Высшее профессиональное образование

Р. Л. Смелянский

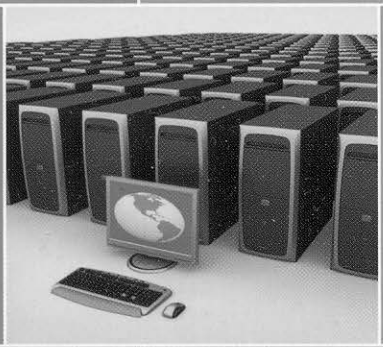
КОМПЬЮТЕРНЫЕ СЕТИ

В двух томах

Том 2

СЕТИ ЭВМ

Учебник



Информатика и вычислительная техника

ACADEMIA

УДК 004.7(075.8)
ББК 32.973.202я73
С501

Рецензенты:
зав. кафедрой вычислительной техники МЭИ (ТУ), д-р техн. наук, проф. *В. В. Топорков*; декан факультета вычислительной математики и кибернетики Казанского госуниверситета, зав. кафедрой системного анализа и информационных технологий, проф. *Р. Х. Латыпов*;
зав. кафедрой теоретической информатики Ярославского государственного университета им. П. Г. Демидова, д-р физ.-мат. наук, проф. *В. А. Соколов*

Смелянский Р. Л.

С501 Компьютерные сети : учебник для студ. высш. учеб. заведений : в 2 т. Т. 2. Сети ЭВМ / Р. Л. Смелянский. — М. : Издательский центр «Академия», 2011. — 240 с.

ISBN 978-5-7695-7153-4

Рассмотрены эталонная модель взаимодействия открытых систем и модель TCP/IP, протоколы IPv4 и IPv6, адресация в Интернете, протоколы ARP и RARP, алгоритмы маршрутизации, протоколы RIP, RIPv2, OSPF, BGP, маршрутизация в сетях MPLS, оптимизация и функционирование транспортного уровня, протоколы TCP и UDP, основные вопросы безопасности информации в сетях ЭВМ, а также такие приложения, как DNS, SNMP, электронная почта, протокол FTP, организация и основные протоколы World Wide Web, понятия web-сервиса и основные компоненты архитектуры SOA.

Для студентов учреждений высшего профессионального образования.

УДК 004.7(075.8)
ББК 32.973.202я73

ISBN 978-5-7695-7153-4 (Т. 2)
ISBN 978-5-7695-7152-7

ПРЕДИСЛОВИЕ

Данный учебник написан на основе материалов лекций по курсу «Компьютерные сети», который читается на факультете «Вычислительная математика и кибернетика» (ВМиК) Московского государственного университета им. М. В. Ломоносова.

Учебник, прежде всего, предназначен для подготовки специалистов и бакалавров по направлениям «Прикладная математика и информатика» и «Фундаментальная информатика и информационные технологии», а также он может быть использован для подготовки специалистов и бакалавров по направлению «Информатика и вычислительная техника».

Основной целью курса «Компьютерные сети» т. 2 «Сети ЭВМ» является приобретение студентами знаний и навыков в следующих областях:

- теоретические основы архитектурной и системотехнической организации вычислительных сетей, построение сетевых протоколов;
- основы Интернет-технологий;
- методы и средства обеспечения информационной безопасности компьютерных сетей;
- конфигурирование локальных сетей, реализация сетевых протоколов с помощью программных средств.

В учебнике не рассматриваются сетевые операционные системы, поскольку, во-первых, сегодня само понятие сетевой операционной системы выглядит странно, так как любая современная операционная система предполагает наличие функциональности для поддержки сетевого взаимодействия, во-вторых, на эту тему уже достаточно много издано, а в-третьих, на факультете ВМиК курсу «Компьютерные сети» предшествуют курсы «Операционные системы», «Базы данных» и »Программирование на языке Java».

На факультете ВМиК МГУ при изучении курса «Компьютерные сети» используется практикум, включающий в себя пять лабораторных заданий. Организация изучения курса на факультете ВМиК также предусматривает систему промежуточного контроля знаний студентов.

Для этой цели было разработано около 1 200 контрольных вопросов и задач. (Автор готов поделиться презентационными материалами, а также материалами для практических упражнений и промежуточного контроля с желающими, для чего достаточно обратиться по адресу smel@CS.MSU.SU.)

По окончании изучения второй части рассматриваемого курса студенты должны иметь представление о современных системах международной и отечественной стандартизации, стандартах построения компьютерных сетей, направлениях развития сетей должны знать:

• эталонную модель взаимодействия открытых систем, основы сетевых стеков OSI и TCP/IP, схему организации и основные принципы функционирования современных сетей, методы и алгоритмы работы сетевых протоколов для локальных, городских и региональных сетей;

• принципы организации и функционирования сетевого уровня, протоколы IPv4 и IPv6, адресацию в Интернете, алгоритмы ARP и RARP маршрутизации RIP, RIPv2, OSPF, иерархическую маршрутизацию и протокол BGP, маршрутизацию в сетях MPLS;

• методы оптимизации функционирования транспортного уровня и настройки протоколов TCP и UDP;

• основные вопросы безопасности информации в сетях ЭВМ и методы их решения (включая алгоритмы и методы шифрования данных в сетях), вопросы аутентификации в сетях (включая организацию электронно-цифровой подписи), системы обнаружения атак, технологию VPN;

• основы функционирования прикладных протоколов в Интернете (таких как DNS, SNMP), организацию и основные протоколы функционирования электронной почты, протокол FTP, организацию и основные протоколы функционирования World Wide Web, понятия web-сервиса и основные компоненты архитектуры SOA .

Изучение данного курса предполагает наличие определенного объема знаний, т.е. студент должен знать архитектуру современных вычислительных систем, организацию и функционирование операционной системы Unix и ее файловой системы, организацию и функционирование систем управления базами данных, элементы теории массового обслуживания (теории очередей), основы дискретной математики в части теории графов и сетей, а также основы теории автоматов.

Также студент должен иметь практический опыт программирования (желательно знать язык Java) и уметь строить математические модели с использованием указанных ранее разделов математики.

Несколько слов необходимо сказать об истории создания данного курса. В мае 1995 г. перед автором была поставлена задача создать за лето курс по компьютерным сетям. Ясно, что за такой срок написать курс невозможно, поэтому я посвятил эти два месяца изучению соответствующей литературы, которую в то время можно было найти на книжных полках. Надо сказать, что выбор был не так уж и велик. Воспользовавшись командировкой в США, я провел несколько дней в книжных магазинах Сан-Франциско, выбирая учебник, который можно было бы взять за основу курса. Просмотрев немало книг, я

остановил свой выбор на книге Тененбаума (Tanenbaum A. S. Computer Networks, 2-nd edition, Prentice Hall, 1988) по следующим соображениям:

- широта охвата материала;
- сбалансированность глубины и общности подачи материала;
- наличие ясной и доступной логики изложения;
- обилие ссылок на первоисточники;
- хороший язык изложения;
- возможность получения электронных версий рисунков, что было немаловажно при подготовке презентаций для лекций.

Однако темпы развития сетевых технологий и глубины их проникновения в общество столь велики, что материал курса постоянно приходилось дополнять, включать в него новые технологии, новые сетевые средства, протоколы. Каждый год курс дополнялся и изменялся. Изменялась при этом и организация изучаемого материала. Так, например, к концу 1990-х гг. стало ясно, что системы передачи данных представляют собой самостоятельную часть сетей, предназначенную для формирования канала для сетевого уровня. При этом в них и возникает задача маршрутизации, но уровень ее сложности существенно ниже, чем на сетевом уровне. Используемые в этих системах решения ближе к вопросам телефонии и коммутации, чем к сложным алгоритмам маршрутизации с многокритериальной оптимизацией, которые используются на сетевом уровне, и к сложным алгоритмам управления транспортными соединениями на транспортном уровне. Совершенствуя курс, автор также постарался учесть тенденцию разделения процессов построения и оптимизации маршрута и коммутации потоков данных на сетевом уровне, нашедшую отражение в концепции MPLS-сетей. Эта тенденция вызвана все возрастающими размерами компьютерной сети и ростом вычислительных затрат на процессы маршрутизации.

Наконец, осталось самое приятное — поблагодарить всех тех, кто помогал автору, критиковал материал, способствуя улучшению курса. Прежде всего, хотелось бы поблагодарить студентов третьего потока факультета ВМиК, которые слушали курс в 1995—2008 гг. и своими замечаниями, критикой способствовали его улучшению. Автор также очень признателен Д. Гамаюнову, помогавшему работать с рукописью и предоставившему часть материалов по вопросам сетевой безопасности, А. Петухову, предоставившему материалы по Web-технологиям, Д. Козлову и М. Забежайло — за обсуждение и конструктивные предложения отдельных частей курса, Н. Трусову и Е. Капинус — за работу по подготовке рукописи. Особенно хотел бы отметить доброжелательность Ю. В. Коровина, без поддержки которого эта книга вряд ли появилась бы.

Москва
август 2010

Р. Л. Смелянский

Глава 1
МОДЕЛЬ И ПРИМЕРЫ СЕТЕЙ ЭВМ

1.1. Модель сети ЭВМ

1.1.1. Общие сведения

Организация сети ЭВМ (см. т. 1 данного учебника) представляет собой совокупность системы передачи данных (СПД), транспортной среды и абонентских машин (рис. 1.1). Напомним, что *A*-машины — это абонентские машины, на которых работают сетевые приложения, а *R*-машины — это маршрутизаторы, отвечающие за доставку сообщений между *A*-машинами. Система передачи данных и *R*-машины образуют транспортную среду (ТС) сети ЭВМ. Шлюзы *G* — это устройства, обеспечивающие сопряжение двух разных транспортных сред, а *B* — устройство, обеспечивающее сопряжение разных СПД.

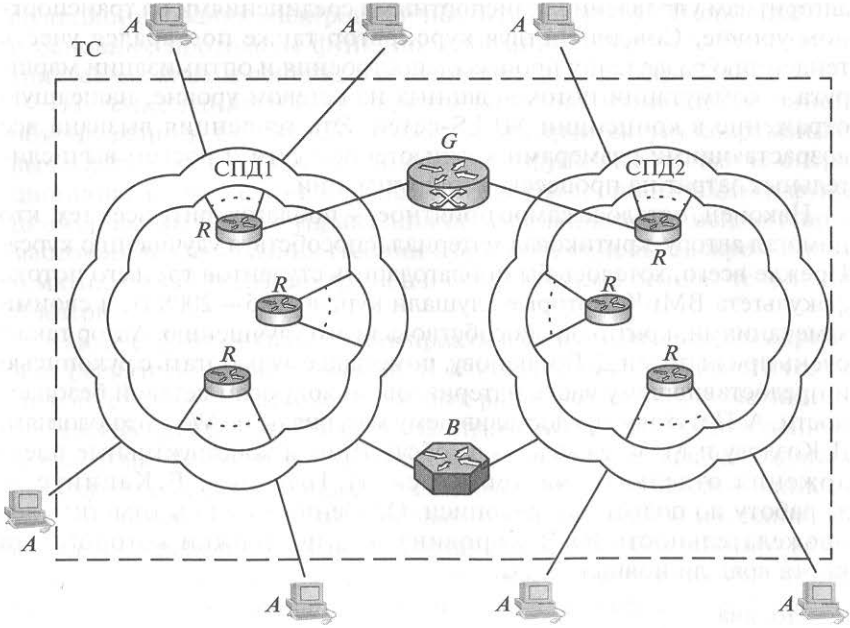


Рис. 1.1. Организация компьютерной сети

Система передачи данных — это совокупность каналов передачи данных и коммутирующих элементов.

В т. 1 данного учебника в качестве модели СПД были выбраны первые три уровня модели OSI с определенными оговоркам по функциям сетевого уровня, связанными с особенностями маршрутизации в СПД. В частности, обращалось внимание на то, что сетевой уровень в СПД сугубо внутренний. Сетевые уровни *A*-машин и *R*-машин с сетевым уровнем в СПД не взаимодействуют.

Теперь мы рассмотрим ряд существующих моделей и на их основе построим модель функционирования современной сети ЭВМ.

1.1.2. Эталонная модель OSI

Напомним, что модель взаимодействия открытых систем — OSI (Open Systems Interconnection), представленная на рис. 1.2, была разработана для определения международных стандартов компьютерных сетей. Эта модель описывает, как должна быть организована

Рис. 1.2. Модель взаимодействия открытых систем

система, открытая для взаимодействия с другими системами. Модель OSI имеет уровневую организацию, т.е. она включает в себя семь уровней: физический, канальный, сетевой, транспортный, сессии, представления и прикладной. Функции физического уровня и уровня канала данных подробно рассмотрены в т. 1 данного учебника, так же, как и функции сетевого уровня, применительно к системам СПД.

Ключевыми в этой модели являются понятия сервиса, интерфейса и протокола. Под сервисом понимают услуги, которые нижерасположенный уровень оказывает по запросам вышерасположенного. В этой модели нижерасположенный уровень свои услуги может предоставлять только вышерасположенному уровню. Интерфейс определяет формирование и передачу запроса на услугу.

Активные элементы уровня, т.е. элементы, которые могут сами совершать действия, в отличие от элементов, над которыми совершают действия, называются *активностями*. Активности могут быть программными и аппаратными. Активности одного и того же уровня на разных машинах называются *равнозначными*, или *одноименными*. Активности уровня $n + 1$ являются *пользователями сервиса*, создаваемого активностями уровня n, которые, в свою очередь, называются *поставщиками сервиса*. Сервис может быть разного качества. Например, с установлением соединения и без установления соединения, с подтверждением получения переданных данных и без подтверждения. Правила и соглашения по установлению соединения, его поддержанию и обмену данными по нему между активностями, расположенными на одинаковом уровне на разных машинах, называется *протоколом*.

Доступ к сервису в модели OSI осуществляется через так называемые *точки доступа к сервису* — SAP (Service Access Points), каждая из которых имеет уникальный адрес. Например, телефонная розетка на стене — это точка доступа к сервису автоматической телефонной станции (АТС). Каждой такой розетке сопоставлен определенный номер — номер телефона.

Взаимодействие между двумя соседними уровнями в этой модели можно описать следующим образом: активность на уровне $n + 1$ передает интерфейсную единицу данных — IDU (Interface Data Unit) активности на уровне n через SAP (рис. 1.3). IDU состоит из сервисной единицы данных — SDU (Service Data Unit) — и управляющей информации. SDU передается далее по сети равнозначной сущности, а затем — на уровень $n + 1$. Управляющая информация необходима нижерасположенному уровню, чтобы правильно передать SDU, но она не является частью передаваемых данных.

Чтобы передать SDU по сети нижерасположенному уровню, может потребоваться разбить ее на части (рис. 1.4). При этом каждая часть снабжается *заголовком* и *концевиком* и передается как самостоятельная единица данных протокола — PDU (Protocol Data Unit). Заголовок в PDU используется протоколом при передаче. В нем указывает-

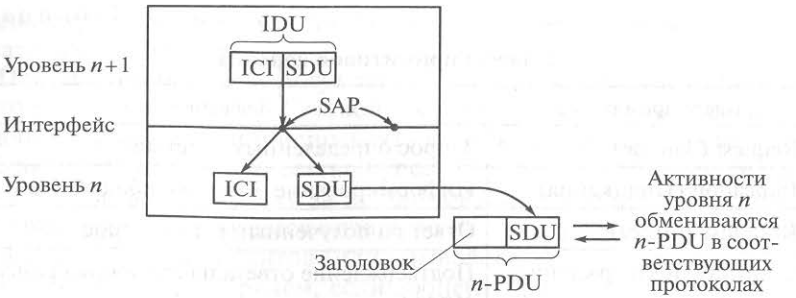


Рис. 1.3. Взаимосвязь уровней Сети через интерфейс:

SAP — точка доступа к сервису; IDU — интерфейсная единица данных; SDU — сервисная единица данных; PDU — единица данных протокола; ICI — контрольная информация интерфейса

ся, какой PDU содержит управляющую информацию, а какой — данные, порядковый номер PDU и т. д.

Формально сервис можно описать в терминах примитивных операций, или *примитивов*, с помощью которых пользователь или какая-либо активность получает доступ к сервису. С помощью этих примитивов активность вышерасположенного уровня сообщает активности нижерасположенного уровня, что необходимо сделать, чтобы вышерасположенная активность получила необходимую услугу (сервис). В свою очередь, нижерасположенная активность может использовать эти примитивы, чтобы сообщить вышерасположенной активности о выполненном действии.

Рис. 1.4. Пример организации потока информации при виртуальном взаимодействии на 5-м уровне Сети:

M — сообщения; H_i — заголовок i-го уровня; T_i — концевик i-го уровня

Таблица 1.1

Классы примитивов сервиса

Класс примитивов	Значение
Request (Запрос)	Запрос определенных действий
Indication (Индикация)	Информирование о каком-либо событии
Response (Ответ)	Ответ на полученный ранее запрос
Confirm (Подтверждение)	Подтверждение ответа на сделанный ранее запрос

Примитивы можно подразделить на четыре класса, показанные в табл. 1.1. Эти классы подробно рассмотрены в подразд. 2.3.4. т. 1 данного учебника.

Сетевой уровень

Основная проблема, решаемая на сетевом уровне, — это выбор оптимального маршрута для пакета от отправителя к получателю. Для решения этой проблемы требуется решить две задачи: вычислить оптимальный маршрут и реализовать этот маршрут, т. е. обеспечить следование пакета именно по рассчитанному маршруту. При этом маршрут может быть определен заранее и прописан в таблице в каждом узле коммутации, которая не изменяется. Маршрут может быть определен в момент установления соединения, и тогда все пакеты, направляемые по данному соединению, будут следовать этим маршрутом. Наконец, маршрут может строиться динамически по ходу передачи в зависимости от загрузки сети и ее текущей конфигурации. Критерии для определения оптимальности маршрута могут быть разными в зависимости от конкретной ситуации.

По понятным причинам в сети ЭВМ не может быть единого центра расчета маршрута, поэтому может оказаться, что в транспортной среде циркулирует слишком много пакетов, использующих одни и те же маршруты либо их фрагменты, что будет приводить к перегрузкам каналов СПД, а следовательно, и к потере пакетов. Эта проблема также решается на сетевом уровне. Причем эту проблему, называемую перегрузкой в сети, не следует путать с проблемой управления потоком, заключающейся в том, чтобы отправитель не «заваливал» пакетами получателя. Различие и методы решения этих проблем подробно рассмотрим далее.

Как уже говорилось, некоторые виды сетевого сервиса предполагают, что каждый пакет движется по своему отдельному маршруту. Это означает, что достигать сетевого уровня получателя они будут в разное время. При этом никто не гарантирует, что ранее отправленный пакет придет раньше отправленного позднее. Следовательно, на

сетевом уровне необходимо уметь восстанавливать исходную последовательность пакетов.

Пакеты при передаче могут теряться и дублироваться (причины этого рассмотрим далее). Сетевой уровень должен уметь бороться с подобными явлениями и исправлять возникающие нарушения.

Как видно из рис. 1.2, сетевой уровень является границей транспортной среды. Поскольку за использование транспортной среды, как правило, предполагается оплата, то на этом уровне присутствуют функции учета числа байтов (или символов), которые послал и получил абонент сети. Причем, если абоненты расположены в разных странах, где действуют различные тарифы, необходимо должным образом скорректировать цену услуги.

Если пакет адресован в другую сеть, т.е. в другую транспортную среду, то при его передаче требуется предпринять надлежащие меры для учета различия форматов пакетов в разных транспортных средах, различия способов адресации, допустимых размеров пакетов и т.д. Все эти проблемы решаются шлюзом (см. устройство *G* на рис. 1.1) на сетевом уровне.

Транспортный уровень

Основная функция транспортного уровня — принять данные с вышерасположенного уровня сессии, разделить их, если требуется, на более мелкие единицы — сегменты, передать на сетевой уровень и позаботиться, чтобы все они дошли в целости до адресата. Все это необходимо сделать эффективно и так, чтобы вышерасположенный уровень не зависел от того, как именно это было сделано. В нормальных условиях транспортный уровень должен создавать специальное сетевое соединение для каждого транспортного соединения по запросу уровня сессии. Если для транспортного соединения требуется высокая скорость передачи, то транспортный уровень может потребовать у сетевого уровня создания нескольких сетевых соединений, между которыми транспортный уровень будет распределять передаваемые данные. И наоборот, если требуется обеспечить недорогое транспортное соединение, то транспортный уровень может использовать одно и то же соединение на сетевом уровне для нескольких транспортных соединений. В любом случае такое мультиплексирование должно быть незаметным на уровне сессии.

Сетевой уровень определяет, какой тип сервиса предоставить вышерасположенным уровням и пользователям сети. Наиболее часто используемым сервисом является канал типа «точка—точка», без ошибок обеспечивающий доставку сообщений или байтов в той последовательности, в какой они были отправлены. Другим видом сервиса является доставка отдельных сообщений без гарантии сохранения их последовательности или, например, рассылка одного

сообщения многим в режиме вещания. В каждом конкретном случае сервис определяют при установке транспортного соединения.

Транспортным называется уровень, обеспечивающий соединение типа точка — точка. Активности транспортного уровня на машине отправителя общаются с равнозначными активностями транспортного уровня на машине получателя. При этом машина-отправитель и машина-получатель не обязаны быть соседними (см. рис. 1.2). Этого нельзя сказать про активности на нижерасположенных уровнях, которые общаются с равнозначными активностями на соседних машинах. В этом заключается одно из основных отличий уровней 1...3 от уровней 4...7. Последние обеспечивают соединение типа точка — точка, что хорошо видно из рис. 1.2.

Внимательно сравнив рис. 1.2 с рис. 1.2 из т. 1 данного учебника, увидим, что если *A*-машины с *R*-машинами взаимодействуют на физическом, канальном, сетевом и других уровнях, то с СПД они взаимодействуют только на физическом и канальном уровнях. Сетевой уровень СПД сугубо внутренний.

Многие *A*-машины мультипрограммные, поэтому транспортный уровень для одной такой машины должен поддерживать несколько транспортных соединений. Чтобы определить, к какому соединению относится тот или иной пакет, в его заголовке (см. H_4 на рис. 1.4) помещается необходимая информация.

Транспортный уровень также отвечает за установление и разрыв транспортного соединения в сети. Это предполагает наличие механизма именования, т. е. процесс на одной машине должен уметь указать, с кем в сети ему следует обменяться информацией. Транспортный уровень также должен предотвращать «захлебывание» получателя при обращении с «очень быстро говорящим» отправителем. Механизм такого предотвращения, называемый *управлением потоком*, имеется и на других уровнях. Однако, как мы увидим далее, управление потоком между *A*-машинами в сети отличается от управления потоком между *R*-машинами (маршрутизаторами).

Уровень сессии

В модели ISO уровень сессии позволяет пользователям *A*-машин (напомним, что пользователем может быть программа) устанавливать между собой *сессии*. Сессия позволяет передавать данные, как это может делать транспортный уровень, а кроме того, этот уровень имеет более сложный сервис, полезный в некоторых приложениях. Например, на уровне сессии можно осуществлять вход в удаленную систему, передавать файл между двумя приложениями и т. д.

Одним из видов услуг на этом уровне является *управление диалогом*. Потоки данных здесь могут быть разрешены одновременно в обоих направлениях либо поочередно в одном направлении. Сервис на уровне сессии будет управлять направлением передачи.

Другой вид сервиса на уровне сессии — *управление маркером*. Для некоторых протоколов недопустимо выполнение одной и той же операции на обоих концах соединения одновременно. В этом случае уровень сессии выделяет активной стороне маркер, и операцию может выполнять только тот, кто владеет маркером.

Также примером сервиса на уровне сессии является *синхронизация*. Пусть требуется передать файл, время пересылки которого составит два часа, между машинами, время наработки на отказ у которых — один час. Ясно, что «в лоб» передачу такого файла средствами транспортного уровня не решить. Уровень сессии позволяет расставлять контрольные точки. В случае отказа одной из машин передача возобновится с последней контрольной точки.

Уровень представления

В модели ISO уровень представления обеспечивает решение часто возникающих проблем, связанных с представлением данных при передаче. В основном это проблемы семантики и синтаксиса передаваемой информации. Данный уровень имеет дело с информацией, а не с потоком битов.

Типичным примером услуги на этом уровне является унифицированная кодировка данных. Дело в том, что на разных машинах используются разные способы кодировки символов, например ASCII, Unicode и другие, а также разные способы представления целых: в прямом, обратном или дополнительном коде. Нумерация битов в байте на разных машинах одной и той же сети тоже может быть разная, т.е. слева направо или справа налево, и т.д.

Пользователи, как правило, используют структуры данных, а не случайный набор байтов. Чтобы машины с разными кодировкой и представлением данных могли взаимодействовать, передаваемые структуры данных определяются специальным абстрактным способом, не зависящим от кодировки, используемой при передаче. Уровень представления работает со структурами данных в абстрактной форме, преобразует это представление во внутреннее представление для конкретной машины и из внутреннего машинного представления в стандартное представление для передачи по сети.

Уровень приложений

Уровень приложений обеспечивает работу часто используемых приложений, например передачу файлов.

Разные операционные системы используют разные механизмы именования, представления текстовых строк и т.д. Для передачи файлов между разными системами необходимо преодолевать подобные различия. Чтобы пользователям не приходилось каждый раз заново бороться с этими трудностями на уровне приложений имеется приложение FTP.

На уровне приложений находятся также такие часто используемые приложения, как электронная почта, удаленная загрузка программ, удаленный просмотр информации и т. д.

1.1.3. Модель TCP/IP

Рассмотрим другую эталонную модель, прототипом для которой послужил прародитель Интернета — сеть ARPA. Далее будет приведена история создания этой сети, а сейчас лишь отметим, что она была создана в результате, научно-исследовательских работ, проведенных по инициативе Министерства обороны США. Позднее к этому проекту подключились сотни университетов и государственных учреждений США.

С самого начала эта сеть задумывалась как объединение нескольких разных сетей.

Одной из основных целей этого проекта была разработка унифицированных способов соединения сетей для создания систем передачи данных, обладающих высокой живучестью. Под живучестью понимается способность системы сохранять в приемлемых пределах качество и объемы предоставляемого сервиса при выходе из строя ее компонентов. Так появилась модель TCP/IP, получившая название по именам двух основных протоколов: протокола управления передачей — TCP (Transmission Control Protocol) и межсетевого протокола — IP (Internet Protocol).

Другой целью проекта ARPA было создание протоколов, не зависящих от характеристик конкретных *А*-машин, маршрутизаторов, шлюзов и т. п. Кроме того, связь должна была поддерживаться, даже если отдельные компоненты сети стали выходить из строя во время соединения. Другими словами, связь в этой сети должна поддерживаться до тех пор, пока источник информации и получатель информации работоспособны. Архитектура сети ARPA не должна была ограничивать приложения, начиная от простой передачи файлов до передачи речи и изображения в реальном времени.

Межсетевой уровень

В силу приведенных требований выбор организации транспортной среды был очевиден: сеть с коммутацией пакетов с межсетевым уровнем без соединений. Такой уровень, называемый *межсетевым*, является основой всей архитектуры сети. Его назначение — обеспечить доставку пакетов, движущихся в сети независимо друг от друга, даже если получатель принадлежит другой сети. Причем пакеты могут поступать к получателю не в том порядке, в котором они были посланы. Упорядочить их в надлежащем порядке — задача вышерасположенного уровня.

Межсетевой уровень определяет межсетевой протокол IP и формат пакета. Обратим внимание, что ни протокол, ни формат пакета не являются официальными международными стандартами в отличие от протоколов эталонной модели OSI, в которой большинство протоколов имеют статус международных стандартов.

Итак, назначением межсетевого уровня в модели TCP/IP является доставка IP-пакета по назначению и по оптимальному маршруту. Критерии оптимизации маршрута могут быть самые разные: например, самый короткий маршрут или самый быстрый, или самый дешевый, или такой, который не проходит через определенные сети, расположенные на недружественных территориях, и т. д. Это как раз то, за что отвечает сетевой уровень в OSI-модели.

Под межсетевым уровнем в TCP/IP-модели «великая пустота». Эта модель никак не регламентирует организацию и функционирование СПД, равно как и связь с ней. Она также ничего не говорит о том, что в СПД происходит, кроме того, что хост-машина* (в данном случае *A*-машина или *R*-машина) должна быть связана с сетью через некоторый протокол. Никаких ограничений на этот протокол, а также рекомендаций в этой модели нет.

Транспортный уровень

Над межсетевым уровнем в модели TCP/IP расположен *транспортный уровень*. Как и в OSI-модели, задача этого уровня — обеспечить связь типа точка — точка между двумя равнозначными активностями. В рамках модели TCP/IP было разработано два транспортных протокола. Первый протокол — TCP (Transmission Control Protocol), т. е. надежный протокол с соединением, который получает поток байтов, фрагментирует его на отдельные сообщения и передает их на межсетевой уровень. На машине-получателе равнозначная активность TCP-протокола собирает эти сообщения в поток байтов. TCP-протокол также обеспечивает управление потоком.

Второй протокол — UDP (User Datagram Protocol), т. е. ненадежный протокол без соединения для тех приложений, которые используют свои механизмы фрагментации и управления потоком. Этот протокол часто используется для передачи коротких сообщений в клиент-серверных приложениях, а также там, где скорость передачи важнее ее аккуратности. Напомним, что ненадежным называется протокол без уведомления о получении кадра, фрейма, пакета, сообщения и т. п.

Уровень приложений

В модели TCP/IP нет уровней сессии и представления, поскольку необходимость в них была неочевидна для ее создателей. В настоящее

* Хостом обычно называют машину в сети, у которой есть IP-адрес

Рис. 1.5. Соответствие эталонных моделей OSI и TCP/IP

время разработчик сложного приложения берет реализацию функций этих уровней на себя.

Уровень приложений располагается сразу над транспортным. Этот уровень сначала включал в себя следующие приложения: передача файлов — FTP, электронная почта — SMTP. Позднее к ним добавились служба имен домена — DNS (Domain Name Service), отображающая логические имена *A*-машин на их сетевые адреса, протокол для работы с гипертекстовыми документами во Всемирной паутине — HTTP и некоторые другие.

На рис. 1.5 показано соответствие уровней двух рассмотренных эталонных моделей.

1.1.4. Выбор модели

Теперь необходимо выбрать модель, которую мы будем использовать для дальнейшего изучения сетей ЭВМ. Сразу оговоримся, что разнообразие моделей сетей не исчерпывается уже рассмотренными. Например, имеется модель SNA (System Network Architecture) от компании IBM и модель DNA (Digital Network Architecture) от компании DEC. Все указанные модели имеют много общего. Например, модель SNA была положена в основу модели OSI. Однако наиболее популярны модели OSI и TCP/IP.

Модели TCP/IP и OSI имеют много общего. Обе эти модели имеют уровневую организацию и поддерживают понятие стека протоколов. Назначение их уровней примерно одинаковое. Все уровни этих моделей от транспортного и ниже используют протоколы для поддержки взаимодействия типа точка—точка, не зависящего от организации СПД, а все уровни выше транспортного ориентированы на приложения.

Наибольшее значение модели OSI методологическое: в ней явно определены и четко выделены понятия сервиса, интерфейса, протокола, уровня. Это разделение строго проведено сверху донизу. Сервис определяет, что делает уровень, но ничего не говорит о том, как он это делает. Интерфейс уровня определяет для вышерасположенного уровня доступ к сервису. Протокол определяет реализацию сервиса.

Здесь можно провести аналогию с объектно-ориентированным программированием. У каждого объекта имеется набор методов — сервис, который определяет те операции, которые этот объект может выполнять. Иными словами, сервис — это семантика методов. Каждый метод имеет интерфейс: набор параметров, имя и т. п. Реализация методов скрыта в объекте (протоколе) и невидима пользователю.

В модели TCP/IP нет столь же четкого выделения этих понятий. В ней понятие протокола оторвано от остальных частей, и в ней нет единой, хорошо продуманной, концепции построения. Этот факт является следствием того, как создавались эти модели.

Модель TCP/IP создавалась post factum, т. е. после реализации основных протоколов, а модель OSI — до начала их реализации, поэтому понятие протокола в ней абсолютно не зависит от остальных частей модели. Например, изначально протоколы канального уровня в модели OSI создавались для соединений типа точка—точка. Позднее, когда появились каналы с множественным доступом, на этот уровень были добавлены соответствующие протоколы. Никаких других изменений не последовало.

Модель TCP/IP была создана, когда стек TCP/IP уже существовал, поэтому:

- в этой модели нет четкого разграничения понятий сервиса, интерфейса и протокола;
- эта модель годится только для описания стека TCP/IP;
- уровень хост—сеть в ней по существу уровнем не является, это больше интерфейс;
- в этой модели отсутствуют уровни, соответствующие СПД.

Модели OSI и TCP/IP имеют разное число уровней. Обе они имеют уровень приложений, транспортный уровень и сетевой уровень. Все остальные уровни в них разные. Модель OSI поддерживает на сетевом уровне и сервис с соединением, и сервис без соединения. На транспортном уровне этой модели поддерживается сервис только с соединением. В модели TCP/IP наоборот: сетевой уровень обеспечивает сервис без соединения, а транспортный — и с соединением, и без соединения.

По существу модель OSI доказала свою эффективность как методологический инструмент и стала популярной, чего нельзя сказать о ее протоколах, а с TCP/IP все наоборот: модели по существу нет, а протоколы получили широкое распространение.

В данном курсе мы будем использовать понятия и организацию модели OSI, а изучать протоколы сетевого, транспортного и при-

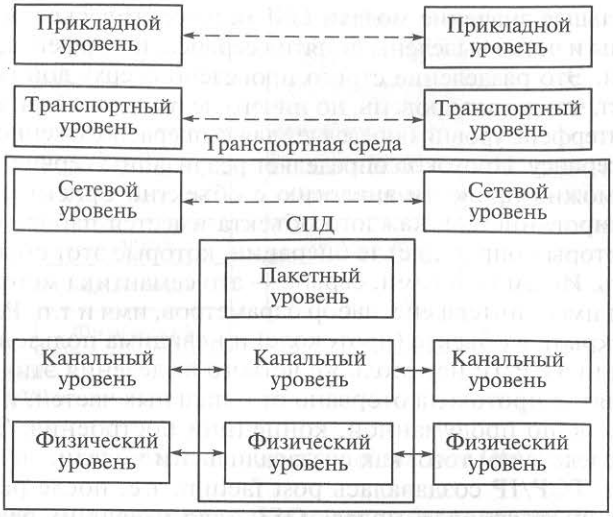


Рис. 1.6. Структура и организация гибридной модели, выбранной для дальнейшего изучения сетей ЭВМ

кладного уровней, следуя модели TCP/IP. Протоколы физического и канального уровней уже рассматривались в т. 1 данного учебника.

Гибридная модель, выбранная для дальнейшего изучения сети ЭВМ, представлена на рис. 1.6. Эта модель состоит из следующих пяти уровней: физического, канала данных, сетевого, транспортного и прикладного. При изучении будем использовать понятия уровня, сервиса, интерфейса, протокола, следуя модели OSI, откуда мы возьмем распределение сервисов и функций между уровнями, а набор протоколов и типовых приложений возьмем из модели TCP/IP.

1.2. Стандартизация в сети Интернет

В т. 1 данного учебника была подробно рассмотрена организация международной стандартизации. Поскольку одним из основных компонентов выбранной нами модели сети является TCP/IP, на которой основывается Интернет, то необходимо рассмотреть, кто есть кто в мире стандартизации Интернета.

Процесс стандартизации для сети Интернет с самого ее зарождения был хорошо организован. Этот процесс начался в 1969 г. с момента начала стандартизации протоколов сети ARPANET и всегда имел значительную финансовую поддержку как по линии государственного бюджета США, так и благодаря спонсированию этой деятельности промышленностью и бизнесом. В частности, такие компании как, например, APNIC, ARIN, Cisco Systems, IBM, Microsoft,

Ripe NCC ежегодно выделяют по 100 тыс. долл. для поддержки развития системы стандартов сети Интернет.

За процесс стандартизации информационных технологий в Интернете отвечают организации ISOC, IAB, IETF, IRTF, IESG [18], которые структурно взаимосвязаны следующим образом: на верхнем уровне иерархии в рассматриваемой организационной структуре располагается ISOC, а ниже расположены IAB, IETF, IRTF и IESG, отвечающие за отдельные направления работ.

ISOC (Internet Society — интернет-сообщество, www.isoc.org) — ассоциация экспертов, отвечающая за разработку стандартов технологий сети Интернет. В рассматриваемой организационной структуре ISOC располагается на верхнем уровне иерархии. Это некоммерческая неправительственная международная профессиональная организация, членами которой являются 175 организаций и около 9000 физических лиц из более чем 170 стран мира.

Работа ISOC сосредоточена на решении следующих основных задач:

- организация процесса стандартизации технологий сети Интернет;
- формирование внешней политики интернет-сообщества;
- поддержка инфраструктуры (организационно-административное управление деятельностью, управление финансами, защита прав интеллектуальной собственности и др.);
- образование и обучение, в том числе организация ежегодных семинаров по обучению интернет-технологиям (Network Training Workshops — NTW), организация системы учебных центров (Sustainable Internet Training Centers — SITCs) и др.;
- поддержка членства в ISOC как для организаций, так и для персональных членов.

IAB (Internet Architecture Board — Совет по архитектуре сети Интернет) — группа технических советников в составе ISOC, отвечающая за развитие архитектуры сети Интернет, управление разработкой и сопровождение стандартов протоколов и сервисов в Интернете и, прежде всего, спецификаций стека протоколов TCP/IP. Данный совет несет ответственность за управление редактированием и публикацией спецификаций RFC (Request for Comments), осуществляемое издательским органом RFC Editor (http://www.rfc-editor.org), а также за управление присваиванием номеров спецификациям RFC.

Результаты стандартизации в сети Интернет публикуются в виде RFC-документов. Эти документы являются доступными на файловых серверах Интернета для всех специалистов, что обеспечивает открытость процессу стандартизации. Каждой вновь разработанной спецификации или редакции уже существующего RFC-документа присваивается очередной свободный номер RFC-документа. Предыдущие версии пересмотренного документа остаются в каталогах системы стандартов с прежними номерами, но помечаются как изменявшиеся.

IAB выполняет представительские функции ISOC при взаимодействии с другими организациями. Деятельность IAB поддерживается

напрямую и косвенно как правительством США, так и промышленностью.

IETF (Internet Engineering Task Force — рабочая группа по проектированию Интернет-технологий, www.ietf.org) по существу является большим международным открытым сообществом разработчиков, операторов, изготовителей и исследователей в области сетевых технологий, занимающихся вопросами развития архитектуры сети Интернет и способов ее использования. IETF открыта для всех, кто интересуется интернет-технологиями. Основная сфера деятельности IETF состоит собственно в разработке стандартов сети Интернет, а также в их эффективной реализации и тестировании.

IRTF (Internet Research Task Force — исследовательская группа интернет-технологий, www.irtf.org) — подразделение IAB, которое выполняет долгосрочные исследовательские программы, связанные с вопросами развития архитектуры, базовых протоколов и сетевых приложений сети Интернет. Руководящие органы IRTF назначаются IAB.

IESG (Internet Engineering Steering Group — группа технического управления сети Интернет, www.ietf.org) отвечает за техническое управление процессом стандартизации интернет-технологий, осуществляет экспертизу проектов спецификаций, разрабатываемых IETF, несет ответственность за принятие интернет-стандартов и их дальнейшее продвижение.

Более подробно вопросы стандартизации в сети Интернет изложены в [17].

1.3. Примеры сетей ЭВМ

1.3.1. ARPANET

В начале 1960-х гг. Министерство обороны США поставило задачу создания командных пунктов и пунктов управления связью, которые были бы способны сохранить работоспособность при ядерной войне. Обычные телефонные линии были ненадежны: поражение АТС района означало потерю связи со всеми абонентами этого района. Для решения этой задачи Министерство обороны США обратилось к своему агентству перспективных разработок — ARPA (Advanced Research Project Agency, иногда DARPA), не имеющему лабораторий, научных сотрудников и т. п. Это бюрократическая организация с самостоятельным бюджетом, из которого она выделяет гранты университетам и компаниям, если их идеи кажутся им интересными.

В 1962 г. исследования ARPA по вопросам военного применения компьютерных технологий возглавил доктор Ликлайдер, который предложил для этих целей использовать взаимодействие имеющихся государственных компьютеров, а также способствовал привлечению

к этим работам частного сектора и университетских ученых. В этом же году появился отчет, выполненный П. Бараном в корпорации RAND по заказу военно-воздушных сил [70], в котором исследовались различные модели коммуникационных систем и оценивалась их живучесть. В отчете предлагалась децентрализованная система управления и связи, которая могла бы функционировать при выходе из строя какой-то ее части. Одна из рекомендаций автора касалась построения системы передачи цифровых данных между большим числом пользователей.

Вскоре основным направлением проводимых агентством исследований стали компьютерные сети. Главная идея состояла в построении сети из равноправных узлов, каждый из которых должен был иметь собственные блоки приема, обработки и формирования сообщений, обеспечивающие высокую живучесть сети даже при выходе из строя множества узлов. Первые эксперименты по объединению удаленных узлов были проведены уже в 1965 г., когда были соединены компьютеры TX-2 Массачусетского технологического института (MIT Lincoln Lab) и Q-32 корпорации SDC (System Development Corporation) в Санта-Монике. Правда, обмена пакетами между ними в это время еще не проводилось, обмен осуществлялся посимвольно.

В 1967 г. на симпозиуме ACM (Association for Computer Machinery) был представлен план создания национальной сети с коммутацией пакетов. Вскоре после симпозиума Робертс опубликовал план построения такой сети — ARPANET (Advanced Research Projects Agency NETwork), и уже в 1969 г. Министерство обороны США утвердило этот проект. Первым узлом новой сети стал университет штата Калифорния в Лос Анжелесе (UCLA) — Центр испытаний сети, а вскоре к нему присоединились Стэнфордский исследовательский институт (SRI), университет штата Калифорнии в Санта-Барбара (UCSB) и университет штата Юта. На узлах этой сети использовались IMP (Interface Message Processor) — машины, разработанные корпорацией Bolt Baranec & Newman, Inc (BBN). Так были осуществлены первые передачи знаков из одних машин в другие.

Согласно плану Робертса каждая подсеть должна была содержать как минимум один IMP-компьютер, соединенный линиями связи с двумя другими такими компьютерами в других подсетях. В подсети должен был использоваться дейтаграммный способ передачи данных. Программное обеспечение состояло из следующих протоколов: IMP — хост, IMP — IMP, IMP-отправитель — IMP-получатель (рис. 1.7).

Позднее для подключения терминалов к сети был создан вариант IMP, названный TIP (Terminal Interface Processor). Затем к одному IMP стали подключать несколько хост-машин, и одна хост-машина получила возможность соединяться с несколькими IMP. После этого начались эксперименты со спутниковой связью и радиосвязью. Например, был поставлен следующий эксперимент: грузовик с обо-

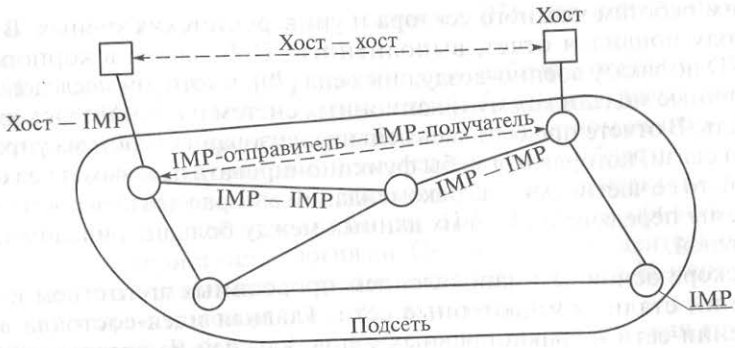


Рис. 1.7. Организация сети APRANET

рудованием двигался по дорогам южной Калифорнии, пытаясь связаться с машинами, находящимися в исследовательском институте Стэнфорда (SRI), которые через спутниковый канал передавали сообщения в Лондон.

В результате проведенных экспериментов стало ясно, что имеющиеся протоколы плохо пригодны для межсетевой связи, поэтому после выполнения дополнительных работ в 1974 г. появился стек протоколов TCP/IP. Контракты на реализацию этих протоколов ARPA передало в университет Беркли, где шли работы над операционной системой UNIX. Версия UNIX BSD 4.3 с реализацией TCP/IP, сокетов и другого сетевого программного обеспечения стала быстро распространяться.

К 1982 г. ARPANET уже насчитывала более 200 IMP-машин.

1.3.2. Хронология развития сети Интернет

1982 г. — рождение современного Интернета. TCP/IP стал официальным протоколом в ARPANET.

1984 г. — число хостов в ARPANET превысило тысячу.

1986 г. — Национальный фонд науки США (The National Science Foumdation) создал сеть NSFNET, открывшую доступ через Интернет к суперкомпьютерным центрам. После этого число сетей, подключенных к ARPANET, стало расти очень быстро. Во второй половине 1980-х гг. этот конгломерат сетей стали рассматривать как сеть сетей, а позднее как Интернет. В значительной степени рост сети Интернет происходил за счет подключения таких сетей, как SPAN — сеть космической физики NASA, HEP — сеть физики высоких энергий, BITNET — сеть машин класса mainframe фирмы IBM, EARN — европейская сеть научно-исследовательских организаций.

1989 г. — рождение идеи Всемирной паутины (World Wide Web — WWW), инициатор — британский ученый Тим Бернерс-Ли.

1990 г. — число сетей в Интернете достигло 3 000, а число машин в них — 200 000.

1991 г. — в Европейской физической лаборатории CERN создан известный всем протокол — HTTP (HiperText Transmission Protocol). Всемирная паутина стала доступна в Интернете. Эта разработка была сделана, прежде всего, для обмена информацией среди физиков. Появляются первые компьютерные вирусы, распространяемые через Интернет.

1992 г. — создание сообщества Internet (ISOC) с целью способствовать распространению Интернета и для управления этой сетью. Число хост-машин в Интернете достигло миллиона.

1993 г. — создание первого интернет-браузера Mosaic М. Андреесеном в Университете штата Иллинойс, обеспечивающего удобство просмотра документов, описанных на языке HTML (HiperText Markup Language). Число интернет-хостов превысило 2 млн, в сети действует уже 600 сайтов.

1995 г. — появление Netscape. Началось битва между браузерами Netscape, созданными под руководством М. Андреесона, и Internet Explorer, разработанным компанией Microsoft. В мире существует 12,8 млн хостов и 500 тыс. сайтов. Всемирная паутина стала ассоциироваться с понятием Интернет. Число пользователей Интернета достигло 20 млн человек.

1999 г. — впервые предпринята попытка цензуры Интернета (сделана попытка нарушить основополагающий принцип: Интернет никому не принадлежит). В ряде стран (Китае, Саудовской Аравии, Иране, Египте, странах бывшего СССР) государственными органами предприняты серьезные усилия, чтобы технически блокировать доступ пользователей к определенным серверам и сайтам политического, религиозного и порнографического характера.

2002 г. — сеть Интернет связывает 689 млн человек и 300 млн хостов. Разрабатываются новые технологии Интернета, которые должны заменить «старый» Интернет, расширить его функции или создать национальные компьютерные сети.

2005 г. — появление термина Веб 2.0 [13], что ознаменовало новый этап в развитии Интернета: из информационного пространства эта сеть сетей превратилась в пространство для сотрудничества (о чем уже немного говорилось во введении).

1.3.3. Другие примеры сетей ЭВМ

Ошибочно было бы полагать, что в мире не было других успешных реализаций стеков протоколов, отличных от TCP/IP. Так, например, фирмой Novell в 1980-е гг. был разработан стек протоколов IPX/SPX [15], который работает под операционной системой Netware и активно используется и по сей день для создания локальных офисных сетей. Одним из основных требований при разработке этого стека было

обеспечение способности его работы на маломощных персональных компьютерах (ПК) с минимальными ресурсами. Этот стек протоколов был очень популярен до середины 1990-х гг.

Другим примером стека сетевых протоколов является NetBIOS (Net Basic Input Output system). Это сетевое расширение стандарта операций ввода-вывода фирмы IBM для компьютеров IBM PC [66], включающего в себя интерфейс сеансового уровня (NetBIOS interface) и протокол транспортного уровня модели OSI. Интерфейс NetBIOS представляет собой стандартный интерфейс разработки приложений (API) для обеспечения сетевых операций ввода-вывода и управления нижерасположенным транспортным протоколом. Приложения, использующие NetBIOS API-интерфейс, могут работать только при наличии протокола, допускающего использование такого интерфейса. Также NetBIOS определяет протокол, функционирующий на сеансовом и транспортном уровнях модели OSI.

Хорошим примером является сетевой стек протоколов SNA (System Network Architecture) фирмы IBM. Эта разработка фирмы IBM была положена в основу модели OSI. Похоже, что ISO (Международный институт по стандартизации) собирался создать модель OSI, аналогичную разработке IBM, и сделать ее международным стандартом, но под своим управлением. В то время IBM настолько доминировала в компьютерном мире, что никому и в голову не приходило, что кто-то может противиться ее разработке. В 1994 г. автору этих строк довелось принимать участие в одной из первых электронных конференций в Интернете, где был задан интересный вопрос: что такое ISO? Ответ был весьма оригинальный (воспроизвожу его по памяти): «Это специальная форма английского языка, непригодная для понимания людьми». К сожалению, следует признать, что одна из причин, по которой эталонная модель ISO не была реализована и внедрена в практику, состояла в том, что она была описана совершенно ужасным языком, действительно очень трудным для понимания.

1.4. Требования, предъявляемые к современным компьютерным сетям

Главным требованием, предъявляемым к современным компьютерным сетям, является обеспечение пользователям доступа к вычислительным сервисам. Все остальные требования (производительность, надежность, безопасность, расширяемость и масштабируемость, прозрачность, управляемость, совместимость) характеризуют качество реализации этих сервисов.

Производительность характеризует скорость работы сети. Эта характеристика определяется числом услуг, предоставляемых сетью в единицу времени. Под услугой может пониматься пропускная способность — число пакетов, пройденных через сеть за секунду,

минуту, час, день, причем различают среднюю, мгновенную, пиковую, минимальную пропускную способность сети. Это также может быть время выполнения определенной операции, т. е. время реакции. Чаще всего пользователь обращает внимание именно на этот индекс производительности, поскольку он характеризует как скорость работы клиента, так и скорость работы сервера и СПД. Индекс, характеризующий только работу СПД, называется *временем передачи* — это время от поступления пакета данных на вход СПД до появления его на выходе (см. гл. 3).

Надежность характеризует способность сети выполнять операции, и если операция запущена, то всегда ли она корректно завершится. Имеется несколько подходов к определению этой характеристики:

- через измерение надежности устройств (времени наработки на отказ, вероятности отказа, интенсивности отказов);
- коэффициента готовности — доли времени, в течение которого система может быть использована;
- вероятности доставки пакета через транспортное соединение;
- вероятности искажения пакета в транспортном соединении;
- отказоустойчивости — способности обнаруживать ошибки функционирования и устранять их.

Безопасность характеризует защищенность информации и ресурсов сети от несанкционированного использования и изменения:

- транспортной среды;
- СПД;
- вычислительных ресурсов;
- данных, программ (доступа, изменения).

Конфиденциальность данных характеризует возможность обеспечения доступа к данным лишь тем, кто имеет на это право.

Целостность данных определяет возможность изменения данных только теми, кто имеет на это право.

Расширяемость характеризует сложность изменения конфигурации сети. Например, если для подключения новой машины в сеть необходимо остановить работу СПД, то такую сеть вряд ли можно назвать легкорасширяемой.

Масштабируемость характеризует способность сети плавно увеличивать вычислительную мощность без деградации ее производительности в целом.

Прозрачность характеризует, насколько «просто» пользоваться сетью. Чем сложнее доступ пользователю к требуемому сервису в сети, тем менее она прозрачна. В идеале, должен реализовываться принцип «сеть — это компьютер»:

- сеть сама распределяет ресурсы и управляет ими;
- обеспечивает среду для разработки и выполнения программ;
- является поставщиком разнообразного сервиса;
- для пользователя она прозрачна (он ее не видит), он лишь сообщает что ему требуется, а как и где это взять решает сеть.

Важной характеристикой сети является способность по одним и тем же СПД, т.е. по так называемым унифицированным системам передачи, передавать разнородные потоки данных. Одной из основных сложностей при этом является синхронность передачи разных потоков, например видео- и аудиопотоков, относящихся к одному и тому же фильму или явлению.

Управляемость характеризует возможность управлять и контролировать работу каждого отдельного устройства в сети из единого центра.

Совместимость характеризует способность подключать оборудование и программное обеспечение разных производителей.

Глава 2
СЕТЕВОЙ УРОВЕНЬ

2.1. Проблемы построения сетевого уровня

2.1.1. Общие сведения

Согласно модели, приведенной на рис. 1.1, маршрутизаторы и шлюзы рассчитывают маршрут и реализуют его. Задача маршрутизаторов — определить маршрут для каждой протокольной единицы данных (PDU) сетевого уровня и направить PDU по нужному каналу. Эта задача включает в себя собственно вычисление маршрута и его оптимизацию и коммутацию, т.е. определение канала, по которому необходимо передать PDU сетевого уровня. Задачей шлюзов является соединение между собой нескольких транспортных сред с разной архитектурой. При этом кроме уже указанных задач возникает задача обеспечения соответствия форматов PDU, используемых соединяемыми транспортными средами.

Сервис сетевого уровня, реализующий программное обеспечение маршрутизатора, должен удовлетворять следующим требованиям:

- быть независимым от технологии передачи, используемой в системе передачи данных, топологии и каких-либо других параметров СПД;
- транспортный уровень не должен зависеть от числа узлов и топологии транспортной среды.

Разработчик сетевого уровня свободен в выборе сервиса этого уровня. Однако цена этой свободы — необходимость решения непростых вопросов: например, должен сетевой уровень быть ориентированным на соединения или нет?

2.1.2. Ориентация на соединение

Спор между сторонниками сервиса с соединениями и сервиса без соединений — это по существу спор о том, где поместить основную вычислительную сложность [19]. Сервис, ориентированный на соединение, предполагает, что эта сложность должна приходиться на сетевой уровень, т.е. на маршрутизаторы и СПД, а сервис без соединений — на транспортный уровень, т.е. на абонентскую машину или хост. Защитники сервиса без соединения говорят, что стоимость вычислительных средств падает, а их мощность растет, так что нет причин не нагружать

хост. В то же время транспортная среда — это всеобщая инвестиция, и часто модернизировать ее вряд ли возможно, следовательно, она должна оставаться неизменной как можно дольше. Кроме того, для многих приложений скорость доставки важнее, чем ее аккуратность.

Сторонники сервиса, ориентированного на соединение, считают, что большинство пользователей не хотят гонять сложные транспортные протоколы на своих машинах. Им требуется надежный сервис, который могут предоставить соединения на сетевом уровне. Более того, многие приложения, например передачу звука и изображения в реальном масштабе времени, легче связывать с соединениями на сетевом уровне, чем с сетевым уровнем без соединений.

Так каким же должен быть сетевой уровень: надежным ориентированным на соединения или ненадежным без соединений? Похоже, что в этом споре побеждают сторонники надежного сервиса. Действительно, современные СПД на основе стандартов X.25, Frame Relay, ATM поддерживают надежный сервис на основе соединений. Другим аргументом в этом споре стали MPLS-сети (будут рассмотрены далее), которые разрабатываются и позиционируются как способ построения высокоскоростных IP-магистралей, реализуемых через такие СПД, как ATM.

2.1.3. Внутренняя организация сетевого уровня

Внутренняя организация сетевого уровня может быть как ориентированной на соединения, так и не ориентированной на соединение. В первом случае соединение называется виртуальным каналом (по аналогии с СПД), а во втором — о пакетах говорят как о дейтаграммах.

Идея виртуального канала — избежать маршрутизации для каждого пакета. Маршрут выбирается один раз при установлении виртуального канала между отправителем и получателем и в дальнейшем не изменяется до тех пор, пока передача не закончится. Транспортная среда запоминает выбранный маршрут. После окончания передачи, когда соединение разрывается, виртуальный канал уничтожается.

При выборе сервиса без соединения каждый пакет маршрутизируется независимо. Разные пакеты могут следовать разными маршрутами. Вследствие такой организации транспортная среда более надежна, способна гибко реагировать на ошибки и перегрузки. Однако для продвижения по разным маршрутам может потребоваться различное время, что может приводить к возникновению определенных проблем. (Позже мы вернемся к обсуждению всех pro и contra этих двух вариантов организации сетевого уровня.)

Каждый маршрутизатор в транспортной среде, ориентированной на виртуальные каналы, должен помнить, какие именно виртуальные каналы проходят через него. При этом каждый маршрутизатор должен иметь таблицу виртуальных каналов, а каждый пакет должен иметь

дополнительное поле для хранения номера виртуального канала. Когда пакет приходит к маршрутизатору, то, зная линию, по которой он пришел, и номер виртуального канала, указанный в пакете, маршрутизатор по таблице устанавливает, по какой линии следует отправить пакет далее.

При установлении соединения номер виртуального канала выбирается из числа неиспользуемых в данный момент на данной машине. Причем, так как каждая машина выбирает номер канала независимо, этот номер имеет лишь локальное значение. Заметим, что каждый процесс должен указать ожидаемое время освобождения виртуального канала, в противном случае могут возникнуть проблемы с принятием решения при его освобождении, например если одна из машин на маршруте «зависла».

Итак, при использовании виртуальных каналов их поддержка полностью обеспечивается транспортной средой. В случае использования дейтаграмм никакой таблицы виртуальных каналов в каждом маршрутизаторе иметь не требуется, поскольку в этом случае у них есть таблица, в которой указано, какую линию надо использовать, чтобы доставить пакет по заданному адресу. Такая таблица необходима и при использовании виртуальных каналов, когда устанавливается соединение.

Каждая дейтаграмма должна иметь полный адрес доставки. В больших сетях этот адрес может быть достаточно длинным (десятки байтов). При поступлении пакета маршрутизатор по таблице и адресу определяет, по какой линии надо отправить эту дейтаграмму, и посылает ее туда.

2.1.4. Сравнение транспортных сред с виртуальными каналами и с дейтаграммами

Выбор внутренней организации транспортной среды требует определенного компромисса. Так, например, использование виртуального канала избавляет от необходимости прописывать в каждом пакете длинный адрес доставки. Однако это предполагает затраты памяти маршрутизатора на хранение таблиц виртуальных каналов, т. е. явно необходим компромисс между пропускной способностью и памятью маршрутизатора.

Также необходим компромисс между временем установки соединения и временем разбора адреса доставки. В транспортных средах с виртуальными каналами требуется время на установление соединения. Однако при использовании однажды установленного виртуального канала дополнительных усилий для направления по нему пакета не требуется. При использовании сервиса без соединения для каждого пакета необходимо выполнять достаточно сложную процедуру маршрутизации, чтобы определить, куда его посылать.

Виртуальные каналы имеют известные преимущества в борьбе с перегрузками, т. е. когда ресурсов СПД и маршрутизаторов не хватает, чтобы «переварить» потоки данных. При использовании виртуального канала можно заранее зарезервировать необходимые ресурсы, а значит, начав передачу пакетов, можно быть уверенным, что имеются необходимые пропускная способность и ресурсы маршрутизатора. Борьба с перегрузками при использовании дейтаграмм намного сложнее.

Для систем обработки транзакций (кредитных карт, всякого рода покупок в системах электронной торговли) установление каждый раз виртуального канала было бы расточительством, т. е. в этих случаях постоянные виртуальные соединения, устанавливаемые вручную на недели или месяцы, вполне оправданны.

Виртуальные каналы слабоустойчивы к сбоям. Если в процессе работы транспортной среды на какое-то время выходят из строя маршрутизаторы, то все виртуальные каналы, проходящие через них, разрушаются. При использовании дейтаграмм самое худшее, что может произойти при временном отказе маршрутизатора, — это пропадут пакеты, находящиеся в это время в памяти маршрутизаторов.

Следует подчеркнуть, что тип сервиса (с соединением или без соединения) и внутренняя организация транспортной среды (использование виртуальных каналов или дейтаграмм) — вопросы независимые. Теоретически возможны все четыре комбинации. Возможен также сервис с соединением в дейтаграммных транспортных средах, если требуется обеспечить очень надежный сервис (например, TCP над IP) и сервис без соединения над виртуальными каналами (IP над ATM).

2.2. Алгоритмы маршрутизации

2.2.1. Общие сведения

Основной задачей сетевого уровня является маршрутизация пакетов. Пакеты маршрутизируются всегда, т. е. независимо от того, какую внутреннюю организацию имеет транспортная среда: с виртуальными каналами или дейтаграммную. Разница состоит лишь в том, что в первом случае этот маршрут устанавливается один раз для всех пакетов, а во втором — для каждого пакета отдельно. Первый случай называют иногда *маршрутизацией сессии*, поскольку маршрут устанавливается на все время передачи данных пользователя, т. е. на время сессии.

Алгоритм маршрутизации реализует программное обеспечение маршрутизатора на сетевом уровне, т. е. он отвечает за определение, по какой из линий, доступных маршрутизатору, отправлять пакет дальше. При этом независимо от выбора маршрута (для сессии или

для каждого пакета в отдельности) алгоритм маршрутизации должен обладать следующими свойствами: корректностью, простотой, устойчивостью, стабильностью, справедливостью и оптимальностью.

Корректность — свойство алгоритма маршрутизации, определяющее, что при любых обстоятельствах этот алгоритм либо найдет маршрут для доставки пакета адресату, либо выдаст сообщение о невозможности его доставки. Третьего варианта быть не может. При этом крайне желательно, чтобы алгоритм также сообщил о причинах невозможности доставки пакета.

Простота — свойство, определяющее вычислительную сложность алгоритма маршрутизации: чем она меньше, тем алгоритм проще, и тем меньше ресурсов маршрутизатора тратится на решение задачи маршрутизации.

Устойчивость — свойство алгоритма маршрутизации сохранять работоспособность независимо от каких-либо сбоев, отказов в системе передачи данных или транспортной среде, а также изменений топологии (отключения хостов или машин транспортной среды, разрушения каналов и т. п.). Алгоритм маршрутизации должен адаптироваться ко всем таким изменениям, не требуя при этом перезагрузки транспортной среды или остановки абонентских машин.

Стабильность — весьма важное свойство алгоритма маршрутизации. Существуют алгоритмы, которые никогда не приводят к какому-либо определенному маршруту, как бы долго они ни работали. Это означает, что адаптация алгоритма к изменениям в топологии или конфигурации транспортной среды может оказаться весьма продолжительной, и более того, она может оказаться сколь угодно долгой.

Справедливость — свойство, означающее, что все пакеты независимо от того, из какого канала они поступили, будут обслуживаться маршрутизатором равномерно, т. е. никакому направлению не будет отдаваться предпочтение, для всех абонентов будет всегда выбираться оптимальный маршрут.

Следует отметить, что справедливость и оптимальность часто могут вступать в противоречие друг с другом при неудачном выборе критерия оптимизации. На рис. 2.1 приведен пример такого противоречия. Например если в качестве критерия оптимизации выбрать расстояние, а трафики между A и A', B и B', C и C' уже заполнили канал между X и X', то вместо наикратчайшего маршрута между X и X' потребуется выбирать какой-то другой маршрут, который не будет оптимальным по критерию наикратчайшего расстояния. При этом ситуация коренным образом изменится, если выбрать в качестве критерия оптимизации время задержки при доставке.

Прежде чем искать компромисс между оптимальностью и справедливостью, необходимо решить, что является *критерием оптимизации маршрута*. Один из возможных критериев — средняя задержка пакета (обратите внимание, что именно средняя задержка). Другой

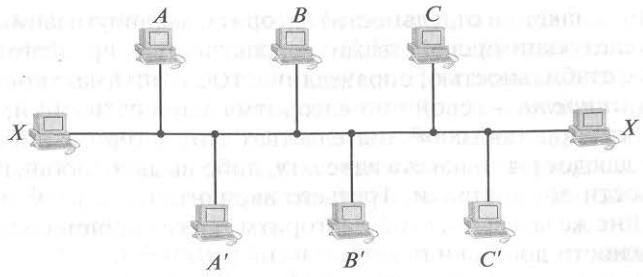


Рис. 2.1. Пример противоречия между справедливостью и оптимальностью

критерий — пропускная способность транспортной среды. Однако эти критерии конфликтуют. Согласно теории массового обслуживания, если система с очередями функционирует близко к своему насыщению, то задержка в очереди увеличивается. Как компромисс во многих сетях минимизируется число переходов между маршрутизаторами. Один такой переход называется *скачком*, или *переходом* (hop). Уменьшение числа скачков сокращает маршрут, а следовательно, сокращает задержку и минимизирует необходимую пропускную способность СПД для передачи пакета.

Алгоритмы маршрутизации можно разбить на два больших класса: *адаптивные* и *неадаптивные*. Неадаптивные алгоритмы не принимают в расчет текущую загрузку сети и ее текущую топологию. Все возможные маршруты вычисляются заранее и загружаются в маршрутизаторы при загрузке сети. Такая маршрутизация называется *статической*.

Адаптивные алгоритмы, наоборот, определяют маршрут исходя из текущей загрузки и топологии транспортной среды. Адаптивные алгоритмы различаются способом получения информации (локально от соседних маршрутизаторов или глобально от всех маршрутизаторов), временем изменения маршрута (через каждые T секунд либо только когда изменяется нагрузка, либо когда изменяется топология) и метрикой, используемой при оптимизации (расстояние, число скачков, ожидаемое время передачи и т. п.).

2.2.2. Свойство оптимального пути

Обоснуем одно важное предположение о свойстве оптимального маршрута, которое будет использоваться в дальнейших рассуждениях [19]. Это свойство состоит в том, что если маршрутизатор J находится на оптимальном пути между маршрутизаторами I и K (рис. 2.2, *а*), то оптимальный маршрут между J и K принадлежит этому оптимальному пути. Это так, поскольку существование между J и K оптимального маршрута, отличного от части маршрута между I и K,

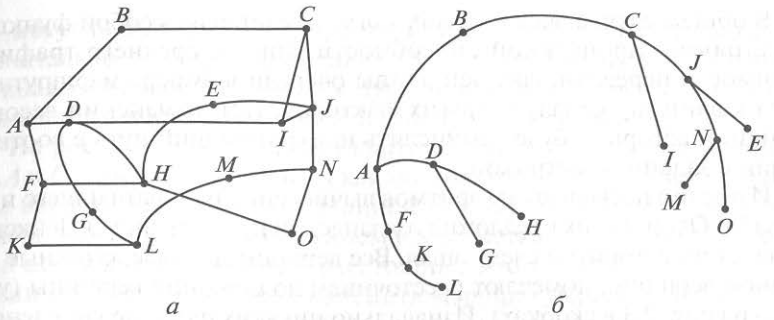


Рис. 2.2. Топология транспортной среды (*а*) и дерево захода (*б*)

противоречило бы утверждению об оптимальности маршрута между I и K. Дело в том, что если представить маршрут от I до K, как от I до J (назовем его S_1) и от J до K (назовем его S_2) и если между J и K имеется маршрут лучше, чем S_2, например S_3, то маршрут S_1S_2 не может быть лучшим. Взяв конкатенацию маршрутов S_1S_3, мы получим лучший маршрут, чем маршрут S_1S_2. Следствием из этого свойства является утверждение, что все маршруты к заданной точке транспортной среды образуют дерево с корнем в этой точке. Это дерево, называемое *деревом захода*, показано на рис. 2.2, *б*.

Поскольку дерево захода — это дерево, то там нет циклов, и каждый пакет будет доставлен за конечное число шагов. На практике же все может оказаться сложнее: маршрутизаторы могут выходить из строя, могут появляться новые маршрутизаторы, каналы могут выходить из строя, разные маршрутизаторы могут узнавать об этих изменениях в разное время и т. д.

2.2.3. Маршрутизация по наикратчайшему пути

Изучение алгоритмов маршрутизации начнем со статического алгоритма, широко используемого на практике вследствие его простоты. Идея этого алгоритма состоит в построении графа транспортной среды (где вершины — маршрутизаторы, а дуги — каналы связи) и нахождении наикратчайшего пути в этом графе.

На рис. 2.3 [19] показана работа алгоритма нахождения наикратчайшего пути от A к D (стрелками обозначены задействованные узлы). На дугах этого графа цифрами указаны веса, которые представляют собой расстояния между узлами. Это расстояние можно измерять в переходах, а можно в километрах. Возможны также и другие меры: например, средняя задержка пакетов в соответствующем канале. В графе с такой разметкой наикратчайший путь — это путь с наименьшим весом, хотя ему необязательно способствует минимальное число переходов или километров.

