, а степень доктора философии в области электротехники — в Массачусетском университете.

- Почему вы решили специализироваться в области распространения мультимедиа по сетям?

  Это произошло совершенно случайно. Будучи аспирантом, я получил возможность участвовать в проекте DARTnet — экспериментальной сети, опутывающей США линиями T1. Сеть DARTnet предназначалась для тестирования групповой рассылки и Интернет-приложений реального времени. Это подвигло меня на написание первого аудиоприложения NeVoT(Net Voice Tool — программа для передачи речи по сети). Благодаря другим участникам проекта DARTnet я был вовлечен в работу только появившейся тогда рабочей группы по аудио- и видеотранспорту в рамках IETF. Позднее эта группа завершила свою работу стандартизацией протокола RTP.

- Как выглядела ваша первая работа в компьютерной промышленности?

  Моя первая «работа» в компьютерной промышленности заключалась в пайке контактов компьютера Altair, когда я был старшеклассником в Ливерморе, штат Калифорния. Затем, вернувшись в Германию, я основал небольшую консультативную компанию, разработавшую программу адресного управления для туристического агентства. В этом приложении данные хранились на магнитофонных кассетах для накопителя TRS-80, а в качестве терминала использовалась печатная машинка IBM Selectric.

Моей первой «настоящей работой» я занимался в лаборатории Bell AT&T. Эта работа заключалась в разработке сетевого эмулятора для создания экспериментальных сетей в лабораторных условиях.

- Каковы цели лаборатории Интернет-приложений реального времени?

  Наша цель заключается в предоставлении фрагментов инфраструктуры будущего Интернета как единой коммуникационной платформы. Сюда входит разработка протоколов, таких как SIP и RTSP для сигнализации или RNAP, YES SIR и BGRP для резервирования ресурсов, измерение производительности Интернет-протоколов и приложений, а также разработка алгоритмов для улучшения качества обслуживания в Интернете. Мы создаем прототипы приложений, таких как серверы и клиенты для Интернет-телефонии, а также копаемся в аппаратуре, создавая собственную телефонную связь через Интернет.

  Не так давно мы начали исследовать вопрос, как можно найти в Интернете мультимедийные и другие сетевые службы. Кроме того, технологии беспроводных локальных сетей предлагают новые возможности по распространению мультимедиа средствами локального копирования. Эти технологии способны заменить собой масштабные сети.

- Каким вы видите будущее передачи мультимедиа по сетям?

  Сейчас мы переживаем переходный период. Нас отделяет всего лишь несколько лет от того времени, когда протокол IP станет универсальной платформой для мультимедийных служб. Мы знаем, что радиосвязь, телефонная связь и телевидение продолжают функционировать даже в условиях снежных буранов и землетрясений, поэтому Интернет возьмет на себя их функции только тогда, когда пользователи смогут рассчитывать на тот же уровень надежности.

  Исследования в области компьютерных сетей сталкиваются с той же проблемой, с которой на протяжении ряда лет сталкивались исследователи операционных систем. Хотя по-прежнему можно пользоваться специализированными операционными системами в небольших сообществах, аналогичные сетевые разработки, как правило, не достигают своих целей. Нам предстоит научиться создавать сетевые технологии, предназначенные для объединения конкурирующих носителей и учитывающие наличие массы невежественных и злобных конечных пользователей.

  Видимые успехи мультимедийных сетей будут во многом обязаны факторам, не относящимся к области мультимедиа, а именно успехам в области увеличения скоростей доступа и передачи данных по магистралям, а также удешевлению компьютерной обработки данных.

- Чем обусловлены многообещающие перспективы протокола SIP?

  По мере того как существующие беспроводные сети сменяются сетями стандарта 3G, есть надежда, что появится единый сигнальный мультимедийный механизм, охватывающий все типы сетей, от кабельных модемов до кор-

поративных телефонных сетей и общественных беспроводных сетей. Вместе с программным радио и при помощи 3G-сетей это сделает возможным использование единого устройства, например беспроводного (Bluetooth) телефона в домашней сети и интерфейса 802.11 в корпоративной и глобальной сетях. Уже сейчас, когда у нас еще нет такого универсального мобильного устройства, отдельные средства мобильности позволяют скрыть различия между сетями. Чтобы связаться с кем-либо, теперь достаточно всего одного идентификатора, вместо того чтобы запоминать полдюжины локальных телефонных номеров.

Кроме того, протокол SIP отделяет транспортировку голосовых данных от голосовых служб. Теперь становится возможным нарушить локальную монополию локальных телефонных сетей. Одна компания может предоставлять транспортировку нейтральных битов, в то время как другие компании обеспечат работу протокола IP и предоставят классические телефонные услуги, такие как шлюзы, перенаправление звонка и идентификация звонящего.

Помимо управления передачей мультимедийных данных протокол SIP предлагает новую услугу, которой недоставало в Интернете: уведомление о событиях. Мы приблизились к этой услуге при помощи неприспособленных для нее протоколов HTTP и электронной почты, но их всегда было недостаточно. Поскольку события для распределенных систем представляют собой глобальную абстракцию, их использование может упростить создание новых служб.

- Что вы можете посоветовать студентам, изучающим компьютерные сети?

Компьютерные сети представляют собой почти классическую дисциплину. Она берет свое начало от электротехники, кибернетики, исследования операций и других дисциплин. Таким образом, исследователи компьютерных сетей должны быть знакомы с многими предметами, находящимися за пределами этой области.

Компьютерные сети позволяют людям общаться, обмениваться идеями. Во многих областях технологии почти достигли предела производительности. Автомобили, поезда и самолеты почти не изменились за последние 20 лет, и, скорее всего, они не станут быстрее еще через 20 лет. Основной прогресс следует ожидать в способности обмениваться информацией, в увеличении безопасности транспорта и в предотвращении пробок.

ГЛАВА 7 Безопасность в компьютерных сетях

Позвольте вам представить Алису и Боба, двух молодых людей, жаждущих связи, но связи «безопасной». Поскольку мы говорим о сетях, Алиса и Боб могут быть двумя маршрутизаторами, которые хотят обменяться таблицами маршрутизации, клиентом и сервером, устанавливающими транспортное соединение, двумя приложениями электронной почты, пытающимися передать друг другу электронные письма. Все эти ситуации мы рассмотрим в этой главе. Вымышленные имена Алисы и Боба очень популярны в мире сетевой безопасности, возможно, потому что использовать эти имена веселее, чем просто буквы «А» и «Б». Рискованная любовная интрижка, военные коммуникации, деловые транзакции — везде нужна безопасная связь.

Итак, Алиса и Боб желают установить друг с другом безопасную связь, но что это означает? Как мы увидим, безопасность представляет собой сложную вещь, то есть у безопасности множество граней. Во-первых, Алиса и Боб хотели бы, чтобы содержимое их разговоров не достигло любопытных ушей (например, ревнивых супругов). Во-вторых, Алиса и Боб хотели бы удостовериться, что общаются они именно друг с другом, а если в их разговор вмешивается кто-то третий, то это вмешательство должно быть гарантированно обнаружено. В первой половине этой главы мы рассмотрим методы шифрования и дешифрирования, методы аутентификации собеседника, а также методы, гарантирующие целостность данных.

До недавних пор вопрос сетевой безопасности практически исчерпывался темами шифрования и дешифрирования, а также аутентификации и целостности данных. Однако в результате ставших популярными в последнее время атак типа отказа в обслуживании в различных сетях и на различных web-сайтах проявился еще один важный аспект сетевой безопасности — необходимость защиты от атакующих сети злоумышленников. Кроме того, мы рассмотрим сетевые брандмауэры, предоставляющие определенную степень изоляции и защиты от злоумышленников, расположенных по другую сторону брандмауэра. Мы также обсудим различные формы

атак на инфраструктуру сети и контрмеры, принимаемые, чтобы расстроить планы атакующих (или хотя бы смягчить негативный эффект от этих атак). В завершение главы мы рассмотрим несколько примеров реализации механизмов сетевой безопасности на прикладном, транспортном, сетевом и канальном уровнях.

Понятие сетевой безопасности

Начнем наше изучение вопроса сетевой безопасности с наших знакомых Алисы и Боба, желающих иметь безопасную связь. Что это означает? Разумеется, Алиса хочет, чтобы только Боб мог понимать отправляемые ею сообщения, несмотря на то, что сигнал проходит по небезопасной линии связи, где злоумышленник может перехватить, прочитать и, возможно, даже подменить сообщение, отправленное Алисой Бобу. Боб также хочет быть уверенным, что получаемые им от Алисы сообщения действительно посланы Алисой, а Алиса также хочет быть уверенной, что она общается именно с Бобом. Кроме того, Алиса и Боб хотели бы быть уверенными, что содержание их сообщений не меняется при передаче. Итак, мы можем сформулировать следующие желательные свойства *безопасной связи*.

- *Конфиденциальность.* Только отправитель и предполагаемый получатель должны быть способны понимать содержимое передаваемых сообщений. Поскольку злоумышленники могут перехватить сообщение, сообщение должно быть каким-то образом *зашифровано* (его данные должны быть скрыты), так чтобы перехвативший сообщение злоумышленник не смог его *расшифровать* (понять). Вероятно, именно этот аспект конфиденциальности имеется в виду, когда говорится о «безопасной связи». Однако обратите внимание, что такое определение является ограниченным не только в смысле безопасности связи (дополнительные аспекты безопасной связи мы перечислим ниже), но и в смысле конфиденциальности. Например, Алиса может также хотеть, чтобы сам факт того, что она общается с Бобом (или частота общения, или время ее выхода на связь), оставался тайной. Мы рассмотрим методы криптографии для шифрования и дешифрирования данных в разделе «Принципы криптографии». Мы увидим, что конфиденциальная связь часто полагается на один или несколько *ключей,* используемых для шифрования и дешифрирования данных. Вопрос распространения ключей мы обсудим в разделе «Передача ключей и сертификация».
- *Аутентификация.* Как отправитель, так и получатель должны быть способны подтвердить личность собеседника — убедиться, что они общаются именно с тем человеком, за которого он себя выдает. При общении лицом к лицу эта проблема легко решается визуально. Когда же собеседники не могут «видеть» друг друга, аутентификация представляет собой значительно более сложную проблему. Почему, например, вы должны верить, что полученное вами электронное письмо действительно было послано вам тем человеком, чье имя указано в строке с адресом отправителя? Если кто-либо звонит по телефону, говорит, что является сотрудником вашего банка, и просит вас сказать ему номер вашего счета, секретный PIN-код (Personal Identification Number — персональный идентификационный номер) и баланс счета для проверки, дадите ли вы эту информацию по телефону?

Скорее всего, нет! Мы рассмотрим методы аутентификации в разделе «Аутентификация», включая и те, в которых используются криптографические методы, описываемые в разделе «Принципы криптографии».

- *Целостность сообщения.* Даже если отправитель и получатель способны удостовериться в подлинности друг друга, они также хотят быть уверенными, что их сообщения не изменяются (случайно или злонамеренно) при пересылке. Для обеспечения целостности сообщения могут применяться различные расширения методов подсчета контрольных сумм, с которыми мы познакомились при изучении надежных транспортных протоколов и протоколов передачи данных. Этот вопрос мы рассмотрим в разделе «Целостность данных». Мы также узнаем, как получатель может убедиться в том, что сообщение пришло именно от того человека, который указан как отправитель сообщения. Как мы увидим, обеспечение целостности данных реализуется при помощи криптографических методов, которым посвящен раздел «Принципы криптографии».
- *Управление доступом.* Вынужденная необходимость в сетевой безопасности стала очевидной в последние годы благодаря многочисленным атакам типа отказа в обслуживании, в результате которых сеть, хост или другой фрагмент сетевой инфраструктуры становился недоступным легитимным пользователям. Возможно, наиболее известными атаками подобного рода стали атаки против web-сайтов солидных компаний. Таким образом, ключевым для безопасной связи является требование о том, чтобы связь вообще состоялась — злоумышленники не должны иметь возможности мешать пользоваться сетевой инфраструктурой. Тот факт, что одни пользователи могут быть легитимными, а другие нет, естественным образом наводит на мысль об управлении доступом — гарантии того, что пользователи, пытающиеся получить доступ к ресурсам, смогут сделать это только в том случае, если обладают соответствующими правами доступа и осуществляют этот доступ определенным образом.

Конфиденциальность, аутентификация и целостность сообщения уже довольно давно считаются ключевыми компонентами безопасной связи [321]. В последние годы доступность и управление доступом стали также считаться компонентами безопасной связи [303], что, безо всякого сомнения, обусловливается соображениями усиления защиты сетевой инфраструктуры от возможных атак злоумышленников. Наиболее верный способ гарантировать, что злоумышленники не смогут причинить вреда, — это предотвратить попадание их пакетов в сеть. *Брандмауэр* представляет собой устройство, располагаемое между внутренними и внешними сетями и защищающее их друг от друга. Это устройство управляет доступом пакетов в сеть и отправкой пакетов из сети. Брандмауэры быстро стали обязательным сетевым компонентом, начиная от небольших домашних сетей и заканчивая сетями самых больших корпораций. Мы поговорим о том, как брандмауэры обеспечивают управление доступом, в разделе «Управление доступом с помощью брандмауэров».

Наше определение безопасной связи, главным образом, фокусировалось на *защите* сетевых и коммуникационных ресурсов. На практике сетевая безопасность включает не только механизмы защиты, но и механизмы *обнаружения* атак, а также механизмы *ответных действий.* Во многих случаях в ответ на атаку администт-

ратор сети может установить дополнительные механизмы защиты. В этом смысле сетевая безопасность достигается с помощью непрерывного цикла: защита, обнаружение, ответ. (Несколько устаревшей, но все еще полезной книгой, посвященной аспектам этого цикла, является [490].)

Таким образом, мы продолжим эту главу (см. раздел «Атака и оборона») обсуждением примеров атак и предпринимаемых контрмер для их отражения. В разделе «Безопасность на разных уровнях» мы завершим эту главу примерами применения различных методов защиты, описанных в предыдущих разделах.

Итак, дав определение сетевой безопасности, рассмотрим, к какой информации может получить доступ злоумышленник и какие действия он может предпринять. Схема диалога Алисы и Боба с участием в нем злоумышленника изображена на рис. 7.1. Алиса хочет отправить данные Бобу. Чтобы обмениваться данными безопасным образом, а также выполнить требования конфиденциальности, аутентификации и целостности данных, Алиса и Боб обмениваются управляющими и информационными сообщениями (подобно тому, как TCP-отправители и TCP-получатели обмениваются управляющими и информационными сегментами). Все или некоторые из этих сегментов, как правило, зашифровываются. Пассивный злоумышленник может прослушивать линию и перехватывать управляющие и информационные сообщения, передаваемые по каналу. Как мы увидим, если не предпринимать соответствующих контрмер, злоумышленник сможет прибегнуть к весьма изощренным атакам, начиная от прослушивания переговоров (при этом возможно похищение злоумышленником паролей и данных) до выдачи себя за одного из собеседников и нарушения работы системы путем ее перегрузки. Эти и другие разновидности атак будут обсуждаться в разделе «Атака и оборона». Этим же темам посвящены дополнительные источники информации [118, 442]. Отчеты о замеченных атаках содержатся на web-сайте координационного центра CERT (Computer Emergency Response Team — бригада компьютерной «скорой помощи») по адресу http://www.cert.org/advisories. См. также [13,87,539].

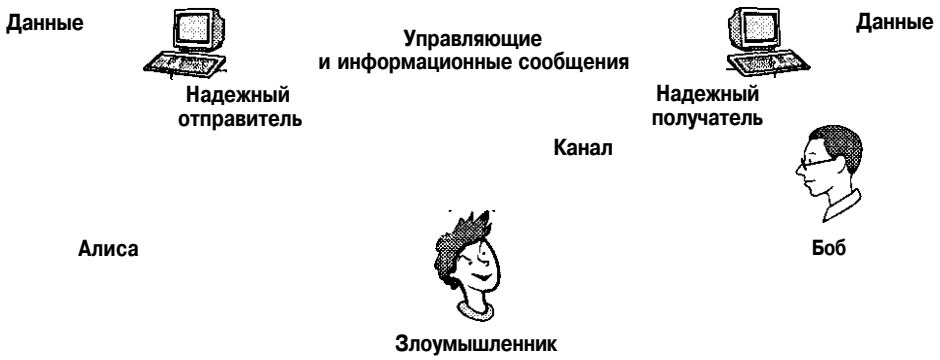


Рис. 7.1. Отправитель, получатель и злоумышленник

Кем могут быть реальные Алиса и Боб? Это могут быть два пользователя на двух оконечных системах, желающие обменяться письмами по электронной почте. Это также могут быть участники коммерческой сделки. Например, Алиса может пожe-

лать переслать номер своей кредитной карты на web-сервер для осуществления покупки. Кроме того, Алиса может захотеть связаться по сети со своим банком. Однако, как отмечается в RFC 1636, безопасная связь может быть также организована между отдельными частями сетевой инфраструктуры. Вспомним, что безопасная связь необходима для обмена таблицами маршрутизации (см. раздел «Маршрутизация в Интернете» в главе 4) между маршрутизаторами в системе DNS (Domain Name System — система доменных имен), обсуждавшейся в разделе «Служба трансляции имен Интернета» главы 2. То же самое справедливо в отношении приложений, занимающихся сетевым управлением, о чем будет рассказано в главе 8. Злоумышленник, способный активно вмешиваться в данные систем DNS, может стать причиной очень больших беспорядков в Интернете.

Перейдем теперь к обсуждению криптографии, темы, представляющей особую важность для большинства аспектов сетевой безопасности.

Принципы криптографии

Хотя криптография имеет долгую историю, например, известно, что простейшие шифры применял Юлий Цезарь (ниже мы рассмотрим так называемый шифр Цезаря), современные криптографические методы, включая многие из тех, что используются в сегодняшнем Интернете, основаны на достижениях последних 30 лет. Захватывающая история криптографии отражена в [254,472]. Подробная (но очень занимательная) техническая дискуссия криптографии имеется в [261]. В [123] предоставляется современное исследование политических и социальных вопросов, причудливо переплетающихся с вопросами криптографии. Для полного обсуждения криптографии требуется целая книга (например, [261,446]), поэтому мы только коснемся существенных аспектов криптографии в том виде, в котором они практикуются в сегодняшнем Интернете. Этой теме также посвящен прекрасный сайт лабораторий RSA (http://www.rsasecurity.com/rsalabs/faq/). Отметим, что хотя основным вопросом, которому посвящен этот раздел, является вопрос использования криптографии для обеспечения конфиденциальности, мы увидим, что криптографические методы тесно связаны с вопросами аутентификации, целостности сообщений и т. п.

Криптографические методы позволяют отправителю скрыть содержимое своих посланий, поэтому злоумышленник не может получить информацию из перехваченных сообщений. Само собой, получатель должен быть способен восстановить исходные данные. Некоторые из наиболее важных понятий иллюстрирует рис. 7.2.

Предположим, что Алиса хочет переслать Бобу сообщение. Сообщение Алисы в его исходном виде (например, «Боб, я люблю тебя. Алиса») называется *открытым текстом*. Алиса зашифровывает это сообщение при помощи *алгоритма шифрования,* в результате *зашифрованное сообщение* выглядит непонятно для злоумышленника. Интересно отметить, что во многих современных криптографических системах, включая те, что используются в Интернете, сам алгоритм шифрования известен, то есть опубликован (например, в RFC 1321, RFC 2437 и RFC 2420), стандартизован и доступен всем и каждому, даже потенциальному злоумышленнику! Очевидно, если метод шифрования данных известен всем, должна быть какая-то секрет-

ная информация, не позволяющая злоумышленнику расшифровать пересылаемые данные. Такой информацией является ключ.

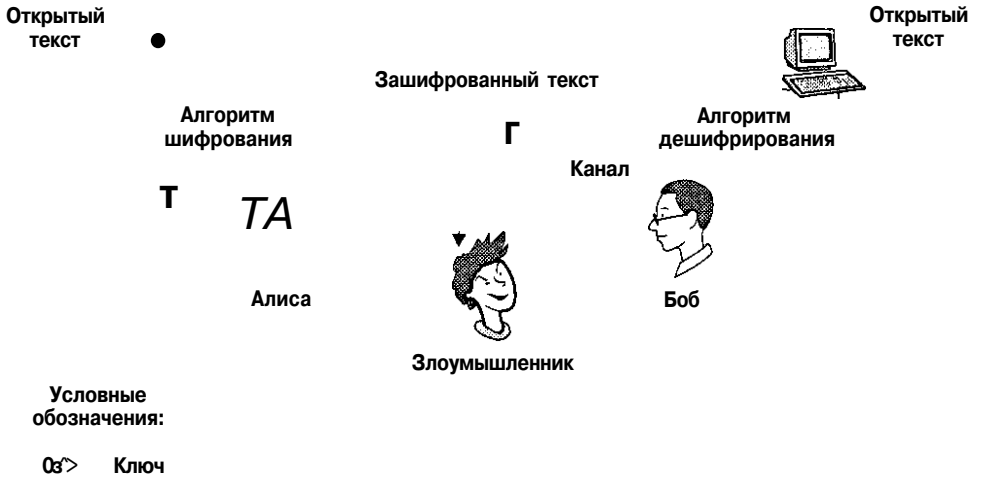


Рис. 7.2. Компоненты криптографической системы

ИСТОРИЧЕСКАЯ СПРАВКА

Первый стандарт шифрования данных (Data Encryption Standard, DES), принятый правительством США в 1977 году, представлял собой 56-разрядный алгоритм, все еще широко применяемый в финансовой отрасли и других отраслях промышленности для защиты информации. Компания RSA финансировала несколько состязаний по взлому шифра DES, чтобы подчеркнуть необходимость в более мощных алгоритмах шифрования, чем текущий 56-разрядный стандарт, широко используемый для засекречивания как государственной (США), так и международной коммерции. Каждое задание заключалось в требовании расшифровать сообщение, зашифрованное 56-разрядным шифром DES за указанный интервал времени. Взломщик шифра, принимая вызов, должен был перебрать все возможные секретные ключи. Приз за успешную попытку взлома составлял 10 000 долларов.

В январе 1997 года компания RSA объявила о конкурсе, цель которого заключалась втом, чтобы показать, что стандарт DES с 56-разрядным ключом обеспечивает лишь ограниченную защиту от целеустремленного противника. Первое состязание выиграла команда из Колорадо, сумевшая получить секретный ключ менее чем за четыре месяца. За годы, прошедшие с тех пор, технология не стояла на месте, поэтому для взлома такого кода нынче требуется значительно меньшее время. В феврале 1998 года конкурс компании RSA DES Challenge 11-1 выиграла группа Distributed.Net, взломав шифр за 41 день, а в июле компания Electronic Frontier Foundation выиграла конкурс компании RSA DES Challenge II-2, взломав шифр DES за 56 часов.

Чтобы выиграть конкурс компании RSA под названием DES Challenge III за рекордно короткий срок в 22 часа и 15 минут, всемирная коалиция компьютерных энтузиастов Distributed.Net в январе 1999 года разработала проект EFF «Deep Cracк», объединяющий специально спроектированный суперкомпьютер и 100 000 персональных компьютеров, подключенных к Интернету. Система группы Distributed.Net перебирала по 245 миллиардов ключей в секунду!

В качестве входных данных для алгоритма шифрования Алиса использует *ключ* K^A, представляющий собой текстовую строку. Алгоритм шифрования принимает ключ и открытый текст m на входе и выдает зашифрованное сообщение на выходе. Для обозначения сообщения m, зашифрованного с помощью ключа K^A, используется нотация $K^A(m)$. Аналогично, Боб предоставляет ключ $K^в$ и зашифрованный текст *алгоритму дешифрирования,* выдающему на выходе оригинальный открытый текст. Таким образом, если Боб получает зашифрованное сообщение $K^A(m)$, он расшифровывает его, вычисляя $K^в(K^A(m)) = m$.*В системах с симметричными ключами* Алисой и Бобом используются идентичные и секретные ключи. В *системах с открытым ключом* применяется пара ключей. Один из ключей известен как Бобу, так и Алисе (а также всем и каждому). Второй ключ известен только Бобу или Алисе (но не обоим). В следующих двух подразделах мы рассмотрим системы с симметричными ключами и системы с открытым ключом более подробно.

Шифрование с симметричными ключами

Все криптографические алгоритмы заменяют один текст другим: открытый текст — зашифрованным. Прежде чем обсудить современные криптографические системы, рассмотрим очень старый простой алгоритм с симметричными ключами, приписываемый Юлию Цезарю и известный как *шифр Цезаря.*

Чтобы зашифровать шифром Цезаря английский текст, нужно каждую букву открытого текста заменить буквой, располагающейся в алфавите на k символов дальше (при этом весь алфавит рассматривается как цикл, то есть за буквой «z» следует буква «a»). Например, если $k = 3$, тогда буква «a» в открытом тексте будет заменена в зашифрованном тексте буквой «d»; буква «Ь» станет буквой «e» и т. д. В данном шифре ключом служит параметр к. Например, открытое сообщение «bob, i love you. alice» превратится в «ere, 1 oryx brx. dolfh». Хотя зашифрованный текст выглядит как полная тарабарщина, чтобы взломать такой шифр, не понадобится много времени, если известно, что использовался шифр Цезаря, так как у этого алгоритма шифрования может быть только 25 значений ключа.

Усовершенствованием шифра Цезаря является *моноалфавитный шифр,* также заменяющий одну букву алфавита другой. Однако вместо того, чтобы заменять символы регулярным образом (например, смещением в алфавите на постоянную величину), любая буква может заменяться любой другой буквой, при условии, что каждая буква замещается уникальным образом. На рис. 7.3 показан пример такого шифра.

Символ открытого текста:	a b c d e f g h i j k l m n o p q r s t u v w x y z
Символ зашифрованного текста:	m n b v c x z a s d f g h j k l p o i u y t r e w q

Рис. 7.3. Моноалфавитный шифр

В этом случае открытое сообщение «bob, i love you. alice» превратится в «nkn, s gktc wky. mgsbc». Таким образом, как и в случае шифра Цезаря, зашифрованное

сообщение выглядит непонятно. Однако моноалфавитный шифр значительно более надежен, нежели шифр Цезаря, так как существует 26! (величина порядка 10^{26}) возможных вариантов соответствий между алфавитами открытого текста и зашифрованного текста. Попытка найти ключ полным перебором всех 10^{26} возможных вариантов потребует слишком больших усилий. Однако статистический анализ позволяет значительно упростить задачу взлома подобного кода. Так, например, известно, что в английском тексте чаще всего встречаются символы «e» и «t» (13 и 9 % соответственно). Также известны значения частот появления двухбуквенных и трехбуквенных сочетаний (например, «in», «it», «the»,«ion», «ing» и т. д.), в результате взломать такой шифр оказывается не так уж и сложно. Если же злоумышленник обладает некоторой информацией о сообщении, задача взлома шифра становится еще проще. Например, если злоумышленником является жена Боба, подозревающая Боба в том, что у него интрижка с Алисой, то она может предположить, что в тексте должны встречаться имена Алисы и Боба. В этом случае, перехватив приведенное выше зашифрованное сообщение, она может отгадать семь из 26 символов ключа, в результате для продолжения взлома путем полного перебора всех вариантов понадобится в 10^{9} раз меньше времени.

Для каждого нового шифра необходимо определить, насколько легко злоумышленник может его взломать. При этом рассматриваются три сценария, различающихся информацией, которой обладает злоумышленник.

- *Атака, при которой у злоумышленника есть только зашифрованный текст.* В некоторых случаях злоумышленник может получить доступ только к перехваченному им зашифрованному тексту без какой-либо информации о содержимом перехваченного сообщения. Как уже отмечалось, для взлома шифра может использоваться статистический анализ.
- *Атака, при которой злоумышленнику известен открытый текст.* Ранее было показано, что если злоумышленник каким-либо образом сумел догадаться о том, что в зашифрованном сообщении встречаются слова «ЬоЬ» и «alice», то он может определить пары (открытый текст, зашифрованный текст) для букв «a», «l», «c», <<e», «Ь» и «o». Кроме того, злоумышленнику может повезти еще больше, если он случайно обнаружит расшифрованную версию одного из сообщений, полученных Бобом (что в случае ревнивой супруги весьма вероятно).
- *Атака, при которой злоумышленник может произвольно выбирать открытый текст.* В этом случае злоумышленник способен выбирать открытый текст и получать соответствующий ему зашифрованный текст. При таком простом алгоритме шифрования (моноалфавитный шифр) злоумышленник может перехватить сообщение, содержащее все буквы алфавита, например «the quick brown fox jumps over the lazy dog», и полностью взломать схему шифрования. Далее будет показано, что в более сложных схемах шифрования подобная атака не обязательно означает взлом системы.

Пятьсот лет назад метод моноалфавитного шифрования был усовершенствован, в результате появился так называемый *полиалфавитный шифр.* Идея этого метода заключается в использовании нескольких моноалфавитных шифров, причем выбор шифра определяется позицией кодируемого символа в открытом сообщении.

Таким образом, один и тот же символ открытого текста может кодироваться поразному. Пример полиалфавитного шифра показан на рис. 7.4. В нем применяются два шифра Цезаря С, и С2 (с параметрами $k = 5$ и $k = 19$). Эти два шифра Цезаря могут использоваться в следующем повторяющемся порядке: Q, С2, С2, Q, С2. То есть первый символ открытого текста кодируется при помощи шифра C^u второй и третий символы — с помощью шифра С2, для шифрования четвертого символа опять используется шифр Q, пятый символ шифруется с помощью шифра С2. Затем вся последовательность повторяется, то есть шестой символ шифруется при помощи шифра Q, седьмой — при помощи шифра С2 и т. д. В результате открытое сообщение «bob, i love уои» превращается в «ghu, n etox dhz». Обратите внимание, что первый символ «Ь» открытого сообщения кодируется шифром Q, тогда как второй символ «Ь» — шифром С2. В этом примере «ключ» шифрования и дешифрирования представляет собой знание двух ключей Цезаря (& = 5 и & = 19) и последовательности Q, С2, С2, Q, С2.

Символ открытого текста:	a b c d e f д h i j k 1 m n o p q r s t u v w x y z
Ci(/c=5):	f **д** h i j k 1 m n o p q r s t u v w x y z a b c d e
C_2(/c=19):	t u v w x y z a b c d e f g h i j k l m n o p q r s

Рис. 7.4. Полиалфавитный шифр, использующий два шифра Цезаря

Перейдем теперь к обсуждению современного шифра DES [347], представляющего собой стандарт шифрования с симметричным ключом, опубликованный в 1997 году и обновленный в 1993 году национальным бюро стандартов США для шифрования коммерческой и несекретной государственной документации. Шифр DES кодирует 64-разрядные блоки кода при помощи 64-разрядного ключа. В действительности длина ключа шифра DES составляет всего лишь 56 бит, так как 8 из 64 бит представляют собой биты четности (по одному для каждого из восьми байтов). Национальный институт стандартов и технологии (National Institute of Standards and Technology, NIST), который сменил национальное бюро стандартов (National Bureau of Standards, NBS), следующим образом формулирует назначение шифра DES [345]: «Цель заключается в том, чтобы полностью скремблировать данные и ключ, так чтобы каждый бит данных в зашифрованном тексте зависел от каждого бита открытого текста и каждого бита ключа... Не должно быть никакой корреляции между зашифрованным текстом и оригинальными данными или ключом».

Основные операции алгоритма DES иллюстрирует рис. 7.5. В нашем обсуждении мы рассмотрим работу алгоритма DES в общих чертах. Подробности вы можете узнать в других источниках, например в [261,446]. В [446] также имеется реализация алгоритма DES на языке С. В алгоритме DES перестановка выполняется в два этапа (первый и последний шаги алгоритма), в которых 64 бита переставляются местами. Между этими двумя этапами выполняется еще 16 идентичных этапов. Входными данными для каждого из этих 16 этапов является результат предыдущего этапа. На каждом этапе левые 32 бита заменяются правыми 32 битами. Все 64 входных бита на ґ-м этапе и 48-разрядный ключ z-го этапа (полученный из 56-разрядного

ключа DES) подаются на вход функции, превращающей 4-разрядные блоки данных в 6-разрядные, складывающей эти 6-разрядные блоки по модулю 2 с полученными при помощи аналогичных операций 6-разрядными блоками 48-разрядного ключа *K»* а также операции замены и еще одного сложения по модулю 2 с левыми 32 битами входных данных. Детали см. в [261,446]. Полученные в результате этих вычислений 32 бита образуют правые 32 бита 64-разрядного результата (рис. 7.5). При дешифрировании используются обратные операции этого алгоритма.

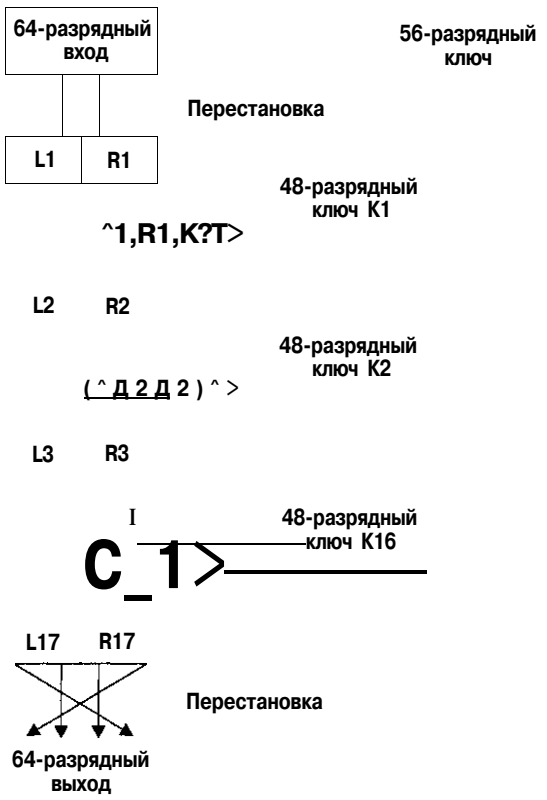


Рис. 7.5. Основные операции алгоритма DES

Насколько хорошо работает алгоритм DES? Насколько он надежен? Никто не может сказать этого наверняка [261]. Как отмечалось выше, в 1997 году компания RSA Data Security, занимающаяся сетевой безопасностью, объявила о конкурсе на взлом кода DES. Участникам конкурса предлагалось декодировать короткую фразу «Strong cryptography makes the world a safer place» (надежная криптография делает мир безопаснее). Взломать шифр менее чем за четыре месяца удалось команде, использовавшей добровольцев в Интернете для систематического перебора всех вариантов ключа. При взломе кода была перебрана всего лишь четверть всех вариантов ключа — около $1{,}8\ 10^{16}$ ключей [435]. Последний конкурс DES Challenge III, проводившийся в 1999 году, был выигран за рекордно короткое время 22 часа при помощи

добровольцев и специализированного компьютера «Deep Crack», построенного менее чем за 250 000 долларов. Информацию об этом можно найти в Интернете [136]. Необходимо также отметить, что мы рассмотрели кодирование только 64-разрядного блока данных. Как правило, длина сообщений превышает 64 бита. При шифровании более длинных сообщений шифр DES часто используется вместе с методом *сцепления блоков шифра,* при котором перед шифрованием *(j* + 1)-го 64-разрядного блока этот блок складывается по модулю 2 с уже зашифрованным *j-м* 64-разрядным блоком данных.

