世界著名计算机教材精选

计算机网络

（第5版）

Andrew S. Tanenbaum
David J. Wetherall 著
严 伟 潘爱民 译

清华大学出版社
北 京

EISBN: 0-13-212695-8

北京市版权局著作权合同登记号 图字:01-2011-3515 号

图书在版编目 (CIP) 数据

计算机网络(第 5 版)/(美)特南鲍姆(Tanenbaum, A. S.),(美)韦瑟罗尔(Wetherall, D. J.)著;严伟,潘爱民译. --北京: 清华大学出版社, 2012.3

书名原文: Computer Networks,5e

(世界著名计算机教材精选)

ISBN 978-7-302-27462-9

Ⅰ. ①计… Ⅱ. ①特… ②韦… ③严… ④潘… Ⅲ. ①计算机网络—教材 Ⅳ. ①TP393

中国版本图书馆 CIP 数据核字(2011)第 249226 号

责任编辑: 龙启铭
封面设计: 傅瑞学
责任校对: 白 蕾
责任印制: 何 芊

出版发行: 清华大学出版社
网 址: http://www.tup.com.cn, http://www.wqbook.com
地 址: 北京清华大学学研大厦 A 座 **邮 编**: 100084
社 总 机: 010-62770175 **邮 购**: 010-62786544
投稿与读者服务: 010-62795954,jsjjc@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015,zhiliang@tup.tsinghua.edu.cn
印 装 者: 北京鑫海金澳胶印有限公司
经 销: 全国新华书店
开 本: 185mm×260mm **印 张**: 47.25 **字 数**: 1147 千字
版 次: 2012 年 3 月第 1 版 **印 次**: 2012 年 3 月第 1 次印刷
印 数: 1～10000
定 价: 89.50 元

产品编号: 036615-01

前　言

现在这本书已经更新到第 5 版了。它的每个版本映衬着人们使用计算机网络的不同阶段。当第 1 版在 1980 年问世时，网络还只是学术上的一种好奇心体现。但到 1988 年出版第 2 版时，网络已经被大学和大型企业用于学术研究和商业应用。当第 3 版于 1996 年出版时，计算机网络，尤其是因特网（Internet）已成为千百万人日常生活中的一部分。而到 2003 年出版第 4 版时，人们利用无线网络和移动电脑访问网页和 Internet 早就司空见惯。现在，正值第 5 版出版之际，计算机网络的重心已偏向内容分发（尤其是通过 CDN 和对等网络获取视频的应用），而且移动电话已成为因特网上的小型电脑。

第 5 版新增内容

第 5 版相比以前的版本改变甚多，其中最重要的一点是 David J. Wetherall 教授成为了本书的合作者。David 教授具有丰富的网络背景，他在城域网的设计领域磨砺了 20 多年，研究范围涉及因特网和无线网络。他是美国华盛顿大学的终身教授，一直讲授和研究计算机网络以及过去十年间相关主题的内容。

当然，书中还有许多内容是随着计算机网络世界的不断变化而更新的。其中，修订和新增内容有：

- 无线网络（802.12 和 802.16）。
- 智能手机使用的 3G 网络。
- RFID 和传感器网络。
- 使用 CDN 的内容分发。
- 对等网络。
- 实时媒体（存储的、流式的以及实况的）。
- 因特网电话（IP 语音）。
- 延迟容忍网络。

更详细的每章变化如下所示。

第 1 章是全书的概述，其功能与第 4 版相同，但内容已被修订，并更新到计算机网络的最新状态。因特网、移动电话网络、802.11、RFID 和传感器网络作为计算机网络实例被加以讨论。有关以太网的起源以及电缆搭接器已被删除，同时被删除的内容还有关于 ATM 的材料。

第 2 章涵盖了物理层的内容，扩展了数字调制（包括广泛用于无线网络的 OFDM）和 3G 网络（基于 CDMA）两部分内容。同时还讨论了一些新技术，包括光纤到户和电力线联网。

第 3 章内容涉及点到点链路，在两个方面有所改进。有关错误检测码和纠正码的内容已被更新，并且还简要描述了实际上非常重要的现代编码技术（例如，*卷积码和 LDPC 码*）。

现在给出的协议实例是 SONET 和 ADSL 上的数据包交换。遗憾的是协议验证部分内容被删掉了，因为它实在是很少被使用。

在第 4 章 MAC 子层中，基本原理是永恒的，改变的只是技术。重新组织了有关网络实例的章节部分，包括千兆以太网、802.11、802.16、蓝牙和 RFID。同时更新的还有 LAN 交换的覆盖范围，包括 VLAN。

第 5 章的网络层涵盖的内容与第 4 版相同，但修订了许多地方。不仅更新了材料，还增加了深度，特别是在服务质量（实时媒体有关）及互联网络方面都有所加强。BGP、OSPF 和 CIDR 相关章节和组播路由同样也有很大扩展，而且还增加了选播路由的新内容。

第 6 章传输层也做了不少修订，补充、修改和删除了一些内容。新增加的内容包括针对延迟容忍网络和一般拥塞控制的描述。修订后的内容更新和扩展了 TCP 的拥塞控制。而针对面向连接的网络层的描述则被删除，因为这些内容现在很少能见到。

第 7 章应用层内容也得到了很大更新和扩充。DNS 和电子邮件部分的内容与第 4 版类似，但过去的几年间 Web 的使用、流媒体和内容分发方面有了很大的发展，因此，关于 Web 和流媒体的相关章节内容已被更新到了最新状态。新增的一个全新小节覆盖了内容分发，包括 CDN 和对等网络的内容。

第 8 章的安全依然包括针对保密性和真实性的对称密钥和公共密钥加密。实际上使用的安全技术相关内容也已经得到更新，包括防火墙和 VPN；而且还增补了 802.11 安全和 Kerberos V5 新内容。

第 9 章重新列出了建议的参考读物和一个超过 300 个引用的最新书目。其中超过一半的读物是 2000 年或稍后发表的文章和书籍，其余的都是一些经典论文。

目　录

第1章 引 言

在过去的三个世纪中，每个世纪都有一种占主导地位的新技术。18 世纪伴随着工业革命到来的是伟大的机械系统时代；19 世纪是蒸汽机时代；在 20 世纪的发展历程中，关键的技术是信息收集、处理和分发。同时我们还看到其他方面的发展，遍布全球的电话网络建设、无线电广播和电视的发明，计算机工业的诞生及其超乎想象的增长速度，通信卫星发射上天，当然还有 Internet。

技术的快速发展使得这些领域在 21 世纪正在迅速地融合，信息收集、传输、存储和处理之间的差别正在快速消失。对于具有数百个办公室的大型组织来说，尽管这些办公室分布在广阔的地理区域，但工作人员有望通过一个按钮就能查看到最远分部的当前状态。随着信息收集、处理和分发能力的不断提高，人们对于更加复杂的信息处理技术的需求增长得更快。

与其他工业相比（例如汽车业和航空运输业），计算机工业还非常年轻；尽管如此，计算机技术却在很短的时间内有了惊人的发展。在计算机诞生之初的 20 年间，计算机系统高度集中，通常被放置于一个很大的房间中。一般来说，这个房间配有透明玻璃墙，供参观的人欣赏房间里这个伟大的电子奇迹。中等规模的公司或者大学可能会有一两台计算机，即使超大型的研究机构也最多只有几十台计算机。要在 40 年内大规模生产出数十亿台功能更强但体积比邮票还小的计算机，在当时的人们看来纯属科学幻想。

计算机和通信的结合对计算机系统的组织方式产生了深远的影响。过去那种用户必须带着任务到一个放置了大型计算机的房间里，再来进行数据处理的“计算机中心”概念，虽然曾经主宰过计算模式，但现在已经完全过时（尽管具有数千台 Internet 服务器的数据中心正在逐步流行起来）。这种由一台计算机服务于整个组织内所有计算需求的老式模型已经被新的模型所取代——大量相互独立但彼此连接的计算机共同完成计算任务。这些系统称为**计算机网络**（computer networks）。如何设计并组织这些网络就是本书的主题。

纵贯全书，我们将使用术语“计算机网络”来表示一组通过单一技术相互连接的自主计算机集合。如果两台计算机能够交换信息，则称这两台计算机是**相互连接的**（interconnected）。这种连接不一定要通过铜线，光纤、微波、红外线和通信卫星都可以用来建立连接。以后我们将会看到，网络可以有不同的大小、形状和形式。这些网络通常连接在一起组成更大的网络，Internet 就是最著名的网络的网络。

在一些文献中，计算机网络和**分布式系统**（distributed system）这两个概念容易使人混淆。两者的关键差别在于，由一组独立计算机组成的分布式系统呈现给用户的是一个关联系统。一般来说，在用户看来，分布式系统只是一个模型或范型。通常在操作系统之上有一层软件负责实现这个模型，这个软件就称为**中间件**（middleware）。最著名的分布式系统例子是**万维网**（World Wide Web）。万维网运行在 Internet 之上，这个模型的所有一切都表现得像是一个文档（Web 页面）一样。

在计算机网络中，这种一致性、模型以及软件都不存在。用户看到的是实际的机器，系统并没有努力使这些机器看起来一样，或者试图使它们的行为保持一致。如果机器有不同的硬件或者不同的操作系统，那么这些差异对于用户来说是完全可见的。如果一个用户希望在一台远程机器上运行一个程序，则他[1]必须首先登录到远程机器上，然后在那台机器上运行该程序。

实际上，分布式系统是建立在网络之上的软件系统。这个软件给予分布式系统以高度的凝聚力和透明性。因此，网络与分布式系统之间的区别在于软件（特别是操作系统），而不是硬件。

然而，这两个主题之间有相当多的重叠。例如，分布式系统和计算机网络都需要移动文件。不同之处在于由谁来发起移动行为，是系统还是用户？虽然本书的主要焦点集中在网络，但是讨论的许多话题对分布式系统也很重要。关于分布式系统的更多信息，请参考（Tanenbaum 和 Van Steen, 2007）。

1.1 使用计算机网络

在开始讨论技术细节之前，值得花一点时间来说明为什么人们对计算机网络会感兴趣，以及可以用计算机网络来做些什么事情。毕竟，如果没有人对计算机网络感兴趣，就不会建设这么多的计算机网络。我们首先讨论针对公司的传统用法，然后再考察家庭网络；并且针对移动用户的最新发展动向，最后以网络的社会问题来结束本章。

1.1.1 商业应用

许多公司都拥有相当数量的计算机。例如，一家公司可能为每个员工配备了一台计算机，员工们用这些计算机设计产品、编写小册子以及做工资表。最初的时候，这些计算机都是独立工作的，但是后来某一天管理部门决定将这些计算机连接起来，以便在整个公司内分发信息。

我们将这个公司的情形稍微通用化一点，这里涉及的问题是**资源共享**（resource sharing）。资源共享的目标是让网络中的任何人都可以访问所有的程序、设备，尤其是数据，并且这些资源和用户所处的物理位置无关。一个非常普遍的例子是一个办公室里的所有工作人员共用同一台打印机。没有一个人真正需要一台私人打印机，而且，一台高性能的网络打印机通常比大量独立的打印机花费更便宜、打印速度更快，而且也更容易维护。

然而，比共享物理资源（比如打印机和磁带备份系统）更重要的是共享信息。一家公司，无论其规模大小，都极其依赖于计算机化的信息。大多数公司都有顾客记录、产品信息、库存数据、财务决算、缴税信息以及其他更多的在线信息。如果所有的计算机突然宕掉，那么一家银行可能坚持不了五分钟，而一个由计算机控制装配线的现代化制造工厂可能持续不到五秒钟。现在，即使是规模很小的旅行社，甚至只有三个人的律师事务所也严重依赖于计算机网络，它们的员工必须通过计算机网络即时访问有关信息和文档。

1 “他”在本书中可以理解为“他”或者“她”。

对小公司而言，可能所有的计算机都集中在一个办公室或者同一座建筑物，但是一个大型公司所拥有的计算机和它们的员工们可能分散在不同国家的许多个办事处和工厂中。位于纽约的一个销售员有时候需要访问位于新加坡的产品库存数据库。一种称为**虚拟专用网络**（VPN，Virtual Private Networks）的网络技术可以将不同地点的单个网络联结成一个扩展的网络。换句话说，即使用户距离数据所在地有1500千米远，这也无法阻挡他使用数据。概括地说，VPN的目标是试图终结“地理位置的束缚”。

按照最简单的形式，可以把一个公司的信息系统想象成是由一个或者多个数据库，以及许多需要远程访问这些数据库的员工们组成的。在这个模型中，数据存储在性能强大的计算机上，这些高性能计算机称为**服务器**（server）。通常，这些服务器集中在一起，由一个系统管理员负责维护。而员工们的桌子上有一些简单的机器，称为**客户机**（client）。通过这些客户机，员工们可以访问远程的数据，例如，正在创建的电子表格（有时候，我们也把客户机的使用者称为“客户”。但是根据上下文，你应该可以判断出到底是指机器，还是指使用机器的用户）。客户机和服务器通过网络连接，如图1-1所示。请注意，这里我们用一个简单的椭圆形来表示一个网络，没有给出其中的任何细节。当我们以最抽象意义表达网络时就使用这种形式，在需要更多细节时我们将会提供详细说明。

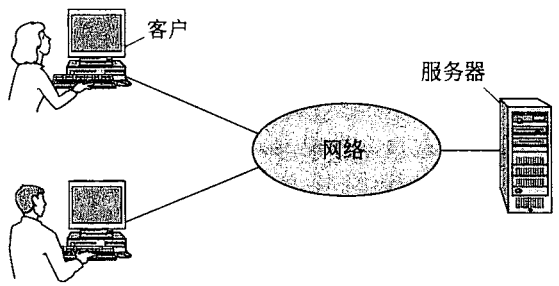


图1-1 一个具有两台客户机和一台服务器的网络

这样的整个安排方式称为**客户机-服务器模型**（client-server model）。这是一种应用很广泛的模型，是许多网络应用的基础。最受欢迎的实现是Web应用。在这种应用中，服务器针对客户请求，根据数据库生成网页，而客户可能会更新数据库内容。当客户和服务器位于同一座建筑物内（比如，属于同一个公司）时，可以采用这种模型；当客户和服务器在地理位置上相隔很远，这种模型也能适用。例如，一个人在家里也可以使用这种模型来访问WWW页面，此时的远程Web服务器是上述模型中的服务器，用户的个人计算机是模型中的客户机。在大多数情况下，一台服务器可以同时处理许多（成百上千个）客户的请求。

如果我们更仔细地考察客户机-服务器模型，可以看到该模型涉及两个进程，一个位于客户机器上，另一个位于服务器机器上。双方的通信形式是这样的：客户机进程通过网络将一个消息发送给服务器进程，然后客户机进程等待应答消息；当服务器进程获得了该请求消息后，它就执行客户所请求的工作，或者查询客户所请求的数据，然后发回一个应答消息。图1-2显示了这些消息及其过程。

构建计算机网络的第二个目标与人有关，而与信息或者甚至计算机都无关。计算机网络可以为员工们提供功能强大的**通信媒介**（communication medium）。现在，几乎每一家公司，只要有两台或更多台计算机，员工们就需要使用**电子邮件**（E-mail，electronic mail）

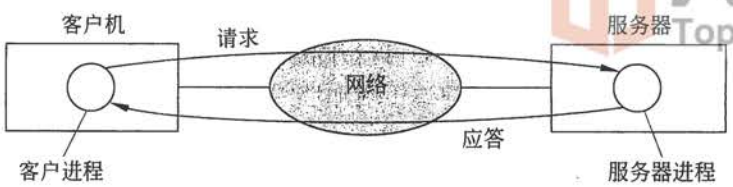


图 1-2 客户机-服务器模型涉及请求和应答

系统进行大量的日常通信。实际上，在办公室里通常可以听到人们在抱怨每天必须处理大量的 E-mail，而其中的一些 E-mail 意义不大，因为公司老板们发现只需点一下鼠标按钮就可以将同样的消息（通常并没有多少实际内容）发送给所有的下属。

员工们之间可以通过计算机网络打电话，而不必再通过电话公司。这项技术称为 **IP 电话**（IP telephony），如果采用了 Internet 技术则称为 **IP 语音**（VoIP，Voice over IP）。通话两端的麦克风和扬声器可能隶属于一个具有 VoIP 功能的手机或者员工的计算机。公司发现这是一个节省电话账单的绝妙方式。

通过计算机网络还可以拥有更丰富的沟通方式。我们可以把视频添加到音频中，使得相距遥远的员工们可以彼此看到和听到对方，犹如他们坐在同一个会议室举行会议一般。这项技术造就了很强大的工具，可以用来消除以前出差所需的费用和时间。**桌面共享**（Desktop sharing）使得远程工作人员与一个图形化计算机屏幕交互工作，因而两个或更多不在同一地点工作的员工可以很容易地读写一块共享的黑板，或者合写一份报告。当一人对某个联机文档做了修改，其他人可立即看到这种改变，而不必等待数天后的信件。这样的加速度使得远程群体之间易于进行协同，而这在以前是不可想象的。远程医疗就是一个例子，类似这种远程协同工作的更宏大形式现在仅仅才开始使用（例如，远程病人监护），但可能会变得更加重要。有时人们会说通信和传输正在进行着一场比赛，无论谁赢得比赛都将淘汰另一个。

对许多公司来说构建计算机网络的第三个目标是做电子商务，特别是与客户和供应商打交道。这种新的模式称为**电子商务**（e-commerce, electronic commerce），近年来得到了迅速增长。航空、书店以及其他零售商已经发现许多客户喜欢那种在家购物的便利性。因此，许多公司提供网上商品和服务目录，并采取网上订购的销售方式。汽车、飞机和计算机制造商从一些供应商处购买各种子系统，然后组装成各部分。利用计算机网络，制造商可以根据需要下电子订单。这样不仅可以减少大量库存，而且可以提高工作效率。

1.1.2 家庭应用

1977 年，Ken Olsen 担任数字设备公司（Digital Equipment Corporation）总裁，这是一家当时排名第二的计算机供应商（仅次于 IBM）。当被问及为什么该公司没有大举进入个人电脑市场时，他说：“没有任何原因促使用户在家里摆放一台计算机”。历史证明事实恰好与之相反，数字设备公司现已不复存在。人们最初购买计算机用于文字处理和玩游戏。近年来，人们购买一台家用电脑的最大原因或许是用来访问 Internet。现在，许多消费类电子设备，比如机顶盒、游戏机、时钟收音机等都具有嵌入式计算机和计算机网络，特别是无线网络，家庭网络已经被广泛应用于娱乐休闲，包括收听和创作音乐，观看照片和视频。

Internet 接入为家庭用户提供了到远程计算机的**连通性**（connectivity）。和企业一样，家庭用户可以访问信息、与他人沟通、购买产品以及享受电子商务服务。现在的主要好处来自于与家庭外面的连接。以太网的发明者 Bob Metcalfe 认为，网络的价值正比于用户数量的平方，因为这大概是网络能拥有的不同连接的数量（Gilder，1993 年）。这个假说称为“Metcalfe 定律”，这有助于解释 Internet 的巨大普及是如何来自于它的规模性。

访问远程信息可以有多种形式。在万维网上冲浪可能是为了获取远程信息，或可能纯粹为了娱乐。可用的信息也有多种多样，包括艺术、商务、烹饪、政府、健康、历史、爱好、娱乐、科学、运动、旅游以及其他等。可以拿出来说的娱乐方式有太多种，当然还有一些方式留下不提更好。

许多新闻报纸已经提供在线阅读形式，并且实行了个性化定制服务。例如，有时候你可以告诉一份报纸，你想要阅读有关腐败政客、火灾、名人丑闻以及流行疾病等这几个方面的信息，但不需要了解足球信息；有时候，可能你还在睡觉时把选出来的文章自动下载到你的计算机。这种趋势继续发展的结果将导致一大批 12 岁报童们的失业，但是报社喜欢这样，因为报纸发行是整个产品链中最薄弱的环节。当然，若要使得这种模式工作良好，它们必须首先找出如何在这个新世界中赢利，由于 Internet 用户期待什么都免费，因此一些事情并不十分明朗。

在线报纸（以及杂志和科学期刊）出现之后是在线数字图书馆。许多专业组织，比如 ACM（www.acm.org）和 IEEE 计算机学会（www.computer.org）早已经将它们的所有期刊和会议论文集放到了网上。电子图书阅读器和在线图书馆可能使印刷书籍成为过去。对此持怀疑态度的人应该注意到印刷出版业给中世纪照明手稿带来的影响。

这些信息中的大部分都是使用客户机-服务器模式访问获得的，但还存在着一种完全不同的获取信息模式，这种模式非常流行，它采用了一种称为**对等（peer-to-peer）通信**的技术（Parameswaran 等，2001）。在这种通信形式下，组成一个松散群体中的个人可以与群体中的其他人通信，如图 1-3 所示。原则上，每个人都可以与一个或多个其他人通信，这里的客户端和服务器没有固定的分工。

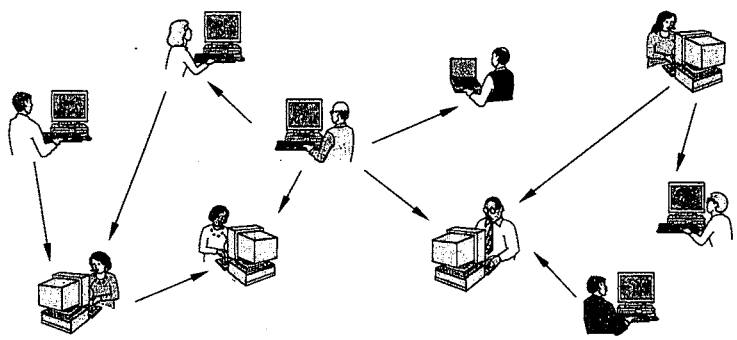


图 1-3 在对等系统中没有固定的客户机和服务器

许多对等系统没有任何中央内容数据库，比如 BitTorrent（Cohen，2003）。相反，每个用户在本地维护他自己的数据库，并为附近群体中的其他成员提供服务。一个新用户到任何一个现有成员那里看看他有什么信息，并且可以获得其他成员的名字，以便检查是否有更多的内容和更多的其他用户名字。这个查询过程可无限重复下去，最终在用户本地建立

一个大型的本地数据库。这是一种令人类倍感乏味的工作，但计算机却擅长于此。

对等通信通常用于共享音乐和视频。它真正引起人们关注是在 2000 年，一个称为 Napster 的音乐共享服务被法院勒令关闭了，这可能是有史以来规模最大的侵犯版权案件（Lam 和 Tan, 2001；Macedonia, 2000）。但是，对等通信的合法应用还是存在的，比如包括乐迷们分享版权公有的音乐、家庭成员之间共享照片和视频以及用户下载公共软件包等。事实上，最流行的 Internet 应用之一，电子邮件，本质上也是对等模式的。这种形式的通信可能在未来有相当大的增长。

所有以上这些应用都涉及了个人与远程信息数据库之间的交互过程。第二类网络应用是人-人通信，基本上可以说，这是21世纪计算机网络对19世纪电话的回应。电子邮件已经成为全世界成千上亿人的日常工作和生活的基础，而且它的使用还在快速增长着。电子邮件的应用包括音频和视频，以及文本和图片。下面再进一步介绍。

十几岁的青少年特别容易沉溺于**即时消息**（instant messaging）。这种应用演变自 1970 年以来一直在使用的 UNIX 系统 talk 程序，它允许两个人相互实时地输入实时消息。除了两人实时交谈外，还有多人参与的消息服务，比如 Twitter 服务，人们可以往自己的社交圈子或其他愿意接受的观众发送简短的文字消息，这种消息称为“推特”（tweet）。

Internet 还可以被用于与音频（例如，Internet 广播电台）、视频（例如 YouTube）有关的应用。除了作为一种廉价的通信方式给远方朋友打电话外，这些应用程序可以提供了丰富的上网体验，比如**远程学习**（telelearning），这意味着再也不用首先下床再坐在教室里上八点钟开始的课了。从长远的观点来看，或许可以证明，网络的使用对于提高人类之间的沟通比任何其他形式都更为重要，它对于那些不得不面对地理障碍的人们将变得尤为重要，因为网络使得他们能访问到与生活在大城市中的人们相同的服务。

人-人之间的通信和访问信息都属于社会网络应用。在这里，信息的流动被人们彼此声明的关系所驱动。Facebook 是目前最流行的社交网站之一，它允许人们更新自己的个人档案，并与那些宣布为朋友的人共享这种更新。其他的社会网络应用可以通过朋友的朋友的介绍，把新闻消息发送给朋友以及更多的人，例如前面所述的 Twitter。

甚至更松散的一群人也可以一起工作来创建内容。例如，wiki 是一个协作型的网站，由一个社团的成员编辑。最著名的 wiki 是维基百科（wikipedia），这是一部任何人都可以编辑的百科全书，除此之外还有数以千计的其他 wiki。

第三类网络应用是广义的电子商务。家庭购物早已流行，用户可以在家里浏览上千家公司的网络商品目录。有些商品目录是交互式的，可以从不同的角度来观看产品，并且可以进行个性化的配置。顾客以电子方式购买了一种商品，如果不清楚如何使用该商品，他可以获得在线技术支持。

电子商务中得到广泛应用的另一个领域是访问金融机构。许多人通过电子方式来支付账单、管理银行的账户和处理他们的投资业务。随着计算机网络变得越来越安全，这种趋势肯定会持续地增长。

另一个几乎人人都没有预料到的领域是电子跳蚤市场（e-flea?）。二手货物的在线拍卖已经变成了一个巨大的行业。与基于客户机-服务器模型的传统电子商务不同，在线拍卖更多地像对等系统，消费者既是卖方也是买方。

由于“to”与“2”在英文中有同样的发音，所以，某些电子商务形式有了一些可爱的

小标签，其中最流行的一些如图 1-4 所示。

标记	全称	例子
B2C	企业对消费者	在线购书
B2B	企业对企业	汽车制造商向供应商订购轮胎
G2C	政府对消费者	政府分发电子税收表单
C2C	消费者对消费者	在线拍卖二手物品
P2P	对等	音乐共享

图 1-4 一些形式的电子商务

网络的第四类应用是娱乐。家庭娱乐方面的应用在近几年有了巨大的发展，音乐、广播和电视节目以及电影通过 Internet 的分发已经开始与传统机制分庭抗礼。用户可以通过网络查找、购买和下载 MP3 歌曲和 DVD 质量的电影，并将它们添加到自己的个人收藏。现在的电视节目通过 IP 电视（IPTV）能到达更多的家庭，IP 电视采用了 IP 技术而不是有线电视电缆或无线传输系统。流媒体应用引导用户调到 Internet 广播电台收听节目，或者观看最近播出的他们喜爱的电视节目。当然，所有这些内容都可以在你房间内不同的设备之间流动，比如显示器和扬声器，而且通常需要使用一个无线网络。

不久以后，或许就有可能史无前例地搜索任何国家的任何电影或电视节目，并即时显示在你的屏幕上。新电影可能成为互动形式，通过事先设定每种情况下的替代场景，偶尔给观众一些提示故事的发展方向（比如，Macbeth 谋杀 Duncan，或者只是等待时机？）。电视直播也有可能成为互动，观众可以参与竞猜节目、选择参赛者，并依此类推出各种形式的活动。

另一种形式的娱乐是玩游戏。我们已经有多人实时仿真游戏，例如在虚拟地牢里藏猫猫（hide-and-seek），双方用飞行模拟器试图击落对方成员。虚拟世界提供了持久性的设置，使得成千上万的用户通过三维图形来体验一个共享的虚拟现实。

最后一类应用是无处不在的**普适计算**（ubiquitous computing），这种计算模式已经融入到了人们的日常生活，正如 Mark Weiser 所描述的那样（1991）。许多家庭安装了有线安全系统，包括门和窗的传感器，还有许多嵌入到智能家居监控的传感器，比如能源消耗。家庭的用电、燃气及水的读数可以通过网络获得。这种方式将节省大量的费用，因为公司不再需要派人去上门抄表。安装的烟雾探测器可直接呼叫消防部门，而不是仅仅发出很大的噪声（如果恰好没有人在家，这种报警器价值不大）。随着传感器和通信成本的下降，越来越多的测量和报告都将通过网络完成。

已经有越来越多的消费类电子设备可以联网了。例如，一些高端相机有无线联网的能力，可用它把照片发送到附近的显示器上供进一步察看。职业体育摄影师首先无线连到一个接入点，然后通过接入点上 Internet，就可以给他们的实时编辑发送自己拍摄的照片。诸如电视机这样的墙插设备可以使用电线网络（power-line networks）来发送信息，而输送电力的线路贯穿了整个房子。这些对象出现在网络上可能并不非常令人惊讶，但是那些我们并不认为是计算机的对象也能感知并发送信息倒是有点令人意想不到。例如，你的淋浴喷头可记录你的洗澡用水，当你浑身泡沫时给你视觉反馈，而在你完成这一切后把这些数据报告给家庭环境监测应用程序，以便帮助你节省水费。

一种称为**射频识别**（RFID，Radio Frequency IDentification）的技术将这一想法推向了

未来。RFID 标签是一种无源芯片（即无电池），大小跟邮票差不多，可以把它贴在家里和外出时携带的书籍、护照、宠物、信用卡以及其他物品上。RFID 读写器可以定位数米范围内的物品，并与之通信，具体的读取范围取决于 RFID 的类型。最初，RFID 被用在商业化过程中，目的是取代条形码。但由于条形码完全免费，而 RFID 标签需要几美分的成本，因而 RFID 尚未获得成功。当然，RFID 标签提供了更多的信息，而且其价格正在迅速下降。它们可能将现实世界变成物联网（ITU，2005）。

1.1.3 移动用户

可移动的计算机，比如笔记本和手持计算机是计算机工业中增长最为迅猛的领域之一。它们的销售早已超过了台式计算机。为什么会有人需要一台移动计算机？人们往往希望在路上的时候还可以使用移动设备阅读和发送电子邮件、微博，观看电影，下载音乐，玩游戏或在 Web 上搜索信息。他们希望在旅途中也能像在家里和办公室一样。自然地，他们希望在陆上、海上或空中任何一个地方做这些与网络有关的事情。

与 Internet 的连通性（connectivity）是这些移动应用的前提，因为汽车、舰船和飞机是不可能拖着一根有线连接的。无线网络领域有许多感兴趣的事情。由电话公司经营的蜂窝网络是大家非常熟悉的一种无线网络，通过基站提供的手机覆盖面把我们连在一起。基于 802.11 标准的**无线热点**（hotspot）是另一种移动计算机的无线网络。它们异军突起在人们出现的每个地方，已经形成了咖啡馆、旅馆、机场、学校、火车和飞机等错落有致的全面覆盖。任何人只要有一台笔记本电脑和一个无线调制解调器，当他们把计算机打开就能通过热点连接到 Internet，仿佛把计算机接入有线网络一样。

无线网络对于运货车队、出租车、快递专车以及修理工与他们的后方基地保持联络特别有用。例如，在许多城市中，出租车司机是独立经营的，他们不受雇于任何一家出租车公司。在这样的城市中，出租车上有一个专门供司机观看的显示器，当有顾客呼叫时，中心调度室就会输入该顾客的上车地点和目的地。于是，这个信息显示在司机的显示器上，并且还会有提示音。第一个在显示器上按下按钮的司机就会接到该顾客的呼叫。

无线网络对于军事用途也非常重要。如果你要在短时间内进行一场战争，那么，指望使用局域网络设施来通信不是个好主意。最好你能够自己搞定一套网络。

虽然无线网络和移动计算常常联系在一起，但它们不是一回事，如图 1-5 所示。这里，我们可以看到**固定无线**（fixed wireless）和**移动无线**（mobile wireless）之间的区别。笔记本电脑有时候可以是有线的。例如，旅游的客人可以把笔记本电脑接入到酒店房间中的电话插孔上，这样即使没有无线网络他也拥有了移动的工作能力。

无线	移动	典型应用
不是	不是	办公室的台式计算机
不是	是	酒店房间里使用的一台笔记本电脑
是	不是	未曾布线的建筑物内的网络
是	是	手持计算机清点商店库存

图 1-5 无线网络和移动计算的结合

反过来，有些无线计算机是不移动的。在家里、办公室或者酒店，由于缺少合适的布

线，那么用无线连接台式机或者媒体播放器比安装有线更加便利。安装一个无线网络只需购买一个里面有些电子元器件的小盒子，拆开包装并插入即可。这种方案比工人在整个大楼内铺设电缆管道要便宜得多。

最后，也有一些真正移动的无线应用，比如用掌上笔记本在仓库周围边走动边记录库存。在许多繁忙的商业飞机场，负责租赁汽车归还工作的雇员多数配备了无线移动计算机。他们输入归还汽车的条形码或者 RFID 芯片，他们的移动计算机向主计算机发出请求获取该汽车的租用信息，然后通过内置的打印机当场打印出账单。

也许，驱动移动无线应用的关键因素是手机。**短消息**（text messaging）或**短信**（texting）盛极一时。手机用户输入一条短消息，然后由蜂窝网络将该消息传递给另一个移动用户。10 年前很少有人预测到乐此不疲地在手机上发送短信的青少年将成为电话公司的一棵巨大摇钱树。但短信（在美国以外地区称为短消息服务）非常赢利，因为运营商中继一条短信息只花了一美分的一小部分，这是一种收费过高的服务。

期待已久的电话和 Internet 的融合终于来临了，这将加速移动应用的增长。**智能手机**（smart phone），结合了移动电话和移动电脑两方面的功能，例如广受欢迎的 iPhone。它们连接的（3G 和 4G）蜂窝网络可提供使用 Internet 的快速数据服务，同时处理电话业务。许多高级的手机还能连接到无线热点，以及在网络之间自动切换以便为用户选择最佳的网络。

其他消费类电子设备也可以使用蜂窝网络和热点网络与远程计算机保持连接。无论用户漫游到哪里都可以用电子图书阅读器下载一本新买的书、最新一期出版的杂志，或当天的报纸。电子相框则可以不出所料地用新鲜的图像来更新它们的显示。

通常移动电话装备有**全球定位系统**（GPS，Global Positioning System）接收器，因而都知道自己当前所处的位置，这样催生了有些服务特别依赖于位置信息。移动地图和方向是很显然的应用候选，因为拥有 GPS 功能的手机和汽车给了你此刻在哪里以及你应该在哪里的更好建议。当然，也可用来搜索附近的书店、中国餐馆或本地天气预报。其他一些服务可能会记录位置信息，比如给照片和视频标注上在哪里生成的注解。这种注解称为“地理标记”（geo-tagging）。

另一个开始使用移动电话的领域是**移动商务**（m-commerce, mobile-commerce）（Senn，2000）。移动电话发出的短消息可作为在购买自动售货机里的食物、电影票等其他小件物品时授权支付，由此省却了现金和信用卡支付。然后这个费用将出现在手机的花费账单上。当配备了**近场通信**（NFC，Near Field Communication）技术后，手机就可以充当 RFID 智能卡和附近的 RFID 读写器互动进而完成消费支付。这一现象背后的驱动力来自移动设备制造商和网络运营商，它们都在努力寻找一种如何获得一块电子商务馅饼的方式。从商店的角度来看，这项支付方案可替他们节省支付给信用卡公司的大部分费用，这笔费用大概有几个百分点。当然，这个方法也可能适得其反，因为在一家商店徘徊的顾客可能在购买物品前使用他们移动设备上的 RFID 或条形码阅读器来检查商家竞争对手的价格，并用它们得到一份附近哪个商店在卖以及售价的详细报告。

移动电子商务得以推行的一个巨大因素是移动电话用户已经习惯了为每件事情付费（相反，Internet 用户希望一切都免费）。如果一个 Internet 网站要收取一定的费用后才允许其客户通过信用卡支付，那么立即就能听到来自用户的巨大反对声。然而，如果移动电话运营商的客户在一家商店的收银台前稍微挥动一下手机就能结账，然后因为这种便利性而

服务费用有所上涨，它可能会被视作正常行为而接受。时间会告诉我们究竟什么样。

毫无疑问，随着未来计算机规模的缩小，移动和无线计算机的使用将更加快速地增长，将会有什么样的可能方式或许现在还没有人能预见。让我们来快速浏览一些可能性。**传感器网络**（sensor network）由能感知到物理世界状态的节点组成，这些节点收集它们感知到的信息，并通过无线中继发送。节点可能是你所熟悉物品的一部分，比如汽车或手机的一部分，或者可能是单独的某个设备。例如，你的汽车可以从车载诊断系统中收集位置、速度、振动和燃油效率等数据，并将这些信息上传至一个数据库（Hull 等，2006）。这些数据有助于行车时发现坑洼，在拥挤的道路周围规划行车路线，并告诉你相比同一段道路上的其他司机，你是一个“油老虎”（如果的确如此）。

传感器网络是革命性的科学，它给从前无法观察到的行为提供了丰富的数据。一个例子是跟踪单个斑马的迁移活动，在每个动物身上放置一个小的传感器（Juang 等，2002）。研究人员已经能把一台无线计算机打包成一个 1 毫米边长的立方体（Warneke 等，2001）。有了如此之小的移动计算机，即使是很小的鸟类、啮齿类动物和昆虫都可以被人类跟踪。

即使是普通的应用也可能得到显著的发展，比如停车咪表，因为这些应用能利用以前没有的数据。无线停车咪表可以通过无线链路的即时验证接受信用卡或借记卡支付；它们还可以通过无线网络报告何时被租用。这样，驾驶员可以下载最新的停车场地图并存到自己的汽车上，以便更容易地找到一个可用的停车点。当然，一旦停车咪表超时，它就检查是否还存在一辆车（通过反射信号，或者，通过检测从汽车反射回的信号），并将超时报告给停车执法部门。据估计，仅在美国，用此方式城市政府就能多收取额外的 10 亿美元（Harte 等，2000）。

可穿戴式计算机（wearable computer）是另一个具有广阔应用前景的领域。带有收音机功能的智能手表自从它们出现在 1946 年的 DickTracy 漫画中，就已经占据了我们心理空间的一部分；而现在你可以购买到它们。其他类似的设备也可被植入到人体，比如心脏起搏器和胰岛素泵。其中有些设备可以通过无线网络控制。这种无线控制使得医生对设备进行测试和重新配置更加容易。但这也可能导致一些讨厌的问题，如果设备像普通 PC 那样不安全，并且很容易被黑客攻破的话情况会很糟糕（Halperin 等，2008）。

1.1.4 社会问题

像 500 年前的印刷机一样，计算机网络使得普通公民以前所未有的方式分发和查看内容。但伴随着网络带来的好处也出现了一些负面问题，这种新发现的自由带来了许多尚未解决的社会、政治和伦理问题。让我们简单地提及其中的几个问题，对这些问题的深入研究至少需要一本完整的书。

社会网络、留言板、内容共享网站和其他应用程序允许人们与志同道合的人分享他们的意见。只要参与者限制在技术主题或类似园艺这样的个人爱好上，就不会有太多的问题出现。当人们真正所关心的话题涉及政治、宗教或者性的时候，麻烦就来了。张贴出来的一些观点可能会深深攻击到某一些人，更糟糕的是，这些观点有可能在政治上是不正确的。而且，这些观点不一定只是文本的形式，在计算机网络上传输高分辨率的彩色图片或者简短的视频片段非常容易。有的人抱着宽以待人的态度，但有的人则认为张贴某些具有特定

含义的材料（比如对于某些国家或者宗教的攻击，或者色情图片等）是完全不可接受的，而且这些材料应该被审查。在这些领域中，不同的国家有不同的法律，有的甚至是相互冲突的。因此，辩论十分激烈。过去，人们起诉网络运营商，声称它们应该对他们传送的内容负责，就像报纸和杂志一样。但得到的必然反应是，网络就像是一个电话公司或邮局，不能指望它监管用户说些什么。

现在应该会有一点轻微的惊喜，因为据说一些网络运营商为了某种自身原因，已经阻塞掉一些内容。一些对等应用程序的用户已经被切断了他们的网络服务，因为这些应用程序能发送大量的流量，而网络运营商发现为这些流量提供传输服务得不到利润。同样是这些运营商可能会以不同的方式对待不同的公司。如果你是一家大公司并且支付良好，那么你可能得到良好的服务；但如果你是一个短时玩家（small-time player），你得到的服务就很差。这种做法的反对者认为对等服务和其他内容服务应该享有同样的方式处理，因为它们都只是网络上的比特而已。这种通信不应该区分内容、来源，或内容提供者的论点称为网络中立（network neutrality）（Wu，2003）。这种辩论肯定还将持续一阵。

在角逐内容分发的过程中涉及许多其他缔约方。举例来说，盗版音乐和电影引发了对等网络的大规模增长，但这使得版权持有人非常不悦，他们威胁说要采取法律行动（有时真的行动了）。现在有一些自动化系统，它们自动搜索对等网络，并向涉嫌侵犯版权的网络运营商和用户发出警告。在美国，这些警告称为数字千年版权法案（Digital Millennium Copyright Act）之后的 DMCA 删除通知（DMCA takedown notices）。这种搜索只是一场军备竞赛，因为它很难可靠地捕捉到侵犯版权。即使你的打印机也可能会被误认为是罪魁祸首（Piatek 等，2008）。

计算机网络使得沟通更加容易。它们也使得运营网络的人更加容易窥探网络流量。这样在一些问题上就产生了冲突，比如员工权益与用人单位权利上的冲突。很多人在工作中阅读和书写电子邮件。许多雇主都声称有权阅读并可能审查员工的邮件，包括在工作时间以外从家庭电脑发出的邮件。并不是所有的员工都同意这种说法，尤其是关于后者。

另一个冲突来自政府和公民权。FBI 已经在许多 Internet 服务提供商处安装了一些系统，用来窥探所有入境和出境邮件。这样的一个早期系统最初称为食肉动物（Carnivore），但是，恶劣的名声迫使其更名为一个现在听起来无辜的名字 DCS1000（Blaze 和 Bellovin, 2000; Sobel, 2001；Zacks, 2001）。但是，这种系统的目标仍然是窥视百万级的人，期望找到有关非法活动的信息。不幸的是，美国宪法的第四修正案明确规定，没有搜查令禁止政府执行任何搜查工作。但美国政府往往忽视这条法律。

当然，威胁到人们隐私的不仅仅只有政府一家。有一些私人信息存储区也同样会泄露个人隐私。例如，Cookie 是 Web 浏览器保存在用户计算机上的小文件，它被公司用来跟踪用户在虚拟空间的活动，也有可能会将信用卡号码、社会安全号码和其他保密信息泄露到 Internet 上（Beighel，2001）。提供基于 Web 服务的公司可能会保存大量有关其用户的个人信息，以便它们直接研究用户的活动。例如，如果你使用谷歌的电子邮件服务 Gmail，谷歌就可以读取你的电子邮件，并根据你的兴趣发布广告信息。

移动设备的一个新的转折是位置隐私（Beresford 和 Stajano, 2003）。在为你的移动设备提供服务时，作为处理的一部分，网络运营商要了解你在一天的不同时间出现在哪里。这

样可以使得它们追踪到你的移动。它们或许知道你最频繁访问的夜总会是哪一家，经常光顾哪些医疗中心。

计算机网络也为用户提供了增加隐私性的潜力，发送匿名消息可以保护当事人的个人信息。在有些情况下，这种能力可能是所需要的。例如，除了防止公司了解你的习惯，还为学生、军人、员工和公民提供检举教授、长官、上级、政府官员等非法行为的有效方式，而不必担心遭到打击报复。另外，在美国以及大多数其他主国家中，法律明确规定了被告有权利要求在法庭上与原告当面对质，因此匿名指控不能作为证据使用。

Internet 有助于快速查找有用信息，但是大量的信息没有得到正确的考虑、误导你或者甚至是完全错误的。当你胸部疼痛，从 Internet 上查到的医疗建议可能来自一个诺贝尔奖得主，也可能来自一个高中辍学的学生。

其他信息经常是你不想要的。电子垃圾邮件（垃圾邮件）已成为生活的一部分，因为垃圾邮件发送者收集了百万计的电子邮件地址，并且他们正想通过这种廉价手段发送计算机生成邮件获得利润呢。这些垃圾邮件造成的流量与真实人群发出的流量旗鼓相当。幸运的是，过滤软件能够读取并丢弃由其他计算机产生的垃圾邮件，它们取得了或多或少的成功。

还有其他一些用作犯罪目的的内容。一些网页和电子邮件含有活动内容（基本上是接收机器执行的程序或宏），这些活动内容可能隐藏着汲取你电脑的病毒。通过这些病毒，犯罪分子可以窃取你的银行账户密码、把你的计算机变成僵尸网络（botnet）或者缺乏免疫力机器池的一部分而发送垃圾邮件。

钓鱼（Phishing）邮件往往伪装成来自一个值得信赖的部门，例如你的银行，这样的邮件诱使你透露敏感信息，例如信用卡号码。身份盗窃正成为一个日益严重的问题，因为盗贼收集有关受害者的足够信息，来获取受害人的信用卡及其他文件。

在 Internet 上防止计算机模拟人类或许很困难。这个问题导致了区分人类和计算机的**完全自动图灵测试**（CAPTCHA，Completely Automated Public Turing test to tell Computers and Humans Apart）的发展。CAPTCHA 的工作原理是这样的：计算机要求一个人解决一个简短的识别任务，例如，在一个扭曲的图像中显示字母输入，以便证明他们是人类（von Ahn，2001）。这个过程是著名的图灵测试的一个变种。在图灵测试中，一个人通过网络发出提问，然后根据对方的响应来判断是否是人类实体在响应。

如果计算机工业界认真考虑安全性，那么，大多数这样的问题都能够得到解决。如果所有的消息都经过加密并被认证，要想犯这样的错误也并不容易。这些技术已经发展得很好，我们将在第 8 章中详细讨论。问题在于硬件和软件制造商们知道，加入这些安全特性需要很高的代价，而且他们的顾客并没有要求这样的安全特性。另外，相当数量的问题是由软件错误引起的，因为软件制造商们不断地在它们的程序中加入新的功能，从而导致代码量越来越大，错误也就越来越多。对新功能增税或许会有所帮助，但这可能会影响几个季度的销售情况。召回有缺陷的软件可能是好事，只不过在刚开始的一年中可能会使整个软件业崩溃。

当计算机网络与旧的法律产生互动时会提出新的法律问题。电子赌博就是这样一个范例。几十年来，计算机已经模拟了许多东西，那么为什么不模拟老虎机、轮盘、二十一点庄主以及更多的赌博设备呢？好吧，因为它在很多地方是不合法的。麻烦的是在很多其他地方（比如英国）赌博是合法的，并且赌场老板拥有与 Internet 赌博的潜力。如果所有的

赌徒、赌场和服务器都分布在不同的国家，并且与法律冲突，那么会发生什么事？绝妙的问题！

1.2 网络硬件

现在我们把注意力从计算机网络的应用和社会问题（甜点）转移到与网络设计（菠菜）有关的技术问题上来。关于计算机网络，没有一种被普遍接受的分类方法，但是有两个维度非常重要：传输技术和网络尺度。下面我们依次讨论这些。

从广义上讲，目前普遍使用的传输技术有两种，分别是广播式链路和点到点链路。

点到点（point-to-point）链路将一对单独的机器连接起来。在一个由点到点链路组成的网络中，为了从源端到达接收方，短消息必须首先访问一个或多个中间机器，这种短消息在某些情况下称为数据包或包[1]（packet）。通常在网络中有可能存在多条不同长度的路由，因此，找到一条好的路由对点到点网络非常重要。点-点传输只有一个发送方和一个接收方，有时候也称为**单播**（unicasting）。

相反，在一个广播网络上，通信信道被网络上的所有机器所共享；任何一台机器发出的数据包能被所有其他任何机器收到。每个数据包的地址字段指定了预期的接收方。当一台机器收到一个数据包时，它要检查地址字段。如果包的目的地就是接收机器，则该机器要处理此数据包；如果包的目的地址是某台其他机器，则该机器就忽略此数据包。

无线网络是广播链路的一个常见例子，一个覆盖区域内的通信由所有该区域内的机器共享，而该区域的划分取决于无线信道和传输机器。我们来看一个形象化的比喻，某个人站在会议室里大声喊“沃森，到这里来。我需要你”。虽然实际上接收（听）到该数据包的人很多，但只有沃森将给出响应，其他人都会忽视它。

广播系统往往还提供将一个数据包发送给所有目标机器的可能性，只要在地址字段中使用一个特殊的编码。如果被传输的数据包带有这样的地址编码，那么网络中的每一台机器都将会接收该包并对其进行处理。这种传输模式称为**广播**（boradcasting）。有些广播系统还支持给一组机器发送数据包的模式，这种传输模式称为**组播**（multicasting）。

另一种对网络进行分类的标准是网络尺度。距离作为分类指标非常重要，因为不同的尺度采用了不同的技术。

在图 1-6 中，我们根据大致的物理尺寸区分了多处理器系统。最顶部的个域网意味着围绕一个人的网络。除了这些个域网外就是范围较大的网络，可进一步分为局域网、城域网和广域网，网络尺度相应地越来越大。最后，两个或多个网络的连接称为**互联网络**（internetwork）。毫无疑问，全球性的 Internet 是最著名的互联网络实例（但不是唯一的）。不久我们将有更大的互联网络，即**星际互联网络**（Interplanetary Internet），它能跨越太空把网络连接起来（Burleigh 等，2003）。

在本书中，我们将关注具有不同尺度的网络。接下来的几小节，我们将根据网络尺度简略地介绍网络硬件。

1 packet 在以前的书籍中称为“分组”。

处理器之间距离	处理器所在位置	例子
1米	一米见方	个域网
10米	同一个房间	局域网
100米	同一栋建筑物	
1千米	同一个园区	
10千米	同一座城市	城域网
100千米	同一个国家	广域网
1000千米	同一个大陆	
10 000千米	同一个行星	互联网

图 1-6 按照不同尺度对互连处理器的分类

1.2.1 个域网

个域网（PAN，Personal Area Network）允许设备围绕着一个人进行通信。一个常见的例子是计算机通过无线网络与其外围设备连接。几乎每一台计算机都有显示器、键盘、鼠标和打印机等外设。如果不使用无线传输技术，那么这些外设必须通过电缆连接到计算机。太多的新用户很难找到合适的电缆并将电缆插入到合适的小孔中（即使这些电缆通常使用了彩色标记），以至于大多数计算机供应商提供了一种服务选项，即派遣技术人员到用户家中安装。为了帮助这些用户，一些公司联合起来设计出了一种短距离无线网络来连接这些计算机组件而无须电线，这种无线网络就是**蓝牙**（bluetooth）。他们的设计想法是这样的，如果设备有蓝牙，那么就不再需要线缆。只要把这些线缆拔下来，然后重新启动机器，这些设备就都能一起工作了。对于许多人来说，这种简易操作是一大利好。

最简单的形式下，蓝牙网络采用主-从操作模式，如图 1-7 所示。系统单元（PC）通常是主设备，与鼠标、键盘等从设备通信。主设备告诉从设备以后广播时使用什么地址、它们能够传输多长时间、它们可以使用什么频率等所有与传输有关的信息。

蓝牙可用在其他设备中。它通常可把耳麦与手机相连而无须连线，也可以把数字音乐播放器与一定范围内的汽车相连。当一些嵌入式医疗设备，比如心脏起搏器、胰岛素泵、助听器受到用户操作的远程控制时，就形成了一种完全不同类型的 PAN。我们将在第 4 章更详细地讨论蓝牙。

PAN 也可以采用其他短程通信技术来搭建，比如智能卡和图书馆书籍上的 RFID。我们将在第 4 章中学习 RFID。

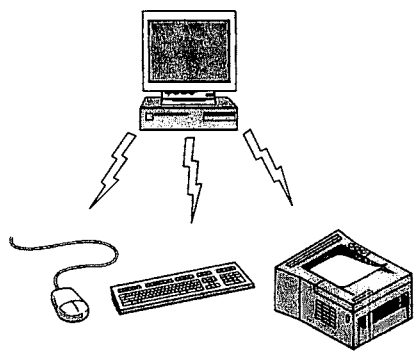


图 1-7 蓝牙 PAN 配置

1.2.2 局域网

接下来的是局域网（LAN，Local Area Network）。局域网是一种私有网络，一般在一座建筑物内或建筑物附近，比如家庭、办公室或工厂。局域网络被广泛用来连接个人计算机和消费类电子设备，使它们能够共享资源（比如打印机）和交换信息。当局域网被用于公司时，它们就称为**企业网络**（enterprise nework）。

无线局域网近来受到非常大的欢迎，尤其是家庭、旧办公楼、食堂和其他一些安装电缆太麻烦的场地。在这些系统中，每台计算机都有一个无线调制解调器和一个天线，用来与其他计算机通信。在大多数情况下，每台计算机与安装在天花板上的一个设备通信，如图 1-8（a）所示。这个设备，称为**接入点**（AP，Access Point）、**无线路由器**（wireless router）或者**基站**（base station），它主要负责中继无线计算机之间的数据包，还负责中继无线计算机和 Internet 之间的数据包。AP 就像是一位在学校备受欢迎的可爱小孩，每个人都乐于和他交谈。然而，如果其他计算机彼此足够靠近，它们可以采用另一种点到点的配置直接进行通信。

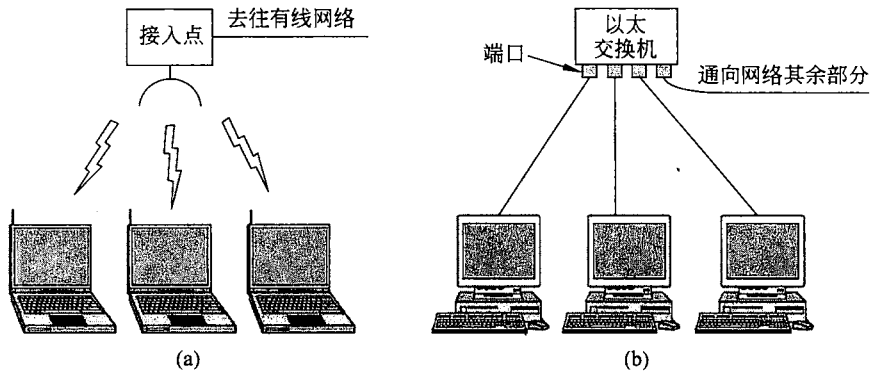


图 1-8 无线和有线 LAN
（a）IEEE 802.11；（b）交换式以太网

无线局域网的一个标准称为 IEEE 802.11，俗称为 WiFi，已经被非常广泛地使用。它在任何地方可以从 11 Mbps 到几百个 Mbps 的速率运行。我们在本书中将沿用传统的线路测量速率：1 Mbps（兆比特/秒）等于 1 000 000 比特/秒；1 Gbps（吉比特/秒）等于 1 000 000 000 比特/秒。我们将在第 4 章讨论 IEEE 802.11。

有线局域网使用了各种不同的传输技术。它们大多使用铜线作为传输介质，但也有一些使用光纤。局域网的大小受到限制，这意味着最坏情况下的传输时间也是有界的，并且事先可以知道。了解这些界限有助于网络协议的设计。通常情况下，有线局域网的运行速度在 100 Mbps 到 1 Gbps 之间，延迟很低（微秒或纳秒级），而且很少发生错误。较新的局域网可以工作在高达 10 Gbps 的速率。和无线网络相比，有线局域网在性能的所有方面都超过了它们。因为通过电线或通过光纤发送信号比通过空气发送信号更容易，道理就这么简单。

许多有线局域网的拓扑结构是以点到点链路为基础的。俗称**以太网**（Ethernet）的 IEEE 802.3 是迄今为止最常见的一种有线局域网。图 1-8（b）显示了一个**交换式以太网**（switched

Ethernet）的拓扑例子。每台计算机按照以太网协议规定的方式运行，通过一条点到点链路连接到一个盒子，这个盒子称为**交换机**（switch）。这就是交换式以太网名字的由来。一个交换机有多个**端口**（port），每个端口连接一台计算机。交换机的工作是中继与之连接的计算机之间的数据包，根据每个数据包中的地址来确定这个数据包要发送给哪台计算机。

为了建立较大的局域网，交换机必须插入到彼此的端口中。如果把它们插入在一起形成一个环，会发生什么事？网络仍然能工作吗？幸运的是，设计者们考虑到了这种情况。这正是协议的工作任务，协议必须梳理数据包的路径，并选择该走的路径，抵达预定的计算机。我们将在第4章了解交换机是如何工作的。

也有可能将一个大的物理局域网分成两个较小的逻辑局域网。你可能会质疑这有什么用。有时，网络设备的布局不一定与组织结构匹配。例如，某个公司的工程和财务部门各自都有计算机，因为它们位于同一座建筑物的侧翼，因此它们的计算机有可能在同一个物理局域网上；如果工程部门和财务部门逻辑上有自己独享的**虚拟局域网**（Virtual LAN 或 VLAN），那么更易于管理各自的系统。在这个设计中，每个端口都带有一个“彩色”的标签，比如说绿色表示工程部门，而红色表示财务部门。然后交换机在转发数据包时，将连到绿色端口的计算机和连到红色端口的计算机区别开来。例如，在红色端口上发送的广播数据包将不会被连到绿色端口上的计算机收到，就好像存在两个不同的局域网一样。我们将在第4章的结尾时讨论 VLAN。

当然，还有其他形式的有线局域网拓扑结构。事实上，交换式以太网是原始以太网设计的一个现代版。在最初的以太网设计中，所有的数据包在一条线性电缆上广播，因而一次至多只有一台机器能够成功发送，为此，需要一个分布式仲裁机制来解决冲突问题。分布式仲裁机制的算法非常简单：只要电缆空闲计算机就可以传输；如果两个或两个以上的数据包发生冲突，每台计算机只是等待一个随机时间后再次试图发送。为清晰起见，我们称该版本的以太网为**经典以太网**（classic Ethernet），正如你心中对此存有疑虑，你将在第4章了解它。

无线和有线广播网络可分为静态设计和动态设计，两种设计取决于如何分配信道。一个典型的静态分配方案是将时间划分成离散的时间间隔（译注：这段间隔就称为时间槽）并使用轮循算法，每台机器只能在分配给它的**时间槽**（time slot）到来时广播。当一台机器在分配给它的时间槽到来时没有任何数据需要发送，这种静态分配算法就浪费了信道容量（此时其他计算机也不允许发送），因此大多数系统都试图动态分配信道（即按需分配）。

一个公共信道的动态分配方法可以是集中式的，也可以是分散式的。在集中式的信道分配方法中，有一个中心实体，例如蜂窝网络中的基站，由这个中心实体决定接下来谁使用信道。具体的算法可能是它接受多个数据包，按照某个内部算法确定这些数据包发送的优先次序。在分散式信道分配方法中，没有一个中央实体；每台机器必须自行决定是否可以传输。你可能会认为，这种做法将导致信道使用的混乱，但事实并非如此。稍后，我们将研究旨在将这种潜在的混乱秩序化的许多算法。

家庭局域网值得花多一点时间讨论。在未来，很可能每台家电都具备与所有其他设备进行通信的能力，而且它们全部都可以访问 Internet。这种发展很可能是那些有远见的概念之一，没有人要求这样（如电视机的遥控器或移动电话），但一旦到达实现了这种愿景的那一天，又没有人能够想象没有它们的生活会怎么样。

许多设备早已具备联网能力。这些设备包括计算机、娱乐设备（诸如电视和DVD等）、手机和其他消费类电子产品（比如照相机）、时钟收音机等家电以及基础设施（比如电表和自动调温器）。这种趋势只会持续地发展下去。举例来说，平均每个家庭大概有十几个时钟（例如，家电中的时钟），如果所有这些时钟都在 Internet 上，那么它们就可以自动适应夏令时（或者夏时制）。家庭的远程监控或许能成为赢家，因为许多成年子女愿意花一些钱，来使居住在自己家中的年迈父母住得更加安全。

虽然我们可以认为家庭网络只是另一种局域网，但它更可能比其他网络拥有不同的属性。第一，网络设备的安装必须非常容易。无线路由器是退货最多的消费类电子产品。人们购买无线路由器是因为他们想在家里搭建一个无线网络，但发现“打开包装后”它不工作了；然后把它作退货处理，而不愿意举着接通技术服务热线的听筒听漫长的电梯音乐。

第二，网络和设备操作上必须万无一失。过去的空调器通常有4个设置旋钮：关、低、中等和高。现在它们的说明手册已经长达30页。一旦把它们联网，预计仅“安全”这一章就需要额外的30页。这的确是个问题，因为只有计算机用户才习惯于容忍那些不能正常工作的产品；购买汽车、电视和冰箱的公众远不如计算机用户那样宽容。他们期望产品100%的工作，而不需要聘请一个电脑怪人（或电脑达人）。

第三，价格廉价是家庭网络成功的关键。人们不会为了 Internet 恒温器而支付50美元的额外费用，因为很少有人把他们家的温度监测工作看得如此重要。虽然，如果只需要额外付5美元，这个恒温器或许就能卖出去。

第四，它必须从一个或两个设备开始，并逐步扩大网络的覆盖范围。这意味着或许能免去格式之战。告诉消费者购买带有 IEEE 1394（火线）接口的外设，几年后回收该设备并告诉消费者说USB 2.0才是本月应当使用的接口，然后不久又切换到802.11 g的接口——哎呀，不，用802.11n——我的意思是用802.16（一种不同的无线网络），如此多变将把消费者变得非常**不安**。网络接口应该几十年保持稳定，像电视广播标准那样。

第五，安全和可靠性非常重要。因电子邮件病毒丢失几个文件是一回事；如果一个窃贼从他的移动计算机解除了你家庭网络的安全系统，然后掠夺你房子里的财产，那可就是一件完全不同的事情。

一个有趣的问题是家庭网络应该是有线还是无线。因为没有布好的线可用，或者更糟糕的是需要改造重新布线，所以从便利性和成本两方面来看，无线网络比较有利。但从安全的角度出发来考虑问题，有线网络却比较有利，因为无线网络使用的无线电波能很好地穿透墙壁。不是每个人都愿意邻居搭上自己的 Internet 连接和阅读自己的电子邮件。在第8章，我们将研究如何利用加密来提供安全性，但说起来容易，让没有经验的用户来做可就困难了。

对于如何设置家庭网络，可能有吸引力的第三个选项是重复利用家里早已存在的网络。最明显的候选网络是遍布整所房子的电线。**电线网络**（power-line network）允许插入到墙上插座上的设备在全房子内广播信息。无论如何你都必须以某种方式把电视插入插座才能使其工作，在这插上插座的同时你便获得了与 Internet 连接。困难的是如何在同一时间携带电源信号和数据信号。解决该问题的部分答案是让它们使用不同的频段。

总之，家庭局域网面临着许多机遇和挑战。大多数挑战与网络易于管理、高可靠和安全性等方面的需求有关，特别是针对非技术用户而言，还要求成本低廉。

1.2.3 城域网

城域网（MAN，Metropolitan Area Network）的范围可覆盖一个城市。最有名的城域网例子是许多城市都有的有线电视网。这种系统由早期的社区天线系统发展而来，主要用在那些从空中接收电视信号条件较差的地区。在这些早期系统中，常常把一个很大的天线放在附近的山上，然后电视信号通过该天线转发到订户的家里。

最初的时候，这些系统都由本地设计，是纯粹的**自组织**（ad hoc）系统；然后一些公司开始参与商业化运作，它们从当地政府部门拿到筹建整个城市电视网的合同。接下来是电视节目的编排，以及专门针对有线电缆而设计的整个频道分配。通常这些频道各司其职，比如有的频道全部是新闻节目，有的频道是体育运动节目，有的是烹饪节目，有的是园艺节目，等等。但是，从初期的网络建设一直到20世纪90年代后期，这些频道只能专用于电视节目的接收。

当 Internet 开始吸引大量观众时，有线电视网络经营商也开始意识到，只需要对原有的系统稍做改动，就可以利用原来尚未使用的频谱来提供双向的 Internet 服务。从这时候起，有线电视系统就从分发电视节目这单一模式演变为一个城域网。近似地，一个 MAN 看起来很像图 1-9 中所示的系统。在图 1-9 中，我们可以看到，电视信号和 Internet 流量都先被送到一个集中式线缆前端（cable headend），然后再分发到居民的家中。我们在第 2 章将再回到这个主题的细节上来。

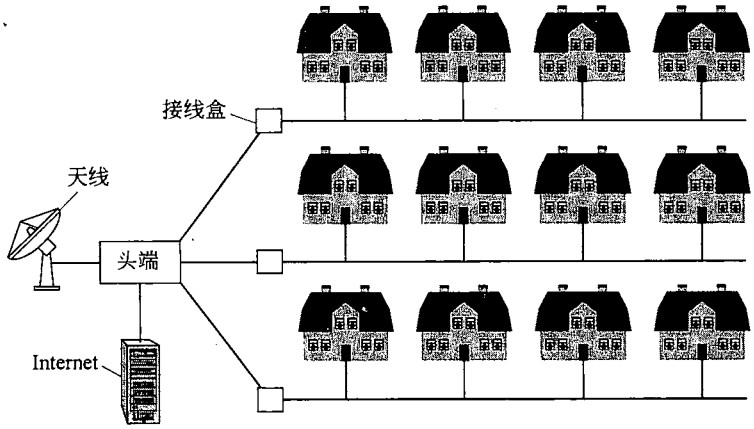


图 1-9 基于有线电视的城域网

有线电视不是唯一的城域网，虽然它是一种局域网。最近发展的高速无线 Internet 接入催生了另一种城域网，并且已经被标准化为 IEEE 802.16，这就是所谓的 WiMAX。我们将在第 4 章了解这种技术。

1.2.4 广域网

广域网（WAN，Wide Area Network）的范围很大，它能跨越很大的地理区域，通常是一个国家、地区或者一个大陆。我们将从有线广域网开始讨论，采用在不同城市有分支机

构的公司作为案例来加以说明。

图 1-10 所示的是一个公司的广域网，它连接了该公司设在墨尔本、珀斯和布里斯班三个城市的办事处。每个办事处都有专门运行用户（即应用）程序的计算机。我们将按照传统的说法把这些机器称为**主机**（host），然后把连接这些主机的网络其余部分称为**通信子网**（communication subnet），或简称为**子网**（subnet）。子网的工作是把信息从一个主机携带到另一个主机，就像电话系统把说话者的话（实际上是声音）传递给接听者一样。

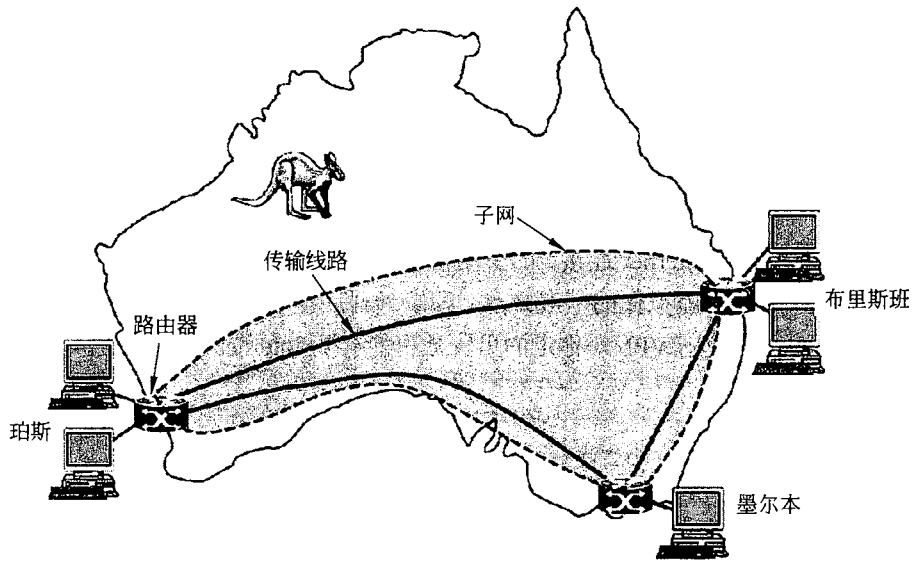


图 1-10 连接澳大利亚 3 个分支办事处的 WAN

在大多数广域网中，子网由两个不同部分组成：传输线路和交换元素。**传输线路**（transmission line）负责在机器之间移动比特。它们可以是铜线、光纤，甚至无线链路。大多数公司没有铺设自己的传输线路，因此，它们从电信公司租赁传输线路。**交换元素**（switching element）或简称为**交换机**（switch）是专用的计算机，负责连接两条或两条以上的传输线路。当数据到达一条入境线路时，交换元素必须选择一条出境线路把数据转发出去。这些负责交换的计算机在过去有各种不同的名称，现在最常用的名称是**路由器**（router）。不幸的是，有些人把这个词发音成“rooter”，而其他人则发“doubter”。确定正确的发音留给读者作为练习题（注：所谓正确的答案可能取决于你居住的地区）。

关于“子网”的含义我们这里给出一个简短的注释。最初，子网的唯一含义是一组路由器和通信线路的集合，主要负责将数据包从源主机移动到目标主机。读者应该知道，它已经拥有第二个并且更新的含义，那就是与网络寻址紧密相关。我们将在第 5 章讨论这个新含义，第 5 章之前我们都沿用这个原有的含义（即一组线路和路由器的集合）。

我们所描述的广域网看起来类似一个大型的有线局域网，但除了线路更长之外也有一些非常重要的差异。通常在广域网中，主机和子网是由不同的人拥有和经营。在我们的例子中，员工们仅仅负责自己的计算机，而该公司的 IT 部门负责网络的其余部分。我们在未来的例子中将看到更清晰的界限，其中子网由网络提供商或者电话公司负责经营。把网络中纯粹的通信方面（子网）与应用方面（主机）分离开来将大大简化整个网络的设计。

广域网和局域网的第二个区别是路由器通常连接不同类型的网络技术。例如，办公室内部网络可能是以太网，而长途传输线路可能是SONET链路（我们将在第2章讨论），这里显然需要某些设备将它们结合在一起。细心的读者会发现这个任务超出了我们对网络的定义。这意味着许多广域网是事实上的**互联网络**（internetwork），或者复合网络，即由多个网络组成的网络。我们将在下一节更多地介绍有关互联网络的内容。

广域网和局域网的最后一个差异在于子网连接什么。子网可以连接单个计算机，就像连接到局域网的情形一样，或者连接整个局域网。这说明了大型网络是如何从小网络构造出来的。只要涉及子网，基本上都做同样的工作。

我们现在来看两种不同类型的广域网。第一种广域网，公司并不租赁专用的传输线路，而是把自己的办事处直接连接到Internet。在这种方式下，办事处之间可以通过虚拟链路相互连接，而这些虚拟链路使用了底层Internet的容量。这样的安排如图1-11所示，称为**虚拟专用网络**（VPN，Virtual Private Network）。相比租赁专线，VPN具有虚拟化的一贯优势，它提供了重用某种资源（Internet连接）的灵活性。考虑添加第四个办事处就能看到这是多么的容易。当然，VPN也有虚拟化的一般缺点，即缺乏对底层资源的控制。采用专用线路能获得的容量是明确的，而使用VPN，你走的里程数可能会随Internet服务的变化而有所不同。

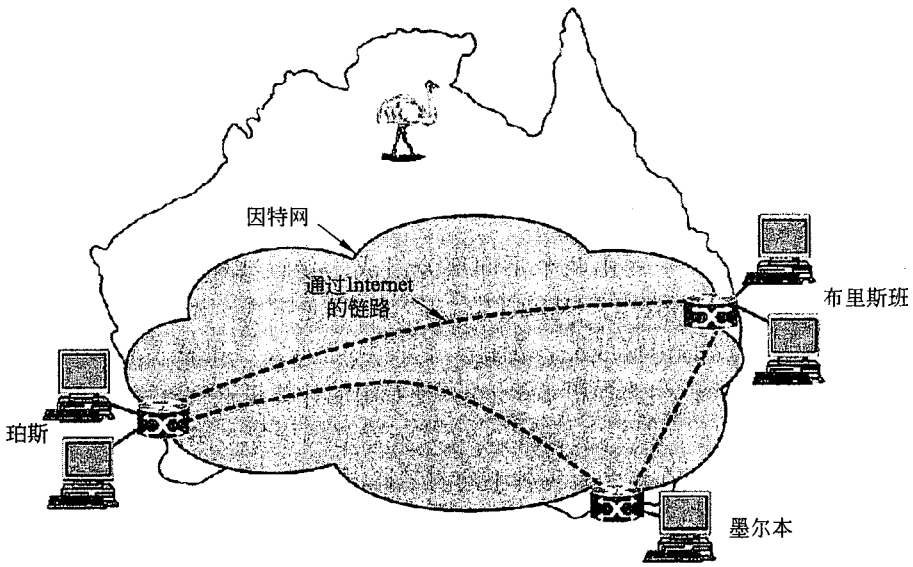


图1-11 使用虚拟专用网络的WAN

在第二种不同的广域网中，子网由不同的公司负责运营。子网经营者称为**网络服务提供商**（network service provider），公司办事处是它的客户。这种结构如图1-12所示。其他客户只要能够支付费用并且它能提供客户所需的服务，子网运营商就把这些客户也连接进来。如果客户只能给连接在同一个网络内的其他客户发送数据包，那么这将是一个令人失望的网络服务，因此子网运营商还与Internet的其他网络相连。这样的子网运营商称为**Internet服务提供商**（ISP，Internet Service Provider），相应的子网称为**ISP网络**（ISP network）。连接到ISP的客户就能享受Internet服务。

我们可以利用ISP网络来预览一些将在后面章节中研究的关键问题。在大多数广域网

中，网络包含了许多传输线路，每条线路连接一对路由器。如果两个想通信的路由器没有共享同一条传输线路，那么它们必须通过其他路由器间接地进行通信。网络中可能存在许多条路径都可以连接这两个路由器。网络如何决定使用哪条路径的策略称为**路由算法**（routing algorithm），这样的算法有许多。每个路由器如何决定把一个数据包发送到哪个位置的策略称为**转发算法**（forwarding algorithm），这样的算法也有许多。我们将在第 5 章中详细研究这两类算法。

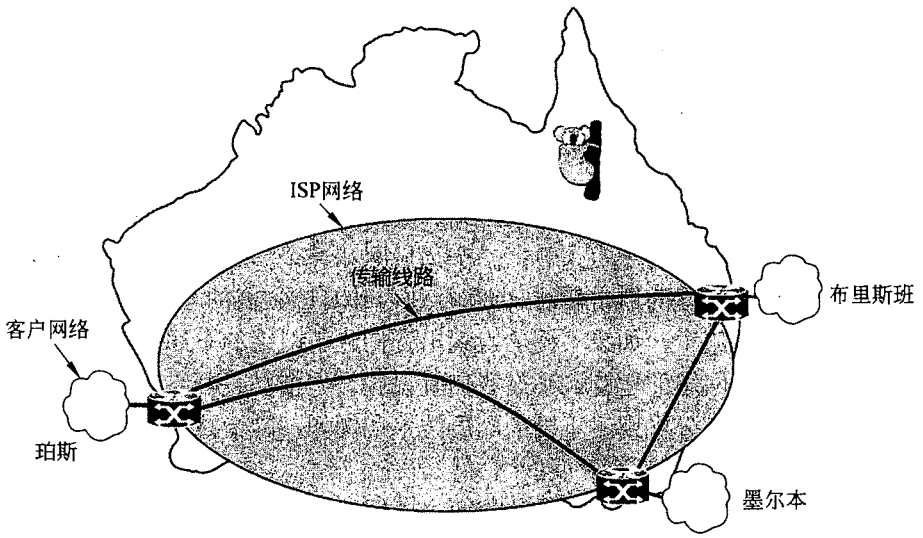


图 1-12 使用一个 ISP 网络的 WAN

其他种类的广域网使用了大量的无线技术。在卫星系统中，地面上的每台计算机都有一个天线，通过它给轨道上的卫星发送数据和接收来自卫星的数据。所有计算机都可以侦听到卫星的输出，而且在某些情况下，还能侦听到同胞计算机向上给卫星的传输。卫星网络在本质上是广播的，在某些广播属性很重要的情况下显得特别有用。

蜂窝移动电话网络是采用无线技术的另一个广域网例子。该系统已经经历了三代，而且第四代已崭露头角。第一代是模拟的，只能传语音。第二代是数字的，也只能传语音。第三代还是数字的，但可同时传语音和数据。每个蜂窝基站的覆盖范围大于无线局域网的覆盖范围，一般用公里来度量而不是几十米。基站通过一个骨干网络连接在一起，这个骨干网络通常是有线的。蜂窝网络的数据传输速率一般为 1 Mbps 左右的量级，远远小于高达 100 Mbps 量级的无线局域网。我们在第 2 章中将详细介绍有关这些网络的内容。

1.2.5 互联网络

世界上存在着许多许多的网络，它们常常使用不同的硬件和软件。连接到一个网络中的人经常要与连接到另一个网络中的人通信。为了做到这一点，那些相互之间不同而且通常不兼容的网络必须能够连接起来。一组相互连接的网络称为**互联网络**（internetwork）或**互联网**（internet）。这些术语一般具有通用意义，而全球范围的**因特网**（Internet）则通常用首字母大写来表示（这是一个特殊的互联网络）。Internet 使用 ISP 网络来连接各种各样

的企业网络、家庭网络和许多其他网络。我们在本书的后面章节将大量讨论 Internet。

子网、网络和互联网络经常发生混淆。术语“子网”通常在广域网的上下文中才有意义，它指网络运营商所拥有的一组路由器和通信线路。与此类似的是电话系统，它由各个电话交换局组成；电话交换局相互之间通过高速线路连接，而电话交换局通过低速线路连接到大量的住宅和办公楼。这些线路和设备构成了电话系统的子网，它们属于电话公司，并由电话公司负责管理。电话本身（相当于网络中的主机）并不是子网的一部分。

一个子网和它的主机结合在一起就形成了一个网络。然而，“网络”（network）这个词的使用往往具有比较松散的含义。一个子网可以是一个网络，如图 1-12 所示的那种“ISP 网络”。一个互联网络也可以是一个如图 1-10 所示的广域网。我们将遵循类似的做法，如果要将网络与其他安排区分开来，我们将坚持网络最初的定义，即由一种单一技术相互连接在一起的计算机集合。

让我们来看看是什么构成了一个互联网络。我们知道，当不同的网络相互连接在一起时就形成了互联网。从这样的观点来看，一个局域网和一个广域网相连，或者把两个局域网连接起来是构成互联网络的惯常做法，但在行业内针对该领域的术语很难达成一致意见。两个经验原则很有用。第一，如果不同的组织出资构建了网络的不同部分，并且各自维持自己出资构建的那部分网络的运营，那么我们就说这是一个互联网络，而不是单个网络；第二，如果网络的不同部分采用了不同的底层技术（例如，广播技术与点到点链路及有线与无线），那么我们就说这是一个互联网络。

继续深入下去，我们必须谈到如何连接两个不同的网络。将两个或多个网络连接起来并提供必要转换的机器，其硬件和软件方面的总称是**网关**（gateway）。工作在协议不同层次的网关是有所区别的。我们从下一节开始将更多地谈到层和协议的层次结构，但现在可以想象，层次越高与应用程序捆绑得就越紧密，比如 Web 应用；而下层则与传输链路相关，比如以太网。

由于形成互联网带来的好处是可以把不同网络上的计算机连接起来，我们不希望使用太低层次的网关，否则我们将无法把不同类型的网络连接起来。同时，我们也不希望使用太高层次的网关，否则只能互连特定的应用程序。“恰到好处”的中间层通常称为网络层，路由器是一个网关，它在网络层交换数据包。现在，我们可以通过找到一个具有路由器的网络来定位/辨别一个互联网。

1.3 网络软件

最初，计算机网络设计主要考虑的是硬件，其次考虑的才是软件。这种策略不再行得通。现在的网络软件已经高度结构化。在这接下来的几小节中，我们将考察软件构造技术的一些细节。这里描述的方法构成了本书的基石，以后还将会多次重复提到。

1.3.1 协议层次结构

为了降低网络设计的复杂性，绝大多数网络都组织成一个**层次栈**（a stack of layer）或

分级栈（a stack of level），每一层都建立在其下一层的基础之上。层的个数、每一层的名字、每一层的内容以及每一层的功能各个网络不尽相同。每一层的目的是向上一层提供特定的服务，而把如何实现这些服务的细节对上一层加以屏蔽。从某种意义上讲，每一层都是一种虚拟机，它向上一层提供特定的服务。

这种分层的概念实际上并不陌生，它已被广泛应用于计算机科学领域中，只是具有不同的称谓，包括信息隐藏、抽象数据类型、数据封装以及面向对象程序设计。其基本思想是一个特定的软件（或硬件）向其用户提供某种服务，但是将内部状态和算法的细节隐藏起来。

一台机器上的第 n 层与另一台机器上的第 n 层进行对话，该对话中使用的规则和约定统称为第 n 层协议。基本上，所谓**协议**（protocol）是指通信双方就如何进行通信的一种约定。作个比喻，当一位女士被介绍给一位先生时，她可能会选择伸出她的手；然后，这位男士可以握她的手或者亲吻她的手，具体的行为要取决于（比如说）她是一次商务会议中的美国律师还是一场正式舞会上的欧洲公主。任何一方违反协议将使得通信更加困难，如果不是完全不可能的话。

图 1-13 显示了一个 5 层网络。不同机器上构成相应层次的实体称为**对等体**（peer）。这些对等体可能是软件过程、硬件设备，或者甚至是人类。换句话说，正是这些对等体为了实现彼此沟通才使用协议来进行通信。

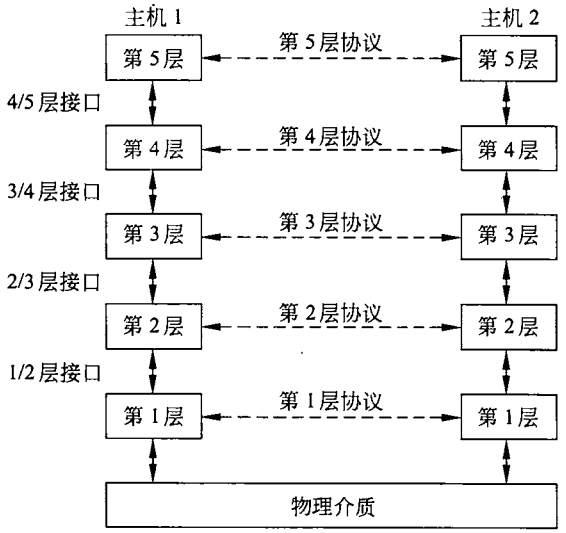


图 1-13 层次、协议和接口

实际上，数据并不是从一台机器的第 n 层直接传递到另一台机器的第 n 层。相反，每一层都将数据和控制信息传递给它的下一层，这样一直传递到最低层。第 1 层下面是**物理介质**（physical medium），通过它进行实际的通信。在图 1-13 中，虚线表示虚拟通信，实线表示物理通信。

在每一对相邻层次之间的是**接口**（interface）。接口定义了下层向上层提供哪些原语操作和服务。当网络设计者在决定一个网络中应该包含多少层，以及每一层应该提供哪些功能时，其中最重要的一个考虑是必须定义清楚层与层之间的接口。为了做到这一点，要求

每一层完成一组特定的有明确含义的功能。除了尽可能地减少层与层之间必须要传递的信息量外，层与层之间清晰的接口使得同层协议的替换更加容易，即某一层的当前协议或实现替换成另一个完全不同的协议或者实现 （比如说，所有的电话线路被替换成卫星信道）；因为对于新协议或新实现来说，它所要做的仅仅是向紧邻的上层提供与旧协议或者旧实现完全相同的服务。一般来说，对于同一个协议不同的主机使用了不同的实现（经常由不同的公司编写代码）。事实上，某个层次的协议本身是可以改变的，无须通知上层和下层。

层和协议的集合称为**网络体系结构**（network architecture）。网络体系结构的规范必须包含足够的信息，以便实现者为每一层编写的程序或者设计的硬件能遵守有关的协议。实现细节和接口规范不属于网络体系结构的内容，因为它们隐藏在机器内部，对于外界是不可见的。甚至，一个网络中所有机器上的接口也不必都一样，只要每台机器能够正确地使用所有的协议即可。一个特定的系统所使用的一组协议，即每一层一个协议，称为**协议栈**（protocol stack）。网络体系结构、协议栈以及协议本身是本书的主要内容。

打个比方也许有助于解释网络体系结构中多层通信的概念。假设有两位哲学家（第 3 层中的对等进程），其中一个会讲乌尔都语（Urdu）和英语，另一个会讲汉语和法语。由于他们两人没有共同会讲的语言，所以他们每个人都雇用了一个翻译（第 2 层中的对等进程），每个翻译又反过来联系了一位秘书（第 1 层中的对等进程）。哲学家 1 希望将他对兔子的感情（oryctolagus cuniculus）传达给哲学家 2。为了做到这一点，他将一条消息（英语）通过 2/3 层之间的接口传递给他的翻译，这条消息说“I like rabbits”，如图 1-14 所示。两位翻译同意用一种双方都能理解的中立语言交流，即荷兰语，所以这条消息被转译为“Ik vind konijnen leuk”。语言的选择是第 2 层协议的事情，所以使用什么样的语言由第 2 层的对等进程决定。

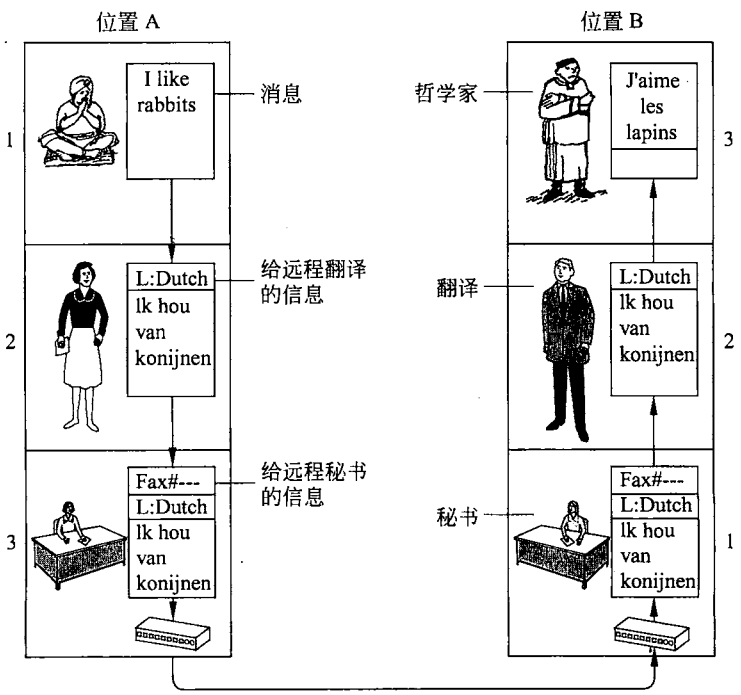


图 1-14 “哲学家-翻译-秘书”体系结构

然后，翻译将消息交给秘书，让她传送出去，比如说，通过电子邮件来传送（这是第1层协议）。当消息到达另一端的翻译时，被翻译成法语；然后通过2/3层之间的接口传送给哲学家。注意，每个协议都完全独立于其他的协议，只要接口不变就能正常工作。两个翻译可以从荷兰语切换成另一种语言，比如说芬兰语，只要双方能够达成一致，并且不改变与第1层和第3层的接口。类似地，秘书可以不用电子邮件而改用电话，根本不会干扰（或者甚至不用通知）其他层次。每个进程还可以加入某些专门给它对等实体的信息。这部分信息不会被传递到上一层。

现在考虑一个更加技术性的例子：如何向图1-15中5层网络的最顶层提供通信。假设在第5层上运行的一个应用进程产生了一条消息M，并且将它传递给第4层以便传给对等进程；第4层在消息的前面加上一个头（header），用来标识该消息，并且把结果传给第3层；该头包含了一些控制信息，例如地址，主要被目标机器的第4层用来递交消息。某些层次所用的控制信息还可以包含消息序号（以防下层不保留消息顺序）、消息大小、时间等。

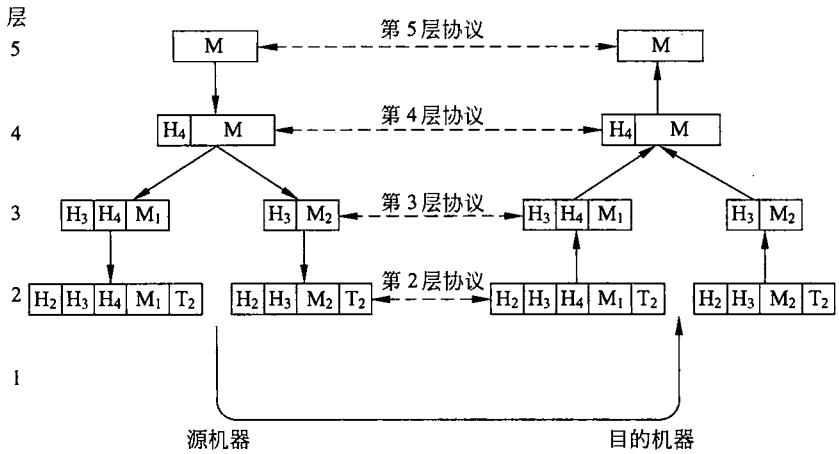


图1-15 支持第5层虚拟通信的信息流

在许多网络中，对于第4层上传递的消息大小没有任何限制，但是几乎所有第3层协议对此总会强加一个限制。因此，第3层必须把入境消息分割成较小的单元，即**数据包**或**包**（packet），并且在每个数据包前面加上第3层的头。在这个例子中，M被分割成两部分：M_1和M_2，这两部分内容是被单独传输的。

第3层决定使用哪些输出线路，并且把分组传递给第2层；第2层不仅在每一个信息上加上一个头信息还要加上一个尾，然后将结果传输单元送给第1层以便进行物理传输。在接收端的机器上，消息自底向上逐层传递，在传递过程中各个头被逐层剥离。没有一个第n层以下的头会被传递到第n层。

正确理解图1-15的最重要事情是明白虚拟通信和实际通信之间的关系，以及协议和接口之间的关系。例如，第4层中的对等进程在概念上可以认为它们的通信是“水平的”，它们使用了第4层协议。每一个对等进程可能都有一个类似于SendToOtherSide和GetFromOtherSide这样的过程，但这些过程实际上是通过3/4层之间的接口与底层进行通信，并不是直接与另一端进行通信的。

关于对等进程的抽象概念对于所有的网络设计都至关重要。利用对等进程的思想，在

设计整个网络时，可以把难以管理的任务分解成几个较小的、易于处理的设计问题，这就是分层设计。

尽管1.3节名为“网络软件”，但值得指出的是，协议层次结构中的较低层往往由硬件或者固件实现。然而，即使它们被（全部或者部分）嵌入到硬件中，仍然会涉及复杂的协议算法。

1.3.2 层次设计问题

计算机网络的一些关键设计问题会一层接着一层地出现。下面，我们将简要地提一些更重要的问题。

可靠性是保证网络正常运行的设计问题，即使网络由本身并不可靠的组件所构成。请想象数据包中的比特穿越网络时的情形。由于存在电气噪声、随机无线信号、硬件缺陷、软件错误等原因，其中的某些比特到达接收端时已经遭到了损坏（即被逆转了）。我们发现并修复这些错误的可能性如何？

从接收到的信息中发现错误所用的一种机制是**检错**（error detection）编码；然后重新传输接收到的不正确信息，直到它被正确接收为止。更强大的编码不仅能检错，还能**纠错**（error correction），即从最初收到的可能不正确的比特中恢复正确的消息。这两种机制的工作都需要在被传的信息中添加冗余信息。这些冗余信息被较低层次用来保障数据包在个别链路上的正确传输，也可被较高层次用来检测接收到的数据包是否包含了正确的内容。

另一个可靠性问题是找到通过网络的工作路径。在源和目的地之间经常存在多条路径，而且在一个大型网络中可能有一些链路或路由器偶尔发生故障。假设德国的网络出现了故障，那么从伦敦发送到罗马的数据包如果选择一条经过德国的路径将注定无法通过，但我们可以把从巴黎发往罗马的数据包改道经过伦敦到达罗马。网络应该能自动做出这种路由决策。这个主题就是所谓的**路由**（routing）。

第二个设计问题涉及网络演进。随着时间的推移，网络的增长越来越大，出现的新设计需要与现有网络连接。我们刚刚了解到支撑这种变化的关键结构化机制，即将整个问题分解开来，进而分而治之，并且隐藏实现的细节：协议分层。当然，还有许多其他的策略。

由于网络上有许多计算机，每一层在特定的消息中都需要一种机制来标识发送方和接收方。这种机制在下层和高层分别称为**寻址**（addressing）和**命名**（naming）。

网络增长的一个方面体现在不同的网络技术往往具有不同的限制。例如，并非所有的通信信道都能维持在其上发送消息的顺序，这个问题导致了对消息进行编号的一些解决方案。另一个例子是网络能够传输的消息的最大长度差异，这又导致了分段机制的出现，对消息进行拆分、传输，然后重组。所有这些主题综合起来就是所谓的**网络互联**（internetworking）。

当网络规模变大时又会出现新的问题。城市可能出现交通堵塞、缺少可用的电话号码，并且很容易迷路。没有多少人会在自己家附近发生这些问题，但扩大到全市范围内这些问题可能就是个大问题。网络规模变大时仍能工作良好的设计被说成是**可扩展的**（scalable）。

第三个设计问题是资源分配。网络基于其底层的资源（比如传输线路的容量）向主机提供服务。要做好这些工作，它们需要一些分配资源的机制，使得一台主机不会太多地干扰到另一台主机。

许多网络设计根据主机的短期需求变化动态共享网络带宽，而不是给每个主机分配可能用也可能不会用的固定比例带宽。这种设计称为**统计复用**（statistical multiplexing），意味着根据统计需求来共享带宽。这种复用可以用在低层次的单条链路上，或者较高层次的网络层，甚至用在网络的应用层上。

在每一层都会发生的一个分配问题是，如何保持快速发送方不会用数据把慢速接收方淹没。这个问题的解决经常使用了从接收方到发送方的反馈机制。这个主题就是**流量控制**（flow control）。有的时候还会出现网络超载问题，因为太多的计算机要发送太多的流量，而网络又没有能力传递所有的数据包。这样的网络超载称为**拥塞**（congestion）。一种策略是当发生拥塞时，每台计算机都减少其对网络的带宽需求。这种策略可用于所有层次。

有趣的是我们可以观察到网络已经不单只有简单的带宽，它可以提供更多的资源。对于诸如传递视频直播的应用来说，传递的及时性非常重要。大多数网络必须为那些需要这种**实时**（real-time）传递的应用程序提供服务，与此同时，它们还必须为那些要求高吞吐量的应用程序提供服务。**服务质量**（Queality of service）是给予调和这些竞争需求机制的名称。

最后一个主要的设计问题是如何保护网络抵御各种不同的威胁。我们前面提到的一种威胁是通信窃听。提供**保密性**（confidentiality）的机制能抵御这种威胁，并且它们可被用在多个层次。**认证**（authentication）机制防止有人假冒他人身份，它们可以用来告知你一个伪装成真实银行的假银行网站，或者蜂窝网络用来检查一个电话真的是从你的手机打出去因而必须由你来支付电话账单。其他的**完整性**（integrity）机制则可以防止消息发生诡秘的变化，比如把“从我的账号中扣除 10 美元”改成“从我的账号中扣除 1000 美元”。所有这些设计都基于加密技术，我们将在第 8 章学习这方面的知识。

1.3.3 面向连接与无连接服务

下层可以向上层提供两种不同类型的服务：面向连接的服务和无连接的服务。在这一小节中，我们将介绍这两种类型的服务，并且考察两者之间的区别。

面向连接的服务（connection-oriented service）是按照电话系统建模的。当你想给某人打电话时，首先要拿起话机，拨对方的号码，然后说话，最后挂机。类似地，为了使用面向连接的网络服务，服务用户首先必须建立一个连接，然后使用该连接传输数据，最后释放该连接。这种连接最本质的方面在于它像一个管道：发送方把对象（数据位）压入管道的一端，接收方在管道的另一端将它们取出来。在绝大多数情况下，数据位保持原来的顺序，所以数据位都会按照发送的顺序到达。

在有些情况下，当建立一个连接时，发送方、接收方和子网一起**协商**（negotiation）一组将要使用的参数，比如最大的消息长度、所要求的服务质量以及其他一些问题。一般情况下，一方提出一个建议，另一方接受或拒绝该建议，甚至提出相反的建议。**电路**（circuit）是与资源关联的连接的另一个名字，比如具有固定的带宽。这个概念可追索到电话网络，在电话网络中，一条电路就是通过铜线传送电话交谈内容的路径。

与面向连接服务相对应的是**无连接服务**（conncetionless service），这是按照邮政系统建模的。每个报文（信件）都携带了完整的目标地址，每个报文都由系统中的中间节点路由，

而且路由独立于后续报文。**报文**（message）在不同的上下文中有不同的称呼：数据包/**包**（packet）是网络层的报文。如果中间节点只能在收到报文的全部内容之后再将该报文发送给下一个节点，那么我们就称这种处理方式为**存储-转发交换**（store-and-forward switching）。有别于此的另一种处理方式是在报文还没有被全部接收完毕之前就向下一个节点传输，这种处理方式称为**直通式交换**（cut-through switching）。通常来说，当两个报文被发往同一个目的地时，首先被发送的报文将会先到达。然而，先发送的报文可能被延迟，因而后发送的报文比它先到达，这种情况也是有可能发生的。

每个服务都可以用一个**服务质量**（quality of service）来表述其特征。有些服务是可靠的，意味着它们从来不丢失数据。一般情况下，一个可靠服务是这样实现的：接收方向发送方确认收到的每个报文，因而发送方可以据此保证报文已经到达接收方。确认过程本身要引入额外的开销和延迟，通常这是值得的，但有时也不一定需要。

最适合用可靠的面向连接服务的一种典型情形是文件传输。文件的拥有者希望保证所有的数据位都能够正确地到达接收方，而且到达的顺序与发送的顺序相同。如果一种文件传输服务偶尔会出现乱码或者丢失数据位，即使它的传输速度再快，恐怕也很少有客户会愿意使用它。

可靠的面向连接服务有两个细微的变异形式：报文序列和字节流。在前一种变异中，报文的边界始终得到保持。发送两个 1024 字节的报文，收到的仍然是两个独立的长度为 1024 字节的报文，而绝不可能变成一个长度为 2048 字节的报文。在后一种变异中，该连接只是一个字节流，没有任何报文边界。当 2048 个字节到达接收方时，接收方无法判断发送方发出的是一个长度为 2048 字节的报文，还是两个长度为 1024 字节的报文，或者是 2048 个长度只有 1 字节的报文。如果一本书的每一页都当作单独的一个报文传输，通过网络发送给一台照排机，此时保持报文的边界可能非常重要。另一方面，在下载一个 DVD 电影时，只需要一个从服务器到用户计算机的字节流。此时电影中的报文边界并不相关。

对于有些应用，因确认而引入的传输延迟是不可接受的。比如说 **IP 语音**（voice over IP）采用的数字语音传输就是这样一个应用。对于电话用户来说，他们宁可时不时地听到线路上有一点噪音，也无法容忍因等待确认而带来的延迟。类似地，当传输一个视频会议时，有少数的像素错误不算什么问题，但是让图像停顿下来纠正传输错误则让人不可接受。

并不是所有的应用都要求面向连接的服务。例如，垃圾邮件发送者给许多用户发送电子垃圾邮件，它们可能并不愿意只为了发送一个邮件就建立连接，然后再拆除连接。百分之百的可靠递交并不那么重要，特别是当它需要付出更多的代价时。所需要的只是一种发送单个报文的方法，这个报文将以极高的概率到达目的地，但不是必须确保到达目的地。不可靠（意味着没有被确认）的无连接服务通常称为**数据报服务**（datagram service），它与电报服务非常类似，一般不会给发送方反馈任何确认消息。尽管它是不可靠的，但在大多数网络中这是一种占主导地位的传输形式，至于个中的原因我们马上就会明白。

在其他情形下，的确需要这种无须建立连接就可发送一个报文的便利性，但是可靠性仍然是基本需求。**有确认的数据报服务**（acknowledged datagram service）就是为这些应用提供的一类服务。这就像在寄挂号信时要求一个回执一样。当发送方收到回执时可绝对相信这封信已经被送到对方手中，肯定没有在投递的途中丢失。手机上的文本消息就是一个实例。

还有另外一种服务是**请求-应答服务**（request-reply service）。在这种服务中，发送方传输一个包含了某个请求的数据报；接受方以一个包含了请求结果的应答数据报作为反馈。请求-应答服务通常用在客户机-服务器模型中：客户机发出一个请求，然后服务器对此做出响应。例如，一个手机客户向地图服务器发出一个请求，要求查询其当前位置的地图数据。图 1-16 概括了上面讨论的服务类型。

	服务	例子
面向连接	可靠的报文流	顺序页面
	可靠的字节流	移动下载
	不可靠的连接	IP语音
无连接	不可靠的数据报	垃圾邮件
	有确认的数据报	文本消息
	请求-应答	数据库查询

图 1-16 6 种不同类型的服务

使用不可靠连接的概念刚开始可能会令人感到疑惑不解。毕竟，为什么会有人愿意舍弃可靠通信而选择不可靠的通信呢？首先，在给定的层次并不是总能使用可靠通信（可靠通信的含义是指有确认）。例如，以太网并没有提供可靠通信。有些数据包可能偶尔会在传输过程中被损坏，上层协议必须处理这个问题。尤其是，许多可靠服务是建立在不可靠数据报服务之上的。其次，为了提供可靠服务而引入的固有延迟可能令人不可接受，特别在诸如多媒体的实时应用中，更是无法忍受额外的延迟。由于这些原因，可靠通信和不可靠通信将并存在任何一个网络中。

1.3.4 服务原语

一个服务由一组**原语**（primitive）正式说明，用户进程通过这些原语（操作）来访问该服务。原语告诉服务要执行某个动作，或者将对等实体所执行的动作报告给用户。如果协议栈位于操作系统中（大多数情况是这样的），则这些服务原语通常是一些系统调用。这些系统调用会陷入内核模式，然后在内核模式中取代操作系统来控制机器发送必要的数据包。

可用的原语取决于底层所提供的服务。面向连接服务的原语与无连接服务的原语是不同的。举一个最小的服务原语例子，为了实现一个可靠的字节流，可以考虑采用如图 1-17 所示的原语。这些原语对于 Berkeley 套接字接口迷非常熟悉，因为原语基本上就是这种接口的简化版本。

原语	含义
LISTEN	阻塞操作，等待入境连接请求
CONNECT	与等待中的对等实体建立连接
ACCEPT	接受来自对等实体的入境连接请求
RECEIVE	阻塞操作，等待入境报文
SEND	给对等实体发送一个报文
DISCONNECT	终止一个连接

图 1-17 为简单面向连接服务提供的 6 个服务原语

这些原语在客户机-服务器环境下可用来实现“请求-应答”交互式应用。为了展示原语是如何工作的，我们给出一个有确认数据报服务的简单协议实现。

首先，服务器执行 LISTEN，表示它已经准备好接收入境连接。通常实现 LISTEN 的方式是将它做成一个阻塞的系统调用。在执行了 LISTEN 原语之后，服务器进程被阻塞，直到有连接请求到来为止。

接着，客户进程执行 CONNECT 原语，以便与服务器建立连接。CONNECT 调用需要指定跟谁建立连接，所以它可能带有一个参数来指定服务器的地址。然后，操作系统通常会向对方发送一个数据包，请求建立连接，如图 1-18 中（1）所示。客户进程被挂起，直到有应答为止。

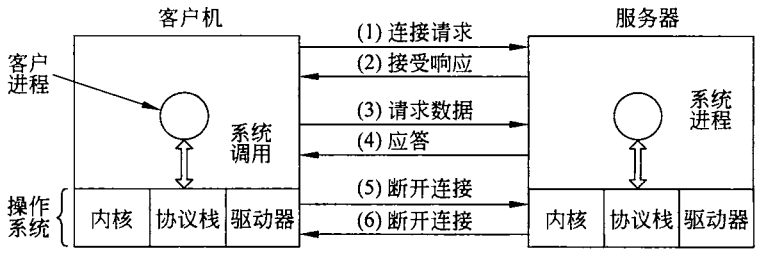


图 1-18 使用有确认数据报的一个简单客户机-服务器交互过程

当该数据包到达服务器时，操作系统看到这是一个请求连接包，它就检查是否存在一个监听进程。如果刚好有个监听进程，它就解除该监听进程的阻塞，然后用 ACCEPT 调用创建连接。为了接受连接，服务器给客户进程发送一个响应，如图 1-18 中（2）所示。接收到该响应报文后客户进程就得到解除恢复运行状态。这时候客户机和服务器都在运行，并且已经建立了一个连接。

在现实生活中，与此协议类似的一种情况是消费者（客户）打电话给一家公司的客户服务经理。每天开始上班后，服务经理就坐在他的电话机旁准备接电话。稍后，一个客户拨了电话，当服务经理拿起电话后，他和客户之间的连接便建立起来了。

下一步是服务器执行 RECEIVE，准备接收第一个请求。通常情况下，一旦服务器从 LISTEN 的阻塞状态被释放后，会马上执行 RECEIVE，这时确认消息往往还没有到达客户方。RECEIVE 调用再次阻塞了服务器

然后，客户执行 SEND 发送它的请求，如图 1-18 中（3）所示，接着执行 RECEIVE 等待服务器应答。请求数据包到达服务器后，首先操作系统解除服务器进程的阻塞，使得服务器进程能处理该客户请求。处理完该请求工作后，服务器通过 SEND 将请求的执行结果送回给客户，如图 1-18 中（4）所示。该数据包到达客户后，客户进程被操作系统解除阻塞，然后检查服务器返回的结果。如果客户还有其他的请求，它现在可以继续发送这些请求。

如果客户的任务已经完成，则它通过执行 DISCONNECT 终止当前连接，如图 1-18 中（5）所示。一般情况下，发起 DISCONNECT 的是一个阻塞调用，该调用将客户进程挂起，并且给服务器发送一个包，表明客户已经不再需要该连接，如图 1-18 中（5）所示。在服务器收到这个包后，它也发出一个自己的 DISCONNECT，作为对客户终止连接的确认并释放该连接，如图 1-18 中（6）所示。在服务器的包到达客户机后，客户进程被解除阻塞恢复运行，至此该连接被正式终止。综上所述，这就是面向连接的通信过程。

当然，生活不会总是这么简单。这里很多地方都有可能出现错误情况。时间顺序可能会出错（比如 CONNECT 在 LISTEN 之前完成）、数据包可能会丢失，等等。我们将在后面的章节中详细讨论这些问题以及如何处理的解决方案。而此刻图 1-18 只是简要地说明了客户机-服务器之间一次基于有确认数据报的通信过程，采用有确认的数据报服务，因而忽略了数据包的丢失情况。

既然完成这个协议总共需要 6 个分组，人们可能会疑惑为什么不使用无连接协议呢？这个问题的答案是在一个完美的环境下采用无连接协议是可以的，此时只需要两个数据包：一个用于请求消息，另一个用于应答消息。然而，无论在哪个方向上只要出现很大的报文（比如 1 MB 大小的文件），传输错误、丢失数据包等都有可能发生，因此情况就发生变化了。如果应答消息包含了几百个数据包，其中一些数据包可能会在传递过程中丢失，那么，客户如何知道是否有数据丢失了呢？客户如何知道最后接收到的数据包就是服务器真正发送的最后一个数据包呢？假如客户还要请求第二个文件，那么客户如何区别接收到的数据包 1 到底属于第二个文件，还是属于第一个文件（因被延迟而迟到的）呢？简而言之，实际上，在一个不可靠的网络上，简单的“请求-应答”协议往往是无法胜任工作需要的。在第 3 章中，我们将详细地学习各种协议，看看它们如何解决这些问题以及其他的一些问题。现在，我们只要明白在两个进程之间有一个可靠的、有序的字节流，有时候真的是非常的方便。

1.3.5 服务与协议的关系

服务和协议是两个截然不同的概念，它们之间的区别非常重要，我们有必要在这里再次强调。服务是指某一层向它上一层提供的一组原语（操作）。服务定义了该层准备代表其用户执行哪些操作，但是它并不涉及如何实现这些操作。服务与两层之间的接口有关，低层是服务提供者，而上层是服务用户。

与此不同的是，协议是一组规则，规定了同一层上对等实体之间所交换的数据包或者报文的格式和含义。对等实体利用协议来实现它们的服务定义，它们可以自由地改变协议，只要不改变呈现给它们用户的服务即可。按照这种方式，服务和协议是完全相分离的，这是任何一个网络设计者应该很好理解的关键概念。

这里为了重复一下这个关键点，服务涉及层与层之间的接口，如图 1-19 所示。相反，协议涉及不同机器上两个对等实体之间发送的数据包。不能混淆这两个概念，这点非常重要。

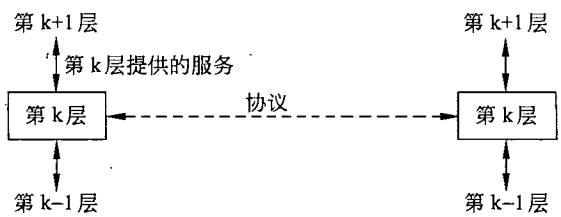


图 1-19 服务和协议之间的关系

值得用编程语言来对这两个概念作一个类比。服务就好像是面向对象语言中的抽象数据类型或者对象，它定义了在对象上可以执行的操作，但是并没有说明如何实现这些操作。

而协议与服务的具体实现有关，它对于该服务的用户是完全不可见的。

许多老的协议并没有将服务和协议区分开。实际上，一个典型的层可能有一个“SEND PACKET”服务原语，并且该原语有个参数；参数就是用户提供的一个指针，指向一个已经完全装配好的数据包。这样的一种设计意味着协议的任何一点改变都立即会暴露给用户。现在，大多数的网络设计者都把这样的设计视为一种严重的失误。

1.4 参考模型

既然我们已经讨论了抽象的层次网络，现在是时间看一些网络实例了。我们将讨论两个重要的网络体系结构：OSI 参考模型和 TCP/IP 参考模型。虽然与 OSI 模型相关的协议没有被任何人所用，但实际上，该模型本身具有相当普遍意义，并仍然有效；它对讨论网络体系结构中每一层的功能还是很重要。TCP/IP 协议模型则具有相反的特性：模型本身没有多大用处，但它的协议却已经广为流传。正是基于这样的原因，我们将详细考察这两个模型。而且，有时候你从失败中学习到的东西比从成功中学到的更多。

1.4.1 OSI 参考模型

OSI 模型如图 1-20 所示（省略了物理介质）。该模型基于国际标准化组织（ISO，International Standards Organization）的提案，作为各层协议迈向国际标准化的第一步（Day

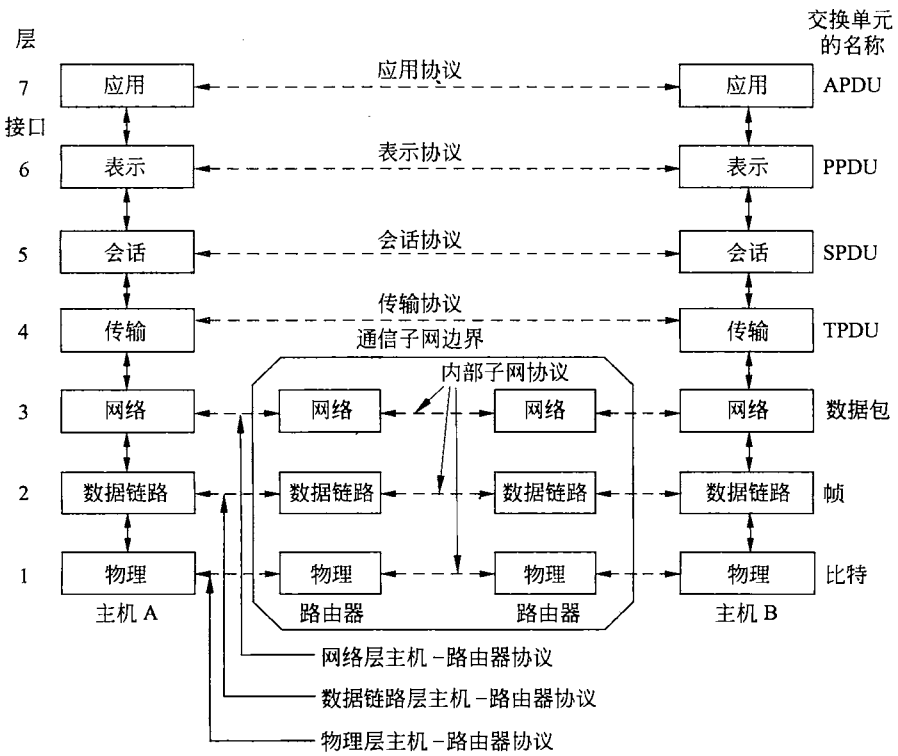


图 1-20 OSI 参考模型

和 Zimmermann，1983），并且于 1995 年进行了修订（Day，1995）。这个模型称为 ISO 的**开放系统互连**（OSI，Open Systems Interconnection）参考模型，因为它涉及如何连接开放的系统——即那些为了与其他系统通信而开放的系统。为了简短起见，我们将它简称为 OSI 模型。

OSI 模型有 7 层。适用于这 7 层的基本原则简要概括如下：

（1）应该在需要一个不同抽象体的地方创建一层。

（2）每一层都应该执行一个明确定义的功能。

（3）每一层功能的选择应该向定义国际标准化协议的目标看齐。

（4）层与层边界的选择应该使跨越接口的信息流最小。

（5）层数应该足够多，保证不同的功能不会被混杂在同一层中，但同时层数又不能太多，以免体系结构变得过于庞大。

下面我们从最底层开始，依次讨论该模型中的每一层。请注意，OSI 参考模型本身并不是一个网络体系结构，因为它并没有定义每一层的服务和所用的协议。它只是指明了每一层应该做些什么事。然而，ISO 已经为所有层都制定了相应的标准，虽然这些标准并不属于参考模型本身。每个协议都是作为单独的国际标准发布的。模型（部分地）得到了广泛的使用，尽管与之相关的协议早已被遗忘多时。

物理层

物理层（physical layer）关注在一条通信信道上传输原始比特。设计问题必须确保当一方发送了比特 1 时，另一方收到的也是比特 1，而不是比特 0。这里的典型问题包括用什么电子信号来表示 1 和 0、一个比特持续多少纳秒、传输是否可以在两个方向上同时进行、初始连接如何建立、当双方结束之后如何撤销连接、网络连接器有多少针以及每一针的用途是什么等。这些设计问题主要涉及机械、电子和时序接口，以及物理层之下的物理传输介质等。

数据链路层

数据链路层（data link layer）的主要任务是将一个原始的传输设施转变成一条没有漏检传输错误的线路。数据链路层完成这项任务的做法是将真实的错误掩盖起来，使得网络层看不到。为此，发送方将输入的数据拆分成**数据帧**（data frame），然后顺序发送这些数据帧。一个数据帧通常为几百个或者几千个字节长。如果服务是可靠的，则接收方必须确认正确收到的每一帧，即给发送方发回一个**确认帧**（acknowledgement frame）。

数据链路层（和大多数高层都存在）的另一个问题是如何避免一个快速发送方用数据“淹没”一个慢速接收方。所以，往往需要一种流量调节机制，以便让发送方知道接收方何时可以接收更多的数据。

广播式网络的数据链路层还有另一个问题：如何控制对共享信道的访问。数据链路层的一个特殊子层，即介质访问控制子层，就是专门处理这个问题的。

网络层

网络层（network layer）的主要功能是控制子网的运行。一个关键的设计问题是如何将数据包从源端路由到接收方。路由可以建立在静态表的基础上，这些表相当于网络内部的

“布线”，而且很少会改变；或者，更常见的情况是路由可以自动更新，以此来避免网络中的故障组件。路由也可以在每次会话（例如一次终端会话）开始时就确定下来，比如登录到一台远程机器上。最后，路由可以是高度动态的，针对每一个数据包都重新确定路径，以便反映网络当前的负载情况。

如果有太多的数据包同时出现在一个子网中，那么这些数据包彼此之间会相互阻碍，从而形成传输瓶颈。处理拥塞也是网络层的责任，一般要和高层协议结合起来综合处理拥塞才有效，高层协议必须适应它们注入网络中的负载。更普遍的是网络所提供的服务质量（延迟、传输时间、抖动等）也是网络层的问题。

当一个数据包必须从一个网络传输到另一个网络才能够到达它的目的地时，可能会发生很多问题。比如，第二个网络所使用的寻址方案可能与第一个网络不同；第二个网络可能无法接受这个数据包，因为它太大了；两个网络所使用的协议也可能不一样，等等。网络层应该解决所有这些问题，从而允许异构网络相互连接成为互联网络。

在广播式网络中，路由问题比较简单，所以网络层往往比较单薄，甚至根本不存在。

传输层

传输层（transport layer）的基本功能是接收来自上一层的数据，在必要的时候把这些数据分割成较小的单元，然后把这些数据单元传递给网络层，并且确保这些数据单元正确地到达另一端。而且，所有这些工作都必须高效率同时以一种上下隔离的方式完成，即随着时间的推移导致底层硬件技术不可避免地发生改变时，对上面各层是透明的。

传输层还决定了向会话层，因而是实际的最终网络用户提供哪种类型的服务。其中最为常见的传输连接是一个完全无错的点到点信道，此信道按照原始发送的顺序来传输报文或者字节数据。然而，其他类型的传输服务也有可能，例如传输独立的报文但不保证传送的顺序、将报文广播给多个目标节点等。服务的类型是在建立连接时就确定下来的（顺便说一下，完全无错的信道是不可能实现的；人们使用这个术语的真正含义是指出错率很低，小到足以忽略掉）。

传输层是真正的端到端的层，它自始至终将数据从源端携带到接收方。换句话说，源机器上的一个程序利用报文头和控制信息与目标机器上的一个类似程序进行会话。在其下面的各层，每个协议涉及一台机器与它的直接邻居，而不涉及最终的源机器和目标机器，即源机器和目标机器可能被多个中间路由器隔离了。第 1～3 层是链式连接的，而第 4～7 层是端到端的，两者之间的区别如图 1-20 所示。

会话层

会话层（session layer）允许不同机器上的用户建立会话。会话通常提供各种服务，包括**对话控制**（dialog control）（记录该由谁来传递数据）、**令牌管理**（token management）（禁止双方同时执行同一个关键操作），以及**同步功能**（synchronization）（在一个长传输过程中设置一些断点，以便在系统崩溃之后还能恢复到崩溃前的状态继续运行）。

表示层

表示层以下的各层最关注的是如何传递数据位，而**表示层**（presentation layer）关注的是所传递信息的语法和语义。不同的计算机可能有不同的内部数据表示法，为了让这些计

算机能够进行通信，它们所交换的数据结构必须以一种抽象的方式来定义，同时还应该定义一种“线上”使用的标准编码方法。表示层管理这些抽象的数据结构，并允许定义和交换更高层的数据结构（比如银行账户记录）。

应用层

应用层（application layer）包含了用户通常需要的各种各样的协议。一个得到广泛使用的应用协议是**超文本传输协议**（HTTP，HyperText Transfer Protocol），它是**万维网**（WWW，World Wide Web）的基础。当浏览器需要一个 Web 页面时，它通过 HTTP.将所要页面的名字发送给服务器，然后服务器将页面发回给浏览器。其他一些应用协议可用于文件传输、电子邮件以及网络新闻等。

1.4.2 TCP/IP 参考模型

现在我们从 OSI 参考模型转到另一个参考模型，该参考模型不仅被所有广域计算机网络的鼻祖 ARPANET 所采用，而且被其继任者——全球范围的 Internet 所使用。尽管后面我们还会简要地介绍 ARPANET 的历史，但是现在有必要先简短地引入 ARPANET 的一些关键要点。ARPANET 是由美国国防部（DoD，U.S. Department of Defense）资助的一个研究性网络。它通过租用的电话线，将几百所大学和政府部门的计算机设备连接起来。后来，当卫星和无线网络也要加入时，发现原来的协议在与它们互连时遇到了很大的麻烦，因而需要一种新的参考体系结构。所以，以无缝的方式将多个网络连接起来是从一开始就制定的主要设计目标之一。这个体系结构后来称为 **TCP/IP 参考模型**（TCP/IP Reference Model），以其中两个最主要的协议命名。该体系结构最初由（Cerf 和 Kahn，1974）描述，后来在（Leiner 等，1989）中又被重新修订并得到 Internet 团体的标准化。TCP/IP 模型背后的设计思想在（Clark，1988）中进行了详细讨论。

由于美国国防部担心其一些贵重的主机、路由器和互联网络的网关可能会在片刻间被来自前苏联的攻击而突然崩溃，所以，另一个主要的设计目标是即使在损失子网硬件的情况下网络还能够继续工作，原有的会话不能被打断。换句话说，美国国防部希望，即使源机器和目标机器之间的一些机器或者传输线路突然不能工作，只要源机器和目标机器还在运作，那么它们之间的连接就要维持不变。此外，由于当时设想不同应用程序对网络的需求差别很大，从文件传输到实时的语音传输，所以迫切需要一种灵活的网络体系结构。

链路层

所有这些要求导致本参考模型选择了数据包交换网络，它以一个可运行在不同网络之上的无连接网络层为基础。模型中的最低层是**链路层**（link layer），该层描述了链路必须完成什么功能才能满足无连接的互联网络层的需求，比如串行线和经典以太网链路。这不是真正意义上的一个层，而是主机与传输线路之间的一个接口。TCP/IP 模型的早期文档很少提到这点。

互联网层

互联网层（internet layer）是将整个网络体系结构贯穿在一起的关键层。它大致对应于

OSI 的网络层，如图 1-21 所示。该层的任务是允许主机将数据包注入到任何网络，并且让这些数据包独立地到达接收方（接收方可能在不同的网络上）。甚至数据包的到达顺序与它们被发送的顺序不同，在这种情况下，如果需要按序递交数据，那么重新排列这些数据包的任务由高层来负责完成。请注意，虽然在因特网（Internet）中也包含了互联网层，但这里的“互联网”（internet）是指一般意义上的互联网络。

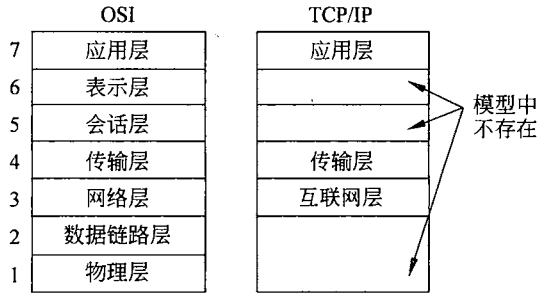


图 1-21 TCP/IP 参考模型

这里我们可以把互联网层比喻为（蜗牛般的）邮政系统。在某个国家，一个人可以将多封国际信件投递到一个邮箱中，只要有一点运气，这些信件大多会被投递到目标国家的正确地址。这些信件或许沿途经过了一个或者多个国际邮件关卡，但是这些对于用户来说是完全透明的。而且，每个国家（即每个网络）有它自己的邮戳、信封大小规格以及投递规则，这些对于用户而言都是不可见的。

互联网层定义了官方的数据包格式和协议，该协议称为**因特网协议**（IP，Internet Protocol），与之相伴的还有一个辅助协议，称为**因特网控制报文协议**（ICMP，Internet Control Message Protocol）。互联网层的任务是将 IP 分组投递到它们该去的地方。很显然，数据包的路由是这里最主要的问题，同时该层还要考虑拥塞控制问题（尽管没有证据表明 IP 能有效地避免拥塞）。

传输层

在 TCP/IP 模型中位于互联网层之上的那一层现在通常称为**传输层**（transport layer）。它的设计目标是允许源主机和目标主机上的对等实体进行对话，犹如 OSI 的传输层一样。这里定义了两个端到端的传输协议。第一个是**传输控制协议**（TCP，Transport Control Protocol），它是一个可靠的、面向连接的协议，允许从一台机器发出的字节流正确无误地交付到互联网上的另一台机器。它把输入的字节流分割成离散的报文，并把每个报文传递给互联网层。在目标机器，接收 TCP 进程把收到的报文重新装配到输出流中。TCP 还负责处理流量控制，以便确保一个快速的发送方不会因发送太多的报文而淹没掉一个处理能力跟不上的慢速接收方。

传输层的第二个协议是**用户数据报协议**（UDP，User Datagram Protocol），它是一个不可靠的、无连接协议，适用于那些不想要 TCP 的有序性或流量控制功能，而宁可自己提供这些功能的应用程序。UDP 被广泛应用于那些一次性的基于客户机-服务器类型的“请求-应答”查询应用，以及那些及时交付比精确交付更加重要的应用，比如传输语音或者视频。IP、TCP 和 UDP 三者之间的关系如图 1-22 所示。自从这个模型被开发以后，许多其他的

网络也都陆续实现了 IP。

应用层

TCP/IP 模型并没有会话层和表示层，因为当时感觉并不需要这两层。相反，应用层简单包含了所需的任何会话和表示功能。来自 OSI 模型的经验已经证明这种观点是正确的：对于大多数应用来说这两层并没有多大用处。

在传输层之上是**应用层**（application layer），它包含了所有的高层协议。最早的高层协议包括虚拟终端协议（TELNET）、文件传输协议（FTP）和电子邮件协议（SMTP）等。经过了这么多年的发展以后，许多其他协议被加入到了应用层。其中我们将要学习的重要协议如图 1-22 所示，包括将主机名字映射到它们网络地址的域名系统（DNS，Domain Name System）、用于获取万维网页面的 HTTP 以及用于传送诸如语音或者电影等实时媒体的 RTP 等。

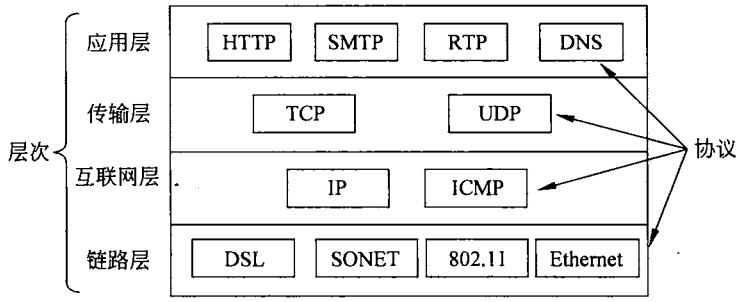


图 1-22 TCP/IP 模型及一些将要学习的协议

1.4.3 本书使用的模型

如前所述，OSI 参考模型的实力在于模型本身（去掉表示层和会话层），它已被证明对于讨论计算机网络特别有益；相反，TCP/IP 参考模型的实力体现在协议，这些协议已被广泛使用多年。由于计算机科学家倾向于使用自己设计的模型，我们将使用如图 1-23 所示的混合模型作为这本书的框架。

图 1-23 本书使用的参考模型

这个模型有 5 层，从物理层往上穿过数据链路层、网络层和传输层到应用层。物理层规定了如何在不同的介质上以电气（或其他模拟）信号传输比特。链路层关注的是如何在两台直接相连的计算机之间发送有限长度的消息，并具有指定级别的可靠性。以太网和 802.11 是链路层协议的例子。

网络层主要处理如何把多条链路结合到网络中，以及如何把网络与网络联结成互联网络，以便使我们可以在两个相隔遥远的计算机之间发送数据包。网络层的任务包括找到传递数据包所走的路径。IP 是我们将要学习的网络层主要协议案例。传输层增强了网络层的传递保证，通常具有更高的可靠性，而且提供了数据交付的抽象，比如满足不同应用需求的可靠字节流。TCP 是传输层协议的一个重要实例。

最后，应用层包含了使用网络的应用程序。许多网络应用程序都有用户界面，比如Web浏览器，但也不是所有的应用程序都有用户界面。然而，我们关心的是应用程序中使用网络的那部分程序。在Web浏览器的情况下就是HTTP协议。应用层也有重要的支撑程序供许多其他应用程序使用，比如DNS。

本书的章节顺序就以此模型为基础安排。通过这种方式，为了便于理解网络体系结构我们保留了OSI模型的价值，但把关注的重点放在实际使用的重要协议上，从TCP/IP协议及相关协议到一些新的协议，例如802.11、SONET和蓝牙。

1.4.4 OSI参考模型与TCP/IP参考模型的比较

OSI和TCP/IP参考模型有很多共同点。两者都以协议栈概念为基础，并且协议栈中的协议彼此相互独立。除此之外，两个模型中各个层的功能也大致相似。例如，在两个模型中，传输层以及传输层以上各层都为希望通信的进程提供了一种端到端的独立于网络的传输服务。这些层组成了传输服务提供者。而且，在这两个模型中，传输层之上的各层都是传输服务的用户，并且是面向应用的。

除了这些基本的相似性以外，两个模型也有许多不同的地方。在这一小节中，我们将注意力集中在两个模型之间的关键区别上。重要的是要注意我们这里比较的是参考模型，而不是对应的协议栈。关于协议本身，我们将在后面讨论。关于比较和对比TCP/IP模型和OSI模型的完整书，请参考（Piscitello和Chanpin, 1993）。

OSI模型的核心是如下3个概念：

（1）服务。

（2）接口。

（3）协议。

或许OSI模型的最大贡献在于明确区分了这3个概念。每一层都为它的上一层执行某些服务。服务定义说明了该层是做什么的，而不是上一层实体如何访问这一层，或这一层是如何工作的。它定义了这一层的语义。

每一层的接口告诉它上面的进程如何访问本层。它规定了有哪些参数，以及结果是什么。但它没有说明本层内部是如何工作。

最后，每一层用到的对等协议是本层自己内部的事情。它可以使用任何协议，只要它能够完成任务就行（也就是说提供所承诺的服务）。它也可以随意地改变协议，而不会影响它上面的各层。

这些思想与现代面向对象的程序设计思想非常吻合。一个对象就如同一个层一样，它有一组方法（或者称为操作）供对象之外的过程调用。这些方法的语义定义了该对象所提供的服务集合。方法的参数和结果构成了对象的接口。对象的内部代码是它的协议，对于外部而言是不可见的，也不需要被外界关心。

最初，TCP/IP模型并没有明确区分服务、接口和协议，尽管人们试图对它进行改进以便使它更像OSI。例如，互联网层提供的真正服务只有发送IP数据包（SEND IP PACKET）和接收IP数据包（RECEIVE IP PACKET）。因此，OSI模型中的协议比TCP/IP模型中的

协议有更好的隐蔽性，当技术发生变化时 OSI 模型中的协议相对更容易被新协议所替换。这种技术改变的透明性就是最初采用分层协议的主要目的之一。

OSI 参考模型在协议发明之前就已经产生了。这种顺序关系意味着 OSI 模型不会偏向于任何一组特定的协议，这个事实使得 OSI 模型更具有通用性。但这种做法也有缺点，那就是设计者在这方面没有太多的经验，因此对于每一层应该设置哪些功能没有特别好的主意。

例如，数据链路层最初只处理点到点网络。当广播式网络出现后，必须在模型中嵌入一个新的子层。而且，当人们使用 OSI 模型和已有协议来构建实际网络时，才发现这些网络并不能很好地满足所需的服务规范，因此不得不在模型中加入一些汇聚子层，以便提供足够的空间来弥补这些差异。最后，标准委员会最初期望每个国家都有一个由政府来运行的网络并采用 OSI 协议，所以当时根本没有考虑网络互连的问题。总而言之，事情并不像预期的那样。

而 TCP/IP 却正好相反：先有协议，TCP/IP 模型只是已有协议的一个描述而已。所以，毫无疑问，协议与模型高度吻合，而且两者结合得非常完美。唯一的问题在于，TCP/IP 模型并不适合任何其他协议栈。因此，要想描述其他非 TCP/IP 网络，该模型并不很有用。

现在我们从两个模型的基本思想转到更为具体的方面上来，它们之间一个很明显的区别是有不同的层数：OSI 模型有 7 层，而 TCP/IP 只有 4 层。它们都有（互联）网络层、传输层和应用层，但是其他层并不相同。

另一个区别在于无连接和面向连接的通信领域有所不同。OSI 模型的网络层同时支持无连接和面向连接的通信，但是传输层只支持面向连接的通信，这是由该层的特点所决定的（因为传输服务对于用户是可见的）。TCP/IP 模型在网络层只支持一种模式（无连接），但是在传输层同时支持两种通信模式，这样可以给用户一个选择的机会。这种选择机会对于简单的“请求-应答”协议特别重要。

1.4.5 OSI 模型和协议的评判

不管是 OSI 模型及其协议，还是 TCP/IP 模型及其协议，都不完美。针对它们都有不少的批评意见。在这一小节以及下一小节，我们来看看对它们的一些评论。首先我们从 OSI 模型开始，然后再看 TCP/IP 模型。

在本书第 2 版出版时（1989 年），这个领域中的大多数专家都觉得 OSI 模型及其协议将统领整个网络世界，所有其他的技术和标准都会出局。但这种情况并没有发生，为什么？回顾总结归纳出一些原因可能会给我们一些启示。这些原因可概括如下：

（1）糟糕的时机。

（2）糟糕的技术。

（3）糟糕的实现。

（4）糟糕的政策。

糟糕的时机

首先我们来看 OSI 模型失败的第一个理由：糟糕的时机。一个标准在什么时候建立对

于该标准的成功与否绝对非常重要。MIT 的 David Clark 有一个关于标准的理论，他称之为两头大象的启示，如图 1-24 所示。

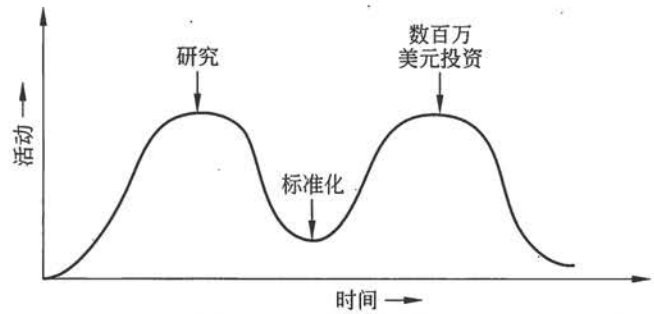


图 1-24 两头大象的启示

该图显示了围绕一个新主题的活动情况。当新主题被首次发现后，会出现大量各种形式的研究活动，比如讨论、论文和会议等。过了一段时间之后，这种活动逐渐趋于平稳，企业发现了该主题，于是数亿美元的投资热潮就开始了。

关键的一点在于，标准的制定工作必须处于两头“大象”的中间。如果标准制定得太早（还没有获得很好的研究成果），该主题仍然处于尚未被很好理解的不成熟阶段，其结果就是一个坏的标准。如果标准制定得太晚，则许多公司可能已经通过各种不同的方式投入了大量的资金，因而标准就会被有效地忽视。如果两头大象之间的间隔太短（因为大家都急于快速启动），那么制定标准的人有可能被夹在中间而举步维艰。

现在来看，标准的 OSI 协议就是这样被夹在了中间。当 OSI 协议出现的时候，与之竞争的 TCP/IP 协议已经被广泛地应用于大学和科研机构。虽然几十亿美元的投资热潮尚未开始，但是，学术市场足够大，使得许多厂商开始谨慎地提供 TCP/IP 产品。当 OSI 出现的时候，这些厂商并不想支持第二个协议栈，除非他们被迫这样做，因此 OSI 没有得到初始的投入。由于每一家公司都在观望等待其他的公司先行一步，结果是没有哪家公司先走一步支持 OSI，所以 OSI 从来没有被真正的实现。悲哉！

糟糕的技术

OSI 一直没有流行起来的第二个原因在于无论是模型还是协议都存在缺陷。之所以选择 7 层的原因很大程度上是出于政策上的考虑，而非技术因素所决定。其中的两层（会话层和表示层）几乎是空的，而另外两层（数据链路层和网络层）又包含了太多内容。

OSI 模型以及相应的服务定义和协议都极其复杂。如果将标准打印出来，堆叠起来的文档可以高达 1 米的高度。它们难以实现，而且操作起来也很低效。在这样的上下文中，令人想起了 Paul Mockapetris 出的一个谜语，并且被（Rose，1993）引用过：

问：当你碰到一个握有国际标准的霸权主义者时会得到什么？

答：他会给你一些你不能理解的东西。

除了难以理解外，OSI 的另一个问题是有些功能重复地出现在每一层，比如编址、流量控制和差错误控制等。（Saltzer 等，1984）已经指出要想真正做到高效率，差错控制必须在最高层上完成，所以在较低层不断地重复这一功能往往是不必要的，也是低效的。

糟糕的实现

由于 OSI 模型和协议过于复杂，最初的那些实现不仅庞大，而且很笨拙，效率也很慢，这一点也不足为奇。任何试图使用 OSI 的人无不被搞得焦头烂额。没过多久，人们就把 OSI 与“糟糕的质量”联系在一起了。尽管随着时间的推移，相应的产品有了改进，但是这种印象已经深深烙印在人们心中。

相反，TCP/IP 的早期实现之一是作为 Berkeley UNIX 的一部分，运行非常好（更不用说它是免费的）。很快，人们就开始使用它，进而形成了一个庞大的用户群，这进一步促进了它的提高和改进，然后又导致了更大的用户群。这是螺旋式上升而不是下降。

糟糕的政策

由于 TCP/IP 最初的实现，很多人（特别在学术界）都把 TCP/IP 看作是 UNIX 的一部分，而 UNIX 在 20 世纪 80 年代的学术圈中盛极一时，备受宠爱。

相反，OSI 则被认为是欧洲电信部门、欧共体以及后来的美国政府的产物。尽管这种观点部分是正确的，但是政府官僚们试图把技术上不足的标准强加给那些实际开发计算机网络的可怜的研究人员和程序员，而政府部门的强制性对 OSI 无济于事。某些人把这种部署类比于 IBM 在 20 世纪 60 年代宣布 PL/I 将成为未来语言那一事件，当时，美国国防部在 IBM 声明之后不久就做了强制更正，宣称未来语言是 Ada。

1.4.6 TCP/IP 参考模型的评判

TCP/IP 模型和协议也有自己的问题。第一，该模型并没有明确区分服务、接口和协议的概念。好的软件工程师在实际工作中都要求区分哪些是规范，哪些是实现，这一点 OSI 模型非常谨慎地做到了，但是 TCP/IP 模型并没有这样做。因此，在使用新技术来设计新网络时，TCP/IP 模型并不是一个很好的参照物。

第二，TCP/IP 模型一点也不通用，它并不适合于用来描述 TCP/IP 之外的任何其他协议栈。例如，试图使用 TCP/IP 模型来描述蓝牙（Bluetooth）是完全不可能的。

第三，在分层协议的上下文中，链路层并不是通常意义上的一层。它是一个接口（位于网络层和数据链路层之间），而接口和层的区别非常重要，我们不能对此粗心大意。

第四，TCP/IP 模型并没有区分物理层和数据链路层。这是两个完全不同的层。物理层必须要考虑铜线、光纤和无线通信的传输特征；而数据链路层的任务则是确定帧的开始和结束，并且按照所需的可靠程度把帧从一边发送到另一边。一个正确的模型应该包括这两个独立的层，TCP/IP 模型没有这样做。

最后，尽管 IP 和 TCP 协议进行了仔细设计，并且很好地实现了，但是还有很多其他协议是自主形成的，通常这些协议由一群不知疲倦的研究生开发出来；然后，开发出来的协议实现被免费发布，因而这些协议得到了广泛的应用，在用户中的地位根深蒂固，导致其他协议难以取而代之。但是随着时间的推移，现在这些协议反而成了一种障碍。例如，虚拟终端协议 TELNET 最初是为每秒 10 个字符的机械电传打字终端设计的，它无法识别图形用户界面和鼠标。然而，30 年之后它仍然在广泛使用。

1.5 网络实例

计算机联网这一主题覆盖了许多不同种类的网络，规模有大有小，有知名的也有不知名的。不同的网络具有不同的目标、尺度和技术。在接下去的几小节，我们将介绍一些网络实例，以便读者对于计算机网络领域中的各种网络有一个认识。

我们首先从 Internet 开始，这或许是世界上最著名的网络，我们将讨论它的历史、发展历程以及相应的技术。其次我们将考虑移动电话网络，从技术角度来看，它与 Internet 有很大的不同，两者形成了鲜明的对比。接下来我们将介绍 IEEE 802.11，即最主要的无线局域网络。最后，我们将考察 RFID 和传感器网络，这些是网络扩展技术，利用它们可将计算机网络的触角延伸到物理世界和日常物品。

1.5.1 因特网

因特网（Internet）并不是单个网络，而是大量不同网络的集合，这些网络使用特定的公共协议，并提供特定的公共服务。Internet 是一个不同寻常的系统，它不是由任何人规划出来的，也不受任何人控制。为了更好地理解 Internet，让我们从它的起源开始，看它是如何发展以及为什么会发展起来的。如果你要了解 Internet 的辉煌历史，强烈建议你看一看 John Naugnton 的书（2000）。这是一本难得的好书，不仅读起来有趣，而且为严谨的网络史学家提供了 20 页的参考引用书目。本节中的有些材料就是取自这本书。

当然，关于 Internet 及其协议有无数的技术书籍。例如，有关更多的信息，你可以参考（Maufer, 1999）。

ARPANET

故事始于 20 世纪 50 年代后期的冷战高峰期，美国国防部希望建立一个“命令-控制”网络，即使在核战争爆发的情况下它也能够生存下来。在当时，所有的军事通信都使用公共电话网络，而电话网络则被认为非常脆弱。从图 1-25（a）我们可以看出秉持这种观点的理由。图中黑色的点代表电话交换局，每个电话交换局都连接着几千部电话；然后，这些交换局再连接到更高层次的交换局（长途局），从而形成全国性的层次结构，整个结构只有少量的冗余。这种系统的脆弱性在于一旦几个关键的长途局遭到破坏，则整个系统有可能被分割成多个孤岛。

1960 年左右，美国国防部给了兰德公司一份合同，授权它寻找一个解决方案。兰德公司的一位雇员 Paul Baran 提出了一个高度分布式的和容错设计方案，如图 1-25（b）所示。由于任何两个交换局之间的路径长度都超过了模拟信号能正确传输（不失真）的最长距离，所以 Baran 建议采用数字的数据包交换技术。Baran 给美国国防部写了几份报告来详细阐述他的思想（Baran，1964）。五角大楼的官员喜欢这个概念，并且请 AT&T 公司，以及后来的美国国家电话局来建立一个原型系统。但 AT&T 公司驳回了 Baran 的想法。这个世界上最大最富有的公司是不会允许一个自以为是的年轻人来告诉自己如何建立一个电话系统。他们说 Baran 的网络是不可能建立起来的，于是 Baran 的想法被扼杀了。

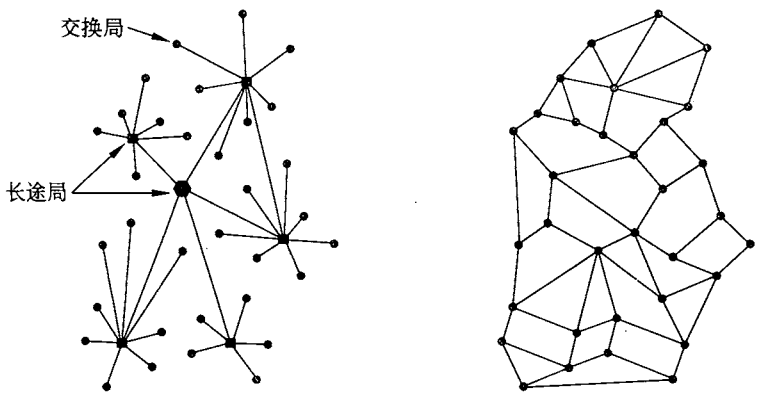


图 1-25
(a) 电话系统的结构; (b) Baran 提议的分布式交换系统

几年以后，美国国防部还是没有得到一个更好的“命令-控制”系统。为了理解接下来所发生的事情，我们必须回到 1957 年的 10 月份，前苏联发射了第一颗人造卫星（Sputnik），美国跟前苏联在太空领域展开了激烈的竞争。当时的美国总统艾森豪威尔试图找出是谁在玩忽职守，他很惊讶地发现，陆军、海军和空军都在为五角大楼的研究经费预算而争吵不停。他的第一反应是建立一个专门的国防研究组织，即高级研究计划局（ARPA，Advanced Research Projects Agency）。ARPA 没有科学家，也没有实验室，事实上，除了一间办公室和少量的预算（按照五角大楼的标准）以外什么也没有。ARPA 通过发放项目资助和签约合同的形式，让那些想法比较有前景的大学或者公司来为它工作。

在刚开始的几年中，ARPA 试图确定自己的使命到底是什么。在 1967 年，ARPA 的项目经理 Larry Roberts 将注意力从如何提供远程访问计算机的技术转到了网络技术上。他联系了不同的专家，以便确定要做些什么。其中一个专家，Wesley Clark 建议建立一个数据包交换子网，将每台主机连接到它自己的路由器上。

在经过了初始的怀疑后，Roberts 同意了这种想法，并且于 1967 年下半年在田纳西州 Gatlinburg 举行的 ACM SIGOPS 会议上宣读了这份有点含糊不清的文章，这个会议是关于操作系统原理的研讨会。令 Roberts 感到惊讶的是此次会议上的另外一篇文章描述了一个类似的系统，该系统不仅有设计，而且英国国家物理实验室（NPL，National Physical Laboratory）在 Donald Davies 的指导下已经实现了该系统。NPL 系统不是一个国家级的系统（它只是将 NPL 园区中的几台计算机连接起来），但它证明了数据包交换的思想是可以工作的。而且，它引用了当时已被弃之不用的 Baran 早期研究工作。Roberts 从 Gatlinburg 回来之后决定建立一个网络，这就是后来称为 ARPANET 的网络。

ARPANET 子网由一些小型机组成，这些小型机称为接口报文处理器（IMP，Interface Message Processors），它们通过 56 kbps 的传输线路连接。为了保证高度可靠性，每个 IMP 至少与两个其他 IMP 连接。子网采用数据报技术，所以，如果有一些线路或者 IMP 被毁坏，报文能被自动地重新路由到其他替代的路径上。

网络中的每个节点都包含一个 IMP 和一台主机，这两台设备位于同一个房间中，通过一条很短的线连接起来。主机给 IMP 发送的报文最长可达 8063 位；然后 IMP 把这些报文

分成最多1008位的数据包，再向目标节点独立地转发这些数据包。每一个数据包必须在完整地到达一个节点之后才可以被再次转发，所以，该子网是第一个电子的、存储-转发的数据包交换网络。

然后，ARPA以招标的形式来建立该子网。有12家公司参与竞标。在评估了所有的项目申请书之后，ARPA选择了BBN，这是麻省剑桥的一家咨询公司，然后ARPA与BBN签署了建立子网并编写子网所需软件的合同。BBN选择了经过特殊改进的Honeywell DDP-316小型机作为IMP。该机器有12 KB 16位字的核心内存，但没有磁盘，因为可移动部件被认为是不可靠的。通过租用电话公司56 kbps的线路将这些IMP相互连接起来。虽然，56 kbps现在已经成了那些支付不起DSL或者线缆费用的青少年们才使用的线路，但在当时这也价格不菲、需要很多钱才能买得到。

软件分成两部分：子网和主机。子网软件包括主机-IMP连接的IMP端、IMP-IMP协议以及一个源IMP-目标IMP的协议，后者是专门为提高传输可靠性而设计的。原始的ARPANET设计如图1-26所示。

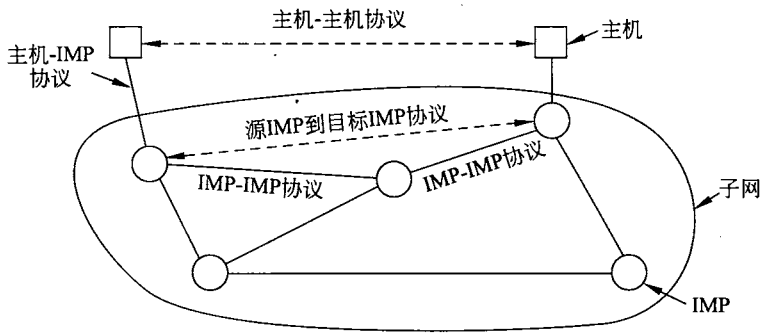


图1-26 最初的ARPANET设计

在子网外还需要其他软件，也就是说，主机-IMP连接的主机端、主机-主机协议以及应用软件。很快地，BBN就意识到当它能够在主机-IMP的线路上接收到报文，并且把报文放到接收方的主机-IMP线路上，它的任务就完成了。

Roberts有一个问题，他想主机也需要软件。为了解决这个问题，1969年夏季，他在犹他州的Snowbird召集了一次网络研究人员的会议，其中大多数是研究生。研究生们希望有一个网络专家向他们解释网络的宏伟设计方案以及所需要的软件，然后给每个人分配编写软件的一部分任务。他们惊讶于发现那里既没有网络专家，也没有宏观设计，他们必须自己想办法来找到该做的事情。

不管怎么样，在1969年12月一个实验性的网络上线运行了，它包含4个节点：UCLA、UCSB、SRI和犹他大学。之所以选择这4个点，是因为这些地方承担了ARPA的许多合同，而且都有许多不同类型且完全不兼容的主机（使得这一项工作更加有趣）。在这两个月之前第一个主机-主机报文就从UCLA节点发往了SRI节点，其中UCLA节点的工作团队由Len Kleinrock领导（他是数据包交换理论的先行者）。随着更多的IMP被交付和安装，网络增长得很快；很快它就扩展到了整个美国。图1-27显示了ARPANET在最初3年中是如何快速增长的。

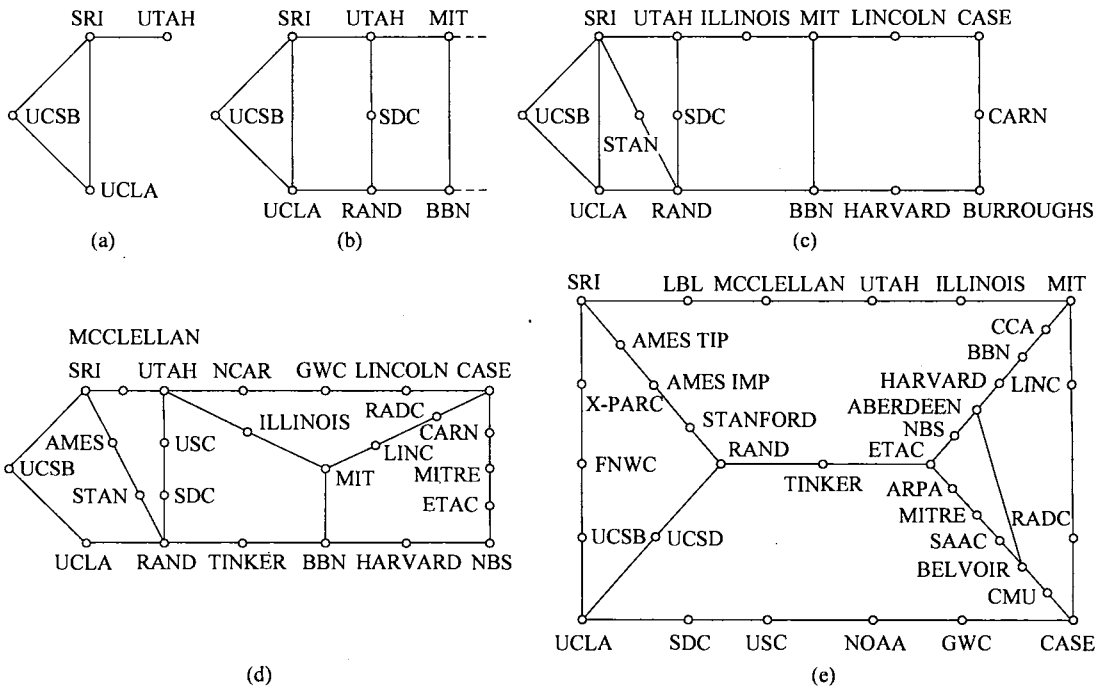


图 1-27 ARPANET 的增长

(a) 1969 年 12 月; (b) 1970 年 7 月; (c) 1971 年 3 月; (d) 1972 年 4 月; (e) 1972 年 9 月

除了帮助羽翼未丰的 ARPANET 成长外，ARPA 还资助了有关卫星网络和移动数据包无线网络应用方面的研究工作。在一次著名的演示中，一辆正在加州行驶的卡车通过数据包无线网络给 SRI 发送报文，然后 SRI 将该报文通过 ARPANET 转发到了东海岸，再通过卫星网络发送到伦敦的大学学院。这样，卡车中的研究人员就可以一边在加州的公路上行驶，一边使用伦敦的计算机了。

这次实验也验证了现有 ARPANET 协议并不适合跨越多个网络运行。这个观察结果导致了更多有关协议的研究工作，并最终发明了 TCP/IP 模型和协议（Cerf 和 Kahn，1974）。TCP/IP 是专门设计用来处理互联网络的通信，随着越来越多的网络挂接到 ARPANET 上，网络互连变得越来越重要了。

为了鼓励采用这些新协议，ARPA 发放了几个在不同计算机平台上实现 TCP/IP 的合同，包括 IBM、DEC 和 HP 系统，以及 Berkeley UNIX。加州大学 Berkeley 分校的研究人员用一种新的编程接口重写了 TCP/IP，这个编程接口就是随着 Berkeley UNIX 4.2BSD 一起发布的著名**套接字**（socket）。他们还编写了许多应用程序、工具以及管理程序，以便展示通过套接字使用网络有多么的容易。

真是机缘巧合，许多大学刚刚得到了第二台或者第三台 VAX 计算机和连接它们的局域网，但是他们却没有联网软件。当 4.2BSD 横空出世，连同 TCP/IP、套接字编程接口以及许多网络工具等整个软件包被立即采纳。而且，通过 TCP/IP 把 LAN 连接到 ARPANET 非常容易，许多 LAN 也的确这样做了。

在 20 世纪 80 年代，额外的一些网络，特别是局域网都连接到了 ARPANET 上。随着网络规模的增长，寻找主机所需要的开销越来越昂贵，所以创建了域名系统（DNS，Domain

Name System）。DNS 将机器组织成域，并且将主机名字映射成 IP 地址。从那时开始，DNS 就成为一个通用的分布式数据库系统，它保存了各种各样与名字有关的信息。我们将在第 7 章详细地介绍 DNS。

NSFNET

到了 20 世纪 70 年代后期，美国国家科学基金会（NSF，U.S. National Science Foundation）看到 ARPANET 对大学研究工作产生的巨大影响。有了 ARPANET，不同国家的科学家们可以共享数据并开展研究项目上的协同工作。然而，对于任何一个大学，如果它想要使用 ARPANET，必须与美国国防部有一个研究合同，而这是许多大学所不具备的。NSF 最初的想法是在 1981 年建设一个计算机科学网络（CSNET，Computer Science Network）。该网络将全美国大学的计算机系和工业研究实验室通过拨号和租用线路连到 ARPANET。到 20 世纪 80 年代后期，NSF 更加深入一步，决定设计一个 ARPANET 的继任网络，对所有大学的研究组都开放。

为了使这件事情得以实质性地展开，NSF 决定建立一个骨干网络，将它的 6 个超级计算机中心连接起来，这 6 个计算机中心分别位于 San Diego、Boulder、Champaign、Pittsburgh、Ithaca 和 Princeton。每台超级计算机都配备一台称为 fuzzball 的 LSI-11 微型计算机。这些微型计算机通过 56 kbps 租用线路连接起来，形成了子网结构，并采用了与 ARPANET 相同的硬件技术。然而，软件技术有点不同， fuzzball 微型计算机从一开始就使用 TCP/IP，因此成为第一个 TCP/IP 广域网。

NSF 还资助了一些（最终大约 20 个）连到骨干网上的区域性网络，这些区域网络允许数以千计的大学、研究实验室、图书馆和博物馆的用户访问任何一台超级计算机，并且用户相互之间可以进行通信。整个网络，包括骨干网和区域网，合起来统称为 NSFNET。它通过 Carnegie-Mellon 机房内一条连接 IMP 和 fuzzball 的链路连接到 ARPANET。最初的 NSFNET 骨干网映射到美国版图上的样子如图 1-28 所示。

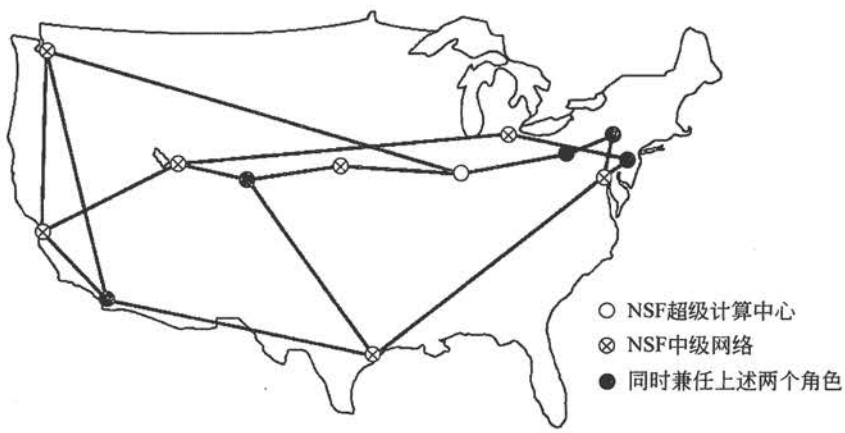


图 1-28　1988 年的 NSFNET 骨干网

NSFNET 很快获得了成功，从一开始就超负荷地运转。NSF 立即着手规划它的下一个网络，并且给总部设在 Michigan 的 MERIT 财团一份合同负责运行这网络。他们从 MCI（已经与 WorldCom 合并）租用了 448 kbps 的光纤信道来构建骨干网的第二个版本，把 IBM

PC-RT 用作路由器。很快地，网络又超负荷了。到 1990 年，第二个骨干网升级到了 1.5 Mbps。

随着网络的不断增长，NSF 意识到政府不可能永远不停地资助网络发展。同时，商业组织也想要加入网络，但是受到 NSF 的章程限制，即商业组织不能使用由 NSF 支付的网络。因此，NSF 鼓励 MERIT、MCI 和 IBM 组成一个非营利性的公司作为迈向商业化道路的第一步，该公司称为高级网络与服务（ANS，Advanced Networks and Services）。1990 年，ANS 正式接管了 NSFNET，并且将 1.5 Mbps 链路升级到 45 Mbps，构成了 ANSNET。该网络运行了 5 年，然后被出售给美国在线（America Online）。此时，许多公司已经都在提供商业性的 IP 服务，很明显现在政府应该退出网络服务的商业圈了。

为了使传输更加容易，并且确保每一个区域网络都可以与每一个其他区域网络进行通信，NSF 与 4 个网络运行商签订合同，让它们都来建立**网络接入点**（NAP，Network Access Point）。这 4 个网络运行商分别是 PacBell（旧金山）、Ameritech（芝加哥）、MFS（华盛顿特区）和 Sprint（纽约市，为了建设 NAP 宾夕法尼亚和新泽西也算在了纽约市内）。任何一个网络运行商，如果要想给 NSF 区域网提供骨干网服务，则它必须与所有的 NAP 连接。

这种安排意味着从任何一个区域网发出的数据包可以选择骨干网承运商，从它所在的 NAP 到达目标 NAP。因此，骨干网承运商不得不为了区域网的业务而在服务和价格上展开激烈竞争，当然，这正是 NSF 的想法。最终的结果是，原来单个默认的骨干网被一个由商业驱动的竞争基础设施所取代。许多人喜欢批评美国联邦政府不会创新，但是在网络领域，正是美国 DoD 和 NSF 建立了作为 Internet 基础的网络基础设施，然后它们又把该设施交给工业界来运行。

在 20 世纪 90 年代，许多其他国家和区域也纷纷建起了国家研究网络，通常采用了 ARPANET 和 NSFNET 的模式。这其中包括欧洲的 EuropaNET 和 EBONE，刚开始的时候它们使用 2 Mbps 线路，后来升级到 34 Mbps 线路。最终欧洲的网络基础设施也交给了工业界来运行。

从此以后 Internet 发生了巨大的改变。它的规模随着 20 世纪 90 年代初万维网的出现而呈现爆炸式的增长。Internet 系统协会（Internet Systems Consortium）的最新数据显示可见的 Internet 主机数已经超过 600 万。这种猜测仅仅是一个保守估计，但是当 1994 年第一届 WWW 会议在欧洲核子研究中心举行时这个数目已远远超过了几百万。

我们使用 Internet 的方式也发生了根本变化。起初，学术用的电子邮件、新闻组、远程登录和文件传输等应用占据主导地位。后来逐步发展到普通人也使用电子邮件，然后是网站和对等内容分发，比如现在已经关闭了的 Napster。现在实时媒体分发、社会网络（例如 Facebook）和微博（例如 Twitter）正腾空而起。这些使用方式的变换给 Internet 带来了更丰富的媒体种类，因而产生了更多的流量。事实上，占 Internet 主导地位的流量似乎具有某种规律性的变化，例如，新的和更好的在网络上处理和传播音乐或电影的方式会很快变得非常流行。

Internet 的体系结构

随着网络规模的爆炸性增长，Internet 的体系结构也发生了很大变化。在本节，我们将尝试概述今天的 Internet 看起来是什么样子的。业务领域的连续动荡使得完整的 Internet 映像非常复杂，电话公司（电信公司）、有线电视公司和互联网服务供应商往往很难分辨谁正

做什么业务。造成这些动荡的驱动因素之一是电信融合，即一个网络有了从前不同的应用。例如，在“三网融合”（triple play）下，以替你省钱的名义，一个公司可以在相同的网络连接上推销电话业务、有线电视和 Internet 服务。因此，相比现实情况这里给出的描述在一定程度上做了必要的简化。今天正确的东西在明天未必依然正确。

Internet 的全局如图 1-29 所示。让我们来逐步检查每个部分，首先从家用计算机开始（在图的边缘）。为了加入 Internet，计算机必须连到 Internet 服务提供商或简称为 ISP，用户从该 ISP 购买了 Internet 接入或连接性服务。这样计算机就可以和所有 Internet 上的其他可接入主机交换数据包。用户可能会发送数据包在网上冲浪或任何数千余种其他使用，具体什么应用无关紧要。Internet 接入有多种类型，通常由它们能提供多大的带宽和需要多大的成本来区分，但最重要的属性是网络连通性。

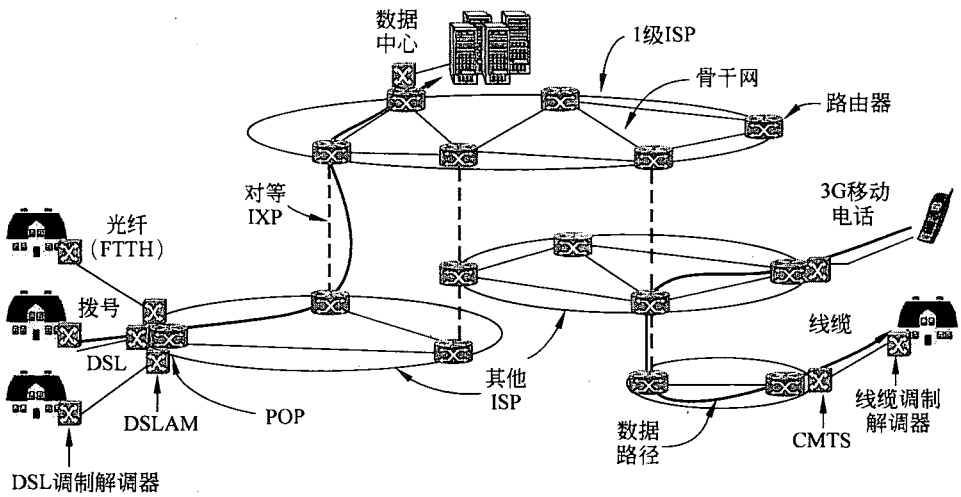


图 1-29 Internet 体系结构概貌

连接到 ISP 的一种常见方法是使用家里的电话线。在这种情况下，电话公司就是你的 ISP。DSL 是**数字用户线**（Digital Subscriber Line）的简称，它重复使用连到各家的电话线进行数字数据的传输。计算机与一个称为 DSL 调制解调器的设备连接。该设备将数字数据包转换成可以畅通无阻地通过电话线传递的模拟信号。在另一端，一台称为**数字用户线接入复用器**（DSLAM，Digital Subscriber Line Access Multiplexer）的设备负责模拟信号和数字数据包之间的转换。

图 1-29 显示了几种连接 ISP 的其他流行方式。DSL 以一种高带宽的方式使用本地电话线，它可以通过传统的电话呼叫来发送比特而不是发送语音对话。这就是所谓的**拨号**（dial-up）上网，通信两端必须要用调制解调器才能工作，调制解调器的种类有很多种。**调制解调器**（modem）这个字取自调制（modulator）与解调（demodulator），指任何一种在数字比特和模拟信号之间作转换的设备。

另一种方法是通过有线电视系统发送信号。像 DSL 一样，这也是一种重用已有基础设施的方式。在这种情况下，利用有线电视没有使用的信道来发送信号。家庭端的设备称为线缆调制解调器，而在有线电视电缆（简称线缆）头端的设备称为**线缆调制解调终端系统**（CMTS，Cable Modem Termination System）。

DSL 和线缆提供的 Internet 接入速率从小于兆比特/秒到多个兆比特/秒不等，确切的速率取于系统。这些数据速率比拨号率高得多，因为语音通话使用窄的带宽，因此拨号速率被限制为 56 kbps。以高于拨号速率接入 Internet 就被称为宽带（broadband）。宽带是指更快网络所用的更宽的带宽，而不是指任何特定的速度。

到目前为止提到的接入方式受到“最后一英里”（last mile）带宽或最后一段传输的限制。通过运行铺设到住宅的光纤，可提供速率在 10～100 Mbps 之间的更快 Internet 接入。这种设计称为光纤到户（FTTH，Fiber to the Home）。对商业领域的企业来说，租用一条从办公室到就近 ISP 的高速传输线路是有意义的。例如，在北美，T3 线路的运行速度大约为 45 Mbps。

无线也可用于 Internet 接入。一个例子是我们将要简短探讨的 3G 移动电话网络。它们可以为覆盖范围内的移动电话和固定用户提供 1 Mbps 或更高速率的数据传输服务。

现在，我们可以在家庭和 ISP 之间移动数据包了。我们把客户数据包进入 ISP 网络使用其服务的位置称为 **ISP 存点**（POP，Point of Presence）。下一步，我们将解释数据包如何在不同的 ISP 存点之间移动。从这个位置开始，系统就完全是数字化的，而且采用的是数据包交换技术。

ISP 网络的规模可能是区域性、国家级或国际范围。我们已经看到它们的体系结构由互连路由器的长距离传输线路组成。路由器位于提供 ISP 服务所在的不同城市的 POP，这种设备称为 ISP **骨干**（backbone）。如果数据包的目的地是由 ISP 直接服务的主机，那么该数据包在骨干网上路由，而且直接传递给目的主机。否则，它必须被移交给另一个 ISP。

ISP 将它们的网络连接到 **Internet 交换点**（IXP，Internet eXchange Point）以便交换流量。相互连接的 ISP 被认为彼此对等（peer）。全世界各地城市有很多 IXP，它们在图 1-29 中被绘制成垂直的虚线，因为 ISP 网络在地理位置上是重叠的。基本上，一个 IXP 是一间放满了路由器的房间，每个 ISP 至少有一台路由器。房间里的局域网连接了所有的路由器，因此数据包可以从任何一个 ISP 骨干网被转发到任何一个其他 ISP 骨干网。IXP 可能超大型，而且拥有独立的设施。最大的一个 IXP 是阿姆斯特丹的 Internet 交换中心，那里连接着数百个 ISP，通过该中心交换着数百个吉比特/秒的流量。

IXP 是否对等取决于 ISP 之间的业务关系，它们之间可以有很多可能的关系。例如，一个小型 ISP 可能会给较大 ISP 一定的费用以便获得其提供的 Internet 连通性服务，从而能够到达遥远的主机，这种情况非常像 ISP 的客户向该 ISP 购买服务的情形。在这种情况下，小型 ISP 向大型 ISP 支付的是中转（transit）费。另外，两个大型 ISP 可能决定互相交换彼此的流量，这样每个 ISP 给其他 ISP 传送一些流量而无须支付中转费。Internet 存在着许多悖论，其中之一就是公开竞争客户的 ISP 往往私下合作成为对等关系（Metz，2001）。

数据包通过 Internet 的路径依赖于对等 ISP 的选择。如果传递一个数据包的 ISP 与目标 ISP 是对等的关系，则它可以直接将数据包传递给对等 ISP。否则，它可能将数据包路由到离付费中转提供商最近的地方，以便中转提供商传递数据包。图 1-29 给出了两条跨越 ISP 的路径。通常，一个数据包所用的路径并不是通过 Internet 的最短路径。

在食物链的顶端是极少数公司，如 AT&T 和 Sprint，这些公司运行着大型国际骨干网络，在那里数以千计的路由器通过高带宽光纤链路相互连接在一起。这些 ISP 不需要支付中转费。它们通常称为 1 级（tier 1）ISP，正是它们形成了 Internet 的骨干网，因为所有其

他ISP都必须与它们连接才能够到达整个Internet。

提供大量内容的公司（比如谷歌和雅虎）都把它们的计算机放在**数据中心**（data center），那里与Internet其余部分有很好的连接。这些数据中心是专为计算机而不是人类设计的，充满着一排排机架，称为**服务器农场**（server farm）。**代管**（colocation）或**托管**（hosting）数据中心让客户把诸如服务器之类的设备放在ISP的POP，以便在服务器与ISP骨干之间形成短程的快速连接。Internet托管行业已日益虚拟化，因此现在租用一台运行在服务器农场的虚拟机器变得很常见，无须真正安装一台物理计算机。这些数据中心是如此之大（几十或几百上千台机器）以至于电费成为其运营的主要成本，所以数据中心有时建立在电力很便宜的地方。

现在我们要结束Internet的快速浏览历程了。在随后的章节中我们将对网络的每个组成部分详细讨论其设计、算法和协议。这里值得一提的是正在发生改变的Internet意味着什么。曾经这样判断一台机器是否连在互联网，看它是否满足下列条件：（1）运行TCP/IP协议栈；（2）有一个IP地址；（3）能够给Internet上的所有其他机器发送IP数据包。然而，ISP往往重复使用IP地址，这取决于当前哪些计算机在上网，家庭网络中的多台计算机通常共享同一个IP地址。这种做法破坏了上述第二个条件。诸如防火墙这样的安全措施也可能部分阻止了计算机接收数据包，这又破坏了上述第三个条件。尽管存在这些困难，把这些机器视为连在Internet上还是有意义的，尽管它们都连接到各自的ISP。

此外还值得顺便提一提的是某些公司已经把它们所有的现行内部网络互连在一起，采用的技术通常与Internet相同。这些内联网（intranet）一般只能在公司内部场所或通过公司笔记本访问，但在其他方面的工作方式与Internet相同。

1.5.2 第三代移动电话网络

人们喜欢在电话里交谈，喜欢的程度甚至超过在网上冲浪，这种需求促使移动电话网络成为了世界上最成功的网络。它在全球有超过40亿的用户。为了客观展望，这个数字大约占世界人口的60%，比Internet主机数和固定电话线的总和还要大（ITU，2009）。

在过去40年间伴随着巨大的增长，移动电话网络的体系结构发生了很大变化。第一代移动电话系统以连续变化的（模拟）信号而非（数字）序列来传输语音通话。**高级移动电话系统**（AMPS，Advanced Mobile Phone System）是一种得到广泛使用的第一代系统，于1982年在美国部署。第二代移动电话系统从模拟传输切换到以数字形式传输语音通话，不仅增加了容量，安全性得到提高，而且还提供了短信服务。1991年开始部署的**全球移动通信系统**（GSM，Global System for Mobile communications）已成为世界上应用最广泛的移动电话系统，它属于2G系统。

第三代或3G系统最初在2001年得到部署，它能同时提供数字语音和宽带数字数据服务。关于3G有很多行业术语和许多不同的标准可供选择。3G由ITU（一个国际标准化组织，我们将在下一节讨论）松散定义成：为行驶中的车辆提供384 kbps的传输速率，为静止或步行的用户提供至少2 Mbps的传输速度。**通用移动通信系统**（UMTS，Universal Mobile Telecommunications System）也称为**宽带码分多址**（WCDMA，Wideband Code Division Multiple Access），是主要的3G系统，目前正在全球范围内迅速部署。它可以提供高达

14 Mbps 的下行链路和近 6 Mbps 的上行链路。未来的 3G 版本将使用多天线和多无线电，为用户提供更大的速度。

3G 系统和之前的 2G 和 1G 系统一样，稀缺的资源依然是无线电频谱。政府给移动电话网络运营商发放使用部分频谱的许可，一般采用的方式是频谱拍卖，网络运营商投标竞拍。拥有一块得到授权的频谱更易于系统设计和运营，因为不允许其他运营商在该频谱上传输，但这种授权往往耗资巨大。例如，2000 年在英国，5 个 3G 牌照拍卖的总额约 400 亿美元。

正因为频谱稀缺，从而导致了蜂窝网络的设计如图 1-30 所示，现在的移动电话网络就采用了这种架构。为了管理用户与用户之间的无线电干扰，系统的覆盖区域被分成一个个蜂窝。在一个蜂窝内，为用户分配互相不干扰的信道，而且分配的信道对相邻蜂窝也不能干扰太大。这样的频率分配方法使得相邻蜂窝中的频谱得以很好的重复使用，即频率重用，从而增加了整个网络的容量。在 1G 系统中，每个语音通话在特定的频段上进行，所用的频率都要经过仔细选择，使得它们不与邻近蜂窝使用的频率发生冲突。这种方式下，一个给定的频率只能一次被几个蜂窝重用。现代的 3G 系统允许每个蜂窝使用全部的频率，但以一种允许相邻蜂窝之间存在可接受干扰程度的方式来分配频率。蜂窝的设计有许多变异方式，其中包括在蜂窝发射塔上使用定向或扇面天线来进一步减少蜂窝之间的频率干扰，但其基本设计思路是相同的。

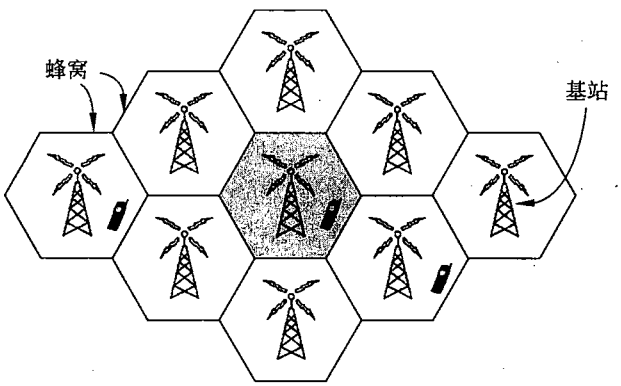


图 1-30 移动电话网络的蜂窝设计

移动电话网络的体系结构与 Internet 体系结构完全不同。它由几部分组成，图 1-31 显示了一个简化版的 UMTS 体系结构。首先，必须有一个**空中接口**（air interface）。这个术语当作一个无线电通信协议的名字很奇特，因为无线通信协议本来就是通过移动设备（如手机）和蜂窝基站之间的空中通信的。过去几十年间空中接口的进步已经大大提高了无线数据传输速率。UMTS 空中接口基于**码分多址**（CDMA，Code Division Multiple Access），我们将在第 2 章研究这种复用技术。

蜂窝基站和它的控制器一起构成了**无线接入网络**（radio access network）。这部分是移动电话网络的无线一则。控制器节点或**无线网络控制器**（RNC，Radio Network Controller）控制如何使用无线电频谱。基站实现空中接口，在图中称为节点 B，一个临时标签。

移动电话网络的其余部分负责运载无线接入网络的流量。这就是所谓的**核心网络**（core network）。UMTS 核心网络是从以前 2G GSM 系统使用的核心网络演化而来。然而，令人

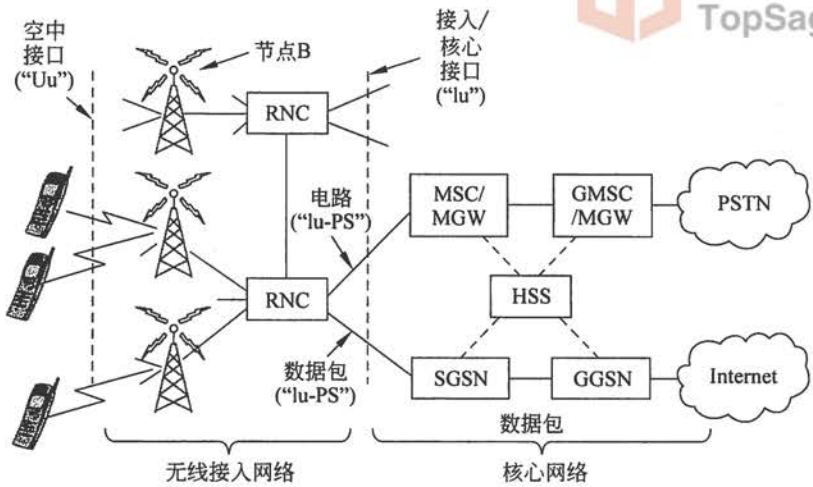


图 1-31 UMTS 3G 移动电话网络体系结构

惊讶的是发生在 UMTS 核心网络的一些情况。

由于在网络开始建设时，存在着支持数据包网络（即无连接的子网）和支持电路网络（即面向连接的子网）的两大阵营，并且它们之间的战争已经持续了一段时间。数据包的主要支持者来自 Internet 社团。在无连接的设计中，每个数据包的路由独立于所有其他的包。因此，如果某些路由器在会话期间出现故障，不会有什么损害，只要系统可以动态地重新配置，后续的数据包就可以发现抵达目的地的其他路径，即使这些路径与以前数据包使用的路径不同。

电路阵营来自电话公司。在电话系统中，主叫方必须拨打被叫方的电话号码，等待电话接通然后才能通话或发送数据。连接设置建立了一条通过电话系统的路由，系统要一直维护该路由直到通话终止。所有的字或包遵循同样的路由。如果路径上的一条线路或交换机出现故障，则通话立即被中止，因而它在容错性方面不如无连接设计。

电路的优点是它们可以更容易支持服务质量。通过提前建立一个连接，子网可以预留，诸如链路带宽、交换机缓冲空间以及 CPU 等资源。如果用户试图建立一个呼叫时系统没有足够的资源可供使用，则该次呼叫将被拒绝，主叫方将听到一个繁忙信号。通过这种方式，一旦连接建立，该连接将得到良好的服务。

在无连接网络中，如果有太多的数据包在同一时刻到达同一个路由器，那么该路由器将被阻塞，并可能丢失数据包。发送方最终将注意到丢包情况，然后重新发送；但相应的服务质量将忽好忽坏，不适合音频或视频传输，除非网络负载很轻。无须多说，提供足够的音频质量是电话公司最关心的事情，因此它们更偏爱有连接的工作方式。

令人惊喜的是在图 1-31 的核心网络中同时包含了数据包交换和电路交换设备。这显示了移动电话网络正在发生转型，移动电话公司能够实现一个或有时同时实现两种服务。旧的移动电话网络使用电路交换核心，以传统的电话网络风格来传递语音通话。这种传统概念体现在 UMTS 网络中的**移动交换中心**（MSC，Mobile Switching Center）、**网关移动交换中心**（GMSC，Gateway Mobile Switching Center）和**媒体网关**（MGW，Media Gateway）元素，网关通过电路交换核心网络来建立连接，比如**公共交换电话网络**（PSTN，Public

Switched Telephone Network)。

数据服务已超越移动电话网络中曾经的业务，成为更为重要的一部分，开始时的服务只有短消息和早期数据包数据服务，比如 GSM 系统中的**通用数据包无线业务**（GPRS，General Packet Radio Service)。这些旧的数据服务运行速率只有几十个 kbps，但用户想要的更多。新的移动电话网络可以 Mbps 的速率传送数据包数据。为了便于比较，语音通话的速率为 64 kbps，一般压缩率达到 3～4 倍。

为了运载所有这些数据，UMTS 核心网络节点直接连接到数据包交换网络。**服务 GPRS 支撑节点**（SGSN，Serving GPRS Support Node）和**网关 GPRS 支撑节点**（GGSN，Gateway GPRS Support Node）将来自移动电话和空中接口的数据包传递到外部数据包网络（比如 Internet)，以及接受来自外部数据包网络的数据包并传递到移动电话和接口。

在目前正在规划和部署的移动电话网络中，这种转换会一直继续下去。手机甚至使用 Internet 协议来建立连接，以便通过数据网络的数据包进行语音通话，通话采用了 IP 语音的方式。从无线接入网络到核心网络的全程都使用 IP 和数据包。当然，IP 网络的设计方式也在不断地发生变化，以便支持更好的服务质量。如果它没有相应的服务质量，则破碎音频和跳跃视频问题就不能打动付费用户。我们在第 5 章将返回到这个主题。

移动电话网络和传统 Internet 之间的另一个差异是移动性。当用户移动出一个蜂窝基站的覆盖范围进入到另一个蜂窝基站的覆盖范围时，数据流必须从旧蜂窝基站重新路由到新蜂窝基站。这种技术称为**移交**（handover）或**切换**（handoff)，如图 1-32 所示。

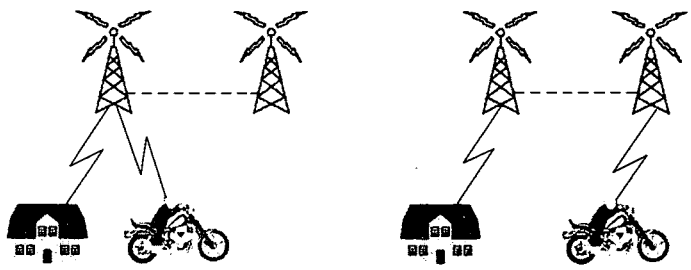


图 1-32 移动电话移交
(a) 之前；(b) 之后

当信号质量下降时移动设备或基站都可能请求移交。在一些基于 CDMA 技术的蜂窝网络中，有可能在与旧基站连接断开之前就连接到新基站。这样可提高移动连接的质量，因为服务期间不会出现中断；移动用户在那么短的一瞬间实际上与两个基站同时连接。这样移交方式称为**软切换**（soft handover)，以示区别于一个**硬切换**（hard handover)。在硬切换中，移动用户先与旧的基站断开连接，然后再与新基站建立连接。

首先要解决的一个与此相关问题是当有入境呼叫（即来电）时如何找到一个移动用户。每个移动电话网络在核心网络中维持一个**归属用户服务器**（HSS，Home Subscriber Server)，该服务器知道每个用户的位置，以及其他用于身份验证和授权的个人资料信息。在这种方式中，可以通过联系 HSS 来找到每个移动用户。

最后一个要讨论的领域是安全问题。从历史上看，长期以来电话公司采取了比 Internet 公司更为严格的安全措施，这是因为服务账单和避免（付款）欺诈所要求的。不幸的是，

除此之外没有其他安全措施可说。然而，在从1G到3G技术的演进过程中，移动电话公司已经能够为移动用户推出一些基本的安全机制。

从2G的GSM系统开始，移动电话分为手机和一个可移动的芯片两部分。这个可移动芯片包含了用户的身份和账户信息，被非正式地称为SIM卡（SIM card），这是**用户识别模块**（Subscriber Identtity Module）的简称。SIM卡可以切换到不同的手机来激活它们，它们提供了安全的基础。当GSM用户到其他国家度假或商务旅行时，他们往往带着自己的手机，但抵达目的地之后花几美元买一张新的SIM卡，就可以在本地通话而无须付漫游费。

为了减少欺诈，SIM卡上的信息被移动电话网络用来验证用户和检查他们是否被允许使用网络。在UMTS网络中，手机也使用SIM卡上的信息来检查它是否连接到了一个合法的网络。

安全的另一个方面是用户隐私。无线信号能广播到附近的所有接收器，因此为了使通话难以被窃听，可以用SIM卡上的加密密钥对传输的通话进行加密。这种方法提供了比1G系统更好的隐蔽性，1G系统很容易被窃听。但加密方法也不是万能的，因为加密方案本身也有弱点。

移动电话网络注定要在未来的网络中发挥核心作用。现在它们的移动宽带应用已经远远多于语音通话，这对空中接口、核心网络体系结构和未来网络的安全性产生了重大影响。更快更好的4G技术正在描绘着所谓**长期演进**（LTE，Long Term Evolution）的蓝图，即使3G设计和部署还在继续。其他无线技术也为固定和移动客户提供宽带Internet接入服务，最著名的是802.16网络，其通用名称是WiMAX。在向未来网络的发展过程中，LTE和WiMAX完全有可能进入一个与对方产生冲突的状态，但冲突后究竟会发生什么却很难预测。

1.5.3 无线局域网：802.11

几乎在笔记本电脑一出现，许多人就有一个梦想，当他们走进办公室时随身携带的笔记本电脑会神奇地自动与Internet连接。因此，不同的工作组开始研究实现这一目标的工作方式。最实际的做法是在办公室和笔记本电脑上都配备短程无线电发射机和接收机，以便它们实行通信。

这一领域的工作迅速引导各种各样的公司进入无线局域网市场，销售各自的产品。麻烦的是，没有任何两个产品相互兼容。标准的发散意味着配备了品牌X无线接口的计算机将无法与配置在同一个房间里品牌Y的基站正常工作。在20世纪90年代中期，工业界决定最好制定一个无线局域网标准，所以从事标准化有线局域网的IEEE委员会被赋予了制定无线局域网标准的任务。

第一个要做的决定最简单：将要制定的标准叫什么。所有其他局域网标准都有一个编号，比如802.1、802.2和802.3，直到802.10，所以无线局域网标准被冠以802.11。它的常用俚语是WiFi，但这是一个重要标准并值得尊重，所以我们还是用恰当的名称来称呼它，即802.11。

接下来的其余问题更难。第一个问题是找到一个合适的频段，并且是可用的，最好在全世界范围内都可用。所采取的解决办法与移动电话网络使用的方法恰好相反。不是使用

昂贵的需要获得许可的频谱，802.11 系统运行在无须授权的频段上，比如工业科学医疗频段（ISM，Industral Scientific, and Medical），这些频段由 ITU-R 定义（例如，902～928 MHz、2.4～2.5 GHz 和 5.725～5.825 GHz）。所有设备都可以使用这个频谱，只要它们限制自己的发射功率，以便允许不同的设备并存。当然，这意味着 802.11 无线电可能会发现自己必须与无绳电话、车库开门器和微波炉竞争。

802.11 网络由客户（比如笔记本电脑和移动电话）和称为**接入点**（AP，Access Point）的基础设施组成。通常 AP 被安装在建筑物内，有时也称为**基站**（base station）。接入点连接到有线网络上，所有客户之间的通信都要通过接入点进行。客户也可以与位于无线电范围内的其他客户直接交谈，比如在一个没有接入点的办公室内，两台计算机直接进行通信。这种通信方式称为**自组织网络**（ad hoc network）。这种模式的使用比接入点模式的使用往往少得多。两种模式如图 1-33 所示。

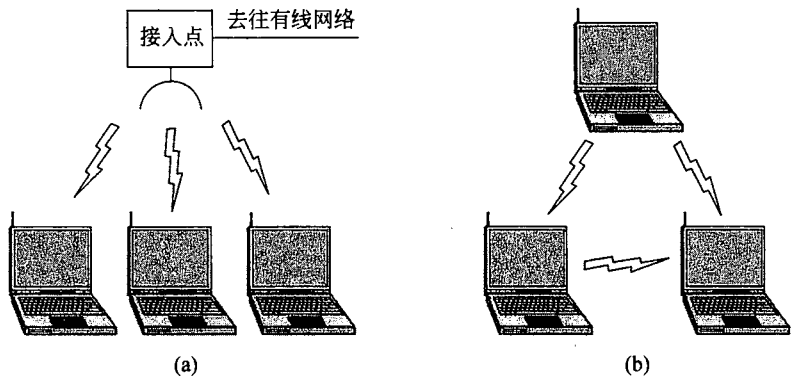


图 1-33
（a）有一个接入点的无线网络；（b）自组织网络

802.11 传输随着无线条件的变化而异常复杂，哪怕是环境中的很小一点变化都会引起无线传输的不同效果。在所使用的 802.11 频率上，无线电信号可以被固态物体反射出去，使得一个传输的多个回波可能沿不同的路径到达接收器。回波可能相互抵消或互为因果，造成接收到的信号出现大幅波动。这种现象称为**多径衰落**（multipath fading），它如图 1-34 所示。

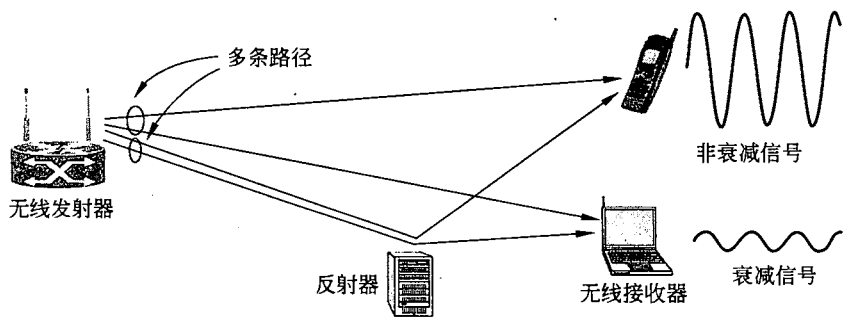


图 1-34 多径衰落

克服这种可变无线环境的关键想法是**路径的多样性**（plath diversity），或沿多条独立的路径发送信息。这样，即使由于衰落造成其中一条路径上的无线条件变差，接收器还是有

可能收到信息。这些独立的路径通常内置在物理层的数字调制方案。调制方案的选项包括使用整个允许频段中的不同频率、遵循不同天线之间的不同空中路径，或者在不同时间段内重复比特。

不同版本的802.11使用了所有这些技术。最初标准（1997）定义的无线局域网运行在1 Mbps或2 Mbps，采用频率跳跃或将信号扩展到所允许频谱的方法。几乎马上人们就抱怨它太慢了，所以工作组开始制定更快的标准。扩频设计得到进一步扩展，并成为运行速率高达11 Mbps的802.11b标准（1999年）。802.11a（1999年）和802.11g（2003年）标准切换到了一个不同的调制方案，该方案称为**正交频分复用**（OFDM，Orthogonal Frequency Division Multiplexing）。OFDM将频谱的宽带分成许多窄带，不同的比特在这些窄带上并行发送。我们将在第2章学习这种改善方案，它将802.11a/g的比特率推到了54 Mbps。虽然这已经是一个显著的进步，但人们仍然希望有更多的吞吐量来支持更苛刻的使用。最新版本是802.11n（2009年）。它采用了更宽的频带，而且每台计算机最多可以有4根天线，达到的速率高达450 Mbps。

由于无线本质上是一种广播介质，802.11无线电还必须处理多个传输同时进行发生而导致的冲突问题，因为同时传输可能会干扰信号的接收。为了解决这个问题，802.11采用了**载波侦听多路访问**（CSMA，Carrier Sense Multiple Access），该方案借鉴了经典有线以太网的设计思想。具有讽刺意义的是，以太网的设计吸取了一个在夏威夷开发的早期无线网络思想，该网络称为ALOHA。计算机在发送前等待一个随机时间间隔，如果它们听到别人已经在发送则推迟自己的发送。这个方案使得两台计算机在同一时间发送的可能性比较小。然而，在无线环境下，它不能像在有线网络情况下工作得那么好。要明白为什么会这样，请考察图1-35。假设计算机A正在给计算机B传输数据，但是A发射器的无线范围太短无法到达计算机C。如果C也要发送数据给B，它可以在开始之前先侦听，但事实上它并没有听到任何东西，这并不意味着它的传输一定会成功。C在开始之前无法听到A将会导致双方的传输发生冲突。发生任何冲突后，发送方必须等待另一个较长的随机时间，然后重发数据包。尽管有这样和那样的一些其他问题，但这个方案实际上工作得足够好。

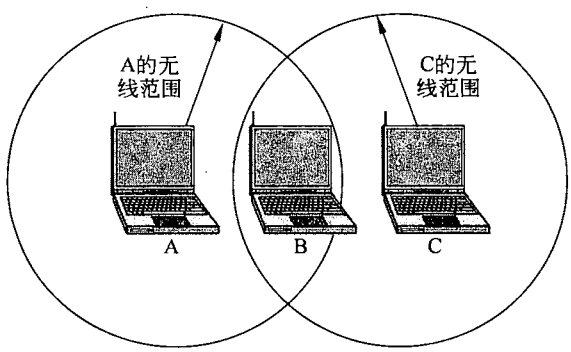


图1-35 一个无线电的传输范围不能覆盖整个系统

另一个问题是移动性。如果移动客户从它正在使用的接入点移动到另一个接入点的覆盖范围内，需要完成一些移交工作。对应的解决方案是802.11网络可以由多个蜂窝组成，每个都有其自己的接入点，并且通过一个分布式系统把这些蜂窝连接在一起。分布式系统

往往是交换式以太网，但它可以使用任何其他技术。随着客户的移动，他们可能会发现另一个信号比他们目前正在使用的信号质量更好，于是改变自己与接入点的关联。从外部来看，整个系统看起来就像是一个单一的有线局域网。

这就是说，到目前为止相比移动电话网络的移动性，802.11 具有有限的移动性。典型情况下，游牧客户从一个固定位置移动到另一个固定位置时，可以使用 802.11 与 Internet 连接，通常不是“在路上”使用。游牧使用并不真的需要移动性。即使使用 802.11 的移动性，它也只是延伸了单个 802.11 网络，至多能覆盖一座大型建筑物。未来的解决方案需要为跨越不同网络和不同技术的客户提供移动性（例如，802.21）。

最后，还有安全问题。由于无线传输是广播性质的，附近的计算机很容易接收到并非发给它们的信息包。为了防止出现这种情况，802.11 标准还包括了一个称为**有线等效保密**（WEP，Wired Equivalent Privacy）的加密方案。当时的想法是让无线网络的安全像有线网络的安全一样，这是一个好主意。但不幸的是，这个方案有缺陷，并且很快被攻破（Borisov 等，2001）。此后它已被更新的方案所替代，802.11i 标准给出了不同的加密细节，这个方案也称为 **WiFi 保护接入**（WiFi Protected Access），最初称为 WPA，现在更名为 WPA2。

802.11 已经引发了一场无线网络的革命，并且一直持续着。除了建筑物之外，它开始被安装在火车、飞机、船只和汽车上，使人们无论身在何处都可以在网上冲浪。移动电话和所有形式的消费电子类产品，从游戏机到数码相机都可以用它进行通信。我们在第 4 章将对此进行详细讨论。

1.5.4 RFID 和传感器网络

到目前为止，我们已经学习过的网络都是由计算设备组成的，无论是计算机，还是移动电话，都有计算能力，因此这些设备很容易被识别。**无线射频识别**（RFID，Radio Frequency IDentification）技术则可以让日常物品也成为计算机网络的一部分。

RFID 标签看起来像一个邮票大小的贴纸，可贴（或嵌入）在某个对象上，因此可以用来跟踪该对象。对象可能是一头牛、一本护照、一本书或一个装运货盘。标签由两部分组成：一个带有唯一标识符的小芯片和一个接收无线电传输的天线。RFID 读写器被安装在跟踪点，当对象进入特定范围时，RFID 读写器可发现它们携带的标签并询问它们的信息，如图 1-36 所示。相关应用范围很广，包括检查身份、管理供应链、计时比赛以及取代条形码等。

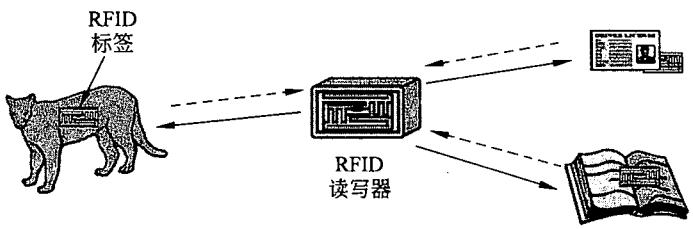


图 1-36 RFID 可用来联网日常物品

RFID 技术有许多不同种类，每一种都各有不同的属性；但也许 RFID 技术方面最迷人

的是大多数 RFID 标签既没有电插头也不用电池。相反，操作所需要的全部能量来自 RFID 读写器提供的无线电波形式。这项技术称为**无源 RFID**（passive RFID），以示区别于（很少见的）**有源 RFID**（active RFID）。在有源 RFID 中，标签上有一个电源。

RFID 的常见形式之一是**超高频 RFID**（UHF RFID，Ultra-High Frequency RFID）。它主要用在货运托盘和一些驾驶执照上。在美国，RFID 读写器可在 902～928 MHz 频段发送信号。数米范围内的标签通过改变它们反射读写器信号的方式进行通信，读写器能检测到这些反射信号。这种操作方式称为**后向散射**（backscatter）。

另一种流行的 RFID 是**高频 RFID**（HF RFID，High Frequency RFID）。它的工作频率为 13.56 MHz，可以用在护照、信用卡、书籍和非接触式支付系统中。因为其物理机制基于感应而不是后向散射，因此高频 RFID 的传输距离较短，典型范围在一米以内。还有使用其他频率的其他形式 RFID，比如**低频 RFID**（LF RFID，Low Frequency RFID），这是在高频 RFID 之前开发的，主要用于动物跟踪。它是一种可能用在你宠物猫身上的 RFID。

RFID 读写器必须以某种方式解决读写器范围内存在多个标签的问题。这意味着一个标签在听到读写器的信号时不能简单地做出回应，否则从多个标签发出的信号可能会发生冲突。解决的办法类似于 802.11 所采取的方法：标签在用自己的标识符响应 RFID 读写器之前，等待一段随机的短时间，以便允许阅读器聚焦到单个标签，并进一步询问它们的信息。

安全性是另一个问题。RFID 读写器具备轻松跟踪对象的能力，因此使用它的人可以通过它侵犯个人隐私。不幸的是，很难确保 RFID 标签的安全，因为它们缺乏计算和通信必需的能量来运行强大的加密算法。相反，RFID 使用了相应微弱的措施，比如密码（可以很容易被破解）。如果身份证可以被远程的边境官员读取，那么还有什么能阻止在你不知情的情况下被其他人跟踪你的身份证吗？真没多少。

RFID 标签刚开始时只作为标识芯片，但很快就转向成为全面配置的计算机。例如，许多标签都有内存，可被更新和查询。这样的 RFID 可以用来记录或存储针对有关对象发生了什么的事件信息。（Rieback 等，2006）表明这意味着所有计算机恶意软件的通常问题都会在 RFID 上发生，现在你的猫或你的护照都可能会被用来传播 RFID 病毒。

进一步加强 RFID 能力的是传感器网络。传感器网络的部署可用来监测物理世界的各个方面。到目前为止，它们大多用于科学实验，例如监测鸟类栖息地、火山活动和斑马迁移等，但商业应用，包括医疗保健、监测振动设备、跟踪冷冻、冷藏或其他易腐货物等方面的应用虽然有点落后，毕竟紧随其后。

传感器节点是一台小型计算机，往往只有一个钥匙环大小，它通常具有温度、振动和其他传感器。许多节点被放置在要监测的环境中。一般情况下，它们用电池，尽管它们可以通过震动或太阳来汲取能量。与 RFID 一样，拥有足够的能量是传感器网络面临的一个关键挑战，所有节点必须谨慎通信以便将它们的传感信息传递给一个外部数据搜集点。常用的策略是各个节点自组织地中继彼此的消息，中继方式如图 1-37 所示。这种设计称为**多跳网络**（multihop network）。

RFID 和传感器网络有可能在未来变得功能更强大更普适。研究人员已经将这两种技术的优点结合起来，开发出可编程的 RFID 标签，使其具有光照、移动和其他传感器能力（Sample 等，2008）。

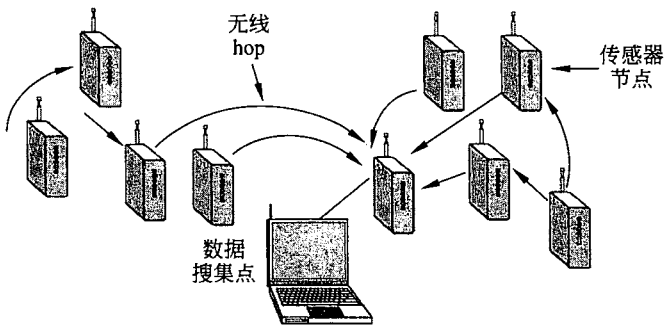


图 1-37 传感器网络的多跳拓扑结构

1.6 网络标准化

世界上有许多网络生产商和供应商，它们都有自己的思维模式和行为方式。如果不加以协调，事情就会变得混乱不堪，用户将无所适从。摆脱这种局面的唯一办法是大家都遵守一些网络标准。好的标准不仅使不同的计算机可以相互进行通信，而且还能扩大遵循相应标准的产品市场。大的市场可以促使大批量的生产、制造业的规模经济、更好的实现，以及其他一些好处，比如更低的价格、用户接受程度的提高。

在这一节中，我们将快速审视非常重要但鲜为人知的国际标准化世界。首先让我们弄明白标准究竟是什么。一个通情达理的人或许会假设，标准就是说明协议应该如何工作，因为只有这样才能很好地实现它。但这个人是错误的。

标准定义了互操作所需要的一切：不能多，也不能少。这样才能促使更大的市场出现，也让企业之间在它们的产品质量上展开竞争。例如，802.11 标准定义了多种传输速率，但却没有规定发送方应使用哪个速率，这是一个良好性能的关键因素。完全由生产产品的厂商来决定。在这种方式下，由于有太多的实现可以选择，而且标准通常又定义了许多选项，因此获得互操作性是很难的。对于 802.11，因为存在这么多的问题，而且实际上已经形成了一个策略，因此一个名为 WiFi 联盟（WiFi Alliance）的贸易集团开始就 802.11 标准的互操作性开展工作。

同样，一种协议标准定义了介质之上运行的协议，但没有定义设备内部的服务接口，当然有助于解释协议的那部分接口除外。真正的服务接口通常是专有的。例如，一台计算机系统内 TCP 与 IP 的接口方式与该计算机和远程主机通信没有多大的关系。它只和远程主机是否运行 TCP/IP 协议有关。事实上，TCP 和 IP 普遍实现在一起，两者之间没有任何明显的接口。这就是说，良好的服务接口对于使用协议是很宝贵的，比如良好的 API，而且最好的接口（例如 Berkeley 的套接字）可能会变得非常受欢迎。

所有的标准可以分为两大类：事实标准和法定标准。事实标准是指那些已经发生但没有任何正式计划的标准。“事实”（De facto）一词是“来自事实”（from the fact）的拉丁语。作为 Web 运行基础的 HTTP 协议就是作为一个事实标准而存在的。它是早期万维网浏览器的一部分，它的使用随着 Web 增长而腾飞。最早的 Web 浏览器由 CERN 的 Tim Berners-Lee 开发。蓝牙是事实标准的另一个例子。它最初由爱立信公司开发，但现在几乎每个人都在

用它。

相反，法定标准是指由一些正规标准化组织采纳的规则。“法定”（De jure）一词是“依据法律”（by law）的拉丁语。国际性的标准化权威组织通常可以分成两类：国家政府通过条约建立起来的组织和自愿的非条约组织。在计算机网络标准的领域中，每一类都有一些组织，著名的有ITU、ISO、IETF和IEEE，下面分别进行讨论。

实际上，标准、企业和标准化机构之间的关系是错综复杂的。事实上的标准往往演变成为法律上的标准，特别是如果标准很成功的话。HTTP就是这种情况的一个例子，它很快被IETF采纳。标准化组织通常认可彼此的标准，目的是增加技术市场，看起来就像朋友之间互相友好拍背问候一样。最近以来，围绕着特定技术形成的许多自组织商业结盟在制定和完善网络标准过程中也发挥着很大的作用。例如，第三代合作伙伴计划（3GPP，Third Generation Partnership Project）是一个电信协会之间的合作组织，旨在推动UMTS 3G移动电话标准。

1.6.1 电信领域有影响力的组织

全世界电话公司的法律地位在不同的国家之间有很大的差异。美国是一个极端例子，它有2000多个独立的（大多数都很小）、私有的电话公司。随着AT&T公司在1984年被分解（AT&T曾经是世界上最大的公司，它为全美80%左右的电话提供电话服务），以及1996年的电信法（Telecommunications Act）修订了监管，从而进一步促进了竞争，很多公司加入到电信领域。

另一个极端是这样一些国家：所有的通信都由国家政府完全垄断，包括邮件、电报、电话，经常还包括广播电台和电视。世界上大多数国家都可归到这一类中。在有些情况下，电信管理局是一个国家公司，而在其他一些情况下，电信管理局只是政府的一个分支机构，通常称为邮电部（PTT，Post，Telegraph&Telephone administration）。全球范围内的发展趋势是朝着自由、竞争脱离政府垄断的方向发展。大多数欧洲国家已经将它们的PTT进行了（部分）私有化改造，但在其他一些地方，这个进程仍然非常缓慢。

有了这么多不同的服务供应商，很显然有必要提供全球范围内的兼容性以确保一个国家的用户（和计算机）可以呼叫另一个国家的用户或者计算机。实际上，这种需求很早以前就存在了。在1865年，欧洲许多政府的代表聚集在一起，形成了一个标准化组织，就是今天**国际电信联盟**（ITU，International Telecommunication Union）的前身。它的任务是对国际电信进行标准化。在当时，所谓国际电信只是指电报。即使在那个时候也很明显：如果一半的国家使用莫尔斯编码而另一半国家使用其他编码，那么国与国之间的电报发送就有问题。当电话也变成一种国际服务时，ITU又承担起了电话标准化的工作。1947年，ITU成为联合国的一个机构。

ITU有大约200名政府成员，包括几乎所有联合国会员国。由于美国没有PTT，但必须有人代表美国出现在ITU中，这个任务落到了美国国务院（the State Department）。可能的理由是ITU必须与外国打交道，而这正是美国国务院所擅长的。ITU有700多个部门和准成员，包括电话公司（例如，AT&T、Vodafone、Sprint公司）、电信设备制造商（例如，思科、诺基亚、北电）、计算机供应商（例如，微软、安捷伦、东芝）、芯片制造商（例如，

英特尔、摩托罗拉、德州仪器）和其他感兴趣的公司（例如，波音公司、哥伦比亚广播公司、VeriSign 公司）。

ITU 有三个主要部门。我们将焦点主要集中在 ITU-T 上，这是电信标准化部门，主要关注电话和数据通信系统。1993 年以前，ITU-T 称为 CCITT，这是如下法文名字中的首字母缩写：Comité Consultatif International Télégraphique et Téléphonique。ITU-R 是无线电通信部门，主要协调全球无线电频率利益集团之间的竞争使用。另一个部门是 ITU-D，即发展部门，它的主要任务是促进信息和通信技术的发展，以便缩小有效获取信息技术的国家和访问受到限制的国家之间的“数字鸿沟”。

ITU-T 的任务是对电话、电报和数据通信接口做出技术性的建议。这些建议通常会变成国际上认可的标准，尽管技术上的建议仅仅是建议，政府可以采纳也可以忽视，根据它们自己的意愿（因为政府就像 13 岁的男孩子，不会欣然接受被命令做什么）。实际上，一个国家采纳一个与世界各地所用标准不同的电话标准是完全自由的，但是必须付出脱离所有其他人与世隔绝的代价。

ITU-T 的实际工作主要由它的研究组完成。它当前有 10 个研究组，通常有 400 多人，覆盖的主题从电话计费到多媒体服务再到安全。例如，SG15 负责标准化普遍被用来连接 Internet 的 DSL 技术。为了尽可能地完成所有该做的事情，研究组又分成各个**工作组**（Working Party），工作组又进一步分为专家组（Expert Team），专家组再分为**专案小组**（ad hoc group）。可见，一旦成为官僚，就总要遵循官僚主义的行事方式。

尽管如此，ITU-T 实际还是完成了很多事情。自从它成立以来，已经产生了 3000 多份建议，大多数建议被广泛应用于实践中。例如，H.264（也是 ISO 标准 MPEG-4 AVC）被广泛用于视频压缩，X.509 公钥证书被用于安全的 Web 浏览和数字签名电子邮件。

自 20 世纪 80 年代开始，电信业开始从整个国家性质转变成全球性的行业，随着这种转变的完成，标准变得越来越重要；而且，越来越多的组织希望参与到标准制订工作中来。关于 ITU 的更多信息，请参看（Irmer，1994）。

1.6.2 国际标准领域有影响力的组织

国际标准是由**国际标准化组织**[1]（ISO，International Standards Organization）制定和发布的，这是一个成立于 1946 年自愿的、非条约性质的组织。它的成员是 157 个成员国的国家标准组织。这些成员包括 ANSI（美国）、BSI（英国）、AFNOR（法国）、DIN（德国）以及其他 153 个成员。

ISO 为大量的学科制订标准，从螺钉和螺帽，一直到电话架的外形（更何况可可豆（ISO 2451）、渔网（ISO 1530）、女式内衣（ISO4416）和不少其他人们不认为可以标准化的东西）。在有关电信标准方面，ISO 和 ITU-T 经常合作（ISO 是 ITU-T 的成员），以免被人讽刺两个官方组织互不兼容的国际标准。

目前已经颁布的标准已经超过了 17 000 个，其中包括 OSI 标准。ISO 有 200 多个**技术委员会**（TC，Technical Committee），这些技术委员会按照创建的顺序进行编号，每个技术

1 确切地说，ISO 的真实名称是标准化的国际组织。

委员会处理一个特定主题。TC1 处理螺钉和螺帽（对螺丝钉的螺纹和斜度进行标准化）事务。 JTC1 处理信息技术，包括网络、计算机和软件。它是第一个（迄今为止只有一个）**联合技术委员会**（Joint Technical Committee），创建于 1987 年；通过合并 TC97 和 IEC 活动组建的 IEC 是另一个标准化机构。每个 TC 有**子委员会**（SC，subcommittee），**子委员会**再细分为**工作组**（WG，working group）。

WG 的实际工作大部分由全球超过 100 000 个志愿者完成。其中许多“志愿者”是被他们的老板指派为 ISO 工作的，因为这些公司的产品正在进行标准化。其他一些“志愿者”则是政府官员，他们期望自己国家做的事情能变成国际标准。学术领域中的专家在许多 WG 中也很活跃。

ISO 采纳国际标准的程序是经过精心设计的，目的是尽可能获得广泛的一致同意。当某个国家标准化组织需要某个领域的国际标准时，就启动标准化程序。首先组成一个工作组，由工作组提出一个**委员会草案**（CD，Committee Draft）；然后该草案被传送给所有的成员审核，他们有 6 个月的时间来评价这份草案。如果绝大多数成员都同意该草案，则再生成一份修订文档，称为**国际标准草案**（DIS，Draft International Standard）；然后该标准草案被发给成员传阅征求意见，并进行投票表决。根据这一轮的结果，形成**国际标准**（IS，International Standard）的最后文本，然后在获得认可之后公开发布。在有较大争议的领域，委员会草案或者国际标准草案可能需要经过几次修订，才能获得足够的投票通过票数，整个过程可能要持续几年。

国家标准和技术协会（NIST，National Institute of Standards and Technology）是美国商业部的一个部门，以前称为**国家标准局**（National Bureau of Standards）。它颁发的标准是美国政府采购必须强制性执行的。当然，美国国防部除外，它有自己的标准。

在标准化领域中另一个有很大影响的组织是**电气和电子工程师协会**（IEEE，Institute of Electrical and Electronics Engineers），它是世界上最大的专业组织。除了每年发行大量的杂志和召开几百次会议以外，IEEE 有一个标准化组，该标准化组专门开发电气工程和计算领域中的标准。IEEE 的 802 委员会已经标准化了很多类型的局域网。我们在本书后面将要学习其中一些标准。实际的工作是由许多工作组完成的，如图 1-38 所列。各个 802 工作组的成功率并不高，有了 802.x 这样的编号并不保证会成功。但是成功的故事（特别是 802.3 和 802.11）对工业界和全球的影响却是非同小可。

1.6.3 Internet 标准领域有影响力的组织

全球性的 Internet 有它自己的标准化机制，这个标准化机制与 ITU-T 和 ISO 的标准化机制截然不同。它们之间的区别可以粗略地类比为参加 ITU 或者 ISO 标准会议的人总是穿戴整齐，而参加 Internet 标准会议的人则穿着休闲（除了在 San Diego 开会外，其余会议都穿着短裤和 T 恤衫）。

ITU-T 和 ISO 会议的参会者由企业官员和政府公务员组成，对于他们来说标准化是他们的工作。他们把标准化看作是一件大好事，而且致力于这项工作。相反，Internet 领域中的人则更喜欢无政府状态，他们把这种自由当作一种原则。然而，成千上万的人都在做自己的事情，他们很少有交流。因此，虽然令人遗憾，有时候标准化还是需要的。在这样的

编号	主题
802.1	LAN概述与体系结构
802.2 ↓	逻辑链路控制
802.3 *	以太网
802.4 ↓	令牌总线（主要被用在制造业）
802.5	令牌环（IBM的LAN技术）
802.6 ↓	双队列双总线（早期城域网）
802.7 ↓	宽带技术的技术咨询组
802.8 †	光纤技术的技术咨询组
802.9 ↓	同步LAN（实时应用）
802.10 ↓	虚拟LAN和安全
802.11 *	无线LAN（WiFi）
802.12 ↓	优先级需求（HP的AnyLAN）
802.13	不吉利的数字；没人愿意使用
802.14 ↓	线缆调制解调器（已消失：行业协会首先使用的）
802.15 *	个域网（蓝牙，Zigbee）
802.16 *	宽带无线（WiMAX）
802.17	弹性数据包环
802.18	无线电规范问题的技术咨询组
802.19	所有标准并存的技术咨询组
802.20	移动宽带无线（类似于802.16e）
802.21	介质独立切换（在不同技术间漫游）
802.22	无线区域网

图 1-38 802 工作组

最重要的用 * 标注；标注为 ↓ 的处于冬眠；标注 † 的为放弃已经解散

氛围下，MIT 的 David Clark 曾经对 Internet 标准化做过一个现今闻名的注释，即 Internet 标准是由“粗糙的共识和可运行代码”（rough consensus and running code）组成的。

在建立 ARPANET 时，美国国防部成立了一个非正式委员会来监督它的运行。1983 年，该委员会更名为 **Internet 活动委员会**（IAB，Internet Activities Board），并且被赋予了更多的使命，即保持 ARPANET 和 Internet 的研究人员大致朝着同一个方向前进，这有点像赶着一群猫往前走一样。缩写词 IAB 后来改为 **Internet 体系结构委员会**（Internet Architecture Board）。

大约每 10 个 IAB 成员牵头从事某个重要方面的工作。IAB 每年开几次会议，讨论研究的结果，并且将讨论意见反馈给美国国防部和 NSF，因为当时的大部分研究经费是由美国国防部和 NSF 提供的。当需要一个新的标准时（比如，一种新的路由算法），IAB 成员就会研究出对应的新标准，然后宣布新标准带来的变化，于是，那些作为软件领域中坚力量的研究生就开始实现该标准。这里的交流过程是通过一系列技术报告进行的，这些报告统称为**请求注释**（RFC，Request For Comments）。RFC 被存储在网上，任何感兴趣的人都可以从 www.ietf.org/rfc 获取它们。所有的 RFC 都按照创建的时间顺序编号，现在已经有超过 5000 个 RFC。本书中我们将会引用到很多 RFC。

到了 1989 年，Internet 增长得如此之快，以至于这种高度非正式的风格无法适应网络的快速变化。当时许多厂商已经开始提供 TCP/IP 产品，它们不想仅仅因为数十个研究人员

有了更好的思路就改变这些产品。于是，在1989年夏季，IAB被重新改组。研究人员被重组到**Internet研究任务组**（IRTF，Internet Research Task Force）中，Internet研究任务组和**Internet工程任务组**（IETF，Internet Engineering Task Force）一起成为IAB的附属机构。后来，IAB又接纳了更多的人参与进来，他们代表了更为广泛的组织，而不仅仅代表研究群体。IAB最初是个“传宗接代”的组，其中的成员服务年限为两年，新成员由老成员指定。后来，建立了**Internet协会**（Internet Society），它由许多对Internet感兴趣的人组成。因此，从某种意义上讲，Internet协会可以与ACM或者IEEE相提并论。它由选举出来的理事会管理，理事会指定IAB成员。

这种将IAB划分成两个组织的思路带来的好处显而易见，IRTF更加专注于长期的研究，而IETF处理短期的工程事项。IETF被分成很多工作组，每个组解决某一个特定问题。初期时，这些工作组的主席合起来组成指导委员会，指导整个工程组的工作。工作组的主题包括新的应用、用户信息、OSI整合、路由与编址、安全、网络管理以及标准。后来，工作组的数量太多（超过了70个），只好按照领域再细分，每个领域的主席集中起来组成指导委员会。

此外，IAB还按照ISO模式采纳了一个更加正式的标准化过程。为了将一个基本思想变成一个**标准提案**（Proposed Standard），首先要在RFC中完整地描述整个思想，并且在Internet社团中引起足够的兴趣，使得人们愿意考虑它。为了进一步推进到**标准草案**（Draft Standard）阶段，必须有一个可正常工作的实现，并且必须经过至少两个独立网站、历时4个月以上的严格测试。如果IAB确信这个想法是健全的，并且软件可以工作，那么它可以声明该RFC成为**Internet标准**（Internet Standard）。有些Internet标准已经成为美国国防部标准（MIL-STD），从而成为美国国防部供应商的强制标准。

对于Web标准，万维网联盟（W3C，World Wide Web Consortium）负责开发协议和给出促进Web长期增长的指导意见。它是一个行业联盟，由Tim Berners-Lee领导，成立于1994年Web真正开始腾飞之时。W3C现在有来自世界各地的300多名成员，已经制定了100多个W3C建议标准，标准覆盖的主题正如其标准的名称所言，比如HTML和Web隐私。

1.7 度量单位

为了避免出现可能的混淆，这里有必要明确声明，根据计算机科学的一贯做法，本书并不采用传统的英国度量单位（弗隆/浪-14磅-14天系统）。主要的度量前缀如图1-39所列。这些前缀通常取它们的首字母缩写，再加上单位的首字母缩写（大写）构成复合单位（比如KB、MB等）。不过，有一个例外（由于历史原因），kbps代表的是kilobits/sec（千位/秒）。因此，1 Mbps通信线路可以传输10^6位/秒，并且，每10^{-10}秒有100 psec（或者100 ps）时钟滴答。由于milli和micro都以字母m开头，所以两者必须进行区分。通常情况下，m代表毫（milli），而μ（希腊字母）代表微（micro）。

另外，还值得指出的是，计算内存、硬盘、文件和数据库大小的计量单位按照工业界的实际做法与实际含义有细微的区别。在这里，kilo是指2^{10}（1024），而不是10^3（1000），因为内存总是2的幂次方。因此，1KB内存包含1024字节，而不是1000字节。还要注意，

指数	显式表示	前缀	指数	显式表示	前缀
10^{-3}	0.001	milli	10^{3}	1 000	Kilo
10^{-6}	0.000001	micro	10^{6}	1 000 000	Mega
10^{-9}	0.000000001	nano	10^{9}	1 000 000 000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1 000 000 000 000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1 000 000 000 000 000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1 000 000 000 000 000 000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1 000 000 000 000 000 000 000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1 000 000 000 000 000 000 000 000	Yotta

图 1-39 主要的度量前缀

这种使用情况下的大写字母 B 表示字节（bytes）（8 比特），而不是像小些字母 b 一样表示 bit。类似地，1MB 内存包含 2^{20}（1 048 576）字节，1 GB 内存包含 2^{30}（1 073 741 824）字节，1 TB 数据库包含 2^{40}（1 099 511 627 776）字节。然而，1 kbps 通信线路每秒传输 1000 位，而 10 Mbps 的 LAN 运行速度为 10 000 000 位/秒，这些速度并不是 2 的幂次方。不幸的是，许多人倾向于将两种系统混淆起来，特别是磁盘的大小。在本书中，为了避免二义性，我们将使用符号 KB、MB、GB 和 TB 分别代表 2^{10}、2^{20}、2^{30} 和 2^{40} 字节；用符号 kbps、Mbps、Gbps 和 Tbps 分别代表 10^{3}、10^{6}、10^{9} 和 10^{12} 位/秒。

1.8 本书其余部分的概要

本书既讨论了计算机网络的原理，又讨论了计算机网络的实践。基本上每章都从相关原理的讨论开始，然后用一些例子来说明这些原理。这些例子通常取自 Internet 和无线网络（比如移动电话网络），因为这两个网络都非常重要，而且差别也非常大。其他的例子也会在相关地方出现。

本书的结构按照图 1-23 所示的混合模型来组织。从第 2 章开始，我们将从协议层次体系的底层开始逐层往上进行讨论。我们引入了数据通信领域中的一些背景知识，它们覆盖了有线传输系统和无线传输系统。这些内容关注于如何在物理信道上传递信息，尽管我们只讨论体系结构，而不涉及硬件方面的问题。这一章还讨论了一些物理层的例子，比如公共交换电话网、移动电话网络和有线电视网络。

第 3 章和第 4 章从两个方面讨论了数据链路层。第 3 章考查了如何在一条链路上发送数据包的问题，其中包括差错检测和差错纠正。我们把 DSL（通过电话线进行宽带 Internet 接入）作为数据链路层协议在现实世界中的一个实例而进行了相应的探讨。

我们在第 4 章考查了介质访问子层。这是数据链路层的一部分，它主要处理如何让多个计算机共享一个传输信道。我们给出的例子包括无线局域网，比如 802.11 和 RFID；同时还讨论了有线局域网，例如经典以太网。我们还讨论了可用来连接 LAN 的数据链路交换机，比如交换式以太网。

第 5 章处理网络层的问题，尤其是路由。这一章介绍了很多路由算法，有静态的，也有动态的。即使有了良好的路由算法，如果实际流量超过了网络所能承载的流量，一些数据包还是会被延迟甚至丢失。我们从如何防止出现拥塞到如何保证一定的服务质量等几个方面，逐步深入讨论了网络拥塞问题。将异构网络连接起来构成互联网络会带来大量的问题，这一章将讨论这些问题。Internet 的网络层将在这里给予广泛的介绍。

第6章处理传输层。重点主要在于面向连接的协议和可靠性，因为许多应用都需要这样的协议。这里对Internet的传输协议UDP和TCP进行了详细的讨论，同时还讨论了它们的性能问题。

第7章处理应用层的问题、协议和应用。第一个主题是DNS，这相当于Internet的电话簿。接下来讨论电子邮件，包括对电子邮件协议的讨论。然后，我们把目光转向Web，详细讨论了静态内容和动态内容，以及在客户端和服务器端发生的事情。我们紧接着考查了网络多媒体，包括流式音视频。最后讨论了内容分发网络，包括对等技术。

第8章与网络安全有关。这个主题涉及网络体系结构中的所有层次，所以，在介绍完全部层次之后再来讨论安全问题就容易得多。这一章从密码学的基本介绍开始，然后介绍了如何通过加密手段来保护通信、电子邮件和Web的安全。本章最后讨论了一些与安全相关的领域，比如隐私、言论自由、审查制度，以及其他社会问题。

第9章包含了一份建议阅读的列表，按照章节的顺序排列。这份参考列表意在帮助那些想进一步从事网络研究的读者。这一章还有一份按字母顺序排列的参考文献列表，包括了本书中引用过的文献。

作者的网站是：

http://www.pearsonhighered.com/tanenbaum

这里有一个页面，给出了指向很多教程、常见问题、企业、行业协会、职业组织、标准组织、技术、论文以及更多的内容的链接。

1.9 本章总结

计算机网络的用途非常广，既可以针对公司也可以针对个人，既可以在家里使用也可以在移动中使用。对于公司而言，使用网络通常采用客户机-服务器模型来共享公司信息，员工桌面机充当访问超强服务器的客户机。对于个人来说，网络为他提供了访问各种各样信息以及很多娱乐资源的手段，同时还可以被用来购买或者销售产品和服务。个人用户往往通过他们的手机或者家里的线缆调制解调器来访问Internet，虽然越来越多的无线接入用的是笔记本电脑和手机。技术的进步使得各种新移动应用和新网络层出不穷，这些新网络通常涉及嵌入了计算机的家电和其他消费类电子设备。同样的进步也带来了社会问题，比如隐私的关注。

粗略地讲，网络可以分为LAN、MAN、WAN和互联网络。典型的LAN覆盖一座建筑物，并且可以很高速率运行。MAN通常覆盖一座城市，比如有线电视系统，现在有许多用户通过这个网络来访问Internet。WAN可以覆盖一个国家或者一个洲。构造这些网络的技术有些是点到点的（比如，一条线缆），而其他一些技术是广播的（比如，无线）。路由器可以连接多个网络，从而形成互联网络，其中Internet是世界上最大和最好的互联网。无线网络正在变得越来越受欢迎，例如802.11 LAN和3G移动电话网络。

网络软件由网络协议组成，而协议是进程通信必须遵守的规则。大多数网络支持协议的层次结构，每一层向它的上一层提供服务，同时屏蔽掉较低层使用的协议细节。协议栈通常基于OSI模型或者TCP/IP模型。这两个模型都有数据链路层、网络层、传输层和应

用层，但它们在其他层不同。设计问题包括可靠性、资源分配、规模增长、信息安全等。这本书的大部分内容主要处理网络协议和相应的设计问题。

网络为它们的用户提供不同的服务。服务的范围从无连接的尽力而为数据包传递服务到面向连接的有质量保证服务。在某些网络中，无连接服务由某一层提供，而面向连接的服务则由它上面的层提供。

著名的网络包括 Internet、3G 移动电话网络和 802.11 无线 LAN。Internet 从 ARPANET 演变而来，那时其他网络纷纷加入到 ARPANET 中，从而构成了一个互联网。现在的 Internet 实际上是成千上万个都采用 TCP/IP 协议栈的网络集合。3G 移动电话网络为无线并且移动的用户提供了多个 Mbps 量级的 Internet 接入服务，当然也能承载语音通信。无线 LAN 基于 IEEE 802.11 标准，大多数都部署在家庭和咖啡馆，这种网络提供了速率超过 100 Mbps 的连通性。新型网络已崭露头角，例如嵌入式传感器网络和基于 RFID 技术的网络。

为了使多台计算机可相互之间通信，需要大量的标准化工作，不管是硬件方面还是软件方面。诸如 ITU-T、ISO、IEEE 和 IAB 这样的组织负责管理标准化进程的各个不同部分。

习 题

1. 假设你已经将你的狗 Bernie 训练成不仅可以携带一小瓶白兰地，还能携带一箱 3 盒 8 毫米的磁带（当你的磁盘满了的时候，你可能会认为这是一次紧急事件）。每盒磁带的容量为 7 GB 字节。无论你在哪里，狗跑向你的速度是 18 千米/小时。试问在什么距离范围内 Bernie 的数据传输速率会超过一条数据速率为 150 Mbps 的传输线（不算额外开销）？试问分别在以下情况下：（1）狗的速度加倍；（2）每盒磁带容量加倍；（3）传输线路的速率加倍。上述结果有什么变化？
2. LAN 的一个替代方案是简单地采用一个大型分时系统，通过终端为用户提供服务。试给出使用 LAN 的客户机-服务器系统的两个好处。
3. 客户机-服务器系统的性能受到两个网络因素的严重影响：网络的带宽（即，网络每秒可以传输多少位数据）和延迟（即，将第一个数据位从客户端传送到服务器端需要多少时间）。请给出一个网络例子，它具有高带宽，但也有高延迟。然后再给出另一个网络例子，它具有低带宽和低延迟。
4. 除了带宽和延迟以外，网络若要为下列流量提供很好的服务质量，试问还需要哪个参数？（1）数字语音流量；（2）视频流量；（3）金融业务流量。
5. 在存储-转发数据包交换系统中，衡量延迟的一个因素是数据包在交换机上存储和转发需要多长时间。假设在一个客户机-服务器系统中，客户机在纽约而服务器在加州，如果交换时间为 10 微秒，试问交换时间是否会成为影响延迟的一个主要因素？假设信号在铜线和光纤中的传播速度是真空光速的 2/3。
6. 一个客户机-服务器系统使用了卫星网络，卫星高度为 40 000 千米。试问在响应一个请求时，最佳情形下的延迟是什么？
7. 在未来，当每个人都有一个家庭终端连接到计算机网络时，就有可能对一个重要的未决案件进行即时公民投票。最终，现在的立法机关都可以撤销了，从而让人民直接表达他

们的意愿。这种直接民主的正面影响是非常显然的，请讨论可能产生的某些负面影响。

8. 5个路由器通过一个点到点子网连接在一起。网络设计者可以为任何一对路由器设置一条高速线路、中速线路、低速线路或根本不设置线路。如果计算机需要100毫秒来生成并遍历每个网络拓扑，试问它需要多长时间才能遍历完所有的网络拓扑？
9. 广播式子网的一个缺点是当多台主机同时企图访问信道时会造成容量浪费。作为一个简单的例子，假设时间被分成了离散的时间槽，共有n台主机；在每个时间槽内，每台主机企图访问信道的概率为p。试问由于冲突而被浪费的时间槽比例是多少？
10. 试问使用层次协议的两个理由是什么？使用层次协议的一个可能缺点是什么？
11. Specialty Paint公司的总裁打算与一个本地的啤酒酿造商合作生产一种无形啤酒罐（作为防止乱扔垃圾的一种措施）。总裁告诉她的法律部门调研此事，后者又请工程部帮忙。结果总工程师打电话给啤酒酿造公司讨论该项目的技术问题。然后两位工程师又各自向他们的法律部门作了汇报。然后，法律部门通过电话安排了有关的法律方面的事宜。最后，两位公司总裁讨论了这次合作在经济方面的问题。试问这个通信机制违反了OSI模型意义上的哪个多层协议原则？
12. 两个网络都可以提供可靠的面向连接的服务。其中一个提供可靠的字节流，另一个提供可靠的报文流。试问这两者是否相同？如果你认为相同，为什么要有这样的区别？如果不相同，请给出一个例子说明它们如何不同。
13. 在讨论网络协议的时候，“协商”意味着什么？请给出一个例子。
14. 图1-19显示了一个服务。试问该图是否还隐含着其他的服务？如果有的话，在哪里？如果没有的话，说明为什么没有。
15. 在有些网络中，数据链路层处理传输错误的做法是请求发送方重传被损坏的帧。如果一帧被损坏的概率为p，试问发送一帧所需要的平均传输次数是多少？假设确认帧永远不会丢失。
16. 一个系统具有n层协议。应用层产生长度为M字节的报文，在每一层加上长度为h字节的报文头。试问报文头所占的网络带宽比例是多少？
17. 试问TCP和UDP的主要不同是什么？
18. 图1-25（b）中的子网被设计用来对抗核战争。试问需要多少颗炸弹才能将这些节点炸成两个互不相连的集合？假设任何一颗炸弹都可以摧毁掉一个节点以及所有与它相连的链路。
19. Internet的规模差不多每隔18个月翻一倍。虽然没有人能够确切地知道具体的数字，但是估计2009年Internet上的主机数目为6亿台。请利用这些数据计算出2010年Internet上预计会有多少台主机。你相信你的结论吗？请说明你为什么相信或者为什么不相信。
20. 当在两台计算机之间传输一个文件时，可以采用两种不同的确认策略。在第一种策略中，该文件被分解成许多个数据包，接收方独立地确认每一个数据包，但没有对整个文件进行确认。在第二种策略中，这些数据包并没有被单独地确认，但是当整个文件到达接收方时会被确认。请讨论这两种方案。
21. 移动网络运营商需要知道它们用户的移动电话（因而知道它们的用户）在哪里。试问这对于用户来说，为什么很糟糕？现在，请给出为什么又很好的理由。
22. 试问在原始802.3标准中，一比特多长（按米来计算）？请使用10 Mbps传输速率，并

且假设同轴电缆的传播速度是真空中光速的2/3。

23. 一幅图像的分辨率为1024×768像素，每个像素用3字节表示。假设该图像没有被压缩。试问，通过56 kbps的调制解调器传输这幅图像需要多长时间？通过1 Mbps的线缆调制解调器呢？通过10 Mbps的以太网呢？通过100 Mbps的以太网呢？
24. 以太网和无线网络既有相同点，也有不同点。以太网的一个特性是同一时刻只能传输一帧数据。试问802.11也具有这个以太网特性吗？请讨论你的答案。
25. 请分别列出网络协议国际标准化后的两个优点和缺点。
26. 当一个系统既有永久（固定）部分又有可移动部分（比如CD-ROM驱动器和CD-ROM）时，系统的标准化显得非常重要。标准化之后，不同的公司可以同时生产永久部分和可移动部分的产品，而且这些产品总能在一起工作。请给出计算机工业界之外的三个例子，在每个例子中都有相应的国际标准。再给出计算机工业界之外的另外三个例子，但这三个例子中都不存在国际标准。
27. 假设实现第k层操作的算法发生了变化。试问这会影响到第k-1和第k+1层的操作吗？
28. 假设由第k层提供的服务（一组操作）发生了变化。试问这会影响第k-1层和第k+1层的服务吗？
29. 请列出一些理由说明客户端的响应时间有可能大于最好情况下的延迟。
30. 请问在ATM网络中使用小的固定长度的信元有什么缺点？
31. 列出你每天与使用计算机有关的活动。如果这些网络突然间全部关闭，试问你的生活将会有什么变化？
32. 找出你所在学校或工作单位使用的网络。请描述这些网络的类型、拓扑结构和交换方式。
33. ping程序使你能够给指定的位置发送一个测试数据包，看看数据包来回需要多长时间。请试着用ping程序测试从你所在的位置到几个已知地点需要多长时间。利用这些数据，绘出Internet上的单向传输时间与距离的函数关系。最好使用大学作为目标，因为大学服务器的位置往往可以确切地知道。例如，berkeley.edu在加州的Berkeley；mit.edu在麻省的Cambridge；vu.nl在荷兰的阿姆斯特丹；www.usyd.edu.au在澳大利亚的悉尼；www.uct.ac.za在南非的Cape Town。
34. 到IETF的网站www.ietf.org，了解它们正在做什么。选择你喜欢的主题，写半页针对该问题的报告，并提出自己的解决方案。
35. Internet由大量的网络构成。这些网络的布局决定了Internet的拓扑结构。网上提供了大量有关Internet拓扑结构的可用信息。请用搜索引擎找出更多有关Internet拓扑结构的信息，并根据你的发现写一个简短的概述报告。
36. 搜索Internet，试找出当前在Internet上路由数据包的一些重要对等节点。
37. 写一个程序实现7层协议模型中从顶层到底层的报文流。针对每一层，程序应包括一个单独的协议函数。协议头为64个字符序列。每个协议函数有两个参数：从高层协议传递下来的报文（一个char缓冲区）和报文的大小。这个函数在报文前面贴上报文头，并在标准输出上打印出新的报文；然后调用较低层协议的协议函数。程序的输入是一个应用程序的报文（一个至多80字符的序列）。

第2章 物 理 层

本章我们将着眼于网络协议模型最底层的物理层，该层定义了比特作为信号在信道上发送时相关的电气、时序和其他接口。物理层是构建网络的基础。物理信道的不同特性决定了其传输性能的不同（比如，吞吐量、延迟和误码率），所以物理层是我们展开网络之旅的最好始发地。

让我们首先从数据传输的理论分析出发，探讨决定信道传输的自然局限。接着给出三类传输介质：有线（铜线和光纤）、无线（陆地无线电）和卫星。每种技术都有其自身独特的性质，而这将影响到采用这些传输技术的网络设计和性能。这部分内容为我们理解现代网络的关键传输技术提供了背景知识。

然后我们将讨论数字调制解调技术，主要解决如何把模拟信号转换成数字比特以及将数字比特还原成模拟信号。在此基础上，引入多路复用方案，探讨如何在同一个传输介质上同时进行多个会话而彼此不会干扰。

最后，我们将重点关注三个被广泛应用于计算机广域网的通信系统实例：（固定）电话系统、移动电话系统和有线电视系统。实际上这三个系统中的任何一个都很重要，因此我们将花费相当的篇幅分别进行介绍。

2.1 数据通信的理论基础

改变诸如电压或电流等某种物理特性的方法可用来在电线上传输信息。如果用一个以时间 t 为自变量的单值函数 f(t)来表示电压或电流的值，就可以对信号的行为进行建模，并用数学手段对其进行分析。本节的主要内容就是介绍文种分析。

2.1.1 傅里叶分析

19 世纪早期，法国数学家傅里叶（Jean-Baptiste Fourier）证明了：任何一个行为合理周期为 T 的函数 g(t)，都可以表示成用正弦函数和余弦函数组成的无穷级数：

$$g(t)=\frac{1}{2}c+\sum_{n=1}^{\infty}a_n\sin(2\pi nft)+\sum_{n=1}^{\infty}b_n\cos(2\pi nft) \qquad (2\text{-}1)$$

其中，f=1/T 是基本频率，a_n 和 b_n 是 n 次谐波（harmonics）的正弦振幅和余弦振幅，c 是常数。这种分解称为傅里叶级数（Fourier series）。利用傅里叶级数可以重构出函数，也就是说，如果已知周期 T，并且给定振幅，则用等式（2-1）进行求和可以得到时间的原始函数 g(t)。

对一个有限时间的数据信号（所有的数据信号都是时间有限的）的处理可以想象成一次又一次地重复着整个模式。即在 T 到 2T 之间的信号与 0 到 T 之间的信号完全一样，以

此类推。

对于任何给定的 g(t)，在等式（2-1）两边同时乘以 sin(2πkft)，然后再从 0 到 T 求积分，则可计算出振幅 a_n。因为：

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)dt = \begin{cases} 0 & \text{当 } k \neq n \\ T/2 & \text{当 } k = n \end{cases}$$