В общем случае веса на дугах могут представлять собой функции расстояния, пропускной способности канала, среднего трафика, стоимости передачи, средней длины очереди в буфере маршрутизатора к данному каналу и других факторов. При изменении весовой функции алгоритм будет вычислять наикратчайший путь в соответствии с заданной метрикой.

Известно несколько алгоритмов вычисления наикратчайшего пути в графе. Один из них предложил голландский программист Э. Дейкстра. Идея этого алгоритма следующая. Все вершины в графе, смежные исходной вершине, помечают расстоянием до исходной вершины (указано на рис. 2.3 в скобках). Изначально никаких путей не выделено, и все вершины помечены бесконечностью. По мере работы алгоритма и нахождения путей метки вершины могут изменяться. При этом мет-

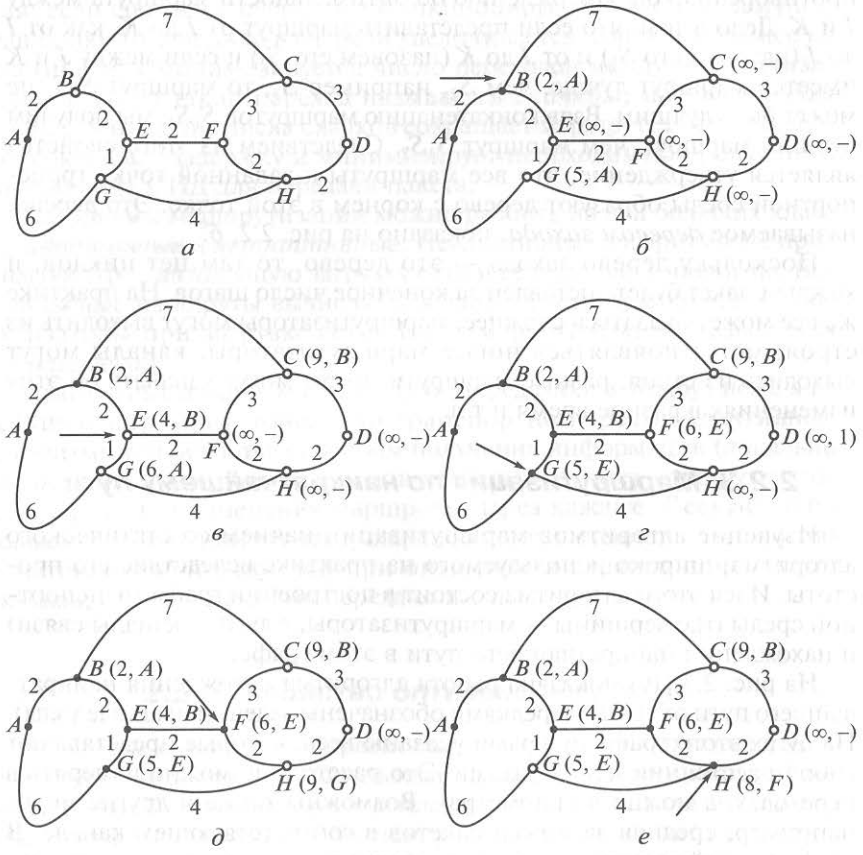


Рис. 2.3. Иллюстрация шагов (*а*...*е*) алгоритма поиска наикратчайшего пути от *A* к *D*

ки могут быть трех видов: временными, рабочими и постоянными. Изначально все метки временные. Когда обнаруживается, что метка принадлежит наикратчайшему пути до исходной вершины, она превращается в постоянную метку и никогда более не изменяется.

На рис. 2.3, *а*...*е* показаны шаги процесса построения маршрута из *A* в *D*. Пометим вершину *A* как постоянную (зачерненной точкой). Все вершины, смежные с вершиной *A*, пометим как временные (незачерненными точками) и укажем в этих метках вершину, из которой они достижимы и за какое расстояние. Это позволит нам впоследствии при необходимости изменить маршрут. Кроме того, все вершины, смежные с вершиной *A*, пометим расстоянием от *A* до этой вершины. Из всех смежных вершин выберем ту, расстояние до которой самое короткое, и объявим ее рабочей. Таким образом, на первом шаге процесса выберем вершину *B*, а затем *E*.

При этом самое интересное происходит на шаге *г*: в соответствии с принципом кратчайшего пути в качестве рабочей вершины выбираем вершину *G*. Теперь на шаге *д*, когда начнем искать вершины, смежные *H*, увидим, что путь от *F* до *H* короче, чем от *G* до *H*, поэтому в качестве рабочей возьмем здесь вершину *F*, а *G* пометим как временную. Затем пометим *H* как рабочую, после чего останется только преобразовать рабочие вершины в постоянные.

2.2.4. Маршрутизация лавиной

Примером статического алгоритма маршрутизации может также служить следующий алгоритм. Каждый поступающий пакет отправляют по всем имеющимся линиям за исключением той, по которой он поступил. Ясно, что если ничем не ограничить число повторно генерируемых пакетов, это число может расти неограниченно. Поэтому, чтобы ограничить область распространения, время жизни таких пакетов ограничивают. Для этого изначально в заголовке каждого генерируемого пакета устанавливается счетчик переходов. При каждой пересылке от маршрутизатора к маршрутизатору этот счетчик уменьшается на единицу. Когда он достигает нуля, пакет сбрасывается и далее не посылается. В качестве начального значения счетчика выбирается наихудший вариант, например диаметр графа, представляющего топологию транспортной среды. Диаметр графа — это максимальное расстояние по всем парам вершин в графе, а расстояние между вершинами — это наименьшее число ребер, которые необходимо пройти, чтобы из одной вершины достичь другую вершину.

Другим приемом, ограничивающим рост числа дублируемых пакетов, является отслеживание на каждом маршрутизаторе тех пакетов, которые через него однажды уже проходили. Такие пакеты сбрасываются и больше не пересылаются. Для этого каждый маршрутизатор, получая пакет непосредственно от абонентской машины, помечает его надлежащим номером, причем каждый маршрутизатор знает на-

бор номеров, которые может использовать только он. Если поступивший пакет уже имеется в списке ранее встречавшихся номеров, то этот пакет сбрасывается. Для предотвращения безграничного роста такого списка вводится ограничительная константа k. Считается, что все номера, начиная с k, уже встречались.

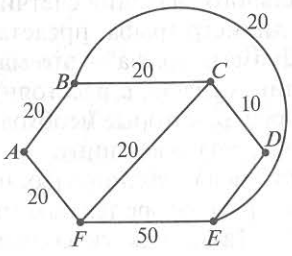
Несмотря на кажущуюся неуклюжесть, этот алгоритм применяется, например, в распределенных базах данных, если требуется параллельно обновить данные во всех базах одновременно. Данный алгоритм всегда позволяет найти наикратчайший маршрут за самое короткое время, поскольку все возможные пути просматриваются параллельно.

2.2.5. Маршрутизация на основе потока

Алгоритмы, которые рассматривались до сих пор, принимали в расчет только топологию транспортной среды и никак не учитывали ее загрузку. Хотя очевидно, что в случае когда наикратчайший маршрут перегружен, лучше воспользоваться пусть более длинным, но менее загруженным маршрутом.

Рассмотрим теперь *статический алгоритм маршрутизации на основе потока*, который учитывает как топологию транспортной среды, так и загрузку СПД.

В некоторых сетях трафик между каждой парой узлов известен заранее и относительно стабилен. Например, при взаимодействии филиалов торгующей организации с головной организацией время подачи, размер и форма отчетов известны заранее. При этих условиях, зная про-



а

Назначение

Источник	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B			8 BC	3 BFD	2 BFE	4 BF
C				3 CD	3 CE	2 CEF
D					3 DCE	4 DF
E						5 EF
F						

б

Рис. 2.4. Топология транспортной среды (*а*) и таблица трафика (*б*)

Таблица 2.1

Итоговый трафик для некоторых пар вершин

i	Линия	λ_1, пак./с	C_1 Кбит/с	μC_1, пак./с	T_1, мс	Значение задержки, мс
1	*AB*	14	20	25	91	0,171
2	*BC*	12	20	25	77	0,146
3	*CD*	6	10	12,5	154	0,073
4	*AE*	11	20	25	71	0,134
5	*EF*	13	50	62,5	20	0,159
6	*FD*	8	10	12,5	222	0,098
7	*BF*	10	20	25	67	0,122
8	*EC*	8	20	25	59	0,098

пускную способность каналов, можно с помощью теории массового обслуживания вычислить среднюю задержку пакета в канале, после чего нетрудно построить алгоритм, вычисляющий путь пакета с минимальной задержкой между двумя узлами.

Для реализации этой идеи требуется заранее знать следующее:

- топологию транспортной среды;
- матрицу трафика F^{ij};
- матрицу пропускных способностей каналов C^{ij};
- алгоритм маршрутизации.

На рис. 2.4, *а* приведен пример топологии транспортной среды с пропускной способностью каналов в килобитах в секунду, а на рис. 2.4, *б* — матрица F^{ij}, где для каждой пары узлов (i, j) указан средний размер трафика в пакетах в секунду и маршрут для этого трафика.

В табл. 2.1 приведен итоговый трафик для некоторых пар соседних вершин.

В табл. 2.1 предполагается, что трафик на каждой линии симметричен, т. е. трафик от *X* к *Y* идентичен трафику от *Y* к *X*. Здесь также приведены среднее число пакетов в секунду (μC_i) при средней длине пакета $1/\mu = 800$ бит и среднее значение задержки, определяемое по формуле

$$T = \frac{1}{\mu C - \lambda},$$

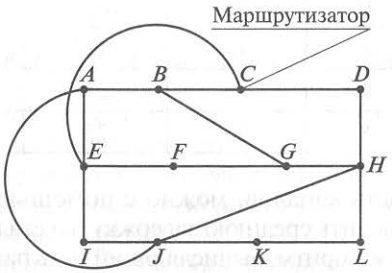
где λ — средняя скорость поступления кадров в секунду.

Эта формула уже приводилась в подразд. 4.3.1 т. 1 данного учебника при построении модели канала с множественным доступом.

Имея эти данные, нетрудно применить алгоритм вычисления наикратчайшего пути к размеченному таким образом графу. Если трафик изменится по какой-либо причине, то достаточно просто заново вычислить таблицу, не изменяя алгоритма.

2.2.6. Маршрутизация по вектору расстояния

Во всех современных транспортных средах используется динамическая маршрутизация, а не статическая. Один из наиболее популярных алгоритмов динамической маршрутизации — *маршрутизация по вектору расстояния*. Этот алгоритм, построенный на идеях алгоритма Беллмана — Форда для нахождения наикратчайшего пути [23] и алгоритма Форда—Фолкерсона [45], определяющего максимальный поток в графе, изначально использовался в сети ARPA и использует-



а

K	A	I	H	K	Новая ожидаемая задержка от J	Линия
A	0	24	20	21	8	A
B	12	38	31	28	20	A
C	15	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
P	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	—
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA	JI	JH	JK		
	задержка 8	задержка 10	задержка 12	задержка 6		
	Векторы, полученные от соседей J				Новая таблица маршрутизации J	

б

Рис. 2.5. Пример маршрутизации по вектору расстояния:

а — топология транспортной среды; *б* — векторы, которые маршрутизатор *J* получил от своих соседей, его замеры задержек до них и итоговая таблица маршрутизации, которую *J* вычислит на основании этой информации

ся в настоящее время в протоколе RIP (Routing IP) [53], а также в сетях Novell, AppleTalk и маршрутизаторах Cisco.

Алгоритм маршрутизации по вектору расстояния работает следующим образом: у каждого маршрутизатора в транспортной среде есть таблица расстояний до всех других маршрутизаторов, принадлежащих этой транспортной среде. Периодически каждый маршрутизатор обменивается этой информацией со своими соседями и обновляет информацию в своей таблице. Каждый элемент этой таблицы включает в себя два поля: первое — номер канала, по которому следует отправлять пакеты, чтобы достичь нужного места, второе — значение задержки до места назначения, которое может измеряться в разных единицах: числе скачков, миллисекундах, длине очереди к каналу и т. д. Фактически в протоколе использовалась версия алгоритма, где эту задержку определяли не на основе пропускной способности канала, а на основе длины очереди к каналу.

Каждые T секунд маршрутизатор шлет своим соседям свой вектор задержек до всех маршрутизаторов в транспортной среде. В свою очередь, он получает такие же векторы от своих соседей. Кроме того, он постоянно замеряет задержки до своих соседей, следовательно, имея векторы расстояний от соседей и зная расстояние до них, маршрутизатор всегда может вычислить наикратчайший маршрут до определенного места в транспортной среде.

На рис. 2.5 приведен пример маршрутизации по вектору расстояния [19].

Рассмотрим, как маршрутизатор J с помощью итоговой таблицы маршрутизации вычислит маршрут до G. Маршрутизатор J знает, что он может достичь A за 8 мс, маршрутизатор A объявляет, что от него до G 18 мс. Таким образом, J может достичь G за 26 мс через A. Аналогично можно подсчитать, что достичь G через I, H и K можно соответственно за 41 (31 + 10), 18 (6 + 12) и 37 (31 + 6) мс. Наилучшее полученное значение — 18, следовательно, этот маршрут и является наилучшим по критерию скорости доставки.

Проблема бесконечного счетчика задержки

Алгоритм маршрутизации по вектору расстояния теоретически работает хорошо, но у него есть один недостаток: он очень медленно реагирует на разрушения каналов в транспортной среде. Информация о появлении хорошего маршрута в транспортной среде распространяется более или менее быстро, а вот данные о потере, разрушении какого-то маршрута распространяются значительно медленнее.

Рассмотрим пример транспортной среды с линейной топологией, показанный на рис. 2.6, *а* [19]. Пусть изначально маршрутизатор A не работал, поэтому у всех маршрутизаторов расстояние до него было равно ∞. Пусть в какой-то момент времени A включился. По истече-

Рис. 2.6. Пример топологии транспортной среды с изначально не работающим маршрутизатором *A* (*а*) и с полностью работоспособными маршрутизаторами и каналами (*б*)

нии определенного времени маршрутизаторы начнут обмениваться векторами, и *B* узнает об *A*. Еще через один обмен векторами об *A* узнает *C* и т. д. Таким образом, информация о новом маршруте будет распространяться линейно шаг за шагом при каждом обмене векторами. Если самый длинный маршрут в транспортной среде имеет длину *N*, то потребуется *N* обменов векторами, пока информация о новом маршруте дойдет до самого удаленного узла.

Теперь рассмотрим обратную ситуацию, показанную на рис. 2.6, *б*, т. е. когда изначально все маршрутизаторы и каналы работоспособны.

Пусть в какой-то момент времени канал между *A* и *B* оказался разрушен. В этом случае *B* перестает видеть *A*, но *C* говорит *B*: «У меня есть маршрут до *A*». При этом *B* не подозревает, что маршрут от *C* до *A* идет через него же. Маршрутизаторы *D* и *E* своих таблиц не изменяют. Их расстояния до *A* на единицу больше, чем у *C*. При втором обмене *C* увидит, что оба его соседа достигают *A* за 3 единицы. Некоторым случайным образом *C* выбирает одного соседа, увеличив значение 3 на единицу. Плохая весть будет распространяться медленно, пока счетчики задержек не примут некоторого очень большого значения, практически бесконечного для данной сети. Только после этого станет ясно, что *A* недостижим ни через *C*, ни через *D*, ни через *E*. Сколько времени на это потребуется зависит от конкретного значения бесконечности в данной транспортной среде.

Разделение направлений (Split Horizon Hack)

Одним из решений проблемы бесконечного счетчика задержки является следующее. Алгоритм маршрутизации по вектору расстояния

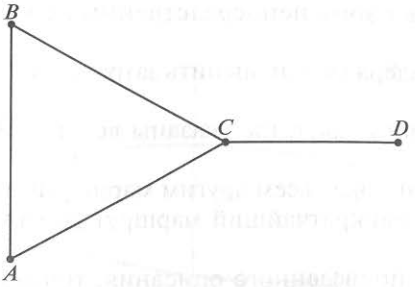


Рис. 2.7. Пример, когда алгоритм разделения направлений не помогает: *A*... *D* — маршрутизаторы

работает так, как было описано ранее, но при передаче вектора по линии, по которой направляются пакеты для маршрутизатора *X*, т. е. по которой достижим маршрутизатор *X*, расстояние до *X* указывается как бесконечность. Теперь у алгоритма маршрутизации по вектору расстояния (см. рис. 2.6, *б*) проблемы бесконечного счетчика не возникнет. Действительно, когда маршрутизатор *A* «упадет», при первом же обмене *B* это обнаружит, но *C* также будет посылать вектор *B*, согласно которому *A* недостижим (∞). При следующем обмене *C* увидит, что *A* недостижим и через *B*, и через *D*, и также отметит его как недостижимый узел.

Однако и в алгоритме разделения направлений есть «дыры». Рассмотрим пример, показанный на рис. 2.7. Если линия между *C* и *D* будет разрушена, то *C* сообщит об этом *A* и *B*. Однако *A* знает, что у *B* есть маршрут до *D*, а *B* знает, что такой маршрут есть и у *A*. И опять мы «сваливаемся» в проблему бесконечного счетчика.

2.2.7. Маршрутизация по состоянию канала

Алгоритм маршрутизации по вектору расстояний использовался в сети ARPANET до 1979 г., после чего он был заменен по двум основным причинам. Первая причина — это то, что в нем никак не учитывалась пропускная способность канала, поскольку задержка в основном определялась длиной очереди. Пока основная масса каналов имела пропускную способность 56 Кбит/с, это было не страшно. Однако, когда появились каналы с пропускной способностью 256 Кбит/с и 1,5 Мбит/с, это стало недостатком. Вторая причина — это медленная сходимость алгоритма при изменениях в транспортной среде. По этим причинам был создан новый алгоритм маршрутизации по состоянию канала.

Идею алгоритма маршрутизации по состоянию канала можно описать в виде пяти основных шагов, которые должен выполнить каждый маршрутизатор в транспортной среде:

1. Определить своих непосредственных соседей и их сетевые адреса.

2. Измерить задержку или оценить затраты на передачу до каждого соседа.

3. Сформировать пакет, где указаны все данные, полученные на шаге 2.

4. Послать этот пакет всем другим маршрутизаторам.

5. Вычислить наикратчайший маршрут до каждого маршрутизатора.

Как видно из приведенного описания, топология транспортной среды и все задержки в СПД оцениваются всеми узлами экспериментально. После этого можно использовать, например, алгоритм Дейкстры для вычисления наикратчайшего маршрута. Теперь рассмотрим подробнее эти пять шагов.

Определение соседей

При загрузке маршрутизатор, прежде всего, определяет, кто его непосредственные соседи. Для этого он рассылает по всем подсоединенным к нему каналам специальный пакет HELLO, и все маршрутизаторы отвечают, указывая свое уникальное имя. Имя маршрутизатора должно быть уникальным в сети, чтобы избежать неоднозначностей.

Если же два и более маршрутизаторов соединены каналом с множественным доступом, как на рис. 2.8 *а*, то этот канал в графе связей представляют отдельным искусственным узлом (рис. 2.8, *б*).

Оценка затрат

Оценка затрат до каждого соседа происходит с помощью другого специального пакета ECHO, который рассылается *всем соседям*, при этом замеряется задержка от момента отправки этого пакета до мо-

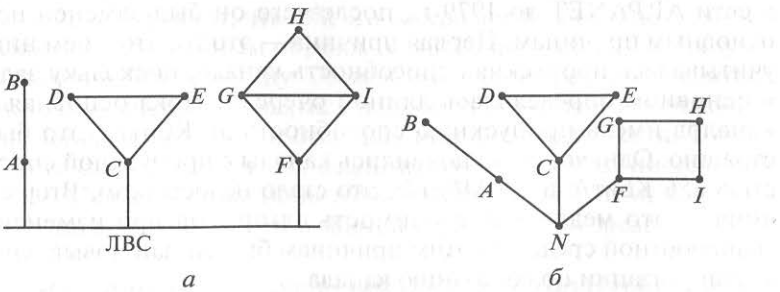


Рис. 2.8. Пример определения соседей:
а — топология ТС; *б* — граф связей

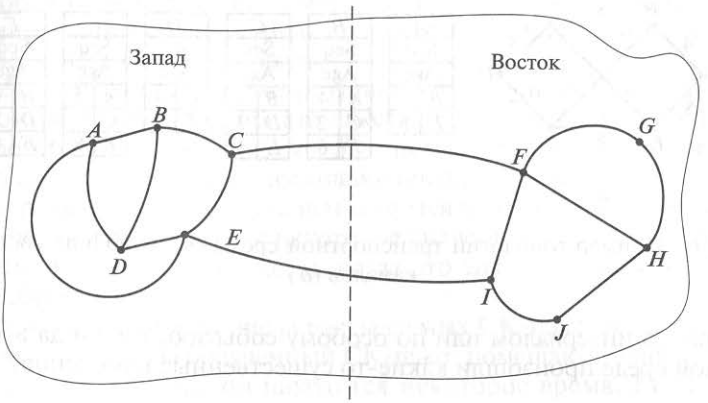


Рис. 2.9. Пример учета загрузки канала при маршрутизации по состоянию

мента его возвращения. Все маршрутизаторы, которые получают такой пакет, обязаны отвечать незамедлительно, отправляя пакет обратно. Такие замеры делают несколько раз и вычисляют их среднее значение. Таким образом, длина очереди к каналу не учитывается.

Здесь необходимо понять: следует учитывать загрузку канала или нет? Если учитывать, то задержку надо замерять от момента поступления пакета в очередь к каналу, а если не учитывать, то — от момента, когда пакет достиг головы очереди. Есть доводы и за учет загрузки, и против ее учета. При учете загрузки можно выбирать между двумя и более каналами с одинаковой пропускной способностью, обеспечивая лучшую производительность.

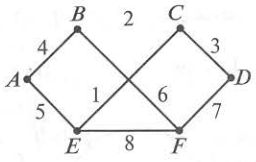
Однако можно привести примеры, когда учет загрузки вызывает проблемы. Например, рассмотрим рис. 2.9, где два одинаковых по производительности канала *CF* и *EI* соединяют две транспортные среды. Когда загрузка одного из этих каналов меньше, то он предпочтительнее другого, что через некоторое время приводит к его перегрузке, и предпочтительнее становится другой канал. Если загрузка не учитывается, то такой проблемы не возникает.

Формирование пакетов состояний каналов

После выполнения всех указанных измерений можно формировать пакеты состояний каналов.

На рис. 2.10, *б* показаны пакеты состояний каналов для примера транспортной среды, приведенной на рис. 2.10, *а*.

В пакетах состояний указываются: отправитель, последовательное число, возраст (назначение этих полей станет ясно позднее), список соседей и задержки до них. Формирование таких пакетов не вызывает проблем. Основной вопрос здесь, когда их формировать: периодически



а

A	
Seq	
Age	
B	4
E	5

B	
Seq	
Age	
A	4
C	2
F	6

C	
Seq	
Age	
B	2
D	3
E	1

D	
Seq	
Age	
C	3
F	7

E	
Seq	
Age	
A	5
C	1
F	8

F	
Seq	
Age	
B	6
D	7
E	8

б

Рис. 2.10. Пример топологии транспортной среды (*а*) и таблица состояний каналов (*б*)

с каким-то интервалом или по особому событию, т. е. когда в транспортной среде произошли какие-то существенные изменения?

Распространение пакетов состояний каналов

Наиболее хитрая часть этого алгоритма — надежное распространение пакетов состояний каналов (далее СК-пакетов). Как только СК-пакет получен и включен в работу, маршрутизатор начинает использовать его при определении маршрута. При неудачном распространении СК-пакетов разные маршрутизаторы могут получить разное представление о топологии транспортной среды, что может привести к возникновению циклов, появлению недостижимых машин и другим проблемам.

Рассмотрим сначала базовый алгоритм, в котором СК-пакеты распространяются методом лавины, т. е. СК-пакет рассылается всем соседям, которые, в свою очередь, пересылают его своим соседям и т. д. Однако чтобы не потерять контроль и не вызвать неограниченного дублирования СК-пакетов, каждый маршрутизатор ведет счетчик последовательных номеров СК-пакетов, которые он сгенерировал. Все маршрутизаторы запоминают пары маршрутизатор — последовательное число, которые они уже встречали среди полученных СК-пакетов. Если поступивший СК-пакет содержит пару, которая еще не встречалась маршрутизатору, то он отправляет этот СК-пакет всем своим соседям, за исключением того соседа, от которого он его получил. Если маршрутизатор уже встречал такой пакет, то этот пакет сбрасывается и никуда не дублируется.

У этого алгоритма есть несколько проблем, но все они разрешимые.

Первая проблема — размер поля последовательных номеров пакетов. Если это поле будет недостаточно велико, то его переполнение приведет к повтору номеров, а следовательно, к некорректной работе всего алгоритма. Решением здесь является использование достаточно большого поля, например 32-разрядного. В этом случае если обмен СК-пакетами происходит раз в секунду, потребуется 137 лет, чтобы возникло переполнение.

Вторая проблема — если маршрутизатор «упал» по какой-либо причине и потерял уже использованные последовательные номера, то неясно, как их восстановить.

Третья проблема — если в результате передачи возникнет ошибка в одном бите, например вместо пакета с номером 4 получим пакет с номером 65540, то все пакеты с 5-го номера по 65 540-й будут сбрасываться как устаревшие, поскольку текущий номер — 65 540.

Для решения этих проблем используется поле «Возраст» СК-пакета, в котором устанавливается некоторое значение, уменьшающееся на единицу при каждом скачке, и, когда это значение достигнет нуля, пакет сбрасывается.

В целях сокращения числа рассылаемых СК-пакетов их рассылают не сразу. Сначала полученный СК-пакет помещают в специальную область задержки, где он находится некоторое время. Если за это время придет другой пакет от того же источника, то эти пакеты сравниваются, и если они одинаковые, то вновь пришедший пакет сбрасывается, если же они различаются, то последний пришедший пакет дублируется и отправляется другим маршрутизаторам, а первый пакет сбрасывается. Все СК-пакеты передаются с уведомлением.

Структура данных, используемая маршрутизатором *B* из примера, приведенного на рис. 2.10, *а*, показана на рис. 2.11 [19]. Здесь каждая строка таблицы соответствует последнему поступившему, но не до конца обработанному СК-пакету. При этом для каждого канала маршрутизатора *B* указано с помощью флага, был ли соответствующий СК-пакет отправлен и подтвержден. Флаг отправления означает, что соответствующий СК-пакет должен быть послан по указанному каналу. Флаг уведомления указывает, что по соответствующей линии должно прийти подтверждение.

На рис. 2.11 СК-пакет от маршрутизатора *A* прибыл и он должен быть переслан в *C* и в *F* и подтвержден для *A*. Аналогичная ситуация с СК-пакетом, поступившим от *F*. Существенно отличается ситуация с маршрутизатором *E*, от которого поступило два СК-пакета: один

			Флаги отправления			Флаги уведомления			
Источник	Номер	Возраст	*A*	*C*	*F*	*A*	*C*	*F*	Данные
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Рис. 2.11. Буфер пакетов для маршрутизатора *B* из примера, приведенного на рис. 2.10, *а*

по маршруту *EAB*, а другой по маршруту *EFB*. Следовательно, СК-пакет должен быть послан только в *C*, а вот уведомления должны быть посланы и в *A*, и в *F*. Наличие дубликатов СК-пакетов легко распознать по состоянию флагов отправки.

Вычисление нового пути

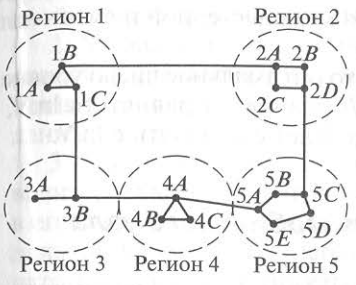
Когда маршрутизатор получил полный комплект СК-пакетов, он может построить топологию транспортной среды и, например, локально запустить алгоритм Дейкстры для вычисления наикратчайшего пути.

В системе, где есть n маршрутизаторов, имеющих по k линий, у каждого из этих маршрутизаторов должно быть достаточно памяти для хранения необходимой информации о сети. При больших значениях n этот объем памяти может стать существенным. Кроме того, в сетях, где число маршрутизаторов достигает десятков или сотен тысяч, проблемы, вызванные сбоем одного из них, могут оказаться весьма серьезными. Например, в случае если из-за сбоя маршрутизатор послал неправильный СК-пакет или неправильно оценил состояние канала, в дальнейшем это может создать большие проблемы.

Маршрутизация по состоянию каналов широко используется в реальных сетях. Например, в протоколе OSPF, который будет подробно рассматриваться далее, и в протоколе IS-IS [11], который был изначально предложен фирмой DEC, а позже адаптирован ISO. Протокол IS-IS широко используется в Интернете. Поскольку OSPF появился позже IS-IS, он вобрал в себя все усовершенствования, сделанные для IS-IS, а основное различие между ними заключается в том, что IS-IS может работать с разными протоколами сетевого уровня, что делает его весьма привлекательным при маршрутизации между разнотипными сетями, чего не может делать протокол OSPF.

2.2.8. Иерархическая маршрутизация

По мере роста транспортной среды размеры таблиц маршрутизации, а следовательно, затраты памяти, времени, необходимого процессору для обработки этих таблиц, пропускной способности каналов на передачу служебной информации для актуализации таблиц, могут превысить разумные пределы. Другими словами, затраты маршрутизатора на вычисление оптимального маршрута и поддержание данных для его вычисления в актуальном состоянии станут столь большими, что подавляющая часть ресурсов будет тратиться не на маршрутизацию, а на ее обеспечение. Маршрутизация начнет «тормозить» остальные сервисы в сети. Таким образом, дальнейший рост транспортной среды, а следовательно, и сети, в которой каждый маршрутизатор знает все о каждом другом маршрутизаторе, будет невозможен. Ре-



а

Назначение	Линия	Переходы
1A	—	—
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1B	5
5D	1C	5
5E	1C	5

б

Назначение	Линия	Переходы
1A	—	—
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

в

Рис. 2.12. Иерархическая маршрутизация:

а — сеть с двухуровневой иерархией из пяти регионов; *б* — полная таблица маршрутизации; *в* — иерархическая таблица

шение этой проблемы — создание иерархии сетей подобно иерархии коммутаторов в телефонной сети.

(Для краткости далее будем использовать термины «сеть» и «транспортная среда» как эквивалентные. Если будет возникать различие в их толковании, это будет оговорено особо.)

На рис. 2.12, *а* приведен пример сети с двухуровневой иерархией из пяти регионов, где каждый из регионов соединен с двумя другими регионами. На рис. 2.12, *б* показана полная таблица маршрутизации для маршрутизатора 1*A* без иерархии, а на рис. 2.12, *в* — та же таблица при двухуровневой иерархии. Нетрудно увидеть, что таблица маршрутизации во втором случае резко сократилась. Однако за эту экономию приходится платить эффективностью маршрутизации. Например, наилучший маршрут от 1*A* к 5*C* проходит через регион 2, но в данном случае он пройдет через регион 3, поскольку большинство машин в регионе 5 могут быть эффективнее достигнуты через регион 3.

При построении иерархии возникает сразу несколько вопросов. Один из них — сколько уровней должно быть в иерархии при заданном размере сети? Например, если наша транспортная среда содержит 920 маршрутизаторов, то без иерархии таблица каждого их них будет иметь 920 строк. Если транспортную среду разбить на 40 регионов, то такая таблица будет содержать 23 строки для маршрутизации внутри региона плюс 40 строк для маршрутизации между регионами. Если использовать трехуровневую иерархию и объединить регионы в кластеры, то при пяти кластерах по восемь регионов с 23

маршрутизаторами в каждом в таблице будет 23 входа для внутрикластерной маршрутизации, семь — для межкластерной и пять — для межрегиональной. Итого 35 входов.

Клейнрок и Камоун показали [58], что оптимальное число уровней иерархии в транспортной среде при N узлах будет равняться $\ln N$, а число строк в таблице маршрутизатора будет составлять $\mathbf{e} \ln N$.

2.2.9. Маршрутизация для мобильного узла

Миллионы людей в наши дни путешествуют, ездят в командировки. Многим из них просто необходимо иметь доступ к своей электронной почте, файловой системе. Как уже отмечалось, мобильные приложения в настоящее время составляют значительную часть сетевых приложений. Прежде всего, постараемся разобраться, в чем заключается проблема использования мобильного узла.

Всех пользователей в сети можно подразделить на две большие группы. *Стационарные* пользователи — это большая группа людей, компьютеры которых подключены к сети стационарными средствами (проводами, кабелями) и которые редко изменяют свое местоположение. *Мобильные* пользователи постоянно изменяют свое местоположение и при этом стремятся поддерживать связь с сетью. С позиции модели работы стационарных пользователей мобильный пользователь должен сначала как-то соединиться со своим домашним стационарным компьютером и уже через него выходить в сеть. Однако если домашний компьютер находится в Москве, а его пользователю, который находится в Сан-Франциско, необходимо соединиться с сервером, расположенным в Сан-Франциско, то, согласитесь, было бы странно создавать сетевое соединение с его домашним сервером в Москве и уже оттуда соединяться с сервером в Сан-Франциско.

Предполагается, что в сети каждый пользователь имеет постоянное домашнее местоположение, которое никогда не изменяется. Проблема мобильной маршрутизации в этих условиях решается посылкой пакетов мобильному пользователю через его домашнее местоположение. При этом место, где находится сам пользователь, не имеет значения.

Всю сеть разобьем на области. Пусть при этом в каждой области есть *агент визитеров*, который знает обо всех мобильных пользователях в своей области, а также в каждой области есть *домашний агент*, который знает обо всех стационарных пользователях в своей области, которые в настоящий момент путешествуют. В этом случае, как только мобильный узел подключается к местной локальной сети, он регистрируется у агента визитеров. Процедура такой регистрации примерно следующая [19]:

1. Периодически агент визитеров рассылает по своей области пакет, где указаны его местоположение и адрес. Если мобильный узел,

подключившись к сети, долго не видит такого пакета, он рассылает свой пакет с просьбой агенту визитеров объявить свои координаты.

2. Мобильный узел регистрируется у агента визитеров, указывая свое текущее местоположение, домашнее местоположение и определенную информацию, связанную с безопасностью передаваемых данных.

3. Агент визитеров обращается через сеть к домашнему агенту визитера, указывая, что один из его пользователей сейчас находится в его области, и передает конфиденциальную информацию, которая должна убедить домашнего агента, что это действительно его пользователь пытается соединиться с ним.

4. Домашний агент изучает конфиденциальные данные, и, если они соответствуют тем данным, которые имеются у него об этом пользователе, он дает подтверждение запросившему агенту визитеров.

5. Агент визитеров, получив подтверждение от домашнего агента, заносит данные о мобильном узле этого пользователя в свои таблицы и регистрирует его.

В идеале, пользователь, покидая область визита, должен закрыть свою временную регистрацию. Однако, как правило, закончив сеанс связи, пользователь просто выключает свой компьютер, поэтому, если по прошествии некоторого времени пользователь не объявился вновь, агент визитеров считает его покинувшим область.

Рассмотрим теперь, что происходит, когда кто-то посылает сообщения мобильному узлу. Пакет поступает на адрес домашнего местоположения пользователя, где его перехватывает домашний агент. Домашний агент инкапсулирует этот пакет в свой пакет, который он отправляет по адресу агента визитеров той области, откуда последний раз был сеанс связи с пользователем. Одновременно с этим домашний агент посылает сообщение отправителю пакета, чтобы он все последующие пакеты мобильному узлу инкапсулировал в сообщениях, направляемых по адресу агента визитеров. Такой механизм инкапсулирования одних пакетов в другие называется *туннелированием* (см. гл. 4 т. 1 данного учебника).

Здесь мы обрисовали в общих чертах лишь основную схему маршрутизации для мобильного узла. Конкретных схем существует множество. Они различаются, прежде всего, тем, как распределяется работа между маршрутизаторами и хостами. Есть схемы, где маршрутизаторы запоминают информацию о местонахождении мобильных узлов и могут вмешиваться в диалог между агентом визитеров и домашним агентом, по-разному маршрутизируя трафик. В некоторых схемах мобильный узел получает некоторый уникальный адрес, а в других — адрес агента, который отвечает за маршрутизацию всего трафика мобильных узлов. Кроме того, схемы различаются разным уровнем безопасности передаваемой информации. Подробнее маршрутизация для мобильных узлов рассмотрена в [28, 49].

2.2.10. Маршрутизация при вещании

В некоторых приложениях возникает потребность пересылки одного и того же сообщения всем машинам, например прогноза погоды, новостей и т.д. Такой режим передачи, как известно, называется *вещанием*. Есть несколько решений реализации этого режима.

Первое решение — источник знает, кому и что надо послать, и генерирует столько сообщений, сколько имеется получателей. Это одно из самых плохих решений. Оно не требует никаких специальных средств, однако весьма затратно. Тратится не только пропускная способность каналов, но и память, так как где-то необходимо хранить весь лист рассылки и экземпляры рассылаемого сообщения.

Второе решение — использование метода лавины. Однако, как мы уже видели, он затратен для каналов типа точка—точка: слишком сильно расходуется пропускная способность.

Третье решение — маршрутизация множественной доставки. В этом случае каждый пакет должен иметь либо лист рассылки, либо маршрут рассылки. Лист рассылки содержит лишь адреса получателей, а маршрут рассылки — последовательность маршрутизаторов, через которые должен проследовать пакет. Каждый маршрутизатор, получив такой пакет, отправляет и дублирует его в соответствии с маршрутом рассылки.

Четвертое решение — использование дерева захода либо любого другого подходящего дерева связей. Дерево захода позволяет избежать циклов и ненужного дублирования пакетов. В этом случае каждый маршрутизатор дублирует пакет вдоль линий, соответствующих дереву захода, кроме той линии, по которой пакет пришел. При таком решении очень рационально используется пропускная способность каналов и генерируется абсолютный минимум пакетов при рассылке. Однако у каждого маршрутизатора должны быть деревья захода.

Пятое решение — неявное использование дерева связей. В этом случае, когда пакет поступает, маршрутизатор проверяет, по какой линии он поступил. Если пакет поступил по линии, которая используется для отправления пакетов источнику вещательного пакета, то вещательный пакет дублируется и рассылается по всем линиям, кроме той линии, по которой пакет пришел. Если пакет поступил не по этой линии, то он сбрасывается.

Данный метод реализации вещания называется *пересылкой вдоль обратного пути*. Рассмотрим пример использования этого метода [19]. На рис. 2.13, *а* показана топология транспортной среды, на рис. 2.13, *б* — дерево захода для вершины *I*, а на рис. 2.13, *в* — схема работы этого алгоритма. Сначала в вершине *I* были сгенерированы, а затем разосланы четыре пакета. Во все четыре вершины (*F*, *H*, *J*, *N*), куда поступили эти пакеты, они пришли с направления, предпочтительного для достижения вершины *I*. Из восьми пакетов, сгенериро-

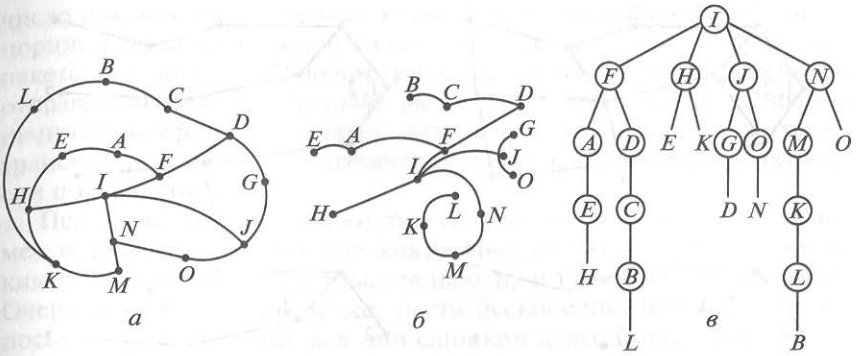


Рис. 2.13. Транспортная среда (*а*), дерево захода для вершины *I* (*б*) и дерево, построенное методом пересылки вдоль обратного пути (*в*)

ванных на следующем этапе, согласно дереву захода только пять поступили по предпочтительному для *I* направлению. На третьем этапе из шести сгенерированных пакетов только три поступили по предпочтительному направлению (в *G*, *D*, *N* поступили дубликаты). После пяти этапов сгенерированы 23 пакета, и рассылка прекращается. Достоинствами этого метода являются простота и легкость реализации, а сложность заключается в построении дерева заходов для каждого случая рассылки.

2.2.11. Маршрутизация при групповой передаче

Групповая передача используется, когда надо обеспечить взаимодействие группы взаимосвязанных процессов, разбросанных по сети. Такие ситуации часто встречаются в распределенных приложениях, например при работе с распределенными базами данных. Если размер такой группы сравним с размерами сети, то можно использовать методы вещания, однако при маленьком размере группы такой подход будет неэкономичным. Кроме того, если рассылаемая в группе информация конфиденциальная, то алгоритмы вещания не подходят. Так мы приходим к проблеме групповой маршрутизации.

В случае обмена информацией в группе кроме алгоритма групповой маршрутизации необходимы алгоритмы управления группой, которые должны обеспечивать средства для ее реконфигурации: включение новых членов, удаление старых и т. п. Однако эти проблемы не затрагивают алгоритм групповой маршрутизации, поэтому здесь мы их рассматривать не будем.

Алгоритм групповой маршрутизации, как правило, основывается на дереве связей, т. е. каждый маршрутизатор в транспортной среде вычисляет дерево связей, охватывающее все остальные маршрутизаторы.

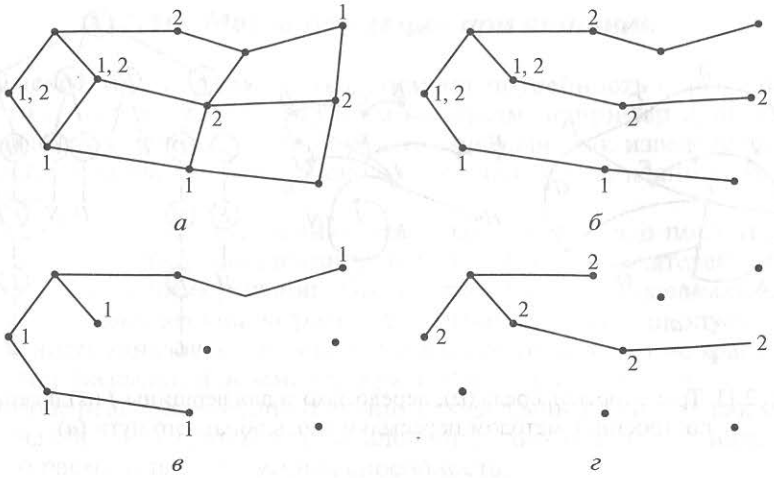


Рис. 2.14. Групповая маршрутизация:
а — топология ТС; *б* — дерево связей для групп 1 и 2 узлов; *в* — дерево связей для группы 1; *г* — дерево связей для группы 2

На рис. 2.14, *а* приведен пример транспортной среды, состоящей из двух групп (их номера указаны у вершин), в которой некоторые вершины принадлежат как группе 1, так и группе 2. На рис. 2.14, *б* показано дерево связей для самой левой вершины. Когда процесс посылает групповой пакет, первый же маршрутизатор редуцирует в своем дереве все связи, которые не ведут в вершины, не являющиеся членами группы. На рис. 2.14, *в* показано редуцированное дерево связей для группы 1, а на рис. 2.14, *г* — для группы 2.

Для редукции дерева связей используются разные алгоритмы, наиболее простой из которых основывается на применении алгоритма маршрутизации по состоянию канала. В этом случае каждый маршрутизатор имеет полную конфигурацию транспортной среды. Редукция дерева связей начинается от вершин — членов группы — и разворачивается в сторону корня, удаляя все маршрутизаторы, которые не ведут к членам группы. Следует отметить, что такая редукция требует дополнительных накладных расходов на обмен служебной информацией между маршрутизаторами в транспортной среде.

Подробнее алгоритмы маршрутизации рассмотрены в [11, 28].

2.3. Алгоритмы управления перегрузками

2.3.1. Общие сведения

Когда в транспортной среде находится в одно и то же время слишком много пакетов, ее производительность начинает падать. Когда

число пакетов, отправляемых абонентскими машинами в сеть, пропорционально пропускной способности сети, то число посланных пакетов пропорционально числу доставленных пакетов. Когда число отправляемых пакетов начинает расти, а число доставленных пакетов начинает непропорционально снижаться, происходит перегрузка транспортной среды, при которой доставка пакетов может практически прекратиться.

Перегрузка может возникнуть в силу нескольких причин. Например, если сразу несколько потоков данных, поступающих по нескольким входным линиям, устремятся на одну и ту же выходную линию. Очередь на этой линии может расти бесконечно, и пакеты начнут посылать повторно, так как они слишком долго будут находиться в очереди. Если буфер маршрутизатора переполнится, то маршрутизатор начнет терять пакеты. Увеличение памяти в этом случае вряд ли исправит положение. Пакеты долго будут находиться в памяти, и отправители начнут их дублировать.

Перегрузки могут возникнуть и из-за недостаточной скорости процессора. Если процессор будет не в состоянии справиться своевременно с рутинными задачами (с размещением пакета в буфере, корректировкой таблиц и т. п.), то даже при наличии линий с достаточной пропускной способностью очередь будет расти. Аналогичная картина может случиться при быстром процессоре, но медленном канале, и наоборот.

Таким образом, причина возникновения перегрузок — *несбалансированность производительности компонентов транспортной среды*.

Перегрузки имеют тенденцию к усилению, т. е. увеличению числа «потерянных» пакетов и ухудшению ситуации. Если у маршрутизатора не хватает памяти буфера, то он начинает сбрасывать пакеты, а отправитель начинает посылать их снова и снова, увеличивая загруженность транспортной среды.

Следует различать управление перегрузками и управление потоком. Перегрузка — это глобальная проблема в сети. Управление перегрузками — это процесс регулирования потоков данных в транспортной среде, при котором эти потоки не превышают ее пропускной способности. Эта глобальная проблема затрагивает поведение всех абонентских машин и всех маршрутизаторов в сети.

Управление потоком возникает между парой взаимодействующих машин. Это локальная проблема, касающаяся только двух взаимодействующих машин. Ее решение гарантирует, что быстрый отправитель сообщений не «завалит» нерасторопного получателя. Например, один быстрый компьютер передает со скоростью 1 Гбит/с файл размером 1 Гбит более медленному компьютеру через СПД с пропускной способностью 1 Тбит/с. Ясно, что здесь не будет перегрузки, хотя быстрый компьютер может создать такой поток пакетов, что он захлестнет медленный компьютер. В то же время если сеть, в СПД

которой пропускная способность любой линии не превышает 1 Мбит/с, будет содержать 1 000 компьютеров и хотя бы половина из них начнет передавать файлы со скоростью 100 Кбит/с другой половине, то ясно, что при неудачной топологии СПД перегрузки не избежать.

Часто управление перегрузкой и управление потоком путают из-за того, что и там, и там применяются одинаковые приемы, например направление источникам специальных пакетов, подавляющих нарастание потоков.

2.3.2. Основные принципы управления перегрузками

В терминологии теории управления все методы управления перегрузками в сетях можно разбить на две большие группы: с открытым контуром управления и с замкнутым контуром управления. Методы с открытым контуром управления предполагают, что все продумано и предусмотрено заранее в конструкции системы, и если нагрузка находится в заданных пределах, то перегрузки не происходит. Если же нагрузка начинает превышать определенные пределы, то заранее известно, когда и где начнется сброс пакетов, в каких точках сети начнется перепланировка ресурсов и т. п. Главное, что все это будет происходить вне зависимости от специфики текущего состояния сети.

Методы управления с замкнутым контуром основаны на использовании обратной связи и включают в себя следующие три этапа:

- наблюдение за сетью для определения, где и когда началась перегрузка;
- передача данных туда, где требуется предпринять надлежащие меры;
- перестройка функционирования сети для устранения проблемы.

При наблюдении за сетью для определения перегрузки используются разные метрики, из которых основными являются:

- процент пакетов, сброшенных из-за нехватки памяти в буферах;
- средняя длина очередей в сети;
- число пакетов, для которых наступил time-out и для которых были сделаны повторные передачи;
- средняя задержка пакета при доставке и среднее отклонение задержки при доставке пакета.

Следующий шаг при использовании обратной связи — передача информации о перегрузке туда, где что-то может быть сделано для исправления положения. Например, маршрутизатор, обнаруживший перегрузку, может направить сообщение об этом всем источникам сообщений, причем ясно, что это увеличит нагрузку в сети именно в тот момент, когда это менее всего желательно. Однако имеются и другие варианты. Например, в каждом пакете можно зарезервировать специальный бит перегрузки, и если какой-то маршрутизатор обна-

ружил перегрузку, то он устанавливает этот бит, тем самым сообщая другим о ней (вспомним структуру кадра во Frame Relay, рассмотренную в т. 1 данного учебника).

Также можно использовать решение, напоминающее прием, применяемый некоторыми радиостанциями: направлять несколько автомашин по дорогам для обнаружения пробок, а затем сообщать о них по радиоканалам, предупреждая водителей и призывая их пользоваться объездными путями. Аналогично в сети рассылаются специальные пробные пакеты, которые позволяют проверить нагрузку, например замеряя время реакции на эти пакеты, и если где-то при этом обнаружится перегрузка, то о ней сообщается всем и пакеты перенаправляются таким образом, чтобы обогнуть перегруженные участки.

Методы управления перегрузками с открытым контуром, в свою очередь, подразделяются на две группы: воздействующие на источники и воздействующие на получателей, а методы с замкнутым контуром — на группы с явной обратной связью и неявной обратной связью. Явная обратная связь предполагает, что источнику посылается специальный пакет, который информирует его о перегрузке. Неявная обратная связь основывается на том, что источник сам определяет факт перегрузки на основе своих локальных наблюдений за трафиком, например по значению задержки поступления уведомления о доставке пакета.

Появление перегрузки означает, что, возможно, временно нагрузка превысила ресурсы сети или некоторой ее части. Есть два выхода из этого положения: увеличить ресурсы или сократить нагрузку. Увеличить ресурсы чаще всего невозможно, поэтому остается только сокращение нагрузки. Для этого имеется несколько способов: отказать некоторым пользователям в сервисе, ухудшить сервис всем или некоторым пользователям, заставить пользователей планировать свои потоки определенным образом.

2.3.3. Методы, предотвращающие перегрузки

Рассмотрение начнем с методов, предотвращающих перегрузки с открытым контуром. Эти методы ориентированы на минимизацию перегрузок при первых признаках их проявлений, а не на борьбу с перегрузками, когда они уже случились.

Факторы, влияющие на перегрузки, могут располагаться и на канальном, и на сетевом, и на транспортном уровнях.

На канальном уровне вызвать перегрузку может, во-первых, повторная пересылка кадров: если у источника сообщений часто возникает time-out и он начинает повторно передавать пакет, то тем самым он лишь усугубляет положение. Во-вторых, перегрузка здесь возможна вследствие нарушения порядка следования пакетов при передаче: если получатель часто сбрасывает пакеты, поступившие не в надлежащем порядке от источника, то их повторная передача будет

также усугублять перегрузку. В-третьих, организация рассылки уведомлений также влияет на перегрузку: если уведомление происходит немедленно и специальными пакетами, то это увеличивает трафик и, следовательно, может привести к перегрузкам. Если для уведомления используются пакеты с сообщениями, то возможны time-out из-за отсутствия уведомлений вовремя и, как следствие, повторные пересылки пакетов, что может привести к перегрузкам. В то же время жесткая схема управления потоком (небольшое окно) сдерживает нарастание трафика и предотвращает появление перегрузок.

На сетевом уровне на появление перегрузок влияет выбор качества сервиса: с виртуальными соединениями или с дейтаграммами. На этом уровне большинство методов борьбы с перегрузками ориентировано на виртуальные соединения. Методы управления очередями, т. е. организации очередей, тоже влияют на появление перегрузок. При этом возможны следующие варианты: одна общая очередь на входе или одна общая на выходе; по одной на каждую входную линию или на каждую выходную линию; по одной очереди на каждую входную и выходную линии. Выбор метода сброса пакетов также влияет на перегрузки. Правильная маршрутизация, равномерно использующая каналы в транспортной среде, позволяет избежать перегрузки. На образование перегрузок влияют и методы, регулирующие время жизни пакета в сети. Если пакет долго блуждает в сети, прежде чем будет принято решение о его сбросе, то это плохо, так как увеличивает трафик и может привести к перегрузке. Если же поторопиться, то преждевременный сброс пакета может привести к повторным передачам, что опять-таки увеличит нагрузку.

На транспортном уровне возникают те же самые проблемы, что и на канальном, однако определить величину time-out здесь намного сложнее. Дело в том, что на транспортном уровне оценить время передачи через СПД намного сложнее, чем время передачи по каналу точка—точка между двумя маршрутизаторами (см. гл. 3). Если это время будет слишком большим, то снизится вероятность перегрузки, но снизится и производительность из-за длительного ожидания поступления пакета, а если оно будет малым, то появятся лишние пакеты.

2.3.4. Формирование трафика

Одной из основных причин перегрузки является нерегулярный взрывообразный трафик. Если бы он был равномерным, то перегрузок можно было бы избежать. Одним из методов, часто используемых для предотвращения перегрузки особенно в АТМ-сетях, является *формирование трафика*, т. е. придание ему определенной формы [19, 36], при которой скорость передачи пакетов регулируется.

Формированием трафика регулируется средняя скорость передачи данных, чем и сглаживается его взрывообразная форма. Следует обратить внимание на то, что протокол скользящего окна, который мы рассма-

тривали при изучении канального уровня, лишь регулирует объем данных, передаваемых за один раз, но не скорость передачи. Здесь же речь идет именно о скорости передачи. Когда виртуальное соединение устанавливается, то пользователь договаривается с транспортной средой о форме трафика. Если пользователь обеспечивает договоренную форму трафика, то транспортная среда обеспечивает ему доставку трафика с определенной скоростью. Для таких приложений, как передача видео- и аудиоданных в реальном времени, это очень важно. Здесь уместно вспомнить организацию работы СПД Frame Relay.

Когда пользователь и транспортная среда договариваются о форме трафика, то они приходят также к соглашению и о том, что произойдет, если эта форма будет нарушена пользователем, т.е. к так называемому *соглашению о трафике*. Технику формирования трафика и соглашение о трафике легче реализовать при использовании виртуальных соединений, чем при использовании дейтаграмм. В случае использования дейтаграмм эти идеи могут быть применены к соединениям на транспортном уровне.

Алгоритм текущего ведра

Идею алгоритма текущего ведра иллюстрирует рис. 2.15, *а* [19]: ведро может наполняться с любой скоростью, но вытекать из него вода будет со строго определенной скоростью, зависящей только от размера отверстия в днище. Если вода будет поступать слишком быстро, то часть ее будет переливаться через край и пропадать.

Этот прием можно применить и к пакетам (рис. 2.15, *б*). Каждая станция, подключенная к сети, имеет в своем интерфейсе буфер, по-

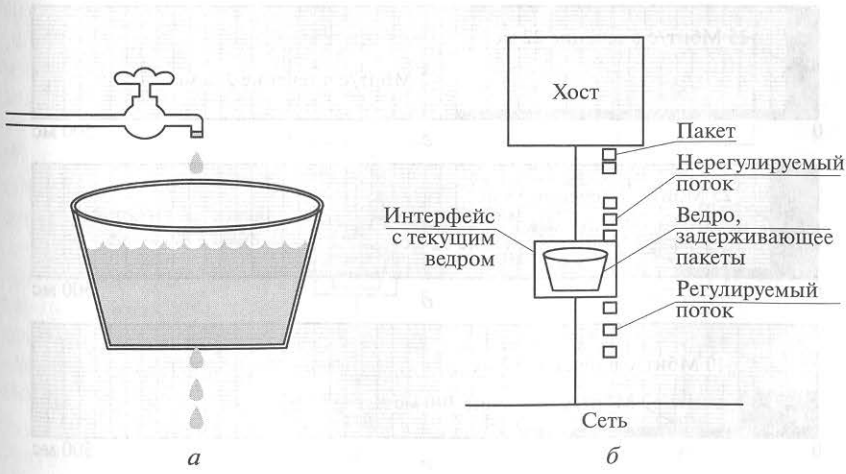


Рис. 2.15. Иллюстрация идеи алгоритма текущего ведра (*а*) и ее применение к пакетам (*б*)

добный текущему ведру. Неважно, сколько процессов посылают пакеты в сеть, если буфер переполнен, то они будут сбрасываться в соответствии с соглашением о трафике.

В качестве регулятора скорости поступления пакетов можно использовать системные часы. При этом устанавливается предел числа пакетов, которые процесс может направить в сеть за один определенный промежуток времени. Этот прием дает хорошие результаты, когда пакеты имеют фиксированную длину, как в СПД АТМ [36]. В случае если пакеты имеют переменную длину, это соглашение ограничивает число байтов, направляемых в сеть. Например, если разрешается

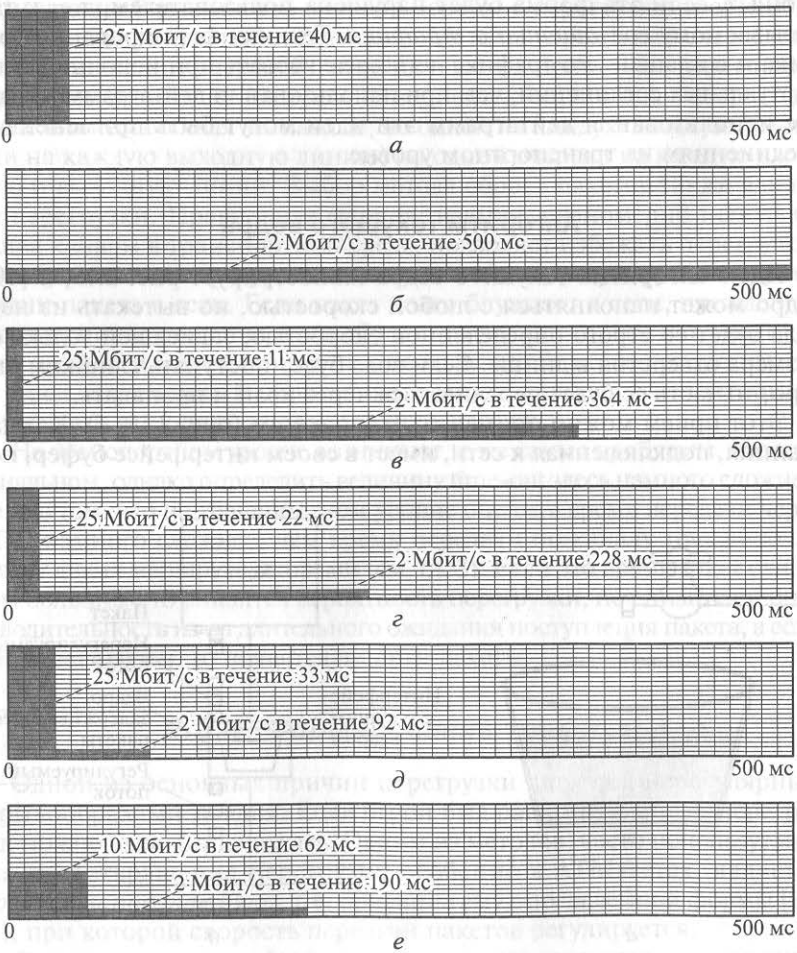


Рис. 2.16. Примеры работы алгоритма текущего ведра при разных скоростях входного потока (*а*...*е*)

за один промежуток времени послать 2 048 байт данных, то это может быть два пакета по 1 024 байт или 4 пакета по 512 байт. Если же пакет больше чем 2 048 байт, например 2 560 байт, то он должен ждать следующего временно́го промежутка.

На рис. 2.16 показан пример использования алгоритма текущего ведра со счетчиком байтов. Если имеется буфер *C* размером 1 Мбит, а скорость передачи равна 2 Мбит/с, то при всплеске размером 1 Мбит в течение 40 мс (рис. 2.16, *а*) система легко справится с таким трафиком. При этих условиях даже неважно, с какой скоростью будет поступать этот 1 Мбит данных, главное, чтобы этот всплеск трафика не растянулся более чем на 500 мс. На рис. 2.16, *в*...*е* показана работа алгоритма текущего ведра при разных скоростях входного потока.

Алгоритм ведра с маркерами

Алгоритм текущего ведра позволяет сгладить трафик, убрать нерегулярность передачи. Однако в целом ряде приложений бывает полезно при наличии всплеска трафика и необходимых ресурсов разрешить ускорить на некоторое время передачу пакетов в сеть. Одним из алгоритмов, позволяющих это сделать, является алгоритм ведра с маркерами, который иллюстрирует рис. 2.17 [19, 59].

Идея этого алгоритма заключается в том, что вместе с пакетами в ведро поступают маркеры (рис. 2.17, *а*) и пакеты уходят из ведра в сеть только при наличии соответствующего числа маркеров (рис. 2.17, *б*). Таким образом можно накапливать маркеры и кратковременно ускорять передачу пакетов в сеть.

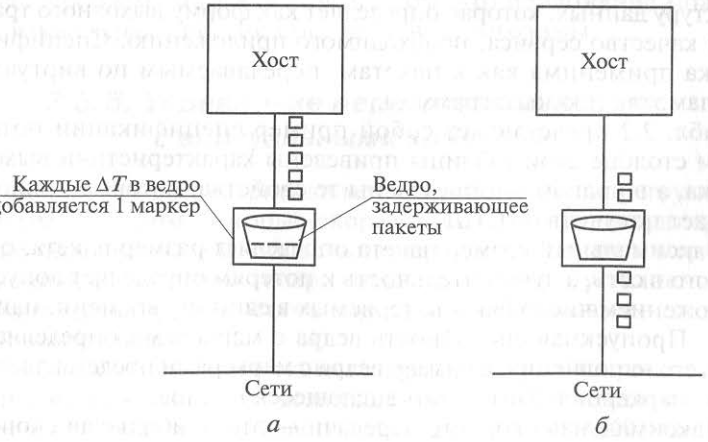


Рис. 2.17. Алгоритм ведра с маркерами:
а — поступление маркеров; *б* — истечение маркеров

Особенность алгоритма ведра с маркером заключается в том, что при переполнении буфера маршрутизатору будет временно запрещено передавать пакеты. Существуют разные варианты этого алгоритма, зависящие от длины пакетов, правил работы со счетчиком маркеров и т. д.

Для реализации алгоритма ведра с маркерами необходима лишь переменная, значение которой увеличивается каждые ΔT секунд и уменьшается с каждым посланным пакетом. В случае использования пакетов с нефиксированной длиной значение этой переменной увеличивается на k байт каждые ΔT секунд и уменьшается на длину каждого посланного пакета. Нетрудно рассчитать длительность всплеска трафика при передаче, осуществляемой на основе уравнения

$$C + \rho S = MS,$$

где C — объем буфера; ρ — скорость поступления маркеров; S — длительность всплеска трафика; M — максимальная скорость «вытекания» данных.

Таким образом, $S = C/(M - \rho)$.

Примеры, приведенные на рис. 2.16, *г*...*е*, иллюстрируют эту функцию для случая, когда буфер $C = 250$ Кбит, $M = 25$ Мбит/с и $\rho = 2$ Мбит/с.

2.3.5. Спецификация потока

Формирование трафика эффективно, если отправитель, получатель и среда передачи заранее договорились о его форме. Такое соглашение, называемое *спецификацией потока*, представляет собой структуру данных, которая определяет как форму выходного трафика, так и качество сервиса, необходимого приложению. Спецификация потока применима как к пакетам, передаваемым по виртуальным каналам, так и к дейтаграммам.

Табл. 2.2 представляет собой пример спецификации потока. В левом столбце этой таблицы приведены характеристики выходного потока, а в правом — определены те свойства, которые приложение ожидает получить от СПД.

Максимальный размер пакета определяет размер пакета, отправляемого в сеть, а чувствительность к потерям определяет допустимое приложением число байтов, теряемых в единицу времени, например в час. Пропускная способность ведра с маркерами определяет скорость его наполнения, а размер ведра с маркерами представляет собой число маркеров в байтах, помещающееся в ведре.

Максимальная скорость передачи — это наибольшая скорость, с которой машина может передавать данные.

Минимально допустимый интервал между потерями — параметр, важный, например при передаче видеоданных. Чувствительность к

Таблица 2.2

Пример спецификации потока

Характеристики выходного потока	Требуемый сервис
Максимальный размер пакета, байт	Чувствительность к потерям, байт
Пропускная способность ведра с маркерами байт/с	Интервал между потерями, мкс
Размер ведра с маркерами, байт	Чувствительность к потерям пакетов, пак.
Максимальная скорость передачи, байт/с	Минимальная задержка, мкс; максимальное отклонение в задержке, мкс; гарантии качества

потерям пакетов характеризуется максимально допустимым для приложения числом потерянных пакетов. Минимальная задержка — это минимальный интервал времени, на которое может опоздать пакет, чтобы для приложения это было незаметно. Например, минимальная задержка пакета при передаче файла и минимальная задержка пакета при передаче видеопотока — разные величины. Максимальное отклонение в задержке — это параметр, необходимый для тех приложений, в которых не так важно абсолютное значение задержки, как ее отклонение от среднего значения, т.е. равномерность. Наконец, гарантия качества указывает на важность перечисленных требований для приложения. Например, если необходимо установить связь с приложением где-то во Владивостоке, то ясно, что требования к качеству будут ниже, чем при соединении с приложением где-то в Москве. Проблема может состоять в том, что приложение может не знать, какие характеристики потока ему необходимы.

2.3.6. Управление перегрузками в сетях с виртуальными каналами

До сих пор мы рассматривали методы управления перегрузками с открытым контуром, которые скорее стараются предотвратить появление перегрузок, чем, обнаружив перегрузку, принять меры к ее устранению. Рассмотрим теперь только один метод устранения уже возникшей перегрузки в сетях с виртуальными каналами — динамический контроль доступа.

Первый прием динамического контроля доступа, широко используемый для сдерживания уже возникшей перегрузки и предотвращения ухудшения положения, называется контролем на входе [49]. Идея этого приема очень проста: если обнаружена перегрузка, то все, что способствует увеличению трафика, запрещено. Прежде всего, запрещается создание новых виртуальных соединений, а значит, и создание новых

соединений на транспортном уровне. Этот метод хотя и грубоват, но прост в реализации и хорошо апробирован в телефонных сетях.

Второй прием разрешает установку новых виртуальных соединений, но только в обход перегруженных маршрутов, даже если такие маршруты далеко неоптимальны.

Третий прием уже упоминался — формирование трафика, при котором абонентская машина и транспортная среда договариваются перед установкой виртуального соединения о форме трафика, объеме передаваемых данных, качестве сервиса и т. п. После этого транспортная среда резервирует количество ресурсов, необходимых ей для выполнения этих соглашений. Такое резервирование может происходить постоянно или только при возникновении перегрузок. Плата за резервирование — неоптимальное использование пропускной способности СПД.

2.3.7. Подавляющие пакеты

Теперь рассмотрим методы управления перегрузками, используемые как в средах с виртуальными каналами, так и в средах с дейтаграммами. Эти методы, как правило, связаны с контролем нагрузки на выходе [49]. Каждый маршрутизатор может контролировать степень загрузки своих выходных линий и другие ресурсы, и поэтому всякий раз, когда при очередном вычислении загрузка оказывается выше некоторого порогового значения, соответствующая линия переводится в состояние предупреждения. Каждый пакет, маршрутизируемый через такую линию, вызывает генерацию подавляющего пакета, направляемого отправителю маршрутизируемого пакета. При этом в пакете отправителя проставляется определенный разряд, предотвращающий генерацию в дальнейшем подавляющих пакетов другими маршрутизаторами.

Когда отправитель получает подавляющий пакет, он сокращает интенсивность своего трафика на определенное значение, которое является параметром метода. Поскольку пакеты, направляемые одному и тому же получателю, могут маршрутизироваться по-разному, отправитель вправе ожидать несколько подавляющих пакетов. В течение определенного интервала времени отправитель будет игнорировать подавляющие пакеты, поступающие с направления получателя. По истечении этого интервала времени отправитель ожидает появления подавляющих пакетов в течение следующего интервала времени, и если появляется хоть один подавляющий пакет, значит, линия перегружена и отправитель опять ждет. Если в течение очередного интервала времени не поступило ни одного подавляющего пакета, отправитель может увеличить интенсивность трафика.

Существует много вариантов приведенного алгоритма [19, 49]. Так, например, можно использовать не только загруженность линии, но и длину очереди, наполненность буфера и параметры спецификации трафика.

У методов управления перегрузками на основе подавляющих пакетов есть один недостаток. Если имеется несколько отправителей, работающих через одну и ту же выходную линию, то при определенных условиях маршрутизатор всем им пошлет подавляющие пакеты. Однако так как отправители независимы, то один, например, может сократить трафик значительно, а все другие — незначительно. Это приведет к несправедливому использованию пропускной способности канала между ними.

