L/R 秒的排队时延。我们将该例子中的计算平均排队时延的问题留给读者作为练习。

以上描述周期性到达的两个例子有些学术味。通常，到达队列的过程是随机的，即到达并不遵循任何模式，分组之间的时间间隔是随机的。在这种更为真实的情况下，量 La/R 通常不足以全面地表征时延的统计量。不过，直观地理解排队时延的范围很有用。特别是，如果流量强度接近于0，则几乎没有分组到达并且到达间隔很大，那么到达的分组将不可能在队列中发现别的分组。因此，平均排队时延将接近0。另一方面，当流量强度接近1时，当到达速率超过传输能力（由于分组到达速率的波动）时将存在时间间隔，在这些时段中将形成队列。当到达速率小于传输能力时，队列的长度将缩短。无论如何，随着流量强度接近1，平均排队长度变得越来越长。平均排队时延与流量强度的定性关系如图1-18所示。

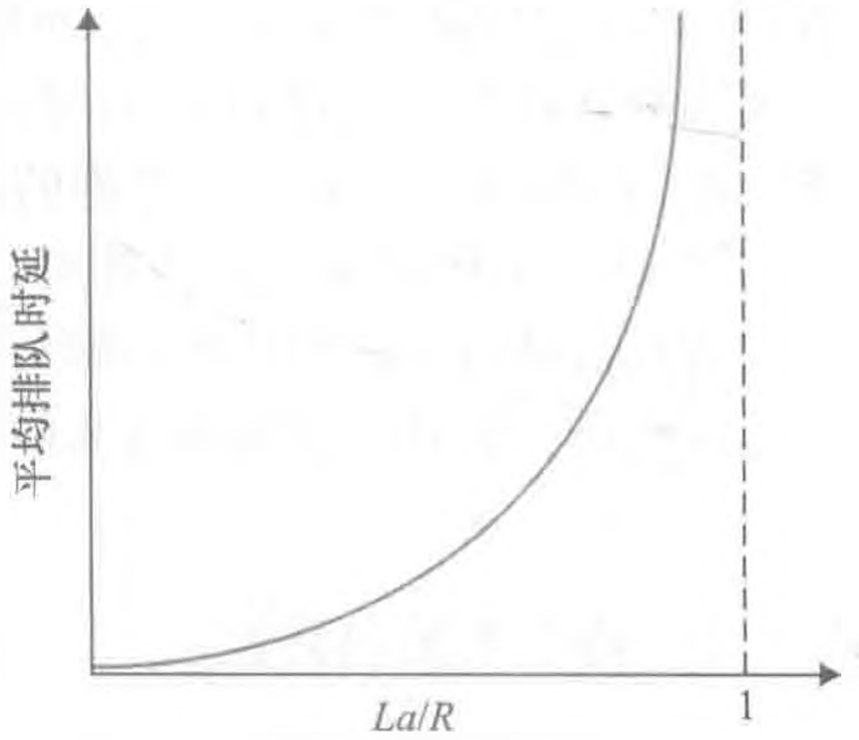


图1-18　平均排队时延与流量强度的关系

图1-18的一个重要方面是这样一个事实：随着流量强度接近于1，平均排队时延迅速增加。该强度的少量增加将导致时延大比例增加。也许你在公路上经历过这种事。如果在经常拥塞的公路上像平时一样驾驶，这条路经常拥塞的事实意味着它的流量强度接近于1，如果某些事件引起一个即便是稍微大于平常量的流量，经受的时延就可能很大。

为了实际感受一下排队时延的情况，我们再次鼓励你访问本书的Web网站，该网站提供了一个有关队列的交互式Java小程序。如果你将分组到达速率设置得足够大，使流量强度超过1，那么将看到经过一段时间后，队列慢慢地建立起来。

丢包

在上述讨论中，我们已经假设队列能够容纳无穷多的分组。在现实中，一条链路前的队列只有有限的容量，尽管排队容量极大地依赖于路由器设计和成本。因为该排队容量是有限的，随着流量强度接近1，排队时延并不真正趋向无穷大。相反，到达的分组将发现一个满的队列。由于没有地方存储这个分组，路由器将**丢弃**（drop）该分组，即该分组将会**丢失**（lost）。当流量强度大于1时，队列中的这种溢出也能够在用于队列的Java小程序中看到。

从端系统的角度看，上述丢包现象看起来是一个分组已经传输到网络核心，但它绝不会从网络发送到目的地。分组丢失的比例随着流量强度增加而增加。因此，一个节点的性能常常不仅根据时延来度量，而且根据丢包的概率来度量。正如我们将在后面各章中讨论的那样，丢失的分组可能基于端到端的原则重传，以确保所有的数据最终从源传送到了目的地。

1.4.3　端到端时延

前面的讨论一直集中在节点时延上，即在单台路由器上的时延。我们现在考虑从源到目的地的总时延。为了能够理解这个概念，假定在源主机和目的主机之间有 $N-1$ 台路由器。我们还要假设该网络此时是无拥塞的（因此排队时延是微不足道的），在每台路由器和源主机上的处理时延是 d_{proc}，每台路由器和源主机的输出速率是 R bps，每条链路的传

播时延是 d_{prop}。节点时延累加起来，得到端到端时延：

$$d_{end-end} = N(d_{proc} + d_{trans} + d_{prop}) \tag{1-2}$$

同样，式中 $d_{trans} = L/R$，其中 L 是分组长度。值得注意的是，式（1-2）是式（1-1）的一般形式，式（1-1）没有考虑处理时延和传播时延。在各节点具有不同的时延和每个节点存在平均排队时延的情况下，需要对式（1-2）进行一般化处理。我们将有关工作留给读者。

1. Traceroute

为了对计算机网络中的端到端时延有第一手认识，我们可以利用 Traceroute 程序。Traceroute 是一个简单的程序，它能够在任何因特网主机上运行。当用户指定一个目的主机名字时，源主机中的该程序朝着目的地发送多个特殊的分组。当这些分组向着目的地传送时，它们通过一系列路由器。当路由器接收到这些特殊分组之一时，它向源回送一个短报文。该短报文包括路由器的名字和地址。

更具体的是，假定在源和目的地之间有 $N-1$ 台路由器。源将向网络发送 N 个特殊的分组，其中每个分组地址指向最终目的地。这 N 个特殊分组标识为从 1 到 N，第一个分组标识为 1，最后的分组标识为 N。当第 n 台路由器接收到标识为 n 的第 n 个分组时，该路由器不是向它的目的地转发该分组，而是向源回送一个报文。当目的主机接收第 N 个分组时，它也会向源返回一个报文。该源记录了从它发送一个分组到它接收到对应返回报文所经历的时间；它也记录了返回该报文的路由器（或目的主机）的名字和地址。以这种方式，源能够重建分组从源到目的地所采用的路由，并且该源能够确定到所有中间路由器的往返时延。Traceroute 实际上对刚才描述的实验重复了 3 次，因此该源实际上向目的地发送了 $3 \times N$ 个分组。RFC 1393 详细地描述了 Traceroute。

