2Mbps，对于 N 和 u 的每种组合绘制出确定最小分发时间的图表。需要分别针对客户 - 服务器分发和 P2P 分发两种情况制作。

P23. 考虑使用一种客户 - 服务器体系结构向 N 个对等方分发一个 F 比特的文件。假定一种某服务器能够同时向多个对等方传输的流体模型，只要组合速率不超过 u_s，则以不同的速率向每个对等方传输。

a. 假定 $u_s/N \leqslant d_{min}$。定义一个具有 NF/u_s 分发时间的分发方案。

b. 假定 $u_s/N \geqslant d_{min}$。定义一个具有 F/d_{min} 分发时间的分发方案。

c. 得出最小分发时间通常是由 $\max\{NF/u_s, F/d_{min}\}$ 所决定的结论。

P24. 考虑使用 P2P 体系结构向 N 个用户分发 F 比特的一个文件。假定一种流体模型。为了简化起见，假定 d_{min} 很大，因此对等方下载带宽不会成为瓶颈。

a. 假定 $u_s \leqslant (u_s + u_1 + \cdots + u_N)/N$。定义一个具有 F/u_s 分发时间的分发方案。

b. 假定 $u_s \geqslant (u_s + u_1 + \cdots + u_N)/N$。定义一个具有 $NF/(u_s + u_1 + \cdots + u_N)$ 分发时间的分发方案。

c. 得出最小分发时间通常是由 $\max\{F/u_s, NF/(u_s + u_1 + \cdots + u_N)\}$ 所决定的结论。

P25. 考虑在一个有 N 个活跃对等方的覆盖网络中，每对对等方有一条活跃的 TCP 连接。此外，假定该 TCP 连接通过总共 M 台路由器。在对应的覆盖网络中，有多少节点和边？

P26. 假定 Bob 加入 BitTorrent，但他不希望向任何其他对等方上载任何数据（因此称为搭便车）。

a. Bob 声称他能够收到由该社区共享的某文件的完整副本。Bob 所言是可能的吗？为什么？

b. Bob 进一步声称他还能够更为有效地进行他的“搭便车”，方法是利用所在系的计算机实验室中的多台计算机（具有不同的 IP 地址）。他怎样才能做到这些呢？

P27. 考虑一个具有 N 个视频版本（具有 N 个不同的速率和质量）和 N 个音频版本（具有 N 个不同的速率和质量）的 DASH 系统。假设我们想允许播放者在任何时间选择 N 个视频版本和 N 个音频版本之一：

a. 如果我们生成音频与视频混合的文件，因此服务器在任何时间仅发送一个媒体流，该服务器将需要存储多少个文件（每个文件有一个不同的 URL）？

b. 如果该服务器分别发送音频流和视频流并且与客户同步这些流，该服务器将需要存储多少个文件？

P28. 在一台主机上安装编译 TCPClient 和 UDPClient Python 程序，在另一台主机上安装编译 TCPServer 和 UDPServer 程序。

a. 假设你在运行 TCPServer 之前运行 TCPClient，将发生什么现象？为什么？

b. 假设你在运行 UDPServer 之前运行 UDPClient，将发生什么现象？为什么？

c. 如果你对客户端和服务器端使用了不同的端口，将发生什么现象？

P29. 假定在 UDPClient. py 中在创建套接字后增加了下面一行：

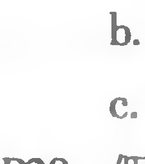
```
clientSocket.bind(('', 5432))
```

有必要修改 UDPServer. py 吗？UDPClient 和 UDPServer 中的套接字端口号是多少？在变化之前它们是多少？

P30. 你能够配置浏览器以打开对某 Web 站点的多个并行连接吗？有大量的并行 TCP 连接的优点和缺点是什么？

P31. 我们已经看到因特网 TCP 套接字将数据处理为字节流，而 UDP 套接字识别报文边界。面向字节 API 与显式识别和维护应用程序定义的报文边界的 API 相比，试给出一个优点和一个缺点。

P32. 什么是 Apache Web 服务器？它值多少钱？它当前有多少功能？为回答这个问题，你也许要看一下维基百科。



套接字编程作业

配套 Web 网站包括了 6 个套接字编程作业。前四个作业简述如下。第 5 个作业利用了 ICMP 协议，在第 5 章结尾简述。第 6 个作业使用了多媒体协议，在第 9 章结尾进行总结。极力推荐学生完成这些作业中的几个（如果不是全部的话）。学生能够在 Web 网站 http://www. pearsonhighered. com/cs-resources 上找到这些作业的全面细节，以及 Python 代码的重要片段。

作业 1：Web 服务器

在这个编程作业中，你将用 Python 语言开发一个简单的 Web 服务器，它仅能处理一个请求。具体而言，你的 Web 服务器将：（1）当一个客户（浏览器）联系时创建一个连接套接字；（2）从这个连接接收 HTTP 请求；（3）解释该请求以确定所请求的特定文件；（4）从服务器的文件系统获得请求的文件；（5）创建一个由请求的文件组成的 HTTP 响应报文，报文前面有首部行；（6）经 TCP 连接向请求的浏览器发送响应。如果浏览器请求一个在该服务器中不存在的文件，服务器应当返回一个“404 Not Found”差错报文。

在配套网站中，我们提供了用于该服务器的框架代码。你的任务是完善该代码，运行你的服务器，通过在不同主机上运行的浏览器发送请求来测试该服务器。如果运行你服务器的主机上已经有一个 Web 服务器在运行，你应当为该 Web 服务器使用一个不同于 80 端口的其他端口。

作业 2：UDP ping 程序

在这个编程作业中，你将用 Python 编写一个客户 ping 程序。该客户将发送一个简单的 ping 报文，接收一个从服务器返回的对应 pong 报文，并确定从该客户发送 ping 报文到接收到 pong 报文为止的时延。该时延称为往返时延（RTT）。由该客户和服务器提供的功能类似于在现代操作系统中可用的标准 ping 程序。然而，标准的 ping 使用互联网控制报文协议（ICMP）（我们将在第 5 章中学习 ICMP）。此时我们将创建一个非标准（但简单）的基于 UDP 的 ping 程序。

你的 ping 程序经 UDP 向目标服务器发送 10 个 ping 报文。对于每个报文，当对应的 pong 报文返回时，你的客户要确定和打印 RTT。因为 UDP 是一个不可靠的协议，由客户发送的分组可能会丢失。为此，客户不能无限期地等待对 ping 报文的回答。客户等待服务器回答的时间至多为 1 秒；如果没有收到回答，客户假定该分组丢失并相应地打印一条报文。

在此作业中，你将给出服务器的完整代码（在配套网站中可找到）。你的任务是编写客户代码，该代码与服务器代码非常类似。建议你先仔细学习服务器的代码，然后编写你的客户代码，可以随意地从服务器代码中剪贴代码行。

作业 3：邮件客户

这个编程作业的目的是创建一个向任何接收方发送电子邮件的简单邮件客户。你的客户将必须与邮件服务器（如谷歌的电子邮件服务器）创建一个 TCP 连接，使用 SMTP 协议与邮件服务器进行交谈，经该邮件服务器向某接收方（如你的朋友）发送一个电子邮件报文，最后关闭与该邮件服务器的 TCP 连接。

对本作业，配套 Web 站点为你的客户提供了框架代码。你的任务是完善该代码并通过向不同的用户账户发送电子邮件来测试你的客户。你也可以尝试通过不同的服务器（例如谷歌的邮件服务器和你所在大学的邮件服务器）进行发送。

作业 4：多线程 Web 代理服务器

在这个编程作业中，你将研发一个简单的 Web 代理服务器。当你的代理服务器从一个浏览器接收到对某对象的 HTTP 请求，它生成对相同对象的一个新 HTTP 请求并向初始服务器发送。当该代理从初始服务器接收到具有该对象的 HTTP 响应时，它生成一个包括该对象的新 HTTP 响应，并发送给该客户。这个代理将是多线程的，使其在相同时间能够处理多个请求。