Для предотвращения такой ситуации был предложен алгоритм справедливого чередования, суть которого состоит в том, что для каждого отправителя у выходной линии строится своя очередь. Отправка пакетов из этих очередей происходит по кругу, поэтому если кто-то из отправителей незначительно сократит трафик, это лишь увеличит скорость роста его очереди.

Однако и у этого алгоритма имеется недостаток: если один отправитель использует длинные пакеты, а другой короткие, то последний получает меньшую долю пропускной способности линии. Для борьбы с этой несправедливостью алгоритм обслуживания очередей модифицируется: пакеты из очередей передаются побайтно, а не весь пакет сразу.

Другие модификации алгоритма чередования связаны с установкой приоритетов между очередями, что обеспечивает бóльшую гибкость в обслуживании отправителей. Так, если имеются очереди сервера и клиента, то, естественно, очередь сервера обслуживается быстрее.

Представленные алгоритмы с подавляющими пакетами плохо работают в высокоскоростных сетях и на больших расстояниях. Дело в том, что пока подавляющий пакет дойдет до отправителя, пройдет много времени и отправитель успеет «напихать» в сеть много пакетов. Для исправления этой ситуации был предложен алгоритм с подавлением по скачкам.

Суть этого алгоритма состоит в том, что как только обнаружится перегрузка на выходной линии и маршрутизатор отправит подавляющий пакет, то ближайший маршрутизатор, получивший этот подавляющий пакет, сократит трафик к маршрутизатору, пославшему подавляющий пакет. То значение, на которое он сократит трафик, является параметром метода, т.е. таким образом скачок за скачком трафик будет быстро падать. Естественно, этот прием увеличит нагрузку на буферы маршрутизаторов, сокращающих трафик, но обеспечит быструю реакцию на возникающую перегрузку.

2.3.8. Сброс нагрузки

Когда ни один из уже рассмотренных методов управления перегрузками не срабатывает, маршрутизатор может применить «тяжелую артиллерию» — сброс нагрузки. При этом было бы слишком примитивно предполагать, что маршрутизатор при возникновении пере-

грузки просто начинает сбрасывать пакеты. Идея метода и его название пришли из области передачи электроэнергии: отключение отдельных групп потребителей при возникновении перегрузок в электросети в целях сохранения работоспособности системы.

Маршрутизатор может сбрасывать пакеты по-разному, например случайным образом. Однако лучше, если он будет делать это исходя из информации о приложении, пославшем эти пакеты. Например, при передаче файла старые пакеты, т. е. пакеты, расположенные ближе к концу файла, лучше не трогать, поскольку приложение может потребовать повторить все пакеты начиная с пропущенного. В случае если вы сбросили пакет с номером 11, но переслали пакеты с номерами 15 и 16, может оказаться, что позднее пакеты 15 и 16 придется пересылать еще раз, при этом чем старее пакет, тем он ценнее. Однако следует иметь в виду, что ценность пакетов может быть относительно небольшой, пока не будет передана половина файла, а затем она начнет нарастать и, возможно, нелинейно.

В некоторых приложениях наоборот: лучше сбрасывать старые пакеты, чем новые. Примером являются видео- и мультимедиа данные. Например, при передаче изображений в целях компрессии сначала посылают всю картинку, а потом лишь ее изменения. Ясно, что при этом потеря одного из изменений лишь ухудшит изображение на некоторое время, а потеря картинки будет означать потерю изображения. Следовательно, сброс первых пакетов крайне нежелателен.

В общем случае метод управления перегрузками на основе сброса нагрузки предполагает определенное взаимодействие между приложением и маршрутизатором. Для обеспечения такого взаимодействия с приложением вводятся приоритеты среди пакетов. Это позволяет маршрутизатору минимизировать потери для приложения, когда маршрутизатор вынужден сбрасывать пакеты.

Для того чтобы приложение не злоупотребляло приоритетными пакетами, можно увязать приоритет пакетов с оплатой трафика: чем больше приоритетных пакетов, тем выше стоимость передачи. Приоритеты можно использовать также при формировании трафика. Например, при использовании алгоритма ведра с маркерами, если пакет пришел, а маркеров нет, можно применить прием, при котором пакет все же будет передан, но с низким приоритетом.

Приоритеты используются, например, в протоколах Frame Relay: пока трафик находится в заранее оговоренных пределах все пакеты идут с определенным приоритетом. При пиковых нагрузках, превосходящих номинальное значение, приоритеты пакетов начинают падать в зависимости от времени: чем дольше длится пиковая нагрузка, тем ниже становятся приоритеты пакетов, ее создающих. Если увеличение размера трафика продолжается недопустимо долго, то вскоре приоритеты пакетов достигнут минимума, и при первых же признаках перегрузки их начинают сбрасывать.

2.3.9. Управление перегрузками при групповой передаче

Все алгоритмы управления перегрузками, которые рассматривались до сих пор, относились к случаю одного источника и одного получателя. Рассмотрим теперь случай управления нагрузкой при групповой передаче при наличии нескольких источников и нескольких получателей. Например, в системе кабельного телевидения может быть несколько источников телепередач (станций вещания), и зрители могут по желанию переключаться с одной станции на другую. Аналогичная ситуация может иметь место в системе видеоконференций, когда слушатели могут переключаться с одной конференции на другую.

Во многих подобных приложениях группы могут возникать и изменяться по составу динамически. В этих условиях метод, при котором источник сообщений резервирует заранее необходимые для передачи ресурсы, неэффективен, поскольку источнику сообщений придется при каждом изменении группы генерировать дерево связей заново. Следовательно, для кабельного телевидения, где группы содержат миллионы зрителей, такой метод не годится к применению.

Одним из возможных решений для подобных случаев стал предложенный в 1993 г. RSVP-протокол (Resource reSerVation Protocol — протокол резервирования ресурсов), описанный в RFC 2205 [27]. Этот протокол позволяет: нескольким отправителям передавать сообщения группам получателей; отдельным получателям переходить из группы в группу; оптимизировать использование пропускной способности каналов, избегая перегрузок.

В простейшей форме RSVP-протокол для групповой маршрутизации использует дерево связей так же, как описывалось ранее. Каждой группе узлов приписана группа адресов. При отправке пакета отправитель помещает в него весь список адресов группы, после чего стандартный алгоритм групповой маршрутизации строит дерево связей, покрывающее все адреса группы. Собственно маршрутизация не является частью RSVP, его основное назначение — резервирование ресурсов для обеспечения заданного качества соединения [59]. Этот протокол основан на понятии soft-state, предложенном Кларком [31]. Это понятие предполагает специальный способ управления установкой time-out, при котором один раз установленное значение (состояние) остается неизменным до тех пор, пока оно подтверждается периодическими сообщениями от получателя (end-system) либо пока не будет им изменено.

Чтобы понять, как работает алгоритм RSVP, рассмотрим следующий пример [19]. На рис. 2.18, *а* показана сеть, в которой 1 и 2 — групповые отправители, 3...5 — групповые получатели, а буквами обозначены узлы трафика. Для упрощения множество отправителей

и получателей здесь не пересекаются. Дерево связей для групповой рассылки от отправителя 1 дано на рис. 2.18, *б*, а от отправителя 2 — на рис. 2.18, *в*.

Чтобы избежать перегрузок, любой получатель в группе шлет надлежащему отправителю резервирующее сообщение. Это сообщение с помощью алгоритма пересылки вдоль обратного пути, рассмотренного в подразд. 2.2.10, движется к отправителю и вызывает резервирование необходимой пропускной способности на каждом промежуточном маршрутизаторе (в форме соответствующего значения timeout), через который оно проходит. Если при прохождении очередного узла сообщению не удается зарезервировать необходимую пропускную способность, то получателю направляется отказ в установлении соединения.

На рис. 2.19, *а* показан путь резервирования между получателем 3 и отправителем 1. Как только канал между ними установлен, получатель 3 может получать поток пакетов от отправителя 1.

Рассмотрим теперь, что произойдет, если получатель 3 захочет также получать данные от отправителя 2. Для этого резервирующее сообщение проложит второй путь, показанный на рис. 2.19, *б*.

Предположим теперь, что получатель 5 также захочет подключиться к отправителю 2 (рис. 2.19, *в*) и запустит резервирующее сообщение, которое свободно пройдет до узла *H*, а там будет обнаружено, что ранее уже были зарезервированы ресурсы для доставки трафика от отправителя 2 до узла *H*. Здесь возможны нюансы. Например, если требования к качеству сервиса у получателей 5 и 3 разные, то резер-

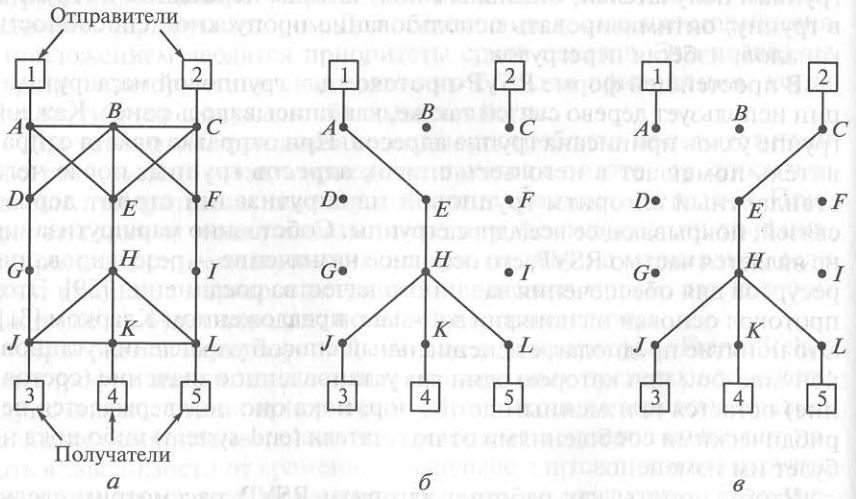


Рис. 2.18. Групповая передача:

а — пример сети; *б*, *в* — деревья связей для групповой рассылки соответственно от отправителя 1 и отправителя 2

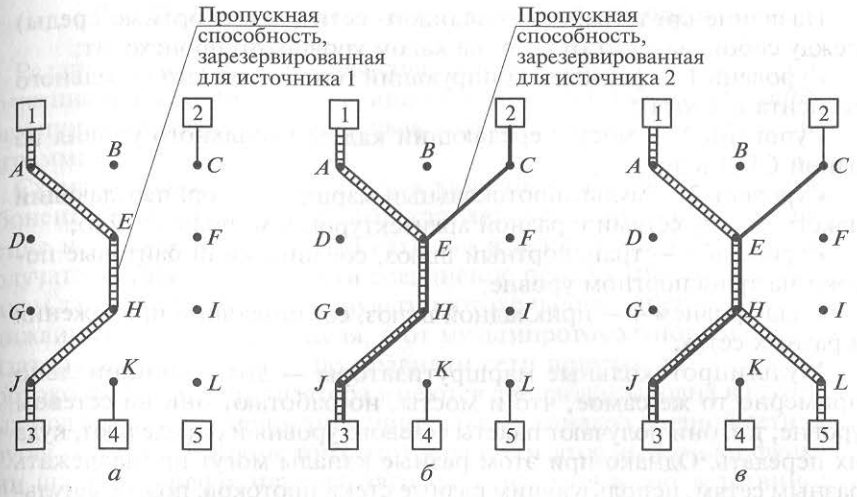


Рис. 2.19. Управление перегрузками при групповой передаче:
а...*в* — пути резервирования

вироваться ресурсы будут по максимуму. Кроме того, при резервировании получатель может указывать не только несколько источников, от которых он хотел бы получать информацию, но также является ли создаваемый канал временным (вплоть до того, на какое время он создается) или он должен долго оставаться открытым. Маршрутизаторы могут использовать эту информацию для оптимизации планирования ресурсов, особенно при разделении каких-либо ресурсов между несколькими получателями. Благодаря такой стратегии протокол RSVP может поддерживать динамику внутри групп.

2.4. Межсетевое взаимодействие

2.4.1. Общие сведения

До сих пор мы предполагали, что соединения возникают в рамках единой однородной транспортной среды. Теперь перейдем к рассмотрению случаев, когда соединение возникает между разными транспортными средами с разной архитектурой.

Существование разных сетей — явление объективное, отражающее объективные тенденции. Кроме архитектуры TCP/IP существует архитектура SNA от IBM, NCP/IPX от Novel, AppleTalk и т. д. Безусловно, будут появляться новые технологии, стандарты, устройства и программное обеспечение, их реализующее. Все эти инновации надо будет как-то связывать с существующими сетями.

Название средства, соединяющего сети (транспортные среды) между собой, зависит от того, на каком уровне это происходит:

- уровень 1 — репитер, копирующий биты из одного кабельного сегмента в другой;
- уровень 2 — мост, передающий кадры канального уровня из одной СПД в другую;
- уровень 3 — мультипротокольный маршрутизатор, передающий пакеты между сетями с разной архитектурой (см. подразд. 2.6);
- уровень 4 — транспортный шлюз, соединяющий байтовые потоки на транспортном уровне;
- над уровнем 4 — прикладной шлюз, соединяющий приложения в разных сетях.

Мультипротокольные маршрутизаторы — это функционально примерно то же самое, что и мосты, но работают они на сетевом уровне, т.е. они получают пакеты сетевого уровня и определяют, куда их передать. Однако при этом разные каналы могут принадлежать разным сетям, использующим разные стеки протоколов, поэтому мультипротокольному маршрутизатору, кроме задачи маршрутизации, приходится решать и задачу сопряжения форматов пакетов на сетевом уровне в сетях с разной архитектурой.

2.4.2. Чем различаются сети

Межсетевое взаимодействие усложняют различия между сетями. Когда пакет, отправленный из одной сети, должен пройти несколько разных сетей прежде чем достичь требуемую сеть, приходится решать множество различных проблем, возникающих на интерфейсах между сетями.

Например, если пакеты были отправлены по виртуальному соединению, а одна из транзитных сетей не поддерживает такие соединения, то порядок следования пакетов может быть нарушен, к чему ни отправитель, ни получатель могут быть не готовы. Также может потребоваться согласование сервисов, поддерживающих протоколы на одноименных уровнях. Например, если протокол в одной сети поддерживает понятие приоритета пакета, а одноименный протокол в другой сети не поддерживает, то такое согласование может оказаться невозможным. Проблемы могут возникнуть с адресацией при переходе из одной сети в другую, с групповой рассылкой и т.д.

Много проблем создает различие максимальной длины пакетов, используемой в разных сетях, а также различие качества услуг в разных сетях. Например, когда надо передать пакет из сети, поддерживающей ограничение на время передачи пакета, через сеть, где нет никаких гарантий на максимальное значение задержки пакета при передаче.

В разных сетях по-разному решаются вопросы управления перегрузками и потоками, обнаружения и исправления ошибок, что тоже может служить источником проблем.

2.4.3. Сопряжение виртуальных каналов

Различают два общих случая сопряжения транспортных сред: сопряжение транспортных сред с виртуальными каналами и сопряжение транспортных сред, не поддерживающих соединение, т.е. через дейтаграммы.

Схема сопряжения виртуальных каналов примерно следующая. Абонентская машина одной сети устанавливает виртуальное соединение не только внутри своей сети, но и в другой сети, вплоть до получателя. Внутри своей сети соединение прокладывается по правилам данной сети вплоть до мультипротокольного маршрутизатора, ближайшего к сети получателя, а от мультипротокольного маршрутизатора до получателя — по правилам сети получателя. У мультипротокольного маршрутизатора имеются две таблицы, одна из которых поддерживает именование виртуальных каналов в одной сети, а другая — в другой. Кроме того, этот маршрутизатор выполняет функции шлюза, переформатируя заголовки пакетов в соответствии с архитектурой той транспортной среды, в которую направляется пакет. Эта схема хорошо работает для сетей с примерно одинаковыми характеристиками.

Достоинства сопряжения виртуальных каналов следующие: буферы можно резервировать заранее, порядок пакетов сохраняется, проще управлять повторной передачей при time-out, короткие заголовки пакетов.

Недостатки сопряжения виртуальных каналов: потребность в дополнительной памяти для хранения таблицы сопряжения, сложность изменения маршрута при перегрузках, требование высокой надежности маршрутизаторов вдоль сопряжения.

2.4.4. Межсетевое взаимодействие без соединений

При соединении транспортных сред через дейтаграммы единственный сервис, который сетевой уровень предоставляет транспортному, — «впрыскивание» дейтаграмм в транспортную среду. Дальше приходится надеяться на удачу. Такое сопряжение сетей возможно, если соединяемые транспортные среды используют одни и те же или очень близкие сетевые протоколы. Вспомним проблемы мостов между СПД в стандартах 802.x (см. т. 1, подразд. 4.5.1 данного учебника).

Проблемы возникают и с адресацией. Различия в адресации могут быть столь велики, что сопряжение станет невозможным. Например, в протоколе IP используется 32-разрядный адрес, а в модели OSI — десятичный номер, подобный телефонному. Первый выход из ситуации — распространение каждой адресации на все машины в мире. Однако очевидно, что это не работает. Другой выход из ситуации — создание универсального пакета, который понимали бы разные сети, — и он тоже не работает. Как известно, существуют пакеты в

формате архитектуры TCP/IP, в формате архитектуры IPX и т.д. Всех уговорить признать один формат как универсальный невозможно.

Основное достоинство дейтаграммного подхода к сопряжению транспортных сред: он не требует, чтобы все объединяемые сети были или дейтаграммными, или с виртуальными каналами.

2.4.5. Туннелирование

В общем случае проблема межсетевого соединения весьма сложная. Однако существует один весьма распространенный прием: это соединение двух одинаковых сетей через промежуточную СПД, например так, как показано на рис. 2.20.

Решение проблемы межсетевого соединения в этом случае обеспечивает применение техники *туннелирования*. Поясним ее на примере автомобиля. Поскольку не разрешается автомобилям своим ходом передвигаться по туннелю под Ла-Маншем из соображений безопасности, их погружают на платформу скоростного поезда. На другом конце туннеля автомобиль с платформы съезжает на шоссе и далее движется своим ходом.

Применительно к сетям суть этой техники состоит в том, что пакет из одной сети упаковывается в кадр промежуточной СПД и передается через нее. При достижении сети назначения кадр распаковывается, пакет передается на сетевой уровень и движется дальше.

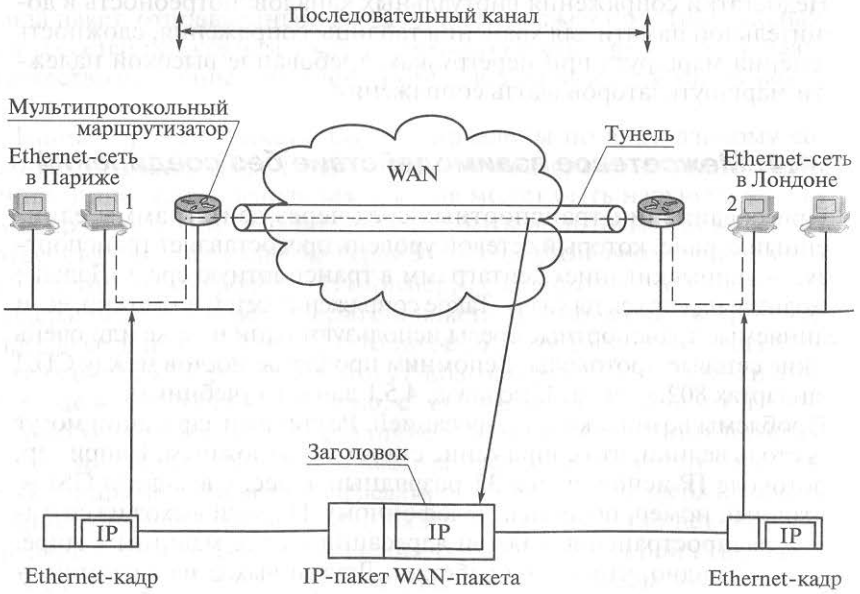


Рис. 2.20. Схема транспортирования пакета методом туннелирования

2.4.6. Межсетевая маршрутизация

Маршрутизация на межсетевом уровне происходит примерно так же, как на сетевом, но с некоторыми дополнительными сложностями. Рассмотрим пример, приведенный на рис. 2.21, где шесть мультипротокольных маршрутизаторов ($A \ldots F$) соединяют пять сетей ($1 \ldots 5$). Имея граф соединений этих маршрутизаторов между собой, можно применять уже известные алгоритмы маршрутизации: по вектору расстояния или по состоянию канала. Таким образом мы придем к двум уровням маршрутизации: внутреннему межшлюзовому протоколу и внешнему межшлюзовому протоколу. Поскольку каждая транспортная среда в определенном смысле автономна, то для нее часто используется термин *автономная система*.

Главная сложность, отличающая внутрисетевую маршрутизацию от межсетевой, — государственные границы. Здесь возникают различия в законах разных стран, различия в оплате трафика, принятой на территориях разных стран, вопросы национальной безопасности и т. д.

(Напомним, что термины *сеть* и *транспортная среда* мы сейчас не различаем, они для нас эквивалентны.)

2.4.7. Фрагментация

В каждой транспортной среде существует свой максимальный размер пакетов. Это ограничение определяется:

- аппаратурой (например, максимальный TDM-слот);
- операционной системой (все буфера по 512 байт);
- протоколами (например, размером поля длины пакета);
- совместимостью с некоторыми национальными и международными стандартами;

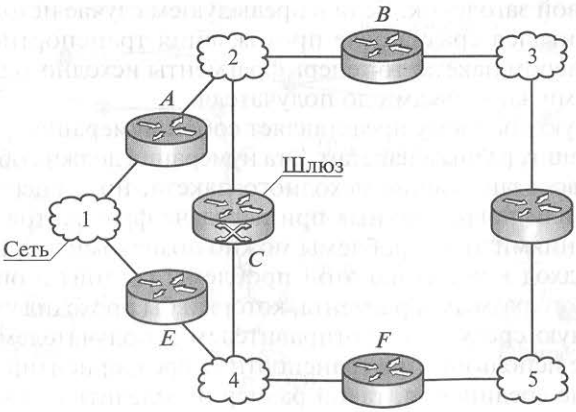


Рис. 2.21. Пример межсетевой маршрутизации

• стремлением сократить вероятность ошибки при передаче, вызывающей повторную передачу;

• желанием предотвратить длительный захват канала одним пакетом.

Максимальный размер пакета колеблется от 48 байт в ATM-сети до 65 515 байт в IP-сети (у протоколов верхних уровней он еще больше).

Очевидно, что при попытке передать большой пакет через сеть, у которой максимальный размер пакета меньше, возникает первая проблема. Одно из решений данной проблемы — это проложить маршрут для таких пакетов таким образом, чтобы избежать подобной ситуации. Однако при этом встает вопрос, что делать, если именно в такой сети расположен получатель. Единственное решение здесь — разрешить шлюзу разбивать пакет на фрагменты и отправлять каждый фрагмент независимо. Однако при этом возникает проблема сборки фрагментов.

Существует два подхода к решению этой проблемы. Первый подход — делать фрагменты столь малыми, чтобы любая сеть на их пути была для них прозрачна. Когда большой пакет поступает, его разбивают на малые пакеты, которые отправляют на один и тот же выходной шлюз, где они снова собираются в большой пакет. При такой фрагментации приходится решать следующие вопросы: как узнать, что все фрагменты достигли выходного шлюза; как выбирать маршрут для фрагментов; как сократить накладные расходы на разбиение и сборку пакета из фрагментов.

Другой подход — разбить пакет на фрагменты и рассматривать каждый из них как независимый отдельный пакет. При этом сборка фрагментов происходит только в узле назначения. Однако при таком подходе, во-первых, каждый маршрутизатор должен уметь собирать пакеты из фрагментов, а во-вторых, резко возрастают накладные расходы на передачу, так как каждый фрагментированный пакет будет нести свой заголовок. Если в предыдущем случае исходный пакет восстанавливался сразу после прохождения транспортной среды с малым размером пакета, то теперь фрагменты исходного пакета пойдут со своими заголовками до получателя.

Отдельную проблему представляет собой нумерация фрагментов, принадлежащих разным пакетам. Эта нумерация должна обеспечивать не только восстановление исходного пакета, но и восстановление утерянных или поврежденных при передаче фрагментов. С некоторыми решениями этой проблемы можно познакомиться в [49]. Наилучший подход к решению этой проблемы состоит в определении минимального размера фрагмента, который бы проходил через любую транспортную среду между отправителем и получателем. Заметим, что в случае использования транспортных сред, ориентированных на виртуальные соединения, такой размер определить несложно, а вот в случае использования дейтаграмм — это действительно проблема. При этом если какая-то промежуточная транспортная среда пропу-

скает фрагменты размера, большего, чем минимальный, то в пакете такой транспортной среды можно объединить несколько минимальных фрагментов.

2.5. Сетевой уровень в Интернете

2.5.1. Общие сведения

Интернет на сетевом уровне можно рассматривать как объединение транспортных сред или сетей, которые называются автономными системами.

Автономная система — это сеть, охватывающая единую территорию, находящаяся под единым административным управлением и имеющая единую систему правил маршрутизации (политику маршрутизации) по отношению ко всем остальным сетям. В Интернете нет какой-либо регулярной специально предусмотренной структуры сетей. Это соединение большого числа сетей, среди которых можно выделить несколько магистральных (backbone). К этим магистральным сетям подключены региональные сети, к которым, в свою очередь, подключены локальные сети организаций. На рис. 2.22 показан пример такого соединения сетей.

Все автономные системы взаимодействуют через IP-протокол. В отличие от других протоколов сетевого уровня IP-протокол с самого начала создавался для объединения сетей. Его целью было наилучшим образом передавать дейтаграммы от одной машины к другой, где бы эти машины ни находились.

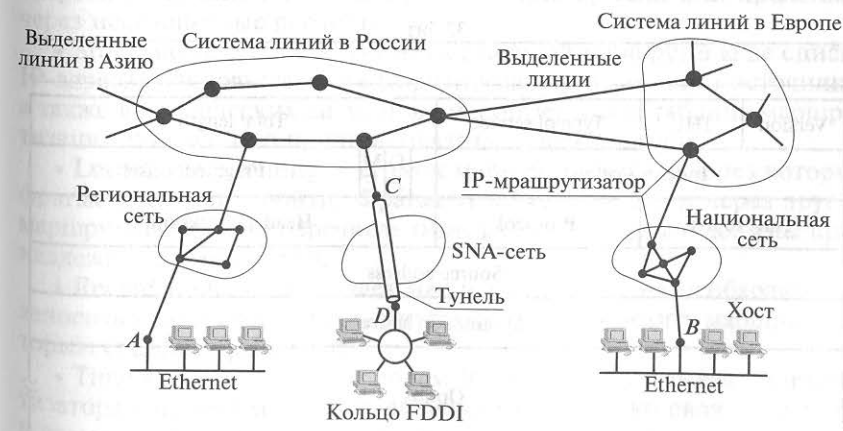


Рис. 2.22. Интернет как сеть, состоящая из множества сетей:
A... *D* — точки подключения

Как уже отмечалось, сетевой уровень в Интернете реализует сервис без соединений и работает следующим образом. Транспортный уровень получает поток данных и делит их на дейтаграммы. Дейтаграммы, которые могут содержать от 1,5 до 64 Кбайт, передаются через сети в Интернет и, если необходимо, делятся на более короткие. Когда все дейтаграммы достигают места назначения, они собираются в исходные дейтаграммы на сетевом уровне и передаются на транспортный уровень, где восстанавливается исходный поток данных.

2.5.2. IP-протокол

В настоящее время в Интернете действуют две версии IP-протокола: IPv4 и IPv6. Начнем рассмотрение с более ранней версии IPv4, а затем рассмотрим и IPv6, поэтому если противное не оговорено, то сейчас под IP-протоколом мы будем понимать IPv4.

На рис. 2.23 показан заголовок IP-пакета (дейтаграммы), имеющий обязательную часть размером 20 байт, которая может быть расширена до 60 байт. Дейтаграмму передают начиная с поля Version. Назначение полей заголовка следующее:

Version — указывает версию IP-протокола;

IHL — длина заголовка, который может содержать от 5 до 60 32-разрядных слов;

Type of service — вид необходимого сервиса. Здесь можно указать различные комбинации скорости и надежности (например, передача голоса, аккуратная доставка строки битов, файла и т. п.), а также задать приоритет дейтаграммы;

Total length — указывает общую длину пакета, включая заголовок и поле данных. Максимальная длина составляет 65 535 байт;

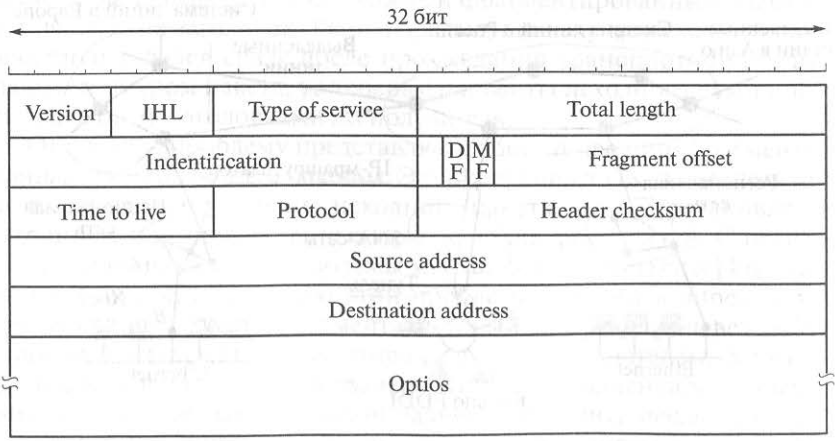


Рис. 2.23. Заголовок IP-пакета

Identification — позволяет отличать фрагменты одного и того же пакета, т. е. указывает, какому пакету принадлежит очередной поступивший фрагмент. Все фрагменты одного и того же пакета имеют в этом поле одно и то же значение;

DF — признак управления фрагментацией, причем если он равен 1, то фрагментация невозможна;

MF — содержит единицу у всех фрагментов пакета кроме последнего. Это поле позволяет отличить последний фрагмент от всех остальных;

Fragment offset — указывает, где в пакете располагается данный фрагмент пакета. Длина всех фрагментов, кроме последнего, должна быть кратна 8 байт. Поскольку поле имеет 13 разрядов, то у одного пакета максимально может быть 8 192 фрагментов. Таким образом, длина минимального фрагмента равна 8 байт;

Time to live — время жизни пакета. Максимальное значение этого поля составляет 255 с. Очень часто здесь используется счетчик скачков;

Protocol — показывает, какому протоколу на транспортном уровне передать собранную дейтаграмму (TCP, UDP и т. д.);

Header checksum — контрольная сумма, охватывающая только заголовок;

Source address, Destination address — идентифицируют машину отправителя и получателя на сетевом уровне;

Options — необязательная часть заголовка, предусмотренная для расширения возможностей протокола, которая может иметь следующие поля:

- Security — указывает уровень секретности передаваемой информации. Маршрутизатор может на основании значения этого поля запретить определенные маршруты, например если они пролегают через небезопасные регионы;
- Strict source routing — указывает полный маршрут в виде списка IP-адресов. Используется в алгоритме маршрутизации от источника, а также в критических ситуациях, например когда таблица маршрутизации по какой-то причине оказалась испорченной;
- Loose source routing — список маршрутизаторов, через которые фрагмент обязан пройти. Фрагмент может пройти и через другие маршрутизаторы, но перечисленные здесь обязательно должны принадлежать его маршруту;
- Record route — указывает маршрутизаторам на необходимость заносить в поле свои адреса. Это позволяет проследить маршрут, которым следовал фрагмент;
- Time stamp — вместе с полем Record route указывает маршрутизаторам на необходимость записывать не только свои адреса, но и время, когда фрагмент проходил через них. Это поле очень полезно при отладке алгоритмов маршрутизации.

Более подробное описание заголовка IP-протокола см. в [14].

2.5.3. IP-адресация

Каждая машина в Интернете имеет уникальный IP-адрес, состоящий из адреса сети и адреса машины в этой сети. Все IP-адреса имеют длину 32 разряда. На рис. 2.24 показаны форматы IP-адресов. Если машина подключена к нескольким сетям, то в каждой сети у нее будут свой IP-адрес и своя сетевая карта.

Все адреса на сетевом уровне в TCP/IP подразделяются на классы. Всего существует пять классов адресов: A, B, C, D, E. В классе A адрес сети занимает 7 бит, под адрес машины в сети выделяется 24 бит. Таким образом, класс A позволяет адресовать до 126 сетей по 16 млн машин в каждой. В классе B под адрес сети выделено 14 разрядов, а под адрес машины в сети — 16. Тем самым класс B позволяет адресовать 16 382 сетей по 64 000 машин в каждой. В классе C под адрес сети отдано 19 разрядов, а под адрес машины — 8, т. е. 2 млн сетей по 256 машин. Адреса класса D предназначены для групповой передачи, а класса E — зарезервированы для развития.

Адреса выделяет только организация Internet Network Information Center (www.internic.net).

На рис. 2.25 показаны IP-адреса, имеющие специальное назначение. IP-адрес с нулевым адресом сети используют для адресации

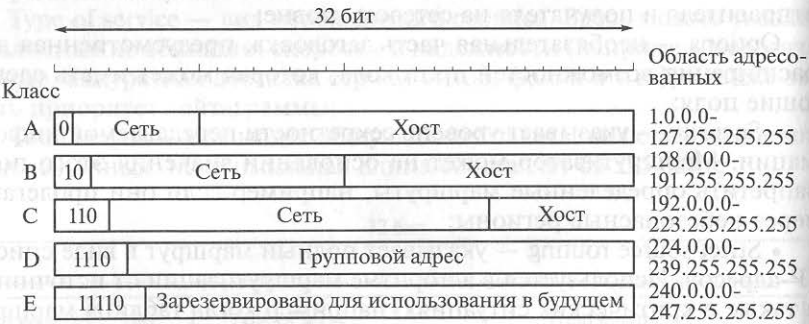


Рис. 2.24. Форматы IP-адресов

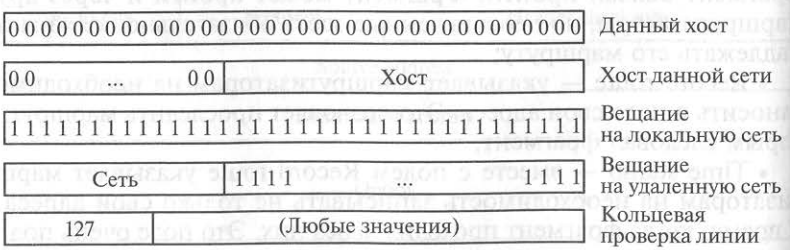


Рис. 2.25. Специальные IP-адреса

машин одной и той же сети. При этом необходимо соблюдать класс сети и соответственно число нулей в поле адреса сети. Адрес из одних нулей соответствует машине, на которой располагается программное обеспечение, используемое при загрузке машины, обратившейся по этому адресу. Адрес из одних единиц относится ко всем машинам в данной сети. Если единицы проставлены только в поле адреса машины, то такой адрес обозначает все хосты* в сети, адрес которой указан в поле адреса сети. Ну, и, наконец, адрес с номером сети 127 используется для тестирования сетевого программного обеспечения, когда сетевой интерфейс хоста заворачивает пакет с таким адресом себе же на вход. Адрес IP записывают в виде четырех десятичных чисел, разделенных точками.

2.5.4. Подсети

Все машины, принадлежащие одной и той же сети, должны иметь одинаковый номер сети в своем адресе. Это приводит к целому ряду проблем. По мере роста сети организации приходится изменять класс сети. Появление новых адресов приводит к проблеме повсеместной модификации таблиц маршрутизации и распространению информации о новых адресах. Перенос машины из одной сети в другую может потребовать изменения маршрутизации. Эти изменения происходят не сразу, и пока они не выполнены, все сообщения идут по старому адресу.

Решением указанных проблем является разделение сети на части, но таким образом, чтобы извне она по-прежнему представляла собой единое целое, а внутри нее каждая часть имела бы свой адрес. Часть адреса, идентифицирующая определенную группу сетевых адресов транспортной среды, называется адресом подсети, а все хосты, имеющие одинаковый адрес подсети, образуют подсеть. Итак, подсеть — это часть транспортной среды, невидимая извне. Изменение адреса подсети или введение новой подсети не требует обращения в NIC или изменения какой-либо глобальной базы данных.

На рис. 2.26 показано разбиение сети класса B на подсети.

Чтобы понять, как используется адрес подсети, следует проследить, как маршрутизатор использует IP-адрес. У маршрутизатора есть две таблицы. В первой таблице содержатся записи вида «сеть — 0», а во второй — вида «эта_сеть, адрес машины»**. Первая таблица показывает, как достичь интересующей сети, а вторая — как достичь хоста внутри сети. Когда поступает IP-пакет, маршрутизатор ищет его адрес доставки в первой таблице маршрутизации. Если этот адрес является

---

* Под термином «хвост» далее будем понимать любую машину в сети, имеющую хотя бы один IP-адрес.

** Под словами «эта сеть» здесь подразумевается адрес сети, к которой принадлежит маршрутизатор.

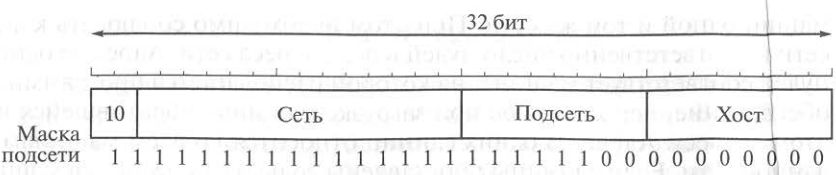


Рис. 2.26. Разбиение сети класса B на подсети

адресом другой сети, то пакет передается тому маршрутизатору, который отвечает за связь с указанной сетью. Если это адрес в сети данного маршрутизатора, то он, используя вторую таблицу, направляет пакет прямо по месту назначения. Если адреса нет во второй таблице, то маршрутизатор направляет пакет специально выделенному по умолчанию маршрутизатору, который должен разобраться с этим случаем с помощью более подробной таблицы.

Из приведенного описания видно, что алгоритм маршрутизации имеет дело только с сетями или локальными машинами, а не с парами сеть—машина. Такая организация алгоритма позволяет существенно сократить размер таблиц в маршрутизаторах.

С появлением подсети структура записей в таблице изменяется. Теперь записи в таблице имеют вид «эта_сеть, подсеть, 0» и «эта_сеть, эта_подсеть, машина». Таким образом, маршрутизатор подсети знает, как достичь любой подсети в данной локальной сети и как найти конкретную машину в своей подсети. Все что ему необходимо для этого — знать маску подсети. С помощью логической операции И и маски, показанной на рис. 2.26, маршрутизатор выделяет адрес подсети, а затем по своим таблицам определяет, как достичь требуемой подсети или (если это локальная подсеть для маршрутизатора) как достичь конкретной машины.

2.5.5. Протоколы управления межсетевым взаимодействием

Протокол ICMP

В Интернете кроме IP-протокола, который используется для передачи данных, имеется несколько протоколов управления, используемых сетевым уровнем, таких как ICMP, ARP, RARP, BOOTP. Рассмотрим последовательно эти протоколы.

Управление функционированием Интернета на сетевом уровне происходит через маршрутизаторы с помощью протокола ICMP (Internet Control Message Protocol), описанного в RFC 792. Этот протокол обеспечивает доставку сообщений любой машине, имеющей IP-адрес, от маршрутизаторов и других хостов в сети. С помощью этого протокола реализуют обратную связь для решения проблем, возни-

кающих при передаче. Он также выявляет и рассылает сообщения о десятках событий. Для доставки своих сообщений протокол ICMP использует пакеты IP-протокола. Приведем наиболее важные сообщения.

Сообщение **destination unreachable** охватывает множество случаев, например, случай, когда маршрутизатор не знает, как достигнуть необходимой подсети или хоста, или случай, когда дейтаграмма при доставке должна быть фрагментирована, но установлен флаг, который запрещает это делать.

Сообщение **time exceeded** посылает маршрутизатор, если он обнаружил дейтаграмму с истекшим времени жизни. Это сообщение также генерирует хост, если он не успел завершить обработку IP-пакета до истечения времени его жизни. (Далее слова «пакет», «IP-пакет», «дейтаграмма» будем понимать как синонимы, если специально не оговорено что-либо другое.)

Синтаксические или семантические ошибки в заголовке IP-пакета вызывают появление сообщения **parameter problem**.

Сообщение **source quench** обеспечивает управление потоком. Маршрутизатор или хост-получатель высылает этот пакет хосту-отправителю, если последнему необходимо понизить скорость передачи. Другими словами, это пример подавляющего пакета (см. подразд. 2.4). Сообщения такого типа будут генерироваться до тех пор, пока скорость поступления пакетов от отправителя не достигнет значения, необходимого хосту-получателю. Это сообщение система может использовать для предотвращения перегрузки, поскольку оно возникает всякий раз, когда маршрутизатор вынужден сбросить пакет из-за переполнения своего буфера.

Сообщение **redirect** позволяет маршрутизатору отправить рекомендацию о лучшем маршруте и впредь посылать пакеты с определенным IP-адресом через другой маршрутизатор.

Сообщения **echo request** и **echo reply** позволяют проверить работоспособность хостов в сети: получатель сообщения echo request обязан ответить сообщением echo reply, причем с теми же параметрами, что и в echo request.

Сообщения **time-stamp request** и **time-stamp reply** позволяют измерять временну́ю задержку в Интернете на сетевом уровне. Этот механизм необходим, например, для работы алгоритма маршрутизации по состоянию канала.

Протокол определения адреса — ARP

Хотя каждая машина в Интернете имеет уникальный IP-адрес, при передаче пакета через СПД от этого мало пользы, так как канальный уровень не понимает IP-адресов. Как правило, машина подключается к СПД через сетевую карту, которая понимает только адреса канального уровня, например Ethernet-адрес, имеющий 48 разрядов.

Сетевая карта знает только такие адреса и ничего не знает о 32-разрядных IP-адресах. Как отобразить 32-разрядный IP-адрес в адресах канального уровня, поясним с помощью рис. 2.27.

Когда машина 1 посылает сообщение машине 2, адрес доставки имеет вид smel@cs.msu.su. Сначала машина 1 с помощью службы имен домена — DNS (Domain Name Service — см. подразд. 4.2 определяет IP-адрес машины 2. Далее для отображения IP-адреса в Ethernet-адресе машина 1 посылает через СПД Ethernet следующий запрос: «У кого такой IP-адрес?». Этот запрос, также содержащий адрес СПД машины 1, увидят все машины в сети 192.31.65.0. Машина 2, увидев такой запрос и свой IP-адрес в нем, пошлет ответ, в котором указан ее СПД-адрес (в данном случае, Ethernet-адрес). Протокол, который реализует рассылку запросов и сбор ответов, т.е. протокол определения адреса — ARP (Address Resolution Protocol), описан в RFC 826. Практически у каждой машины в Интернете имеется этот протокол.

Теперь, зная СПД-адрес получателя, т.е. адрес требуемой машины в СПД (в данном случае, MAC-адрес), пакет с адресом 192.31.65.5 можно передать на канальный уровень. На канальном уровне полученный пакет помещают в Ethernet-кадр, в заголовке которого размещается найденный MAC-адрес машины 2. Машина 1 при этом запоминает MAC-адрес машины 2 на случай, если ей еще раз к ней придется обратиться. Ясно, что протокол должен учитывать время актуальности этой информации. Например, если на машине 2 заменят сетевую карту, то изменится и ее сетевой адрес. Поэтому данная информация должна периодически обновляться. Для того чтобы машине 2 не пришлось повторять всю процедуру поиска MAC-адреса машины 1, в заголовке кадра машина 1 укажет свой MAC-адрес, а в пакете — свой IP-адрес.

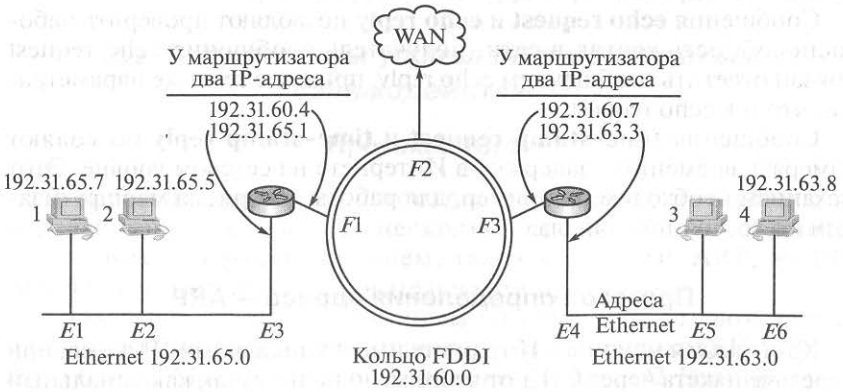


Рис. 2.27. Пример объединения сети класса C:
*E*1 ... *E*3; *F*1 ... *F*3 — адреса точек подключения

Теперь рассмотрим случай, когда обращение посылается в другую подсеть. Здесь возможны два решения. Первое решение: имеется определенный маршрутизатор, который принимает все сообщения, адресованные в определенную подсеть или группе адресов. Доступ в эту определенную подсеть возможен только через данный маршрутизатор, который называется proxy и на котором работает протокол proxy ARP. Этот маршрутизатор знает, как найти адресуемую машину в этой определенной подсети. Второе решение — использование выделенного маршрутизатора, который управляет маршрутизацией удаленного трафика. Машина-отправитель сама определяет, что обращение идет в удаленную сеть, и посылает сообщение на этот выделенный маршрутизатор. В нашем случае на маршрутизатор с MAC-адресом E3, который распаковывает кадр и находит искомый IP-адрес. По своим таблицам этот маршрутизатор определяет, что все пакеты для машин сети 192.31.60.0 следует пересылать через интерфейс с MAC-адресом F3. Маршрутизатор с MAC-адресом E3 посылает сообщение маршрутизатору F3 по кольцу FDDI. Если же он не знает MAC-адреса адреса F3, то посылает вещательный запрос с указанием искомого IP-адреса, на которые откликнется маршрутизатор с MAC-адресом F3.

Подробнее ARP-протокол рассмотрен в [8].

Обратный протокол определения адреса — RARP

Иногда возникает необходимость по известному СПД-адресу определить соответствующий IP-адрес, например при удаленной загрузке бездисковой станции. Как же эта станция определит свой IP-адрес и IP-адреса соседних машин?

В сети имеется определенный хост, на котором работает обратный протокол определения адреса — RARP (Reverse Address Resolution Protocol). Этот хост также называют RARP-сервером. При необходимости определить по Ethernet-адресу соответствующий IP-адрес посылают следующий запрос RARP-серверу: «Вот Ethernet-адрес, кто знает соответствующий IP-адрес?». RARP-сервер отлавливает такие запросы, а затем формирует и отсылает ответ. Происходит это следующим образом. RARP-сервер направляет всем машинам в сети запрос, где указан заданный Ethernet-адрес, и просит указать соответствующий ему IP-адрес. Машина с указанным СПД-адресом отправляет ответ, в котором указан ее IP-адрес. Получив ответ, RARP-сервер формирует ответ запросившему хосту [8].

У RARP-протокола имеется один существенный недостаток: пакеты с одним и тем же запросом рассылаются всем хостам, в том числе и маршрутизаторам, что увеличивает накладные расходы. Для устранения этого недостатка был предложен протокол BOOTP, который в отличие от RARP рассылает свои сообщения только маршрутизаторам. Этот протокол также используется в бездисковых станциях, у которых в памяти «зашит» IP-адрес выделенного маршрутизатора.

2.5.6. Внутренний протокол маршрутизации шлюзов — OSPF

Как уже отмечалось, Интернет состоит из сетей, управляемых разными организациями. Внутри каждой такой сети работают свои алгоритмы маршрутизации и управления. При этом сама сеть называется *автономной системой* (АС). Алгоритмы маршрутизации, применяемые внутри АС, называются *внутренними протоколами шлюзов*. Алгоритмы маршрутизации, применяемые для маршрутизации между АС, называются *внешними протоколами шлюзов*. Наличие стандартов на протоколы маршрутизации позволяет преодолеть различия во внутренней организации автономных систем и обеспечить их совместное функционирование.

Изначально в качестве внутреннего протокола шлюзов использовался протокол по вектору расстояния — RIP (см. подразд. 2.2.6). Этот протокол работал хорошо, пока автономная система была небольшой. Однако по мере роста АС он стал работать все хуже и хуже. Проблемы «бесконечного счетчика» и медленной сходимости не получили удовлетворительного решения. В 1979 г. он был замещен протоколом маршрутизации по состоянию каналов. В 1988 г. инженерный комитет Интернета принял решение о разработке нового протокола маршрутизации: внутреннего протокола маршрутизации шлюзов — OSPF (Open Shortest Path First), который стал стандартом в 1990 г. (RFC 1247).

Требования к протоколу OSPF

На основе имеющегося опыта был составлен длинный список требований к протоколу OSPF.

1. Алгоритм должен быть опубликован в открытой литературе (отсюда «open»).

2. Алгоритм не должен быть собственностью какой-либо компании.

3. Алгоритм должен уметь работать с разными метриками маршрутов (расстоянием, пропускной способностью, задержкой и т. п.) и при этом быть динамическим, т. е. реагировать на изменение топологии сети автоматически и быстро.

4. Алгоритм должен поддерживать разные виды сервиса, а также маршрутизацию для трафика в реальном времени одним способом, а для других типов трафика — другим. Для этого в IP-пакете есть поле Type of service (см. рис. 2.23), которое не использовалось существующими в то время протоколами.

5. Алгоритм должен обеспечивать балансировку нагрузки во избежание перегрузки и при необходимости направлять потоки по разным каналам (все предыдущие протоколы использовали только один канал — наилучший).

6. Алгоритм должен поддерживать иерархическую маршрутизацию. К 1988 г. Интернет стал столь большим, что ни один маршрутизатор был уже не в состоянии хранить всю топологию. Поэтому новый протокол должен был быть сконструирован таким образом, чтобы по нему мог бы работать любой маршрутизатор на любом уровне иерархии.

7. В алгоритме должна быть усилена безопасность маршрутизаторов для обеспечения защиты от злоумышленников и просто любопытствующей публики, например студентов. Поскольку первые за счет изменения таблиц маршрутизации добивались изменения маршрутизации далеко не в безобидных целях, а вторые развлекались тем, что подсовывали маршрутизаторам неверную информацию о маршрутах.

8. В алгоритме должна быть обеспечена возможность маршрутизаторам общаться с помощью туннелирования.

Виды соединений в OSPF

Протокол OSPF поддерживает три вида СПД.

1. Каналы типа точка—точка между двумя маршрутизаторами.

2. СПД на основе каналов с множественным доступом и вещанием (большинство ЛВС).

3. СПД на основе коммутации каналов или коммутации пакетов (например, региональные сети с коммутацией пакетов).

На рис. 2.28, *а* показаны эти три вида сетей. При этом протокол OSPF абстрагируется от конкретных сетей и строит их модель в форме ориентированного графа, каждая дуга в котором имеет вес, представляющий собой определенную метрику канала: задержку, расстояние и т. п. В этом графе наикратчайший путь определяется на основе весов дуг. Последовательный дуплексный канал между узлами изображается двумя дугами, которые могут иметь разный вес. СПД с множественным доступом представляются звездообразным графом, в котором центральный узел соединен дугами, имеющими нулевой вес с другими узлами (рис. 2.28, *б*).

Многие АС сами по себе представляют большие сети. OSPF позволяет разбивать их на области, где каждая область — это либо сеть, либо последовательность сетей. Эти области не пересекаются. Область — это обобщение понятия подсети. Извне топология области не видна.

Каждая АС имеет *магистральную область*, называемую областью 0. Все области одной АС соединяются через магистральную область, возможно с помощью туннелирования. Туннель представляется в графе взвешенной дугой. Любой маршрутизатор, соединенный с двумя или более областями, является частью магистральной области. Топология магистральной области так же, как и топологии других областей, не видна извне.

Внутри области у всех маршрутизаторов одинаковая база данных состояний каналов, и все они используют одинаковый алгоритм нахождения наилучшего пути. Задача маршрутизатора — вычислить наилучший путь до другого маршрутизатора этой области, включая маршрутизатор, соединенный с магистральной областью. Маршрутизатор, соединенный с двумя областями, должен иметь две базы данных и два алгоритма поиска наилучшего пути.

Для поддержания разных типов сервисов OSPF использует три графа: первый с разметкой, где веса представляют собой задержку, второй — пропускную способность каналов, третий — надежность каналов. Хотя все три графа и требуют соответствующих вычислений, но зато при этом получаем три маршрута, оптимизированных по задержке, пропускной способности и надежности.

Во время работы сети возникают три вида маршрутов: внутри области, между областями и между АС (рис. 2.29). Внутри области вычислить маршрут просто — это наикратчайший путь до маршрутизатора получателя. Маршрутизация между областями всегда выполняется в три этапа: от источника до магистральной области, от магистральной области до области назначения и внутри области на-

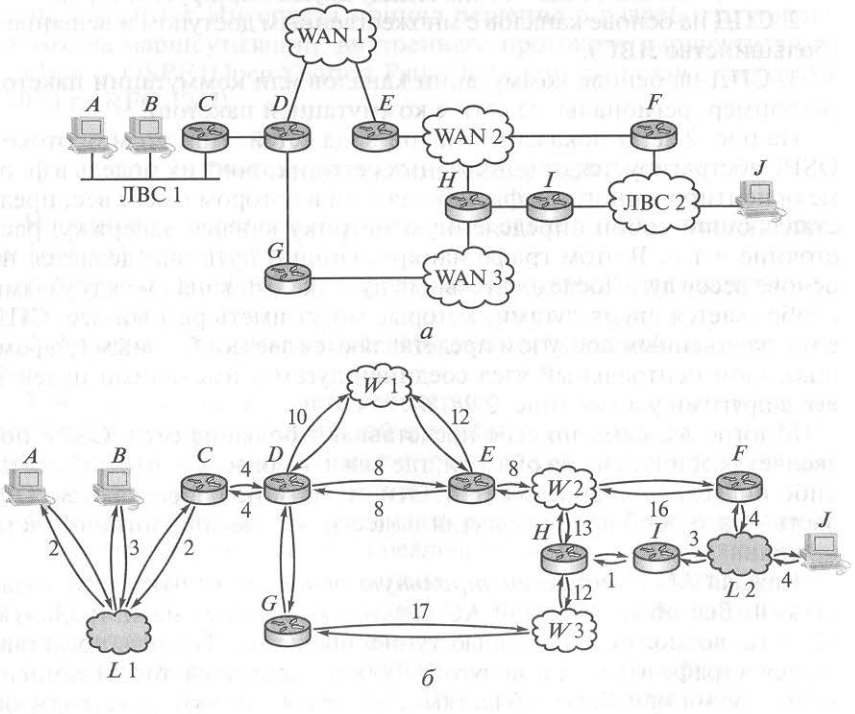


Рис. 2.28. Виды сетей, поддерживаемые в OSPF (*а*), и их представление в виде графа (*б*)

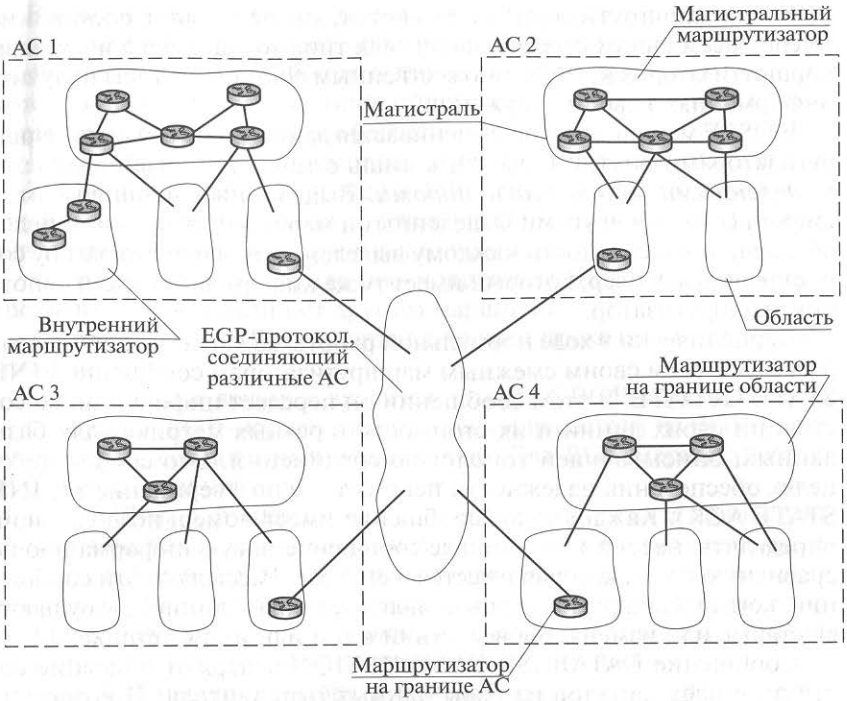


Рис. 2.29. Примеры маршрутов и маршрутизаторов различных классов в OSPF

значения. Этот алгоритм навязывает звездообразную топологию АС: магистральная область — это центр (ось), остальные области — лучи (спицы). Пакеты маршрутизируются без изменений за исключением случая, когда область получателя соединена с магистральной областью туннелем.

Маршрутизаторы в OSPF

В OSPF различают четыре класса маршрутизаторов.

1. Внутренние, находящийся целиком внутри одной области.
2. Пограничные, соединяющие несколько областей.
3. Магистральные, принадлежащие магистральной области.
4. Пограничные, соединенные с маршрутизаторами других АС.

Маршрутизаторы разных классов могут пересекаться. Например, все пограничные маршрутизаторы являются магистральными, а маршрутизатор из магистральной области, но не находящийся на ее границе — внутренним. Примеры разных классов маршрутизаторов показаны на рис. 2.29.

Когда маршрутизатор загружается, он рассылает сообщение «Hello» всем своим соседям на линиях типа точка—точка и группам маршрутизаторов в ЛВС с множественным доступом, чтобы получить информацию о своем окружении.

В OSPF маршрутизатор обменивается данными не со всеми маршрутизаторами внутри области, а лишь с теми, которые объявлены *выделенными маршрутизаторами*. Выделенный маршрутизатор смежен со всеми другими выделенными маршрутизаторами. В целях обеспечения надежности каждому выделенному маршрутизатору сопоставляется дублер, который имеет ту же информацию, что и основной маршрутизатор.

Периодически в ходе нормальной работы каждый маршрутизатор рассылает всем своим смежным маршрутизаторам сообщение LINK STATE UPDATE. В этом сообщении он передает информацию о состоянии своих линий и их стоимости в разных метриках для базы данных, описывающей топологию соединений. Это сообщение в целях обеспечения надежности передается с подтверждением (LINK STATE ACK). Каждое такое сообщение имеет номер, позволяющий определить, несет ли пришедшее сообщение новую информацию по сравнению с той, которая имеется в его базе. Рассылают эти сообщения, когда у маршрутизаторов появляются новые линии, разрушаются старые или изменяется вес линии в той или иной метрике.

Сообщение DATABASE DESCRIPTION содержит описание состояния всех каналов из базы данных отправителя. Получатель, сравнивая свои значения с теми, которые имеются у отправителя, может определить, у кого из них наиболее свежая информация.

Используя сообщение LINK STATE REQUEST, маршрутизатор может в любой момент запросить информацию о любой линии у другого маршрутизатора. Все сообщения передаются как IP-пакеты.

В магистральной области маршрутизаторы выполняют все указанное ранее, а также обмениваются информацией с пограничными маршрутизаторами в целях обеспечения вычисления наилучшего маршрута от любого маршрутизатора магистральной области до любого другого маршрутизатора.

2.5.7. Протокол пограничных шлюзов BGP

Для маршрутизации пакетов между АС используется протокол пограничных шлюзов — BGP (Border Gateway Protocol), описанный в RFC 1624. Его предшественником был протокол EGP. Однако с ростом Интернета он перестал удовлетворять возросшие требования, и был заменен. Основное отличие BGP от OSPF определяется различием целей маршрутизации между АС и внутри АС. При маршрутизации внутри АС основная цель — это выбор наилучшего маршрута, а при маршрутизации между АС необходимо учитывать также ряд условий, вызванных политикой конкретной АС.

Например, АС может не допускать маршрутизацию пакетов через себя транзитом ни из какой другой АС (то, что у вас из-за этого не будет другого наикратчайшего пути, — это ваши проблемы), а также может разрешать транзит лишь определенным АС. Типичные примеры таких ограничений следующие:

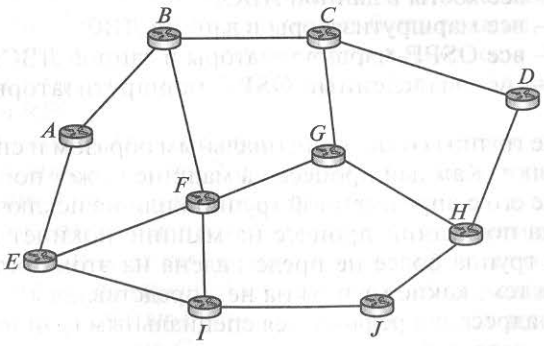
1. Трафик не должен проходить через определенные АС.
2. Маршрут, начинающийся в Министерстве обороны России, никогда не должен проходить через территорию военных действий или территории недружественных государств.
3. Трафик через территорию недружественной страны может проходить, только если нет другого маршрута.
4. Трафик из АС IBM никогда не должен проходить через АС Microsoft.

Такие правила вручную вводятся в каждый BGP-маршрутизатор.

С позиции BGP-маршрутизатора весь мир состоит из BGP-маршрутизаторов, соединенных между собой. Два BGP-маршрутизатора соединены, если у них есть общая сеть.

Для BGP-маршрутизатора все сети делятся на три категории: *тупиковые сети*, которые никуда не ведут и не могут использоваться для транзита пакетов через сеть; *сети с множественными соединениями*, которые могут использоваться для транзита, если его допускает политика определенной АС; *транзитные сети*, предназначенные для транзита трафика, возможно с некоторыми ограничениями.

Два BGP-маршрутизатора взаимодействуют через транспортное TCP-соединение (см. гл. 3), что обеспечивает надежность передачи информации и скрывает все подробности от сетей, через которые она проходит.



F получает информацию о *D* от своих соседей:

B сообщает: «Я использую *BCD*»;
G сообщает: «Я использую *GCD*»;
I сообщает: «Я использую *IFGCD*»;
E сообщает: «Я использую *EFGCD*»

Рис. 2.30. Транспортная сеть из BGP-маршрутизаторов

BGP — это протокол на основе алгоритма вектора расстояний. Однако вместо стоимости для каждого места в сети он хранит конкретный маршрут, а своим соседям передает не вектор расстояний, а те маршруты, которые он использует (рис. 2.30).

BGP-протокол легко решает проблему «бесконечного счетчика». Предположим, что маршрутизатор *G* или линия *FG* отказали. Тогда *F* получит от своих соседей три оставшихся маршрута до *D*. Поскольку маршруты *IFGCD* и *EFGCD* проходят через *F*, то он их отбросит и воспользуется маршрутом *FBCD*.

2.5.8. Групповая адресация в Интернете

Обычно в IP-сетях один отправитель взаимодействует с одним получателем. Однако в ряде приложений бывает полезным одно и то же сообщение передавать сразу нескольким получателям, например при поддержке обновления данных в реплицируемых базах данных, передаче биржевой информации, поддержке телеконференций.

В IP-сетях групповая адресация поддерживается с помощью адресов класса D, в которых имеется 28 разрядов для адресации группы, т. е. можно адресовать 250 млн групп. При передаче сообщения группе делается все возможное, чтобы каждый член группы получил сообщение, однако это не гарантируется.

Поддерживаются два типа групповых адресов: постоянные и временные.

Примеры постоянных групповых адресов:

224.0.0.1 — все хосты в данной ЛВС;

224.0.0.2 — все маршрутизаторы в данной ЛВС;

224.0.0.5 — все OSPF-маршрутизаторы в данной ЛВС;

224.0.0.6 — все выделенные OSPF-маршрутизаторы в данной ЛВС.

Временные группы создают специальным образом и специальным образом удаляют. Каждый процесс на машине может послать запрос на соединение его с определенной группой или на исключение его из группы. Когда последний процесс на машине покинет группу, это означает, что группа более не представлена на этом хосте. Каждый хост следит за тем, какие группы на нем представлены.

Групповая адресация реализуется специальным групповым маршрутизатором, который может размещаться отдельно от обычного маршрутизатора. Раз в минуту каждый групповой маршрутизатор посылает через канальный уровень запрос всем машинам в ЛВС с требованием указать, каким группам принадлежат их процессы. Эти запросы и ответы на них регулируются IGMP-протоколом (Internet Group Management Protocol), который очень похож на протокол ICMP, описанный в RFC 1112.

2.5.9. Бесклассовая маршрутизация внутри домена

Популярность Интернета обернулась против него: не стало хватать адресов. В 1987 г. считалось, что 100 000 сетей — это очень много и что это число будет достигнуто не скоро, однако оно было превышено в 1996 г. Проблема состоит в том, что адреса выделяются классами, при этом многие организации не используют всего диапазона адресов выделенного им класса. Например, наиболее часто используемый класс B, слишком велик для многих организаций.

На основе накопленного опыта стало ясно, что было бы неплохо, если класс C имел не 256 машин, а 1 024.

Проблемой также является взрывообразный рост таблиц маршрутизации. Маршрутизатор не должен знать о каждой машине в сети, но должен знать о каждой сети. Уже к середине 1990-х гг. было выделено полмиллиона адресов класса C, следовательно, в таблице маршрутизации должно быть не менее полумиллиона элементов, каждый из которых показывает, как достичь той или иной сети.

Кроме того, многие из алгоритмов маршрутизации требуют, чтобы маршрутизаторы периодически обменивались своими таблицами. При этом чем больше эти таблицы, тем больше шансов, что при передаче они будут повреждены и переданы неверно.

Выходом из этой ситуации является увеличение иерархии адресов в Интернете, т.е. указывать страну, область, город, район, машину. Однако 32 бит при этом не хватит. Кроме того, Лихтенштейн, например, будет иметь столько же адресов, сколько и Россия.

Таким образом, каждое решение несет свои проблемы. Маршрутизаторы рассматривают IP-адресную среду на двух уровнях: адреса сети и адреса ЭВМ, при этом практически они работают только с адресами сетей. Данная проблема может быть решена, если забыть про разбиение всей совокупности IP-адресов на классы. Такая модель реализуется в рамках протокола CIDR (Classless InterDomain Routing), описанного в RFC 1519.

Идея протокола CIDR основана на том, что сети класса C почти не использовались, а их более 2 млн. Поэтому по запросу организации можно выделить несколько последовательных сетей этого класса таким образом, чтобы охватить требуемое число машин. Например, если организация подает заявку на 2 000 машин, можно выделить ей восемь последовательных сетей класса C, т.е. 2 048 машин. При этом мир был поделен на четыре зоны, и каждой из этих зон была выделена часть адресов класса C:

194.0.0.0 ... 195.255.255.255 — Европа;
198.0.0.0 ... 199.255.255.255 — Северная Америка;
200.0.0.0 ... 201.255.255.255 — Центральная и Южная Америка;
202.0.0.0 ... 203.255.255.255 — Азия и Тихий Океан.

Таким образом, каждая зона получила 32 млн адресов для раздачи пользователям, а 320 млн адресов класса C (с 204.0.0.0. по 223.255.255.255)

были зарезервированы на будущее. Это существенно упростило работу с таблицами маршрутизации. Например, любой маршрутизатор, получив адрес в диапазоне 194.0.0.0...195.255.255.255, знает, что его надо переслать одному из европейских маршрутизаторов. Этим приемом мы добавили еще один уровень в иерархию IP-адресов.

2.5.10. Протокол IPv6

Появление новой версии протокола IP, названного IPv6 (в настоящее время используется версия IPv4 и идет процесс перехода на IPv6), было обусловлено целым рядом причин. Одна из основных — стремительный рост сети Интернет. Фундаментальным принципом построения сетей на основе протокола IP, необходимого для правильной маршрутизации и доставки пакетов, является уникальность сетевых адресов, т.е. каждый IP-адрес может принадлежать только одному устройству. Отметим, что это теперь необязательно абонентская машина или маршрутизатор. Например, это может быть некоторое устройство в автомобиле, обеспечивающее доступ к приложениям Интернета. Учитывая, что уровень автомобилизации общества растет и что практически каждый, у кого есть автомашина, имеет компьютер или телевизор, который тоже может быть подключен к Интернету, проблема с количеством адресов становится очевидной. На сегодняшний день остались нераспределенными около 1 400 000 000 адресов из 4 294 967 296, т.е. примерно 30 %, которых должно хватить на несколько лет. Появившиеся в последнее время устройства для доступа в Интернет и развитие цифрового телевидения, которое собирается превратить каждый телевизор в интернет-устройство, могут быстро исчерпать имеющиеся запасы неиспользованных адресов.