这里有一个 Traceroute 程序输出的例子，其中追踪的路由从源主机 gaia. cs. umass. edu（位于马萨诸塞大学）到 cis. poly. edu（位于布鲁克林的理工大学）。输出有 6 列：第一列是前面描述的 n 值，即路径上的路由器编号；第二列是路由器的名字；第三列是路由器地址（格式为 xxx. xxx. xxx. xxx）；最后 3 列是 3 次实验的往返时延。如果源从任何给定路由器接收到的报文少于 3 条（由于网络中的丢包），Traceroute 在该路由器号码后面放一个星号，并向那台路由器报告少于 3 次往返时间。

```
1  cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2  128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3  -border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4  -acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5  -agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6  -acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7  -pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8  -gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9  -p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

在上述跟踪中，在源和目的之间有 9 台路由器。这些路由器中的多数有一个名字，所有都有地址。例如，路由器 3 的名字是 border4- rt- gi- 1- 3. gw. umass. edu，它的地址是 128. 119. 2. 194。看看为这台路由器提供的数据，可以看到在源和路由器之间的往返时延：3 次实验中的第一次是 1. 03ms，后继两次实验的往返时延是 0. 48ms 和 0. 45ms。这些往返时延包括刚才讨论的所有时延，即包括传输时延、传播时延、路由器处理时延和排队时延。因为该排队时延随时间变化，所以分组 n 发送到路由器 n 的往返时延实际上可能比分

组$n+1$发送到路由器$n+1$的往返时延更长。的确，我们在上述例子中观察到了这种现象：到路由器6的时延比到路由器7的更大！

你想自己试试Traceroute程序吗？我们极力推荐你访问http://www.traceroute.org，它的Web界面提供了有关路由跟踪的大量源列表。你选择一个源，并提供任何目的地的主机名，该Traceroute程序则会完成所有工作。有许多为Traceroute提供图形化界面的免费软件，其中我们喜爱的一个程序是PingPlotter[PingPlotter 2016]。

2. 端系统、应用程序和其他时延

除了处理时延、传输时延和传播时延，端系统中还有其他一些重要时延。例如，希望向共享媒体（例如在WiFi或电缆调制解调器情况下）传输分组的端系统可能有意地延迟它的传输，把这作为它与其他端系统共享媒体的协议的一部分；我们将在第6章中详细地考虑这样的协议。另一个重要的时延是媒体分组化时延，这种时延出现在IP语音（VoIP）应用中。在VoIP中，发送方在向因特网传递分组之前必须首先用编码的数字化语音填充一个分组。这种填充一个分组的时间称为分组化时延，它可能较大，并能够影响用户感受到的VoIP呼叫的质量。这个问题将在本章结束的课后作业中进一步探讨。

1.4.4　计算机网络中的吞吐量

除了时延和丢包，计算机网络中另一个至关重要的性能测度是端到端吞吐量。为了定义吞吐量，考虑从主机A到主机B跨越计算机网络传送一个大文件。例如，也许是从一个P2P文件共享系统中的一个对等方向另一个对等方传送一个大视频片段。在任何时间瞬间的**瞬时吞吐量**（instantaneous throughput）是主机B接收到该文件的速率（以bps计）。（许多应用程序包括许多P2P文件共享系统，其用户界面显示了下载期间的瞬时吞吐量，也许你以前已经观察过它！）如果该文件由F比特组成，主机B接收到所有F比特用去T秒，则文件传送的**平均吞吐量**（average throughput）是F/T bps。对于某些应用程序如因特网电话，希望具有低时延和在某个阈值之上（例如，对某些因特网电话是超过24kbps，对某些实时视频应用程序是超过256kbps）的一致的瞬时吞吐量。对于其他应用程序，包括涉及文件传送的那些应用程序，时延不是决定性的，但是希望具有尽可能高的吞吐量。

为了进一步深入理解吞吐量这个重要概念，我们考虑几个例子。图1-19a显示了服务器和客户这两个端系统，它们由两条通信链路和一台路由器相连。考虑从服务器传送一个文件到客户的吞吐量。令R_s表示服务器与路由器之间的链路速率；R_c表示路由器与客户之间的链路速率。假定在整个网络中只有从该服务器到客户的比特在传送。在这种理想的情况下，我们要问该服务器到客户的吞吐量是多少？为了回答这个问题，我们可以想象比特是流体，通信链路是管道。显然，这台服务器不能以快于R_s bps的速率通过其链路注入比特；这台路由器也不能以快于R_c bps的速率转发比特。如果$R_s < R_c$，则在给定的吞吐量R_s bps的情况下，由该服务器注入的比特将顺畅地通过路由器“流动”，并以速率R_s bps到达客户。另一方面，如果$R_c < R_s$，则该路由器将不能像接收速率那样快地转发比特。在这种情况下，比特将以速率R_c离开该路由器，从而得到端到端吞吐量R_c。（还要注意的是，如果比特继续以速率R_s到达路由器，继续以R_c离开路由器的话，在该路由器中等待传输给客户的积压比特将不断增加，这是一种最不希望的情况！）因此，对于这种简单的两链路网络，其吞吐量是$\min\{R_c, R_s\}$，这就是说，它是**瓶颈链路**（bottleneck link）的传输速率。在决定了吞吐量之后，我们现在近似地得到从服务器到