对本作业而言，配套 Web 网站对该代理服务器提供了框架代码。你的任务是完善该代码，然后测试你的代理，方法是让不同的浏览器经过你的代理来请求 Web 对象。

Wireshark 实验：HTTP

在实验 1 中，我们已经初步使用了 Wireshark 分组嗅探器，现在准备使用 Wireshark 来研究运行中的协议。在本实验中，我们将研究 HTTP 协议的几个方面：基本的 GET/回答交互，HTTP 报文格式，检索大 HTML 文件，检索具有内嵌 URL 的 HTML 文件，持续和非持续连接，HTTP 鉴别和安全性。

如同所有的 Wireshark 实验一样，对该实验的全面描述可查阅本书的 Web 站点 http://www.pearsonhighered.com/cs-resources。

Wireshark 实验：DNS

在本实验中，我们仔细观察 DNS 的客户端（DNS 是用于将因特网主机名转换为 IP 地址的协议）。2.4 节讲过，在 DNS 中客户角色是相当简单的：客户向它的本地 DNS 服务器发送一个请求，并接收返回的响应。在此过程中发生的很多事情均不为 DNS 客户所见，如等级结构的 DNS 服务器互相通信递归地或迭代地解析该客户的 DNS 请求。然而，从 DNS 客户的角度而言，该协议是相当简单的，即向本地 DNS 服务器发送一个请求，从该服务器接收一个响应。在本实验中我们观察运转中的 DNS。

如同所有的 Wireshark 实验一样，在本书的 Web 站点 http://www.awl.com/kurose-ross 可以找到本实验的完整描述。

人物专访

Marc Andreessen 是 Mosaic 的共同发明人，Mosaic 是一种 Web 浏览器，正是它使万维网在 1993 年流行起来。Mosaic 具有一个清晰、易于理解的界面，是首个能嵌在文本中显示图像的浏览器。在 1994 年，Marc Andreessen 和 Jim Clark 创办了 Netscape 公司，其浏览器是到 20 世纪 90 年代中期为止最为流行的。Netscape 也研发了安全套接字层（SSL）协议和许多因特网服务器产品，包括邮件服务器和基于 SSL 的 Web 服务器。他现在是风险投资公司 Andreessen Horowitz 的共同奠基人和一般股东，对包括 Facebook、Foursquare、Groupon、Jawbone、Twitter 和 Zynga 等公司的财产投资搭配进行监管。他服务于包括 Bump、eBay、Glam Media、Facebook 和 HP 等在内的多个董事会。他具有美国伊利诺伊大学厄巴纳－香槟分校的计算机科学理学学士学位。

Marc Andreessen

- 您是怎样变得对计算感兴趣的？您过去一直知道您要从事信息技术吗？

在我长大成人的过程中，视频游戏和个人计算正好成为成功而风行一时的事物，在 20 世纪 70 年代后期和 80 年代初期，个人计算成为新技术发展前沿。那时不只有苹果和 IBM 的个人计算机，而且有如 Commodore 和 Atari 等数以百计的新公司。我在 10 岁时用一本名为《简明 BASIC 速成》（Instant Freeze-Dried BASIC）的书进行自学，并在 12 岁时得到自己的第一台计算机（TRS-80 Color Computer，查查它！）。

- 请描述您职业生涯中干过的一两个最令人激动的项目。最大的挑战是什么？

毋庸置疑，最令人兴奋的项目是 1992～1993 年的初始 Mosaic Web 浏览器，最大的挑战是让任何人从此往后都认真地对待它。在那个时候，每个人都认为交互式未来将是由大型公司宣布的“交互式电视”，而非由新兴公司发明的因特网。

- 您对网络和因特网未来的什么东西感到兴奋？您最为关注什么？

最为兴奋的东西是程序员和企业家能够探索的巨大的尚待开发的应用和服务领域，即因特网已经释放的创造性到达了一种我们以前从未预见到的水平。我最关注是“意想不到的后果”的原则，即我们并不总是知道我们所做事情的后果，例如因特网被政府所用，使监视居民到达了一种新水平。

- 随着 Web 技术的进展，学生们有什么应当特别要了解的？

改变的速度，即对学习来说，最重要的东西是学习的方法，在特定的技术中如何灵活地适应改变，当你在职业生涯中前行时，在新的机会和可能性方面如何保持开放的思想。

- 是谁激发了您的职业灵感？

他们是：Vannevar Bush，Ted Nelson，Doug Engelgart，Nolan Bushnell，Bill Hewlett 和 Dave Packard，Ken Olsen，Steve Jobs，Steve Wozniak，Andy Grove，Grace Hopper，Hedy Lamarr，Alan Turing，Richard Stallman。

- 对于要在计算和信息技术领域谋求发展的学生，您有什么忠告？

尽可能深入地理解技术是怎样创造的，然后补充学习商业运作的原理。

- 技术能够解决世界的问题吗？

不能，但是通过经济增长我们推动人们生活标准的改善。综观历史，大多数经济增长来自技术，因此就像技术带来的好处一样。

第3章

Computer Networking: A Top-Down Approach, Seventh Edition

运 输 层

运输层位于应用层和网络层之间，是分层的网络体系结构的重要部分。该层为运行在不同主机上的应用进程提供直接的通信服务起着至关重要的作用。我们在本章采用的教学方法是，交替地讨论运输层的原理和这些原理在现有的协议中是如何实现的。与往常一样，我们将特别关注因特网协议，即 TCP 和 UDP 运输层协议。

我们将从讨论运输层和网络层的关系开始。这就为研究运输层第一个关键功能打好了基础，即将网络层的在两个端系统之间的交付服务扩展到运行在两个不同端系统上的应用层进程之间的交付服务。我们将在讨论因特网的无连接运输协议 UDP 时阐述这个功能。

然后我们重新回到原理学习上，面对计算机网络中最为基础性的问题之一，即两个实体怎样才能在一种会丢失或损坏数据的媒体上可靠地通信。通过一系列复杂性不断增加（从而更真实!）的场景，我们将逐步建立起一套被运输协议用来解决这些问题的技术。然后，我们将说明这些原理是如何体现在因特网面向连接的运输协议 TCP 中的。

接下来我们讨论网络中的第二个基础性的重要问题，即控制运输层实体的传输速率以避免网络中的拥塞，或从拥塞中恢复过来。我们将考虑拥塞的原因和后果，以及常用的拥塞控制技术。在透彻地理解了拥塞控制问题之后，我们将研究 TCP 应对拥塞控制的方法。

3.1 概述和运输层服务

在前两章中，我们已对运输层的作用及其所提供的服务有所了解。现在我们快速地回顾一下前面学过的有关运输层的知识。

运输层协议为运行在不同主机上的应用进程之间提供了**逻辑通信**（logic communication）功能。从应用程序的角度看，通过逻辑通信，运行不同进程的主机好像直接相连一样；实际上，这些主机也许位于地球的两侧，通过很多路由器及多种不同类型的链路相连。应用进程使用运输层提供的逻辑通信功能彼此发送报文，而无须考虑承载这些报文的物理基础设施的细节。图 3-1 图示了逻辑通信的概念。

如图 3-1 所示，运输层协议是在端系统中而不是在路由器中实现的。在发送端，运输层将从发送应用程序进程接收到的报文转换成运输层分组，用因特网术语来讲该分组称为运输层**报文段**（segment）。实现的方法（可能）是将应用报文划分为较小的块，并为每块加上一个运输层首部以生成运输层报文段。然后，在发送端系统中，运输层将这些报文段传递给网络层，网路层将其封装成网络层分组（即数据报）并向目的地发送。注意到下列事实是重要的：网络路由器仅作用于该数据报的网络层字段；即它们不检查封装在该数据报的运输层报文段的字段。在接收端，网络层从数据报中提取运输层报文段，并将该报文段向上交给运输层。运输层则处理接收到的报文段，使该报文段中的数据为接收应用进程使用。