В компьютерных сетях для выхода в Интернет могут применяться технологии типа NAT (Network Address Translation — преобразование сетевого адреса), в которых для выхода в Интернет имеется всего несколько уникальных IP-адресов, предоставляемых устройству временно по запросу. При этом внутри локальной сети адресация может быть достаточно произвольной. Для сетевого телевизора такой подход не годится, так как каждому устройству требуется свой уникальный адрес.

Кроме того, новые устройства предъявляют к протоколам сетевого уровня, например к IP-протоколу, совершенно новые требования в части легкости получения и смены адресов, а также полной автоматизации конфигурирования (представьте себе домохозяйку, настраивающую DNS своего телевизора). При этом было ясно, что если новый протокол не появится своевременно, то фирмы-провайдеры начнут внедрять свои собственные разработки, что может привести к невозможности обеспечения гарантированного соединения типа «всех со всеми». Была осознана также потребность в наличии открытого протокола, который должен удовлетворять следующие требова-

ния: размер адресного пространства, легкость конфигурирования и маршрутизации, способность работать совместно с имеющимся протоколом IPv4. Новый протокол должен был также сохранить способность к соединению между собой любых устройств, поддерживающих IP-протокол.

Опыт использования IPv4 позволил осознать следующее положение: уникальность адреса вовсе не означает, что устройство будет правильно функционировать. Адреса в первую очередь необходимы не для того чтобы «всех пересчитать», а для правильной маршрутизации при доставке пакетов. Таким образом, для беспрепятственного роста Интернета необходимо не только наличие свободных адресов, но и определенная методика их выделения, позволяющая решить проблему масштабируемости и в определенном смысле маршрутизации.

Сведение к минимуму накладных расходов на маршрутизацию является сегодня одной из основных проблем, и ее важность будет возрастать в дальнейшем по мере роста сети Интернет. Просто присвоить устройству адрес недостаточно, необходимо еще обеспечить условия для правильной маршрутизации с минимальными накладными расходами.

В настоящее время только иерархическая маршрутизация позволяет за счет приемлемых технических издержек обеспечить доставку пакетов в сети размером с Интернет. Технология иерархической маршрутизации, которую мы рассматривали подробно ранее, заключается в разбиении всей сети на более мелкие подсети, где маршрутизация производится самостоятельно. Подсети, в свою очередь, могут разбиваться на еще более мелкие подсети и т.д. В результате образуется древовидная структура, причем в качестве узлов в ней выступают маршрутизаторы, а в качестве листьев — оконечные устройства (хосты). Путь, который проделывает пакет, передаваемый от одного листа к другому, может быть длиннее, чем при иной организации адресации, но зато он всегда может быть рассчитан с наименьшими издержками.

История нового протокола начинается с конца 1992 г. Именно тогда IETF (Internet Engineering Task Force — рабочая группа по технической поддержке Интернета) приступила к анализу данных, необходимых для разработки нового протокола IP. К концу 1994 г. был утвержден рекомендательный стандарт и разработаны все необходимые для реализации протокола вспомогательные стандарты и документы.

IPv6 является новой версией старого протокола, разработанной таким образом, чтобы обеспечить совместимость и «мягкий» переход к ее использованию, не приуроченный к конкретной дате и не требующий одновременных действий всех участников. По некоторым прогнозам два протокола могут использоваться совместно лет десять и более. Учитывая, что среди выделенных типов адресов IPv6 имеется специальный эмулирующий адрес IPv4, можно ожидать относительно спокойный переход на новый протокол, не сопровождающий-

ся крупными неудобствами и неприятностями. Фактически на одном компьютере могут работать оба протокола, каждый из которых подключается по мере необходимости.

Однако использование старых адресов не является выходом из положения, поэтому протокол IPv6 предусматривает специальные возможности по присвоению новых адресов и их замене без вмешательства (или при минимальном вмешательстве) специально обученного персонала. Для этого предусмотрена привязка к компьютеру не IP-адреса, а интерфейса. Сам же интерфейс может иметь несколько адресов следующих трех категорий: действительный, прошлый, недействительный.

При замене адреса «на лету» новый адрес становится действительным, а тот, который был раньше, — прошлым. Все вновь осуществляемые соединения производятся с помощью действительного адреса, а уже имеющиеся соединения продолжаются маршрутизироваться по прошлому адресу. Через некоторое время, которое может быть выбрано достаточно большим, чтобы гарантировать полный разрыв всех соединений по прошлому адресу, этот адрес переходит в категорию недействительных. Тем самым обеспечивается автоматическая замена адреса без участия персонала. Для полностью гарантированной автоматической замены адреса потребовалось бы внесение изменений в протоколы TCP и UDP, которые не входят в состав IP-протокола.

Замена адресов осуществляется двумя способами: явным и неявным. Явный способ использует соответствующим образом доработанный протокол динамической конфигурации хоста — DHCP (Dynamic Host Configuration Protocol), который служит для установки конфигурационных параметров компьютеров, подключенных к Интернету [38,42, 93]. DHCP имеет два компонента: протокол установки конфигурационных параметров хоста и протокол выделения хосту сетевых адресов.

DHCP построен по схеме клиент—сервер, где DHCP-сервер динамически выделяет сетевые адреса и доставляет конфигурационные параметры надлежащим хостам. Протокол IP требует установки многих параметров, а так как он может быть использован самым разным сетевым оборудованием, значения этих параметров нельзя угадать заранее. Кроме того, схема распределенного присвоения адресов зависит от механизма выявления уже используемых адресов.

DHCP поддерживает три механизма выделения IP-адресов: при «автоматическом выделении» DHCP присваивает клиенту постоянный IP-адрес; при «динамическом выделении» DHCP присваивает клиенту IP-адрес на ограниченное время; при «ручном выделении» IP-адрес выделяется клиенту сетевым администратором, а DHCP используется просто для передачи адреса клиенту. Конкретная сеть может использовать какой-то один или несколько из этих механизмов в зависимости от политики сетевого администрирования.

Динамическое выделение адресов — это единственный механизм, позволяющий автоматически повторно использовать адрес, который уже стал не нужен одному клиенту как адрес для другого клиента.

Протокол IPv6 допускает также неявный способ выделения адреса, который не требует наличия сервера DHCP, а использует адрес подсети, получаемый от соседей и мостов. В качестве адреса хоста здесь используется просто MAC-адрес хоста, т. е. адрес, используемый на канальном уровне. Этот способ при всем своем изяществе по понятным причинам не может присваивать адреса, совместимые с IPv4, и поэтому в переходный период его применение будет ограничено.

Протокол IPv6 предполагает также значительные улучшения при работе в локальной сети. Единый протокол распознавания соседей — NDP (Neighbor Discovery Protocol) заменяет применяемые в IPv4 протоколы ARP, ICMP и имеет более широкие функциональные возможности. Вместо используемых в протоколе ARP широковещательных пакетов канального уровня в IPv6 применяются групповые сообщения (multicast), т. е. сообщения, адресованные всем членам подсети, притом не на канальном, а на сетевом уровне, что должно значительно снизить широковещательный трафик, являющийся «бичом» локальных сетей Ethernet.

В IPv6 усовершенствованы функции протокола ICMP, что облегчает работу разных подсетей в одном физическом сегменте, а также включен механизм распознавания неисправных маршрутизаторов, что позволяет повысить устойчивость к сбоям оборудования. В дополнение к имевшимся ранее двум типам адресации: Unicast и Multicast (доставке уникальному получателю и группе получателей), здесь добавлен тип адресации Anycast, при котором осуществляется доставка каждому получателю из группы.

Существенное отличие нового протокола от старого заключается в том, что длина его адресной части составляет 128 бит, что в четыре раза больше, чем 32 бит в IPv4. Чтобы представить это значение, достаточно сказать, что на каждом квадратном метре поверхности суши и моря можно разместить примерно $6{,}7 \times 10^{23}$ адресов. Из заголовка IP-пакета изъяты как неиспользуемые некоторые поля, что позволило сократить издержки, связанные с их обработкой, и уменьшить размер заголовка (который всего в два раза длиннее, чем в IPv4, несмотря на учетверенный размер адресной части).

На рис. 2.31 показана структура заголовка пакета в формате IPv6 (сравните с рис. 2.23).

Значение четырехбитового поля Version (Версия) равно 6.

Поле Priority (Приоритет) длиной 8 бит используется для установки приоритета пакета, который увеличивается с ростом значения этого поля. Значения 0...7 используются для пакетов, время доставки которых не лимитировано (значение 1 рекомендуется использовать для новостей, 2 — для почты, 7 — для служебного трафика, например трафика, генерируемого SNMP-протоколом (см. подразд. 4.2.2)).

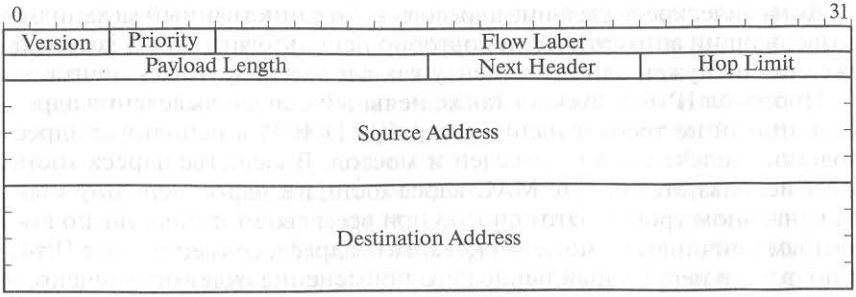


Рис. 2.31. Структура заголовка пакета в протоколе IPv6

Значения 8...15 используются для пакетов, задержка доставки которых нежелательна, например аудио- и видеоданных в реальном времени.

Поле Traffic Class, первоначально называвшееся Flow Label, имеет длину 20 бит. Оно служит для идентификации последовательности пакетов, его значение присваивается с помощью генератора случайных чисел и оно одинаковое у всех пакетов данной последовательности.

Поле Payload Length содержит размер данных, следующих за заголовком, в байтах и имеет длину 16 бит.

Поле Next Header идентично по назначению полю Protocol протокола IPv4. Это поле раскрывает секрет возможности использования упрощенного заголовка. Дело в том, что после обычного 40-байтового заголовка могут идти дополнительные (необязательные) заголовки. Это поле сообщает, какой из шести дополнительных заголовков (на текущий момент) следует за основным. В последнем IP-заголовке поле Next Header сообщает, какой обрабатывающей программе передать пакет.

Поле Hop Limit длиной 8 бит аналогично по назначению полю Time to Live, т. е. оно устанавливается источником согласно разумным предположениям о длине маршрута, а затем уменьшается на единицу при каждом прохождении через маршрутизатор. При снижении значения поля до нуля пакет снимается как «заблудившийся».

Последними в заголовке пакета идут поля адресов источника и приемника длиной 128 бит (16 байт) каждое.

Адреса в стандарте IPv6 имеют более сложную структуру, чем в протоколе IPv4, при этом в адресах используются префиксы разной длины (табл. 2.3).

Специальные типы адресов в IPv6 обеспечивают более гибкие, чем в IPv4, маршрутизацию и использование. Например, выделяемые провайдером уникальные адреса (Provider-Based Unicast Address) служат для глобальной связи. Структура таких адресов показана на рис. 2.32.

Таблица 2.3

Префиксы протокола IPv6

Назначение	Префикс
Зарезервировано	0000 0000
Зарезервировано для адресов NSAP	0000 001
Зарезервировано для адресов IPX	0000 010
Provider-Based Unicast Address	010
Зарезервировано для Neutral-Interconnect-Based Unicast Addresses	100
Link Local Use Addresses	1111 1110 10
Site Local Use Addresses	1111 1110 11
Multicast Addresses	1111 1111

Адрес содержит: префикс 010; поле Registry Id, идентифицирующее организацию, зарегистрировавшую провайдера; поле Provider Id, идентифицирующее провайдера; поле Subscriber Id, идентифицирующее организацию-клиента, и собственно адрес.

Адреса для локального использования (Link Local Use и Site Local Use), структуры которых показаны на рис. 2.33, предназначены для применения внутри сети одной организации, т. е. пакеты с такими адресами не маршрутизируются за границы ее сети. Эти пакеты могут быть использованы, например, при автоматическом присвоении адресов.

Для выхода в глобальную сеть может быть использована подстановка адресов с помощью технологии NAT. Если под заполнители-нули выделено достаточно места, то организация, ранее не имевшая соединения с Интернетом, может легко провести замену адресов на глобальные с помощью конкатенации Registry Id + Provider Id + Subscriber Id и локального адреса.

К специальным типам адресов в IPv6 также относятся адреса, совместимые с IPv4 (рис. 2.34): во-первых, адреса, предназначенные для туннелирования пакетов IPv6 через существующую инфраструктуру IPv4. и во-вторых, адреса, отображающие в IPv6 подмножество адресов IPv4 для тех устройств, которые не поддерживают новый протокол.

Широковещательный адрес (Multicast Address) благодаря полям Flags и Scope (рис. 2.35) может также использоваться более гибко. В четырехбитовом поле Flags пока используется только младший бит для указания того, что данный адрес является постоянным и выделенным соответствующими организациями, ответственными за выдачу адресов, или он используется единовременно. Поле Scope используется для ограничения области распространения широковещательных пакетов. Значения этого поля приведены в табл. 2.4.

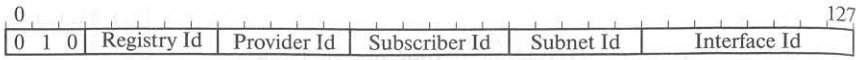


Рис. 2.32. Структура адреса абонента, основанного на адресе провайдера

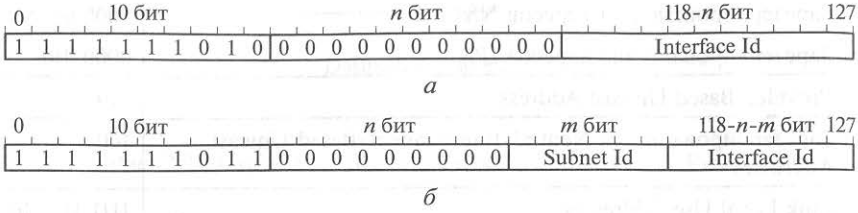


Рис. 2.33. Структуры локальных адресов для использования в пределе сегмента (*а*) и в пределе локальной сети (*б*)

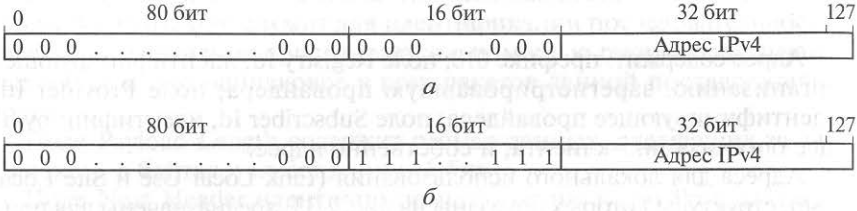


Рис. 2.34. Структура адресов, построенных на основе IPv4:
а — совместимый; *б* — для устройств, не поддерживающих IPv4

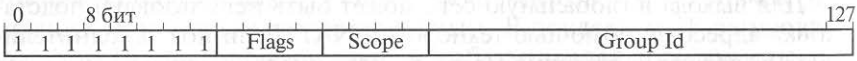


Рис. 2.35. Структура широковещательного адреса

Таблица 2.4

Значения поля Scope

Значение Scope	Область распространения пакета
0	Зарезервировано
1	Внутри узла
2	Внутри сегмента
5	Внутри локальной сети
8	Внутри организации
0Eh	Глобальное
0Fh	Зарезервировано

При рассмотрении возможностей, предоставляемых новым протоколом, может возникнуть вопрос о том, зачем он все-таки нужен, поскольку большинство функций либо уже имеются в IPv4, либо могут быть реализованы посредством доработки соответствующих протоколов. Так, автоматическое выделение адресов производится с помощью протокола DHCP, адресный барьер преодолевается с помощью протокола NAT и т.д. Однако разработка всех необходимых заплаток для протокола IPv4 потребовала бы усилий, не меньших (а то и больших), чем создание нового протокола «с чистого листа». Разумеется, этот лист был бы не совсем чистым, поскольку вопросы совместимости и совместной работы обоих протоколов предполагались с самого начала проектирования. В конце концов Интернет все равно ожидал бы дефицит адресов, так что заблаговременная разработка и постепенное внедрение протокола IPv6 были более чем уместны.

Для реализации перехода на новый протокол образовалась неформальная некоммерческая организация 6bone, включающая в себя более 100 различных организаций, в основном сетевых провайдеров и университетов. Главная задача этой организации — создание инфраструктуры, обеспечивающей транспортировку пакетов стандарта IPv6 по всей сети Интернет. Как и существующая сегодня инфраструктура IPv4, она будет состоять из большого числа провайдеров и локальных сетей, объединенных в единую сеть. В настоящее время в состав 6bone входят представители 41 страны.

Необходимость создания такой инфраструктуры объясняется, прежде всего, тем, что без широкомасштабного тестирования и готовой инфраструктуры (или ее подобия) коммерческие провайдеры (и потребители, занимающиеся в отличие от университетов не исследованиями, а бизнесом) вряд ли будут охотно внедрять новый протокол. Таким образом, задачей сети 6bone является не организация параллельной инфраструктуры, а скорее тестирование и отработка методик взаимодействия клиент—провайдер.

Сама сеть 6bone состоит из островков-сетей, полностью поддерживающих IPv6 и соединенных виртуальными туннелями, работающими на установленном у провайдеров оборудовании, которое используется и в коммерческих целях. Согласно существующему в настоящее время мнению организация 6bone будет существовать до тех пор, пока будет актуальна ее основная цель — популяризация протокола IPv6.

2.6. Введение в архитектуру MPLS

2.6.1. Общие сведения

MPLS (MultiProtocol Label Switching) — это технология быстрой коммутации пакетов в транспортной среде мультипротокольных сетей,

основанная на использовании меток. MPLS разрабатывается и позиционируется как способ построения высокоскоростных IP-магистралей над произвольной СПД. Однако область ее применения не ограничивается протоколом IP, а распространяется на трафик любого сетевого протокола с маршрутизацией. Традиционно главными требованиями, предъявляемыми к технологии магистральной сети, были высокая пропускная способность, малая задержка и хорошая масштабируемость. Однако современные приложения диктуют новые требования: доступ к интегрированным сервисам сети, организация виртуальных частных сетей (VPN, см. гл. 4) и ряд других интеллектуальных услуг.

Для решения задач, возникающих при удовлетворении этих требований, и разрабатывается архитектура MPLS, которая обеспечивает построение магистральных сетей, имеющих бóльшие, чем у рассмотренных ранее протоколов маршрутизации, возможности масштабирования, повышенную скорость обработки трафика за счет разделения собственно процесса вычисления маршрута и процесса коммутации пакетов.

За развитие архитектуры MPLS отвечает рабочая группа с одноименным названием, входящая в секцию по маршрутизации консорциума IETF. В деятельности группы принимают активное участие представители крупнейших поставщиков сетевых решений и оборудования. Эта архитектура выросла из системы Tag Switching, предложенной Cisco Systems, однако некоторые идеи были заимствованы у конкурирующей технологии IP-коммутации, созданной компанией Ipsilon, и проекта ARIS корпорации IBM. В архитектуре MPLS собраны наиболее удачные элементы всех упомянутых разработок. Можно ожидать, что вскоре появится стандарт на эту технологию благодаря усилиям IETF и компаний, заинтересованных в скорейшем продвижении данной технологии на рынок. Рассмотрим архитектуру MPLS [6].

2.6.2. Принцип коммутации

В основе MPLS лежит принцип обмена меток. Любой передаваемый пакет несет метку так называемого класса эквивалентности при коммутации — FEC (Forwarding Equivalence Class). Значение метки уникально лишь для участка пути между соседними узлами сети MPLS, которые называются также маршрутизаторами, коммутирующими по меткам (Label Switching Router — LSR). Метка передается в составе каждого пакета, причем способ ее привязки к пакету зависит от используемой технологии канального уровня, т.е. от СПД.

Маршрутизатор LSR получает информацию о топологии сети с помощью алгоритмов маршрутизации OSPF, BGP и IS-IS. Затем он начинает взаимодействовать со смежными маршрутизаторами, распределяя метки, которые в дальнейшем будут применяться для коммутации. Обмен метками может производиться с помощью как специального протокола распределения меток (Label Distribution

Protocol — LDP), так и модифицированных версий других протоколов (например, незначительно видоизмененных протоколов маршрутизации, резервирования ресурсов и др.).

Распределение меток между LSR-маршрутизаторами приводит к установлению внутри АС множества путей MPLS с коммутацией по меткам (Label Switching Path — LSP). Каждый маршрутизатор LSR содержит таблицу, которая ставит в соответствие каждой паре входной интерфейс—входная метка вектор <префикс адреса получателя—выходной интерфейс—выходная метка>. Получая пакет, LSR по номеру интерфейса, на который он пришел, и по значению сопоставленной пакету метки определяет для него выходной интерфейс. (Значение префикса применяется лишь для построения таблицы и в самом процессе коммутации не используется.) Старое значение метки заменяется новым, содержавшимся в поле «Выходная метка» таблицы, и пакет отправляется к следующему устройству, следуя по пути LSP.

Вся операция требует лишь одноразовой идентификации значений полей в одной строке таблицы. Это занимает гораздо меньше времени, чем сравнение IP-адреса отправителя с наиболее длинным адресным префиксом в таблице маршрутизации, которое используется при традиционной маршрутизации.

Сеть MPLS подразделяется на две функционально различные зоны: ядро и граничную (рис. 2.36). Ядро образуют устройства, минимальными требованиями к которым являются поддержка MPLS и участие в процессе маршрутизации трафика для того протокола, в котором коммутацию осуществляет MPLS. Маршрутизаторы ядра занимаются только коммутацией. Все функции классификации пакетов по различным FEC, а также реализацию таких дополнительных сервисов, как фильтрация, явная маршрутизация, выравнивание нагрузки и управление трафиком, берут на себя граничные LSR. В результате интенсивные вычисления приходятся на граничную область, а высокопроизводительная коммутация выполняется в ядре, что позволяет оптимизировать конфигурацию устройств MPLS в зависимости от их местоположения в сети.

Таким образом, главная особенность MPLS — **отделение процесса коммутации пакета от процесса анализа IP-адресов в его заголовке**. Такое отделение открывает ряд привлекательных возможностей. Например, очевидным следствием такого отделения является то, что очередной сегмент LSP может не совпадать с очередным сегментом маршрута, который был бы выбран при традиционной маршрутизации. Поскольку на установление соответствия пакетов определенным классам FEC могут влиять не только IP-адреса, но и другие параметры, нетрудно реализовать, например, назначение различных LSP пакетам, относящимся к различным потокам RSVP или имеющим разные приоритеты обслуживания. Конечно, подобный сценарий удается осуществить и в обычных маршрутизируемых сетях, но ре-

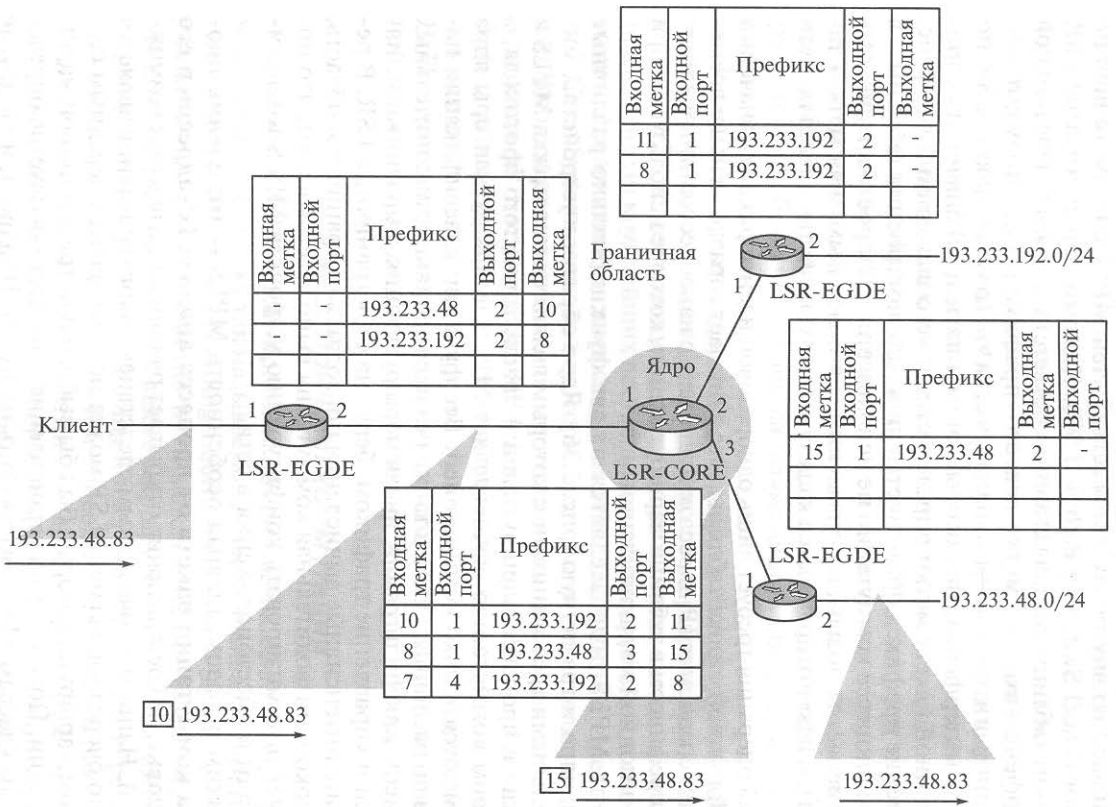


Рис. 2.36. Схема коммутации MPLS

шение на базе MPLS оказывается проще и к тому же гораздо лучше масштабируется.

Каждый из классов FEC обрабатывается отдельно от остальных не только потому, что для него строится свой LSP-путь, но и в целях обеспечения доступа к общим ресурсам (полосе пропускания канала и буферному пространству). В результате технология MPLS позволяет очень эффективно поддерживать требуемое качество обслуживания, не нарушая предоставленных пользователю гарантий. В LSR предусмотрено использование целого ряда механизмов, которые позволяют оператору сети MPLS контролировать распределение ресурсов и изолировать трафик отдельных пользователей.

Использование явно задаваемого маршрута в сети MPLS свободно от недостатков стандартной IP-маршрутизации от источника, поскольку вся информация о маршруте содержится в метке и пакету не требуется нести адреса промежуточных узлов, что улучшает управление распределением нагрузки в сети.

2.6.3. Элементы архитектуры

Метки и способы маркировки

Метка — это короткий идентификатор фиксированной длины, который определяет класс FEC, т. е. по значению метки определяется принадлежность пакета к определенному классу на каждом из участков коммутируемого маршрута. Как уже отмечалось, метка должна быть уникальной лишь в пределах соединения между каждой парой логически соседних LSR. Следовательно, LSR-маршрутизатор может использовать одно и то же ее значение для связи с различными соседними маршрутизаторами, если только можно определить, от какого из них пришел пакет с данной меткой. Другими словами, в соединениях типа точка—точка допускается применение одного набора меток на интерфейс, а для сред с множественным доступом необходим один набор меток на все устройство. В реальных условиях угроза исчерпания пространства меток очень маловероятна.

Перед включением в состав пакета метка определенным образом кодируется. В случае использования протокола IP она помещается в специальный «тонкий» заголовок пакета, инкапсулирующего IP. В других ситуациях метка записывается в заголовок протокола канального уровня или кодируется в виде определенного значения в заголовке уровня ATM (в СПД ATM). Метка для пакетов протокола IPv6 может размещаться в поле идентификатора потока.

Стек меток

В архитектуре MPLS вместе с пакетом разрешено передавать не одну метку, а целый стек меток. Операции добавления/изъятия мет-

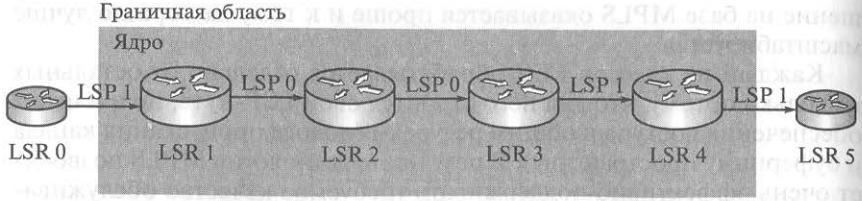


Рис. 2.37. Компоненты коммутируемого соединения

ки определены как операции на стеке (push/pop). Результат коммутации определяет лишь верхняя метка стека, а нижние метки передаются прозрачно до операции изъятия верхней. Такой подход позволяет создавать иерархию потоков в сети MPLS и организовывать туннельные передачи. Стек состоит из произвольного числа элементов, каждый из которых имеет длину 32 бит: 20 бит составляют собственно метку, 8 бит отводятся под счетчик времени жизни пакета, 1 бит указывает на нижний предел стека, а 3 бит не используются. Метка может принимать любое значение кроме нескольких зарезервированных.

Коммутируемый путь (LSP) одного уровня состоит из последовательного набора участков, коммутация на которых происходит с помощью метки данного уровня (рис. 2.37). Например, LSP нулевого уровня проходит через устройства LSR 0, LSR 1, LSR 3, LSR 4 и LSR 5. При этом LSR 0 и LSR 5 являются соответственно входным (ingress) и выходным (egress) маршрутизаторами для LSP нулевого уровня, а LSR 1 и LSR 3 — для LSP первого уровня. При этом первый из них производит операцию добавления метки в стек, а второй — операцию изъятия метки. С позиции трафика нулевого уровня LSP первого уровня является прозрачным туннелем. В любом сегменте LSP можно выделить верхний и нижний пути по отношению к трафику. Например, для сегмента LSR 4... LSR 5 четвертый маршрутизатор будет верхним, а пятый — нижним.

Привязка и распределение меток

Под привязкой понимается соответствие между определенным классом FEC и значением метки для данного сегмента LSP. Привязку всегда осуществляет нижний маршрутизатор LSR, поэтому и информация о ней распространяется только в направлении от нижнего LSR к верхнему. Вместе с этими сведениями могут передаваться и атрибуты привязки.

Обмен информацией о привязке меток и атрибутах осуществляется между соседними LSR с помощью протокола распределения меток. Архитектура MPLS не зависит от конкретного протокола. Очень эффективно реализуется RSVP-протокол (см. подразд. 2.3.9) при со-

вмещении резервирования ресурсов и организации LSP для различных потоков.

Существуют два режима распределения меток: независимый и упорядоченный. Независимый режим предусматривает возможность уведомления верхнего узла о привязке до получения конкретным LSR информации о привязке для данного класса от своего нижнего соседа. Упорядоченный режим разрешает высылать подобное уведомление только после получения указанной информации от нижнего LSR, за исключением случая, когда маршрутизатор LSR является выходным для этого класса FEC. Распространение информации о привязке может инициироваться запросом от верхнего устройства LSR (downstream on-demand) или осуществляться спонтанно (unsolicited downstream).

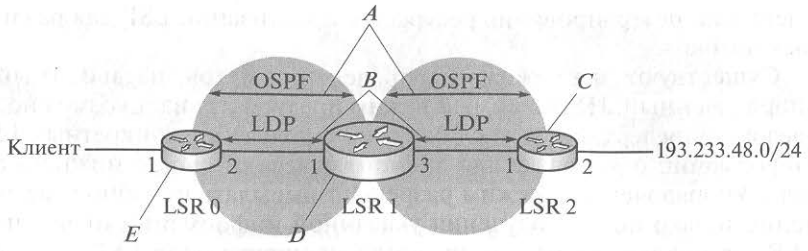
2.6.4. Построение коммутируемого маршрута

Рассмотрим автоматическое создание системой MPLS пути LSP в простейшем случае — с помощью протокола LDP (Label Distribution Protocol). Архитектура MPLS не требует обязательного применения протокола LDP, однако в отличие от других возможных вариантов он наиболее близок к окончательной стандартизации.

Сначала посредством многоадресной рассылки LDP-сообщений коммутирующие маршрутизаторы определяют своих «соседей». Кроме соседства (adjacency) на канальном уровне протокол LDP может устанавливать связь между «логически соседними» LSR, не принадлежащими одному каналу. Это необходимо для реализации туннельной передачи. После установления соседства протокол LDP открывает TCP-соединение на транспортном уровне между участниками сеанса, по которому передаются запросы на установку привязки и сама информация о привязке. Кроме того, участники сеанса периодически проверяют работоспособность друг друга, отправляя тестовые сообщения (keep alive message).

Рассмотрим на примере, как происходит заполнение таблиц меток по протоколу LDP (рис. 2.38). Предположим, что выбран упорядоченный режим распределения меток LSP со спонтанным распространением сведений о привязке. На стадии *A* каждое из устройств сети MPLS строит топологию сети, задействуя любой из современных протоколов маршрутизации (здесь это OSPF). На стадии *B* маршрутизаторы LSR применяют процедуру нахождения соседних LSR и устанавливают с ними сеансы протокола LDP.

На стадии *C* маршрутизатор LSR 2 на основе анализа собственных таблиц маршрутизации обнаруживает, что он является выходным LSR для пути, ведущего к IP-сети 193.233.48.0. Тогда LSR 2 ассоциирует класс FEC с пакетами, адрес получателя которых соответствует префиксу данной сети, и присваивает этому классу случайное значение метки (в нашем случае 18). Получив привязку, протокол LDP уведом-



Входная метка	Входной порт	Префикс	Выходной порт	Выходная метка
-	-	193.233.48	2	
-	-	193.233.48	2	
-	-	193.233.48	2	33

Входная метка	Входной порт	Префикс	Выходной порт	Выходная метка
		193.233.48	3	
		193.233.48	3	18
33	1	193.233.48	3	18

Входная метка	Входной порт	Префикс	Выходной порт	Выходная метка
		193.233.48	2	
18	1	193.233.48	2	
18	1	193.233.48	2	

Рис. 2.38 Построение коммутируемого пути по протоколу LDP

ляет верхний маршрутизатор LSR (LSR 1) о том, что потоку, адресованному сети с префиксом 193.233.48, присвоена метка 18, и LSR 1 помещает это значение в поле выходной метки своей таблицы.

На стадии *D* устройство LSR 1, которому известно значение метки для потока, адресованного префиксу 193.233.48, присваивает собственное значение метки данному FEC и уведомляет верхнего соседа (узел LSR 0) об этой привязке. Теперь LSR 0 записывает полученную информацию в свою таблицу. После завершения данного процесса все готово для передачи пакетов из сети клиента в сеть с адресом 193.233.48.0, т. е. по выбранному пути LSP.

Спецификация класса FEC может содержать несколько компонентов, каждый из которых определяет набор пакетов, соответствующих данному классу. На сегодняшний день определены два компонента FEC: адрес узла (host address) и адресный префикс (address prefix). Пакет классифицируется как принадлежащий к данному классу FEC, если адрес получателя точно совпадает с компонентом адреса узла либо имеет максимальное совпадение с адресным префиксом. В нашем примере узел LSR 0 выполняет в процессе передачи классификацию пакетов, поступающих к нему из сети клиента, и (если адрес получателя в них совпадает с префиксом 193.233.48), присвоив пакету метку 33, отправляет его через интерфейс 2.

Более полную информацию о MPLS-сетях можно найти в [6].

Глава 3
ТРАНСПОРТНЫЙ УРОВЕНЬ

3.1. Сервис

3.1.1. Сервис для верхних уровней

Рассмотрим теперь, какие виды сервиса транспортный уровень предоставляет прикладному уровню, как характеризуется качество предоставляемого сервиса, как обеспечивается доступ к сервису транспортного уровня из прикладного уровня и каковы интерфейсы транспортного уровня со смежными уровнями.

Основная цель транспортного уровня — обеспечить эффективный, надежный и дешевый сервис для пользователей на прикладном уровне. Достижение этой цели является и задачей сервиса, предоставляемого сетевым уровнем. Работу транспортного уровня выполняет *транспортный агент*, который может располагаться в ядре операционной системы, в отдельном процессе пользователя, в библиотеке сетевого приложения или на карте сетевого интерфейса.

В некоторых случаях компания оператор сети, или, как еще говорят, сервис-провайдер, может предоставлять надежный транспортный сервис, при котором транспортный агент располагается на специальной интерфейсной машине на границе транспортной среды, к которой подключены абонентские машины. Подобно сетевому уровню, транспортный уровень может поддерживать два вида сервиса: с соединениями и без соединений. Транспортный сервис, ориентированный на соединение, имеет много общего с аналогичным сетевым сервисом. Например, на обоих уровнях аналогичны задачи адресации и управления потоком.

Возникает вопрос: если сервис сетевого уровня аналогичен сервису транспортного, то зачем выделять два разных уровня? Причина состоит в том, что сетевой уровень — это часть транспортной среды, которой управляет сервис-провайдер. Что же будет, если сетевой уровень предоставит ненадежный сервис, ориентированный на соединения? Предположим, что на этом уровне часто пропадают пакеты. Что делать, если маршрутизатор время от времени отказывает? У пользователя такой транспортной среды отсутствуют средства для решения возникающих проблем, поэтому поверх сетевого необходимо организовать еще один уровень, который позволит повысить качество сервиса сетевого уровня. Если транспортному уровню придет сообще-

ние, что соединение на сетевом уровне неожиданно было разорвано, то он должен иметь возможность установить новое сетевое соединение и с его помощью выяснить, что произошло, какие данные были переданы, а какие не переданы, после чего предпринять необходимые действия.

Задача транспортного уровня заключается в том, чтобы сервис, используемый прикладным уровнем, сделать более надежным, чем сервис сетевого уровня для транспортного. Также важным свойством транспортного уровня является то, что прикладная программа, опираясь на транспортный сервис, становится независимой от транспортной среды и может работать в сети с любым сетевым сервисом. И, наконец, с транспортным сервисом работает прикладная программа, а с сетевым — транспортный уровень, поэтому интерфейс с транспортным уровнем должен быть дружественным, удобным и эффективным.

В силу приведенных доводов первые четыре уровня в модели OSI (см. подразд. 1.1.2) называют поставщиками транспортного сервиса, а уровни выше четвертого — пользователями транспортного сервиса.

3.1.2. Качество сервиса

Мы уже встречались с понятием качества сервиса при рассмотрении сетевого уровня и рассматривали набор параметров, характеризующих это понятие. Транспортный уровень позволяет приложению в момент установления соединения определить желаемые, допустимые и минимальные значения для параметров, характеризующих качество сервиса. Далее на транспортном уровне решается задача: можно ли с помощью сетевого сервиса удовлетворить эти запросы и до какой степени.

Приведем основные параметры качества сервиса и заметим, что лишь немногие сети поддерживают все эти параметры:

- **Connection establishment delay** — задержка на установку соединения, определяющая время между запросом на установку соединения и подтверждением его установки;
- **Connection establishment failure probability** — вероятность того, что соединение не будет установлено за время, равное задержке на установку соединения;
- **Throughput** — пропускная способность транспортного соединения, определяющая число байтов пользователя, передаваемое за одну секунду;
- **Transit delay** — задержка при передаче, определяющая время от момента, когда сообщение ушло с машины-отправителя, до момента, когда оно получено машиной-получателем;
- **Residual error ration** — доля ошибок при передаче. Этот параметр определяет отношение числа сообщений, при передаче которых

были ошибки, включая потерянные сообщения, к общему числу переданных сообщений. Теоретически этот параметр должен быть равен нулю, если транспортный уровень надежно передает сообщение, однако на практике это не так;

- **Protection** — параметр, позволяющий определить уровень защиты передаваемых данных от несанкционированного доступа третьей стороной. Косвенно он определяет, на какие затраты готов пойти пользователь для защиты своих данных от перехвата при передаче на транспортном уровне;
- **Priority** — приоритет, позволяющий пользователю указать степень важности для него данного соединения среди остальных соединений;
- **Resilience** — устойчивость, определяющая вероятность разрыва транспортным уровнем соединения в силу своих внутренних проблем или перегрузки.

Желаемое и минимальное значения параметров, характеризующих качество сервиса, определяет приложение—отправитель данных в момент установления транспортного соединения. Если требуемое качество недостижимо, то транспортный уровень сразу сообщает об этом приложению-отправителю, указывая причины неудачи. При этом приложению-получателю ничего не сообщается. Процедура согласования параметров качества сервиса называется *согласованием возможностей*.

3.1.3. Примитивы транспортного уровня

Примитивы транспортного уровня открывают пользователю доступ к транспортному сервису. Транспортный сервис аналогичен сервису сетевого уровня. Однако между ними существует одно различие: качеством сервисов сетевого уровня, включая надежность передачи, на прикладном уровне управлять нельзя. Сетевой уровень принадлежит транспортной среде, которую прикладной уровень контролировать не может. Задача транспортного сервиса как раз в том и состоит, чтобы обеспечить доставку сообщений с качеством, заданным приложением. Два процесса прикладного уровня, соединенные между собой, ничего не должны знать о том, как физически они соединены. Один помещает данные на вход транспортного уровня, другой получает их. Задача транспортного уровня скрыть и от получателя, и от отправителя все детали передачи, исправления ошибок и т. п.

Теоретически транспортный сервис может быть как ориентированным на соединения, так и не ориентированным на соединения. Однако дейтаграммный транспортный сервис — это редкость, поэтому мы будем рассматривать транспортный сервис, ориентированный на соединения.

Другое важное различие между сетевым и транспортным сервисами заключается в их применении: сетевой сервис используется транс-

портным сервисом, а транспортный сервис используется прикладными программами. Транспортный сервис должен быть ориентирован на пользователя, т.е. быть удобным и простым в применении.

Примитивы транспортного уровня аналогичны примитивам сетевого уровня. Общее представление о примитивах транспортного сервиса дадим с помощью следующего примера. Сервер приложения выполняет примитив LISTEN, в результате чего он блокируется до поступления запросов от клиентов. Клиент для установления соединения выполняет примитив CONNECT. Транспортный агент на стороне клиента блокирует клиента и посылает серверу пакет с запросом на установление соединения.

Напомним, что транспортные агенты обмениваются пакетами, имеющими специальное название — TPDU (Transport Protocol Data Unit), которые мы будем называть сегментами.

По примитиву CONNECT транспортный агент со стороны клиента посылает сегмент CONNECTION REQUEST. Транспортный агент сервера, увидев, что сервер заблокирован по примитиву LISTEN, разблокирует сервер и посылает сегмент CONNECTION ACCEPTED. После этого транспортное соединение считается установленным и начинается обмен данными с помощью примитивов SENT и RECEIVE.

По окончании обмена данными транспортное соединение должно быть разорвано. Есть два варианта разрыва соединения: асимметричный и симметричный.

Асимметричный разрыв предполагает, что для разрыва соединения только одна из сторон посылает сегмент DISCONNECT. Сторона, получившая такой TPDU-сегмент, считает соединение разорванным.

При симметричном разрыве каждая сторона закрывает соединения отдельно, и только после этого соединение считается разорванным.

Когда одна сторона посылает сегмент DISCONNECT, это означает, что с ее стороны больше данных не будет.

На рис. 3.1 показана диаграмма состояний при установлении и разрыве соединения.

В табл. 3.1 показан другой набор примитивов — это так называемые *сокеты Беркли*. В этом наборе имеется два основных отличия от того, что мы рассмотрели ранее. Первые четыре примитива выполняются сервером в том порядке, в каком они указаны в таблице. Примитив SOCKET создает структуру данных по определенному шаблону для подключения к серверу, резервируя для нее место в таблице транспортного агента. Параметры обращения определяют формат адреса, тип желаемого сервиса, протокол и т.д. По примитиву BIND сервер выделяет сокету адрес. Причина, по которой адрес выделяется не сразу, состоит в том, что некоторые процессы сами управляют своим адресным пространством, жестко закрепленным за ними. Примитив LISTEN не является блокирующим, он выделяет

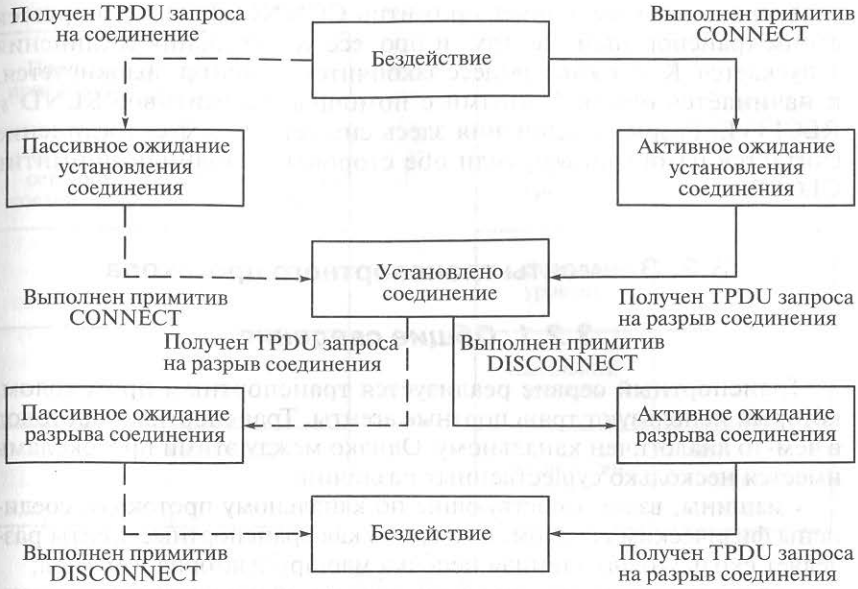


Рис. 3.1. Диаграмма состояний при установлении и разрыве соединения

ресурсы и создает очередь, если несколько клиентов будут обращаться за соединением в одно и то же время. Примитив ACCEPT является блокирующим в ожидании запроса на соединение.

Таблица 3.1

Сокеты Беркли

Примитив	Значение
SOCKET	Создание новой точки подключения
BIND	Прикрепление локального адреса к сокету
LISTEN	Объявление готовности принимать соединения; сообщение о размере очереди
ACCEPT	Блокировка вызывающего в ожидании запроса на соединение
CONNECT	Активная попытка установления соединения
SEND	Отправка данных через данное соединение
RECEIVE	Получение данных через данное соединение
CLOSE	Разрыв соединения

Когда клиент выполняет примитив CONNECT, он блокируется своим транспортным агентом, и процесс установления соединения запускается. Когда этот процесс закончится, клиент разблокируется, и начинается обмен данными с помощью примитивов SEND и RECEIVE. Разрыв соединения здесь симметричен, т.е. соединение считается разорванным, если обе стороны выполнили примитив CLOSE.

3.2. Элементы транспортного протокола

3.2.1. Общие сведения

Транспортный сервис реализуется транспортным протоколом, который используют транспортные агенты. Транспортный протокол в чем-то аналогичен канальному. Однако между этими протоколами имеется несколько существенных различий:

- машины, взаимодействующие по канальному протоколу, соединены физическим каналом, в то время как транспортные агенты разделяет сколь угодно длинная цепочка маршрутизаторов и мостов;
- процессы на канальном уровне взаимодействуют непосредственно через общий канал, поэтому процедура установления соединения здесь намного проще, чем на транспортном;
- среда, в которой работает транспортный протокол, использует память маршрутизаторов, которая может терять свое содержимое;
- число соединений, которые могут возникать на транспортном уровне, намного больше, чем число соединений на канальном уровне, что создает дополнительные проблемы для буферизации и управления потоком.

Транспортный протокол должен обеспечивать решение следующих вопросов:

1. Как адресовать прикладной процесс, с которым необходимо установить соединение?
2. Как корректно установить соединение? Поскольку сегменты могут теряться, как отличить сегменты нового соединения от повторных сегментов, оставшихся от старого соединения?
3. Как корректно разрывать соединение?

Рассмотрим последовательно существующие решения для этих проблем.

3.2.2. Адресация

Проблема адресации состоит в том, как указать на транспортном уровне, с каким удаленным прикладным процессом требуется установить соединение. Обычно для этого используется транспортный адрес, по которому прикладной процесс может слушать запросы на

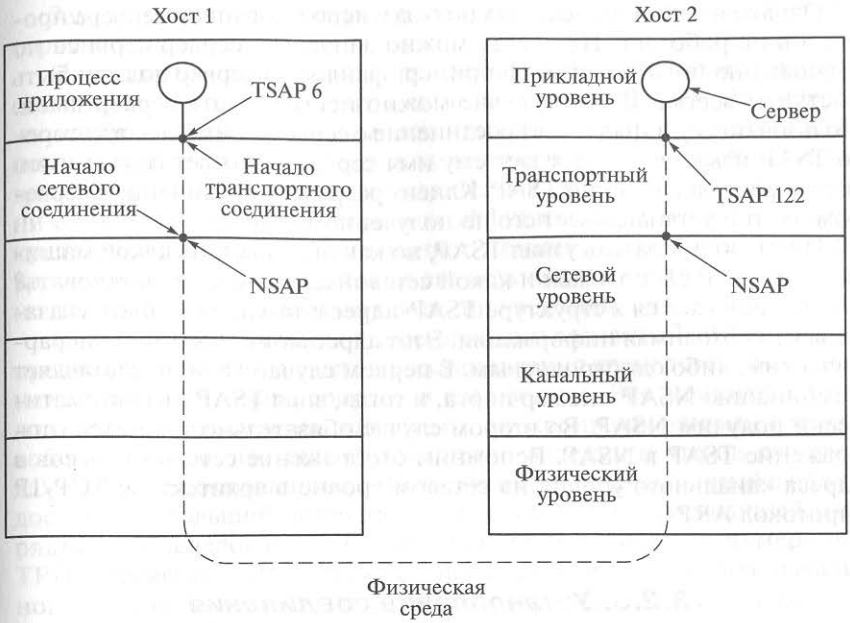


Рис. 3.2. Взаимосвязь TSAP и NSAP

соединение. Вместо транспортного адреса мы будем здесь использовать понятие Transport Service Access Point — TSAP. Напомним что, аналогичное понятие существует и на сетевом уровне — IP-адрес, или NSAP для сетевого уровня.

На рис. 3.2 показана взаимосвязь TSAP и NSAP, а также сценарий использования TSAP для установления соединения между двумя удаленными процессами.

Из рис. 3.2 неясно только, как прикладной процесс на машине 1 узнает, что интересующий его сервер подключен к TSAP 122 на машине 2. Возможно, что, если данный сервер всегда подключен к TSAP 122, все процессы об этом знают.

Это решение хорошо работает для часто используемого сервиса с длительным периодом активности, но как быть прикладным процессам пользователя, которые активизируются спорадически на короткое время? В операционной системе Unix для этого используется *протокол установления начального соединения*, работающий следующим образом. На каждой машине имеется, специальный сервер процессов, который представляет все процессы, исполняемые на этой машине. Этот сервер слушает несколько TSAP, куда могут поступить запросы на TCP-соединение. Если свободного TSAP, способного выполнить запрос, не имеется, то соединение устанавливается с сервером процессов, который переключает соединение на необходимый сервер, как только тот освободится.

Однако в некоторых случаях подход с использованием сервера процессов не работает. Не всегда можно запускать сервер сервиса по требованию пользователя. Например, файловый сервер должен быть доступен всегда. В этом случае можно использовать сервер имен. Пользователь устанавливает соединение с сервером имен, для которого TSAP известен, и передает ему имя сервиса. В ответ сервер имен пересылает надлежащий TSAP. Клиент разрывает соединение с сервером имен и устанавливает его по полученному адресу.

Пусть пользователь узнал TSAP, но как он узнает, на какой машине этот TSAP расположен и какой сетевой адрес надо использовать? Ответ заключается в структуре TSAP-адреса: там должна быть указана вся необходимая информация. Этот адрес может быть либо иерархическим, либо одноуровневым. В первом случае TSAP представляет комбинацию NSAP + номер порта, и тогда, зная TSAP, мы автоматически получим NSAP. Во втором случае обязательно требуется отображение TSAP в NSAP. Вспомним отображение сетевых адресов в адреса канального уровня на сетевом уровне в архитектуре TCP/IP (протокол ARP).

3.2.3. Установление соединения

Проблема установления транспортного соединения сложна, поскольку пакеты могут теряться, храниться и дублироваться на сетевом уровне. Типичным примером является установление соединения с банком в системе интернет-банка для перевода денег с одного счета на другой, где вследствие перегрузки в сети или по какой-либо другой причине может произойти большая задержка. Тогда по истечении time-out активная сторона вышлет еще один запрос. При этом сегменты-дубли могут вызвать повторное соединение и вторичный перевод денег. Как же быть?

Одно из возможных решений этой проблемы — использование временного TSAP. После того как оно использовано, TSAP с таким адресом более не должно возникать. При этом решении не работает модель с сервером процессов, в которой определенные TSAP имеют фиксированные адреса.

Для решения данной проблемы каждому транспортному соединению можно сопоставить уникальный номер. Когда соединение разрывается, этот номер заносится в специальный список. К сожалению, такой список может расти бесконечно. Кроме того, в случае сбоя машины он может быть потерян, и тогда ...

Альтернативой здесь может быть ограничение времени жизни сегментов, которое можно достичь следующими способами:

- ограничение топологии транспортной среды;
- установка счетчиков скачков в каждом сегменте;
- установка временной метки на каждом сегменте.

Заметим, что последний метод требует синхронизации маршрутизаторов в сети.

На практике необходимо сделать так, чтобы недействительными стали не только сами сегменты, но и уведомления о них. Это значит, что следует ввести максимальный интервал времени T, который в несколько раз превышает время жизни сегмента в сети. Этот интервал времени T вводится как произведение максимального времени жизни сегмента в сети и некоторого коэффициента, зависящего от конкретного протокола. Он позволяет немного увеличить реальное время жизни сегмента, чтобы по его истечении в сети гарантированно не осталось ни самого сегмента, ни уведомления о нем.

На основе ограничения времени жизни сегмента Томлинсон [92] предложил безопасный метод восстановления транспортного соединения, идея которого состоит в следующем. Все машины в сети должны быть оснащены таймерами, синхронизированными между собой. Считаем, что таймер абсолютно надежен и что он работает даже в случае сбоя машины. Каждый таймер — это двоичный счетчик достаточно большой разрядности, равной или превосходящей разрядность последовательных чисел, используемых для нумерации TPDU-сегментов. При установлении соединения в качестве начального номера сегмента берут значения нескольких младших разрядов этого таймера. Главное, чтобы последовательности номеров сегментов в каждом соединении не приводили к переполнению счетчиков и их обнулению. Заметим, что эти номера можно также использовать для управления потоком в протоколе скользящего окна.

Проблема возникает, когда машина восстанавливается после сбоя. В этот момент транспортный агент не знает, какое число можно использовать для очередного номера сегмента. Чтобы избежать повторного использования порядкового номера, который уже был сгенерирован перед сбоем машины, строят специальную область запрещенных номеров.

Необходимость введения этой области демонстрирует пример, приведенный на рис. 3.3, *а* [19]. Предположим для простоты, что $T = 60$ с, часы «тикают» один раз в секунду и в момент x начальный номер первого сегмента будет x. Таким образом, максимальное время жизни сегмента будет равно 60 с. Пусть в момент времени $t = 30$ с был послан сегмент с номером 80, после чего наступил сбой в работе. Пусть в момент $t = 60$ с машина была восстановлена после сбоя и в момент $t = 70$ с был сформирован сегмент с номером 80. Однако старый сегмент с номером 80 все еще существует, так как он будет жить до момента времени $t = 90$ с, поэтому для каждого момента времени вводится область запрещенных номеров.

Машина, восстановленная после сбоя, не может выбирать номера из запретной зоны, поэтому после восстановления следует подождать время, равное T, пока все ранее посланные сегменты не перестанут существовать. На практике поступают иначе. Чтобы не тратить впу-

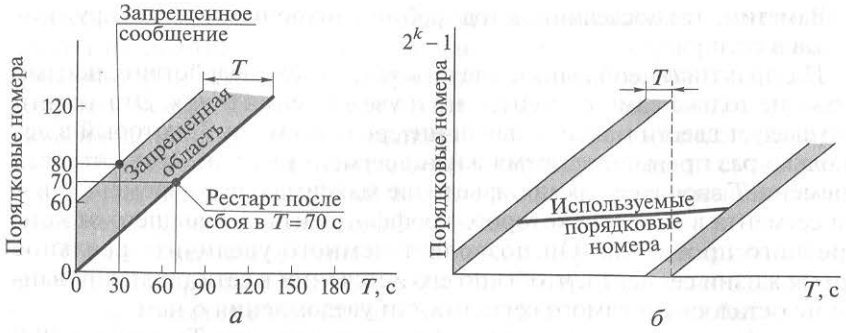


Рис. 3.3. Примеры построения области запрещенных номеров (*а*) и решения проблемы использования зоны запрещенных номеров (*б*)

стую это время T, строят кривую скорости генерации номеров вида $n = at$, и после сбоя выбирают номера по формуле $n = at + T$.

При решении, предложенном Томлинсоном, может возникать две проблемы. Причинами первой проблемы может быть либо слишком быстрое генерирование машиной пакетов и соединений, либо слишком медленное. В обоих случаях фактическая крутизна кривой скорости генерации номеров будет либо больше, либо меньше чем расчетной кривой (рис. 3.3, *б*), т. е. будем попадать в запрещенные зоны номеров: в первом случае — снизу слева от запрещенной области, а во втором — справа.

Решением проблемы в первом случае является установление минимального интервала времени, в течение которого может быть сгенерирован только один сегмент. Во втором случае TCP-агент должен просчитывать наперед, не окажутся ли номера сегментов в запрещенной зоне в ближайшее время, если сегменты будут появляться медленно. Если расчет покажет, что в ближайшее время произойдет попадание в запрещенную зону номеров, то TCP-агент должен выдержать необходимую паузу и только потом продолжить отправку сегментов. В любом случае необходимо контролировать скорость появления сегментов по заданному соединению: чем больше разрядность счетчика последовательных номеров, тем дальше отодвигается момент попадания в запрещенную область.

Второй непростой проблемой является надежное установление соединения, поскольку сегменты могут пропадать и дублироваться. Дублирование запроса на соединение приведет к дублированию соединения. Для решения этой проблемы Томлинсон предложил процедуру «троекратного рукопожатия» (three-way handshake), показанную на рис. 3.4, *а*.

Эта процедура предполагает, что машина 1 посылает запрос на установление соединения под номером x. Машина 2 посылает подтверждение на запрос x, но со своим номером y. Машина 1 подтверж-

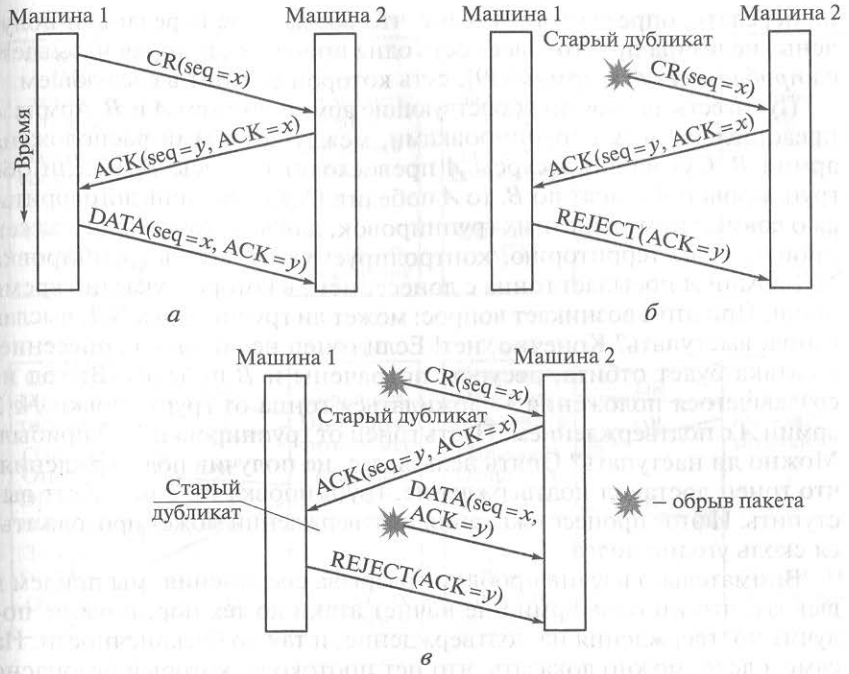


Рис. 3.4. Примеры (*а* ... *в*) процедуры «троекратного рукопожатия»

дает получение подтверждения с номером *y*. На рис. 3.4, *б*, *в* показано, что будет, если поступят запоздавший запрос на соединение и запрос и подтверждение на него. Отметим, что нумерация TPDU у отправителя и получателя не синхронизирована. Каждая сторона использует свои номера. Это важно, поскольку не требует синхронизации между отправителем и получателем. Другое решение проблемы дублирования запроса см. в [94].

3.2.4. Разрыв соединения

Разрыв соединения, как уже говорилось, может быть асимметричным или симметричным. Асимметричный разрыв, т.е. когда одна сторона разрывает соединение, не дожидаясь согласия другой стороны, может привести к потере данных (рис. 3.5). Из рис. 3.5 видно, что в тот момент, когда узел 2 послал сообщение о разрыве соединения (DR), узел 1 отправил данные (DATA). Поскольку узел 2 разорвал соединение, то посланные узлом 1 данные он не примет.

Симметричный разрыв каждая сторона проводит самостоятельно, когда она передала весь имеющийся объем данных. Однако если каждая из сторон заранее не знает объем данных, который она долж-

на передать, определить тот факт, что все данные переданы и получены, не всегда просто. Здесь есть одна проблема, которая называется *проблемой двух армий* [19], суть которой состоит в следующем.

Пусть есть две противоборствующие армии, скажем *A* и *B*. Армия *A* представлена двумя группировками, между которыми расположена армия *B*. Суммарно ресурсы *A* превосходят ресурсы *B*, и если обе группировки *A* ударят по *B*, то *A* победит. Остается лишь договориться о совместном ударе этих группировок, для чего гонец от *A* должен пройти через территорию, контролируемую *B*. Пусть группировка № 1 армии *A* посылает гонца с донесением, в котором указано время атаки. При этом возникает вопрос: может ли группировка №1, выслав гонца, выступать? Конечно, нет! Если гонец не доставил донесение, то атака будет отбита, ресурсы потрачены и *B* победит. Выход из создавшегося положения — дожидаться гонца от группировки № 2 армии *A* с подтверждением. Пусть гонец от группировки № 2 прибыл. Можно ли наступать? Опять нельзя, т. е. не получив подтверждения, что гонец доставил подтверждение, группировка № 2 не может выступить. И этот процесс ожидания подтверждений может продолжаться сколь угодно долго.

Внимательно изучив проблему разрыва соединения, мы придем к выводу, что ни одна армия не начнет атаки до тех пор, пока не получит подтверждения на подтверждение, и так до бесконечности. На самом деле, можно доказать, что нет протокола, который безопасно разрешает эту ситуацию.

На рис. 3.6, *а*... *г* показаны соответственно четыре сценария разрыва соединения по схеме с «троекратным рукопожатием»: нормальный случай с «троекратным рукопожатием», с потерей последнего подтверждения, с потерей ответа, с потерей ответа и последующих данных. Хотя сценарии, показанные на рис. 3.6, и не безупречны, но на практике обычно они работают успешно. Обратим внимание, что

Рис. 3.5. Разрыв соединения с потерей данных

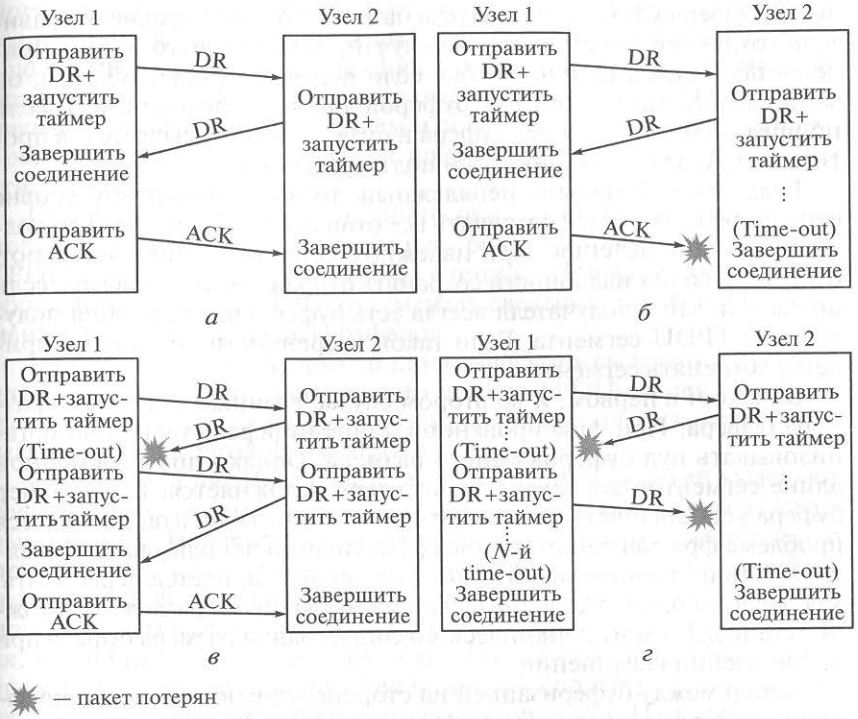


Рис. 3.6. Четыре сценария (*а*...*г*) разрыва соединения

многое в этих сценариях зависит от правильного выбора значения time-out, по истечении которого принимается решение о разрыве. Если оно будет не достаточно большим, можно потерять данные.

3.2.5. Управление потоком и буферизация

Теперь, уяснив, как устанавливают соединение, рассмотрим, как им управляют. Прежде всего, рассмотрим управление потоком. Эта проблема на транспортном уровне в чем-то аналогична проблеме управления потоком на сетевом уровне. Различие заключается в том, что у маршрутизатора число каналов невелико, в то время как на транспортном уровне соединений может быть намного больше.

Канальный протокол сохраняет кадры, как на стороне отправителя, так и на стороне получателя до тех пор, пока они не будут подтверждены. Если имеется 64 соединения и поле «Время жизни» сегмента занимает четыре разряда, то потребуется суммарная емкость буферов на 1 024 TPDU-сегмента. Число буферов можно сократить, если имеется информация о надежности сетевого уровня или о на-

личии буфера у получателя. Пусть на транспортном уровне отправитель сохраняет все сегменты на случай, если какой-то из них придется посылать вторично. Тогда, если получатель знает об этом, он может иметь лишь один пул буферов для всех соединений, и если пришел сегмент, и ему нет буфера в пуле, то он сбрасывается, в противном случае он сохраняется и подтверждается.

Если сетевой уровень ненадежный, то на транспортном уровне отправитель вынужден сохранять все отправленные сегменты до подтверждения получения. При надежном сетевом сервисе, наоборот, отправителю нет надобности сохранять отправленные сегменты, если он уверен, что у получателя всегда есть буфер для сохранения полученного TPDU-сегмента. Если такой уверенности нет, то ему придется сохранять сегменты.

Однако и в первом, и во втором случае возникает проблема размера буфера. При фиксированной длине буфера естественно организовывать пул буферов одного размера. Однако при переменной длине сегментов эта задача значительно усложняется. Если размер буфера устанавливать по максимальной длине сегмента, появляется проблема фрагментации, т. е. неэффективного использования памяти, а если по минимальной — то один сегмент придется пересылать, как несколько, т. е. с дополнительными накладными расходами. Можно установить схему динамического согласования размера буфера при установлении соединения.

Выбор между буферизацией на стороне отправителя и буферизацией на стороне получателя зависит от типа трафика. Например, для низкоскоростного нерегулярного трафика буферизацию лучше делать на обоих концах. В общем случае вопрос о числе буферов лучше всего решать динамически, необходимо только позаботиться о решении проблемы потери управляющих сегментов.

Проблемами также являются согласование доступного числа буферов и пропускная способность сетевого уровня. В настоящее время память дешевеет, поэтому буферизация скоро не будет являться проблемой. Другое дело — проблема пропускной способности сетевого уровня. Здесь дело в том, что пропускная способность транспортной среды между двумя определенными хостами ограничена, и если поток между этими двумя хостами превысит пропускную способность транспортной среды, возникнет перегрузка. Эту проблему лучше всего решать динамически с помощью управляющих сообщений.

Механизм управления потоком должен, прежде всего, учитывать пропускную способность транспортной среды, а уже потом — возможности буферизации. Располагаться этот механизм должен на стороне отправителя, чтобы предотвращать накопление большого числа неподтвержденных сообщений. При установлении и разрыве соединений, изменении интенсивности трафика отправитель и получатель должны изменять стратегию выделения буферов. Буферы

могут выделяться под каждое соединение или сразу на несколько соединений. Когда получатель выделяет буферы под соединение, он еще не знает, каким будет трафик и может случиться так, что ему придется сократить число выделяемых буферов по сравнению с изначально выделенным значением и сообщить об этом отправителю. Транспортный протокол должен предусматривать такие возможности.