只留下了一项 a_n，b_n 项完全消失了。类似地，在等式（2-1）两边乘以 cos(2πkft)，然后再从 0 到 T 求积分，同样可以计算出 b_n。另外，只要直接在等式两边求积分，可以得到 c。执行这些操作的结果如下所示：

$$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)dt \quad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)dt \quad c = \frac{2}{T}\int_0^T g(t)dt$$

2.1.2 带宽有限的信号

上述数学分析与数据通信的关联在于实际信道对不同频率信号有不同的影响。我们考虑一个特殊的例子：传输 ASCII 字符 b。该字符被编码成一个 8 比特长的字节，发送的比特模式是 01100010。图 2-1（a）的左半部分显示了计算机传输该字符时的电压输出。对该信号进行傅里叶分析，可以得到以下的系数：

$$a_n = \frac{1}{\pi n}[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)]$$

$$b_n = \frac{1}{\pi n}[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)]$$

$$c = 3/4$$

最初几项的均方根振幅 $\sqrt{a_n^2 + b_n^2}$ 如图 2-1（a）右边所示。我们之所以对这些值感兴趣，是因为它们的平方与对应频率的传输能量成正比。

所有的传输设施在传输过程中都要损失一些能量。如果所有的傅里叶分量都等量地衰减，则结果信号将会在振幅上有所减小，但不会变形（即它将与图 2-1（a）有同样的方波形状）。不幸的是，所有的传输设施对于不同傅里叶分量的衰减程度并不相同，从而导致接收端收到的结果信号发生变形。一般情况下对导线而言，在 0 到某个频率 f_c 的这段范围内，振幅在传输过程中不会衰减（这里 f_c 可以用 Hz（赫兹）来度量），而在此截止频率 f_c 之上的所有频率的振幅都将有不同程度的减弱。这段在传输过程中振幅不会明显减弱的频率的宽度就称为**带宽**（bandwidth）。实际上，截止频率并没有那么尖锐，所以，通常引用的带宽是指从 0 到使得接收能量保留一半的那个频率位置。

带宽是传输介质的一种物理特性，通常取决于介质的构成、厚度和电线或者光纤的长度。滤波器一般可用来进一步限制信号的带宽。例如，802.11 无线信道允许使用的带宽约为 20 MHz，因此 802.11 无线装置将信号带宽过滤成这个大小。又比如，在传统（模拟）电视系统中，每个信道在线缆或者空中占用 6 MHz 的宽度。使用过滤技术可以使得多个信号共享一段给定范围内的频谱，从而改进系统的整体效率。但是，这意味着某些信号的频率范围不能从零开始了，不过这没关系。因为带宽是指通过的频率的宽度，其所承载的信息仅仅取决于这个频率的宽度而不是频率的起始位置和终止位置。一般将从 0 到某个最大频率的信号称为**基带**（baseband）信号，将被搬移并占用某个更大频率范围的信号称为**通**

带（passband）信号，通带信号与所有的无线传输情况一样。

现在我们来考虑，如果带宽很低，以至于只有几个最低频率才能被传输（也就是说只取等式（2-1）的前面几项作为函数的近似值），那么图2-1（a）的信号将会怎么样？图2-1（b）显示了原始信号经过一个只允许第一个谐波（即基频f）通过的信道后得到的结果信号。类似地，图2-1（c）～图2-1（e）分别显示了经过高带宽信道后得到的频谱和重构函数。对于数字传输来说，目标是接收到的信号具有足以重构出原始发送比特序列的精度。我们在图2-1（e）早已做到这点，因此使用更多的谐波来接收更为精确的比特显然是浪费的。

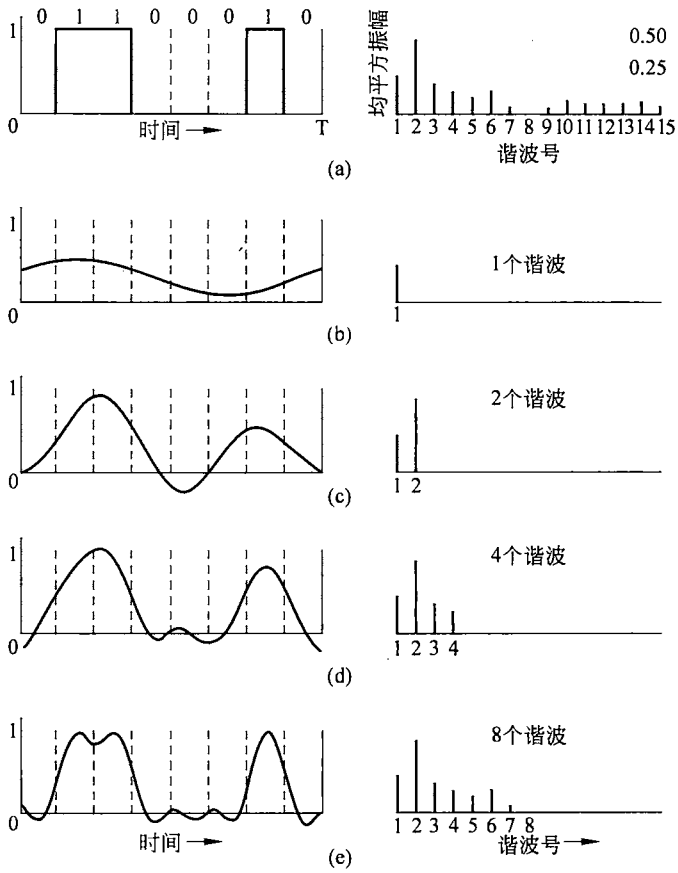


图 2-1

（a）一个二进制信号与它的平方根傅里叶振幅；（b）～（e）逐渐接近原始信号

假设在我们的例子中比特率为b比特/秒，发送8比特（一次发送一个比特）所需要的时间为8/b秒，因此信号的第一个谐波频率是（b/8）Hz。一条被人们称为语音级线路（voice-grade line）的普通电话线，人为引入的截止频率大约在3000 Hz以上。这个限制意味着在电话线上可以通过的最高谐波数大约是3000/（b/8）或者24 000/b（该截止频率不尖锐）。

对于一些数据传输速率，这些数字之间的关系如图2-2所示。这些数字清楚地表明，要想在语音级电话线上以9600 bps的速率发送数据，必须将图2-1（a）所示的信号变换成图2-1（c）所示的信号。如此精确地接收原始的二进制比特流需要更复杂的工作。应该看到，即使传输设备没有任何噪声，也不可能以高于38.4 kbps的数据速率传输任何二进制信号。换句话说，限制了带宽也就限制了数据传输率，即使对理想的信道也是如此。然而，

一些采用了几级电压值的编码模式可以获得更高的数据传输率。我们将在本章后面讨论这些方案。

bps	T(毫秒)	第一个谐波(Hz)	发生的谐波数
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19 200	0.422	400	1
38 400	0.21	4800	0

图 2-2 在我们例子中数据率与频率之间的关系

带宽（bandwidth）这个词很容易引起人们的混淆，因为在电气工程师和计算机科学家眼里它有不同的含义。对电气工程师来说，（模拟）带宽是以赫兹（Hz）来度量的（就像我们上面所描述的那样）；而对计算机科学家来说，（数字）带宽表示一个信道的最大数据速率，以每秒多少个比特（bps）来计量。数据速率是数字传输过程中采用一个物理信道的模拟带宽所能获得的最终结果，两者关系密不可分，我们下面马上讨论这点。阅读本书应该根据上下文来确定讨论的是模拟带宽（Hz）还是数字带宽（bps）。

2.1.3 信道的最大数据速率

早在 1924 年，AT&T 的工程师尼奎斯特（Henry Nyquist）就认识到即使一条理想的信道，其传输能力也是有限的。他推导出一个公式，用来表示一个有限带宽的无噪声信道的最大数据传输率。1948 年，香农（Claude Shannon）进一步把尼奎斯特的工作扩展到有随机噪声（热动力引起的）的信道的情形（Shannon，1948）。香农的文章是所有信息理论领域里最重要的文章。我们在这里简单地总结一下他们的经典结论。

尼奎斯特证明，如果一个任意信号通过了一个带宽为 B 的低通滤波器，那么只要进行每秒 2B 次（确切）采样，就可以完全重构出被过滤的信号。由于通过样值能恢复出来的高频成分已经被滤掉了，所以高于每秒 2B 次的采样毫无意义。如果信号包含了 V 个离散等级，则尼奎斯特的定理为：

$$\text{最大数据速率} = 2B \log_2 V \text{（比特/秒）} \qquad (2\text{-}2)$$

例如，无噪声的 3 kHz 信道不可能以超过 6000 bps 的速率传输二进制（即只有两级的）信号。

到现在为止，我们只考虑了无噪声信道。如果存在随机噪声，情况会急剧地恶化。并且，由于系统中分子的运动，随机（热）噪声总是存在的。热噪声的数量可以用信号功率与噪声功率的比值来度量，这样的比值称为**信噪比**（SNR，Signal-to-Noise Ratio）。如果我们将信号功率记作 S，噪声功率记作 N，则信噪比为 S/N。通常情况下为了适用很大的范围，该比率表示成对数形式 $10\log_{10}S/N$ ，对数的取值单位称为分贝（dB，decibel）。deci 意味着 10，而选择 bel 则是为了向发明了电话的贝尔（Alexander Graham Bell）致敬。10 的信噪比为 10 分贝，100 的信噪比为 20 分贝，1000 的信噪比为 30 分贝，等等。立体声放大器的制造商通常这样来形容它们的带宽（频率范围），即使在两端都存在 3 分贝的噪声

它们的产品在该频率范围内仍然是线性的。这恰好近似于放大因子的一半处（因为$10\log_{10}0.5 < -3$）。

香农的重大成果是：对于一条带宽为B Hz、噪声比是S/N的有噪声信道，其最大数据速率或者容量（capacity）是

$$\text{最大比特率}=B\ \log_2(1+S/N) \qquad (2\text{-}3)$$

这结论告诉了我们实际信道能获得的最大容量。例如，在普通电话线上提供访问Internet的ADSL（Asymmetric Digital Subscriber Line）使用了大约1 MHz的带宽。线路上信噪比的程度取决于住宅和电话交换局之间的距离，对于1～2千米的短距离来说40分贝的信噪比算是很好的状况了。正是因为电话线具有这样的特性，因此无论采用多少个信号等级，也不管采样频率多快或多慢，永远也不可能在该信道上获得高于13 Mbps的数据率。实际上，ADSL的最高速率为12 Mbps，虽然用户通常看到的速率要比这低得多。这个数据率实际上已经很好了，通信技术60多年的发展已经极大地缩小了香农容量与实际系统容量之间的沟壑。

香农结论取自信息理论，并适用于遭受热噪声的任何信道。任何反例都被可归纳到永动机一类。对于ADSL而言，要超过13 Mbps，必须改进信噪比（例如在靠近用户一端插入数字中继器）或者使用更大的带宽，就像已经演进到ADSL2+所做的那样。

2.2 引导性传输介质

物理层的作用是将比特从一台机器传输到另一台机器。实际传输所用的物理介质可以有多种选择。每一种传输介质都有独特的性质，体现在带宽、延迟、成本以及安装和维护难易程度上的不同，因此分别有自己适用的场合。大致上可以将介质分为引导性介质（也称为有线介质，比如铜线和光纤）和非引导性介质（也称为无线介质，比如地面无线电、卫星和激光）两大类。我们将在本节探讨引导性传输介质，在下节探讨非引导性传输介质。

2.2.1 磁介质

将数据从一台机器传输到另一台机器的最常见办法是将数据写到磁带或其他可擦写介质上（例如可刻录DVD），然后用物理的方法将磁带或者磁盘运送到目标机器，再将数据从磁带或磁盘里读出来。虽然这种方法不像使用地球同步通信卫星那么复杂，但它却更加有效，尤其适合于那些高带宽或者单个比特传输成本是关键因素的应用系统。

用一个简单的计算就能很容易看清楚这一点。工业标准的Ultrium磁带可以容纳800 GB。一个60厘米×60厘米×60厘米大小的盒子可以装下1000个这样的磁带，因此盒子的总容量为800 TB，或者6400 Tb（即6.4 Pb）。通过联邦快递或者其他快递公司，在24小时之内可以将这一盒磁带快递到美国的任何一个地方。这样一次传输的有效带宽是6400 Tb/86 400秒，或者70 Gbps。如果开车到目的地只需要一个小时，则有效带宽将超过1700 Gbps。到目前为止，还没有一个计算机网络能接近这样的传输能力。当然，网络速度正在变得越来越快，但磁带密度也同样在不断增长中。

如果我们现在来看一下成本，可以发现类似的情形。Ultrium 磁带批量购买的价格大约是每盘 40 美元。一盘磁带至少可以重复使用 10 次，所以一盒磁带每次使用的代价差不多是 4000 美元。加上额外 1000 美元的运输费用（可能还会更少一些），计算得出运送 800 TB 的成本大约是 5000 美元。这样算来，运输 1 GB 数据的费用还不到半个美分，这样的低成本还没有一个网络能与之抗衡。这个例子的寓意在于：

永远不要低估一辆满载着磁带在高速公路上飞驰的旅行车的带宽。

2.2.2 双绞线

尽管磁带具有优良的带宽特性，但其延迟特性却很差。传输时间以分钟或者小时来计，而不是以毫秒来计算，而这恰好是许多在线应用所需要的。一种最老但至今最常用的传输介质是**双绞线**（twisted pair）。双绞线由两根相互绝缘的铜线组成，铜线的直径通常大约为 1 毫米。两根铜线以螺旋状的形式紧紧地绞在一起，就像一个 DNA 分子链。之所以将两根铜线缠绕在一起，是因为两根平行的线会构成一个很好的天线。当两根线绞在一起后，不同电线产生的干扰波会相互抵消，从而能显著降低电线的辐射。信号通常以两根电线的电压差来承载，这样对外部噪声有更好的免疫力。因为噪声对两根电线的干扰是相同的，而它们的电压差却不会改变。

双绞线最常见的应用是电话系统。几乎所有的电话都是通过双绞线连接到电话公司的端局。打电话和 ADSL 接入 Internet 都发生在这些双绞线上。双绞线可以延伸几千米而不需要放大；如果距离很远，信号衰减得就很厉害，必须要使用中继器。当许多双绞线并行经过一定距离时，比如一座公寓楼内的所有的双绞线都连接到电话公司交换局时，那么应该把它们捆成一束，再外加一层保护套。这些被捆起来的双绞线如果不缠绕就会相互干扰。在有的地区，电话线沿地面上的电线杆走，常常可以看到直径为几厘米厚的电话线束。

双绞线既可以用于传输模拟信号，也可以用于传输数字信号。所获得的带宽取决于导线的厚度（即直径）以及传输距离的远近。许多情况下，双绞线传输几千米的距离可以达到几 Mbps 的带宽。由于双绞线具有足够的传输性能以及相对较低的成本，所以它的应用非常广泛，并且可能还会在未来持续很多年。

双绞线可以分成几大类。部署在许多办公大楼内的称为 5 类线（Category 5）或“猫 5”（Cat 5）。5 类双绞线由两根绝缘导线轻轻地扭在一起，4 对这样的双绞线被套在一个塑料保护套内。塑料外套既保护了双绞线又把多根导线捆在一起，如图 2-3 所示。

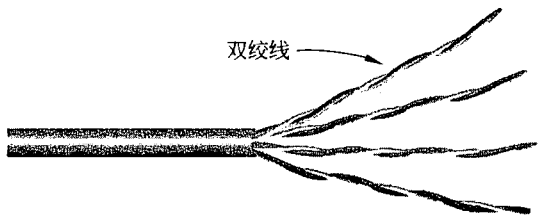


图 2-3 有四对双绞线的 5 类 UTP

不同的局域网标准或许使用不同的双绞线。例如，100 Mbps 以太网使用了（4 对之中

的）两对双绞线，分别用于两个方向上的传输。为了达到更高的速率，1 Gbps 以太网在双向传输中同时使用了全部的 4 对线。这就要求接收器能分解出本地传输的信号。

现在我们给出一些通用的术语。可以双向同时使用的链路称为**全双工**（full-deplex）链路，就像双车道一样；相对应地，可以双向使用但一次只能使用一个方向的链路称为**半双工链路**（half-duplex），就像单轨铁路线；第三类，只允许一个方向上传输的链路则称为**单工链路**（simplex），就像单行街一样。

再回到双绞线，“猫 5”取代了早期的 3 类线，虽然与 3 类线具有相同的连接器，但每米内的双绞线扭得更紧了。单位长度内缠绕得更紧可以导致更少的串扰，而且在长距离传输过程中还能使信号质量更好，这就使得“猫 5”更加适用于高速计算机通信，尤其是 100 Mbps 和 1 Gbps 以太局域网。

新双绞线很有可能是 6 类甚至是 7 类线。这些类别的双绞线具有更严格的规范来处理高带宽信号。某些 6 类和更高类的双绞线信号速率高于 500 MHz，可以支撑 10 Gbps 的链路。这些双绞线将很快得到部署。

到 6 类为止，所有的双绞线都称为**非屏蔽双绞线**（UTP，Unshielded Twisted Pair），这些双绞线仅由导线和绝缘层简单地构成。相对应地，7 类双绞线在每对双绞线外面加了一个屏蔽层，然后在整个线缆外面再加一个屏蔽层（内层的是塑料保护套）。屏蔽层能够减弱外部干扰和来自附近线缆的串扰，从而满足苛刻的性能规范要求。该类双绞线令人回想起 IBM 在 20 世纪 80 年代早期引入的屏蔽双绞线，当时的双绞线既笨拙价格又昂贵，除 IBM 自用外没有引起业界广泛的兴趣。显然，现在到了该再次试试看的时候了。

2.2.3 同轴电缆

另一类常用的传输介质是**同轴电缆**（coaxial cable），相对于许多其他的传输介质，常常简称为“coax”，发音为“co-ax”。它比非屏蔽双绞线有更好的屏蔽特性和更大的带宽，所以它能以很高的速率传输相当长的距离。广泛使用的同轴电缆有两种。一种是 50 Ω 电缆，从一开始它就被用于数字传输；另一种是 75 Ω 电缆，一般用于模拟传输和有线电视传输。这种划分是基于历史的原因，而并非技术因素（例如，早期的偶极天线阻抗为 300 Ω 阻抗，所以，很容易利用现有的 4∶1 阻抗匹配变压器）。从 20 世纪 90 年代中期开始，有线电视运营商开始在有线电缆上提供访问 Internet 的业务，这样 75 Ω 的同轴电缆相对数据通信来说显得更为重要。

同轴电缆由硬的铜芯和外面包上的一层绝缘材料组成。绝缘材料的外面是一层密织的网状圆柱导体，外层导体再覆盖上一层保护塑料外套。同轴电缆的剖面图如图 2-4 所示。

同轴电缆的结构和屏蔽性使得它既有很高的带宽，又拥有很好的抗噪性。带宽可能取决于电缆的质量和长度。现代电缆能达到几个 GHz 的带宽。过去，同轴电缆被广泛应用于长途电话系统，但现在大部分长途干线已经被光纤所取代。即便如此，同轴电缆仍然是有线电视和计算机城域网的常用传输介质。

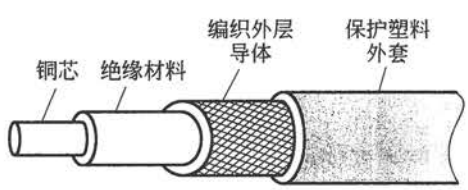


图 2-4 同轴电缆示意图

2.2.4 电力线

电话网络和有线电视网络并不是可重复用于数据通信的唯一有线资源。实际上还存在着一种甚至更为普遍的导线：电力线。电力线把电能传送到千家万户，室内的电线又把电能分布到每个电源插座。

将电力线用于数据通信的想法并不新鲜。电力公司多年前就已经利用电力线进行较低速率的通信（比如远程测量），智能家居用电力线对室内电子电器设备进行控制（译注：比如 X10 标准——这是 1975 年由 Pico Electronics 公司研发的国际通用智能家居电力载波协议）。近几年旧话重提，出现了对用电力线进行高速率通信的新的兴趣。这里的高速率通信既可以用在家庭内部构建局域网，又可以用在室外作为访问 Internet 的宽带接入。我们把注意力集中在更普遍的使用场景——家庭室内使用电力线。

用电力线组建网络的便利性不言而喻。只需简单地把电视机和接收器插入墙上的电源插座（这一步你是无论如何都不能省，因为电器工作需要电能），它们就可以通过电线发送和接收电影了。这种配置如图 2-5 所示，此时不再需要任何插头或者无线电。当两种信号同时使用电线时，数据信号叠加在低频电力信号上（在主动的或“热”电线上）。

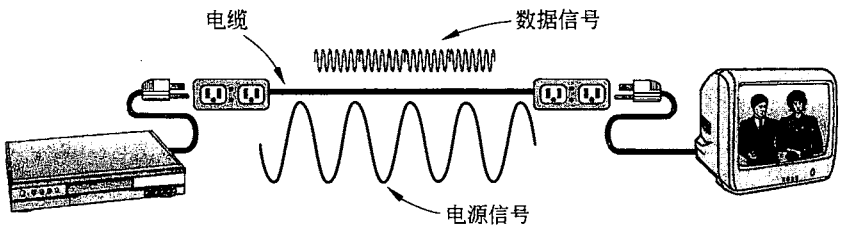


图 2-5 使用家庭电气布线的网络

家庭网络使用电力线的困难在于电线是专为分发电源信号而设计的。分发电能与分发数据信号是两项完全不同的工作，因此家庭布线是一项很可怕的工作。电信号以 50～60 Hz 频率发送，高速率数据通信所需的更高频率（MHz）在电线上会产生严重的衰减。而且电线的电气特性并不是户户相同，而是随着各家电器设备使用时电源的开/关不同而变化莫测。电器设备开/关时的瞬态电流将在很宽的频率范围内造成电噪声。如果双绞线没有小心地扭在一起，电力线就将成为很好的天线，吸收外部信号并辐射到自身信号。这种行为意味着为了满足监管要求，数据信号必须把许可频率排出在外，例如业务无线电爱好频段。

尽管存在许多困难，只要采用的通信方案具有抗频率损耗和突发错误的能力，在典型的居家电力线上达到不小于 100 Mbps 的发送还是切实可行的。许多用电力线组网的产品采用了各自专用的标准，因此国际标准正在积极地制定中。

2.2.5 光纤

令计算机工业界很多人引以为豪的是计算机技术如此快速的发展，它遵循摩尔关于每块芯片上晶体管数量大约每两年翻一番的预测定律（Schaller，1997）。最初的 IBM PC 以 4.77 MHz 的时钟速度运行（1981 年），28 年后的 PC 则可在 4 个内核上以 3 GHz 的速度运

行。增长因子大约为2500倍，或每10年16倍。的确令人印象深刻。

在计算机技术发展的同时，广域数据通信链路从45 Mbps（电话系统中的T3线路）发展到100 Gbps（现代长途线路）。这样的增长速度同样令人兴奋。数据速率增长了大约2000多倍，几乎每10年增长了16倍还多；但同时误码率却从每比特10^{-5}下降到几乎为零。而且，单个CPU的能力正在接近其物理极限，这就是为何现在每个芯片上CPU的数量不断在增加。相比之下，光纤技术的可达带宽超过50 000 Gbps（50 Tbps），我们现在还远未达到这个极限。当前实际带宽只能达到约100 Gbps，之所以无法获得更高带宽的限制，在于我们无法把电气和光学信号之间的转换进行的更快。为了建设高容量的链路，可在单条光纤并发地使用多个信道来传送信号。

在本节中，我们将了解光纤的传输技术是如何工作的。计算和通信两者之间正在进行着比赛，正是因为有了光纤网络，通信很有可能会赢。这种现状隐含着本质上无限的网络带宽和一个新的共识，即计算机（在大量计算需求的面前）将会显得无可救药地慢，以至于网络应不惜一切代价尽量避免计算，而不管有多少带宽被浪费。这种认识上的变化需要很长时间才能被一代计算机科学家和工程师所理解，他们所学到的是香农针对铜芯介质给出的低带宽限制。

当然，这种情况并不说明全部，因为这里没有考虑成本。在“最后一英里”安装光纤到达每个用户，并且绕过电话线的低带宽和频谱的有限可用性所带来的成本是巨大的。而且比特移动所花费的能量比计算所需要的能量更多。但现实中我们总会见到一些不平等的地方，即无论是计算还是通信基本上是免费的。例如，在Internet边缘，我们把计算和存储问题集中在内容的压缩和缓存上，所做的这一切都是为了更好地使用Internet接入链路；而在Internet内部，我们做的恰好相反，比如谷歌这样的IT公司把大量数据通过网络移动到储存或计算更便宜的地方。

光纤主要用于网络骨干的长途传输、高速局域网（虽然到目前为止，铜芯仍然在设法追赶光纤）以及高速Internet接入，比如光纤到户（FttH，Fiber to the Home）。光纤传输系统由三个关键部件构成：光源、传输介质和探测器。按照惯例，一个光脉冲表示比特1，没有光脉冲表示比特0。传输介质是超薄玻璃纤维。光探测器探测到光时产生一个电脉冲。在光纤两端分别接上光源和探测器，我们就有了一个单向数据传输系统。该系统接收电子信号，将其转换成光脉冲并传输出去，然后在另一端把光脉冲转换回电子信号输出给接收端。

这种传输系统会漏光，在实际中并没有什么用处，而且它不是一种有趣的物理学原理。当一束光线从一种介质到达另一种介质的时候，譬如从二氧化硅到空气中，在二氧化硅和空气的边界上，光线会发生折射（弯曲），如图2-6（a）所示。这里我们看到一束光入射在界面上的角度是α_1，折射出来的时候角度为β_1。折射的角度取决于两种介质的特性（尤其是它们的折射率）。如果入射角度超过了某一个特定的临界值，则光就会被反射回二氧化硅，不会再有光漏到空气中。因此，入射角度大于等于临界值的光将被限定在光纤内部，如图2-6（b）所示，它可以传播好几千米而事实上没有损失。

图2-6（b）只显示了一束截留光。但是，由于任何入射角度大于临界值的光束都会在内部反射，所以许多不同的光束以不同的角度来回反射着向前传播。可以说每一束光都有不同的模式，所以，一根具有这种特性的光纤称为**多模光纤**（multimode fiber）。

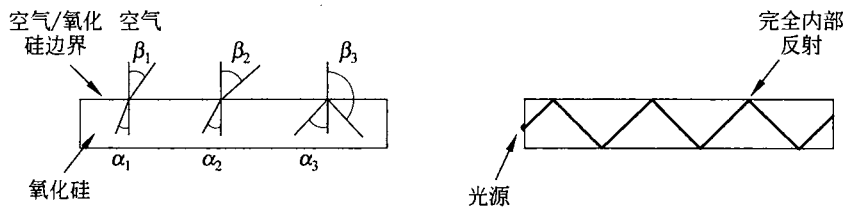


图 2-6

（a）氧化硅内部的光以不同角度射到空气/氧化硅边界；（b）完全内部反射的光

然而，如果光纤的直径减小到只有几个光波波长大小的时候，则光纤就如同一个波导，光只能按直线传播而不会反射，由此形成了**单模光纤**（single-mode fiber）。单模光纤比较昂贵，广泛应用于长距离传输。目前可用的单模光纤可以 100 Gbps 的速率传输数据到 100 千米远而不用放大器。而在实验室内，短距离传输时还能得到更高的数据传输率。

光纤传输光

光纤由玻璃制成，玻璃又是由沙子做成的，而沙子这种原始廉价的材料取之不尽用之不竭。玻璃的制造可以追溯到古埃及，但那个时候的玻璃厚度不能超过 1 毫米，否则光就透不过来。足够透明到能用于窗户的玻璃是在文艺复兴时期发明的。现代光纤所用的玻璃异常透明，如果把海洋中的海水全部替换成这样的玻璃，那么你可以从海面一直看到海底，就好像晴天在飞机上往下看地面一样清楚。

光通过玻璃的衰减取决于光的波长（以及玻璃的某些物理特性）。光的衰减定义为输入输出信号功率的比值。对于光纤中所使用的玻璃而言，这种衰减如图 2-7 所示，图中显示了光纤每千米衰减的分贝。例如，如果损失了一半的能量，则衰减为 $10\log_{10}2 = 3$ 分贝。该图显示了频谱中接近红外的那部分，也是实际中所用的部分。可见光的波长要稍微短一些，从 0.4 微米到 0.7 微米（1 微米等于 10^{-6} 米）。真正的度量纯粹主义者更喜欢将波长写成 400～700 纳米，但我们会坚持传统的使用方式。

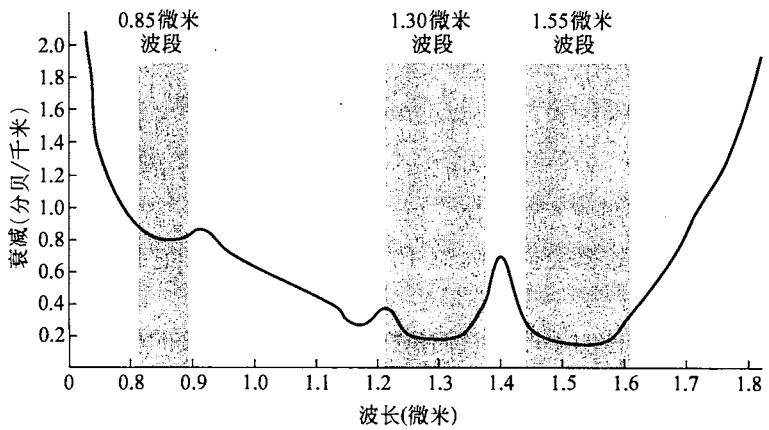


图 2-7 红外区域的光通过光纤时的衰减情况

现在光纤通信中最常用的三个波段，它们分别集中在 0.85 微米、1.30 微米和 1.55 微米处。所有三个波段都具有 25 000～30 000 GHz 的宽度。0.85 微米的波段首先被使用，它的

衰减比较大，只能用于短程通信；但在这个波段上，我们可以从相同的材料（砷化镓）获得激光和电子。后两个波段有很好的衰减特性（每千米的损失小于5%）。1.55微米波段目前被广泛用于掺铒放大器，它直接工作在光域。

光脉冲沿光纤传播时会散开，这就是所谓的色散传播（chromatic dispersion）。散开的数量与波长有关。防止这些散开脉冲发生重叠的一种办法是加大它们之间的距离，但只有降低了信号速率才能做到这一点。幸运的是，现在已经发现，通过将脉冲做成一种特殊的形状（该形状与双曲余弦的倒数有关），几乎所有的色散效应就都不再存在，因而有可能将光脉冲发送几千千米而不会有明显的波形失真。这些脉冲称为孤波（soliton）。为了使孤波尽快走出实验室进入实用领域，大量的研究工作正在进行之中。

光缆

光缆和同轴电缆非常相似，只不过光缆没有那一层密织的网。图2-8（a）显示了一根光纤从侧面看到的内部结构。中间是玻璃芯，光脉冲通过它传播。在多模光纤中，玻璃芯的直径通常是50微米，相当于一根人头发丝的粗细。在单模光纤中，玻璃芯的直径为8～10微米。

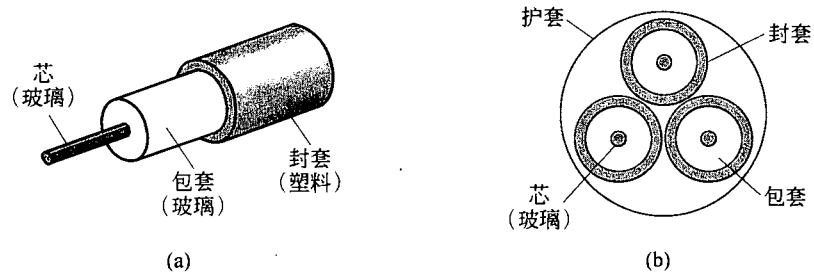


图 2-8
（a）单根光纤的侧面图；（b）三根光纤的横截面视图

玻璃芯的外面是一个玻璃覆盖层，覆盖层的折射率比玻璃芯低，这样可以保证所有的光都限制在玻璃芯内。在这外面是一层薄薄的塑料封套，用来保护里边的玻璃包层。光纤通常被捆扎成束，最外面再加一层保护套。图2-8（b）显示了一个内含三根光纤的截面图。

陆地上的光纤通常被埋在地表表面1米以下，有时候它们偶尔会被农用工具或者地老鼠破坏。在靠近岸边的地方，越洋光纤通过一种称为海犁的工具被埋在电缆沟里。在深水中，它们直接被放入海底，有可能被渔船撞坏或者被大一点的鱼咬坏。

光纤可以按照三种不同的方式连接。第一种方式，用连接器终止一根光纤，然后再把它插入到光纤插座中。连接器会损失大约10%～20%的光，但它使系统的重新配置工作做起来比较容易。

第二种方式，通过机械的手段把它们拼接起来。机械拼接的做法是将两根小心切割好的光纤头放在一个特殊的套管中，然后将它们适当夹紧。接口对齐可改善通过拼接处的光，适当进行调整使信号尽可能达到最大，由此来提高拼接处的光质量。对于受过训练的专业人士来说，机械拼接过程大约需要5分钟，并且会有10%的光损失。

第三种方式，把两根光纤融合（熔合）在一起形成一个非常结实的连接。融合后的光纤几乎与单根光纤的性能一样好，但尽管如此，少量的衰减仍然存在。

对于所有这三种拼接手段，在接合点上都可能会发生光的反射，并且反射的能量可能会干扰原来的信号。

通常用作信号源的有两种光源，它们是发光二极管（LED，Light Emitting Diodes）和半导体激光。这两种光源的特性不同，如图2-9所示。通过在光源和光纤之间插入Fabry-Perot或者Mach-Zehnder干涉仪，可以对波长进行调节。Fabry-Perot干涉仪是一个由两面平行镜子构成的简单共振腔。光线垂直于镜面入射，共振腔的长度会筛选出那些波长为其整数倍的波。Mach-Zehnder干涉仪将光分为两束，这两束光经过的距离稍有不同。它们在终端处重新整合在一起，于是只有特定波长的波才能同相。

项目	LED	半导体激光
数据率	低	高
光纤类型	多模	多/单模
距离	短	长
寿命	长寿	短命
温度敏感性	不敏感	敏感
成本	廉价	昂贵

图2-9 半导体光源和LED光源的比较

光纤的接收端是一个光电二极管。当遇到光照时，光电二极管就发出一个电脉冲。光电二极管的响应时间，即把光信号转换成电域所需要的时间，限制了数据传输率在100 Gbps左右。热噪声也是一个问题，所以光脉冲必须保证具有足够的能量才能够被接收端探测到。如果脉冲的能量足够强，错误率就可以被降低到任意小。

光纤与铜线的比较

把光纤和铜线作个对比很有意义。光纤有很多优点。首先，光纤比铜线能够处理更高的带宽，这使得它非常适用于高端网络。由于光纤具有相对较低的衰减，所以在较长的线路上，大约每50千米才需要一个中继器；而对于铜线而言，大约每5千米就需要一个中继器。很显然，这节约了网络的很大成本。另外，光纤还具有不受电源浪涌、电磁干扰或电源故障等影响的优点。而且它也不受空气中腐蚀性化学物质侵蚀的影响，因此它能够被应用于恶劣的工业环境。

奇怪的是，电话公司喜欢光纤却是因为别的理由：光纤细小而且重量较轻。许多现有的电缆管道都已经被塞得满满，几乎没有富余的空间再增加新的容量。将所有原来的铜线清除掉，替换上光纤，可以腾出电缆管道的空间，然后把铜线转卖给那些炼铜商，他们将铜线看作一种高级的矿物原材料。而且，光纤比铜轻得多。1000根1千米长的双绞线重约8000千克，两根光纤的容量就能超过这1000根双绞线的容量，但它们的重量却只有100千克。所以，使用光纤可以极大地降低对于管线机械支撑系统的需求，而建立和维护这种机械系统的费用非常昂贵。对于新的线路来说，光纤比铜线更有优势，因为它的安装费用相对要低得多。最后，光纤不会漏光，而且不易被接入，这些特性使得光纤很难被搭线窃听，因而具有很高的安全性。

然而，从另外一方面来看，光纤也有不足。光纤是一项相对陌生的技术，要求较高的操作技能，而且这种技能并不是每一个工程师都具备；当光纤被过度弯曲时容易折断。由于光传输技术本质上是单向的，所以双向通信要求使用两根光纤，或者在一根光纤上划分

两个频段。最后，光纤接口的成本远远高于电子接口的成本。不过，将来更多小范围内的所有固定数据通信显然都应该使用光纤。有关光纤和光纤网络的讨论，请参考（Hecht，2005）。

2.3 无线传输

我们这个时代已经产生了那些需要随时都在线的信息瘾君子。对这些移动用户来说，双绞线、同轴电缆和光纤都毫无意义。他们希望从便携式电脑、笔记本电脑、衬衣口袋、掌上电脑或腕表电脑上获取他们“点中”的数据，而不能被拴在某种地面通信基础设施上。对这些用户来说无线通信就是答案。

在下面的章节中，我们将着眼于一般的无线通信。无线通信除了为用户提供进行 Web 冲浪的连接外还有许多其他重要的应用。在某些情况下 无线具有的优势甚至超过了固定设备的优势。例如，由于地形（山区、丛林、沼泽等）等陆地原因造成把光纤拉到一座建筑物非常困难时，无线或许是更好地选择。值得注意的是，现代无线数字通信始于夏威夷群岛。在那里用户和他们的计算机中心被大块的太平洋所隔离，而且电话系统也不够完善。

2.3.1 电磁频谱

当电子运动时会产生电磁波，电磁波可在空中传播（即使在真空中）。英国物理学家马克斯韦尔（James Clerk Maxwell）在 1865 年就预言了这种波的存在，但直到 1887 年才第一次被德国物理学家赫兹（Heinrich Hertz）观测到。电磁波每秒振动的次数称为它的**频率**（frequency），通常用 f 表示，以赫兹（Hz）来度量（以此纪念物理学家赫兹）。两个相邻的波峰（或者波谷）之间的距离称为**波长**（wavelength），通常用希腊字母 λ（lambda）表示。

当一个大小适中的天线被连接到一个电路上，电磁波就可以有效地被广播出去，在一定距离内的接收者能收到该电磁波。所有的无线通信都是基于这样的原理完成的。

在真空中，所有的电磁波按同样的速度传播，跟它们的频率无关。这个速度通常称为光速，用 c 表示，它近似等于 3×10^8 m/s，或者每纳秒大约 1 英尺（30 厘米）（有个例子关于重新定义英尺单位：1 英尺等于光在真空中 1 纳秒时间内经过的距离，而不是根据某个远古时期国王鞋子的大小而做出的度量）。在铜线或者光纤中，电磁波的速度会慢一些，大约是光速的 2/3，而且跟频率有关。光速是速度的极限，没有物体或者信号可以运动得比光速还快。

f、λ 和 c（真空中）之间的基本关系是：

$$\lambda f = c \qquad (2\text{-}4)$$

由于 c 是常数，如果我们知道 f，就可以算出 λ，反之亦然。一条经验规则是，如果 λ 的单位是米，f 的单位是 MHz，则 $\lambda f < 300$。例如，100 MHz 的波长大约为 3 米，1000 MHz 的波长大约是 0.3 米，0.1 米波长的频率为 3000 MHz。

电磁波谱如图 2-10 所示。波谱中的无线电波、微波、红外光和可见光都可以通过调制波的振幅、频率或者相位来传输信息。紫外线、X 射线和 γ 射线用来传输信息的效果可能

会更好，因为它们的频率更高。但是这种波很难产生和调制，其穿透建筑物的能力也不好，而且对生物有害。图 2-10 列出的波段是官方 ITU（国际电信联盟）依据波长给出的命名，所以低频（LF）波段为 1～10 公里（30～300 kHz）。术语 LF、MF 和 HF 分别指低频、中频和高频。很显然，当初命名时没有人期望会超越 10 MHz 以上的频段，因此，这些高频波段后来被命名为甚高频（Very）、超高频（Ultra）、特高频（Super）、极高频（Extremely）和巨高频（Tremendously High Frequency），相应的缩写分别为 VHF、UHF、SHF、EHF 和 THF。再往上就没有名字了，但难以置信的（incredibly）、惊人的（astonishingly）和非凡的（prodigiously）高频段（IHF、AHF 和 PHF）听起来也不错。

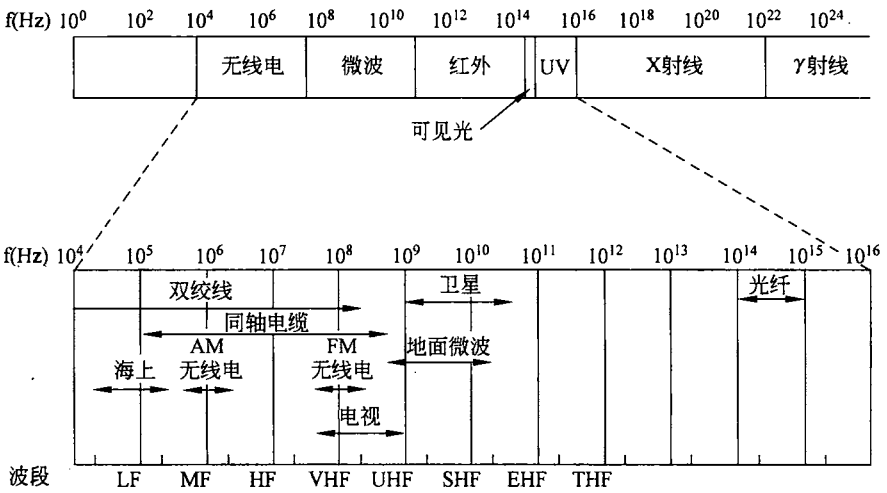


图 2-10 电磁频谱以及在通信中的使用

从香农定理（等式（2-3））我们知道诸如一个电磁波的信号能携带的信息量取决于接收能量，并且与带宽成正比。从图 2-10 可以清楚地看出为何网络用户多么喜欢光纤。在微波频段有许多 GHz 的带宽可用于数据传输，而光纤波段甚至有更多 GHz 可用，因为这些带宽进一步加大了等式右边的对数尺度。作为一个例子，我们考虑图 2-7 中的 1.3 微米波段，它宽为 0.17 微米。如果运用等式（2-4）找出该波开始和结束处的起始频率和终止频率，我们就会发现该频率范围将达到 30 000 GHz。在 10 dB 这样合理的信噪比环境里，数据率高达 300 Tbps。

大多数信息传输都使用相对窄的频段（即 $\Delta f/f<<1$）。这些技术重点关注窄频内信号的频谱使用效率以及用多大能量的传输来获得合理的数据率。然而，在有些情况下，也会使用较宽的频段，使用方式一共有三种。第一种是**跳频扩频**（frequency hopping spread spectrum），发射器以每秒几百次的速率从一个频率跳到另一个频率。这种技术在军事通信中很流行，它使得我方的通信过程很难被敌方检测到，因而敌方也就无法干扰到我方通信。而且，它对于多径衰减和窄频干扰现象也有很好的抵抗能力，因为接收器不会在一个受损频率上停留足够长时间而导致通信中断。这种通信方式的鲁棒性对于频谱中的拥挤那部分非常有用，例如我们即将描述的 ISM 频段。跳频技术早已被应用到商业领域，例如蓝牙和旧版的 802.11。

有意思的是，这项技术是由出身在奥地利的性感女神 Hedy Lamarr 参与合作发明的。

她是第一个裸体出演电影的女明星（1933年的捷克电影《Extase》），其第一任丈夫是武器制造商。他告诉她如何阻塞无线电信号，然后用无线电信号来控制鱼雷是如何容易。当她发现自己的丈夫卖武器给希特勒时非常害怕，于是便乔装成女仆逃离了丈夫，飞到好莱坞继续她的演艺事业。在闲暇之余，她发明了跳频技术来帮助盟军。她的跳频方案用到了88个频率，这个数字正好是钢琴的琴键数（频率）。她们的这项发明（她和她的作曲家朋友George Antheil）获得了美国专利2 292 387。然而，她们没能说服当时的美国海军相信这项发明有任何实际用途，所以他们没有得到任何版税。在专利过期之后很多年，这项技术才变得流行起来。

第二种扩展频谱的方式是**直接序列扩频**（direct sequence spread spectrum）。这种方法使用了一个码片序列，并且将数据信号展开到一个很宽的频段上。这是一项能使多个信号共享同一频段的频谱效率方法，现已被广泛应用于商业领域。信号被赋予不同的码片，这种称为**码分多址**（CDMA，Code Division Multiple Access）的方法我们将在本章后面再讨论。图2-11给出了这种方法与跳频方法的对比。CDMA构成了3G移动电话网络的基础，还被用于全球定位系统中（GPS，Global Positioning System）。即使没有不同的码片，像跳频这样的直接序列扩频也可以容忍窄带干扰和多径衰落，因为丢失的只是所需信号中的一小部分。我们可以从802.11b无线局域网的老版本中看到这种通信方式的作用。对于扩频通信的神奇作用和详细历史，请参考（Scholtz，1982）。

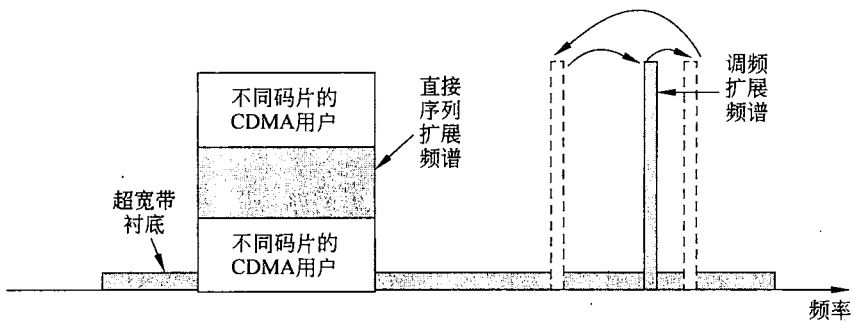


图2-11 扩展频谱和超宽带（UWB）通信

第三种宽带通信方式是超宽带通信（UWB，Ultra-WideBand）。UWB发送一系列快速脉冲，这些脉冲随着通信信息而不断变化自己的位置。这种位置的快速变换导致信号被稀疏分布在一个很宽的频带上。UWB的定义是那些至少有500 MHz带宽或中心频率至少占频率波段20%的信号。图2-11中还显示了超宽带。有了这么多的带宽，UWB就拥有了高速率通信的潜力。因为它的信号被分散在很宽的频率波段上，所以它可以容忍大量来自其他窄带信号相对来说比较强的干扰。同样重要的是当UWB被用在短距离传输时，在任何给定的频率上只需要非常少的能量，因此它不会对其他窄带无线电信号产生有害干扰。UWB称为是其他信号的衬底（underlay）。UWB和其他信号的这种和平共处使得它在无线个域网（PAN）上的应用运行速率高达1 Gbps，尽管商业上的成功时好时坏。它也可以用于通过固体物体成像（地面、墙壁和身体）或作为精确定位系统的一部分。

我们现在从无线电开始讨论如何使用图2-11所示电磁频谱中的不同部分。除非特别声明，我们假设所有的传输都使用窄带频段。

2.3.2 无线电传输

无线电频率（RF，Radio Frequency）的波形很容易产生。它可以传输很长的距离，并且很容易穿透建筑物。所以无线电波被广泛应用于通信领域，无论是室内通信还是室外通信。无线电波是全方向传播的，这意味着它们从信号源沿着所有的方向传播出去，因此发射设备和接收设备不需要在物理上小心地对齐。

有的时候全向无线电是好事，但某些时候也会坏事。20 世纪 70 年代，通用汽车公司决定在所有新型号的卡迪拉克汽车上安装由计算机控制的防抱死刹车。当驾驶员踩刹车踏板时，计算机将刹车时闭时开，而不是一下将刹车锁死。有天阳光普照，一个俄亥俄州的高速公路巡警开始用他那新的移动无线电给总部打电话，突然他车旁的凯迪拉克车像匹咆哮的野马一样反应失常。当巡警将他拦下时，车主抱怨说他根本什么都没有做，那车子自己突然发疯了。

最终人们渐渐地发现了规律：卡迪拉克有时会发狂，但只发生在俄亥俄州的主要公路，并且只有当公路巡警在巡视时才有可能会发疯。有很长一段时间，通用汽车公司都无法理解为什么这款汽车在所有其他州都工作良好，甚至在俄亥俄州的非主要公路上也没有出现问题。在经过了大量的调查研究后，他们才发现卡迪拉克的线路系统恰好构成了俄亥俄州高速公路巡警新无线电系统频率的一根极好的天线。所以当公路巡警使用无线电时严重干扰了汽车本身的控制线路。

无线电波的特性与频率有关。在低频部分，无线电波能够很好地穿透障碍物；但是随着离信号源越来越远，其能量急剧下降。因为信号能量稀疏地分布在大面积的表面，在空中传播时的信号能量至少以 $1/r^2$ 的速度衰减（译注：r 是离信号源端的距离）。这种衰减称为**路径损耗**（path loss）。在高频部分，无线电波倾向于以直线传播，并且遇到障碍物会反弹回来。虽然接收到的信号很大程度上取决于信号的反射，路径损耗依然降低了能量。相比低频无线电波，高频无线电波更容易被雨水和其他障碍物吸收。在所有频率上，无线电波都会受到马达和其他电气设备的干扰。

把无线电波的衰减和引导性介质上的信号衰减做个比较很有意思。光纤、同轴电缆和双绞线上的信号在单位距离下降的能量比例相同，例如信号沿双绞线传播每 100 米能量下降 20 分贝。而在无线电中，信号下降的速度与距离的平方成反比，例如在自由空间中距离每增加一倍信号下降 6 分贝。这种行为意味着无线电波可以传播很长距离，但用户之间的干扰是个问题。出于这个原因，各国政府严格管制无线电发射器的使用，我们将在本章后面讨论少数几个特例。

在 VLF、LF 和 MF 波段，无线电波沿着地面传播，如图 2-12（a）所示。在较低频率上，这些电波可以在 1000 千米外被检测到，如果频率高些则能检测到电波的距离范围要小一些。调幅（AM）无线电广播使用了 MF 波段，这就是为什么波士顿调幅广播电台发出的地面波在纽约听不到的原因。这些波段中的无线电波很容易穿透建筑物，这也是为什么便携式收音机可以在室内使用的原因。使用这些波段进行数据通信的主要问题在于它们的带宽太低（参见等式（2-4））。

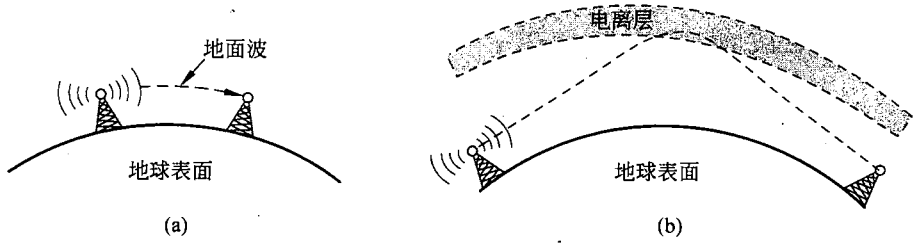


图 2-12
（a）VLF、LF 和 MF 波段的无线电波沿着地球曲面传播；（b）HF 波段的无线电波被电离层反弹回来

在 HF 和 VHF 波段，地面波会被地球表面吸收。然而，当无线电波到达电离层——围绕着地球、位于地球上方 100～500 千米高空处的带电粒子层时，电磁波就被电离层折射回地球，如图 2-12（b）所示。在某些特定的大气条件下，信号可以被反弹多次。业余无线电爱好者可以使用这些波段进行长距离通话。军队也使用 HF 和 VHF 波段进行通信。

2.3.3 微波传输

在 100 MHz 以上频段，电磁波几乎按直线传播，因此它们可以被聚集成窄窄的一束。通过抛物线形状的天线（就像常见的碟形卫星电视天线），可以把所有的能量集中于一小束，从而获得极高的信噪比，但是要求发射端和接收端的天线必须精准地相互对齐。而且，这种定向传播允许多个排成一行的发射器与多个排成一行的接收器同时进行通信；只要它们的空间排布满足最小间距规则，相互之间就不会干扰。在光纤出现之前的几十年中，这种微波构成了长途电话传输系统的核心。实际上，在撤销管制之后，AT&T 的第一批竞争者之一 MCI 公司就使用微波通信技术建立了它的整个通话系统。该系统包括很多微波塔，塔与塔之间相距几十千米。MCI 至今已改成使用光纤，经过一系列的公司兼并和电信系统洗牌过程中的破产保护 MCI 已成为 Verizon 的一部分（译注：Verizon 是美国最大运营商）。

微波按直线传播。如果两个微波塔相距太远，那么地球本身就会阻挡传输路径（想象一下从旧金山到阿姆斯特丹的链路）。因此，每隔一段距离就需要一个中继器。微波塔越高，微波传的距离就越远。中继器之间的距离大致与塔高的平方根成正比。即对于高度为 100 米的微波塔，两个中继器之间的距离可以为 80 千米。

与低频无线电波不同的是，微波不能很好地穿透建筑物。而且，即使微波在发射器已经聚集成束，它在空中传播时仍然会有一些发散。有些微波会被过低的大气层折射回来，比直接波传得更远一些。延迟抵达的微波与直接传输的微波可能不同相，因而信号会相互抵消。这种传播效果称为**多径衰落**（multipath fading），在无线传输中这通常是一个很严重的问题。多径衰落与天气和频率有关。有些运营商将 10%的信道保持为空闲，当多径衰减现象使得某些频段临时失效时立即切换到这空闲频段继续工作。

对于频段越来越多的需求驱使运营商们开始向高频发展。现在使用的频段已经扩展到了 10 GHz，但是在 4 GHz 左右，出现了一个新的问题：微波被水吸收了。这些微波只有几个厘米长，会被雨水吸收。如果有人打算在室外建造一个巨大的微波炉用来烧烤飞过的小鸟，这种吸收效果倒是不错。但是对于通信来说，这是个很严重的问题。与多径衰落现象一样，唯一的解决方案是停止使用这些受雨水影响的链路，并避开这些链路。

总而言之，微波通信已经被广泛应用于长途电话通信、移动电话和电视转播，其他频谱严重短缺领域中的应用也已经被开发出来。相比光纤而言，微波有几个关键性的优点。最主要的优点是不需要铺设线缆的**路权**（right of way），任何人只要在每 50 千米处购买一小块地，在其上建造一个微波塔，就可以完全绕过电话系统进行通信了。这就是 MCI 公司迅速崛起成为一个新长途电话公司所采取的做法（与 AT&T 竞争的另一个对手 Sprint 公司则走了另外一条截然不同的道路。该公司是由南太平洋铁路公司组建起来的，而该公司已经拥有了大量的路权，所以它只要沿着铁路铺设光纤就可以了）。

微波相对来说比较不那么昂贵。建造两个简单微波塔（只是一根大的柱子再加上四根固定用的绳索）并且在每个塔上架设天线的成本有可能比在拥挤的都市或者山上铺设 50 千米的光纤要便宜得多，也可能比租用电话公司的光纤便宜，特别是当电话公司铺设光纤时还没有完全付清被卷起来的铜线时，租用其光纤的租金不便宜。

电磁频谱政策

为了避免出现混乱，各个国家和国际组织针对谁可以使用哪一段频率都有相应的协定。因为人人都希望得到更高的数据传输率，所以每个人都想要更多的频谱资源。国家政府机构为本国的调频/调幅无线电台、电视、移动电话等应用分配了相应的频谱，同时也为电话公司、警察、海军、航空、军队、政府和许多其他的竞争用户等组织和机构规定了使用的频谱。全球性的代理（WRC）机构 ITU-R 试图协调这些分配方案，以便厂商能够制造出在多个国家可用的无线电设备。然而，各国并不受 ITU-R 建议的约束，例如，美国负责频谱分配的是联邦通信委员会（FCC，Federal Communication Commission），它就曾经拒绝过 ITU-R 的建议（通常这些建议要求一些政治上强大的组织放弃某段频谱，因而遭到拒绝）。

即使某段频谱已经确定分配用于某种用途（比如移动电话），仍然会存在另外一个问题，即如何进一步分配频段中的频率给不同的运营商使用。过去曾经有三个做法被广泛使用过。最早的方法，通常称为选美比赛（beauty contest），要求每个运营商解释为什么它的提案能给公众谋取最好的利益，然后由政府官员来决定他们最欣赏谁讲的故事。让某些政府官员拥有将价值几十亿美元的财产授予他认为最好公司的权利通常会导致行贿、受贿、腐败、袒护裙带关系等甚至更糟糕的行为。此外，即使是一个认真诚实的政府官员，如果他认为一家外国公司比国内任何一家公司都能够更出色地完成某项任务，他必需做很多的解释工作才能得到大家理解。

第一种方法出现的问题导致了第二个方法的诞生：让所有感兴趣的公司摸彩（lottery）。这种做法带来的一个问题是即使那些对频谱使用没有兴趣的公司也可以参与摸彩活动。也就是说，如果一家快餐连锁店或者鞋帽连锁商店摸到了彩，那么它就可以将频谱高价转卖给某个运营商，对它来说，这种做法既没有任何风险，又可以获得巨额利润。

上述这种将巨额财产赠与并非精心挑选而是有一定随机性公司的做法招来了各界的严厉批评，因而导致了第三个分配方法的出台：将带宽拍卖（auction）给出价最高的竞标者。2000 年，英国拍卖了用于第三代移动通信系统的频率，拍卖前的期望值是 40 亿美元，但拍卖后实际获得了 400 亿美元。运营商们争先恐后拼个你死我活，唯恐错失进入移动通信领域的良好时机。这次事件打开了其他国家政府的贪婪之门，并启发了他们也要掌控类似拍卖活动的想法。这个做法非常有效，但是也给一些运营商留下了太多的债务，以至于他

们濒临破产的境地。即使在最好的情况下，运营商们也必须要经营很多年才能补偿这笔巨大的频谱许可费。

一种与上述方法完全不同的频率分配方法是根本不分配频率。相反地，这种方法允许任何人随意地传输数据，但对所用的功率进行控制，使得发射台只能在很短的距离内工作，因而不会和其他用户形成相互干扰。因此，大多数政府把一些频段保留下来用于非许可性应用，这些频段称为工业科学医学（ISM，Industrial, Scientific, Medical）。车库门控制器、无绳电话、无线电遥控的玩具、无线鼠标，以及许多其他的无线家用设备都使用 ISM 频段。为了把这些未经协调的设备之间的干扰降到最小，FCC 强制所有在 ISM 频段上工作的设备都必须限制发射功率（比如，1 瓦），并且使用其他技术把它们的信号分散到一个很大的频率范围。这些设备还必须注意避免干扰到雷达装置。

ISM 频段在频谱中的位置在不同的国家有不同的规定。例如，在美国功率小于 1 瓦的联网设备可以使用图 2-13 所示的频段而无须 FCC 的许可。900 MHz 频段曾经被用在 802.11 的早期版本，但是现在太拥挤了。2.4 GHz 频段在大多数国家都可以免费使用，虽然它会受到来自微波炉和雷达装置的干扰，但依然被普遍地应用于蓝牙和 802.11b/g。5 GHz 频段包括非许可的国家信息基础设施（U-NII，Unlicensed National Information Infrastructure）频段，该频段相对来说还没有被完全开发。因为 5 GHz 频段拥有最大的带宽，并且已经被用于 802.11a，所以它很快变得流行起来。

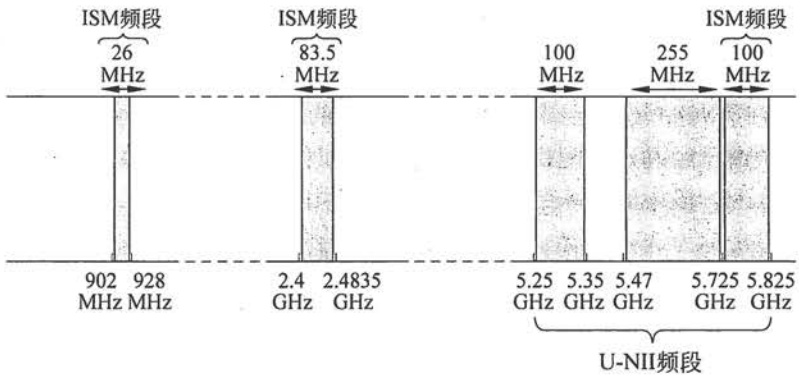


图 2-13 美国无线设备使用的 ISM 和 U-NII 频段

非许可频段已在过去 10 年间获得巨大的成功。免费使用该频谱的能力激发了无线局域网和无线个域网领域的大量创新，诸如 802.11 和蓝牙技术的广泛部署很好地印证了这点。这种创新的持续必然需要更多的频谱。一个令人振奋的发展是 2009 年美国联邦通讯委员会决定开放大约 700 MHz 的空白频段可未经许可自由地使用。空白频段（white spaces）指已经被分配出去但还未在本地被使用的那些频段。2010 年美国从模拟电视广播过渡到全数字电视广播后释放出大约 700 MHz 的白色空间。目前唯一的困难在于若要使用这些空白频段，非许可设备必须具备能检测出附近可能存在的任何许可发射器（包括无线麦克风）的能力，因为这些设备已经获得了使用频率波段的优先权。

另一场正在各地发生的一系列活动围绕着 60 GHz 频带展开。美国联邦通信委员会于 2001 年把 57～64 GHz 的频段开放给无牌照经营。如此巨大的频谱范围超过了所有其他 ISM 频段相加后的总和，因此可用来支撑高速网络的运营。这样的高速网络能将高清晰电视节

目穿过客厅传输到卧室。在 60 GHz，无线电波会被氧气吸收掉。这意味着信号不能传播得很远，因而非常适合于小范围网络。高频（60 GHz 是刚好低于红外辐射的极高频，或毫米级波段）对设备制造商构成了最初的挑战，但市场上已经有产品出现。

2.3.4 红外传输

非引导性的红外波被广泛应用于短程通信。电视机、录像机和立体声音响的遥控器都采用红外线通信。相对来说，红外线的传播具有方向性、便宜并且易于制造，但是它们有个很大的缺点：就是不能穿过固体物体（你可以试着站在遥控器和电视机之间，看看遥控器是否还能工作）。一般而言，当从长波无线电向可见光变化时，波的行为越来越像可见光，同时也越来越不像无线电波。

另外，红外线不能很好地透过固体墙壁也是一个优势。这意味着建筑物中某个房间内的一个红外系统不会干扰其相邻房间或相邻建筑物内的另一个类似系统，就好像你不可能用自己的遥控器去控制邻居家的电视机一样。并且，正是这一点使得红外系统的防窃听安全性比无线电系统要好。因此，经营红外系统并不需要政府的许可；相反地，如果无线电系统工作在 ISM 频段以外的频率，则必须获得政府的许可。红外通信在桌面环境中也有一定的用途，例如，将笔记本电脑与打印机用红外数据协会（IrDA，Infrared Data Association）标准连接起来。但在整个通信的游戏中，它并不是个重要的玩家。

2.3.5 光通信

非引导性光信号或自由空间光学已经被人们使用了几个世纪。Paul Revere 在他著名的旅程之前就在老北教堂（Old North Church）使用过二进制光信号。更现代一点的应用是将两个建筑物内的 LAN 通过安装在各自房顶上的激光连接起来。使用激光的光信号本质上是单向的，所以通信的每一端都必须有自己的激光发生器和光探测器。这种方案以极低的成本提供了非常大的带宽，相对来说安装容易；而且与微波传输不同的是它不要求获得 FCC 许可。

激光的强度，体现在一个很窄的一束光，同时这也是它的弱点。将一束 1 毫米宽的激光，瞄准 500 米开外只有一个针头大小的目标，这样的要求类似于近代（19 世纪）Annie Oakley 那样准的枪法。通常，在系统中放置一个镜头可以让激光束轻微地张开。更糟糕的是，风和温度的变化可以扭曲激光束的形状；而且激光束无法穿透雨水或大雾，虽然在阳光明媚的日子里它们工作得很好。然而，这些因素在用激光连接两个航天器时就不是个问题。

有一次，作者之一（AST）参加了在欧洲某个现代宾馆召开的会议。会议的组织者为与会者们提供了一个布满终端设备的房间，以便与会者在无聊的报告期间可以阅读自己的电子邮件。由于本地的 PTT 不愿意为了只有 3 天的会议而安装大量的电话线，所以会议组织者在屋顶上放置了一个激光设备，并使其对准几千米开外一所大学的计算机科学楼。在会议前一天晚上，会议组织者测试了该系统，一切工作良好。第二天，天气很好，晴空万里，但从上午九点开始无线链路突然被中断，而且一整天都无法恢复正常工作。这种情况持续了两天，直到会议结束后，会议组织者才发现了问题症结所在。原来，白天太阳的热

量使得建筑物房顶的气流上升，如图2-14所示。这些纷扰的气流导致激光束产生了偏差，总是在激光探测器附近晃动，就像大热天波光粼粼的路面一样。这个例子告诉我们，要想在好的条件和坏的条件下同时工作得很好，非引导性光学链路必须具备足够的容错工程设计。

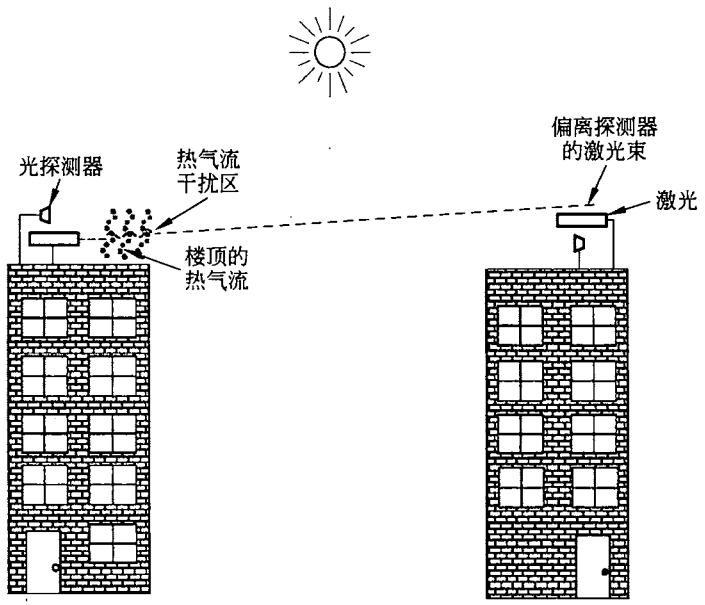


图2-14 对流的气流可干扰激光通信系统（这里显示了有两个激光器的双向系统）

非引导性光通信看上去像今天网络技术中的异类，但它可能很快会变得普及起来。我们周围充斥着相机（传感光）和显示器（使用LED和其他技术来发光）。通过特定的编码方案，可以将数据通信建立在这些显示器的层次之上。信息可以根据LED指示灯的开启和关闭来编码，但开或关应低于人类的感知阈值。这种以可见光方式实行通信在本质上是安全的，可用来创建一个围绕着显示器的低速网络，并由此衍生出各种各样无处不在的梦幻普适计算场景。紧急车辆的警示灯闪提醒附近的交通灯和车辆帮助它清空一条道路。即使是节日灯也可随着灯光的显示同步广播歌曲。

2.4 通信卫星

早在20世纪50年代和60年代初期，人们尝试着利用金属化的气象气球对信号的反射作用来建立通信系统。不幸的是，由于接收到的信号强度太弱，根本没有任何实际价值。后来，美国海军注意到空中存在一个永久性的气象气球——月球，它们通过月球对信号的反射作用建立了一个可实际运行的船-岸通信系统。

直到第一颗通信卫星发射上天，天体通信领域才有了进一步发展。人造卫星和真实卫星之间的关键区别在于人造卫星把信号送回来之前先对它们进行了放大处理。由此，人类的好奇心促成了一种强大通信系统的诞生。

通信卫星有一些令人感兴趣的特性，这些特性对于许多应用具有很大的吸引力。按照

最简单的方式理解，可以把一个通信卫星想象成天空中的一个大型微波中继器。它包含几个**转发器**（transponder），每个转发器侦听频谱中的某一部分，对入境信号进行放大；然后在另一个频率上将放大后的信号重新广播出去；出境信号采用不同的频率可避免与入境信号相互干扰。这种操作模式称为弯管（bent pipe）。还可以将数字化处理添加到分别处理数据，或者把数据流重定向到整个波段，甚至在卫星接收数字信息后再重新广播。以这种方式重新生成的信号相比弯管性能更好，因为卫星没有将上行信号中的噪声放大。下行波束可以很宽，覆盖地球表面相当大的一部分；也可以很窄，仅仅覆盖几百千米直径的区域。

根据开普勒原理，一颗卫星的轨道周期随着轨道半径之 3/2 次幂而变化。卫星越高，则轨道周期就越长。接近地球表面的周期大约为 90 分钟。因此，低轨道卫星惊鸿一瞥从人们的视线中消失，为了提供地球表面连续的覆盖区域需要很多这样的卫星，而且地面天线必须能跟踪它们。高度大约为 35 800 千米的轨道周期为 24 小时；高度为 384 000 千米的轨道周期大约是 1 个月，任何人只要观察一下月亮的运行规律就能证实这一点。

卫星的周期非常重要，但它并不是确定卫星安放位置的唯一因素。另一个问题是范艾伦辐射带（Van Allen belts）的存在。所谓范艾伦辐射带，是指受地球磁场影响的一些高带电粒子层。任何飞进范艾伦辐射带中的卫星会很快被毁坏。综合考虑上述因素，可以算出安全放置卫星的三个区域。图 2-15 列出了这三个区域以及它们的一些特性。下面我们简要地描述每个区域中的卫星情况。

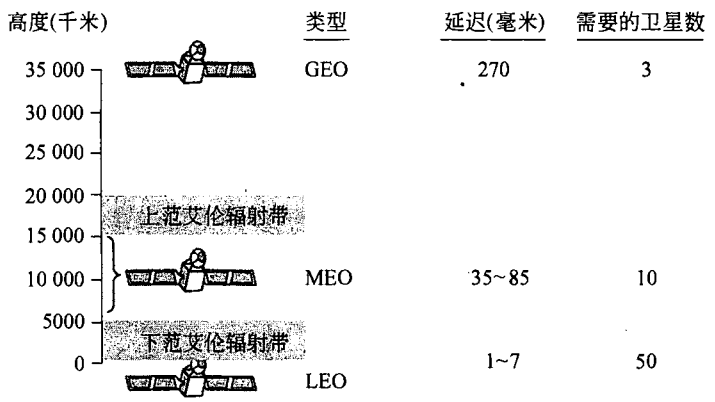


图 2-15 通信卫星以及它们的一些特性，包括离地球的高度、来回延迟时间和覆盖全球所需要的卫星个数

2.4.1 地球同步卫星

1945 年，科幻小说作家 Arthui C. Clarke 计算出，在赤道圆形轨道上方 35 800 千米高度的卫星可保持在空中静止不动，所以这样的卫星不需要考虑跟踪问题（Clarke，1945）。他继续描述了一个完整的通信系统，该系统利用这些（载人）地球同步卫星（geostationary satellite），还描述了轨道、太阳电池板、无线电频率以及发射程序。然而不幸的是，他得出的结论是这样的卫星不切实际。因为他认为不可能在轨道中放上耗电的、易碎的真空管放大器，所以他后来不再进一步深入思考这个想法。但是，他写了一些有关这种卫星的科幻小说。

晶体管的发明改变了这一切。1962 年 7 月，人类发射了第一颗人造通信卫星（Telstar）。

自那以后，通信卫星领域变成了一宗几十亿美元的买卖市场，太空也变得非常有利可图了。这些高空中飞行的卫星通常称为地球静止轨道卫星（GEO，Geostationary Earth Orbit）。

有了当前的技术水平，在 360° 赤道平面内把两颗同步卫星之间的距离设置成小于 2° 显然是不明智的，因为这样会产生干扰。按照 2° 的空间间隔，太空中同时最多只能放置 360/2=180 颗同步卫星。然而，每个转发器可以使用多个频率和多种极性来提高可用带宽。

为了避免太空中出现混乱，轨道槽的分配工作由 ITU 来统一完成。这个过程已经被高度政治化了，因为任何一个国家（只要它已经度过了石器时代）都希望拥有自己的轨道槽（它可以将轨道槽租赁给出价最高的国家）。然而，其他国家则主张，国家的主权不应该扩展到月球上，而且任何一个国家都不应该拥有其领土上空轨道槽的合法权利。更加具有对抗意义的是，商业通信并不是同步卫星的唯一应用，电视广播公司、政府和军队都希望从轨道蛋糕中分得一块。

现代卫星非常巨大，重量可达 5000 千克，消耗的电能要几千瓦，这些电能由太阳电池板产生。太阳、月亮以及行星引力都试图将卫星从它们预定的轨道槽和方向上移开，这种影响需要通过卫星上的火箭发动机来消除。由此进行的微调活动称为轨道控制（station keeping）。然而，当发动机的燃料耗尽时（通常需要 10 年左右的时间），卫星将会漂流，甚至开始翻滚乱动，所以必须将它关闭。最后，轨道衰落，卫星重新进入大气层，最终被烧毁（偶尔会撞击到地球上）。

轨道槽并不是各个国家争抢的唯一焦点。频率也是争抢的资源之一，因为下行链路的传输会干扰到原有的微波用户。因此，ITU 给卫星用户分配了特定的频率，其中主要的频率如图 2-16 所示。C 频段首先被分配给商业卫星用。在这个频段中目前分配了两个频率范围，其中较低的频率用于下行链路流量（从卫星发出），较高的频率用于上行链路流量（发向卫星）。为了能够同时在两个方向上传输流量，需要两个信道，每个方向一个信道。这些信道早已拥挤不堪，因为它们同时被许多公共运营商用做地面微波链路。L 和 S 频段是在 2000 年根据国际协议加入的。但是，这两个频段都很窄，也很拥挤。

频段	下行链路	上行链路	带宽	问题
L	1.5 GHz	1.6 GHz	15 MHz	低带宽，拥挤
S	1.9 GHz	2.2 GHz	70 MHz	低带宽，拥挤
C	4.0 GHz	6.0 GHz	500 MHz	地面干扰
Ku	11 GHz	14 GHz	500 MHz	雨水
Ka	20 GHz	30 GHz	3500 MHz	雨水，设备成本

图 2-16 主要的卫星频段

商业电信运营商可用的次最高频段是 Ku（K under）频段。该频段目前还不拥挤，在它的最高频率上，卫星的空间间隔可以近到 1° 。然而，存在另一个问题：雨水，水能很好地吸收这些波长短的微波。幸运的是，大暴雨通常发生在局部地区，所以使用几个相距较远的地面站而不是一个地面站就可以解决这个问题，当然这需要增加投入，包括额外的天线、电缆，以及用来快速切换地面站的电子设备。Ka（K above）频段的带宽也已经被分配用作商业卫星流量，但是使用该频段所必须配置的设备非常昂贵。除了这些商业频段以外，还有许多政府和军队使用的频段。

一颗现代卫星大约有 40 个转发器，大多数转发器具有 36 MHz 带宽。通常，每个转发器像一个弯曲的管道一样工作；但是最新的卫星具备了一定的处理能力，这就使得它可以执行一些复杂的操作。在最早的卫星上，转发器之间的信道划分是静态的：整个带宽被简单地分成固定的频段。现在，每个转发器的波束被分成多个时间槽，不同的用户可以轮流使用这些时间槽。我们在本章后面将详细学习两种技术(即频分多路复用和时分多路复用)。

第一颗地球同步卫星只有一个空间波束，它大约可以覆盖 1/3 的地球表面，它的范围称为它的足迹（footprint）。随着价格、尺寸、微电子功耗的不断下降，同步卫星已经有可能使用更为复杂的广播策略。每颗同步卫星都装配了多个天线和多个转发器。每个下行波束可以聚集到一个很小的地理区域中，所以，多个上行和下行传输可以同时进行。通常情况下，这些所谓的**点波束**（spot beam）呈现椭圆形状，覆盖范围可以小到直径只有几百公里。美国的通信卫星往往用一个宽的波束覆盖 48 个相互连接的州，而在阿拉斯加和夏威夷则使用点波束技术。

通信卫星领域的最新发展是低成本的微型站，有时候也称为小孔径终端（VSAT，Very Small Aperture Terminals）（Abramson，2000）。这些微型终端有一个 1 米或者更小的天线（相比之下，标准的 GEO 天线为 10 米），消耗的功率 1W 左右。上行链路一般可达 1 Mbps，质量还非常好，但下行链路往往高达数 Mbps。直播卫星电视使用这项技术来实现单向传输。

在许多 VSAT 系统中，微型站没有足够的功率实现相互之间的直接通信（当然是通过卫星）。相反，一种特殊的地面站可用来中继 VSAT 之间的流量，如图 2-17 所示。这种站称为**中继站**（hub），具有很大的高增益天线。在这种操作模式中，不管是发送方，还是接收方，都有一个大天线和强大的放大器。这里存在一种折中，就是用较长的延迟来换取廉价的终端用户站。

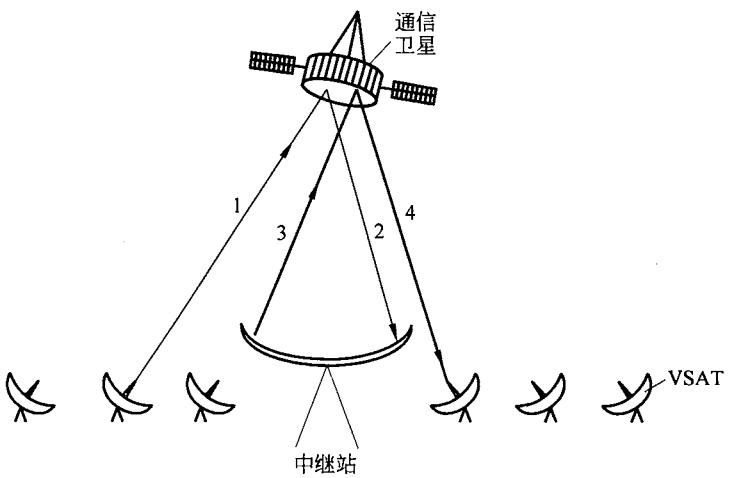


图 2-17 VSAT 使用中继站的图示

VSAT 在农村地区有很大的应用潜力。虽然它现在还没有受到广泛的欢迎，但是，世界上一半以上人口的居住地离最近的电话有 1 小时以上的行走路程。将几千个小村庄用电话线连接起来，所需的费用远远超出了大多数第三世界政府的预算，但是安装 1 米 VSAT 碟形天线并用太阳能电池来供电，却往往是可行的一种选择。VSAT 提供的技术可以将整

个世界连接起来。

通信卫星具备的一些特性与地面上的点到点链路在本质上有很大的不同。首先，即使信号从地面到卫星的往返两程都是以光速（接近每秒300 000千米）来传播，但对GEO卫星来说，这么长的来回距离引入了相当大的延迟。根据用户与地面站之间的距离及卫星的海拔高度，可以得出端到端的传输时间在250～300毫秒之间。典型的来回延迟时间值是270毫秒（对于使用了中继站的VSAT系统来说，一般为540毫秒）。

从比较的角度来看，地面微波链路的传输延迟大致上是每千米3微秒；同轴电缆或光纤链路的传输延迟大约是每千米5微秒。后者之所以比前者慢，是因为电磁信号在空中传播比在固体材料中传播得更快。

卫星的另一个重要特性在于它本质上是一种广播介质。它给转发器足迹范围内的上千个站发送一条消息的成本并不比发送给一个站所需要的成本高。对于某些应用，这种特性非常有用。例如，你可以想象这样的情景：一颗卫星将许多流行的Web页面在一个很广的范围内广播，信号覆盖范围内的大量计算机可以缓存该页面，以便后续的快速访问。尽管广播也可以用点到点的线路来模拟，但显然卫星广播更加便宜。另一方面，从隐私的角度来看，卫星通信却完全是个灾难：任何人都可以听到正在进行的所有一切传输。当需要安全性的保障时，必须对卫星通信进行加密处理。

卫星还有另外一种特性：即传输一条消息的成本与该消息所经过的距离无关。越洋通话服务并不比相邻街道之间的通话更贵。卫星的错误率极低，而且几乎可以立即部署，因而成为紧急救灾和军事通信中最主要的考虑因素。

2.4.2 中地球轨道卫星

在海拔较低的上空，两条范艾伦带之间，我们找到了中地球轨道（MEO，Medium Earth Orbit）卫星。站在地球的角度来看，这些卫星缓慢地漂过经线，大约6小时绕地球一圈。因此，当它们在空中移动时必须对它们的轨迹进行跟踪。因为它们比GEO低，所以它们覆盖在地面上的足迹要小一些，功率弱一些的发射器也能够与这些卫星通信。目前这种卫星只用于导航系统，尚未用于通信领域，所以我们这里不再进一步介绍。在20 200千米高空的轨道上有30颗全球定位系统（GPS，Global Positioning System）卫星，这是MEO卫星的最大应用实例。

2.4.3 低地球轨道卫星

海拔再往下移一点就来到了低地球轨道（LEO，Low Earth Orbit）卫星所在的位置。由于它们的运动速度极快，一个完整的系统需要大量的LEO卫星。另一方面，因为这些卫星与地球相距如此之近，地面站并不需要多大的功率就能收发往来卫星的信号，上行和下行的往返延迟只有几个毫秒。这种卫星的发射成本也因而很便宜。本节我们将介绍两个卫星星座的实例，一个关于语音服务，另一个与铱星（Iridium）及全球星（Globalstar）有关。

在卫星时代到来之前30年，人们很少使用低轨道卫星，因为这些卫星速度太快了，它们刚刚进入视野，转眼便又离开。1990年，Motorola公司突破了这种局面，它向FCC提

交了一份申请，要求允许其发射 77 颗低轨道卫星用于“铱计划”（Iridium project）（第 77 号元素是铱）。该计划后来修订为只用 66 颗卫星，如果按照原来的命名规则，计划名称也应该更名为“镝”（Dysprosium）（第 66 号元素是镝），但是这个名字的发音听起来有点像一种疾病。铱星计划的基本想法是一旦一颗卫星移出了视线，另一颗卫星立即就能替代它的位置。这份提议激发了其他通信公司的效仿狂潮。一时间，每一家公司都想推出一个低轨道卫星链。

经过长达 7 年与合作伙伴的磨合以及财政准备之后，铱星计划于 1998 年正式开始为公众提供通信服务。不幸的是，自从 1990 年以来移动电话网络发展迅猛，体积大且笨重的卫星电话的商业需求被完全忽略掉了。因此，铱星公司并没有获得利润，被迫于 1999 年 8 月宣布破产，成为历史上最悲壮的惨败事件。这些卫星以及其他资产（价值 50 亿美元）后来作为外太空旧货的形式，以 2500 万美元的售价卖给了一家投资商。

峰回路转的是铱星服务在 2001 年被重新启动，从此发展良好。用户通过手持设备直接与铱星卫星进行通信来获得铱星提供的语音、数据、寻呼、传真和导航服务，这些服务无论在陆地、海洋还是空中的任何地方都可用。铱星的客户包括海军、航空业和石油开采业，以及在地球上某些缺少电信基础设施地区（例如，沙漠、高山、南极和某些第三世界国家）旅行的人们。

铱星卫星位于高度为 750 千米处的圆形极地轨道上。它们被排列成南北向的项链状，每隔 32 纬度有一颗卫星，如图 2-18 所示。每颗卫星最多有 48 个单元格（点波束）和 3840 个信道容量，一些信道用于寻呼和导航，其他的信道用于数据通信和语音通信。

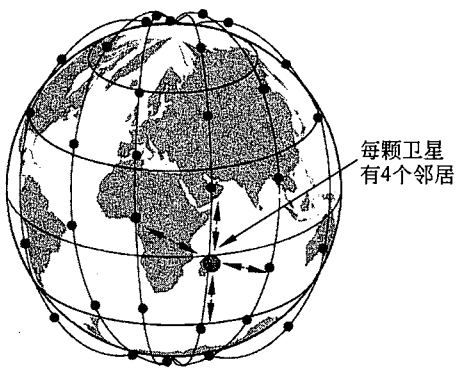


图 2-18 铱星卫星构成了围绕地球的 6 条项链

6 条卫星链按照图 2-18 所建议的那样把整个地球表面全部覆盖了。铱星有一个很有趣的特性，那就是，相距遥远的客户之间的通信必须通过太空进行，如图 2-19（a）所示。从这里我们可以看到，北极的一个呼叫者直接与一颗卫星联络。每颗卫星可以和 4 个邻居通信，其中 2 个位于同一个链中（图中已给出），另外两个位于相邻链上（图中未画出）。卫星中继北极用户的呼叫通过网格，直到最终被转发到位于南极的被叫者。

与铱星相对应的另外一种设计称为全球星（Globalstar）。全球星系统有 48 颗 LEO 卫星，但是使用了不同于铱星的交换模式。鉴于铱星系统中需要在卫星之间中继呼叫，因而必须在卫星上装备复杂的交换设备，全球星系统采用了一种传统的弯管设计思想。图 2-19（b）中北极发出的呼叫信号首先被送回到地球，被 Santa 工场的大型地面站捕获到；然后，该呼叫通过地面网络被路由到离被叫者最近的一个地面站；并且再通过一个弯管连接传递给被叫者。这种模式的好处在于它把大量的复杂性放在了地面上，相比空中处理的复杂性，在地面上管理要容易得多。而且，使用大型地面站天线还有额外的好处，它可发出强烈的信号并接收微弱的信号，这种特性意味着可以使用低功耗的电话。毕竟，电话发出的信号只有几个毫瓦的功率，当它回到地面站时已经非常弱，即使被卫星放大之后仍然很弱。

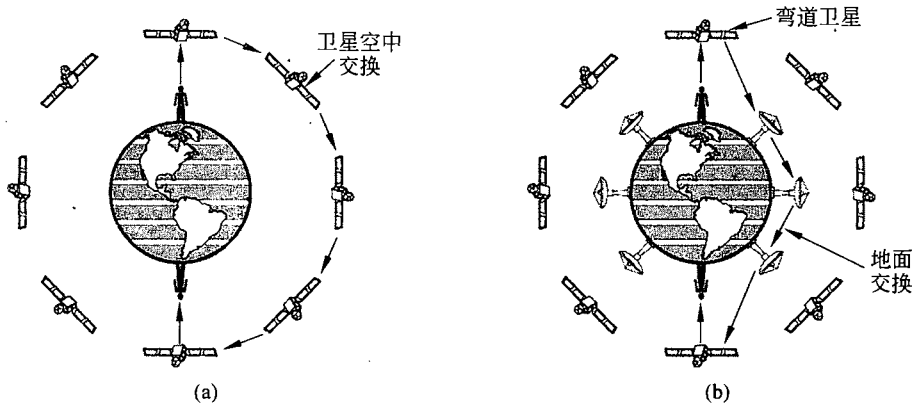


图 2-19
（a）空中中继；（b）地面中继

卫星连续不断地以每年20颗的速度被发射升空，而且卫星的重量也越来越大，包括现在重达5000千克的卫星。但还有许多专门为那些精打细算组织发射的微型卫星。为了使太空研究更容易，1999年来自加州理工和斯坦福大学的学者聚在一起共同定义了微型卫星及其相关发射标准，由此大大降低了发射成本（Nugent等，2008）。立方星（CubeSats）是以10厘米×10厘米×10厘米的立方体计量且重量不超过1千克的卫星单位，每单位的发射费用低至40 000美元。运载火箭是商业太空任务的第二项搭载的载荷。它基本上是一个管状物体，大约3个立方星长，使用弹簧把它们放入轨道。迄今为止，大约发射了20个立方星，还有许多正在制造中。这些立方星中的大多数与地面站通信使用的是UHF和VHF波段。

2.4.4 卫星与光纤

将卫星通信和地面通信作一番比较非常有意义。早在25年前，卫星的成功使人们认为未来的通信将建立在通信卫星的基础上。毕竟，在过去的100年中，电话系统的变化非常小，而且没有任何迹象表明在接下去的100年内将会有大的变化。这种似冰川移动般缓慢的发展态势很大程度上是由于监管环境变化很小造成的：人们期望电话公司以合理的价格提供良好的语音服务（电话公司基本上也做到了），同时作为回报电话公司可以获得投资的收益。对于想传输数据的用户，他们可以使用1200 bps的调制解调器，这一切在当时显得相当不错。

1984年在美国以及稍后在欧洲引入的竞争很快改变了这一切。电话公司开始用光纤代替其长途电话网络，并引入了诸如**非对称数字用户线**（ADSL，Asymmetric Digital Subscriber Line）这样的高带宽服务。它们还停止了长期以来用虚高的长途话费贴补本地服务的做法。突然间，地面光纤连接似乎成了大赢家。

然而，通信卫星拥有某些光纤所不具备（并且，有时甚至是不可能）的主要商机市场。首先，当快速部署成为主要问题时，卫星的优势一下子脱颖而出。对于战争年代的军事通信系统以及和平时代的灾难救助来说，快速响应能力至关重要。例如，2004年12月苏门答腊的大地震和随后的海啸中，通信卫星在24小时内就恢复了通信。因为世界上有一个非常发达的卫星服务供应商市场，因此这种特殊情况下的快速反应是完全可能的。供应商中

的一些大玩家，诸如拥有超过 50 颗卫星的国际通信卫星（Intelsat）服务商，可以租出去的容量几乎能满足任何地方的通信需求。而对于现有的卫星网络服务客户来说，可以方便快捷地安置好一个 VSAT，并由此获得到世界任何其他地方的 Mbps 链路。

卫星的第二个商机是在那些地面基础设施不发达的地区提供通信服务。现在许多人希望无论走到任何地方都能够通信。移动电话网络在人口密度较大的区域覆盖性能良好，但在其他一些地方则做得还远远不够（例如，在海上或沙漠中）。相反，铱星提供的语音服务则可无处不在地延伸到地球上的每个角落，甚至南极。而且地面基础设施的安装可能非常昂贵，要取决于具体的地形条件和必要的路权。例如，印度尼西亚就拥有其自己的卫星用于本地电话通信，显然发射一颗卫星比在 13 677 个群岛岛屿之间拉数千根海底电缆要便宜得多。

第三个商机在于当广播成为一种必不可少的需求时。数以千计的地面站可同时接收到卫星发送的消息。正是由于这个原因，卫星还可被用来分发许多网络电视节目到各地方电视台。现在出现了一个大型卫星广播服务市场，通过住宅卫星和汽车卫星，消费者能直接接收数字电视和广播节目。利用卫星广播特性还可以发布不同类型的其他内容。例如，如果一个组织需要将大量的股票、债券、商品价格等信息发送给几千个经销商，那么选择实用卫星系统来发送这些信息可能比采用地面上的模拟广播更加便宜。

总而言之，未来的主流通信看起来将是地面光纤与蜂窝无线电通信的结合，但是对于某些特殊用途，卫星通信更有优势。然而，无论选择什么样的方式，有个普遍的适用原则：经济学。虽然光纤提供了更多的带宽，但可以想象地面通信和卫星通信在价格上将展开激烈的竞争。如果技术的发展使得部署一颗卫星的成本大大降低（比如，将来的某一种航天飞机一次可以发射几十颗卫星），或者低轨道卫星能很好地抓住市场契机，那么光纤未必能赢得所有的市场。

2.5 数字调制与多路复用

前面我们已经学习了有线和无线信道的性质，现在把注意力转到发送数字信息的问题上。有线和无线信道运载模拟信号，模拟信号可表示成诸如连续变化的电压、光照强度或声音强度。为了发送数字信息，我们必须设法用模拟信号来表示比特。比特与代表它们的信号之间的转换过程称为**数字调制**（digital modulation）。

我们首先学习如何把数据比特直接转换成信号的一些方案。这些方案导致了所谓的**基带传输**（baseband transmission），即信号的传输占有传输介质上从零到最大值之间的全部频率，而最大频率则取决于信令速率。这是有线介质普遍使用的一种调制方法。然后我们将考虑通过调节载波信号的幅值、相位或频率来运载比特的调制模式。这些转换方案导致了**通带传输**（passband transmission），即信号占据了以载波信号频率为中心的一段频带。这是无线和光纤信道最常使用的调制方法，因为在这样的传输介质中只能在给定的频带中传输信号。

信道通常被多个信号共享。毕竟，用单根线缆传送几个信号比为每个信号铺设一根线缆要便利得多。这种信道的共享形式称为**多路复用技术**（multiplexing），多路复用技术可

以通过几种不同的方式实现。我们将给出一些时分复用、频分复用和码分复用的多路复用方法。

我们在本节描述的调制和复用技术已被广泛地用于电缆、光纤、地面无线和卫星信道。在下面的章节中，我们将着眼于这些技术在实际应用中的网络实例。

2.5.1 基带传输

数字调制的最直接形式是用正电压表示1，用负电压表示0。对光纤而言，可用光的存在表示1，没有光表示0。这种编码方案称为**不归零**（NRZ，Non-Return-to-Zero）。这个名字听起来有点奇怪，这里有其历史原因，只是简单表示信号遵循数据而定。图2-20（b）给出了一个例子。

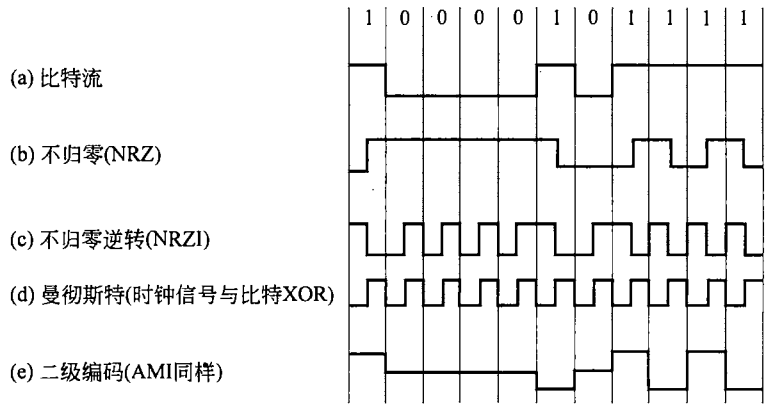


图2-20 线性码

一旦NRZ信号被发出去，它就沿线缆传播。在线缆的另一端，接收器以一定周期对信号采样，然后把采样信号转换成比特。接收到的信号看上去与发出来的不完全一样，这是因为信道本身造成的信号衰减和信号失真，以及接收器噪声对接收信号造成的影响所致。为了从信号中解码出比特，接收器把信号样值映射到最接近的符号。对于NRZ来说，接收到正电压表示发送过来的是1，接收到负电压表示发送过来的是0。

NRZ方案非常简单，因而是我们学习编码技术的一个很好起点。但也正因为它简单，所以实际上很少被使用。稍微复杂的编码方案能将比特转换成更加满足工程考虑的信号。这些模式称为**线路编码**（line codes）。下面，我们将描述有助于增加带宽效率、时钟恢复和DC平衡的线性编码方案。

带宽效率

采用NRZ编码，每2个比特信号（在1和0交替的情况下）可能在正电压和负电压之间循环。这意味着我们需要至少B/2 Hz的带宽才能获得B bps的比特率。带宽和速率之间的这种关系源自于尼奎斯特定律（参见等式（2-2））。这是个根本性的限制，如果没有更多的带宽可用我们将不可能获得比NRZ更快的速率。带宽通常是一种有限资源，即使对有线信道也一样。信号频率越高衰减越大，其可用性就越小，而且高频信号还需要更快的电子设备。

利用有限带宽的一种更有效策略是使用两个以上的信号级别。例如，采用 4 个电压级别，我们可以用单个**符号**（symbol）一次携带 2 个比特。只要接收器收到的信号强度足够大到能区分出信号的 4 个级别，这种方案就切实可行。此时信号变化的速率只是比特率的一半，因而减少了所需的带宽。我们把信号改变的速率称为**符号率**（symbol rate），以示区别于**比特率**（bit rate）。比特率是符号率与每个符号的比特数的乘积。符号率的较早称谓是**波特率**（baud rate），尤其在电话调制解调器设备的应用中很普遍，该调制解调器能将数字数据通过电话线发送出去。在一些文献中，比特率和波特率术语的使用往往不那么妥当。

请注意，信号的级别数不一定非是 2 的幂次方。实际情况往往不是这样，某些信号级别被用于防止出错和简化接收器的设计。

时钟恢复

对于所有将数据比特编码到符号的方案，接收器必须知道何时一个符号结束和下一个符号开始，才能对正确信号进行采样。在 NRZ 编码方法中，符号简单地对应为电压等级，一长串的 0 或 1 使信号级别保持不变。经过一段时间以后，接收器很难区分出各个比特，比如 15 个 0 看起来很像 16 个 0，除非有一个非常准确的时钟。

精确的时钟有助于解决上述问题，但对商品设备来说这个解决方案太昂贵了。请记住，我们是在以许多 Mbps 速率运行的链路上计时比特，因此时钟的漂移应该比最长允许运行的微秒零头还要小。这样的时钟漂移对慢速链路或短消息可能是合理的，但它显然不是个通用的解决方案。

一种策略是给接收器发送一个单独的时钟信号。额外的一根时钟线对于计算机总线或者短电缆来说没什么大不了的，因为本来它们就有许多平行的线。但对于大多数网络链路来说却是非常浪费的一种做法，如果我们能用另外一条线发送时钟信号，那么我们宁可用它来发送数据。聪明点的办法是把数据信号和时钟信号异或混合在一起，而不是用额外的一根线来传时钟，如图 2-20（d）所示。这里时钟在每个比特时间内产生一次跳变，所以它以两倍于比特率的速度运行。当时钟与 0 电压异或时，只是简单地将时钟信号产生一次“从低到高”的转变，这种信号跳变表示逻辑 0；当时钟与 1 电压异或时产生一次“从高至低”的相反转变，这种信号跳变表示逻辑 1。这样的编码方案称为**曼彻斯特**（Manchester）编码，主要用在经典以太网上。

由于上述时钟信号的缘故，曼彻斯特编码的主要缺点在于需要两倍于 NRZ 编码的带宽，而且我们已经了解到带宽是非常重要的资源。很自然地人们产生了一种基于下列想法的不同的策略，就是应该以确保存在足够信号跳变的方式对数据进行编码。NRZ 编码模式只有在面临一长串 0 和 1 的时候才存在时钟恢复问题。如果有频繁的信号转换，对接收器来说就很容易地与入境符号流保持同步。

作为朝着正确方向迈出的第一步，我们可以把编码简化成这样：1 定义为信号有跳变，反之 0 定义为信号无转变。这种编码方案称为**不归零逆转**（NRZI，Non-Return-to-Zero Inverted）。图 2-20（c）给出了一个例子。现在很流行用来连接计算机外设的**通用串行总线**（USB，Universal Serial Bus）标准就采用了 NRZI 编码模式。有了 NRZI，再长的一串 1 都不会产生时钟恢复问题。

当然，一长串的0仍然有问题，我们必须加以解决。如果我们是电话公司，我们可能只是简单地要求发送方不能传输太多的0。在美国最老的数字电话线，称为T1线路（T1 lines），事实上就要求用户不能连续发送超过15个0，才能保证它们正常工作。要真正解决这个问题，我们可以通过映射被传输的部分比特来打破连续多个0的禁锢。即包含连续多个0的比特组被映射成稍微长一点的比特模式，使得最终发送出去的信号中没有太多的连续0。

解决这个问题的著名编码方式就是所谓的4B/5B。每4个比特被映射成一个5比特模式，如何映射则按照一张固定的转换表进行。5位比特模式的选择使得映射结果永远不会出现连续三个0，如图2-21所示的映射关系。这种编码模式增加了25%的带宽开销，显然比曼彻斯特编码所需增加的100%带宽开销要好。由于有16个输入组合和32个输出组合，因此某些输出组合根本就没被使用。抛开那些具有太多连续0的组合，仍然剩下许多代码组合可用。作为该编码模式的额外收获，我们可以使用这些非数据代码组合来表示物理层的控制信号。例如，在某些场合下用“11111”表示线路空闲，而用“11000”表示一个帧的开始。

数据(4B)	码字(5B)	数据(4B)	码字(5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

图2-21 4B/5B映射

还有另一种方法，使得数据看起来很随机，这种编码方式称为**扰频/倒频**（scrambling）。在这种情况下，它很可能会出现频繁的信号转换。**扰频器**（scrambler）的工作原理是在发送数据之前，用一个伪随机序列异或（XORing）该数据。这种混合操作使得数据像伪随机序列一样随机（假设它是独立于伪随机序列的）。然后接收器用相同的伪随机序列对入境数据进行异或操作，由此恢复出真正的数据。为了实际操作可行，伪随机序列必须易于生成。随机序列的生成过程包括为一个简单的随机数发生器指定生成随机数的种子。

扰频方式因其不增加带宽或时间开销而很具吸引力。事实上，它常常有助于调节信号，使得在可能产生电磁干扰的主导频率成分（由重复数据模式引起的）不具有能量，因而降低了电磁辐射干扰。扰频的作用在于随机信号往往是“白色”，或者能量分散在整个频率成分上。

但是，扰频无法保证不会出现长期保持一种状态（电压），偶尔运气不好时还是可能会出现一直处于某一种状态。如果数据与伪随机序列恰好相同，那么它们的异或结果将是全0。这种结果一般不会发生在很难预测的长伪随机序列上。然而，如果是一个长度很短的随机序列或是可预见的随机序列，那么一些可能的恶意用户就会通过发送比特模式使得扰频结果出现一长串0，从而最终导致链路失败。电话系统中的SONET链路可用来发送IP数据包，为此制定的早期标准版本就存在这种缺陷（Malis和Simpson，1999）。对用户来说完全有可能发送肯定能引发问题的特定“杀手数据包”而自己没有任何意识。

平衡信号

在很短的时间内正电压与负电压一样多的信号称为**平衡信号**（balanced signals）。信号的均值为零，这意味着它们没有直流（DC）电气分量。没有直流分量是个优点，因为对于诸如带有变压器的同轴电缆或线路来说，其信道对直流分量有强烈的衰减，这是传输介质的物理性质所决定的。同样的，把接收器连接到信道上的**电容耦合**（capacitive coupling）方法只允许信号的交流（AC）部分通过。在这两种情况下，如果我们发送了一个平均值不为零的信号，其直流分量将被过滤掉，因而造成能源的浪费。

由于存在一个正电压和负电压的混合，所以平衡有助于提供时钟恢复所需的转换。平衡还提供了一个简单的校准接收器方法，因为测量所得的信号平均值可以作为解码符号的决策阈值。对非平衡信号来说，信号平均值可能漂离真正的判决级别，比如高密度的 1 将导致被接收器错误解码出（比实际个数）更多的符号。

一种构造平衡码的简单方法是使用两个电压级别来表示逻辑 1，比如用+1V 或-1V 表示 1，而用 0V 表示逻辑 0。发送 1 时，发射器在+1V 和-1V 之间选择，使得它们总是达到信号平衡。这种方案称为**双极编码**（bipolar encoding）。在电话网络中，则称为交替标记逆转（AMI，Alternate Mark Inversion）。这个称谓是建立在旧术语之上的，以前 1 称为“标记”，0 称为“空白”。图 2-20（e）给出了一个编码实例。

双极编码通过增加电压级别来实现信号的平衡。另外，我们还可以用类似于 4B/5B 的映射方法来达到平衡（以及用于时钟恢复的转换）。这类平衡码的一个例子是 8B/10B 的线性编码，它将输入流中的 8 个比特映射至 10 个比特输出，编码效率与 4B/5B 线性编码一样都是 80%。8 位中的 5 比特被分到一组，该组被映射到 6 比特，剩余 3 比特组成另一组被映射到 4 比特。6 比特和 4 比特符号被级联在一起组成一个输出组，被一同发送出去。在每一组中，某些输入模式可被映射到具有相同数目 0 和 1 的平衡输出模式。例如，“001”被映射成“1001”，显然这是平衡的。但实际上没有足够的组合用来作为所有的平衡输出模式。出现这种情况时，每个输入模式被映射到两个输出模式，其中一个输出模式有一个额外的 1，而另一个有一个额外的 0。例如，“000”被同时映射到“1011”和它的补“0100”。当输入比特被映射到输出比特时，编码器记住前一符号的**不均等性**（disparity）。不等指信号失去平衡的 0 或 1 的总数。然后编码器从输出模式或备用输出模式两个中选择一个，选择的依据是减少不等。采用 8B/10B 编码模式，不等至多为 2 个比特。因此，这种编码模式下的信号将永远不会远离平衡，而且也永远不会出现超过五个连续的 1 或 0，所有这些都有助于时钟恢复。

2.5.2 通带传输

一般情况下，我们在一个信道上发送信息所使用的频率范围并不是从零开始的。对于无线信道来说，发送非常低频率的信号很不切实际，因为天线的大小与信号的波长成比例，低频信号需要相当大的天线。在任何情况下，监管约束和避免干扰的需要往往决定了频率的选择。即使是电线，把信号放置在一个给定的频带上非常有用，因为这样在信道上可以

允许不同信号共存。这类传输称为**通带传输**（passband transmission），因为任意的一个频率波段都可用来传递信号。

幸运的是，在本章早些时候获得的基本结果都是基于带宽或者频带宽度。绝对频率值对于容量并不重要。这意味着，我们可以将一个占用 0~BHz 的基带信号搬移到频谱位置在 S~S+BHz 的通带上，而不会改变该信号所携带的信息，即使搬移后的信号看上去完全不同。为了在接收器处理信号，我们可以把它搬回到基带，这样更便于符号的检测。

数字调制可借助通带传输完成，即针对通带内的载波信号进行调节或调制。我们可以调制载波信号的振幅、频率或相位。针对这些信号特征的调制方法都有一个对应的名称。在**幅移键控**（ASK，Amplitude Shift Keying）中，通过两个不同的振幅分别表示 0 和 1。如图 2-22（b）所示的例子中，采用了一个非零幅值和一个零幅值。两个或更多的幅值等级可用来表示更多的符号。类似地，频移键控（FSK，Frequency Shift Keying）采用了两个或更多个不同的频率。如图 2-22（c）给出的例子中恰好使用了两个频率。最简单形式的**相移键控**（PSK，Phase Shift Keying）在每个符号的周期中，系统把载波波形偏移 0° 或 180°。由于只有两种相位，因此该调制方法也称为**二进制相移键控**（BPSK，Binary Phase Shift Keying）。这里“二进制”指的是两个符号，而不是每个符号代表 2 比特。图 2-22（d）给出了一个实例。更加有效地利用信道带宽的一个更好方案是使用 4 个偏移，例如 45°、135°、225°或 315°，这样每个符号可传输 2 个比特信息。这个版本的调制方式称为**正交相移键控**（QPSK，Quadrature Phase Shift Keying）。

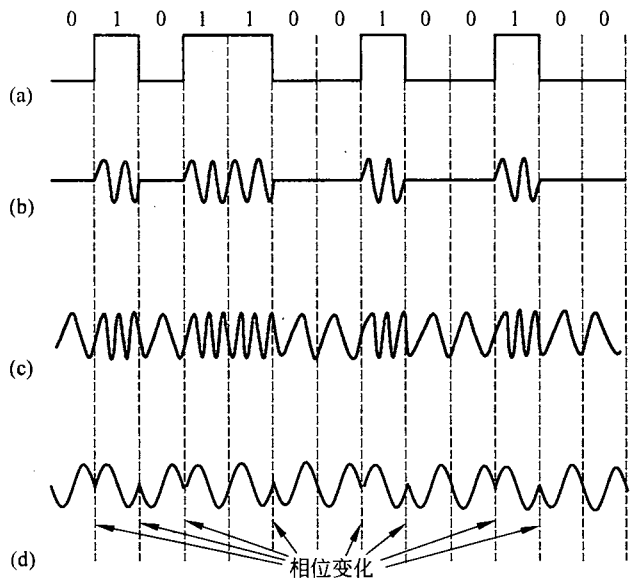


图 2-22

（a）二进制信号；（b）幅移键控；（c）频移键控；（d）相移键控

我们可以把这些调制模式结合起来综合使用，以便使每个符号传输更多的比特。因为频率和相位有关，即频率是相位随时间的变化率，所以一次只能调制频率和相位两个中的一个。通常情况下，振幅和相位可以结合起来一起调制。图 2-23 给出了 3 个实例。在每个例子中，黑点给出了每个符号合法的振幅和相位结合。在图 2-23（a）中，我们看到在 45°、

135°、225°和315°处有等距离的点。一个点的相位是以它为起点到原点的线与x正轴之间的角度来表示，一个点的振幅则是该点到原点的距离。该图表示了QPSK调制模式。

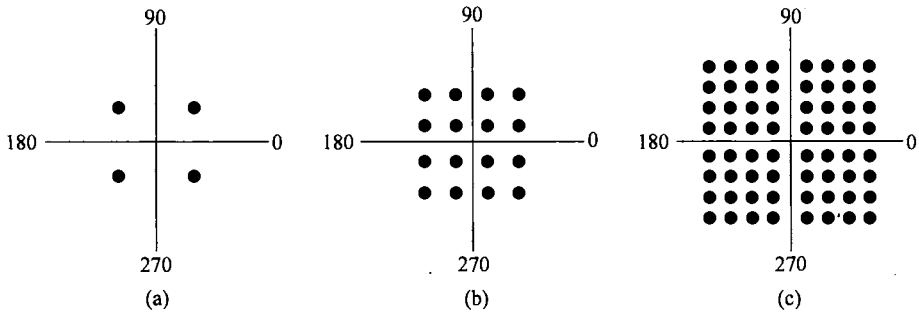


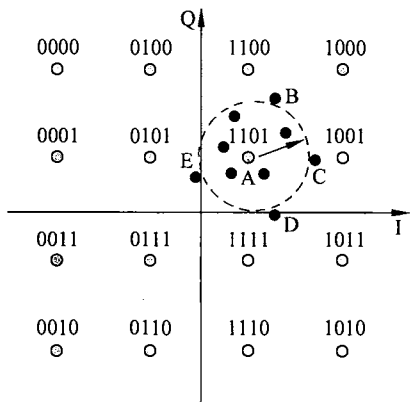
图 2-23
（a）QPSK；（b）QAM-16；（c）QAM-64

这种类型的图称为星座图（constellation diagram）。在图2-23（b）中我们看到一个具有密集星座的调制方案。该方案使用了振幅和相位的16种组合，因此可用每个符号传输4个比特。这种调制方式称为QAM-16，其中QAM表示**正交调幅**（Quadrature Amplitude Modulation）。图2-23（c）是个更加密集的调制方案，共使用了振幅和相位的64种不同组合，因此每个符号可传输6个比特，这就是所谓的QAM-64。甚至还可以使用更高阶的QAM调制方案。正如你看到这些星座图可能会产生的猜想一样，制造一个电子产品来产生基于所有轴组合值的符号要容易一些；而相对地，制造电子产品来产生基于振幅值和相位值组合的符号要难一些。这就是为什么这种调制方案看上去像四方形而不是同心圆的原因。

我们迄今所看到的星座图没有说明如何为符号分配比特。在决定如何分配时，一个重要考虑是接收器的少量突发噪音不会导致许多比特出错。如果我们把连续的比特值分配给相邻的符号，这种情况就有可能发生。例如，在QAM-16中，如果一个符号表示0111，其相邻符号表示1000，那么当接收器错误地采样到相邻符号时将会引起所有比特出错。一种更好的解决方案是把比特影射到符号，使得相邻两个符号只有1个比特的位置不同。这种映射方法称为**格雷码**（Gray code）。图2-24给出了格雷编码后的QAM-16星座图。现在，如果接收器把一个符号解码错了，在被解码符号接近发送符号的预期情况下，只会产生单个比特的错误。

2.5.3 频分复用

我们已经看到调制方案可以沿着有线或无线链路发送运载比特的信号。但是，规模经济在我们如何使用网络中发挥着重要作用。在两个不同办公室之间安装和维护一条高带宽传输线的成本和一条低带宽传输线的成本几乎相差无几（即主要成本来自挖壕沟的费用，而不在于壕沟里铺设什么样的电缆或光缆）。因此，人们研究出复用模式使多个信号共享传输线路。



当发送1101时：

点	解码成	比特错误
A	1101	0
B	1100	1
C	1001	1
D	1111	1
E	0101	1

图 2-24 QAM-16 格雷码

频分复用（FDM，Frequency Division Multiplexing）利用通带传输的优势使多个用户共享一个信道。它将频谱分成几个频段，每个用户完全拥有其中的一个频段来发送自己的信号。AM 调幅无线电广播就是 FDM 的一个应用实例。为它分配的频谱为 1 MHz，从 500～1500 kHz。给不同的逻辑信道（站）分配不同的频率，每个频率工作在频谱中的一部分，并且相邻信道之间的频谱间隔足够大，以便防止干扰。

更详细的例子可参见图 2-25，图中展示了采用 FDM 技术复用的三个语音级电话信道。滤波器将每个语音级信道限制成大约为 3100 Hz 的可用带宽。当多个信道被复用在一起时，为每个信道分配 4000 Hz 带宽。比语音通信所需多出来的那部分频带称为**保护带**（guard band），它使信道之间完全隔离。采用频分多路复用时，首先，每个语音信道的频率得到不同程度的提升；然后，把它们合并在一起。之所以能混合多个信道，是因为现在没有两个信道占据相同的频谱。注意，即使信道之间有保护带形成的间隔，相邻信道之间仍然可能存在某种重叠。重叠的出现在于真正的滤波器达不到理想的锐利边缘。这种现象意味着一个信道的锐利边缘对相邻信道来说是非热噪声。

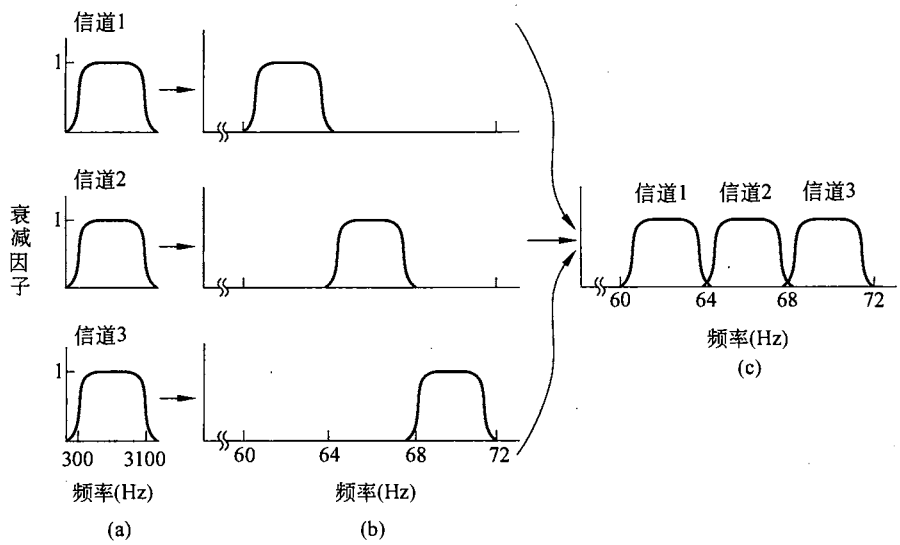


图 2-25 频分多路复用

（a）原始带宽；（b）提升到频谱上的带宽；（c）多路复用的信道

FDM 方案曾经被电话系统用来复用电话呼叫许多年，但现在已被其他技术所取代。然而，电话网、蜂窝电话、地面无线和卫星网络仍然在使用更高层粒度的 FDM。

发送数字数据时完全有可能把频谱更有效率地划分成没有保护带。在**正交频分复用**（OFDM，Orthogonal Frequency Division Multiplexing）中，信道带宽被分成许多独立发送数据的子载波（例如 QAM）。子载波在频域中被紧紧地包裹在一起。因此，从每个子载波发出的信号能扩散到相邻子载波。然而，如图 2-26 所示，每个子载波的频率响应被设计成在相邻子载波的中心为零。因而可以在子载波的中心频率采样而不会受到它们邻居的干扰。为了正常工作，需要一个保护时间来及时重复符号信号的一部分，以便获得所需要的频率响应。然而，这种开销远远少于许多保护带所需的开销。

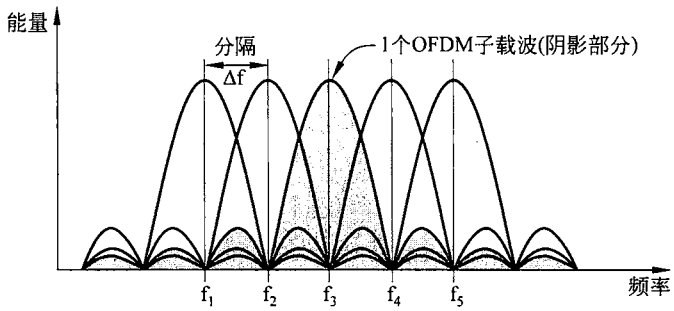


图 2-26 正交频分多路复用（OFDM）

OFDM 的想法已经存在很长一段时间，但仅在过去的 10 年间才被广泛采用，自此以后人们认识到依据数字数据的傅里叶变换，在所有子载波上（而不是针对每个子载波单独进行调制）有效实现 OFDM 是完全可能的。OFDM 已经被广泛用于 802.11、有线电视网络和电力线网络，而且正被计划用于第四代蜂窝系统。一般来说，一个高速率的数字信息流被分成许多个低速率流，这些低速率流通过子载波平行地传送出去。这种划分非常有价值，因为在子载波一级更易于应付信道退化（degraded）问题；而且，为了接收器更好地接收子载波，某些子载波或许被降级或者完全排除在外。

2.5.4 时分复用

相对频分复用的另一种复用方法是**时分多路复用**（TDM，Time Division Multiplexing）。在这种方式下，用户以循环的方式轮流工作。每个用户周期性地获得整个带宽非常短的一个时间，图 2-27 给出了三个流通过 TDM 复用的示例。每个输入流的比特从一个固定的**时间槽**（time slot）取出并输出到混合流。该混合流以各个流速率的总和速度发送。这种工作方式要求输入流在时间上必须同步。类似于频率保护带，为了适应时钟的微小变化可能要增加**保护时间**（guard time）间隔。

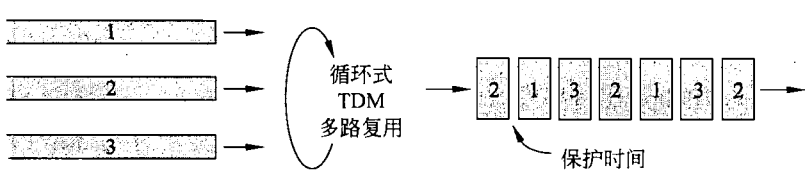


图 2-27 时分多路复用（TDM）

TDM被广泛用于电话网络和蜂窝网络，已经成为组成它们的一部分。为了避免混淆，我们必须清楚TDM完全不同于另一个**统计时分复用**（STDM，Statistical Time Division Multiplexing）。这里的前缀“统计”表明组成多路复用流的各个流没有固定的调度模式，而是根据其需求产生。STDM是包交换的另一个代名词。

2.5.5 码分复用

还有第三种多路复用技术以完全不同于FDM和TDM的方式工作。**码分复用**（CDM，Code Division Multiplexing）是**扩展频谱**（spread spectrum）通信的一种形式，它把一个窄带信号扩展到一个很宽的频带上。这种方法更能容忍干扰，而且允许来自不同用户的多个信号共享相同的频带。由于码分复用技术最常用于第二个目的，因此它称为**码分多址**（CDMA，Code Division Multiple Access）。

CDMA允许每个站利用整个频段发送信号，而且没有任何时间限制。利用编码理论可以将多个并发的传输分离开。CDMA不再假设冲突的帧被完全丢弃掉。相反，它假设多个信号可以线性叠加。在讨论具体算法之前，我们来看一个类似的场景：在一个机场候机大厅里，许多人正在两两交谈。TDM可以看作是所有的人都聚集在大厅里按顺序进行交谈。FDM可以看作是大厅里的人以不同的语调交谈，某些语调高些，某些语调低些，所有的交谈可同时进行并相互独立。CDMA可以看作是大厅里的每一对交谈使用不同的语言。讲法语的这一对在谈论有关法国的事情，并且把所有与法国无关的内容都当作噪声拒绝掉。因此，CDMA的关键在于：能够提取出期望的信号，同时拒绝所有其他的信号，并把这些信号当作噪声。下面简单描述CDMA的工作原理。

在CDMA中，每个比特时间被再细分成m个更短的时间间隔，这更短的时间间隔就称为**码片**（chip）。通常情况下，每个比特被分成64或者128个码片。但在下面给出的例子中，为了简便起见，我们将使用8个码片来说明CDMA的工作原理。每个站被分配得到唯一的m位码，称为**码片序列**（chip sequence）。为了教学目的，我们采用双极符号把码片序列写成一系列的-1和+1，这样更方便理解。下面就用括号表示码片序列。

若要发送比特1，站就发送分配给它的码片序列；若要发送比特0，它就发送其码片序列的反码。除此之外，不允许发送任何其他模式。因此，对于m=8，如果站A分配得到的码片序列是（-1 -1 -1 +1 +1 -1 +1 +1），那么它发送该序列就表示发出的是比特1，而发送（+1 +1 +1 -1 -1 +1 -1 -1）则表示发出的是比特0。实际上真正发出的是这些电压值的信号，但已足够令我们依据事先得到的码片序列来思考传来的是什么比特。

按照这种编码方式，本来每秒发送b个比特，现在变成每秒要发送mb个码片，这意味着采用CDMA的站比不使用CDMA的站所需发送的带宽增加了m倍（假设调制解调或编码解码技术没有任何变化）。如果我们有1 MHz的频段被100个站使用，那么采用FDM，每个站将得到10 kHz频段，它可以10 kbps的速率发送信息（假设每个Hz发送1个比特）。采用CDMA，每个站可使用全部的1 MHz频段，所以每个比特的码片速率为100，并且被扩展到信道上站的10kbps比特率中。

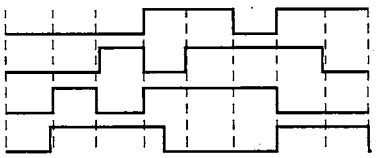
在图2-28（a）和2-28（b）中，我们给出了一个示例，显示分配给4个站的码片序列和它们表示的信号。每个站都有自己唯一的码片序列。我们假设用符号S表示站S的m码

片向量，用$\overline{S}$表示它的反码。所有的码片序列都两两正交（orthogonal），这意味着任何两个不同的码片序列S和T的归一化内积（写为S·T）为0。我们知道，利用**Walsh码**（Walsh code）可以产生这样的正交码片序列。按照数学方法，码片序列的正交性可以表示成：

$$S \cdot T \equiv \frac{1}{m}\sum_{i=1}^{m} S_i T_i = 0 \qquad (2\text{-}5)$$

A=(−1−1−1+1+1−1+1+1)
B=(−1−1+1−1+1+1+1−1)
C=(−1+1−1+1+1+1−1−1)
D=(−1+1−1−1−1−1+1−1)

(a)



(b)

S_1=C =(−1+1−1+1+1+1−1−1)
S_2=B+C =(−2 0 0 0+2+2 0−2)
S_3=A+$\overline{B}$ =(0 0−2+2 0−2 0+2)
S_4=A+$\overline{B}$+C =(−1+1−3+3+1−1−1+1)
S_5=A+B+C+D =(−4 0−2 0+2 0+2−2)
S_6=A+B+$\overline{C}$+D =(−2−2 0−2 0−2+4 0)

(c)

S_1·C=[1+1−1+1+1+1−1−1]/8=1
S_2·C=[2+0+0+0+2+2+0+2]/8=1
S_3·C=[0+0+2+2+0−2+0−2]/8=0
S_4·C=[1+1+3+3+1−1+1−1]/8=1
S_5·C=[4+0+2+0+2+0−2+2]/8=1
S_6·C=[2−2+0−2+0−2−4+0]/8=−1

(d)

图 2-28
（a）4个站的码片序列；（b）序列表示的信号；（c）6个传输实例；（d）站C信号的恢复

简单地说，相等的数目与不相等的数目一样多。后面将证明这种正交性质非常关键。注意，如果S·T=0，则$\overline{S}$·$\overline{T}$也是0。任何码片序列与自身的归一化内积一定是1：

$$S \cdot S = \frac{1}{m}\sum_{i=1}^{m} S_i S_i = \frac{1}{m}\sum_{i=1}^{m} S_i^2 = \frac{1}{m}\sum_{i=1}^{m} (\pm 1)^2 = 1$$

因为m项中的每一项内积都是1，所以求和的结果是m。同样地，$S \cdot \overline{S} = -1$。

在每个比特时间内，一个站可以传输比特1（发送自己的码片序列），也可以传输比特0（发送自己的码片序列的反码）；或者它保持沉默什么也不发送。现在我们假设所有的站在时间上都是同步的，因此所有的码片序列从同一个时刻开始。当两个或者多个站同时传输时，它们的双极序列线性相加在一起。例如，如果在一个码片周期中，3个站输出+1，一个站输出-1，则收到的是+2。你可以将这看成是信道上的电压值相加：3个站输出+1伏电压，一个站输出-1伏电压，结果得到2伏电压。例如，在图2-28（c）中，我们看到6个关于一个或者多个站同时传输比特1的例子。在第一个例子中，C传输比特1，所以我们只得到C的码片序列；在第二个例子中，B和C同时传输比特1，所以，我们得到它们的双极码片序列的和，即：

(−1 −1 +1 −1 +1 +1 +1 −1) + (−1 +1 −1 +1 +1 +1 −1 −1) = (−2 0 0 0 +2 +2 0 −2)

为了恢复出某个特定站的比特流，接收方必须预先知道这个站的码片序列。只要计算收到的码片序列与该站的码片序列的归一化内积，就可以恢复出该站的比特流。如果收到的码片序列为S，接收方正在监听的那个站的码片序列为C，那么，它只要计算两者的归一化内积，即S·C。

为了弄明白为什么这样做，请考虑这样的情形：两个站A和C同时传输比特1，并且B同时传输比特0。接收方看到的是和值$S = A + \overline{B} + C$，然后计算：

$$S \cdot C = (A + \overline{B} + C) \cdot C = A \cdot C + \overline{B} \cdot C + C \cdot C = 0 + 0 + 1 = 1$$

前两项消失了，因为所有的码片序列都是精心挑选出来的，它们两两正交，正如等式（2-5）所示。现在应该很清楚，为什么码片序列必须具备这样的正交特性。

为了使解码过程更加具体化，我们给出了图2-28（d）中的6个例子。假设接收方对从6个信号S_1～S_6中提取出站C发送的比特感兴趣。它将收到的S与图2-28（a）中的向量C逐个分量求乘积，再将结果累加起来，然后取结果的1/8（因为这里m=8）。6个例子中包括了站C沉默什么也没传输、传输一个比特1和传输一个比特0时，以及与其他传输相结合的情形。正如图中所示，每次都能解码出正确的比特。这就好像在机场大厅里讲法语一样。

原则上，给定足够的计算能力，只要接收方并发地为每个发送方运行相应的解码算法，就可以一次收听到所有发送方发出的信息。在现实生活中，我只想说谈何容易，知道哪个发送方在传输或许更有用。

在理想情况下，也就是我们在这里讨论的无噪声CDMA系统中，通过采纳更长的码片序列，使得同时发送的站的数量可以任意大。对于2^n个站，Walsh码可以提供2^n个长度为2^n的正交码片序列。然而，这里存在一个重要的限制，那就是我们假设所有的码片在接收方都是同步的。这种同步在某些应用中近乎不太可能成立，比如蜂窝网络（20世纪90年代就开始广泛部署CDMA了）。这样就导致了不同的设计。我们将在本章后面再次回到这个主题，重点描述异步CDMA如何不同于同步CDMA。

除了蜂窝网络，CDMA还被用于卫星通信和有线电视网络。我们在这里的简短介绍掩盖了其中许多复杂的因素。想要深刻理解CDMA的工程师应该阅读（Viterbi，1995）和（Lee&Miller，1998）。然而，阅读这些参考文献需要相当多的通信工程背景知识。

2.6 公共电话交换网络

如果同一公司或者同一组织的两台计算机相距很近，现在它们需要进行通信，则通常最容易的做法是用一根电缆把它们连接起来。局域网就是这样工作的。然而，当距离很远，或者有很多台计算机，或者这条电缆必须穿过一条公共道路或者其他公共场所，铺设私有电缆的费用往往是不切实际的。而且，几乎在所有的国家中，在公共地产上架设（或者在地下铺设）传输线路都是非法的。因此，网络设计者必须依赖已有的电信设施来建设网络。

这些设施一般很多年以前就已经设计好了，特别是**公共交换电话网络**（PSTN，Public Switched Telephone Network），它们具有完全不同的目标：以一种或多或少可识别的形式来传输人类语音。把它们用作计算机之间的通信，最多只是勉强够格而已。要看清楚问题的大小，考虑一条运行在两台计算机之间的廉价商用电缆，它可以1Gbps或甚至更高的速率传输数据。相比之下，电话调制解调器的快速替代品——典型的ADSL，运行速率大约为1Mbps。这两种传输介质之间的差别犹如飞机在空中游弋与双足在地面散步之间的差异。

尽管如此，电话系统与（广域）计算机网络紧密地交织在一起，因此值得我们投入一些时间来详细研究它。制约联网的因素最后发现原来在连接客户的“最后一英里”处，而不是电话网络内部的中继线（trunk）和内部交换机。光纤和数字技术逐渐出现在网络边缘，已经大大改善了这种情况，但这仍然需要时间和金钱。在漫长的等待中，计算机系统设计人员花费了大量的时间和精力来解决如何有效地使用电话网络涉及的一系列问题，他们已

经习惯了那些性能级别至少好三个以上的系统。

在下面的章节中，我们将描述电话系统及其如何工作。至于电话系统内部结构更多的信息请参见（Bellamy，2000）。

2.6.1 电话系统结构

当贝尔（Alexander Graham Bell）于1876年获得了电话专利之后不久（仅比其竞争对手 Elisha Gray 早了几个小时），他的新发明就有了大量的需求。最初的市场是电话销售，当时电话还是成对出售的，顾客必须自己在一对电话之间拉上一条线。如果电话主人想跟其他 n 个电话主人通话，则他必须拉 n 根单独的电话线到 n 个朋友家。如此工作方式下，不到一年城市就布满了电话线，这些电话线杂乱无章地穿过房屋和树林。很快情况就明朗了，像图 2-29（a）所示的那样，将每部电话与其他所有电话实行全连接的模式是行不通的。

令人欣慰的是贝尔早就发现了这个问题，他组建了贝尔电话公司，于 1878 年开放了他的第一个交换局（在康涅狄格州的 New Haven 城市）。该公司为每个客户拉一条通到家里或者办公室的电话线。当客户要打电话的时候，首先摇动手柄使得电话公司办公室铃声响起，从而引起接线员的注意；然后接线员手工通过短跳线电缆把主叫方与被叫方连接起来。单个交换局的模式如图 2-29（b）所示。

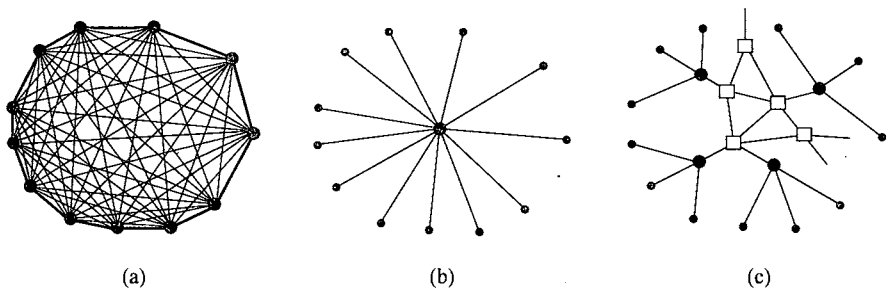


图 2-29
（a）全连通网络；（b）中心交换网络；（c）两层体系网络

很快地，贝尔系统的交换局遍布到了各地，人们又要求能够在不同城市之间打长途电话，所以贝尔系统开始将交换局连接起来。最初的问题很快再次出现了：每个交换局与其他每个交换局直接通过电话线连接起来，很快地整个电话系统结构变得无法管理。所以发明了二级交换局。过了一段时间以后，多个二级交换局出现了，如图 2-29（c）所示。最终，电话系统的整个层次结构增长到了五级。

到了 1890 年，电话系统的 3 个主要部分已经全部就位：交换局、客户与交换局之间的线路（现在使用的是平衡型绝缘双绞线，原来使用有接地回路的裸线），以及交换局之间的长距离连接。有关电话系统的简短技术史，请看（Hawley，1991）。

从那时起虽然上述三个领域一直在不断改善，但基本贝尔系统模式已经保持了这 100 多年而完好无损。下面的描述已被高度简化，但是给出了电话系统的基本面貌。每部电话机有两根铜线直接连接到电话公司最近的**端局**（end office），端局也称为**本地中心局**（local central office）。这段距离通常为 1～10 千米长，在城市比在农村要短一些。在美国，大约有 22 000 个端局。每个电话客户的电话机与端局之间的双线连接在电话行业中称为**本地回**

路（local loop）。如果世界上所有的本地回路都端到端地连接起来，则其长度相当于地球到月球来回1000次的长度。

曾经有一段时间，AT&T公司资产的80%是本地回路中的铜。因此，AT&T公司实际上是世界上最大的铜矿主。幸运的是，这样的事实并没有在投资圈里广为人知。如果被别人知道，个别市场捣乱者就有可能买下AT&T公司，停止全美国的所有电话服务；然后把所有的电话线收回来卖给炼铜商，以此获得暴利。

连接到某个端局的用户如果呼叫另一个也连接到该端局的用户，则局内的交换机制会在这两个本地回路之间建立一条直接的电气连接。在整个通话过程中这个连接保持不变。

如果被叫电话连接到另一个端局，则建立连接的过程就不同了。每个端局都有一些出境线路连到一个或者多个附近的交换中心，这些交换中心称为**长途局**（toll office），如果它们在同一个本地地区，则称为**汇接局**（tandem office）。这些出境线路统称为**长途连接中继线**（toll connection trunk）。交换中心的种类和它们的拓扑结构因国而异，主要取决于所在国的电话密度。

如果主叫方和被叫方各自所在的端局碰巧都有一条长途中继线连接到同一个长途局（如果它们相距比较近，则这种可能性很大），那么双方的连接就能在这个长途局内建立起来。图2-29（c）显示了一个简单的电话网络，该网络仅包含电话（小圆点表示）、端局（大圆点表示）和长途局（方块表示）。

如果主叫方和被叫方没有共同的长途局，那么它们之间的路径将在更高层次上的某个地方建立。长途局相互之间通过高带宽的长途中继线（intertoll trunk，或者interoffice trunk）通信。到1984年AT&T解体之前，美国电话系统采用分层路由来找到一条路径，即逐级向上进入更高的层次，直到到达一个双方共同的交换局。此后，这种连接方式被更灵活的非层次路由所取代。图2-30显示了一个长途连接是如何被路由的。

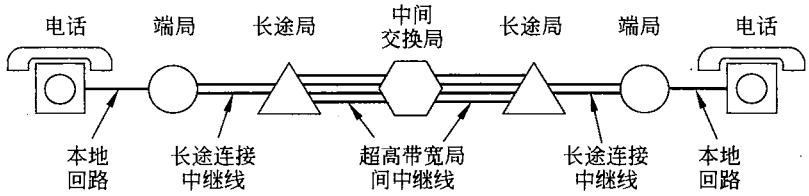


图2-30 一个长途呼叫的典型电路路由

电信领域用到了各种传输介质。现代化的写字楼通常用5类双绞线，与此不同的是接入家庭的本地回路多数是3类双绞线，光纤的使用刚刚开始出现。在交换局之间，广泛使用的是同轴电缆、微波，特别是光纤。

在过去，整个电话系统中的传输都是模拟的，实际的语音信号以电压的形式从源端传输到接收方。随着光纤、数字电路和计算机的出现，现在所有的中继线和交换设备都是数字的，只有本地回路仍然是模拟的，它也是整个系统中最后保留使用模拟技术的地方。数字传输之所以成为优先选择，是因为它不需要像模拟传输那样经过一系列放大器之后必须精确地还原出模拟波。对数字传输而言，只需要接收方能够正确地区分出比特0和比特1就足够了。这种特性使得数字传输比模拟传输更加可靠。而且，系统的维护工作更加容易，且维护成本也要便宜得多。

总而言之，电话系统由以下三个主要部分构成：

（1）本地回路（进入家庭和公司的模拟双绞线）。

（2）中继线（连接交换局的数字光纤）。

（3）交换局（电话呼叫在这里从一条中继线被接入到另一条中继线）。

在稍微跑题介绍完电话政治学之后，我们将回到这三个主要组成部分进行详细的讨论。本地回路为每一个人提供了接入到整个电话系统中的途径，所以它们至关重要。不幸的是，它们也是系统中最薄弱的环节。对于长途中继线，主要的问题是如何将多个呼叫合并起来，并且通过同一条光纤发送出去。这项技术称为多路复用，我们将采用 FDM 和 TDM 来做到这一点。最后，我们将讨论两种基本但完全不同的交换方法。

2.6.2 电话政治化

在 1984 年之前的几十年中，贝尔系统在全美大多数地区既提供本地电话服务，也提供长途电话服务。在 20 世纪 70 年代，美国联邦政府认为这是一种非法垄断，并且提起诉讼要将它分解。政府赢了，1984 年 AT&T 公司被分解为 AT&T 长话公司（AT&T Long Lines）、23 个 Bell 运行公司（BOC，Bell Operating Company）和其他一些部门。23 个 BOC 组织成 7 个区域性 BOC（RBOC），使得它们在经济上可以生存下去。美国电信业的整体本质在一夜之间被法官的判决（而不是国会的法案）改变了。

这次分解案的细节由一份名为修订的最终判决书（MFJ，Modified Final Judgement）说明，这个名字本身就是一个矛盾，如果判决书可以被修改的话，它就不是最终的。这次事件导致了更为激烈的竞争，各电话公司无论对个人还是对企业都在更好的服务和更低的长途电话费用上展开竞争。然而，本地服务的价格反而上升了，因为失去了来自长途电话业务的补贴，本地服务必须要自负盈亏。许多其他国家现在已经引入了类似竞争办法。

与我们学习内容直接相关的是这种新的竞争框架导致一种关键技术被加入到电话网络体系结构中。为了明确谁能做什么，美国被分成 164 个本地接入和传输区域（LATA，Local Access and Transport Areas）。很粗略地说，一个 LATA 和一个地区码覆盖的区域差不多一样大。在每个 LATA 内部，有一个本地交换运营商（LEC，Local Exchange Carrier），它垄断了该区域内的传统电话服务。全美国大约有 1500 个独立的电话公司以 LEC 的身份在运营。尽管某些 LATA 拥有上述电话公司中的一个或者多个，但最重要的 LEC 还是 BOC。

新特性在于所有 LATA 之间的流量由一种完全不同类型的公司来处理，这样的公司称为跨区运营商（IXC，IntereXchange Carrier）。最初，AT&T 长话公司是唯一正式的 IXC，但现在 IXC 行业也有诸如 Verizon 和 Sprint 这样的竞争对手。在分解 AT&T 公司时非常关注的一点是要确保所有的 IXC 在线路质量、关税制度以及客户拨打电话所需拨的号码位数等方面都能够同等对待。这样处理的方式如图 2-31 所示。在这个例子中我们看到有 3 个 LATA，每个 LATA 有几个端局。LATA2 和 LATA3 还有一个内含汇接局（即 LATA 内部的长途局）的小型层次结构。

任何一个 IXC，如果它愿意处理来自某个 LATA 的呼叫，则可在该 LATA 内建立一个称为**存点**（POP，Point of Presence）的交换局。LEC 负责将每个 IXC 连接到每个端局。这种连接可以是直接的，比如像图中的 LATA1 和 LATA3 那样；也可以是间接的，比如像图

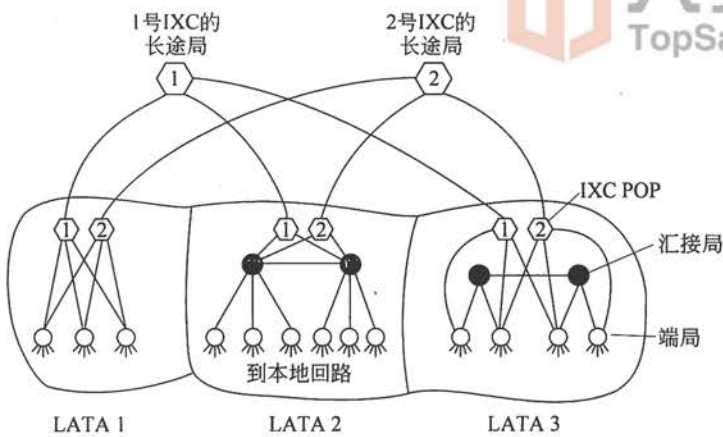


图 2-31 LATA、LEC 和 IXC 之间的关系
所有的圆圈表示 LEC 交换局。每个六边形属于它标号所示的 IXC

中的 LATA2 那样。而且，这种连接关系，无论从技术方面还是经济方面来看，对所有的 IXC 都必须一视同仁。通过这种方式，LATA1 中的电话用户可以选择任何一个 IXC 来呼叫 LATA3 中的用户。

作为 MFJ 的一部分，IXC 被禁止提供本地电话服务，而 LEC 则被禁止提供跨 LATA 的电话服务，但两者在任何其他业务范围内都不受限制，比如经营炸鸡餐馆。在 1984 年，这是一份毫无歧义的声明。不幸的是，技术的发展以一种非常有趣的方式使法律变得过时了。这份协定没有涉及有线电视和移动电话。随着有线电视从单向变成双向，以及移动电话变得越来越流行，LEC 和 IXC 都开始购买或者兼并有线电视和移动电话运营商。

到了 1995 年，美国国会看到那种试图维护各类公司之间业务界限的方法已经不再可行，于是拟订了一份草案，允许有线电视公司、本地电话公司、长途电话运营商和移动电话运营商进入彼此的业务领域。当时的想法是以后任何公司都可以为它的客户提供一个综合的数据包服务，其中包括有线电视、电话和信息服务，并且不同公司在服务和价格上应展开竞争。这份草案于 1996 年 2 月被正式批准，成为电信监管的一次重大改革。最终的结果是，有些 BOC 变成了 IXC，而一些其他公司（比如有线电视运营商）则开始提供本地电话服务与 LEC 进行竞争。

1996 法案的一个有趣属性是它要求 LEC 实现本地电话号码具有可携带特性。这意味着，一个客户无须改变本地电话号码就可改换到另一家本地电话公司。移动电话号码（和固定与移动线路之间）的可携性在 2003 年得到贯彻。这种灵活性消除了很多人的疑虑，使得他们更加倾向于改变 LEC。如此一来，美国电信业的面貌变得更具竞争力，其他国家纷纷开始仿效。通常其他国家会等待一段时间，看看这种实验在美国的效果怎么样。如果效果好的话，它们就会跟着仿照；如果效果不好，它们可以试一试其他的做法。

2.6.3 本地回路：调制解调器、ADSL 和光纤

现在到了开始详细学习电话系统如何工作的时候了。让我们从大多数人都熟悉的地方开始，即电话公司端局与家庭住宅之间的双线本地回路。本地回路常常称为“最后一英里”，

虽然其真正的长度可达数英里。本地回路运载模拟信息已有 100 多年的历史，由于转换成数字系统的高成本，它还有可能在未来的一段时间继续发挥着同样的作用。

许多努力一直致力于从已经部署的本地回路的铜介质上挤出数据网络。电话调制解调器在狭窄的信道上发送数字数据，而这些信道原本是被电话网络用来进行语音通话的。调制解调器曾经一度被广泛使用，但现在多数已经被诸如 ADSL 那样的宽带技术所取代。ADSL 重用了电话系统的本地回路，在其上把数字数据从客户端发送到端局，在那里它们被虹吸到 Internet。调制解调器和 ADSL 必须都能处理旧本地回路的一些限制：相对窄的带宽、信号衰减和失真，以及诸如串音等电气噪声的敏感性。

在一些地方，本地回路已经被光纤所取代，这些光纤被安装到家庭住宅（或非常接近的地方）。光纤是未来的发展方向。这些基础设施从根本上支撑着计算机网络，本地回路具有充足的带宽用于数据服务。有了光纤，限制因素变成了用户愿意支付什么，而不受制于本地回路的物理特性。

在本节，我们将学习本地回路，包括老式的和新型的两种。我们将涉及电话调制解调器、ADSL 以及光纤到户。

电话调制解调器

要在本地回路或任何其他物理信道上发送比特，必须把比特转换为可在信道上传输的模拟信号。用我们在上一节学习过的数字调制方法可以完成这种转换。在信道的另一端，模拟信号被还原成比特。

执行数字比特流和模拟信号流（代表这些数字比特）之间转换的设备称为**调制解调器**（modem），调制解调器是调制器（modulator）和解调器（demodulator）的缩写。调制解调器分为许多类型：电话调制解调器、DSL 调制解调器、有线电视调制解调器和无线调制解调器等。调制解调器可内置到计算机中（现在常见的电话调制解调器），或者用一个单独的盒子（常见的 DSL 调制解调器和有线电视调制解调器）。从逻辑上讲，调制解调器安装在（数字）计算机和（模拟）电话系统之间，如图 2-32 所示。

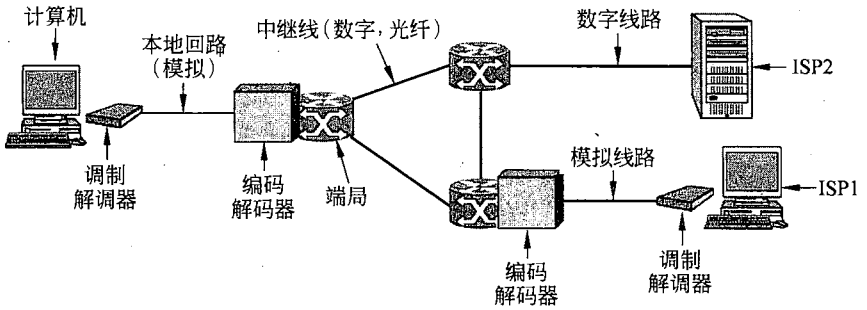


图 2-32 计算机-计算机呼叫中的模拟和数字传输。转换由调制解调器和编码解码器完成

两台计算机通过一条语音级电话线发送比特要用到调制解调器，此时的线路上通常充斥着语音交谈信号。但这么做存在一个主要困难：语音级电话线被限制在 3100 Hz 内。这一带宽虽然足以确保双方进行语音交谈，但相比以太网或 802.11（WiFi）所使用的带宽至少小了 4 个数量级。因而不出所料，电话调制解调器的数据传输速率也比以太网和 802.11 少了 4 个数量级。

让我们来算算为什么会这样。尼奎斯特定理告诉我们，即使是一个完美的 3000Hz 线路（电话线断然不可能是完美的），以快于 6000 波特的速度发送符号也没有什么意义。实际上，大多数调制解调器的发送速率是 2400 符号/秒或 2400 波特，而且它们关注的重点在于从每个符号中能获得多少个比特，同时还要允许两个方向同时发送（不同方向使用不同频率）。

低级的 2400 bps 调制解调器用 0 伏电压表示逻辑 0，1 伏电压表示逻辑 1，每个符号为 1 个比特。进一步地，它可以使用 4 种不同的符号，类似于 QPSK 的 4 个相位，从而以每个符号传 2 个比特的方式得到 4800 bps 的数据传输率。

随着技术的改进，数据速率已得到很大的提高。较高速率需要一组更大的符号或星座。由于有太多的符号，在探测幅度或相位时即使存在很少量的噪声都有可能导致一个错误。为了减少出错的机会，高速调制解调器标准采用了一些额外符号进行差错检测。这种方案称为**网格编码调制**（TCM，Trellis Coded Modulation）（Ungerboeck，1987）。

V.32 调制解调器标准使用了 32 星座点，每个符号传输 4 个数据比特和 1 个校验比特，以 2400 波特率获得具有差错检测能力的 9600 bps 数据率；比 9600 bps 高一级的是 14 400 bps，称为 V.32bis。它的波特率为 2400，每个符号传输 6 个数据比特和 1 个校验比特；然后是 V.34，每个符号传输 12 个数据比特，以 2400 的波特率能达到 28 800 bps。现在星座图拥有上千个点。该系列中的终极调制解调器是 V.34 bis，每个符号传输 14 个数据比特，以 2400 的波特率达到 33 600 bps 数据率。

至此，为什么停滞不前呢？对于标准的调制解调器，之所以停留在 33 600 源于电话系统受制于香农极限。该极限值大约为 35 kbps，这是根据本地回路的平均长度和这些线路质量推算出来的。高于这个速度将会违反物理定律（热力学）。

然而，有一个方法可以用来改变这种状况。在电话公司端局，数据被转换为数字形式，然后在电话网内传输（很久以前电话网核心就已从模拟转换成数字）。35 kbps 的限制指存在两个本地回路的情况，即每一端都有一条本地回路，这样在两端都会引入噪声。如果我们能去掉本地回路中的一条，就能提高信噪比，将最大速率提高一倍。

这种方法说明了为何 56 kbps 调制解调器能正常工作。通常来说，一端是一个 ISP，它从最近的端局获得高品质的数字内容。因此，当连接一端是高品质的信号（就像现在大多数 ISP 提供的），能获得的最大数据传输率可高达 70 kbps。两个家庭用户通过调制解调器和模拟线路连接，能获得的最高速率仍然只有 33.6 kbps。

使用 56 kbps 调制解调器（而不是 70 kbps 调制解调器）的究其原因是由尼奎斯特定理决定的。电话系统内的电话信道运载的是数字样本值。每个电话信道 4000 Hz 宽，包括保护带在内。因此重构信号所需要的采样数应该是每秒 8000 次。在美国每个样本值为 8 比特，其中一比特用于控制目的，所以每个用户数据能获得 56 000 bps 的数据速率。在欧洲，全部的 8 比特样本值都供用户使用，因此他们能使用 64 000 bps 的调制解调器，但国际协议标准选择了 56 000。

最终结果是 V.90 和 V.92 调制解调标准。它们分别提供了 56 kbps 的下行信道（ISP 到用户）及 33.6 kbps 和 48 kbps 的上行信道（用户到 ISP）。这种不对称的主要原因在于从 ISP 传输到用户的数据量比其他方式传输的数据量要多得多。这也意味着更多的有限带宽被分配给下行信道来增加它以 56 kbps 工作的机会。

数字用户线

当电话行业最后达到 56 kbps 时，它因很好地完成了一项任务而彻底放松了下来。与此同时，有线电视行业正在共享的电缆上提供高达 10 Mbps 速度的数据传输。随着 Internet 接入在日常业务中的重要性日益凸现，电话公司（LEC）开始意识到它们需要一种更具竞争力的产品。它们的答案就是利用本地回路提供新的数字服务。

刚开始时，许多相互重叠的服务都使用了数字用户线路（xDSL，Digital Subscriber Line）这样的统称，只是在 x 上有所不同。比标准电话服务具有更多带宽的服务有时称为宽带服务，尽管这个称谓的商业概念要高于特定的技术概念。后面我们将讨论这些服务中最流行的一种，即非对称数字用户线（ADSL，Asymmetric DSL）。我们还会使用术语 DSL 或者 xDSL 作为所有服务的简称。

调制解调器之所以如此慢的根本原因在于电话的发明是为了承载人类的语音，而且整个电话系统也是为了该目的而被小心地作了优化。而数据业务好像不受重视，一直没有得到很好的支持。每条本地回路在端局戛然而止，在那里有个滤波器把所有 300 Hz 以下、3400 Hz 以上的频率都削弱下去。截断处并不尖锐，300 Hz 和 3400 Hz 都是 3 dB 点。尽管两个 3 dB 点之间的宽度是 3100 Hz，但通常这一频段被当作 4000 Hz 来引用。所以，数据被限制在这一狭窄的频段中。

xDSL 得以工作的诀窍在于当一名客户预订了这项服务时，入境线路被连接到了另一种不同的交换机上，这种交换机没有上述滤波器，因而可以充分发挥本地回路的全部承载能力。于是，通信能力受到限制的因素就变成了本地回路的物理特性（粗略支持 1 MHz），而不再是由滤波器引入的 3100 Hz 带宽限制。

不幸的是，本地回路的容量随着用户住宅与端局之间距离的增大而快速下降，因为信号沿线路的衰减也随着距离的增加而相应地递增。本地回路的容量还依赖于双绞线的粗细和综合质量。图 2-33 给出了潜在带宽与距离的函数关系图示。该图假设所有关于本地回路的其他因素都是最佳的（新的线路、适当的捆扎等）。

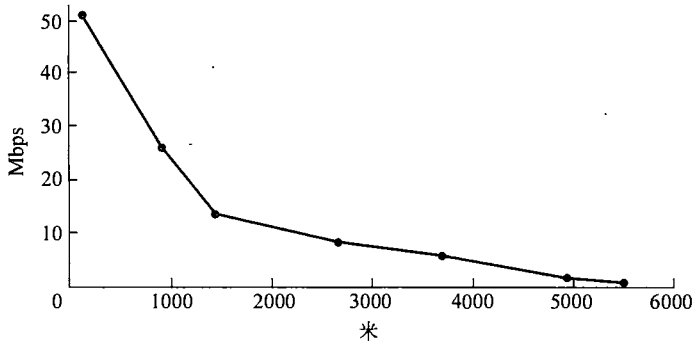


图 2-33 DSL 所用的 3 类 UTP 的带宽与距离

该图对于电话公司来说隐含着一个问题。当电话公司承诺提供某一个速度的传输率时，它同时也划定了一个以端局为中心的圆，超出了这个圆的范围就无法提供这样的服务。这意味着，当远距离的客户试图申请这项服务，电话公司会告诉他们“谢谢你对此感兴趣，但你住的地方离提供该项服务的端局还差 100 多米远，你能搬得近点吗？”。数据速率越低，

服务区域的半径就越大，从而覆盖面越广。但是，速度越低，这项服务的吸引力就越小，愿意付钱购买这项服务的人也就越少。这正是商业与技术的巅峰对决所在。

所有的 xDSL 服务都有特定的设计目标。第一，服务必须在现有的本地回路 3 类双绞线上工作。第二，它们不能影响客户原来的电话和传真业务。第三，它们必须比 56 kbps 快。第四，这些服务应该总是可用的，按月租方式收费而不是按每分钟收费。

为了满足技术目标，本地回路上的 1.1 MHz 频谱被分成 256 条独立的信道，每条信道宽 4312.5 Hz。这样的安排如图 2-34 所示。我们在前一节所学的 OFDM 编码方案可用在这些信道上发送数据，尽管在 ADSL 文档中它经常称为**离散多音**（DMT，Discrete MultiTone）。信道 0 用于**简单老式电话服务**（POTS，Plain Old Telephone Service）。信道 1～5 空闲，目的是防止语音信号与数据信号相互干扰。在剩下的 250 条信道中，一条用于上行流控制，另一条用于下行流控制，其他的信道全部用于用户数据。

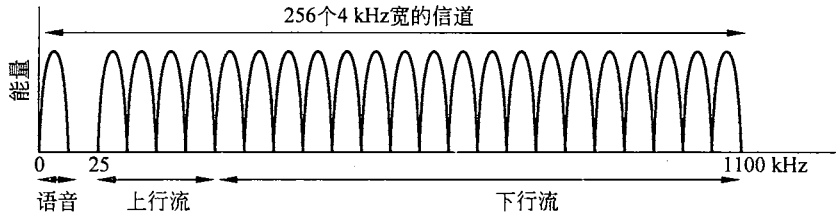


图 2-34 使用离散多音调制方法的 ADSL 操作

原理上，剩下的每条信道都可以被用作全双工数据流，但是谐波、串音和其他影响使得实际系统的性能大大低于理论限制。服务提供商决定了多少信道用于上行数据流，多少信道用于下行数据流。上行和下行各占 50%的做法在技术上是可行的，但是大多数提供商倾向于将 80%～90%的带宽分配给下行信道，因为大多数用户的下载数据量超过上载数据量。这种选择正好暗示了 ADSL 中的第一个字母 A（非对称）。一种常见的分法是 32 条信道用于上行数据流，其余的用于下行数据流。还有一种可能的做法是让最高端的一些上行信道成为双向信道，以便增加带宽。但是这种优化要求增加一部分特殊的电路来消除回声。

ADSL 的国际标准于 1999 年获得批准，称为 G.dmt。它允许高达 8 Mbps 的下行速度和 1 Mbps 的上行速度。这个标准已经被 2002 年发布的 ADSL2 超越，下行速度经过不断的改进，目前已经可以达到 12 Mbps，上行速度仍然是 1 Mbps。现在，我们又有了 ADSL2+，它使用双绞线上的双倍带宽（2.2 MHz）把下行速度翻了一倍，达到 24 Mbps。

但是，这里所引用的数字是针对离端局比较近（1～2 千米内）的良好线路上的最快速度。实际上很少有线路支持这样的速度，也很少有提供商提供这样的速度。通常情况下，提供商提供类似 1 Mbps 的下行和 256 kbps 的上行（标准服务）、4 Mbps 的下行和 1 Mbps 的上行（改进服务），以及 8 Mbps 的下行和 2 Mbps 上行（高级服务）。

在每条信道内使用了 QAM 调制方案，速率约为 4000 符号/秒。每条信道的线路质量被时刻监控，必要时通过一个更大或者更小的星座图来调整数据速率，就像图 2-23 那样。不同的信道可能有不同的数据传输率，具有高 SNR 的信道发送的每个符号可以携带多达 15 个比特，而对于较低 SNR 的信道，发送的每个符号所携带的比特个数可以降到 2、1，甚至没有比特。

典型的 ADSL 部署结构如图 2-35 所示。在这种方案中，电话公司的技术人员必须在客

户住所安装一个**网络接口设备**（NID，Network Interface Device）。这个小的塑料盒代表了电话公司财产的终结和客户财产的开始。靠近NID(有时候组合在一起)是一个分离器(splitter)，分离器是一个模拟滤波器，它将POTS使用的0～4000 Hz频段与数据分开。POTS信号被路由到已有的电话机或者传真机，而数据信号则被路由到ADSL调制解调器，该调制解调器使用数字信号处理器来实现OFDM。由于当前大多数的ADSL调制解调器都是外置的，所以计算机必须通过高速方式与它相连。通常的做法是，使用以太网、USB电缆或者802.11。

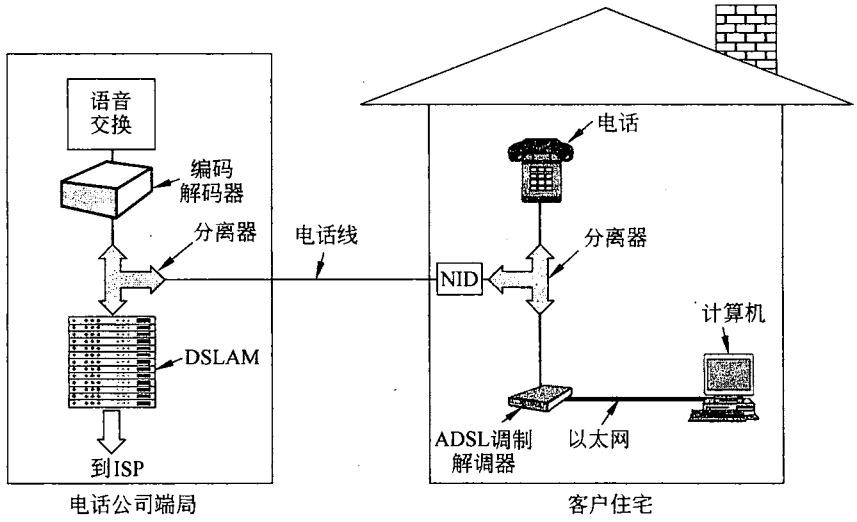


图2-35 一种典型的ADSL设备配置

在线路的另一头（即电话公司端局）也要安装一个对应的分离器。在这里，信号中的语音部分被过滤出来后送到正常的语音交换机中。频率在26 kHz以上的信号则被路由到一种新设备中，这种设备称为**数用用户线路接入复用器**（DSLAM，Digital Subscriber Line Access Multiplexer)。该设备包含一个数字信号处理器，与ADSL调制解调器中的一样。一旦从信号中恢复出比特，就可以据此构造出数据包，并将数据包发送给ISP。

这种语音系统和ADSL的完全分离使得电话公司部署ADSL相对比较容易。电话公司所需要做的只是购买一个DSLAM和分离器，并且将ADSL用户接到分离器上。其他高带宽的服务（比如ISDN）要求对现有的交换设备作更大的改动。

图2-35所示的设计方案有一个缺点，那就是需要在客户住处安装NID和分离器。这些设备只能由电话公司的技术人员来安装，从而产生昂贵的“上门服务”费用（即派遣一个技术人员到客户住处)。因此，另外一种无分离器的设计已经被标准化，它的非正式名称为G.lite。它的部署结构与图2-35一样，只不过在客户住所不再需要分离器，仍然使用现有的电话线。唯一的区别是在每个电话与电话线之间或者ADSL调制解调器与电话线之间的电话插孔中装入一个微滤波器。电话的微滤波器是一个低通滤波器，它消除了3400 Hz以上的频率成分；ADSL调制解调器的微滤波器是一个高通滤波器，它消除了26 kHz以下的频率成分。然而，这个系统的可靠性不如带分离器的系统，所以G.lite的实际使用至多只能达到1.5 Mbps（相比之下，带分离器的ADSL可以达到8 Mbps)。有关ADSL的更多信息，请参考（Starr，2003)。

光纤到户

已部署的铜质本地回路限制了 ADSL 和电话调制解调器的性能。为了让它们运行得更快，网络服务更好，电话公司正在利用每个机会逐步升级本地回路，把光纤一路拉到家庭住宅和办公楼宇。这种本地回路的改进结果就是所谓的**光纤到户**（FttH，Fiber to the Home）。尽管光纤到户技术已经有一段时间可用，但真正开始部署却在 2005 年。那时使用 DSL 和有线电视电缆的用户想要从 Internet 下载影片，从而产生了高速访问 Internet 的需求。在美国，约 4%的住宅现在通过光纤接入 FttH，Internet 接入速度超过 100Mbps。

FttX 形式（其中 X 代表地下室、马路边石或街区）有好几种，引起人们注意的是光纤部署尽可能地接近住宅。在这种情况下，铜（双绞线或同轴电缆）在最后一段短距离内提供了足够快的速度。因此把光纤铺到哪里的选择是一个经济问题，必须在成本与预期收益之间取得平衡。无论如何，光纤已经跨越了传统的“最后一英里”障碍。我们在下面的讨论中重点关注 FttH。

如同之前的铜线一样，光纤本地回路也是无源的。这意味着不需要供电设备来放大或处理信号。光纤只是简单地运送住宅和端局之间的信号。这反过来又降低了成本，提高了可靠性。

一般来说，住宅里的光纤被捆绑在一起，用一根光纤连接到端局，每根光纤包括一组大约 100 个住户。在下行流方向，光分离器（splitters）把从端局传来的信号分离开，使得它们能到达所有的住户。如果只允许一家对信号进行解码，则出于安全的考虑，必须对信号进行加密。在上行流方向，光合并器（combiner）把来自各个住户的信号合并成一个信号发送给端局。

这种体系结构称为**无源光网络**（PON，Passive Optical Network），如图所示 2-36。它通常使用一个波长被所有住户共享，用作下行流的传输；而使用另一个波长被所有住户共享，用作上行流的传输。

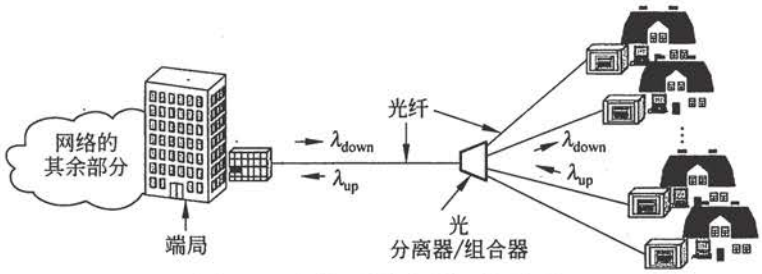


图 2-36 用作光纤到户的无源光网

即使信号被分离，光纤具有的巨大带宽和低衰减意味着 PON 可为 20 千米范围内的用户提供高数据速率的传输。实际的数据速率和其他细节取决于 PON 的类型。比较常见的 PON 有两种。**千兆级 PON**（GPONs，Gigabit-capable PONs）来自电信世界，由 ITU 标准定义。**以太 PON**（EPONs，Ethernet PONs）更倾向于 Internet 世界，因而由 IEEE 标准定义。两者都以千兆速率运行，并可传递不同服务的流量，包括 Internet、视频和音频。例如，GPON 提供了 2.4 Gbps 的下行速率和 1.2 Gbps 或 2.4 Gbps 的上行速率。

为了在不同住户之间共享端局的单根光纤容量，需要某种协议的支持。下行流方向比

较容易，端局可以按照任何它喜欢的次序将消息发送到每个不同的住户。然而，在上行流方向，不同住户却不能同时发送消息，否则来自不同住户的信号会发生冲突。这些住户听不到其他住户的传输，因此它们在传输前无法进行侦听。对此的解决方案是住户的某个设备在发送前先请求，获得由端局设备分配的时间槽，然后在该时间槽内发送。为了这项工作能正常进行，有个测距过程对住户的传输时间进行调整，以便端局接收到的信号能够同步。这种设计类似于有线电视调制解调器，我们将在本章后面部分讨论。欲了解更多有关PON的未来信息，请参阅（Grobe 和 Elbers，2008）。

2.6.4 中继线和多路复用

电话网络中的中继线不仅比本地回路快得多，而且在其他两个方面与本地回路也有所不同。首先，电话网络核心传送的是数字信息而不是模拟信息，即传的是比特而不是声音。为了在长途中继线上传输数字形式，需要在端局对信号进行一次转换。其次，中继线上同时正进行着数以千计，甚至上百万的电话呼叫。这种多个电话共享一根中继线的方式对于实现规模经济非常重要，因为在两个电话端局之间安装和维护高带宽中继线所需要的基本开销和安装并维护一个低带宽中继线所需的费用差不多。高带宽中继线的共享可通过TDM和FDM多路复用方法实现。

下面我们简要介绍如何将语音信号数字化，使得它们可以通过电话网络传输。之后，我们将看到TDM是如何在中继线上运送比特的，包括光纤（SONET）采用的TDM系统。然后，我们将目光转向适用于光纤的FDM，就是所谓的波分多路复用。

数字化语音信号

在电话网络发展的早期，核心系统将语音呼叫作为模拟信息来对待并处理。许多年来人们一直利用FDM技术把4000 Hz的语音信道（3100 Hz加上保卫带）多路复用到越来越大的单位。例如，60～108 kHz波段的12个电话呼叫组成一组，称为**群**（group）组，5个群（共60个电话呼叫）再组成一组，称为**超群**（supergroup）等。这些FDM方法仍然被用在铜线和微波信道。然而，FDM需要模拟电路，而且并不适合计算机的处理。与此相反，TDM完全通过数字电子来处理，因此近年来得到了更为普遍的应用。由于TDM只能用于数字数据传输而本地回路产生的又是模拟信号，因此在端局必须把模拟信号转换成数字信号，所有个人本地回路被组合在一起发送到出境中继线上。

在端局，把模拟信号数字化的工作由一个称为**编码解码器**（codec，coder-decoder）的设备完成。编码解码器每秒采集8000个样值（125微秒/样值），根据尼奎斯特定理，这个采样率足以捕捉一切来自4 kHz电话信道带宽上的信息。若采样率较低，信息就会被丢失；若采样率较高，也得不到更多的信息。每个信号的样值幅度被量化成一个8比特的数字。

这种技术就是**脉冲编码调制**（PCM，Pulse Code Modulation），它构成了现代电话系统的核心。因此，几乎电话系统内的所有时间间隔均为125微秒的倍数。也正是因为这个原因，语音级电话呼叫的标准化未压缩数据率是每125微秒8比特，或64 kbps。

在电话呼叫的另一端，必须从随时传来的量化样值中重新生成模拟信号（并且对此进行平滑处理）。即使遵照尼奎斯特定理采样，由于样值已经进行了量化处理，所以还原出来

的模拟信号不会和原始模拟信号完全相同。为了减少由于量化带来的误差，量化级别被设置成不均匀状。使用对数尺度，让相对较小的信号幅度用更多的比特表示，而相对较大的信号幅度用较少的比特表示。这种非均匀量化后的量化错误与信号幅度成比例。

广泛应用的量化版本有两个：μ-规则（μ-law）和 A-规则（A-law）。前者被广泛用在北美和日本地区，后者被应用在欧洲和世界其他各地地区。这两个版本都遵守由 ITU G.711 规定的标准。整个处理过程可以等效地想象成：信号的动态范围（或者信号可能的最大值和可能的最小值之间的比例）在（均匀）量化之前被压缩了，然后在模拟信号被还原时再被扩展恢复。基于这个原因，它称为**压缩扩展**（companding）。对这个数字化后的样值还可以进一步压缩，以便在带宽小于 64 kbps 的信道上传输。不过，我们把这个主题留待探索音频应用（比如 IP 电话）时再讨论。

时分多路复用

基于 PCM 的 TDM 可在中继线上运送多路电话语音，每 125 微秒为每路电话发送一个语音样值。在数字传输作为一个可行技术出现时，国际电信联盟（ITU，当时称为 CCITT）未能就一项 PCM 国际标准达成协议。因此，现在全世界不同国家使用的各种 PCM 模式都不兼容。

北美和日本地区使用的是 T1 载波（T1 carrier），如图 2-37 所示（从技术上而言，这种格式称为 DS1，而载波称为 T1，但是按照广泛的行业传统，我们这里不作细分）。T1 载波包含 24 条被复用在一起的语音信道，每个信道依次将 8 比特的样值插入到输出流中。

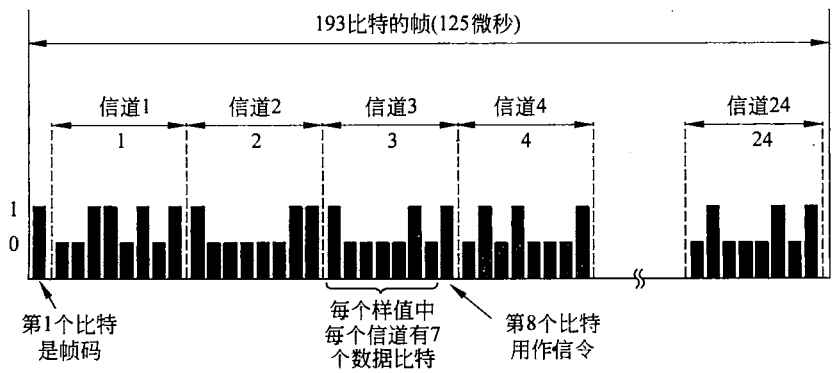


图 2-37 T1 载波（1.544 Mbps）

每帧包含 24×8=192 个比特，再加上额外一个比特用于控制，因而每 125 微秒产生 193 个比特。这样得到的数据传输率为 1.544 Mbps，其中 8 kbps 用于信令控制。第 193 个比特用于帧同步和信令。在一种基于该基本方法的变异情况下，第 193 个比特的使用涉及群内所有 24 帧，称为**扩展超帧**（extended superframe）。分布在第 4、8、12、16、20、24 个帧相应位置上的 6 个比特取交替模式 001011……通常情况下，接收方必须不断地检查这个固定模式，以确保自己没有失去同步。然后再使用 6 个比特的差错交验码来帮助接收方确定自己是否保持同步。如果接收方失去了同步，则可以搜索上面的比特模式，同时利用差错校验码来重新获得同步。其余的 12 个比特则用于网络操作和维护的控制信息，比如向远端报告性能。

T1 格式有多种变化。较早版本在**带内**（in-band）发送信令信息，这意味着在相同的数据信道内使用了一些数据比特来发送控制信息。这种设计是**信道相关信令**（channel-associated signaling）的一种形式，因为每个信道都有自己专用的信令子信道。分配信令子信道的一种方法是：将每 6 个帧中每个信道 8 比特样值中的最低有效位拿出来。该方法有个很形象的名称——**强占比特信令**（robbed-bit signaling）。这种想法来自于从语音通话中偷出几位无关紧要，没有人会感觉得到。

然后，对于数据则完全是另一回事。至少可以这样说，传送错误比特毫无用处。如果用 T1 旧版本来传输数据，则 24 个信道上 8 比特中的 7 比特可用来传数据，速率为 56 kbps。相反地，T1 新版本提供了纯信道，在其上通信时所有的比特都可被用来发送数据。纯信道是企业租用 T1 线路所要求的，这些企业希望通过传送语音的电话网络发送数据。任何语音呼叫的控制信令由**带外**（out-of-band）处理，这意味着在数据信道之外还存在着一条独立的信道。通常情况下，信令是通过**公共信道信令**（common-channel signaling）来完成的，其中有一个共享的信令信道。24 个信道中的一个信道可用于这一目的。

在北美及日本以外地区，使用 2.048 Mbps 的 E1 载波代替 T1 载波。E1 载波有 32 个 8 比特数据样值被封装在 125 微秒的帧中。32 个信道中的 30 个用于传输信息，其余 2 个信道用作信令。每四个帧为一组，提供了 64 个信令比特，其中一半用于信令（是否信道相关或共同信道），另一半用于帧同步或各个国家保留使用。

时分多路复用允许将多个 T1 载波复用到一个更高阶的载波中。图 2-38 显示了一种可能的做法。在左侧，我们看到 4 条 T1 信道被复用到一条 T2 信道中。T2 以及更高级的复用是按位完成的，而 24 条语音信道构成的一个 T1 帧的复用则是按字节完成的。4 个 1.544 Mbps 的 T1 流应该产生 6.176 Mbps，但 T2 实际上是 6.312 Mbps。这些多出来的比特用于成帧，或者当载波失去同步时的恢复。T1 和 T3 广泛用于顾客，而 T2 和 T4 仅用在电话系统内部，所以它们较少为人所知。

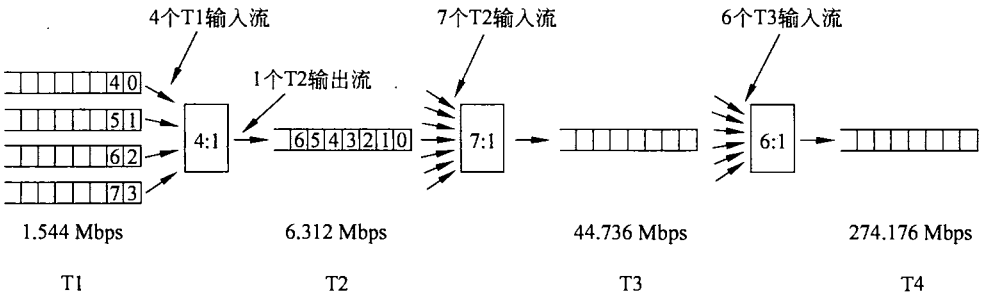


图 2-38 T1 载波被复用到更高级的载波中

在下一个级别上，7 个 T2 流按比特组合起来形成一个 T3 流。然后，6 个 T3 流又组合起来形成一个 T4 流。在每一步的组合过程中，都会添加少量的开销用于成帧，或者当发送方和接收方失去同步时的恢复。

正如美国和其他国家或地区在**基本传输载波**（basic carrier）上没有达成共识一样，类似地，在如何多路复用得到更高带宽载波的做法上也同样不一致。美国按照 4、7 和 6 逐步升级的方案并没有吸引别人也跟着这样做，所以 ITU 标准要求每一级都是将 4 个流复用到一个流中。而且，在美国和 ITU 标准中关于如何成帧、如何恢复数据的规定也不同。ITU

针对32、128、512、2048和8192信道的层次结构运行速度分别是2.048、8.848、34.304、139.264和565.148 Mbps。

SONET/SDH

在光纤早期阶段，每个电话公司都有自己专用的光纤TDM系统。当AT&T公司于1984年被分解以后，本地电话公司必须连接到多家长途电话运营商，而这些运营商各有各的光纤TDM系统，所以标准化的需求变得非常迫切。1985年，RBOC的研究机构Bellcore开始制定这样的标准，称为**同步光网络**（SONET，Synchronous Optical NETwork）。

后来，ITU也加入进来一起工作，从而于1989年产生了一个SONET标准以及一组并行的ITU建议（G.707、G.708和G.709）。这些ITU建议称为**同步数字系列**（SDH，Synchronous Digital Hierarchy），它们仅仅在一些很小的方面不同于SONET。在美国几乎所有的长途电话中继线在物理层都运行SONET，其他地方的许多中继线也是如此。要想了解更多关于SONET的信息，请参考（Bellamy，2000；Goralski，2000；Shepard，2001）。

SONET设计有4个主要目标。第一个目标也是最重要的，SONET必须使不同的运营商可以协同工作。为了达到这个目标，必须定义一个公共的信令标准，其中涉及波长、时序、帧结构以及其他的问题。

第二个目标，需要一种用来统一美国、欧洲和日本数字系统的方法。这些系统都基于64 kbps的PCM信道，但是，组合信道的方式却各不相同，而且互不兼容。

第三个目标，SONET必须提供一种办法来复用多条数字信道。在设计SONET时，美国广泛使用的最高速度的数字线路实际上是T3，其速度为44.736 Mbps。T4已经被定义，但尚未广泛使用，而且高于T4速度以上的载波还没有任何定义。SONET的部分使命就是继续推进这样的速度层次，目标是达到Gbps，甚至更高速度。而且，将多条慢速信道复用到一条SONET信道中的标准方法也是必要的。

第四个目标，SONET必须支持操作（operation）、管理（administration）和维护（maintenance），即OAM。这是管理网络必须涉及的几个方面，以前的系统在这方面做得并不好。

早期的决定是将SONET设计成一个传统的TDM系统，光纤的全部带宽都分配给一条信道使用，再将这条信道的时间槽划分给不同的子信道。按照这种设计理念，SONET就是一个同步系统。每个发送方和接收方都必须绑定到一个公共时钟。系统由一个主时钟控制，时钟精度大约为$1/10^9$。SONET线路上的比特按照由主时钟控制的极为精确的时间间隔发送出去。

基本SONET帧是每隔125微秒发送长为810个字节的数据块。由于SONET是同步系统，所以，不管是否有任何有用的数据需要发送，这个帧都要被发送出去。每秒8000帧的速率正好符合所有数字电话系统中使用的PCM信道的采样率。

810字节的SONET帧最好描述成具有90列宽、9行高的矩形。因此，每秒钟传输8000次，每次8×810=6480比特，总的数据传输率为51.84 Mbps。这是基本的SONET信道，称为**同步传输信号-1**（STS-1，Synchronous Transport Signal-1）。所有的SONET中继线都是STS-1的倍数。

每一帧的前三列保留，用于传输系统管理信息，如图2-39所示。其中，前三行包含段

（section）的开销；接下来的 6 行包含线路（line）开销。段开销的生成和校验在每一段的开始和结束时进行，而线路开销则是在每条线路开始和结束时生成和校验。

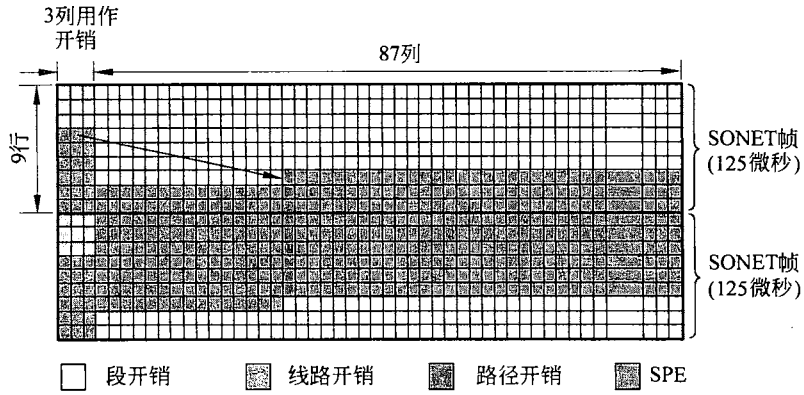


图 2-39 两个连续的 SONET 帧

SONET 发送器连续地发送 810 个字节的帧，帧之间没有任何间隙，即使没有数据需要发送时也要发送（这种情况可以看作发送空数据帧）。站在接收方的角度，它所看到的是一个连续的比特流，那么它如何知道每一帧从哪里开始呢？答案在于每一帧的前两个字节有一个固定的模式，接收方搜索这种模式就可以定位帧的起始。如果接收方在大量的连续帧中相同的位置发现了这种模式，则它假定自己已经与发送方同步。理论上讲，一个用户可以按照常规方式在有效载荷中插入这种模式；但实际上，由于多个用户的数据被复用到了同一帧中，以及其他一些原因，用户这样做是不行的。

每帧中剩下的 87 列包含了 87×9×8×8000=50.112 Mbps 的用户数据。这些用户数据可以是语音样值（全部通过 T1 和其他载波传送）或者数据包。承载用户数据的**同步有效载荷信封**（SPE，Synchronous Payload Envelope）并不总是从第 1 行、第 4 列开始。SPE 可以从帧内的任何一个地方开始。线路开销中的第一行包含了一个指针，它指向 SPE 的第一个字节。SPE 的第一列是路径开销（即端到端路径子层协议的头）。

SONET 允许 SPE 从帧内任何一个地方开始，甚至可以跨越两帧（如图 2-39 所示），这种能力增加了系统的灵活性。例如，当源端正在构造 SONET 空帧时来了一个有效载荷数据，它就可以将这些数据插入到当前帧中，而不用等到下一帧的开始。

SONET 的多路复用层次如图 2-40 所示。从 STS-1 到 STS-768 的数据传输率已经被定义，范围在 T3 到 40Gbps 之间。随着时间的推移，肯定还会定义出甚至更高的速率，160 Gbps 的 OC-3072 在技术上成为可行时将是下一个线路标准。对应于 STS-n 的光纤载波称为 OC-n，其复用也是按比特进行的，除了因同步所需对特定比特进行重新排序外，其他的比

SONET		SDH	数据率(Mbps)		
电子	光	光	总速率	SPE	用户速率
STS-1	OC-1		51.84	50.112	49.536
STS-3	OC-3	STM-1	155.52	150.336	148.608
STS-12	OC-12	STM-46	22.08	601.344	594.432
STS-48	OC-48	STM-16	2488.32	2405.376	2377.728
STS-192	OC-192	STM-64	9953.28	9621.504	9510.912
STS-768	OC-768	STM-256	39813.12	38486.016	38043.648

图 2-40 SONET 和 SDH 多路复用率

特不变。SDH名称有所不同，它们从OC-3开始，因为基于ITU的系统没有接近于51.84 Mbps速率的标准。我们显示了两者的共同速率，它们从OC-3开始并以4的倍数递增。总数据率要包括所有的开销。SPE数据传输速率不包括线路和段的开销。用户数据速率不包括所有开销，并只计算87列的有效载荷。

顺便提一下，如果一个载波（比如OC-3）没有被复用，而是仅承载了来自单个源的数据，则在线路名称后面加一个字母c（表示级联）。因此，OC-3表示由三条独立的OC-1载波构成的一条155.52 Mbps载波，而OC-3c则表示来自于单个源的155.52 Mbps数据流。一条OC-3c流内的三个OC-1流按列交替插入，首先在第1列中依次插入第1个流、第2个流和第3个流，然后在第2列中依次插入第1个流、第2个流和第3个流，以此类推，最后形成的帧包括270列宽、9行高。

波分多路复用

频分多路复用的一种形式和TDM一样，都是利用了光纤信道的巨大带宽。这种形式称为**波分多路复用**（WDM，Wavelength Division Multiplexing）。光纤WDM的基本原理如图2-41所示。这里，4条光纤汇合到一个光纤组合器（combiner）中，每条光纤的能量位于不同的波长处。四束光波被组合到一条共享的光纤上，然后传输给远处的接收方。在远端，这束光又被分离到与输入端一样多的光纤上。每条输出光纤包含一个短的、特殊结构的核，该核能过滤掉其他所有波长而仅留下某一个波长。然后，结果信号被路由到它们的目的地，或者以其他不同的方式组合起来以便再次复用传输。

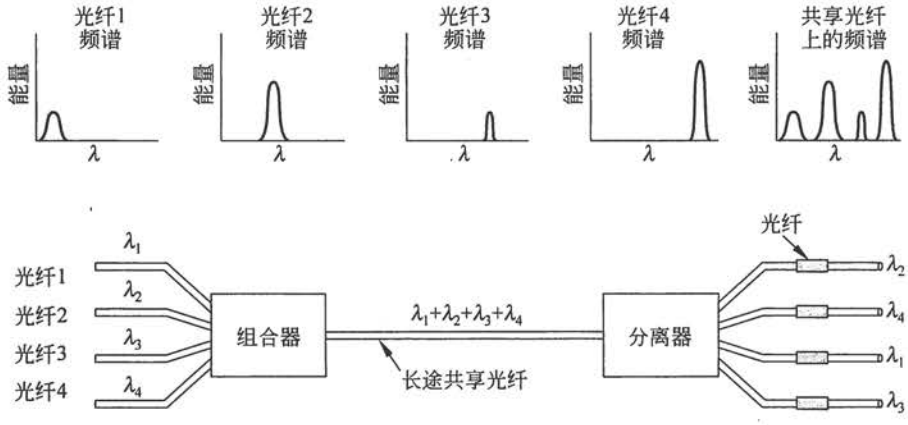


图2-41 波分多路复用

这里并没有真正的新技术，只不过是在极高频率上的频分多路复用。术语WDM源于对光纤信道的描述，采用了它们的波长或者“颜色”，而非频率。只要每条信道有它自己的频率（即波长）范围，并且所有的频率范围都是分开的，那么，它们就可以被复用到长途光纤上。与电子FDM之间的唯一区别在于使用衍射光栅的光纤系统是完全无源的，因此可靠性极高。

WDM得以流行的理由是单信道上的能量通常只有几个GHz宽，这是由目前电信号和光信号之间转换的最快速度所限制的。在同一根光纤上以不同的波长并行运行多个信道，聚合后的带宽随信道数量的增加而线性递增。由于单根光纤的带宽大约为25 000 GHz（见

图 2-7)，所以理论上来说，即使在 1 比特/Hz 的情况下(更高的速率也有可能)，仍然有 2500 个 10 Gbps 信道的发展空间。

WDM 技术已经发展到了一种让计算机技术望尘莫及的程度。WDM 是在 1990 年前后发明的。第一个商业系统有 8 条信道，每条信道的带宽为 2.5 Gbps。到 1998 年，市场上就已经出现 40 条信道的系统了（每条信道的带宽为 2.5 Gbps)。到 2006 年，具有 192 个信道（每条信道的带宽为 10 Gbps）和 64 个信道（每条信道的带宽为 40 Gbps）的产品问世，这样的系统具有 2.56 Tbps 的能力。如此高的带宽足够每秒钟传输 80 部完整的 DVD 电影。多达 200 个或 100 个信道紧紧地挤在光纤上，每个至少具有 50 GHz。公司在口头宣传后展示的技术表明，在实验室内光纤已经能拥有 10 倍的传输能力，但从实验室走向市场通常至少需要几年的时间。当信道的数目很大，并且波长的间隔非常接近，这样的系统通常称为**密集波分多路复用**（DWDM，Dense WDM)。

波分复用技术的驱动力之一是全光元件的发展。以前，每隔 100 千米必须把光纤上的全部信道分离开来，把每个信道的光信号转换成电子信号做单独的放大处理，然后再把这些电子信号重新转换成光信号，合并后再发送出去。如今，全光放大器可每隔 1000 千米重新生成整个信号，无须进行多次的光电转换。

在 2-41 例子中，我们有一个固定波长系统。从光纤 1 输入的比特被输出到光纤 3，从光纤 2 输入的比特被输出到光纤 1，等等。然而，也有可能构建一个 WDM 系统，它在光域内切换。在这种装置中，采用 Fabry-Perot 或 Mach-Zehnder 干涉仪的输出滤波器是可调的。这些设备允许由一台控制计算机来动态地改变被选频率。这种频率可调能力为系统带来了很大的灵活性，可在由一组固定光纤组成的电话网络上提供大量不同波长的路径。如需详细的有关光网络和 WDM 信息，请参阅（Ramaswami 等，2009 年)。

2.6.5 交换

站在普通电话工程师的角度，电话系统分为两个基本部分：局外部分（本地回路和中继线，因为从物理位置来看，看它们都位于交换局外部）和局内部分（交换机，它们在交换局内部)。我们刚才看过了局外部分，现在该看一看局内部分了。

当前，电话系统中用到了两种不同的交换技术：电路交换和数据包交换。传统的电话系统基于电路交换技术，但随着 IP 技术之上的语音通信兴起，数据包交换已经取得了长足的进步。下面我们将详细地讨论电路交换，并将它与数据包交换作比较。这两类交换技术是如此之重要，我们在网络层还将继续对此进行讨论。

电路交换

概念上讲，当你或者你的计算机发出一个电话呼叫时，电话系统的交换设备就会全力以赴地寻找一条从你电话通向接收方电话的物理路径。这项技术就是所谓的**电路交换**(circuit switching)，其过程如图 2-42（a）所示。其中 6 个矩形代表了电话运营商的交换局(端局、长途局等)。在这个例子中，每个局有 3 条入境线路和 3 条出境线路。当电话呼叫通过一个交换局时，在电话入境线路与某一条出境线路之间就会建立起一个物理连接（概念上)，如图中的虚线所示。

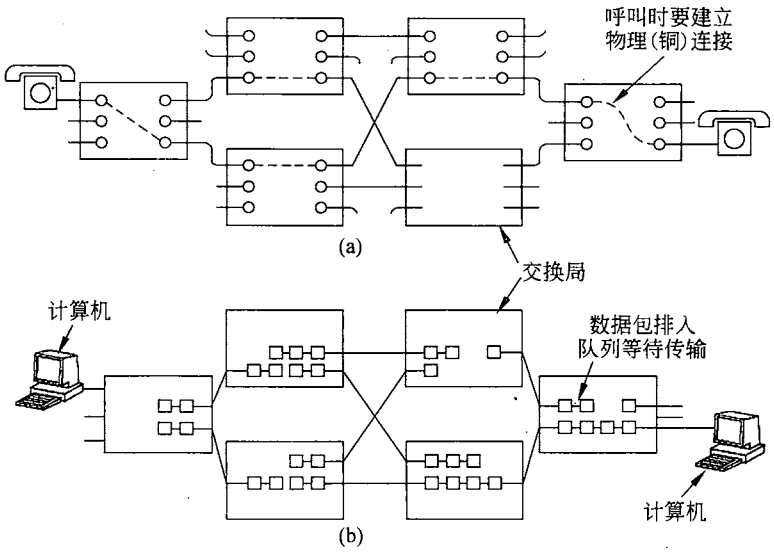


图 2-42
（a）电路交换；（b）数据包交换

在电话业的早期，接线员在输入插槽和输出插槽中插入一段跳接电缆，就完成了相应的连接建立过程。实际上，在发明自动电路交换设备的过程中有一个多少有点令人惊讶的小故事。自动化的电路交换设备是由19世纪密苏里州的一位殡葬工Almon B. Strowger发明的。在电话发明之后不久，每当有人去世后，死者的亲属就会呼叫镇上的接线员，并说“请替我接殡葬工”。对Strowger先生来说，非常不幸的是这个镇上一共有两个殡葬工，而电话接线员正好是另一位殡葬工的妻子。因此，他很快明白要么去发明一个自动化的电话交换设备，要么他就得失业。Strowger先生选择了前一条路。将近100年来，全世界使用的电路交换设备都称为Strowger装置（历史并没有记载当时那个失业了的电话接线员是否获得了信息操作员的工作，每天在回答诸如“殡葬工的电话号码是多少”之类的问题）。

当然，图2-42（a）显示的模型被高度简化了。因为事实上，两个电话之间的物理路径可能部分是微波，部分是光纤。而且在光纤链路上，会有成千上万的电话呼叫被复用在一起。然而，基本思路仍然是有效的：一旦一个呼叫被建立起来，在两端之间就会存在一条专用的路径，并且这条路径会一直持续到该次呼叫结束。

电路交换的一个重要特点是在发送数据之前需要建立一条端到端的路径。从拨完号码到开始响铃，这段时间可能需要10秒钟，长途电话或者国际电话所需要的时间更长。在这段时间中，电话系统正全力以赴寻找到达目的地的路径，如图2-43（a）所示。注意，在开始传输数据之前，呼叫请求信号必须一路传向接收方，并且要被接收方确认。对于许多计算机应用来说（比如销售点的信用卡验证），长时间的建立过程是不可取的。

在电话呼叫双方之间保留路径带来的结果是：一旦完成连接的建立，那么数据传输的唯一延迟是电磁信号的传播时间，每1000千米大约需5毫秒。建立一条路径的另一个结果是不存在拥塞的危险，也就是说，一旦电话呼叫被接通就永远不会再听到忙音。当然，在建立连接之前，由于交换能力或者中继线传输能力的不足，可能会听到忙信号。

包交换

替代电路交换的一个方案是包交换，如图 2-43（b）所示，第 1 章对此已经有所描述。有了这项技术，数据包尽可能快地被发出，这里无须像电路交换那样要事先设立一条专门的路径。路由器使用存储-转发传输技术，把经过它的每个数据包发送到通往该包目的地的路径上。这个转发过程与电路交换不同。在电路交换中，连接的建立过程预留了从发送端到接收端一路上的带宽资源，该条电路上的所有数据将走相同的路径。另一方面，让所有的数据遵循同样的路径意味着它们到达接收端的秩序不可能出现混乱。而在数据包交换中，没有固定的路径，不同的数据包可以走不同的路径，路经的选择取决于它们被传输时的网络状况，所以它们到达接收端的秩序可能出现混乱。

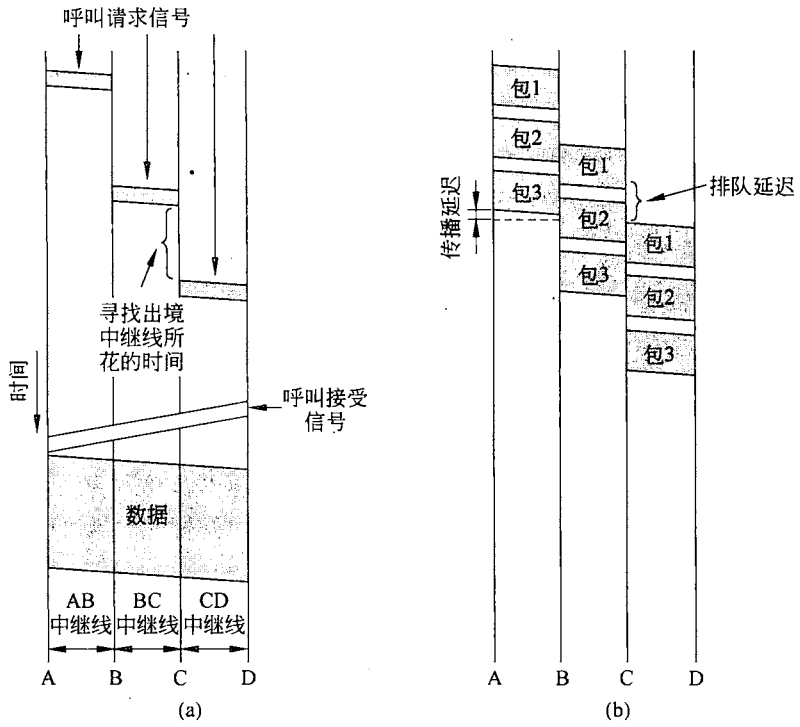


图 2-43 事件时序图
（a）电路交换；（b）数据包交换

包交换网络对数据包的大小规定了严格的上限。这样可确保没有任何用户可以长时间（比如许多毫秒）霸占任何传输线路，因此数据包交换网络可以处理交互式的网络通信。它还能减少延迟，因为在长消息的第二个数据包完全到达某个中间节点之前该消息的第一个数据包已经被转发出去了。然而，路由器内存中的数据包在被发往下一个路由器之前积累的存储-转发延迟超过了电路交换的延迟。在电路交换网络中，比特就好像流过线路一样连续地通过路由器。

数据包和电路交换在其他方面也有所不同。因为在数据包交换中没有为传输数据预留带宽，数据包可能不得不等待一段时间才能被转发。这样就引入了**排队延迟**（queuing delay），如果许多包要在同一时间被发送出去还会引入拥塞。另一方面，数据包交换却不存在用户听到一个忙音并且无法使用网络的危险。因此，对于这两种交换技术来说，拥塞

发生的时间不同，在电路交换中拥塞发生在建立电路时，而在数据包交换中拥塞发生在转发数据包时。

如果一条电路已经被预留给某一个特定的用户，但是并没有流量通过这条电路，那么这条电路的带宽就会被浪费，因为它不可能再被用于传输其他用户的流量。数据包交换不会浪费带宽，因此，从整个系统角度来看数据包交换的效率更高。了解这里的权衡对理解电路交换和数据包交换之间的差异非常重要。我们需要理解的权衡在于：要么保证服务，但是有可能浪费资源；要么不保证服务，但也不会浪费资源。

数据包交换比电路交换的容错性能更好。事实上，这也是为什么它被发明出来的原因。如果某个交换机出现故障，所有使用它的电路都将被终止，没有数据能从这些电路上发送出去。而用数据包交换，数据包可以绕过死掉的那个交换机。

电路交换和数据包交换之间的最后一个差别是收费方法不同。在电路交换中，从历史沿袭下来的做法是按距离和时间收费。对于移动电话，距离通常并不那么重要（除非是国际长途），时间扮演着一个粗糙的角色（举例来说，一个允许2000分钟免费通话的呼叫方案可能比一个允许1000分钟免费通话的呼叫方案更加贵，有时候夜里或者周末的话费比平常要便宜）。在数据包交换中，连接时间不是问题，流量大小却是个主要因素。对于家庭用户，ISP常常按月租费的方式来收取费用，因为这样可以减轻它们的收费工作负担，并且客户也很容易理解这种收费模型；但是骨干网运营商则按照流量大小收取区域网络的费用。

图2-44总结了电路交换和数据包交换的不同之处。传统意义上，电话网络使用电路交换技术来提供高质量的电话呼叫服务，而计算机网络使用的数据包交换技术则更加简洁和高效。然而，也存在着一些不容忽视的例外。一些较老的计算机网络内部是电路交换的（例如X.25），而一些较新的电话网络则使用了数据包交换的IP电话技术。这在外人看来就像是一个标准的电话呼叫，但在网络内部交换的却是语音数据包。这种方法已经使得用廉价电话卡打国际长途电话成为市场新贵，虽然通话质量也许比常规的电话呼叫质量要低。

项目	电路交换	包交换
呼叫建立	需要	不需要
专用物理路径	是	不是
每个包遵循相同的路由	是	不是
包按序到达	是	不是
交换机崩溃是否致命	是	不是
可用带宽	固定	动态
可能拥塞的时间	在建立时	在每个包
潜在浪费带宽	是	不是
存储-转发传输	不是	是
收费	按分钟计	按包计

图2-44 电路交换和数据包交换网络的比较

2.7 移动电话系统

传统的电话系统，即使有一天用几个速率为Gpbs的光纤连接端到端也仍然不能满足这样一群正在不断增长的用户需求：那就是运动中的用户。现在，人们希望在飞机上、汽

车上、游泳池中，或者在公园漫步时都能打电话。几年之内，他们还会希望在所有这些场所可以发送电子邮件或者浏览 Web 网页。因此，无线电话将有极大的利益和市场机会。本节中，我们将稍微详细地学习这个主题。

移动电话系统可用于广域范围的语音通信和数据通信。**移动电话**（mobile phone）有时称为**蜂窝电话**（cell phone），它的发展已经历了三代，俗称 1G、2G 和 3G。每一代有不同的技术：

（1）模拟语音。

（2）数字语音。

（3）数字语音和数据（Internet、电子邮件等）。

注意，别把移动电话与**无绳电话**（cordless phone）弄混。无绳电话由一个基站和一个手持听筒组成，一般成套出售，主要用于居家环境。它们从不被用来联网，所以我们对此不予深入讨论。

尽管我们的讨论绝大多数关于这些系统的技术，但是，有趣的是要注意政治和细微的营销决策可以产生巨大的影响。第一个移动系统由美国 AT&T 公司设计，并且得到 FCC 的允许在全国部署实施。结果，整个美国只有一个（模拟）系统，在加州购买的电话在纽约也可以使用。相反，当移动电话在欧洲出现时，每个国家都设计了自己的一套移动电话系统，从而导致移动电话系统市场的一片混乱。

欧洲从它的这次失败中吸取了教训，因此当数字系统到来时，由各国政府操控的邮电部门集中在一起，共同完成了一个标准化系统（GSM）。所以，现在任何一部欧洲移动电话都可以在欧洲的任何地方使用。而那个时候，美国政府认为它不应该卷入到移动电话的标准化事务中，它把数字系统留给了市场自己去运行。这个决策的结果是不同的设备厂商生产了不同种类的移动电话。因此，美国现在有两个主要的但互不兼容的数字移动电话系统，以及其他一些小系统。

尽管移动电话最初由美国领导，但现在欧洲移动电话的拥有量和应用大大超过了美国。全欧洲统一使用同一个系统，当然这只是一部分原因，其他还有一些原因。美国和欧洲数字移动电话系统的第二个不同点在于电话号码的分区规范。在美国，移动电话号码与常规的固定电话号码混合在一起。因此，呼叫方无法识别一个号码，比如（212）234-5678，是一个固定电话（费用便宜，甚至免费），还是一个移动电话（费用昂贵）。为了避免人们对使用电话神经过敏，电话公司决定移动电话机主接听电话要付费。结果是，许多人害怕因接听电话而必须支付大笔费用，而对是否购买移动电话犹豫不决。在欧洲，移动电话有一个特殊的区域码（类似于 800 和 900 号码），所以移动电话号码很容易和固定号码区分。因此，“主叫方付费”的通用规则适用于欧洲的移动电话（不过，国际呼叫的费用另有算法）。

第三个问题，对用户选择移动电话有很大的影响，欧洲广泛使用了预付费移动电话（在某些地区达到了 75%）。这些电话在很多商店都可以买到，其手续并不比买一个数字相机更加烦琐。你付钱就可以拿走电话。这些电话已经预先支付了一定数量的费用，比如 20 欧元或者 50 欧元，而且当余额下降到零时还可以续费（通过一个保密的 PIN 码）。结果，欧洲几乎每一个青少年和许多很小的孩子都有移动电话（通常是预付费的），这样他们的父母总是可以知道他们在哪里，并且不用担心他们会花掉大笔的电话费用。如果只是在偶尔情况下使

用移动电话的话，这种用法也非常合适，因为它没有月租费用，或者接听电话不需要付费。

2.7.1 第一代移动电话（1G）：模拟语音

有关移动电话的政治和市场情况已经说得够多了。现在我们来看移动电话的技术，从最早的系统开始。在20世纪最初的几十年中，移动无线电话偶尔出现在海军和军事通信中。1946年，圣·路易斯建立起了第一个可用在汽车上的电话系统。该系统使用了一个被安置在高大建筑物顶上的大型发射器，并且只有一个信道用于发送和接收。为了通话，用户必须按一下按钮以便打开发送功能并关闭接收功能。这样的系统称为**按钮通话系统**（push-to-talk system），20世纪50年代后期开始陆续在几个城市安装了这样的系统。电视节目中的CB无线电、出租车和警车通常使用这项技术。

在20世纪60年代，**改进型移动电话系统**（IMTS，Improved Mobile Telephone System）开始被安装使用。它也使用了一个放置在一座小山顶上的大功率（200瓦）发射器，但是这次有两个频率，一个用于发送，一个用于接收。因此，启动通话的按钮就不再需要了。由于所有来自移动电话的通信工作在入境信道上，该信道不同于出境信号信道，所以移动用户相互之间听不到别人的通话（不同于出租车使用的按钮通话系统）。

IMTS支持23个信道，频率范围为150～450 MHz。由于信道的数量比较少，所以，用户常常需要等待很长时间才能听到拨号音。而且，由于小山顶上发射器的功率很大，邻近的系统必须相距几百千米远才能避免干扰。总而言之，有限的容量使得该系统无法被真正实际使用。

高级移动电话系统

高级移动电话系统（AMPS，Advanced Mobile Phone System）的出现改变了所有的一切。该系统由贝尔实验室发明，并且于1982年首次在美国安装部署。随后英国和日本也安装了这样的系统，在英国称它为TACS，在日本称之为MCS-L1。AMPS于2008年正式退休，但我们还将考察它的细节，这样才能很好地理解基于改进它而得到的2G和3G系统。

在所有的模拟电话系统中，一个地理区域被分成许多个**蜂窝**（cell），这就是为什么有时候移动电话称为**蜂窝电话**（cell phone）的原因。在AMPS中，每个蜂窝通常为10～20千米的跨度；在数字系统中，蜂窝要小一些。每个蜂窝使用一组频率，这组频率不会被它任何一个邻居所使用。蜂窝系统之所以比以前的系统拥有更大的容量，关键在于它使用了相对较小的蜂窝，并且较近（但不相邻）的蜂窝可重复使用相同的传输频率。假设在100千米范围内只有一个IMTS系统，在每个频率只能有一个电话呼叫；而在同样的区域范围内，AMPS系统可以有100个范围为10千米的蜂窝，并且每个频率上可以有10～15个电话呼叫。因此，蜂窝设计思想至少使系统增加了一个数量级的容量，而且蜂窝越小，容量增加得越多。此外，小蜂窝意味着所需要的功率更小，因而发射器和手持设备也更小、更便宜。

频率重用的思想如图2-45（a）所示。蜂窝近似粗糙的圆形，但是很容易用六角形作为它们的模型。在2-45（a）中，蜂窝大小都相同，它们每7个组成一组。每个字母代表了一组频率。注意，对于每组频率，都有一个两蜂窝宽的缓冲带，在缓冲带中该频率组不会被重用，从而保证了良好的隔离性和较低的干扰。

找到一个较高的空中位置来放置基站天线是蜂窝系统的一个主要问题。这个问题导致了一些电信运营商与罗马天主教会结成了联盟，因为罗马天主教会在全世界拥有大量很高的潜在天线放置地点，并且都在其统一管理之下。

在有些地区，用户数量的增长超出了系统的负载能力，移动运营商可通过降低功率和将超载的蜂窝切分成更小的**微蜂窝**（microcell），使得更多的频率被重用，如图 2-45（b）所示。有时，当举行运动赛事、摇滚音乐会和其他什么场合聚集大量移动用户多达几个小时时，电话公司就会利用便携式塔（塔上有卫星链路），建立一些临时的微蜂窝，来缓解系统压力。

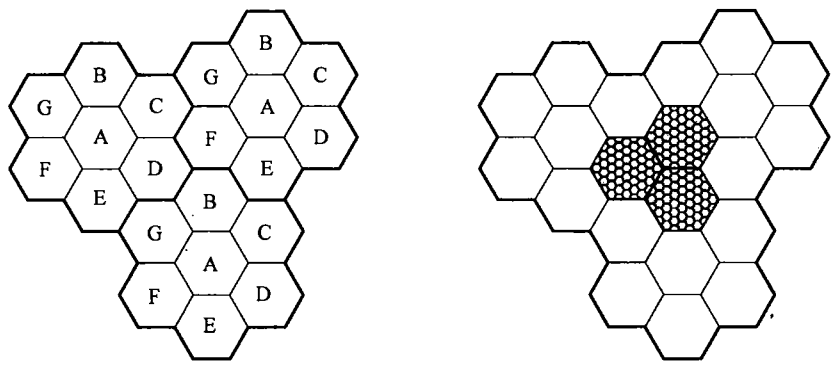


图 2-45
（a）在相邻的蜂窝中不能重用频率；（b）为增加更多的用户可使用更小的蜂窝

在每个蜂窝的中心是一个基站，负责蜂窝中的所有电话传输。基站是由一台计算机和连接到一个天线上的发射器/接收器组成的。在一个小规模的系统中，所有的基站都连接到一个称为**移动电话交换局**（MTSO，Mobile Telephone Switching Office）或者**移动交换中心**（MSC，Mobile Switching Center）的设备上。规模大一点的系统可能需要几个 MTSO，所有的 MTSO 都连接到一个二级 MTSO 上，以此类推。MTSO 基本上类似于电话系统中的端局，而且事实上，它们也确实连接到至少一个电话系统端局。MTSO 与基站、其他 MTSO 以及 PSTN 通过数据包交换网络进行通信。

在任何时刻，每个移动电话逻辑上属于某个特定的蜂窝，并且受该蜂窝基站的控制。当一个移动电话在物理上离开一个蜂窝时，它的基站会注意到该电话的信号越来越弱，于是询问周围的基站它们从该电话上得到的功率多大。根据获得的回答，该基站将所有权转交给获得最强信号的那个蜂窝；绝大多数情况下，这个蜂窝就是该电话当前所在的那个蜂窝。然后，该电话就会接到通知，自己有新老板了；而且，如果当时正在通话，它就会被要求切换到一个新的信道上（因为老的信道在任何一个相邻蜂窝内是无法重用的）。这个过程就称为**切换**（handoff），大约需要 300 毫秒。信道分配由系统的神经中枢 MTSO 来完成，基站实际上只负责无线电波的中继。

信道

AMPS 使用 FDM 来划分信道。系统使用了 832 个全双工信道，每个信道由一对单工信道组成。这样的一种安排称为**频分双工**（FDD，Frequency Division Duplex）。频率范围

为824～849 MHz的832个单工信道被用作移动电话到基站的发送信道，频率范围为869～894 MHz的另外832个单工信道被用作基站到移动电话的发送信道。每个单工信道的频宽为30 kHz。

832个信道分成4类。控制信道（从基站到移动电话）用于管理系统；寻呼信道（从基站到移动电话）用于提醒移动用户有呼叫到来；接入信道（双向）用于呼叫的建立和信道分配；最后，数据信道（双向）承载语音、传真或数据。由于相同的频率不能在相邻的蜂窝中重用，而且每个蜂窝保留了21条信道用于控制，因此每个蜂窝中实际可用的语音信道的数目远远小于832，通常只有45个左右。

呼叫管理

在AMPS系统中，每部移动电话有一个32比特的序列号和一个10位数字的电话号码，这些数字存放在其可编程的只读存储器中。电话号码的表示方法是：3位数字的区域码占10比特，7位数字的用户号码占24比特。当电话开机时，它对预先设置的21条控制信道进行扫描，找到最强的那个信号；然后，电话广播自己32个比特的序列号和34个比特的电话号码。尽管语音信道本身是模拟的，但是，如同AMPS中所有的控制信息一样，这个数据包以数字形式被多次发送，并且带有纠错码。

当基站听到移动电话的广播信息后，它就告诉MTSO；然后MTSO记录下新客户的到达情况，同时通知该客户的家乡MTSO它的当前位置。在正常的操作过程中，移动电话每隔15分钟左右重新注册一次。

移动用户想打电话时先打开电话，在键板上输入被叫电话号码，再按下发送（SEND）按钮；然后，电话将被叫号码以及它自己的标识通过接入信道发送出去。如果发生碰撞，它会试着再次发送。当基站接到了来自电话的呼叫请求时，它就通知MTSO。如果主叫者是该MTSO公司（或者它的某一个合作伙伴）的客户，则MTSO为这次呼叫寻找一条空闲的信道。如果找到了可用的信道，它就通过控制信道将可用信道的号码发回电话；然后移动电话自动切换到被选中的语音信道上等待，直到被叫方拿起电话。

入境电话呼叫的工作方式有所不同。刚开始时，所有空闲的电话都在不断地监听寻呼信道，以便检测是否有消息发给它们。当呼叫一部移动电话时（从固定电话上发起呼叫，或者从另一部移动电话上发起呼叫），被叫方的家乡MTSO就会收到一个分组，询问被叫方现在哪里；然后，一个数据包被发送到被叫电话当前所在蜂窝的基站，基站在寻呼信道上发送一条广播消息，消息形如“14号，你在吗？”；然后被叫电话在接入信道上回答“是的，我在”。基站接着就会这样说“14号，有人在3号信道上呼叫你”。此时，被叫电话切换到3号信道，铃声响起（或者播放一段优美的旋律，这段音乐是电话主人以前收到的一份生日礼物）。

2.7.2 第二代移动电话（2G）：数字语音

第一代移动电话是模拟的，第二代移动电话则是数字的。从模拟切换到数字有几个优点。首先，通过将语音信号数字化处理和压缩带来了容量上的收益；其次，通过对语音和控制信号实行加密改进了安全性，这反过来又防止了欺诈和窃听，不管是对有意扫描还是

因射频传播导致的其他呼叫干扰。最后，它催生了诸如手机短信等新服务的展开。

如先前第一代移动电话没有全球标准化一样，第二代移动电话的发展也没有形成全球化的统一标准，已经开发出来并已被广泛获得部署的系统有好几种。**数字高级移动电话系统**（D-AMPS，Digital Advanced Mobile Phone System）是数字版本的 AMPS。它可与 AMPS 并存，使用 TDM 把多个电话呼叫复用在同一频率信道。D-AMPS 由国际标准的 IS-54 和其继任 IS-136 描述。**全球移动通信系统**（GSM，Global System for Mobile communications）已成为占主导地位的系统。虽然在美国流行比较慢，但实际上现在它在全球已无处不在。与 D-AMPS 一样，GSM 也是 FDM 和 TDM 的混合。**码分多址**（CDMA，Code Division Multiple Access）则是一个完全不同类型的系统，它既不基于 FDM 也不基于 TDM，由国际标准 IS-95 描述。尽管 CDMA 没有成为占主导地位的 2G 系统，但其技术已成为 3G 系统的基础。

此外，在推销资料中有时也用**个人通信服务**（PCS，Personal Communications Services）这个名称来表示第二代系统（即数字的）。这个名称的本意是使用 1900 MHz 频段的移动电话，但现在很少这样区分了。

我们现在描述 GSM，因为它主导着 2G 系统。在下一节描述 3G 系统时，我们将更多地关注 CDMA。

GSM——全球移动通信系统

全球移动通信系统始于 20 世纪 80 年代，作为欧洲单一 2G 标准化的努力而诞生。这项任务分配给了一个名为 Groupe Speciale' Mobile（法文）的电信组织。第一个 GSM 系统在 1991 年开始部署，并很快取得成功。随着它被远在澳洲的国家所接受，人们明白 GSM 将会获得比在欧洲更大的成功，因此更名 GSM 使得它具有更多的全球吸引力。

GSM 和其他我们将要了解的移动电话系统保留了 1G 系统的设计理念，以蜂窝为基础、频率可跨蜂窝复用，并随着用户的移动而切换蜂窝。但 2G 系统和 1G 系统在细节上有很大的不同。这里我们简要讨论 GSM 的一些主要特性。不过，打印出来的 GSM 标准超过 5000 页（原文）。文档中的很大一部分涉及系统的工程方面，尤其是处理多径信号传播的接收器设计以及发射器和接收器的同步。所有这些内容这里都不予叙述。

图 2-46 显示了 GSM 体系结构，虽然组件的名称不同，但与 AMPS 体系非常相似。现在，移动电话本身可以分成手机和一个可移动芯片两部分。芯片具有用户和账户信息，称为 SIM 卡，即**用户识别模块**（Subscriber Identity Module）的简称。正是 SIM 卡激活了手机，并包含了移动电话和网络相互识别对方和加密通话所需要的机密。SIM 卡可以被取出并插

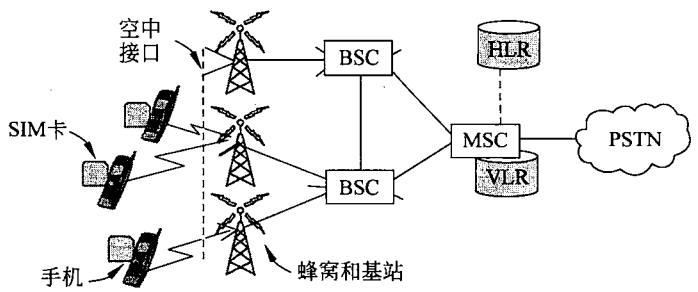


图 2-46 GSM 移动网络体系结构

入到一个不同的手机，对网络而言该手机就成了你的移动电话。

移动电话通过**空中接口**（air interface）与蜂窝基站通话，我们稍后将描述该接口。每个蜂窝基站都连接到一个**基站控制器**（BSC，Base Station Controller），该控制器控制蜂窝的无线资源分配并处理切换事务。BSC 又被连接到一个 MSC（就像 AMPS 一样），由 MSC 负责电话呼叫的路由并和 PSTN（公共交换电话网）相连。

为了能够路由呼叫，MSC 需要知道目前在哪里可以找到手机。它维护着一个称为**访问位置寄存器**（VLR，Visitor Location Register）的数据库，该数据库包括了所有附近的移动电话，这些移动电话都与它所管理的蜂窝关联。移动网络中还有另外一个数据库，记录了每个移动电话的最后一个已知位置。这就是所谓的**归属位置寄存器**（HLR，Home Location Register）。这个数据库用来把入境呼叫路由到正确的位置。这两个数据库必须在移动电话从一个蜂窝移动到另一个蜂窝时及时更新。

现在我们详细描述空中接口。GSM 可在很大的频率范围内运行，包括 900 MHz、1800 MHz 和 1900 MHz。这是为了支持更多的用户数量，为 GSM 分配的频谱比 AMPS 多。与 AMPS 类似，GSM 也是一种频分双工蜂窝系统。也就是说，每个移动电话在某个频率上发送而用另一个更高的频率接收（GSM 的接收频率高达 55 MHz，而 AMPS 的接收频率高达 80 MHz）。然而，与 AMPS 不同的是，GSM 的一对频率按照时分多路复用又被细分成多个时间槽。这样多个移动电话可共享这一对频率。

为了处理多个移动电话，GSM 信道比 AMPS 信道宽了许多（GSM 的每个信道宽 200 kHz，而 AMPS 的信道宽 30 kHz）。图 2-47 给出了一个 200 kHz 的信道。运行在 900 MHz 频域的 GSM 系统有 124 对单工信道。每个单工信道宽 200 kHz，采用时分复用技术可支持 8 个单独的连接。每个当前活跃的移动电话被分到某对信道上的某个时间槽。从理论上讲，每个蜂窝可支持 992 个信道，但其中有不少是不能用的，这主要是为了避免与邻近蜂窝的频率冲突。在图 2-47 中，八个阴影标示的时间槽属于相同的连接，每个方向上 4 个时间槽。发送和接收没有出现在同一个时间槽内，因为 GSM 无线电不能在同一时间进行发送和接收，而从一个状态切换到另一个需要一定的时间。如果移动设备分配得到 890.4/935.4 MHz 和时间槽 2，当它需要给基站发送时，它就使用下面 4 个阴影时间槽（和后续的那些），在每个时间槽内发送一些数据直到发送完全部数据。

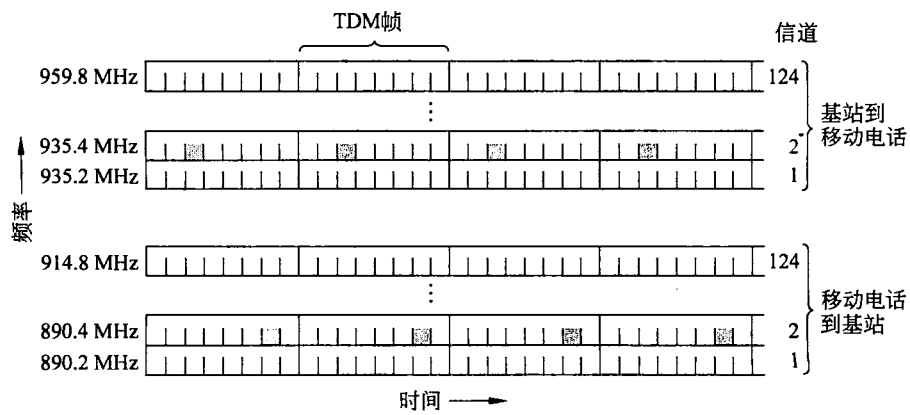


图 2-47　GSM 使用 124 个频率信道，每个信道使用 8 个时间槽的 TDM 系统

图 2-47 显示的 TDM 时间槽只是复杂成帧层次结构中的一部分。每个 TDM 时间槽是一个特定的结构，一组 TDM 时间槽组合起来形成多帧（multiframe）结构，多帧也有特定的结构形式。帧的层次结构的一个简化版本如图 2-48 所示。从这里我们可以看到，每个 TDM 时间槽包含一个 148 比特的数据帧，它占用信道 577 微秒（包括每个时间槽之后的 30 微秒保护时间）。每个数据帧的开始和结束都有 3 个比特 0，用于帧的分界；还包含 2 个 57 比特的 Information（信息）字段。每个 Information 字段都有一个控制比特，指出随后的 Information 字段包含的是语音还是数据。在两个 Information 字段之间是一个 26 比特的 Sync（训练用）字段，接收方利用这个字段同步到发送方的帧边界。

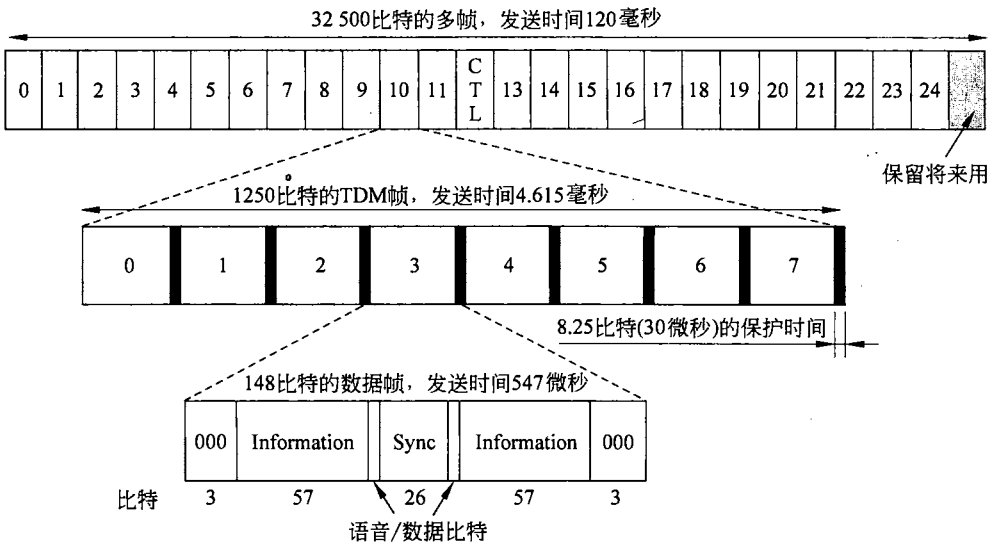


图 2-48 GSM 帧结构的一部分

发送一个数据帧需要 547 微秒，但在每 4.615 毫秒以内，一个发射器只允许发送一个数据帧，因为它与其他 7 个站共享同一个信道。每条信道的总传输率为 270 833 bps，分给 8 个用户使用。然而，如同 AMPS 一样，各种额外开销吃掉了相当大一部分的带宽，最终每个用户在纠错之前拥有的有效载荷只有 24.7 kbps。经过纠错之后，留给语音的只剩下 13 kbps。虽然相比固定电话网络中 64 kbps 未经压缩的 PCM，这里的语音带宽少得多，但在移动设备上经过压缩后的效果只损耗了很少一点。

从图 2-48 可以看出，8 个数据帧构成了一个 TDM 帧，26 个 TDM 帧由构成了一个 120 毫秒的多帧。在一个多帧的 26 个 TDM 帧中，时间槽 12 用于控制，时间槽 25 保留为将来使用，所以只有 24 个时间槽可用于传输用户流量。

然而，除了图 2-48 中显示的具有 26 个时间槽的多帧结构以外，GSM 还使用了 51 个时间槽的多帧结构（图中没有显示）。这些时间槽中有一些被用于几个控制信道，GSM 通过这些控制信道来管理系统。**广播控制信道**（broadcast control channel）从基站输出一个连续流，其中包含了该基站的标识和信道的状态。所有的移动站都监视它们的信号强度，以便了解何时移动到了一个新的蜂窝中。

专用控制信道（dedicated control channel）用于移动用户的位置更新、注册和呼叫的建立。尤其是，每个 BSC 都维护了一个关于当前在它管辖下的移动站的数据库，即 VLR。

维护 VLR 所需要的信息都是在专用控制信道中发送的。

最后，还有一条**公共控制信道**（common control channel），它被分成 3 个逻辑子信道。第一个子信道是**寻呼信道**（paging channel），基站用它通告有关入境呼叫的情况。每个移动站都要不停地监视该信道，以便发现那些应该由自己来回答的电话呼叫。第二个子信道是**随机接入信道**（random access channel），它允许用户在专用控制信道上请求一个时间槽。如果两个请求冲突，它们都会遭到拒绝，必须以后再重新尝试发送请求。移动站利用专用控制信道中的时间槽来发起一次电话呼叫。第三个子信道为**接入授予信道**（access grant channel），用于宣布分配获得的专用控制信道的时间槽。

最后，GSM 不同于 AMPS 之处还在于如何处理切换。在 AMPS 中，MSC 完全负责切换而无须移动设备的协作。随着 GSM 中信道被划分成时间槽，移动设备在大部分时间内既没有发送动作也没有接收动作。这些空闲的时间槽就给了移动设备测量到附近其他基站的信号质量的好机会。事实上，它也正是这样做的，它把测量获得的信息发送给 BSC。BSC 用这些信息来确定移动电话是否正在离开一个蜂窝并进入另一个蜂窝，从而决定是否执行切换。这种设计称为**移动辅助切换**（MAHO，Mobile Assisted HandOff）。

2.7.3 第三代移动电话（3G）：数字语音和数据

第一代移动电话是模拟话音，第二代移动电话是数字语音，第三代移动电话，或所谓的 3G，则包含了数字语音和数据。

移动通信行业被一些因素推动着迈向了 3G。首先，固定网络上的数据流量早就超过了话音流量，并呈几何级数增长，而语音流量发展基本持平。许多业内专家预计移动设备上的数据流量将很快主宰语音流量。其次，电话、娱乐和计算机行业都已经数字化，并正在快速融合。很多人对那些同时可用作电话、音视频播放器、电子邮件终端、网页接口、游戏机等多种功能的轻巧便携式设备垂涎三尺，这些设备具备广域的无线连接能力，以高带宽与 Internet 连接。

苹果的 iPhone 就是此类 3G 设备的一个极好例子。有了它，人们沉迷于无线数据服务。随着 iPhone 受欢迎程度的普及，AT&T 的无线数据量正在急剧上升。麻烦的是，iPhone 使用的是 2.5G 网络（一种增强型的 2G 网络，不是真正意义上的 3G 网络），而且也没有足够的数据能力使用户满意。3G 移动电话正是致力于提供足够的无线带宽，使这些未来的用户满意。

1992 年 ITU 针对这个愿景试图描绘出更具体的设想。它发出了实现该愿景的蓝图，即国际移动通信-2000（IMT-2000，International Mobile Telecommunications-2000）。IMT-2000 网络给它的用户提供的基本服务有：

（1）高质量的语音传输。

（2）消息（代替电子邮件、传真、SMS、聊天等）。

（3）多媒体（播放音乐、观看视频、电影、电视等）。

（4）Internet 接入（Web 浏览，包括带音频和视频的页面）。

其他的服务还可能包括视频会议、远程出席（telepresence）、群组游戏和移动商务（m-commerce）（在商店购买物品时用移动电话来支付费用）。而且，所有这些服务都应该

是全球性的（找不到地面网络时通过卫星自动建立连接）、即时可完成的（总是可用的），并且有一定的服务质量保证。

ITU 为 IMT-2000 设想了单一的全球技术，因此制造厂商们可以生产出在世界上任何地方销售和使用的设备（就好像 CD 播放器和计算机，而不像移动电话和电视机）。单一的技术也使得网络运营商的工作更加简单，并且可以鼓励更多的人使用这些服务。格式之争对于企业并不是件好事，比如当年录像机刚出来时 Betamax 与 VHS 之间的争斗。

事实证明，有点过于乐观了。2000 这个数字有 3 个含义：（1）在 2000 年提供服务，（2）运行在 2000 频率上（以 MHz 为单位），（3）提供服务的带宽应该达到 2000（以 kbps 计）。但这 3 项中的任何一项都没有实现，到了 2000 年什么都没有发生。ITU 建议所有国家政府预留出 2 GHz 频谱，以便移动设备可以在各国之间实行无缝漫游。但只有中国一个国家保留了所需的带宽，除此之外，没有任何一个其他国家是这样做的。最后，人们终于认识到对于太快移动的用户 2 Mbps 在目前根本不可行（由于切换速度不够快）。更现实的做法是为固定的室内用户提供2 Mbps（这是为了与ADSL竞争），为室外行走用户提供384 kbps，而为坐在快速运行汽车里的用户提供 144 kbps 连接。

尽管出师不利，经受了这样或者那样的初始挫折，但迄今为止已经取得了很大成就。经过几番甄选，几个 IMT 建议脱颖而出，它来自两个主要阵营。第一个，**宽带码分多址**（WCDMA，Wideband CDMA），由爱立信公司提出并获得欧洲联盟推进，称为**全球移动通信系统**（UMTS，Universal Mobile Telecommunications System）。另一个竞争者是由高通公司（Qualcomm）提出的 CDMA2000。

来自两大阵营的这些系统的相似性超过了它们之间的差异性，因为它们都是以宽带 CDMA 为基础：WCDMA 使用 5 MHz 信道而 CDMA2000 采用 1.25 MHz 信道。如果把爱立信和高通的工程师们安置在一个会议室，要求他们完成一个共同设计，他们也许很快地就能设计出方案。麻烦的是，真正的问题不在工程，而在政治（通常都如此）。欧洲希望建立一个与 GSM 互联的系统，而美国则希望有一个能与本土已广泛部署的网络（IS-95）相兼容的系统。每一方都支持自己的本土公司（爱立信公司总部位于瑞典，高通公司则在加利福尼亚）。最后，爱立信公司和高通公司都参与了许多针对各自 CDMA 专利的诉讼。

在全球范围内，10～15%的移动用户已经在使用 3G 技术。在北美和欧洲，这个比例大一些，大约三分之一的移动用户是 3G 用户。日本是最早部署 3G 的国家，现在几乎所有的移动电话都是 3G。这些数字包括部署的 UMTS 和 CDMA2000，3G 仍然是一个随着市场震动的活动沸点。更为混乱的是，UMTS 成为具有多个不兼容选项的单一 3G 标准，包括 CDMA2000。这种改变原本是为了统一各个阵营而作的一种努力，但是掩盖了技术上的差异，使得当前工作的重点变得不清晰。我们将使用 UMTS 来表示 WCDMA，以便与 CDMA2000 区别。

我们将重点放在讨论如何在蜂窝网络中使用 CDMA，因为这是两个系统的显著特色。CDMA 既不是频分复用也不是时分复用，而是一种每个用户可在同一时间用同一频段发送的混合技术。当它第一次被提议用于蜂窝系统时，业界的反应就如同当年伊莎贝拉女王首次听哥伦布说由于航行方向错误而到达印度的反应一样。但是，经过高通一家公司的坚持不懈努力，CDMA 获得终于成功，成为 2G 系统（IS-95），并且成熟发展成 3G 的技术基础。

为了使CDMA能在移动电话装置中工作，技术要求比我们在前一节描述的基本CDMA技术更高。具体来说，我们介绍了同步CDMA，其中码片序列完全正交。这种设计之所以能正常工作在于所有用户的码片序列开始时间是严格同步的，就像基站给移动电话发送信号一样。基站能够在同一时间开始发射码片序列，这样信号得以正交并能被分开。然而，要使一个个独立的移动电话保持传输同步则非常困难。稍不小心，它们的传输就会在不同的时间到达基站，这样的话就无法保证码片的正交性。为了让移动电话在不同步时也能给基站发送，我们希望码片序列在一切可能的信号偏移量内保持正交，而不是简单地在开始时就要保持一致。

尽管不太可能为这种一般情况找到完全正交的码片序列，但长伪随机序列已足够接近该目标。长伪随机序列具有这样的属性，它们在所有偏移量内互相关性非常低（cross-correlation）的概率相当高。这就是说，一个序列乘以另一个序列计算获得的内积和很小，而如果它们是正交的则结果将是零（直观上，随机序列应该看上去总是不相同的，把它们相乘生成一个随机信号，其内积和的计算结果很小）。这就使得接收器可以把接收信号中的不必要部分过滤掉。另外，除了在偏移量为零的那点外，伪随机序列的**自相关性**（auto-correlation）很小的概率也相当高。这意味着，一个随机序列乘以被延迟的自身复制，计算得出的内积和结果也很小，除非延迟值是零（直观地说，被延迟的随机序列看起来就像是另一个不同的随机序列，因此问题的本质又回到了互相关性上）。这样，接收器就能在接收信号中锁住想要传输的开始。

伪随机序列的应用使基站能接收来自非同步移动电话的CDMA消息。然而，在我们讨论CDMA时隐含了这样的假设，即所有移动电话的功率在接收方是相同。如果接收信号的功率不同，一个强信号的很小互相关性都有可能压倒一个弱信号的很大自相关性。因此，移动电话的发射功率必须加以控制，才能最小化竞争信号之间的干扰。正是这种干扰限制了CDMA系统的容量。

基站接收到的能量级别取决于发射器离基站的远近以及它们的发送功率。任何时候都可能存在许多移动电话，它们与基站的距离不等。一种好的启发式均衡接收功率的方法是针对每个移动电话，其发往基站的功率与它从基站接收到的功率强度相反。换句话说，如果移动电话接收到来自基站的信号很弱，则它使用更多的发射功率来获得更强的信号。为了更加准确，基站也会给每个移动电话反馈信息，告诉它们应该增加、减少或保持稳定的发射功率。因为良好的功率控制对于最小化干扰非常重要，因此这种反馈非常频繁（每秒1500次）。

对我们前面介绍过的基本CDMA方案的另一个改进是让不同的用户以不同的速率发送数据。这一技巧在CDMA中很自然地就能做到，只要固定码片的传输速率并且为用户分配不同长度的码片序列。例如，在WCDMA中，码片速率为3.84 Mchips/s，扩频码长度从4～256不同。用长度为256的码片时，差错检测后还剩下大约12 kbps，这个容量足够进行语音通话；而用长度为4的码片时，用户数据率将接近1 Mbps；中间长度的码片给出了中间速率的数据传输。若要获得多个Mbps的速率，移动电话必须一次使用超过5 MHz的信道。

我们已经了解到CDMA正常工作必须处理的问题，现在让我们来看看CDMA的优势。它有3个主要优点。首先，CDMA可提升容量，这是利用了一些发射器静止沉默时的小周期优势。在礼貌的语音通话中，一方在说话时另一方会沉默倾听。平均而言，只有40%的线路时间处于忙状态。然而，谈话之间的停顿可能会很小，而且难以预料。用TDM或FDM

系统，重新分配时间槽或频率信道的速度不可能足够快到能从中受益。然而，在 CDMA 中，只要简单地不给一个用户发送信号就能降低对其他用户的干扰，而且在一个繁忙的蜂窝中任何时间可能只有一部分用户在发射。因此，CDMA 利用了可预测沉默期的优势，做到了支持更多数量呼叫的同时进行。

其次，用 CDMA，每个蜂窝可使用相同的频率。与 GSM 和 AMPS 不同的是，这里不需要 FDM 来区分不同用户的传输。由此消除了复杂的频率规划工作，也提高了系统容量。而且有利于基站使用多个**定向天线**或**扇形天线**(sectored antennas)，而不是必须用全向天线。定向天线把信号集中在某个预定方向，并降低信号强度，从而减少对其他方向的干扰。这反过来又增加了容量。通常设计方案是采用三个扇形天线。基站必须时刻跟踪移动电话从一个扇形区域移动到另一个扇形区域。这种跟踪对 CDMA 来说非常容易，因为所有的频率在所有扇形区域都可使用。

最后，CDMA 采用了**软切换**（soft handoff）技术，移动电话在与老基站完全中断之前就已被新基站接管。如此一来，不会丢失连接的连贯性。软切换如图 2-49 所示。软切换对 CDMA 而言做起来很容易，因为每个蜂窝使用了所有的频率。与之对应的另一种切换方案是**硬切换**（hard handoff），即移动电话在被新基站接管之前必须与老基站连接中断。如果新基站无法接管它（例如，因为没有可用的频率），呼叫就会突然中断。用户往往会注意到这一点，但它在当前的设计中不可避免会偶然发生。硬切换对于 FDM 的设计规范是正常的，这是为了避免在两个频率上同时进行发送或接收带来的成本控制。

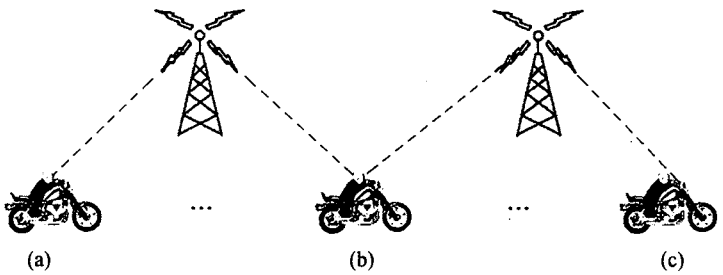


图 2-49 软切换
（a）之前；（b）之中；（c）之后

有关 3G 已有许多文章发表，其中大部分都把它称赞为有史以来最伟大的一件事。与此同时，许多运营商已经朝着 3G 方向迈开了谨慎的脚步，即转到了 2.5G，虽然说 2.1G 可能更准确些。这样的系统是**增强数据率的 GSM 演进**（EDGE，Enhanced Data rates for GSM Evolution），它仍然是 GSM，只是每个符号运送更多的比特。麻烦的是，每个符号运送更多的比特意味着每个符号的更多错误，所以 EDGE 有 9 种不同的调制和纠错方案，这些方案的不同之处在于用多大的带宽处理因高速率引入的错误。EDGE 是沿着一条从 GSM 到 WCDMA 所定义的进化路径迈出的一步。同样地，针对运营商也有一个从 IS-95 进化到 CDMA2000 网络所定义的进化路径。