客户传输一个 F 比特的大文件所需要的时间是 $F/\min\{R_c, R_s\}$。举一个特定的例子，假定你正在下载一个 $F=32\times10^6$ 比特的 MP3 文件，服务器具有 $R_s=2$Mbps 的传输速率，并且你有一条 $R_c=1$Mbps 的接入链路。则传输该文件所需的时间是 32 秒。当然，这些吞吐量和传输时间的表达式仅是近似的，因为它们并没有考虑存储转发、处理时延和协议等问题。

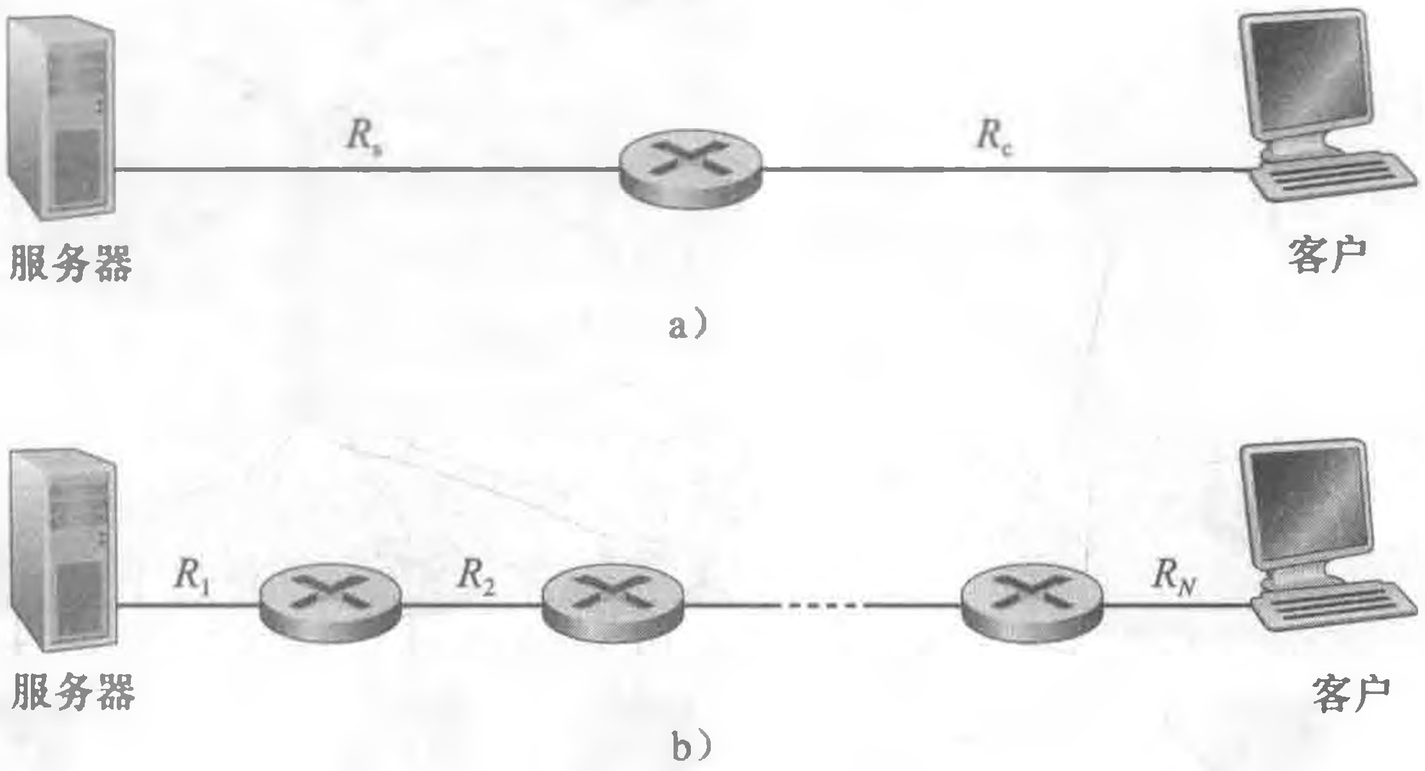
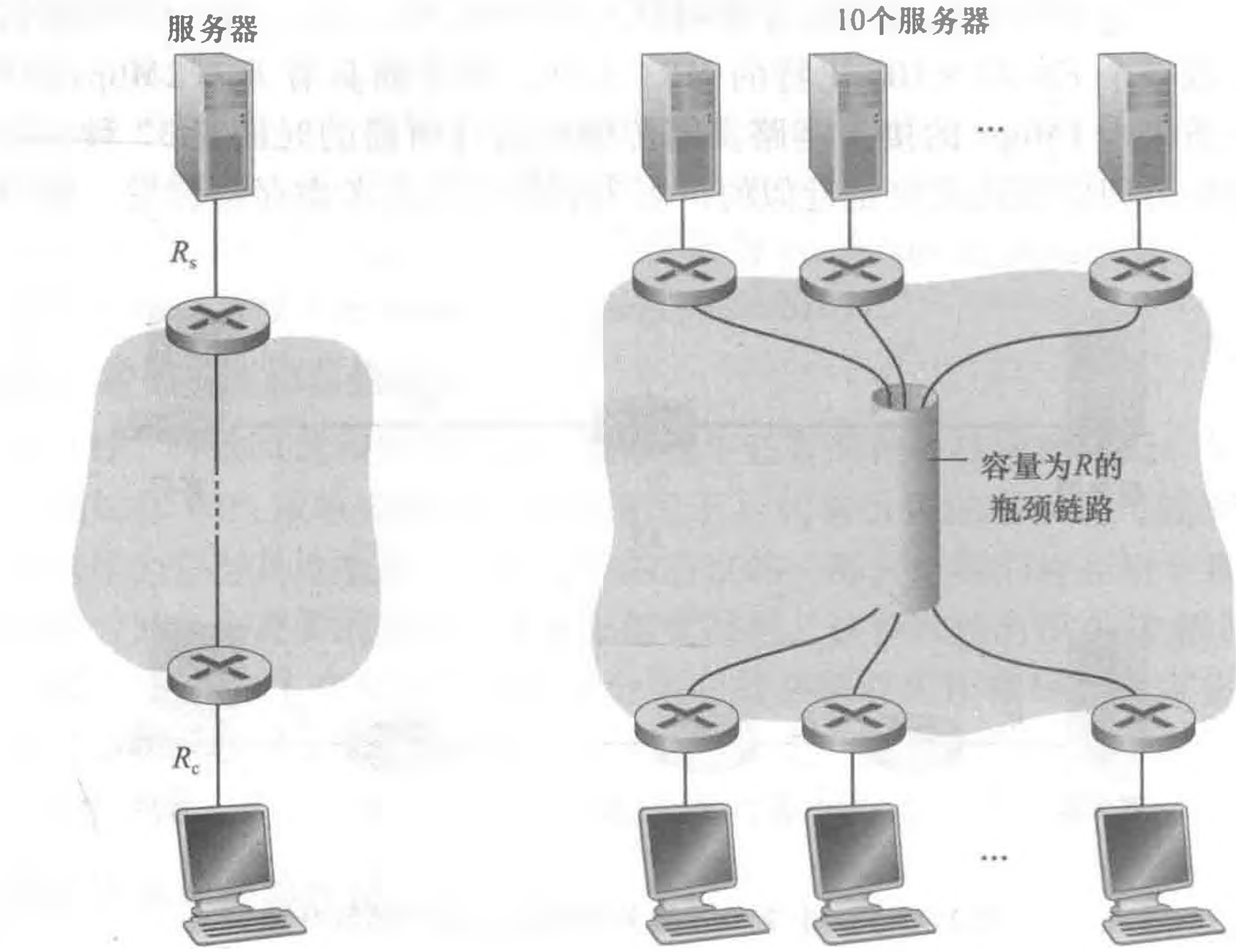


图 1-19　一个文件从服务器传送到客户的吞吐量

图 1-19b 此时显示了一个在服务器和客户之间具有 N 条链路的网络，这 N 条链路的传输速率分别是 R_1，R_2，…，R_N。应用对两条链路网络的分析方法，我们发现从服务器到客户的文件传输吞吐量是 $\min\{R_1, R_2, \cdots, R_N\}$，这同样仍是沿着服务器和客户之间路径的瓶颈链路的速率。

现在考虑由当前因特网所引发的另一个例子。图 1-20a 显示了与一个计算机网络相连的两个端系统：一台服务器和一个客户。考虑从服务器向客户传送一个文件的吞吐量。服务器以速率为 R_s 的接入链路与网络相连，且客户以速率为 R_c 的接入链路与网络相连。现在假定在通信网络核心中的所有链路具有非常高的传输速率，即该速率比 R_s 和 R_c 要高得多。目前因特网的核心的确超量配置了高速率的链路，从而很少出现拥塞。同时假定在整个网络中发送的比特都是从该服务器到该客户。在这个例子中，因为计算机网络的核心就像一个粗大的管子，所以比特从源向目的地的流动速率仍是 R_s 和 R_c 中的最小者，即吞吐量 $=\min\{R_s, R_c\}$。因此，在今天因特网中对吞吐量的限制因素通常是接入网。

作为最后一个例子，考虑图 1-20b，其中有 10 台服务器和 10 个客户与某计算机网络核心相连。在这个例子中，同时发生 10 个下载，涉及 10 个客户 - 服务器对。假定这 10 个下载是网络中当时的唯一流量。如该图所示，在核心中有一条所有 10 个下载通过的链路。将这条链路 R 的传输速率表示为 R。假定所有服务器接入链路具有相同的速率 R_s，所有客户接入链路具有相同的速率 R_c，并且核心中除了速率为 R 的一条共同链路之外的所有链路，它们的传输速率都比 R_s、R_c 和 R 大得多。现在我们要问，这种下载的吞吐量是多少？显然，如果该公共链路的速率 R 很大，比如说比 R_s 和 R_c 大 100 倍，则每个下载的吞吐量将仍然是 $\min\{R_s, R_c\}$。但是如果该公共链路的速率与 R_s 和 R_c 有相同量级会怎样呢？在这种情况下其吞吐量将是多少呢？让我们观察一个特定的例子。假定 $R_s=2$Mbps，$R_c=1$Mbps，$R=5$Mbps，并且公共链路为 10 个下载平等划分它的传输速率。这时每个下载的瓶颈不再位于接入网中，而是位于核心中的共享链路了，该瓶颈仅能为每个下载提供 500kbps 的吞吐量。因此每个下载的端到端吞吐量现在减少到 500kbps。



a）客户从服务器下载一个文件　　b）10个客户从10个服务器下载文件

图 1-20　端到端吞吐量

图 1-19 和图 1-20 中的例子说明吞吐量取决于数据流过的链路的传输速率。我们看到当没有其他干扰流量时，其吞吐量能够近似为沿着源和目的地之间路径的最小传输速率。图 1-20b 中的例子更一般地说明了吞吐量不仅取决于沿着路径的传输速率，而且取决于干扰流量。特别是，如果许多其他的数据流也通过这条链路流动，一条具有高传输速率的链路仍然可能成为文件传输的瓶颈链路。我们将在课后习题中和后继章节中更仔细地研究计算机网络中的吞吐量。

1.5　协议层次及其服务模型

从我们到目前的讨论来看，因特网显然是一个极为复杂的系统。我们已经看到，因特网有许多部分：大量的应用程序和协议、各种类型的端系统、分组交换机以及各种类型的链路级媒体。面对这种巨大的复杂性，存在着组织网络体系结构的希望吗？或者至少存在着我们对网络体系结构进行讨论的希望吗？幸运的是，对这两个问题的回答都是肯定的。

1.5.1　分层的体系结构

在试图组织我们关于因特网体系结构的想法之前，先看一个人类社会与之类比的例子。实际上，在日常生活中我们一直都与复杂系统打交道。想象一下有人请你描述比如航线系统的情况吧。你怎样用一个结构来描述这样一个复杂的系统？该系统具有票务代理、行李检查、登机口人员、飞行员、飞机、空中航行控制和世界范围的导航系统。描述这种系统的一种方式是，描述当你乘某个航班时，你（或其他人替你）要采取的一系列动作。你要购买机票，托运行李，去登机口，并最终登上这次航班。该飞机起飞，飞行到目的

地。当飞机着陆后，你从登机口离机并认领行李。如果这次行程不理想，你向票务机构投诉这次航班（你的努力一无所获）。图 1-21 显示了相关情况。

图 1-21　乘飞机旅行的一系列动作

我们已经能从这里看出与计算机网络的某些类似：航空公司把你从源送到目的地；而分组被从因特网中的源主机送到目的主机。但这不是我们寻求的完全的类似。我们在图 1-21 中寻找某些结构。观察图 1-21，我们注意到在每一端都有票务功能；还对已经检票的乘客有托运行李功能，对已经检票并已经检查过行李的乘客有登机功能。对于那些已经通过登机口的乘客（即已经经过检票、行李检查和通过登机口的乘客），有起飞和着陆的功能，并且在飞行中，有飞机按预定路线飞行的功能。这提示我们能够以水平的方式看待这些功能，如图 1-22 所示。