Подобно алгоритму скользящего окна отправитель, получив от получателя доступное ему число буферов, с каждым отправленным TPDU уменьшает это значение на единицу. Получатель отправляет обратно на попутных TPDU-сегментах сведения о полученных сегментах и имеющемся числе буферов.

На рис. 3.7 показан пример динамического выделения буферов для случая 4-битового порядкового номера TPDU-сегментов. Пусть для простоты запрос на выделение буферов пересылается в отдельном сегменте, а не в заголовке попутного сегмента.

Сначала машина *A* запрашивает восемь буферов, но ей выделяют только четыре. Затем она посылает три сегмента, последний из которых теряется. На шестом шаге машина *A* получает подтверждение о получении сегментов 0 и 1, а также сообщение, что остались свободными три буфера. Машина *A* считает, что сегмент 2 она послала, поэтому высылает сегменты 3 и 4. После этого машина *A* блокируется, что происходит на восьмом шаге. На девятом шаге у машины *A* срабатывает таймер по истечении time-out на подтверждение сегмента 2, поэтому она посылает этот сегмент еще раз. На десятом шаге машина *B* подтверждает получение всех отправленных сегментов и выделяет буферное пространство машине *A*. На одиннадцатом шаге машина *B* выделяет буфер под один сегмент.

A	Сообщение	*B*	Комментарий
1	→ <request 8 buffers>	→	*A* хочет 8 буферов
2	← <ack — 15, buf — 4>	←	*B* гарантирует получение сообщений с 0 по 3
3	→ <seq — 0, data — m0>	→	У *A* осталось 3 буфера
4	→ <seq — 1, data — m1>	→	У *A* осталось 2 буфера
5	→ <seq — 2, data — m2>	···	Сообщение потеряно, но *A* полагает, что осталось одно
6	← <ack — 1, buf — 3>	←	*B* подтверждает 0 и 1, разрешает отправку 2-4
7	→ <seq — 3, data — m3>	→	У *A* остался буфер
8	→ <seq — 4, data — m4>	→	У *A* осталось 0 буферов, и он должен остановиться
9	→ <seq — 2, data — m2>	→	У *A* срабатывает таймаут, и он перепосылает сообщение
10	← <ack — 4, buf — 0>	←	Все подтверждения приходят, но *A* все еще блокирован
11	← <ack — 4, buf — 1>	←	Теперь *A* может отправить 5
12	← <ack — 4, buf — 2>	←	У *B* появился еще буфер
13	→ <seq — 5, data — m5>	→	У *A* остался 1 буфер
14	→ <seq — 6, data — m6>	→	*A* снова заблокирован
15	← <ack — 6, buf — 0>	←	*A* все еще заблокирован
16	··· <ack — 6, buf — 4>	←	Потенциальный дедлок

Рис. 3.7. Динамическое выделение буферов

Описанная схема таит одну опасность. Дело в том, что управляющие сегменты не подтверждаются, поэтому если сообщение, посланное машиной *B* на шестнадцатом шаге о выделении буферного пространства машине *A*, будет утеряно, то машина *A* окажется надолго в ожидании выделения буферного пространства. Чтобы предотвратить такие ситуации, каждая машина должна периодически посылать по каждому транспортному соединению сегменты с подтверждениями на полученные сегменты и информацию об имеющихся буферах.

Далее при рассмотрении протокола TCP будет приведена схема управления размером скользящего окна, определяемым исходя из текущей пропускной способности канала. Такая схема обеспечивает наиболее полное использование возможностей канала.

Подробно вопросы управления буферизацией рассмотрены в [59, 71].

3.2.6. Мультиплексирование

Потребность в мультиплексировании нескольких потоков данных одного уровня на одном соединении, виртуальном канале, физической линии нижерасположенного уровня возникала у нас и ранее. Возникает она и на транспортном уровне.

Например, если пользователь за терминалом установил транспортное соединение и отошел попить кофе, то транспортное соединение продолжает поддерживаться, под него резервируется буферное пространство, пространство в таблице маршрутизации и т. д. В целях снижения стоимости можно отобразить несколько транспортных соединений на одно сетевое. Такое отображение называется нисходящим мультиплексированием.

В некоторых случаях, наоборот, в целях увеличения пропускной способности по отдельным транспортным соединениям можно отобразить транспортное соединение на несколько сетевых, и на каждом сетевом соединении иметь свое скользящее окно. Тогда, быстро исчерпав возможности одного буферного пространства в одном окне, можно переключиться на другое сетевое соединение и продолжить передачу по этому соединению. В результате получим канал, пропускная способность которого равна сумме пропускных способностей отдельных каналов на сетевом уровне. Такое мультиплексирование называется восходящим.

3.2.7. Восстановление после сбоев

Восстановление после сбоев рассмотрим в предположении, что транспортный агент целиком располагается на абонентской машине. Восстановить сетевой уровень достаточно просто. Если сетевой уровень предоставляет дейтаграммный сервис, то транспортный уровень

должен быть ориентирован на то, что TPDU-сегменты будут теряться, и поэтому он должен уметь исправлять подобные ситуации. При сервисе, ориентированном на соединение, транспортный уровень должен уметь восстанавливать потерянное соединение и стараться в диалоге с транспортным агентом на другой стороне выяснить, что успели передать, а что не успели.

Проблема усложняется, когда надо восстанавливать работоспособность машины, включая транспортный уровень. Рассмотрим случай, когда транспортный сервер взаимодействует с клиентами. Предположим, сервер «упал» и надо восстановить его функционирование. Прежде всего, серверу необходимо узнать у клиента, какой TPDU-сегмент был последним неподтвержденным, и попросить повторить его. В свою очередь, клиент может находиться в одном из следующих состояний: S_1 — имеется неподтвержденный TPDU-сегмент; S_0 — все TPDU-сегменты подтверждены.

Казалось бы все просто. Однако рассмотрим проблему внимательнее. Сервер, получив TPDU-сегмент, либо сначала посылает подтверждение, а затем записывает полученное TPDU в буфер приложения, либо сначала записывает полученное TPDU, а затем посылает подтверждение. Если сервер «упал», послав подтверждение, но до того, как он осуществил запись, то клиент будет находиться после восстановления сервера в состоянии S_0, хотя подтвержденный TPDU-сегмент потерян. Пусть, наоборот, сервер сначала записал TPDU-сегмент, а затем «упал». Тогда после сбоя сервер найдет клиента в состоянии S_1 и решит, что надо повторить неподтвержденный TPDU-сегмент. В результате получим повторный TPDU-сегмент.

Можно формально показать, что проблема восстановления после сбоев только средствами транспортного уровня не решается. Необходимо, записав TPDU-сегмент, информировать об этом приложение и только после этого посылать подтверждение. При восстановлении следует опрашивать не только клиента на транспортном уровне, но и приложение.

3.3. Транспортные протоколы в Интернете

3.3.1. Общие сведения

В Интернете имеется два основных транспортных протокола: TCP — протокол, ориентированный на соединение, и UDP — протокол, не ориентированный на соединение. Поскольку сервис, реализуемый протоколом UDP — это практически сервис, реализуемый протоколом IP с добавлением небольшого заголовка, то основное внимание мы уделим рассмотрению протокола TCP.

TCP (Transmission Control Protocol) — специально созданный протокол для надежной передачи потока байтов по соединению типа

точка—точка через ненадежную сеть. TCP был сознательно разработан таким образом, чтобы он мог адаптироваться к условиям и особенностям разных сетей, устойчиво и эффективно функционировать в условиях Интернета (нескольких сетей). На каждой машине, поддерживающей TCP, есть TCP-агент, который располагается либо в ядре операционной системы, либо в процессе, который управляет TCP-потоками и доступом к сервису IP-протокола.

TCP получает данные в виде потока байтов от прикладного процесса, дробит этот поток на сегменты не более чем по 65 Кбайт (на практике не более 1,5 Кбайт) таким образом, чтобы сегмент помещался в IP-пакет, добавляет 20-байтовый заголовок и отправляет сегменты на сетевой уровень.

Поскольку IP-уровень не гарантирует доставку каждого пакета, то в задачи TCP-агента входят определение потерь и организация повторной передачи потерянного. Поскольку на сетевом уровне в Интернете соединения не поддерживаются, то сегменты могут поступать к получателю в произвольном порядке и задачей TCP-агента является восстановление исходного порядка.

3.3.2. Модель сервиса TCP

Доступ к TCP-сервису происходит через сокет. Сокет состоит из IP-адреса хоста и 16-разрядного локального номера на хосте, называемого *порт*. Сокеты создаются как отправителем, так и получателем. Порт — это TSAP для TCP. Каждое соединение идентифицируется парой сокетов, между которыми оно установлено. Один и тот же сокет может быть использован для разных соединений. Никаких дополнительных виртуальных соединений не создается.

Порты до 256 номера зарезервированы для стандартных сервисов, которые постоянно активны и готовы к работе. Например, для обеспечения FTP-передачи файла (см. гл. 4) соединение должно выполняться через 21-й порт, где находится FTP-демон, а для TELNET — через 23-й порт. Полный список таких портов можно найти в RFC 1700.

Все TCP-соединения — дуплексные, т.е. передача по ним происходит независимо в оба направления. TCP-соединение поддерживает только соединение точка—точка. Не существует TCP-соединений типа «от одного ко многим».

TCP-агент поддерживает поток байтов, а не поток сообщений. Напомним, это означает, что границы сообщений не поддерживаются автоматически в потоке. Например, если по TCP-соединению передается текст, разбитый на страницы, то TCP-агент не будет каким-либо образом обозначать конец каждой страницы.

После передачи приложением данных TCP-агенту, эти данные могут быть отправлены сразу на сетевой уровень, а могут быть буферизованы. Как поступить в этом случае решает TCP-агент. Однако в ряде случаев бывает необходимо, чтобы данные были отправлены

сразу, например если эти данные представляют собой команду для удаленной машины. Для этого в заголовке TCP-сегмента имеется флаг PUSH, и если он установлен, то это говорит TCP-агенту о том, что данные должны быть переданы немедленно.

Наконец, если в заголовке TPDU-сегмента установлен флаг URGENT, то TCP-агент передает такой сегмент незамедлительно. Когда срочные данные поступают к месту назначения, то их передают получателю немедленно.

3.3.3. Протокол TCP

Как уже говорилось, приложение передает на транспортный уровень поток байтов. Каждый байт в TCP-соединении имеет 32-разрядный номер. В сети с пропускной способностью 10 Мбит/с потребуется не менее часа, чтобы исчерпать все номера с 0 до 2^{32}. Эти номера используются как для уведомления, так и в механизме управления окнами. Нетрудно подсчитать, что в сети с пропускной способностью 100 Мбит/с эти номера будут исчерпаны менее чем за 10 мин, а в сети с пропускной способностью 1 Гбит/с — менее чем за минуту.

TCP-агенты обмениваются сегментами данных (TPDU-сегментами). Каждый сегмент имеет заголовок от 20 байт и более (по выбору) и тело переменной длины. Один сегмент может включать в себя байты от разных отправителей или часть данных от одного отправителя. Длину тела сегмента, определяемую TCP-агентом, ограничивают два фактора. Во-первых, длина сегмента не должна превышать максимальную длину IP-пакета, т. е. 64 Кбайт. Во-вторых, каждая сеть имеет максимальную единицу передачи — MTU (Maximum Transfer Unit), и каждый сегмент должен помещаться в MTU. В противном случае маршрутизаторам придется применять фрагментацию. При этом возрастают накладные расходы на передачу в транспортной среде, так как каждый фрагмент оформляется как самостоятельный пакет.

Основным протоколом, который используется TCP-агентом, является протокол скользящего окна. Это означает, что каждый посланный сегмент должен быть подтвержден. Одновременно с отправлением сегмента запускается таймер. Подтверждение придет либо с очередными данными в попутном направлении, если они имеются, либо без данных само по себе и будет иметь порядковый номер очередного ожидаемого получателем сегмента. Если таймер сработает прежде, чем придет подтверждение, то сегмент посылается повторно.

Несмотря на кажущуюся простоту, TCP-протокол достаточно сложен и должен решать следующие основные проблемы:

- восстанавливать порядок сегментов;
- убирать дубликаты сегментов, в каком бы виде они не поступали;

• определять разумную задержку для time-out на подтверждение в получении сегмента;
• надежно устанавливать и разрывать соединения;
• управлять потоком;
• управлять перегрузками.

3.3.4. Заголовок сегмента в TCP

Заголовок сегмента TCP-протокола показан на рис. 3.8.

Как уже говорилось, максимальная длина раздела данных составляет 65 495 байт, а заголовок включает в себя следующие поля:

Source port и **Destination port**, указывающие сокеты на стороне соответственно отправителя и получателя;

Sequence number и **Acknowledgement number**, содержащие порядковый номер ожидаемого байта и следующего ожидаемого байта, а не последнего полученного байта;

6-битовое поле флагов, содержащее следующие биты:

- URG, используемый вместе с полем **Urgent pointer**, которое указывает на начало области срочных данных;
- ACK, равный 1 если поле **Acknowledgement number** используется, а в противном случае равный 0;
- PSH, равный 1, если отправитель просит транспортного агента на стороне получателя сразу передать эти данные приложению и не буферизовать их;
- RST, используемый для переустановки соединения, которое по какой-либо причине стало некорректным. Получение сегмента с таким флагом означает наличие проблемы, с которой необходимо разбираться;

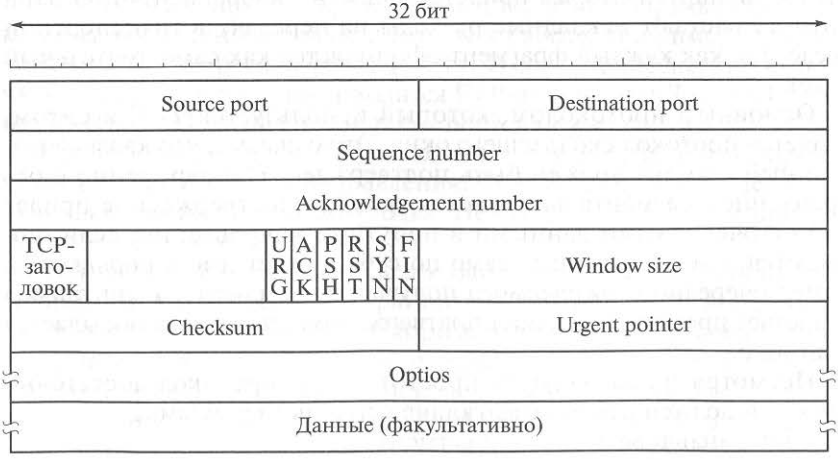


Рис. 3.8. Структура заголовка сегмента TCP-протокола

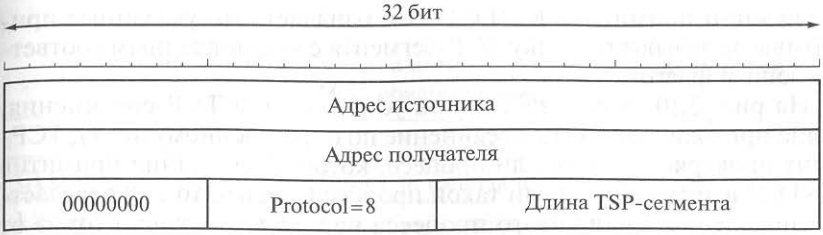


Рис. 3.9. Структура псевдозаголовка, включаемого в контрольную сумму TCP-сегмента

- SYN, равный 1 при запросе на соединение. Флаг ACK указывает наличие или отсутствие подтверждения. При SYN = 1 и ACK = 0 — запрос на соединение, а при SYN = 1 и ACK = = 1 — подтверждение соединения;
- FIN, используемый при запросе на разрыв соединения. Установка этого флага означает, что у отправителя нет больше данных;

Window size, используемое алгоритмом управления окном. В нем получатель указывает, сколько байтов может послать отправитель, начиная с последнего подтвержденного байта;

Options, используемое для возможностей, не предусмотренных стандартным заголовком. Например, здесь часто указывается максимальный размер поля данных, допустимый по данному соединению;

Checksum, используемое для обнаружения ошибок при передаче. Оно охватывает заголовок, данные и псевдозаголовок, показанный на рис. 3.9.

Псевдозаголовок содержит 32-разрядные IP-адреса отправителя и получателя, номера протокола (для TCP — 6) и число байтов в TCP-сегменте, включая заголовок.

Понятно, что для высокоскоростных каналов размер сегмента 64 Кбайт неэффективен: большую часть времени передачи отправитель будет ожидать подтверждений. Поэтому в RFC 1323 появился параметр «Масштаб окна», значение которого определяется при установке соединения. Этот параметр позволяет увеличить поле «Размер окна» до 14 разрядов влево, увеличивая при этом размер окна до 1 Гбайт.

3.3.5. Управление соединениями в TCP

Как уже говорилось, установление TCP-соединения происходит по протоколу «троекратного рукопожатия». Флаги SYN и ASK в заголовке сегмента используются для реализации примитивов CONNECTION REQUEST и CONNECTION ACCEPTED, а флаг RST — для

реализации примитива REJECT. Это означает, что указанные примитивы вызывают посылку TCP-сегмента с установленным соответствующим флагом.

На рис. 3.10, *а* показана схема установления TCP-соединения. Когда приходит запрос на соединение по определенному порту, TCP-агент проверяет, имеется ли процесс, который выполнил примитив LISTEN в этом порту. Если такой процесс имеется, то ему передается управление. Если такого процесса нет, то в ответ идет отказ от установления соединения.

На рис. 3.10, *б* показана ситуация, когда два хоста одновременно пытаются установить соединение между двумя одинаковыми сокетами (коллизия). Поскольку каждое соединение идентифицируется парой сокетов, то установлено будет только одно из соединений.

Таймер для последовательных номеров сегментов тактируется с частотой 4 мкс, максимальное время жизни сегмента — 120 с. Напомним, что начальный номер сегментов никогда не равен нулю (см. подразд. 3.2.3). Для генерации последовательных номеров сегментов используется механизм логических часов.

TCP-соединение, как уже говорилось, — дуплексное, т.е. в каждом направлении данные передаются независимо и соединение разрывается независимо по каждому направлению, поэтому лучше всего представлять его как два симплексных соединения. Если в очередном сегменте флаг FIN=1, то в этом направлении данных больше не будет. При получении подтверждения для данного сегмента соединение в этом направлении считается разорванным. В другом направлении передача может продолжаться сколь угодно долго. Если подтверждения на первый флаг FIN нет в течение двух интервалов жизни сегментов, то соединение считается разорванным по time-out. Противоположная сторона также по истечении этого периода времени узнает, что никто от нее не ждет ответа.

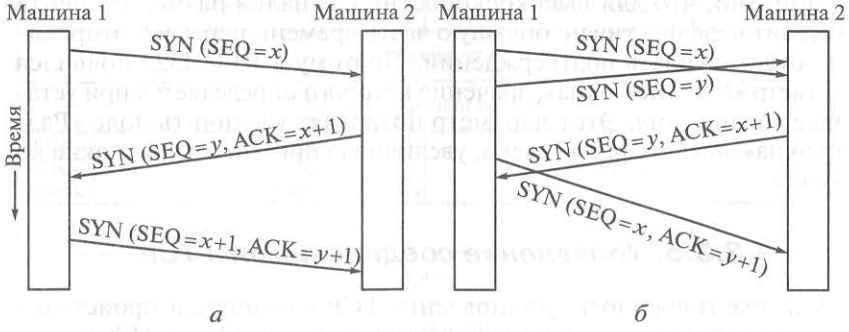


Рис. 3.10. Схемы установления TCP-соединения в одностороннем (*а*) и двухстороннем (*б*) порядке

Таблица 3.2

Состояния, используемые в конечном автомате управления TCP-соединениями

Состояние	Описание
CLOSED	Нет активных или ожидающих соединений
LISTEN	Сервер ожидает входящего вызова
SYN RCVD	Запрос на соединение доставлен; ожидание подтверждения
SYN SENT	Приложение открывает соединение
ESTABLISHED	Состояние нормальной передачи данных
FIN WAIT 1	Приложение сообщило об окончании работы
FIN WAIT 2	Другая сторона согласилась разорвать соединение
TIMED WAIT	Ожидание, пока все сегменты прекратят свое существование
CLOSING	Попытка обеих сторон одновременно закрыть соединение
CLOSE WAIT	Противоположная сторона инициировала разрыв
LAST ACK	Ожидание прекращения существования всех сегментов

В табл. 3.2 указаны состояния, используемые в конечном автомате управления TCP-соединениями.

На рис. 3.11 представлена процедура установления и разрыва соединения в виде диаграммы конечного автомата, описывающей изменение состояний клиента и сервера TCP в ходе обработки соединений.

Состояния клиента:

1. Первоначально соединение находится в состоянии CLOSED.
2. После выполнения клиентом запроса на открытие соединения (CONNECT) клиент устанавливает флаг SYN и переходит в состояние SYN SENT.
3. После получения от сервера флага SYN+ACK и его подтверждения клиент переходит в состояние открытого соединения (ESTABLISHED), в котором может передавать и принимать данные.
4. После выполнения вызова CLOSE клиент формирует сегмент FIN и переходит в состояние FIN WAIT 1.
5. Из состояния FIN WAIT 1 клиент переходит либо в состояние CLOSING по получении флага FIN и после отправки флага ACK, либо в состояние FIN WAIT 2 по получении только флага ACK, либо в состояние TIMED WAIT по получении флага FIN+ACK от сервера и после отправки флага ACK.

6. Из состояния CLOSING клиент переходит в состояние TIMED WAIT по получении флага ACK.

7. Из состояния FIN WAIT 2 клиент переходит в состояние TIMED WAIT по получении флага FIN и после отправки флага ACK.

8. Из состояния TIMED WAIT клиент переходит в состояние CLOSED по time-out (по истечении таймера времени жизни сегмента).

9. Если в состоянии ESTABLISHED клиент получил флаг FIN, то он отправляет флаг ACK и переходит в состояние CLOSE WAIT. В этом случае инициатором разрыва соединения выступает сервер.

10. В состоянии CLOSE WAIT клиент выполняет вызов CLOSE и отправляет флаг FIN, переходя в состояние LAST ACK.

11. После получения подтверждения от сервера соединение переходит в состояние CLOSED.

Состояния сервера:

1. Первоначально сервер находится в состоянии CLOSED, из которого переходит в состояние LISTEN после выполнения вызова

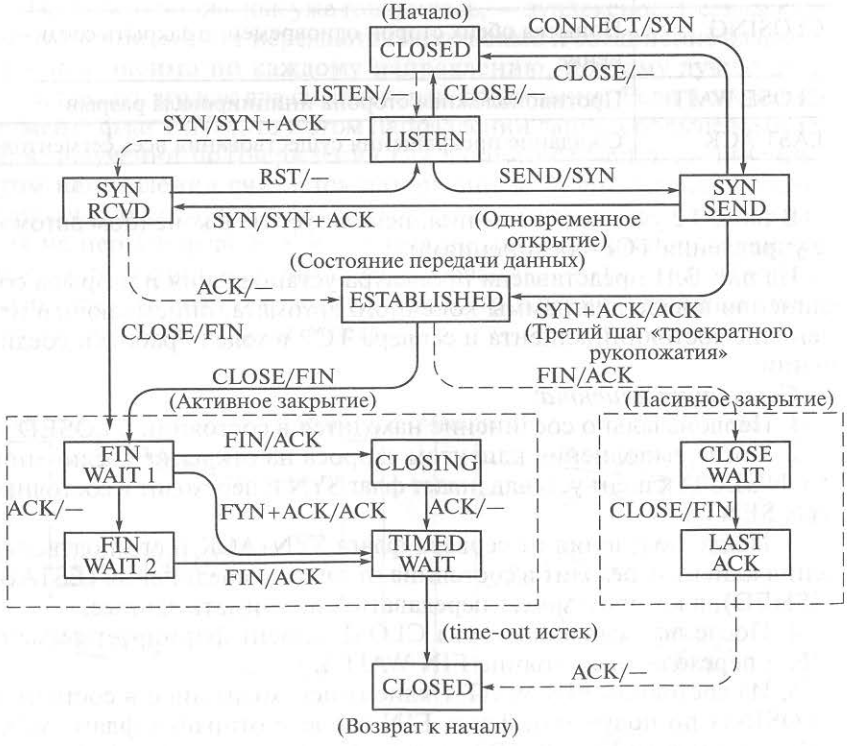


Рис. 3.11. Диаграмма состояний конечного автомата установления и разрыва TCP-соединения

LISTEN. В данном состоянии сервер находится до получения от клиента запроса на установление соединения.

2. После получения флага SYN сервер отправляет флаг SYN+ACK и переходит в состояние SYN RCVD.

3. В состоянии SYN RCVD сервер может либо отправить флаг RST и сразу перейти обратно в состояние LISTEN, либо выполнить вызов CLOSE, отправить флаг FIN и перейти в состояние FIN WAIT1, либо по получении флага ACK от клиента перейти в состояние ESTABLISHED.

4. Разрывы соединений для клиента и сервера выполняются одинаково.

3.3.6. Стратегия передачи в TCP

Управление окнами в протоколе TCP, как и управление потоком на канальном уровне, не связано прямо с поступлением подтверждений. Предположим, что у получателя есть буферы размером 4 096 байт. Если отправитель послал сегмент размером 2 048 байт, то получатель, получив и подтвердив этот сегмент, будет показывать окно размером 2 048 байт до тех пор пока приложение не возьмет часть данных из полученного сегмента размером 2 048 байт. Отправитель посылает следующие 2 048 байт. Теперь размер окна равен 0 байт.

При поле WIN = 0 отправитель может послать сегмент в двух случаях: если это данные URGENT (например, когда требуется «убить» процесс на удаленной машине) или если это однобайтовый сегмент. Это может потребоваться, чтобы заставить получателя показать текущее состояние буфера, что очень важно, так как позволяет обойти тупик, который мог бы возникнуть в случае потери объявления о размере окна (см. подразд. 3.2.5).

Заметим, что протокол TCP не требует от транспортного агента отправителя передавать сегмент сразу, как только данные поступили от приложения. Эту свободу TCP использует, чтобы повысить свою производительность. Рассмотрим к примеру удаленный текстовый редактор TELNET. Если передавать по сети каждое движение пользователя мышкой или нажатие им клавиши на клавиатуре, то обмен будет очень неэффективным. На передачу одного символа будет приходиться передача сегмента размером 160 байт. Поэтому часто при реализации протокола TCP вводят специальную задержку на посылку подтверждения и состояния окна, чтобы дать отправителю накопить буфер для отправки. Другую стратегию предложил Нагл [64]: если работа идет с приложением, которое генерирует однобайтовые сообщения, надо первый байт послать, а все остальные буферизовать до тех пор пока не придет подтверждение на посланный байт. Все буферизованные байты следует послать одним сегментом, после чего буферизовать все байты, пока не придет подтверждение на посланный сегмент.

Алгоритм Нагла работает хорошо. Однако есть приложения, где его следует отключать, например X-Windows. Здесь перемещения мыши по экрану надо пересылать сразу без буферизации.

Существенно снизить производительность протокола TCP может также так называемый синдром дурацкого окна. Это явление возникает, когда приложение на стороне отправителя передает TCP-агенту данные большими блоками, а приложение на стороне получателя читает данные побайтово. В этой ситуации может произойти следующее. Буфер на стороне получателя полон, и отправитель знает об этом. Приложение-получатель считывает 1 байт из буфера. TCP-агент получателя радостно посылает сообщение о доступном буфере размером 1 байт. Отправитель обязан послать 1 байт. После чего буфер получателя опять полон и т. д.

Кларк [32] предложил запретить сообщать получателю в таких случаях об освободившемся месте размером 1 байт. Получателя принуждают ждать, пока либо не освободится место размером, равным максимальной длине сегмента, объявленной при установлении соединения, либо пока не освободится половина буфера. Отправитель также должен стараться избегать крохотных сегментов, а посылать большие.

У TCP-агента получателя также есть средства улучшить производительность соединения. Например, можно запретить приложению использовать примитив READ до тех пор, пока у TCP-агента есть данные в буфере. Конечно, такое решение увеличит время отклика, но для неинтерактивных приложений это не страшно.

Проблемой для получателя, понижающей производительность соединения, является также нарушение порядка поступления сегментов. Например, если поступили сегменты 0...7, получатель может подтвердить сегменты 0...2 и буферизовать сегменты 4...7. Тогда отправитель по time-out перешлет сегмент 3, после чего получатель подтвердит сегменты 4...7.

3.3.7. Управление перегрузками в TCP

Рассмотрим, как протокол TCP борется с перегрузками. В основе всех методов лежит принцип сохранения числа сегментов: не посылать новый сегмент, пока старый не покинет сеть, т. е. не будет доставлен. Основная идея очень проста — при возникновении перегрузки не посылать новых сегментов. В протоколе TCP это реализуется динамически с помощью механизма окон.

Прежде всего протокол TCP обнаруживает перегрузку по росту числа возникновения time-out. Если это значение превышает некоторый предел, являющийся параметром протокола, фиксируется перегрузка. Причин перегрузки может быть две: шум в канале и сброс сегментов маршрутизатором, различить которые сложно. В наши дни каналы достаточно надежные, а вторая причина остается актуальной.

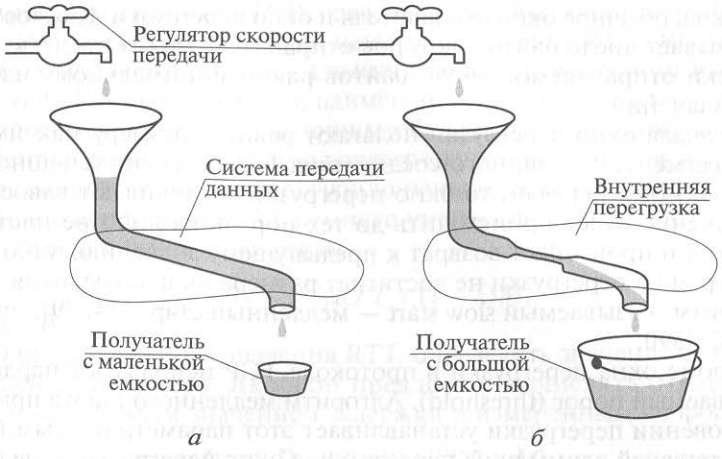


Рис. 3.12. Иллюстрация причин перегрузок:

а — недостаточная емкость получателя; *б* — недостаточная емкость сети

На рис. 3.12 [19] дана иллюстрация причин перегрузок. Перегрузки возникают по двум причинам: нехватка буфера на стороне получателя, т.е. недостаточная емкость получателя (рис. 3.12, *а*); перегрузка внутри сети, т.е. недостаточная емкость сети (рис. 3.12, *б*).

В Интернете эти случаи различают как внутреннюю емкость сети и емкость получателя, поэтому каждый отправитель поддерживает

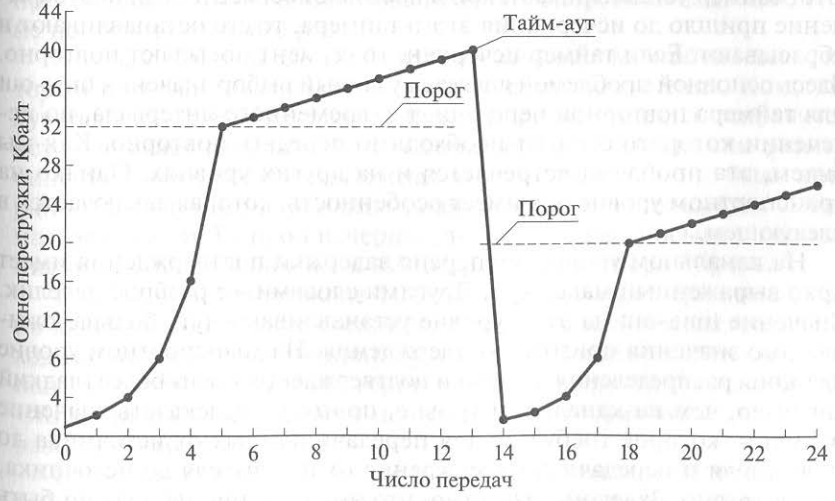


Рис. 3.13. Алгоритм управления перегрузками в Интернете

два окна: обычное окно отправителя и окно перегрузки. Каждое окно показывает число байтов, которое отправитель может послать. Фактически отправляемое число байтов равно минимальному из этих двух значений.

Сначала окно перегрузки полагают равным размеру максимального сегмента для данного соединения. Если сегмент успешно (без time-out) был передан, то окно перегрузки увеличивают вдвое. Это увеличение будет происходить до тех пор, пока либо не наступит time-out и произойдет возврат к предыдущему значению, либо пока размер окна перегрузки не достигнет размера окна получателя. Этот алгоритм, называемый slow start — медленный старт [54, 59], описан в RFC 3390.

Кроме окна перегрузки в протоколе TCP используют параметр, называемый порог (threshold). Алгоритм медленного старта при возникновении перегрузки устанавливает этот параметр равным половине текущей длины окна перегрузки, а окно перегрузки — не более максимального сегмента для данного соединения. Окно перегрузки растет экспоненциально до тех пор пока не сравняется с порогом, после чего оно растет линейно пока не достигнет размера окна получателя. На этом рост прекращается до первой перегрузки. Работа этого алгоритма показана на рис. 3.13.

3.3.8. Управление таймером в TCP

Протокол TCP использует несколько таймеров для управления передачей. Наиболее важный из них — *таймер повторной передачи*. Этот таймер устанавливают, когда посылают сегмент. Если подтверждение пришло до исчерпания этого таймера, то его останавливают и сбрасывают. Если таймер исчерпан, то сегмент посылают повторно. Здесь основной проблемой является удачный выбор значения time-out для таймера повторной передачи, т. е. временно́го интервала, по истечении которого сегмент необходимо передать повторно. Как мы знаем, эта проблема встречается и на других уровнях. Однако на транспортном уровне она имеет особенность, которая заключается в следующем.

На канальном уровне дисперсия задержки подтверждения имеет ярко выраженный максимум. Другими словами, ее разброс невелик. Значение time-out на этом уровне устанавливают чуть больше ожидаемого значения прихода подтверждения. На транспортном уровне функция распределения задержки подтверждения носит более гладкий характер, чем на канальном уровне, поэтому предсказать значение времени, которое требуется для передачи данных от источника до получателя и передачи подтверждения от получателя до источника, очень трудно. Заведомо известно, что это значение не должно быть постоянным в силу гладкости функции распределения.

Алгоритм протокола TCP, предложенный Якобсоном в 1988 г. [54, 59], основывается на специальной переменной RTT (Round Trip Time), которая используется для вычисления оптимального значения time-out. Это значение равно наименьшему времени подтверждения, которое вычисляется следующим образом. При каждой передаче сегмента измеряется задержка подтверждения M. Если при очередной передаче подтверждение поступило раньше, чем наступил time-out, значение переменной RTT немного уменьшают, а в противном случае — увеличивают по формуле

$$RTT = \lambda RTT + (1 - \lambda)M,$$

где $\lambda = 0{,}87$.

Однако, даже зная значения RTT, определить значение ожидания оказалось непросто. Якобсон предложил вычислять отклонение между ожидаемым значением задержки и измеренным по формуле

$$D = \lambda D + (1 - \lambda)\,|RTT - M|.$$

Кроме того, он показал, как, зная D, вычислить значение time-out:

$$time\_out = RTT + 4*D.$$

Позднее Ф. Карн модифицировал эту формулу для случая повторно передаваемых сегментов [57]. Действительно, когда поступает подтверждение для сегмента, то неясно, это подтверждение для первой передачи или для последней. Ф. Карна интересовал вопрос вычисления значения RTT для передачи данных с помощью протоколов TCP/IP по радиоканалу в коротковолновом диапазоне. Эксперименты показали, что для повторно передаваемых сегментов эффективно просто удваивать значение ожидания до тех пор, пока подтверждение не поступит с первого раза.

Важным в протоколе TCP является *таймер настойчивости*, который позволяет бороться с тупиками следующего типа. Когда получатель посылает сообщение с нулевым размером окна, отправитель останавливает передачу и ждет сообщения об изменении размера окна. Наконец, получатель послал это сообщение, а оно было потеряно. Все ждут. Во избежание такой ситуации используется таймер настойчивости. Если он исчерпан, то отправитель посылает сообщение получателю, напоминая ему о проблеме размера буфера.

Также важен в протоколе TCP *таймер функционирования*. Если по какой-либо причине соединение долго не использовалось, необходимо проверить, функционирует ли оно. Когда этот таймер исчерпан, то соответствующая сторона посылает другой стороне запрос «Жива ли ты?» Если ответа не поступает, то соединение считается разорванным.

Итак, мы видим, что протокол TCP использует большое число таймеров, управление которыми требует определенных вычислительных затрат и является не тривиальной задачей.

3.3.9. Протокол UDP

Как уже неоднократно отмечалось, Интернет поддерживает также транспортный протокол без соединений — UDP, предназначенный для обмена дейтаграммами между процессами на абонентских машинах, входящих в единую сеть с коммутацией пакетов, который описан в RFC 768. В качестве протокола нижнего уровня в UDP-протоколе используется протокол IP.

Протокол UDP предоставляет прикладным программам возможность отправлять сообщения другим приложениям, используя минимальное число параметров протокола. При этом протокол UDP не обеспечивает достоверность доставки пакетов, защиту от дублирования данных или от сбоев в передаче. За исключением параметров приложения — номеров портов отправителя и получателя пакета — UDP практически ничего не добавляет к IP-дейтаграмме.

Протокол UDP намного проще, чем TCP, и полезен в ситуациях, когда мощные механизмы обеспечения надежности протокола TCP не требуются или будут только помехой для решения определенного рода задач, например аутентификации пользователей. Его используют многие приложения, например почтовый протокол SMTP, протокол виртуального терминала Telnet, протокол всемирной паутины HTTP, протокол передачи файлов FTP, протокол управления сетью SNMP, протокол службы доменных имен DNS и многие другие (см. гл. 4).

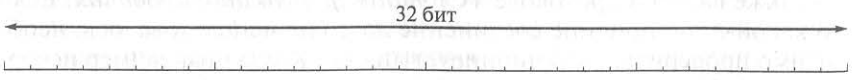
Структура UDP-заголовка показана на рис. 3.14:

Source Port (16 бит) — порт отправителя. Это поле может содержать номер порта, с которого был отправлен пакет, когда это имеет значение (например, когда отправитель ожидает ответа). Если это поле не используется, оно заполняется нулями;

Destination Port (16 бит) — порт назначения, т.е. порт компьютера, на который пакет будет доставлен;

Length (16 бит) — поле длины. Длина (в байтах) этой дейтаграммы, включая заголовок и данные. Минимальное значение этого поля равно 8;

Checksum (16 бит) — поле контрольной суммы, которая вычисляется так же как и в протоколе TCP, включая псевдозаголовок. Контрольная сумма UDP-пакета представляет собой побитовое до-



Source port	Destination port
UDP length	UDP checksum

Рис. 3.14. Структура UDP-заголовка

полнение 16-битовых суммы 16-битовой слов (аналогично TCP). В вычислении участвуют данные пакета, заголовок UDP-пакета, псевдозаголовок (информация от IP-протокола), поля выравнивания по 16-битовой границе (нулевые).

Преимущество протокола UDP состоит в том, что он требует минимум установок параметров для соединения двух процессов между собой. Этот протокол используется при работе серверов доменов (Name Servers), при работе протокола TFTP (Trivial File Transfer), при работе с SNMP-протоколом и при построении систем аутентификации. Идентификатор UDP в IP-заголовке — число 17. Более подробное описание протокола UDP приведено в RFC 768.

3.3.10. TCP и UDP в беспроводных коммуникациях

Теоретически TCP не должен зависеть от того, над какой средой он работает: оптической или беспроводной. Однако на практике дело обстоит иначе. TCP-протокол тщательно оптимизировался при разных предположениях, которые не выполняются в беспроводной среде. Так, например, при разработке TCP предполагалось, что сетевая среда достаточно надежна и рост числа time-out — это результат перегрузки, а не потери пакетов.

В беспроводной среде потеря пакета — явление частое, поэтому применение протокола TCP «в лоб» с алгоритмом медленного старта Якобсона лишь усугубит положение. Так, например, если потери составляют 20 %, то при пропускной способности канала 100 пакетов в секунду фактически будут проходить лишь 80 пакетов. Согласно алгоритму медленного старта при увеличении числа time-out надо снизить скорость, например до 50 пакетов в секунду, что определит фактическую скорость 40 пакетов в секунду. Поэтому если число отказов увеличилось в обычном канале, необходимо сбросить скорость, а если это число возросло в беспроводной среде, следует не снижать скорость, а посылать пакеты повторно. Так что без знания того, в какой среде происходит передача, принять правильное решение трудно.

Основным источником рассмотренной проблемы является то, что в современных сетях передачи данных соединения часто неоднородные, т. е. в них могут встречаться как проводные каналы (коммутируемые и некоммутируемые), так и беспроводные. Для неоднородных транспортных соединений была предложена модификация TCP, идея которой состояла в том, чтобы разбивать такое соединение на два соединения и чтобы каждое из них стало однородным. В этом случае в ответ на time-out в первом соединении отправитель может снизить скорость, а во втором — увеличить. Другие параметры этого протокола также могут настраиваться независимо в каждом из этих двух соединений.

Недостатком такого решения является потеря концептуальной целостности. По сути TCP — одно соединение. Однако теперь мы должны иначе трактовать подтверждения получения сегментов. Кроме того, шлюзы на границах неоднородности должны иметь TCP-агентов, чтобы устанавливать, управлять и разрывать промежуточные TCP-соединения.

Имеются и другие решения этой проблемы, связанные с модификацией не самого TCP, а сетевого уровня в шлюзах [19, 59, 71].

3.4. Вопросы производительности сети

3.4.1. Проблемы производительности в сетях

Производительность вычислительных сетей и машин — один из основных показателей их эффективности. Сегодня настройка производительности сетей и систем больше искусство, чем наука. Многое из того, что здесь изложено — результат практики и относится к сфере деятельности сетевых и системных администраторов. Мы уже рассмотрели вопросы производительности, например сетевого уровня. Однако вопросы производительности сети в целом как системы относятся к транспортному уровню.

Рассмотрим пять основных аспектов производительности сетей:

- причины падения производительности;
- измерение производительности;
- влияние организации транспортной среды на производительность;
- быстрая обработка TPDU-сегментов;
- протоколы для высокопроизводительных сетей.

Одной из причин резкого падения производительности в сетях являются перегрузки вследствие несбалансированности ресурсов и нагрузки. Если нагрузка на маршрутизаторы больше, чем они могут переработать, возникает перегрузка (см. подразд. 2.3.7).

Производительность сети может падать вследствие *структурной несбалансированности ресурсов*. Например, если персональную машину подключить к гигабитному каналу, то она будет захлебываться от наплыва пакетов из-за несбалансированности скорости процессора и скорости канала.

Так называемые *синхронные перегрузки* являются синхронной реакцией на некоторые действия в сети. Например, если TPDU-сегмент содержит неверный параметр (номер порта или процесса), то в ответ пойдет сообщение об ошибке. Если такой сегмент получили сотни или тысячи машин, то в ответ последует ураган сообщений. Этой проблеме был подвержен протокол UDP до тех пор, пока в него не было внесено изменение, которое определяло случаи, когда разрешается, а когда запрещается посылать подтверждения. Другим

примером является перезагрузка машин в сети после сбоя питания. В этом случае все машины разом ринутся за надлежащей информацией на RARP-сервер и файл-сервер, в результате чего произойдет коллапс серверов.

Другой причиной падения производительности в сетях является несоответствующая настройка системы. Например если машины в сети имеют достаточно мощные процессоры и достаточно памяти, но под буферы памяти выделено мало, пакеты будут теряться из-за переполнения буферов. Также сегменты будут теряться, если планировщик процессов в операционной системе не дает достаточно высокий приоритет процессу обработки TPDU.

Параметром настройки является и время time-out. Если time-out на подтверждение слишком короткое, то будет много повторных посылок, а если большое — скорость передачи упадет из-за простоев ожидания. Например если время ожидания попутного сообщения, с которым можно отправить подтверждение о полученном сегменте, маленькое, то будет много дополнительных сегментов-уведомлений, а если большое — получатель может генерировать запросы по time-out, а значит, скорость передачи упадет.

Появление гигабитных сетей определило новые причины потери производительности сетей. Пусть необходимо передать пакет из Москвы во Владивосток. У отправителя имеется линия с пропускной способностью 1 Гбит/с, а у получателя — буфер размером 64 Кбайт. Пусть задержка сигнала на передачу в одном направлении по оптоволоконному каналу составляет в этом случае около 40 мс. В этом случае через 0,5 мкс все 64 Кбайт данных будут в канале, и отправитель должен будет остановиться и ждать подтверждения. В результате пропускная способность канала будет использована примерно на 0,6 %! Дело в том, что буфер у получателя должен иметь размер, равный произведению пропускной способности канала и значения задержки, т. е. в нашем случае примерно 5 Мбайт. На практике буфер должен быть даже чуть больше. Ведь получатель может не сразу отреагировать на поступившие данные.

3.4.2. Измерение производительности в сети

Когда пользователи сети обнаруживают падение производительности их приложений, они обращаются к администратору сети с жалобами. Последний обязан выяснить, что случилось, и принять необходимые меры. Типичный порядок действий при исправлении производительности сети следующий:

1. Измерить надлежащие параметры сети и производительность;
2. Постараться понять, что происходит;
3. Изменить один параметр.

Эти шаги надо повторять либо до тех пор пока не удастся повысить производительность, либо пока не станет ясно, что имеющимися ресурсами этого сделать нельзя.

При этом измерения можно проводить в разных местах и разными способами. Основная идея всех измерений состоит в том, чтобы запустить какую-то активность и измерить, как долго она продолжается, и определить, какие события ее сопровождают. Измерение длительностей и сбор информации о событиях таят много подвохов. Приведем лишь некоторые из них.

1. **Количество испытаний должно быть достаточно велико.** Измерить время доставки одного TPDU-сегмента недостаточно. Это надо проделать миллионы раз. Тогда вычисление среднего значения и дисперсии будет свободно от влияния случайных факторов. В курсе математической статистики можно посмотреть, как выполняются такие вычисления.

2. **Выборка испытаний должна быть репрезентативной.** Проводить испытания необходимо в разное время дня и года. Что толку измерять производительность сети в университете в августе? Если измерения проводятся с 12 до 14 часов, они опять-таки неточны, поскольку в это время люди часто уходят на обед.

3. **Следует учитывать разрешающую способность часов.** Как правило, таймер машины работает от сети 50 Гц, поэтому измерять моменты наступления событий, происходящих чаще чем через 20 мс, им нельзя. Однако если измерить интервал, в котором произошло миллион регулярных событий, то можно вычислить необходимое (среднее) значение.

4. **Ничего неожиданного во время измерений происходить не должно.** Следует быть уверенным, что измерения происходят при «типичных» нагрузках, т. е. что нет единичных всплесков активности, например лабораторных работ. При этом нельзя быть уверенным, что все «тихо» только потому, что измерения происходят в три часа ночи, хотя бы потому, что архивация данных в локальной сети обычно происходит именно по ночам.

5. **Кэш-буфер может разрушить ваши измерения.** Пусть, например, мы хотим измерить время передачи файла. Для этого необходимо его открыть, передать, закрыть и измерить общее время. Сделать это надо много раз. Однако если файл меньше размера кэш-буфера, мы будем измерять скорость работы кэш-буфера, и только первое сделанное измерение будет показывать производительность сети. Чтобы избежать этого эффекта, необходимо выбирать файлы достаточно большого размера по сравнению с размером кэш-буферов.

Аналогичное влияние может оказывать буферизация. Например если UDP получает подтверждение от сетевого уровня, как только сетевой уровень получил дейтаграмму, и на сетевом уровне есть буфер на 1 000 дейтаграмм, то, проведя 999 испытаний, мы получим скорость

передачи, которая может оказаться выше, чем пропускная способность физического канала. В действительности проведенный эксперимент будет показывать скорость буферизации на сетевом уровне.

6. **Необходимо четко осознавать, что вы измеряете.** Когда измеряется время считывания удаленного файла, то эти измерения зависят от СПД транспортной среды, операционных систем клиента и сервера, их сетевых плат, драйверов и т. п. Если необходимо настроить взаимодействие в сети при конкретной конфигурации, то такое измерение имеет смысл. Если эти измерения будут использованы для выбора сетевых карт, то собранные таким образом данные не годятся. Например, может оказаться, что драйвер, используемый в измеряемой конфигурации, работает отвратительно с выбранной сетевой платой.

7. **Следует быть очень осторожным при экстраполяции результатов.** Проведя измерения при определенной нагрузке, следует быть очень осторожным при их экстраполяции. Во многих случаях предположение о линейной экстраполяции может быть неверным. Вообще следует отметить, что многие параметры, или, как еще говорят, индексы производительности, немасштабируемы. Их зависимость от многих параметров функционирования сетей до сих пор неизвестна и не имеет аналитического выражения. Поэтому с экстраполяцией результатов, полученных при одной конфигурации сети, на случай другой конфигурации надо быть крайне осторожным.

3.4.3. Правила, улучшающие производительность

Повышать производительность существующей сети можно лишь в определенных пределах. Куда большие возможности для этого имеются при проектировании новой сети. Приведем некоторые правила, определенные исключительно на опыте создания многих сетей.

Правило 1: скорость процессора важнее, чем емкость сети. Здесь полезно вспомнить законы Мура и Гилдера. Опыт показал, что накладные расходы на работу операционной системы и стека протоколов значительно больше, чем накладные расходы на передачу по физическому каналу [59, 71]. Согласно [19] теоретически RPC (удаленный вызов процедуры) в Интернете должен занимать не более 102 мкс. На практике редко удается уменьшить это время до 1 500 мкс. Основные задержки происходят в системном программном обеспечении.

Аналогично при использовании высокоскоростных каналов важно повысить скорость доставки байта до канала и его обработку при получении. Другими словами, удвоив скорость процессора на хосте, можно удвоить пропускную способность сети, а удвоив емкость сети, можно ничего не получить, так как узким местом будет программное обеспечение хоста.

Правило 2: сокращай число пакетов, чтобы снизить накладные расходы программного обеспечения. При обработке TPDU-сегментов часть накладных расходов приходится на обработку самого TPDU, например заголовка сегмента, а часть — на обработку тела TPDU, например на подсчет контрольных сумм. Поэтому при использовании TPDU размером 128 байт накладные расходы в 32 раза выше, чем при использовании TPDU размером 4 Кбайт.

На нижних уровнях сети большое влияние имеют прерывания, сопровождающие поступление кадра, пакета. В RISC-процессорах они «ломают» конвейер, вызывают переключение контекста в операционной системе, изменяют содержимое кэш-памяти и т. д. Заметим, что алгоритмы Нагла и Кларка для «дурацкого окна» сокращают именно число прерываний (см. подразд. 3.3.6).

Правило 3: минимизируй переключение контекста. Минимизировать переключения контекста удается с помощью специальных процедур, позволяющих накапливать пакеты и сообщения в буферах, прежде чем передавать их процессу более высокого уровня. Здесь большое значение имеет то, как организована работа с сетью в операционной системе. Например, может оказаться, что при поступлении пакета управление от прикладного процесса будет передано ядру операционной системы, от ядра сетевому менеджеру, от сетевого менеджера опять ядру и только потом от ядра процессу-получателю.

Правило 4: минимизируй число копирований. Куда больший ущерб производительности, чем переключение контекста, наносит множественное копирование одного и того же пакета. Обычно пакет, принятый сетевым интерфейсом, сначала буферизуют на сетевой плате, потом в ядре на сетевом уровне, затем в буфере транспортного уровня и лишь только после этого в буфере приложения!

Хорошая операционная система буферизует пакет со скоростью один такт на слово. Однако обычно операция буферизации занимает на менее пяти команд на слово. При процессоре, выполняющем 50 MIPS (миллионов команд в секунду), создание трех копий при пяти командах на 32-битовое слово потребует 75 нс на 1 байт. Таким образом, максимальная скорость приема данных будет не более 10^7 Мбит/с, а учитывая обработку заголовков, прерываний, переключение контекстов, — не более 50 Мбит/с. Очевидно, что и речи быть не может о работе с таким процессором на линии с пропускной способностью 1 Гбит/с.

Отметим также, что 50 Мбит/с — это скорость приема данных без учета обращений в память, а с учетом таких обращений она будет раза в три меньше, т. е. около 16 Мбит/с.

Правило 5: увеличение пропускной способности не сократит задержку. Если вам необходима пропускная способность, вы можете купить ее, например проложив еще один кабель. Однако это не сократит задержку при передаче. Для этого требуется улучшение про-

граммного обеспечения стека протоколов, системного программного обеспечения и сетевых интерфейсов.

Правило 6: лучше избегать перегрузок, чем восстанавливаться после них. При перегрузках теряются пакеты, расходуется пропускная способность, возникают бесполезные задержки и т. п. Восстановление после всего этого требует времени и усилий. Поэтому лучше избегать перегрузок.

Правило 7: избегайте наступления time-out. Таймеры неизбежны в сетях. Однако пользоваться ими следует очень аккуратно, и минимизировать при этом их (time-out) наступление, поскольку это влечет необходимость выполнения затем специальных действий. Поэтому установка значения таймера требует тщательных измерений, а уточнять начальное значение надо осторожно и постепенно.

3.4.4. Быстрая обработка TPDU-сегментов

Из всего сказанного напрашивается один вывод: основным препятствием для быстрой работы сети является программное обеспечение стека протоколов. Рассмотрим некоторые способы ускорения работы этого программного обеспечения.

Затраты на обработку TPDU-сегментов подразделяют на затраты на обработку каждого TPDU-сегмента и затраты на обработку каждого байта. Посмотрим, как можно сократить оба вида этих затрат. Для ускорения обработки собственно TPDU-сегмента можно использовать следующую идею. Основная доля TPDU-сегментов обрабатывается в режиме Established (см. рис. 3.11), поэтому важно максимально ускорить обработку TPDU-сегмента в этом режиме, для чего необходимо уметь быстро отличать этот (назовем его типичным) случай от остальных специальных случаев, например от разрыва соединения.

Рассмотрим пример. Для простоты предположим, что транспортный агент расположен в ядре операционной системы (рис. 3.15). На самом деле эта идея применима и для других случаев, например если транспортный агент является частью прикладного процесса или библиотечной функцией. На стороне отправителя прикладной процесс через программное прерывание передает TPDU-сегмент транспортному агенту в ядре. Агент с помощью проверок определяет, во-первых, какой случай имеет место: типичный (отправка TPDU-сегмента) или специальный (разрыв соединения); во-вторых, то что отправляется регулярный TPDU-сегмент, а не специальный; в-третьих, что окно получателя имеет достаточный размер. Если все эти условия выполнены, то можно запускать ускоренный процесс отправки.

В типичном случае заголовки последовательных TPDU-сегментов почти одинаковые. Чтобы воспользоваться преимуществом этого факта, транспортный агент сохраняет образец заголовка у себя при запуске процедуры быстрой обработки. Обычно это делают макси-

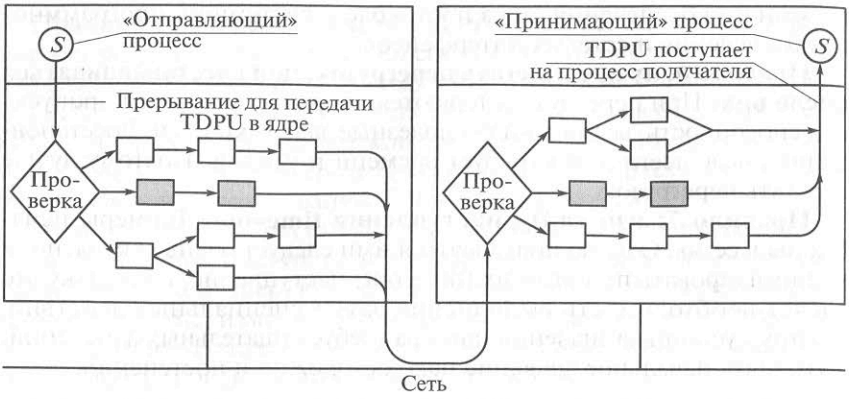
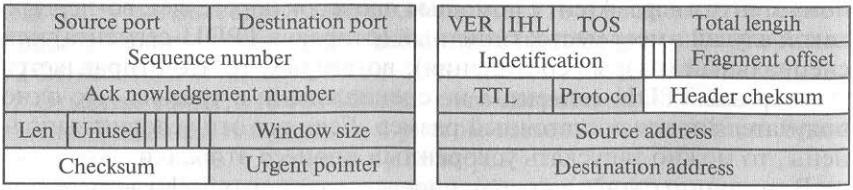


Рис. 3.15. Схема быстрой обработки TDPU-сегмента

мально быстро на регистровом буфере. Поля TPDU-сегмента, которые изменяются, переписываются в буфере. Затем указатель на TPDU-сегмент и указатель на тело данных передаются сетевому агенту на сетевой уровень, где может быть применена та же схема, после чего сетевой агент передает пакет на канальный уровень.

Рассмотрим, как эта идея работает в случае использования стека протоколов TCP/IP. На рис. 3.16, *а* показан заголовок TPDU, где серым тоном выделены поля, которые изменяют свои значения от сегмента к сегменту. Пять слов образца TPDU-сегмента копируют в буфер, вычисляют контрольную сумму, увеличивают порядковый номер. IP-процедура, в свою очередь, копирует пять слов образца (рис. 3.16, *б*) своего заголовка, заполняя соответствующими данными поля Identification и Header Checksum.

Теперь рассмотрим, что происходит на стороне получателя. Прежде всего транспортный агент на стороне получателя должен найти запись о соединении для поступившего TPDU-сегмента. Для протокола TCP эта запись может храниться в хеш-таблице. Ключом к этой таблице может служить информация о портах отправителя и получателя и их IP-адресах. Другой подход к поиску записи о соединении предложил



а *б*

Рис. 3.16. Структуры TCP-заголовка (*а*) и IP-заголовка (*б*)

Кларк: использовать последнюю использованную запись [33]. Как показала практика, эта эвристика работает хорошо.

Затем выполняются проверки, чтобы убедиться, что имеет место типичный случай, т. е. что нет разрыва соединения, нет URGENT-флага и т. п. В состоянии Established данные копируются в приложение, и при этом вычисляется контрольная сумма. Если контрольная сумма правильная, то корректируется запись о соединении, формируется подтверждение о получении и посылается отправителю.

Описанная в общих чертах схема, имеющая название *предвидение заголовка*, используется во многих реализациях.

Ускорение работы сети возможно также за счет оптимизации управления буферизацией и таймерами.

Основная идея ускорения при управлении буферизацией — избегать излишнего копирования.

Управление таймерами основывается на том, что хотя таймер устанавливается для каждого TPDU-сегмента, срабатывает он лишь для немногих из них. Общая схема, оптимизирующая работу с таймерами, заключается в следующем. Записи о таймерах связываются в список. В очередном элементе списка указывается, сколько тактов от срабатывания предыдущего таймера должно пройти, чтобы сработал текущий таймер. Поэтому, если есть три таймера, которые должны сработать в моменты тактов 3, 10 и 12, то список соответственно будет иметь вид 3, 7, 2. При такой организации достаточно при каждом такте корректировать только первую запись в списке, а не все таймеры.

Имеется и другой способ оптимизации работы с таймерами, называемый *колесом времени*. Его применяют, когда заранее известно максимально возможное значение таймера. В этом случае используют массив, длина которого пропорциональна максимальной длине временно́го интервала, соответствующего максимальному значению таймера. Каждый элемент в этом массиве соответствует одному такту часов. Если на текущий такт времени было запланировано окончание интервала time-out, то в ячейке таблицы, соответствующей текущему моменту, указатель устанавливается на список таймеров, которые должны сработать в этот момент.

3.4.5. Протоколы для гигабитных сетей

В начале 1990-х гг. стали появляться гигабитные сети, т. е. коммуникационное оборудование, способное работать на таких скоростях. Первой естественной реакцией специалистов было использовать на этом оборудовании те же протоколы, что и для обычных сетей. Однако сразу возникло довольно много неожиданных проблем. Рассмотрим некоторые из этих проблем и направления развития новых протоколов, которые должны помочь их решить.

Первой проблемой является длина поля для последовательной нумерации. Например, 16- или 32-разрядные поля для Ethernet с пропускной способностью 10 Мбит/с будут исчерпаны даже при нумерации каждого байта менее чем за сутки. Здесь неявно предполагалось, что так долго сегмент существовать в сети не может. При скорости 1 Гбит/с 32-разрядный счетчик будет исчерпан за 32 с. Среднее время жизни сегмента в Интернете составляет около 120 с. Безусловно, оно тоже сократится, но не настолько. Ведь основную долю этого времени занимают не передачи сегмента, а его обработка. Можно, конечно, нумеровать не каждый байт, а лишь сегменты, но это ненамного облегчит положение. Причина этой проблемы состоит в том, что предположение о времени жизни сегмента, ранее верное, теперь при скорости 1 Гбит/с неверно!

Вторая проблема возникает из-за того, что, как уже говорилось, скорость передачи растет быстрее производительности процессоров. Согласно законам Мура и Гилдера (см. гл. 1 т. 1 данного учебника) пропускная способность каналов растет в 3 раза быстрее, чем производительность средств обработки. В начале 1970-х гг. скорость передачи составляла порядка 56 Кбит/с, быстродействие процессоров — 1 MIPS, размер пакета — 1 Кбит. Это означает, что если пакеты поступали со скоростью 56 пакетов в секунду (со скоростью времен ARPANET), то на обработку одного пакета отводилось 18 мс или 18 000 команд процессора. Если оставлять 50 % производительности под приложение, то получим 9 000 команд на обработку одного пакета, что весьма неплохо.

В настоящее время при скорости передачи 1 Гбит/с, производительности процессора 100 MIPS и размере пакета 4 Кбайт пакеты следуют со скоростью 30 000 в секунду, т. е. на обработку одного пакета остается не более 15 мкс, если оставить половину производительности для приложения. За 15 мкс процессор со скоростью 100 MIPS успеет выполнить 1 500 команд. При этом необходимо учитывать, что функционально команда RISC-процессора беднее команды CISC-процессора. Следовательно, на работу сетевого программного обеспечения остается все меньше времени, а значит, протоколы должны становиться проще!

Третья проблема состоит в том, что гигабитный канал принципиально отличается от мегабитного. Гигабитный канал более чувствителен к значению задержки, чем к скорости передачи.

На рис. 3.17 показана зависимость времени передачи от скорости передачи для файла размером 1 Мбит на расстояние 4 000 км, из которого видно, что, начиная с гигабитных скоростей, увеличение пропускной способности не обеспечивает существенного прироста скорости передачи. Следовательно, протоколы старт-стопного типа, например RPC (удаленного вызова процедуры), будут существенно ограничены в производительности на таких каналах. То же самое можно сказать о протоколах с откатами, которые применяются, на-

Рис. 3.17. Зависимость времени передачи от скорости передачи для файла размером 1 Мбит на расстояние 4 000 км

пример на канальном уровне, когда часть пакетов в последовательности была утеряна. Откат увеличивает задержку в канале и увеличивает объем повторно передаваемой информации.

Четвертая проблема — это скорее проблема, вызванная новыми приложениями, чем проблема собственно протоколов. Причем это проблема приложений, для которых важно не среднее значение задержки, а минимальное отклонение от среднего значения, т. е. максимально равномерный поток данных. Примером таких приложений являются многие мультимедиа-приложения.

Теперь рассмотрим, что означают гигабитные скорости для организации протоколов. Раньше протоколы создавались, прежде всего, с учетом минимизации затрат пропускной способности для служебных целей: поля делали маленькими насколько это было возможно и упаковывали их плотно в байты. В настоящее время существует большой запас пропускной способности. Проблемой теперь является сложность сетевого программного обеспечения.

Одна из возможных идей решения указанной проблемы — создание сетевого сопроцессора, на который можно возложить исполнение только сетевого программного обеспечения. Однако мощность такого сопроцессора будет примерно равна мощности основного процессора, что существенно удорожит систему. Кроме того, предположение, что центральный процессор будет чем-то загружен пока сетевой работает, — это миф. Если центральный и сетевой процессоры будут примерно равны по мощности, то их синхронизация и взаимодействие, в свою очередь, превратятся в самостоятельную проблему, решение которой потребует самостоятельных накладных расходов. (Здесь уместно вспомнить закон Амдаля.)

Важны также последствия использования гигабитных скоростей для проблемы обратной связи в сетевых протоколах. Например, ожидание подтверждений, даже групповых, как в протоколе скользящего окна, в этом случае очень невыгодно. Куда выгоднее заранее огово-

рить допустимые скорость и время, в течение которого эта скорость должна выдерживаться.

Также примером обратной связи является протокол медленного старта, в котором многократно ведется испытание пропускной способности сети. На гигабитных скоростях каждое испытание — это огромные накладные расходы. Значит, куда выгоднее заранее договориться о допустимой максимальной пропускной способности и заранее зарезервировать ее на все время сеанса.

Теперь несколько слов об организации структуры пакета для гигабитной сети. Полей должно быть как можно меньше, чтобы сократить время обработки заголовка. Причем поля должны быть достаточно длинными, т. е. они должны обеспечивать решение проблемы адресации, контроля времени жизни пакета в сети и т. п. Поля также должны быть выровнены по слову, чтобы облегчить их обработку в процессоре.

Поле данных должно быть переменной длины и достаточно большим, чтобы поддерживать эффективность операций и загрузку канала. Чем длиннее поле данных, тем короче заголовок при прочих равных условиях. Об этом не стоит забывать.

Заголовок и поле данных должны иметь раздельные контрольные суммы. Во-первых, для того чтобы можно было проверить корректность заголовка, не трогая данных, во-вторых, чтобы можно было проверить заголовок до того, как начнется копирование данных в приложение, а в-третьих, для того чтобы копирование данных можно было совместить с проверкой контрольной суммы для данных. Если же контрольные суммы совмещены, то совмещать копирование и проверку было бы нерационально, так как если начать проверку вместе с копированием, то при ошибке в заголовке копирования не потребуется.

Глава 4
УРОВЕНЬ ПРИЛОЖЕНИЙ

4.1. Сетевая безопасность

4.1.1. Общие сведения

Теперь рассмотрим прикладной уровень нашей модели, который является прикладным уровнем в сети Интернет.

Пока сети применялись лишь в университетах для научных исследований и в крупных организациях для совместного использования устройств, например принтеров, вопросы о безопасности информации в них просто не возникали. Теперь, когда сети вошли в нашу повседневную жизнь и ими пользуются рядовые граждане для управления банковским счетом, оплаты покупок, заказов товаров, работы, обучения и общения между собой, при этом дома они также окружены сетями персональных бытовых приборов, имеющими выход в Интернет, а организации хранят в сети критическую для своей деятельности информацию, обеспечение безопасности информации в сети ЭВМ становится серьезной проблемой.

Безопасность информации — это состояние информации, обрабатываемой средствами вычислительной техники или автоматизированной системы в сети ЭВМ, характеризующееся ее защищенностью от *внутренних* и *внешних угроз*, т.е. от нарушения конфиденциальности, целостности, доступности, а также от незаконного тиражирования, которые наносят материальный и моральный ущерб владельцу или пользователю этой информации.

Угроза безопасности информации — это потенциально возможное преднамеренное или непреднамеренное происшествие, которое может оказать нежелательное воздействие на саму систему в сети ЭВМ, а также привести к потере безопасности информации, хранящейся в ней.

Уязвимость сети ЭВМ — это некая характеристика сети, которая делает возможным возникновение угрозы безопасности информации.

Атака на сеть ЭВМ — это действие, предпринимаемое злоумышленником, заключающееся в поиске и использовании уязвимости сети.

В сфере обеспечения безопасности информации можно выделить следующие три группы проблем.

1. Секретность:

• конфиденциальность — только санкционированный доступ к информации (никто не может прочесть ваши письма без вашего ведома);

• целостность — только санкционированное изменение информации (никто без вашего разрешения не может изменить данные о вашем банковском счете).

2. Идентификация подлинности пользователей и документов:

• имея с кем-то дело через сеть, вы должны быть уверены, что это тот, за кого он себя выдает (если вы получили сообщение от налоговой инспекции уплатить определенную сумму денег, вы должны быть уверены, что это не шутка);

• получив через сеть электронную версию документа, необходимо определить, что он подлинный, а не фальсифицирован.

3. Надежность управления или доступность ресурсов и сетевых услуг:

• несанкционированное использование ресурсов (если вы получите счет за телефонные переговоры, которые вы не вели, вам это вряд ли понравится);

• обеспечение доступности ресурсов для авторизованных пользователей.