Если 56-разрядный шифр DES считается недостаточно надежным, можно просто использовать его несколько раз с разными ключами. Именно таким образом работает государственный стандарт США *3DES* (triple-DES — тройной шифр DES) [346], призванный заменить уже устаревший стандарт DES, применение которого сворачивается и разрешается только в старых системах. В шифре 3DES открытый текст шифруется трижды с тремя различными ключами. Стандарт 3DES был предложен для использования в протоколе PPP (RFC 2420) для канального уровня (см. раздел «Протокол PPP» в главе 5). Подробное обсуждение длины ключей и оценку времени и средств, необходимых для взлома шифра DES, можно найти в [30].

В ноябре 2001 года институт NIST объявил о преемнике стандарта DES — стандарте AES (Advanced Encryption Standard — улучшенный стандарт шифрования) [344], также известном как алгоритм Рийндала [107]. Шифр AES представляет собой алгоритм с симметричным ключом, обрабатывающий данные 128-разрядными блоками, и может работать с ключами длиной по 128, 192 и 256 бит. Национальный институт стандартов и технологии NIST полагает, что машине, способной взломать 56-разрядный шифр DES за одну секунду (то есть перебирающей 2^{55} ключей/с), для взлома 128-разрядного ключа AES потребуется 149 триллионов лет.

Шифрование с открытым ключом

В течение более 2000 лет (со времен Цезаря до 70-х годов XX века) для шифрованной связи требовалось, чтобы две общающиеся стороны хранили общий секрет — для шифрования и дешифрирования использовался один и тот же ключ. Таким образом, обе стороны должны были как-то договориться об общем ключе, но для этого им опять же нужна была безопасная связь! Поэтому обеим сторонам необходимо было сначала лично встретиться и договориться о ключе (например, два центуриона могли встретиться в римских банях), и лишь после этого они получали возможность общаться при помощи шифрованных посланий. Однако в сетевом мире две общающиеся стороны могут никогда не встретиться. Поэтому возникает вопрос: могут ли две стороны обмениваться по сети шифрованными сообщениями, не имея изначально общего секретного ключа? В 1976 году Диффи и Хеллман [122] продемонстрировали алгоритм (называемый теперь алгоритмом обмена ключа Диффи и Хеллмана), обеспечивающий именно такую возможность — принципиально отличный и элегантный метод безопасной связи, приведший к разработке современных криптографических систем с открытым ключом. Далее будет показано, что криптографические системы с открытым ключом обладают замечательными свойствами, благодаря которым могут использоваться не только для шифрования данных, но также для аутен-

тификации и цифровых подписей. Интересно, что идеи, сходные с описываемыми в [122,419], независимо от них высказывались в начале 70-х годов в ряде секретных отчетов британских исследователей из CESG (Communications-Electronics Security Group — группа безопасности связи и электроники) [93]. Как это часто бывает, замечательные идеи возникают независимо в разных местах.

Концепция шифрования с открытым ключом очень проста. Пусть Алиса хочет связаться с Бобом. Как показано на рис. 7.6, вместо того чтобы использовать один общий ключ (как это делается в системах с симметричными ключами), у Боба (получатель сообщений Алисы) есть два ключа — *открытый ключ,* доступный всем и каждому (в том числе злоумышленнику), и *личный ключ,* известный только Бобу. Для открытого и личного ключей Боба мы будем использовать обозначения $K^{в}$ и $K^{в}$~ соответственно. Для общения с Бобом Алиса зашифровывает свое сообщение *m* при помощи известного (возможно, стандартизованного) алгоритма и открытого ключа Боба. То есть Алиса вычисляет $K^{в}\{m\}$. Боб получает зашифрованное Алисой сообщение и расшифровывает его известным (например, стандартизованным) алгоритмом дешифрирования с помощью своего личного ключа. То есть Боб вычисляет $K^{в}\sim(K^{в+}(m))$. Далее будет показано, что существуют алгоритмы шифрования и дешифрирования, а также методы выбора открытого и личного ключей, таких что $K^{в}(K^{в}(m)) = m$; то есть в результате дешифрирования личным ключом сообщения, зашифрованного открытым ключом, мы снова получим исходное сообщение. Это замечательный результат! Таким образом, Алиса может использовать доступный всем и каждому ключ Боба, чтобы посылать Бобу секретные сообщения. При этом отпадает необходимость в передаче секретного ключа! Как вы узнаете далее, можно поменять местами открытый и личный ключи, получив все тот же замечательный результат: $K^{в}\{K^{в}\{m)) = K^{в}\{K^{в}\sim(m)) = m$.

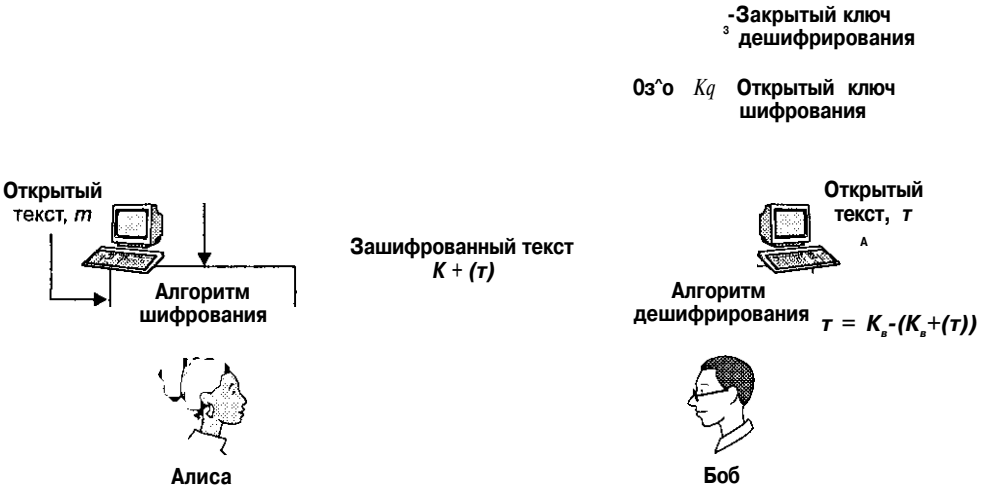


Рис. 7.6. Шифрование с открытым ключом

Итак, концепция шифрования с открытым ключом проста. Однако тут же возникают две проблемы. Во-первых, злоумышленнику, перехватившему сообщение

Алисы, известен открытый ключ Боба и алгоритм шифрования. Таким образом, злоумышленник может предпринять атаку с произвольно выбираемым открытым текстом. То есть злоумышленник может попытаться закодировать открытым ключом предполагаемые варианты сообщений или фрагментов сообщений и сравнить результат с перехваченной шифровкой. Разумеется, система шифрования с открытым ключом должна быть создана так, чтобы злоумышленник не мог по открытому ключу получить личный ключ (или это должно быть настолько трудно, что можно считать невыполнимым), а также сравнивать зашифрованные фрагменты сообщения с сообщением, зашифрованным целиком. Вторая проблема заключается в том, что, поскольку для шифрования используется открытый ключ Боба, то, очевидно, описанная выше простая схема не обеспечивает гарантии подлинности, то есть кто угодно может послать Бобу сообщение от имени Алисы. В системе с симметричными секретными ключами сам факт того, что отправителю известен секретный ключ, удостоверял его личность. В системе же с открытым ключом это не так, поэтому для подтверждения личности отправителя необходима цифровая подпись. Подробнее этот вопрос будет обсуждаться в разделе «Целостность данных».

Хотя в системах шифрования с открытым ключом могут применяться самые разные алгоритмы, один такой алгоритм, *алгоритм RSA* (названный по инициалам его разработчиков, Рона Ривеста, Ади Шамира и Леонарда Адлемана), стал почти синонимом шифрования с открытым ключом. Рассмотрим сначала, как работает алгоритм RSA, а затем поговорим о том, почему он работает. Алгоритм RSA подразумевает два тесно связанных этапа.

1. Выбор открытого и личного ключей.
2. Шифрование и дешифрирование.

Чтобы получить открытый и личный ключи, Боб должен выполнить следующие действия.

1. Выбрать два больших простых числа *p* и *q.* Насколько большими должны быть числа *p* и *q?* Чем больше эти числа, тем труднее взломать шифр RSA, но тем больше времени потребуется для шифрования и дешифрирования. В RSA Laboratories рекомендуют выбирать числа*p* и *q* так, чтобы их произведение было длиной около 1024 бит для корпоративного использования и 768 бит для «информации менее важной» [437] (возможно, кого-то удивит, почему корпоративное использование считается более важным, нежели какое-либо иное!). Тема нахождения больших простых чисел обсуждается в [52].
2. Вычислить

$$n=pqwz=(p - 1)(g - 1).$$

3. Выбрать число *e,* меньшее, чем *n,* у которого нет общих делителей (кроме 1) с числом **2.** (В этом случае говорят, что числа e и z являются относительно простыми.) Буква «e» используется потому, что с нее начинается английское слово «encryption» (шифрование).
4. Найти число *d,* такое чтобы *ed* - 1 без остатка делилось на *г.* Буква «d» используется, потому, что с нее начинается английское слово «decryption» (дешифрирование). Другими словами, при заданном числе *e* мы выбираем число *d* такое,

что остаток от деления *ed* на z равен 1. (Остаток от деления целого числа *x* на целое число *n* обозначается как *x* mod /?.)

5. Открытый ключ $K^{+}_{в}$, доступ к которому Боб предоставляет всему миру, — это пара чисел (я, *e)*\ а личный ключ Боба, $K_{в}$~ — пара чисел *(/?, d).*

Операции шифрования и дешифрирования выполняются следующим образом.

- Предположим, Алиса хочет послать Бобу последовательность битов, или число т, такое, что $m<n$. Зашифрованная последовательность битов, с, получается как остаток от деления m^e на *n:*

$$c = m^e \bmod n.$$

- Чтобы расшифровать полученное сообщение с, Боб вычисляет

$$m = c^{(l} \bmod n,$$

для чего требуется его личный ключ *(n, d).*

Рассмотрим работу алгоритма RSA на простом примере. Пусть Боб выбирает числа $p = 5$ и $q = 7$. (Разумеется, эти значения слишком малы, чтобы обеспечивать безопасность.) В этом случае /? = 5 x 7 = 35,az = 4 x 6 = 24. Далее Боб выбирает число е = 5, так как у чисел 5 и 24 нет общих делителей. Наконец, Боб выбирает число $d = 29$, так как число 5 x 29 - 1 (то есть *ed - 1)* без остатка делится на 24. Боб открывает всем два значения я = 35 и в = 5и сохраняет в секрете число 29. Пусть теперь Алиса хочет послать Бобу символы «l», «o», «v» и «e», зашифровав их открытым ключом Боба. Алиса интерпретирует каждую букву числом в диапазоне от 1 до 26 («a» соответствует 1, а «z» соответствует 26), после чего зашифровывает каждый символ по отдельности. Этапы шифрования *(n* = 35 и *e* = 5) и дешифрирования *(n* = 35 и *d* = 29) показаны в табл. 7.1 и 7.2 соответственно.

Таблица 7.1. Этапы шифрования

Символ открытого текста	т:числовое представление	me	Зашифрованный текст: с = me mod п
l	12	248 832	17
o	15	759 375	15
v	22	5 153 632	22
e	5	3 125	10

Таблица 7.2. Этапы дешифрирования

Зашифрованный текст с	cd	т = = cd mod п	Открытый текст
17	48196857210675091509141182522307169 7	12	l
15	12783403948858939111232757568359375	15	o
22	85164331908653770195619449972110603059 2	22	v
10	10000000000000000000000000000	5	e

Как выбираются большие простые числа? Как затем выбрать числа ***end?*** Каким образом вычислять степени больших чисел? Обсуждение этих важных вопросов выходит за рамки темы этой книги; подробности см. в [261].

Следует заметить, что возведение в степень, применяемое в алгоритме RS A, требует массу процессорного времени. Для сравнения, алгоритм DES работает в 100 раз быстрее в программном исполнении и от 1000 до 10 000 раз быстрее в аппаратной реализации [436]. В результате на практике алгоритм RSA часто применяется в комбинации с алгоритмами DES или AES. Например, если Алиса хочет срочно послать Бобу большое количество зашифрованных данных, она зашифровывает данные алгоритмом DES, для чего использует произвольно выбранный ключ K^s, иногда называемый *ключом сеанса*. Этот ключ Алиса зашифровывает открытым ключом Боба, то есть вычисляет $c = (K^s)^e \bmod n$. Данные, зашифрованные алгоритмом DES, и ключ сеанса, зашифрованный открытым ключом Боба, Алиса пересылает Бобу. Боб расшифровывает ключ сеанса своим открытым ключом, после чего он может расшифровать сами данные.

Описанные операции шифрования и дешифрирования по алгоритму RSA напоминают какой-то фокус. Каков принцип работы этого алгоритма? Чтобы понять, как работает алгоритм RSA, нам нужно познакомиться с арифметическими операциями по модулю *n*. В подобной арифметике выполняются обычные действия сложения, умножения и возведения в степень. Однако результат каждого действия заменяется целым остатком от деления этого результата на *n*. В алгоритме RSA используется число $n = pq$, где p и q являются большими простыми числами.

Вспомним, что при шифровании по алгоритму RSA сообщение (представляемое в виде целого числа) m сначала возводится в степень e при помощи арифметики по модулю *n*. Дешифрирование осуществляется также возведением зашифрованного символа в степень d опять же при помощи арифметики по модулю n. Таким образом, результат выполнения этих двух операций представляет собой $(m^e)^d$. Что можно сказать об этой величине? Итак:

$$(m^e)^d \bmod n = m^{ed} \bmod n.$$

Теория чисел утверждает, что если p и q являются простыми числами и $n = pq$, тогда $x \bmod n = x^{T}<$Кр- Dfa-**1**» $\bmod n$ [261]. Используя этот результат, получаем:

$$(m^e)^d \bmod n = m^{iedmod(p\_1)07\_1\})} \bmod n.$$

Но, как мы помним, мы выбрали числа e и d такие, что $ed - 1$ делится без остатка на $(p - 1)(q - 1)$ или, другими словами, ed делится на $(p - 1)(q - 1)$ с остатком 1, то есть $ed \bmod (p - 1)(q - 1) = 1$. Итак, мы получаем

$$(m^e)^d \bmod n = m^{\{} \bmod n = m.$$

Таким образом:

$$(m^e)^d \bmod n = m.$$

Именно на этот результат мы и рассчитывали! Сначала возводя число m в степень e (то есть зашифровывая), а затем возводя в степень d (то есть расшифровывая), мы

получаем исходное значение *m*. Еще более замечателен тот факт, что если мы сначала возведем число *m* в степень *d,* а затем в степень *e,* то есть выполним операции шифрования и дешифрирования в обратном порядке, мы также получим исходное значение. Далее мы увидим, что это свойство алгоритма RSA оказывается очень полезным.

Безопасность алгоритма RSA основывается на том факте, что алгоритм быстрого определения делителей числа, в данном случае алгоритм нахождения делителей/? и *q* числа *n,* пока не найден. Если вам известны значения/? и *q,* тогда, зная открытое число *e,* нетрудно вычислить секретный ключ *d*. С другой стороны, быстрые алгоритмы нахождения делителей числа всего лишь не найдены, отсутствие таких алгоритмов не доказано, поэтому, строго говоря, безопасность алгоритма RSA не гарантирована.

Аутентификация

Аутентификацией называют процесс подтверждения чьей-либо личности. Люди могут убедиться в личности друг друга различными способами: мы распознаем лица при встрече, а голоса по телефону, сотрудники различных государственных служб могут проверить личность гражданина, сравнив его лицо с фотографией в паспорте.

В этом разделе мы поговорим об аутентификации в сети, ограничившись только аутентификацией при интерактивной связи двух сторон. Мы увидим, что эта проблема несколько отличается от проблемы подтверждения того факта, что сообщение, полученное от некоторого отправителя в прошлом, действительно им отправлено. Эта последняя проблема носит название проблемы *цифровой подписи,* и рассмотрим мы ее в разделе «Целостность данных».

При аутентификации по сети общающиеся стороны не могут полагаться на биометрическую информацию, такую как внешний вид или тембр голоса, В самом деле, как будет показано далее, аутентификация часто требуется сетевым элементам, таким как маршрутизаторы и процессы клиентов и серверов. Таким образом, аутентификация должна осуществляться исключительно на основе сообщений, которыми обмениваются участники *протокола аутентификации.* Как правило, протокол аутентификации запускается *прежде,* чем две общающиеся стороны запустят другой протокол (например, надежный протокол передачи данных, протокол обмена таблицами маршрутизации или протокол электронной почты). Сначала протокол аутентификации устанавливает аутентичность участников сеанса связи, и только после этого участники сеанса получают возможность продолжать сеанс связи.

Как и при разработке протокола надежной передачи данных (см. главу 3), мы разработаем несколько версий протокола аутентификации, который назовем *ap* (authentication protocol — протокол аутентификации), отмечая недостатки каждой версии. Если вам нравится подобный эволюционный подход к разработке, рекомендуем заглянуть на web-сайт http://web.mit.edu/kerberos/www/dialogue.html, где излагается воображаемая история о разработчиках системы аутентификации для

открытой сети, открывающих для себя по ходу разработки все новые и новые проблемы.

Предположим, Алиса должна подтвердить свою личность Бобу.

Протокол аутентификации ap 1.0

Возможно, самым простым протоколом аутентификации, который мы можем себе представить, является такой протокол, при котором Алиса просто посылает Бобу сообщение о том, что она Алиса (рис. 7.7). Недостаток такого протокола очевиден — у Боба нет способа убедиться в том, что тот, кто послал ему сообщение «Я Алиса», действительно Алиса. Точно такое же сообщение может быть послано Бобу злоумышленником.

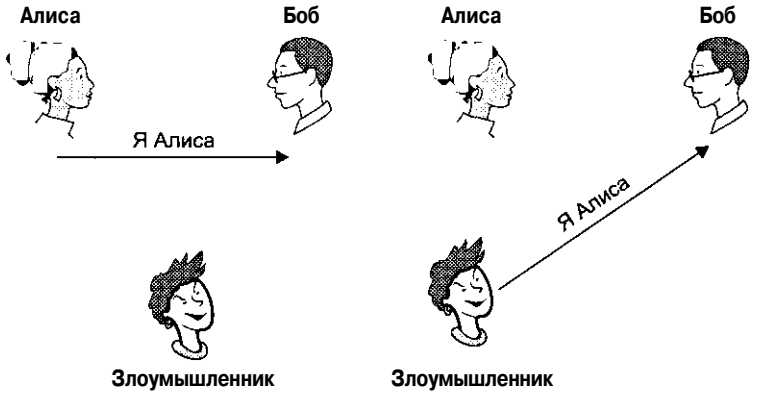


Рис. 7.7. Протокол аутентификации ap 1.0

Протокол аутентификации ap 2.0

В том случае, если у Алисы есть известный сетевой адрес, используемый ею при связи (например, IP-адрес), Боб может попытаться аутентифицировать Алису по ее IP-адресу. Однако это может остановить только очень наивного злоумышленника, но не защитит от целеустремленного студента, читающего эту книгу, а также от многих других!

Поскольку мы уже изучили сетевой и канальный уровни, мы знаем, что узнать сетевой адрес не так уж и трудно (например, если у злоумышленника есть доступ к исходным текстам операционной системы, что, например, характерно для Linux и некоторых других операционных систем, он может построить собственное ядро операционной системы). В этом случае злоумышленник сможет создавать IP-дейтаграммы и указывать в них IP-адрес нужного источника (например, IP-адрес Алисы). Эти дейтаграммы злоумышленник может отправлять в сеть ближайшему маршрутизатору по протоколу канального уровня. С этого момента сеть послушно переправит Бобу дейтаграмму с фиктивным адресом отправителя. Этот метод, который иллюстрирует рис. 7.8, представляет собой хорошо известную разновидность

атаки с поддельным адресом отправителя (см. раздел «Атака и оборона»). Избежать подобной атаки можно, если настроить первый маршрутизатор в сети злоумышленника на отправку только тех дейтаграмм, которые содержат настоящие IP-адреса (см. RFC 2827). Однако такой метод борьбы со злоумышленниками, к сожалению, применяется не везде. Поэтому Бобу не следует полагаться на то, что сетевой администратор злоумышленника (которым, кстати, может быть сам злоумышленник) установил подобную защиту от подделки IP-адресов.

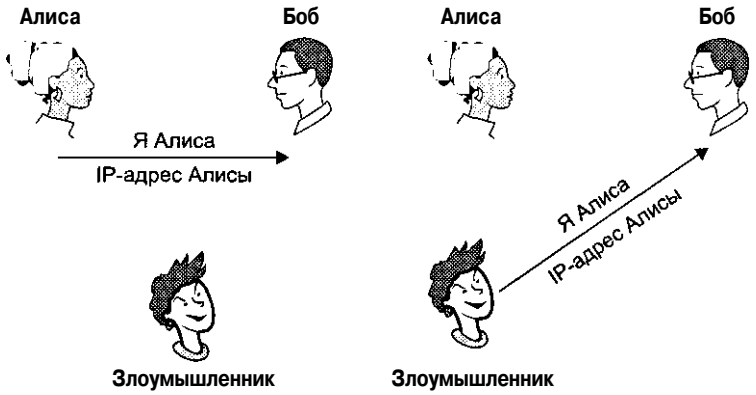


Рис. 7.8. Протокол аутентификации ap 2.0

Протокол аутентификации ap 3.0

Классический подход к аутентификации состоит в использовании паролей. У нас есть PIN-коды, позволяющие подтвердить нашу личность торговым автоматам, и пароли для входа в операционные системы. Пароль хранится в тайне от всех, и знают его только два участника процесса аутентификации — тот, кто подтверждает свою личность, и тот, кто ее проверяет. В подразделе «Взаимодействие пользователя с сервером» раздела «Web и HTTP» главы 2 было показано, что в протоколе HTTP применяется схема парольной аутентификации. Такая же схема аутентификации используется в протоколах FTP и TELNET. Таким образом, в протоколе аутентификации ap 3.0 Алиса посылает Бобу свой секретный пароль (рис. 7.9).

Поскольку пароли так широко применяются, мы можем предположить, что протокол аутентификации ap 3.0 надежен. Однако это не так. И взломать этот протокол совсем несложно. Злоумышленнику нужно всего лишь перехватить сообщение, посылаемое Алисой, чтобы узнать ее пароль. При использовании протокола TELNET для работы на удаленном сервере пароль также пересылается в открытом виде. С любого компьютера, подключенного к локальной сети TELNET-клиента и локальной сети сервера, можно перехватывать все пакеты, посылаемые по этой локальной сети, и, следовательно, украсть пересылаемый в открытом виде пароль. Этот способ кражи паролей давно и широко известен [252]. Подобная угроза вполне реальна, поэтому протокол аутентификации ap 3.0, очевидно, не годится.

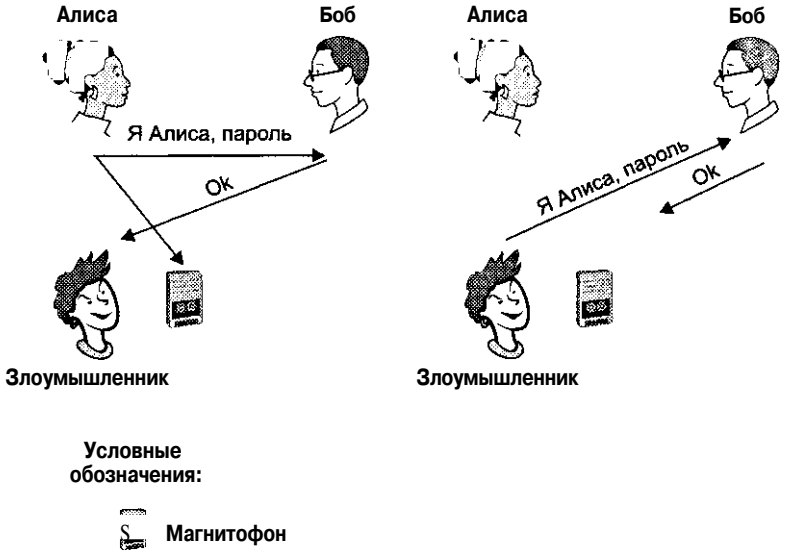


Рис. 7.9. Протокол аутентификации ap 3.0

Протокол аутентификации ap 3.1

Таким образом, очевидно, пересылаемый пароль необходимо зашифровывать. Тем самым мы не позволим злоумышленнику узнать пароль. Если предположить, что Алиса и Боб пользуются общим симметричным ключом K_{A-B}, тогда Алиса может зашифровать пароль и послать его Бобу. Затем Боб расшифрует пароль и, при условии, что пароль верный, убедится, что этот пакет действительно послан Алисой. Боб уверен в том, что это Алиса, так как она не только знает пароль, но также знает общий секретный ключ, необходимый для шифрования пароля. Назовем этот протокол ap 3.1.

И хотя протокол аутентификации ap 3.1 действительно не позволяет злоумышленнику узнать пароль Алисы, он не решает проблемы аутентификации, так как злоумышленнику все равно, в каком виде, зашифрованном или открытом, перехватить пароль Алисы, а потом повторить его в своем пакете. Подобная разновидность атаки называется *атакой повторного воспроизведения*. Таким образом, ситуация практически не отличается от случая использования протокола аутентификации ap 3.0.

Протокол аутентификации ap 4.0

Недостаток протокола аутентификации ap 3.1 заключается в том, что пароль не меняется. Один из методов решения проблемы состоит в том, чтобы каждый раз использовать новый пароль. Алиса и Боб могут договориться о последовательности отправки паролей (или алгоритме генерации паролей) и применять каждый пароль из последовательности всего один раз. Эта идея реализована в системе S/KEY (RFC 1760), в которой для генерации последовательности паролей применяется метод Лампорта [290].

Однако вместо того, чтобы остановиться на этом решении, рассмотрим более общий метод борьбы с атаками повторного воспроизведения. Слабость протокола ap 3.0 вызвана тем фактом, что Боб не мог отличить оригинальный пароль Алисы от его копии, предъявленной злоумышленником. Внимательный читатель вспомнит, что для решения той же проблемы протоколу TCP требуется процедура тройного рукопожатия — сервер не устанавливает TCP-соединения, если полученный SYN-сегмент является старой копией (повторной передачей) SYN-сегмента из предыдущего соединения. Как сервер определяет, что клиент «еще жив»? Он выбирает начальный порядковый номер, не использовавшийся в течение долгого времени, посылает этот номер клиенту, а затем ждет ответа клиента с этим номером. Мы можем задействовать ту же идею для аутентификации.

Число, используемое протоколом всего один раз, называют *поясом*. Наш протокол ap 4.0 использует ноне следующим образом.

1. Алиса отправляет Бобу сообщение «Я Алиса».
2. Боб выбирает ноне R и посылает его Алисе.
3. Алиса зашифровывает ноне с помощью симметричного секретного ключа $K^{A}\_^{B}$, совместно используемого Алисой и Бобом, и посылает зашифрованный ноне $K^{A}\_^{B}(R)$ обратно Бобу. Как и в протоколе ap 3.1, Боб понимает, что полученное им сообщение послано Алисой, потому что отправителю этого сообщения известен общий для Алисы и Боба ключ $K^{A}\_^{B}$, а ноне гарантирует, что это сообщение является оригиналом, а не копией.
4. Боб расшифровывает полученное сообщение. Если расшифрованный ноне совпадает с тем, который он послал Алисе, то аутентификация Алисы считается успешной.

Работу протокола ap 4.0 иллюстрирует рис. 7.10.

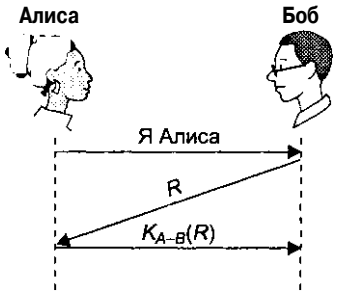


Рис. 7.10. Протокол аутентификации ap 4.0

Протокол аутентификации ap 5.0

В основе успешной работы протокола аутентификации ap 4.0 лежат ноне и шифрование с симметричным ключом. Естественный вопрос заключается в том, можем ли мы для решения проблемы аутентификации использовать нонсы и шифрование с открытым ключом (вместо шифрования с симметричным ключом). Шифро-

вание с открытым ключом позволило бы устранить недостаток любой системы шифрования с симметричными ключами, заключающийся в проблеме передачи секретного ключа, который должны знать две стороны. Протокол шифрования с открытым ключом мы назовем протоколом аутентификации ap 5.0.

1. Алиса посылает Бобу сообщение «Я Алиса».
2. Боб выбирает ноне *R* и посылает его Алисе. Как и в предыдущем протоколе, ноне должен защитить от атаки повторного воспроизведения.
3. Алиса зашифровывает ноне с помощью своего *личного* ключа K^A и посылает зашифрованный ноне $K^A{\sim}(R)$ обратно Бобу. Боб понимает, что полученное им сообщение послано Алисой, потому что отправителю этого сообщения известен личный ключ Алисы K^A, а никто кроме Алисы не может генерировать значение $K^A{\sim}(R)$.
4. Боб расшифровывает полученное сообщение открытым ключом Алисы $K^{A+}(R)$; то есть вычисляет $K^{A+}(K^A{\sim}(R))$. Как отмечалось в разделе «Принципы криптографии», $K^{A+}(K^A{\sim}(R)) = R$. Таким образом, Боб вычисляет *R* и убеждается, что сообщение действительно отправлено Алисой.

Работу протокола аутентификации ap 5.0 иллюстрирует рис. 7.11. Так ли безопасен протокол ap 5.0, как протокол ap 4.0? В обоих протоколах используются нонсы. Поскольку в протоколе ap 5.0 выполняется шифрование с открытым ключом, Боб должен получить открытый ключ Алисы. В результате возможен интересный сценарий, показанный на рис. 7.12, в котором злоумышленник может выдать себя за Алису.

1. Злоумышленник посылает Бобу сообщение «Я Алиса».
2. Боб выбирает ноне *R* и посылает его Алисе, но это сообщение перехватывается злоумышленником.
3. Злоумышленник зашифровывает ноне с помощью своего *личного* ключа $K^m{\sim}$ и посылает зашифрованный ноне $K^T{\sim}(R)$ Бобу.
4. Теперь Боб должен получить открытый ключ Алисы K^{A+} > чтобы с его помощью расшифровать полученное значение. Боб посылает Алисе сообщение с просьбой выслать ему ее открытый ключ *K/* (Боб может также получить открытый ключ Алисы на ее web-сайте). Злоумышленник перехватывает и это сообщение и возвращает Бобу свой открытый ключ iC^{r+}. Боб вычисляет $K^{T+}(K^T{\sim}(R)) = R$ и принимает злоумышленника за Алису!

Из приведенного выше сценария видно, что протокол ap 5.0 безопасен ровно настолько, насколько безопасна процедура передачи открытого ключа. К счастью, существуют безопасные способы передачи открытого ключа, которые рассматриваются в разделе «Передача ключей и сертификация».

В сценарии на рис. 7.12 Боб и Алиса могут заметить, что что-то случилось, если Боб как-то объявит Алисе о своем контакте с ней. Однако злоумышленник может применить еще более хитрую тактику и избежать обнаружения. В сценарии, показанном на рис. 7.13, Алиса и Боб общаются друг с другом, но, используя ту же самую дыру в системе защиты, злоумышленник *прозрачно* вклинивается между Алисой и Бобом. В частности, если Боб пересылает Алисе данные, зашифрованные

ключом, полученным от злоумышленника, злоумышленник может прочитать содержимое сообщений и переправить их дальше Алисе, зашифровав открытым ключом Алисы.

При таком сценарии Алиса и Боб могут общаться, не подозревая о том, что злоумышленнику удалось расшифровать их переписку. И даже последующая личная встреча Алисы и Боба не позволит им обнаружить факт утечки информации, так как во время сетевого сеанса Алиса получила от Боба именно те данные, которые посылал Боб. Такая разновидность атаки называется *атакой с человеком посредине.* Иногда ее также называют *атакой бригады пожарников,* так как злоумышленник передает данные от Боба Алисе и обратно, подобно тому как на пожаре цепочка людей передает друг другу ведра с водой.

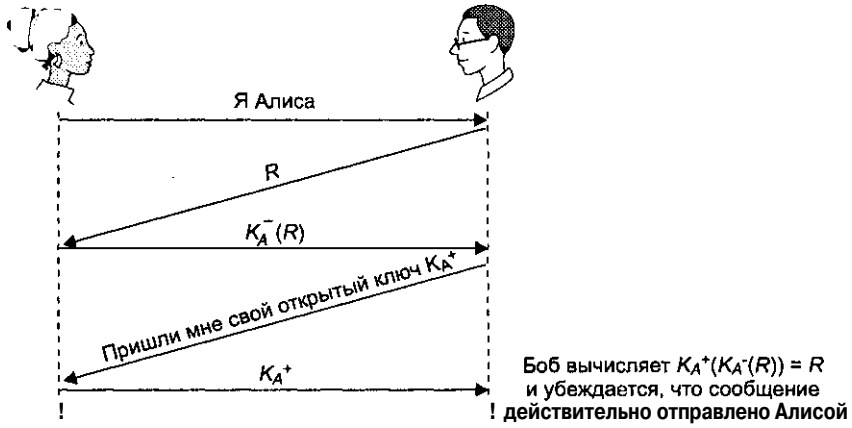


Рис. 7.11. Корректная работа протокола аутентификации ap 5.0

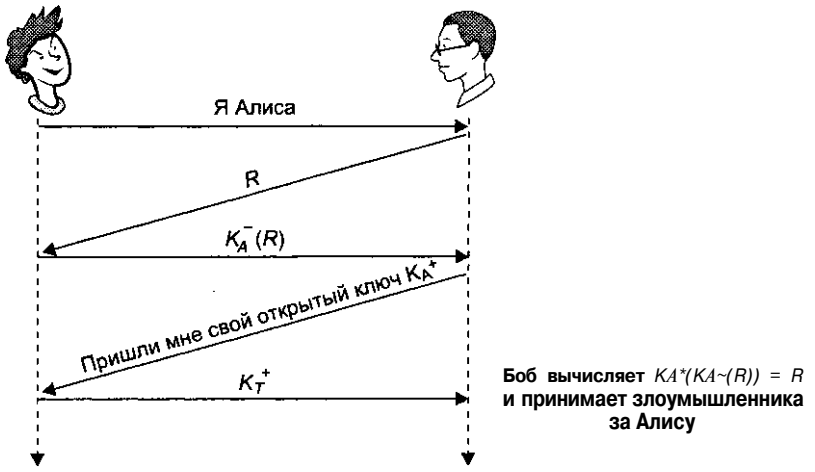


Рис. 7.12. Дыра в системе безопасности протокола аутентификации ap 5.0

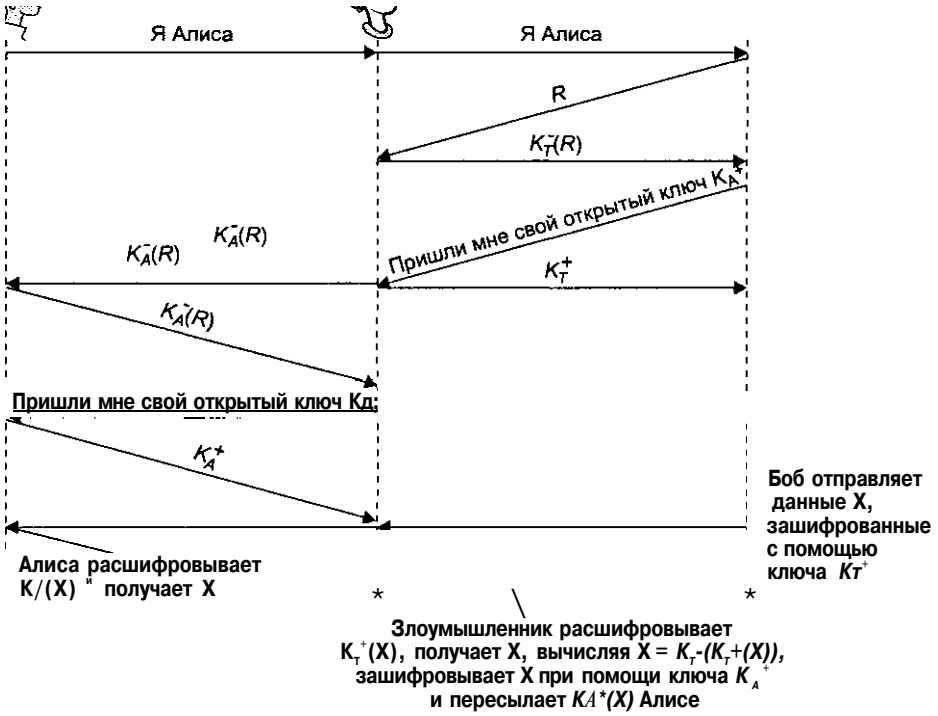


Рис. 7.13. Атака с человеком посредине

Целостность данных

Вспомните, сколько раз вы ставили свою подпись на различных' документах в течение последней недели. Вы подписываете чеки, квитанции о получении кредитных карт, письма и прочие документы. Ваша подпись подтверждает факт, что вы (а не кто-либо иной) ознакомились и/или согласились с содержимым документа. В цифровом мире также часто возникает необходимость указать автора или согласиться с содержимым документа. Для этого в цифровом мире применяется основанный на криптографии метод, называемый *цифровой подписью*.