图 1-22 将航线功能划分为一些层次，提供了我们能够讨论航线旅行的框架。注意到每个层次与其下面的层次结合在一起，实现了某些功能、服务。在票务层及以下，完成了一个人从航线柜台到航线柜台的转移。在行李层及以下，完成了人和行李从行李托运到行李认领的转移。注意到行李层仅对已经完成票务的人提供服务。在登机口层，完成了人和行李从离港登机口到到港登机口的转移。在起飞/着陆层，完成了一个人和手提行李从跑道到跑道的转移。每个层次通过以下方式提供服务：①在这层中执行了某些动作（例如，在登机口层，某飞机的乘客登机和离机）；②使用直接下层的服务（例如，在登机口层，使用起飞/着陆层的跑道到跑道的旅客转移服务）。

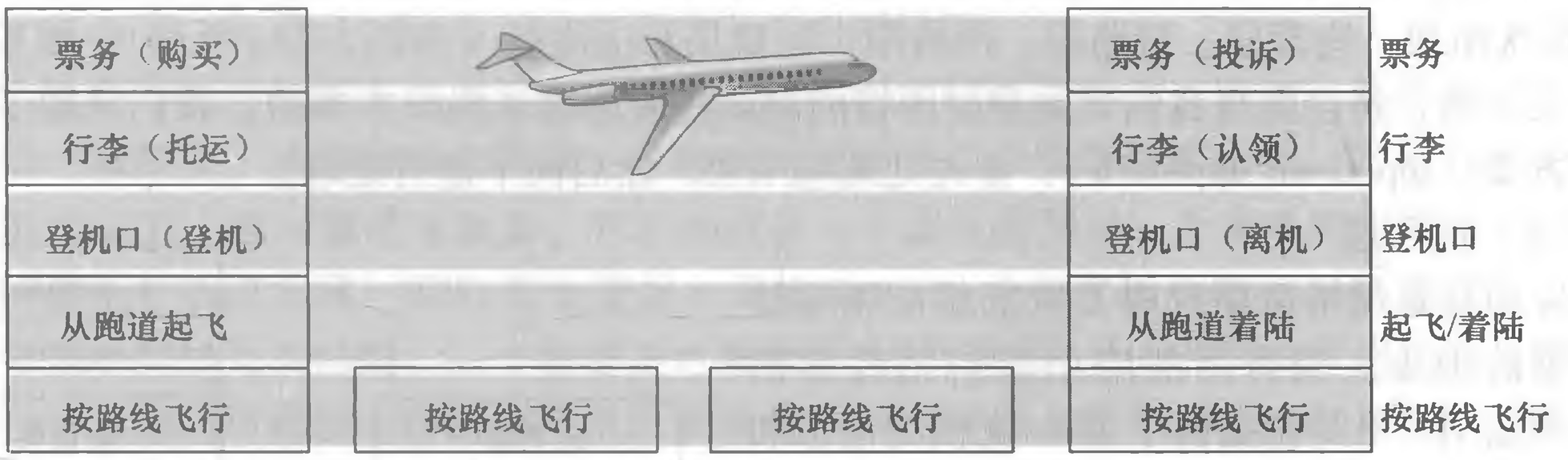


图 1-22　航线功能的水平分层

利用分层的体系结构，我们可以讨论一个大而复杂系统的定义良好的特定部分。这种简化本身由于提供模块化而具有很高价值，这使某层所提供的服务实现易于改变。只要该层对其上面的层提供相同的服务，并且使用来自下面层次的相同服务，当某层的实现变化时，该系统的其余部分保持不变。（注意到改变一个服务的实现与改变服务本身是极为不同的！）例如，如果登机口功能被改变了（例如让人们按身高登机和离机），航线系统的其余部分将保持不变，因为登机口仍然提供相同的功能（人们登机和离机）；改变后，它仅是以不同的方式实现了该功能。对于大而复杂且需要不断更新的系统，改变服务的实现而不影响该系统其他组件是分层的另一个重要优点。

1. 协议分层

我们对航线已经进行了充分讨论，现将注意力转向网络协议。为了给网络协议的设计提供一个结构，网络设计者以**分层**（layer）的方式组织协议以及实现这些协议的网络硬件和软件。每个协议属于这些层次之一，就像图 1-22 所示的航线体系结构中的每种功能属于某一层一样。我们再次关注某层向它的上一层提供的**服务**（service），即所谓一层的**服务模型**（service model）。就像前面航线例子中的情况一样，每层通过在该层中执行某些动作或使用直接下层的服务来提供服务。例如，由第 n 层提供的服务可能包括报文从网络的一边到另一边的可靠交付。这可能是通过使用第 $n-1$ 层的边缘到边缘的不可靠报文传送服务，加上第 n 层的检测和重传丢失报文的功能来实现的。

一个协议层能够用软件、硬件或两者的结合来实现。诸如 HTTP 和 SMTP 这样的应用层协议几乎总是在端系统中用软件实现，运输层协议也是如此。因为物理层和数据链路层负责处理跨越特定链路的通信，它们通常在与给定链路相关联的网络接口卡（例如以太网或 WiFi 接口卡）中实现。网络层经常是硬件和软件实现的混合体。还要注意的是，如同分层的航线体系结构中的功能分布在构成该系统的各机场和飞行控制中心中一样，一个第 n 层协议也分布在构成该网络的端系统、分组交换机和其他组件中。这就是说，第 n 层协议的不同部分常常位于这些网络组件的各部分中。

协议分层具有概念化和结构化的优点［RFC 3439］。如我们看到的那样，分层提供了一种结构化方式来讨论系统组件。模块化使更新系统组件更为容易。然而，需要提及的是，某些研究人员和联网工程师激烈地反对分层［Wakeman 1992］。分层的一个潜在缺点是一层可能冗余较低层的功能。例如，许多协议栈在基于每段链路和基于端到端两种情况下，都提供了差错恢复。第二种潜在的缺点是某层的功能可能需要仅在其他某层才出现的信息（如时间戳值），这违反了层次分离的目标。

将这些综合起来，各层的所有协议被称为**协议栈**（protocol stack）。因特网的协议栈由 5 个层次组成：物理层、链路层、网络层、运输层和应用层（如图 1-23a 所示）。如果你查看本书目录，将会发现我们大致是以因特网协议栈的层次来组织本书的。我们采用了**自顶向下方法**（top-down approach），首先处理应用层，然后向下进行处理。

（1）应用层

应用层是网络应用程序及它们的应用层协议存留的地方。因特网的应用层包括许多协议，例如 HTTP（它提供了 Web 文档的请求和传送）、SMTP（它提供了电子邮件报文的传输）和 FTP（它提供两个端系统之间的文件传送）。我们将看到，某些网络功能，如将像 www. ietf. org 这样对人友好的端系统名字转换为 32 比特的网络地址，也是借助于特定的应用层协议即域名系统（DNS）完成的。我们将在第 2 章中看到，创建并部署我们自己的新应用层协议是非常容易的。

a）5层因特网协议栈　　b）7层ISO OSI参考模型

图 1-23　因特网协议栈和 OSI 参考模型

应用层协议分布在多个端系统上，而一个端系统中的应用程序使用协议与另一个端系统中的应用程序交换信息分组。我们把这种位于应用层的信息分组称为**报文**（message）。

（2）运输层

因特网的运输层在应用程序端点之间传送应用层报文。在因特网中，有两种运输协议，即 TCP 和 UDP，利用其中的任一个都能运输应用层报文。TCP 向它的应用程序提供了面向连接的服务。这种服务包括了应用层报文向目的地的确保传递和流量控制（即发送方/接收方速率匹配）。TCP 也将长报文划分为短报文，并提供拥塞控制机制，因此当网络拥塞时，源抑制其传输速率。UDP 协议向它的应用程序提供无连接服务。这是一种不提供不必要服务的服务，没有可靠性，没有流量控制，也没有拥塞控制。在本书中，我们把运输层的分组称为**报文段**（segment）。

（3）网络层

因特网的网络层负责将称为**数据报**（datagram）的网络层分组从一台主机移动到另一台主机。在一台源主机中的因特网运输层协议（TCP 或 UDP）向网络层递交运输层报文段和目的地址，就像你通过邮政服务寄信件时提供一个目的地址一样。