Прежде чем обсуждать проблемы обеспечения безопасности следует рассмотреть основные виды работы с информацией в сети: передачу, обработку, хранение и представление.

При передаче основную угрозу безопасности информации представляет несанкционированный доступ к каналу ее передачи, который возможен как в виде физического доступа к линии передачи, каналообразующей аппаратуре, коммутаторам СПД, так и в виде радиоперехвата и подслушивания вторичного электромагнитного излучения от физической линии, по которой передают информацию. Угрозу также представляет блокирование канала передачи информации.

При хранении основную угрозу безопасности информации представляет несанкционированный доступ к ее хранилищу: основной памяти компьютера и внешней памяти. Такой доступ, например к файлу, может осуществляться на физическом и логическом уровнях сети в целях обеспечения утечки, изменения или уничтожения информации.

При обработке основная угроза безопасности информации исходит от несанкционированного изменения алгоритма ее обработки, в результате чего происходят несанкционированные передача, изменение либо уничтожение информации, а также блокирование доступа к процессору, например при загрузке его «тяжелым» процессом. Такое изменение алгоритма обработки информации может происходить либо в результате замены штатной программы ее вредоносным аналогом, либо в результате внедрения в код существующей программы чужого вредоносного кода вследствие наличия так называемых уязвимостей в базовом программном обеспечении компьютера.

При представлении информации, т.е. при вводе ее в сеть и выводе из сети, также могут происходить несанкционированные доступ

и изменение информации. Однако методы, которые используются для этого, далеки от темы данного учебника, поэтому мы их здесь рассматривать не будем [10].

Продолжая рассмотрение проблем обеспечения безопасности информации, заметим, что и вне электронного пространства людям приходится иметь дело с этими проблемами: распознавание подделки документов, ограничение доступа к информации (уровни секретности), опознавание людей с помощью биометрических методов и т. д. Поэтому очень многие методы защиты информации в сетях ЭВМ взяты из жизни людей.

Прежде всего подумаем, в каком месте стека протоколов должна располагаться защита информации в сети. Одного такого места не существует: защита возможна на каждом уровне сети. Например, на физическом уровне для контроля доступа к каналу можно поместить кабель в опечатанную трубу, заполненную газом под давлением. Тогда любая попытка просверлить эту трубу приведет (хотя бы кратковременно) к падению давления газа, срабатыванию датчика давления, а следовательно к включению сигнала тревоги, а если пустить в трубу ядовитый газ, то на сигнал тревоги можно не спешить.

На канальном уровне данные могут быть зашифрованы на одной машине, а расшифрованы на другой, и об этом шифре верхние уровни могут ничего не знать. Однако поскольку пакет дешифруется на каждом маршрутизаторе (хотя бы частично), то в памяти маршрутизатора этот пакет может стать предметом атаки, т. е. предметом для осуществления попытки несанкционированного доступа. Тем не менее при передаче данных этот метод, называемый шифрованием канала, часто применяется в сетях.

На сетевом уровне распространенным решением является наличие сетевого экрана, или брандмауера (firewall) (см. подразд. 4.1.7).

На транспортном уровне проблему секретности данных при передаче решают шифрованием всех сегментов транспортного соединения и применением так называемых сеансовых шлюзов. Однако несмотря на все существующие меры защиты в сети до сих пор нет удовлетворительного решения проблемы безопасности.

4.1.2. Обычное шифрование

История шифрования богата и разнообразна [85]. Традиционно ее развивали четыре группы людей: военные, дипломаты, любители вести дневники и любовники. Последние, например, активно использовали газеты для переписки. Дело в том, что в XIX—XX веках в Великобритании можно было опубликовать бесплатно объявление в газете в специальном разделе, в то время как стоимость почтовых услуг была достаточно высока (вспомните рассказы Конан-Дойля о Шерлоке Холмсе). Основоположник принципов организации вычислителей Ч. Беббидж, например, развлекался тем, что раскрывал

шифры объявлений незадачливых любовников и публиковал объявление, зашифрованное их шифром.

Стандартная схема шифрования следующая (рис. 4.1). Исходный текст, называемый также открытым текстом, обрабатывают по определенному алгоритму со специальным параметром, называемым ключом. В результате этой обработки получают так называемый шифр-текст, или криптограмму. Злоумышленник может аккуратно копировать все шифр-тексты. Однако в отличие от получателя у него нет ключа, и быстро прочесть сообщение он не может, для этого ему необходимо вскрыть шифр, т.е. узнать алгоритм и ключ, использованные при шифровании этого сообщения. Иногда злоумышленник может не только скопировать сообщение, но позже отправлять свои сообщения, имитируя настоящего отправителя. Такого злоумышленника называют активным. Искусство создания шифра называют криптографией, а искусство его вскрытия — криптоанализом. Вместе эти дисциплины образуют криптологию.

Смена алгоритма шифрования, его создание, тестирование, внедрение всегда сопровождаются огромными затратами. Эти организационные моменты всегда являются точками уязвимости любого шифра. Перед людьми, отвечающими за применение и использование шифра, всегда стоят вопросы, как часто надо менять шифр или ключ и как определить, что шифр уже вскрыт?

Один из способов повышения надежности шифра — шифрование на основе ключа. Ключ — относительно короткая строка текста, которая используется при шифровании и расшифровке сообщений. При этом сам алгоритм шифрования может быть хорошо известен, а менять требуется только ключи.

Естественно, при создании нового алгоритма шифрования возникает вопрос, как проверить его устойчивость к взлому? Для этого сам алгоритм публикуют. Публикуя алгоритм шифрования, его автор даром получает консультации многих исследователей в этой области.

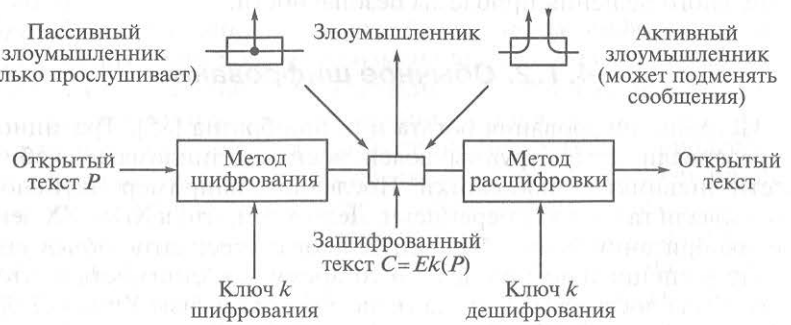


Рис. 4.1. Стандартная схема шифрования

Если ни один из них в течение пяти лет не объявит, что он вскрыл алгоритм, то такой алгоритм можно считать вполне надежным.

Основа секретности шифра — ключ. Длина ключа — один из основных вопросов разработки. Рассмотрим для примера комбинационный цифровой замок. Все знают, что его ключ — это последовательность цифр. При этом для вскрытия замка из двух цифр необходимо перебрать 100 комбинаций, а из трех — 1 000 и т.д. Длина ключа определяет объем работы, который придется выполнить криптоаналитику, чтобы вскрыть шифр. Этот объем растет экспоненциально в зависимости от длины ключа. Секретность достигается не только за счет сложности алгоритма, но и за счет длины ключа. При этом чтобы защитить от прочтения вашу почту 64-разрядного ключа вполне достаточно, а для засекречивания государственных документов требуется ключ в 128 или даже 256 разрядов.

Шифрование замещением

Все приемы шифрования исторически подразделяются на шифрование замещением и шифрование перестановкой.

Шифрование замещением состоит в том, что буква или группа букв замещается другой буквой или группой букв из того же самого либо из другого алфавита. Если используется один и тот же алфавит, замещение называется моноалфавитным, а если алфавиты разные, — полиалфавитным. Например, шифр Юлия Цезаря заключался в замене каждой буквы в слове третьей буквой, следующей за ней в алфавите:

а б в г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я

г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я а б в

пришел увидел победил тулыио целжзо тсдзжло

Это так называемое моноалфавитное замещение, где ключом является 33-буквенная строка, соответствующая алфавиту. Здесь возможны $33! = 4 \cdot 10^{33}$ ключа. Даже если на проверку одного ключа компьютер будет тратить 1 мкс, то на расшифровку уйдет около 10^{13} лет.

Этот прием можно применять, когда алфавиты исходного текста и шифрограммы разные. Например, алфавит исходного текста — кириллица, а алфавит шифрограммы — целые двухзначные числа. Алфавитов шифрограммы может быть несколько, и они могут изменяться по определенному правилу, зависящему от ключа. Примером такого шифра является Великий шифр Виженера [85].

Однако для того чтобы прочесть сообщение, необязательно тупо перебирать все возможные варианты ключей. Найти необходимый ключ быстрее можно, используя знание частотных характеристик

языка: частоты встречаемости отдельных букв, двухбуквенных буквосочетаний, трехбуквенных сочетаний и т.д. Для этого надо подсчитать частоту букв в шифр-тексте и попытаться сопоставить наиболее часто встречающимся буквы в шифре с буквами, наиболее часто встречающимися в языке. Затем найти устойчивые буквосочетания и т.д. Следовательно, здесь большое значение имеют дополнительные сведения о шифрограмме: на каком языке написано исходное сообщение, его длина, типичные приветствия в данном языке и т.д. Чем длиннее сообщение, тем представительнее будет выборка для его анализа по частоте встречаемости букв и буквосочетаний.

В [85] приведено много примеров того, как с помощью частотных характеристик английского языка можно вскрывать не только шифры, построенные на основе моноалфавитного замещения, но и куда более сложные. Там же описано, как Ч. Беббидж вскрыл великий шифр Виженера, впервые применив знание частотных характеристик английского языка при криптоанализе, а также реальная история «железной маски» (вспомните роман А. Дюма об отпрыске королевской крови, скрытом под железной маской), которая, как выяснилось из секретной переписки Людовика XIV со своим военным министром спустя почти 300 лет, оказалась намного прозаичней.

Шифрование перестановкой

Шифрование перестановкой состоит в изменении порядка набора букв без изменения самих букв. Для примера рассмотрим метод шифрования по столбцам. Выбираем ключ — последовательность неповторяющихся символов, которые нумеруем в соответствии с их местом в алфавите. Номер один получает буква, расположенная ближе всего к началу алфавита, номер два — буква, следующая в алфавите за ней, и т.д. Чем ближе к началу алфавита символ, тем меньше его номер. Шифруемый текст размещается по строкам. Длина строки — длина ключа. В результате получаем массив, где столбцы нумеруются в соответствии с номером символа в ключе. Каждому столбцу соответствует символ ключа, который имеет определенный номер. Упорядочим столбцы по возрастанию этих номеров: сначала выпишем все символы первого столбца, затем символы второго столбца и т.д. (рис. 4.2). Этот метод можно усовершенствовать многими способами.

Ш	У	Т	О	Ч	К	А
7	5	4	3	6	2	1
В	Ы	С	Ы	Л	А	Й
Т	Е	Д	Е	Н	Ь	Г
И	Б	О	Ч	К	А	М
И	_	_	_	_	_	_

Открытый текст:
высылайтеденьгибочками

Зашифрованный текст:
йгм_аьа_лнк_ыеч_сдо_ыеб_втии

Рис. 4.2. Шифрование перестановкой

Для раскрытия этого шифра криптоаналитик прежде всего должен убедиться, что имеет дело с шифрованием перестановкой. Для этого он должен подсчитать частоту встречаемости букв в шифре, и если она соответствует частотным характеристикам языка, то это означает, что это именно метод перестановки. Намек на порядок столбцов могут дать устойчивые буквосочетания, имеющиеся в языке.

4.1.3. Алгоритмы с секретными ключами

Особенность современной криптологии состоит в следующем: если раньше алгоритм был простой, а вся сложность шифрования заключалась в ключе, то теперь, наоборот, стараются алгоритм делать как можно изощреннее, чтобы криптоаналитик, получив как угодно много зашифрованного текста, не смог из него ничего извлечь.

Методы перестановки и замещения реализуются с помощью простых электронных схем, показанных на рис. 4.3, *а*, *б*. В случае объединения *P*-схем и *S*-схем в сложные каскады (рис. 4.3, *в*) выход становится очень сложной функцией входа. На рис. 4.3, *а* *P*-схема выполняет перестановку над словом из восьми разрядов. На рис. 4.3, *б* схема *S* действует несколько сложнее, выполняя операцию замещения, она кодирует трехразрядное слово одной из восьми линий на выходе, устанавливая ее в 1. Затем *P*-схема переставляет эти восемь разрядов, после чего *S*-схема выполняет замещение 8 на 3.

Ключ по-прежнему определяет стойкость шифра. Ясно, что чем длиннее ключ, тем больше возможных вариантов и тем больше времени потребуется на перебор этих вариантов. При этом надо помнить, что сами алгоритмы шифрования являются вычислительно сложными процедурами.

Все алгоритмы шифрования с позиции использования ключа подразделяются на алгоритмы с секретным ключом и алгоритмы с открытым ключом. Алгоритмы с секретным ключом используют один ключ и для шифрования, и для дешифрования, поэтому их часто называют симметричными алгоритмами шифрования. Этот ключ является строжайшим секретом, известным только тому, кто шифрует сообщение, и тому, кто это сообщение расшифровывает. Слабым местом этих шифров является этап установки общего секретного ключа.

Одним из широко известных шифров с секретным ключом является шифр DES (Data Encryption Standard). Этот шифр, созданный на базе разработки фирмы IBM, был принят как стандарт в области шифрования в январе 1977 г. правительством США. Разница между этими разработками состояла в том, что DES предполагал 56-разрядный ключ, а у IBM он был 128-разрядный.

Алгоритм DES состоит из 19 этапов. На первом этапе исходный текст разбивается на блоки по 64 бит каждый, и над каждым блоком выполняется перестановка. Последний этап является инверсией

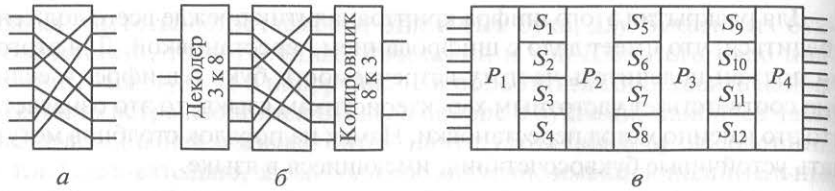


Рис. 4.3. Основные составляющие шифрования:
а — *P*-схема; *б* — *S*-схема; *в* — сложный каскад

первой перестановки. Предпоследний этап заключается в обмене местами 32 самых левых битов и 32 самых правых битов. Для этапов со 2-го по 17-й с помощью специального преобразования исходного 56-разрядного ключа строятся 16 частных ключей, которые используются для преобразования данных.

Подробно алгоритм DES описан в [19].

У алгоритма DES имеется два недостатка. Во-первых, он представляет собой моноалфавитное замещение с 64-разрядным символом, а всегда при подаче одних и тех же 64 разрядов исходного текста на вход, те же самые 64 разряда получают на выходе. DES сохраняет структуру сообщения, т. е. одни и те же поля исходного текста попадут в одни и те же места шифр-текста, чем может воспользоваться злоумышленник. Зная структуру исходного сообщения и длину его полей, он просто переставит необходимые поля в шифр-тексте, чтобы несанкционировано изменить сообщение.

Во-вторых, для начала шифрования надо иметь сразу весь 64-разрядный блок исходного текста, а это не совсем удобно, если речь идет об интерактивных приложениях. Кроме того, эти 64 разряда надо накапливать в открытом виде, что делает схему шифрования уязвимой. (Как можно устранить эти недостатки см. в [19].)

В 1977 г. Диффи и Хеллман из Стэнфорда опубликовали проект машины стоимостью в 20 млн долларов, которая вскрывала DES [39]. Достаточно было подать на вход этой машины небольшой фрагмент зашифрованного текста и соответствующий фрагмент исходного текста, и она в течение дня находила ключ шифра.

Существует много способов атаковать шифр. Все они основаны на распараллеливании перебора множества возможных ключей. Например, 140 000 человек, вооруженных компьютерами, способными выполнять 250 000 шифрований в секунду, смогут за месяц перебрать пространство из $7 \cdot 10^{16}$ ключей.

Следовательно, DES не является надежным шифром и его нельзя использовать для ответственных документов. Однако удвоение длины ключа до 112 бит кардинально меняет ситуацию. В этом случае даже если использовать миллиард аппаратных дешифраторов, выполняющих по миллиарду операций в секунду, потребуется 100 млн лет,

чтобы перебрать пространство из $10^{16} \cdot 10^{16}$ 112-разрядных ключей. Такие рассуждения наталкивают на идею увеличения длины ключа за счет двухкратного применения DES с разными ключами K_1 и K_2.

Однако в 1981 г. Хеллман и Меркл обнаружили, что простое использование двух ключей не дает надежной схемы защиты [51, 62]. Дело в том, что применение дешифрации к зашифрованному тексту дает шифр, который получается после применения первого ключа к исходному тексту. Эту мысль поясняют следующие формулы:

$$C = EK_2(EK_1(P)); \; DK_2(C) = EK_1(P),$$

где P и C — соответственно открытый текст и шифрограмма; EK_i и DK_i — функции соответственно шифрования и дешифрования с ключом K_i при $i = 1,2$.

В этом случае можно предложить следующую процедуру взлома:

- вычислить все возможные применения функции E к шифруемому тексту;
- вычислить все возможные дешифрации зашифрованного текста однократным применением дешифрирующей функции D;
- отсортировать полученные таблицы и найти совпадающие строки. При этом полученная пара номеров строк — пара ключей;
- проверить эту пару на совпадение шифрования. Если получен неудачный результат, процедуру повторить с первого шага.

Тройное шифрование совершенно меняет дело. На рис. 4.4, *а* показана модификация схемы шифрования с двумя ключами в три этапа — EDE-схема, а на рис. 4.4, *б* — соответствующая схема дешифрования. Эту схему, положенную в основу международного стандарта, пока еще никому не удалось вскрыть. Здесь может возникнуть два вопроса. Первый: почему в этой схеме используются два, а не три ключа? Второй: почему используется схема EDE, а не EEE? Ответ на первый вопрос состоит в том, что двух ключей более чем достаточно для большинства применений. Использование схемы EDE вместо EEE связано с особенностями организации алгоритма DES.

Надо отметить, что существуют и другие алгоритмы шифрования [83].

Хорошо известен международный алгоритм шифрования данных IDEA (International Data Encryption Algorithm), разработанный специалистами из Швейцарии, в котором используется 128-разрядный

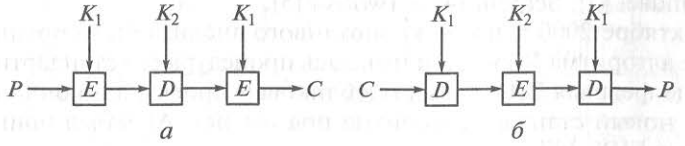


Рис. 4.4. Схема тройного шифрования с помощью алгоритма DES (*а*) и соответствующая схема дешифрования (*б*)

ключ [60]. Подобно DES этот алгоритм разбивает исходный текст на 64-разрядные блоки, над которыми производят определенные итерации, каждая из которых имеет параметры. На каждой итерации значение каждого выходного бита зависит от всех входных битов, поэтому здесь достаточно восьми итераций, а не 19, как в DES. Подробно этот алгоритм описан в [19].

В январе 1997 г. Национальный институт стандартов и технологий США — NIST (National Institute of Standards and Technologies) объявил о намерении выбрать преемника для алгоритма DES в открытом конкурсе криптоалгоритмов. Любая организация или лаборатория могла предложить свой алгоритм. К новому стандарту шифра предъявлялись следующие требования:

- длина блока шифра 128 бит;
- возможность поддерживать ключи длиной не менее 128, 192 и 256 бит.

Дополнительно NIST выдвинула следующие рекомендации, которые по сути и стали определяющими критериями при выборе нового стандарта [68]:

- алгоритм должен быть стойким к криптоаналитическим атакам, известным на время проведения конкурса;
- структура алгоритма должна быть ясной, простой и обоснованной. Такая структура, прежде всего, облегчила бы анализ алгоритма в рамках конкурса, а также дала бы некоторую гарантию отсутствия в нем специально внедренных «закладок», облегчающих авторам шифра расшифровку;
- отсутствие слабых (менее устойчивых к взлому перебором) и эквивалентных ключей (т.е. двух и более ключей, для которых результат шифрования одинаковый);
- скорость шифрования данных должна быть высокой на всех потенциальных аппаратных платформах — от 8-битовых до 64-битовых;
- структура алгоритма должна допускать распараллеливание операций;
- алгоритм должен предъявлять минимальные требования к оперативной и энергонезависимой памяти.

К 20 августа 1998 г. были поданы заявки, и на 1-й конференции AES (Advanced Encryption System) был объявлен список из 15 кандидатов. В марте 1999 г. прошла 2-я конференция AES, а в августе 1999 г. были объявлены пять алгоритмов финалистов: MARS [1], RC6 [2], Rijndael [3], Serpent [4] и Twofish [5].

В октябре 2000 г. после кропотливого анализа было объявлено о победе алгоритма Rijndael, и началась процедура его стандартизации. В конце февраля 2001 г. был опубликован проект, а в конце ноября 2001 г. новый стандарт алгоритма под именем AES был принят как стандарт FIPS 197.

Описание алгоритма AES. Шифруемые данные для алгоритма AES представляют в виде двухмерных байтовых массивов размером

4×4 байт. Все операции производятся над отдельными байтами массива, независимо над столбцами и строками.

На каждой итерации алгоритма выполняются следующие преобразования массива:

1. Операция SubBytes, представляющая собой замену каждого байта массива данных в соответствии со специальной таблицей кодирования.

2. Операция ShiftRows, представляющая собой циклический сдвиг влево всех строк массива данных за исключением нулевой. Сдвиг i-й строки массива (для $i = 1, 2, 3$) производится на i байт.

3. Операция MixColumns, выполняющая умножение каждого столбца массива данных, который рассматривается как полином степени 2^8, на фиксированный полином $a(x)$:

$$a(x) = 3x^3 + x^2 + x + 2.$$

Умножение выполняется по модулю $x^4 + 1$.

4. Операция AddRoundKey (рис. 4.5), преобразующая массив данных с расширенным ключом итерации. (Такое преобразование называется наложением ключа.) Процедура получения расширенного ключа для каждой итерации такова: над i-м столбцом массива данных ($i = 0 \ldots 3$) побитово выполняется логическая операция Исключающее ИЛИ (XOR) с расширенным ключом W_{4r+i}, где r — номер текущей итерации алгоритма, начиная с 1 (процедура расширения ключа будет описана далее). Количество итераций R алгоритма зависит от длины ключа (табл. 4.1). Перед первой итерацией алгоритма AES выполняется предварительное наложение расширенного ключа $W_0 \ldots W_3$ на открытый текст первых четырех слов итерации с помощью операции AddRoundKey. Последняя итерация отличается от предыдущих тем, что в ней нет операции MixColumns.

В соответствии с требованиями NIST в алгоритме AES используются ключи шифрования трех фиксированных размеров: 128, 192, и 256 бит. В зависимости от длины ключа конкретный вариант алгоритма AES обозначается как AES-128, AES-192 и AES-256.

Цель процедуры расширения ключа состоит в формировании необходимого числа слов расширенного ключа для их использования в операции AddRoundKey. Как уже говорилось, под «словом» здесь по-

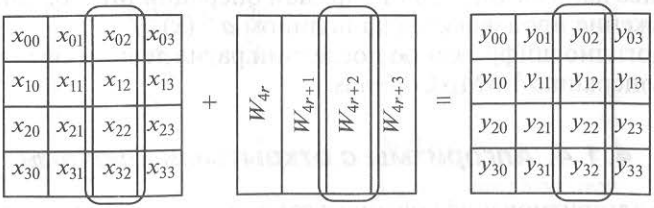


Рис. 4.5. Операция AddRoundKey алгоритма AES

Таблица 4.1

Зависимость числа итераций от длины ключа

Длина ключа, бит	R
128	10
192	12
256	14

нимается 4-байтовый фрагмент расширенного ключа, один байт которого используется в первичном наложении ключа и по одному байту используется в каждой итерации алгоритма. Таким образом, в процессе расширения ключа формируется $4(R + 1)$ слов.

Поскольку для дальнейшего изложения материала знание процедуры расширения ключа не требуется, то мы ее здесь приводить не будем. Отметим лишь, как несомненное ее достоинство, что расширение ключа может выполняться «на лету», т.е. параллельно с шифрованием данных.

Дешифрация. Дешифрация выполняется посредством применения обратных операций в обратной последовательности. Соответственно перед первой итерацией дешифрации выполняется операция AddRoundKey (которая является обратной самой себе), заключающаяся в наложении на шифр-текст четырех последних слов расширенного ключа, т.е. $W_{4R}... W_{4R+3}$.

Затем выполняется R итераций дешифрации, заключающихся в следующих преобразованиях:

1. Операция InvShiftRows выполняет циклический сдвиг вправо трех последних строк массива данных на то же число байтов, на которое выполнялся сдвиг операцией ShiftRows при шифровании.

2. Операция InvSubBytes выполняет побайтово обратную табличную замену в соответствии со специальной таблицей перекодирования.

3. Операция AddRoundKey, как и при шифровании, выполняет наложение на обрабатываемые данные четырех слов расширенного ключа $W_{4R}... W_{4R+3}$. Однако нумерация итераций R при расшифровке производится в обратную сторону: от $R - 1$ до 0.

4. Операция InvMixColumns выполняет умножение каждого столбца массива данных аналогично прямой операции MixColumns, однако умножение производится на полином $a^{-1}(x)$.

Аналогично шифрованию последний раунд дешифрования не содержит операцию InvMixColumns.

4.1.4. Алгоритмы с открытыми ключами

Идея алгоритмов шифрования с открытыми ключами, предложенная в 1976 г. Диффи и Хеллманом, состоит в следующем. Пусть име-

ются алгоритмы шифрования E и дешифрования D, которые удовлетворяют следующим требованиям:

- $D(E(P))$ *равно исходному тексту* P;
- чрезвычайно трудно получить D, зная E;
- E нельзя вскрыть через анализ исходных текстов.

Алгоритм шифрования E и его ключ, так называемый открытый ключ, публикуют или помещают таким образом, чтобы каждый мог их получить. Алгоритм D также публикуют, чтобы подвергнуть его изучению и проверке на стойкость, а вот ключ к нему хранят в секрете. Это так называемый секретный, или закрытый, ключ.

Взаимодействие двух абонентов A и B происходит следующим образом. Пусть A хочет послать B текст P. Абонент A шифрует текст $E_B(P)$, зная алгоритм и открытый ключ абонента B для шифрования. Абонент B, получив текст $E_B(P)$ и использовав секретный ключ и алгоритм D_B, вычисляет $D_B(E_B(P)) = P$. Никто не прочтет текст P кроме абонентов A и B, так как по условию алгоритм E_B нераскрываем, а алгоритм D_B нельзя вывести из E_B.

Примером такого алгоритма является алгоритм RSA, предложенный Ривестом, Шамиром и Адлеманом в 1978 г. Общая схема этого алгоритма следующая [78]:

1. Выберем два больших (больше 10^{100}) простых числа p и q.
2. Вычислим $n = p \times q$ и $z = (p - 1) \times (q - 1)$.
3. Выберем простое d, относительно простое по отношению к z.
4. Найдем e, удовлетворяющее условию $e \times d = 1 \bmod z$.

Разобьем исходный текст на блоки длиной p таким образом, чтобы каждый блок как число не превосходил n. Для этого выберем наибольшее k, при котором выполняется условие $p = 2^k < n$. Шифр сообщения p получим, вычислив шифрованный текст $C = p^e (\bmod n)$. Для расшифровки найдем $P = C^d (\bmod n)$.

Для шифрования требуются числа e, n, представляющие собой открытый ключ, а для дешифрования — числа d, n, т. е. закрытый ключ. Можно доказать, что для любого сообщения P в указанном диапазоне функции шифрования и дешифрования взаимообратные. Безопасность, обеспечиваемая применением этого метода шифрования, осно-

Открытый текст (P)		Шифрованный текст (C)			После расшифровки	
Символ	Число	p^3	$p^3 (\bmod 33)$	c^7	$c^7 (\bmod 33)$	Символ
S	19	6859	28	13492928512	19	S
U	21	9261	21	1601066541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	1	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	5	E
		Вычисления отправителя		Вычисления получателя		

Рис. 4.6. Пример использования RSA-алгоритма

вана на высокой вычислительной сложности операции разложения на простые множители больших чисел. Так, например, разложение на простые множители 200-разрядного числа потребует 4 млрд лет.

На рис. 4.6 приведен простой учебный пример применения RSA-алгоритма при $p = 3$, $q = 11$, $n = 33$, $z = 20$, $d = 7$, откуда получаем $e = 3$.

Один из основных недостатков алгоритма RSA — медленная работа.

4.1.5. Протоколы установления подлинности

Протоколы установления подлинности — аутентификации — позволяют процессу убедиться, что он взаимодействует с тем, с кем должен, а не с тем, кто лишь представляется таковым. Очень часто путают авторизацию, т. е. проверку прав на выполнение тех или иных операций, с аутентификацией, или идентификацией. Аутентификация отвечает на вопрос: «Вы взаимодействуете с тем, с кем надо, или Ваш vis-à-vis фальсифицирован, т. е. кто-то выдает себя за него». Если, например, к серверу обратился процесс с запросом удалить файл x.dat и объявил себя процессом «Вася», то сервер должен убедиться, что перед ним действительно Вася и что Вася имеет право делать то, что просит. Ключевым здесь, конечно, является первый вопрос, а ответ на второй вопрос — это дело авторизации, которая сводится к просмотру таблицы прав.

Общая схема всех протоколов аутентификации следующая: сторона A и сторона B начинают обмениваться сообщениями между собой или с центром раздачи ключей (ЦРК). При этом предполагается, что ЦРК — всегда надежный партнер, т. е. его нельзя фальсифицировать. Протокол аутентификации должен быть устроен таким образом, чтобы даже в случае если злоумышленник перехватит сообщения между A и B, то ни A, ни B не спутают друг друга со злоумышленником.

Аутентификация на основе закрытого разделяемого ключа

Основная идея первого протокола аутентификации, называемого протоколом ответа по вызову, состоит в том, что одна сторона посылает некоторое число (вызов), а другая сторона, получив это число, преобразует его по определенному алгоритму и отправляет обратно. Увидев результат преобразования и зная исходное число, инициатор может определить, правильно сделано преобразование или нет. Алгоритм преобразования является общим секретом взаимодействующих сторон.

Предположим, что стороны A и B имеют общий закрытый ключ K_{AB}, который взаимодействующие стороны как-то установили заранее, соблюдая конфиденциальность, нам неважно как (рис. 4.7).

На рис. 4.8 приведена схема с сокращенным числом передач между сторонами по сравнению со схемой, показанной на рис. 4.7.

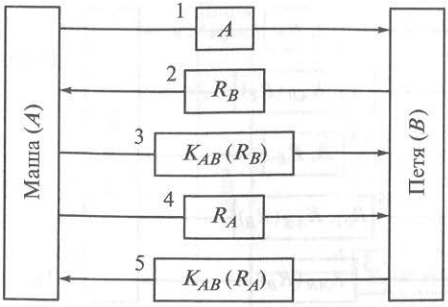


Рис. 4.7. Пример двухсторонней аутентификации с запросом и подтверждением:

A, *B* — идентификаторы взаимодействующих сторон (Маши и Пети соответственно); R_i — вызов, индекс которого указывает, кто его послал; K_i — ключ, индекс которого указывает его владельца (в данном случае закрытый ключ Маши и Пети); 1...5 — номера сообщений

На рис. 4.9 показана уязвимость схемы на рис. 4.8 и то, как злоумышленник может этой уязвимостью воспользоваться, выполнив так называемую атаку отражением. Идея этой атаки состоит в том, чтобы «заставить» Петю дать некоторое число R_B (второй шаг), после чего на третьем шаге подсунуть ему это же число как свой вызов. На этот вызов Петя согласно протоколу ответит преобразованным ключом $K_{AB}(R_B)$. В результате злоумышленник получит и число R_B, и ключ $K_{AB}(R_B)$, что ему и надо, чтобы выдать себя за Машу.

Существует несколько общих правил построения протоколов аутентификации:

- инициатор передачи должен доказать, кто он есть, прежде чем вы пошлете ему какую-либо важную информацию;
- инициатор и отвечающий должны использовать разные ключи;

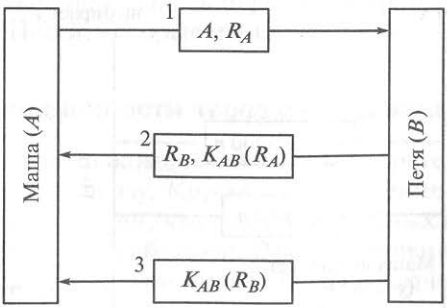


Рис. 4.8. Схема сокращенной по сравнению с рис. 4.7 двухсторонней аутентификации с запросом и подтверждением

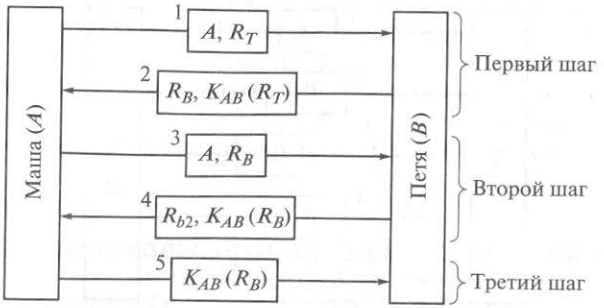


Рис. 4.9. Схема атаки отражением

• инициатор и отвечающий должны использовать начальные вызовы из разных непересекающихся множеств.

В схеме, приведенной на рис. 4.9, все эти три правила нарушены.

Установка общего закрытого ключа

До сих пор мы предполагали, что Маша и Петя имеют общий закрытый ключ. Рассмотрим теперь, как они могут его установить. Например, они могут воспользоваться телефоном, но при этом Пете необходимо убедиться, что ему звонит именно Маша, а не злоумышленник. Можно также договориться о личной встрече, куда принести паспорт и прочие документы, удостоверяющие личность. Однако существует протокол, который позволяет двум незнакомым людям установить общий закрытый ключ даже при условии, что за ними следит злоумышленник. Это протокол Диффи—Хеллмана, схема которого показана на рис. 4.10 [40].

Прежде всего, Маша и Петя должны договориться об использовании двух больших простых чисел n и g, удовлетворяющих опреде-

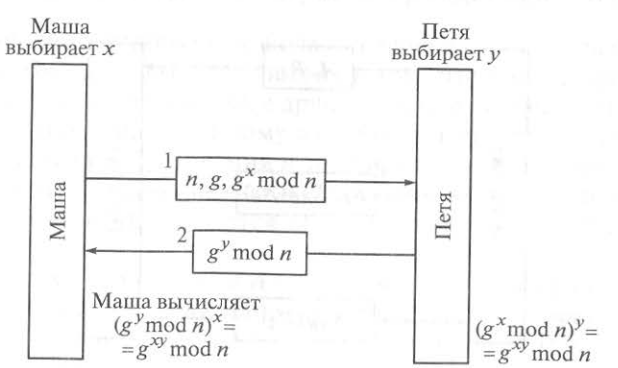


Рис. 4.10. Схема установки общего закрытого ключа Диффи—Хеллмана

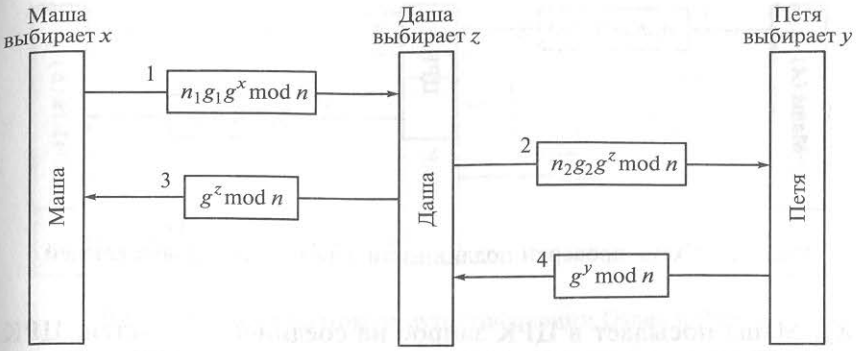


Рис. 4.11. Схема взлома алгоритма, называемая «чужой в цепочке»

ленным условиям. Причем эти числа могут быть общеизвестны. Затем Маша выбирает большое число, например x, и хранит его в секрете, а Петя выбирает число y и также держит его в секрете.

Маша посылает Пете сообщение вида (n, g, $g^x \bmod n$), а Петя — отвечает сообщением вида ($g^y \bmod n$). Теперь Маша выполняет операцию $(g^y \bmod n)^x$, а Петя — операцию $(g^x \bmod n)^y$. Удивительно, но теперь они имеют общий ключ ($g^{xy} \bmod n$). Например, если $n = 47$, $g = 3$, $x = 8$, $y = 10$, то Маша посылает Пете сообщение вида (47, 3, 28), поскольку $3^8 \bmod 47 = 28$, а Петя — ответ вида (17). Маша вычисляет: $17^8 \bmod 47 = 4$, а Петя вычисляет $28^{10} \bmod 47 = 4$. Ключ установлен — 4.

Злоумышленник следит за всем этим, и единственное, что мешает ему вычислить x и y, это то, что неизвестен алгоритм с приемлемой сложностью вычисления логарифма от модуля для простых чисел. Однако и у этого алгоритма имеется слабое место. Прием, используемый в этом случае, называется «чужой в цепочке» (рис. 4.11).

В данном случае чужой (Даша) инициирует установку закрытых общих ключей по описанному ранее протоколу с Машей и Петей. Теперь Даша может пересылать через себя транзитом сообщения между Машей и Петей, которые и знать-то об этом не будут.

Проверка подлинности через центр раздачи ключей

Договориться с незнакомцем об общем секрете можно, но вряд ли это следует делать сразу. Кроме того, общение с n людьми потребует хранения n ключей, что для общительных или популярных личностей может быть проблемой. Другим решением данной проблемы является введение надежного центра раздачи ключей — ЦРК (рис. 4.12).

Идея протокола проверки подлинности через ЦРК состоит в следующем. Маша выбирает ключ сессии K_S. Используя свой ключ

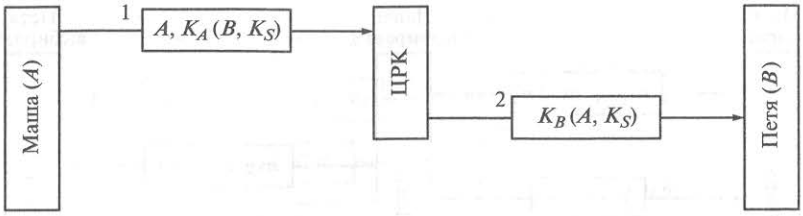


Рис. 4.12. Схема проверки подлинности через центр раздачи ключей

K_A, Маша посылает в ЦРК запрос на соединение с Петей. ЦРК знает Петю и его ключ K_B. С помощью этого ключа ЦРК сообщает Пете ключ сессии K_S и информацию о том, кто хочет с ним соединиться.

Однако это решение уязвимо. Пусть злоумышленник как-то убедил Машу связаться с Петей и скопировал весь обмен сообщениями. Позже он может воспроизвести этот обмен за Машу и заставить Петю действовать так, как если бы с ним говорила Маша. Этот способ называется *атака подменой*.

Примером атаки подменой может служить следующий сценарий. Злоумышленник устраивается работником к состоятельному клиенту, использующему интернет-банк. По истечении нескольких недель работы, когда задолжность оплаты за его работу станет существенной, он просит этого состоятельно клиента оплатить свою работу, переведя деньги на определенный счет. Предварительно этот мнимый работник устанавливает и подключает к компьютеру клиента записывающую аппаратуру. Ничего не подозревающий клиент инициирует операцию оплаты со своего компьютера. После этого работник исчезает и с другого компьютера воспроизводит транзакцию с банком, записанную у клиента.

Против такой атаки подменой имеется несколько решений.

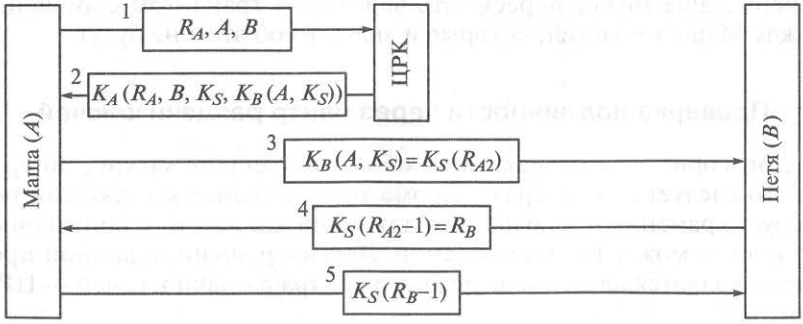


Рис. 4.13. Схема протокола аутентификации Нидхема—Шредера

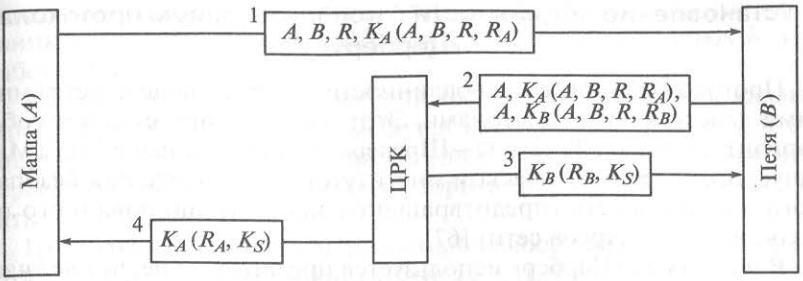


Рис. 4.14. Схема протокола аутентификации Отвея и Рииса

Во-первых, использование временны́х меток. Это решение требует синхронизации часов в сети. Поскольку в сети всегда есть расхождение в показаниях часов отдельных машин, то надо задать определенный интервал, в течение которого допустимо считать сообщение верным. Однако злоумышленник может реализовать атаку подменой в течение этого интервала времени.

Во-вторых, использование разовых меток. Однако при этом каждая из сторон должна помнить все разовые метки, использованные ранее. Это обременительно. Кроме того, если список использованных разовых меток будет утерян по каким-либо причинам, то этот метод перестанет работать. Можно комбинировать использование разовых и временны́х меток.

В-третьих, смена ключей для каждой новой транзакции, для чего необходимо иметь заранее согласованный список одноразовых ключей. В этом случае ключ повторно использован быть не может. На сегодня это наиболее часто используемое решение.

Надежное решение проблемы установления подлинности обеспечивает многосторонний протокол «Вызов-ответ», хорошо известным примером которого является протокол Нидхема—Шредера [65]. Один из вариантов этого протокола показан на рис. 4.13.

Сначала Маша сообщает ЦРК, что она хочет взаимодействовать с Петей. ЦРК сообщает ключ сессии и разовую метку R_A, шифруя сообщение ключом Маши. Разовая метка защищает Машу от подмены. Теперь, имея ключ сессии, Маша начинает обмен сообщениями с Петей. Разовые метки R_{A2} и R_B защищают Машу и Петю от подмен.

Хотя этот протокол в целом надежен, но и при его использовании имеется небольшая опасность. Если злоумышленник раздобудет все-таки старый ключ сессии, то он сможет подменить третье сообщение старым и убедить Петю, что это Маша. На рис. 4.14 приведена схема исправленного протокола, который предложили Отвей и Риис [69]. В этой модификации ЦРК следит, чтобы R было одним и тем же в обеих частях второго сообщения.

Установление подлинности с использованием протокола «Цербер»

Протокол установления подлинности «Цербер» используется многими действующими системами. Этот протокол представляет собой вариант протокола Нидхема—Шредера, разработанный в MIT (Массачусетском технологическом институте) для обеспечения безопасного доступа в сеть (предотвращения несанкционированного использования ресурсов сети) [67].

В протоколе «Цербер» используется предположение, что все часы в сети хорошо синхронизованы, а также предполагается использование кроме рабочей станции A еще трех серверов:

- сервера установления подлинности (СП), проверяющего пользователей на этапе входа в систему (login);
- сервера выдачи квитанций (СВБ), обеспечивающего идентификацию квитанции;
- сервера B, обеспечивающего выполнение работы, необходимой A.

Сервер подлинности аналогичен ЦРК и знает секретный пароль для каждого пользователя, а СВБ выдает квитанции, которые подтверждают подлинность заказчиков работ.

На рис. 4.15 показана работа протокола «Цербер». Сначала пользователь садится за рабочую станцию и посылает открыто свое имя СП, который отвечает ключом сессии и квитанцией вида $<K_S, K_{TGS}(A, K_S)>$. Все это зашифровано закрытым ключом A. Когда второе сообщение приходит на рабочую станцию, у A запрашивается пароль, чтобы по нему установить K_A для расшифровки второго сообщения. Причем пароль перезаписывается с временно́й меткой, чтобы предотвратить его захват злоумышленником. Выполнив операцию login, пользователь может сообщить станции, что ему требуется сервер B. Рабочая станция обращается к СВБ за квитанцией для использования сервера B. Клю-

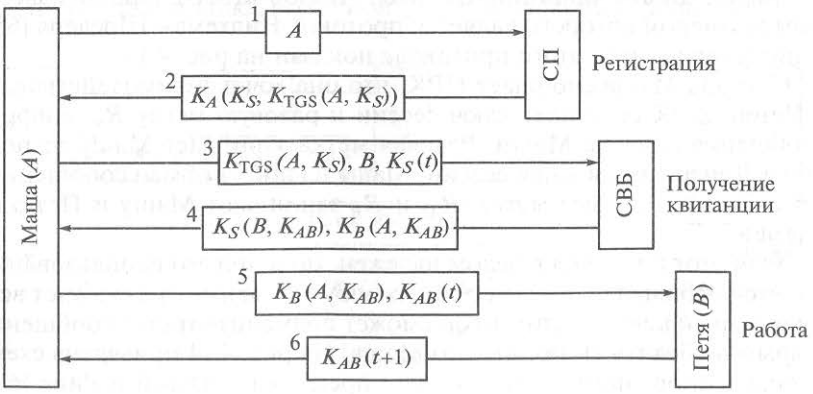


Рис. 4.15. Схема протокола «Цербер»

чевым элементом этого запроса является ключ $K_{TGS}(A, K_S)$, зашифрованный закрытым ключом СВБ. В ответ СВБ посылает ключ K_{AB} для работы A и B.

Теперь A может обращаться непосредственно к B с этим ключом. Это взаимодействие сопровождается временными метками, чтобы защититься от подмены. Если позднее A понадобится работать с сервером C, то A должна будет повторить третье сообщение, но указать в нем сервер C.

Поскольку сеть может быть очень большой, то нельзя требовать, чтобы все использовали один и тот же СП, т.е. сеть разбивают на области, в каждой из которых будут свои СП и СВБ, взаимодействующие между собой.

Установление подлинности с помощью шифрования с открытым ключом

Установить взаимную подлинность можно с помощью шифрования с открытым ключом. Пусть Маша и Петя уже знают открытые ключи друг друга и используют, чтобы установить подлинность друг друга, а затем применяют шифрование с секретным ключом, которое на несколько порядков быстрее.

На рис. 4.16 показана схема установления подлинности с помощью шифрования с открытыми ключами. Здесь R_A и R_B используются, чтобы Маша и Петя могли убедиться в подлинности друг друга. Единственным слабым местом этого протокола является предположение, что Маша и Петя уже знают открытые ключи друг друга, так как обмен этими ключами уязвим для атаки типа «чужой в цепочке».

Ривст и Шамир предложили протокол, защищенный от атаки типа«чужой в цепочке». Это так называемый протокол с внутренним замком, идея которого заключается в том, что после обмена открытыми ключами Маша и Петя передают свои сообщения в два этапа,

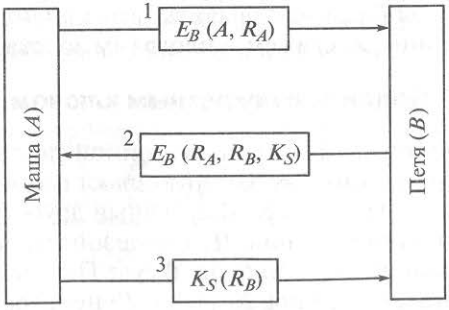


Рис. 4.16. Взаимная аутентификации с помощью шифрования с открытым ключом

например сначала только четные биты, а затем нечетные. В этом случае злоумышленник, имея только четные биты сообщения Пети, не может расшифровать всего сообщения, а значит, не может послать сообщение Маши, зашифрованное ее открытым ключом. Не может он и просто переслать сообщение Пети, так как у Маши и у Пети разные открытые ключи.

4.1.6. Электронная цифровая подпись

Подлинность многих юридических, финансовых и прочих документов устанавливается наличием подписи уполномоченного лица. Поскольку имеются способы, позволяющие отличить фотокопии от подлинника, то фотокопии мы рассматривать не будем. Подпись на документе — это факт, подтверждающий, что лицо, подписавшее документ, либо является автором документа, либо знакомо с документом.

Проблема создания электронного аналога ручной подписи весьма сложна. Требуется система, которая позволяла бы одной стороне посылать «подписанный» документ другой стороне таким образом, чтобы выполнялись следующие условия:

- получатель должен иметь возможность удостовериться в подлинности отправителя;
- отправитель не должен иметь возможность отречься от документа;
- получатель не должен иметь возможность подделать документ.

Первое условие важно, например, при взаимодействии с банком, когда необходимо убедиться, что тот, кто проводит операцию, действительно является владельцем счета.

Второе условие важно, например, в случае, когда клиент запросил продать ему тонну золота, а цена на него на бирже после этого запроса неожиданно упала, и у клиента может возникнуть соблазн отказаться от своей заявки.

Третье условие предотвращает ситуацию, в которой, например, цена на золото на бирже неожиданно подскочила, и тогда у банка может появиться соблазн изобразить, что клиент ранее запросил продать ему не тонну, а, скажем, килограмм золота.

Подпись с секретным ключом

Одно из решений проблемы электронной подписи* — наделить полномочиями третью сторону, которую знают все, которая знает всех и которой верят все. Назовем ее «Сердечный друг» (СД). На рис. 4.17 показана схема такого решения. Что произойдет, если Маша позже откажется от посланного сообщения. В суде Петя предъявит сообщение P и электронную подпись $K_{СД}(A, t, P)$ непререкаемому автори-

* Иногда ее называют электронной цифровой подписью. Здесь для краткости мы будем использовать термин «электронная подпись».

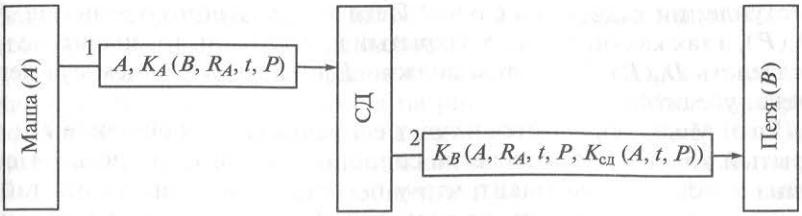


Рис. 4.17. Схема решения проблемы электронной подписи с использованием протокола «Сердечный друг»

тету СД, который расшифрует своим ключом эту запись, и все увидят A, t, P, т.е. что сообщение P было отправлено в момент времени t Машей.

Единственная слабость такого решения заключается в том, что злоумышленник может скопировать диалог между отправителем и получателем через СД и позже его повторить. Механизм временных меток позволяет ослабить эту проблему. Кроме того, сохранение последних R_A позволяет Пете заметить их повторное использование.

Подпись на основе открытого ключа

Недостаток рассмотренного решения состоит в том, что все должны доверять СД, который может читать сообщения. Кандидатами на эту роль могут быть специальные органы государственной власти, банк, нотариус. Однако далеко не все испытывают доверие к этим организациям.

На рис. 4.18 показана схема электронной подписи с использованием открытых ключей.

Предположим $E(D(P)) = P$ дополнительно к $D(E(P)) = P$ (этим свойством обладает алгоритм шифрования RSA, рассмотренный в подразд. 4.1.4.).

В этой схеме имеются два недостатка. Оба основаны на том, что схема работает до тех пор, пока Маша либо умышленно не рассекретит свой ключ, либо не изменит его в одностороннем порядке. При

Рис. 4.18. Схема решения проблемы электронной подписи на основе открытого ключа:

D_x — закрытый ключ; E_x — открытый ключ; x — Маша или Петя

наступлении судебного случая Петя предъявляет сообщение P и $D_A(P)$, а так как он не знает закрытый ключ Маши, то, значит, не мог подделать $D_A(P)$. При этом должно $E_A(D_A(P)) = P$, в чем суд легко может убедиться.

Если Маша обращается в суд, т. е. предъявляет сообщение P и открытый ключ $E_A(P)$, это легко сопоставить с тем, что есть у Пети. Однако если Маша заявит, что у нее украли ключи, а сама тайно передаст их либо сменит, не сообщив об этом Пете, то в последнем случае текущий открытый ключ E_A будет неприменим к тому закрытому ключу $D_A(P)$, который предъявит Петя. При этом надо сопоставить даты передачи сообщения и смены ключей.

Профиль сообщения

Рассмотренные решения проблемы электронной подписи критикуют за то, что они подменяют задачу установления подлинности задачей шифрования [19], когда зачастую необходимо только установление подлинности. Кроме того, шифрование — медленная операция, поэтому часто желательно просто только поставить подпись, удостоверяющую подлинность сообщения. Рассмотрим метод, который не требует шифрования всего сообщения.

Этот метод основан на использовании функции перемешивания (хэш-функции), которая по сообщению вычисляет битовую строку фиксированной длины. Эта функция, называемая профилем сообщения (Message Digest — MD), обладает тремя важными свойствами:

- у функции MD нет обратной функции;
- для заданного сообщения P вычислить функцию MD(P) просто;
- имея MD(P), невозможно восстановить P;
- никто не сможет подобрать два таких сообщения, MD от которых будут одинаковыми.

Этот метод можно применять как с закрытым ключом, так и с открытым. В случае использования закрытого ключа, как в схеме на рис. 4.17, надо элемент $K_{СД}$ в сообщении от СД к Пете заменить на $K_{СД}(A, t, \text{MD}(P))$. При этом СД подписывает не все сообщение, а лишь его профиль. В случае возникновения спора Петя легко докажет в суде, что он получил именно сообщение P, так как нет другого со-

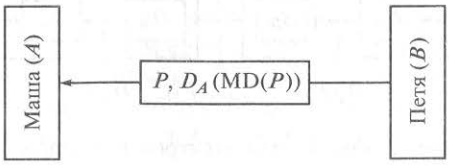


Рис. 4.19. Электронная подпись с использованием профиля сообщения

общения, которое даст такое же MD(P). Маша также легко докажет свою правоту, так как MD вычисляет СД.

Этот метод также работает в случае использования открытого ключа, схема которого показана на рис. 4.19. Здесь Маша шифрует своим ключом не все сообщение, а лишь его профиль. При этом Петя защищен от злоумышленника, так как если последний подменит сообщение, то Петя, получив сообщение и вычислив его MD, сравнит его с MD Маши и обнаружит подмену.

Были предложены несколько алгоритмов для MD, наиболее распространенные из которых MD5[77] и SHA [84].

4.1.7. Межсетевые экраны

Рассмотренные алгоритмы шифрования и проверки подлинности призваны решить проблему обеспечения конфиденциальности и целостности информации криптографическими методами. Однако надо помнить что шифрование — дорогое удовольствие, сопряженное со значительными вычислительными затратами, и его следует применять в тех случаях, когда информация транспортируется или хранится в среде, где велик риск нарушения ее конфиденциальности и целостности. Если же этот риск невелик, от шифрования следует отказаться.

Например, в сети организации — интранете. Способность любого компьютера соединяться, где бы он ни был, с любым другим компьютером — благо для пользователя, но сущее наказание для службы безопасности любой организации. Кроме угрозы потери информации здесь существует угроза притока в сеть вредоносного программного обеспечения: вирусов, червей и пр. Справедливости ради надо заметить, что согласно результатам проведенных исследований примерно 50 % опасности для сети таится вне ее, а 50 % — внутри, т.е. среди сотрудников.

При построении интранета используется хорошо известная идея построения крепости: все, что находится внутри крепостных стен, дружественно, т.е. это среда, которой можно доверять до определенной степени, а все, что вне крепостных стен, — враждебно. Число входов в крепость строго ограничено, и каждый вход охраняется: каждого входящего проверяют, имеет ли он право на проход в крепость, и все что ввозится в крепость также проверяется. Также процедуре проверки подвергают всех выходящих и все вывозимое.

Число точек входа-выхода в интранете строго ограничено и все информационные потоки, проходящие через эти точки, строго проверяют. Точки входа-выхода образуют так называемый периметр сети организации. Только эти точки являются выходами корпоративной сети в Интернет, и только через них возможен доступ извне к размещенным в корпоративной сети сервисам (веб-сайту компании, публичному почтовому серверу и т.п.).

Одним из видов средств для контроля входа и выхода являются межсетевые экраны (МСЭ). Наиболее часто их применяют для построения периметра интранета. Рассмотрим основные типы межсетевых экранов.

Итак, нужен механизм, который отличал бы «чистые» биты от «нечистых». Одним из таких механизмов является шифрование данных. Однако шифрование бессильно против вирусов, хакеров и прочих подобных угроз. Одним из средств борьбы с такими угрозами служат сетевые экраны сетевого уровня, или чистилище (firewall). Иногда их еще называют брандмауэры.

Чистилище — это средство, позволяющее ограничить число точек входа в сеть на сетевом уровне и обеспечить в этих точках жесткий контроль входящих и исходящих пакетов. Компания может иметь сколь угодно сложную сеть (интранет), объединяющую множество локальных сетей, однако весь трафик в интранет и из интранета направляется только через один шлюз, где происходит проверка пакета на соответствие определенным требованиям. Если пакет не удовлетворяет этим требованиям, то он не допускается в сеть или не выпускается из нее.

На рис. 4.20 показана организация чистилища [20], состоящего из двух маршрутизаторов, фильтрующих пакетов и шлюза приложений. Фильтры содержат таблицы, где перечислены маршрутизаторы и абонентские машины, от которых можно принимать и которым можно передавать пакеты. Шлюз приложений ориентирован на конкретные приложения, например шлюз для электронной почты, который анализирует поле данных и принимает решение, сбросить пакет или нет. Набор таких шлюзов полностью зависит от политики информационной безопасности конкретной организации. Чаще всего это шлюзы серверов электронной почты и всемирной паутины (WWW).

Межсетевые экраны могут также использоваться для разграничения доступа к внутренним сетевым ресурсам компании между различными ее подразделениями. Функциями межсетевого экрана являются сокрытие внутренней структуры сети, состава ресурсов сети и блокирование доступа к отдельным ресурсам.

МСЭ может быть реализован в виде программно-аппаратного комплекса на базе специализированной операционной системы, а может быть программной частью ОС общего назначения и ее ядра.

Все межсетевые экраны подразделяются на три типа:

- пакетные фильтры;
- шлюзы уровня соединения;
- шлюзы уровня приложений.

Все три типа МСЭ могут одновременно находиться на одном устройстве. Принцип их функционирования прост: МСЭ проверяет заголовок PDU соответствующего уровня на соответствие его параметров заранее введенным в МСЭ правилам. Если заголовок PDU не

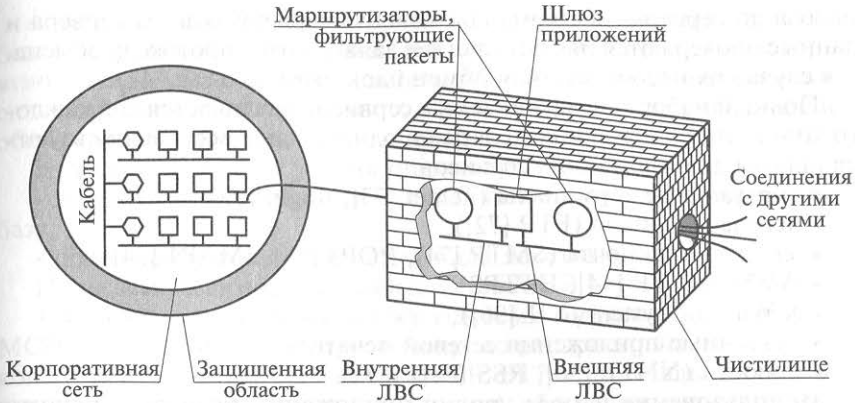


Рис. 4.20. Организация чистилища

соответствует хотя бы одному правилу, то МСЭ такой PDU не пропускает.

Пакетные фильтры, являющиеся наиболее простым типом межсетевых экранов, работают на сетевом уровне. Описание правил прохождения пакетов для МСЭ выполняется в виде табл. 4.2. Поля «Протокол», «Порт источника», «Порт назначения» относятся к правилам МСЭ транспортного уровня. Поле «Действие» может принимать значения пропустить или отбросить PDU, а флаги здесь те же, что и в заголовке IP-пакета.

Шлюз уровня соединения выполняет анализ заголовков сегментов в рамках каждого транспортного соединения (например, TCP-соединения). Если обычный пакетный фильтр анализирует и принимает решение для каждого пакета независимо, то шлюз уровня соединения принимает решение для соединения целиком. Прежде всего, такой анализ снижает вычислительную нагрузку на МСЭ, так как для установленного соединения нет необходимости проверять каждый пакет на соответствие всем правилам экрана, т.е. эти проверки можно выполнить один раз при установлении соединения.

Шлюзы уровня приложений, устанавливаемые между клиентами и серверами конкретных приложений, анализируют сессии соответствующих протоколов — SMNP, FTP, HTTP и т.д. (см. подразд. 4.2.3...4.2.5). Соединение между клиентом и сервером де-факто разбито на два независимых соединения: от клиента до шлюза и от

Таблица 4.2

Описание правила для межсетевого экрана

Действие	Протокол	Адрес источника	Порт источника	Адрес назначения	Порт назначения	Флаги

шлюза до сервера. Все команды клиента серверу, ответы сервера и данные проверяются на соответствие заявленному протоколу обмена, и в случае их несоответствия обмен блокируется.

Полный набор поддерживаемых сервисов различается для каждого конкретного межсетевого экрана, однако чаще всего используются шлюзы для следующих сервисов:

- виртуальные терминалы (Telnet [73], rlogin [55]);
- передача файлов (FTP [72]);
- электронная почта (SMTP [74], POP3 [63], IMAP4 [34]);
- WWW (HTTP [44], HTTPS);
- X Window System (X11 [96]);
- различные приложения сетевой печати;
- новости (NNTP [56], RSS [82]) и т.д.

Использование шлюза уровня приложений позволяет решить следующую важную задачу: сделать невидимой извне структуру корпоративной сети. Другой полезной функцией шлюза уровня приложений является обеспечение возможности аутентификации пользователей корпоративной сети централизованно на шлюзе и разграничения их полномочий доступа к сетевым ресурсам предприятия в соответствии с его политикой безопасности.

При описании правил доступа используются следующие параметры: название сервиса, имя пользователя, допустимый временной диапазон использования сервиса, IP-адреса компьютеров, с которых можно пользоваться сервисом, и схемы аутентификации. Шлюзы прикладного уровня позволяют обеспечить наиболее высокий уровень защиты, поскольку взаимодействие с внешним миром здесь реализуется через небольшое число прикладных программ, полностью контролирующих весь входящий и выходящий трафик.

Достоинствами пакетных фильтров являются:

- невысокая стоимость;
- высокая скорость обработки трафика (низкая вычислительная сложность);
- небольшая задержка при прохождении пакетов.

Недостатки пакетных фильтров следующие:

- каждый пакет анализируется вне контекста соединения и сетевого трафика;
- диапазон параметров фильтрации ограничен полями заголовков протоколов IP, TCP и UDP;
- аутентификацию с использованием IP-адреса можно обмануть, используя подмену IP-адресов (IP spoofing — атакующая система, выдающая себя за другую, используя ее IP-адрес [90]);
- отсутствие аутентификации на пользовательском уровне.

Преимуществами шлюзов уровня приложений являются:

- отсутствие непосредственного сетевого соединения между клиентом и сервером, что позволяет скрыть внутреннюю структуру сети от внешнего мира;

• наличие защиты на уровне приложений позволяет осуществлять большое число дополнительных проверок, снижая тем самым вероятность ее взлома с использованием уязвимостей программного обеспечения (ошибок в программах или в их настройке, позволяющих обойти механизмы защиты, аутентификации и т. д.).

Недостатки шлюзов уровня приложений следующие:

• высокие вычислительная сложность и нагрузка на аппаратную базу;

• производительность ниже, чем для пакетных фильтров.

Приведем уязвимости и слабые места МСЭ.

1. *Сложность защиты новых сетевых сервисов.* Как правило, МСЭ разграничивают доступ по широко распространенным протоколам, таким как HTTP, Telnet, SMTP, FTP и др. Реализуется это с помощью механизма посредников (proxy), обеспечивающих контроль трафика, передаваемого по этим протоколам. Несмотря на то что число таких посредников достаточно велико, они все-таки существуют не для всех новых протоколов и сервисов. И хотя эта проблема не столь остра (многие пользователи используют не более десятка протоколов и сервисов), при появлении в сети нового типа сервиса требуется добавление либо нового proxy, либо новых функций анализа в межсетевой экран.

2. *Возможность обхода межсетевого экрана.* Межсетевые экраны не могут защитить ресурсы корпоративной сети в случае неконтролируемого использования в ней модемов, мобильных телефонов и беспроводных точек доступа. Возможность доступа в сеть через GPRS по протоколу PPP или WiFi в обход межсетевого экрана делает ее практически незащищенной. Достаточно распространена ситуация, когда сотрудники какой-либо организации, находясь дома, с помощью программ удаленного доступа типа Remote Desktop или по протоколу SSH обращаются к данным или программам, установленным на своем рабочем компьютере, или получают через него доступ в Интернет.

3. *Незащищенность от вредоносного программного обеспечения и компьютерных атак.* Изначально межсетевые экраны не имели встроенных механизмов защиты от вредоносного программного обеспечения (компьютерных вирусов, сетевых червей [9, 91]), а также от компьютерных атак. Такая защита долгое время создавалась специализированными системами — антивирусами — и системами обнаружения атак. В настоящее время наблюдается тенденция к объединению функций межсетевых экранов, сетевых антивирусов, систем обнаружения атак и систем контроля сетевого трафика в единые комплексы смысловой фильтрации сетевого трафика. Поскольку любой поток данных можно представить как на некоторую синтаксическую конструкцию, необходимо проверять соответствие структуры потока заранее сформулированным правилам. Можно рассматривать поток с содержательной точки зрения, например как поток

команд исполняемого кода, как видеопоток, как телефонный разговор и т. д. В некоторых случаях по структуре потока можно определить его содержание (или, как говорят, семантику), а в некоторых нельзя, т. е. требуются специальные методы анализа.

4.1.8. Системы обнаружения и предотвращения атак

Будем считать состояние информационной системы безопасным, если отсутствуют нарушения конфиденциальности, целостности и доступности входящих в нее ресурсов. Под компьютерной атакой на систему будем понимать последовательность целенаправленных действий субъекта (злоумышленника), направленных на нарушение состояния ее информационной безопасности. Таким образом, компьютерная атака — это любая целенаправленная последовательность действий, нарушающая либо одно, либо сразу несколько свойств состояния информационной безопасности системы.

Создание информационных систем, гарантированно устойчивых к компьютерным атакам, сопряжено с существенными затратами как времени, так и материальных ресурсов. Кроме того, существует известная обратная зависимость между удобством пользования системой и ее защищенностью: чем совершеннее защита, тем сложнее пользоваться основным функционалом информационной системы.

В 1980-е гг. в рамках военных проектов США предпринимались попытки создания распределенных информационных систем специального назначения — MMS (Military Message System) [62], для которых математически доказывалась выполнимость основной теоремы безопасности: в системе не существует последовательности действий, которые могут привести к потере состояния ее информационной безопасности. В этих системах использовалось специализированное программное обеспечение на всех уровнях, включая операционную систему. Однако подобные системы не получили развития, и обычно для организации информационных систем используются операционные системы общего назначения, такие как ОС семейства Microsoft Windows, GNU/Linux, FreeBSD и различные клоны SysV UNIX (Solaris, HP-UX и т. д.).

Сложность и высокая стоимость разработки защищенных систем, свойства безопасности которых доказаны формально, обусловили появление и развитие направления информационной безопасности, связанного с обнаружением нарушений безопасности информационных систем [23, 38]. Данное направление получило название «обнаружение атак» (intrusion detection), и за прошедшие годы в рамках академических разработок были созданы сотни таких систем для различных платформ: от систем класса mainframe до современных опе-

рационных систем общего назначения, СУБД и распространенных приложений [16].