网络应用程序可以使用多种的运输层协议。例如，因特网有两种协议，即 TCP 和 UDP。每种协议都能为调用的应用程序提供一组不同的运输层服务。

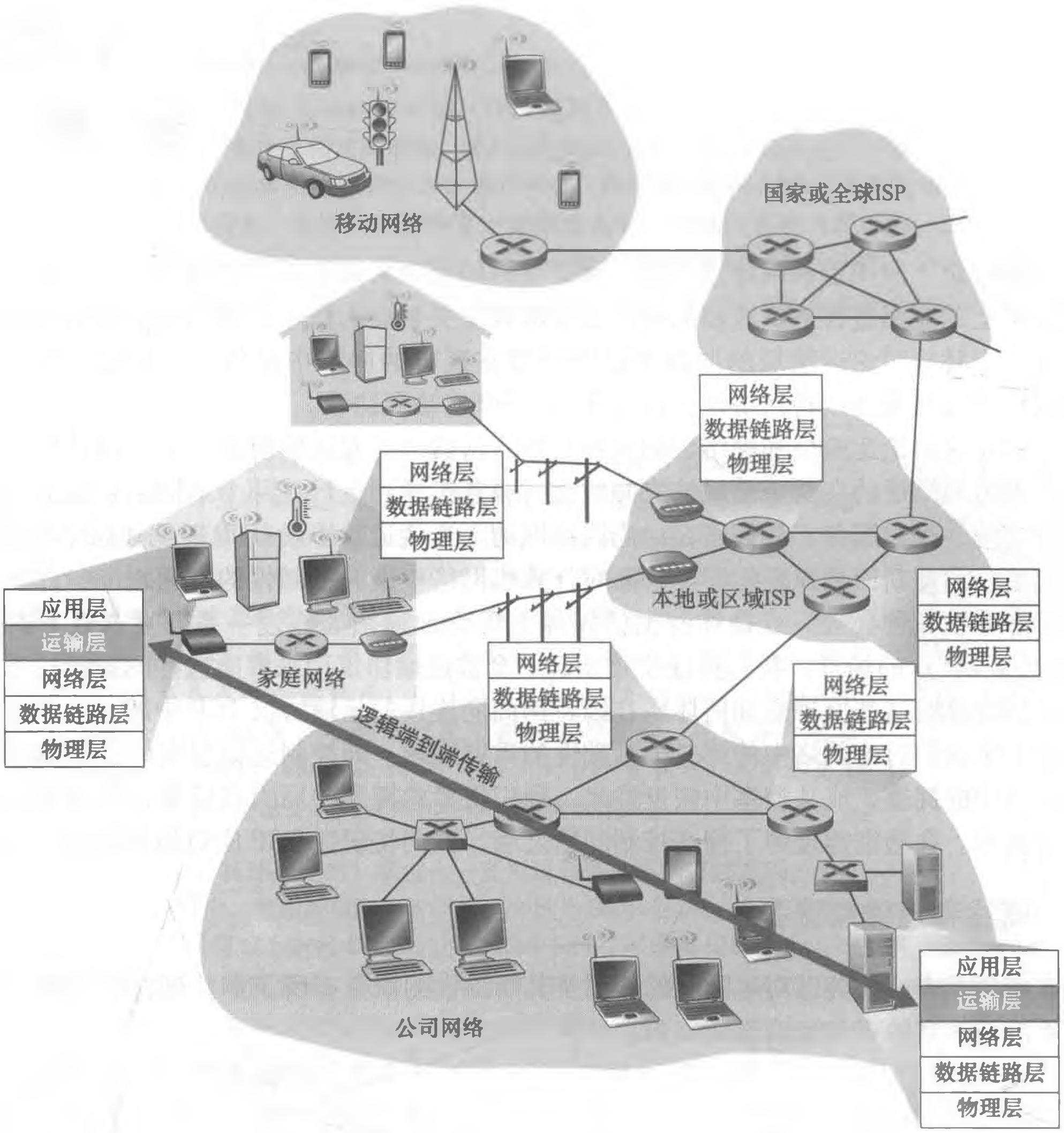


图 3-1　运输层在应用程序进程间提供逻辑的而非物理的通信

3.1.1　运输层和网络层的关系

前面讲过，在协议栈中，运输层刚好位于网络层之上。网络层提供了主机之间的逻辑通信，而运输层为运行在不同主机上的进程之间提供了逻辑通信。这种差别虽然细微但很重要。我们用一个家庭类比来帮助分析这种差别。

考虑有两个家庭，一家位于美国东海岸，一家位于美国西海岸，每家有 12 个孩子。东海岸家庭的孩子们是西海岸家庭孩子们的堂兄弟姐妹。这两个家庭的孩子们喜欢彼此通信，每个人每星期要互相写一封信，每封信都用单独的信封通过传统的邮政服务传送。因此，每个家庭每星期向另一家发送 144 封信。（如果他们有电子邮件的话，这些孩子可以省不少钱!）每一个家庭有个孩子负责收发邮件，西海岸家庭是 Ann 而东海岸家庭是 Bill。每星期 Ann 去她的所有兄弟姐妹那里收集信件，并将这些信件交到每天到家门口来的邮政服务的邮车上。当信件到达西海岸家庭时，Ann 也负责将信件分发到她的兄弟姐妹手上。在东海岸家庭中的 Bill 也负责类似的工作。

在这个例子中，邮政服务为两个家庭间提供逻辑通信，邮政服务将信件从一家送往另一家，而不是从一个人送往另一个人。在另一方面，Ann 和 Bill 为堂兄弟姐妹之间提供了逻辑通信，Ann 和 Bill 从兄弟姐妹那里收取信件或到兄弟姐妹那里交付信件。注意到从堂兄弟姐妹们的角度来看，Ann 和 Bill 就是邮件服务，尽管他们只是端到端交付过程的一部分（即端系统部分）。在解释运输层和网络层之间的关系时，这个家庭的例子是一个非常好的类比。

应用层报文 = 信封上的字符
进程 = 堂兄弟姐妹
主机(又称为端系统) = 家庭
运输层协议 = Ann 和 Bill
网络层协议 = 邮政服务(包括邮车)

我们继续观察这个类比。值得注意的是，Ann 和 Bill 都是在各自家里进行工作的；例如，他们并没有参与任何一个中间邮件中心对邮件进行分拣，或者将邮件从一个邮件中心送到另一个邮件中心之类的工作。类似地，运输层协议只工作在端系统中。在端系统中，运输层协议将来自应用进程的报文移动到网络边缘（即网络层），反过来也是一样，但对有关这些报文在网络核心如何移动并不作任何规定。事实上，如图 3-1 所示，中间路由器既不处理也不识别运输层加在应用层报文的任何信息。

我们还是继续讨论这两家的情况。现在假定 Ann 和 Bill 外出度假，另外一对堂兄妹（如 Susan 和 Harvey）接替他们的工作，在家庭内部进行信件的收集和交付工作。不幸的是，Susan 和 Harvey 的收集和交付工作与 Ann 和 Bill 所做的并不完全一样。由于年龄更小，Susan 和 Harvey 收发邮件的次数更少，而且偶尔还会丢失邮件（有时是被家里的狗咬坏了）。因此，Susan 和 Harvey 这对堂兄妹并没有提供与 Ann 和 Bill 一样的服务集合（即相同的服务模型）。与此类似，计算机网络中可以安排多种运输层协议，每种协议为应用程序提供不同的服务模型。

Ann 和 Bill 所能提供的服务明显受制于邮政服务所能提供的服务。例如，如果邮政服务不能提供在两家之间传递邮件所需时间的最长期限（例如 3 天），那么 Ann 和 Bill 就不可能保证邮件在堂兄弟姐妹之间传递信件的最长期限。与此类似，运输协议能够提供的服务常常受制于底层网络层协议的服务模型。如果网络层协议无法为主机之间发送的运输层报文段提供时延或带宽保证的话，运输层协议也就无法为进程之间发送的应用程序报文提供时延或带宽保证。

然而，即使底层网络协议不能在网络层提供相应的服务，运输层协议也能提供某些服务。例如，如我们将在本章所见，即使底层网络协议是不可靠的，也就是说网络层协议会使分组丢失、篡改和冗余，运输协议也能为应用程序提供可靠的数据传输服务。另一个例子是（我们在第 8 章讨论网络安全时将会研究到），即使网络层不能保证运输层报文段的机密性，运输协议也能使用加密来确保应用程序报文不被入侵者读取。