因特网的网络层包括著名的网际协议 IP，该协议定义了在数据报中的各个字段以及端系统和路由器如何作用于这些字段。IP 仅有一个，所有具有网络层的因特网组件必须运行 IP。因特网的网络层也包括决定路由的路由选择协议，它根据该路由将数据报从源传输到目的地。因特网具有许多路由选择协议。如我们在 1.3 节所见，因特网是一个网络的网络，并且在一个网络中，其网络管理者能够运行所希望的任何路由选择协议。尽管网络层包括了网际协议和一些路由选择协议，但通常把它简单地称为 IP 层，这反映了 IP 是将因特网连接在一起的黏合剂这样的事实。

（4）链路层

因特网的网络层通过源和目的地之间的一系列路由器路由数据报。为了将分组从一个节点（主机或路由器）移动到路径上的下一个节点，网络层必须依靠该链路层的服务。特别是在每个节点，网络层将数据报下传给链路层，链路层沿着路径将数据报传递给下一个节点。在该下一个节点，链路层将数据报上传给网络层。

由链路层提供的服务取决于应用于该链路的特定链路层协议。例如，某些协议基于链路提供可靠传递，从传输节点跨越一条链路到接收节点。值得注意的是，这种可靠的传递服务不同于 TCP 的可靠传递服务，TCP 提供从一个端系统到另一个端系统的可靠交付。链路层的例子包括以太网、WiFi 和电缆接入网的 DOCSIS 协议。因为数据报从源到目的地传送通常需要经过几条链路，一个数据报可能被沿途不同链路上的不同链路层协议处理。例如，一个数据报可能被一段链路上的以太网和下一段链路上的 PPP 所处理。网络层将受到来自每个不同的链路层协议的不同服务。在本书中，我们把链路层分组称为**帧**（frame）。

（5）物理层

虽然链路层的任务是将整个帧从一个网络元素移动到邻近的网络元素，而物理层的任务是将该帧中的一个个比特从一个节点移动到下一个节点。在这层中的协议仍然是链路相关的，并且进一步与该链路（例如，双绞铜线、单模光纤）的实际传输媒体相关。例如，以太网具有许多物理层协议：一个是关于双绞铜线的，另一个是关于同轴电缆的，还有一个是关于光纤的，等等。在每种场合中，跨越这些链路移动一个比特是以不同的方式进行的。

2. OSI 模型

详细地讨论过因特网协议栈后，我们应当提及它不是唯一的协议栈。特别是在 20 世纪 70 年代后期，国际标准化组织（ISO）提出计算机网络围绕 7 层来组织，称为开放系统

互连（OSI）模型［ISO 2016］。当那些要成为因特网协议的协议还处于襁褓之中，只是许多正在研发的不同协议族之一时，OSI 模型就已经成形。事实上，初始 OSI 模型的发明者在创建该模型时心中可能并没有想到因特网。无论如何，自 20 世纪 70 年代后期开始，许多培训课程和大学课程都围绕 7 层模型挑选有关 ISO 授权和组织的课程。因为它在网络教育的早期影响，该 7 层模型继续以某种方式存留在某些网络教科书和培训课程中。

显示在图 1-23b 中的 OSI 参考模型的 7 层是：应用层、表示层、会话层、运输层、网络层、数据链路层和物理层。这些层次中，5 层的功能大致与它们名字类似的因特网对应层的功能相同。所以，我们来考虑 OSI 参考模型中附加的两个层，即表示层和会话层。表示层的作用是使通信的应用程序能够解释交换数据的含义。这些服务包括数据压缩和数据加密（它们是自解释的）以及数据描述（这使得应用程序不必担心在各台计算机中表示/存储的内部格式不同的问题）。会话层提供了数据交换的定界和同步功能，包括了建立检查点和恢复方案的方法。

因特网缺少了在 OSI 参考模型中建立的两个层次，该事实引起了一些有趣的问题：这些层次提供的服务不重要吗？如果一个应用程序需要这些服务之一，将会怎样呢？因特网对这两个问题的回答是相同的：这留给应用程序开发者处理。应用程序开发者决定一个服务是否是重要的，如果该服务重要，应用程序开发者就应该在应用程序中构建该功能。

1.5.2　封装

图 1-24 显示了这样一条物理路径：数据从发送端系统的协议栈向下，沿着中间的链路层交换机和路由器的协议栈上上下下，然后向上到达接收端系统的协议栈。如我们将在本书后面讨论的那样，路由器和链路层交换机都是分组交换机。与端系统类似，路由器和链路层交换机以多层次的方式组织它们的网络硬件和软件。而路由器和链路层交换机并不实现协议栈中的所有层次。如图 1-24 所示，链路层交换机实现了第一层和第二层；路由器实现了第一层到第三层。例如，这意味着因特网路由器能够实现 IP 协议（一种第三层协议），而链路层交换机则不能。我们将在后面看到，尽管链路层交换机不能识别 IP 地址，但它们能够识别第二层地址，如以太网地址。值得注意的是，主机实现了所有 5 个层次，这与因特网体系结构将它的复杂性放在网络边缘的观点是一致的。

图 1-24 也说明了一个重要概念：**封装**（encapsulation）。在发送主机端，一个**应用层报文**（application-layer message）（图 1-24 中的 M）被传送给运输层。在最简单的情况下，运输层收取到报文并附上附加信息（所谓运输层首部信息，图 1-24 中的 H_t），该首部将被接收端的运输层使用。应用层报文和运输层首部信息一道构成了**运输层报文段**（transport-layer segment）。运输层报文段因此封装了应用层报文。附加的信息也许包括了下列信息：允许接收端运输层向上向适当的应用程序交付报文的信息；差错检测位信息，该信息让接收方能够判断报文中的比特是否在途中已被改变。运输层则向网络层传递该报文段，网络层增加了如源和目的端系统地址等网络层首部信息（图 1-24 中的 H_n），生成了**网络层数据报**（network-layer datagram）。该数据报接下来被传递给链路层，链路层（自然而然地）增加它自己的链路层首部信息并生成**链路层帧**（link-layer frame）。所以我们看到，在每一层，一个分组具有两种类型的字段：首部字段和**有效载荷字段**（payload field）。有效载荷通常是来自上一层的分组。

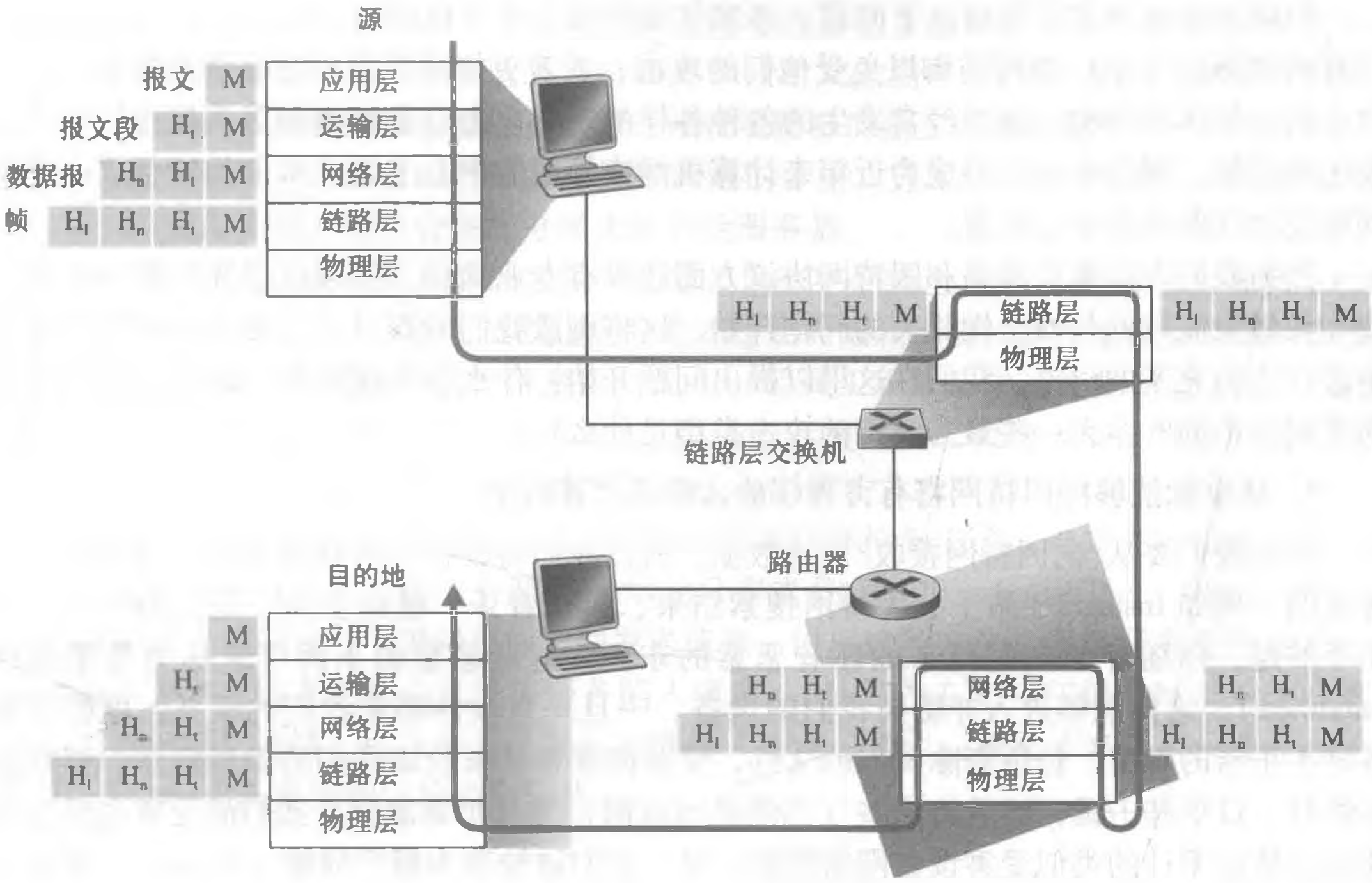


图 1-24 主机、路由器和链路层交换机，每个包含了不同的层，反映了它们的功能差异