虽然 3G 网络还没有全面展开部署，一些研究人员认为 3G 木已成舟。这些人已经开始着手 4G 系统的研究，4G 系统取名**长期演进**（LTE，Long Term Evolution）。4G 建议的一些特性包括：高带宽、普适性（连接无处不在）、与其他有线和无线 IP 网络的无缝集成（包括 802.11 接入点）、频谱和资源的自适应管理以及高品质的多媒体服务。欲了解更多信息，

请参考（Astely 等，2009；Larmo 等，2009）。

与此同时，具备 4G 性能水平的无线网络早已可用。主要的例子是 802.16，也称为 WiMAX。有关移动 WiMAX 的概述，请参阅（Ahmadi，2009）。如果说工业界一直处于飘忽不定状态的说法太过轻描淡写，我们只要往回看看几年间究竟发生了什么就不觉得奇怪了。

2.8 有线电视

我们已经相对仔细地学习了固定电话系统和无线电话系统。很明显，在将来的网络中，它们都将继续扮演着重要的角色。然而，在过去 10 年间还出现了另一种可用于 Internet 接入的方法：即有线电视网络。许多人已经将他们的电话和 Internet 服务放到了有线电视网络上。在本节中，我们将从网络的角度来详细地介绍有线电视系统，并且将它与我们刚刚学过的电话系统作一对比。有关有线电视系统的更多信息，请参看（Laubach 等，2001；Louis 2002；Ovadia，2001；Smith，2002）。

2.8.1 共用天线电视

关于有线电视的构想在 20 世纪 40 年代后期就已经有了，当时的主要目的是为了给居住在农村和山区的人们提供更好的收视效果。系统最初包括以下几个部分：一个大天线，一般放在山顶上，以便将电视信号从空中接收下来；一个放大器，也称为**头端**（headend），它可以加强信号；一根同轴电缆，它将电视信号送到用户住宅，如图 2-50 所示。

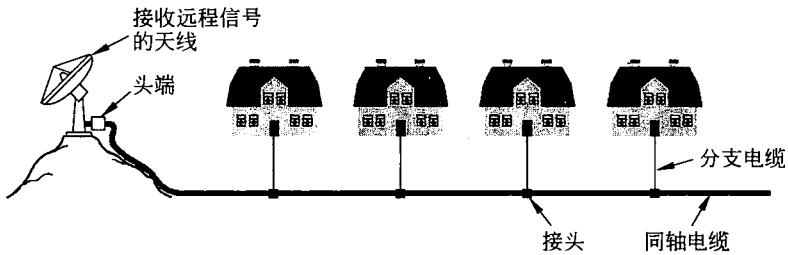


图 2-50 早期的有线电视系统

在早期阶段，有线电视称为**共用天线电视**（Community Antenna Television）。它是一种家庭作坊式的运作模式：任何人配备一些电子设备就可以在其所在城镇建立这样的服务，然后，用户凑钱来支付所有的费用。随着用户数量的增加，在原来电缆的基础上，还需要拼接额外的电缆，并且需要时还得加入放大器。在这样的系统中，信号传输是单向的，从头端传输到用户处。到 1970 年，已经存在几千个这样的独立系统。

1974 年，时代公司（Time, Inc.）开辟了一个新的频道——Home Box Office，该频道播放一些新的内容（电影），并且只通过电缆广播。随后又出现了一些只在电缆上播放的频道，内容涉及新闻、体育、烹饪，以及其他许多话题。这种发展引起了工业界的两个变化。第一，大公司开始购买已有的电视系统，并且铺设新的电缆来吸引新用户。第二，市场出现了将多个电视系统连接起来的新需求，通常需要将相距较远的城市连接起来以便传播新的电视频道内容。有线电视公司开始在城市之间铺设电缆，目的是将这些城市连接到同一个

系统中。这种模式非常类似于电话工业在80年前所做的事情，当时电话公司也面临着将孤立的端局连接起来提供长途电话服务的需求。

2.8.2 线缆上的 Internet

有线电视系统经过多年的发展业已成熟并不断壮大，各个城市之间的电缆已经被替换成了高带宽的光纤，整个过程非常类似于电话系统的发展历程。如果一个系统中长距离使用的是光纤，而连接到家庭的是同轴电缆，则这样的系统称为**混合光纤电缆**（HFC，Hybrid Fiber Coax）系统。系统的光学部分和电子部分之间的接口是光电转换器，称为**光纤节点**（fiber node）。因为光纤的带宽远远超过同轴电缆的带宽，所以，一个光纤节点可以连接多根同轴电缆。图2-51（a）显示了一个现代HFC系统的一部分。

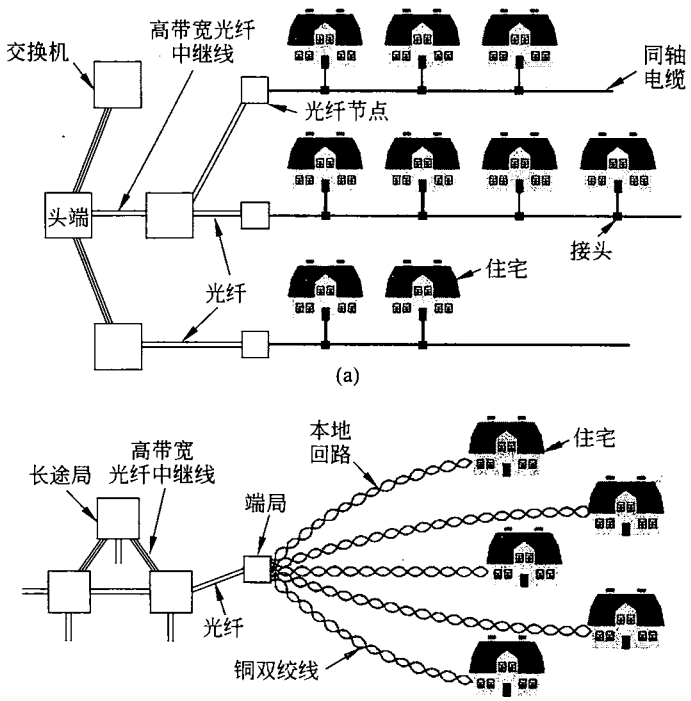


图 2-51
（a）有线电视；（b）固定的电话系统

在过去的10年中，许多有线电视运营商决定进入Internet接入业务，而且通常也涉足电话业务。有线电视设备和电话设备技术上的差异对如何实现上述目标有很大的影响。一方面，系统中的所有单向放大器必须替换成双向放大器，才能支持上行流和下行流的传输。虽然这种情况正在发生，早期线缆上的Internet利用有线电视网络进行下行流的传输，而通过电话网络的拨号连接进行上行流的传输。这是一个聪明的解决办法，但相比网络应该有的样子却还相差甚远。

然而，图2-51（a）所示的HFC系统与图2-51（b）所示的电话系统之间还存在着一个难以克服的差异。在有线电视系统的下游邻居中，许多住户共享同一根同轴电缆；而在电

话系统中，每个住宅都有自己专用的本地回路。当这根电缆被用于电视广播时，这种共享是自然的。因为所有的电视节目都在这根电缆上广播，无论有 10 个用户还是 10 000 个用户都没有什么区别。然后，当同样这根电缆被用作 Internet 接入时，10 个用户与 10 000 个用户的情形就有很大的不同。如果一个用户要下载一个非常大的文件，则他很有可能将其他用户的潜在带宽都拿过来使用了。用户数量越多，带宽的竞争就越激烈。电话系统则没有这个特点，用户通过 ADSL 线路下载一个大文件并不会降低其邻居的带宽。但另一方面，同轴电缆的带宽远远高于双绞线的带宽，因此如果你的邻居不在上网或者不常使用网络，那你就很幸运地能拥有很大的带宽。

有线电视行业解决这个问题的办法是将长的电缆分解开（截短），然后将每一段直接连接到光纤节点上。从头端到每一个光纤节点的带宽几乎是无限的，所以，只要每一段电缆上没有太多的用户，那么流量还是可以管理的。现在一根典型的电缆通常有 500～2000 户家庭；但是，随着越来越多的用户通过电缆连接到 Internet，负载会变得非常大，因而要求更加细粒度地分解电缆，并引入更多的光纤节点。

2.8.3 频谱分配

如果有线电视公司放弃所有的电视频道，将电缆设施完全用于 Internet 接入，则有可能会激怒大量的客户，所以电视公司一直犹豫着。而且，大多数城市对于有线电缆上传送的内容都有严格的管制，因此即使有线电视运营商真的想这样做，它们恐怕也得不到许可。结果是它们需要找到一种办法让电视和 Internet 在同一根电缆上共存。

解决方案的基础是频分多路复用。在北美，有线电视频道通常占用 54～550 MHz 范围（除了 88～108 MHz 用于 FM 无线电台以外）。这些电视频道都是 6 MHz 宽，其中包括保护频段。在欧洲，低端通常在 65 MHz，由于 PAL 和 SECAM 所要求的分辨率较高，所以每个频道为 6～8 MHz 宽；除此以外，在其他方面分配方案非常类似。频带的较低部分并未使用。现代电缆在 550 MHz 以上也能工作得很好，通常可以达到 750 MHz 或者更高。选择的解决方案是在 5～42 MHz 频段（在欧洲还可以更高一些）引入上行流信道，并且使用高端的频率作为下行流信号。电缆的频谱如图 2-52 所示。

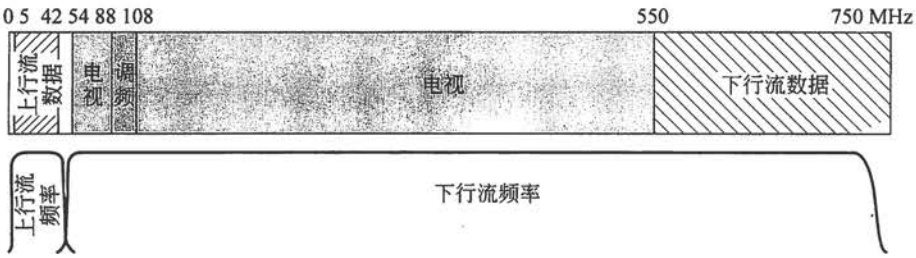


图 2-52 典型有线电视系统中用作 Internet 接入的频率分配

请注意，由于电视信号全部是下行的，所以上行流放大器可以仅工作在 5～42 MHz 区域范围内，下行流放大器仅工作在 54 MHz 以上，如图所示。因此，在这种方案中，因为电视频道之上的可用频段范围大于电视频道之下的频段范围，我们可获得非对称的上行和下行带宽。另一方面，大多数用户希望获得更多的下行流量，因此有线电视运营商也不会

对此感到不快。正如我们以前所看到过的，即使没有任何技术理由，电话公司通常提供一种非对称的 DSL 服务。

除了更换放大器以外，运营商还不得不更换头端，将它从一个哑放大器替换成一个智能的数字计算机系统，并且通过一个高带宽的光纤接口将它连接到 ISP。这样一来，通常连名字也需要更换，从原来的“头端“变成**线缆**[1]**调制解调终端系统**（CMTS，Cable Modem Termination System）。在下面的介绍中，我们将坚持使用传统的“头端”，而不使用更换之后的名字 CMTS。

2.8.4 线缆调制解调器

为了通过有线电视电缆接入 Internet，需要一个**线缆调制解调器**（cable modem），这是一个具有两个接口的设备：一个接口连接计算机，另一个接口连接有线电视网络。在通过有线电视接入 Internet 的早期，每个运营商都有自己专用的线缆调制解调器，该设备必须由有线电视公司的技术员来安装。然而，很快局势就变得明朗起来，一个开放的标准将创造一个有竞争的线缆调制解调器市场，并且可以降低价格，从而鼓励大家使用这项服务。而且，让客户去商店里购买线缆调制解调器并回家自己安装（就好像使用无线接入点一样）还可以避免可怕的上门服务。

因此，较大的有线电视运营商与一家称为 CableLabs 的公司合作，制定了一个线缆调制解调器标准，并且兼顾测试其产品的兼容性。这个标准称为**线缆数据服务接口规范**（DOCSIS，Data Over Cable Service Interface Specification），替代了大多数原来的专用调制解调器。DOCSIS1.0 于 1997 年问世，很快就被 2001 年的 DOCSIS2.0 所取代。该版本增加了上行流的速率以便更好地支持对称服务（比如 IP 电话）。最新的版本是 DOCSIS3.0，于 2006 年发布。为了在两个方向上增加数据速率，新版本使用了更多的带宽。这些标准的欧洲版本称为 EuroDOCSIS。然而，并不是所有的运营商都喜欢标准化的做法，它们中的许多正通过租借调制解调器从已有的客户群中赚取利润。一个开放的标准，以及商店中销售的来自几十家厂商的线缆调制解调器，终结了这些运营商获得利润丰厚的做法。

调制解调器与计算机的接口非常简单，通常是以太网，或者偶尔是 USB。另一端要稍微复杂点，因为它要采用 FDM、TDM 和 CDMA 这全部的复用技术来完成用户对线缆带宽的复用。

当插上线缆调制解调器并加电后，它就扫描下行流信道，寻找一个特殊的数据包；该数据包由头端定期发送，头端通过它向刚刚上线的调制解调器提供系统参数。新的调制解调器找到了该数据包后，就在某个上行信道中宣布自己的存在。头端立即作出响应，为新的调制解调器分配上行和下行信道。以后，如果头端认为有必要均衡负载的话，这些上行和下行信道的分配方案还可以改变。

6 MHz 或 8 MHz 信道的使用属于 FDM 部分。每个线缆调制解调器在一个上行流信道和一个下行流信道上发送数据，如果采用 DOCSIS3.0 则在多个信道上发送数据。通常的方案是针对每个 6 MHz（或 8MHz）下行信道采用 QAM-64 调制方法；如果信道质量超级好，

1 为简明起见，这里将有线电视电缆简称为线缆。

还可以采用 QAM-256 调制方法。在 6 MHz 信道采用 QAM-64 调制方法，可以获得 36 Mbps 的数据率。去掉开销，剩余的网络有效载荷还有 27 Mbps。如果用 QAM-256 调制方法，则网络的有效载荷高达 39Mbps。欧洲地区能获得的有效载荷大概是这个数字的 1/3。

对于上行流，因为系统最初不是为数据通信设计的因而存在较多的 RF 噪声，并且噪声来自多个都往头端发送的用户，所以要采用更为保守的方案。保守方案可以从 QPSK 到 QAM-128 这些复杂性不同的方案中选出，这些方案的一个共性是都把一些符号用作差错保护，并且采用了网格编码调制（Trellis Coded Modulation）。上行流中的每个符号携带较少的比特，上行和下行速率之间的不对称性远远超过由图 2-52 建议的。

然后，多个用户以 TDM 方式来共享上行流的带宽；否则，这些用户传输的信号将在头端产生冲突。时间被细分为**迷你槽**（minislots），不同的用户在不同的迷你槽中发送。为了能够工作，调制解调器必须确定它离头端有多远的距离。具体做法是先发送一个特殊的数据包，再看需要多长时间才得到应答。这个过程称为**测距**（ranging）。对于调制解调器而言，知道自己与头端之间的距离非常重要，这样它可以获得正确的访问时机。在头端接收信号时，每个上行数据包的抵达时间必须符合一个或多个连续的迷你槽。头端周期性地宣布新一轮迷你槽的开始，由于沿着电缆传播的时间并不完全相同，因而所有的调制解调器不可能同时听到头端的发令枪响。通过测算自己离头端的距离，每个调制解调器就可以计算出第一个迷你槽真正开始的确切时间。迷你槽的长度与网络有关，典型的有效载荷长为 8 个字节。

在初始化过程中，头端为每个调制解调器分配一个迷你槽，用来请求上行流信道的带宽。当一台计算机想要发送数据包时，它先将数据包传输给调制解调器；然后调制解调器为该数据包请求一定数量的迷你槽。如果该请求被头端接受，则头端通过下行流信道发送一个确认，告诉该调制解调器已经为它的数据包保留了哪些迷你槽。然后，在所分配的迷你槽开始时，该数据包就可被发送出去。如果还有其他的数据包要发送，则可以利用头部的一个字段来请求。

作为一项规则，多个调制解调器将被分配在同一个迷你槽中，从而产生了竞争。针对这种竞争存在两种不同的可能性。第一个方案是多个用户利用 CDMA 来共享迷你槽。这样就解决了竞争问题，因为拥有各自 CDMA 码片序列的所有用户可以在同一时间发送，虽然速率会降低。

第二个方案是不用 CDMA。由于在头端产生冲突，相应的请求将得不到确认。在这种情况下，调制解调器等待一个随机时间后再次尝试。每次失败后，等待再次尝试的随机时间增加一倍（对于有点熟悉网络的读者知道这个算法就是带有二进制指数回退的分槽 ALOHA。以太网不能用在电缆上，因为有线电视无法感知介质。我们在第 4 章还会回来讨论这些问题）。

下行信道的管理与上行信道不同。首先，下行信道只有一个发送者（头端），所以不会发生竞争，因此不需要迷你槽，实际上下行信道只是一个时分统计多路复用。其次，下行流量通常远远大于上行流量，所以下行信道使用了 204 个字节的固定数据包长度。在这些字节中，包含了 Reed-Solomon 纠错码和其他一些开销，留给用户的有效载荷只有 184 个字节。之所以选择这些数字，是考虑到与 MPEG-2 数字电视保持兼容，所以，电视信道和下行数据信道使用了同样的格式化方法。从逻辑上讲，连接如图 2-53 所示。

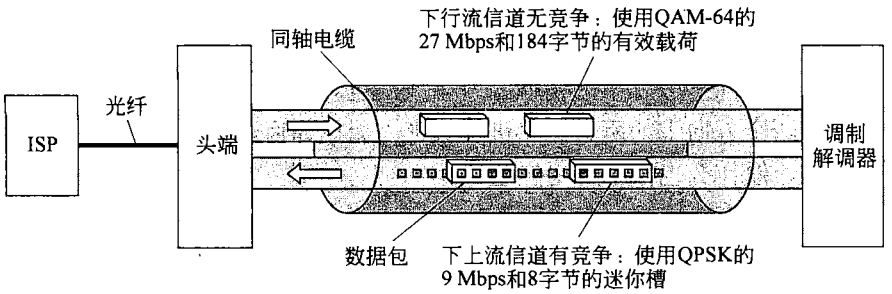


图 2-53 北美地区上行流和下行流信道的典型细节

2.8.5 ADSL 与有线电视电缆

ADSL 与有线电视电缆哪个更好？这就好像在问哪一个操作系统更好，或哪一种语言更好，或哪一种信仰更好一样。你会得到哪一个答案要取决于你问的是对象是谁。下面我们从几个方面来比较一下 ADSL 和有线电缆。在骨干网上，两者都使用了光纤，但是在网络边缘它们使用了不同的介质。有线电缆使用了同轴电缆，而 ADSL 则使用了双绞线。从理论上讲，同轴电缆的承载容量超过双绞线几百倍。然而，有线电视电缆的全部容量并不能都被数据用户使用，因为电缆的大部分带宽被浪费在诸如电视节目这样的无关素材上。

实际上，有线电视运营商很难就有效容量一概而论。ADSL 运营商可以明确声明带宽（例如，下行流量 1 Mbps，上行流量 256 kbps），而且通常也可以达到其声明的 80%。有线电视运营商可能人为地给每个用户设置一个带宽上限，以便帮助他们自己确定预期的性能，但它们却无法作出任何带宽保证，因为有效容量取决于当前有多少人连接在同一段电缆上。有时候它可能比 ADSL 好，有时候可能还不如 ADSL。不过，真正令人讨厌的是它的不可预测性。上一分钟有很好的服务并不能保证下一分钟还能得到同样好的服务，因为镇上的那个最大带宽黑洞可能刚刚打开了他的计算机。

ADSL 系统赢得了更多的用户，新增加的用户对原来的用户几乎没有影响，因为每个用户都有自己专用的连接。但在有线电视网络中，当申请 Internet 服务的用户越来越多时，原有用户的性能就会下降。唯一的解决办法是运营商将繁忙的电缆拆分成多段，然后将每一段直接连接到光纤节点上。这样做既耗时也耗财，所以，出于业务的压力应该尽量避免。

顺便提一句，我们早就学习过的移动电话系统跟有线电缆类似，都有一个共享的信道。在这个系统中也有一群用户共享固定数量的带宽，我们将这群用户称为蜂窝伴侣（cellmate）。对于相当平滑的语音流量，系统利用 FDM 和 TDM 把带宽硬性划分成固定大小的块，然后分配给当前的活跃用户。但是，对于数据流量，这种带宽的硬性划分非常低效，因为数据用户经常会空闲，此时为他们保留的带宽就完全被浪费掉了。至于有线电缆，采用了一种更为动态的手段来分配共享带宽。

可用性也是 ADSL 和有线电视电缆不一样的地方。每个人都有一部电话，但是并不是所有的人离端局都足够近到可以申请 ADSL。另一方面，并不是所有的人都有有线电视，但是，如果你已经有了有线电视，并且有线电视公司确实提供 Internet 接入服务，那么，你就可以使用这项服务。离光纤节点或者头端的距离不是个问题。同时值得注意的是，由

于有线电缆最初是作为传播介质建立起来的，所以它承载的商业服务非常少。

ADSL 作为一种点到点的传输介质，本质上比电视电缆安全得多。任何有线电视用户都可以很容易地读取同一根电缆上传输的所有数据包。出于这个原因，任何正规的有线电视运营商都会对两个方向上的所有流量进行加密。无论如何，与其让你的邻居得到一个加密消息，还不如让他什么也得不到更安全。

电话系统一般比有线电视系统更加可靠。例如，它有备份电源，即使停电了它还能继续工作。而在有线电视系统中，如果沿途任何一个放大器的电源断了，则所有位于它下游的用户立即就会被中断连接。

最后，绝大多数 ADSL 运营商都允许用户选择 ISP。有时候，甚至法律要求它们这样做。但是，对于有线电视运营商，事情并不总是这样的。

结论是 ADSL 和有线电缆的相同性多于差异性。它们提供类似的服务，而且，随着两者之间竞争的白热化，它们的服务价格或许也会进入竞争阶段。

2.9 本章总结

物理层是所有网络的基础。物理性质给所有信道强加了两个根本限制，而这些限制决定了它们的带宽。这些限制分别是处理无噪声信道的尼奎斯特极限和处理有噪声信道的香农极限。

传输介质可以是引导性的或非引导性的。主要的引导性介质有双绞线、同轴电缆和光纤。非引导介质包括地面无线电、微波、红外线、通过空气传输的激光和卫星。

数字调制方式可以通过引导性和非引导性介质上的模拟信号来发送比特。线性编码以基带方式运行，通过调节载波的振幅、频率和相位把信号放置到一个通带上。信道可以时分、频分和码分复用的方式被多个用户共享使用。

大多数广域网络的关键元素是电话系统。电话系统的主要组件有本地回路、中继线和交换机。ADSL 在本地回路上可提供高达 40 Mbps 的数据率，具体做法是将本地回路分割成许多个可同时运行的子载波。这样的速度远远超过了电话调制解调器的速度。PON 将光纤引入住户，可提供比 ADSL 还要高的接入速率。

中继线传送数字信息。用 WDM 对光纤进行多路复用就可以在其上提供许多条高容量链路，同时通过 TDM 使得许多用户共享每条高速链路。电路交换和包交换都很重要。

对于移动应用而言，固定电话系统显然并不适用。移动电话在目前被广泛应用于语音通信，并越来越多地用于数据通信。它们已经经历了三代。第一代 1G 是模拟的，由 AMPS 主宰。第二代 2G 是数字化，目前在全球部署最广泛的移动电话系统是 GSM。第三代 3G 是数字的并且以宽带 CDMA 为基础，现在正在部署的有 WCDMA 和 CDMA2000。

另一种网络接入系统是有线电视系统。它已逐渐从同轴电缆演变为混合光纤同轴电缆，从单纯的电视演进为电视和 Internet。它的潜在带宽很高，但实际带宽则主要取决于其他用户，因为它是共享式的。

习 题

1. 计算函数 f(t)=t(0≤t≤1)的傅里叶系数。
2. 每 1 毫秒对一条无噪声 4 kHz 信道采样一次。试问最大数据传输率是多少？如果信道上有噪声，且信噪比是 30 dB，试问最大数据速率将如何变化？
3. 电视信道宽 6 MHz。如果使用 4 级数字信号，试问每秒可发送多少个比特？假设电视信道为无噪声的。
4. 如果在一条 3 kHz 的信道上发送一个二进制信号，该信道的信噪比为 20 dB，试问可达到的最大数据率为多少？
5. 试问在 50 kHz 的线路上使用 T1 载波需要多大的信噪比？
6. 试问光纤作为传输介质，相比铜芯有什么优势？是否存在不足？
7. 试问在 0.1 微米频谱上 1 微米波长的带宽是多少？
8. 现在需要在一条光纤上发送一系列的计算机屏幕图像。屏幕的分辨率为 2560×1600 像素，每个像素 24 比特。每秒钟产生 60 幅屏幕图像。试问需要多少带宽？在 1.30 微米波段需要多少微米的波长？
9. 试问尼奎斯特定理对高质量的单模光纤成立吗？还是它只适用于铜线？
10. 当无线电天线的直径等于无线电波波长时天线通常工作得最好。常见的天线直径范围为 1～5 米。试问这将覆盖多大的频率范围？
11. 一束 1 毫米宽的激光对准了 100 米外建筑物顶上的一个 1 毫米宽探测器。试问若要使该激光束对准探测器，则激光束必须小于多大的角度？
12. 铱计划中的 66 颗低轨道卫星被分成绕着地球的六条链。它们在自己的高度绕地球一圈周期是 90 分钟。试问对于一个固定的发射器，切换的平均间隔是多少？
13. 试分别计算在两个 GEO（高度：35 800 千米）、MEO（高度：18 000 千米）和 LEO（高度：750 千米）卫星之间一个数据包的端-端传输时间。
14. 如果使用铱星卫星通信，试问从北极发出一个电话呼叫到达南极的延迟是多少？假设卫星交换时间是 10 微秒，地球半径为 6371 千米。
15. 如果信号传输使用 NRZ、MLT-3 和曼彻斯特编码，试问为了达到 B bps 速率，至少需要多少带宽？请解释你的答案。
16. 试证明在 4B/5B 编码模式中，至少每 4 个比特时间要发生一次信号跳变。
17. 1984 年以前每个端局由三位数字的区域号和本地号码中的前三位命名，试问那时共有多少个电话端局？区域号数字由 2～9 开始，第二位是 0 或者 1，最后一位数字任意取值。本地号码的前两个数字总在 2～9 范围内，第三个数字可以是任何数字。
18. 一个简单的电话系统包括两个端局和一个长途局，每个端局通过一条 1 MHz 的全双工中继线连接到长途局。在每 8 小时的工作日中，平均每部电话发出 4 次呼叫，每次呼叫平均持续 6 分钟，并且 10%的呼叫是长途（即要通过长途局）。试问端局最多能支持多少部电话（假设每条电路为 4 kHz）？请解释为什么电话公司决定支持的电话数要少于端局的最大电话数？

19. 一个区域电话公司有1000万个用户。每部电话通过双绞线连接到一个中心局。这些双绞线的平均长度为10千米。试问本地回路中的铜价值多少？假设每股线的横截面直径为1毫米，铜的密度是9.0克/立方厘米，并且每千克铜可以卖6美元。
20. 试问石油管道是单工系统、半双工系统还是全双工系统？或者三者都不是？河流或者类似对讲机的通信是什么系统？
21. 高速微处理器的价格已经降到了可以在每个调制解调器中都安装一个的程度。试问这对电话线路的错误处理有什么影响？它能否决第二层差错检测/纠正的需求？
22. 一个类似于图2-23的调制解调器星座图有以下几个数据点：(1, 1)、(1, −1)、(−1, 1)和(−1, −1)。试问一个具备这些参数的调制解调器以1200符号/秒的速率能获得多少bps？
23. 如果波特率是1200并且不需要差错检测，试问V.32标准调制解调器能达到的最大比特率是多少？
24. 试问一个全双工QAM-64调制解调器使用了多少频率？
25. 有10个信号，每个需要4000 Hz带宽，现在用FDM将它们复用在一条信道上。试问对于被复用的信道，需要的最小带宽是多少？假设保护带为400 Hz宽。
26. 试问为什么PCM采样时间被设置为125微妙？
27. 试问T1载波的百分比开销为多少？也就是说，1.544 Mbps中有百分之多少没有给端用户使用？OC-1或0C-768线路的百分比开销又是多少？
28. 若将无噪声的4 kHz信道用于下面的用途，请比较它们的最大数据传输率：
    （a）每个样值2比特的模拟编码（比如QPSK）。
    （b）T1 PCM系统。
29. 如果一个T1载波系统失去同步不知道自己在哪里，它试图在每一帧的第1位重新同步。试问在出错概率为0.001的情况下，平均要检查多少帧才能重新获得同步。
30. 试问调制解调器的解调部分与编码解码器的编码部分有没有区别？如果有的话，区别是什么？（毕竟两者都将模拟信号转换成数字信号）
31. SONET时钟的漂移率大约是1/109。试问，经过多长时间才能使得漂移等于1比特的宽度？该结果有什么实际含义吗？若有，请解释之。
32. 使用如图2-17所示的集线器，试问需要多长时间才能把一个1 GB的文件从一个VSAT发送到另一个？假设上行链路是1 Mbps，下行链路是7 Mbps，采用电路交换技术，电路的建立时间是1.2秒。
33. 在上题中，如果采用包交换，试问数据包的传输时间是多少？假设数据包长度为64 KB，在卫星和集线器上的交换时延是10微秒，数据包的包头大小为32字节。
34. 在图2-40中，OC-3用户的数据传输率规定为148.608 Mbps。试问该数值是如何从SONET OC-3的参数得出的。对于OC-3072线路来说，SPE和用户数据率是多少？
35. 为适应比STS-1低的速率，SONET有一个虚拟支流（VT，virtual tribntary）系统。一个VT是指被插入到STS-1帧中的部分有效载荷，并且可以与其他部分有效载荷组合起来填满数据帧。VT1.5使用STS-1帧的3列，VT2使用4列，VT3使用6列，VT6使用12列。试问，哪个VT可以满足：
    （a）DS-1服务（1.544 Mbps）？
    （b）欧洲的CEPT-1服务（2.048 Mbps）？

（c）DS-2 服务（6.312 Mbps）？

36. 试问一个 OC-12c 连接的用户可用带宽是多少？
37. 有三个包交换网络，每个包含 n 个节点。第一个网络采用星形拓扑结构，有一个中心交换机；第二个网络采用双向环结构；第三个网络则采用全连通结构，每个节点都有一条线路与其他的每个节点相连。试问，从传输路径的跳数来看，哪个最好？哪个其次？哪个最差？
38. 比较在一个电路交换网络和一个（负载较轻的）包交换网络中，沿着 k 跳路径发送一个 x 位长度消息的延迟。假设电路建立时间为 s 秒，每一跳的传播延迟为 d 秒，数据包的大小为 p 位，数据传输率为 b bps。试问在什么条件下数据包网络的延迟比较短？请解释之。
39. 假定在一个包交换网络中用户数据长度为 x 位，将以一系列数据包的形式沿着一条 k 跳路径传输，每个数据包包含 p 位数据和 h 位头，这里 $x >> p + h$。线路的比特率为 b bps，传播延迟忽略不计。试问什么样的 p 值使得总延迟最小？
40. 在一个六角形蜂窝的典型移动电话系统中，不允许相邻蜂窝重复使用频段。如果总共有 840 个频率可用，试问对于一个给定的蜂窝最多可以使用多少个频率？
41. 蜂窝的实际布局很少像图 2-45 那样规则。即使单个蜂窝的形状也往往是不规则的。试给出一个可能的理由说明为什么会这样？这些不规则形状对每个蜂窝的频率分配有什么影响？
42. 为了覆盖旧金山（120 平方千米），请粗略估算需要多少个直径为 100 米的 PCS 微蜂窝？
43. 有时当一个移动用户从一个蜂窝边界跨越进入另一个蜂窝时，当前的电话呼叫会被突然中止，即使所有的发射器和接收器都在正常工作。试问这是为什么？
44. 假设在一个 CDMA 系统中，A、B 和 C 同时传输比特 0，它们的码片序列如图 2-28（a）所示。试问结果码片序列是什么？
45. 考虑用另一种方式来看待 CDMA 码片序列的正交特性。一对序列中的每一位要么匹配，要么不匹配。试按照匹配和不匹配来表示正交特性。
46. 一个 CDMA 接收器得到了下面的码片：（-1+1-3+1-1-3+1+1）。假设码片序列如图 2-28（a）所定义，试问哪些移动站传输了数据？每个站发送了什么比特？
47. 在图 2-28 中，有 4 个站可以传输。假设增加了 4 个站，试给出这些站的码片序列。
48. 在低端，电话系统呈星型结构，邻近范围内的所有本地回路都集中到端局。相反，有线电视网的低端则使用一条长电缆蜿蜒穿过邻近范围内的所有住户。假设未来的有线电视电缆是 10 Gbps 的光纤，而不再是铜线。试问，它可以模拟电话模型，即每个住户都有自己的专用线路连接到端局吗？如果可以的话，试问一根光纤上可以挂接多少个只有一部电话的住户？
49. 一个有线电视公司决定为一个有 5000 个住户的区域提供 Internet 接入服务。该公司使用一根同轴电缆，它的频谱分配方案允许每根电缆有 100 Mbps 的下行带宽。为了吸引客户，公司决定在任何时候都保证每个住户至少有 2 Mbps 的下行带宽。试问该公司需要采取什么措施才能提供这样的带宽保证。
50. 利用图 2-52 显示的频谱分配方案以及课本中给出的信息，试问一个有线电视系统分配的上行流带宽和下行流带宽分别是多少个 Mbps？

51. 如果网络空闲，一个有线电视用户的接收数据率是多少？假设用户接口分别是：
    （a）10 Mbps 以太网。
    （b）100 Mbps 以太网。
    （c）54 Mbps 无线局域网。
52. 多个 STS-1 数据流的复用在 SONET 中扮演了非常重要的角色，这些 STS-1 数据流称为支流（tributary）。一个 3：1 多路复用器将三个输入的 STS-1 支流复用到一个 STS-3 输出流中。复用过程按字节进行，也就是说，前三个输出字节分别是支流 1、2 和 3 的第一个字节；接下去的三个字节分别是支流 1、2 和 3 的第二个字节；以此类推。请编写一个程序来模拟这样的 3：1 多路复用器。程序应该包含 5 个进程。1 个主进程创建 4 个进程，其中三个进程分别对应于三个 STS-1 支流，另一个对应于多路复用器。每个支流进程从一个输入文件中读入连续的 810 字节作为一个 STS-1 帧，它们将这些帧（逐个字节）发送给多路复用器进程。多路复用器进程接收这些字节，然后一个字节一个字节地输出到标准输出设备上。进程之间的通信使用管道。
53. 写一个实现 CDMA 的程序。假设，码片序列的长度是 8，发射站的数目为 4。程序应该包括三组进程：4 个发射进程（t0、t1、t2 和 t3）、一个联结进程和四个接收器进程（r0、r1、r2 和 r3）。主程序，同时作为联结进程，首先从标准输入设备读入 4 个码片序列（双极表示）和一个 4 比特的序列（每个发射进程负责发射 1 个比特），并且派生出 4 对发射和接收子进程。每对发射/接收进程（t0,r0；t1,r1；t2,r2；t3,r3）分配得到一个码片序列，每个发射进程还分配得到 1 个比特（第一个比特分配给 t0，第二个比特给 t1，以此类推）。然后，每个发射进程计算它要发射的信号（8 比特的序列），并将该信号发送到联结进程。在收到全部四个发射进程发来的信号后，联结进程把这些信号组合起来；然后把组合后的信号发射到 4 个接收器，并打印到标准输出。进程之间的通信采用管道形式进行。

第 3 章　数据链路层

在本章，我们将学习网络模型中的第二层（即数据链路层）的设计原则。学习内容涉及两台相邻机器实现可靠有效的完整信息块（称为帧）通信的一些算法，而不像物理层那样只关注单个比特传输。这里的相邻指两台机器通过一条通信信道连接起来，通信信道在概念上就像一条线路（比如同轴电缆、电话线或者无线信道）。信道像一条线路的本质特性使得信道上传递的比特顺序与发送顺序完全相同。

刚开始，你可能认为这个问题非常简单，似乎没有什么内容需要学习——机器 A 把比特放到线路上，然后机器 B 将这些比特取下来。不幸的是，通信线路偶尔会出错。而且，它们只有有限的数据传输率，并且在比特的发送时间和接收时间之间存在一个非零延迟。这些限制对数据传输的效率有非常重要的影响。通信所采用的协议必须考虑所有这些因素。这些协议正是本章的主题。

在介绍了数据链路层的关键设计问题之后，我们将通过考察错误的本质以及如何检测和纠正这些错误来开始数据链路层协议的学习。然后，我们将学习一系列复杂性逐步递增的协议，每个协议解决了本层中越来越多的问题。最后，我们将给出一些数据链路层协议的例子来结束本章。

3.1　数据链路层的设计问题

数据链路层使用物理层提供的服务在通信信道上发送和接收比特。它要完成一些功能，包括：

（1）向网络层提供一个定义良好的服务接口。

（2）处理传输错误。

（3）调节数据流，确保慢速的接收方不会被快速的发送方淹没。

为了实现这些目标，数据链路层从网络层获得数据包，然后将这些数据包封装成帧（frame）以便传输。每个帧包含一个帧头、一个有效载荷（用于存放数据包）以及一个帧尾，如图 3-1 所示。帧的管理构成了数据链路层工作的核心。在后面的章节中，我们将详细地讨论上面提到的这些问题。

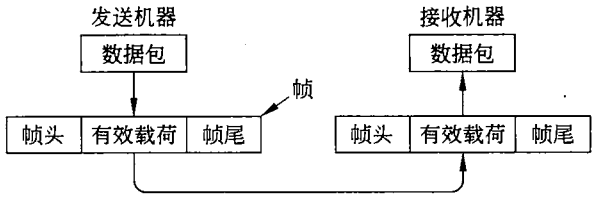


图 3-1　数据包和帧的关系

虽然本章明确讨论数据链路层及其协议，但是，我们在本章中学习的许多原理，比如错误控制和流量控制等同样可以在传输层和其他协议中寻觅到类似的踪迹。这是因为可靠性是网络的总目标，这个目标的实现需要各层次的紧密配合。实际上，在许多网络中，这些功能最常出现的地方是上层，数据链路层只要做很少的一点工作就已经“足够好”。然而，不管它们出现在哪里，原理是非常一致的。在数据链路层中，它们通常表现出最为简单和纯粹的形式，因此，数据链路层是详细学习这些原理的绝佳之地。

3.1.1 提供给网络层的服务

数据链路层的功能是为网络层提供服务。最主要的服务是将数据从源机器的网络层传输到目标机器的网络层。在源机器的网络层有一个实体（称为进程），它将一些比特交给数据链路层，要求传输到目标机器。数据链路层的任务就是将这些比特传输给目标机器，然后再进一步交付给网络层，如图3-2（a）所示。实际的传输过程则是沿着图3-2（b）所示的路径进行的，但很容易将这个过程想象成两个数据链路层的进程使用一个数据链路协议进行通信。基于这个原因，在本章中我们将隐式使用图3-2（a）的模型。

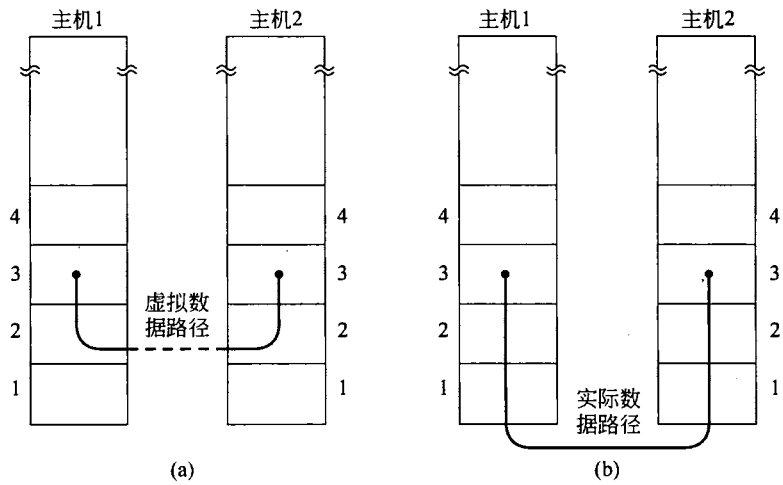


图 3-2
（a）虚拟通信；（b）实际通信

数据链路层可以设计成向上提供各种不同的服务。实际提供的服务因具体协议的不同而有所差异。一般情况下，数据链路层通常会提供以下3种可能的服务：

（1）无确认的无连接服务。

（2）有确认的无连接服务。

（3）有确认的有连接服务。

无确认的无连接服务是指源机器向目标机器发送独立的帧，目标机器并不对这些帧进行确认。以太网就是一个提供此类服务的数据链路层极好实例。采用这种服务，事先不需要建立逻辑连接，事后也不用释放逻辑连接。若由于线路的噪声而造成了某一帧的丢失，数据链路层并不试图去检测这样的丢帧情况，更不会去试图恢复丢失的帧。这类服务合适两种场合，第一种是错误率很低的场合，此时差错恢复过程可以留给上层来完成；第二种

是实时通信，比如语音传输，因为在实时通信中数据迟到比数据受损更糟糕。

迈向可靠性的下一步是有确认的无连接服务。当向网络层提供这种服务时，数据链路层仍然没有使用逻辑连接，但其发送的每一帧都需要单独确认。这样，发送方可知道一个帧是否已经正确地到达目的地。如果一个帧在指定的时间间隔内还没有到达，则发送方将再次发送该帧。这类服务尤其适用于不可靠的信道，比如无线系统。802.11（WiFi）就是此类服务的一个很好例子。

或许有一点值得强调，那就是在数据链路层提供确认只是一种优化手段，永远不应该成为一种需求。网络层总是可以发送一个数据包，然后等待该数据包被确认。如果在计时器超时之前，该数据包的确认还没有到来，那么发送方只要再次发送整个报文即可。这一策略的麻烦在于它可能导致传输的低效率。链路层对帧通常有严格的长度限制，这是由硬件所决定的；除此之外，还有传播延迟。但网络层并不清楚这些参数。网络层可能发出了一个很大的数据包，该数据包被拆分并封装到（比如说）10 个帧中，而且 20%的帧在传输中被丢失，那么这个数据包可能需要花很长的时间才能传到接收方。相反地，如果每个帧都单独确认和必要时重传，那么出现的差错就能更直接并且更快地被检测到。在可靠信道上，比如光纤，重量级数据链路协议的开销可能是不必要的；但在无线信道上，由于信道内在的不可靠性，这种开销还是非常值得的。

我们再回到有关服务的话题上，数据链路层向网络层提供的最复杂服务是面向连接的服务。采用这种服务，源机器和目标机器在传输任何数据之前要建立一个连接。连接上发送的每一帧都被编号，数据链路层确保发出的每个帧都会真正被接收方收到。它还保证每个帧只被接收一次，并且所有的帧都将按正确的顺序被接收。因此，面向连接的服务相当于为网络层进程提供了一个可靠的比特流。它适用于长距离且不可靠的链路，比如卫星信道或者长途电话电路。如果采用有确认的无连接服务，可以想象丢失了确认可能导致一个帧被收发多次，因而将浪费带宽。

当使用面向连接的服务时，数据传输必须经过三个不同的阶段。在第一个阶段，要建立连接，双方初始化各种变量和计数器，这些变量和计数器记录了哪些帧已经接收到，哪些帧还没有收到。在第二个阶段，才真正传输一个或者多个数据帧。在第三个也是最后一个阶段中，连接被释放，所有的变量、缓冲区以及其他用于维护该连接的资源也随之被释放。

3.1.2　成帧

为了向网络层提供服务，数据链路层必须使用物理层提供给它的服务。物理层所做的只是接收一个原始比特流，并试图将它传递给目标机器。如果信道上存在噪声，就像大多数无线链路和某些有线链路那样，物理层就会在它的信号中添加某种冗余，以便将误码率降到一定程度。然而，数据链路层接收到的比特流不能保证没有错误。某些比特的值可能已经发生变化，接收到的比特个数可能少于、等于或者多于发送的比特数量。检测错误和纠正错误（有必要的话）的工作正是数据链路层该做的。

对于数据链路层来说，通常的做法是将比特流拆分成多个离散的帧，为每个帧计算一个称为校验和的短令牌（本章后面将讨论校验和算法），并将该校验和放在帧中一起传输。当帧到达目标机器时，要重新计算该帧的校验和。如果新算出来的校验和与该帧中包含的

校验和不同，则数据链路层知道传输过程中产生了错误，它就会采取措施来处理错误（比如丢掉坏帧，可能还会发回一个错误报告）。

拆分比特流的实际工作比初看上去的要复杂得多。一个好的设计方案必须使接收方很容易发现一个新帧的开始，同时所使用的信道带宽要少。我们将考察下列4种方法：

（1）字节计数法。

（2）字节填充的标志字节法。

（3）比特填充的标志比特法。

（4）物理层编码违禁法。

第一种成帧方法利用头部中的一个字段来标识该帧中的字符数。当接收方的数据链路层看到字符计数值时，它就知道后面跟着多少个字节，因此也就知道了该帧在哪里结束。这项技术如图3-3（a）所示，其中4帧的大小分别为5、5、8和8个字节。

这个算法的问题在于计数值有可能因为一个传输错误而被弄混。例如，如果第2帧中的计数值5由于一个比特反转而变成了7，如图3-3（b）所示，则接收方就会失去同步，它再也不可能找到下一帧的正确起始位置。即使校验和不正确，接收方知道该帧已经被损坏，它仍然无法知道下一帧从哪里开始。在这种情况下，给发送方发回一个帧，要求重传也无济于事，因为接收方并不知道应该跳过多少个字节才能到达重传的开始处。正是由于这个原因，字节计数方法本身很少被使用。

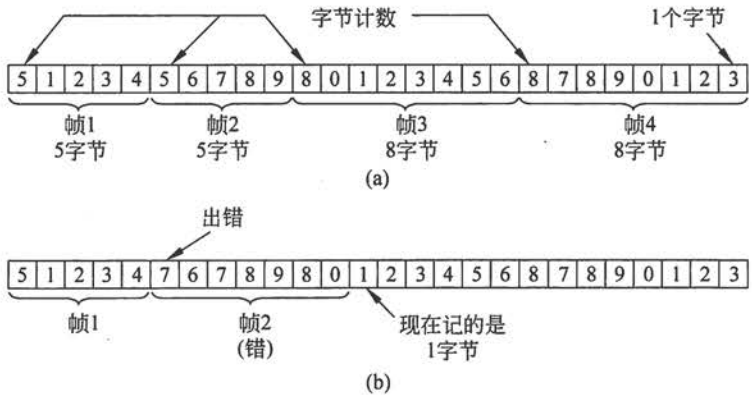


图3-3 字节流

（a）没有错误；（b）有一个错误

第二种成帧方法考虑到了出错之后的重新同步问题，它让每个帧用一些特殊的字节作为开始和结束。这些特殊字节通常都相同，称为**标志字节**（flag byte），作为帧的起始和结束分界符，如图3-4（a）中的FLAG所示。两个连续的标志字节代表了一帧的结束和下一帧的开始。因此，如果接收方丢失了同步，它只需搜索两个标志字节就能找到当前帧的结束和下一帧的开始位置。

然而，还有问题必须要解决。当标志字节出现在数据中时，尤其是当传输二进制数据（比如照片或歌曲）时，这种情景往往会严重干扰到帧的分界。有一种方法可以解决这个问题，发送方的数据链路层在数据中“偶尔”出现的每个标志字节的前面插入一个特殊的转义字节（ESC）。因此，只要看它数据中标志字节的前面有没有转义字节，就可以把作为帧分界符的标志字节与数据中出现的标志字节区分开来。接收方的数据链路层在将数据传递

给网络层之前必须删除转义字节。这种技术就称为**字节填充**（byte stuffing）。

当然，接下来的问题就是：如果转义字节也出现在数据中，那该怎么办？答案是同样用字节填充技术，即用一个转义字节来填充。在接收方，第一个转义字节被删除，留下紧跟在它后面的数据字节（或许是另一个转义字节或者标志字节）。图 3-4（b）给出了一些例子。在所有情况下，去掉填充字节之后递交给网络层的字节序列与原始的字节序列完全一致。我们仍然可以通过搜索两个标志字节来定位帧的边界，无须顾虑撤销转义字节的原意。

图 3-4 中描述的字节填充方案是 **PPP 协议**（Point-to-Point Protocol）使用的略微简化形式，该协议通常用在通信链路上传送数据包。我们将在本章后面讨论 PPP 协议。

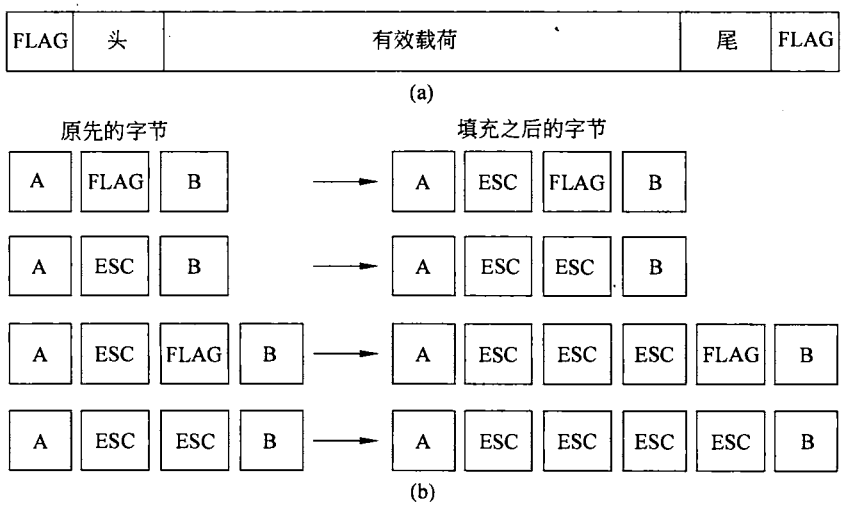


图　3-4

（a）由标志字节分界的帧；（b）字节填充之前和之后的字节序列示例

第三种区分比特流边界的方法考虑了字节填充的缺点，即只能使用 8 比特的字节。帧的划分可以在比特级完成，因而帧可以包含由任意大小单元（而不是只能以 8 比特为单元）组成的二进制比特数。这种方法是为曾经非常流行的 HDLC（高级数据链路控制）协议而开发的。每个帧的开始和结束由一个特殊的比特模式，01111110 或十六进制 0x7E 标记。这种模式是一个标志字节。每当发送方的数据链路层在数据中遇到连续五个 1，它便自动在输出的比特流中填入一个比特 0。这种比特填充类似于字节填充，在数据字段的标志字节之前插入一个转义字节到出境字符流中。比特填充还确保了转换的最小密度，这将有助于物理层保持同步。正是由于这个原因，USB（通用串行总线）采用了比特填充技术。

当接收方看到 5 个连续入境比特 1，并且后面紧跟一个比特 0，它就自动剔除（即删除）比特 0。比特填充和字节填充一样，对两台计算机上的网络层是完全透明的。如果用户数据中包含了标志模式 01111110，这个标志传输出去的是 011111010，但在接收方内存中存储还是 01111110。图 3-5 给出了一个比特填充的例子。

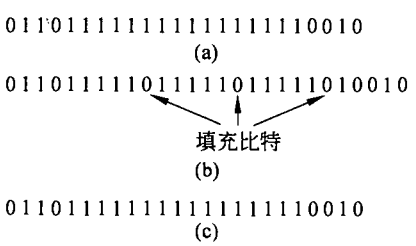


图 3-5　比特填充

（a）原始数据；（b）出现在线路上的数据；（c）存储在接收方内存中的数据

有了比特填充技术，两帧之间的边界可以由标志模式明确区分。因此，如果接收方失去了它的接收轨迹，它所要做的只是扫描输入比特流，找出其中的标志序列；因为这些标志只可能出现在帧的边界，而决不会出现在帧内的数据中。

采用比特填充和字节填充的一个副作用是一帧的长度现在要取决于它所携带的数据内容。例如，如果数据中没有标记字节，100 个字节或许被一个大约长为 100 字节的帧所携带。然而，如果数据完全由标志字节组成，每个标志字节都要被转义，那么最后发送出去帧的长度就变成大约 200 个字节。采用比特填充技术，帧的长度增幅大约为 12.5%，因为每个字节增加 1 个比特。

成帧的最后一种方法是一条使用物理层的捷径。我们在第 2 章看到比特编码成信号通常包括一些冗余比特，以便帮助接收器同步接收。这种冗余意味着一些信号将不会出现在常规数据中。例如，在 4B/5B 线性编码模式下，4 个数据位被映射成 5 个信号比特，通过这种方法确保线路上的信号有足够的跳变。这意味着 32 个可能的信号中有 16 个是不会被使用的。我们可以利用这些保留的信号来指示帧的开始和结束。实际上，我们使用“编码违法”来区分帧的边界。这种方案的优点在于，因为这些用作分界符的信号是保留不用的，所以很容易通过它们找到帧的开始和结束，而且不再需要填充数据。

许多数据链路协议为安全起见综合使用了这些方法。以太网和 802.11 使用了共同的分界模式，即用一个定义良好的比特模式来标识一帧的开始，该比特模式称为**前导码**（preamble）。这种定界模式可能很长（802.11 典型使用 72 位），目的是让接收方准备接收输入的数据包。前导码之后是头的长度字段（即计数），这个字段将被用来定位帧的结束处。

3.1.3 差错控制

解决了如何标识每一帧的起始和结束位置之后，我们现在来看下一个问题：如何确保所有的帧最终都被传递给目标机器的网络层，并且保持正确的顺序。现在假设接收方可以知道它收到的帧包含了正确的或者错误的信息（我们将在 3.2 节考察用于检测和纠正传输错误的编码）。对于无确认的无连接服务，不管发出去的帧是否正确抵达目标机器，发送方只要把出境帧留存就可以了。但是对于可靠的、面向连接的服务，这样做肯定还远远不够。

确保可靠传递的常用方法是向发送方提供一些有关线路另一端状况的反馈信息。通常情况下，协议要求接收方发回一些特殊的控制帧，在这些控制帧中，对于它所接收到的帧进行肯定的或者否定的确认。如果发送方收到了关于某一帧的肯定确认，那么它就知道这帧已经安全地到达了。另一方面，否定的确认意味着传输过程中产生了错误，所以这帧必须重传。

更为复杂的是存在这样的可能性，有时候由于硬件的问题，一个帧被完全丢失了（比如一个突发噪声）。在这种情况下，接收方根本不会有任何反应，因为它没有根据做出反应。类似地，如果确认帧丢失，发送方也不知道该如何处理。显然，如果在一个协议中，发送方发出了一帧之后就等待肯定的或者否定的确认，那么，若由于硬件故障或通信信道出错等原因而丢失了某一帧，则发送方将永远等待下去。

这种可能性可以通过在数据链路层中引入计时器来解决。当发送方发出一帧时，通常

还要启动一个计时器。该计时器的超时值应该设置得足够长，以便保证在正常情况下该帧能够到达接收方，并且在接收方进行处理后再将确认返回到发送方。一般情况下，在计时器超时前，该帧应该被正确地接收，并且确认帧也被传了回来。这种情况下，计时器被取消。

然而，如果帧或者确认被丢失，则计时器将被触发，从而警告发送方存在一个潜在的问题。一种显然的解决方案是重新发送该帧。然而，当有的帧被发送了多次之后，可能会出现这样的危险：接收方将两次或者多次接收到同一帧，并且多次将它传递给网络层。为了避免发生这样的情形，一般有必要给发送出去的帧分配序号，这样接收方可以根据帧的序号来有效区分原始帧和重传帧。

管理好计时器和序号，以便保证每一帧最终都恰好一次地被传递给目标机器的网络层，这是数据链路层（以及上层）工作的重要组成部分。在本章后面，我们将通过一系列复杂性逐渐增加的例子，来考察如何做好计时器和序号的管理工作。

3.1.4　流量控制

在数据链路层（以及更高的各层）中，另一个重要的设计问题是如果发送方发送帧的速度超过了接收方能够接受这些帧的速度，发送方该如何处理。当发送方运行在一台高速并能量强大的计算机上，而接收方运行在一台慢速并且低端机器上时，这种情况很容易发生。一种常见的场景是一个智能手机向一个超强服务器请求一个 Web 页面。这就像突然打开了消防栓一样，大量的数据涌向可怜无助的手机，直到它被彻底淹没。即使传输过程不会出错，接收方也无法以数据到来的速度那样快地处理持续到来的帧，此时必然会丢弃一些帧。

很显然，必须要采取某种措施来阻止这种情况发生。常用的办法有两种。第一种方法是**基于反馈的流量控制**（feedback-based flow control），接收方给发送方返回信息，允许它发送更多的数据，或者至少告诉发送方自己的情况怎么样。第二种方法是**基于速率的流量控制**（rate-based flow control），使用这种方法的协议有一种内置的机制，它能限制发送方传输数据的速率，而无须利用接收方的反馈信息。

在本章，我们将学习基于反馈的流量控制方案，因为基于速率的方案仅在传输层（第 5 章）的一部分中可见，而基于反馈的方案则可同时出现在链路层和更高的层次。后者在近来较为常见，在这种情况下，链路层硬件设计的运行速度足够快到不会造成丢帧。例如，作为链路层硬件实现的**网络接口卡**（NIC，Network Interface Cards），有时声称能以“线速”运行，这意味着它们能以帧到达的速度来处理帧。因而，任何过载不再是链路层的问题，它们必须由高层来处理。

基于反馈的流控制方案有许多种，但是绝大多数使用了同样的基本原理。协议包含了许多定义良好的规则，这些规则规定了发送方什么时候可以发送下一帧。这些规则通常在没有得到接收方许可（隐式或者显式的）之前，禁止继续发送帧。例如，当建立一个连接时，接收方可能会这样说：“你现在可以给我发送 n 个帧，但是在发送完 n 个帧之后就别再发送，直到我告诉你可以继续发送。”稍后我们将讨论这些细节。

3.2 差错检测和纠正

正如我们在第2章中所看到的那样，通信信道有许多不同的特征。某些信道，例如电话系统中的光纤，其错误率很低，因而很少发生传输错误。但是其他信道，尤其是无线链路和老化的本地回路错误率高出光纤好几个数量级。对于这些信道而言，传输出错是常态。从性能的角度来看，这些错误不能在合理的成本开销内彻底解决。因此结论是传输错误非常普遍。我们必须知道如何处理传输错误。

网络设计者针对错误处理已经研究出两种基本策略。这两种策略都在发送的数据中加入冗余信息。一种策略是在每一个被发送的数据块中包含足够多的冗余信息，以便接收方能据此推断出被发送的数据是什么。另一种策略也是包含一些冗余信息，但这些信息只能让接收方推断出是否发生了错误（而推断不出哪个发生了错误），然后接收方可以请求发送方重传。前一种策略使用了**纠错码**（error-correcting code），后一种策略使用了**检错码**（error-detecting code）。使用纠错码的技术通常也称为**前向纠错**（FEC，Forward Error Correction）。

这里的每一项技术都占据着不同的生态位置。在高度可靠的信道上（比如光纤），较为合算的做法是使用检错码，当偶尔发生错误时只需重传整个数据块。然而，在错误发生很频繁的信道上（比如无线链路），更好的做法是在每一个数据块中加入足够的冗余信息，以便接收方能够计算出原始的数据块。FEC被用在有噪声的信道上，因为重传的数据块本身也可能像第一次传输那样出错。

这些编码的一个关键考虑是可能发生的错误类型。无论是纠错码还是检错码都无法处理所有可能的传输错误，因为提供保护措施的冗余比特很可能像数据比特一样出现错误（可危及它们的保护作用）。如果信道给予冗余比特的待遇好于数据比特那当然好，可惜事实并非如此。对信道而言，它们都是同样的比特。这意味着为了避免漏检错误，编码必须强大到足以应付预期的错误。

错误模型有两种。在第一种错误模型中，偶尔出现的极端热噪声快速淹没了信号，引起孤立的单个比特错误。在另一种错误模型中，传输错误往往呈现突发性而不以单个形式出现，这种错误源自物理过程，比如无线信道上的一个深衰落，或者有线信道上的瞬态电气干扰。

两种错误模型实际上造成的问题以及处理方式有不同的权衡。突发的错误相比单个比特错误有自己的优势，也有不足。从优势方面来看，计算机数据总是成块发送。假设数据块大小为1000个比特，误差率为每比特0.001。如果错误是独立的，大多数块将包含一个错误。但如果错误以100个比特的突发形式出现，则平均来说100块中只有一块会受到影响。突发错误的缺点在于当它们发生时比单个错误更难以纠正。

同时还存在着其他类型的错误。有时候，一个错误的位置可以获得，或许因为物理层接收到的一个模拟信号远离了0或1的预期值，因而可以宣布该比特被丢失。这种情形称为**擦除信道**（erasure channel）。擦除信道比那些把比特值翻转的信道更易于纠错，因为即使某个比特被丢失，至少我们还能知道哪个比特出了错。然而，我们往往不能从信道的擦

除性质上受益。

下面我们将同时考察纠错码和检错码。请记住两点。首先，我们在链路层讨论这些编码有客观因素，因为这里是面临数据可靠传输相关问题的首要地方。然而，这些编码方案被广泛使用的根本原因在于可靠性是整个系统所关注的问题。纠错码也会出现在物理层，特别是有噪声干扰的信道，同时还会出现在更高的层次，特别是实时流媒体应用和内容分发应用。检错码更是经常被用在链路层、网络层和传输层。

第二点要记住的是差错编码是应用的数学。除非特别熟悉伽罗瓦（Galois）领域或稀疏矩阵的性质，否则，应该从可靠的来源获得性质更优良的编码，而不是自己设计编码方案。事实上，这正是为何很多协议标准一次又一次采用了相同的编码方法。在下面的材料中，我们将学习一个简单的编码方法，然后再简要描述先进的编码方法。这样，我们可以从简单编码中来理解如何权衡，并且通过先进编码来讨论实际使用的编码方案。

3.2.1　纠错码

我们将考察以下 4 种不同的纠错编码：

（1）海明码。

（2）二进制卷积码。

（3）里德所罗门码。

（4）低密度奇偶校验码。

上述所有编码都将冗余信息加入到待发送的信息中。一帧由 m 个数据位（即信息）和 r 个冗余位（即校验）组成。在**块码**（block code）中，r 个校验位是作为与之相关的 m 个数据位的函数计算获得的，就好像在一张大表中找到 m 位数据对应的 r 校验位。在**系统码**（systematic code）中，直接发送 m 个数据位，然后发出 r 个校验位，而不是在发送前对它们进行编码。在**线性码**（line code）中，r 个校验位是作为 m 个数据位的线性函数被计算出来的。异或（XOR）或模 2 加是函数的流行选择，这意味着编码过程可以用诸如矩阵乘法或简单逻辑电路来完成。除非另有说明，我们在本节中考察的是线性码、系统块状码。

令数据块的总长度为 n（即 n=m+r）。我们将此描述为（n, m）码。一个包含了数据位和校验位的 n 位单元称为 n 位**码字**（codeword）。**码率**（code rate）或者简单地说速率，则定义为码字中不包含冗余部分所占的比例，或者用 m/n 表示。实际上这个码率变化很大。在一个有噪声信道上码率或许是 1/2，在这种情况下接收方所收到的信息中有一半是加入的冗余位；而在高品质的信道上码率接近 1，只有少数的校验位被添加到一个大块消息中。

为了理解发生传输错误后如何进行处理，有必要先来看一看错误到底是什么样的。给定两个被发送或接收的码字，比如说 10001001 和 10110001，完全可能确定这两个码字中有多少个对应位是不同的。此时，上述两个码字有 3 位不同。为确定有多少个不同位，只需 XOR 两个码字，并且计算结果中 1 的个数。例如：

$$\begin{array}{r} 10001001 \\ \underline{10110001} \\ 00111000 \end{array}$$

两个码字中不相同的位的个数称为**海明距离**（Hamming distance）（Hamming，1950）。

它的意义在于，如果两个码字的海明距离为 d，则需要 d 个 1 位错误才能将一个码字转变成另一个码字。

给定计算校验位的算法，完全可以构建一个完整的合法码字列表，然后从这个列表中找出两个具有最小海明距离的码字。这个距离就是整个编码的海明距离。

在大多数数据传输应用中，所有 2^m 种可能的数据报文都是合法的；但是，根据检验位的计算方法，并非所有 2^n 种可能的码字都会被用到。事实上，对于 r 校验位，可能的报文中只有很少一部分 $2^m/2^n$ 或 $1/2^r$ 是合法的码字。正是这种空间稀疏方法，即报文被嵌入到码字空间中，使得接收方能检测并纠正错误。

块码的检错和纠错特性跟它的海明距离有关。为了可靠地检测 d 个错误，需要一个距离为 d+1 的编码方案，因为在这样的编码方案中，d 个 1 位错误不可能将一个有效码字改变成另一个有效码字。当接收方看到一个无效码字时，它就知道发生了传输错误。类似地，为了纠正 d 个错误，需要一个距离为 2d+1 的编码方案，因为在这样的编码方案中，合法码字之间的距离足够远，即使发生了 d 位变化，结果也还是离它原来的码字最近。这意味着在不太可能有更多错误的假设下，可以唯一确定原来的码字，从而达到纠错的目的。

看一个纠错码实例，考虑一个只有下列 4 个有效码字的编码方案：

0000000000，0000011111，1111100000，1111111111

该编码方案的距离是 5，这意味着它可以纠正 2 个错误或者检测双倍的错。如果接收到码字 0000000111 并且期望只有单个或者 2 个错误，则接收方知道原始的码字一定是 0000011111。然而，如果发生了三个错误，0000000000 变成了 0000000111，则以上编码就无法正确地纠正错误了。另外，如果我们期望所有这些错误都会发生，我们也可以检测出它们。只要没有收到合法的码字，就必然发生了错误。很明显在这个例子中，我们不能同时纠正 2 个错误和检测 4 个错误，因为这需要我们以两种不同的方式来解释接收到的码字。

在我们的例子中，解码的任务就是找出最接近接收码字的合法码字。不幸的是，大多数情况下全部码字都将作为候选被评估，这是一件非常耗时的搜索。相反，实际的代码被设计成允许使用快捷方式找出最有可能的原始码字。

设想我们要设计一种编码方案，每个码字有 m 个消息位和 r 个校验位，并且能够纠正所有的单个错误。对于 2^m 个合法消息，任何一个消息都对应有 n 个非法的码字，它们与该消息的距离为 1。这些非法的码字可以这样构成：将该消息对应的合法码字的 n 位，逐个取反，可以得到 n 个距离为 1 的非法码字。因此，每个 2^m 中的合法消息需要 n+1 个位模式来标识它们。由于总共只有 2^n 个位模式，所以，我们必须有 $(n+1)2^m \leqslant 2^n$。由 n=m+r，这个要求变成了

$$(m+r+1) \leqslant 2^r \qquad (3\text{-}1)$$

在给定 m 的情况下，这个条件给出了纠正单个错误所需要的校验位数的下界。

事实上这个理论下限可使用海明方法（1950）获得。在海明码中，码字的位被连续编号，从最左端的位开始，紧跟在右边的那位是 2，依次从左到右编号。2 的幂次方的位（1，2，4，8，16 等）是校验位，其余位（3，5，6，7，9 等）用来填充 m 个数据位。这种模式如图 3-6 所示的（11,7）海明码，其中 7 个数据位和 4 个校验位。每一个校验位强制进行模 2 加，或对某些位的集合，包括其本身进行偶（或奇）校验。一位可能被包括在几个校验位的计算中。若要查看在数据 k 位上的校验位，必须将 k 改写成 2 的幂之和。例

如，11 =1 +2 +8 和 29 =1 +4 +8 +16。校验某一位只需要检查那些覆盖了该位的校验位（例如，校验 1、2 和 8 位就可确定 11 位是否出错）。在这个例子中，我们采用偶校验计算 ASCII 字母“A”的校验和。

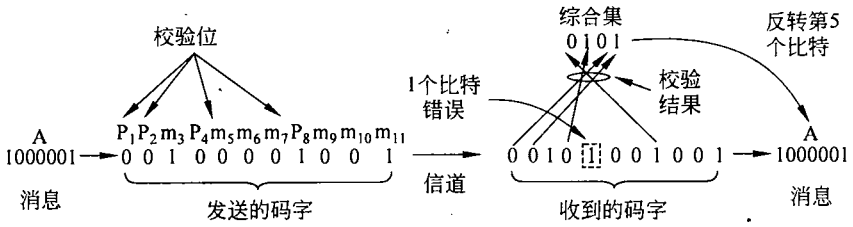


图 3-6　纠正单个错位的（11, 7）海明码示例

这种结构给出了海明距离为 3 的编码，意味着它可以纠正单个错误（或检测 2 个错误）。针对信息位和校验位小心编号的原因在解码的处理过程中表现得非常明显。当接收到一个码字，接收机重新计算其校验位，包括所收到的校验位，得到的计算结果我们称之为校验结果。如果校验位是正确的，对于偶校验和而言，校验结果应该是零。在这种情况下，码字才被认为是有效的，从而可以接收。

然而，如果校验结果不是全零，则意味着检测到了一个错误。校验结果的集合形成的错误综合集，可用来查明和纠正错误。在图 3-6 中，信道上发生了 1 位错误，因此分别针对 k =8，4，2，1 的校验结果是 0，1，0 和 1。由此得出的综合集为 0101 或 4 + 1 = 5。按照设计方案，这意味着第五位有误。把不正确的位（这可能是一个校验位或数据位）取反，并丢弃校验位就得到正确的消息——ASCII 字符“A”。

海明距离对理解块码是有价值的，而且海明码还被用在纠错存储器中。然而，大多数网络使用了更强大的编码。我们将考察的第二个编码是**卷积码**（convolutional code）。这是我们讨论的编码方法中唯一不属于块码的编码。在卷积码中，编码器处理一个输入位序列，并生成一个输出位序列。在块码中没有自然消息大小或编码边界。输出取决于当前的输入和以前的输入。也就是说，编码器有内存。决定当前输出的以前输入位数称为代码的**约束长度**（constraint length）。卷积码由它们的速率和约束长度来标识。

卷积码已被广泛应用于实际部署的网络中，例如，它已经成为 GSM 移动电话系统的一部分，在卫星通信和 802.11 中都得到应用。作为一个例子，图 3-7 给出了一个流行的卷积码。这个代码称为 NASA（美国航天局）卷积码，其 r=1/2 和 k=7。因为它是第一个被用在 1977 年的旅行者号航天飞行任务中的编码。从那以后，它被随意重用于许多其他地方，例如，已成为 802.11 的一部分。

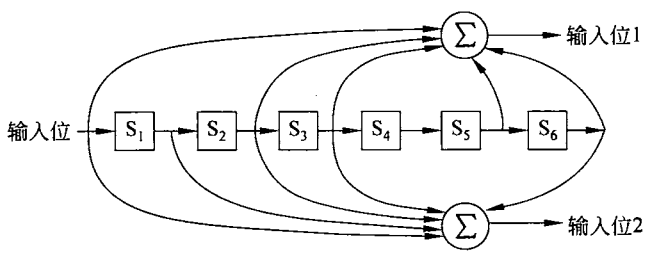


图 3-7　应用于 802.11 的 NASA 二进制卷积码

在图 3-7 中，左边每个输入位产生右边的两个输出位，输出位是输入位和内部状态的 XOR 和。由于它处理的是比特位并执行线性运算，因此是二进制的线性卷积码。又因为 1 个输入产生 2 个输出，因此码率为 1/2。这里的输出位不是简单的输入位，从而它不属于系统码。

内部状态保存在 6 个内存寄存器中。每当输入一位寄存器的值就右移一位。例如，如果输入序列为 111，初始状态是全零，则在输入第一、第二和第三位后从左到右的内部状态变化成 100000、110000 和 111000。对应的输出位分别是 11、10 和 01。这个过程需要 7 次移位才能完全清空输入，从而不影响输出。因此，该卷积码的约束长度是 k= 7。

卷积码的解码过程是针对一个输入位序列，找出最有可能产生观察到的输出位序列（包括任何错误）。对于较小值的 k，一种广泛使用的算法是由 Viterbi 开发的（Forney，1973）。该算法逐个检查观察到的序列，记住每一步和输入序列的每个可能内部状态，即输入序列产生观察序列可能产生的错误。最终其中那个具有最少错误的输入序列就是最有可能的消息。

卷积码实际上已经非常流行，它之所以很容易被采纳的一个因素在于解码 0 或 1 的不确定性。例如，假设–1 V 表示逻辑 0，+1 V 表示逻辑 1，接收方可能接收到的 2 位分别是 0.9 V 和–0.1 V。卷积码不是简单地将这些信号绝对映射成逻辑 1 和 0，而是把 0.9 V 看成“很可能是 1”，把–0.1 V 看成“很可能是 0”，从而最终获得正确的整个序列。Viterbi 算法的扩展适用于这些不确定因素，因而能提供更强的纠错功能。这种带有一位不确定性的工作方法称为**软判决解码**（soft-decision decoding）。相反，在执行纠错之前就决定了每个位是 0 或 1 的工作方法称为**硬判决解码**（hard-decision decoding）。

我们将描述的第三种纠错码是**里德所罗门码**（Reed-Solomon code）。像海明码一样，里德所罗门码是线性块码，而且往往也是系统码。但与海明码不同的是，里德所罗门码对 m 位符号进行操作，而不是针对单个位处理。当然，这里需要更多的数学参与，因此我们将用类比的方式来描述该编码方案。

里德所罗门码基于这样的事实：每一个 n 次多项式是由 n+1 点唯一确定的。例如，一条具有 ax+b 形式的线由两个点所决定。同一条线上的额外点都是冗余的，这将有助于纠错。可以想象有两个数据点代表了一条线，并且我们给这两个数据点额外加上两个校验点，该两个校验点选自同一条线。如果收到的其中一个点出现错误，我们仍然可以通过接收点的拟合线来恢复这个数据点。三个点将处在同一条直线上，而出错的那个点不在这条线上。只要找到这条线，我们就可以纠正错误。

里德所罗门码实际上被定义成一个在有限域上操作的多项式，但工作方式相同。对于 m 位符号而言，码字长 2^m-1 个符号。一种流行的选择是 m=8，这样符号就是字节。因此，一个码字为 255 个字节长。（255，233）码被广泛使用，它在 233 个数据符号上增加了 32 个冗余符号。带有纠错功能的解码算法由 Berlekamp 和 Massey 开发，它能有效执行中等长度的解码（Massy，1969）。

里德所罗门码得到广泛应用的原因还在于其强大的纠错性能，尤其针对突发错误。它们被用在 DSL、线缆上的数据通信、卫星通信、最无处不在的 CD、DVD 和蓝光光盘。因为它们基于 m 位符号，因此一位错误和 m 位突发错误都只是作为一个出错的符号来处理。当加入 2t 个冗余符号后，里德所罗门码能够纠正传输符号中的任意 t 个错误。这意味着，

例如在（255，233）中，由于有 32 个冗余符号，因此可以纠正多达 16 个符号错误。因为符号是连续的，并且每个 8 位长，所以可以纠正高达 128 位的突发错误。如果错误模型是擦除的（例如，一张 CD 消除了一些符号划痕）则情况更好。在这种情况下，高达 2t 个错误都可以得到更正。

里德所罗门码通常与其他编码结合在一起使用，如卷积码。这种想法的依据在于：卷积码在处理孤立的比特错误时很有效，但当接收到的比特流中有太多的错误（和突发错误类似），卷积码就无法处理了。在卷积码内加入里德所罗门码，因为里德所罗门码可以横扫突发错误，因此两者的结合就能将纠错任务完成得非常好，综合起来的编码模式对单个错误和突发错误都有良好的保障作用。

我们考察的最后一个纠错码是**低密度奇偶校验码**（LDPC，Low-Density Parity Check）。LDPC 码是线性块码，由 Robert Gallagher 在他的博士论文中首次提出（Gallagher，1962）。像大多数学位论文一样，它们很快被人遗忘，直到 1995 年计算能力的进步使得它们重新走向实际应用。

LDPC 码中的每个输出位由一小部分的输入位形成。这样使得编码可以用一个 1 的密度很低的矩阵来表示，这也是编码名称的由来。接收到的码字通过一个近似算法解码获得，该算法通过迭代不断改进接收到的数据与合法码字的最佳匹配。如此来纠正错误。

LDPC 码比较适用于大块数据，而且具有出色的纠错能力，因而性能优于其他许多编码（包括我们曾经考察过的那些）。正是基于这个原因，它们迅速被新的协议所采纳，成为数字视频广播、万兆以太网、电力线网络，以及最新版本 802.11 标准的一部分。我们希望它们出现在未来的更多网络中。

3.2.2　检错码

纠错编码被广泛应用于无线链路。众所周知，相比光纤，无线链路嘈杂不堪而且容易出错，如果没有纠错码，将很难从该链路获得任何信息。然而，光纤或高品质铜线的错误率要低得多，因此对于偶尔出现的错误采用差错检测和重传的处理方式通常更加有效。

我们将考察 3 种检错码。这些检错码都是线性的系统块码：

（1）奇偶。

（2）校验和。

（3）循环冗余校验（CRC）。

为了看清楚检错码如何比纠错码更有效，考虑第一个检错码——把单个奇偶校验位附加到数据中。奇偶位的选择原则是使得码字中比特 1 的数目是偶数（或奇数）。这样处理等同于对数据位进行模 2 加或异或操作来获得奇偶位。例如，当以偶校验方式发送 1011010 时，在数据末尾添加一位成为 10110100。若采用奇校验方式发送 1011010 时，则结果为 10110101。具有单个校验位的编码具有码距 2，因为任何 1 位错误都将使得码字的奇偶校验码出错。这意味着奇偶码可以检测出 1 位错误。

考虑这样的信道，其上发生的错误都是孤立的，并且每个比特的出错率是 10^{-6}。这看似微小的错误率，对长距离的线缆已经是最好的条件了，但对错误检测却是一个挑战。典型局域网链路的误码率大约为 10^{-10}。令数据块大小为 1000 位。为了提供相应的纠错功能，

我们根据等式（3-1）可知需要10个校验位。因此1兆大的数据块将需要10 000个校验位。但是，如果只为检测出该块数据中是否存在1位错误，每块数据仅一个校验位就足够了。如此一来，每1000个数据块中才有一块出现错误，只需要重传额外的一块（1001位）即可修复错误。每1兆数据用于错误检测和重传的总开销只有2001位，相比海明码需要的10 000位显然效率高得多。

这种校验方案的困难在于单个校验位只能可靠地检测出1位错误。如果数据块因一个长的突发错误造成严重乱码，那么这种错误被检测出来的概率只有0.5，显然这是令人难以接受的。如果发送的每个数据块作为一个n位宽和k位高的长方形矩阵来处理，则检测出错误的概率可望得到很大提高。现在，如果我们为每一行计算和发送一个校验位，只要每一行最多只有一个错误发生，那么我们就能可靠地检测出k位错误。

然而，我们还可以做点其他事情来提高针对突发错误的更好保护：改变计算校验位的次序，即以不同于数据位发送的次序来计算校验位。这种处理方式就是所谓的交错校验（interleaving）。在这种情况下，我们将为n列中的每列计算校验位，按k行发送全部的数据位，发送次序是从上到下发送每一行，行内数据位通常按从左到右的次序发送。在最后一行，发送n个校验位。这种传输顺序如图3-8所示，其中n=7和k=7。

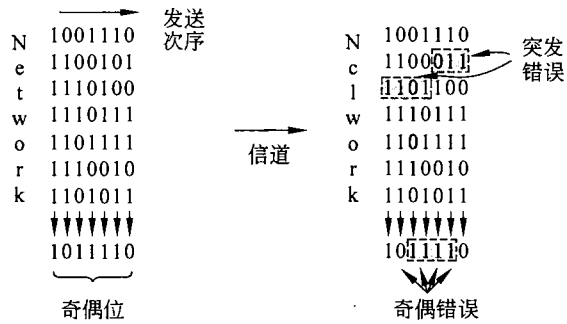


图3-8 检测一个突发错误的交错奇偶校验位

交错校验是一种将检测（或纠正）单个错误的编码转换成能检测（或纠正）突发错误的通用技术。在图3-8中，当发生一个长度为n=7的突发错误，出错的位恰好分散在不同的列（突发错误并不隐含着所有的位都出错；它只意味着至少第一位和最后一位错误。如图3-8所示4位错误分布在7位范围内）。n列中的每一列至多只有一位受到影响，因此这些列中的校验位将能检测到该错误。这种方法对于nk长度的数据块使用了n个校验位就能检测出一个长度小于等于n的突发错误。

然而，一个长度为n+1的突发错误将被遗漏，如果第一位和最后一位反转，所有其他位都正确，这样的错误无法检测出来。如果数据块被一个长突发错误或多个短突发错误所扰乱，该n列中任何一列有正确校验位的概率偶尔有0.5，所以一个不该接受而被接收的坏块的概率是2^{-n}。

第二类检错码是校验和，与一组奇偶校验位密切相关。**校验和**（checksum）这个词通常用来指与信息相关的一组校验位，不管这些校验位是如何计算出来的。一组奇偶校验位是校验和的一个例子。然而，还有其他种类的校验和，强大的校验和基础是对消息中的数据位进行求和计算。校验和通常放置在消息的末尾，作为求和功能的补充。这样一来，通

过对整个接收到的码字（包含了数据位和校验和）进行求和计算就能检测出错误。如果计算结果是零，则没有检测出错误。

校验和的一个例子是 16 位的 Internet 校验和，作为 IP 协议的一部分用在所有 Internet 数据包中（Braden 等，1988）。该校验和是按 16 位字计算得出的消息位总和。由于此方法针对字而不是像奇偶校验那样针对位进行操作，因此奇偶校验没能检测出的错误此时仍然影响着校验和，从而能被检测出来。例如，如果两个字的最低位都从 0 错误地变成了 1，其上的奇偶校验将无法检测到这个错误。但是，两个 1 增加到 16 位校验和中将产生不同的结果。这个错误就能被检测出来。

Internet 校验和是以补码运算而非 2^{16} 模加运送得到的。在补码运算中，负数是其正数的按位补。现代计算机采用双补算法（two's complement），负数是该数的补码加一。在双补计算机中，一个数的补码等价于模 2^{16} 加，并且任何高序位溢出被放回到低序位上。该算法为校验和位提供了更均匀的数据覆盖面。否则，两个高序位可能因相加、溢出而被丢失，从而对校验和没有任何作用。这种双补算法还有另一个好处，0 的补码有两种表示方法：全 0 和全 1。这就允许用一个值（例如，全 0）表示没有校验和而不需要用额外一个字段作特别的说明。

几十年来，人们一直有这样的假设，即计算校验和所针对的帧包含了随机位。校验和算法的所有分析都是基于这样的假设而进行。但是由（Partridge 等，1995）检查的真实数据表明这种假设是十分错误的。因此，在某些情况下未发现的错误比以前想象得更为常见。

尤其是 Internet 校验和，它有效而简单；但在某些情况下提供的保护很弱，正是因为它只是一个简单的和。它检测不出 0 数据的增加或删除，也检测不出消息中被掉换的那部分，而且对两个数据包拼接起来的消息只有弱保护作用。这些错误在随机过程中似乎不太可能发生，但恰恰是一种能发生在有缺陷硬件上的错误。

一个更好的选择是 Fletcher 校验和（Fletcher，1982）。它包括一个位置组件，将数据和其位置的乘积添加到总和中。这样能对数据位置的变化提供更强大的检测作用。

尽管前面两种方案对高层而言有时候已足够，但实际上，链路层广泛使用的是第三种更具检错能力的方法：**循环冗余校验码**（CRC，Cyclic Redundancy Check），也称为**多项式编码**（polynomial code）。多项式编码的基本思想是：将位串看成是系数为 0 或 1 的多项式。一个 k 位帧看作是一个 k-1 次多项式的系数列表，该多项式共有 k 项，从 x^{k-1} 到 x^0。这样的多项式认为是 k-1 阶多项式。高次（最左边）位是 x^{k-1} 项的系数，接下来的位是 x^{k-2} 项的系数，依此类推。例如，110001 有 6 位，因此代表了一个有 6 项的多项式，其系数分别为 1、1、0、0、0 和 1：即 $1x^5+1x^4+0x^3+0x^2+0x^1+1x^0$。

多项式的算术运算遵守代数域理论规则，以 2 为模来完成。加法没有进位，减法没有借位。加法和减法都等同于异或。例如：

$$\begin{array}{cccc} 10011011 & 00110011 & 11110000 & 01010101 \\ \underline{-11001010} & \underline{+11001101} & \underline{-10100110} & \underline{-10101111} \\ 01010001 & 11111110 & 01010110 & 11111010 \end{array}$$

长除法与二进制中的除法运算一样，只不过减法按模 2 进行。如果被除数与除数有一样多的位，则该除数要“进入到”被除数中。

使用多项式编码时，发送方和接收方必须预先商定一个**生成多项式**（generator

polynomial）G(x)。生成多项式的最高位和最低位系数必须是1。假设一帧有m位，它对应于多项式M(x)，为了计算它的CRC，该帧必须比生成多项式长。基本思想是在帧的尾部附加一个校验和，使得附加之后的帧所对应的多项式能够被G(x)除尽。当接收方收到了带校验和的帧之后，它试着用G(x)去除它。如果有余数的话，则表明传输过程中有错误。

计算CRC的算法如下：

（1）假设G(x)的阶为r。在帧的低位端加上r个0位，使得该帧现在包含m+r位，对应多项式为$x^rM(x)$。

（2）利用模2除法，用对应于G(x)的位串去除对应于$x^rM(x)$的位串。

（3）利用模2减法，从对应于$x^rM(x)$的位串中减去余数（总是小于等于r位）。结果就是将被传输的带校验和的帧。它的多项式不妨设为T(x)。

图3-9显示了采用生成多项式为$G(x)=x^4+x+1$计算帧1101011111校验和的情形。

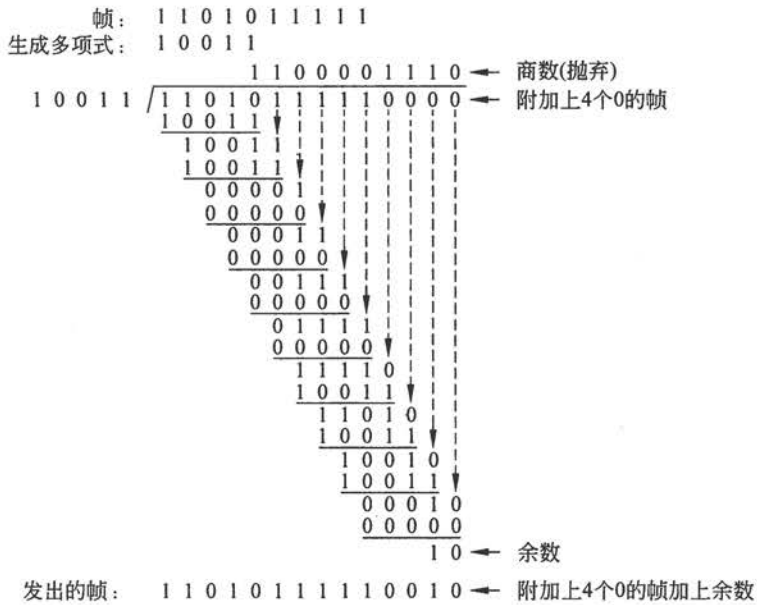


图3-9 CRC计算示例

显然，T(x)可以被G(x)除尽（模2）。在任何一种除法中，如果将被除数减掉余数，则剩下的差值一定可以被除数除尽。例如，在十进制中，如果210 278被10 941除，则余数为2399。从210 278中减去2399，得到207 879，它可以被10 941除尽。

现在我们来分析这种方法的能力。什么样的错误能被检测出来？想象一下在传输过程中发生了一个错误，因此接收方收到的不是T(x)，而是T(x)+E(x)，其中E(x)中对应的每位1都变反了。如果E(x)中有k个1，则表明发生了k个1位错误。单个突发错误可以这样描述：初始位是1，然后是0和1的混合，最后一位也是1，所有其他位都是0。

接收方在收到了带校验和的帧之后，用G(x)来除它；也就是说，接收方计算[T(x) + E(x)]/G(x)。T(x)/G(x)是0，因此计算结果简化为E(x)/G(x)。如果错误恰好发生在作为因子的G(x)多项式中，那么将无法被检测到；所有其他的错误都能够检测出来。

如果只有一位发生错误，即$E(x) = x^i$，这里i决定了错误发生在哪一位上。如果G(x)

包含两项或者更多项，则它永远也不会除尽 E(x)，所以，所有的一位错误都将被检测到。

如果有两个独立的一位错误，则 $E(x) = x^i+x^j$，这里 i>j。换一种写法，E(x)可以写成 $E(x) = x^j(x^{i-j}+1)$。如果我们假定 G(x)不能被 x 除尽，则所有的双位错误都能够检测出来的充分条件是：对于任何小于等于 i-j 最大值（即小于等于最大帧长）的 k 值，G(x)都不能除尽 x^k+1。简单地说，低阶的多项式可以保护长帧。例如，对于任何 k<32 768，$x^{15}+x^{14}+1$ 都不能除尽 x^k+1。

如果有奇数个位发生了错误，则 E(x)包含奇数项（比如 x^5+x^2+1，但不能是 x^2+1）。有意思的是，在模 2 系统中，没有一个奇数项多项式包含 x+1 因子。因此，以 x+1 作为 G(x)的一个因子，我们就可以捕捉到所有包含奇数个位变反的错误情形。

最后，也是最重要的，带 r 个校验位的多项式编码可以检测到所有长度小于等于 r 的突发错误。长度为 k 的突发错误可以用 $x^i(x^{k-1}+x^{k-2}+ \cdots + 1)$来表示，这里 i 决定了突发错误的位置离帧的最右端的距离有多远。如果 G(x)包含一个 x^0 项，则它不可能有 x^i 因子，所以，如果括号内表达式的阶小于 G(x)的阶，则余数永远不可能为 0。

如果突发错误的长度为 r+1，则当且仅当突发错误与 G(x)一致时，错误多项式除以 G(x)的余数才为 0。根据突发错误的定义，第一位和最后一位必须为 1，所以它是否与 G(x)匹配取决于其他 r-1 个中间位。如果所有的组合被认为是等概率的话，则这样一个不正确的帧被当做有效帧接收的概率是 $1/2^{r-1}$。

同样可以证明，当一个长度大于 r+1 位的突发错误发生时，或者几个短突发错误发生时，一个坏帧被当做有效帧通过检测的概率为 $1/2^r$，这里假设所有的位模式都是等概率的。

一些特殊的多项式已经成为国际标准，其中一个被 IEEE 802 用在以太网示例中，多项式为：

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$$

除了其他优良特性外，该多项式还能检测到长度小于等于 32 的所有突发错误，以及影响到奇数位的全部突发错误。20 世纪 80 年代以后它得到广泛应用。但是，这并不意味着它是最好的选择。（Castagnoli 等，1993）和（Koopman，2002）采用穷尽计算搜索，发现了最好的 CRC。这些 CRC 针对典型消息长度获得的海明距离为 6，而 IEEE 标准的 CRC-32 海明距离只有 4。

虽然计算 CRC 所需的运算看似很复杂，但在硬件上通过简单的移位寄存器电路很容易计算和验证 CRC（Pdterson&Brown，1961）。实际上，这样的硬件几乎一直被使用着。数十种网络标准采用了不同的 CRC，包括几乎所有的局域网（如以太网、802.11）和点到点链接（例如，SONET 上的数据包）。

3.3　基本数据链路层协议

为了引入协议主题，我们先从 3 个复杂性逐渐增加的协议开始。对此感兴趣的读者，可以通过 Web 网站（参见前言）获得这些协议以及后面介绍的一些协议的模拟器。在考察这些协议之前，先明确一些有关底层通信模型的基本假设是完全有必要的。

首先，我们假设物理层、数据链路层和网络层都是独立的进程，它们通过来回传递消

息进行通信。图 3-10 给出了一个通用实现。物理层进程和某些数据链路层进程运行在一个称为网络接口卡（NIC，Network Interface Card）的专用硬件上；链路层进程的其他部分和网络层进程作为操作系统的一部分运行在主 CPU 上，链路层进程的软件通常以设备驱动器的形式存在。然而，其他的实现方案也是有可能的（比如，把 3 个进程下载到一个称为网络加速器的专用硬件上运行，或者 3 个进程按照软件定义的速率运行在主 CPU）。实际上，首选的实现方式随着技术权衡每 10 年都会发生变化。无论如何，将这 3 层作为独立的进程来讨论有助于使概念更加清晰，同时也可以强调每一层的独立性。

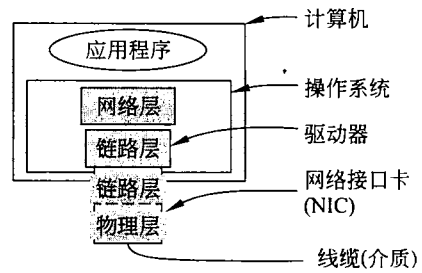


图 3-10 物理层、数据链路层和网络层的实现

另一个关键的假设是：机器 A 希望用一个可靠的、面向连接的服务向机器 B 发送一个长数据流。以后我们再考虑 B 同时向 A 发送数据的情形。假定 A 要发送的数据总是已经准备好，不必等待这些数据被生成出来。或者说，当 A 的数据链路层请求数据时，网络层总能够立即满足数据链路层的要求（这个限制后面也将被去掉）。

我们还假设机器不会崩溃。也就是说，这些协议只处理通信错误，不处理因为机器崩溃和重新启动而引起的问题。

在涉及数据链路层时，通过接口从网络层传递到数据链路层的数据包是纯粹的数据，它的每一位都将被递交到目标机器的网络层。目标机器的网络层可能会将数据包的一部分解释为一个头，这不属于数据链路层的考虑范围。

当数据链路层接收到一个数据包，它就在数据包前后增加一个数据链路层头和尾，由此把数据包封装到一个帧中（见图 3-1）。因此，一个帧由一个内嵌的数据包、一些控制信息（在头中）和一个校验和（在尾部）组成。然后，帧被传输到另一台机器上的数据链路层。我们假设有一个现成的代码库，其中过程 to_physical_layer 发送一帧，from_physical_layer 接收一帧。这些过程负责计算和附加校验和，并检查校验和是否正确（这部分工作通常由硬件完成），所以我们无须关心本节数据链路层协议的这部分内容。例如，它们或许会用到上节讨论的 CRC 算法。

刚开始时，接收方什么也不做。它只是静静地等待着某些事情的发生。在本章给出的示例协议中，我们用过程调用 wait_for_event（&event）标示数据链路层正在等待事情发生。只有当确实发生了什么事情（比如，到达了一个帧），该过程才返回。过程返回时，变量 event 说明究竟发生了什么事件。对于不同的协议，可能的事件集合也是不同的，每个协议都需要单独定义和描述事件集合。请注意，在一个更加实际的环境中，数据链路层不会像我们所建议的那样在一个严格的循环中等待事件，而是会接收一个中断；中断将使它终止当前的工作，转而处理入境帧。然而，为了简便起见，我们将忽略数据链路层内部所有并发进行的活动细节，假定它全部时间都在处理一个信道。

当一帧到达接收方，校验和被重新计算。如果计算出的帧校验和不正确（即发生了传输错误），则数据链路层会收到通知（event = cksum_err）。如果到达的帧没有受到任何损坏，数据链路层也会接到通知（event=frame_arrival），因此它可以利用 from_physical_layer 得到

该帧，并对其进行处理。只要接收方的数据链路层获得了一个完好无损的帧，它就检查头部的控制信息；如果一切都没有问题，它就将内嵌的数据包传递给网络层。无论在什么情况下，帧头部分的信息都不会被交给网络层。

为什么网络层永远得不到任何帧头的信息，理由非常简单，那就是要保持网络层和数据链路层的完全分离。只要网络层对数据链路协议和帧格式一无所知，那么当数据链路协议和帧格式发生变化时，网络层软件可以不作任何改变。每当一块新 NIC 安装在计算机上时这种情况就会发生。在网络层和数据链路层之间提供一个严格的接口可以大大地简化任务设计，因为不同层上的通信协议可以独立地发展。

图 3-11 给出了后面要讨论的许多协议公用的一些声明（C 语言）。这里定义了 5 个数据结构：boolean、seq_nr、packet、frame_kind 和 frame。boolean 是一个枚举类型，可以取值 true 和 false。seq_nr 是一个小整数，用来对帧进行编号，以便我们可以区分不同的帧。这些序号从 0 开始，一直到（含）MAX_SEQ，所以，每个需要用到序号的协议都要定义它。packet 是同一台机器上网络层和数据链路层之间，或者不同机器上的网络层对等实体

```
#define MAX_PKT 1024                    /*determines packet size in bytes */

typedef enum {false, true} boolean;     /* boolean type */
typedef unsigned int seq_nr;            /* sequence or ack numbers */
typedef struct {unsigned char data      /* packet definition */
  [MAX_PKT];} packet;
typedef enum {data, ack, nak}           /* frame_kind definition */
  frame_kind;


typedef struct{                         /* frames are transported in this layer */
  frame_kind kind;                      /* what kind of frame is it? */
  seq_nr seq;                           /* sequence number */
  seq_nr ack;                           /* acknowledgement number */
  packet info;                          /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);
/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);
/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);
/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);
/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);
/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);
/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);
/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);
/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);
/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);
/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);
/* Macro inc is expanded in-line: increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

图 3-11　协议需要遵守的某些定义。这些定义放置在 protocol.h 文件中

之间交换的信息单元。在我们的模型中，它总是包含 MAX_PKT 个字节数据，但在实际环境中，它的长度应该是可变的。

一个帧由 4 个字段组成：kind、seq、ack 和 info，其中前 3 个包含了控制信息，最后一个可能包含了要被传输的实际数据。这些控制字段合起来称为**帧头**（frame header）。

kind 字段指出了帧中是否有数据，因为有些协议需要区分只有控制信息的帧和同时包含控制信息和数据信息的帧。seq 和 ack 分别用作序号和确认，后面还会详细描述它们的用法。数据帧的 info 字段包含了一个数据包；控制帧的 info 字段没有用处。一个更加实际的协议实现将会使用一个变长的 info 字段，而对于控制帧则完全忽略。

再次强调的是理解数据包和帧之间的关系非常重要。网络层从传输层获得一个报文，然后在该报文上增加一个网络层头，由此创建了一个数据包。该数据包被传递给数据链路层，然后被放到输出帧的 info 字段中。当该帧到达目标机器时，数据链路层从帧中提取出数据包，将数据包传递给网络层。在这样的工作方式中，网络层就好像机器一样可以直接交换数据包。

图 3-11 还列出了许多过程。这些库程序的细节与具体实现有关，它们的内部工作机制不是我们下面讨论需要关心的。前面已经提过，wait_for_event 是一个严格的循环过程，它等待事情发生。过程 to_network_layer 和 from_network_layer 被数据链路层用来向网络层传递数据包或者从网络层接收数据包。注意，from_physical_layer 和 to_physical_layer 在数据链路层和物理层之间传递帧。换句话说，to_network_layer 和 from_network_layer 处理第二层和第三层之间的接口，而 from_physical_layer 和 to_physical_layer 则处理第一层和第二层之间的接口。

在大多数协议中，我们假设信道是不可靠的，并且偶尔会丢失整个帧。为了能够从这种灾难中恢复过来，发送方的数据链路层每当发出一帧，就要启动一个内部计时器或者时钟。如果在预设的时间间隔内没有收到应答，则时钟超时，数据链路层会收到一个中断信号。

在我们的协议中，这个过程是这样实现的：让过程 wait_for_event 返回 event = timeout。过程 start_timer 和 stop_timer 分别打开和关闭计时器。只有当计时器在运行并且调用 stop_timer 之前，超时事件才有可能发生。在计时器运行的同时，允许显式地调用 start_timer；这样的调用只是重置时钟，等到再经过一个完整的时钟间隔之后引发下一次超时事件（除非它再次被重置，或者被关闭）。

过程 start_ack_timer 和 stop_ack_timer 控制一个辅助计时器，该定时器被用于在特定条件下产生确认。

过程 enable_network_layer 和 disable_network_layer 用在较为复杂的协议中。在这样的协议中，我们不再假设网络层总是有数据要发送。当数据链路层启动网络层后，它就允许网络层在有数据包要发送时中断自己。我们用 event=network_layer_ready 来表示这种情况。当网络层被禁用后，它不会引发这样的事件发生。通过非常谨慎地设置何时启用网络层以及何时禁用网络层，数据链路层就能有效避免网络层用大量的数据包把自己淹没，即耗尽所有的缓存空间。

帧序号总是落在从 0 到 MAX_SEQ（含）的范围内，不同协议的 MAX_SEQ 可以不相同。通常有必要对序号按循环加 1 处理（即 MAX_SEQ 之后是 0）。宏 inc 可以执行这项序

号递增任务。之所以把这项工作定义成宏是因为在帧处理的关键路径上都要用到该参数。正如后面我们将会看到，限制网络性能的因素通常在于协议处理过程，所以把这种简单操作定义成宏并不会影响代码的可读性，却能够提高性能。

图 3-11 中的声明是我们下面将要讨论的每个协议的一部分。为了节省空间和方便引用，它们被提取出来列在一起；但从概念上讲，它们应该与协议本身合并在一起。在 C 语言中，合并的方法是，把这些定义放在一个特殊的头文件中，这里是 protocol.h 文件，然后在协议文件中使用 C 预处理器的#include 设施将这些定义包含进来。

3.3.1　一个乌托邦式的单工协议

作为第一个例子，我们来考虑一个简单得不能再简单的协议，它不需要考虑任何出错的情况。在这个协议中，数据只能单向传输。发送方和接收方的网络层总是处于准备就绪状态。数据处理的时间忽略不计。可用的缓存空间无穷大。最强的一个条件是数据链路层之间的通信信道永远不会损坏帧或者丢失帧。这个完全不现实的协议我们给它一个昵称“乌托邦”（Utopia），我们将以此作为构建后续协议的基本结构。图 3-12 给出了这个协议的实现。

```
/* Protocol 1 (Utopia) provides for data transmission in one direction only,
   from sender to receiver. The communication channel is assumed to be error
   free and the receiver is assumed to be able to process all the input
   infinitely quickly.Consequently, the sender just sits in a loop pumping
   data out onto the line as fast as it can. */
typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender1(void)
{
   frame s;                          /* buffer for an outbound frame */
   packet buffer;                    /* buffer for an outbound packet */
   while (true) {
     from_network_layer(&buffer); /* go get something to send */
     s.info = buffer;             /* copy it into s for transmission */
     to_physical_layer(&s);        /* send it on its way */
}                                    /* Tomorrow, and tomorrow, and tomorrow,
                                    Creeps in this petty pace from day to day
                                    To the last syllable of recorded time.
                                     - Macbeth, V, v */
}
void receiver1(void)
{
   frame r;
   event_type event;                 /* filled in by wait, but not used here */
   while (true) {
     wait_for_event(&event);       /* only possibility is frame_arrival */
     from_physical_layer(&r);      /* go get the inbound frame */
     to_network_layer(&r.info);    /* pass the data to the network layer */
   }
}
```

图 3-12　一个乌托邦式的单工协议

协议由两个单独的过程组成：一个发送过程和一个接收过程。发送过程运行在源机器的数据链路层上；接收过程运行在目标机器的数据链路层上。这里没有用到序号和确认，所以不需要 MAX_SEQ。唯一可能的事件类型是 frame_arrival（即到达了一个完好无损

的帧）。

发送过程是一个无限的 while 循环，它尽可能快速地把数据放到线路上。循环体由三个动作组成：从（总是就绪的）网络层获取一个数据包、利用变量 s 构造一个出境帧，然后通过物理层发送该帧。这个协议只用到了帧结构中的 info 字段，因为其他字段都跟差错控制或者流量控制有关，而这里并没有差错控制或者流量控制方面的限制。

接收过程同样很简单。开始时，它等待某些事情的发生，这里唯一可能的事件是到达了一个未损坏的帧。到达了一个帧后，过程 wait_for_event 返回，其中的参数 event 被设置成 frame_arrival（这里被忽略）。调用 from_physical_layer 将新到达的帧从硬件缓冲区中删除，并且放到变量 r 中，以便接收方的代码可以访问该帧。最后，该帧的数据部分被传递给网络层，数据链路层返回继续等待下一帧的到来，即实际上它把自己挂起来，直到下一帧到来为止。

乌托邦协议是不现实的，因为它不处理任何流量控制或纠错工作。其处理过程接近于无确认的无连接服务，必须依赖更高层次来解决上述这些问题，即使无确认的无连接的服务也要做一些差错检测的工作。

3.3.2 无错信道上的单工停-等式协议

现在我们将处理这样的问题：发送方以高于接收方能处理到达帧的速度发送帧，导致接收方被淹没。这种情形实际上很容易出现，因此协议是否能够防止它非常重要。然而，我们仍然假设通信信道不会出错，并且数据流量还是单工的。

一种解决办法是建立足够强大的接收器，使其强大到能处理一个接着一个帧组成的连续流（或等价于把数据链路层定义成足够慢，慢到接收器的处理速度完全跟得上）。它必须有足够大的缓冲区和以线速率运行的处理能力，而且必须能足够快地把所接收到的帧传递给网络层帧。然而，这是一个最坏情况下的解决方案。它需要专用的硬件，而且如果该链路的利用率十分低还可能浪费资源。此外，它只是把发送方太快这个问题转移到了其他地方，在这种情况下就是网络层。

这个问题的更一般化解决方案是让接收方给发送方提供反馈信息。接收方将数据包传递给网络层之后给发送方返回一个小的哑帧，实际上这一帧的作用是给发送方一个许可，允许它发送下一帧。发送方在发出一帧之后，根据协议要求，它必须等待一段时间直到短哑帧（即确认）到达。这种延缓就是流量控制协议的一个简单例子。

发送方发送一帧，等待对方确认到达后才能继续发送，这样的协议称为**停-等式协议**（stop-and-wait）。图 3-13 给出了一个单工停-等式协议的例子。

虽然这个例子中的数据流量是单工的，即只是从发送方传到接收方，但是帧可以在两个方向上传送。因此，两个数据链路层之间的通信信道必须具备双向传输信息的能力。然而，这个协议限定了流量的严格交替关系：首先发送方发送一帧，然后接收方发送一帧；接着发送方发送另一帧，然后接收方发送另一帧，以此类推。这里采用一个半双工的物理信道就足够了。

就像在协议 1 中那样，发送方首先从网络层获取一个数据包，用它构造一帧，然后发送出去。但现在，与协议 1 不同的是，发送方在开始下一轮循环从网络层获取下一个数据

```
/* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data
   from sender to receiver. The communication channel is once again assumed to
   be error free, as in protocol 1. However, this time the receiver has only a
   finite buffer capacity and a finite processing speed, so the protocol must
   explicitly prevent the sender from flooding the receiver with data faster
   than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"
void sender2(void)
{
   frame s;                          /* buffer for an outbound frame */
   packet buffer;                    /* buffer for an outbound packet */
   event_type event;                 /* frame_arrival is the only possibility */
   while (true) {
      from_network_layer(&buffer);   /* go get something to send */
      s.info = buffer;               /* copy it into s for transmission */
      to_physical_layer(&s);         /* bye-bye little frame */
      wait_for_event(&event);        /* do not proceed until given the go ahead */
   }
}
void receiver2(void)
{
   frame r, s;                       /* buffers for frames */
   event_type event;                 /* frame_arrival is the only possibility */
   while (true) {
      wait_for_event(&event);        /* only possibility is frame_arrival */
      from_physical_layer(&r);       /* go get the inbound frame */
      to_network_layer(&r.info);     /* pass the data to the network layer */
      to_physical_layer(&s);         /* send a dummy frame to awaken sender */
   }
}
```

图 3-13 一个单工停等式协议

包之前必须等待，直到确认帧到来。发送方的数据链路层甚至根本不检查接收到的帧，因为对它而言只有一种可能性，即入境帧总是来自接收方的确认。

在 receiver1 和 receiver2 之间的唯一区别在于 receiver2 将数据包递交给网络层之后，在进入下一轮等待循环之前，要先给发送方返回一个确认帧。对于发送方来说，确认帧的到来比帧内包含什么内容更重要，因此接收方根本不需要往确认帧中填充任何特别的信息。

3.3.3 有错信道上的单工停-等式协议

现在让我们来考虑比较常规的情形，即通信信道可能会出错。帧可能会被损坏，也可能完全被丢失。然而，我们假设，如果一帧在传输过程中被损坏，则接收方硬件在计算校验和时能检测出来。如果一帧被损坏了之后校验和仍然是正确的（这种情况不太可能会出现），那么这个协议（以及所有其他的协议）将会失败（即给网络层递交了一个不正确的数据包）。

粗看起来，协议 2 稍作修改就能应付这项工作：增加一个计时器。发送方发出一帧，接收方只有在正确接收到数据之后才返回一个确认帧。如果到达接收方的是一个已经损坏的帧，则它将被丢弃。经过一段时间之后发送方将超时，于是它再次发送该帧。这个过程将不断重复，直至该帧最终完好无损地到达接收方。

这个方案有一个致命的缺陷。在继续往下读之前，请仔细想一想这个问题，并努力找

出哪里可能会出错。

为了看清楚哪里可能出错，必须牢记：数据链路层的目标是在两个网络层进程之间提供无差错的、透明的通信。机器A上的网络层将一系列数据包交给它的数据链路层，而它的数据链路层则必须保证这些数据包将丝毫不差地由机器B上的数据链路层递交给它的网络层。尤其是机器B上的网络层不可能知道数据包是否被丢失，或者被复制了多份，所以数据链路层必须保证不可能出现没有形式的传输错误会导致一个数据包被多次递交给了网络层，然而这似乎不太可能。

考虑下面的场景。

（1）机器A的网络层将数据包1交给它的数据链路层。机器B正确地接收到该数据包，并且将它传递给机器B上的网络层。机器B给机器A送回一个确认帧。

（2）确认帧完全丢失了，它永远也不可能到达机器A。如果信道出错后只丢数据帧而不丢控制帧，则问题就大大简化了；但遗憾的是，信道对这两种帧并没有区别对待。

（3）机器A上的数据链路层最终超时。由于它没有收到确认，所以它（不正确地）认为自己发的数据帧丢失了，或者被损坏了，于是它再次发送一个包含数据包1的帧。

（4）这个重复的帧也完好无损地到达机器B上的数据链路层，并且不知不觉中被传给了网络层。如果机器A正在给机器B发送一个文件，那么文件中的一部分内容将会被重复发送（即机器B得到的文件副本是不正确的，而且错误没有被检测出来）。换句话说，该协议失败了。

显然，对于接收方来说，它需要有一种办法能够区分到达的帧是第一次发来的新帧，还是被重传的老帧。为了做到这一点，很显然的做法是让发送方在它所发送的每个帧的头部放上一个序号。然后，接收方可以检查它所接收到的每个帧的序号，由此判断这是个新帧还是应该被丢弃的重复帧。

因为协议必须正确，并且处于链路效率的考虑包含在帧头中的序号字段可能很小，于是问题出现了：序号至少需要用多少位表示？帧头可以提供1位、数位，1个字节或多个字节作为序号，具体多少位由链路层协议自己确定。最重要的一点在于帧携带的序号必须足够大到保证协议能正确工作，否则它就不那么像一个协议。

在这个协议中，唯一不明确的地方出现在一帧（序号为m）和它的直接后续帧（序号为m+1）之间。如果m号帧被丢失，或者被损坏，接收方将不会对其进行确认，从而发送方将会不停地发送该帧。一旦该帧被正确地接收到了，接收方将给发送方返回一个确认。正是在这里可能出现潜在的问题。依据确认帧是否正确地返回发送方，发送方决定发送m号帧还是m+1号帧。

在发送方，触发其发送m+1号的事件是m号帧的确认帧按时返回了。但是这种情况隐含着m–1号帧已经被接收方正确地接收，而且它的确认帧也已经正确地被发送方接收。否则发送方就不会开始发送m号帧，更不用说发送m+1号帧了。所以，唯一不明确的地方在于一帧和它的前一帧或者和它的后一帧之间，而不在于它的前一帧和它的后一帧两者之间。

因此，一位序号（0或者1）就足以解决问题。在任何一个时刻，接收方期望下一个特定的序号。当包含正确序号的帧到来时，它被接受下来并且被传递给网络层。然后，接收方期待的下一个的序号模2增1（即0变成1，1变成0）。任何一个到达的帧，如果包含了错误序号都将作为重复帧而遭到拒绝。不过，最后一个有效的确认要被重复，以便发送方

最终发现已经被接收的那个帧。

图 3-14 给出了这类协议的一个例子。如果在一个协议中，发送方在前移到下一个数据之前必须等待一个肯定确认，这样的协议称为**自动重复请求**（ARQ，Automatic Repeat reQuest）或**带有重传的肯定确认**（PAR，Positive Acknowledgement with Retransmission）。与协议 2 类似，这类协议也只在一个方向上传输数据。

```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */
  #define MAX_SEQ1                    /* must be 1 for protocol 3 */
  typedef enum {frame_arrival, cksum_err, timeout} event_type;
  #include "protocol.h"
void sender3(void)
{
  seq_nr next_frame_to_send;          /* seq number of next outgoing frame */
  frame s;                            /* scratch variable */
  packet buffer;                      /* buffer for an outbound packet */
  event_type event;
  next_frame_to_send = 0;             /* initialize outbound sequence numbers */
  from_network_layer(&buffer);        /* fetch first packet */
  while (true) {
    s.info = buffer;                  /* construct a frame for transmission */
    s.seq = next_frame_to_send;       /* insert sequence number in frame */
    to_physical_layer(&s);            /* send it on its way */
    start_timer(s.seq);               /* if answer takes too long, time out */
    wait_for_event(&event);           /* frame_arrival, cksum_err, timeout */
    if (event == frame_arrival) {
      from_physical_layer(&s);        /* get the acknowledgement */
      if (s.ack == next_frame_to_send) {
        stop_timer(s.ack);            /* turn the timer off */
        from_network_layer(&buffer);  /* get the next one to send */
        inc(next_frame_to_send);      /* invert next_frame_to_send */
      }
    }
  }
}
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;
  frame_expected = 0;
  while (true) {
  wait_for_event(&event);             /* possibilities: frame_arrival, cksum_err */
  if (event == frame_arrival){        /* a valid frame has arrived */
    from_physical_layer(&r);          /* go get the newly arrived frame */
    if (r.seq == frame_expected){     /*this is what we have been waiting for */
      to_network_layer(&r.info);      /* pass the data to the network layer */
      inc(frame_expected);            /* next time expect the other sequence nr */
    }
   s.ack = 1 - frame_expected;
```

图 3-14　带有重传机制的肯定确认协议

协议 3 与其前任协议的不同之处在于，当发送方和接收方的数据链路层处于等待状态时，两者都用一个变量记录下了有关的值。发送方在 next_frame_to_send 中记录了下一个要发送的帧的序号；接收方则在 frame_expected 中记录了下一个期望接收的序号。每个协议在进入无限循环之前都有一个简短的初始化阶段。

发送方在发出一帧后启动计时器。如果计时器已经在运行，则将它重置，以便等待另一个完整的超时时间间隔。在选择超时值时，应该保证它足够长，确保帧到达接收方，按

照最坏的情形被接收方所处理，然后确认帧被返回发送方所需要的全部操作时间。只有当这段时间间隔过去之后，发送方才可以假定原先的数据帧或者它的确认帧已经被丢失，于是重发原先的数据帧。如果超时间隔设置得太短，发送方将会发送一些不必要的帧。虽然这些额外的帧不会影响协议的正确性，但是会严重损害协议的性能。

发送方在送出一帧并启动了计时器后，它就等待着相关事情的发生。此时只有3种可能的情况：确认帧完好无损地返回、确认帧受到损伤蹒跚而至或者计时器超时。如果到达了一个有效的确认帧，则发送方从它的网络层获取下一个数据包，并把它放入缓冲区覆盖掉原来的数据包，同时它还会递增帧的序号。如果到达了一个受损的确认帧，或者计时器超时，则缓冲区和序号都不作任何改变，以便重传原来的帧。无论如何，在这3种情况下，缓冲区里的内容被发送出去（下一个数据包或者重复数据包）。

当一个有效帧到达接收方时，接收方首先检查它的序号，确定是否为重复数据包。如果不是，则接受该数据包并将它传递给网络层，然后生成一个确认帧。重复帧和受损帧都不会被传递给网络层，但它们的到来会导致最后一个正确接收到的数据帧的确认被重复发送，返回给发送方，以便发送方做出前进到下一帧或重发那个受损帧的决策。

3.4 滑动窗口协议

在前面的协议中，数据帧只在一个方向上传输。而在大多数实际环境中，往往需要在两个方向上同时传输数据。实现全双工数据传输的一种办法是运行前面协议的两个实例，每个实例使用一条独立的链路进行单工数据传输（在不同的方向上）。因此，每条链路由一个“前向”信道（用于数据）和一个“逆向”信道（用于确认）组成。两种情况下的逆向信道带宽几乎完全被浪费了。

一种更好的做法是使用同一条链路来传输两个方向上的数据。毕竟，协议2和协议3已经在两个方向上传输帧，而且逆向信道与前向信道具有同样的容量。在这种模型中，从机器A到机器B的数据帧可以与从机器A到机器B的确认帧混合在一起。接收方只要检查入境帧头部的kind字段，就可以区别出该帧是数据帧还是确认帧。

尽管让数据帧和控制帧在同一条链路上交错传输是对前面提到的两条独立物理链路方案的一种改进，但仍然有更进一步改进的可能。当到达一个数据帧时，接收方并不是立即发送一个单独的控制帧；而是抑制自己并开始等待，直到网络层传递给它下一个要发送的数据包。然后，确认信息被附加在往外发送的数据帧上（使用帧头的ack字段）。实际上，确认信息搭了下一个出境数据帧的便车。这种暂时延缓确认以便将确认信息搭载在下一个出境数据帧上的技术就称为**捎带确认**（piggybacking）。

与单独发确认帧的方法相比，使用捎带确认的最主要好处是更好地利用了信道的可用带宽。帧头的ack字段只占用很少几位，而一个单独的帧则需要一个帧头、确认信息和校验和。而且，发送的帧越少，也意味着接收方的处理负担越轻。在下面讨论的协议中，捎带字段只占用帧头中的1位，它很少会占用许多位。

然而，捎带确认法也引入了一个在单独确认中不曾出现过的复杂问题。为了捎带一个确认，数据链路层应该等网络层传递给它下一个数据包，要等多长时间？如果数据链路层

等待的时间超过了发送方的超时间隔，那么该帧将会被重传，从而违背了确认机制的本意。如果数据链路层是一个先知者，能够预测未来，那么它就知道下一个网络层数据包什么时候会到来，因此可以确定是继续等待下去，还是立即发送一个单独的确认帧，这取决于计划的等待时间是多长。当然，数据链路层不可能预测将来，所以它必须采用某种自组织方法，比如等待一个固定的毫秒数。如果一个新的数据包很快就到来，那么确认就可立即被捎带回去。否则的话，如果在这段间隔超时之前没有新的数据包到来，数据链路层必须发送一个单独的确认帧。

接下去的 3 个协议都是双向协议，它们同属于一类称为**滑动窗口**（sliding window）的协议。这 3 个协议在效率、复杂性和缓冲区需求等各个方面有所不同，本节后面将会逐个讨论。如同所有的滑动窗口协议一样，在这 3 个协议中，任何一个出境帧都包含一个序号，范围从 0 到某个最大值。序号的最大值通常是 2^n-1，这样序号正好可以填入到一个 n 位的字段中。停-等式滑动窗口协议使用 n=1，限制了序号只能是 0 和 1，但是更加复杂的协议版本可以使用任意的 n。

所有滑动窗口协议的本质是在任何时刻发送方总是维持着一组序号，分别对应于允许它发送的帧。我们称这些帧落在**发送窗口**（sending window）内。类似地，接收方也维持着一个接收窗口（receiving window），对应于一组允许它接受的帧。发送方的窗口和接收方的窗口不必有同样的上下界，甚至也不必有同样的大小。在有些协议中，这两个窗口有固定的大小，但是在其他一些协议中，它们可以随着帧的发送和接收而增大或者缩小。

尽管这些协议使得数据链路层在发送和接收帧的顺序方面有了更多的自由度，但是我们绝对不能降低基本需求，即数据链路层协议将数据包递交给网络层的次序必须与发送机器上数据包从网络层被传递给数据链路层的次序相同。我们同样也不能改变这样的需求：物理通信信道就像一根“线”，也就是说，它必须按照发送的顺序递交所有的帧。

发送方窗口内的序号代表了那些可以被发送的帧，或者那些已经被发送但还没有被确认的帧。任何时候当有新的数据包从网络层到来时，它被赋予窗口中的下一个最高序号，并且窗口的上边界前移一格。当收到一个确认时，窗口的下边界也前移一格。按照这种方法发送窗口持续地维持了一系列未被确认的帧。图 3-15 显示了一个例子。

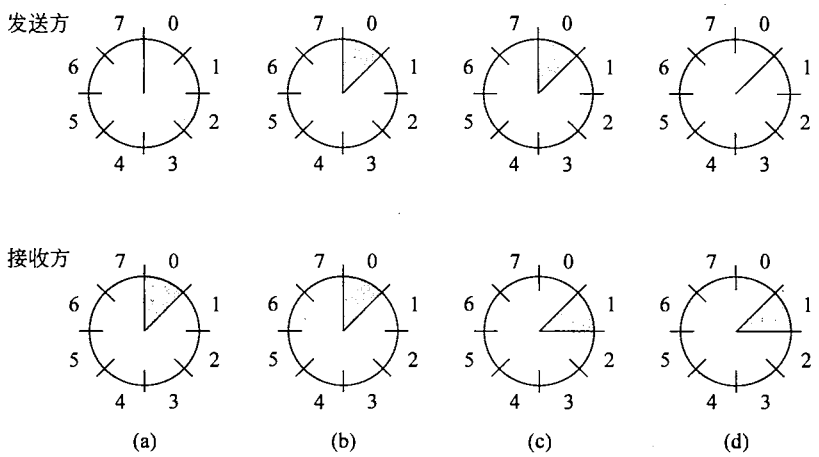


图 3-15　大小为 1 的滑动窗口，序号 3 位

（a）初始化；（b）第一帧发出之后；（c）第一帧被接收之后；（d）第一个确认被接收之后

由于当前在发送方窗口内的帧最终有可能在传输过程中丢失或者被损坏，所以，发送方必须在内存中保存所有这些帧，以便满足可能的重传需要。因此，如果最大的窗口尺寸为 n，则发送方需要 n 个缓冲区来存放未被确认的帧。如果窗口在某个时候达到了它的最大尺寸，则发送方的数据链路层必须强行关闭网络层，直到有一个缓冲区空闲出来为止。

接收方数据链路层的窗口对应于它可以接受的帧。任何落在窗口内的帧被放入接收方的缓冲区。当收到一个帧，而且其序号等于窗口下边界时，接收方将它传递给网络层，并将整个窗口向前移动 1 个位置。任何落在窗口外面的帧都将被丢弃。在所有情况下，接收方都要生成一个确认并返回给发送方，以便发送方知道该如何处理。请注意，窗口大小为 1 意味着数据链路层只能按顺序接受帧，但是对于大一点的窗口，这一条便不再成立。相反，网络层总是按照正确的顺序接受数据，跟数据链路层的窗口大小没有关系。

图 3-15 显示了一个最大窗口尺寸为 1 的例子。起初，没有帧发出，所以发送方窗口上界和下界相同；但随着时间的推移，窗口的变化进展如图所示。与发送方窗口不同，接收方窗口始终保持着它的初始大小，窗口的旋转意味着下一帧被接受并被传递到网络层。

3.4.1 1 位滑动窗口协议

在讨论一般情形以前，我们先来考察一个窗口尺寸为 1 的滑动窗口协议。由于发送方在发出一帧以后，必须等待前一帧的确认到来才能发送下一帧，所以这样的协议使用了停-等式办法。

图 3-16 描述了这样一个协议。跟其他协议一样，它也是从定义变量开始的。next_frame_to_send 指明了发送方试图发送的那一帧。类似地，frame_expected 指明了接收方等待接收的那一帧。两种情况下，0 和 1 是唯一的可能。

在一般情况下，两个数据链路层中的某一个首先开始，发送第一帧。换句话说，只有一个数据链路层程序应该在主循环外面包含 to_physical_layer 和 start_timer 过程调用。初始启动的机器从它的网络层获取第一个数据包，然后根据该数据包创建一帧，并将它发送出去。当该帧（或者任何其他帧）到达目的地，接收方的数据链路层检查该帧，看它是否为重复帧，如同协议 3 一样。如果这帧正是接收方所期望的，则将它传递给网络层，并且接收方的窗口向前滑动。

确认字段包含了最后接收到的正确帧的序号。如果该序号与发送方当前试图发送的帧的序号一致，则发送方知道，存储在 buffer 中的帧已经处理完毕，于是，它就可以从网络层获取下一个数据包。如果序号不一致，则它必须继续发送同一帧。无论什么时候只要接收到一帧，就要返回一帧。

现在我们来看协议 4 对于不正常情形的适应能力怎么样。假设计算机 A 试图将它的 0 号帧发送给计算机 B，同时 B 也试图将它的 0 号帧发送给 A。假定 A 发出一帧给 B，但是 A 的超时间隔设置得有点短。因此，A 可能会不停地超时而重发一系列相同的帧，并且所有这些帧的 seq=0，以及 ack=1。

当第一个有效帧到达计算机 B，它会被接受，并且 frame_expected 设置为 1。所有到达的后续帧都将遭到拒绝，因为 B 现在等待的是序号为 1 的帧，而不是序号为 0 的帧。而且，由于所有的重复帧都有 ack=1，并且 B 仍然在等待 0 号帧的确认，所以它不会从网络

```
/* Protocol 4 (Sliding window) is bidirectional. */
  #define MAX_SEQ1                        /* must be 1 for protocol 4 */
  typedef enum {frame_arrival, cksum_err, timeout} event_type;
  #include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;              /* 0 or 1 only */
  seq_nr frame_expected;                  /* 0 or 1 only */
  frame r, s;                             /* scratch variables */
  packet buffer;                          /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                 /* next frame on the outbound stream */
  frame_expected = 0;                     /* frame expected next */
  from_network_layer(&buffer);            /* fetch a packet from the network layer */
  s.info = buffer;                        /* prepare to send the initial frame */
  s.seq = next_frame_to_send;             /* insert sequence number into frame */
  s.ack = 1 - frame_expected;             /* piggybacked ack */
  to_physical_layer(&s);                  /* transmit the frame */
  start_timer(s.seq);                     /* start the timer running */
  while (true) {
    wait_for_event(&event);               /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival){          /* a frame has arrived undamaged */
      from_physical_layer(&r);            /* go get it */
      if (r.seq == frame_expected){       /* handle inbound frame stream */
        to_network_layer(&r.info);        /* pass packet to network layer */
        inc(frame_expected);              /* invert seq number expected next */
      }
      if (r.ack == next_frame_to_send){ /*handle outbound frame stream */
        stop_timer(r.ack);                /* turn the timer off */
        from_network_layer(&buffer);      /* fetch new pkt from network layer */
        inc(next_frame_to_send);          /* invert sender's sequence number */
      }
    }
    s.info = buffer;                      /* construct outbound frame */
    s.seq = next_frame_to_send;           /* insert sequence number into it */
    s.ack = 1 - frame_expected;           /* seq number of last received frame */
    to_physical_layer(&s);                /* transmit a frame */
    start_timer(s.seq);                   /* start the timer running */
  }
}
```

图 3-16 一个 1 位滑动窗口协议

层获取新的数据包。

在每一个被拒绝的重复帧到达后，B 向 A 发送一帧，其中包含 seq=0 和 ack=0。最后，这些帧中总会有一帧正确地到达 A，引起 A 开始发送下一个数据包。丢失的帧或者过早超时的任何一种混合出错情况都不会导致该协议向网络层递交重复的数据包、漏掉一个数据包或者死锁。协议正确！

然而，为了显示协议交换是多么的细微，我们注意到双方同时发送一个初始数据包时出现的一种极为罕见的情形。这种同步的困难程度如图 3-17 所示。图 3-17（a）显示了该协议的正常操作过程。图 3-17（b）显示了这种罕见的情形。如果 B 在发送自己的帧之前先等待 A 的第一帧，则整个过程如图 3-17（a）所示，每一帧都将被接受。

然而，如果 A 和 B 同时发起通信，则它们的第一帧就会交错，数据链路层进入图 3-17（b）描述的状态。在图 3-17（a）中，每一帧到来后都带给网络层一个新的数据包，这里没有任何重复；在图 3-17（b）中，即使没有传输错误，也会有一半的帧是重复的。类似的情形同样发生在过早超时的情况下，即使有一方明显地首先开始传输也会发生这样的情形。实际上，

如果发生多个过早超时，则每一帧都有可能被发送三次或者更多次，严重浪费了宝贵带宽。

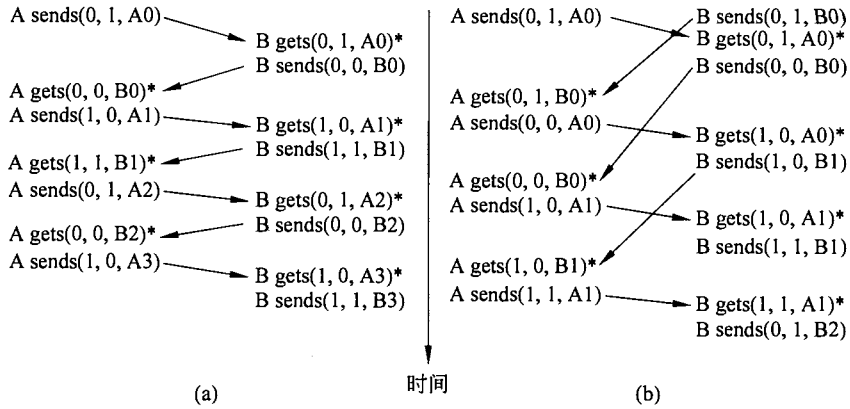


图 3-17 协议 4 的两种场景
（a）正常情况；（b）异常情况
记号表示（序号，确 认，包号），星号表示网络层接受了一个包

3.4.2 回退 N 协议

到现在为止，我们一直有这样的默契假设，即一个帧到达接收方所需要的传输时间加上确认帧回来的传输时间可以忽略不计。有时候，这种假设明显是不正确的。在这些情形下，过长的往返时间对于带宽的利用效率有严重的影响。举一个例子，考虑一个 50 kbps 的卫星信道，它的往返传播延迟为 500 毫秒。我们想象一下，在该信道上用协议 4 来发送长度为 1000 位的帧。在 t=0 时刻，发送方开始发送第一帧；在 t=20 ms 时，该帧被完全发送出去；直到 t=270 毫秒时该帧才完全到达接收方；在 t=520 毫秒时，确认帧才回到发送方，而且所有这一切还是在最好情况下才可能发生（即在接收方没有停顿并且确认帧很短）。这意味着，发送方在 500/520 或 96%的时间内是被阻塞的。换句话说，只有 4%的有效带宽被利用了。很显然，站在效率的角度，长发送时间、高带宽和短帧这三者组合在一起就是一种灾难。

这里描述的问题可以看作是这种规则的必然结果，即发送方在发送下一帧之前必须等待前一帧的确认。如果我们放松这一限制，则可以获得更好的带宽利用率。这个方案的基本思想是允许发送方在阻塞之前发送多达 w 个帧，而不是一个帧。通过选择足够大的 w 值，发送方就可以连续发送帧，因为在发送窗口被填满之前前面帧的确认就返回了，因而防止了发送方进入阻塞。

为了找到一个合适的 w 值，我们需要知道在一帧从发送方传播到接收方期间信道上能容纳多少个帧。这种容量由比特/秒的带宽乘以单向传送时间所决定，或数据链路层有责任以链路的**带宽-延迟乘积**（bandwidth-delay product）序列把数据包传递给网络层。我们可以将这个数量拆分成一帧的比特数，从而用帧的数量来表示。我们将这个数值称为 BD。因此，w 应设置为 2BD+1。如果考虑发送方连续发送帧并且在往返时间内收到一个确认，那么两倍的带宽-延时就是发送方可以连续发送的帧的个数；“+1”是因为必须接收完整个帧之后确认帧才会被发出。

在上述例子中，具有 50 kbps 的带宽和 250 毫秒的单向传输时间，带宽-延迟乘积为

12.5 kb 或 12.5 个长度为 1000 位的帧。因此，2BD+1 是 26 帧。假设发送方还和以前一样开始发送 0 号帧，并且每隔 20 毫秒发送一个新帧。到 t=520 时，它已经发送了 26 帧，这时 0 号帧的确认刚好返回。此后，每隔 20 毫秒就会到达一个确认，因此必要时发送方总是可以发送帧。从那时起，25 或 26 个未被确认的帧将始终在旅途中。换言之，发送方的最大窗口尺寸是 26。

对于较小尺寸的窗口，链路的利用率将小于 100%，因为发送方时常会被阻塞住。我们可以将链路利用率表示成发送方未被阻塞的时间比例：

$$\text{链路利用率} \leqslant \frac{w}{1+2BD}$$

这个值是个上限，因为它不容许有任何帧的处理时间，并且视确认帧的长度为零（因为它通常很短）。该方程显示无论带宽-延迟乘积多大发送方的窗口 w 一定要大。如果延迟很高，发送方将迅速耗尽窗口，即使对于卫星通信那样的中等带宽；如果带宽很高，即使对于普通延迟发送方也将很快耗尽其窗口，除非它有一个很大的发送窗口（例如，一个具有 1 毫秒延迟的 1 Gbps 链路能容纳 1 Mb）。停-等式协议的 w=1，如果延迟传播甚至只有一帧时间，协议效率都将低于 50%。

保持多个帧同时在传送的技术是**管道化**（pipelining）的一个例子。在一个不可靠的通信信道上像管道一样传送帧会引起一些严重的问题。首先，位于某个数据流中间的一个帧被损坏或丢失，会发生什么事情？在发送方发现问题之前大量的后续帧已经发出，并且即将到达接收方。当损坏的那个帧到达接收方时，显然它应该被丢弃，但接收方该如何处理所有那些后续到达的正确帧呢？请记住，接收方的数据链路层有责任按正确的顺序把数据包传递给网络层。

有两种基本办法可用来处理管道化传输中出现的错误，图 3-18 显示了这两种技术。

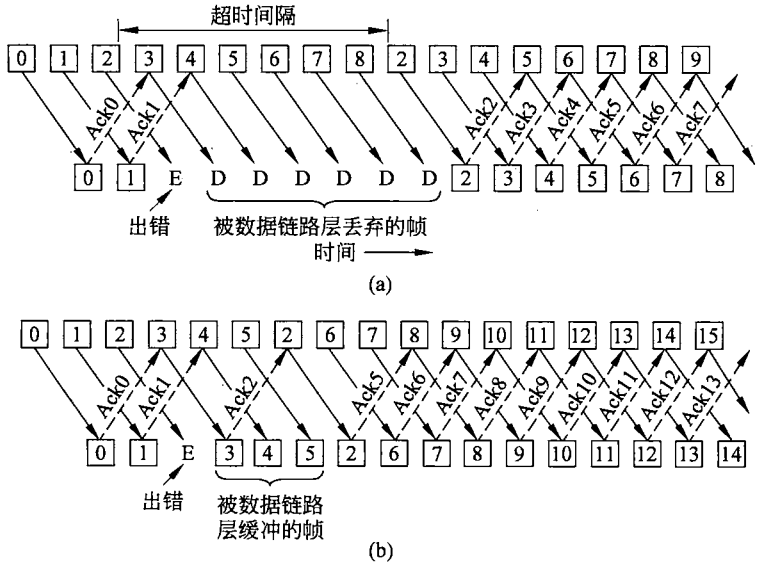


图 3-18　管道化以及差错恢复

（a）接收方窗口为 1 时错误的影响；（b）接收方窗口很大时错误的影响

一种选择办法称为**回退 n**（go-back-n），接收方只需简单丢弃所有到达的后续帧，而且

针对这些丢弃的帧不返回确认。这种策略对应于接收窗口大小为 1 的情形。换句话说，除了数据链路层必须要递交给网络层的下一帧以外，它拒绝接受任何帧。如果在计时器超时以前，发送方的窗口已被填满，则管道将变为空闲。最终，发送方将超时，并且按照顺序重传所有未被确认的帧，从那个受损或者被丢失的帧开始。如果信道的错误率很高，这种方法会浪费大量的带宽。

在图 3-18（b）中，我们可以看到回退 n 帧的情形，其中接收方的窗口比较大。0 号帧和 1 号帧被正确地接收和确认。然而，2 号帧被损坏或者丢失。发送方并有意识到出现了问题，它继续发送后续的帧，直到 2 号帧的计时器超时。然后，它退回到 2 号帧，从这里重新发送 2 号、3 号、4 号帧等，一切从头再来。

针对管道化发送帧发生的错误，另一种通用的处理策略称为**选择重传**（selective repeat）。使用这种策略，接收方将收到的坏帧丢弃，但接受并缓存坏帧后面的所有好帧。当发送方超时，它只重传那个最早的未被确认的帧。如果该重传的帧正确到达接收方，接收方就可按序将它缓存的所有帧递交给网络层。选择重传对应的接收方窗口大于 1。如果窗口很大，则这种方法对数据链路层的内存需求很大。

选择重传策略通常跟否定策略结合起来一起使用，即当接收方检测到错误（例如，帧的校验和错误或者序号不正确），它就发送一个**否定确认**（NAK，negative acknowledgement）。NAK 可以触发该帧的重传操作，而不需要等到相应的计时器超时，因此协议性能得以提高。

在图 3-18（b）中，0 号帧和 1 号帧被正确接收，并得到确认；2 号帧丢失了。当 3 号帧到达接收方时，那里的数据链路层注意到自己错过了一帧，所以它针对错失的 2 号帧返回一个 NAK，但是将第 3 帧缓存起来。当 4 号和 5 号帧到达之后，它们也被数据链路层缓存起来，而没有被传递给网络层。2 号帧的 NAK 抵达发送方后，发送方立即重传 2 号帧。当该帧到达接收方时，数据链路层现在有了 2 号、3 号、4 号和 5 号帧，于是就将它们按照正确的顺序传递给网络层，也可以确认所有这些帧（从 2 号帧到 5 号帧），如图中所示。如果 NAK 被丢失，则发送方的 2 号帧计时器最终超时，发送方就会重新发送 2 号帧（仅仅这一帧），但是，这可能已经过了相当长一段时间。

这两种不同方法恰好是带宽使用效率和数据链路层缓存空间之间的权衡。根据资源的紧缺程度，选择使用其中一种方法。图 3-19 显示了一个回退 n 协议的示例，其中接收方的

```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may
   transmit up to MAX_SEQ frames without waiting for an ack. In addition,
   unlike in the previous protocols, the network layer is not assumed to have
   a new packet all the time. Instead, the network layer causes a network_layer
   _ready event when there is a packet to send. */

#define MAX_SEQ 7
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"
static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <= b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
    return(true);
  else
    return(false);
}
static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
```

图 3-19 一个采用了回退 n 机制的滑动窗口协议

```
{
/* Construct and send a data frame. */
  frame s;                              /* scratch variable */
  s.info = buffer[frame_nr];            /* insert packet into frame */
  s.seq = frame_nr;                     /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1); /* piggyback ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(frame_nr);                /* start the timer running */
}
void protocol5(void)
{
  seq_nr next_frame_to_send;            /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                  /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;                /* next frame expected on inbound stream */
  frame r;                              /* scratch variable */
  packet buffer[MAX_SEQ + 1];           /* buffers for the outbound stream */
  seq_nr nbuffered;                     /* number of output buffers currently in use */
  seq_nr i;                             /* used to index into the buffer array */
  event_type event;
  enable_network_layer();               /* allow network_layer_ready events */
  ack_expected = 0;                     /* next ack expected inbound */
  next_frame_to_send = 0;               /* next frame going out */
  frame_expected = 0;                   /* number of frame expected inbound */
  nbuffered = 0;                        /* initially no packets are buffered */
  while (true) {
    wait_for_event(&event);             /* four possibilities: see event_type above */
    switch(event) {
    case network_layer_ready:           /* the network layer has a packet to send */
      /* Accept, save, and transmit a new frame. */
      from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
      nbuffered = nbuffered + 1;        /* expand the sender's window */
      send_data(next_frame_to_send, frame_expected, buffer);/* transmit the
      frame */ inc(next_frame_to_send);/* advance sender's upper window edge */
      break;
    case frame_arrival:                 /* a data or control frame has arrived */
      from_physical_layer(&r);          /* get incoming frame from physical layer */
      if (r.seq == frame_expected) {
        /* Frames are accepted only in order. */
        to_network_layer(&r.info);      /* pass packet to network layer */
        inc(frame_expected);            /* advance lower edge of receiver's window */
      }
      /* Ack n implies n - 1, n - 2, etc. Check for this. */
      while (between(ack_expected, r.ack, next_frame_to_send)) {
        /* Handle piggybacked ack. */
        nbuffered = nbuffered - 1;      /* one frame fewer buffered */
        stop_timer(ack_expected);       /* frame arrived intact; stop timer */
        inc(ack_expected);              /* contract sender's window */
      }
      break;
    case cksum_err: break;              /* just ignore bad frames */
    case timeout:                       /* trouble; retransmit all outstanding frames */
      next_frame_to_send = ack_expected;  /* start retransmitting here */
      for (i = 1; i <= nbuffered; i++) {
        send_data(next_frame_to_send, frame_expected, buffer);/* resend frame */
      inc(next_frame_to_send);          /* prepare to send the next one */
      }
    }
    if (nbuffered < MAX_SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
```

图 3-19 （续）

数据链路层按序接收入境帧，发生错误之后的所有入境帧都将被丢弃。在这个协议中，我们第一次抛掉了网络层总是有无穷多的数据包要发送的假设。当网络层希望发送一个数据包时，它触发一个 network_layer_ready 事件。然而，为了使得流量控制机制可以限制任何时候的发送窗口大小或者未确认的出境帧个数，数据链路层必须能够阻止网络层给予它过多的工作。库过程 enable_network_layer 和 disable_network_layer 可以完成这样的功能。

任何时候，可以发送的帧的最大个数不能等同于序号空间的大小。对回退 n 协议，可发送的帧最多为 MAX_SEQ 个，即使存在 MAX_SEQ+1 个不同的序号（分别为 0、1、2、…、MAX_SEQ）。我们马上就看到对于下一个选择重传协议，这种限制更严格。为了看清楚为什么必须有这个限制，请考虑下面 MAX_SEQ=7 的场景：

（1）发送方发送 0～7 号共 8 个帧。

（2）7 号帧的捎带确认返回到发送方。

（3）发送方发送另外 8 个帧，其序列号仍然是 0～7。

（4）现在 7 号帧的另一个捎带确认也返回了。

问题出现了：属于第二批的 8 个帧全部成功到达了呢，还是这 8 个帧全部丢失了（把出错之后丢弃的帧也算作丢失）？在这两种情况下，接收方都会发送针对 7 号帧的确认，但发送方却无从分辨。由于这个原因，未确认帧的最大数目必须限制为不能超过 MAX_SEQ。

虽然协议 5 没有缓存出错后到来的帧，但是它也没有因此完全摆脱缓存问题。由于发送方可能在将来的某个时刻要重传所有未被确认的帧，所以，它必须把已经发送出去的帧一直保留，直到它能肯定接收方已经接受了这些帧。当 n 号帧的确认到达，n–1 号帧、n–2 号帧等都会自动被确认。这种类型的确认称为**累计确认**（cumulative acknowledgement）。当先前一些捎带确认的帧被丢失或者受损之后，这个特性显得尤为重要。每当到达任何一个确认，数据链路层都要检查看是否可以释放出一些缓冲区。如果能释放出一些缓冲区（即窗口中又有了一些可以利用的空间），则原来被阻塞的网络层又可以激发更多的 network_layer_ready 事件了。

对于这个协议，我们假设链路上总是有反向的数据流量可以捎带确认。协议 4 不需要这样的假设，因为它每次接收到一帧就送回一帧，即使它刚刚发送出一帧也必须再发一个帧。在下一个协议中，我们将用一种非常巧妙的办法来解决这个单向流量问题。

因为协议 5 有多个未被确认的帧，所以逻辑上它需要多个计时器，即每一个未被确认的帧都需要一个计时器。每一帧的超时是独立的，相互之间没有关系。然而，用软件很容易模拟所有这些计时器：只需使用一个硬件时钟，它周期性地引发中断。所有未发生的超时事件构成了一个链表，链表中的每个节点包含了离超时还有多少个时钟滴答、超时对应的那个帧，以及一个指向下一个节点的指针。

为了示范如何实现多个计时器，请考虑图 3-20（a）中的例子。假设每 1 毫秒时钟滴答一次。初始时，实际时间为 10:00:00.000；有三个超时事件，分别定在 10:00:00.005、10:00:00.013 和 10:00:019。每当硬件时钟滴答一次，实际的时间就被更新，链表头上的滴答计数器被减 1。当滴答计数器变成 0 的时候，就引发一个超时事件，并将该节点从链表中移除，如图 3-20（b）所示。虽然这种实现方式要求在调用 start_timer 或 stop_timer 时扫描链表，但在每次滴答中断时并不要求更多的工作。在协议 5 中，start_timer 和 stop_timer 这两个过程都带一个参数，表示针对哪一帧计时。

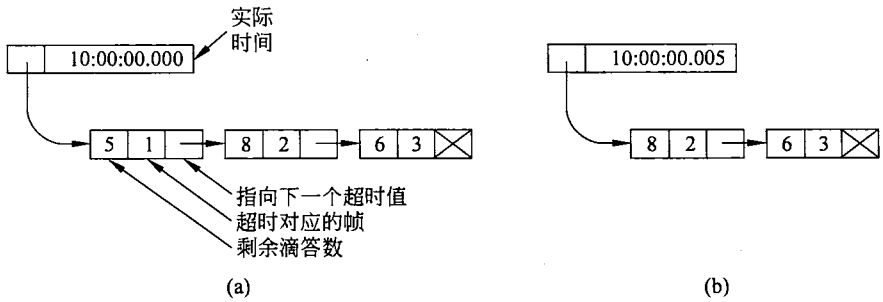


图 3-20　软件模拟多个计时器
（a）队列中的计时器；（b）第一个计时器超时后的情形

3.4.3　选择重传协议

如果错误很少发生，则回退 n 协议可以工作得很好；但是，如果线路质量很差，那么重传的帧要浪费大量带宽。另一种处理错误的策略是选择重传协议，允许接收方接受并缓存坏帧或者丢失帧后面的所有帧。

在这个协议中，发送方和接收方各自维持一个窗口，该窗口分别包含可发送或已发送但未被确认的和可接受的序号。发送方的窗口大小从 0 开始，以后可以增大到某一个预设的最大值。相反，接收方的窗口总是固定不变，其大小等于预先设定的最大值。接收方为其窗口内的每个序号保留一个缓冲区。与每个缓冲区相关联的还有一个标志位（arrived），用来指明该缓冲区是满的还是空的。每当到达一帧，接收方通过 between 函数检查它的序号，看是否落在窗口内。如果确实落在窗口内，并且以前没有接收过该帧，则接受该帧，并且保存在缓冲区。不管这帧是否包含了网络层所期望的下一个数据包，这个过程肯定要执行。当然，该帧只能被保存在数据链路层中，直到所有序号比它小的那些帧都已经按序递交给网络层之后，它才能被传递给网络层。图 3-21 显示了一个使用该算法的协议。

```
/*Protocol 6 (Selective repeat) accepts frames out of order but passes packets to
the network layer in order. Associated with each outstanding frame is a timer.
When the timer expires, only that frame is retransmitted, not all the outstanding
frames, as in protocol 5. */

#define MAX_SEQ7                        /* should be 2^n - 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout}
event_type;
#include "protocol.h"
boolean no_nak = true;                  /* no nak has been sent yet */
seq_nr oldest_frame = MAX_SEQ + 1;  /* initial value is only for the simulator */
static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol 5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}
static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected,
packet buffer[ ])
{
  /* Construct and send a data, ack, or nak frame. */
  frame s;                              /* scratch variable */
```

图 3-21　一个采用了选择重传机制的滑动窗口协议

```
  s.kind = fk;                          /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                     /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;        /* one nak per frame, please */
  to_physical_layer(&s);                /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                     /* no need for separate ack frame */
}
void protocol6(void)
{
  seq_nr ack_expected;                  /* lower edge of sender's window */
  seq_nr next_frame_to_send;            /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                /* lower edge of receiver's window */
  seq_nr too_far;                       /* upper edge of receiver's window + 1 */
  int i;                                /* index into buffer pool */
  frame r;                              /* scratch variable */
  packet out_buf[NR_BUFS];              /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];               /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];             /* inbound bit map */
  seq_nr nbuffered;                     /* how many output buffers currently used */
  event_type event;
  enable_network_layer();               /* initialize */
  ack_expected = 0;                     /* next ack expected on the inbound stream */
  next_frame_to_send = 0;               /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                        /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
  while (true) {
    wait_for_event(&event);             /* five possibilities: see event_type above */
    switch(event) {
      case network_layer_ready:         /* accept, save, and transmit a new frame */
        nbuffered = nbuffered + 1;      /* expand the window */
        from_network_layer(&out_buf[next frame to_send % NR_BUFS]);
                                        /* fetch new packet */
        send_frame(data, next_frame_to_send, frame_expected, out_buf);
                                        /* transmit the frame */
        inc(next_frame_to_send);        /* advance upper window edge */
        break;
      case frame_arrival:               /* a data or control frame has arrived */
        from_physical_layer(&r);        /* fetch incoming frame from physical layer */
        if (r.kind == data) {
          /* An undamaged frame has arrived. */
          if ((r.seq != frame_expected) && no_nak)
            send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
          if (between(frame_expected,r.seq,too_far) && (arrived[r.seq%NR_BUFS]==
            false)) {
          /* Frames may be accepted in any order. */
            arrived[r.seq % NR_BUFS] = true; /* mark buffer as full */
            in_buf[r.seq % NR_BUFS] ='r.info; /* insert data into buffer */
            while (arrived[frame_expected % NR_BUFS]) {
              /* Pass frames and advance window. */
              to_network_layer(&in_buf[frame_expected % NR_BUFS]);
              no_nak = true;
              arrived[frame_expected % NR_BUFS] = false;
              inc(frame_expected);  /* advance lower edge of receiver's window */
              inc(too_far);         /* advance upper edge of receiver's window */
              start_ack_timer();    /* to see if a separate ack is needed */
            }
          }
        }
        if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next_frame_
        to_send))
          send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);
        while (between(ack_expected, r.ack, next_frame_to_send)) {
          nbuffered = nbuffered - 1;   /* handle piggybacked ack */
```

图 3-21 （续）

```
    stop_timer(ack_expected % NR_BUFS);  /* frame arrived intact */
    inc(ack_expected);   /* advance lower edge of sender's window */
   }
   break;
  case cksum_err:
   if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* damaged frame */
   break;
  case timeout:
   send_frame(data, oldest_frame, frame_expected, out_buf); /* we timed out */
   break;
  case ack_timeout:
   send_frame(ack,0,frame_expected, out_buf);  /* ack timer expired; send ack */
 }
 if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
}
}
```

图 3-21 （续）

非顺序接收引发了一些特殊问题，这些问题对于那些按顺接受帧的协议是不用考虑的。我们用一个例子就很容易说明麻烦之处。假设我们用 3 位序号，那么发送方允许连续发送 7 个帧，然后开始等待确认。刚开始时，发送方和接收方的窗口如图 3-22（a）所示。现在发送方发出 0～6 号帧。接收方的窗口允许它接受任何序号落在 0～6（含）之间的帧。这 7 个帧全部正确地到达了，所以接收方对它们进行确认，并且向前移动它的窗口，允许接收 7、0、1、2、3、4 或 5 号帧，如图 3-22（b）所示。所有这 7 个缓冲区都标记为空。

此时，灾难降临了，闪电击中了电线杆子，所有的确认都被摧毁。协议应该不管灾难是否发生都能正确工作。最终发送方超时，并且重发 0 号帧。当这帧到达接收方时，接收方检查它的序号，看是否落在窗口中。不幸的是，如图 3-22（b）所示，0 号帧落在新窗口中，所以它被当作新帧接受了。接收方同样返回（捎带）6 号帧确认，因为 0～6 号帧都已经接收到了。

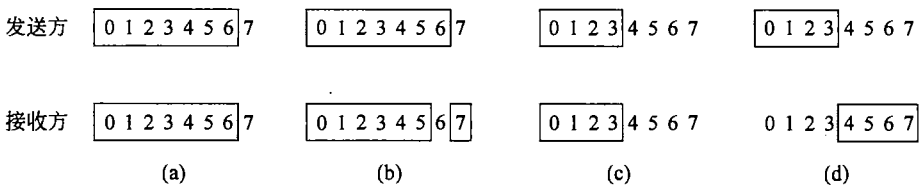


图　3-22

（a）窗口大小为 7 的初始情形；（b）发出 7 个帧并且接收 7 帧，但尚未确认；
（c）窗口大小为 4 的初始情形；（d）发出 4 个帧并且接收 4 帧，但尚未确认

发送方很高兴地得知所有它发出去的帧都已经正确地到达了，所以它向前移动发送窗口，并立即发送 7、0、1、2、3、4 和 5 号帧。7 号帧将被接收方接收，并且它的数据包直接传递给网络层。紧接着，接收方的数据链路层进行检查，看它是否已经有一个有效的 0 号帧，它发现确实已经有了（即前面重发的 0 号帧），然后把内嵌的数据包作为新的数据包传递给网络层。因此，网络层得到了一个不正确的数据包。协议失败！

这个问题的本质在于：当接收方向前移动它的窗口后，新的有效序号范围与老的序号范围有重叠。因此，后续的一批帧可能是重复的帧（如果所有的确认都丢失了），也可能是新的帧（如果所有的确认都接收到了）。可怜的接收方根本无法区分这两种情形。

解决这个难题的办法是确保接收方向前移动窗口之后，新窗口与老窗口的序号没有重叠。为了保证没有重叠，窗口的最大尺寸应该不超过序号空间的一半，如图 3-22（c）和图 3-22（d）所做的那样。如果用 3 位来表示序号，则序号范围为 0～7。在任何时候，应该只有 4 个未被确认的帧。按照这种方法，如果接收方已经接收了 0～3 号帧，并且向前移动了窗口，以便允许接收第 4～7 帧，那么，它可以明确地区分出后续的帧是重传帧（序号为 0～3），还是新帧（序列号为 4～7）。一般来说，协议 6 的窗口尺寸为（MAX_SEQ+1）/2。

一个有意思的问题是：接收方必须拥有多少个缓冲区？无论如何，接收方不可能接受序号低于窗口下界的帧，也不可能接受序号高于窗口上界的帧。因此，所需要的缓冲区的数量等于窗口的大小，而不是序号的范围。在前面的 3 位序号例子中，只需要 4 个缓冲区就够了，编号为 0～3。当 i 号的帧到达时，它被放在（i mod 4）号缓冲区中。请注意，虽然 i 和（i+4）mod 4 在“竞争”同一个缓冲区，但它们永远不会同时落在窗口内，因为如果那样的话，则窗口的尺寸至少为 5。

出于同样的原因，需要的计时器数量等同于缓冲区的数量，而不是序号空间的大小。实际上，每个缓冲区都有一个相关联的计时器。当计时器超时，缓冲区的内容就要被重传。

协议 6 还放宽了隐含假设，即信道的负载很重。我们在协议 5 中作了这个假设，因为协议要依赖反方向上的数据帧来捎带确认信息。如果逆向流量很轻，确认会被延缓很长一段时间，这将导致问题。在极端的情况下，如果在一个方向上有很大的流量，而另一个方向上根本没有流量，那么当发送方的窗口达到最大值后协议将被阻塞。

为了不受该假设的约束，当一个按正常次序发送的数据帧到达接收方之后，接收方通过 start_ack_timer 启动一个辅助的计时器。如果在计时器超时之前，没有出现需要发送的反向流量，则发送一个单独的确认帧。由该辅助计时器超时而导致的中断称为 ack_timeout 事件。在这样的处理方式下，缺少可以捎带确认的反向数据帧不再是一个问题，因此那些只存在单向数据流的场合依然可以使用该协议。只要一个辅助计时器就可以正常工作，如果该计时器在运行时又调用了 start_ack_timer，则该次调用不起作用。计时器不会被重置或者被扩展，因为它的作用是提供确认的某种最小速率。

辅助计时器的超时间隔应该明显短于与数据帧关联的计时器间隔，这点非常关键。这是确保下列情况的必要条件：即一个被正确接收的帧应该尽早地被确认，使得该帧的重传计时器不会超时因而不会重传该帧。

协议 6 采用了比协议 5 更加有效的策略来处理错误。当接收方有理由怀疑出现了错误时，它就给发送方返回一个否定确认（NAK）帧。这样的帧实际上是一个重传请求，在 NAK 中指定了要重传的帧。在两种情况下，接收方要特别注意：接收到一个受损的帧，或者到达的帧并非是自己所期望的（可能出现丢帧错误）。为了避免多次请求重传同一个丢失帧，接收方应该记录下对于某一帧是否已经发送过 NAK。在协议 6 中，如果对于 frame_expected 还没有发送过 NAK，则变量 no_nak 为 true。如果 NAK 被损坏了，或者丢失了，则不会有实质性的伤害，因为发送方最终会超时，无论如何它都会重传丢失的帧。如果一个 NAK 被发送出去之后丢失了，而接收方又收到一个错误的帧，则 no_nak 将为 true，并且辅助计时器将被启动。当该辅助计时器超时后，一个 ACK 帧将被发送出去，以便使发送方重新同步到接收方的当前状态。

在某些情形下，从一帧被发出去开始算起，到该帧经传播抵达目的地、在那里被处理，

然后它的确认被传回来，整个过程所需要的时间（几乎）是个常数。在这些情形下，发送方可以把它的计时器调得“紧”一些，让它恰好略微大于正常情况下从发送一帧到接收到其确认之间的时间间隔。此时 NAC 的作用就微乎其微了。

然而，在其他一些情况下这段时间的变化非常大。例如，如果反向流量零零散散，那么如果刚好有逆向流量则接收到帧延缓发送确认之前的这段时间会比较短；反之，如果没有逆向流量，则这段时间会很长。因此发送方必须做出选择，要么将计时器的间隔设置得比较小（其风险是不必要的重传），要么将它设置得比较大（发生错误之后长时间地空等）。两种选择都会浪费带宽。一般来说，当确认间隔的标准偏差与间隔本身相比非常大的时候，计时器应该设置得“松”一点。然后，NAK 可以加快丢失帧或者损坏帧的重传速度。

与超时和 NAK 紧密相关的一个问题是确定由哪一帧引发超时。在协议 5 中，引发超时的帧总是 ack_expected 指定的帧，因为它总是最早的那个帧。在协议 6 中，没有一种很具体的办法来确定谁来引发超时。假定已经发送了 0～4 号帧，这意味着未确认帧的列表是 01234，按照时间先后顺序排列。现在想象这样的情形：0 号帧超时，5 号帧（新帧）被发送出去，1 号帧超时，2 号帧超时，6 号帧（又一新帧）被发送出去。这时候，未确认帧的列表是 3405126，也是按照从最老的帧到最新的帧顺序排列。如果所有入境流量（即那些包含确认的帧）一下子全部被丢失，那么这 7 个未确认的帧将会依次超时。

为了避免使该例子过于复杂，我们没有显示计时器的管理过程。相反，我们只是假设在超时变量 oldest_frame 设置好之后，它就能指出哪一帧超时。

3.5　数据链路协议实例

在单个建筑物内，局域网被广泛用于互连主机，但大多数广域网的基础设施是以点到点方式建设的。我们将在第 4 章重点关注局域网，这里要考察的是那些出现在 Internet 两种常见情形下的数据链路协议，这些协议主要用在点到点的线路上。第一种情形是通过广域网中的 SONET 光纤链路发送数据包。例如，这些链路被广泛用于连接一个 ISP 网络中位于不同位置的路由器。

第二种情形是运行在 Internet 边缘的电话网络本地回路上的 ADSL 链路。这些链接把成千上百万的个人和企业连接到 Internet 上。

针对上述这些使用场景 Internet 需要点到点链路，还会用到拨号调制解调器、租用线路和线缆调制解调器等。所谓**点到点协议**（PPP，Point-to-Point Protocol）的标准协议就是使用这些链路来发送数据包。PPP 由 RFC1661 定义，并在 RFC1662 中得到进一步的阐述（Simpson，1994a，1994b）。SONET 和 ADSL 链路都采用了 PPP，但在使用方式上有所不同。

3.5.1　SONET 上的数据包

2.6.4 章节涉及的 SONET 是物理层协议，它最常被用在广域网的光纤链路上，这些光纤链路构成了通信网络的骨干网，其中包括电话系统。它提供了一个以定义良好速度运行

的比特流，比如 2.4 Gbps 的 OC-48 链路。比特流被组织成固定大小字节的有效载荷，不论是否有用户数据需要发送，每隔 125 微秒要发出一个比特流。

为了在这些链路上承载数据包，需要某种成帧机制，以便将偶尔出现的数据包从传输它们的连续比特流中区分出来。运行在 IP 路由器上的 PPP 就提供了这种运行机制，如图 3-23 所示。

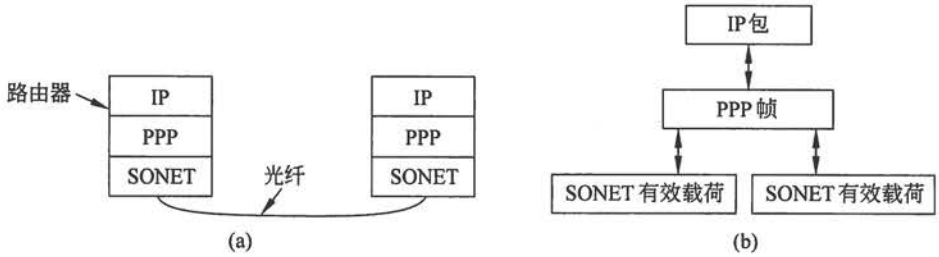


图 3-23 SONET 之上的数据包
（a）协议栈；（b）帧关系

PPP 功能包括处理错误检测链路的配置、支持多种协议、允许身份认证等。它是一个早期简化协议的改进，那个协议称为**串行线路 Internet 协议**（SLIP，Serial Line Internet Protocol）。伴随着一组广泛选项，PPP 提供了 3 个主要特性：

（1）一种成帧方法。它可以毫无歧义地区分出一帧的结束和下一帧的开始。

（2）一个链路控制协议。它可用于启动线路、测试线路、协商参数，以及当线路不再需要时温和地关闭线路。该协议称为**链路控制协议**（LCP，Link Control Protocol）。

（3）一种协商网络层选项的方式。协商方式独立于网络层协议。所选择的方法是针对每一种支持的网络层都有一个不同的**网络控制协议**（NCP，Network Control Protocol）。

因为没有必要重新发明轮子，所以 PPP 帧格式的选择酷似 HDLC 帧格式。HDLC 是**高级数据链路控制协议**（High-level Data Link Control），是一个早期被广泛使用的家庭协议实例。

PPP 和 HDLC 之间的主要区别在于：PPP 是面向字节而不是面向比特的。特别是 PPP 使用字节填充技术，所有帧的长度均是字节的整数倍。HDLC 协议则使用比特填充技术，允许帧的长度不是字节的倍数，例如 30.25 字节。

然而，实际上还有第二个主要区别。HDLC 协议提供了可靠的数据传输，所采用的方式正是我们已熟悉的滑动窗口、确认和超时机制等。PPP 也可以在诸如无线网络等嘈杂的环境里提供可靠传输，具体细节由 RFC1663 定义。然而，实际上很少这样做。相反，Internet 几乎都是采用一种“无编号模式”来提供无连接无确认的服务。

PPP 帧格式如图 3-24 所示。所有的 PPP 帧都从标准的 HDLC 标志字节 0x7E（01111110）开始。标志字节如果出现在 Payload 字段，则要用转义字节 0x7D 去填充；然后将紧跟在后面的那个字节与 0x20 进行 XOR 操作，如此转义使得第 6 位比特反转。例如 0x7D 0x5E 是标志字节 0x7E 的转义序列。这意味着只需要简单扫描 0x7E 就能找出帧的开始和结束之处，因为这个字节不可能出现在其他地方。接收到一个帧后要去掉填充字节。具体做法是，扫描搜索 0x7D，发现后立即删除；然后用 0x20 对紧跟在后面的那个字节进行 XOR 操作。此外，两个帧之间只需要一个标志字节。当链路上没有帧在发送时，可以用多个标志字节填充。

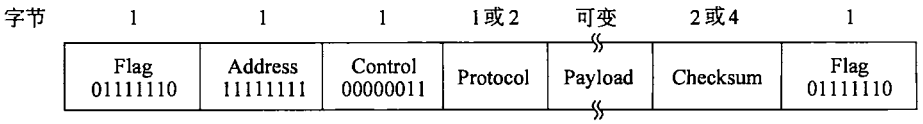


图 3-24　无编号模式操作下的完整 PPP 帧格式

紧跟在帧开始处标记字节后面出现的是 Address（地址）字段。这个字段总是被设置为二进制值 11111111，表示所有站点都应该接受该帧。使用这个值可避免如何为数据链路层分配地址这样的问题。

Address 字段后面是 Control（控制）字段，其默认值是 00000011。此值表示一个无编号帧。

因为 Address 和 Control 字段总是取默认配置的常数，因此 LCP 提供了某种必要的机制，允许通信双方就是否省略这两个字段进行协商，去掉的话可以为每帧节省 2 个字节的空间。

PPP 的第四个字段是 Protocol（协议）字段。它的任务是通告 Payload 字段中包含了什么类型的数据包。以 0 开始的编码定义为 IP 版本 4、IP 版本 6 以及其他可能用到的网络层协议，比如 IPX 和 AppleTalk。以 1 开始的编码被用于 PPP 配置协议，包括 LCP 和针对每个网络层协议而设置的不同 NCP。Protocol 字段的默认大小为 2 个字节，但它可以通过 LCP 协商减少到 1 个字节。协议设计师们也许过于谨慎了，认为有一天可能使用超过 256 个协议。

Payload（有效载荷）字段是可变长度的，最高可达某个协商的最大值。如果在链路建立时没有通过 LCP 协议进行协商，则采用默认长度 1500 字节。如果需要，在有效载荷后填充字节以便满足帧的长度要求。

Payload 字段后是 Checksum（校验和）字段，它通常占 2 个字节，但可以协商使用 4 个字节的校验和。事实上，4 字节的校验和与 32 位的 CRC 相同，其生成多项式就是 3.3.2 章节末尾给出的那个多项式。2 字节的校验和也是一个工业标准 CRC。

PPP 是一个成帧机制，它可以在多种类型的物理层上承载多种协议的数据包。为了在 SONET 上使用 PPP，RFC2615 列出了一些可用的选择（Malis 和 Simpson，1999）。因为这是检测物理层、数据链路层和网络层传输错误的主要手段，因此采用了 4 字节的校验和。它还建议不要压缩 Address、Control 和 Protocol 字段，因为 SONET 链路的运行速度已经达到很高了。

PPP 还有一个不同寻常的特点。PPP 的有效载荷在插入到 SONET 的有效载荷之前先进行扰码操作（正如 2.5.1 所描述的那样）。用长伪随机序列对有效载荷进行扰码 XOR 之后再传送出去。问题在于为了保持同步 SONET 比特流，必须频繁地进行比特跳变。这些跳变很自然地与语音信号的变化结合在一起，但在数据通信中用户选择发送的信息和数据包可能包含一长串的 0。有了扰频技术，用户因为发送一长串 0 而导致问题的可能性降到极低。

在通过 SONET 线路发送 PPP 帧之前，必须建立和配置 PPP 链路。PPP 链路的启动、使用和关闭的一系列阶段如图 3-25 所示。

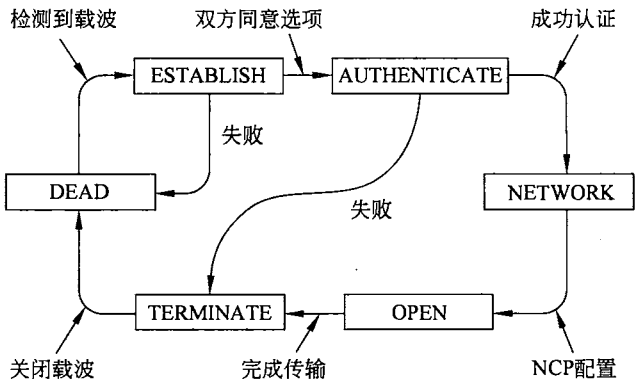


图 3-25 PPP 链路建立到释放的状态转换图

链路的初始状态为 DEAD（死），这意味着不存在物理层连接。当物理连接被建立起来，链路转移到 ESTABLISH（建立）状态。此时，PPP 对等实体交换一系列的 LCP 报文进行上面所说的那些 PPP 选项的协商，这些 LCP 报文放在 PPP 帧的 Payload 字段。初始发起连接的实体提出自己的选项请求，对等实体可以部分或者全部接受，甚至全部拒绝，同时它也可以提出自己的选项要求。

如果 LCP 选项协商成功，链路状态进入 AUTHENTICATE（认证）状态。现在，如果需要，双方可以互相检查对方的身份。如果认证成功，则链路进入 NETWORK（网络）状态，通过发送一系列的 NCP 包来配置网络层参数。NCP 协议很难一概而论，因为每个协议特定于实际采用的某个网络层协议，允许执行针对特定于该网络层协议的配置请求。例如，对于 IP 协议而言，为链路的两端分配 IP 地址是最重要的可能操作。

一旦进入 OPEN（打开）状态，双方就可以进行数据传输。正是在这个状态下，IP 数据包被承载在 PPP 帧中通过 SONET 线路传输。当完成数据传输后，链路进入 TERMINATE（终止）状态；当物理层连接被舍弃后从这里回到 DEAD 状态。

3.5.2 对称数字用户线

ADSL 以 Mbps 速率将百万计的家庭用户连接到 Internet 上，并且使用的是与普通老式电话服务相同的本地回路。在 2.5.3 节我们介绍了如何把一种称为 DSL 调制解调器的设备安置在家庭一端。它通过本地回路发出的数据比特到达一个称为 DSLAM（DSL 接入复用器，发音为“dee-slam”）的设备，该设备安置在电话公司端局。现在，我们将更详细地探讨如何在 ADSL 链路上承载数据包。

ADSL 使用的协议和设备的概貌如图 3-26 所示。不同的网络可以部署不同的协议，所以我们选择了最流行的场景。在家里，比如 PC 机那样的计算机使用以太网链路把 IP 数据包发送到 DSL 调制解调器；然后 DSL 调制解调器通过本地回路把 IP 数据包发送到 DSLAM，发送数据包所用的协议就是我们将要学习的协议。在 DSLAM 设备（或连接到它路由器，视具体实施而定）上 IP 数据包被提取出来，并被注入到 ISP 网络，因此它们最终能到达 Internet 上的任何目的地。

在图 3-26 中，ADSL 链路之上的协议底部是 ADSL 物理层。它们基于称为正交频分复

用（也称为离散多音）的数字调制方案，就像我们在 2.5.3 节看到的那样。接近协议栈顶部，恰好位于 IP 网络层正下方的是 PPP。该协议与我们刚刚考察过在 SONET 上传输数据包的 PPP 相同。它以同样的方式来建立和配置链路，以及运载 IP 数据包。

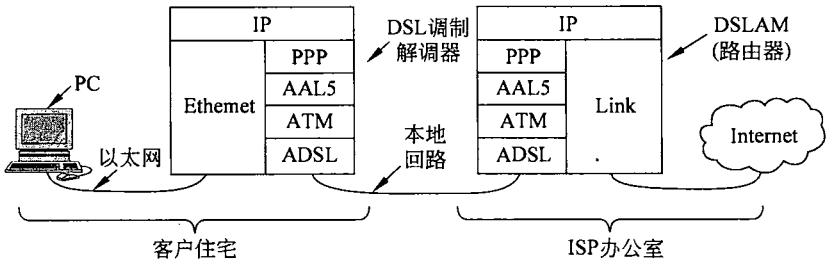


图 3-26　ADSL 协议栈

在 ADSL 和 PPP 两者之间是的 ATM 和 AAL5。这些是我们以前从未见过的新协议。**异步传输模式**（ATM，Asynchronous Transfer Mode）早在 20 世纪 90 年代初就被设计了出来，并且以令人难以置信的炒作方式公布于众。它承诺的网络技术可以解决世界上的电信问题，它把语音、数据、有线电视、电报、信鸽、串起来的罐头、击鼓，以及其他所有一切合并成一个综合系统，声称能为每个人做每件事。但事实并非如此。在很大程度上，ATM 的问题类似于我们介绍的 OSI 协议，也就是说，错误的时机、错误的技术、错误的实施以及错误的政治。不过，ATM 的命运远好于 OSI。虽然它并没有在世界各地流行起来，但它仍然被广泛应用在合适的领域，包括诸如 DSL 那样的宽带接入线路以及电话网络内部的广域网链路。

ATM 是一种链路层，它的传输基于固定长度的信息**信元**（cell）。其名称中的“异步”意味着这些信元并不总是以连续的方式发送，这与 SONET 在同步线路上发送的方式不同。只有当出现需要运载的信息时，才发送信元。ATM 是一种面向连接的技术。每个信元在它的头部带有**虚电路**（virtual circuit）标识符，交换设备根据此标识符沿着连接建立的路径转发信元。

每个信元 53 字节长，由一个 48 字节的有效载荷以及一个 5 字节的头组成。利用这些小信元，ATM 可以为不同用户灵活划分物理层链路的带宽。这种能力对网络应用非常有用，例如，在同一条链路上发送语音信息和数据信息，不会因为出现很长的数据包而导致声音样本值的延迟变化过大。信元长度的与众不同选择（例如，相比之下，更自然的选择是 2 的幂次方）恰好说明 ATM 的设计是多么的政治化。选择有效载荷的长度为 48 字节正是解决欧洲和美国分歧的一个妥协，欧洲希望信元取 32 字节长，而美国方面则希望信元取 64 字节长。有关 ATM 的简要概述请参考（Siu 和 Jain，1995）。

为了在 ATM 网络上发送数据，需要将数据映射成一系列的信元。这个映射由 ATM 适配层完成，映射过程称为**分段**（segmentation）和**重组**（reassembly）。针对不同的服务定义了几个适配层，从周期性的声音样本到数据包数据。其中一个主要用于数据包数据的适配层是 **ATM 适应层 5**（AAL5，ATM Adaptation Layer 5）。

一个 AAL5 帧如图 3-27 所示。除了帧头，它还有一个帧尾，给出了帧的长度和用于错误检测的 4 字节 CRC。很自然，这里的 CRC 与 PPP 和诸如以太网那样的 IEEE 802 局域网使用的相同。（Wang 和 Crowcroft，1992）已表明该 CRC 强大到足以检测出非传统错误（诸

如信元重新排序）。除了有效载荷外，AAL5帧还需要被填充。填充的目的是使得帧的总长度是48字节的倍数，以便帧被均匀地划分成多个信元。这里不需要地址，因为每个信元携带的虚电路标识将引导它到达正确的目的地。

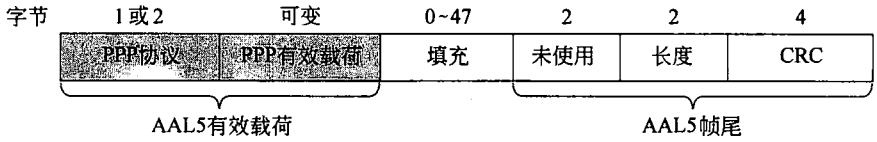


图3-27 运载PPP数据的AAL5帧

现在，我们已经简单描述了ATM，但仅涉及在ADSL的情况下PPP如何使用ATM。这项工作由另一个称为ATM上的PPP（PPPoA，PPP over ATM）标准完成。这个标准不是真正意义上的协议（所以没出现在图3-26中），但详细说明了PPP和AAL5帧如何工作的，该标准由RFC2364描述（Gross等，1998）。

只有PPP协议和有效载荷字段被放置在AAL5的有效载荷，如图3-27所示。协议字段告诉远端DSLAM有效载荷里包含的是一个IP数据包，或是另一种协议的数据包（比如LCP协议）。远端当然知道信元包含了PPP信息，因为ATM虚电路就是为这个目的而建立的。

在AAL5帧内，PPP的成帧功能并不是必需的，因为在这里起不到任何作用；ATM和AAL5早已提供了成帧的功能。更多的成帧考虑将毫无价值。同样，PPP的CRC也没有必要，因为AAL5已经包括了相同的CRC。这个错误检测机制补充了ADSL的物理层功能，ADSL物理层采用Reed-Solomon码来检测错误，同时还用1字节的CRC来检测遗漏的任何错误。该方案具有比在SONET上发送数据包更为复杂的错误恢复机制，因为ADSL是一种非常嘈杂的信道。

3.6 本章总结

数据链路层的任务是将物理层提供的原始比特流转换成由网络层使用的帧流。链路层为这样的帧流提供不同程度的可靠性，范围从无连接无确认的服务到可靠的面向连接服务不等。

链路层采用的成帧方法各种各样，包括字节计数、字节填充和比特填充。数据链路协议提供了差错控制机制来检测或纠正传输受损的帧，以及重新传输丢失的帧。为了防止快速发送方淹没慢速接收方，数据链路协议还提供了流量控制机制。滑动窗口机制被广泛用来以一种简单方式集成差错控制和流量控制两大机制。当窗口大小为1个数据包时，则协议是停-等式的。

纠错和检错码使用不同的数学技术把冗余信息添加到消息中。卷积码和里德所罗门码被广泛用于纠错，低密度校验码越来越受到欢迎。实际使用的检错码包括循环冗余校验和校验和两种。所有这些编码方法不仅可被应用在链路层，而且也可用在物理层和更高的层次。

我们考察了一系列协议，这些协议在更现实的假设下通过确认和重传，或者ARQ（自

动重复请求）机制为上层提供了一个可靠的链路层。从一个无错误的环境开始，即接收方可以处理传送给它的任何帧，我们引出了流量控制，然后是带有序号的差错控制和停-等式算法。然后，我们使用滑动窗口算法允许双向通信，并引出捎带确认的概念。最后给出两个协议把多个帧的传输管道化，以此来防止发送方被一个有着漫长传播延迟的链路所阻塞。接收方可以丢弃所有乱序的帧，或者为了获得更大的带宽效率而缓冲这些乱序帧，并且给发送方反馈否定确认。前一种策略是回退 n 协议，后一种策略是选择重传协议。

Internet 使用 PPP 作为点到点线路上的主要数据链路协议。PPP 协议提供了无连接的无确认服务，使用标志字节区分帧的边界，至于错误检测则采用 CRC。该协议通常被用在运载数据包的一系列链路上，包括广域网中的 SONET 链路和家庭 ADSL 链路。

习　题

1. 一个上层数据包被分成 10 个帧，每一帧有 80%的机会无损地到达目的地。如果数据链路协议没有提供错误控制，试问，该报文平均需要发送多少次才能完整地到达接收方？
2. 数据链路协议使用了下面的字符编码：

   A: 01000111；　B: 11100011；　FLAG: 01111110；　ESC: 11100000

   为了传输一个包含 4 个字符的帧：A B ESC FLAG，试问使用下面的成帧方法时所发送的比特序列（用二进制表达）是什么？

   （a）字节计数。

   （b）字节填充的标志字节。

   （c）比特填充的首尾标志字节。
3. 一个数据流中出现了这样的数据段：A B ESC C ESC FLAG FLAG D，假设采用本章介绍的字节填充算法，试问经过填充之后的输出是什么？
4. 试问字节填充算法的最大开销是多少？
5. 你的一个同学 Scrooge 指出每一帧的结束处用一个标志字节而下一帧的开始处又用另一个标志字节，这种做法非常浪费空间。用一个标志字节也可以完成同样的任务，这样可以节省下来一个字节。试问你同意这种观点吗？
6. 需要在数据链路层上被发送一个比特串：0111101111101111110。试问，经过比特填充之后实际被发送出去的是什么？
7. 试问在什么样的环境下，一个开环协议（比如海明码）有可能比本章通篇所讨论的反馈类协议更加适合？
8. 为了提供比单个奇偶位更强的可靠性，一种检错编码方案如下：用一个奇偶位来检查所有奇数序号的位，用另一个奇偶位来检查所有偶数序号的位。请问这种编码方案的海明距离是多少？
9. 假设使用海明码来传输 16 位的报文。试问，需要多少个校验位才能确保接收方能同时检测并纠正单个比特错误？对于报文 1101001100110101，试给出传输的比特模式。假设在海明码中使用了偶校验。
10. 接收方收到一个 12 位的海明码，其 16 进制值为 0xE4F。试问该码的原始值是多少？

假设至多发生了一位错。

11. 检测错误的一种方法是按 n 行、每行 k 位来传输数据，并且在每行和每列加上奇偶位。其中最右下角是一个校验其所在行和列的奇偶位。试问这种方案能检测出所有的 1 位错吗？2 位错误呢？3 位错误呢？请说明这种方案无法检测出某些 4 位错误。
12. 假设数据以块形式传输，每块大小 1000 比特。试问，在什么样的最大错误率下，错误检测和重传机制（每块 1 个校验位）比使用海明码更好？假设比特错误相互独立，并且在重传过程中不会发生比特错误。
13. 一个具有 n 行 k 列的块使用水平和垂直奇偶校验位进行差错检测。假设正好有 4 位由于传输错误被反转。请推导出该错误无法被检测出来的概率表达式。
14. 利用如图 3-7 所示的卷积码，当输入序列是 10101010（从左到右）并且内部状态初始化为全零时，试问，输出序列是什么？
15. 假设使用 Internet 校验和（4 位字）来发送一个消息 1001110010100011。试问校验和的值是什么？
16. x^7+x^5+1 被生成多项式 x^3+1 除，试问，所得余数是什么？
17. 使用本章介绍的标准 CRC 方法传输比特流 10011101。生成多项式为 x^3+1。试问实际传输的位串是什么？假设左边开始的第三个比特在传输过程中变反了。请说明这个错误可以在接收方被检测出来。给出一个该比特流传输错误的实例，使得接收方无法检测出该错误。
18. 发送一个长度为 1024 位的消息，其中包含 992 个数据位和 32 位 CRC 校验位。CRC 计算采用了 IEEE 802 标准，即 32 阶的 CRC 多项式。对于下面每种情况，说明在信息传输中出现的错误能否被接收方检测出来：
    （a）只有一位错误。
    （b）有 2 个孤立的一位错误。
    （c）有 18 个孤立的一位错误。
    （d）有 47 个孤立的一位错误。
    （e）有一个长度为 24 位的突发错误。
    （f）有一个长度为 35 位的突发错误。
19. 在 3.3.3 节讨论 ARQ 协议时，概述了一种场景，由于确认帧的丢失导致接收方接收了两个相同的帧。试问，如果不会出现丢帧（消息或者确认），接收方是否还有可能收到同一帧的多个副本？
20. 考虑一个具有 4 kbps 速率和 20 毫秒传输延迟的信道。试问帧的大小在什么范围内，停-等式协议才能获得至少 50%的效率？
21. 在协议 3 中，当发送方的计时器已经在运行时，它还有可能启动该计时器吗？如果可能，试问这种情况是如何发生的？如果不可能，试问为什么不可能？
22. 使用协议 5 在一条 3000 千米长的 T1 中继线上传输 64 字节的帧。如果信号的传播速度为 6 微秒/千米，试问序号应该有多少位？
23. 想象一个滑动窗口协议，它的序号占用的位数相当多，使得序号几乎永远不会回转。试问 4 个窗口边界和窗口大小之间必须满足什么样的关系？假设这里窗口的大小固定不变，并且发送方和接收方的窗口大小相同。

24. 在协议 5 中，如果 between 过程检查的条件是 a≤b≤c，而不是 a≤b<c，试问这对于协议的正确性和效率有影响吗？请解释你的答案。
25. 在协议 6 中，当一个数据帧到达时，需要检查它的序号是否不同于期望的序号，并且 no_nak 为真。如果这两个条件都成立，则发送一个 NAK；否则，启动辅助定时器。假定 else 子句被省略掉，试问这种改变会影响协议的正确性吗？
26. 假设将协议 6 中接近尾部的内含三条语句的 while 循环去掉，试问这样会影响协议的正确性吗？或者只是仅仅影响了协议的性能？请解释你的答案。
27. 地球到一个遥远行星的距离大约是 9×10^{10} 米。如果采用停-等式协议在一条 64 Mbps 的点到点链路上传输帧，试问信道的利用率是多少？假设帧的大小为 32 KB，光的速度是 3×10^{8} m/s。
28. 在前面的问题中，假设用滑动窗口协议来代替停-等式协议。试问多大的发送窗口才能使得链路利用率为 100%？发送方和接收方的协议处理时间可以忽略不计。
29. 在协议 6 中，frame_arrival 代码中有一部分是用来处理 NAK 的。如果入境帧是一个 NAK，并且另一个条件也满足，则这部分代码会被调用。请给出一个场景，在此场景下另一个条件非常关键。
30. 考虑在一条 1 Mbps 的完美线路（即无差错）上使用协议 6。帧的最大长度为 1000 位。每过 1 秒产生一个新数据包。超时间隔为 10 毫秒。如果取消特殊的确认计时器，那么就会发生不必要的超时事件。试问，平均报文要被传输多少次？
31. 在协议 6 中，MAX_SEQ=2^n-1。虽然这种情况显然是希望尽可能利用头部空间，但我们无法证明这个条件是基本的。试问，当 MAX_SEQ=4 时协议也能够正确工作吗？
32. 利用地球同步卫星在一个 1 Mbps 的信道上发送长度为 1000 位的帧，该信道的传播延迟为 270 毫秒。确认总是被捎带在数据帧中。帧头非常短，序号使用了 3 位。试问，在下面的协议中，可获得的最大信道利用率是多少？
    （a）停等式?
    （b）协议 5?
    （c）协议 6?
33. 在一个负载很重的 50 kbps 卫星信道上使用协议 6，数据帧包含长度为 40 位的头和长度为 3960 位的数据，试问浪费的带宽开销（帧头和重传）占多少比例？假设从地球到卫星的信号传播时间为 270 毫秒。ACK 帧永远不会发生。NAK 帧长 40 位。数据帧的错误率是 1%，NAK 帧的错误率忽略不计。序号占 8 位。
34. 考虑在一个无错的 64kbps 卫星信道上单向发送 512 字节长的数据帧，来自另一个方向反馈的确认帧非常短。对于窗口大小为 1、7、15 和 127 的情形，试问最大的吞吐量分别是多少？从地球到卫星的传播时间为 270 毫秒。
35. 在一条 100 千米长的线缆上运行 T1 数据速率。线缆的传播速度是真空中光速的 2/3。试问线缆中可以容纳多少位？
36. PPP 使用字节填充而不是比特填充，这样做的目的是防止有效载荷字段偶尔出现的标志字节造成的混乱。请至少给出一个理由说明 PPP 为什么这么做。
37. 试问，使用 PPP 发送一个 IP 数据包的最低开销是多少？如果只计算 PPP 自身引入的开销，而不计 IP 头开销，试问最大开销又是多少？

38. 在本地回路上使用 ADSL 协议栈来发送一个长为 100 个字节的 IP 数据包。试问一共发出去多少个 ATM 信元？请简要描述这些信元的内容。

39. 本实验练习的目标是用本章描述的标准 CRC 算法实现一个错误检测机制。编写两个程序：generator 和 verifier。generator 程序从标准输入读取一行 ASCII 文本，该文本包含由 0 和 1 组成的 n 位消息。第二行是个 k 位多项式，也是以 ASCII 码表示。程序输出到标准输出设备上的是一行 ASCII 码，由 n+k 个 0 和 1 组成，表示被发送的消息。然后，它输出多项式，就像它输入的那样。verifier 程序读取 generator 程序的输出，并输出一条消息指示正确与否。最后，再写一个程序 alter，它根据参数（从最左边开始 1 的比特数）反转第一行中的比特 1，但正确复制两行中的其余部分。通过键入：