3.1.2 因特网运输层概述

前面讲过因特网为应用层提供了两种截然不同的可用运输层协议。这些协议一种是 UDP（用户数据报协议），它为调用它的应用程序提供了一种不可靠、无连接的服务。另一种是 TCP（传输控制协议），它为调用它的应用程序提供了一种可靠的、面向连接的服务。当设计一个网络应用程序时，该应用程序的开发人员必须指定使用这两种运输协议中

的哪一种。如我们在2.7节看到的那样，应用程序开发人员在生成套接字时必须指定是选择UDP还是选择TCP。

为了简化术语，我们将运输层分组称为报文段（segment）。然而，因特网文献（如RFC文档）也将TCP的运输层分组称为报文段，而常将UDP的分组称为数据报（datagram）。而这类因特网文献也将网络层分组称为数据报！本书作为一本计算机网络的入门书籍，我们认为将TCP和UDP的分组统称为报文段，而将数据报名称保留给网络层分组不容易混淆。

在对UDP和TCP进行简要介绍之前，简单介绍一下因特网的网络层（我们将在第4和5章中详细地学习网络层）是有用的。因特网网络层协议有一个名字叫IP，即网际协议。IP为主机之间提供了逻辑通信。IP的服务模型是**尽力而为交付服务**（best-effort delivery service）。这意味着IP尽它“最大的努力”在通信的主机之间交付报文段，但它并不做任何确保。特别是，它不确保报文段的交付，不保证报文段的按序交付，不保证报文段中数据的完整性。由于这些原因，IP被称为**不可靠服务**（unreliable service）。在此还要指出的是，每台主机至少有一个网络层地址，即所谓的IP地址。我们在第4和5章将详细讨论IP地址；在这一章中，我们只需要记住每台主机有一个IP地址。

在对IP服务模型有了初步了解后，我们总结一下UDP和TCP所提供的服务模型。UDP和TCP最基本的责任是，将两个端系统间IP的交付服务扩展为运行在端系统上的两个进程之间的交付服务。将主机间交付扩展到进程间交付被称为**运输层的多路复用**（transport-layer multiplexing）与**多路分解**（demultiplexing）。我们将在下一节讨论运输层的多路复用与多路分解。UDP和TCP还可以通过在其报文段首部中包括差错检查字段而提供完整性检查。进程到进程的数据交付和差错检查是两种最低限度的运输层服务，也是UDP所能提供的仅有的两种服务。特别是，与IP一样，UDP也是一种不可靠的服务，即不能保证一个进程所发送的数据能够完整无缺地（或全部!）到达目的进程。在3.3节中将更详细地讨论UDP。

另一方面，TCP为应用程序提供了几种附加服务。首先，它提供**可靠数据传输**（reliable data transfer）。通过使用流量控制、序号、确认和定时器（本章将详细介绍这些技术），TCP确保正确地、按序地将数据从发送进程交付给接收进程。这样，TCP就将两个端系统间的不可靠IP服务转换成了一种进程间的可靠数据传输服务。TCP还提供**拥塞控制**（congestion control）。拥塞控制与其说是一种提供给调用它的应用程序的服务，不如说是一种提供给整个因特网的服务，这是一种带来通用好处的服务。不太严格地说，TCP拥塞控制防止任何一条TCP连接用过多流量来淹没通信主机之间的链路和交换设备。TCP力求为每个通过一条拥塞网络链路的连接平等地共享网络链路带宽。这可以通过调节TCP连接的发送端发送进网络的流量速率来做到。在另一方面，UDP流量是不可调节的。使用UDP传输的应用程序可以根据其需要以其愿意的任何速率发送数据。

一个能提供可靠数据传输和拥塞控制的协议必定是复杂的。我们将用几节的篇幅来介绍可靠数据传输和拥塞控制的原理，用另外几节介绍TCP协议本身。3.4～3.8节将研究这些主题。本章采取基本原理和TCP协议交替介绍的方法。例如，我们首先在一般环境下讨论可靠数据传输，然后讨论TCP是怎样具体提供可靠数据传输的。类似地，先在一般环境下讨论拥塞控制，然后讨论TCP是怎样实现拥塞控制的。但在全面介绍这些内容之前，我们先学习运输层的多路复用与多路分解。

3.2 多路复用与多路分解

在本节中，我们讨论运输层的多路复用与多路分解，也就是将由网络层提供的主机到主机交付服务延伸到为运行在主机上的应用程序提供进程到进程的交付服务。为了使讨论具体起见，我们将在因特网环境中讨论这种基本的运输层服务。然而，需要强调的是，多路复用与多路分解服务是所有计算机网络都需要的。

在目的主机，运输层从紧邻其下的网络层接收报文段。运输层负责将这些报文段中的数据交付给在主机上运行的适当应用程序进程。我们来看一个例子。假定你正坐在计算机前下载 Web 页面，同时还在运行一个 FTP 会话和两个 Telnet 会话。这样你就有 4 个网络应用进程在运行，即两个 Telnet 进程，一个 FTP 进程和一个 HTTP 进程。当你的计算机中的运输层从底层的网络层接收数据时，它需要将所接收到的数据定向到这 4 个进程中的一个。现在我们来研究这是怎样完成的。

首先回想 2.7 节的内容，一个进程（作为网络应用的一部分）有一个或多个**套接字**（socket），它相当于从网络向进程传递数据和从进程向网络传递数据的门户。因此，如图 3-2 所示，在接收主机中的运输层实际上并没有直接将数据交付给进程，而是将数据交给了一个中间的套接字。由于在任一时刻，在接收主机上可能有不止一个套接字，所以每个套接字都有唯一的标识符。标识符的格式取决于它是 UDP 还是 TCP 套接字，我们将很快对它们进行讨论。