Как и обычная подпись, цифровая подпись должна быть выполнена таким образом, чтобы ее можно было проверить, чтобы ее невозможно было подделать и чтобы от нее невозможно было отречься. Таким образом, цифровая подпись призвана доказать, что документ подписан действительно тем, кем он подписан, и только он мог его подписать (подпись должно быть невозможно подделать, и подписавшаяся сторона не должна иметь возможности заявить, что она не подписывала документ). Эти задачи легко могут быть решены методами шифрования с открытым ключом.

Генерирование цифровой подписи

Предположим, Боб хочет снабдить цифровой подписью документ *m*. Этим документом может быть файл или сообщение, которое Боб собирается послать. Как показано на рис. 7.14, чтобы подписать этот документ, Боб просто вычисляет значение $K^{в}(m)$ с помощью своего личного ключа $K^{в}{\sim}$. Таким образом, проверить соответствие значения $K^{в}{\sim}(m)$ документу *m* может кто угодно, но сгенерировать $K^{в'}(m)$ может только Боб, что и требуется для подписи. На первый взгляд может показаться странным, что Боб использует свой личный, а не открытый ключ. Однако в данной ситуации цель Боба является диаметрально противоположной шифрованию документа. В данном случае необходимо не скрыть данные, а подписать их.

Удовлетворяет ли цифровая подпись $K^{в}(m)$ нашим требованиям? Предположим, у Алисы есть документ *m* и цифровая подпись $K^{в}{\sim}(m)$. Она хочет доказать в суде, что Боб действительно подписал этот документ и что он единственный, кто мог его подписать. Алиса расшифровывает цифровую подпись при помощи открытого ключа Боба $K^{в}{>}$ то есть вычисляет $K^{в+}(K^{в}{\sim}(m))$. В результате она получает документ *m,* точно совпадающий с оригиналом. Затем Алиса заявляет, что только Боб мог подписать этот документ по следующим причинам.

- Тот, кто подписал этот документ, должен знать личный ключ Боба $K^{в}{\sim}$, чтобы вычислить значение $K^{в}{\sim}(m)$, такое что $K^{в+}(K^{в}{\sim}(m)) = m$.
- Единственный человек, которому известен личный ключ Боба $K^{в}{\sim}$, это сам Боб. Как отмечалось в разделе «Принципы криптографии», знание открытого ключа $K^{в}$ не поможет узнать личный ключ Боба $K^{в}$. Таким образом, единственным человеком, который может знать личный ключ Боба $K^{в>}$ может быть тот человек, который сгенерировал пару ключей $(K^{в+}, K^{в}{\sim})$. Предполагается, что Боб никому не давал свой личный ключ $K^{в}{\sim}$, и никто не мог украсть личный ключ Боба $K^{в}{\sim}$.

Также необходимо отметить, что если изменить оригинальный документ *m,* преобразовав его к виду *m* ', созданная Бобом подпись не будет действительной для модифицированного документа *m*', так как $K^{в+}(K^{в}{\sim}(m))$ не равно *m*

Сообщение: *т*

Дорогая Алиса, извини, я так долго не мог тебе написать. С тех пор как мы...

Алгоритм шифрования

Подписанное сообщение: *Кв'(т)*

fadfg54986fgnzmcnv
T98734ngldskg02j
ser09tugkjdflg

Боб

Личный ключ Боба *Кв'*

Рис. 7.14. Создание цифровой подписи документа

Итак, мы убедились, что методы шифрования с открытым ключом предоставляют простой и элегантный способ создания цифровой подписи документа, подписи, которую можно проверить, невозможно подделать и, следовательно, от которой нельзя отречься. Кроме того, эта же подпись защищает документ от последующей модификации.

Дайджест сообщения

Мы обсудили вопросы использования технологии шифрования с открытым ключом для создания цифровой подписи. Однако шифрование и дешифрирование требуют больших вычислительных затрат. Когда вы подписываете действительно важный документ, например соглашение об объединении двух больших транснациональных корпораций или договор с ребенком о том, что он обязуется еженедельно убирать свою комнату, затраты на вычисления значения не имеют. Однако многие сетевые устройства и процессы (например, маршрутизаторы, обменивающиеся информацией таблиц маршрутизации, почтовые агенты, обменивающиеся электронной почтой) постоянно обмениваются данными, которые шифровать не нужно. Тем не менее им необходимо гарантировать, что:

- отправителем данных является то лицо, которое подписало данные;
- переданные данные не были изменены после того, как отправитель их создал и подписал.

Учитывая накладные расходы по шифрованию и дешифрированию, полноценное шифрование при создании цифровой подписи может быть излишним. Более эффективный подход состоит в вычислении так называемого дайджеста сообщения.

Дайджест сообщения во многом напоминает контрольную сумму. Алгоритм дайджеста вычисляет по сообщению *m* некий блок данных фиксированной длины, представляющий собой как бы «отпечаток пальца» сообщения *H(m).* Дайджест сообщения защищает данные от изменения, так как дайджест измененного сообщения *Щт*)* не будет совпадать с дайджестом оригинального сообщения *H(m).* Как может дайджест сообщения использоваться для создания цифровой подписи? Идея заключается в том, что Боб подписывает не весь документ, а только его дайджест, то есть вычисляет не $K^{в}{\sim}(m)$, а $K^{в}{\sim}(H(m))$. Для этого требуется, чтобы сообщение *m* и электронная подпись $K^{в}{\sim}(H(m))$ вместе обеспечивали невозможность подделки, возможность проверки и невозможность отречения. Невозможность подделки означает, что алгоритм вычисления дайджеста сообщения должен обладать некими особыми свойствами, которые мы рассмотрим далее.

Наше определение дайджеста сообщения может напомнить определение контрольной суммы (например, применяемой в Интернет-протоколах, о чем рассказывалось в подразделе «Контрольная сумма UDP-сегмента» раздела «Протокол UDP — передача без установления соединения» главы 3) или более сложного кода обнаружения ошибок, например, циклического избыточного кода (см. раздел «Обнаружение и исправление ошибок» в главе 5). Есть ли какое-либо отличие дайджеста от данных алгоритмов? Контрольные суммы, циклические многочлены и дайджесты сообщений представляют собой примеры так называемых *хэш-функций.* Как

показано на рис. 7.15, хэш-функция принимает на входе число *m* и вычисляет соответствующую ему строку фиксированной длины, называемую хэшем. Под это определение подходят контрольные суммы Интернет-протоколов, код CRC и дайджесты сообщений. Нам необходимо, чтобы подпись дайджеста сообщения могла заменить подпись всего сообщения. То есть требуется, чтобы подпись дайджеста сообщения было также невозможно подделать, как и цифровую подпись самого сообщения. Соответственно, алгоритм вычисления дайджеста должен удовлетворять следующему требованию: должно быть практически невозможно (из-за объема вычислений) найти два сообщения *x* и *y,* таких что *H(x) = H(y).*

Длинное сообщение: *т*

Дорогая Алиса, это письмо ОЧЕНЬ длинное, так как я должен так много всего сказать...

Дайджест сообщения фиксированной длины *H(т)*

Необратимая хэш-функция

Opgmdvboijrtnsd gghPPdogm;lcvkb

Боб У

Рис. 7.15. Использование хэш-функции для создания дайджеста сообщения

Это требование означает, что злоумышленник не сможет заменить одно сообщение другим так, чтобы подпись дайджеста не нарушилась. То есть, если отправитель создает пару *(m, H(m)),* злоумышленник не сможет подобрать другое сообщение *y* с таким же значением дайджеста, как у оригинального сообщения. Когда Боб подписывает сообщение *m,* вычисляя $K_{б}^{-}(H(m))$, мы знаем, что сообщение *m* не может быть заменено никаким другим сообщением. Более того, цифровая подпись хэш-функции *H(m)* уникально идентифицирует Боба как человека, подписавшего хэш-функцию *H(m),* а следовательно, само сообщение *m,* что отмечалось в подразделе «Генерирование цифровой подписи» данного раздела.

На рис. 7.16 показана схема создания цифровой подписи. Боб пропускает оригинальное длинное сообщение через хэш-функцию, создавая дайджест сообщения. Затем он создает цифровую подпись дайджеста сообщения с помощью своего личного ключа. После этого оригинальное сообщение открытым текстом вместе с цифровой подписью дайджеста сообщения (далее называемой цифровой подписью сообщения) посылается Алисе. На рис. 7.17 показана схема процедуры проверки целостности полученного сообщения. Алиса расшифровывает дайджест сообщения открытым ключом Боба. Кроме того, Алиса получает дайджест сообщения, применяя к самому сообщению хэш-функцию. Если оба дайджеста совпадают, Алиса может быть уверенной в целостности сообщения, а также в его подлинности.

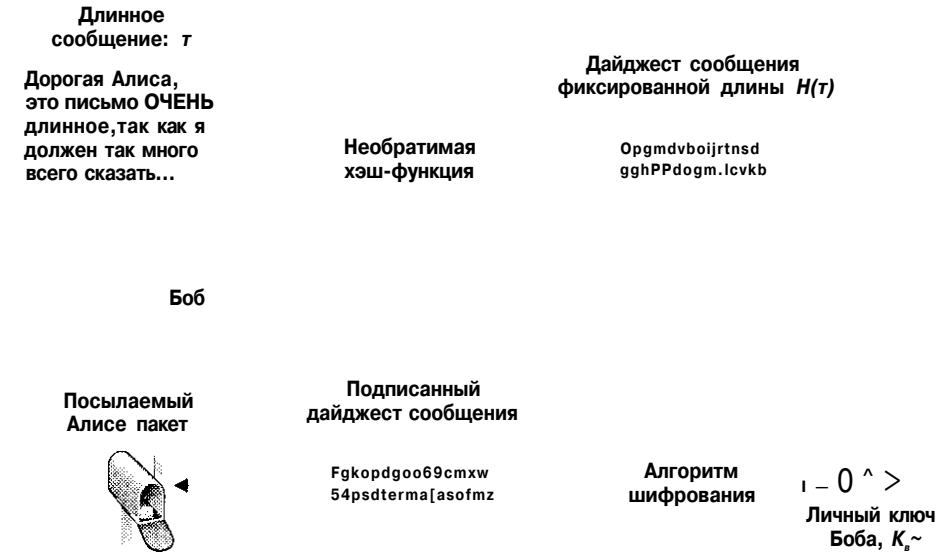


Рис. 7.16. Отправка сообщения, снабженного цифровой подписью

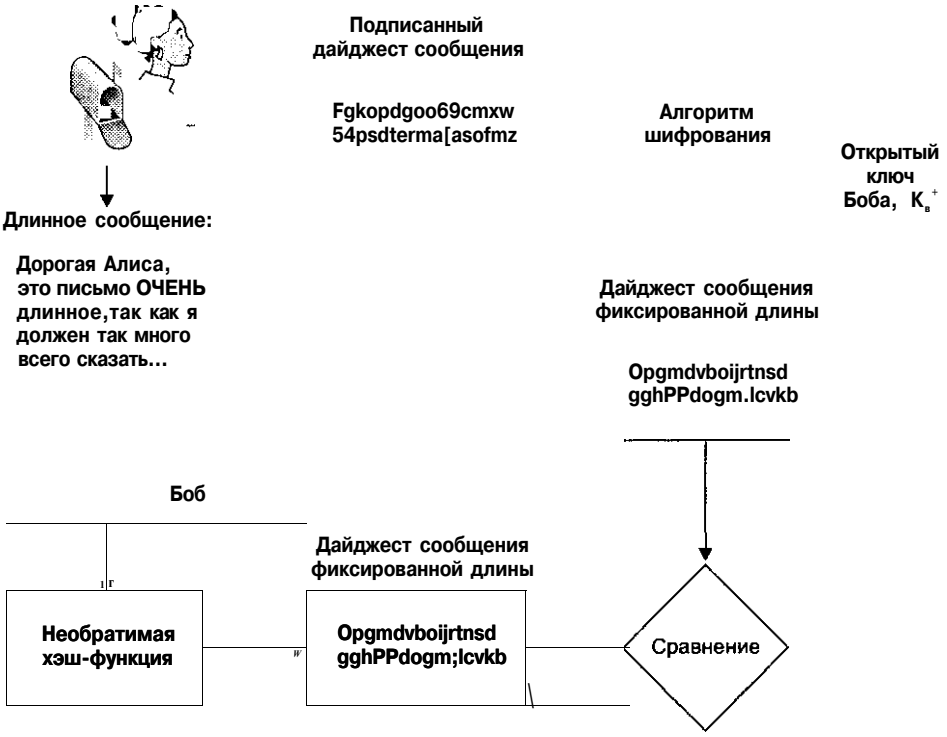


Рис. 7.17. Проверка целостности сообщения, снабженного цифровой подписью

Алгоритмы хэширования

Убедимся в том, что простая контрольная сумма, вроде той, что применяется в Интернет-протоколах, плохо подходит для вычисления дайджеста сообщения. Вместо того чтобы выполнять арифметические действия в дополнительном коде (как в Интернет-протоколах), мы будем вычислять контрольную сумму, обращаясь с каждым символом как с байтом и суммируя все байты блоками по четыре байта. Пусть Боб задолжал Алисе 100 долларов и 99 центов и отправляет ей долговую расписку в виде текстовой строки: «ЮШОО.ЭЭВОВ». В формате ASCII (в шестнадцатеричной нотации) эти символы выглядят следующим образом: 49, 4F, 55, 31,30,30, 2E, 39, 39, 42,4F, 42.

В верхней части рис. 7.18 показано, что четырехбайтовая сумма для этого сообщения равна B2 Cl D2 AC. Немного отличающееся сообщение (со значительно большей суммой) показано в нижней части рисунка. Однако у сообщений «IOU 100.99BOB» и «IOU900.19BOB» *одинаковая* контрольная сумма! Таким образом, этот простой алгоритм нарушает два приводившихся ранее требования к дайджесту сообщения. При использовании данного алгоритма нетрудно найти другое сообщение с той же контрольной суммой. Очевидно, что для обеспечения безопасности нам необходима более мощная, нежели контрольная сумма, хэш-функция.

Сообщение	Представление в кодировке ASCII	
I 0 U 1	49 4F 55 31	
0 0 . 9	30 30 2F 39	**Контрольная**
9 B 0 B	39 42 4F 42	суммэ
	B2 CI D2 AC	

Сообщение	Представление в кодировке ASCII	
I 0 U 9	49 4F 55 39	
0 0 . 1	30 30 2F 31	**Контрольная**
9 B O B	39 42 4F 42	**сумма**
	B2 CI D2 AC	

Рис. 7.18. Оригинальное и поддельное сообщения с той же контрольной суммой

Сегодня получил широкое распространение алгоритм вычисления дайджеста сообщения MD5 (RFC 1321), разработанный Роном Ривестом. Он позволяет вычислять 128-разрядный дайджест сообщения при помощи четырехэтапного процесса. Сначала к сообщению добавляется единица и нули для дополнения длины сообщения до определенного норматива. Затем к сообщению добавляется 64-разрядное представление исходной длины сообщения. Потом инициализируется аккумулятор, и, наконец, сообщение обрабатывается блоками в цикле. Действительно ли алгоритм MD5 удовлетворяет приведенным требованиям, неизвестно. Автор алгоритма утверждает, что, предположительно, сложность получения двух сообщений с одинаковым дайджестом оценивается значением 2^{e4} операций, а сложность

получения сообщения с тем же дайджестом, что и у исходного сообщения, составляет около 2^{128} операций. Это утверждение никем не оспорено. Описание алгоритма MD5 (включая его реализацию на языке C) см. в RFC 1321.

Другим распространенным сегодня алгоритмом вычисления дайджеста является алгоритм SHA-1 (Secure Hash Algorithm — безопасный алгоритм хэширования, версия 1) [143]. Этот алгоритм основан на принципах, сходных с используемыми в алгоритме MD4 (см. RFC 1320), который появился раньше MD5. Алгоритм SHA-1 является федеральным стандартом США. Его предписывается использовать для федеральных приложений, когда требуется надежное вычисление дайджеста. Алгоритм формирует 160-разрядный дайджест сообщения. Благодаря большей длине дайджеста алгоритм SHA-1 считается более надежным.

Передача ключей и сертификация

В разделе «Принципы криптографии» упоминалось о недостатке схемы шифрования с симметричными ключами — двум общающимся сторонам нужно заранее договариваться об общем секретном ключе. В схеме шифрования с открытым ключом необходимость в такой предварительной договоренности отпадает. Однако, как отмечалось в том же разделе, схема шифрования с открытым ключом обладает собственными недостатками, в частности существует проблема получения настоящего открытого ключа. Обе эти проблемы — выбор общего ключа для шифрования с симметричными ключами и безопасное получение открытого ключа при шифровании с открытым ключом — могут быть решены при помощи *доверенных посредников.* В случае шифрования с симметричными ключами такой доверенный посредник называется *центром распределения ключей* (Key Distribution Center, KDC) и представляет собой единый доверенный сетевой орган, с которым другие сетевые объекты устанавливают общий секретный ключ. Мы увидим, что у центра распределения ключей можно получить общие ключи, необходимые для безопасной связи с любыми другими сетевыми объектами, обойдя при этом множество проблем, упомянутых в разделе «Аутентификация». В схемах шифрования с открытым ключом доверенный посредник называется *сертификационным центром* (Certification Authority, C A). Сертификационный центр подтверждает принадлежность открытого ключа определенному сетевому объекту (человеку или организации). Если вы доверяете сертификационному центру, то можете быть уверены в том, кому принадлежит открытый ключ, сертифицированный этим центром. Затем этот сертифицированный ключ может распространяться каким угодно способом, например через сервер открытых ключей, личную web-страницу или дискету.

Центр распределения ключей

Предположим, что Боб и Алиса хотят обменяться по сети данными, используя шифрование с симметричными ключами. Пусть они никогда не встречались друг с другом и, таким образом, не могли заранее договориться об общем ключе. Как им согласовать общий ключ, учитывая, что они могут общаться только по сети? В таких случаях часто используют доверенный центр распределения ключей.

Центр распределения ключей представляет собой сервер, совместно использующий симметричные секретные ключи со всеми своими зарегистрированными пользователями. Эти ключи могут согласовываться с сервером вручную при первой регистрации пользователя. Центру распределения ключей известен секретный ключ каждого пользователя, и каждый пользователь может безопасно общаться с центром распределения ключей при помощи этого ключа. Посмотрим, как знание этого ключа позволит пользователю безопасным образом получить ключ для общения с другим зарегистрированным пользователем. Предположим, Алиса и Боб являются пользователями центра распределения ключей. Им известны только их собственные ключи K^{A_KDC} и K^{B_KDC}. Алиса начинает процесс (рис. 7.19).

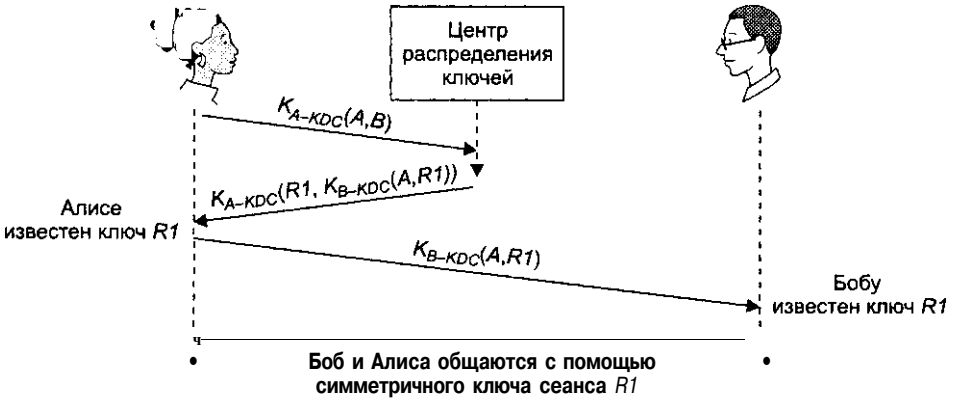


Рис. 7.19. Согласование одноразового ключа сеанса с помощью центра распределения ключей

1. Используя ключ $K^{A.KDC}$ для шифрования обмена данными с центром распределения ключей, Алиса посылает центру сообщение, в котором указывает, что она *(A)* хочет установить связь с Бобом *(B).* Это сообщение обозначим как K^{A_KDC} *(A, B).*
2. Зная ключ K^{A_KDO} центр распределения ключей расшифровывает^ сообщение $_{KA-KDC}$ (А *B).* Затем центр распределения ключей генерирует случайное число *R1.* Это значение общего ключа, которым будут пользоваться Алиса и Боб для симметричного шифрования сообщений друг другу. Этот ключ называется *одноразовым ключом сеанса,* так как Алиса и Боб будут использовать его только в текущем сеансе. Чтобы сообщить Алисе и Бобу значение *R1,* центр распределения ключей посылает Алисе зашифрованное ключом *K^юс* сообщение, содержащее следующие данные.
   - Одноразовый ключ сеанса *R1,* которым будут пользоваться Алиса и Боб.
   - Пару значений *А* и *R1,* зашифрованных центром распределения ключей при помощи ключа Боба K^{B_KDC}. Это сообщение мы будем обозначать как $_{KB-KDC}$ (А Важно отметить, что центр распределения ключей посылает Алисе не только значение *R1,* но также значение *R1* и имя Алисы, зашифрованные при помощи ключа Боба. Алиса не может расшифровать эту пару значений в сообщении (ей не известен ключ Боба), но ей это и не требуется.
   - „ Далее будет показано, что Алиса просто переправляет эту зашифрованную пару значений Бобу, который может их расшифровать.

Центр распределения ключей помещает эти значения в сообщение, зашифровав их при помощи ключа Алисы, и посылает это сообщение Алисе. Таким образом, сообщение, посылаемое центром распределения ключей Алисе, - это $K^{A}\_^{KDC}(R1, K^{B}\_^{KDC}$ (А *R1))*.

Алиса получает сообщение от центра распределения ключей, расшифровывает его и извлекает из него значение *R1.* Теперь Алисе известен одноразовый ключ сеанса *R1.* Кроме того, Алиса извлекает сообщение $K^{B}\_^{KDC}$ *(A, R1)* и переправляет его Бобу.

Боб расшифровывает полученное сообщение $K^{B}\_^{KDC}$ (Л, *R1)* с помощью ключа $K^{B}\_^{KDC}$ и извлекает из него *А* и *R1.* Теперь Бобу известен ключ сеанса *R1* и тот, с кем он будет обмениваться зашифрованными этим ключом данными. Разумеется, прежде чем продолжить обмен данными, Боб выполняет с Алисой процедуру аутентификации (проверяя ее личность) с помощью только что полученного ключа сеанса *R1.*

Сертификация открытых ключей

Одна из основных особенностей криптографии с открытым ключом заключается в том, что два участника сеанса связи могут обмениваться секретными сообщениями без обмена секретными ключами. Например, когда Алиса хочет послать Бобу секретное сообщение, она просто зашифровывает его открытым ключом Боба и посылает это сообщение Бобу. Ей не нужно знать личный ключ Боба, а Бобу не нужно знать ее личный ключ. Таким образом, при шифровании с открытым ключом отпадает необходимость в центре распределения ключей.

Разумеется, в данной ситуации участникам сеанса связи все равно приходится обмениваться открытыми ключами. Пользователь может опубликовать свой открытый ключ разными способами, например разместить его на своей личной web-странице, «положить» на сервер открытых ключей или послать по электронной почте. Коммерческий web-сайт может разместить свои открытые ключи на своем сервере таким образом, что, соединяясь с сайтом, браузер будет автоматически загружать открытый ключ. Маршрутизаторы могут размещать свои открытые ключи на сервере открытых ключей, тем самым, предоставляя доступ к ним и другим сетевым объектам.

Однако и у шифрования с открытым ключом есть своя проблема. Чтобы заглянуть в глубь этой проблемы, рассмотрим пример торговли через Интернет. Предположим, Алиса занимается доставкой пиццы, принимая заказы по Интернету. Любитель пиццы Боб посылает Алисе открытым текстом сообщение, в котором указывает свой домашний адрес и сорт пиццы, который ему нужен. В это сообщение Боб также помещает цифровую подпись (то есть подписанный дайджест оригинального сообщения). Как отмечалось в разделе «Целостность данных», Алиса может получить открытый ключ Боба (с его личной web-страницы, с сервера открытых ключей или по электронной почте) и проверить цифровую подпись. Таким образом она убеждается в том, что это Боб, а не какой-либо юный шутник заполнил заказ.

ПРИНЦИПЫ И ПРАКТИКА

Kerberos[278, 354] представляет собой разработанную в Массачусетском технологическом институте службу аутентификации, в которой используются методы шифрования с симметричными ключами и центр распределения ключей. Хотя концептуально система Kerberos не отличается от описываемого в разделе «Передача ключей и сертификация» центра распределения ключей, терминология здесь несколько иная. Система Kerberos также содержит несколько модификаций и расширений базовых механизмов центра распределения ключей. Она была разработана для аутентификации пользователей, получающих доступ к сетевым серверам, и изначально предназначалась для использования в пределах одного административного домена, например компании или университета. Таким образом, Kerberos обслуживает пользователей, которым необходим доступ к сетевым службам (серверам) при помощи сетевых программ прикладного уровня, таких как Telnet (для регистрации на удаленном сервере) и NFS (для доступа к удаленным файлам), а не для клиентов обычного сетевого соединения, желающих удостовериться в личности друг друга, как в наших предыдущих примерах. Тем не менее ключевые методы остаются теми же.

Сервер аутентификации системы Kerberos играет роль центра распределения ключей. Сервер аутентификации хранит не только секретные ключи всех пользователей (поэтому каждый пользователь может безопасно общаться с самим сервером), но также информацию о том, кто из пользователей обладает привилегиями доступа к определенным сетевым службам. Когда Алиса хочет получить доступ к службе Боба (которого мы теперь будем считать сервером), протокол выполняет действия, аналогичные тем, что показаны в примере на рис. 7.19.

1. Алиса связывается с сервером аутентификации, указывая, что она хочет воспользоваться службой Боба. Все данные, которыми обменивается Алиса и сервер аутентификации, шифруются при помощи секретного ключа, общего для Алисы и сервера аутентификации. В системе Kerberos Алиса сначала сообщает свое имя и пароль локальному хосту. Затем локальный хост Алисы и сервер аутентификации определяют одноразовый секретный ключ сеанса для шифрования данных, которыми будут обмениваться Алиса и сервер аутентификации.
2. Сервер аутентификации подтверждает личность Алисы, проверяет наличие у нее прав доступа к службам Боба и генерирует одноразовый симметричный ключ сеанса R1 для связи между Алисой и Бобом. Сервер аутентификации, в системе Kerberos называемый сервером предоставления билетов (Ticket Granting Server), посылает Алисе значение R1, а также билет для служб Боба. В билете указываются имя Алисы, ключ сеанса между Алисой и Бобом R1, а также срок действия билета. Вся эта информация зашифровывается секретным ключом Боба (известным только Бобу и серверу аутентификации), как показано на рис. 7.19. Билет действителен только на протяжении указанного срока и будет отвергнут Бобом, если его предъявят после этого срока. В системе Kerberos V4 максимальный срок действия билета составляет около 21 часа. В системе Kerberos V5 максимальный срок действия должен истечь до конца 9999 года, что известно как проблема 10 000-го года.
3. Алиса посылает свой билет Бобу. Она также отправляет зашифрованную ключом R1 метку времени, используемую в качестве нонса. Боб расшифровывает билет своим секретным ключом, извлекает из него ключ сеанса и расшифровывает им метку времени. Боб посылает ноне обратно Алисе, зашифровывая его ключом R1, тем самым демонстрируя, что Бобу известен ключ R1 и, следовательно, Боб «жив».

Последняя версия Kerberos (Kerberos V5) поддерживает несколько серверов аутентификации, делегирование прав доступа и многоразовые билеты. За подробностями обращайтесь в [261, 278].

Все это прекрасно до тех пор, пока не появляется злоумышленник. Как показано на рис. 7.20, злоумышленник решает подшутить. Он посылает Алисе сообщение, в котором объявляет себя Бобом, указывает домашний адрес Боба и заказывает пиццу. Злоумышленник также прилагает к заказу цифровую подпись, выполненную, естественно, при помощи собственного личного ключа. Затем злоумышленник продолжает маскарад, посылая Алисе собственный открытый ключ, утверждая, что он принадлежит Бобу. В данном примере Алиса расшифровывает цифровую подпись открытым ключом злоумышленника (полагая, что это ключ Боба), и приходит к выводу, что сообщение действительно создано Бобом. Вероятно Боб будет очень удивлен, когда ему доставят пиццу!

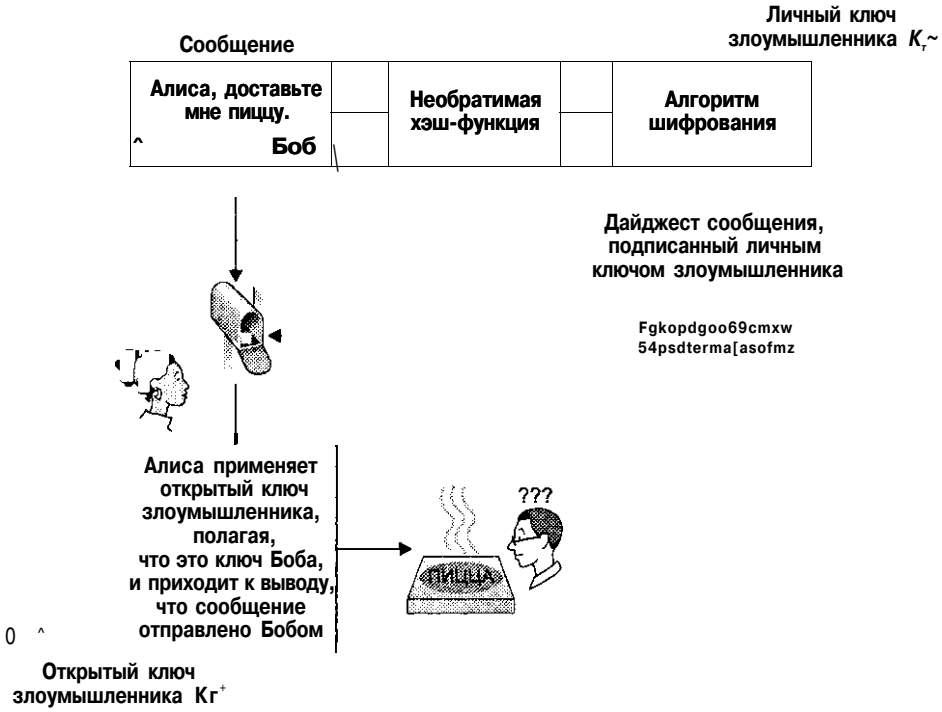


Рис. 7.20. Злоумышленник маскируется под Боба, используя шифрование с открытым ключом

Как явствует из этого примера, чтобы шифрованием с открытым ключом можно было пользоваться, сетевые объекты (пользователи, браузеры, маршрутизаторы и т. д.) должны знать *наверняка,* что у них есть открытый ключ именно объекта, с которым они общаются. Например, когда Алиса связывается с Бобом, используя шифрование с открытым ключом, она должна быть уверена, что у нее в руках действительно открытый ключ Боба. Если вы помните, те же проблемы вставали перед нами при разработке протокола аутентификации (см. рис. 7.12 и 7.13).

Привязка открытого ключа к определенным сетевым объектам, как правило, осуществляется уже упоминавшимся *сертификационным центром* (CA), работа которого заключается в подтверждении подлинности и выдаче сертификатов.

1. Сертификационный центр проверяет, является ли объект (человек, маршрутизатор и т. д.) тем, кем он себя объявляет. Конкретный способ аутентификации никак не оговаривается. Сетевые объекты должны доверять сертификационному центру выполнение строгой процедуры проверки подлинности. Например, если бы злоумышленник мог просто зайти в некий сертификационный центр, заявить «Я — Алиса» и получить сертификат, ассоциированный с личностью под именем «Алиса», тогда можно было бы особо не доверять сертификатам, выданным этим сертификационным центром. С другой стороны, некоторые пользователи склонны (или, наоборот, не склонны) больше доверять сертификационному центру, являющемуся частью федеральной программы (или программы правительства отдельного штата, например штат Юта выдает лицензии сертификационным центрам, расположенным в этом штате [527]). Доверять «личности», ассоциированной с открытым ключом, можно только в той степени, в которой вы доверяете сертификационному центру и его методам проверки.
2. После проверки подлинности объекта сертификационный центр создает *сертификат,* который связывает открытый ключ объекта с идентификатором этого объекта. Сертификат содержит открытый ключ и глобально уникальный идентификатор владельца этого ключа (например, имя человека или его IP-адрес). Сертификат снабжается цифровой подписью сертификационного центра. Эти этапы показаны на рис. 7.21.