```
generator <file | verifier
```

你应该能看到正确的消息，但键入：

```
generator <file | alter arg | verifier
```

你只能得到错误的消息。

第 4 章　介质访问控制子层

网络链路可以分成两大类：使用点到点连接和使用广播信道。我们在第 2 章学习了点到点链路，本章将讨论广播网络和相应的协议。

在任何一个广播网络中，关键的问题是当多方竞争信道的使用权时如何确定谁可以使用信道。为了把这个问题表述得更加清晰，我们用一个电话会议的场景来模拟说明：现在有 6 个人分别守在 6 部不同的电话机旁，这些电话相互之间都有连接，所以每个人都可以听到其他人说话，也可以对其他人讲话。当一个人停止说话时，很可能马上有两个或者更多个人开始说话，从而导致交流的一片混乱。而在面对面坐着的会议上，这种混乱局面可以通过外部途径得到解决，比如，让与会者通过举手的方式请求获得发言权。当只有一条信道可供使用时，确定下一个使用者的确非常困难。解决这个问题的许多协议已众所周知。这些协议组成了本章的内容。在有些文献中，广播信道有时候也称为**多路访问信道**（multiaccess channel）或者**随机访问信道**（random access channel）。

用来确定多路访问信道下一个使用者的协议属于数据链路层的一个子层，该层称为**介质访问控制**（MAC，Medium Access Control）子层。在 LAN 中，MAC 子层显得尤为重要，特别是在无线局域网中，因为无线本质上就是广播信道。相反，WAN 则使用点到点链路，当然，卫星网络除外。因为多路访问信道和 LAN 如此紧密相关，所以，在本章我们将从总体上讨论 LAN，其中也包括一些从严格意义上讲并不属于 MAC 子层的内容，但是总的主题还是关于信道控制。

技术上，MAC 子层位于数据链路层底部，所以，逻辑上我们应该在第 3 章讨论所有点到点协议之前学习 MAC 子层。然而，对于大多数人来说，在很好地理解了只有两方参与的协议之后，再来理解涉及多方协同的协议要容易得多。正是基于这样的原因，我们才稍微偏离了本书自底向上的严格顺序。

4.1　信道分配问题

本章的主题是如何在竞争用户之间分配单个广播信道。信道可以是一个地理区域内的一部分无线频谱，也可以是连接着多个节点的单根电缆或者光纤，这都无关紧要。在这两种情况下，信道把每个用户与所有其他用户连接在一起，任何正在使用信道的用户和其他也想使用该信道的用户会相互干扰。

我们首先考察静态分配方案在突发流量情况下的缺点；然后，我们给出一些关键假设，这些假设在后面讨论动态分配方案的模型时要用到。

4.1.1　静态信道分配

在多个竞争用户之间分配单个信道的传统做法是把信道容量拆开分给多个用户使用，

具体方法可以采用我们在2.5节中讨论的某种多路复用技术，比如FDM（频分多路复用），就像多个竞争用户多路复用电话中继线一样。如果总共有N个用户，则整个带宽被分成N等份，每个用户获得一份。由于每个用户都有各自专用的频段，所以用户之间不会发生干扰。当用户数量比较少且固定不变时，如果每个用户都有稳定的流量或者负载繁重，这种信道分割是一种简单而有效的分配机制。FM无线电广播就是一个无线信道的多路复用例子。每个电台获得FM波段一部分的使用权，大部分时间用该波段广播自己的信号。

然而，当发送方的数量非常多而且经常不断变化，或者流量呈现突发性，FDM就存在一些问题。如果整个频谱被切割成N份，并且当前只有很少的用户（比N少得多）需要进行通信，那么大量宝贵的频谱将被浪费掉。但是，如果希望进行通信的用户数超过了N个，则有些用户将因带宽不够而遭到拒绝；即使有些已经被分配了频段的用户什么都不发送或者什么都不接收，也无法将它们的频段让给其他需要的用户使用。

然而，即使我们假设用户数量能够维持在N个固定不变，将单个信道划分成多个静态子信道的做法本质上也是低效的。基本问题在于，当有些用户停止通信时，他们的带宽实际上就被白白浪费了。他们自己不使用这些带宽，其他用户也不允许使用。静态分配方案很难适应大多数计算机系统，在计算机系统中数据流量表现出极端的突发性，通常峰值流量与平均流量之比能达到1000∶1。因此，大多数信道在多数时候是空闲的。

静态FDM如此差的性能通过一个简单的排队理论计算很容易看得更清楚。我们考虑在一个容量为C bps的信道上发送一帧所需要的平均时延T。假设，随机到达帧的平均到达率为λ帧/秒，帧的长度可变，其均值为每帧1/μ位。利用这些参数，可以算出信道的服务率为μC帧/秒。标准排队理论的结果是：

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

（很好奇的是这个结果是一个M/M/1排队。它要求帧到达的时间差和帧的长度符合遵循指数分布的随机过程，或者等价于泊松过程的结果。）

在我们的例子中，如果C为100 Mbps，平均帧长1/μ等于10 000位，帧的到达率为5000帧/秒，则T=200微秒。请注意，如果我们忽略排队延迟，只是问“在一个100 Mbps的网络上发送一个10 000位的帧需要多长时间”，那么我们将得到（不正确的）答案100微秒。这样的结果只有当不存在信道竞争时才成立。

现在我们将单个信道分成N个独立的子信道，每个子信道的容量为C/N bps。现在，每个子信道的平均到达率变成λ/N。重新计算T，我们得到：

$$T_N=\frac{1}{\mu(C/N)-(\lambda/N)}=\frac{N}{\mu C-\lambda}=NT \qquad (4\text{-}1)$$

如果所有帧都能很神奇地排在一个大的中心队列，那么划分信道后单个信道的平均延迟比不分的情况差N倍。同样的结论适用在银行。在一家银行的大堂摆满了ATM机，设立单个到达所有这些ATM机的队列比为每台ATM机器设立一个单独队列的效果要好。

适用于FDM的结论同样也适用于其他静态划分信道的方法。如果我们打算使用时分多路复用（TDM）技术为每个用户固定分配每N个时间槽，当一个用户没有使用分配给他的时间槽，那么该时间槽就会空闲。如果我们从物理上将网络分割开，则存在同样的问题。继续引用前面的例子，如果我们用10个10 Mbps的网络来代替一个100 Mbps的网络，并

且为每个用户固定分配其中的一个网络，则平均延迟将从 200 微秒跳跃到 2 毫秒。

既然所有的传统静态信道分配方法都不适应突发性的流量，我们现在就来研究动态的信道分配方法。

4.1.2　动态信道分配的假设

本章将介绍许多种信道分配方案，在讨论第一种方案之前，有必要认真地形式化信道分配问题。在这方面做的所有工作都是以下面 5 个关键假设为基础的。

（1）**流量独立**（independent traffic）。该模型是由 N 个独立的站（比如计算机、电话）组成的，每个站都有一个程序或者用户产生要传输的帧。在长度为 Δt 的间隔内，期望产生的帧数是 λΔt，这里 λ 为常数（新帧的到达率）。一旦生成出一帧，则站就被阻塞，直到该帧被成功地发送出去。

（2）**单信道**（Single Channel）。所有的通信都用这一个信道。所有的站可以在该信道上传输数据，也可以从该信道接收数据。所有站的能力都相同，尽管协议可能为站分配不同的角色（比如，优先级）。

（3）**冲突可观察**（observable Collision）。如果两帧同时传输，则它们在时间上就重叠，由此产生的信号是混乱的，这种情况称为**冲突**（collision）。所有的站都能够检测到冲突事件的发生。冲突的帧必须在以后再次被发送。除了因冲突而产生错误外，不会再有其他的错误。

（4）**时间连续或分槽**（Continuous or slotted time）。时间可以假设是连续的，即在任何时刻都可以开始传输帧。另一种选择是把时间分槽或者分成离散的间隔（称为时间槽）。帧的传输只能从某个时间槽的起始点开始。一个时间槽可能包含 0、1 或者多个帧，分别对应于空闲的时间槽、一次成功发送，或者一次冲突。

（5）**载波侦听或不听**（Carrier Sense or no carrier sense）。有了载波侦听的假设，一个站在试图用信道之前就能知道该信道当前是否正被使用。如果信道侦听结果是忙，则没有一个站会再去试图使用该信道。如果没有载波侦听，站就无法在使用信道之前侦听信道，它们只能盲目地传输，以后再判断这次传输是否成功。

下面依次对这些假设进行讨论。第一个假设意味着帧的到达是独立的，无论是跨越多个站还是某个特定的站，帧的产生不可预测但以恒定的速率产生帧。其实，正如众所周知的那样，针对网络流量的这种假设并不是个很好的模型，因为数据包在一个时间尺度范围内的到达呈现突发性（Paxson 和 Floyd，1995；Leland 等，1994）。尽管如此，随着被频繁地应用，泊松模型因其在数学上易于处理而成为一个很有用的工具。它们能帮助我们分析协议，理解协议在大致操作范围内的性能如何变化，以及如何与其他协议比较。

单信道的假设是该模型的核心。除了这个信道外，没有任何外部途径可以通信。这些站不可能举起手来请求老师准许发言，因此我们必须拿出更好的解决方案。

其余的三个假设依赖于该系统的工程。当我们考察一个特定的协议，我们会说这些假设是成立的。

冲突假设是最基本的。当站发送时需要一些方法来检测是否发生了冲突，由此决定重传帧而不是任由那些帧被丢失。对于有线信道，节点的硬件可设计成一边发送一边检测冲

突；然后，如果发生了冲突，该站可提前终止传输，以免浪费信道容量。这种检测对于无线信道很难做到，所以冲突的检测通常被推迟到确信没有出现预期的确认帧这样一个既成事实之后。同时，卷入冲突的一些帧也有可能最终被成功接收，这主要取决于具体的信号强度和接收硬件的能力。不过，这种情况很少见，因此我们假设发生冲突后所有涉及的帧都被丢失。我们还将看到一些协议被设计成旨在防止冲突的发生。

对时间给出两种不同假设的理由在于时间槽可用来改善协议性能。然而，这要求所有站遵循一个主时钟或者它们的行动与其他站同步，才能将时间分为离散的间隔。因此，它并不总是可用的。我们对这两类时间都进行了讨论和分析。对于一个给定的系统，它只能支持一种时间假设。

类似地，一个网络可能具有载波侦听功能，也可能没有载波侦听功能。有线网络通常具有载波侦听功能。然而，无线网络不能有效地使用它，因为并不是每个站都在其他各站的无线电广播范围内。同样，在站不能直接和其他各站通信的一些其他设置中，载波侦听也不可用，例如线缆调制解调器，站必须通过线缆头端才能通信。注意，这个意义上的载波（carrier）一词是指信道上的信号，与公共运营商（例如，电话公司）没有任何关系，说起公共运营商可以追溯到小马快递的日子。

为了避免任何误解，值得注意的是没有多路访问协议能保证可靠传送。即使没有发生冲突，也有这样或者那样的原因使得接收器错误地复制了帧的某些部分。因此，要由链路层的其他部分或比链路层更高的层次来提供数据传输的可靠性。

4.2 多路访问协议

分配一个多路访问信道的算法有许多。在下面这一节中，我们将学习一些比较有意思的算法，并给出在实际中如何应用它们的实例。

4.2.1 ALOHA

我们的第一个 MAC 故事开始于 20 世纪 70 年代“原始的”夏威夷。在这种情况下，“原始的”（pristine）一词可以解释为“没有一个可运行的电话系统”。这并没有使生活在这里的夏威夷大学研究员 Norman Abramson 和他的同事感觉愉快，他们正试图把偏远岛屿上的用户连接到檀香山的主计算机。把自己的电缆穿过太平洋海底显然不是个办法，因此他们寻找着不同的解决方案。

他们找到了一种用于短程无线电通信的方法。所有的用户终端共享同一个上行频率给中央计算机发送帧。其中包括了一个简单而巧妙的方法来解决信道分配问题。自此以后，他们的工作得到了许多研究者的扩充（Schwartz 和 Abramson，2009）。虽然 Abramson 的工作（称为 ALOHA 系统）使用了地面无线电广播，其基本思路同样适用于任何非协调用户竞争使用单个共享信道的系统。

我们在这里将要讨论两个版本的 ALOHA：纯 ALOHA 和分槽 ALOHA。它们的区别在于时间是连续的，那就是纯粹版的 ALOHA；或者时间分成离散槽，所有帧都必须同步到

时间槽中。

纯 ALOHA

ALOHA 系统的基本思想非常简单：当用户有数据需要发送时就传输。当然，这样做可能会产生冲突，冲突的帧将被损坏。发送方需要某种途径来发现是否发生了冲突。在 ALOHA 系统中，每个站在给中央计算机发送帧之后，该计算机把该帧重新广播给所有站。因此，那个发送站可以侦听来自集线器的广播，以此确定它的帧是否发送成功。在其他系统中，比如有线局域网中，发送方在发送的同时能侦听到冲突的发生。

如果帧被损坏了，则发送方要等待一段随机时间，然后再次发送该帧。等待的时间必须是随机的，否则同样的帧会一次又一次地冲突，因为冲突帧被重发的节奏完全一致。如果系统中多个用户共享同一个信道的方法会导致冲突，则这样的系统称为**竞争**（contention）系统。

图 4-1 给出了一个 ALOHA 系统中帧的框架结构。我们使所有的帧具有同样的长度，因为对于 ALOHA 系统，采用统一长度的帧比长度可变的帧更能达到最大的吞吐量。

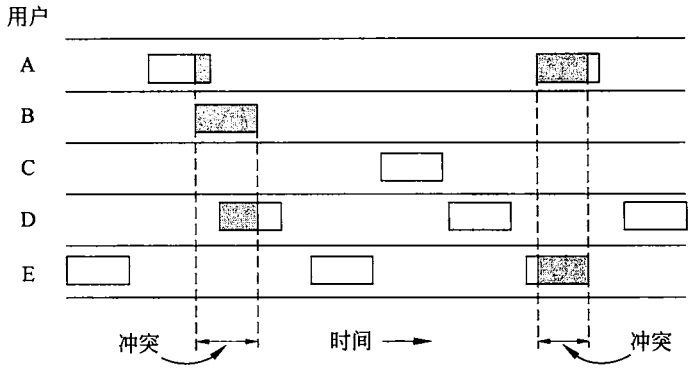


图 4-1　在纯 ALOHA 中帧的发送次序是任意的

无论何时，只要两个帧在相同时间试图占用信道，冲突就会发生（如图 4-1 所示），并且两帧都会被破坏。如果新帧的第一位与几乎快传完的前一帧的最后一位重叠，则这两帧都将被彻底毁坏（即具有不正确的校验和），稍后都必须被重传。校验和不可能（也不应该）区分出是完全损坏还是局部差错。坏了就是坏了。

一个有趣的问题是：ALOHA 信道的效率怎么样？换句话说，在这样混乱的情况下，能够侥幸逃脱冲突而被传输出去的帧占多大比例？我们首先考虑这样的情形：有无穷多个交互用户坐在他们的终端（站）前面。每个用户总是处于两种状态中的一种：敲键或等待。刚开始时，所有的用户都处于敲键的状态。当输入完一行之后，用户停止敲键，开始等待应答；然后站在共享信道上给中央计算机发送一个包含了该行字符的帧，并且检查信道看是否传输成功。如果传输成功，则用户会看到应答，回到敲键状态继续输入。如果不成功，则在站一次次重传该帧时用户继续等待，直到该帧被成功发送出去为止。

我们用“帧时”（frame time）来表示传输一个标准的、固定长度的帧所需要的时间（即帧的长度除以比特率）。现在我们假定站产生的新帧可以模型化为一个平均每“帧时”产生 N 个帧的泊松分布（假设存在无穷多个用户是必要的，因为这样可以确保 N 不会随着用户变成阻塞状态而下降）。如果 $N > 1$，则用户群生成帧的速率大于信道的处理速度，因此，

几乎每个帧都要经受冲突。为了取得合理的吞吐量，我们应该期望 0<N<1。

除了新生成的帧以外，每个站还会产生由于先前遭受冲突而重传的那些帧。我们进一步假设在每个“帧时”中，老帧和新帧合起来也符合泊松分布，每帧时的平均帧数为 G。显然，G≥N。在负载较低的情况下（即 N<0），冲突很少发生，因此重传也很少，于是 G<N。在负载较高时，将会有很多冲突，所以 G>N。在所有这些负载的情况下，吞吐量 S 就是负载 G 乘以成功传输的概率 P_0——$S=GP_0$，其中 P_0 是一帧没有遭受冲突的概率。

如果从一帧被发送出去开始算起，在一个“帧时”内没有发出其他的帧，则这一帧不会遭到冲突，如图 4-2 所示。在什么样的条件下，图中的阴影帧将毫无损坏地到达目的地？假设发送一帧所需的时间为 t。如果其他用户在 t_0～t_0+t 之间生成了一帧，则该帧的结尾部分将与阴影帧的开始部分发生冲突。实际上，阴影帧的命运在它的第一位被发出去之前就已经注定了。但是，由于在纯 ALOHA 中，站在传输之前并不去侦听信道，所以，它无法知道是否有其他的帧已经在使用信道上了。类似地，在 t_0+t～t_0+2t 之间开始发送的任何其他帧都会和阴影帧的结尾部分冲突。

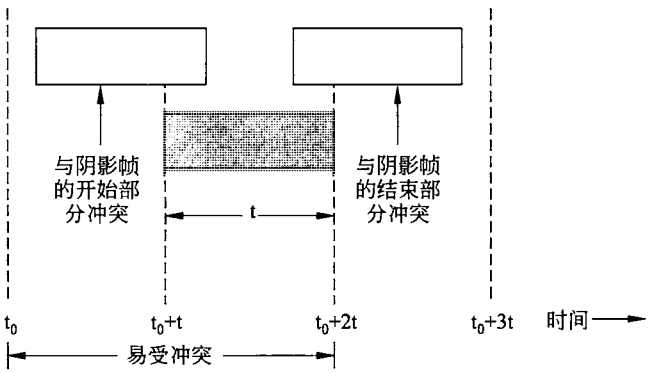


图 4-2 阴影帧的易受冲突周期

在给定的一个“帧时”内希望有 G 帧，但生成 k 帧的概率服从泊松分布：

$$Pr[k]=\frac{G^k e^{-G}}{k!} \qquad (4\text{-}2)$$

所以，生成零帧的概率为 e^{-G}。在两个“帧时”长的间隔中，生成帧的平均数是 2G。因此，在整个易受冲突期中，不发送帧的概率是 $P_0=e^{-2G}$。利用 $S=GP_0$，则可以得到：

$$S=Ge^{-2G}$$

流量与吞吐量之间的关系如图 4-3 所示。最大的吞吐量出现在当 G = 0.5 时，S = 1/2e，大约等于 0.184。换句话说，我们可以希望的最好信道利用率为 18%。这个结果并不令人鼓舞，但是对于这种任何人都可以随意发送的传输方式，要想达到 100%的成功率几乎是不可能的。

分槽 ALOHA

ALOHA 出现不久，Roberts 发表了一种能将 ALOHA 系统的容量增加一倍的方法（Roberts，1972）。他的建议是将时间分成离散的间隔，这种时间间隔称为**时间槽**（slot），每个时间槽对应于一帧。这种方法要求用户遵守统一的时间槽边界。取得同步时间的一种办法是由一个特殊的站在每个间隔起始时发出一个脉冲信号，就好像一个时钟一样。

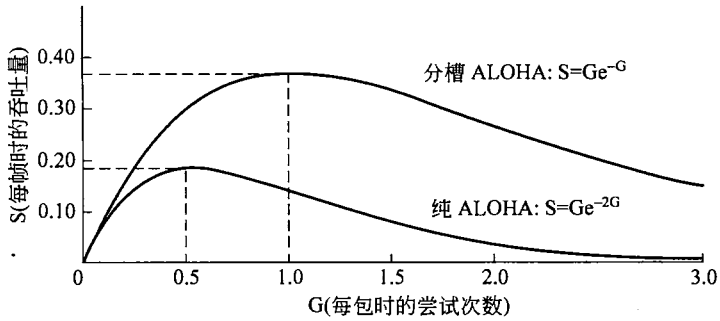


图 4-3　ALOHA 系统的吞吐量与负载的关系

Roberts 方法称为分槽 ALOHA（slotted ALOHA）。与纯 ALOHA 不同的是，在分槽 ALOHA 中，站不允许用户每次敲入回车键就立即发送帧。相反，它必须要等到下一个时槽的开始时刻。因此，连续的纯 ALOHA 变成了离散的 ALOHA，这将易受冲突期减小了一半。为了解清楚这点，请看图 4-3，并且想象现在可能发生的冲突。在测试帧所在的同一个时间槽中没有其他流量的概率是 e^{-G}，于是可以得到：

$$S=Ge^{-G} \qquad (4\text{-}3)$$

正如你可以从图 4-3 中看到的那样，分槽 ALOHA 的尖峰在 G=1 处，此时吞吐量为 S=1/e，大约等于 0.368，是纯 ALOHA 的两倍。如果系统运行在 G=1 处，则空时间槽的概率为 0.368（从等式（4-2）可以得出）。使用分槽 ALOHA，我们期望的最好结果是 37%为空时间槽、37%为成功，剩下 26%为冲突。如果在更高的 G 值上运行，则空时间槽数会降低，但冲突时间槽数会呈指数增长。为了看出冲突时间槽数是如何随着 G 的变化而快速增长，请考虑一个测试帧的传输过程。该测试帧能够避免一次冲突的概率是 e^{-G}，即所有其他用户在该时间槽中静止不发帧的概率。于是，冲突的概率为 $1-e^{-G}$。需要 k 次尝试才能成功传输的概率（即 k-1 次冲突之后才有一次成功的概率）为：

$$P_k=e^{-G}(1-e^{-G})^{k-1}$$

于是，每帧传输次数期望 E，即终端键入一行的概率为：

$$E=\sum_{k=1}^{\infty}kP_k=\sum_{k=1}^{\infty}ke^{-G}(1-e^{-G})^{k-1}=e^{G}$$

所以，E 随 G 呈指数增长的结果是信道负载的微小增长也会极大地降低信道的性能。

分槽 ALOHA 引起关注的一个原因在于它的重要性在刚开始时没有得到重视。它是在 20 世纪 70 年代被设计出来，曾经用在一些实验系统中，之后差不多就被大家遗忘掉了。当通过有线电视电缆访问 Internet 技术被发明出来时，立即出现了一个问题，那就是如何在多个竞争用户之间分配一条共享信道，于是分槽 ALOHA 被人们从遗忘的角落中翻了出来，从而拯救了世界。后来，多个 RFID 标签和同一个 RFID 读写器通信时也出现了同样的问题，只是表现形式不同。又是分槽 ALOHA，和其他想法结合起来再次成了救世主。经常会出现这样的情况：一些非常完善有效的协议由于政策上的原因（比如某个大公司希望每个人都按照它的方式来行事），或者因为不断变化的技术发展趋势而被弃置不用。然后，多年以后，一些聪明人就会发现某个长期被束之高阁的协议恰好可以解决他们的当前问题。出于这样的原因，在本章我们将学习一些非常优秀的协议，尽管它们目前尚未得到广泛应

用，只要有足够的网络设计师了解它们，在将来的应用中它们很有可能会发挥作用。当然，我们也会学习许多当前正在使用的协议。

4.2.2 载波侦听多路访问协议

利用分槽 ALOHA，可以达到的最佳信道利用率是 1/e。这么低的结果并不令人惊奇，因为每个站都可以随意地发送数据，它并不知道其他站是否也在发送数据。所以，频繁地发生冲突是难免的。然而，在局域网中，站是完全有可能检测到其他站当前在做什么，然后再根据情况调整自己的行为。这些网络可以获得比 1/e 好得多的利用率。在这一小节中，我们将讨论一些提高性能的协议。

如果在一个协议中，站监听是否存在载波（即是否有传输），并据此采取相应的动作，则这样的协议称为**载波侦听协议**（carrier sense protocol）。很久以前许多这类协议就被提了出来，而且都已经被详细地分析过了。例如，参考（Kleinrock 和 Tobagi，1975）。下面我们看几个载波侦听协议。

坚持和非坚持 CSMA

我们将要学习的第一个载波侦听协议称为 1-坚持载波检测多路访问（CSMA，Carrier Sense Multiple Access）。这是最简单的 CSMA 方案。当一个站有数据要发送时，它首先侦听信道，确定当时是否有其他站正在传输数据；如果信道空闲，它就发送数据。否则，如果信道忙，该站等待直至信道变成空闲；然后，站发送一帧。如果发生冲突，该站等待一段随机的时间，然后再从头开始上述过程。这样的协议之所以称为 1-坚持，是因为当站发现信道空闲时，它传输数据的概率为 1。

除了罕见的多个站同时发送情况外，你或许期望这个方案能避免冲突，但实际上它不能。如果两个站在第三个站的发送过程中准备好了拟发送的数据，它们俩都会礼貌地等待，直到当前的传输结束；然后双方将确定同时开始传输，这种行为显然会导致冲突发生。如果它们不那么急躁，冲突的可能性将会少得多。

更微妙的是，信号的传播延迟对冲突有着重大影响。这里存在一个时机，在某个站开始发送后，另一个站也刚好做好发送的准备并侦听信道。如果第一个站的信号没能到达第二个站，后者侦听到信道是空闲的，因而也开始发送，由此导致双方的冲突。可见这个机会取决于信道上适合的帧数，或信道的**带宽延迟积**（bandwidth-delay product）。如果信道只够容纳帧的很小一部分（这种情况符合大多数局域网的环境），由于信号的传播延迟小，冲突发生的机会就小。带宽延迟积越大，这种影响就变得愈加重要，因而协议的性能越差。

即便如此，上述协议的性能也比纯 ALOHA 协议要好得多，因为这两个站都非常礼貌，不会去打扰第三个站的帧。确切地，同样的结论也适用于分槽 ALOHA。

第二个载波侦听协议是非坚持 CSMA（nonpersistent CSMA）。在这个协议中，站在试图发送数据之前要理智得多，不像前一个协议那样贪婪。和前一个协议一样，站在发送数据之前要先侦听信道。如果没有其他站在发送数据，则该站自己开始发送数据。然而，如果信道当前正在使用中，则该站并不持续对信道进行监听，以便传输结束后立即抓住机会发送数据。相反，它会等待一段随机时间，然后重复上述算法。因此，该算法将会导致更

好的信道利用率，但是比起 1–坚持 CSMA，也带来了更大的延迟。

最后一个协议是 p–坚持 CSMA（p-persistent CSMA）。它适用于分时间槽的信道，其工作方式如下所述。当一个站准备好要发送的数据时，它就侦听信道。如果信道是空闲的，则它按照概率 p 发送数据；而以概率 q=1–p，将此次发送推迟到下一个时间槽。如果下一个时间槽信道也是空闲的，则它还是以概率 p 发送数据，或者以概率 q 再次推迟发送。这个过程一直重复，直到帧被发送出去，或者另一个站开始发送数据。如果发生了后一种情况，那么这个极不走运的站按照冲突发生时的处理过程一样（即等待一段随机的时间，然后再重新开始）。如果该站刚开始时就侦听到信道为忙，则它等到下一个时间槽，然后再应用上面的算法。IEEE 802.11 对 p 坚持 CSMA 作了细微的改良，我们将在 4.4 节讨论它。

图 4-4 显示了上述 3 个协议以及纯 ALOHA 和分槽 ALOHA 的可计算吞吐量和负载之间的关系。

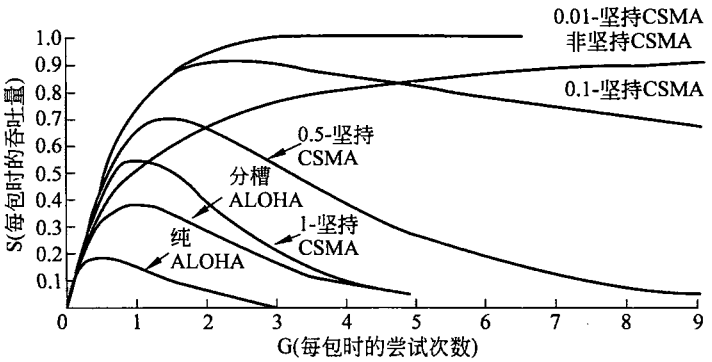


图 4-4　不同随机访问协议的信道使用率与负载比较

带冲突检测的 CSMA

坚持和非坚持 CSMA 协议无疑是对 ALOHA 的改进，因为这些协议都确保了信道忙时，所有的站都不再传送数据。然而，如果两个站侦听到信道为空，并且同时开始传输，则它们的信号仍然会产生冲突。因此，另一个改进是每个站快速检测到发生冲突后立即停止传输帧（而不是继续完成传输），因为这些帧已经无可挽回地成为乱码。这种策略可以节省时间和带宽。

这种协议称为**带冲突检测的 CSMA**（CSMA/CD，CSMA with Collision Detection）。它是经典以太局域网的基础，所以，值得我们专门花一点时间来详细地介绍它。重要的是要认识到，冲突检测是一个模拟过程。站的硬件在传输时必须侦听信道。如果它读回的信号不同于它放到信道上的信号，则它就知道发生了碰撞。言外之意是接收信号相比发射信号不能太微弱（这对无线来说很难做到，因为接收信号可能 1 000 000 倍弱于发射信号），并且必须选择能被检测到冲突的调制解调技术（比如，两个 0 伏信号的冲突很可能无法检测到）。

如同许多其他 LAN 协议一样，CSMA/CD 也使用了图 4-5 所示的概念模型。在标记为 t_0 点，一个站已经完成了帧的传送，其他需要发送帧的站现在可以试图发送了。如果有两个或者多个站同时进行传送，冲突就会发生。如果一个站检测到冲突，它立即中止自己的传送，等待一段随机时间，然后再重新尝试传送（假定在此期间没有其他站已经开始传送）。

因此，我们的 CSMA/CD 模型将由交替出现的竞争期、传输期，以及当所有站都静止的空闲期（比如没有传输任务）组成。

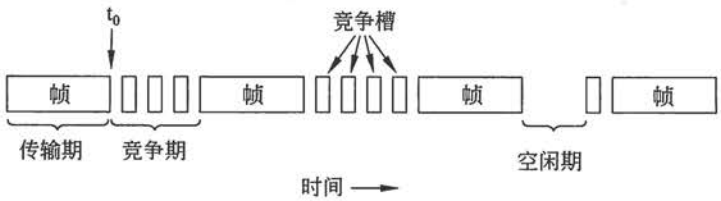


图 4-5 CSMA/CD 可能处于竞争、传输或空闲状态

现在我们来看竞争算法的细节。假定两个站同时在 t_0 时刻开始传送数据。它们需要多长时间才能意识到发生了冲突呢？该问题的答案对于确定竞争周期的长度至关重要，进而会影响到延迟和吞吐量。

检测冲突的最小时间恰好是将信号从一个站传播到另一个站所需要的时间。基于这个信息，你可能会认为一个站在开始发送后，经过一段特定的时间还未监听到冲突，它就可以确定自己“抓住”（seize）了电缆。这里“抓住”的意思是指所有其他站都知道该站正在传送数据，所以不会干扰它；而那段特定的时间表示整条线缆的传播时间。这个结论是错误的。

考虑下面给出的最坏情形。假设两个相距最远的站传播信号所需要的时间为 τ。在 t_0 时刻，一个站开始传送数据。在 $t_0+\tau-\varepsilon$ 时刻，也就是在信号到达最远那个站之前的一刹那，那个站也开始传输。当然，它几乎立刻就检测到冲突，并且立即停止了传输，但由这次冲突引起的微小噪声尖峰要到 $2\tau-\varepsilon$ 时刻才能回到原来的那个发送站。换句话说，在最差的情况下，只有当一个站传输了 2τ 之后还没有监听到冲突，它才可以确保自己经抓住了信道。

有了这种认识，我们可以把 CSMA/CD 竞争看成是一个分槽 ALOHA 系统，时间槽宽度为 2τ。在 1 千米长的同轴电缆上，$\tau < 5$ 微秒。CSMA/CD 和分槽 ALOHA 的区别在于，只有一个站能用来传输的时间槽（即信道被抓住了）后面紧跟的那些时间槽被用来传输该帧的其余部分。如果帧时相比传播时间长很多，这种差异将能大大提高协议的性能。

4.2.3 无冲突协议

在 CSMA/CD 中，一旦站已经确定无疑地抓住了信道，冲突就不会再发生；尽管如此，在竞争期中冲突仍有可能发生。这些冲突严重地影响了系统的性能，特别是当带宽延迟积很大，比如电缆很长（即 τ 很大）而帧的长度又很短时。冲突不仅降低了带宽，而且使得发送一个帧的时间变得动荡不定，这样就无法很好地适应实时流量，比如 IP 语音。而且 CSMA/CD 也不是普遍适用的。

在本节，我们将介绍一些协议，它们以根本不可能产生冲突的方式解决了信道竞争问题，即使在竞争期中也不会发生冲突。大多数这样的协议目前并没有被用在主流系统中；但是，在一个快速变化的领域，拥有一些具有优异特性可适用于未来系统的协议，通常是一件好事。

在接下来描述的协议中，我们假定共有 N 个站，每个站都有唯一的地址，地址范围从

0 到 N–1。有些站在一部分时间中可能是不活跃的，不过这无关紧要。我们还假定传播延迟可以忽略不计。但是，基本问题仍然存在：在一次成功的传输之后哪个站将获得信道？我们继续使用带有离散竞争时间槽的图 4-5 模型。

位图协议

我们要介绍的第一个无冲突协议采用了**基本位图法**（basic bitmap method），每个竞争期正好包含 N 个槽。如果 0 号站有一帧数据要发送，则它在第 0 个槽中传送 1 位。在这个槽中，不允许其他站发送。不管 0 号站做了什么，1 号站有机会在 1 号槽中传送 1 位，但是只有当它有帧在排队等待时才这样做。一般地，j 号站通过在 j 号槽中插入 1 位来声明自己有帧要发送。当所有 N 个槽都经过后，每个站都知道了哪些站希望传送数据。这时候，它们便按照数字顺序开始传送数据了，如图 4-6 所示。

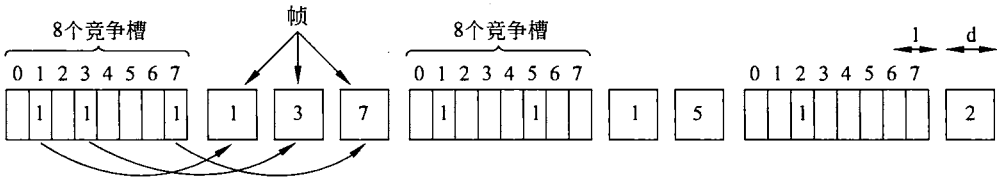


图 4-6　基本位图协议

由于每个站都同意下一个是谁传输，所以永远也不会发生冲突。当最后一个就绪站传送完它的帧后，这是所有站都很容易检测到的事件，于是，另一个 N 位竞争期又开始了。如果一个站在它对应的位槽刚刚经过就准备好了要发送的数据，那它就非常不幸，只能保持沉默；直到每个站都获得了发送数据的机会，新的位图再次到来，它才能通过位槽表明自己有传输的意愿。

像这样在实际传送数据之前先广播自己有发送数据愿望的协议，称为**预留协议**（reservation protocol）。现在我们简要分析这个协议的性能。为了简便起见，我们将用竞争位槽作为单位来计量时间，假定数据帧由 d 个时间单位构成。

在负载很低的情况下，数据帧非常少，位图简单地一次又一次地重复出现。我们从序号较低的站，比如 0 号站或者 1 号站的角度来考虑。典型情况下，当它已经做好发送数据的准备时，“当前”槽将处于位图中间的某个地方。平均而言，该站必须等待完成当前扫描的 N/2 个槽，再等待完成下一次扫描的另外 N 个槽，然后才能开始传输数据。

从高序号的站来看则情形会好很多。一般地，它只需等待半个扫描周期（N/2 个位槽）就可以开始传输数据了。高序号的站往往不必等待到下一次扫描。由于低序号的站必须等待平均 1.5N 个槽，而高序号的站必须等待平均 0.5N 个槽，因此对所有站而言，要平均等待 N 个槽。

在低负载情况下的信道效率很容易计算。每一帧的额外开销是 N 位，数据长度为 d 位，于是信道利用率为 d/(N+d)。

在高负载的情况下，若所有站在任何时候都有数据要发送，则 N 位竞争期被分摊到 N 个帧上，因此，每一帧的额外开销只有 1 位，或者，信道利用率为 d/(d+1)。一帧的平均延迟等于它在站内的排队时间，加上它到达队列头部之后另外的(N–1)d+N 时间。它要等待的这个间隔是所有其他站轮流发送一帧以及一个位图的时间。

令牌传递

位图协议的实质是让每个站以预定义的顺序轮流发送一帧。完成同样事情的另一种方法是通过传递一个称为**令牌**（token）的短消息，该令牌同样也是以预定义的顺序从一个站传到下一个站。令牌代表了发送权限。如果站有个等待传输的帧队列，当它接收到令牌就可以发送帧，然后再把令牌传递到下一站。如果它没有排队的帧要传，则它只是简单地把令牌传递下去。

在**令牌环**（token ring）协议中，网络的拓扑结构被用来定义站的发送顺序。所有站连接成一个单环结构，一个站依次连接到下一个站。因此令牌传递到下一站只是单纯地从一个方向上接收令牌和在另一个方向上发送令牌，如图4-7所示。帧也按令牌方向传输。这样，它们将绕着环循环，到达任何一个目标站。然而，为了阻止帧陷入无限循环（像令牌一样），一些站必须将它们从环上取下来。这个站或许是最初发送帧的原始站，在帧经历了一个完整的环游后将它取下来，或者是帧的指定接收站。

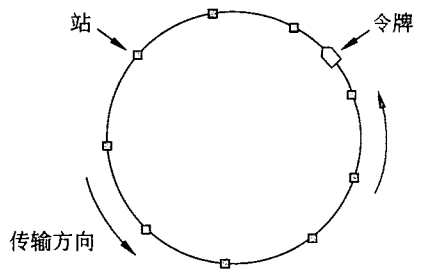


图4-7 令牌环

请注意，我们并不需要一个物理环来实现令牌传递。相反，连接各站的信道可以是一根长总线。每个站通过该总线按照预定义的顺序把令牌发给下一站。令牌的拥有者可以利用总线发送帧，如前所述。这个协议称为**令牌总线**（token bus）。

令牌传递的性能类似于位图协议，尽管现在竞争槽和帧被混合在一个周期中。发送一帧后，每个站必须等待所有N个站（包括其自身）把令牌发给各自的邻居，以及其他N-1个站发送完一帧（如果它们有帧需要发送）。两者的一个细微差别在于，因为在周期中所有的位置是均等的，所以不存在偏向低编号或者高编号一说。对于令牌环，每个站在协议采取下一步动作之前只将令牌尽可能发送给它的邻居。在协议前进到下一步之前不需要将每个令牌传播给全部站。

令牌环随之显身于MAC协议，这些协议具有某种一致性。早期的令牌环协议（称为令牌环（Token Ring），并标准化为IEEE 802.5）在20世纪80年代非常流行，是经典以太网之外的另一种局域网选择。到了20世纪90年代，一种更快的令牌环，称为光纤分布式数据接口（FDDI，Fiber Distributed Data Interface）被交换式以太网击败。2000年后，一个称为弹性数据包环（RPR，Resilient Packet Ring）的令牌环被定义为IEEE 802.17，这是对ISP使用的城域环组合所制定的标准化。我们很想知道2010年以后将提供什么样的服务。

二进制倒计数

基本位图协议存在一个问题。每个站的开销是1位，所以该协议不可能很好地扩展到含有上千个站的网络中，扩展后的令牌传递也有同样的问题。通过使用二进制的站地址，我们可以做得更好。如果一个站想要使用信道，它就以二进制位串的形式广播自己的地址，从高序的位开始。假定所有地址都有同样的长度。不同站地址中相同位在同时发送时被信道布尔或（BOOLEAN OR）在一起。我们把这样的协议称为**二进制倒计数**（binary countdown）协议，它曾经被用在Datakit中（Fraser，1987）。它隐式地假设传输延迟可忽

略不计，因此，所有站几乎能同时看到地址宣告位。

为了避免冲突，必须使用一条仲裁规则：一个站只要看到自己的地址位中的 0 值位置被改写成了 1，则它必须放弃竞争。例如，如果站 0010、0100、1001 和 1010 都试图要获得信道，在第一位时间中，这些站分别传送 0、0、1 和 1。它们被 OR 在一起，得到 1。站 0010 和 0100 看到了 1，它们立即明白有高序的站也在竞争信道，所以它们放弃这一轮的竞争。而站 1001 和 1010 则继续竞争信道。

接下来的位为 0，于是两者继续竞争；再接下来的位为 1，所以站 1001 放弃。最后的胜者是 1010，因为它有最高的地址。在赢得了竞争后，它现在可以传输一帧，之后又开始新一轮竞争。该协议由图 4-8 举例说明。该协议具有这样一种特性，高序站的优先级比低序站的优先级高，这可能是好事，也可能是坏事，取决于上下文。

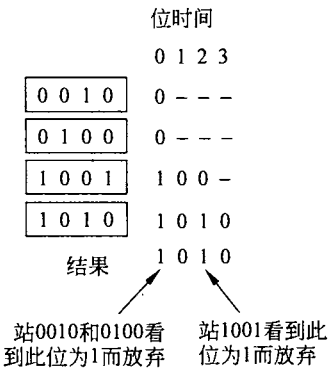


图 4-8　二进制倒计数协议
虚线表示不参与

这种方法的信道利用率为 $d/(d+\log_2 N)$。然而，如果精心设计帧格式，使得发送方的地址正好是帧内的第一个字段，那么，甚至 $\log_2 N$ 位也不会被浪费，所以信道利用率为 100%。

二进制倒计数协议是一个简单的、精致的和高效的协议实例，它有待于被重新发现。希望有一天它能找到一个新的用武之地。

4.2.4　有限竞争协议

如何在一个广播网络中获取信道，我们已经考虑了两种基本策略：一种是竞争的方法，如同 CSMA 的做法那样；另一种是无竞争协议。每一种都可以用两个重要性能指标来衡量：低负载下的延迟，以及高负载下的信道利用率。在负载较轻的情况下，竞争方法（即纯 ALOHA 或者分槽 ALOHA）更为理想，因为它的延迟较短（冲突很少发生）。随着负载的增加，竞争方法变得越来越缺乏吸引力，因为信道仲裁所需要的开销变得越来越大。而对于无冲突协议，则结论刚好相反。在低负载情况下，它们有相对高的延迟，但是随着负载的增加，信道的效率反而得到提高（因为开销是固定的）。

显然，如果我们能够把竞争协议和无冲突协议的优势结合起来，那就太好了。这样得到的新协议在低负载下采用竞争的做法而提供较短的延迟，但在高负载下采用无冲突技术，从而获得良好的信道效率。我们把这样的协议称为**有限竞争协议**（limited-contention protocol），实际上这样的协议的确存在，我们将用它来结束关于载波侦听网络的学习。

到现在为止我们学习过的竞争协议都是对称的，也就是说，每个站企图获得信道的概率为 p，并且所有的站都使用同样的 p。很有意思的是，如果协议为不同的站分配不同的概率，有时系统的整体性能会有所提高。

在讨论非对称协议之前，让我们快速回顾一下对称协议的性能情况。假设共有 k 个站竞争信道的使用权。每个站在每个时间槽中的传输概率为 p。那么，在一个给定的时间槽中，某个站能够成功地获得信道的概率是任何一个站以概率 p 传输，而所有 k-1 个站以 1-p

概率把传输延缓到下个时间槽，这个值为 $kp(1-p)^{k-1}$。为了找到 p 的最优值，我们按照 p 对它求微分，再将结果设置为 0，解出 p 值。这样做之后，我们发现，p 的最佳值为 1/k。将 p=1/k 代入，则得到

$$\Pr[\text{p 为最佳值时的成功概率}]\left(\frac{k-1}{k}\right)^{k-1} \tag{4-4}$$

这个概率的曲线如图 4-9 所示。对于站数较少的情形，成功的概率很高；但是，一旦站的数量达到 5 个以后，概率很快下降，接近于它的逼近值 1/e。

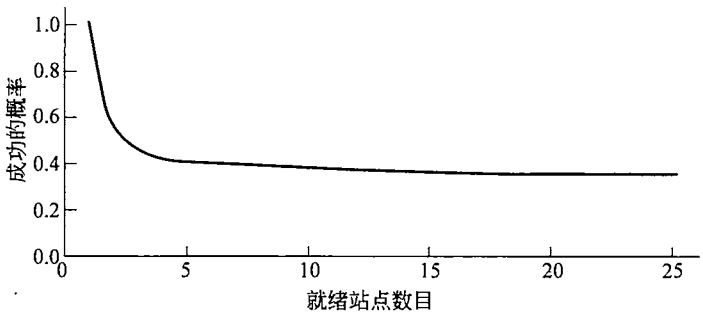


图 4-9 对称竞争信道的获得概率

从图 4-9 可以很明显地看到，只要减少参与竞争的站数量，则站获得信道的概率就会增加。有限竞争协议正是这样做的。它们首先将所有的站划分成组（这些组不必是两两不相交的）。只有 0 号组的成员才允许竞争 0 号时间槽。如果该组中的一个成员竞争成功了，则它获得信道，可以传送它的帧。如果该时间槽是空闲的，或者发生了冲突，则 1 号组的成员竞争 1 号时间槽，以此类推。通过适当的分组办法，可以减少每个时间槽中的竞争数量，从而使得每个时间槽中的行为接近图 4-9 的左侧。

协议的诀窍在于如何将站分配到各个时间槽中。在讨论一般情形以前，我们先考虑一些特殊情形。在一种极端情况下，每个组只包含一个站。这样的分配方案可以保证永远不会发生冲突，因为对于任何给定的时间槽，至多只有一个站参与竞争。前面我们已经看到过这样的协议（比如二进制倒计数协议）。另一种特殊的情形是每个组有两个站。在一个时间槽中，两个站都要传送数据的概率是 p^2，对于很小的 p，这个值可以忽略不计。随着分配在同一个时间槽中的站数越来越多，冲突的概率也随之增加，但是给予各站竞争机会所需的位图扫描长度却缩小了。有限的情况是每个组包含全部的站（即分槽 ALOHA）。我们所需要的是一种动态地将站分配到时间槽的方法，当负载很低时，每个时间槽中的站点数量就多一些；当负载很高时，每个时间槽中的站点数量就少一些，甚至只有一个站。

自适应树遍历协议

有一种特别简单方法可以用来执行必要的信道分配任务，那就是采用第二次世界大战中美国军方为了测试士兵是否感染梅毒而设计的算法（Dorfman，1943）。简短来说，军方从 N 个士兵身上提取血样。然后从每份血样中各取一部分倒入同一个试管中，再对这份混合的血样进行抗体测试。如果没有发现抗体，则所有的士兵都是健康的。如果出现了抗体，则再准备两份新的混合血样，一份由 1 到 N/2 士兵的血样混合而成，另一份由剩余士兵的血样混合而成。这个过程重复递归，直到找出被感染的士兵。

至于该算法的计算版本（Capetanakis，1979）很自然地把站看作是二叉树的叶节点，如图 4-10 所示。在一次成功传送之后的第一个竞争槽，即 0 号槽中，允许所有的站尝试获取信道。如果它们之中的某一个获得了信道，则很好；如果发生了冲突，则在 1 号槽中，只有位于树中 2 号节点之下的那些站才可以参与竞争。如果其中的某个站获得了信道，则在该站发送完一帧之后的那个槽被保留给位于节点 3 下面的那些站。另一方面，如果节点 2 下面的两个或者多个站都要传输数据，则在 1 号槽中就会发生冲突，此时，下一个槽，即 2 号槽就由位于节点 4 下面的站来竞争。

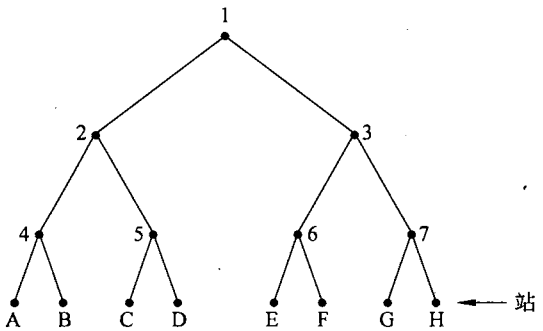


图 4-10　包含 8 个站的树

本质上，如果在 0 号槽中发生了冲突，则整棵树都会被遍历到，按深度优先策略找到所有就绪站。每一个位槽都跟树中某些特定的节点相关联。如果发生了冲突，则在该节点的左子节点和右子节点上继续递归地进行搜索。如果一个位槽是空闲的，或者在位槽中只有一个站传送数据，则停止该节点的搜索，因为这表明已经找到了该节点下面所有就绪的站（如果有多个就绪站，一定会发生冲突）。

当系统负载较重，将 0 号槽专门指定给节点 1 几乎不值得，因为只有当精确地只有一个站有帧要发送，这才是有意义；而当负载较重时这种事件发生的可能性非常小。类似地，有人可能会说，基于同样的理由，节点 2 和节点 3 也可以跳过去。考虑更为一般的情形，到底应该从树的哪一级开始搜索呢？很明显，系统负载越重，则越是应该从树的下面节点开始搜索。我们假设每个站对全部就绪站的数目有良好的估算 q，例如，可以从监测当前的流量来推导出就绪站的估算数。

若要继续，让我们对树的级数从上往下进行编号，在图 4-10 中，节点 1 位于第 0 级，节点 2 和节点 3 位于第 1 级，以此类推。请注意，第 i 级上的每个节点是其下面总站数的 2^{-i}。如果 q 个就绪站均匀分布，则在第 i 级上某一个特定节点下面期望的就绪站数是 $2^{-i}q$。直观上，我们期望开始搜索的最优级数应该是每个槽中参与竞争的平均站数为 1，也就是说，在这级上，$2^{-i}q=1$。求解这个方程，我们可以得到 $i=\log_2 q$。

这个基本算法已经有了大量的改进算法，（Bertsekas 和 Gallager，1992）对这些算法作了详细的讨论。例如，考虑这样一种情况：只有站 G 和 H 想要发送数据。在结点 1 上，发生冲突，所以节点 2 下面的站开始竞争，却发现它是空闲的；此时探测节点 3 毫无意义，因为它肯定会冲突（我们已经知道在节点 1 下面有两个或者多个站就绪，并且这些站都不在节点 2 的下面，所以，它们肯定都在节点 3 的下面）。对节点 3 的探测可以跳过去，直接探测节点 6。当这次探测结果仍然是空时，节点 7 可以跳过去，尝试节点 G 了。

4.2.5 无线局域网协议

笔记本电脑通过无线电进行通信，它们组成的系统可以看作是**无线局域网**（wireless LAN），就像我们在1.5.3节讨论的一样。这样的局域网是广播信道的一个例子。它具有某些和有线局域网不同的属性，这些属性导致了无线局域网不同的MAC协议。在本节，我们将研究这些协议中的一些。在4.4节，我们将着眼于802.11（WiFi）细节。

无线局域网的一种常见配置是在一座办公大楼内，有策略地放置一些环绕大楼的**接入点**（AP）。AP通过铜缆或光纤连接在一起，并为与之联系的站提供接入服务。如果AP和笔记本电脑的发射功率调整在一个数十米的范围内，附近的房间就变得像单个蜂窝，而整个大楼就像我们在第2章学习的蜂窝电话系统；但有一点除外，那就是每个蜂窝只有一个信道可用。这个信道被蜂窝内所有的站共享，包括AP。它通常提供Mbps的带宽，最大可高达600 Mbps。

我们已经说过，无线通信系统通常不能检测出正在发生的冲突。站接收到的信号可能很微弱，也许比它发出去的信号弱上百万倍。要发现这样的冲突信号就像在海上寻找涟漪一样的困难。相反，确认机制可用在事后发现冲突和其他错误。

无线局域网和有线局域网之间还存在着一个更重要的差异。由于无线电传输范围有限，无线局域网中的站或许无法给所有其他站发送帧，也无法接收来自所有其他站的帧。在有线局域网中，一个站发出一帧，所有其他站都能接收到。正是由于无线局域网缺乏这种属性，导致了它的一系列复杂性。

我们将做出简化的假设：即每个无线电发射器有某个固定的传播范围，用一个圆形覆盖区域表示，在这区域内的另一个站可以侦听并接收该站的传输。重要的是要认识到，实际上的覆盖区域没有那么规则，因为无线电信号的传播依赖于所在的环境。墙壁和其他障碍物对信号都会造成衰减和反射，这些可能会导致信号在不同方向上表现出显著不同。但是一个简单的圆形模式有助于达到我们的描述目的。

使用无线LAN的一种单纯想法是尝试使用CSMA：每个站侦听是否有其他站在传输，并且只有当没有其他站在传送数据时它才传输。这种方法的麻烦在于协议并未考虑无线传输特性，因为这里的冲突发生在接收方，而不是发送方。为了看清楚问题的实质，考虑如图4-11所示的4个无线站的情形。对于我们的问题描述来说，这些无线站到底是基站还是笔记本电脑无关紧要。无线电的覆盖范围是这样的：A和B都在对方的范围内，可能潜在地相互之间有干扰；C也可能潜在地干扰到B和D，但不会干扰A。

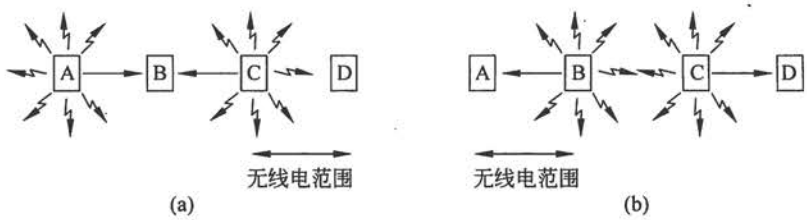


图4-11 一个无线LAN

（a）在给B传输时A和C是隐藏终端；（b）在给A和D传输时B和C是暴露终端

首先考虑当 A 和 C 向 B 传送数据时的情形，如图 4-11（a）所示。如果 A 开始发送，然后 C 立即进行侦听介质，它将不会听到 A 的传输，因为 A 在它的覆盖范围之外。因此 C 错误地得出结论：它可以向 B 传送数据。如果 C 传送数据，将在 B 处产生冲突，从而扰乱 A 发来的帧。（我们假设没有类似 CDMA 的编码模式可以提供多信道传输，因此冲突会扰乱信号，从而破坏两个帧）我们需要一个 MAC 协议，它能防止这种冲突的发生，因为冲突将导致带宽的浪费。由于竞争者离得太远而导致站无法检测到潜在的竞争者，这个问题称为**隐藏终端问题**（hidden station problem）。

现在让我们来考虑另一种不同的情形：B 向 A 传送数据，同时 C 想给 D 发送数据，如图 4-11（b）所示。如果 C 侦听介质，它会听到有一个传输正在进行，从而会错误地得出结论：它不能向 D 发送数据（图中的虚线表示）。事实上，C 所听到的传输只会搞坏 B 和 C 之间区域中的接收，但是，两个接收方都不在这个危险区域。我们需要一个 MAC 协议，它能防止此类延迟传输的发生。这个问题称为**暴露终端问题**（exposed station problem）。

这里的困难在于开始传输之前站真正希望知道的是在接收方周围是否有无线电活动。CSMA 只能告知在侦听载波的站附近是否有活动发生。对于有线情形，所有的信号能传播到全部的站，所以这种区别并不存在；然而，在同一时刻，无论在系统中任何地方都只能有一个传输在进行。在一个基于短程无线电波的系统中，多个传输可以同时发生，只要它们有不同的目的地，并且这些目的地都不在彼此的范围内。我们希望当蜂窝变得越来越大时这种并发性能够发生。类似的情形发生在聚会中，舞会中的人们不会等到房间里的每个人都沉默了才开始交谈；在一个大房间里可以同时发生多个交谈，只要交谈者不在相同的位置。

能处理无线 LAN 这些问题的一个早期且有影响力的协议是**冲突避免多路访问**（MACA，Multiple Access with Collision Avoidance）（Karn，1990）。MACA 的基本思想是发送方刺激接收方输出一个短帧，以便其附近的站能检测到该次传输，从而避免在接下去进行的（较大）数据帧传输中也发送数据。这项技术被用来替代载波侦听。

图 4-12 说明了 MACA 协议。现在我们来考虑 A 如何向 B 发送一帧。A 首先给 B 发送一个 RTS（Request To Send）帧，如图 4-12（a）所示。这个短帧（30 字节）包含了随后将要发送的数据帧的长度；然后，B 用一个 CTS（Clear to Send）作为应答，如图 4-12（b）所示。此 CTS 帧也包含了数据长度（从 RTS 帧中复制过来）。A 在收到了 CTS 帧之后便开始传输。

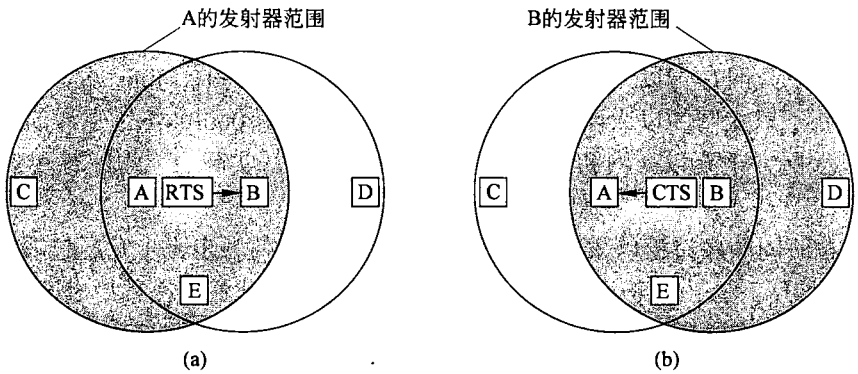


图 4-12　MACA 协议
（a）A 给 B 发送一个 RTS；（b）B 作为响应给 A 返回一个 CTS

现在我们来看，如果其他站也听到了这些帧，它们会如何反应。如果一个站听到了 RTS 帧，那么它一定离 A 很近，它必须保持沉默，至少等待足够长的时间以便在无冲突情况下 CTS 被返回给 A。如果一个站听到了 CTS，则它一定离 B 很近，在接下来的数据传送过程中它必须一直保持沉默，只要检查 CTS 帧，该站就可以知道数据帧的长度（即数据传输要持续多久）。

在图 4-12 中，C 落在 A 的范围内，但不在 B 的范围内。因此，它听到了 A 发出的 RTS，但是没有听到 B 发出的 CTS。只要它没有干扰 CTS，那么在数据帧传送过程中，它可以自由地发送任何信息。相反，D 落在 B 的范围内，但不在 A 的范围内。它听不到 RTS 帧，但是听到了 CTS 帧。只要听到了 CTS 帧，这意味着它与一个将要接收数据帧的站离得很近；所以，它就延缓发送任何信息直到那个帧如期传送完毕。站 E 听到了这两条控制消息，与 D 一样在数据帧完成之前它必须保持安静。

尽管有了这些防范措施，冲突仍有可能会发生。例如，B 和 C 可能同时给 A 发送 RTS 帧。这些帧将发生冲突，因而丢失。在发生了冲突的情况下，一个不成功的发送方（即在期望的时间间隔内没有听到 CTS）将等待一段随机的时间，以后再重试。

4.3 以 太 网

现在我们已经完成了关于信道分配协议的抽象讨论，是时候看看这些原则如何应用于实际系统了。许多为个域网、局域网和城域网所做的设计都已经在 IEEE 802 名义下进行了标准化。有几个协议幸免了下来，但还有许多没能逃脱被休眠的命运，正如我们在图 1-38 看到的。一些相信轮回的人甚至开玩笑说，一定是卡尔·达尔文转世加入了 IEEE 标准协会，从而淘汰那些不合时宜的标准。最重要的幸存者是 802.3（以太网）和 802.11（无线局域网）。蓝牙（无线 PAN）得到了广泛部署，但现在其标准化的工作已经独立于 802.15 之外。至于 802.16（无线城域网），现在说它还为时过早。请参考本书第 6 版来深入了解它。

我们将开始研究太网网实际系统，这可能是现实世界中最普遍的一种计算机网络。以太网有两类：第一类是**经典以太网**（classic Ehernet），它使用我们在本章已学过的技术解决了多路访问问题；第二类是**交换式以太网**（switched Ethernet），使用了一种称为交换机的设备连接不同的计算机。重要的是要注意，虽然它们都称为以太网，但它们有很大的不同。经典以太网是指以太网的原始形式，运行速度从 3～10 Mbps 不等；而交换式以太网正是成就了以太网的以太网，可运行在 100、1000 和 10000 Mbps 那样的高速率，分别以快速以太网、千兆以太网和万兆以太网的形式呈现。实际上，现在使用的只有交换式以太网。

我们将按时间顺序讨论以太网的历史，展示它们是如何一步一步发展起来的。由于以太网和 IEEE 802.3 几乎完全一致，除了一个微小差别外（我们将在稍后讨论），许多人交替使用“以太网”和“IEEE 802.3”这两个术语。我们也将这样做。若需了解更多的以太网信息，请参见（Spurgeon，2000）。

4.3.1　经典以太网物理层

以太网的故事始于 ALOHA 同时期，确切的时间是在一个名叫 Bob Metcalfe 的学生获得麻省理工学院的学士学位后，搬到河对岸的哈佛大学攻读博士学位之际。在他学习期间，他接触到了 Abramson 的工作，他对此很感兴趣。从哈佛毕业后，他决定在前往施乐帕洛阿尔托研究中心（Palo Alto Research Center）正式工作之前留在夏威夷度暑假，以便帮助 Abramson 工作。当他到达帕洛阿尔托研究中心，他看到那里的研究人员已经设计并建造出后来称为个人计算机的机器，但这些机器都是孤零零的；他便运用帮助 Abramson 工作获得的知识与同事 David Boggs 设计并实现了第一个局域网（Metcalfe 和 Boggs，1976）。该局域网采用一个长的粗同轴电缆，以 3Mbps 速率运行。

他们把这个系统以发光性乙醚的名称（luminiferous ether）命名为以太网（Ethernet），人们曾经认为通过它可以传播电磁辐射（当 19 世纪英国物理学家 James Clerk Maxwell 发现电磁辐射可以用一个波方程来描述，科学家们一直认为空中充满着一种空灵的介质，电磁辐射才得以传播。只有在 1887 年著名的 Michelson-Morley 实验之后，物理学家才发现电磁辐射可以在真空中传播）。

施乐以太网（Xerox Ethernet）获得了巨大的成功，以至于 DEC、英特尔和施乐公司在 1978 年制定了一个 10Mbps 以太网标准，称为 DIX 标准（DIX standard）。做了少许修订后，DIX 标准在 1983 年正式成为 IEEE 802.3 标准。不幸的是，早已拥有一个历史性开创发明（例如个人计算机）的施乐公司后来却失败于商业化过程。《探索未来》（Fumbling the Future）告诉了你一个故事（Smith 和 Alexander，1988）。当施乐表现出对以太网除了协助制订标准外没有多达兴趣后，Metcalfe 创建了自己的公司 3Com，出售用于个人计算机的以太网适配器。它卖出了好几百万。

经典以太网用一个长电缆蜿蜒围绕着建筑物，这根电缆连接着所有的计算机。这种体系结构如图 4-13 所示。第一个产品，俗称粗以太网（thick Ethernet），用类似黄色的花园用软管连接着所有的计算机，在电缆的每 2.5 米处有个标记，这个标记就是连接计算机的位置（802.3 标准实际上并没有要求电缆是黄色，但它确实建议用黄色）。它的继任者是细以太网（thin Ethernet），电缆比粗以太网柔软，更加易于弯曲，并且使用了工业标准 BNC 接头。细以太网更便宜也更容易安装，但它单段电缆仅 185 米（而不是粗以太网的 500 米）长，且每段电缆只能处理 30 台计算机（而不是 100 台）。

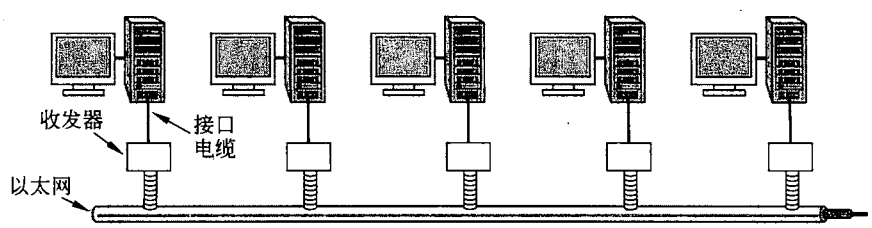


图 4-13　经典以太网体系结构

以太网的每个版本都有电缆的最大长度限制（即无须放大的长度），这个范围内的信号

可以正常传播，超过这个范围信号将无法传播。为了允许建设更大的网络，可以用**中继器**（repeater）把多条电缆连接起来。中继器是一个物理层设备，它能接收、放大（即再生）并在两个方向上重发信号。至于软件方面，一系列由中继器连接起来的电缆段与一根单独的电缆没有什么不同（除了由中继器引入的少量延迟外）。

在这些电缆上，信息的发送使用我们在 2.5 节学过的曼彻斯特编码。以太网可以包含多个电缆段和多个中继器，但是不允许任意两个收发器之间的距离超过 2.5 千米，并且任意两个收发器之间经过的中继器不能超过 4 个。之所以有这样的限制在于这是保证 MAC 协议正常工作的需要，我们下面将着眼于这点。

4.3.2 经典以太网 MAC 子层协议

帧格式如图 4-14 所示。首先是 8 个字节的前导码（Preamble），每个字节包含比特模式 10101010（除了最后一个字节的最后 2 位为 11）。这最后一个字节称为 802.3 的帧起始定界符（Start of Frame，SOF）。比特模式是由曼彻斯特编码产生的 10 MHz 方波，每个波 6.4 微秒，以便接收方的时钟与发送方同步。最后两个“1”告诉接收方即将开始一个帧。

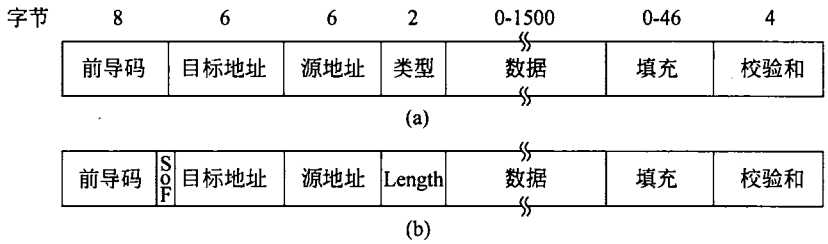


图 4-14 帧格式
（a）以太网（DIX）；（b）IEEE 802.3

接下来是两个地址字段，一个标识目的地址，另一个标识帧的发送方。它们均为 6 个字节长。如果传输出去的目标地址第一位是 0，则表示这是一个普通地址；如果是 1，则表示这是一个组地址。组地址允许多个站同时监听一个地址。当某个帧被发送到一个组地址，该组中的所有站都要接收它。往一组地址的发送行为称为**组播**（multicasting）。由全 1 组成的特殊地址保留用作**广播**（broadcasting）。如果一个帧的目标地址字段为全 1，则它被网络上的所有站接收。组播是更多的选择，但它涉及确定组内有哪些成员的组管理。相反，广播根本不区分站，因此不需要任何组管理机制。

站的源地址有一个有趣的特点，那就是它们具有全球唯一性。该地址由 IEEE 统一分配，因此确保了在世界任何地方没有两个站的地址是相同的。只要给出正确的 48 位数字，任何站都可以唯一寻址到该数字代表的任何其他站。要做到这一点，地址字段的前 3 个字节用作该站所在的**组织唯一标识符**（OUI，Organizationally Unique Identifier）。该字段的值由 IEEE 分配，指明了网络设备制造商。制造商获得一块大小为 2^{24} 的地址。地址字段的最后 3 个字节由制造商负责分配，并在设备出厂之前把完整的地址用程序编入 NIC。

接下来是类型（Type）或长度（Length）字段，究竟采用哪个字段取决于是以太网帧还是 IEEE 802.3 帧。以太网使用类型字段告诉接收方帧内包含了什么。同一时间在同一台机器上或许用了多种网络层协议，所以当一个以太帧到达接收方时，操作系统需要知道应

该调用哪个网络层协议来处理帧携带的数据包。类型字段指定了把帧送给哪个进程处理。例如，一个值为 0x0800 的类型代码意味着帧内包含一个 IPv4 的数据包。

IEEE 802.3 以其智慧决定该字段携带帧的长度，因为以太网的长度必须由其内部携带的数据来确定——如果真是这样的话，则违反了分层规定。当然，IEEE 的处理方式意味着接收方没有办法确定如何处理入境帧。这个问题由数据内包含的另一个**逻辑链路控制**（LLC，Logical Link Control）协议头来处理。它使用 8 个字节来传达 2 个字节的协议类型信息。

不幸的是，在 802.3 标准出炉时，已经有太多的 DIX 以太网硬件和软件在使用，以至于很少有厂家和客户有热情来重新包装类型和长度字段。1997 年，IEEE 认输并表示这两种字段都可以接受。幸运的是，所有在 1997 年之前使用的类型字段其值都大于 1500，因此规定了最大数据长度。现在的规则是，任何值小于或等于 0x600（1536）可解释为长度字段，任何大于 0x600 可解释为类型字段。现在 IEEE 可以认为每个人都使用了它的标准，并且其他人可以继续做他们正在做的事情（不能打扰 LLC），它因而无须感到内疚。

接下来是数据（Data）字段，最多可包含 1500 个字节。在制定 DIX 标准时，这个值的选择有一定的随意性。当时最主要的考虑是收发器需要足够的内存（RAM）来存放一个完整的帧，而 RAM 在 1978 年时还很昂贵。这个上界值越大意味着需要的 RAM 更多，因而收发器的造价更高。

除了有最大帧长限制外，还存在一个最小帧长的限制。虽然有时候 0 字节的数据字段也是有用的，但它会带来一个问题。当一个收发器检测到冲突时，它会截断当前的帧，这意味着冲突帧中已经送出的位将会出现在电缆上。为了更加容易地区分有效帧和垃圾数据，以太网要求有效帧必须至少 64 字节长，从目标地址算起直到校验和，包括这两个字段本身在内。如果帧的数据部分少于 46 个字节，则使用填充（Pad）字段来填充该帧，使其达到最小长度要求。

限制最小帧长的另一个（也是更重要的）理由是避免出现这样的情况：当一个短帧还没有到达电缆远端的发送方，该帧的传送就已经结束；而在电缆的远端，该帧可能与另一帧发生冲突。这个问题如图 4-15 所示。在 0 时刻，位于电缆一端的站 A 发出一帧。我们假设该帧到达另一端的传播时间为 τ。正好在该帧到达另一端之前的某一时刻（即，在 τ-ε 时刻），位于最远处的站 B 开始传送数据。当 B 检测到它所接收到的信号比它发送的信号更强时，它知道已经发生了冲突，所以放弃了自己传送，并且产生一个 48 位的突发噪声以

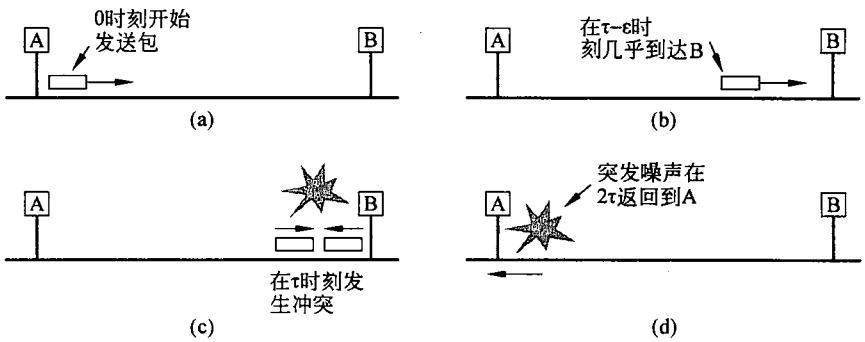


图 4-15　冲突检测只需要 2τ 时间

警告所有其他站。换句话说，它阻塞了以太电缆，以便确保发送方不会漏检这次冲突。大约在2τ后，发送方看到了突发噪声，并且也放弃自己的传送。然后它等待一段随机的时间，再次重试。

如果一个站试图传送非常短的帧，则可以想象：虽然发生了冲突，但是在突发噪声回到发送方（2τ）之前，传送已经结束。然后，发送方将会得出刚才一帧已经成功发送的错误结论。为了避免发生这样的情况，所有帧必须至少需要2τ时间才能完成发送，这样当突发噪声回到发送方时传送过程仍在进行。对于一个最大长度为2500米、具有4个中继器的10 Mbps LAN（符合802.3规范），在最差情况下，往返一次的时间大约是50微秒（其中包括了通过4个中继器所需要的时间）。因此，允许的最小帧长必须至少需要这样长的时间来传输。以10 Mbps的速率，发送一位需要100纳秒，所以500位是保证可以工作的最小帧长。考虑到加上安全余量，该值被增加到512位，或者64字节。

最后一个字段是校验和（Checksum）。它是我们在3.2节学过的32位CRC。事实上，它由我们曾经给出的生成多项式确切定义，这个CRC同样被用于PPP、ADSL和其他链路层协议。CRC是差错检测码，用来确定接收到的帧比特是否正确。它只提供检错功能，如果检测到一个错误，则丢弃帧。

二进制指数后退的CSMA/CA

经典以太网使用1-坚持CSMA/CD算法，我们在4.2节对此有所描述。这个算法意味着当站有帧需要发送时要侦听介质，一旦介质变为空闲便立即发送。在它们发送的同时监测信道上是否有冲突。如果有冲突，则立即中止传输，并发出一个短冲突加强信号，在等待一段随机时间后重发。

现在我们来了解当冲突发生后如何确定随机等待的时间。我们仍然使用图4-5所示的模型。在冲突发生后，时间被分成离散的时间槽，其长度等于最差情况下在以太介质上往返传播时间（2τ）。为了达到以太网允许的最长路径，时间槽的长度被设置为512比特时间，或51.2微秒。

第一次冲突发生后，每个站随机等待0个或者1个时间槽，之后再重试发送。如果两个站冲突之后选择了同一个随机数，那么它们将再次冲突。在第二次冲突后，每个站随机选择0、1、2或者3，然后等待这么多个时间槽。如果第三次冲突又发生了（发生的概率为0.25），则下一次等待的时间槽数从0到2^3-1之间随机选择。

一般地，在第i次冲突之后，从0～2^i-1之间随机选择一个数，然后等待这么多个时间槽。然而，达到10次冲突之后，随机数的选择区间被固定在最大值1023，以后不再增加。在16次冲突之后，控制器放弃努力，并给计算机返回一个失败报告。进一步的恢复工作由高层协议完成。

这个算法称为**二进制指数后退**（binary exponential backoff），它可以动态地适应发送站的数量。如果所有冲突的随机数最大值都是1023，则两个站第二次发生冲突的概率几乎可以忽略；但是，在一次冲突之后的平均等待时间将是数百个时间槽，这样会引入很大的延迟。另一方面，如果有100个站都要发送数据，每个站总是等待0个或者1个时间槽，那么，它们将一而再、再而三地一次次发生冲突，直到其中的99个站选择1而剩下一个站选择0。这种情况可能需要等上好几年的时间才有可能发生。随着连续冲突的次数越来越多，

随机等待的间隔呈指数增加，这样算法能确保两种情况：如果只有少量站发生冲突，则它可确保较低的延迟；当许多站发生冲突时，它也可以保证在一个相对合理的时间间隔内解决冲突。将延迟后退的步子截断在 1023 可避免延迟增长得太大。

如果没有发生碰撞，发送方就假设该帧可能被成功传递了。也就是说，无论是 CSMA/CD 还是以太网都不提供确认。这样的选择适用于出错率很低的有线电缆和光纤信道。确实发生的任何错误必须通过 CRC 检测出来并由高层负责恢复。对于无线信道，因其出错率高还得使用确认手段，这点我们将在后面章节部分说明。

4.3.3　以太网性能

现在，让我们简要地探讨在重负载和恒定负载条件下（即总是有 k 个站要传送数据）经典以太网的性能。关于二进制指数后退算法的严格分析非常复杂。因此，我们采用了（Metcalfe 和 Boggs，1976）方法，并假定每个时间槽中重传的概率是个常数。如果每个站在一个竞争时间槽中传送帧的概率为 p，那么，在这个时间槽中，某一个站获得信道的概率 A 为：

$$A=kp(1-p)^{k-1} \tag{4-5}$$

当 p=1/k 的时候，A 最大；并且当 k→∞的时候，A→1/e。竞争间隔正好等于 j 个时间槽的概率为 $A(1-A)^{j-1}$，所以每一次竞争的平均时间槽数为：

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1}=\frac{1}{A}$$

由于每个时间槽的间隔时间为 2τ，因此平均竞争间隔 w 为 2τ/A。假设最优 p，并且竞争时间槽的平均数永远不超过 e，于是，w 至多为 2τe≈5.4τ。

如果传送一帧平均需要 P 秒，那么，当许多站都要传送帧时，信道效率

$$\text{信道效率}=\frac{P}{P+2\tau/A} \tag{4-6}$$

这里我们可以看到任何两个站之间的最大电缆距离也会影响到性能。电缆越长，则竞争间隔也越长。这正是为什么以太网标准规定了最大电缆长度的原因。

针对每个帧 e 个竞争时间槽的最优情形，按照帧的长度 F、网络带宽 B、电缆长度 L 和信号的传播速度 c，来制定等式（4-6）是有指导意义的。利用 P= F/B，等式（4-6）变成：

$$\text{信道效率}=\frac{1}{1+2BLe/cF} \tag{4-7}$$

当分母中的第二项变大时，网络的效率将会变低。特别是，在给定帧长度的情况下，增加网络带宽或者距离（即 BL 的乘积）将会降低网络效率。不幸的是，许多网络硬件方面的研究都要把增大此乘积值作为目标。人们总是希望在长距离上拥有高带宽（例如，光纤城域网），而用这种方式实现的以太网可能并不是这些应用的最佳系统。我们将在下节看到实现以太网的其他一些方法。

图 4-16 给出了利用等式（4-7），在 2τ=51.2 微秒以及 10 Mbps 数据速率的情况下，信道效率与就绪站数之间的关系。对于 64 字节的时间槽时间，64 字节的帧并不是最有效的，这并不奇怪。另一方面，当帧长度为 1024 字节，以及每个竞争间隔趋近于 e 个 64 字节时

间槽时，竞争期为174字节长，效率为85%。这个结果比分槽ALOHA的37%利用率要好得多。

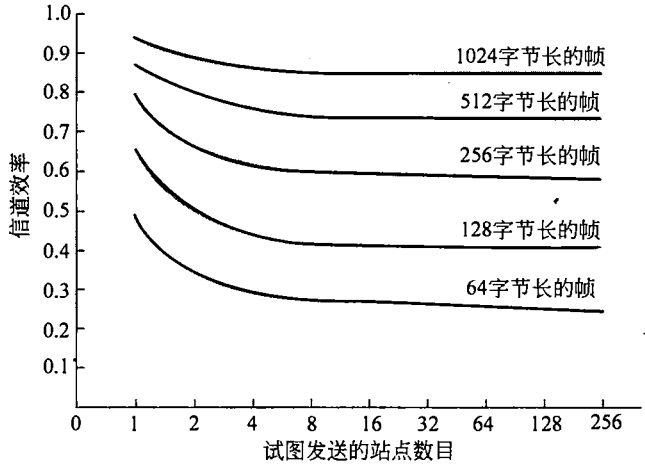


图4-16 具有512位时间槽的10Mbps以太网效率

或许有一点值得一提，关于以太网（和其他的网络）有大量理论性的性能分析成果。大多数研究结果都只有很少的参考价值，原因有两个。首先，几乎所有理论工作都假设网络流量符合泊松分布。当研究人员开始观察实际数据，他们发现网络流量很少呈现泊松分布，而是在一定的时间尺度上表现出自相似或者突发性（Paxson 和 Floyd，1995；Leland等，1994）。这意味着即使在一段较长的时间上进行平均也不能使流量变得平滑。除了理论分析时采用的可疑模型外，许多研究把分析重点放在异常高负载情况下的“有趣”性能上。（Boggs等，1988）通过实验显示以太网在实际情况下工作很好，即使在适度的高负载下。

4.3.4 交换式以太网

以太网的发展很快，从单根长电缆的典型以太网结构开始演变。单根电缆存在的问题，比如找出断裂或者松动位置等与连接相关的问题，趋势人们开发出一种不同类型的布线模式。在这种模式中，每个站都有一条专用电线连接到一个中央集线器。集线器只是在电气上简单地连接所有连接线，就像把它们焊接在一起。这种配置如图4-17（a）所示。

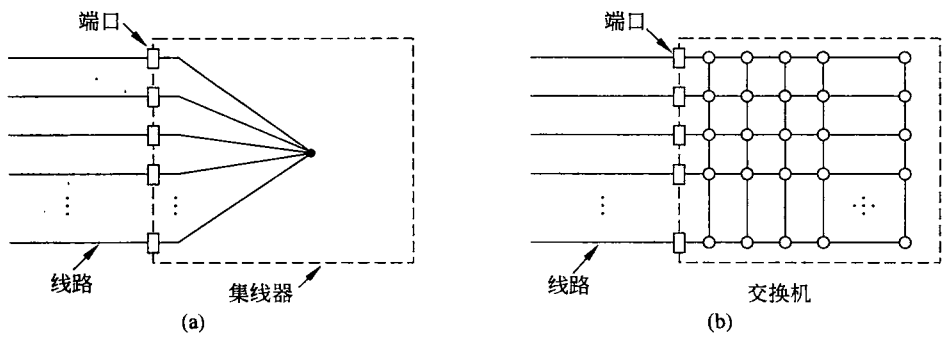


图 4-17
（a）集线器；（b）交换机

电线就是电话公司的双绞线，因为大多数写字楼已布有这种线，而且通常足够富裕。电话线的这种重用是个双赢局面，但它把允许的最长电缆长度减少到目前离集线器不得大于 100 米的限制（如果是高品质的 5 类双绞线则可扩展到 200 米）。在这种配置下，添加或删除一个站非常容易，电缆断裂也很容易被直接检测出来。因为具有使用现有布线和易于维护等优点，双绞线集线器迅速成为以太网的主要形式。

然而，集线器不能增加容量，因为它们逻辑上等同于单根电缆的经典以太网。随着越来越多的站加入，每个站获得的固定容量共享份额下降。最终，LAN 将饱和。一条出路是提升到更高的速度，比如从 10 Mbps 升到 100 Mbps、1 Gbps，甚至更高的速率。但是，随着多媒体和功能强大服务器的增长，即使 1 Gbps 以太网也会变得饱和。

幸运的是，还有另一条出路可以处理不断增长的负载：即交换式以太网。这种系统的核心是一个交换机（switch），它包含一块连接所有端口的高速背板，如图 4-17（b）所示。从外面看，交换机很像集线器。它们都是一个盒子，通常拥有 4～48 个端口，每个端口都有一个标准的 RJ-45 连接器用来连接双绞线电缆。每根电缆把交换机或者集线器与一台计算机连接，如图 4-18 所示。交换机具有集线器同样的优点。通过简单的插入或者拔出电缆就能完成增加或者删除一台机器，而且由于片状电缆或者端口通常只影响到一台机器，因此大多数错误都很容易被发现。这种配置模式仍然存在一个共享组件出现故障的问题，即交换机本身的故障；如果所有站都失去了网络连接，则 IT 人员知道该怎么解决这个问题：更换整个交换机。

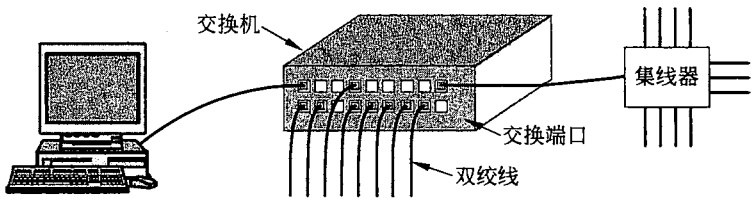


图 4-18　一个以太网交换机

然而，进入交换机内部就能看到某些与集线器完全不同的事情。交换机只把帧输出到该帧想去的端口。当交换机端口接收到来自某个站的以太网帧，它就检查该帧的以太网地址，确定该帧前往的目的地端口。这一步要求交换机能够知道端口对应哪个地址，这个过程我们放在 4.8 节学习了交换机之间互连的一般情况后再来讨论。现在，我们只假设交换机知道帧的目标端口。交换机把帧通过它的高速背板传送到目标端口。通常背板的运行速度高达许多个 Gbps，并且使用专用的协议。这些交换机专用协议无须标准化，因为它们完全隐藏在交换机内部；然后，目标端口在通往目标站的双绞线上传输该帧，没有任何其他端口知道这个帧的存在。

如果同时有多个站或者端口都要传送数据，会发生什么情况？在这点上，交换机再一次体现了与集线器的不同。在集线器中，所有站都位于同一个**冲突域**（collision domain），它们必须使用 CSMA/CD 算法来调度各自的传输。在交换机中，每个端口有自己独立的冲突域。通常情况下，电缆是全双工的，站和端口可以同时往电缆上发送帧，根本无须担心其他站或者端口。现在冲突不可能发生，因而 CSMA/CD 也就不需要了。然后，如果电缆是半双工的，则站和端口必须以通常的 CSMA/CD 方式竞争传输。

交换机的性能优于集线器有两方面的原因。首先，由于没有冲突，容量的使用更为有效。其次，也是更重要的，有了交换机可以同时发送多个帧（由不同的站发出）。这些帧到达交换机端口并穿过交换机背板输出到适当的端口。然而，由于两帧可能在同一时间去往同一个输出端口，交换机必须有缓冲，以便它暂时把输入帧排入队列直到帧被传输到输出端口。总体而言，这些改进获得了较大的性能改进，而这些改进手段对于集线器来说又是不可能做到的。系统总吞吐量通常可以提高一个数量级，主要取决于端口数目和流量模式。

帧被发送到输出端口还有利于安全。大多数 LAN 接口都支持**混杂模式**（promiscuous mode），这个模式下所有的帧都被发到每台计算机，而不只是那些它寻址的机器。每个连到集线器上的计算机能看到其他所有计算机之间的流量。间谍和好管闲事者最喜欢这一功能了。有了交换机，流量只被转发到它的目的端口。这种限制提供了更好的隔离措施，使得流量不会轻易泄露而落入坏人之手。不过，如果真正需要安全，最好还是对流量实行加密。

由于交换机只是希望每个输入端口出现的是标准以太网帧，就有可能把一部分端口用作集中器。在图 4-18 中，右上角的端口并没有连接一台计算机，而是连到一个 12 口的集线器。当帧到达集线器，它们按通常的方式竞争以太网，包括冲突和二进制指数后退。竞争成功的帧被传到集线器，继而到达交换机，在那里得到的处理就像任何其他输入帧一样。交换机不知道它们是经过竞争才到这里的。一旦进入交换机，它们就被发送到高速背板上的正确输出线。也有可能正确目的地是连接到集线器的一根线，在这种情况下，帧早已经被交付给目标机器，因此交换机就把它丢弃。集线器比交换机简单而且便宜，但由于交换机的价格在不断下降，集线器已经成为濒危物种。现代网络大量使用交换以太网。然而，传统的集线器依然存在。

4.3.5 快速以太网

在同一时间，交换机变得越来越受欢迎，10 Mbps 以太网备受压力。起初，10 Mbps 速率似乎完美无缺，如同线缆调制解调器在电话调制解调器用户看来像天堂一样。但是，新鲜感很快消失。作为帕金森定律（“工作总会填满原来计划用的时间”）的一种必然结果，似乎数据传输也总是会占用所有的可用带宽。

许多安装需要大量的带宽，因而许多 10 Mbps 的局域网被中继器、集线器和交换机连成了迷宫；虽然对网络管理员而言，有时他们觉得自己和泡泡糖和铁丝网挤在一起。但即使有以太网交换机，一台计算机的最大带宽还是受制于连接它到交换机端口的电缆。

在这种环境下，IEEE 于 1992 年重新召集 802.3 委员会，指示他们赶快提出一个快速 LAN 建议。其中一个建议是仍然保持 802.3 原来的面貌不变，但要运行得更快。另一个建议是完全重新设计 802.3，给予它更多的特性，比如支持实时流量和数字化语音，但是仍保留原来的名称（出于市场考虑的原因）。经过多次激烈争论之后，802.3 委员会决定仍然保留 802.3 原来的工作方式，但是让它运行得更快。这一策略将使标准化工作得以在技术革新之前完成，并且避免了全新设计带来的不可预见问题。新设计还将向后兼容现有的以太网局域网。在这种情况下那些提议遭到否决的人如同自尊的计算机工业界习惯做法那样：他们跺着脚形成了自己的委员会并标准化他们提议的局域网（最终作为 802.12）。但他们还

是以失败告终，草草收场。

这项工作很快就完成了（按照标准委员会的规范），其结果就是 802.3u，于 1995 年被 IEEE 正式批准。技术上，802.3u 并不是一个新标准，而是原有 802.3 标准的一份补充（因而也加强了它的向后兼容性）。这种策略被采用了许多次。因为实际上大家都称它为**快速以太网**（fast Ethernet），而不是 802.3u，所以我们也将使用这样的叫法。

快速以太网的基本思想非常简单：保留原来的帧格式、接口和过程规则，只是将比特时间从 100 纳秒降低到 10 纳秒。技术上，它可以照搬 10 Mbps 的经典以太网，只要将电缆的最大长度降低到十分之一，以便及时检测冲突。然而，由于双绞线具有压倒性的优势，所以快速以太网基本上完全基于这种设计。因此，所有的快速以太网系统也使用集线器和交换机；而不允许使用带插入式分接头或者 BNC 连接器的多支路电缆。

然而，还有其他一些选择仍然要确定下来，其中最重要的是支持什么样的电缆类型。一种观点是 3 类双绞线，其理由是几乎西方国家的每个办公室都有至少 4 组 3 类（或更好的）双绞线，从办公室连接到 100 米以内的电话接线柜。有时候会有两条这样的电缆。因此，若采用 3 类双绞线，则无须重新布线就有可能在桌面计算机使用快速以太网，这对于许多组织来说具有极大的好处。

使用 3 类双绞线的主要缺点是它不能够在 100 米长的电缆上承载 100 Mbps 的信号，而这是 10 Mbps 集线器规定的从计算机到集线器的最大距离。相反，5 类双绞线可以很容易地传输 100 米，光纤可以走得更远。最后折中的选择是允许所有这三种可能介质，如图 4-19 所示，但是，对于 3 类双绞线的方案，需要增加所需的额外承载能力。

名称	线缆	最大长度	优点
100Base-T4	双绞线	100 米	可用3类UTP
100Base-TX	双绞线	100 米	全双工速率100 Mbps(5类UTP)
100Base-FX	光纤	2000 米	全双工速率100 Mbps，距离长

图 4-19　最初的快速以太线缆

3 类 UTP 方案称为 100Base-T4，它使用了 25 MHz 的信令速度，比标准以太网的 20 MHz 仅仅快了 25%（记住，2.5 节中讨论的曼彻斯特编码中，对于 10 Mbps 中的每个比特要求两个时钟周期）。然而，为了达到所要求的比特率，100Base-T4 要求 4 对双绞线。其中一对总是去往集线器，另一对总是来自集线器，剩余的两对则动态切换到当前的传输方向。为了从传输方向上的三对双绞线上获得 100 Mbps，每对双绞线上使用了一种相当复杂的编码方案。该方案涉及用三个不同电压等级来发送三元数字。这个方案不太可能赢得任何奖项，我们将跳过细节。然而，由于过去几十年来标准的电话线每根电缆中都有 4 对双绞线，所以大多数办公室都能够使用已有的布线。当然，这也意味着要放弃你的办公室电话。但是，相对于能获得快速的电子邮件所付出的这点代价应该算不了什么。

随着许多办公大楼重新布线采用了 5 类 UTP，100Base-T4 也随之落下了帷幕。使用 5 类 UTP 的 100Base-TX 以太网很快占领了市场。因为 5 类双绞线可以处理 125 MHz 的时钟速率，所以这种设计要简单得多。每个站只用到两对双绞线，一对用于发送信号到集线器，另一对用于从集线器接收信号。这里没有使用直接的二进制编码（即 NRZ），也没有采用曼彻斯特编码，相反，它使用了一种我们在 2.5 节中描述过的 4B/5B 编码方案。4 个数据

位被编码成5个信号比特，并以125 MHz速率发送，可提供100 Mbps的数据率。这种方案很简单，但具有保持同步所需的足够跳变，对线缆带宽的使用相对很好。100Base-TX系统是全双工的；站可以在一对双绞线上以100 Mbps速率发送数据，同时也可以在另一对双绞线上以100 Mbps速率接收数据。

最后一种选择方案是100Base-FX，它使用两根多模光纤，每个方向用一根；所以它可以进行全双工操作，同时在每个方向上以100 Mbps速率发送。这种设置下，站和交换机之间的距离可以达到2千米。

快速以太网允许集线器和交换机相互之间互连。为了确保CSMA/CD算法继续工作，为了把网络传输速率从10 Mbps提高到100 Mbps，必须保持最小帧长和最大电缆长度之间的关系。所以，要么按比例把最小帧为64字节大小的限制往上提，要么把2500米的最大电缆长度降下来。最简单的选择是把任意两个站之间的最大距离降10倍，因为100米电缆的集线器早已可用。然而，2千米100Base-FX的线缆对于正常的以太网冲突算法来说过长。相反，这些线缆必须被连接到一个交换机，并工作在全双工模式下，因此才没有冲突。

用户很快就开始部署了快速以太网，但他们并不打算扔掉老式计算机上的10 Mbps以太网卡。因此，几乎所有的快速以太网交换机可处理10 Mbps和100 Mbps的混合情况。为了便于升级，标准本身提供了一个称为**自动协商**（autonegotiation）的机制，允许两个站自动协商最佳速度（10 Mbps或100 Mbps）和双工模式（半双工或全双工）。这个机制大部分时间运作良好，但已经发现会导致双工不匹配问题，即链路的一端启动了自动协商，但另一端没有启动，并设置了全双工模式（Shalunov和Carlson，2005）。大多数以太网产品都使用该功能来配置自己。

4.3.6 千兆以太网

快速以太网标准的墨迹未干，802委员会就开始制定一项更快的快速以太网，称为**千兆以太网**（gigabit Ethernet）。1999年IEEE批准了最普遍的形式802.3ab。下面我们将讨论千兆以太网的一些关键特性。更多信息请参考（Spurgeon，2000）。

对于千兆位以太网，委员会的目标基本上与快速以太网的目标相同：增加十倍的性能，并同时保持与所有现存以太网标准的兼容。特别是，千兆以太网必须提供单播和广播的无确认数据报服务，使用相同的48位地址方案，保持相同的帧格式，包括最小和最大帧尺寸要求。最终发布的标准符合所有这些目标。

与快速以太网一样，千兆以太网使用点到点链路。最简单的配置，如图4-20（a）所示，两台计算机直接相连。然而，较常见的情况下，使用一个交换机或集线器连接多台计算机，和额外的交换机或集线器，如图4-20（b）所示。在这两种配置下，每根单独的以太网电缆上正好有两个设备，一个也不多一个也不少。

千兆以太网支持两种不同的操作模式：全双工模式和半双工模式。“正常”模式是全双工模式，它允许两个方向上的流量同时进行。这种模式适用于一台中心交换机将周围的计算机（或者其他的交换机）连接起来。在这种配置下，所有的线路都具有缓存能力，所以每台计算机或者交换机在任何时候都可以自由地发送帧。发送方在发送之前不必侦听信道才能确定是否有别人在使用信道，因为竞争不可能发生。在一台计算机与交换机之间的连

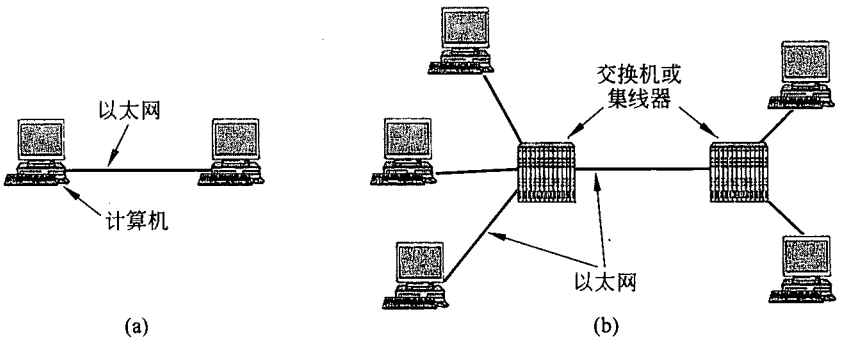


图　4-20
（a）一个包含两个站的以太网；（b）一个有多个站的以太网

线上，唯一可能的发送方是计算机，即使交换机正在给计算机发送数据，计算机的传输操作也会成功（因为线路是全双工的）。由于这里不可能存在竞争，所以不需要使用 CSMA/CD 协议，因此线缆的最大长度由信号强度来决定，而不是由突发噪声在最差情况下传回到发送方所需的时间来决定。交换机可以自由地混合和匹配各种速度。就如同快速以太网一样，千兆以太网也支持自动配置特性，现在的选择有：10、100 和 1000 Mbps。

另一种操作模式是半双工模式，当计算机被连接到一个集线器而不是交换机的时候，就会用到这种模式。集线器无法将入境帧缓存，相反，它在内部用电子的方式将所有这些线路串接起来，模拟经典以太网中所使用的多路分支电缆。在这种模式下，冲突有可能发生，所以要求使用标准的 CSMA/CD 协议。因为一个 64 字节帧（再短也是允许）现在的传输速度是经典以太网的 100 倍快，所以最大的线缆长度下降了 100 倍，或 25 米，这样才能维持以太网的本质特性，即在最差情况下，当突发噪声返回时发送方仍在传送数据。对于一根 2500 米长的线缆，发送方以 1 Gbps 速率发送一个 64 字节帧，在该帧尚未到达距离另一端的十分之一路程之前就已经发送完毕，更不用说到达另一端再返回了。

考虑到 25 米的长度限制太痛苦了，因此在标准中加入了两个特性，使得最大线缆长度增加到 200 米，这个条件对于大多数办公室都是能满足的。第一个特性称为**载波扩充**（carrier extension），它的本质是让硬件在普通的帧后面增加一些填充位，将帧的长度扩充到 512 字节。由于这些填充位是由发送方硬件加进来，并且由接收方硬件删除掉，所以软件对此并不知情，这意味着现有软件无须作任何改变。当然，对于用户数据只有 46 字节（即 64 字节帧的有效载荷字段）的情形来说，使用 512 字节之后线路的效率只有 9%。

第二个特性称为**帧突发**（frame bursting），允许发送方将多个待发送帧级连在一起，一次传输出去。如果级联起来的整个突发仍然小于 512 个字节，则硬件会再次对它进行填充。如果有足够的帧在等待传输，则这种方案非常高效，应该优先于载波扩充。

客观地讲，你很难想象一个组织会购买装有千兆以太网卡的现代计算机，然后将计算机连接到一个老式集线器上来模拟有冲突的经典以太网。千兆以太网接口和交换机曾经非常昂贵，但它们的价格随着销量的上升下降很快。诚然，在计算机领域中，向后兼容性至关重要，所以 802.3z 委员会要求做到这一点。今天，大多数计算机都带有具备 10、100 和 1000 Mbps 运行能力的以太网卡，所以兼容性就不再是个问题。

千兆以太网既支持铜线，也支持光纤，如图 4-21 所示。以 1 Gbps 或者接近于 1 Gbps

的速率传输信号，要求每纳秒编码并发出一个比特。这一招最初在短程屏蔽铜电缆上实现（1000Base-CX 版本）。对于光纤而言，允许使用两个波长，因而导致两个不同的版本：0.85 微米（短波，1000Base-SX）和 1.3 微米（长波，1000Base-LX）。

名称	线缆	最大长度	优点
1000Base-SX	光纤	550 米	多模光纤(50、62.5 微米)
1000Base-LX	光纤	5000 米	单模光纤(10 微米)或多模光纤(50、62.5微米)
1000Base-CX	2对STP	25 米	屏蔽双绞线
1000Base-T	2对UTP	100 米	标准5类UTP

图 4-21 千兆以太网线缆

短波信号可以用便宜的 LED 获得。它可用于多模光纤，对建筑物内的连接非常有用，因为它可以运行在 500 米的 50 微米光纤上。长波信号需要昂贵的激光器。另一方面，与单模光纤（10 微米）结合，线缆长度可长达 5 千米。这个限制允许长距离连接建筑物，比如作为一个专门的点到点链路用在校园骨干网中。标准后来甚至允许更长的单模光纤。

要在千兆以太网的各版本上发送比特，我们在 2.5 节描述过的 8 B/10 B 编码借用了一项称为**光纤信道**（Fibre Channel）的网络技术。这种编码方案将 8 个数据位编码为 10 个比特的码字发送到电缆或者光纤上，因而被命名为 8 B/10 B。码字的选择使得它们必须是平衡的（即具有相同数目的 0 和 1），具备时钟恢复足够的跳变。采用 NRZ 编码技术发送的比特比不编码发送比特所需的信号带宽要多 25%，相比带宽增幅达 100%的曼彻斯特编码则有很大的改善。

然而，所有这些编码选项需要新的铜缆或光缆才能支持更快的信号。早已为快速以太网而安装的大量 5 类非屏蔽双绞线并没有被采用。不到一年，1000Base-T 标准填补了这一空白，从此以后它一直是千兆以太网的最流行形式。人们显然不喜欢重新布线他们的建筑物。

若要使得以太网在 5 类双绞线上以 1000 Mbps 速率运行，需要更复杂的信号。为此，电缆中的所有四对双绞线都要被派上用场。每一对可同时在两个方向上传输，利用数字信号处理技术来区分信号。在每根线上，5 个电压级别携带 2 个比特，信号速率为 125 M 符号/秒。把比特映射成符号不是件容易的事，它涉及扰码和纠错码，前者为生成足够跳变而作，后者把 4 个值嵌入到 5 个信号级别。

1 Gbps 的速度非常快。例如，如果接收方正忙于其他事情，即使只有 1 毫秒的时间，因为没有清空某条线路上的输入缓冲区，在这么短的时间内最多可累积 1953 个帧。还有，当千兆以太网上的一台计算机沿着线路给另一台位于经典以太网中的计算机发送数据，缓冲区极有可能发生溢出。作为这两种情形的一个必然结果，千兆以太网必须支持流量控制。在流量控制机制下，一端的机器给另一端机器发送一个特殊的控制帧，告知对方暂停一段时间。这些暂停（PAUSE）控制帧是普通的以太帧，类型字段设置为 0x8808。暂停时间是最小帧时的整数倍。对于千兆以太网，时间单位是 512 纳秒，允许的最大暂停时间是 33.6 毫秒。

千兆以太网还引入了另一个扩展。巨型帧（Jumbo frame）允许帧的长度超过 1500 字节，通常高达 9 KB。这个扩展是其专有的，没有得到标准的认可。因为如果使用巨型帧，

则无法与以太网的早期版本兼容，但大多数厂商都支持这个扩展。理由是 1500 字节是千兆位速度下的很短单位。处理更大块的信息可以降低帧的速率，因而与之有关的处理时间也有所下降，比如中断处理器告诉它到达了一帧，或拆分与重组一个太长的消息以便适应一个以太网帧。

4.3.7 万兆以太网

一旦千兆以太网被标准化，802 委员会的委员们觉得无聊，他们想回去工作。IEEE 告诉他们开始**万兆以太网**（10 gigabit Ethernet）的标准化进程。这项工作遵循了许多以前以太网标准的模式，2002 年首次发布了光纤标准，2004 年发布了屏蔽铜电缆标准，紧接着 2006 年发布了铜双绞线标准。

万兆以太网真正的速度惊人，比原先的以太网快 1000 倍。它可能会用在哪里？答案是数据中心和交换局内部，可以用它们来连接高端路由器、交换机和服务器；除此之外，还可以用作端局之间的长途高带宽中继线，这些端局使整个城域网得以基于以太网和光纤来构建。长距离的连接使用光纤，而短距离的连接可以使用铜缆或光纤。

万兆以太网的所有版本只支持全双工操作。CSMA/CD 不再属于设计的一部分，标准的重点在于以超高速率运行的物理层细节。兼容性依然重要；虽然如此，万兆以太网接口能自动协商，并能降低到由线路两端同时支持的最高速度。

万兆以太网的主要种类如图 4-22 所列。0.85 微米（短）波长的多模光纤用于中距离，1.3 微米（长）和 1.5 微米（扩展）的单模光纤用于长距离。10GBase-ER 运行距离可达 40 千米，使其适用于广域网场合。所有这些版本发送的一串信息流，都是先通过绕码数据位，然后再经过 64 B/66 B 编码生成。这种编码比 8 B/10 B 码的开销更少。

名称	线缆	最大长度	优点
10GBase-SR	光纤	最多300米	多模光纤(0.85微米)
10GBase-LR	光纤	10千米	单模光纤(1.3微米)
10GBase-ER	光纤	40千米	单模光纤(1.5微米)
10GBase-CX4	4对双轴	15米	双轴铜缆
10GBase-T	4对UTP	100米	6a类UTP

图 4-22　万兆以太网布线

第一个铜版本由 10GBase-CX4 定义，采用了四对双轴铜电缆。每对使用 8 B/10 B 编码，以 3.125 G 符号/秒运行，提供了 10 Gbps 的数据率。这个版本比光纤便宜，很早就流向市场，但它是否会被运行更多金属双绞线上的 10 千兆以太网打败仍有待观察。

10GBase-T 是使用 UTP 电缆的版本。虽然要求 6a 类布线，为了更短的运行可以使用较低类别的双绞线（包括 5 类），以便重用已安装的电缆。毫不奇怪，为了在双绞线上达到 10 Gbps，其物理层相当复杂。我们只给出一些较高层次的细节。四对双绞线的每一对可用来在两个方向上以 2500 Mbps 速率发送。要达到这个速率必须用到 800 M 符号/秒的符号速率，而符号又使用了 16 个电压等级。这些符号的产生首先对数据扰码，然后用**低密度奇偶校验**（LDPC，Low Density Parity Check）码保护，再进一步进行纠错编码。

万兆以太网还在市场上摇晃，802.3 委员会便已向前挺进。截至 2007 年底，IEEE 创建

了一个小组对40 Gbps和100 Gbps的以太网进行标准化。此次升级将使以太网有能力去竞争非常高性能的设施，包括骨干网络中的长距离和设备背板上的短程连接。标准还没有完成，但专利产品已经上市。

4.3.8 以太网回顾

以太网已经发展30多年了，在发展过程中还没有出现过真正有实力的竞争者，所以它可能还会持续发展很多年。对于CPU结构、操作系统或者编程语言，很少有哪一个能独占鳌头30年不倒。很明显，以太网已经走上了一条正确道路。为什么是这样的？

以太网具有如此强大生命力的最主要理由可能是它的简单性和灵活性。实际上，简单性带来了可靠、廉价，以及易维护等特性。一旦集线器和交换机体系结构被采纳后，失败就极为罕见了。人们犹豫着是否要替换掉长期以来工作得非常出色的事物，尤其是当他们得知在计算机工业界有许多非常糟糕的事情，许多所谓的“升级”比原来的还要更糟糕后，基本上注定了以太网的不可撼动地位。

简单性也可以理解为造价低廉。作为硬件组件的双绞线相对来说非常便宜。在过渡时可能有点昂贵，比如引入新千兆以太网NIC或者交换机，但这些仅仅是一个构建良好网络的补充（不是完全替换），并且随着销量上升价格会很快回落。

以太网非常易于维护。不需要安装特殊的软件（驱动程序除外），也不需要管理配置表。而且，增加新的主机也非常简单，只要将它们接入即可。

另外，TCP/IP使得以太网很容易实行互联，而TCP/IP已经占据了主导地位。IP是一个无连接协议，所以它非常适合于以太网，因为以太网也是无连接的。IP不太适合面向连接，比如ATM。这种不一致性无疑削弱了ATM的发展机会。

最后，或许是最重要的，以太网已经在多个关键的方面取得了显著的进展。从速率来看，以太网已经提升了几个数量级；从结构来看，集线器和交换机被引了进来；但是这些变化并不要求软件也跟着发生变化，而且通常允许重用已有的线缆。如果网络销售人员在展示一项大型网络装置时这样说“我为你们带来了一种新奇的网络。你们所需要做的事情只是丢弃所有的硬件，并且重写所有的软件”，那么，他一定有问题了。

有许多你可能甚至没有听说过的技术在推出时的速度比以太网快。这份名单上的技术包括ATM、**光纤分布式数据接口**（FDDI，Fiber Distributed Data Interface）和光纤信道（Fibre Channel[1]）。后两个都是基于光纤局域网的双环结构，并且都不与以太网兼容。但没有一个是成功的。它们太复杂了，导致芯片异常复杂，而且价格居高不下。这里我们应该学到的一课是“保持简单，否则就傻”。最后，以太网赶上并超越了它们，在速度方面以太网借用了它们的一些技术，比如从FDDI那里借用了4B/5B编码，从光纤信道那里借用了8B/10B编码。这样，它们仅存的优势也悉数失去，最终悄然退出舞台或沦落成特殊的配角地位。

看上去以太网还将在一段时间内继续扩大它的应用。万兆以太网已经摆脱了CSMA/CD的距离限制。很多努力正在致力于研究**电信级以太网**（carrier-grade Ethernet），以便网络提供商为它们的城域网和广域网客户提供基于以太网的服务（Fouli和Maler，

1 它称为“Fibre Channel”而不是“Fiber Channel”，因为文档编辑是大不列颠人。

2009）。这种应用在长距离光纤上运载太网帧，并且要求更好的管理功能，以便有助于运营商提供可靠优质的服务。超高速网络也将有用武之地，它们可用在大型路由器或服务器的背板连接网络组件中。这些用途都是除了在办公室的计算机之间发送数据帧之外的额外应用。

4.4　无线局域网

无线 LAN 越来越普及，家庭、办公室、咖啡厅、图书馆、机场、动物园等公共场所都有相应的设施，通过它们可以把计算机、PDA 和智能手机连接到 Internet。无线局域网也可用来使得附近的两台或多台计算机直接进行通信而无须接入 Internet。

无线局域网的主要标准是 802.11。我们在 1.5.3 给出了一些背景信息。现在是对其技术一探究竟的时候了。在下面的章节中，我们将考察 802.11 的协议栈、物理层无线传输技术、MAC 子层协议、帧结构以及所提供的服务。如需有关 802.11 的更多信息，请参阅（Gast，2005）。为了获得真理，咨询已发表的标准，即 IEEE 802.11-2007。

4.4.1　802.11 体系结构和协议栈

802.11 网络的使用模式有两种。最普遍使用的把客户端（比如笔记本电脑和智能手机客户端），连接到另一个网络（比如公司内联网或 Internet）。这种使用模式如图 4-23（a）所示。在有架构模式下，每个客户端与一个接入点（AP，Access Point）关联，该接入点又与其他网络连接。客户端发送和接收数据包都要通过 AP 进行。几个接入点可通过一个称为**分布式系统**（distribution system）的有线网络连接在一起，形成一个扩展的 802.11 网络。在这种情况下，客户端可以通过它们的接入点向其他客户端发送帧。

另一种模式，如图 4-23（b）所示，是一种**自组织网络**（ad hoc network）。这种模式下的网络由一组相互关联的计算机组成，它们相互之间可以直接向对方发送帧。这里没有接入点。由于 Internet 接入是无线的杀手级应用，自组织网络并没有那么受欢迎。

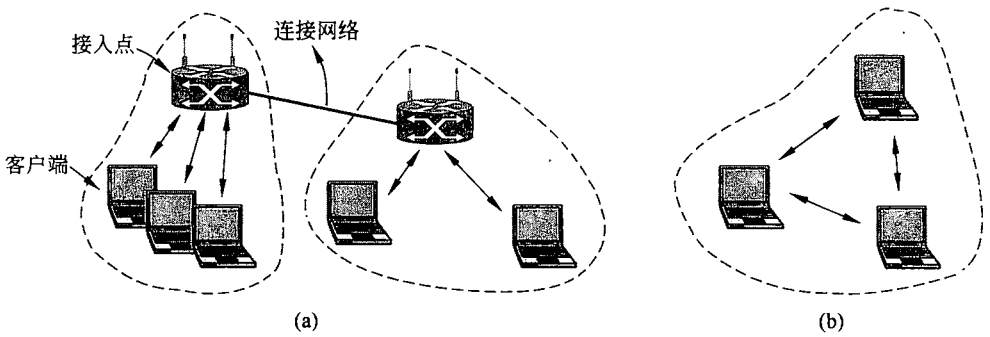


图 4-23　802.11 体系结构
（a）有架构模式；（b）自组织模式

现在我们来看协议。所有的 802 协议（包括 802.11 和以太网）都有某些结构上的共性。图 4-24 给出了 802.11 协议栈的组成部分。客户端和 AP 的协议栈相同。物理层对应于 OSI

的物理层，但所有802协议的数据链路层分为两个或更多个子层。在802.11中，介质访问控制（MAC，Medium Access Control）子层决定如何分配信道，也就是说下一个谁可以发送。在它上方的是**逻辑链路控制**（LLC，Logical Link Control）子层，它的工作是隐藏802系列协议之间的差异，使它们在网络层看来并无差别。这应该是一个非常重大的责任，但这些日子以来LLC一直作为一个承上启下的子层，标识802.11帧内携带的协议（例如，IP）。自从802.11在1997年首次出现以后，随着它的不断演变，在物理层已经添加了几种传输技术。最初采用的两种技术：电视遥控器方式的红外和2.4 GHz频段的跳频方式现在都已不复存在。最初的第三个技术：在2.4 GHz频段的1或2 Mbps直接序列扩频被扩展为运行速率高达11 Mbps，并很快一炮走红。它就是所谓的802.11b。

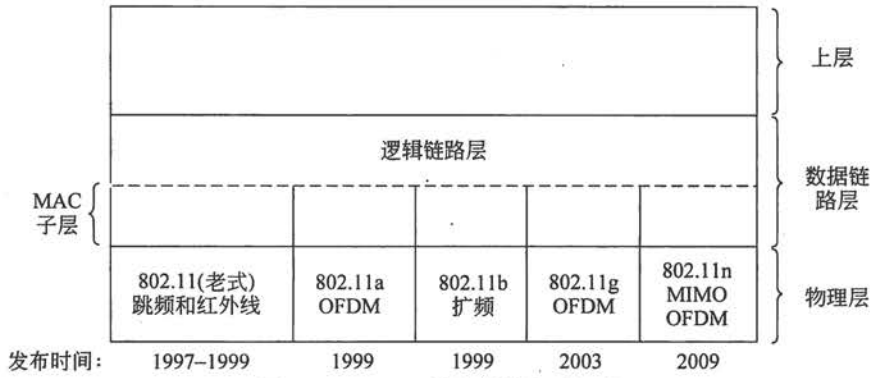


图4-24 802.11协议栈的组成部分

为了给无线迷们一个更快的速度提升，基于**正交频分复用**（OFDM，Orthogonal Frequency Division Multiplexing）编码方案的新传输技术分别在1999年和2003年被引入进来，我们在2.5.3节介绍过OFDM。首先是802.11a，它使用了5 GHz这个不同的频段；第二是802.11g，它坚守在2.4 GHz，并保持兼容性。两者都提供高达54 Mbps的数据率。

最近，同时使用多个天线的技术被引入到无线局域网，使得发送器和接收器的速度得到突飞猛进，相应的标准终于在2009年10月定稿为802.11n。有了4根天线和更宽的信道，802.11标准现在定义的速率达到了令人吃惊的600 Mbps。

现在，我们简要地介绍这些传输技术。然而，我们将只包括那些还在使用的传输技术，跳过一些传统的802.11传输方法。技术上，这些内容同属于物理层，应在第2章讨论；但由于它们与普通局域网密切相关，尤其是802.11局域网，因此我们在这里讨论。

4.4.2 802.11物理层

每一种传输技术都有可能使一个站从空中发送MAC帧到另一个站。然而，这些传输技术彼此之间还是有区别的，主要体现在使用的技术和可以达到的速度上。对这些技术的详细讨论远远超出了本书的范围，但在下面的讨论中我们还是会用若干语句涉及在2.5节描述的内容，并且为有兴趣的读者提供从其他地方搜索更多信息的主要关键术语。

所有的802.11技术都使用短程无线电传输信号，通常在2.4 GHz或5 GHz频段，这两个频段属于ISM，我们在2.3.3节中已有描述。这些频段的最大优点是无须许可证，任何人

只要愿意遵守一些限定都可免费使用，这些限定包括发射器的辐射功率不能超过 1 瓦（对于一般无线局域网来说 50 毫瓦已经很高了）。不幸的是，这一事实也被车库门制造商、无绳电话、微波炉以及无数其他设备制造商获知，所有这些设备都与笔记本电脑竞争相同的频谱。相比 5 GHz 频带，2.4 GHz 频段已经非常拥挤，因此对某些应用程序而言 5 GHz 频段更好，即使它由于较高的频率而传输范围较小。

所有的传输方法都定义了多个速率。设计多速率的主要想法在于根据当前的条件采用不同的速度。如果无线信号较弱，则采用较低的速率；如果信号很清晰，可使用最高速率。这种调整速率的方法就是**速率自适应**（rate adaptation）。由于速率变化范围可多达 10 倍，甚至更多，因此一个优秀的速率自适应算法对网络的良好性能至关重要。当然，因为它不需要网络节点之间的互操作，因此标准没有规定如何自适应调整速率。

我们考查的第一个传输方法是 802.11b。它是一种扩展频谱的方法，支持 1、2、5.5 和 11 Mbps 速率。但实际上运行速率几乎总是在 11 Mbps。它类似于我们在 2.5 节考查过的 CDMA 系统，但只有一个扩频码被所有用户共享。扩展频谱是为了满足 FCC 的要求，即把信号能量扩展到 ISM 频谱。802.11b 使用的扩展序列是一个巴克序列（Barker sequence）。除非序列完全一致，巴克序列的自相关性很低。这个属性允许接收器锁定一个传输的开始。若以 1 Mbps 的速度发送，则使用巴克序列和 BPSK 调制技术每 11 个码片发送 1 个比特，码片传输速率为 11M 码片/秒；若要以 2 Mbps 的速率发送，则使用巴克序列和 QPSK 调制技术，每 11 个码片发送 2 个比特。再高速率采用的技术就不同了。这些速率使用一种称为**补码键控**（CCK，Complementary Code Keying）技术来编码，而不是采用巴克序列。5.5 Mbps 速率每 8 个码片发送 4 个比特，11 Mbps 速率每 8 个码片发送 8 个比特。

接下来我们看 802.11a，其在 5 GHz 的 ISM 频段支持的速率可高达 54 Mbps。你或许已经预料 802.11a 应该比 802.11b 早出现，但事实并非如此。虽然 802.11a 小组先成立，但 802.11b 标准先获得批准，并且其产品投放市场的时机也远远领先于 802.11a 产品，这里面的原因部分在于 5 GHz 频段操作上的困难。

因为 OFDM 使用频谱更有效率，并且能抵抗多径等无线信号的衰减，因此 802.11a 方案**基于正交频分复用**（OFDM，Orthogonal Frequency Division Multiplexing）。比特在 52 个并行的子载波上发送，48 个子载波携带数据和 4 个子载波用于同步控制。每个符号持续 4 微秒，可发送 1、2、4 或 6 个比特。比特首先采用二进制卷积码进行纠错编码，因此只有 1/2、2/3 或 3/4 比特没有冗余。采用不同的组合，802.11 可以运行在 8 个不同的速率上，从 6～54 Mbps 不等。这些速率明显快于 802.11b 的速率，而且 5 GHz 频段上的干扰少。然而，802.11b 的覆盖范围约为大于 802.11a 的 7 倍多，在许多情况下这是更重要的考量。

即使拥有更大的范围，802.11b 用户也没有让这颗冉冉升起的新星赢得速度冠军的意图。幸运的是，2002 年 5 月，FCC 放弃了其制定的长期规定，即要求所有无线通信设备在美国 ISM 频带的操作必须使用扩频，所以 802.11g 开展了相应的工作，最终于 2003 年获得 IEEE 批准。它复制了 802.11a 的 OFDM 调制方法，但工作在狭窄的 2.4 GHz ISM 频段，与 802.11b 一起工作在同一频段。它提供了与 802.11a 相同的速率（6～54 Mbps），以及显而易见地与任何附近的 802.11b 设备兼容。为了不让客户对所有这些不同的选择产生混淆，常见的无线局域网产品在单一的网卡上同时支持 802.11a/b/g。

IEEE 委员会并没有就此满足而停滞不前，他们又开始开展高吞吐量物理层的工作，这个物理层称为 802.11n。这个标准在 2009 年获得批准。802.11n 的目标是去掉所有无线开销后吞吐量至少达到 100 Mbps。这个目标要求原始速率至少要增加 4 倍。为了做到这一点，委员会把信道加宽一倍，从 20 MHz 扩大到 40 MHz，并且允许同时发送一组帧来降低成帧的开销。然而，更重要的是，802.11n 在同一时间可以使用 4 根天线发送四个信息流。这些流的信号虽然在接收端会相互干扰，但可以通过使用**多入多出**（MIMO，Multiple Input Multiple Output）通信技术把它们分离开来。多天线的使用带来了速度上的极大提升，但没有带来更大的覆盖范围和更高的可靠性。像 OFDM 一样，MIMO 也是那些巧妙通信想法中的一种，人类的聪明才智正在改变着无线网络的设计，我们乐意在未来听到更多有关这方面的想法。对于 802.11 多天线的简要介绍请参考（Halperin 等，2010）。

4.4.3 802.11 MAC 子层协议

现在让我们从电气工程领域返回到计算机科学领域。802.11 MAC 子层协议与以太网有很大的不同，这种本质上的差异性来自于无线通信的两大因素。

首先，无线电几乎总是半双工的，这意味着它们不能在一个频率上传输的同时侦听该频率上的突发噪声。接收到的信号很容易变得比发射信号弱上一百万倍，因此它无法在同一时间听到这么微弱的信号。而在以太网中，一个站只要等到介质空闲，然后开始传输。如果它没有在发送的前 64 个字节期间收到返回的突发噪声，则几乎可以肯定帧能正确地传送出去。但对于无线介质，这种冲突检测机制根本不起作用。

相反，802.11 试图避免冲突，采用的协议称为**带有冲突避免的** CSMA（CSMA/CA，CSMA with Collision Avoidance）。该协议在概念上类似于以太网的 CSMA/CD，在发送前侦听信道和检测到冲突后指数后退。然而，需要发送帧的站必须以随机后退开始（除非它最近没有用过信道，并且信道处于空闲状态），而且它不等待冲突的发生。在 OFDM 物理层情况下，后退选择的时间槽数范围在 0～15 之间。该站将等待，直到信道处于闲置状态。具体做法是：通过侦听确定在一个很短的时间内（这段时间称为 DIFS，我们在下面解释）没有信号；然后倒计数空闲时间槽，当有帧在发送时暂停该计数器；当计数器递减到 0，该站就发送自己的帧。如果帧发送成功，目标站立即发送一个短确认。如果没有收到确认，则可推断出传输发生了错误，无论是冲突或是其他什么错。在这种情况下，发送方要加倍后退选择的时间槽数，再重新试图发送。如此反复，连续像以太网那样以指数后退，直到成功发送帧或达到重传的最大次数。

图 4-25 给出了一个发送帧的时序例子。A 站首先发出一个帧。当 A 发送时，B 站和 C 站准备就绪发送。它们看到信道正忙，便等待它变成空闲。不久，A 收到一个确认，信道进入空闲状态。然而，不是两个站都发出一帧从而立即产生冲突，而是 B 站和 C 站都执行后退算法。C 站选择了一个较短的后退时间，因而先获得发送权。B 站侦听到 C 在使用信道时暂停自己的倒计时，并在 C 收到确认之后立即恢复倒计时。一旦 B 完成了后退，立即发送自己的帧。

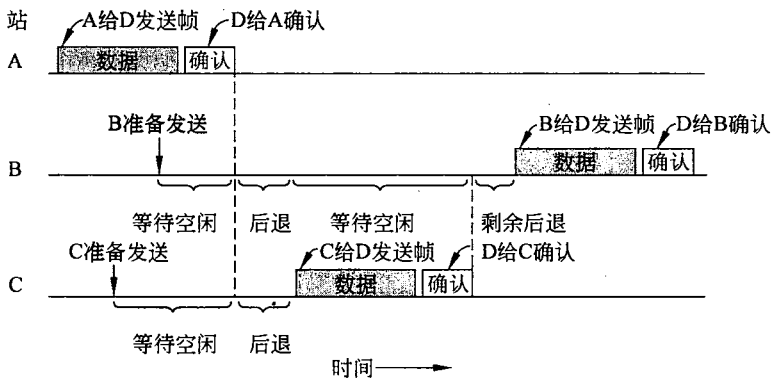


图 4-25　在 CSMA/CA 机制下发送帧

和以太网相比，这里有两个主要区别。首先，早期的后退有助于避免冲突。冲突避免在无线传输中非常重要，即使只发生一个冲突因为整个帧都被传输了出去，因此冲突的代价非常昂贵。其次，利用确认来推断是否发生冲突，因为冲突无法被检测出来。

这种操作模式称为**分布式协调功能**（DCF，Distributed Coordination Function），因为每个站都独立行事，没有任何一种中央控制机制。标准还包括一个可选的操作模式，称为**点协调功能**（PCF，Point Coordination Function）。在这种模式下，AP 控制自己覆盖范围内的一切活动，就像蜂窝通信中的基站一样。然而，实际使用中很少用到 PCF，因为通常没有办法阻止临近网络中的站发送竞争流量。

第二个问题在于不同站的传输范围可能有所不同。有线环境下，系统被设计成让所有站都可以听到对方。在射频传播的复杂环境下，这种状况并不适用于无线站。因此，类似前面提到的隐藏终端等问题就会出现，图 4-26（a）再次说明了隐藏终端问题。因为不是所有的站都在彼此的无线电广播范围内，因此蜂窝中一部分正在进行的传输无法被同一蜂窝中的其他地方收听到。在这个例子中，C 站正在给 B 站发送。如果 A 站侦听通道，它将听不到任何东西，因而错误地认为现在它可以开始给 B 站传输了。显然，这一决定将导致冲突的发生。

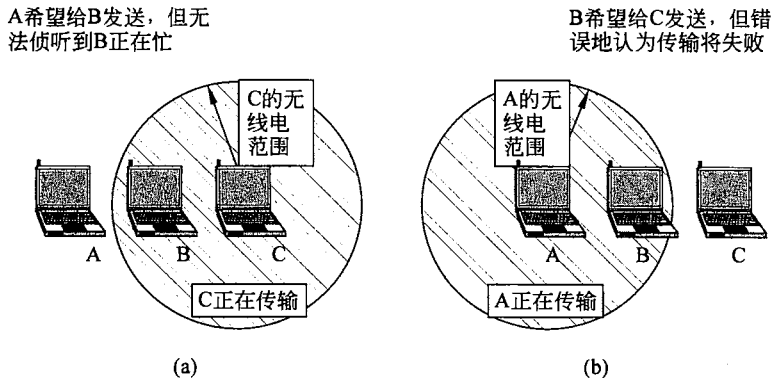


图　4-26
（a）隐藏终端问题；（b）暴露终端问题

相反的情况是暴露终端问题，如图 4-26（b）所示。在这里，B 想要给 C 发送，所以它去侦听信道。当 B 听到信道上有帧在传送，它便错误地认为可能无法发送到 C，即使事

实上 A 或许在给 D（未显示）传输。显然，这个决定浪费了一次传输机会。

为了减少究竟哪个站在发送的模糊不清，802.11 定义信道侦听包括物理侦听和虚拟侦听两部分。物理侦听只是简单地检查介质，看是否存在有效的信号；有了虚拟侦听，每个站可以保留一个信道何时要用的逻辑记录，这是通过跟踪**网络分配向量**（NAV，Network Allocation Vector）获得的。每个帧携带一个 NAV 字段，说明这个帧所属的一系列数据将传输多长时间。无意中听到这个帧的站就知道无论自己是否能够侦听到物理信号，由 NAV 所指出的时间段信道一定是繁忙的。例如，一个数据帧的 NAV 给出了发送一个确认所需要的时间。所有听到该数据帧的站将在发送确认期间推迟发送，而不管它们是否能听到确认的发送。

可选的 RTS/CTS 机制使用 NAV 来防止隐藏终端在同一时间发送。该机制如图 4-27 所示。在这个例子中，A 想给 B 发送，C 是 A 范围内的一个站（也有可能在 B 的范围内，但这并不重要）。D 在 B 范围内，但不在 A 的范围内。

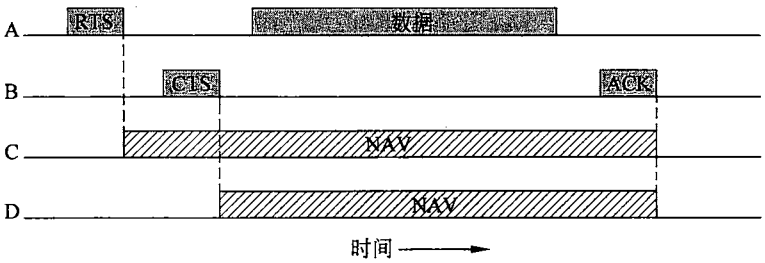


图 4-27 使用 CSMA/CA 的虚拟信道侦听

该协议开始于当 A 决定向 B 发送数据时。A 首先给 B 发送一个 RTS 帧，请求对方允许自己发送一个帧给它。如果 B 接收到这个请求，它就以 CTS 帧作为应答，表明信道被清除可以发送。一旦收到 CTS 帧，A 就发送数据帧，并启动一个 ACK 计时器。当正确的数据帧到达后，B 用一个 ACK 帧回复 A，完成此次交流。如果 A 的 ACK 计时器超时前，ACK 没有返回，则可视为发生了一个冲突，经过一次后退整个协议重新开始运行。

现在让我们从 C 和 D 的角度来看这次数据交流。C 在 A 的范围内，因此它可能会收到 RTS 帧。如果收到了，它就意识到很快有人要发送数据。从 RTS 请求帧提供的信息，可以估算出数据序列将需要传多长时间，包括最后的 ACK。因此，它采取了有利于所有人的做法，停止传输任何东西，直到此次数据交换完成。它通过更新自己的 NAV 记录表明信道正忙，如图 4-27 所示。D 无法听到 RTS，但它确实听到了 CTS，所以它也更新自己的 NAV。请注意，NAV 信号是不传输的，它们只是由站内部使用，提醒自己保留一定时间内的安静。

然而，尽管 RTS/CTS 在理论上听起来不错，但它却落入那些已被证明在实践中几乎没有价值的设计之列。为什么它在现实中很少被使用，有几个众所周知的原因。首先，它对短帧（替代 RTS 发送）或 AP（按照定义，大家都能彼此听到）一点好处都没有。对于其他情况，该机制只会降低操作速度。802.11 中的 RTS/CTS 与我们在 4.2 节看到的 MACA 还是有点不同，因为每个站都能听到 RTS 或 CTS，因而在此期间保持沉默以便 ACK 无冲突地通过。正因为如此，它无助于暴露终端问题的解决；正如 MACA 一样，只对隐藏终端有好处。大多数情况下隐藏终端很少，而且不管什么原因，CSMA/CA 通过后退发送失败

的站来缓解隐藏终端问题，使得传输更可能获得成功。

带有物理侦听和虚拟侦听的 CSMA/CA 是 802.11 协议的核心。然而，与之相关的几个其他机制也已经被开发出来。每种机制的开发都源自于实际运行的需要，所以我们也将简要地介绍它们。

首先我们需要了解的是可靠性。相对于有线网络，无线网络环境嘈杂，并且不可靠，这是因为相当大一部分要受到来自其他种类设备的干扰，如微波炉等这些也使用无须许可 ISM 频段的家用设备。使用确认和重传只能起到很少一点的帮助作用，因为当帧得以成功传输的概率很小时，再确认和重传也帮助不大。

增加传输成功概率所用的策略是降低传输速率。在一个给定的信噪比环境下，速度放慢可以使用更健壮的调制解调技术，帧就越有可能被正确接收。如果有太多的帧被丢失，站可降低速率。如果帧传输时损失很少，则站可以偶尔测试较高速率，看它是否应该采用较高的速率来传输帧。

另一种改善帧成功传输的策略是发送短帧。如果任何一位出错的概率为 p，那么一个 n 位长的帧被完全正确接收的概率为$(1-p)^n$。例如，对 $p=10^{-4}$，一个完整以太网帧（12 144 位）被正确接收的概率小于 30%。大多数帧都将被丢失。但是，如果帧只有三分之一长（4048 位），那么三分之二都将被正确接收。现在大多数帧都能得以通过，很少需要重传。

短帧可以通过降低来自网络层消息的最大尺寸来加以实现。另外，802.11 允许把帧拆分成更小的单元——称为**段**（fragment），每个段有自己的校验和。标准没有固定段的大小，但把它作为一个可以由 AP 调整的参数。这些段独立编号，使用停-等式协议（即发送方不能发送第 K+1 段，直到它已收到了第 K 段的确认）对其进行确认。一旦获得信道，可以突发多个段。这些段一个接着一个发送，两个段之间是确认（或许还要重传）直到整个帧被成功发送或达到最大允许的发送时间。NAV 机制保证了其他站在该帧的传输期间保持沉默，直到下一个确认（这个说法也值得斟酌），但另一种机制（见下文）用来允许突发多个段，并且发送期间不会有其他站发送。

第二个我们需要讨论的是节省电源。对移动无线设备来说，电池的寿命始终是个问题。802.11 标准非常重视电源管理问题，因此客户没有信息需要发送或接收时不应该浪费能量。

节能的基本机制建立在**信标帧**（beacon frames）基础上。信标帧由 AP 定期广播（例如，每 100 毫秒发一个）。该帧向客户通告 AP 的存在，同时传递一些系统参数，比如 AP 的标识、时间（下一帧多久再来）和安全设置。

客户端可以在它发送到 AP 的帧中设置一个**电源管理位**（power-management），告诉 AP 自己进入**省电模式**（power-save mode）。在这种模式下，客户端可以打个盹，AP 将缓冲所有发给该客户的流量。为了检查入境流量，客户端在每次信标帧来时苏醒过来，并检查作为信标帧一部分的流量图。这张图告诉客户是否有为它缓冲的流量。如果有，则客户给 AP 发送一个 poll 消息，示意 AP 将缓冲的流量发送过来；在接收了缓冲的流量之后，客户可以再回去打盹，直到下一个新标帧被发送出来时苏醒。

另一个省电机制称为**自动省电交付**（APSD，Automatic Power Save Delivery），在 2005 年被添加到 802.11。有了这个新机制，AP 依然为休眠的客户端缓冲帧，但只在客户端发送帧到 AP 后才将其缓存的帧发送到客户端。这样一来，客户端可以安心进入睡眠状态，直到它有更多的流量需要发送（和接收）才醒来。这种机制对于诸如在两个方向上都有频繁

流量的 VoIP 等应用运作良好。例如，一个 VoIP 无线手机可能会使用该机制每 20 毫秒发送和接收帧，这个频度远远大于信标帧的 100 毫秒间隔，而在这之间客户端可以休眠。

第三个也是最后一个我们需要考查的是服务质量。当前面例子中的 VoIP 流量与对等流量（peer-to-peer）竞争时，VoIP 通信将受到影响。尽管 VoIP 流量所需要的带宽较低，但在和高带宽的对等流量竞争中还是有可能被延迟，这些延迟极有可能降低语音通话的质量。为了防止这种情况恶化，我们希望让 VoIP 通信先于对等流量传输，因为它拥有更高的优先级。

802.11 用一个巧妙的机制来提供这种服务质量，它是作为一组扩展于 2005 年被引入的，标准命名为 802.11e。它扩展了 CSMA/CA，并且仔细定义帧之间的各种时间间隔。一帧发出去后，需要保持一段特定时间的空闲以便检查信道不在被用，然后任何站才可以发送帧。这里的关键就在于为不同类型的帧确定不同的时间间隔。

5 个时间间隔如图 4-28 描述。常规的数据帧之间的间隔称为 **DCF 帧间隔**（DIFS，DCF InterFrame Spacing）。任何站都可以在介质空闲 DIFS 后尝试抓取信道发送一个新帧。采用通常的竞争规则，如果发生冲突或许还需要二进制指数后退。最短的间隔是**短帧间间隔**（SIFS，Short InterFrame Spacing）。它允许一次对话的各方具有优先抓住信道的机会。例子包括让接收方发送 ACK、诸如 RTS 和 CTS 的其他控制帧序列，或者让发送方突发一系列段。发送方只需等待 SIFS 即可发送下一段，这样做是为了阻止一次数据交流中间被其他站横插一帧。

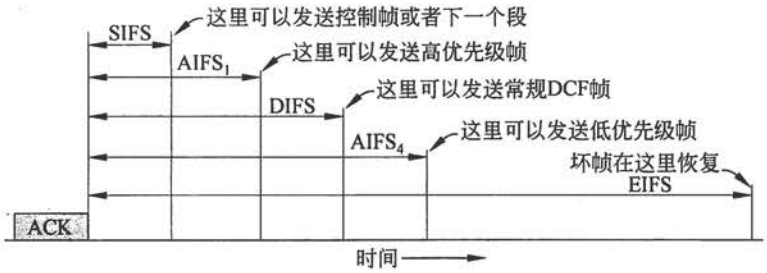


图 4-28　802.11 中的帧间间隔

两个**仲裁帧间空间**（AIFS，Arbitration InterFrame Space）间隔显示了两个不同优先级的例子。短的时间间隔 AIFS1 小于 DIFS，但比 SIFS 长。它被 AP 用来把语音或其他高优先级流量移到行头。AP 将等待一段较短的时间间隔，然后发送语音流量，这样语音流量得以在常规流量之前发送出去。较长的时间间隔 $AIFS_4$ 比 DIFS 还大，用它来发送可以延迟到常规流量之后发送的背景流量。AP 在发送这种流量之前将等待较长的时间间隔，以便给常规流量优先发送的机会。完善的服务质量机制定义了四种不同的优先级别，分别具有不同的后退参数和不同的空闲参数。

最后一个时间间隔是**扩展帧间间隔**（EIFS，Extended InterFrame Spacing），仅用于一个站刚刚收到坏帧或未知帧后报告问题。设置这个间隔的想法是因为接收方可能不知道该怎么处理，所以应该等待一段时间，以免干扰到两站之间正在进行的对话。

服务质量扩展的深入部分是 TXOP 或**传输机会**（transmission opportunity）概念。原始 CSMA/ CA 机制允许站一次发送一帧。这种设计很好，直到速率的范围增大。用 802.11a/g，

一个站可能以 6 Mbps 发送，另一站可能以 54 Mbps 发送。它们每个获得发送一帧的机会，6 Mbps 站所需要的时间（忽略固定间接费用）是 54 Mbps 站所需要时间的九倍。这一速率上的差距产生了不幸的副作用，它把快速发送方给拖累了，使得它在与慢速发送方竞争时速度大致降到慢速发送方的程度。例如，再次忽略固定开销，当单独以 6 Mbps 和 54 Mbps 速率发送时，发送方将获得自己的速率；但当它们一起发送时，两者都将达到 5.4 Mbps 的平均水平，这对快速发送方而言是个僵硬惩罚。这个问题就是所谓的速率异常（rate anomaly）（Heusse 等，2003）。

有了传输机会，每个站得到等量的通话时间，而不是相同数量的帧。以较高速率发送的站在它们的通话时间将获得更高的吞吐量。在我们的例子中，当一个 6 Mbps 和 54 Mbps 的发送方一起发送时，现在它们两个将分别得到 3 Mbps 和 27 Mbps 的吞吐量。

4.4.4　802.11 帧结构

802.11 标准定义了空中三种不同类型的帧：数据帧、控制帧和管理帧。每一种帧都有一个头，包含了与 MAC 子层相关的各种字段。除此以外，还有一些头被物理层使用，这些头绝大多数被用来处理传输所涉及的调制技术，所以这里我们不讨论它们。

数据帧的格式如图 4-29 所示。首先是帧控制（Frame Control）字段。它本身有 11 个子字段，其中第一个子字段是协议版本（Protocol Version），正是有了这个字段，将来可以在同一个蜂窝内同时运行协议的不同版本。接下来是类型（Type）字段（比如数据帧、控制帧或者管理帧）和子类型（Subtype）字段（比如 RTS 或者 CTS）。去往 DS（To DS）和来自 DS（From DS）标志位分别表明该帧是发送到或者来自于与 AP 连接的网络，该网络称为分布式系统。更多段（More fragment）标志位意味着后面还有更多的段。重传（Retry）标志位表明这是以前发送的某一帧的重传。电源管理（Power management）标志位指明发送方进入节能模式。更多数据（More data）标志位表明发送方还有更多的帧需要发送给接收方。受保护的（Protected Frame）标志位指明该帧的帧体已经被加密。我们将在下一节简要讨论安全问题。最后，顺序（Order）标志位告诉接收方高层希望严格按照顺序来处理帧序列。

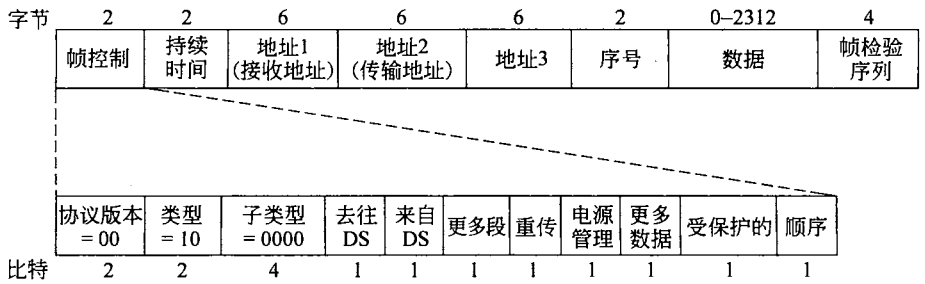


图 4-29　802.11 数据帧的格式

数据帧的第二个字段为持续时间（Duration）字段，它通告本帧和其确认帧将会占用信道多长时间，按微秒计时。该字段会出现在所有帧中，包括控制帧，其他站使用该字段来管理各自的 NAV 机制。

接下来是地址字段。发往 AP 或者从 AP 接收的帧都具有 3 个地址，这些地址都是标准的 IEEE 802 格式。第一个地址是接收方地址，第二个地址是发送方地址。很显然，这两个地址是必不可少的，那么第三个地址是做什么用的呢？请记住，当帧在一个客户端和网络中另一点之间传输时，AP 只是一个简单的中继点。这网络中的另一点也许是一个远程客户端，或许是 Internet 接入点。第三个地址就指明了这个远程端点。

序号（Sequence）字段是帧的编号，可用于重复帧的检测。序号字段可用 16 位，其中 4 位标识了段，12 位标识了帧，每发出去一帧该数字递增。数据（Data）字段包含了有效载荷，其长度可以达到 2312 个字节。有效载荷中前面部分字节的格式称为**逻辑链路控制**（LLC，Logical Link Control）。这层是个黏胶层，标识有效载荷应该递交给哪个高层协议处理（比如 IP）。最后是帧校验序列（Frame check sequence）字段，与我们在 3.2.2 节以及其他地方看到的 32 位 CRC 相同。

管理帧的格式与数据帧的格式相同，其数据部分的格式因子类型的不同而变（比如，信标帧中的参数）。控制帧要短一些，像所有帧一样，它们有 Frame control、Duration 和 Frame check sequence 字段。然而，它们只有一个地址，并且没有数据部分。大多数关键信息都转换成 Subtype 字段了（比如 ACK，RTS 和 CTS）。

4.4.5 服务

802.11 标准定义了服务，客户端、接入点和连接它们的网络必须是一个符合标准的无线局域网。这些服务可以分为几个组。

关联（Association）服务被移动站用来把自己连接到 AP 上。典型情况下，当一个移动站进入到某个 AP 的无线电覆盖范围之内时，就会用到这种服务。抵达后 AP 覆盖范围后，通过 AP 发送的信标帧移动站获知 AP 的标识符和能力，或者它直接询问 AP 获得有关信息。AP 能力包括所支持的数据率、安全考虑、节能能力、支持的服务质量等。移动站向 AP 发出一个与之关联的请求，AP 可能会接受，也可能会拒绝该请求。

重新关联（Reassociation）服务允许站改变它的首选 AP。这项服务对于那些从一个 AP 移动到另一个 AP 的移动站来说非常有用，就像蜂窝网络中的切换。如果这项服务使用正确，则切换的结果不会有数据丢失（但是如同以太网一样，802.11 也只是一种尽力而为服务）。不管是移动站，还是 AP 都有可能会解除关联（Disassociation）。一个站在离开本地或者关闭之前，应该先使用这项服务。AP 在停下来进行维护之前也可能会用到该服务。

站在通过 AP 发送帧之前必须认证（Authentication），但处理认证有多种方式，具体采用哪种方式取决于选择的安全模式。如果 802.11 网络是“开放”（open）的，那么任何人都可以使用它。否则，必须凭据进行身份验证。推荐的认证模式称为 WiFi 保护接入 2（WPA2，WiFi Protected Access 2），它实现了由 802.11i 标准定义的安全性（简单 WPA 只是一个过渡模式，它实现了 802.11i 的一个子集，我们将跳过它直接进入完整模式）。有了 WPA2，AP 可以直接和认证服务器联系，以便确定是否允许某个站接入网络，该认证服务器拥有一个用户名和密码数据库。另外一个安全机制是可以配置的预共享密钥（preshared key），这是网络密码的一个奇特称呼。站与 AP 之间通过“质询-回应”（challenge and response）方式来回交换多个帧，使站自我证明具有正确的安全凭据。这种交换发生在关联之后。

WPA 之前使用的模式称为**有线等效保密**（WEP，Wired Equivalent Privacy）。对于这种模式，预共享密钥的身份验证发生在关联之前。然而，它的使用令人气馁，因为设计上的缺陷使得 WEP 很容易妥协。WEP 被人攻破的第一个实际演示发生在 Adam Stubblefield 在 AT&T 公司做暑期实习生时（Adam Stubblefield 等，2002 年）。他能够在一个星期内编写代码并测试该攻击，其中大部分时间都花在从 AT&T 管理部门获得许可，从而购买实验所需 WiFi 网卡。现在破解 WEP 密码的软件可自由获得。

一旦帧到达 AP，**分发**（Distribution）服务决定了如何路由帧。如果帧的目的地对于 AP 来说是本地的，则该帧将被 AP 直接发送到空中；否则的话，它们必须通过有线网络来转发。如果一帧需要发往 802.11 局域网以外，或者从 802.11 局域网外面接收帧，可使用**集成**（Integration）服务来处理这里的任何协议转换。一般情况下，无线局域网与 Internet 连接。

数据传输是重中之重，因此 802.11 自然提供了**数据传送**（data delivery）服务。这项服务可让站采用我们在本章前面所述的协议来发送和接收数据。由于 802.11 基于以太网的模型，以太网传输不能保证 100%可靠，在 802.11 上传输同样不能保证任何可靠性。上层必须处理差错检测和纠正事宜。

无线是一种广播信号。若对通过无线局域网发送的信息要求保密，必须对其进行加密。这个目标通过**隐私**（privacy）服务实现，该服务管理加密和解密的细节。WPA2 加密算法基于**高级加密标准**（AES，Advanced Encryption Standard），这是美国政府在 2002 年批准的标准。加密所使用的密钥在认证过程中确定。

为了处理不同优先级业务，还有一个 **QoS 流量调度**（QOS traffic scheduling）服务。它使用我们描述过的这种协议：给予语音和视频流量比尽力而为和背景流量更高的优待。相伴的还有一个服务为高层提供计时器同步。这可让站协调它们的行动，这对于流媒体处理或许有用。

最后，还有两个服务帮助站管理它们的频谱使用。**发射功率控制**（transmit power control）服务为站提供了发射功率必须满足监管限制的信息，因为这种限制在不同地区有不同的规定。**动态频率选择**（dynamic frequency selection）服务为站提供了避免在 5 GHz 频段发送所需要的信息，因为这个频段刚好被附近的雷达在用。

有了这些服务，802.11 为附近移动客户端连接到 Internet 提供了丰富的功能集。这是一个巨大的成功，标准已多次被修订，增加了更多的功能。对于标准的未来和现状请参见（Hiertz 等，2010）。

4.5 宽带无线

我们在室内停留得太久了，现在让我们走出去看看非常有趣的所谓“最后一英里”（last mile）网络。由于在许多国家，电话系统已经被解除了管制，所以与传统电话公司竞争的公司现在通常也被允许提供本地语音和高速 Internet 服务。实际上，这种需求大量存在。问题在于将光纤或者同轴电缆铺设到数百万家庭和商务场所，其代价将会高得惊人。那么，竞争公司该怎么办呢？

答案是宽带无线。在城外的小山上竖立一根大天线，这比挖沟铺电缆要容易得多，也便宜得多。因此，与电信竞争的公司开始进行数兆比特速率的无线通信服务实验，包括语音、Internet、点播电影等应用。

为了刺激市场，IEEE 成立了一个小组对宽带无线城域网进行标准化。802 序号空间可用的下一个是 802.16，所以标准就取了这个数字。该技术被非正式称为**全球微波接入互操作性**（WiMAX，Worldwide Interoperability for Microwave Access）。我们将互换着使用术语 802.16 和 WiMAX。

第一个 802.16 标准于 2001 年 12 月获得批准。早期版本提供了一个固定点之间的无线本地回路，但必须采用视线（line of sight）链路传输。这种设计很快使得 WiMAX 成为替代线缆和 DSL 接入 Internet 的强有力竞争对手。2003 年 1 月，802.16 被修订成支持非视线链路传输，它采用了运行在 2 GHz 和 10 GHz 频率的 OFDM 技术。这一变化使得网络部署更加容易，尽管站依然是固定位置的。随着 3G 蜂窝网络的崛起所带来的潜在高数据率和移动性威胁，作为回应，802.16 被再次改进，支持车辆速度的移动性，该标准于 2005 年 12 月发布。移动宽带 Internet 接入是当前标准——IEEE 802.16-2009 的目标。

像其他 802 标准一样，802.16 也受到 OSI 模型的严重影响，包括（子）层、术语、服务原语等。不幸的是，跟 OSI 一样，它也是相当复杂。事实上，**WiMAX 论坛**（WiMAX Forum）的建立是为了给商业产品定义一个互操作的标准子集。在下面的章节中，我们将简短描述 802.16 空中接口的常见形式，但免不了支离破碎而且缺少很多细节。如需有关 WiMAX 和一般无线宽带的信息，请参见（Andrews 等，2007）。

4.5.1 802.16 与 802.11 和 3G 的比较

此时，你也许会想：为什么还要制定一个新的无线标准？为什么不使用 802.11 或 3G 呢？事实上，WiMAX 结合了 802.11 和 3G 两个方面，使其看起来更像 4G 技术。

和 802.11 一样，WiMAX 也是把设备通过无线而不使用线缆或 DSL 连接到 Internet，但其连接速度在 Mbps。设备可以移动，或至少是便携式的。WiMAX 没有开展语音蜂窝网络的低速率数据；802.16 被设计成在空中传送 IP 数据包，并以最小代价连接到基于 IP 的有线网络。数据包可携带对等流量、网络电话或流媒体应用。还有一点与 802.11 相像，它也基于 OFDM 技术和 MIMO 技术，前者用来确保无线信号在诸如多径衰落时有良好的性能，而后者则用来实现高吞吐率的传输。

然而，在几个关键方面 WiMAX 更像 3G（因而不像 802.11）。关键技术问题是如何有效使用频谱资源实现高容量，以便一个覆盖区域内的大量用户都能获得高吞吐量。典型的传输距离至少比 802.11 网络大 10 倍以上。因此，WiMAX 基站比 802.11 接入点（AP）要更强大。为了处理过大距离上的弱信号，基站使用了更大的功率和更好的天线，并进行更多的差错处理。为了最大限度地提高吞吐量，为每个特定用户精心调度数据包的传输；频谱使用不再采用 CSMA/CA，因为这种机制会因冲突而浪费容量。

正如预期的那样 WiMAX 使用了许可频谱，在美国该频谱通常在 2.5 GHz。整个系统的优化程度大大超过 802.11。考虑到购买所需的许可频谱花费了大量的资金，因而这种复杂性是值得的。与 802.11 不同，802.16 是一个可管理且可靠的服务，并且具备对服务质量

的良好支持。

具备了上述所有这些特性，802.16 最接近 4G 蜂窝网络，该网络目前正在进行标准化且被命名为**长期演进**（LTE，Long Term Evolution）。虽然 3G 蜂窝网络基于 CDMA，并且支持语音和数据传输，4G 蜂窝网络将基于 MIMO 和 OFDM 技术，它们的目标是数据传输，语音只是一个应用程序而已。在技术和应用方面，WiMAX 和 4G 技术看起来处于一种冲突的过程。也许，这种交汇不足为奇，毕竟 Internet 是杀手级应用，而 OFDM 和 MIMO 是有效利用频谱的最知名技术。

4.5.2　802.16 体系结构与协议栈

802.16 体系结构如图 4-30 所示。基站直接连接到提供商的骨干网络，该网络再连接到 Internet。基站通过无线空中接口与站通信。网络中并存着两种站：第一种是保持在一个固定位置的用户站，例如家庭宽带接入 Internet；第二种是可以在移动中接收服务的移动站，例如一辆配备了 WiMAX 装置的汽车。

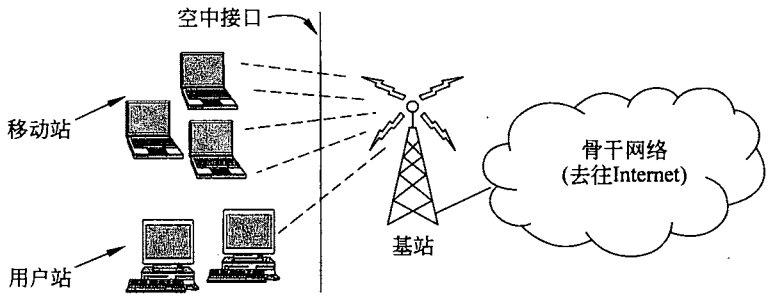


图 4-30　802.16 体系结构

用于空中接口的 802.16 协议栈如图 4-31 所示。总体结构与其他 802 网络相似，但有了更多的子层。最底层主要处理信号传输，这里我们只给出 802.16 的流行产品，分别是固定和移动 WiMAX。每个产品有不同的物理层。这两层运行在 11 GHz 以下的许可频谱并使用 OFDM，但方式上有所不同。

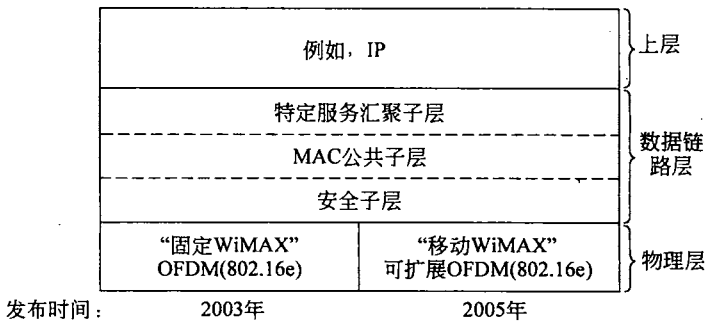


图 4-31　802.16 协议栈

物理层上面是数据链路层，它包括 3 个子层。最下面的子层涉及隐私和安全，公共室外网络相比私人室内网络，在安全性方面要求更高，因此这个子层尤为重要的。它管理加密、解密和密钥管理。

接着往上是常见的 MAC 子层。这是整个协议栈的最主要部分，比如信道管理。这里的模型是基站完全控制系统。它可以非常有效地调度下行链路（即基站到用户）的信道，同时对管理上行链路（即用户到基站）有着非常重要的作用。MAC 子层的这个不寻常特点与其他 802 协议不同，它是完全面向连接的，能为电话和多媒体通信服务提供质量保障。

特定服务汇聚子层采用了其他 802 协议的逻辑链路子层。它的功能是为网络层提供接口。协议定义了不同的汇聚子层，以便与不同的上层进行无缝连接。尽管标准还定义了与诸如以太网和 ATM 协议的映射，最重要的选择还是 IP。由于 IP 是无连接，而 802.16 MAC 子层是面向连接的，因此两层之间必须在地址和连接之间进行映射。

4.5.3 802.16 物理层

大多数 WiMAX 的部署都使用了 3.5 GHz 或 2.5 GHz 周围的许可频谱。与 3G 类似，寻找可用频谱是一个关键问题。为了有助于该问题的解决，802.16 标准的设计非常灵活。在 2 GHz 到 11 GHz 之间的范围内它都可以运行，并且支持大小不同的信道，例如，对于固定 WiMAX 是 3.5 MHz，而对于移动 WiMAX 可以选择的范围从 1.25 MHz 到 20 MHz 不等。

这些信道上的传输采用了 OFDM 技术，我们在 2.5.3 节中对此有过描述。相比 802.11，802.16 OFDM 系统进行了优化设计，因而获得了许可频谱和广域传输的最大效益。信道被分成具有一个更长符号持续时间的许多子载波，以便忍受无线信号的更大衰减；WiMAX 参数大约是 802.11 参数的 20 倍。例如，在移动 WiMAX 中，5 MHz 信道有 512 个子载波，并且在每个子载波上发送的符号大约持续 100 微秒。

每个子载波上的符号以 QPSK、QAM-16 或 QAM-64 调制方案发送，这些调制方案我们在 2.5.3 节都有过描述。当移动用户站或固定用户站就在基站附近，因而接收信号具有较高的信噪比（SNR）时，则采用 QAM-64 调制技术，每个符号发送 6 个比特。为了使信号到达信号质量比较差，即信噪比低的远程站，可采用 QPSK 调制技术，在每个符号上发送 2 比特。数据首先用卷积编码（或更好的方案）进行纠错编码，这种编码我们在 3.2.1 中介绍过，通常被用在嘈杂的信道，能容忍一些比特错误而无须重传。事实上，调制和编码方法现在应该听起来比较熟悉了，因为它们出现在许多我们已经学过的网络中，包括 802.11、线缆和 DSL。最终结果是基站在每个 5 MHz 信道和每对天线上可以支持高达 12.6 Mbps 的下行流量和 6.2 Mbps 的上行流量。

有一件事情是 802.16 设计者不喜欢的，就是 GSM 和 DAMPS 的某些工作方式。这两个系统的上行流量和下行流量使用了相等的频段。也就是说，它们隐含地假设下行流量与上行流量一样多。对于语音通信，大部分流量是对称的；但对于 Internet 接入（以及当然的网上冲浪）往往是下行流量大于上行流量，这个比例通常是 2∶1、3∶1，或更多。

因此，802.16 的设计者选择了一个在站之间划分信道的灵活方案，该方案称为**正交频分多址**（OFDMA，Orthogonal Frequency Division Multiple Access）。有了 OFDMA，可以为不同的站分配不同的子载波集，因此可以同时有多个站发送或接收。如果是 802.11，则在任何特定时刻所有的子载波只能被一个站用来发送。在如何分配带宽上的灵活性带来的直接好处是提高了系统性能，因为一个给定的子载波可能因多径效应在一个接收器上衰减得厉害，但在另一个接收器上却很清晰。可以把子载波分配给使用效果最好的站。

除了具有非对称流量，站通常交错着发送和接收。这种方法称为**时分双工**（TDD，Time Division Duplex）。在另一种称为**频分双工**（FDD，Frequency Division Duplex）的方式下，站可以在同一时间发送和接收（在不同的子载波频率上）。WiMAX 允许这两种方法，但 TDD 是首选，因为它更容易实施也更灵活。

图 4-32 显示了一个帧结构，随着时间不断重复的例子。它以一个前导码开始来同步所有的站，然后是来自基站的下行链路传输。首先，基站发送流量图，告诉所有站如何分配帧的下行子载波和上行子载波。基站控制着流量图，所以它可以根据每个站的需要一个帧一个帧地分配不同的带宽。

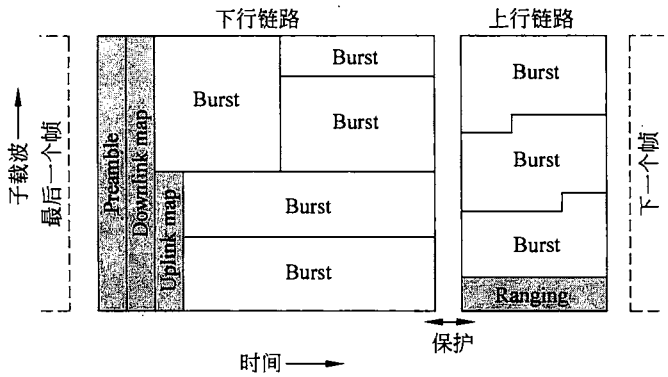


图 4-32　时分双工 OFDMA 的帧结构

接下来，基站给不同的固定用户和移动站发送突发流量，所用的子载波由同时出现的流量图规定。下行链路传输结束后，还要另外加一段保护时间，这样便于站从接收状态切换到发送状态。最后，用户站和移动站通过上行链路向基站发送突发流量，所用的子载波通过流量图保留。其中一个上行链路突发被保留用作**测距**（ranging），这是一个过程，新的站通过它调整自己的时间，并向连接的基站请求初始带宽。由于在这个阶段还没有建立连接，新站只是把请求发送出去，并希望没有冲突。

4.5.4　802.16 的 MAC 子层协议

数据链路层被分成三个子层，我们在图 4-31 中已经看到了。由于我们要到第 8 章才学习密码学，所以，现在很难表达清楚安全子层是如何工作的。现在我们只需知道，通过安全子层，所有传输的数据都经过了加密，从而是保密的。在一个帧中，只有有效载荷部分被加密，帧的头部并没有加密。这个特性意味着一个窥探者可以看出谁在跟谁通话，但是不知道他们之间在说什么。

如果你对密码学已有所了解，那么，下面一段只是关于安全子层的描述。如果你对密码学一无所知，那么你不太可能会发现下一段内容有多大的启发性（但是你可以考虑在学完了第 8 章之后再重读这一段）。

当一个用户连接到基站时，用户和基站利用 X.509 证书以及 RSA 公钥密码算法完成双向认证过程。有效载荷本身被一个对称密码系统加密，加密方法采用密码块链接的 AES（Rijndael）或 DES。完整性检查可以使用 SHA-1 算法。这还不算太糟糕，是不是？

现在我们来看MAC子层的公共部分。MAC子层面向连接，并且支持点对多点传输，这意味着一个基站可以和多个用户站通信。这种设计在很大程度上借用了线缆调制解调技术，那里有个线缆头端控制着客户端多个线缆调制解调器的传输。

下行链路方向的传输非常简单。基站控制着物理层的“突发”（burst），用来给不同用户站发送信息。MAC子层把自己的帧封装到这结构中。为了减少开销，有几种不同的选择。例如，MAC帧可能被单独发送，或背靠背地打包成组一起发送。

上行链路信道要复杂一些，因为多个需要接入的用户要竞争它的使用权。上行信道的分配方案与服务质量问题有紧密关系。服务分为4类，分别定义如下：

（1）恒定比特率服务。

（2）实时可变比特率服务。

（3）非实时可变比特率服务。

（4）尽力而为服务。

802.16中的所有服务都是面向连接的，每个连接属于上述4类服务之一，这是在建立连接时确定的。这种设计与802.11或者以太网截然不同，那两个网络在MAC子层是无连接的。

恒定比特率服务的目标是传输未经压缩的语音信号。这种服务需要在预定的时间间隔中发送预定数量的数据。具体做法是为这种类型的每个连接分配特定的“突发”。一旦带宽获得分配，则可自动用来发送“突发”，无须再次请求。

实时可变比特率服务的目标是传输被压缩的多媒体，或者其他一些软实时应用。在所谓的软实时应用中，每个时刻需要的带宽可能都会发生变化。802.16提供这种类型服务的做法是基站以一定的周期轮询用户，询问它每次传输需要多少带宽。

非实时可变比特率服务的目标是非实时但繁重的传输任务，比如大文件传输。对于这种服务，基站要经常轮询用户，但不是按照严格预定的时间间隔。这种服务的连接也可以使用尽力而为的服务（接下来描述）来请求带宽。

最后，尽力而为服务可以用于其他所有的应用。这里没有询问，用户必须与其他尽力而为型的用户竞争带宽。通过上行链路流量图中标记为可用的“突发”向基站发送带宽请求。如果请求获得成功，则该成功信息将被标注在下一个下行链路流量图中；如果请求不成功，则对应的请求用户必须以后重新发送请求。为了最低限度地避免冲突，系统也使用了以太网的二进制指数后退算法。

4.5.5 802.16帧结构

所有MAC帧都以一个通用的头作为开始。头之后跟着一个可选的有效载荷字段和一个可选的校验和（CRC）字段，如图4-33所示。在控制帧中有效载荷字段是不需要的，例如，那些请求信道时间槽的控制帧。（令人惊讶的是）校验和也是可选的，这是因为在物理层上进行了纠错，并且事实上实时帧也不需要考虑重传。如果没有打算重传，那为什么还要烦劳校验和呢？但如果有一个校验和，并且它是标准IEEE 802的CRC，那么可以使用确认和重传来提高可靠性。

下面简要地解释图4-33（a）中的头子段。EC比特指明了有效载荷字段是否经过了加密。Type（类型）字段标识了该帧的类型，通常指明是否存在封装和分段。CI比特表明本

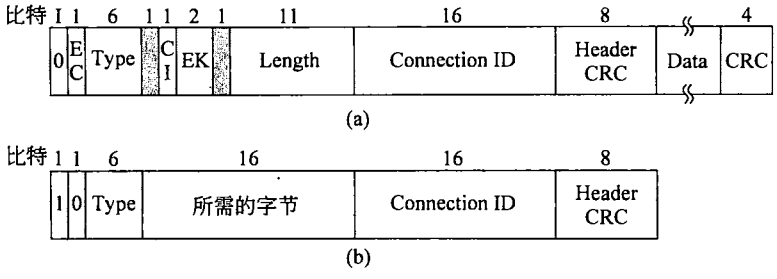


图　4-33
（a）通用帧；（b）带宽请求帧

帧是否包含最后的校验和。EK 字段指明了所使用的加密密钥（如果用到了加密的话）。Length（长度）字段给出了该帧的总长度，包括头部。Connection ID 字段指明了该帧属于哪个连接。最后，Header CRC 字段是一个仅仅针对头的校验和，所用的多项式为 x^8+x^2+x+1。

802.16 协议有许多不同类型的帧。图 4-33（b）显示了其中的一种，这是一个用来请求带宽的控制帧。它的开始处是一个“1”而不是“0”，它的第 2 个和第 3 个字节构成了一个 16 位的数字，该数字说明了需要多少带宽来运载规定数量的字节，除了这两个字节以外，其他与通用头类似。带宽请求帧并不包含有效载荷，也不包含针对整帧的 CRC。

关于 802.16 还有很多内容要介绍，但这里并不适合长篇大论。更多的信息请参考 IEEE 802.16-2009 标准。

4.6 蓝　　牙

1994 年，爱立信公司对用无线连接它的手机和其他设备（例如笔记本）产生了浓厚的兴趣。在 1998 年，它与其他 4 家公司（IBM、Intel、Nokia 和 Toshiba）一起组建了一个特别兴趣组（SIG，Special Interest Group）。兴趣组的目标是开发一个无线标准，可用来将计算设备、通信设备或其他附件通过短距离、低功耗和低成本的无线电连接起来。这个项目被命名为**蓝牙**（Bluetooth），名字来源于北欧的一个海盗王 Harald Blaatand（Bluetooth）II（940-981），他统一（即征服）了丹麦和挪威，当然也没有使用线缆。

蓝牙 1.0 发布于 1999 年 7 月，此后 SIG 一直往前从未回头。现在所有消费类电子设备都在使用蓝牙，从手机和笔记本电脑到耳机、打印机、键盘、鼠标、游戏机、钟表、音乐播放器、导航设备等。蓝牙协议使这些设备能互相发现并连接，从而安全地传送数据。彼此发现并连接的行为称为**配对**（pairing）。

该协议已经发展了近 10 年。在最初的协议稳定后，2004 年发布的蓝牙 2.0 添加了更高的数据传输速率。2009 年发布了蓝牙 3.0 版本，蓝牙可用来配对，结合 802.11 的设备，以便获得高吞吐量的数据传输。 2009 年 12 月发布的 4.0 版本规定了低功率操作，对于那些觉得定期更换居家所用设备电池很麻烦的人来说，真是太方便了。下面我们将介绍蓝牙的主要方面。

4.6.1 蓝牙体系结构

作为学习蓝牙系统的开始，我们首先快速浏览蓝牙系统中所包含的内容以及它所针对的目标。蓝牙系统的基本单元是一个**微网**（piconet），微网包含一个主节点，以及 10 米距离之内至多 7 个活跃的从节点。在同一个大房间中可以同时存在多个微网，它们甚至可以通过一个桥节点连接起来，如图 4-34 所示，该桥节点必须加入多个微网。一组相互连接的微网称为一个**散网**（scatternet）。

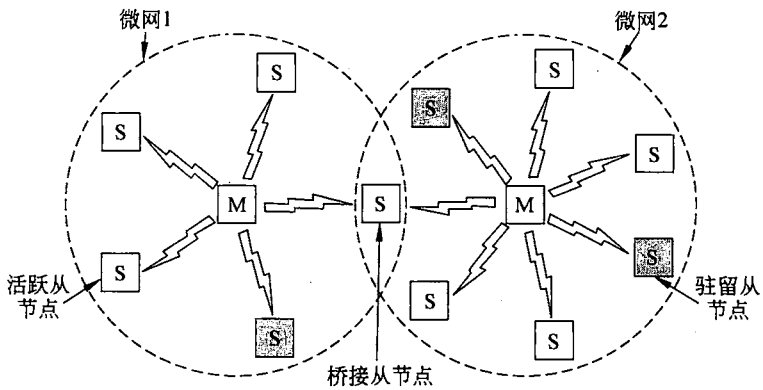


图 4-34 两个微网可以连接成一个散网

在一个微网中，除了 7 个活跃的从节点以外，还可以有多达 255 个驻留节点。所谓驻留节点（parked node）是指这样的设备，主节点已经将它们切换到一种低功耗状态，以便降低它们的电源消耗。一个处于驻留状态的设备，除了响应主节点的激活或者信标信号外，不做其他任何事情。还有两个中间电源状态：保持（hold）和嗅探（sniff）状态，这两种状态我们这里暂时放一放。

这种主/从模式的设计理念在于设计者期望一个完整的蓝牙芯片的实现代价低于 5 美元。在这种决策思想的指导下，从设备基本上都是哑设备，只能完成一些主节点告诉它们该做的事情。本质上，微网是一个集中式的 TDM 系统，主节点控制时钟，并决定每个时间槽被哪个设备用来通信。所有的通信都是在主节点和从节点之间进行；从节点与从节点不可能进行直接通信。

4.6.2 蓝牙应用

大多数网络协议只是为通信实体提供信道，至于用这些协议来做哪些事情，是应用设计者们应该考虑的问题。例如，802.11 并没有规定用户如何使用他们的笔记本电脑来收发邮件、浏览 Web 页面，或者别的事情。与此相反，蓝牙 SIG 规范却列出了蓝牙所支持的专门应用以及每一种应用对应的不同协议栈。在写这本书时，蓝牙已经可以支持 25 种应用程序，这些应用通称为**轮廓**（profiles）。不幸的是，这种做法导致了极大的复杂性。这里我们忽略复杂性，只是简略地考查这些轮廓，以便更清楚地了解蓝牙 SIG 努力要达到的目标。

六个轮廓专门针对音频和视频的不同用途。例如，对讲机（intercom）轮廓允许两个电

话相互连接，以对讲机的方式使用。无线耳麦（headset）和免提（hands-free）轮廓都提供了耳机与其基站之间的语音通信，就像开车时的免提电话。其他一些轮廓可用于流媒体应用，例如立体声品质的音频和视频，用在便携式音乐播放器到耳机（headphone），或从数码相机到电视机之间。

人机接口轮廓用于把键盘和鼠标连接到计算机。其他一些轮廓让手机或其他计算机接收来自摄像机的图像或把图像发送至打印机。更感兴趣也许是有一个轮廓，把移动电话作为（配蓝牙的）电视机的遥控器来使用。

其他轮廓依然需要联网。个域网轮廓允许蓝牙设备自我形成一个自组织网络或通过一个接入点远程访问另一个网络，比如 802.11 局域网。拨号联网轮廓实际上是整个项目的最初动机。它允许一台笔记本电脑无线连接到一台内置了调制解调器的移动电话。

更高层次的信息交流所需的轮廓也已得到了定义。同步轮廓用于离家时上载数据到移动电话，而回家时从移动电话收集数据。

我们会跳过其余的轮廓，但那些搭建上述轮廓服务所需要的基础轮廓我们还是会提到。通用访问（generic access）轮廓是构建所有其他轮廓的基础，它提供了一种建立和维护主站和从站之间安全链路（信道）的方式。其他通用轮廓定义了对象交流和音视频传输的基础。实用程序（utility）轮廓广泛用于诸如模拟串行线等功能，这对许多传统应用程序尤其有用。

真的有必要分清楚所有这些应用的细节，并且为每一种应用提供不同的协议栈吗？也许并没有这个必要。但是，由于存在多个不同的工作组，他们分别负责设计标准的不同部分，因此，每个工作组都关注特定的问题，从而形成了自己的轮廓。你可以把这看成是 Conway 法则在起作用（在 1968 年 4 月份的 Datamation 杂志上，Melvin Conway 评述说，如果你安排 n 个人编写一个编译器，那么你将会得到一个 n 步的编译器，或者更一般地，软件结构反映出了编写该软件的小组结构）。或许蓝牙标准根本不用 25 个协议栈，两个就可以了，一个用于文件传输，另一个用于流式实时通信。

4.6.3　蓝牙协议栈

蓝牙标准有许多协议，它们松散地分成多个层次，如图 4-35 所示。第一次观察得到的印象是该结构不遵循 OSI 模型中、TCP/ IP 模型、802 模型，或任何其他模型。

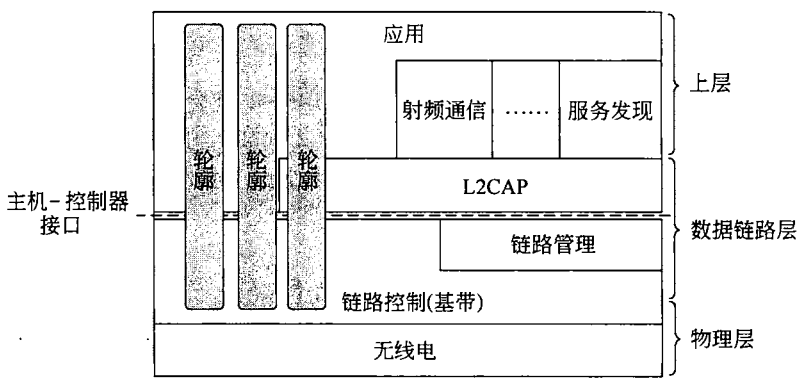


图 4-35　蓝牙协议体系结构

底层是物理无线电层，对应于OSI模型和802模型的物理层，它涉及无线传输和调制解调。这里更多关注的是如何达到使系统价格低廉成为大众市场目标的目的。

链路控制（或基带）层有点类似于MAC子层，但是包括物理层的元素。它涉及主节点如何控制时间槽以及如何将这些时间槽组成帧。

接下来的两个协议使用了链路控制协议。链路管理器处理设备之间的逻辑信道建立，包括电源管理、配对和加密以及服务质量。它位于主机控制器接口线的下面。该接口是为了整个系统的实现方便而设置的：一般情况下，接口线下面的协议由蓝牙芯片实现，接口线上面的协议由蓝牙设备实现，蓝牙芯片就装在该设备上。

接口线上面的链路协议是**逻辑链路控制适配协议**（L2CAP，Logical Link Control Adaptation Protocol）。帧携带可变长度的消息，如果需要还能提供可靠性。许多协议要用到L2CAP，例如图示的两个实用程序。**服务发现**（service discovery）协议用于在网络中寻找到可用服务。**射频通信**（RFcomm，Radio Frequency communication）协议模拟PC上的标准串行端口，用于连接键盘、鼠标和调制解调器等其他设备。

最上层是应用程序的所在位置。轮廓由垂直的条状块表示，因为它们各自定义了实现特定目标的协议栈切片。特定的轮廓，如耳机轮廓，通常只包含该应用程序所需要的协议，而没有包含该应用用不到的协议。例如，如果有数据包要发送，则轮廓可能包括L2CAP；但如果它们只有一个稳定的音频样本，则轮廓就会跳过L2CAP。

在下面的章节中，我们将考察蓝牙无线电层和不同的链路协议，因为它们大致对应于我们已经考察过的其他网络协议栈的物理层和MAC子层。

4.6.4 蓝牙无线电层

无线电层将比特信息从主节点移动到从节点，或者从节点移动到主节点。蓝牙是一个低功率的系统，距离范围为10米，运行在2.4 GHz的ISM频段上。该频段被分成79个信道，每个信道宽1 MHz。为了与使用ISM频段的其他网络共存，蓝牙使用了跳频扩展技术，每秒1600跳，驻留时间625微秒。一个微网中的所有节点同步跳频，遵循主节点规定的时间槽和伪随机调频序列。

不幸的是，事实证明，蓝牙的早期版本与802.11之间的干扰足以毁掉双方的传输。一些公司对此作出的反应是完全禁止蓝牙，但最终制定出了一个技术解决方案：蓝牙自适应其调频序列，排除掉有其他射频信号的信道。这个过程减少了有害干扰，就是所谓的**自适应跳频**（adaptive frequency hopping）。

在信道上发送比特可采用三种形式的调制解调技术。基本方案是使用频移键控每微秒传送1比特符号，提供1Mbps的总数据速率。蓝牙2.0版本引进了增强型数据速率。这些速率使用相移键控每符号发送2个或3个比特，提供2或3 Mbps的总数据速率。增强型速率只能用在帧的数据部分。

4.6.5 蓝牙链路层

链路控制（或基带）层是蓝牙中最接近MAC子层的部分。它将原始比特流转换成帧，

并定义了一些关键格式。在最简单的形式中，每个微网中的主节点定义了一组 625 微秒的时间槽，主节点在偶数时间槽中开始传输，从节点在奇数时间槽中开始传输。这个方案就是传统的时分多路复用，主节点获得一半的时间槽，从节点共享剩余的一半时间槽。帧可以是 1、3 或 5 个时间槽长。每个帧有一个 126 位的开销用作访问码和头，再加上每跳 250～260 微秒的稳定时间使廉价的无线电路变得稳定。为了达到保密要求，帧的有效载荷部分可用密钥加密，该密钥在主从节点建立连接时选定。频率的跳动只发生在两帧之间，而在一帧传输期间频率保持不变。这样的设计结果是一个 5 个时间槽的帧比 1 个时间槽的帧更为有效，因为开销不变，但发送了更多的数据。

链路管理协议负责建立逻辑信道。逻辑信道称为**链路**（link），主设备和从设备通过它运载帧，当然这些主从设备必须彼此能发现。主从设备的彼此发现要经过一次配对过程，目的就是要确保两个设备在使用链路之前允许通信。旧的配对方法是两个设备必须配置相同的四位**个人识别号码**（PIN，Personal Identification Number）。PIM 码的匹配结果给设备提供了确保正在连接的远程设备是正确的途径。然而，缺乏想象力的用户和设备把 PIM 默认设置成如“0000”和“1234”，这种方法在实际使用中提供的安全性很小。

新的**安全简单配对**（secure simple pairing）方法使用户能够确认这两个设备都显示相同的密码，或在一台设备上观察到密码，再输入到第二个设备。这种方法更加安全，因为用户不需要选择或设置 PIN。他们仅仅确认一个由设备产生的长密钥。当然，它不能被用在输入/输出手段有限的某些设备上，比如免提耳麦（hands-free headset）。

一旦配对成功，链路管理协议就在两个设备之间建立链路。用来运载用户数据的有两种主要形式的链路。第一种是**同步有连接**（SCO，Synchronous Connection Oriented）链路，它主要用于实时数据的传输，比如电话连接。这种链路在每个方向分配固定的时间槽。一个从节点与它的主节点之间可以有多达 3 条 SCO 链路。每条 SCO 链路可以传送一个 64 000 bps 的 PCM 音频信道。由于 SCO 链路的实时性本质，在这种链路上发送的帧永远不会被重传。

另一种是**异步无连接**（ACL，Asynchronous ConnectionLess）链路。这类链路用来以数据包方式交换那些无时间规律的数据。ACL 流量基于尽力而为（best-effort）的投递，没有任何保证。帧可能会丢失，也可能需要重传。一个从节点与主节点之间只可以有一条 ACL 链路。

在 ACL 链路上发送的数据来自上面的 L2CAP 层。该层有四个主要功能。首先，它从上层接受高达 64 KB 的数据包，并把数据包拆分成帧传输；在另一端，帧被重组成数据包。其次，它处理多个数据包源的多路复用和分用。当一个包被重组后，L2CAP 层决定由哪一个上层协议来处理，例如，RFcomm 或服务发现。再次，L2CAP 处理差错控制和重传。它检测错误，并重新发送没有被确认的数据包。最后，L2CAP 强制多个链路之间的服务质量要求。

4.6.6　蓝牙帧结构

蓝牙定义了好几种帧格式，其中最重要格式如图 4-36 所示。帧的开头是一个**访问码**（access code），它通常标识了主节点，所以，当从节点同时位于两个主节点的无线电覆盖范围内时，它可以利用这个访问码来区分本帧发自哪个主节点。接下来是一个 54 位的**头**

(header)，其中包含了典型 MAC 子层的字段。如果帧是以基本速率发送的，则紧接着的是数据字段，对于 5 个时间槽的传输最多可包含 2744 位。对于一个单时间槽的帧，除了数据字段长度减少到 240 位以外，格式的其他方面都一样。

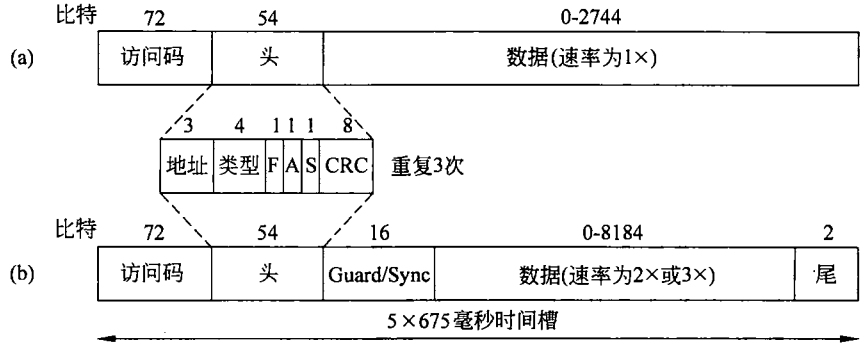


图 4-36 两种情况下的典型蓝牙数据帧
（a）基本型；（b）增强型

如果帧以增强型速率发送，数据部分可能多达 2～3 倍，因为每个符号携带 2 个或 3 个比特，而不是 1 个比特。在数据字段之前是一个保护字段和同步模式，用来切换到更快的数据速率。也就是说，访问代码和头是按基本速率传输，只有数据部分用更快的速率传输。增强型速率的帧以一个短帧尾结束。

现在让我们快速地浏览头中的内容。地址（Addr）字段标识了该帧的接收目标，指向 8 个活跃设备中的某一个。类型（Type）字段标识了帧的类型（ACL、SCO、轮询或者空）、数据字段中所使用的纠错码类型，以及该帧的时间槽长度。F（Flow）流标志位由从节点设置，当它缓冲区为满不能再接收任何数据时，利用该比特来声明不能接收数据的事实。这是流量控制的基本形式。A（Acknowledgement）确认标志指明在一帧中捎带了一个 ACK。S（Sequence）序号被用于帧的编号，以便接收方检测重传帧。协议采用停-等式控制机制，因此 1 位序号就够了。然后是 8 比特的头校验和 CRC。整个 18 位的头重复了 3 次，由此构成了如图 4-36 所示的 54 位头。在接收方，一个简单的电路检查这 3 份副本中的每一位。如果 3 份副本都相同，则该位被接受；如果不相同，则少数服从多数。因此，54 位的容量被用来发送 10 位头。这种做法的原因在于，要想在一个噪声环境中用廉价的、计算能力弱的低功耗（2.5 毫瓦）设备可靠地发送数据，大量的冗余是必需的。

ACL 帧和 SCO 帧的数据字段用到了多种格式。我们把基本速率的 SCO 帧作为学习的简单例子：数据字段总是 240 位。这里共定义了 3 个变种格式，分别允许实际有效载荷为 80、160 或者 240 位，余下的位用于纠错。在最可靠的版本中（80 位有效载荷），内容被重复了 3 次，与头的处理方法相同。

我们可以这样分析此帧的容量。由于从节点只能使用奇数时间槽，跟主节点一样它得到 800 槽/秒。若用 80 位的有效载荷，来自从节点的信道容量为 64 000 bps，与来自主节点的信道容量一致。这样的容量对于单个全双工的 PCM 语音信道恰好足够（这就是为什么选择每秒 1600 跳的原因）。也就是说，尽管原始带宽为 1 Mbps，一个全双工未压缩的语音信道就可以完全饱和微网。这里建立时间花了 41%，帧头花了 20%，重复编码花了 26%，因此最终获得的效率是 13%。这个缺点凸显了增强型速率和多个时间槽的价值。

关于蓝牙技术还有很多内容值得介绍，但是限于篇幅，在此只能到此为止。好奇心强的读者，可参考蓝牙 4.0 规范，它给出了全部的细节。

4.7 RFID

我们已经考查了 MAC 层的设计，从局域网开始，大到城域网，小到个域网。作为最后一个例子，我们将学习一类低端的无线设备：**无线射频识别**（RFID，Radio Frequency IDentification）标签和读写器，这类设备人们可能还不太承认可用来组成计算机网络。我们在 1.5.4 节简单描述过。

RFID 技术有多种形式，通常被用在智能卡中，比如植入到宠物、护照、图书馆书籍中等。我们将着眼于一种形式：**电子产品码**（EPC，Electronic Product Code），它的研究始于 1999 年美国麻省理工学院 Auto-ID 中心。EPC 是条形码的替代品，它可以携带更多的信息，并且 10 米以内可被电子读写器读出，即使不可见也可以。相比以前那种在执行事务时必须离读卡器非常近的技术，RFID 是一种完全不同于以往的技术，例如，可在护照中使用 RFID 技术。远程通信的能力使得 EPC 更贴近我们的研究。

EPCglobal 成立于 2003 年，目标是商业化由 Auto-ID 中心开发的 RFID 技术。当 2005 年沃尔玛要求其前 100 名供应商用 RFID 标签所有货物，这种努力得到了提升。虽然与成本低廉的条形码在竞争上有一定难度，阻碍了 RFID 的广泛部署，但新的用途正在不断增长，例如驾驶执照。我们将介绍这项技术的第二代，即非正式的 EPC Gen2（EPCglobal，2008）。

4.7.1 EPC Gen 2 体系结构

EPC Gen 2 RFID 网络体系结构如图 4-37 所示。它有两个关键部分：标签和读写器。RFID 标签是很小的廉价设备，它有一个唯一的 96 位 EPC 识别码和少量内存，内存信息可以由 RFID 读写器读取和写入。内存可用来记录某个项目的历史位置，例如，记录它在供应链中的移动轨迹。

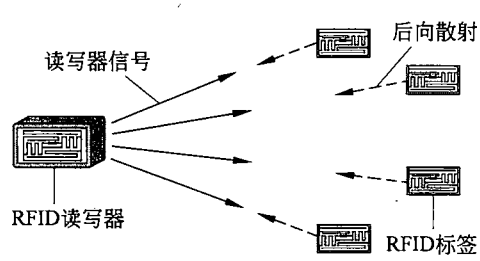


图 4-37　RFID 体系结构

通常情况下，标签的样子有点像可以贴在物品上的贴纸，例如，商店货架上牛仔裤上贴的那种。大部分贴纸采用了被打印上去的天线。中间的一个小点就是 RFID 集成电路。另外，RFID 标签可以被集成到一个物体中，比如驾驶执照。在这两种情况下，标签没有电池，它们必须从附近 RFID 读写器的无线传输中收集运行所需的功率。这类标签称为“1 类”标签，有别于带电池且具有更多能力的标签。

系统中的读写器是智能的，类似于蜂窝和 WiFi 网络中的基站和接入点。阅写器的功率远远高于标签。它们有自己的电源，通常有多个天线，并且负责标签发送和接收消息。通常在一个读写范围内存在多个标签，因此读写器必须解决多路访问问题。而且，在同一区

域可能存在多个读写器，它们之间还要竞争。

读写器的主要工作是盘点附近的标签，就是发现附近标签的识别码。盘点的过程涉及物理层协议和标签识别协议，这些内容将在以下章节描述。

4.7.2 EPC Gen 2 物理层

物理层定义了 RFID 读写器和标签之间如何发送比特。发送无线信号采用的大多数方法我们以前已经了解过。在美国，EPC Gen2 使用了 902～928 MHz 的无许可 ISM 频段来传输信号。这个频段落在**超高频**（UHF，Ultra High Frequency）范围内，所以标签也称为超高频 RFID 标签。读写器以至少每 400 毫秒的速度进行跳频，把信号分散到整个信道上，以此来限制干扰和满足监管要求。读写器和标签使用**幅移键控的调制**（ASK，Amplitude Shift Keying）形式进行编码，这种调制解调技术我们在 2.5.2 节学习过。读写器和标签轮流发送比特，因此链路是半双工的。

这里的物理层与我们学过的其他物理层有所不同，主要存在两个主要差异。首先，读写器永远在发射信号，无论当前是读写器还是标签在通信。读写器自然是通过发射信号将比特传递给标签。如果标签要给读写器发送比特，读写器就发送一个不携带任何比特的固定载波信号。标签收获该信号来获得运行所需要的能量；否则，标签将根本不能传送任何比特。为了发送数据，标签必须改变信号，要么把来自读写器的信号反射回去，就像雷达信号从目标反射回来一样，要么把来自读写器的信号吸收。

这种方法称为**后向散射**（backscatter）。它不同于所有我们迄今为止看到的其他无线情况，即发送方和接收方从来没有在同一时间同时传输。后向散射是一种低能量的方式，标签用它来生成一个显示在读写器上的微弱信号。读写器为了解码入境信号，必须从入境信号中筛选出自己正在传的出境信号。因为标签信号很弱，标签只能以低速率给读写器发送比特，标签不能接收，甚至侦听发自其他标签的传输。

第二个区别在于这里采用的调制解调技术非常简单，因为唯有简单才能在标签上实现。标签的运行能耗非常低，而且制造成本只有几美分。为了将数据发送给标签，读写器使用了两个振幅。比特 0 或 1 的确定取决于读写器在低功耗时期之前等待的时间长短。标签测量两次低功耗时期的时间，并将这段时间与前导码期间测得的参考值比较。如图 4-38 所示，比特 1 的时间长于比特 0 的时间。

标签的响应包括以固定的时间间隔交错它的后向散射状态，以便在信号中生成一系列脉冲。无论何处都可用一个到八个脉冲周期来编码每个比特 0 或比特 1，具体依赖于所需的可靠性。比特 1 要比比特 0 的跳变少，图 4-38 给出了一个双脉冲周期编码的例子。

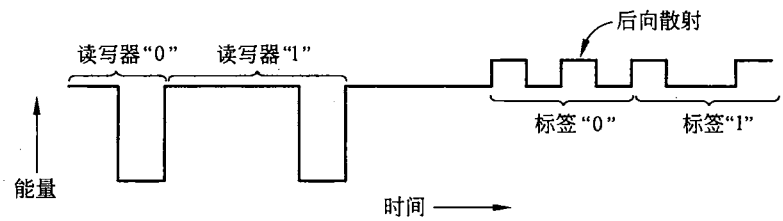


图 4-38 读写器和标签后向散射信号

4.7.3　EPC Gen 2 标签标识层

为了盘点附近的标签，读写器需要接收来自每个标签的消息，该消息给出了标签的标识符。这种情形就是多路访问问题，而且标签的数量在一般情况下是未知的。读写器可能会广播一个查询消息，要求所有标签发送它们的标识符。然而，标签如果都马上应答，则一定会发生冲突，如同一个经典以太网上的站同时发送一样。

我们已经看到本章有许多解决多路访问问题的方法。当前情形是标签不能听到彼此的传输，与此最接近的协议是分槽 ALOHA，这是我们最早学习的协议。该协议可适应第二代 RFID（Gen 2 RFID）。

识别一个标签所使用的消息序列如图 4-39 所示。在第一个时间槽（slot 0），读写器发送一个 Query（查询）消息来启动这个进程。每个 QRepeat 消息前进到下一个时间槽。读写器还告诉标签时间槽的范围，标签就在这个范围内随机传输。确定随机范围是必要的，因为读写器在启动该过程时要与标签同步；与以太网中的站不同，标签不会在自己选择的时间到时被消息唤醒。

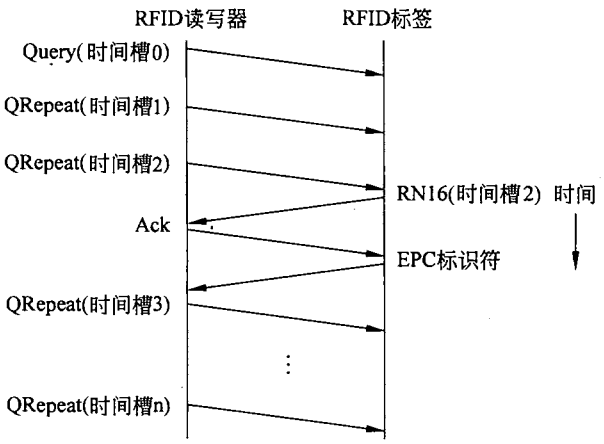


图 4-39　识别一个标签的消息交换示例

标签随机选择一个时间槽来应答。在图 4-39 中，标签在时间槽 2 中应答。然而，标签没有在第一次应答时发送自己的标识符；相反，标签通过 RN16 消息发送一个简短的 16 位随机数。如果没有发生冲突，读写器就能收到此消息并发送它自己的 ACK 消息。到这个阶段，标签已经获得了时间槽，并且发送其 EPC 标识符。

采取这种消息交换模式的原因在于 EPC 标识符很长，识别符冲突的代价太昂贵。相反，一个简短的消息交换可用来测试标签是否可以安全地使用时间槽来发送其标识符。一旦它的标识符被成功发送，标签就暂时停止响应新的 Query 消息，以便所有剩余标签可以被读写器逐一识别。

对读写器而言，关键的问题是调整时间槽数目来避免冲突，但又不能使用太多的时间槽以免影响性能。这种调整类似于以太网中的二进制指数后退算法。如果读写器看到太多时间槽没有任何响应或者太多的时间槽内发生冲突，它可以发送一个 QAdjust 消息来减少或增加时间槽的范围，即标签响应随机选择的后退时间槽数目。

RFID 读写器还可以对标签执行其他操作。例如，它可以选择盘点一组标签，允许它收集一类物品的库存，比如只需要盘点牛仔裤的库存，那么牛仔裤上的标签要有所反应，但衬衫上的标签就不能有反应。读写器还可以把数据写入标签。此功能可用于记录销售点或其他相关信息。

4.7.4 标签标识消息格式

Query 消息的格式如图 4-40 所示，这是一个读写器发给标签的消息示例。因为下行链路速率有限，通常只有 27～128 kbps，所以消息结构非常紧凑。Command（命令）字段携带代码 1000 用来标识该消息是 Query（查询）消息。

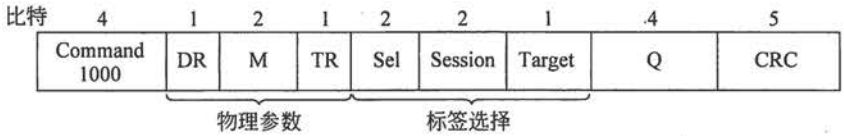


图 4-40 查询消息的格式

接下来是标志位：DR、M 和 TR，确定了读写器传输和标签响应的物理层参数。例如，响应率可设定为 5～640 kbps。我们跳过这些标志位的细节描述。

然后是 3 个字段：Sel、Session 和 Target，即选择标签作出回应。除了读写器选择一个标签子集要求回复外，标签最多可跟踪四个并发的会话以及它们是否在这些会活中已被识别。这样，多个读写器可以在多个重叠覆盖区域通过不同的会话实施各自的操作。

接着是最重要的命令参数 Q。该字段定义了标签的响应范围，从 0～2^Q-1。最后，是一个保护消息字段的 CRC。CRC 只有 5 位，它比我们所看到的大多数 CRC 都短，但 Query 信息比大多数数据包也短了许多。

标签到读取器的消息较为简单。由于读写器完全控制了传输，因此它知道对于自己每次传输期望得到的响应是什么样的。标签的响应非常简单，只是传输数据，例如 EPC 标识符。

最初标签只有标识作用。然而，随着时间的推移，它们已经成长为非常小的计算机。一些研究标签有传感器，能够运行小程序来收集和处理数据（Sample 等，2008）。这项技术的一个愿景是“物联网”（Internet of things），把物理世界中的物体连接到 Internet（Welbourne 等，2009；与 Gershenfeld 等，2004）。

4.8 数据链路层交换

许多组织有多个局域网，并希望将它们连接在一起。如果我们能把多个局域网联结起来组成更大的局域网岂不是更加方便？事实上，当我们采用称为**网桥**（bridge）的设备来连接这些局域网就可以做到这一点。我们在 4.3.4 节描述的以太网交换机是网桥的现代名称；它们提供的功能超越了传统的以太网和以太网集线器，可以很容易地把多个局域网加入到一个更大更快的网络上。我们将交替着使用术语“网桥”和“交换机”。

网桥工作在数据链路层，因此它们通过检查数据链路层地址来转发帧。由于它们不应该审查被转发帧的有效载荷字段，因此它们可以处理 IP 数据包，也可以处理其他类型的数据包，比如 AppleTalk 包。与此相反，**路由器**（router）检查数据包的地址，并基于这些地址路由数据包，所以它们只能以预先设计好的协议工作。

在本节，我们将考查网桥是如何工作的，即它们如何把多个物理局域网连接成一个逻辑局域网。我们还将考查如何颠倒过来，把一个物理局域网看成多个逻辑局域网，这种技术称为虚拟局域网（VLAN，Virtual LAN）。这两种技术为网络管理提供了非常有用的灵活性。有关网桥、交换机以及相关主题的综合处理，请参阅（Seifert 和 Edwards，2008；Perlman，2000）。

4.8.1　网桥的使用

在讨论网桥技术之前，有必要先看一些使用网桥的常见情形。我们将列举出三个理由来说明为什么一个组织应该结束多个 LAN 的局面。

第一，许多大学和公司部门都有自己的 LAN，这些 LAN 主要将部门内部的个人计算机、服务器和其他设备（比如打印机）连接起来。由于各个部门的目标不同，所以，不同部门可能选择了不同的 LAN，往往不会顾及其他部门正在做什么。但迟早各部门需要相互沟通，所以需要网桥。在这个例子中，之所以存在多个 LAN 的原因在于各个部门自治管理自己的内部网络。

第二，一个组织可能在地理上分布在几个楼宇，这些楼宇之间有一定的距离。在每个楼内有一个独立的 LAN，然后通过网桥和光纤链路将这些 LAN 连接起来，这种做法比起把全部电缆连到一个中央交换机要经济实惠得多。即使很容易做到铺设线缆，也还要受到线缆长度的限制（比如，对于千兆以太网来说双绞线的长度不得超过 200 米）。因为过大的信号衰减或来回延迟，网络无法在长距离的线缆上工作。唯一的办法是把 LAN 进行划分，并安装网桥连接每个分区，以此增加网络覆盖的总物理距离。

第三，有时候可能有必要将一个逻辑上的单个 LAN 分成多个独立的 LAN（用网桥连接）以便适应网络的负载。例如，在许多综合大学中，需要几千台工作站供学生和教师使用。公司也可能拥有上千名员工。这种系统的规模很大，因而不适合把所有的工作站都放在一个 LAN 上 ——因为需要上网的计算机数目远大于任何以太网集线器的端口数，或者说站的数目多于单个经典以太网所允许的最大站数。

即使有可能通过布线把所有的工作站连一起，把更多的站放在一个以太网集线器或经典以太网内也不会增加容量。所有的站共享固定容量的带宽。连的站越多，每站获得的平均带宽越少。

然而，两个独立的局域网有两倍于单个 LAN 的容量。网桥让局域网结合在一起，同时保持了这种容量。这里的关键是不往不需要去的端口发送流量，使得每个局域网得以全速运行。这种行为也增加了传输的可靠性，因为单个局域网上的一个出错节点连续输出垃圾流量可以堵塞整个网络。通过决定什么可以转发什么不能转发，网桥就像建筑物的防火门，防止已经发狂的单个节点拖垮整个系统。

为了更容易获得这些好处，理想的网桥应该是完全透明的。走出去把网桥买回来，然

后把 LAN 线缆插入网桥，顷刻间，一切都能很好地工作。此时，不需要更改硬件、不需要更改软件、不需要设置交换机地址、不需要下载路由表或参数，总之，什么都不需要做。只要插上线缆，然后走开。此外，现有的局域网操作不应该受到网桥的任何一点影响。只要站是连着的，应该观察不到有桥和没桥之间存在任何的区别，即显示不出它们是否属于桥接 LAN 的一部分。从一个桥接的局域网中删除一个站非常容易，如同把它从单个 LAN 中删除一样的方便。

足以令人瞠目结舌的是实际上有可能创建一个透明的网桥。这里需要用到两种算法：一个是**后向学习算法**（backward learning），用来阻止不需要发送的流量；另一个是**生成树算法**（spanning tree），用来打破不管三七二十一把交换机线缆连接起来而可能形成的环路。现在让我们考查这些算法，然后反过来看这些算法是如何完成这项神奇功能的。

4.8.2 学习网桥

两个局域网桥接在一起的拓扑结构分两种情况，如图 4-41 所示。在左侧，两个多点局域网，比如经典以太网通过一个特殊的站连接在一起，这个站就是同属于两个局域网的网桥。在右侧，局域网用点到点电缆连接在一起，包括一个集线器。网桥是站和集线器都能与之相连的设备。如果 LAN 技术是以太网，则网桥就是广为人知的以太网交换机。

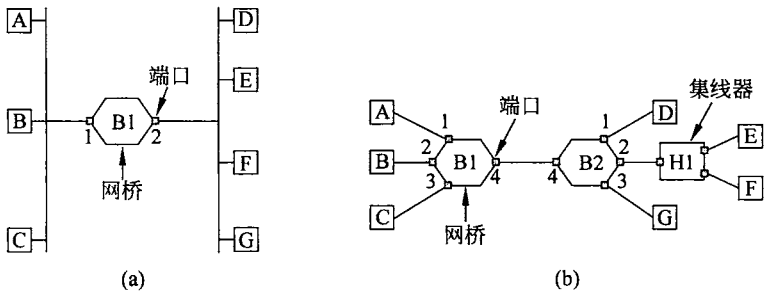


图 4-41

（a）连接两个多点 LAN 的网桥；（b）连接 7 个点到点站点的网桥（和集线器）

网桥开发之时正是经典以太网被广泛使用之际，所以它们常常表现为具有多点电缆的拓扑结构，如图 4-41（a）所示。然而，所有今天遇到的拓扑结构都由点到点线缆和交换机组成。网桥在两种设置下以相同的方式工作。所有附在网桥同一端口的站都属于同一个冲突域，该冲突域和其他端口的冲突域是不同。如果存在多个站，例如传统的以太网、集线器或半双工链路，那么帧的发送需要用到 CSMA/ CD 协议。

然而，在如何构建桥接局域网上，两种情况还是有区别的。为了桥接多点局域网，网桥的加入犹如在每个多点局域网上增加了一个新站，如图 4-41（a）所示。为了桥接点到点局域网，要么集线器连到网桥，或者最好换成网桥以便提高性能。图 4-41（b）显示除了一个集线器外，其余都用网桥替代了。

不同种类的线缆都可以附接到一个网桥上。例如图 4-41（b）中，连接网桥 B1 和 B2 的线缆可能是长距离光纤连接，而连接网桥和站的线缆可能是短距离双绞线。这种安排对于桥接不同楼宇内的局域网非常有用。

现在让我们来看看网桥内部会发生什么。每个网桥工作在混杂模式下，也就是说，它接受隶属于每个端口的站发送的帧。网桥必须决定是否转发或丢弃收到的每一帧，而且，如果是前者还要决定在哪个端口传输帧。做出决定的依据是帧的目标地址。作为一个例子，考虑图 4-41（a）的拓扑结构。如果站 A 发送一个帧给站 B，网桥 B1 将在端口 1 上接收该帧。这个帧立即被丢弃不再啰唆，因为它已经在正确的端口上。然而，在图 4-41（b）的拓扑结构中， 假设站 A 发送一帧给站 D。网桥 B1 将在端口 1 收到此帧，并从端口 4 转发出去；然后网桥 B2 将从端口 4 接收此帧，并将其从端口 1 转发出去。

执行这项工作的一个简单方法是为每个网桥配备一个大的（哈希）表。该表列出每个可能的目的地以及它隶属的输出端口。例如，在图 4-41（b）中，B1 上的表将列出 D 属于端口 4，因为 B1 所要知道的全部就只是从哪个端口出发能到达 D。事实上，当抵达 B2 的帧对 B1 来说不是感兴趣的（B1 对抵达 B2 的帧不感兴趣），更多的转发将会发生。

当网桥被第一次接入网络时，所有的哈希表都是空的。没有一个网桥知道哪个目标地址该往哪里去，因此网桥使用了一种**泛洪算法**（flooding algorithm）：对于每个发向未知目标地址的入境帧，网桥将它输出到所有的端口，但它来的那个输入端口除外。随着时间的推移，网桥将会学习到每个目标地址在那里。一旦知道了一个目标地址，以后发给该地址的帧只被放到正确的端口，而不再被泛洪到所有端口。

网桥所用的算法是**后向学习法**（backward learning）。正如上面所提到的，网桥工作在混杂模式下，所以，它们可以看得到每个端口上发送的所有帧。通过检查这些帧的源地址，网桥就可获知通过那个端口能访问到哪些机器。例如，如果在图 4-41（b）中，网桥 B1 看到端口 3 上的一帧来自站 C，那么它就知道通过端口 3 一定能到达 C，因此它就在哈希表中构造一项。以后所有抵达 B1 要去 C 的帧都将被转发到端口 3。

当打开、关闭或者移动机器和网桥时，网络的拓扑结构会发生变化。为了处理这种动态的拓扑结构，一旦构造出一个哈希表项后，帧的到达时间也被记录在相应的表项中。当一帧到达时，如果它的源地址已经在表中，那么对应表项中的时间值被更新为当前时间。因此，与每个表项相关联的时间值反映了网桥最后看到该机器发出一帧的时间。

在网桥中有一个进程定期扫描哈希表，并且将那些时间值在几分钟以前的表项都清除掉。按照这种方法，如果将一台计算机从 LAN 上拔下来，然后搬到同一个楼内的另一个地方，再将它重新接入到网络中，那么几分钟之内该计算机就可以回到正常的运行状态，而无须任何人工干预。这个算法也意味着，如果一台计算机静止了几分钟时间（即几分钟之内不发送任何数据），那么任何发送给它的流量又将被泛洪，直到它下次发出一帧为止。

对于一个入境帧，它在网桥中的路由过程取决于它从哪个端口来（源端口），以及它要往哪个目标地址去（目标端口）。整个转发过程如下：

（1）如果去往目标地址的端口与源端口相同，则丢弃该帧。

（2）如果去往目标地址的端口与源端口不同，则转发该帧到目标端口。

（3）如果目标端口未知，则使用泛洪法，将帧发送到所有的端口，除了它入境的那个。

你可能想知道第一种情况是否会出现在点到点链接。答案是可能的，用集线器把一组计算机连到网桥。看图 4-41（b）中的一个例子，站 E 和 F 都连到集线器 H1，进而再连接

到网桥 B2。如果 E 发送一个帧给 F，集线器将中继该帧到 B2 以及 F。这就是集线器该做的事情——它们用有线把所有端口连在一起，这样从一个端口输入的帧只是输出到所有其他端口。该帧最终将从端口 2 到达 B2，这正是它到达目的地的正确输出端口。网桥 B2 只需丢弃该帧。

每到达一个帧，都要使用该算法，所以通常算法的实现采用了专用的大规模集成电路芯片。这种芯片查找和更新哈希表中的表项，所有这些只需几微秒的时间即可完成。因为网桥只要看到 MAC 地址就可决定如何转发帧，有可能一旦从入境线路输入帧的目的地址字段马上就可转发，此时帧的其余部分还在输入，即在完整接收一帧之前就开始了转发（当然，前提是输出线路可用）。这种设计降低了帧通过网桥的延迟，以及网桥必须能缓冲的帧数。这种转发方式称为**直通式交换**（cut-through switching）或**虫孔路由**（wormhole routing），通常由硬件处理。

我们可以按照协议栈来考察网桥的操作，从而理解为何把网桥看作是链路层设备。考虑图 4-41（a）的配置，其中局域网都是以太网。现在站 A 给站 D 发送一帧。该帧将穿过一个网桥。协议栈的处理如图 4-42 所示。

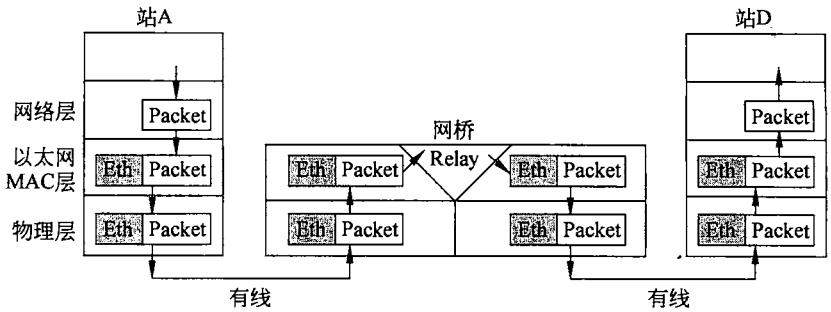


图 4-42 网桥上的协议处理

数据包来自一个更高的层次，下降进入以太网 MAC 层。它获取一个以太网头（还有一个尾，图中没有显示）组成传输单元。该单元被传到物理层，通过电缆传播，然后被网桥接收。

在网桥中，帧从物理层往上传给以太网的 MAC 层。相比普通站协议栈里的以太网 MAC 层，网桥的这一层扩伸了处理功能。它把帧传递到**中继**（relay）模块，该模块仍属于 MAC 层。网桥的中继功能仅仅使用了以太网的 MAC 头来确定如何处理帧。在这种情况下，它把帧传递给通往站 D 的那个端口的以太网 MAC 层，然后帧继续向前传输。

一般情况下，在给定层上中继时可以重写该层的头。VLAN 就提供了这样的一个例子。在任何情况下，网桥都不能查看帧的内部，以及了解帧携带的是否是一个 IP 数据包；这与网桥的处理毫不相关，而且违反了协议分层原则。还应该注意到，一个具有 K 个端口的网桥将有 k 个 MAC 和物理层实例。 在我们的简单示例中 k 值为 2。

4.8.3 生成树网桥

为了提高可靠性，网桥之间可使用冗余链路。在图 4-43 所示的例子中，在一对网桥之间并行设置了两条链路。这种设计可确保一条链路宕掉后，网络不会被分成两组计算机，

使得它们之间无法通信。

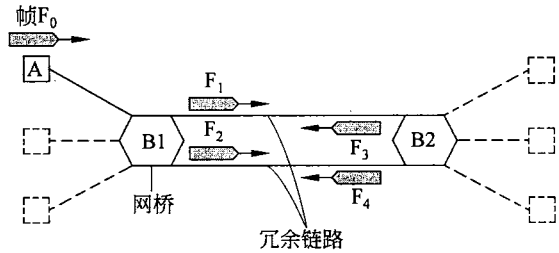


图 4-43　具有两条平行链路的网桥

然而，这种冗余引入了一些额外的问题，因为它生成了拓扑环路。这些问题我们可以用图 4-43 的例子来说明。考虑如果 A 给以前没有观察到的某个站发帧，这个帧将会如何被处理。每个网桥遵循着常规的处理规则：对于未知目的地的帧，泛洪该帧到所有其他端口。把从 A 到达网桥 B1 的帧称为 F_0。网桥把这个帧的副本发送到所有其他端口（除了它来的那个端口）。我们只考虑连接 B1 到 B2 的网桥端口（虽然帧还将被发送到其他端口）。因为有两条链路从 B1 连到 B2，因此 F_0 的两个副本将到达 B2。它们显示为图 4-43 中的 F_1 和 F_2。

很快，网桥 B2 接收到这些帧。然而，B2 不知道（也无法知道）这些帧是同一个帧的副本，它把它们当作两个前后到达的不同帧来处理。因此网桥 B2 将 F_1 副本发送到所有其他端口，同样把 F_2 的副本发送到所有其他端口。由此产生了 F_3 和 F_4，这两个帧又沿着两条链路发送回网桥 B1。然后 B1 看到两个未知目的地址的新帧，同样复制它们后泛洪。这个循环将会无限进行下去。

这个难题的解决途径是让网桥相互之间通信，然后用一棵可以到达每个网桥的生成树覆盖实际的拓扑结构。实际上，在构造一个虚拟的无环拓扑结构时网桥之间某些潜在的连接被忽略掉了，这个无环拓扑结构是实际拓扑结构的一个子集。

例如，在图 4-44 中我们看到 5 个网桥互联在一起，同时还有站与这些网桥连接。每个站只与一个网桥相连。在网桥之间有一些冗余连接，因此如果这些链路都用，帧就有可能沿着环路转发。我们可以将这种拓扑结构抽象成一个图，网桥为节点。点到点的链路是边。如图 4-44 所示，通过去掉一些链路（图中用虚线表示），整个图即被简化为一棵生成树，按照定义这树上没有环路。利用这棵生成树，从每个站到每个其他站恰好只有一条路径。一旦网桥同意这棵生成树，则站之间的所有转发都将沿着这棵树进行。由于从每个源到每个目标都只有唯一一条路径可走，所以不可能产生环路。

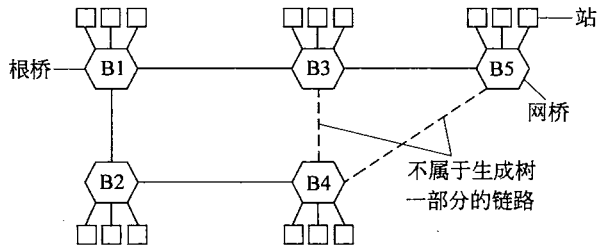


图 4-44　一棵连接 5 座网桥的生成树。虚线表示不属于生成树的链路

为了建立生成树，网桥运行一个分布式算法。每个网桥周期性地从它的所有端口广播一个配置消息给邻居，同时处理来自其他网桥的消息，处理过程如同下面描述的那样。这些消息不被转发，因为它们的用途是构建树，这棵树将被用于随后帧的转发。

全体网桥必须首先选择一个网桥作为生成树的根。为了作出这种选择，每个网桥在自己的配置消息中包含一个标识符，以及它们认为应该作为根的网桥的标识符。网桥标识符基于MAC地址，而MAC地址由网卡制造商预先设置好，能确保全世界独一无二。因此，网桥标识符的生成既便利又具有唯一性。网桥选择具有最低标识符的网桥成为生成树的根。经过足够的消息交换和扩散，最终所有网桥将都同意这个根。在图4-44中，网桥B1具有最低标识符，因而成为生成树的根。

然后，构造从根到每个网桥的最短路径树。在图4-44中，从网桥B1可直达网桥B2和B3，最短路径都是一跳。从根通过B2或B3都可到达B4，最短路径均为两跳。为了打破这种平局，选择一条经过具有最低标识符的网桥的路径，因此到达B4的路径通过B2。经网桥B3两跳可抵达网桥B5。

为了找到这些最短路径，网桥在它们的配置消息中还包括与根的距离。每个网桥记住它找到的到根的最短路径。然后，网桥关闭那些不属于最短路径一部分的端口。

虽然树跨越了所有的网桥，但并不是所有的链接（甚至桥）都必然要出现在树中。之所以关闭一些端口，是因为可以从网络中修剪掉某些链路，从而防止出现环路。即使在已建立生成树之后，在网络正常操作期间该算法也要继续运行，以便自动检测拓扑结构的变化，并更新生成树。

构造生成树的算法由Radia Perlman发明。她的工作是解决局域网无环连接问题。她有一个星期的时间来做这件事，但她一天就有了生成树算法的想法。幸运的是，这让她有足够的时间来写一首诗（Perlman，1985）：

I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span.
So packets can reach every LAN.
First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me
Then bridges find a spanning tree.

生成树算法随后被标准化为IEEE 802.1D，并被使用了许多年。在2001年该算法被修订，以便在拓扑变化后更为迅速地找到一棵新的生成树。如需详细的网桥处理资料，请参见（Perlman，2000）。

4.8.4　中继器/集线器/网桥/交换机/路由器和网关

到现在为止，我们在本书中已经考查过将帧或者数据包从一台计算机转移到另一台计算机的多种方法。我们也已经提到过中继器、集线器、网桥、交换机、路由器和网关。所有这些设备都有实际的应用价值，但是，它们的工作方式或多或少有些微妙或者明显的差别。由于有如此多种联网设备，所以我们值得考查它们在工作方式上的相似和不同之处。

理解这些设备的关键是它们运行在不同的层次上，如图 4-45（a）所示。之所以存在不同层的问题，是因为不同的设备使用不同的信息来决定如何交换。在典型场景中，用户生成某些数据，然后将这些数据发送给一台远程的机器。这些数据先被传递给传输层，传输层会加上一个头（比如 TCP 头），然后将结果单元往下传递给网络层。网络层也会加上一个头，形成一个网络层数据包（比如，形成一个 IP 包）。在图 4-45（b）中，我们看到，灰色阴影中的是 IP 分组。然后，该分组再往下到达数据链路层，数据链路层加上它自己的头和校验和（CRC），并将结果帧交给物理层传送出去，比如，通过一个 LAN 传送出去。

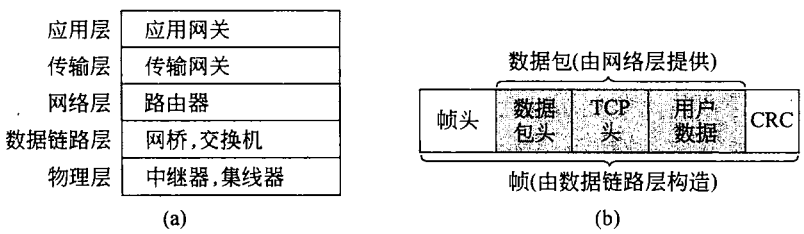


图　4-45
（a）设备的工作层次；（b）帧、数据包和头

现在我们考查交换设备，并了解它们与数据包和帧的关系。在最底层，即物理层中，我们可以看到有中继器。中继器是模拟设备，主要用来处理自己所连的线缆上的信号。在一个线缆上出现的信号被清理、放大，然后再被放到另一个线缆上。中继器并不理解帧、数据包或帧头，它们只知道把比特编码成电压的符号。例如，在经典以太网中，为了将电缆的最大长度从 500 米扩展到 2500 米，以太网允许最多使用 4 个中继器来增强信号。

接下来看集线器。集线器有许多条输入线路，它将这些输入线路连接在一起。从任何一条线路上到达的帧都被发送到所有其他的线路上。如果两帧同时到达，它们将会冲突，就好像它们在同一根同轴电缆上遇到后发生碰撞一样。连接到同一个集线器上的所有线路必须以同样的速度运行。集线器与中继器不同，它们（通常）不会放大入境信号，并且可以有多个输入线路，但是，两者之间的差别并不大。与中继器一样，集线器也是物理层设备，因而它不会检查链路层地址，也不以任何方式使用该地址。

现在让我们往上移到数据链路层，在那里我们看到有网桥和交换机。我们刚刚花了一定篇幅介绍了网桥。网桥连接两个或多个局域网。跟集线器一样，一个现代网桥有多个端口，通常具有 4～48 条某种类型的输入线。与集线器不同的是网桥的每个端口被隔离成它自己一个冲突域；如果端口是全双工的点到点线路，则需要用到 CSMA/CD 算法。当到达一帧时，网桥从帧头提取出帧的目的地址，并用该地址查询一张应该把帧发往哪里去的表。对于以太网，地址是 48 位的目标地址，如图 4-14 所示。网桥只把帧输出到所需要的端口，

在同一时间可转发多个帧。

网桥比集线器提供了更好的性能，隔离网桥端口还意味着输入线路可以不同的速度运行，甚至可以是不同的网络类型。一个常见的例子是网桥具有连接到10、100和1000 Mbps以太网的端口。从一个端口接受一帧并从另一个不同端口发送出去，需要网桥内部进行缓冲。如果帧的速度大于网桥的重发速度，网桥就可能耗尽缓冲空间从而开始不得不丢弃帧。例如，如果一个千兆以太网以最快的速度向一个10 Mbps以太网倾泻比特流，网桥将不得不缓冲这些比特，希望不会因此耗尽内存。即使所有的端口都以相同的速度运行，这个问题也仍然存在，因为有可能多个端口都往某个特定的目的端口发送帧。

网桥最初是被用来连接不同种类的局域网，例如，把一个以太网和令牌环网连接在一起。然而，由于不同局域网之间的差异，这方面的工作总是做得不是很好。不同的帧格式要求复制和重新格式化帧、需要计算新的校验和，这些处理都需要CPU时间，并且有可能引入由于网桥内存出错而导致差错控制出现错误。不同的最大帧长也是一个没有得到很好解决的严重问题。基本上，大到无法转发的帧必须被丢弃。总之有太多的透明性问题。

局域网之间的差异还可以体现在另外两个方面：安全性和服务质量。某些局域网使用了链路层的加密机制，例如802.11，而有的局域网却不具备任何安全性，例如以太网。某些局域网具备服务质量的特性，例如802.11，而有的却没有服务质量的概念，例如以太网。于是，当一个帧必须在这些局域网之间穿越时，发送方期待的安全性和服务质量可能无法得到保证。基于所有这些原因，现代网桥通常都工作在一种网络类型，然后用我们很快会看到的路由器来连接不同类型的网络。

交换机是现代网桥的另一个称呼。它们的差异更多地体现在市场上而不是技术方面。但有关交换机和网桥还是有几点值得了解。开发网桥时正是经典以太网被广泛使用之际，网桥倾向于连接相对数目较少的局域网，因而端口数也相对较少。现在“交换机”一词更为流行。此外，现代交换机的安装都使用了点到点链接（例如双绞线），单个计算机通过双绞线直接插入到交换机端口，因此交换机的端口数往往有许多个。最后，“交换机”也可作为一般术语使用。使用网桥，功能是明确的。另一方面，交换机可以指以太网交换机，也可以指一个完全不同类型的转发决策设备，例如电话交换机。

到现在为止，我们已经看过了中继器和集线器，以及网桥和交换机。其中，中继器和集线器非常类似，网桥和交换机也有许多相似之处。现在我们再往上转到路由器，它完全不同于前面提到的所有设备。当一个数据包进入到路由器时，帧头和帧尾被剥掉，帧的有效载荷字段中（如图4-45中的阴影部分）的数据包被传给路由软件。路由软件利用数据包的头信息来选择输出线路。对于一个IP数据包，包头将包含一个32位（IPv4）或者128位（IPv6）地址，而不是48位的IEEE 802地址。该路由软件看不到帧地址，甚至不知道数据包来自哪个LAN或哪条点到点线路。我们将在第5章学习路由器和路由算法。

再往上一层我们可以发现传输网关。它们将两台使用了不同面向连接传输协议的计算机连接起来。例如，假设一台计算机使用了面向连接的TCP/IP协议，另一台计算机使用了一个不同的面向连接传输协议——SCTP，现在它们需要通话。于是，传输网关将数据包从一个连接复制到另一个连接上，并且根据需要对数据包重新进行格式化。

最后，应用网关能理解数据的格式和内容，并且可以将消息从一种格式转换为另一种格式。例如，电子邮件网关可以将Internet邮件转译为移动电话的SMS消息。与“交换机”

一样、术语“网关”也是一个通用术语。它是指一个运行在较高层次的转发进程。

4.8.5　虚拟局域网

局域网联网的早期，在许多办公楼沿着电缆管道铺设了黄色的粗电缆。这些电缆把经过的每一台计算机接入到局域网中。不需要思考哪台计算机属于哪个 LAN。相邻办公室的所有人都被放在同一个 LAN 上，也不管他们是否应该属于同一个 LAN。那个时候，网络的地理位置超越了企业的组织结构。

随着 20 世纪 90 年代双绞线和集线器的出现，办公楼被重新布线（以相当昂贵的代价），去掉了所有的黄色庭院软管，从每个办公室到中心接线柜之间安装了许多双绞线。通常中心接线柜位于每个走廊的尽头或者在一个中心机房内，布线结构如图 4-46 所示。如果负责布线工作的副总裁卓有远见，就会选择安装 5 类双绞线；如果他是一个善于控制费用的专家，则他会使用现有的（3 类）电话线（只有当几年以后出现快速以太网，这些电话线才被替换掉）。

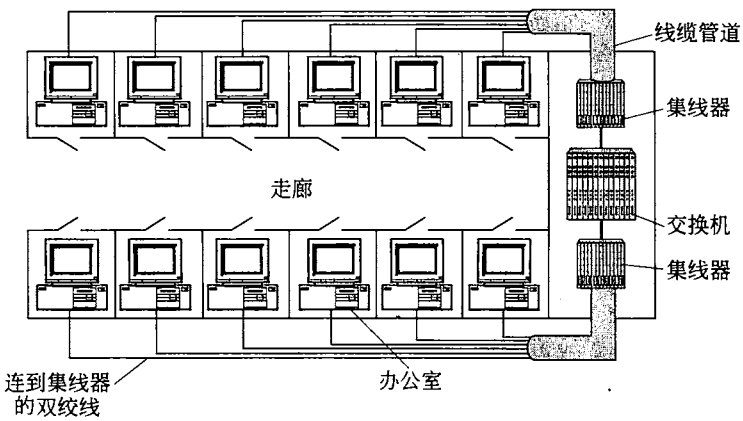


图 4-46　大楼使用了集线器和交换机的集中式布线

今天，线缆已经发生了变化，而且集线器变成了交换机，但布线模式依然如此。这种模式使得管理员有可能按照逻辑关系而不是物理位置来配置局域网。例如，如果一家公司想要 k 个 LAN，那么它只要购买 k 个集线器。通过谨慎地选择插入到哪一个集线器上，该公司可以按照特定的组织意义来构建每一个 LAN，而无须太多顾及地理位置。

谁连在哪一个 LAN 真的很重要吗？毕竟，在几乎所有的组织中，所有的 LAN 都是相互连接的。简单的回答是肯定的，谁在哪一个 LAN 很重要。网络管理员希望将 LAN 上的用户分成适当的组，以便反映出用户的组织结构，而不是大楼的物理布局结构。他们有很多个理由这么考虑。一个问题是安全性。一个 LAN 上可能驻扎着 Web 服务器，这个服务器供其他计算机公共访问；另一个 LAN 或许连接着包含人力资源部门记录的计算机，这些资料不能被流传到部门之外。在这种情形下，把所有计算机放在一个 LAN 中，并且不让任何服务器被 LAN 以外的用户访问是很有意义的。管理层如果听到网络管理员无法这样安排局域网的答复时一定会颇感不悦。

第二个理由是负载。某些 LAN 比其他 LAN 具有更重的负载，也许需要将它们隔离开

来。例如，研究人员在运行各种出色的实验程序时有可能会大量消耗网络资源，从而使他们的LAN流量达到饱和，这时正在开视频会议的管理部门人员可能并不愿意贡献出他们的容量来帮助研究部门。而且，这还会给管理部门留下需要安装一个更快网络的印象。

第三个理由是广播流量。网桥在未知目标地址时会广播，而且上层协议也会使用广播。例如，当一个用户希望将一个数据包发送给IP地址x时，它怎样知道应该在帧中放哪一个MAC地址呢？我们将在第5章讨论这个问题，现在先简单介绍一下解决方案。答案是它先广播一帧，该帧中包含了这样一个问题“谁拥有IP地址x？”，然后它等待应答的到来。随着LAN内计算机数目的增多，广播数目也随之增多。每次广播消耗的容量比一个常规帧消耗的容量多得多，因为这个广播流量要传递给LAN中的每台计算机。将LAN保持在不需要那么大的规模，可降低广播流量的影响。

与广播有关的一个问题是，一旦网络接口崩溃或被错误配置后，将会产生无休止的广播帧流。如果网络真的不走运，这类帧中一些会导致更多的流量。这种广播风暴（broadcast storm）的后果是：（1）整个LAN的容量全部被这些帧占用；而且（2）所有这些互连LAN上的所有机器将忙于处理和丢弃所有被广播的帧。

初看起来，利用网桥或者交换机将多个LAN隔离开可以将广播风暴限制在一定的范围内；但是，如果要想达到透明的目标（即一台计算机跨过网桥移动到另一个不同LAN中而不引起他人的注意），那么网桥必须要转发广播帧。

看清楚了为什么有些公司希望采用多个有独立限定范围的LAN后，我们回到原来的问题：将逻辑拓扑结构与物理拓扑结构脱离开。搭建一个反映组织结构的物理拓扑要增加工作量和成本，即使采用集中式布线和交换机。例如，两个同一部门的工作人员在不同的建筑物内上班，把他们接到不同LAN中的交换机非常容易；即使不是这样，一个工作人员在公司内部从一个部门被调动到另一个部门，但是没有换办公室；或者他没有变换部门，但是换了办公室。这可能导致用户出现在了错误的LAN上，除非管理员把他的连接器从一个交换机拔下来换插到另一个交换机。此外，属于不同部门的计算机数目或许不能很好匹配所在交换机的端口；某些部门太小，其他部门又太大，因此它们需要不同的交换机。这样可能导致交换机端口的浪费（它们没有被全部使用）。

在许多公司中，组织结构总是在不停地发生着变化，这意味着系统管理员要花大量的时间拔下连接器再插入到另外的交换机。而且，在某些情况下，这种改变甚至是根本不可能的，因为用户机器的双绞线离恰当的集线器太远了（比如在其他大楼中），或者可用交换机的端口在其他不恰当的LAN上。

为了响应用户对灵活性的需求，网络提供商开始考虑用软件方式对大楼重新进行布线。结果得到的概念称为虚拟LAN（VLAN，Virtual LAN）。IEEE 802委员会已经将VLAN标准化，而且现在已经被许多组织采纳。接下来我们将讨论VLAN。有关VLAN更多的信息，请参见（Seifert和Edwards，2008）。

VLAN基于VLAN感知（VLAN-aware）交换机。为了搭建一个基于VLAN的网络，网络管理员首先要确定共有多少个VLAN、哪些计算机位于哪个VLAN、每个VLAN叫什么名称。通常VLAN用颜色来命名（非正式的），因为这样做之后就有可能打印出一张色彩斑斓的图来显示机器的物理布局，其中红色LAN中的成员用红色表示，绿色LAN中的成员用绿色表示，以此类推。按照这种方式，在单个视图中可以同时显示局域网的物理布

局和逻辑布局。

作为例子考虑图 4-47 中的桥接 LAN，其中 9 台机器属于 G（灰色）VLAN，5 台机器属于 W（白色）VLAN。位于灰色 VLAN 中的机器可以分散在两个交换机上，包括通过一个集线器连到交换机上的两台机器。

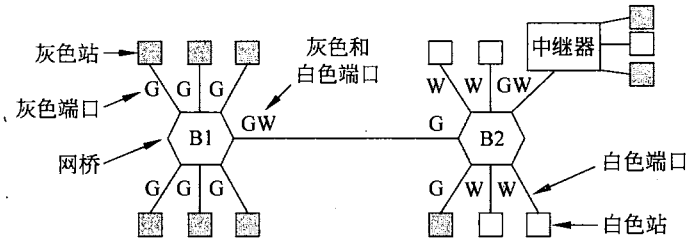


图 4-47　一个桥接 LAN 上的两个 VLAN，分别是灰色的和白色的

为了使 VLAN 正常地运行，网桥必须建立配置表。这些配置表指明了通过哪些端口可以访问到哪些 VLAN。当一帧到来时，比如说来自灰色 VLAN，那么这帧必须被转发到所有标记为 G 的端口。这一条规则对于网桥不知道目的地位置的普通流量（即单播）以及组播和广播流量都适用。注意，一个端口可以标记为多种 VLAN 颜色。

作为一个例子，假设插入在图 4-47 中网桥 B1 的一个灰色站给一个以前没有观察到的站发送一帧。网桥 B1 将接收此帧，发现它来自于灰色 VLAN 的一台机器，因此它在所有标有 G 的端口上泛洪帧（除了入境端口）。该帧将被发送到连在 B1 上的其他五个站，以及通过链路再发送到与之相连的 B2；在网桥 B2 上，帧同样被转发给标有 G 的所有端口；这样帧被进一步发到灰色站和集线器（集线器再将帧转交给它的所有站）。集线器有两个标签，因为它连接着分属于两个 VLAN 的机器。帧不会被发送到标记不是 G 的其他端口，因为网桥知道通过这些端口（非 G）到不了灰色 VLAN 上的机器。

在我们的例子中，帧从网桥 B1 发送到网桥 B2，因为有灰色 VLAN 上的机器与 B2 连接。再来考查白色 VLAN，我们可以看到网桥 B2 连到网桥 B1 的端口没有被标记 W，这意味着在白色 VLAN 上的帧将不会从网桥 B2 转发到网桥 B1。这种行为是正确的，因为没有白色 VLAN 上的机器被连接到 B1。

IEEE 802.1Q 标准

为了实现这个方案，网桥需要知道入境帧属于哪个 VLAN。如果没有这个信息，例如图 4-47 中，当网桥 B2 从网桥 B1 获取一个帧，它不知道该把帧转发到灰色还是白色 VLAN 上。如果我们在设计一个新型局域网时，要做到这点非常容易，只需在帧头添加一个 VLAN 字段。但对以太网该如何处理呢，毕竟它主宰着 LAN，并且没有留下任何空闲字段可用作 VLAN 标识符。

1995 年，IEEE 802 委员会终于将这个问题提到议事日程上。经过多次讨论之后，非常不可思议的是以太网的帧头被修改了。新的格式于 1998 年发表在 IEEE 标准 802.1Q 中。新格式包含了一个 VLAN 标签（tag）；稍后我们将会讨论该标签。毫不奇怪，改变一些事情，如同建立以太网头，并不是一件微不足道的事情。很快可以想到随之出现的一些问题包括：

（1）我们需要抛弃现有成千上百万的以太网卡吗？

（2）如果不抛弃这些网卡，谁来生成新的字段？

（3）对于那些已经达到最大长度的帧该怎么办？

当然，802委员会知道这些问题（只是太痛苦了），而且也必须拿出可行的解决方案，这是它的职责。

解决问题的关键是要认识到实际使用VLAN字段的只是网桥和交换机，用户机器不需要知道它们。因此，在图4-47中，重要的是它们出现在连接网桥的线路上，而不是连到最终站的线路上。另外，为了使用VLAN，网桥必须能感知VLAN。这一事实使得设计方案具备了实现的可行性。至于说是否要扔掉所有的现有以太网卡，答案是否定的。请记住，802.3委员会甚至无法让人们把类型字段转变成长度字段。你能想象当他们宣布所有现有的以太网卡都必须扔掉时公众的反应。然而，新的以太网卡与802.1Q兼容，并且能正确地填写VLAN字段。

因为存在一些计算机（和交换机）无法感知VLAN，因此第一个VLAN感知的网桥在帧上添加一个VLAN字段，路径上的最后一个网桥把添加的VLAN字段删除。混合拓扑的一个例子如图4-48所示。在图中，VLAN感知的计算机直接生成标记帧（即802.1Q），后续交换时要用到这些标记。阴影符号是VLAN感知的，空白符号是不能感知VLAN的。

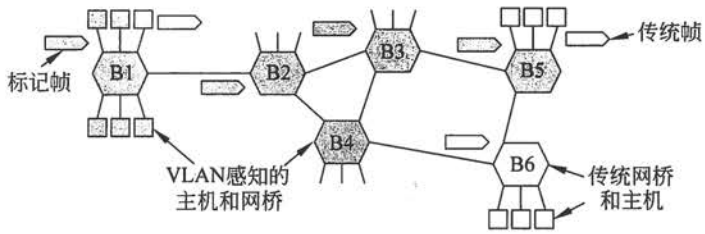


图4-48 只有部分VLAN感知的桥接LAN
阴影符号是VLAN感知的空白符号不能感知VLAN

有了802.1Q，帧被染上颜色，具体色彩取决于接收它们的端口。对于这种工作方法，一个端口上的所有机器必须属于同一个VLAN，这降低了灵活性。例如，在图4-47中，这个属性对于把单个计算机连到网桥的所有端口来说是成立的，但对于集线器连到网桥B2的端口来说就不成立了。

此外，网桥可以使用更高层协议来选择颜色。这样，到达一个端口的帧或许被放在不同的VLAN，要根据其是否携带IP数据包或者PPP帧而定。

其他方法也是可能的，但它们没有得到802.1Q的支持。举一个例子，可以用MAC地址来选择VLAN的颜色，这或许对来自附近802.11局域网的帧有用，当笔记本电脑在移动时通过不同的端口来发送帧。此时，把MAC地址映射到一个固定的VLAN，而不管它从哪个端口进入局域网。

至于帧的长度超过1518字节这个问题，802.1Q提出把帧长的限制提高到1522字节。幸运的是，只有VLAN感知的计算机和交换机必须支持这些长帧。

现在我们来看802.1Q帧的格式。如图4-49所示。唯一的变化是加入了一对2字节字段。第一个2字节是VLAN协议标识符（VLAN protocol ID），它的值总是0x8100。由于这个数值大于1500，因此，所有的以太网卡都会将它解释成类型（type），而不是长度（length）。对于这样的帧传统网卡做什么处理都没有意义，因为这样的帧是不会被发送给传

统网卡的。

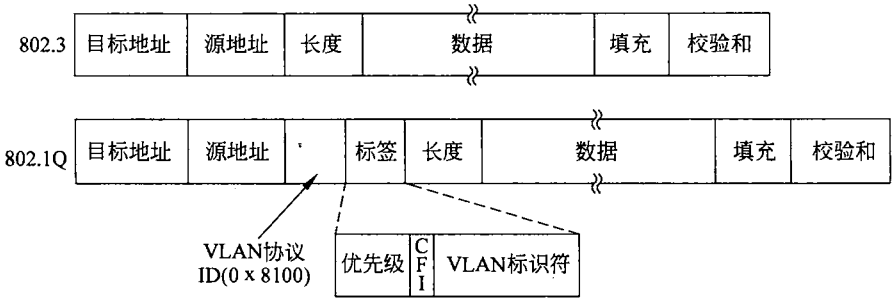


图 4-49　802.3（传统的）和 802.1Q 以太帧格式

第二个 2 字节包含三个子字段。最主要的是 VLAN 标识符（VLAN identifier）字段，它占用低 12 位。这正是整个事情的关键所在——帧属于哪种颜色的 VLAN。3 比特的优先级（Priority）字段与 VLAN 根本没有关系；但是，由于修改以太网头是 10 年才会发生一次的事件，而且这次事件花了三年时间，涉及上百人，因此为什么在做这件事情的时候不引入一些好的特性呢？这个字段使得交换设备有可能从时间敏感流量中进一步区分出硬实时流量与软实时流量，以及在以太网上提供更好的服务质量。以太网上的话音通信就需要服务质量保证（不过，公平地来讲，IP 有个类似的字段已经存在了四分之一世纪，但是几乎没有人使用它）。

最后一个字段，规范格式指示器（CFI，Canonical Format Indicator）应该称为企业自我指示器（CEI，Corporate Ego Indicator）。它最初的意图是用来指明 MAC 地址中比特的次序（little-endian 还是 big-endian），但是，在其他争论中这种用法渐渐被遗忘了。现在，这个比特用来指明有效载荷中是否包含一个冻干了的 802.5 帧。它希望通过以太网可以使得两个相隔遥远的 802.5 局域网用户进行通信，尽管消息需要以太网来承载。当然，这整个设想同样跟 VLAN 一点关系也没有。但是，标准委员会的政治学与常规的政治学也没有什么不同：如果你赞成我提出的比特，那么我也会给你的比特投票。

正如我们上面所提到的，当一个标记帧到达一个 VLAN 感知交换机，该交换机利用 VLAN 标识符作为索引，在一张表中查找帧要去哪些端口。但是，这张表从何而来？如果手工来构造，那么我们又回到了问题的起点：手工配置网桥。透明网桥的美丽之处就在于它即插即用，不要求任何手工配置。丢失这个特性简直是一个很可怕的耻辱。幸运的是，VLAN 感知网桥也可以根据它们观察到的标记自动进行配置。如果一个标记为 VLAN4 的帧来自端口 3，那么，很显然端口 3 上有台机器属于 VLAN4。802.1Q 标准说明了如何动态地建立这张表，其中主要引用了在 802.1D 标准中的恰当部分。

在结束 VLAN 路由主题之前，有一种现象值得最后提一下。Internet 和以太网领域中的许多人极其推崇无连接的网络技术，而极力反对任何在数据链路层或者网络层上建立连接的做法。然而，令人惊奇的是，VLAN 实际上引入了一种类似于连接的机制。为了正确使用 VLAN，每个帧携带一个新的特殊标识符，该标识符被当作索引用来查询交换机内部的一张表，从中找出这一帧要去的目标端口。这个过程与面向连接网络惊人的一致。在无连接的网络中，真正被用来路由的是目标地址，而不是某种形式的连接标识符。我们将在第 5 章更详细地讨论这种匍匐式联结。

4.9 本章总结

有些网络只有一个信道可用于全部的通信。在这些网络中，关键的设计问题在于如何在希望使用信道的竞争站之间分配信道。当站的数目较小并且固定，而且流量呈连续型时，FDM 和 TDM 是简单而有效的分配方案。这两种模式被广泛使用在这种情况下，例如，电话中继线带宽的划分。然而，当站的数目较大并且可变，或者流量呈现相当的突发时——正是计算机网络的一般情况，FDM 和 TDM 就是糟糕的选择。

许多动态信道分配算法被设计了出来。ALOHA 协议，包括分槽的与不分槽的，被用在许多实际系统的衍生物中，例如线缆调制解调器和 RFID。当可以侦听信道状态后，作为一种改进，站在其他站传输时可避免启动自己的传输。这种技术，即载波侦听，导致了局域网和城域网的 CSMA 各种协议。它是典型以太网和 802.11 网络的基础。

众所周知有一类协议能完全消除竞争，或者至少能大大减少了竞争。位图协议、拓扑结构（比如环）以及二进制倒计数协议完全消除了竞争。树遍历协议能减少竞争，具体做法是动态划分成两个大小不同的相邻组；并且只允许同一个组内的站竞争；最理想的组选择是当它允许发送时只有一个站要发送。

无线局域网提出了新的问题，就是很难侦听到传输冲突，而且站所覆盖的区域可能有所不同。在主宰无线局域网的 IEEE 802.11 中，站使用 CSMA/CA，通过留有很小的时间间隔来避免冲突，从而减轻第一个问题。站还可以使用 RTS/CTS 协议来对抗由于第二个问题引起的隐藏终端。IEEE 802.11 通常被用于把笔记本电脑和其他设备连接到无线接入点，但它也可以用在设备之间的自组织联网。任何一种物理层都可以被使用，包括有或没有多个天线的多信道频分复用，以及扩频技术。

跟 802.11 一样，RFID 读写器和标签使用随机访问协议来交换标识符。其他无线个域网和无线城域网有不同的设计。蓝牙系统可连接耳机和许多不同类型的外设到计算机而无须任何线缆。IEEE 802.16 为固定和移动计算机提供了广域的无线 Internet 数据服务。这两类网络都使用了一个集中并面向连接的设计，其中蓝牙主节点和 WiMAX 基站决定每个站何时可以发送或接收数据。对于 802.16 而言，这种设计可为诸如电话那样的实时流量和像 Web 浏览这样的交互式流量提供服务质量保障。至于蓝牙，复杂性放置在主节点上，从而导致从节点非常的价廉物美。

以太网是有线局域网的主要形式。经典以太网使用 CSMA/CD 来分配黄色花园软管内电缆的信道，这个软管在机器之间蜿蜒穿梭。随着速度由原来 10 Mbps 提升到 10 Gbps 并继续攀升，体系结构已经发生改变。现在，诸如双绞线那样的点到点链路连接着集线器和交换机。有了现代交换机和全双工链路，链接上已经不存在竞争，交换机可以在不同的端口之间并行地转发帧。

若要把局域网布满整个建筑物，就需要一种互连局域网的方法。即插即用的网桥就能用于此目的。搭建网桥时需要用到后向学习算法和生成树算法。由于此功能已被纳入现代交换机，因此术语“网桥”和“交换机”可以互换着使用。为了帮助桥接局域网的管理，VLAN 技术使得物理拓扑结构可划分为不同的逻辑拓扑结构。VLAN 标准，即 IEEE 802.1Q

协议，引入了一种新的以太网帧格式。

习　题

1. 在这个练习中，请使用本章中的一个公式，但在计算之前请先说明这个公式。帧随机到达一个 100 Mbps 信道，并等待传输。如果帧到达时信道正忙，那么它必须排队等待。帧的长度呈指数分布，均值为 10 000 位/帧。对于下列每一种帧到达率，试问平均一帧的延迟是多少（包括排队时间和传输时间）？
   （a）90 帧/秒。
   （b）900 帧/秒。
   （c）9000 帧/秒。
2. N 个站共享一个 56 kbps 的纯 ALOHA 信道。每个站平均每 100 秒输出一个 1000 位长的帧，即使前面的帧还没有被发送出去（比如，站可以将出境帧缓存起来）。试问 N 的最大值是多少？
3. 考虑在低负载情况下纯 ALOHA 和分槽 ALOHA 的延迟。试问哪个延迟更小？请说明你的答案。
4. 一大群 ALOHA 用户每秒钟产生 50 个请求，包括原始的请求和重传的请求。时间槽单位为 40 毫秒。
   （a）试问：第一次发送成功的机会是多少？
   （b）试问：恰好 k 次冲突之后成功的概率是多少？
   （c）试问：所需传输次数的期望值是多少？
5. 在一个有无限用户的分槽 ALOHA 系统中，一个站在冲突之后到重传之间的平均等待时间槽数目为 4。请画出该系统的延迟与吞吐量之间的关系图。
6. 试问在下列两种情况下 CSMA/CD 的竞争时间槽长度是多少？
   （a）一个 2 千米长的双导电缆（twin-lead cable）（信号的传播速度是信号在真空中传播速度的 82%）？
   （b）40 千米长的多模光纤（信号的传播速度是信号在真空中传播速度的 65%）？
7. 在一个使用基本位图协议的局域网中，最坏的情况下一个站（比如 s）要等多久才可以传输它的帧？
8. 在二进制倒计数协议中，试问为什么一个编号较低的站有可能得不到发送数据包的机会。
9. 编号为 1～16 的 16 个站使用自适应树遍历协议来竞争一个共享信道。如果所有站的地址预先设定，并且突然一次就绪，试问解决竞争需要多少个比特槽？
10. 考虑 5 个无线站：A、B、C、D 和 E。站 A 可与所有其他站通信。B 可以与 A、C 和 E 通信。C 可以与 A、B 和 D 通信。D 可以与 A、C 和 E 通信。E 可以和 A、D 和 B 通信。
    （a）当 A 给 B 发送时，试问可能进行的其他通信是什么？
    （b）当 B 给 A 发送时，试问可能进行的其他通信是什么？
    （c）当 B 给 C 发送时，试问可能进行的其他通信是什么？

11. 6个站的编号从A到F，它们使用MACA协议进行通信。试问有可能同时发生两个传输操作吗？请说明你的答案。
12. 一个七层办公楼的每一层有15个相邻的办公室。每个办公室的前面墙上包含一个终端插口。所以，在垂直面上，这些插口构成了一个矩形网格，在水平方向和垂直方向上插口之间均有4米远的距离。假定在任何一对插口之间，无论是水平的、垂直的，或是对角的，都可以直接拉一根线缆，试问若使用下面的配置需要多少米线缆才能将所有的插口连接起来：
    （a）正中间放置一台路由器的星型结构。
    （b）经典802.3 LAN。
13. 试问经典10 Mbps以太网的波特率是多少？
14. 假设经典以太网使用曼彻斯特编码，请画出比特流0001110101的编码输出。
15. 一个1千米长、10 Mbps的CSMA/CD LAN（不是802.3），其传播速度为200米/微秒。这个系统不允许使用中继器。数据帧的长度是256位，其中包括32位的头、校验和以及其他开销。在一次成功传输后的第一个比特槽被预留给接收方，以便它抓住信道发送32位的确认帧。假定没有冲突，试问除去开销之后的有效数据率是多少？
16. 两个CSMA/CD站都企图传送大文件（多个帧）。每发出一帧，它们就使用二进制指数后退算法竞争信道。试问在第k轮结束竞争的概率是多少？每个竞争周期的平均次数是多少？
17. 一个通过以太网传送的IP数据包长60字节，其中包括所有的头。如果没有使用LLC，试问需要往以太网帧中填补字节吗？如果需要，试问需要填补多少个字节？
18. 以太网帧必须至少64字节长，才能确保当电缆另一端发生冲突时，发送方仍处于发送过程中。快速以太网也有同样的64字节最小帧长度限制，但是它可以快10倍的速度发送数据。试问它如何有可能维持同样的最小帧长度限制？
19. 有些书将以太网帧的最大长度说成是1522字节而不是1500字节。它们错了吗？请说明你的回答。
20. 试问千兆以太网每秒钟能够处理多少个帧？请仔细想一想，并考虑所有相关情形。提示：请考虑千兆以太网的实质。
21. 请说出两个网络，它们允许将多个连续的帧背靠背地打包在一起。试问为什么这个特性值得专门提出来？
22. 在图4-27中有4个站A、B、C和D。试问你认为后两个站中哪一个最接近A？为什么？
23. 试举例说明802.11协议中的RTC/CTS与MACA协议有哪点不同。
24. 一个无线局域网内有一个AP和10个客户站。4个站的数据速率为6 Mbps，另外4个站有18 Mbps的数据速率，最后两个站有54 Mbps的数据速率。试问当全部10个站一起发送数据，并且下列条件成立时，每个站能获得的数据速率是多少？
    （a）没有用TXOP。
    （b）采用了TXOP。
25. 假设一个11 Mbps的802.11b LAN正在无线信道上传送一批连续的64字节帧，比特错误率为10^{-7}。试问平均每秒钟将有多少帧被损坏？
26. 一个802.16网络有一个20 MHz宽的信道。试问可以多大的bps给固定用户站发送？

27. 为什么有些网络用纠错码而不用检错和重传机制？请给出两个理由。
28. 请分别列出 WiMAX 与 802.11 类似的两种方法和它与 802.11 不同的两种方法。
29. 从图 4-34 中，我们可以看到一个蓝牙设备可同时位于两个微网中。试问有理由说明为什么一个设备不可能同时是这两个微网中的主节点？
30. 试问在基本速率下，一个 3 槽蓝牙帧的数据字段最大长度是多少？请解释你的答案。
31. 图 4-24 显示了几种物理层协议。试问这些协议中哪个最接近蓝牙物理层协议？它们之间的最大差异是什么？
32. 在 4.6.6 节中提到，基本速率下一个 1 槽帧重复编码后的效率约为 13%，试问基本速率下一个 5 槽帧重复编码后的效率是多少？
33. 在 802.11 的跳频扩展频谱变种中，信标帧包含了停留时间。试问你认为蓝牙中类似的信标帧也包含了停留时间吗？请讨论你的答案。
34. 假设有 10 个 RFID 标签围绕在 RFID 读写器的周围。试问最好的 Q 值是多少？在给定的槽内一个标签无冲突响应的可能性有多大？
35. 请列出 RFID 系统的一些安全隐患。
36. 一个专门为快速以太网设计的交换机有一个传输速率为 10 Gbps 的背板。试问在最差情况下它可以多大的帧/秒来处理帧？
37. 请简单描述存储-转发型交换机和直通型交换机之间的区别。
38. 考虑图 4-41（b）用网桥 B1 和 B2 连接的扩展局域网。假设两个网桥的哈希表是空的。对于下面的数据传输序列，请列出转发数据包所用的全部端口：
    （a）A 发送一个数据包给 C。
    （b）E 发送一个数据包给 F。
    （c）F 发送一个数据包给 E。
    （d）G 发送一个数据包给 E。
    （e）D 发送一个数据包给 A。
    （f）B 发送一个数据包给 F。
39. 从损坏帧的角度来看，存储-转发型交换机比直通型交换机更有优势。请说明这种优势是什么。
40. 本章 4.8.3 节中提到，一些网桥甚至可能不会出现在生成树种。请描绘一个场景，其中一个网桥可能无法出现在生成树中。
41. 为了使得 VLAN 正常工作，在网桥内部需要有相应的配置表。如果图 4-47 中的 VLAN 使用集线器而不是交换机，情况会怎么样呢？集线器也需要配置表吗？为什么需要，或者为什么不需要？
42. 在图 4-48 中，右侧传统终端域中的交换机是一个 VLAN 感知交换机。试问在那里有可能使用传统的交换机吗？如果可能，试问它如何工作？如果不可能，请问为什么？
43. 请编写一个程序来模拟以太网上 CSMA/CD 协议的行为：当一帧正在被发送时，有 N 个站都准备要发送。你的程序应该报告每一个站成功开始发送帧的时间。假设每个时间槽（51.2 微秒）时钟滴答一次，并且冲突检测和发送干扰序列只需要一个时间槽。所有帧都具有最大允许的长度。

第5章 网 络 层

网络层关注的是如何将源端数据包一路送到接收方。为了将数据包送到接收方，可能沿途要经过许多跳（hop）中间路由器。这种功能显然与数据链路层的功能不同，数据链路层的目标没那么宏伟，只是将帧从线路一边传送到另一边。因此，网络层是处理端到端数据传输的最底层。

为了实现这个目标，网络层必须知道网络拓扑结构（即所有路由器和链路的集合），并从中选择出适当的路径，即使是大型网络也要选出一条好路经。同时，网络层还必须仔细选择路由器，避免某些通信线路和路由器负载过重，而其他线路和路由器空闲。最后，当源端和接收方位于不同网络时，还会出现新的问题，这些问题都需要由网络层来解决。在本章，我们将讨论所有这些问题，并主要结合 Internet 及其网络层协议（IP）来深入探讨。

5.1 网络层的设计问题

在本节，我们将简要描述网络层设计人员必须重视的一些问题，其中包括为传输层提供的服务以及网络的内部设计。

5.1.1 存储转发数据包交换

在介绍网络层细节之前，有必要再次说明网络层协议运行的上下文。图 5-1 大致勾画出了这样的环境。网络中最主要的组件是网络服务提供商（ISP）的设备（通过传输线路连接的路由器）和客户端设备，在图中 ISP 的设备位于阴影椭圆内，而客户设备位于椭圆之外。主机 H1 直接连接到 ISP 的路由器 A，这或许是一台家用计算机，通过 DSL 调制解调器接入；而 H2 则位于一个局域网内，这可能是一个办公室以太网，其上还有台路由器 F，客户拥有这台路由器并负责其运行。路由器 F 通过一条租用线路连接到 ISP 的设备上。然而，为了本章的目的，还是把客户端的路由器作为 ISP 网络的一部分来考虑，因为它们运行的算法与 ISP 路由器上运行的算法相同（我们本章关注的就是算法）。

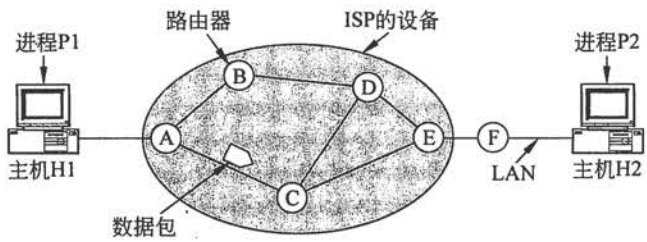


图 5-1 网络层协议的环境

这种网络配置的使用方式如下所述。如果一台主机要发送一个数据包，它就将数据包

传输给最近的路由器，路由器可能在它自己的 LAN 上，也可能在一条通向 ISP 的点到点链路上。在该数据包到达路由器，并且路由器的链路层完成了对它校验和的验证之后，它先被存储在路由器上；然后沿着路径被转发到下一个路由器，直至到达目标主机，这里就是数据包的目的地。这种机制即为存储-转发数据包交换，正如在前面几章已经看到的那样。

5.1.2 提供给传输层的服务

网络层通过网络层/传输层接口向传输层提供服务。一个重要的问题是明确网络层向传输层提供什么类型的服务。在设计网络层服务时，一定要牢记下面这些目标：

（1）向上提供的服务应该独立于路由器技术。

（2）应该向传输层屏蔽路由器的数量、类型和拓扑关系。

（3）传输层可用的网络地址应该有一个统一编址方案，甚至可以跨越 LAN 和 WAN。

给定这些目标后，网络层设计者有很大的自由度来编写提供给传输层的详细服务规范。这种自由度通常演变为两个竞争派别之间的激烈争斗。最终讨论的焦点集中在网络层应该提供面向连接的服务还是提供无连接的服务。

一个阵营（以 Internet 社团为代表）认为，路由器的任务仅仅是传送数据包，不用再做别的事情。按照他们的观点（基于 40 年来从一个实际计算机网络获得的经验），不管如何设计网络，从本质上讲它总是不可靠的。因此，主机应该接受这样的事实，自己来完成错误控制（即错误检测和纠正）和流量控制任务。

这种观点很快地导致了这样的结论：网络服务应该是无连接的，只需要原语 SEND PACKET 和 RECEIVE PACKET，以及少量其他的原语就够了。特别是，数据包的排序和流量控制不应该在这里完成，因为主机将会完成这些工作，做两遍同样的工作通常不会带来更多好处。这个推理就是端-端论点的例子，这种设计理念对 Internet 形成有着很大的影响。而且，每个数据包必须携带完整的目标地址，因为每个数据包的运送独立于它前面的那些数据包（如果此前有数据包的话）。

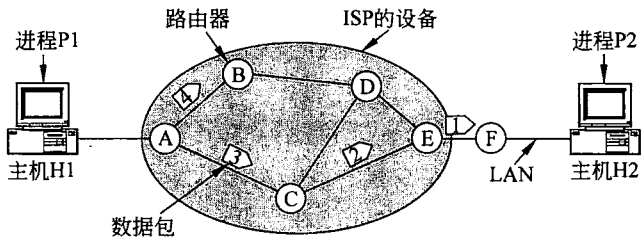
另一大阵营（以电话公司为代表）认为，网络应该提供可靠的、面向连接的服务。他们声称，具有 100 多年成功经验的全球电话系统就是一个极好的范例。按照他们的观点，服务质量是最主要的因素，并且如果在网络中没有连接，要实现服务质量非常困难，特别对于诸如语音和视频这样的实时流量。

即使几十年以后，这场争论仍然十分活跃。早期被广泛使用的数据网络都是面向连接的，比如 20 世纪 70 年代的 X.25 和 80 年代其继任者**帧中继**（Frame Relay）。然而，自从有了 ARPANET 和早期 Internet，无连接网络层得到了突飞猛进的普及。现在 IP 协议俨然是一个无处不在的成功象征。虽然在 20 世纪 80 年代，它受到一种叫做 ATM 的面向连接技术的威胁，该技术就是为了推翻 IP 而开发的；结果刚好相反，现在 ATM 终于找到了自己的用武之地，而 IP 正在接管电话网络。然而，在幕后，Internet 正朝着面向连接的特性进化，因为服务质量变得越来越重要了。与此相关的两个面向连接技术的例子是多协议标签交换（MPLS，MultiProtocol Label Switching）和 VLAN，我们将在本章描述 MPLS，有关 VLAN 我们在第 4 章中已经介绍过了。目前这两种技术已经被广泛使用。

5.1.3 无连接服务的实现

在考查了网络层所提供的两类服务之后，我们现在来看网络层内部是如何工作的。根据提供的服务类型，可能存在两种不同的组织方式。如果提供的是无连接的服务，那么，所有的数据包都被独立地注入到网络中，并且每个数据包独立路由，不需要提前建立任何设置。在这样的上下文中，数据包通常称为**数据报**（datagram），它类似于电报（telegram），对应的网络称为**数据报网络**（datagram network）。如果使用了面向连接的服务，那么，在发送数据包之前，必须首先建立起一条从源路由器到目标路由器之间的路径。这个连接称为**虚电路**（VC，virtual circuit），它类似于电话系统中建立的物理电路，对应的网络称为**虚电路网络**（virtual-circuit network）。在本节，我们将讨论数据报网络；在下节，我们再讨论虚电路网络。

现在我们来看数据报网络是如何工作的。假设图 5-2 中的进程 P1 有一个很长的消息要发送给 P2。它将消息递交给传输层，并指示传输层将消息传送给主机 H2 上的进程 P2。传输层代码运行在 H1 上，通常在操作系统内部。它在消息的前面加上一个传输头，然后将结果交给网络层，这里的网络层可能是操作系统内部的另一个过程。



A的表(初始化)

A	—
B	B
C	C
D	B
E	C
F	C

A的表(稍后)

A	—
B	B
C	C
D	B
E	B
F	B

C的表

A	A
B	A
C	—
D	E
E	E
F	E

E的表

A	C
B	D
C	C
D	D
E	—
F	F

图 5-2 数据报网络中的路由过程

让我们假设在这个例子中消息的长度是最大数据包长度的 4 倍，所以，网络层必须将消息拆分成 4 个数据包：1、2、3 和 4，然后用某种点到点协议（比如 PPP）将这些数据包依次发送给路由器 A。到这里，ISP 将消息的传输任务接管过来。每一台路由器都有一个内部表，它指明了针对每一个可能的目标地址应该将数据包送到哪里去。每个表项由两部分数据组成：目标地址和通往目标地址所使用的出境线路。当然，只能使用直接连接的线路。例如，在图 5-2 中，A 只有两条出境线路——分别通向 B 和 C，所以，每一个入境数据包必须被转发给这两台路由器之一，即使它最终目标地址是其他某一台路由器。A 的初始路由表如图中标示的“初始化”（initially）。

在路由器 A，数据包 1、2 和 3 分别到达入境线路并且经过验证校验和之后，被路由器暂时保存起来。然后，根据 A 上的表，每个数据包被放在一个新帧中，并且被转发到通往

C 的出境链路；之后数据包 1 被转发给 E，进一步又被转发给 F。当它到达 F 时，它被封装在一个帧内通过连有 H2 的 LAN 被发送出去。数据包 2 和 3 遵循同样的路径。

然而，数据包 4 的情形有所不同。当它到达 A 之后，尽管它的目标地址也是指向 F，但它被 A 转发给了路由器 B。出于某种原因，A 决定采用不同于前三个数据包的路径来发送数据包 4。或许它了解到在 ACE 路径上发生了流量拥塞，因而更新了路由表，如图中标示的“稍后”（later）。管理这些路由表并做出路由选择的算法称为**路由算法**（routing algorithm）。路由算法是我们本章学习的一个主要议题，正如我们将会看到的那样，有几种不同类型的路由算法。

IP 协议（Internet Protocol）是整个 Internet 的基础，它是无连接网络服务的重要范例。每个数据包携带一个目标 IP 地址，路由器使用该地址来单独转发每一个数据包。IPv4 数据包的地址是 32 位，IPv6 数据包的地址是 128 位。我们在本章后面部分将详细描述 IP。

5.1.4 面向连接服务的实现

对于面向连接的服务，我们需要一个虚电路网络。现在我们讨论它是如何工作的。隐藏在虚电路背后的思想是避免为每个要发送的数据包选择一条新路径（像图 5-2 那样）。相反，当建立一个连接时，从源机器到目标机器之间的一条路径就被当作这个连接的一部分确定了下来，并且保存在这些中间路由器的表中。所有需要在这个连接上通过的流量，都使用这条路径，这与电话系统的工作方式完全一致。当连接被释放之后，虚电路也随之消失。在面向连接的服务中，每个数据包包含一个标识符，指明了它属于哪一条虚电路。

作为一个例子，请考虑图 5-3 的情形。在这里，主机 H1 已经建立了一条与主机 H2 之间的连接 1。这条连接被记录在每个路由表中的第一项中。A 路由表的第一行说明如果一个标示了连接标识符 1 的数据包来自于 H1，那么它将被发送到路由器 C，并且赋予连接标识符 1。类似地，C 路由表中的第一项将该数据包路由到 E，也赋予连接标识符 1。

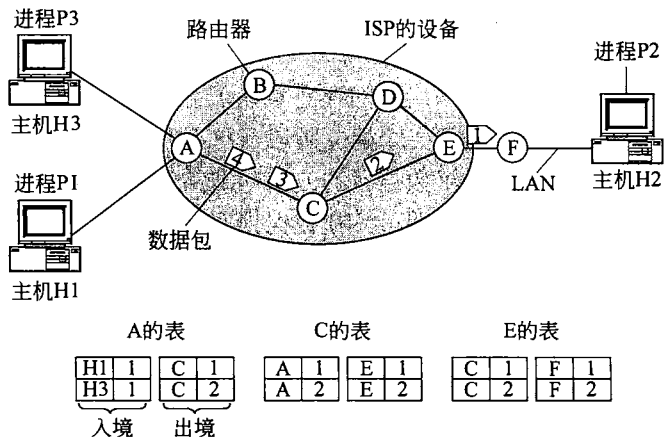


图 5-3 虚电路网络的路由过程

现在我们来考虑如果 H3 也希望与 H2 建立连接则情形会怎么样。H3 选择连接标识符 1（因为是它发起连接，而且这是它唯一的连接），并且告诉网络要建立虚电路。因此路由表增加了第二行。请注意，这里有一个冲突，因为，尽管 A 很容易区分出标识连接 1 的数

据包是来自H1还是来自H3，但是，C无法区分它们。基于这个原因，A给第二个连接的出境流量分配一个不同的连接标识符。这种避免冲突的做法说明了为什么路由器需要具备替换出境数据包中连接标识符的能力。

在有些上下文中，这个过程称为**标签交换**（label switching）。一种面向连接的网络服务例子是**多协议标签交换**（MPLS，MultiProtocol Label Switching），它主要被用在Internet的ISP网络，IP数据包被一个有20位连接标识或标签的MPLS头包裹着。MPLS往往对客户端是隐藏的，客户看不到这些标签，ISP用它来为超大流量建立长期的连接；但是，当服务质量变得很重要而且还需要协助其他ISP完成流量管理任务时，MPLS的作用越来越突出。我们将在本章后面更多地介绍有关MPLS的细节。

5.1.5 虚电路与数据报网络的比较

虚电路和数据报都有各自的支持者和反对者。我们现在试图从多个角度对两组论点作个总结。图5-4列出了最主要的问题，尽管纯粹主义者总是有可能找到图中每一项的反例。

问题	数据报网络	虚电路网络
电路建立	不需要	需要
寻址	每个包包含全部的源和目标地址	每个包包含简短的VC号
状态信息	路由器不保留连接状态	针对每个连接，每条VC都需要路由器保存其状态
路由方式	每个数据包被单独路由	建立VC时选择路由，所有包都遵循该路由
路由器失效的影响	没影响，除了那些路由器崩溃期间丢失的包	穿过故障路由器的所有VC都将中断
服务质量	困难	容易，如果在预先建立每条VC时有足够的资源可分配
拥塞控制	困难	容易，如果在预先建立每条VC时有足够的资源可分配

图5-4 数据报网络和虚电路网络的比较

在网络内部，数据报和虚电路网络之间存在着几个方面的权衡。一个权衡在于建立时间和地址解析时间。使用虚电路需要一个建立阶段，这个阶段既花费时间也消耗资源。然而，一旦付出了这个代价，处理一个数据包的方法却非常简单：路由器只要使用电路号作为索引，在表中找到该数据包的去向即可。在数据报网络中，不需要建立电路，但路由器需要执行一个更为复杂的查找过程以便找到目标表项。

与此相关的问题是数据报网络所用的目标地址比虚电路网络所用的电路号要长，因为数据报网络的目标地址具备全局意义。如果数据包相当短，在每个数据包中都包括完整的目标地址可能意味着大量的协议开销，因而造成带宽资源的浪费。

另一个问题是路由器内存所要求的表空间的数量。在数据报子网中，针对每一个可能的目标地址都要求有一个表项，而在虚电路网络中，只要为每一条虚电路提供一个表项即可。然而，这种优势有点虚幻并非绝对，因为虚电路网络在建立连接阶段所用的数据包也需要被路由，并且它们也使用目标地址，如同数据报网络的做法一样。

从保证服务质量以及避免网络拥塞的角度来看，虚电路有一定的优势，因为在建立连

接时，资源可以提前预留（比如缓冲区空间、带宽和 CPU 周期）。一旦数据包开始到来，所需要的带宽和路由器容量都已经准备就绪。而对于数据报网络，避免拥塞更困难些。

对于事务处理系统（比如商场购物时通过电话验证信用卡的有效性），用于建立和清除虚电路所需要的开销有可能会削弱虚电路的优势。如果系统中大部分流量都是这种类型，那么在网络内部使用虚电路就毫无意义。另一方面，公司的两个办公楼之间长期运行诸如 VPN 流量，这种情况下永久性的虚电路或许更加有用（可手工建立虚电路，并且持续使用几个月或者几年）。

虚电路也存在脆弱性问题。如果一台路由器崩溃并且内存中的数据全部丢失，那么即使它一秒钟之后又重新启动，所有从它这里经过的虚电路都将不得不中断。相反，如果一台数据报路由器宕机，只有那些当时尚留在路由器队列中的数据包用户会受到影响（甚至这些用户也不会全部受到影响，因为发送方可能很快重传这些数据包）。一条通信线路的失败对于使用了该线路的虚电路来说是致命的，但如果使用数据报，则这种失败很容易得到弥补。数据报还允许路由器平衡网络流量，因为一个长序列数据包的传输路径可以在序列传输的中途改变。

5.2 路 由 算 法

网络层的主要功能是将数据包从源机器路由到目标机器。在大多数网络中，数据包需要经过多跳（hop）才能到达目的地。唯一一个值得指出的例外是广播网络，但即使在广播网络中，如果源机器和目标机器不在同一个网络段中时，路由仍然是一个问题。选择路由的算法以及这些算法所用的数据结构是网络层设计的最主要内容。

路由算法（routing algorithm）是网络层软件的一部分，它负责确定一个入境数据包应该被发送到哪一条输出线路上。如果网络内部使用了数据报，那么路由器必须针对每一个到达的数据包重新选择路径，因为自上一次选择了路径之后，最佳路径可能已经发生了改变。如果网络内部使用虚电路，那么只有当建立一条新的虚电路时，才需要做路由决策；此后，数据包只要沿着已经建立的路径向前传递即可。后一种情形有时候也称为**会话路由**（session routing），因为在整个会话过程中（比如 VPN 上的一个终端登录会话），路径必须保持有效。

有的时候对路由和转发这两个功能进行区分是非常有用的，路由即对使用哪一条路径做出决策，而转发则是当一个数据包到达时该采取什么动作。可以把路由器想象成内部有两个进程。其中一个进程在每个数据包到达的时候对它进行处理，它在路由表中查找该数据包所对应的出境线路。这个进程即为**转发**（forwarding）进程；另一个进程负责生成和更新路由表，这正是路由算法发挥作用的地方。

无论是针对每个数据包独立选择路由，还是仅在建立新连接时选择路径，路由算法必须满足某些特性：正确性、简单性、鲁棒性、稳定性、公平性和有效性。正确性和简单性无须多加解释，但乍一看对鲁棒性的要求则没有那么显而易见了。一旦一个重要网络投入运行，它有可能需要连续运行数年而不能出现波及系统范围的失败。在此期间，将会有各种各样的硬件和软件发生故障。主机、路由器和线路可能会重复失败，网络拓扑结构也可能

会多次发生变化。路由算法应该能够处理拓扑结构和流量方面的各种变化，而且不能要求所有主机都停止所有的工作。可以想象每次路由器崩溃时都需要网络重新启动的严重后果！

稳定性对于路由算法来说也是一个重要目标。有一些路由算法无论运行多长时间从来都没有收敛到一个固定的路径集合。一个稳定的算法能达到平衡，并且保持平衡状态。因此它应该迅速收敛，在路由算法达到平衡之前，通信可能无法正常进行。

公平性和有效性听起来显然是理所应当的——肯定不会有人反对这两种特性。但是，事实证明它们往往是两个相互矛盾的目标。举一个简单的例子来说明这种冲突，请看图 5-5。假设在 A 和 A′之间、B 和 B′之间以及 C 和 C′之间有足够的流量使得水平的链路达到饱和。为了使总流量达到最大，X 和 X′之间的流量应该完全被切断。不幸的是，X 和 X′可能看不到这一点。很显然，在全局效率和单个连接的公平性之间必须有一种折中的处理办法。

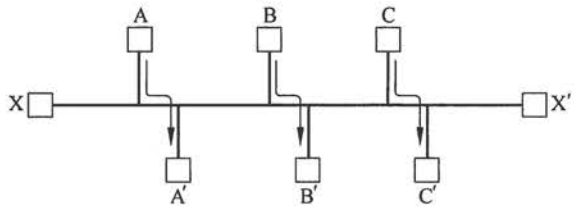


图 5-5 公平性和效率产生冲突的网络

我们在试图找到公平性和有效性之间的权衡办法之前，首先必须确定要优化什么性能指标。使数据包的平均延迟达到最小是有效发送流量的一种很明显的选择，但是使网络的总吞吐量最大化也是一种不错的选择。而且，这两个目标也是相互冲突的，因为运行任何一个接近容量的排队系统意味着排队延迟很长。作为一种折中，许多网络企图最小化一个数据包必须经过的跳数或简单降低其经历的跳数。两种选择趋向于减小延迟，同时减少每个数据包消耗的带宽数量，从而来提高了整个网络的吞吐量。

路由算法可以分成两大类：非自适应算法和自适应算法。**非自适应算法**（nonadaptive algorithm）不会根据当前测量或者估计的流量和拓扑结构，来调整它们的路由决策。相反，从 I 到 J（对所有的 I 和 J）所使用的路由选择是预先在离线情况下计算好，并在网络启动时被下载到路由器中的。这个过程有时候也称为**静态路由**（static routing）。因为它无法响应故障，所以静态路由对于路由选择已经很清楚的场合非常有用。例如，在图 5-3 中，路由器 F 应该通过路由器 E 把数据包发到网络中，根本不管数据包的最终目的地在哪里。

与此相反，**自适应算法**（adaptive algorithm）则会改变它们的路由决策以便反映出拓扑结构的变化，通常也会反映出流量的变化情况。这些动态路由（dynamic ronting）算法在多个方面有所不同：获取信息的来源不同（例如，来自于本地、相邻路由器，或者所有的路由器）、改变路径的时间不同（比如，每当拓扑发生变化时，或者每隔 Δt 秒随负载变化）以及用于路由优化的度量不同（比如，距离、跳数或者估计的传输时间）。

在下面的章节中，我们将讨论不同的路由算法。算法除了从源端发送数据包到接收方外，还涉及传递模式。有时候路由的目标是把数据包发送到多个地址、全部地址或者一组目标地址中的一个。我们在这里描述的所有路由算法都基于拓扑结构进行路由决策；我们把基于流量水平做路由决策的可能性推迟到 5.3 节介绍。

5.2.1 优化原则

在讨论具体的算法之前，我们先不考虑网络拓扑结构和流量情况，而是给出最优路径的一般性论述，这有助于后续学习。这个论述称为**最优化原则**（optimality principle）（Bellman，1957）。最优路径的一般陈述如下：如果路由器 J 在从路由器 I 到路由器 K 的最优路径上，那么从 J 到 K 的最优路径也必定遵循同样的路由。为了更好地理解这点，我们将从 I 到 J 的路径部分记作 r_1，余下的路径部分记作 r_2。如果从 J 到 K 还存在一条路由比 r_2 更好，那么，它可以与 r_1 级联起来，从而可以改善从 I 到 K 的路由，这与 r_1r_2 是最优路径的假设相违背。

作为最优化原则的一个直接结果，从所有的源到一个指定目标的最优路径的集合构成了一棵以目标节点为根的树。这样的树称为**汇集树**（sink tree），如图 5-6（b）所示，图中的距离度量是跳数。所有路由算法的目标是为所有路由器找到这样的汇集树，并根据汇集树来转发数据包。

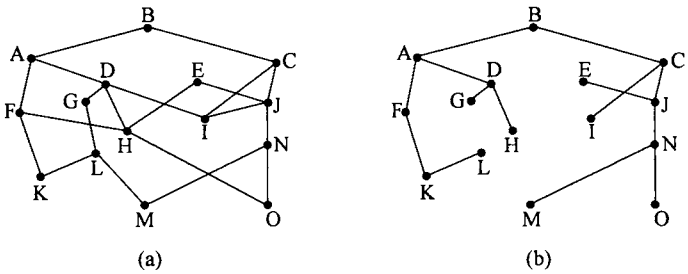


图 5-6
（a）一个网络；（b）路由器 B 的汇集树

请注意，汇集树不一定是唯一的；有可能存在具有相同路径长度的其他汇集树。如果我们允许选择所有可能的路径，则树就变成了更一般的结构，称为**有向无环图**（DAG，Directed Acyclic Graph）。DAG 没有环路。我们将把汇集树用作两种情况下的便利速记。两种情况都取决于路径不会互相干扰这样的技术假设，因此一条路径上的流量拥堵不会造成另一条路径的流量波动。

由于汇集树确实是一棵树，它不包含任何环，所以每个数据包将在有限的跳数内完成传递。然而，实际情形并非如此简单。在运行过程中，链路和路由器可能会故障，然后又恢复运行。所以，不同的路由器对当前拓扑结构的了解可能有所不同。而且，我们必须悄悄做出决定：每台路由器是独立地获取用于计算汇集树的信息，还是通过其他的方法来收集这些信息。稍后我们将很快回到这个问题。不管怎么样，最优化原则和汇集树为测量其他路由算法提供了一个基准。

5.2.2 最短路径算法

现在我们从一个简单技术开始学习路由算法。给定一个完整的网络视图，这项技术可用来计算最优路径。这些路径是那些我们希望分布式路由算法发现的，即使不是全部路由器都知道网络的所有细节。

基本想法是构造一张网络图，图中的每个节点代表一个路由器，每条边代表一条通信线路或者链路。为了选择一对给定路由器之间的路由，算法只需要在图中找出它们之间的最短路径。

最短路径（shortest path）概念值得做一些解释。一种测量路径长度的方法是跳数。若采用这种度量方法，则图 5-7 中 ABC 和 ABE 两条路径是等长的。另一种度量是以千米为单位的地理距离，在这种情况下，ABC 明显比 ABE 长很多（假定该图是按照比例画的）。

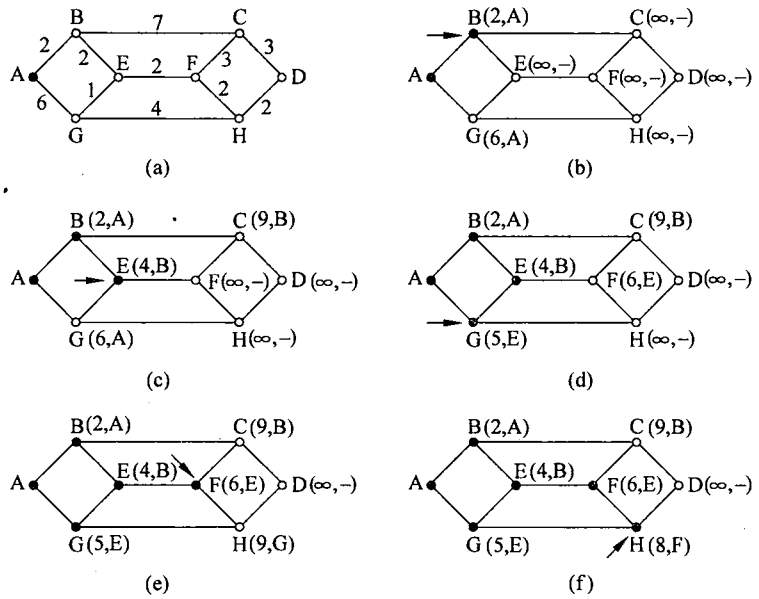


图 5-7 计算从 A 到 D 的最短路径的前 6 个步骤
箭头指向工作节点

然而，除跳数和物理距离之外，还可以用许多其他度量来标示路径的长短。例如，图中每条边用一个标准测试包的平均延迟来标记，这是每小时的测量结果。如果采用这种标记方法，则最短路径就是最快路径，而不是边数最少或者距离最短的路径。

一般情况下，边上面的标记可以作为距离、带宽、平均流量、通信成本、平均延迟等其他因素的一个函数，通过计算得出。通过改变函数的权重，路由算法就可以根据任何一种标准或者多种标准的组合来计算“最短”路径。

给定一个图，计算两个节点之间最短路径的算法已经有几个。这要归功于（Dijkstra，1959）设计的一种算法，这种算法能找出网络中一个源节点到全部目标节点的最短路径。每一个节点都标出了（在圆括号内）从源节点沿着已知的最佳路径到达本节点的距离。距离必须是非负的，如果基于带宽和延迟这样的实际测量值，那么距离本来就不可能为负。初始时，所有的路径都不知道，因此所有节点都被标记为无限远。随着算法的不断进行，陆续有一些路径被找到，于是节点的标记可能发生变化，以便反映出更好的路径。每个标记可能是暂时的，也可能是永久的。初始时，所有的标记都是暂时的。当发现一个标记代表了从源节点到该节点的最短可能路径，该标记就变成永久，以后不再改变。

为了说明标记算法的工作过程，请看图 5-7（a）中的加权无向图，这里的权值代表了一种度量，比如说距离。我们现在要找到从 A 到 D 的最短路径。开始时，将节点 A 标记

为永久，在图中用一个实心圆表示；然后我们依次检查每一个与 A（工作节点）相邻的节点，并且用它们与 A 之间的距离重新进行标记。为了可以重构出最终路径，每当一个节点被重新标记时，也要标记出这次探测动作的出发节点（即前一个节点）。

在检查了每一个与 A 相邻的节点之后，我们检查整个图中全部具有暂时性标记的节点，并且使得其中具有最小标记的那个节点成为永久性的，如图 5-7（b）所示。这个节点就变成新的工作节点。

现在我们从 B 开始，检查所有与 B 相邻的节点。对于每一个与 B 相邻的节点，如果节点 B 上的标记加上从 B 到该节点的距离小于该节点原来的标记，说明我们找到了一条更短的路径，所以需要重新标记该节点。

在检查了所有与工作节点相邻的节点，并且可能时修改了暂时性标记之后，算法需要对整个图进行搜索，找到具有最小标记值的暂时性节点。这个节点的标记将变成永久性，并且成为下一轮的工作节点。图 5-7 显示了算法的前 6 个步骤。

为了看清楚该算法的工作原理，请看图 5-7（c）。此时，我们刚刚把 E 变成永久性节点。假设存在一条比 ABE 还要短的路径，比如说 AXYZE（对某个 X 和 Y），则有两种可能性：节点 Z 已经是永久性的，或者它还没有变成永久性节点。如果 Z 是永久性的，则 E 已经被探测过了（当 Z 变成永久性节点之后的下一轮探测），所以路径 AXYZE 不会逃脱我们的搜索范围，因而它不可能是一条最短路径。

现在考虑 Z 仍然是暂时性标记的情形。如果节点 Z 上的标记大于或者等于 E 上的标记，则 AXYZE 不可能是一条比 ABE 更短的路径；如果 Z 上的标记小于 E 上的标记，则应该首先是 Z 而不是 E 变成永久性节点，允许从 Z 探测 E。

图 5-8 给出了这个算法的描述。全局变量 n 和 dist 描述了图，在 shortest_path 被调用之前对这两个变量初始化。这段程序与前面描述的算法之间的唯一区别是，在图 5-8 所示的程序中，我们从终结节点 t 开始计算最短路径，而不是从源节点 s 开始。

```
#define MAX_NODES 1024              /* maximum number of nodes */
#define INFINITY 1000000000         /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /*dist[i][j] is the distance from i to j */
void shortest_path(int s, int t, int path[])
{ struct state{                                 /* the path being worked on */
      int predecessor;                          /* previous node */
      int length;                               /* length from source to this node */
      enum {permanent, tentative} label;        /* label state */
  } state[MAX_NODES];
  int i, k, min;
  struct state *p;
  for (p = &state[0]; p < &state[n]; p++){      /*initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
  }
  state[t].length = 0; state[t].label = permanent;
```

图 5-8 用图来计算最短路径的 Dijkstra 算法