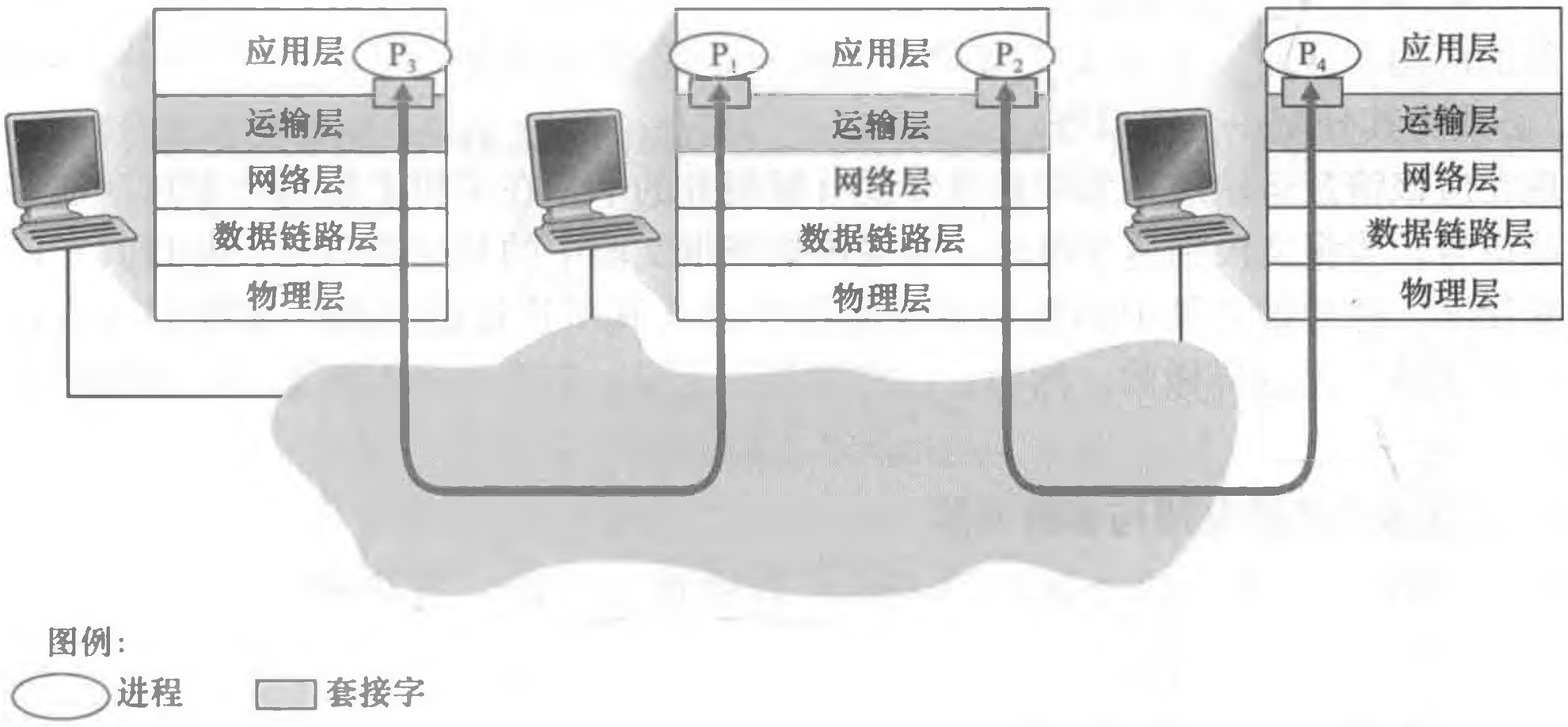


图 3-2 运输层的多路复用与多路分解

现在我们考虑接收主机怎样将一个到达的运输层报文段定向到适当的套接字。为此目的，每个运输层报文段中具有几个字段。在接收端，运输层检查这些字段，标识出接收套接字，进而将报文段定向到该套接字。将运输层报文段中的数据交付到正确的套接字的工作称为**多路分解**（demultiplexing）。在源主机从不同套接字中收集数据块，并为每个数据块封装上首部信息（这将在以后用于分解）从而生成报文段，然后将报文段传递到网络层，所有这些工作称为**多路复用**（multiplexing）。值得注意的是，图 3-2 中的中间那台主机的运输层必须将从其下的网络层收到的报文段分解后交给其上的 P_1 或 P_2 进程；这一过程是通过将到达的报文段数据定向到对应进程的套接字来完成的。中间主机中的运输层也必须收集从这些套接字输出的数据，形成运输层报文段，然后将其向下传递给网络层。尽

管我们在因特网运输层协议的环境下引入了多路复用和多路分解，认识到下列事实是重要的：它们与在某层（在运输层或别处）的单一协议何时被位于接下来的较高层的多个协议使用有关。

为了说明分解的工作过程，再回顾一下前面一节的家庭类比。每一个孩子通过他们的名字来标识。当 Bill 从邮递员处收到一批信件，并通过查看收信人名字而将信件亲手交付给他的兄弟姐妹们时，他执行的就是一个分解操作。当 Ann 从兄弟姐妹们那里收集信件并将它们交给邮递员时，她执行的就是一个多路复用操作。

既然我们理解了运输层多路复用与多路分解的作用，那就再来看看在主机中它们实际是怎样工作的。通过上述讨论，我们知道运输层多路复用要求：①套接字有唯一标识符；②每个报文段有特殊字段来指示该报文段所要交付到的套接字。如图 3-3 所示，这些特殊字段是**源端口号字段**（source port number field）和**目的端口号字段**（destination port number field）。（UDP 报文段和 TCP 报文段还有其他的一些字段，这些将在本章后继几节中进行讨论。）端口号是一个 16 比特的数，其大小在 0 ~ 65535 之间。0 ~ 1023 范围的端口号称为**周知端口号**（well-known port number），是受限制的，这是指它们保留给诸如 HTTP（它使用端口号 80）和 FTP（它使用端口号 21）之类的周知应用层协议来使用。周知端口的列表在 RFC 1700 中给出，同时在 http://www.iana.org 上有更新文档［RFC 3232］。当我们开发一个新的应用程序时（如在 2.7 节中开发的一个简单应用程序），必须为其分配一个端口号。

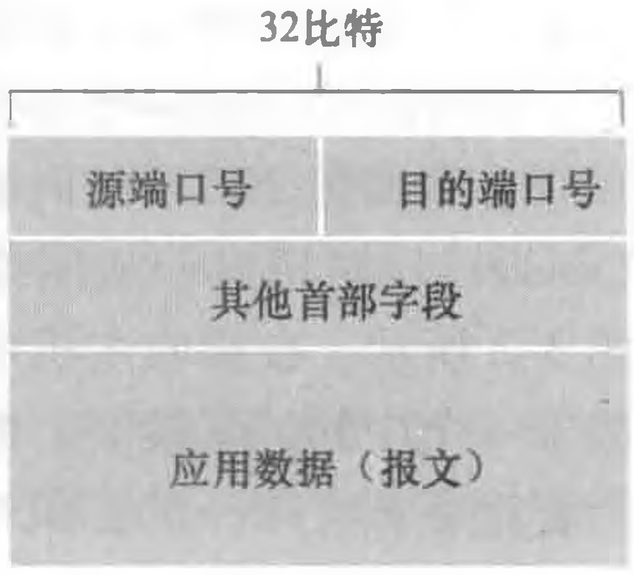


图 3-3　运输层报文段中的源与目的端口字段

现在应该清楚运输层是怎样能够实现分解服务的了：在主机上的每个套接字能够分配一个端口号，当报文段到达主机时，运输层检查报文段中的目的端口号，并将其定向到相应的套接字。然后报文段中的数据通过套接字进入其所连接的进程。如我们将看到的那样，UDP 大体上是这样做的。然而，也将如我们所见，TCP 中的多路复用与多路分解更为复杂。

1. 无连接的多路复用与多路分解

2.7.1 节讲过，在主机上运行的 Python 程序使用下面一行代码创建了一个 UDP 套接字：

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

当用这种方式创建一个 UDP 套接字时，运输层自动地为该套接字分配一个端口号。特别是，运输层从范围 1024 ~ 65535 内分配一个端口号，该端口号是当前未被该主机中任何其他 UDP 端口使用的号。另外一种方法是，在创建一个套接字后，我们能够在 Python 程序中增加一行代码，通过套接字 bind() 方法为这个 UDP 套接字关联一个特定的端口号（如 19157）：

```
clientSocket.bind(('', 19157))
```

如果应用程序开发者所编写的代码实现的是一个“周知协议”的服务器端，那么开发者就必须为其分配一个相应的周知端口号。通常，应用程序的客户端让运输层自动地（并且是透明地）分配端口号，而服务器端则分配一个特定的端口号。

通过为 UDP 套接字分配端口号，我们现在能够精确地描述 UDP 的复用与分解了。假

定在主机 A 中的一个进程具有 UDP 端口 19157，它要发送一个应用程序数据块给位于主机 B 中的另一进程，该进程具有 UDP 端口 46428。主机 A 中的运输层创建一个运输层报文段，其中包括应用程序数据、源端口号（19157）、目的端口号（46428）和两个其他值（将在后面讨论，它对当前的讨论并不重要）。然后，运输层将得到的报文段传递到网络层。网络层将该报文段封装到一个 IP 数据报中，并尽力而为地将报文段交付给接收主机。如果该报文段到达接收主机 B，接收主机运输层就检查该报文段中的目的端口号（46428）并将该报文段交付给端口号 46428 所标识的套接字。值得注意的是，主机 B 能够运行多个进程，每个进程有自己的 UDP 套接字及相应的端口号。值得注意的是，主机 B 可能运行多个进程，每个进程都具有其自己的 UDP 套接字和相联系的端口号。当 UDP 报文段从网络到达时，主机 B 通过检查该报文段中的目的端口号，将每个报文段定向（分解）到相应的套接字。

注意到下述事实是重要的：一个 UDP 套接字是由一个二元组全面标识的，该二元组包含一个目的 IP 地址和一个目的端口号。因此，如果两个 UDP 报文段有不同的源 IP 地址和/或源端口号，但具有相同的目的 IP 地址和目的端口号，那么这两个报文段将通过相同的目的套接字被定向到相同的目的进程。

你也许现在想知道，源端口号的用途是什么呢？如图 3-4 所示，在 A 到 B 的报文段中，源端口号用作“返回地址”的一部分，即当 B 需要回发一个报文段给 A 时，B 到 A 的报文段中的目的端口号便从 A 到 B 的报文段中的源端口号中取值。（完整的返回地址是 A 的 IP 地址和源端口号。）举一个例子，回想 2.7 节学习过的那个 UDP 服务器程序。在 UDPServer. py 中，服务器使用 recvfrom()方法从其自客户接收到的报文段中提取出客户端（源）端口号，然后，它将所提取的源端口号作为目的端口号，向客户发送一个新的报文段。

