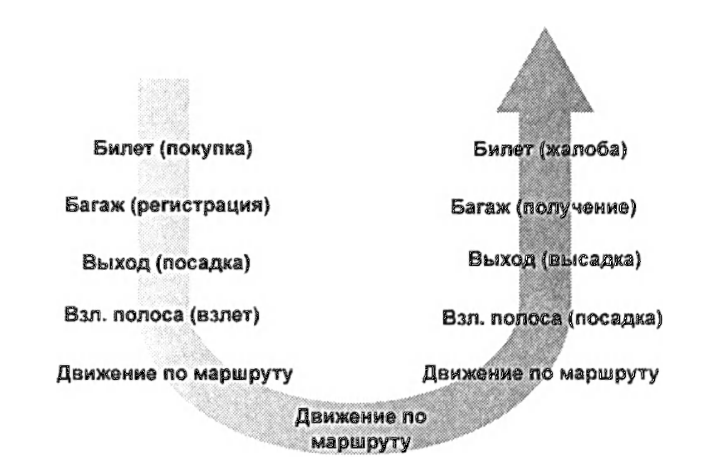
**Уровни протоколов и модели их обслуживания**

Из нашего с вами обсуждения можно сделать вывод, что Интернет — *очень* сложная система. Она состоит из многих компонентов: многочис­ленных приложений и протоколов, различных типов конечных систем, коммутаторов пакетов, а также различных сред передачи. Учитывая та­кую необычайную сложность, возникает вопрос, возможно ли как-либо организовать сетевую архитектуру или, по крайней мере, организовать обсуждение сетевой архитектуры. К счастью, на эти оба вопроса мы мо­жем ответить «да».

**Многоуровневая архитектура**

Перед тем как попытаться систематизировать наши знания архитек­туры Интернета, давайте обратимся к аналогии из мира людей. В нашей повседневной жизни мы все имеем дело со сложными системами. Пред­ставьте себе, что кто-нибудь попросил вас описать, например, как орга­низована система воздушных сообщений. Как бы вы стали описывать такую сложную структуру, которая включает в себя агентство по прода­же авиабилетов, службу по проверке багажа, сотрудников терминалов, летный персонал, парк воздушных судов, а также международные дис­петчерские службы? Один из возможных способов состоит в том, чтобы показать набор действий, которые вы производите (или которые про­изводятся для вас), когда вы совершаете авиаперелет. Такие действия включают в себя, например, покупку билета, регистрацию багажа, про­ход через выход терминала и посадку на борт самолета. После этого са­молет взлетает и движется по маршруту до места назначения. Затем вы приземляетесь, покидаете самолет через выход терминала и получаете свой багаж. Если вы остались недовольны полетом, то подаете жалобу авиаперевозчику. Сценарий такого авиаперелета показан на рис. 1.21.

Тут мы сразу замечаем аналогию с функционированием компьютер­ных сетей: авиакомпания доставляет вас от исходного пункта до пункта назначения. Пакет в Интернете тоже доставляется от хоста-источника до хоста-приемника. Но этим сходство не ограничивается. Здесь про­сматривается некоторая *структура.* Мы замечаем, что на обоих концах данной структуры находятся функции, связанные с билетами; с бага­жом для пассажиров, которые уже получили билеты, а также со стойкой регистрации и выходом для тех пассажиров, которые уже приобрели би­лет и прошли регистрацию багажа. Для пассажиров, которые уже прош­ли регистрацию, то есть купили билеты, сдали багаж и миновали ворота, есть функция, связанная со взлетом и посадкой, а также присутствуют функции в течение полета, относящиеся к движению лайнера по марш­руту. Можно взглянуть на всю эту функциональность, представленную на рис. 1.21, и изобразить ее в виде *горизонтальных* уровней.

****

**Рис. 1.21. Действия при воздушном путешествии**

На рис. 1.22 функциональность, предоставляемая авиакомпанией, разделена на уровни, и эту структуру мы будем обсуждать, рассматри­вая воздушные путешествия. Обратим внимание, что каждый уровень в комбинации с находящимися под ним, предоставляет некоторую функ­циональность или некоторые *службы.* Например, на билетном уровне и ниже пассажир проходит путь от кассира авиакомпании до кассира авиакомпании (от покупки билета до возможного получения компенса­ции по жалобе). На багажном уровне и ниже производится обслужива­ние пассажира от регистрации до получения багажа. Обратим внимание, что багажный уровень обслуживает только тех пассажиров, которые уже имеют билеты. На уровне выхода выполняется передача пассажира и багажа от стойки регистрации вылета до стойки регистрации прибы­тия. На уровне взлет-посадка производится обслуживание пассажиров и их багажа от взлетной полосы до взлетной полосы. Следовательно, на каждом уровне, во-первых, выполняются определенные действия, отно­сящиеся к этому уровню (например, на уровне выхода производятся посадка на самолет и высадка с него) и, во-вторых, используются службы, предоставляемые уровнями ниже (например, уровень выхода включает также передачу пассажиров от взлетной полосы до взлетной полосы, ко­торую предоставляет уровень взлет/посадка).

****

**Рис. 1.22. Функциональность авиакомпании в виде горизонтальных уровней**

Такая многоуровневая архитектура позволяет нам обсуждать более детально отдельные части большой и сложной системы. Это упрощение само по себе очень ценно, так как дает возможность модульной органи­зации, позволяя намного проще изменять реализацию функций и служб, предоставляемых уровнем. Так как один уровень предлагает одни и те же услуги и функции уровню над ним и использует те же самые услуги и функции с уровня ниже, то изменение реализации одного уровня не затронет всю остальную систему. Заметим, что изменение реализации предоставляемых услуг сильно отличается от изменения самой услуги, то есть, например, если функции выхода поменялись (например, посад­ка пассажиров производится по росту), то вся остальная система воз­душного сообщения не претерпит изменений, так как данный уровень выхода все так же выполняет свои функции (посадка и высадка пасса­жиров); он просто реализует эту функцию немного по-другому после внесения изменений. Для больших и сложных систем, которые постоян­но обновляются, способность изменять реализацию предоставляемых услуг (сервисов), не затрагивая другие компоненты системы, является еще одним важным преимуществом многоуровневой структуры.

***Уровни протоколов***

Но достаточно об авиаперелетах. Давайте обратим наше внимание на сетевые протоколы. Структура сетевых протоколов — аппаратное и программное обеспечение, реализующее эти протоколы — органи­зована с помощью **уровней.** Каждый протокол принадлежит какому-то определенному одному уровню, точно так же, как каждая функция в архитектуре воздушной линии на рис. 1.22. Мы будем рассматривать **услуги,** которые уровень предоставляет уровню выше — так называе­мую **модель обслуживания** уровня. Как в случае с нашим примером авиалинии, каждый уровень предоставляет свои услуги, во-первых, вы­полняя определенные действия внутри себя, и, во-вторых, используя услуги уровня, находящегося ниже. Например, услуги, уровня *п,* могут включать надежную доставку сообщений с одного конца сети на другой. Это может быть реализовано с помощью ненадежной сквозной доставки сообщения на уровне *п-*1 с добавлением функциональности уровня *п* для обнаружения и передачи потерянных сообщений.

Уровни протоколов могут быть реализованы в программном, в ап­паратном обеспечении либо в их комбинации. Протоколы прикладного уровня — такие, как HTTP и SMTP — почти всегда реализованы в про­граммном обеспечении конечных систем; то же самое можно сказать о протоколах транспортного уровня. Поскольку физический и каналь­ный уровень отвечают за коммуникации по линиям связи, они обычно реализованы в сетевых интерфейсных картах, например Ethernet или Wi-Fi, соединенных с линией связи. Сетевой уровень обычно исполь­зует как аппаратную, так и программную реализацию. Обратим также внимание, что, как и функции в многоуровневой архитектуре воздуш­ных сообщений перераспределены между различными аэропортами и диспетчерскими службами, образующими систему, так и протокол уровня *п распределен* между конечными системами, коммутаторами и другими компонентами, образующими компьютерную сеть. Таким об­разом, в каждой компьютерной сети существуют компоненты протоко­лов уровня *п.*

Разделение протоколов на уровни имеет свои преимущества. Как мы уже видели, такой подход позволяет разложить структуру на ком­поненты. Принцип модульности облегчает обновление и модернизацию составляющих частей системы. Однако следует сказать, что некоторые специалисты сетевых коммуникаций выступают против уровней. Одним из потенциальных недостатков такой структуры является то, что один уровень может дублировать функции других, например, нижележащих. Допустим, обработка ошибок во многих стеках протоколов может выполняться на нескольких уровнях на сквозной основе. Еще одним недостатком является то, что на одном уровне может потребоваться ин­формация (например, значение времени), которая представлена только на другом, а это нарушает принцип изолированности.