```
    k = t;                                  /* k is the initial working node */
    do{                                     /* Is there a better path from k? */
        for (i = 0; i < n; i++)             /* this graph has n nodes */
            if (dist[k][i] != 0 && state[i].label == tentative) {
                if (state[k].length + dist[k][i] < state[i].length) {
                    state[i].predecessor = k;
                    state[i].length = state[k].length + dist[k][i];
                }
            }
        /* Find the tentatively labeled node with the smallest label. */
        k = 0; min = INFINITY;
        for (i = 0; i < n; i++)
            if (state[i].label == tentative && state[i].length < min) {
                min = state[i].length;
                k = i;
            }
        state[k].label = permanent;
    } while (k != s);
    /* Copy the path into the output array. */
    i = 0; k = s;
    do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
}
```

图 5-8 （续）

由于在一个无向图中，从 t 到 s 的最短路径与从 s 到 t 的最短路径是相同的，所以，无论从哪一个节点开始计算其实并没有多大关系。之所以选择后向搜索的原因是，每个节点都被标记了它的前任节点而不是它的继任节点。因此，当最终的路径被复制到输出变量 path 中的时候，该路径也被逆转了过来。两次逆转的效果刚好相互抵消，因此，结果按正确的顺序产生。

5.2.3 泛洪算法

在实现路由算法时，每个路由器必须根据本地知识而不是网络的全貌做决策。一个简单的本地技术是**泛洪**（flooding），这种技术将每一个入境数据包发送到除了该数据包到达的那条线路以外的每条出境线路。

很显然，泛洪法会产生大量的重复数据包。事实上，除非采取某些措施来抑制泛洪过程，否则将会产生无限多的数据包。其中一项措施是在每个数据包的头中设置一个跳计数器，每经过一跳该计数器减一，当计数器到达 0 时就丢弃该数据包。理想情况下，跳计数器的初始值应该等于从源端到接收方之间路径的长度。如果发送方不知道该路径有多长，它可以将计数器的初始值设置为最坏情形下的长度，即网络的直径。

带有跳计数器的泛洪能够产生随着跳数增大而指数增长的重复数据包，而且路由器要复制以前已经看到过的数据包。抑制数据包泛滥的一种更好技术是让路由器跟踪已经泛洪过的数据包，从而避免第二次发送它们。实现这个目标的一种方式是让每个源路由器在接收来自主机的数据包时填上一个序号，然后每个路由器为每个源路由器准备一张表，记录

已经观察到的来自源路由器的序号。如果入境数据包在这张表中，它就不能再被泛洪到其他路由器。

为了防止该表无限地膨胀，每个表应该使用一个计数器 k 作为参数，它表示直到 k 的所有序号都已经观察到了。当一个数据包抵达时，很容易检查该数据包是否已经被泛洪过（只需要比较该数据包的序号和 k）；如果是被泛洪过，则丢弃该数据包。而且，不需要记录 k 以下的整个列表，因为 k 本身就是这部分数据的有效概括。

对于发送大多数数据包来说，泛洪算法是不切实际的，但它确实有一些重要的用途。首先，它确保数据包能被传送到网络中的每个节点。如果数据包的目的地只有一个，这种发送途径是一种浪费；但对于广播信息来说，这是一种有效的广播手段。在无线网络中，站传送的全部信息都可以被位于其无线范围内的所有其他站接收，这实际上就是一种形式的泛洪，一些无线路由算法利用了这个特性。

其次，泛洪途径的鲁棒性非常好。即使大量路由器被炸成碎片（例如，位于战争地区的一个军事网络），泛洪也能找到一条路径（如果存在），使得数据包到达目的地。泛洪需要的安装很少，路由器仅仅需要知道自己的邻居即可。这意味着，泛洪可以作为其他路由算法的基本构件，那些算法更有效但需要更多的处理。泛洪还可用作其他路由算法进行比较的性能度量。因为泛洪能并发选择每一条可能的路径，因此总能选出最短的那条路径，没有其他算法能够产生一个更短的延迟（如果我们忽略泛洪过程本身产生的开销）。

5.2.4 距离矢量算法

计算机网络通常使用动态路由算法。虽然动态路由算法比上述泛洪算法更复杂，但由于这些算法能找到当前网络拓扑中的最短路径，因而更加有效。特别地，两个动态算法最为流行，即距离矢量路由算法和链路状态路由算法。在这一小节中，我们考查前一个距离矢量路由算法；在下一小节，我们再讨论链路状态路由算法。

距离矢量路由（distance vector routing）算法是这样工作的：每个路由器维护一张表（即一个矢量），表中列出了当前已知的到每个目标的最佳距离，以及所使用的链路。这些表通过邻居之间相互交换信息而不断被更新，最终每个路由器都了解到达每个目的地的最佳链路。

距离矢量路由算法有时候也被叫做其他名称，最为常见是分布式 Bellman-Ford 路由算法，这种叫法是根据其设计者的名字来命名的（Bellman，1957；Ford 和 Fulkerson，1962）。距离矢量路由算法是最初 ARPANET 使用的路由算法，也曾被用于 Internet，相应的协议名称为 RIP 协议。

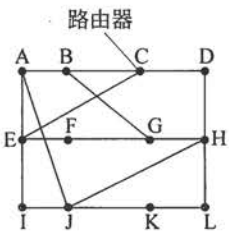
在距离矢量路由算法中，每个路由器维护一张路由表，它以网络每个路由器为索引，并且每个路由器对应一个表项。该表项包含两部分：到达该目标路由器的首选出境线路，以及到达该目标路由器的距离估计值。距离的度量可能是跳数，或者其他因素，正如我们在计算最短路径时讨论的那样。

假定路由器知道它到每一个邻居的“距离”。如果所用的度量是跳数，那么该距离就是 1 跳。如果度量值为传播延迟，则路由器很容易测量出链路的传播延迟。它只要直接发送一个特殊的 ECHO 数据包给邻居，邻居收到后盖上时间戳，尽可能快地发回来即可。

举个例子，假设使用延迟作为距离度量，并且路由器知道它到每个邻居的延迟。每隔

T 毫秒，每个路由器向它的每个邻居发送一个列表，该表包含了它到每个目标的延迟估计值；同时，它也从每个邻居那里接收到一个类似的表。想象一个路由器接收了来自邻居 X 的一个表，其中 X_i 表示邻居 X 估计的到达路由器 i 所需要的时间。如果该路由器知道它到邻居 X 的延迟为 m 毫秒，那么它也能明白在 X_i+m 毫秒之内经过 X 可以到达路由器 i。对每个邻居都执行这样的计算，最终可以发现到每个目标的最佳估计值，然后在新的路由表中使用这个最佳估计值以及对应的出境线路。请注意，在上述到目标距离的计算过程中没有使用老的路由表。

这个更新过程如图 5-9 所示。其中图 5-9（a）显示了一个网络。图 5-9（b）的前 4 列显示了 J 从邻居路由器接收到的延迟矢量。A 声称它到 B 有 12 毫秒的延迟，到 C 有 25 毫秒的延迟，到 D 有 40 毫秒的延迟，等等。假定 J 已经测量和估计了它到邻居 A、I、H 和 K 的延迟分别为 8、10、12 和 6 毫秒。



(a)

到	A	I	H	K	J新估算的延迟 线路
A	0	24	20	21	8 A
B	12	36	31	28	20 A
C	25	18	19	36	28 I
D	40	27	8	24	20 H
E	14	7	30	22	17 I
F	23	20	19	40	30 I
G	18	31	6	31	18 H
H	17	20	0	19	12 H
I	21	0	14	22	10 I
J	9	11	7	10	0 —
K	24	22	22	0	6 K
L	29	33	9	9	15 K
	JA	JI	JH	JH	
	延迟是8	延迟是10	延迟是12	延迟是6	J的新路由表

从J的4个邻居收到的矢量

(b)

图 5-9

（a）一个网络示例；（b）来自 A、I、H、K 的输入，以及 J 的新路由表

现在考虑 J 如何计算它到路由器 G 的新路径。它知道在 8 毫秒之内可以到达 A，并且 A 声称可以在 18 毫秒内到达 G，所以 J 知道，如果它将那些目标地址为 G 的数据包转发给 A 的话，那么到 G 的延迟为 26 毫秒。类似地，它计算出经过 I、H 和 K 到达 G 的延迟分别为 41（即 31+10）、18（即 6+12）和 37（即 31+6）毫秒。在这些计算得出的距离值之中，最好的结果是 18，所以在 J 的路由表中，对应于 G 的表项中的延迟值为 18 毫秒，所用的路径是经过 H 的那条。对于所有其他的目标地址执行同样的计算过程，最后得到的新路由表如图中最后一列所示。

无穷计算问题

整个网络最佳路径的寻找过程称为**收敛**（convergence）。距离矢量路由算法作为一项简单技术很有用，因为路由器可以在所有路径中有选择地计算出一条最短路径。但实际上它有一个严重的缺陷：虽然它总是能够收敛到正确的答案，但速度可能非常慢。尤其是，它对于好消息的反应非常迅速，而对于坏消息的反应异常迟缓。考虑这样一个路由器，它到目标 X 的最佳路径非常大。如果在下一次交换信息时，邻居 A 突然报告说它到 X 有一条

延迟很短的路径，那么，该路由器只是将发送给 X 的流量切换到通向 A 的线路。经过一次矢量交换，好消息就发挥作用了。

为了看清楚好消息的传播有多快，请考虑图 5-10 中一个五节点（线型）网络，这里的延迟度量为跳数。假定 A 最初处于停机状态，所有其他的路由器都知道这一点。换句话说，它们都将到 A 的延迟记录为无穷大。

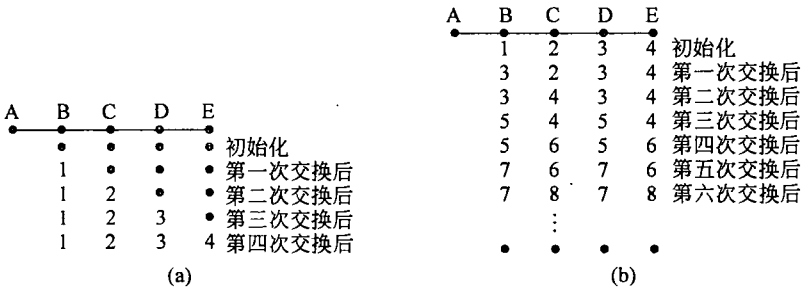


图 5-10 无穷计算问题

当 A 启动时，其他的路由器通过矢量交换知道了这一点。为了简化起见，我们假定在某个地方有个巨大的锣鼓，它定期敲击来启动所有路由器同时进行矢量交换。在第一次交换时，B 知道它左边的邻居到 A 的延迟为 0。于是 B 在它的路由表中建立一个表项，说明 A 离它一跳远。所有其他的路由器仍然认为 A 是停机的。这时候，针对 A 的路由表项如图 5-10（a）中的第二行所示。在接下去的交换中，C 知道 B 有一条路径通向 A，并且路径长度为 1，所以它更新自己的路由表，指明它到 A 的路径长度为 2，但是 D 和 E 要到以后才能听到这个好消息。很显然，好消息扩散的速度是每交换一次往远处走一跳。如果一个网络中最长路径是 N 跳，那么经过 N 次交换之后，每个路由器都将知道新恢复的链路和路由器。

现在我们考虑图 5-10（b）的情形，在这里所有的线路和路由器最初都是正常工作的。路由器 B、C、D 和 E 到 A 的距离分别为 1、2、3 和 4。突然间 A 停机了，或者 A 和 B 之间的链路断了（从 B 的角度来看这两种情况的效果是相同的）。

在第一次信息交换时，B 没有听到来自 A 的任何信息。幸运的是，C 说“别担心，我有一条通向 A 的长度为 2 的路径”。B 并不怀疑 C 的路径经过自身，B 能知道的全部只是 C 可能有 10 条链路，每条都独立地通向 A，并且长度为 2。因此，B 认为它可以通过 C 到达 A，路径长度为 3。在第一次交换之后，D 和 E 并不更新它们的 A 表项。

在第二次信息交换时，C 注意到，它的每一个邻居都声称有一条通向 A 的长度为 3 的路径。它随机地挑选出一条，并且将它到 A 的距离更新为 4，如图 5-10（b）中的第三行所示。通过后续的交换，可以得到图 5-10（b）中余下的记录历史。

从这个图中，我们应该很清楚地看出为什么坏消息传播得很慢：没有一个路由器具有一个比它所有邻居的最小值还大于 1 的值。逐渐地，所有的路由器都会趋向无穷大，但是所需交换的次数依赖于代表无穷大的数值。由于这样的原因，明智的做法是将无穷大设置为最长的路径加 1。

并不完全出人意料，这个问题称为**无穷计数**（count-to-infinity）问题。已经有许多试图解决该问题的工作，例如，防止路由器向邻居返回一个从该邻居获得的最佳路径，这个方法称为带有染毒逆向的水平分裂法，在 RFC 1058 中对此进行了讨论。然而，这些启发

式工作尽管有丰富多彩的名称，但实际工作效果并不好。问题的核心在于当 X 告诉 Y 它有一条通往某个地方的路径，Y 无从知道自己是否已在这条路径上。

5.2.5 链路状态路由

1979 年以前 ARPANET 一直使用距离矢量路由算法，而在此之后则改为使用链路状态路由算法。导致距离适量算法退位的主要问题在于，当网络拓扑结构发生变化后距离矢量路由算法需要太长时间才能收敛到稳定状态（由于无穷计数问题）。因此，距离矢量路由算法被一个全新的算法所替代，该算法称为**链路状态路由算法**（link state routing）。今天，链路状态路由算法的变种算法——IS-IS 或者 OSPF 已经成为大型网络或 Internet 应用最为广泛的路由算法。

链路状态路由算法的设计思想非常简单，可以用五个部分加以描述。每一个路由器必须完成以下的事情，算法才能正常工作：

（1）发现它的邻居节点，并了解其网络地址。

（2）设置到每个邻居节点的距离或者成本度量值。

（3）构造一个包含所有刚刚获知的链路信息包。

（4）将这个包发送给所有其他的路由器，并接收来自所有其他路由器的信息包。

（5）计算出到每个其他路由器的最短路径。

实际上，算法将完整的拓扑结构分发给了每一个路由器。然后每个路由器运行 Dijkstra 算法就可以找出从本地到每一个其他路由器的最短路径。下面我们详细地考虑上述每一个步骤。

发现邻居

当一个路由器启动时，它的第一个任务是找出哪些路由器是它的邻居。为了实现这个目标，它只需在每一条点到点线路上发送一个特殊的 HELLO 数据包。线路另一端的路由器应该返回一个应答说明自己是谁。这些名字必须是全局唯一的，因为当一个远程路由器以后听到有三个路由器都能连接到 F 时，它必须能够确定这三个路由器所提到的 F 是同一个路由器 F。

当两个或者多个路由器通过一个广播链路连接（比如一个交换机、环或经典以太网），情形会稍微复杂一些。图 5-11（a）显示了一个广播 LAN 直接与三个路由器 A、C 和 F 连接的情形。如图所示，每个路由器都连接到一个或者多个其他的路由器上。

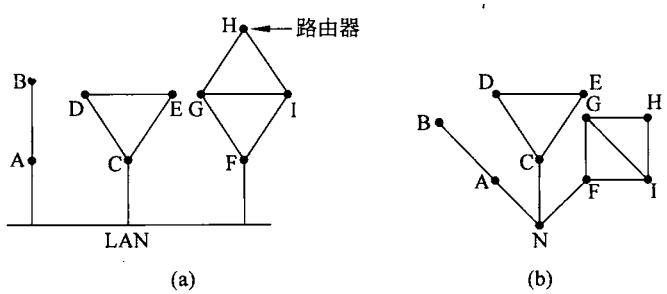


图 5-11

（a）9 个路由器和一个广播 LAN；（b）左侧网络的模型图

广播 LAN 为连接到其上的任何一对路由器提供了彼此的连通性。然而，把 LAN 建模成许多个点到点链路会增大拓扑结构，从而导致浪费消息。模拟局域网的一个更好方法来是把它看作一个节点，如图 5-11（b）所示。在这里，我们引入了一个新的人造节点 N，它与 A，C 和 F 连接。LAN 上的一个**指定路由器**（designated router）被选中替代 N 运行路由协议。事实上，从 LAN 上的 A 到 C 是可能的，这里用路径 ANC 表示这条路径。

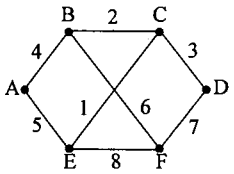
设置链路成本

为了寻找最短路径，链路状态路由算法需要每条链路以距离或成本度量。到邻居的成本可自动设置或由网络运营商配置的度量。一种常用的选择是使成本与链路带宽成反比。例如，1Gbps 以太网的成本可能是 1，而 100 Mbps 以太网的成本可能是 10。这样可以使得高容量的路径成为路由更好的选择。

如果网络在地理上分散，链路的延迟可以作为成本的组成部分，这样才能更好地选择较短链路上的路径。确定这种延迟的最直接方法是通过线路给另一边发送一个特殊的 ECHO 数据包，要求对方立即发回。通过测量往返时间再除以 2，发送路由器可以得到一个合理的延迟估算值。

构造链路状态包

一旦收集到了所需要的交换信息，每个路由器的下一步工作是构建一个包含所有这些信息的数据包。该数据包的内容首先是发送方的标识符，接着是一个序号（Seq）和年龄（Age，后面再介绍），以及一个邻居列表。对于每个邻居，同时要给出到这个邻居的延迟。图 5-12（a）显示了一个网络实例，每条线路上标出了延迟信息。这 6 个路由器所对应的链路状态数据包如图 5-12（b）所示。



(a)

链路状态包

A	
序号	
年龄	
B	4
E	5

B	
序号	
年龄	
A	4
C	2
F	6

C	
序号	
年龄	
B	2
D	3
E	1

D	
序号	
年龄	
C	3
F	7

E	
序号	
年龄	
A	5
C	1
F	8

F	
序号	
年龄	
B	6
D	7
E	8

(b)

图 5-12

（a）一个网络示例；（b）该网络的链路状态包

构造链路状态数据包很容易。难的是确定什么时候构造数据包。一种可能的做法是周期性地创建数据包，也就是说，以一定的时间间隔创建链路状态数据包。另一种可能做法是每当发生某些重要的事情时才创建数据包，比如当一条线路断掉或者一个邻居节点停机时，或当它们重新恢复运行时，或当它们的特性发生了一定变化时。

分发链路状态包

链路状态路由算法最技巧的部分在于分发链路状态数据包。所有路由器必须快速并可靠地获得全部的链路状态数据包。如果不同的路由器使用了不同版本的拓扑结构，那么它们计算出来的路由可能会不一致，例如出现环路、目标机器不可达以及其他的问题。

首先，我们描述最基本的发布算法，然后再对它进行改进。基本思路是使用泛洪法将链路状态数据包分发给所有路由器。为了控制泛洪规模，每个数据包都包含一个序号，序

号随着每一个新数据包发出而逐一递增。路由器记录下它所看到的所有（源路由器、序号）对。当一个新的链路状态数据包到达时，路由器检查这个新来的数据包是否已经出现在上述观察到的列表中。如果这是一个新数据包，则把它转发到除入境线路之外的所有其他线路上。如果这是一个重复数据包，则将它丢弃。如果数据包的序号小于当前所看到过的来自该源路由器的最大序列号，则它将被当作过时数据包而拒绝接受，因为路由器已经有了更新的数据。

这个算法还有几个问题有待处理，不过，这些问题都是可管理的。第一，如果序号绕回，可能会产生混淆。这里的解决方案是使用一个32位的序号。即使每秒钟产生一个链路状态数据包，也需要137年才可能发生绕回，所以，这种可能性可以忽略不计。

其次，如果一个路由器崩溃了，那么它将丢失所有的序号记录表。如果它再从0开始，那么，下一个数据包将被作为重复数据包而拒绝。

再次，如果一个序号被破坏了，比如发送方发送的序号为4，但是由于产生了1位错误，所以接收方看到的序号是65 540，那么，序号从5到65 540的数据包都将被当作过时数据包而拒绝接受，因为接收方认为当前的序号是65 540。

所有这些问题的解决方案都一样：在每个数据包的序号之后包含一个年龄（age）字段，并且每秒钟将年龄减1。当年龄字段的值被减到0时，来自路由器的该信息将被丢弃。通常情况下，每隔一段时间，比如说10秒，一个新的数据包就会到来；所以，只有当一个路由器停机时（或者6个连续的数据包被丢失，这种情形发生的可能性不大），路由器信息才会超时。在初始泛洪过程中，每个路由器也要递减age字段，这样可以确保没有数据包丢失，也不会无限制生存下去（如果一个数据包的age为0，则被丢弃）。

对这个算法做一些改进可以使它更加健壮。当一个链路状态数据包被泛洪到一个路由器时，它并没有立即被排入队列等待传输。相反，它首先被放到一个保留区中等待一段时间。如果在这个数据包被转发出去之前，另一个来自于同一个源路由器的链路状态数据包也到来了，那么就比较它们的序号。如果两个数据包的序号相等，则丢弃重复数据包。如果两者不相等，则丢弃老的数据包。为了防止线路产生错误导致丢包和错包，所有的链路状态数据包都要被确认。

在图5-12（a）所示的网络中，路由器B使用的数据结构如图5-13所示。这里的每一行对应于一个刚刚到达的，但是还没有完全处理完毕的链路状态数据包。该表记录了数据包的来源、序号和年龄，以及状态数据。而且，针对B的三条线路（分别到A、C和F）还记录了发送和确认标志。发送标志表明该数据包必须在所指示的线路上发送。确认标志表明它必须在这条线路上得到确认。

			发送标志			ACK标志			
源路由器	序号	年龄	A	C	F	A	C	F	数据
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

图5-13 在图5-12（a）中路由器B的状态包缓冲区

在图 5-13 中，来自 A 的链路状态数据包可以直接到达，所以 B 必须将它发送给 C 和 F，并且按照标志位的指示向 A 发回确认。类似地，必须把来自 F 的数据包转发给 A 和 C，并且向 F 返回确认。

然而，第三个数据包，即来自 E 的数据包有所不同。它到达两次，一次经过 EAB，另一次经过 EFB。因此，它只需被发送给 C，但是要向 A 和 F 确认，正如标志位所示。

如果一个重复数据包到来时原来的数据包仍然在缓冲区中，那么标志位必须作相应的改变。例如，如果表中第四项被转发出去之前，C 的链路状态数据包的一份副本从 F 到达，那么，这六位将被改变为 100011，以表明该数据包必须向 F 确认，但是不用转发了。

计算新路由

一旦路由器已经积累了全部的链路状态数据包之后，它就可以构造出完整的网络图，因为每条链路都已经被表示出来了。事实上，每条链路被表示了两次，每个方向各表示一次。不同方向的链路可能有不同的成本。最短路径计算可找到从 A 到 B 与从 B 到 A 不同的路径。

现在可以在路由器本地运行 Dijkstra 算法，以便构建出从本地出发到所有可能目标的最短路径。这个算法的运行结果告诉路由器到达每个目的地能够走哪条链路。这个信息被安装在路由表中，而且恢复正常操作。

相比距离矢量算法，链路状态路由算法需要更多的内存和计算。对于一个具有 n 个路由器的网络，每个路由器有 k 个邻居，那么，用于存储输入数据所要求的内存与 kn 成正比，这至少与列出全部目的地的路由表一样大。而且，计算时间的增长快过 kn，即使采用最有效的数据结构，在大型网络中运行这个算法依然是个问题。不过，在许多实际场合，链路状态路由算法工作得很好，因为它没有慢收敛问题。

链路状态路由算法被广泛地应用于实际网络中，所以现在提一下某些使用该算法的例子。许多 ISP 使用中间系统到中间系统（IS-IS，Intermediate System-Intermediate System）链路状态协议（Oran，1999），它是专门为 DECnet 而设计的，后来被 ISO 采纳用于 OSI 协议；自此以后，它被作了多次修改以便能够处理其他的协议，例如最著名的 IP 协议。开放最短路径优先（OSPF，Open Shortest Path First）是另一个主流链路状态协议。它是在 IS-IS 之后几年由 IETF 设计的，而且它吸收了 IS-IS 的许多创意。这些创意包括：一种泛洪链路状态更新的自稳定方法、LAN 上的指定路由器概念以及计算和支持路径分裂和多个度量的方法。因此，IS-IS 和 OSPF 之间的差异非常小。其中一个最重要的差别是，IS-IS 可同时携带多个网络层协议的信息（比如，IP、IPX 和 AppleTalk），OSPF 不具备这个特性。对于大型的多协议环境这一特性是个优势。我们将在 5.6.6 节介绍 OSPF。

下面依次对路由算法做出一般性的评论。连接状态、距离向量和其他路由算法依赖于所有路由器计算路径的处理。硬件或软件，甚至少数路由器的问题都可以肆虐破坏网络。例如，如果一个路由器声称有一条实际上并不存在的链路，或者忘记一条实际上存在的链路而声称没有，由此产生的网络图都将是不正确的。如果一个路由器转发数据包失败，或者在转发期间自己发生故障，对应的路由将无法正常工作。最后，如果内存不足或者路由计算出错，就会发生不好的事情。随着网络规模几十、几百或者上千个节点的增长，某些路由器偶尔会失败的概率变得不可忽视。应对这个困境的诀窍是尽量做好安排，当发生不

可避免的错误时把危害降到最低。（Perlman，1988）详细讨论了这些问题并给出了可能的解决方案。

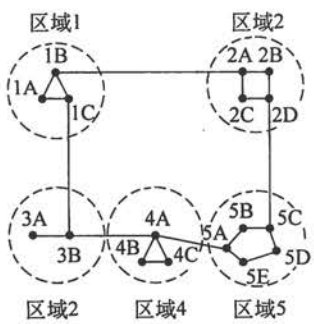
5.2.6 层次路由

随着网络规模的增长，路由器的路由表也成比例地增长。不断增长的路由表不仅消耗路由器内存，而且还需要更多的 CPU 时间来扫描路由表以及更多的带宽来发送有关的状态报告。当网络增长到一定时可能会达到某种程度，此时每个路由器不太可能再为其他每一个路由器维护一个表项。所以，路由不得不分层次进行，就好像电话网络中的做法那样。

在采用了分层路由之后，路由器被划分成区域（region）。每个路由器知道如何将数据包路由到自己所在区域内的目标地址，但是对于其他区域的内部结构毫不知情。当不同的网络被相互连接在一起，很自然地就会将每个网络当作一个独立的区域，一个网络中的路由器不必知道其他网络的拓扑结构。

对于大型网络，两级的层次结构可能还不够；可能有必要将区域组织成簇（cluster），将簇组织成区（zone），将区组织成群（group），等等，直到将所有的集合名词用完为止。举一个多层结构的例子，请考虑如何将一个数据包从加利福尼亚州的伯克利路由到肯尼亚的马琳迪。伯克利的路由器知道加利福尼亚州的详细拓扑结构，但是可能将所有州际的流量发送给洛杉矶的路由器；洛杉矶的路由器能够将流量直接路由给美国其他的路由器，但是它会将国际流量发到纽约；纽约的路由器直接将所有的流量发送至目标国家中负责处理国际流量的路由器，比如肯尼亚的内罗毕。最后，该数据包将沿着肯尼亚国家中的路径树往下传送，一直到达马琳迪（Malindi）。

图 5-14 给出了一个两级层次的定量分析例子，其中包含了 5 个区域。路由器 1A 的完整路由表有 17 个表项，如图 5-14（b）所示。如果采用分级路由，则路由表如图 5-14（c）所示，所有针对本区域内的路由器表项都跟原先一样；但是，所有到其他区域的路由都被压缩到了单个路由器中。因此，所有到区域 2 的流量都要经过 1B-2A 线路，其余的远程流



(a)

1A的完整表

目标地址	线路	跳数
1A	—	—
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

1A的层次表

目标地址	线路	跳数
1A	—	—
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

图 5-14 层次路由

量都经过 1C-3B 线路。层次路由使得路由器 1A 的路由表长度从 17 项降低为 7 项。随着区域数与每个区域中路由器数量之比值的增加，节省下来的表空间也随之增加。

不幸的是，这种空间的节省不是免费得来的，它需要付出代价，其代价的形式是增加了路径长度。例如，从 1A 到 5C 的最佳路径是经过区域 2，但采用了层次路由之后，所有到区域 5 的流量都要经过区域 3，因为对于区域 5 中的大多数目标来说这是更好的选择。

当单个网络变得非常大时，一个有趣的问题是“应该分多少层？”。例如，考虑一个具有 720 个路由器的子网。如果没有分层，每个路由器需要 720 个路由表项；如果子网被分成 24 个区域，每个区域 30 个路由器，那么每个路由器只需要 30 个本地表项，加上 23 个远程表项，总共 53 个表项；如果采用三级层次结构，总共 8 个簇，每个簇包含 9 个区域，每个区域 10 个路由器，那么，每个路由器需要 10 个表项用于记录本地路由器，8 个表项用于到同一簇内其他区域的路由，7 个表项用于远程的簇，总共 25 个表项。（Kamoun 和 Kleinrock，1979）发现，对于一个包含 N 个路由器的网络，最优的层数是 ln N，每个路由器所需的路由器表项是 eln N 个。他们还证明了，由于分层路由而导致的平均路径长度的实际增长非常小，通常是可以接受的。

5.2.7 广播路由

在有些应用中，主机需要给其他多个或者全部主机发送消息。例如，用于发布天气预报、股市行情最新报告或者现场直播节目等服务，它们的最佳工作方式是将消息广播给所有的机器，然后让那些感兴趣的机器读取数据。同时给全部目标地址发送一个数据包称为**广播**（broadcasting）；为了实现广播，人们已经提出了各种各样的方法。

一种不要求网络具有任何特殊性质的广播方法是让源机器简单地给每一个目标单独发送一个数据包。这种方法不仅浪费带宽，而且还要求源机器拥有所有目标机器的完整地址列表。实际上这种做法不够理想，即使它广泛适用。

一种改进方法称为**多目标路由**（multidestination routing），每个数据包包含一组目标地址，或者一个位图，由该位图指定所期望到达的目标。当一个数据包到达一个路由器时，路由器检查数据包携带的所有目标，确定哪些输出线路是必要的（只要一条输出线路是到达至少一个目标的最佳路径，那么它就是必要的）。路由器为每一条需要用到的输出线路生成一份该数据包新的副本，在这份副本中只包含那些使用这条线路的目标地址。实际上，原来的目标集合被分散到这些输出线路上。在经过了足够多的跳数之后，每个数据包将只包含一个目标地址，因此可以被当作普通的数据包来对待。多目标路由方法就如同单个地址的数据包一样，只不过当多个数据包必须遵循同样的路径时，其中一个数据包承担了全部的费用，而其他的数据包则是免费搭载。因此，网络带宽的利用率更高。然而，这种模式依然要求源端知道全部的目标地址，对于路由器来说，要确定从哪些线路转发多目标数据包的工作量太大，尤其是处理多个不同的数据包时。

我们早就看到了更好的广播路由技术——泛洪。当每个源实现了序号，泛洪能有效地利用链路，而且路由器要做的决策相对地简单。虽然泛洪方法不适合普通的点到点通信，但它被认为值得考虑用作广播。然而，事实证明，一旦计算出普通数据包的最短路径，我们可以把广播做得更好。

逆向路径转发（reverse path forwarding）思想一经提出就获得了关注，它被认为是一种既优雅又相当简单的广播技术。当一个广播数据包到达一个路由器时，路由器检查它到来的那条线路是否正是通常用来给广播源端发送数据包用的那条线路。如果是，说明这是一个极好的机会，该广播数据包是沿着最佳路径被转发过来的，因而是到达当前路由器的第一份副本。如果是这种情况，则路由器将该数据包转发到除了到来的那条线路之外的所有其他线路上。然而，如果广播数据包是从其他任何一条并非首选的到达广播源的线路入境的话，该数据包被当作一个可能的重复数据包而丢弃。

逆向路径转发算法的一个例子如图 5-15 所示。图 5-15（a）部分显示了一个网络，图 5-15（b）部分显示了该网络中路由器 I 的一棵汇集树，图 5-15（c）部分显示了逆向路径算法是如何工作的。在第一跳，I 发送数据包给 F、H、J 和 N，如树中第二行所示。这些数据包中的每一个都是在通向 I 的首选路径（假定首选的路径都沿着汇集树）到来的，这点用字母外面加一个圆圈来表示。在第二跳，共产生了 8 个数据包，其中，在第一跳接收到数据包的路由器各产生 2 个数据包。结果，所有这 8 个数据包都到达了以前没有访问过的路由器，其中 5 个是沿着首选线路到来的。在第三跳所产生的 6 个数据包中，只有 3 个是沿首选路径（在 C、E 和 K）到来的，其他的都是重复数据包。在经过五跳和 24 个数据包以后，广播过程终止。相比之下，如果完全沿着汇集树的话，只需要 4 跳和 14 个数据包。

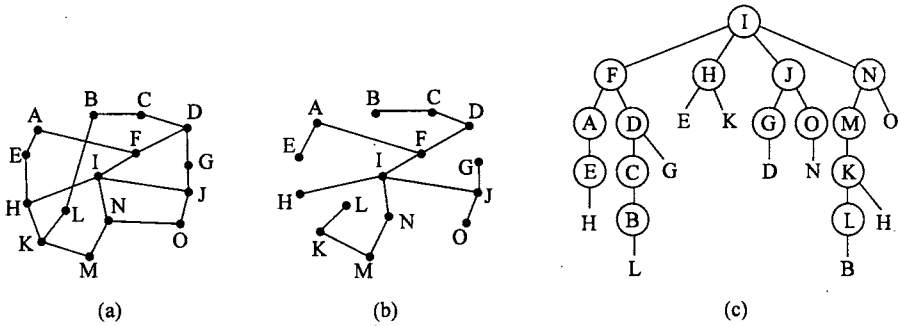


图 5-15　逆向路径转发

（a）一个网络示例；（b）一棵汇集树；（c）由逆向路径转发构成的树

逆向路径转发的主要优点是它有效而且易于实现。它只往每个方向上的链路发送一次广播数据包，就像泛洪一样简单，而仅仅要求路由器知道如何到达全部目标；路由器无须记住序号（或使用其他机制来停止泛洪）或者在数据包中列出全部的目标地址。

最后一种算法改进了逆向路径转发的行为。它明确使用了以发起广播的路由器为根的汇集树，或者任何其他便利的生成树。**生成树**（spanning tree）是网络的一个子集，它包含所有的路由器，但是没有任何环路。汇集树是生成树的一种。如果每个路由器都知道它的哪些线路属于生成树，那么，它就可以将一个入境广播数据包复制到除了该数据包到来的那条线路之外的所有生成树线路上。这种方法可最佳使用带宽，并且所生成的数据包也绝对是完成这项任务所需要的最少数量。例如，图 5-15（b）的汇集树就被用作生成树，广播数据包的副本最少，只有 14 个。唯一的问题是，每个路由器都必须知道这棵生成树才可以正常工作。有时候这样的信息是可以得到的（比如采用了链路状态路由算法，所有路由器都知道完整的网络拓扑，因而它们可以计算出一棵生成树），但是有时候无法获得这样的信

息（比如采用了距离矢量路由算法）。

5.2.8 组播路由

有些应用，比如多人游戏或者体育赛事视频直播到几个观看点，这样的应用将数据包发送给多个接收者。除非组的规模很小，否则给每个接收者单独发不同的数据包的代价很昂贵。另一方面，如果在一个由百万节点组成的网络中有一个由 1000 个机器组成的组，采用广播技术发送数据包显然也是一种浪费，因为大多数接收者对广播的消息并不感兴趣（甚至最糟糕的是他们虽然感兴趣，但不应该看到这些消息）。因此，我们需要有一种办法能够给明确定义的组发送消息，这些组的成员数量虽然很多，但相比整个网络规模却很小。

给这样的一个组发送消息称为**组播**（multicasting），使用的路由算法称为组播路由（multicast routing）。所有的组播方案都需要一些方法来创建和撤销特定的组，并确定哪些路由器是组的成员。如何完成这些任务不是路由算法要关注的。现在，我们假定每个组由一个组播地址标识，并且路由器知道自己属于哪些组。我们在 5.6 节讨论 Internet 网络层时再重温组成员的有关内容。

组播路由方案建立在我们已经学习过的广播路由方案的基础上，为了将数据包传递给组的成员同时又有效利用带宽，数据包可沿着生成树发送。然而，最佳生成树的使用取决于组的密度分布。密集分布指接收者遍布在网络的大部分区域；稀疏分布指大部分网络都不属于组。在本节，我们考虑这两种情况。

如果组的分布是密集的，那么广播是一个良好的开端，因为它能有效地把数据包发到网络的每个角落。但广播可能将到达一些不属于该组成员的路由器，因而也是一种浪费。（Deering 和 Cheriton，1990）探索出一个解决方案，就是通过修剪广播生成树把不通往组成员的链路从树中删掉。修剪结果得到的是一棵有效的组播生成树。

作为一个例子，考虑如图 5-16（a）网络，其中有两个组：组 1 和组 2。有些路由器连接的主机属于其中的一个组或同时属于两个组，如图所示。最左边的路由器的一棵生成树如图 5-16（b）所示。此树可用于广播，对于组播来说则过度了，这从下面显示的两个修剪版本可见一斑。在图 5-16（c）中，所有不通往组 1 成员的主机的链路已被删除，结果是一棵针对最左边路由器发送到组 1 的生成树。数据包的转发只能沿着这棵树进行，可见这比广播树有效，因为这里只有 7 条链路而不是 10 条链路。图 5-16（d）显示了一棵针对组 2 修剪后的组播生成树。相比广播树它也更加有效，此时只有这 5 条链路。这个例子表明，不同的组播组有不同的生成树。

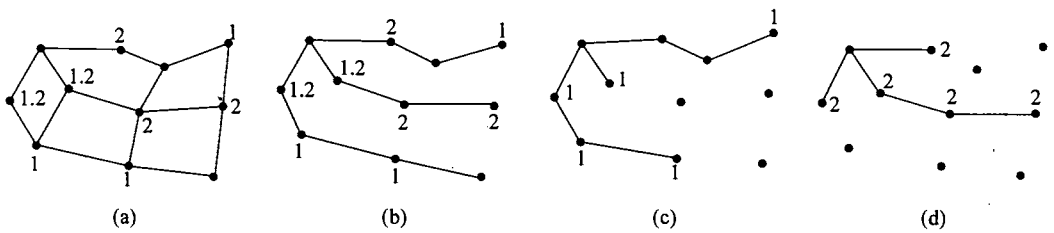


图 5-16

（a）一个网络示例；（b）最左侧路由器的一棵汇集树；（c）组 1 的一棵组播树；（d）组 2 的一棵组播树

生成树的修剪方式有许多种。如果路由器使用了链接状态路由算法，并且每个路由器知道完整的网络拓扑结构，包括了解哪些主机属于哪个组，那么可以使用一种最简单组播算法。每个路由器针对组内的每个发送者构造一棵它自己的修剪后生成树，具体做法是先按常规方法构造一棵以发送者为根的汇集树，然后从汇集节点上删除所有不连到组成员的链路。组播（MOSPF，Multicast OSPF）就是一个以这种方式工作的链路状态协议例子（Moy，1994）。

如果采用距离矢量路由算法，则要遵循不同的修剪策略。基本算法是逆向路径转发。然而，一旦一个路由器没有任何主机对某个组感兴趣，并且没有连接到需要接收该组播消息的其他路由器，那么它要用 PRUNE 消息作为接收组播消息的响应，告诉发送该消息的邻居不要再给自己发送任何来自该组发送者的消息。如果一个路由器自己所连的主机没有一个属于该组成员，并且从它以前转发组播消息的所有线路都接收了这样的修剪消息，那么它也同样以 PRUNE 消息来响应。通过这种递归方式，最终修剪出一棵生成树。**距离矢量组播路由协议**（DVMRP，Distance Vector Multicast Routing Protocol）就是一个以这种方式工作的组播路由协议例子（Waitzman 等，1988）。

修剪过程的最后结果得到的是一棵有效生成树，该树只用到那些抵达组成员真正需要的链路。这种方法的一个潜在缺点是路由器需要做大量的工作，特别是大型网络。假设一个网络有 n 个组，每个组平均有 m 个节点。在每个路由器上针对 n 组，每个组有 m 棵修剪生成树，因此总共需要存储 mn 棵生成树。例如，图 5-16（c）给出了最左边路由器给组 1 成员发送所用的生成树。最右边的路由器给组 1 发送数据包所用的生成树（未显示）将完全不同，因为组播数据包直接朝着组成员去而不会通过图的左侧发送。反过来，这意味着，路由器转发数据包将沿着不同的方向进行，具体方向取决于组内哪个节点是发送者。当存在大量的组，并且组内发送者很多时，需要大量的空间来存储所有的树。

另一种设计是采用**基于核心树**（core-based trees）的技术，计算某个组的单棵生成树（Ballardie 等，1993）。采用这种方法时所有路由器都同意某个路由器作为根，这个根称为**核心**（core）或**会聚点**（rendezvous point），然后每个成员通过给根发送一个数据包来建立这棵树。树是组播数据包遵循的路径集合。图 5-17（a）显示了一棵组 1 的核心树。为了把数据包发送到这个组，发送者把数据包发给核心；当数据包到达核心后，它再被沿着树往下转发。图 5-17（b）显示了网络右侧一个发送者的组播过程。作为性能优化的一种措施，发往该组的数据包并不需要先发送到核心然后再开始组播。一旦数据包到达树，它便沿着树向上转发给根，但同时沿着树转发到其他分枝。这种情况由 5-17（b）中的右上角发送者显示。

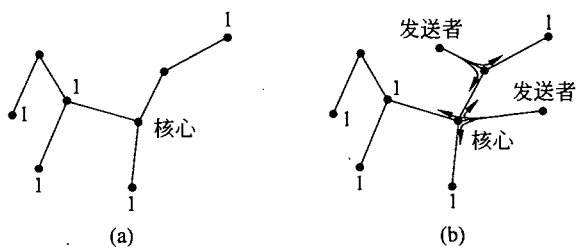


图 5-17
（a）组 1 的基于核心树；（b）给组 1 成员发送数据包

对于所有的组播源使用同一棵共享树是无法达到最优的。例如，在图 5-17（b）中，从网络右侧的发送者到达右上方的组成员通过核心要三跳，如果直接发送或许不需要三跳。共享树的低效率取决于核心和发送者的位置，把核心设置在所有发送者的中间往往是一种合理的做法。如果只有一个发送者，比如视频流传输到一个组，那么将发送者作为核心是最优的。

另外值得注意的是共享树可以大大节省存储开销、消息发送和计算。每个路由器只要为每个组保存一棵树，而不是 m 棵树。此外，不属于这棵共享树一部分的路由器根本不需要为组做任何工作。正是出于这个原因，像这类基于核心树的共享树方法被用于 Internet 的稀疏组播，成为流行协议的一部分，例如**协议独立组播**（PIM，Protocol Independent Multicast）（Fenner 等，2006）。

5.2.9 选播路由

到目前为止，我们已经学习了三种数据包传递模型：源给单个目标节点发送（称为单播）、给所有目标节点发送（称为广播）以及给一组目标节点发送（称为组播）。实际上，还有另一种称为**选播**（Anycast）的传递模型有时也非常有用。在选播方式下，数据包被传递给最近的一个组成员（Partridge 等，1993）。发现这些路径的方案就是所谓的**选播路由**（Anycast routing）。

我们为什么还要选播？有的时候，节点提供了诸如报时或者内容分发等服务，这类服务对客户而言最重要的是获得正确的信息而不是与哪个节点取得联系，任何节点都可以，只要它能提供所需的服务。例如，选播就作为域名系统的一部分广泛应用于 Internet 上，关于域名系统我们将在第 7 章讨论。

幸运的是，我们不需要为选播制定新的路由方案，因为普通的距离矢量和链路状态路由算法可用来生成选播路由。假设我们要选播数据包到组 1 的成员，该组成员都将被赋予一个组地址“1”而不是一个个独立的地址。距离矢量路由将像往常一样分发向量，并且节点将只选择到目的地 1 的最短路径。这将导致数据包被发送到目的地 1 的最近实例。图 5-18（a）显示了选播路由。此过程之所以能正常工作是因为路由协议并没有意识到目的地有多个实例。也就是说，它认为节点 1 的实例都是同一个节点，如图 5-18（b）所示的拓扑结构。

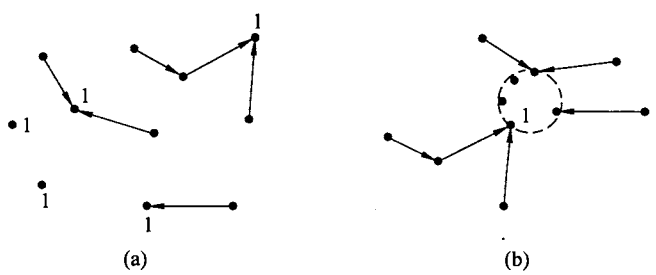


图 5-18
（a）组 1 的选播路由；（b）路由协议看到的拓扑结构

这个过程也同样适用于链路状态路由协议，虽然要额外考虑这样的情况，路由协议似

乎找不到通过节点 1 的最短路径。这将导致跨越网络空间的跳跃，因为节点 1 的实例是那些真正位于网络不同部分的节点。然而，链路状态协议已经能区分路由器和主机。我们早先掩盖了这一事实是因为我们没讨论它的必要。

5.2.10 移动主机路由

现在成千上万的人在移动的过程中使用计算机。比如坐在移动的汽车内使用无线设备，这是一类真正的移动情形；在一系列不同的位置使用笔记本电脑，这是一类漫游的情形。我们将使用术语**移动主机**（mobile host）示意上述任何一类场景中的设备，以便与从不移动的固定主机截然区分。越来越多的人希望无论走到世界上任何地方都能保持联系，就好像他们在家里一样。这些移动主机引入了新的复杂性：路由一个数据包到移动主机，网络首先要找到该主机。

我们将考虑的模型世界中，假设所有主机都有一个永久的**家乡位置**（home location），该位置永远不会改变。每个主机也有一个永久的**家乡地址**（home address），用来确定其家乡位置。家乡地址有点类似于电话号码，比如 1-212-5551212 表明这个号码在美国（国家代码 1）的曼哈顿地区（212）。移动主机所在系统的路由目标是使人们有可能利用固定的家乡地址来发送数据包，无论他们在哪里都能有效地把数据包送到。当然，这里的关键是要找到他们。

针对这种通信模型进行适当的讨论是有意义的。一种不同的模型是每当移动主机移动，以及拓扑发生变化后就重新计算路由。然后我们可以简单地采用本节前面所述的路由方案。然而，随着移动主机的数目越来越多，这种模式将很快导致整个网络陷入不断地计算新路由。使用家乡地址能大大降低这种负荷。

另一种方法是在网络层之上提供移动，这就是今天笔记本电脑典型的用法。当它们被转移到新的 Internet 位置，笔记本电脑就获得一个新的网络地址。这里新老地址之间不存在任何关联；网络也不知道它们属于同一台笔记本电脑。在此模型中，一台笔记本电脑可用于浏览网页，但其他主机无法给它发送数据包（例如，一个入境呼叫），除非有更高层的位置服务，例如移动之后再次登录 Skype。此外，主机在移动期间无法保持与网络的连接，而是必须重新启动建立新的连接。网络层的移动性对解决这些问题非常有用。

Internet 和蜂窝网络的移动路由采用的基本想法是移动主机把当前自己在哪里告诉给家乡位置的一台主机。这台主机称为**家乡代理**（home agent），它将以移动主机的名义采取行动。一旦它知道移动主机的当前位置，它就可以转发数据包以便数据包传递给移动主机。

图 5-19 显示了移动路由采取的动作。一个在西北部城市西雅图的发送者想发送一个数据包给通常设在美国纽约的主机。我们关心的一种情况是当移动主机不在家，相反它暂时到了圣地亚哥，怎么办？

在圣地亚哥的移动主机必须获得一个本地网络地址，然后才能使用网络。这是主机获得网络地址通常会发生的情况，我们将在本章后面讨论 Internet 时涉及这方面内容。本地地址称为**转交地址**（care of address）。一旦移动主机有了这个地址，它可以告诉其家乡代理它现在哪儿。它给家乡代理发送一个带有转交地址的注册消息（第 1 步）。该消息用图 5-19 中的虚线表示，以表明它是一个控制信息而不是一个数据报文。

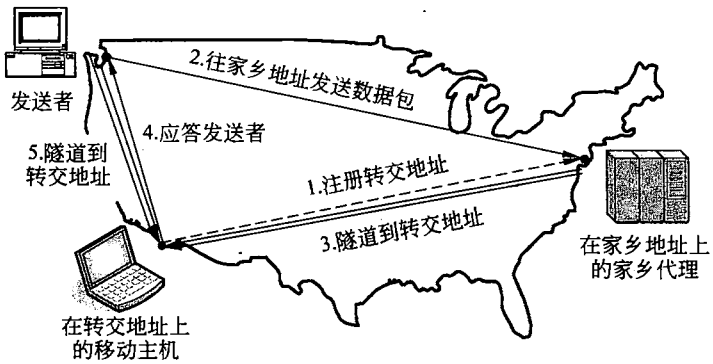


图 5-19 移动主机的数据包路由过程

接下来，发送者使用其永久地址发送一个数据包给移动主机（第 2 步）。这个数据包被网络路由到主机的家乡位置，因为这是主机家乡地址的所属地。在纽约，家乡代理截获该数据包，因为移动主机已经离家；然后用一个新的头包裹或者**封装**（encapsulated）该数据包，并把捆绑后的结果发送给转交地址（第 3 步）。这种机制称为**隧道**（tunneling）。该机制在 Internet 上非常重要，所以我们以后会更详细地考查它。

当封装后的数据包到达转交地址，移动主机解开它并提取出来自发送者的数据包；然后移动主机直接给发送者发应答数据包（第 4 步）。整个路由过程称为**三角路由**（triangle routing），因为如果远程位置离家乡位置很远时，这条路由可能是迂回的。作为第 4 步的一部分，发送者可借鉴当前的转交地址，把随后的数据包直接发送给移动主机。具体做法是通过隧道发送到转交地址（第 5 步），完全绕过家乡位置。无论什么原因，在移动主机移动时如果丢失了连接，则家乡地址可随时用来寻址到移动主机。

我们在整个描述过程中忽略了一个很重要的方面，即安全性。一般来说，当一个主机或路由器得到这种形式的信息“现在开始，请把 Stephany 的所有邮件发送给我，”时，可能会产生两个问题，我在跟谁谈话，以及这是否是个好主意。安全信息被包含在消息中，因此可以用加密协议检查消息的合法性。我们将在第 8 章学习加密协议。

移动路由算法有许多变种。上面叙述的方案是 IPv6 流动模型，这种移动主要形式用在 Internet（Johnson 等，2004），以及诸如 UMTS 蜂窝网络中基于 IP 的那部分。我们发现如果发送者是个固定节点，则情况能简单些；但设计方案时必须考虑这两个节点都是移动的情形。另外，主机可能是移动网络的一部分，例如，在一个平面上的计算机。对基本方案进行扩展就可以支持移动网络，不涉及任何主机部分的工作（Devarapalli 等，2005）。

有些方案利用外地（即远程）代理，类似家乡代理但放置在远程位置，或类似于蜂窝网络中的**访问位置寄存器**（VLR，Visitor Location Register）。然而，最近的研究方案并不需要外地代理；移动主机自己承担外地代理的行为。在这两种情况下，移动主机的临时位置仅限于被少量的主机获得（例如，移动主机、家乡代理和发送者），因此大型网络中的许多路由器不需要重新计算路由。

有关移动路由的更多信息，请参考（Perkin，1998，2002），以及（Snoeren 和 Balakrishnan，2000）。

5.2.11 自组织网络路由

我们现在已经看到当主机移动而路由器固定时，路由工作是如何进行的。另一种更极端的情形是路由器本身也是移动的。这种可能性主要发生在以下几种情形下：地震现场的紧急救援工作、战场上的军事车辆、海上航行的一队舰船或者一群配备了笔记本电脑的人们聚集在一个没有802.11网络的地区。

在所有这样或那样的情形中，每个节点用无线通信，并同时承担路由器和主机的双重角色。如果网络中的节点彼此靠近，那么该网络称为**自组织**（Ad hoc）网络，或者**移动自组织网络**（MANET，Mobile Ad hoc NETworks）。现在我们对Ad hoc网络进行简略的讨论，更详细的信息请参见（Perkins，2001）。

Ad hoc网络区别于有线网络的原因在于网络拓扑这个概念突然间被彻底抛到了九霄云外。节点可以来来去去，消失一会后突然又出现在一个新的地方。在有线网络中，如果一台路由器有一条通往某个目标的有效路径，那么该路径将会持续有效（除非出现故障）。而在Ad hoc网络中，拓扑结构可能每时每刻都在变化，所以路径的可取性和有效性自发地改变着，甚至没有任何警告。毋庸多说，这些情况使得Ad hoc网络的路由比固定路由更具挑战性。

针对Ad hoc网络目前已经提出了许多许多种路由算法。然而，由于Ad hoc网络相比移动网络很少被实际使用，因此无法确定哪种协议最有用。作为一个学习案例，我们将考查其中一个最流行的路由算法，即Ad hoc按需距离矢量（AODV，Ad hoc Ondemand Distance Vector）路由算法（Perkins和Royer，1999）。它是相对的距离矢量算法，适应在移动环境中工作，即节点的带宽有限和电源寿命相对较短。现在我们来看这个算法如何发现和维护路由的。

路由发现

在AODV中，到某个目的地的路由是按需发现的，即只有当某个节点要给目标节点发送数据包时才去找路径。这种方式可以节省大量不必要的工作，比如在路由使用之前拓扑就发生变化，因而之前的所有路由工作都浪费了。在任何时刻，Ad hoc网络都可以用连接节点的图来描述。如果两个节点可以直接通信，则这两个节点是连接的（也就是说，在图中它们之间有一条弧）。我们采用了一个简单但恰当的模型，即每个节点都可以与位于其覆盖范围内的其他节点通信。真实网络要复杂得多，可能有建筑物、山坡或者其他妨碍通信的障碍物，还可能出现这种情况，节点A连接到节点B，但是节点B却无法连接到A，因为A有比B更强大的功率。然而，为简化起见，我们假设所有的连接都是对称的。

为了描述AODV算法，请考虑图5-20中新形成的Ad hoc网络。假设节点A上的一个进程想要给节点I上的一个进程发送数据包。AODV算法在每个节点上维护了一张距离矢量表，以目标节点作为关键字，每个表项给出了有关该目标节点的信息，包括将数据包发送给哪个邻居才可以到达这个目标。首先，A检查自己的表，没有发现针对I的表项。因此，它现在必须去发现一条通向I的路径。这种只有当需要时才找寻路径的特性使得这个算法是“按需”（on demand）进行的。

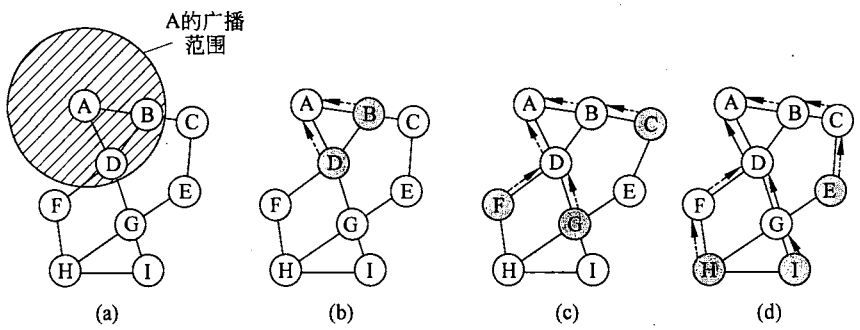


图 5-20

(a) A 的广播范围；(b) B 和 D 接收之后；(c) C、F 和 G 接收之后；
(d) E、H 和 I 接收之后阴影节点是新接收者，虚线表示可能的逆向路由，实线表示发现的路由

为了找到节点 I，A 构造一个 ROUTE REQUEST（路由请求）包，并且使用我们在 5.2.3 节描述的泛洪算法广播它。该包从 A 被传输到 B 和 D，如图 5-20（a）所示。每个节点重新广播，该包继续达到节点 F、G 和 C，见图 5-20（c）和 H、E 和 I，见图 5-20（d）。源端设置的一个序号用来淘汰泛洪过程中的重复请求包。例如，D 丢弃来自 B 的传输，如图 5-20（c），因为它已经转发过请求数据包了。

最后，请求包到达节点 I，节点 I 构造一个 ROUTE REPLY（路由应答）包。这个包沿着请求包所遵循的路径逆向单播给发送者。这项工作要求每一个中间节点必须记住给它发送请求包的节点。图 5-20（b）～图 5-20（d）显示了存储的逆向路由信息。每一个中间节点在转发应答数据包时还要将跳数递增一，告诉节点到目标节点多远。应答数据包还告诉每一个中间节点，使用哪个邻居能到达目标节点：正是给它们发送应答数据包的节点。中间节点 G 和 D 在处理应答数据包时把听到的最好的路由填入它们的路由表中。当该应答包到达节点 A，一条新路由 ADGI 就被创建了出来。

在一个大型网络中，AODV 算法会产生许多广播包，即使对于比较近的目标也是如此。为了减少开销，可以使用 IP 包的 Time to live 字段来限制广播的范围。该字段由发送者初始化，并且每经过一跳该字段被减 1。如果该字段被减到 0，则把包丢弃，不再广播。因此，路由发现过程可以作如下修改。为了发现一个目标，发送者广播一个 ROUTE REQUEST 包，其中 TTL 字段设置为 1。如果在一段合理的时间内没有应答包返回，则再发送一个 ROUTE REQUEST 包，这次将 TTL 字段设置为 2。以后的每一次尝试分别使用 TTL 的 3、4、5 值等。按照这种方法，搜索过程首先在本地进行，然后逐步扩大到更广的范围。

路由维护

因为节点可以移动或者关闭，所以网络的拓扑结构自然会发生变化。例如，在图 5-20 中，如果 G 关闭了，那么 A 并不会立即意识到它刚刚用过的通向 I 的路径（ADGI）已不再有效了。路由算法必须能够处理这样的情况。每个节点周期地广播一个 HELLO 消息，并且期望它的邻居做出响应。如果没有响应到来，消息的广播者就知道它的邻居已经离开接收范围或者失效，因而不再跟自己有连接。类似地，如果它试图给一个邻居发送数据包而没有得到响应，那么它也知道该邻居已经不再可用。

这些信息可被用来清除掉那些不再有效的路由。对于每一个可能的目标节点，每个节

点（比如说 N）记录自己的活跃邻居，即那些在最近的 ΔT 秒内给它发过到达该目标的数据包。当 N 的任何邻居变得不可达时，它就检查自己的路由表看哪些通往目标节点的路由使用了刚刚离开的邻居。对于这些路径中的每一个，通知相应的活跃邻居，因为它们经过 N 的路径不再有效，必须从路由表中删除。在我们的例子中，节点 D 从路由表中删除到 G 和 I 的表项，并通知 A 也删除到 I 的表项。一般情况下，活跃邻居再告诉它们自己的活跃邻居，如此递归下去，直到所有依赖走掉节点的路由从全部的路由表中删除掉。

在这个阶段，无效的路由已被清除出网络，发送者可以使用我们描述过的路由发现机制来发现新的有效路由。然而，这里有点复杂。回想一下，距离矢量协议可能遭受慢收敛问题，或者拓扑发生变化之后的无穷计数问题，这些问题把新的有效路由和旧的无效路由混淆在一起。

为了保证快速收敛，路由包括了一个由目标节点控制的序号。目标序号就像个逻辑时钟。目标节点每次发送一个新 ROUTE REPLY 时就递增该序号。发送者请求发现一条新路由时，就在 ROUTE REQUEST 请求报文中包括它最后使用的那条路由的序号，这个序号要么是刚刚被清除的路由的序号，要么作为初始值被设置为 0。该请求报文将被广播直到发现一条具有更高序号的路由。中间节点存储具有更高序号的路由，或者当前序号下具有更少跳的路由。

按需协议的精髓在于中间节点只存储正在使用中的路由。在广播期间了解的其他路由信息经过短暂延迟后会超时。相比需要定期广播路由更新信息的标准距离向量协议，只发现并存储那些要使用的路由有助于节省带宽和电池寿命。

到目前为止，我们已经考虑了网络中只有单条路由的情况，即从 A 到 I。为了进一步节省资源，路由发现和路由维护可被重叠的路由共享。例如，如果 B 也想给 I 发送数据包，它将执行路由发现。然而，在这种情况下，请求报文首先到达 D，而 D 已经有一条路由可以通向 I。因此节点 D 生成一个路由应答报文，告诉 B 这条路由而无须做任何额外的工作。

还有许多其他的 Ad hoc 路由方案。另一个著名的按需路由方案是**动态源路由**（DSR，Dynamic Source Routing）（Johnson 等，2001）。**贪婪边界无状态路由**（GPSR，Greedy Perimeter Stateless Routing）（Karp 和 Kung，2000）则探讨了另一种基于地理位置信息的不同的路由策略。如果所有节点知道它们的地理位置，则数据包的转发无须进行路由计算，只需简单地朝着正确的方向，并绕回躲开任何死角即可。究竟哪个路由协议能胜出将取决于 Ad hoc 网络的类型，必须实践证明对这种网络是有用的。

5.3 拥塞控制算法

（一部分）网络中存在太多的数据包导致数据包被延迟延迟和丢失，从而降低了传输性能，这种情况称为**拥塞**（congestion）。网络层和传输层共同承担着处理拥塞的责任。由于拥塞发生在网络内，正是网络层直接经历着拥塞，而且必须由它最终确定如何处理过载的数据包。然而，控制拥塞的最有效方法是减少传输层注入网络的负载。这就需要网络层和传输层共同努力协同工作。在这一章中，我们将着眼于拥塞控制在网络层方面的处理；在第 6 章，我们将通过覆盖拥塞控制在传输方面的处理来结束拥塞控制主题。

图 5-21 描绘了拥塞的发生。当主机发送到网络的数据包数量在其承载能力范围之内时，送达的数据包数与发送的数据包数成正比例增长。如果发送量增加了两倍，则送达量也增长了两倍。然而，随着负载接近承载能力，偶尔突发的流量填满了路由器内部的缓冲区，因而某些数据包会被丢失。这些丢失的数据包消耗了部分容量，因此，送达的数据包数量低于理想曲线。网络现在开始拥挤了。

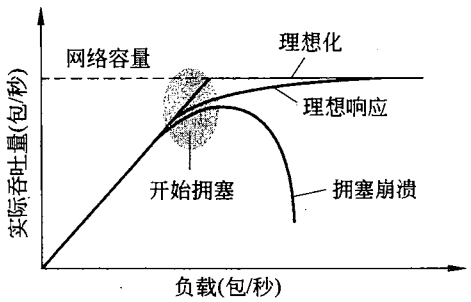


图 5-21 太多的流量导致性能急剧下降

除非网络是精心设计的，否则它极有可能会遭遇**拥塞崩溃**（congestion collapse），表现为随着注入负载的增加到超出网络的容量，网络性能骤降。这种情况非常有可能发生，因为数据包在网络内部遭遇了足够的延迟，使得它们离开网络后已经不再有用。例如，在早期 Internet 中有许多慢速的 56 kbps 链路，那么通过这样的链路发送数据包，包花在等待积压在前面数据包排空的时间远远超过了允许其在网络中的最大生存期，因此不得不被扔掉。这种被延迟很长时间的数据包会被认为已经丢失，因而需要重传。当发送方重传这些数据包时就出现了一个不同的故障模式。在这种情况下，相同数据包的副本将通过网络传送，再次浪费了网络的容量。为了捕捉这些因素，图 5-21 中的 y 轴代表**实际吞吐量**（goodput），表示网络传递有用（useful）数据包的传送速率。

我们想设计出这样的网络：首先尽可能地避免产生拥挤，其次如果它们确实变得拥挤时不会遭遇拥塞崩溃。不幸的是，拥塞不能完全避免。如果突然间，从三四条线路入境的数据包流都需要转发到相同的输出线路，则会形成一个队列。如果没有足够的内存来容纳所有这些数据包，数据包将被丢弃。为路由器添加更多的内存可以缓解这一点，但（Nagle，1987）认识到如果路由器拥有无限内存，拥堵情况将更加恶化而不是缓解。这是因为当数据包排到队列前面时，它们早已经超时（重复地）并且它们的副本也已经发送。这样使得事情变得更糟，而不是更好——最终导致拥塞崩溃。

低带宽链路或者处理数据包速度比线路速率还低的路由器也会变得拥挤不堪。在这种情况下，把一些流量导出瓶颈区域指向网络的其他部分可以改善这种拥塞状况。然而，最终网络的所有地区都将出现挤塞。在这种情况下，没有什么办法，只好卸下负载或建立一个更快的网络。

值得指出的是拥塞控制和流量控制之间有很大的差异，它们之间的关系非常微妙。拥塞控制的任务是确保网络能够承载所有到达的流量。这是一个全局性的问题，涉及各方面的行为，包括所有的主机和所有的路由器。与此相反，流量控制只与特定的发送方和特定的接收方之间的点到点流量有关。它的任务是确保一个快速的发送方不会持续地以超过接收方接收能力的速率传输数据。

为了看清楚这两个概念之间的差异，请考虑一个光纤网络，它由 100 Gbps 的光纤链路组成。在这个网络上，一台超级计算机试图给一台个人计算机传送一个大文件，这台个人计算机只有处理 1 Gbps 速率的能力。尽管这里并没有拥塞（网络本身没有任何问题），但是流量控制却十分需要，以便强迫超级计算机经常停下来，给个人计算机以喘息的机会。

作为另一个极端的情形，请考虑这样一个网络：它的线路是1 Mbps，有1000台大型计算机，其中一半的机器试图给另一半机器以100 kbps的速率传送文件。这里的问题并不是快速的发送方会淹没慢速的接收方，而是交给网络的总流量超过了网络的处理能力。

拥塞控制和流量控制之所以常常被混淆的原因在于处理上述两个问题的最好方式都是迫使主机慢下来。因此，一台主机接收到的“减速慢行”消息既可能来自不能处理负载的接收方，也可能来自不能处理负载的网络。我们将在第6章回到这个话题。

我们首先考查在不同时间尺度可使用的一些方法，作为学习拥塞控制的开始；然后，我们将考查把预防拥塞放在首位的方法，其次学习一旦拥塞发生以后的各种动态处理算法。

5.3.1 拥塞控制的途径

拥塞的出现意味着负载（暂时）大于资源（在网络的一部分）可以处理的能力。很自然人们能想到两个解决方案：增加资源或减少负载。如图5-22所示，这些解决方案通常应用在不同的时间尺度上，要么预先避免拥塞，要么一旦发生拥塞随之做出反应。

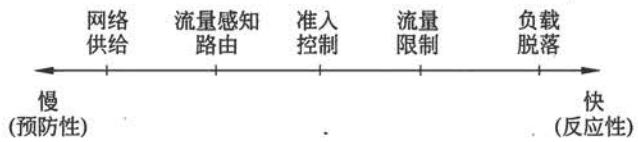


图5-22 拥塞控制方法的时间尺度

避免拥塞的最基本方法是建立一个与流量匹配良好的网络。如果存在一条低带宽的链路，大多数流量所走的路径都要经过这条链路，那么发生拥塞的可能性非常大。有时当出现严重拥塞时，可以动态增加网络资源，例如，把通常只用来备份的路由器或线路（使系统容错）打开，或者在公开市场上购买带宽。更多的时候，经常大量使用的链路和路由器都尽早实行升级。这就是所谓的**供给**（provisioning），在长期流量趋势推动下大约需要几个月的时间。

为了充分利用现有的网络容量，根据每天的流量模式度身定制路由，因为不同时区的网络用户每天醒来和睡觉的时间是不同的。例如，通过改变最短路径的权重可更改数据包路由，以便使流量远离频繁使用的路径。一些本地广播电台有直升机围绕着城市上空飞来飞去，及时报告城市道路拥堵情况，帮助它们的移动听众路由它们的数据包（汽车）绕开热点地区。这称为**流量感知的路由**（traffic-aware routing）。把流量拆分到多个路径也是有用的。

然而，有的时候不可能增加容量。那么对抗拥塞的唯一的办法就是降低负载。在一个虚电路网络中，如果新的连接将导致网络变得拥挤不堪，那么就应该拒绝这种新连接的建立。这种控制称为**准入控制**（admission control）。

以更细一点的粒度，当拥塞已迫在眉睫时，网络可以给造成问题的数据包源端传递反馈信息，要求这些源端抑制它们的流量，或者减缓流量本身。

这种方法存在两个困难：第一，如何确定网络开始拥塞了；第二，如何通知源端减缓速度。为了解决第一个问题，路由器可监测平均负荷、排队延迟或丢包情况。在所有情况下，数值上升表明拥挤情况越来越严重。

为了解决第二个问题，路由器必须参与到源端的反馈循环中。为了正常工作，必须仔细调整时间尺度。如果每次连续到达两个数据包，路由器就叫喊 STOP；而每当路由器空闲 20 微秒时，它就喊 GO，那么系统将会剧烈地摇摆不定，无法收敛。另一方面，如果它等待 30 分钟才能确定说什么，则拥塞控制机制的反应过于迟缓，一点用途也没有。及时提供反馈并不是件小事。需要额外关注的是，当网络早已被堵塞时路由器或许要发送更多的消息。

最后，当一切努力都失败，网络不得不丢弃它无法传递的数据包。这种方法的通用名称是**负载脱落**（load shedding）。一个选择丢弃哪些数据包的良好策略可以防止拥塞崩溃。

5.3.2 流量感知路由

我们考查的第一种方法是流量感知路由。我们在 5.2 节看过的路由方案采用固定链路权重。这些方案能适应拓扑结构的变化，但不能适应负载的变化。在计算路由时考虑链路负载的目的是把热点地区的流量转移出去，而热点地区是指网络中体验到拥塞的第一位置。

最直接的方式是把链路权重设置成一个（固定）链路带宽、传输延迟、（可变）测量负载或平均排队延迟的函数。在所有其他条件都相同的情况下，最小权重的路径更青睐那些轻负载的路径。

早期 Internet 使用的流量感知路由就是按照这个模型设计的（Khanna 和 Zinky，1989）。然而，这种路由存在一个危险。考虑图 5-23 所示的网络，这里网络被分为东部和西部，这两部分通过链路 CF 和 EI 相连。假设东西之间的大部分流量使用链路 CF，因此，这个链路负荷超重因而延迟增大。如果把排队延迟加入到计算最短路径的权重中，那么链路 EI 将变得更具吸引力。当新的路由表被安装好之后，大部分东西方的流量现在改走链路 EI，由此增加了此链路的负载。因此，在下一次路由更新时，CF 将成为最短路径。结果，路由表可能会剧烈地摇摆不定，从而导致不稳定的路由和许多潜在的问题。

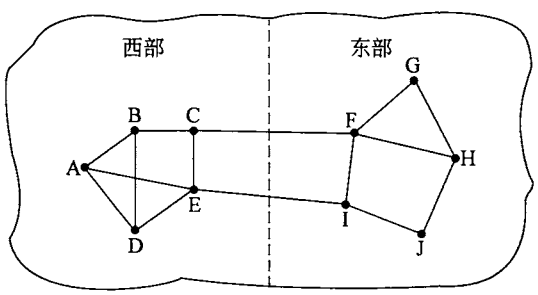


图 5-23 一个网络的东西两个部分由两条链路连接

如果忽略负载，只考虑带宽和传输延迟，这个问题就不会发生。尝试在路由权重中包括负载但将其限定在一个狭窄的范围可减缓路由振荡。两种技术有助于获得成功的解决方案。首先是多路径路由，即从源到目的地可以存在多条路径。在我们的例子中，这意味着可以把整个流量分散到东部和西部的两条链路上。第二个技术是把流量慢慢迁移，足够慢到路由算法得到收敛，比如（Gallagher，1977）方法。

由于存在这些困难，Internet 路由协议通常不依赖于负载来调整自己的路由。相反，在

路由协议外部通过慢慢改变它的输入来调整路由。这种方法就是所谓的**流量工程**（traffic engineering）。

5.3.3 准入控制

一种广泛应用于虚电路网络，防止出现拥塞的技术是**准入控制**（admission control）。其基本思想非常简单：除非网络可以携带额外的流量而不会变得拥塞，否则不再建立新的虚电路。因此，任何建立新的虚电路的尝试或许会失败。这种方法比起那种让更多的人进入一个已经繁忙的网络更好，因为更多的人只会使情况变得更糟糕。类似地，在电话系统中，当一台交换机实际超载时，它也会采用准入控制的方法，不再送出拨号音。

这种方法的诀窍是当一条新的虚电路将导致拥塞时如何工作。这项任务在电话网络中比较简单，因为电话呼叫所需的带宽固定（64 kbps 的无压缩音频）。然而，计算机网络中的虚拟电路有各种形状和大小。因此，如果我们想要采用准入控制，必须归纳出虚电路的一些流量特性。

流量往往用其速度和形状来描述。如何以一种简单而又有意义的方式来描述流量是困难的，因为流量呈现突发性——平均速率只讲述了故事的一半。例如，浏览网页时的流量变化很大，比具有固定长期吞吐量的流式电影更难以处理，因为突发性的网页流量更容易堵塞住网络中的路由器。捕获这个效果通常采用的描述符是**漏桶**（leaky bucket）或**令牌桶**（token bucket）。一个漏桶有两个参数约束了平均速率和瞬时突发流量大小。由于漏桶被广泛应用于服务质量，我们将在 5.4 节仔细描述它。

有了流量说明，网络就能决定是否接受新的虚电路。一种可能性是网络为它的每条虚电路保留其沿途的足够容量，这样就不会出现拥塞。在这种情况下，流量说明是一个网络向用户提供保证的服务约定。我们已经预防拥塞，但有点过早地转向了有关服务质量的话题；我们将在下一节返回到这里。

即使没有做出保证，网络也可以利用流量说明来进行准入控制。然后，我们的任务是估计出多少条电路能被网络的承载能力所容纳，而不会拥塞。假设虚拟电路可能爆发的流量速率高达 10 Mbps，如果所有流量都要通过同一条 100 Mbps 的物理链路，那么应该准许多少条电路？显然，10 条电路是可以准许的，并且不会有拥塞的危险；但这在正常情况下会浪费带宽，因为可能很少发生所有 10 个用户在同一时间以全速率传送数据的情形。在实际网络中，对过去行为的测量，即捕获数据传送的统计特征，可用来估计准入的虚电路数量，从而以可接受的风险换取更好的性能。

准入控制还可以和流量感知路由相结合，在虚电路建立过程中，考虑绕开流量热点区域的路由。例如，考虑图 5-24 的网络，这里显示的两台路由器已经被堵塞。

假设一台连接到路由器 A 的主机想要与另一台连接到路由器 B 的主机建立一个连接。通常情况下，这个连接将会经过其中一台拥塞的路由器。为了避免这种情形，我们可以重画网络，如图 5-24（b）所示，去掉拥塞的路由器和它们所有的线路。图中的虚线显示了一条可能的虚电路路径，它避开了拥塞的路由器。（Shaikh 等，1999）给出了这类负载敏感的路由方案设计。

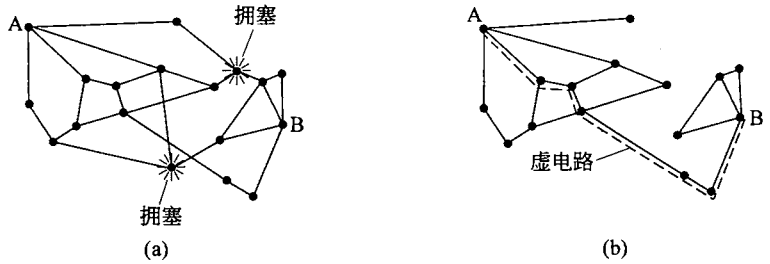


图 5-24
（a）一个拥塞的网络；（b）网络中不拥塞的部分（图中显示了从 A 到 B 的一条虚电路）

5.3.4 流量调节

在 Internet 和许多其他计算机网络中，发送方调整它们的传输速度以便发送网络能实际传送的流量。在这种设置下，网络的目标是在拥塞发生之前正常工作。而当拥塞迫在眉睫，它必须告诉发送方紧急刹车放慢传输速度。这种反馈是一种常态而不是针对特殊情况的一种处理。术语**拥塞避免**（congestion avoiddance）有时用来对比某个操作点，以示与已经变得（过度）拥挤的网络区分开。

现在让我们考查一些限制流量的方法，这些方法可用在数据报网络和虚电路网络。每个方法必须解决两个问题。首先，路由器必须确定何时快要接近拥塞，最好在拥塞发生之前能确定。为此，每个路由器可连续监测它正在使用的资源。三种可能的资源分别是输出线路的利用率、在路由器内缓冲的排队数据包，以及由于没有足够的缓冲而丢失的数据包数量。在这些可能性中，第二个是最有用的。平均利用率并没有直接考虑大多数流量的突发——50%的利用率对平滑流量来说或许很低，但对于变化很大的流量来说就太高了。丢失数据包的计数来得太迟。在数据包丢失时拥塞早就已经形成。

路由器内部的排队延迟直接捕获了数据包经历过的任何拥塞情况。它在大部分时间应该很低，但当有一个突发流量产生积压时会跳跃。为了维持良好的排队延迟估计 d，假设 s 表示瞬时队列长度的样值，则 d 可定期生成，并按如下方式进行更新

$$d_{new} = \alpha d_{old} + (1-\alpha)s$$

其中常数 α 决定路由器多快忘记最近的历史。这就是所谓的**指数加权移动平均**（EWMA，Exponentially Weighted Moving Average）。它能平滑流量的波动，相当于一个低通滤波器。每当 d 升高到某个阈值之上，路由器就要注意开始拥塞了。

要考虑的第二个问题是，路由器必须及时把反馈信息传递给造成拥塞的发送方。拥塞是网络的一种体验，但缓解拥塞则需要使用网络的发送方采取行动。为了传递反馈信息，路由器必须标识适当的发送方；然后提醒它们小心谨慎，别向本已拥挤的网络发送更多的数据包。不同的方案使用不同的反馈机制，我们马上对此分别描述。

抑制包

通知拥塞发送方的最直接方式是直接告诉发送方。在这种方法中，路由器选择一个被拥塞的数据包，给该数据包的源主机返回一个**抑制包**（choke packet）。抑制包中的目标地址取自该拥塞数据包。同时，在原来的拥塞数据包上添加一个标记（设置头部中的一位），

因而它在前行的路径上不会产生更多的抑制包。除此以外，数据包的转发过程如同平常一样。

当源主机收到了抑制包，按照要求它必须减少发送给指定目标的流量，比如说减少50%。在数据报网络中，发生拥塞时路由器简单地随机选择一个数据包，很有可能就把抑制包发给了快速发送方，因为发送方的速度越快，它的数据包排队在路由器队列中的数目就越多。这个协议隐含的反馈有助于防止拥塞，但又不会抑制任何发送方，除非真的发生了拥塞。出于同样的原因，很可能多个抑制包被发送到了一个给定的主机和目的地。主机应该忽略掉在固定时间间隔内到达的这些额外抑制包，直至其减缓流量的行为产生了效果。超过这个固定的时间间隔，从路由器进一步反馈来的抑制包则指出网络仍然处于拥塞状态。

早期Internet使用的一个抑制包例子是SOURCE-QUENCE消息（Postel，1981）。尽管它从来没有流行起来，不过，部分原因在于它产生的情况和效果都没有明确说明。现代Internet使用了另外一种“通知”设计方案，我们下面就描述它。

显式拥塞通知

除了生成额外的包发出拥塞警告外，路由器可以在它转发的任何数据包上打上标记（设置数据包头的某一个标志位）发出信号，表明它正在经历着拥塞。当网络传递数据包时，接收方可以注意到有个拥塞已经发生，在它发送应答包时顺便告知发送方。然后发送方可以像以前那样紧急刹车降低传输速率。

这种设计方案称为**显式拥塞通知**（ECN，Explicit Congestion Notification），且已被用在Internet上（Ramakrishnan等，2001）。这是早期拥塞信令协议的细化，尤其是对二进制反馈方案（Ramakrishnan和Jain，1988）的细化，该方案曾经用在DECNET体系结构。IP数据包头中的两位用来记录数据包是否经历了拥塞。数据包从源端发出时没有标记，如图5-25所示。如果它们通过的任何一个路由器正经历着拥挤，该路由器在转发数据包时将其标记为经历拥塞；然后接收方在它的下一个应答数据包里回显该标志作为显式拥塞信号。这显示为图中的一条虚线，它表明在IP层以上（如在TCP）。发送方必须紧踩油门控制传输，如同采用抑制包的情况。

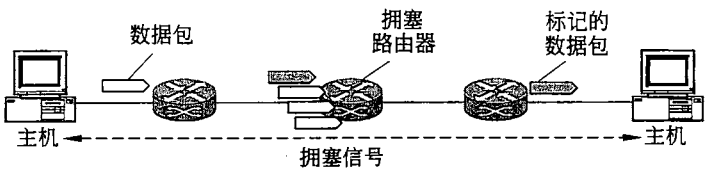


图5-25 显式拥塞通知

逐跳后压

当网络速度很高或者距离很远时，由于传播延迟的缘故，拥塞信号发出后到它产生作用这期间又有许多新的数据包已经被注入到网络。例如，请考虑这样的情形：旧金山的一台主机（图5-26中的路由器A）正在给纽约的一台主机（图5-26中的路由器D）发送数据，速度为OC-3的155 Mbps。如果纽约的主机现在用完了缓冲空间，它将花40毫秒的时间才能将抑制包发回给旧金山主机，告诉它降低传输速度。如果采用ECN指示，则需要更

长的时间，因为抑制消息通过接收方传递。抑制包的传播过程如图 5-26（a）中的第 2、3、4 步所示。在这 40 毫秒期间，A 又给 D 发送了 6.2 Mb 的数据。即使旧金山的主机立刻停机，传输管道里的 6.2 Mb 数据也将连续倾入网络，而且网络必须要对它们进行处理。只有在图 5-26（a）的第 7 个图中，纽约的路由器才会注意到数据包流减慢了。

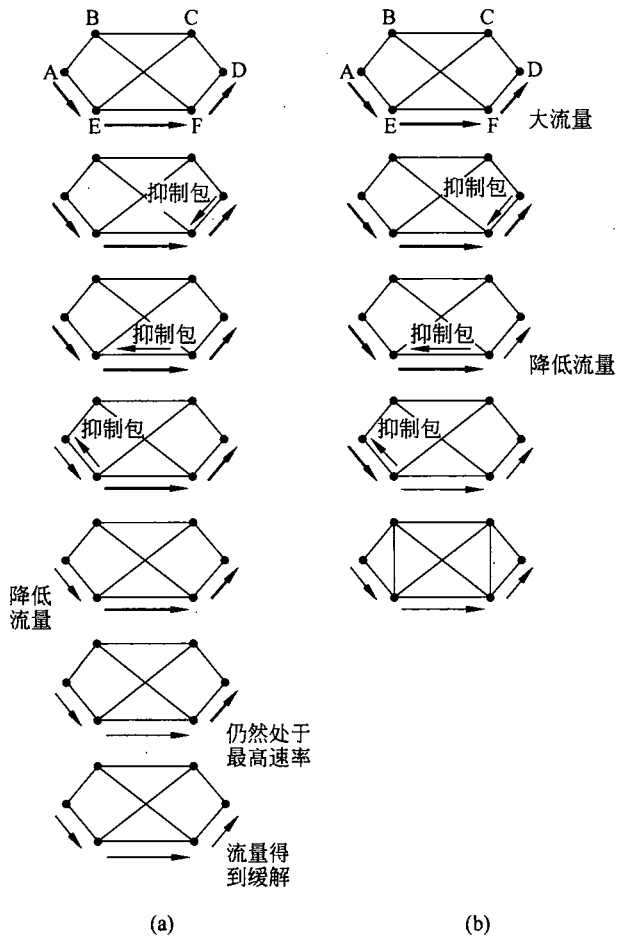


图 5-26
（a）抑制包只影响源端；（b）抑制包影响途径的每一跳

另一种办法是让抑制包在沿途的每一跳都发挥作用，如图 5-26（b）中的序列所示。在这里，只要抑制包到达 F，则 F 必须按照要求减慢发给 D 的数据包流。这样做的结果是要求 F 为 D 的数据包流分配更多的缓冲区，因为源主机仍然在全速发送数据，但是 F 这么做却让 D 立刻得到缓解，就好像电视广告中的头痛疗法一样。在下一步，抑制包到达 E，它告诉 E 减慢给 F 的数据包流。这一行动给 E 的缓冲区带来更大需求，但是却让 F 立即得到缓解。最后，抑制包到达 A，数据包流才真的减慢下来。

这种逐跳方案的实际效果是拥塞点上的拥塞现象很快得到了缓解，但是其代价是上游路径需要消耗更多的缓冲区空间。使用这种方法可以将拥塞消灭在萌芽状态，而不会丢失任何数据包。有关该算法思想的详细讨论请参考（Mishra 等，1996）。

5.3.5 负载脱落

当以上任何一种方法都无法消除拥塞时，路由器可以亮出它的杀手锏，即负载脱落。**负载脱落**（load shedding）是一种富有想象力的说法，它指当路由器因为来不及处理数据包而面临被这些数据包淹没的危险时，就将它们丢弃。此术语来源于电力发电领域，在炎热的夏日，当电力需求超过了供电能力，为了避免电力系统崩溃而有意切断某些特定区域的电力供给。

对于一个被数据包淹没的路由器来说，关键的问题是选择丢弃哪个数据包。首选的方案可能取决于使用网络的应用程序类型。对于文件传输，旧的数据包价值要高于新的数据包。这个原因我们可以用一个例子加以说明。如果丢弃数据包6，而保持数据包7～10，只会迫使接收方做更多的工作来缓冲它已经接收但尚不能使用的数据。相比之下，对于实时媒体流，新的数据包价值超过老的数据包。因为如果数据包被延迟并且错失了给用户的播放时间，那么该数据包就变得一无所用。

前一种策略（即旧的比新的好）通常称为葡萄酒（wine）策略，而后一种策略（即新的比旧的好）通常称为牛奶（milk）策略，因为大多数人更愿意饮用新鲜牛奶和品尝陈年葡萄酒。

更智能的卸载方式需要发送方的合作。一个例子是携带路由信息的数据包。这些数据包比普通数据包要重要得多，因为它们是被用来建立的路由；如果丢失它们，或许就会丢失网络连接。另一个例子是视频压缩算法，如MPEG，这些算法定期发送全帧，然后发送一系列与上一个全帧的差异帧。在这种情况下，应该优先丢弃那些属于差异帧的数据包，而不能丢弃属于全帧的数据包，因为未来数据包的工作依赖于它们之前的那个全帧。

为了实现智能丢弃政策，应用程序必须在它们的数据包上打上标记，指示网络它们有多重要。然后，当不得不丢弃数据包时，路由器可以首先丢弃重要性最轻一类数据包，然后是次重要一类数据包，以此类推。

当然，除非有一些明显的激励措施来避免每个数据包都标志成“非常重要”——“永远不要丢弃”，否则没有人会愿意把自己的数据包标示成不那么重要。通常计费和金钱可用来阻止轻浮的标志。例如，如果发送方购买了某项服务，该服务规定了一定的发送速度，网络可能允许发送方的发送速度超过他们购买的额度，只要他们把超出部分的数据包标记为低优先级。这种策略其实并不是一个坏主意，因为它可以更加有效地利用闲置资源；只要没有其他人对此有兴趣，主机就可以使用这些资源，但不能给它们在网络困难时也拥有这种权利。

随机早期检测

在拥塞刚出现苗头时就处理它比等拥塞形成之后再设法解决它更加有效。这一观察导致了一个针对负载脱落的有趣转折，即在所有的缓冲空间都精疲力竭之前丢弃数据包。

这一观点的动机在于大多数Internet主机没有从路由器获得ECN形式的拥塞信号。相反，主机从网络能获得的唯一可靠的拥塞迹象是丢包。毕竟，很难构造出一个路由器，在它超负荷工作时能做到不丢包。因此，诸如TCP这样的传输协议硬性规定了对丢包现象做

出拥塞反应，减缓源端的响应。这背后的逻辑推理是TCP是专为有线网络设计的，并且有线网络非常可靠，所以丢包大多是由于缓冲区溢出而不是传输错误造成的。无线链路必须恢复链路层传输错误（所以它们没有看到在网络层）以便TCP能正常工作。

可以利用这种情形来帮助缓解拥塞。在局面变得毫无希望之前让路由器提前丢包，但这里有个时间点的确定问题，即发送方何时采取行动以免为时过晚。解决这个问题的一个流行算法称为**随机早期检测**（RED，Random Early Detection）（Floyd和Jacobson，1993）。为了确定何时开始丢弃数据包，路由器要维护一个运行队列长度的平均值。当某条链路上的平均队列长度超过某个阈值时，该链路就被认为即将拥塞，因此路由器随机丢弃一小部分数据包。随机选择丢弃的数据包使得快速发送方发现丢包的可能性更大；因为在数据报网络中，路由器不能分辨出哪个源引起了网络的最大麻烦，因此随机选择丢弃的数据包或许是最佳选择。当没有出现期待的确认信息时，受此影响的发送方就会发现丢包，然后传输协议将放慢速度。因此，丢失的数据包起到了传递抑制包的同样作用，但却是隐含的，无须路由器发送任何显式信号。

相比那些只在缓冲区溢出才丢包的路由器，RED路由器能提高网络性能，虽然它们可能需要调整正常工作方式。举例来说，理想的丢包数量取决于有多少发送方必须得到拥塞通知。然而，如果ECN可用，那么它就是首选的选项。它的工作方式几乎完全一样，但提供了一个显式拥塞信号而不是依据丢包来判断是否拥塞；RED用在主机不能接收显式信号的环境里。

5.4 服 务 质 量

我们在上一节考查的技术主要用来减少拥塞并提高网络性能。然而，存在一些应用（和客户），它们对网络的性能保障有很强的需求，而不仅仅只是“在当前情况下尽力而为”。特别是多媒体应用往往需要具备最小延迟和最大吞吐量条件才能正常工作。在本节中，我们将继续我们的网络性能研究，但现在更加注重如何提供与应用需求相匹配的服务质量。这是一个Internet正在经历长期变革改进的领域。

提供良好服务质量的一个简单解决方案，就是建设有足够容量的网络，无论扔给它什么样的流量都能承担。这种解决方案的名称是**过度配置**（overprovisioning）。假设有一个体面的路由方案，那么这种网络将承载应用流量而没有重大丢失，并且以低延迟传递数据包。网络性能不会比这更好。在一定程度上，电话系统就是过度配置的一个例子，因为拿起电话而没有拨号音的情况很罕见。这里有太多的可用容量，所以需求总是能够得到满足。

这种解决方法的麻烦在于成本太昂贵。它基本上是用一堆钱来解决问题。服务质量机制让一个小容量网络以较低的成本来满足应用需求的解决途径。此外，过度配置基于预期的流量模式。如果流量模式变化太大，那么所有的赌注都将一去不回。有了服务质量机制，网络可以兑现其所做的性能保证，即使在流量高峰期拒绝了一些请求。

确保服务质量必须解决如下4个问题：

（1）应用程序需要网络什么样的质量？

（2）如何规范进入网络的流量？

（3）为了保障性能如何在路由器预留资源？

（4）网络能否安全地接受更多流量？

没有一种单一的技术能有效地解决上述所有问题。相反，研究人员已经陆陆续续开发出许多可用在网络层（和传输层）的各种技术。实际的服务质量解决方案要结合多种技术。为此，我们将介绍 Internet 服务质量的两个版本，即综合服务和区分服务。

5.4.1 应用需求

从一个源端发到一个接收方的数据包流称为一个**流**（flow）（Clark，1988）。在面向连接的网络中，一个流或许是一个连接上的全部数据包；而在无连接网络中，一个流是从一个进程发到另一个进程的所有数据包。每个流的需求可由四个主要参数来表示：带宽、延迟、抖动和丢失。总之，这些参数决定了一个流要求的**服务质量**（QoS，Quality of Service）。

图 5-27 列出了几种常见的应用和它们分别对网络质量的严格要求。请注意，网络需求小于应用需求，在这些情况下，应用程序可以改善由网络提供的服务。特别是，网络并不需要为可靠的文件传输真正做到无丢失，也并不需要为音频和视频的播放传递具有相同延迟的数据包。丢失部分可以通过重传来修复，一些抖动可以通过接收端缓冲数据包来平滑。然而，如果网络提供的带宽太少或者延迟太大，则无论应用程序做什么都无法补救这种情况。

应用	带宽	延迟	抖动	丢失
电子邮件	低	低	低	中等
文件共享	高	低	低	中等
浏览网页	中等	中等	低	中等
远程登录	低	中等	中等	中等
音频点播	低	低	高	低
视频点播	高	低	高	低
电话	低	高	高	低
视频电话	高	高	高	低

图 5-27 应用程序服务质量需求的严格程度

应用程序对它们的带宽需求是不同的，电子邮件、各种形式的音频和远程登录不需要太多的带宽，但文件共享和所有形式的视频应用则需要大量的带宽。

更有趣的是对网络延迟的需求。文件传输应用，包括电子邮件和视频，对延迟并不敏感。如果所有的数据包被均匀地延迟几秒钟，则无伤大雅；交互式应用，比如网上冲浪和远程登录，对延迟比较敏感；而实时应用，例如电话和视频会议，则有严格的延迟要求。如果电话中的每个字都被拖延得太久，那么用户会发现这样的连接是无法令人接受的。另一方面，播放一个服务器上的音频或视频不要求低延迟。

延迟的变化（即标准方差）或者数据包到达时间的变化称为**抖动**（jitter）。图 5-27 的前三个应用对相邻两个数据包到达时间的无规律性不敏感。远程登录对此有点敏感，如果连接遭遇太多的抖动，屏幕更新将呈现小的突发状。视频，尤其是音频对抖动非常敏感。假设用户正在通过网络观看视频，如果帧都刚好延迟了 2.000 秒，则不会损害播放效果。

但是，如果传输时间随机地在 1 秒和 2 秒之间变化，那么结果将非常可怕，除非应用程序能隐藏抖动。对于音频，即使是几毫秒的抖动都能被清楚听到。

相比音视频应用前 4 种应用对于丢失有更严格的要求，因为任何一位都不允许被错误递交给接收方。实现这个目标的通常做法是由传输层把被网络丢失的数据包重传。这是一项浪费的工作；如果网络一开始就将可能丢失的数据包拒绝或许更好。音视频应用可以容忍少量数据包丢失而无须重传，因为人们不会注意到声音短暂的停顿或者画面偶尔的跳跃。

为了适应各种应用，网络可以支持不同类别的 QoS。一个比较有影响的例子是 ATM 网络。该网络曾经是网络互联宏大远景的一部分，但却早已成为一种利基技术。ATM 网络支持：

（1）恒定比特率（比如电话）。

（2）实时可变比特率（比如压缩的视频会议）。

（3）非实时可变比特率（比如点播电影）。

（4）可用比特率（比如文件传输）。

这样的分类对于其他的用途或者其他的网络也是有用的。恒定比特率是指试图模拟一条线路，提供恒定的带宽和恒定的延迟。当视频信号被压缩时，由于有些帧的压缩比率大于其他的帧，所以会发生比特率变化的情况。如果发送的帧中包含了很多细节，那么它的发送就要求许多位，而只包含一面白色墙壁的帧可能会被压缩成极少的位。观看点播的电影实际上并不那么实时，因为接收端在开始播放前很容易地就缓冲了好几秒钟的视频，因此网络中的抖动仅仅导致“已存但未播放视频”（stored-but-not-played video）的变化。可用比特率适用于电子邮件这类应用，它们对延迟或抖动并不敏感，而且能做到得到什么样的带宽就用什么样的带宽工作。

5.4.2 流量整形

在网络做出服务质量保证之前，它必须知道要保证哪些流量的服务质量。在电话网络中，这种特性是简单的。例如，语音通话（非压缩格式）需要 64 kbps，它由一系列每 125 微秒的 8 位样本值组成。然而，数据网络中的流量是突发性的。通常包的到达是非均匀的，比如当流量速率可变（例如，压缩视频会议）、用户与应用程序交互（例如，浏览一个新的网页），以及计算机在不同的任务之间切换。突发流量比固定比特率的流量更难以处理，因为它们可以填满缓冲区并导致数据包丢失。

流量整形（traffic shaping）是指调节进入网络的数据流的平均速率和突发性所采用的技术。它的目标是允许应用程序发送适合它们需求的各种各样流量，包括带有某种程度的突发，但要有一个简单而有用的方式向网络描述可能的流量模式。当一个流在建立时，用户和网络（即客户和网络提供者）就该流的流量模式（即流量的形状）达成一致的协议。实际上，客户可以向服务提供者说“我的流量模型看起来是这样子的；你能处理它吗？”

有时候客户和服务提供者之间的约定称之为**服务等级约定**（SLA，service level agreement），特别是当它由聚合流量组成，并且存在一段比较长期的时间时，比如一个给定客户的全部流量。只要顾客履行了约定中的义务，并且根据商定的合约发送数据包，那么，服务提供者就承诺按时将数据包递交到目的地。

流量整形技术可以减少拥塞，因此有助于服务提供者兑现它的承诺。然而，要使流量整形工作，还有一个问题需要解决，那就是服务提供商如何确定客户是否遵守它们之间的约定，以及如果客户没有遵守约定，服务提供商该怎么办。超出约定模式之外的数据包可能会被丢弃，或者被打上低优先级标记。对一个流进行监测称为**流量监管**（traffic policing）。

流量整形和流量监管对于 P2P 和其他可能消耗掉任何可用带宽的数据传输并不那么重要，但它们对实时数据传输有非常重大的影响，比如音频和视频连接等这些具有严格服务质量需求的数据流。

漏桶和令牌桶

我们已经看过一种限制应用程序发送的数据量的方式：滑动窗口，它通过一个参数限制在任何特定时间内可以发送的数据量，因而间接地限制了数据速率。现在我们来看一种描述流量特征的更普遍的方式，漏桶算法和令牌桶流量。方法略有不同，但效果基本均等。请想象这样一个桶，它的底部有一个小洞，如图 5-28（b）所示。无论流入漏桶的水的速率多大，只要漏桶中还有水，那么水流出桶的速率是个恒定速率 R；如果桶内没有水，则流出桶的速率降为 0。此外，一旦桶内的水达到桶的容量 B，任何额外进入桶的水都会沿着桶侧分流，最终流失。

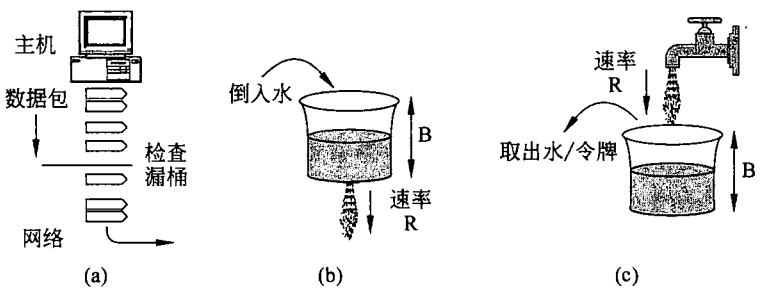


图 5-28
（a）整形数据包；（b）漏桶；（c）令牌桶

漏桶可以应用到注入网络的数据包上，对其进行整形和监管，如图 5-28（a）所示。概念上，每个主机在连接到网络的接口中包含一个漏桶。为了向网络发送数据包，必须有可能往漏桶中灌入更多的水。如果漏桶满时来了一个数据包，那么该数据包必须排入队列等漏桶空出来时再接纳，或者被丢弃。前一种策略作为操作系统的一部分可用在对主机进入网络的流量实施整形；后一种策略可用在服务提供商的网络接口，通过硬件对进入网络的流量实施监管。这个技术由（Turner，1986）提出，称为**漏桶算法**（leaky bucket algorithm）。

一个形式不同但效果相当的方法是把网络接口想象成一个漏桶，正在往里灌水，如图 5-28（c）所示。水龙头速率为 R，水桶容量为 B（跟以前一样）。现在，为了发送一个数据包，我们必须能够从桶内掏出水或令牌，俗称为内容（而不是往桶内注水）。桶内只可累积固定数量的令牌，即 B，不可能有更多的令牌；如果桶是空的，我们必须等更多的令牌到达才能发送另一个数据包。该算法称为**令牌桶算法**（token bucket algorithm）。

漏桶和令牌桶限制了一个流的长期速率，但允许其短期突发某个最高调节长度，不会改变延迟也不会受到任何人为的拖延。大量的突发数据将被一个漏桶流量整形器进行平滑处理，以便减少网络拥塞。作为一个例子，假设一个计算机能够产生高达 1000 Mbps 的数

据（125 万字节/秒），而且该网络的第一条链路也是以这个速度运行。主机生成的流量模式如图 5-29（a）所示。这种模式就是突发性的。每秒的平均速率超过 200 Mbps，即使该主机以峰值 1000 Mbps 的速率发送 16000 KB 突发数据。

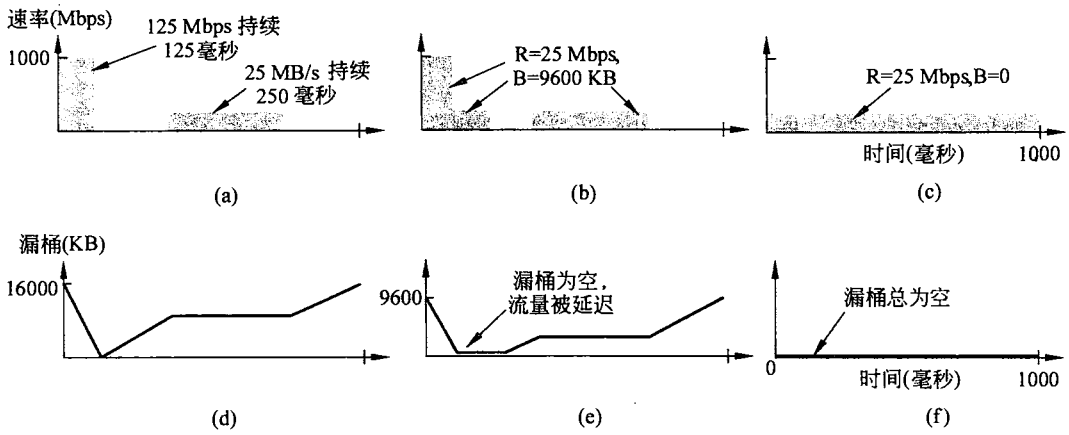


图 5-29

（a）主机发出的流量。通过一个具有以下容量速率为 200 Mbps 的令牌桶整形输出；（b）9600 KB 和（c）0 KB；通过一个以下容量速率为 200 Mbps 的令牌桶整形级别：（d）16 000 KB；（e）9600 KB；（f）0 KB

现在假设路由器可以在很短的时间间隔内接受峰值速率的数据，直到缓冲区填满为止。缓冲区大小为 9600 KB，远远小于突发流量。在很长的时间内，路由器的最佳工作速率不能超过 200 Mbps（比方说，因为这是给予客户的全部带宽）。这里的含义是如果以这种模式发送流量，其中有一些可能会被网络丢弃，因为它不适合路由器的缓冲区。

为了避免数据包丢失，我们可以在主机端用一个令牌桶对其流量进行整形。如果我们使用速率 R 为 200 Mbps，容量 B 为 9600 KB，则流量属于网络能够处理的范围。令牌桶的输出显示如图 5-29（b）所示。主机可以 1000 Mbps 全速发送一小段时间，直到它耗尽桶的容量；然后，它必须把发送速率削减到 200 Mbps，直到突发数据被发送出去。其效果是把突发分散在一段时间内，因为突发量太大无法一次都处理完。该令牌桶的水平如图 5-29（e）所示。开始的时候它是全速率发送，很快被突发耗尽；当它到达零时，只能以填满缓冲区的速率发送新的数据包；此时，不可能再有突发，直到桶恢复。没有流量发送时桶被慢慢填满，当以填充速度发送时桶保持当前水平。

我们也可以把流量整形成更少的突发。图 5-29（c）给出了一个以 R=200 Mbps 和容量为 0 的令牌桶输出。这是一个极端的案件，流量被完全平滑了：不允许任何突发，并且流量以一个稳定速率进入网络。对应的桶水平如图 5-29（f）所示，桶的水平总是空的。主机上的流量排队等待输出到网络，并且一旦允许总有一个数据包在等待发送。

最后，图 5-29（d）给出了一个 R=200 Mbps，容量 B=16 000 KB 令牌桶的水平。这是最小的令牌桶，通过它的流量不会被改变。这个桶可被网络路由器用来监管主机发送的流量。如果主机发送的流量符合它与网络商定的令牌桶，则流量将通过设置在网络边缘路由器上的相同令牌桶。如果主机以高于最快的或者突发速率发送，则令牌桶将耗尽。如果发生这种情况，流量监管器就知道主机发送的实际流量不是事先说明的那样；然后，它要么丢弃多余的数据包，要么降低其优先级，具体采取何种策略则取决于网络的设计。在我们

的例子中，桶在初始突发结束后被短暂清空，然后恢复到足够下一个突发的水平。

漏桶和令牌桶实现起来很容易。现在，我们描述一个令牌桶的操作。尽管我们已经描述水连续不断地流入和流出漏桶，但真正的实现必须离散度量。令牌桶用一个桶的水平计数器实现。计数器每时钟嘀嗒 ΔT 秒前进 R/ΔT 单位。在我们上面的例子中是每毫秒为 200 Kb。每次发送一个单位的流量到网络，然后计数器递减；只要计数器非零就一直可以发送。

如果数据包都是一样大小，桶的水平可以数据包计数（例如，200 Kb 对应 20 个大小为 1250 字节的数据包）。然而，通常采用的数据包大小是可变的。在这种情况下，桶的水平就由字节计量。如果剩余的字节数太低，不够发送一个大的数据包，则该数据包必须等待，直到下一个时钟滴答（或者甚至更长的时间，如果填充率很小）。

计算最大突发长度（直到桶清空）需要一点技巧。突发长度刚好要超过 9600 KB 除以 125 KB/s 的除数，因为输出的同时有更多的令牌到达。如果我们称突发长度为 S 秒，最大输出率为 M B/s，令牌桶的容量为 B 字节，令牌到达率为 R B/s，我们可以看到，突发输出最多可包含 B+RS 字节。同时，我们也知道 S 秒时间的最大速度突发长度为 MS。因此，我们有

$$B + RS = MS$$

解上述等式，得到 S=B/(M−R)。代入参数 B=9600 KB，M=125 MB/s，R=25 MB/s，我们得到突发时间大约 94 毫秒。

令牌桶算法的一个潜在问题是它把大的突发传输下降到了一个长期速率 R。人们通常需要把峰值速率降下来，但又不希望降到长期速率（同时也不能提高长期速率让更多的流量流入网络）。平滑流量的一个办法是在第一个令牌桶之后插入第二个令牌桶。第二个桶的速率应比第一个桶高许多。基本上，第一桶表述流量特征，确定其平均速率，但允许少量突发；第二桶降低了进入网络的突发峰值速率。例如，如果第二个令牌桶的速率设定为 500 Mbps，并且容量设置为 0，则初始突发进入网络的峰值速率为 500 Mbps，这比之前的 1000 Mbps 的速率低了许多。

有效使用所有这些桶可能需要不少技巧。当令牌桶被用来整形主机流量时，数据包必须排队并延迟到桶允许它们被发送时；当令牌桶被网络路由器用于流量监管时，该算法确保了发送的数据包不会比允许发送的更多。不过，这些工具提供了把网络流量整形成更易于管理形式的方式，从而协助满足服务质量的要求。

5.4.3 包调度

调整网络流量的形状是保证服务质量的一个良好开端。然而，要想提供性能保证，我们必须沿着数据包经过网络的路径预留足够的资源。为了做到这点，我们假设一个流的数据包都遵循同样的路径。如果这些数据包被随机地分散在路由器上是很难保证什么的。因此，有必要在源端和接收方之间建立起类似虚电路的路径，属于这个流的所有数据包必须遵循这条路由。

在同一个流的数据包之间以及在竞争流之间分配路由器资源的算法称为（**数据**）**包调度算法**（packet scheduling algorithms）。为不同的流可以预约的潜在资源有以下 3 种。

（1）带宽。

（2）缓冲区。

（3）CPU 周期。

第一种资源“带宽”最为显见。如果一个流要求 1 Mbps，而输出线路的容量为 2 Mbps，那么，试图在这条线路上直接发送 3 个流将不可能正常工作。因此，预留带宽意味着对任何一条输出线路都不能超额预订。

第二种常常短缺的资源是缓冲区空间。当一个数据包抵达时，它通常被保留在路由器的缓冲区直到可以从选择的输出线路上发送出去。当流相互竞争时缓冲区可以起到吸收小的突发流量的作用。如果没有可用的缓冲区，那么该数据包不得不被丢弃，因为没有地方可以存放数据包。对于好的服务质量，可以为某个特定的流预留一定的缓冲区，从而该流不必跟其他的流争用缓冲区。因此，只要该流需要，总能获得可用的缓冲区，直到达到某个最大的限额。

最后，CPU 周期也是一种稀有资源。每个数据包的处理需要占用路由器的 CPU 时间，所以，一台路由器每秒只能处理一定数量的数据包。虽然现代路由器能快速处理大多数数据包，但某些类型的数据包需要得到 CPU 更快的处理，比如我们将在 5.6 节讨论的 ICMP 数据包。为了保证这些数据包及时得到处理，必须确保 CPU 没有过载使用。

数据包调度算法负责分配带宽和其他路由器资源，具体做法是确定下一次把缓冲区中的哪些数据包发送到输出线路。我们在解释路由器工作时已经描述了最直截了当的调度程序。每个路由器把需要转发的数据包排入相应输出线路的队列，直到它们可以发送，并且发送顺序与到达队列的顺序相同。这种算法称为**先入先出**（FIFO，First-In First-Out），或等价的**先来先服务**（FCFS，First-Come First-Serve）。

FIFO 路由器在队列满时通常丢弃新到的数据包。由于新近到达的数据包会排在队列末尾，因此这种行为称为**尾丢包**（tail drop）。这种处理方式非常直截了当，或许你可能想知道是否还存在其他的处理方式。事实上，我们在 5.3.5 节所描述的 RED 算法在平均队列长度增大时随机选择丢弃一个新到达的数据包。我们将要描述的一些其他调度算法在缓冲区满时也会创造其他的机会来确定丢弃哪些数据包。

FIFO 调度算法易于实现，但它无法提供良好的服务质量，因为当存在多个流时，一个流很容易影响到其他流量的性能。如果第一个流来势汹汹并且发送大的突发数据包，它们将盘踞在队列中。按数据包的到达顺序处理意味着咄咄逼人的发送方能吃掉其数据包穿越的路由器的大部分容量，因而饿死其他流量，降低它们的服务质量。雪上加霜的是想通过路由器的其他流的数据包很有可能被延迟，因为它们不得不排在队列中那个大流量发送者的许多数据包后面。

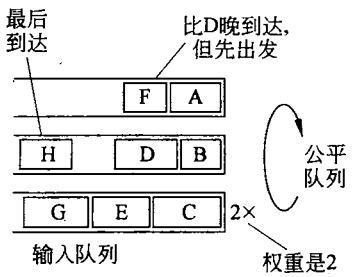
研究人员已经制定了许多数据包调度算法，这些算法可提供很强的流间隔离，并且能阻止干扰企图（Bhatti 和 Crowcroft，2000 年）。其中第一个数据包调度算法是由（Nagle，1987）提出的**公平队列**（fair queueing）算法。该算法的实质是针对每条输出线路，路由器为每个流设置单独的队列。当线路空闲时，路由器循环扫描各个队列，如图 5-30 所示；然后，从下一个队列中取出第一个数据包发送。以这种方式，如果某条输出线路被 n 个主机竞争，则每发送 n 个数据包中每个主机获得发送一个数据包的机会。正是这个意义上的公平，使得所有流量以同样的速率发送数据包。即使源端发送更多的数据包也不会提高这个

速率。

图 5-30 公平队列的循环机制

虽然从一开始，该算法就有一个缺陷：它给使用大数据包的主机比使用小数据包的主机提供了更多的带宽。（Demers 等，1990）建议对该算法的循环策略进行改进，把原来的“数据包接数据包”的循环方式改成“字节接字节”的循环方式。这里的诀窍是计算一个虚拟时间，这个时间指每个数据包发送完毕所需要的轮数。每一轮循环从所有有数据待发送的队列中排空一个字节；然后，按照数据包的结束时间顺序排队，并以该顺序真正发送数据包。

图 5-31 显示了该算法以及该算法的一个例子，例子给出了分别属于三个流的数据包到达和完成时间。如果一个数据包的长度为 L，它的完成时间恰好是启动之后的第 L 轮循环。启动时间要么是前一个数据包的完成时间，或者是数据包的到达时间（如果它到达时队列为空）。



数据包	到达时间	长度	完成时间	输出次序
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8

(b)

图 5-31
（a）加权公平队列；（b）数据包的完成时间

现在看图 5-32（b）中的表，考查最上面两个队列的前两个数据包，数据包的到达顺序是 A、B、D 和 F。数据包 A 在第 0 轮循环到达，长度为 8 个字节，因此其完成时间是第 8 轮；同样，数据包 B 的完成时间为 11；当数据包 B 在发送时数据包 D 到达，因此它的完成时间要从 B 结束时开始算 9 字节循环，最终完成时间为 20。类似地，F 的完成时间为 16。如果没有新的数据包到达，则相对的发送顺序是 A、B、F、D，尽管 F 在 D 之后到达。有可能在最上面的那个流到达另一个很小的数据包，它的完成时间在 D 的完成时间之前。如果 D 的传输尚未开始，那么该小数据包就会跳跃到 D 的前面。公平队列不能抢占当前正在传输的数据包。因为数据包的发送是整体行为，因此公平队列只是理想“字节接字节”方案的近似法。但这是一个很好的近似，任何时候数据包都保持着其理想的传输方案。

实际上这个算法存在一个缺点，即它给所有主机以相同的优先级。在许多情况下，比如，给予视频服务器比文件服务器更多的带宽也是可取的。要做到这点很容易，只要每轮循环时给视频服务器两个或两个以上字节。这种修改后的算法称为**加权公平队列**（WFQ，

Weighted Fair Queueing)。设每一轮的字节数是一个流的权重 W，我们现在可以给出计算完成时间的公式：

$$F_i = \max(A_i, F_{i-1}) + L_i / W$$

其中 A_i 为到达时间，F_i 为完成时间，L_i 是数据包 i 的长度。在图 5-31（a）中，最下面的队列权重为 2，所以你可以看到在图 5-31（b）给出的完成时间表中，它的数据包被发送得更快。

另一个实际的考虑是算法实现的复杂度。WFQ 要求数据包按照它们的完成时间插入到一个有序队列中。如果有 n 个流，则针对每个数据包，至少需要 O(logN)操作，这在同时存在许多流的高速路由器上很难实现。（Shreedhar 和 Varghese，1995）描述了一种称为**赤字循环**（deficit round robin）的近似算法，该算法的实现非常有效，针对每个数据包只有 O(1)次操作。WFQ 广泛使用了这个近似算法。

还存在着一些其他类型的调度算法。一个简单的例子是优先级调度，每个数据包被标记一个优先级别。高优先级数据包始终先于任何缓冲的低优先级数据包发送。具有相同优先级的数据包按照 FIFO 顺序发送。但是，优先级调度的缺点是一个高优先级的突发数据包可以饿死低优先级的数据包，这将导致后者无限期地等待下去。WFQ 通常提供了一个更好的选择。通过给高优先级队列更大的权重，例如，高优先级数据包往往会连续发送好几个（因为只有相对较少的数据包具有高优先级），然而，即使存在高优先级的流量，仍然有一定比例的低优先级数据包被陆续发送。一个高低优先级系统基本上是一个双队列的 WFQ 系统，其中高优先具有无限的权重。

数据包调度的最后一个例子是数据包携带时间戳并且按照时间戳顺序发送。(Clark 等，1992）描述了一种设计，当数据包被路径上的一系列路由器发送时，时间戳记录该数据包离调度之后或者之前还有多远。排在路由器队列中其他数据包后面的数据包趋向于稍后调度，而首先获得服务的数据包则趋向于优先调度。按数据包的时间戳顺序发送数据包有利于加快慢速数据包，而且同时放缓快速数据包。结果是网络传递的所有数据包具有更一致的延迟。

5.4.4 准入控制

现在我们已经看过了服务质量的所有必要因素，是时候把它们放在一起真正提供服务质量保证了。服务质量的保证通过准入控制的过程来建立。我们首先看的是用于控制拥塞的准入控制，尽管它还很脆弱，但却是性能的保证。我们现在考虑的保障机制更强大，但模型是相同的。用户向网络提供一个有 QoS 需求的流量；然后，网络根据自己的容量以及向其他流做出的承诺决定是否接受或拒绝该流。如果接受，网络就要提前在路由器上预留容量，以便保证新流发送时的服务质量。

沿着数据包经过网络所用的路由，沿途每个路由器都要预留相应的资源。路径上任何一台没有预留的路由器可能变得拥挤不堪，而且单个拥塞的路由器就可打破 QoS 保证。许多路由算法都是在每个源和每个接收方之间找出一条最好路径，并且通过该最好路径发送流量。如果最佳路径上没有足够的剩余容量，这种算法就可能导致某些流遭遇拒绝。如果新流所需要的带宽超过了剩余容量，那么通过选择另外一条产能过剩的不同路径，仍然可能为新流提供 QoS 保证。这就是所谓的 **QoS 路由**（QoS routing）。(Chen 和 Nahrstedt，1998）

对这些技术做了概述。有可能把到达每个目的地的流量拆分到多条路径，以便更容易地发现产能过剩的路径。一个简单的方法是选择同等成本（equal-cost）的路径，并且将流量均等或者按比例分摊到出境链路的容量。然而，更复杂的算法也是可用的（Nelakuditi 和 Zhang，2002）。

给定一条路径，决定接受或拒绝流量并不是一件简单的事情，相对而言，比向一个产能过剩的路由器请求资源（带宽、缓冲区和 CPU 周期）还要稍微复杂一些。首先，尽管某些应用程序可能知道它们的带宽需求，但很少知道有关缓冲区或 CPU 周期的需求，因此至少需要不同的方式来描述流，并且将这种描述转换成路由器资源。我们马上就会看到这点。

其次，某些应用比其他应用更能容忍偶尔错过的最后期限。应用程序必须从网络做出的服务质量保障类型中做出选择，是否硬性保证或者大部分时间都能得到保证。如果所有其他条件都相同，则每个人都希望获得硬性保障，但困难在于它们的代价太昂贵，因为它们限制了最坏情况下的行为。对于应用来说为大多数数据包提供保证往往足够了，而且获得此类保障的更多流可被固定容量支持。

最后，某些应用程序可能愿意就流的参数讨价还价，而其他一些应用可能不会。例如，一个电影观众通常运行速率在 30 帧/秒，如果没有足够的可用带宽来支持 30 帧/秒，他可能愿意把运行速率降到 25 帧/秒。类似地，每帧的像素数、音频带宽等其他特征或许要做出相应的调整。

因为流协商过程中会涉及许多方（包括发送方、接收方，以及发送方和接收方之间沿途的所有路由器），所以必须用特定参数来精确描述流，并且这些参数是可以为各方所协商的。这样的一组参数称为**流规范**（flow specification）。通常，发送方（比如视频服务器）生成一个流规范，在此流规范中指出它所希望使用的参数；当这份流规范沿着路径传播时，沿途每个路由器对它进行检查，并根据需要修改相应的参数。参数只能往降低质量方面修改，而不能修改成提高流的质量（比如说，修改成低数据率，而不能提高数据率）。当流规范到达另一端的时候，流的参数就被建立起来了。

作为一个关于流规范的例子，考虑如图 5-32 所示的例子。它基于综合服务的 RFC 2210 和 RFC 2211，综合服务是 QoS 的一种设计方案，我们在下一节将对此进行详细的讨论。例子显示的流规范共有 5 个参数。前两个参数“令牌桶速率”和“令牌桶容量”，使用了一个令牌桶，第一个参数给出了发送方可以传输的最大持续速率，即在相当长时间间隔内的平均速率；第二个参数给出了短时间隔内可以发送的最大突发量。

参数	单位
令牌桶速率	字节/秒
令牌桶容量	字节
峰值速率	字节/秒
最小数据包尺寸	字节
最大数据包尺寸	字节

图 5-32 流规范实例

第三个参数，峰值速率指网络能容忍的最大传输速率，即使在很短的时间间隔内也要受此约束。发送方在短时间内突发传输时也不得超过这个速率。

最后两个参数指定了数据包的最小和最大长度，包括传输层和网络层的头（比如 TCP 和 IP）。最小长度的规定是有用的，因为不管一个数据包有多短，在对它进行处理时需要一些固定时间。一台路由器可能每秒钟能够处理 10 000 个 1 KB 长度的数据包，但是可能无法处理 100 000 个 50 字节长度的数据包，尽管后者的数据率比前者更低一些。最大数据包长度很重要，因为可能会存在一些无法超越的内部网络限制。例如，如果路径上有一段

是以太网，那么，不管其他网络如何处理数据包，数据包的最大长度将被限制于不得超过1500字节。

一个有趣的问题是路由器如何将一个流规范转变成一组特定的资源而进行预留。乍一看，似乎很明显，假设一个路由器有条链路运行速率为1Gbps，并且平均包的长度为1000位，那么它能以每秒一百万个包的速率处理数据包。实际情况并非如此，由于负载的统计波动，链路总是会有空闲周期。如果链接需要利用容量的每一位来完成自己的工作，那么甚至数位的空转都可能生成一个永远也无法摆脱的积压。

即使实际负载略微轻于理论容量，也可能排起队列，而且延迟仍然可能会发生。请考虑这样的情形：数据包随机到达，平均到达率为每秒λ个数据包；数据包具有随机长度，并且可以平均速率每秒 μ 个数据包发送到链路上。假设数据包的到达和服务均服从泊松（Poisson）分布（这就是所谓的M/M/1排队系统，其中M指Markov，即Poisson），则利用排队理论可以证明，一个数据包所经历的平均延迟T为：

$$T=\frac{1}{\mu}\times\frac{1}{1-\lambda/\mu}=\frac{1}{\mu}\times\frac{1}{1-\rho}$$

这里ρ=λ/μ是CPU的利用率。第一个因子1/μ是在没有竞争情况下的服务时间。第二个因子是指由于跟其他流竞争而导致的减慢因素。例如，如果 λ=950 000 数据包/秒，μ=1 000 000数据包/秒，那么，ρ=0.95，并且每个数据包经历的平均延迟将是20微秒，而不是1微秒。这个时间值包含了排队时间和服务时间，正如当负载很低时（即λ/μ≈0）能看出这一点。如果在流的路径上有30台路由器，那么仅仅排队延迟就会达到600微秒。

（Parekh 和 Gallagher，1993，1994）给出了一个对应于带宽和延迟性能保证的流规格说明方法。该方法在流量源端采用（R，B）令牌桶整形，而在路由器采用WFQ。每个流都有一个WFQ权重W，该权重值足够大到能排空速率为R的令牌桶，如图5-33所示。例如，如果流的速率为1 Mbps，路由器和输出链路的容量为1 Gbps，那么在路由器的输出链路上该流的权重必须大于所有流的全部权重的千分之一。这保证了该流具有最低带宽。如果流得不到足够大的速度，那么该流就不允许进入网络。

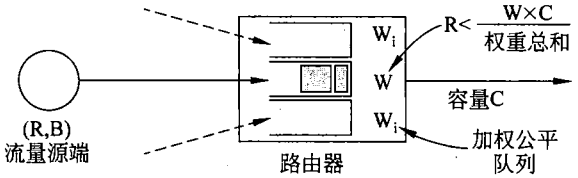


图5-33 采用令牌桶和WFQ的带宽和延迟保证

流的最大排队延迟是令牌桶突发大小的函数。考虑两个极端情况。如果流量是平缓的，没有任何突发，数据包将以它们到达的速率被路由器排空，此时将不会有排队延迟（忽略打包效果）。另一方面，如果流量被保存起来形成堆积，那么一次到达路由器的最大突发数为B。在这种情况下，最大排队延迟D将是以保证带宽排空突发的时间，或B/R（再次，忽视打包效果）。如果这个延迟过大，那么流必须向网络请求更多的带宽。

这些保障是硬性的。令牌桶约束了源端的突发性，公平队列隔离了给予不同流的带宽。这意味着无论其他竞争对手在路由器上的行为如何，该流将得到其带宽和延迟保证。那些其他流量不能破坏这种保障，即使积累流量并一次全部发送出来也不会影响到上述流量的

服务质量。

此外，这个结果对任何网络拓扑结构下通过多个路由器的路径都成立。每个流得到了最低带宽，因为这是每个路由器做出的带宽保证。每个流得到最大延迟的理由则有点微妙。在最坏的情况下，冲击第一个路由器的突发流量与其他流的流量展开竞争，它将被推迟到最大延迟 D。然而，这种延迟能平滑突发流量。反过来，这意味着这次突发将再也不会在后面的路由器遭受进一步的排队延迟。总的排队延迟最大为 D。

5.4.5 综合服务

从 1995 年到 1997 年之间，IETF 做了很大的努力来设计流式多媒体的体系结构。这项工作最后产生了 20 多个 RFC，从 RFC 2205～2210。此项工作的通用名称是**综合服务**（integrated service），主要针对单播和组播应用。比如说，用户从一个新闻网站抓取一段视频片段就是单播应用的一个例子；一组数字电视台将它们的节目以 IP 数据包流的方式广播到各地的许多接收端，就是组播应用的一个例子。下面我们将重点关注组播，因为单播可以看作是组播的一个特例。

在许多组播应用中，组的成员可能会动态地发生变化。例如，用户进入一个视频会议，然后感觉厌烦了就切换到一个肥皂剧或者橄榄球频道。在这样的情况下，让发送方提前预留带宽的做法并不能很好地工作，因为这要求每个发送方必须跟踪它的听众们的全部加入和离开情况。对于一个拥有上百万个用户的电视传输系统而言，这样的设计显然不能工作。

RSVP——资源预留协议

在综合服务体系结构中，最主要的也是网络用户可见的那部分是**资源预留协议**（RSVP，Resource reSerVation Protocol）。RFC 2205～2210 文档对该协议进行了详细描述。该协议的主要功能是预留资源，发送数据则需要使用其他协议。RSVP 允许多个发送方给多个接收组传送数据，也允许接收方自由地切换频道，并且在消除拥塞的同时优化带宽的使用。

在最简单的形式下，RSVP 使用了基于生成树的组播路由，关于用生成树来实现组播路由前面已经有所讨论。每个组都分配一个组地址。为了给一个组发送数据包，发送方把该组的地址放到这些数据包中。然后，标准的组播路由算法建立起一棵覆盖所有组成员的生成树。路由算法并不是 RSVP 的一部分。与常规组播的唯一不同之处是这里的组播需要一些额外的信息，这些信息被周期性地组播给生成树中的路由器，告诉它们在内存中维护特定的数据结构。

举例来说，请考虑图 5-34（a）中的网络。主机 1 和主机 2 是组播发送方，主机 3、4 和 5 是组播接收方。在这个例子中，发送方和接收方是分离的；但是一般情况下，这两个集合可以重叠。针对主机 1 和主机 2 的组播树分别如图 5-34（b）和（c）所示。.

为了获得更好的接收效果并且消除拥塞，一个组中的任何接收方都可以沿着树给组播发送方发送一条预留消息。利用前面讨论过的逆向路径转发算法，该预留消息被传播到发送方。在沿途的每一跳，路由器会注意到此预留消息，并预留必要的带宽。在前一节中我们已经了解到如何使用加权公平队列包调度算法来进行资源预留。如果没有足够的带宽，它就返回报告失败。当这条预留消息到达组播发送方时，从发送方到该接收方一路上的带宽都得到预留，因为沿着生成树完成了预留。

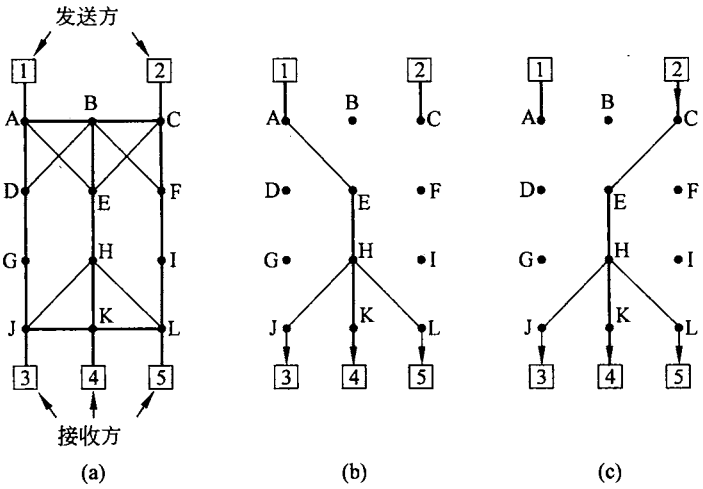


图 5-34

（a）一个网络；（b）主机 1 的组播生成树；（c）主机 2 的组播生成树

图 5-35（a）显示了一个这种资源预留的例子。在这里，主机 3 请求一条通向主机 1 的信道。一旦该信道被建立起来，则从主机 1 到主机 3 的数据包流将不会再遭遇拥塞。现在请考虑，如果接下来主机 3 为了能同时观看两套电视节目，要预留一条通向另一个发送方（即主机 2）的信道，会怎么样？第二条预留的路径如图 5-35（b）所示。请注意，从主机 3 到路由器 E 之间需要两条独立的信道，因为传输的是两个独立的流。

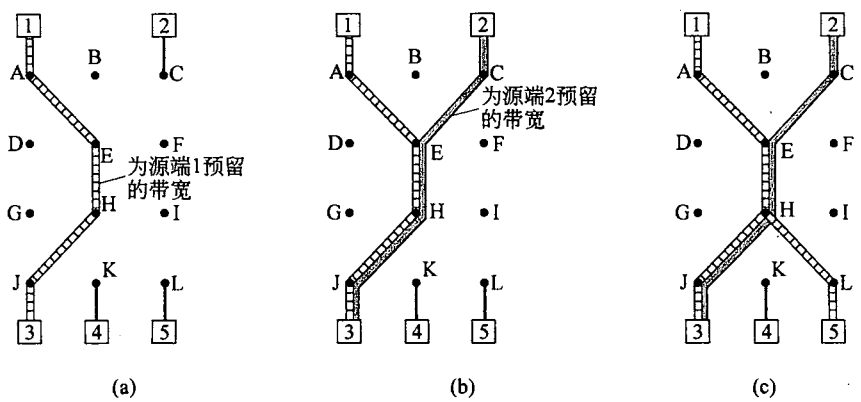


图 5-35

（a）主机 3 向主机 1 请求一条信道；（b）然后主机 3 向主机 2 请求第二条信道；（c）主机 5 向主机 1 请求一条信道

最后，在图 5-35（c）中，主机 5 决定观看主机 1 传送的节目，因而也请求预留带宽。首先，在主机 5 到路由器 H 之间，要预留专门的带宽。然而，路由器 H 看到自己已经有了一份来自主机 1 的流，所以既然所需的带宽已经被预留了，就没有必要再次预留。请注意，主机 3 和主机 5 请求的带宽数量可能会不相同（比如，主机 3 在小屏幕上播放节目，只需要低分辨率信息），所以，预留的带宽数量必须足够大，才能满足最贪婪的那个接收方。

在进行资源预留时，接收方可以（有选择地）指定一个或者多个期望接收的数据源。它也可以说明在预留期间这些选择是否固定不变，或者是否希望以后还可以改变数据源。路由器利用这些信息来优化带宽的使用计划。尤其是，如果两个接收方都同意以后不再改

变数据源的话，那么它们只需共享一条路径即可。

采用这种完全动态策略的理由是将被预留的带宽与数据源的选择分离开。一旦接收方已经预留了带宽，那么它可以切换到另一个数据源，保留现有路径上那部分带宽并且对新数据源仍然有效。例如，如果主机 2 正在实况传输几个视频流，比如这是一个拥有多个节目频道的 TV 电视台，那么主机 3 可以随意地在这几个频道之间切换，而根本不用改变它所做的预留：路由器并不关心接收方正在观看什么节目。

5.4.6 区分服务

基于流的算法有能力为一个或者多个流提供非常好的服务质量，因为它们在沿途路由上预留了必要的资源。然而，这些算法有一个缺点，它们都需要为每个流预先进行设置。当存在数千或数百万个流时，这些算法就不能很好地扩展使用；而且，在路由器中为每个流维护一个内部状态很容易导致路由器的崩溃；最后，为了设置数据流，需要修改的路由器代码量很大，并且还涉及复杂的路由器-路由器之间消息交换。而且，尽管综合服务方面的研究工作还在继续并往前推进，但 RSVP 几乎没有实际部署，甚至类似的实现也很少。

基于这些原因，IETF 还设计了另一个更加简单的服务质量方法，该方法很大程度上由每个路由器本地实现，无须事先提前设置流，也不牵涉整条路径。这种方法称为**基于类别**（class based）的服务质量（相对于基于流的服务质量），还有另一个更流行的名称为**区分服务**（differentiated service）。IETF 已经对该方法的体系结构进行了标准化，RFC 2474、2475 和其他一些 RFC 文档对区分服务进行了详细描述。下面我们介绍区分服务。

区分服务可以由一组路由器提供，这些路由器构成了一个管理域（比如一个 ISP 或者一家电话公司）。管理规范定义了一组服务类别，每个服务类别对应于特定的转发规则。如果一个客户已经订购了区分服务，那么进入到该管理域的客户数据包就会被标上它们属于哪类服务。这个信息可由 IPv4 和 IPv6 数据包（在 5.6 节讨论）的区分服务字段携带。服务类别定义为**单跳行为**（PHP，per hop behaviors），对应于数据包在每个路由器得到的待遇，而不是对数据包在整个网络中的保证。具有某种单跳行为（比如优质服务）的数据包相比其他数据包（比如普通服务）可以获得更好的服务。属于同一个类别的通信流量可能要先经过处理以便符合特定的形状特征，比如通过一个具有特定排空速率的漏桶。商业嗅觉灵敏的运营商可能对每个优质服务类别的数据包收取额外的费用，或者每个月收取固定的额外费用并允许最多 N 个优质服务类别的数据包。请注意，这种方案并不要求提前设置，也没有资源预留，更不需要花时间为每个流进行端到端的协商。这些服务特征与综合服务有很大的不同，也使得区分服务相对容易实现。

基于类别的服务在其他工业领域也有。例如，包裹托运公司通常提供昼夜送达、两天内送达和三天内送达等不同的服务；航空公司提供头等舱、商务舱和经济舱服务；长途列车一般也有多个服务等级。甚至巴黎的地铁还有两种服务类别。对数据包而言，类别之间的差异可以体现在延迟、抖动、发生拥塞时被丢弃的概率，以及其他的可能性（如果不是以太帧的话，还有其他一些可能）。

为了更好地理解基于流的服务质量与基于类别的服务质量之间的差异，请考虑一个实例：Internet 电话。如果采用基于流的方案，每个电话呼叫都拥有它自己专用的资源，从而

服务质量可以保证。如果采用基于类别的方案，则所有的电话呼叫合起来预留同一份预留给电话类别的资源。这些资源不能被网页浏览类别或者其他类别的数据包夺走，但是，任何一个电话呼叫也得不到任何独自享用的私有资源。

加速转发

服务类别的选择取决于每个运营商，但是由于通常情况下大多数数据包需要在不同运营商的网络之间转发，所以，IETF 已经定义了某些与网络无关的服务类别。最简单的服务类别是**加速转发**（expedited forwarding），RFC 3246 对此有详细的描述。我们先从这个种类别开始讨论。

加速转发背后的思想非常简单。我们可以把服务类别分为两种：常规的和加速的。绝大多数通信流量属于常规流量，但是有一小部分数据包需要加速转发。加速类别的数据包应该可以直接通过网络，就好像不存在其他任何数据包一样。这样，它们将获得低丢失、低延迟和低抖动服务——这正是 VoIP 所需要的服务。这种“双管道”系统的一种符号化表示方法如图 5-36 所示。请注意，这里依然只有一条物理线路，图中的两根逻辑管道只是表示了为不同类别服务预留带宽的方法，而不是有两条物理线路。

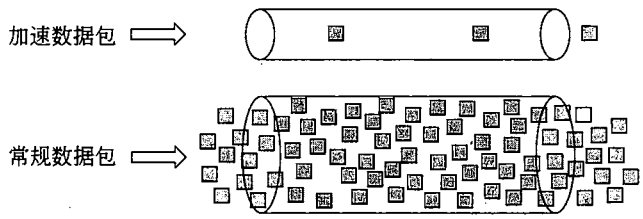


图 5-36 加速数据包体验一个无流量网络

实现这种策略的一种做法如下：把数据包分类成加速和常规两种，并做相应的标记。这一步可以在发送主机上完成，或者在入口（第一个）路由器上做。在发送主机上进行分类的好处是它可以利用更多的信息来确定哪些数据包属于哪些流。这个任务可以由网络软件甚至操作系统来执行，以免更改现有的应用程序。例如，主机为 VoIP 数据包标记上加速服务标记的做法正在逐步流行起来。如果这些数据包穿越支持加速服务的公司网络或 ISP，它们将获得优惠待遇；如果网络不支持加急服务，也没有什么坏处。

当然，如果标记工作由主机来做，则入口路由器很可能就承担着监管流量的任务，确保客户没有发送比他们所付费用更多的加速流量。在网络内部，路由器针对每条出境线路可以有两个输出队列，一个用于加速数据包，另一个用于常规数据包。当一个包到达时，它被排入相应的队列。加速队列获得的优先级要高于常规队列，例如，使用优先级数据包调度算法。通过这种方式，加速数据包看到的是一个没有负载的网络，即使事实上普通流量的负载很重。

确保转发

管理服务类别的一种更精细方案称为**确保转发**（assured forwarding）。确保转发服务由 RFC 2597 定义，它规定了 4 种优先级别，每种级别都拥有自己的资源。前 3 种服务类别或许可分别称为金质、银质和铜质。而且，标准还针对正经历拥塞的数据包，定义了 3 种丢弃概率：低、中、高。将这两个因素结合起来，共有 12 种服务类别。

图 5-37 显示了一种确保转发下的数据包处理办法。第一步是将数据包分成 4 个优先级别之一。如前所述，这一步有可能在发送主机上完成（如图中所示），也可能在入口路由器上完成，高优先级数据包的速率可作为服务供给的一部分由运营商限制。

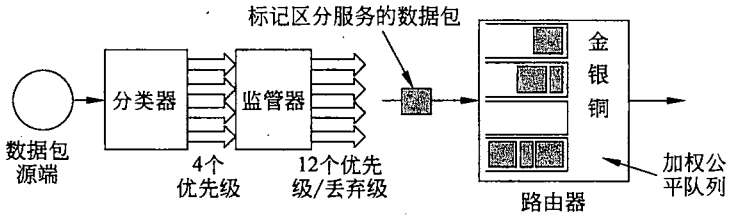


图 5-37 确保转发的一种可能实现

下一步是确定每个数据包的丢包类别。每个优先级的数据包穿过一个诸如令牌桶这样的流量监管器，可以完成这个工作。该监管器让所有的流量都通过，但它根据突发流量大小来标识数据包的丢包概率：符合小突发量的标为低概率、大于小突发量的标为中概率、超过大突发量的标为高概率。然后将优先级和丢包概率结合起来，编码到每个数据包中。

最后，数据包由网络内部的路由器处理，路由器上的数据包调度器区分不同类别的数据包。一般的选择是针对 4 个优先级类别采用加权公平队列，给予较高的类别以较高的权重。在这种方式中，高优先类别的数据包将获得大部分的带宽，但较低优先级类别的数据包也不会完全被饿死。例如，如果一个类别的权重是低一级类别权重的两倍，则高类别的数据包将获得两倍于低类别数据包的带宽。在同类别的优先级内部，具有较高丢包概率的数据包可被运行的算法优先丢弃，比如我们在 5.3.5 节了解过的 RED 算法（随机早期检测）。当刚刚出现拥塞苗头且路由器还有缓冲空间时，RED 就开始丢弃数据包。在这个阶段，路由器仍然有缓冲空间用来接受低丢包概率的数据包，而丢弃高丢包概率的数据包。

5.5 网络互联

到现在为止，我们一直隐含着假设所讨论的网络是一个同质网络，即每台机器在每一层上使用同样的协议。不幸的是，这样的假设太过于乐观。实际上，存在着许多不同的网络，包括 PAN、LAN、MAN 和 WAN。我们已经描述了以太网、线缆上的 Internet、固定和移动电话网络、802.11、802.16 等。大量协议被广泛应用于这些网络的各个层次。在下面的章节中，我们将详细讨论当两个或者多个网络连接起来形成**网络互联**（internetwork），或简单的**互联网**（internet）时所涉及的一些问题。

如果每个人都使用单一的网络技术，那么将网络连接起来非常简单，而且经常会出现某种类型的网络占据主导位置，比如以太网。一些专家推测技术的多样性会自动消失，只要每一个人都意识到 [填入你最喜爱的网络] 是多么的美妙。但是，不要指望这种情况会发生，历史证明这是一相情愿的想法。不同类型的网络解决不同的问题，因此以太网和卫星网络可能永远都不会相同。重用现有的系统增加了限制，比如在线缆之上、电话网络之上和电源线之上运行数据网络，从而造成网络特性的发散和分叉。这就是异质性的由来。

如果总会有不同的网络，如果我们不需要互连它们，事情更简单。显然，这也是不可

能的。Bob Metcalfe 推测一个具有 N 个节点的网络的价值等于节点之间的连接数，或者 N^2（Gilder，1993）。这意味着大型网络比小型网络更有价值，因为它们允许更多的连接，所以始终有将小型网络联结起来的激励。

Internet 是这种**互连**（interconnection）的最佳例子（我们将把 Internet 的首字母写成大写的“I”以示和其他互联网或者连接在一起的网络区别开来）。纳入所有这些网络的目的是使得任何一种网络的用户都可以和其他种类的网络用户沟通。当你向 ISP 支付 Internet 服务费用时，收取的费用取决于你的线路带宽，但你真正支付的是能够与同样连接到 Internet 上的其他主机交流数据包的能力。毕竟，如果你只能将数据包发送到同一城市的其他主机，Internet 就不会像现在这样受欢迎。

由于网络往往在一些重要方面有所不同，因此一个网络得到来自另一个网络的数据包并不总是那么容易。我们必须解决异质性的问题，以及因相互连接起来的互联网增长非常大而造成的规模问题。我们首先考查不同的网络是如何的不同，以便寻找到相应的解决方法；然后，我们将学习 Internet 网络层协议 IP（Internet 协议）获得成功的经验，包括穿越网络的隧道、互联网络的路由和数据包拆分技术。

5.5.1 网络如何不同

网络的不同可体现在不同方面。比如不同的调制解调技术或帧格式这样的差异属于物理层和数据链路层内部，这些差异我们在这里不关心。相反，在图 5-38 我们列出了暴露在网络层面的一些差异。正是这些被掩盖的差异使得对互联网络的操作比对单个网络操作更加困难。

项目	某些可能性
提供的服务	无连接与面向连接
寻址	不同大小，扁平或层次
广播	提供或者缺乏(组播同样)
数据包尺寸	每个网络有自己的最大尺寸
有序性	有序和无序传递
服务质量	提供或缺乏；许多不同种类
可靠性	丢包的不同级别
安全性	隐私规则，加密等
参数	不同超时值，流规范等
记账	按连接时间、包数、字节数或不收费

图 5-38 网络的某些不同之处

当一个网络上的某个源端发出的数据包必须要经过一个或者多个外部网络才能到达目标网络时，网络之间的接口可能会产生许多问题。首先，源端必须能够寻址接收方。如果源端在以太网络上，而接收方在 WiMAX 网络上，我们应该做什么？假设我们可以在以太网络标识一个 WiMAX 目标，数据包将从一个无连接网络穿越到面向连接的网络，可能需要在短时间内建立一个新的连接，这将带来延迟，而且，如果连接不是被更多的数据包使用，开销很大。

许多特殊的分歧也必须能够被容纳。我们如何在一个不支持组播的网络上把数据包分发给一组用户？不同网络规定的最大数据包尺寸不同，这也是困扰网络互联的主要因素。如何通过一个最大尺寸为1500字节的网络传递长度为8000个字节的数据包？如果一个面向连接网络上的数据包经过一个无连接网络，它们可能以不同于发送顺序到达接收方。这可能是发送方没有想到的，因而可能会引起接收方的（不愉快的）诧异。

经过一些努力，这类差异可以被掩盖掉。例如，连接两个网络的网关可以为每个接收方生成单独的数据包，来替代支持组播的更好网络；一个大的数据包可能被拆分，分段发送，然后再重组还原。接收端缓冲收到的数据包，并按顺序递交它们。

网络还可能在其他大的方面有所区别，而这些方面是难以调和的。最明显的例子是服务质量。如果一个网络具有强大的QoS，而其他网络只提供尽力而为的服务，那么就不可能为端-端的实时流量做带宽和延迟保证。事实上，除非运行的尽力而为网络利用率较低，否则服务质量可能只是说说而已很难真正采用，因此这不大可能成为大多数ISP的目标。安全机制也有问题，但至少可以在不具备安全性的网络顶端设置加密技术，来保证保密性和数据完整性。最后，当平时正常使用的网络突然变得昂贵起来，计费上面的差异可能会产生令人不悦的账单，正如有数据计划的漫游手机用户所发现的那样。

5.5.2 何以连接网络

连接不同网络的方式有两种基本选择：第一，我们可以制造这样的设备，它能将每种网络的数据包翻译或转换成每个其他类别网络的数据包；第二，像出色的计算机科学家那样，尝试在不同网络的上面增加一个间接层，并且构造一个公共层来解决这个问题。这两种情况下，新设备被放置在网络之间的边界上。

早期，（Cerf和Kahn，1974）提出用一个公共层来隐藏现有网络的差异。这种方法已经取得了巨大成功，他们提出的层最终被分别融入到TCP和IP协议。差不多40年之后，IP成为了现代Internet的基础。由于这个成绩，Cerf和Kahn于2004年被授予图灵奖，该奖相当于计算机科学领域的诺贝尔奖。IP提供了一种通用的数据包格式，所有路由器都认识这种数据包，因而这种数据包几乎可以通过所有的网络传递。IP已经将其研究活动从计算机网络扩展到电话网络。它还可以运行在传感器网络和其他微型设备上，这些微小设备一度被认定资源太少而无法支持IP。

我们已经讨论了几种用来连接网络的不同设备，包括中继器、集线器、交换机、网桥，路由器和网关。中继器和集线器只是将比特从一根导线移动到另一根导线，它们大多是模拟设备，不了解有关高层协议的任何知识。网桥和交换机工作在链路层。它们可以被用来构建网络，但只能处理轻微的协议转换，例如，在10、100和1000 Mbps以太网交换机之间传递帧。我们在这一节的重点是相互连接工作在网络层的设备，即路由器，把网关等高层互连设备留到以后介绍。

让我们先探讨在较高的层次如何用一个共同的网络层来互连不同的网络。一个由802.11、MPLS和以太网网络组成的互联网络，如图5-39（a）所示。假设源主机在802.11网络上，要给以太网上的目标机器发送数据包。由于这两个网络技术不同，而且它们又被另一类型的网络（MPLS）隔离，因此在网络之间的边界需要做一些额外的处理。

因为，一般来说不同的网络有不同形式的地址，数据包携带一个网络层地址，它可以标示这三个网络上的主机。当一个数据包从 802.11 网络被发送到 MPLS 网络时，首先到达第一个网络边界。802.11 提供了无连接服务，但 MPLS 提供了面向连接的服务。这意味着，必须建立一条穿过该网络的虚电路。只要数据包沿此虚电路传输，就能到达以太网网络。在第二个网络边界，数据包可能太大以至于无法通过，因为 802.11 可以使用比以太网大得多的帧。为了解决这个问题，数据包被拆分为段，每个段单独发送。当这些段达到接收方后，它们被重新组合在一起。这时，数据包才完成自己的旅程。

这个过程中的协议处理如图 5-39（b）所示。源端接受来自传输层的数据，并生成一个带有公共网络层的头，这个例子中采用的是 IP 协议。网络层的头包含了最终的接收方地址，这个地址被用来确定数据包应该通过第一个路由器发送。因此，包被封装在 802.11 帧内并且被发送出去，帧的目标地址是第一个路由器的地址。在路由器，数据包从帧的数据字段中被提取出来，802.11 帧头被丢弃；现在路由器检查数据包中的 IP 地址，并查询其路由表；根据这个地址，路由器决定将数据包发送到第二个路由器。对于路径中的这部分，路由器必须建立一条到第二个路由器的 MPLS 虚电路，并且必须用 MPLS 报头封装该数据包。在远端，MPLS 头被丢弃，并在此检查网络地址，以便寻找下一跳网络层。这下一跳就是目标网络本身。由于数据包太长无法通过以太网发送，它被拆分为两部分。每一部分被放入以太网帧的数据字段，并被发送到目的地的以太网地址。在接收方，剥掉每个帧的以太网帧头，将帧的数据内容重新组合。至此，数据包终于到达了它的目的地。

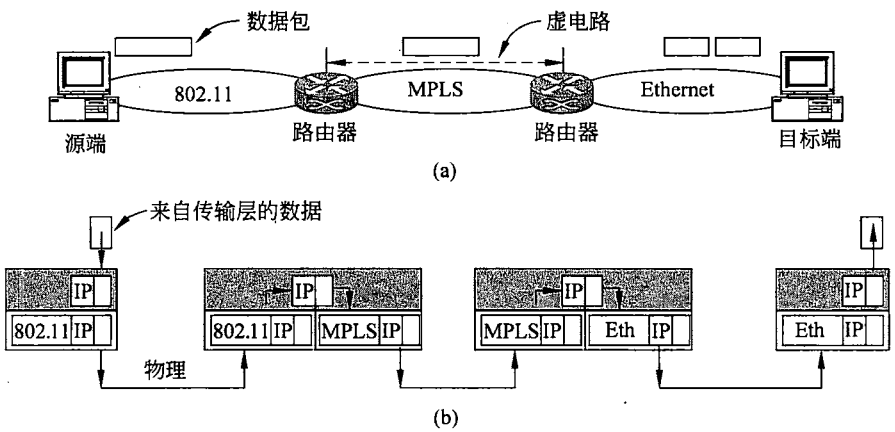


图 5-39

（a）跨越不同网络的数据包；（b）网络层和链路层的协议处理

这里可以观察到路由情况和交换（或桥接）情况的本质区别。在路由器上，数据包被从帧中提取出来，数据包中的网络地址被用来决定把它转发到哪里；而在交换机（或网桥）上，整个帧是根据其 MAC 地址传送的。交换机不必了解正在被交换的数据包所采用的网络层协议，同样路由器也不必了解交换机所采用的链路层协议。

不幸的是，网络互联并不像我们说的那么容易。事实上，引入网桥的目的就是用它们将不同类型的网络联结起来，或者至少把不同类型的局域网联结在一起。它们的做法是把一个 LAN 的帧翻译成另一个 LAN 的帧。然而，这样的工作并不理想，出于同样的原因网络互联是很困难的：LAN 特征上的差异很难掩盖，比如不同的最大数据包尺寸、LAN 有

优先级或者没有。今天，网桥主要用来连接链路层的同类网络，路由器用来连接网络层不同的网络。

网络互联技术在建设大型网络方面非常成功，但仅当有一个共同网络层时才能工作。事实上，随着时间的推移已经存在许多网络协议。当公司认定有一个自己控制的专用格式是它们的商业优势时，想要获得大家对单一格式的一致认可就很困难了。除了IP这个例外，现在近乎普遍的网络协议是IPX、SNA和AppleTalk，这些协议仍然没有得到广泛使用，但总是会有其他协议的。现在最相关的例子可能是IPv4和IPv6。虽然这些协议是IP的两个版本，但它们是不兼容的（否则就没有必要建立IPv6了）。

可以处理多个网络协议的路由器称为**多协议路由器**（multiprotocol router）。它必须翻译协议，或者把连接留给更高的协议层。这两种方法都不完全令人满意。更高层的连接，比如说TCP，要求所有的网络都实现TCP（也许并非如此）。因此，它限制网络只能被使用TCP的应用使用（不包括许多实时应用）。

另一种方法是在网络之间转换数据包。然而，除非数据包格式比较相近，具有相同的信息字段，否则这种转换将永远是不完整的，并且往往注定要失败。例如，IPv6地址为128位长，不管路由器如何努力尝试，它们肯定不适合填入32位的IPv4地址字段。在一个网络上同时运行IPv4和IPv6已被证明是部署IPv6的一个主要障碍（为公平起见，首先要让客户明白为什么他们应该使用IPv6）。在两个根本不同的协议之间进行翻译可能产生更大的问题，比如无连接和面向连接的网络协议。鉴于这些困难，很少有人去尝试这种转换。可以说，甚至IP也只能在作为最小分母的一种服务时才工作得很好。IP对运行自己的网络要求很少，但它提供的只能是尽力而为的服务。

5.5.3 隧道

处理两个不同网络相互连接时的一般情况超级困难。然而，却存在一种最常见的甚至可管理不同网络协议的情形。这种情形就是源主机和目标主机所在网络的类型完全相同，但它们中间却隔着一个不同类型的网络。举例来说，请考虑一家跨国银行，它在巴黎有一个IPv6网，在伦敦也有一个IPv6网，但是连接巴黎和伦敦办事处的却是IPv4 Internet，图5-40显示了这样的情形。

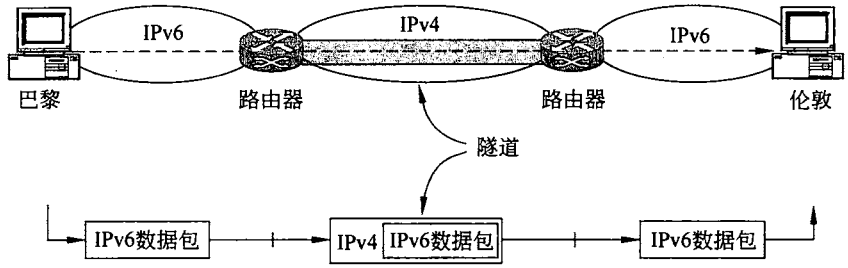


图5-40 从巴黎隧道一个数据包到伦敦

这个问题的解决方案是一种称为**隧道**（tunneling）的技术。为了给伦敦办事处的主机发送一个IP数据包，巴黎的主机构造一个包含伦敦IPv6地址的数据包，然后将该数据包发送到连接巴黎IPv6网络到IPv4 Internet上的多协议路由器；当该多协议路由器获得IPv6

数据包后，它把该数据包用一个 IPv4 头封装，封装后的 IPv4 数据包指向多协议路由器另一边的 IPv4，该网络与伦敦的 IPv6 网络相连；也就是说，路由器将一个（IPv6）数据包放入到一个（IPv4）数据包中。当这个包裹着的数据包到达伦敦路由器，原来的 IPv6 数据包被提取出来，并被发送给最终的目标主机。

可以把通过 IPv4 Internet 的路径看作是一根从一个多协议路由器延伸到另一个多协议路由器的大隧道。IPv6 数据包只是从隧道的一端旅行到隧道的另一端，封装在漂亮的盒子里非常舒适。它全然不必担心与 IPv4 的相处事宜。巴黎和伦敦的主机也不需要担心任何有关 IPv4 的事情。只有多协议路由器必须了解 IPv4 和 IPv6 数据包。实际上，从一个多协议路由器到另一个多协议路由器的整个行程就像单条链路上的一跳。

用一个类比可能会使隧道的概念更加清晰。考虑一个人驾着一辆汽车从巴黎出发要去伦敦。在法国境内，该汽车可以依靠自己的马力向前行驶，但是当它到达英吉利海峡，它被装到高速列车中，经过海底铁路隧道到达英国（汽车不允许直接在隧道中行驶）。实际上，这里的汽车被当作货物一样运到另一端，如图 5-41 所示。到了隧道另一端，汽车被放在英国公路上，它又可以依靠自己的马力向前行驶了。数据包在通过外部网络时所使用的隧道技术也是以同样的方式工作。

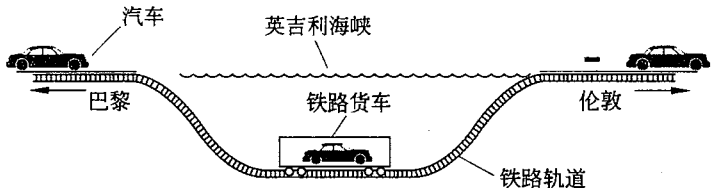


图 5-41 通过隧道把一辆车从法国运到英国

隧道被广泛用于连接那些因使用其他网络而被隔离的主机和网络。结果生成的网络就是所谓的**覆盖**（overlay）网络，因为它有效地覆盖在基础网络之上。部署一个具有新特性的网络协议是采用隧道的一个共同原因，就像我们说明的“IPv4 之上的 IPv6”例子。隧道的缺点是无法到达位于隧道之下网络的主机，因为数据包无法从隧道中间逃生。

然而，隧道的这个限制变成了虚拟专用网络（VPN，Virtual Private Networks）的优势。VPN 就是一个提供安全措施的简单覆盖网络。我们将在第 8 章探讨 VPN。

5.5.4 互联网路由

通过互联网络的路由所面临的基本问题与单个网络中路由的基本问题相同，但复杂性有所增加。首先，内部网络可能使用不同的路由算法。例如，一个网络可以使用链接状态路由，而另一个网络使用了距离矢量路由。由于链路状态算法需要知道拓扑但距离矢量算法不需要，仅仅这个差异就导致两个网络都不清楚如何在互联的网络上找到最短路径。

网络由不同运营商运行带来了更大的问题。首先，运营商对于什么是通过网络的好路径有不同的想法。一个运营商可能希望用最少延迟的路由，而另一个可能要最便宜的路由。这将导致运营商在设置最短路径成本时使用不同的度量（例如，毫秒计的延迟与货币计的成本）。这些网络上的权重没有可比性，因此互联网上的最短路径将得不到明确的定义。

更糟糕的是，一个运营商甚至可能不希望另一个运营商了解它网络的路径细节，因为

权重和路径多少反映出一些敏感信息（比如货币成本），这些信息代表了一种商业竞争优势。

最后，互联网可能比构成它的任何一个网络都大。因此，它或许需要采用层次结构的扩展性较好的路由算法，即使没有一个网络需要使用层次结构路由。

所有这些因素导致了两级路由算法。在每个网络中，使用一个**域内**（intradomain）或者**内部网关协议**（interior gateway protocol）进行路由（“网关”是“路由器”的旧称）。这可能是我们已经描述过的一种链路状态协议。为了让数据包跨越构成互联网的网络，就需要用到**域间**（interdomain）或**外部网关协议**（exterior gateway protocol）。网络可能全部使用不同的域内协议，但它们必须使用相同的域间协议。在 Internet 上，域间路由协议称为**边界网关协议**（BGP，Border Gaterway Protocol）。我们将在下节描述它。

还有一个更重要的术语要介绍。由于每个网络独立于所有其他网络运营，因此这样的网络通常称为一个**自治系统**（AS，Autonomous Sysem）。AS 的良好默认模型是 ISP 网络。事实上，一个 ISP 网络或许由多个 AS 组成，如果它管理或收购了多个网络。但两者之间的差异通常不显著。

这两级路由通常没有严格的层次。如果一个庞大的国际网络和一个较小的区域网络都被抽象成一个单一网络，可能导致极不优化的路径。然而，暴露出来的有关网络内部路由信息相对太少，以至于无法找到互联网络上的路由。但这有助于解决所有的复杂性，改善网络尺度，并允许运营商使用自选的路由协议来自由地选择网络内部的路由，而且不需要比较不同网络的权重或者将敏感信息暴露在网络以外。

然而，到目前为止我们很少提到有关如何在组成互联网的这些网络之间路由。在 Internet 上，一大决定因素是 ISP 之间的商业安排。每个 ISP 可能因替其他 ISP 承载流量而收取或接收相应的费用。另一个因素在于，如果国际互联网络的路由需要跨越国界，各国的法律可能会突然开始发挥作用，比如瑞典严格的隐私法律条款禁止出口有关瑞典公民的个人资料。所有这些非技术因素都包裹在一个路由策略（routing policy）概念中，控制着自治网络自主选择所用的路由。我们将在描述 BGP 协议时返回到路由策略这个主题上。

5.5.5 数据包分段

每个网络或链路都会限制其数据包的最大长度。这种限制来自多方面的原因，其中包括：

（1）硬件（比如以太网帧的长度限制）。

（2）操作系统（比如所有的缓冲区都是 512 字节）。

（3）协议（比如，数据包长度字段中的位数）。

（4）遵从某个国家（或国际）标准。

（5）期望将错误引入的重传次数减少到某种程度。

（6）期望防止数据包占用信道时间太长。

所有这些因素导致的结果是网络设计者们无法自由地选择任何他们所期望的最大数据包长度。一些常用技术的最大有效载荷长度：以太网为 1500 字节，802.11 协议为 2272 字节，IP 协议更通用一些，允许数据包的长度最多可达 65 515 字节。

主机一般倾向于传输大的数据包，因为这样可以降低开销，比如浪费在头字节上的带宽。当一个大数据包要穿过一个最大数据包尺寸太小的网络时，一个明显的网络互联问题

就出现了。这个滋扰一直是网络互联的老大难问题，随着从 Internet 上获得的丰富经验，相应的解决方案得到了很大发展。

一种解决方案是首先确保不会发生这个问题。然而，这说起来容易做起来难。源端通常不知道数据包通过网络到达接收方的路径，因此它当然不知道到达接收方的数据包尺寸有多小。这个数据包尺寸称为**路径最大传输单元**（MTU，Path Maximum Transmission Unit）。即使源端知道路径 MTU，数据包在无连接网络中也是独立路由的，比如 Internet。这种路由意味着路径可能会突然改变，因而意外地更改路径 MTU。

另一种解决办法是允许路由器将数据包拆分成段，将每个段作为一个独立的网络层数据包发送。不过，正如每个小孩父母都知道，将一个大物体转换成小的碎片远比相反的过程更容易（物理学家甚至给这个效果起了个名称：热力学第二定律）。数据包交换网络把段重新整合起来也同样麻烦多多。

将分段重新组成原始的数据包，可以采用两种对立的策略。第一种策略，由“小数据包”网络引起的分段过程对于沿途后续的网络都是透明的，也就是说，从该网络一直到最终的目标途中的每个网络都感觉不到曾经发生过分段，如图 5-42（a）所示。在这种方法中，当一个超大数据包到达 G_1，该路由器将它分割成多个段；每个段都发向同样的出口路由器 G_2，在这里这些段被重新组合起来。按照这种方法，任何一个数据包通过这样的小数据包网络都是透明的。后续的网络根本感觉不到曾经发生过分段。

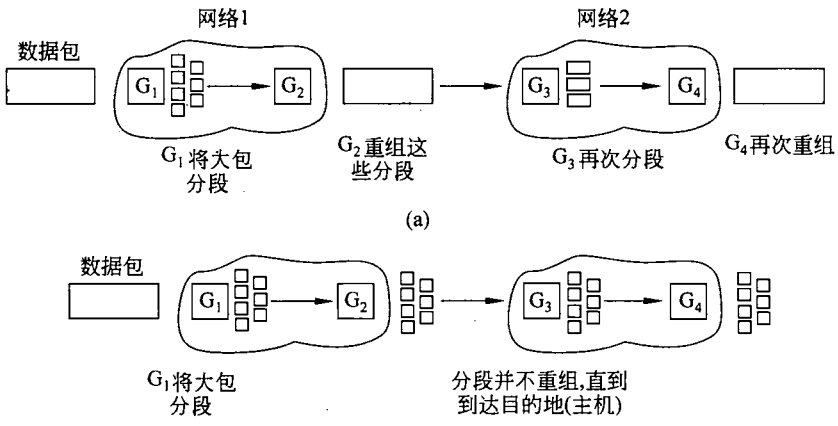


图 5-42
（a）透明分段；（b）非透明分段

透明的分段过程非常直接、简单，但是也有一些问题。首先，出口路由器必须知道什么时候它已经接收到了全部的段，所以每个分段中必须提供一个计数字段或者一个“数据包结束”标志位。其次，由于所有的数据包必须经过同一个出口路由器才能进行重组，因此路由受到了限制。由于不允许有些段沿着一条路径到达最终目标，而另一些段沿着一条不相交路径到达最终目标，所以，可能会损失一些性能。更为重要的是，路由器可能不得不做大量的工作。如果不是所有的段都已到达，它还需要缓冲到达的段，并且决定何时丢弃这些段。某些工作可能纯粹是一种浪费，因为当一个数据包需要通过一系列的小数据包网络时，需要多次被分段和重组。

另一种分段策略是避免在任何一个中间路由器上重新组合分段。一旦一个数据包已经

被分段，则每个段都被当作原始的数据包一样来对待。路由器传递这些段的情形如图 5-42（b）所示，重组过程只在目标主机上进行。

非透明分段的主要优点是路由器所做的工作比较少。IP 就是以这种方式工作的。一个完整的设计要求分段以可以重新构建原有数据流的方式编号。IP 采用的设计思想是：给每个段一个数据包序号（所有的数据包都携带）、一个数据包内的绝对字节偏移量和一个指明是否到达数据包末尾的标志位。图 5-43 给出了一个例子。虽然这种设计简单，但有一些吸引人的特性。段到达目的地后可以被放置在一个缓冲区中以便重组，即使这些分段到达的秩序凌乱不堪；当段要穿过一个 MTU 更小的网络时，还可以被路由器再次进行分段，如图 5-43（c）显示的那样；数据包的重传（如果所有的段都没有收到）可以被分割成不同的段。最后，段可以任意大小，最小的段是一个字节加上数据包头。在所有情况下，接收方只需简单地使用数据包的序号和段偏移量即可把数据放置在合适的位置，并利用数据包结束标志位来确定何时有了一个完整的数据包。

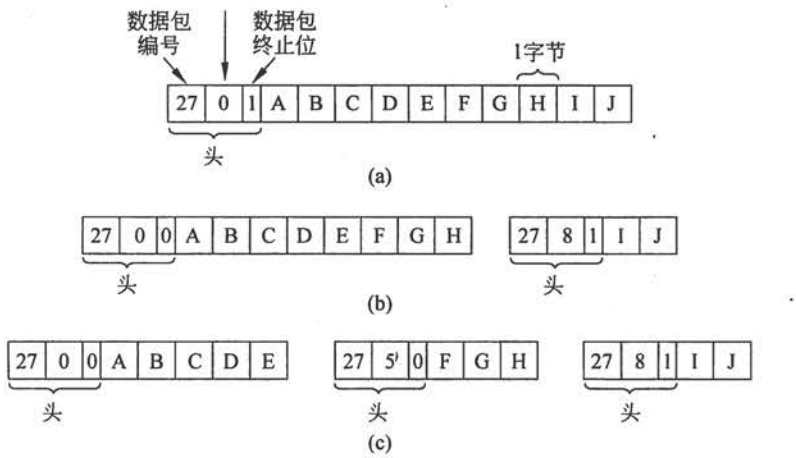


图 5-43 基本数据大小是 1 个字节时的分段
（a）原始数据包，包含 10 个字节的数据；（b）经过一个最大包尺寸为 8 的网络后的分段加上头；（c）经过一个大小为 5 的网关后的分段

不幸的是，这样的设计还是有问题。因为现在某些链路上运载的段的头或许是没必要的。但真正的问题首先还是因为段的存在，因此开销可能比透明分段高。（Kent 和 Motul，1987）认为分段不利于性能，因为除了增加头开销，数据包的丢失概率也增加了；任何一个段的丢失都将导致整个数据包的丢失；而且对主机而言，分比不分带来了更大的突发。

这又导致我们回到了最初的解决方案，就是在网络中避免分段操作，这种策略被现代 Internet 所采用。这个过程称为**路径 MTU 发现**（path MTU discovery）（Mogul 和 Deering，1990）。它的工作原理如下所示。每个 IP 数据包发出时在它的头设置一个比特，指示不允许对该数据包实施分段操作。如果一个路由器接收的数据包太大，它就生成一个报错数据包并发送给源端，然后丢弃该数据包，如图 5-44 所示。当源端收到报错数据包，它就使用报错数据包携带的信息重新将出错数据包分段，每个段足够小到报错路由器能处理。如果沿着路径前进又遇到一个 MTU 更小的路由器，那么重复上述过程。

路径 MTU 发现的优点是源端现在知道应该发送多长的数据包了。如果路由和路径 MTU 发生改变，新的报错数据包将被触发返回，因而源端将适应新的路径。然而，仍然需

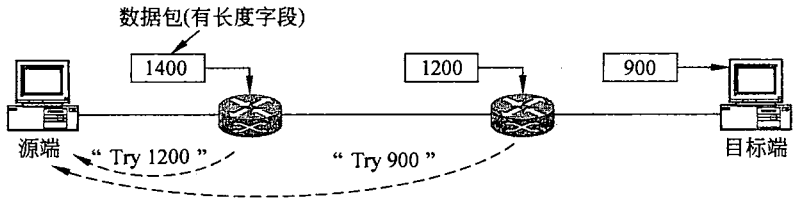


图 5-44 路径 MTU 的发现过程

要在源端和接收方之间实施分段，除非更高层次的协议了解路径 MTU，并把适量的数据传给 IP。TCP 和 IP 通常是一起实现的（正如 TCP/IP 称呼），因而能够传递这类信息。即使其他协议不是这样实现，仍然可以把分段移出网络，让主机承担。

路径 MTU 发现方法的缺点是有可能增加发送数据包的启动延迟。在任何数据被传递到目的地之前，探讨路径并且找出该路径 MTU 所花的时间可能需要不止一个往返延迟。这就引出了一个问题，是否有更好的设计？答案很可能“是的”。考虑这样的设计：每个路由器把那些超出其 MTU 的数据包简单地拦腰截断。这样一来将确保接收方尽可能快地了解 MTU（从传递过来的数据量），同时接收了部分数据。

5.6 Internet 的网络层

现在是时候详细讨论 Internet 的网络层了。但在进入 Internet 细节之前，值得先看看当初驱动其设计并导致其今天成功的原则。现在，很多时候人们似乎已经忘记了这些原则。RFC 1958 列举了这些原则，并对它们进行了讨论，该文档很是值得一读（这是所有协议设计者的必修课——最后应该有个期末考试）。这份 RFC 文档着重描述了由（Clark，1988）和（Saltzer 等，1984）提出的想法。现在，我们总结我们所认为的 10 大原则（从最重要的到最不重要的）。

（1）**保证工作**。直到多个原型系统成功地与对方相互通信，方可完成设计或者确定标准。设计者常常首先写出一个 1000 页的标准，并获得批准，过后才发现存在严重的缺陷，根本无法工作。然后他们再编写一个 1.1 版本的标准。这不是正确的工作方式。

（2）**保持简单**。有疑问时应该使用最简单的解决方案。William of Occam 在 14 世纪就提出了这条原则（称为奥卡姆的剃刀）。换成现代术语就是：决斗特性。如果一项特性并非绝对不要，那么就放弃该特性。尤其是，通过组合其他的特性也能够获得同样效果的时候。

（3）**明确选择**。如果有几种方法可以完成同样的事情，则选择其中一种方法。用两种或者多种方法来做同样的事情简直是自找麻烦。通常标准会有多个选项、多种模式或多个参数，因为多个实力强大的参与方坚持认为他们的方法是最好的。设计者应该坚决抵制这种倾向，学会说“不”。

（4）**模块开发**。这条原则直接导致了协议栈的思想，每一层的协议完全独立于所有其他的协议。按照这种方法，如果实际环境中要求改变一个模块或者一层，则其他模块或层都不会受到影响。

（5）**期望异构性**。在任何一个大型的网络中，可能存在不同类型的硬件、传输设施和

应用程序。为了处理它们，网络的设计必须简单、通用和灵活。

（6）**避免静态选项和参数**。如果不可避免要使用参数的话（比如最大数据包长度），那么，最好的办法是让发送方和接收方协商一个值，而不是定义固定的参数值。

（7）**寻找好的而不是完美的设计**。通常设计者有一个好的设计，但是它不能够处理一些怪异的特殊情况。设计者不应该乱改设计，而是坚持这个好的设计，将围绕着特殊情况而展开的工作负担转移到那些强烈需求的人身上。

（8）**严格发送，宽容接收**。换句话说，只发送那些严格符合标准的数据包，但是，容许接收那些不完全符合标准的数据包，并且试图对它们进行处理。

（9）**考虑可扩展性**。如果系统需要有效地处理上百万台主机和几十亿用户，那么，没有一种中心数据库是可以容忍的，同时必须将负载尽可能均匀地分布到所有可用的资源上。

（10）**考虑性能和成本**。如果一个网络的性能很差，或者成本很高，那么没有人会使用它。

现在，让我们离开通用设计原则，开始 Internet 网络的探索之旅。在网络层，可以把 Internet 看作是一种相互关联的网络或自治域（自治系统）集合。没有真正的结构，但存在几个主要骨干网。这些都是由高带宽线路和快速路由器组成。这些骨干网中最大的一个称为**一级网络**（Tier 1 networks），每个骨干网都与它连接，进而到达其他骨干网。连接到骨干网上的是 **Internet 服务提供商**（ISP，Internet Service Provider），它为家庭和企业、数据中心和服务器托管设施，以及区域（中级）网络提供 Internet 接入服务。数据中心提供了许多通过 Internet 发送的内容。连接到区域网络的是更多的 ISP、许多大学和公司的局域网和其他边缘网络。图 5-45 给出了一个准分层组织的轮廓图。

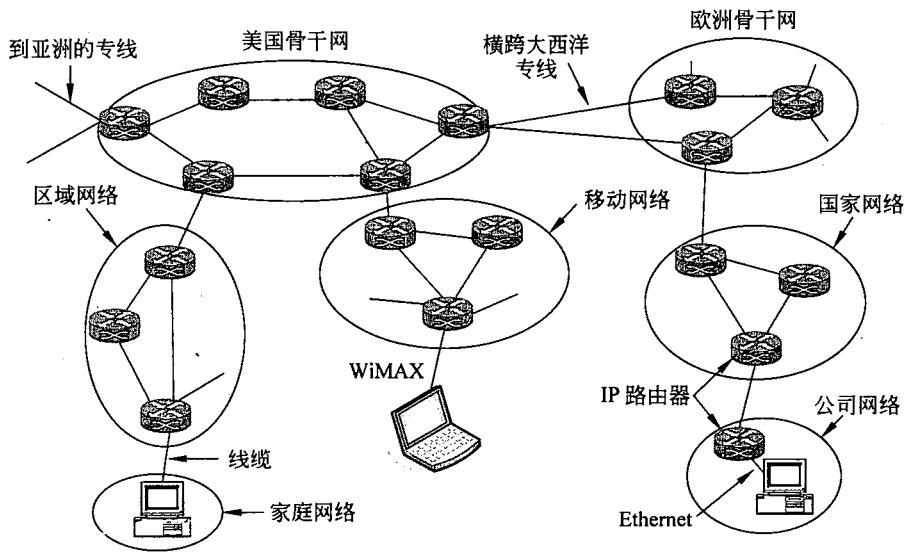


图 5-45　Internet 是互相连接的许多网络

将整个 Internet 黏合在一起的正是网络层协议，即 **Internet 协议**（IP，Internet Protocol）。与大多数老式网络层协议不同的是，IP 协议在设计之初就把网络互联作为目标。应该这样看待网络层：IP 的任务是提供一种尽力而为（best-effort）地把数据包从源端传输到接收方的方法（即不提供任何保证），无须考虑这些机器是否在同一个网络，也不必关心它们之间

是否还有其他网络。

Internet 中的通信过程是这样的。传输层获取数据流，并且将数据流拆分成段，以便作为 IP 数据包发送。理论上，每个数据包最多可容纳 64 KB，但实际上，数据包通常不超过 1500 个字节（因而它们正好可被放到一个以太网帧中）。IP 路由器转发每个数据包穿过 Internet，沿着一条路径把数据包从一个路由器转发到下一个路由器，直到数据包到达目的地。在接收方，网络层将数据交给传输层，再由传输层交给接收进程。当所有的数据段最终都抵达目标机器，它们被网络层重新组装还原成最初的数据报；然后该数据报被网络层传给传输层。

在图 5-45 所示的例子中，家庭网络上的一个主机要穿越四个网络和大量 IP 路由器，才能到达位于公司网络上的目标主机。这种情况在实际中并不罕见，而且还有很多更长的路径。在 Internet 上存在着很多冗余连接，骨干网和 ISP 在多个位置相互连接。这意味着两个主机之间存在着许多可能的路径。决定使用哪些路径正是 IP 路由协议的任务。

5.6.1 IPv4 协议

开始学习 Internet 网络层最恰当的方式是从 IP 数据报本身的格式开始。每个 IP 数据报包含两部分，一个头和一个正文，正文部分也称之为有效净荷。头由一个 20 字节的定长部分和一个可选的变长部分组成。图 5-46 显示了 IP 数据报的头格式。IP 数据报头的传输从左到右并从上到下，Version 字段的高序字节最先被传送出去（这就是“big-endian”网络字节序。在 little-endian 字节序机器上，比如 Intel x86 计算机，在传输和接收时需要进行字节顺序的软件转换）。现在回想起来，little-endian 字节序是更好的选择，但在设计 IP 协议时没有人能预测到今天 little-endian 流行计算世界。

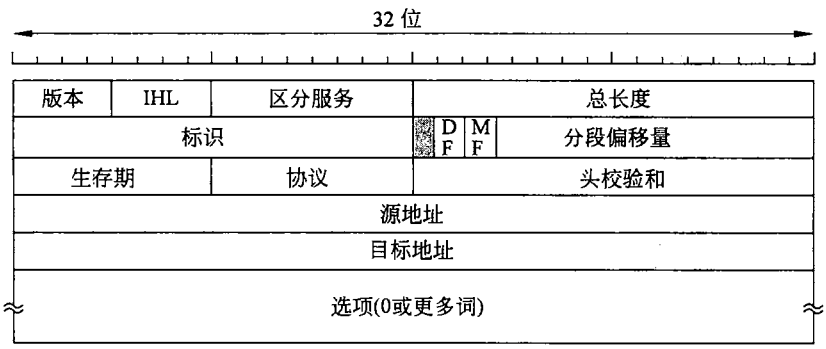


图 5-46 IPv4 （Internet 协议）头

版本（Version）字段记录了数据报属于协议哪个版本。版本 4 主宰着今天的 Internet，这也就是为什么我们从这里开始学习的真正原因。在每个数据报开始包含版本信息，使得版本之间的迁移过程可以持续很长一段时间。事实上，IPv6，即 IP 协议的下一个版本已经被定义了十多年，但它的部署却还刚刚开始不久。当某个国家 2^{31} 人口中每个人都拥有一个台式 PC、一台笔记本电脑或者一个 IP 电话时，最终将不得不使用 IPv6。我们将在本节后面介绍 IPv6。关于版本编号，这里顺便提一下，IPv5 是一个试验性的实时流协议，它一直没有被广泛应用。

由于头的长度不固定，所以头的 IHL 字段指明了头到底有多长（以 32 位字长度为单位）。IHL 的最小值为 5，这表明头没有可选项。该 4 位字段的最大值为 15，把头的长度限制为最大 60 字节，因此选项（Options）字段最多为 40 字节。对于某些选项，比如记录一个数据包路径的选项，40 字节往往太小，这使得这样的选项其实没有什么用处。

区分服务（Differentiated services）字段是少数几个在意义上随岁月（轻微）改变的字段之一。该字段最初称为服务类型（Type of service）。它曾经并且仍然用来区分不同的服务种类。可靠性和速度的各种组合都是可能的选择。对于数字化的话音数据，加速传递优先于精确传递；对于文件传输，正确传输比加速传输更加重要。最初时，服务类型字段包含 6 位，其中 3 位表示优先级，3 位代表主机最关心的是延迟、吞吐量或可靠性中的哪一个。然而，没人真正知道路由器用这些位做什么，因此这些位空着许多年没人用。在设计区分服务时，IETF 承认自己的失败，并重新启用这个字段。现在，前 6 位用来标记数据包的服务类别，我们在本章前面描述过的加速服务和确保服务；后 2 位用来携带显式拥塞通知信息，比如数据包是否经历了拥塞，我们在本章的拥塞控制部分描述了显式拥塞通知。

总长度（Total length）字段包含了该数据报中的所有内容，即头和数据。最大长度是 65 535 个字节。目前情况下，这样的上界还是可以容忍的，但在未来网络中，可能需要更大的数据报。

标识（Identification）字段的用途是让目标主机确定一个新到达的分段属于哪一个数据报。同一个数据报的所有段包含同样的标识值。

接下来是一个未使用的位，这很令人惊讶，因为 IP 头中可供使用的地方实在是很珍贵。作为愚人节开的一个玩笑，（Bellovin，2003）提出用该位来检测恶意流量。因为已知带有该“邪恶”位的数据包是由攻击者发送，因此路由器可把它丢弃，这样可以大大简化安全控制。不幸的是，网络安全并不是这么简单。

接下来的两个 1 位字段与分段有关。DF 代表“不分段”（Don't Fragment）标志位。这是针对路由器的一条命令，它不允许路由器分割该数据报。最初，该字段用来支持没有能力组装还原数据包的主机。现在该字段可用在发现路径 MTU 过程中，路经 MTU 是能经过路经而无须分段的最大数据包。通过在发出的数据包中设置 DF 位，发送方知道这个数据包要么完整地到达目的地，要么有个报错消息反馈回来。

MF 代表“更多的段”（More Fragments）标志位。除了最后一个段以外，其他所有的段都必须设置这一位。它的用途是接收方可以知道什么时候一个数据报的所有分段都已经到达了。

分段偏移量（Fragment offset）字段指明了该段在当前数据报中的位置。除了数据报的最后一个段以外，其他所有段的长度必须是 8 字节的倍数。由于该字段有 13 位，所以每个数据报最多有 8192 个段，由此支持 Total length 字段限制的最大数据报。Identification、MF 和 Fragment offset 这 3 个字段协同工作，可用来实现 5.5.5 节描述的分段操作。

生存期（Time to live）字段是一个用于限制数据包生存期的计数器。这里的计数单位最初设置为秒，因此最大的生存期为 255 秒。在每一跳上该计数器必须被递减，而且，当数据报在一台路由器上排队时间较长时，该计数器必须多倍递减。实际上，它只是跳计数器，当它递减到 0 时，数据包就被丢弃，并且路由器给数据包的源主机发回一个报警包。此项特性可以避免数据包永远逗留在网络中，有时候当路由表被破坏之后可能会发生这样

的事情。

当网络层组装完成一个完整的数据包之后，它需要知道该如何对它进行处理。协议（Protocol）字段指明了该将它交给哪个传输进程。TCP 是一种可能，但是 UDP 或者其他的协议也有可能。协议的编号在整个 Internet 是全球统一的。RFC 1700 中列出了以前的协议和其他分配的编号，现在的协议编号包含在一个位于 www.iana.org 的在线数据库中。

由于头携带了诸如地址那样致命的信息，因此它用自己的校验和加以保护，即头校验和（Header checksum）字段。校验算法的执行过程是这样的：当数据到达时，所有的 16 位（半字）累加起来，然后再取结果的补码。该算法的目的是到达数据包的头校验和计算结果应该为 0。这样的校验和对于检测数据包穿过网络时是否发生错误非常有用。请注意，在每一跳必须重新计算头校验和字段，因为至少有个字段总是不断在改变（即生存期字段），但是，采用一些技巧可以加速计算。

源地址（Source address）字段和目标地址（Destination address）字段表示源网络接口和目标网络接口的 IP 地址。我们将在下一小节介绍 Internet 地址。

选项（Options）字段的设计意图是提供一种途径，允许后续版本协议包含一些原设计中没有出现的信息，以便实验人员尝试新的想法、避免为那些不常使用的信息分配头字段。选项具有可变长长度。每个选项的第一个字节是一个标识码，它标明了该选项类别。有的选项后面跟着一个 1 字节的选项长度字段，然后是一个或多个数据字节。选项字段用来将整个选项长度填充到 4 字节的倍数。最初设计时定义了 5 个选项，如图 5-47 所示。

选项	描述
安全性	标明数据报的安全级别
严格源路由	给出数据报遵循的完整路径
松散源路由	给出一些不能错过的路由器
记录路由	要求每个路由器加上自己的IP地址
时间戳	要求每个路由器加上自己的IP地址和时间戳

图 5-47 某些 IP 选项

安全（Security）选项指明了信息的秘密程度。理论上，军用路由器可能使用这个字段来指定路由时，不允许通过某些在军事上被认为是“敌对”的国家。实际上，所有的路由器都忽略该选项，所以，它仅有的实际用途是帮助间谍们更加容易找到好的材料。

严格源路由（Strict source routing）选项给出了从源到目标的完整路径，其形式是一系列 IP 地址。数据报必须严格地遵循这条路径向前传输。对于系统管理员，这个选项非常有用，他们可以在路由表被破坏时发送紧急数据包，或者用这个选项来测量时间。

松散源路由（Loose source routing）选项要求该数据包穿越所指定的路由器列表，并且要求按照列表中的顺序前进；但是，在途中也允许经过其他路由器。通常情况下，该选项往往只提供少数几个路由器，用来强迫数据包走一条特殊的路径。例如，为了强迫一个从伦敦到悉尼的数据包必须向西走而不是向东走，该选项可以指定纽约、洛杉矶和檀香山的路由器作为路由必经节点。当出于政治或者经济的考虑而要求经过或者避开某些国家的时候，这个选项最有用。

记录路由（Record route）选项告诉沿途的路由器，将自己的 IP 地址附加到可送字段中。

这样系统管理员就可以跟踪路由算法中的错误（比如，“为什么从休斯敦到达拉斯的数据包要经过东京？”）。当ARPANET刚开始建立时，没有一个数据包会经过9个以上的路由器，所以40字节的选项足够了。正如前面所提到的，现在40字节显得太小了。

最后，时间戳（Timestamp）选项类似于记录路由选项，只不过每个路由器除了记录自己的32位IP地址以外，还要记录一个32位时间戳。对于网络测量，这个选项也是最有用的。

如今，IP选项已失宠。许多路由器忽略它们或者不能有效地处理它们，它们作为一种罕见案例被搁置了起来。也就是说，它们只是得到部分支持，但很少被使用。

5.6.2 IP地址

IPv4的一个明确特征是它的32位地址。Internet上的每台主机和每个路由器都有一个IP地址，可用在IP数据包的Source address和Destination address字段。重要的是要注意，一个IP地址并不真正指向一台主机，而是指向一个网络接口，所以如果一台主机在两个网络上，它必须有两个IP地址。然而，大多数主机都连在一个网络，因而只有一个IP地址。与此相反，路由器有多个接口，从而有多个IP地址。

前缀

与以太网地址不同的是IP地址具有层次性。每个32位地址由高位的可变长网络和低位的主机两部分数据组成。同一网络上（比如以太局域网）的所有主机，其地址的网络值是相同的。这意味着一个网络对应一块连续的IP地址空间，这块地址空间就称为地址的**前缀**（prefix）。

IP地址的书写方式是点分十进制表示法。按此格式，4个字节中的每个写成十进制，取值范围从0到255。例如，32位十六进制地址80D00297写成128.208.2.151。前缀的书写给出了块的最低IP地址和块的大小。块大小由网络部分的位数决定；地址中主机部分的剩余位数可以有所变化。这意味着，块的大小必须是2的幂。按照惯例，网络地址的书写格式是前缀IP地址后跟一个斜线，斜线后是网络部分的位长度。在我们的例子中，如果前缀包含2^8个地址，所以留下了24位用于网络部分，写成128.208.0.0/24。

因为前缀长度仅从IP地址无法推断出来，路由协议必须把前缀携带给路由器。有时候，前缀很简单地由长度描述，比如“/16”，读音为“slash 16”。前缀长度相当于网络部分中1的二进制掩码。以这种格式书写时称为**子网掩码**（subnet mask），它可以与一个IP地址进行AND操作，以便提取出该IP地址的网络部分。在我们的例子中，子网掩码为255.255.255.0。图5-48显示了一个前缀和一个子网掩码。

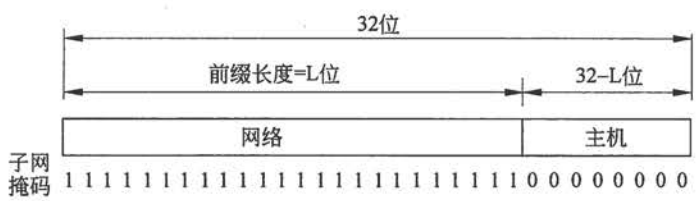


图5-48 IP的前缀和子网掩码

层次化的地址具有显著的优点和缺点。前缀的主要优势是路由器仅仅根据地址的网络部分即可转发数据包，只要每个网络都有一个唯一的地址块。对路由器来说网络地址的主机部分并不重要，因为同一网络上的所有主机都在同一个方向，只有当数据包到达它们的目的地网络后才被转发到正确的主机。这样可使得路由表远远小于按每个 IP 地址索引所需要的大小，考虑到 Internet 的主机数量已接近十亿，这对于每个路由器需要维护的路由表来说实在是太大了。然而，通过分层的方式路由器只需要保持约 30 万条前缀的路由。

虽然使用层次路由使得 Internet 路由规模化，但它有两个缺点。首先，一个主机的 IP 地址取决于它位于网络上的位置。以太网地址可用于世界上的任何地方，但每个 IP 地址属于一个特定的网络，路由器只能传递注定要到该网络上某个地址的数据包。比如移动 IP 这样的设计必须支持主机在网络之间移动，但同时保持相同的 IP 地址。

第二个缺点在于层次结构浪费了地址，除非精心管理地址空间。如果给网络分配（太）大块的地址，将有（很多）被分配掉但不会使用。这种分配将没有多大意义，如果有大量的地址空间，这倒没什么关系。然而，二十年以前人们就认识到 Internet 的巨大增长正迅速消耗着自由地址空间。IPv6 就是地址短缺问题的解决方案，但直到它被广泛部署，否则分配 IP 地址都将面临巨大的压力，所以必须有效地使用地址。

子网

为了避免冲突，网络地址的管理由一个称为 **Internet 域名和地址分配机构**（ICANN，Internet Corporation for Assigned Names and Number）的非营利性公司负责。ICANN 一次把部分地址空间授权给各区域机构，这些机构再把 IP 地址发放给 ISP 和其他公司。这就是一家公司获得一块 IP 地址的过程。

然而，这个过程仅仅是故事的开始，因为 IP 地址的分配随着企业的成长也在不断地消耗着。我们曾经说过，按前缀路由需要网络中的所有主机具有相同的网络号。随着网络的发展这个属性可能会产生问题。例如，我们用一所大学开始举例说明有关地址问题。/16 分配给计算机科学系，供它以太网上的所有计算机使用；一年后，电机工程学系想上 Internet，不久艺术系紧跟其后。这些系应该用什么 IP 地址？另外申请一个地址块不仅远离学校，而且可能既昂贵又不方便。此外，早已经分配获得的/16 足够 60 000 台主机的地址。这可能是为了将来有发展空间才申请的，但至少现在还没到那个规模，为同一所大学再分配另一块地址显然是一种浪费。因此，需要一种不同的地址组织。

问题的解决方案是，在内部将一个网络块分成几个部分供多个内部网络使用，但对外部世界仍然像单个网络一样。这就是所谓的**子网划分**（subnetting），分割一个大型网络得到的一系列结果网络（比如以太网）称为**子网**（subnet）。正如我们在第 1 章中提到的那样，你应该意识到这个词的新用法和旧的用法有冲突，子网的以前含义是指网络中的所有路由器和通信线路的集合。

图 5-49 显示了子网如何帮助解决我们例子中的问题。一个/16 地址空间被分割成几片。这种分割并不要求均匀，但每片必须对齐以便可以把较低的任何位用作地址的主机部分。在这种情况下，块的一半（一个/17）分配给了计算机科学系，四分之一分配给了电机工程学系（一个/18），八分之一（一个/19）分配给了艺术系，剩余的八分之一未分配。了解地址块如何分割的不同方式是看以二进制表示的结果前缀：