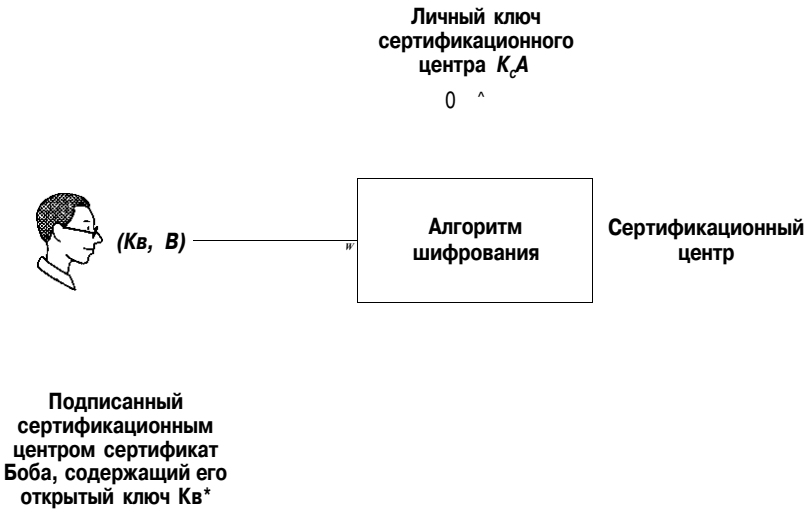


Рис. 7.21. Боб получает сертификат в сертификационном центре

Посмотрим теперь, как сертификаты могут использоваться для борьбы с шутниками, заказывающими пиццу тем, кто ее не любит, а также с другими злоумышленниками. Когда Алисе приходит заказ от Боба, она получает его сертификат, который может размещаться на web-странице Боба, на сертификационном сервере или прилагаться к заказу, посылаемому по электронной почте. С помощью открытого ключа сертификационного центра Алиса проверяет подлинность сертификата. Если мы предположим, что сам открытый ключ сертификационного центра известен всем (например, он может быть опубликован в общеизвестном месте, которому можно доверять, например газета *The New York Times* известна всем, что исключает мистификацию), тогда Алиса может быть уверена, что она на самом деле имеет дело с Бобом. Этапы получения сертификата показаны на рис. 7.21. Вы можете найти сертификаты, хранящиеся в вашем web-браузере Netscape, в меню Communicator • Tools • Security Info • Certificates, а в Internet Explorer в меню Tools • Internet Options • Content • Certificates.

Стандарты для сертификационных центров были разработаны Международным союзом телекоммуникаций (International Telecommunications Union, ITU) и Проблемной группой разработок для Интернета (Internet Engineering Task Force, IETF). Стандарт ITU X.509 [232] описывает службу аутентификации, а также синтаксис сертификатов. В RFC 1422 представлена система управления ключами на основе сертификационных центров для безопасной электронной почты, пересылаемой через Интернет. Этот стандарт совместим с X.509, но он шире, так как устанавливает процедуры и соглашения для архитектуры управления ключами. Некоторые наиболее важные поля сертификата перечислены в табл. 7.3.

Таблица 7.3. Некоторые поля сертификата открытого ключа стандартов X.509 и *RFC* 1422

Название поля	Описание
Версия	Номер версии спецификации стандарта X.509
Порядковый номер	Уникальный идентификатор сертификата, назначаемый сертификационным центром
Подпись	Алгоритм для подписи сертификата, используемый сертификационным центром
Имя эмитента	Идентификатор сертификационного центра, выпустившего данный сертификат (RFC 2253), в формате DN (Distinguished Name — различимое имя)
Срок действия	Время начала и окончания действия сертификата
Имя субъекта	Идентификатор объекта, чей открытый ключ ассоциирован с этим сертификатом в формате DN
Открытый ключ субъекта	Открытый ключ субъекта, а также обозначение алгоритма шифрования с открытым ключом (а также параметры алгоритма), используемого с этим ключом

С ростом популярности электронной коммерции значительно выросли потребность в безопасных транзакциях и интерес к сертификационным центрам. Среди компаний, предоставляющих подобные услуги, можно назвать Digital Signature Trust Company [125] и Verisign [534].

Сертификат, выпущенный корпорацией Thawte Consulting to Netfarmers Enterprises, Inc., в окне браузера Netscape показан на рис. 7.22.

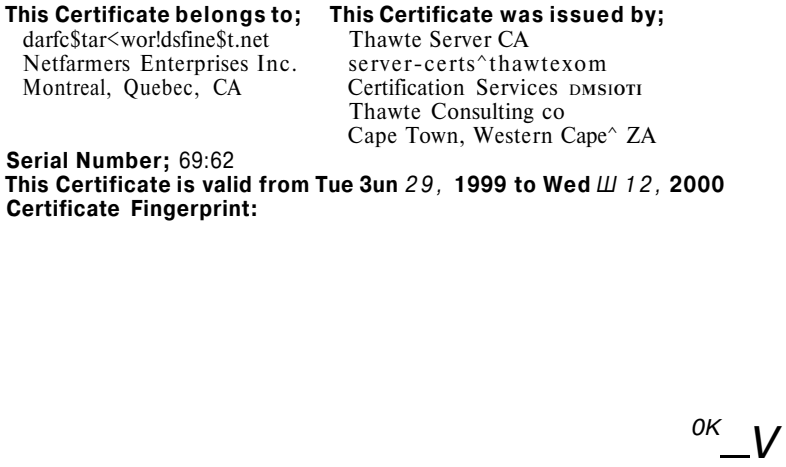


Рис. 7.22. Сертификат корпорации Thawte Consulting to Netfarmers Enterprises, Inc.

Управление доступом с помощью брандмауэров

Интернет — не слишком безопасное место. Злоумышленники могут стать источником самых разнообразных неприятностей. С точки зрения администратора сети мир делится на два лагеря: «хорошие парни» (работающие в организации, занимающейся администрированием сети, и имеющие практически ничем не ограниченный доступ к сетевым ресурсам) и «плохие парни» (все остальные, чей доступ к сетевым ресурсам нужно тщательно регламентировать). В помещениях многих организаций (от средневековых замков до офисов современных корпораций), как правило, имеется один вход (он же выход), через который как «хорошие», так и «плохие» парни могут проникать на территорию организации и покидать ее. При этом у всех входящих и выходящих можно проверить право посещения организации. В замке проверку удобнее всего осуществлять у ворот или на разводном мосту перед воротами; в корпоративном здании проверка выполняется на проходной. В компьютерной сети проверкой входящего и исходящего трафика занимается устройство, называемое брандмауэром.

Брандмауэр представляет собой программно-аппаратное устройство, изолирующее внутреннюю сеть организации от остального мира (Интернета) путем пропуска одних пакетов и блокирования других. Брандмауэр позволяет сетевому администратору регулировать доступ пользователей из внешнего мира к внутренним ресурсам сети, управляя потоками данных, направленных к этим ресур-

сам и обратно. На рис. 7.23 схематично изображен брандмауэр, располагающийся на границе между сетью и остальным Интернетом. Хотя крупные организации могут располагать брандмауэры в несколько уровней (так называемые распределенные брандмауэры [242]), при использовании одного брандмауэра проще управлять входными и выходными потоками, проводя политику безопасного доступа к ресурсам.

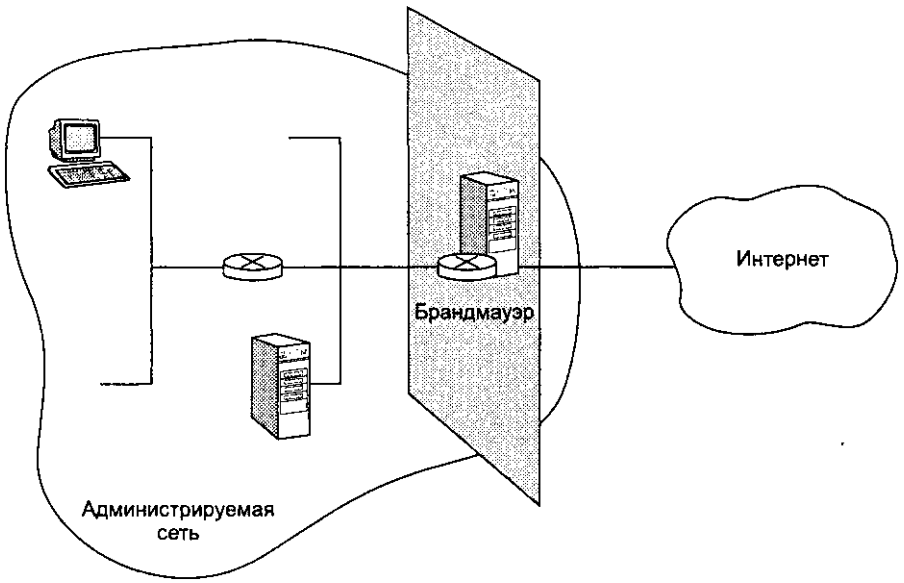


Рис. 7.23. Брандмауэр

Существуют два типа брандмауэров: *брандмауэры, фильтрующие пакеты* (работающие на сетевом уровне), и *шлюзы прикладного уровня* (работающие на прикладном уровне). В следующих двух подразделах мы поочередно рассмотрим оба типа брандмауэров.

Фильтрация пакетов

Как показано на рис. 7.23, у организации, как правило, есть шлюзовый маршрутизатор, соединяющий внутреннюю сеть организации с Интернет-провайдером (и, соответственно, с Интернетом). Весь трафик, поступающий во внутреннюю сеть и покидающий ее, проходит через этот маршрутизатор, и именно на этом маршрутизаторе осуществляется *фильтрация пакетов.* Пакетные фильтры исследуют заголовки дейтаграмм и пропускают или отбрасывают дейтаграммы в соответствии с правилами фильтрации, заданными администратором. Решения, как правило, основываются на:

- IP-адресе отправителя или получателя;
- номере TCP- или UDP-порта отправителя или получателя;

- поле типа сообщения протокола ICMP;
- дейтаграммах инициализации соединения, в которых используются биты SYN или ACK протокола TCP.

Например, фильтр может блокировать все UDP-сегменты и все Telnet-соединения. Подобная конфигурация не позволит находящимся за пределами корпоративной сети пользователям регистрироваться на внутренних хостах, а пользователям самой сети регистрироваться на хостах, находящихся за пределами сети при помощи протокола Telnet, так как брандмауэр будет блокировать все TCP-сегменты (каждый из которых содержится в дейтаграмме), если указанный в них номер порта отправителя или получателя равен 23 (что соответствует протоколу Telnet). Фильтрация UDP-трафика является также популярной политикой, применяемой корпорациями (к большому неудовольствию ведущих производителей сетевых аудио- и видеоприложений, использующих протокол UDP по умолчанию). Фильтрация Telnet-соединений также популярна, так как она не позволяет внешним пользователям регистрироваться на внутренних машинах. На web-сайте http://www.cert.org/tech_tips/packet_filtering.html содержится список рекомендаций по фильтрации комбинаций портов и протоколов, которые могут помочь залатать многие известные на сегодня дыры в системе безопасности существующих сетевых приложений.

Политика фильтрации также может основываться на комбинации IP-адреса и номера порта. Например, фильтрующий маршрутизатор может блокировать все дейтаграммы протокола Telnet (с номером порта 23), кроме тех, чьи IP-адреса отправителей или получателей содержатся в специальном списке. Такая политика позволяет устанавливать Telnet-соединения с хостами (и от хостов), адреса которых содержатся в списке разрешенных адресов. К сожалению, этот метод не защищает от злоумышленников, подделывающих IP-адреса в своих дейтаграммах. Подобные атаки мы рассмотрим в следующем разделе.

Фильтрация также может быть основана на значении бита ACK в TCP-заголовке пакета. Это может быть полезно, если организация разрешает своим сотрудникам соединяться с внешними серверами, но не хочет, чтобы внешние клиенты соединялись с серверами организации. В разделе «Протокол TCP — передача с установлением соединения» главы 3 отмечалось, что в первом сегменте каждого TCP-соединения бит ACK сброшен в 0, тогда как во всех остальных сегментах этот бит установлен в 1. Таким образом, если организация не желает, чтобы внешние клиенты устанавливали соединения с внутренними серверами, она может просто блокировать все входящие сегменты с нулевым битом ACK. Такая политика закроет доступ всем входящим TCP-соединениям, но позволит устанавливать исходящие TCP-соединения.

Хотя по этим примерам может показаться, что формировать правила фильтрации не слишком сложно, в этой области есть множество подводных камней. Чтобы проиллюстрировать некоторые проблемы, рассмотрим простой пример из [68]. Процедура фильтрации пакетов последовательно применяет правила фильтрации к исследуемой дейтаграмме. Предположим, Алиса администрирует корпоративную сеть 222.22.0.0/16 и хочет запретить доступ к ее сети из Интернета (назовем это

правилом R3). Однако Алиса сотрудничает с Бобом и его коллегами из университета. Поэтому Алиса хочет предоставить пользователям из университета Боба (адрес сети которого — 111.11/16) доступ к специальной подсети 222.22.22/24, входящей в сеть ее компании (правило R1). Однако в данной ситуации есть один усложняющий фактор, состоящий в том, что Алисе известно, что в университете Боба есть злоумышленник, адрес подсети которого 111.11.11/24. Поэтому Алиса хочет запретить всякий трафик из этой подсети (правило R2). Правила фильтрации пакетов сведены в табл. 7.4.

Таблица 7.4. Правила фильтрации пакетов

Правило	Адрес отправителя	Адрес получателя	Действие	Комментарии
R1	**111.11/16**	**222.22.22/24**	**Разрешить**	**Пропускать дейтаграммы из сети университета Боба в специальную подсеть**
R2	**111.11.11/24**	**222.22/16**	**Отказать**	**Не пропускать трафик из подсети злоумышленника в сеть Алисы**
R3	**0.0.0.0/0**	**0.0.0.0/0**	**Отказать**	**Не пропускать трафик в сеть Алисы**

В табл. 7.5 показаны примеры обработки дейтаграмм брандмауэром Алисы. При этом в первом случае (предпоследняя колонка таблицы) правила применяются в последовательности R2, R1, R3, то есть в первую очередь проверяются наиболее конкретные адреса (подсеть злоумышленника), затем более широкое множество адресов (сеть университета Боба) и, наконец, все прочие адреса. В этом случае мы получаем желаемый результат. Например, дейтаграммы P1 и P2 блокируются при первой же проверке (правило R2) независимо от того, направляются они в специальную подсеть или в остальную часть корпоративной сети. Дейтаграммы же, отправляемые из университетской сети, но не злоумышленником (дейтаграммы P3 и P4), пропускаются в специальную подсеть (дейтаграмма P3), но не пропускаются в остальную часть корпоративной сети (дейтаграмма P4).

Предположим, однако, что те же правила применяются в последовательности R1, R2, R3 (последняя колонка табл. 7.5). В этом случае посланная злоумышленником дейтаграмма P2 ошибочно пропускается в специальную подсеть, так как правило R1 применяется прежде, чем правило R2. Таким образом, очевидно, что порядок применения правил фильтрации пакетов на брандмауэре имеет значение. Учитывая, что в реальных брандмауэрах применяются тысячи разнообразных правил, нужно быть очень внимательным, продумывая их очередность. Возможно, вы решите, что нужно всего лишь применять более специфические правила в первую очередь. Однако, как мы увидим далее, это не обязательно так (см. упражнения в конце главы).

Таблица 7.5. Результат фильтрации пакетов в соответствии с порядком применения правил

Номер дейтаграммы	IP-адрес отправителя	IP-адрес получателя	Желаемое действие	Действие при R2, R1, R3	Действие при R1, R2, R3
P1	111.11.11.1 (подсеть хакера)	222.22.6.6 (корпоративная сеть)	Отказать	Отказать (R2)	Отказать (R2)
P2	111.11.11.1 (подсеть хакера)	222.22.22.2 (специальная подсеть)	Отказать	Отказать (R2)	Разрешить (R1)
P3	111.11.6.6 (университетская сеть, не подсеть хакера)	222.22.22.2 (специальная подсеть)	Разрешить	Разрешить (R1)	Разрешить (R1)
P4	111.11.6.6 (университетская сеть, не подсеть хакера)	222.22.6.6 (корпоративная сеть)	Отказать	Отказать (R3)	Отказать (R3)

Шлюзы прикладного уровня

В приведенном выше примере мы видели, что фильтрация пакетов позволяет организации грубо разделять потоки данных на основании содержимого заголовков протоколов IP и TCP/UDP, включая IP-адреса, номера портов и биты подтверждения. Например, мы видели, что фильтрация на основе комбинации IP-адреса и номера порта может позволить устанавливать исходящие Telnet-соединения и в то же время, запретить установку входящих Telnet-соединений. Но что если организация хочет предоставить услуги Telnet-соединений лишь ограниченному числу внутренних пользователей? И что, если организация хочет, чтобы эти привилегированные пользователи перед установкой Telnet-соединения проходили процедуру аутентификации? Такие задачи уже не по плечу простому фильтру, работающему на уровне сетевого и транспортного протоколов. В самом деле, информация о личности внутренних пользователей не включается в IP/TCP/UDP-заголовки, а переносится в данных прикладного уровня.

Для обеспечения более высокого уровня безопасности брандмауэры должны объединять функции пакетных фильтров и шлюзов прикладного уровня. *Шлюз прикладного уровня* представляет собой сервер, через который должна проходить вся информация прикладного уровня (входная и выходная). На одном и том же хосте могут работать сразу несколько шлюзов прикладного уровня, но каждый такой шлюз — это отдельный сервер с собственными процессами.

Попробуем спроектировать брандмауэр, разрешающий устанавливать исходящие Telnet-соединения лишь ограниченному множеству внутренних пользователей и не разрешающий устанавливать входящие Telnet-соединения внешним клиентам. Подобная политика может быть реализована путем объединения пакетного фильтра (на маршрутизаторе) и прикладного Telnet-шлюза (рис. 7.24). Фильтр марш-

рутизатора сконфигурирован так, чтобы блокировать все Telnet-соединения, кроме тех, которые поступают из прикладного шлюза, что определяется по IP-адресу прикладного шлюза. Подобная настройка фильтра заставляет все исходящие Telnet-соединения проходить через прикладной шлюз. Рассмотрим теперь внутреннего пользователя, желающего установить Telnet-соединение с внешним миром. Этот пользователь должен сначала установить Telnet-соединение с прикладным шлюзом. Работающее на шлюзе приложение, прослушивающее все входящие Telnet-соединения, запрашивает у пользователя его идентификатор и пароль. Когда пользователь предоставляет эту информацию, прикладной шлюз проверяет, имеет ли данный пользователь разрешение на установку Telnet-соединения с внешним миром. Если разрешения у данного пользователя нет, тогда шлюз разрывает Telnet-соединение. В противном случае шлюз, во-первых, запрашивает у пользователя адрес внешнего хоста, с которым тот хочет установить связь, во-вторых, устанавливает Telnet-соединение между шлюзом и внешним хостом и, во-третьих, переправляет внешнему хосту все данные, поступающие от пользователя, а все данные, поступающие от внешнего хоста, переправляет пользователю. Таким образом, прикладной Telnet-шлюз не только осуществляет аутентификацию, но также выступает в роли Telnet-сервера и Telnet-клиента, ретранслируя все данные между пользователем и удаленным Telnet-сервером. Обратите внимание, что фильтр не запрещает устанавливать Telnet-соединение между шлюзом и внешним хостом, так как оно устанавливается шлюзом.

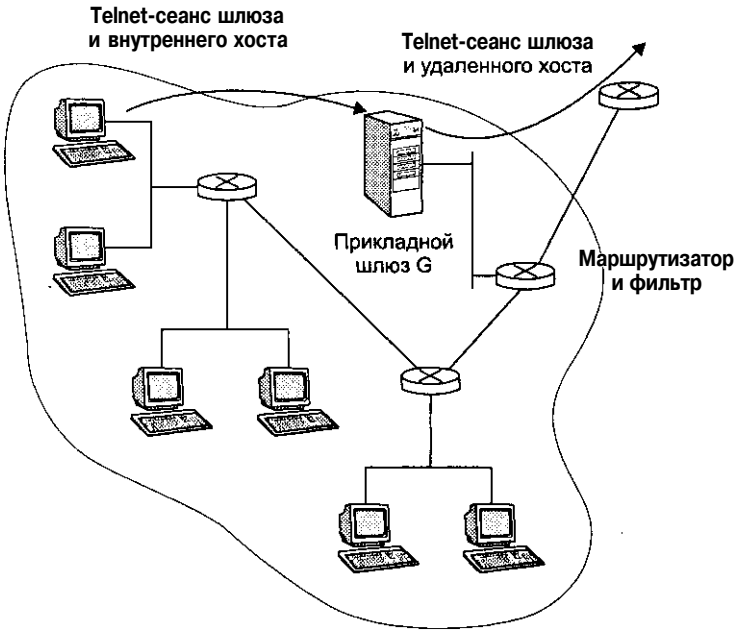


Рис. 7.24. Брандмауэр, состоящий из прикладного шлюза и фильтра

Во внутренних сетях часто работают сразу несколько прикладных шлюзов, например для протоколов Telnet, HTTP, FTP и для электронной почты. Действительно,

почтовый сервер организации (см. раздел «Электронная почта» в главе 2) и web-кэш являются прикладными шлюзами.

Прикладные шлюзы обладают некоторыми недостатками. Во-первых, для каждого приложения требуется отдельный прикладной шлюз. Во-вторых, применение прикладных шлюзов снижает производительность, так как все данные должны проходить через шлюз. Снижение производительности становится особенно заметным, если множество пользователей или приложений в качестве шлюза задействуют один и тот же компьютер. В-третьих, шлюз требует настройки, поскольку программное обеспечение клиента должно знать, как соединяться со шлюзом вместо внешнего сервера и как сообщить прикладному шлюзу, с каким именно внешним сервером необходимо установить связь. Еще один возможный вариант настройки шлюза может иметь место в ситуации, когда пользователю предлагается явно связаться с внешним сервером через прикладной шлюз.

В заключение этого раздела скажем, что брандмауэры ни в коем случае не являются панацеей при решении проблем безопасности. Использование брандмауэров всегда связано с выбором между уровнем открытости и уровнем безопасности. Поскольку фильтры не могут бороться с подделкой IP-адресов и номеров портов, они часто применяются как устройства либо все разрешающие (например, весь UDP-трафик), либо все запрещающие. Шлюзы также могут содержать программные ошибки, позволяющие злоумышленникам их преодолевать. Наконец, эффективность брандмауэра вообще сводится на нет, если пользователи внутри сети получают возможность связываться с внешним миром, минуя брандмауэр, например при помощи модемов и телефонных линий или при помощи беспроводной связи.

Атака и оборона

Итак, мы рассмотрели такие вопросы, как шифрование и дешифрирование, аутентификация, целостность данных, передача ключа и брандмауэры. Поговорим теперь о том, как эти методы могут использоваться для отражения разнообразных сетевых атак. В некоторых получивших широкую известность случаях сетевых атак, таких как атаки червя CodeRed [64] и вируса Melissa [63], Интернет использовался для распространения вирусов, атаке же подверглась не сама сеть, а операционные системы (в случае CodeRed это было переполнение буфера Microsoft IIS) или прикладное программное обеспечение (Microsoft Word в случае вируса Melissa). Поскольку наша книга посвящена сетевым вопросам, мы здесь рассмотрим только те атаки, в которых тем или иным образом затрагивается сеть (атакуется или используется для передачи вируса).

Сбор информации

В «реальном мире» атаке часто предшествует сбор информации. Гангстеры из фильмов изучают место будущего ограбления, солдаты производят разведку. Цель этих действий ясна — чем больше известно о цели перед атакой, тем выше вероятность успеха. Это справедливо и для компьютерного мира. Прежде чем атаковать сеть,

злоумышленникам необходимо узнать, какие IP-адреса у машин этой сети, какие операционные системы на них установлены, какие услуги они предоставляют. С этой информацией их атаки становятся более конкретными и с меньшей вероятностью выявляются системой безопасности.

Для определения IP-адресов машин, подключенных к сети, может использоваться такая команда (программа), как ping. При этом нужно просто наблюдать, какие адреса отвечают на команду ping. Злоумышленники также применяют метод, называемый *сканированием портов* и представляющий собой последовательные попытки соединиться (либо запрашивая установку TCP-соединения, либо просто посылая UDP-дейтаграмму) с различными номерами портов машины и анализ ответов на эти попытки. По ответам можно определить, какие службы (например, HTTP или FTP) работают на опрашиваемой машине. Популярной программой, осуществляющей сканирование портов, является утилита Nmap [360]. Многие брандмауэры, например производимые компанией Checkpoint [71], обнаруживают запросы, выполняемые командой ping, факты сканирования портов, а также другие «враждебные действия» ихообщают о подобной активности сетевому администратору.

Анализатор пакетов

Анализатор пакетов представляет собой программу, работающую на присоединенном к сети устройстве и пассивно получающую *все* информационные кадры канального уровня, проходящие через сетевой адаптер устройства. В широковещательной среде, такой как локальная сеть Ethernet, это означает, что анализатор пакетов получает все пакеты, отправляемые всеми хостами локальной сети, а также все пакеты, получаемые этими хостами. Роль анализатора пакетов может исполнять любой хост с Ethernet-картой, так как в *неупорядоченном режиме* сетевой Ethernet-адаптер будет получать все проходящие по сети кадры. Эти кадры могут передаваться прикладной программе, извлекающей данные прикладного уровня. Например, в изображенном на рис. 7.25 сценарии хост C перехватывает запрос пароля, посылаемый хостом A хосту B, а также сам пароль, посылаемый хостом B хосту A. Получив пароли, злоумышленники могут впоследствии использовать их для проведения атаки отказа в обслуживании, которая будет обсуждаться далее. Анализ пакетов представляет собой «палку о двух концах» — он может быть очень полезен сетевому администратору для мониторинга сети и управления сетью (см. главу 8) и не менее полезен хакеру для его «черных» дел. Программное обеспечение анализаторов пакетов можно бесплатно загрузить с различных web-сайтов или купить. Известны случаи, когда (о, ужас!) преподаватели на лабораторных занятиях давали студентам задание написать программу, анализирующую пакеты и реконструирующую данные прикладного уровня. Программа Carnivore [142], выполняющая анализ пакетов, применяется ФБР для юридически санкционированного перехвата сетевого трафика.

Для обнаружения анализаторов пакетов достаточно обнаружить сетевые интерфейсы, работающие в неупорядоченном режиме. На предприятии сетевые администраторы могут установить на всех компьютерах предприятия специальное

программное обеспечение, предупреждающее о переходе сетевого интерфейса в неупорядоченный режим. Для дистанционного обнаружения сетевых интерфейсов, работающих в неупорядоченном режиме, могут применяться различные методы. Например, хост, отвечающий на эхо-запрос протокола ICMP (то есть дейтаграмму, вызывающую эхо-ответ — см. подраздел «Протокол DHCP» в разделе «Интернет-протокол» главы 4) с корректным IP-адресом, но некорректным MAC-адресом (адрес канального уровня), скорее всего, работает с интерфейсом в неупорядоченном режиме. Тем не менее можно обеспечить безопасную работу сети и в условиях несанкционированного анализа пакетов. Для этого нужно просто шифровать все данные, передаваемые по сети (особенно пароли).

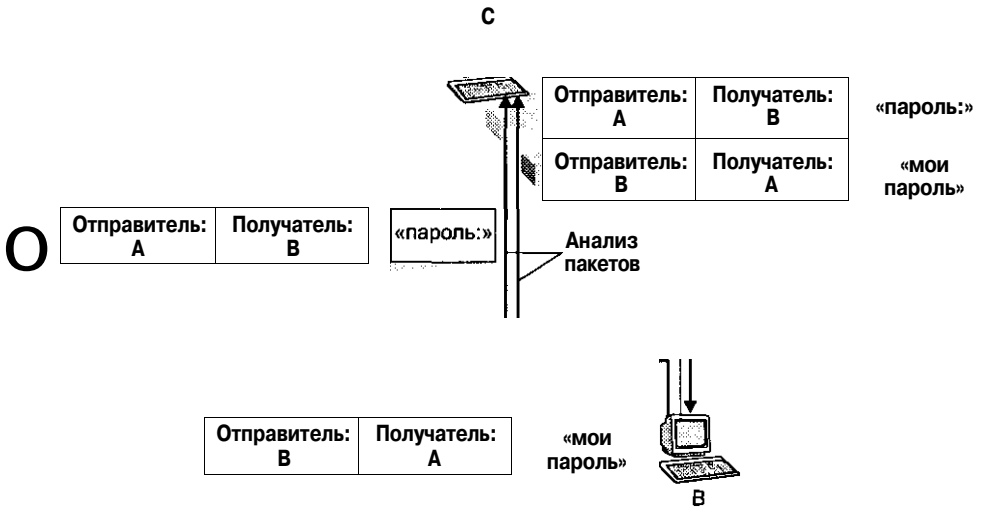


Рис. 7.25. Анализ пакетов

Подделка IP-адресов

Любое соединенное с Интернетом устройство обязательно посылает в сеть IP-дейтаграммы. В главе 4 отмечалось, что в этих дейтаграммах содержится IP-адрес отправителя, а также данные более высоких уровней. Пользователь, обладающий полным контролем над программным обеспечением этого устройства (в частности, над операционной системой), может модифицировать протоколы устройства так, чтобы в поле адреса отправителя дейтаграммы оказался нужный IP-адрес. Такой метод называется *подделкой IP-адресов,* или *IP-спуфингом.* Таким образом, пользователь может послать IP-дейтаграмму с произвольным содержимым от имени любого другого пользователя. Подделка IP-адресов часто применяется в атаках отказа в обслуживании (подробнее о них рассказано в следующем подразделе), чтобы скрыть истинный источник атаки. Если дейтаграмма содержит фальшивый IP-адрес отправителя, найти хост, с которого она была отправлена в сеть, довольно трудно.

С технической точки зрения предотвратить подделку IP-адресов нетрудно. Маршрутизаторы, осуществляющие *входную фильтрацию,* проверяют IP-адреса вхо-

дящих дейтаграмм и определяют, попадает ли данный IP-адрес в то множество адресов, которое доступно через данный сетевой интерфейс. Такая проверка может осуществляться на границе сети, например на корпоративном шлюзе или брандмауэре, где диапазон адресов хостов корпоративной сети очень легко определяется. На сегодняшний день входная фильтрация рассматривается как лучший метод борьбы с подделкой IP-адресов (RFC 2827). Хотя технически входную фильтрацию выполнить довольно просто, заставить это сделать организацию, владеющую маршрутизатором, против ее воли невозможно. В результате входная фильтрация применяется далеко не везде.

Атаки отказа в обслуживании и распределенного отказа в обслуживании

Атаки *отказа в обслуживании* (Denial of Service, DoS) составляют целый класс угроз безопасности. Как можно догадаться по названию, атаки отказа в обслуживании переводят сеть, хост или другой фрагмент сетевой инфраструктуры в состояние, в котором он не может использоваться легитимными пользователями. Как правило, атака отказа в обслуживании настолько повышает нагрузку на атакуемую инфраструктуру, что инфраструктура просто не успевает выполнять задачи, для которых она предназначена. При так называемой *синхронной атаке* [65] злоумышленник бомбардирует сервер шквалом SYN-пакетов протокола TCP, в каждом из которых содержится фальшивый IP-адрес. Сервер, не способный отличить настоящий SYN-пакет от поддельного, выполняет второй шаг TCP-рукопожатия (см. раздел «Протокол TCP — передача с установлением соединения» в главе 3), выделяя память для структуры данных запрашиваемого TCP-соединения. Третий этап тройного рукопожатия злоумышленником не выполняется, в результате количество «полуоткрытых» TCP-соединений на сервере постоянно растет. В конце концов, необходимость обрабатывать все поступающие SYN-пакеты и нехватка свободной памяти ставят сервер «на колени». Сходная разновидность атаки заключается в том, что на хост посылаются фрагменты IP-дейтаграмм, которых недостаточно, чтобы составить из них полную дейтаграмму. Атакуемый хост продолжает накапливать фрагменты, безуспешно ожидая оставшиеся, чтобы составить из них полную дейтаграмму, пока не израсходует всю память. При так называемой *smurf-атаке* [62] большое количество хостов провоцируется на эхо-ответы на ICMP-запросы (см. подраздел «Протокол ICMP» в разделе «Интернет-протокол» главы 4), содержащие поддельный IP-адрес. В результате хост, чей адрес был сфальсифицирован, получает огромное количество пакетов с эхо-ответами.

В случае атаки *распределенного отказа в обслуживании* (Distributed Denial of Service, DDoS) злоумышленник сначала получает доступ к множеству хостов Интернета (например, путем анализа пакетов), затем, как показано на рис. 7.26, устанавливает и запускает на каждом узурпированном им хосте подчиненную программу, ожидающую команды от управляющей программы. Когда количество взломанных хостов достигает определенного уровня, управляющая программа связывается с подчиненными программами, давая им команду начать атаку на выбранный хост. В результате атакуемый хост попадает под шквал пакетов, обработать которые он не в состоянии.

Рис. 7.26. DDoS-атака

В феврале 2000 года атаки отказа в обслуживании попали в заголовки газет, когда были атакованы такие web-сайты, как eBay, Yahoo и CNN [92]. Защититься от DoS-атак трудно, а от DDoS-атак еще труднее. Фильтровать пакеты в этом случае сложно, так как трудно отличить «хорошие» дейтаграммы от «плохих». Например, как может фильтрующий пакеты брандмауэр отличить SYN-пакет, посланный для установки легального TCP-соединения (например, от пользователя, желающего что-либо купить на web-сайте компании) от SYN-пакета, посланного злоумышленником для связывания ресурсов сервера. Использование фальшивых IP-адресов затрудняет обнаружение реального источника атаки. В ряде недавних исследований [7, 444] рассматривались методы маркировки IP-заголовков при прохождении их через маршрутизаторы, что позволило бы отследить источник DoS-атаки. Как только узурпированный хост идентифицирован, его можно временно изолировать, хотя, как правило, это медленный процесс, требующий участия человека. Отражение DDoS-атак представляет собой еще более сложный и долгий процесс.

Кража соединения

Предположим, Алиса и Боб установили соединение, а злоумышленник перехватывает все пакеты, которыми они обмениваются. В этом случае злоумышленник может воспользоваться данной ситуацией для обмана Алисы и Боба. Например, злоумышленник может сначала предпринять атаку отказа в обслуживании на Алису, чтобы вывести ее из игры, а затем от ее имени продолжить диалог с Бобом. Отслеживая соединение Алисы и Боба, злоумышленник знает полное состояние этого соединения (например, порядковый номер кадра, номер подтверждения, объявленное окно получателя). Таким образом, злоумышленник может подделать IP-дейтаграммы Алисы и послать их Бобу. Можете себе представить, какой хаос может наступить в отношениях Боба и Алисы!