Принято разделять методы обнаружения атак на методы обнаружения аномалий и методы обнаружения злоупотреблений. Методы обнаружения аномалий используют описание нормального поведения наблюдаемых объектов в сети, и любое отклонение от нормального поведения считается аномальным — нарушением. Методы обнаружения злоупотреблений используют описание запрещенных действий объектов в сети, например описание известных атак, и если наблюдаемое поведение некоторого объекта сети совпадает с описанием запрещенного, то действие объекта блокируют.

Методы обнаружения злоупотреблений используются в большинстве современных коммерческих систем (Cisco IPS, ISS RealSecure, NFR) [91]. В этих системах каждая известная сетевая атака представлена сигнатурой — шаблоном значений полей заголовков сетевых пакетов и содержимого их полей данных, сопровождающих атаку. Основной недостаток методов обнаружения злоупотреблений состоит в том, что они не способны обнаруживать новые атаки, как и антивирусные системы, т. е. если описание вируса отсутствует в базе данных системы, то она не будет реагировать на этот вирус [17, 22].

Существует множество академических разработок в области обнаружения аномалий, но в промышленных системах они используются редко и с большой осторожностью, так как такие системы порождают большое количество ложных срабатываний.

Существуют два основных типа систем обнаружения атак (СОА): узловые и сетевые. Узловая СОА располагается на отдельном узле и отслеживает признаки атак на узел. Сетевая СОА находится на отдельном узле сети, через который проходит весь сетевой трафик или строго определенная его часть в данном сегменте.

Узловые СОА

Узловая СОА состоит из сенсоров и анализаторов. Задача сенсоров заключается в регистрации событий, важных с позиции безопасности на защищаемом узле: вход-выход пользователя в систему, изменения в составе программного и аппаратного обеспечения узла, взаимодействие с другими узлами сети и т. п. Анализаторы узловой СОА получают от сенсоров данные о событиях безопасности и выявляют в них признаки атак. Работа узловой СОА требует наличия ресурсов на защищаемом узле, составляющих 5...15 % от общего процессорного времени.

Выделяют три типа узловых СОА: анализаторы журналов; анализаторы системных вызовов; анализаторы поведения приложений.

Анализаторы журналов работают с журналами приложений, безопасность которых является критичной, а также с журналами операционной системы. Если такой анализатор встретит запись, соответствующую некоторому шаблону в своей базе, то он зарегистрирует событие нарушения безопасности.

Анализаторы журналов по своей природе являются реактивными системами. Иными словами, они реагируют на событие уже после того, как оно произошло. Таким образом, журнал будет содержать сведения о том, что проникновение в систему состоялось. В большинстве случаев анализаторы журналов не способны предотвратить осуществляемую атаку на систему.

Анализаторы системных вызовов следят за вызовами между приложениями и операционной системой. Сенсоры узловой СОА данного типа размещаются между операционной системой и приложениями и «подменяют» стандартные системные вызовы ОС, а часто и функции стандартных библиотек, таких как libc, библиотека сокетов и т. п. Когда приложению требуется выполнить системный вызов, параметры вызова попадают в «обертку», создаваемую узловой СОА, и анализируются в соответствии с набором заданных шаблонов. Шаблоны различаются для каждого типа приложения и для каждой известной атаки на узел.

Анализаторы системных вызовов отличаются от анализаторов журналов тем, что часто они могут предотвращать развитие атаки в том случае, если являются частью операционной системы и могут разрешать или запрещать выполнение системного вызова. Если приложение генерирует вызов, который может быть использован, например для атаки на переполнение буфера, то датчик может блокировать этот вызов и сохранить систему в состоянии информационной безопасности.

Анализатор поведения приложений оперирует описаниями нормального (разрешенного) поведения приложений, например в виде профилей разрешенных для приложений системных вызовов ОС. Анализатор наблюдает за поведением приложений, и в случае появления «запрещенного» вызова выполнение приложения принудительно останавливается.

Например, веб-серверу обычно разрешается принимать сетевые соединения через порт 80, считывать файлы в веб-каталоге и передавать эти файлы по соединениям через порт 80. Если веб-сервер попытается записать или считать файлы из другого места или открыть новые сетевые соединения, анализатор обнаружит несоответствующее спецификации поведение сервера и заблокирует такое его действие.

Сетевые СОА

Сетевая СОА располагается на узле, через который проходит бóльшая часть трафика определенной части сети. Здесь надо быть очень

аккуратным. Например если в СПД транспортной среды используются коммутаторы, а не концентраторы, то сенсор сетевой СОА, подключенный к порту коммутатора, не будет получать весь сетевой трафик сегмента сети, а будет контролировать только трафик, проходящий через порт, к которому подключен сенсор. В случае с коммутируемой СПД существуют два варианта подключения датчиков сетевых СОА: через специальный порт, отслеживающий весь трафик, проходящий через сетевой коммутатор (SPAN-порт), и через дубликатор трафика (network tap) [91].

В настоящее время большинство сетевых СОА используют для обнаружения атак методы обнаружения злоупотреблений.

Сетевые СОА обладают следующими преимуществами по сравнению с узловыми:

- сетевую СОА можно полностью скрыть в сети, и злоумышленник не будет знать о том, что за ним ведется наблюдение;
- один сенсор сетевой СОА может использоваться для мониторинга трафика в сегменте сети с большим числом систем-целей, потенциальных для атак;
- сетевая СОА может перехватывать содержимое всех пакетов, направляющихся на определенный узел.

Недостатки сетевых СОА:

- отсутствие адаптивности к неизвестным атакам, сигнатуры которых отсутствуют в базе СОА;
- сетевая СОА не может определить, была ли атака успешной (так как она не видит результата атаки);
- как правило, сетевая СОА не может просматривать зашифрованный трафик.

В настоящее время получают развитие гибридные системы, которые объединяют в себе узловые и сетевые сенсоры, сохраняя все достоинства соответствующих систем [17].

Функции систем обнаружения атак

Системы обнаружения атак могут быть использованы для различных целей: обнаружения атак, предотвращения атак, сбора доказательной базы. О первых двух целях СОА уже много говорилось, поэтому сразу перейдем к рассмотрению третьей цели.

СОА может оказаться полезной после обнаружения атаки или нарушения правил пользования ресурсами сети кем-то из сотрудников организации. Дело в том, что, как показывает статистика, не менее 40 % атак инициируют сотрудники самой организации. С помощью СОА можно собирать доказательства подобных инцидентов. Сетевую СОА можно настроить на отслеживание определенных соединений и ведение полноценного журнала по учету трафика (табл. 4.3). В то же время фиксировать все записи журнала для определенной учетной записи системы можно с помощью узловой СОА.

Таблица 4.3.

Пример использования СОА для контроля политики безопасности

Политика безопасности	Сетевая СОА	Узловая СОА
Обнаружение атак	Весь трафик, поступающий на потенциально атакуемые системы (межсетевые экраны, веб-серверы, серверы приложений и т.д.)	Неудачные попытки входа. Попытки соединения. Удачный вход с удаленных систем. Запуск запрещенных приложений
Предотвращение атак	То же	То же
Сбор доказательств	Содержимое всего трафика, формируемого на системе-цели или атакующей системе	Все успешные подключения, исходящие от атакующей системы. Все неудачные соединения от атакующих систем. Все нажатия клавиш из интерактивных сеансов на атакующих системах

4.1.9. Виртуальные частные сети

До недавнего времени многие организации для соединения с географически удаленными филиалами и с сетями других (партнерских) организаций использовали так называемые частные сети — PN (Private Network). Под частной сетью понимается объединение выделенных каналов связи, которые организация арендует у телефонных компаний и поставщиков услуг Интернета, в СПД с единой IP-адресацией и единой политикой маршрутизации. По существу это продолжение СПД транспортной среды организации. С точки зрения организации частные сети обладают множеством преимуществ перед открытыми каналами:

- передаваемая по ним информация сохраняется в секрете, так как не смешивается с трафиками других организаций;
- задержки при передаче данных между филиалами меньше, чем при использовании СПД общего доступа;
- географически удаленные пользователи взаимодействуют с интранетом организации точно так же, как если бы они физически находились в центральном офисе.

Основной недостаток частных сетей — высокая стоимость их создания и эксплуатации. Использование частных сетей — очень дорогое удовольствие.

Альтернативой частным сетям являются виртуальные частные сети — VPN (Virtual Private Network), которые обеспечивают многие преимущества частных сетей за меньшую цену. Пусть мы хотим передавать через Интернет конфиденциальные данные организации,

используя СПД общего доступа, но при этом мы должны быть уверены, что сохраняется конфиденциальность трафика. Каким же образом можно отделить свой трафик от трафика остальных пользователей сети? Это обеспечивается за счет использования шифрования и тунеллирования (см. гл. 1).

Значительная часть трафика в Интернете передается в открытом нешифрованном виде, и любой узел на пути следования пакетов может читать, сохранять и даже изменять эти пакеты. Это относится к большей части приложений, например почтовому и веб-трафику, а также сеансам связи через протокол FTP. Однако трафики шифрованного протокола терминального доступа SSH (Secure Shell) и протокола HTTPS (HyperText Transfer Protocol Secure) являются шифруемыми, и в общем случае их практически невозможно дешифровать или изменить на промежуточных узлах следования пакетов. Тем не менее протоколы SSH и HTTPS не могут быть использованы для построения VPN (далее объясним, почему).

Основные принципы функционирования виртуальных частных сетей следующие:

- трафик шифруется для обеспечения защиты от прослушивания и несанкционированного изменения;
- каждый подключаемый к VPN узел проходит аутентификацию;
- VPN прозрачен для протоколов уровня приложений, т. е. любой прикладной протокол между двумя узлами, входящими в VPN, функционирует так, как если бы эти узлы находились в родной сети организации.

Реализации VPN, как правило, состоят из двух типов компонентов: пользовательских и узловых. Пользовательские используются для подключения отдельных удаленных пользователей к сети организации, а узловые — для объединения локальных сетей географически удаленных филиалов организации в единую сеть.

Пользовательские VPN

Пользовательские виртуальные частные сети предназначены для соединения компьютера пользователя с интранетом организации, когда последний находится вне сети организации. Сервер VPN может быть межсетевым экраном организации либо отдельным VPN-сервером. Пользователь подключается к Интернету через телефонную линию к локальному поставщику услуг, канал DSL (см. гл. 2) или модем и инициирует VPN-соединение с узлом организации через Интернет.

VPN-сервер организации запрашивает у пользователя аутентификационные данные и в случае успешной аутентификации открывает ему доступ к внутренней сети организации, как если бы этот пользователь находился внутри сети. Однако скорость организованного таким образом сетевого соединения будет ограничиваться скоростью подключения пользователя к Интернету.

Одновременно с VPN-соединением с интранетом организации пользователь может работать с Интернетом или выполнять другие действия как обычный пользователь Интернета. Поддержку сети VPN на компьютере пользователя осуществляет специальное приложение — VPN-клиент.

Использование пользовательской VPN для подключения удаленного сотрудника имеет один серьезный недостаток: пользователь получает полный доступ к ресурсам сети организации так же, как если бы он находился на своем рабочем месте в офисе, но при этом на него не накладываются ограничения по взаимодействию со сторонними ресурсами в Интернете, т. е. в этом случае между пользователем и Интернетом нет ни межсетевого экрана, ни шлюза уровня приложений, ни системы обнаружения атак.

Таким образом возникает риск потери конфиденциальной информации организации из-за того, что компьютер удаленного пользователя защищен в меньшей степени, чем если бы он находился внутри периметра сети организации. Возможно заражение компьютера вирусами, установка на него троянских программ и кража аутентификационной информации для несанкционированного подключения к сети организации.

Узловые VPN

Узловые виртуальные частные сети используются организациями для объединения нескольких географически удаленных филиалов или офисов в логически единую сеть. Как правило, VPN соединяет межсетевой экран или пограничный маршрутизатор одного филиала с аналогичным устройством другого филиала.

Соединение VPN между филиалами может поддерживаться постоянно или инициироваться на каждую сессию обмена данными. Во втором случае оконечные узлы VPN (VPN-концентраторы, межсетевые экраны, пограничные маршрутизаторы) устанавливают соединение друг с другом сразу, как только какой-либо из внутренних узлов одного филиала пытается установить соединение с узлом другого филиала.

Основным недостатком узловых VPN является то, что они расширяют периметр защищаемой сети, т. е. вместо набора периметров сетей каждого из филиалов с независимой защитой получается как бы объединение всех периметров в один. В этом случае общая защищенность сети будет определяться защищенностью наиболее уязвимой сети филиала, и если этот уровень невысокий, то VPN может позволить злоумышленнику получить доступ к ресурсам центрального офиса и других филиалов.

При использовании узловых VPN особого внимания требует адресация. Если узловая VPN используется для соединения локальных сетей одной организации, то необходимо позаботиться о единой

схеме адресации для всех этих сетей, что в данном случае не представляет особой сложности. Если же VPN используется для соединения сетей двух различных организаций, потребуются особые меры, чтобы избежать коллизий, связанных с адресацией.

Компоненты VPN

Сеть VPN включает в себя четыре основных компонента:

- сервер VPN;
- алгоритмы шифрования;
- системы аутентификации;
- протокол VPN.

Конкретное сочетание алгоритмов шифрования, схемы аутентификации, используемого протокола и реализации сервера VPN определяет свойства защищенности сети, а также ее производительность. Правильность реализации архитектуры VPN зависит от правильности определения требований к защите и производительности сети, которые должна устанавливать организация до внедрения конкретной технологии VPN.

Требования к защите должны включать в себя:

- срок актуальности конфиденциальной информации организации, т.е. время, которое зашифрованный трафик должен оставаться нерасшифрованным гипотетическим злоумышленником. В соответствии с данным требованием выбирается алгоритм шифрования;
- число одновременно существующих соединений пользователей;
- ожидаемые типы соединений пользователей (сотрудники, работающие дома или находящиеся в поездке);
- число соединений с удаленным сервером;
- ожидаемый объем входящего и исходящего трафиков на удаленных узлах;
- график доступа филиалов и пользователей.

При разработке системы полезно указать дополнительные требования, связанные с ожидаемым местоположением сотрудников, а также типы служб, которые будут работать через VPN.

Сервер VPN представляет собой компьютер с надлежащим программным обеспечением, выступающий в роли конечного узла соединения VPN.

Основные функции сервера VPN:

- аутентификация узлов и пользователей VPN;
- организация прозрачного туннеля для сетевых служб;
- шифрование и дешифрование трафика, передаваемого по туннелю.

Алгоритм шифрования, используемый в VPN, должен обладать требуемым уровнем стойкости к криптоанализу. Гораздо большее значение, чем стойкость алгоритма шифрования, имеет уровень безопасности реализации VPN-системы. Неправильно реализованная

система может сделать бесполезным самый мощный алгоритм шифрования.

Для того чтобы получить доступ к информации, передаваемой через VPN, злоумышленник должен:

- захватить весь сеанс соединения, т. е. разместить устройство прослушивания между противоположными концами соединения в месте, через которое проходит весь трафик VPN;
- иметь большие вычислительные мощности и большое количество времени для перехвата ключа и дешифрования трафика.

Следовательно, злоумышленнику гораздо проще использовать уязвимость информации в компьютере пользователя либо украсть портативный компьютер, например в аэропорту или в гостинице.

Система аутентификации VPN, как уже говорилось, должна быть двухфакторной: пользователи могут проходить аутентификацию с использованием тех данных, которые только они знают и которые у них имеются, или с помощью данных о том, кем они являются (идентификационная информация).

Хорошей комбинацией средств аутентификации являются смарт-карты, где прописана вся необходимая для доступа в сеть информация, которую пользователь не знает, и индивидуальные данные пользователя, например его биометрические данные (отпечатки пальцев, цвет и форма радужной оболочки глаз и т. п.).

Протокол VPN определяет, каким образом система VPN взаимодействует с другими системами в Интернете, а также уровень защищенности трафика. Например, в протоколе VPN используется обмен ключами шифрования между двумя конечными узлами сети. Если этот обмен не защищен, злоумышленник может перехватить ключи и затем расшифровать трафик, сведя на нет все преимущества VPN.

В настоящее время стандартными для VPN являются протоколы семейства IP Security [47], касающиеся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP-пакетов. Это почти 20 предложений по стандартам и 18 RFC [11]. Также в VPN используется протокол PPTP, распространенный в сетях Microsoft Windows [50].

4.2. Примеры протоколов уровня приложений

4.2.1. Доменная система имен — DNS

Рассмотрим подробно некоторые протоколы уровня приложений, используемые в Интернете.

Как уже говорилось, каждая машина в Интернете должна иметь IP-адрес. Однако оперировать числовыми IP-адресами неудобно, поэтому рассмотрим, как в Интернете можно использовать символьные имена вместо IP-адресов и как пользователь на абонентской

машине может узнать IP-адреса других абонентских машин, зная их имена.

Все Интернет-приложения позволяют при обращении к узлам сети вместо числовых адресов использовать имена, зафиксированные в специальной распределенной базе данных DNS, которая поддерживает иерархическую систему имен для идентификации абонентских машин или узлов в сети Интернет. Такой способ адресации на прикладном уровне называется символьной адресацией.

При изучении символьной адресации в Интернете воспользуемся ее аналогией с обычной почтовой службой. Сетевые числовые адреса вполне аналогичны почтовой индексации. Машины, сортирующие корреспонденцию на почтовых узлах, ориентируются именно по индексам, и только если с индексами выходит какая-то несуразность, передают почту на рассмотрение людям, которые по адресу могут определить правильный индекс почтового отделения места назначения. Людям же приятнее и удобнее иметь дело с географическими названиями — аналогами доменных имен.

Процесс поиска адреса в Интернете аналогичен процессу поиска индекса для письма, не имеющего почтового индекса. Как определяется этот индекс? Все регионы страны пронумерованы, и это первые цифры индекса. Письмо пересылается на центральный почтамт региона, где имеется справочник с нумерацией районов этого региона — это следующие цифры индекса. Теперь письмо поступает на центральный почтамт соответствующего района, где уже известны все почтовые отделения в подопечном районе. Таким образом по географическому адресу определяется соответствующий почтовый индекс.

Аналогично определяется и адрес компьютера в Интернете, но путешествует здесь не послание, а запрос вашего компьютера об этом адресе. И в отличие от случая с почтой информация доходит до вас, как если бы районный почтамт места назначения отправлял вам письмо, любезно уведомляя на будущее об индексе, которого вы не знали.

Конечно, здесь тоже существуют свои собственные проблемы. Прежде всего, следует убедиться, что никакие компьютеры, включенные в сеть, не имеют одинаковых имен. Необходимо также обеспечить преобразование имен в числовые адреса, чтобы машины (и программы) могли понимать людей, использующих имена, поскольку техника по-прежнему общается на языке цифр.

Сначала Интернет был невелик, и иметь дело с именами было довольно просто. Организация NIC создала регистратуру. Можно было послать запрос, и в ответ получить список имен и адресов. Этот файл, называвшийся host file (файл абонентских машин), регулярно распространялся по всей сети, т.е. рассылался всем машинам. Имена были простыми словами, и все они были уникальны. Если вы использовали имя, то ваш компьютер просматривал этот файл и подставлял вместо имени реальный числовой адрес так же, как работает

телефонный аппарат со встроенным списком абонентов. Всем хватало простых имен, в сети был один Мокий, один Мокридий, один Пафнутий и одна Перепетуя.

По мере развития и расширения Интернета возрастало число абонентских машин, а следовательно, разрастался и указанный файл. Стали возникать значительные задержки при регистрации и получении имени для новых компьютеров, стало сложно найти имена, которые еще никто не использовал, и слишком много сетевого времени затрачивалось на рассылку этого огромного файла всем упомянутым в нем машинам. Стало очевидно, что при таких темпах изменений и роста сети требуется распределенная оперативная система, работающая на новом принципе. Такая система была создана, и ее назвали доменной системой имен — DNS (Domain Name Service), а способ адресации в этой системе — способом адресации по доменному принципу. Также эту систему иногда называют региональной системой наименований.

Структура региональной системы имен

Доменная система имен — это метод, при котором в сетевой группе выделяется абонентская машина, отвечающая за назначение имен машинам в группе и обладающая полнотой информации о всех именах машин группы и их IP-адресах. При этом группы первого уровня могут быть объединены в группы второго уровня, группы второго уровня — в группы третьего уровня и т. д., причем ни одна группа не может входить в две и более групп. Каждая группа в такой иерархии называется доменом. Теперь, чтобы указать путь к интересующей нас машине, достаточно перечислить имена от самого верхнего домена до самого нижнего, содержащего интересующую нас машину.

Домены в имени отделяют друг от друга точками: cs. msu. su, math. msu. su. В имени может быть различное число доменов, но практически не более пяти. Первым в имени стоит название абонентской машины — реального компьютера с IP-адресом. Это имя создано и поддерживается группой, к которой он относится (например, компьютер redsun в группе cs, относящийся к факультету вычислительной математики и кибернетики). Группа входит в более крупное подразделение msu (университетское объединение — сеть МГУ), которое, в свою очередь, является частью национальной сети (например, стран бывшего СССР, домен su).

Все пространство доменов распределено на зоны. Имена зон можно условно подразделить на организационные и географические. На высшем уровне в этой иерархии зарегистрированы следующие организационные зоны:

- com — commercial (коммерческие);
- edu — educational (образовательные);
- gov — government (правительственные);

- mil — military (военные);
- net — network (обеспечивающие работу сети);
- org — organization (некоммерческие).

В настоящее время, чтобы разгрузить домен com, создано несколько новых доменов.

В организационных зонах обычно размещаются непосредственно домены организаций.

Каждая страна (государство) имеет свой географический домен из двух букв, например:

- ae — United Arab Emirates (Объединенные Арабские Эмираты);
- au — Australia (Австралия);
- be — Belgium (Бельгия);
- br — Brazil (Бразилия);
- by — Belarus (Белоруссия).

В доменном имени слева в конце цепочки доменных имен должно быть указано имя абонентской машины. Это имя может быть собственным или функциональным. Имена собственные каждый придумывает в меру своей фантазии: машинам присваивают имена членов семьи, животных, растений, музыкантов и артистов, литературных персонажей, т.е. кто во что горазд.

Имена функциональные вытекают из функций, выполняемых машиной:

- www — сервер HTTP (WWW);
- ftp — FTP-сервер;
- ns, nss, dns — сервер DNS (Name);
- mail — почтовый сервер;
- relay — почтовый сервер обмена;
- proxy — соответствующий proxy-сервер.

Однако считается нежелательным присваивать машине функциональное имя, поскольку в любой момент может потребоваться перенести соответствующую функцию на другую машину. Лучше всего использовать псевдонимы, с помощью которых можно перенаправлять запросы к данному имени на записи, относящиеся к другому имени.

Доменная группа может создавать или изменять любые принадлежащие ей имена. Например, если группа cs решит ввести в эксплуатацию новый компьютер и назвать его chronos, то для этого ей не надо ни у кого спрашивать разрешения, все что требуется — это добавить новое имя в соответствующую часть соответствующей базы данных, и рано или поздно каждый, кому необходимо, узнает это имя. Если каждая группа придерживается этих простых правил и при этом поддерживается уникальность имен компьютеров в группе, то у всех систем в сети Интернет всегда будут разные доменные имена.

Описанный механизм аналогичен механизму присвоения почтовых адресов. Названия стран разные. Разные и названия, например, областей, республик в Российской Федерации, и эти названия утверж-

даются в государственном масштабе федеральным центром (конечно, обычно сами регионы заботятся об уникальности этих названий, поэтому здесь царит полная демократия: как республика хочет, так она и называется). В республиках — субъектах федерации — решаются вопросы о названиях районов и округов, которые в пределах одной республики также различаются. Аналогично решаются вопросы с названиями городов и улиц городов. Причем в разных городах могут быть улицы с одинаковыми названиями, например почти во всех городах СССР были улицы Ленина и Мира, однако это были улицы в разных городах.

В пределах одного населенного пункта улицы всегда имеют разные названия, причем именование улиц контролирует соответствующий центральный орган местной администрации (мэрии, сельсовета, горсовета).

Поскольку Интернет — сеть всемирная, то нужен был механизм распределения имен на самом верхнем межгосударственном уровне. Сейчас принята двухбуквенная кодировка государств, оговоренная в RFC 822 [34]. Так, например, домен «Канада» называется «ca», бывший СССР — «su», США — «us» и т.д. Всего кодов стран почти 300. Единый каталог Интернета находится в государственной организации SRI International (Менло-Парк, Калифорния, США) [90].

Поиск адреса по доменному имени

Теперь, после того как мы узнали, как соотносятся домены и создаются имена, рассмотрим, как использовать эту замечательную систему. Работает она автоматически, т.е. нам не надо разыскивать адрес, соответствующий имени, или подавать специальную команду для его поиска. Все компьютеры в Интернете могут пользоваться доменной системой.

Когда используют имя, например www.lvk.cs.msu.su, его необходимо преобразовать в IP-адрес. Для этого приложение формирует запрос к DNS-серверу, где работает DNS-служба. Эта служба — приложение, обладающее соответствующей базой данных, с помощью которой оно обслуживает такого рода запросы. Обработка имени DNS-сервером выполняется справа налево, т.е. сначала производится поиск адреса в самой верхней группе иерархии, а затем он постепенно опускается по иерархии, тем самым сужая область поиска.

В нашем примере сначала локальная DNS-служба запрашивает у DNS-службы, отвечающей за домен su, адрес DNS-службы, отвечающей за домен msu. Получив этот адрес, наша локальная DNS-служба обращается по нему, чтобы узнать адрес DNS-службы домена lvk. И только обратившись к DNS-службе домена lvk, мы узнаем адрес www-сервера в этом домене. Однако в любом случае в целях сокращения поиска на первом шаге сначала опрашивается локальный узел DNS. При этом возможны три варианта ответов:

• местный сервер знает адрес, потому что этот адрес содержится в его базе данных. Например, если вы подсоединены к сети лаборатории «Вычислительные комплексы» факультета ВМиК МГУ, то ее DNS-сервер должен обладать информацией о всех компьютерах домена lvk;

• местный сервер знает адрес, потому что кто-то недавно уже запрашивал его, и он сохранил у себя в кэш-памяти этот адрес. Когда запрашивается адрес, DNS-сервер придерживает его у себя в кэш-памяти некоторое время на случай, если кому-нибудь потребуется тот же адрес, что повышает эффективность системы;

• местный сервер адрес не знает. В этом случае запускают ранее описанную процедуру опроса DNS-серверов доменов, указанных в имени справа налево.

Серверы имен

Должно быть понятно, что нет и не может быть единого сервера, содержащего базу DNS, охватывающую весь Интернет. Его не может быть как по причине обеспечения вопросов безопасности и надежности функционирования сети Интернет, так и по причине обеспечения производительности. Чтобы сделать базу распределенной, все пространство имен доменов разбивают на непересекающиеся зоны. На рис. 4.21 показан пример такого разбиения. Границы каждой зоны определяет ее администратор. Каждая такая зона покрывает часть дерева доменов, и в нее входят серверы имен этих доменов. Обычно в каждой зоне есть основной сервер имен зоны и несколько вспомогательных серверов имен. Часто из соображений надежности сервер зоны располагают вне этой зоны.

Весь процесс поиска IP-адреса по имени домена (см. подразд. 4.2.1) реализуют серверы имен. Если запрос относится к юрисдикции того сервера имен, к которому обратились, т. е. запрашиваемый домен

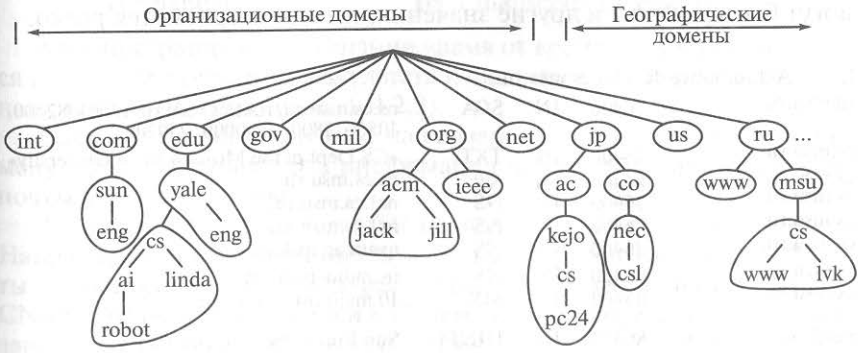


Рис. 4.21. Часть пространства доменных имен с делением на зоны

находится в ведении данного сервера имен, то этот сервер генерирует ответ, содержащий записи всех ресурсов, соответствующих запросу, и этот ответ считается авторитетным, т.е. содержащаяся в нем информация считается apriori верной. Если запрос относится к удаленному домену, то сервер имен генерирует запрос к соответствующему удаленному серверу имен.

Однако прежде чем обратиться к удаленному серверу имен обращающийся сервер посмотрит записи ресурсов в своей кэш-памяти. При этом записи в кэш-памяти не являются авторитетными. Время актуальности содержащейся в них информации определяет поле времени жизни.

Записи ресурсов

С каждым доменом связано множество ресурсов, отнесенных к этому домену. Записи об этих ресурсах хранятся в базе DNS. Когда происходит обращение к DNS-серверу с каким-либо именем, в ответ приходит не только IP-адрес, но и запись о ресурсах, соответствующих указанному имени.

Запись о каждом ресурсе состоит из пяти полей (рис. 4.22): «Имя домена» (Domain name), «Время жизни» (Time to live), «Класс» (Class), «Тип» (Type), «Источник полномочий» — SOA (Start Of Authority).

В поле «Имя домена» указывается имя домена, к которому относится эта запись. При обращении к базе DNS с таким ключом в ответ поступают все записи, у которых в этом поле указано заданное имя.

В поле «Время жизни» указывается интервал времени в секундах, в течение которого значение этого поля считается неизменным. Например, 86 400 — это число секунд в сутках. Если в этом поле указано такое число, то это значит, что запись изменяется не чаще одного раза в сутки.

В поле «Класс» указывается значение IN, если ресурс, к которому относится эта запись, является ресурсом Интернета. Однако здесь могут быть указаны и другие значения, но они встречаются редко.

```
;       Authoritative data for cs.msu.ru
cs.msu.ru.              86400   IN      SOA     ns.cs.msu.ru. root.cs.msu.ru. (2001082400,
                                                10800, 1800, 3600000, 259200)
cs.msu.ru.              86400   IN      TXT     «CS Dept of the Moscow State University»
cs.msu.ru.              86400   IN      NS      ns.cs.msu.ru.
cs.msu.ru.              86400   IN      NS      ns1.cs.msu.ru.
cs.msu.ru.              86400   IN      NS      ns1.cs.msu.net.
cs.msu.ru.              86400   IN      NS      ipsun.ac.msk.su.
cs.msu.ru.              86400   IN      NS      ns.radio-msu.net.
cs.msu.ru.              86400   IN      MX      10 mailhost.cs.msu.ru.

mailhost.cs.msu.ru.     86400   IN      HINFO   Sun Enterprise 450, Solaris 10
```

Рис. 4.22. Фрагмент возможной базы DNS для cs.msu.ru

Таблица 4.4

Основные типы записи ресурса DNS

Тип	Значение	Описание
SOA	Источник полномочий	Параметры данной зоны
A	IP-адрес хоста	32-разрядное число
MX	Обмен электронной почтой	Приоритет; домен, принимающий электронную почту
NS	Сервер имен	Имя сервера для данного домена
CNAME	Каноническое имя	Имя домена
PTR	Указатель	Псевдоним для IP-адреса
HINFO	Описание хоста	Центральный процессор и оперативная система в кодировке ASCII
TXT	Текст	Неинтерпретируемый ASCII-текст

Значения поля «Тип» указаны в табл. 4.4.

В поле «Источник полномочий» указывается имя источника информации о зоне сервера имен (об этом сервере будет сказано далее). Также здесь указывается адрес электронной почты администратора сервера имен и другая служебная информация. Если в этом поле указано значение A, то это означает, что в следующем поле указан IP-адрес этого ресурса. При указании в этом поле значения MX за ним следует имя машины, которая может получать почту для данного домена.

Покажем, что можно сделать с помощью записей типа MX. Пусть, например, есть компания «Horn and Hoof». Эта компания по каким-то причинам не хочет или не может иметь выделенный почтовый сервер. Тогда она может, предварительно договорившись с факультетом ВМиК МГУ, попросить внести в базы данных DNS соответствующего домена следующую запись:

HornandHoof.ru IN 1 mailserver.cs.msu.ru

Администратор этой компании время от времени будет связываться с почтовым сервером факультета и скачивать почту по протоколу, например POP3 (см. подразд. 4.2.3).

Записи типа NS указывают на серверы имен, относящиеся к домену верхнего уровня. Эта информация необходима при пересылке почты в другие домены.

Записи типов CNAME и PTR позволяют создавать псевдонимы. Например, человек может иметь несколько адресов электронной почты, но все они будут относиться к одному почтовому ящику. Запись CNAME отличается от записи PTR тем, что интерпретация последней зависит от контекста ее использования.

Запись типа HINFO позволяет определять тип машины и операционной системы соответствующего ресурса.

Замечания по региональной системе имен

Распространено несколько заблуждений, с которыми можно столкнуться, имея дело с именами, поэтому сделаем несколько замечаний по организации и работе доменной службы имен.

Доменная служба имен указывает на ответственного за поддержку имени, т. е. в чьем ведении находится поддержка информации об этом имени. Однако она ничего не сообщает о владельце компьютера, т. е. где эта машина находится географически (несмотря на коды стран). Вполне возможно существование в Антарктиде машины с именем inr.msk.su. Это, конечно, ненормально, но никаким законам не противоречит.

Понятия доменного имени и сети не связаны. Часто доменные имена и сети перекрываются, и жестких связей между ними нет, т. е. две машины одного домена могут не принадлежать к одной сети. Например, системы io.cs.msu.su и fox.cs.msu.su могут находиться в совершенно разных сетях, поскольку доменные имена указывают ответственного за домен.

У машины может быть много имен. В частности, это верно для машин, предоставляющих какие-либо службы, которые в будущем могут быть помещены на другую машину. Когда эти службы будут перемещены, то имя, под которым такая машина выступала в качестве их сервера, будет передано новой машине-серверу вместе с услугами. При этом для внешних пользователей ничего не изменится, т. е. они будут продолжать пользоваться этой службой, запрашивая ее по имени, независимо от того, какой компьютер на самом деле реализует ее. Имена, по смыслу относящиеся к службе и называемые каноническими, в Интернете встречаются довольно часто.

Для связи имена необязательны. Например, вам может прийти сообщение «адресат неизвестен», означающее, что Интернет не может преобразовать использованное вами имя в адрес, т. е. имя более недействительно в том виде, в котором его знает ваш компьютер. Однажды заполучив числовой эквивалент имени, ваша система перестает использовать для связи на машинном уровне доменную форму адреса. Запоминать лучше имена, а не числовые адреса. Некоторым кажется, что система имен это еще одно звено в цепи, которое может выйти из строя, но эти адреса привязаны к конкретным точкам сети. Если компьютер, предоставляющий некие услуги, переносится из одного здания в другое, его сетевое расположение, а значит, и адрес скорее всего изменятся. Имя же менять не требуется и не следует. Когда администратор присваивает новый адрес, ему необходимо только обновить запись имени в базе данных таким образом, чтобы обновленное имя указывало на новый адрес, а поскольку имя работает по-прежнему, вас совершенно не должно заботить то, что компьютер расположен уже в другом месте.

Доменная система имен возможно и выглядит сложной, но она является одной из составляющих, делающих общение с сетью более простым и удобным. Несомненное преимущество доменной системы состоит в том, что она разбивает громадье Интернета на набор вполне обозримых и управляемых частей. Хотя сеть включает в себя миллионы компьютеров, все они поименованы, и именование это организовано в удобной и рациональной форме, упрощающей работу [42, 53].

4.2.2. Протокол управления сетью — SNMP

Когда сеть из компьютеров охватывает небольшие пространства (несколько комнат, этаж), то в случае возникновения каких-то неполадок можно обойти все помещения и проверить работоспособность каждого устройства и его программного обеспечения. Когда сеть охватывает большие территории и включает в себя оборудование, принадлежащее разным организациям, то такой обход уже невозможен, хотя бы потому, что потребуется много времени на согласование доступа в помещения разных организаций, а также на сам обход и проверку устройств.

Поэтому, когда сети стали охватывать большие территории, потребовались адекватные средства для управления ими. Под словом «управление» в данном случае будем понимать возможность оперативно получать информацию о состоянии каждого устройства в сети: о его работоспособности и истории функционирования.

В 1990 г. была опубликована первая версия протокола управления сетью (Simple Network Management Protocol — SNMP v.1). В RFC 1155 [79] и RFC 1157 [29] было описано, как организовано систематическое наблюдение за сетью (какая, как и где может накапливаться информация) и управление ею (как и какие параметры работы устройств сети можно изменять). Этот протокол получил широкое распространение и был реализован практически во всех устройствах, используемых в сетях.

Опыт использования протокола SNMP v.1 выявил в нем ряд недостатков. Например, в нем недостаточно были проработаны вопросы безопасности. При опросе устройств явно указывался пароль, глядя на который, устройство аутентифицировало запрашивающее устройство.

Во второй версии протокола SNMP (RFC 1441...1452) была введена криптографическая защита механизма аутентификации. Далее кратко рассматривается именно SNMP v.2.

Модель управления

Модель управления, принятая в протоколе SNMP, показана на рис. 4.23. Эта модель использует четыре типа сущностей:

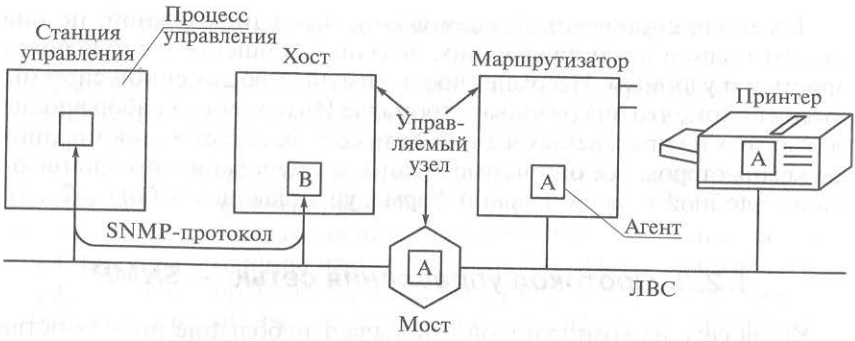


Рис. 4.23. Устройство модели управления SNMP

- станции управления;
- управляемые устройства;
- управляющая информация;
- протокол управления.

Управляют сетью станции управления, т. е. компьютеры, на которых выполняются процессы, собирающие и накапливающие информацию об управляемых устройствах в сети. Сбор этой информации происходит по запросу от управляющей станции к управляемому устройству. Запросы, передача и другие действия выполняются с помощью команд протокола SNMP.

На управляемых устройствах работают специальные SNMP-агенты (далее для краткости просто агенты), которые выполняют команды, передаваемые с помощью SNMP-протокола, и фиксируют определенный набор параметров функционирования управляемого устройства. Управляемым устройством может быть маршрутизатор, мост, рабочая станция, устройство печати, т. е. любое устройство, где может работать SNMP-агент. Каждый агент поддерживает локальную базу данных MIB (Management Information Base). В этой базе хранится информация о состоянии агента, история его функционирования и переменные, характеризующие работу устройства, где функционирует агент.

Структура управляющей информации — SMI

В сети используется аппаратура сотен различных производителей. Естественно, агент должен формировать данные о функционировании управляемого устройства в некотором унифицированном виде, например по составу или способу представления независимо от того, кто изготовил это устройство.

В соответствии с терминологией, принятой в стандарте протокола SNMP, переменную, в которой агент накапливает информа-

цию, будем называть объектом. Все объекты собираются в группы, определяемые стандартом, а группы — в модули. Чтобы все объекты имели единые правила идентификации, поступают следующим образом. Строят дерево стандарта, в котором отражают иерархию используемых понятий, и это дерево является поддеревом дерева стандартов.

На рис. 4.24 [20] показана часть такого дерева, в первом ярусе которого расположены названия организаций, имеющих право выпускать международные стандарты (ISO, МСЭ). Есть в этом дереве и место для Интернета — ярус 4. В последнем ярусе указаны группы. В скобках рядом с именем каждой группы указано число объектов в ней. Объекты могут быть следующих примитивных типов: Integer, Bit String, Octet String, Null, Object Identifier, причем последний тип — это, по существу, указатель на объект. Любой объект в любом стандарте можно представить через Object Identifier. Все объекты в этом дереве могут быть заданы указанием пути.

SMI (Structure of Management Information) — это в определенном смысле язык для определения структур данных, представляющих собой объекты в базе данных MIB.

В табл. 4.5 указаны типы данных, используемые для определения объектов, отслеживаемых протоколом SNMP.

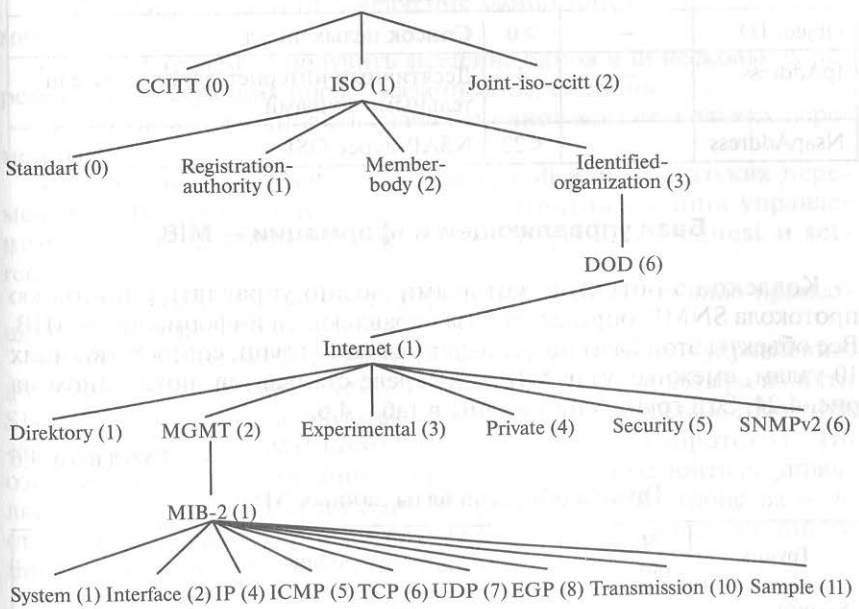


Рис. 4.24. Подмножество дерева стандартов

Таблица 4.5

Типы данных, используемых при мониторинге через протокол SNMP

Имя	Тип	Байт	Значение
Integer	Числовой	4	Целое число (32 бит в текущих реализациях)
Counter32	—	4	Беззнаковый 32-битовый счетчик с переносом на новую строку
Gauge32	—	4	Беззнаковое значение без перехода на новую строку
Integer32	—	4	32 бит, даже на 64-битовом процессоре
UInteger32	—	4	Как у Integer32, но беззнаковый
Counter64	—	8	64-битовый счетчик
TimeTicks	—	4	В сотых долях секунды с определенного начала отсчета
Bit String	Строка	4	Битовое отображение от 1 до 32 бит
Octet String	—	≥0	Строка бит переменной длины
Opaque	—	≥0	Устарело. Используется только для обеспечения совместимости с ранними версиями
Object ID	—	>0	Список целых чисел
IpAddress	—	4	Десятичный интернет-адрес с разделительными точками
NsapAddress	—	<22	NSAP-адрес OSI

База управляющей информации — MIB

Коллекцию объектов, которыми можно управлять с помощью протокола SNMP, определяет база управляющей информации — MIB. Все объекты этой базы подразделяются на 10 групп, соответствующих 10 узлам, смежных узлу MIB-2 в дереве стандартов, показанном на рис. 4.24. Эти группы приведены в табл. 4.6.

Таблица 4.6

Группы объектов базы данных MIB-2

Группа	Число объектов	Описание
System	7	Название, местоположение и описание оборудования

Окончание табл. 4.6

Группа	Число объектов	Описание
Interfaces	23	Сетевые интерфейсы и их измеряемый трафик
AT	3	Трансляция адреса
IP	42	Статистика IP-пакета
ICMP	26	Статистика полученных ICMP-сообщений
TCP	19	Алгоритмы, параметры и статистика ICMP
UDP	6	Статистика трафика UDP
EGP	20	Статистика трафика протокола EGP
Transmission	0	Зарезервирована для обусловленных средой MIB
SNMP	29	Статистика трафика SNMP

Управление в сети с помощью протокола SNMP

SNMP-протокол определяет пять типов сообщений, которыми обмениваются станция управления и управляемое устройство (рис. 4.25):

- get-request — получить значение одной или нескольких переменных;
- get-next-request — получить значение одной или нескольких переменных, следующих после указанной переменной;
- set-request — установить значение одной или нескольких переменных;
- get-response — выдать значение одной или нескольких переменных. Это сообщение возвращается агентом станции управления в ответ на операторы get-request, get-next-request и set-request;
- trap — уведомить станцию управления, когда что-либо произошло с агентом.

Первые три из этих сообщений использует станция управления, а последние два — управляемое устройство. Так как четыре из пяти SNMP-сообщений реализуются простой последовательностью типа «запрос—ответ», SNMP-протокол использует UDP-протокол. Это означает, что запрос от станции управления может не дойти до управляемого устройства, как и отклик от управляемого устройства — до станции управления. В этом случае будет задействован механизм time-out и выполнена повторная передача.

Станция управления отправляет все три запроса на UDP-порт 161. Управляемое устройство устанавливает ловушки (программные прерывания trap) на UDP-порт 162. Так как используются два разных

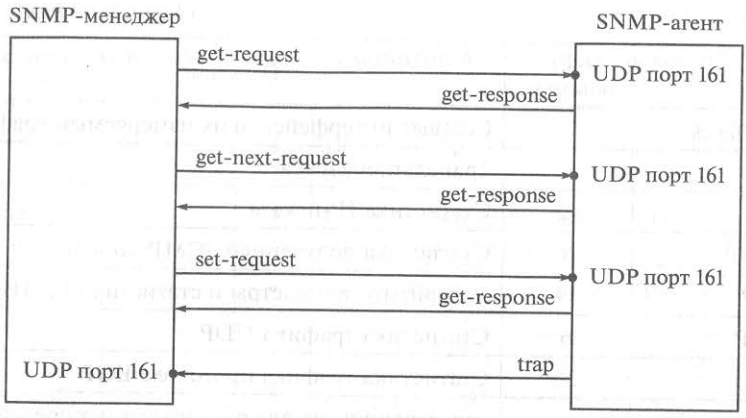


Рис. 4.25. Пять операторов SNMP

порта, одна и та же система может выступать и как станция управления, и как управляемое устройство.

На рис. 4.26 показан формат SNMP-сообщений, инкапсулированных в UDP-дейтаграмму. Здесь в байтах указаны только размеры IP- и UDP-заголовков, а значение поля «Версия» равно 0. В действительности это значение равно номеру версии, уменьшенному на 1.

В табл. 4.7 показано значение поля «Тип блока данных протокола» (PDU type).

При взаимодействии между станцией управления и управляемым устройством используется пароль, представляющий собой 6-символьную строку, которую в SNMP v.1 передавали в открытом виде. В операторах get, get-next и set станция управления устанавливает иденти-

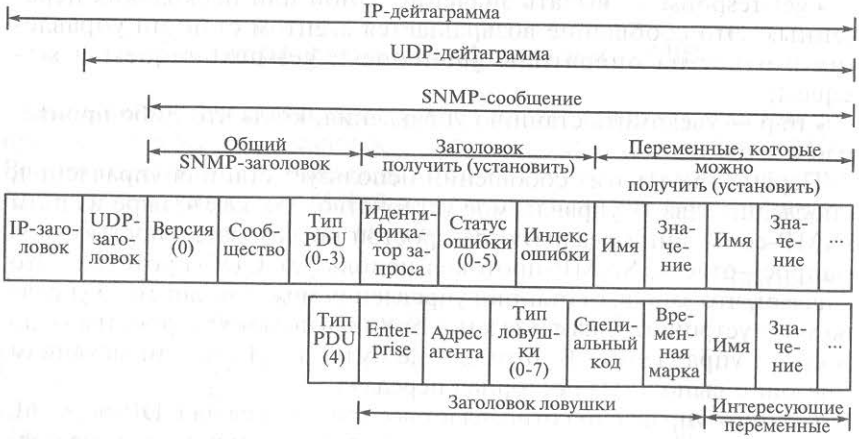


Рис. 4.26. Формат SNMP-сообщений

Таблица 4.7

Типы PDU-сообщений SNMP

Тип PDU	Имя
0	get-request
1	get-next-request
2	set-request
3	get-response
4	trap

Таблица 4.8

Значения поля статуса ошибки SNMP

Статус ошибки	Имя	Описание
0	noError	Все в порядке
1	tooBig	Устройство не может поместить отклик в одно SNMP-сообщение
2	noSuchName	Запрос указывает на несуществующую переменную
3	badValue	В запросе на установку использовано недопустимое значение или сделана ошибка в синтаксисе
4	readOnly	Станция управления попыталась изменить переменную, которая помечена как «только для считывания»
5	genErr	Неопознанная ошибка

фикатор запроса (request ID), который возвращается управляемым устройством в сообщении get-response, что повышает безопасность при взаимодействии. Это поле также позволяет станции управления выдать несколько запросов одному или нескольким устройствам, а затем отсортировать полученные отклики.

Статус ошибки (error status) — это целое число, которое возвращается агентам и указывает на ошибку. В табл. 4.8 показаны значения, имена и описания ошибок.

Более подробно SNMP-протокол рассматривается в [82, 87].

4.2.3. Электронная почта

Поначалу возможности электронной почты сводились к передаче файлов с одним ограничением: первая строка файла должна была

содержать адрес получателя. Со временем этого оказалось недостаточно в силу следующих обстоятельств:

- посылать одно и то же сообщение сразу нескольким получателям было неудобно;
- сообщение не имело внутренней структуры, что усложняло его обработку на машине;
- отправитель никогда не знал, получено сообщение или нет;
- если, отправляясь в командировку, кто-то хотел перенаправить свои сообщения кому-то другому, это было невозможно;
- интерфейс пользователя был неудобен, поскольку пользователь должен был от работы в редакторе файлов переходить в систему отправки файлов;
- было невозможно отправить в одном и том же сообщении и текст, и голос, и видео.

Исторически первыми были системы в архитектуре TCP/IP, описанные в RFC 821 и 822. В 1984 г. появилось решение X.400 в рамках эталонной модели OSI/ISO. Десять лет спустя X.400 использовалось лишь в единичных организациях, почти везде использовалось решение Sendmail в архитектуре TCP/IP [81].

Архитектура и сервис

Архитектура почтовой системы включает в себя два основных компонента: агента пользователя и агента передачи сообщений. Первый отвечает за интерфейс с пользователем, составление и отправку сообщений, а второй — за доставку сообщения от отправителя к получателю.

Обычно почтовая система поддерживает пять базовых функций.

1. *Композиция*. Обеспечивает создание сообщений и ответов. Хотя для формирования тела сообщения может использоваться любой текстовый редактор, система автоматизирует заполнение многочисленных полей заголовка сообщения. Например если формируется ответ, система автоматически выделит адрес из исходного сообщения и подставит его как адрес получателя.
2. *Передача*. Обеспечивает передачу сообщения от отправителя к получателю без вмешательства пользователей.
3. *Отчет о доставке*. Передает было ли сообщение доставлено, отвергнуто или потеряно. Для многих приложений эти отчеты крайне важны.
4. *Визуализация сообщения*. Выполняет перекодировку сообщения, изменение формата и т.д.
5. *Размещение*. Определяет, что делать с сообщением: уничтожить после (до) прочтения или, если сохранить, то где. Поиск интересующего сообщения, перенаправление сообщения, повторное прочтение ранее полученного сообщения относятся также к данной функции.

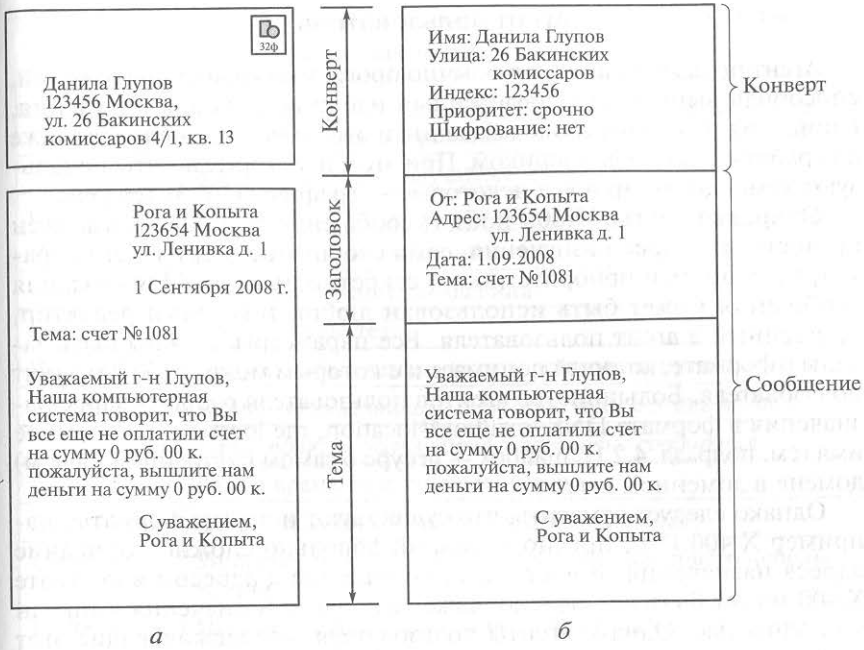


Рис. 4.27. Примеры почтового (*а*) и электронного (*б*) писем

Кроме указанных обязательных функций в большинстве почтовых систем имеется и ряд других функций. Например если пользователь уехал, он может перенаправить поступающие в его отсутствие сообщения куда-либо еще. Во многих системах пользователь может создавать так называемые внутренние почтовые ящики для поступающих сообщений; создавать лист рассылки, по которому одно и то же сообщение будет разослано всем его участникам; сортировать сообщения по определенным директориям в зависимости от их характеристик и многое другое.

Ключевой функцией всех современных почтовых систем является разделение почтового отправления на конверт сообщения и собственно сообщение. Система доставки использует только конверт, содержащий всю необходимую ей информацию о сообщении: адрес назначения, приоритет, секретность, требование об уведомлении и т. д.

Сообщение внутри конверта имеет заголовок и тело. Заголовок содержит всю необходимую информацию о теле для агента пользователя, а тело предназначено исключительно для пользователя. Примеры почтового и электронного писем приведены на рис. 4.27.

Агент пользователя

Агент пользователя — это обычно программа уровня приложений, способная выполнять определенный набор команд для получения, написания и композиции сообщения и ответа на сообщение, а также для работы с почтовым ящиком. При этом некоторые агенты используют командную строку, а некоторые — графический интерфейс.

Отправка почты. Чтобы послать сообщение, пользователь должен предоставить адрес назначения, само сообщение и другие его параметры, например приоритетность, секретность и т. п. Для создания сообщения может быть использован любой текстовый редактор, встроенный в агент пользователя. Все параметры должны быть заданы в формате, который понимает и с которым может работать агент пользователя. Большинство агентов пользователя ожидает адрес назначения в формате DNS: mailbox@location, где location — доменное имя (см. подразд. 4.2.1), mailbox — ресурс в самом внутреннем (левом) домене в доменном имени.

Однако следует отметить, что существуют и другие форматы, например X.400 [80], предполагающий довольно сложное описание адреса назначения. В частности, сообщение с адресом в формате X.400 может быть доставлено, даже если адрес назначения выписан не полностью. Однако агенты пользователя, поддерживающие этот формат, стали доступны намного позднее службы, описанной в RFC 822 [35], поэтому такой формат адреса не прижился.

Агент пользователя также поддерживает лист рассылки, который позволяет рассылать одно и то же сообщение сразу нескольким пользователям. Причем сообщение размножается необязательно самим агентом, а там, где поддерживается лист рассылки.

Чтение почты. Прежде чем агент пользователя (далее АП) что-либо высветит на экране при загрузке, он просмотрит почтовый ящик на предмет новых поступлений и высветит на экране его содержимое с краткой аннотацией каждого сообщения. В простых почтовых АП высвечиваемые поля встроены в АП, а в развитых — пользователь сам определяет, что показывать, а что нет (эта информация содержится в файле user profile).

Формат сообщений

Рассмотрим формат самого сообщения. Начнем с формата, описанного в RFC 822, а затем перейдем к его мультимедийному расширению.

Формат RFC 822. Сообщение состоит из простейшего конверта (описанного в RFC 821), полей заголовка, пустой строки и тела сообщения. Каждая строка заголовка — это строка ASCII-текста, содержащая название поля, двоеточие и какое-то значение. Стандарт 822 не различал четко заголовок и конверт. В современных почтовых систе-

Таблица 4.9

Поля заголовка RFC 822

Заголовок	Значение
Поля, относящиеся к транспортировке сообщения	
To:	Адрес основного получателя
Cc:	Адрес дополнительного получателя
Bcc:	Адрес для скрытых копий
From:	Создатель данного сообщения
Sender:	Адрес отправителя
Received:	Строка, добавляемая каждым агентом на пути сообщения
Return-Path:	Запись, определяющая обратный путь к отправителю
Некоторые поля, используемые в заголовке сообщения	
Date:	Дата и время отправки сообщения
Reply-To:	Адрес, на который требуется отправить ответ
Message-Id:	Уникальный номер сообщения для дальнейшего использования
In-Reply-To:	Идентификатор сообщения, ответом на которое является текущее сообщение
References:	Другие релевантные идентификаторы сообщения
Keywords:	Ключевые слова, выбранные пользователем
Subject:	Краткий отрывок сообщения для отображения одной строкой

мах это различие более четкое, и агент пользователя имеет дело с заголовком, а агент передачи — с конвертом, формируемым на основе заголовка.

Наиболее важные поля заголовка приведены в табл. 4.9.

Формат MIME (Multipurpose Internet Mail Extension). Когда Интернет только начинал развиваться, почтовые системы способны были передавать только текстовые сообщения на английском языке в формате ASCII. Для этих целей RFC 822 было достаточно. В наши дни этих возможностей уже недостаточно. Необходимо, чтобы почтовая система умела работать:

- с сообщениями на европейских языках (на французском, немецком и т.д.);
- с сообщениями не в латинском алфавите (на русском, арабском и т.д.);
- с сообщениями вне алфавита (на японском, китайском);
- с сообщениями, содержащими не только текст (звук, видео, графику).

Таблица 4.10

Пять заголовков, определенных для MIME в RCF 822

Заголовок	Значение
MIME-Version:	Идентифицирует версию MIME
Content-Description:	Строка, описывающая содержимое сообщения
Content-Id:	Уникальный идентификатор
Content-Transfer-Encoding:	Способ подготовки тела письма к передаче
Content-Type:	Тип сообщения

Решение указанной проблемы, предложенное в RFC 1341 [25] и RFC 1521 [26], называется MIME — многоцелевое расширение почтовой службы в Интернете. Основная идея MIME — расширение RFC 822 в целях структурирования тела сообщения и введения правил кодировки ASCII-сообщений. Естественно, что введение MIME повлияло на программы доставки и отправки сообщений.

В формате MIME определены пять новых заголовков, указанных в табл. 4.10.

Заголовок MIME-Version сообщает агенту пользователя, что он имеет дело с MIME-сообщением и какая версия MIME используется. Заголовки Content-Description и Content-Id характеризуют сообщение. Например, второй заголовок можно использовать для фильтрации сообщений.

Заголовок Content-Transfer-Encoding определяет подготовку сообщения для передачи через сеть, для чего используются четыре основных схемы.

Простейшая схема применяется для передачи ASCII-текста — 7 бит на символ (для учета национальных алфавитов используется схема 8 бит на символ) при условии, что длина строки не превышает 1 000 символов в строке.

Для корректной передачи двоичных данных (например, исполняемого кода программ) используется схема base64 encoding, которая разбивает сообщение на блоки по 24 бит. Каждый блок разбивается на четыре группы по 6 бит каждая.

Для сообщений, которые являются «почти» ASCII-сообщениями с небольшими исключениями, используется схема quoted-printable encoding.

Можно также указать и какую-то особую схему в поле Content-Transfer-Encoding.

В поле заголовка Content-Type указывается тип сообщения. Возможные значения этого поля указаны в табл. 4.11.

Таблица 4.11

Типы и подтипы сообщений MIME

Тип сообщения	Подтип сообщения	Описание сообщения
Text	Plain	Неформатированный текст. Используется для передачи простого текста
	Richtext	Текст с простым форматированием. Используется для передачи текста в формате RTF
Image	Gif	Изображение в формате GIF
	Jpeg	Изображение в формате JPEG
Audio	Basic	Звук. Сообщает агенту пользователя, что данное поле содержит звуковые данные
Video	Mpeg	Фильм в формате MPEG
Application	Octet-stream	Неинтерпретируемая последовательность байтов. Используется для передачи вложений, которые имеют тип, не описанный в MIME
	Postscript	Готовый к печати документ в формате PostScript
Message	Rfc822	Сообщение MIME, описанное в RFC 822
	Partial	Сообщение, перед передачей разбитое на несколько частей
	External-body	Непосредственно сообщение, которое следует передать через сеть
Multipart	Mixed	Независимые части сообщения в определенной последовательности. Используется для передачи сложных документов, содержащих данные различных типов MIME, а также для передачи сообщений с электронной подписью
	Alternative	То же в различных форматах
	Parallel	Части сообщения, которые необходимо просматривать вместе
	Digest	Части сообщения, являющиеся самостоятельными сообщениями в RFC 822

Передача сообщений

Основная задача системы передачи почтовых сообщений — надежная доставка сообщения от отправителя к получателю. Самым

простым способом в этом случае является использование простого протокола передачи почты — SMTP (Simple Mail Transfer Protocol).

В Интернете почта передается следующим образом. Машина-отправитель устанавливает TCP-соединение с 25-м портом машины-получателя, на котором находится почтовый демон, работающий по протоколу SMTP. Этот порт принимает соединение и распределяет поступающие сообщения по почтовым ящикам машины-получателя.

После установки соединения машина-отправитель работает как клиент, а машина-получатель — как сервер. Сервер посылает текстовую строку, идентифицирующую его и готовность принимать почту. Если он не готов принимать почту, то клиент разрывает соединение и повторяет всю процедуру позднее. Если сервер подтвердил свою готовность принимать сообщение, то клиент сообщает, от кого и кому оно предназначено. Если сервер подтвердил наличие получателя, то он дает команду клиенту и сообщение передается без контрольных сумм и подтверждений, так как TCP-соединение обеспечивает надежный поток байтов. Если сообщений несколько, то все они передаются. Обмен по соединению происходит в обоих направлениях.

Приведем пример мультимедийного сообщения:

```
From: elinor@abc.com
To: carolin@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abc.com>
Content-Type: multipart/alternative;
Boundary-qwerlyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble.
The user agent ignores it. Have a nice day.

-- qwerlyuiopasdfghjklzxcvbnm
Content-Type: text/richtext

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

-- qwerlyuiopasdfghjklzxcvbnm--
Content-Type: message/external-body;
        access-type="anon-ftp";
        site="bicycle.abc.com";
        directory="pub";
        name="birthday.snd"
```

```
content-type: audio/basic
content-transfer-encoding: base 64
--qwerlyuiopasdfghjklzxcvbnm--
```

Обратите внимание, что заголовок Content-Type в нем встречается трижды. Первый раз он указывает, что сообщение состоит из альтернативных мультимедиачастей: текста со своими заголовками и аудиочасти также со своим заголовком.

Заметим, что клиент всегда посылает четырехсимвольные команды, а команды сервера в основном цифровые.

В приведенном примере сообщение передается только одному получателю:

```
                S: 220 xyz com SMTP service ready
C: HELO abc.com
                S: 250 xyz com says hello to abc com
C: MAIL FROM: <elinor@abc.com>
                S: 250 sender ok
C: RCPT TO: <carolin@xyz.com>
                S: 250 tecipient ok
C: DATA
                S: 354 Send mail: end with " " on a line by
                itselt
C: From: elinor@abc.com
C: To: carolin@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abc.com>
C: Content-Type: multipart/alternative; birthday=
   qwerlyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have
   a nice day.
C:
C: qwerlyuiopasdfghjklzxcvbnm
C: Content-Type: text/richtext
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: qwerlyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:      access-type="anon-ftp";
C:      site="bicycle.abc.corn";
```

```
C:      directory="pub";
C:      name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base 64
C: --qwerlyuiopasdfghjklzxcvbnm--
C:
                S: 250 message accepted
C: QUIT
                S: 221 xyz com closing connection
```

Однако получателей может быть несколько. В этом случае используют несколько RCTP-команд.

У SMTP-протокола, несмотря на то что он хорошо описан в RFC 821, имеются следующие недостатки:

1. Длина сообщения не может превосходить 64 Кбайт.

2. Наличие time-out. Если время ожидания подтверждения у отправителя и получателя не согласовано, то один будет разрывать соединение, не дождавшись, тогда как другой просто будет очень загружен.

3. Возможность возникновения почтового урагана. Пусть машина-получатель имеет лист рассылки, где указана машина-отправитель, и наоборот. Тогда отправка сообщения по листу рассылки вызовет бесконечно долгие обмены сообщениями между этими машинами.

Для преодоления этих проблем в RFC 1425 был описан протокол ESMTP, по которому клиент сначала посылает команду EHLO, и если она отвергается сервером, это означает, что сервер работает по SMTP.

Почтовые шлюзы. Протокол SMTP хорош, когда обе машины находятся в Интернете. Однако это не всегда так. Многие компании в целях сетевой защиты соединяют свои сети через надлежащие средства либо используют другие протоколы. Например, отправитель или получатель могут использовать протокол X.400 (рис. 4.28). В этом случае отправитель передает сообщение шлюзу, тот его буферизует и

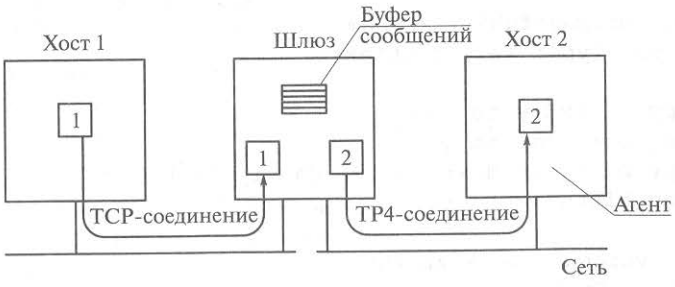


Рис. 4.28. Передача электронного письма с использованием почтового шлюза на прикладном уровне

позднее передает получателю. Звучит просто, но на самом деле возникают следующие проблемы:

1. Соответствие адресов.
2. Соответствие структур конвертов и заголовков.
3. Соответствие структуры тела сообщения.

Например, что делать в случае если тело содержит аудиофайл, а на стороне получателя с ним работать не умеют, или, что делать если отправитель поставил условие «если передача сообщения не пройдет по почтовому соединению, повторить его по факсу», а получатель не умеет работать с факсом. Однако для простых неструктурированных ASCII-сообщений SMTP-шлюз способен решить такие проблемы.

Доставка получателю. До сих пор мы предполагали, что машина пользователя может и отправлять сообщения, и получать их. Однако часто машина пользователя — это персональный компьютер или ноутбук, которые время от времени связываются с почтовым сервером, чтобы отправить или получить почту.

Как это происходит? Самый простой протокол для изъятия почты из удаленного почтового ящика — POP3 (Post Office Protocol), описанный в RFC 1225 [63], позволяет входить в удаленную систему и выходить из нее, передавать письма и принимать их, а главное — он позволяет забирать почту с сервера и хранить ее на машине пользователя.

Более сложный протокол IMAP — Interactive Mail Access Protocol, описанный в RFC 1064 [34], позволяет одному и тому же пользователю заходить с разных машин на сервер, чтобы прочесть или отправить почту. Сервер в этом случае, по существу являющийся удаленным хранилищем писем, позволяет, например, получать доступ к письму не только по его номеру, но и по содержанию.

Часто используемый протокол — DMSP (Distributed Mail System Protocol, описанный в RFC 1056, не предполагает, что пользователь работает все время с одной и той же почтовой службой, т. е. пользователь может обратиться к серверу и забрать почту на свою локальную машину, после чего разорвать соединение. Обработав почту, он может ее отправить позднее, когда будет установлено очередное соединение.

Важными почтовыми сервисами являются:

- фильтры (спам-фильтры, сортировщик почты и т. д.), которые в UNIX-системах часто реализуются с помощью программы procmail [75];
- возможность пересылки поступающей почты на другие адреса, которая в UNIX-системах легко настраивается через файл $HOME/.forward;
- демон отсутствия, который в UNIX-системах настраивается утилитой vacation;
- почтовый робот (программа, анализирующая входящие письма и отвечающая на них).