****

**Рис. 1.23. Стек протоколов Интернета (а) и эталонная модель OSI (б)**

Набор протоколов различных уровней называется **стеком прото­колов.** Он состоит из пяти уровней: физического, канального, сетевого, транспортного и прикладного, как показано на рис. 1.23а.

***Прикладной уровень***

Прикладной уровень (иначе называемый уровнем приложений) поддерживает сетевые приложения и их протоколы. Прикладной уро­вень Интернета включает множество протоколов, таких как HTTP (обе­спечивающий запрос и передачу веб-документов), SMTP (отвечающий за сообщения электронной почты) и FTP (для обмена между двумя ко­нечными системами).

Мы с вами увидим, что определенные сетевые функции, такие, как трансляция понятных человеку имен конечных систем в Интернете, на­пример [**www.ietf.org**](http://www.ietf.org)в 32-разрядные сетевые адреса также выполняются при помощи специального протокола прикладного уровня, называемого DNS (domain name system, система доменных имен).

Протокол прикладного уровня обслуживает множество конечных систем, при этом приложение одной конечной системы обменивается порциями данных с приложением другой конечной системы. Порцию данных прикладного уровня назовем сообщением.

*Транспортный уровень*

Транспортный уровень Интернета осуществляет передачу сообще­ний прикладного уровня между конечными приложениями. Два транс­портных протокола, существующих в Интернете и организующих пе­редачу сообщений прикладного уровня, — это TCP и UDP. Протокол TCP предлагает приложениям службы с установлением соединения. Эти службы обеспечивают надежную доставку сообщений прикладного уровня получателям, а также контроль переполнения (то есть регули­рование скорости потока). TCP также разбивает длинные сообщения на более короткие сегменты и обеспечивает механизм для контроля перегрузок таким образом, что при перегрузке сети источник снижает свою скорость передачи. Протокол UDP предоставляет приложениям службы без установления соединения. При этом не гарантируется на­дежность передачи, нет контроля переполнения и контроля перегрузок. Мы назовем порцию данных транспортного уровня **сегментом.**

*Сетевой уровень*

Сетевой уровень Интернета отвечает за передачу порций данных, известных как **дейтаграммы,** от одного хоста сети к другому. Протоколы транспортного уровня (TCP и UDP) передают сегмент транспортного уровня и адрес назначения на сетевой уровень точно так же, как вы от­правляете письмо на почту с указанием адреса доставки. Сетевой уро­вень, в свою очередь, обеспечивает службу для доставки этого сегмента на транспортный уровень хоста-получателя.

Сетевой уровень Интернета включает протокол IP, который опреде­ляет поля дейтаграмм, а также действия, которые должны производить конечные системы и маршрутизаторы с этими полями. Протокол IP един для всего Интернета, и все компоненты, работающие на сетевом уровне, должны исполнять его. Сетевой уровень содержит также про­токолы маршрутизации, которые определяют маршруты прохождения дейтаграмм между хостами-источниками и хостами-приемниками. Та­ких протоколов маршрутизации в Интернете довольно много. Как мы видели ранее, Интернет является сетью сетей, и каждая сеть, входящая в него, может по желанию администратора использовать свой собственный протокол маршрутизации. Несмотря на то, что сетевой уровень содержит кроме протокола IP еще и многочисленные протоко­лы маршрутизации, их объединяют в одном протоколе IP, потому что на самом деле именно он является связующим звеном Интернета.

***Канальный уровень***

Сетевой уровень обеспечивает передачу дейтаграммы по цепоч­ке маршрутизаторов от источника к приемнику в Интернете. Чтобы переместить пакет от одного узла (хоста или маршрутизатора) к сле­дующему на маршруте, сетевой уровень использует службы каналь­ного уровня; в частности, на каждом узле передает дейтаграмму ниже на канальный уровень, который доставляет ее к следующему узлу на маршруте, а затем канальный уровень передает дейтаграмму вверх на сетевой.

Службы, предоставляемые канальным уровнем, зависят от кон­кретного протокола канального уровня, который используется на определенной линии связи. Например, некоторые протоколы каналь­ного уровня обеспечивают надежную доставку по линии связи от пе­редающего узла к принимающему. Важно отметить, что надежность доставки на канальном уровне отличается от той, что предлагается в протоколах TCP и обеспечивает надежную доставку от одной ко­нечной системы до другой. Примером протоколов канального уровня могут служить Ethernet, Wi-Fi, а также протокол кабельных сетей до­ступа DOCSIS. Когда дейтаграммы на пути следования от источника к приемнику проходят несколько линий связи, то на каждой из них они могут быть обработаны различными протоколами канального уровня. Например, на одном участке маршрута дейтаграмму обрабаты­вает протокол Ethernet, а на другом — протокол РРР. Таким образом, различные протоколы канального уровня предоставляют различные службы для сетевого уровня. Мы назовем порции данных канального уровня **кадрами.**

***Физический уровень***

В то время как работой канального уровня является передача кадров между соседними узлами сети, физический уровень предназначен для передачи отдельных битов кадра между этими узлами. Протоколы фи­зического уровня опять же зависят от используемой линии связи и от реальной среды передачи этой линии (медная витая пара, одномодовое оптоволокно и т.д.). Например, Ethernet поддерживает множество про­токолов физического уровня: один для витой медной пары, другой для коаксиального кабеля, третий для оптоволоконного и так далее. В каж­дом из этих случаев биты передаются по линии связи различными спо­собами.

***Модель OSI***

Мы с вами подробно обсудили стек протоколов Интернета. Но стоит упомянуть, что это не единственный набор протоколов для компьютер­ных сетей. В конце 70-х годов международная организация по стандар­тизации (International Organization for Standardization, ISO) предло­жила так называемую эталонную **модель взаимодействия открытых систем** (Open Systems Interconnection model или просто **модель OSI.** В ней предлагается организовать компьютерные сети с помощью семи уровней. Модель OSI формировалась в то время, когда будущие про­токолы Интернета только начинали свое развитие среди множества других протоколов; в действительности разработчики первоначальной модели OSI, создавая ее, не думали об Интернете. Тем не менее, начиная с конца 70-х годов все образовательные программы по компьютерным сетям строились на базе концепции семиуровневой модели. Поскольку такая модель оказала значительное влияние на образование в области компьютерных сетей, она до сих пор не потеряла свое значение и упо­минается во многих учебниках и обучающих курсах по сетевым техно­логиям.

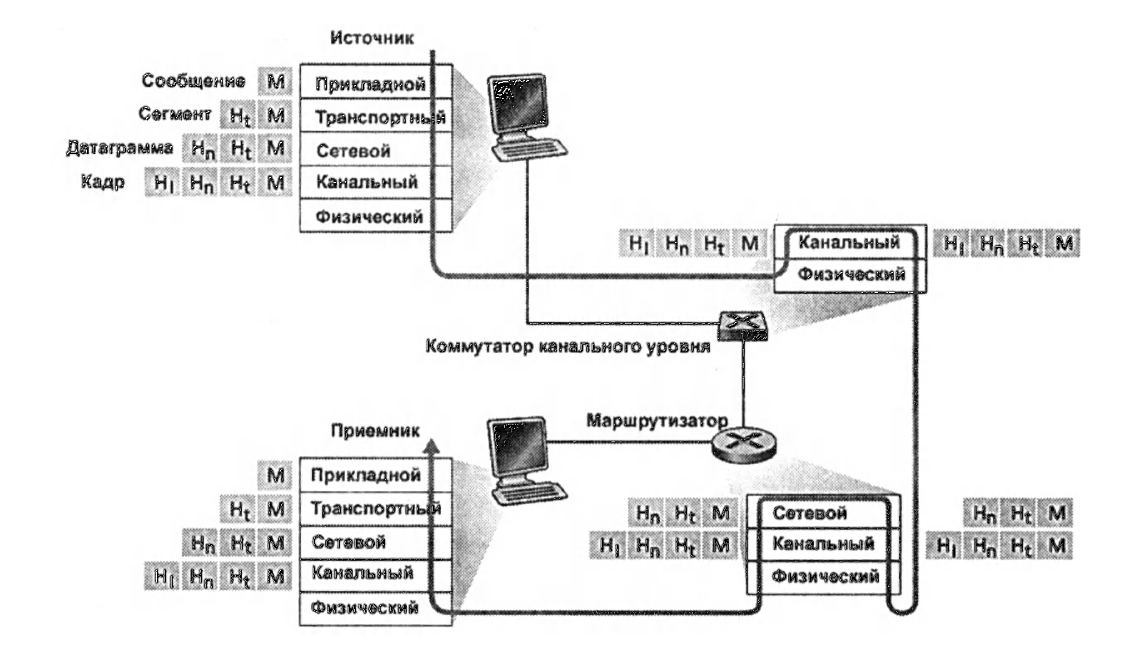
Семь уровней модели OSI, представленные на рис. 1.23б — это при­кладной уровень, уровень представления, сеансовый уровень, транс­портный уровень, сетевой уровень, канальный уровень и физический уровень. Функциональность пяти из этих уровней та же самая, что и у аналогичных уровней стека протоколов Интернета. Давайте рассмо­трим два дополнительных уровня, представленных в эталонной моде­ли OSI — уровень представления и сеансовый уровень. Основная роль уровня представления — обеспечить сервис, позволяющий взаимодей­ствующим приложениям интерпретировать данные, которыми они об­мениваются. Это включает сжатие данных, их шифрование (название говорит само за себя), а также описание (эти службы позволяют приложениям не заботиться о внутренних форматах, в которых данные представлены/сохранены, и которые могут отличать­ся от одного компьютера к другому). Сеансовый уровень обеспечивает разграничение и синхронизацию данных в процессе обмена, предостав­ляет средства для контроля за сеансом и его восстановление в случае разрыва соединения.