Различные сетевые атаки и угрозы безопасности обсуждаются в [118, 442]. Отчеты о зарегистрированных атаках можно найти на web-странице координационного центра группы компьютерной «скорой помощи» CERT (http://www.cert.org/advisories). Этой теме посвящены и другие источники информации [13, 87, 539].

Безопасность на разных уровнях

В предыдущих разделах мы обсудили фундаментальные вопросы сетевой безопасности, включая шифрование с симметричными и открытым ключами, аутентификацию, передачу ключа, целостность данных и цифровую подпись. Теперь мы поговорим о том, как эти механизмы используются в целях обеспечения безопасности в Интернете. Интересно отметить, что безопасность можно обеспечить на любом из четырех верхних уровней стека Интернет-протоколов [337]. Когда безопасность требуется для определенного протокола прикладного уровня, тогда использующее этот протокол приложение может прибегнуть к помощи одной или нескольких служб безопасности, таких как службы конфиденциальности, аутентификации или целостности данных. Когда безопасность требуется для протокола транспортного уровня, тогда все приложения, использующие этот протокол, могут задействовать службы безопасности этого транспортного протокола. Когда безопасность требуется на сетевом уровне, тогда все сегменты транспортного уровня (а следовательно, и данные прикладного уровня) могут использовать службы безопасности сетевого уровня. Когда безопасность требуется на канальном уровне (в конкретном канале), тогда данные во всех кадрах, пересылаемых по каналу, обслуживаются службами безопасности этого канала.

В данном разделе мы поговорим о том, как безопасность обеспечивается на прикладном, транспортном, сетевом и канальном уровнях. Начнем мы, в соответствии с общей структурой книги, с вершины стека протоколов, то есть с прикладного уровня. Вопрос обеспечения безопасности на прикладном уровне мы рассмотрим на примере электронной почты, а затем постепенно переместимся вниз по стеку протоколов, исследуем протокол SSL (Secure Sockets Layer — слой защищенных сокетов) на транспортном уровне, протокол IPsec на сетевом уровне и, наконец, рассмотрим вопросы безопасности протокола беспроводных локальных сетей IEEE802.il.

Возможно, вы удивитесь, почему безопасность в Интернете обеспечивается сразу на нескольких уровнях. Разве недостаточно обеспечить безопасность на сетевом уровне? На этот вопрос есть два ответа. Во-первых, хотя безопасность на сетевом уровне может обеспечить «общую защиту» за счет шифрования всех данных в дейтаграммах (то есть всех сегментов транспортного уровня) и аутентификации всех IP-адресов отправителей, она не может обеспечить безопасность на пользовательском уровне. Например, коммерческий сайт не может полагаться на безопасность на IP-уровне при аутентификации клиента, приобретающего товары на этом сайте. Таким образом, имеется необходимость в безопасной работе более высоких уровней, а также общая защита на более низких уровнях. Во-вторых, новые Интернет-службы, включая службы безопасности, как правило, легче реализовывать на более

высоких уровнях стека протоколов. Ожидая широкого применения методов обеспечения безопасности на сетевом уровне, что, вероятно, произойдет не скоро, многие производители программного обеспечения реализуют эти методы в своих приложениях. Классическим примером является криптографическая система PGP (Pretty Good Privacy — достаточно хорошая степень конфиденциальности), обеспечивающая безопасную переписку по электронной почте (см. далее). Система PGP, для которой требуются только две прикладные программы — клиент и сервер, стала первой системой безопасности, завоевавшей широкую популярность в Интернете.

ИСТОРИЧЕСКАЯ СПРАВКА

Создателем системы PGP является Филип Р. Циммерманн. За это он на три года стал мишенью криминального расследования. В 1991 году его обвинили в нарушении ограничений на экспорт криптографического программного обеспечения, так как система PGP распространялась бесплатно (она была выпущена как условно бесплатная и размещена в Интернете, где стала доступной иностранным гражданам). По федеральному закону США криптографические программы классифицируются как военное снаряжение, и экспорт их запрещен.

Несмотря на отсутствие средств, наемного персонала и компании-опекуна, а также на жесткое давление государства, система PGP стала самой популярной в мире программой для шифрования электронной почты. Забавно, но своими действиями против Циммерманна правительство США, возможно, лишь добавило системе PGP популярности.

Дело Циммерманна было прекращено в начале 1996 года (объявление об этом было отпраздновано активистами Интернета) и стало исторической вехой в борьбе невиновного человека за свои права. Поражение правительства США в деле Циммерманна было особенно впечатляющим, так как примерно в то же время в Конгрессе США проходила кампания за введение цензуры в Интернете, а ФБР пыталось продавить закон, дающий этой организации большую свободу в области электронного шпионажа.

После прекращения судебного дела Циммерманн основал корпорацию PGP Inc., которая в декабре 1997 года была приобретена Network Associates. Сегодня Циммерманн является независимым консультантом в области криптографии.

Безопасная электронная почта

В этом разделе мы попытаемся создать безопасную электронную почту, используя методы, описанные в предыдущем разделе. Мы будем развивать наш проект поэтапно, на каждом этапе добавляя новую службу безопасности. Чтобы разговор был предметным, в наших рассуждениях мы продолжим опираться на любовную интрижку между Алисой и Бобом, завязавшуюся в разделе «Понятие сетевой безопасности». Пусть Алиса хочет послать Бобу письмо по электронной почте, а ревнивая жена Боба, скажем, Труди (имя «Trudy» напоминает слово «intruder» — злоумышленник), жаждет вмешаться.

Прежде чем перейти к проектированию безопасной электронной почты для Алисы и Боба, мы должны сначала определить, какие функции безопасности необходимы

им в первую очередь. Прежде всего, им требуется *конфиденциальность.* Как уже отмечалось в разделе «Понятие сетевой безопасности», ни Алиса, ни Боб, не хотят, чтобы Труди прочитала письмо Алисы. Кроме того, Алисе и Бобу нужна *аутентификация отправителя.* В частности, получив от Алисы сообщение «Я не люблю тебя больше. Я не хочу тебя видеть. Когда-то твоя, Алиса.», Боб, разумеется, хотел бы удостовериться в том, что это сообщение пришло от Алисы, а не от Труди. Еще нашим любовникам требуется обеспечить *целостность сообщения,* то есть нужна уверенность в том, что отправленное Алисой сообщение не изменено по дороге. Наконец, электронная почта должна обеспечивать *аутентификацию получателя,* то есть Алиса хочет быть уверенной в том, что она действительно посылает письмо Бобу, а не кому-либо еще, кто выдает себя за Боба (например, Труди).

Итак, начнем с решения первоочередного требования к безопасной системе электронной почты, а именно обеспечения конфиденциальности. Наиболее простой способ обеспечения конфиденциальности состоит в шифровании Алисой сообщений при помощи алгоритма с симметричными ключами (например, DES или AES). Получив письмо, Боб должен сначала расшифровать его. Как отмечалось в разделе «Принципы криптографии», если длина симметричного ключа достаточно велика и ключ известен только Алисе и Бобу, то кому-либо еще (например, Труди) прочитать зашифрованное сообщение будет весьма затруднительно. Хотя такой метод довольно прост, у него есть существенный недостаток, о котором мы тоже говорили, — трудно передать ключ так, чтобы его копии были только у Алисы и Боба. Таким образом, мы, естественно, рассмотрим альтернативный подход, а именно алгоритм шифрования с открытым ключом (например, RSA). При шифровании с открытым ключом Боб публикует свой открытый ключ (например, на сервере открытых ключей или на своей личной web-странице), Алиса зашифровывает свое сообщение открытым ключом Боба и посылает зашифрованное сообщение по адресу электронной почты Боба. (К зашифрованному сообщению добавляются заголовки стандарта MIME, после чего оно посылается по обычному протоколу SMTP, обсуждавшемуся в разделе «Электронная почта» главы 2.) Получив сообщение, Боб расшифровывает его своим личным ключом. При условии уверенности Алисы в том, что используемый ею открытый ключ действительно принадлежит Бобу, а длина этого ключа достаточно велика, подобный метод превосходно обеспечивает требуемую конфиденциальность. Однако один из его недостатков заключается в том, что шифрование с открытым ключом относительно неэффективно, особенно для длинных сообщений. (Сегодня, благодаря вложениям, содержащим изображения, аудио и видео, длинные сообщения становятся весьма распространенным явлением в Интернете.)

Чтобы решить проблему неэффективности, будем использовать ключ сеанса (см. раздел «Передача ключей и сертификация»). В частности, Алиса, во-первых, случайным образом выбирает симметричный ключ сеанса K^s, во-вторых, зашифровывает этим ключом сеанса K^s свое сообщение m, в-третьих, зашифровывает ключ сеанса K^s открытым ключом Боба $K_в^+$, в-четвертых, формирует из всех зашифрованных данных один пакет и, в-пятых, посылает этот пакет по адресу электронной почты Боба. Эти действия иллюстрирует рис. 7.27. (На этом и на последующих рисунках плюс обозначает операцию конкатенации, а минус — противоположную операцию,

то есть разъединение.) Получив сообщение, Боб, во-первых, с помощью своего личного ключа K^{s}~ получает ключ сеанса K^{s}, во-вторых, с помощью ключа сеанса K^{s} расшифровывает сообщение m.

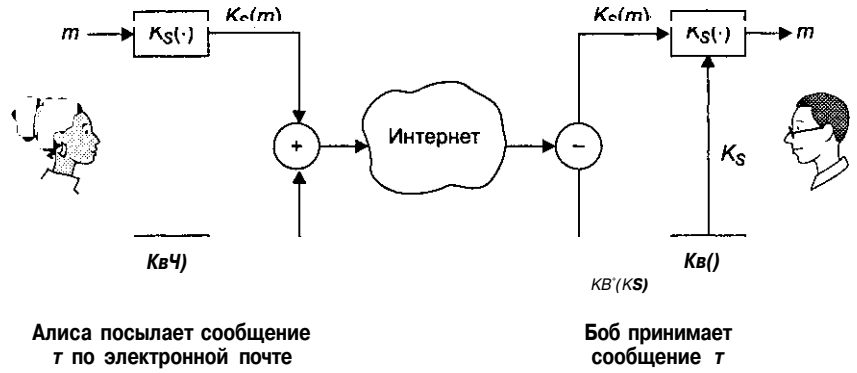


Рис. 7.27. Использование шифрования с ключом сеанса для обеспечения конфиденциальности

Итак, мы спроектировали безопасную систему электронной почты, обеспечивающую конфиденциальность. Спроектируем теперь другую систему, обеспечивающую аутентификацию отправителя и целостность сообщения. Предположим на время, что Алисе и Бобу более не требуется конфиденциальность (они готовы поделиться своими чувствами со всеми!), и их интересуют только аутентификация отправителя и целостность сообщения. Для решения данной задачи мы воспользуемся цифровыми подписями и дайджестами сообщений, о которых рассказывалось в разделе «Целостность данных». Итак, Алиса, во-первых, применяет к сообщению m хэш-функцию Я (например, MD5) и получает дайджест сообщения, во-вторых, подписывает результат хэш-функции своим личным ключом K^{A}~, в-третьих, объединяет исходное (незашифрованное) сообщение с подписью, формируя пакет, и, в-четвертых, отправляет этот пакет по адресу электронной почты Боба. Получив сообщение, Боб, во-первых, применяет открытый ключ $K/$ Алисы к подписи дайджеста сообщения и, во-вторых, сравнивает результат этой операции с хэшем сообщения, который он вычисляет сам. Эти действия иллюстрирует рис. 7.28. Как отмечалось в разделе «Передача ключей и сертификация», если результаты этих двух вычислений совпадают, Боб может быть уверен, что сообщение пришло от Алисы и оно не модифицировано.

Теперь попробуем объединить в одной системе службы обеспечения конфиденциальности, аутентификации отправителя и целостности сообщения. Сначала Алиса создает предварительный пакет (как на рис. 7.28), состоящий из исходного сообщения и подписанного цифровой подписью хэша этого сообщения. Затем весь предварительный пакет зашифровывается как единое сообщение (см. рис. 7.27), формируется новый пакет, который и отправляется Бобу. Вместе эти действия иллюстрирует рис. 7.29. Получив пакет, Боб сначала применяет к нему действия, показанные на рис. 7.27, а затем действия, показанные на рис. 7.28. Очевидно, что такая схема обеспечивает конфиденциальность, аутентификацию отправителя и

целостность сообщения. Обратите внимание, что Алиса дважды использует шифрование с открытым ключом: один раз со своим личным ключом и второй раз с открытым ключом Боба. Соответственно, Боб также использует шифрование с открытым ключом дважды: один раз со своим личным ключом и второй раз с открытым ключом Алисы.

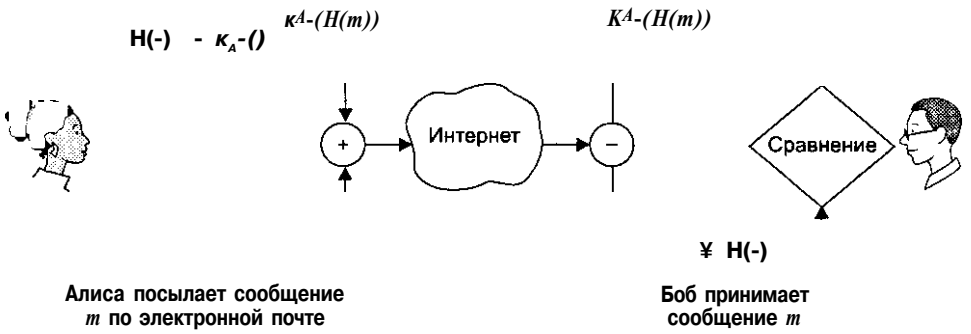


Рис. 7.28. Использование хэш-функции и цифровой подписи для аутентификации и обеспечения целостности сообщения

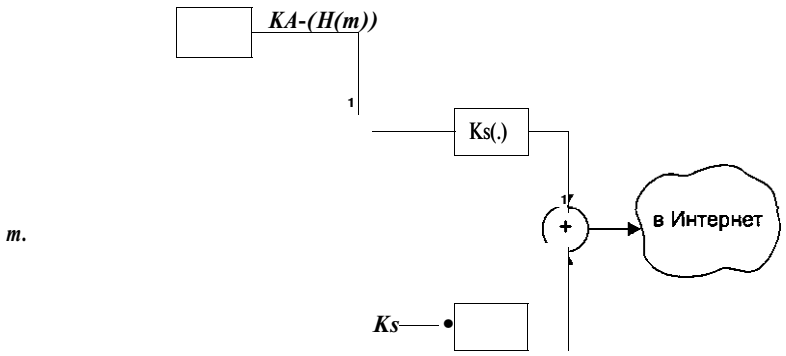


Рис. 7.29. Схема, обеспечивающая конфиденциальность, аутентификацию и целостность сообщения

Показанная на рис. 7.29 схема безопасной электронной почты, вероятно, обеспечит достаточный уровень безопасности для большинства пользователей. Но нам необходимо решить еще один важный вопрос. В данной схеме Алиса должна получить открытый ключ Боба, а Боб — открытый ключ Алисы. Как отмечалось в разделе «Передача ключей и сертификация», открытые ключи обычно распространяют центры сертификации ключей.

Созданная Филом Циммерманном в 1991 году система *PGP(*Pretty Good Privacy — достаточно хорошая степень конфиденциальности) представляет собой схему шифрования электронной почты, ставшую фактическим стандартом. На web-сайт, посвященный PGP (http://www.pgpi.org), ежемесячно поступает более миллиона запросов от пользователей из 166 стран мира. Здесь можно бесплатно получить

различные версии PGP для разных платформ, а также прочитать много интересного. Особенно интересно эссе самого автора [569]. Система PGP также распространяется и на коммерческой основе в виде подключаемых модулей для различных почтовых агентов, включая Exchange и Outlook корпорации Microsoft, а также Eudora компании Qualcomm.

Устройство системы PGP, по существу, не отличается от схемы, изображенной на рис. 7.29. В зависимости от версии PGP для вычисления цифрового дайджеста используется алгоритм MD5 или SHA, для шифрования с симметричными ключами — метод 3DES или IDEA и для шифрования с открытым ключом — алгоритм RSA. Кроме того, в системе PGP применяется сжатие данных.

При запуске система PGP создает для пользователя пару ключей — открытый и личный. Затем открытый ключ может быть опубликован на web-сайте пользователя или на сервере открытых ключей. Личный ключ защищен паролем. Пользователь должен вводить пароль при каждом применении личного ключа. Система PGP позволяет пользователю подписать сообщение цифровой подписью, а также зашифровать сообщение. Сообщение, подписанное при помощи системы PGP, изображено на рис. 7.30. При пересылке по электронной почте это сообщение предваряется заголовком формата MIME. Зашифрованные данные сообщения представляют собой подписанный цифровой подписью дайджест сообщения, то есть $K^{A}(H(m))$. Как уже отмечалось, для проверки целостности полученного сообщения Бобу нужно получить открытый ключ Алисы.

```
———BEGIN PGP SIGNED MESSAGE———
Hash: SHAl

Bob:

Can I see you tonight?

Passionately yours, Alice
———BEGIN PGP SIGNATURE———
Version: PGP for Personal Privacy 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
———END PGP SIGNATURE———
```

Рис. 7.30. Подписанное PGP-сообщение

На рис. 7.31 показано секретное сообщение, зашифрованное при помощи системы PGP. Это сообщение также предваряется заголовком формата MIME. Разумеется, открытый текст не включается в секретное сообщение. Когда отправитель (например, Алиса) хочет добиться одновременно конфиденциальности и гарантии целостности, система PGP помещает зашифрованное сообщение (см. рис. 7.31) внутрь подписанного сообщения (см. рис. 7.30).

Система PGP также предоставляет механизм сертификации открытых ключей, но он существенно отличается от традиционного механизма, предполагающего

использование сертификационного центра. Открытые PGP-ключи сертифицируются так называемой «паутиной доверия». Алиса сама может сертифицировать любую пару (ключ, имя пользователя), если она уверена, что этот ключ действительно соответствует этому пользователю. Кроме того, система PGP позволяет Алисе заявить, что она доверяет ручательству другого пользователя. Некоторые пользователи системы PGP образуют особые группы, члены которых подписывают ключи друг друга. Для этого пользователи собираются вместе, обмениваются дискетами со своими открытыми ключами и подписывают эти ключи своими личными ключами. Кроме того, открытые ключи PGP распределяются по Интернету серверами открытых ключей PGP. Когда пользователь предоставляет свой открытый ключ такому серверу, сервер сохраняет копию ключа, посылает ключ на другие серверы открытых ключей и предоставляет этот ключ любому, кто его запросит. Однако самыми популярными методами распространения ключей остаются публикация на личных web-страницах пользователей и пересылка по электронной почте.

```
———BEGIN PGP MESSAGE———
Version: PGP for Personal Privacy 5.0
u2R4d+/jKran8Bc5+hgDsqAewsDfrGdszX681iKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLoO+l/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
———END PGP MESSAGE
```

Рис. 7.31. Секретное PGP-сообщение

Протоколы SSL и TLS

В предыдущем разделе мы говорили о том, как могут использоваться на прикладном уровне обсуждавшиеся в начале этой главы механизмы обеспечения безопасности (на примере электронной почты), такие как шифрование, аутентификация, распределение ключей, целостность данных и цифровые подписи. В этом разделе мы продолжим изучение механизмов обеспечения безопасности на транспортном уровне на примере Интернет-коммерции, поскольку финансовые и коммерческие транзакции являются важной движущей силой развития системы безопасности в Интернете.

Рассмотрим типичный сценарий Интернет-коммерции. Боб, «усевшись» в браузер, путешествует по Интернету и попадает на web-сайт компании Алиса Incorporated, продающей некие товары длительного пользования. На web-сайте Боб обнаруживает форму, в которой нужно указать требуемое ему количество товара, свой адрес и номер кредитной карты для оплаты товара. Боб вводит нужную информацию, щелкает мышью на кнопке submit (утвердить), а затем ожидает получения (например, по обычной почте) купленных им товаров. Кроме того, он рассчитывает увидеть результат покупки в следующем отчете о состоянии свой кредитной карты. Все это звучит прекрасно, но, если не предпринять специальных мер, таких как шифрование и аутентификация, Боба могут ожидать несколько сюрпризов.

- Злоумышленник может перехватить заказ и, таким образом, получить информацию о кредитной карте Боба. Впоследствии злоумышленник сможет совершать покупки за счет Боба.
- Web-сайт может отображать знаменитый логотип компании Алиса Incorporated, но в действительности являться сайтом злоумышленника, замаскированным под web-сайт компании Алиса Incorporated. Злоумышленник может взять деньги Боба и сбежать. Или злоумышленник может совершить собственные покупки, расплатившись карточкой Боба.

Возможны и другие сюрпризы (о некоторых из них мы поговорим в следующем подразделе), но перечисленные являются наиболее важными. В Интернет-коммерции применяется протокол SSL, решающий обе проблемы.

Протокол *SSL* (Secure Sockets Layer — слой защищенных сокетов), разработанный компанией Netscape, предназначен для шифрования данных и аутентификации между web-клиентом и web-сервером. Протокол начинает свою работу с фазы рукопожатия, при которой две стороны соединения договариваются об используемом алгоритме шифрования (например, DES или IDEA) и ключах. В этой же фазе производится аутентификация сервера и при необходимости клиента для сервера. После завершения фазы рукопожатия начинается передача прикладных данных. При этом все данные зашифровываются при помощи ключа сеанса, о котором стороны договариваются на стадии рукопожатия. Протокол SSL широко применяется в Интернет-коммерции. Он реализован почти во всех популярных web-браузерах и web-серверах. Кроме того, он составляет основу протокола *TLS* (Transport Layer Security — безопасность на транспортном уровне), спецификация которого содержится в RFC 2246.

Сфера применения протоколов SSL и TLS не ограничивается Всемирной паутиной. Например, они также могут использоваться для аутентификации и шифрования данных в протоколе ШАР (Internet Mail Access Protocol — протокол доступа к почте Интернета). Протокол SSL может рассматриваться как некий слой между прикладным и транспортным уровнями. На передающей стороне протокол SSL получает данные (например, HTTP- или IMAP-сообщения от приложения), зашифровывает их и направляет в TCP-сокет. На принимающей стороне протокол SSL считывает данные из TCP-сокета, расшифровывает их и направляет приложению. Хотя протокол SSL может применяться во многих Интернет-приложениях, мы рассмотрим его в контексте web, где сегодня он используется в основном для Интернет-коммерции.

Протокол SSL предоставляет следующие функции.

- *Аутентификация сервера* позволяет пользователю убедиться в подлинности сервера. На поддерживающем протокол SSL web-браузере хранятся список доверенных сертификационных центров и их открытые ключи. Когда браузер желает связаться с web-сервером, браузер получает у сервера сертификат, содержащий открытый ключ сервера. Сертификат подписывается цифровой подписью того сертификационного центра, который есть в списке клиента. Таким образом, браузер может убедиться в подлинности сервера, прежде чем пользователь сообщит серверу номер своей кредитной карты. В контексте предыду-

щего примера аутентификация сервера позволяет Бобу убедиться, что он действительно посылает номер своей кредитной карты серверу компании Алиса Incorporated, а не кому-то другому, маскирующемуся под компанию Алиса Incorporated.

- *Аутентификация клиента* позволяет серверу убедиться в подлинности клиента. Для этого также используются сертификаты клиентов, издаваемые сертификационным центром. Эта аутентификация важна, если сервер, например, является банком, посылающим своему клиенту конфиденциальную информацию. Аутентификация клиента поддерживается протоколом SSL, но ее применение не является обязательным. В дальнейшем мы не будем рассматривать этот вопрос.
- *Шифрование SSL-сеанса.* При шифровании сеанса вся информация, пересылаемая между браузером и сервером, зашифровывается передающей программой (браузером или web-сервером) и расшифровывается принимающей программой (браузером или web-сервером). Эта конфиденциальность может быть важной как для клиента, так и для владельца web-сайта. Кроме того, протокол SSL предоставляет механизм обнаружения вмешательств злоумышленников.

Функционирование протокола SSL

Пользователь, например Боб, путешествует в браузере по Всемирной паутине и щелкает мышью на ссылке, приводящей его на безопасную страницу поддерживающего протокол SSL сервера Алисы. Протокольная часть URL-адреса этой web-страницы представляет собой слово «https» вместо обычного «http». Затем браузер и сервер запускают процедуру рукопожатия протокола SSL, в ходе которой, во-первых, осуществляется аутентификация сервера, во-вторых, генерируется пара симметричных ключей. На обоих этапах используется технология открытого ключа RSA. Основной поток событий процедуры рукопожатия показан на рис. 7.32. Во время этой процедуры Алиса посылает Бобу свой сертификат, из которого Боб получает открытый ключ Алисы. Затем Боб создает случайный симметричный ключ, зашифровывает его открытым ключом Алисы и посылает Алисе. Теперь у Боба и Алисы есть по симметричному ключу сеанса. По завершении процедуры рукопожатия все данные, пересылаемые между браузером и сервером (по TCP-соединениям), зашифровываются при помощи симметричного ключа сеанса каждого из участников.

Итак, мы получили общее представление о протоколе SSL. Рассмотрим теперь некоторые наиболее важные детали и в частности процедуру рукопожатия.

1. Браузер посылает серверу номер версии используемого браузером протокола SSL и криптографические предпочтения, чтобы договориться об алгоритме шифрования с симметричными ключами.
2. Сервер посылает браузеру свой номер версии протокола SSL, криптографические предпочтения и свой сертификат. Сертификат включает открытый RSA-ключ сервера и подпись некоего сертификационного центра.
3. У браузера есть список доверенных сертификационных центров, а также открытый ключ каждого сертификационного центра. Когда браузер получает от

сервера сертификат, он смотрит, есть ли такой сертификационный центр в его списке. Если такого сертификационного центра в списке нет, пользователю сообщается, что безопасное соединение не может быть установлено. В противном случае браузер с помощью открытого ключа сертификационного центра проверяет сертификат и получает открытый ключ сервера.

Браузер генерирует симметричный ключ сеанса, зашифровывает его открытым ключом сервера и посылает серверу.

Браузер посылает серверу сообщение, информируя его о том, что дальнейшие сообщения клиент будет шифровать ключом сеанса. Затем он посылает отдельное (зашифрованное) сообщение, в котором указывает, что браузер завершил свою часть рукопожатия.

Сервер посылает браузеру сообщение, информирующее его о том, что дальнейшие сообщения сервер будет шифровать ключом сеанса. Затем он посылает отдельное (зашифрованное) сообщение, в котором указывает, что серверная часть рукопожатия завершена.

После завершения SSL-рукопожатия начинается SSL-сеанс. Браузер и сервер используют ключи сеанса для шифрования, дешифрирования посылаемых друг другу данных. Кроме того, симметричные ключи сеанса гарантируют целостность данных.

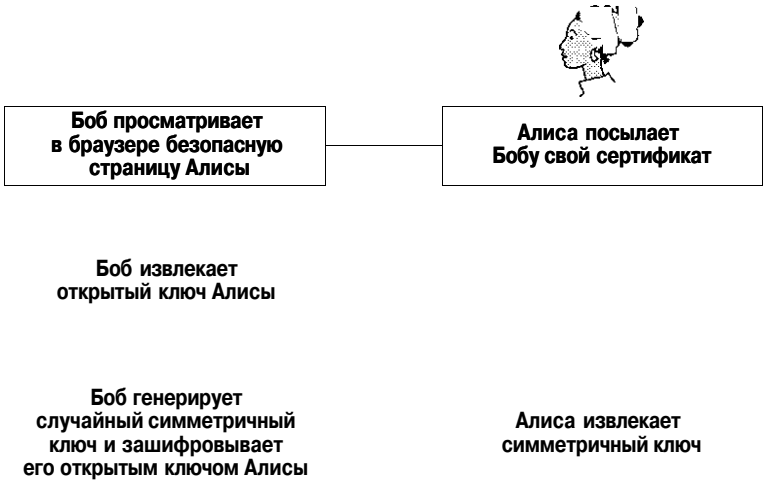


Рис. 7.32. Общая схема фазы рукопожатия протокола SSL

Процедура рукопожатия протокола SSL подразумевает выполнение гораздо большего числа шагов, чем перечислено выше. Дополнительную информацию о протоколе SSL можно найти на сайте Security Developer Central корпорации Nescape (http://developer.netscape.com/tech/security/). Отметим также, что протокол SSL может применяться не только для оплаты покупок кредитными картами в Интернете, но и для других финансовых транзакций, включая банковские и биржевые операции.

Ограничения на применение протокола SSL в Интернет-коммерции

Благодаря своей простоте протокол SSL получил широкое распространение в браузерах, серверах и коммерческих Интернет-продуктах. Эти серверы и браузеры обеспечивают популярную платформу для использования кредитных карт в Интернет-коммерции. Тем не менее следует помнить, что протокол SSL разрабатывался не для Интернет-коммерции, а в целях обеспечения безопасной надежной связи между клиентом и сервером. Поэтому протоколу SSL присущи некоторые органические недостатки, из-за которых он не может считаться полноценным протоколом для Интернет-коммерции.

Поговорим еще раз о том, что происходит, когда Боб осуществляет покупку на web-сайте Алисы, используя протокол SSL. Подписанный сертификат, который Боб получает от Алисы, убеждает его в том, что он действительно имеет дело с компанией Алиса Incorporated и это реальная, а не фиктивная компания. Однако сертификат общего вида не сообщает, имеет ли компания Алиса Incorporated право на операции с кредитными картами, а также является ли Алиса Incorporated надежным коммерческим партнером. Все это создает дополнительные возможности для мошенничества. Аналогичная проблема проявляется и при аутентификации клиента. Даже если клиент аутентифицируется при помощи протокола SSL, сертификат клиента никак не связан с его кредитной картой. Таким образом, Алиса не получает никаких гарантий относительно кредитоспособности Боба. Это также создает дополнительные возможности для самых различных форм мошенничества, включая использование украденных кредитных карт и отказ от приобретенных товаров.

Безопасность на сетевом уровне

Безопасность на сетевом уровне обеспечивает протокол *IPsec* (IP security — безопасный протокол IP), по сути представляющий собой набор протоколов. Протокол IPsec довольно сложен, различные его аспекты описывают более десятка документов RFC. В этом разделе мы обсудим протокол IPsec в весьма специфическом контексте, а именно исходя из предположения, что его поддерживают *все* хосты Интернета. Хотя реальная ситуация уже много лет иная, данный контекст упростит наше обсуждение и поможет понять ключевые особенности протокола IPsec. Основные документы, описывающие протокол IPsec, — это RFC 2401, в котором описывается общая архитектура протокола IPsec, и RFC 2411, содержащий обзор составляющих IPsec протоколов и перечень документов, их описывающих.

Прежде чем перейти к особенностям протокола IPsec, поговорим о том, что означает обеспечивать безопасность на сетевом уровне. Начнем с вопроса конфиденциальности. Конфиденциальность на сетевом уровне можно обеспечить, если зашифровывать все данные, переносимые в IP-дейтаграммах. То есть каждый раз, прежде чем отправить дейтаграмму в сеть, хост должен зашифровывать ее поле данных. В принципе шифрование может осуществляться алгоритмом с симметричными ключами, алгоритмом с открытым ключом или с помощью ключа сеанса, о котором стороны договариваются по алгоритму с открытым ключом. Поле дан-

ных может быть TCP-сегментом, UDP-сегментом, ICMP-сообщением и т. д. Если бы подобная сетевая служба была реализована, то все данные, пересылаемые между хостами, включая электронную почту, web-страницы, управляющие сообщения (такие как ICMP и SNMP), были бы скрыты от всех посторонних, «прослушивающих» сеть. Таким образом, подобная служба обеспечила бы определенный уровень общей защиты Интернет-трафика, существенно повысив безопасность сетей.

Помимо конфиденциальности было бы неплохо, если бы сеть обеспечивала *аутентификацию источника.* Когда хост-получатель принимает IP-дейтаграмму с определенным IP-адресом отправителя, он должен быть уверен, что указанный адрес является истинным. Подобная служба могла бы поставить заслон многим разновидностям сетевых атак.

В набор протоколов IPsec входят два основных протокола: протокол *AH* (Authentication Header — заголовок аутентификации) и протокол *ESP* (Encapsulation Security Payload — безопасная инкапсуляция полезной нагрузки). Когда хост-отправитель посылает дейтаграмму хосту-получателю, он использует либо протокол AH, либо протокол ESP. Протокол AH обеспечивает аутентификацию источника и целостность данных, но не обеспечивает конфиденциальности. Протокол ESP обеспечивает аутентификацию источника, целостность данных и конфиденциальность. Этот протокол предоставляет больше возможностей и, разумеется, является более сложным и требует больших усилий по обработке, чем протокол AH. Ниже мы обсудим оба протокола.

В обоих протоколах, прежде чем отправить безопасные дейтаграммы от хоста-отправителя хосту-получателю, отправитель обменивается рукопожатиями с сетевыми хостами и создает логическое соединение сетевого уровня. Этот логический канал называется *безопасным соединением* (Security Association, SA). Таким образом, протокол IPsec превращает сетевой уровень Интернета, традиционно функционирующий без установления соединений, в уровень с логическими соединениями! Логическое SA-соединение является симплексным, то есть однонаправленным. Если оба хоста хотят пересылать друг другу безопасные дейтаграммы, то необходимо установить два SA-соединения по одному в каждом направлении. SA-соединение уникальным образом определяется тремя параметрами:

- идентификатором протокола безопасности (AH или ESP);
- IP-адресом источника симплексного соединения;
- 32-разрядным идентификатором соединения, называемым индексом параметра безопасности (Security Parameter Index, SPI).

Для данного SA-соединения (то есть логического соединения между хостом-источником и хостом-получателем) в каждой IP-дейтаграмме содержится специальное поле для SPI, одинаковое для всех дейтаграмм.

Протокол AH

Как уже отмечалось, протокол AH обеспечивает аутентификацию хоста-источника и целостность данных, но не конфиденциальность. Когда один хост-источник хочет отправить одну или несколько дейтаграмм определенному получателю, он

сначала устанавливает с получателем безопасное соединение (SA-соединение). Установив SA-соединение, источник может посылать хосту-получателю безопасные дейтаграммы. Каждая безопасная дейтаграмма содержит AH-заголовок, включаемый между исходными данными IP-дейтаграммы (например, TCP- или UDP-сегментом) и IP-заголовком, как показано на рис. 7.33. Таким образом, поле AH расширяет оригинальное поле данных, после чего расширенное поле данных помещается в стандартную IP-дейтаграмму. При этом в поле протокола IP-заголовка помещается значение 51. Получив эту IP-дейтаграмму, хост-получатель обнаруживает значение 51 в поле протокола IP-заголовка и поэтому обрабатывает дейтаграмму по протоколу AH. (Как было показано ранее, поле протокола в IP-дейтаграмме используется для обозначения протокола более высокого уровня, например UDP, TCP или ICMP, которому следует передать поле данных IP-дейтаграммы.) Промежуточные маршрутизаторы обрабатывают дейтаграммы так же, как и всегда — они переправляют их дальше в соответствии с IP-адресами получателей.