```
计算机科学系:  10000000  11010000  1|xxxxxxx xxxxxxxx
电子工程学系:  10000000  11010000  00|xxxxxx xxxxxxxx
艺术系:        10000000  11010000  011|xxxxx xxxxxxxx
```

这里的竖线（|）表示子网号和主机部分的边界。

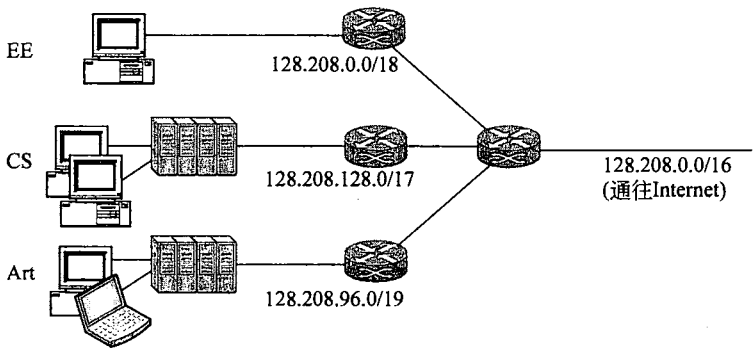


图 5-49 将 IP 前缀进一步细分为网络和子网

当一个 IP 数据包到达主路由器，路由器如何知道应该将它转发到哪个子网？这必须进入到深入了解前缀的细节。一种方法是每个路由器有一张具有 65 536 个项的表，该表告诉它校园网络上每台主机对应的输出线路是哪条。但是，这样会破坏掉使用层次路由带来的可扩展性。相反，路由器只需简单了解校园网络的子网掩码。

当数据包到达时，路由器会查看该数据包的目标地址，并检查它属于哪个子网。具体做法是：路由器把数据包的目标地址与每个子网的掩码进行 AND 操作，看结果是否对应于某个前缀。例如，考虑一个发往 IP 地址 128.208.2.151 的数据包。为了看它是否属于计算机科学系，我们把该目标地址与 255.255.128.0 进行 AND 操作，获得前 17 位（即 128.208.0.0），并且检查它们是否匹配计算机系的前缀地址（即 128.208.128.0）。显然它们不匹配。然后再检查前 18 位，在与电机工程学系的子网掩码 AND 操作后，我们得到 128.208.0.0，这恰好匹配电机工程系的前缀地址，所以数据包被转发到通往电机工程系网络的接口。

如果有必要，子网的划分还可以改变，只需要更新校园网内部路由器上的所有子网掩码即可。在网络外面，子网的划分是不可见的，因此分配一个新的子网不需要联系 ICANN 或者改变任何外部数据库。

CIDR——无类域间路由

即使 IP 地址按块分配，因此地址空间得以有效地使用，但还有一个问题依然存在：路由表爆炸。

一个组织（比如大学）中位于网络边缘的路由器必须为每个自己的子网设立一个表项，该表项告诉路由器使用哪条线路到达该子网。对于通往组织外部某个目的地的路由，这些路由器可以利用简单的默认路由把数据包发送到通往 ISP 的线路上，ISP 把组织与 Internet 其余部分相连。其他目的地址必定在外面的某个地方。

ISP 和骨干路由器之间的路由没有这样奢侈，这些骨干路由器位于 Internet 中间。它们必须知道通过哪些方式可到达每个网络，这里没有简单的默认路由可用。这些核心路由器处在一个 Internet 默认自由区（default-free zone）。没有人真正知道有多少网络连接到

Internet，但它肯定是一个很庞大的数字，可能至少100万。这样的规模会产生一个巨大的路由表。用计算机标准来衡量这个数字并不大，但我们要意识到路由器转发每一个数据包时都必须查询这张表，而大型ISP的一个路由器可能每秒要转发数百万个数据包。必须用专门的硬件和快速内存才能以如此高的速率处理数据包，一般的通用计算机是无法胜任此项工作的。

此外，路由算法要求每个路由器与其他路由器交换有关它能到达的地址信息。表越大，需要通信和处理的信息量也越大，并且处理时间至少随表的大小呈线性增长。更多的通信增加了丢失部分路由表的可能性，至少有暂时丢失的可能，这又可能导致路由的不稳定。

路由表的问题本来可通过一个更深的层次来解决，类似电话网那样的多层结构。例如，让每个IP地址包含一个国家、州/省、市、网络和主机这些字段可能会奏效。然后，每个路由器只需要知道如何去每个国家、本国的州或省、本州或本省的城市以及本城的网络。不幸的是，这种解决方案需要比32位多得多的IP地址，而且地址的使用缺乏效率（并且，列支敦士登也必须像美国一样，使用许多位的地址）。

幸运的是，我们可以做一些事情来减小路由表的长度。我们可以运用与子网划分相同的观点：不同地点的路由器可以知道一个给定IP地址的不同大小前缀。然而，不是将一块地址分割成子网，相反，在这里我们把多个小前缀的地址块合并成一个大前缀的地址块。这个合并过程称为**路由聚合**（route aggregation），由此产生的较大前缀地址块有时称为**超网**（supernet），以便有别于地址块的分割。

有了地址聚合，IP地址可包含大小不等的前缀。同样一个IP地址，一台路由器把它当作/22的一部分对待（该地址块包含2^{10}个地址），而另一台路由器把它当作一个更大的/20一部分对待（其中包含2^{12}个地址）。这是因为每个路由器有相应的前缀信息。这个设计和子网划分协同工作，统称为**无类域间路由**（CIDR，Classless Inter-Domain Routing），发音为“cider”。它的最新版本由RFC 4632说明（Fuller和Li，2006年）。这个名字突出了与有类别的地址层次编码的不同，稍后我们将简要介绍这点。

为了使CIDR更加易于理解，让我们来考虑一个例子，有一块地址从194.24.0.0开始，可用的IP地址有8192个。假设剑桥大学需要2048个地址，它分配获得的地址范围从194.24.0.0到194.24.7.255，掩码为255.255.248.0，这是一个/21前缀。接下来，牛津大学申请4096个地址。由于4096个地址的块必须位于4096的字节边界上，所以，牛津大学申请的地址不可能从194.24.8.0开始。相反，它们获得了从194.24.16.0至194.24.31.255的地址块，子网掩码为255.255.240.0。最后，爱丁堡大学申请1024个地址，它获得了从194.24.8.0至194.24.11.255的地址块，掩码为255.255.252.0。图5-50概括了这些地址块的分配情况。

大学	第一个地址	最后一个地址	多少地址	前缀
剑桥	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
爱丁堡	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(可用)	194.24.12.0	194.24.15.255	1024	194.24.12.0/22
牛津	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

图5-50 一组分配的IP地址

在默认自由区的所有路由器现在都被告知这3个网络的IP地址。靠近大学的路由器针

对每个前缀可能需要发送到不同的出境线路，所以它们需要在自己的路由表中为每个前缀设立相应的表项。作为例子，图 5-51 给出了伦敦路由器示意图。

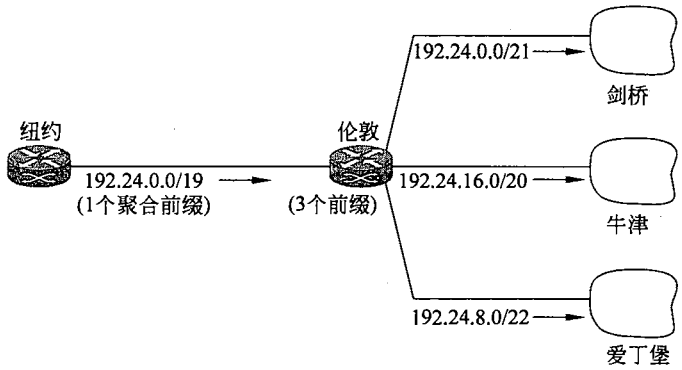


图 5-51 IP 前缀的聚合

现在让我们从位于纽约的一个远程路由器的角度来看这 3 所大学。3 个前缀的所有 IP 地址都应该从纽约（或更一般地，从美国）发送到伦敦。伦敦的路由进程注意到这个，并且将 3 个前缀合成一个聚合表项，即前缀 194.24.0.0/19，然后传递给纽约路由器。这个前缀包含了 8K 个地址，并涵盖了 3 所大学的地址和其他未分配的 1024 个地址。通过聚合，3 个前缀已经减少为一个，由此减少了必须告知纽约路由器的前缀，并且纽约路由器的路由表表项。

当聚合功能被启用后，这个过程完全自动。这依赖于给 Internet 分配什么样的前缀，而不是给网络分配地址的管理员行动。聚合技术在 Internet 中被大量使用，已经把路由器表的大小减少到目前的约 200 000 万前缀。

更为扭曲的是前缀允许重叠。规则是数据包按最具体路由的方向发送，即具有最少 IP 地址的**最长匹配前缀**（longest matching prefix ）。最长匹配前缀路由提供了有益的灵活性，正如从图 5-52 显示的纽约路由器的行为。该路由器仍使用单一聚合前缀把三所大学的流量发送到伦敦。然而，该前缀中的先前那块未用地址现在已经分配给了旧金山的网络。一种可能是纽约路由器保持四个前缀，其中三个前缀的数据包发送到伦敦，第四个前缀的数据包发送到旧金山。相反，最长匹配前缀路由可以用图中显示的两个前缀来转发。一个总的前缀指示把整个地址块的流量发到伦敦；一个更具体的前缀用来指示该大前缀的一部分流量发往旧金山。有了最长匹配前缀规则，到旧金山网络的 IP 地址的流量将被发送到通往旧金山的出境线路，并且发往大前缀中所有其他 IP 地址的流量将被送往伦敦。

图 5-52 纽约路由器中的最长匹配前缀路由

从概念上讲，CIDR 的工作原理如下所述。当一个数据包到达时，路由器扫描路由表以便确定目的地是否在前缀的地址块内。有可能多个具有不同前缀的表项得到匹配，在这

种情况下，使用具有最长前缀的表项。因此，如果有一个匹配/20 掩码的表项，同时还有一个匹配/24 掩码的表项，则使用/24 表项来查询数据包的出境线路。然而，如果真正一个表项一个表项地扫描表，这个过程将非常冗长乏味。相反，人们设计了复杂的算法来加快地址匹配过程（Ruiz-Sanchez 等，2001）。商用路由器使用了定制的 VLSI，这些算法被嵌入到了硬件中。

分类和特殊寻址

为了帮助你更好地理解为什么 CIDR 如此有用，我们简要介绍之前的地址设计方案。在 1993 年以前，IP 地址被分为图 5-53 列出的 5 个类别。这种分配已经称为**分类寻址**（classful addressing）。

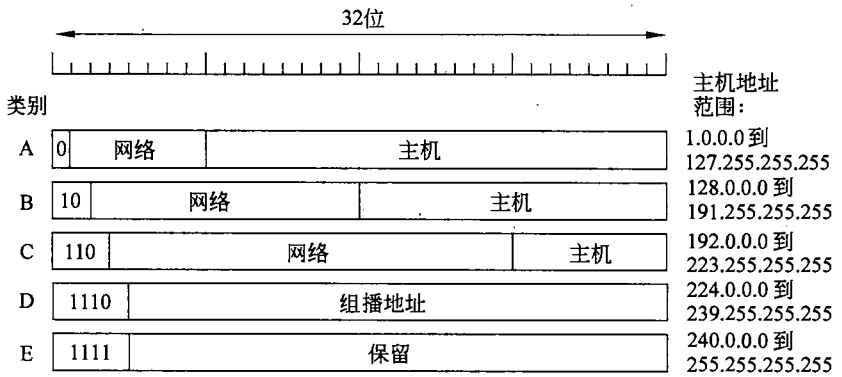


图 5-53 IP 地址格式

A、B、C 类地址格式分别允许多达 128 个网络，每个网络 1600 万台主机；16 384 个网络，每个网络 65 536 台主机；以及 200 万个网络（比如 LAN），每个网络多达 256 台主机（不过有些地址是特殊的）。IP 地址分类同时支持组播（D 类地址），即数据包被直接发送给多台主机。以 1111 开头的地址是保留地址，以备将来使用。在 IPv4 地址空间已经耗尽的情况下，使用它们将是有价值的。不幸的是，许多主机将不再把它们视为有效地址而接受，因为它们已经被禁锢了这么久，很难教会旧主机一些新花样。

这是一个层次化的设计，但与 CIDR 不同于，这里的地址块大小是固定的。超过 20 亿个地址，通过类别划分来组织地址空间会浪费数百万的地址。尤其是，真正的罪魁祸首是 B 类网络。对于大多数组织，具有 1600 万个地址的 A 类网络太大，而具有 256 个地址的 C 类网络又太小。一个具有 65 536 个地址的 B 类网络恰到好处。在 Internet 民间传说中，这种情况称为 3 只熊问题（犹如《Goldilocks and the Three Bears》（Southey，1848））。

在现实中，一个 B 类地址对大多数组织而言还是过于庞大。研究表明，超过半数的 B 类网络少于 50 台主机。一个 C 类网络足够应付得过来。但毫无疑问，每个组织都会申请一个 B 类地址，它们想到总有一天自己的网络规模将超过 8 位主机地址空间的范围。现在回想起来，给 C 类网络分配 10 位的主机号而不是 8 位主机号或许更好，因为这样的分配使得每个网络可以拥有 1022 台主机。如果真是这样，大多数组织可能用一个 C 类网络就能解决问题，而且将有差不多 50 万个 C 类网络（相对于 B 类网络只有 16 384 个）。

很难指责 Internet 的设计者没有提供更多（和更小）的 B 类地址。在当时决定创建这

三个类别的地址时，Internet 还只是一个连接美国主要研究型大学的研究网络（加上极少数公司和从事网络研究的军事基地）。没有人认为 Internet 将成为一个具有大众市场的通信系统，而且还能和电话网络相媲美。当时，人们毫无疑问地说："美国有大约 2000 所大专院校。即使它们都连接到 Internet，甚至其他国家的多所大学也加入进来，我们也永远达不到 16 000，因为整个世界没那么多的大学。而且，主机号是一个整数字节可加快数据包的处理速度"（当时完全由软件完成）。也许有一天人们回顾过去，会责备设计电话号码的人并说："谁这么白痴。为什么不把地球编号包括在电话号码中？"但在当时，似乎没有什么必要。

为了处理这些问题，引入了子网这个概念，以便在一个组织内部灵活分配地址块。后来，又引入了 CIDR 技术来减小全局路由表的大小。今天，表明一个 IP 地址是否属于 A、B 或 C 类网络的标志位已不再使用，但在文献中对这些类的引用仍然很普遍。

为了看清楚地址分类如何使得转发更为复杂，先考虑在旧的分类系统中它有多简单。当一个数据包到达路由器，IP 地址的一个副本被右移 28 位，产生一个 4 位的类别号；然后 16 路分支把数据包归纳到 A、B、C（及 D 和 E）类，A 类有八种情况，B 类有四种情况，以及 C 类有两种情况；然后每个类别用掩码取出 8 位、16 位或 24 位的网络号，并且右对齐成 32 位的字。然后用该网络号查询 A、B 或 C 表，通常查询 A 和 B 类网络采用索引方式，而查询 C 网络则采用哈希算法。一旦找到对应的表项，即可查到相应的出境线路，进而转发数据包。这种查询操作比最长匹配前缀操作简单，因为最长匹配前缀面对的 IP 地址不是固定前缀长度，因而不能使用一个简单的查表操作。

D 类地址可继续用于 Internet 组播。实际上，或许可以更准确地说它们开始被用于组播了，因为过去 Internet 还没有广泛部署过组播。

还有几个其他地址有特殊的含义，如图 5-54 所示。IP 地址 0.0.0.0，这是最低的地址，由主机在启动时使用。这个地址意味着"这个网络"或者"这个主机"。全 0 的 IP 地址作为网络号指的是当前网络。这些地址允许机器在不知道网络号的情况下访问自己所在的网络（但它们必须知道网络掩码包括多少个 0）。由全 1 或 255.255.255.255 组成的地址，是最高的地址，用来标识指定网络中的所有主机。它允许在本地网络上广播，通常是局域网。具有正确的网络号和主机字段全 1 的地址允许机器向在 Internet 任何地方的遥远局域网发送广播数据包。然而，许多网络管理员禁用了此功能，主要原因在于这是一个安全隐患。最后，所有 127.xx.yy.zz 形式的地址保留给回环测试用。发送到该地址的数据包并没有被真正放在线路上，它们如同入境数据包一样在本地处理。这样在发送方不知道网络号的情况下，可以给主机发送数据包，这对测试网络软件很有用。

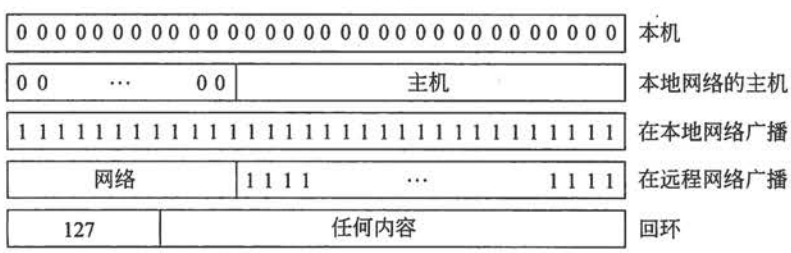


图 5-54 特殊 IP 地址

NAT——网络地址转换

IP 地址非常匮乏。一个 ISP 或许有一个/16 地址块，给它的 65 534 个可用主机编号。如果有比这更多的客户，那就有问题了。

这种匮乏导致了一些保守使用 IP 地址的技术。其中一种方法是为一台连在网上并使用网络的计算机动态分配一个 IP 地址，而且在该主机不活跃时收回分配给它的 IP 地址；然后该 IP 地址可以被分配给另一台活跃的计算机。以这种方式，一个/16 地址可以处理多达 65 534 个活跃用户。

这种动态分配策略在某些情况下运行良好，例如，那些可能暂时缺席或断电的拨号上网、移动和其他计算机。但是，它并不能很好地为企业客户工作。企业中的许多 PC 是要持续打开的。有些员工的机器在晚上需要做备份，而有些服务器可能需要在顷刻间服务于远程客户的请求。这些企业通常有一条总是提供与其余 Internet 连通的接入线路。

渐渐地，因为没有连接费用（仅仅是包月收费），这种情况出现在订购了 ADSL 或线缆上 Internet 服务的家庭用户。这些用户中的很多人在家里有两个或更多台计算机，通常每个家庭成员都有一台，而且他们都希望所有的时间都在线。相应的解决方案是通过局域网把所有的计算机连接到一个家庭网络，并且在家庭网络上放置一台（无线）路由器；然后路由器连接到 ISP。从 ISP 的角度来看，家庭网络现在就像一个拥有少数计算机的小型企业。用我们迄今为止所看到的技术，每台计算机必须整天都有它自己的 IP 地址。对于有成千上万客户的 ISP，特别是类似小型企业的商业客户和家庭客户，对 IP 地址的需求很快超出了可用的地址块。

IP 地址短缺问题并不是一个只有在将来某个时候才可能发生的理论问题。现在，此时此地，这个问题已经发生。对于整个 Internet 而言，长期的解决方案是迁移到 IPv6，它有 128 位地址。这个迁移过程正在缓慢地进行着，可能需要很多年才能完成。为了在这期间解决地址短缺问题，需要一个速战速决的方案。今天普遍使用的快速方案就是**网络地址转换**（NAT，Network Address Translation）形式，RFC 3022 描述了 NAT，下面我们简要介绍这项技术。有关更多的信息，请参考（Dutcher，2001）。

NAT 的基本思想是 ISP 为每个家庭或每个公司分配一个 IP 地址（或者，最多分配少量的 IP 地址），用这个 IP 地址来传输 Internet 流量。在客户网络内部，每台计算机有唯一的 IP 地址，该地址主要用来路由内部流量。然而，当一个数据包需要离开客户网络，发向其他 ISP 时，它必须执行一个地址转换，把唯一的内部 IP 地址转换成那个共享的公共 IP 地址。这种地址转换使用了 IP 地址的三个范围，这些地址已经被声明为私有化。任何网络可以在内部随意地使用这些地址。仅有的规则是不允许包含这些地址的数据包出现在 Internet 上。这 3 个保留的地址范围为：

10.0.0.0～10.255.255.255/8	（16 777 216 个主机）
172.16.0.0～172.31.255.255/12	（1 048 576 个主机）
192.168.0.0～192.168.255.255/16	（65 536 个主机）

第一段地址提供了 16 777 216 个地址（按常规，去掉全 0 和全 1 的地址），即使网络不那么大，通常大家都喜欢选择这个地址。

NAT 的操作过程如图 5-55 所示。在客户办公室内，每台机器都有一个形如 10.x.y.z 的

地址。然而，当一个数据包离开客户驻地时，它首先要通过一个**NAT盒子**（NAT box），此NAT盒子将内部的IP源地址（图中为10.0.0.1）转换成该客户所拥有的真实IP地址，在本例中为198.60.42.12。NAT盒子通常与防火墙组合成一个单一设备，防火墙提供了一种安全机制，它仔细地控制着进出客户网络的流量。我们将在第8章学习防火墙知识。另外还有一种可能，将NAT盒子集成到路由器或者ADSL调制解调器中。

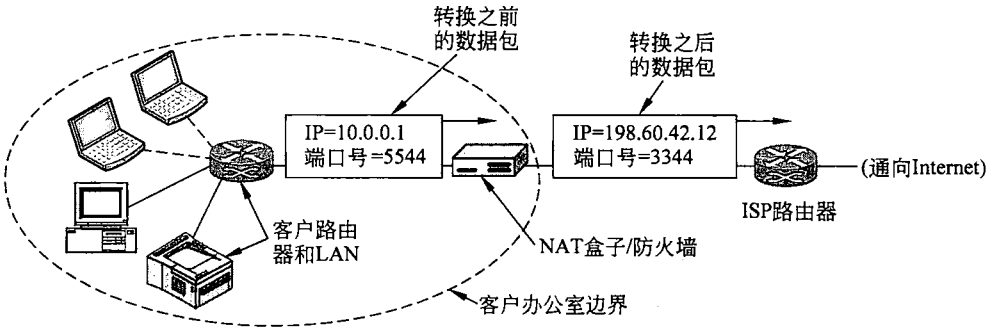


图5-55 NAT盒子的放置和操作过程

至此，我们掩盖了一个微小但至关重要的细节：当应答数据包返回时（比如从Web服务器返回的应答包），本质上它的目标地址是198.60.42.12，那么，NAT盒子如何知道该用哪一个地址来替代呢？这就是NAT需要解决的问题。如果在IP头有一个备用的字段，那么NAT盒子就可以用该字段记录真正的发送方到底是谁，但是在IP头仅仅只有1位还没有用。从原理上来说，我们还可以创建一个新的IP头选项来存放真实的源地址，但是，这么做将要求改变整个Internet上所有机器的IP代码，使得这些机器都能够处理新的选项。这显然也不是一种有效的快速解决方案。

NAT的实际做法如下所述。NAT设计者们注意到，大多数IP数据包携带的要么是TCP有效载荷，要么是UDP有效载荷。当我们在第6章学习TCP和UDP时，将会看到这两个协议的头都包含了一个源端口和一个目标端口字段。下面我们只讨论TCP端口，但是所讨论的内容同样适用于UDP端口。这里的端口是16位整数，它指示了TCP连接从哪里开始，以及到哪里结束。正是这些端口字段为NAT提供了工作所需的字段。

当一个进程希望与另一个远程进程建立TCP连接时，它把自己绑定到一个本地机器尚未使用的TCP端口上。该端口称为**源端口**（source port），它告诉TCP代码凡是属于该连接的入境数据包都应该发送给该端口。这个进程还要提供一个**目标端口**（destination port），以指明数据包传输到远程机器上之后应该交给谁处理。0～1023之间的端口都是保留端口，用于一些知名的服务。例如，端口80是Web服务器使用的端口，所以远程客户能找到Web服务器进程。每个出境TCP消息都包含一个源端口和一个目标端口。这两个端口合起来标识了客户机和服务器两端正在使用该连接的一对进程。

打个比方也许能更清楚地说明端口的用途。假设一家公司只有一个主电话号码。当人们拨打该号码，他们的呼叫到达接线员那里，接线员问他们要哪个分机，然后为他们接通所需的分机。这里的主号码就好比是公司的IP地址，两端的分机号就好比是端口。端口实际上是另一个16位的地址值，它标识了哪个进程接受入境数据包。

利用Source port（源端口）字段，我们可以解决前面的映射问题。任何时候当出境数

据包进入 NAT 盒子，其源地址 10.x.y.z 被客户的真实 IP 地址所取代，而且，TCP 的 Source port 字段被一个索引值取代，该索引值指向 NAT 盒子的地址转换表中 65 536 个表项之一。该表项包含了原来的 IP 地址和原来的源端口。最后，NAT 盒子重新计算 IP 头和 TCP 头的校验和，并将校验和插入到数据包中。这里之所以要替换 source port 域，是因为来自机器 10.0.0.1 和 10.0.0.2 连接可能碰巧使用了同一个端口，比如都使用了 5000，所以仅仅使用 Source port 还不足以唯一性地标识发送进程。

当一个数据包从 ISP 到达 NAT 盒子时，Source port 从 TCP 头中提取出，并被用作为索引值查找 NAT 盒子的映射表。找到对应的表项后，从该表项提取出内部 IP 地址和原来的 TCP Source port，并将它们插入到数据包中。然后重新计算 IP 和 TCP 校验和，并插入到数据包中。最后将该数据包传递给客户内部的路由器，它使用 10.x.y.z 地址进行正常的路由。

虽然这种方案从某种程度上解决了问题，但是，IP 团体中的网络纯粹主义者认为它不伦不类（地球上的丑八怪）。简单地概括，这里列举出一些反对意见。第一，NAT 违反了 IP 的结构模型。IP 的结构模型声明每个 IP 地址均唯一标识了世界上的一台机器。Internet 的软件结构也是建立在这样的事实基础之上的。采用了 NAT 之后，成千上万台机器可能会使用地址 10.0.0.1（事实上也确实如此）。

第二，NAT 打破了 Internet 的端-端的连接模型，即任何一个主机可在任何时间给任何一台其他主机发送数据包。因为 NAT 盒子上的映射是由出境数据包建立的，只能在出境数据包之后到达的入境数据包才能被接受。实际上，这意味着，家庭网络用户可以通过 NAT 与一台远程 Web 服务器建立 TCP/IP 连接，但远程用户无法与家庭网络内的一台游戏服务器建立连接，这种情况需要特殊的配置技术或者 **NAT 穿越**（NAT traversal）技术。

第三，NAT 将 Internet 从一个无连接网络改变成一个面向连接网络特有的形式。问题在于 NAT 盒子必须为每一个从它这里经过的连接维护必要的信息（即映射关系）。让网络维护连接状态是面向连接网络的一种特性，而不是无连接网络的特性。如果 NAT 盒子崩溃，并且它的映射表丢失，那么它的所有 TCP 连接都将被摧毁。没有 NAT 的情况下，路由器可以崩溃并重启，对 TCP 连接没有长远影响。发送进程只是在几秒钟内发生超时，然后重传所有未被确认的数据包。使用了 NAT 之后，Internet 如同电路交换网络一样，变得非常脆弱。

第四，NAT 违反了最基本的协议分层规则：第 k 层不应该对第 k+1 层在本层的有效载荷字段中放什么作任何假设。这条基本原则可以保证层与层之间的独立性。如果 TCP 后来升级到 TCP-2，它的头结构发生了变换（比如使用 32 位的端口号），那么 NAT 将不再能正常工作。分层协议的总体思想是保证某一层的变化不会要求其他层也跟着改变。NAT 破坏了这种独立性。

第五，Internet 上的进程并不一定必须使用 TCP 或者 UDP。如果机器 A 上的一个用户决定使用一种新的传输协议与机器 B 上的用户进行通话（比如，一个多媒体应用），那么，由于 NAT 的介入，这样的应用将无法工作，因为 NAT 盒子将无法正确地定位到 TCP 的 Source port。

第六及相关问题是，有些应用以规定的方式使用多个 TCP/IP 连接或者 UDP 端口。例如，标准的**文件传输协议**（FTP，File Transfer Protocol）在数据包正文插入 IP 地址，接收

方正文中提取出这些地址，并使用它们。由于NAT对这些地址安排一无所知，它不可能重写这些IP地址，或者说明情况。无法理解就意味着除非采取特殊的防范措施，否则FTP和其他一些应用，比如H.323 Internet电话协议（我们将在第7章讨论该协议），在必须通过NAT的情况下不能正常工作。一般可能的做法是在NAT盒子上打补丁。但是，每次有新应用出现时都要为NAT盒打补丁显然不是一个好主意。

最后，由于TCP Source port字段16位长，所以，至多只有65 536台机器可以被映射到同一个IP地址上。实际上，这个数值还要略小一些，因为前4096个端口被保留作特殊的用途。然而，如果有多个IP地址可用，那么，每个地址可以处理多达61 440台机器。

RFC 2993针对NAT的这些问题和其他一些问题进行了讨论。尽管存在问题，实际上，作为处理IP地址短缺的权宜之计技术，NAT已被广泛应用，特别是家庭和小型企业网络。它已与防火墙和隐私性结合在一起，因为它能默认阻止未经请求的入境数据包。出于这个原因，甚至当IPv6被广泛部署后，它也不太可能消失。

5.6.3 IPv6协议

IP已被大量使用了几十年。IP工作得相当好，Internet的指数级增长就是一个佐证。不幸的是，IP已经成为其自己知名度的牺牲品：地址即将殚精竭思。即使通过CIDR和NAT使用地址更加谨慎保守，预计最后IPv4地址有望在2012年底前被ICANN分配完毕。这种迫在眉睫的灾难几乎20年前就得到了大家的公认，它在Internet社团内部引发了大量应该做些什么的讨论和争议。

在本节，我们将介绍两个问题，并提出若干解决方案。唯一的长期解决方案是移动到更大的地址空间。IPv6（IP版本6）就是能做到这点的一个替换设计。它采用128位地址，在可预见的将来任何时间都不可能出现地址短缺这个问题。但是，IPv6已经被证明其部署非常困难。这是一个不同的网络层协议，尽管和IPv4有许多相似之处，但它并没有真正与IPv4实现互通。此外，公司和用户真的不知道为什么他们应该在任何情况下都要用IPv6。其结果是虽然部署了IPv6，但只有（估计为1%）一小部分的Internet在使用，而相应的Internet标准1998就已经发布。今后几年将会非常有趣，因为能分配的剩余IPv4地址已经为数不多了。人们会在eBay上开始拍卖自己的IPv4的地址？会因此而崛起一个IP地址黑市吗？天知道！

除了这些技术问题以外，还有其他隐藏在背后的问题。早期的Internet主要被大学、高科技工业和美国政府（特别是美国国防部）使用。20世纪90年代中期开始，随着对Internet兴趣的不断膨胀，Internet开始为各种人群所使用，通常人们的需求各不相同。首先，大量携带智能手机的用户通过Internet与他们的**家庭基地**（home bases）保持联系。其次，随着计算机、通信和娱乐业的不断交融，有可能在不久的将来，世界上的每一部电话和每一个电视都变成了Internet节点，从而几十亿台机器可进行音频和视频点播。很显然，在这样的形势下，IP必须要演进，要变得更加灵活。

1990年这些问题初露端倪，IETF开始启动IP新版本的设计工作，新版本的IP将有用不完的地址，并要解决许多其他的问题，同时必须更加灵活和高效。它的主要目标是：

（1）即使地址空间的分配效率不高，也能支持几十亿台主机。

（2）降低路由表的大小。

（3）简化协议，使路由器更快速地处理数据包。

（4）提供更好的安全（认证和隐私）。

（5）更加关注服务类型，特别是针对实时数据。

（6）辅助指定范围内的组播。

（7）主机漫游时无须改变地址。

（8）允许协议向未来演进。

（9）允许新老协议共存多年。

IPv6 协议的设计提出了一个重大机遇，可以借此机会改善 IPv4 缺少但现在又需要的所有特性。为了开发出一个满足所有这些需求的协议，IETF 在 RFC 1550 中发出了一个寻求提案和讨论的呼吁，刚开始共收到 21 份响应材料。到 1992 年 12 月，有 7 个提案被拿到了桌面上讨论。这些提案相差甚大，涉及范围非常广泛，从“对 IP 做微小的修改”，到“完全抛掉 IP 而用一个全然不同的协议来替代”。

其中一种提案是在 CLNP 之上运行 TCP。CLNP 是一个由 OSI 设计的网络层协议，它有 160 位地址，因为它可为海洋中的每个分子都分配足够的地址用来建立小型网络（大约为 2^5），所以地址空间永远够用。而且这种选择还将统一两个主要的网络层协议。然而，许多人认为，这样做好像承认了 OSI 领域所做的事情实际上是正确的，而在 Internet 圈子中却存在策略性的错误。CLNP 的模式与 IP 非常相近，所以这两种协议并没有实质性的不同。实际上，最终选中的协议与 IP 之间的差异，远远超过了 CLNP 与 IP 之间的差异。反对 CLNP 的另一个理由是它对服务类型的支持太差，而这对于有效传输多媒体数据是非常必要的。

IEEE Network 发表了三种比较好的提案（Deering，1993；Francis，1993；Katz 和 Ford，1993）。在经过多次讨论、修订和定位之后，Deering 和 Francis 两份提案被组合起来并又作了修改，然后得到一个现在称为简单 Internet 协议+（SIPP，Simple Internet Protocol Plus）的协议，最终，它被选中，并指定为 IPv6。

IPv6 很好地满足了 IETF 的设计目标。它保持了 IP 的优良特性，丢弃或者削弱了 IP 中不好的特性，并且在必要的地方增加了新的特性。一般而言，IPv6 并不与 IPv4 兼容；但是它与其他一些辅助性的 Internet 协议则是兼容的，包括 TCP、UDP、ICMP、IGMP、OSPF、BGP 和 DNS；若要处理更长的地址则需要做一点小小的改动。下面讨论 IPv6 的主要特性，有关更多的信息可以在 RFC 2460～2466 中找到。

第一个，也是最重要的是 IPv6 有比 IPv4 更长的地址。IPv6 的地址 128 位长，这解决了 IPv6 一开始就想要解决的问题：提供一个有效的无限量 Internet 地址。稍后我们马上还要更多地谈论地址。

IPv6 的第二个主要改进是对头进行了简化。它只包含 7 个字段（相比之下，IPv4 有 13 个字段）。这一变化使得路由器可以更快地处理数据包，从而提高了吞吐量，并缩短延迟。同样地，我们后面还要讨论 IPv6 的头结构。

第三个主要改进是更好地支持选项。这一变化对于新的头来说是本质的，因为以前那些必需的字段现在变成了可选；而且选项的表达方式也有所不同，这使得路由器可以非常简单地跳过那些与它无关的选项。此特性也加快了数据包的处理速度。

第四个改进代表了 IPv6 的重大进步，即在安全性方面的改进。IETF 已经听腻了关于

早熟的12岁少年用他们的个人计算机闯入Internet银行和军事堡垒的新闻故事，因此它们存在着一种强烈的意识，要在增强安全性方面做点事情。在新的IP中，认证和隐私是安全方面的关键特征。然而，后来这些特征也被引入到IPv4中，所以IPv6和IPv4在安全性方面的差异已经没有那么大了。

最后，更加值得关注的是服务质量。过去，人们在这方面做了大量半心半意的努力来改善QoS，但是现在，随着多媒体在Internet上的增长，这种紧迫性更加强烈了。

主要的IPv6头

IPv6的头如图5-56所示。对于IPv6，版本（Version）字段总是6（对于IPv4，该字段总是4）。在从IPv4到IPv6的迁移过程中（这个过程已经持续了10年），路由器通过检查该字段来确定数据包的类型。顺便提一下，做这样的测试在关键路径上需要浪费少量的指令，所以，一些路由器可能会跳过该项检查，数据链路层的头通常指名了多路分用中的网络层协议。例如，以太网的Type字段针对IPv4有效载荷和IPv6有效载荷，分别给出了不同的值。在“做得对”（Do it right）与“做得快”（Make it fast）两大阵营之间的讨论无疑将会既漫长又激烈。

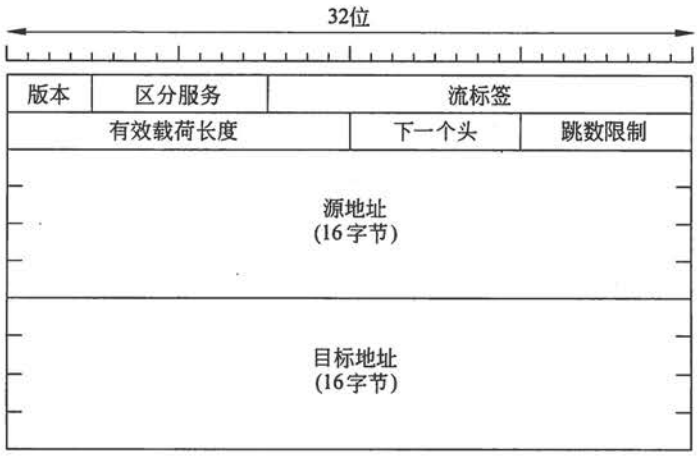


图5-56 IPv6固定头（必需的）

区分服务（Differentiated services，最初称为流量类别）字段的用途主要是区分数据包的服务类别，这些数据包具有不同的实时传递需求。它主要被用在服务质量的区分服务体系中，使用方式与IPv4数据包的同名字段一样。此外，最低2位用来发送显式拥塞指示，也与IPv4的方式相同。

流标签（Flow label）字段为源端和接收方提供了一种建立伪连接的方式，即源端和接收方把一组具有同样需求并希望得到网络同等对待的数据包打上标记。例如，从某台特定主机上一个进程到一台特定主机上一个进程之间的数据包流可能有严格的延迟要求，因此需要预留带宽。这时可以提前设置一个流（flow），并分配一个标识符。当一个流标签字段非0的数据包出现时，所有的路由器都在自己的内部表中查找该流标签值，看它要求哪一种特殊的待遇。实际上，这样的流是两种传输模型相结合的一种尝试：数据报网络的灵活性和虚电路网络的保障性。

为了保障服务质量，每个流由源地址、目标地址和流编号来指定。这意味着在给定的一对 IP 地址之间，可以同时有 220 个活跃的流。而且还意味着来自不同主机的两个流即使有相同的流标签，当它们通过同一台路由器时，路由器也能够根据源地址和目标地址将它们区分开。流标签的选取最好随机，而不是从 1 开始顺序分配，因此，路由器可对它们进行哈希处理。

有效载荷长度（Payload length）字段指明了紧跟在图 5-56 中 40 个字节头之后还有多少字节数。在 IPv4 中该字段的名字为总长度（Total length），之所以改成现在的名字是因为含义略有不同：40 字节的头不再像以前那样算作长度中的一部分。

正是下一个头（Next header）字段显示了 IPv6 的与众不同关键之处。IPv6 头得以简化的原因在于它可以有额外的（可选）扩展头。该字段指明了当前头之后还有哪种扩展头（当前已经定义了 6 种扩展头），如果有的话。如果当前的头是最后一个 IP 头，那么下一个头字段指定了该数据包将被传递给哪个传输协议处理（比如 TCP、UDP）。

跳数限制（Hop limit）字段被用来避免出现数据包永垂不朽的情形。实际上，它与 IPv4 中的 TTL（Time to live）字段是一样的，也就是说，在每一跳上该字段中的值都要被路由器递减。理论上，IPv4 中的 TTL 是一个以秒为单位的时间值，但是所有的路由器都不按照时间值来操作，所以在 IPv6 中将名字作了修改，以便反映出它的实际用法。

接下来是源地址（Source address）和目标地址（Destination address）字段。在 Deering 的原始提案（SIP）中，他提出使用 8 字节地址，但是在提案审阅过程中，许多人认为，如果用 8 个字节作为 IPv6 中的地址，那么在几十年之内地址空间将再次被用完，而如果使用 16 字节的地址，则永远也用不完；其他人则认为 16 个字节是矫枉过正，而还有些人主张使用 20 字节的地址以便与 OSI 数据报协议兼容；甚至还有人建议使用可变长度的地址。经过了太多的讨论和无法在教科书上说明的争执之后，最终做出决定，固定长度的 16 字节地址是最好的折中方案。

为了便于书写 16 字节的地址，一种新的标记法被设计了出来。16 个字节被分成 8 组来书写，每一组 4 个十六进制数字，组之间用冒号隔开，如下所示：

8000:0000:0000:0000:0123:4567:89AB:CDEF

由于许多地址的内部可能有很多个 0，所以，三种优化方法获得授权。第一，在一个组内可以省略前导 0，因此 0123 可以写成 123；第二，16 个“0”构成的一个或多个组可以用一对冒号来代替，因此，上面的地址现在可以写成：

8000::123:4567:89AB:CDEF

第三，IPv4 地址现在可以写成一对冒号再加上老式的点分十进制数，例如：

::192.31.20.46

弄清楚 IPv6 有多少个地址可能并没有必要，但是 16 字节的地址确实非常多。具体来说，应该有 2^{128} 个地址，近似等于 3×10^{38} 个地址。如果整个地球，包括陆地和水面都被计算机覆盖，那么，IPv6 将保证每平方米有 7×10^{25} 个地址。化学系的学生将会注意到，这个数值超出了阿伏伽德罗常数（Avogadro's number）。虽然 IPv6 的目标并不是要为地球表面上的每一个分子都分配单独的 IP 地址，但我们离那个目标并不遥远。

实际上，地址空间的使用效率不会非常高，就好像电话号码空间一样（曼哈顿的区号 212 几乎已经用满，而怀俄明州的区号 307 几乎还是空的）。在 RFC 3194 中，Durand 和

Huitema 对此作了计算，他们利用电话号码分配方案作为参照，经计算得到的结果是即使在最最不利的情形下，地球表面（包括陆地和水面）上每平方米仍将有远远超过 1000 个 IP 地址。在任何可能的场景下，每平方米将有几万亿个 IP 地址。总而言之，在可预见的将来，我们不太可能用得完这些地址。

比较 IPv4（图 5-46）与 IPv6 头（图 5-56）看看在 IPv6 中省掉了什么非常有意义。IHL 字段没有了，因为 IPv6 头有固定的长度。协议（Protocol）字段也被拿掉了，因为下一个头（Next header）字段指明了最后的 IP 头后面跟的是什么（比如 UDP 或者 TCP 段）。

所有与分段有关的字段都被去掉了，因为 IPv6 采用了另一种方法来实现分段。首先，所有遵从 IPv6 的主机都应该能够动态地确定将要使用的数据包长度。主机使用我们在 5.5.5 节描述的路径 MTU 发现过程就能做到这点。简要地说，当主机发送了一个非常大的 IPv6 数据包时，如果路由器不能转发这么大的数据包，它并不对该数据包进行分段，而是向发送主机返回一条报错消息。这条消息告诉主机，所有将来发送给该目标地址的数据包都要分段。让主机从一开始就发送大小合适的数据包，比让沿途路由器动态地对每个数据包进行分段要有效得多。而且，最小数据包长度也从 576 字节增加到 1280 字节，以便允许 1024 字节的数据和许多个头。

最后，校验和（Checksum）字段也被去掉了，因为计算校验和会极大地降低性能。现在使用的是大多是可靠网络，而且数据链路层和传输层通常有它们自己的校验和，所以在网络层上再使用校验和，相比它所付出的性能代价是不值得的。去掉了所有这些特性之后得到的是一个精简的网络层协议。因此，这份设计方案满足了 IPv6 的目标，即一个快速，但灵活，并且具有足够大地址空间的协议。

扩展头

有些省略掉的 IPv4 字段偶尔还会有用，所以，IPv6 引入了（可选的）扩展头（extension header）这一概念。这些扩展头可以用来提供一些额外的信息，但是它们以一种更有效的方式编码。目前定义了 6 种扩展头，如图 5-57 所列。每一种扩展头都是可选的，但是如果有多个扩展头出现，那么它们必须直接跟在固定头部的后面，而且最好使用表中列出的顺序。

扩展头	描述
逐跳选项	路由器的混杂信息
目标选项	给目的地的额外信息
路由	必须访问的松散路由器列表
分段	管理数据报分段
认证	验证发送方的身份
加密安全有效载荷	有关加密内容的信息

图 5-57 IPv6 扩展头

有些扩展头有固定的格式，其他扩展头包含数目不定的可变长度选项。对于所有这些可变选项，每一项都被编码成一个（Type, Length, Value）三元组。Type（类型）字段占一个字节，它指明这是什么选项。Type 的值有特殊的选取方法，它的前 2 位告诉路由器如果不知道如何处理该选项时应该如何处置数据包。选择方案有：跳过此选项；丢弃该数据包；丢弃该数据包并返回一个 ICMP 数据包；丢弃该数据包，但是对于组播地址不发送 ICMP

数据包（这样可以避免一个坏的组播数据包产生数百万个 ICMP 报告。）

长度（Length）字段也占一个字节，它说明了 Value 字段有多长（从 0～255 个字节）。值（Value）字段是任何扩展头所需要的信息，可以长达 255 个字节。

逐跳头（hop-by-hop header）用来存放沿途所有路由器必须要检查的信息。到现在为止，已经定义了一个选项：支持超过 64 KB 的数据报。该头的格式如图 5-58 所示。使用这种扩展头时，固定头中的有效载荷长度（Payload length）字段要设置为 0。

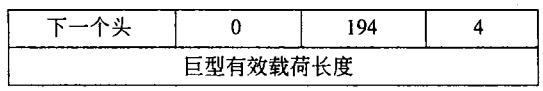


图 5-58 用于大型数据报（巨型报）的逐跳扩展头

与所有的扩展头一样，逐跳扩展头的起始字节也指定了接下去是哪一种头。该字节之后的字节指示了当前逐跳扩展头有多少个字节，其中不包括起始的 8 个字节，因为这 8 个字节是强制的。所有的扩展头都是以这种方式开始。

接下去的两个字节表明该选项定义了数据报的长度（代码 194），并且长度值以 4 字节计数。最后 4 个字节给出了数据报的长度。小于 65 535 的长度值是不允许的，第一台路由器将会丢弃这样的数据包，并且返回一个 ICMP 错误消息。使用这种扩展头的数据报称为巨型数据报（jumbogram），对于那些必须要通过 Internet 传输千兆字节数据的超级计算机应用来说，巨型数据报的使用非常重要。

目标选项扩展头（destination options header）用于那些只需被目标主机翻译的字段。在 IPv6 的初始版本中，唯一定义的选项是空选项（null option）。利用空选项可以将当前头拉长到 8 字节的倍数，所以它最初没有被使用。之所以设置这个选项是为了以防万一，也许某一天有人会想到一种新的目标选项，这样就能确保新的路由软件和主机软件可以对它进行处理。

路由扩展头列出了在通向目标的途中必须要经过的一台或者多台路由器。它非常类似于 IPv4 的松散源路由。在松散源路由机制中，凡是列出来的地址，必须要严格按顺序被访问到，但是，这些地址中间也可以经过一些没有列出来的其他路由器。路由头的格式如图 5-59 所示。

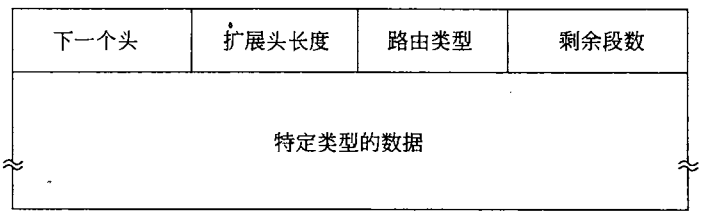


图 5-59 路由扩展头

路由扩展头的前 4 个字节包含了 4 个单字节整数。下一个头（Next header）和扩展头长度（Header extension length）字段如上面所述。路由类型（Routing type）字段给出了该扩展头剩余部分的格式。类型 0 表示在第一个字后面是一个保留的 32 位字，然后是一定数量的 IPv6 地址。将来根据需要还可以发明其他的类型。最后，剩余段数（Segments left）字段记录了在地址列表中还有多少个地址尚未被访问到。每当一个地址被访问时，该字段

中的数值减一。当它被减到0的时候，该数据包就完全获得自由，它不需要再遵循任何路由路径了。通常到这个时候它离目标已经非常接近，所以最佳路径也非常显然了。

分段扩展头（Fragment header）涉及与分段有关的事项，其处理方法与IPv4的做法非常类似。该扩展头保存了数据报的标识符、分段号，以及指明了后面是否还有更多段的标志位。然而，与IPv4不同的是，在IPv6中，只有源主机才可以将一个数据包进行分段。沿途的路由器可能不会进行分段。这一改变是对原始IP的重大哲学突破，但是符合IPv4的现行做法；而且它简化了路由器的工作，使得路由过程更快。正如上面所提到的，如果路由器面临一个太大的数据包，那么它可以丢弃该数据包并且向源主机发回一个ICMP包。这一信息允许源主机使用本扩展头把数据包分割成小的片段，然后再试着重新发送。

认证扩展头（Authentication header）提供了一种让数据包接收方确定发送方身份的机制。加密安全有效载荷扩展头（Encrypted security payload header）使得有可能对数据包的内容进行加密，因此，只有真正的接收方才可以读取数据包内容。这两个扩展头使用密码学技术来完成它们的任务。

争论

鉴于开放式设计过程和所涉及许多人的强烈意见，毫不奇怪，IPv6做了很多极具争议性的选择。下面我们简短地对这些争议做一番总结。要想了解全面的细节，请参考有关的RFC文档。

我们前面已经提到了有关地址长度的争论，最终的结果是一个折中的方案：16字节固定长度的地址。

另一个争论发生在跳数限制（Hop limit）字段。有一方强烈地认为，将最大跳数限制在255以内（也就是说，用一个8位的字段）是一个很显然的错误。毕竟，32跳的路径现在非常普遍，10年以后比这更长的路径也会变得十分普遍。这些人争辩说，使用大的地址长度很有远见，但是，使用小的跳计数则非常短视。在他们眼里，计算机科学家能够犯的最大错误就是在有些地方提供太少的数据位。

他们得到的回应是一旦贯彻就可能要增加每一个字段，从而导致一个非常臃肿的头部。跳数限制字段用来避免数据包长时间地逗留在网络中，但65 535跳实在是太长了。而且，随着Internet的增长，越来越多的长距离链路将被建立起来，从而使得从一个国家到达另一个国家可能至多不超过六跳。如果从源端和接收方分别到达它们相应的国际网关需要超过125跳，那么它们的国家主干网一定有问题了。所以，最终8位支持者们赢得了胜利。

另一个棘手的问题是最大数据包长度。超级计算机社团希望数据包长度可以超过64 KB。当一台超级计算机开始传输数据时，它的业务就开始了，当然不希望每64 KB被中断一次。反对派的观点是如果一个1MB的数据包被送到一条1.5 Mbps的T1线路上，那么该数据包将会占用线路5秒钟，对于共享同一线路的交互用户而言，这是一个明显感觉得到的延迟。在这一点上，最终达成的一致意见是：正常的数据包被限制在64 KB以内，但是允许使用逐跳扩展头来传送巨型数据报。

第三个热点话题是去掉IPv4校验和。有些人把这种做法比作从一辆汽车上拆除了刹车。拆除了刹车以后，汽车轻多了，所以它可以跑得更快；但如果发生意外事件，那就有问题了。

校验和反对派的论点是任何真正关心数据完整性的应用，不管用什么方法，一定要有一个传输层校验和，所以在IP中使用另一个校验和（而且还有一个数据链路层的校验和）纯属多余。更进一步，经验表明计算IP校验和是IPv4协议的一个主要开销。反对校验和阵营最终赢得了胜利，所以IPv6没有保留校验和。

移动主机也是一个争论的焦点。如果一台便携式计算机正在世界各地旅行，那么，它是继续使用同样的IPv6地址还是必须使用需要家乡代理支持的方案呢？有人希望在IPv6中对移动主机提供显式支持。由于任何一个提案都没有达成一致的意见，所以最终这一努力以失败而告终。

可能最大的争论在于安全性。每个人都承认安全性非常重要，争议的焦点在于在哪里和如何实现安全。首先是在哪里实现安全性。支持在网络层实现安全性的观点是，安全性变成一种标准的服务，于是所有的应用都可以使用安全服务，而无须任何提前规划。反对派则认为，真正的安全应用一般只需要端到端的加密，其中源端应用完成加密过程，接收方应用完成解密过程。网络层的实现可能会有错误，用户对此没有任何控制能力却要受到它的约束。对这种观点的回应是这些应用可以不使用IP的安全特性，改而自己实现端到端的安全性。对方又反驳道，那些不信任网络的人自己实现了安全性，即使禁用了网络层的安全功能也不想为这一慢速而又庞大的IP实现付出额外的费用。

与何处实现安全性相关的另一个话题是许多（并非全部）国家都有针对密码系统的严格出口法规。有些国家，特别是法国和伊拉克，限制在国内使用密码系统，因此老百姓对政府来说没有任何秘密可守。结果，任何一个使用了一定强度密码系统的IP实现不可能从美国（和许多其他的国家）出口给全球的客户。为此，软件厂商不得不维护两套软件，一套给国内客户使用，另一套出口到国外，这遭到了许多计算机厂商的强烈反对。

有一点倒是大家都没有任何异议，那就是没有人期望在某个星期天晚上IPv4 Internet被突然关闭，然后星期一早晨切换到IPv6 Internet。相反，对被隔离的IPv6“孤岛”进行改造，刚开始的时候可以通过隧道方式进行通信，我们在5.5.3节描述了隧道技术。随着IPv6岛的增加，它们将合并成更大的岛，最终所有的岛屿都合并到一起，于是Internet将完全转变为IPv6过来。

至少，这是个计划。已有的部署证明了IPv6的死穴。即使所有主要操作系统完全支持它，它也仍然很少被使用。大多数的IPv6部署是为了满足网络运营商需要大量IP地址的新情形，例如移动电话运营商。已经定义了许多策略，以便有助于缓解过渡压力。其中一些方法可以被主机用来自动配置隧道，通过IPv4 Internet运载IPv6；还有一些方法可使主机自动找到隧道出入口。双栈主机同时实现了IPv4和IPv6协议，可以根据数据包的目的地选择要使用的协议版本。这些战略将大幅简化IPv6的后续部署，当IPv4地址耗尽时IPv6的大量部署似乎是不可避免的。有关IPv6的更多信息，请参阅（Davies，2008）。

5.6.4 Internet控制协议

除了用于数据传输的IP协议外，Internet在网络层还有几个辅助控制协议。它们包括ICMP协议、ARP协议和DHCP协议。在本节，我们将依次考查每个协议，给出对应于IPv4的协议描述，因为这些协议是最常用的。针对IPv6，ICMP和DHCP有类似的版本，而与

ARP 等价的协议则称为 NDP（邻居发现协议）。

ICMP——Internet 控制消息协议

路由器严密监视 Internet 的操作。当路由器在处理一个数据包的过程中发生了意外，可通过 **Internet 控制消息协议**（ICMP，Internet Control Message Protocol）向数据包的源端报告有关事件；ICMP 还可以用来测试 Internet。已经定义的 ICMP 消息类型大约有 10 多种，每一种 ICMP 消息类型都被封装在一个 IP 数据包中。图 5-60 列出了最重要的一些消息类型。

消息类型	描述
目的地不可达	数据包无法传递
超时	TTL字段减为0
参数问题	无效的头字段
源抑制	抑制包
重定向	告知路由器有关地理信息
回显和回显应答	检查一台机器是否活着
请求/应答时间戳	与回显一样，但还要求时间戳
路由器通告/恳求	发现附近的路由器

图 5-60 主要的 ICMP 消息类型

当路由器不能定位一个目标，或者当一个设置了 DF 标志位的数据包由于途中经过一个“小数据包”网络而不能被递交时，路由器可以使用目的地不可达（DESTINATION UNREACHABLE）消息来报告这种情况。

当一个数据包由于它的 TTL（生存期）达到 0 而被丢弃时，路由器发送超时（TIME EXCEEDED）消息。这种事件往往预示着数据包进入了路由循环，或者计时器的超时值设置得太小。

这个错误信息可巧妙地被用于 Traceroute 工具，该工具由 Van Jacobson 在 1987 年开发。Traceroute 可发现从主机到目的地的路径上的 IP 地址，而且不需要任何特权网络的支持就能发现这种信息。方法其实很简单：给目标地址发送一系列的数据包，分别将 TTL 设置为 1、2、3，以此类推。这些数据包的计数值沿路径延伸而被后续路由器逐步递减为零，因此这些路由器各自乖乖地发送一个超时消息给发送主机。根据收到的这些返回信息，主机就可以确定路径沿途的路由器 IP 地址，以及跟踪路径各部分的统计数据和时间开销。这不是超时消息的本意，但它也许是最有用的网络调试工具。

参数问题（PARAMETER PROBLEM）消息表示在头字段中检测到一个非法值。这个问题说明了发送主机的 IP 软件中存在错误，或者也可能是中途路由器软件存在错误。

源抑制（SOURCE QUENCH）消息以前被用来抑制那些发送太多数据包的主机。当一台主机接收到这条消息时，它应该将发送速度减慢下来。这种消息现在很少使用了，因为当拥塞发生的时候，再发这些包无疑是火上浇油。现在，Internet 的拥塞控制任务主要由传输层完成；我们将在第 6 章中学习有关的细节。

当路由器注意到一个数据包看起来被错误地路由时，它使用重定向（REDIRECT）消息将可能的错误信息告诉源端主机。

回显（ECHO）和回显应答（ECHO REPLY）消息可以用来判断一个指定的目标是否可达，以及是否还活着。目标主机接收到回显消息之后，应该立即送回一个回显应答消息。这些消息主要被 ping 工具用在探测 Internet 上是否存在某一台特定的主机。

请求时间戳（TIMESTAMP REQUEST）和应答时间戳（TIMESTAMP REPLY）消息的用途类似，只不过在应答消息中还包含了请求消息的到达时间和应答消息的发出时间。此项设施可以用来测量网络的性能。

路由器通告（ROUTER ADVERTISEMENT）和路由器恳求（ROUTER SOLICITATION）消息使得主机拥有寻找附近路由器的能力。主机至少需要学习一个路由器的 IP 地址才能发送离开本地网络的数据包。

除了这些消息之外，标准还定义了其他消息类型。详细的消息类型可在线查询 www.iana.org/assignments/icmp-parameters。

ARP——地址解析协议

尽管 Internet 上的每台机器都有一个（或多个）IP 地址，但是仅有这些地址还不足以支撑发送数据包。数据链路层硬件网卡，比如以太网卡并不理解 Internet 地址。在以太网的环境里，每一块 NIC 在出厂时都配置了一个唯一的 48 位以太网地址。以太网 NIC 的制造商从 IEEE 请求一块以太网地址，确保不会出现任何两块网卡有相同的地址（以避免两块网卡出现在同一个 LAN 上时发生冲突）。NIC 根据其 48 位以太网地址来发送和接收帧。它们对 32 位的 IP 地址完全一无所知。

现在问题来了：如何将 IP 地址映射到数据链路层的地址，比如以太网地址呢？为了解释这一工作过程，我们来看图 5-61 给出的例子，这个例子演示了一个规模较小的大学，它只有两个/24 网络。计算机系有一个交换式以太网（CS），它的前缀是 192.32.65.0/24；另一个在电子工程学系，它的前缀为 192.32.63.0/24。这两个局域网通过一个 IP 路由器相连。以太网上的每台机器和路由器上的每个接口都有一个唯一的以太网地址，我们将它们标记为 CS 或 EE 网络上的 E1 至 E6。

我们来看 CS 网络上主机 1 的用户如何给主机 2 的用户发送数据包。假设发送方知道目标接收方的名字，可能像 eagle.cs.uni.edu 这样的名字。第一步是找到主机 2 的 IP 地址。这个查找过程由域名系统（DNS，Domain Name System）完成，我们将在第 7 章学习域名系统。此刻，我们假设 DNS 返回主机 2 的 IP 地址（192.32.65.5）。

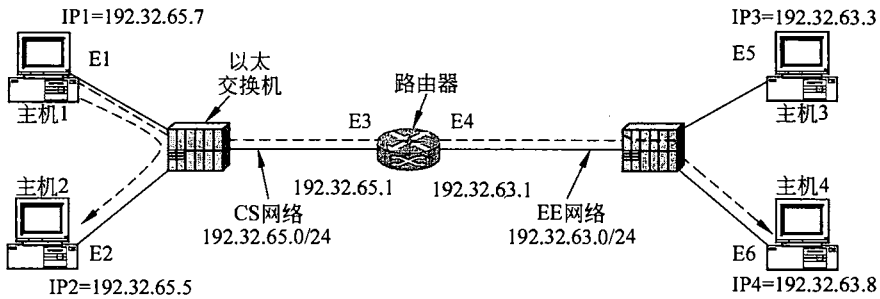
主机 1 上层软件现在构建一个数据包，其 Destination address 字段为 192.32.65.5，然后它将该数据包交给 IP 软件来发送。IP 软件看到该地址后发现这个目标地址就在 CS 网络上（即它自己所在的网络）。然而，它仍然需要某一种办法来找出目标主机的以太网地址才能发送帧。一种解决方案是在系统中设置一个配置文件，该配置文件给出了从 IP 地址到以太网地址的映射关系。虽然这种方案当然是可能的，但是对于拥有几千台机器的组织来说，在所有主机上保持配置文件并及时更新，则是一件既容易出错，又费时的任务。

一个更好的解决方案是主机 1 发送一个广播包到以太网络上请求拥有 IP 地址 192.32.65.5 的主机。该广播包将会到达 CS 网络上的每一台主机，并且每台主机都会检查自己的 IP 地址。只有主机 2 会用自己的以太网地址（E2）作为应答。通过这种方式，主机 1 得知 IP 地址 192.32.65.5 是一台拥有以太网地址为 E2 的主机。请求和获得应答两个过程

所使用的协议称为**地址解析协议**（ARP，Address Resolution Protocol）。几乎 Internet 上的每一台机器都运行这个协议。RFC 826 定义了 ARP。

与采用配置文件相比，使用 ARP 协议的优点是简单。系统管理员只要给每台机器分配一个 IP 地址，并且确定好子网掩码，不用做其他任何事情。ARP 会负责处理好所有其他的事情。

这时候，主机 1 上的 IP 软件构建一个以太网帧，其目标地址为 E2，并且把 IP 数据包（目标地址为 192.32.65.5）放到以太网帧的有效载荷字段中，然后将它发送到以太网上。图 5-61 给出了该数据包的 IP 地址和以太网地址。主机 2 的以太网 NIC 检测到这一帧，并识别出这是发给自己的帧，于是将它接收进来，并产生一个中断；以太网驱动程序从有效载荷中提取出 IP 数据包，并将它传递给 IP 软件，IP 软件看到它的目标地址正是指向自己，于是对它进行处理。



帧	源IP	源以太	目的IP	目的以太
主机1到主机2，CS网络	IP1	E1	IP2	E2
主机1到主机4，CS网络	IP1	E1	IP4	E3
主机1到主机4，EE网络	IP1	E4	IP4	E6

图 5-61 通过路由器联结两个交换式以太 LAN

为了使 ARP 的效率更高，可以进行各种优化处理。首先，一旦一台机器已经运行了 ARP，那么，它可以将结果缓存起来，以便稍后它与同一台机器通信时再用。当下次通信时，它就可以在缓存中找到所需的地址映射关系，从而避免了进行第二次广播。在许多情况下，主机 2 必须返回一个应答，这就迫使它也要运行 ARP 来确定发送方的以太网地址。此 ARP 广播也是可以避免的，因为主机 1 可以将它的“IP-以太网”地址映射关系包含在它的 ARP 包中。当 ARP 广播包到达主机 2 时，（192.31.65.7, E1）这对映射关系也被存入主机 2 的 ARP 缓存。事实上，以太网上的所有机器都可以将这对映射关系放到它们的 ARP 缓存中。

为了允许映射关系发生变化，比如为一台主机配置了一个新 IP 地址（但保留其老的以太网地址），ARP 缓存中的相应表项应该在几分钟之后发生超时。保持缓存信息最新状态并能优化性能的一种更聪明方式是，让每台机器配置之后广播它的地址映射关系。这次广播通常以 ARP 的形式发送，即主机发送一个 ARP 请求查找它自己的 IP 地址。按理来说，网络上应该不会有任何应答，这个广播数据包的副作用就是其他主机在 ARP 缓存中加入一个映射表项。这就是所谓的免费 ARP（gratuitous ARP）。如果意外地收到了一个应答，那么一定是两台机器被分配了相同的 IP 地址。网络管理员必须解决这个问题，否则这两台机

器都不能使用网络。

现在我们再来看图 5-61，只不过这一次假设主机 1 想要给 EE 网络上的主机 4（192.32.63.8）发送数据包。主机 1 发现目标 IP 地址不在 CS 网络。同时它知道应该把所有这些网络外的流量发给路由器，该路由器称为**默认网关**（Default gateway）。按照惯例，默认网关具有网络上的最低地址（198.31.65.1）。为了给路由器发送帧，主机 1 必须知道该路由器在 CS 网络上的接口地址。因此，主机 1 发送一个 ARP 广播报文，请求 198.31.65.1 对应的以太网地址，从该广播报文的应答报文它获知所需的以太网地址为 E3；然后用该地址给路由器发送帧。数据包沿着一条 Internet 路径从一个路由器被转发到下一个路由器时就是采用这种查询机制的。

当路由器的以太网 NIC 得到该帧后，它将数据包交给 IP 软件。IP 软件从网络掩码了解到这个数据包要发送到 EE 网络，在 EE 网络再到达主机 4。如果路由器不知道主机 4 的以太网地址，它可以再次使用 ARP。在图 5-61 的表中列出了从 CS 和 EE 网络观察到的帧的源和目标的以太网地址和 IP 地址。我们观察到出现在每个网络上的帧的以太网地址发生了改变，而 IP 地址保持不变（因为它们表示所有互联网络的端点）。

还有一种可能，当主机 1 不知道主机 4 在另一个不同网络时，仍然可以从主机 1 发送一个数据包给主机 4。解决办法是让 CS 网络上的路由器回答针对主机 4 的 ARP 请求，并且以 E3 作为响应。直接由主机 4 来响应对自己的 ARP 请求报文是不可能的，因为它根本看不到 ARP 请求（路由器不会转发以太网级的广播报文）。然后，路由器将收到发给 192.32.63.8 的帧，并将该帧转发到 EE 网络。这个解决方案称为 **ARP 代理**（proxy ARP）。这种方案常用在这样一种特殊情况下：一个主机想出现在一个网络上，即使它实际上在另一个网络上。例如，一个常见的情况是有个移动笔记本，当它离开家乡网络时，其他节点仍然能给它发送数据包。

DHCP——动态主机配置协议

ARP（以及其他 Internet 协议）都做了这样的假设，即主机配置了一些基本信息，比如自己的 IP 地址。主机如何获得此信息？手动配置每台计算机是可能的，但那既乏味又容易出错。有一个更好的方法可以完成这件事情，就是**动态主机配置协议**（DHCP，Dynamic Host Configuration Protocol）。

采用 DHCP 时，每个网络必须有一个 DHCP 服务器负责地址配置。当计算机启动时，它有一个嵌入在 NIC 中的内置以太网地址或其他链路层地址，但没有 IP 地址。像 ARP 一样，该计算机在自己的网络上广播一个报文，请求 IP 地址。这个请求报文就是 DHCP DISCOVER 包，这个包必须到达 DHCP 服务器。如果 DHCP 服务器没有直接连在本地网络，那么必须将路由器配置成能接收 DHCP 广播并将该请求报文中继给 DHCP 服务器，由 DHCP 服务器来处理 DHCP 报文。

当 DHCP 服务器收到请求，它就为该主机分配一个空闲的 IP 地址，并通过 DHCP OFFER 包返回给主机（这个报文或许也要通过路由器中继）。为了在主机没有 IP 地址的情况下完成此项工作，服务器用主机的以太网地址来标识这台主机（主机的以太网地址由 DHCP DISCOVER 包携带过来）。

IP 地址的自动分配有个问题，就是从地址池中取出一个 IP 地址分配给主机能用多久。

如果一个主机离开网络，并且没有把分配给它的IP地址返回给DHCP服务器，那么该地址将永久丢失。过一段时间，很多地址都可能因此而丢失。为了防止这种情况发生，可以为每个分配的IP地址指定一段固定时间，这种技术称为租赁（leasing）。在租赁期满前，主机必须请求DHCP续订。如果没有提出续订请求或请求被拒绝，主机或许不能使用以前分配给它的IP地址。

DHCP由RFC2131和RFC2132描述。它已被广泛应用于Internet，可为主机配置除了IP地址以外的其他各种参数。它还被用于企业和家庭网络，ISP使用DHCP来设置Internet接入链路上的设备参数，因此客户不需要给他们的ISP打电话来获得这些信息。常见的配置信息例子包括网络掩码、默认网关的IP地址、DNS服务器和时间服务器的IP地址。DHCP已在很大程度上取代了先前使用的协议（称为RARP和BOOTP），因为这些协议的功能比较有限。

5.6.5 标签交换和MPLS

到目前为止，在我们的Internet网络层巡视中，注意力都专注于数据包，因为这是IP路由器转发的数据报。还有另一种技术正在开始广为应用，特别是被ISP用来在它们的网络之间移动Internet流量。这种技术称为**多协议标签交换**（MPLS，MultiProtocol Label Switching），它非常接近电路交换。尽管Internet社团中的很多人对面向连接的网络有强烈的反感，但这个想法似乎又走了回头路。正如Yogi Berra曾指出的那样，这是从头再来似曾相识。然而，Internet处理路由建立的方式与面向连接网络对路由的处理方式两者之间存在着本质上的区别，所以这种技术肯定不是传统的电路交换。

MPLS在每个数据包前面增加一个标签，路由器根据数据包标签而不是数据包目标地址实施转发。用标签作为一个内部表的索引，快速查找该表找出正确的输出线路，因而这只是一个表查操作。使用这种技术，路由器的转发速度非常快。这种优势是MPLS的最原始设计动机。刚开始时这项独特技术有各种名称，其中包括**标记交换**（tag switching）。最终，IETF开始规范化这样的想法，相应的描述发表在RFC 3031和许多其他RFC文档。随着时间的推移，MPLS的好处显现了出来，主要表现在灵活的路由和快速以及适合服务质量的转发。

第一个要问的问题是标签放在哪里？由于IP数据包并不是针对虚电路设计的，所以在IP头并没有可以存放虚电路号的空间。由于这个原因，必须要在IP数据包的头前面加上一个新的MPLS头。如果从一台路由器到另一台路由器之间的线路使用PPP作为成帧协议，那么帧格式中包含了PPP、MPLS、IP和TCP头，如图5-62所示。

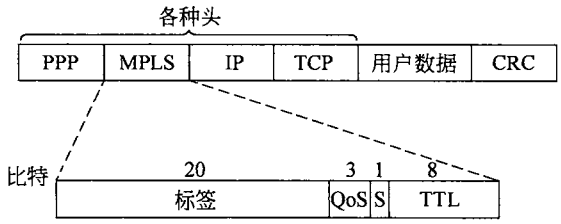


图5-62 使用IP、MPLS和PPP传输一个TCP段

通用的MPLS头有4个字节长，并且包含4个字段。其中最重要的是标签（Label）字段，它存放的是索引。QoS字段指明了服务的类别。S字段涉及在层次网络中叠加多个标签的做法（下面会进一步讨论）。TTL字段指出该数据包还能被转发多少次，每经过一个路由器被递减1。如果降为0，则该数据包将被丢弃，这个特性可以防止在路由不稳定的情况下出现无限循环问题。

MPLS介于IP网络层协议和PPP链路层协议之间。它不是一个真正的第3层协议，因为它依赖IP或其他网络层地址来建立标签路径；但它也不是一个真正的第2层协议，因为它既可以在多跳之间转发数据包，也不是一条单一链路。出于这个原因，MPLS有时称为2.5层协议。这是一个例子，说明真正的协议并不总是符合我们的理想分层协议模型。

MPLS具有其光明的一面，因为MPLS头既不属于网络层数据包也不属于数据链路层帧的一部分，所以MPLS在相当大的程度上独立于这两层。除此之外，这个属性还意味着有可能制造出能同时转发IP数据包和非IP数据包的MPLS交换机，具体如何转发取决于线路上出现的是什么。这个特性就是MPLS名字中“多协议”的由来。MPLS还可以通过非IP网络来运载IP数据包。

当一个MPLS增强型数据包到达一个**标签交换路由器**（LSR，Label Switched Router），标签就被用作查找一个表的索引，以便确定要使用的出境线路和新标签。所有的虚电路网络几乎都采用这种标签替换技术。标签只有本地意义，两个不同的路由器可以给两个都到另一台路由器但相互之间毫无关系的数据包分配相同的标签，这两个数据包的进一步传输将使用同一条出境线路。为了在虚电路的另一端能区别出这两个数据包，标签必须在每一跳重新进行映射。我们已经在图5.3看过这个机制是如何工作的。MPLS使用了与此相同的技术。

顺便说一句，有些人将路由器的转发行为和交换行为进行了区分。转发是一个过程，从一个表中找出与目标地址最佳匹配的表项，并据此决定将数据包发往哪里。一个例子是IP转发使用了最长前缀匹配算法。与此相反，交换则使用取自数据包的标签作为索引来查询转发表，这个过程更为简单和迅速。然而，这些定义还远未普及。

由于大多数主机和路由器都不理解MPLS，因此我们还应该考虑何时以及如何把标签附加到数据包上。这事发生在一个IP数据包到达MPLS网络边缘。**标签边缘路由器**（LER，Label Edge Router）检查数据包的目标IP地址和其他字段，确定该数据包应当遵循哪条MPLS路径，然后右侧的标签贴在数据包前面。进入MPLS网络后，这个标签就被用来转发数据包。在MPLS网络的另一边，标签已经完成自身使命并功成身退；删除标签后得到原始的IP数据包，然后再被下一个网络转发。这个过程如图5-63所示。MPLS与传统虚电路的一个区别是聚合水平。在通过MPLS网络时，为每个流设置它自己的一组标签当然是可能的；然而，对于路由器来说，更常见的做法是将终止在某个特定路由器或者LAN的多个流合并成一组，并为这些流使用同一个标签。这些被合起来共享同一个标签的流称为属于同一个**转发等价类**（FEC，Forwarding Equivalence Class）。该类别不仅覆盖了数据包的去向，而且覆盖了它们的服务类别（从区分服务的意义上），因为从转发的角度而言，所有的数据包都被同样对待。

若使用传统的虚电路路由方法，要想把几条具有不同端点（endpoint）的独立路径组合到同一个虚电路标识符上是不可能的，因为在最终接收方无法将它们区分开。有了MPLS，

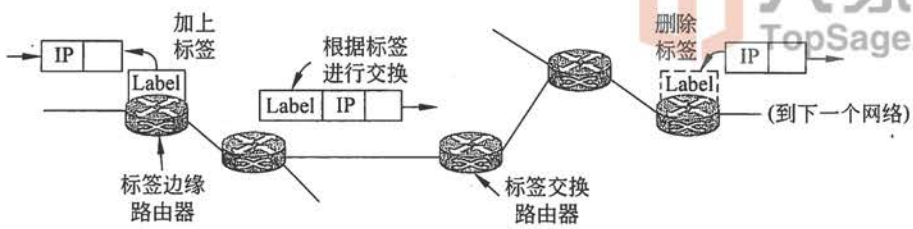


图 5-63 通过一个 MPLS 网络转发 IP 数据包

数据包除了标签以外，仍然包含了它们的最终目标地址，所以，在标签路径的末端，标签头可以被去掉，然后利用网络层的目标地址，按照常规的方法继续向前转发。

其实，MPLS 走得更远。它可以一次在多个标签层次上运行，具体做法是在数据包前面添加多个标签。例如，假设有许多具有不同标签的数据包（因为我们要处理的数据包位于网络中的不同地方），遵循同一条共同的目标路径。对此，不是设置多条标签交换路径，即为每个不同的标签设置一条标签交换路径，而是为这些数据包设置一条单一路径。当已标记的数据包到达这条路径的开始端点处，另一个标签被添加到数据包的前面。这就是所谓的标签栈（stack of label）。最外面的标签指导这些数据包沿着该条公共路径前行。在路径的结束端点处，取出进入标签交换路由前贴上的标签，原来的标签（如果有的话）用来指导数据包进一步的转发。图 5-62 中的 S 标志位告诉删除外层标签的路由器是否还剩下额外的标签。对于最底部的标签，该标志位设为 1；而对所有其他标签，该标志位设置为 0。

最后一个我们会考虑的问题是如何建立标签转发表，因而数据包可遵循该表的指示被转发。这是 MPLS 与传统虚电路设计两者之间的主要区别。在传统的虚电路网络，当用户希望建立一个连接时，会向网络发出一个设置包，通过该包来创建路径并生成转发表的表项。MPLS 在设置阶段不涉及用户。如果用户除了发送数据包之外，还要做任何其他事情将打破现有的 Internet 软件。

相反，转发信息由一些协议设置，这些协议一般是路由协议和连接建立协议的组合。这些控制协议与标签转发分离得干干净净，这种分离具有可以使用多个不同控制协议的便利。其中的一个变种是这样工作的：当路由器启动时，它会检查看看自己是哪些路由的最终目的地（例如，哪个前缀属于它的接口）；然后它为这些路由创建一个或多个 FEC，并且为每个 FEC 分配一个标签，然后把这些标签发给其邻居。反过来，这些邻居在自己的转发表中添加这些标签，再把新标签发送给其邻居；如此一般地往外传播，直到所有的路由器都获得了相应的路径。资源也可以像路径构造一样被预留下来，以便保证适当的服务质量。其他的变种可以设置不同的路径，比如流量工程路径考虑了未使用的容量，并且按需创建路径以便支持服务产品，比如服务质量。

尽管 MPLS 背后的基本思想非常直截了当，但它的细节异常复杂，并且它有许多变种方法和实际部署的使用实例。有关更多的信息，请参考（Davie 和 Farrel，2008；Davie 和 Rekhter，2000）。

5.6.6 OSPF——内部网关路由协议

我们现在已经完成了如何在 Internet 转发数据包的学习。现在继续前进到下一个主题：

Internet 路由。正如我们前面提到的，Internet 由大量的独立网络或**自治系统**（AS，Autonomous System）构成，并由不同的组织运营，这些组织通常是公司、大学或 ISP。在自己网络内部，一个组织可以使用自己的内部路由算法，或者更流行的名称叫**域内路由算法**（intradomain routing）。不过，流行的只有极少数几个标准协议。在本节，我们将了解域内路由问题，并考查 OSPF 协议，这是一个被普遍实际使用的路由协议。域内路由协议也称为**内部网关协议**（interior gateway protocol）。在下一节，我们将探讨独立运营网络之间的路由问题，或**域间路由**（interdomain routing）问题。在这种情况下，所有网络必须使用相同的域间路由协议和**外部网关协议**（exterior gateway protocol）。Internet 采用的域间路由协议是**边界网关协议**（BGP，Border Gateway Protocol）。

早期的域内路由协议采用了距离矢量的设计思想，基于分布式 Bellman-Ford 算法，该算法继承自 ARPANET。**路由信息协议**（RIP，Routing Information Protocol）是当时运行的一个主要例子。在小型网络系统中 RIP 运作良好，但随着网络规模变得越来越大它工作得就不那么好了；而且它还遭受无穷计数问题的困扰，收敛速度一般很慢。因为这些问题，1979 年 5 月 ARPANET 切换到一个链路状态协议，1988 年 IETF 开始为域内路由设计一个链路状态路由协议。该协议在 1990 年成为标准，它就是**开放最短路径优先**（OSPF，Open Shortest Path First）。它借鉴了另一个称为**中间系统到中间系统**（IS-IS，Intermediate-System to Intermediate-System）的协议，该协议已经成为一个 ISO 标准。由于它们的共同根源，这两个协议非常的大同小异。有关它们的完整故事，请参见 RFC 2328。它们在域内路由协议中占有绝对的优势，大多数路由器制造商都同时支持这两个协议。OSPF 更广泛地应用在公司网络，而 IS-IS 则更多地应用在 ISP 网络。这两者之间，我们将给出 OSPF 如何工作的大致轮廓。

由于有了其他路由协议的长期工作经验，负责设计新协议的工作组列出了一系列必须要满足的需求。第一，该算法必须发表在公开的文献中，这便是 OSPF 中 O（开放的）的含义。由某一家公司拥有的私有方案是无法做到这一点的。第二，新的协议必须支持多种距离度量，包括物理距离、延迟等。第三，它必须是一个动态算法，能够自动而且快速地适应网络拓扑变化。

第四，对于 OSPF 来说这是新的需求，它必须支持基于服务类型的路由。新的协议必须能够区分实时流量和其他的流量，并使用不同的路由方法。当时，IP 协议有一个服务类型（type of service）字段，但是没有一个路由协议使用这个字段。OSPF 也包含了该字段，但是仍然没有人使用它，最终它又被去掉了。也许这种需求有点超前，因为它先于 IETF 的区分服务工作展开前提出，区分服务模式使得服务质量重新焕发了活力。

第五条与上面一条有关，OSPF 必须实现负载均衡，即把负载分散到多条线路上。大多数以前的协议都将所有的数据包通过最优路径发送出去，即使存在两条同等程度好的路由也只选择一条使用，其他路径根本不用。在许多情况下，将负载分散到多条线路上可以获得更好的性能。

第六，必须支持层次化系统。到 1988 年，一些网络已经增长到相当大的规模，以至于任何一台路由器都不可能知道其完整的拓扑结构。OSPF 必须设计成不要求路由器知道完整的拓扑结构也能很好地工作。

第七，要求提供适度的安全性，以防止恶作剧的学生向路由器发送虚假路由信息来欺

骗路由器。

第八，对于那些通过隧道连接到 Internet 的路由器，新协议也必须能够对它们进行处理。以前的协议并不能很好地解决这样的问题。

OSPF 同时支持点到点链路（比如，SONET）和广播网络（例如，大多数局域网）。其实，它能够支持拥有多个路由器的网络，这些路由器中的每一个都可以直接与其他路由器通信（称为多路访问网络），即使它们没有广播能力。此前的协议不能很好地处理这种情况。

图 5-64（a）给出了一个自治系统网络。这里主机被省略了，因为它们在 OSPF 中通常不起作用，真正参与路由协议的是路由器和网络（其中可能包含主机）。图 5-64（a）中的大多数路由器通过点到点链路连接到其他路由器和网络，最终到达这些网络中的主机。然而，路由器 R3、R4 和 R5 则是通过一个广播局域网连接，诸如交换式以太网。

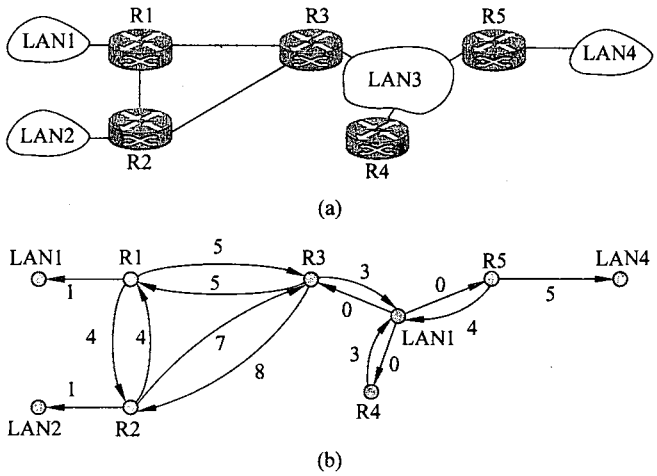


图 5-64

（a）一个自治系统示例；（b）针对图（a）的图形表示

OSPF 的工作方式本质上是对一张图进行操作：将一组实际网络、路由器和线路抽象到一个有向图中，图中的每条弧有一个权值（距离、延迟等）。两台路由器之间的点到点连接可以用一对弧来表示，每个方向上一个，两个方向上的权值可以不同。广播网络用一个节点表示，加上网络上每台路由器用一个节点表示，从网络节点到路由器节点之间的弧段权值为 0。尽管如此，它们是重要的，因为没有它们就没有通过网络的路径。其他网络只有主机，因此只需要一条到达网络的弧，没有返回弧。这种结构使得路由可以到达主机但不能穿过主机。

图 5-64（b）显示的是图 5-64（a）所示网络的图形表示。OSPF 协议从根本上做了两件事情，首先用一个类似这样的图来表示实际的网络，然后每个路由器使用链路状态方法计算从自身出发到所有其他节点的最短路径。有可能协议会发现多个同样短的路径，在这种情况下，OSPF 记住最短路径集合，并在报文转发期间把流量分摊到这些路径上。这种多路径路由方法有助于负载均衡。该方法称为**等价成本多路径**（ECMP，Equal Cost MultiPath）。

Internet 中的许多 AS 本身非常庞大，而且不便于管理。OSPF 可以将这样的 AS 划分成编号的区域（area），每个区域是一个网络，或者一组互连的网络。区域不能相互重叠，但

是也不必面面俱到，也就是说有些路由器可能不属于任何一个区域。全部属于一个区域的路由器称为**内部路由器**（internal router）。区域是单个网络的一种泛化形式。在区域外部，能见到的是它的目的地而不是拓扑结构。

每个 AS 有一个**骨干区域**（backbone area），称为 0 号区域。该区域中的路由器称为骨干路由器（backbone router）。所有区域都必须连接到骨干区域，连接方式有可能会通过隧道进行；所以，从 AS 内的任何一个区域出发，经过骨干区域，总是有可能到达该 AS 的任何其他区域。在图形表示法中，隧道也用一个弧来表示，并且有一个权值。如同所有其他的区域一样，骨干区域的拓扑结构对于其外部也是不可见的。

每个连接到两个或更多区域的路由器称为**区域边界路由器**（border router）。它必须是骨干区域的一部分。区域边界路由器的工作任务：概括本区域的目的地信息并注入到与自己连接的其他区域。这种概括包含成本信息，但不包括区域内的所有拓扑细节。传递成本信息可以使得其他区域内的主机找到进入本区域的最好区域边界路由器。不传递拓扑信息可以减少流量和简化其他区域路由器的最短路径计算。然而，如果只有一个边界路由器通往区域外，甚至路由信息概要都不需要传递。通往该区域外部目的地的路由总是被指令“前往边界路由器”，这类区域称为**存根区域**（stub area）。

最后一种路由器是 **AS 边界路由器**（AS boundary router）。它把通往其他 AS 的外部路由注入到本区域。然后外部路由就呈现为可以通过 AS 边界路由器可达的目的地，该路由当然会有某种成本。外部路由可以注入一个或更多个 AS 边界路由器。图 5-65 给出了自治系统、区域和各种路由器之间的关系。一个路由器可以扮演多种角色，例如，一个边界路由器同时还是一个骨干路由器。

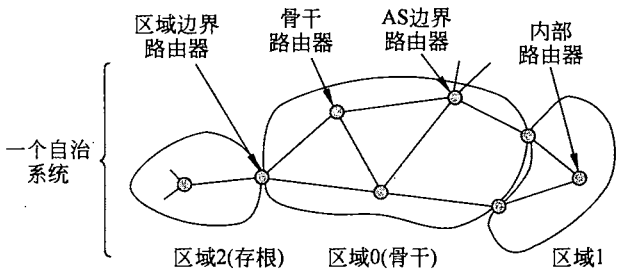


图 5-65 OSPF 中 AS、骨干网和区域相互之间的关系

在正常操作期间，每个区域内的路由器有相同的链路状态数据库，并运行相同的最短路径算法。其主要工作是计算从自身出发到每个其他路由器和整个 AS 内网络的最短路径。区域边界路由器需要所有与之连接区域的数据库，并且为每个区域分别运行最短路径算法。

对于在同一区域内的源和目的地，选择最好的区域内路由（全部位于该地区内）。对于不在同一个区域内的源和目的地，区域间路由必须先从源所在区域到骨干区域，再从骨干区域到目标区域，最后到达目的地。这种算法强制把 OSPF 配置成星型结构，骨干区域相当于集线器，其他区域是向外辐射区域。因为路由算法选择的是最小成本路由，因此位于网络不同位置的路由器可能会选择不同的区域边界路由器进入骨干区域和目标区域。从源到目的地的数据包被“如此这般地”路由，它们不需要封装或者隧道（除非目标区域与骨干区域的唯一连接是一个隧道）。此外，通往外部目的地的路由如果需要可以包括外部成本，

或者仅包含 AS 内部成本。外部成本指从 AS 边界路由器通往目的地的外部路径的成本。

当一台路由器启动时，它在所有的点到点线路上发送 HELLO 消息，并且通过 LAN 将 HELLO 消息组播到一个包含所有其他路由器的组。每台路由器从应答消息中得知谁是自己的邻居。同一个 LAN 上的路由器都是邻居。

OSPF 协议需要在邻接的路由器之间相互交换信息才能工作，**邻接**（adjacent）路由器与邻居路由器是不同的。尤其是，让一个 LAN 中的每台路由器都跟本 LAN 中的其他每台路由器进行交换路由信息显然非常低效。为了避免出现这样的情形，OSPF 要求从每个 LAN 中选举一台路由器作为**指定路由器**（designated router）。指定路由器与本 LAN 上的所有其他路由器是邻接的，并且与它们交换信息。实际上，它就是一个代表本 LAN 的单个节点。邻居但不是邻接的路由器相互之间并不交换信息。有一台备份的指定路由器总是保持最新的状态数据，以便缓解主指定路由器崩溃时的转接和取代主指定路由器的需要。

在正常操作过程中，每台路由器周期性地泛洪 LINK STATE UPDATE 消息到它的每台邻接路由器。这些消息给出了它的状态信息，并提供了拓扑数据库用到的成本信息。这些泛洪消息需要被确认，以保证它们的传输可靠性。每条消息都有一个序号，路由器据此判断一条入境 LINK STATE UPDATE 消息比它当前拥有的信息更老还是更新。当一条线路启用、停止或者其成本发生改变时，路由器也要发送 LINK STATE UPDATE 消息。

DATABASE DESCRIPTION 消息给出了由发送方持有的所有链路状态表项的序号。通过把自己相应的值与发送方传过来的这些值进行比较，接收方即可决定谁拥有最新的值。当一条链路启动时使用这些消息。

通过使用 LINK STATE REQUEST 消息，每一对邻接路由器中的任一个路由器都可以向另一个路由器请求链路状态信息。这个算法的结果是，每一对邻接路由器都可检查谁有最新的数据；新的信息就是通过这种方式被传播到整个区域。所有这些消息都是以 IP 数据包的形式直接被发送出去。图 5-66 概括了这 5 类消息。

消息类型	描述
HELLO	用来发现所有的邻居
LINK STATE UPDATE	提供发送者到其邻居的成本
LINK STATE ACK	对链路状态更新消息的确认
DATABASE DESCRIPTION	声明发送者的链路状态更新情况
LINK STATE REQUEST	请求链路状态信息

图 5-66 OSPF 的 5 类消息

最后，我们把所有的工作综合到一起。通过泛洪法，每个路由器把它与其他路由器和网络的链路以及链路成本告诉给它所在区域中的所有其他路由器。这些信息使得每台路由器都可以构建出它所在区域的拓扑图，并且计算最短路径。骨干区域也是这样工作的。而且，为了计算出从每个骨干路由器到每个其他路由器的最佳路由，路由器还要接受来自每个区域边界路由器的信息。最佳路由信息又被传回到区域边界路由器，区域边界路由器再将这些信息在本区域内广播。利用这些信息，内部路由器可以选择通往区域外目的地的最优路由，包括通向骨干区域的最佳出口路由器。

5.6.7 BGP——外部网关路由协议

在一个 AS 内部，推荐使用的路由协议是 OSPF 和 IS-IS。在 AS 之间，则可以使用另一个协议，称为**边界网关协议**（BGP，Border Gateway Protocol）。之所以在 AS 之间需要一个完全不同的协议，是因为域内协议和域间协议的目标不同。域内协议所需要做的只是尽可能有效地将数据包从源端传送到接收方，它不必考虑政治方面的因素。

相反，域间路由协议则必须要考虑大量和有关政治的因素（Metz，2001）。例如，一个公司的 AS 可能希望能给所有的 Internet 站点发送数据包，同时也能够接收来自任何一个 Internet 站点的数据包。然而，它可能不愿意承载那些源自一个外部 AS 而终止于另一个外部 AS 的数据包，即使它自己的 AS 正好位于这两个外部 AS 之间的最短路径上（“那是他们的问题，不关我们的事”）。另一方面，它可能愿意转送其邻居们的流量，或者愿意为那些已经付费的一些特殊 AS 提供流量中转服务。例如，电话公司可能很愿意为它们的客户充当运载工具，但是不愿意为别人也提供这样的服务。无论是一般意义上的外部网关协议，还是特殊的 BGP 协议，它们都被设计成允许多种路由策略，这些策略可被强制用在那些跨越 AS 的流量传送上。

典型的路由策略可能涉及政治、安全或者经济方面的考虑因素。下面是一些可能的路由限制例子：

（1）教育网络不承载商业流量。

（2）五角大楼发出的流量永远不要走经过伊拉克的路径。

（3）使用 TeliaSonera 而不用 Verizon，因为前者便宜。

（4）不要使用澳大利亚的 AT&T，因为它的性能太差。

（5）起止于苹果的流量不应该经过谷歌中转。

正如你从这个列表可能想象的那样，路由策略可以因人而异。它们通常是专有的，因为它们包含了特定的敏感商业信息。然而，我们可以通过描述一些模式来捕捉上述公司的理由，而且常常以此作为出发点。

路由策略的实施决定了哪些流量可以流过 AS 之间的哪些链路。一个常见的政策是客户 ISP 给提供商 ISP 付费，以便将数据包传送到 Internet 上的任何其他目的地以及接收来自 Internet 上任何其他目的地的数据包。可以说客户 ISP 从提供商 ISP 购买了**中转服务**（transit service）。这就像家庭客户从 ISP 购买了 Internet 接入服务一样。为了能工作，提供商应该把到达 Internet 上全部目的地的路由通过它们之间的链路通告给客户。这样，客户就有一条用来传送数据包到任何地方的可用路由。相反，客户应该向提供商通告它自己网络通往目的地的路由。这样提供商可以只给客户发送要去往来那些目的地址的流量；客户不希望处理来往其他目的地的流量。

我们看一个中转服务例子，如图 5-67 所示。这里有 4 个相互连接的 AS。连接通常采用了 Internet 交换点（IXP，Internet eXchange Point）的链路，为了与其他 ISP 连接，许多 ISP 都有一条链路连到该设施。AS2、AS3 和 AS4 都是 AS1 的客户，它们从 AS1 购买了中转服务。因此，当源 A 给目的地 C 发送时，数据包从 AS2 经过 AS1，最后到达 AS4。路由通告的方向和数据包传送方向相反。为了源端能经过 AS1 到达 C， AS4 向它的中转服

务提供商，即 AS1 通告 C 是一个目的地。以后，AS1 向它的其他客户公告到达 C 的路由，其中包括 AS2，以便客户知道它们可以通过 AS1 发送流量给 C。

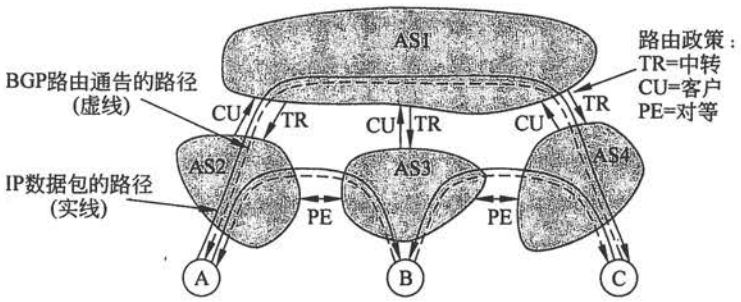


图 5-67 4 个自治系统的路由政策

在图 5-67 中，所有其他 AS 向 AS1 购买中转服务。这种中转服务提供了良好的连接性，使它们可以和 Internet 上的任何主机联系。然而，它们必须为此特权付出代价。假设 AS2 和 AS3 之间有大量的流量需要交流。由于它们的网络早就连接，因此如果它们愿意，它们可以使用不同的政策——直接给彼此发送免费流量。这将减少必须通过 AS1 替它们传递的流量，并有望降低它们的账单。这一政策称为对等（peering）传输。

为了实现对等传输，两个 AS 相互通告到达目的地在自己网络的路由。这样做就有可能使 AS2 把从 A 到 B 的数据包发给 A3，反之亦然。但是，必须注意，对等不等于中转。在图 5-67 中，AS3 和 AS4 相互对等，这种对等关系允许从 C 到 B 的流量可直接发送到 AS4。如果 C 要发送一个数据包给 A 会发生什么事呢？AS3 仅仅通告了一条到 B 的路径给 AS4，并没有通告有一条到 A 的路由。因此，流量无法从 AS4 传到 AS3，再传到 AS2，即使存在一条物理路径。这种限制正是 AS3 所希望得到的，它与 AS4 对等交换流量，但不希望运载来自 AS4 去往 Internet 其他地方的流量，因为它得不到对方支付的这笔中转费用。相反，AS4 从 AS1 获得中转服务。因此，正是 AS1 负责携带从 C 到 A 的数据包。

现在，我们知道了有关中转流量和对等流量，我们可以看到 A、B、和 C 都有中转安排。例如，A 必须购买 AS2 的 Internet 接入服务。A 可能是一台简单的家庭计算机，或者一个具有多个局域网的公司网络。然而，它并不需要运行 BGP 协议，因为它只是一个存根网络，只有一条链路与 Internet 的其余部分连接。所以，它给网络外部目的地发送数据包的唯一途径就是通往 AS2 的链路，除此之外无路可寻。这条路径的设置非常简单，只需设立一个默认路由即可。出于这个原因，我们没有在图中显示出作为自治系统的 A、B 和 C 参与域间路由。

另一方面，有些公司的网络连接到多个 ISP。这种技术主要用来提高可靠性，因为如果通往一个 ISP 的路径失败，公司可以使用通往其他 ISP 的路径。这种技术称为**多穴寻址**（multihoming）。在这种情况下，公司的网络有可能运行域间路由协议（比如 BGP），告诉其他 AS 通过哪些 ISP 链路可以到达哪些地址。

这些中转和对等流量的政策可能有许多变种，但它们早已显示了路由通告的商业关系和控制，以及这些因素如何实现不同类型的政策。现在我们要更详细地考虑运行 BGP 的路由器*如何相互*通告路由信息和选择转发数据包的路由。

BGP 是距离矢量协议的一种形式，但它与域内距离矢量协议（比如 RIP）有很大的不

同。我们已经看到用政策而不是最小距离来选择使用哪些路由。另一个很大的区别是，BGP不仅维护到每个目的地的成本，而且每个BGP路由器还跟踪所使用的路径。这种方法称为**路径矢量协议**（path vector protocol）。路径由下一跳路由器（有可能在ISP的另一侧，不一定相邻）和一系列AS或者AS路径组成，这些自治系统或者相应的路径序列是该路由必须遵守的（以相反顺序给出）。最后，一对BGP路由器通过建立TCP连接而相互通信。这种方式既提供了可靠的通信，也隐藏了正在穿越的网络细节。

图5-68中例子显示了BGP通告路由的过程。这里有三个自治系统，中间的ISP为左右两个ISP提供中转服务。AS3启动一个前缀C的路由通告。当该路由通告信息通过跨界链路到达R2c（在图的上方），它就具有一条到AS3的简单路径，并且下一跳路由器是R3a。在底部，它有一条相同的AS路径，但下一跳路由器不同，因为它来自不同的链路。这则路由广告继续传播，并跨越AS边界进入AS1。在路由器R1a（在图的上方），AS路径为AS2、AS3，并且下一跳是R2a。

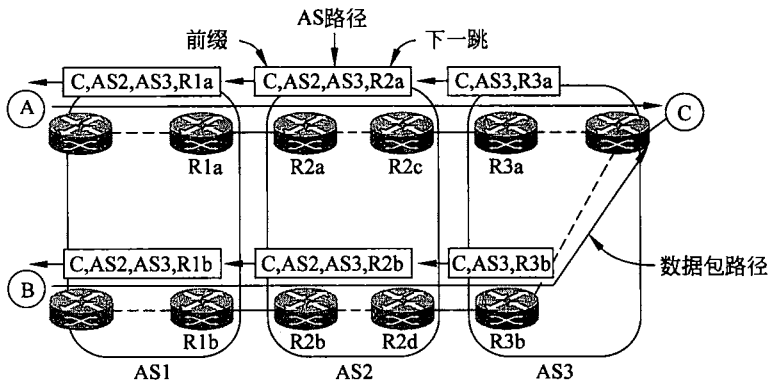


图5-68 BGP路由通告的传播

路由通告消息中携带完整的路径易于让接收路由器发现和打破路由循环。相应的规则是每个路由器在往AS外部发送路由时附加上自己的AS编号（这就是为什么列出的路径是相反顺序的）。当路由器收到一个路由通告时，它会检查消息，检查自己的AS号是否已经出现在AS路径中。如果是，则说明检测到了一个路由循环，因而必须丢弃路由通告。然而，有点讽刺意义的是，20世纪90年代后期人们才认识到尽管有这种预防措施，BGP仍然遭受无穷计数问题的困扰（Labovitz等，2001）。BGP没有长寿的路由环路，但有时路由会收敛得很慢，而且有中转循环。

给出一个AS列表来指定一条路径是非常粗糙的表达方式。一个AS可能是一个小公司，也可能是一个国际骨干网络。没有任何途径告知这些路由的详情，BGP甚至不去尝试了解。因为不同的AS可能使用不同的域内路由协议，因此这些路由协议的成本无法比较；即使它们能相互比较，一个AS或许还不愿意透露其内部的路由度量值。这正是域间路由协议与域内路由协议在方式上的不同所在。

至此，我们已经看到了两个ISP如何通过链路发送路由通告消息。我们仍然需要一些方法将BGP路由从ISP的一侧传播到另一侧，只有这样它们才能被发送到下一个ISP。这个任务可以由域内协议承担，但由于BGP非常善于扩展到大型网络，因此通常使用BGP的一个变种。这就是所谓的内部BGP（iBGP，internal BGP），以示区别于BGP的常规应

用，即外部BGP（eBGP，external BGP）。

在ISP内部传播路由通告有一定的规则：位于ISP边界的每个路由器为了一致性，要学习所有其他边界路由器看到的路由。如果ISP的一个边界路由器学习到一个去往IP 128.208.0.0/16的前缀，那么该ISP的所有其他路由器都要学习这个前缀。然后从ISP的任何地方都可到达该前缀，不管数据包从其他AS如何进入ISP。

为避免混乱我们并没有在图5-68中显示这种传播，但是，例如，路由器R2b知道通过顶部的路由器R2c或者底部的路由器R2d可以到达C。随着路由在ISP内穿越，下一跳得到更新，因此位于ISP远端的路由器知道使用另一侧的哪些路由器离开ISP。这就是为什么最左边的路由看到的下一跳路由器在同一个ISP内，而不是下一个ISP内的路由器。

现在，我们可以描述BGP协议的最关键部分，就是路由器如何为每个目的地选择使用哪条路由。每个BGP路由器从与它连接的下一个ISP路由器学习如何到达给定目的地的路由，也可以从所有其他边界路由器那里（它们已经从与其他ISP连接的路由器上听到不同的路由）学习到给定目标的路由。每个路由器必须决定这些路由集合中哪条路由最好。最终的答案是由ISP指定某些政策，根据政策来挑选其中一条首选路由。然而，这种解释太笼统，无法令人满意，所以我们可以至少说明一些共同策略。

第一个策略，优先选择通过对等网络的路由，而不是通过中转提供商的路由。因为前者免费，而后者要付成本费用。类似的策略是给予客户路由最高的优先。只有良好的商业模式才直接给付费客户发送流量。

另一种不同的策略是将“短路径更好”作为默认规则。这个规则值得商榷。因为一个AS可能是任何规模的网络，所以通过三个小AS的路径实际上可能比通过一个大AS的路径短。然而，平均来说短路径趋向于更好，并且这条规则是一个普遍规律。

最后一个策略是优先选择具有ISP内最小成本的路由。这就是图5-68实施的策略。从A发往C的数据包从顶部路由器R1a离开AS1。从B发出的数据包通过底部路由器R1b出口。这么选择的原因在于无论A和B采取的都是离开AS1最小成本路由或者最快路由。因为A和B位于ISP的不同部分，因此对它们每个来说离开AS的最快出口是不同的。同样的事情发生在数据包通过AS2。在最后一站，AS3必须用自己的网络来运载来自B的数据包。

这一策略称为提前退出（early exit）或热土豆路由（hot-potato routing）。它有很好奇的副作用，即使得路由呈现不对称性。例如，考虑C发送一个数据包返回到B时采取的路径。数据包将通过顶部的路由器很快离开AS3，以避免浪费资源；同样，当AS2以尽可能快的速度将该数据包传给AS1时，它一直留在顶部的路由上；然后，该数据包将在AS1内经过一条较长的路程。这是一条从B到C所用路径的镜像。

上述讨论应该明确每个BGP路由器从已知的可能路由中选择自己的最佳路由。情况并非如人们天真地预期那样，BGP按照AS粒度来选择路径，OSPF选择每个AS的内部路径。BGP和内部网关协议的集成相当的根深蒂固。这意味着，例如，BGP可以找到从一个ISP到下一个ISP的最好出口点，并且这个点在ISP内是各不相同的，如同热土豆策略情况。这也意味着位于一个AS内不同部分的BGP路由器到达相同的目的地可能选择不同的AS路径。ISP必须小心配置全部的BGP路由器，以便自由地做出兼容的路由选择，但是这可以在实践中逐步完成。

令人惊讶的是，我们的讨论只触及了 BGP 的皮毛。有关详细信息，请参阅由 RFC4271 和相关 RFC 文档给出的 BGPv4 规范。然而，要认识到其复杂性更多地在于政策，这不是 BGP 协议规范所描述的内容。

5.6.8 Internet 组播

普通的 IP 通信发生在一个发送方和一个接收方之间。然而，对于有些应用，它们需要一个能够同时向大量接收方发送数据的进程。这样的应用例子有许多，比如给许多观众视频直播体育比赛、向复制服务器池更新程序以及处理数字会议（即多方会议）中的电话呼叫。

IP 用 D 类 IP 地址来支持一对多的通信，或组播。每个 D 类地址标识了一组主机。总共有 28 位可用于标识组播组，因此网络中可同时并存 250 万个组。当一个进程给一个 D 类地址发送数据包时，网络会尽力而为地将这些数据包投递给指定组中的所有成员，但是并不保证一定投递成功。有些成员可能收不到数据包。

IP 地址 224.0.0.0/24 范围内的地址保留用作本地网络组播。在这种情况下，不需要路由协议的支持。带有一个组播地址的数据包被简单广播到局域网上，从而达到组播的目的。局域网上的所有主机接收广播数据包，只有属于组成员的主机对该数据包进行处理。路由器不会将数据包转发到局域网外。本地组播地址的例子有：

224.0.0.1	LAN 上的全部系统
224.0.0.2	LAN 上的全部路由器
224.0.0.5	LAN 上的全部 OSPF 路由器
224.0.0.251	LAN 上的全部 DNS 服务器

其他组播地址或许有成员分布在不同的网络上。在这种情况下，就需要一个路由协议。但首先组播路由器必须了解哪些主机属于某个组的成员。一个进程要求它的主机加入到某个指定的组中；它也可以要求它的主机离开该组。每台主机跟踪记录当前它的哪些进程属于哪些组。当一台主机上的最后一个进程离开一个组时，该主机就不再属于这个组。大约每分钟一次，每个组播路由器向它所在 LAN 上的所有主机发送一个查询数据包（当然使用本地组播地址 224.0.0.1），要求这些主机报告自己当前属于哪些组。组播路由器可能与标准的路由器在同一台机器上，也可能分属两台机器。每个主机收到查询消息后，返回一个响应包，其中包括了自己感兴趣的所有 D 类地址。这些查询包和应答包使用了一个称为 **Internet 组管理协议**（IGMP，Internet Group Management Protocol）的协议，该协议由 RFC3376 详细描述。

几个组播路由协议中的任何一个都可用于建立组播生成树，该树给出了一条从发送方到组内所有成员的路径。所用的算法是我们在 5.2.8 节中描述的那些。在 AS 内，主要使用的是**协议独立组播协议**（PIM，Protocol Independent Protocol）。PIM 适合几种场合。在密集模式 PIM 中，算法创建了一棵修剪的逆向路径转发树。这棵树比较适合于组成员分布在网络各处的情况，比如数据中心网络把文件分发给多个服务器。在稀疏模式 PIM 中，算法创建的生成树类似于核心树。这种方式比较适合内容提供商向它 IP 网络上的用户组播 TV。这种设计的一种变体，称为**特定源组播 PIM**（Source-Specific Multicast PIM）可用来优化只有一个发送方的情况。最后，当组成员分布在不止一个 AS 时，需要 BGP 或隧道的组播扩展来创建组播路由。

5.6.9 移动 IP

许多 Internet 用户拥有移动计算机，当他们离开家乡甚至在旅途的过程中，他们也希望能够与 Internet 保持连接。不幸的是，IP 的寻址系统使得这样的异地办公说起来容易做起来很难，正如我们马上要描述的那样。当人们开始产生在任何地点都能上网的需求时，IETF 成立了一个工作组来寻求解决方案。该工作组很快制定出了任何一种解决方案都必须达到的一系列目标，其中最主要的有以下这些：

（1）每台移动主机必须能在任何地方使用它的家乡 IP 地址。

（2）不允许修改固定主机的软件。

（3）不允许改动路由器软件和各类表。

（4）发给移动主机的大多数数据包不应该绕道而行。

（5）移动主机在家时不应该有任何开销。

选择的解决方案是我们在 5.2.10 节描述的那样。简单地说，每个允许用户漫游的网点必须创建一个称为家乡代理（home agent）的助手（helper）。当移动主机出现在一个外地网点时，它获得一个外地网点的新 IP 地址（称为转交地址）；然后移动主机通过该转交地址告诉家乡代理自己现在哪里。当发给该移动主机的数据包到达家乡网点，恰好移动主机不在家外出到了其他地方，那么家乡代理截取该数据包，并隧道给接入到当前转交地址上的移动主机。不管通信对方是谁，移动主机都可以直接发送应答数据包，但仍把家乡地址作为应答数据包的源地址。这个解决方案符合上述所有目标，除了移动主机的数据包必须走弯路这一点外。

既然我们已经覆盖了 Internet 的网络层，现在可以了解该解决方案的更多细节。对移动性支持的需求首先来自于 IP 寻址模式本身。每个 IP 地址包含一个网络号和主机号。例如，考虑一台机器具有 IP 地址 160.80.40.20/16。160.80 给出了该机器所在网络的编号，40.20 是所在网络内的主机号。世界各地的路由器都有路由表说明到达 160.80 网络该用哪条链路。每当到达一个数据包，如果其目标 IP 地址形如 160.80.xxx.yyy，那么路由器就把它从该条链路发送出去。如果突然间，该地址上的机器被搬运到某个遥远网点，寻址该地址的数据包将继续被路由到其家乡局域网（或路由器）。

在这一阶段，有两种选择，但都没有吸引力。第一，我们可以创建一个到更特殊前缀的路由。也就是说，如果一个遥远网点通告了一条到 160.80.40.20/32 的路由，那么发往该目标地址的数据包首先会被发送到适当的地方。这个选项取决于路由器使用的最长匹配前缀算法。然而，我们必须为单个 IP 地址增加一条到一个 IP 前缀的路由。如果每个人带着计算机移动时都要修改全局 IP 路由，那么每个路由器将有数百万的表项/表项，这是个天文数字的成本。显然这种方法在 Internet 上是行不通的。

第二个选择是改变移动主机的 IP 地址。诚然，发送到家乡 IP 地址的数据包将不再被传递，直到所有相关人员、程序和数据库都知晓移动主机的地址变化情况。但移动主机仍然可以在新的位置使用 Internet 浏览网页和运行其他应用程序。这个选项在更高的层次处理移动性，通常发生在用户带着一台笔记本电脑到咖啡店坐着，通过本地无线局域网络使用 Internet。这个选项的缺点是它要中断某些应用程序，因为移动主机来回移动时没有保持

与网络的连接。

顺便说一句，移动性也可以在一个较低层次处理，比如链路层。这通常发生在一台笔记本电脑通过单个 802.11 无线网络上网。此时移动主机的 IP 地址不会发生改变，因而网络路径保持不变。这正是无线链路提供了对移动性的支持。然而，移动性的程度非常有限。如果笔记本电脑移动得太远，它就必须通过另一个具有不同 IP 地址的网络接入到 Internet。

针对 IPv4 的移动 IP 解决方案由 RFC 3344 给出。它可在现有的 Internet 路由下工作，允许主机移动外出时用自己的 IP 地址保持与网络连接。为了能工作，移动主机必须能够发现自己在移动。这点可以通过 ICMP 路由器**通告**（advertisement）和**恳求**（solicitation）消息做到。移动主机定期监听路由器的通告报文或通过恳求报文来发现最近的路由器。如果该路由器的地址不同于移动主机平常在家用的路由器地址，则它可以断定自己必定在某个外地网络。如果这个路由器地址自上一次修改以来又发生了变化，则说明移动主机已经转移到了另一个外地网络。同样的机制可以使得移动主机发现它们的家乡代理，从而确定自己从外地返回了家乡。

为了在外地网络得到一个转交 IP 地址，移动主机只需简单地使用 DHCP 服务。另外，如果 IPv4 地址短缺，移动主机可以通过外地代理发送和接收数据包，该外地代理早已经有一个当地网络上的 IP 地址。移动主机用来发现外地代理使用的 ICMP 机制与用来寻找家乡代理使用的机制相同。一旦移动主机获得一个 IP 地址或找到一个外地代理，它就能够利用当地网络给家乡代理发送信息，告知其当前位置。

当移动主机不在家乡网络时，其家乡代理需要某种方式来拦截发给移动主机的数据包。ARP 协议提供了一个方便的机制。要通过以太网给一个 IP 主机发送数据包，路由器需要知道主机的以太网地址。通常的机制是由路由器发送一个 ARP 请求查询报文，例如问“160.80.40.20 的以太网地址是什么？”。当移动主机在家乡网络时，它以自己的 IP 地址与自己的以太网地址回答 ARP 查询。当移动主机外出时，家乡代理用自己的以太网地址来响应此查询。然后路由器把发给 160.80.40.20 的数据包发给家乡代理。回想一下，这就是所谓的 ARP 代理。

为了在移动主机离开或返回家乡网络时快速更新 ARP 映射，采用了另一个 ARP 技术，称为免费 ARP（gratuitous ARP）。基本上，移动主机或家乡代理自己会发送针对该移动主机 IP 地址的 ARP 查询报文，在该查询报文中提供了正确的答案，因此该路由器注意后及时更新其映射。

家乡代理给移动主机转发数据包其实很简单，在家乡代理和连接在转交地址上的移动主机之间建立一个隧道，然后通过该隧道来转发数据包。具体做法是只要用目标地址是转交地址的另一个 IP 头封装该数据包，当封装后的数据包到达转交地址后，外层 IP 头被拆除，提取出里面的数据包交给移动主机。

与许多 Internet 协议一样，麻烦在细节上，并且大多数往往体现在部署时与其他协议的兼容性细节上。这次存在两种并发症。首先，NAT 盒子依赖于经过的 IP 数据包，通过偷窥其 IP 头来查看 TCP 头或 UDP 头。隧道移动 IP 的原始形式没有使用这些头，因此不能与 NAT 盒子一起工作。对此的解决办法是改变封装，使得封装后的数据包包括 UDP 头。

第二个复杂性在于一些 ISP 检查数据包的源 IP 地址，看它们是否匹配路由协议认可的应该出现的地方。这种技术称为入口过滤（ingress filtering），它是一种安全措施，路由器

会丢弃那些看似不正确地址的数据包，因为这有可能是恶意的流量。然而，当移动主机在外地网络，它发送到其他 Internet 主机的数据包的源地址不属于所在的外地网络，所以这样的数据包将被外地网络的路由器丢弃。为了解决这个问题，移动主机可以利用转交地址作为隧道的源把数据包返回给家乡代理；从家乡网络，它们再被发送到 Internet，这时的数据包显然出现在正确地方。其代价是更加迂回的路由。

还有另一个我们没有讨论的问题是安全性。当家乡代理接收到一条消息，请求它将所有给 Roberta 的数据包统统转发给某一个 IP 地址时，那么，除非它能够确认这个请求真的是 Roberta 发送的，并且肯定不是有人在模仿它，否则最好不要贸然就答应这个请求。基于密码学的认证协议可用来实现这样的身份验证任务，我们将在第 8 章中学习这样的协议。

IPv6 的移动性协议建立在 IPv4 的基础。上述方案遭到来自三角路由问题的困惑，即发送到移动主机的数据包在遥远的家乡代理那里拐了个弯。在 IPv6 中，针对该问题进行了路由优化，在最初的数据包遵循冤枉路径之后，移动主机和其他 IP 地址可以使用一条两者之间的直接路径。移动 IPv6 由 RFC 3775 定义。

Internet 还定义了另一种移动性。一些飞机上有内置的无线网络，乘客可以使用他们的笔记本电脑连接到 Internet。这架飞机有一个路由器通过无线链路连接到 Internet 的其余部分（难道你指望有线链路？）。所以，现在我们有一个飞行的路由器，这意味着整个网络在移动。网络移动性设计支持这种场景，而笔记本电脑却全然不知飞机在移动。至于它们所关注的这只是另一个网络。当然，某些笔记本电脑可能会使用移动 IP 来保持它们的家乡地址，而它们实际上在飞机上，所以我们有两个层次的移动性。IPv6 的网络移动由 RFC 3963 定义。

5.7 本章总结

网络层向传输层提供服务，它既可以基于虚电路，也可以基于数据报。在这两种情形下，它的主要任务是将数据包从源端路由到接收方。在数据报网络中，路由决策针对每一个数据包而作；在虚电路网络中，路由决策在建立虚电路时做出。

计算机网络用到了许多路由算法。泛洪是最简单的算法，它把数据包发送到所有的路径上。大多数算法寻找一条最短路径，并能自适应网络拓扑的变化。主要的算法是距离矢量算法和链路状态算法。大多数网络实际使用了其中的某一个算法。其他重要的路由话题包括大型网络的层次路由、移动主机的路由、广播路由、组播路由和选播路由。

网络很容易变得拥塞，从而增加了数据包延迟和丢失。网络设计者企图通过一系列手段来避免拥塞，其中包括设计具有足够容量的网络、选择未拥堵的路由、拒绝接受更多的流量、给源端发信号降低速度以及负载脱落。

处理拥塞控制的下一步是努力获得所承诺的服务质量。一些应用程序在乎吞吐量而另一些应用程序却更关心延迟和抖动。提供不同服务质量的方法包括流量整型、路由器上的资源预留以及准入控制。专门被设计用于提供良好服务质量的几种途径包括 IETF 的综合服务（包括 RSVP）和区分服务。

网络在很多方面都有所不同，所以，当多个网络相互连接时，问题就出现了。当不同

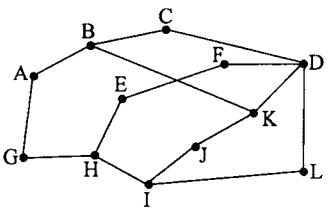
的网络具有不同的最大数据包长度时，可能需要分段。不同网络内部可运行不同的路由协议，但外部必须运行公共的路由协议。有的时候，隧道一个数据包穿越一个敌对网络能解决问题，但是，如果源网络和目标网络的类型不同，这种方法就会失败。

Internet 的网络层有丰富多样的协议。这些协议包括数据报协议 IP 和控制相关协议，比如 ICMP、ARP 和 DHCP。一个称为 MPLS 的面向连接协议携带 IP 数据包穿过某些网络。网络中使用的主要路由协议是 OSPF，穿越网络用的路由协议是 BGP。Internet 正快速消耗着 IP 地址，所以 IPv6 作为 IP 的新版本已经开发出来，并正在被如此之慢地部署着。

习 题

1. 请列举出两个适合使用面向连接服务的计算机应用实例，再列举出两个最好使用无连接服务的计算机应用实例。
2. 数据报网络将每个数据包当作独立的单位进行路由，路由过程彼此独立。虚电路网络不必采用这种方式，因为每个数据包都沿着一条预先确定的路由。试问，这是否意味着虚电路网络不需要具备将单个数据包从任意源端路由到任意接收方的能力呢？请解释你的答案。
3. 请给出 3 个在建立连接时可能需要协商的协议参数例子。
4. 假设所有的路由器和主机都正常工作，并且它们的软件也都没有错误。试问是否存在一个数据包被递交到错误目的地的可能性（无论可能性有多小）？
5. 请给出一个简单的启发式算法，找出一个网络中从指定源端到指定目标端之间的两条路径，要求这两条路径在失去任何一条通信线路的情况下都能够幸免于难（假设存在这样的两条路径）。可以认为路由器足够可靠，因此不必担心路由器崩溃的可能性。
6. 考虑图 5-12（a）中的网络。使用距离矢量路由算法，路由器 C 刚刚收到下列矢量：来自 B 的（5, 0, 8, 12, 6, 2）；来自 D 的（16, 12, 6, 0, 9, 10）；来自 E 的 （7, 6, 3, 9, 0, 4）。从 C 到 B、D 和 E 的链路成本分别为 6、3 和 5。请给出 C 的新路由表，包括使用的出境线路和成本。
7. 在一个有 50 个路由器的网络中，如果成本以 8 位数字表示，并且距离矢量每分钟交换两次，试问每条（全双工）线路有多少带宽被这个分布式路由算法吞噬掉？假设每个路由器都有三条线路连到其他路由器。
8. 在图 5-13 中，每一行上的两组 ACF 位布尔或（OR）的结果是 111。试问这仅仅是一种偶然情况，还是在所有情况下对于所有网络都成立？
9. 一个有 4800 台路由器的网络采用了层次路由。试问对于三层结构来说，应该选择多大的区域和簇才能将路由表的尺寸降低到最小？一个好的起点是假设这样的方案接近最优：有 k 个簇，每个簇有 k 个区域，每个区域有 k 个路由器。这意味着 k 大约是 4800 的立方根（约等于 16）。反复试验找出所有这三个参数在 16 附近的各种组合。
10. 在正文中提到当一台移动主机不在家乡网络时，发送至它本地 LAN 的数据包将被该 LAN 上的家乡代理所截获。针对一个 802.3 LAN 上的 IP 网络，试问家乡代理如何完成这样的截获工作？

11. 参照图 5-6 中的网络。试问若使用以下方法，从 B 发出的一次广播将生成多少个数据包？
    （a）逆向路径转发。
    （b）汇集树。
12. 考虑图 5-15（a）中的网络。想象在 F 和 G 之间加入一条新的线路，但是图 5-15（b）中的汇集树仍然不变。试问对于图 5-15（c）有什么变化？
13. 请计算下面网络中路由器 C 的组播生成树。组成员分布在路由器 A、B、C、D、E、F、I 和 K 上。



14. 假设图 5-20 中节点 B 刚刚重新启动，它的路由表中没有任何信息。现在它突然需要一条到达 H 的路由，于是它发送广播包，其 TTL 分别被设置为 1、2、3 等。试问它需要经过几轮广播之后才能找到一条路由？
15. 在内部采用虚电路的网络中，可能采用这样一种拥塞控制机制：路由器推迟确认收到的数据包，直到（1）它知道沿着虚电路的最后一次传输已经被成功接收，并且（2）它有一个空闲缓冲区。为了简单起见，假定路由器使用了停-等式协议，并且每条虚电路的每个方向都有一个专用的缓冲区。如果传输一个数据包（数据或者确认）需要 T 秒，在路径上有 n 台路由器，试问数据包被递交给目标主机的速率是多少？假设几乎没有传输错误，并且从主机到路由器之间连接的速度为无限快。
16. 一个数据报网络允许路由器在必要的时候丢弃数据包。路由器丢弃一个数据包的概率为 p。请考虑这样的情形：源主机连接到源路由器，源路由器连接到目标路由器，然后目标路由器连接到目标主机。如果任何一台路由器丢掉了一个数据包，则源主机最终会超时，然后再重试发送。如果主机至路由器以及路由器至路由器之间的线路都计为一跳，试问：
    （a）每次传输数据包的平均跳数是多少？
    （b）数据包的平均传输次数是多少？
    （c）每个接收到数据包所需的平均跳数？
17. 针对两个拥塞避免方法 ECN 和 RED，请给出它们之间的两个主要区别。
18. 流量整型采用了令牌桶方案。每 5 微秒一个新的令牌被放入桶中。每个令牌刚好用于一个短数据包，数据包包含 48 个字节数据。试问最大的可持续数据率是多少？
19. 在一个 6 Mbps 网络上有一台主机，其流量通过一个令牌桶进行整型。令牌桶的填充速率为 1 Mbps。初始时令牌桶被填满到容量 8 MB。试问该计算机能以 6 Mbps 的全速率传输多长时间？
20. 图 5-34 中的网络使用 RSVP 预留资源，主机 1 和主机 2 的组播树如图中所示。假设主机 3 请求一条带宽为 2 Mbps 的信道用于接收主机 1 的流，以及一条带宽为 1 Mbps 的信道用于接收主机 2 的流。同时，主机 4 请求一条带宽为 2 Mbps 的信道用于接收主机

1 的流；主机 5 请求一条带宽为 1 Mbps 的信道用于接收主机 2 的流。试问在路由器 A、B、C、E、H、K、J 和 L 上，总共需要为这些请求预留多少带宽？

21. 一个路由器可以每秒钟处理 200 万个数据包。提供给路由器的负载为每秒钟 150 万个数据包。如果从源端到接收方的路径上有 10 个路由器，试问路由器花在排队和服务上的时间为多少？
22. 假设网络采用区分服务模型。考虑使用加速转发服务的用户。试问是否可以保证加速型数据包比常规数据包的延迟更短？为什么是，或者为什么不是？
23. 假设主机 A 与路由器 R1 连接，R1 又与另一个路由器 R2 连接，R2 与主机 B 连接。假定一个要发给主机 B 的 TCP 消息被传递给主机 A 的 IP 代码，其中包含了 900 个字节的数据和 20 个字节的 TCP 头。请写出在三条链路上传输的每个数据包中 IP 头部的 Total length、Identification、DF、MF 和 Fragment offset 字段。假定链路 A-R1 链路可以支持的最大帧长为 1024 字节，其中包括 14 字节的帧头；链路 R1-R2 可以支持的最大帧长为 512 字节，其中包括 8 字节的帧头；链路 R2-B 可以支持的最大帧长为 512 字节，其中包括 12 字节的帧头。
24. 一个路由器往外发送大量的 IP 数据包，这些数据包的总长度（数据+头）为 1024 字节。假设这些数据包的生存期为 10 秒，试问路由器运行的最大线速度达到多少才不会发生 IP 数据报的 ID 编号空间重绕的危险？
25. 一个 IP 数据报使用了严格源路由（Strict source routing）选项，现在它必须被分段。你认为该选项应该被复制到每个段中，还是只需放到第一个段中就足够了？请解释你的答案。
26. 假定最初的时候 B 类地址的网络部分不是 16 位，而是 20 位。试问将有多少个 B 类网络？
27. 一个 IP 地址的十六进制表示为 C22F1582，请将它转换成点分十进制表示法。
28. Internet 上一个网络的子网掩码为 255.255.240.0。试问它最多能够容纳多少台主机？
29. 尽管 IP 地址特定于一个网络，以太网地址却不是。你能想到一个好理由，说明为什么以太网地址做不到吗？
30. 从 198.16.0.0 开始有大量连续的 IP 地址可以使用。假设 4 个组织 A、B、C 和 D 按照顺序依次申请 4000、2000、4000 和 8000 个地址。对于每一个申请，请用 w.x.y.z/s 的形式写出所分配的第一个 IP 地址、最后一个 IP 地址以及掩码。
31. 一个路由器刚刚接收到以下新的 IP 地址：57.6.96.0/21、57.6.104.0/21、57.6.112.0/21 和 57.6.120.0/21。如果所有这些地址都使用同一条出境线路，试问它们可以被聚合吗？如果可以，它们被聚合到哪个地址上？如果不可以，请问为什么？
32. 从 29.18.0.0 到 29.18.128.255 的一组 IP 地址已经被聚合到 29.18.0.0/17。然而，这里有一个空闲地址块，即从 29.18.60.0 到 29.18.63.255 之间的 1024 个地址还没有被分配。现在这块空闲地址突然要被分配给一台使用不同出境线路的主机。试问是否有必要将聚合地址分割成几块，然后把新的地址块加入到路由表中，再来看是否可以重新聚合？如果没有必要这样做，请问该怎么办呢？
33. 一个路由器的路由表中有如下的表项：

地址/掩码	下一跳
135.46.56.0/22	Interface 0
135.46.60.0/22	Interface 1
192.53.40.0/23	Router 1
default	Router 2

对于下列IP地址，如果到达的数据包带有这些地址，试问路由器该如何处理？

（a）135.46.63.10

（b）135.46.57.14

（c）135.46.52.2

（d）192.53.40.7

（e）192.53.56.7

34. 许多公司采取这样的策略：通过两个或者多个路由器将公司连接到Internet。这种冗余度保证了其中一个路由器停机时网络还能使用。试问采用NAT策略之后，仍然能正常工作吗？请解释你的答案。
35. 你刚刚向一个朋友解释了ARP协议。当你解释完之后，他说："我明白了，ARP给网络层提供了一项服务，所以它是数据链路层的一部分。"你该如何向他解释呢？
36. 请给出一种在目标主机上重组IP分段的方法。
37. 大多数IP数据报重组算法有一个计时器，以免丢失的段永远占用重组缓冲区。假设一个数据报被分为4个段。前3段到达目的地，但最后一个被延迟了。最终该计时器超时，接收方内存中的3个段被丢弃。过了一会儿，最后一段姗姗来迟。试问应该用它做什么呢？
38. 在IP中，校验和仅仅覆盖了头，而没有包括数据部分。你认为选择这种设计方案的理由是什么？
39. 有一个人生活在Boston，现在她带着自己的笔记本电脑去Minneapolis旅游。让她惊讶的是，在Minneapolis目的地的局域网是一个无线IP局域网，所以她根本不需要插网线。试问，她是否仍然需要通过家乡代理和外部代理这一整套过程才能正确地接收电子邮件或者其他的流量？
40. IPv6使用16个字节的地址。如果每隔1 ps就分配掉一百万个地址，试问整个地址空间可以持续分配多久？
41. IPv4头中的Protocol字段并没有出现在IPv6的固定头中。试问为什么？
42. 当IPv6协议被引入时，ARP协议需要作相应的改变吗？如果需要，这种改变是概念性的还是技术性的？
43. 编写一个程序来模拟泛洪路由算法。每个数据包应该包含一个计数器，在每一跳上该计数器值减一。当计数器值到达0时，该数据包就被丢弃。时间是离散的，每条线路在每个间隔中只处理一个数据包。请完成该程序的三个版本：所有的线路都被泛洪、除了入境线路外其他所有的线路都被泛洪、只有最佳的k条线路（静态选择）才被泛洪。按照延迟和所用带宽比较泛洪算法和确定性路由算法（k=1）的性能。
44. 编写一个程序来模拟使用离散时间的计算机网络。在每个时间间隔中，每个路由器队列中的第一个数据包向前走一跳。每个路由器只有有限数量的缓冲区。如果一个数据

包到来时，路由器没有可用的缓冲区空间，那么该数据包将被丢弃，并且不再重传。同时，另有一个端到端协议，它完全支持超时和确认数据包。最终源路由器会重新生成丢弃的数据包。请给出网络吞吐量与端到端超时间隔之间的函数关系，错误率是一个参数。

45. 编写一个函数来完成IP路由器的转发过程。该过程有一个参数，即IP地址。它还要访问一张全局表，表项由三元组构成。每个三元组包含三个整数：IP地址、子网掩码和所用的输出线路。该函数利用CIDR技术查询该表，然后返回作为参数给出的IP地址所对应的输出线路值。

46. 使用traceroute（UNIX）和tracert（Windows）程序跟踪从你的计算机到其他国家不同大学的路由。列出你发现的跨洋链路。可以尝试的一些站点是：

www.berkeley.edu	（California）
www.mit.edu	（Massachusetts）
www.vu.nl	（Amsterdam）
www.ucl.ac.uk	（London）
www.usyd.edu.au	（Sydney）
www.u-tokyo.ac.jp	（Tokyo）
www.uct.ac.za	（Cape Town）

第6章 传 输 层

传输层与网络层一起构成了网络协议层次的核心。网络层使用数据报或虚电路技术为端到端通信提供了数据包交付服务。传输层架构在网络层提供的服务之上，把数据传递服务从两台计算机之间扩展到了两台计算机上的进程之间，并且服务所需的可靠性程度独立于当前使用的物理网络。传输层为应用层使用网络提供了抽象的模式。如果没有传输层，分层协议的整个概念将毫无意义。在本章，我们将详细学习传输层，包括它的服务和 API 设计的选择，其中涉及可靠性、连接和拥塞控制，协议（比如 TCP 和 UDP）和性能等问题的解决。

6.1 传 输 服 务

在下面的章节中，我们将简要介绍传输层服务。我们要考查传输层向应用层提供什么样的服务。为了使传输服务的问题更具体，我们将考查两套传输层原语。先看一个简单的（假想的）原语，以便了解传输层的基本性质，然后再看 Internet 的常用接口。

6.1.1 提供给上层的服务

传输层的最终目标是向它的用户提供高效的、可靠的和成本有效的数据传输服务，它的用户通常是应用层的进程。为了实现这个目标，传输层需要充分利用网络层提供给它的服务。在传输层内，完成这项工作的硬件和/或软件称为**传输实体**（transport entity）。传输实体可以实现在主机的不同位置，可能在操作系统内核，或者以一个链接库的形式绑定到网络应用中，或者以一个独立的用户进程运行，甚至可以实现在网络接口卡上。前两种实现方式在 Internet 上最常见。网络层、传输层和应用层之间的（逻辑）关系如图 6-1 所示。

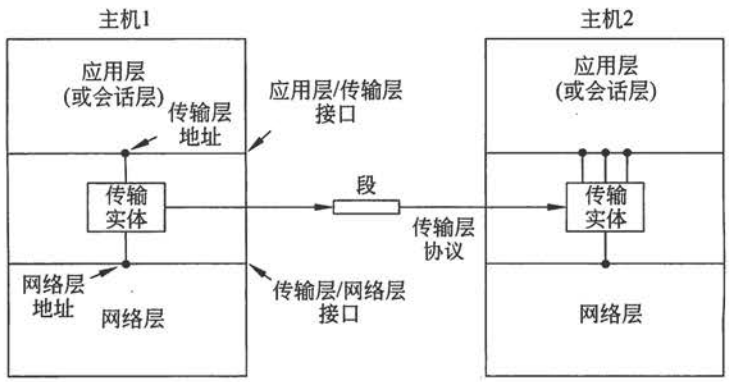


图 6-1 网络层、传输层和应用层

与网络层提供面向连接和无连接两种服务一样，传输层的服务类型也分为两种。面向

连接的传输服务在许多方面与面向连接的网络服务类似，两者的连接都要经历 3 个阶段：连接建立、数据传输和连接释放。在这两层上，寻址和流量控制非常相像。另外，无连接的传输服务与无连接的网络服务也极为相似。然而请注意，在一个面向连接的网络服务之上提供无连接传输服务可能很困难，因为为了发送单个数据包要建立一个连接、发送完毕后还要立即拆除这个连接，效率实在是太低了。

于是，一个很显然的问题出现了：既然传输层服务与网络层服务如此相似，为什么还要设立两个独立的层？为什么一层不够？问题的答案有点微妙，但非常关键。传输层的代码完全运行在用户的机器上，但是网络层代码主要运行在由运营商操作的路由器上（至少对于广域网是如此）。如果网络层提供的服务不够用，怎么办？如果它频繁地丢失数据包该怎么办？如果路由器时常崩溃又该怎么办？

问题发生了，这就是答案。用户对网络层没有真正的控制权，因为他们不拥有路由器，所以不能用更好的路由器或者在数据链路层上用更好的错误处理机制来解决服务太差的问题。唯一的可能是在网络层之上再加一层，由该层来提高网络的服务质量。如果在一个无连接网络中，数据包被丢失或者发生错位，则传输实体可以检测到问题所在，并通过重传来弥补这种错误。如果在一个面向连接网络中，传输实体在执行一个漫长传输任务期间，突然接到通知说它的网络层连接已经被意外终止，而且也不知道当前正在传输的那些数据到底怎么样，那么，该传输实体可以与远程的传输实体建立一条新的网络层连接。利用新建立的连接，它可以向对等实体询问哪些数据已经到达，哪些数据还没有到达，然后从中断的地方开始继续向对方发送数据。

本质上，由于传输层的存在，使得传输服务有可能比网络服务更加可靠。而且，传输服务原语可以通过调用库程序来实现，从而使得这些原语独立于网络服务原语。不同网络上的网络服务原语可能有很大的差别（比如，无连接以太网服务可能完全不同于面向连接的 WiMAX 服务）。将网络服务隐藏在一组传输服务原语的背后，带来的好处是，一旦改变了网络服务，只需要替换一组库程序即可；新的库程序使用了不同的底层网络服务，但是实现了同样的传输服务原语。

值得庆幸的是，正是有了传输层，应用程序员才可以按照一组标准的原语来编写代码，并且程序可以运行在各种各样的网络上；他们根本无须处理不同的网络接口，也不用担心传输的可靠性。如果所有实际的网络都完美无缺，具有相同的服务原语，并保证不会发生变化，那么传输层或许就不再需要。然而，在现实世界中，传输层承担了把上层与技术、设计和各种缺陷隔离的关键作用。

基于这个原因，许多人习惯于将网络分成两部分：第 1 层至第 4 层为一部分，第 4 层之上为另一部分。下面的 4 层可以看作是**传输服务的提供者**（transport service provider），而上面的层次则可视为**传输服务的用户**（transport service user）。这种服务提供者与服务用户的区分对于协议层的设计有重要的影响，同时也把传输层放到了一个关键位置，因为它构成了可靠数据传输服务的提供者和用户两者之间的主要边界。这就是应用层能看到的传输层。

6.1.2 传输服务原语

为了允许用户访问传输服务，传输层必须为应用程序提供一些操作，也就是说，提供

一个传输服务接口。每个传输服务都有它自己的接口。在本小节中，我们首先介绍一个简单的（假想的）传输服务以及相应的接口，通过它了解传输服务的本质所在。在下一小节我们将介绍一个实际的例子。

传输服务类似于网络服务，但是两者之间有一些重要的区别。最主要的区别在于网络服务毫不掩盖地按照实际网络提供的服务来建立模型。实际网络可能会丢失数据包，所以网络服务一般来说是不可靠的。

与此相反，面向连接的传输服务是可靠的。当然，实际网络并非没有错误，但是，这恰好是传输层的目标——在不可靠的网络之上提供可靠的服务。

作为一个例子，请考虑一个 UNIX 系统中通过管道（或者任何其他进程之间的通信设施）连接起来的两个进程。假设两者之间的连接是十分完美。它们并不想了解有关确认、数据包丢失、拥塞以及诸如此类的情况。它们所要的是一个 100%可靠的连接。进程 A 把数据放进管道的一端，进程 B 可以从管道的另一端将数据取出来。这就是面向连接传输服务的真正含义所在——将网络服务的各种缺陷隐藏起来，因此用户进程只要假设它们两者之间存在一个无错的比特流，即使双方位于不同的机器上时也一样。

另一方面，传输层也可以提供不可靠（数据报）服务。然而，相对来说关于这种服务除了“它是数据报”外并没有太多内容可说，所以本章我们将注意力主要集中在面向连接的传输服务上。不过，有一些应用建立在无连接传输服务上，比如客户机-服务器计算和流式多媒体应用，所以，我们在后面还是会稍微介绍这种服务。

网络服务和传输服务之间的第二个区别在于它们的服务对象不同。网络服务仅仅被传输实体使用。通常用户不会编写自己的传输实体，因此，很少有用户或者程序能看到裸露的网络服务。相反，许多程序（和程序员）可以看到传输原语。因此，传输服务的使用必须非常方便、容易。

为了了解传输服务的基本面貌，请考虑图 6-2 中列出的 5 个原语。这个传输接口是真正的赤裸裸，但它给出了一个面向连接的传输接口应该完成的基本工作。它允许应用程序建立并使用连接，用完之后再释放连接，对于许多应用来说这已经足够了。

原语	发出的包	含义
LISTEN	(无)	阻塞，直到某个进程试图与之连接
CONNECT	CONNECTION REQ	主动尝试建立一个连接
SEND	DATA	发送信息
RECEIVE	(无)	阻塞，直到到达一个DATA 包
DISCONNECT	DISCONNECTION REQ	请求释放连接

图 6-2 一个简单传输服务的原语

为了看清楚这些原语的可能用法，请考虑一个应用，它有一个服务器和多个远程客户。首先，服务器执行 LISTEN 原语，一般的做法是调用一个库过程，由它执行阻塞该服务器的系统调用，直到有客户请求连接。当一个客户希望与该服务器进行通话时，它就执行 CONNECT 原语。传输实体执行该原语并阻塞调用方，然后给服务器发送一个包。封装在该包有效载荷中的是一条发送给服务器传输实体的传输层消息。

现在我们快速说明术语的用法。由于缺乏更好的表达，我们将使用术语**段**（segment）来表示传输实体之间发送的消息。一些老协议使用了更加笨拙的名称——**传输协议数据单**

元（TPDU，Transport Protocol Data Unit）。这个术语现在已不再使用，但仍可在一些老文章和书籍中看到该用法。

因此，段（传输层之间交换的单元）被包裹在数据包（网络层之间交换的单元）中，而数据包则被包含在帧（数据链路层之间交换的单元）中。当一帧到达时，数据链路层对帧头进行处理，如果帧目标地址与本地传递地址匹配，则把帧的有效载荷字段中的内容传递给网络实体。网络实体对数据包头进行类似处理，然后把数据包的有效载荷字段内容向上传递给传输实体。这种嵌套关系如图 6-3 所示。

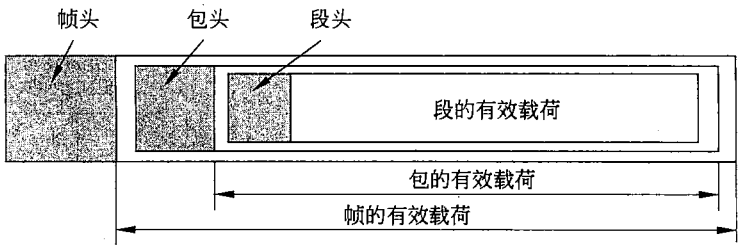


图 6-3 段、包和帧的嵌套关系.

再回到我们的客户机-服务器例子，客户的 CONNECT 调用导致传输实体发送一个 CONNECTION REQUEST 段给服务器。当该段到达服务器时，传输实体检查服务器是否阻塞在 LISTEN 状态（即服务器对处理请求感兴趣）。如果是，则解除服务器的阻塞，并给客户送回一个 CONNECTION ACCEPTED 段。当该段返回到客户机时，客户机的阻塞也被解除，于是连接被建立了起来。

现在双方可以通过 SEND 和 RECEIVE 原语交换数据。在最简单的形式中，任何一方都可以执行（阻塞的）RECEIVE 原语，等待另一方执行 SEND 原语。当段到来时，接收端被解除阻塞；然后它可以对这个段进行处理，并发送一个应答。只要双方保持应该谁发送数据的次序，这种方案就可以工作得很好。

请注意，在传输层上，即使一个非常简单的单向数据交换过程也比网络层的交换过程复杂得多。发送的每个数据包（最终）都要被确认。携带控制段的数据包也要被确认，无论是隐式方式还是显式方式。这些确认由使用网络层协议的传输实体来管理，并且它们对于传输用户是不可见的。类似地，传输实体需要关心计时器和重传。这些机制对于传输用户全部是透明的。对传输用户而言，连接就是一个可靠的比特管道：一个用户在管道一端将比特塞进去，这些比特就会神奇地出现在管道的另一端。正是这种隐藏复杂性的能力才使得分层协议成为如此强大的工具。

当不再需要一个连接时必须将它释放，以便释放两个传输实体内部的表空间。中断连接有两种方式：非对称的和对称的。在非对称方式中，任何一方都可以发出 DISCONNECT 原语，从而驱使它的传输实体将一个 DISCONNECT 段发送给远程的传输实体。当该段到达另一端时，连接就被释放了。

在对称方式中，连接的两个方向彼此独立，因此需要单独关闭每个方向。当一方执行了 DISCONNECT，这意味着它没有更多数据需要发送，但是它仍然愿意接受对方发送过来的数据。在这种模型下，只有当双方都执行了 DISCONNECT 原语，一个连接才算真正被释放。

图 6-4 给出了使用这些简单原语建立和释放连接的状态图。每个状态的迁移都是由某种事件触发的，这些事件可能是本地的传输用户执行了一个原语，或者是接收到了一个数据包。为了简化起见，我们假设每个段单独确认；还假设采用了对称的连接释放模型，并且由客户先释放连接。请注意，这种模型相当不成熟，后面我们在描述 TCP 如何工作时会考察更实际的模型。

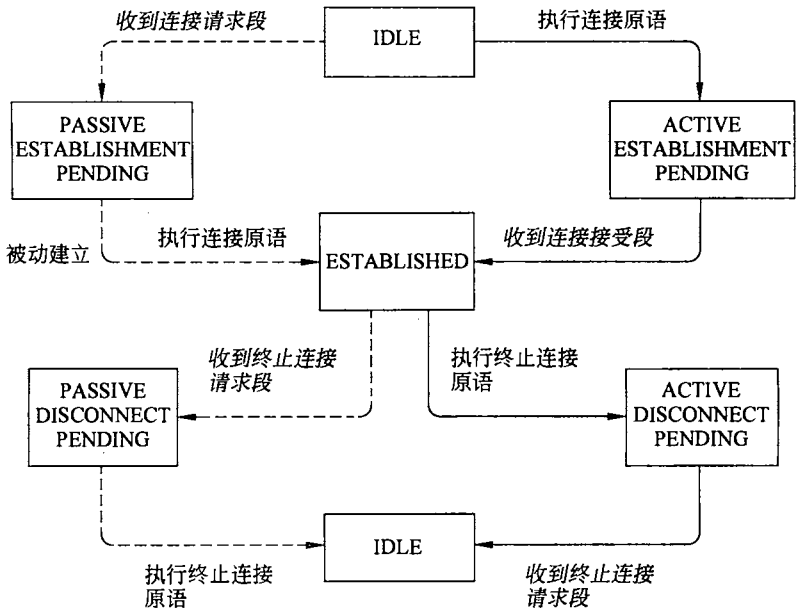


图 6-4 一个简单连接管理方案的状态图

斜体标记的状态转移是由到达的包引起的。实线表示客户的状态序列，虚线表示服务器的状态序列

6.1.3 Berkeley 套接字

现在让我们简要地考查另一组传输原语，即 TCP 所用的**套接字**（socket）原语。作为 Berkeley UNIX4.2 BSD 软件的一部分。套接字首次发布在 1983 年，这些原语很快得到流行，现在已被许多操作系统广泛应用于 Internet 程序设计中，尤其是基于 UNIX 的系统，Windows 系统也有一个套接字风格的 API，称为“winsock”。

图 6-5 列出了一些原语。粗略地说，这些原语遵循我们第一个例子的模型，但提供了更多的功能和灵活性。我们在这里不去探究相应的段，把讨论留到后面。

原语	含义
SOCKET	创建一个新通信端点
BIND	将套接字与一个本地地址关联
LISTEN	声明愿意接受连接；给出队列长度
ACCEPT	被动创建一个入境连接
CONNECT	主动创建一个连接
SEND	通过连接发送一些数据
RECEIVE	从连接上接收一些数据
CLOSE	释放连接

图 6-5 TCP 的套接字原语

表中列出的前4个原语由服务器按照顺序执行。SOCKET原语创建一个新的端点（end point），并且在传输实体中为它分配相应的表空间。此调用的参数说明了采用的地址格式、所需的服务类型（比如可靠的字节流），以及所用的协议。SOCKET调用成功则返回一个普通的文件描述符，供后续的调用使用，SOCKET调用与对文件实施的OPEN调用工作方式一样。

新近创建的套接字没有网络地址。通过BIND原语可以为套接字分配地址。一旦服务器已经将一个地址绑定到一个套接字，则远程客户就能够与它建立连接。之所以不让SOCKET调用直接创建一个地址是因为有些进程对于它们的地址比较在意（比如，它们多年来一直使用同样的地址，所以每个人都知道它们的地址），而其他的进程并不在乎。

接下来是LISTEN调用，它为入境呼叫分配队列空间，以便在多个客户同时发起连接请求时，将这些入境的连接请求排入队列依次处理。与我们第一个例子中的LISTEN不同的是，套接字模型中的LISTEN并不是一个阻塞调用。

为了阻塞自己以便等待入境连接的到来，服务器执行ACCEPT原语。当一个请求连接的段到达时，传输实体创建一个新的套接字并返回一个与其关联的文件描述符，这个新套接字与原来的套接字具有同样的属性；然后服务器可以派生一个进程或者线程来处理这个新套接字上的连接，而服务器自身又回到原来的套接字上等待下一个连接请求的到来。ACCEPT返回一个文件描述符，服务器可以按照标准的方式对它进行读或者写操作，就像访问文件一样。

现在我们来看客户端的情形。同样地，这里首先也必须使用SOCKET原语创建一个套接字，但是因为客户端使用什么地址对服务器而言无所谓，所以客户端不需要调用BIND原语。CONNECT原语阻塞调用方，并主动发起建立连接过程。当CONNECT调用完成（即接收到服务器发送过来的确认段），客户进程被解除阻塞，于是连接就被建立起来。现在双方都可以使用SEND或者RECV，在新建的全双工连接上发送或者接收数据。如果SEND和RECV调用不要求特殊选项的话，服务器或者客户也可以使用标准的UNIX系统调用READ和WRITE。

在套接字模型中，连接的释放是对称的。当双方都执行了CLOSE原语之后，连接就被释放了。

套接字盛极一时，非常流行，并且已经成为抽象于应用层的传输服务的事实上标准。套接字API通常与TCP协议结合向用户提供了一种称为**可靠字节流**（reliable byte stream）的面向连接的服务，这是我们描述过的简单可靠比特管道。然而，其他协议也可以被用来实施这项服务，使用相同的API，并且对传输服务用户来说应该是相同的。

套接字API的强大体现在应用程序可通过它使用其他的传输服务。举例来说，套接字可以与无连接传输服务结合使用。在这种情况下，CONNECT设置远程传输对等实体的地址，SEND和RECEIVE发送数据报给远程对等实体和接收来自远程对等实体的数据报（通常还使用一组扩展调用，例如，SENDTO和RECEIVEFROM强调报文，并且不限制应用只针对单个传输对等实体）。套接字也可与提供消息流而不是字节流的传输协议一起工作，而且可以进行拥塞控制，也可以不进行拥塞控制。例如，**数据报拥塞控制协议**（DCCP，Datagram Congestion Controlled Protocol）是一个带有拥塞控制机制的UDP版本（Kohler等，2006）。传输层用户必须理解他们得到的是什么服务。

然而，套接字不太可能是传输接口上的最后一个字。例如，应用程序通常有一组相关的流，比如Web浏览器，它向同一台服务器请求多个对象。有了套接字，最自然契合应用程序的是为每个对象设立一个流。这种结构意味着拥塞控制可应用到每个数据流，而不是不甚理想地作用于一个组。这样的处理方式把管理包袱抛给了应用程序。为此必须制定新的协议和接口，使得应用程序更加有效并简单地管理一组相关流。其中的两个例子，分别是由RFC 4960定义的**流控制传输协议**（SCTP，Stream Control Transmission Protocol）和**结构化流传输**（SST，Structured Stream Transport）（Ford，2007）。这些协议必须略微修改套接字API才能获得把相关数据流组成一组的好处，它们也支持混合功能，比如把面向连接的流量和无连接流量混合在一起，甚至可以组合多条网络路径组合。时间会告诉我们它们能否获得成功。

6.1.4 套接字编程实例：Internet文件服务器

作为一个真实套接字调用的基本事实例子，考虑图6-6中所示的客户端和服务器代码。这里我们有一个非常原始的Internet文件服务器，和一个使用该服务器的客户实例。这份代码有许多限制（后面将会提到），但原则上，服务器代码可以编译成二进制代码，并且可运行在任何连接到Internet的UNIX系统上。客户代码也可以编译，并且可运行在Internet任何地方的其他UNIX机器上。客户代码在执行的时候需要正确的参数，以便获取服务器机器上的任何文件，这些文件是服务器本身能够访问的。文件被送到标准输出，当然，可以将标准输出重定向到某一个文件或者一个管道。

```
/* A client program that can request a file from the server program.
 * The server responds by sending the whole file.
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345       /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096           /* block transfer size */


int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                   /* buffer for incoming file */
  struct hostent *h;                    /* info about server */
  struct sockaddr_in channel;           /* holds IP address */

  if (argc != 3) fatal("Usage: client server-name file-name");
  h = gethostbyname(argv[1]);           /* look up host's IP address */
  if (!h) fatal("gethostbyname failed");
```

图6-6 使用套接字的客户端代码

```
    s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s <0) fatal("socket");
    memset(&channel, 0, sizeof(channel));
    channel.sin_family= AF_INET;
    memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
    channel.sin_port= htons(SERVER_PORT);

    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");
  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
    bytes = read(s, buf, BUF_SIZE);     /* read from socket */
    if (bytes <= 0) exit(0);            /* check for end of file */
    write(1, buf, bytes);               /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}

#include <sys/types.h>                  /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345               /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                   /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                   /* buffer for outgoing file */
  struct sockaddr_in channel;           /* holds IP address */
  /* Build address structure to bind to socket. */
  memset(&channel, 0, sizeof(channel));         /* zero channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);
```

图 6-6 （续）

```
    /* Passive open. Wait for connection. */
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);     /* create socket */
    if (s < 0) fatal("socket failed");
    setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));
    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE_SIZE);                         /* specify queue size */
    if (l < 0) fatal("listen failed");

    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
      sa = accept(s, 0, 0);                            /* block for connection request */
      if (sa < 0) fatal("accept failed");

      read(sa, buf, BUF_SIZE);                         /* read file name from socket */
    /* Get and return the file. */
    fd = open(buf, O_RDONLY);                          /* open the file to be sent back */
    if (fd < 0) fatal("open failed");

    while (1) {
      bytes = read(fd, buf, BUF_SIZE);                 /* read from file */
      if (bytes <= 0) break;                           /* check for end of file */
      write(sa, buf, bytes);                           /* write bytes to socket */
    }
    close(fd);                                         /* close file */
    close(sa);                                         /* close connection */
  }
}
```

图 6-6 （续）

我们首先来看服务器的代码。在开始处它包含一些标准的头文件，其中最后 3 个头文件包含了一些主要的与 Internet 有关的定义和数据结构。接下来是一个宏定义，将 SERVER_PORT 定义成 12 345。这个数值是任意选取的。1024～65 535 之间的任何一个数值都可以正确地工作，只要不被其他进程使用；小于 1024 的数字为特权用户保留。

服务器代码中接下来两行定义了两个用到的常数。第一个常数决定了在文件传输过程中数据块的大小。第二个常数决定了允许多少个连接请求在排队等待处理，一旦连接请求到达这个数目以后，后续到达的连接请求将被丢弃。

在声明了局部变量之后，服务器代码开始执行。首先它对一个用来存放服务器 IP 地址的数据结构进行初始化。该数据结构很快被绑定到服务器的套接字中。memset 调用将这个数据结构初始化为全 0。紧随其后的 3 条赋值语句分别填充了该数据结构的 3 个字段。其中最后一个字段包含了服务器的端口。函数 htonl 和 htons 是必要的，它们将参数中的值转换成一种标准格式，所以，该服务器的代码既可以运行在 little-endian 字节序的机器（比如

Intel x86）上，也可以运行在 big-endian 字节序的机器（比如 SPARC）上。我们这里并不关心它们的确切语义。

接下来为服务器创建一个套接字，并且检查是否出错（由 s<0 标示）。在产品版本的服务器代码中，错误消息可能包含更多的说明信息。调用 setsockopt 是必要的，它允许这个端口可以被重复使用，以便服务器能够永久地运行下去，处理一个又一个请求。现在，要绑定 IP 地址到套接字中，然后检查调用 bind 是否成功。初始化过程中的最后一步是调用 listen，这样就宣告了本服务器愿意接受入境呼叫，并告诉系统当服务器在处理当前请求时如果有新的请求到来，就将新请求挂起，挂起等待处理的请求个数可以多达 QUEUE_SIZE。如果队列满了之后又有新的请求到来，则直接将后来的请求丢弃即可。

到了这点，服务器便进入了它的主循环，这是一个永不退出的循环。终止服务器的唯一做法是从外部将服务器进程杀死。accept 调用阻塞服务器，直到某个客户试图与它建立连接。如果调用 accept 成功，则它返回一个套接字描述符，利用该描述符可以进行读写操作，就好像利用文件描述符从管道读写数据一样。然而，与单向管道不同的是，套接字是双向的，所以，既可以用 sa（接受的套接字）从连接中读取数据，也可以用它往连接上写数据；而一个管道描述符可以用来读管道中的数据，也可以往管道内写数据，但不能同时读写。

当连接被建立之后，服务器从连接中读取文件名。如果文件名不可用，则服务器被阻塞住，等待文件名的到来。服务器获得了文件名之后，它打开该文件，然后进入一个循环：交替地从文件中读取数据块并且将数据块写到套接字中，这个过程一直持续到整个文件被复制完毕为止。然后，服务器关闭文件和连接，并等待下一个连接的到来。它无限地重复这个循环。

现在让我们来看客户代码。为了理解客户代码的工作过程，首先有必要理解客户被调用的方式。假设客户程序名为 client，那么，一个典型的调用如下：

```
client  flits.cs.vu.nl  /usr/tom/filename >f
```

只有当服务器已经在 flits.cs.vu.nl 机器上运行，该机器上确实存在/use/tom/filename 文件，并且服务器对该文件有读访问权限时，上面这个 client 调用才能生效。如果调用成功，那么，服务器上的文件将通过 Internet 被传递给客户端，并且写到客户端的本地文件 f 中，然后客户程序退出。由于在一次传输之后，服务器仍然在运行，所以客户可以再次启动，以获取其他的文件。

客户代码首先是一些包含语句和声明。开始执行时它先判断用户调用自己是否有正确的参数个数（这里 argc=3 代表了程序名加上两个参数）。请注意，argv[1]包含了服务器的名字（比如 flits.cs.vu.nl），通过 gethostbyname 把它转换为一个 IP 地址。这里的函数 gethostbyname 使用了 DNS 来查询机器名字对应的地址。我们将在第 7 章学习 DNS。

接下来创建一个套接字并执行初始化。之后，客户调用 connect 函数，企图与服务器建立一个 TCP 连接。如果在指定名字的机器上，服务器已经启动运行，并且被绑定到了 SERVER_PORT 端口，那么在服务器当前空闲，或者在 listen 队列中还有空间时，连接（最终）将被建立起来。客户利用该连接可以将文件的名字发送过去，做法很简单，只要在套接字上执行写操作即可。发送的字节数是名字长度加 1，必须在名字后面加一个字节 *0* 并

发送过去，服务器才能知道文件名在哪里结束。

现在客户进入了一个循环，它将文件从套接字中逐个数据块地读出来，再复制到标准输出上。完成这个过程之后，它就退出。

过程 fatal 打印出一条错误消息，然后退出程序。服务器代码也需要同样的过程，但是考虑到页面空间紧张，所以它被省略掉了。由于客户和服务器是单独编译的，而且通常运行在不同的计算机上，所以它们并不共享同一份 fatal 代码。

这两个程序（以及与本书相关的其他资料）可以从本书的网站获取：