Отсутствие этих двух уровней эталонной модели OSI в стеке про­токолов Интернета вызывает пару интересных вопросов: разве службы, предоставляемые этими уровнями, не так важны? Что если приложению *необходимы* эти службы? Интернет отвечает на эти вопросы одинаково — на усмотрение разработчика приложения. Он сам должен решить, важна конкретная служба или нет, и в случае, если она *действительно* важна, он же решает, как встроить такую функциональность в прило­жение.

**Инкапсуляция**

На рис. 1.24 показан путь, который проходят данные от отправляю­щей конечной системы вниз по стеку протоколов этой системы, затем вверх и вниз по протоколам коммутатора и маршрутизатора, и затем вверх по уровням протоколов принимающей конечной системы. Как мы с вами обсудим позднее, и маршрутизаторы, и комму­таторы канального уровня занимаются коммутацией пакетов. Подобно конечным системам, аппаратное и программное обеспечение данных устройств также организовано в виде уровней. Но, в отличие от конеч­ных систем, маршрутизаторы и коммутаторы не используют *все* уровни стеков протоколов. Они обычно работают на нижних уровнях. Как мы видим из рис. 1.24, коммутатор канального уровня использует уровни 1 и 2; в маршрутизаторах реализованы уровни с 1 по 3. Это означает, на­пример, что маршрутизаторы в Интернете могут работать с протоко­лом IP (с протоколом третьего уровня), в то время как коммутаторы канального уровня не способны это делать. Они не могут распознавать IP-адреса, но зато работают с адресами второго уровня, такими как адреса Ethernet. Обратим внимание, что хосты поддерживают все пять уровней, то есть можно сделать вывод, что вся сложность архитектуры Интернета ложится на плечи конечных устройств сети.

Рисунок 1.24 демонстрирует также важное понятие **инкапсуля­ции.** На отправляющем хосте **сообщение прикладного уровня** (М на рис. 1.24) передается на транспортный уровень. В простейшем случае транспортный уровень принимает сообщение и добавляет к нему до­полнительную информацию (так называемый заголовок транспортно­го уровня Ht на рис. 1.24), которая будет впоследствии использоваться транспортным уровнем на принимающей стороне. Сообщение приклад­ного уровня вместе с информацией заголовка транспортного уровня составляют **сегмент транспортного уровня.** Таким образом, сегмент транспортного уровня *инкапсулирует* сообщение прикладного уров­ня. Дополнительная информация, включаемая в заголовок, может со­держать данные, позволяющие транспортному уровню принимающей стороны доставлять сообщения необходимому приложению. Сюда же включаются биты контроля ошибок, с помощью которых получатель определяет количество измененных битов сообщения на маршруте. За­тем транспортный уровень передает сегмент на сетевой уровень, который, в свою очередь, добавляет информацию заголовка своего уровня (Hn на рис. 1.24), например, адреса конечных систем источника и приемника, создавая, таким образом, **дейтаграмму сетевого уровня.** Дейтаграмма за­тем пересылается на канальный уровень, который (естественно) также добавляет свой собственный заголовок и создает **кадр канального уров­ня.** Таким образом, мы видим, что пакет на каждом уровне содержит два типа поля — поле заголовка и **поле данных,** которые обычно содержат пакет из уровня, расположенного над ним.



**Рис. 1.24. Хосты, маршрутизаторы и канальные коммутаторы реализуют различные наборы уровней, что отражает разницу в их функциональности**

Уместным будет провести аналогию с пересылкой сообщения из одного офиса компании в другой с помощью почтовой службы. Пред­положим, Алиса, работающая в одном офисе, хочет отправить Бобу в другой офис сообщение. Оно будет аналогично *сообщению прикладно­го уровня.* Алиса помещает письмо в корпоративный конверт и пишет на его лицевой стороне имя Боба и название подразделения, в котором он **работает. *Корпоративный конверт* аналогичен *сегменту транспортного*** *уровня.* Он содержит информацию заголовка (имя Боба и номер подраз­деления), а также инкапсулирует (упаковывает) сообщение прикладно­го уровня (письмо от Алисы). Когда этот конверт приходит к работни­кам офиса, отвечающим за отправку корреспонденции, они помещают его внутрь еще одного конверта, который можно послать по обычной городской почте. Работники административного отдела пишут почтовый адрес отправляющего и принимающего офисов на почтовом конверте. В данном случае *почтовый конверт —* это аналог *дейтаграммы —* он ин­капсулирует сегмент транспортного уровня (корпоративный конверт), который, в свою очередь, инкапсулирует исходное сообщение (сообще­ние от Алисы). Почтовая служба доставляет почтовый конверт в офис принимающей стороны — в отдел корреспонденции. Там начинается процесс деинкапсуляции. Сотрудники отдела по работе с корреспон­денцией извлекают из почтового конверта корпоративный и направля­ют его Бобу. Наконец Боб открывает конверт и извлекает сообщение.

Конечно, процесс инкапсуляции может быть гораздо сложнее, чем в описанном выше примере. Например, большое сообщение может быть разделено на ряд сегментов транспортного уровня (а те, в свою очередь, на множество дейтаграмм сетевого уровня). На принимающем конце такие сегменты должны быть заново собраны из входящих в него дей­таграмм.

**Атаки на сети**

Интернет стал играть большую роль в деятельности сегодняшних организаций, включая большие и малые компании, образовательные учреждения и органы власти. Многие люди также полагаются на Ин­тернет как на помощника в профессиональной деятельности, в обще­ственной и личной жизни. Но, несмотря на все достоинства Интернета и удобства, которые он предоставляет, существует и другая — темная сторона, когда «плохие парни» пытаются внести хаос в нашу повседнев­ную жизнь, нанося вред нашим компьютерам, парализуя работу служб сети, от которых мы стали сильно зависимы, а также вторгаясь в нашу личную жизнь.

Задача безопасности сетей заключается в том, чтобы знать, как эти плохие парни могут атаковать компьютерные сети, и как мы, будущие эксперты, можем противостоять этим атакам, а лучше всего, как мы мо­жем построить новую архитектуру сетей, которая в первую очередь бы подразумевала защиту от таких угроз. Учитывая частоту и разнообразие существующих атак, а также угрозу новых, более разрушительных в бу­дущем, безопасность сети становится важнейшей темой в области ин­формационных технологий. Актуальность сетевой безопасности — это один из ключевых моментов изучения в данном курсе.

Так как мы еще недостаточно опытны в области компьютерных се­тей и протоколов Интернета, мы начнем с обзора самых распространен­ных сегодняшних проблем, связанных с безопасностью. Это подогреет наш аппетит для более предметных обсуждений в будущем, так что мы начнем с простых вопросов: что может произойти плохого? Насколько уязвимы компьютерные сети? Каковы на сегодняшний день самые основные типы атак на сети?