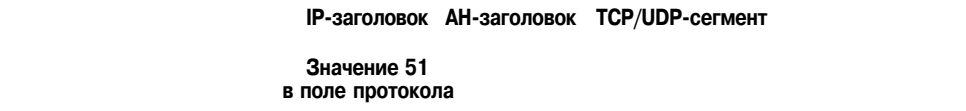


Рис. 7.33. Положение AH-заголовка в IP-дейтаграмме

AH-заголовок содержит несколько полей.

- *Следующий заголовок.* Это поле играет роль поля протокола в обычной дейтаграмме. Оно указывает, является ли следующее за AH-заголовком поле данных TCP-сегментом, UDP-сегментом, ICMP-сегментом и т. д. (Как уже отмечалось, поле протокола IP-дейтаграммы теперь обозначает протокол AH, поэтому для обозначения протокола транспортного уровня требуется дополнительное поле.)
- *Индекс параметра безопасности* (SPI). Это поле содержит произвольное значение, которое в комбинации с IP-адресом получателя уникальным образом идентифицирует SA-соединение для данной дейтаграммы.
- *Порядковый номер.* 32-разрядное поле, содержащее порядковый номер каждой дейтаграммы. Изначально (во время установки SA-соединения) это поле устанавливается равным 0. Поле порядкового номера используется протоколом AH для отражения атак повторного воспроизведения и атак с человеком посредине (см. раздел «Аутентификация»).
- *Данные аутентификации.* Это поле переменной длины содержит подписанный дайджест сообщения (то есть цифровую подпись) для данной дейтаграммы. Дайджест сообщения вычисляется по оригинальной IP-дейтаграмме. Таким образом обеспечивается аутентификация хоста-отправителя и целостность IP-дейтаграммы. Цифровая подпись вычисляется при помощи алгоритма, определенного при установке SA-соединения (например, MD5 или SHA).

Получив IP-дейтаграмму с AH-заголовком, хост-получатель определяет для данной дейтаграммы SA-соединение, а затем, обработав поле данных аутентифика-

ции, убеждается в целостности дейтаграммы. В процедуре аутентификации протокола IPsec (как для протокола AH, так и для протокола ESP) используется схема вычисления зашифрованного дайджеста сообщения, называемая HMAC (RFC 2104). В схеме HMAC для аутентификации сообщений применяется общий секретный ключ, а не шифрование с открытым ключом. Дополнительные сведения о протоколе AH можно найти в RFC 2402.

Протокол ESP

Протокол ESP обеспечивает конфиденциальность на сетевом уровне, а также аутентификацию хоста-отправителя и целостность данных. Работа протокола также начинается с установки SA-соединения с хостом-получателем. Затем хост-отправитель может посылать хосту-получателю безопасные дейтаграммы. Как видно на рис. 7.34, безопасная дейтаграмма создается из оригинальной IP-дейтаграммы добавлением к ее полю данных заголовка и концевика. Для поля протокола в IP-заголовке используется значение 50. Получив IP-дейтаграмму, хост-получатель замечает, что в поле протокола IP-заголовка содержится значение 50, поэтому он обрабатывает дейтаграмму с помощью протокола ESP. Как показано на рис. 7.34, оригинальные данные IP-дейтаграммы зашифровываются вместе с полем ESP-концевика. Конфиденциальность обеспечивается при помощи алгоритма шифрования DES-CBC (RFC 2405). ESP-заголовок состоит из 32-разрядного поля для индекса SPI и 32-разрядного поля для порядкового номера, роль которых аналогична соответствующим полям протокола AH. Концевик включает поле следующего заголовка, решающее ту же задачу, что и соответствующее поле протокола AH. Обратите внимание, поскольку поле следующего заголовка зашифровывается вместе с исходными данными, злоумышленник не сможет определить, какой транспортный протокол используется. Следом за концевиком располагается поле данных аутентификации, выполняющее ту же функцию, что и аналогичное поле протокола AH. Дополнительную информацию о протоколе ESP можно найти в RFC 2406.

Рис. 7.34. Поля протокола ESP в IP-дейтаграмме

SA-соединение и управление ключом

Для успешного развертывания протокола IPsec необходимы масштабируемые и автоматизированные схемы управления ключом и установления SA-соединения. Для этого было определено несколько протоколов.

- Протокол *IKE* (Internet Key Exchange — обмен ключей по Интернету), описанный в RFC 2409, является протоколом управления ключами для IPsec по умолчанию.

- Протокол *ISKMP* (Internet Security Association and Key Management — безопасное Интернет-соединение и управление ключами) определяет процедуры установления и разрыва SA-соединений. Протокол ISKMP определен в RFC 2407 и RFC 2408. Процедуры установления и разрыва SA-соединений протокола ISKMP полностью отделены от процедур обмена ключами (IKE).

На этом мы завершаем наше обсуждение набора протоколов IPsec. Мы рассмотрели набор протоколов IPsec в контексте протокола IPv4 и «транспортного режима». В наборе протоколов IPsec также определен «туннельный режим», при котором функции безопасности предоставляются маршрутизаторами, а не хостами. Кроме того, в наборе протоколов IPsec предусмотрены процедуры шифрования как для протокола IPv4, так и для протокола IPv6.

Безопасность в беспроводных локальных сетях стандарта IEEE 802.11

В беспроводных сетях, где переносящие кадры радиоволны могут распространяться намного дальше, чем это необходимо, вопрос безопасности стоит особенно остро. В [467] описан интересный эксперимент. Автор в течение полутора лет ездил по Сан-Франциско с лэптопом и сетевой картой 802.11 и искал беспроводные сети, «видимые» за пределами офисных зданий. Он насчитал более 9000 таких сетей. На одном лишь перекрестке он обнаружил сразу шесть сетей! Но самым интересным оказался тот факт, что 85 % из этих 9000 сетей не используют протокол *WEP* (Wired Equivalent Privacy — эквивалент кабельной безопасности), предназначенный для обеспечения безопасности в сетях стандарта 802.11! В этом разделе мы рассмотрим протокол WEP и некоторые дыры в его системе безопасности. Дополнительные сведения об этом протоколе, а также ссылки на дополнительную литературу можно найти в [546, 549].

Определенный стандартом IEEE 802.11 протокол WEP обеспечивает как аутентификацию, так и шифрование между хостом и точкой доступа в сеть (то есть базовой станцией), для чего используется алгоритм шифрования с общими симметричными ключами. В спецификации протокола WEP не определяется алгоритм управления ключами, поэтому предполагается, что хост и точка беспроводного доступа каким-то образом об этом договариваются. Аутентификация реализуется так же, как и в придуманной нами модели протокола ap4.0 (см. подраздел «Протокол аутентификации ap 4.0» в разделе «Аутентификация»). Процесс аутентификации состоит из четырех этапов.

1. Беспроводной хост запрашивает аутентификацию у базовой станции (точки доступа).
2. Базовая станция отвечает на этот запрос 128-разрядным значением нонса.
3. Беспроводной хост зашифровывает полученный ноне при помощи симметричного ключа и посылает зашифрованный ноне базовой станции.
4. Базовая станция расшифровывает зашифрованный хостом ноне. Если полученное значение совпадает с исходным нонсом, базовая станция подтверждает личность хоста.

Алгоритм шифрования данных протокола WEP иллюстрирует рис. 7.35. Предполагается, что секретный 40-разрядный симметричный ключ K^s известен и хосту и базовой станции. К этому ключу добавляется 24-разрядный вектор инициализации (Initialization Vector, IV), в результате образуется 64-разрядный ключ шифрования кадра. Для каждого следующего кадра используется новый вектор инициализации, и таким образом каждый кадр зашифровывается новым 64-разрядным ключом. Шифрование выполняется следующим образом. Сначала для поля полезной нагрузки вычисляется четырехбайтовое значение CRC (см. раздел «Обнаружение и исправление ошибок» в главе 5). Затем полезная нагрузка и четыре байта CRC зашифровываются с помощью потокового шифра RC4. Мы не будем рассматривать здесь детали алгоритма RC4 (подробности можно узнать в [546]), нам достаточно знать, что, получив на входе значение ключа *(Ks, IV),* алгоритм RC4 формирует на выходе поток значений ключей k_1^{IV}, k_2^{IV}, k_3^{IV} и т. д., используемых для шифрования содержащихся в кадре данных и CRC. Будем считать, что эти операции выполняются побайтно. Шифрование реализуется путем сложения по модулю 2 г-го байта данных d_i с г-м ключом kj^v из потока ключей, генерированного парой *(Ks, IV),* в результате чего формируется г-й байт зашифрованного текста с,:

$$c_i = d_i \text{ XOR } k_i^{IV}.$$

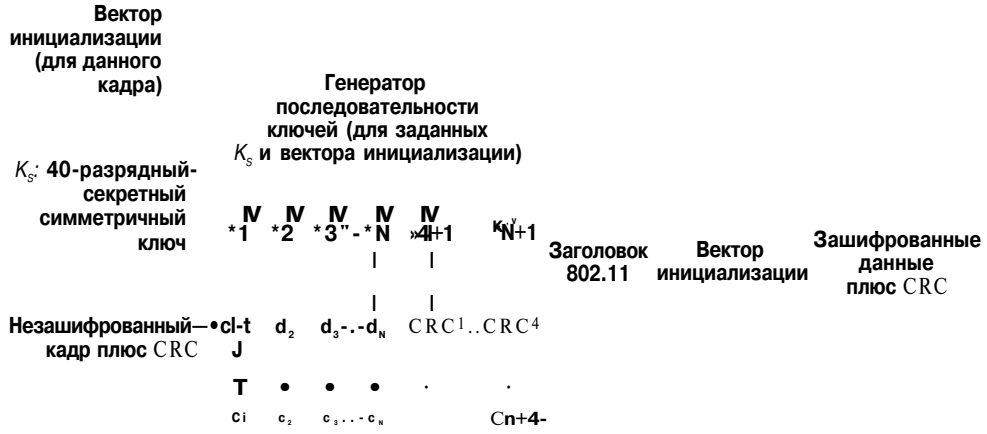


Рис. 7.35. Протокол WEP 802.11

Значение вектора инициализации изменяется от одного кадра к другому и включается в заголовок каждого зашифрованного кадра 802.11 открытым текстом, как показано на рисунке. Получатель берет секретный 40-разрядный симметричный ключ, также известный отправителю, добавляет к нему вектор инициализации и использует полученный 64-разрядный ключ (идентичный ключу отправителя) для расшифровки кадра:

$$d_i = c_i \text{ XOR } k_i^{Iv}.$$

Хотя сам алгоритм RC4 считается безопасным [439], для его правильного применения требуется, чтобы одно и то же 64-разрядное значение никогда не исполь-

зовалось дважды. Вспомним, что ключ WEP изменяется для каждого кадра. Для заданного ключа K^s (который изменяется крайне редко, если изменяется вообще) это означает, что уникальными будут только 2^{24} ключей. Если эти ключи выбираются редко, можно показать [546], что вероятность выбора того же значения вектора инициализации (а следовательно, и того же 64-разрядного ключа) превысит 99 % уже после 12 000 кадров. При размере кадра в 1 Кбайт и скорости передачи данных в 11 Мбит/с для передачи 12 000 кадров потребуется всего лишь несколько секунд. Кроме того, поскольку вектор инициализации передается в кадре открытым текстом, злоумышленник легко обнаружит, когда используется дубликат вектора инициализации.

Чтобы понять, чем плохо повторное использование ключа, рассмотрим следующий вариант атаки с произвольно выбираемым открытым текстом. Предположим, злоумышленник (возможно, подделывая IP-адреса) посылает Алисе запрос (например, HTTP- или FTP-запрос) на передачу файла с известным содержимым d^1 d^2, d^3, d^4,... Злоумышленник также наблюдает за последовательностью зашифрованных данных c^1 c^2, c^3, c^4,... Поскольку $d^i = c^i$ XOR k^v_i, мы можем, сложив с c^t обе стороны этого равенства по модулю 2, получить значение ключа:

$$d \text{ XOR } c = k^v_i.$$

Таким образом, зная d^t и c_t, злоумышленник может вычислить значение ключа k^{Iv}_i. В следующий раз, увидев то же значение вектора инициализации, злоумышленник будет знать последовательность ключей k^v_i, k^{2IV}, k^{3IV},... и таким образом сможет расшифровать зашифрованное сообщение. Это всего лишь одна из нескольких дыр в системе безопасности протокола WEP. Более подробное обсуждение этого вопроса содержится в [513, 546].

Резюме

В этой главе мы рассмотрели различные механизмы, которые могут помочь нашим тайным любовникам, Бобу и Алисе, сделать свою связь «безопасной». Как мы видели, Алиса и Боб заинтересованы в конфиденциальности (чтобы только они могли понять содержание пересылаемых друг другу сообщений), аутентификации (чтобы удостовериться в том, что общаются именно друг с другом) и целостности сообщений (чтобы удостовериться в том, что сообщения не были изменены по пути). Разумеется, безопасная связь нужна не только тайным любовникам. Как было показано в разделах «Управление доступом с помощью брандмауэров», «Атака и оборона» и «Безопасность на разных уровнях», безопасность требуется на различных сетевых уровнях для защиты от злоумышленников, обладающих обширным арсеналом средств нападения.

В первой части этой главы были представлены основополагающие принципы безопасной связи. В разделе «Принципы криптографии» мы рассмотрели криптографические методы шифрования и дешифрирования данных, включая шифрование с симметричными ключами и шифрование с открытым ключом, на примере алгоритмов DES и RSA, используемых в сегодняшних сетях. В разделе «Аутенти-

фикация» мы изучили вопросы аутентификации и разработали серию протоколов аутентификации возрастающей сложности. Эти протоколы должны были гарантировать, что собеседник действительно является тем, за кого он себя выдает, и что это «живой» человек, а не записанные ранее данные. Было показано, что и шифрование с симметричными ключами, и шифрование с открытым ключом могут играть важную роль не только в сокрытии данных (путем шифрования), но и в аутентификации. Методы снабжения цифрового документа «подписью», которую можно проверить и невозможно подделать (или отречься от нее), рассмотрены в разделе «Целостность данных». Здесь главную роль также играют криптографические методы. Цифровые подписи и дайджесты сообщений позволяют сократить время, требуемое для создания цифровой подписи, а также размер самой подписи, что крайне важно для длинных документов. Раздел «Передача ключей и сертификация» посвящен протоколам распространения ключей. Было показано, что для распространения общих симметричных ключей между общающимися по сети участниками соединения может использоваться центр распределения ключей — доверенный сетевой объект. Для шифрования с открытым ключом сертификационные центры распространяют сертификаты, подтверждающие открытые ключи.

Будем надеяться, Алиса и Боб — это студенты, прослушавшие курс о компьютерных сетях, а потому прочитавшие эту главу и знающие, как скрыть свою связь от зловредной Труди! Однако конфиденциальность представляет собой лишь небольшую часть проблемы сетевой безопасности. В последнее время все большее значение получает проблема защиты сетевой инфраструктуры от атак злоумышленников. В разделе «Управление доступом с помощью брандмауэров» мы поговорили о брандмауэрах, регулирующих доступ к защищаемой сети извне и к внешним объектам из сети, а в разделе «Атака и оборона» — о разнообразных атаках, которые могут предпринять находящиеся за пределами сети злоумышленники, а также о возможных контрмерах. Мы завершили эту главу разделом «Безопасность на разных уровнях» с примерами применения описанных методов на прикладном, транспортном, сетевом и канальном уровнях.

Вопросы и задания для самостоятельной работы

1. В чем различие между конфиденциальностью сообщений и целостностью сообщений? Возможно ли одно без другого? Аргументируйте свой ответ.
2. В чем различие между активным и пассивным злоумышленниками?
3. В чем принципиальное отличие системы с симметричными ключами от системы с открытым ключом?
4. Предположим, что злоумышленник сумел раздобыть зашифрованное сообщение, а также то же самое сообщение в виде открытого текста. Какую разновидность атаки сможет предпринять в этом случае злоумышленник?
5. Предположим, JV человек хотят общаться по сети с каждым из оставшихся $N - 1$ человек, используя шифрование с симметричными ключами. Все данные, которыми обменивается любая пара из группы, видимы всем остальным членам группы. Предполагается, что никто из группы, кроме двух обменива-

ющихся данными участников, не должен иметь возможности расшифровать эти данные. Сколько всего ключей потребуется для такой системы? Теперь предположим, что используется шифрование с открытым ключом. Сколько ключей понадобится в этом случае?

6. В чем заключается назначение нонса в протоколе аутентификации?
7. Что имеется в виду, когда говорится, что ноне представляет собой одноразовое значение?
8. Что означает атака с человеком посредине? Возможна ли такая атака при использовании симметричных ключей?
9. Что имеется в виду, когда говорят, что должна быть возможность проверить подписанный документ, не должно быть возможности подделать подписанный документ, не должно быть возможности отречься от него?
10. Почему дайджест сообщения обеспечивает лучшую проверку целостности сообщения, чем применяемая в Интернет-протоколах контрольная сумма?
11. Почему при создании цифровой подписи, как правило, открытым ключом шифруется дайджест сообщения, а не само сообщение?
12. Зашифровывается ли сообщение, объединенное со своим дайджестом? Аргументируйте свой ответ.
13. Что такое центр распределения ключей? Что такое сертификационный центр?
14. Перечислите основные различия в услугах, предоставляемых протоколами AH и ESP.

Упражнения

1. При помощи моноалфавитного шифра (см. рис. 7.3) зашифруйте сообщение «This is an easy prоЫет». Расшифруйте сообщение «rmij'u uamu xyj».
2. Покажите, что если злоумышленнику известен открытый текст семи зашифрованных посланий, число проверяемых подстановок в примере из подраздела «Шифрование с симметричными ключами» в разделе «Принципы криптографии» уменьшается приблизительно в 10^9 раз.
3. Рассмотрим полиалфавитную систему (см. рис. 7.4). Достаточно ли знать, как выглядит зашифрованная фраза «the quick brown fox jumps over the lazy dog», чтобы расшифровать все сообщения? Почему?
4. Зашифруйте фразу «hello» при помощи алгоритма RS A, используя значения;? = 3 и q = 11. Расшифруйте зашифрованное сообщение и получите исходную фразу.
5. Рассмотрите наш протокол аутентификации ap 4.0, который позволяет Бобу убедиться, что он действительно общается с Алисой. Как мы видели, этот протокол хорошо работал. Теперь предположим, что в то же самое время Боб также должен подтвердить свою личность Алисе. Приведите сценарий, в котором злоумышленник, притворившись Алисой, может обмануть Боба. Учтите, что последовательности операций протокола ap4.0, как инициированных злоумышленником, так и инициированных Бобом, могут произвольно чередоваться. Обратите особое внимание на тот факт, что Боб и Алиса используют нонсы,

поэтому, если не предпринять специальных мер, один и тот же none может быть задействован несколько раз.

6. В атаке с человеком посредине (см. рис. 7.13) Алиса не аутентифицирует Боба. Можно ли избежать подобной атаки при использовании протокола ap5.0, если потребовать от Боба прохождения процедуры аутентификации? Аргументируйте свой ответ.
7. В протоколе маршрутизации BGP для создания подписи BGP-сообщений используется алгоритм создания дайджеста сообщения MD5, а не шифрование с открытым ключом. Как вы думаете, почему предпочтение было отдано алгоритму MD5?
8. Придумайте третье сообщение, отличное от двух сообщений, показанных на рис. 7.18, но с тем же значением контрольной суммы.
9. Почему Боб не должен проходить явную процедуру аутентификации в протоколе, представленном на рис. 7.19?
10. Почему в протоколе, представленном на рис. 7.19, не используется явная процедура аутентификации? Необходима ли здесь аутентификация? Почему?
11. Рассмотрим сервер распределения ключей и сервер сертификационного центра. Предположим, сервер распределения ключей останавливается. Какое влияние это окажет на способность пользователей безопасно обмениваться данными? Аргументируйте свой ответ. Предположим теперь, что отключается сервер сертификационного центра. Что произойдет в этом случае?
12. Рассмотрим следующий вариант фильтрующего пакеты брандмауэра (см. раздел «Управление доступом с помощью брандмауэров»). Предположим, Алиса хочет запретить доступ к ее сети 222.22.0.0/16 из Интернета (правило R3 в табл. 7.4). Так же как и в прошлый раз, Алиса сотрудничает с Бобом и его коллегами из университета, поэтому Алиса хочет предоставить пользователям из университета Боба (адрес сети которого 111.11/16) доступ к специальной подсети 222.22.22/24 (правило R1). Алисе известно, что в университете Боба есть злоумышленник, адрес подсети которого 111.11.11 /24. Поэтому Алиса хочет запретить всякий трафик из этой подсети в свою сеть (правило R2), *кроме* как (новое правило R1) в специальную подсеть 222.22.22/24 (это исключение является важным отличием от нашего примера из раздела «Управление доступом с помощью брандмауэров»). Новые правила фильтрации пакетов сведены в следующую таблицу.

Правило	Адрес отправителя	Адрес получателя	Действие	Комментарии
R1	**111.11/16**	**222.22.22/24**	**Разрешить**	**Пропускать дейтаграммы из сети университета Боба в специальную подсеть**
R2	**111.11.11/24**	**222.22/16**	**Отказать**	**Не пропускать трафик из подсети злоумышленника в сеть Алисы**
R3	**0.0.0.0/0**	**0.0.0.0/0**	**Отказать**	**Не пропускать трафик в сеть Алисы**

Заполните показанную ниже таблицу, в которой рассматривается два варианта применения правил фильтрации.

Каким будет результат удаления правила R2 для пакетов P1, P2, P3 и P4?

Номер дейтаграммы	IP-адрес отправителя	IP-адрес получателя	Желаемое действие	Действие при R2, R1, R3	Действие при R1, R2, R3
P1	111.11.11.1 (подсеть хакера)	222.22.6.6 (корпоративная сеть)	Отказать		
P2	111.11.11.1 (подсеть хакера)	222.22.22.2 (специальная подсеть)	Разрешить		
P3	111.11.6.6 (университетская сеть, не подсеть хакера)	222.22.22.2 (специальная подсеть)	Разрешить		
P4	111.11.6.6 (университетская сеть, не подсеть хакера)	222.22.6.6 (корпоративная сеть)	Отказать		

13. На рис. 7.29 показаны действия, которые должна выполнить Алиса, чтобы добиться конфиденциальности, аутентификации и целостности сообщений. Изобразите операции, которые должен выполнить Боб с пакетом, полученным от Алисы.

Дополнительные вопросы и задания

1. Предположим, что злоумышленник может вставлять в сеть DNS-сообщения и удалять DNS-сообщения из сети. Представьте три сценария, иллюстрирующих возможные атаки злоумышленника.
2. Безопасность алгоритмов 3DES и RSA формально не доказана. Что тем не менее свидетельствует об их безопасности?
3. Если набор протоколов IPsec обеспечивает безопасность на сетевом уровне, почему возникает необходимость в алгоритмах безопасности на более высоких уровнях?
4. Зайдите на web-страницу программы PGP (http://www.pgpi.org/). Какую версию программы PGP вам разрешат загрузить (учитывая страну, в которой вы находитесь)?

Интервью

Стивен М. Белловин является стипендиатом AT&T в исследовательской лаборатории сетевых служб при AT&T Labs Research в Флорэм Парк, штат Нью-Джерси. Он занимается сетями и безопасностью, а также проблемами их несовместимости.

В 1995 году ему была присуждена премия Usenix Lifetime Achievement Award за работу по созданию сети Usenet, первой сетевой конференции, объединившей несколько компьютеров и позволившей пользователям обмениваться информацией и участвовать в дискуссиях. Стив М. Белловин также является выбранным членом Национальной инженерной академии. Он получил степень бакалавра наук в Колумбийском университете, а степень доктора философии в Университете Северной Каролины в Чэпл Хилл.

- Почему вы решили специализироваться в области сетевой безопасности?

  Это, может, покажется странным, но ответ элементарен:мне просто нравилось этим заниматься. Я получил образование в области системного программирования и системного администрирования, поэтому переход к вопросам сетевой безопасности был естественным. Кроме того, меня всегда интересовала проблема коммуникаций, это началось еще в колледже, когда я в свободное от учебы время занимался системным программированием.

  У моей деятельности в области безопасности два стимула — желание сохранить работоспособность компьютеров, защитив их от злоумышленников, и желание защитить право людей на конфиденциальность.

- Каким было ваше представление о сети Usenet, когда вы разрабатывали ее? И каково оно теперь?

  Изначально мы рассматривали сеть Usenet как средство обмена идеями о кибернетике и программировании, а также решения самых разных локальных административных вопросов, таких как реклама и пр. В то время я предполагал получать одно-два сообщения в день с 50-100 сайтов. Но настоящий рост оказался связанным с темами взаимоотношений между людьми, включая (но не ограничиваясь этим) темы взаимодействия человека с компьютером. С годами моими любимыми конференциями стали **rec. woodworking** *и* **sci. crypt.**

  До некоторой степени сетевые новости Usenet были заменены Всемирной паутиной. Если бы я начал разрабатывать Usenet сегодня, это была бы совсем другая система. Но и сейчас она представляет собой уникальный механизм доступа к огромной и заинтересованной определенным вопросом аудитории, не требующий никакой поддержки со стороны конкретных web-сайтов.

- Кто помог вам достичь успеха в вашей профессиональной деятельности?

  Профессор Фред Брукс — основатель и первый председатель факультета кибернетики Университета Северной Каролины в Чэпл Хилл, руководитель группы разработчиков операционных систем IBM S/360 и OS/360, а также автор книги «The Mythical Man Month» — оказал огромное влияние на мою карьеру. По большей части он учил мировоззрению и компромиссам — тому, как следует смотреть на проблемы в контексте реального мира (и насколько реальный мир сложнее и запутаннее, чем хотелось бы теоретику), как находить баланс противоположных интересов при поиске решения. Большая часть работы компьютерщика представляет собой инженерное искусство *— искусство нахождения правильных компромиссов для достижения множества противоречивых целей.*

- Каким вы видите будущее сетей и сетевой безопасности?

  До сих пор сетевая безопасность, по большей части, обеспечивалась путем изоляции. Например, брандмауэр предотвращает доступ к определенным машинам и службам. Но мы живем в эпоху, когда количество соединений и каналов связи все увеличивается. Поэтому изолировать объекты друг от друга становится все труднее. Что еще хуже, для наших производственных систем требуется все больше удаленных друг от друга частей, соединенных друг с другом сетями. Обеспечение безопасности таких систем является нашей самой сложной проблемой.

- Не могли бы вы рассказать нам, над какими проектами вы работаете сейчас?

  В основном я занимаюсь рабочими вопросами. Мало, чтобы система была безопасной, она должна быть еще и полезной. Это, в свою очередь, означает, что сетевые администраторы должны иметь возможность следить за происходящим. Но необходимость анализа и безопасность часто находятся в конфликте друг с другом — безопасность подразумевает сокрытие и защиту компьютеров и информации, в то время как администраторы должны выявлять определенные вещи. Кроме того, системы должны продолжать функционировать, даже если ключевые компоненты отключены. Как доставить информацию тому, кому она необходима, но так, чтобы не допустить к ней злоумышленников ? Именно поиском ответа на этот вопрос я сейчас и занимаюсь.

- Что, по-вашему, наиболее полезного сделано в области сетевой безопасности? Что еще предстоит сделать?

  По крайней мере, теоретически, мы знакомы с криптографическими методами. Это нам очень помогло. Однако большинство проблем безопасности вызвано ошибками в программах, и эта проблема очень сложна. В самом деле, это самая старая нерешенная проблема кибернетики, и я думаю, ее никогда не удастся решить. Задача заключается в том, чтобы создать безопасную систему из небезопасных компонентов. Мы уже добились определенных успехов в деле обеспечения надежности ненадежной аппаратуры. Возможно ли то же самое в области безопасности?

- Что вы можете посоветовать студентам, изучающим компьютерные сети?

  Изучить механизмы проще всего. Гораздо труднее научиться мыслить нестандартно. Вы должны помнить, что теория вероятностей здесь неприменима — злоумышленники ищут и находят самые невероятные сочетания условий. И помните, что основные проблемы, как обычно, кроются в мелочах.

ГЛАВА 8 Сетевое администрирование

Итак, осилив семь глав этой книги, мы хорошо понимаем, что сеть состоит из многих сложных аппаратных и программных элементов — линий связи, мостов, маршрутизаторов, хостов и прочих устройств, составляющих физические компоненты сети, а также различных протоколов (реализованных как аппаратно, так и программно), осуществляющих управление этими устройствами. Когда сотни или тысячи подобных компонентов собираются вместе, образуя сеть, не удивительно, что отдельные компоненты вдруг начинают неправильно работать, сетевые элементы оказываются настроенными неправильно, сетевые ресурсы используются неэффективно, а некоторые компоненты просто ломаются (например, кто-то спотыкается и рвет кабель или опрокидывает на маршрутизатор банку с пивом). Сетевой администратор, работа которого заключается в поддержании сети в работоспособном состоянии, должен уметь правильно реагировать на такие несчастные случаи (а еще лучше, не допускать их). При большом количестве сетевых компонентов, разбросанных по большой территории, очевидно, что сетевому администратору необходимы специальные средства, помогающие следить за состоянием сети и управлять ею. В этой главе мы изучим архитектуру, протоколы и информацию, используемые сетевым администратором при решении данной задачи.

Понятие сетевого администрирования

Прежде чем углубиться в вопросы сетевого администрирования, рассмотрим несколько примеров из реального (не сетевого) мира, в которых администратор должен следить за состоянием сложной системы, состоящей из множества взаимодействующих компонентов, контролировать ее работу, управлять ею. На электростанциях (по крайней мере, если верить популярному фильму «The China Syndrome») есть диспетчерская, где различные циферблаты, датчики и индикаторы показывают состояние (температуру, давление, скорость) удаленных клапанов, труб, камер и других компонентов электростанции, а некоторые из устройств могут даже сигнализировать о неполадках (знаменитые мигающие красные огоньки). Такие устройства помогают оператору контролировать текущее состояние станции и предпри-

нимать те или иные действия в зависимости от ситуации. Подобным же образом различными циферблатами и лампочками оснащена кабина самолета, позволяя пилоту следить за состоянием компонентов самолета и управлять ими. В этих двух примерах администратор *отслеживает состояние* (осуществляет мониторинг) удаленных устройств, *анализирует* показания датчиков, чтобы удостовериться в работоспособности устройств и соответствии их параметров заданным критериям (например, в том, что температура ядерного реактора не превышает определенного предела, или в том, что топлива в самолете *достаточно),реактивно контролирует* систему, то есть реагирует на изменение параметров системы или ее окружения, а также осуществляет *упреждающее управление* системой (например, обнаружив тенденции аномального поведения, предпринимает определенные действия по предотвращению серьезной проблемы). Аналогичным образом сетевой администратор следит за состоянием доверенной ему системы, контролирует ее и управляет ею.

На заре развития сетевой инфраструктуры, когда компьютерные сети представляли собой, скорее, объект исследования, нежели собственно инфраструктуру, которой пользуются миллионы людей в день, о сетевом администрировании никто не слыхал. Если в сети возникала проблема, источник проблемы обнаруживался при помощи нескольких команд ping, после чего пользователь, обнаруживший проблему, мог сам изменить параметры, перезагрузить аппаратуру или программное обеспечение, позвонить удаленному коллеге и попросить его сделать это. (Очень занимательное обсуждение первого крупного «краха» сети ARPAnet, случившегося 27 октября 1980 года, задолго до появления инструментов сетевого администрирования, а также усилий по устранению неисправности и попыток разобраться в причинах «краха», приводится в RFC 789.) По мере развития Интернета и превращения частных интрасетей в огромные глобальные структуры потребность в систематическом управлении огромным количеством аппаратных и программных компонентов этих сетей становилась все насущнее.

Начнем наше изучение сетевого администрирования с простого примера. На рис. 8.1 показана небольшая сеть, состоящая из трех маршрутизаторов и нескольких хостов и серверов.

Даже в такой простой сети возможны ситуации, в которых соответствующие инструменты сетевого управления окажутся необходимыми сетевому администратору.

- *Обнаружение неисправности интерфейсной карты хоста или маршрутизатора.* С помощью соответствующих инструментов сетевого управления сетевой объект (например, маршрутизатор А) может сообщить сетевому администратору о том, что один из его интерфейсов вышел из строя. (Разумеется, это предпочтительнее телефонного звонка в сетевой операционный центр от разгневанного пользователя, сообщающего о том, что сетевое соединение не работает!) Сетевой администратор, активно отслеживающий и анализирующий сетевой трафик, может обнаружить неисправность интерфейсной карты и заменить ее еще до того, как она окончательно выйдет из строя. Так, например, сетевой администратор может заметить увеличение ошибок контрольной суммы в кадрах, посылаемых «помирающим» интерфейсом.

- *Мониторинг хостов.* Сетевой администратор может периодически проверять состояние подключенных к сети хостов. И в этом случае он может обнаружить проблему прежде, чем она станет заметна пользователю.

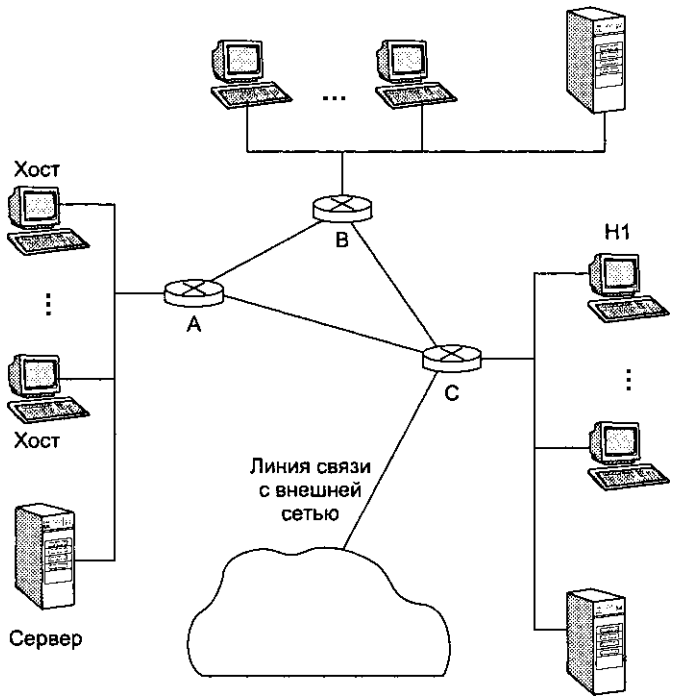


Рис. 8.1. Простой сценарий, иллюстрирующий задачи сетевого администрирования

- *Мониторинг трафика с целью контроля за распределением ресурсов.* Сетевой администратор может отслеживать рисунок сетевого трафика и заметить, например, что трафик, пересекающий множество локальных сетей, может быть существенно снижен, если соединить определенные серверы напрямую. Представьте, как обрадуются пользователи сетей (особенно руководители высшего звена), если увеличения производительности удастся добиться без дополнительных капиталовложений. Аналогично, отслеживая коэффициент использования линий, сетевой администратор может определить наличие перегрузки в сегменте локальной сети или канале связи с внешним миром, в связи с чем потребуется установить линию с большей пропускной способностью (увы, дополнительные расходы). Сетевой администратор также может настроить программу мониторинга, чтобы та автоматически уведомляла его о превышении определенного уровня загрузки линий — это даст возможность заранее подготовиться к высоким уровням загрузки и предотвратить серьезную перегрузку.
- *Обнаружение быстрых изменений в таблицах маршрутизации.* Частые изменения в таблицах маршрутизации указывают на нестабильность маршрутов или на неверно настроенный маршрутизатор. Разумеется, сетевой администратор

предпочтет сам обнаружить ошибку, прежде чем сеть окажется неработоспособной.