```
http://www.pearsonhighered.com/tanenbaum
```

这台服务器在服务器领域不是最后的定论，纯粹为了记录而已。它的错误检查能力微薄，它的错误报告也很一般。由于它严格按顺序处理所有的请求（因为它只有一个线程），因此它的性能很差。它显然从来没有听说过安全性，并且使用裸露的 UNIX 系统调用并且没采用独立于平台的方式。同时它还做了一些在技术上是非法的假设，比如假设文件名一定适合缓冲区，而且文件传输以原子方式执行。尽管有这些缺点，但它仍然是一个可运行的 Internet 文件服务器。在本章的习题中，邀请读者对其进行改进。套接字编程的更多信息，请参阅（Donahoo，2008）和（Calvert，2009）。

6.2 传输协议的要素

传输服务由传输协议实现，两个传输实体之间的通信必须使用传输协议。传输协议在有些方面类似于我们在第 3 章中学习过的数据链路协议。这两种协议都要处理错误控制、顺序性和流量控制以及其他一些问题。

然而，两者之间也存在着重大的差别。这些差别是因为这两种协议的运行环境不同而造成的。如图 6-7 所示。在数据链路层，两台路由器通过一条有线或者无线物理信道直接进行通信；而在传输层，该物理信道被整个网络所替代。这种环境差异对于协议设计有很大的影响。

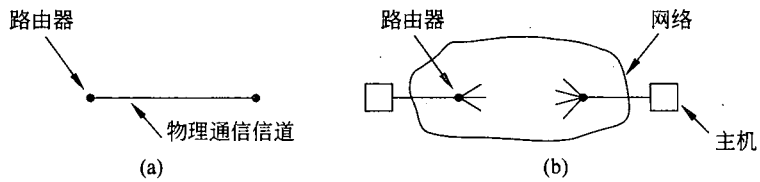


图 6-7
（a）数据链路层环境；（b）传输层环境

首先，在点到点链路上，无论是电缆或者光纤，路由器不必指定它要与哪一台路由器进行通话——每条出境线路直接通向一台特定的路由器。而在传输层，必须显式地指定接收方的地址。

其次，图 6-7（a）中，在一条线路上建立一个连接的过程非常简单：另一端总是在那里（除非它崩溃了才不在那里）。两边都不需要做很多事情。即使在无线链路上，建立过程也没有多大的不同，只要发出的消息足够到达所有的其他接收方。如果因发生错误而消息

没有被确认，可以再次重发。而在传输层中，初始的连接建立过程非常复杂，正如我们将会看到的那样。

数据链路层和传输层之间的另一个（非常恼人的）差别是，网络存在着潜在的存储容量。当路由器发送一帧到一条链路上后，该帧可能到达对方也可能丢失，但是它不可能先蹦跶一会儿，再躲到远处一个角落中，然后在其他数据包发送出去很久后突然又冒了出来。如果网络使用数据报技术，即网络内部的路由是独立进行的，那么就存在一个不可忽略的概率：一个包可能选取了风景优美的路线，一路观光姗姗来迟到达目的地，这将扰乱预期的接收顺序，甚至它的重复数据包都已经到达目的地它还没到呢。网络具有的这种延迟和重复数据包的特性所产生的后果有时是灾难性的，因此要求使用特殊的协议，以便正确地传输信息。

数据链路层和传输层之间最后一个差别是程度上的差别，而并非类别上的差别。这两层都需要缓冲和流量控制，但是，由于传输层上存在着大量并且数量可变的连接，而且由于连接之间的相互竞争造成连接的可用带宽上下波动，因此需要一种不同于数据链路层使用的方法。在第 3 章中讨论的有些协议为每条线路分配了固定数量的缓冲区，所以，当一帧到达时，总是有缓冲区可用。在传输层中，由于必须要管理大量的连接并且每个连接获得的带宽又是可变的，因此，为每条线路分配多个缓冲区的思路不再有吸引力。在本节中，我们将讨论所有这些重要的以及其他的问题。

6.2.1 寻址

当一个应用（比如一个用户）进程希望与另一个远程应用进程建立连接时，它必须指定要连接到哪个应用进程上（无连接的传输也有同样的问题：消息发送给谁？）。通常使用的方法是为那些能够监听连接请求的进程定义相应的传输地址。在 Internet 中，这些端点称为**端口**（port）。我们将使用通用术语**传输服务访问点**（TSAP，Transport Service Access Point）来表示传输层的一个特殊端点。网络层上的类似端点（即网络层地址）毫不奇怪地称为**网络服务访问点**（NSAP，Network Service Access Point）。IP 地址是 NSAP 的实例。

图 6-8 显示了 NSAP、TSAP 和传输连接之间的关系。应用进程（包括客户和服务器）可以将自己关联到一个本地 TSAP 上，以便与一个远程 TSAP 建立连接。这些连接运行在每台主机的 NSAP 之上，如图所示。在有些网络中，每台计算机只有一个 NSAP，但是可能有多个传输端点共享此 NSAP，因此需要某种方法来区分这些传输端点。

使用传输连接的一种可能场景如下所述。

（1）主机 2 上的邮件服务器进程将自己关联到 TSAP 1522 上，等待入境连接请求的到来。至于进程如何将自己关联到 TSAP 上，这超出了网络模型的范畴，完全取决于本地操作系统。例如，用我们的 LISTEN 调用就可以做到这一点。

（2）主机 1 上的应用进程希望发送一个邮件消息，所以它把自己关联到 TSAP1208 上，并且发出一个 CONNECT 请求。该请求消息指定主机 1 上的 TSAP 1208 作为源，主机 2 上的 TSAP 1522 作为目标。这个动作最终导致在应用进程和服务器之间建立了一个连接。

（3）应用进程发送邮件消息。

（4）作为响应，邮件服务器表示它将传递该消息。

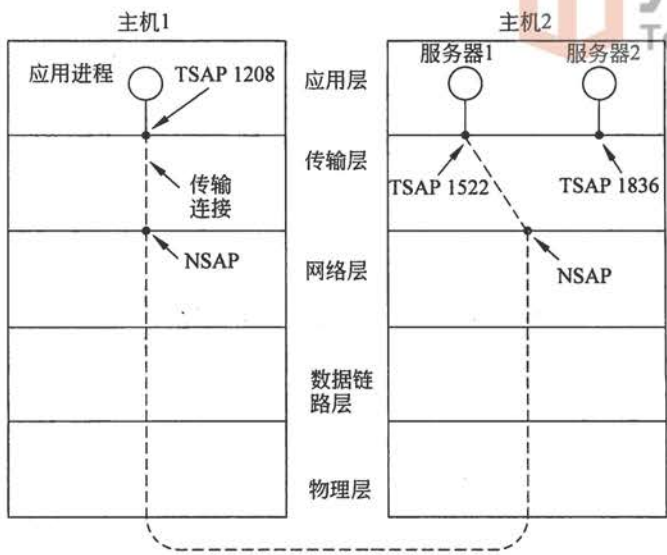


图 6-8 TSAP、NSAP 和传输连接

（5）传输连接被释放。

请注意，在主机 2 上很可能还有其他的服务器被关联到其他的 TSAP 上，它们也在等待经过同一个 NSAP 到达的入境连接请求。

上面描绘的场景非常美好，除了我们已经把这个细微问题解决了：主机 1 上的用户进程如何知道邮件服务器被关联到了 TSAP 1522 上？一种可能是，很多年以来邮件服务器一直与 TSAP 1522 关联着，渐渐地所有的网络用户都知道了这个信息。在这个模型中，具有固定 TSAP 地址的服务被列出在一些知名的文件中，比如 UNIX 系统上的/etc/services 文件，该文件列出了哪些服务器被永久地关联到哪些端口上，事实上，我们可以发现邮件服务器被固定在 TCP 端口 25。

虽然固定的 TSAP 地址适用于少数永不改变的关键服务（比如 Web 服务器），但一般来说，用户进程经常需要跟其他用户进程通信，这些用户进程的 TSAP 地址无法预先得知，而且可能只存在较短的时间。

为了处理这些情形，通常考虑使用另一种方案。在这个方案中，存在一个称为**端口映射器**（portmapper）的特殊进程。为了找到一个给定服务名字（比如“BitTorrent”）相对应的 TSAP 地址，用户需要与端口映射器（它总是在监听一个知名的 TSAP）建立一个连接；然后，用户通过该连接发送一条消息指定它想要的服务名字；端口映射器返回相应的 TSAP 地址。之后，用户释放它与端口映射器之间的连接，再与所需的服务建立一个新的连接。

在这个模型中，当一个新的服务被创建时，它必须向端口映射器注册，把它的服务名字（通常是一个 ASCII 字符串）和 TSAP 告诉端口映射器。端口映射器将该信息记录到它的内部数据库中，所以，以后当用户查询时，它就知道答案了。

端口映射器的功能类似于电话系统中的查号操作员——提供的是从名字到电话号码之间的映射关系。如同在电话系统中一样，很重要的一点是端口映射器使用的知名 TSAP 地址一定是真正众所周知的。如果你不知道查号操作员的号码，那么你根本不可能呼叫查号

操作员来查询所需要的号码。如果你认为自己所拨打的信息服务号码是非常显然的，那么当你有机会到国外旅行的时候可以试一试。

一台机器上可能存在多个服务器进程，其中有许多服务很少被人使用。如果让这些服务器进程一直都活跃着，并整天监听一个稳定的 TSAP 地址，则是一种浪费。另一种方案的简化形式如图 6-9 所示，这个方案称为**初始连接协议**（initial connection protocol）。不是每个想象的服务器都在一个知名 TSAP 上监听，每台希望向远程用户提供服务的机器有一个特殊的**进程服务器**（process server）充当那些不那么频繁使用的服务器的代理。这个服务器在 UNIX 系统中称为 inetd。它在同一时间监听一组端口，等待连接请求的到来。一个服务的潜在用户发出一个连接请求，并指定他们所需服务的 TSAP 地址。如果该 TSAP 地址上没有服务器正等着，则他们得到一条与进程服务器的连接，如图 6-9（a）所示。

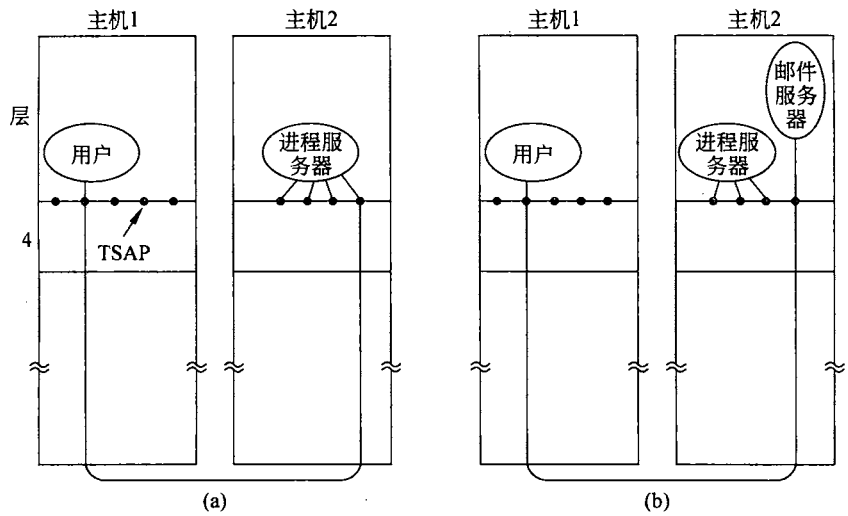


图 6-9 主机 1 上的一个用户进程是如何通过进程服务器与主机 2 上的邮件服务器建立连接的

在获取入境请求后，进程服务器派生出被请求的服务器，允许该服务器继承与用户的现有连接。新服务器完成要求的工作，而进程服务器可回去继续监听新的连接要求，如图 6-9（b）所示。这种方法只适用于服务器可按需创建的场合。

6.2.2 连接建立

建立一个连接听起来很容易，但是实际上却出奇的琐碎。初看起来，好像一个传输实体只要给接收方发送一个 CONNECTION REQUEST 段，然后等待 CONNECTION ACCEPTED 应答就足够了。当网络可能出现丢失、延迟、损坏和重复数据包的时候，问题就发生了。这些行为将导致极为严重的复杂性。

想象一个网络非常拥塞，以至于几乎所有的确认数据包都无法及时地返回到发送端，因此每个数据包都超时，因而被迫重传两次或者三次。假设该网络内部使用数据报技术，并且每个数据包遵循不同的路径。有些数据包可能受到网络流量拥挤的影响，从而需要很长时间才能到达目的地，也就是说，它们被网络延迟了，许久以后才被送到接收端，而此时发送端认为这些数据包已经丢失了。

可能出现的最坏噩梦是这样的情形：用户建立了一个与银行的连接，并且发送消息告诉银行把一大笔钱转移到一个并不是完全值得信赖的人的账户。不幸的是，数据包决定采取一条到目的地的风景路径，去探索网络的某个偏僻角落。然后，发送端超时，并且再次发送这些消息。这一次，数据包采取了一条最短路径，并且很快被交付给接收端，所以发送端释放连接。

不幸的是，最终原先的那批数据包终于从某个隐藏处冒了出来，并到达目的地，要求银行建立一个新的连接和汇款（再次）。银行没有办法得知这些是重复请求。它必须假设这是第二个并且独立的交易请求，因此再次转钱。

这种场景可能听起来似乎不可能发生，甚至令人难以置信，但问题就是这样：协议必须被设计成在所有情况下都能正确运行。在普通情况下协议的实现必须有效，以便取得良好的网络性能；但当出现罕见情况时协议也必须能够应付不会中断。如果协议做不到这点，则说明我们建立的一个公平的全天候网络竟然在一些条件下没有发出警告就失败了。

本节的其余部分，我们将研究延迟重复的问题，强调以可靠方式建立连接的算法，因此类似上面的梦魇不可能再发生。问题的症结在于被延迟的重复数据包在接收端看来是新数据包，从而造成重复接收。我们不能防止出现数据包的重复和被延迟的情况，但如果真的发生了这种情况，那么必须拒绝重复的数据包，不能把它们当做新数据包来处理。

这个问题可以有各种方式的解决途径，但没有一种令人非常满意。一种方法是使用一次性的传输地址。在这种方法中，每次传输都需要一个地址，而且是一个新产生的地址。当连接被释放时，该地址将被丢弃，并从来不被重复使用。这样，延迟的重复数据包将永远找不到抵达传输进程的途径，从而不会损害协议的正常运行。然而，这种方法使得首次与一个进程建立连接变得异常困难。

另一种可能的做法是连接发起方为每个连接分配一个唯一标识符（即一个序号，每建立一个连接，该序号就递增），并且将该标识符放在每个段中，也包括请求建立连接的那个段。当释放某个连接时，每个传输实体可以更新一张内部表，该表列出了所有过期的连接，形式为“对等传输实体，连接标识符”。每当到达一个连接请求，传输实体就检查该表，看它是否属于某一个以前已被释放掉的老连接。

不幸的是，这种方案有个基本缺陷：它要求每个传输实体无限期地维护一定数量的历史信息。这种历史信息必须在源机器和目标机器上保持一致。而且，如果一台机器崩溃，丢失了它的全部内存，那么它就不可能知道哪些连接标识符已经被对等实体用过了。

因此，我们需要采用不同的策略来简化问题。我们不再允许数据包在网络中无限期地生存下去，而是设计一种机制来杀死那些已经过时但仍在网络中蹒跚的数据包。有了这个限制，问题就变得好管理多了。

数据包的生存期可以用以下一种（或多种）技术被限定在一个给定的最大值之内：

（1）限制网络设计。

（2）在每个数据包中放置一个跳计数器。

（3）为每个数据包打上时间戳。

第一种技术包括任何一种避免数据包进入循环的方法，并结合某种限定延迟的方法，这个延迟要包括（已知的）最长可能路径上拥塞延迟的上界。第二种方法是，将跳计数器初始化为某个适当的值，然后每次数据包被转发的时候该跳计数器减一。网络协议简单丢

弃掉那些跳计数器变成零的数据包。第三种方法要求每个数据包携带它的创建时间，路由器负责丢弃那些年限超过某个预设值的数据包。后一种方法要求同步所有路由器的时钟，而这本身不是一件很容易完成的任务，实际上用跳计数器足够接近生存期。

实际上，我们不仅要保证一个数据包死亡，而且要保证它的所有确认最终也死亡，所以我们现在引入周期 T，它是数据包实际最大生存期的某个不太大的倍数。最大数据包的生存期是一个网络的保守常数，对于 Internet 来说，它有点随意地取了 120 秒。而倍数与具体的协议有关，它只影响 T 的长短。如果在一个数据包被发送出去之后，等待了 T 秒时间，那么可以确信该数据包的所有痕迹现在都没有了，它和它的确认将来都不会突然冒出来而使问题复杂化。

限定了数据包生存期的上界之后，我们就有可能制定一种实际并万无一失的方法来拒绝重复的段。下面描述的方法来源于（Tomlinson，1975），又被（Sunshine 和 Dalal，1978）做了进一步的提炼。它的一些变种方法已得到广泛实际应用，包括 TCP。

方法的核心是源端用序号作为段的标签，使得该段在 T 秒内不被重用。T 的大小以及数据包速率（数据包/秒）确定了序号的大小。这样，在任何给定的时间内只能出现一个给定序号的数据包。这个数据包的重复现象仍然有可能发生，但是它们必须被接收方丢弃。然而，这样的情况不会再发生：即一个具有相同序号的重复的老数据包击败一个新数据包，取而代之地被接收方接受。

为了解决一台机器崩溃之后丢失所有内存的问题，一种可能的解决方法是要求传输实体在机器崩溃之后空闲 T 秒。这段空闲期将确保所有老的段全部死掉，因而发送端可以启用任何一个序号值。然而，在复杂的互联网络中，T 可能会很大，因此这种策略不具吸引力。

相反，Tomlinson 建议每台主机都配备一个日时钟（time-of-day clock）。不同主机上的时钟不需要同步。假定每个时钟均采用了二进制计数器的形式。该计数器以统一的时间间隔递增自己，而且，计数器的位数必须等于或者超过序号的位数。最后，也是最重要的假设是，即使主机停机时钟也不停下，它将持续不停地运行。

当建立一个连接时，时钟的低 k 位被用于同样是 k 位的初始序号。因此，与第 3 章中的协议不同的是，每个连接都从一个完全不同的初始序号开始对它的段进行编号。序号空间应该足够大，以便当序号回绕时，原来那些具有老序号的段都已经消失。时间和初始序号之间的这种线性关系如图 6-10（a）所示。禁止区域显示了段序号非法使用的对应时间。

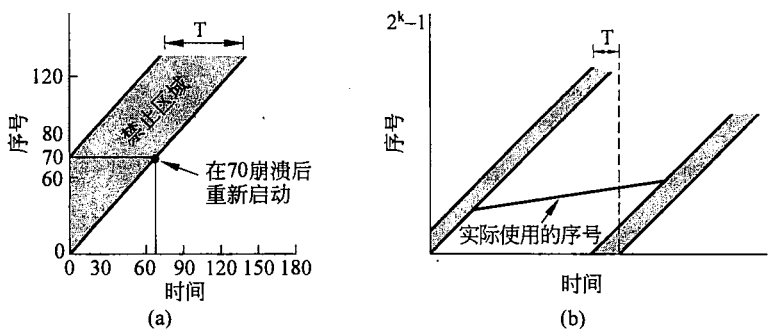


图 6-10

（a）段不能进入的禁止区域；（b）重新同步问题

任何发出去的段，如果其序号恰好落在禁止地区，那么它可能被延迟，因而假冒了其后不久发出的具有相同序号的段。例如，如果主机在70秒时崩溃并重新启动，它将使用的初始序号基于它崩溃后的时钟，因此主机不可能启用一个位于禁止区域的较低序号。

一旦双方的传输实体已经统一了初始序号，则可以用任何一个滑动窗口协议来控制数据流的传输。窗口协议将发现并丢弃早就已经接受的重复数据包。事实上，初始序号曲线（图中的粗线）并不是线性的，而是一条梯状线，因为时钟是离散前进的。为了简化起见，我们忽略这一细节。

为了防止序号进入禁止区域，我们必须兼顾两个方面。协议可能以两种不同方式遇到麻烦。如果一台主机在一个新打开的连接上发送得太快，则实际序号与时间的曲线可能比初始序号与时间的曲线还要陡，导致序号进入禁止。为了防止这种情况出现，每个连接上的最大数据速率是每个时钟滴答一次发送一段。这同时也意味着在机器崩溃并重启动之后，传输实体必须等待时钟滴答，才能打开一个新的连接，以免同样的序号被使用两次。这两点都倾向于使用短的时钟滴答（1微秒甚至更短）。但是相对序号来说时钟不能滴答得太快。假设时钟速率为C，序号空间大小为S，则有S/C > T，才能使得序号不至于回绕得太快。

由于发送速度太快而造成由下方往上进入禁止区域并不是带来麻烦的唯一情形。从图6-10（b）我们可以看出，如果以任何低于时钟速率的数据率发送，实际使用的序号与时间之间的曲线最终会从左边进入到禁止区域中。实际序号曲线的斜度越大，则这种事件发生得越晚。为了避免这种情况发生，应该限制连接序号递增的速度不能太慢（或连接可能会持续多久）。

基于时钟的方法解决了无法区分重复段的问题。然而，要把这种方法应用到连接建立过程还要拔掉一个钉子。因为我们通常不会记住接收方在该连接上的序号，所以我们依然无法判断CONNECTION REQUEST段中包含的那个初始序号是否与当前连接的序号重复。这个问题在连接期间不会存在，因为滑动窗口能记住当前序号。

为了解决这个特殊问题，（Tomlinson，1975）引入了**三次握手**（three-way handshake）。这是一个建立连接采用的协议，它要求一方检查连接请求是否的确是当前的。图6-11（a）显示了主机1发起连接请求时的正常建立过程。主机1选择一个序号x，并且发送一个包含x的CONNECTION REQUEST段给主机2。主机2回应一个ACK段作为对x的确认，并且宣告它自己的初始序号y。最后，主机1在它发送的第一个数据段中，对主机2选择的初始序号进行确认。

现在我们来看当出现延迟的重复控制段时三次握手协议是如何工作的。在图6-11（b）中，第一个段是一个旧连接上被延迟了的重复CONNECTION REQUEST。该段到达主机2，而主机1对此并不知情。主机2对这个段的回应是给主机1发送一个ACK段，其效果相当于验证主机1是否真的请求建立一个新的连接。当主机1拒绝了主机2的连接建立请求后，主机2就意识到被一个延迟的重复段所欺骗了，于是放弃连接。这样，一个延迟的重复段没有造成任何伤害。

最坏的情况发生在延迟的CONNECTION REQUEST和ACK同时出现在网络中时。这种情形如图6-11（c）所示。如同前一个例子一样，主机2得到一个延迟的CONNECTION REQUEST，并回应了该请求。这里的关键是必须意识到，主机2已经建议使用y作为从主

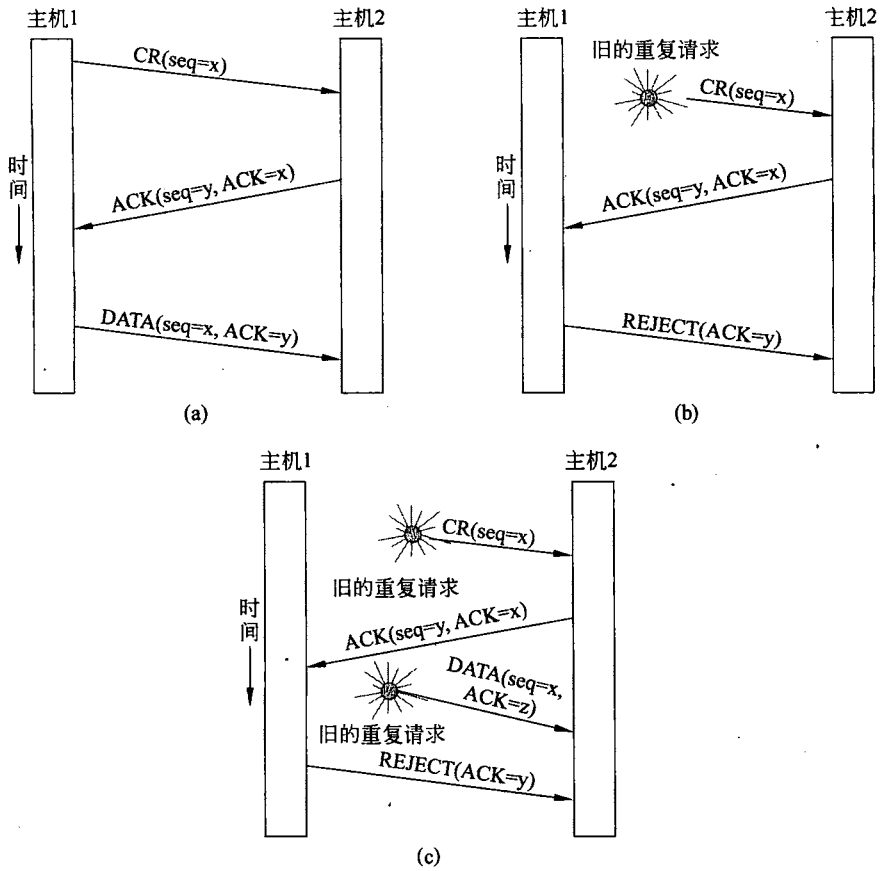


图 6-11 采用三次握手建立连接时的三种协议场景（CR 表示 CONNECTION REQUEST）
（a）正常操作；（b）出现老的重复 CONNECTION REQUEST；（c）重复 CONNECTION REQUEST 和重复 ACK

机 2 到主机 1 之间流量的初始序号，同时也要知道，现在已经没有包含序号为 y 的段或者对 y 的确认。当第二个延迟的段到达主机 2 时，主机 2 注意到确认的是 z 而不是 y，这个事实告诉主机 2 这也是一个老的重复数据包。在这里，必须认识的一件非常重要的事情是，老的段的任何组合都不能够让协议失败，也不会出人意料地偶然建立一个连接。

TCP 使用三次握手机制来建立连接。在连接期间，时间戳被用来辅助扩展 32 位序号，以便它在最大数据包生存期间不会回绕，甚至对于每秒千兆位的连接也一样。当 TCP 被用在越来越快的链路上时，这种机制是对 TCP 的必要修复。该机制由 RFC1323 描述，称为**防止序号回绕**（PAWS，Protection Against Wrapped Sequence numbers）。在 PAWS 机制发挥作用以前，整个连接期间对于初始序号，TCP 最初使用了刚才所描述的基于时钟的方案。然而，结果显示竟然存在一个安全漏洞。时钟使得攻击者很容易预测到下一个初始序号，然后发送数据包欺骗三次握手，从而建立一个伪造的连接。为了堵住这个漏洞，在实际使用中采用了伪随机的初始序号。然而，仍然很重要的是在一个间隔内初始序号不能重复，即使它们随机出现。否则，延迟的重复数据包可能肆虐成灾。

6.2.3 连接释放

释放一个连接要比建立一个连接容易得多。然而，这其中也存在着许多让人意想不到的陷阱。正如我们前面提到过的，终止连接的方式有两种：非对称释放和对称释放。非对称释放连接是电话系统的工作方式：当一方挂机后，连接就被中断了。对称释放连接是把连接看成两个独立的单向连接，要求单独释放每一个单向连接。

非对称释放方法太冒失，可能导致数据的丢失。请考虑图 6-12 中的场景。当连接被建立之后，主机 1 发送一段，它正确地到达了主机 2；然后，主机 1 发送另一段。不幸的是，主机 2 在第二个段到达之前发出了 DISCONNECT。结果该连接被释放，而数据却丢失了。

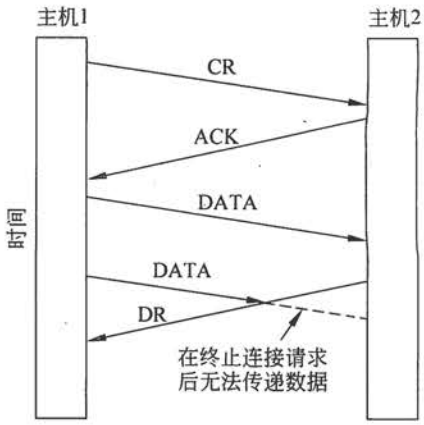


图 6-12 突然中止连接导致数据丢失

很显然，我们需要一个更加复杂的释放协议来避免数据丢失。一种方法是使用对称释放方式，每个方向被单独释放，两个方向的连接释放相互独立。因而，即使当主机发送了 DISCONNECT 段以后，它仍然可以接收数据。

当连接两端的每个进程有固定数量的数据要发送，并且清楚地知道何时发送完这些数据时，对称释放方法可以断开连接。在其他的情形中，要确定所有的工作都已完成并且连接应该被终止并不是那么显而易见的容易。你可以想象这样一个协议，主机 1 说“我已经完成任务了，你也完成了吗？”如果主机 2 回答：“我也完成了。再见，可以安全地释放连接了。”

不幸的是，这个协议并不总能正确地工作。有一个著名的问题说明了这点。它就是所谓的**两军对垒问题**（two-army problem）。请想象一支白军被围困在一个山谷中，如图 6-13 所示。两旁的山上都是蓝军。白军的实力超过了两旁任何一支蓝军单独的力量，但是两支蓝军合起来的实力却超过了白军。如果任何一支蓝军单独发起进攻，则它将被白军击败；然而，如果两支蓝军同时发起攻击，则它们将会取得胜利。

图 6-13 两军对垒问题

两支蓝军希望能够同时发动攻击。然而，它们唯一的通信介质是派士兵穿过山谷传递

消息，而在穿越山谷时士兵可能会被白军抓住，从而丢失消息（即它们必须使用一条不可靠的通信信道）。现在的问题是：是否存在一个让蓝军获胜的协议？

假设蓝军1号的指挥官发送这样一条消息："我建议我们在3月29日的黎明时分发起进攻。怎么样？"现在假设该消息到达了蓝军2号，2号指挥官同意这一建议，并且他的回信安全地回到了蓝军1号。进攻会如期进行吗？可能不会，因为蓝军2号指挥官不知道他的回信是否能送到。如果回信没有送过去，蓝军1号将不会发动进攻，所以对他来说，贸然发动进攻将是十分愚蠢的。

现在我们对协议进行改进，将它变成一个三次握手协议。最初建议的发起方必须对应答消息进行确认。假设没有消息丢失，蓝军2号将得到确认；但是，蓝军1号指挥官现在犹豫了。毕竟，他不知道他的确认信是否送过去了，如果确认信没有送到蓝军2号，他知道蓝军2号就不会发动进攻。我们当然可以设计一个四次握手协议，但是，这并不能帮助我们解决上面任何一个问题。

实际上，可以证明完成这一任务的协议并不存在。我们用反证法，假设存在某一个协议可以正确工作。该协议的最后一条消息可能至关重要，也可能不那么重要。如果不重要，则从协议中去掉这条消息（以及其他所有无关紧要的消息），这样我们得到的协议中每条消息都是至关重要的。如果最后一条消息没有被送过去，则情形会怎么样呢？我们刚才说过了它是至关重要的，所以如果它丢失了，则进攻就不会如期进行。由于最后一条消息的发送端永远也无法确定它是否正确地到达，所以他不会冒险发动进攻。更糟的是，另一支蓝军也明白这一点，所以它也不会发动进攻。

为了看清楚两军对垒问题与释放连接问题之间的相关性，只要用"断开连接"来代替"发动进攻"就可以。如果任何一方要在确定另一方做好了断开连接的准备之后才断开连接，那么，断开连接的操作将永远也不可能发生。

实际上，我们可以把这种双方协商释放连接的需求推给传输层的用户，使得连接两端独立决定是否释放连接，从而避免出现上述窘态。由此这个问题很容易地就解决了。图6-14显示了使用三次握手法释放连接的4个场景。虽然这个协议并非完全没有错误，但是通常情况下已经足够了。

在图6-14（a）中，我们看到正常情况下，一个用户发送一个DR（DISCONNECTION REQUEST）段来启动释放连接过程。当该段到达对方，接收端也发回一个DR段，并启动一个计时器，设置计时器的目的是为了防止它的DR丢失。当这个DR到达时，最初的发送端发回一个ACK段，并且释放连接。最后，当ACK返回后，接收端也释放连接。释放一个连接意味着传输实体将有关该连接的信息从它的内部表（记录了所有当前已打开的连接）中删除，并且通知该连接的所有者（传输用户）。这个动作与传输用户发出一个DISCONNECT原语有所不同。

如果最后的ACK段被丢失，则如图6-14（b）所示，这种情形可以通过一个计时器来补救。当计时器超时，无论如何连接都要被释放。

现在考虑第二个DR被丢失的情形。发起释放连接的用户接收不到期望的响应，所以它将超时，于是再次尝试释放连接。在图6-14（c）中，我们可以看到这个工作过程，图中假定第二次过程中没有段丢失，所有的段都被及时地递交给正确的接收端。

图6-14（d）所示的最后一个场景与图6-14（c）中相同，但由于丢失段的原因，所有

重传 DR 的尝试都失败。经过 N 次重试之后，发送端放弃了，并且释放连接。同时，接收端超时，于是退出连接。

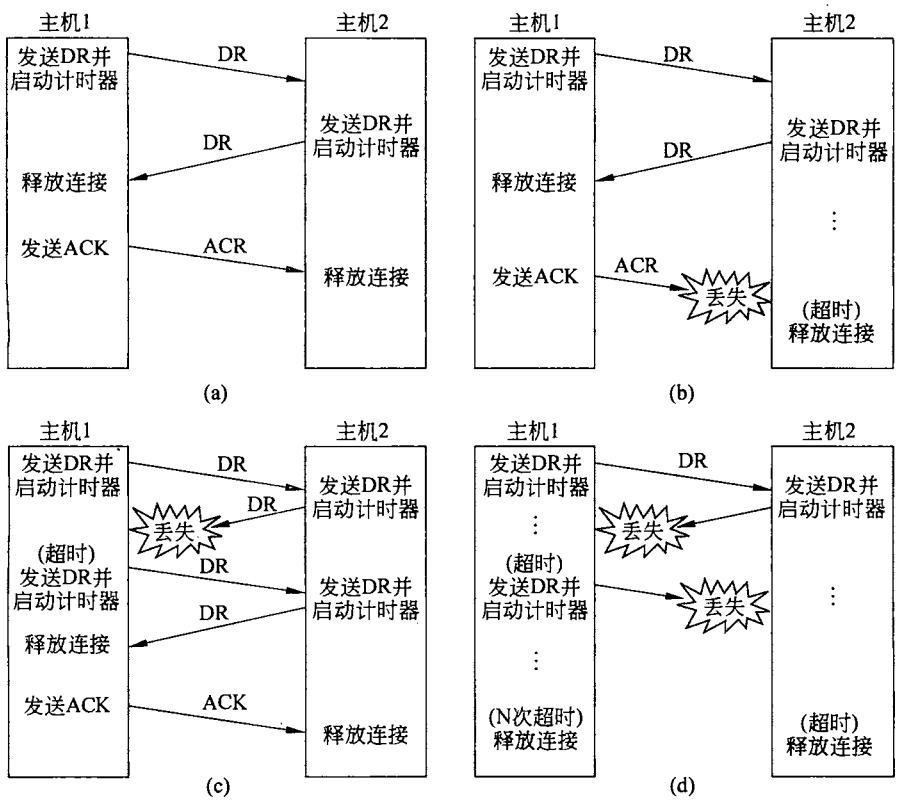


图 6-14 释放连接的 4 种协议场景

（a）正常的三次握手情况；（b）最后一个 ACK 被丢失了；（c）响应丢失了；（d）响应和随后的 DR 都丢失

虽然这个协议在一般情况下已经足够了，但在理论上，如果初始的 DR 和 N 次重传全部丢失，则协议可能失败。发送端将最终放弃发送释放连接请求并强行释放连接，而另一方对所有的释放连接尝试一无所知，它的连接仍然处于活跃状态。这种情况会导致一个半开的（half-open）连接。

如果我们不允许发送端在经过 N 次重试之后放弃，而是让它不停地重试，直至得到对方的响应，那么，这个问题就可以避免。然而，如果另一方允许超时，那么发送端将真的要永远尝试下去，因为没有响应会出现。如果我们不允许接收端超时，那么图 6.14（d）中的协议将被悬挂起来。

杀死半开连接可以采用这样一条规则：如果在规定的一段时间内没有段到来，该连接将被自动断开。以这种方式，如果一方断开连接，则另一方将检测到该连接没有活动，于是也会将其释放。这条规则也需要考虑连接中断的情况（由于网络不再在主机之间传送数据包），此时连接两端谁都没有断开连接。当然，如果引入这条规则，那么每当传输实体发送一个段的时候，它必须要先停止一个计时器，然后再重新启动。如果该计时器超时，则传送一个哑段，其目的仅仅是为了避免另一方断开此连接。另一方面，采用了自动断开连接规则后，如果一个空闲连接上多个连续的哑段全部丢失，则首先是一方然后是另一方都

将自动断开连接。

我们不再进一步讨论有关释放连接的问题了，但是现在你应该很清楚，释放一个连接而不会丢失数据并不像初看起来那么简单。这里传达的经验是，传输层用户必须参与进来决定何时断开连接——这个问题传输实体本身无法彻底地解决。要看清楚应用程序在这里的重要性，考虑这样的情形：虽然 TCP 通常采用对称释放方式来断开连接（每一边在发送完自己的数据之后用一个 FIN 数据包独立关闭其一半的连接），许多 Web 服务器给客户端发送一个 RST 包，导致突然关闭连接，这种工作方式更像非对称释放方式。之所以 Web 服务器能这样处理，是因为它知道数据交换模式。首先，它接收来自客户端的请求，这是客户端发送的所有数据；然后它给客户端发送一个响应。当 Web 服务器完成其响应工作后，意味着两个方向上的所有数据都已经发送完毕。服务器可以给客户端发送一个警告，然后骤然关闭连接。如果客户端得到这样的警告，它会释放它的连接状态；如果客户端没有得到警告，最终它会意识到服务器不再与自己交谈，并释放连接。在这两种情况下数据都已经得到成功传送。

6.2.4 差错控制和流量控制

在详细讨论了连接的建立和释放过程以后，现在我们来看如何在使用连接的过程中进行管理。关键问题是差错控制和流量控制。差错控制确保数据传输具备所需的可靠性，通常指所有的数据均被无差错地传送到目的地。流量控制是防止快速发送端淹没慢速接收端。

这两个问题我们在数据链路层已经考查过。传输层采用的解决方案与我们在第 3 章学过的一样。这里再简单回顾一下：

（1）帧中携带一个检错码（比如，CRC 或者校验和）用于检测信息是否被正确接收。

（2）帧中携带的序号用于标识本帧，发送方在收到接收方成功接收后返回的确认之前，必须重发帧。这种机制称为**自动重复请求**（ARQ，Automatic Repeat reQuest）。

（3）任何时候允许发送方发送一定数量的帧，如果接收方没有及时返回确认，则发送方必须暂停。如果只允许发送一帧，则协议称为**停等式**（stop-and-wait）协议。较大的窗口可使得发送管道化，因而提高距离长且速度快的链路性能。

（4）**滑动窗口**（sliding window）协议结合了这些功能，还能被用于支持数据的双向传送。

鉴于这些机制已成功应用于链路层上的帧，很自然地人们会怀疑它们是否也适用于传输层上的段。然而，实际上链路层和传输层的重复不多。即使使用相同的机制，它们在功能上和程度上都有很大的差异。

首先是功能上的差异，考虑错误检测。链路层的校验和保护一个穿过单条链路的帧。传输层的校验和则保护跨越整个网络路径的段，这是一个端到端的校验机制，与每条链路上的校验并不相同。（Saltzer 等，1984）描述了一种场景，数据包在一个路由器内部被损坏的情形。链路层校验和只保护在一条链路上经过的数据包，而没有考虑它们在路由器内部出错的情况。因此，即使根据每条链路的校验和检查都正确，数据包仍然会被不正确地传递。

Saltzer 等给出的这个例子和其他的例子阐明了端到端的观点。根据这种观点，执行端

到端的传输层校验对传输的正确性至关重要，而链路层的校验反而不是必不可少，但也仍然有价值，因为能提高性能（如果没有链路层的校验，损坏的数据包就必须沿整条路径重传，这显然是没有必要的）。

至于传输层和数据链路层在程度上的差异，考虑重传和滑动窗口协议。对于大多数无线链路，卫星链路除外，发送方一次只能发送一帧。也就是说，链路的带宽延迟乘积很小，甚至不能在链路上容纳一个完整的帧。在这种情况下，一个小的窗口足够产生良好的性能。例如，802.11使用停-等式协议，每个帧被传输或者重传，直到它被接收方确认窗口才能移动到下一帧。大于一帧的窗口不仅没有改善协议性能，而且增加了控制复杂性。对于有线和光纤链路，比如（交换式）以太网或ISP骨干网，链路的误码率很低，因而可以忽略重传，因为端到端的重传会修复剩余的那些被链路层丢失的帧。

另一方面，许多TCP连接的带宽延迟乘积远远大于单个段。考虑美国的一个连接，发送速率为1 Mbps，往返时间为100毫秒。即使这样一个慢速连接，在发送端发送一段并接收确认所需要的这段时间内，已经在接收端存储了200 Kb的数据。这种情况下，必须使用一个大的滑动窗口。停-等式的控制会削弱协议性能。在我们的例子中，它会限制性能，无论实际网络多块，总是每200毫秒发送一帧，或者说每秒发送5段。

既然传输协议通常使用较大的滑动窗口，我们将着眼于更仔细的缓冲数据问题。由于主机可能有多个连接，每个连接单独处理，因此它可能需要一个相当数量的缓冲区用作滑动窗口。发送端和接收端两端都需要缓冲区。当然，发送端需要用这些缓冲区来保存所有已经传输但尚未得到确认的段。之所以发送端需要缓冲区是因为发送的段可能会丢失，必须重传，因此要保存已发但未确认的段。

然而，由于发送端缓冲着发送的段，接收端可能会或可能不会为特定的连接设置专用的特定缓冲区，具体按它认为合适的方式做。例如，接收端可能只设立一个缓冲池让所有连接共享。当一个段到达接收端，接收端企图动态获得一个新缓冲区。如果刚好有个缓冲区可用，则该段被接受，否则将被丢弃。由于发送端时刻准备重新传输网络中丢失的段，因此接收端丢弃段并没有造成永久的伤害，虽然要浪费一些资源。发送端不断尝试发送，直到它得到确认为止。

源缓冲区和目标缓冲区之间的最佳权衡取决于由连接承载的流量类型。对于低带宽的突发流量（比如交互式终端产生的流量），不设置任何缓冲区是合理的；相反，如果段肯定偶尔会被丢弃，那么在连接两端动态获取缓冲区，具体的做法依赖于发送端的缓冲。另一方面，文件传输和其他高带宽流量，如果接收端有专用的满窗口的缓冲区最好，这样能允许数据流以最大速率发送。这就是TCP使用的策略。

还有一个问题涉及如何组织缓冲池。如果大多数段的长度都差不多，很自然的做法是将缓冲区组织成一个由大小统一的缓冲区构成的池，每个缓冲区容纳一个段，如图6-15（a）所示。然而，如果段长度的差异范围很大，可能是一个请求Web网页的短数据包，也可能是P2P文件传输中的大数据包，那么，用固定长度的缓冲区池就有问题。如果将缓冲区设置成最大可能的段长，那么当短数据包到来时就会浪费空间；如果将缓冲区设置成比最大的段要小，则长的段就需要多个缓冲区，从而带来额外的复杂性。

解决缓冲区大小问题的另一个方法是使用可变大小的缓冲区，如图6-15（b）所示。这里的优点是能获得更好的内存利用率，付出的代价是缓冲区的管理更加复杂。第三种可能

的方案是为每个连接使用一个大的循环缓冲区，如图 6-15（c）所示。这个系统古朴典雅，并且不依赖于段的大小，但只有当连接重载时，内存的使用情况才很好。

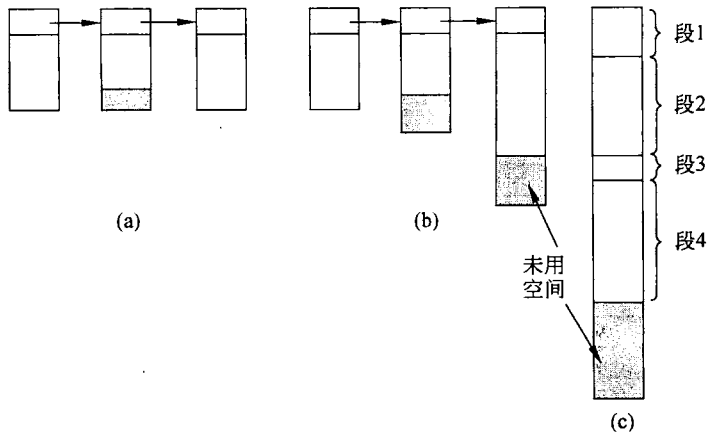


图 6-15
（a）链式固定大小的缓冲区；（b）链式可变大小的缓冲区；（c）每个连接一个大循环缓冲区

随着连接被打开和关闭，以及流量模式的变化，发送端和接收端需要动态地调整它们的缓冲区分配策略。因此，传输协议应该允许发送主机请求另一端的缓冲区空间。缓冲区可以分配给每个连接，或者综合分配给两台主机之间当前正在运行的所有连接。另一种做法是，接收端知道自己的缓冲区情况（但是不知道流量的情况），可以告诉发送端“我已经为你预留了 X 个缓冲区。”如果打开的连接数量增加，或许有必要减少为每个连接分配的缓冲区数。因此，协议应该提供这种协商能力。

为了管理动态缓冲区的分配，一种合理的惯常做法是将缓冲与确认机制分离，这种做法与第 3 章介绍的滑动窗口协议做法不同。动态的缓冲区管理实际上意味着一个可变大小的窗口。初始时，发送端根据它期望的需求，请求一定数量的缓冲区。然后，接收端根据它的能力分配尽可能多的缓冲区。每次发送端传输一段，它必须减小分配给它的缓冲区数，当分配给它的缓冲区数达到 0 时，完全停止发送。然后，接收端在逆向流量中捎带上单独的确认和缓冲区数。TCP 采用这种模式，将缓冲区的分配捎带在头的 Window size 字段中。

图 6-16 显示了一个数据报网络如何动态管理窗口的例子，其中序号用 4 位标识。在例子中，段的数据流从主机 A 发往主机 B，段的确认和缓冲区分配流逆向传输。初始时，A 想要 8 个缓冲区，但是，B 只分配了 4 个；然后 A 发送 3 个段，其中第 3 个段被丢失。在第 6 行，B 确认已经接收到直至（含）序号 1 的所有段，从而允许 A 释放这些缓冲区；同时进一步通知 A，允许发送从序号 1 之后开始的 3 个段（即段 2、3 和 4）。A 知道自己已经发送了段 2，所以它认为现在可以发送段 3 和段 4，于是它接下去就这么做了。此刻，A 被阻塞，它必须等待分配更多的缓冲区。然而，在阻塞过程中，因超时而导致的重传仍然可以进行（第 9 行），因为重传帧所用的缓冲区早已被分配。在第 10 行，B 确认已经接收到直至（含）序号 4 的所有段，但是拒绝让 A 继续发送。这样的情形对于第 3 章中的固定大小窗口协议是不可能发生的。下一个从 B 到 A 的段表明 B 已分配了另一个缓冲区，从而允许 A 继续发送。当 B 有缓冲空间，并很可能 B 的传输用户接受了更多的段数据时，就会发生这种情况。

	主机A	消息	主机B	注释
1	→	< request 8 buffers>	→	A 需要8 个缓冲区
2	←	<ack = 15, buf = 4>	←	B 只同意接收消息0~3
3	→	<seq = 0, data = m0>	→	A 现在剩3 个缓冲区
4	→	<seq = 1, data = m1>	→	A 现在剩2 个缓冲区
5	→	<seq = 2, data = m2>	...	消息丢失，但A 认为还剩1 个
6	←	<ack = 1, buf = 3>	←	B 确认0 和1，允许消息2~4
7	→	<seq = 3, data = m3>	→	A 有1 个缓冲区
8	→	<seq = 4, data = m4>	→	A 耗尽缓冲区，必须停止
9	→	<seq = 2, data = m2>	→	A 超时，并且重传
10	←	<ack = 4, buf = 0>	←	B 确认了所有的消息，但A 仍然阻塞着
11	←	<ack = 4, buf = 1>	←	A 现在可以发送消息5
12	←	<ack = 4, buf = 2>	←	B 从某处找到了1 个新缓冲区
13	→	<seq = 5, data = m5>	→	A 还剩下1 个缓冲区
14	→	<seq = 6, data = m6>	→	A 现在被再次阻塞
15	←	<ack = 6, buf = 0>	←	A 仍旧被阻塞着
16	...	<ack = 6, buf = 4>	←	潜在的死锁

图 6-16 动态缓冲区分配

箭头显示传输的方向。省略号指出丢失一个段

在数据报网络中，如果控制段可能丢失（肯定会丢），则这种缓冲区分配方案有可能引发一些潜在的问题。请看第 16 行，B 现在为 A 分配了更多的缓冲区，但是，这个缓冲区分配的段丢失了。由于控制段没有序号，也不会超时，所以 A 现在死锁了。为了避免出现这种情况，每台主机应该定期地在每个连接上发送控制段，这些控制段给出确认和缓冲区状态。采用这种方法，死锁迟早会被打破。

到现在为止，我们已经心照不宣地假设对发送端数据传输速率的唯一限制在于接收端的可用缓冲空间量。情况往往并非如此。内存价格曾经很昂贵，但目前已大幅下降。主机可配备足够的内存，因此缺乏缓冲区已经不是个问题；如果真是个问题，也只是针对广域连接。当然，这取决于缓冲区是否被设置得足够大，并非总是 TCP 的情况（zhang 等，2002 年）。

当缓冲区空间不再限制最大数据流时，另一个“瓶颈”将会出现：网络的承载能力。如果相邻路由器之间每秒至多交换 x 个数据包，并且在一对主机之间有 k 条互不相交的路径，那么，不管每条连接的两端有多少可用的缓冲区空间，它们每秒交换的段不可能超过 kx 个。如果发送端推进得太快（即发送速率超过 kx 段/秒），网络将变得拥塞，因为它不可能以超过段的入境速度来递交这些段。

这里需要的是一种基于网络承载容量而不是接收端缓冲容量的机制。（Belsnes，1975）建议使用一种滑动窗口流量控制方案。在他的方案中，发送端动态地调整窗口的大小，以便与网络的承载容量相匹配。这意味着动态滑动窗口可以同时实现流量控制和拥塞控制。如果网络每秒钟能够处理 c 个段，并且往返时间是 r（包括发送、传播、排队、接收端的处理以及确认的返回），那么，发送端的窗口应该是 cr。使用这样的窗口，发送端通常可以按流水线方式全速地运行。此时，网络性能只要减小一点点就有可能导致发送端被阻塞。因为每个给定流的可用网络容量随时在变，因此窗口大小也应该频繁地做出调整来跟踪当前网络承载能力的变化。正如我们后面将看到的那样，TCP 就采用了类似的控制策略。

6.2.5 多路复用

多路复用（或多个会话共享连接、虚电路和物理链路）在网络体系结构的不同层次发挥着作用。在传输层，有几种方式需要多路复用。例如，如果主机只有一个网络地址可用，则该机器上的所有传输连接都必须使用这个地址。当到达了一个段，必须有某种方式告知把它交给哪个进程处理。这种情况，称为**多路复用**（multiplexing），如图 6-17（a）所示。在图中，4 个独立的传输连接都使用了相同的网络连接（即 IP 地址）到达远程主机。

多路复用对传输层之所以非常有用，还存在另一个原因。例如，假设一台主机有多条网络路径可用。如果用户需要的带宽和可靠性比其中任何一条路径所能提供的还要多，那么一种解决办法是以轮询的方式把一个连接上的流量分摊到多条网络路径，如图 6-17（b）所示。这种手法称为**逆向多路复用**（inverse multiplexing）。若打开了 k 条网络连接，则有效带宽可能会增长 k 个因子。逆向多路复用的一个例子是**流控制传输协议**（SCTP，Stream Control Transmission Protocol），它可以把一条连接运行在多个网络接口上。相反，TCP 使用了单个网络端点。逆向多路复用也可应用在链路层，把几条并发运行的低速率链路当作一条高速率链路使用。

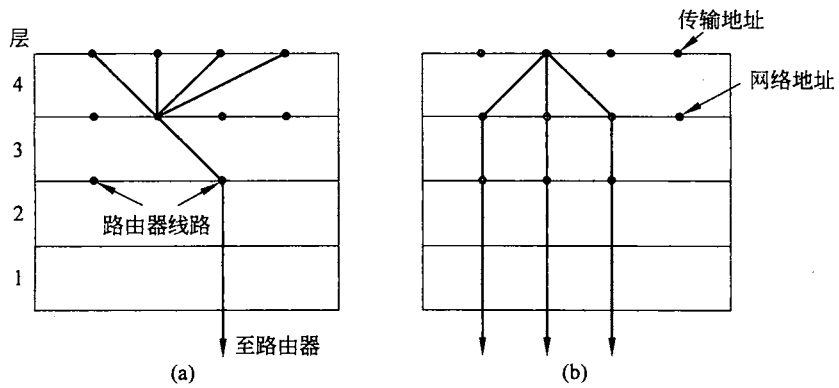


图 6-17
（a）多路复用；（b）逆向多路复用

6.2.6 崩溃恢复

如果主机和路由器崩溃，或者连接持续时间较长（比如正下载一个大软件或媒体文件），那么，如何从这些崩溃事件中恢复运行就成为一个问题。如果传输实体完全位于主机内，则从网络和路由器的崩溃中恢复比较直截了当。传输实体总是在期待着丢失的段，而且它们知道通过重传来处理这些段的丢失。

一个更加麻烦的问题是如何从主机的崩溃中恢复过来。尤其是，当服务器崩溃并且快速启动之后，客户可能期望还能够继续工作。为了说明其中的困难程度，我们假设一台主机（客户）使用一个简单的停-等式协议，正在给另一台主机（文件服务器）发送一个大文件。服务器上的传输层只是简单地将入境段逐个传递给传输用户。在传输到一半时，服务器崩溃了。当服务器恢复运行以后，它的内部表被重新进行了初始化，所以，它根本不知

道原来的传输过程进行到了哪里。

为了试图从原先的状态中恢复过来，服务器可以给所有其他主机发送一个广播段，宣告自己刚才崩溃了，并要求它的客户们告知关于所有打开连接的状态信息。每个客户可能处于以下两种状态之一：发出一个段，但尚未确认，S1状态；或者，没有未完成的段，S0状态。根据这一状态信息，客户必须确定是否重传最近的段。

初看起来似乎很明显：当客户知道服务器崩溃时，当且仅当有一个未确认的段悬挂着时（即处于状态S1）它才重传该段。然而，进一步探究这种天真的做法便会发现其中的困难。例如，请考虑这样的情形：服务器的传输实体首先发送一个确认，然后在确认被发出以后再执行写操作，把数据传给应用进程。将一个段写到输出流和发送一个确认，这是两件不能同时进行的独立事件。如果在确认被发送出去之后，但是在写操作完成之前服务器崩溃了，那么客户将会接收到确认；当服务器崩溃恢复之后发出的广播消息到来的时候，该客户处于状态S0。因此客户不会重传段，因为它（不正确地）认为段已经到达服务器了。客户的这一决定将导致错失一个段。

此刻你可能会这样想："这个问题很容易解决，只要重新修改传输实体的程序，让它先执行写操作再发送确认即可。"你可以再试一试。请想象一下，如果写操作已经完成，但是在发送确认之前服务器崩溃了，那么，客户将处于状态S1中，因此它会重传，从而导致在服务器应用进程的数据流中出现一个未被检测出来的重复段。

无论如何编写客户和服务器程序的代码，总是存在使协议无法正确恢复运行的情形。服务器程序可以编成两种方式：先发送确认，或者先写数据。而客户程序可以编成四种不同方法：总是重传最后一段、永远不重传最后一段、仅当在状态S0时才重传，或者仅当在状态S1时才重传。服务器和客户两者结合起来有8种操作组合，但是正如我们将会看到的，对于每一种组合都存在一些使协议失败的事件。

服务器端可能发生3种事件：发送一个确认（A）、将数据写到输出进程（W）和崩溃（C）。这3种事件的发生有6种不同的顺序：AC（W）、AWC、C（AW）、C（WA）、WAC和WC（A），这里的括号表示A或者W不可能发生在C的后面（即一旦崩溃那就彻底崩溃）。图6-18显示了客户和服务器策略的8种组合，以及每一种组合的有效事件序列。请注意，对于每一种策略，总是存在使协议失败的事件序列。例如，如果客户总是重传的话，则AWC事件将生成一个无法检测到的重复，即使其他两个事件可以正确地工作。

	接收主机采用的策略					
	先ACK，后写			先写，后ACK		
发送主机采用的策略	AC(W)	AWC	C(AW)	C(AW)	WAC	WC(A)
总是重传	OK	DUP	OK	OK	DUP	DUP
永远不重传	LOST	OK	LOST	LOST	OK	OK
在状态S0时重传	OK	DUP	LOST	LOST	DUP	OK
在状态S1时重传	LOST	OK	OK	OK	OK	DUP

OK＝协议功能正确
DUP＝协议产生了一个重复消息
LOST＝协议丢失了一个消息

图6-18 客户和服务器策略的不同组合

进一步精心修改协议也无济于事。即使在服务器试图写之前，客户与服务器再多交换几段，使得客户确切地知道将要发生什么事情，客户也没有办法知道崩溃动作发生在写操作之前还是之后。结论是不可避免的：在事件不同步的基本规则下——即单独的事件一个接着一个发生，而不是同时发生，那么主机的崩溃和恢复就无法做到对上层透明。

按照更加一般的术语表达，这个结论可以重新叙述为："从第 N 层崩溃中的恢复工作只能由第 N+1 层完成"，并且仅当高层保留了问题发生前的足够状态信息时才有可能恢复。这与上面提到的观点一致，只要连接两端记录了它当前的状态信息，传输层才能够从网络层的故障中恢复过来。

这个问题促使我们思考究竟所谓的端到端确认意味着什么。原则上，传输协议是端到端的，而不像下面层次那样是链式的。现在请考虑这样的情形：一个用户向一个远程数据库发出了一些事务请求。假设远程传输实体先将段传递给上一层，然后再确认。在这个例子中，即使用户机器收到返回的确认也不一定意味着远程主机一定能够撑足够长的时间来真正完成数据库的更新操作。一个真正的端到端确认或许没有可能做得到，因为一旦接收到这个确认就意味着工作确实已经完成，而缺乏确认则表明工作尚未完成。（Saltzer 等，1984）详细地讨论了这一点。

6.3 拥 塞 控 制

如果许多机器上的传输实体以太快的速度发送太多的数据包，就会使得网络不堪重负而变得拥塞，继而数据包被延迟和丢失，从而导致网络性能严重下降。避免这个问题的拥塞控制是网络层和传输层的共同责任。拥塞发生在路由器上，因此在网络层检测拥塞。然而，拥塞究竟还是由传输层注入到网络中的流量引起的，因此控制拥塞的唯一途径是传输层放缓往网络中发送数据包的速度。

在第 5 章，我们学习了网络层的拥塞控制机制。本节，我们将学习问题的另一半，即传输层的拥塞控制机制。在描述完拥塞控制的目标后，我们将描述主机如何调节它们往网络发送数据包的速率。Internet 的拥塞控制严重依赖于传输层，特殊的算法被嵌入到 TCP 和其他协议中。

6.3.1 理想的带宽分配

在我们描述如何调节流量之前，我们必须理解运行拥塞控制算法要努力达到的目标是什么。也就是说，我们必须说明所谓一个好的拥塞控制算法在网络中的运行状态。拥塞控制算法的目标是更加易于避免拥塞，即为使用网络的传输层找到一种好的带宽分配方法。一个良好的带宽分配方法能带来良好的性能，因为它能利用所有的可用带宽却能避免拥塞，而且它对于整个竞争的传输实体是公平的，并能快速跟踪流量需求的变化。我们将依次精确地说明这些标准。

效率和功率

为整个传输实体有效分配带宽应该利用所有可用的网络容量。然而，假设存在一条

100 Mbps 的链路，5 个传输实体共同使用这条链路，每个实体将获得 20 Mbps，如果真这样想就大错特错了。要想获得良好的性能，它们获得的带宽应该小于 20 Mbps。其中的缘由在于流量通常呈现突发性。回忆一下，在 5.3 节我们描述过实际吞吐量（或到达接收端的有用数据包速率）是提交负载的函数。图 6-19 给出了这条曲线以及延迟曲线，与正常输出一样，延迟也是提交负载的函数。

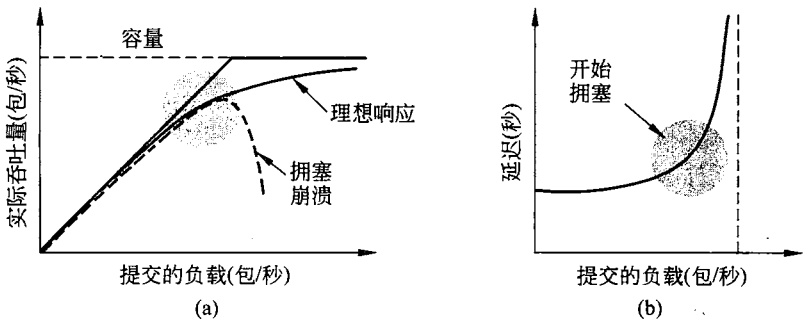


图 6-19
（a）实际吞吐量；（b）延迟作为提交负载的函数

在图 6-19（a）中，随着负载的增加实际吞吐量最初以同样的速度增加，但随着负载接近网络容量，实际吞吐量的上升逐渐增多。由于突发流量可能导致网络内缓冲区偶尔被充满并造成一些数据包丢失，因而出现实际吞吐量的衰减。如果传输协议设计不当，重传的数据包依然会被延迟但还未丢弃，此时网络内的数据包越积越多最终拥塞崩溃。在这种状态下，发送端拼命地发送数据包，但完成的有益工作只能增加一点点。

图 6-19（b）给出了相应的延迟变化情况。最初的延迟是固定的，表示穿过整个网络的传播延迟。随着负载接近网络容量，延迟逐步上升，开始上升速度比较缓慢，然后骤然上升。这也是因为突发流量在高负荷下被堆积起来的缘故。延迟实际上不可能真正达到无穷，除非在一个路由器有无限大缓冲区的模型中。相反，数据包在经历了最大的缓冲延迟后被路由器丢弃。

对于实际吞吐量和延迟，在拥塞出现时性能开始下降。直观地说，如果我们逐步加大分配的带宽，将从网络获取最佳的性能，直到延迟开始迅速攀升的那点。而这一点恰好低于网络容量。为了标识它，（Kleinrock，1979）提出了功率的度量，其中

$$\text{功率}=\text{负载}/\text{延迟}$$

功率最初将随着提交负载的上升而上升，延迟仍然很小并且基本保持不变；但随着延迟快速增长功率将达到最大，然后开始下降。达到最大功率的负载表示了传输实体放置在网络上的有效负载。

最大-最小公平性

在前面的讨论中，我们没有涉及如何在多个传输实体之间划分带宽。这看上去只回答了一个简单问题——给全部的发送端以均等的带宽比例，但这里有几点值得考虑。

也许，首先考虑的是拥塞控制究竟要解决什么问题。毕竟，如果网络给发送端一些带宽供它使用，那么发送端应该只使用那么多的带宽。然而，通常情况下，网络无法为每个流或者连接执行严格的带宽预留。如果网络支持服务质量，那么它们将为某些流预留带宽，

但是许多连接将寻求可用的任何带宽，或者被网络合并在一起共同分配带宽。例如，IETF的区分服务就将流量分成两类，每个类中的连接竞争带宽的使用。IP路由器通常让所有的连接竞争相同的带宽。在这种情况下，正是拥塞控制机制来为竞争的各个连接分配带宽。

第二个要考虑的是公平对网络中的流意味着什么。其实这点很简单，如果N个流使用一条链路，在这种情况下，它们都应该拥有1/N的带宽（虽然在流量是突发性的时候，效率决定了它们能使用的要略微少一些）。但如果有不同的流，并且这些流的网络路径出现重叠，会发生什么情况？例如，一个流可能穿越三条链路，其他流可能跨越一条链路。走三条链路的流将消耗更多的网络资源。给它分配的带宽少于只走一条链路的流获得的带宽可能在某种意义上更公平。当然有可能通过减少三条链路流的带宽来支持更多的一条链路的流。这一点体现了公平与效率之间的内在张力。

不过，我们将采取的公平概念不依赖于网络路径的长度。即使这个简单的模型，让连接分配同等比例的带宽也有点复杂，因为不同的连接将采用通过网络的不同路径，这些路径本身有不同的容量。在这种情况下，有可能一个流在下行链路上被阻塞，并且在上行链路上获得比其他流更小的流量；此时减少其他流的带宽不仅会使得这些流慢下来，而且对瓶颈流也不会有任何帮助。

公平的形式是**最大-最小公平**（max-min fairness），通常表示理想的网络使用情况。最大-最小公平分配指的是，如果分配给一个流的带宽在不减少分配给另一个流带宽的前提下无法得到进一步增长，那么就不给这个流更多带宽。也就是说，增加一个流的带宽只会让不太富裕的那些流的情况变得更糟。

让我们看一个例子。图6-20显示了最大最小公平分配方法，在这个网络中存在4个流，分别标识为A、B、C和D。路由器之间的每条链路具有相同的容量，用1个单位表示，虽然在一般情况下链路的容量是不同的。3个流竞争左下角位于路由器R4和R5之间的链路。因此，这些流中的每一个都得到1/3的链路容量。其余的流，A与B竞争从路由器R2到R3的链路。由于B已经分配得到1/3，A得到剩余的2/3链路容量。请注意，所有的其他链路都有空余容量。但是，这种空余的容量不能给予任何其他流使用，除非降低另一个流的更低容量。例如，如果把路由器R2和R3之间的带宽给B分配得多一些，那必定要降低流A的容量。这对于已经有更多带宽的流A来说是合理的。然而，流C或D（或两者）的容量必须减少才能给流B更多的带宽，而这些流的带宽将小于B。因此，这种分配方法是最大-最小公平的。

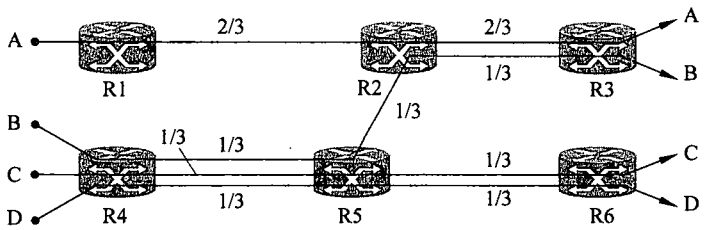


图6-20 针对4个流的最大-最小带宽分配

只要获知网络的全局知识最大-最小分配就可以计算得出来。一种直观的思考方式是想象所有的流从速率零开始，然后缓慢地增加速率。当任何一个流的速率遇到瓶颈，就停止

该流的速率增加；所有其他的流继续增加各自的速率，平等共享可用容量，直到它们也达到各自的“瓶颈”。

第三个需要考虑的因素是在什么程度上考虑公平性。一个网络的公平性可以体现在连接上、每一对主机之间的所有连接上，或者每个主机的所有连接上。我们在 5.4 节讨论**加权公平队列**（WFQ，Weighted Faire Queueing）时考查过这个问题，并得出了这样一个结论，即每种定义方法都存在着自己的问题。例如，如果针对每个主机定义公平性，则意味着一个繁忙的服务器将不会比手机获得的带宽更多；而若对每个连接定义公平性，实际上又起到鼓励主机打开更多连接的效果。鉴于没有明确的答案，经常把公平性视为针对每个连接，但确切的公平通常不予考虑。实际上，连接得不到带宽远比所有的连接得到精确的相同数量的带宽要重要得多。事实上，有可能用 TCP 打开多个连接，并且更积极地参与竞争带宽。这种战术通常被用于那些需要高带宽的应用，比如对等文件共享系统 BitTorrent。

收敛

最后一个标准是拥塞控制算法能否快速收敛到公平而有效的带宽分配上。上面讨论的理想操作点假设了一个静态的网络环境。然而，网络中的连接总是来来去去，而且一个给定连接所需要的带宽也会随时间而变化，例如，一个用户浏览网页和偶尔会下载大的视频。

由于需求的变化，网络的理想操作点也随着时间推移而改变着。一个良好的拥塞控制算法，应迅速收敛到理想的操作点，并跟踪这随时变化的操作点。如果收敛速度太慢，则算法永远无法接近已经改变的操作点；如果算法不稳定，它也可能在某些情况下无法收敛到正确的操作点，或者甚至围绕着正确的操作点振荡。

图 6-21 给出了一个带宽分配的例子，它能随着时间变化并且快速收敛。最初，流 1 拥有全部的带宽。一秒钟以后，流 2 启动，它也需要带宽。带宽分配算法迅速改变分配结果，给每个流总带宽一半的份额。在第 4 秒，第三个流加入。然而，这个流仅仅使用了 20%的带宽，这低于它应该获得的公平份额（应该是三分之一）。流 1 和流 2 迅速调整，将可用带宽每个调整到 40%。在第 9 秒，第二个流离开，第三个流保持不变。第一个流迅速捕获 80%的带宽。在整个过程中的任何时候，分配的总带宽大约是 100%，从而网络得到充分利用，各个竞争的流得到平等的待遇（但不必使用比它们需要的更多带宽）。

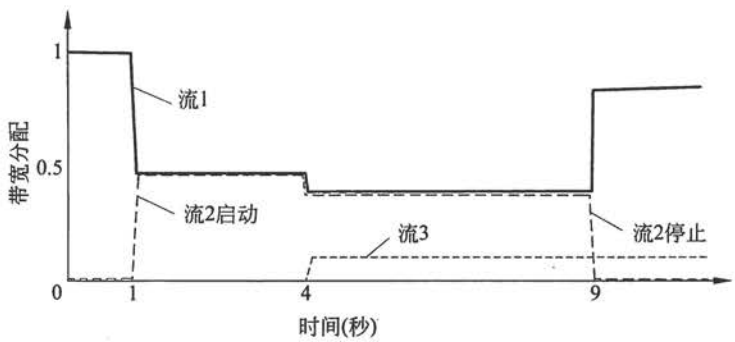


图 6-21 随着时间的推移而变化的带宽分配

6.3.2 调整发送速率

现在是时候进入主要的课程学习了。我们如何调整发送速率，以便获得一个理想的带

宽分配？发送速率可能受到两个方面因素的限制。首先是流量控制，在接收端没有足够缓冲区的情况下，必须进行流量控制。第二个因素是拥塞控制，在网络容量不足的情况下必须继续进行拥塞控制。在图 6-22 中，我们可以看到用液压说明的这个问题。在 6-22（a）中，一根粗粗的管道接到一个小容量的接收器。这是流量控制限制的情况。只要发送端不发送比漏桶能容纳的更多水，水就不会因溢出而丢失。在图 6-22（b）中，限制因素不是桶的容量，而是网络内部的承载能力。如果太多的水太快地注入，它就会回流，因而造成丢失（在这种情况下，水从漏斗溢出）。

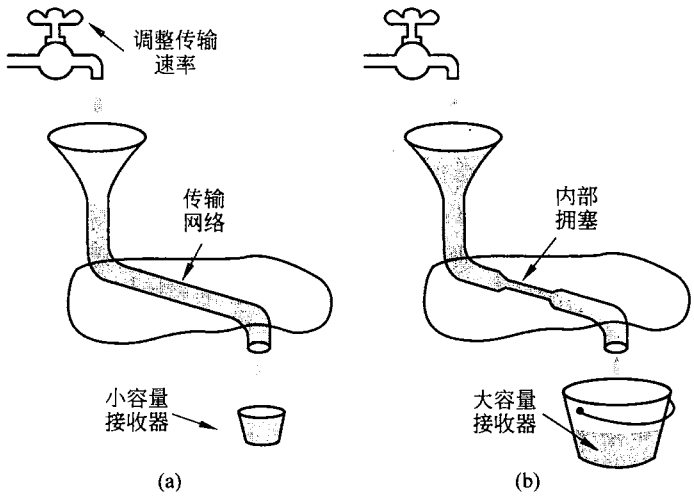


图 6-22
（a）一个快速网络往小容量接收器中注水；（b）一个小容量网络往大容量接收器中注水

类似的这种情况可能会出现在发送端，由于传输速度过快导致数据包丢失。然而，它们有不同的原因，因而需要不同的解决方案。我们已经讨论过采用一个可变大小窗口的流量控制解决方案。现在，我们考虑拥塞控制解决方案。由于上述这些问题都可能发生，所以传输协议一般需要同时运行两种解决方案，并且当其中一个问题发生时放慢发送速度。

传输协议调节发送速率的方式依赖于网络返回的反馈意见形式。不同的网络层可能会返回不同类型的反馈。反馈可能是显式的，也可能是隐式的；而且反馈信息可能精确，也可能不精确。

一个显式并精确的设计例子是路由器告诉数据包的源端，它们能以多大的速率发送数据包。在文献资料中，**显式拥塞协议**（XCP，eXplicit Congestion Protocol）就是以这种方式工作的（Katabi 等，2002）。另一个显式的但不精确的设计例子是使用 TCP 的**显式拥塞通知**（ECN，Explicit Congestion Notification）。在这个设计方案中，路由器在经历拥塞的数据包中设置警告比特来警告发送端放慢速率，但它们不告诉源端该减缓多少。

在其他设计方案中，没有明确的信号。FAST TCP 测量端到端往返延迟，并用该指标作为避免拥塞的信号（Wei 等，2006）。最后，在今天的 Internet 上最流行的拥塞控制形式是带有尾丢包的 TCP 或 RED 路由器，数据包的丢失要通过推断得出，并且丢包被当作网络是否变得拥挤的信号。这种形式的 TCP 有许多变种，包括 Linux 系统下采用的 CUBIC TCP（Ha 等，2008）。组合信号也是有可能的。例如，Windows 的 Compound TCP 同时使用了丢

包和延迟作为拥塞的反馈信号（Tan 等，2006）。图 6-23 给出了这些设计思想的概况。

协议	信号	显式否	精确否
XCP	使用速率	是	是
TCP	拥塞警告	是	否
FAST TCP	端到端延迟	否	是
Compound TCP	数据包丢失&端到端延迟	否	是
CUBIC TCP	数据包丢失	否	否
TCP	数据包丢失	否	否

图 6-23 一些拥塞控制机制的信号

如果给出一个显式的和精确的信号，那么传输实体就可以利用该信号来调整发送速率到新的操作点。例如，如果 XCP 告诉发送端使用多大的发送速率，那么发送端只需简单地采用该速率来发送数据包。然而，在其他情况下，涉及一些猜测工作。在没有拥塞信号的情况下，发送端应该增加发送速率；当给出拥塞信号时，发送端应降低发送速率。增加或减少速率的方式由**控制法则**（control law）决定。这些法则对性能有重大的影响。.

（Chiu 和 Jain，1989）研究了二进制拥塞反馈的情况，他们得出的结论是**加法递增乘法递减**（AIMD，Additive Increase Multiplicative Decrease）法则是达到有效和公平操作点的适当流量规则。为了表明这种情况，他们构建了一个简单实例的图形表示，在这里有两个流竞争单条链路的带宽。在图 6-24 中，分配给用户 1 的带宽用 x 轴表示，分配给用户 2 的带宽用 y 轴表示。当分配满足公平性时，两个用户将获得相同数量的带宽。这在图中用一条虚的公平线表示。当分配的带宽总和为 100%时，即达到链路容量时分配是有效的，这在图中用一条虚线效率线表示。当两个用户分配的带宽总和超过这条线时，网络就给两个用户发出一个拥塞信号。这些线的交点就是理想的操作点，此时两个用户具有相同的带宽，并且所有的网络带宽都被使用上。

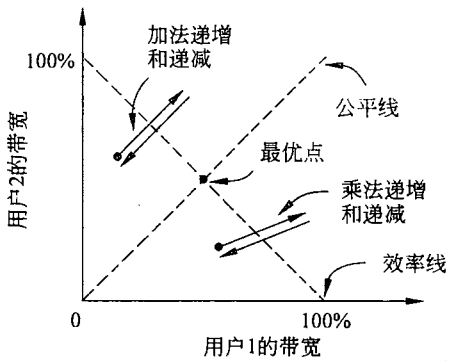


图 6-24 加法递增乘法递减的带宽调整

如果用户 1 和用户 2 都随着时间推移按加法递增法则增加各自的带宽，我们考虑从某些时候开始带宽的分配会出现什么情况。例如，两个用户以每秒增加 1 Mbps 的速度递增各自的发送速率。最终，操作点穿过效率线，两个用户都收到一个从网络传来的拥塞信号。在这个阶段，他们必须减少各自分配的带宽。然而，加法递减只会导致他们沿着递增线振荡。这种情况如图 6-24 所示。这种行为将保持操作点接近效率线，但它并不一定是公平的。

同样，考虑当两个用户随着时间的推移按乘法（加倍）法则递增带宽直到他们收到一个拥塞信号的情况。例如，用户可能每秒增加 10%的发送速率。然后，如果他们乘法递减各自的发送速率，用户的操作点将沿着乘法线振荡。这种行为也如图 6-24 所示。乘法线的斜率和加法线的斜率不同 （它指向源点，而加法线呈 45°角）。不过它也没有更好的选择。在这两种情况下，用户都难以收敛到同时兼顾公平与效率的最佳发送速率。

现在考虑这样的情况，用户加法递增他们的带宽，然后当收到拥塞信号时乘法递减他

们的带宽。这种行为就是 AIMD 控制法则，如图 6-25 所示。由此可以看出，这种行为的轨迹路径能收敛到兼顾公平与效率的最佳点。这种收敛不管从什么出发点开始都能发生，因而使得 AIMD 得到了广泛使用。同样的论点，唯一的其他组合，即乘法递增和加法递减，将会偏离最佳点。

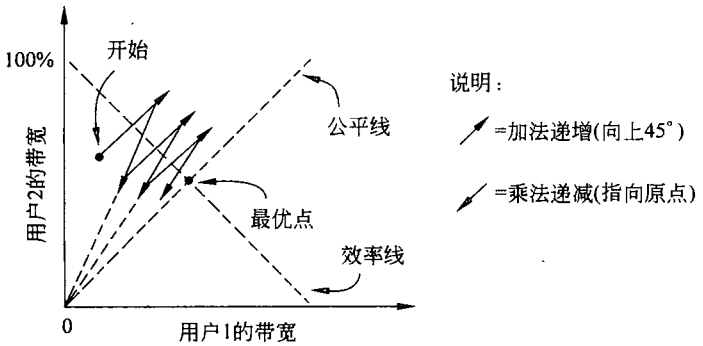


图 6-25 加法递增乘法递减（AIMD）控制法则

AIMD 是 TCP 采用的拥塞控制法则，它基于这个观点和另一个稳定性观点（即驱使网络拥堵非常容易而从中恢复却很难，所以递增政策应轻柔，而递减政策应积极）。但这个法则不是那么的公平，因为 TCP 连接根据每次的往返时间测量值来调整窗口的大小，而不同的连接有不同的往返时间。这导致在所有其他条件都相同的情况下，接近主机的连接比远离主机的连接获得的带宽更多。

在 6.5 节，我们将详细描述 TCP 是如何实现 AIMD 控制法则来调整发送速率并实行拥塞控制的。这个任务做起来比听起来要困难得多，因为要在某个时间间隔测量速率，而且流量是突发的。实际上通常使用的策略不是直接调整发送速率，而是调整滑动窗口的大小。TCP 就是使用了这种策略。如果窗口大小是 W，往返时间是 RTT，则等价的发送速率是 W/RTT。这种策略很容易和流量控制机制结合起来，因为流量控制机制已经使用了一个窗口。同时这种策略还有个优点，发送端可以通过确认来测定数据包的速度，而且如果它停止接收数据包离开网络的报告那么还可以放缓 RTT。

最后一个问题是关于多协议的竞争。网络中可能存在许多不同的传输协议，都往网络发送流量。为了避免拥塞，不同的协议采用了不同的控制法则，如果这些不同的协议竞争时会发生什么？那就是带宽分配不平等。由于 TCP 是 Internet 拥塞控制的主要形式，因此设计新的传输协议要承受相当显著的社会压力，新协议必须能与 TCP 进行公平的竞争。早期的流媒体协议就是因为产生了不公平竞争问题，结果是过度减少了 TCP 吞吐量。这种现象导致了 TCP 友好（TCP-friendly）拥塞控制这一概念的提出。在 TCP 友好的拥塞控制中，TCP 和非 TCP 传输协议可以自由地混合，而没有任何不良影响（Floyd 等，2000）。

6.3.3 无线问题

诸如 TCP 这样的传输协议实现拥塞控制应该独立于下面的网络层和链路层技术。理论上这是好的，但在实际的无线网络中却存在问题。主要的问题在于通常的传输协议一般把丢包视作拥塞发生的信号，包括我们刚刚讨论的 TCP 也是这样认为的。而在无线网络中，

丢包几乎都是因为传输错误引起的。

采用 AIMD 控制法则，较高的吞吐量要求非常小的数据包丢失。（Padhye 等，1998）分析表明吞吐量作为丢包率的逆平方根上升。在实践中这意味着快速 TCP 连接的丢失率非常小；1%是中等程度的丢失率，待丢失率达到 10%时连接就几乎停止工作了。然而，无线网络，比如 802.11 局域网，至少 10%的帧丢失率是很常见的情况。这种差异意味着如果没有保护测量措施，把丢包率作为一个信号的拥塞控制方案将把运行在无线链路上的连接压制到非常低的速率，而这显然是没有必要的。

为了在无线网络中也能正常工作，拥塞控制算法观察到的丢包只能是那些因带宽不足造成的丢包，而不能把由于传输错误造成的丢包也算在内。这个问题的一种解决方案是通过使用无线链路上的重传机制把无线链路的丢包掩盖起来。例如，802.11 使用停-等式协议来传递每个帧，在向上层报告丢包之前，如果需要会进行多次重传。在正常情况下，每个数据包的交付对高层是不可见的，尽管可能曾经发生过瞬间的传输错误。

图 6-26 显示了一条由有线和无线链路组成的路径，传输层采用了屏蔽策略。有两个方面需要注意。首先，发送端并不一定要知道该路径包括一条无线链路，因为它所能看到的只是与它连接的有线链路。Internet 上的路径是异构的，而且不存在一种通用的方法用来告知发送端路经由什么链路组成。这样的现状加深了拥塞控制问题的复杂性，因为没有简单的方法使得无线链路采用一个协议而有线链路采用另一个协议。

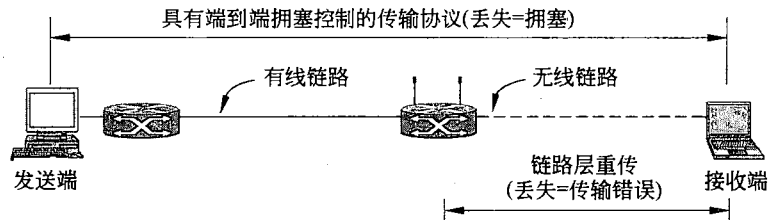


图 6-26 针对具有无线链路的路径的拥塞控制

第二个方面是一道谜。该图显示了由丢失驱动的两个机制：链路层的帧重传和传输层的拥塞控制。让人困惑的是这两种机制如何并存而不会混淆。毕竟，丢失应该只引起一个机制采取行动，因为它要么是一次传输错误，要么是一个拥塞信号。它不能同时是这两个原因造成的。如果两种机制都采取行动（重传帧并且减慢发送速率），那么我们又回到了原先的传输问题，即在无线链路上的运行过于缓慢。考虑一下这个难题，看看你是否能找到这个问题的解决方案。

解决的办法是这两种机制作用在不同的时间尺度。对于诸如 802.11 的无线链路，链路层的重传发生在微秒到毫秒级；而传输层的丢失计时器发生在毫秒到秒。区别就在于这三个时间尺度。这样的话，在传输层推测出丢包之前，无线链路已经检测出丢帧并重传帧，由此修复了传输错误。

屏蔽策略足以让大多数传输协议在大多数无线链路上运行得良好。但是，这并不总是一个合适的解决方案。某些无线链路的往返时间很长，比如卫星。对于这些链路，必须使用其他技术来掩盖丢包事实，比如**前向纠错**（FEC，Forward Error Correction），或者传输协议在拥塞控制时必须使用一种非丢失（non-loss）信号。

与无线链路上的拥塞控制有关的第二个问题是链路的容量可变。也就是说，随着时间的推移无线链路的容量会发生变化，而且因为节点的移动或者信道条件的变化而引起的信噪比的变化，链路容量的变化有时非常突然。这点与有线链路有很大的不同，有线链路的容量是固定不变的。传输协议必须适应无线链路容量的变化，否则将阻塞网络，或者无法使用可用的链路容量。

这个问题的一个可能解决办法是不解决，也就是说根本不必担心这个问题。这种策略其实是可行的，因为拥塞控制算法早就应该能处理新用户进入网络或现有用户改变发送速率等这些情况。即使有线链路的容量是固定的，其他用户不断变化的行为本身也会对给定用户的可用带宽带来变化。因此，在一条具有 802.11 无线链路的路径上简单地运行 TCP，有可能获得合理的性能。

然而，当存在太多的无线变异时，继续使用专为有线链路设计的传输协议可能会有麻烦，并且由此带来的性能比较差。在这种情况下，唯一的解决方案是专门设计一个针对无线链路的传输协议。一个特别具有挑战性的环境是无线网状网。在这样的网络中，多个相互干扰的无线链路彼此交错，路由因移动性而不断地发生着变化，并且存在大量的丢包。这方面的研究正在进行之中。（Li 等，2009）给出了一个无线传输协议设计的例子。

6.4 Internet 传输协议：UDP

Internet 的传输层有两个主要协议，无连接和面向连接各一个，两个协议互为补充。无连接协议是 UDP，它除了给应用程序提供发送数据包功能并允许它们在所需的层次之上架构自己的协议之外，几乎没有做什么特别的事情。面向连接的协议是 TCP，该协议几乎做了所有的事情。它建立连接，并通过重传机制增加了可靠性，同时还进行流量控制和拥塞控制，代表使用它的应用程序做了所有的一切。

在下面的章节中，我们将学习 UDP 和 TCP。我们从 UDP 开始，因为它是最简单的传输协议。我们也会考察 UDP 的两种应用模式。由于 UDP 是传输层协议，通常运行在操作系统中，而使用 UDP 的协议通常运行在用户空间，因而可以把对 UDP 的使用视为应用程序。然而，它们的使用技术对许多应用都有用，而且把这些技术视作属于传输层更好，所以这里我们将介绍它们。

6.4.1 UDP 概述

Internet 协议集支持一个无连接的传输协议，该协议称为**用户数据报协议**（UDP，User Datagram Protocol）。UDP 为应用程序提供了一种无需建立连接就可发送封装的 IP 数据报的方法。RFC 768 描述了 UDP。

UDP 传输的**段**（segment）由 8 字节的头和有效载荷字段构成。图 6-27 描述了头信息。两个端口（port）分别用来标识源机器和目标机器内部的端点。当一个 UDP 数据包到来时，它的有效载荷被递交给与目标端口相关联的那个进程。在调用了 BIND 原语或者其他某种类似的原语之后，这种关联关系就建立了起来，正如我们在图 6-6 中介绍 TCP 时已经看过

的那样（UDP 的绑定过程是一样的）。可以把端口看作应用程序租来接收数据包的邮箱。在我们描述同样使用端口的 TCP 时，还将对此多说几句。实际上，采用 UDP 而不是原始 IP 的最主要价值在于增加了源端口和目标端口。如果没有端口字段，传输层将无从知道如何处理每个入境数据包；而有了端口字段之后，它就能把内嵌的段递交给正确的应用程序处理。

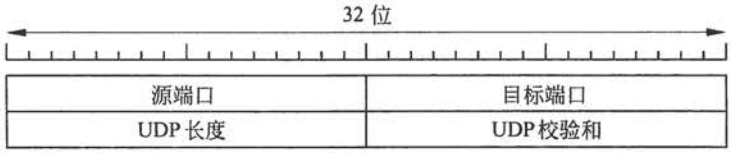


图 6-27 UDP 头格式

当接收端必须将一个应答返回给源端时，源端口（Source Port）字段则是必需的。只要将入境段中的源端口字段复制到出境段中的目标端口（Destination Port）字段，接收端就指定了由发送端机器上的哪个进程来接收应答。

UDP 长度（UDP Length）字段包含 8 字节的头和数据两部分的总长度。最小长度是 8 个字节，刚好覆盖 UDP 头。最大长度是 65 515 字节，恰好低于填满 16 比特的最大字节数，而这是由 IP 数据包限制的。

一个可选的校验和（UDP Checksum）字段还提供了额外的可靠性。它校验头、数据和一个概念性的 IP 伪头。执行校验和计算时，校验和字段先被设置为零，如果数据字段的长度是奇数则用零填充成偶字节。校验和算法很简单，先按 16 位字的补码相加求和，然后再取总和的补码。因此，当接收端对整个段计算校验和时，要包括 UDP 校验和字段，正确的结果应该为 0。如果发送端没有计算校验和，则将该字段值填为 0，因为补码计算结果可能碰巧真的是 0，则存储为全 1。然而，关闭校验和计算不是明智之举，除非数据传输的质量并不重要（例如，数字化语音）。

IPv4 的伪头部如图 6-28 所示，它包含源机器和目标机器的 32 位 IP 地址、UDP 的协议号（17），以及 UDP 段（包括头）的字节计数。IPv6 的伪头与其类似，但有些许不同。在 UDP 校验和计算中包含伪头将有助于检测出被错误递交的数据包，但是传输层在计算校验和时需要地址的做法违反了协议分层原则，因为 IP 地址属于 IP 层，而不属于 UDP 层。TCP 在计算校验和的时候也使用了同样的伪头。

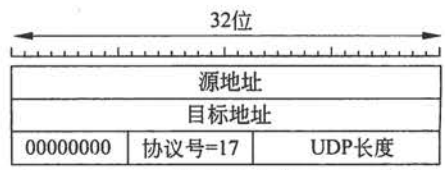


图 6-28 UDP 校验和包括了 IPv4 伪头

或许值得明确一提的是 UDP 没有做的事情。它没有流量控制、拥塞控制，或者接收到一个坏段后的重传机制。所有这一切都必须由用户进程来完成。它只是提供了一个与 IP 协议的接口，并在此接口上增加通过端口号复用多个进程的功能，以及可选的端到端错误检测功能。这就是 UDP 所做的一切。

对于需要对数据包流实施精确控制、错误检测或者时序功能的应用，UDP 提供的服务恰到好处。UDP 协议特别有用的一个领域是客户机-服务器应用开发。一般情况下，客户端向服务器发送一个简短的请求报文，并期待来自服务器的简短回复报文。如果请求或回复报文丢失，客户端就会超时，然后再试一次。以 UDP 实现的代码相比需要初始建立连接的协议（比

如 TCP）不仅代码简单，而且需要交换的报文也少（对于连接来说，每个方向都需要一份）。

以这种方式使用 UDP 的一个应用是**域名系统**（DNS，Domain Name System），我们将在第 7 章中学习这个系统。简单地说，如果一个程序需要查询某个主机名的 IP 地址，比如 www.cs.berkeley.edu，那么它可以给 DNS 服务器发送一个包含该主机名的 UDP 数据包。服务器用一个包含了该主机 IP 地址的 UDP 数据包作为应答。事先不需要建立连接，事后也不需要释放连接。只有两条消息通过网络就够了。

6.4.2 远程过程调用

在某种意义上，向一台远程主机发送一个消息并获得一个应答，就如同在编程语言中执行一个函数调用一样。在这两种情形下，启动时都需要提供一个或者多个参数，然后获得一个结果。这种观察导致人们试图将网络上的请求-应答交互过程安排成像过程调用那样。这样的安排使得网络应用更加易于编程也更为人们所熟悉。例如，请想象一个名为 get_IP_address（host_name）的过程，它的工作方式是向 DNS 服务器发送一个 UDP 数据包，然后等待应答；如果应答返回的速度不够快，则超时并重试。通过这种方式，向程序员隐藏了网络的所有细节。

这个领域中的关键工作是由（Birrell 和 Nelson，1984）完成的。一言以蔽之，Birrell 和 Nelson 的建议是允许本地程序调用远程主机上的过程。当机器 1 上的进程调用机器 2 上的一个过程时，机器 1 上的调用进程被挂起，而机器 2 上的被调用过程则开始执行。信息以参数的形式从调用方传输到被调用方，而过程的执行结果则从反方向传递回来。应用程序员看不到任何消息的传递。这项技术称为**远程过程调用**（RPC，Remote Procedure Call），目前已经成为许多网络应用的基础。传统上，调用过程称为客户，被调用过程称为服务器，这里我们将沿用这些名字。

RPC 背后的思想是尽可能地使一个远程过程调用看起来像本地过程调用一样。在最简单的形式中，为了调用一个远程过程，客户程序必须绑定（链接）到一个小的库过程，这个库过程称为**客户存根**（client stub），它代表了客户地址空间中的服务器过程。类似地，服务器需要绑定到一个称为**服务器存根**（server stub）的过程。正是这些过程，隐藏了从客户机不在本地调用服务器的事实。

图 6-29 给出了执行 RPC 的实际步骤。第 1 步是客户调用客户存根。这是一个本地过程调用，其参数按照常规的方式压入到栈中。在第 2 步，客户存根将参数封装到一个消息中，然后通过系统调用发送该消息。封装参数的过程称为**列集**（marshalling）。在第 3 步，操作系统将消息从客户机器发送到服务器机器上。在第 4 步，操作系统将入境数据包传递给服务器存根。最后，在第 5 步中，服务器存根利用**散集**（unmarshaled）得到的参数调用服务器过程。调用的结果沿着相反的方向按同样的路径传递。

这里需要注意的关键点是用户编写的客户过程只是按照普通过程调用的方式来调用客户存根，而且客户存根与服务器过程有同样的名字。由于客户过程和客户存根在同样的地址空间中，所以，参数的传递也是按通常的方式进行。类似地，服务器过程被同一地址空间中的一个过程调用，并且它接收到的也正好是它所期望的参数。按照这种方式，网络通信并不是在套接字上完成输入和输出，而是通过仿造一个普通的过程调用来完成的。

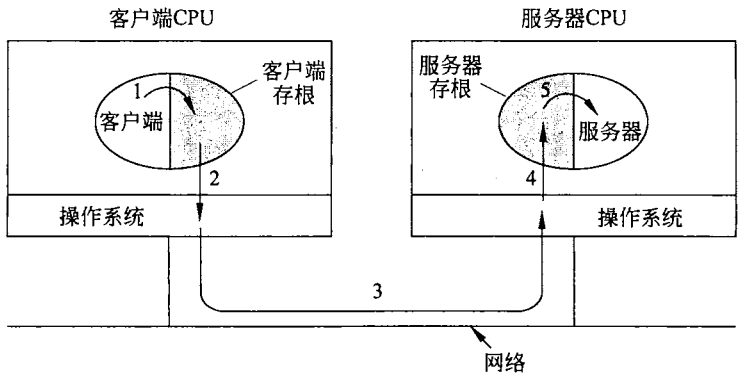


图 6-29 远程过程调用的步骤（阴影部分表示存根）

不管 RPC 的概念有多么的精巧，这其中还隐藏着一些陷阱。最大的陷阱是指针参数的用法。在正常情况下，将指针传递给一个过程不是问题。被调过程可以像调用方那样使用这个指针，因为双方生存在同样的虚拟地址空间中。而对于 RPC，传递指针是不可能的，因为客户和服务器位于不同的地址空间中。

在有些情况下，可以使用一些技巧来实现指针的传递。假设第一个参数是一个指向某个整数 k 的指针。客户存根可以对 k 进行列集，并发送给服务器；然后，服务器存根创建一个指向 k 的指针，并且将该指针传递给服务器过程，而这正是服务器过程所期望的。当服务器过程将控制权返回给服务器存根时，后者将 k 送回给客户。在客户端，新的 k 值被复制到老的 k 中，就好像服务器对它做了修改一样。实际上，整个过程相当于用按**复制-恢复调用**（call-by-copy-restore）代替了按**引用调用**（call-by-reference）的标准调用序列。不幸的是，这种技巧并不总是能正确地工作，例如，若指针指向一个图形或者其他复杂的数据结构。由于这个原因，对于远程调用的参数必须强加一些限制。

第二个问题是在一些弱类型的语言中（比如 C），编写一个计算两个矢量（即数组）内积，但不指定矢量大小的过程是完全合法的。每个矢量都有一个专门的值作为终止标记，而且该终止值只有调用过程和被调用过程才知道。在这种情况下，客户存根要想对参数执行列集操作从本质上讲是不可能的：它无从知道该如何确定参数的长度。

第三个问题在于并不是总能推断出参数的类型，即使从正式的规范或者代码本身都未必能做得到。一个例子是 printf，它可能有任意多个参数（至少一个），而且这些参数可以是整数、短整数、长整数、字符、字符串、可变长度的浮点数和其他类型的任意组合。试图以远程过程的方式来调用 printf 不太现实，因为 C 语言对类型的要求特别宽松。然而，有一条规则是这样的：只要不用 C（或者 C++）编写程序，那么就可以使用 RPC，但这种说法没有得到程序员的广泛认可。

第四个问题涉及全局变量的使用。正常情况下，调用过程和被调用过程除了通过参数进行通信以外，也可以使用全局变量作为通信的手段。如果被调用的过程被转移到了远程机器上，那么，这样的代码就会失败，因为全局变量无法再为双方所共享了。

这些问题并不意味着 RPC 就没有希望。事实上，它的应用仍然十分广泛，但是在实际运用中，需要有一些限制才能保证它正常工作。

关于传输层协议，UDP 是一个实现 RPC 的良好基础。请求和应答可作为单个 UDP 数

据包以最简单的形式发送，并且可以快速进行操作。然而，RPC 的实现必须包括其他机器。因为请求或应答数据包可能会丢失，客户端必须维持一个计时器用来重发请求。请注意，应答数据包可当作一个请求的隐含确认，因此该请求不再需要单独确认。某些时候，参数或结果可能大于最大的 UDP 数据包大小，在这种情况下需要传递大块消息的某种协议。如果多个请求和应答可以重叠（在并发编程情况下），则需要一个标识符把应答和请求相匹配。

一个更高层次的关注是操作可能不是幂等的（即重复安全）。一个简单的幂等操作例子是 DNS 请求和应答。客户如果没有收到答复，可以放心地一次又一次地重发请求，无须担心服务器是否永远收不到请求，或者返回的应答报文会丢失。当应答报文终于来临时，答案将是相同的（假设在此期间 DNS 数据库没有更新）。然而，并非所有的操作都是幂等的，因为它们有严重的副作用，比如递增一个计数器。对于这些非冥等操作 RPC 需要较强的语义，以便当程序员调用一个过程时它不会被多次执行。在这种情况下，可能有必要建立一个 TCP 连接并在连接上发送请求，而不是使用 UDP 发送请求报文。

6.4.3 实时传输协议

客户-服务器 RPC 是 UDP 被广泛应用的一个领域，UDP 的另一个应用领域是实时多媒体应用。尤其是，随着 Internet 广播电台、Internet 电话、音乐点播、视频会议、视频点播和其他的多媒体应用越来越普及，人们发现每一种应用都在或多或少地重复设计几乎相同的实时传输协议。逐渐地人们意识到，为多媒体应用制定一个通用的实时传输协议是一种很好的想法。

于是**实时传输协议**（RTP，Real-time Transport Protocol）诞生了。RTP 由 RFC 3550 描述，目前已经广泛应用于多媒体应用程序。我们将描述实时传输的两个方面。第一个是 RTP 协议，它以数据包形式传输音频和视频数据。第二个主要是接收端的处理，涉及在正确的时间播放音频和视频。这些功能在协议栈中的位置如图 6-30 所示。

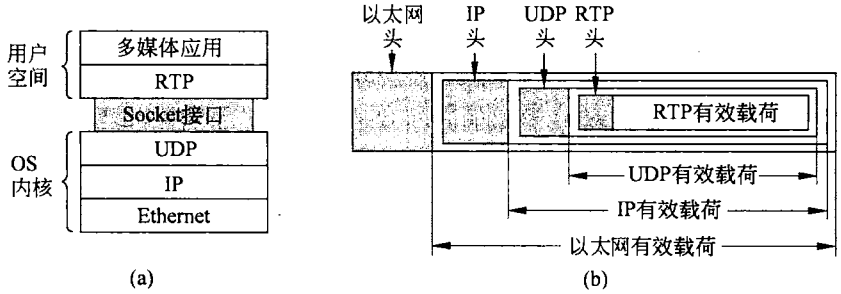


图 6-30
（a）RTP 在协议栈中的位置；（b）数据包的嵌套

RTP 通常运行在用户空间，位于 UDP 之上。它的操作方式如下所述。多媒体应用由多个音频、视频、文本或可能其他流组成。这些流被送入到 RTP 库中，RTP 库和多媒体应用一起都位于用户空间。然后，RTP 库将这些流复用并编码到 RTP 数据包中，RTP 数据包再被塞进到一个套接字中。在套接字的操作系统那一端，RTP 数据包被封装到新生

成的 UDP 数据包，这些 UDP 数据包被交给 IP，以便发送到链路上，比如以太网。在接收端的处理与发送端的处理次序相反。最终多媒体应用从 RTP 库函数接收到多媒体数据，它负责播放所收到的媒体。这种情况的协议栈如图 6-30（a）所示。数据包的嵌套封装情况如图 6-30（b）所示。

这种设计的后果是很难说清楚 RTP 位于哪一层。由于它运行在用户空间，并且与应用程序链接，所以它无疑看起来像是一个应用协议。另一方面，它又是一个与具体应用无关的通用协议，它仅仅提供了一些传输设施，所以它看起来也像一个传输协议。可能最恰当的描述是：它是一个在应用层上实现的传输协议，这就是为什么我们在本章涉及它的原因。

RTP—实时传输协议

RTP 的基本功能是将几个实时数据流复用到一个 UDP 数据包流中。这个 UDP 流可以被发送给一台目标主机（单播传输模式），也可以被发送给多台目标主机（组播传输模式）。因为 RTP 仅仅使用了常规的 UDP，所以路由器不会对它的数据包有任何特殊的对待，除非开通了某些通常的 IP 服务质量特性。特别地，这里没有任何保障传递可靠性的措施，数据包可能会丢失、延迟或者损坏等。

RTP 格式包含了几种有助于接收端处理流媒体信息的特性。在 RTP 流中发送的每个数据包被赋予一个编号，而且每个数据包的编号比它前面的数据包编号大 1。这种编号使得接收方能够确定是否有数据包丢失。如果一个数据包丢失，则接收方能够采取的最佳动作是交给应用程序来处理。如果数据包携带的是视频数据，或许可以跳过该视频帧；如果数据包携带的是音频数据，或许可以利用插值法近似地估计出已丢失的值。重传不是一种切合实际的选择方式，因为重传的数据包可能到达得太晚以至于不再有用。因此，RTP 没有确认，也没有请求重传的机制。

每个 RTP 有效载荷可能包含多个样本，它们可以按照应用系统希望的任何一种方式进行编码。为了允许网络互连，RTP 定义了几种配置轮廓（比如单音频流）；而且对于每一种轮廓，又可以允许多种编码格式。例如，一个单音频流的编码方式包括使用增量编码以 8 kHz 的 8 位 PCM 采样、预测编码、GSM 编码、MP3 编码等。RTP 在头中提供了一个字段让源端用来指定编码方法；除此以外，RTP 不涉及如何进行编码。

许多实时应用需要的另一种设施是时间戳机制。这里的想法是允许源端将每个数据包中时间戳与第一个样本关联起来。这里的时间戳是相对于整个流的起始时间，因此，只有时间戳之间的差值才是重要的，时间戳的绝对值没有任何意义。就像我们马上要描述的那样，这种机制使得接收方可以做少量的缓冲工作，然后在整个流开始之后的正确毫秒时间点上播放每一个样本，并且独立于每个样本所在数据包的到达时间。

时间戳机制不仅减小了网络延迟变化的影响，而且还允许在多个流之间实行同步。例如，一个数字电视节目可能有一个视频流和两个音频流。两个音频流中的内容可能是立体声广播，也可能是电影节目中的两个语言声道（一个声道是原始语种的声音，另一个声道是本地语言的配音），从而让观众有选择语言的机会。每个流分别来自于不同的物理设备；但是，如果它们的时间戳始于同一个计数器，那么，即使这些流的传输和/或者接收过程稍微有一点不规律，它们仍然可以非常同步地播放出来。

RTP 的头结构如图 6-31 所示。它包含 3 个 32 位的字，并有潜在的一些扩展。第一

个字包含了版本字段，现在版本号已经达到了 2。我们希望这个版本非常接近于最终的版本，因为这里只剩下一个值了（虽然可以将 3 定义成说明版本号在一个扩展字中的特殊含义）。

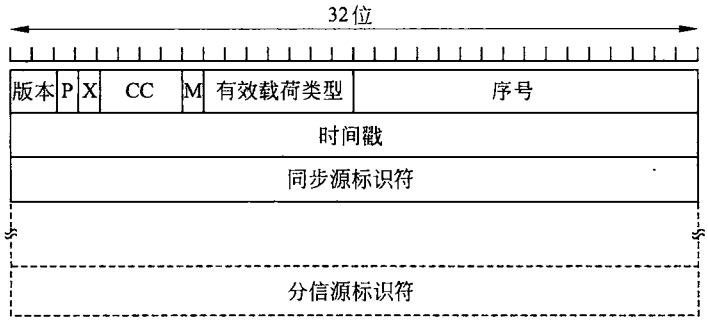


图 6-31 RTP 头格式

P 位表示该数据包被填充到了 4 字节的倍数。最后的填充字节指明了有多少个字节被填充进来。X 位表示有一个扩展头。扩展头的格式和含义没有定义。唯一定义的事情是扩展头的第一个字给出了扩展头的长度。这是针对任何不可预见之需求的最后退路。

CC 字段指明了后面共有多少个贡献源（contributing source），可以从 0 到 15（见下面的说明）。M 位是一个与应用相关的标记位，可以用来标记一个视频帧的开始、音频信道中一个字的开始，或者其他由应用解释的某些事情。有效载荷类型（Payload type）说明使用了哪一种编码算法（比如未压缩的 8 位音频、MP3 等）。由于每个数据包都带有这个字段，所以在传输过程中可以改变编码方法。序号（Sequence number）只是一个计数器，每发送一个 RTP 数据包该计数器都要递增。该字段可被用来检测数据包的丢失。

时间戳（Timestamp）由媒体流的源端产生，它注明了数据包中第一个样本什么时候产生的。这个值有助于减缓接收端的时间变化——称为**抖动**（jitter），具体做法是将播放与数据包的到达时间分离开。同步源标识符（Synchronization source identifier）指明了该数据包属于哪一个流。正是通过这个字段，可以将多个数据流复用到一个 UDP 数据包流中，或者从一个 UDP 数据包流中分用出多个数据流。最后，如果在演播室中使用了混合器，则要使用贡献源标识符（Contributing source identifier）。在这种情况下，混合器是同步源，被混合的流就列在这里（即贡献源标识符）。

RTCP——实时传输控制协议

RTP 有一个姊妹协议，称为**实时传输控制协议**（RTCP，Realtime Transport Control Protocol）。它和 RTP 一起由 RFC 3550 描述。RTCP 能处理反馈、同步和用户接口等信息，但不传输任何媒体样值。

RTCP 的第一个功能是可以向源端提供有关延迟、抖动、带宽、拥塞和其他网络特性的反馈信息。编码进程充分利用了这些信息，当网络状况比较好的时候，它提高数据速率（从而达到更好的质量）；而当网络状况不好时候，它减低数据速率。根据连续的反馈信息，编码算法可以不断地调整，从而在当前条件下提供尽可能最好的质量。例如，如果在传输过程中带宽增加或者减少，那么编码算法可以根据需要从 MP3 切换到 8 位 PCM，或者切换到增量编码。利用有效载荷类型字段可以告诉接收方当前数据包使用的是哪一种编码算

法，因而有可能根据需要动态地改变编码算法。

提供反馈信息的一个问题是要求所有参与者发送 RTCP 报告。对于一个规模较大的组播应用，RTCP 协议所使用的带宽会变得很大。为了防止这种情况发生，RTCP 发送端缩减其报告率，使得总的带宽不超过媒体带宽的 5%。要做到这一点，每个参与者需要了解媒体的带宽和参与者的数目。对于媒体带宽，它可从发送端那里学习到这点；对于参与人数，它可以通过监听其他参与者的 RTCP 报告估算出来。

RTCP 也处理流之间的同步。问题在于不同的流有可能使用不同的时钟，并且有不同的粒度和不同的漂移速率。利用 RTCP 可使它们保持同步。

最后，RTCP 还提供了一种命名不同源的方法（比如使用 ASCII 字符）。这些信息可以显示在接收端的屏幕上，告诉用户此刻在跟谁通话。

有关 RTCP 的更多信息，请参考（Perkins，2003）。

带有缓冲和抖动控制的播放

一旦媒体信息到达接收端，它必须在合适的时间播放出来。一般情况下，这个时间不是 RTP 包到达接收端的时间，因为数据包通过网络传输所需要的时间略有不同。即使在发送端，数据包以正确的时间间隔被依次注入网络，它们到达接收端的相对时间也不同。这种延迟的变化称为**抖动**（jitter）。即使是少量的数据包抖动，如果简单地按它到达的时间播放出来，也可能导致媒体成品发散，如抖动的画面帧和难以辨认的音频。

这个问题的解决办法是在接收端播放媒体之前对其进行缓冲，以此来减少抖动。图 6-32 给出了一个实例，我们看到一个数据包流在传输过程中遭受了很大的抖动。在 t=0 时刻，服务器发出数据包 1，该数据包在 t=1 时刻到达客户端；随后的数据包 2 遭受了更多的延迟，它经过 2 秒才到达客户端。随着数据包的到达，它们均被缓冲在客户端机器上。

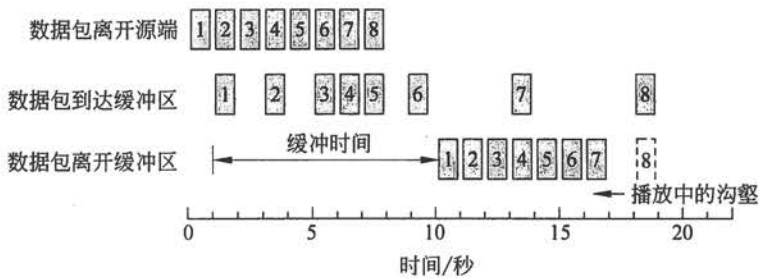


图 6-32 通过缓冲数据包来平滑输出流

在 t=10 秒时，播放器开始播放媒体。这时，从数据包 1 到数据包 6 都已经在缓冲区中，因此它们以平滑播放的统一间隔从缓冲区中移出。一般情况下，没有必要使用统一的间隔，因为 RTP 的时间戳告诉了播放器应该在何时播放相应的媒体。

不幸的是，我们可以看到数据包 8 被延迟得太多，当播放到它时还不可用。这种情况下可以有两种选择。第一种是跳过数据包 8 继续播放后续的数据包。另一种方法是停止播放，直到数据包 8 到达，此时在播放的音乐或电影中就会出现一个恼人的间隙。在现场直播的媒体应用中，比如语音 IP 电话，延迟的数据包通常会被跳过去。直播应用在暂缓播放时无法很好地工作。在流媒体应用中，播放器或许可以暂停。通过延迟媒体的开始播放时间，可以缓解这个问题，但为此需要使用较大的缓冲区。对于流式音频或视频播放器，经

常采用约 10 秒的缓冲区，以确保播放器按时接收所有数据包（即那些没有被网络丢弃的数据包）。对于像视频会议这样的实时应用，响应所需要的缓冲区较短。

平滑播出的一个主要考虑因素是**播放点**（playback point）设置在哪里，或接收器在播放媒体之前要等待多久。决定等待多长时间要取决于抖动。低抖动和高抖动连接之间的区别如图 6-33 所示。两者之间的平均延迟没有很大的不同，但如果存在高抖动，则播放点可能需要比低抖动多捕获超过 99%的数据包。

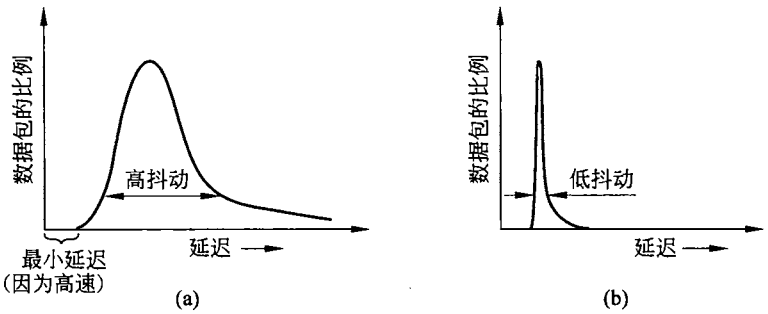


图 6-33
（a）高抖动；（b）低抖动

为了寻找好的播放点，应用程序可以测量抖动，即考查 RTP 时间戳和数据包到达时间之间的差异。每个差值给出了延迟的一个样本值（加上一个任意值，得到固定偏移）。然而，由于流量竞争和路由变化等其他原因，延迟可能随时改变。为了适应这种变化，应用程序在运行时可以自适应播放点。但是，如果做得不好，改变播放点可能会产生用户观察到的毛刺。对于音频来说，避免这个问题的一种方法是适应音频**会话峰**（talkspurts）之间的播放点，即交谈中的间隙。没有人会注意到一个短的和稍长一点静默之间的差异。RTP 协议允许应用程序为此目的设置 M 标志位，表明一个音频会话峰的开始。

如果数据包的绝对延迟太长，以至于媒体播放时还没到来，那么直播应用将受到影响。如果已经使用了直接路径，那么无论做什么都无法减少传播延迟。可以简单地拖曳住播放点不动，然后接受太晚到达的大部分延迟数据包。如果这种处理方式不能接受，那么拽住播放点的唯一途径是通过使用一种更优质的服务来减少抖动，例如，加速转发的区分服务。也就是说，需要更好的网络。

6.5 Internet 传输协议：TCP

UDP 是一个简单的协议，它有一些非常重要的用途，比如客户-服务器交互和多媒体应用，但是对于大多数 Internet 应用来说，它们需要可靠的、按序递交的传输特性。UDP 不能提供这样的功能，所以 Internet 还需要另一个协议。这就是 TCP，它是 Internet 上的主力军。现在让我们来详细学习这个协议。

6.5.1 TCP 概述

传输控制协议（TCP，Transmission Control Protocol）是为了在不可靠的互联网络上提

供可靠的端到端字节流而专门设计的一个传输协议。互联网络与单个网络有很大的不同，因为互联网络的不同部分可能有截然不同的拓扑结构、带宽、延迟、数据包大小和其他参数。TCP的设计目标是能够动态地适应互联网络的这些特性，而且具备面对各种故障时的健壮性。

TCP的正式定义由1981年9月的RFC 793给出。随着时间的推移，已经对其做了许多改进，各种错误和不一致的地方逐渐被修复。为了让你感受到TCP的扩展历程，现在重要的RFC有：RFC 793 plus：澄清了说明，RFC 1122修复了bug、RFC 1323做了高性能扩展，RFC 2018定义了选择性确认，RFC 2581说明了拥塞控制、RFC 2873定义了为服务质量而重用的头字段，RFC 2988改进了重传计时器，RFC 3168定义了显式拥塞通知。完整的协议集合很大，因而专门发布了一个针对许多RFC的指南，它就是作为另一个RFC文档公布的RFC 4614。

每台支持TCP的机器都有一个TCP传输实体。TCP实体可以是一个库过程、一个用户进程，或者内核的一部分。在所有这些情形下，它管理TCP流，以及与IP层之间的接口。TCP传输实体接受本地进程的用户数据流，将它们分割成不超过64 KB（实际上去掉IP和TCP头，通常不超过1460数据字节）的分段，每个分段以单独的IP数据报形式发送。当包含TCP数据的数据报到达一台机器时，它们被递交给TCP传输实体，TCP传输实体重构出原始的字节流。为简化起见，我们有时候仅仅用“TCP”来代表TCP传输实体（一段软件）或者TCP协议（一组规则）。根据上下文语义你应该能很清楚地推断出其实际含义。例如，在“用户将数据交给TCP”这句话中，很显然这里指的是TCP传输实体。

IP层并不保证数据报一定被正确地递交到接收方，也不指示数据报的发送速度有多快。正是TCP负责既要足够快地发送数据报，以便使用网络容量，但又不能引起网络拥塞；而且，TCP超时后，要重传没有递交的数据报。即使被正确递交的数据报，也可能存在错序的问题，这也是TCP的责任，它必须把接收到的数据报重新装配成正确的顺序。简而言之，TCP必须提供可靠性的良好性能，这正是大多数用户所期望的而IP又没有提供的功能。

6.5.2 TCP服务模型

TCP服务由发送端和接收端创建一种称为**套接字**（socket）的端点来获得，正如我们在6.1.3节中所讨论的那样。每个套接字有一个套接字编号（地址），该编号由主机的IP地址以及一个本地主机的16位数值组成的。这个16位数值称为端口（port），端口是TCP的TSAP名字。为了获得TCP服务，必须显式地在一台机器的套接字和另一台机器的套接字之间建立一个连接，有关套接字的调用如图6-5中所列。

一个套接字有可能同时被用于多个连接。换句话说，两个或者多个连接可能终止于同一个套接字。每个连接可以用两端的套接字标识符来标识，即（socket1, socket2）。TCP不使用虚电路号或者其他的标识符。

1024以下的端口号被保留，只能用作由特权用户（比如UNIX系统的root）启动的标准服务。这些端口称为**知名端口**（well-known port）。例如，如果一台机器上的任何一个进程希望检索邮件，那么它可以连接到目标主机的143号端口与IMAP守护进程联系。可以在www.iana.org上找到所有知名端口的列表。目前已经分配了700多个。图6-34列出了一

些尤为知名的端口。

端口	协议	用途
20, 21	FTP	文件传输
22	SSH	远程登录，Telnet的替代品
25	SMTP	电子邮件
80	HTTP	万维网
110	POP-3	访问远程邮件
143	IMAP	访问远程邮件
443	HTTPS	安全的Web（SSL/TLS之上的HTTP）
543	RTSP	媒体播放控制
631	IPP	打印共享

图 6-34 一些分配的端口号

1024～49 151 之间的其他端口可以通过 IANA 注册，由非特权用户使用，但是应用程序可以选择自己的端口号。例如，BitTorrent 对等文件共享应用（非正式的）使用了 6881～6887 端口号，但也可以运行在其他端口号上。

让 FTP 守护进程在系统启动时关联到 21 号端口，让 SSH 守护进程在系统启动时关联到 22 号端口等类似的做法是完全可能的。然而，这样做将会使内存散乱在这些守护进程中，而且大多数时间这些进程都是空闲的。因此，一般的做法是让一个守护进程同时关联到多个端口上，然后等待针对这些端口的第一个入境连接。这个特殊的守护进程在 UNIX 中称为 inetd（Internet daemon）。当出现一个入境连接请求，inetd 就派生出一个新的进程，在这个进程中调用适当的守护程序，然后由这个守护程序来处理连接请求。按照这种方式，除了 inetd，其他所有守护程序都只在确实有工作需要它们做时才被真正激活。inetd 通过一个配置文件知道哪个端口应该使用哪个守护程序。因此，系统管理员可以这样配置系统使得比较忙的端口（比如 80 端口）使用永久守护程序，而其他端口则让 inetd 来处理。

所有的 TCP 连接都是全双工的，并且是点到点的。所谓全双工，意味着同时可在两个方向上传输数据；而点到点则意味着每个连接恰好有两个端点。TCP 不支持组播或者广播传输模式。

一个 TCP 连接就是一个字节流，而不是消息流。端到端之间不保留消息的边界。例如，如果发送进程将 4 个 512 字节的数据块写到一个 TCP 流中，那么这些数据有可能按 4 个 512 字节块、2 个 1024 字节块、1 个 2048 字节块或者其他的方式被递交给接收进程（见图 6-35）。接收端不管多么努力尝试，都无法获知这些数据被写入字节流时的单元大小。

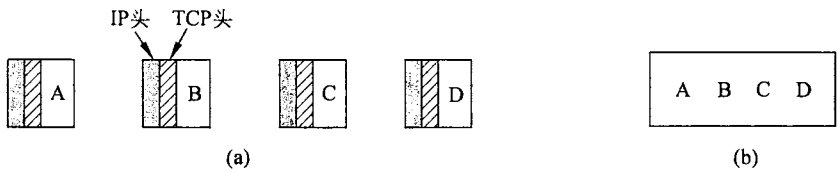


图 6-35

（a）4 个 512 字节段作为单独的 IP 数据报发送；（b）在一次 READ 调用中 2048 字节数据被递交给应用程序

UNIX 系统中的文件也具有这样的特性。读文件的程序无法判断该文件是一次写入一块，还是一次写入一个字节，或者整个文件被一次性写入的。如同 UNIX 文件一样，TCP 软件不理解字节流的含义，而且对此也不感兴趣。一个字节就只是一个字节而已。

当一个应用将数据传递给TCP时，TCP可能立即将数据发送出去，也可能将它缓冲起来（为了收集更多的数据一次发送出去），这完全由TCP软件自己来决定。然而，有时候，应用程序确实希望数据立即被发送出去。例如，假设一个交互式游戏的用户希望发送一个更新流，至关重要的是这个流应该被立即发送出去，而不是缓存起来直到收集到足够的数据后一起发送。为了强制将数据发送出去，TCP有个PUSH标志位概念，由数据包携带的PUSH标志原意是让应用告诉TCP不要延迟传输。然而，应用程序在发送数据时不能从字面上设置PUSH标志。相反，不同的操作系统已经演化出了不同的方案来加速传输（例如，在Windows和Linux的TCP_NODELAY）。

有关Internet的悠长历史，我们还要提到有关TCP服务的一个有趣特性，该特性还保留在协议中，但很少被使用，那就是**紧急数据**（urgent data）的处理。当一个具有高优先级数据的应用立即被处理时，例如，一个交互式用户键入Ctrl+ C来中断一个已经开始运行的远程计算时，发送端应用程序把一些控制信息放在数据流中，并且将它连同URGENT标志一起交给TCP。这一事件将导致TCP停止积累数据，将该连接上已有的所有数据立即传输出去。

当接收方接收到紧急数据时，接收端应用程序被中断（比如，按UNIX的术语是得到了一个信号），它停止当前正在做的工作，并且读入数据流以便找到紧急数据。紧急数据的尾部应该被标记出来，因而应用程序能够知道紧急数据的结束位置。紧急数据的起始处并没有被标记出来，如何找出紧急数据取决于具体的应用程序。

这个方案提供了一种略微粗糙的信号机制，而把一切留给应用程序来处理。然而，尽管紧急数据有潜在的应用价值，但在早期没有人发现它有令人信服的应用，因而遭到淘汰。因为这种信号机制让应用程序自己来处理，现在它的用途因为各种版本实现上的差异而令人气馁。也许未来的传输协议能提供更好的信号机制。

6.5.3 TCP协议

在这一节，我们将概述性地介绍TCP协议。在下一节我们将逐个讨论TCP协议头的每个字段。

TCP的一个关键特征，也是主导整个协议设计的特征是TCP连接上的每个字节都有它自己独有的32位序号。在Internet初期，路由器之间的线路绝大多数是56 kbps的租用线路，所以，一台满负荷全速运行的主机差不多一周才能遍历完所有的序号。以现代的网络速度，序号的消耗速度相当惊人，后面我们将会看到这一点。数据包携带的32位序号可用在一个方向上的滑动窗口以及另一个方向上的确认，正如下面我们讨论的那样。

发送端和接收端的TCP实体以段的形式交换数据。TCP段（TCP segment）由一个固定的20字节的头（加上可选的部分）以及随后0个或者多个数据字节构成。TCP软件决定了段的大小。它可以将多次写操作中的数据累积起来，放到一个段中发送，也可以将一次写操作中的数据分割到多个段中发送。有两个因素限制了段的长度。首先，包括TCP头在内的每个段，必须适合IP的65 515个字节有效载荷；其次，每个网络都有一个**最大传输单元**（*MTU*，maximum transfer unit）。发送端和接收端的每个段必须适合MTU，才能以单个不分段的数据包发送和接收。实际上，MTU通常是1500字节（以太网的有效载荷大小），

因此它规定了段长度的上界。

然而，当IP数据包穿过一条网络路径，其上某条链路有更小的MTU时，还是有可能要对携带TCP段的该IP数据包实行分段操作。如果这种情况发生，则会降低性能并引起其他问题（Kent和Mogul, 1987）。相反，现代TCP使用RFC 1191给出的技术实现了“路径MTU发现”（path MTU discovery）功能，我们在5.5.5节对此有过描述。该技术利用ICMP错误消息来发现某条路径上任意一条链路的最小MTU。

TCP实体使用的基本协议是具有动态窗口大小的滑动窗口协议。当发送端传送一段时，它启动一个计时器。当该段到达接收方时，接收端的TCP实体返回一个携带了确认号和剩余窗口大小的段（如果有数据要发送的话，则包含数据，否则就不包含数据），并且确认号的值等于接收端期望接收的下一个序号。如果发送端的计时器在确认段到达之前超时，则发送端再次发送原来的段。

尽管这个协议听起来简单，但是有时候涉及许多非常微妙的细节，后面我们将会介绍。段到达的顺序可能是错误的，所以可能3072～4095字节的数据已经到达了，但是它不能被确认，因为2048～3071字节还没有到达。段在传输过程中可能会被延迟很长时间，因而发送端超时并重传段。重传的段包含的字节范围有可能与原来传输的段的字节范围不同，所以，这就要求仔细管理，以便跟踪哪些字节已经被正确地接收到了。然而，由于数据流中的每个字节都有它自己唯一的偏移值，所以这项管理工作是可以完成的。

TCP必须做好处理这些问题的准备，并且以一种有效的方法来解决这些问题。尽管面临各种各样的网络问题，研究人员还是做了相当多的努力来优化TCP流的性能。下面将要讨论一些被许多TCP实现采用的算法。

6.5.4 TCP段的头

图6-36显示了TCP段的结构。每个段的起始部分是一个固定格式的20字节头。固定的头部之后可能有头的选项。如果该数据段有数据部分的话，那么在选项之后是最多可达65 535-20-20=65 495个字节的数据，这里的第一个20是指IP头，第二个20指TCP头。没有任何数据的TCP段也是合法的，通常被用作确认和控制消息。

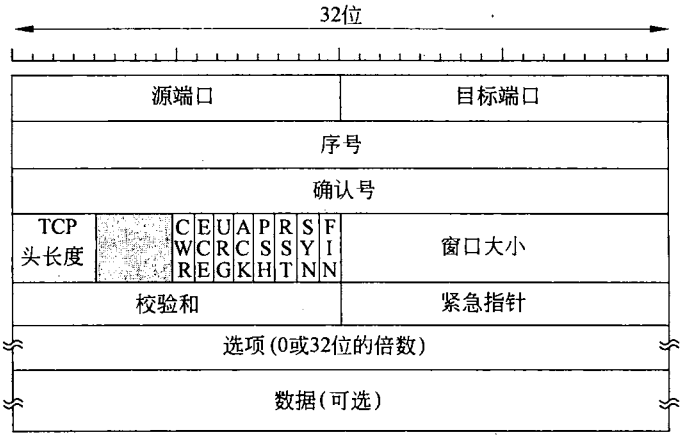


图6-36 TCP头格式.

现在让我们逐个字段地剖析 TCP 头。源端口（Source port）和目标端口（Destination port）字段标识了连接的本地端点。TCP 端口加上所在主机的 IP 地址组成了 48 位的唯一端点。源端点和目标端点一起标识了一条连接。这个连接标识符就称为 5 元组（5 tuple），因为它由 5 个信息组成：协议（TCP）、源 IP 地址和源端口号、目标 IP 地址和目标端口号。

序号（Sequence number）和确认号（Acknowledgement number）字段执行它们的常规功能。请注意，后者指定的是下一个期待的字节，而不是已经正确接收到的最后一个字节。它是累计确认（cumulative acknowledgement），因为它用一个数字概括了接收到的所有数据，它不会超过丢失的数据。这两个字段都是 32 位长，因为一个 TCP 流中的每一个数据字节都已经被编号了。

TCP 头长度（TCP header length）字段指明了 TCP 头包含多少个 32 位的字。这个信息是必需的，因为选项（Options）字段是可变长的，因而整个头也是变长的。从技术上讲，这个字段实际上指明了数据部分在段内的起始位置（以 32 位字作为单位进行计量），但因为这个数值正好是按字为单位计量的头长度，所以，两者的效果是等同的。

接下来是长度为 4 位没有被使用的字段。这些位（原先的 6 位中只有 2 位被重新声明使用了）30 年没被使用的事实恰好证明了 TCP 自始至终运行良好。差一点的协议需要这些位来修正原先设计中的错误。

接着是 8 个 1 比特的标志位。当采用 RFC 3168 说明的显式拥塞通知（ECN，Explicit Congestion Notification）时，CWR 和 ECE 就用作拥塞控制的信号。当 TCP 接收端收到了来自网络的拥塞指示后，就设置 ECE 以便给 TCP 发送端发 ECN-Echo 信号，告诉发送端放慢发送速率。TCP 发送端设置 CWR，给 TCP 接收端发 CWR 信号，这样接收端就知道发送端已经放慢速率，不必再给发送端发 ECN-Echo 信号。我们将在 6.5.10 节中讨论 TCP 拥塞控制中的 ECN 规则。

如果使用了紧急指针（Urgent pointer），则将 URG 设置为 1。紧急指针指向从当前序号开始找到紧急数据的字节偏移量。这个设施是中断消息的另一种途径。正如我们上面所述，该设施允许发送端以少得不能再少的方式给接收端发信号，无须 TCP 本身卷入中断的事由，但是这种设施很少被用到。

ACK 被设置为 1 表示确认号字段是有效的。几乎所有的数据包都会用到这个标志位。如果 ACK 为 0，则该段不包含确认信息，所以，确认号字段可以被忽略。

PSH 指出这是被推送（PUSH）的数据。特此请求接收端一旦收到数据后立即将数据递交给应用程序，而不是将它缓冲起来直到整个缓冲区满为止（这样做的目的可能是为了效率的原因）。

RST 被用于突然重置一个已经变得混乱的连接，混乱有可能是由于主机崩溃，或者其他什么原因造成的。该标志位也可以被用来拒收一个无效的段，或者拒绝一个连接请求。一般而言，如果你得到的段被设置了 RST 位，那说明你遇到问题了。

SYN 被用于建立连接过程。在连接请求中，SYN=1 和 ACK=0 表示该段没有使用捎带确认字段。但是，连接应答捎带了一个确认，因此 SYN=1 和 ACK=1。本质上，SYN 位被用来同时表示 CONNECTION REQUEST 和 CONNECTION ACCEPTED，然后进一步用 ACK 位来区分这两种可能情况。

FIN 被用来释放一个连接。它表示发送端已经没有数据需要传输了。然而，在关闭一个连接之后，关闭进程可能会在一段不确定的时间内继续接收数据。SYN 和 FIN 段都有序号，从而保证了这两种段以正确的顺序被处理。

TCP 中的流量控制是通过一个可变大小的滑动窗口来处理的。窗口大小（Window size）字段指定了从被确认的字节算起可以发送多少个字节。窗口大小字段为 0 是合法的，说明到现在为止已经接收到了多达确认号-1 个字节，但是接收端没有更多的机会来消耗数据，希望别再发数据。以后，接收端可以通过发送一个具有同样确认号但是非零窗口大小字段的段来通知发送端继续发送段。

在第 3 章的协议中，确认接收到的帧和允许发送新帧是捆绑在一起。这是每个协议采用固定窗口大小的必然结果。在 TCP 中，确认和允许发送额外数据是完全分离的两种机制。实际上，接收端可以这样说："我已经接收到了序号 k 之前的字节，但是现在我不想要任何更多的数据。"这种分离（事实上，是可变大小的窗口）带来了格外的灵活性。后面我们将详细地学习这种机制。

校验和（Checksum）提供了额外的可靠性。它校验的范围包括头、数据，以及与 UDP 一样的概念性伪头。除了伪头的协议号为 TCP（6），并且校验和必需强制执行。有关校验和的详细描述见 6.4.1。

选项（Options）字段提供了一种添加额外设施的途径，主要针对常规头覆盖不到的方面。协议定义了许多选项，有几个已经被广泛使用。选项的长度可变长，但必须是 32 位的倍数，不足部分用零填充。选项可以扩展到 40 个字节，这是说明的最长 TCP 头。某些选项用在连接建立期间，主要协商或者通知另一端的能力；其他选项用在连接生存期间，通过数据包携带。每个选项具有类型-长度-值（Type-Length-Value）编码。

用途最广的选项允许每台主机指定它愿意接受的最大段长（MSS，Maximum Segment Size）。采用大的段通常比小的段更有效率，因为这 20 字节头的开销可以分摊到更多的数据上，但是小型主机可能处理不了大的段。在连接建立过程中，每一端可以宣布它的最大段长，并且查看对方给出的最大值。如果一台主机没有使用这个选项，那么它默认可以接受 536 字节的有效载荷。所有 Internet 主机都要求能够接受 536+20=556 个字节的 TCP 段。两个方向上的最大段长可以不同。

对于具有高带宽、高延迟，或者两者兼备的线路，64 KB 的窗口对应于 16 位的字段是个问题。在一条 OC-12 的线路上（大约 600 Mbps），只需 1 毫秒就可输出一个完整的 64 KB 窗口。如果往返传播延迟是 50 毫秒（越洋光缆的典型值），则发送端将有 98%空闲，等待确认的到来。大的窗口允许发送端不停地送出数据。窗口尺度（Window scale）选项允许发送端和接收端在连接建立阶段协商窗口尺度因子。双方使用尺度因子将窗口大小字段向左移动至多 14 位，因此允许窗口最大可达 2^{30} 个字节。大多数的 TCP 实现都支持这个选项。

时间戳（Timestamp）选项携带由发送端发出的时间戳，并被接收端回应。一旦在连接建立阶段启用了它，那么每个数据包都要包含这个选项，主要用来计算来回时间样值，该样值被用在估算多久之后数据包可以被认为丢失。它还可以被用作 32 位序号的逻辑扩展。在一条快速连接上，序号空间很快回绕到零，这样将导致分不清新数据包和老数据包。防

止序号回绕（PAWS，Protection Against Wrapped Sequence numbers）方案根据时间戳丢弃入境段，从而解决这个序号回绕问题。

最后，**选择确认**（SACK，Selective ACKnowledgement）选项使得接收端可以告诉发送端已经接收到段的序号范围。这是对确认号的补充，可用在一个数据包已丢失但后续（或者重复）数据到达的特定情况下。新到达的数据无法反映出头的确认号字段，因为该字段给出的仅仅是下一个期待的有序字节。有了 SACK，发送端可以明显地感知到接收端已经接收了什么数据，并据此确定应该重传什么数据。SACK 由 RFC 2108 和 RFC 2883 定义，并逐渐为人们所应用。我们将在 6.5.10 节的拥塞控制中描述 SACK 的使用。

6.5.5 TCP 连接建立

TCP 使用了三次握手法来建立连接，这个方法我们在 6.2.2 节中讨论过。为了建立一个连接，某一端，比如说服务器，必须先依次执行 LISTEN 和 ACCEPT 原语，然后被动地等待入境连接请求；并且可以指定只接受一个特定的请求源，也可以不指定。

另一端，比如说客户，执行 CONNECT 原语，同时说明它希望连接的 IP 地址和端口、它愿意接受的最大 TCP 段长，以及一些可选的用户数据（比如口令）等参数。CONNECT 原语发送一个 SYN 标志位置为 on 和 ACK 标志位置为 off 的 TCP 段，然后等待服务器的响应。

当这个段到达接收方时，那里的 TCP 实体检查是否有一个进程已经在目标端口字段指定的端口上执行了 LISTEN。如果没有，则它发送一个设置了 RST 的应答报文，拒绝客户的连接请求。

如果某个进程正在该端口上监听，那么，TCP 实体将入境的 TCP 段交给该进程处理。该进程可以接受或者拒绝这个连接请求。如果它接受，则发送回一个确认段。正常情况下，发送的 TCP 段顺序如图 6-37（a）所示。请注意，SYN 段只消耗了 1 个字节的序号空间，所以它可被毫无异议地确认。

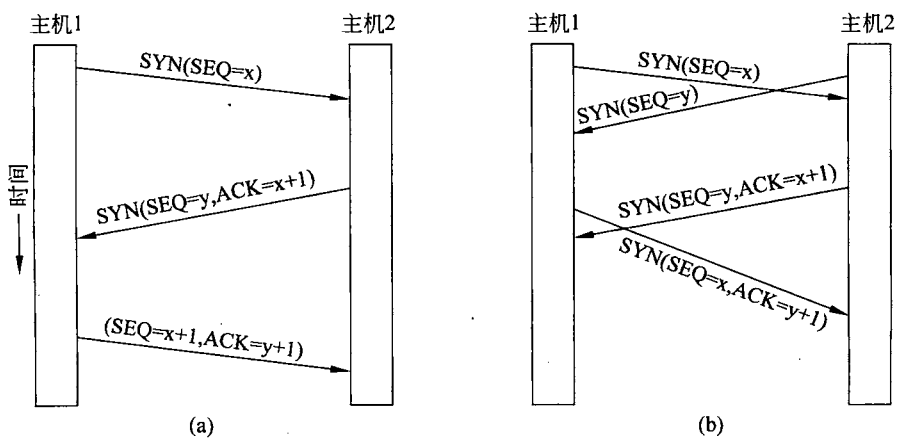


图 6-37

（a）正常情况下的 TCP 连接建立；（b）两端同时建立连接的情形

如果两台主机同时企图在一对套接字之间建立连接，则事件序列如图 6-37（b）所示。这些事件的结果是恰好只建立了一个连接，而不是两个，因为所有的连接都是由它们的端点来标识的。如果第一个请求产生了一个由(x, y)标识的连接，第二个请求也建立了这样一个连接，那么，实际上只有一个表项，即(x, y)。

回想一下，每个主机选择的初始序号应该循环比较缓慢，而且不能是一个常数，比如 0。这个规则是为了防止被延迟的重复数据包，正如我们在 6.2.2 节所讨论的。最初，采用了一个基于时钟的方案，时钟每 4 微秒滴答一次，由此来解决初始序号问题。

然而，三次握手的实现方式有个漏洞，当监听进程接受一个连接请求，并立即以 SYN 段作为响应时，它必须记住该 SYN 段的序号。这意味着一个恶意的发送端很容易地占据一个主机资源。具体做法是这样的：恶意用户绵绵不绝地发送 SYN 段请求服务器的连接，但又故意不完成连接建立的后续过程，由此可消耗掉一台主机的资源。这种攻击称为 **SYN 泛洪**（SYN flood），它在 20 世纪 90 年代致使许多 Web 服务器瘫痪。

抵御这种攻击的一种方法是使用 SYN“小甜饼”（SYN Cookie）。主机不去记忆序号，而是选择一个加密生成的序号，将它放在出境段中，并且忘记它。如果三次握手完成后，该序号（加 1）将返回主机。主机运行相同的加密函数，只要该函数的输入是已知的（例如，其他主机的 IP 地址和端口，和一个本地密钥），它就能重新生成正确的序号。这个过程允许主机检查确认号是否正确，而不必记住单独的序号。这里存在一些注意事项，比如无法处理 TCP 选项，所以只有当主机容易受到 SYN 泛洪时，才可以用 SYN Cookie。然而，这是连接建立过程的一种有趣扭曲。欲了解更多信息，请参阅 RFC 4987 和（Lemon, 2002）。

6.5.6 TCP 连接释放

虽然 TCP 连接是全双工的，但是，为了理解 TCP 连接是如何释放的，最好将 TCP 连接看成一对单工连接。每个单工连接的释放彼此独立。为了释放一个连接，任何一方都可以发送一个设置了 FIN 标志位的 TCP 段，这表示它已经没有数据要发送了。当 FIN 段被另一方确认后，这个方向上的连接就被关闭，不再发送任何数据。然而，另一个方向上或许还在继续着无限的数据流。当两个方向都关闭后，连接才算被彻底释放。通常情况下，释放一个连接需要 4 个 TCP 段：每个方向上一个 FIN 和一个 ACK。然而，第一个 ACK 和第二个 FIN 有可能被组合在同一个段中，从而将所需段总数降低到 3 个。

正如在电话通话过程中，双方说完再见之后同时挂断电话一样，一个 TCP 连接的两端也可能会同时发送 FIN 段。这两个段按常规的方法被单独确认，然后关闭连接。实际上，两台主机可以先后释放连接，或者同时释放连接，这两者之间并没有本质的区别。

为了避免两军对垒问题（在 6.2.3 节中讨论过的），需要使用计时器。如果在两倍于最大数据包生存期内，针对 FIN 的响应没有出现，那么 FIN 的发送端直接释放连接。另一方最终会注意到似乎没人在监听连接，因而也会超时。虽然这种方案不是完美无缺，但由于理论上根本不存在完美的解决方案，所以，TCP 也只好这样做了。实际上，这种做法很少产生问题。

6.5.7 TCP 连接管理模型

建立连接和释放连接所需要的步骤可以用一个有限状态机来表示，该状态机的 11 种状态如图 6-38 所列。在每一种状态中，都存在特定的合法事件。当一个合法事件发生时，可能需要采取某个动作。当发生其他事件时，则报告一个错误。

状态	描述
CLOSED	没有活跃的连接或者挂起
LISTEN	服务器等待入境呼叫
SYN RCVD	到达一个连接请求；等待ACK
SYN SENT	应用已经启动了打开一个连接
ESTABLISHED	正常的数据传送状态
FIN WAIT1	应用没有数据要发了
FIN WAIT2	另一端同意释放连接
TIME WAIT	等待所有数据包寿终正寝
CLOSING	两端同时试图关闭连接
CLOSE WAIT	另一端已经发起关闭连接
LAST ACK	等待所有数据包寿终正寝

图 6-38 TCP 连接管理有限状态机使用的状态

每个连接都从 CLOSED 状态开始。当它执行了一个被动打开操作（LISTEN），或者一个主动打开操作（CONNECT）后，它就离开 CLOSED 状态。如果另一端执行了相反的操作，则连接就建立起来，当前状态变成 ESTABLISHED。连接的释放过程可以由任何一方发起。当释放完成时，状态又回到 CLOSED。

有限状态机本身如图 6-39 所示。在图中用粗线表示客户主动连接到一个被动服务器上的一般情形，其中客户部分用的是实线，服务器部分用的是虚线。细线表示了不常用的事件序列。图 6-39 中的每条线都标记成一对“事件/动作”（event/action）。这里的事件既可以是用户发起的系统调用（CONNECT、LISTEN、SEND 或者 CLOSE），也可以是到达了一个段（SYN、FIN、ACK 或者 RST），或者是发生了两倍于最大数据包生存期的超时事件。动作可以是发送一个控制段（SYN、FIN 或者 RST），或者什么也不做（在图中标记为-）。括号中显示的是说明。

为了更好地理解这个图，你可以首先沿着客户的路径（粗实线），然后沿着服务器的路径（粗虚线）来查看。当客户机器上的一个应用程序发出 CONNECT 请求，本地的 TCP 实体创建一条连接记录，并将它标记为 SYN SENT 状态，然后发送一个 SYN 段。请注意，在一台机器上可能同时有许多个连接处于打开（或者正在被打开）状态，它们可能代表了多个应用程序，所以，状态是针对每个连接的，并且每个连接的状态被记录在相应的连接记录中。当 SYN+ACK 到达的时候，TCP 发出三次握手过程的最后一个 ACK 段，然后切换到 ESTABLISHED 状态。现在可以发送和接收数据了。

当一个应用结束时，它执行 CLOSE 原语，从而使本地的 TCP 实体发送一个 FIN 段，并等待对应的 ACK（虚线框标记了“主动关闭”）。当 ACK 到达时，状态迁移到 FIN WAIT 2，而且连接的一个方向被关闭。当另一方也关闭时，会到达一个 FIN 段，然后它被确认。现在，双方都已经关闭了连接，但是，TCP 要等待一段长度为最大数据包生存期两倍的时

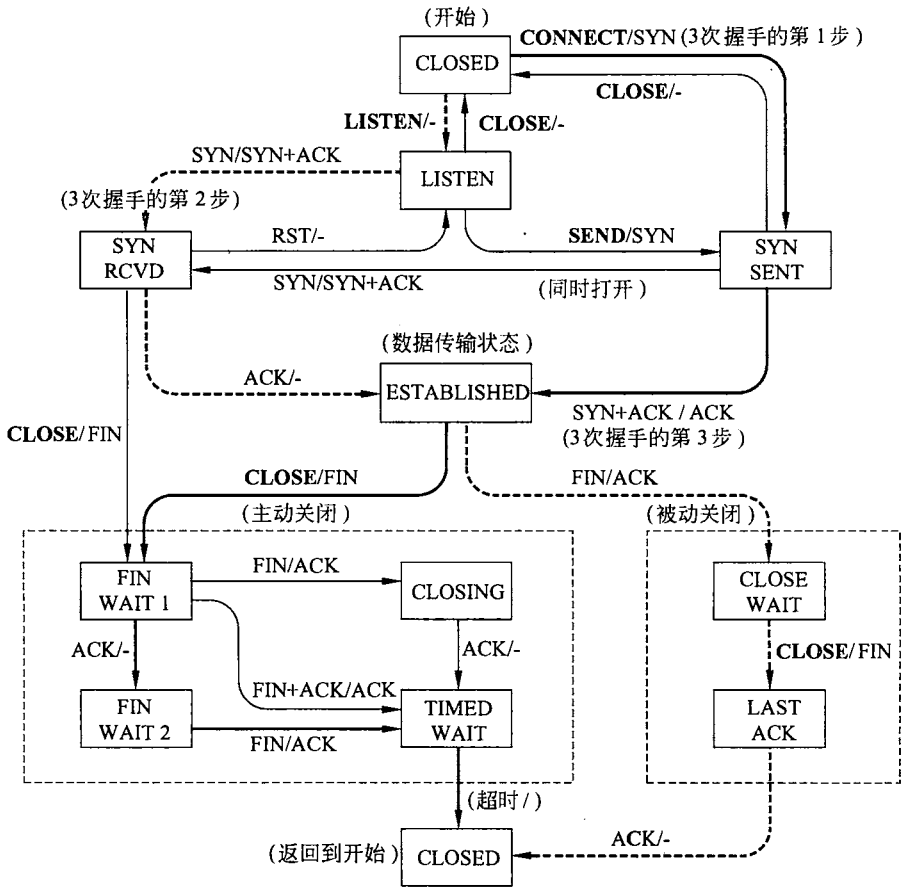


图 6-39 TCP 连接管理有限自动机

粗实线是客户端的正常路径，粗虚线是服务器端的正常路径。细线是不常发生的事件。每次状态迁移由引发的事件以及相应动作标记，事件和动作用斜杠分割

间，才能确保该连接上的所有数据包都已寿终正寝，以防万一发生确认被丢失的情形。当计时器超时后，TCP 删除该连接记录。

现在让我们从服务器的角度来看连接管理的情况。服务器执行 LISTEN，并等待入境连接请求。当收到一个 SYN 时，服务器就确认该段并且进入到 SYN RCVD 状态。当服务器本身的 SYN 被确认后，就标志着三次握手过程的结束，服务器进入到 ESTABLISHED 状态。从现在开始双方可以传输数据了。

当客户完成了自己的数据传输，它就执行 CLOSE，从而导致 TCP 实体发送一个 FIN 到服务器（虚线框标记了“被动关闭”）。然后，服务器接到信号；当它也执行了 CLOSE 时，TCP 实体给客户发送一个 FIN 段。当该段的来自客户的确认返回后，服务器释放该连接，并且删除相应的连接记录。

6.5.8 TCP 滑动窗口

正如前面所提到，TCP 的窗口管理将正确接收段的确认和接收端的接收缓冲区分配分离开来。例如，假设接收端有一个 4096 字节的缓冲区，如图 6-40 所示。如果发送端传送

了一个2048字节的数据段，并且该数据段已被正确地接收，那么，接收端将确认该数据段。然而，由于接收端现在只剩下2048字节的缓冲区空间（在应用程序从缓冲区中取走数据之前），所以它将宣告下一个期望字节开始窗口为2048。

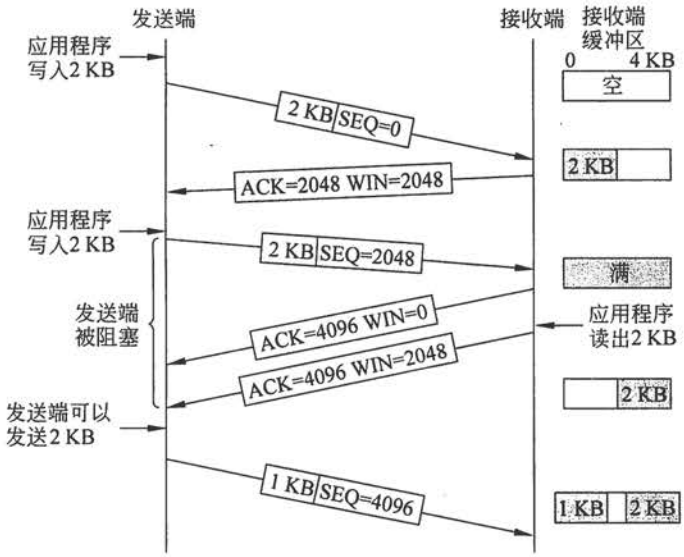


图6-40 TCP的窗口管理

现在发送端又传输了另一个2048字节长的段，该段被确认了，但是接收端宣告的窗口大小变成了0。因此，发送端不得不停止，等待接收主机上的应用进程从缓冲区中取走一些数据。到那时候，TCP实体可以向发送端宣告一个更大的窗口，发送端才能发送数据。

当窗口变为0时，发送端不能如通常那样发送段了，但这里有两种意外情形。第一，紧急数据仍可以发送，比如，允许用户杀掉远程机器上运行的某一个进程。第二，发送端可以发送一个1字节的段，以便强制接收端重新宣告下一个期望的字节和窗口大小。这种数据包称为**窗口探测**（window probe）。TCP标准明确地提供了这个选项，来防止窗口更新数据包丢失后发生死锁。

发送端并不一定一接到应用程序传递来的数据就马上将数据传送出去；同样，接收端也不一定必须尽可能快地发送确认段。例如，在图6-40中，当第一块2 KB数据到来时，TCP知道它有4 KB的窗口可以使用，所以它完全可以将这些数据缓冲起来，直到另一个2 KB的数据到来后，发送一个包含4 KB有效载荷的段。TCP实体可以充分利用这种自由度来提高传输性能。

考虑一个远程终端连接，比如使用SSH或者Telnet，该连接对用户的每次击键动作都做出响应。在最差的情况下，当一个字符到达发送端的TCP实体，TCP创建一个21字节的TCP段，并将它交给IP组成一个41字节的IP数据报，然后发送出去；在接收端，TCP立即发送一个40字节的确认（20字节的TCP头加上20字节的IP头）。以后，当远程终端读取了这个字节之后，TCP发送一个窗口更新段，它将窗口向前移动1个字节。这个数据包也是40字节长。最后，当远程终端处理了该字符以后，它发送一个41字节的数据包作为该字符的回显。总共累计起来，对于每次敲入的字符，需要使用162字节的带宽，并发送4个数据段。对于带宽紧缺的场合，这种处理方法显然并不合适。

针对这样的情况，许多 TCP 实现采用了一种称为**延迟确认**（delayed acknowledgement）的优化方法。基本想法是将确认和窗口更新延迟 50 毫秒，希望能够获得一些数据免费搭载过去。假设远程终端在 500 毫秒内回显，则现在远程用户只需要送回一个 41 字节的数据包即可，从而将数据包数和所用带宽减少了一半。

尽管通过延迟确认减少了接收端给予网络的负载，但是发送端发送多个小数据包的工作方式仍然非常低效（比如，41 字节的数据包只包含 1 个字节的数据）。避免这种用法的一种办法是采用 Nagle 算法（Nagle，1984）。Nagle 的建议非常简单：当数据每次以很少量方式进入到发送端时，发送端只是发送第一次到达的数据字节，然后将其余后面到达的字节缓冲起来，直到发送出去的那个数据包被确认；然后将所有缓冲的字节放在一个 TCP 段中发送出去，并且继续开始缓冲字节，直到下一个段被确认。这就是说，任何时候只有第一个发送的数据包是小数据包。如果在一个来回时间内应用程序发送了许多数据，那么 Nagle 算法可以将这些数据放置在一个段中发送，由此大大地减少所需的带宽。另外，如果应用传递来的数据足够多，多到可以填满一个最大数据段，则该算法也允许发送一个新的段。

什么时候不用它更好？特别是，在 Internet 上玩互动游戏时，玩家通常需要一个快速的短数据包流。如果把更新收集起来以突发的方式发送将使得游戏的响应不稳定，这显然会引起用户不满。一个更微妙的问题是 Nagle 算法有时可能会延迟确认的相互作用，从而造成暂时的死锁：接收端等待（上层）数据到来以便可以捎带确认，而发送端等待确认的到来以便能够发送更多的数据。这种相互作用可能导致延迟网页的下载。由于存在这些问题，可以禁用 Nagle 算法（这就是所谓的 TCP_NODELAY 选项）。（Mogul 和 Minshall，2001）讨论了这个问题以及其他的解决方案。

降低 TCP 性能的另一个问题是**低能窗口综合症**（silly window syndrome）（Clark, 1982）。当数据以大块形式被传递给发送端 TCP 实体，但是接收端的交互式应用每次仅读取一个字节数据的时候，这个问题就会发生。为了看清此问题，请参考图 6-41。初始时，接收端的 TCP 缓冲区为满，发送端知道这一点（即它有一个大小为 0 的窗口）。然后，交互式应用从 TCP 流中读取一个字符，这个动作使得接收端的 TCP 欣喜若狂，它立刻发送一个窗口更新段给发送端，告诉它现在可以发送 1 个字节过来。发送端很感激，立即发送 1 个字节。现在缓冲区又满了，所以，接收端对这 1 字节的数据段进行确认，同时设置窗口大小为 0。这种行为可能会永久地持续下去。

Clark 的解决方案是禁止接收端发送只有 1 个字节的窗口更新段。相反，它强制接收端必须等待一段时间，直到有了一定数量的可用空间之后再通告给对方。特别是，只有当接收端能够处理它在建立连接时宣告的最大数据段，或者它的缓冲区一半为空时（相当于两者之中取较小的值），它才发送窗口更新段。而且，发送端不发送太小的段也会有所帮助。相反，它应该等待一段时间，直到可以发送一个满的段，或者至少包含接收端缓冲区一半大小的段。

Nagle 的算法和 Clark 针对低能窗口综合症的解决方案相互补充。Nagle 试图解决由于发送端应用每次向 TCP 传递一个字节而引起的问题；Clark 则试图解决由于接收端应用每次从 TCP 流中读取一个字节而引起的问题。这两种方案都有效的，而且可以一起工作。发送端的目标是不发送太小的数据段，接收端也不要请求太小的段。

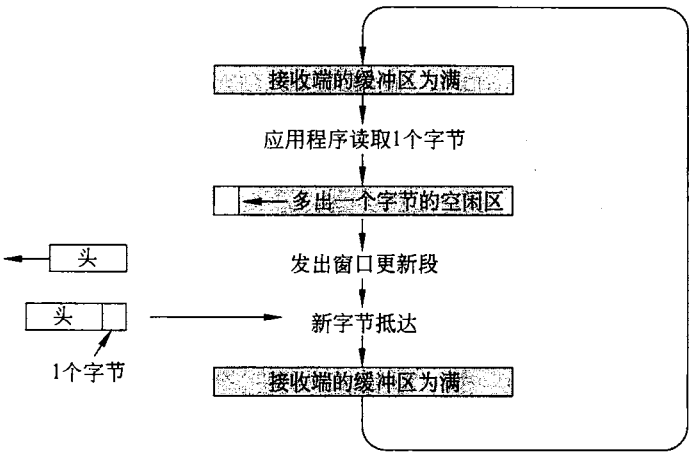


图 6-41 低能窗口综合症

接收端的 TCP 实体除了向发送端宣告较大的窗口以外，还可以进一步提高性能。如同发送端的 TCP 一样，接收端的 TCP 也可以缓冲数据，所以它可以阻塞上层应用的 READ 请求，直至它积累了大块的数据。这样做可以减少调用 TCP 的次数，从而减少额外的开销。当然，这样做也增加了响应的时间，但是，对于像文件传输这样的非交互式应用来说，效率可能比单个请求的响应时间更加重要。

接收端必须处理的另一个问题是乱序到达的数据段。接收端将缓冲这些数据，直至可以按照顺序递交给应用程序为止。实际上，没有什么比丢弃乱序到达的段更糟糕的事情，因为若丢弃这些段，那么它们将被发送端重传，这是一种浪费。

只有当确认字节之前的所有数据都到达之后才能发送确认，这种方式称为**累计确认**（cumulative acknowledgement）。如果接收端已经获得段 0、1、2、4、5、6、7，它可以确认直到段 2（包括段 2）之前的数据。当发送端超时，然后重发段 3。因为接收端已经缓冲了段 4～段 7，一旦它收到段 3 就可立即确认直到段 7 的全部字节。

6.5.9 TCP 计时器管理

TCP 使用多个计时器（至少从概念上讲是计时器）来完成它的工作。其中最重要的是**重传计时器**（RTO，Retransmission TimeOut）。当 TCP 实体发出一个段时，它同时启动一个重传计时器。如果在该计时器超时前该段被确认，则计时器被停止。另一方面，如果在确认到来之前计时器超时，则段被重传（并且该计时器被重新启动）。于是问题就来了：超时间隔应该设为多长？

这个问题在传输层上比诸如 802.11 那样的数据链路协议要更加困难。在数据链路层，期望的延迟是可以测量的毫秒数，并且是高度可预测的（即方差很小），所以，计时器可以被设置成恰好比期望的确认到达时间稍长即可，如图 6-42（a）所示。由于在数据链路层中确认极少被延迟（因为不存在拥塞），所以，如果在预期的时间内确认没有到来，则往往意味着帧或者确认已经被丢失了。

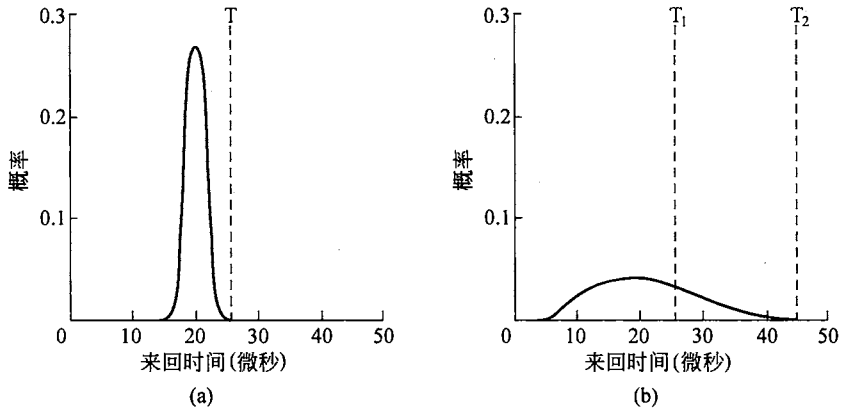


图 6-42

(a) 数据链路层的确认到达时间的概率密度；(b) TCP 确认到达时间的概率密度

TCP 面临着截然不同的环境。TCP 确认回到发送端所需时间的概率密度函数看起来更像 6-42（b），而不是图 6-42（a），它更大也更可变。要想确定到达接收方的往返时间非常棘手，即使知道了这段时间，要确定超时间隔也非常困难。如果超时间隔设置得太短，比如说图 6-42(b)中的 T1，则会发生大量不必要的重传，这么多无用数据包反而堵塞 Internet；如果超时间隔被设置得太长（比如 T2），则一旦数据包丢失之后，由于太长的重传延迟，所以性能会受到影响。而且，确认到达时间分布的均值和方差也会随着拥塞的发生或者解决而在几秒钟时间内迅速地改变。

解决方案是使用一个动态算法，它根据网络性能的连续测量情况，不断地调整超时间隔。TCP 通常采用的算法是（Jacobson，1988）算法，其工作原理如下所述。对于每一个连接，TCP 维护一个变量 SRTT（Smoothed Round-Trip Time，平滑的往返时间），它代表到达接收方往返时间的当前最佳估计值。当一个段被发送出去时，TCP 启动一个计时器，该计时器有两个作用，一是看该段被确认需要多长时间； 二是若确认时间太长，则触发重传动作。如果在计时器超时前确认返回，则 TCP 测量这次确认所花的时间，比如说 R。然后它根据下面的公式更新 SRTT：

$$SRTT = \alpha\, SRTT + (1 - \alpha)\, R$$

这里 α 是一个平滑因子，它决定了老的 RTT 值所占的权重。典型情况下 α=7/8。这类公式是**指数加权移动平均**（EWMA，Exponentially Weighted Moving Average），或者丢弃样值中噪声的低通滤波器。

即使有了一个好的 SRTT 值，要选择一个合适的重传超时间隔仍然不是一件容易的事情。在最初的实现中，TCP 使用 2xRTT，但经验表明常数值太不灵活，当发生变化时它不能够很好地做出反应。尤其是，随机流量的排队模型（即泊松分布）预测的结果是当负载接近容量时，延迟不仅变大，而且变化也大。这可能导致重传计时器超时而重新传输一个数据包的副本，尽管原先的数据包仍然在网络中传输着。这在高负载的情况下也有可能发生，最糟糕的是将额外的数据包发到网络中。

为了解决这个问题，Jacobson 提议让超时值对往返时间的变化以及平滑的往返时间要变得敏感。这种改变要求跟踪另一个平滑变量，RTTVAR（往返时间变化，Round-Trip Time

VARiation），即更新为下列公式：

$$RTTVAR = \beta\, RTTVAR + (1 - \beta) \mid SRTT - R \mid$$

这就像以前的EWMA，典型地，$\beta = 3/4$。因此，重传超时值，RTO为

$$RTO = SRTT + 4 \times RTTVAR$$

这里选择因子4多少有点随意，但是乘以4的操作用一个移位操作就可以实现，而且，所有数据包中大于标准方差4倍以上的不足总数的1%。注意RTTVAR并不是确切地等于标准方差（真正的它是平均方差），但它实际上足够接近。Jacobson文章充满着一些聪明的技巧，使用整数加法、减法和移位操作来计算超时值。这种经济运算对于现代主机并不是很需要，但它已成为文化的一部分，使得TCP可运行在所有种类的设备上，从超级计算机到微型设备。到目前为止，还没有人把它搬到RFID上，但这天会来吗？天知道！

计算超时值的详细过程，包括变量的初始值设置，可参考RFC 2988。重传计时器的最小值为1秒，无论估算值是多少。这是为了防止根据测量获得的欺骗性重传值而选择的保守值。

在采集往返时间的样值R过程中有可能引发的一个问题是，当一个段超时并重新发送以后该怎么办？确认到达时，无法判断该确认是针对第一次传输，还是针对后来的重传。若猜测错误，则会严重影响重传超时值。Phil Karn发现这个问题很艰难。Karn是一名业余无线电爱好者，他对在一种极不可靠的无线介质上传输TCP/IP数据包有浓厚的兴趣。他提出了一个很简单的建议： 不更新任何重传段的估算值。此外，每次连续重传的超时间隔值加倍，直到段能一次通过为止。这个修正算法称为**Karn算法**（Karn和Partridge，1987）。大多数TCP实现都采用了此算法。

重传计时器并不是TCP使用的唯一计时器。第二个计时器是**持续计时器**（persistence timer）。它的设计意图是为了避免出现以下所述的死锁情况。接收端发送一个窗口大小为0的确认，让发送端等一等。稍后，接收端更新了窗口，但是，携带更新消息的数据包丢失了。现在，发送端和接收端都在等待对方的进一步动作。当持续计时器超时后，发送端给接收端发送一个探询消息。接收端对探询消息的响应是将窗口大小告诉发送端。如果它仍然为0，则重置持续计时器，并开始下一轮循环。如果它非0，则现在可以发送数据了。

有些TCP实现使用了第三个计时器，即**保活计时器**（keepalive timer）。当一个连接空闲了较长一段时间以后，保活计时器可能超时，从而促使某一端查看另一端是否仍然还在。如果另一端没有响应，则终止连接。这个特性是有争议的，因为它增加了额外的开销，而且有可能由于瞬间的网络分区而终止掉一个本来很正常的连接。

每个TCP连接使用的最后一个计时器被应用于连接终止阶段，即在关闭过程中，当连接处于TIMED WAIT状态时所使用的计时器。它的超时值为两倍于最大数据包生存期，主要用来确保当连接被关闭后，该连接上创建的所有数据包都已完全消失。

6.5.10 TCP拥塞控制

我们把TCP的关键功能之一保留到最后，即拥塞控制。当提供给任何网络的负载超过它的处理能力时，拥塞便会产生。Internet也不例外。当路由器上的队列增长到很大时，网络层检测到拥塞，并试图通过丢弃数据包来管理拥塞。传输层接收到从网络层反馈来的拥

塞信息，并减慢它发送到网络的流量速率。在 Internet 上，TCP 在控制拥塞以及可靠传输中发挥着主要的作用。这就是为什么 TCP 是如此特殊的一个协议。

我们在 6.3 节中介绍了拥塞控制的一般情况。关键在于如果一个传输协议使用 AIMD（加法递增乘法递减）控制法则来响应从网络传来的二进制拥塞信号，那么该传输协议应该收敛到一个公平且有效的带宽分配。TCP 的拥塞控制就是在这个方法的基础上实现的，它使用了一个窗口，并且把丢包当作二进制信号。要做到这一点，TCP 维持一个**拥塞窗口**（congestion window），窗口大小是任何时候发送端可以往网络发送的字节数。相应的速率则是窗口大小除以连接的往返时间。TCP 根据 AIMD 规则来调整该窗口的大小。

回想一下，除了维护一个拥塞窗口外，还有一个流量控制窗口，该窗口指出了接收端可以缓冲的字节数。要并发跟踪这两个窗口，可能发送的字节数是两个窗口中较小的那个。因此，有效窗口是发送端认为的应该大小和接收端认为的应该大小两者中的较小者。真所谓一个巴掌拍不响。如果拥塞窗口或流量控制窗口暂时已满，则 TCP 将停止发送数据。如果接收端说“发送 64 KB 数据”，但发送端知道超过 32 KB 的突发将阻塞网络，它就只发送 32KB；另一方面，如果接收端说“发送 64 KB 数据”，发送端知道高达 128 KB 的突发通过网络都毫不费力，它会发送要求的全部 64 KB。流量控制窗口在前面已经所述过，下面我们只描述拥塞窗口。

现代拥塞控制被添加到 TCP 的实现中很大程度上要归功于（Van Jacobson，1988）的努力。这是一个引人入胜的故事。从 1986 年开始，早期 Internet 的日益普及导致了第一次出现称为**拥塞崩溃**（congestion collapse）的问题，由于网络拥塞，在很长一段时间内实际吞吐量急剧下降（即下降幅度超过了 100 倍）。Jacobson（和许多其他人）着手了解发生了什么事并试图采取补救措施。

Jacobson 实施的高层次修补程序近似于一个 AIMD 拥塞窗口。TCP 拥塞控制中最有趣也是最复杂的部分在于如何在不改变任何消息格式的前提下，把这个修补措施部署到现有的 TCP 实现中。开始，他观察到丢包可以作为一个合适的拥塞信号。这个信号来得有点晚（因为网络早已拥挤不堪），但它相当值得信赖。毕竟，很难建造一个超负荷时也不丢包的路由器，这是一个不太可能改变的事实。即使可用兆兆位（terabyte）的存储器来缓冲大量的数据包，我们或许会有兆位/秒的网络，它们依然会填满这些存储器。

然而，用丢包作为拥塞信号依赖于传输错误相对比较少见的场合。对于诸如 802.11 的无线链路，传输错误比较常见，这也就是为什么它们在链路层有自己的重传机制。由于无线重传，网络层因传输错误导致的丢包通常被屏蔽在无线网络中。这也是在其他链路上很罕见的情况，因为电线和光纤通常具有较低的误码率。

Internet 上的所有 TCP 算法都假设丢包是由拥塞和监控器超时所引起的，而且寻找引起麻烦的蛛丝马迹犹如矿工观看他们饲养的金丝雀一样困难。需要一个良好的重传计时器来准确检测到丢包信号，而且必须以一种及时的方式。我们已经讨论了 TCP 重传计时器如何包括了往返时间的均值和方差的估算。为了修复该计时器，Jacobson 工作中很重要的一步是在估算中要包括方差因子。有了良好的重传超时，TCP 发送端可以跟踪发出去的字节，这些就是网络的负载。它只是简单地着眼于传输的序号和确认的序号之间的差异。

现在看来我们的任务似乎很容易。我们所需要做的是使用序号和确认号来跟踪拥塞窗口，并使用一个 AIMD 规则来调整拥塞窗口的大小。正如你可能已经预料到的，这项工作

做起来比听上去的更复杂。首先要考虑的是数据包发送到网络的方式，即使在很短的时间内也必须匹配网络路径，否则就有可能造成拥塞。例如，考虑一个主机有一个 64 KB 的拥塞窗口，它通过一条 1 Gbps 的链路连到交换式以太网。如果主机一次发送整个窗口，这样的突发流量或许要穿过前方路径上的某条缓慢的 1 Mbps ADSL 线路。这个突发在 1 Gbps 线路上只需要半毫秒，但在 1 Mbps 的线路上则需要半秒，因而阻塞了该条线路，完全扰乱了诸如 IP 语音这样的传输协议。这种行为告诉我们，设计一个能避免拥塞的协议而不是一个能控制拥塞的协议或许是更好的主意。

然而，事实证明，我们可以使用小的突发数据包来突出我们的优势。图 6-43 显示了一个突发数据包的过程。在这个例子中，快速网络上的发送端（1 Gbps 链路）突发 4 个包给一个慢速网络上的接收端（1 Mbps 链路），这条链路是路径上的瓶颈或者最慢的部分，我们来看会发生什么情况。最初的 4 个包以发送端的发送速率尽快通过链路；因为通过一条慢速链路发送数据包所需要的时间比路由器从快速链路上接收下一个数据包所需要的时间更长，所以在路由器中，它们有的正在被发送，有的在排队等候输出。但是队列并不长，因为只有少量的数据包被一次发送过来。特别注意在慢速链接上数据包的长度有所增长。同样的数据包，比如说 1 KB，在慢速链路上的发送时间比在快速链路上的发送时间更长，因而相对链路而言，数据包的长度更长。

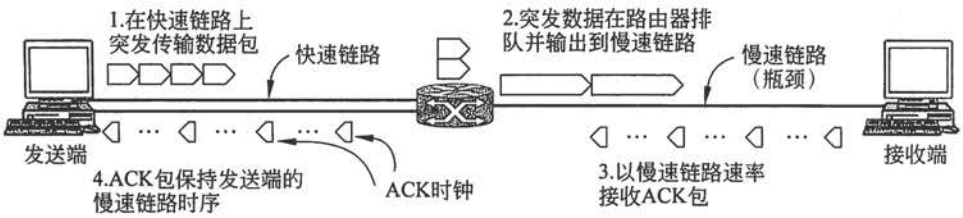


图 6-43 发送端的突发数据包以及返回的确认时钟

最终，数据包抵达接收端，在那里它们得到确认。确认的时间反映了数据包穿越慢速链路后到达接收端的时间。相比在快速链路上的原始数据包，这些确认更分散。随着这些确认穿过网络并传回到发送端，它们保留了这个时序。

关键的观察是这样的：确认返回到发送端的速率恰好是数据包通过路径上最慢链路时的速率。这正是发送端应该使用的精确发送速率。如果发送端以这个速率往网络注入新的数据包，这些数据包就能以慢速链路允许的速率一样快的速度被转发出去，但它们不会再排队和堵塞沿途上的任何一个路由器。这个时序就是**确认时钟**（ack clock）。这是 TCP 的基本组成部分。通过使用一个确认时钟，TCP 平滑输出流量和避免不必要的路由器队列。

第二个考虑是如果网络拥塞窗口从一个很小的规模开始，那么在快速网络上应用 AIMD 规则将需要很长的时间才能达到一个良好的操作点。考虑一个适度的网络路径，它可以支持 10 Mbps 的传输速率，并且往返时间 RTT 为 100 毫秒。适当的拥塞窗口是带宽延迟积，即 1 MB 或 100 个具有 1250 字节的数据包。如果拥塞窗口从 1 开始，每个 RTT 递增 1 个数据包，那么需要 100 个 RTT，或 10 秒，连接才能以正确的速率运行。在达到正确的传输速度之前要等待一段漫长的时间。如果从一个更大的初始窗口开始，比如 50 个数据包，那么我们就可以减少启动时间。但是，这个窗口对于慢速或者短程链路又太大了点，如果一次全部用完窗口，它会造成拥堵，正如我们刚才所讨论的那样。

相反，针对这两种情况 Jacobson 选择的解决方案是一个线性增长和乘法增长相混合的方法。当建立连接时，发送端用一个很小的值初始化拥塞窗口，最多不超过 4 个段；在 RFC 3390 中描述了该方案的细节，早期的实现采用的初始值是 1，后根据经验改成了用 4 作为初始值。然后发送端发送该初始窗口大小的数据。数据包必须经过一个往返时间才被确认。对于每个重传计时器超时前得到承认的段，发送端的拥塞窗口增加一个段的字节量。此外，随着该段获得确认，现在网络中又少了一段。结果是每一个被确认的段允许发送两个段，每经过一个往返时间拥塞窗口增加一倍。

这种算法称为**慢速启动**（slow start），但它根本不慢——它呈指数增长——相比以前的算法，即一次发送整个流量控制窗口大小的数据，它应该算启动得不快。慢速启动算法如图 6-44 所示。在第一次往返时间，发送端把一个数据包注入网络（并且接收端接收到一个数据包）。在接下来的一个往返时间，发送端发出两个数据包，然后在第三个往返时间发送四个。

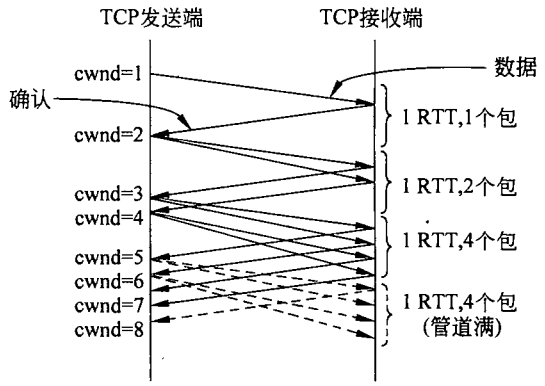


图 6-44 初始拥塞窗口为一个段的慢速启动过程

慢速启动在一定范围内的链接速度和往返时间上都能行之有效，并且使用确认时钟将发送端的传输速率与网络路径相匹配。考查图 6-44 中确认段从其发送端发送到接收端的方式。当发送端得到一个确认，它就把拥塞窗口大小增加 1，并立即将两个数据包发送到网络中（其中一个包是新增加的一个；另一个包是替代那个已被确认并离开网络的数据包。任何时候未确认的数据包的数量就是拥塞窗口的大小）。不过，这两个数据包到达接收端的紧随程度不一定非要和它们被发送出来的间隔一样。例如，假设发送端在 100 Mbps 的以太网上。每个 1250 字节长的数据包需要 100 微秒的发送时间。因此两个数据包之间的延迟只有 100 微秒。但是如果这些数据包在当前路径的任何地方穿过一条 1 Mbps 的 ADSL 链路时，形势就发生了变化。现在发送同样的数据报需要 10 毫秒。这意味着两个数据包之间的最小时间间隔增长了 100 倍。除非数据包必须在此后的某条链路上排队等候，否则它们之间的间距仍然很大。

在图 6-44 中，这种效果体现在强制数据包以它们之间的最小间距到达接收端。接收端在发送确认时要维持这种间隔，从而发送端也会以同样的间距接收确认。如果网络路径缓慢，确认将抵达得较晚（超过一个 RTT 的延迟）。如果网络路径很快，确认会迅速到达（同样，在一个 RTT 后）。发送端所要做的只是按照确认时钟的时间来注入新的数据包，这就是慢速启动。

由于慢速启动导致拥塞窗口按指数增长，最终（很快而不是很晚）它将太多的数据包以太快的速度发到网络。当发生这种情况时，网络中将很快建立起队列。当队列满时，一个或多个包会被丢弃。此后，当确认未能如期返回发送端时，TCP 发送端将超时。有证据表明，如图 6-44 的慢速启动增长太快。经过 3 个 RTT，网络中有四个数据包。这四个数据包经过完整的 RTT 后到达接收端。也就是说，四个包的拥塞窗口大小就是该连接的正确大小。然而，随着这些数据包被确认，慢速启动持续增长拥塞窗口，在另一个 RTT 达到 8 个包。这些数据包中只有四个在一个 RTT 内到达接收端，无论发送了多少个数据包。也就是说，网络管道已经为满。发送端把额外的数据包放置到网络中将在路由器上产生队列，因为它们被传递到接收器的速度不够快。拥塞和丢包即将发生。

为了保持对慢速启动的控制，发送端为每个连接维持一个称为**慢启动阈值**（slow start threshold）的阈值。最初，这个值被设置得任意高，可以达到流量控制窗口的大小，因此它不会限制连接速度。TCP 以慢速启动方式不断增加拥塞窗口，直到发生超时，或者拥塞窗口超过该阈值（或接收端的窗口为满）。

每当检测到丢包，比如超时了，慢启动阈值就被设置为当前拥塞窗口的一半，整个过程再重新启动。基本想法是当前的窗口太大，因为是它在过去造成了阻塞，所以现在才检测到超时。一较早时间成功使用的一半窗口也许是拥塞窗口的更好估计值，它更接近路径的容量而不会造成丢失。我们在图 6-44 的例子中，当拥塞窗口增长到 8 个数据包时，可能造成丢失，而在前一个 RTT 中具有 4 个数据包的拥塞窗口就是正确值。然后，拥塞窗口被复位到其初始值，恢复慢速启动。

一旦慢速启动超过了阈值，TCP 就从慢速启动切换到线性增加（即加法递增）。在这种模式下，每个往返时间拥塞窗口只增加一段。像慢速启动一样，这通常也是为每一个被确认的段而不是为每一次 RTT 实施窗口的增加。回忆一下，拥塞窗口 cwnd 和最大段长 MSS，一个常见的近似做法是这样的：针对 cwnd/MSS 中可能被确认的每个数据包，将 cwnd 增加（MSS×MSS）/cwnd。这种增长速度并不需要很快。整个想法是在一个 TCP 连接所花费的大量时间中，它的拥塞窗口接近最佳值——不至于小到降低吞吐量，但也没大到发生拥塞情况。

线性递增如图 6-45 所示，这是相同情况下的慢速启动。在每一个 RTT 结尾，发送端的拥塞窗口增长到足够它可以向网络注入额外一个数据包。相比慢速启动，线性增长速率要慢得多。这对于小的拥塞窗口差别并不大，就像这里的例子一样，但当随着时间拥塞窗口增长到一定程度时，比如说 100 段，差异就大了。

我们还可以做些其他事情来提高性能。到目前为止这个方案中的缺陷是等待超时。超时时间相对较长，因为它们必须是保守的。当一个包被丢失后，接收端不能越过它确认，因此确认号将保持不变，发送端因为拥塞窗口为满将无法发送任何新包到网络。这种情况可以持续一段比较长的时期，直到计时器被触发和重传丢失的包。在这个阶段，TCP 再次慢速启动。

发送端有一个快速方法来识别它的包已经被丢失。当丢失数据包的后续数据包到达接收端时，它们触发给发送端返回确认。这些确认段携带着相同的确认号，称为**重复确认**（duplicate acknowledgement）。发送端每次收到重复确认时，很可能另一个包已经到达接收端，而丢失的那个包仍然没有出现。

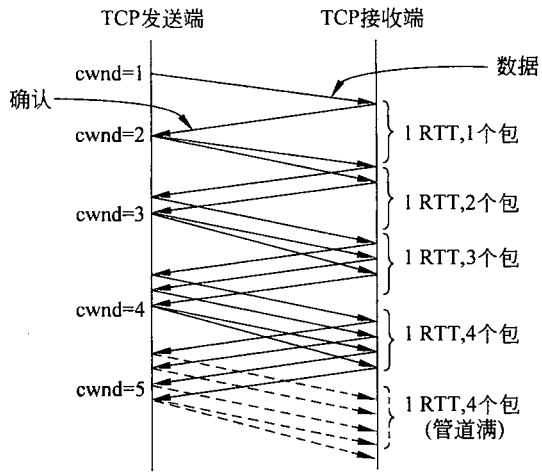


图 6-45 初始拥塞窗口为一个段的线性增长过程

因为包可以选择网络中不同的路径，所以它们可能不按发送顺序到达接收端。这样就会触发重复确认，即使没有数据包被丢失。然而，这种情况在 Internet 上并不多见。当存在跨越多条路径的包时，收到的数据包通常需要重新排序的不会太多。因此，TCP 有点随意地假设三个重复确认意味着已经丢失一个包。丢失包的序号可以从确认号推断出来，它是整个数据序列中紧接着的下一个数据包。因此，这个包可以被立即重传，在其计时器超时前就重新发送出去。

这种启发式机制称为**快速重传**（fast retransmission）。重传后，慢启动阈值被设置为当前拥塞窗口的一半，就像发生了超时一样。重新开始慢启动过程，拥塞窗口被设置成一个包。有了这个窗口大小，如果在一个往返时间内确认了该重传的数据包以及丢包之前已发送的所有数据，则发出一个新的数据包。

图 6-46 给出了迄今为止我们已经构建的拥塞算法示意图。此版本的 TCP 称为 TCP Tahoe，它包括了 1988 年发布的 4.2BSD TCP Tahoe。这里的最大段长为 1 KB。最初，拥塞窗口为 64 KB；但发生超时后，这样的阈值被设置为 32 KB，并且拥塞窗口被设置为 1 KB，从 0 开始传输。拥塞窗口呈指数增长，直到达到阈值（32 KB）。每次到达一个新的确认就增加窗口的大小，而不是连续增加，因而导致窗口的大小表现出离散的阶梯模式。当窗口大小达到阈值后，窗口大小就按线性增长。每个 RTT 只增加一段。

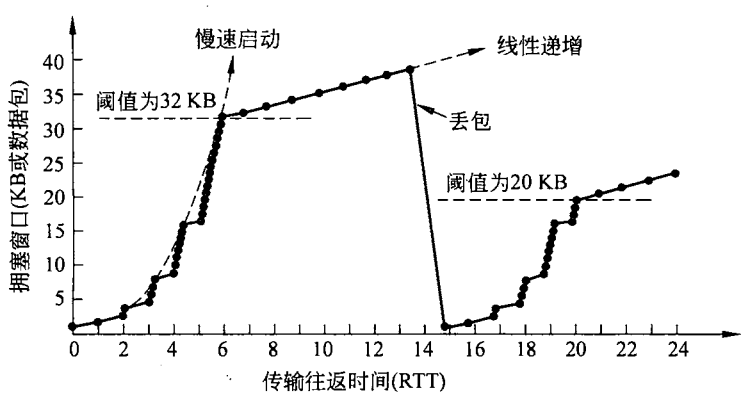


图 6-46 TCP Tahoe 的慢速启动后面接着线性递增

第13轮传输很不幸（它们应该都知道），其中一个包在网络中丢失了。当到达了3个重复确认时，发送端就检测出丢包行为。此时，发送端重传丢失的包，并将阈值设置为当前窗口的一半（现在是40 KB，因此一半是20 KB），并再次开始慢速启动算法。重新启动时，拥塞窗口大小为一个数据包，在往返时间内，所有以前传输的数据要离开网络并得到承认，而且重传数据包也要得到确认。拥塞窗口的增长与以前慢速启动时一样，直到它达到20 KB的新阈值；然后，窗口再次以线性模型增长。窗口持续以线性增长直到通过重复确认检测到另一个丢包或超时（或者接收端的窗口达到上限）。

TCP Tahoe（其中包括良好的重传计时器）提供了一个可实际工作的拥塞控制算法，它解决了网络的拥塞崩溃问题。Jacobson意识到它有可能做得更好。在快速重传时，连接还工作在一个太大的拥塞窗口，但它仍然同样以确认时钟的速率在运行。每次到达另一个重复确认时，可能另一个数据包已经离开网络。使用重复确认来计算网络中的数据包，有可能让一些包离开网络，并继续为每一个额外的重复确认发送一个新的包。

快速恢复（Fast recovery）就是实现这种行为的启发式机制。这是一个临时模式，其目的是保持拥塞窗口上运行确认时钟，该拥塞窗口有一个新阈值或者快速重传时把拥塞窗口值减半。要做到这一点，对重复确认要计数（包括触发快速重传机制的那三个重复确认），直到网络内的数据包数量下降到新阈值。这大概需要半个往返时间。从此时往后，每接收到一个重复确认就发送一个新的数据包。快速重传之后的一个往返时间后，丢失的包将被确认。在这个时间点，重复确认流将停止，快速恢复模式就此退出。拥塞窗口将被设置到新的慢启动阈值，并开始按线性增长。

这种启发式的结果是TCP避免了慢速启动，只有第一次启动或者发生超时时才进入真正的慢速启动。当发生多个数据包丢失时，快速重传机制不足以恢复，仍然可能会发生超时。此时，不是反复地进入慢速启动，而是当前连接的拥塞窗口遵循一种**锯齿**（*sawtooth*）模式，即加法递增（每个RTT增加一段）和乘法递减（每个RTT减半）。这正是我们力求实现的AIMD规则。

这种锯齿行为如图6-47所示。它由TCP Reno提供，这是以1990年发布的4.3BSD Reno命名的，在那个早期版本中包括了上述机制。TCP Reno基本上是TCP Tahoe加上快速恢复机制。经过一个初始的慢速启动，拥塞窗口大小直线攀升，直至通过重复确认检测到丢包。丢失的数据包被重传，并且启用快速恢复机制来维持确认时钟运行，直到重传的数据包被

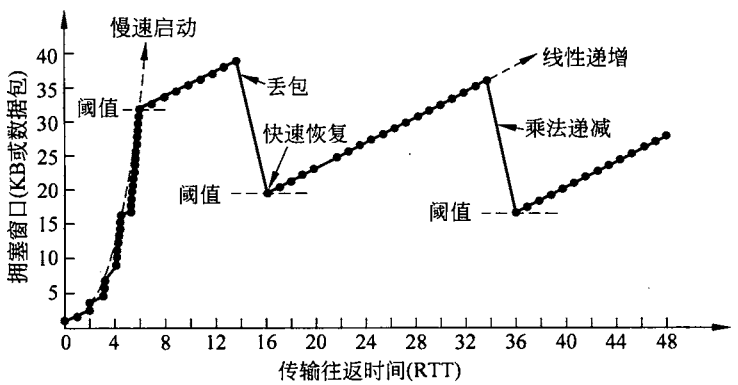


图6-47 TCP Reno的快速恢复和锯齿模式

确认。此时，拥塞窗口恢复到新的慢启动阈值，而不是从 1 开始。这种行为无限期地继续下去，在连接花费的大部分时间内其拥塞窗口接近于带宽延迟积的最佳值。

TCP Reno 及它用来调整拥塞窗口的机制形成了 TCP 的拥塞控制，至今已有超过二十年的历史。在这期间的大多数变化只是以轻微的方式调整这些机制，例如，修改了初始窗口的选择和消除了各种含糊不清的定义。一些改进针对从数据包窗口中两个或两个以上数据包的丢失中恢复过来。例如，TCP NewReno 版本在重传后使用部分提前的确认号来发现并修复另一个丢失的包（Hoe，1996），RFC 3782 对此进行了详细的描述。自 20 世纪 90 年代中期以来，出现了一些 TCP 变种，所有这些变种都遵循我们描述过的原则，但使用了略微不同的控制法则。例如，Linux 使用的 TCP 变种称为 CUBIC TCP（Ha 等，2008），Windows 则包括了一个称为组合 TCP（Compound TCP）的变种（Tan 等，2006）。

两个较大的变化对 TCP 的实现也有影响。首先， TCP 的许多复杂性来自于从一个重复确认流中推断出已经到达和已经丢失的数据包。累计确认号无法提供这种确切的信息。一个简单的解决方法是使用**选择确认**（SACK，Selective ACKnowledgement），该确认列出了 3 个已接收的字节范围。有了这个信息，发送端在实现拥塞窗口时可以更直接地确定哪些数据包需要重传，并跟踪那些还在途中的数据包。

当发送端和接收端建立连接时，它们各自在 TCP 段的选项字段发送允许 SACK，通知对方它们理解选择确认。一旦在连接上启用 SACK，它的工作原理如图 6-48 所示。接收端在正常的方式下使用 TCP 的确认号字段，作为已收到的最高顺序字节的累积确认。当它接收到乱序的数据包 3 时（因为数据包 2 丢失了），它发送一个确认段，其中包括针对包 1 的（重复）累计确认和包 3 的 SACK 选项。SACK 选项给出高于累计确认已收到的字节范围。第一个范围是引发重复确认的数据包；接下来的范围，如果存在的话，是旧的数据块。通常采用三个范围。当收到包 6 时，要用到两个 SACK 的字节范围，来指明包 6 和包 3～包 4 已收到，还有额外的到包 1 的所有数据包。从它接收到的每个 SACK 选项中包含的信息，发送端可以决定重发哪些数据包。在这种情况下，重传数据包 2 和包 5 将是一个好主意。

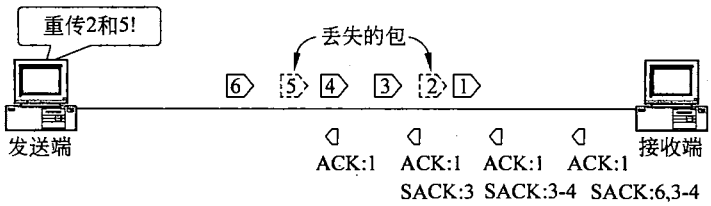


图 6-48 选择确认

SACK 是严格意义上的咨询信息。实际上使用重复确认来检测丢包和调整拥塞窗口还是像以前一样。然而，有了 SACK，TCP 可以更加容易地从同时丢失多个包的情况下恢复过来，因为 TCP 发送端知道哪些数据包尚未收到。SACK 目前已被广泛部署。RFC 2883 对 SACK 做了描述，而关于 TCP 拥塞控制如何使用 SACK 则在 RFC 3517 中描述。

第二个变化是除了用丢包作为拥塞信号外，**显式拥塞通知**（ECN，Explicit Congestion Notification）的使用。ECN 是 IP 层的机制，主要用来通知主机发生了拥塞，我们在 5.3.4 节中有过描述。有了它，TCP 接收端可以接收来自 IP 的拥塞信号。

当发送端和接收端在建立阶段过程中设置了 ECE 和 CWR 标志位，双方均表示它们能

够使用这些标志位后，该 TCP 连接就可启用 ECN。如果使用 ECN，每个携带 TCP 段的数据包在 IP 头打上标记，表明它可以携带 ECN 信号。支持 ECN 的路由器在接近拥塞时就会在携带 ECN 标志的数据包上设置拥塞信号，而不是在拥塞发生后丢弃这些数据包。

如果到达的任何数据包携带了 ECN 拥塞信号，则会告知 TCP 接收端。然后接收端使用 ECE（ECN Echo）标志位给 TCP 发送端发通知，告知它的数据包经历了拥塞。发送端通过**拥塞窗口减少**（CWR，Congestion Window Reduced）标志位告诉接收端它已经收到拥塞信号了。

TCP 发送端收到这些拥塞通知消息的反应和它根据重复确认检测到丢包的处理方式完全相同。不过，严格来说这种情况更好。拥塞已经被检测出来，而且没有包受到任何方式的损害。ECN 的详细描述见 RFC 3168。它要求主机和路由器都必须支持该功能，目前尚未在 Internet 上广泛应用。

有关 TCP 实现中拥塞控制行为的详细信息，请参见 RFC 5681。

6.5.11 TCP 未来

作为 Internet 的主力军，TCP 已经被用于许多应用，并随着时间的推移做了许多扩展以便在广泛的网络上性能表现良好。许多实际部署的版本在实现上跟我们所描述的经典算法略微有所不同，尤其是拥塞控制和针对攻击的鲁棒性方面。很有可能 TCP 将伴随着 Internet 的发展而持续地改进着。这里，我们会特别提到两个问题。

第一个问题是 TCP 没有为所有应用程序提供所需的传输语义。例如，某些应用程序需要保留其发送的消息或记录边界。其他的应用程序要处理一组相关的对话，如 Web 浏览器从同一台服务器传输几个对象。还有其他应用程序希望更好地控制它们使用的网络路径。但是 TCP 的标准套接字接口并不满足这些需求。从本质上讲，应用程序承担着解决任何 TCP 没有解决的问题的负担。这导致了研究者们提出了具有稍微不同接口形式的新协议的建议，其中两个比较著名的例子是由 RFC 4960 定义的**流控制传输协议**（SCTP，Stream Control Transmission Protocol）和**结构化流传输**（SST，Structured Stream Transport）（Ford，2007）。然而，每当有人提出要对一些已经很好地工作了很长时间的东西改革时，总会爆发“用户要求更多的功能”和“如果不打破就不能解决”两大阵营之间的巨大战争。

第二个问题是拥塞控制。你或许期望经过我们的审议以及随着时间推移而开发出来的一系列机制已经解决了这个问题。事实并非如此。我们介绍的 TCP 拥塞控制的形式虽然已经被广泛使用，但这是基于丢包作为拥塞的信号。当（Padhye 等，1998）根据锯齿模式为 TCP 吞吐量建立模型时，他们发现随着速度的加快丢包率呈现急剧下降的趋势。对于往返时间为 100 毫秒，数据包长度为 1500 字节，要达到 1 Gbps 的吞吐量，大约每隔 10 分钟要丢一个包。这种情况下的丢包率大约是 2×10^{-8}，显然这个数字令人难以置信的小。把丢包率作为一个良好拥塞信号实在是太罕见了，任何其他来源的损失（例如，数据包的传输误码率 10^{-7}）可以很容易地发挥主宰作用，由此真正限制了吞吐量。

这种关系在过去不是一个问题，但网络速度越来越快，导致许多人重写拥塞控制。一种可能性是干脆使用另一种拥塞控制，不把丢包当作拥塞信号。我们在 6.2 节中给出了几个例子。拥塞信号可以是往返时间，当网络变得拥塞时往返时间会增长，如 FAST TCP 所

用的那样（Wei 等，2006）。其他方法也是可能的，时间会告诉我们什么是最好的控制方式。

6.6 性 能 问 题

在计算机网络中，性能问题非常重要。当成百上千台计算机相互连接在一起时，无法预知结果的复杂交互过程很常见。这种复杂性常常会导致很差的性能，而且无人知道其中的缘由。在下面的章节中，我们将讨论许多与网络性能有关的问题，以便了解可能存在哪些问题以及如何处理这些问题。

不幸的是，理解网络性能更像是一门艺术，而不是一门科学。这里很少有可在实践中应用的基础理论。我们能够做的最好方法是给出一些来自于实践的经验规则，并且展示一些真实世界中的例子。我们有意将这部分讨论推迟到学习了 TCP 的传输层之后，因为应用程序获得的性能依赖于传输层、网络层和链路层的结合，而且能够在不同场合中使用 TCP 作为例子。

在下面的章节中，我们将考查网络性能的如下 6 个方面。

（1）性能问题。

（2）网络性能的测量。

（3）快速网络中的主机设计。

（4）快速处理段。

（5）头的压缩。

（6）“长肥”网络的协议。

上述这些方面同时从主机和网络，以及网络速度和规模增长来考虑网络性能。

6.6.1 计算机网络中的性能问题

有些性能问题（比如拥塞）是由于临时资源过载所引起的。如果到达一台路由器的流量突然超过了它的处理能力，那么，拥塞就会发生，从而导致性能问题。在本章和前一章中我们已经详细地学习了拥塞控制。

当网络中存在结构性的资源不平衡时，性能也会退化。例如，如果将一条千兆位的通信线路连接到一台低档的 PC 上，则性能较差的 CPU 将无法快速地处理入境数据包，因此有些数据包将会被丢失。这些数据包最终会被重传，既增加了延迟，又浪费了带宽，而且往往降低了性能。

过载也可能被同步触发。例如，如果一个段包含了一个坏参数（比如，它的目标端口），那么，在许多情形下，接收端会尽职地返回一个错误通知。现在请考虑，如果将一个坏段广播给 1000 台机器将发生什么样的情况：每台机器都可能送回一个错误消息。由此引发的**广播风暴**（broadcast storm）可能会严重地削弱网络的性能。以前，UDP 协议就遭受过这个问题，直到 ICMP 协议作了修改，使主机不再对发送给广播地址的 UDP 段中错误做出响应，以免遭受广播风暴。对于无线网络来说，由于广播本质上会发生，而且无线带宽有限，因此尤其要小心避免未经检查的广播响应。

同步触发过载的另一个例子是掉电之后发生的情形。当电源恢复时，所有的机器同时启动系统。一个典型的启动序列可能如下：首先与某一台（DHCP）服务器联系，以便获得一个真实的身份；然后与某一台文件服务器联系，以便获得操作系统的一份副本。如果一个数据中心的成千上万台机器同时做这些事情，则服务器可能因不堪重负而崩溃。

即使不存在同步过载，并且资源也足够，同样也可能由于缺少系统总体协调而发生性能退化的现象。例如，如果一台机器有足够的CPU能力和内存，但是并没有为缓冲区空间分配足够多的内存，则流量控制机制将放缓段的接收，从而限制了传输性能。当Internet变得越来越快，而流量控制窗口仍然维持在64 KB大小时，许多TCP连接都存在这个问题。

另一个调整问题是超时间隔的设置。当一个段被发送出去时，TCP通常会设置一个计时器来预防该段的丢失。如果超时间隔设置得太短，将会发生不必要的重传，从而堵塞线路；如果超时间隔设置得太长，当段被丢失之后将导致不必要的延迟。其他可以调整的参数包括捎带确认应该等待数据段多长时间之后未果才发送单独的确认以及应该尝试经过多少次重传之后再放弃。

对于诸如音频和视频的实时应用，它们的另一个性能问题是抖动。仅仅有足够的平均带宽还不足以有良好的性能。这些应用还要求较小的传输延迟。要想同时获得较短的延迟必须小心规划网络中的流量负载，同时还需要链路层和网络层共同支持服务质量。

6.6.2 网络性能测量

当一个网络的运行效果很差时，它的用户通常会向网络运营商抱怨并要求提高网络质量。为了改善网络性能，网络操作人员首先必须弄清楚发生了什么问题。为了找出真正的问题所在，操作人员需要对网络进行测量。在本小节中，我们来考查网络性能的测量问题。下面的许多讨论以（Mogul，1993）的工作为基础。

测量工作可以有许多不同的方式，也可以在不同的地点进行（既指物理位置，也指协议栈中的位置）。最基本的一种测量手段是在开始某一个动作时启动一个计时器，然后确定该动作需要多长时间。例如，知道一个段需要多长时间才能被确认是很关键的一个测量指标。其他一些测量指标可以通过计数器来完成，即记录某种事件发生的次数（比如丢失的段数量）。最后，人们通常对于某些事物的数量比较感兴趣，比如在特定的时间间隔内所处理的字节数。

测量网络的性能和参数有许多潜在的陷阱。以下我们列出其中一部分。任何一种系统化的网络性能测量手段都应该小心地避免这些陷阱。

确保样值空间足够大

不要仅仅测量发送一个段的时间，而是要重复地测量，比如说测量100万次，然后再取平均值。启动效应可能放慢第一个段的速度，而且排队会引入变化，比如802.16 NIC或者线缆调制解调器在一个空闲周期后获得带宽预留。采用大量的样本可以减小测量均值和标准方差中的不确定性。这种不确定性可以利用标准的统计公式计算出来。

确保样值具有代表性

理想情况下，这一百万次测量的完整序列应该分布在一天或者一周的不同时刻重复进行，以此来看不同的网络条件对测量数值的影响。例如，对于拥塞的测量，如果仅仅在没有拥塞的那一刻来测量网络拥塞，那么这样的测量以及结果没什么用。有时候测量结果初看起来可能不符合直觉，比如在上午 11 点和下午 1 点钟网络严重拥塞，但是中午时候却没有拥塞（所有的用户都去吃午饭了）。

在无线网络中，信号的传播位置是一个很重要的变量。由于天线的差异，即使测量节点放置在靠近无线客户端也可能无法观察到与客户端相同的数据包。在研究中最好从无线客户端进行测量，考查它究竟看到了什么。如果做不到这一点，那就使用技术把从不同角度的无线测量结合起来以便对正在发生的事情有个更全面的了解（Mahajan 等，2006）。

缓存可以破坏测量结果

如果协议使用了缓存机制，那么重复多次测量将返回一个不可预测的快速答复。例如，获取一个 Web 页面或者查询一个 DNS 名字（来发现其 IP 地址）可能只有第一次涉及网络交换，以后返回的都是从缓冲区获得的结果，没有真正往网络上发送任何数据包。这样的测量结果本质上是没有什么价值的（除非你要测量的就是缓冲性能）。

缓冲机制也有类似的影响。已经有 TCP/IP 性能测试报告称 UDP 可以获得比网络允许的要好得多的性能。这是怎么发生的呢？调用 UDP 时一旦内核接受了消息并且消息被排入传输队列之后，通常控制权就马上返回给应用程序了。如果主机上有足够的缓冲空间，则执行 1000 次 UDP 调用也不意味着所有的数据都已经被发送出去。大多数数据包仍然在内核中，但是性能测试程序却认为它们都已经被传送出去了。

建议测试时，你要绝对确保网络操作是如何缓存以及缓冲数据的。

确保测试期间不会发生不可知事情

测量时如果有人在你的网络上运行一个视频会议，那么通常得到的测量结果和没有视频会议时的测量结果不同。最好在一个空闲的网络上进行测试，并且根据需要自己创建所需的工作负载。不过这种做法也存在缺陷。尽管你认为在凌晨 3 点钟时不会有人使用网络，但有可能自动备份程序恰好在这个时间点开始将所有的磁盘数据复制到磁带上。或者，其他时区的用户可能来访问你制作精美的 Web 页面，也会导致繁重的流量。

无线网络在这方面更具挑战性，因为它往往无法区分各种干扰源。即使附近没有其他无线网络在发送流量，或许有人在用微波炉爆玉米花，不经意间造成了干扰，从而降低 802.11 性能。基于这些原因，良好的做法是监控整个网络的活动，这样你至少可以认识到何时发生了意想不到的事情。

小心使用粗粒度时钟

计算机时钟的功能是以固定的时间间隔递增某个计数器。例如，一个毫秒计时器每隔 1 毫秒计数器就加 1。使用这样的计时器来测量一个持续时间小于 1 毫秒的事件是有可能的，但要非常小心。当然，有些计算机有更精确的时钟，但总会有时间更短的事件需要测量。请注意，时钟并不总是和返回的时间一样精确。

例如，为了测量建立一个连接所需要的时间，应该在进入传输层代码以及离开传输层

代码时，分别读出时钟值（比如说以毫秒为单位计数）。如果真正的连接建立时间是300微秒，则两次读取的时间差值要么是0，要么是1，显然这两个结果都是错误的。然而，如果重复测量一百万次，把所有测量的总和累加起来，再除以一百万，则平均时间比1微秒还要精确得多。

小心推断结果

假设你利用模拟的网络负载来进行测量，网络负载从0（空闲）到0.4（40%的容量）。例如，在802.11网络上发送IP语音包可以用如图6-49中的数据点和实线表示。如果能推断出随着负载增加而响应时间呈线性增长，如图中点线所示那样，那么结果是很诱人的。然而，许多排队论的研究结果都涉及一个1 / (1−ρ)因子，其中ρ是负载，所以真正的值可能看起来更像是虚线，当负载变高时其上升速度远远高于线性增长。也就是说，竞争的影响在高负载下更为明显。

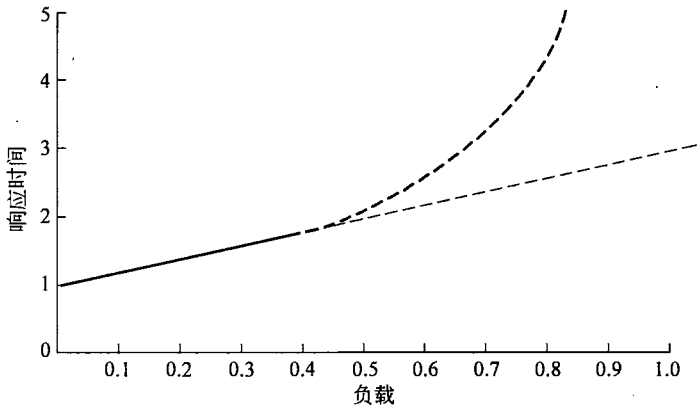


图6-49 响应时间是负载的函数

6.6.3 针对快速网络的主机设计

测量性能并进行修补可以在一定程度上提高网络性能，但是，它们不可能代替一个在一开始就有的良好设计。一个设计不佳的网络其得以改进的余地毕竟有限，除此之外，它必须从头开始重新设计。

在本节中，我们将就主机实现网络协议提出一些经验法则。令人惊奇的是，经验表明通常的性能瓶颈不在快速网络而在于主机。这里存在两个原因。首先网络接口卡（NIC，Network Interface Cards）和路由器在工程上早已经能以“线速率”（wire speed）运行。这意味着它们处理数据包的速度和数据包到达链路的速度一样快。其次，相关的性能是指应用程序能获得的。它不是链路容量，而是经过网络和传输层处理之后的吞吐量和延迟。

减少软件开销可以提高吞吐量和减少延迟，从而达到提高网络性能的目的。同时，它还能减少花在网络上的能量，能耗这个因素对于移动计算机是很重要的考虑。这些想法中的大多数多年来早已成为大多网络设计师必守的常识。它们首次由（Mogul，1993）明确提出，我们主要遵循他的观点。另一个相关的知识源来自（Metcalfe，1993）。

主机速度比网络速度更重要

长期的经验表明在几乎所有的高速网络中，操作系统和协议开销占据着线路的主要实际时间。例如，从理论上讲，1 Gbps 以太网上的最小 RPC 时间是 1 微秒，对应着最小（512 字节）的请求和后续的一个最小（512 字节）应答。实际上，克服了软件开销，随时随地使得 RPC 时间接近理论值都是一个了不起的成就，但却很少真正发生。

类似地，在 1 Gbps 上运行的最大问题是应该足够快地将比特从用户缓冲区中取出放到网络上，以及接收主机以比特到达的速度来处理它们。如果你将主机（CPU 和内存）速度加倍，那么，你通常可以获得接近两倍的吞吐量。如果瓶颈在主机，那么加倍网络容量是没有效果的。

减少包技术来降低开销

每个段都有特定数量的开销（例如，头）以及数据（比如有效载荷）。这两部分都需要消耗带宽，同时也都需要处理（例如，处理段头和计算校验和）。发送 100 万字节时，不管段的大小是多大，数据成本是相同的。然而，长度为 128 字节的段意味着每个段的开销是长度为 4 KB 段的开销的 32 倍。带宽和处理开销加起来极大地降低了吞吐量。

每个包在较低层次的开销放大了这种效果。如果主机速度跟得上，每个到达的数据包都会导致一次新的中断。在一个现代化的流水线处理器中，每次中断都要中断 CPU 管道、干扰缓存、需要改变内存管理的上下文、废除分支预测表并强制保存大量的 CPU 寄存器。因此把发送的段减少 n 倍，可以降低 n 倍的中断和数据包开销。

你可能会说，人与计算机执行多任务的能力都很差。这种观察奠定了发送尽可能大 MTU 数据包的需求，这样的 MTU 能通过网络路径而无须分片。诸如 Nagle 算法和 Clark 解决方案的一些机制都试图避免发送小的数据包。

最小化数据预取

实现一个分层协议栈的最简单方法是每一层用一个模块实现。不幸的是，这将导致每一层重复许多工作（或至少通过多次传递才能访问到数据）。例如，在 NIC 收到一个数据包后，它通常是把该包复制到内核缓冲区。从那里，它再被复制到一个网络层的缓冲区，供网络层实体处理；然后该包又被复制到传输层缓冲区，由传输层来处理；最终，由应用程序接收。对于一个入境数据包来说，在将它包含的段递交出去之前被复制三次或四次并不罕见。

所有这些复制都大大降低了协议性能，因为内存操作比寄存器指令慢了一个量级。例如，如果 20%的指令要对实际内存进行操作（即未命中高速缓存），这在预取入境数据包时是很有可能的，则平均指令执行时间要放慢 2.8 倍（$0.8\times1+0.2\times10$）。这里，硬件辅助将无能为力。这个问题的根源在于操作系统执行了太多的复制操作。

一个聪明的操作系统通过把多个层次的处理结合在一起，以此来最小化复制操作。例如，TCP 和 IP 通常实现在一起（正如 TCP/IP 所示的那样），这样当从网络层的处理切换到传输层的处理时，不再需要复制数据包的有效载荷。另一种常见的伎俩是在传递单个数据时在一层内执行多个操作。例如，在复制数据时通常要计算校验和（当必须要复制时），新近计算出的校验和可贴在尾部。

最小化上下文切换

一个相关的规则说明上下文切换（例如，从内核模式到用户模式）是致命的。它们把中断和复制操作结合在一起，而这是个很坏的属性。这个成本就是为什么传输协议通常实现在内核中的理由。就像减少包计数一样，发送数据的库函数可以在内部进行缓冲直到它有大量的数据再发送，通过这种内部缓冲机制可以减少上下文切换。同样，在接收端，小的入境段应收集在一起一举传递给用户而不是个别地传递，以此最大限度地减少上下文切换。

在最好的情形下，一个入境数据包导致从用户模式到内核模式的一次切换，然后再切换到接收进程并将新到达的数据交给它。不幸的是，在许多操作系统中，除此以外还需要额外的上下文切换。例如，如果网络管理器以特殊进程的形式运行在用户空间中，那么，一个数据包的到来可能引发一次从当前用户到内核的上下文切换；然后从内核切换到网络管理器；之后再切换回内核；最后从内核切换到接收进程。这个切换过程如图6-50所示。在每个数据包上的所有这些上下文切换对CPU时间是个严重浪费，同时也严重地影响了网络性能。

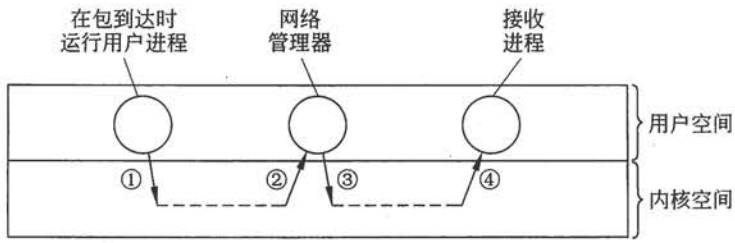


图6-50 处理一个数据包要经历从用户空间到网络管理器的4次上下文切换

避免拥塞比从中恢复更好

“一分预防胜过十分治疗”这句古老的格言无疑特别适用于网络拥塞。当网络拥塞时，数据包被丢失，带宽被浪费，还引入了无用的延迟等一系列问题。所有为此付出的成本都是没有必要的，而且从拥塞中恢复还需要时间和耐心。最好的做法是从一开始就不让拥塞发生。避免拥塞就好像你接种DTP疫苗：当你接种的时候你会感觉疼一下，但在将来使你避免更多的疼痛。

避免超时

计时器在网络中是必要的，但应该尽量少用计时器，而且应该尽量少发生超时。当一个计时器超时，通常需要重复执行某个动作。如果确实需要重复执行这个动作，则执行该动作；否则，不必要的重复就是一种浪费。

避免额外工作的办法是小心设置计时器的超时值，将计时器设置成稍稍超过一点保守时间边界。如果将计时器的超时间隔设置得太长，那么所在的连接（不太可能）出现段丢失事件时会增加少量的额外延迟。如果一个计时器过早超时，则会消耗掉主机资源、浪费带宽，而且毫无理由地给很多路由器增加额外的负载。

6.6.4 快速处理段

既然我们已经覆盖了通用规则，我们将考查某些加快段处理速度的特殊方法。要想了

解更多的信息，请参考（Clark 等，1989；Chase 等，2001）。

段的处理开销由两部分组成：每个段的处理开销和每个字节的处理开销。这两部分都有改进的余地。快速处理段的关键是分离出一般情况和成功情况（单向数据传输），并对它们作特殊处理。许多协议倾向于强调出现某种错误情况时该怎么处理（比如，丢失了一个数据包），但是若要使得协议快速运行，协议的设计者应该把目标定在一切正常运行时如何最小化处理时间，而把出现问题时如何最小化处理放在次要地位考虑。

尽管为了进入 ESTABLISHED 状态需要一系列特殊的段，但一旦进入该状态，段的处理就是直截了当的，直到连接的某一端关闭连接。让我们从 ESTABLISHED 状态下的发送端开始，看看它有数据要发送时的情形。为了清楚起见，假设传输实体运行在内核中，尽管同样的想法适用于其他的情形：传输实体是一个用户空间的进程，或者是发送进程内部的一个库。在图 6-51 中，发送进程陷入内核并执行 SEND。传输实体要做的第一件事情是测试，检查是否为正常情形：即状态必须为 ESTABLISHED，两端均不打算关闭当前连接，发送的是一个普通段（即不是带外数据），而且接收端有足够的窗口空间可供使用。如果上述所有的条件都满足，则无须进一步测试，使用发送端传输实体的快速路径。在通常情况下，这条快速路径占据了大部分的处理时间。

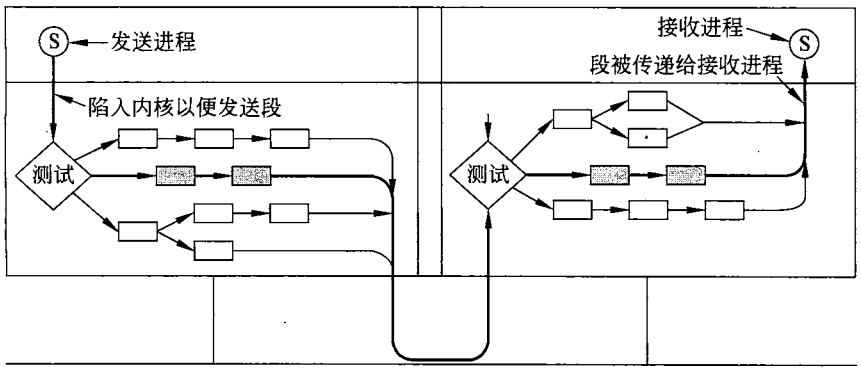
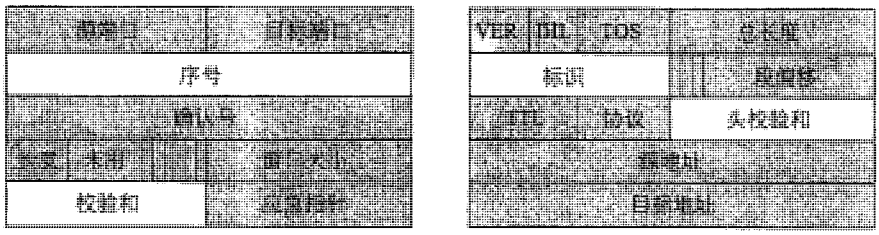


图 6-51 一条重载线路上从发送端到接收端的快速路径（阴影标示了这条路径上的处理步骤）

通常情况下，连续的数据段的头几乎是相同的。为了充分利用这个事实，传输实体在内部保存一个原型头。在快速路径的起始处，该原型头被尽可能快地逐字复制到一个临时缓冲区中。对于那些段与段之间不同的字段，则复制时覆盖掉缓冲区中的相应字段。这些字段往往可以很容易地从状态变量中导出，比如“下一个序号”。然后，一个指向完整段头的指针和一个指向用户数据的指针被传递给网络层；因此，网络层可以遵循同样的策略（图 6-51 没有显示）。最后，网络层将结果数据包交给数据链路层传输。

上述原理在实际中是如何工作的，我们用一个 TCP/IP 的一个例子加以说明。图 6-52（a）显示了 TCP 头。在单向数据流中前后两个连续段有一些字段总是相同的，图中用阴影标识了这样的字段。发送端传输实体所要做的只是：将 5 个字从原型头复制到输出缓冲区中，然后填充下一个序号（从内存中的一个字复制过来）、计算校验和、递增内存中的序号；然后将头和数据交给一个特殊的 IP 过程，由它来发送一个常规的、最大的段。紧接着，IP 将它 5 个字的原型头（如图 6-52（b））复制到缓冲区中，然后填充标识（Identification）字段、计算校验和。现在这个数据包已经可以发送出去了。



(a) (b)

图 6-52
（a）TCP 头；（b）IP 头
这两种情况下它们都取自原型，没有改变

现在让我们来看图 6-51 中接收端的快速路径处理过程。第 1 步是找到与入境段相对应的连接记录。对于 TCP，连接记录可能被保存在一张哈希表中，该哈希表把两个 IP 地址和两个端口作为关键字执行某个简单函数。一旦在该表中找到相应的连接记录，则比较两个地址和两个端口，以便验证找到了正确的连接记录。

加速连接记录查询过程的一种优化措施是维护一个指向最近使用过的连接记录的指针，并且在开始查找时首先使用它。（Clark 等，1989）试用了这种方法，发现它的命中率超过 90%。

找到了连接记录后，接收端就检查入境段，看它是否正常：连接状态是 ESTABLISHED、连接的两端均不打算关闭该连接、段完好无缺、段没有特殊的标志位，并且它的序号正好是接收端所期望的。这些测试都是一些指令。如果所有的条件都满足，则调用一个特殊的快速路径 TCP 过程。

快速路径更新连接记录，并且将数据复制给用户。在复制数据时，它同时计算校验和，从而消除了对数据的再次遍历操作。如果校验和正确，则更新连接记录，并返回发送端一个确认。这种首先快速检查头然后再详细处理的通用模式称为**头预测**（header prediction），即首先快速地检查段的头，确定该段是否自己所期望的段，然后再调用一个特殊的过程来处理。许多 TCP 实现都使用了这种方法。当这种优化方案与本章中讨论的其他优化方案一起使用后，有可能使 TCP 的运行速度达到本地内存到内存复制速度的 90%，这里假设网络本身的速度足够快。

两个方面的性能有可能获得较大幅度的提高，它们分别是缓冲区管理和计时器管理。缓冲区管理中的主要问题是避免不必要的复制操作，正如我们上面介绍过的那样。计时器管理非常重要，因为几乎所有的计时器设置都不会超时。设置计时器是为了防止段的丢失，但是，大多数段和它们的确认都会正确地到达目的地。因此，在计时器很少出现超时的情况下，优化计时器的管理尤为重要。

一种常见的方案是用链表结构将计时器事件按超时值进行排序保存。链表中的头表项包含一个计数器，指明了离超时时间还有多少个滴答数。每个后续表项也包含一个计数器，指明在前一个表项超时后还有多少个滴答。因此，如果 3 个计时器分别在 3、10 和 12 个滴答之后超时，则 3 个计数器分别为 3、7 和 2。

每滴答一次，表头表项中的计数器被减 1。当它减到 0 的时候，它的事件被处理，链表中的下一个表项变成链表的头。头表项中的计数器值不用改变。在这种方案中，插入和

删除计时器是很昂贵的操作，其执行时间正比于链表的长度。

如果最大的计时器间隔有上界，并且可以预先知道，那么我们可以使用一种更加高效的方法。这里，要用到一个称为**计时轮**（timing wheel）的数组，如图 6-53 所示。每个槽对应于一个时钟滴答。图中显示的当前时间为 T=4。三个计时器分别在从现在开始的 3、10 和 12 个滴答时调度。如果一个新的计时器突然被设置成在 7 个滴答后超时，那么只需在第 11 个槽中加入一个表项即可。类似地，如果要取消一个设置在 T+10 超时的计时器，则只要搜索第 14 个槽中开始的链表，并将所请求的表项移除即可。请注意，图 6-53 中的数组不可能容纳超过 T+15 的计时器。

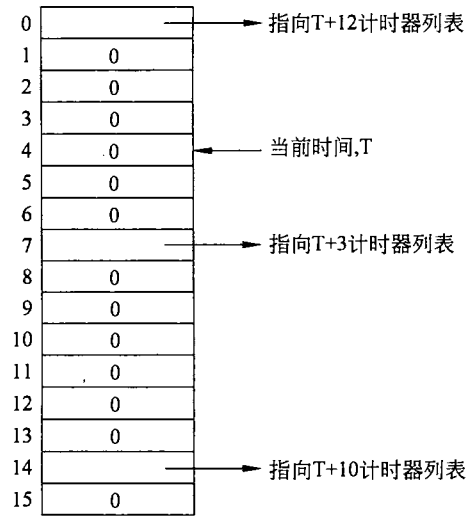


图 6-53 计时轮

当时钟滴答时，当前时间指针向前移动一个槽（循环移动）。如果现在指向的表项不为 0，则该表项中的所有计时器都要被处理。（Varghese 和 Lauch，1987）讨论了基于这个基本思想的许多变种方法。

6.6.5 头压缩

我们很长以来一直在考查高速网络，那里还有更多的内容需要关注。现在让我们考虑无线网络和其他一些网络的性能，这些网络的带宽非常有限。降低软件开销有助于移动计算机运行得更加高效，但当网络链路成为瓶颈时，它对提高性能却是无能为力。

为了更好地使用带宽，协议头和有效载荷应该携带尽可能少的比特。对于有效载荷来说，这意味着必须使用信息的压缩编码，比如 JPEG 格式的图像，而不是位图或者诸如包含了压缩内容 PDF 的文档格式。这也意味着应用程序级的缓存机制应该摆在首位，比如用降低传输的 Web 缓存机制。

协议头怎么样呢？在无线网络中的链路层，典型的头是压缩的，因为它们始终秉承着专为稀缺带宽而设计的思想。例如，802.16 头采用了短的连接标识符，而不是长长的地址。然而，高层协议，比如 IP、TCP 和 UDP 在所有链路层上采用的是同一版本，而且它们不是专为紧凑压缩头设计的。事实上，为减少软件开销而采用的流水型处理往往导致头无法压缩，而本来是可以做到的（例如，IPv6 有一个比 IPv4 更松散的头）。

高层协议的头是一个重要的性能因素。考虑 IP 语音数据，它通过 IP、UDP 和 RTP 几个协议的组合传输。这些协议需要 40 个字节长的头（IPv4 头 20 字节，UDP 协议 8 字节和 RTP 协议头 12）。IPv6 的情况更糟：需要的头长达 60 个字节，其中包括 40 个字节的 IPv6 报头。头占据了传输数据的大部分，并且消耗了一半多的带宽。

头压缩（header compression）技术可用来降低高层协议头消耗的链路带宽，通常采用专门设计的方案，而不是通用的方法。这是因为头一般都很短，所以它们单独压缩的效果不那么好，而且解压缩时要求之前的所有数据都已经收到。显然，如果一个数据包丢失，

就无法满足这个要求。

使用协议格式的知识来压缩头可以获取巨大的收益。第一个方案由（Van Jacobson，1990）设计，对通过低速串行链路上的TCP/IP头进行压缩。它能够将一个典型的40个字节的TCP/IP头压缩到平均3个字节。这种方法的技巧如图6-52给出。一个数据包与另一个数据包之间的许多头字段并没有改变。因为没有必要改变，例如，发送的每一个数据包中具有相同的IP TTL字段或相同的TCP端口号字段。这些字段可以在链路的发送端省略，在链路的另一端接收端填充上。

类似地，其他字段以可预见的方式改变着。例如，除非丢失，TCP序号随着数据发送而向前推进。在这种情况下，接收端可以预测到可能的值。即使如此，它可以是先前值的一个很小的变化，因为随着在相反方向收到的新数据，确认号也随之在递增。

采用头压缩，高层协议的头就有可能非常简单，而且可以在低带宽链路上进行压缩编码。**鲁棒头压缩**（ROHC，RObust Header Compression）是头压缩的现代版本，作为一个框架由RFC 5795定义。它的设计目标是能够容忍发生在无线链路上的丢失。每一组被压缩的协议都有一个轮廓（profile）文件，比如IP/UDP/RTP。最终传输的压缩的头是携带一个指向上下文的引用，它本质上是一个连接；对于同一个连接上的数据包，其头中的字段可以很容易地预测出来，但不同的连接上的数据包头字段则无法预测。在典型的操作中，ROHC将IP/UDP/RTP头从40个字节经过压缩降低到1～3个字节。

虽然头压缩的目标主要针对的是减少带宽需求，但它也有助于降低延迟。延迟由传播延迟和传输延迟两部分数据组成，前者对于给定的网络路径而言是固定的，而后者则取决于带宽和发送的数据量。例如，在1 Mbps链路上发送1比特需要1微秒。在无线网络的介质上，网络速度相对较慢，因此传输延迟成为整体延迟中的一个重要因素，而且低延迟对于服务质量一贯重要。

头压缩技术有助于减少发送的数据，从而降低传输延迟。发送较小的数据包，可以达到同样的效果。为了降低传输延迟，将增加软件开销。请注意，延迟的另一个潜在来源是访问无线链路的排队延迟。这个延迟也很显著，因为作为网络中的有限资源，无线链路往往被过度地使用。在这种情况下，无线链路必须具备服务质量机制，给实时数据包以较低的延迟。仅仅进行头压缩是不够的。

6.6.6 长肥网络的协议

20世纪90年代初，出现了长距离传输数据的千兆网络。由于结合了快速网络或“肥管道”（fat pipeline）和长延迟，这些网络称为**长肥网络**（long fat networks）。当这些网络出现时，人们的最初反应是使用已有的协议，但是很快各种各样的问题随之出现。在这一小节中，我们将讨论随着网络协议的速度和延迟尺度增加会出现的一些问题。

第一个问题是许多协议使用了32位长的序号。当Internet刚开始时，路由器之间的线路主要是56 kbps的租用线路，所以一台全速发送数据的主机需要一星期的时间才可能用完一轮序号（即导致序号回绕）。对于TCP的设计者们来说，2^{32}是一个相当体面的无穷大近似数，因为一周以前发送的老数据包仍然停留在网络中的危险性几乎不存在；对于10 Mbps以太网，序号回绕时间变成了57分钟，尽管时间更短了，但仍然可以管理；对于

1 Gbps 以太网，如果主机全速发送数据到 Internet 上，则序号回绕时间大约是 34 秒钟，大大低于 Internet 上的最大数据包生存期，即 120 秒。突然间，2^{32} 不再是一个良好的无穷大近似数。而且，当老的数据包仍然存在网络中时，快速发送端就开始循环使用序号空间了。

问题在于许多协议的设计者简单地认为使用完整序号空间所需要的时间大大超过包的最大生存期，而且这种假设还没有声明。因此，没有必要甚至不用担心序号回绕时老的重复数据包仍然存在这个问题。但是对千兆位的速度，这种未声明的假设无疑是失败的。幸运的是，已经证明可以通过采用时间戳来扩展有效的序号范围，这个时间戳作为 TCP 头的选项作为以高序列比特的形式被每个数据包携带着。这种机制称为**防止序号回绕**（PAWS，Protection Against Wrapped Sequence numbers），在 RFC 1323 文档中有详细描述。

第二个问题是流量控制窗口的大小必须增长得很快。例如，考虑从圣地亚哥发送一个 64KB 的突发数据到波士顿，以便填满接收端的 64 KB 缓冲区。假设链路速率 1 Gbps，光在光纤中的单向延迟为 20 毫秒。最初，在 t = 0，管道是空的，如图 6-54（a）所示。500 微秒后，如图 6-54（b）所示所有的段都被发送到光纤上了。前面的段现在出现在 Brawley 附件的某个地方，仍然位于南加州的腹地。然而，发送端必须停止，直到它收到一个窗口更新段。

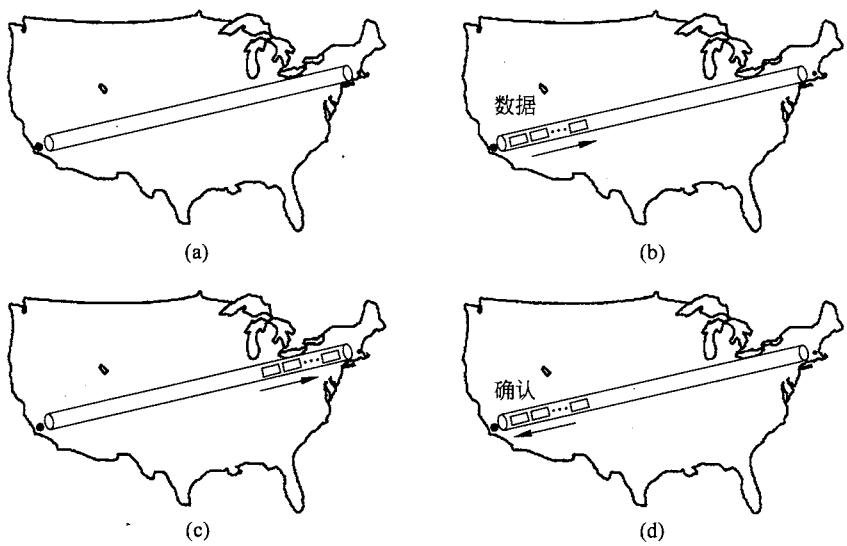


图 6-54 从圣地亚哥给波士顿发送 1 Mb 的状态
（a）在 t = 0 时；（b）500 微秒之后；（c）20 毫秒之后；（d）40 毫秒之后

20 毫秒后，先遣段到达波士顿，如图 6-54（c）所示，并得到确认。最后，在开始发送后 40 毫秒，第一个确认返回到发送端，然后开始发送第二次突发数据块。由于传输线路只使用了 100 中的 1.25 毫秒，因此其效率大约是 1.25%。这是老协议运行在千兆线路上的典型情况。

在分析网络性能时要时刻牢记的一个有用数值——带宽延迟乘积。它是往返延迟时间（秒）乘以带宽（比特/秒）。这个乘积就是从发送端到接收端并且返回的管道的容量（比特）。

对于图 6-54 的例子来说，带宽延迟乘积是 40 个百兆比特。换句话说，发送端必须突发 40 个百兆的比特才能在第一个确认返回前保持全速传输。也就是说，需要这么多个位来填充管道（在两个方向上）。这就是为什么突发百兆位的一半只能达到 1.25%的效率：它仅为管道容量的 1.25%。

这里可以得出的结论是为了获得良好的性能，接收端的窗口必须至少和带宽时延积一样大，最好再大点，因为接收端可能无法立即响应。对于一个跨洲的千兆线路，需要至少5 MB。

第三个相关问题是简单重传策略，比如回退N协议，在一条带宽延迟乘积非常大的线路上性能非常差。考虑一条1 Gbps横贯大陆的线路，其往返传输时间是40毫秒。在这段时间中发送端可以传输5 MB的数据。如果一个错误被接收端检测出来，那么40毫秒以后发送端才能接到错误通知。如果使用回退N协议，则发送端不仅要重传坏的数据包，而且要重传该数据包后面的5 MB数据。很明显，这是资源的极大浪费，因此需要更复杂的协议，例如选择重传协议。

第四个问题是千兆位线路与百兆位线路在本质上的不同，长距离的千兆位线路的制约因素是延迟而不是带宽。在图6-55中，我们看到将一个1 Mb文件以不同的传输速度传输4000千米所需要的时间。在速度增加到1 Mbps以前，传输时间主要受到发送速率的制约；超过1 Gbps时，40毫秒的往返延迟时间远远超过了将数据发送到光纤上所需要的1毫秒时间。此时，进一步增加带宽也不会有任何帮助。

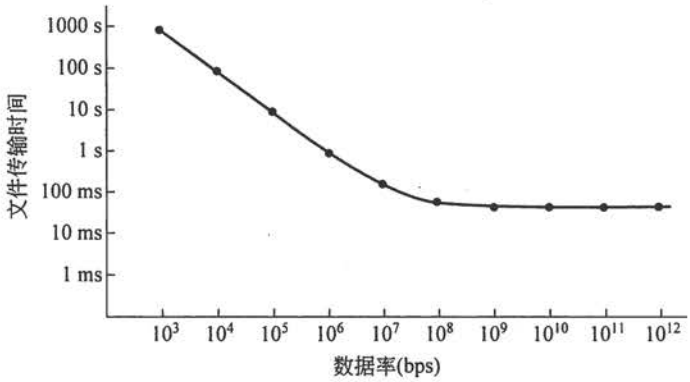


图6-55 在一条4000公里长的线路上发送1M位文件需要的传送和确认时间

图6-55对于网络协议有一些不幸的影响。从图中可以看出，比如像RPC这样的停-等式协议其性能有一个固有的上界。这种限制来自于光速的限制，光学的技术再怎么进步也无法改善这种状况（不过，新的物理法则可能会有所帮助）。除非在一台主机等待应答时千兆线路可以用作一些其他用途，否则千兆线路比百兆线路没什么好的，只是更昂贵而已。

第五个问题是通信速度已经得到很大提高，远远超越了计算速度（计算机工程师们请注意：冲出去，击败那些通信工程师！我们都指望你了）。在20世纪70年代，ARPANET只能运行在56 kbps的线路上，而当时的计算机已经达到1MIPS的计算速度。将这些数字与在1 Gbps线路上交换数据包的1000MIPS计算机进行比较。每个字节的指令数量已经下降了超过10倍。确切的数字值得商榷，具体依赖于测试日期和场景，但由此得出的结论是：协议处理的可用时间少于过去协议处理所用的时间，因此协议必须变得更加简洁。

现在让我们从存在的问题转到处理这些问题的办法上来。所有高速网络设计者必须牢记在心的基本原则是：

为速度而不是为带宽优化进行设计。

老协议的设计目标通常是最小化线路上传输的比特个数，一般的做法是使用小的字段，

并且将这些小的字段组合成字节或者字。这种关注对于无线网络依然有效，但对于千兆位的快速网络不再适用。协议处理是个问题，所以，协议的设计重点变成如何将协议本身最小化。显然，IPv6 的设计者们清楚这条原则。

一种很吸引人的加速办法是用硬件来构建快速网络接口。这种策略的困难在于除非协议极其简单，否则，硬件就是带有第二个 CPU 和自己独立程序的一块插卡。为了确保**网络协处理器**（network coprocessor）比主 CPU 便宜，它通常是一块慢速芯片。这种设计的直接结果是主（快速）CPU 的许多时间是空闲的，它要等待第二个（慢速）CPU 完成关键性的工作。认为主 CPU 在等待期间可以去做其他工作的想法是不切实际的。而且，当两个通用 CPU 相互通信时，可能会产生竞争条件，所以两个处理器之间的通信协议需要精心设计，以便保证两者能正确地同步并且避免竞争。一般来说，最好的途径是使协议尽可能的简单，并且让主 CPU 来完成协议处理工作。

数据包的布局结构也是千兆网络中一个很重要的考虑因素。头应该尽可能包含少的字段，以便减少包的处理时间，而且头中的这些字段应该足够大到正常工作，另外，这些字段应该按字对齐以便于快速处理。在这里的上下文中，“足够大”意味着类似于下面的问题不会发生：比如序号回绕时老的数据包仍然存在，或者由于窗口字段太小导致接收端不能宣告足够大的窗口空间。

为了降低软件开销和有效操作，最大的数据长度应该非常大。对于高速网络来说，1500 个字节太小，这就是为什么千兆以太网支持大到 9KB 的巨型数据帧以及 IPv6 支持大于 64 KB 的大块数据包传输的理由。

现在我们来看高速协议中的反馈问题。由于（相对）较长时间的延迟回路，应该尽可能地避免反馈：接收端给发送端发送信号需要太长的时间。第一个反馈例子是利用滑动窗口协议来控制传输速率。为了避免由于接收端给发送端发送窗口更新信息而造成固有的（长时间）延迟，未来的协议最好使用基于速率的协议，这样才能避免接收端发送窗口更新段给发送端所带来的固有的（长）延迟。在这样的协议中，发送端可以发送所有它希望传送的数据，只要发送速率不超过双方预先协商好的速率即可。

第二个反馈例子是 Jacobson 的慢速启动算法。这个算法需要执行多次探测来确定网络的处理能力。对于高速网络，执行几次小的探测才能确定网络的响应能力将浪费大量带宽。一种更加有效的方法是让发送端、接收端和网络三者在建立连接时都预留下必要的资源。提前预留资源还有另一个好处，就是更加容易减少抖动。简而言之，网络越是向高速发展，设计就越是被无情地推向面向连接的操作，或者非常接近于面向连接的操作。

另一个有价值的特性是在连接请求过程中顺带发送正常量数据的能力。按照这种方法，可以节省一次往返时间。

6.7 延迟容忍网络

我们将通过描述一种全新的传输来结束本章，这种传输有朝一日可能会成为 Internet 的重要组成部分。TCP 和大多数其他传输协议建立在这样的假设基础之上：发送端和接收端通过一些工作路径持续地连接在一起，否则协议就会失败且数据将无法传递。在某些网

络中，往往没有端到端的路径。一个例子是空间网络，比如**低地球轨道**（LEO，Low-Earth Orbit）卫星网络，卫星频繁进出地面站的探测范围。一个给定的卫星或许只能在特定的时间与地面站通信，而两颗卫星可能在任何时间都永远也无法彼此沟通，甚至通过一个地面站作中转也无法直接通信，因为其中一颗卫星可能总是超出地面站范围。其他类似的网络涉及潜艇、公共汽车、移动电话和其他具有计算机的设备，由于移动性或极端条件的限制，使得它们只有间歇性的连通性。

在这些偶尔连接的网络中，可以把数据存储在某些节点，并在随后有工作链路时再转发这些数据，因此数据通信仍然可以进行，这种技术称为**消息交换**（message switching）。最终数据将被中继到目的地。基于这种通信方法构建的网络体系结构称为**延迟容忍网络或中断容忍网络**（DTN，Delay-Tolerant Network, or a Disruption-Tolerant Network）。

DTN 的工作始于 2002 年，当时 IETF 成立了一个研究小组来研究这个主题。DTN 的灵感来自一个不太可能的源头：努力在太空中发送数据包。空间网络必须处理间歇性的通信和很长的延迟。Kevin Fall 观察到这些星际互联网的思想可以应用到地球上那些间歇性连接呈现常态的网络（Fall，2003）。该模型给出了 Internet 的一个有用的概括，即在 Internet 上通信期间可以进行存储和延迟。数据传输是类似于邮政系统或者电子邮件的传递，而不是路由器上的数据包交换。

自 2002 年以来，DTN 体系结构已经经过了细化，DTN 模型的应用也有了很大增长。作为主流应用，考虑由科学实验、媒体事件或基于 Web 服务生成的许多 TB 级的大量数据集，需要将这些数据集复制到位于世界各地不同地点的数据中心。运营商希望在非高峰时间发送这大块的流量，以便使用已经支付但没有被使用的带宽，并且愿意容忍一些延迟。它喜欢在晚上做备份，当其他应用程序没有大量使用网络时。问题在于对于全球性的服务，非高峰时间在世界各地不同的位置是不同的。波士顿和珀斯数据中心的非高峰期网络带宽在时间上很少有重叠，因为一个城市的夜晚是另一个城市的白天。

然而，DTN 模型允许在数据传输过程中进行存储和延迟。有了这个模型，就有可能使用非高峰期的带宽把数据集从波士顿发送到阿姆斯特丹，因为这两个城市的时差仅相隔 6 个小时的时区；然后将数据集存储在阿姆斯特丹，直到阿姆斯特丹和珀斯之间有非高峰期带宽可用；然后将数据集发送到珀斯，完成这次数据传送。（Laoutaris 等，2009）对该模型进行研究，并发现它可以较小的代价提供很大的容量；与传统的端到端模型相比，经常使用 DTN 模型通常能将容量增加一倍。

下面我们将描述 IETF DTN 体系结构和协议。

6.7.1 DTN 体系结构

Internet 的主要假设是在整个通信会话期间源端和接收方之间存在一条端到端路径，这正是 DTN 寻求释放的条件。当情况并非如此时，正常的 Internet 协议就会失败。DTN 能避开这种缺乏端到端连接这个问题，关键在于它采用了如图 6-56 所示的基于消息交换的体系结构。DTN 的目的还在于容忍低可靠性和大延迟的链路。DTN 的体系结构由 RFC 4838 说明。

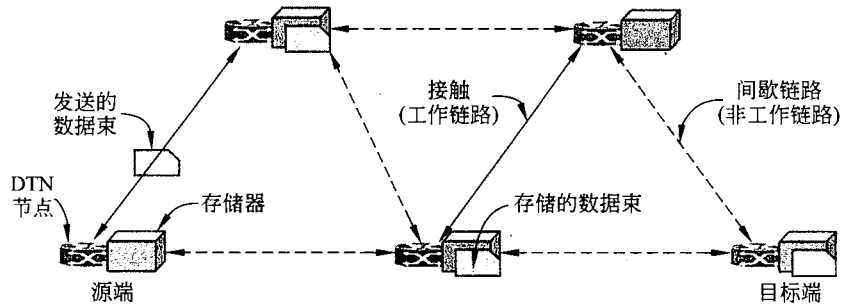


图 6-56 延迟容忍网络体系结构

按照 DTN 的术语，一条消息称为一个数据束（bundle）。DTN 节点都配备了存储器，典型的是持久性存储介质，比如磁盘或闪存。它们将数据束存储起来，直到链路变得可用，然后转发该束。链路的工作呈现间歇性特点。图 6-56 显示了 5 条当前都没有在工作的间歇性链路和两条正在工作的链路。一条工作链路称为一次接触（contact）。图 6-56 还显示了存储在两个 DTN 节点上的数据束，它们正在等待接触的到来以便向前发送。以这样的方式，通过一次次接触数据束从源被中继到它们的目的地。

在 DTN 节点存储和转发包，听起来类似于路由器上的排队和转发数据包，但两者之间有质的区别。在 Internet 的路由器上，排队发生以毫秒计，或最多以秒计。而在 DTN 节点，数据束可存储好几个小时，直到公交车到达小镇、飞机完成一次飞行、一个传感器节点吸收足够的太阳能开始运行、处于睡眠状态的计算机被唤醒等。这些例子还指出了 DTN 节点存储的第二个区别，那就是存储了数据的节点可能会移动（比如公交车或飞机），而且这种运动甚至是数据传递的关键部分。Internet 上的路由器不允许移动。移动数据束的整个过程可能更好地称为“存储-携带-转发”。

作为一个例子，考虑如图 6-57 所示的场景，这是在空中第一次使用 DTN 协议（Wood 等，2008）。数据束源是一种 LEO（低地球轨道卫星），它正在记录地球图像，作为灾害监测星座卫星的一部分。记录下来的图像必须返回给采集点。然而，卫星在绕地球旋转时只能间歇性地接触三个地面站。它依次与每个地面站进入接触。卫星、地面站和收集点每个都作为一个 DTN 节点。在每一次接触时，一个数据束（或一束的一部分）被发送到地面站；然后经过骨干陆地网络，数据束被发送到采集点，由此完成地球图像的传送。

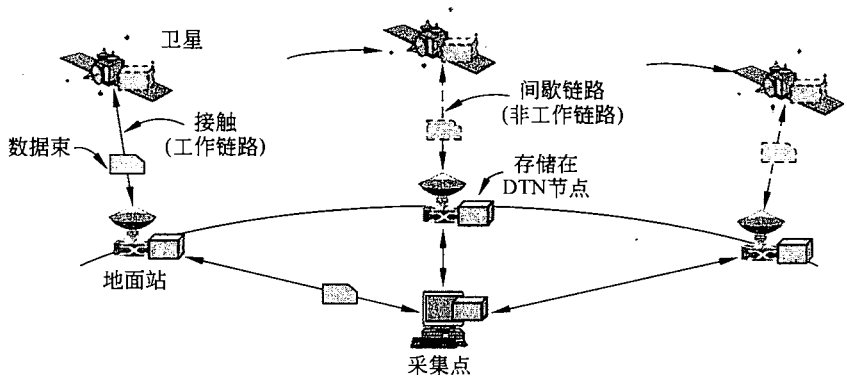


图 6-57 DTN 在空中的应用

在这个例子中，DTN 体系结构的主要优点是它自然而然适应了卫星需要存储图像的情形，因为在图像拍摄时根本没有连接。除此之外，在这种应用场景中该体系还有另外两个优点。首先，恐怕不存在一个单一的接触，持续时间足够长到发送完整个图像。然而，它们可以分散到与三个地面站的接触上。其次，使用卫星和地面站的接触可以将链路与骨干网络彻底脱钩。这意味着卫星的下载不会受到慢速地面链路的限制。它可以全速率地处理，数据束存储在地面站上，直到它可以中继给采集点。

这个体系结构没有说明的一个重要问题是如何寻找一条穿过 DTN 节点的良好路径。良好的路由取决于描述何时发送数据的体系结构性质，也取决于采用哪次接触。有些接触在使用之前就已经知道。一个很好的例子是在空间的天体运动。对于太空实验而言，早就知道何时会发生接触，每经过一个地面站的接触时间间隔为 5～14 分钟，下行链路的容量为 8.134 Mbps。有了这方面的知识，可以提前计划传输一束图像的路径。

在其他情况下，接触也可以预测，但是缺少确定性。这样的例子包括公交车，由于事先有时间表，通常公交车以最常规的方式与每个车站接触；类似地，虽然有些变化，ISP 网络的非高峰时间和带宽也可以从过去的数据中预测出来。另一种极端情况是接触完全是偶然和随机的。一个例子是用户通过移动电话传递数据，那么这种接触就取决于在一天之中哪些用户彼此接触。当存在着不可预测的接触时，可采用这样的路由策略：沿着不同的路径发送数据束的副本，希望在束的生存期到达之前其中的一个副本能被传递到目的地。

6.7.2 数据束协议

为了更清楚地了解 DTN 操作，我们现在来考查 IETF 协议。DTN 是一种新兴的网络，因而没有使用 IETF 协议的需求，所有实验性质的 DTN 都使用了不同的协议。然而，这些协议至少是我们开始和突出许多关键问题的良好的开端。

DTN 协议栈如图 6-58 所示。关键的协议是**数据束协议**（Bundle protocol），该协议由 RFC 5050 描述。它负责接受来自应用程序的消息，并通过“存储-携带-转发”操作将这些数据作为一个或者多个数据束发送到目标 DTN 节点。从图 6-58 中很明显地看出，数据束协议运行在 TCP/IP 层之上。换句话说，TCP/IP 可以被用在 DTN 节点之间的每个接触上，把数据束从接触一端的 DTN 节点移动到另一端的 DTN 节点。这种协议定位提出了数据束协议究竟是一个传输层协议还是应用层协议这样一个问题。正如 RTP 一样，我们采取这样的立场，尽管它运行在传输协议之上，由于束协议为许多不同的应用提供了传输服务，所以我们在本章覆盖 DTN 网络。

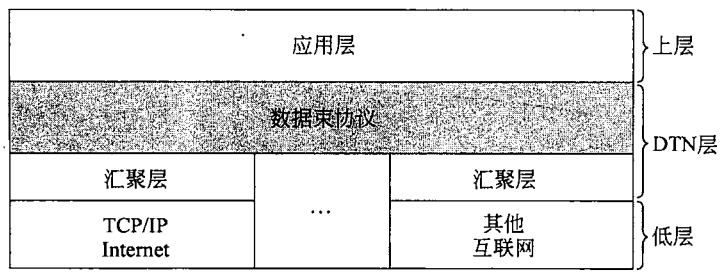


图 6-58 延迟容忍网络协议栈

在图 6-58 中我们看到，数据束协议可能运行在其他协议之上，比如 UDP，或者甚至其他种类的互联网协议之上。例如，在一个空间网络中链路可能有很长的延迟。地球和火星之间的往返时间很轻易地达到 20 分钟，这取决于行星的相对位置。可以设想一下，在这样的链路上 TCP 确认和重传将如何工作，尤其是针对相对短的消息。它根本无法工作。相反，另一种使用纠错码的协议可能会有作用。或者在资源受到非常约束的传感器网络中，可以使用一个比 TCP 更轻量级的协议。

由于数据束协议是固定的，但它主要运行在各种传输协议之上，因此在功能上协议之间必定留有沟壑。这种沟壑正是在图 6-58 中引入汇聚层的原因。汇聚层就像是匹配所连协议接口的粘胶层。根据定义，针对每个不同的底层传输协议，有一个不同的汇聚层。汇聚层通常是在标准中加入新的和现有协议。

数据束协议的消息格式如图 6-59 所示。这些消息中的不同字段告诉我们一些由数据束协议处理的关键问题。

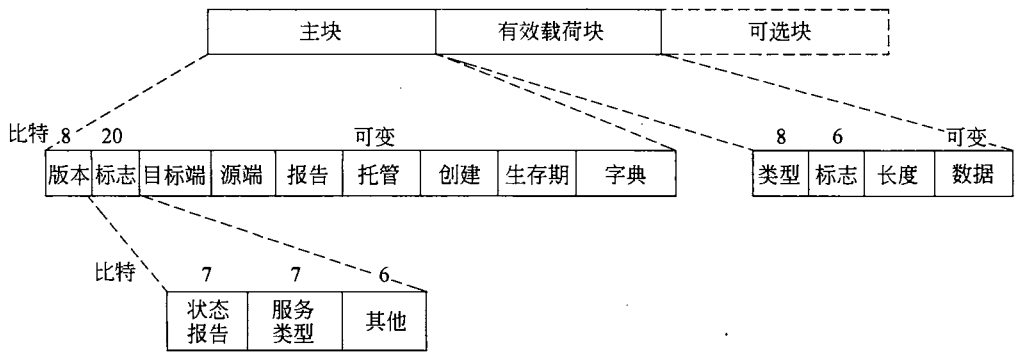


图 6-59 数据束协议的消息格式

每个消息由一个主块（Primary block）、一个有效载荷块（Payload block）和一个可选块（Optional block）组成。主块可以被看作是头，有效载荷包含着数据，而其他块可以携带一些安全参数。主块始于版本（Version）字段（当前为 6），紧接着一个标志（Flags）字段。在其他功能中，标志位编码了一类服务，让源端把它的数据束标记成更高或更低的优先级，以及诸如接收方是否应该确认该数据束等其他处理请求。

然后是地址字段，其中突出了 3 个有趣的设计部分。除了目标端（Destination）和源端（Source）标识符字段外，还有一个托管（Custodian）标识符。托管是负责照看包交付的当事人。在 Internet 中源节点通常就是托管人，因为如果数据没有最终交付到目的地，那么它就是重发的节点。然而，在 DTN 中，源节点可能并不总能连接上，而且可能没有办法知道这些数据是否已经传递到目的地。DTN 使用了托管传送（custody transfer）来处理这个问题。在托管传送方式中，另一个接近接收方的节点假设承担照看数据安全传递的责任。例如，如果一个包裹被装在飞机上，在稍后时间和地点被转发，那么飞机就成为包裹的托管人。

第二个有趣的方面是这些标识符不是 IP 地址。由于数据束协议是为了能在跨越各种传输协议和互联网上工作，它定义了自己的标识符，这些标识符实际上比诸如 IP 地址这样的底层地址更像个高层名称，如网页的 URL。它们给 DTN 提供了应用程序级的路由，比如

电子邮件传送或软件更新分发。

第三个有趣的方面是标识符的编码方式。还有一个报告（Report）标识符用于诊断消息。所有的标识符被编码为一个指向可变长度字典（Dictionary）字段的引用。当托管人或报告节点与源或目标节点相同时，这种方式就提供了压缩。事实上，消息的格式设计牢记着可扩展性和效率，因而使用了可变长度字段的压缩表示法。压缩表示法对于无线链路和资源受限的节点（比如传感器网络）至关重要。

接下来是一个创建（Creation）字段，该字段携带数据束的创建时间、从远端开始排序的序号以及生存期（Lifetime）字段，生存期给出了该束数据的有效时间。这些字段必须给出，因为数据可能被存储在DTN节点上相当一段时间，必须有某种方式从网络中删除过时的数据。与Internet不同的是，它们需要DTN节点有松散同步的时钟。

主块的结尾是字典（Dictionary）字段；接着是有效载荷块，该块由一个简短的类型（Type）字段开始，标识了它的有效载荷；接着是一些标志（Flag）字段，描述了如何处理该数据束的选项；然后是数据（Data）字段，之前是长度（Length）字段。最后，还有可能有其他可选块，比如携带安全参数的块。

有关DTN的许多方面正处于学术研究界的探索之中。良好的路由策略取决于接触的性质，正如上面所述的那样。存储在网络内部的数据提出了其他问题。现在拥塞控制必须考虑节点上的存储器，因为存储器变成了另一种可耗尽的资源。缺乏端到端的通信同时加剧了安全问题。在DTN节点成为数据束的托管人之前，它要知道发送端是否授权使用网络，并且数据束是否是接收方想要的。这些问题的解决方案将取决于DTN的类型，因为空间网络与传感器网络是完全不同的两种网络。

6.8 本章总结

传输层是理解分层协议的关键。它提供了各种服务，其中最重要的服务是一个从发送端至接收端的端到端的、可靠的、面向连接的字节流。通过一组服务原语可以访问此服务，原语可用来建立、使用和释放连接。Berkeley套接字提供了一个常用的传输层接口。传输协议必须能在不可靠的网络上完成连接管理工作。由于延迟的重复数据包可能会重新出现在不恰当的时刻，因而连接的建立过程非常复杂。为了处理这些问题，需要使用三次握手法来建立连接。释放连接比建立连接要容易得多，但由于存在两军对垒问题，释放连接也并非轻而易举。

即使网络层完全可靠，传输层也有大量的工作要做。它必须处理所有的服务原语、管理连接和计时器、拥塞控制与分配带宽以及运行大小可变的滑动窗口流量控制。

拥塞控制应该在竞争流之间公平地分配所有可用带宽，并跟踪网络使用的变化。AIMD控制法则能收敛到公平和有效的带宽分配。

Internet有两个主要的传输协议：UDP和TCP。UDP是一个无连接的协议，它主要对IP数据包进行了包装，由此引入了多个进程复用和分用一个IP地址这一特性。UDP可以被用于客户机-服务器之间的交互（比如RPC），UDP也可以被用来建立实时协议（比如RTP）。

主要的Internet传输协议是TCP。它提供了一个可靠的、双向的、拥塞可控的字节流，

构成字节流的所有段都有一个 20 个字节长的头。大量的工作针对 TCP 性能进行了优化，它们主要是 Nagle、Clark、Jacobson、Karn 和其他的算法。

网络性能通常由协议和段的处理开销来主宰，在速度很高的时候这种状况变得更糟。协议应该被设计成最小化段的数量，而且在带宽延迟乘积大的路径上工作良好。对于千兆网络，要求使用尽可能简单的协议，并实行流水线式处理。

延迟容忍网络提供了一个跨网络的传递服务，这种网络具有偶尔的连通性或链路的延迟很长。中间节点存储、携带并转发捆绑在一起的数据束，因此最终得以传递到目的地，即使在任何时间都不存在一条从发送端到接收端的工作路径。

习 题

1. 在我们的图 6-2 所示传输原语例子中，LISTEN 是一个阻塞调用。试问这是严格要求的吗？如果不是，请解释如何使用一个非阻塞的原语。与正文中描述的方案相比，你的方案有什么优点？
2. 传输服务原语假设在两个端点之间建立连接的过程是不对称的，一端（服务器）执行 LISTEN，而另一端（客户端）执行 CONNECT。然而，在对等应用中，比如 BitTorrent 那样的文件共享系统，所有的端点都是对等的。没有服务器或客户端功能之分。试问如何使用传输服务原语来构建这样的对等应用？
3. 在图 6-4 所示的底层模型中，它的假设条件是网络层的数据包有可能被丢失，因此，数据包必须被单独确认。假如网络层百分之百可靠，并且永远不会丢失数据包，试问图 6-4 需要做什么修改吗？如果需要的话。
4. 在图 6-6 的两部分中，有一条注释说明了 SERVER_PORT 在客户机和服务器中必须相同。试问为什么这一条如此重要？
5. 在 Internet 文件服务器例子中（图 6-6），除了服务器端的监听队列为满之外，试问还有其他原因能导致客户机的 connect（）系统调用失败吗？假设网络完美无缺。
6. 评判一个服务器是否全程活跃，或者通过进程服务器来按需启动它的一个标准是它所提供服务的使用频率。试问你能想出作出这一决定的任何其他标准吗？
7. 假设采用时钟驱动方案来生成初始序号，该方案用到了一个 15 位的时钟计数器。每隔 100 毫秒时钟滴答一次，最大数据包生存期为 60 秒。试问，每隔多久需要重新同步一次？
   （a）在最差情况下。
   （b）当数据每分钟用掉 240 个序号的时候。
8. 试问，为什么最大数据包生存期 T 必须足够大，大到确保不仅数据包本身而且它的确认也消失在网络中？
9. 想象用两次握手过程而不是三次握手过程来建立连接。换句话说，第三个消息不再是要求的。试问现在有可能死锁吗？请给出一个例子说明存在死锁，或者证明死锁不存在。
10. 想象一个广义的 n-军队对垒问题，在这里任何两支蓝军达成一致意见后就足以取得胜利。试问是否存在一个能保证蓝军必赢的协议？
11. 请考虑从主机崩溃中恢复的问题（图 6-18）。如果写操作和发送确认之间的间隔可以设

置得相对非常小，那么，为了使协议失败的概率最小，试问两种最佳的发送端-接收端策略是什么？

12. 在图6-20中，假设加入了一个新的流E，它的路径为从R1到R2，再到R6。试问对于5个流的最大-最小带宽分配有什么变化？
13. 请讨论信用协议与滑动窗口协议的优缺点。
14. 拥塞控制的公平性方面有一些其他的策略，它们是加法递增加法递减（AIAD，Additive Increase Additive Decrease）、乘法递增加法递减（MIAD，Multiplicative Increase Additive Decrease）、乘法递增乘法递减（MIMD，Multiplicative Increase Multiplicative Decrease）。请从收敛性和稳定性两个方面来讨论这三项政策。
15. 试问，为什么会存在UDP？用户进程使用原始IP数据包还不够吗？
16. 请考虑一个建立在UDP之上的简单应用层协议，它允许客户从一个远程服务器获取文件，而且该服务器位于一个知名地址上。客户端首先发送一个请求，该请求中包含了文件名；然后服务器以一个数据包序列作为响应，这些数据包中包含了客户所请求的文件的不同部分。为了确保可靠性和顺序递交，客户和服务器使用了停-等式协议。忽略显然存在的性能问题，试问你还能看得出这个协议存在的另一个问题吗？请仔细想一想进程崩溃的可能性。
17. 一个客户通过一条1 Gbps的光纤向100千米以外的服务器发送一个128字节的请求。试问在远过程调用中这条线路的效率是多少？
18. 继续考虑上一个问题的情形。对于给定的1 Gbps线路和1 Mbps线路两种情况，请计算最小的可能响应时间。试问由此你得出了什么结论？
19. UDP和TCP在传递消息时，都使用了端口号来标识接收方实体。请给出两个理由说明为什么这两个协议要发明一个新的抽象ID（端口号），而不用进程ID？在设计这两个协议的时候，进程ID早已经存在。
20. 一些RPC实现为客户端提供了一个选项，使用实现在UDP之上的RPC还是使用实现在TCP之上的RPC。试问在什么样的条件下，客户端更喜欢使用基于UDP的RPC？在什么条件下，他或许更喜欢使用基于TCP的RPC？
21. 考虑两个网络N1和N2，在源端A和目的端D之间有相同的平均延迟。在N1中，不同的数据包所经历的延迟是均匀分布，且最大延迟为10秒；而在N2中，99%的数据包所经历的延迟都小于1秒，且没有最大延迟上界。请讨论在这两种情况下如何用RTP来传输实时音频/视频流。
22. 试问最小TCP MTU的总长度是多少？包括TCP和IP的开销，但是不包括数据链路层的开销。
23. 数据报的分段和重组机制由IP来处理，对于TCP不可见。试问，这是否意味着TCP不用担心数据错序到达的问题？
24. RTP被用来传输CD品质的音频，这样的音频信号包含一对16位的采样值，采样的频率为每秒钟44 100次，每个采样值对应于一个立体声声道。试问RTP每秒钟必须传输多少个数据包？
25. 试问有可能将RTP代码放到操作系统内核中，与UDP代码放在一起吗？请解释你的答案。

26. 主机1上的一个进程已经被分配了端口p，主机2上的一个进程也已经被分配了端口q，试问这两个端口之间有可能同时存在两个或者多个TCP连接吗？
27. 在图6-36中，我们看到除了32位的确认号字段外，在第四个字还有一个ACK标志位。试问这个标志位有额外的意义吗？为什么有？或者为什么没有？
28. TCP段的最大有效载荷为65 495字节。试问为什么选择如此奇怪的数值？
29. 描述从图6-39中进入SYN RCVD状态的两种途径。
30. 请考虑在一条往返时间为10毫秒的无拥塞线路上使用慢速启动算法的效果。接收窗口为24 KB，最大段长为2 KB。试问需要多长时间才能首次发送满窗口的数据？
31. 假设TCP的拥塞窗口被设置为18 KB，并且发生了超时。如果接下来的4次突发传输全部成功，试问拥塞窗口将达到多大？假设最大段长为1 KB。
32. 如果TCP往返时间RTT的当前值是30毫秒，紧接着分别在26、32、24毫秒确认到达，那么，若使用Jacobson算法，试问新的RTT估计值为多少？请使用$\alpha = 0.9$。
33. 一台TCP机器正在通过一条1 Gbps的信道发送满窗口的65 535字节数据，该信道的单向延迟为10毫秒。试问可以达到的最大吞吐量是多少？线路的效率是多少？
34. 一台主机在一条线路上发送1500字节的TCP有效载荷，最大数据包生存期为120秒，要想不让序号回绕，试问该线路的最快速度为多少？要考虑TCP、IP和以太网的开销。假设可以连续发送以太网帧。
35. 为了解决IPv4的局限性，主要经过IETF的努力，导致了IPv6的设计，但仍然有很多人不愿意采用这个新版本。然而，却没有人做出解决TCP限制所需要的重大努力。请解释为什么会这样。
36. 在一个网络中，最大段长为128字节，段的最大生存期为30秒，序号为8位，试问每个连接的最大数据率是多少？
37. 假设你正在测量接收一个段所需要的时间。当发生一个中断时，你读出系统时钟的值（以毫秒为单位）。当该段被完全处理后，你再次读出时钟的值。你测量的结果是270 000次为0毫秒，730 000次为1毫秒。试问接收一个段需要多长时间？
38. 一个CPU执行指令的速率为1000 MIPS。数据复制可以按每次64位来进行，每个字的复制需要花费10条指令。如果一个入境数据包要被复制4次，试问这个系统能处理一条1 Gbps的线路吗？为了简单起见，假设所有的指令，包括读或者写内存的指令，都以1000 MIPS的全速率运行。
39. 为了避开当序号回绕时老的数据包仍然存在这个问题，可以使用64位序号。然而，从理论上讲，光纤的运行速度可以达到75 Tbps。试问在未来的75 Tbps网络中使用64位序号，数据包的最大生存期为多少才能确保不会发生回绕问题？假设每个字节都有自己的序号，像TCP那样。
40. 在6.6.5节中，我们计算了主机以80 000包/秒的速度往一条千兆线路发送，只使用了6250指令，留下一半的CPU时间给应用程序。这里计算时假设数据包大小为1500字节。针对ARPANET大小的数据包（128字节）重复上述计算。在这两种情况下，假设给出的数据包大小均包括了所有开销。
41. 对于一个运行在4000千米距离上的1 Gbps网络，限制因素是延迟而并非带宽。请考虑这样一个MAN，源端和接收方之间的平均距离为20千米。试问多大的速率使得1 KB

数据包的传输延迟等于光速的往返延迟？

42. 试计算下列网络的带宽-延迟乘积：（1）T1（1.5 Mbps）；（2）以太网（10 Mbps）；（3）T3（45 Mbps）和（4）STS-3（155 Mbps）。假设RTT为100毫秒。请回忆TCP头有16位保留用作窗口大小（Window Size）。根据你的计算，试问它有什么隐含的意义吗？

43. 对于地球同步卫星上的一条50 Mbps信道，试问它的带宽-延迟乘积是多少？如果所有的数据包都是1500字节（包括开销），试问窗口应该为多大（按数据包为单位）？

44. 图6-6所示的文件服务器远非完美，它在许多方面都还可以改进。请作以下修改：
（a）给客户增加第三个参数，它指定了一个字节范围。
（b）给客户增加一个标志w，允许将文件写入服务器上。

45. 所有的网络协议都需要的一种常见功能是处理消息。回想一下，协议处理消息通过添加/拆除头来实现的。有些协议还可能将一个单一消息分解成多个片段，在稍后再把这些多个片段组合成一个单一消息。为此，请设计并实施一个消息管理库。该库提供了创建一个新消息、附加一个消息头到消息上、从消息上剥离消息头、将一个消息分成两个消息、将两个消息组合成一个消息，以及保存消息副本功能。你的实现必须尽量减少把数据从一个缓冲区复制到另一个缓冲区的操作。至关重要的是处理消息的操作不应该触碰到消息的数据，而只处理消息指针。

46. 设计和实现一个聊天系统，它允许多组用户同时进行聊天活动。有一个聊天协调器位于某一个知名的网络地址上，它使用UDP与聊天客户通信，并且为每个聊天会话建立聊天服务器，而且还维护了一个聊天会话目录。每个聊天会话都有一个聊天服务器。聊天服务器使用TCP与客户端进行通信。聊天客户端允许用户启动、加入或者离开一个聊天会话。请设计和实现聊天协调器、聊天服务器和聊天客户端代码。

第7章 应 用 层

在完成了所有预备知识的学习后，我们现在来到应用层，这里可以找到所有的网络应用。应用层下面的各层提供了传输服务，但它们并不真正为用户工作。在本章中，我们将学习一些实际的网络应用。

然而，即使在应用层也仍然需要协议的支持，以便各种应用程序能够工作。因此，在开始介绍这些应用之前，我们将先介绍其中的一个协议。这个协议就是DNS，它负责处理Internet命名问题。之后，我们将介绍3个实际应用：电子邮件、万维网（World Wide Web）和多媒体。我们将以对内容分发应用的简单介绍来结束本章，包括对等网络的基本概念。

7.1 DNS——域名系统

虽然在理论上，所有程序通过使用它们存储的计算机网络地址（例如IP），就可以访问Web主页、邮箱和其他资源，但是这些地址很难让人记住。而且浏览一个公司在128.111.24.41上的Web主页，意味着如果该公司将主页移到了另一台机器上，且该机器具有了不同的IP地址时，那么必须将该机器的IP地址通知到每个人。因此，人们引入了可读性好的高层名字，以便将机器名字与机器地址分离开。在这种方式下，无论真正实用的IP地址是什么，人们熟知的公司Web服务器为www.cs.washington.edu。然而，因为网络本身只能理解数字形式的地址，所以需要某种机制将名字转换成网络地址。下面，我们学习如何在Internet上完成这种从名字到地址的映射。

我们回到早期的ARPANET，那时只有一个简单的文件，名叫hosts.txt，它列出了所有的计算机名字和它们的IP地址。每天晚上，所有主机都从一个维护此文件的站点将该文件取回，然后在本地进行更新。对于一个拥有几百台大型分时机器的网络而言，这种方法工作得相当好。

然而，当几百万台PC连接到Internet以后，使用网络的每个人都意识到这种方法将不能继续有效工作了。一方面，这个文件变得非常庞大。然而，更重要的是，除非集中管理机器名字，否则主机名冲突的现象将会频繁发生；而且在一个巨大的国际性网络中，由于负载和延迟，要实现这种集中式的管理简直难以想象。为了解决这些问题，1983年人们发明了**域名系统**（DNS，Domain Name System）。自发明以来，它一直是Internet的关键组成部分。

DNS的本质是发明了一种层次的、基于域的命名方案，并且用一个分布式数据库系统加以实现。DNS的主要用途是将主机名映射成IP地址，但它也可以用于其他用途。RFC 1034、1035、2181给出了DNS的定义，后来其他文档对它又做了进一步的阐述。

简要地说，DNS的使用方法如下所述。为了将一个名字映射成IP地址，应用程序调用一个名为**解析器**（resolver）的库程序，并将名字作为参数传递给此程序。在图6-6中我

们看到的 gethostbyname 就是一个解析器例子。解析器向本地 DNS 服务器发送一个包含该名字的请求报文；本地 DNS 服务器查询该名字，并且返回一个包含该名字对应 IP 地址的响应报文给解析器，然后解析器再将 IP 地址返回给调用方。查询报文和响应报文都作为 UDP 数据包发送。有了 IP 地址以后，应用程序就可以与目标主机建立一个 TCP 连接，或者给它发送 UDP 数据包。

7.1.1 DNS 名字空间

管理一个大型并且经常变化的名字集合不是个简单的问题。在邮政系统中，名字管理要求所有的信件必须（隐式地或显式地）指定国家、州或省、城市、街道地址以及收件人的名字。通过这种地址的层次结构，纽约州 White Plains 市 Main 大街上的 Marvin Anderson 与德克萨斯州奥斯汀市 Main 大街上的 Marvin Anderson 就不会产生混淆。DNS 采用了同样的工作方式。

对于 Internet，命名层次结构的顶级由一个专门组织负责管理。该组织名为 Internet **名字与数字地址分配机构**（ICANN，Internet Corporation for Assigned Names and Numbers），创建于 1998 年，它是 Internet 成长为全球性并且受到经济关注的部分成熟标志。从概念上讲，Internet 被划分为超过 250 个**顶级域名**（top-level domains），其中每个域涵盖了许多主机。这些域又被进一步划分成子域，这些子域可被再次划分，依此类推。所有这些域可以表示为一棵树，如图 7-1 所示。树的叶子代表没有子域的域（当然要包含机器）。一个叶结点域可能包含一台主机，或者代表一个公司，该公司包含数以千计的主机。

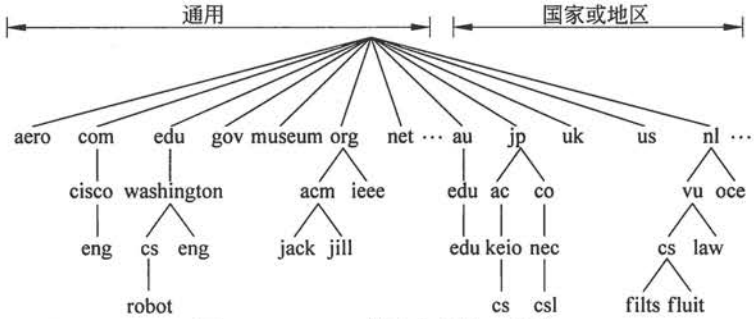


图 7-1 Internet 域名空间的一部分

顶级域名分为两种类型：通用的和国家或地区的。在图中 7-2 中列出的通用域名包括从 20 世纪 80 年代开始的原始域名和向 ICANN 申请引入的新域名。将来或许会增加其他的通用顶级域名。

国家或地区域名包括每个国家或地区，国家或地区的域名由 ISO 3166 文档定义。2010 年推出了使用非拉丁字母的国际化国家或地区域名。这些域名使得说阿拉伯语、西里尔文、中文或其他语言国家的人以自己的母语来命名他们的主机。

获得一个二级域名容易得多，比如 name-of-company.com。顶级域名由 ICANN 委任的**注册机构**（registrar）负责运行。要获得一个域名，只要到相应的注册机构（在这种情况下是 com），检查所需的名字是否可用，并且不是别人的商标。如果没有问题，请求注册域名的一方向管理域名的注册方支付一小笔年费，即可得到心仪的域名。

然而，随着 Internet 变得越来越商业化和越来越国际化，它也变得更具争议性，特别是在有关域名的事务上。这样的争议包括 ICANN 本身。例如，×××域名创造了好几年，法院已经审理要解决它。自愿将成人内容放在自己的域名空间中到底是好事还是坏事（有些人不希望成人内容对 Internet 上的所有人都开放，而其他人则想把成人内容放在一个域名中，以便让保姆过滤器很容易地找到并阻止儿童看到）？有些域名可以自我组织，而另一些则对谁可以得到一个该域名空间中的名字有限制，正如图 7-2 中所指出的那样。但是，什么样的限制才是适当的呢？我们用 pro 域名作为例子说明。这是一个分配给合格的专业人员所用的名字。但问题是谁才是专业的呢？显然医生和律师是专业人士，但自由摄影师、钢琴教师、魔术师、管道工、理发师、灭虫者、纹身艺术家和雇佣军等算专业人士吗？这些职业有资格当选吗？按照谁的标准呢？

域	预定使用	启用时间	限制否
com	商业	1985	不限
edu	教育科研	1985	限
gov	政府	1985	限
int	国际组织	1988	限
mil	军事	1985	限
net	网络提供商	1985	不限
org	非营利组织	1985	不限
aero	航空运输	2001	限
biz	公司	2001	不限
coop	合作	2001	限
info	信息	2002	不限
museum	博物馆	2002	限
name	人	2002	不限
pro	专业	2002	限
cat	加泰罗尼亚	2005	限
jobs	就业	2005	限
mobi	移动设备	2005	限
tel	联络资料	2005	限
travel	旅游业	2005	限
×××	色情业	2010	不限

图 7-2 通用的顶级域名

域名还与金钱挂钩。图瓦卢（国）将它的域名 tv 出售，获得租赁权 50 万美元，就是因为这个国家的域名非常适合广告电视网站。实际上，com 域名下几乎采用了每一个普通词（英文），而且这些词有最常见的拼写错误。家居用品、动物、植物以及身体各部位等名字都被人尝试注册了域名。注册所有人一转身就以高得多的价格把域名出售给感兴趣的人，而所有的一切都只是一个名字而已。这就是所谓的**域名抢注**（cybersquatting）。当 Internet 时代来临时，许多公司起跑得比较缓慢，以至于突然发现显而易见应该是自己的域名已经被他人注册了，于是它们试图收购这样的名字。在一般情况下，只要商标没有受到侵犯，而且没有涉及欺诈，那么域名的授予是先到先得的。然而，用以解决域名纠纷的政策仍在不断细化之中。

每个域的名字是由它向上到（未命名的）根节点的路径来命名的，路径上的各个部分用句点（读作“dot”）分开。因此，Cisco 公司的工程部门可能是 eng.cisco.com.，而不是像

UNIX 风格的名字/com/cisco/eng。请注意，这种层次的命名机制意味着 eng.cisco.com.中的 eng 不会与 eng.washington.edu.中的 eng 发生冲突，华盛顿大学的英语系可能会取它作为自己的域名。

域名可以是绝对的，也可以是相对的。绝对域名总是以句点作为结束（例如 eng.cisco.com.），而相对域名则不然。相对域名必须在一定的上下文环境中被解释才有的真正含义。无论是绝对域名还是相对域名，一个域名对应于域名树中一个特定的结点，以及它下面的所有结点。

域名不区分大小写，因此，edu、Edu 和 EDU 的含义都一样。各组成部分的名字最多可以有 63 个字符，整个路径的名字不得超过 255 个字符。

原则上，域名可以被插入到域名树中的通用域名空间或者国家域名空间。例如，cs.washington.edu 完全可以被列在国家域名 us 的下面，从而变成 cs.washington.wa.us。然而，实际上美国的大多数组织都位于某一个通用域名下面，而美国之外的大多数组织则位于其国家域名的下面。没有规则不允许一个组织在多个顶级域下注册域名。大型公司通常就这么做（例如 sony.com 和 sony.nl）。

每个域自己控制如何分配它下面的子域。例如，日本的 ac.jp 和 co.jp 域分别对应于 edu 和 com；但荷兰不做这样的区分，它把所有的组织直接放在 nl 下。因此，以下 3 个域名都表示大学的计算机科学系：

（1）cs.washington.edu（美国华盛顿大学）。

（2）cs.vu.nl（荷兰阿姆斯特丹自由大学）。

（3）cs.keio.ac.jp（日本庆应义塾大学）。

为了创建一个新域，创建者必须得到包含该新域的上级域的许可。例如，如果华盛顿大学建立了一个 VLSI（超大规模集成电路）组，并且希望将它命名为 vlsi.cs.washington.edu，那么这个组必须获得管理 cs.washington.edu 域名的管理员许可。类似地，如果创立了一所新的大学，比如说 University of Northern South Dakota，那么它必须请求负责 edu 域名的管理员将 unsd.edu 分配给它。按照这种方式分配域名就可以避免名字冲突，并且每个域都可以跟踪它所有的子域。一旦创建并注册了一个新域，则该新域就可以创建属于自己的子域，比如 cs.unsd.edu，而无须得到域名树中任何上层域的许可。

命名机制遵循的是以组织为边界，而不是以物理网络为边界。例如，即使计算机科学系和电子工程系在同一幢楼里，并使用同一个 LAN，但它们仍然可以属于完全不同的域。同样地，即使计算机科学系分散在 Babbage Hall 和 Turing Hall 这两幢楼里，但这两幢楼里的主机通常仍属于同一个域。

7.1.2 域名资源记录

无论是只有一台主机的域还是顶级域，每个域都有一组与它相关联的**资源记录**（resource record）。这些记录组成了 DNS 数据库。对于一台主机来说，最常见的资源记录就是它的 IP 地址，但除此以外还存在着许多其他种类的资源记录。当解析器把一个域名传递给 DNS 时，它能获得的 DNS 返回结果就是与该域名相关联的资源记录。因此，DNS 的基本功能是将域名映射到资源记录。

一条资源记录是一个五元组。尽管出于效率考虑资源记录被编码成了二进制的形式，但是大多数说明性资料还是用 ASCII 文本来表示资源记录，每一条记录占一行。在接下来的介绍中，我们将使用如下的格式：

```
Domain_name Time_to_live  Class    Type    Value
```

Domain_name（域名）指出了这条记录适用于哪个域。通常每个域有许多条记录，并且数据库的每份副本保存了多个域的信息。因此 Domain_name 是匹配查询条件的主要搜索关键字。数据库中资源记录的顺序则无关紧要。

Time_to_live（生存期）指明了该条记录的稳定程度。极为稳定的信息会被分配一个很大的值，比如 86 400（1 天时间里的秒数）；而非常不稳定的信息则会被分配一个较小的值，比如 60（1 分钟的秒数）。稍后当我们讨论缓存机制时将再回到这个议题。

每条资源记录的第三个字段是 Class（类别）。对于 Internet 信息，它总是 IN。对于非 Internet 信息，则可以使用其他的代码，但实际上很少见。

Type（类型）字段指出了这是什么类型的记录。DNS 记录有许多类型，图 7-3 列出了最重要的一些类型。

类型	含义	值
SOA	授权开始	本区域的参数
A	主机的IPv4地址	32位整数
AAAA	主机的IPv6地址	128位整数
MX	邮件交换	优先级，愿意接受邮件的域
NS	域名服务器	本域的服务器名字
CNAME	规范名	域名
PTR	指针	IP地址的别名
SPF	发送者的政策框架	邮件发送政策的文本编码
SRV	服务	提供服务的主机
TXT	文本	说明的ASCII文本

图 7-3 主要的 DNS 资源记录类型

SOA 记录给出了有关该名字服务器区域（后面将会介绍）的主要信息源名称、名字服务器管理员的电子邮件地址、一个唯一的序号以及各种标志位和超时值。

最重要的记录类型是 A（地址）记录，它包含了某台主机一个网络接口的 32 位 IP 地址。对应的 AAAA，或“quad A”记录包含了一个 128 位的 IPv6 地址。每台 Internet 主机必须至少有一个 IP 地址，以便其他机器能与它进行通信。某些主机有两个或多个网络接口，在这种情况下，它们就有两个或多个 A 或 AAAA 资源记录。因此，对于单个域名的查询可能获得多个地址。

最常用的记录类型是 MX 记录，它指定了一台准备接受该特定域名电子邮件的主机的名字。因为并非每台机器都做好了接收电子邮件的准备，因此必须用一个记录作特别说明。例如，如果有人打算发送电子邮件给某个人，比如 bill@microsoft.com，则发送主机必须找到在 microsoft.com 中愿意接收电子邮件的邮件服务器。MX 记录就是用来提供这样的信息。

另一个重要记录类型是 NS 记录。它指明了一台用于所在域和子域的名字服务器。这是一台拥有一份某域数据库副本的主机。这条记录被用于域名查询处理。稍后我们将对此

做简单的讨论。

CNAME 记录允许创建别名。例如，如果一个人很熟悉 Internet 的常规命名规则，他打算给 MIT 计算机科学系名叫 paul 的人发送邮件，他就猜测此人的邮箱名是 paul@cs.mit.edu。实际上，这个地址不正确，因为 MIT 计算机科学系的域名是 csail.mit.edu。但是，MIT 可以创建一条 CNAME 记录指向人和程序的正确方向，为那些不知情的人提供一项服务。类似下面这样的一条记录就可以完成此任务：

```
cs.mit.edu  86400  IN    CNAME    csail.mit.edu
```

如同 CNAME 一样，PTR 指向另一个名字。但是 CNAME 只是一个宏定义（即可以用另一个串来替代一个串的机制），而 PTR 与 CNAME 不同，它是一种正规的 DNS 数据类型，它的确切含义取决于所在的上下文。实际上，PTR 几乎总是被用来将一个名字与一个 IP 地址关联起来，以便能够通过查询 IP 地址来获得对应机器的名字，这种功能称为逆向查询（reverse lookups）。

SRV 是一个新的记录类型，它把主机标识为域内的一种给定服务。例如，cs.washington.edu 的 Web 服务器可以标识成 cockatoo.cs.washington.edu。这个记录能产生 MX 记录，并执行相同的任务，但只适用于邮件服务器。

SPF 也是一个新的记录类型。它可以让一个域把邮件服务信息进行编码，即域内哪台机器负责把邮件发送到 Internet 其余地方。这条记录有助于接收机器检查邮件是否有效。如果接收到的邮件来自一台通话本身躲躲闪闪的机器，但域名记录说明邮件只能来自域外一台称为 smtp 的机器，那么该邮件就是伪造的垃圾邮件。

列表的最后一行给出了 TXT 记录，这是最初为了允许域以任意方式标识自己而提供的一种途径。如今，它们通常被编码成机器可读信息，一般是 SPF 信息。

最后，资源记录还包括 Value 字段，该字段的值可以是一个数字、一个域名或者一个 ASCII 字符串，其语义取决于记录的类型。图 7-3 中给出了每种主要记录类型的 Value 字段的简短描述。

图 7-4 显示的例子给出了在一个域的 DNS 数据库中可能找到的信息种类。它描述了图 7-1 中 cs.vu.nl 域（假设的）数据库的一部分。该数据库包含 7 种资源记录。

图 7-4 中第一个非注释行给出了该域的一些基本信息，我们以后将不再关心这些信息。接下来的两个表项给出了第一个和第二个投递电子邮件给 person@cs.vu.nl 的地点。首先尝试的是 zephyr（一台特定的机器），如果给它发送失败了，则下一个应该尝试给 top 发送。下一行标识了该域的名字服务器为 star。

接下来是一个空白行，加入空白行的目的是为了增加可读性。再下面的几行给出了 star、zephyr 和 top 的 IP 地址。紧跟这些行后面的是别名 www.cs.vu.nl，因此可以使用这个地址而无须指派一台特殊的机器。创建这个别名的好处是允许 cs.vu.nl 改变它的 WWW 服务器，无须使得人们原来所使用的地址失效。类似地，ftp.cs.vu.nl 也是一个别名。

有关机器 flits 的信息部分列出了两个 IP 地址和处理发送至 flits.cs.vu.nl 电子邮件的三种选择。首先选择的是 flits 本身，但是如果它失效，zephyr 和 top 是第二个和第三个选择。

; cs.vu.nl的授权数据				
cs.vu.nl	86 400	IN	SOA	star boss (9527, 7200, 7200, 241 920, 86 400)
cs.vu.nl.	86 400	IN	MX	1 zephyr
cs.vu.nl.	86 400	IN	MX	2 top
cs.vu.nl.	86 400	IN	NS	star
star	86 400	IN	A	130.27.56.205
zephyr	86 400	IN	A	130.37.20.10
top	86 400	IN	A	130.37.20.11
www	86 400	IN	CNAME	star.cs.vu.nl
ftp	86 400	IN	CNAME	zephyr.cs.vu.nl
flits	86 400	IN	A	130.37.16.112
flits	86 400	IN	A	192.31.231.165
flits	86 400	IN	MX	1 tlits
flits	86 400	IN	MX	2 zepnyr
flits	86 400	IN	MX	3 top
rowboat		IN	A	130.37.56.201
		IN	MX	1 rowboat
		IN	MX	2 zephyr
little-sister		IN	A	130.37.62.23
laserjet		IN	A	192.31.231.216

图 7-4 针对 cs.vu.nl 域名可能的 DNS 数据库一部分

接下来的 3 行包含了一台计算机的典型表项，在这个例子中是 rowboat.cs.vu.nl。它提供的信息包括 IP 地址、主要邮件存放处和次要邮件存放处。 然后是一个针对一台计算机的表项，这台计算机自己不能接收邮件；紧随其后的一个表项可能是指一台连接到 Internet 的打印机。

7.1.3 域名服务器

至少在理论上，一台域名服务器就可以包含整个 DNS 数据库，并响应所有对该数据库的查询。但实际上，这台服务器会因负载过重而变得毫无用处。而且，一旦它停机，则整个 Internet 将会瘫痪。

为了避免由于单个信息源带来的各种问题，DNS 名字空间被划分为一些不重叠的区域（zones）。针对图 7-1 的名字空间，一种可能的划分方法如图 7-5 所示。每个圈起来的区域包含域名树的一部分。

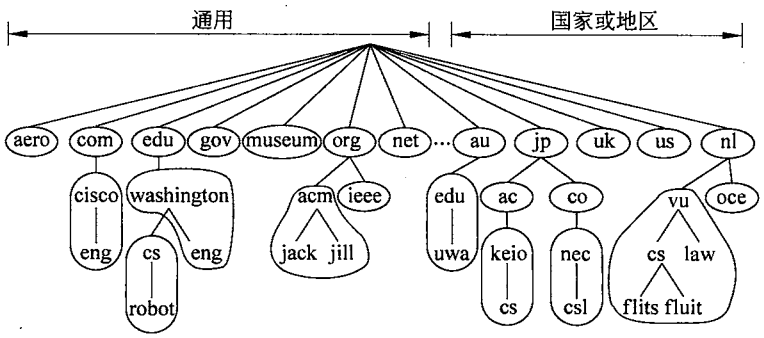


图 7-5 按区域划分（圈起来的）的部分 DNS 名字空间

区域边界应该放置在区域中的什么位置由该区域的管理员来决定。这个决定在很大程度上取决于需要在哪里使用多少个名字服务器。例如，在图 7-4 中，华盛顿大学有一个区

域 washington.edu，它要处理 eng.washington.edu 但不能处理 cs.washington.edu，这是另一个区域，有它自己的域名服务器。当有些系（比如英语系）不希望运行自己的域名服务器，而另外一些系（比如计算机科学系）却希望运行自己的域名服务器时，就有可能作出这样的决定。

每个区域都与一个或多个域名服务器关联。这些服务器是持有该区域数据库的主机。通常情况下，一个区域有一个主域名服务器和一个或多个辅域名服务器。主服务器从自己磁盘的一个文件读入有关域名信息，辅域名服务器从主域名服务器获取域名信息。为了提高可靠性，一些域名服务器可以设置在区域外面。

查询一个名字和找出其对应地址的过程称为域名解析（name resolution）。当解析器需要查询一个域名，它就把该查询传递给一个本地域名服务器。如果需要寻找的域恰好落在该域名服务器管辖下，比如 top.cs.vu.nl 在 cs.vu.nl 的管辖下，则该域名服务器就返回权威资源记录。一个**权威记录**（authoritative record）由管理该记录的权威部门提供，因此总是正确的。权威记录的权威性是相对**缓存记录**（cached record）而言的，缓存的记录有可能过时了。

如果被查询域在远端，比如 flits.cs.vu.nl 要找到华盛顿大学（UW）robot.cs.washington.edu 的 IP 地址，会发生什么？在这种情况下，如果本地没有关于相关域的缓存信息，那么域名服务器启动一次远程查询。该查询遵循如图 7-6 所示的过程。第 1 步，显示查询报文被发送到本地域名服务器。查询中包含了被查询的域名、类型（A）和类（IN）。

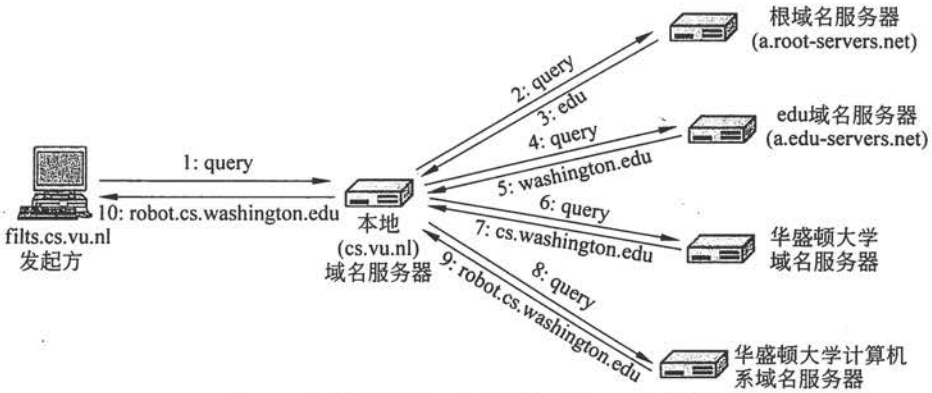


图 7-6 解析器查询一个远程名字的 10 个步骤

下一步是通过请求其中之一的根域名服务器来启动域名层次结构顶部的查询。这些域名服务器包含每个顶级域名的有关信息。这显示在图 7-6 中的第 2 步。为了与一个根服务器取得联系，每个域名服务器必须有一个或多个根域名服务器的信息。这个信息通常放在一个系统配置文件中，在 DNS 服务器启动时把该文件加载到 DNS 缓存。这个文件很简单，只是列出了关于根服务器的 NS 记录和相应的 A 记录。

总共有 13 个根 DNS 服务器，毫无创意地被命名为从 a-root-servers.net 到 m.root-servers.net。每个根服务器可以是逻辑上的一台计算机。然而，由于整个 Internet 依赖于根服务器，因此它们必须是能力超强的，并且被大量复制的计算机。大多数服务器被放置在多个地理位置，查询报文通过选播路由到达其中一台服务器。选播是一种特殊的路由，它能将数据包路由到最近的一个目标地址实例。我们在第 5 章中描述了选播路由。复制技术

能提高域名系统的可靠性和性能。

根域名服务器不可能知道在UW的一台机器的地址，可能还不知道UW的域名服务器。但是，它必须知道edu域的域名服务器，因为cs.washington.edu属于edu域管辖下。在第3步，它返回查询的答案，其中包括了名字和IP地址。

然后本地域名服务器继续尽职地执行任务。它将整个查询发给edu域名服务器（a.edu-servers.net）。该域名服务器返回UW的域名服务器。上述过程显示在图中的第4步和第5步。现在，快接近目标了，本地域名服务器把查询发送给UW的域名服务器（第6步）。如果被查询的域名是英语系，则马上能找到答案，因为UW域包括了英语系。但是，计算机科学系选择运行自己的域名服务器，因此该查询返回的是UW计算机科学系的域名服务器和IP地址（第7步）。

最后，本地域名服务器查询UW计算机科学系的域名服务器（第8步）。这台服务器是cs.washington.edu的权威机构，所以它必定有答案。它返回最终的答案（第9步），即作为响应flits.cs.vu.nl（第10步）转发的本地域名服务器。至此，查询的域名得到了解析。

你可以用标准的工具来探寻整个查询过程，比如安装在大多数UNIX系统中的dig程序。例如，键入：

```
dig@a.edu-servers.net robot.cs.washington.edu
```

向a.edu-servers.net域名服务器发出一个针对robot.cs.washington.edu的查询，并打印出查询结果。这个结果显示了上面例子中第4步获得的信息，你将了解到UW域名服务器的名字和IP地址。

有关这种长距离远程查询的场景，有3个技术要点值得讨论。首先，在图7-6中两个不同的查询机制在工作。当主机flits.cs.vu.nl将查询发送给本地域名服务器后，该域名服务器就代替该主机处理域名解析工作，直到它返回所需的答案。这里的答案必须是完整的，它不能返回部分答案。虽然部分答案可能也会有所帮助，但不是查询应该得到的结果。这个机制称为**递归查询**（recursive query）。

另一方面，根域名服务器（和每个后续的域名服务器）并不是递归继续查询本地域名服务器。它只是返回一个部分答案，并移动到下一个查询操作。本地域名服务器负责继续解析，具体做法是发出进一步的查询报文。这个机制称为**迭代查询**（iterative query）。

一次域名解析可以涉及这两种机制，如同该例子所表明的那样。递归查询似乎总是最可取的，但许多域名服务器（尤其是根服务器）并没有采用这种处理方式，它们太忙了。迭代查询则将重负压在了查询发起方。本地域名服务器支持递归查询的理由是它负责为其域内的主机提供服务。这些主机不必配置成运行一个完整的域名服务器，它们只要刚好能到达本地域名服务器即可。

值得讨论的第二点是缓存。所有的查询答案，包括所有的部分答案都会被缓存。这样，如果另一台cs.vu.nl主机需要查询robot.cs.washington.edu，那么答案早就有了。更妙的是，如果在同一个域中的一台不同主机需要查询另一台主机，比如galah.cs.washington.edu，那么此次查询可直接发送到权威的域名服务器。类似地，针对washington.edu其他域的查询也可以从washington.edu域名服务器直接开始。这种使用缓存答案的做法可大大降低一次查询的步骤，并能提高查询性能。事实上，我们在例子中勾勒出的原始场景是最坏的情况，

通常发生在没有任何缓存有用信息时。

然而，高速缓存的答案不具权威性，因为在 cs.washington.edu 所做的信息变化将不会传播到世界上所有可能知道它的缓存。出于这个原因，缓存的表项不应该生存得太长。这就是为什么在每条资源记录中要包含 Time_to_live 字段的原因。它告诉远程域名服务器可缓存本记录多长时间。如果某台机器已经多年使用相同的 IP 地址，将它的信息缓存 1 天可能是安全的。而对于波动比较大的信息，几秒钟或一分钟后清除掉记录的做法或许更安全。

第三个讨论的问题关于查询和响应使用的传输层协议。域名系统采用的是 UDP。DNS 消息通过 UDP 数据包发送，格式非常简单，只有查询和响应；域名服务器可用此数据包继续进行解析操作。我们不讨论这种报文格式的细节。如果在很短的时间内没有响应返回，DNS 客户端必须重复查询请求；如果重复一定次数后仍然失败，则尝试域内另一台域名服务器。查询过程这样设计的主要目的是为了应付出现服务器关闭以及查询或响应包丢失的情况。每个查询报文都包含 16 位的标识符，这个标识符将被复制到响应包中，以便域名服务器将接收到的答案与相应的查询匹配，即使它同时发出了多个查询也不会弄混查询结果。

即使 DNS 的目的很简单，人们应该明确这是一个庞大而复杂的分布式系统，它由数百万计的域名服务器组成，这些服务器要一起协同工作。DNS 在人类可读域名和机器 IP 地址之间形成了一个关键的衔接。为了提高性能和可靠性，它还利用了复制和缓存机制，同时设计得具有很强的鲁棒性。

我们没有涉及任何安全，但正如你可能想象的那样，如果心怀恶意，那么拥有改变域名到地址映射的能力就可能造成灾难性的后果。出于这个原因，研究者们已经开发了专用于 DNS 的安全扩展，这个安全机制称为 DNSSEC。我们将在第 8 章介绍这些内容。

应用程序对域名的使用方式希望更加灵活化，这样的应用需求逐渐显露。例如，首先对内容进行命名，然后解析出拥有该内容的附近一个主机的 IP 地址。这种映射模式特别符合搜索一个电影并下载它的应用模式。这时，人们关注的是电影本身，而不是某个拥有电影拷贝的计算机，因此解析所要的全部结果只是附近具有该影片拷贝的任何一台计算机 IP 地址。内容分发网络就是完成这个映射的一种实现方式。我们将在 7.5 节介绍如何利用本章后面的 DNS 来构建这样的网络。

7.2 电子邮件

电子邮件或者更常用的 E-mail，已经存在 30 多年了。由于比纸质信件更快更便宜，电子邮件成为自早期 Internet 出现以来最广泛的应用。在 1990 年以前，它主要被用于学术界。在整个 20 世纪 90 年代，它变得普及起来并呈指数形式增长，以至于现在每天发送的电子邮件数量远远超过了**传统的纸质邮件**（snail mail）数量。其他形式的网络通信，比如即时消息和 IP 语音在近 10 年也有了极大的发展，但是电子邮件仍然是 Internet 通信的主要负载。电子邮件广泛地用于业界公司内部的通信，例如，分散在世界各地的员工们就一个复杂项目进行协同。不幸的是，像纸质信件一样，大多数电子邮件都是邮寄宣传品或者**垃圾邮件**（spam），某些甚至 10 封邮件里面高达 9 封是垃圾邮件（McAfee，2010）。

与大多数其他通信方式一样，电子邮件有它自己的约定和风格。特别是它非常不拘小

节，使用户门槛很低。那些从来没有梦想过给某个大人物打一个电话或者写一封信的人，可以毫不犹豫地给他发一封随意书写的电子邮件。由于消除了与社会地位、年龄和性别有关的大多数暗示，通过电子邮件展开辩论通常关注于内容本身而不是状态。有了电子邮件，某个暑期学生的绝妙想法比一个执行副总裁的愚钝想法影响力更大。

电子邮件中充满了诸如 BTW（By The Way，顺便）、ROTFL（Rolling On The Floor Laughing，笑得满地打滚）和 IMHO（In My Humble Opinion，恕我直言）等行话。很多人还在他们的电子邮件中使用一些称为**微笑图标**（smiley）的 ASCII 符号，这种符号始于普适的“:-)”。如果你不熟悉这个符号，那么把本书旋转 90°就能明白。这种符号和其他情绪图标（emoticon）有助于传递邮件语气，现在它们已经被扩散到其他形式的通信中，比如即时消息。

电子邮件协议在其使用期间经历了很大的演变。第一个电子邮件系统简单地由文件传输协议和约定组成，约定规定每个邮件的第一行（即文件）必须给出收件人地址。随着时间的推移，文件传输和电子邮件分歧越来越大，最终电子邮件从文件传输中分离出来并增加了许多功能，例如发送一个邮件给一组收件人的能力。在 20 世纪 90 年代，多媒体功能变得非常重要，邮件可以包括图像和其他非文字材料。相应地，阅读电子邮件的程序也变得更为复杂，从单纯的基于文本阅读转变成图形用户界面，并且为用户增加了在任何地方通过笔记本电脑访问邮件的能力。最后，随着垃圾广告邮件的盛行，邮件阅读器和邮件传输协议现在必须具备发现这些不想要的邮件并删除它们的能力。

在我们的电子邮件描述中，我们将精力集中在用户之间如何移动邮件，而没有考查和体验邮件阅读程序。然而，在描述了整体架构后，我们将开始介绍电子邮件系统中面向用户的那一部分，因为它是为大多数读者所熟悉。

7.2.1 体系结构和服务

在本节，我们将提供一个概述，说明电子邮件系统如何组织以及它们可以做什么。电子邮件系统的体系结构如图 7-7 所示。它包括两类子系统：**用户代理**（user agent）和邮件**传输代理**（message transfer agent）。人们通过用户代理阅读和发送电子邮件；邮件传输代理负责将用户邮件从源端移动到目的地。我们把邮件传输代理非正式地称为**邮件服务器**。

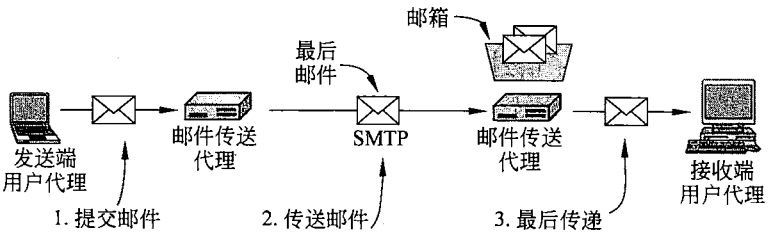


图 7-7 电子邮件系统的体系结构

用户代理是一个程序，用户通过它与电子邮件系统交互。用户代理提供了一个图形界面，有时是一个基于文本和基于命令的接口。它包括了撰写邮件、回复邮件、显示入境邮件信息的手段，同时还提供了如何过滤、搜索和删除邮件的组织方式。把新邮件发送给邮件系统，并通过它传递的行为称为**邮件提交**（mail submission）。

有些用户代理可能会自动完成对邮件的处理，预测用户想要什么。例如，为了提取出或者降低可能是垃圾邮件的优先级，入境邮件可能会先被过滤。某些用户代理还包括一些先进功能，比如安排电子邮件的自动回复（“现在我正在度一个美妙假期，一旦我回去将立即给你回复”）。用户代理运行在用户阅读邮件的同一台计算机上。这只是另一个程序，而且或许只运行一段时间。

邮件传输代理通常是系统进程。它们运行在邮件服务器机器的后台，并始终保持运行状态。它们的工作是通过系统自动将电子邮件从发送端移动到收件人，采用的协议是**简单邮件传输协议**（SMTP，Simple Mail Transfer Protocol）。这是邮件传输的必经之路。

SMTP 最早由 RFC 821 说明，之后被修订成为当前的 RFC 5321。它通过一个连接发送邮件、返回传递状态和任何错误的报告。对许多应用来说，确认交付非常重要，甚至可能具有法律上的意义（“哦，先生，我的电子邮件系统没那么可靠，所以我猜测电子传票可能遗失在某个地方了”）。

邮件传输代理还实现了**邮件列表**（mailing list）功能，一个邮件的完全相同副本被传递到电子邮件地址列表中的每个人。其他先进的功能包括抄送、秘密抄送、高优先级电子邮件、秘密（即加密）电子邮件；如果主要收件人当前不方便接收邮件，那么可指定另一个接收者，以及阅读老板邮件并代替回复邮件的辅助能力。

将用户代理和邮件传输代理衔接起来的是邮箱这个概念，以及电子邮件的标准格式。**邮箱**（mailbox）存储用户收到的电子邮件。邮箱由邮件服务器负责维护，用户代理只需向用户展示邮箱中的内容即可。要做到这一点，用户代理向邮件服务器发送操纵邮箱的命令，包括检查邮箱内容、删除邮件等。邮件检索在图 7-7 中是最后交付（第 3 步）。在这样的体系结构下，一个用户可以在多台计算机上使用不同的用户代理来访问同一个邮箱。

在两个邮件传输代理之间发送的邮件具有标准的格式。原来由 RFC 822 规定的格式已被修订为当前的 RFC 5322，并且扩展成支持多媒体内容和国际文本。这个方案称为 MIME，稍后将对此进行讨论。虽然，人们仍然将 Internet 电子邮件看作 RFC 822。

电子邮件系统的一个关键思想是将**信封**（envelope）与邮件内容区分开来。信封将消息封装成邮件，它包含了传输消息所需要的所有信息，例如目标地址、优先级和安全等级，所有这些都有别于消息本身。消息传输代理根据信封来进行路由，就好像邮局的做法一样。

信封内的消息由两部分组成：**邮件头**（header）和**邮件体**（body）。邮件头包含用户代理所需的控制信息。邮件体则完全提供给收件人，代理和邮件传输代理都不在意邮件体包含了什么信息。图 7-8 中显示了信封和邮件。

下面，我们按照一个用户给另一个用户发送电子邮件涉及的每个步骤，来详细考查这种体系结构下的每个组成部分。这个考查之旅就从用户代理开始。

7.2.2 用户代理

用户代理是一个程序（有时也称为**电子邮件阅读器**），它接收各种各样命令，从接收和回复邮件到操纵邮箱的命令。目前流行的用户代理有许多，其中包括谷歌 Gmail、微软的 Outlook、Mozilla 的 Thunderbird 和苹果的 Apple Mail。这些用户代理在外形上相差很大。大多数用户代理有一个菜单或图标驱动的图形界面，需要使用鼠标进行操作；在小型移动

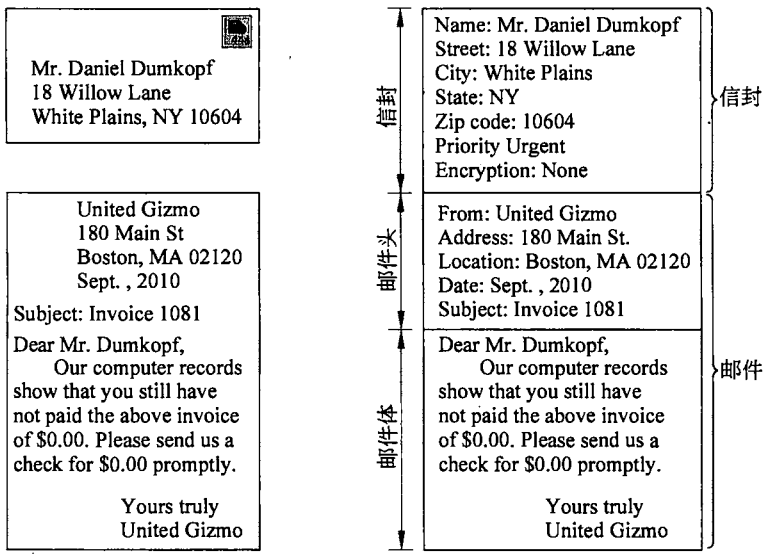


图 7-8 信封和邮件
(a) 纸质信件；(b) 电子邮件

设备上的界面通常是触摸界面。老的用户代理，比如 Elm、mh 和 Pine 提供的是基于文本的界面，期望用户从键盘输入一个字符命令。从功能上看，这些界面都相同，至少对文本消息是一样的。

用户代理界面的典型元素如图 7-9 所示。你的邮件阅读器可能比这个要华丽得多，但或许具有同等的功能。当用户代理被启动时，它一般会给出用户邮箱内邮件的摘要信息。通常情况下，摘要信息是每个邮件占一行，并且按照某种排序排列。摘要突出的是从邮件信封或邮件头中提取出来的一些关键字段。

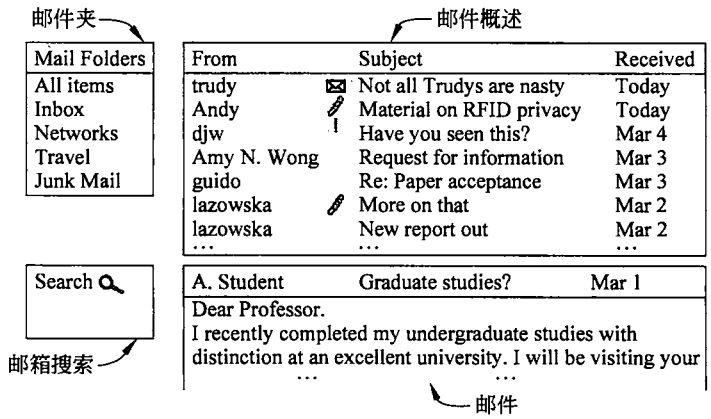


图 7-9 用户代理接口的典型元素

图 7-9 显示了的例子中的摘要信息有 7 行。每一行使用了发件人、标题和何时接收三个字段，显示的顺序可以按照邮件的发送者、邮件的标题以及邮件的接收时间来排列。所有信息的格式都以用户友好的方式呈现，而不是显示消息字段的文字内容，但无论采用什么方式都必须基于邮件字段。因此，如果人们在发送邮件时不包括标题字段，那么他们常

常会发现针对该电子邮件的回复通常得不到最高优先级。

摘要中也可能包括许多其他字段或指示信息。例如，图 7-9 中邮件标题旁边的图标可能表明未读邮件（信封）、有附件（回形针）和重要邮件，至少发件人判定为重要（感叹号）。

还可以按照其他规则进行邮件的排序。最常见的是根据接收到邮件的时间，将最近收到的邮件优先显示，同时还使用一些指示消息是新的还是已被用户读取过的图标。在邮件摘要显示的字段和排序方式用户可以自己的喜好定制。

需要时用户代理必须能够显示入境邮件信息，方便人们阅读他们的电子邮件。通常系统会提供一个短消息预览，如图 7-9 所示，帮助用户决定何时进一步阅读该邮件。预览可以使用小图标或图像来描述邮件的内容。用户代理还提供其他表示的处理方式，包括重新格式化邮件来适应当前的显示，或者翻译或转换邮件内容成更便利的显示格式（例如，可识别文本的数字化语音）。

邮件被阅读过后，用户可以决定用它来做什么。这就是所谓的邮件处置（message disposition）。邮件的进一步处理选项包括删除邮件、回复邮件、把邮件转发给另一个用户，以及保存邮件供日后参考。大多数用户代理可以用多个文件夹保存一个邮箱的入境邮件。文件夹允许用户根据发件人、标题或一些其他分类方法来保存消息。

在用户阅读邮件之前，用户代理可以自动完成邮件的归档。一个常见的例子，系统检查邮件的字段和内容，根据有关以前邮件的用户反馈，使用邮件字段和内容来确定是否可能是垃圾邮件。许多 ISP 和企业都运行这样的软件，给邮件贴上重要或者垃圾邮件的标签，这样用户代理可以将这些邮件归档到相应的邮箱中。对于许多用户来说，ISP 和公司具有预先看到邮件的优势，并且它们可能拥有已知垃圾邮件发送者的名单。如果几百个用户恰好刚刚收到类似的消息，那么这个邮件很有可能是垃圾邮件。通过预先将入境邮件分类到“可能合法”和“可能垃圾邮件”的做法，用户代理可以为用户节省从垃圾邮件中分离出有用信息的大量工作。

那么，什么是最流行的垃圾邮件？它是由一组被感染的计算机产生的邮件，具体内容取决于你居住的地方。这种被感染的计算机集合就称为僵尸网络（botnet）。在亚洲，典型的垃圾邮件是假文凭的制作；在美国，廉价药品和其他可疑产品的供给是典型的垃圾邮件；而有关无人认领的尼日利亚银行账户比比皆是；增强身体各部位的假药则无处不在。

用户还可以构造其他归档规则。每个规则指定了一个条件和相应的一个动作。例如，规则可以将来自老板的邮件都放到一个要立即阅读的文件夹中，而指定从某个特定邮件列表发来的任何消息存到另一个文件夹供稍后阅读。图 7-9 显示了几个文件夹。其中最重要的文件夹是 Inbox（收件箱），用来存放没有预先规定存放的邮件，即没有其他地方可去的入境邮件；还有一个文件夹是 Junk Mail（垃圾邮件），被认为是垃圾邮件的邮件都归类到该文件夹。

除了显式构造自己喜欢的各种文件夹外，用户代理还为用户提供了丰富的搜索邮箱功能。此功能如图 7-9 所示。搜索能力为用户提供了快速查找邮件的手段，比如上个月有人发送的有关“在哪里买 Vegemite”的邮件。

电子邮件自从刚出现时作为一种文件传输以来，已经走过了很长的路。现在服务提供商经常能为用户支持高达 1 GB 的邮箱用来存储邮件，而且这些邮件是用户在很长一段时间内邮件交互的细节。用户代理先进的邮件处理的能力，包括搜索和自动分类处理，使得

它可以管理这些大量的电子邮件。对于一年要发送和收到数千封电子邮件的用户来说，这些工具非常宝贵。

另一个有用的功能是用户代理能够以某种方式自动响应邮件。一种响应是将入境的电子邮件转发到一个不同的地址，例如，由一个商业传呼服务所操纵的计算机，通过无线电或卫星寻呼用户，并在他的寻呼机上显示“主题：”行。这些自动响应功能必须运行在邮件服务器上，因为用户代理可能无法在所有的时间内都保持运行状态，而且很有可能只是偶尔检索电子邮件。基于这些因素的考虑，用户代理不能提供一个真正的自动响应服务。然而，自动响应的接口通常由用户代理表示。

另一个不同的自动响应例子是**休假代理**（vacation agent）。这是一个程序，它首先检查每个入境消息，然后给发件人回送一个平淡的答复，比如：“嗨。我在度假。我将于 8 月 24 日回来后立即联系你。”这样的答复也可以指定如何处理在此期间发生的紧急事项，或者针对某人就某个特殊问题的邮件给予回复等。大多数度假代理跟踪它们已经给哪些用户发过这种“罐装回复”（canned reply），以免给同一人重复发送相同的回复。然而，这些代理依然有缺陷。例如，它给来自一个大邮件列表的群发邮件回送一个罐装答复显然是不可取的。

让我们现在把注意力转到一个用户将消息发送给另一个用户的场景。用户代理支持的基本功能之一是邮件的组成，这点我们还没有讨论过。邮件的组成涉及创建邮件、回复邮件，以及为了将邮件交付给收件人而将这些邮件发送到邮件系统的其余部分。虽然在创建邮件时，可以使用任何文本编辑器来撰写邮件正文，但邮件正文编辑器通常集成在用户代理中，以便它可以为每个邮件提供类似寻址和多个头字段的辅助功能。例如，在回复邮件时，电子邮件系统可以从收到的邮件中提取发件人的地址，并自动插入到回复邮件的适当的地方。其他一些共同功能包括在邮件底部追加一个**签名块**（signature block）、更正拼写错误和计算表明该消息是否有效的数字签名。

发往邮件系统的邮件遵循一个标准格式，这个格式的创建基于提供给用户代理的信息。传输消息的最重要部分是信封，信封中最重要的是目的地址。邮件目的地址必须是邮件传输代理可以处理的格式。

期望的地址形式是用户@DNS 地址（user@dns-address）。由于我们已经在本章前面研究过 DNS，我们在这里就不再赘述。然而，值得注意的是还存在其他形式的地址。特别是，X.400 地址看上去和 DNS 地址完全不同。

X.400 是邮件处理系统的 ISO 标准，它曾经是 SMTP 的竞争对手。SMTP 轻而易举地取得了胜利，虽然仍有人在使用 X.400 系统，而且使用者大多在美国以外的地区。X.400 地址由“属性=值”对组成，相互之间用斜线分开，例如：

```
/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/
```

这个地址指定了一个国家、国内的州、地方城市、个人的地址和通用名字（Ken Smith）。标准还允许许多其他属性，你可以发送电子邮件给某个人，他的准确电子邮件地址你并不知道，只要你知道关于他的足够其他属性（例如，公司和职位）。

虽然 X.400 名字使用的方便程度大大低于 DNS 域名，但这个问题对用户代理没有实际意义，因为它们有用户友好的别名（有时也称为昵称），允许用户输入或选择一个人的名字，

并得到正确的电子邮件地址。因此，它通常没有必要实际输入这些奇怪的字符串。

最后一点与发送邮件有关的是邮件列表，即允许用户通过一个命令把相同的邮件发送给群发列表中的每个人。如何维护这个邮件列表有两个选择。第一个，可以由用户代理在本地维护。在这种情况下，用户代理只是给每个目标收件人发送一个单独的邮件。

另一个选择是远程维护一个在邮件传输代理上的群发名单。然后，邮件在整个邮件传输系统中被扩散，等同于多个用户给群发列表中的用户发送邮件的作用。例如，如果一群观鸟者有一个称为 birders 的邮件列表，该群发列表安装在传输代理 meadowlark.arizona.edu 上，那么任何发送到 birders@meadowlark.arizona.edu 的邮件都将被路由到美国亚利桑那大学，并在那里被进一步扩散到邮件列表中的所有成员，这些成员可能分布在全世界的任何地方。这个邮件列表中的用户无法分辨这是一个邮件列表，它可能只是 Gabriel O. Birders 教授的个人邮箱。

7.2.3 邮件格式

现在我们从用户接口转到邮件消息格式本身。用户代理发出的邮件必须设置成邮件传输代理能处理的标准格式。首先，我们将考查使用 RFC 5322 的基本 ASCII 电子邮件，然后我们再考查基本格式的多媒体扩展。RFC 5322 是 Internet 邮件格式的最新版本，最初的 Internet 邮件格式由 RFC 822 描述。

RFC 5322——Internet 邮件格式

邮件由一个基本的信封（作为 SMTP 的一部分由 RFC 5321 描述）、数个头字段、一个空行和邮件体组成。头的每个字段（逻辑上）由一行 ASCII 文本组成，其中包括域名、一个冒号，对于大多数头的字段来说还包括一个值。几十年前完成设计的最初 RFC 822 没有明确区分开信封字段和头字段。虽然 RFC 5322 已经对 RFC 822 做了许多修订，但由于 RFC 822 已经被广泛使用，因此要想完全重新设计是不可能的。在一般的用法中，用户代理创建一个邮件并把它传递给邮件传输代理，然后邮件传输代理利用某些头字段来构造出实际的信封，这个信封有点像老式的邮件与信封混合体。

图 7-10 中列出了与邮件传输相关的主要头字段。“To:”字段给出了主要收件人的 DNS 地址，邮件系统允许有多个收件人。“Cc:”字段给出了所有次要收件人的地址。从邮件投递的角度来看，主收件人和次收件人没有区别。这完全是一种心理上的差别，这种差别也许对于相关的人而言很重要，但是对邮件系统来说无关紧要。“Cc:”（Carbon copy，抄送）这个术语有点过时，因为计算机不使用复写纸，但是它已经被约定成俗了。“Bcc:”（Blind carbon copy，密件抄送）字段与 Cc:字段类似，只是在发送给主收件人和次收件人的所有副本中，这一行被删除了。这个特性允许人们在主收件人和次收件人都不知道的情况下，向第三方发送邮件副本。

接下来的两个字段“From:”和“Sender:”分别指出邮件的撰写者与邮件的发送者，这二者不一定相同。例如，一个公司经理撰写了一个邮件，但她的助理可能是实际发送该邮件的人。在这种情况下，经理被列在“From:”字段中，而她的助理应该出现在“Sender:”字段。“From:”字段是必需的，但是，如果“Sender:”字段与 From:字段的值相同，那么

可以省略“Sender:”字段。一旦邮件无法投递并且必须被退回发件人时，这些字段就都是必需的了。

邮件头	含义
To:	主要收件人的电子邮件地址
Cc:	次要收件人的电子邮件地址
Bcc:	密件抄送的电子邮件地址
From:	标识了邮件撰写者
Sender:	邮件实际发送者的电子邮件地址
Received:	该行由沿途的每个传输代理填入
Return-Path:	可用于标识邮件发送者的回复路径

图 7-10 RFC 5322 中与邮件传输相关的头字段

在邮件投递的路径上，每个邮件传输代理都会给该邮件加上一行。该行包含“Received:”字段，给出了该代理的标识符、接收到此邮件的日期和时间，以及其他一些可用来查找路由系统中错误的信息。

“Return-Path:”字段由最后一个邮件传输代理添加，主要用来指示如何返回至发件人。理论上，这个信息可以从所有的“Received:”头字段（除了发件人邮箱的名字）收集得到，但实际上很少填充该字段，它一般只包含发件人的地址。

除了图 7-10 中列出的字段以外，RFC 5322 消息还包含了各种供用户代理或者收信人使用的字段。图 7-11 中列出了最常见的一些头字段，它们大部分的含义都不言自明，因此这里我们不对所有的头字段都做详细的介绍。

邮件头	含义
Data:	邮件发出的日期和时间
Reply-To:	回复邮件应采用的电子邮件地址
Message-Id:	稍后引用本邮件的唯一编号
In-Reply-To:	本回复邮件所针对的邮件ID
References:	其他相关邮件的ID
Keywords:	用户选择的关键字
Subject:	一行显示的邮件概要

图 7-11 RFC 5322 邮件头使用的某些字段

有的时候，当邮件的撰写者或者邮件的发送者都不希望看到邮件的回复时，就可以使用“Reply-To:”字段。例如，市场部经理撰写了一个电子邮件消息，向客户介绍一个新产品。秘书发送了这个消息，但“Reply-To:”字段列出的是销售部门的负责人，因为他可以回答问题并处理订单。当发件人有两个电子邮件账号，并希望消息被回复到其中另一个账号时，这个字段也很有用。

“Message-Id:”字段是自动产生的，用于链接邮件（比如，当使用“In-Reply-To:”字段时）并防止重复传递邮件。

RFC 5322 文档明确地指出用户可以发明新的邮件头供自己私人使用，只要这些邮件头（自从 RFC 822 以来）以字符串“X-”开头即可。RFC 822 保证将来的邮件头不会使用以X-作为开头的名字，以免官方邮件头与私用邮件头之间发生冲突。有时，一些聪明的大学生拼凑出像“X-Fruit-of-the-Day:”或“X-Disease-of-the-Week:”这样的字段，尽管它们不

一定有启发性的含义，但它们却是合法的。

在邮件头之后是邮件体。用户可以在这里放置他们想放的任何内容。有些人用精心设计的签名来结束他们的邮件，这些签名包括对大大小小某些权威著作的引用、政治声明以及各种各样的声明（例如，XYZ公司不对我的观点负责；事实上，它甚至不理解我的观点）。

MIME——多用途Internet邮件扩展

在ARPANET早期，电子邮件只能由文本消息组成，这些消息用英文书写并且以ASCII码形式表示。在这样的环境下，RFC 822完全能够胜任所需的工作：它指定了邮件头，但是把邮件内容完全留给用户自己。在20世纪90年代，Internet在全球范围得到广泛使用，发件人希望通过邮件系统发送更为丰富多彩邮件内容的需求越来越强烈，因此原先的这种方法不再够用。主要问题包括了以下情况下的发送和接收两个方面：用带有重音符的语言（例如，法语和德语）来撰写的邮件、用非拉丁字母来撰写的邮件（例如，希伯来语和俄语）、用不带字母的语言来撰写的邮件（例如，中文和日文）以及完全不包含文本的邮件（例如，音频、图像或二进制文档和程序）。

解决方案是**多用途Internet邮件扩展**（MIME，Multipurpose Internet Mail Extensions）。它已被广泛应用于在Internet上收发邮件消息，除此之外它还可以描述诸如Web浏览器等其他应用的内容。有关MIME的信息由RFC 2045～2047、4288、4289及2049文档描述。

MIME的基本思想是继续使用RFC 822格式（在RFC 5322之前MIME就已经被提出来了），但在邮件体中增加了结构性，并且为传送非ASCII码的邮件定义了编码规则。由于它没有偏离RFC 822，因此已有的邮件传输代理和协议（基于RFC 821，现在是RFC 5321）都可以发送MIME消息。需要改变的只是发送和接收程序，而这些可以由用户自己来完成。

MIME定义了5种新的邮件头，如图7-12所示。第一个邮件头只是告诉接收邮件的用户代理，它正在处理一条MIME消息，以及该消息使用的是哪个版本的MIME。任何不包含“MIME-Version:”邮件头的消息都被假定是英语明文消息（或者至少只使用ASCII字符），并按照这种假定情况下的方式来进行处理。

邮件头	含义
MIME-Version:	标识了MIME版本
Content-Description:	描述邮件的可读字符串
Conten-Id:	唯一标识符
Conten-Transfer-Encoding:	指出传输时如何打包
Conten-Type:	邮件内容的类型和格式

图7-12 MIME添加的邮件头

“Content-Description:”头是一个ASCII字符串，它指出了邮件包含什么内容。这个头是必需的，这样收件人才能知道是否值得解码和阅读该消息。如果这个字符串说的是“圣巴巴拉仓鼠的照片”，而收到该消息的人并不是一个狂热的仓鼠迷，那么这条消息就可能被丢弃，而不会被解码成一幅高清晰的彩色照片。

“Content-Id:”头标识了邮件的内容，它采用了与标准的“Message-Id:”头相同的格式。

“Content-Transfer-Encoding:”头指出了如何包装邮件体，以便通过网络进行传输。当开发MIME时，邮件传送协议（SMTP）只能处理一行不超过1000个字符的ASCII邮件，

这是一个很关键的问题。ASCII 字符占了每 8 位字节中的 7 位，而二进制数据（比如可执行程序和图像）则使用了每个字节的全部 8 位作为扩展的字符集。如果没有任何保障措施，是无法安全传送这种数据的。因此，需要某种方式使得携带二进制数据的邮件看起来就像常规的 ASCII 邮件消息一样。自从 MIME 开发出来之后，SMTP 也做了相应的扩展，允许邮件正文传送 8 位的二进制数据，即使今天二进制数据不进行编码，可能仍然不能被邮件系统正确传送。

MIME 提供了 5 种传送编码方法，还有一个扩充新方案的入口（只是为了以防万一）。最简单的编码方案就是 ASCII 文本消息。每个 ASCII 字符占 7 位，可以被邮件协议直接传送，只要每行都不超过 1000 个字符。

次简单的方案与上一个 ASCII 方案相同，但使用的是 8 位字符。也就是说，从 0 到 255 并且包括 255 在内的所有值都是允许的。使用 8 位编码的邮件仍然必须遵循标准的最大行长度。

然而，存在一些邮件使用了真正的二进制编码。它们是任意的二进制文件，不仅使用全部的 8 位，甚至也不遵循每行 1000 个字符的限制。可执行程序就属于这一类消息。现在，邮件服务器可以协商是否发送二进制（或者不是 8 位的）编码消息，如果双方都不支持该扩展那么就回退到 ASCII 字符。

二进制数据的 ASCII 编码方式称为基于 64 编码（base64 encoding）。在这种方案中，每 24 位编成一个组，每组被分成 4 个 6 位的单元，每个单元被当作一个合法的 ASCII 码字符来发送。编码方式是“A”表示成 0、“B”表示成 1，以此类推。接着是 26 个小写字母、10 个数字，最后是“+”和“/”分别表示成 62 和 63。“==”和“=”分别表示最后一个组只含有 8 位或者 16 位。回车和换行被忽略，所以它们可被任意插入到编码字符流中以便保证每一行足够短。使用这种编码方案后，任意的二进制文本都可以被安全地发送出去，尽管效率不高。在二进制的邮件服务器开发出来之前，这种编码方式非常流行。现在仍然经常有人在用。

对于那些几乎全是 ASCII 字符，只有少量非 ASCII 字符的邮件，bas64 编码的效率有些低。发送这种邮件时采用了另一种替代方案，这种方案称为引用可打印编码（quoted-printable encoding）。它正好是 7 位 ASCII 编码，但所有超过 127 的字符都被编码成一个等号后面跟着 2 个用十六进制数字来表示的字符值。控制字符、某些标点符号、数学符号以及结尾空白也用这种方式编码。

最后，如果有足够的理由不使用上述任何一种编码方案，那么还可以通过“Content-Transfer-Encoding:”头说明一种用户自定义的编码。

图 7-12 中显示的最后一个邮件头实际上是最有趣的一个。它指定了邮件体的本质特性，而且具有超出电子邮件范畴的影响。例如，从 Web 下载的内容被打上 MIME 类型的标签，这样浏览器就知道如何表示该内容。同样地，对于诸如 IP 语音这样的流式媒体和实时传输，作为邮件内容发送时也作同样的处理。

最初，RFC 1521 定义了 7 种 MIME 类型。每种类型都有一个或多个子类型。类型和子类型由一个斜线隔开，比如“Content-Type: video/mpeg”。从此以后，针对每个类型都有数百个子类型被扩充进来。整个过程中，每当开发出某种新的内容类型时要添加额外的表项。分配的类型和子类型列表由 IANA 在线维护 www.iana.org/assigments/media-types。

类型以及相关的常用子类型例子如图 7-13 所示。让我们简单地浏览一下，从“text（文本）”开始。“text/plain（文本/纯文本）”结合起来可用于表达普通邮件，显示的就是收到的，无须编码，也不需要进一步处理。这个选项允许以 MIME 方式传输普通邮件，只需少数几个额外的头。“text/html（文本/网页）”子类型在 Web（RFC 2854）变得流行后允许在 RFC 822 电子邮件中发送网页。“text/xml（文本/可扩展标记语言）”子类型在 RFC 3023 中定义。XML 文档刺激了 Web 的发展。我们将在 7.3 节学习 HTML 和 XML。

类型	子类型实例	描述
text	plain, html, xml, css	不同版本格式的文本内容
image	gif, jpeg, tiff	照片
audio	basic, mpeg, mp4	声音
video	mpeg, mp4, quicktime	影片
model	vrml	3D模型
application	octet-stream, pdf, javascript, zip	由应用程序生成的数据
message	http, rfc822	封装的邮件
multipart	mixed, alternative, parallel, digest	多个类型的组合

图 7-13 MIME 内容类型和子类型实例

下一个 MIME 类型是 image（图像），它可传输静态图像。现在广泛用来存储和传输图像的格式有许多种，它们既可以是压缩的，也可以是未压缩的。其中的一些，包括 GIF、JPEG 和 TIFF 几乎被内置到所有的浏览器中。除此以外还存在很多其他的图像格式和相应的子类型。

audio（音频）和 video（视频）类型分别用于表示声音和运动图像。请注意，video 类型只包含可视信息而不包含声音。如果要传输一部有声电影，那么视频和音频部分必须被分开传输，具体如何操作则要依据所使用的编码系统。第一个定义的视频格式称为**运动图像专家组**（MPEG，Moving Picture Experts Group），这种格式就是由这组专家设计的，但此后又增加了其他一些格式。除了“audio/basic”以外，在 RFC 3003 中增加了一种新的音频类型“audio/mpeg”，从而使人们可以通过电子邮件发送 MP3 音频文件。“video/mp4”和“audeo/mp4”类型说明以新 MPEG 4 格式来存储视频和音频数据。

在其他内容类型之后增加了 model（模型）类型，主要目的是用来描述 3D 模型数据。然而，该类型至今尚未得到广泛的应用。

application（应用）类型是对那些未被其他类型覆盖但又需要应用程序来解释数据的未知格式的统称。例如，我们分别为 PDF 文档、JavaScrip 程序和 Zip 压缩包列出了 pdf、javascript 和 zip 子类型。接收到这种邮件内容的用户代理使用第三方库函数或者外部程序来显示内容；显示程序可以集成到用户代理，也可以不和用户代理集成在一起。

通过使用 MIME 类型，用户代理获得了一种处理应用内容的扩展能力。随着新类型应用程序被不断地开发出来，其传输的内容类型也可能是全新的，因此用户代理的这种可扩展处理能力好处很大。但另一方面，许多新形式内容由应用程序执行或者解释也带来了一些危险。显然，运行一个任意可执行程序是有安全隐患的，虽然通过邮件系统传送的可执行程序可能出自“朋友们”。但这种程序可能会对它可以访问到的那部分计算机带来各种令人讨厌的损害，尤其是如果它能够读取和写入文件，以及使用网络时危害更大。不那么明显的是文档格式也可以带来同样的危险。这是因为这些格式（如 PDF）根本是变相的编程

语言。虽然它们被限定在一定范围内解释，但解释器中的错误常常让狡猾的文件挣脱限制的束缚。

因为存在许多应用，除了这些例子外还存在许多各种各样的应用程序子类型。如果没有其他已知的子类型更合适描述邮件，那么作为备用“octet-stream”子类型表示一个无法解释的字节序列。在接到这样一个字节流时，用户代理很可能会显示它，建议用户将其复制到一个文件。然后，在用户大概知道内容是什么样子之后，再决定对该内容做何种后续处理。

最后两个类型对于撰写和处理邮件本身非常有用。message 类型使得一个邮件可以被完全封装在另一个邮件中。这种方案对于转发电子邮件很有用。例如，当一个完整的 RFC 822 邮件被封装在另一个外部邮件中时，则应该使用 RFC 822 子类型。类似地，对于封装 HTML 文档也很普遍。partial 子类型使得有可能将一个被封装的邮件分成多个部分并分别传输它们（例如，如果被封装的邮件太长）。通过一些参数，接收方能够将各部分按照正确的顺序重新组装起来。

最后一个类型是 multipart，它允许一个邮件包含多个部分，并且明确地分出每个部分开始和结束的界限。“mixed”子类型允许每个部分类型都不相同，而且无须强制任何额外结构。许多电子邮件程序允许用户在一个文本消息中提供一个或多个附件，这些附件可以 multipart 类型发送。

与 mixed 相反的是，alternative 子类型允许同一个消息被包含多次，但是被表示成两种或多种不同的媒体。例如，一个消息可以按照 3 种形式来发送：纯粹的 ASCII 文本、HTML 和 PDF。一个设计合理的用户代理在接收到这样的消息时，将按照用户的偏好显示。如果可能的话，首选用 PDF 格式显示。第二选择应该是 HTML 格式。如果这两者都不可能，则显示简单的 ASCII 文本。这些部分应该按从简单到复杂的顺序排列，以帮助那些使用非 MIME（pre-MIME）用户代理的收件人也能够在一定程度上理解消息的含义（例如，即使非 MIME 的用户也可以阅读简单的 ASCII 文本）。

alternative 子类型也可被用来表示多种语言。在这个上下文意义下，罗塞塔石碑（Rosetta Stone）或许可以被看成是一条早先的 multipart/alternative 消息。

有关子类型的其他两个例子是 parallel 和 digest 子类型。当邮件所有部分必须被同时“观看”时，要使用 paralled 子类型。例如，电影往往由一个音频和视频组成。如果这两个频道不是并行播放而是先后播放，那么电影就无法得到应有的播放效果。当多个邮件被打包成一个复合邮件时，要使用 digest 子类型。例如，Internet 上的一些讨论组往往收集来自各个本组成员的信息，然后将它们作为单个“multipart/digest”邮件定期发送到该讨论组。

下面通过一个例子说明电子邮件如何采用 MIME 类型，图 7-14 给出了一个多媒体邮件。在这里，生日问候信息同时被当作 HTML 和音频文件通过邮件代理传送。假设收件人有播放音频的功能，他的用户代理在展示邮件时将播放声音文件。在这个例子中，声音是作为 message/external-body 子类型被引用的，因此用户代理首先必须通过 FTP 获取声音文件 birthday.snd。如果收件人没有音频能力，那么歌词就会悄无声息地显示在屏幕上。两部分之间的界限由两个连字符来划分，连字符后面跟着一个字符串（由软件生成的），字符串说明了 boundary 参数。

```
From: alice@cs.washington.edu
To: bob@ee.uwa.edu.au
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@cs.washington.edu>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/html
<p>Happy birthday to you<br>
Happy birthday to you<br>
Happy birthday dear <b> Bob </b><br>
Happy birthday to you</p>

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
    access-type="anon-ftp";
    site="bicycle.cs.washington.edu";
    directory="pub";
    name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

图 7-14 由 HTML 和音频多个部分组成的邮件

请注意，在这个例子中，“Content-Type”邮件头出现在 3 个位置上。在顶层，它指出该邮件由多个部分组成。在每个部分中，它指出了该部分的类型和子类型。最后，在第二个部分的正文中，它必须告诉用户代理应该获取什么类型的外部文件。为了显示这种用法的细微不同之处，我们在这里使用了小写字母，尽管所有的邮件头是不区分大小写的。同样地，对于不是按 7 位 ASCII 编码的外部邮件体，则需要提供“Content-Transfer-Encoding”头。

7.2.4 邮件传送

既然我们已经描述了用户代理和邮件消息，现在已经做好一切准备来进一步考查邮件传输代理如何将邮件从发件人中继给收件人。邮件传送采用的协议是 SMTP。

移动邮件最简单的方法是建立一个从源机器到目标机器的传输连接，然后在该连接上传输邮件。这是 SMTP 的最初工作方式。然而，经过多年的发展，邮件的传输逐步区分出了两种使用 SMTP 的不同方式。第一种使用方式是**邮件提交**（mail submission），如图 7-7 所示的电子邮件体系结构中的第 1 步。这一步表示用户代理把邮件提交给邮件系统。第二种使用方式是邮件传输代理之间的邮件传送（图 7-7 的第 2 步）。这个序列将邮件从发送邮件传输代理传递到接收邮件传输代理，全程只有一跳。完成最终邮件的交付使用了不同的协议，我们将在下一节中描述。

在本小节，我们将描述基本的 SMTP 协议和它的扩展机制。然后，我们将讨论如何利

用它完成邮件提交和邮件传送。

SMTP（简单邮件传输协议）及其扩展

在 Internet 上，发送电子邮件的计算机首先与目标计算机的 25 号端口建立一个 TCP 连接，然后在此连接上传送电子邮件。在这个端口上监听的是邮件服务器，它遵守**简单邮件传输协议**（SMTP，Simple Mail Transfer Protocol)。这个服务器接受入境连接请求、执行某些安全检查，并接受传递过来的邮件。如果一个邮件无法被投递，则向邮件发送方返回一个错误报告，该错误报告包含了无法投递邮件的第一部分。

SMTP 是一个简单的 ASCII 协议。这不是一个弱点，而是一种特性。因为使用 ASCII 文本，使得协议更加易于开发、测试和调试。通过手动发送命令就可以进行测试，而且记录的消息易于阅读。现在大多数应用程序级的 Internet 协议都是以这种方式工作的（比如 HTTP)。

我们将考查负责传递消息的邮件服务器如何完成一个简单邮件的传输过程。在建立了与 25 端口的 TCP 连接后，作为客户端的发送机器等待接收机器首先说话，这个接收机器是作为服务器运行的。服务器开始发送一个文本行给客户端，该行表明了自己的标识，并告诉客户机是否已经准备好接收邮件。如果服务器没有这样做，那么客户端就释放连接，稍后再次尝试与服务器联系。

如果服务器愿意接收电子邮件，则客户端声明这封电子邮件来自于谁以及将要交给谁。如果接收方确实存在这样的收件人，则服务器指示客户发送邮件；然后客户发送邮件，服务器予以确认。因为 TCP 提供了可靠的字节流传输，所以这里不需要校验和。如果还有更多的电子邮件需要传输，那么现在可以继续发送。当两个方向上所有的电子邮件都交换完毕后，连接被释放。图 7-14 所示的发送邮件会话实例如图 7-15 所示，图中还包含了 SMTP 所使用的数字代码。客户端发送的行用“C:”标识，而服务器发送的行则用“S:”标识。

来自客户的第一条命令实际意味着 HELO。在 HELLO 的各种 4 个字符缩写组合中，这种缩写相比于其竞争者而言有诸多优点。随着时间的流逝，为什么所有的命令必须是 4 个字符的原因现在已经无从追究了。

在图 7-15 中，邮件只发送给一个收件人，因此这里只使用了一条 RCPT 命令。RCPT 命令可以将一个邮件发送给多个收件人。每个邮件被单独确认或拒绝。即使发给有些收件人的邮件遭到了拒绝（由于接收方并不存在这样的收件人)，邮件也可以被发送给其他的收件人。

最后，尽管客户端命令的语法被严格地限定为四字符，但是回复消息的语法就不那么严格了。实际上，只有数字代码才具有真正的意义。每一种实现都可以在数字代码后面加上它所喜欢的任何字符串。

基本 SMTP 运作良好，但它在几个方面存在不足。首先它不包括认证。这意味着例子中的 FROM 命令可以为所欲为地给出任何发件人的地址。这个特性对于发送垃圾邮件相当有用。其次，SMTP 传输的是 ASCII 消息而不是二进制数据。这就是为什么需要 Base64 MIME 内容传送编码方案。然而，使用该编码的邮件在传输时带宽使用效率低下，这对传输大邮件是个问题。最后，SMTP 发送的邮件以明文形式出现。它没有任何加密功能可用来提供防止窥探隐私的措施。

```
          S: 220 ee.uwa.edu.au SMTP service ready
C: HELO abcd.com
          S: 250 cs.washington.edu says hello to ee.uwa.edu.au
C: MAIL FROM: <alice@cs.washington.edu>
          S: 250 sender ok
C: RCPT TO: <bob@ee.uwa.edu.au>
          S: 250 recipient ok
C: DATA
          S: 354 Send mail; end with "." on a line by itself
C: From: alice@cs.washington.edu
C: To: bob@ee.uwa.edu.au
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@ee.uwa.edu.au>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/html
C:
C: <p>Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Bob </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:     access-type="anon-ftp";
C:     site="bicycle.cs.washington.edu";
C:     directory="pub";
C:     name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
          S: 250 message accepted
C: QUIT
          S: 221 ee.uwa.edu.au closing connection
```

图 7-15 alice@cs.washington.edu 给 bob@ee.uwa.edu.au 发送一个邮件

为了处理这些以及其他与邮件处理相关的问题，SMTP 已经被修订过，增加了一个扩展机制。这个机制是 RFC 5321 标准的强制性执行部分。带有扩展功能的 SMTP 就称为扩展的 SMTP（ESMTP, Extended SMTP）。

想要使用扩展版本的客户端必须发送 EHLO 消息，而不是原先的 HELO。如果该消息遭到了服务器的拒绝，那么说明对方是一个普通的 SMTP 服务器，客户端应该以从前的方式与其交互。如果 EHLO 消息被服务器接受，服务器就用它支持的扩展给予回复。然后客户端可以使用这些扩展功能中的任何一种。几种常见的扩展功能如图 7-16 所示。该图给出了扩展机制所用的关键字，以及这些关键字具有的新功能说明。我们不会对这些扩展做详

细的讨论。

关键字	描述
AUTH	客户认证
BINARYMIME	服务器接受二进制邮件
CHUNKING	服务器接受巨型邮件
SIZE	在发送前检查邮件大小
STARTTLS	切换至安全传输（TLS，参见第8章）
UTF8SMTP	国际化地址

图 7-16 某些 SMTP 扩展功能

为了更好地理解 SMTP 协议和本章描述的其他协议是如何工作的，最好试用这些协议。无论使用什么协议，首先必须来到一台已接入 Internet 的计算机前面。在 UNIX（或 Linux）系统上，在 shell 中键入：

```
telnet  mail.isp.com  25
```

请用你 ISP 的邮件服务器的 DNS 名字来替换 mail.isp.com。在 Window XP 系统中，首先单击“开始”（Start），然后单击“运行”（Run），并且在对话框中键入上述命令。在 Vista 或 Windows 7 机器上，你必须首先安装 telnet 程序（或等价的其他程序），然后你自己启动该程序运行。这条命令将与那台机器的 25 端口建立一个 telnet（也就是 TCP）连接。25 号端口是 SMTP 端口；参考图 6-34，它列出了部分常用的端口。你可能会得到类似这样的回应：

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^]'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

前三行来自于 telent 程序，告诉你它在做什么。最后一行来自远程机器上的 SMTP 服务器，声明它愿意与你通话，并愿意接受电子邮件。为了找出该 SMTP 服务器可以接受哪些命令，请键入：

```
HELP
```

如果服务器愿意接受你的邮件，那么从现在开始，你有可能看到像图 7-16 所示的命令序列。

邮件提交

最初用户代理和负责发送的邮件传输代理运行在同一台计算机上。在这样的设置中，发送邮件所需要做的只是通过我们刚才所描述的对话框，用户代理与本地邮件服务器交互完成邮件的发送。但是，这种设置现在已经不常用了。

用户代理通常运行在笔记本电脑、家用电脑和移动电话上。它们并不总是能连接到 Internet。邮件传输代理则运行在 ISP 和公司的服务器上，它们始终和 Internet 保持连接。这种差异意味着，在波士顿的用户代理可能需要联系他在西雅图的常规邮件服务器发送邮件，因为这个用户正在旅行中。

就其本身而言，这种远程通信不会构成任何问题。它恰好是TCP/IP协议旨在支持的功能。然而，ISP或公司通常不希望任何远程用户将邮件提交给它的邮件服务器再传递到其他地方。ISP或公司并没有为了提供公共服务而运行邮件服务器。此外，这种**开放邮件中继**（open mail relay）会吸引垃圾邮件发送者，因为这种方式提供了一种清洗原始发件人的途径，从而使得邮件更难以被识别为垃圾邮件。

鉴于这些因素，SMTP通常被用在提交电子邮件时启用了AUTH扩展。这个扩展可以让服务器检查客户端的凭据（用户名和密码），以便确认自己是否应该为该客户提供邮件服务。

在SMTP提交邮件的方式上还存在几个其他方面的差异。例如，587端口优先于端口25，SMTP服务器可以检查和纠正由用户代理发送的邮件格式。有关更多使用SMTP来提交邮件受到的限制，请参阅RFC 4409。

邮件传送

一旦接收到来自用户代理发送的邮件，邮件传输代理便使用SMTP将该邮件传送给接收邮件传输代理。要做到这一点，发送者必须使用目标地址。考虑图7-15中发给bob@ee.uwa.edu.au的邮件。该邮件应该被传递给哪个邮件服务器？

为了确定要联系的正确邮件服务器，必须咨询DNS。在上一节中，我们介绍了DNS如何包含多种类型的资源记录，包括MX记录，或邮件交换器。在这种情况下，发出一次DNS查询，请求ee.uwa.edu.au域的MX记录。这个查询得到的响应是一个包含一个或多个邮件服务器名字和IP地址的有序列表。

然后，发送方的邮件传输代理与邮件服务器IP地址的端口25建立一个TCP连接，通过该服务器能到达接收方的邮件传输代理，并使用SMTP来中继消息；然后接收方的邮件传输代理将发给用户Bob的邮件放置在其正确的邮箱里，供Bob在以后的时间读取。如果接收方有一个庞大的电子邮件基础设施，那么这种本地交付邮件的步骤可能涉及在计算机之间移动邮件。

在传递邮件的过程中，从最初的邮件传输代理到最终的邮件传输代理一跳完成邮件的传输，在这传输过程中不涉及任何中间服务器。然而，邮件传递过程则可能发生需要多次传递的情况。我们已经描述过的一个例子就属于这种情况：当邮件传输代理实现了一个邮件列表时。在这种情况下，邮件要被传递到列表中的每个收件人。邮件在维护该列表的邮件传输代理上被复制成多个邮件，分别发给列表中每个成员的地址。

作为邮件中继的另一个例子，我们来看即使Bob从麻省理工学院毕业后，还可以通过地址bob@alum.mit.edu与他联系。与通过多个账户阅读邮件方式不同的是，Bob可以安排把发送到上述邮件地址的邮件转发到bob@ee.uwa.edu。在这种情况下，发送到bob@alum.mit.edu的邮件将经历两次交付过程。首先，它被发送到邮件服务器alum.mit.edu；然后，它又被转发到邮件服务器ee.uwa.edu.au。这一系列的每一步都是一个完整并且独立的交付过程，只关注到邮件传输代理。

时下另一个需要考虑的是垃圾邮件。今天发送的10个消息中有9个是垃圾邮件（McAfee，2010）。虽然很少有人愿意接收更多的垃圾邮件，但难以避免，因为垃圾邮件通常伪装成普通邮件。在接受邮件之前，进行额外的安全检查能减少接收垃圾邮件的机会。

给 Bob 的邮件发自 alice@cs.washington.edu。接收端的邮件传输代理可以在 DNS 查找发送端的邮件传输代理，即检查 TCP 连接另一端的 IP 地址是否与 DNS 域名相匹配。更一般地，接收代理可以在 DNS 中查询发送域，看它是否有一个邮件发送政策。这种信息往往由 TXT 和 SPF 记录提供，它还能表明可以进行其他检查。例如，从 cs.washington.edu 发出的邮件永远来自主机 june.cs.washington.edu，如果发送方的邮件传输代理不是 june，那么邮件肯定有问题。

如果其中任何一项检查失败，那么邮件可能被伪造成用了一个假的发送地址。在这种情况下，应该丢弃该邮件。然而，通过检查的邮件也并不意味着就一定不是垃圾。检查只能确保该邮件似乎来自网络中它声称的区域。我们的想法是应该强制垃圾邮件发送者在发送垃圾邮件时必须使用其正确的发送地址。这样可以使得垃圾邮件在不受欢迎时更容易被识别和删除。

7.2.5 最后传递

我们的邮件几乎要被交付了。它已经抵达 Bob 邮箱。剩下的工作就是将邮件的一个副本传送到 Bob 的用户代理以便显示。这是图 7-7 体系结构中的第 3 步。这个任务很简单，在早期 Internet 时代，当用户代理和邮件传输代理作为不同的进程运行在同一台机器上时就已经完成了。邮件传输代理简单地把新邮件写到邮箱文件的结尾处，然后用户代理只需检查邮箱文件看是否添加了新邮件。

如今，运行在 PC、笔记本电脑或移动电话上的用户代理可能与运行 ISP 或公司邮件服务器的设备不是同一台机器。用户希望无论在哪里都能够远程访问他们的邮件。他们既想在工作中访问邮件，也想回家时从家用电脑上网读取邮件，甚至出差时用笔记本电脑或者度假时从所谓的网吧来访问电子邮件。他们还希望能够脱机工作，需要时重新连接 Internet 以便接收入境邮件和发送出境邮件。此外，每个用户或许要运行几个用户代理，具体使用哪个代理则取决于当时什么电脑最方便。甚至有可能在同一时间运行多个用户代理。

在这样的设置中，用户代理的工作是给出邮箱内容的一个概览，并允许用户远程对邮箱进行操纵。提供这种功能的协议有许多种，但 SMTP 不是其中之一。 SMTP 是一种“基于推”（push-based）的协议,它获取一个邮件，并且连接到远程服务器来传递邮件。邮件的最终交付不能以这种方式实现，因为两个原因：第一，邮件传输代理上的邮箱必须可以连续存储；第二，用户代理可能在 SMTP 试图中继邮件的那一刻无法连接到 Internet 上。

IMAP——Internet 邮件访问协议

最终交付邮件使用的主要协议之一是 **Internet 邮件访问协议**（IMAP，Internet Message Access Protocol）。RFC 3501 定义了 IMAP 协议版本 4。为了使用 IMAP，邮件服务器必须运行 IMAP 服务器，它负责监听端口 143。用户代理必须运行 IMAP 客户端。客户端与服务器连接，并开始发出如图 7-17 中列出的命令。

首先，如果要启用安全机制（为了加密邮件和命令）客户端必须启动一个安全传输，然后登录或以其他方式向服务器验证自己。一旦成功登录后，可以执行很多命令，包括列表显示文件夹和邮件、获取邮件或者甚至部分邮件、给邮件标记供以后删除，并将邮件组

命令	描述
CAPABILITY	列表显示服务器能力
STARTTLS	启用安全传输(TLS，参见第8章)
LOGIN	登录服务器
AUTHENTICATE	用其他方式登录服务器
SELECT	选择一个文件夹
EXAMINE	选择一个只读文件夹
CREATE	创建一个文件夹
DELETE	删除一个文件夹
RENAME	重命名一个文件夹
SUBSCRIBE	在活动集中添加文件夹
UNSUBSCRIBE	从活动集中删除文件夹
LIST	列出可用的文件夹
LSUB	列出活动的文件夹
STATUS	获取一个文件夹的状态
APPEND	往文件夹内添加一个邮件
CHECK	获取一个文件夹的断点
FETCH	从一个文件夹内获取邮件
SEARCH	在一个文件夹内搜索邮件
STORE	变更邮件标志
COPY	复制一个文件夹内的某个邮件
EXPUNGE	删除做了删除标志的邮件
UID	用唯一标识法发命令
NOOP	什么也不做
CLOSE	删除邮件并关闭文件夹
LOGOUT	登出并关闭连接

图 7-17 IMAP（版本 4）命令

织到各文件夹内。为了避免混淆，请注意，这里我们使用了术语“文件夹”（folder），以便和本节的其余部分保持一致，即一个用户有一个由多个文件夹组成的邮箱。然而，在 IMAP 规范中采用了术语“邮箱”（mailbox）来代替文件夹。因此，一个用户有许多 IMAP 邮箱，每个邮箱通常表现为提交给用户的一个文件夹。

IMAP 还有许多其他功能。它能不通过邮件号而是使用属性来寻址邮件（例如，Alice 发给我的第一个邮件）。还可以在服务器上执行搜索功能，找到满足某个特定标准的邮件，因此客户只需要获取这些邮件。

IMAP 是较早使用的最终交付协议——**邮局协议版本 3**（POP3，Post Office Protocol, version 3）的改进版。POP 3 由 RFC 1939 说明，它是个非常简单的协议，只支持较少的功能，而且其典型使用方式不太安全。它通常把邮件下载到用户代理所在的计算机上，而不是留在邮件服务器上。虽然这种处理方式使得服务器的工作更加容易，但用户难以为生。用户希望在多台不同的计算机上阅读邮件非常困难，而且如果用户代理所在的计算机发生故障，那么所有的电子邮件将可能永久地丢失。尽管如此，你仍然会发现有人还在使用 POP 3。

因为邮件服务器和用户代理之间运行的协议可以由同一家公司提供，因此也可以使用其他专有协议。Microsoft Exchange 就是一个运行其专有协议的邮件系统。

Webmail

一种日益流行可提供电子邮件服务的是 Web，它可以利用其接口来发送和接收邮件，这种方式有望替代 IMAP 和 SMTP。目前被广泛使用的 Webmail 系统包括谷歌 Gmail、微软 Hotmail 和雅虎 Mail。Webmail 是一个软件实例（在这种情况下，就是邮件用户代理），它利用 Web 为用户提供邮件服务。

在这样的体系结构中，为了接收来自端口 25 并且使用 SMTP 的用户邮件，服务提供商照常运行邮件服务器。然而，此时的用户代理与以前是不同的。它不再是作为一个独立的程序运行，而是通过网页提供了一个用户界面。这意味着用户可以使用自己喜欢的任何浏览器来访问他们的邮件，以及发送新邮件。

我们至今还没有学习 Web，但现在先给出一个关于它的简短介绍，或许以后你会回过头再看它。当用户进入服务提供商的电子邮件网页时，会看到一个表单，要求用户输入登录名和密码。然后登录名和密码被发送到服务器，在服务器端进行验证。如果登录成功，那么服务器会找出用户的邮箱，并建立一个网页，该网页列出了邮箱的大致内容；然后将该网页发送到用户的浏览器显示。

页面上显示的许多邮箱表项都是可点击的，因此邮件可以被读取、删除等。为了使界面对用户的行为有所反应，网页往往包括 JavaScript 程序。这些程序运行在本地客户端以便响应任何一个本地事件（比如鼠标点击）、在后台下载或上传邮件，或者准备下一个邮件的显示或提交一个新邮件。在这种模型中，邮件提交采用了普通的 Web 协议，把数据张贴到 URL。Web 服务器负责将邮件注入到传统的邮件传递系统中，这个我们已经在前面描述过了。出于安全方面的考虑，可使用标准的 Web 协议。这些协议本身涉及加密网页，而不管网页的内容是否是一个邮件消息或是一般网页内容。

7.3 万 维 网

Web 是万维网（World Wide Web）的俗称，它是一个体系结构框架。该框架把分布在整个 Internet 数百万台机器上的内容链接起来供人们访问。Web 刚出现时在瑞士被科研人员用来相互之间协同设计高能物理实验，仅十年间它就演变成今天被数百万人认为的“Internet”应用。它的迅速普及和流行源自这样的事实：它易于初学者使用并且提供了丰富多彩的图形界面。通过这些界面用户可以访问巨大的信息财富，内容几乎覆盖了每一个可以想到的主题，从 aardvarks（土豚）到 Zulus（祖鲁族人）什么都有。

Web 诞生于 1989 年的欧洲原子能研究中心 CERN。最初的想法是帮助大型研究组成员通过修改报告、计划、绘图、照片和其他文档的方式来进行合作，这些文档由粒子物理实验产生，并且研究组的成员通常分散在好几个国家或好几个时区。将文档链接成 Web 的提议由 CERN 物理学家 Tim Berners-Lee 提出，18 个月后第一个（基于文本的）原型系统投入运行。该系统的公开文档发表在 Hypertext' 91 会议，它立即引起了另一个研究组的注意，该研究组由伊利诺伊大学的 Marc Andreessen 领导，他最终开发了第一个图形浏览器。这就是 Mosaic 浏览器，正式发布于 1993.2。

正如他们所说的，其余的现在已经成为历史。Mosaic 是如此的受欢迎，一年后

Andreessen 离开学校组建了网景通信公司（Netscape Communications Corp.），公司目标是开发 Web 软件。接下来的 3 年，网景的 Netscape Navigator 和微软的 IE 浏览器进入了一场浏览器大战，每一个都试图捕捉这个新兴市场的更大份额，为此疯狂地加入比对手更多的功能（而且导致了更多的错误）。

从 20 世纪 90 年代到本世纪初，网站和网页（称为 Web 内容）成指数倍地增长，直到达到具有数百万计网站和数十亿网页的规模。这些网站中的一小部分盛极一时，网站和它们背后的公司主要定义了 Web，正如今天人们所体验的那样。这些公司包括书店（亚马逊于 1994 年成立，市值 500 亿美元）、跳蚤市场（易趣，1995 成立，市值 300 亿美元）、搜索（谷歌，1998 成立，市值 150 亿美元）和社会网络（Facebook，2004 成立，私人公司，价值超过 150 亿美元）。到 2000 年期间，许多 Web 公司一夜之间晋升数百万美元身价，当最后表明原来一切都只是炒作时，几乎接近破产。这一切甚至还有一个特殊名称，即所谓的点 com 时代（dot com era）。新的想法仍然丰富着 Web 世界。许多新想法来自年轻的学生。例如，当 Mark Zuckerberg 开始创建 Facebook 时是哈佛大学的学生；Sergey Brin 和 Larry Page 创建 Google 时是斯坦福大学的学生。也许你会想出下一件什么大事来。

1994 年，CERN 和 MIT 签署了建立万维网联盟（W3C，World Wide Web Consortium）的协议。W3C 是一个组织，它致力于进一步开发 Web、对协议进行标准化，并鼓励站点之间实行互操作。Berners-Lee 担任了联盟的主管。从那时起，已经有几百所大学和公司加入了该联盟。尽管现在关于 Web 的书籍数不胜数，但获取关于 Web 最新信息的最佳之处（很自然地）还是在 Web 本身。W3C 联盟的主页是 www.w3.org，感兴趣的读者可以从那里找到涵盖该联盟所有文档和活动的页面链接。

7.3.1 体系结构概述

站在用户的角度看，Web 由大量分布在全球范围的内容组成，这些内容以 **Web 页面**（Web page）或简称为**页面**（page）的形式表示。每个页面可以包含指向其他页面的**链接**（link），这些页面可以分布在全球任何地方。用户单击一个链接就可以跟随这个链接来到它所指向的页面。这个过程可无限地重复下去。让一个页面指向另一个页面的想法现在称为**超文本**（hypertext），这种想法在 1945 年由一个卓有远见的 MIT 电子工程系教授 Vannevar Bush 发明（Bush，1945），也就是说早在 Internet 被发明出来之前就已经有了 Web。事实上，它在商业计算机之前就已经存在了，虽然几所大学生产出来的粗糙原型机能填满大型会议室，而且能力还不及一个现代化的袖珍计算器。

通常观看页面的程序称为**浏览器**（browser）。Firefox、Internet Explorer 和 Chrome 是比较流行的浏览器。浏览器取回所请求的页面，对页面内容进行解释，并在屏幕上以恰当的格式显示出来。页面内容本身可能是文本、图像和格式化的命令混合体，表现的形式多种多样：可以表现成传统的文档形式，或者表现成其他内容的形式（比如视频），或者是一个能产生图形界面的程序，用户通过该界面实行与网页的交互方式。

页面的照片如图 7-18 左边一般所示。这是华盛顿大学计算机科学与工程系的一个页面。这个页面显示了文本和图形元素（大多数内容太小无法阅读）。页面中的某些部分与指向其他页面的链接有关。与另一个页面相关的一小段文字、一个图标、一个图像等都称为超链

接（hyperlink）。为了跟随一个链接，用户将鼠标光标放在页面区域的链接部分（这会使光标发生变化），然后单击链接。点击链接只是告诉浏览器去获取另一个页面的简单方式。在Web初期，链接通过下划线和彩色文本来突出强调，以便使它们脱颖而出。如今，Web页面的创作者已经有各种方法来控制链接区域的外表，因此一个链接可能会作为一个图标出现，或当鼠标滑过它时改变外观。正是页面的创造者使得链接在视觉上表现鲜明，从而提供了一个可用的接口。

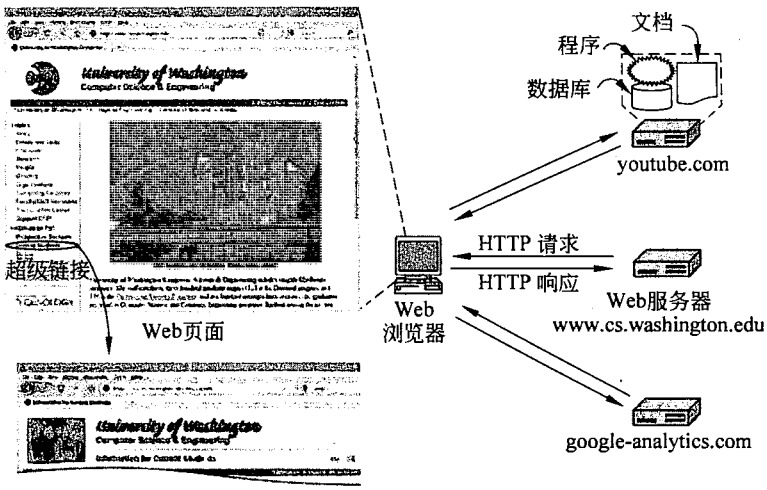


图 7-18 Web 的体系结构

系里的学生跟随一个链接到一个特别为他们而设计内容的页面，能学到更多东西。通过点击圈起来的区域就可以访问该链接。然后，浏览器抓取新的一页并显示出来，如图左下角所示的那部分。除了这个例子外，第一页还包含着指向数十个其他网页的链接。每个其他页面可以由包括在同一台机器上的页面组成（就像第一页一样），或者由位于世界各地机器上的内容组成，对此用户无从知晓。页面的抓取由浏览器完成，无须任何来自用户的帮助。因此，在观看网页的内容时，网页内容在机器之间的移动是无缝进行的。