***Злоумышленники могут устанавливать вредоносное ПО на вашем компьютере, используя Интернет***

Мы с вами подключаем различные устройства к Интернету для того, чтобы получить какую-нибудь информацию из сети либо передать ее туда. Наша активность в Интернете связана с различными полезными видами деятельности, включая просматривание веб-страниц, получение сообщений электронной почты, скачивание музыкальных файлов, рабо­ту в поисковых системах, телефонные вызовы, видео в реальном вре­мени и так далее. Но, к сожалению, Интернет может оказать и вредное влияние, например, инфицировать наши с вами компьютеры широко известным **вредоносным программным обеспечением.** Однажды по­пав на наш компьютер, такое вредоносное ПО способно на различные пакости, включая удаление файлов и установку шпионских программ, которые могут собирать нашу личную информацию, такую, как персо­нальные пароли, а затем пересылать ее (естественно, используя Интер­нет) злоумышленникам. Наш подвергшийся атаке компьютер может стать частью сети из тысяч подобных устройств. Такую сеть, известную как **ботнет,** злоумышленники контролируют и используют для рассыл­ки спама, а также проведения атак типа отказа в обслуживании (вскоре будет обсуждаться) против целевых хостов.

Большинство вредоносных программ на сегодняшний день явля­ются **саморазмножающимися.** Заразив один хост, такая программа ищет способ попасть на другие устройства, используя Интернет, а, за­разив их, пытается распространиться дальше. Таким образом, распро­странение вредоносного ПО может происходить с достаточно большой скоростью. Такое ПО обычно бывает в форме вируса или червя. **Виру­сы** — это вредоносные программы, которые требуют некоторой фор­мы взаимодействия с пользователем для заражения устройства этого пользователя. Классическим примером вируса является вложение по­чтового сообщения, которое содержит вредоносный исполняемый код. Когда пользователь получает электронную почту и открывает такое вложение, то он непроизвольно запускает вредоносное ПО на своем устройстве. Как правило, такие почтовые вирусы размножаются сами по себе: после первого запуска вирус способен послать идентичное со­общение с тем же самым вредоносным вложением, например, каждому получателю из адресной книги пользователя зараженного компьюте­ра. **Черви** — это разновидность вредоносного ПО, которая попадает на устройство без явного взаимодействия с пользователем. Например, если тот запускает какое-нибудь сетевое приложение, где существу­ет уязвимость, используя которую злоумышленник может отправить вредоносную программу. В некоторых случаях приложение без вмеша­тельства пользователя может принимать вредоносную программу из Интернета и запускать ее, создавая тем самым червя. Затем червь в но­вом зараженном устройстве сканирует сеть в поисках другого хоста, на котором запущено то же сетевое приложение с той же уязвимостью. Найдя такие уязвимые хосты, червь посылает на них копию самого себя. На сегодняшний день распространение вредоносного ПО стало повсеместным, и защититься от него достаточно непросто. В процессе работы с книгой постарайтесь подумать, как ответить на вопрос: что могут предпринять разработчики компьютерных сетей для защиты устройств, подсоединенных к Интернету, от атак вредоносного про­граммного обеспечения?

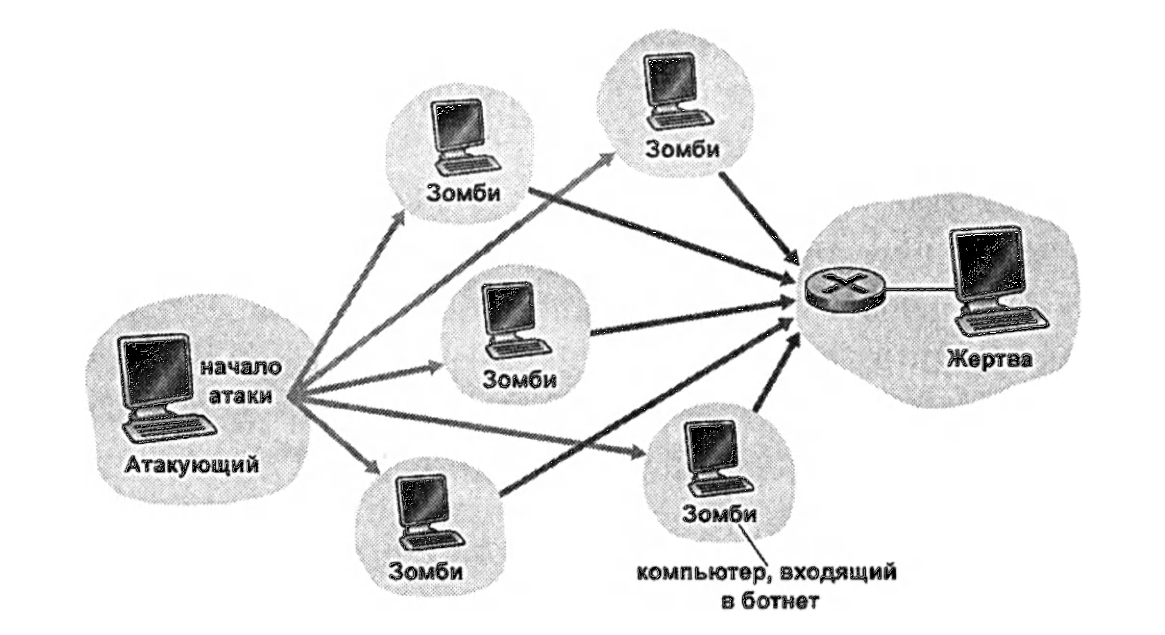
***Злоумышленники могут атаковать сервера и сетевую инфраструктуру***

Еще один широкий класс угроз безопасности представляют собой **DoS-атаки** (denial-of-service, отказ в обслуживании). Как следует из названия, DoS-атаки перегружают ресурсы сети и делают их недоступ­ными для пользователей. Объектами DoS-атак могут быть веб-серверы, почтовые серверы, DNS-серверы, а также сети любой организации. DoS-атаки в Интернете очень распространены: каждый год их происходят тысячи. Большинство DoS-атак в Ин­тернете можно разделить на три категории:

• *Атака на уязвимость.* Данный вид атаки предполагает отправку не­скольких хорошо продуманных сообщений приложению или опе­рационной системе, запущенным на сетевом хосте и имеющим уяз­вимость. Если послать определенную последовательность пакетов уязвимому приложению или операционной системе, то это может привести к остановке работы служб или, что еще хуже, к выводу из строя всего устройства.

* *Переполнение полосы пропускания.* Злоумышленник посылаетогромное количество пакетов на целевой хост — такое огромное, что соединение целевого хоста становится перегруженным и, как след­ствие, сервер становится недоступным для пакетов пользователей.
* *Переполнение запросов на соединение.* Атакующий устанавливаетбольшое количество полуоткрытых TCP-соединений на целевом хосте. В результате хост настолько перегружается этими фиктивными соединениями, что прекращает принимать соединения законных пользователей.

Давайте более подробно рассмотрим атаку с помощью переполнения полосы пропускания. Если вспомнить наше обсуждение задержек и по­терь, то будет очевидно, что если сервер имеет скорость *R* бит/с, то злоумышленнику необходимо отправлять трафик со скоро­стью приблизительно равной *R,* чтобы причинить ущерб серверу. При достаточно высоком значении *R* единственный атакующий источник не способен сгенерировать для этого достаточно трафика. Более того, если трафик исходит из одного источника, то принимающий маршрутизатор вполне может обнаружить атаку и блокировать все пакеты от данного источника, и они не дойдут до сервера. При использовании **распреде­ленных DoS-атак** (distributed DoS или **DDoS),** показанных на рис. 1.25, злоумышленник управляет множественными источниками, и каждый из них направляет огромный трафик на целевой компьютер.

****

**Рис. 1.25. Распределенная DoS-атака**

В таком случае совокупная скорость трафика от всех источников должна быть приблизительно равна *R,* чтобы навредить службам сер­веров. DDoS-атаки, использующие ботнеты с тысячами зараженных хостов — это широко распространенное явление в наши дни. По сравне­нию с DoS-атаками с одного хоста, DDoS-атаки представляют большую угрозу, так как их сложнее обнаружить и от них труднее защититься.