- *Мониторинг уровней обслуживания.* За последние несколько лет, с появлением соглашений об уровне обслуживания (Service Level Agreement, SLA), определяющих специфические параметры производительности и соответствующие им приемлемые показатели работы сети, заинтересованность в отслеживании трафика существенно возросла [224,292]. UUNet и AT&T — это всего лишь два из множества провайдеров, предоставляющих гарантии соблюдения соглашений SLA своим клиентам [18, 529]. Эти соглашения включают параметры доступности службы, задержки, пропускной способности и требования к уведомлению о простоях. Очевидно, что если критерии производительности должны входить в соглашение об обслуживании между сетевым провайдером и его клиентами, тогда измерение и контроль производительности являются важной задачей сетевого администратора.
- *Обнаружение вторжения.* Возможно, сетевой администратор хотел бы знать о случаях поступления сетевого трафика от подозрительного источника (например, хоста или номера порта) или к подозрительному получателю. Кроме того, сетевой администратор, возможно, хотел бы иметь возможность обнаруживать (и во многих случаях фильтровать) определенные типы трафика (например, пакеты с маршрутизацией от источника или большое количество SYN-пакетов, направляющихся к определенному хосту), которые могут быть признаками атак на систему безопасности (см. главу 7).

Международная организация по стандартизации (International Organization for Standardization, ISO) разработала модель сетевого администрирования. Были определены пять областей сетевого администрирования.

- *Контроль производительности.* Цель контроля производительности заключается в том, чтобы измерить производительность, сообщить о ней, проанализировать полученные данные и предпринять необходимые действия по поддержанию определенного уровня производительности (например, коэффициента использования или пропускной способности) в различных сетевых компонентах. Этими компонентами могут быть конкретные устройства (например, линии связи, маршрутизаторы, хосты) или абстракции (например, сетевые маршруты). Вскоре мы увидим, что центральную роль в управлении производительностью в Интернете играют стандарты протоколов (см. RFC 2570), таких как SNMP (Simple Network Management Protocol — простой протокол сетевого администрирования).
- *Контроль неисправностей.* Цель контроля неисправностей заключается в обнаружении неисправностей, их регистрации и принятия соответствующих ответных мер. Связь между контролем неисправностей и контролем производительности довольно расплывчата. Мы можем думать о контроле неисправностей как о немедленном исправлении неустойчивых сетевых поломок (например, аппаратных или программных сбоев линии связи, хоста или маршрутизатора), тогда как задача контроля производительности состоит в долгосрочном обеспечении приемлемых уровней производительности, несмотря на изменяющиеся

требования трафика и выход из строя сетевых устройств. Как и в контроле производительности, в контроле неисправностей центральную роль играет протокол SNMP.

- *Управление конфигурацией.* Управление конфигурацией позволяет сетевому администратору определить, какие устройства входят в администрируемую сеть, а также аппаратную и программную конфигурацию этих устройств. Обсуждение управления конфигурацией и требования к IP-сетям можно найти в RFC 3139.
- *Управление учетными записями.* Управление учетными записями позволяет сетевому администратору указать правила доступа пользователя или устройства к сетевым ресурсам, регистрировать запросы доступа, а также контролировать доступ. К управлению учетными записями также относятся квоты пользователей, взимание с пользователей платы, объем которой определяется используемыми ими ресурсами, а также предоставление пользователям привилегий на доступ к ресурсам.
- *Управление безопасностью.* Цель управления безопасностью заключается в контроле доступа к сетевым ресурсам в соответствии с некоторой политикой. Компонентами управления безопасностью являются центры распределения ключей и сертификационные центры, рассматривавшиеся в разделе «Передача ключей и сертификация» главы 7. Для мониторинга и контроля доступа к сетевым ресурсам извне используются брандмауэры, обсуждавшиеся в разделе «Управление доступом с помощью брандмауэров» той же главы.

В этой главе мы обсудим основы сетевого администрирования. Мы намеренно сужаем сферу рассматриваемых вопросов только *инфраструктурой* сетевого администрирования — общей архитектурой, протоколами сетевого управления и информационной базой, позволяющей сетевому администратору поддерживать сеть в рабочем состоянии. Мы не описываем процесс принятия сетевым администратором решений, который должен планировать, анализировать и реагировать на получаемую им информацию. К этой области относятся такие темы, как обнаружение и контроль неисправностей [260, 327], профилактическое обнаружение аномалий [518] и др. Мы также не станем затрагивать более широкую тему управления службами [462, 486]. В этой области применяются такие сложные и запутанные стандарты, как TMN [182,468] и TINA [160]. Например [197], стандарт TINA описывает «набор общих целей, принципов и концепций, относящихся к управлению службами и ресурсами, а также отчасти к распределенной среде обработки (Distributed Processing Environment, DPE)». Каждому из этих вопросов можно посвятить отдельную книгу. Поэтому, как мы уже отмечали, наша скромная цель заключается в обсуждении важных деталей инфраструктуры, позволяющей сетевому администратору управлять потоками битов.

Итак, что такое сетевое администрирование? Выше мы обсудили необходимость сетевого администрирования и привели несколько примеров. Мы завершим этот раздел определением сетевого администрирования из [445].

> *Сетевое администрирование включает разработку, интеграцию и координацию аппаратуры, программного обеспечения и людей для мониторинга,*

тестирования, опроса, конфигурирования, анализа, оценки и контроля сети и ресурсов для удовлетворения требований реального времени, производительности и качества обслуживания за приемлемую цену.

Это хорошее рабочее определение. В следующих разделах мы постараемся «нарастить плоть» на этот «скелет» определения сетевого администрирования.

Инфраструктура сетевого администрирования

В предыдущем разделе было показано, что для сетевого администрирования необходима возможность «мониторинга, тестирования, опроса, конфигурации... и контроля» аппаратных и программных компонентов сети. Поскольку сетевые устройства распределены на большой территории, для этого, как минимум, требуется, чтобы сетевой администратор мог собирать данные с удаленного объекта (например, для мониторинга) и производить необходимые изменения на этом удаленном объекте (например, контролировать объект). Чтобы лучше понять инфраструктуру сетевого администрирования, рассмотрим аналогию из мира людей.

Представьте, что вы руководите большой организацией с разветвленной сетью офисов по всему миру. Ваша работа состоит в том, чтобы гарантировать, что все фрагменты вашей организации работают слаженно друг с другом. Как этого добиться? Как минимум, вы должны периодически собирать информацию из ваших филиалов в виде отчетов и различных численных данных, отражающих активность, производительность и бюджет. Иногда (но не всегда) вас будут явно уведомлять о наличии проблем в одном из филиалов. Управляющий филиалом, желающий взобраться вверх по служебной лестнице (возможно, нацеливаясь на ваше место), может по собственной инициативе послать вам сообщение о том, как замечательно все работает в его офисе. Вы просматриваете получаемые сообщения, надеясь узнать, что все работает прекрасно, но, без сомнения, обнаружите проблемы, требующие вашего внимания. Вы можете начать диалог с одним из ваших филиалов, собрать больше информации, чтобы понять проблему, а затем передать менеджеру филиала нужную директиву.

В этом общем сценарии неявно задействована инфраструктура контроля организации — босс (вы), удаленные контролируемые сайты (филиалы), ваши удаленные агенты (менеджеры филиалов), протоколы связи (для передачи стандартных отчетов и диалогов) и данные (содержимое отчетов и численные данные, отражающие активность, производительность и бюджет). У каждого из этих компонентов есть аналоги в компьютерных сетях.

Архитектура сетевой системы управления концептуально идентична этому простому примеру. В сетевом управлении используется своя специфическая терминология. Как показано на рис. 8.2, архитектура сетевого управления состоит из трех основных компонентов: управляющего объекта (босса), управляемых устройств (филиалов) и протокола сетевого управления.

Управляющий объект представляет собой приложение, в работе которого, как правило, принимает участие человек, сидящий за терминалом в сетевом операционном центре. Управляющий объект — сердцевина сетевого управления. Он контролирует сбор, обработку, анализ и/или отображение информации. Именно здесь инициируются действия по управлению поведением сети, и здесь сетевой администратор (человек) взаимодействует с сетевыми устройствами.

Рис. 8.2. Основные компоненты архитектуры сетевого управления

Управляемое устройство представляет собой часть сетевого оборудования (включая программное обеспечение), располагающееся в управляемой сети. Это филиал в нашей аналогии. Управляемым устройством может быть хост, маршрутизатор, мост, хаб, принтер или модем. В управляемом устройстве может быть несколько так называемых *управляемых объектов*. Эти управляемые объекты являются физическими фрагментами управляемых устройств (например, сетевой интерфейсной картой), а также наборами настраиваемых параметров для этих аппаратных и программных фрагментов (например, протокол внутренней маршрутизации, такой как протокол RIP). В нашей аналогии из мира людей управляемые объекты могут быть отделами филиала. С этими управляемыми объектами ассоциированы определенные данные, объединяемые в *базу управляющей информации* (Management Information Base, MIB). Мы увидим, что эта информация доступна (и во многих случаях может задаваться) управляющему устройству.

В нашей аналогии из мира людей базе управляющей информации соответствуют количественные данные (измерения активности, производительности, финансовая информация, при этом управляющий объект может активно влиять на финансовые данные!), которыми обмениваются главный офис и филиалы. Более подробно мы рассмотрим базу управляющей информации в разделе «Архитектура управляющих Интернет-стандартов». Наконец, в каждом управляемом устройстве находится *агент сетевого управления,* то есть процесс, работающий в управляемом устройстве и непосредственно взаимодействующий с управляющим объектом. В нашей аналогии из мира людей агенту сетевого управления соответствует менеджер филиала.

Третий фрагмент нашей архитектуры сетевого управления представляет собой *протокол сетевого управления*. Этот протокол работает между управляющим объектом и управляемыми устройствами, позволяя управляющему объекту запрашивать состояние управляемых устройств и неявно предпринимать действия по управлению этими устройствами через его агентов. Агенты могут использовать протокол сетевого управления для информирования управляющего объекта об исключительных ситуациях (например, о выходе из строя определенного компонента или о нарушении показателей производительности). Важно отметить, что протокол сетевого управления сам по себе не управляет сетью, а является инструментом, с помощью которого сетевой администратор может управлять (осуществлять мониторинг, тестирование, опрос, конфигурирование, анализ, оценку и контроль) сетью. Это различие неочевидно, но важно.

Хотя инфраструктура сетевого управления концептуально проста, часто возникает путаница с терминологией сетевого управления, то есть такими понятиями, как «управляющий объект», «управляемое устройство», «управляющий агент», «база управляющей информации». Например, в нашем простом сценарии «управляющие агенты», располагающиеся на «управляемых устройствах», периодически опрашиваются «управляемым устройством» — идея проста, но язык можно сломать! В любом случае аналогия из мира людей будет нам полезна.

Выше мы обсудили общую архитектуру сетевого управления, которая применима ко многим стандартам сетевого управления. Стандарты сетевого управления начали свое становление в конце 80-х годов с международной программы стандартизации обмена данными между компьютерными системами различных производителей OSI (Open System Interconnection — взаимодействие открытых систем), в рамках которой были приняты такие стандарты, как *CMISE*(Common Management Information Services Element — служба общей управляющей информации) и *CMIP* (Common Management Information Protocol — протокол общей управляющей информации) [183, 388, 482], а также Интернет-протокол *SNMP* (Simple Network Management Protocol — простой протокол сетевого администрирования) [57, 425, 483]. Стандарты CMISE/CMIP и SNMP оказались наиболее важными [331,495]. Поскольку протокол SNMP был разработан и внедрен, когда необходимость в сетевом управлении стала очевидной, он быстро получил широкое распространение и стал наиболее популярным протоколом в архитектуре сетевого управления. Мы подробно рассмотрим его в следующем разделе.

Архитектура управляющих Интернет-стандартов

Сетевое управление в Интернете представляет собой нечто большее, чем просто протокол для перемещения данных между управляющим объектом и его агентами, и нечто более сложное, чем кажется, если судить по названию протокола (простой протокол сетевого управления). Современная архитектура управляющих Интернет-стандартов (Internet-Standard Management Framework) обязана своим происхождением (см. RFC 1028) протоколу SGMP (Simple Gateway Monitoring Protocol — простой протокол мониторинга шлюза). Протокол SGMP был разработан группой университетских исследователей, пользователей и администраторов, чей опыт работы с ним позволил спроектировать, реализовать и внедрить протокол SNMP [301]. С тех пор сменилось уже три версии протокола SNMP. Третья версия (RFC 2570) была выпущена в апреле 1999 года.

При описании любой архитектуры сетевого управления необходимо ответить на определенные вопросы.

- За чем (с семантической точки зрения) следует следить? И какую форму контроля должен применять сетевой администратор?
- В какой форме должна передаваться информация в отчетах и директивах?
- Какой протокол связи должен использоваться для обмена этой информацией?

Вспомним нашу аналогию из предыдущего раздела. Босс и менеджеры филиалов должны договориться о единицах измерения активности, производительности и бюджета, используемых в отчетах о состоянии филиалов. Кроме того, им следует договориться о мерах, которые может принимать босс (например, урезать бюджет, приказать менеджеру филиала изменить определенный аспект работы филиала или уволить сотрудников и закрыть филиал). На нижнем уровне детализации они должны договориться о форме, в которой будут передаваться отчеты. Например, в какой валюте (в долларах, евро) будут сообщаться финансовые данные? В каких единицах будет измеряться производительность? Наконец, следует определить способ, которым должна переправляться информация между головным офисом и филиалами (то есть протокол связи).

На эти вопросы отвечает архитектура управляющих Интернет-стандартов. Она состоит из четырех частей.

- Определения *объектов сетевого управления,* называемых MIB-объектами. В архитектуре управляющих Интернет-стандартов управляющая информация представляется как множество объектов, которые вместе образуют виртуальное хранилище информации, называемое базой управляющей информации (MIB). MIB-объект может быть счетчиком, например, измеряющим количество IP-дейтаграмм, отброшенных маршрутизатором из-за ошибок в заголовке, или количество ошибок опроса несущей в Ethernet-карте; описательная информация, например версия программного обеспечения, работающего на DNS-сервере; информация о состоянии устройства (например, правильно ли оно функционирует) или такая специфическая для протокола информация, как маршрут до

получателя. Таким образом, MIB-объекты определяют информацию управления, поддерживаемую управляемым устройством. Связанные MIB-объекты объединяются в *MIB-модули.* В нашей аналогии из мира людей база управляющей информации определяет информацию, переправляемую между филиалом и главным офисом.

- *Язык определения данных,* называемый структурой управляющей информации (Structure of Management Information, SMI), определяет типы данных, модель объектов и правила для записи и изменения управляющей информации. На этом языке описываются MIB-объекты. В нашей аналогии из мира людей язык SMI используется для определения деталей *формата* обмениваемых данных.
- *Протокол SNMP* служит для передачи данных и команд между управляющим объектом и агентом, работающим от имени этого объекта в управляемом сетевом устройстве.
- *Функции безопасности и управления.* Добавление этих функций отличает протокол SNMPv3 от протокола SNMPv2.

Таким образом, архитектура управляющих Интернет-стандартов состоит из отдельных модулей с независимыми от протокола языком определения данных и базой управляющей информации, а также с независимым от MIB протоколом. Интересно, что эта модульная архитектура изначально была разработана для облегчения перехода от сетевого управления, основанного на протоколе SNMP, к конкурирующей с этим стандартом архитектуре управляющих Интернет-стандартов, разрабатываемой Международной организацией по стандартизации (ISO). Этот переход так и не состоялся. Однако со временем модульность протокола SNMP позволила ему пройти через три ревизии, при этом были усовершенствованы все четыре составляющие протокола SNMP, независимо друг от друга. Таким образом, решение о модульном устройстве протокола SNMP было верным, несмотря на то что изначальная цель, для которой оно выбиралось, оказалась ложной.

В следующем разделе мы более подробно рассмотрим четыре основных компонента архитектуры управляющих Интернет-стандартов.

Структура управляющей информации

Структура управляющей информации (SMI) — это язык (весьма странное название языка), используемый для описания управляющей информации, хранящейся в объекте управляемой сети. Этот язык определений призван гарантировать, что синтаксис и семантика данных сетевого управления хорошо определены и непротиворечивы. Обратите внимание, что SMI определяет не конкретный элемент данных в объекте управляемой сети, а язык, на котором описывается подобная информация. Документация на язык SMI для протокола SNMPv3 (которая почему-то называется SMIv2, что вносит дополнительную путаницу) имеется в RFC 2578, RFC 2579 и RFC 2580. Рассмотрим структуру управляющей информации снизу вверх, начав с основных типов данных SMI, а затем поговорим о том, как управляемые объекты описываются в SMI, а также как связанные управляемые объекты группируются в модули.

Основные типы данных SMI

В RFC 2578 определены основные типы данных языка определения M1B-модулей, то есть языка SMI. Хотя язык SMI базируется на объектно-ориентированном языке ASN.1 [230, 231], о котором рассказывается в разделе «ASN.1», благодаря введению некоторых специфических типов данных SMI можно рассматривать как вполне самостоятельный язык. В табл. 8.1 перечислены 11 основных типов данных, определенных в RFC 2578. Кроме этих скалярных объектов на множество MIB-объектов также можно наложить табличную структуру с помощью конструкции SEQUENCE OF (подробности см. в RFC 2578). Большая часть представленных типов данных должна быть знакома большинству читателей. Более детально мы рассмотрим тип данных OBJECT IDENTIFIER, используемый для именования объекта.

Таблица 8.1. Основные типы данных SMI

Тип данных	Описание
INTEGER	**32-разрядное целое, как определено в ASN.1; значения могут изменяться в диапазоне от - 2^{31} до 2^{31} - 1**
Integer32	**32-разрядное целое; значения могут изменяться в диапазоне от - 2^{31} до 2^{31} - 1**
Unsigned32	**32-разрядное целое без знака; значения могут изменяться в диапазоне от 0 до 2^{32} - 1**
OCTET STRING	**Байтовая строка формата ASN.1 длиной до 65 535 байт, содержащая произвольные двоичные или текстовые данные**
OBJECT IDENTIFIER	**Список (структурированное имя) целых чисел формата ASN. 1 (см. подраздел «База управляющей информации» данного раздела)**
IPaddress	**32-разрядный Интернет-адрес, с тем порядком следования байтов, в котором они используются в сети**
Counter32	**32-разрядный циклический счетчик, увеличивающийся от 0 до 2^{32} - 1**
Counter64	**64-разрядный циклический счетчик**
Gauge32	**Целое 32-разрядное число без знака, не переходящее через О**
TimeTicks	**Время, измеряемое в сотых долях секунды, начиная от некоторого момента**
Opaque	**Неинтерпретируемая строка формата ASN.1, используемая для обратной совместимости**

Высокоуровневые конструкции SMI

Помимо основных типов данных язык определения данных SMI также предоставляет высокоуровневые конструкции языка.

Конструкция OBJECT IDENTIFIER используется для обозначения типа данных, состояния и семантики управляемого объекта. В подобных управляемых объектах содержатся данные, составляющие сердцевину сетевого управления. В различных документах RFC определено около 10 ООО объектов (RFC 2570). Конструкция OBJECT-TYPE состоит из четырех предложений. Предложение SYNTAX конструкции OBJECT-TYPE описывает тип основных данных, ассоциированный с объектом. Предложение MAX-ACCESS показывает, доступен ли управляемый объект для чтения,

записи, создания и т. д. Предложение **STATUS** показывает, является ли определение объекта действительным, устаревшим (в этом случае определение объекта не должно исполняться и остается только для истории) или частично устаревшим (в этом случае оно может исполняться для совместимости со старыми реализациями). Предложение **DESCRIPTION** содержит текстовое описание объекта, «документирующее» его назначение. Это описание должно предоставлять всю семантическую информацию, необходимую для управления объектом.

В качестве примера конструкции **OBJECT-TYPE** рассмотрим определение типа объекта **ipInDelivers** из RFC 2011. Этот объект определяет 32-разрядный счетчик, учитывающий количество IP-дейтаграмм, полученных управляемым устройством и успешно доставленных протоколу верхнего уровня. Последняя строка определения касается имени объекта (это мы обсудим в следующем разделе).

```
ipInDelivers OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
            "Суммарное количество входных дейтаграмм.
            успешно доставленных пользовательским протоколам IP
            (включая протокол ICMP)."
    ::- { ip 9 }
```

Конструкция **MODULE-IDENTITY** позволяет объединять связанные объекты в «модули». Например, в RFC 2011 описывается MIB-модуль, определяющий управляемые объекты (включая **ipInDelivers)** для управляющих реализаций протокола IP и связанного с ним протокола ICMP. В RFC 2012 описывается MIB-модуль для протокола TCP, а в RFC 2012 — для протокола UDP. MIB-модуль для удаленного монито-**pHHra** (Remote MONitoring, RMON) описывается в RFC 2021. Конструкция **MODULE-IDENTITY** содержит предложения для документирования сведений об авторе модуля, даты последнего обновления, истории обновлений и текстового описания модуля. Для примера рассмотрим определение модуля для управления протоколом IP:

```
ipMIB   MODULE-IDENTITY
    LAST-UPDATED  "9411010000Z"
    ORGANIZATION  "IETF SNMPv2 Working Group"
    CONTACT-INFO
                    Keith McCloghrie
            Postal: Cisco Systems. Inc.
                    170 West Tasman Drive
                    San Jose. CA 95134-1706
                    US
            Phone:  +1 408 526 5260
            E-mail: kzm@cisco.com"
    DESCRIPTION
             "The MIB module for managing IP and ICMP
             implementations, but excluding their
             management of IP routes."
    REVISION  "9103310000Z"
    DESCRIPTION
             "The initial revision of this MIB module
              was part of MIB-11."
    ::= { mib-2 48}
```

Конструкция **NOTIFICATION-TYPE** применяется для спецификации информации, касающейся сообщений «SNMPv2-Trap» и «InformationRequest», генерируемых агентом или управляющим объектом (см. подраздел «Операции и транспортное соответствие протокола SNMP» данного раздела). Эта информация включает текстовое описание (предложение **DESCRIPTION)** того, когда могут посылаться подобные сообщения, а также список значений, которые должны включаться в генерируемое сообщение (подробности см. в RFC 2578). Конструкция **MODULE-COMPLIANCE** определяет набор управляемых объектов в модуле, который должен быть реализован агентом. Конструкция **AGENT-CAPABILITIES** описывает возможности агента в отношении определений объекта или уведомлений о событиях.

База управляющей информации

Ранее уже отмечалось, что *базу управляющей информации* (MIB) можно рассматривать как виртуальное хранилище управляемых объектов, значения которых совместно отражают текущее состояние сети. Эти значения могут запрашиваться и/или устанавливаться управляющим объектом при помощи SNMP-сообщений, посылаемых агенту, работающему на управляемом устройстве от имени управляющего объекта. Управляемые объекты специфицируются при помощи обсуждавшейся выше конструкции **OBJECT-TYPE** языка SMI и объединяются в *MIB-модули* с помощью конструкции **MODULE-IDENTITY.**

Стандарты MIB-модулей для маршрутизаторов, хостов и другого сетевого оборудования были разработаны группой IETF. В MIB-модули должны входить основные идентификационные данные об определенном фрагменте аппаратуры, а также управляющая информация о сетевых интерфейсах и протоколах устройства. На момент выхода третьей версии протокола SNMP (середина 1999 года) было около сотни стандартных и еще больше нестандартных (фирменных) MIB-модулей отдельных производителей. При таком количестве стандартов группе IETF требовался способ идентификации и именования стандартных MIB-модулей, а также специфических управляемых объектов внутри модулей. Вместо того чтобы начать с нуля, группа IETF воспользовалась стандартизированной структурой идентификации (именования) объектов, уже разработанной международной организацией по стандартизации (ISO). Как это часто случается со многими стандартами, у ISO были «большие планы» относительно стандартизированной структуры идентификации объектов. ISO намеревалась идентифицировать все стандартизированные объекты (например, форматы данных, протоколы или фрагменты информации) всех сетей, независимо от сетевых стандартов (например, IETF, ISO, IEEE или ANSI), производителя оборудования или владельца сети. Цель была действительно грандиозная! Разработанная ISO структура идентификации объектов является частью объектно-ориентированного языка ASN. 1 [230,231], которому посвящен одноименный раздел. В этой всеохватывающей структуре именования у стандартизированных MIB-модулей есть свой уютный уголок.

Как видно на рис. 8.3, объекты в структуре именования ISO именуются иерархическим образом. Обратите внимание, что каждая ветвь дерева имеет как имя, так и номер (показанный в скобках). Таким образом, каждый узел дерева идентифици-

руется последовательностью имен или чисел, определяющих путь от корня дерева к его узлу. Идентификационное дерево (основанное на информации, поставляемой добровольцами) можно найти на web-сайте http://www.alvestrand.no/harald/objectid/top.html.

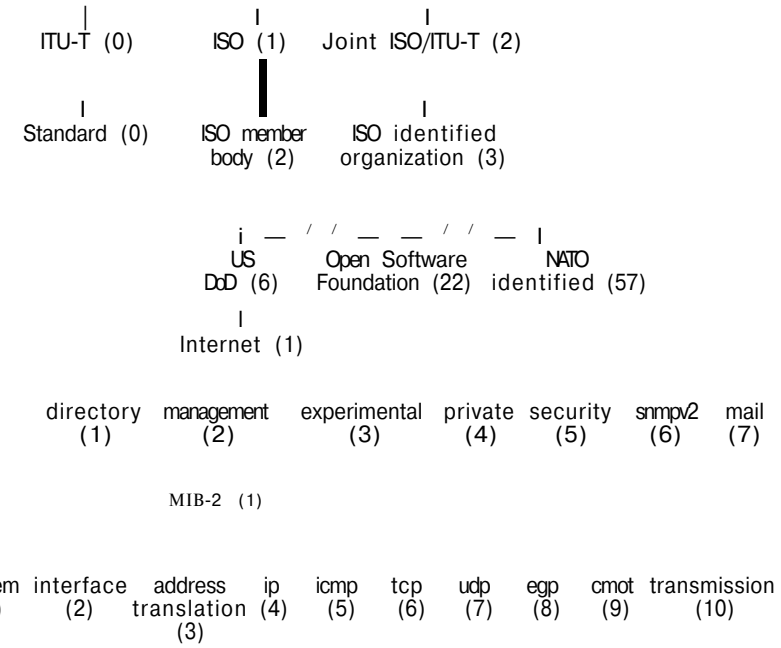


Рис. 8.3. Идентификационное дерево объектов ASN.1

На вершине дерева располагаются две главные организации по стандартизации, имеющие дело с ASN.1, — ISO и ITU-T (сектор телекоммуникаций союза ITU), а также ветвь, отражающая плоды их совместных усилий. Под узлом ISO мы обнаруживаем узлы для всех стандартов ISO (1.0), а также для стандартов, изданных организациями по стандартизации различных стран-членов ISO (1.2). Хотя это и не показано на рисунке, США также присутствует на этом дереве, в узле 1.2.840, под которым располагаются такие организации, как IEEE и ANSI, а также стандарты отдельных компаний, например RSA (1.2.840.11359) и Microsoft (1.2.840.113556). Под узлом корпорации Microsoft помещается узел для Microsoft File 'Formats (1.2.840.113556.4) со стандартами форматов файлов для различных продуктов корпорации Microsoft, таких как Word (1.2.840.113556.4.2). Но в данной книге нас, прежде всего, интересуют компьютерные сети (а не файлы Microsoft Word), поэтому рассмотрим ветвь 1.3, на которой «зреют» стандарты, изданные организациями, признаваемыми ISO. К ним относятся Министерство обороны США (1.3.6), под узлом которого мы обнаружим Интернет-стандарты (1.3.6.1), фонд Open Software Foundation (1.3.22), ассоциация авиалиний SITA (1.3.69), NATO (1.3.57) и многие другие.

Под узлом Internet (1.3.6.1) располагается список [236] имен и кодов предприятий более чем 4000 частных компаний, зарегистрированных в уполномоченной организации по назначению номеров Интернета (Internet Assigned Numbers Authority, IANA) [235]. Под узлами management (1.3.6.1.2) и MIB-2 (1.3.6.1.2.1) дерева идентификации объектов располагаются определения стандартизированных MIB-модулей.

На нижнем уровне дерева на рис. 8.3 показаны некоторые важные аппаратно-ориентированные MIB-модули (system и interface), а также модули, ассоциированные с основными Интернет-протоколами. В RFC 3000 перечислены все стандартизированные MIB-модули. И хотя относящиеся к MIB-модулям документы RFC написаны довольно скучным и сухим языком, будет полезно (подобно тому как полезно питаться овощами) рассмотреть определения некоторых MIB-модулей, чтобы лучше понять, какого рода информация содержится в модуле.

Управляемые объекты под узлом system содержат общую информацию об управляемых устройствах; системные MIB-объекты должны поддерживать все управляемые устройства. В табл. 8.2 перечислены управляемые объекты группы system в соответствии с RFC 1213, а в табл. 8.3 — управляемые объекты MIB-модуля для протокола UDP.

Таблица 8.2, Управляемые объекты группы MIB-2.system

Идентификатор объекта	Название	Тип	Описание (из RFC 1213)
1.3.6.1.2.1.1.1	sysDescr	OCTET STRING	Полное название и версия типа аппаратного обеспечения системы, операционной системы и сетевого программного обеспечения
1.3.6.1.2.1.1.2	sysObjectID	OBJECT IDENTIFIER	Назначаемый производителем идентификатор объекта, обеспечивающий простой и точно выраженный способ определить, что представляет собой управляемый объект
1.3.6.1.2.1.1.3	sysUpTime	TimeTicks	Время (в сотых долях секунды) с момента последней инициализации системы
1.3.6.1.2.1.1.4	sysContact	OCTET STRING	Человек, отвечающий за управление данным узлом, и его адрес или телефон
1.3.6.1.2.1.1.5	sysName	OCTET STRING	Административно назначаемое название управляемого узла. По соглашению, это полное имя домена
1.3.6.1.2.1.1.6	sysLocation	OCTET STRING	Физическое расположение узла
1.3.6.1.2.1.1.7	sysServices	Integer32	Код, обозначающий набор служб, доступных этому узлу: физических (например, для повторителя), канальных (например, для моста), служб Интернета (например, для IP-шлюза), сквозных (например, для хоста), прикладных

Таблица 8,3, Управляемые объекты модуля MIB-2 для протокола UDP

Идентификатор объекта	Название	Тип	Описание (из RFC 1213)
1.3.6.1.2.1.7.1	udpInDatagrams	Counter32	Суммарное количество UDP-дейтаграмм, доставленных UDP-пользователям
1.3.6.1.2.1.7.2	udpNoPorts	Counter32	Суммарное количество полученных UDP-дейтаграмм, для которых нет приложения в порту получателя
1.3.6.1.2.1.7.3	udpInErrors	Counter32	Суммарное количество полученных UDP-дейтаграмм, которые не могут быть доставлены по причинам, не имеющим отношения к отсутствию приложения в порту получателя
1.3.6.1.2.1.7.4	udpOutDatagrams	Counter32	Суммарное количество UDP-дейтаграмм, отправленных данным объектом
1.3.6.1.2.1.7.5	udpTable	SEQUENCE of UdpEntry	Последовательность используемых приложением объектов UdpEntry, содержащих IP-адрес и номер порта по одному для каждого порта, открытого текущим приложением

Операции и транспортное соответствие протокола SNMP

Протокол SNMP (Simple Network Management Protocol — простой протокол сетевого администрирования), описанный в RFC 1905, используетсядля передачи M1Ё-информации между управляющими объектами и агентами, работающими от их имени. Чаще всего протокол SNMP используется в *режиме запрос-ответ,* в котором управляющий SNMPv2-объект посылает запрос SNMPv2**-areHTy,** который получает запрос, выполняет определенное действие и посылает ответ на запрос. Как правило, запрос требуется, чтобы узнать (получить) или изменить (установить) значения MIB-объекта, ассоциированные с управляемым объектом. Кроме того, протокол SNMP может применяться агентом для отправки управляющему объекту сообщений, отправляемых без запросов и называемых *прерывающими сообщениями.* Прерывающие сообщения используются для уведомления управляющего объекта об исключительных ситуациях, возникших в результате изменения значений MIB-объектов. В разделе «Понятие сетевого администрирования» было показано, что сетевой администратор может быть заинтересован в получении прерывающих сообщений, например в случае выхода интерфейса из строя или определенного уровня перегрузки. Не забывайте о существенной разнице между режимом запрос-ответ и режимом прерываний (см. упражнения в конце главы).

Протоколом SNMPv2 определены семь типов сообщений, как правило, называемых единицами обмена (табл. 8.4). Формат единиц обмена показан на рис. 8.4.

- Единицы обмена GetRequest, GetNextRequest и GetBulkRequest отправляются управляющим объектом агенту для запроса значения одного или нескольких MIB-

объектов на устройстве, управляемом агентом. Идентификаторы M1B-объектов указываются в переменной части единицы обмена. Единицы обмена **GetRequest, GetNextRequest** и **GetBulkRequest** отличаются степенью детализации запрашиваемых данных. При помощи единицы обмена **GetRequest** можно запросить произвольное количество MIB-значений. Для работы со списком или таблицей MIB-объектов можно воспользоваться несколькими единицами обмена **GetNextRequest.** Единица обмена **GetBulkRequest** позволяет получить большой блок данных, избегая накладных расходов, связанных с отправкой нескольких сообщений **GetRequest** или **GetNextRequest.** Во всех трех случаях агент отвечает сообщением **Response,** в котором содержатся идентификаторы объектов и ассоциированные с ними значения.

- Единица обмена **SetRequest** используется управляющим объектом для установки значения одного или нескольких MIB-объектов в управляемом устройстве. Агент отвечает единицей обмена **Response,** в поле состояния ошибки которой помещается значение **noError** (без ошибки) в подтверждение того факта, что значение действительно установлено.
- Единица обмена **Inform Request** используется управляющим объектом для передачи другому управляющему объекту MIB-информации. Получающий объект отвечает единицей обмена **Response,** в поле состояния ошибки которой помещается значение **noError** (без ошибки), подтверждающее факт получения единицы обмена **Inform Request.**
- Последним типом единицы обмена протокола SNMPv2 является прерывающее сообщение. Прерывающие сообщения генерируются асинхронно, то есть не в ответ на запрос, а в ответ на событие, уведомление о котором необходимо управляемому объекту. В RFC 1907 определяются типы прерываний, включая холодный или горячий запуск устройства, включение или отключение канала связи, исчезновение соседа, неудачно проведенная аутентификация. Управляющее устройство не обязано отвечать на полученное прерывающее сообщение.