Пославший почту, естественно, предполагает, что ее никто не читает кроме адресата. Однако если об этом специально не позаботиться, гарантировать этого нельзя. Рассмотрим две широко распространенные безопасные почтовые системы — PGP и PEM.

PGP (Pretty Good Privacy). Почтовая служба с «вполне хорошей конфиденциальностью» — PGP — разработана Ф. Зиммерманом [97]. Это полный пакет программ для обеспечения безопасности электронной почты, который включает в себя средства конфиденциальности, установления подлинности, электронной подписи и построения профиля сообщения в удобной для использования форме. Благодаря тому что эта разработка качественная, работающая как на платформе UNIX, так и на платформах MS-DOS/Windows и Macintosh, и распространяющаяся бесплатно, она очень широко используется.

Ф. Зиммерман был обвинен в нарушении ряда законов США о шифровании. Дело в том, что в США действует закон, ограничивающий экспорт вооружений, под действие которого подпадают системы и алгоритмы шифрования. Ф. Зиммерман передал свою разработку приятелю из Швейцарии, а тот выложил ее в Интернете.

PGP, используя алгоритмы шифрования RSA, IDEA и MD5 (см. подразд. 4.1.3.), поддерживает компрессию передаваемых данных, их секретность, электронную подпись и средства управления доступом к ключам.

Схема работы PGP показана на рис. 4.29.

Отметим, что секретный ключ для IDEA, строящийся автоматически в ходе работы PGP на стороне пользователя A и называемый ключом сессии — K_A, шифруется алгоритмом RSA с открытым ключом пользователя B. Также следует обратить внимание на то, что медленный алгоритм RSA используется для шифрования коротких фрагментов текста: 128-разрядного ключа для MD5 и 128-разрядного ключа для IDEA.

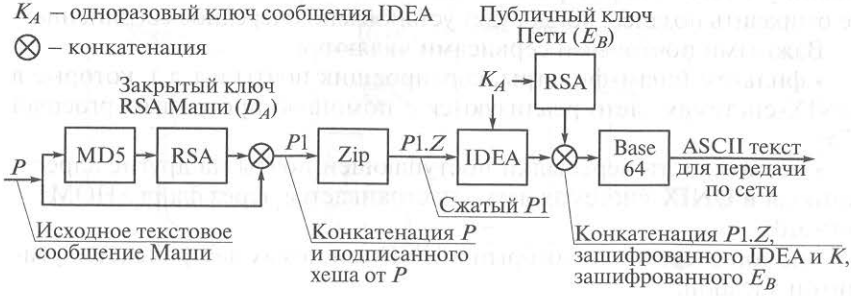


Рис. 4.29. Схема работы PGP при отправке сообщения

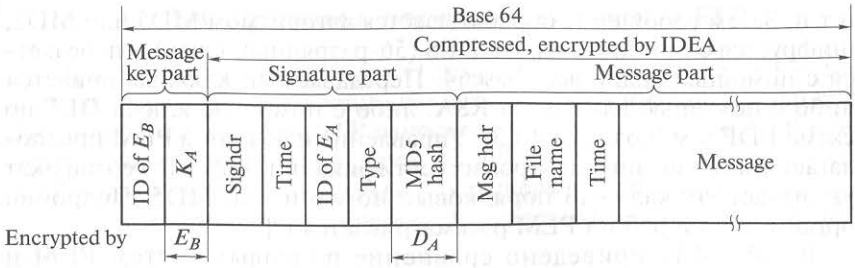


Рис. 4.30. Формат PGP-сообщения

Служба PGP первых версий поддерживала три длины ключей:

- обычную — 314 бит (ключ может быть раскрыт за счет больших затрат);
- коммерческую — 512 бит (ключ может быть раскрыт специализированными организациями, названия которых, как правило, состоят из трех букв, например ЦРУ, АНБ, ФСБ, МВД, ФБР);
- военную — 1 024 бит (ключ, который в теории не может быть раскрыт пока никем на планете Земля*).

В настоящее время существуют современные версии стандарта PGP (например, GnuPG), в которых используется алгоритм DSA, и размер ключа де-факто никак не ограничивается.

Формат PGP-сообщения, показанный на рис. 4.30, включает в себя следующие поля:

- ID of E_B — ID ключа получателя;
- K_A — ключ сообщения, зашифрованный публичным ключом получателя;
- Sighdr — заголовок подписи;
- Time — метка времени подписи;
- ID of E_A — ID ключа отправителя;
- Type — тип алгоритма хэширования;
- MD5 hash — хэш (MD5) от сообщения, зашифрованный закрытым ключом отправителя;
- Msghdr — заголовок сообщения;
- File name — имя файла (генерируется при отправке);
- Time — метка времени;
- Message — исходное нешифрованное сообщение отправителя.

PEM (X.509). Почтовая служба с повышенной конфиденциальностью — PEM — имеет статус интернет-стандарта (RFC 1421, 1424). Сообщения, пересылаемые с помощью PEM, сначала преобразуются в каноническую форму, в которой соблюдены соглашения относительно спецсимволов типа табуляции, последовательных пробелов

* Если не применять квантовый компьютинг.

и т. п. Затем сообщение обрабатывается алгоритмом MD5 или MD2, шифруется с помощью ключа DES (56-разрядный ключ) и передается с помощью кодировки base64. Передаваемый ключ защищается либо с помощью алгоритма RSA, либо с помощью ключа DES по схеме EDE (см. подразд. 4.1.3). Управление ключами в PEM предполагает использование центров их сертификации. Каждый сертификат указывает уникальный порядковый номер и хеш MD5. Подробно организация и работа PEM рассматривается в [20, 95].

В табл. 4.12 приведено сравнение почтовых систем PEM и PGM.

Таблица 4.12

Сравнение почтовых систем PGM и PEM

Признак	PGP	PEM
Поддержка шифрования	Да	Да
Поддержка аутентификации	Да	Да
Поддержка невозможности отказа от авторства	Да	Да
Поддержка сжатия	Да	Нет
Поддержка канонизации (приведение к стандартному имени)	Нет	Да
Поддержка списка рассылки	Нет	Да
Использование кодировки base64	Да	Да
Алгоритм шифрования текущих данных	IDEA	DES
Длина ключа для шифрования данных, бит	128	56
Алгоритм управления ключом	RSA/DSA	RSA или DES
Длина ключа для управления ключом, бит	384/512/1 024	Варьируется
Место имени пользователя	Определяется пользователем	X.400
Согласование с X.509	Нет	Да
Необходимость доверия кому-то третьему	Нет	Да (IPRA)
Отзыв ключа	Бессистемный	Лучший
Возможность перехвата сообщения	Нет	Нет
Возможность перехвата подписи	Нет	Да
Отношение к интернет-стандартам	Относится	Не относится
Разработчик	Небольшая команда	Комитет по стандартизации

4.2.4. Протокол передачи файлов — FTP

Протокол передачи файлов (File Transfer Protocol — FTP) — это один из первых и все еще широко используемых интернет-сервисов. Первые спецификации FTP относятся к 1971 г. С тех пор FTP претерпел множество модификаций [47, 72].

Протокол FTP предназначен для решения следующих задач:

- разделение доступа к файлам на удаленных абонентских машинах;
- прямое или косвенное использование ресурсов удаленных компьютеров;
- обеспечение независимости клиента от файловых систем удаленных абонентских машин;
- эффективная и надежная передача данных.

FTP — это протокол прикладного уровня, который, как правило, использует в качестве транспортного протокола TCP. Его нельзя использовать для передачи конфиденциальных данных, поскольку он не обеспечивает защиты передаваемой информации и передает между сервером и клиентом открытый текст. Сервер FTP может потребовать от клиента FTP аутентификации (т.е. при установлении соединения с сервером FTP-пользователь должен будет ввести свой идентификатор и пароль). Однако и пароль, и идентификатор пользователя будут переданы от клиента на сервер открытым текстом.

Простейшая модель работы протокола FTP представлена на рис. 4.31, где введены следующие обозначения:

- User Interface — пользовательский интерфейс работы с FTP;
- User PI (User Protocol Interpretator) — интерпретатор команд пользователя. Эта программа взаимодействует и с Server-PI, чтобы

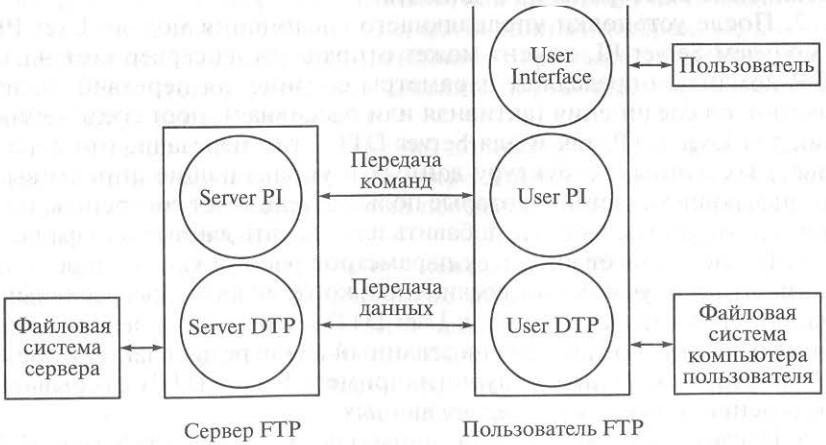


Рис. 4.31. Модель работы FTP

обмениваться командами управления передачей данных по каналу передачи команд, и с модулем User DTP, который осуществляет непосредственную передачу данных по каналу передачи данных;

• User DTP (User Data Transfer Process) — модуль, осуществляющий обмен данными между клиентом и сервером FTP по каналу передачи данных по командам от модуля User PI. Этот объект взаимодействует с файловой системой пользователя и объектом Server DTP;

• Server PI (Server Protocol Interpretator) — модуль управления обменом данных со стороны сервера по каналу передачи команд;

• Server DTP (Server Data Transfer Process) — модуль обмена данными со стороны сервера по каналу передачи данных;

• сервер FTP — собственно сервер FTP, который состоит из модуля Server PI управления передачей и модуля Server DTP, осуществляющего передачу;

• пользователь FTP — модуль клиента FTP, состоящий из модуля управления передачей User PI и модуля, осуществляющего передачу, User DTP.

FTP поддерживает сразу два канала соединения: канал передачи команд (и статусов их обработки) и канал передачи данных. Канал передачи данных может использоваться для передачи и в одном, и в другом направлениях, кроме того, он может закрываться и открываться по командам управляющих модулей в процессе работы. Канал передачи команд открывается с установлением соединения и используется только для передачи команд и получения ответов после их обработки.

Алгоритм работы FTP следующий:

1. Сервер FTP устанавливает в качестве управляющего соединение с портом 21 TCP, который всегда находится в состоянии ожидания соединения со стороны FTP-клиента.

2. После установки управляющего соединения модуля User PI с модулем Server PI, клиент может отправлять на сервер команды. FTP-команды определяют параметры соединения передачи: роли участников соединения (активная или пассивная), порт соединения (как для User DTP, так и для Server DTP), тип передачи, тип передаваемых данных, структуру данных и управляющие директивы, обозначающие действия, которые пользователь хочет совершить, например сохранить, считать, добавить или удалить данные или файл.

3. После согласования всех параметров работы канала передачи данных один из участников соединения, который является пассивным (например, клиентский модуль User DTP), переходит в режим ожидания открытия соединения на заданный для передачи данных порт. После этого активный модуль (например, Server DTP) открывает соединение и начинает передачу данных.

4. После окончания передачи данных соединение между Server DTP и User DTP закрывается, но управляющее соединение Server PI — User PI остается открытым. Пользователь, не закрывая сессии FTP,

может еще раз открыть канал передачи данных, передать необходимую информацию и т. д.

Протокол FTP можно использовать при передаче файлов не только между клиентом и сервером, но и между двумя FTP-серверами. Для этого пользователь сначала устанавливает управляющие соединения с двумя FTP-серверами, а затем устанавливает между ними канал передачи данных. В этом случае управляющая информация передается через модуль User PI, но данные транслируются через канал Server1 DTP — Server2 DTP.

Алгоритм работы FTP в этом случае имеет следующий вид:

1. Пользователь User PI указывает серверу Server1 PI работать в пассивном режиме, после чего сервер Server1 PI отправляет пользователю адрес User PI и номер порта N, который он будет слушать.

2. User PI назначает серверу Server2 PI быть активным участником соединения и указывает ему передавать данные на хост Server1 PI порта N.

3. User PI подает серверу Server1 PI команду «сохранить поступившие данные в таком-то файле», а серверу Server2 PI — «передать содержание такого-то файла».

4. Между серверами Server1 DTP и Server2 DTP образуется поток данных, управляемый клиентским хостом.

Основу передачи данных в протоколе FTP составляют механизм установления соединения между соответствующими портами и механизм выбора параметров передачи. Каждый участник FTP-соединения должен поддерживать 21 порт передачи данных по умолчанию. По умолчанию User DTP использует тот же порт, что и для передачи команд (обозначим его U), а Server DTP использует управляющий порт с номером L1. Однако, как правило, участниками соединения используются порты передачи данных, выбранные для них User PI, поскольку из всех управляющих процессов, участвующих в соединении, только он может изменять порты передачи данных как у User DTP, так и у Server DTP.

Пассивная сторона соединения должна до подачи команды начала передачи слушать свой порт передачи данных. Активная сторона, подающая команду на начало передачи, определяет направление перемещения данных.

После установки соединения между Server DTP и User DTP начинается передача. Одновременно по каналу Server PI — User PI передается уведомление о получении данных. Протокол FTP требует, чтобы управляющее соединение было открыто все время пока по каналу обмена данными идет передача. Сессия FTP считается закрытой только после закрытия управляющего соединения.

Как правило, сервер FTP ответственен за открытие и закрытие канала передачи данных. Сервер FTP должен самостоятельно закрывать канал передачи данных в следующих случаях:

• сервер закончил передачу данных в формате, требующем закрытия соединения;

• сервер получил от пользователя команду на прерывание соединения;

• пользователь изменил параметры порта передачи данных;

• было закрыто управляющее соединение;

• возникли ошибки, при которых невозможно возобновить передачу данных.

FTP-протокол имеет двух «братьев»: TFTP (Trivial FTP) и SFTP (SSH FTP).

TFTP — это простейший протокол передачи файлов, работающий поверх транспортного протокола UDP и обеспечивающий выполнение только самых элементарных операций передачи файлов: запись и считывание файлов. TFTP разрабатывался как облегченный вариант FTP [88], т.е. он не позволяет вызывать список каталога и не имеет никаких средств аутентификации, но может передавать 8-битовую информацию в соответствии со всеми интернет-стандартами.

SFTP (SSH File Transfer Protocol) — это FTP-протокол для передачи файлов через SSH-соединение. Также он предназначен для копирования и выполнения других операций с файлами поверх надежного и безопасного соединения по протоколу SSH. Существует заблуждение, что SFTP — это просто обычный FTP, работающий поверх SSH. В действительности SFTP — это новый протокол, разработанный специально для надежной передачи файлов, который иногда путают с Simple File Transfer Protocol. Повторим, что SFTP — это SSH File Transfer Protocol, а не Simple File Transfer Protocol.

SSH (от *англ.* secure shell — безопасная оболочка) — это протокол удаленного терминального доступа к узлам сети, который позволяет регистрироваться на удаленном узле сети, выполнять на нем команды, а также копировать и перемещать файлы между компьютерами. Протокол SSH организует защищенное безопасное соединение поверх небезопасных каналов связи. В состав типичной реализации протокола SSH входит также и примитивный клиент для удаленного копирования файлов через SSH-канал — SCP (Secure CoPy). В сравнении с довольно примитивным протоколом SCP протокол SFTP позволяет выполнять намного больше операций с файлами, например продолжать передачу файла после разрыва соединения или удалять файл на сервере. Для протокола SFTP имеются графические и псевдографические клиенты. При этом сам по себе протокол SFTP не обеспечивает безопасность работы, это делает нижерасположенный протокол SSH.

В настоящий момент протокол SFTP не принят в качестве официального стандарта в сети Интернет. Есть только предварительный вариант версии 6 этого протокола.

4.2.5. Всемирная паутина

Всемирная паутина (World Wide Web, или просто Web) основывается на использовании гипертекста. Идея создать сеть из документов, расположенных на разных машинах и связанных гиперссылками, была сформулирована Т. Бернерс-Ли в 1989 г. Создатель Web, работавший в лаборатории физики частиц института CERN в Женеве и ощутивший потребность распространения информации по физике высоких энергий для исследователей по всему миру, предложил распространить известную идею гиперссылок, связывающую документы в одной файловой системе, на всю компьютерную сеть. Это позволило бы его коллегам связывать ссылками документы, размещенные в компьютерной сети, в «единый» гипертекстовый документ.

Для реализации этой идеи были необходимы специальный протокол, умеющий работать с гипертекстовыми документами, средство описания документов и средство визуализации документа, собранного из отдельных документов, соединенных ссылками. Специальный протокол HTTP (Hyper Text Transmission Protocol) используется в Интернете с 1990 г. (HTTP регламентируется стандартом RFC 2616 [76]). Для описания документов и связывания их гиперссылками служит язык HTML (Hyper Text Markup Language), а для просмотра документов используются специальные программы — браузеры.

Всемирная паутина состоит из серверов, которые предоставляют доступ к хранящейся в них информации через графический интерфейс. Графический интерфейс реализуется через специальное программное обеспечение, работающее на абонентских машинах сети, — браузеры. Способность предоставлять информацию в виде видео, аудио, текста и изображений через стандартный набор элементов графического интерфейса, который не зависит от платформы, делает Web привлекательным ресурсом для всех категорий пользователей. Начали использовать Web в январе 1992 г. в Женеве (Швейцария), где исследователи могли получить данные с узла CERN. При этом Т. Бернерс-Ли предложил хранить документы на компьютерах, которые он назвал веб-серверами.

Язык разметки HTML

Важно помнить, что Web является всего лишь одним из приложений Интернета. Существует заблуждение, что Web и Интернет — одно и то же, а это далеко не так. Работу интернет-приложений обеспечивают протоколы Интернета, многие из которых мы уже рассмотрели.

Для представления документов на серверах WWW (или Web-серверах) и связывания их между собой используется специальный язык разметки — Hypertext Markup Language (HTML). Этот язык не относится к алгоритмическим языкам программирования, таким как C++ или Visual Basic. Файл на языке HTML, являющийся документом,

содержит набор данных и правил их отображения: какие использовать шрифты, цвета, какие данные заключить в таблицу, где поместить изображение, видео, какого цвета будет фон и т.д. За загрузку документов с Web-серверов, интерпретацию языка разметки и отображение информации пользователю отвечают браузеры.

Предтечей языка HTML является прикладной программный язык разметки, базирующийся на модели стандартного обобщенного языка разметки SGML (Standard Generalized Markup Language), описанного в стандарте ISO 8879. Язык SGML представляет собой систему, которая в течение многих лет применялась специалистами по документированию как средство разметки технических документов. В качестве языка форматирования в HTML используются декларации SGML и метод определения типа документа (Document Type Definition) — DTD.

Язык HTML решает задачи унифицированного представления документов в разных операционных системах на разных платформах и в разных сетях, обеспечивая их широкую доступность. Как уже говорилось, эта проблема была давно осознана, а для компьютерных сетей было предложено ее решение в рамках эталонной модели OSI ISO (см. гл. 1). В этой модели проблема решалась на уровне представления данных, для чего был разработан язык ASN (Abstract Syntax Notation) [19, 43], закрепленный в международном стандарте IS 8824. Этот язык является основой языка SMI, используемого для представления данных в базе MIB протокола SNMP (см. подразд. 4.2.2). Однако язык ASN, как и модель OSI ISO, не получил широкого распространения, чего нельзя сказать о языке HTML.

Вначале HTML был просто языком форматирования гипертекстовых документов. Элементы формата ограничивались заголовками, абзацами и небольшим набором форматов текста, таких как полужирный шрифт или курсив. Такие элементы формата называются тегами, представляющими собой ключевые слова, заключенные в угловые скобки. Например, чтобы выделить текст курсивом, необходимо заключить его в теги: текст.

Стандарт HTML 4.0 добавил к первоначальному языку ряд новых мощных функций и превратил ограниченный язык задания формата в полноценный инструмент структурирования. Полностью возможности HTML были осознаны, когда появилась возможность смотреть гипертекст не только в текстовом браузере, но и с помощью графического пользовательского интерфейса. Это случилось в 1993 г., когда появился на свет браузер Mosaic. В каждой версии спецификации HTML изменения все более углублялись: от статической информации до динамичного, развитого дизайна.

Язык HTML 4.0 был стандартизован консорциумом W3C (см. подразд. 1.2), что существенно ограничило деятельность компаний (Microsoft, Netscape) по расширению его спецификации своим синтаксисом. Деятельность консорциума W3C в настоящее время состоит в наблюдении за стандартизацией HTML, а также в наблюдении за

использованием различных протоколов и языков, связанных с Web, включая XML, CSS, HTTP, URL, FTP, NNTP и SGML [31].

Каскадные таблицы стилей CSS. Основной задачей четвертой версии стандарта HTML было отделение структуры и текста документа от его представления и стиля, т.е. бóльшая часть стандарта HTML 4.0 посвящена тому, как вынести из документа элементы предыдущих версий языка, используемых для представления или определения стиля информации, в отдельный блок. Таким образом появились стили и язык описания стилей CSS (Cascade Style Sheets).

Идея использования таблиц стилей CSS очень проста. Если раньше в HTML необходимо было прямо в документе указывать, как должен выглядеть тот или иной элемент, то при использовании описаний CSS такие указания выносятся в отдельный блок, который может загружаться в виде отдельного файла (как директива include в языке C/C++).

Очевидны следующие преимущества такого подхода. Во-первых, значительно облегчается изменение внешнего вида сайта и отдельных его элементов, т.е. достаточно изменить определение соответствующего стиля в единственном CSS-файле и эти изменения распространятся на весь сайт. Во-вторых уменьшается размер документов, что особенно заметно на «красивых» страницах, а это способствует их скорейшей загрузке на клиентские машины [21].

Набор технологий DHTML для создания интерактивных страниц. По мере развития WWW стало очевидно, что концепция о неизменности HTML-страницы после ее загрузки с сервера существенно ограничивает возможности представления и логику обработки информации. Например, эта концепция не поддерживала такой привычный интерфейс навигации по web-страницам, как «выпадающие» меню. Кроме того, вся логика по обработке информации реализовывалась на сервере, значит, для проверки корректности данных, введенных клиентом в формы, необходимо было отправить запрос на сервер, что существенно замедляло скорость работы пользователя в WWW.

Перечисленные недостатки HTML стали предпосылками появления концепции DHTML (Dynamic HTML), которая не была оформлена в виде стандарта и которую не следует рассматривать как новую спецификацию языка HTML. В концепции DHTML определяется набор технологий, позволяющих браузеру динамически изменять загруженные HTML-документы в ответ на пользовательские действия без взаимодействия с Web-сервером. Таким образом, часть логики работы с HTML-документом выносилась на сторону клиента, т.е. на абонентскую машину.

Основу концепции DHTML составляют следующие основные компоненты: язык HTML, язык описания каскадных таблиц стилей CSS, скриптовый язык, который интерпретируется браузером (например, JavaScript, vbscript), и объектная модель HTML-документов — DOM.

Язык HTML был расширен конструкциями специального типа — событиями. Так, с любой структурной единицей документа можно связать событие и обработчика этого события. Например, для некоторой таблицы можно указать, что по нажатию пользователем левой кнопки мыши над этой таблицей, она окрасится в цвет, код которого введен в специальное текстовое поле на этой же странице.

Обработчики таких событий реализуются с помощью специального кода, называемого скриптом, который и интерпретирует браузер. Самым распространенным языком реализации этих обработчиков является JavaScript. Однако остается вопрос: как программа на языке JavaScript сможет поменять структуру или представления загруженного HTML-документа? Ведь в этом случае браузер должен предоставить обработчику некоторый интерфейс для доступа к HTML-документу. Браузеров много, и вряд ли их производители договорятся о едином интерфейсе. Поэтому консорциум W3C разработал стандарт, т. е. объектную модель HTML-документа — DOM (Document Object Model), являющуюся четвертой и последней составляющей концепции DHTML, и интерфейс, который должен использоваться для работы с этой моделью [49].

Java-апплеты и технология Flash

Концепция вынесения логики работы с Web-сервера на сторону клиента не остановилась в своем развитии на концепции DHTML. Дело в том, что JavaScript и другие языки, интерпретируемые браузером, имеют ограниченные возможности по сравнению с интерпретируемыми языками общего назначения, например такими как Java.

Понимая это, компания Sun Microsystems разработала технологию Java-апплетов. Java-апплет — это прикладная программа на языке Java в форме байт-кода. В отличие от программ на языке JavaScript, которые интерпретируются браузером, Java-апплет выполняет виртуальная Java-машина (JVM), которая и является интерпретатором байт-кода и которую устанавливают на хосте отдельно. Java-апплеты были внедрены в первой версии языка Java в 1995 г.

Java является языком общего назначения, а следовательно, необходимо предпринимать специальные меры по обеспечению безопасного выполнения Java-апплетов на компьютерах пользователей, ведь загруженный из Интернета код вместе с HTML-страницей может обладать как полезными, так и вредными функциями. Для обеспечения безопасности пользователей апплеты выполняются в специальной песочнице (sandbox), которая ограничивает взаимодействие байт-кода Java с окружением: запрещает операции считывания и записи файлов, запуск других приложений на компьютере пользователя, а сетевой доступ из апплета допускает только к тому хосту, с которого он был загружен.

Апплету разрешено считывать значения параметров (цвета, шрифты, файлы с графическими изображениями и т. д.) с содержащей его Web-страницы и в соответствии с этими параметрами изменять свое поведение. Кроме того, параметры апплета можно изменять динамически [25].

Альтернативной технологией для вынесения логики на сторону клиента стали приложения Adobe Flash, которые в настоящее время являются наиболее популярным расширением функциональности языка HTML. Flash-приложения реализуются на языке ActionScript (последняя версия 3.0), являющемся объектно-ориентированным языком программирования и одним из диалектов стандарта ECMAScript, который добавляет интерактивность, обработку данных и многое другое в содержимое этих приложений. Язык ActionScript компилируется в байт-код и исполняется виртуальной машиной ActionScript Virtual Machine. В основе технологии Flash лежит векторный морфинг, т. е. плавное «перетекание» одного ключевого кадра в другой, что позволяет делать достаточно сложные мультипликационные сцены, задавая лишь несколько ключевых кадров для каждого персонажа.

Технология Flash в основном используется для написания игр, небольших полуинтерактивных анимаций и для красивого оформления рекламы, т. е. в сфере развлечений и дизайна. Для серьезных Web-приложений, где взаимодействие с пользователем должно быть без ущерба красоте, обычно используется язык JavaScript либо вообще не используются никакие технологии кроме тех, которые работают при любых настройках безопасности в браузерах (HTML, CSS).

HTTP-протокол

HTTP является текстовым протоколом, а это означает, что http-запросы можно послать web-серверу.

Общий вид HTTP-запроса следующий:

```
Запрос =  Метод SP URI-Запроса SP Версия-HTTP CRLF
          Заголовок-Запроса CRLF
          Заголовок-Запроса CRLF
          ...
          Заголовок-Запроса CRLF
          CRLF
          [Тело запроса]
```

Первая строка этого сообщения указывает метод, который должен быть применен к запрашиваемому ресурсу, идентификатор ресурса и используемую версию протокола.

В HTTP-запросе могут использоваться следующие методы:

```
Метод = "GET" | "HEAD" | "PUT" | "POST" | "DELETE" |
        "LINK" | "UNLINK" | дополнительный_метод
```

Метод GET служит для получения любой информации, идентифицированной URI-запросом. Если URI-запрос ссылается на процесс, выдающий данные, в качестве ответа будут выступать данные, сгенерированные указанным процессом (если они не являются выходными данными процесса), а не код самого процесса.

Если в заголовке запроса присутствовало поле «If-Modified-Since», то в ответ тело запрашиваемого ресурса передается только, если ресурс изменялся после даты, указанной в заголовке «If-Modified-Since», в соответствии со следующим алгоритмом:

- если код статуса ответа на запрос будет отличаться от «200 OK» или если дата, указанная в поле заголовка «If-Modified-Since», некорректна, ответ будет идентичен ответу на обычный запрос GET;
- если после указанной даты ресурс изменялся, ответ будет также идентичен ответу на обычный запрос GET;
- если ресурс не изменялся после указанной даты, сервер вернет код статуса «304 Not Modified».

Использование рассмотренного метода, называемого условным GET, направлено на разгрузку сети, так как он позволяет не передавать по сети избыточную информацию.

Согласно стандарту HTTP многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами), что позволяет кэшировать ответы на него.

Метод HEAD аналогичен методу GET за исключением того, что клиенту возвращается только заголовок сообщения-ответа (усеченный GET). Этот метод в основном используется для тестирования гиперссылок и проверки доступа к ресурсам.

Метод PUT служит для сохранения передаваемого на сервер ресурса с идентификатором URI (идентификатор URI будет рассмотрен далее).

Метод POST предназначен для передачи серверу информации, включенной в запрос как дополнительной к ресурсу, указанному в поле URI-запроса.

Метод POST был разработан как общий для осуществления следующих целей:

- аннотация существующих ресурсов;
- добавление сообщений в группы новостей, почтовые списки и другие подобные группы статей;
- доставка блоков данных процессам, обрабатывающим данные;
- расширение баз данных через операцию добавления.

В отличие от метода GET при многократном повторении одного и того же запроса с методом POST можно получать разные результа-

ты (например, после каждой отправки комментария в форум будет появляться новая копия этого комментария).

Метод DELETE используется для удаления ресурса, определенного идентификатором URI.

Как правило, методы PUT и DELETE в современных web-серверах запрещены, и управление данными осуществляется только через метод POST.

Web-серверы — обработчики HTTP-запросов

Как уже говорилось, протокол HTTP основан на парадигме запрос-ответ. Браузер устанавливает соединение с web-сервером и посылает ему запрос, содержащий метод запроса, URI и версию протокола, за которой следует сообщение в формате MIME, включающее в себя управляющую информацию запроса, информацию о клиенте и может быть тело сообщения.

В гл. 1 т. 1 данного учебника мы говорили о том, что есть задачи, которые необходимо решать на каждом уровне модели ISO OSI. Одной из таких задач является задача адресации. Для адресации на прикладном уровне большинство протоколов используют универсальные идентификаторы ресурса — URI (Universal Resource Identifier).

Самые известные примеры идентификатора URI — это URL и URN.

URL — это идентификатор URI, который помимо идентификации ресурса предоставляет еще и информацию о ее местонахождении. URL — это стандартизированный способ записи адреса ресурса в сети Интернет, который был также изобретен Т. Бернерсом-Ли в 1990 г. в стенах Европейского совета по ядерным исследованиям (Conseil Européen pour la Recherche Nucléaire — CERN) в Женеве и стал фундаментальной инновацией в Интернете. Изначально URL предназначался для обозначения мест расположения ресурсов (чаще всего файлов) во Всемирной паутине. Сейчас URL применяется для обозначения адресов почти всех ресурсов Интернета. Стандарт URL закреплен в документе RFC 1738 (предыдущая версия была определена в RFC 1630). Сейчас URL позиционируется как часть более общей системы идентификации ресурсов URI, а сама аббревиатура URL постепенно уступает место более широкому обозначению URI.

URN — это идентификатор URI, который идентифицирует ресурс в определенном пространстве имен (и соответственно в определенном контексте). Например, ISBN 0-395-36341-1 — это URI, указывающий на ресурс (книгу) 0-395-36341-1 в пространстве имен ISBN, но в отличие от идентификатора URL идентификатор URN не указывает на местонахождение этого ресурса. Впрочем, в последнее время появилась тенденция говорить просто URI о любой строке-идентификаторе без дальнейших уточнений, т.е. возможно, что обозначение URL и URN скоро уйдут в прошлое.

Формат URI описывается в стандарте RFC 3986. Идентификатор URI имеет гибкую расширяемую структуру, которая имеет следующий базовый вид:

```
<схема>:<идентификатор-в-зависимости-от-схемы>,
```

где схема — это протокол, используемый для обращения к ресурсу, например HTTP или FTP.

Наиболее типичный шаблон идентификатора ресурса в рамках конкретной схемы следующий:

```
<источник><путь><запрос><фрагмент>
```

Примерами URI могут служить следующие идентификаторы:

http://www.ics.uci.edu/pub/ietf/uri/#Related
ftp://ftp.is.co.za/rfc/rfc1808.txt
mailto:John.Doe@example.com
sip:911@pbx.mycompany.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/

Web-сервер, получив запрос, разделяет его на части и выделяет идентификатор URI запрашиваемого ресурса. Далее по пути, указанному в идентификаторе URI-запроса, в файловой системе сервера находится запрашиваемый объект. В случае если для запрашиваемого объекта не указана программа выполнения (например, файл является HTML-документом или объектом мультимедиа), web-сервер возвращает его в теле HTTP-ответа. Если для запрашиваемого объекта указана программа выполнения (например, файл является скриптом на языке perl, php, ruby или python) либо если объект сам является исполняемым файлом, выполняют следующие действия:

- инициируют выполнение объекта, указанного в URI. Входными данными для этого объекта являются параметры HTTP-запроса. Правила доступа к параметрам HTTP-запроса из кода исполняемого объекта определяются конкретной технологией сопряжения Web-сервера с Web-приложением;
- исполняемый объект динамически генерирует тело HTTP-ответа в зависимости от входных параметров;
- сгенерированный выполненным объектом HTTP-ответ возвращается клиенту и обрабатывается браузером стандартным образом;
- сервер отвечает сообщением, содержащим строку статуса (включая версию протокола и код статуса — успех или ошибка), за которой следует сообщение в формате MIME, включающее в себя информацию о сервере, метаинформацию о содержании ответа и само тело ответа.

По мере развития WWW стало понятно, что языки программирования общего назначения (C/C++, Pascal, Fortran, Modula, Ada) неудобны для реализации программ обработки HTTP-запросов и динамической генерации HTML-страниц. Постепенно программисты стали обращаться к специализированным языкам, предназначенным для обработки текстов (Perl — 1988 г., PHP — 1994 г.), которые впоследствии стали ассоциироваться только с WWW.

Большим шагом в развитии технологий генерации динамического контента был выпуск компанией Sun Microsystems платформы Java Enterprise Edition в 1999 г. Платформа J2EE — это не просто библиотека необходимых классов, она также включает в себя методологию разработки и внедрения Web-приложений от малого до корпоративного масштаба [7].

Ответом Microsoft на стремительно набирающую популярность платформу J2EE стало создание платформы Microsoft .NET, которая также включает в себя не просто большой набор классов объектов на каждый день, но и методологию разработки интерактивных web-систем. Одной из основных идей платформы Microsoft .NET является обеспечение совместимости различных служб, написанных на разных языках. Например, служба, написанная на языке C++ для платформы Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на языке Object Pascal с расширением .NET; на языке C# можно написать класс, наследуемый от класса, написанного на языке Visual Basic .NET, а исключение, созданное методом, написанным на языке C#, может быть перехвачено и обработано программой на языке Object Pascal с расширением .NET.

Так же как и в языке Java, среда разработки .NET создает байт-код, предназначенный для исполнения виртуальной машиной. Входной язык этой машины на платформе .NET называется MSIL (Microsoft Intermediate Language), или более поздний вариант CIL (Common Intermediate Language), или просто IL. Применение байт-кода позволяет достигать независимости от конкретной среды выполнения (платформы). Перед запуском сборки программной системы в среде исполнения (Common Language Runtime — CLR) ее байт-код преобразуется встроенным в среду JIT-компилятором в машинные коды целевого процессора.

Организация сеансов в протоколе HTTP

В соответствии со спецификацией протокола HTTP web-сервер не поддерживает постоянного соединения с клиентом, и каждый запрос обрабатывается как новый, т.е. без какой-либо связи с предыдущими, поэтому нельзя ни отследить запросы от одного и того же посетителя, ни сохранить для него переменные между просмотрами отдельных страниц. Для того чтобы различные запросы одного пользователя связать в единую логическую цепочку, исполь-

зуется механизм управления сессиями. При этом сама цепочка связанных запросов называется сессией. Информацию о сессии должен хранить браузер.

Для реализации механизма управления сессиями используются два дополнительных поля HTTP-заголовка: Set-Cookie и Cookie.

Схема взаимодействия клиента и сервера при использовании механизма Cookie следующая:

- клиент запрашивает какой-либо документ с сервера;
- сервер включает в заголовок ответа поле Set-Cookie, в котором сообщает клиенту информацию, предназначенную для сохранения, и параметры, задающие область действия этой информации;
- при каждом следующем запросе клиент возвращает серверу сохраненную информацию с помощью поля Cookie в заголовке запроса, если запрашиваемый ресурс попадает в область действия, заданную сервером;
- ресурс на сервере анализирует значение поля Cookie и идентифицирует клиента.

Первоначально механизм Cookie был описан в спецификации Netscape Communications Persistent Client State HTTP Cookie, в дальнейшем был принят документ RFC-2109 (HTTP State Management Mechanism).

Интерфейс CGI

Интерфейс CGI (Common Gateway Interface) — это один из первых стандартов сопряжения web-серверов и программ. Основная задача такого сопряжения — определить, как передать программе параметры HTTP-запроса и как передать обратно сформированный программой HTTP-ответ.

Интерфейс CGI реализует самый простой и очевидный способ: файл, указанный в HTTP-запросе (если он не является статическим ресурсом), запускается в виде отдельного процесса-обработчика. При этом:

- стандартный поток ввода процесса-обработчика ассоциируется с потоком вывода web-сервера;
- стандартный поток вывода процесса-обработчика ассоциируется с потоком ввода web-сервера;
- параметры HTTP-запроса передаются процессу-обработчику через аргументы командной строки и через переменные окружения.

Таким образом, разработчики программ для динамического формирования HTML-страниц при использовании интерфейса CGI никак не ограничиваются в выборе технологий разработки. С помощью интерфейса CGI может выполняться и бинарный файл, написанный на языке C/C++, Pascal или Аде, и скрипт, написанный на языке Perl, PHP или Python. Очевидным же недостатком такого сопряжения является необходимость подготовки и запуска нового про-

цесса на каждый HTTP-запрос, что является достаточно дорогой операцией во всех операционных системах.

Технология асинхронного взаимодействия с web-сервером — AJAX

С использованием программ для динамической генерации HTML-страниц на стороне web-сервера и интерактивного взаимодействия с пользователем на стороне клиента начался золотой век технологий WWW. Однако существовавшие технологии не позволяли решать любые задачи. Рассмотрим следующую ситуацию. У пользователя есть на некотором сайте фотоальбом в режиме доступа on-line. В соответствии с логикой функционирования при последовательной навигации по своему альбому пользователю необходимо перегружать только фотографии, а не всю страницу. Однако описанные ранее технологии (кроме технологий апплетов Flash и Java, которые являются слишком тяжеловесными для реализации такой простой функции) могут предложить только полную перезагрузку страницы вместе со всеми остальными элементами, на что будет тратиться время, трафик и, возможно, терпение пользователя.

Для решения этой проблемы в 2005 г. была предложена концепция асинхронного взаимодействия с web-сервером — AJAX (Asynchronous JavaScript and XML). Основой технологии AJAX является динамический язык HTML (DHTML), а главная ее задача состоит в уменьшении числа обращений к серверу. При обновлении информации AJAX-приложение не перегружает страницу полностью. Вместо этого код на языке JavaScript отправляет XML-запрос на сервер, а затем заменяет необходимую часть страницы, используя интерфейс модели DOM для обновления. Широко известным примером AJAX-приложения является Google™ Maps. Асинхронный обмен данными более удобен для работы, поскольку пользователю не приходится смотреть в пустой экран, дожидаясь перезагрузки страницы. Однажды загруженный интерфейс web-приложения уже не исчезнет с дисплея. В идеале пользователь вообще не должен замечать, когда приложение обратилось к серверу, так как данные подгружаются в фоновом режиме мелкими порциями.

Безопасность в HTTP: установление подлинности

В начале этой главы мы рассматривали вопросы обеспечения информационной безопасности безотносительно какого-либо сетевого протокола. Рассмотрим теперь методы обеспечения подлинности, конфиденциальности и целостности, применяемые в протоколе HTTP.

Протокол HTTP предоставляет простой механизм аутентификации пользователя ресурсов web-сервера, построенный на обмене инфор-

мацией между клиентом и сервером ресурса. Этот механизм позволяет выбрать схему аутентификации и уровень секретности передаваемых данных. Стандартом HTTP описываются две схемы аутентификации: Basic и Digest. Стоит подчеркнуть, что механизм аутентификации реализует web-сервер, а не web-приложение.

Схема Basic представляет собой простейший механизм аутентификации, в котором браузер пользователя делает запрос к защищенной странице, а web-сервер возвращает указание на необходимость аутентификации:

```
HTTP/1.1 401 Authorization Required
```

Браузер пользователя запрашивает имя пользователя и его пароль через специальное диалоговое окно. Затем он объединяет полученные параметры аутентификации и кодирует полученную строку с помощью алгоритма base64, после чего повторяет запрос к странице, передавая сформированную строку в специальном заголовке запроса. Недостатком этой формы аутентификации является передача по сети имени пользователя и его пароля в открытом виде.

Схема Digest предназначена для того, чтобы дать пользователю возможность доказать серверу, что он знает правильный пароль без передачи его по сети. Для этого в качестве параметра аутентификации серверу передается результат применения алгоритма MD5 (см. подразд. 4.1.5) к строке, состоящей из имени пользователя, его пароля, адреса запрашиваемой страницы и специального счетчика запросов, который хранится на сервере. При каждом новом запросе значение счетчика увеличивается на единицу. В результате данные об имени и пароле пользователя передаются по сети зашифрованными стойким шифром.

Следует отметить важность счетчика запросов в схеме Digest: при его отсутствии схема аутентификации была бы уязвимой для атак повторением. При работе по схеме Digest сервер удостоверяется, что значение счетчика запросов строго больше его предыдущего значения. Таким образом, включение счетчика запросов в хэшируемую по MD5 строку позволяет защититься от атак повторением, так как после каждого последующего запроса пользователя значение аутентификационной информации, переданной пользователем в прошлый раз, становится устаревшим и рассматривается сервером как некорректное.

Программист web-приложения может отказаться от стандартных схем аутентификации и реализовать свою схему. Самый распространенный на сегодняшний день вид аутентификации — это аутентификация через формы. В этом случае логин и пароль пользователя вводятся в специальные поля web-приложения и передаются как обычные параметры в запросах POST или GET, а параметры аутентификации обрабатываются не web-сервером, а самим web-приложением. Преимущество такой схемы — это возможность реализации

гибкой системы управления пользовательскими учетными записями и ролей пользователей.

Безопасность в HTTP: обеспечение конфиденциальности и целостности

Как уже говорилось, HTTP — текстовый протокол. Для обеспечения шифрования данных при взаимодействии двух систем по протоколу HTTP обычно используется протокол SSL, основанный на криптографии с открытым ключом. Протокол SSL обеспечивает три основных механизма защиты: взаимную аутентификацию, обеспечение конфиденциальности и целостности данных.

На этапе взаимной аутентификации сервер передает клиенту свой сертификат, а клиент может передавать серверу свой. Сертификат представляет собой документ в электронной форме, содержащий открытый ключ, принадлежащий его держателю, и дополнительную информацию о его владельце (например, ФИО владельца и название организации, где он работает, адрес электронной почты и т. п.), который подписан удостоверяющим центром (Certificate Authority). Основная задача сертификата — связать открытый ключ с личностью его владельца (владельца парного ему закрытого ключа). Сертификаты имеют срок действия, по окончании которого они становятся недействительными. Срок действия отражен в содержании сертификата.

Получив сертификат сервера, клиент проверяет его соответствие открытому ключу сервера, а сервер, в свою очередь, проверяет сертификат клиента, если он был передан. Использование сертификатов гарантирует, что клиент связан именно с требуемым ему сервером, а сервер точно знает, кто к нему подключился. В случае небольших групп пользователей механизм сертификатов может использоваться для аутентификации пользователей и авторизации их доступа к ресурсам сервера. Весь обмен сообщениями между SSL-сервером и SSL-клиентом шифруется с помощью ключа и алгоритма шифрования, которые согласовываются во время начального SSL-сеанса.

Перед установкой SSL-соединения протокол проверяет целостность данных (при шифровании она сохраняется автоматически). Для реализации службы целостности сообщений в протоколе SSL используется комбинация алгоритма шифрования с общим закрытым ключом и хэш-функцией.

Следует отметить, что протокол SSL не регламентирует алгоритмы шифрования, которые должны использоваться сторонами, т. е. он может быть интегрирован с различными алгоритмами шифрования, в том числе отличными от стандартного протокола SSL, где длина ключей ограничена экспортной политикой США в отношении вооружений. В настоящее время установлены следующие ограничения длины ключей, применяемых в экспортируемых из США программах

шифрования: для симметричного шифрования — 40 разрядов, а для несимметричного — 512.

Прокси-серверы

При организации доступа к WWW из локальной сети часто используется специальный класс программ — прокси-серверы (см. подразд. 2.5). В данном случае прокси — это программа, которая передает запросы клиентских программ (браузеров и других) в Интернет, получает ответы и передает их обратно. По своему назначению эта программа ближе к межсетевым экранам (см. подразд. 4.1.7). Необходимость наличия такой программы возникает обычно, если пользователь в силу тех или иных причин не может работать в Интернете напрямую, например в силу политики информационной безопасности компании. Тогда на определенном хосте в интранете, подключенном к Интернету, ставится прокси, а все остальные хосты в интранете компании настраиваются таким образом, чтобы работа велась через этот прокси.

HTTP-прокси предназначен для организации работы браузеров и других программ, использующих протокол HTTP. Браузер передает прокси-серверу запрос к URL-ресурсу, а прокси-сервер направляет его надлежащему web-серверу и получает от него ответ, который и передает затем запросившему браузеру.

HTTP-прокси может выполнять следующие функции:

- *кэширование полученных ресурсов*. Впоследствии, если запрашиваемый ресурс уже скачивался, не надо заново обращаться в Интернет. Тем самым увеличивается скорость доступа и снижаются затраты на трафик. По различным подсчетам кэширование позволяет сэкономить 10...15 % трафика в Интернет;
- *ограничение доступа к ресурсам Интернета*. Например, чтобы сотрудники на рабочих местах не играли в игры и не «болтали» в чатах, можно завести «черный список» сайтов, на которые прокси не будет пускать пользователей (или определенную часть пользователей, или в определенное время и т. д.). Ограничение доступа можно реализовывать по-разному, например, можно просто не выдавать ресурс, выдавая вместо него страницу «запрещено администратором» или «не найдено», а можно спрашивать пароль и авторизованных пользователей допускать к просмотру;
- *модификация запрашиваемых HTML-документов*. Например, можно вместо рекламных баннеров и счетчиков показывать пользователям прозрачные картинки, не нарушающие дизайн сайта, но существенно экономящие время и трафик за счет исключения загрузки картинок извне;
- *ограничение скорости работы с Интернетом*. Например, можно ограничивать скорость работы для отдельных пользователей, групп или ресурсов. Так, например, можно установить прави-

ло, чтобы файлы *.mp3 скачивались со скоростью не более 1 Кб/с, что предотвратит забивание интернет-канала мультимедиа трафиком;

• *биллинг*. Может вести журналы обращений для подсчета трафика за заданный период времени по заданному пользователю и выяснять популярность тех или иных ресурсов и т.д.;

• *маршрутизация запросов*. Существует возможность маршрутизировать запросы в WWW, например часть запросов направлять напрямую, а часть — через другие прокси (прокси провайдера, спутниковые прокси и т.д.), что тоже поможет эффективнее управлять стоимостью трафика и скоростью работы прокси в целом.

Веб-сервисы

Как уже отмечалось в т. 1 данного учебника, в течение последних нескольких лет WWW претерпевает качественные изменения. Если еще совсем недавно Всемирная паутина представляла собой, главным образом, совокупность серверов, содержащих статические документы со ссылками друг на друга, то современную WWW практически невозможно представить без интерактивных web-приложений, обрабатывающих различные запросы и помещающих результаты обработки этих запросов как в базы данных, так и на динамически генерируемые HTML-страницы.

Однако эволюция WWW не остановилась на web-приложениях. Взаимная интеграция бизнесов различных компаний, происходящая сейчас во всем мире, неизбежно влечет за собой появление технологий и стандартов для интеграции обслуживающих их приложений и корпоративных информационных систем. Появился сервис-ориентированный Web, основанный на двух относительно новых технологиях: SOAP (Simple Object Access Protocol) и XML (teXt Markup Language), которые уже упоминались (см. введение т. 1 данного учебника). Согласно концепции сервис-ориентированной архитектуры (SOA) информационная система предприятия состоит из набора серверов приложений, обменивающихся информацией в формате XML по протоколу SOAP.

Основой сервис-ориентированной архитектуры является web-сервис — набор логически связанных функций, которые могут быть программно вызваны через Интернет. Информация о том, какие функции предоставляет данный web-сервис, содержится в документе WSDL (Web Services Description Language), а для поиска существующих web-сервисов предполагается использовать специальные реестры, совместимые со спецификацией UDDI (Universal Description, Discovery and Integration).

Веб-сервисом принято называть ресурс, реализующий бизнес-функцию и обладающий следующими свойствами:

• является повторно используемым;

• определяется одним или несколькими явными интерфейсами, независимыми от технологии (достигается за счет XML);

• может быть вызван посредством протоколов взаимодействия ресурсов между собой.

По сути web-сервисы представляют собой новый вид web-приложений для связи разнородных приложений на основе использования общих стандартов. Благодаря web-сервисам функции любой прикладной программы становятся доступными через Интернет.

Все web-сервисы реализуются на следующих общих принципах:

• создатель конкретного web-сервиса определяет формат запросов к нему и формат ответов на данные запросы;

• с любого хоста в Интернете можно сделать запрос к данному web-сервису;

• web-сервис выполняет заданную последовательность действий и отправляет обратно результат.

Все web-сервисы базируются на открытых стандартах и протоколах, ключевыми из которых являются:

• SOAP — протокол доступа к простым объектам, т.е. механизм для передачи информации между удаленными объектами на базе протокола HTTP и некоторых других протоколов прикладного уровня;

• WSDL — язык описания web-сервисов;

• UDDI — протокол поиска ресурсов в Интернете.

Изначально протокол SOAP был предложен фирмой Microsoft в качестве средства для удаленного вызова процедур (RPC) по протоколу HTTP, а спецификация SOAP 1.0 была тесно связана с технологией COM. Фирма IBM и ряд других компаний внесли определенный вклад в развитие этого протокола. Комитет W3C подготовил проект стандарта.

Спецификация SOAP [87] определяет XML-конверт для передачи сообщений, метод для кодирования программных структур данных в формате XML, а также средства связи по протоколу HTTP. Сообщения SOAP могут быть двух типов: запрос (Request) и ответ (Response). Запрос вызывает метод удаленного объекта, а ответ возвращает результат выполнения данного метода.

Для того чтобы приложения могли использовать web-сервисы, программные интерфейсы последних должны быть детально описаны. Описание может включать в себя следующую информацию: протокол, адрес сервера, номер используемого порта, список доступных операций, формат запроса и ответа и т.п.

Для описания этой информации было предложено несколько языков. Одним из них был язык SDL (Service Description Language), разработанный Microsoft и входивший в первую версию Microsoft SOAP Toolkit. Компания IBM переработала спецификацию и, использовав спецификацию Network Accessible Service Specification Language (NASSL), выпустила версию NASSL Toolkit как часть SOAP4J. Идеи, реализованные в NASSL, повлияли на спецификацию

языка SOAP Contract Language — SCL, предложенную Microsoft. В настоящее время обе спецификации (NASSL и SDL/SCL), а также предложения других фирм учтены в спецификации языка WSDL. Для описания бизнес-логики компании IBM и Microsoft работают над спецификацией языка Web Services Flow Language — WSFL.

Назначение протокола UDDI — предоставить механизм для обнаружения web-сервисов на основе реестра, в котором провайдеры web-сервисов могут регистрировать сервисы, а разработчики — искать необходимые им сервисы. Компании IBM, Microsoft и Ariba создали собственные UDDI-реестры (в скором времени эти реестры будут объединены в Web-реестр), в одном из которых разработчики могут зарегистрировать свои web-сервисы, после чего данные будут автоматически реплицированы в другие реестры.

Протокол UDDI использует четыре элемента: Business Entity, Business Service, Binding Template и Technology Model.

Элемент Business Entity описывает отрасль промышленности, индустрию, для которой предназначен данный web-сервис. Этот элемент может включать в себя описания категорий для данной индустрии, облегчающие более детальный поиск сервисов.

Business Service — это класс сервисов в рамках определенной отрасли промышленности или услуг, т. е. каждая отрасль принадлежит определенному элементу Business Entity.

Элементы Binding Template и Technology Model вместе определяют web-сервис: Technology Model содержит абстрактное описание, а Binding Template — конкретную спецификацию сервиса. Каждый элемент Binding Template принадлежит определенному элементу Business Service, но несколько элементов Binding Template могут ссылаться на один элемент Technology Model.

Бизнес-реестр UDDI сам является SOA web-сервиса, поддерживает операции создания, модификации, удаления и поиска всех четырех рассмотренных элементов [13].

СПИСОК ЛИТЕРАТУРЫ

1. Алгоритм MARS. [Electronic resource]. Mode access : http://domino.research.ibm.com/comm/research_projects.nsf/pages/security.mars.html

2. Алгоритм RC6. [Electronic resource]. Mode access : http://www.rsa.com/rsalabs/node.asp?id=2512

3. Алгоритм Rijndael. — [Electronic resource]. — Mode access : http://www.iaik.tu-graz.ac.at/research/krypto/AES/old/%7Erijmen/rijndael/

4. Алгоритм Serpent. — [Electronic resource]. — Mode access : http://www.cl.cam.ac.uk/~rja14/serpent.html

5. Алгоритм Twofish. — [Electronic resource]. — Mode access : http://www.schneier.com/twofish.html

6. *Алексеев И.* Введение в архитектуру MPLS //Открытые системы / И. Алексеев. — [Электронный ресурс]. — Режим доступа : http://www.osp.ru/nets/1999/12/144399/

7. *Алур Д.* Образцы J2EE. Лучшие решения и стратегии проектирования / Д. Алур, Д. Крупи, Д. Малкс. — М. : Лори, 2004. — 400 с.

8. *Брежнев А. Ф.* Семейство протоколов TCP/IP / А. Ф. Брежнев, Р. Л. Смелянский. — [Электронный ресурс]. Режим доступа : http://www.mark-itt.ru/FWO/tcpip/

9. Вредоносное программное обеспечение // Наука и образование. — [Электронный ресурс]. — Режим доступа : http://articles.excelion.ru/science/info/38492159.html

10. *Коротыгин С.* .IPSec-протокол защиты сетевого трафика на IP-уровне / С. Коротыгин. — [Электронный ресурс]. Режим доступа : http://www.ixbt.com/comm/ipsecure.shtml

11. Маршрутизация OSI. — [Электронный ресурс]. — Режим доступа : http://www.citforum.ru/nets/ito/28.shtml

12. *Ньюкомер Э.* Веб-сервисы: XML, WSDL, SOAP и UDDI / Э. Ньюкомер. — Спб. : Питер, 2003. — 256 с.

13. *О'Рейли* Т. Что такое Web 2.0 / Т. О'Рейли // Компьютерра. — 2005. — № 37, 38.

14. *Семенов Ю. А.* IP-протокол. / Ю. А. Семенов. — [Электронный ресурс]. — Режим доступа : http://book.itep.ru/4/44/ip_441.htm

15. Сети: NetWare. CITforum [Электронный ресурс]. — Режим доступа : http://www.citforum.ru/nets/netware.shtml

16. Смелянский Р. Л. Современные некоммерческие средства обнаружения атак / Р. Л. Смелянский, Д. Ю. Гамаюнов // Программные системы и инструменты. Тематический сборник факультета ВМК МГУ им. М. В. Ломоносова № 3. — М. : Издательский отдел факультета ВМК МГУ. — 2002.

17. *Сухомлин В.А.* Введение в анализ информационных технологий. / В.А. Сухомлин. — М. : Горячая линия — Телеком. — 2003. — 457 с.

18. *Сухомлин В.А.* Промышленные консорциумы и профессиональные организации. Курс лекций / В.А. Сухомлин. — [Электронный ресурс]. — Режим доступа : http://sukhomlin.oit.cmc.msu.ru/AnalizeIT/Ch2_5.html

19. *Тенненбаум Э.С.* Компьютерные сети / Э.С. Тенненбаум. — Спб. : Питер, 2007. — 992 с.

20. *Шмитт К.* CSS. Рецепты программирования / К. Шмитт. — Спб. : БХВ, 2007. — 592 с.

21. *Amoroso E.G.* Intrusion Detection / E.G. Amoroso NJ : Intrusion.Net Books, 1999. — 218 p.

22. *Anderson J.P.* Computer Security Threat Monitoring and Surveillance / J.P. Anderson. — Fort Washington : PA, 1980.

23. *Bellman R.* Wikipedia / R. Bellman. — [Electronic resource]. — Mode access : http://en.wikipedia.org/wiki/Richard_Bellman

24. *Boese E.S.* Java Applets / E.S. Boese. — B&W, 2008. — 324 p.

25. *Borenstein N.* MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies / N. Borenstein, N. Freed. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1341

26. *Borenstein N.* MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies / N. Borenstein, N. Freed. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1521

27. *Braden R.* Resourse Reservation Protocol (RSVP) / R. Braden [и др.] — Version 1 Fanctional Specification. RFC 2205, 1997.

28. *Carroll J.* Routing TCP/IP / J. Carroll. — Cisco Press, 2007. V. 1. — 936 p.

29. *Case J.* Simple Network Management Protocol (SNMP) / J. Case, M. Fedor, M. Schoffstall. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1157

30. *Castro E.* HTML 4 for the World Wide Web / E. Castro. — Peachpit press, 1999.

31. *Clark D.* The Design Philosophy of the DARPA Internet Protocols / D. Clark // Computer Communications Review. — 1988. — V. 18. — №4. — P. 106—114.

32. *Clark D.D.* Window and Acknowledgement Strategy in TCP/ D.D. Clark. — RFC 813, 1982.

33. *Clark D.D.* An Analysis of TCP Processing Overhead / D.D. Clark, V. Jacobson, J. Romkey, H. Salwen // IEEE Comm. Magazin. — 1989. — V. 27. — P. 23—29.

34. *Crispin M.* Internet Message Access Protocol — Version 4rev1. / M. Crispin. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc3501

35. *Crocker D.H.* Standart for the format of ARPA Internet text messages / D.H. Crocker. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc822

36. *De Prycker M.* Asynchronouse Transfer Mode / M. De Prycker. — N.Y.: Prentice Hall, 1995. — 380 p.

37. Denning D. An Intrusion-Detection Model / D. Denning // IEEE Transactions on Software Engineering. — 1987. — V. SE-13. — №2.

38. DHCP RFC — Dynamic Host Configuration Protocol RFC's (IETF). — [Electronic resource]. — Mode access : http://www.bind9.net/rfc-dhcp

39. Diffie W. Exhaustive Cryptoanalysis of the NBS Data Encription Standard / W. Diffie, M. E. Hellman // IEEE Computer Magazine. — 1977. — V. 10. — P. 74—84.

40. *Diffie W.* New directions in Cryptography / W. Diffie, M. E. Hellman // IEEE Transactions on Software Engineering. — 1976. — P. 644—654.

41. DNS for rocket scientists. — [Electronic resource]. — Mode access : http://www.zytrax.com/books/dns/

42. *Droms R.* Dynamic Host Configuration Protocol. / R. Droms. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc2131

43. *Dubuisson O.* ASN.1 — Communication between heterogeneous systems / O. Dubuisson // Morgan Kaufmann Publishers. — 2000. — P. 588.

44. *Fielding R.* Hypertext Transfer Protocol — HTTP 1.1. / R. Fielding, J. Gettys, J. Mogul и др. — [Electronic resource]. Mode access : http://tools.ietf.org/html/rfc2068

45. *Ford L. R.* Flows in Networks / L. R. Ford, D. R. Fulkerson. — NJ : Princeton University Press, 1962.

46. *Friedl S.* An Illustrated Guide to IPSec / S. Friedl. — [Electronic resource]. — Mode access : http://www.unixwiz.net/techtips/iguide-ipsec.html

47. FTP Reviewed. [Electronic resource]. — Mode access : http://pintday.org/whitepapers/ftp-review.shtml

48. *Goodman D.* JavaScript & DHTML Cookbook / D. Goodman. — O'Reilly, 2003. — 540 p.

49. *Graziani R.* Routing Protocols and Concepts / R. Graziani, A. Johnson. — Cisco Press, 2007. — 640 p.

50. *Hamzeh K.*, Point-to-Point Tunneling Protocol (PPTP) / K. Hamzeh [и др.]. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc2637

51. *Hellman M. E.* A Cryptoanalytic Time-Memory Tradeoff / M. E. Hellman // IEEE Trans on Information Theory. — 1980. — V. IT-26. — №6. — P. 401—406.

52. History of DNS. — [Electronic resource]. — Mode access : http://www.lagunainternet.com/techsupport/history_of_dns.htm

53. IEEE 802 LAN/MAN Standards Committee. — [Electronic resource]. — Mode access : http://www.ieee802.org/

54. *Jacobson V.* Congestion Avoidance and Control / V. Jacobson // Proc. SIGCOMM'88 Conf. ACM. — 1988. — P. 314—329.

55. *Kantor B.* BSD Rlogin. / B. Kantor. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1282

56. *Kantor B.* Network News Transfer Protocol / B. Kantor, P. Lapsley. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc977

57. *Karn P.* MACA — A New Channel Access Protocol for Packet Radio / P. Karn // ARRL/CRRL Ninth Computer Networking Conf. — 1990. — P. 134—140.

58. *Kleinrock L.* Hierarchical Routing for Large Networks, Performance Evaluation and Optimization / L. Kleinrock, F. Kamoun // Computer Networks. — 1977. — V. 1. — № 3. — P. 155—174.

59. *Kurose J. F.* Computer Networking: Top-Down Approach / J. F. Kurose, K. W. Ross. — Pearson Education, 2008. — 852 p.

60. *Lai X.* A Proposal for a New Block Encrypcion Standard. Proc. of Eurocrypt 90 / X. Lai, J. Massey. — NY: Springer-Verlag, 1990. — P. 389—404.

61. *Landwehr C. E.* A security model for military message systems / C. E. Landwehr. — [Electronic resource]. — Mode access : http://www.cs.purdue.edu/homes/ninghui/courses/Fall03/papers/landwehr_etal_84.pdf

62. *Merkle R. C.* On the Security of Multiple Ecription / R. C. Merkle, M. E. Hellman // Communications of the ACM. — 1981. — V. 24. — №6. — P. 465—467.

63. *Myers J.* Post Office Protocol — Version 3 / J. Myers, M. Rose. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1939

64. *Nagle J.* On Packet Switches with Infinite Storage / J. Nagle // IEEE Transactions on Commun. — 1987. — V.35. — P. 435—438.

65. Needham R. M. Using Encryption for Authentification in Large Networks of Computers / R. M. Needham, M. D. Schroeder // CACM. — 1978. — V.21. — №12. — P. 993—999.

66. NetBIOS. Wikipedia. — [Electronic resource]. — Mode access : http://ru.wikipedia.org/wiki/NetBIOS

67. *Neuman B. C.*: An Authentication Service for Computer Networks / B. C. Neuman, T. Kerberos // IEEE Commun. Magazine. — 1994. — V. 32. — №9. — P. 33—38.

68. NIST. Архив по конкурсу FIPS-197. — [Electronic resource]. — Mode access : http://csrc.nist.gov/archive/aes/index.html

69. *Otway C. G.* Efficient and Timely Mutual Authentication / C. G. Otway, O. Rees // Operating Systems Rev. — 1987. — №1. — P. 8—10.

70. Paul Baran and the Origins of the Internet. — [Electronic resource]. — Mode access : http://www.rand.org/about/history/baran.html

71. *Perlman R.* Interconnections : Bridges, Routers, Switches and Internetworking Protocols / R. Perlman. — Addison-Wesley, 1999. — 537 p.

72. *Postel J.* File Transfer Protocol (FTP) / J. Postel, J. Reynolds. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc959

73. *Postel J.* Telnet protocol specification / J. Postel, J. Reynolds. — [Electronic resource]. Mode access : http://tools.ietf.org/html/rfc854

74. *Postel J. B.* Simple Mail Transfer Protocol / J. B. Postel. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc821

75. Procmail. — [Electronic resource]. — Mode access : http://www.procmail.org/

76. RFC 2616. Hypertext Transfer Protocol — HTTP/1. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc2616

77. *Rivest R. L.* The MD5 Message-Digest Algorithm. RFC1320. 1992.

78. *Rivest R. L.* A Method for Obtaining Digital Signatures and Public Key Cryptosystems / R. L. Rivest, A. Shamir // Communications of the ACM. — 1978. — V. 21. — №2. — P. 120—126.

79. *Rose M.* Structure and Identification of Management Information for TCP/IP-based Internets. / M. Rose, K. McCloghrie. — [Electronic resource]. Mode access : http://tools.ietf.org/html/rfc1155

80. *Rose M. T.* Simple Book: An Introduction to Management of TCP/I-based Internet / M. T. Rose. — NJ: Prentice Hall, 1994.

81. *Rose M. T.* How to use your network using SNMP / M. T. Rose, K. McCloghrie. — NJ: Prentice Hall, 1995.

82. RSS 2.0 Specification. — [Electronic resource]. — Mode access : http://www.rssboard.org/rss-specification

83. *Schneier B.* Applied Cryptography / B. Schneier. — NY : J. Wiley, 1995.

84. Secure Hash Algorithm U.S. Government Information Processing Standards 180, 1993.

85. *Singh S.* The CODE BOOK / S. Singh. — London : 4th Estate, 2000. — 402 p.

86. SNMP handbook. — [Electronic resource]. — Mode access : http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNMP.html

87. SOAP версии 1.2. Рекомендация W3C от 27 апреля 2007 года. — [Electronic resource]. — Mode access : http://www.w3.org/TR/soap12-part1/

88. *Sollins K.* The TFTP Protocol / K. Sollins. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc1350

89. SRI International. — [Electronic resource]. — Mode access : http://www.sri.com/

90. *Tanase M.* IP Spoofing: An Introduction / M. Tanase. — [Electronic resource]. — Mode access : http://www.securityfocus.com/infocus/1674

91. *Tipton F.* Information Security Management Handbook / F. Tipton, M. Krause. — Auerbach, 2000. — 711 p.

92. *Tomlinson R. S.* Selecting Sequence Numbers / R. S. Tomlinson // ACM Operating Systems Review. — 1975. — V. 9. — №3. — P. 11—23.

93. *Venaas S.* Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6) / S. Venaas, T. Chown, B. Volz. — [Electronic resource]. — Mode access : http://tools.ietf.org/html/rfc4242

94. *Watson R. W.* Timer-Based Mechanisms in Reliable Transport Protocol Connection Management / R. W. Watson // Computer Networks. — 1981. — V. 5. — P. 47—56.

95. X.509: Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks. — [Electronic resource]. — Mode access : http://www.itu.int/rec/T-REC-X.509/en

96. X11 Project. — [Electronic resource]. — Mode access : http://www.x.org/wiki/

97. *Zimmermann P.* PGP source code and internals / P. Zimmermann. — Cambridge, MASS. : MIT Press, 1995.

ОГЛАВЛЕНИЕ

Учебное издание

Смелянский Руслан Леонидович

КОМПЬЮТЕРНЫЕ СЕТИ

Том 2

Сети ЭВМ

Учебник

Редактор *В. Н. Махова*. Технический редактор *Е. Ф. Коржуева*
Компьютерная верстка: *Л. А. Смирнова*. Корректоры *Н. Л. Котелина, Т. В. Кузьмина*

Изд. № 101115442. Подписано в печать 27.10.2010. Формат 60×90/16.
Гарнитура «Ньютон». Бумага офсетная. Печать офсетная. Усл. печ. л. 15,0.
Тираж 2 000 экз. Заказ № 30833.

Издательский центр «Академия». www.academia-moscow.ru
125252, Москва, ул. Зорге, д. 15, корп. 1, пом. 26б.
Адрес для корреспонденции: 129085, Москва, пр-т Мира, 101В, стр. 1, а/я 48.
Тел./факс: (495) 648-0507, 616-0029.
Санитарно-эпидемиологическое заключение № 77.99.60.953.Д.007831.07.09 от 06.07.2009.

Отпечатано в соответствии с качеством предоставленных издательством электронных носителей в ОАО «Саратовский полиграфкомбинат».
410004, г. Саратов, ул. Чернышевского, 59. www.sarpk.ru