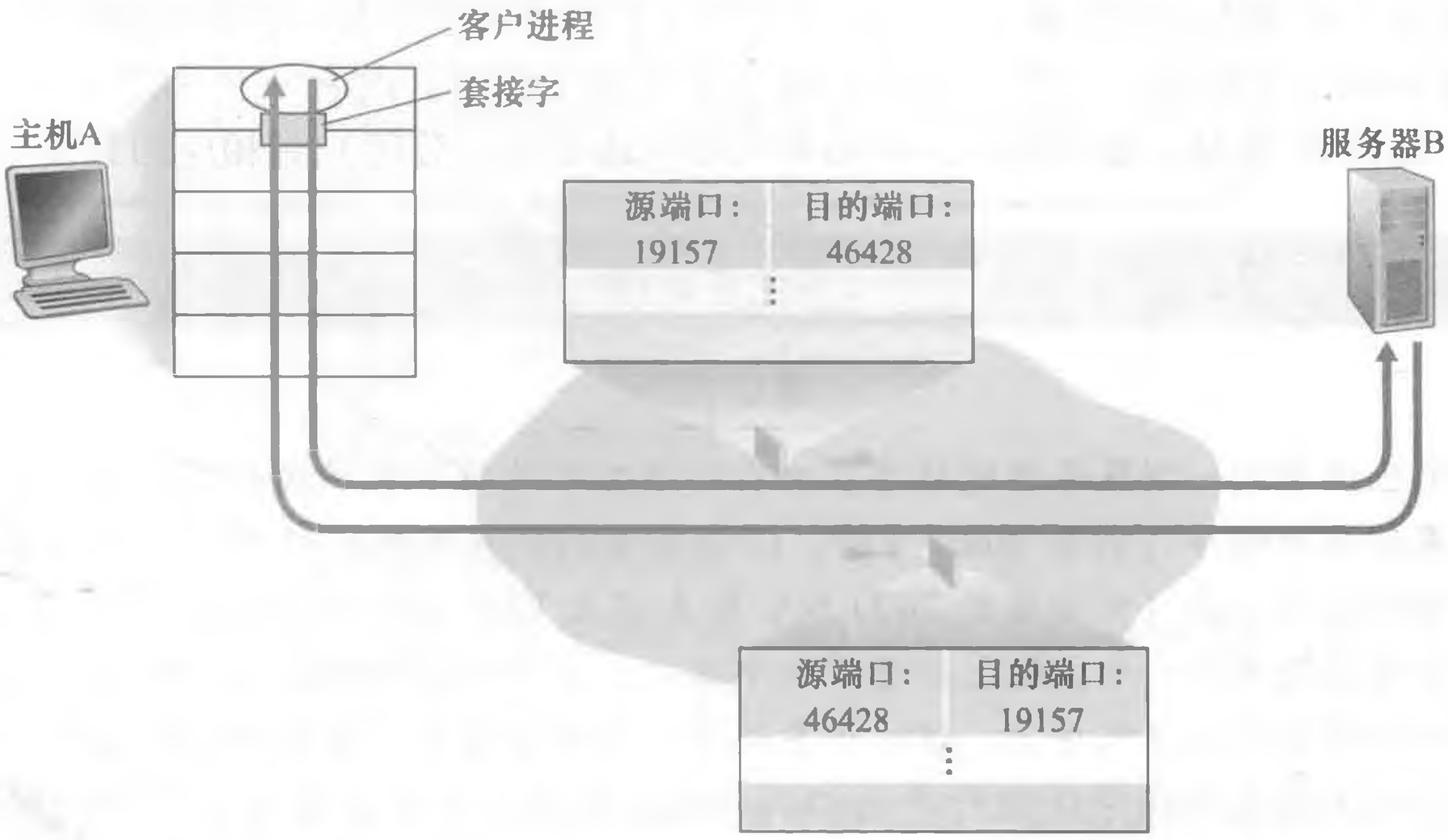


图 3-4 源端口号与目的端口号的反转

2. 面向连接的多路复用与多路分解

为了理解 TCP 多路分解，我们必须更为仔细地研究 TCP 套接字和 TCP 连接创建。TCP 套接字和 UDP 套接字之间的一个细微差别是，TCP 套接字是由一个四元组（源 IP 地址，源端口号，目的 IP 地址，目的端口号）来标识的。因此，当一个 TCP 报文段从网络到达

一台主机时，该主机使用全部4个值来将报文段定向（分解）到相应的套接字。特别与UDP不同的是，两个具有不同源IP地址或源端口号的到达TCP报文段将被定向到两个不同的套接字，除非TCP报文段携带了初始创建连接的请求。为了深入地理解这一点，我们再来重新考虑2.7.2节中的TCP客户－服务器编程的例子：

- TCP服务器应用程序有一个“欢迎套接字”，它在12000号端口上等待来自TCP客户（见图2-27）的连接建立请求。
- TCP客户使用下面的代码创建一个套接字并发送一个连接建立请求报文段：

```
clientSocket = socket(AF_INET, SOCK_STREAM)
               clientSocket.connect((serverName,12000))
```

- 一条连接建立请求只不过是一个目的端口号为12000，TCP首部的特定“连接建立位”置位的TCP报文段（在3.5节进行讨论）。这个报文段也包含一个由客户选择的源端口号。
- 当运行服务器进程的计算机的主机操作系统接收到具有目的端口12000的入连接请求报文段后，它就定位服务器进程，该进程正在端口号12000等待接受连接。该服务器进程则创建一个新的套接字：

```
connectionSocket, addr = serverSocket.accept()
```

- 该服务器的运输层还注意到连接请求报文段中的下列4个值：①该报文段中的源端口号；②源主机IP地址；③该报文段中的目的端口号；④自身的IP地址。新创建的连接套接字通过这4个值来标识。所有后续到达的报文段，如果它们的源端口号、源主机IP地址、目的端口号和目的IP地址都与这4个值匹配，则被分解到这个套接字。随着TCP连接完成，客户和服务器便可相互发送数据了。

服务器主机可以支持很多并行的TCP套接字，每个套接字与一个进程相联系，并由其四元组来标识每个套接字。当一个TCP报文段到达主机时，所有4个字段（源IP地址，源端口，目的IP地址，目的端口）被用来将报文段定向（分解）到相应的套接字。

关注安全性

端口扫描

我们已经看到一个服务器进程潜在地在一个打开的端口等待远程客户的接触。某些端口为周知应用程序（例如Web、FTP、DNS和SMTP服务器）所预留；依照惯例其他端口由流行应用程序（例如微软2000 SQL服务器在UDP 1434端口上监听请求）使用。因此，如果我们确定一台主机上打开了一个端口，也许就能够将该端口映射到在该主机运行的一个特定的应用程序上。这对于系统管理员非常有用，系统管理员通常希望知晓有什么样的网络应用程序正运行在他们的网络主机上。而攻击者为了“寻找突破口”，也要知道在目标主机上有哪些端口打开。如果发现一台主机正在运行具有已知安全缺陷的应用程序（例如，在端口1434上监听的一台SQL服务器会遭受缓存溢出，使得一个远程用户能在易受攻击的主机上执行任意代码，这是一种由Slammer蠕虫所利用的缺陷[CERT 2003-04]），那么该主机已成为攻击者的囊中之物了。

确定哪个应用程序正在监听哪些端口是一件相对容易的事情。事实上有许多公共域程序（称为端口扫描器）做的正是这种事情。也许它们之中使用最广泛的是nmap，该程

序在 http://nmap.org/上免费可用，并且包括在大多数 Linux 分发软件中。对于 TCP，nmap 顺序地扫描端口，寻找能够接受 TCP 连接的端口。对于 UDP，nmap 也是顺序地扫描端口，寻找对传输的 UDP 报文段进行响应的 UDP 端口。在这两种情况下，nmap 返回打开的、关闭的或不可达的端口列表。运行 nmap 的主机能够尝试扫描因特网中**任何地方**的目的主机。我们将在 3.5.6 节中再次用到 nmap，在该节中我们将讨论 TCP 连接管理。