***Злоумышленники могут перехватывать пакеты***

Сегодня многие пользователи выходят в Интернет, используя бес­проводные устройства, как, например, ноутбуки, подключенные через Wi-Fi, или портативные устройства типа планшетов или смартфонов, получающие доступ с помощью операторов мобильной связи. В то время как повсеместный доступ в Интернет — исключительно удобная вещь, позволяющая мобильным пользова­телям работать с многочисленными приложениями, это же, с другой стороны, создает большую уязвимость — если в непосредственной близости от беспроводного передатчика расположить пассивный при­емник, то можно получить копию любого переданного пакета! Такие пакеты могут содержать, в том числе любые виды конфиденциальной информации, включая пароли, PIN-коды, другую важную секретную информацию, а также личные сообщения. Такой пассивный приемник, делающий копию каждого пакета, называется **сниффером (анализа­тором) пакетов.**

Снифферы могут быть развернуты также и в проводных средах. Например, во многих ЛВС на базе Ethernet сниффер может получать копии широковещательных пакетов, посланных через ЛВС. Технологии кабельного доступа также используют широко­вещательные пакеты и, следовательно, тоже подвержены перехвату па­кетов. Более того, злоумышленник, получив доступ к маршрутизатору или линии соединения с Интернетом какой-то организации, способен заставить анализатор копировать любой отправленный или получен­ный пакет. Перехваченные пакеты впоследствии могут быть проанали­зированы на предмет конфиденциальной информации в другом месте. Снифферы пакетов доступны как свободное ПО на различных веб­сайтах, а также в качестве коммерческих продуктов. Профессора, пре­подающие курсы по компьютерным сетям, часто используют в сво­их уроках упражнения, включающие написание анализатора пакетов и программ для восстановления данных прикладного уровня. Поскольку программы такого рода являются пассивным программным обеспечени­ем — то есть они не генерируют никаких пакетов, — их очень сложно об­наружить. Следовательно, когда мы посылаем пакеты в беспроводную линию, мы должны понимать, что существует вероятность перехвата их злоумышленниками. Как вы, наверно, догадались, один из лучших ме­тодов защиты от перехвата пакетов — это криптография.

***Злоумышленники могут маскироваться под кого-то, кому вы доверяете***

Существует на удивление простой способ создать пакет с произвольным адресом источника, содержимым и адре­сом приемника и затем отправить этот самодельный пакет в Интернет, который послушно перенаправит его по адресу назначения. Представь­те себе ничего не подозревающего получателя (например, Интернет-маршрутизатор), который примет такой пакет, воспримет адрес источ­ника (ложный) как истинный и затем выполнит некоторые команды, встроенные в содержимое пакетов (которые, например, модифицируют его таблицы маршрутизации). Такая возможность внедрить в сеть паке­ты с ложным адресом источника называется **IP-спуффингом** (или под­меной адреса) и является примером способа, с помощью которого один пользователь может маскироваться под другого.

Чтобы решить данную проблему, нам потребуется *аутентификация конечного пользователя,* то есть механизм, который позволит нам с уве­ренностью определить, что сообщение исходит именно оттуда, откуда мы думаем.

**ПРИКЛАДНОЙ УРОВЕНЬ**

В сетевых приложениях заключается весь смысл существования компьютерных сетей — если бы не было полезных приложений, нам бы не нужны были сетевые протоколы, их поддерживающие. С самого на­чала существования Интернета было создано множество как серьезных и полезных, так и развлекательных сетевых приложений, которые, без сомнения, явились основным движущим фактором успеха всемирной сети, мотивируя людей сделать ее частью своей повседневной деятель­ности на работе, дома и на учебе.

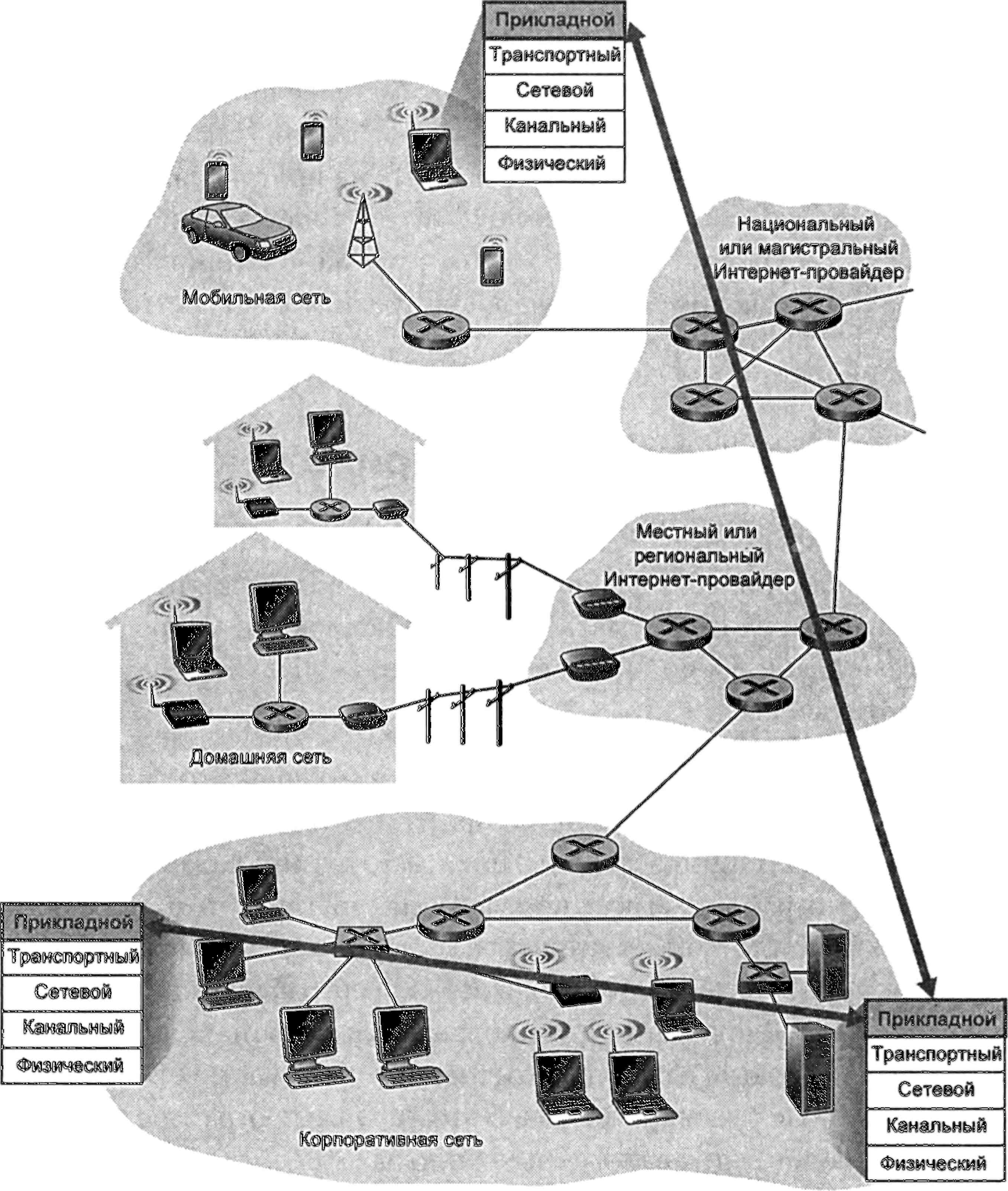
Интернет-приложения включают в себя классические текстовые приложения, которые стали популярны еще в 70-х и 80-х годах: тексто­вая электронная почта, удаленный доступ к устройствам по сети, пере­дача файлов, сообщения групп новостей. Среди них также ключевые приложения середины 90-х годов, которые легли в основу всемирной паутины: веб-обозреватели, поисковые системы, системы электронной торговли. В этот же ряд можно добавить приложения для систем мгно­венных сообщений и одноранговых файлообменных сетей, представ­ленные в конце тысячелетия. Начиная с 2000 года мы наблюдали взрыв популярности аудио- и видеоприложений, включая IР-телефонию, а также видеоконференции по IP-сетям, такие как Skype, приложений для размещения пользовательского контента, например, YouTube, для за­грузки видеоконтента по запросу, таких как NetFlix. В это же время рас­пространяется всеобщее увлечение многопользовательскими онлайн-играми, такими как Second Life или World Of Warcraft. Совсем недавно появилось новое поколение приложений для социальных сетей, таких как Facebook или Twitter, которые создали своеобразную надстройку из общественных сетей над сетями из маршрутизаторов и линий связи. Очевидно, что эта сфера будет развиваться и в дальнейшем, и вполне возможно, что некоторые из наших читателей станут создателями сле­дующего поколения важнейших Интернет-приложений!

**Принципы сетевых приложений**

Предположим, у вас есть идея нового сетевого приложения. Возможно, что оно принесет величайшую пользу человечеству или просто порадует вашего учителя, сделает вас миллионером или просто займет на досуге — какой бы ни была ваша мотивация, давайте подумаем, как вам воплотить идею в жизнь.

Ключевую часть разработки сетевых приложений составляет написание программ, которые работают на различных конечных системах и общаются друг с другом по сети. Например, веб-приложение — это две различные программы, взаимодействующие друг с другом: браузер, запущенный на хосте пользователя (настольном компьютере, ноутбуке, планшете, смартфоне и так далее), и веб-сервер, работающий на серверном хосте. Другой пример — это одноранговые системы совместного до­ступа к файлам, где на каждом из хостов, который участвует в файловом обмене, запущена такая программа. В этом случае программы на различных хостах могут быть аналогичными или даже идентичными.