Таблица 8,4. Типы единиц обмена протокола SNMPv2

Тип единицы обмена	Отправитель-получатель	Описание
GetRequest	От управляющего агенту	Получить значение одного или нескольких экземпляров M1B-объекта
GetNextRequest	От управляющего агенту	Получить значение следующего экземпляра MIB-объекта в списке или таблице
GetBulkRequest	От управляющего агенту	Получить значения в большом блоке данных, например значения в большой таблице
InformRequest	От управляющего управляющему	Информировать удаленный управляющий объект о значениях MIB-объектов

продолжение &

Таблица 8.4 *(окончание)*

Тип единицы обмена	Отправитель-получатель	Описание
SetRequest	От управляющего агенту	Установить значение одного или нескольких экземпляров M1B-объекта
Response	От агента управляющему или от управляющего управляющему	Генерируется в ответ на единицу обмена GetRequest, GetNextRequest, GetBulkRequest, SetRequest или InformRequest
SNMPv2-Trap	От агента управляющему	Информирует управляющий объект об исключительном событии

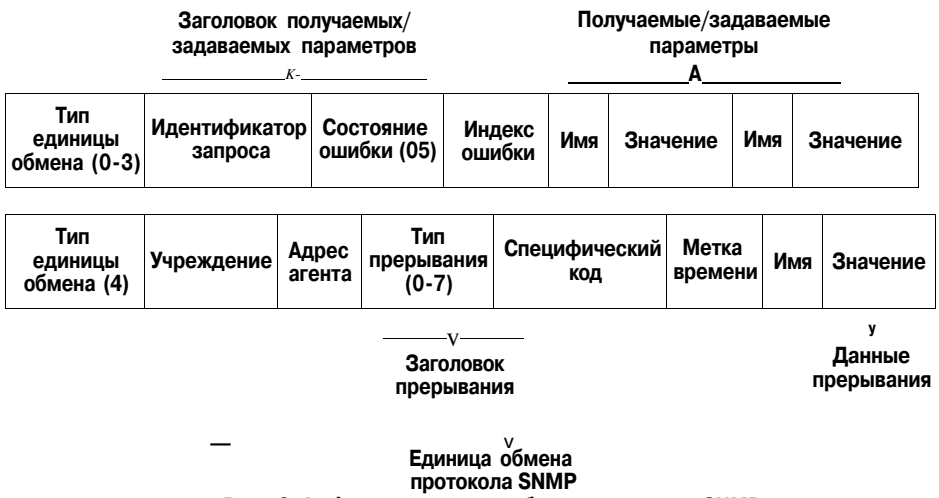


Рис. 8.4. Формат единицы обмена протокола SNMP

Хотя единицы обмена могут переноситься при помощи различных транспортных протоколов, как правило, для этой цели используются UDP-дейтаграммы. В самом деле, то, что протокол UDP является «предпочтительным транспортом», утверждается в RFC 1906. Поскольку UDP представляет собой ненадежный транспортный протокол, нет никакой гарантии, что запрос или ответ будет получен тем, кому он направлен. Поле идентификатора запроса единицы обмена используется управляющим объектом для нумерации запросов, посылаемых агентам. Это значение помещается в ответное сообщение агента. Таким образом, поле идентификатора запроса позволяет управляющему объекту обнаружить потерянный запрос или ответ. Решение о том, повторять или нет запрос, на который не получен ответ в установленный срок, принимает управляющий объект. В частности, стандартом SNMP не предписывается какая-либо специальная процедура повторной передачи и даже сама необходимость повторной передачи. Там только сказано, что управляющий объект «должен действовать ответственно в отношении частоты и длительности повторных передач». Само собой, это заставляет крепко задуматься над тем, как ведут себя «ответственные» протоколы.

Безопасность и администрирование

Разработчики протокола SNMPv3 заявляют, что он может рассматриваться как протокол SNMPv2 с дополнительными возможностями в области безопасности и администрирования (RFC 2570). Действительно, между протоколами SNMPv3 и SNMPv2 есть отличия, но нигде они не заметны так, как в областях безопасности и администрирования. Центральная роль безопасности в протоколе SNMPv3 особенно важна, так как из-за недостатка адекватных механизмов безопасности протокол SNMP применялся в основном не для контроля, а для мониторинга (например, единица обмена SetRequest в протоколе SNMPvl почти не используется).

Функциональность протокола SNMP к третьей версии выросла, однако вместе с ней, к сожалению, увеличился и объем документации. Чего стоит один лишь факт существования документа RFC 2571, в котором «описывается архитектура для описания архитектуры управления SNMP»! От «архитектуры для описания архитектуры» может закружиться голова, но цель RFC 2571 заключается в создании общего языка описания функциональности и действий, предпринимаемых SNMPv3-**areH**-том или управляющим SNMPv3-объектом. Архитектура **SNMPv3-cyuJHOCTH** проста, и ее изучение поможет нам лучше понять протокол SNMP.

К так называемым *SNMP-приложениям* относятся генератор команд, приемник уведомлений, прокси-передатчик (эти три элемента, как правило, присутствуют в управляющем объекте), ответчик команд, генератор уведомлений (эти два элемента, как правило, присутствуют в агенте) и др. Генератор команд генерирует единицы обмена GetRequest, GetNextRequest, GetBulkRequest и SetRequest, которые мы рассматривали в подразделе «Операции и транспортное соответствие протокола SNMP» данного раздела, а также обрабатывает получаемые ответы на эти единицы обмена. Ответчик команд (приложение, отвечающее на команды) выполняется на агенте. Он получает, обрабатывает и отвечает (с помощью сообщения Response) на приходящие единицы обмена GetRequest, GetNextRequest, GetBulkRequest и SetRequest. Работающий на агенте генератор уведомлений генерирует единицы обмена Trap. Эти единицы обмена принимаются и обрабатываются приемником уведомлений на управляющем объекте. Прокси-передатчик занимается продвижением запросов, уведомлений и ответных единиц обмена.

Посылаемая SNMP-приложением единица обмена, прежде чем попасть к соответствующему транспортному протоколу, проходит через «движущий механизм» протокола SNMP. На рис. 8.5 показано, как генерируемая генератором команд единица обмена сначала попадает в модуль диспетчеризации, где определяется версия протокола SNMP. Затем единица обмена обрабатывается системой подготовки сообщений, где к ней добавляется заголовок сообщения, содержащий версию протокола SNMP, идентификатор сообщения и размер сообщения. Если требуется шифрование или аутентификация, то в заголовок также включаются соответствующие поля (детали см. в RFC 2571). Наконец, SNMP-сообщение (созданная приложением единица обмена плюс заголовок) передается соответствующему транспортному протоколу. Как правило, для передачи SNMP-сообщений используются протокол UDP (то есть SNMP-сообщение помещается в поле полезной нагрузки UDP-дейтаграммы) и порт 161. Для прерывающих сообщений используется порт 162.

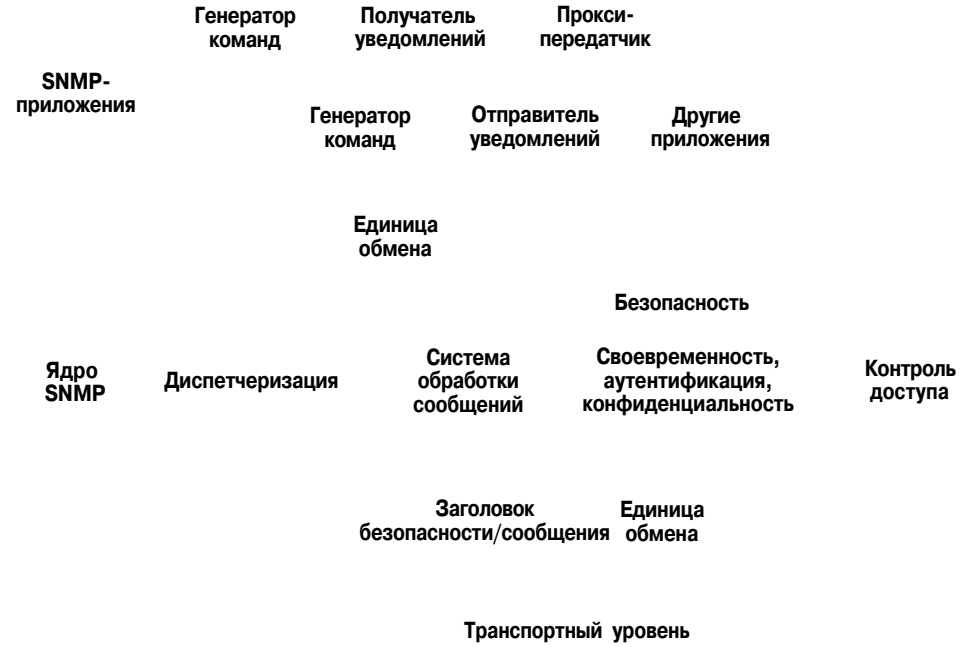


Рис. 8.5. Приложения протокола SNMPv3

Итак, мы выяснили, что SNMP-сообщения предназначаются не только для мониторинга, но и для контроля (например, с помощью команды SetRequest) сетевых элементов. Очевидно, злоумышленник, способный интерпретировать SNMP-сообщения, а также генерировать собственные SNMP-пакеты, может причинить немало неприятностей в сети. Таким образом, важно, чтобы SNMP-сообщения пересылались в зашифрованном виде. Удивительно, но вопросу безопасности было уделено подобающее внимание лишь в самой последней версии протокола SNMP. В протоколе SNMPv3 применяется (см. RFC 2574) модель так называемой «пользовательской безопасности» (User-Based Security, USM). В этом случае такая информация, как значение ключа или привилегии доступа, ассоциируется с пользователем, который идентифицируется по имени и паролю. Протокол SNMPv3 предоставляет шифрование, аутентификацию, защиту от атак повторного воспроизведения (см. разделы «Принципы криптографии» и «Аутентификация» в главе 7), а также контроль доступа.

- *Шифрование.* Единицы обмена протокола SNMP могут быть зашифрованы по стандарту DES в режиме сцепления блоков шифра (см. раздел «Принципы криптографии» в главе 7). Обратите внимание, что, поскольку DES представляет собой систему шифрования с общим ключом, секретный ключ должен быть известен получателю зашифрованных данных.
- *Аутентификация.* Протокол SNMP объединяет хэш-функцию (см. раздел «Целостность данных» в главе 7) со значением секретного ключа, чтобы

одновременно обеспечивать аутентификацию и защиту от злоумышленников. Концепция метода HMAC (Hashed Message Authentication Codes — коды аутентификации хэшированных сообщений) проста (RFC 2104). Предположим, у отправителя есть единица обмена *m* протокола SNMP, которую он хочет переслать получателю. Возможно, эта единица обмена уже зашифрована. Предположим также, что отправителю и получателю известен общий секретный ключ iC, не обязательно тот же самый, что используется для шифрования данных. Отправитель посылает получателю единицу обмена *m*. Но вместо того чтобы просто послать вместе с сообщением код целостности сообщения (Message Integrity Code, MIC) *MIC(m)*} вычисляемый по значению *m* (см. раздел «Целостность данных» в главе 7), чтобы защититься от злоумышленников, отправитель прилагает к *m* код *M1C(m, K),* то есть код целостности сообщения, вычисленный одновременно по значению *m* и ключу K Получив *m,* получатель добавляет к нему секретный ключ *K* и вычисляет *M1C(m, K).* Если вычисленное получателем значение *M1C(m, K)* совпадает со значением, переданным отправителем, тогда получатель может быть уверен не только в том, что сообщение не было изменено злоумышленником, но также в том, что оно было послано тем, кому известно значение секретного ключа *K.* Таким образом, одновременно с целостностью данных обеспечивается аутентификация отправителя. Операция конкатенации и вычисления хэш-кода производится алгоритмом HMAC дважды, при этом значение ключа каждый раз слегка изменяется (детали см. в RFC 2104).

- *Защита от атак повторного воспроизведения.* Как уже отмечалось в главе 7, для защиты от атак повторного воспроизведения могут использоваться нонсы. Соответствующий метод применяется и в протоколе SNMPv3. Чтобы гарантировать, что полученное сообщение представляет собой не повторное воспроизведение какого-либо ранее посланного сообщения, получатель требует, чтобы отправитель в каждое сообщение включал определенное значение, основанное на счетчике получателя. Этот счетчик отражает интервал времени с момента последней перезагрузки программного обеспечения сетевого управления получателя, а также суммарное число перезагрузок с момента последнего конфигурирования программного обеспечения сетевого управления получателя. До тех пор пока значение счетчика в полученном сообщении находится в определенных допустимых пределах, сообщение принимается и может быть расшифровано и/или аутентифицировано. Детали см. в RFC 2574.
- *Контроль доступа.* Протокол SNMPv3 обеспечивает контроль доступа (RFC 2575), то есть определяет управляющую информацию, которая может запрашиваться и/или изменяться пользователями. SNMP-объект хранит информацию о правах доступа и политиках в локальной базе данных конфигурации (Local Configuration Datastore, LCD). К фрагментам этой базы данных предоставляется доступ как к управляемым объектам (RFC 2575), и, таким образом, ими можно управлять дистанционно с помощью протокола SNMP.

ПРИНЦИПЫ И ПРАКТИКА

На сегодняшний день существуют сотни (если не тысячи) программ сетевого управления, в каждой из которых в какой-то мере реализованы архитектура управляющих Интернет-стандартов и основы SNMR обсуждаемые в этом разделе. Обзор этих программных продуктов выходит за рамки темы книги. Поэтому здесь мы приведем лишь ссылки на информацию об этих продуктах. Изучение вопроса хорошо начать с главы 12 в [495].

Инструменты сетевого управления можно разделить на те, которые предназначены для управления оборудованием какого-либо определенного производителя, и те, которые предназначены для управления сетями с разнородным оборудованием. К первой группе инструментов относится система CiscoWorks2000 [88], предназначенная для управления локальными и глобальными сетями, построенными на основе устройств компании Cisco. Система Nortel Optivity Network Management System [326] обеспечивает управление сетями, службами и политиками.

Среди популярных инструментальных средств для управления сетями с разнородным оборудованием можно назвать систему OpenView компании Hewlett-Packard [216], систему Spectrum [14] компании Aprlsma, а также систему сетевого управления Solstice компании Sun [496]. Во всех трех системах реализована архитектура распределенных систем, в которой множество серверов собирают управляющую информацию в доменах, которыми они управляют. Затем станция сетевого управления может получить у этих серверов результаты, отобразить их и предпринять корректирующие действия. Все эти продукты поддерживают протоколы SNMP и CMIP, а также помогают связать событие и сигнал тревоги.

ASN.1

В этой книге мы рассмотрели множество интересных вопросов, относящихся к компьютерным сетям. Раздел о языке ASN.1 (Abstract Syntax Notation One — нотация абстрактного синтаксиса, версия 1), вероятно, нельзя отнести к десятке самых занимательных. Как и поедание шпината, изучение языка ASN.1, говорят, может оказаться полезным. ASN.1 представляет собой стандарт, выпущенный ISO и используемый в ряде протоколов Интернета, относящихся в основном к сфере сетевого управления. Например, в разделе «Архитектура управляющих Интернет-стандартов» было показано, что MIB-переменные протокола SNMP неразрывно связаны с языком ASN.1. Поэтому, несмотря на сухость материала о языке ASN.1 в данном разделе, мы надеемся, читатель примет на веру утверждение о его важности.

Проведем следующий мысленный эксперимент. Предположим, мы имеем возможность надежно копировать данные из памяти нашего компьютера в память удаленного компьютера. Решит ли это проблему связи? Ответ на этот вопрос зависит от того, что мы называем «проблемой связи». Очевидно, подобное идеальное копирование из памяти в память обеспечит передачу битов и байтов между компьютерами. Но означает ли подобное точное копирование, что программное обеспечение, работающее на принимающем компьютере, обратившись к данным, получит те же значения, которые хранятся в памяти передающего компьютера? Оказывается, что этот вопрос не так прост, как может показаться.

Дело в том, что на компьютерах с различной архитектурой, с различными операционными системами и разными компиляторами используются разные соглашения для хранения и представления данных. Если данные должны передаваться с одного компьютера на другой, проблему представления данных необходимо решить.

Рассмотрим эту проблему на примере простого фрагмента программы на языке C. Как может эта структура располагаться в памяти?

```
struct {
  char code;
  int x;
  } test;
test.x = 259;
test.code = 'a';
```

Слева на рис. 8.6 показано возможное расположение данных в гипотетической архитектуре: на символ 'a' выделяется один байт, за которым следует 16-разрядное слово, содержащее целое значение 259, начиная со старшего байта. Расположение той же структуры данных на другом компьютере показано в правой части рисунка. Следом за символом 'a' располагается целое значение, хранящееся, начиная с младшего байта и к тому же выровненное по границам слов (по четным байтам). Очевидно, простое копирование памяти с одного компьютера на другой здесь неприменимо.

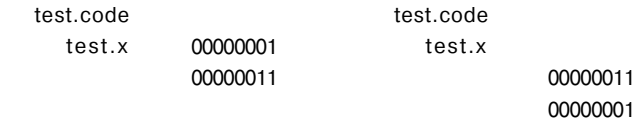


Рис. 8.6. Две формы хранения данных

Тот факт, что в компьютерах с разной архитектурой используются разные внутренние форматы хранения данных, представляет сложную проблему. Формат с прямым порядком следования байтов (самым первым хранится самый старший байт) используется в процессорах SPARC и Motorola, тогда как формат с обратным порядком следования байтов (самым первым хранится самый младший байт) используется в процессорах фирмы Intel и процессорах Alpha компаний DEC/Compaq. В англоязычной литературе эти два формата называются «тупоконечным» и «остроконечным» по аналогии с лилипутами из книги Джонатана Свифта «Путешествие Гулливера», которые никак не могли договориться, с какой стороны разбивать вареные яйца. В ситуации с хранением данных спор из-за пустяка также привел к большим проблемам.

Как же должен решать эту проблему сетевой протокол? Например, если SNMP-агент собирается послать ответное сообщение, содержащее целое значение счетчика полученных UDP-дейтаграмм, в каком формате следует ему представить это целое число? Один из вариантов заключается в том, чтобы пересылать байты в том же порядке, в котором они хранятся в управляющем объекте. Другой вариант состоит в том, чтобы пересылать байты без преобразования формата, то есть так,

как они хранятся в передающем компьютере. В любом случае может потребоваться преобразование формата данных либо перед их отправкой, либо после их получения. Кроме того, отправитель и получатель должны договориться о формате пересылки данных.

Третий вариант заключается в использовании независимого от машины, платформы и языка программирования метода описания целых чисел и данных других типов (иными словами, в использовании языка описания данных), а также правил пересылки по сети каждого из этих типов данных. Этот вариант применяется в языке SMI, который мы рассматривали в разделе «Архитектура управляющих Интернет-стандартов», и в языке ASN.1. В терминах ISO эти два стандарта описывают *службу представления,* то есть службу передачи и преобразования информации из одного машинного формата в другой. Пример проблемы представления информации, взятый из реального мира, показан на рис. 8.7. Ни один из получателей сообщения не понимает смысла сказанного. Как показано на рис. 8.8, эта проблема может быть решена при помощи службы представления.

Рис. 8.7. Проблема представления •

В табл. 8.5 перечислены некоторые типы данных языка ASN.1. Вспомним, что мы уже встречали типы **INTEGER, OCTET STRING** и **OBJECT IDENTIFIER** при обсуждении языка SMI. Те, кого интересуют подробности, могут ознакомиться со стандартами, в которых описываются типы языка ASN.1 и конструкторы, такие как **SEQUENCE** и **SET,** позволяющие определять структурные типы данных [291].

Помимо языка определения данных ASN.1 предоставляет *основные правила кодирования* (Basic Encoding Rules, BER), описывающие способ передачи по сети экземпляров объектов, определенных с помощью языка ASN.1. Правилами BER для кодирования передаваемых данных принят так называемый *метод TLV* (Type, Length, Value — тип, длина, значение). То есть для каждого элемента данных по сети пересылаются код его типа, длина и значение. Такой способ пересылки данных облегчает их интерпретацию при получении.

На рис. 8.9 показан простой пример пересылки двух элементов данных. В этом примере отправитель хочет переслать символьную строку «smith», за которой сле-

дует целое число 259 (в двоичном представлении это 00000001 00000011, то есть сначала идет байт со значением 1, а за ним байт со значением 3, так как используется прямой порядок следования байтов). Первый байт в передаваемом потоке имеет значение 4, указывающее на то, что следом идут данные типа OCTET STRING. Во втором байте потока передается длина строки байтов типа OCTET STRING, в данном случае это число 5. Третьим байтом потока начинается строка байтов типа OCTET STRING длины 5. В нем содержится ASCII-представление символа «s». Вслед за символьной строкой передается второй элемент данных, начинающийся с байта 2 (тип INTEGER), еще одного байта 2 (длина целого числа) и самого числа 259 с прямым порядком следования байтов.

Рис. 8.8. Решение проблемы представления

Таблица 8.5. Некоторые типы данных языка ASN.1

Тег	Тип	Описание
1	BOOLEAN	Значением «истина» или «ложь»
2	INTEGER	Может быть произвольно большим
3	BITSTRING	Список из одного или нескольких битов
4	OCTET STRING	Список из одного или нескольких байтов
5	NULL	Нет значения
6	OBJECT IDENTIFIER	Имя из стандартного дерева именования ASN.1 (см. подраздел «База управляющей информации» в разделе «Архитектура управляющих Интернет-стандартов»)
9	REAL	Число с плавающей точкой

В нашем обсуждении мы рассмотрели лишь небольшое и простое подмножество языка ASN.1. Дополнительные сведения о языке ASN.1 можно найти в [230, 231, 291,363].

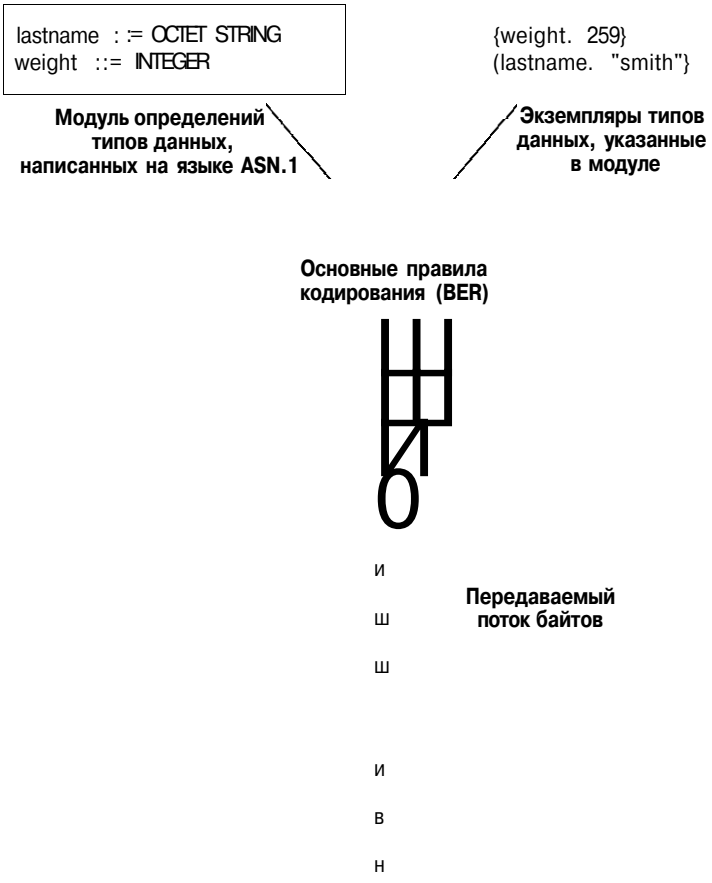


Рис. 8.9. Пример кодирования по правилам BER

Резюме

Итак, наше изучение сетевого управления (и компьютерных сетей в целом) закончено!

В этой последней главе, посвященной сетевому управлению, мы начали с рассказа об инструментальных средствах, необходимых сетевому администратору для мониторинга, тестирования, опроса, конфигурирования, анализа, оценки и контроля сети. Необходимость средств управления была проиллюстрирована на примерах управления такими системами, как электростанции, самолеты и коммерческие организации. Мы видели, что архитектура систем сетевого управления состоит из пяти ключевых компонентов: во-первых, сетевого администратора, во-вторых, множества удаленных (от сетевого администратора) устройств, в-третьих, базы управляющей информации (MIB) на этих устройствах, содержащей данные о состоянии и работе устройств, в-четвертых, удаленных агентов, сообщающих сетевому администратору M1B-инфор-

мацию и функционирующих под контролем сетевого администратора, и, в-пятых, протокола для связи между сетевым администратором и удаленными устройствами.

Затем мы углубились в детали архитектуры управляющей информации Интернета и, в частности, в детали протокола SNMP. Было показано, как в протоколе SNMP реализованы упомянутые пять ключевых компонентов сетевого управления. Мы изучили MIB-объекты, язык определения данных для описания МШ-объектов (язык SMI) и сам протокол SNMP. Было отмечено, что языки SMI и ASN.1 неразрывно связаны друг с другом, и язык ASN. 1 играет ключевую роль на уровне представления в семиуровневой модели ISO/OSI. Затем мы кратко рассмотрели язык ASN.1. Кроме того, мы отметили необходимость преобразования форматов данных при передаче их с одной машины на другую. Некоторые сетевые архитектуры явно поддерживают данную функцию на уровне представления, однако в стеке протоколов Интернета такой уровень отсутствует.

Также следует заметить, что мы намеренно не стали рассматривать многие вопросы сетевого управления, например обнаружение и исправление неисправностей, предотвращение неисправностей, управление службами. Эти темы также важны, но для их освещения потребовалась бы еще одна книга. Ссылки на нужную литературу и стандарты можно найти в разделе «Понятие сетевого администрирования».

Вопросы и задания для самостоятельной работы

1. Какая польза сетевому администратору от инструментальных средств сетевого управления? Опишите пять сценариев.
2. Какие пять областей сетевого управления определены ISO?
3. В чем различие между сетевым управлением и управлением службами?
4. Определите следующие понятия: управляющий объект, управляемое устройство, агент управления, MIB, протокол сетевого управления.
5. В чем состоит роль языка SMI в сетевом управлении?
6. В чем принципиальное различие сообщения «запрос-ответ» и прерывающего сообщения в протоколе SNMP?
7. Какие семь типов сообщений применяются в протоколе SNMP?
8. Что такое «ядро SNMP»?
9. Каково назначение дерева идентификации объектов ASN.1?
10. Какова роль языка ASN.1 в уровне представления эталонной модели ISO/OSI?
И. Есть ли в Интернете уровень представления? Если нет, то как решается проблема разницы компьютерных архитектур, например различное представление целых чисел на различных машинах?
12. Что такое TLV-кодирование?

Упражнения

1. Рассмотрим два способа связи между управляющим объектом и управляемым устройством: режим «запрос-ответ» и прерывания. Каковы плюсы и минусы

обоих методов в терминах, во-первых, накладных расходов, во-вторых, времени на уведомление в случае возникновения исключительной ситуации и, в-третьих, устойчивости к потерям сообщений, пересылаемых между управляющим объектом и устройством?

2. В разделе «Архитектура управляющих Интернет-стандартов» было показано, что для транспортировки SNMP-сообщений предпочтение отдается ненадежным UDP-дейтаграммам. Как вы полагаете, почему разработчики протокола SNMP выбрали для этой цели протокол UDP, а не TCP?
3. Как выглядит идентификатор объекта для протокола ICMP (см. рис. 8.3)?
4. Предположим, вы работаете в американской компании, которая хочет разработать свою базу MIB для управления производственной линией. В каком месте дерева идентификации объектов (см. рис. 8.3) будет зарегистрирована эта база? Для ответа на этот вопрос вам, вероятно, придется немного покопаться в RFC и другой документации.
5. Предположим, корпорация Microsoft разработала новый формат файлов для нового программного продукта. В каком месте дерева идентификации объектов он будет зарегистрирован?
6. Вернемся к рис. 8.9. Какова будет BER-кодировка элементов {вес, 271} {фамилия, "Джексон"}?
7. Вернемся рис. 8.9. Какова будет BER-кодировка элементов {вес, 296} {фамилия, "Браун"}?

Дополнительные вопросы и задания

1. Приведите пример (кроме электростанции и самолета) сложной распределенной системы, требующей управления.
2. Рассмотрим сценарий из раздела «Понятие сетевого администрирования». Что еще, как вы полагаете, желал бы отслеживать сетевой администратор? Аргументируйте свой ответ.
3. Прочитайте документ RFC 789. Как можно было бы избежать крушения сети ARPAnet (или упростить ее восстановление), случившегося в 1980 году, если бы у администраторов сети ARPAnet были сегодняшние инструментальные средства сетевого управления?

Интервью

Джефф Кейс является основателем и техническим директором корпорации SNMP Research, Inc. Эта корпорация — основной разработчик Интернет-стандартов и стандартных продуктов сетевого управления. Джефф Кейс получил две степени бакалавра наук (в области промышленного обучения и в области технологии электротехники), а также две степени магистра наук (в области промышленного обучения и в области электротехники) в Университете Пердью. Степень доктора философии в области технического обучения он получил в Университете Иллинойса, Урбана-Шампейн.

- Почему вы решили специализироваться в области компьютерных сетей?

  С тех пор как я начал ходить, мне всегда нравилось соединять различные объекты вместе, например втыкать шпильки для волос в электрические розетки. Со временем это хобби преобразовалось в интерес к аудиооборудованию — когда я стал чуть постарше, то занялся созданием звуковых систем для рок-групп. Это оборудование было достаточно мощным, чтобы крошить бетон. Во время учебы в колледже я занимался ремонтом теле- и радиоаппаратуры (по большей части аудиооборудования). В это время я заразился компьютерным вирусом и меня стало интересовать все цифровое, включая железо и программное обеспечение. Мне понравилось разрабатывать интерфейсы, соединяющие разные странные штуковины. Сначала я разрабатывал интерфейсы между процессорами и периферийными устройствами. Позднее я занялся интерфейсами между системами. Сеть — это предельный случай интерфейса. На сегодняшний день экстремальной сетью является Интернет.

  Какой была ваша первая работа в компьютерной промышленности?

  Большая часть моих первых лет профессиональной деятельности связана с университетом Пердью. Я преподавал почти на всех курсах по технологии электротехники и кибернетики. Я также принимал участие в разработке новых курсов по только что появившимся тогда темам микропроцессорного аппаратного и программного обеспечения. В одном из семестров наш класс занимался проектированием компьютера из микросхем, при этом часть курса проходила в лаборатории, где одна команда работала над созданием центрального процессора, другая создавала подсистему памяти, третья занималась разработкой подсистемы ввода-вывода и т. д. В следующем семестре мы писали программное обеспечение для построенной нами машины.

  В то же самое время я начал работать в качестве администратора университетской сети и закончил директором университетского компьютерного сервисного центра.

  Что самое сложное в вашей работе?

  Сложнее всего успевать следить за всеми изменениями как в технике, так и в бизнесе. Я сугубо технический менеджер, поэтому в последнее время становится все труднее соответствовать техническим новшествам в нашей промышленности. Моя должность также требует от меня следить за изменениями в области бизнеса, такими как слияния и покупка компаний.

- Каким вы видите будущее сетей и Интернета?

  Все больше, больше и больше. Более высокая скорость. Большая распространенность. Большие объемы данных. Большая напряженность между анархией и управляемостью. Больше спама. Больше борьбы со спамом. Больше проблем в сфере безопасности. Больше решений этих проблем. Наконец, следует ожидать чего-нибудь неожиданного.

- Кто помог вам достичь успеха в вашей профессиональной деятельности?

  Мой покойный отец, бывший весьма успешным бизнесменом; Дилберт; доктор Винт Серф, доктор Джон Постел; доктор Маршалл Роуз и Чак Дэйвин, хорошо известные фигуры в Интернет-промышленности; Билл Сиферт, в настоящее время сотрудник VC; доктор Руперт Эванс, руководитель моей диссертации; моя жена, работающая со мной в бизнесе; и, наконец, но не в последнюю очередь, Иисус.

- Я читал, что вы являетесь обладателем замечательной коллекции изречений. Когда вы преподавали кибернетику в университете, предлагали ли вы какие-либо изречения вашим студентам?

  Один пример стоит двух книг (кажется, из Гаусса).

  Иногда существует разрыв между теорией и практикой. В теории разрыв между теорией и практикой не так велик, как разрыв между теорией и практикой на практике (понятия не имею, кто это придумал).

- Что препятствует созданию Интернет-стандартов?

  Деньги. Политика. Эго. Ошибки руководства.

- Какое применение технологии SNMP стало наиболее неожиданным?

  Все. Мне пришлось всерьез заняться вопросами управления Интернетом и потребовались соответствующие инструментальные средства для управления сетевой инфраструктурой. Широкий успех, который получила работа многих людей, столкнувшихся со сходными проблемами, был вызван интуитивной прозорливостью, удачей и тяжелым трудом. Важно, что мы уже тогда получили требуемую архитектуру.

Ссылки

Здесь вы найдете ссылки на дополнительные источники информации. Ссылки на web-сайты приводятся там, где это возможно. Хотя все предоставленные ниже адреса в июне 2002 года были действительны (проверялись), они могли устареть. Обновленную версию библиографии можно найти на web-странице этой книги (http://www.awl.com/kurose-ross).

Копии документов RFC можно найти на разных web-сайтах. Все перечисленные ниже ссылки на документы RFC относятся к архиву института информатики ISI (Information Sciences Institute). Кроме того, документы RFC можно найти на web-сайтах http://www.faqs.org/rfcs, http://www.pasteur.fr/other/computer/RFC (во Франции) и http://www.csl.sony.co.jp/rfc/ (в Японии). Документы RFC могут обновляться или объявляться устаревшими последующими документами RFC, поэтому мы рекомендуем заглядывать на указанные выше сайты на предмет поиска самой свежей информации. В этом вам поможет специальная поисковая система на сайте http://www.rfc-editor.org/rfc.html.

1. 3Com Corporation, «Network Interface Cards», http://www.3com.com/products/nics.html.
2. A Distributed Testbed for National Information Provisioning, http://ircache.nlanr.net/.
3. Abitz P. and Liu C, DNS and BIND, O'Reilly & Associates, Petaluma, CA, 1993.
4. About i-mode, http://www.nttdocomo.co.jp/english/p_s/imode/index.htmL
5. Abramson N., «Development of the Alohanet», IEEE Transactions on Information Theory, Vol. IT-31, No. 3 (Mar. 1985), pp. 119-123.
6. Abramson N., «The Aloha System-Another Alternative for Computer Communications^ Proceedings of Fall Joint Computer Conference, AFIPS Conference, p. 37,1970.
7. Adler M., «Tradeoffs in Probabilistic Packet Marking for IP Traceback», Proceedings of 34th ACM Symposium on Theory of Computing (STOC), May 2002, http://www.cs.umass.edu/~micah/pubs/traceback.ps.
8. Ahn J. S., Danzig P. B., Liu Z., and Yan Y., «Experience with TCP Vegas: Emulation and Experiment^ Proceedings of ACM SIGCOMM '95 (Boston, MA, Aug. 1995), pp. 185-195, http://www.acm.org/sigcomm/sigcomm95/papers/ahn.htmL