图 3-5 图示了这种情况，图中主机 C 向服务器 B 发起了两个 HTTP 会话，主机 A 向服务器 B 发起了一个 HTTP 会话。主机 A 与主机 C 及服务器 B 都有自己唯一的 IP 地址，它们分别是 A、C、B。主机 C 为其两个 HTTP 连接分配了两个不同的源端口号（26145 和 7532）。因为主机 A 选择源端口号时与主机 C 互不相干，因此它也可以将源端口号 26145 分配给其 HTTP 连接。但这不是问题，即服务器 B 仍然能够正确地分解这两个具有相同源端口号的连接，因为这两条连接有不同的源 IP 地址。

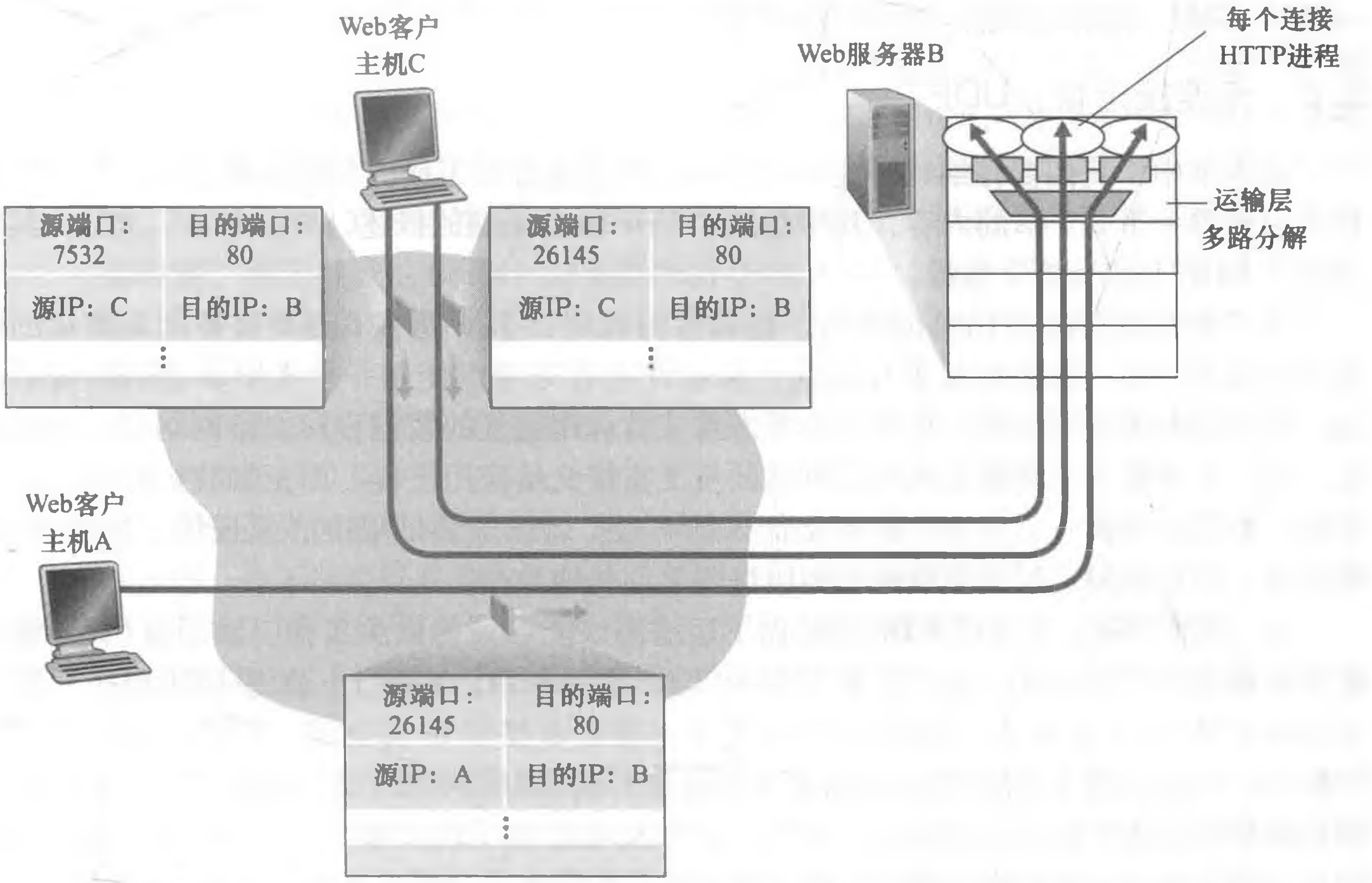


图 3-5 两个客户使用相同的目的端口号（80）与同一个 Web 服务器应用通信

3. Web 服务器与 TCP

在结束这个讨论之前，再多说几句 Web 服务器以及它们如何使用端口号是有益的。考虑一台运行 Web 服务器的主机，例如在端口 80 上运行一个 Apache Web 服务器。当客户（如浏览器）向该服务器发送报文段时，所有报文段的目的端口都将为 80。特别是，初始连接建立报文段和承载 HTTP 请求的报文段都有 80 的目的端口。如我们刚才描述的那样，该服务器能够根据源 IP 地址和源端口号来区分来自不同客户的报文段。

图 3-5 显示了一台 Web 服务器为每条连接生成一个新进程。如图 3-5 所示，每个这样的进程都有自己的连接套接字，通过这些套接字可以收到 HTTP 请求和发送 HTTP 响应。

然而，我们要提及的是，连接套接字与进程之间并非总是有着一一对应的关系。事实上，当今的高性能 Web 服务器通常只使用一个进程，但是为每个新的客户连接创建一个具有新连接套接字的新线程。（线程可被看作是一个轻量级的子进程。）如果做了第 2 章的第一个编程作业，你所构建的 Web 服务器就是这样工作的。对于这样一台服务器，在任意给定的时间内都可能有（具有不同标识的）许多连接套接字连接到相同的进程。

如果客户与服务器使用持续 HTTP，则在整条连接持续期间，客户与服务器之间经由同一个服务器套接字交换 HTTP 报文。然而，如果客户与服务器使用非持续 HTTP，则对每一对请求/响应都创建一个新的 TCP 连接并在随后关闭，因此对每一对请求/响应创建一个新的套接字并在随后关闭。这种套接字的频繁创建和关闭会严重地影响一个繁忙的 Web 服务器的性能（尽管有许多操作系统技巧可用来减轻这个问题的影响）。读者若对与持续和非持续 HTTP 有关的操作系统问题感兴趣的话，可参见［Nielsen 1997，Nahum 2002］。

既然我们已经讨论过了运输层多路复用与多路分解问题，下面我们就继续讨论因特网运输层协议之一，即 UDP。在下一节中，我们将看到 UDP 无非就是对网络层协议增加了一点（多路）复用/（多路）分解服务而已。

3.3　无连接运输：UDP

在本节中，我们要仔细地研究一下 UDP，看它是怎样工作的，能做些什么。我们鼓励你回过来看一下 2.1 节的内容，其中包括了 UDP 服务模型的概述，再看看 2.7.1 节，其中讨论了 UDP 上的套接字编程。

为了激发我们讨论 UDP 的热情，假如你对设计一个不提供不必要服务的最简化的运输层协议感兴趣。你将打算怎样做呢？你也许会首先考虑使用一个无所事事的运输层协议。特别是在发送方一侧，你可能会考虑将来自应用进程的数据直接交给网络层；在接收方一侧，你可能会考虑将从网络层到达的报文直接交给应用进程。而正如我们在前一节所学的，我们必须做一点点事，而不是什么都不做！运输层最低限度必须提供一种复用/分解服务，以便在网络层与正确的应用级进程之间传递数据。