这里一个有用的类比是经过公共邮政服务在某公司办事处之间发送一封备忘录。假定位于某办事处的 Alice 要向位于另一办事处的 Bob 发送一封备忘录。该备忘录类比于应用层报文。Alice 将备忘录放入办事处之间的公函信封中，并在公函信封上方写上了 Bob 的名字和部门。该办事处之间的公函信封类比于运输层报文段，即它包括了首部信息（Bob 的名字和部门编号）并封装了应用层报文（备忘录）。当发送办事处的收发室拿到该办事处之间的备忘录时，将其放入适合在公共邮政服务发送的信封中，并在邮政信封上写上发送和接收办事处的邮政地址。此处，邮政信封类比于数据报，它封装了运输层的报文段（办事处之间的公函信封），该报文段封装了初始报文（备忘录）。邮政服务将该邮政信封交付给接收办事处的收发室。在此处开始了拆封过程。该收发室取出了办事处之间的公函信封并转发给 Bob。最后，Bob 打开信封并拿走了备忘录。

封装的过程能够比前面描述的更为复杂。例如，一个大报文可能被划分为多个运输层的报文段（这些报文段每个可能被划分为多个网络层数据报）。在接收端，则必须从其连续的数据报中重构这样一个报文段。

1.6 面对攻击的网络

对于今天的许多机构（包括大大小小的公司、大学和政府机关）而言，因特网已经成为与其使命密切相关的一部分了。许多人也依赖因特网从事各种职业、社会和个人活动。目前，数以亿计的物品（包括可穿戴设备和家用设备）与因特网相连。但是在所有这一切背后，存在着一个阴暗面，其中的“坏家伙”试图对我们的日常生活进行破坏，如损坏我们与因特网相连的计算机，侵犯我们的隐私以及使我们依赖的因特网服务无法运行。

网络安全领域主要探讨以下问题：坏家伙如何攻击计算机网络，以及我们（即将成为计算机网络的专家）如何防御以免受他们的攻击，或者更好的是设计能够事先免除这样的攻击的新型体系结构。面对经常发生的各种各样的现有攻击以及新型和更具摧毁性的未来攻击的威胁，网络安全已经成为近年来计算机网络领域的中心主题。本书的特色之一是将网络安全问题放在中心位置。

因为我们在计算机网络和因特网协议方面还没有专业知识，所以这里我们将从审视某些今天最为流行的与安全性相关的问题开始。这将刺激我们的胃口，以便我们在后续章节中进行更为充实的讨论。我们在这里以提出问题开始：什么会出现问题？计算机网络是如何受到攻击的？今天一些最为流行的攻击类型是什么？

1. 坏家伙能够经因特网将有害程序放入你的计算机中

因为我们要从/向因特网接收/发送数据，所以我们将设备与因特网相连。这包括各种好东西，例如 Instagram 帖子、因特网搜索结果、流式音乐、视频会议、流式电影等。但不幸的是，伴随好的东西而来的还有恶意的东西，这些恶意的东西可统称为**恶意软件**（malware），它们能够进入并感染我们的设备。一旦恶意软件感染我们的设备，就能够做各种不正当的事情，包括删除我们的文件，安装间谍软件来收集我们的隐私信息，如社会保险号、口令和击键，然后将这些（当然经因特网）发送给坏家伙。我们的受害主机也可能成为数以千计的类似受害设备网络中的一员，它们被统称为**僵尸网络**（botnet），坏家伙利用僵尸网络控制并有效地对目标主机展开垃圾邮件分发或分布式拒绝服务攻击（很快将讨论）。

至今为止的多数恶意软件是**自我复制**（self-replicating）的：一旦它感染了一台主机，就会从那台主机寻求进入因特网上的其他主机，从而形成新的感染主机，再寻求进入更多的主机。以这种方式，自我复制的恶意软件能够指数式地快速扩散。恶意软件能够以病毒或蠕虫的形式扩散。**病毒**（virus）是一种需要某种形式的用户交互来感染用户设备的恶意软件。典型的例子是包含恶意可执行代码的电子邮件附件。如果用户接收并打开这样的附件，不经意间就在其设备上运行了该恶意软件。通常，这种电子邮件病毒是自我复制的：例如，一旦执行，该病毒可能向用户地址簿上的每个接收方发送一个具有相同恶意附件的相同报文。**蠕虫**（worm）是一种无须任何明显用户交互就能进入设备的恶意软件。例如，用户也许运行了一个攻击者能够发送恶意软件的脆弱网络应用程序。在某些情况下，没有用户的任何干预，该应用程序可能从因特网接收恶意软件并运行它，生成了蠕虫。新近感染设备中的蠕虫则能扫描因特网，搜索其他运行相同网络应用程序的易受感染的主机。当它发现其他易受感染的主机时，便向这些主机发送一个它自身的副本。今天，恶意软件无所不在且防范成本高。当你用这本书学习时，我们鼓励你思考下列问题：计算机网络设计者能够采取什么防御措施，以使与因特网连接的设备免受恶意软件的攻击？

2. 坏家伙能够攻击服务器和网络基础设施

另一种宽泛类型的安全性威胁称为**拒绝服务攻击**（Denial-of-Service（DoS）attack）。顾名思义，DoS 攻击使得网络、主机或其他基础设施部分不能由合法用户使用。Web 服务器、电子邮件服务器、DNS 服务器（在第 2 章中讨论）和机构网络都能够成为 DoS 攻击的目标。因特网 DoS 攻击极为常见，每年会出现数以千计的 DoS 攻击［Moore 2001］。访问数字攻击图（Digital Attack Map）站点可以观看世界范围内每天最厉害的 DoS 攻击［DAM 2016］。大多数因特网 DoS 攻击属于下列三种类型之一：

- 弱点攻击。这涉及向一台目标主机上运行的易受攻击的应用程序或操作系统发送制作精细的报文。如果适当顺序的多个分组发送给一个易受攻击的应用程序或操作系统，该服务器可能停止运行，或者更糟糕的是主机可能崩溃。
- 带宽洪泛。攻击者向目标主机发送大量的分组，分组数量之多使得目标的接入链路变得拥塞，使得合法的分组无法到达服务器。
- 连接洪泛。攻击者在目标主机中创建大量的半开或全开 TCP 连接（将在第 3 章中讨论 TCP 连接）。该主机因这些伪造的连接而陷入困境，并停止接受合法的连接。