Таким образом, для разработки нового приложения вам нужно на­писать программное обеспечение, которое бы работало на различных конечных системах. Вы можете использовать, например, языки Си, Java или Python. Важно отметить здесь, что вам не нужно писать програм­мное обеспечение для устройств, составляющих ядро сети, таких как маршрутизаторы или коммутаторы канального уровня. Но даже если бы вы и задались такой целью, вы бы не смогли это сделать, ведь, как было показано ранее на рис. 1.24, данные устройства функционируют не на прикладном уровне, а на сетевом и более низких уровнях.



**Рис. 2.1. Взаимодействие сетевых приложений происходит между конечными**

**системами на прикладном уровне**

Именно тот принципиальный факт, что программное обеспечение приложений для сети относится исключительно к конечным системам (как показано на рис. 2.1), способствовал быстрому развитию и распро­странению огромной массы сетевых приложений.

**Архитектура сетевых приложений**

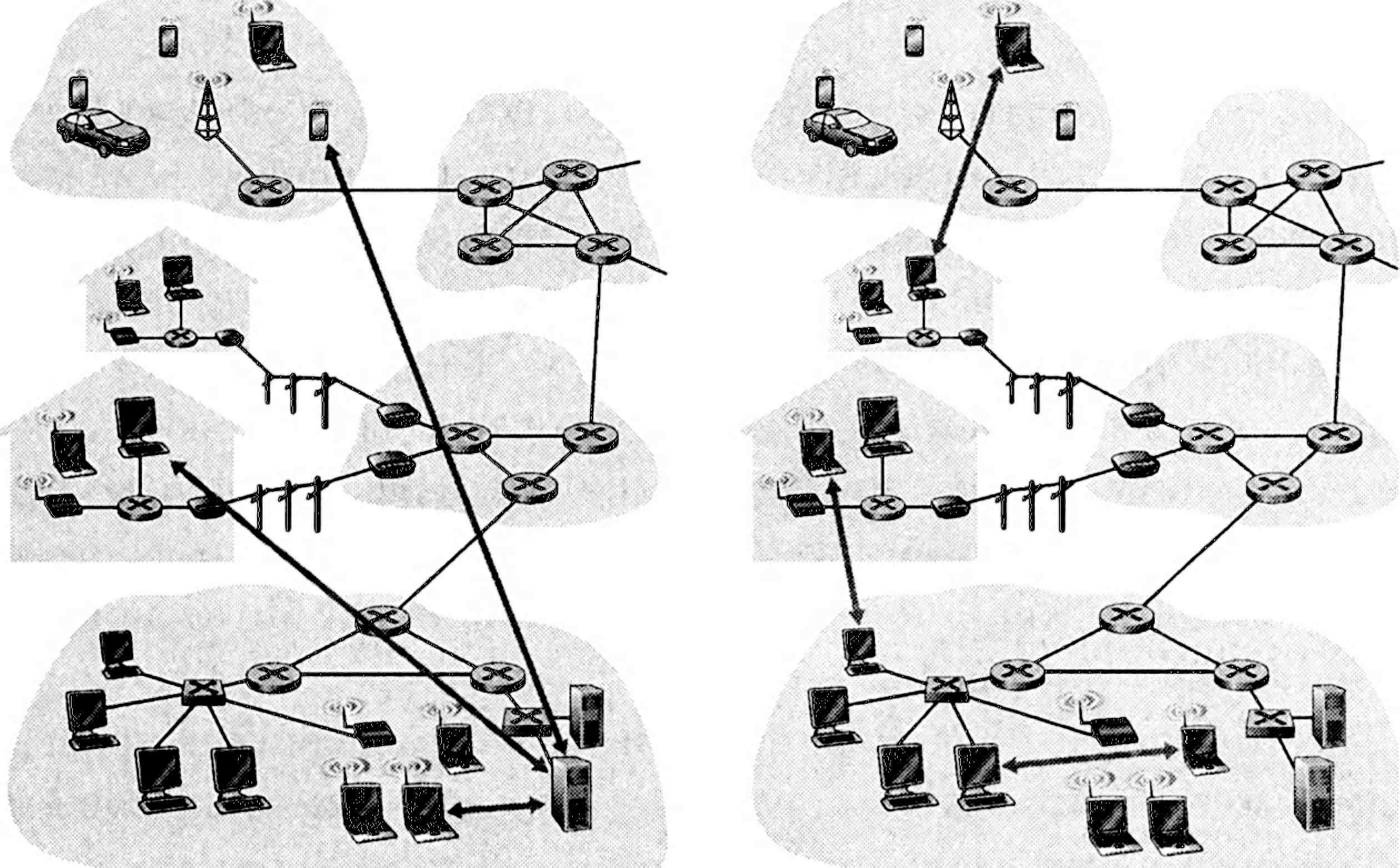
Перед тем как углубляться в разработку программного обеспечения, нам нужно набросать план построения вашего приложения. Необходимо помнить, что архитектура приложений и архитектура сети (например, пятиуровневая архитектура Интернета — это разные вещи. С точки зрения разработчика приложения, архитектура сети постоянна и предлагает определенный набор служб приложениям. С другой стороны, **архитектура приложения** создается разработчиком этого приложения и определяет, каким образом оно будет строиться на различных конечных системах. Выбирая архитектуру приложения, разработчик, скорее всего, выберет одну из двух доминирующих архитектурных парадигм, используемых в современной разработке сетевых приложений: клиент-серверная или одноранговая (Р2Р).

В **клиент-серверной архитектуре** существует один хост, называе­мый *сервером,* который постоянно находится в режиме онлайн и обслу­живает запросы от других многочисленных хостов, называемых *клиен­тами.* Классическим примером является веб-приложение, для которого постоянно работающий веб-сервер обслуживает запросы, поступающие от браузеров, запущенных на клиентских хостах. Когда веб-сервер полу­чает запрос объекта от клиентского хоста, он в ответ отправляет запра­шиваемый объект этому хосту. Заметим, что в данном виде архитектуры клиенты непосредственно не связываются друг с другом; например, в веб-приложении два браузера напрямую не обмениваются инфор­мацией. Еще одной характеристикой клиент-серверной архитектуры является то, что сервер имеет фиксированный, известный всем адрес, называемый IP-адресом (обсудим вскоре). Поскольку сервер имеет постоянный, известный адрес и всегда включен, клиент может всегда взаимодействовать с ним, отправляя пакеты на IP-адрес этого сервера. Некоторые из хорошо известных приложений клиент-серверной ар­хитектуры — это Всемирная паутина, FTP, Telnet и электронная почта. Данная архитектура показана на рис. 2.2а.

Очень часто при работе клиент-серверных приложений серверный хост не способен в одиночку обрабатывать многочисленные запросы от клиентов. Например, если бы веб-сайт популярной социальной сети включал в себя один сервер, то он быстро был бы перегружен запросами. По этой причине очень часто используются **центры обработки данных (дата-центры),** содержащие в себе большое количество хостов и обра­зующие мощный виртуальный сервер. Наиболее популярные Интернет-службы — такие как поисковые системы (например, Google или Bing), сервисы продаж через Интернет (например, Amazon или е-Вау), элек­тронная почта с доступом через веб-интерфейс (например, Gmail или Yahoo Mail), приложения социальных сетей (например, Facebook или Twitter) — используют один или более центров обработки данных. Компания Google имеет от 30 до 50 таких центров по всему миру, которые совместно обрабатывают поисковые запросы, обеспечивают работу Gmail, YouTube и других служб. Современные центры обработки данных (ЦОД) могут насчитывать сотни тысяч серверов, а значит, провайдеру услуг потребуются значительные расходы не только на поддержку и обслуживание ЦОДов, но и дополнительные затраты, связанные с передачей данных из их центров обработки.

В **одноранговой (Р2Р) архитектуре** применение серверов или цен­тров обработки сведено до минимума или вообще до нуля. Вместо них приложения используют непосредственное взаимодействие между па­рой соединенных хостов, называемых *пирами (а также партнерами или узлами).* Пирами являются обычные настольные компьютеры или ноутбуки, контролируемые пользователями и размещаемые в учебных заведениях, офисах или домах. Так как пиры взаимодействуют без выде­ленного сервера, такая архитектура называется одноранговой (peer-to-peer). Многие из сегодняшних популярных приложений, использующих наиболее интенсивный трафик, основываются на такой одноранговой архитектуре. Сюда можно отнести файлообменные приложения, напри­мер, BitTorrent, ускорители загрузок (например, Xunlei), IP-телефонию (например, Skype) и IP-телевидение (например, Kankan и PPstream). Одноранговая архитектура показана на рис. 2.26. Отметим, что некото­рые приложения построены с использованием гибридной архитектуры, где совмещаются клиент-серверный и одноранговый подходы. Напри­мер, во многих приложениях систем мгновенных сообщений серверы используются для отслеживания IP-адресов пользователей, но сообще­ния от пользователя к пользователю посылаются напрямую между двумя хостами (не проходя через промежуточные серверы).