由［RFC 768］定义的 UDP 只是做了运输协议能够做的最少工作。除了复用/分解功能及少量的差错检测外，它几乎没有对 IP 增加别的东西。实际上，如果应用程序开发人员选择 UDP 而不是 TCP，则该应用程序差不多就是直接与 IP 打交道。UDP 从应用进程得到数据，附加上用于多路复用/分解服务的源和目的端口号字段，以及两个其他的小字段，然后将形成的报文段交给网络层。网络层将该运输层报文段封装到一个 IP 数据报中，然后尽力而为地尝试将此报文段交付给接收主机。如果该报文段到达接收主机，UDP 使用目的端口号将报文段中的数据交付给正确的应用进程。值得注意的是，使用 UDP 时，在发送报文段之前，发送方和接收方的运输层实体之间没有握手。正因为如此，UDP 被称为是*无连接的*。

DNS 是一个通常使用 UDP 的应用层协议的例子。当一台主机中的 DNS 应用程序想要进行一次查询时，它构造了一个 DNS 查询报文并将其交给 UDP。无须执行任何与运行在目的端系统中的 UDP 实体之间的握手，主机端的 UDP 为此报文添加首部字段，然后将形成的报文段交给网络层。网络层将此 UDP 报文段封装进一个 IP 数据报中，然后将其发送给一个名字服务器。在查询主机中的 DNS 应用程序则等待对该查询的响应。如果它没有收到响应（可能是由于底层网络丢失了查询或响应），则要么试图向另一个名字服务器发送该查询，要么通知调用的应用程序它不能获得响应。

现在你也许想知道，为什么应用开发人员宁愿在UDP之上构建应用，而不选择在TCP上构建应用？既然TCP提供了可靠数据传输服务，而UDP不能提供，那么TCP是否总是首选的呢？答案是否定的，因为有许多应用更适合用UDP，原因主要以下几点：

- 关于发送什么数据以及何时发送的应用层控制更为精细。采用UDP时，只要应用进程将数据传递给UDP，UDP就会将此数据打包进UDP报文段并立即将其传递给网络层。在另一方面，TCP有一个拥塞控制机制，以便当源和目的主机间的一条或多条链路变得极度拥塞时来遏制运输层TCP发送方。TCP仍将继续重新发送数据报文段直到目的主机收到此报文并加以确认，而不管可靠交付需要用多长时间。因为实时应用通常要求最小的发送速率，不希望过分地延迟报文段的传送，且能容忍一些数据丢失，TCP服务模型并不是特别适合这些应用的需要。如后面所讨论的，这些应用可以使用UDP，并作为应用的一部分来实现所需的、超出UDP的不提供不必要的报文段交付服务之外的额外功能。
- 无须连接建立。如我们后面所讨论的，TCP在开始数据传输之前要经过三次握手。UDP却不需要任何准备即可进行数据传输。因此UDP不会引入建立连接的时延。这可能是DNS运行在UDP之上而不是运行在TCP之上的主要原因（如果运行在TCP上，则DNS会慢得多）。HTTP使用TCP而不是UDP，因为对于具有文本数据的Web网页来说，可靠性是至关重要的。但是，如我们在2.2节中简要讨论的那样，HTTP中的TCP连接建立时延对于与下载Web文档相关的时延来说是一个重要因素。用于谷歌的Chrome浏览器中的QUIC协议（快速UDP因特网连接［Iyengar 2015］）将UDP作为其支撑运输协议并在UDP之上的应用层协议中实现可靠性。
- 无连接状态。TCP需要在端系统中维护连接状态。此连接状态包括接收和发送缓存、拥塞控制参数以及序号与确认号的参数。我们将在3.5节看到，要实现TCP的可靠数据传输服务并提供拥塞控制，这些状态信息是必要的。另一方面，UDP不维护连接状态，也不跟踪这些参数。因此，某些专门用于某种特定应用的服务器当应用程序运行在UDP之上而不是运行在TCP上时，一般都能支持更多的活跃客户。
- 分组首部开销小。每个TCP报文段都有20字节的首部开销，而UDP仅有8字节的开销。

图3-6列出了流行的因特网应用及其所使用的运输协议。如我们所期望的那样，电子邮件、远程终端访问、Web及文件传输都运行在TCP之上。因为所有这些应用都需要TCP的可靠数据传输服务。无论如何，有很多重要的应用是运行在UDP上而不是TCP上。例如，UDP用于承载网络管理数据（SNMP，参见5.7节）。在这种场合下，UDP要优于TCP，因为网络管理应用程序通常必须在该网络处于重压状态时运行，而正是在这个时候可靠的、拥塞受控的数据传输难以实现。此外，如我们前面所述，DNS运行在UDP之上，从而避免了TCP的连接创建时延。

如图3-6所示，UDP和TCP现在都用于多媒体应用，如因特网电话、实时视频会议、流式存储音频与视频。我们将在第9章仔细学习这些应用。我们刚说过，既然所有这些应用都能容忍少量的分组丢失，因此可靠数据传输对于这些应用的成功并不是至关重要的。此外，TCP的拥塞控制会导致如因特网电话、视频会议之类的实时应用性能变得很差。由于这些原因，多媒体应用开发人员通常将这些应用运行在UDP之上而不是TCP之上。当分组丢包率低时，并且为了安全原因，某些机构阻塞UDP流量（参见第8章），对于流式

媒体传输来说，TCP 变得越来越有吸引力了。

应用	应用层协议	下面的运输协议
电子邮件	SMTP	TCP
远程终端访问	Telnet	TCP
Web	HTTP	TCP
文件传输	FTP	TCP
远程文件服务器	NFS	通常 UDP
流式多媒体	通常专用	UDP 或 TCP
因特网电话	通常专用	UDP 或 TCP
网络管理	SNMP	通常 UDP
名字转换	DNS	通常 UDP

图 3-6　流行的因特网应用及其下面的运输协议

虽然目前通常这样做，但在 UDP 之上运行多媒体应用是有争议的。如我们前面所述，UDP 没有拥塞控制。但是，需要拥塞控制来预防网络进入一种拥塞状态，在拥塞状态中可做的有用工作非常少。如果每个人都启动流式高比特率视频而不使用任何拥塞控制的话，就会使路由器中有大量的分组溢出，以至于非常少的 UDP 分组能成功地通过源到目的的路径传输。况且，由无控制的 UDP 发送方引入的高丢包率将引起 TCP 发送方（如我们将看到的那样，TCP 遇到拥塞将减小它们的发送速率）大大地减小它们的速率。因此，UDP 中缺乏拥塞控制能够导致 UDP 发送方和接收方之间的高丢包率，并挤垮了 TCP 会话，这是一个潜在的严重问题［Floyd 1999］。很多研究人员已提出了一些新机制，以促使所有的数据源（包括 UDP 源）执行自适应的拥塞控制［Mahdavi 1997；Floyd 2000；Kohler 2006；RFC 4340］。

在讨论 UDP 报文段结构之前，我们要提一下，使用 UDP 的应用是可能实现可靠数据传输的。这可通过在应用程序自身中建立可靠性机制来完成（例如，可通过增加确认与重传机制来实现，如采用我们将在下一节学习的一些机制）。我们前面讲过在谷歌的 Chrome 浏览器中所使用的 QUIC 协议［Iyengar 2015］在 UDP 之上的应用层协议中实现了可靠性。但这并不是无足轻重的任务，它会使应用开发人员长时间地忙于调试。无论如何，将可靠性直接构建于应用程序中可以使其“左右逢源”。也就是说应用进程可以进行可靠通信，而无须受制于由 TCP 拥塞控制机制强加的传输速率限制。

3.3.1　UDP 报文段结构

UDP 报文段结构如图 3-7 所示，它由 RFC 768 定义。应用层数据占用 UDP 报文段的数据字段。例如，对于 DNS 应用，数据字段要么包含一个查询报文，要么包含一个响应报文。对于流式音频应用，音频抽样数据填充到数据字段。UDP 首部只有 4 个字段，每个字段由两个字节组成。如前一节所讨论的，通过端口号可以使目的主机将应用数据交给运行在目的端系统中的相应进程（即执行分解功能）。长度字段指示了在 UDP 报文段中的字节数（首部加数据）。因为数据字段的长度在一个 UDP 段中不同于在另一个段中，故需要一个明确的长度。接收方使用检验和来检查在该报文段中是

图 3-7　UDP 报文段结构