我们现在更详细地研究这种带宽洪泛攻击。回顾 1.4.2 节中讨论的时延和丢包问题，显然，如果某服务器的接入速率为 R bps，则攻击者将需要以大约 R bps 的速率来产生危害。如果 R 非常大的话，单一攻击源可能无法产生足够大的流量来伤害该服务器。此外，如果从单一源发出所有流量的话，某上游路由器就能够检测出该攻击并在该流量靠近服务器之前就将其阻挡下来。在图 1-25 中显示的**分布式 DoS**（Distributed DoS，DDoS）中，攻击者控制多个源并让每个源向目标猛烈发送流量。使用这种方法，遍及所有受控源的聚合流量速率需要大约 R 的能力来使该服务陷入瘫痪。DDoS 攻击充分利用由数以千计的受害主机组成的僵尸网络，这在今天是屡见不鲜的［DAM 2016］。相比于来自单一主机的 DoS 攻击，DDoS 攻击更加难以检测和防范。

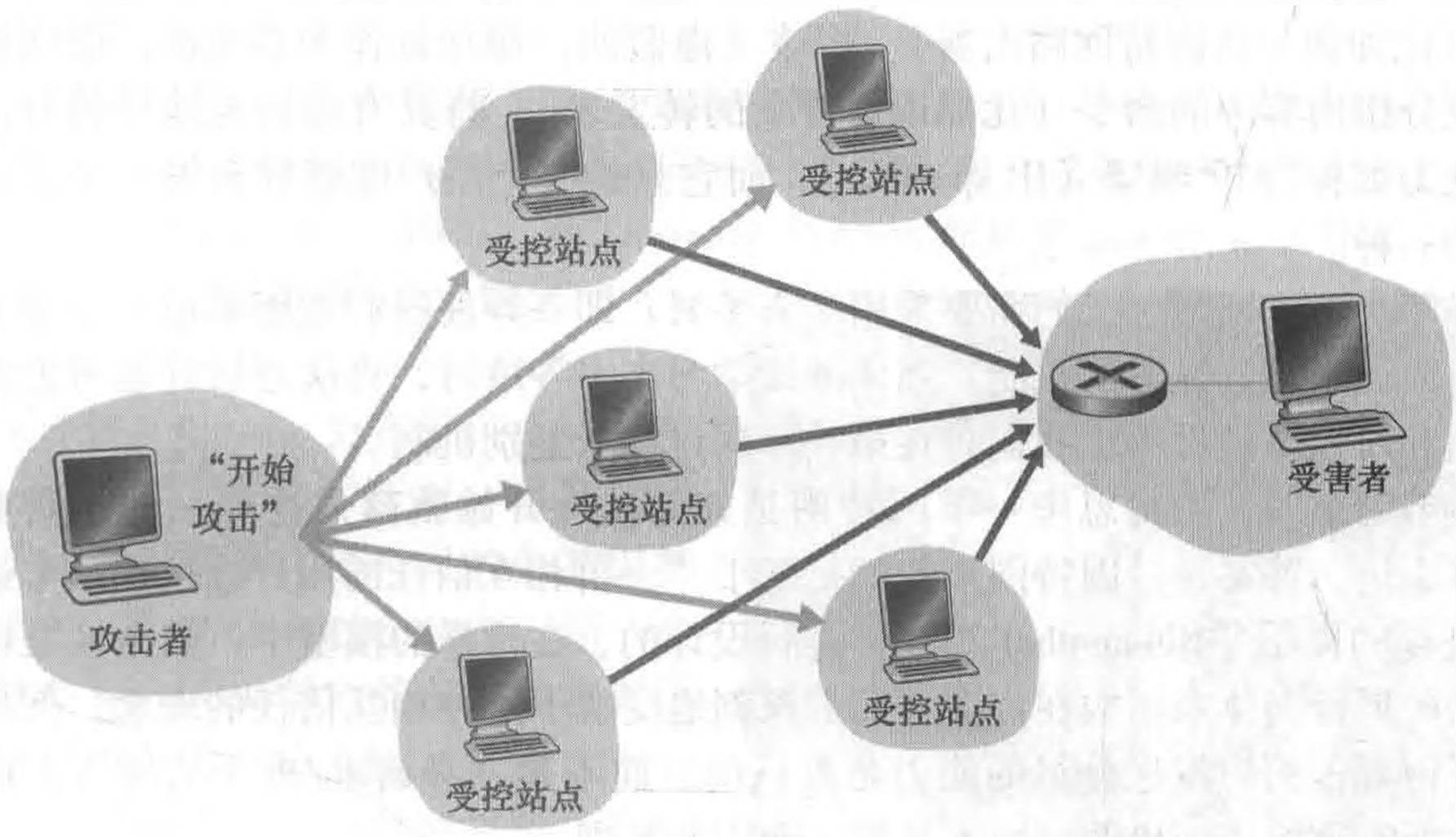


图 1-25　分布式拒绝服务攻击

当学习这本书时，我们鼓励你考虑下列问题：计算机网络设计者能够采取哪些措施防止 DoS 攻击？我们将看到，对于 3 种不同类型的 DoS 攻击需要采用不同的防御方法。

3. 坏家伙能够嗅探分组

今天的许多用户经无线设备接入因特网，如 WiFi 连接的膝上计算机或使用蜂窝因特网连接的手持设备（在第 7 章中讨论）。无所不在的因特网接入极为便利并让移动用户方便地使用令人惊奇的新应用程序的同时，也产生了严重的安全脆弱性——在无线传输设备的附近放置一台被动的接收机，该接收机就能得到传输的每个分组的副本！这些分组包含了各种敏感信息，包括口令、社会保险号、商业秘密和隐秘的个人信息。记录每个流经的分组副本的被动接收机被称为**分组嗅探器**（packet sniffer）。

嗅探器也能够部署在有线环境中。在有线的广播环境中，如在许多以太网 LAN 中，

分组嗅探器能够获得经该 LAN 发送的所有分组。如在 1.2 节中描述的那样，电缆接入技术也广播分组，因此易于受到嗅探攻击。此外，获得某机构与因特网连接的接入路由器或接入链路访问权的坏家伙能够放置一台嗅探器以产生从该机构出入的每个分组的副本，再对嗅探到的分组进行离线分析，就能得出敏感信息。

分组嗅探软件在各种 Web 站点上可免费得到，这类软件也有商用的产品。教网络课程的教授布置的实验作业就涉及写一个分组嗅探器和应用层数据重构程序。与本书相关联的 Wireshark［Wireshark 2016］实验（参见本章结尾处的 Wireshark 实验介绍）使用的正是这样一种分组嗅探器！

因为分组嗅探器是被动的，也就是说它们不向信道中注入分组，所以难以检测到它们。因此，当我们向无线信道发送分组时，我们必须接受这样的可能性，即某些坏家伙可能记录了我们的分组的副本。如你已经猜想的那样，最好的防御嗅探的方法基本上都与密码学有关。我们将在第 8 章研究密码学应用于网络安全的有关内容。

4. 坏家伙能够伪装成你信任的人

生成具有任意源地址、分组内容和目的地址的分组，然后将这个人工制作的分组传输到因特网中，因特网将忠实地将该分组转发到目的地，这一切都极为容易（当你学完这本教科书后，你将很快具有这方面的知识了！）。想象某个接收到这样一个分组的不会猜疑的接收方（比如说一台因特网路由器），将该（虚假的）源地址作为真实的，进而执行某些嵌入在该分组内容中的命令（比如说修改它的转发表）。将具有虚假源地址的分组注入因特网的能力被称为 **IP 哄骗**（IP spoofing），而它只是一个用户能够冒充另一个用户的许多方式中的一种。

为了解决这个问题，我们需要采用*端点鉴别*，即一种使我们能够确信一个报文源自我们认为它应当来自的地方的机制。当你继续学习本书各章时，再次建议你思考怎样为网络应用程序和协议做这件事。我们将在第 8 章探讨端点鉴别机制。

在本节结束时，值得思考一下因特网是如何从一开始就落入这样一种不安全的境地的。大体上讲，答案是：因特网最初就是基于“一群相互信任的用户连接到一个透明的网络上”这样的模型［Blumenthal 2001］进行设计的，在这样的模型中，安全性是没有必要的。初始的因特网体系结构在许多方面都深刻地反映了这种相互信任的理念。例如，一个用户向任何其他用户发送分组的能力是默认的，而不是一种请求/准予的能力，还有用户身份取自所宣称的表面价值，而不是默认地需要鉴别。

但是今天的因特网无疑并不涉及“相互信任的用户”。但是，今天的用户仍然需要通信，当他们不必相互信任时，他们也许希望匿名通信，也许间接地通过第三方通信（例如我们将在第 2 章中学习的 Web 高速缓存，我们将在第 7 章学习的移动性协助代理），也许不信任他们通信时使用的硬件、软件甚至空气。随着我们进一步学习本书，会面临许多安全性相关的挑战：我们应当寻求对嗅探、端点假冒、中间人攻击、DDoS 攻击、恶意软件等的防护办法。我们应当记住：在相互信任的用户之间的通信是一种例外而不是规则。欢迎你到现代计算机网络世界！

1.7　计算机网络和因特网的历史

1.1 节到 1.6 节概述了计算机网络和因特网的技术。你现在应当有足够的知识来给家人和朋友留下深刻印象了。然而，如果你真的想在下次鸡尾酒会上一鸣惊人，你应当在你

的演讲中点缀一些有关因特网引人入胜的历史轶闻［Segaller 1998］。

1.7.1　分组交换的发展：1961～1972

计算机网络和今天因特网领域的开端可以追溯到20世纪60年代早期，那时电话网是世界上占统治地位的通信网络。1.3节讲过，电话网使用电路交换将信息从发送方传输到接收方，这种适当的选择使得语音以一种恒定的速率在发送方和接收方之间传输。随着20世纪60年代早期计算机的重要性越来越大，以及分时计算机的出现，考虑如何将计算机连接在一起，并使它们能够被地理上分布的用户所共享的问题，也许就成了一件自然的事。这些用户所产生的流量很可能具有突发性，即活动的间断性，例如向远程计算机发送一个命令，接着是静止的时间段，这是等待应答或对接收到的响应进行思考的时间。