Одним из наиболее выигрышных свойств одноранговой архитекту­ры является ее **самомасштабируемость.** Например, в файлообменном приложении каждый пир не только нагружает систему своими запро­сами файлов, но и, раздавая файлы другим узлам, в то же время увели­чивает скоростной ресурс системы. Такая архитектура эффективна еще и с точки зрения стоимости, так как обычно не требует значительных расходов на серверную инфраструктуру (в отличие от дорогих центров обработки данных в случае с клиент-серверной архитектурой).



**а. Клиент-серверная архитетура б. Одноранговая архитетура**

**Рис. 2.2. (а) Клиент-серверная архитектура; (б) Одноранговая архитектура**

Однако использование одноранговых приложений связано с темя довольно значительными проблемами:

1. *Адаптация к сетям доступа.* Большинство резидентных сетей до­ступа (включая DSL и кабельные сети) построены по асимметрич­ному принципу, когда скорость входящего трафика значительно превышает скорость исходящего. Поскольку одноранговые прило­жения для обработки видеопотока и обмена файлами смещают ис­ходящий трафик от серверов к сетям доступа, нагрузка на эти сети значительно увеличивается. Поэтому в будущем одноранговые сети нужно проектировать таким образом, чтобы они были адаптированы к сетям доступа в плане балансировки нагрузки.
2. *Безопасность.* Одноранговые сети в связи с их широким распростра­нением и открытой природой представляют собой проблему безопасности
3. *Стимулирование.* Успешное будущее одноранговых сетей связано с задачей привлечения пользователей, которые добровольно будут предлагать свои ресурсы вычисления, хранения и передачи данных для приложений. Организация такой системы стимулирования также является одной из проблем будущего одноранговых сетей.

**Взаимодействие процессов**

Перед тем как начать разрабатывать сетевое приложение, у вас долж­но быть базовое понимание того, как программы, запущенные на много­численных конечных системах, взаимодействуют друг с другом. С точ­ки зрения операционных систем, на самом деле взаимодействуют не программы, а **процессы.** Процесс можно рассматривать как программу, запущенную на конечной системе. Когда процессы работают на одной конечной системе, они могут общаться друг с другом с помощью средств межпроцессного взаимодействия, используя набор правил, регулируе­мый операционной системой конкретного устройства. Но нас не интересует, как общаются между собой процессы на одном хо­сте. Вместо этого мы будем рассматривать взаимодействие процессов, запущенных на *различных* хостах (и, вполне вероятно, отличающимися операционными системами).

Процессы на двух различных конечных системах взаимодействуют друг с другом, обмениваясь **сообщениями** через компьютерную сеть. Процесс на хосте-источнике создает и отправляет сообщения в сеть; процесс на хосте-приемнике получает эти сообщения и, возможно, от­правляет в ответ обратные. Рисунок 1.1 показывает, что процессы взаи­модействуют друг с другом на прикладном уровне пятиуровневого стека протоколов.

**Клиентский и серверный процессы**

Сетевое приложение состоит из пар процессов, которые отправля­ют сообщения друг другу по сети. Например, в веб-приложении процесс браузера клиента обменивается сообщениями с процессом веб-сервера. В системе однорангового файлового обмена файл передается из процес­са одного хоста в процесс другого хоста. Для каждой пары взаимодей­ствующих процессов обычно обозначают один из них как **клиентский,** а другой как **серверный.** Например, браузер — это клиентский процесс, а веб-сервер — это серверный процесс. В системе файлового обмена пир, который загружает файл, является клиентом, а пир, выгружающий файл, считается сервером.

Вы, возможно, замечали, что в некоторых приложениях (например, одноранговый файловый обмен) процесс может быть как клиентским, так и серверным. В действительности в такой системе процесс может как выгружать, так и загружать файлы. Тем не менее в контексте любого конкретного сеанса взаимодействия между парой процессов мы всегда обозначаем один процесс как клиент, а другой как сервер. Определим понятия клиентского и серверного процессов следующим образом:

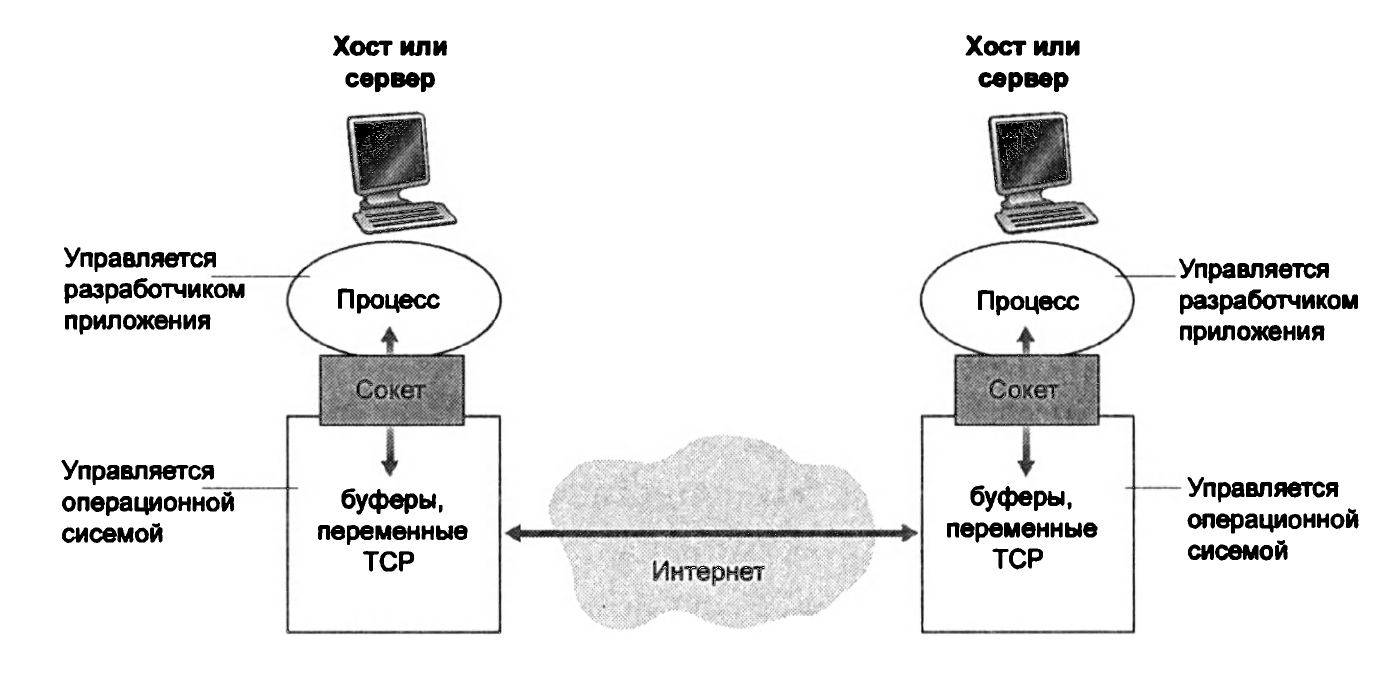
*В контексте сеанса взаимодействия между парой процессов тот, который инициирует это взаимодействие (то есть первоначально кон­тактирует с другим процессом в начале сеанса), обозначается как* ***кли­ентский.*** *Процесс, который ожидает контакта для начала сеанса, обо­значаем как* ***серверный.***

Итак, процесс браузера инициирует контакт с процессом веб-сервера; следовательно, процесс браузера является клиентским, а процесс веб-сервера является серверным. При одноранговом файловом обмене, ког­да пир А запрашивает файл у пира Б, то пир А — клиент, а пир Б — сервер в контексте этого конкретного сеанса взаимодействия. Иногда мы ис­пользуем терминологию «клиентская и серверная сторона (или часть) приложения».

**Интерфейс между процессом и компьютерной сетью**

Как было отмечено выше, большинство приложений состоят из пар взаимодействующих процессов, отправляющих друг другу сообщения. Любое из них должно проходить через нижележащую сеть. Процессы отправляют и принимают сообщения через программный интерфейс, называемый **сокетом.** Рассмотрим аналогию для лучшего понимания процессов и сокетов. Процесс аналогичен дому, а его сокет подобен двери в этом доме. Когда процесс пытается отправить сообщение другому процессу на удаленном хосте, он «проталкивает» сообщение через свою дверь (сокет). Такая отправка предполагает, что с другой стороны этой двери существует некоторая транспортная инфраструктура, которая организует доставку сообщения до двери процесса назначения. Когда сообщение прибывает на хост назначения, оно проходит через дверь принимающего процесса (через сокет). Принимающий процесс затем обрабатывает это сообщение.