全世界有3个研究组首先发明了分组交换，以作为电路交换的一种有效的、健壮的替代技术。这3个研究组互不知道其他人的工作［Leiner 1998］。有关分组交换技术的首次公开发表出自Leonard Kleinrock［Kleinrock 1961；Kleinrock 1964］，那时他是麻省理工学院（MIT）的一名研究生。Kleinrock使用排队论，完美地体现了使用分组交换方法处理突发性流量源的有效性。1964年，兰德公司的Paul Baran［Baran 1964］已经开始研究分组交换的应用，以在军用网络上传输安全语音，同时在英国的国家物理实验室（NPL），Donald Davies和Roger Scantlebury也在研究分组交换技术。

MIT、兰德和NPL的工作奠定了今天的因特网的基础。但是因特网也经历了很长的“边构建边示范（let's-build-it-and-demonstrate-it）”的历史，这可追溯到20世纪60年代早期。J. C. R. Licklider［DEC 1990］和Lawrence Roberts都是Kleinrock在MIT的同事，他们转而去领导美国高级研究计划署（Advanced Research Projects Agency，ARPA）的计算机科学计划。Roberts公布了一个ARPAnet［Roberts 1967］的总体计划，它是第一个分组交换计算机网络，是今天的公共因特网的直接祖先。在1969年的劳动节，第一台分组交换机在Kleinrock的监管下安装在美国加州大学洛杉矶分校（UCLA），其他3台分组交换机不久后安装在斯坦福研究所（Stanford Research Institute，SRI）、美国加州大学圣巴巴拉分校（UC Santa Barbara）和犹他大学（University of Utah）（参见图1-26）。羽翼未丰的因特网祖先到1969年年底有了4个节点。Kleinrock回忆说，该网络的最先应用是从UCLA到SRI执行远程注册，但却导致了该系统的崩溃［Kleinrock 2004］。

图1-26　一台早期的分组交换机

到了1972年，ARPAnet已经成长到大约15个节点，由Robert Kahn首次对它进行了公开演示。在ARPAnet端系统之间的第一台主机到主机协议——称为网络控制协议

(NCP)，就是此时完成的［RFC 001］。随着端到端协议的可供使用，这时能够写应用程序了。在1972年，Ray Tomlinson编写了第一个电子邮件程序。

1.7.2　专用网络和网络互联：1972～1980

最初的ARPAnet是一个单一的、封闭的网络。为了与ARPAnet的一台主机通信，一台主机必须与另一台ARPAnet IMP实际相连。20世纪70年代早期和中期，除ARPAnet之外的其他分组交换网络问世：ALOHAnet是一个微波网络，它将夏威夷岛上的大学［Abramson 1970］以及DARPA的分组卫星［RFC 829］和分组无线电网［Kahn 1987］连接到一起；Telenet是BBN的商用分组交换网，它基于ARPAnet技术；由Louis Pouzin领衔的Cyclades是法国的一个分组交换网［Think 2012］；还有如Tymnet和GE信息服务网这样的分时网络，以及20世纪60年代后期和70年代初期的类似网络［Schwartz 1977］；IBM的SNA（1996～1974），它与ARPAnet同时在运行［Schwartz 1977］。

网络的数目开始增加。人们事后看到，研制将网络连接到一起的体系结构的时机已经成熟。互联网络的先驱性工作（得到了美国国防部高级研究计划署（DARPA）的支持）由Vinton Cerf和Robert Kahn［Cerf 1974］完成，本质上就是创建一个网络的网络；术语网络互联（internetting）就是用来描述该项工作的。

这些体系结构的原则体现在TCP中。然而，TCP的早期版本与今天的TCP差异很大。TCP的早期版本将通过端系统重传的可靠按序数据传递（仍是今天的TCP的一部分）与转发功能（今天该功能由IP执行）相结合。TCP的早期实验以及认识到对诸如分组语音这样的应用程序中不可靠的、非流控制的、端到端传递服务的重要性，导致IP从TCP中分离出来，并研制了UDP协议。我们今天看到的3个重要的因特网协议——TCP、UDP和IP，到20世纪70年代末在概念上已经完成。

除了DARPA的因特网相关研究外，许多其他重要的网络活动也在进行中。在夏威夷，Norman Abramson正在研制ALOHAnet，这是一个基于分组的无线电网络，它使在夏威夷岛上的多个远程站点互相通信。ALOHA协议［Abramson 1970］是第一个多路访问协议，允许地理上分布的用户共享单一的广播通信媒体（一个无线电频率）。Metcalfe和Boggs基于Abramson的多路访问协议，研制了用于有线共享广播网络的以太网协议［Metcalfe 1976］。令人感兴趣的是，Metcalfe和Boggs的以太网协议是由连接多台PC、打印机和共享磁盘在一起的需求所激励的［Perkins 1994］。在PC革命和网络爆炸的25年之前，Metcalfe和Boggs就奠定了今天PC LAN的基础。

1.7.3　网络的激增：1980～1990

到了20世纪70年代末，大约200台主机与ARPAnet相连。到了20世纪80年代末，连到公共因特网的主机数量达到100 000台，那时的公共因特网是网络的联盟，看起来非常像今天的因特网。20世纪80年代是联网主机数量急剧增长的时期。

这种增长是由几个显著成果即创建计算机网络将大学连接到一起引起的。BITNET为位于美国东北部的几个大学之间提供了电子邮件和文件传输。建立了CSNET（计算机科学网），以将还没有接入ARPAnet的大学研究人员连接在一起。1986年，建立了NSFNET，为NSF资助的超级计算中心提供接入。NSFNET最初具有56kbps的主干速率，到了20世纪80年代末，它的主干运行速率是1.5Mbps，并成为连接区域网络的基本主干。

在ARPAnet界中，许多今天的因特网体系结构的最终部分逐渐变得清晰起来。1983