Рисунок 2.3 демонстрирует взаимодействие между двумя процес­сами через сокеты по Интернету. (Этот рисунок предполагает, что базо­вый транспортный протокол, используемый процессами — это протокол TCP.) Как мы видим из рисунка, сокет — это интерфейс между приклад­ным и транспортным уровнями внутри хоста. Его определяют также как **интерфейс программирования приложений (Application Programming Interface, API)** между приложением и сетью, так как сокет — это про­граммный интерфейс, с помощью которого строятся сетевые приложения. Разработчик приложения управляет всем, что находится со стороны при­кладного уровня сокета, но со стороны транспортного уровня он способен только контролировать, во-первых, выбор транспортного протокола и, во-вторых, возможность фиксировать некоторые параметры транспортного уровня, такие как максимальный буфер и максимальный размер сегмента. Когда разработчик приложения выбира­ет транспортный протокол (если такой выбор доступен), то приложение создается с использованием служб транспортного уровня, предоставляе­мых этим протоколом.

****

**Рис. 2.3. Сокеты — интерфейс между процессами приложений и транспортным**

**протоколом**

**Адресация процессов**

Для того чтобы отправить письмо конкретному адресату, нужно иметь его адрес. То же самое можно сказать и о процессах. Когда запу­щенный на одном хосте процесс пытается отправить пакеты процессу, работающему на другом хосте, то необходимо знать адрес второго про­цесса. Для идентификации процесса-получателя необходима информация о двух вещах: во-первых, об адресе хоста назначения и, во-вторых, об идентификаторе, который определяет нужный нам процесс на хосте назначения.

В Интернете хост можно идентифицировать по его **IP-адресу.** IP-адрес представляет собой 32-разрядное число, которое однозначно определяет хост. В дополнение к адресу хоста отправляющий процесс должен идентифицировать процесс принимающий (а точнее, принимающий сокет), запущенный на хосте-получателе. Такая информация необходима, потому что в общем случае на хосте могут быть запущены многие сетевые приложения. **Номер порта** назначения решает эту задачу. Наиболее популярным сетевым приложениям назначены определенные номера портов. Например, веб-сервер идентифицируется портом 80. Процесс почтового сервера, использующий протокол SMTP, использует порт 25. Список номеров широко известных портов для всех стандартных протоколов Интернета можно посмотреть на сайте [**www.iana.org**](http://www.iana.org)**.**

**Транспортные службы, доступные приложениям**

Вспомним, что сокет является интерфейсом между процессом при­ложения и протоколом транспортного уровня. Приложение на переда­ющей стороне проталкивает сообщения через сокет. С другой стороны сокета протокол транспортного уровня отвечает за доставку сообщений к сокету принимающего процесса.

Во многих сетях, включая Интернет, используется более чем один транспортный протокол. Когда вы разрабатываете приложение, вам нужно выбрать один из доступных протоколов транспортного уровня. Как сделать этот выбор? Вероятнее всего, вы изучите службы, которые предлагают доступные протоколы, и затем выберете протокол с теми службами, которые наиболее подходят для нужд вашего приложения. Ситуация похожа на ту, когда вы, собираясь в путешествие, выбираете вид транспорта — самолет или поезд. Вам нужно выбрать один из двух, причем каждый из них предлагает различные виды доставки (например, поезд может предложить промежуточные остановки, в то время как са­молет — более короткое время путешествия).

Какие службы протокол транспортного уровня предоставляет ис­пользующим его приложениям? Мы можем классифицировать возможные службы по четырем критериям: надежная передача данных, про­пускная способность, время доставки и безопасность.

**Надежная передача данных**

Как обсуждалось ранее, в компьютерных сетях происходит потеря пакетов. Например, пакет может переполнить буфер маршрутизатора или быть отброшен хостом или маршрутизатором после получения по­врежденных битов. Потеря данных во многих приложениях — таких как электронная почта, передача файлов, удаленный доступ, финансовые программы — может иметь довольно значительные последствия (в по­следнем случае — как для банка, так и для клиента!). Таким образом, чтобы решить эту проблему, нужна какая-то гарантия, что отправлен­ные данные с одного конца приложения полностью и без ошибок будут доставлены на другой конец. Если протокол обеспечивает гаранти­рованную службу доставки, то говорят, что обеспечивается **надежная передача данных.** Одна из важнейших служб, которые транспортный протокол может предоставить приложению — это как раз надежная пе­редача данных от процесса к процессу. Если транспортный протокол предоставляет такую службу, то передающий процесс может просто от­править данные в сокет и быть уверенным, что они прибудут в прини­мающий процесс без ошибок.

В случае, когда транспортный протокол не может обеспечить надеж­ную передачу, то часть данных, отправленных передающим процессом, могут никогда не достигнуть процесса-получателя. Такое вполне может быть приемлемо для ряда **приложений, устойчивых к потерям** данных, например, некоторых мультимедиа-приложений, где потеря данных не является критической, а может привести всего-навсего к незначитель­ным помехам.

**Пропускная способность**

Ранее мы познакомились с понятием доступной пропускной способности, которая в контексте сеанса взаимодействия между двумя процессами в сети является ничем иным, как скоростью, с которой передающий процесс может доставлять биты процессу принимающему. Из-за того, что доступную полосу пропускания будут разделять и другие сеансы, а также по причине того, что количество этих сеансов будет непостоянно, значение доступной пропускной способности мо­жет изменяться во времени. Учитывая это, можно прийти к естественному выводу, что транспортный протокол должен также предоставлять еще одну службу, а именно гарантированную доступную пропускную способность, то есть доставку данных с определенной минимальной скоростью. Используя такую службу, приложение может запрашивать гарантированную пропускную способность *r бит/с,* и транспортный протокол должен будет заботиться о том, чтобы доступная скорость передачи данных была не меньше *r бит/с*. Такая гарантированная пропускная способность необходима многим приложениям. Например, если приложение IP-телефонии кодирует голосовые сообщения со скоростью 32 Кбит/с, то необходимо, чтобы данные отправлялись в сеть и доставлялись принимающему приложению с этой скоростью. Если транспортный протокол не может предоставить такую пропускную способность, приложению нужно будет осуществлять кодирование на более низкой скорости (и получать достаточную пропускную способность для поддержки этой низкой скорости кодировки), либо приложение завершит свою работу, так как не сможет использовать, например, имеющуюся в его распоряжении половину необходимой пропускной способности. Приложения, которым требуется определенная пропускная способность, называются **чувствительными к скорости передачи данных.** Такими являются многие мультимедиа-приложения, хотя в некоторые из них уже встроены инструменты для адаптации кодировки голоса или видео на скорости, соответствующей доступной пропускной способности.

В отличие от чувствительных к скорости передачи приложений, **эластичные приложения** используют доступную пропускную спо­собность. Эластичными являются: электронная почта, передача фай­лов, веб-приложения. В любом случае, чем шире полоса пропускания канала, тем лучше. Ведь как говорится «ширины канала много не бывает».

**Время доставки**

Протокол транспортного уровня может также обеспечивать гаран­тии относительно времени доставки сообщений. Временные гарантии тоже предоставляются в различной форме. Например, протокол может гарантировать, что каждый бит, отправленный передающей стороной в сокет, приходит на сокет получателя не более чем через 100 мс. Служба такого рода будет полезна интерактивным приложениям реального времени, таким как IP-телефония, виртуальное окружение, телеконфе­ренции, а также многопользовательские игры. Все эти виды приложений для эффективной своей работы требуют ограничений по времени доставки. Из-за продолжительных задержек в IP-телефонии, например, могут возникать паузы в разговоре; в многопользовательской игре или в виртуальном интерактивном окружении долгая задержка между действием и ответом на него (например, от другого игрока на другом конце соединения) может привести к потере реалистичности. Для приложений, не являющихся приложениями реального времени, низкие задержки, конечно, тоже предпочтительны, но жестких ограничений по времени нет.

**Безопасность**

Наконец, транспортный протокол может предоставлять приложени­ям одну или несколько служб, относящихся к безопасности. Например, транспортный протокол на передающем хосте способен шифровать все данные, отправленные процессом-источником, а затем на принимаю­щем хосте расшифровывать их перед доставкой процессу-получателю. Это обеспечит конфиденциальность между двумя процессами. В до­полнение к конфиденциальности транспортный протокол может пред­лагать службы, обеспечивающие целостность данных, а также конечную аутентификацию.