页面显示背后的基本工作模型如图7-18所示。在这里，客户机器上的浏览器正在显示一个Web页面。每一页的抓取都是通过发送一个请求到一个或多个服务器，服务器以页面的内容作为响应。抓取网页所用的“请求-响应”协议是一个简单的基于文本协议，它运行在TCP之上，就像SMTP一样。这个协议就是所谓的**超文本传输协议**（HTTP，HyperText Transfer Protocol）。内容可能只是一个磁盘读取的文档，或者是数据库查询和程序执行的结果。如果每次显示的是相同的一个文档，则称该网页为**静态页面**（static page）。相反，如果每次显示的是程序按需产生的内容，或者页面本身包含了一个程序，则称该网页为**动态页面**（dynamic page）。

一个动态的页面每次显示时本身表现可能是不同的。例如，电子商店的首页可能对每个访问者显示的内容不尽相同。如果一个书店的顾客在过去买了一些推理小说，那么当这位顾客访问商店的主页后，可能会看到突出显示的新的惊悚小说，而另一位更喜欢烹饪的顾客可能首先映入眼帘的是新的烹饪书籍。网站如何跟踪哪位顾客喜欢什么我们马上就会说明。简单地说，其答案涉及一种Cookie。

在图中，浏览器接触三个服务器获取了两个页面，这三个服务器分别是 edu、cs.washington、youtube.com 和 google-analytics.com。来自这些不同服务器的内容集成在一起通过浏览器显示。显示细化了网页处理的范围，主要取决于什么样的内容。除了渲染文字和图形外，它可能还涉及播放一段视频，或者运行一个脚本把作为页面一部分的用户界面呈现出来。在这种情况下，cs.washington.edu 服务器提供了主网页，youtube.com 服务器提供了一段嵌入的视频，而 google-analytics.com 服务器没有提供任何用户可见的内容，但它追踪访问网站的用户。我们稍后将详细讨论跟踪器。

客户端

现在让我们来看图 7-18 中 Web 浏览器这边的详细情况。基本上，一个浏览器是一个程序，它可以显示 Web 页面并且捕获鼠标在显示页面上点击的表项。当一个表项被击中时，浏览器就跟踪相应的超链并获取被选中的页面。

当 Web 最初被建立时，为了让一个页面指向另一个 Web 页面，很显然需要某些机制来命名和定位页面。尤其是，在显示一个被选中的页面之前，首先必须回答 3 个问题：

（1）这个页面叫什么？

（2）这个页面在哪里？

（3）如何访问这个页面？

如果每个页面都以某种方式被分配了一个唯一的名字，那么在标识页面时就不会存在任何歧义。但是，问题还没有解决。考虑把人与页面作个对比。在美国，几乎每个人都有一个社会保险号，因为任何两个人都不可能拥有相同的社会保险号，因此社会保险号是一个具有唯一性的标识符。然而，如果你仅仅知道一个社会保险号，那么你是无法找到该保险号所有者的地址，当然你也无法知道到底应该用英语、西班牙语还是汉语来给他写信。Web 基本上也有同样的问题。

Web 选择的解决方案是用一种能同时解决上述 3 个问题的方式来标识页面。每个页面被分配一个**统一资源定位符**（URL，Uniform Resource Locator），用来有效地充当该页面在全球范围内的名字。URL 包括 3 个部分：协议（也称为方案（scheme））、页面所在机器的 DNS 名字，以及唯一指向特定页面的路径（通常是读取的一个文件或者运行在机器上的一个程序）。一般情况下，路径是一个模仿文件目录结构的层次名字。然而，如何解释路径是服务器的事，而且路径可能反映了实际的目录结构，也可能不反映实际的目录结构。

作为一个例子，图 7-18 显示的页面的 URL 是：

```
http://www.cs.washington.edu/index.html
```

这个 URL 由 3 部分组成：协议（http）、主机的 DNS 域名（www.cs.washington.edu）和路径名（index.html）。

当用户点击一个超链，浏览器就执行一系列的步骤来获取该超链指向的网页。让我们跟踪点击例子中链接时所发生的步骤：

（1）浏览器确定 URL（通过观察选中的什么）。

（2）浏览器请求 DNS 查询服务器 www.cs.washington.edu 的 IP 地址。

（3）DNS 返回 128.208.3.88。

（4）浏览器与128.208.3.88机器的80端口建立一个TCP连接，80端口是HTTP协议的知名端口。

（5）浏览器发送HTTP报文，请求/index.html页面。

（6）www.cs.washington.edu服务器发回页面作为HTTP响应，例如发送文件/index.html。

（7）如果该页面包括需要显示的URL，那么浏览器经过同样的处理过程获取其他URL。在这种情况下，URL包括多个取自www.cs.washington.edu的内嵌图像、一个取自youtube.com的内嵌视频和一个取自google-analytics.com的脚本。

（8）浏览器显示页面/index.html，如图7-18所示。

（9）如果短期内没有向同一个服务器发出其他请求，那么释放TCP连接。

许多浏览器会在屏幕底部的状态栏中显示它们目前正在执行哪一步。这样，当性能较差时，用户可以看到是由于DNS没有响应、服务器没有响应，或者干脆正在一个缓慢或拥塞的网络上传输页面。

URL设计是开放式的，在某种意义上它很简单，允许浏览器使用多种协议去获得各种不同的资源。事实上，已经定义了针对各种其他协议的URL。图7-19列出了稍微简化了的常见形式。

名字	用途	实例
http	超文本(HTML)	http://www.ee.uwa.edu/~rob/
https	安全的超文本	https://www.bank.com/accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	本地文件	file:///usr/suzanne/prog.c
mailto	发送邮件	mailto:JohnUser@acm.org
rtsp	流式媒体	rtsp://youtube.com/montypython.mpg
sip	多媒体呼叫	sip:eve@adversary.com
about	浏览器信息	about:plugins

图7-19 某些公共的URL方案

让我们简单地浏览上述列表。http协议是Web的母语，即Web服务器讲的语言。HTTP代表**超文本传输协议**（HyperText Transfer Protocol）。我们将在本节后面详细讨论它。

ftp协议用于通过FTP访问文件，这是Internet的文件传输协议。FTP早在Web之前就有，已经被使用超过30年。通过提供一个简单的可点击界面而不是一个命令行界面，Web使得用户更加易于获得存放在世界各地众多FTP服务器上的文件。这种获取信息的方式改进是蔚为壮观Web增长的原因之一。

使用file协议或者更简单地只要给出一个文件名就有可能通过Web页面来访问本地文件。这种方法并不需要一个服务器。当然，它仅适用于本地文件，而不能用于远程文件。

mailto协议并没有真正抓取网页，但反正是有用的。它允许用户从Web浏览器发送电子邮件。大多数浏览器针对一个mailto链接会以启动用户的邮件代理作为回应，返回一个已填写好地址字段的邮件。

rtsp和sip协议用于创建流式媒体会话和音视频的呼叫。

最后，about协议是一种惯常方式，主要用来提供有关浏览器的信息。例如，about: plugins链接会导致大部分浏览器显示一个网页，该网页列出它们利用浏览器扩展可以处理的

MIME 类型，这种浏览器扩展称为**插件**（plugin）。

总之，URL 的设计不仅允许用户浏览网页，而且允许用户运行一些旧的协议，比如 FTP 和电子邮件，以及涉及音频和视频的新协议；除此之外，还提供了访问本地文件和浏览器信息的便利方法。这种方法不需要专门为其他服务设计特殊的用户界面程序，而是把几乎所有的 Internet 访问集成到单一的程序：Web 浏览器。如果这种想法不是由一个在瑞士研究实验室工作的英国物理学家发明的，那么它可能很容易得到一些软件公司广告的梦想计划的支持。

抛开所有这些漂亮的属性，越来越多的 Web 使用已经暴露出 URL 方案中的固有弱点。一个 URL 指向一个特定的主机，但有时引用一个页面而无须告诉它在哪里也是很有用的。例如，对于被大量引用的页面，需要有多个相距甚远的副本，以便减少网络流量。但用户没有办法说："我希望得到页面 xyz，但我不关心在哪里得到它。"

为了解决这类问题，URL 已经被推广到**统一资源标识符**（URI，Uniform Resource Identifiers）。某些 URI 告诉浏览器如何定位资源，这些就是 URL；其他 URI 告诉了资源名字，但没有说明到哪里可以找得到它，这些 URI 就称为**统一资源名**（URN，Uniform Resource Names）。如何书写 URI 的规则由 RFC 3986 给出说明，而不同 URI 方案的使用则由 IANA 负责跟踪。除了在图 7-19 列出的方案外，还有许多不同类型的 URI，但这些图 7-19 列出的这些方案在今天 Web 的使用中占据了主导地位。

MIME 类型

为了能够显示新页面（或任何页面），浏览器必须了解其格式。为了让所有的浏览器都了解所有网页，网页必须以一种标准化的语言编写，这个语言就称为 HTML。它是 Web 的通用语（现在）。我们将在本章后面详细讨论该语言。

虽然浏览器基本上是一个 HTML 解释器，但大多数的浏览器有众多的按钮和功能，帮助用户更容易地浏览网页。大多数浏览器有一个返回到前一页、前进到下一页的按钮（只有当用户操作了回退动作后），和一个直接通往用户首选页的按钮。大多数浏览器还有一个按钮或菜单项用来在给定页面上设置书签，而且还提供了另一个显示书签列表的按钮或菜单项，这种方式下只需要点击几下鼠标就有可能重新审视书签。

正如我们的例子所显示的那样，HTML 页面包含了丰富的内容元素，而不只是简单的文本和超文本。为了增加一般性，并不是所有的页面都需要包含 HTML。一个页面可能由一段 MPEG 格式的视频、一个 PDF 格式的文件、一张 JPEG 格式的照片、一首 MP3 格式的歌曲，或任何数百种其他文件类型的内容组成。由于标准的 HTML 页面可以链接到任何一种格式的内容，因此当浏览器命中一个它不知道如何解释的页面时，问题就产生了。

不是把浏览器越做越大，使其内置的解释器能适应快速增长的文件类型，相反大多数浏览器都选择了一个更为一般性的解决方案。当一台服务器返回一个页面时，它同时也返回了一些关于此页面的其他信息。这些信息包括页面的 MIME 类型（见图 7-13）。具有 text / html 类型的页面可被直接显示，就像其他一些内置类型的网页一样。如果 MIME 类型不是一个内置的类型，那么浏览器查阅其 MIME 类型表，以便确定如何显示该页面。此表将 MIME 类型与观众关联。

有两种可能的方式：插件和辅助应用程序。**插件**（plug-in）是一个第三方代码模块，

作为扩展被安装到浏览器中，如图 7-20(a)所示。常用的插件例子有 PDF、Flash 和 Quicktime，主要用来呈现文档和播放音视频。由于插件运行在浏览器内部，因此它们可以访问当前的页面，也可以修改页面的外观。

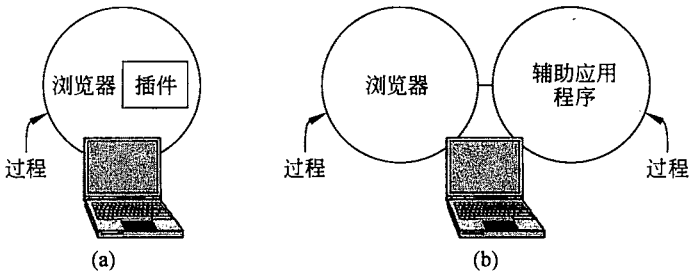


图 7-20
(a) 浏览器插件；(b) 辅助应用程序

每种浏览器都有一组过程，所有的插件必须实现这些过程，这样浏览器才可以调用插件。例如，通常浏览器的基本代码会调用一个专门过程，将待显示的数据传递给插件。这组过程就是插件的接口，它与特定的浏览器相关。

另外，浏览器也为插件提供了一组它自己的过程，以便向插件提供服务。浏览器接口较为典型的过程包括申请和释放内存、在浏览器的状态栏上显示一条消息，以及向浏览器查询有关的参数。

在一个插件被使用以前，首先必须安装该插件。一般的安装过程是，用户到插件的 Web 站点下载一个安装文件。执行安装文件将插件解压缩，并且执行适当的调用以便注册该插件的 MIME 类型，并将浏览器与该插件关联起来。浏览器通常预载了一些流行的插件。

另一种扩展浏览器的方式是使用一个**辅助应用程序**（helper application）。这是一个作为独立进程运行的完整程序，如图 7-20(b)所示。因为辅助应用程序是独立运行的程序，因此接口与浏览器非常靠近。它通常只是接受一个存储了待显示内容的临时文件名字，然后打开该文件并显示其内容。典型地，辅助应用程序是一些独立于浏览器而存在的大型程序，比如 Microsoft Word 或者 PowerPoint。

许多辅助应用程序使用 MIME 的 application 类型。因此，目前已经定义了相当多的子类型，例如，application/vnd.ms-powerpoint 表示 PowerPoint 文件。Vnd 代表特定于开发商的格式。通过这种方式，URL 就可以直接指向一个 PowerPoint 文件，并且当用户单击它时，浏览器自动启动 PowerPoint，并将待显示的内容传递给它。辅助应用程序并不受限于只能使用 MIME 的 application 类型。例如，image/x-photoshop 代表 Adobe Photoshop。

因此，可以将浏览器配置成能处理几乎无限多种文档的类型，而且无须改变浏览器本身。现代 Web 服务器常常被配置成具有数百种“类型/子类型”的组合，每当安装一个新程序时，新的类型/子类型组合也随之被加入进来。

针对一些子类型，比如 video/mpeg，当多个插件和辅助应用程序都适用的时候，就会产生冲突。当注册的最后一个覆盖掉已有与 MIME 类型关联的插件或者辅助应用程序，从而为自己捕获类型时，会发生什么呢？因此，安装一个新的程序可能会改变浏览器处理现有类型的方式。

浏览器也能够打开本地的文件，而不一定非得从远程的 Web 服务器上取回文件，就像

没有网络一样。然而，浏览器需要某种方式来确定文件的MIME类型。标准的方式是操作系统将文件的扩展与MIME类型关联起来。在典型的配置中，当浏览器打开foo.pdf时，它就会用application/pdf插件，而当打开bar.doc文件时则通过application/msword辅助应用程序调用Word。

同样地，由于许多程序都希望（实际上是渴望）处理某些类型的文件，比如说mpg，所以这里还可能发生冲突。在安装过程中，供经验丰富用户使用的程序常常会针对它们处理的MIME类型和扩展名，给用户显示一些复选框，并且允许用户选择合适的程序，这样就不会无意中覆盖掉已有的关联关系。面向消费者市场的程序则假设用户根本不知道什么是MIME类型，它们只是尽可能地抓取到所有的类型，而根本不关心以前安装的程序做过什么。

能够将浏览器扩展成拥有处理大量新类型的能力确实非常方便，但这也导致了麻烦。当Windows个人计算机上的浏览器抓取了一个扩展名为exe的文件时，它知道这个文件是可执行程序，因而没有对应的辅助应用程序。很显然它应该运行这个程序，然而，这可能是一个巨大的安全漏洞。一个恶意的Web站点所需做的全部事情只是生成一个包含许多图片（比如电影明星或体育明星）的Web页面，并且将所有的图片都链接到一个病毒上。于是，单击任何一幅图片都会导致取回一个未知的、可能恶意的可执行程序，并且在用户的机器上运行。为了预防这样的不速之客入侵，Firefox和其他浏览器配置成特别小心地自动运行未知程序，但是，并不是所有的用户都懂得什么样的配置才是安全的而不是便利的。

服务器端

关于客户端的介绍已经很多了，现在让我们来看看服务器端。正如我们在前面所看到的，当用户键入一个URL或者单击一行超文本时，浏览器会解析URL，并且将http://和下一个斜线之间的那部分解释成待查找的DNS名字。有了服务器的IP地址以后，浏览器与该服务器的端口80建立一个TCP连接；然后它发送一条命令，其中包含了URL的剩余部分，即该服务器上某个页面的路径；最后服务器返回该页面供浏览器显示。

乍一看，Web服务器与图6-6中的服务器非常类似。该图中的服务器得到一个待查找的文件名字，并且通过网络返回结果。在这两种情况下，服务器在它的主循环中执行如下步骤：

（1）接受来自客户端（浏览器）的TCP连接。
（2）获取页面的路径，即被请求文件的名字。
（3）获取文件（从磁盘上）。
（4）将文件内容发送给客户。
（5）释放该TCP连接。

现代的Web服务器具有更多的功能，但本质上这就是Web服务器在最简单情况下所做的工作，即获取一个包含网页内容的文件。对于动态内容，第三步必须替换成运行一个程序（由路径确定），该程序能返回网页的内容。

然而，为了单位时间处理多个请求，Web服务器的实现具有不同的设计。简单设计的一个问题是文件访问通常成为瓶颈。相对程序执行而言，读磁盘的速度很慢，而且通过操作系统调用重复读取相同的磁盘文件。另一个问题是一次只能处理一个请求。文件或许会

很大，当系统在传输该文件时其他请求都被阻塞了。

一种显而易见的改进方法（所有的 Web 服务器都采用了这种方法）是在内存中维护一个缓存，其中保存着 n 个最近使用过的文件或者数千兆量的内容。服务器在从磁盘读取文件之前，首先检查缓存。如果缓存中有该文件，则直接从内存中取出文件，从而消除了访问磁盘的时间。虽然真正有效的缓存需要大量的内存，以及一定的额外处理时间来完成检查缓存和管理其内容，但是，节省下来的时间几乎总是抵得上这些开销和费用。

为了解决一次只能服务一个请求的问题，一种策略是将服务器设计成**多线程模式**（multithreaded）。在其中一种设计方案中，服务器由一个**前端模块**（front-end module）和 k 个**处理模块**（processing module）组成，如图 7-21 所示。前端模块接受所有入境请求；k+1 个线程全部属于同一个进程，这样所有处理模块都可以访问当前进程地址空间中的缓存。当一个请求到达时，前端模块接受它，并为其创建一条描述该请求的简短记录，然后将该记录递交给其中一个处理模块。

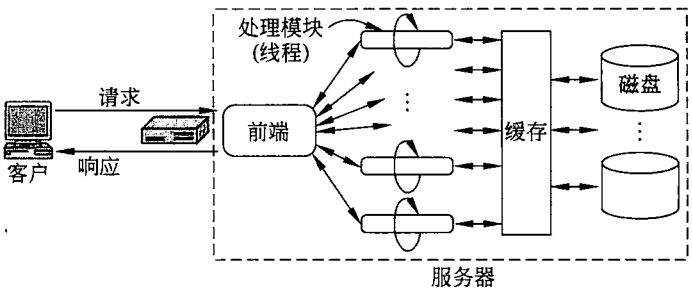


图 7-21 具有一个前端和若干个处理模块的多线程 Web 服务器

处理模块首先检查缓存，查看其中是否有所需的文件。如果缓存中有该文件，则处理模块修改记录，在记录中增加一个指向该文件的指针；如果缓存中没有该文件，则处理模块执行一次磁盘操作将该文件读入缓存（可能要丢弃其他一些缓存的文件，以便腾出空间）。从磁盘上读取文件后，将该文件放入缓存，同时把它发送给客户。

这种方案的优点是当一个或多个处理模块因为等待磁盘操作或者网络操作的完成而被阻塞时（因此不占用 CPU 时间），其他模块可以继续处理其他的请求。有了 k 个处理模块，吞吐量可以达到单线程服务器下的 k 倍。当然，当磁盘或者网络成为受限因素时，必须使用多个磁盘或者更快的网络才能获得比单线程模式实质性的性能提高。

现代 Web 服务器所做的不只是接受文件名和返回文件。事实上，每个请求的实际处理过程可能非常复杂。由于这个原因，在许多服务器中，每个处理模块要执行一系列步骤。前端将每个入境请求传递给第一个可用模块，然后该模块根据这个特定请求的需要，执行下列步骤中的某个子集。这些步骤发生在 TCP 连接和任何安全传输机制（例如 SSL/TLS，将在第 8 章介绍）建立之后。

（1）解析被请求的 Web 页面的名字。

（2）执行对该页面的访问控制。

（3）检查缓存。

（4）从磁盘上获取请求的页面或者运行一个创建页面的程序。

（5）确定响应中的其余部分（比如要发送的 MIME 类型）。

（6）把响应返回给客户。

（7）在服务器的日志中增加一个表项。

第（1）步是必需的，因为入境请求中可能并没有包含文件或者程序的字面意义上的实际名字。它或许包含了需要转换的内置缩写。作为一个简单例子，在URL http://www.cs.vu.nl中的文件名是空的。在这种情况下，有必要将它扩展到某个默认的文件名，通常是index.html。另一个常用的规则是将~user/映射到user的web目录。这些规则可以联合使用。因此本书作者之一（AST）的主页可以通过下面的URL到达：

http://www.cs.vu.nl/~ast/

即使在特定默认目录下的实际文件名是index.html。

同样地，现代浏览器可以指定配置信息，比如浏览器软件和用户的默认语言（例如意大利语或英语）。这样，如果可能，服务器就能为移动设备选择具有小图片和恰当语言的Web页面。一般来说，文件名的扩展并不像它首次出现时那样微不足道，因为对于如何把路径映射到文件目录和程序有各种各样的惯例。

第（2）步检查是否存在与该页面关联的任何访问限制。并非所有的网页都向公众开放。确定客户端是否有权限获取一个页面可能依赖于客户端的身份（例如，给出用户名和密码），或者客户端在DNS或IP地址空间的位置。例如，一个页面可能被限制于只向公司内部的员工开放。如何实现这一点则取决于服务器的设计。例如，对于流行的Apache服务器，惯常的做法是把后缀名为.htaccess 的文件放置在被限制访问的页面所在的目录，该文件列出了对页面的访问限制。

第（3）步和第（4）步涉及页面的获取。是否可以从缓存中获取页面取决于处理规则。例如，因为程序每次运行都可能产生不同的结果，因此由正在运行的程序创建的页面并不总是可以被缓存。甚至文件也应该偶尔被检查一下，看看它们的内容是否已经发生改变，以便旧的内容从缓存中删除。如果页面需要运行一个程序，那么还存在一个设置程序参数或输入的问题。这些数据来自请求的路径或请求中的其他部分。

第（5）步确定响应页面内容时有关的其他部分。MIME类型就是一个例子。它可能来自文件的扩展名、一个文件的前几个字或程序的输出、一个配置文件，以及其他可能的来源。

第（6）步通过网络返回页面。为了提高性能，一个TCP连接可以被客户端和服务器用来获取多个页面。这种连接的重用性意味着需要一些逻辑，将一个请求映射到一个共享的连接并且返回请求的每个响应，以便它与正确的请求关联。

第（7）步是为了行政管理的需要在系统日志中增加一个表项反映此次访问，同时还记录任何其他重要的统计数据。这种日志可以在日后被用来挖掘出有关用户行为的有价值信息，例如，人们访问网页的次序。

Cookie

到目前为止，正如我们所描述的那样，浏览网站只涉及一系列独立页面的获取。完全没有登录会话的概念。浏览器发送一个请求给服务器，并获得返回的文件。然后，服务器彻底忘记它已经看到过哪些特定的客户。

这种模式对于检索公开可用的文件资源完全足够，而且在Web创造初期运作良好。然而，它不适合为不同的用户返回不同的页面，应该给用户返回什么页面取决于他们之前在

服务器上做了什么。这样的行为需要与网站进行多次交互。例如，某些网站（例如报纸）要求客户注册（并且有可能要付费）才能使用。这就提出了一个问题，服务器如何区分来自以前注册过的用户的请求还是来自没有注册过的其他人的请求？第二个例子是电子商务。如果用户在电子商店漫步，不时地折腾放到虚拟购物车中的项目，服务器如何保持跟踪其购物车的内容？第三个例子是定制的门户网站，比如雅虎。用户可以设置其个性化的初始页面，该页面只给出他们想要的信息（例如，他们的股票和喜爱的运动队），但是，如果服务器不知道用户是谁，它该如何显示正确的页面？

乍一看，读者可能会立即想到服务器可以通过观察用户的 IP 地址来跟踪他们。然而，这个想法并不奏效。首先，许多用户通过共享的计算机工作，特别是在家里；而且 IP 地址仅仅标识了计算机而不标识用户。其次，更糟糕的是，许多公司使用 NAT，因而所有用户发出的出境数据包具有相同的 IP 地址。也就是说，所有 NAT 盒子后面的计算机从服务器的角度来看是同一台机器。

这个问题的解决方案是采用了一项称为小甜饼（Cookie，它是一小段文本信息，伴随着用户请求页面在 Web 服务器和浏览器之间传递）的备受批评的技术。这个名称来自于很早以前的程序员行话，即程序调用一个过程并得到一些信息，这些信息稍后在完成某些工作时可能需要用得到。从这种意义上来说，UNIX 文件描述符或 Windows 对象句柄都可以被看作是一种 Cookie。Cookie 首次实现在 1994 年 Netscape 浏览器中，后来在 RFC 2109 中对此作了正式定义。

当客户请求一个 Web 页面时，服务器除了提供所请求的页面以外，还以 Cookie 的形式提供了一些附加的信息。Cookie 是一个相当小的命名的串（最多 4 KB），服务器将它与浏览器关联。这种关联与用户关联不一样，但它非常接近而且比 IP 地址更有用。浏览器把服务器所提供的 Cookie 通常存储在客户机磁盘 Cookie 目录下一段时间，这样在整个浏览器调用期间一直坚持 Cookie，除非用户禁用 Cookie。Cookie 只是字符串，而不是可执行程序。原则上，一个 Cookie 可能包含病毒，但由于 Cookie 只被当作数据处理，因而不存在病毒得以实际运行从而造成损害的正式途径。然而，始终有可能存在一些黑客利用一个浏览器的漏洞，来激活病毒。

一个 Cookie 可能包含至多 5 个字段，如图 7-22 所示。域（Domain）指出 Cookie 来自何方。浏览器应该检查服务器没有谎报它们的域名。每个域为每个客户端应该存储不超过 20 个 Cookie。路径（Path）是服务器目录结构中的一个路径，它标识了服务器文件树的哪些部分可能使用该 Cookie。路径通常是/，这意味着整棵树。

域	路径	内容	过期	安全
toms-casino.com	/	CustomerID=297793521	15-10-1017:00	是
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-1114:22	不
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-2023:59	不
sneaky.com	/	UserID=4627239101	31-12-19 23:59	不

图 7-22 Cookie 的某些例子

内容（Content）字段采用“名字＝值”的形式。这里名字和值都可以是服务器期望的任何内容。这个字段正是存放 Cookie 内容的地方。

过期时间（Expires）字段指定了该 Cookie 何时过期。如果这个字段不存在，则浏览器

在退出时将丢弃Cookie，这样的Cookie称为非持续Cookie（nonpersistent cookie）；如果针对某个Cookie提供了时间和日期，那么这样的Cookie称为持续Cookie（persistent cookie），它被一直保存直至过期为止。过期时间采用了格林尼治标准时间。为了从客户的硬盘上删除一个Cookie，服务器只需将它再发送一次，但是将该Cookie的过期时间指定为一个已经过去的时间即可。

最后，通过设置安全（Secure）字段指示浏览器只向使用安全传输连接的服务器返回Cookie，所谓的安全传输就是SSL/TLS（我们将在第8章描述）。这个功能可用于电子商务、银行和其他安全类应用。

我们现在已经看到了如何获得Cookie，但如何使用Cookie呢？在浏览器向某个Web站点发出一个页面请求之前，浏览器检查它的Cookie目录，确定这个请求前往的目标域是否在当前客户端放置了Cookie。如果存在相应的Cookie，则该域放置的所有Cookie都被包含到请求消息中。服务器得到了这些Cookie以后，就可以按它所期望的方式来解释它们。

我们现在来看Cookie的一些可能用法。在图7-22中，第一个Cookie由toms-casino.com设置，并且被用来标识一个顾客。当该顾客下个星期登录进去花掉了一些钱时，浏览器将Cookie发送过去，因此服务器就知道他是谁了。有了顾客的ID，服务器就可以在数据库中查找该顾客的消费记录，并利用这些记录信息来创建一个合适的Web页面显示给该顾客。如果已知该顾客爱好赌博，则为其显示的页面可能由扑克游戏、今日赛马比赛名单，或老虎机组成。

第二个Cookie来自joes-store.com。这里的场景是顾客在商店里闲逛，寻找想购买的好东西。当她发现一件便宜货并单击它时，服务器将该货物放入她的购物车（由服务器维护），同时创建一个包含商品数量和产品代码的Cookie，并把它发送给客户。当客户继续在商店里闲逛时，每当客户点击一个新页面，该Cookie随着该新页面的请求被返回到服务器端。随着顾客购买的物品累计得越来越多，服务器将它们统统加入到Cookie中。最后，当客户单击“前往结账”（PROCEED TO CHECKOUT）时，这个Cookie现在包含了她想购买商品的完整列表，它和当前的请求一起被发送给服务器。通过这种方式，服务器就能够精确地知道顾客想购买哪些东西。

第三个Cookie用于Web门户站点。当用户单击一个指向该门户的链接时，浏览器将Cookie发送过去。它告诉该门户站点创建一个页面，包含Cisco公司和Oracle公司的股票价格，以及纽约喷气机队（New York Jets）橄榄球赛的结果。由于Cookie最多可以达到4 KB长，所以还有足够的空间来存放关于报纸头条新闻、本地天气、特价信息等更多用户喜好的信息。这样网站经营者可以了解用户如何浏览他们的网站，广告商为一个特定用户建立浏览的广告或网站的概况。备受争议的是用户通常不知道他们的活动被跟踪了，即使详细的资料和跨越看似与网站无关。尽管如此，**Web跟踪**（Web tracking）是一项大生意。提供并跟踪广告的DoubleClick跻身于世界上最繁忙的前100网站，这是由Web监测公司Alexa作出的排名。Google Analytics为运营商跟踪站点的使用情况，它已被世界上最繁忙的10万个网站中一半以上的网站所使用。

利用Cookie，服务器很容易地就能跟踪用户的活动。假设一个服务器想知道它有多少个只来过一次的访问者，以及每个人在离开该站点前看过多少页面。当第一个请求到来时，它不会随带任何Cookie，因此服务器返回一个包含“Counter=1”的Cookie。用户浏览本

站点的后续网页将会把这个 Cookie 送回给服务器。服务器每次递增该计数器并送回给客户。通过跟踪这个计数器，服务器就可以知道有多少人在看了第一个页面后就离开了，有多少人看了第二个页面才离开，等等。

跟踪用户跨越站点的浏览行为仅仅稍微复杂一点。其工作过程如下所述。一家广告代理商，比如说 Sneaky 广告，它联系了一些主要的 Web 站点，在这些站点的页面上放置一些广告来宣传它企业客户的产品，为此它当然向这些站点的所有者支付一定的费用。广告公司并不是向这些站点提供一个放在每个页面上的 GIF 或 JPEG 文件，而是提供一个 URL，这个 URL 被加入到每个页面中。广告公司给出的每个 URL 包含一个路径上的唯一数字，例如

http://www.sneaky.com/382674902342.gif

当用户第一次访问一个包含该广告的页面 P 时，浏览器获取 HTML 文件。然后浏览器检查该 HTML 文件，并看到了指向 www.sneaky.com 上图像文件的链接，因此它向 www.sneaky.com 发送一个对该图像的请求。一个包含广告的 GIF 文件，连同一个包含了唯一用户 ID（即图 7-22 中 4627239101）的 Cookie，一起被返回给客户。Sneaky 公司记录下这样的事实：此 ID 的用户曾经访问过页面 P。要做到这点很容易，因为被请求的路径（382674902342.gif）只在页面 P 上被引用过。当然，实际的广告可能出现在成千上万个页面，但每次都有一个不同的文件名。Sneaky 公司也许在每次发布广告时从产品制造商那里获得几美分的收益。

稍后，当这个用户访问另一个包含 Sneaky 广告的 Web 页面时，浏览器首先从服务器获回 HTML 文件，然后它看到页面上有一个指向 http://www.sneaky.com/193654919923.gif 的链接，于是请求该文件。由于浏览器已经有一个来自 sneaky.com 域的 Cookie，于是它将这个含有用户 ID 的 Sneaky Cookie 也包含在请求中。Sneaky 现在知道该用户已经访问了第二个页面。

在适当的时候，即使用户从未点击过任何广告，Sneaky 也能够建立起有关该用户浏览习惯的详细轮廓。当然，它还没有用户的名字（虽然它有用户的 IP 地址，也许根据其他数据库信息就足以推断出用户的名字）。然而，如果该用户曾经将他的名字提供给任何一个与 Sneaky 合作的站点，那么现在 Sneaky 就拥有了一份带有用户名字的完整轮廓信息，它可以卖给任何一个有意购买此类信息的人。出售这些信息可能会给 Sneaky 带来足够的利润，使它能在更多的 Web 站点上放置更多的广告，从而收集更多的信息。

如果 Sneaky 想要做得更加偷偷地不为人知，则广告不必做成传统的横幅广告。一个由背景颜色单像素组成的“广告”（因此是不可见的）与横幅广告具有完全相同的效果：它要求浏览器取回这个 1×1 像素的 gif 图像，并将该像素所属域的所有 Cookie 发送给服务器。

Cookie 已经成为是否侵害用户在线隐私的一个辩论焦点，因为像上面这样的跟踪行为侵犯了用户的在线隐私。在整个商业中最阴险的环节在于许多用户完全不知道自己的信息被他人收集，甚至可能认为自己是安全的，因为他们没有点击任何广告。出于这个原因，跨站点跟踪用户的 Cookie 被许多人认为是**间谍软件**（spyware）。看看你的浏览器中是否早就已经存储了 Cookie。大多数浏览器会显示此信息，同时还会显示当前的隐私偏好设置。你或许会惊讶地发现姓名、电子邮件地址或密码以及不透明的标识符。希望你不会发现信用卡号码，但 Cookie 被滥用的潜在性非常明显。

为了维护外表上的隐私，有些用户将他们的浏览器配置成拒绝所有的Cookie。然而，这会带来一些问题，因为许多Web站点没有Cookie无法正常工作。另外，大多数浏览器允许用户阻止第三方Cookie。**第三方Cookie**（third-party cookie）是从一个不同于主页网站的其他网站获得的Cookie，例如，sneaky.com Cookie就是在与一个完全不同网站的P页面交互时所用的。阻止这些Cookie有助于防止跨网站的跟踪。安装浏览器的扩展也可以提供细粒度的控制如何使用Cookie（或者说，不使用Cookie）。随着辩论的继续，许多公司正在制定有关隐私政策，限制它们如何共享信息，以便防止用户信息被滥用。当然，政策只是公司简单地说它们将如何处理信息。例如："我们可能会使用从你那里收集到的信息来指导我们的业务"——这或许正是销售信息。

7.3.2 静态Web页面

Web的基础是将Web页面从服务器传输到客户端。在最简单的形式中，Web页面是静态的。也就是说，它们就是存放于服务器上的文件，每次被客户端获取和显示表现的都是一样的方式。然而，仅仅因为它们是静态的，并不意味着页面在浏览器端是呆滞的。包含一段视频的页面也可以是一个静态Web页面。

正如前面提到的，在这个Web中，大多数页面以HTML语言来编写。教师的网页通常是静态的HTML页面，而公司的网页则通常是由一个网页设计公司制作的动态页面。在本节中，我们将简要介绍静态HTML页面，这是以后学习的基础。如果读者早已熟悉HTML，可以直接跳到下一节，在那里我们描述了动态网页内容和Web服务。

HTML——超文本标记语言

Web引入了一种称为**超文本标记语言**（HTML，HyperText Markup Language）的语言。HTML允许用户生成一个包含了文本、图形和指向其他Web页面指针的Web页面。HTML是一种标记语言，或一种描述了如何格式化文档的语言。"标记"这个术语来自过去，那时编辑在要印刷的文档上实际标记出有关信息，告诉印刷工（那时还是人工印刷）应该使用什么字体等。因此，标记语言包含了格式化的显式命令。例如，在HTML中，意味着粗体字型的开始，表示粗体字型的结束。LaTeX和TeX是标记语言的其他实例，这些语言为大多数学术作者所知所用。

相对于其他无显式标记的语言来说，标记语言的关键优点是将内容与其应该如何表示相分离。如此一来，编写浏览器就非常简单：浏览器只需理解标记命令并将这些命令应用于内容即可。在每个HTML文件中嵌入所有的标记命令，并且标准化这些标记命令，使得任何一个Web浏览器都可以读取任何Web页面，并对页面重新格式化。这点非常关键，因为一个页面有可能是在一台具有1600×1200大小24位颜色窗口的高端计算机上被设计出来的，但它必须能在一个移动电话只有640×320大小8位颜色的窗口中显示。

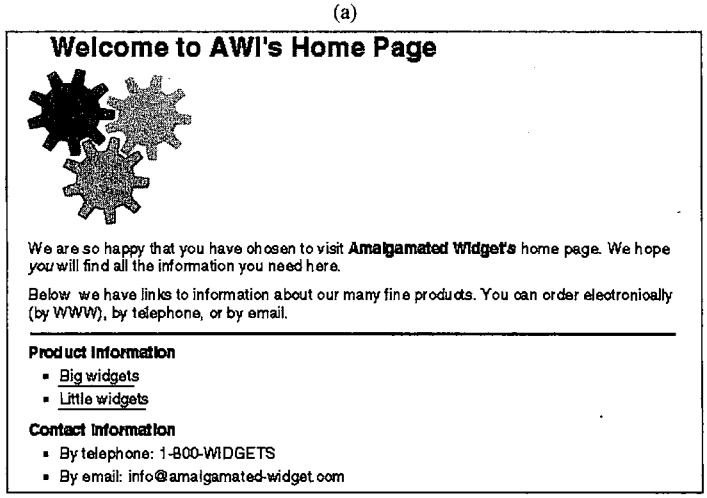
尽管可以用任何一个纯文本编辑器来编写这样的文档，而且确实有许多人也是这样做的，但是，也可以使用文字处理器或者专用的HTML编辑器来完成大部分工作（但是，相应地给予用户对最终结果的所有细节的控制能力也有所减弱）。

图7-23给出了一个以HTML编写的简单Web页面及其在浏览器上的显示。一个Web页面由一个头部和一个主体组成，它们都被包括在<html>和</html>标签（tag）之间，这些

标签就是格式化命令；虽然缺少这两个标签大多数浏览器也不会出错。正如图 7-23(a)所示，头部由<head>和</head>标签括起来，而主体部分则由<body>和</body>标签括起来。标签中的字符串称为指示符（directive）。大多数 HTML 标签（并不是全部）都采用这种格式。即用<something>标注 something 的开始，而用</something>标注 something 的结束。

```
<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by email. </p>
<hr>
<h2> Product information </h2>
<ul>
    <li> <a href="http://widget.com/products/big"> Big widgets </a> </li>
    <li> <a href="http://widget.com/products/little"> Little widgets </a> </li>
</ul>
<h2> Contact information </h2>
<ul>
    <li> By telephone: 1-800-WIDGETS </li>
    <li> By email: info@amalgamated-widget.com </li>
</ul>
</body>
</html>
```

(a)



(b)

图 7-23

(a) 一个 Web 页面的 HTML；(b) 格式化后的页面

标签既可以是小写也可以是大写。因此<head>和<HEAD>含义相同，但为了兼容，最好采用小写。HTML 文档的实际布局结构无关紧要。HTML 解析器将忽略额外的空格和回

车，因为它们必须重新对文本进行格式化，以便使得这些文本适合当前的显示区域。因此，可以任意添加空格使得 HTML 文档更具可读性，而大部分空格并不是所需要的。另外一个后果是空行无法真正划分出单独一段，因为它们在显示时被忽略掉了。因此需要一个显式的标签来分段。

有些标签带有（命名的）参数，这些参数称为属性（attribute）。例如，在图 7-23 中，标签<img>被用来在文字里内嵌一个图片。它有两个属性，分别是 src 和 alt。第一个属性给出了图片的 URL。 HTML 标准没有指定只能允许什么样的图像格式。实际上，所有的浏览器都支持 GIF 和 JPEG 文件。浏览器自由支持其他格式的图片，但这种扩展是一把双刃剑。如果用户习惯于支持 TIFF 文件的浏览器，他有可能在他的 Web 页面内包括这些格式，但稍后他会惊讶地发现其他浏览器根本忽略了他的奇妙艺术。

第二个属性给出了在图像无法显示时的替代文字。对于每个标签，HTML 标准给出了一个允许的参数列表，包括参数及其含义。由于每个参数都有名字，因而给定参数的出现顺序就不重要了。

从技术上讲，HTML 文档是用 ISO 8859-1 Latin-1 字符集编写的，但是如果用户的键盘只支持 ASCII 字符，则需要用转义序列来代表特殊的字符，比如 è。标准给出了特殊字符的列表。它们都以“&”符号作为开头，以“;”作为结束。例如， 表示空格、è表示 è、é表示 é。由于<、>和 &有特殊的含义，所以它们只能用转义序列来表示，分别是<、>和 &。

头部中的主要内容是标题（title），它由<title>和</title>来定界。某种特定的元信息也可以出现在这里，虽然在我们的例子中没有给出来。标题本身不显示在页面上。有些浏览器用它来标记页面的窗口。

图 7-23 使用了几种头。每个头由<hn>标签生成，其中 n 是一个 1 到 6 之间的数字。因此，<h1>是最重要的标题；<h6>是最不重要的标题。如何在屏幕上突出这些标题是浏览器的责任。通常较小数字的标题会采用较大和较重的字体来显示。浏览器也可能为每种级别的标题选择使用不同的颜色。一般来说，<h1>标题的字体较大较粗，并且在标题的上方和下方至少有一个空行。相对地，<h2>标题较小，且上方和下方只有较少的空白。

标签<b> 和 <i>分别用来表示进入粗体和斜体模式。<hr>标签强制中断，并画一条横跨屏幕的水平线。

<p>标签开始一段。例如，浏览器可能会显示插入一个空行和一些缩进量。有趣的是，为了标志一个段落结束的</ P>常常被懒惰的 HTML 程序员忽略。

HTML 提供了许多机制来构造列表，包括嵌套列表。未排序的列表，例如图 7-23 所示，以<ul>开始；表项都以<li>标记。还可以用<ol>标签开始一个有序的列表。未排序列表中的个别表项通常使用标签（•）开始；而有序列表中的个别表项由浏览器对其进行编号。

最后，我们讨论超链。图 7-23 中给出的一些超链实例使用了锚 <a> 和 标签。<a> 有不同的参数，最重要的是 href，即链接的 URL。<a> 和之间的文本是显示用的。如果这段文本被选中，则浏览器跟随超链转到一个新的页面。还可以链接其他元素。例如，在<a> 和之间也可以放一个用<img>标签的图像。在这种情况下，显示的是图像，单击图像可以激活该超链接。

在这个简单例子中我们没有看到许多其他的 HTML 标记和属性。例如，<a>标签可以

带一个参数 name，该参数设置一个超链接，允许一个超链接指向一个页面的中间。这个参数非常有用，例如，对于那些由可点击表内容启动的 Web 页面。用户通过点击表的内容中的一个表项，就可以跳转到同一页面的相应部分。另一个不同的标记
，它迫使浏览器回车并开始新的一行。

也许了解标签的最好办法是看它们如何发挥作用。要做到这一点，你可以选择一个 Web 页面，然后看你浏览器中的 HTML 页面是如何放在一起的。大多数浏览器有一个查看源文件（VIEW SOURCE）的菜单项（或类似的东西）。选择该项目显示当前页面的 HTML 源代码，而不是它的格式化输出。

我们已经勾画出从早期 Web 起就存在的标签的大致轮廓。自此以后，HTML 保持着不断的演进。图 7-24 显示了一些 HTML 后续版本添加的功能。HTML 1.0 指 Web 引入时使用的 HTML 版本。随着 Web 空间的爆炸，在几年时间内 HTML 版本快速连续推出了 2.0、3.0 和 4.0。在 HTML 4.0 之后，几乎用了将近十年，才走到下一个主要版本 HTML 5.0 的标准化。因为它是一个重大的升级，巩固了浏览器处理丰富内容的方式，因此 HTML 5.0 的努力还在进行着，预计最早在 2012 年以前不会产生新的标准。不管标准如何，主流浏览器在功能上早已支持 HTML 5.0。

项目	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
超链接	×	×	×	×	×
图像	×	×	×	×	×
列表	×	×	×	×	×
活动地图和图像		×	×	×	×
表单		×	×	×	×
方程式			×	×	×
工具条			×	×	×
表格			×	×	×
访问功能				×	×
对象嵌入				×	×
风格				×	×
样式表				×	×
脚本				×	×
视频和音频				×	×
内联矢量图型					×
XML表示					×
后台线程					×
浏览器存储					×
画曲线					×

图 7-24 HTML 版本之间的某些差异

整个 HTML 版本的进展都是关于加入新的功能。这些功能是人们所希望有的，但在成为标准化之前不得不以非标准的方式来处理（比如，插件）。例如，HTML 1.0 和 HTML 2.0 没有表格功能，表格功能在 HTML 3.0 才被添加进来。一个 HTML 表中包含一个或多个行，每行包含一个或多个**表单元**（table cell），这些单元可以包含更广泛的素材（比如，文本、图像和其他表）。在 HTML 3.0 之前，需要一张表的作者不得不诉诸特殊方法来达到表格效果，诸如包括一幅显示表信息的图像。

在HTML 4.0中，增加了更多的新功能。这些功能包括残障用户的访问功能、对象嵌入（由<img>标签生成，因此其他对象也可以被嵌入到页面中）、支持脚本语言（允许显示动态的内容）以及更多的辅助功能。

HTML 5.0包含了许多功能来处理富媒体，即现在经常在网络上使用的丰富媒体。页面可以包括视频和音频，并且浏览器可以正常播放而无须用户安装插件。画画作为矢量图形可内置在浏览器中，而不必使用位图图像格式（如JPEG和GIF）。还有更多支持在浏览器中运行脚本，诸如计算和存储访问的后台线程。所有这些功能有助于支持Web页面比文档看起来更像一个具有用户接口的传统应用。这是Web发展的方向。

输入和表单

目前我们尚未讨论的一个重要功能是输入。HTML 1.0基本上是单向的。用户可以从信息提供者获取网页，但很难在另一个方向将信息发送回去。很快，双向通信的需求变得很明显，比如允许通过网页提交产品订货单、在线填写注册卡、输入搜索的词汇等。

将用户的输入发送到服务器（通过浏览器）需要两种支持。首先，它要求HTTP可以在这个方向携带数据。我们在本节后面描述这是怎么做的：它使用了POST方法。第二个要求是能够提出用户界面元素，用来收集和封装输入的数据。HTML 2.0具备这种功能，并包括了表单。

表单上包含框或者按钮，允许用户填写信息或做出选择，然后将信息发送回页面的所有者。正如在图7-25的例子中看到的那样，表单的编写就像HTML的其他部分。请注意，表单仍然是静态内容。无论谁使用它们，它们都表现出相同的行为。我们将在后面涉及的动态内容提供了更复杂的方法来收集用户输入，具体方法是发送一个其行为可能依赖于浏览器环境的程序。

像所有的表单一样，这个表单也是被包含在<form>和</form>标签之间。这个标签的属性说明了针对数据输入应该做什么，在这种情况下，使用POST方法将数据发送到指定的URL。没有被包括在标签之内的文本只是用来显示的。一个表单中允许出现所有常见的标签（例如，<b>），以便让页面的作者控制表格在屏幕上的外观。

这张表使用了三种输入框，每个输入框使用了<input>标签。它有各种参数来确定显示框的大小、性质和用法。最常见的形式是空白字段，用来接受用户输入的文本；框被检查之后，通过按下submit（提交）按钮把用户输入的数据返回给服务器。

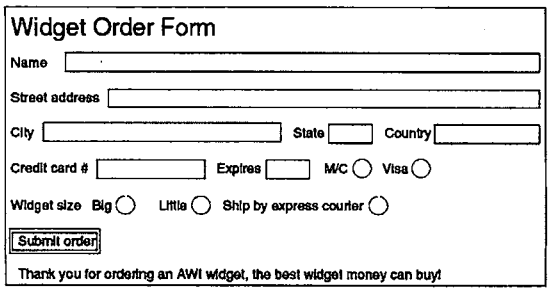
第一种输入框是text（文本）框，出现在文本“Name”之后。这个框有46个字符宽并期待用户输入一个字符串，然后这个字符串被存储在customer变量中。

表单的下一行要求输入用户的街道地址，它宽40个字符。接着下面一行是城市、州和国家。在这些字段之间没有使用<p>标签，因此，如果放得下，浏览器会在一行上（而不是作为分开的一段段）显示这些字段。至于浏览器，这段包含了六个表项：三个字符串与三个框交替出现。接下来一行要求输入信用卡号和失效日期。只有当采取了足够的安全措施以后，方才可以在Internet上传输信用卡号。我们将在第8章中讨论相关的安全话题。

在失效日期之后我们遇到了一个新的特性：单选按钮（radio button）。当用户必须在两个或者更多个选项中选择其中之一时，需要用到单选按钮。这里的智力模型像带有六个选台按钮的汽车收音机。点击其中一个按钮的同时关掉了同组中的其他按钮。按钮的视觉表

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="Submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

(a)



(b)

图 7-25

(a) 一张订单的 HTML；(b) 格式化后的页面

现由浏览器决定。构件大小（Widget Size）也使用了两个单选按钮。这两组按钮由它们的 name 字段来区分，而不是通过使用像<radiobutton> ... </radiobutton>这样的标签来划定静态范围。

Value（值）参数用来指示按下了哪个单选按钮。例如，根据用户选择的是哪个信用卡按钮，变量 cc 将被设置为字符串“mastercard”或者“visacard”。

在两组单选按钮之后我们来到发货选项，它用复选框（checkbox）表示。复选框既可以被选中，也可以未被选中。单选按钮必须选中一个，而与此不同的是每个复选框都可以被选中也可以未被选中，相互之间完全独立。

最后，我们来到 submit（提交）按钮。value 字符串就是按钮的标签，并且显示在按钮上。当用户点击 submit 按钮时，浏览器将收集到的信息打包成一个很长的行；并将其发送回服务器；该服务器由作为<form>标签一部分的 URL 提供。发送时要用到一个简单的编码。“&”用来分隔字段，“+”用来表示空格。对于我们示例的表单，发送的行可能看起来像

图 7-26 所示的内容。

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&
state=NY&country=USA&cardno=1234567890&expires=6/14&cc=mastercard&
product=cheap&express=on
```

图 7-26 浏览器对服务器的一种可能响应，包括了用户填入的信息

该字符串作为一行被发送回服务器（这里因为页面不够宽，它被分成了三行）。只有服务器能理解这个串的意义，它最有可能做的是将信息传递到一个处理这个字符串的程序。我们将在下一节讨论如何做到这点。

还可以有其他类型的输入，在这个简单的例子中没有显示出来。两种其他类型的输入是 password（密码）和 textarea（文本区域）。password 框与 text 框（这是不需要命名的默认类型）相同，只有一点除外，那就是在输入字符时并不显示出来。textarea 框也与文本框相同，但是它可以包含多个行。

对于那些必须从中作出选择的长列表，可以用提供的<select>和</SELECT>标签将一系列替代品清单包括起来。这个列表经常呈现为一个下拉菜单。下拉菜单的语义就是那些单选按钮的语义，除非给出多个参数，在这种情况下语义变成复选框的语义。

最后，还有一些方法可用来表示默认或初始值，这些值用户是可以改变的。例如，如果一个 text 框给出了一个 value 字段，那么表单中的内容会显示出来，以便用户编辑或删除。

CSS——层叠样式表

HTML 的最初目标是指定文档的结构，而不是文档的外观。例如：

```
<h1> Deborah's Photos </h1>
```

指示浏览器要强调标题，但没有说明有关字型、点的大小或颜色等任何信息。标题的外观如何表现留给浏览器考虑，因为浏览器应该知道显示器的属性（例如，它有多少像素）。然而，许多网页设计师希望能绝对控制自己设计的网页如何呈现，因此新的标签被添加到 HTML 以便控制页面的外观，例如：

```
<font face="helvetica" size="24" color="red"> Deborah's Photos </font>
```

此外，还添加了在屏幕上准确定位的控制方法。这种方法的麻烦在于它既乏味，又产生不可移植的臃肿 HTML。虽然一个页面可能在创建它的浏览器上表现非常完美，但它可能在另一个浏览器、其他同一浏览器的不同版本下或在不同屏幕分辨率的显示器上表现得一塌糊涂。

一种更好的替代方案是使用样式表（style sheet）。文本编辑器中的样式表允许作者将文本与逻辑风格关联，而不是与物理风格相关，例如，“初始段”而不是“斜体文本”。每种风格的外观单独定义。通过这种方式，如果作者决定改变以蓝色 14 点斜体表示的初始段用令人震惊的粉红色粗体 18 点表示，那么所需要做的只是改变转换整个文档的一个定义。

层叠样式表（CSS，Cascading Style Sheets）将样式表引入到 HTML 4.0 的 Web，虽然在 2000 年以前它没有被广泛使用，也没有得到浏览器的广泛支持。CSS 定义了一种简单的

语言，用来描述控制标签内容外观的规则。让我们来看一个例子。假设，AWI 希望时髦的网页用乳白色的背景，以 Arial 字体显示海军文本，而且每个级别的标题将文本分别额外放大 100%和 50%。图 7-27 的 CSS 定义给出了这些规则。

```
body {background-color:linen; color:navy; font-family:Arial;}
h1 {font-size:200%;}
h2 {font-size:150%;}
```

图 7-27 一个 CSS 例子

可以看出样式的定义很紧凑。每一行选择一个它适用的元素并且给出属性的值。一个元素的属性作为默认值适用于所有它包含的其他 HTML 元素。因此，body 的式样设置了主体中文本段落的式样。也有方便速记的颜色名称（例如，red）。任何没有定义的样式都以浏览器的默认值填充。这种行为使得样式表的定义成为可选，没有定义也可以产生某种合理的表现。

样式表可以被放置在 HTML 文件中（例如，使用<style>标签），但更常见的做法是将它们放置在一个单独的文件中，然后引用它们。例如，AWI 页面的<head>标签可以修改成引用文件 awistyle.css 中的样式表，如图 7-28 所示。这个例子还显示了 CSS 文件的 MIME 类型是 text/css。

```
<head>
<title> AMALGAMATED WIDGET, INC. </title>
<link rel="stylesheet" type="text/css" href="awistyle.css" />
</head>
```

图 7-28 包括了一个 CSS 样式表

这种策略有两大优势。首先，它可以使得一组样式被应用到一个网站上的许多网页。这种组织方式带来了页面外观上的一致性，即使这些页面在不同的时间由不同的作者开发；而且允许通过编辑一个 CSS 文件而不是 HTML 来改变整个网站的外观。这个方法可以与 C 程序中的#include 文件相类比：在这里改变一个宏的定义就等效于改变了所有包括该头文件的程序。第二个好处是下载的 HTML 文件保持得很小。这是因为浏览器可以下载 CSS 文件的一个副本，供所有引用它的页面使用。浏览器并不需要为每个网页的定义下载一份新副本。

7.3.3 动态 Web 页面和 Web 应用

到目前为止我们使用的静态页面模型将页面作为多媒体文档处理，这些文档被方便地链接在一起。这是 Web 初期的一个拟合模型，因为大量的信息都可在线查询。如今，围绕着 Web 的许多兴奋点在于可将它用作应用程序和服务。应用例子包括在电子商务网站购买产品、检索图书馆目录、探索地图、阅读和发送电子邮件，以及进行文档合作。

这些新的用途犹如传统的应用软件（例如，邮件阅读器和文字处理器）。扭曲的只是这些应用程序运行在浏览器内部，而用户数据存储在 Internet 数据中心的服务器上。它们利用 Web 协议通过 Internet 访问信息，而浏览器显示用户界面。这种方法的优点是用户不需要安装单独的应用程序，可以从不同的计算机访问用户数据，而且由服务提供商负

责备份。它被证明是如此地成功，几乎可以和传统的应用软件相媲美。当然，这些应用程序由大型服务提供商免费提供的事实也有助于推动应用的进一步展开。这种模式就是云计算（cloud computing）的普遍形式，它将计算从个人台式电脑转移到 Internet 上的共享服务器集群。

为了运行应用程序，Web 页面不能再是静态的了，动态内容是必需的。例如，一个图书馆目录的页面应该反映出哪些书籍当前可借和哪些书籍已经借出因而不可用等书籍的当前状态。类似地，一个有用的股票市场页面将允许用户与页面交互，以便查看不同时期的股票价格，以及计算利润和亏损。这些事例表明，动态内容可以由服务器上或浏览器内（或者同时运行在两个地方）运行的程序产生。

在本节中，我们将依次考查这两种情况。一般的情况如图 7-29 所示。例如，考虑一个地图服务：让用户输入一个街道地址，然后给出相应位置的地图。给定一个位置请求，Web 服务器必须使用一个程序来创建一个页面，该页面显示了该街道地址在地图上的对应位置，而位置信息则是从一个街道数据库和其他地理信息数据库中提取出来的。这一系列动作反映在图中的第 1～3 步。请求（第 1 步）导致在服务器上运行一个程序；该程序查询一个数据库以便生成相应的页面（第 2 步），并将该页面返回给浏览器（第 3 步）。

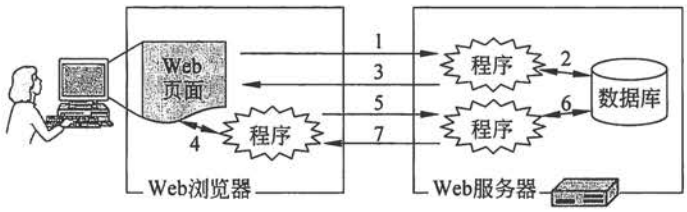


图 7-29 动态页面

然而，有更多的动态内容。返回的页面本身还可能包含着在浏览器中运行的程序。以我们的地图为例，该程序将允许用户发现一条路线，以及不同详细程度地探索周围的附近区域。它会更新页面，跟随用户的指示缩小或放大（第 4 步）。为了处理一些交互事件，该程序可能需要从服务器端获取更多的数据。在这种情况下，程序将给服务器发送一个请求（第 5 步），服务器从数据库中检索出更多的信息（第 6 步），并且给浏览器返回一个响应（第 7 步）。然后，该程序将继续更新页面（第 4 步）。请求和响应都发生在后台，用户甚至可能根本意识不到它们的存在，因为页面的 URL 和标题通常不会改变。通过引入客户端程序，页面可以提供一个比单靠服务器端程序更能响应用户的接口。

服务器端动态 Web 页面生成

让我们更详细地来看服务器端是如何生成动态页面内容的。服务器端必须处理的一个简单情况是表单的使用。回到图 7-25 的例子中，考虑用户填写了 AWI 邮购表单，如图 7-25（b）所示，然后点击“提交订单”按钮。当用户点击该按钮后，一个请求就被发送到由 URL 指定的服务器，请求包含表单以及用户填写的表单内容（在这种情况下，是一个到 http://widget.com/cgi-bin/order.cgi 的 POST）。这些数据必须由某个程序或脚本来处理。因此，该 URL 指定了要运行的程序，提供的数据作为该程序的输入。在这种情况下，处理将涉及在 AWI 的内部系统输入订单、更新客户记录和收取信用卡的付费。这个请求返回的页面将

取决于处理过程中发生的事情，它并不像一个静态页面那样固定不变。如果订单成功，返回的页面可能给出了预计的出货日期；如果订单不成功，返回的页面可能会给出订购的部件缺货或出于某种原因信用卡不再有效等有关信息。

究竟如何在服务器上运行一个程序而不是检索一个文件则取决于 Web 服务器的设计。Web 协议本身没有说明。这是因为该接口可以是专有的，浏览器并不需要知道里面的细节。至于浏览器，它仅仅负责发出请求和获取页面。

然而，为了让 Web 服务器能够调用程序，已经开发出了标准的 API。这些接口的存在使得开发人员更加容易地把不同的服务器扩展到 Web 应用程序。我们将通过简要地考查两个 API，让你感觉它们是如何工作的。

第一种处理动态页面请求的方法称为**公共网关接口**（CGI，Common Gateway Interface），由 RFC 3875 定义，这是自从有了 Web 就一直可用的方法。CGI 提供了一个接口，允许 Web 服务器与后端程序及脚本通信；这些后端程序和脚本接受输入信息（例如，来自表单），并生成 HTML 页面作为响应。通常，这些后端程序可以用任何一种开发者便利的语言编写，通常都是些方便开发的脚本语言。可以选择 Python、Ruby、Perl 或任何你喜欢的语言。

按照惯例，通过 CGI 调用的程序常驻在一个称为 cgi-bin 的目录下，该目录在 URL 中可以看到。服务器将一个请求映射到这个目录下一个程序名，并且以一个单独的进程执行该程序。它把与请求一起发送过来的任何数据作为程序的输入，而程序的输出则提供了一个返回给浏览器的网页。

在我们的例子中，调用的程序是 order.cgi，它以图 7-26 所示形式的编码作为输入。它解析参数并处理订单。一个有用的约定是如果没有提供表单的输入数据，那么该程序将返回订单的 HTML。通过这种方式，程序一定要知道表单的表示。

我们将看到的第二种常见的 API 方法相当不同。这里的方法是在 HTML 页面中嵌入少量的脚本，然后让服务器来执行这些脚本以便生成最终发送给客户的页面。编写这些脚本的一种流行语言是**超文本预处理器**（PHP，Hypertext Preprocessor）。为了使用 PHP，服务器必须能够理解 PHP，就好像浏览器必须理解 CSS 才可以解释具有样式表的 Web 页面一样。通常，服务器用文件扩展名 php 来标识包含 PHP 的 Web 页面，而不是用 html 或者 htm 扩展名。

PHP 的使用类似于 CGI。作为一个展示其如何处理表单的例子，我们看图 7-30(a)所示的例子。此图的顶部包含了一个含有简单表单的普通 HTML 页面。这时，<form>标签指明了当用户提交表单后应该调用 action.php 文件来处理参数。该页面显示了两个文本框，一个要求输入名字，另一个要求输入年龄。当用户填写了这两个框并且表单被提交后，服务器解析类似图 7-26 那样的字符串，并且把名字放到 name 变量中，把年龄放到 age 变量中；然后，作为应答，服务器开始处理 action.php 文件，如图 7-30(b)所示。在处理这个文件的过程中，要执行 PHP 命令。如果用户在框中输入的是“Barbara”和“32”，那么返回的 HTML 文件将如图 7-30(c)所示。因此，使用 PHP 以后处理表单变得非常简单。

尽管 PHP 非常易于使用，但它实际上是一种功能强大的程序设计语言，是 Web 和服务器数据库的接口。它具有变量、字符串、数组，以及在 C 语言中可以找得到的绝大多数控制结构，而且它还有比 printf 更强大的 I/O 功能。PHP 是开放源码，可以免费获取，而

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

(b)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 33
</body>
</html>
```

(c)

图 7-30

(a) 包含了一个表单的 Web 页面；(b) 处理该表单输出的 PHP 脚本；
(c) 当输入分别是“Barbara”和“32”时 PHP 脚本的输出

且用途广泛。经过精心设计，PHP 能与 Apache 服务器很好地协同工作；而 Apache 也是开放源码的，是世界上使用最为广泛的 Web 服务器。要了解 PHP 的更多信息，请参看（Valade，2009）。

我们现在已经看到了两种用来生成动态 HTML 页面的不同方法：CGI 脚本和内嵌的 PHP。除此之外，还存在几种其他技术可选。**Java 服务器页面**（JSP，JavaServer Pages）与 PHP 非常相似，只不过页面中的动态部分是用 Java 编程语言而不是 PHP 编写的。使用这种技术的页面具有文件扩展名.jsp。**活动服务器页面**（ASP.NET，Active Server Page）是 Microsoft 版本的 PHP 和 JSP。它使用 Microsoft 专用的.NET 网络应用框架来生成动态内容。使用这种技术的页面具有文件扩展名.aspx。在这 3 种技术中，究竟选择哪种通常更多地与策略有关（开放源码与 Microsoft）而并非取决于技术，因为这 3 种语言旗鼓相当。

客户端动态 Web 页面生成

PHP 和 CGI 脚本解决了处理表单输入以及服务器与数据库的交互问题。它们都可以接受来自表单的入境信息、查询一个或多个数据库，然后利用查找结果生成 HTML 页面。它们所不能做的是响应鼠标移动事件，或者直接与用户交互。为了达到这个目的，有必要在 HTML 页面中嵌入脚本，而且这些脚本必须运行在客户机上而不是在服务器上。从 HTML

4.0 开始，可以通过<script>标签来启用这样的脚本。用来产生这种交互式 Web 页面的技术统称为**动态 HTML**（dynamic HTML）。

最流行的客户端脚本语言是 JavaScript，我们现在快速简短地了解它。无论名字看起来有多么的类似，JavaScript 与 Java 编程语言几乎没有共同之处。与其他脚本语言一样，JavaScript 是一种非常高级的语言。例如，仅仅只要一行 JavaScript 代码就可以弹出一个对话框、等待文本输入，然后将结果字符串存放到一个变量中。这种高级特性使得 JavaScript 非常适合于设计交互式 Web 页面。另一方面，它的变异速度比用 X 光机捕捉到的果蝇还要快，所以，要想编写出能在所有平台上都良好工作的 JavaScript 程序非常困难，但也许有一天它会稳定下来。

作为 JavaScript 程序的一个例子，考虑图 7-31。与图 7-30 类似，它显示了一个表单要求输入名字和年龄，并且预测这个人明年的年龄。主体部分基本上与 PHP 例子相同，主要的区别在于 submit 按钮的声明，以及其中的赋值语句。赋值语句告诉浏览器，当按钮被单击时调用 response 脚本，并且将表单作为一个参数传递给它。

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test#form) {
  var person = test#form.name.value;
  var years = eval(test#form.age.value) + 1;
  document.open();
  document.writeln("<html> <body>");
  document.writeln("Hello " + person + ".<br>");
  document.writeln("Prediction: next year you will be " + years + ".");
  document.writeln("</body> </html>");
  document.close();
}
</script>
</head>
<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>
```

图 7-31 使用 JavaScript 处理表单

这里全新的部分在于 HTML 文件头部对 JavaScript 函数 response 的声明，HTML 文件的头部通常是为标题、背景颜色等保留的。这个函数从表单的 name 字段提取出相应的值，并将它作为一个字符串存放在 person 变量中。它还提取出 age 字段的值，并且通过 eval 函数将该值转化为一个整数，然后加 1 并把结果存放在 years 变量中。接着，它打开一个用于输出的文档，利用 writeln 方法向这个文档做了四次写操作，然后关闭文档。该文档是一个 HTML 文件，这可以从文档中的各种 HTML 标签看出来。然后浏览器在屏幕上显示这

个文档。

PHP 和 JavaScript 看上去很相似，它们都是嵌入在 HTML 文件中的代码，但它们的处理方式完全不同，理解这一点非常重要。在图 7-30 的 PHP 例子中，当用户单击了 submit 按钮后，浏览器将表单中的信息收集到一个长字符串中，然后将它发送给服务器请求一个 PHP 页面。服务器加载该 PHP 文件，并且执行内嵌的 PHP 脚本，由此产生一个新的 HTML 页面。然后该页面被送回给浏览器以便显示出来。浏览器不能确定这是由一个程序生成的。这个过程由图 7-32(a)的第 1～4 步表示。

在图 7-31 的 JavaScript 示例中，当用户点击 submit 按钮，浏览器就解释该页面中包含的一个 JavaScript 函数。所有的工作都在本地完成，即在浏览器内部完成。这时没有与服务器取得任何联系。这个处理过程如图 7-32(b)的第 1 步和第 2 步所示。因此，结果几乎在瞬间就显示出来，而用 PHP 生成的 HTML 在到达客户端之前可能有几秒钟的延迟。

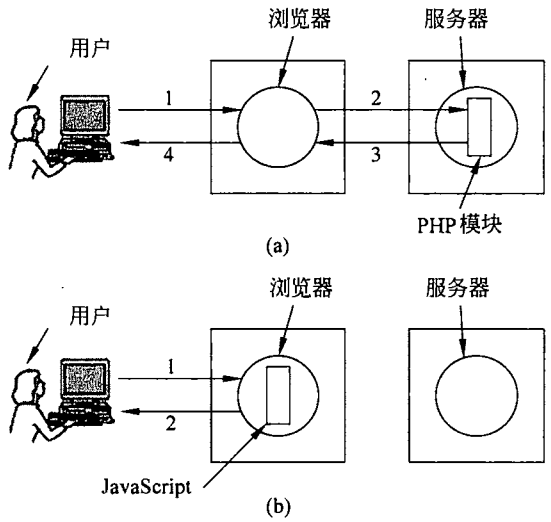


图 7-32

(a) 使用 PHP 的服务器端脚本；(b) 使用 JavaScript 的客户端脚本

这种区别并不意味着 JavaScript 比 PHP 更好。它们的用途完全不同。PHP（同时也暗示着 JSP 和 ASP）用在需要与远程数据库进行交互时；而 JavaScript（和其他我们将要提及的客户端语言，比如 VBScript）则用在需要客户端计算机与用户交互时。当然有可能将两者结合起来，正如我们马上要看的。

JavaScript 并不是使 Web 页面具备高度交互性的唯一方法。另外一种适用于 Windows 平台的方法是 VBScript，它基于 VisualBasic。还有另一种跨平台的流行方法是使用小程序（Applet）。这些 Java 小程序已经被编译成一种虚拟机的机器指令，这种虚拟机称为 Java 虚拟机（JVM，Java Virtual Machine）。Applet 可以被嵌入到 HTML 页面中（在<applet>和</applet>之间），并被具有 JVM 能力的浏览器解释执行。因为 Java 小程序是被解释执行而不是被直接执行的，因此 Java 解释器可以防止它们做坏事。至少理论上是这样的。实际上，小程序编写者已经发现在 Java I/O 库中有近乎无限的错误需要被发掘。

Microsoft 对于 Sun 公司 Java 小程序的回应是允许 Web 页面包含 ActiveX 控件，这是指一种被编译成 x86 机器指令的程序，可以直接在硬件裸机上执行。这种特性使得 ActiveX

控件比解释执行的 Java 小程序更快更灵活，因为它可以完成任何普通程序能够做到的事情。当 Internet Explorer 浏览器在某个 Web 页面中发现了一个 ActiveX 控件时，它会下载该控件，检验它的身份然后执行该控件。然而，下载和运行外部程序将会带来巨大的安全问题，我们将在第 8 章中讨论这些安全问题。

由于几乎所有的浏览器都可以解释 Java 程序和 JavaScript，所以如果设计者希望制作出具有高度交互性的 Web 页面，则他至少有两种技术可供选择；另外，如果多平台的移植性不是一个问题，则还可以考虑使用 ActiveX 技术。作为一般规则，JavaScript 比较容易编写，Java 小程序执行得更快一些，而 ActiveX 控件是运行速度最快的。而且，由于所有的浏览器实现了完全相同的 JVM，但是没有两个浏览器实现了相同版本的 JavaScript，所以 Java 小程序比 JavaScript 程序具有更好的移植性。要想了解关于 JavaScript 的更多信息，有很多书可以参考，每一本都有很多页（通常超过 1000 页），例如（Flanagan，2010）。

AJAX——异步 JavaScript 和 XML

引人注目的 Web 应用程序需要用户界面具备可响应能力，而且能无缝访问存储在远程 Web 服务器上的数据。客户端上的脚本（例如，使用 JavaScript）和服务器上的脚本（比如 PHP）只是提供某种解决方案的基本技术，这些技术通常与其他几个关键技术结合起来一起使用，这些技术的组合就称为异步 JavaScript 和 XML（AJAX，Asynchronous JAvascript and Xml）。许多全功能的 Web 应用，比如谷歌的 Gmail、地图和文档都是以 AJAX 编写的。

AJAX 有点混乱，因为它不是一种语言。它是一组需要一起协同工作的技术，正是这些技术使得 Web 应用程序和传统的桌面应用一样能响应用户的每个动作。这些技术包括：

（1）用来表现页面信息的 HTML 和 CSS。

（2）浏览时改变部分页面的 DOM（Document Object Model）。

（3）使得程序与服务器交换应用数据的 XML（eXtensible Markup Language）。

（4）程序发送和检索 XML 数据的异步方式。

（5）将所有功能组合在一起的 JavaScript。

由于这是一个相当大的集合，我们将逐个考查每一块的属性。我们已经看过了 HTML 和 CSS。它们标准化了如何描述内容和如何显示内容。可以产生 HTML 和 CSS 的任何程序都可以使用 Web 浏览器作为显示引擎。

文档对象模型（DOM，Document Object Model）是通过程序访问 HTML 页面的一种表示。这种表示结构化成一棵反映 HTML 元素的树。例如，图 7-30(a)中 HTML 对应的 DOM 树如图 7-33 所示。根是一个 HTML 元素，代表整个 HTML 块。这个元素是 body 元素的父节点，body 元素反过来又是 form 元素的父节点。表单有两个属性，画在表单的右侧，一个属性为表单的方法（POST），另一个属性为表单的动作（请求的 URL）。这个元素有三个子结点，反映了表单包含的两个段落标签和一个输入标签。在树的底部是叶结点，包含了元素或者文字，如文本字符串。

DOM 模型的意义在于它给程序提供了一种改变部分页面内容的简单方法。没有必要重写整个页面。只有包含了需要改变内容的树结点才对页面内容进行替换。当页面内容发生变化时，浏览器将相应地更新显示内容。例如，如果部分页面上的一个图像被 DOM 改变了，浏览器将更新该图像，而不改变页面的其他部分内容。当图 7-31 所示的 JavaScript 示

例被添加到 document 元素中，导致新的一行文本出现在浏览器窗口的底部，我们已经看到上述行为反映在 DOM 动作中。DOM 是一个强大的制作网页方法，并且可以演变发展。

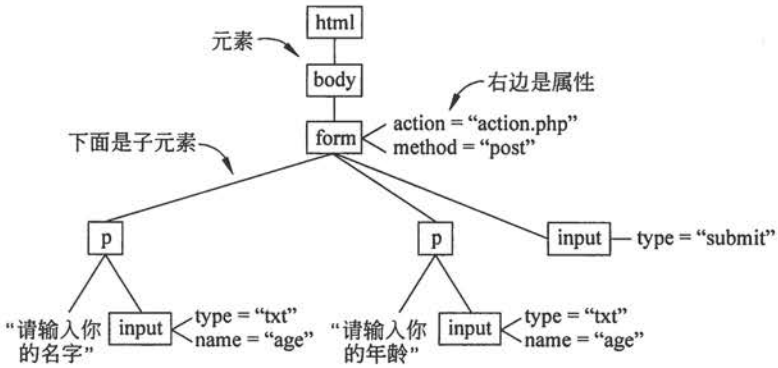


图 7-33 如图 7-30(a)所示 HTML 的 DOM 树

第三个技术，**可扩展标记语言**（XML，eXtensible Markup Language）是一个说明结构化内容的语言。HTML 将内容与格式混合在一起，因为它关注的是信息的表示。然而，随着 Web 应用程序变得越来越普遍，将独立的结构化内容从它的表示中分离开来的需求也越来越强烈。例如，考虑一个程序，针对一些书它去搜索 Web，以期找出最好的价格。为此它需要分析许多 Web 页面，寻找表项的标题和价格。对程序来说，要找出 HTML 网页中哪里是标题、哪里是价格是一件非常困难的事情。

出于这个原因，W3C 开发了 XML（Bray 等，2006），为了能够自动处理允许网页内容结构化。与 HTML 不同，XML 没有定义任何标签。每个用户可以定义她自己的标签。图 7-34 给出了一个简单的 XML 文档例子。它定义了一个名为 book_list 的结构，这是一个图书列表。每本书有 3 个字段：标题、作者、出版年份。这些结构非常简单。结构允许有重复的字段（例如，多个作者）、可选字段（例如，有声读物 URL）和替代字段（例如，如果它是印刷版，给出一家书店的 URL；或者如果它是绝版书，给出拍卖网站的 URL）。

在这里例子中，三个字段中的每一个都是不可分割的整体，但它允许被进一步划分。例如，作者字段可作如下划分，以便对搜索和格式化做细粒度的控制：

```
<author>
<first_name> George </first_name>
<last_name> Zipf </last_name>
</author>
```

每个字段进一步划分成子字段和子字段，可以达到任意深度。

图 7-34 全部文件所做的只是定义了一个包含 3 本书的图书清单。它非常适合于浏览器端运行的程序和服务器传输信息，但它没有说明如何显示作为网页的文档。要做到这一点，一个分析信息并判定 1949 是图书出版年的程序可能输出这样的 HTML：标题被标记为斜体文本。另外，一种称为**可扩展样式表语言转换**（XSLT，eXtensible Stylesheet Language Transformations）的语言可以用来定义如何将 XML 转换成 HTML。XSLT 类似于 CSS，但比 CSS 更强大。我们将竭尽全力为你提供一些细节。

```
<?xml version="1.0" ?>
<book#list>
<book>
<title> Human Behavior and the Principle of Least Effort </title>
<author> George Zipf </author>
<year> 1949 </year>
</book>
<book>
<title> The Mathematical Theory of Communication </title>
<author> Claude E. Shannon </author>
<author> Warren Weaver </author>
<year> 1949 </year>
</book>
<book>
<title> Nineteen Eighty-Four </title>
<author> George Orwell </author>
<year> 1949 </year>
</book>
</book#list>
```

图 7-34 一个简单的 XML 文档

不用 HTML 而采用 XML 来表示数据的另一个优点是数据更易于被程序分析。HTML 最初是手工编写（而且现在往往仍然如此），所以它多少有点马虎。有时候会忘记写上结束标签，比如</p>。其他标签没有匹配的结束标签，像
。还有其他标签可能被不恰当地嵌套着，而且标签和属性名可以有所不同。大多数浏览器都各尽其能地工作。XML 在定义上更严格也更清晰。标签名字和属性总是小写的，标签必须始终以它们打开的相反次序关闭（或者清楚地表明它们是否是一个没有相应关闭的空标签），而且属性值必须用引号括起来。这种精确定义使得程序解析更加容易，而且绝无二义性。

甚至 HTML 也可以按照 XML 术语来定义。这种方法称为**扩展超文本标记语言**（XHTML，eXtended HyperText Markup Language）。基本上，它是一个非常挑剔的 HTML 版本。XHTML 页面必须严格符合 XML 规则，否则它们不会被浏览器接受。这种情况下，没有更多伪劣的网页，而且也不存在浏览器之间的不一致。就 XML 而言，其目的是为了产生更好的网页供程序处理（即 Web 应用程序）。虽然 XHTML 自 1998 年就已经出现，但它的发展一直很缓慢。生成 HTML 的人们不明白为什么他们需要 XHTML，而且浏览器的支持也已经落后。现在 HTML 5.0 已经被定义出来，因此页面可以表示成 HTML 也可以表示成 XHTML，这样有助于过渡。最终，XHTML 将取代 HTML，但在完成过渡之前将是一个漫长的过程。XML 也被证明可用作程序之间的流行通信语言。当这种通信由 HTTP 协议承载时（在下一节中描述 HTTP 协议），就称为 Web **服务**（Web service）。特别是，**简单对象访问协议**（SOAP，Simple Object Access Protocol）是一个实现了 Web 服务的方法，它以一种语言无关并且系统无关的方式执行程序之间的 RPC。客户端构造一个 XML 消息的请求，并使用 HTTP 协议把它发送到服务器；服务器返回一个 XML 格式的消息作为应答。这样，异构平台上的应用程序可以进行通信。

回到 AJAX，我们的观点很简单，当运行在浏览器和服务器上的程序需要进行数据交换时，XML 是一种有用的格式。然而，为了在发送或接收数据时为浏览器提供可响应的接

口，它必须尽可能用脚本来执行异步 I/O 而不会阻塞页面的显示，同时等待对一个请求的响应。例如，考虑在浏览器中滚动一幅地图。当浏览器收到滚动动作的通知，地图页面上的脚本可能会向服务器请求更多的地图数据，尤其是显示的地图接近数据的边缘。在获取这些数据时界面不应该呆滞，因为这样的接口不会赢得任何用户的赞赏。相反，地图的滚动应该继续平滑地进行。当数据到达后要通知该脚本以便它可以使用新到的数据。如果一切顺利，新的地图数据应该在需要它之前获得。现代浏览器都支持这种通信模型。

拼图中的最后一块是将 AJAX 保持住的脚本语言，对上述列表中的技术提供访问方式。在大多数情况下，这种语言是 JavaScript，但也有诸如 VBScript 的替代品。我们之前介绍了 JavaScript 的一个简单例子。注意不要被这种简单性所愚弄，JavaScript 有许多怪癖；但它是一个完全成熟的编程语言，拥有所有 C 或 Java 的能力。它有变量、字符串、数组、对象、函数和所有通常的控制结构。它还具有与浏览器和 Web 页面的特殊接口。JavaScript 可以在屏幕上跟踪鼠标在对象上的移动，这样很容易突然弹出一个菜单，使得 Web 页面看起来更加生动活泼。它还可以使用 DOM 来访问页面、操作 HTML 和 XML，并执行异步 HTTP 通信。

在我们离开动态页面这个主题之前，让我们简要地总结到目前为止涉及的技术，并用一张图把它们联系起来。完整的 Web 页面可以用服务器机器上的各种脚本动态生成。这些脚本可以用诸如 PHP、JSP 或 ASP.NET 这样的服务器扩展语言编写，或者由作为单独的 CGI 进程运行，从而可以用任何语言来编写。这些选项如图 7-35 所示。

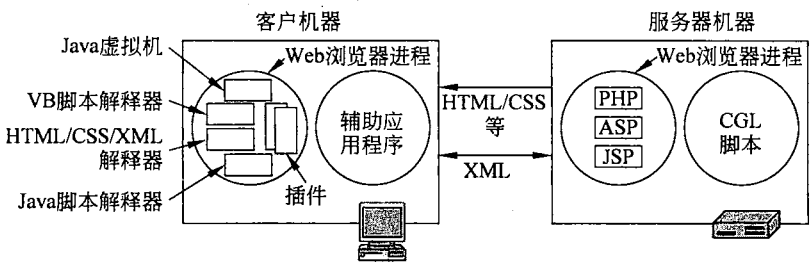


图 7-35 用来生成动态页面的不同技术

一旦浏览器接收到这些网页，就把它们视为 HTML、CSS 和其他 MIME 类型的正常页面对待，并且显示给用户。可以通过安装运行在浏览器内部的插件和运行在浏览器之外的辅助应用程序，来扩展浏览器支持的 MIME 类型。

在客户端生成动态内容也是可能的。嵌入到网页中的程序可以用 JavaScript、VBScript、Java 和其他语言编写。这些程序可以执行任意的计算，并且更新屏幕显示。使用 AJAX，Web 页面中的程序可以异步地与服务器交换 XML 和其他各种数据。这种模式支持丰富的 Web 应用程序，使它们看起来就像传统的应用程序一样，只有一点除外，那就是它们运行在浏览器内而且访问的信息存储在 Internet 的服务器上。

7.3.4 HTTP——超文本传输协议

既然我们已经理解了 Web 内容和应用，现在是时候考察在 Web 服务器和客户之间传输所有这些信息的协议了。这就是**超文本传输协议**（HTTP，HyperText Transfer Protocol），

由 RFC 2616 说明。

HTTP 是一个简单的请求-响应协议，它通常运行在 TCP 之上。它指定了客户端可能发送给服务器什么样的消息以及得到什么样的响应。请求和响应消息的头以 ASCII 码形式给出，就像 SMTP 一样；而消息内容则具有一个类似 MIME 的格式，也像 SMTP 一样。这个简单模型是早期 Web 成功的有功之臣，因为它使得开发和部署是那么的直截了当。

在本节，我们将着眼于时下使用 HTTP 的一些更加重要的属性。然而，在进入 HTTP 的细节之前，我们注意到它在 Internet 上的使用方式在不断地演化。HTTP 是一个应用层协议，因为它运行在 TCP 之上，并且与 Web 密切相关。这就是为什么我们在本章讨论它。然而，在另一种意义上，HTTP 变得越来越像一个传输层协议，因为它为进程之间跨越不同网络进行内容通信提供了一种方式。这些进程不一定必须是 Web 浏览器和 Web 服务器。媒体播放器也可以使用 HTTP 与服务器通信并请求专辑信息。防病毒软件可以使用 HTTP 来下载最新的病毒库更新。开发人员可以使用 HTTP 来获取项目文件。消费电子产品比如数码相框通常使用内嵌的 HTTP 服务器作为与外界的接口。机器与机器之间的通信越来越多地通过 HTTP 运行。例如，航空公司服务器可以使用 SOAP（一种运行在 HTTP 之上的 XML RPC）联系一家汽车租赁服务器，并且进行汽车预订，所有这些都作为度假产品的一部分。这种趋势有可能继续下去，伴随着 HTTP 的使用而不断扩大。

连接

浏览器与服务器联系最常用的方法是与服务器机器上的端口 80 建立一个 TCP 连接，虽然这个过程并不是正式要求的。使用 TCP 的意义在于浏览器和服务器都不需要担心如何处理长消息、可靠性或拥塞控制。所有这些事情都由 TCP 实现负责处理。

在 Web 早期 HTTP 1.0 中，连接被建立起来以后浏览器只发送一个请求，之后一个响应消息被发回来，再然后 TCP 连接就被释放了。那时，整个 Web 页面通常只包含 HTML 文本，因此这种方法足够用了。很快，Web 页面发展成还含有大量的嵌入式内容链接，比如图标以及其他精美的东西。建立一个单独的 TCP 连接来传输每个图标的操作方式代价太昂贵。

这种现象导致了 HTTP 1.1 的诞生，它支持**持续连接**（persistent connection）。有了这种连接，就有可能建立一个 TCP 连接，在其上发送一个请求并得到一个响应，然后再发送额外的请求并得到额外的响应。这种策略也称为**连接重用**（connection reuse）。通过把 TCP 连接的建立、启动和释放等开销分摊到多个请求上，相对于每个请求的 TCP 开销就被大大地降低了。而且，还可以发送流水线请求，也就是说在请求 1 的响应回来之前发送请求 2。

这 3 种情况的性能差异如图 7-36 所示。图 7-36(a)部分显示了 3 个请求，一个接着一个，每一个都是一个单独的连接。让我们假设这表示一个 Web 页面，其中有两个嵌入式图像在同一台服务器上。图像的 URL 被确定为主页，并且已经获取，因此在主页之后这些图像也已经获取过来。如今，一个典型的页面大约有 40 个对象，这些对象必须都取来才能表示出来，但是这将使我们的图太大，所以我们只使用两个嵌入式对象。

在图 7-36(b)中，通过一个持续连接来获取页面。也就是说，TCP 连接在启动后是开放的，然后发送相同的 3 个请求，一个接着一个，后一个跟前一个一样，然后才关闭连接。可以观察到页面的提取任务更快速地完成了。加速的原因有两个：首先，时间没有浪费在

建立更多的连接上。每个 TCP 连接至少需要一个往返时间才能建立；其次，相同图像的传输更加迅速。为什么会这样？这是因为 TCP 的拥塞控制。在一个连接的开始阶段，TCP 使用慢启动过程来增加吞吐量，直到它了解网络路径的行为。这个预热期的后果是多个短 TCP 连接比一个长 TCP 连接传输信息所需的时间更长。

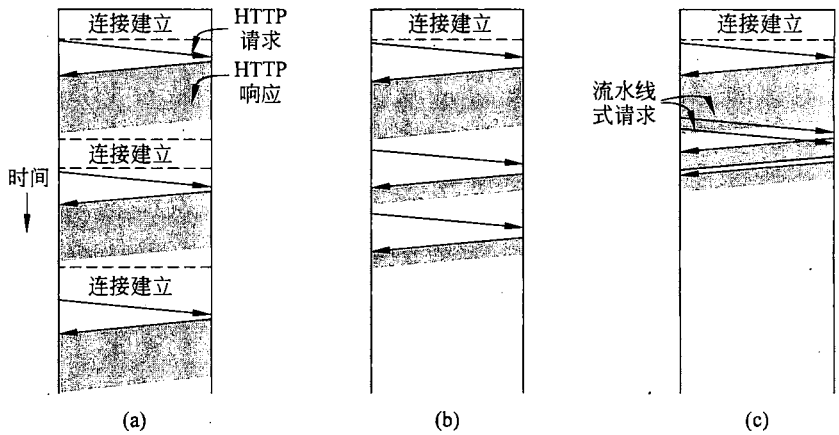


图 7-36　3 种情况下的 HTTP

(a) 具有多个连接和系列请求；(b) 具有一个持续连接和系列请求；(c) 具有一个持续连接和流水线式请求

最后，在图 7-36(c)中，有一个持久连接和请求流水线。具体来说，第二个和第三个要求尽快发送出去，一旦检索主页足够识别出必须预取图像就立即发送请求。这些请求的响应最终紧随其后。这种方法减少了服务器处于空闲状态的时间，因此进一步提高了性能。

然而，持续连接不是免费的。一个新的问题出现了，那就是什么时候关闭连接。一个到服务器的连接在页面加载时应该保持开放。然后呢？对于用户来说有个很好的机会，他可以点击一个链接从服务器请求另一个页面。如果连接保持打开状态，那么下一个请求被立即发送出去。不过，谁也不能保证任意时间后客户将向服务器发出另一个请求。实际上，客户机和服务器通常将持续连接保持打开状态，直到它们已经闲置一小段时间（例如 60 秒），或者它们已经打开了大量的连接必须要关闭一些。

细心的读者可能已经注意到还有一种组合，我们到目前为止还没有提及。那就是，在每个 TCP 连接上发送一个请求，但同时并行打开多个 TCP 连接。这种**并行连接**（parallel connection）方法在持续连接之前被浏览器广泛使用。它具有与顺序连接相同的缺点——额外的开销，但性能比顺序连接的要好。这是因为建立和加大并行连接隐藏着一些延迟。在我们的例子中，两个嵌入式图像的连接可以在同一时间建立。然而，在同一台服务器上运行多个 TCP 连接会令人沮丧。原因在于 TCP 为每个连接单独执行拥塞控制。因此，连接之间彼此竞争，造成丢包率上升，而且聚合起来比单个连接的网络用户更具侵略性。持续连接则更优越，应该比并行连接优先使用，因为它们避免了开销，而且不会遭受流量拥塞问题。

方法

尽管 HTTP 是为 Web 而设计的，但出于放眼未来面向对象的使用，它的设计有意识地比 Web 所需要的更加通用。正是这个原因，HTTP 不仅支持请求一个 Web 页面，而且支持

操作——称为方法（method）。这种通用性是 SOAP 得以存在的主要原因。

每个请求由一行或多行 ASCII 文本组成，其中第一行的第一个词是被请求的方法名字。图 7-37 列出了内置的方法。方法名区分大小写，因此 GET 是合法的方法而 get 不是。

方法	描述
GET	读取一个Web页面
HEAD	读取一个Web页面的头
POST	附加一个Web页面
PUT	存储一个Web页面
DELETE	删除一个Web页面
TRACE	回应入境请求
CONNECT	通过代理连接
OPTIONS	一个页面的查询选项

图 7-37 内置的 HTTP 请求方法

GET 方法请求服务器发送页面（在大多数情况下，当我们说“页面”时我们指的是“对象”，但是把一个页面想象成一个文件，理解这个概念就足够了）。该页面被适当编码成 MIME。大部分发送给 Web 服务器的请求都是 GET 方法。GET 的通用形式是：

```
GET filename HTTP/1.1
```

其中 filename 是预取的页面名字，1.1 是协议版本号。

HEAD 方法只请求消息头，不需要真正的页面。这个方法可以收集建索引所需要的信息，或者只是测试一下 URL 的有效性。

当提交表单时需要用到 POST 方法。POST 方法与 GET 方法都可被用作 SOAP Web 服务。与 GET 类似，它也携带一个 URL，但不是简单地检索一个页面，而是上载数据到服务器（即表单的内容或者 RPC 参数）。然后，服务器利用这些数据做某件事，具体取决于 URL，概念上是将数据“附加”到对象上。效果或许是购买一个表项，或者调用一个过程。最后，方法返回一个指出结果的页面。

其余的方法对于浏览 Web 并不常用。PUT 方法与 GET 方法相反：它不是读取页面，而是写入页面。通过这个方法可以在远程服务器上建立起一组 Web 页面。这个请求的主体包含了页面。页面可以利用 MIME 来编码，在这种情况下，跟在 PUT 后面的行可能包含了认证头，以便证明调用者的确有权执行所请求的操作。

DELETE 方法的用途可能与你想象的一样：删除页面，或者至少指出 Web 服务器已经同意删除该页面。与 PUT 方法类似，认证和许可机制在这里起到了很重要的作用。

TRACE 方法用于调试。它指示服务器发回收到的请求。当请求没有被正确地处理而客户希望知道服务器实际得到的是什么样的请求时，这个方法非常有用。

CONNECT 方法使得用户通过一个中间设备（比如 Web 缓存）与 Web 服务器建立一个连接。

OPTIONS 方法提供了一种办法让客户向服务器查询一个页面并且获得可用于该页面的方法和头。

每个请求都会得到一个响应，每个响应消息由一个状态行及可能的附加信息（例如全部或者部分 Web 页面）组成。状态行包括一个 3 位数字的状态码，该状态码指明了这个请求是否被满足；如果没有满足，那么原因是什么。第一个数字把响应分成 5 大组，如图 7-38

所示。1××码实际上很少被使用。2××码意味着这个请求被成功地处理，并且返回了相应的内容（如果有的话）。3××码告诉客户应该检查其他地方：使用另一个不同的URL，或者在它自己的缓存中查找（后面讨论）。4××码意味着由于客户错误而导致请求失败，比如无效请求或者不存在的页面。最后，5××错误码意味着服务器自身出现内部问题，有可能是服务器代码中有错误，也可能是临时负载过重。

代码	含义	例子
1××	信息	100 = 服务器同意处理客户请求
2××	成功	200 = 请求成功；204 = 没有内容
3××	重定向	301 = 移动页面；304 = 缓存的页面仍然有效
4××	客户错误	403 = 禁止页面；404 = 页面没找到
5××	服务器错误	500 = 服务器内部错误；503 = 稍后再试

图 7-38 响应组的状态码

消息头

请求行（例如 GET 方法的行）后面可能还有额外的行，其中包含了更多的信息。它们同称为**请求头**（request header）。这些信息可以与一个过程调用的参数相类比。响应消息也有**响应头**（response header）。有些头可以用在两个方向上。图 7-39 列出的是一些最重要的

头	类型	内容
User-Agent	请求	有关浏览器及其平台的信息
Accept	请求	客户可处理的页面类型
Accept-Charset	请求	客户可接受的字符集
Accept-Encoding	请求	客户可处理的页面编码
Accept-Language	请求	客户可处理的自然语言
If-Modified-Since	请求	检查新鲜度的时间和日期
If-None-Match	请求	先前为检查新鲜度而发送的标签
Host	请求	服务器的DNS名字
Authorization	请求	列出客户的信任凭据
Referer	请求	发出请求的先前URL
Cookie	请求	给服务器发回Cookie的先前URL
Set-Cookie	响应	客户存储的Cookie
Server	响应	关于服务器的信息
Content-Encoding	响应	内容如何编码（比如，gzip）
Content-Language	响应	页面使用的自然语言
Content-Length	响应	页面以字节计的长度
Content-Type	响应	页面的MIME类型
Content-Range	响应	标识了页面内容的一部分
Last-Modified	响应	页面最后修改的时间和日期
Expires	响应	页面不再有效的时间和日期
Location	响应	告诉客户向谁发送请求
Accept-Ranges	响应	指出服务器能接受的请求的字节范围
Date	请求/响应	发送消息的日期和时间
Range	请求/响应	标识一个页面的一部分
Cache-Control	请求/响应	指示如何处理缓存
ETag	请求/响应	页面内容的标签
Upgrade	请求/响应	发送方希望切换的协议

图 7-39 某些 HTTP 消息头

消息头。这个列表不短，正如你可以想象的那样，每个请求和响应通常具有不同的头。

User-Agent 头允许客户将它的浏览器实现告知服务器（比如，Mozilla/5.0 和 Chrome/5.0.375.125）。这些信息非常有用，服务器可以据此裁剪给浏览器的响应，因为不同浏览器的能力和行为大相径庭。

如果客户对于可接受的信息有一定的限制，可以用 4 个 Accept 头告诉服务器愿意接受什么。第一个头指定了哪些 MIME 类型是可以接受的（例如，text/html）。第二个头给出了字符集（比如 ISO-8859-5 或者 Unicode-1-1）。第三个头处理压缩方法（比如 gzip）。第四个头指明了一种自然语言（例如西班牙语）。如果服务器有多个页面可供选择的话，那么它可以利用这些信息向客户提供所需要的页面。如果客户的请求不能满足，则返回一个错误码并且请求失败。

If-Modified-Since 和 If-None-Match 头用于缓存。它们允许客户在网页的缓存副本不再有效时请求该网页。我们稍后将描述缓存机制。

Host 头是服务器的名字，它取自 URL。这个头是强制性质的。有些 IP 地址可能对应了多个 DNS 名字，所以服务器需要某种方法来分辨出应该把请求传递给哪个主机来处理。

对于那些被保护的页面，Authorization 头是必需的。在这种情况下，客户可能需要证明自己有权来查看被请求的页面。这个头就是用在这种情况下。

客户端使用拼写错误的 Referer 头给出了请求当前页面的 URL。大多数情况下，这是前一页的 URL。这个头对于跟踪 Web 浏览特别有用，因为它告诉了服务器客户如何到达该页面的。

尽管 Cookie 是在 RFC 2019 中而不是 RFC 2616 中被阐述的，它们也有头。Set-Cookie 头是服务器向客户端发送 Cookie，期待客户端保存 Cookie，并且在后续的请求中返回给服务器，返回时要使用 Cookie 头（请注意，针对 Cookie 最近出了个规范说明了一种新的头，即 RFC 2965。但是它在很大程度上已经遭到行业拒绝，不太可能得到广泛实施）。

响应消息中还用到了许多其他的头。Server 头允许服务器在愿意时标识构建它的软件。下面 5 个头，都从 Content-起始，允许服务器描述它发送的页面的属性。

Last-Modified 头说明了页面最后被修改的时间，Expires 头说明了页面能保持有效多长时间。这两个头在页面缓存机制中发挥着重要的作用。

Location 头被服务器用来通知客户，它应该尝试另外一个不同的 URL。如果页面已经被移动，或者允许多个 URL 指向同一个页面（可能在不同的服务器上），则可以使用这个头。它也可被用于另外一种情况，即公司的主页在 com 域中，但是根据客户的 IP 地址或者首选的语言，将客户的请求重定向到一个国家的或者地区的页面中。

如果一个页面非常大，则小客户可能不想一次获取所有的内容。有些服务器可以接受对于有字节范围限定的请求，因此该页面可以通过多个小单位的请求取回来。Accept-Ranges 头声明了服务器愿意处理这种部分页面请求。

现在我们来到了可以在两个方向上同时使用的头。Date 头可用在两个方向上，并且包含了发送消息的日期和时间，而 Range 头则告诉了响应消息提供的网页的字节范围。

ETag 头给出了一个简短的标签，作为页面内容的名称。它主要被用于缓存。Cache-Control 头给出了其他有关如何缓存（或更通常的是，如何不缓存）网页的明确指示。

最后，Upgrade 头用来切换到一个新的通信协议，比如未来的 HTTP 协议或安全的传

输协议。它允许客户宣布自己可以支持什么协议，同时允许服务器声称自己正在使用什么协议。

缓存

人们往往会返回到以前浏览过的 Web 页面，而且相关的网页往往具有相同的嵌入式资源。比如有些例子包括用于整个网站导航的图像，以及常见的样式表和脚本。如果每次显示这些页面都要去服务器获取全部的资源，这将非常浪费，因为浏览器已经有了一个副本。

积攒已经获取的网页供日后使用的处理方式称为**缓存**（caching）。其优点是当缓存的页面被重复使用时，没有必要进行重复传输。HTTP 内置了一种技术支撑，帮助客户标识他们何时可以放心地重用页面。这种支撑能减少网络流量和延迟，从而提高了性能。但现在存在一种权衡，即浏览器必须存储页面。因为本地存储成本已经很低廉，因此这几乎总是一个值得做的选项。这些网页通常保存在磁盘上，当稍后浏览器运行时可以直接使用它们。

HTTP 缓存的困难问题在于如何确定以前缓存的页面和将要重新获取的页面是相同的。这个确定不能仅依靠 URL 来作出。例如，URL 可能会给出一个页面，显示了最新的新闻项目。即使 URL 保持不变，该页面的内容也会经常更新。另外一种情况是，页面的内容可能是来自希腊和罗马神话中的诸神列表，那么这种页面的改变速度应该不那么快。

HTTP 使用两种策略来解决这个问题。这些策略如图 7-40 所示，作为请求（第 1 步）和响应（第 5 步）之间的处理形式。第一种策略是页面验证（第 2 步）。访问高速缓存，如果对所请求的 URL 它有该页面的副本，而且该副本已知是新鲜的（即仍然有效），那么就没有必要从服务器重新获取。此时，直接返回缓存的页面。该缓存页面最初获取时返回的 Expires 头以及当前的日期和时间可以被用来作出该副本是否有效的决定。

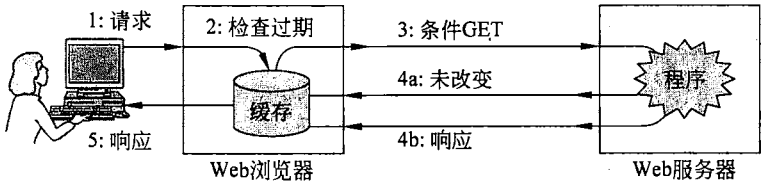


图 7-40 HTTP 缓存

然而，并非所有的网页都有个方便的 Expires 头来告诉你何时必须重新获取网页。毕竟，做出预测才是硬道理——尤其关于未来的预测。在这种情况下，浏览器可能使用启发式的方法来作出决策。例如，如果页面在过去的一年尚未修改（从 Last-Modified 头获知），那么它极有可能在接下来的一个小时里不会有所改变，这或许是一个相当安全的赌注。然而，这里没有任何保证，也许下了个坏的赌注。例如，股市已经关闭一天，因此该页面数个小时都不曾改变，但下一个交易时段开始后它会迅速改变。因此，一个页面的缓存性随着时间的推移可能变化很大。基于这个原因，应谨慎使用启发式策略，尽管它们往往实际工作得挺好。

找到没有过期的页面是使用缓存的最大收益，因为这意味着根本不需要联系服务器。不幸的是，它并不总是能工作得很好。服务器必须谨慎地使用 Expires 头，因为它们可能无法确定一个页面何时将被更新。因此，缓存的副本可能仍然是新鲜的，但客户并不知道。

第二个策略用在这种情况下。它询问服务器缓存的副本是否仍然有效。这个请求是一个**条件 GET**（conditional GET），如图 7-40 中第 3 步所示。如果服务器知道缓存的副本仍然是有效的，它可以发送一个简短的答复说是的（第 4a 步）。否则，它必须发送完整的响应消息（第 4b 步）。

更多的头字段可用来让服务器检查一个缓存的副本是否仍然有效。客户端从 Last-Modified 头可以获知缓存页面的最后更新时间。它可以使用 If-Modified-Since 头将该时间发送给服务器，询问服务器所请求的页面在此期间是否发生过改变。

另一种选择是，服务器可能会返回一个包含 ETag 头的页面。这个头给出了一个标签，是页面内容的一个短名称，有点像校验和但比它更好。（它可以是一个加密的哈希值，我们将在第 8 章中描述）。客户端可以向服务器请求验证缓存的副本，具体做法是给服务器发送 If-None-Match 头，该头列出了缓存副本的标签。如果任何一个标签与内容相匹配，服务器将会予以响应，相应的缓存的副本就可以使用。在不太方便时可以使用这种方法，这种方法对确定页面是否新鲜很有用。例如，一个服务器可能会为相同的 URL 返回不同的内容，这完全取决于首选什么样的语言和 MIME 类型。在这种情况下，仅仅一个修改日期无助于服务器确定缓存页面是否新鲜。

最后，请注意这两种缓存策略都会被 Cache-Control 头携带的指令所覆盖。当页面不恰当被缓存时，这些指令可以被用来限制缓存（例如，no-cache）。比如，下一次抓取的动态页面将是不同的，因此不能缓存；或者那些需要授权的页面也不能被缓存。

关于如何缓存的方法有很多，但我们只有空间谈两个要点。首先，缓存可以设置在除浏览器之外的其他地方。一般情况下，HTTP 请求可以通过一系列的缓存路由。使用浏览器之外的外部高速缓存方法称为**代理缓存**（proxy caching）。每增加一个缓存级别可有助于进一步减少请求链。诸如 ISP 和企业这些组织经常使用这种缓存方法，它们运行代理缓存为不同的用户提供了从缓存页面尽快获得响应的好处。我们将在本章结尾处的 7.5 节关于更广泛的内容分布主题中再来讨论代理缓存。

其次，高速缓存对提升性能是个很大的推动，但它并没有人们所希望的那么多。原因是虽然在 Web 上肯定有许多流行的文档，但也有很多人们获取的文档不是那么流行，其中很多还很长（例如，视频）。不受欢迎的“长尾巴”文档占用了缓存空间，而且缓存能处理的请求数量随着高速缓存大小的增加而增长缓慢。Web 高速缓存能够处理的请求数量可能总是达不到全部请求数量的一半。对于这方面的更多信息，请参考（Breslau 等，1999）。

HTTP 实验

因为 HTTP 是一个 ASCII 协议，所以对于一个坐在终端（浏览器的对面）前面的人来说很容易与 Web 服务器直接通话。这里所需要的全部只是一个连到服务器端口 80 的 TCP 连接。鼓励读者亲自尝试用下面的命令序列来做实验。在大多数 UNIX shell 和 Windows 的命令窗口下这些命令都可以工作（只要 telnet 程序可用）。

```
telnet  www.ietf.org  80
GET  /rfc.html  HTTP/1.1
Host: www.ietf.org
```

这个命令序列启动一个 telnet 连接（即 TCP 连接），该连接指向端口为 80 的 IETF 的

Web 服务器 www.ietf.org。然后是 GET 命令，指定了 URL 的路径和协议。尝试你自己选择的服务器和 URL。接下来一行是强制的 Host 头。最后一个头后面必须跟一个空行。它告诉服务器没有更多的请求头了。然后服务器发送响应消息。根据不同的服务器和 URL，可以观察到不同类型的头和页面。

7.3.5 移动 Web

Web 通常被大多数类型的计算机所用，这其中包括移动电话。在移动的同时通过无线网络浏览网页非常有用，但要做到这点面临着许多技术问题，因为太多的 Web 内容是为具有宽带连接并有华丽表示的台式计算机而设计的。在本节，我们将描述如何从移动设备访问 Web，或者**移动 Web**（mobile Web），这些技术正处在开发之中。

相比工作或家庭用的台式计算机，用移动电话来浏览 Web 存在几个方面的困难：

（1）相对小的屏幕妨碍了大页面和大图像的显示。

（2）有限的输入能力使得 URL 的输入很乏味或者输入法很慢。

（3）无线链路的网络带宽有限，尤其是蜂窝（3G）网络，费用还很高。

（4）网络的连通性断断续续。

（5）由于电池寿命、大小、散热以及成本等诸多原因，导致计算能力有限。

这些困难意味着简单地把台式机内容作为移动 Web 很可能会传递出一种令人懊恼的用户体验。

移动 Web 的早期方法采用了一种新的协议栈，这种协议栈专门针对能力有限的无线设备而发明出来的。**无线应用协议**（WAP，Wireless Application Protocol）是这种策略的最知名的例子。经过各大移动电话厂商的努力，1997 年 WAP 正式启动，其中包括诺基亚、爱立信和摩托罗拉公司。然而，一路上意想不到的事情发生了。在接下来的十年间，随着 3G 数据服务的部署，以及移动电话有了更大的彩色显示屏、更快的处理器和 802.11 无线功能，网络带宽和设备能力得到了巨大的提高。突然间，完全有可能在移动电话上运行简单的 Web 浏览器。虽然这些移动电话与台式机之间始终存在着一个沟壑，并且这个沟壑永远也不会消除，但是当初推动一个单独协议栈的技术问题现在已经淡化。

当前使用得越来越多的方法是移动电话和台式机运行相同的 Web 协议，并且当用户恰好在移动设备上时 Web 站点负责提供移动友好（mobile-friendly）的内容。通过查看请求头信息，Web 服务器能够检测出应该返回桌面版本的网页还是移动版本的网页。User-Agent 头在这方面特别有用，因为它标识了浏览器软件。因此，当 Web 服务器接收到一个请求时，它可能首先查看头，然后给 iPhone 返回图像小、文字少和简单的导航页面，而给笔记本电脑用户返回一个全功能的网页。

W3C 从几个方面鼓励这种移动终端和计算机通用的方法。一种方式是标准化移动 Web 内容的最佳实践（best practice）。第一个规范提供了 60 个这样的最佳实践（Rabin 和 McCathieNevile，2008）。因为通信成本高于计算成本，其中的大多数实践采取了明智的措施来减少页面的大小，包括使用压缩算法，并且最大限度地提高缓存的有效性。这种方法鼓励了网站，尤其是大型网站创建移动 Web 版本的页面内容，因为这是虏获移动 Web 用户所必须要做的事情。为了帮助这些用户，还有一个标志指示可以在移动 Web 上（很好）

浏览的页面。

另一种有用的工具是一个精简版的 HTML，称为**基本 XHTML**（XHTML Basic）。这种语言是 XHTML 的一个子集，专用于移动电话、电视机、个人数字助理、自动售货机、传呼机、汽车、游戏机，甚至手表。出于这个原因，它不支持样式表、脚本或框，但大多数的标准标签它都有。它们被分成 11 个模块。有些是必需的，而有些是可选的。所有的模块都以 XML 定义。图 7-41 列出了一些模块和标签例子。

模块	需要否	功能	标签实例
Structure	是	文档结构	body, head, html, title
Text	是	信息	br, code, dfn, em, hn, kbd, p,strong
Hypertext	是	超链接	a
List	是	明细表	dl, dt, dd, ol, ul, li
Forms	否	填入表单	form, input, label, option, textarea
Tables	否	矩形表格	caption, table, td, th, tr
Image	否	图像	img
Object	否	Java小程序、地图等	object, param
Meta-information	否	额外的信息	meta
Link	否	类似于<a>	link
Base	否	URL起始点	base

图 7-41 基本 XHTML 模块和标签

然而，并非所有的页面都能设计成在移动 Web 上工作良好。因此，一种互补的方法是使用**内容转换或转码**（content transformation or transcoding）。在这种方法中，将一台计算机设置在移动电话和服务器之间，它从移动电话获得请求，然后从服务器预取内容，最后把它转换成移动 Web 内容。一种非常简单的转换方法是减少大型图片的尺寸，将它重新格式化成一个较低的分辨率。当然还可以使用其他许多小而有用的转换方法。自从移动 Web 出现以来，转码技术的使用已经取得了一些成功。例如，有关这方面的信息可参考（Fox 等，1996）。然而，当这两种方法都使用时，究竟由服务器还是转码器来作出移动内容的决策，两者之间存在着一种紧张关系。例如，一个 Web 站点针对移动 Web 可能会选择一种图像和文字的特殊组合，只能由一个转码器来改变图像的格式。

我们讨论至今一直关注着 Web 内容，而没有涉及协议，因为实现移动 Web 的最大问题正是由内容引起的。然而，我们简要提及协议问题。Web 所用的 HTTP、TCP 和 IP 协议可能消耗掉相当数量的带宽，因为这些协议的开销不小（比如协议头）。为了解决这个问题，WAP 和其他解决方案定义了特殊用途的协议。这在很大程度上是没有必要的。报头压缩技术，比如第 6 章中表述的鲁棒报头压缩（ROHC, RObust Header Compression）可以减少这些协议的开销。通过这种方式，很可能只需要一组协议（HTTP，TCP，IP），无论是高带宽还是低带宽链路都使用这组协议。在低带宽链路上使用时只需要打开头压缩技术。

7.3.6 Web 搜索

为了完成对 Web 的描述，我们将讨论可以说是最成功的 Web 应用程序：搜索。1998 年，时任斯坦福大学研究生的 Sergey Brin 和 Larry Page 组建了一个称为谷歌（Google）的新兴公司，目标是建立一个更好的网络搜索引擎。他们秉持当时激进的想法：一个搜索算

法计算每个页面多少次被其他页面所指是评价该页面重要性的更好手段，比每个页面包含多少个被搜索关键词的手段更好。例如，许多页面链接到思科（Cisco）的主页，对一个搜索“Cisco”的用户而言，这个主页比思科以外的公司碰巧有个页面多次出现“思科”的页面要重要得多。

他们是对的。已经证明可以建立一个更好的搜索引擎，于是人们蜂拥而至。有了风险资本的支持，谷歌大幅增长。它在2004年成为一家上市公司，市值230亿美元。到2010年，据估计它分布在世界各地的数据中心有超过100万台服务器在运行。

从某种意义上说，搜索仅仅是另一个Web应用程序，尽管它是最成熟的Web应用程序之一，因为自Web初期以来它一直在被不断地开发。然而，Web搜索已经成为日常生活中不可缺少的应用。估计每天有超过10亿个网页被搜索。寻找所有其他信息方式的人把搜索作为起始点。例如，要找出在西雅图哪里有卖蔬菜和调味品，刚开始没有明显的网站可以使用。但机会存在于搜索引擎知道一个网页具有所需的信息，因而可以快速地引导你找到答案。

要以传统的方式进行Web搜索，用户将她的浏览器指向一个Web搜索网站的URL。主要的搜索网站包括谷歌、雅虎和Bing。接下来，用户使用表单提交搜索词。这种行为导致搜索引擎对它的数据库执行一次查询，搜索相关的页面或图像，或者想搜索的任何类型的资源，并且将查询结果作为一个动态页面返回。然后，用户可以按照链接找到已被发现的网页。

Web搜索是进行讨论的一个有趣话题，因为它对网络的设计和使用有很大影响。首先，存在一个Web搜索如何找到网页的问题。Web搜索引擎必须有一个运行查询算法的页面数据库。每个HTML页面可能包含指向其他页面的链接，而且每一件有趣的事情（或者至少是可搜索的）链接在某个地方。这意味着，从少数页面开始通过遍历所有网页和链接，在理论上可能找出Web上的所有其他页面。这个过程称为**Web爬虫**（Web crawling）。所有的Web搜索引擎均使用**Web爬虫程序**（Web crawler）。

与爬虫有关的一个问题是它能找到什么样的页面。获取静态文件，然后跟随链接很容易做到。然而，许多网页包含一些程序，根据用户的交互显示不同页面。其中一个例子是在线商店的目录。该目录可能包含根据产品数据库创建的动态页面，以及针对不同产品的查询。这种类型的内容不同于很容易遍历的静态页面。Web爬虫如何找到这些动态页面？答案是，在大多数情况下，它们抓取不到。这种隐藏的内容称为**深层Web**（deep Web）。如何搜索深部Web是一个开放的问题，研究人员正在努力解决之。例如，有关这方面的信息可参考（Madhavan等，2008）。也有一些约定，网站提供一个页面（称为robots.txt）告诉爬虫程序该站点的哪些部分应该或者不应该访问。

第二个需要考虑的问题是如何处理所有爬虫抓到的数据。为了使得索引算法能在大量的数据上运行，页面必须被有效存储起来。虽然估计值在不断地变化，但主要的搜索引擎被认为具有数百亿页面的索引，这些页面取自Web的可见部分。平均页面大小大约为320 KB。这些数字意味着，抓取一个Web的副本需要20 PB量级或2×10^{16}个字节量级的存储量。虽然这是一个真正庞大的数字，但它是Internet数据中心可以轻松存储和处理的数据量（Chang等，2006）。例如，如果磁盘的存储成本20美元/TB，那么2×10^{4} TB的成本为*400 000*美元，这个数字对于像谷歌、微软和雅虎这样的公司实在不能算大。虽然网

络还在扩大，但磁盘的成本也在急剧下降，因此存储整个 Web 在可预见的未来对于大公司仍然是可行的。

要使这些数据有意义则是另外一回事。你或许欣赏 XML 如何帮助程序轻松地提取数据结构，自主特设的格式会导致很多猜测。还有格式之间的转换问题，甚至是语言之间的翻译。但是，即使知道数据的结构也只是解决了问题的一部分。明白它的意思才是硬道理。从更相关的结果页面开始搜索查询，这是可以获得更大价值的关键所在。最终目标是要能够回答问题，例如，在你城市的哪里能买到便宜但像样的烤箱。

Web 搜索的第三个方面是它已经提供了更高层次的命名。没有必要记住一个长长的 URL，假设你记名字比记 URL 记得更好，那么用一个人的名字来搜索网页一样可靠（或者更多）。这一策略越来越成功。犹如 DNS 域名贬低了计算机的 IP 地址，Web 搜索降低了计算机的 URL。此外，有利于搜索的是它能纠正拼写和输入错误，否则如果你输入了一个错误的 URL，你只能得到错误的页面。

最后，Web 搜索向我们显示的事情很少与网络设计有关，但与某些 Internet 服务增长有关的是：广告中有太多的钱。广告是驱动 Web 搜索增长的经济引擎。从印刷广告到 Web 广告的主要变化在于：Web 广告具备根据人们正在寻找的对象来发放广告的能力，以此提高了广告的相关性。拍卖机制的变异方法被用来匹配搜索查询与最有价值的广告（Edelman 等，2007）。当然，这种新的模式已经引起了新的问题，比如“点击欺诈”，在这种情况下，程序模仿用户点击广告造成没有真正赢利的报酬。

7.4 流式音视频

Web 应用和移动 Web 并不是网络应用领域中唯一令人振奋的发展。对许多人来说，音频和视频才是网络的圣杯。当提及“多媒体”这个词时，无论技术人员和商人都会不约而同地垂涎三尺。前者看到的是为每一台计算机提供 IP 语音和视频点播所隐含的巨大技术挑战，而后者则看到它同样隐含的丰厚的利润。

虽然至少从 20 世纪 70 年代开始就有了通过 Internet 发送音频和视频的想法，但大约自 2000 开始**实时音频**（real-time audio）和**实时视频**（real-time video）流量才有了真正极度的增长。实时流量与 Web 流量有很大的不同，它必须以预先设定的速率播放才有用。毕竟，观看一个忽停忽动的慢动作视频不是大多数人的乐趣。相反，Web 可以有短暂的中断，而且页面加载可以花费更多或更少的时间，在一定限度内这不是一个主要的问题。

发生的两件事情极大地促进了这种实时流量的增长。首先，计算机变得更加强大，并且都配备了麦克风和摄像机，因此它们很容易地就能够输入、处理和输出音频及视频数据。其次，Internet 带宽的大量扩张已经大大提高了 Internet 的可用性。Internet 核心骨干网的长途链路具有许多个 kbps，而且宽带和 802.11 无线已经达到 Internet 边缘用户。这些发展使 ISP 可以在它们的骨干网携带巨大的流量，而且这意味着普通用户可以用比 56 kbps 的电话调制解调器快上 100～1000 倍的速度连接到 Internet。

带宽扩张导致音频和视频流量的增长，但这里也有不同的原因。电话呼叫占用相对较少的带宽（原则上是 64 kbps，压缩后更少），但电话服务费历来比较昂贵。公司从中看到

了契机，如果使用现有的带宽通过 Internet 承载语音流量，那么就可以减少他们的电话账单。比如 Skype 这样的新兴公司找到了一个方法，让客户使用他们的 Internet 连接拨打免费电话。傲慢自负的电话公司也找到了一种廉价方式，使用 IP 网络设备进行传统的语音通话。其结果是通过 Internet 网络承载语音数据的服务发生了大爆炸，这种方式称为 **IP 语音**（voice over IP）或 **Internet 电话**（Internet telephony）。

与音频不同，视频占用了大量的带宽。合理质量的 Internet 视频编码压缩率大约为 1 Mbps，一个典型 DVD 电影有 2 GB 的数据。在宽带 Internet 接入前，通过网络发送电影简直令人望而却步。现在已今非昔比了。随着宽带的传播，用户在家观看令人满意的视频流终于成为可能，对此人们乐此不疲。对任何一天的估计表明大约四分之一的 Internet 用户会访问 YouTube，这是一个广受欢迎的视频共享网站。电影租赁业务已经转移到在线下载。而且规模庞大的视频改变了 Internet 流量的整体构成。大多数 Internet 流量早已经是视频，据估计在几年内 90% 的 Internet 流量将是视频（Cisco，2010）。

由于有足够的带宽来承载音频和视频，那么设计音视频流和视频会议应用的关键问题就是网络的延迟。音频和视频需要实时演示，这意味着它们必须以预定速率播放出来才有用。长延迟意味着应该互动的呼叫不再能互动得起来。这个问题是很清楚的，如果你曾经用过卫星电话就应该有所体会，当延迟达到半秒钟时相当的令人分散注意力。在网络上播放音乐或电影，绝对的延迟并不要紧，因为它仅仅影响到媒体何时开始播放。但延迟的变化——称为**抖动**（jitter），仍然非常重要。抖动必须被播放器掩盖，否则播放出来的音频听起来令人难以理解，播放出来的视频则会看上去很生涩或呆滞。

在本节中，我们将讨论如何处理延迟问题的某些策略，以及用来设置音频和视频会话的协议。在介绍完数字音频和视频后，我们的介绍将分成三种情况，分别采用了不同的设计。第一，也是最简单的情况是如何处理存储的流媒体，就像观看 YouTube 上的视频一样。接下来的一种情况难度适中，就是如何处理流媒体直播。流媒体直播有两个例子，分别是 Internet 广播和 IPTV，其中广播电台和电视台通过 Internet 给许多用户进行直播。最后，也是最困难的情况是如何处理 Skype，或者更普遍的交互式音频和视频会议呼叫。

顺便说一句，**多媒体**（multimedia）这个术语经常出现在 Internet 的上下文中，它意味着视频和音频。从字面上看，多媒体是两个或两个以上的媒体。按照这个定义，这本书就是一个多媒体范例，因为它包含了文字和图形（数字）。然而，这可能不是你心目中的多媒体，因此我们使用“多媒体”一词意味着两个或两个以上的**连续媒体**（continuous media），也就是说，媒体必须以某种良好定义的时间间隔播放。两个媒体通常指有音频的视频，也就是说，有声音的移动图像。许多人把单纯的音频称为多媒体，比如 Internet 电话或 Internet 广播，这显然不是很恰当。其实，所有这些情况下应该使用的一个更好术语是**流媒体**（streaming media）。尽管如此，我们跟随大流，把实时音频也当作多媒体来考虑。

7.4.1 数字音频

音频（声音）波是一种一维声（压力）波。当声波进入耳朵时，鼓膜发生振动，从而引起内耳的微细感骨与之一起振动，并向大脑发送神经脉冲。听者感觉到的这些脉冲就是声音。类似的方式是当声波碰击麦克风时，麦克风产生一个电信号，该电信号将声音振幅

表示为时间的函数。

人耳能听到的声音频率范围是20～20 000 Hz。有些动物，特别是狗，能够听到更高的频率。耳朵听见的声音符合对数关系，因此，功率为A和B的两种声音的比率，通常被表示成**分贝**（dB，decibels）：$dB=10\log_{10}(A/B)$。如果我们将1 kHz正弦波的能听度低限（约20微帕的声音压力）定义为0 dB，那么普通谈话大约是50 dB，耳朵感觉到痛的阈值大约是120 dB。可见变化的范围达100万倍。

令人惊讶的是人耳对于持续时间仅仅几毫秒的声音变化也异常的敏感。相反地，眼睛却察觉不到持续时间为几毫秒的光度变化。这种观察的结果是多媒体播放中仅仅几毫秒的抖动对于可感觉到的声音质量的影响，要远远大于对可感觉到的图像质量的影响。

数字音频是声波的数字表示，这种表示能重现音频。通过**模数转换器**（ADC，Analog Digital Converter）可以将音频波转换成数字形式。ADC以电压作为输入，生成一个二进制数作为输出。在图7-42(a)中，我们看到一个正弦波的例子。为了用数字方式来表示该信号，我们可以每隔 ΔT 秒对它采样一次，如图7-42(b)中的竖条状所示。如果一个声波不是纯正弦波，而是若干正弦波的线性叠加，并且其中最高的频率成分为f，那么，根据尼奎斯特定理（参见第2章），以2f的频率对该信号采样足够了。更高的采样频率没有意义，因为更高采样获得的更高频率这里无法检测出来。

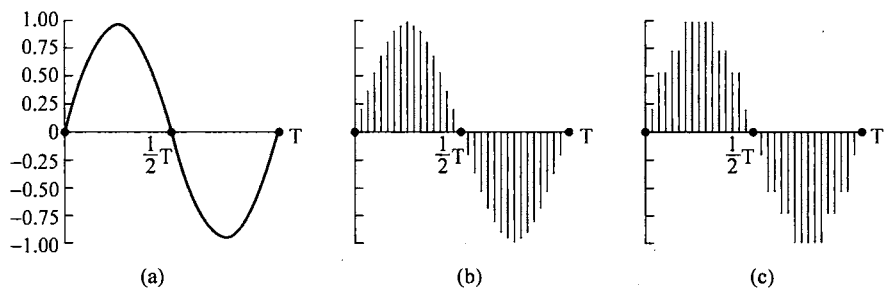


图 7-42

(a) 一个正弦波 (b) 正弦波样值 (c) 量化成4位

相反的过程是获取数字值并且产生一个模拟电压。这个过程由**数模转换器**（DAC，Digital-to-Analog Converter）来完成。然后，扬声器将模拟电压转换成声波发送出去，因此人们就能听到声音了。

数字采样永远不可能做到精确。图7-42(c)中的样值只允许9个值，从-1.00～+1.00，步长为0.25。一个8位的采样允许有256个不同的值。一个16位的采样可以有65 536个不同的值。由于每个采样值的位数有限而引入的误差称为**量化噪声**（quantization noise）。如果量化噪声太大，人耳就能觉察得出来。

使用声音采样技术的两个众所周知例子是电话和音频**压缩光盘**（CD，compact disc）。电话系统使用的脉冲编码调制，以每秒8000次的频率采样，并用8位表示采样值。为了最小化感觉到的失真，量化尺度是非线性的；而且限定在每秒只有8000采样，任何高于4 kHz的频率都将被丢弃。在北美和日本，采用了μ-l规则编码技术；在欧洲和国际上则广泛采用A-l规则编码技术。每种编码技术的数据率都是64 000 bps。

音频CD是数字化的，其采样率为每秒44 100次，这足以捕获高达22 050 Hz的频率；

22 050 Hz 对于人来说已经非常好了，但对于犬类音乐爱好者而言却很糟糕。每个样值都用16位表示，并且在振幅范围内量化尺度是线性的。但是请注意，16位样值只有65 536个不同的值，甚至人耳可识别的动态声音范围大约为100万。因此，纵使CD质量的音频远远好于电话质量的音频，但如果每个样值仅使用16位，因而引入了一定的量化噪声（尽管CD并没有覆盖整个动态范围——CD不应该会刺伤或刺痛人耳）。一些狂热的音乐发烧友还是喜欢33-RPM的LP唱片比CD更多一些，因为唱片没有截取22 kHz的尼奎斯特频率，而且没有量化噪声（但必须非常小心放置，否则会有划痕）。在每秒44 100个采样并且每个样值16位的情况下，无压缩的CD质量音频单声道需要705.6 kbps的带宽，立体声则需要1.411 Mbps的带宽。

音频压缩

为了减少带宽需求和传输时间，音频往往被进行了压缩处理，即使音频数据率比视频数据传输速率要低得多。所有的压缩系统需要两种算法：一种用在源端上压缩数据，另一种用在接收方上解压数据。在文献中，这些算法分别称为**编码**（encoding）算法和**解码**（decoding）算法。我们将沿用这个术语。

压缩算法表现出一定的不对称性，理解这点非常重要。我们首先考虑音频，这种不对称性对于视频也是适用的。对于许多应用程序来说，多媒体文件只被编码一次（当它存储在多媒体服务器上时），但将被解码几千次（每当被客户播放时）。这种不对称性意味着，编码算法较慢而且需要昂贵的硬件是可以接受的，只要解码算法速度快而且不需要昂贵的硬件。一个流行音频（或视频）服务器的运营商可能会很乐意买一个计算机集群，来为它的整个数字图书馆编码，但如果要求客户做同样的事情才能听到音乐或看到电影，那么该运营商恐怕不会成功。许多实用的压缩系统竭尽全力使得解码算法既快速又简单，甚至以编码算法缓慢而且复杂性高作为代价。

另一方面，对于直播的音频和视频，比如IP语音呼叫，缓慢的编码是不可接受的。编码必须实时地动态执行。因此，相比存储在磁盘上的音频或视频，实时多媒体使用不同的算法或参数，往往适当减少压缩。

第二个不对称性在于编码/解码过程不必是可逆的。也就是说，压缩数据文件、传输、然后解压缩它，用户期望得到原始的准确的数据，直到最后一位数据。对于多媒体而言，这一要求并不一定存在。音频（或视频）信号经过编码，然后再被解码，得到的信号与原始的略有不同，这通常是可以接受的，只要它听起来（或看起来）和原始的一样即可。当解码输出的信号不完全等同于原始输入的信号，就说该系统是**有损耗的**（lossy）。如果输入和输出完全相同，则该系统就是**无损的**（lossless）。有损系统很重要，因为接受少量信息的丢失，通常意味着在可能的压缩率方面能获得巨大的回报。

从历史上看，电话网络中的长途带宽非常昂贵，所以工作中需要一个坚实的声码器（是“语音编码器”的简称），**声码器**（vocoder）负责为特殊语音进行压缩音频。人的讲话往往在600～6000 Hz范围内，由一个机械过程产生；这个过程取决于讲话人的声道、舌头和下颚。一些声码器利用发声系统的模型来减少语音，只需要使用几个参数（例如，各种腔的大小和形状）和低至2.4 kbps的数据传输速率。然而，这些声码器是如何工作的超出了本书的范围，这里不再赘述。

我们主要关注通过 Internet 发送音频，一种通常接近 CD 音质的音频。通过 Internet 发送时必须减少这种音频的数据传输速率。1.411 Mbps 的立体声音频会占用许多宽带链路，给视频和其他 Web 流量留下的余地很少。音频压缩后的数据率可降低一个数量级，而用户几乎感觉不到音质的损失。

压缩和解压需要对信号进行处理。幸运的是，数字化的声音和电影可以很容易由计算机的软件处理。事实上，完成这件事情的程序已经有数十个，它们允许用户记录、显示、编辑、合成和存储来自多个源的流媒体。这种技术的应用已经导致了大量的音乐和电影可以通过 Internet 唾手可得——当然，并不是所有这些都是合法的——因此，这又导致了许多来自艺术家和版权拥有人的诉讼。

已经开发了许多音频压缩算法。可能最流行的格式是 **MPEG 音频层 3**（MP3，MPEG audio layer 3）和**高级音频编码**（AAC，Advanced Audio Coding），后者是由 MP4（MPEG-4）文件给出的。为了避免混淆，需要注意的是 MPEG 提供了音频和视频压缩。MP3 是指 MPEG-1 标准中的音频压缩部分（第三部分），而不是第三个版本的 MPEG。事实上，没有发布过 MPEG 的第三个版本，只有 MPEG-1、MPEG-2 和 MPEG-4。AAC 是 MP3 的继任者；也是 MPEG-4 中使用的默认音频编码。MPEG-2 允许 MP3 和 AAC 音频。现在说清楚了吗？有关标准的好处就是有这么多的选择。如果你不喜欢它们之中的任何一种，只有再等上一年或两年。

音频压缩可以通过两种方式完成。在**波形编码**（waveform coding）中，它通过傅里叶变换，将信号数学地转换成频率分量。在第 2 章中，我们用图 2-1(a)所示的例子显示了一个时间函数以及它的傅里叶变换。然后以最小的方式对每个分量的强度进行编码。这样做的目标是尽可能在另一端以尽可能少的位数重现该波形。

另一种方式称为**感知编码**（perceptual coding），它利用了人类听觉系统中的某些特定缺陷来对信号进行编码，即使通过示波器观察到的信号与原信号有相当大的差异，但在人耳听起来还是一样的。感知编码方法建立在**心理声学**（psychoacoustics）的基础上，即人类是如何感觉到声音的。MP3 和 AAC 都是基于感知编码的。

感知编码的关键特性是某些声音可以**屏蔽**（mask）其他声音。请想象在一个温暖的夏日里，你正在现场直播一场长笛音乐会。然后，突然间，附近的一群工人打开他们的手提电钻开始在街道上施工。于是，没有人能够听得到长笛的声音。长笛的声音被手提电钻完全屏蔽了。如果仅仅考虑传输的话，则只需对手提电钻所使用的频段进行编码就足够了，因为听众反正也听不到长笛的声音。这种情形称为**频率屏蔽**（frequency masking）——某一频段中较大的声音隐藏了另一个频段中较柔和的声音，这种能力即为频率屏蔽。事实上，即使手提电钻停下来，也有很短的一段时间听不到长笛的声音，因为当手提电钻工作的时候耳朵降低了它的增益，它需要一段有限的时间再次提高增益。这种效果称为**暂时屏蔽**（temporal masking）。

要使这些效果更明显，请想象做实验 1。一个人在一个安静的房间里，戴着一个连接到计算机声卡的耳机。计算机生成一个较弱的 100 Hz 的纯正弦波信号，然后逐渐增加功率。下达的科目要求他听到声音的时候就敲击一个键。计算机记录下此时的功率值，然后用 200 Hz、300 Hz 和其他所有人类听力范围以内的频率重复这个实验。把多个人的数据进行平均，从而得到了如图 7-43(a)所示的一个对数（log-log）图，它表示一个音调被人耳听见

需要多少功率。根据图中的曲线，可以直接得出的一个结论是永远不需要对一个功率低于可听度阈值的频率进行编码。例如，在图 7-43(a)中。如果 100 Hz 的功率是 20 dB，那么可以在输出中省略掉它而不会产生可感觉的品质损失，因为在 100 Hz 频率上 20 dB 低于可听度水平。

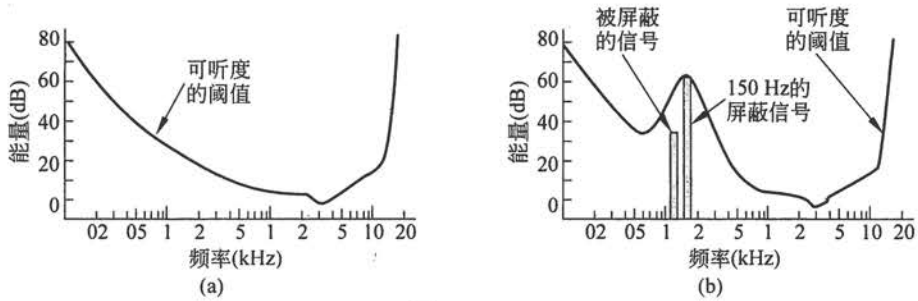


图 7-43

(a) 作为频率函数的可听度阈值；(b) 屏蔽的效果

现在考虑实验 2。计算机再次运行实验 1，但这次在测试频率上叠加了一个固定振幅的正弦波，它的频率是 150 Hz。我们发现，在 150 Hz 附近频率的可听度阈值增加了，如图 7-43(b)所示。

这次新观测的结果是记录下哪些信号被附近频段中更强的信号所屏蔽，然后在编码信号时省略掉这些越来越多的频率，从而节省数据位。在图 7-43 中，125 Hz 的信号可以在输出中被完全省略，而且没有人能够听得出差异来。在某个频段中，即使当一个功率较强的信号停止以后，考虑到它的暂时屏蔽特性，我们可以继续省略被屏蔽掉的频率一段时间（即耳朵恢复听力的时间）。MP3 和 AAC 的本质就是对声音进行傅里叶变换以便得到每个频率处的功率，然后只传输未被屏蔽的频率，并且用尽可能少的位数对它们进行编码。

有了这个信息作为背景，我们现在可以来看编码是如何完成的。音频压缩是以 8～96 的采样率对波形进行采样，以此模仿 CD 音质的声音，一般的采样率为 44.1 kHz。采样可以对一个信道（单声道）或两个（立体声）信道进行。下一步要选择输出比特率。MP3 可以将一个立体声摇滚乐光盘压缩到 96 kbps，带来感官上的音质损失很小，甚至摇滚乐迷也听不大出来。对于一场钢琴音乐会，至少需要 128 kbps 的 AAC。这两种情况下需要的比特率之所以不同的原因在于摇滚乐的信号信噪比要比钢琴演奏会的信噪比高得多（无论如何这是工程上的意义）。它也可以选择较低的输出率，并且接受某些质量上的损失。

采样值以小批量的形式处理。每个批次的信号通过一排数字滤波器以便获得频段。接着频率信息再被送入一个心理声学模型，目的是确定屏蔽掉的频率；然后，对可用比特在频段之间进行划分：给具有最大频谱功率的无屏蔽频段分配更多的比特，给具有小频谱功率的无屏蔽频段分配较少的比特，不给任何被屏蔽的频段分配任何比特。最后，使用 Huffman 编码算法对比特进行编码，给经常出现的数字分配短码，给那些不经常发生的数字分配长码。对此具有好奇心的读者可能希望介绍更多的细节。欲了解有关的更多信息，请参阅（Branden-burg，1999）。

7.4.2 数字视频

现在我们知道了所有关于耳朵的事情，是时间把注意力转移到眼球上了（当然本节不会在眼睛之后再转到鼻子上）。人类的眼睛有个特点，当一幅图像出现在视网膜上时，图像在退化之前会在视网膜上停留数毫秒。如果一个图像序列以每秒 50 幅的速率出现，眼睛是不会注意到这些图像是离散的。所有视频系统正是利用了这个原理来产生动画效果的。

视频最简单的数字表示法是一个帧序列，每个帧包含一个由图像元素或者**像素**（pixel）组成的矩形网格。每个像素可以是一个单独的位，表示黑或者白。然而，这样一个系统的质量极差。你可以试着用一个熟悉的图像编辑器将一张彩色图像的相片转换成黑白形式（而且不能有灰度阴影），看看效果如何。

接下来，每个像素用 8 位来表示 256 个灰度级。这种方案可以表示高质量的黑白（black-and-white）视频。对于彩色视频，许多系统使用 8 位来表示红、绿和蓝（RGB）三原色中的每一个分量。这种表示法是可能的，因为任何一种颜色可以通过线性叠加适当强度的红、绿、蓝构造出来。每个像素用 24 位来表示，就有大概 1600 万种颜色，远远超出了人类眼睛可以识别的范围。

在彩色 LCD 计算机显示器和电视机上，每个离散的像素是由紧密排列的红色、绿色和蓝色子像素组成的。帧由子像素的强度来显示，而眼睛混合了颜色分量。

常见的帧速率是 24 帧/秒（继承自 35 毫米的电影胶片）、30 帧/秒（取自 NTSC 美国电视）和 25 帧/秒（取自几乎全世界都在用的 PAL 电视系统）。（对于真正的挑剔者来说，NTSC 彩色电视的确切运行速率是 29.97 帧/秒。最原始的黑白系统以 30 帧/秒速率运行，但在引入颜色时，工程师需要一点额外的带宽作为信令，所以他们将帧速率减少到 29.97。NTSC 制式的视频在计算机上真正使用的帧速率还是 30）。PAL 制式在 NTSC 之后被发明出来，实际使用 25.000 帧/秒。为了使得这个故事看起来更完整，还有第三个系统，即法国、法属非洲和东欧使用的 SECAM。它是由当时的共产主义东德引进到东欧的，因此东德人无法观看西德（PAL）电视，以免他们得到不好的想法。但是现在许多东欧国家正在往 PAL 制式切换。技术和政治两个方面都在尽力而为地努力着。

事实上，对于广播电视而言，用 25 帧/秒的速率来播放流畅的动作效果还不够好，因此图像被分割分为两个**域**（field），一个由奇数扫描线构成，另一个由偶数扫描线构成。两个域（一半的分辨率）按顺序播放，给出接近 60（NTSC）/秒或确切地 50（PAL）域/秒的速率，这样的一个系统就称为**隔行扫描**（interlacing）。在计算机上观看的视频是**渐进的**（progressive），也就是说不使用隔行扫描，因为计算机显示器在图形卡上有个缓冲区，可使得 CPU 以 30 次/秒的速率将新图像放入缓冲区中，但图形卡以 50 或者甚至 100 次/秒的速率重绘屏幕来消除闪烁。模拟电视机没有像计算机那样的帧缓冲区。当快速运动的隔行扫描视频显示在计算机上时，在阴影边缘能隐约看到短的水平线，这就是称为**梳理**（combing）的效果。

通过 Internet 发送视频的帧的大小差异很大，个中的原因很简单，因为较大的帧需要更多的带宽，而这个要求可能并不总能满足。低分辨率的视频可能只有 320×240 像素，“全屏”视频是 640×480 像素。这些尺寸分别近似于早期的计算机显示器和 NTSC 电视。宽高

比（aspect ratio），或宽度与高度的比例为4：3，与标准电视一样。**高清晰电视**（HDTV，High-Definition TeleVision）视频可下载为1280×720像素。这些“宽屏”图像的宽高比为16：9，更加接近电影的3：2宽高比。相比之下，标准的DVD视频通常是720×480，蓝光光盘的视频通常是高清1080×720。

在Internet上，像素的数目仅是故事的一部分，因为媒体播放器可以不同的大小来表示相同的图像。视频只是在计算机屏幕上的另一个窗口，它可以被拉大或缩小。像素越多，其作用是提高了图像质量，所以当它被扩大时看起来并不模糊。然而，许多显示器可以显示甚至比HDTV更多的像素图像（和视频）。

视频压缩

从我们关于数字视频的讨论可以很明显地看出，通过Internet发送视频的关键是压缩。即使是标准质量的视频，即帧具有640×480像素、每个像素有24位的颜色信息、播放速率为30帧/秒，这样的视频也需要超过200 Mbps。这个数字远远超过了大多数公司办公室连接到Internet的带宽，更不用说家庭用户；而且这还只是单一视频流。由于传输无压缩视频几乎根本不可能，至少在广域网络上如此，那么唯一的希望是进行大规模的压缩。幸运的是，过去几十年大量的研究已经产生了许多压缩技术和算法，它们使得通过Internet传输视频成为可能。

通过Internet发送的视频使用了多种格式，一些是专有的，还有一些是标准化的。最流行的编码是各种形式的MPEG。它是一个开放标准，适用以mpg和mp4为扩展名的文件，以及其他容器格式的文件。在本节，我们将通过MPEG来学习如何完成视频压缩。首先，我们将着眼于用JPEG对静止图像进行压缩。一个视频仅仅是一个图像序列（加上声音）。压缩视频的方式之一就是连续编码每一幅图像。第一个近似方法就是MPEG，它只是对每一帧进行JPEG编码，再加上一些额外的功能，比如消除跨帧的冗余度。

JPEG标准

联合图像专家组（JPEG，Joint Photographic Experts Group）标准用来压缩连续色调的静止图像（比如照片），该标准是由图像专家组在ITU、ISO和IEC（另一个标准化组织）的共同主持下完成的。它已得到广泛应用（扩展的jpg文件），并且经常提供的压缩比为10：1，对自然图像的压缩效果更好。

JPEG由国际标准10918定义。实际上，相比一个单一的算法，它更像一个购物清单。它所定义的四个模式中，只有一种有损顺序模式与我们的讨论有关。我们将重点关注JPEG被通常用于24位RGB视频图像编码的方式，而且为简单起见还会留下一些选项和细节供读者自己进一步查阅。

该算法如图7-44所示。第1步是块准备。为了说明特征，让我们假设JPEG的输入是一幅640×480的RGB图像，具有24位/像素，如图7-45（a）所示。RGB并不是压缩使用的最好颜色模型。视频信号中的眼睛对**亮度**（luminance）或亮光，比对**色度**（chrominance）或颜色要敏感得多。因此，我们首先从R、G和B分量计算亮度Y，然后再计算两个色度Cb和Cr。下面公式用于8位值，变化范围从0到255：

$$Y = 16 + 0.26R + 0.50G + 0.09B$$

$$Cb = 128 + 0.15R - 0.29G - 0.44B$$

$$Cr = 128 + 0.44R - 0.37G + 0.07B$$

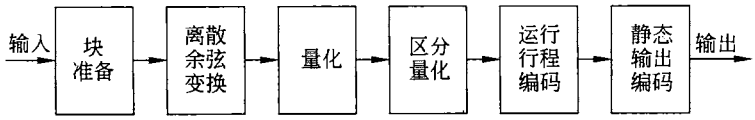


图 7-44 JPEG 有损顺序编码的步骤

为 Y、Cb 和 Cr 分别构造一个独立的矩阵。接下来，在 Cb 和 Cr 矩阵中，针对每四个像素组成的方块计算其平均值，以便将矩阵减小至 320×240。这种缩减是有损的，但是眼睛几乎不会注意到，因为眼睛对亮度比对色度更加敏感。然而，这样做却能够把整个数据量压缩一半。现在从这三个矩阵的每个元素中减去 128，以便使元素范围的中间值变成 0。最后，每个矩阵被分割成 8×8 的块。Y 矩阵有 4800 个块；其他两个矩阵有 1200 个块，如图 7-45(b)所示。

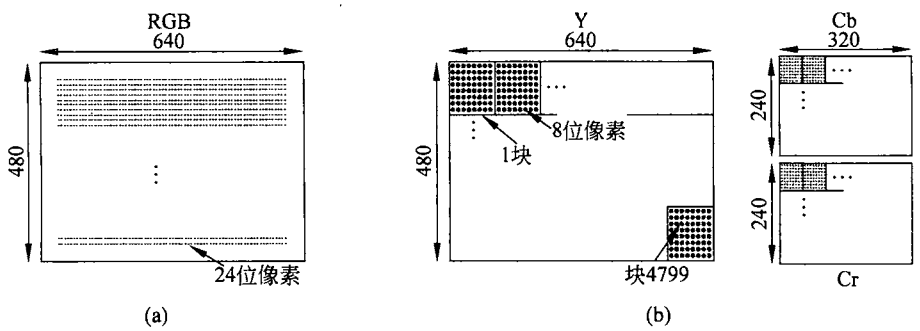


图 7-45
(a) RGB 输入数据；(b) 块准备之后

JPEG 的第 2 步是对 7200 个块中的每一个块单独应用**离散余弦变换**（DCT，Discrete Cosine Transformation）。每个 DCT 的输出是一个 8×8 的 DCT 系数矩阵。DCT 元素(0, 0)是每个块的平均值。其他的元素表明了在每个空间频率处的频谱功率。通常，随着元素离原点(0, 0)的距离越来越远，它们的值也迅速地减小，如图 7-46 所示。

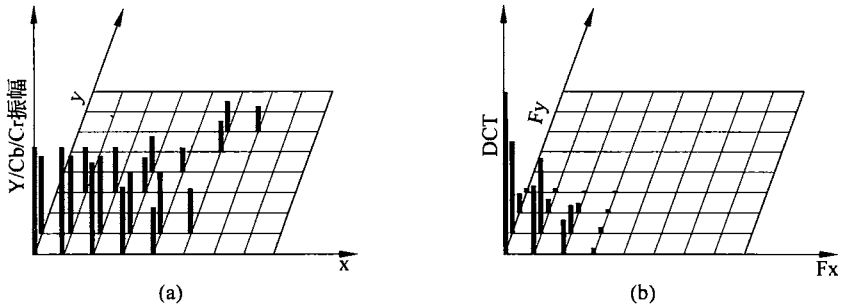


图 7-46
(a) Y 矩阵的一块；(b) DCT 系数

一旦完成 DCT，JPEG 编码进入第 3 步，这一步称为**量化**（quantization）。在这一步中，一些不那么重要的 DCT 系数被除掉。这种（有损）变换是这样来完成的：将 8×8 DCT 矩阵中的每个系数除以一个权值，而该权值是通过查询一张表获得的。如果所有的权值都是

1，则变换实际上没有做任何事情。然而，如果权值表从原点开始迅速递增，则较高的空间频率将被快速丢弃。

图 7-47 给出了这一步的一个例子。在这里，我们可以看到初始的 DCT 矩阵、量化表，以及每个 DCT 元素除以量化表中对应元素之后得到的结果。量化表中的值不是 JPEG 标准的一部分。每个应用必须提供它自己的量化表，从而允许它自己来控制损失-压缩之间的权衡。

DCT系数

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	28	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

量化表

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

C量化后的系数

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

图 7-47 量化 DCT 系数的计算

第 4 步将每一块的(0, 0）元素的值用它与前一块中对应元素的差值来替代。因为这些元素是它们各自块的平均值，它们应该变化得很缓慢，因此，采用差值之后可以把它们中的大部分减成很小的数值。其他的元素不计算差值。

第 5 步将 64 个元素排列起来，并且对此列表使用行程编码方法。按照从左到右再从上往下的方法对块进行扫描，这种方式不会将 0 集中到一起，所以 JPEG 采用了“Z”字形的扫描模式，如图 7-48 所示。在这个例子中，“Z”字形扫描模式在矩阵末端产生 38 个连续的 0。这个 0 串能够被减少为一个计数值，即由计数值来表明这 38 个 0，这项技术即为**行程编码**（run-length encoding)。

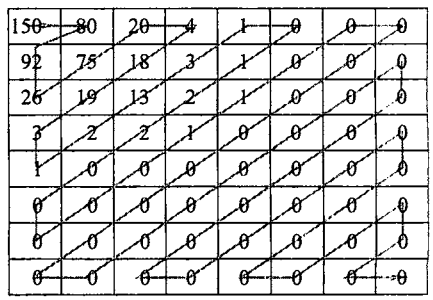


图 7-48 量化之后值的发送次序

现在我们有了一列数值来代表图像（在变换空间中)。第 6 步对这些数值进行霍夫曼编码，以便于存储或传输；给常见的数值分配较短的编码，即不常见的数值被分配较长的编码。

JPEG 或许看起来很复杂，它的确如此，因为它确实复杂。然而，能够达到 20∶1 的压缩效率也是物有所值。解码 JPEG 图像需要运行后向算法。JPEG 算法基本上是对称的：解码需要和编码花一样长的时间。正如我们将看到的那样，并不是所有的压缩算法都具有这种特性。

MPEG 标准

最后，我们来到问题的核心：**运动图像专家组**（MPEG，Motion Picture Experts Group）标准。虽然研究者开发出了许多专用的算法，但这些标准定义了压缩视频使用的主要算法。它们从 1993 年开始已经成为国际标准。因为电影同时包含了图像和声音，所以，MPEG 既可以压缩音频同时也可以压缩视频。我们已经讨论过音频压缩和静止图像的压缩，现在来讨论视频压缩。

标准 MPEG-1（它包括了 MP3 音频）首次发布于 1993 年，从此获得了广泛的应用，直到今天依然如此。它的目标是产生录像机品质的输出，而这个输出必须被压缩 40∶1 才能达到 1 Mbps 左右的速率。这样的视频适应用户通过 Internet 广泛使用 Web 站点。如果你不记得视频录像机也不必焦虑——如果有的话，MPEG-1 可用来在 CD 上存储电影。如果你连 CD 是什么也不知道，我们将很快转移到 MPEG-2。

标准 MPEG-2 发布于 1996 年，它的设计目标是压缩广播品质的视频。现在它已非常流行，因为它是 DVD 视频编码（这是通向 Internet 无法回避的方式）和数字广播电视（作为 DVB）的基础。DVD 品质的视频一般的编码速率是 4～8 Mbps。

MPEG-4 标准有两种视频格式。第一种格式发布于 1999 年，以基于对象的表示来编码视频。它允许自然图像和合成图像，以及其他各类媒体的混合，例如，一个气象预报员站在气候地图之前。有了这种结构，很容易地让程序与电影数据交互。第二种格式于 2003 年发布，称为 **H.264** 或**高级视频编码**（AVC，Advanced Video Coding）。它的目标是编码率为早期相同品质水平的视频编码率的一半，以便更好地支持通过网络进行视频传输。这个编码器用于大多数蓝光光盘的 HDTV。

所有这些标准的细节非常多而且种类繁多。后来的标准相比以前的标准有许多更多的功能和编码选项。不过，我们不会详谈这些细节。在大多数情况下，随着时间的推移，视频压缩的收益都来自许多小的改进，而在如何压缩视频的方法上没有根本性的转变。因此，我们将描绘整体概念。

MPEG 能同时压缩音频和视频。由于音频和视频编码器独立工作，这就存在一个如何在接收端同步两个流的问题。解决的办法是用一个时钟，将当前时间的时间戳输出到两个编码器。这些时间戳被包含在编码后的输出流中，并且一路传播到接收端，接收端的播放器可以用这些时间戳来同步音频流和视频流。

MPEG 视频压缩利用了电影中存在的两种冗余优势：空间和时间。通过用 JPEG 对每一帧单独进行简单编码可以获得空间冗余。这种方法偶尔会被使用，就像编辑视频制作那样，特别是当需要随机访问每一帧时。在这种模式下，可以达到 JPEG 的压缩水平。

还有一种现象也可以考虑用来压缩图像。连续的帧往往几乎是相同的，充分利用这样的事实还可以获得更大的压缩率。其效果比第一次出现的要差一些，因为许多电影导演在剪辑电影时差不多每隔 3 秒或 4 秒就切换场景（试试计时一个电影片段，并且对场景计数）。然而，运行 75 个甚至更多个高度相似的帧比简单地利用 JPEG 对每一帧进行独立编码提供了更多的冗余，因而具有更大的压缩潜力。

考虑这样的场景：摄像机和背景都固定不变并且只有一两个演员慢慢地走来走去，帧和帧之间几乎所有的像素都是相同的。在这里，只要从前一帧中减去当前帧，并且对两帧之差运行 JPEG 算法，效果就会非常好。然而，对于那些场景：摄像机在移动或者不断地推拉，这项技术的效果会很差。这就需要某种办法来补偿这种运动。这恰好是 MPEG 擅长的工作，也是 MPEG 和 JPEG 的主要区别。

MPEG 的输出包括 3 类帧：

（1）I-帧（Intracoded frames）：帧内编码帧包含了压缩的静止图片。

（2）P-帧（Predictive frames）：预测帧是与前一帧的逐块差值。

（3）B-帧（Bidirectional frames）：双向帧是与前一帧和后一帧的逐块差值。

I-帧只是一些静止图片。它们采用JPEG或者其他类似的算法进行编码。在输出流中周期性地出现I-帧（比如，每秒1或者2次）是有价值的，其中的原因有三。第一，MPEG可能被用于组播传输，观众可以随意收看视频节目。如果所有的帧都依赖于前面的帧，那么一直要回溯到第一帧，否则错过第一帧的人就永远也解不出任何一个后续帧。第二，如果任意一帧发生了接收错误，则后续帧的解码将再也不可能进行：从那时开始的一切都将不知所云。第三，若没有I帧，那么，在快进或者回退时，解码器将不得不计算经过的每一帧，以便知道暂停在某个帧的完整值。因此I-帧必须周期性地出现在媒体流中。

相反，P-帧是对帧间的差值进行编码。P帧建立在**宏块**（macroblock）想法的基础上，宏块覆盖了亮度空间中的16×16像素，以及色度空间中的8×8像素。通过搜索前一帧中与自己相同的部分或稍有不同的部分来进行宏块的编码。

图7-49显示了P帧用法的一个例子。在这里，我们看到3个连续的帧，它们具有相同的背景，但是人的位置有所不同。包含这3个帧背景场景的宏块能够精确地匹配，但是包含人的宏块在位置上有一定的未知量偏移，而且必须被跟踪。

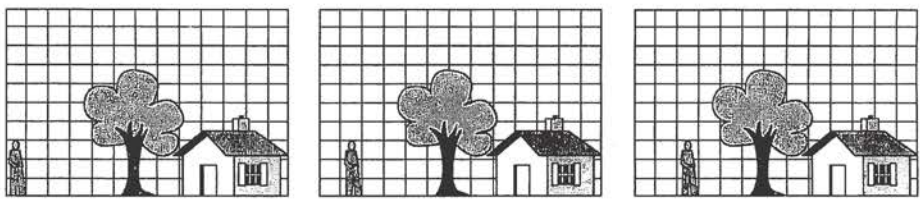


图7-49 3个连续帧

MPEG标准并没有规定如何进行搜索、搜索的范围多大，也没有规定如何评价一个匹配的好坏。这些都由算法的每个实现自行决定。例如，一个实现可能是这样的：在前一帧的当前位置上搜索一个宏块，并且也在x方向上偏移$\pm\Delta x$范围内和y方向偏移$\pm\Delta y$范围内的所有位置上搜索该宏块。对于每个位置，计算亮度矩阵的匹配数量。得分值最高的那个位置只要超过某个预定的阈值，则它被认为胜出者。否则，该宏块称为已丢失。当然，还有可能存在其他更加复杂的算法。

如果找到了一个宏块，则利用它与前一帧中对应宏块之间的差值进行编码（包括亮度和两个色度）。然后，对这些差值矩阵进行离散余弦变换、量化、行程编码和霍夫曼编码，跟通常的做法一样。再者，在输出流中该宏块的值就是运动向量（相对于前一个位置在每个方向上运动了多远），紧随其后的是它差值的编码。如果在前一帧中没有定位到该宏块，则对当前值进行编码，就如同I-帧的做法一样。

很明显，这个算法是高度非对称的。只要自己愿意，任何一个实现都可以任意试探前一帧中每一个合理的位置，不顾一切地找出每一个刚过去的宏块，无论该宏块移动到哪里。这种做法以极慢的编码速度为代价，换来最小化MEPG输出流。对于电影资料库的一次性编码应用来说，这种方法非常好，但是对于实时视频会议来说则非常可怕。

类似地，每个实现可以自由决定什么组成了“找到的”宏块。这种自由度允许各个算法实现在质量和速度上进行竞争，但总是会产生兼容的MPEG输出。

到目前为止，对MPEG的解码非常直截了当。解码I-帧类似于解码JPEG图像。解码P-帧则要求解码器缓冲前面的帧，然后利用完全编码的宏块以及包含与前一帧差值的宏块，

在另一个缓冲区中构造出新的一帧。这个新帧由一个宏块接着一个宏块组合起来。

B-帧类似于 P-帧，不同之处在于，它允许参照的宏块可以是前一帧，也可以是后一帧。这种更大的自由度带来了更强的运动补偿，例如，当有些物体在其他物体的前面、后面或者左右通过时，这种方法非常有用。为了进行 B-帧解码，解码器需要同时在内存中保留一系列帧：过去的帧、当前正在被解码的帧以及未来的帧。同样，尽管 B-帧的解码更复杂，而且增加了某些额外的延迟。这是因为给定一个 B-帧不能马上被解码出来，必须要等到它依赖的后续帧到达。因此，尽管 B-帧提供了最好的压缩算法，由于其更强的复杂性以及对缓冲区的需求，并不是总能适用于一切场合。

MPEG 标准包含许多增强型功能，这些技术实现了优良的压缩级。AVC 可以超过 50∶1 的压缩率来压缩视频，从而把网络带宽的需求减少到相当的程度。有关 AVC 的更多信息，请参阅（Sullivan 和 Wiegand，2005）。

7.4.3 流式存储媒体

现在，让我们把注意力转移到网络应用。我们的第一种情况针对早已存储在文件中的流媒体。最常见的例子是在 Internet 上观看视频。这是**视频点播**（VoD，Video on Demand）的一种形式。其他形式的视频点播使用了服务提供商网络来传送影片，这种网络通常独立于 Internet（例如，有线电视网络）。

在下一节，我们将考查如何现场直播流媒体，例如，广播 IPTV 和 Internet 广播。然后我们将考查第三种情况的实时会议，比如语音 IP 电话或 Skype 视频会议。这三种情况都通过网络提供音频和视频，但对网络的需求越来越严格，我们必须越来越多地关注延迟和抖动。

Internet 上充满了音乐和视频站点，这些站点流化存储在其上的媒体文件。实际上，处理存储媒体的最简单方式不是流化它。想象一下，你想要创建一个在线电影租赁站点，与苹果的 iTunes 竞争。常规的站点允许用户下载视频，然后再观看该视频（当然，是在付费之后）。这个步骤序列如图 7-50 所示，我们把它们拼凑出来，以便与下一个例子作对比。

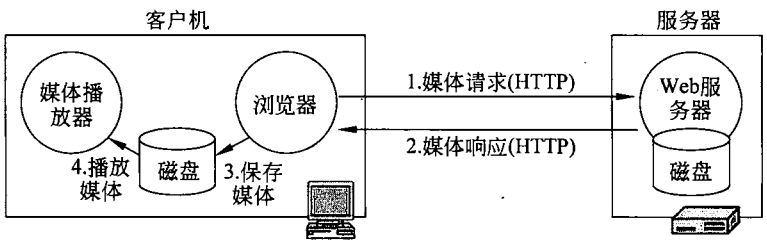


图 7-50 通过简单的下载在 Web 上播放媒体

当用户单击一个电影时浏览器开始行动了。在第 1 步中，浏览器给该电影链接的 Web 服务器发送一个 HTTP 请求，请求该电影。接着第 2 步，服务器从磁盘上获取该电影（只是一个 MP4 或其他某种格式的文件）并将它发回浏览器。使用 MIME 类型，例如 video/mp4，浏览器查找应该如何显示此文件。在这种情况下，它是一个媒体播放器，显示为辅助应用程序，尽管它可能也是一个插件。浏览器将整个电影保存到磁盘上的一个临时文件中。然

后，浏览器启动媒体播放器，并将临时文件的名字传递给它。最后，在第4步中，媒体播放器开始读取文件并播放该电影。

原则上，这种方法完全正确，它将播放出电影。这里没有任何实时网络问题需要处理，因为下载电影只是一次简单的文件下载。唯一的麻烦是在播放电影之前，必须通过网络传输整个电影。大多数客户恐怕都不愿意为他们的“视频点播”等上一个小时。因此这种模式可能会产生问题，甚至对音频也一样。想象一下，你在购买一张唱片之前想试听一首歌曲。如果歌曲是4 MB，这是一首典型的MP3歌曲大小，而且宽带连接为1 Mbps，那么用户将遭遇半分钟的静谧，然后才能开始试听。这样的模式不太可能卖出许多唱片。

为了避免这个问题而且不改变浏览器的工作方式，站点可以使用如图7-51的设计。链接到电影的页面不是实际的电影文件。相反，它是一个称为**元文件**（metafile）的文件。元文件是一种非常简短的文件，仅仅给出了电影的名字（可能还有其他关键描述符）。一个简单的元文件可能只有一行ASCII文本，看上去像这样：

rtsp://joes-movie-server/movie-0025.mp4

浏览器像往常一样得到这个页面，现在是只有一行长的文件，如图中第1步和第2步所示。然后，它启动媒体播放器，并将这一行长的文件传递给播放器，如第3步所示，一切照常进行。媒体播放器读取元文件，发现可以获得电影的一个URL。接着，播放器与joes-movie-server联系，并向服务器请求该电影，见第4步。然后，该电影被流式发回给媒体播放器，见第5步。这种安排的优点在于媒体播放器很快就能启动，仅仅在那个非常短的元文件下载后即可启动。一旦启动后，浏览器就再也不出现在循环中。媒体被直接发给播放器，在整个文件被下载之前就可以播放电影了。

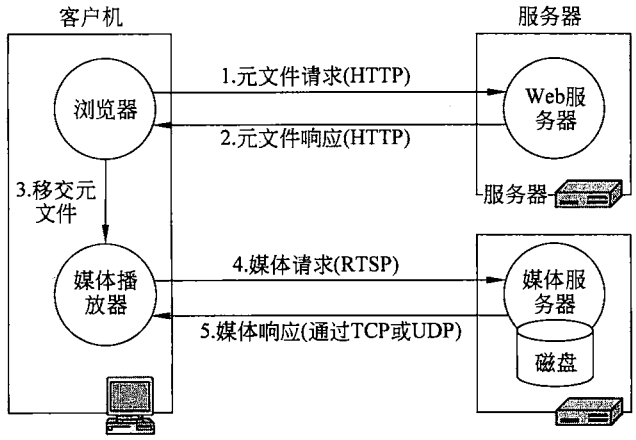


图7-51 使用了Web和媒体服务器的流式媒体

在图7-51中，我们展示了两个服务器，因为元文件中给出的服务器通常与Web服务器不是同一个。事实上，它一般就不是HTTP服务器，而是一个专门的媒体服务器。在这个例子中，媒体服务器使用了**实时流协议**（RTSP，Real Time Streaming Protocol），正如方案名RTSP所指示的那样。

媒体播放器主要完成以下4项工作：

（1）管理用户界面。

（2）处理传输错误。

（3）解压缩内容。

（4）消除抖动。

现在大多数媒体播放器都有华丽的用户界面，有的甚至模拟成一个立体声音响，面板上带有各种按钮、旋钮、调节滑块和可视显示窗等。通常播放器的面板膜样可以自由被替换成其他膜样，这样的面板称为皮肤（skin），用户可以将它拖放到播放器上。媒体播放器必须管理所有这些事项，并且与用户实行交互。

其他的几项工作与网络协议相关，并且取决于网络协议。我们将依次讨论每一个，首先从处理传输错误开始。究竟怎么处理传输错误依赖于用什么协议来传输媒体，是像 HTTP 那样基于 TCP 的传输协议，还是像 RTP 那样基于 UDP 的传输协议。这两种传输协议都可以实际使用。如果使用的是一个基于 TCP 的传输协议，那么媒体播放器不需要纠正错误，因为 TCP 通过使用重传机制已经提供了可靠性。这是一种简单的处理错误方法，至少对媒体播放器是这样，但它在后面的消除抖动步骤中很复杂。

另外，像 RTP 这种基于 UDP 的传输协议，可以用来移动数据。我们在第 6 章讨论过这点。使用这些协议，没有重传机制。因此，由于网络拥塞或传输错误，造成数据包的丢失将意味着一些媒体数据传不到播放器。这个问题要由媒体播放器来处理。

让我们来了解将要面对的困难。因为客户不喜欢他们的歌曲或电影出现较大的缺口，因此丢包是个问题。然而，丢包的损失又不像传输一个普通文件出现丢包时那么严重，因为少量媒体的丢失不需要降低用户的播放。对于视频，用户不太可能注意到偶尔每秒只出现 24 个新帧，而不是 25 个新帧。对于音频，播放期间的短暂间隙可以用时间接近的声音来掩盖。用户不可能发现这种替换，除非他们非常非常关注。

然而，上述推理的关键在于音频或者视频流中的缺口很小。网络拥塞或传输错误通常会导致整个数据包的丢失，并且丢包往往呈现少量的突发。面对这种问题，有两种策略可以用来降低数据包丢失对媒体丢失的影响：FEC 编码和交错编码。我们将依次描述每种策略。

前向纠错（FEC，Forward Error Correction）适用于应用程序级的简单纠错编码，我们已经在第 3 章学习过。整个数据包的奇偶校验提供了一个范例（Shacham 和 McKenny，1990）。每发送 4 个数据包，就据此构造第 5 个奇偶包，组成一组一起发送。如图 7-52 所示中的数据包 A、B、C 和 D。奇偶包 P 包含了冗余位，这些冗余位取自 4 个数据包中每个数据位的奇偶或“异或”和。我们希望对于大多数 5 个包组成的组中所有数据包都能到达接收端。如果果真如此，那么接收器只需简单地丢弃奇偶包。否则，如果只是丢失了校验包，也没有什么损害。

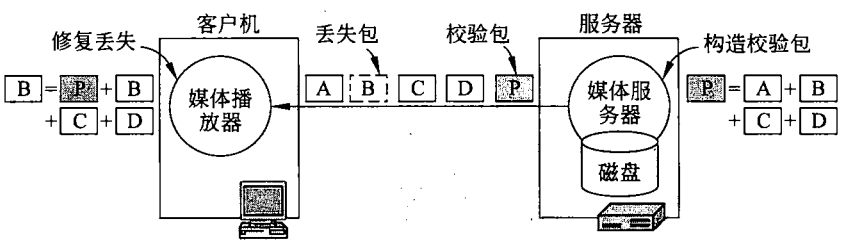


图 7-52 使用奇偶校验包来修复丢失包

然而，有时在传输过程中可能会丢失一个数据包，如图7-52中的B数据包。媒体播放器只接收到了三个数据包A、C和D，加上奇偶包P。通过设计，丢失那个数据包中的数据位可以从校验位重构出来。具体而言，假设“+”代表“异或”或模2加，那么利用异或特征（即X+Y+Y=X），可重构出B，B=P+A+C+D。

FEC可以通过修复一些丢失的数据包，来减缓媒体播放器的损失程度，但它也只能起到一定程度的作用。如果在5个一组的包中丢失了两个包，那我们无论做什么都无法恢复出丢失的数据。需要注意的FEC另一个特征是为获得这种保护措施，我们必须支付的成本。每四个包必须生成第五个包，所以带宽的需求量比媒体所需要的多25%。并且，还增加了接收端解码的延迟，因为我们可能需要等到奇偶包的到达，才可以重新建造出这个奇偶包之前丢失的数据包。

在上面所述的技术中还有一个聪明的技巧。第3章中我们描述了奇偶校验提供的是错误检测。而在这里，我们用奇偶校验进行纠错。怎样才能做到这点呢？答案是在这种情况下已经知道丢失的数据包。丢失的数据称为**擦除**（erasure）。在第3章中，当我们认为接收到的一个数据帧中某些位出错时，我们不知道究竟是哪些位出现了差错。这种情况比擦除情况更难处理。因此，有了擦除信息，奇偶校验就具备了纠错功能，而如果没有擦除信息，奇偶校验就只能提供错误检测功能。当我们讨论组播情景时，很快就能看到奇偶校验带来的另一个意想不到的好处。

第二种策略是所谓的**交错编码**（interleaving）。这种方法的基本原理是这样的：在传输之前把媒体的顺序混合或交错编码起来，然后在接收端拆混或解开交错的媒体。这样，如果一个数据包（或数据包的突发）丢失，因为发送前的混合操作，丢包的损失将随着时间推移分摊到多个数据包。它不会导致媒体播放时出现一个很大的缺口。例如，一个包可能含有220个立体声采样值，每个都包含一对16位数字，通常为5毫秒的好品质音乐。如果采样值按顺序发送，一个信息包的丢失表现在音乐流中会出现5毫秒的缺失。相反，如果按如图7-53所示的方式来发送采样值。所有10毫秒时间间隔内的所有偶数采样值都通过一个数据包发送，而所有奇数采样值通过下一个数据包发送。现在，丢失3个包的损失并不代表丢失5毫秒的音乐，仅仅损失了10毫秒内对应的每个其他采样值。媒体播放器很容易地就能处理这种损失：使用前面和后面的样本值交叉插入丢失位。造成的结果是这10毫秒的音乐具有较低的临时分辨率，但没有明显的媒体缺失。

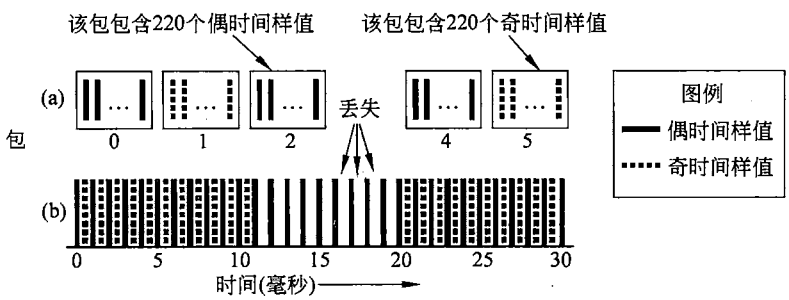


图 7-53

当数据包携带交错的样值时，一个包的丢失只降低了临时分辨率而不会造成播放时间上的沟壑

上面的交错方案只适用于未压缩的采样值。然而，交错发送方式（在短时间内，而不是单个样本值的交替）也可以用于压缩后的样本值，只要有一种方法能找到压缩数据流的采样边界。RFC 3119 给出了一个针对压缩音频的交错方案。

交错编码是一个有吸引力的技术，使用它时无须增加额外的带宽，这点和 FEC 不同；然而，与 FEC 一样，交错编码增加了传输延迟，因为需要等待一组数据包都到达（才能被解交错）。

媒体播放器的第三个工作是解压缩内容。虽然这项任务是计算密集型的，但相当简单。棘手的问题在于如果网络协议不能纠正传输错误，播放器如何解码媒体。在许多压缩方案中，后面的数据不能先于较早的媒体被解压缩，直到较早的数据已经被解压缩出来才能解压缩后面的数据，因为以后的数据是相对于以前数据的编码。对于基于 UDP 的传输协议，还可能出现丢包。因此，编码过程必须设计成不管丢不丢包都能进行解码。这项需求就是为什么 MPEG 采用 I-帧、P-帧和 B-帧的缘故。每个 I-帧可以独立其他帧进行解码，以便从任何先前帧的损失中恢复。

第四个工作是消除抖动，这是所有实时系统的克星。我们在 6.4.3 节描述的一般解决方案是使用播放缓冲区。所有的流媒体系统启动播放前缓冲 5～10 秒的媒体，如图 7-54 所示。播放器定期从缓冲区排出媒体，以便听到清晰的音频和看到平滑的视频。启动延迟给了缓冲区一个机会来填补**至低水位标记**（low-water mark）。我们的想法是数据现在应该有规律地到达足够多，使得缓冲区永远不会完全被排空。如果缓冲区被排空，那么媒体播放器不得不被卡住。缓冲的价值在于如果有时由于网络拥塞导致数据到达得缓慢，那么缓冲的媒体将保持播放器的正常播放，直至到达新媒体并且缓冲区被补充填满。

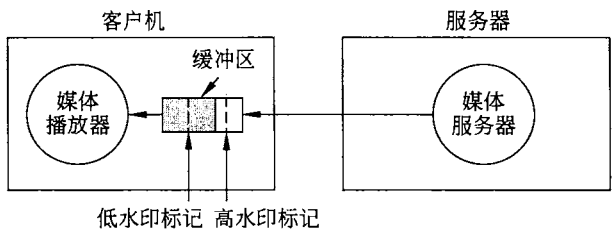


图 7-54 媒体播放器缓冲来自媒体服务器的输入，并且播放缓冲的内容而不是播放直接来自网络的内容

需要多大的缓冲区，媒体服务器要以多快的速率发送媒体才能填满缓冲区，这些决策都取决于网络协议。有很多种可能性。设计中考虑的最大的因素是使用基于 UDP 的传输协议还是使用基于 TCP 的传输协议。

假设采用像 RTP 这种基于 UDP 的传输协议。进一步假设从媒体服务器到媒体播放器有足够的带宽用来发送数据包，只有少量的丢失，而且网络上的其他流量很少。在这种情况下，数据包可按照播放媒体所需要的准确速率发送。每一个数据包通过网络传输，在一定传播延迟后恰好在播放的适当时间到达媒体播放器，然后被播放出来。此时需要的缓冲很少，因为数据包的延迟没有可变性。如果采用交错编码或 FEC 编码，那么需要更多的缓冲，至少需要保证交错或 FEC 得以执行的一组数据包。然而，这仅仅增加了对少量缓冲的需求。

不幸的是，这种场景是不现实的。这里存在两方面的原因。首先，通过网络路径的带

宽变化很大，因此，媒体服务器通常不清楚在它尝试流出媒体之前是否有足够的带宽。一个简单的解决方法是以多种分辨率来编码媒体，并让每个用户选择一个由他 Internet 连接所支持的分辨率。一般来说只有两个层次：高品质的和低质量的，也就是说以 1.5 Mbps 或更高速率的编码能带来高质量的媒体，而以 512 kbps 或更少速率编码的媒体质量较差。

其次，会有一些抖动，或者媒体样值穿越网络所需要的时间会发生变化。这种抖动有两个来源。在网络中通常存在一些相当可观的竞争流量——它们中的某些可能来自多任务用户本身，他们一边浏览网页一边在观看一个流式电影。这种流量会引起媒体到达时间出现波动。而且，我们所关心的是视频帧和音频采样值的到达，而不是数据包的到达。采用压缩技术后，尤其是视频帧可能很大，也可能很小，这完全取决于该帧的内容。一个动作序列通常比一个静止的景观需要更多的比特来编码。因为这些原因，如果网络带宽恒定，媒体的传递随时间的变化率会有所不同。更多的抖动或者延迟的更大变化，要求缓冲区的低水位标记有更大的规模，才能避免欠载造成的播放损失。

现在假设使用基于 TCP 的传输协议来传送媒体，比如 HTTP。因为执行重传机制和等待数据包直到能提供有序的数据包所花费的时间，使得 TCP 增加了媒体播放器可观察到的抖动，而且更加显著。其结果是需要更大的缓冲区和更高的低水位标记。然而，这样做也有一个好处。TCP 将以网络运载数据尽可能快的速度来发送。如果丢失的包必须修复，那么有时候媒体可能会延迟。但大部分时间，网络将能够以快于媒体播放的速度来传递媒体。在此期间，缓冲区将被填满，可起到防止未来欠载的效果。如果网络速度明显快于平均的媒体速率，这是常有的情况，那么启动后不久缓冲区将被迅速填满，这样缓冲区是否会被排空很快将不再是一个令人关切的问题。

基于 TCP 或基于 UDP，并且传输速率超过播放速率，一个问题是媒体播放器和媒体服务器都愿意遥遥领先于播放点开始处理媒体。它们往往愿意下载整个文件。

但是，遥遥领先于播放点开始工作其实是不必要的，这样可能需要大量的存储空间，对避免缓冲区欠载并没有必要。当不想要太早开始处理时，媒体播放器的解决方案是定义一个缓冲区的**高水位标记**（high-water mark）。基本原则是服务器不断地输出数据直到缓冲区被填满至高水位标记。然后媒体播放器告诉服务器暂停发送。由于数据将继续源源不断地流来，直到服务器收到暂停请求，缓冲区中高水位标记和缓冲区末尾之间的距离必须大于网络的带宽延迟乘积。在服务器停止后，缓冲区将开始排出。当它降到低水位标记时，媒体播放器告诉媒体服务器重新启动发送。为了避免欠载，在要求媒体服务器恢复发送媒体时，低水位标记必须考虑网络的带宽延迟乘积。

要启动和停止媒体流，媒体播放器需要对其进行远程控制。这就是 RTSP 协议所提供的功能。该协议在 RFC 2326 中被定义，它给播放器提供了控制服务器的机制。除了启动和停止媒体流外，它可以后退或前进到某个位置、播放特定的时间间隔以及快进或者快退。它不提供数据流的传输，不过，这通常是 UDP 之上的 RTP，或者基于 TCP 的 HTTP 之上的 RTP 的任务。

图 7-55 列出了 RTSP 协议所提供的主要命令。这些命令有一个简单的文本格式，类似于 HTTP 消息，并且通常通过 TCP 来传递命令。RTSP 也可以在 UDP 上运行，因为每个命令都需要确认（并且，如果没有获得确认，就必须重新发送）。

命令	服务器动作
DESCRIBE	列出媒体参数
SETUPE	在播放器和服务器之间建立一个逻辑信道
PLAY	开始给客户发送数据
RECORD	客户开始接受数据
PAUSE	暂停发送数据
TEARDOWN	释放逻辑信道

图 7-55 播放器发给服务器的 RTSP 命令

尽管 TCP 似乎不太适合实时流量的传输，但在实际上它还是经常被采用。最主要的原因是它比 UDP 更容易穿过防火墙，尤其是在通过 HTTP 端口运行时。大多数管理员把防火墙配置成保护他们的网络被不受欢迎的访客访问。他们几乎总是允许来自远程端口 80 的 TCP 连接，以便 HTTP 和 Web 流量通过，因为阻塞该端口将迅速引发校园网的不满。然而，大多数其他端口还是被封锁的，其中包括 RSTP 和 RTP，它们分别使用端口 554 和端口 5004。因此，流媒体通过防火墙的最简单的方式是假装它是一个 HTTP 服务器，正在发送一个普通的 HTTP 响应，至少要让防火墙这么认为。

使用 TCP 还有其他一些优点。因为它提供了可靠性，TCP 给客户提供的媒体是一个完整副本。这很容易让用户倒回到某个播放点观看而不用担心数据丢失。最后，TCP 将尽可能快地缓冲尽可能多的媒体。当缓冲空间变得越来越便宜（当磁盘用于存储时），媒体播放器可以在用户观看的同时下载媒体。一旦下载完成，用户就可以不间断地观看，即使他丢失网络连接也无妨。这个属性对移动电话非常有用，因为对它们而言在运动中与网络的连接可能快速变化着。

TCP 的缺点是增加了启动延迟（因为 TCP 启动需要时间），同时也增加了更高的低水位标记。然而，只要网络带宽超过媒体速率相当多个量级，这点只是很少的一个惩罚。

7.4.4 流式直播媒体

不仅只有录制的视频在网络上红极一时。直播流媒体也非常受欢迎。一旦在 Internet 上观看音频和视频流成为可能，商业电台和电视台就都有了通过 Internet 以及空中广播其节目的想法。没过多久，学院广播站也开始把它们的信号放在 Internet 上。然后，大学生开始了他们自己的 Internet 广播。

今天，人们和各种规模的企业通过网络发送实况音频和视频。这个领域是技术和标准演变的创新温床。直播流媒体用于各大电视台的网上业务，这就是所谓的 **IP 电视**（IPTV，IP TeleVision）。它也可以用作广播电台，例如 BBC，这就是所谓的**网络广播**（Internet radio）。IPTV 和 Internet 广播向全球观众报道各类事件，从时装表演到世界杯足球赛，再到墨尔本板球场测试赛的直播。通过 IP 直播媒体流作为一种技术还被有线电视运营商用来建立自己的广播系统。而且它已被广泛用于低预算的运营，从成人网站到动物园网站。使用目前的技术，几乎任何人都可以快速启动流媒体直播，并且所需费用很少。

流媒体直播的方法之一是把节目记录到磁盘上。观众可以连接到服务器的存档文件，拖拽任何节目，并下载下来慢慢听。**播客**（podcast）就是以这种方式检索的一个片段。对于预先设定的事件，也可以在现场直播之后把内容存储起来，因此存档是正在进行的，或

者说，半小时或更少时间后的直播内容。

事实上，这种做法和我们刚刚讨论过流式媒体所使用的相同。这是容易做到的，我们讨论过的所有技术都可以完成它的工作，而且观众可以从存档文件中的所有节目中挑选自己喜欢的。

另一种不同的方法是在 Internet 上广播实况。观众们调到正在直播的媒体流，就像打开电视机一样。然而，媒体播放器为用户提供的附加功能更多，用户可以暂停或倒转回放媒体。实况媒体将继续流向播放器，被播放器缓冲起来直到用户已经准备好观看。站在浏览器的角度，它看起来就像流存储媒体一样。对于播放器而言，内容是否来自一个文件或者是通过网络正在发送过来的直播根本不重要，而且通常也不告诉他（除了不可能往前快进一个实况流之外）。由于机制的相似性，我们前面讨论的许多内容都适用，但也有一些关键的差别。

重要的是，仍然需要在客户端进行缓冲，以便平滑抖动。事实上，对于流媒体直播通常需要大量的缓冲（独立于用户可能暂停播放作做的考虑）。当媒体流来自一个文件，那么可以一个大于播放速率的速率把媒体推送出来。这种做法将迅速建立一个缓冲区，来补偿网络抖动（如果它不希望缓冲更多的数据，那么播放器将停止流）。相比之下，实况流媒体总是以它产生的精确速率传输，这个速率也是与它被播放的速度相同。它不能以更快的速率发送。因此，缓冲区必须大到足以处理全方位的网络抖动。实际上，10~15 秒的启动延迟通常就足够了，所以这不是一个大问题。

其他重要的区别在于流媒体直播事件通常拥有数百或数千个同时观看相同内容的观众。在这种场景下，流媒体直播很自然的解决方案是使用组播。这与流存储媒体的情况不同，因为那种情况下的用户在任何特定时间通常观看不同的内容。于是，直播给许多用户同时观看的流由许多单个的流会话组成，这些会话发生在同一时间。

组播流媒体的方案以如下方式工作。服务器将每个媒体包通过 IP 组播一次发给一个组地址。网络给每个组成员传递一份包的副本。要想接收媒体流的客户必须加入该组。客户使用 IGMP 就可做到这一点，而不必向媒体服务器发送一个 RTSP 消息。这是因为媒体服务器早就已经在发送实时流了（之前第一个用户加入情况除外）。我们所需要的就是安排在本地接收流。

由于组播是一个一对多的传递服务，媒体被 RTP 包携带，并通过 UDP 传输。TCP 仅用在单个发送者和单个接收者之间。由于 UDP 不能提供可靠性，可能会丢失一些数据包。为了把媒体丢失水平降低到可接受的水平范围内，我们可以使用前面所述的 FEC 和交错编码技术。

在 FEC 的情况下，存在一个组播的良性互动，如图 7-56 所示的奇偶实例。当数据包被组播发送时，不同的客户可能会丢失不同的数据包。例如，客户端 1 丢失了数据包 B，客户端 2 丢失了奇偶包 P，客户端 3 丢失了数据包 D，而客户端 4 没有丢失任何数据包。然而，即使在客户端丢失了 3 个不同的数据包，这个例子中的每个客户端仍然可以恢复所有的数据包。所有这一切的前提是每个客户端不能丢失超过一个包，不论丢哪一个都无所谓，因此丢失的数据包可以通过奇偶计算得以恢复。（Nonnenmacher 等，1997）描述了如何利用这个想法来提高组播传输的可靠性。

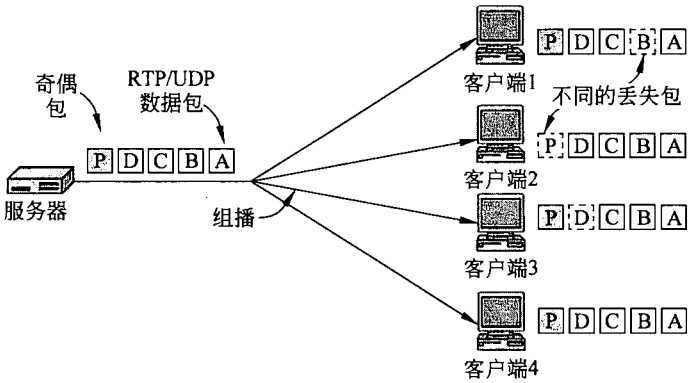


图 7-56 带有奇偶包的组播流媒体

对于拥有大量客户的服务器来说，以 RTP 和 UDP 数据包来组播流媒体无疑是最有效的运营方式。否则，服务器有 N 个客户时必须发送 N 个流，对于同时需要处理大量流事件的服务器来说将需要非常大的网络带宽。

实际上 Internet 不是这样工作的，看到这里你可能会大吃一惊。通常发生的情况是每个用户与服务器建立一个单独的 TCP 连接，媒体流通过该连接传给用户。对客户端来说，这与观看流存储媒体的方式是相同。至于流存储媒体，之所以有这个看似糟糕的选择存在以下几个原因。

第一个原因是 IP 组播在 Internet 上并没有真正获得部署。一些 ISP 和网络仅在自己内部支持它，但它通常不支持跨越其网络边界的组播，因为这需要广域的流传输。其他原因还在于 TCP 相比 UDP 具有某些优势，正如前面所讨论的那样。基于 TCP 传输的媒体流几乎能到达 Internet 上所有的客户，尤其是当伪装成 HTTP 后可穿越防火墙，而且可靠的媒体流传递允许用户执行倒带回放。

然而，在一个重要情况下可以用 UDP 来组播媒体流：在服务提供商的网络内部。例如，有线电视公司可能决定给客户机顶盒广播电视频道，具体做法是使用 IP 技术来代替传统的视频广播。利用 IP 包分发广播视频的技术被广泛地称为 IPTV，如上所述。由于有线电视公司可以完全控制自己的网络，所以它可以把自己的网络工程化为支持 IP 组播，而且为基于 UDP 的分发分配足够的带宽。所有的这一切对于客户都是不可见的，因为 IP 技术只存在于供应商的**围墙花园**（walled garden）内。在服务方面它看起来就像有线电视，但是它的基础是 IP，机顶盒是运行 UDP 的计算机，电视机只是简单连接到计算机的显示器。

返回到 Internet 的情形，在 TCP 上面直播流媒体的缺点是服务器必须为每个客户端发送一个单独的媒体副本。这对于用户中等数量的客户群是可行的，特别是传输音频流。这里的诀窍在于找到一个具有良好 Internet 连接的位置放置媒体服务器，使得它有足够的带宽。通常，这意味着向托管服务提供商租用一台数据中心的服务器，而不是使用家里一台只有宽带 Internet 连接的服务器。托管服务提供商的市场竞争力非常激烈，因此租用一台服务器不会很昂贵。

事实上，任何人，甚至一个学生，都很容易建立和运营一个流媒体服务器，比如一个 Internet 电台。这个电台的主要组成部分如图 7-57 所示。电台的基础是一台普通 PC，带有一个像样的声卡和麦克风。运行流行的软件来捕获音频，并将之编码成各种格式，例如

MP4，媒体播放器像往常一样用来收听音频。

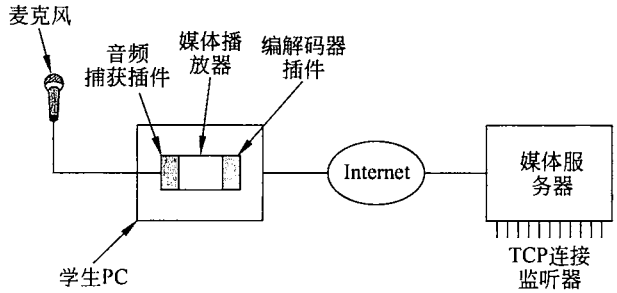


图 7-57 一个学生广播电台

然后，在 PC 上捕获的音频流通过 Internet 被送入一个媒体服务器，作为播客的存储文件流或直播媒体流。该媒体服务器具有良好的网络连接，它处理大量通过 TCP 连接分发媒体的任务。同时该服务器还承担前端 Web 站点的角色，提供电台的介绍页面以及指向可用流媒体内容的链接。所有这些事情都可利用商业软件包来负责管理，也可利用开源软件包来实施，比如 icecast。

然而，针对一个拥有非常大量客户群的电台来说，一台服务器使用 TCP 将流媒体发送到每个客户端的做法就变得不可行了。简单地说，一台服务器根本没有足够的带宽来做这件事情。对于大型流媒体网站，流媒体的分发使用了一组服务器来完成的，而且这组服务器在地理上是分散的，从而使客户端可以连接到最近的服务器。这就是内容分发网络，我们将在本章结尾处学习这部分内容。

7.4.5 实时会议

曾几何时，在公共交换电话网络上可以进行语音通话，而且网络流量主要是语音流量，数据流量只在这里和那里存在一点点。然后，Internet 来了，Web 出现了。数据流量增长、增长、再增长，直到 1999 年数据流量和语音流量一样多（因为语音现在都被数字化了，所以语音流量和数据流量都可以测量比特数）。到 2002 年，数据流量超过语音流量一个量级，并且仍然呈几何级数增长趋势，反观语音流量基本保持持平。

这种增长的后果彻底翻转了电话网络。现在语音流量也用 Internet 技术来运载了，而且只占一小部分的网络带宽。这种颠覆性的技术称为 **IP 语音**（voice over IP），也称为 **Internet 电话**（Internet telephony）。

IP 语音的使用形式各种各样，基本上都是由强大的经济因素所驱动（它能节省金钱，因此人们使用它）。一种形式看起来像普通电话（老式电话？）一样，但是电话插入以太网，并通过网络发出呼叫。Pehr Anderson 是麻省理工学院的本科学生，当他和他的朋友为一个课程项目原型实现了这个设计，结果他们得到的成绩是“B”。愤愤不平的他在 1996 年创办了一个称为 NBX 的公司，并开创了这种基于 IP 的语音传输；3 年后，他将公司以 9000 万美元出售给 3Com。公司喜欢这种方法，因为有了这种技术可以让他们摆脱单独的电话线，并且用他们已经拥有的网络来做这件事。

另一种方法是利用 IP 技术来建立一个长途电话网络。在一些国家，比如美国，这个网

络可以用来接入有竞争力的长途电话服务。用户只需要拨打一个特殊的号码前缀，然后包含其声音采样值的数据包被注入到网络，并通过网络传输；当这些数据包离开网络时其中的声音采样值被提取出来。由于IP设备比电信设备便宜得多，因此这导致了更廉价的服务。

顺便说一句，在价格上的差异并不完全是技术性的。几十年来，电话服务是一个受管制的垄断企业，保证电话公司在其成本之上有一个固定百分比的利润。毫不奇怪，这样导致了成本的上涨，例如，有很多很多的冗余硬件只是以提供更好的可靠性为名义（电话系统只允许每40年故障2个小时，平均3分钟一年）。这种效果通常称为“镀金电线杆综合征”。由于放松了管制，效果已明显下降，当然，传统设备仍然存在。IT行业从未有过任何类似的经营历史，所以它总在不断地精益求精。

不过，我们将专注于IP语音的形式，这可能对用户是最直观的：使用一台计算机呼叫另一台计算机。这种形式伴随着个人计算机开始与麦克风、扬声器、相机捆绑销售、CPU速度足够快到能处理媒体，以及人们开始在家里以宽带速率连接到Internet等一系列的软硬件环境变化逐步普及的。一个众所周知的例子是2003年发布的Skype软件。Skype和其他公司还提供网关，使得用户很容易地呼叫固定电话号码，同时也能呼叫计算机的IP地址。

随着网络带宽的增加，视频电话加入到了语音通话中。最初，视频通话主要在公司内部使用。视频会议系统设计成能交换两个或两个以上位置的视频，使得不同地点的管理人员在举行会议时能够看到对方。然而，有了良好的宽带Internet连接和视频压缩软件，家庭用户也可以参与视频会议。诸如Skype这样的工具，最初只能路由音频，现在能够路由视频，因此分布在世界各地的朋友和家人不仅可以看到对方还能听到对方。

从我们的角度来看，Internet语音或视频通话也属于流媒体问题，但是比存储的流媒体文件或现场直播活动的限制更大。这里新增的约束在于双向通话所必需的低延迟。电话网络允许单向延迟至多为150毫秒，这是用户可接受的最低限度，超过这个延迟就会被与会者视为烦人而拒绝（国际长途电话可能有高达400毫秒的延迟，这一点严重损害了用户体验的积极性）。

这样的低延迟实际上很难达到。当然，缓冲5～10秒的媒体是无法工作（就像它广播一场体育赛事实况）。相反，视频和IP语音系统必须设计并实现多种技术来最小化延迟。这一目标的明确意味着选择UDP而不是TCP，因为TCP的重传机制至少引入了一个往返延迟。然而，一些形式的延迟是不能减少，即使使用UDP也是如此。例如，西雅图和阿姆斯特丹之间的距离接近8000千米，在这么长距离的光纤上光的传播延迟为40毫秒。好啦，祝你好运。实际上，通过网络的传播延迟时间会更长，因为它要覆盖一个较大的距离（比特不会遵循一个大的循环路由），而且还有每个IP路由器存储和转发数据包的传输延迟。这种固定的延迟时间消耗了可接受的延迟预算。

延迟的另一个根源与数据包大小有关。通常情况下，大的包是使用网络带宽的最好方式，因为它们更高效。然而，一个64 kbps的音频采样率，1 KB的数据包需要125毫秒来填补（甚至更长的时间，如果采样值被压缩）。这种延迟会消耗大部分的总延迟预算。此外，1 KB的数据包如果通过1 Mbps的宽带接入链路发送将需要8毫秒的传输时间。然后再加上另一端宽带连接所需要的另一个8毫秒。显然，大的数据包将无法工作。

相反，IP语音系统使用短的数据包，以牺牲带宽效率来换取延迟的降低。它们打包成一批更小单位的音频样本，通常20毫秒一个。按照64 kbps速率，数据长160字节，有少

量压缩。然而，根据定义，这个数据包的延迟将是20毫秒。传输延迟也会小一些，因为包的长度很短。在我们的例子中，它会减少到大约1毫秒。通过使用短的数据包，西雅图到阿姆斯特丹的最低单向延迟已经从一个不可接受的181毫秒（40+125+16）减少到可以接受的62毫秒（40+20+2）。

我们甚至还没有谈论软件开销，但是它也会吃掉一些延迟预算。尤其对于视频的确如此，因为通常需要对视频进行压缩才能使得视频适应当前的可用带宽。这和一个存储的流媒体文件传输完全不同，此时没有时间运行一个计算密集的编码器来获得高层次的压缩比，编码器和解码器都必须快速地运行。

缓冲仍然需要，主要作用还是用来按时播放媒体样值（以避免难以理解的音频和忽停忽动的视频），但是缓冲量必须保持得非常小，因为在我们延迟预算中的剩余时间是以毫秒为单位计量的。当一个数据包经过很长时间到达后，播放器必须跳过丢失的样本，也许用环境噪声或重复帧给用户掩盖损失的那部分媒体。这里用于处理抖动的缓冲区大小和丢失的媒体量之间存在着一个权衡。较小的缓冲区能减少延迟，但会导致由于抖动带来的更多丢失。最终，随着缓冲区空间的缩小，对用户而言丢失将变得更为明显。

细心的读者可能已经注意到在本节到目前为止的讨论中，我们没有提及任何有关网络层协议的内容。通过使用服务质量机制，网络可以减少延迟，或者至少减少抖动。这个问题之所以一直没有提出来的原因在于，以前尽管有很大的延迟流媒体还是能够运作，即使在实况流的情况下。如果延迟不是一个大问题，那么终端主机的缓冲区就足以应付抖动问题。然而，对于实时会议系统，通常减少网络延迟和抖动非常重要，这样才有助于媒体流满足延迟预算。唯一不重要的是当有这么多网络带宽可用时，每个人都能得到良好的服务。

第5章，我们描述了有助于实现这个目标的两个服务质量机制。一种机制是**区分服务**（DS，Differentiated Services），在这种机制下，包被标记为不同的类别，在网络内部将受到不同的处理。适当的做法是将语音IP数据包标记为低延迟。实际上，系统将DS编码点设置成众所周知的**加速转发类**（Expedited Forwarding）和**低延迟型服务**（Low Delay）。这在宽带接入链路特别有用，因为这些链路往往拥挤不堪，当Web流量或其他流量交织在一起竞争使用链接时。对于一条稳定的网络路径，拥塞会造成延迟和抖动的增加。每1KB的数据包在1 Mbps的链路上发送需要8毫秒，而且被排在Web流量的后面，那么语音IP数据包将遭受拥塞带来的延误。然而，打上低延迟标记的语音IP数据包将会跳到队列的前头，绕过Web数据包，由此来降低延迟。

可以用来减少延迟的第二个机制是确保有足够的带宽。如果可用带宽变化不定，或者传输速率波动不稳（犹如压缩视频），并且有时没有足够的带宽，那么在路由器上将建立队列，并且随着队列的增长而延迟增加。这种情况甚至在DS机制下也会发生。为了确保有足够的带宽，在网络内部可以采用预留资源的方法。这种能力就是综合服务所提供的。非常遗憾的是这种机制并没有得到广泛部署。相反，网络被工程化为预期的流量级别，或者网络客户获得一份针对给定流量级别的服务等级约定书。应用程序必须低于这个级别运行，以免出现网络拥塞并造成不必要的延误。对于在家偶尔使用的视频会议，用户可以选择一个视频质量作为带宽需求的代理，或者由该软件测试网络路径，并自动选择适当的质量。

上述任何一个因素都可导致数据包的延迟成为不可接受，因此，实时会议要求应该高度重视所有这些因素。有关IP语音概述和影响延迟的因素分析报告，请参阅（Goode，2002）。

既然我们已经讨论了媒体流路径上的延迟问题，我们现在将注意力移动到会议系统必须解决的其他主要问题。这个问题就是如何建立和拆除呼叫。我们将着眼于两个被广泛应用的协议，H.323 和 SIP。Skype 是另一个重要的系统，但其内部工作原理是私有的。

H.323

在开始通过 Internet 进行音频和视频呼叫之前，有一件事情是每个人都很清楚的，那就是如果每个厂商都设计它自己的协议栈，那么系统将无法工作。为了避免这个问题，许多感兴趣的团体在 ITU 的倡导下，聚集在一起制定相关标准。1996 年，ITU 发布了推荐标准 **H.323**，其标题是“无服务质量保障的局域网可视电话系统与设备”（Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service）。只有电话行业才会想得出这样一个名字。很快在 1998 年它就被修订成“基于数据包的多媒体通信系统”。H.323 是第一代应用广泛的 Internet 电话系统的基础，它仍然是获得最广泛部署的解决方案，2009 年发布了第 7 个版本。

与其说 H.323 是一个特定的协议，不如说它是 Internet 电话总体结构的概述。在语音编码、呼叫建立、信令、数据传输和其他领域中，H.323 引用了大量特殊的协议，而不是自己制定一套新规定。图 7-58 描述了它的总体模型。在中心处是一个**网关**（gateway），它将 Internet 与电话网络连接起来。网关在 Internet 一端使用 H.323 协议，在电话端则使用 PSTN 协议。通信设备称为**终端**（terminal）。一个 LAN 可能有一个**网守**（gatekeeper），它控制其管辖范围内的端点，这个管辖范围称为**区域**（zone）。

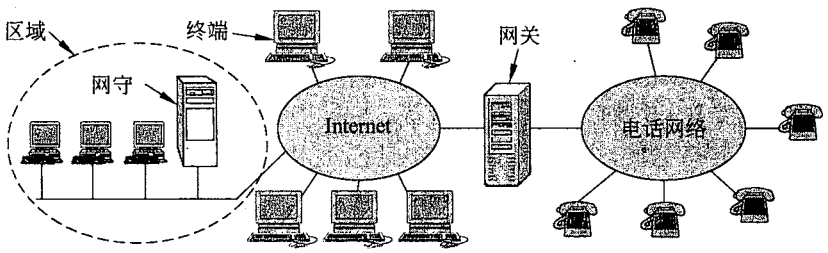


图 7-58 Internet 电话的 H.323 体系结构模型

电话网络需要许多协议。首先，需要一个协议用来对音频和视频进行编码和解码。标准电话把一个语音信道表示成 64 kbps 的数字语音（每秒 8000 次采样），这是由 ITU 推荐的标准 **G.711** 所定义的。所有的 H.323 系统必须支持 G.711。其他压缩语音的编码方法也是允许的，但不是必需的。这些方法采用了不同的压缩算法，并且在质量和带宽之间作出不同的权衡。对于视频，必须支持我们前面描述过的视频压缩 MPEG 形式，包括 H.264。

由于允许使用多个压缩算法，所以为了允许终端协商即将使用哪一个算法，还需要一个协议。这个协议称为 **H.245**。它还能协商一个连接的其他方面，比如数据速率。RTCP 被用来控制 RTP 信道。还需要有一个协议用来建立和释放连接、提供拨号音、产生响铃声以及完成标准电话的其他功能。这里使用的是 **ITU Q.931**。终端需要一个协议跟网守（如果有的话）进行通信，**H.225** 可作为此用途。它管理的从 PC 到网关的信道称为“**注册/许可/状态**”（RAS，Registration /Admission /Status）信道。该信道允许终端加入和离开该区域、请求和返回带宽、提供状态更新，以及其他一些事情。最后，还需要有一个协议用于实际

的数据传输，UDP之上的RTP可用作此用途。如同往常一样它由RTCP来管理。所有这些协议的相互关系如图7-59所示。

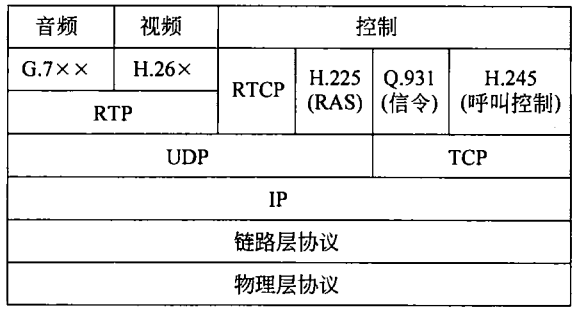


图7-59 H.323协议栈

为了看清楚这些协议是如何相互配合工作的，考虑在一个LAN（有一个网守）中有一台PC终端需要呼叫一部远程电话的情形。该PC首先必须找到网守，因此它向端口1718广播一个UDP网守发现包；当网守有了回应以后，PC就知道了网守的IP地址。现在，PC向网守发送一条RAS注册消息，该消息被封装在一个UDP数据包中。在该消息被接受以后，PC向网守发送一条RAS许可消息，请求所需要的带宽。只有在获得了所需的带宽以后，PC才可以开始请求建立呼叫。事先请求带宽的做法是为了让网守限制呼叫的数量，避免超额使用出境线路，从而有助于提供必要的服务质量。

顺便说一句，电话系统做了同样的事情。当你拿起听筒，一个信号就被发送到本地端局。如果本地端局有足够的空闲容量用作一个呼叫，它会产生一个拨号音。如果没有，你将什么也听不到。如今，电话系统的规模已经超大到你几乎瞬间就能听到拨号音，但在初期的电话中，往往要花几秒钟才能听到拨号音。因此，如果你的孙子辈问你“为什么有拨号音？”现在你知道答案了。除非到那时，电话已不再存在。

为了建立呼叫，现在PC建立了一个到网守的TCP连接。建立呼叫使用了现有的电话网络协议，这些协议都是面向连接的，因此需要使用TCP。相反地，电话系统不需要电话机来宣布自己的存在，因此没有像RAS这样协议，所以H.323的设计者可以自由使用UDP或TCP来实现RAS，最终他们选择了开销较小的UDP。

既然该PC已经被分配了所需要的带宽，它就可以通过TCP连接发送一条Q.931 SETUP消息。该消息指定了被叫电话的号码（如果被呼叫的是一台计算机，则给出的是IP地址和端口）。网守以一条Q.931 CALL PROCEEDING消息作为响应，确认它已正确地收到了请求。然后网守将SETUP消息转发给网关。

网关的一半是计算机，另一半是电话交换机，它向目标（普通）电话发出一个普通电话呼叫。目标电话所在的端局使被叫电话发出响铃声，同时该端局还发回一条Q.931 ALERT消息，告诉主叫方PC电话已经开始响铃了。当另一端的用户拿起电话时，该端局发送回一条Q.931 CONNECT消息，以便通知PC呼叫连接建立好了。

一旦连接被建立，网守就不再出现在通信过程中，当然网关仍然还在。后续的数据包将绕过网守，直接抵达网关的IP地址。至此，我们才有了一个运行于两端之间的直接管道。这只是一个用来传输比特的物理层连接，仅此而已没有其他。通信双方中的任何一方对另

一方都一无所知。

现在使用 H.245 协议来协商当前呼叫的参数，这里使用的是 H.245 控制信道，它始终是打开的。每一方都从宣布它的处理能力开始，例如，它是否能处理视频（H.323 可以处理视频）或会议呼叫、它支持哪些编解码器，等等。一旦一方知道了另一方具备哪些处理能力以后，就建立两个单向的数据信道并且为每个信道指定一个编解码器和其他一些参数。由于每一方可能有不同的设备，所以前向和逆向信道上的编解码器有可能完全不同。当所有的协商工作都完成以后，就可以开始用 RTP 来传送数据流了。RTP 的管理由 RTCP 负责，RTCP 在拥塞控制中起着重要的作用。如果有视频的话，RTCP 还要处理音频/视频的同步。图 7-60 显示了各种信道。当任何一方挂断电话时，为了释放不再需要的资源，在完成通话后使用 Q.931 呼叫信令信道来拆除连接。

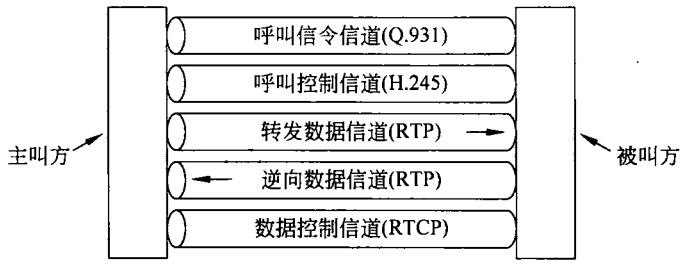


图 7-60 一次呼叫过程中主叫和被叫之间的逻辑信道

当呼叫结束时，主叫方 PC 再次用一条 RAS 消息与网守联系，以便释放分配给它的带宽。或者，它也可以启动另一次呼叫。

我们没有涉及 H.323 中有关服务质量的任何内容，即使我们说过这是实时会议取得成功的一个非常重要部分。之所以没有讨论这部分内容是因为 QoS 超出了 H.323 的范围。如果底层网络能够提供一个从主叫 PC 到网关之间稳定而无抖动的连接，那么该呼叫的服务质量就会很好；否则，服务质量无法得到保证。然而，电话那端的呼叫部分将是无抖动的，因为电话网络就是这样设计的。

SIP——会话发起协议

H.323 是由 ITU 设计的，Internet 社团中的许多人把它看成是一个典型的电信产品：庞大、复杂而且不灵活。因此，IETF 建立了一个委员会，专门设计一种更加简单和模块化的方法来实现 IP 语音。迄今为止的主要成果是**会话发起协议**（SIP，Session Initiation Protocol），最新版本是 RFC 3261，发布于 2002 年。该协议描述了如何建立 Internet 电话呼叫、视频会议和其他多媒体连接。H.323 是一个完整的协议集，与此不同的是 SIP 只是单个模块，但是它被设计成能很好地与现有的 Internet 应用协同工作。例如，它将电话号码定义成 URL，所以 Web 页面可以包含电话号码，用户只要单击一个链接就可以发起一次电话呼叫（与 mailto 方法相同，mailto 允许用户单击一个链接来启动发送电子邮件消息的程序）。

SIP 可以建立两方会话（普通的电话呼叫）、多方会话（每个人都可以听和说）以及组播会话（一个发送方，多个接收方）。这些会话可以包含音频、视频或数据，后者对于诸如多方实时游戏之类的应用非常有用。SIP 只处理会话的建立、管理和终止。还需要使用诸如 RTP/RTCP 之类的其他协议来实施数据传输。SIP 是一个应用层协议，它可以运行在 UDP

或TCP之上。

SIP支持多种服务，包括查找被叫方的位置（他可能不在家里的计算机旁边）、确定被叫方的处理能力，以及处理呼叫建立和终止的机制。在最简单的情形中，SIP建立一个从主叫方计算机到被叫方计算机之间的会话，所以我们首先来讨论这种情形。

SIP中的电话号码被表示成采用sip方案的URL，例如，sip:ilse@cs.university.edu代表一个DNS域名为cs.university.edu主机上一个名叫ilse的用户。SIP URL也可以包含IPv4地址、IPv6地址或者实际的电话号码。

SIP协议是一个仿照HTTP的基于文本的协议。一方以ASCII文本的形式发送一条消息，消息的第一行包含一个方法名，接下来的几行包含一些传递参数的头。很多头都来自于MIME，以便SIP能与现有的Internet应用协同工作。图7-61列出了核心规范中定义的6种方法。

方法	描述
INVITE	请求发起一次会话
ACK	确认会话已经启动
BYE	请求终止会话
OPTIONS	查询主机的能力
CANCEL	取消正在进行中的请求
REGISTER	通知服务器重定向用户的当前位置

图7-61 SIP方法

为了建立一个会话，主叫方要么与被叫方建立一个TCP连接并在其上发送一条INVITE消息，要么以UDP数据包形式发送INVITE消息。在这两种情况下，第二行和随后各行中的头描述了消息体的结构，其中包含主叫方的处理能力、媒体类型和格式。如果被叫方接受此次呼叫，那么它回应一个HTTP类型的应答码（使用图7-48中响应组的三位数字，200表示接受）。在应答码行的后面，被叫方还可以提供有关自己的处理能力、媒体类型和格式等信息。

利用三次握手方法，可以建立一个连接，所以主叫方以一条ACK消息作为响应，以便结束此协议，并确认已收到了200消息。

双方中的任何一方都可以通过发送一条包含BYE方法的消息来结束当前会话。当另一方确认了该消息，会话就结束了。

OPTIONS方法被用于向一台机器查询它的处理能力。通常主叫方在发起一个会话前使用OPTIONS方法来查明该机器是否具备处理IP语音的能力，或者预期的会话类型。

REGISTER方法与SIP的跟踪能力有关，SIP能够追踪一个用户，并且当该用户不在家时与他建立连接。该消息被发送给一个SIP位置服务器，正是该位置服务器记录了每个用户当前在哪里。以后，可以向它查询某个用户的当前位置。重定向的操作如图7-62所示。在这里，主叫方将INVITE消息发送给一个代理服务器，以便把可能的重定向过程隐藏起来；然后代理服务器查找被叫用户的位置信息，并将INVITE消息发送至该处；然后，代理服务器为后续的三次握手过程中的消息充当中继节点。LOOKUP和REPLY消息并不是SIP的一部分；这里可以使用任何方便的协议，取决于使用了什么类型的位置服务器。

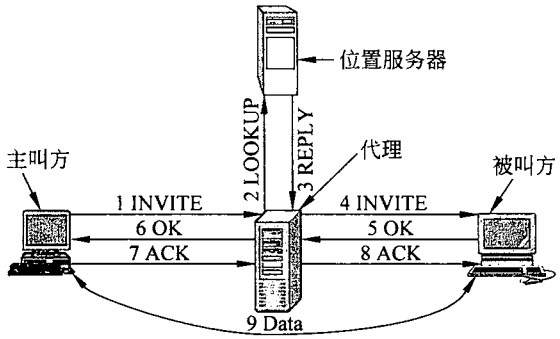


图 7-62 使用代理服务器和 SIP 重定向

SIP 还有许多其他功能，其中包括呼叫等待、呼叫屏蔽、加密和认证等，我们这里不再一一赘述。如果在 Internet 和电话系统之间有可用的合适网关，则 SIP 还具备从计算机向普通电话发起呼叫的能力。

H.323 与 SIP 比较

H.323 与 SIP 都允许用计算机和电话作为端点来实现两方和多方呼叫。它们都支持参数协商、加密和 RTP/RTCP 协议。图 7-63 总结了它们之间的相似之处和不同之处。

项目	H.323	SIP
由谁设计	ITU	IETF
是否与PSTN兼容	是	大部分是
是否与Internet兼容	是，随时间推移	是
体系结构	庞大	模块化
是否完整	完整的协议栈	只有用来控制建立的SIP
是否允许参数协商	是	是
呼叫信令	TCP之上的Q.931	TCP/UDP之上的SIP
消息格式	二进制	ASCII
媒体传输	RTP/RTCP	RTP/RTCP
是否支持多方呼叫	是	是
多媒体会议	是	否
寻址方式	URL或电话号码	URL
呼叫终止	显式或TCP释放	显式或超时
即时消息	否	是
加密	是	是
标准大小	1400页	250页
实现	庞大而复杂	中等，但有问题
状态	世界范围，尤其视频	可替代，尤其音频

图 7-63 H.323 和 SIP 的对比

尽管两个协议的功能集类似，但它们的哲学思想却相差甚远。H.323 是一个典型的重量级电话工业标准，它规定了完整的协议栈，并且精确定义了什么是允许的，什么是禁止的。这种方法使得每一层都有定义完好的协议，从而使得互操作性的任务变得容易。但是，其付出的代价是一个庞大的、复杂的、严格的标准，而且这个标准难以适应未来的应用。

与此相反，SIP 是一个典型的 Internet 协议，只需要交换短短几行 ASCII 文本就可以工作。它是一个轻量级的模块，与其他的 Internet 协议协同工作得很好，但是与现有电话系

统的信令协议一起工作则稍差一些。

因为IETF的IP语音模型是高度模块化的，所以它很灵活，并且很容易适应新的应用。其缺点是遭遇的如何实现互操作性，尤其当人们试图解释什么是标准化方法时比H.323稍逊一筹。

7.5 内容分发

Internet通常被用于通信目的，就像电话网络一样。在早期，研究者要与远程计算机通信，必须通过网络登录到远程机器才能执行任务。很长一段时间人们使用电子邮件来互相沟通，现在，大家都使用视频和IP语音。然而，自从Web出现之后，在Internet上内容已经多于通信。很多人使用网络查找资料，受到电影、音乐和节目访问的驱动，出现了大量可共享的对等文件。内容交换已如此明显，以至于现在大多数Internet带宽被用来传递存储的视频。

由于分发内容的任务与通信完全不同，它对网络提出了不同的需求。例如，如果Sally想与Jitu谈话，她可能向他的移动电话发出IP语音呼叫。通信必须在特定的计算机上完成；它呼叫Paul的计算机将不能做得很好。但是，如果Jitu想观看他团队的最新板球比赛，他很幸福地可以从任何提供服务的计算机上获得视频流。他不介意计算机是Sally的或者是Paul的，或者更加可能的是一台Internet上的未知服务器。也就是说，相对于内容来说位置无关紧要，除了它会影响性能（和合法性）。

另一个不同是一些提供内容的Web站点已红极一时。YouTube是一个典型的例子。它允许用户共享他们自己创作的影片，无论任何一个可以想象的主题都能找到相应的视频。很多人都想这样做，其余的我们就是观众。所有这些都是高带宽需求的视频，据估计，直至今天的Internet流量中的10%被YouTube账户占用。

没有单个服务器足以强大到或者可靠到足以应付如此令人吃惊程度的需求。相反，YouTube和其他大型内容提供商建立了它们自己的内容分发网络。这些网络使用了遍布在世界各地的数据中心，为数量极其庞大的客户群提供具有良好性能和可用性的内容。

随着时间的推移，内容分发所用技术也已经被开发了出来。早在Web发展初期，其流行度几乎将其毁灭。对内容的更多需求导致服务器和网络频繁地超载。很多人开始称WWW为全球等待（World Wide Wait）。

为了响应消费者的需求，在Internet核心供应了超大量的带宽，并且在网络边缘推出更快的宽带连接。带宽是提高性能的关键，但它仅仅是解决方案的一部分。为了减少无休止的延迟，研究人员还针对内容分发开发了使用带宽的不同体系结构。

一种体系结构就是**内容分发网络**（CDN，Content Distribution Network）。采用这种体系结构，内容提供商在Internet的某些位置建立一组分布式机器，然后通过它们给客户提供内容。这是大牌球员的选择。另一种体系结构是**对等网络**（P2P，Peer-to-Peer）。采用这种体系结构，一组计算机把它们的资源集中起来，彼此互相服务提供内容，无须单独配置服务器或者任何中央控制节点。这种想法抓住了人们的想象力，因为通过共同努力一致行动，许多小球员可以团结起来形成一股巨大的冲击力。

在本节，我们将着眼于在 Internet 上分发内容面临的问题，以及实际采用的一些解决方案。在简要讨论内容的普及和 Internet 流量后，我们将介绍如何建立强大的 Web 服务器，并使用缓存来提高 Web 客户端的性能。然后我们再考查分发内容的两个主要体系结构：CDN 和 P2P 网络。正如我们将看到的那样，这两大结构在设计和性能上有很大的不同。

7.5.1 内容和 Internet 流量

为了设计和工程化工作良好的网络，我们需要理解网络运载的流量。例如，随着 Web 重心转移到了内容上，服务器已经从公司办公室迁移到了 Internet 数据中心，中心提供了大量拥有超级优秀网络连接的机器。如今要运行一个小型服务器，从 Internet 数据中心租用一台托管的虚拟服务器，甚至也比在家里或办公室运行一台与 Internet 有宽带连接真实的机器更容易和更方便。

幸运的是，关于 Internet 流量有必要了解的事实只有两个。第一个事实是，流量的变化很快，不仅表现在细节上，而且在整体上也如此。在 1994 年之前，大部分流量是传统的 FTP 文件传输（在计算机之间移动程序和数据集）和电子邮件。然后 Web 出现了，并成倍地增长。在 2000 年网络泡沫之前 Web 流量将 FTP 和电子邮件的流量远远地抛在了后面。从 2000 年左右开始，共享音乐，然后是共享电影的 P2P 文件异军突起。到 2003 年，大多数 Internet 流量变成了 P2P 流量，它把 Web 流量远远地抛在了后面。在 2000 年后期的某个时候，使用内容分发方法的网站提供视频流，比如 YouTube 等，视频流开始超过 P2P 流量。到 2014 年，思科预计所有 Internet 流量的 90%将是这种或那种形式的视频（Cisco，2010）。

流量并不总是那么重要。例如，虽然语音 IP 流量蓬勃发展，即使在 2003 年 Skype 开始提供服务后，但它的流量永远是图表上的一个小亮点，因为音频的带宽需求低于视频所需的两个数量级。然而，IP 语音流量强调了网络其他方面的重要性，因为它对延迟敏感。作为另一个例子，自从 2004 年 Facebook 横空出世以来，在线社交网络疯狂地生长着。2010 年，Facebook 的每天 Web 用户首次超过了谷歌用户。即使将流量放在一边不予考虑（的确有超大量的流量），在线社交网络非常重要，因为它们正在改变着人们通过 Internet 实行互动的方式。

我们必须指出的一点是 Internet 流量迅速发生着震撼般的转变，并呈现出一定的规律性。接下来会发生什么？请看本书的第 6 版，我们将告诉你答案。

有关 Internet 流量的第二个基本事实是它的高度倾斜特性。我们所熟悉的许多特性都是聚集起来取个平均值。例如，大多数成年人的平均身高接近。有一些很高的人和一些很矮的人，但很少有非常非常高或非常非常矮的人。对于这类特性，有可能设计出一个范围，它不是很大但仍然捕获了大多数人口。

Internet 流量与此不同。长期以来，人们一直认为少量 Web 站点具有大量的流量，大量的 Web 站点具有很少的流量。这个特征已经成为网络语言的一部分。早期的研究论文谈到流量时用了**数据包火车**（packet train）这个字眼，想法是载有大量数据包的特快列车会突然穿过链路（Jain 和 Routhier，1986）。这种情况被正式定义为**自相似**（self-similarity）概念，按照我们的目的可以把它看作具有许多大大小小沟壑的网络流量，即使从不同的时间尺度来看都是不连续的一块一块（Leland 等，1994）。后来的研究工作将大流量说成是**大**

象，把短流量说成是小鼠。只有少数的大象和大量的小鼠，但是大象因为庞大而举足轻重。

回到 Web 内容，相同的倾斜显而易见的。录影带出租店、公共图书馆和其他类似组织的经验表明，并非所有项目都同样受欢迎。实验中，当有 N 个电影可用，所有请求第 k 个最流行电影的比例大约为 C/K。在这里，C 用来计算规一化的和，即

$$C = 1 / (1 + 1/2 + 1/3 + 1/4 + 1/5 + \cdots + 1/N)$$

因此，最流行电影的流行度是排名第七流行度电影的 7 倍。这个结果称为**齐普夫定律**（Zipf's law）（Zipf，1949），它以 George Zipf 的名字命名，这是一位哈佛大学的语言学教授，他指出一个字在大段文字中的使用频率和它的排名成反比的。例如，第 40 个最常见字的使用次数是第 80 个最常见字的使用次数的两倍，是第 120 个最常见字的使用次数的 3 倍。

图 7-64(a)显示了齐普夫分布。它捕获的概念是有少数受欢迎的项目和很多不那么受欢迎的项目。要认识这种形式的分布，把数据绘制在两轴的对数尺度就容易看出，如图 7-64(b)所示。结果应该是一条直线。

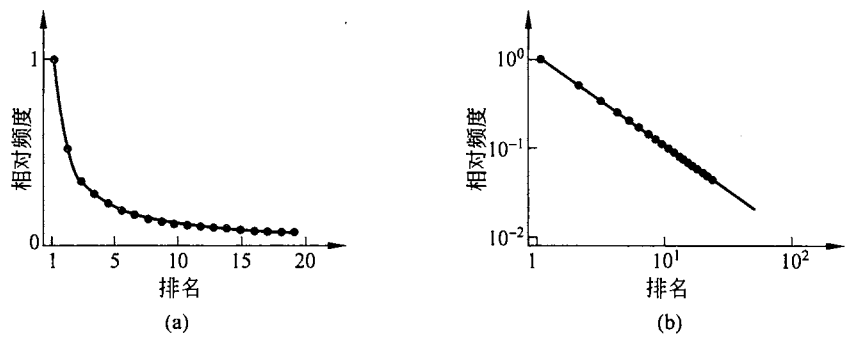


图 7-64 齐普夫分布
(a) 线性尺度；(b) 对数尺度

当人们查看流行的网页时，结果表明它大致遵循齐普夫定律（Breslau 等，1999）。齐普夫分布是一个称为**幂定律**（power law）家族分布的例子。幂定律在许多人类现象上是显而易见的，比如城市人口和财富的分布。在描述几个大牌球员和大量不知名小球员方面它们具有相同的特性，而且它们在对数图上也表现为一条直线。人们很快发现 Internet 的拓扑结构也可大致描述为幂定律（Faloutsos 等，1999）。紧接着，研究人员开始将每一个可以想象的 Internet 特性绘制成对数尺度，在观察到一条直线后大声惊呼："幂定律！"

然而，比一张对数图上的直线更要紧的是这些分布对网络设计和使用的意义何在。鉴于许多形式的内容具有齐普夫或幂定律的分布特征，似乎 Internet 上的网站普遍与齐普夫类似。反过来这意味着平均站点并不具备有用的代表性。站点最好描述成受欢迎或不受欢迎。这两种站点才是至关紧要的。流行的网站明显是个问题，因为一些热门网站可能要对 Internet 上的大多数流量负责。也许令人惊讶的是不受欢迎的网站也会是个问题。这是因为所有指向不受欢迎网站的流量总额占用了总流量的很大一部分。原因就在于有那么多不受欢迎的网站。许多不受欢迎的选择组合起来就是个问题，这种概念已经通过书籍得到普及，比如《长尾理论（The Long Tail）》（Anderson，2008a）。

类似图 7-64(a)显示衰减的曲线图很常见，但它们各不相同。特别地，当衰减率正比于

剩余多少材料（比如不稳定的放射性原子）时，这样的情形揭示了**指数衰减**（exponential decay），其下降速度远远超过齐普夫定律。在时间 t 后的项目数（比如说原子）通常表示为 $e^{-t/\alpha}$，其中常数 α 确定了如何快速地衰变。指数衰减和齐普夫定律之间的差异在于：对于指数衰减，忽略尾部是安全的；而对于齐普夫定律，尾部的总权值很显著，不可被忽视。

为了在这个倾斜世界里有效地工作，我们必须能够构建两种类型的 Web 站。不受欢迎的网站很容易处理。使用 DNS，把许多不同的站点实际上指向 Internet 上的同一台计算机，在这台计算机上运行全部不受欢迎的网站。另一方面，流行的网站很难应付。没有单个的计算机足够强大到能运行所有流行的网站，甚至远程也没有；而且如果使用一台计算机，那么当它发生故障将使数百万用户无法访问网站。为了处理这些网站，我们必须建立内容分发系统。下面我们开始探讨这个问题。

7.5.2 服务器农场和 Web 代理

到目前为止我们看到的 Web 设计都只有一个单独的服务器，这台机器和多个客户端机器交谈。为了构建大型 Web 站点并且运行性能良好，我们可以加快服务器端的处理或者客户端的处理。在服务器端，用更强大的 Web 服务器建设成一个服务器农场，这里用一个计算机集群替代单个的服务器为客户提供服务。在客户端，可以通过更好的缓存技术来实现更好的性能。尤其是，代理缓存为一组客户提供了大容量的共享缓存。

我们将依次描述这些技术。然而，请注意，没有一种技术足以用来建立最大的 Web 站点。这些流行的站点需要用到我们在下面章节描述的内容分发，这些方法还必须与坐落在许多不同地点的计算机结合起来一起发挥作用。

服务器农场

无论一台机器的带宽多大，在超负荷之前它也只能为一定数量的 Web 请求服务。这种情况的解决方案是使用多台计算机来充当 Web 服务器。这个想法导致了如图 7-65 所示的服务器农场模型的诞生。

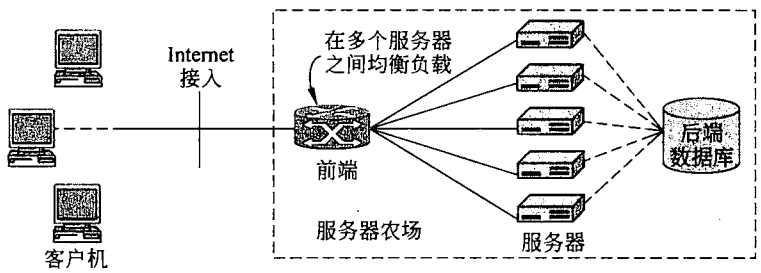


图 7-65 服务器农场

这个模型看似简单，实质上它的困难之处在于组成服务器农场的一组计算机，在客户看起来必须像单个逻辑网站。如果做不到这样，那我们只是建设了不同的网站，让这些网站并行运行而已。

如何使一组服务器以一个网站的形式出现有几种可能的解决方案。所有的解决方案都假设任何一台服务器可以处理来自任何客户端的请求。要做到这一点，每个服务器必须有

一份 Web 站点的副本。为此目的，这些服务器都连接到一个共同的后端数据库，图中用虚线表示。

一种解决方法是使用 DNS 将请求分散到服务器农场中的各个服务器。当来自一个针对 Web URL 的 DNS 请求时，DNS 服务器会返回一个服务器 IP 地址的循环列表。每个客户端尝试一个 IP 地址，通常取列表上的第一个。这个效果正如我们预期的那样，与不同客户通过不同服务器来访问相同网站的效果完全不同。 DNS 方法是 CDN 的核心，我们将在本节稍后再来讨论它。

其他的解决方案是基于**前端**（front end）的方式，由前端把入境请求分散到服务器农场的服务器池中。当客户端使用单一目标 IP 地址联系服务器农场时，就会发生这种情况。前端通常是一个链路层交换机或 IP 路由器，也就是说，是一台处理帧或包的设备。所有的解决方案都基于该台设备（或服务器），该设备窥视网络层、传输层或应用层的头并以非标准的方式使用偷看来的信息。Web 请求和响应都通过 TCP 连接运载。为了正常工作，前端设备必须把一个请求的所有数据包分发到同一台服务器。

一个简单的设计是前端将所有的入境请求广播给所有的服务器。每个服务器按照事先的约定回答一小部分请求。例如，16 台服务器可能查看请求的源 IP 地址，只有源 IP 地址最后 4 位与它们配置的选择器相匹配时才回答该请求，其他的包都将被丢弃。虽然这样浪费了入境带宽，但往往响应包远远长于请求包，所以它并不像听起来的那样低效。

在更通用的设计中，前端可以检查数据包的 IP、TCP 和 HTTP 头，并任意地将它们映射到服务器。这种映射称为**负载均衡**（load balancing）策略，因为目标是为了均衡服务器的工作负载。这种策略可能很简单，也可能很复杂。一种简单策略是依次使用服务器，一个接着一个，或者以一种循环的形式分配服务器的使用。采用这种方法时，前端必须记住每个请求与服务器对应的映射关系，才能确保作为同一请求组成部分的随后包被发送到同一台服务器。此外，为了使网站比单个服务器更可靠，前端应注意到当某个服务器失败后，必须立即停止向它们发送请求。

与 NAT 类似，这种通用设计有点危险，或者说至少是脆弱的，因为我们刚刚创建的设备违反了分层协议的最基本原则：每一层都必须使用自己的头来实行控制目的，无论什么目的都不能检查和使用有效载荷的信息。但无论如何，人们设计了这样的系统，在未来如果因为更高层次协议的变化导致这些设备无法使用时他们或许会惊讶。在这种情况下的前端设备是一个交换机或路由器，但它根据传输层或更高层次的信息采取相应的行动。这样的盒子称为**中间件盒**（middlebox），因为它把自己插入到一条网络路径的中间，而根据协议栈的规定它是没有这项业务的。在这种情况下，前端设备最好考虑作为服务器农场内部的一部分，将所有层次都终结到应用层（因此可以使用所有这些层的头信息）。

然而，与 NAT 一样，实际上这种设计非常有用。查看 TCP 头的原因是它可以比只用 IP 信息来处理负载均衡工作得更好。例如，一个 IP 地址可能代表了整个公司，因而可能会产生许多请求。只有通过查看 TCP 或更高层次的信息，才能将这些请求映射到不同的服务器。

查看 HTTP 头的原因稍微有些不同。许多 Web 交互访问和更新数据库，比如当消费者要查看她的最近购买记录时。处理该请求的服务器将查询后台数据库。将来自同一个用户的后续请求直接指向同一台服务器是有用的，因为该服务器已经缓存了有关该用户的信息。

引起这种情况发生的最简单方法就是使用 Web Cookie（或其他区分用户的信息），而且检查 HTTP 头来找到这些 Cookie。

最后要注意的一点，尽管我们已经描述了 Web 站点的设计，同样可以用其他类型的服务器来构建服务器农场。一个例子是 UDP 之上的流媒体服务器。此时仅仅需要改变的是前端，它必须能均衡这些请求的负载（将使用与 Web 请求不同的协议头字段）。

Web 代理

使用 HTTP 发送 Web 请求和响应。在 7.3 节中，我们介绍了浏览器如何缓存响应和重用这些缓冲信息来回答未来的请求。浏览器使用各种头字段和规则来确定一个缓存的网页副本是否还很新鲜。在这里，我们不再重复这些内容。

高速缓存因为缩短了响应时间并减少了网络负载，因此能提高性能。如果浏览器可以自己判断缓存的页面是新鲜的，那么页面可以立即从缓存中提取出来，根本不会产生网络流量。然而，即使浏览器必须询问服务器来确认该页面是否依然新鲜，响应时间也缩短，网络负载也得到降低，特别对大页面尤为如此，因为只需要发送一个小的消息。

然而，浏览器可以做的最好事情是缓存用户以前访问过的所有网页。从我们关于流行网站的讨论中，你可能还记得一些热门的网页被许多人多次访问，也有很多很多不那么流行的网页。实际上，这种特性限制了浏览器缓存的有效性，因为大量的页面只被给定用户访问一次，而这些网页总是必须从服务器获取。

使缓存更有效的策略之一是在多个用户之间共享缓存。通过这种方式，为一个用户获取的一个页面可以在另一个用户发出同样请求时返回给他。没有浏览器缓存，这两个用户都需要从服务器获取该页面。当然，这种共享不适用于加密的流量和不可缓存的页面（比如，当前股票的价格），因为前者需要认证，后者是由程序返回。特别是由程序创建的动态网页，仍处于不断增长的情况，因此缓存它是无效的。不过，也有大量的网页可被许多用户访问，而且无论哪个用户发出请求看到的都是同样内容（例如，图像）。

Web 代理（Web proxy）是可以被多个用户共享的一个高速缓存。代理代表别人做某种行为，比如用户。有许多不同种类的代理。例如，一个 ARP 代理代表一个在别处的用户（他不能自己回答）应答 ARP 请求。Web 代理代表它的用户获取 Web 请求。它通常提供缓存的 Web 响应，而且因为它被多个用户共享，因此它有一个比所有用户浏览器都大得多的高速缓存。

当采用代理时，典型的设置是一个组织为它的所有用户运行一个 Web 代理。该组织可能是一个公司或一个 ISP。组织和个人都可以通过加快用户的 Web 请求，并减少带宽需求来获得收益。虽然独立于使用，定价对最终用户来说都是一样的，大多数公司和 ISP 根据用户所使用的带宽来收费的。

这种设置如图 7-66 所示。为了使用代理，每个浏览器必须配置成向代理而不是页面的真实服务器发出页面请求。如果代理有相应的页面，它就立刻返回页面。如果它没有，它就从服务器获取页面，并将它添加到缓存以备将来使用，同时返回给请求该页面的客户。

除了向代理而不是真正服务器发送 Web 请求外，客户还要用它的浏览器高速缓存来执行它们自己的缓存。只有在浏览器尝试以自己的高速缓存页面无法满足请求之后，才会咨询代理。也就是说，代理提供了一个第二级的缓存。

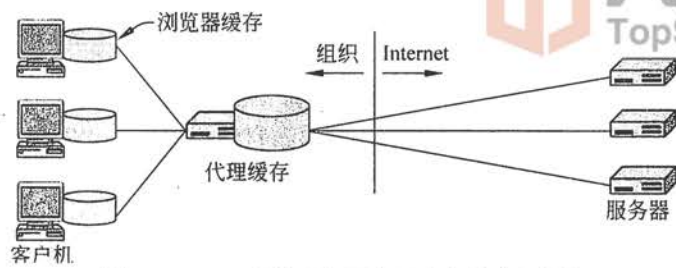


图 7-66 Web 浏览器与服务器之间的代理缓存

还可以加入进一步的代理来提供更多一级的缓存。每个代理（或浏览器）向其**上行流代理**（upstream proxy）发出请求。每个上行流代理为**下行流代理**（downstream proxy）（或浏览器）缓存页面。因此，有可能一个公司内的浏览器使用一个公司代理，而公司代理再使用一个 ISP 代理，该代理直接和 Web 服务器联系。然而，一般来说，我们给出的单级代理缓存（如图 7-66 所示）实际上也往往能获得大部分的潜在好处。相应存在的问题是流行的长尾效应。针对网站流量的研究表明，当用户数量达到一个小公司的规模（比如，100 人）时，共享缓存特别有益。随着人数增长较大，共享缓存的好处变得边缘化，由于缺乏存储空间，不流行的得不到缓存（Wolman 等，1999）。

Web 代理还提供了额外的好处，这往往是决定部署它们的一个因素。好处之一是内容过滤。管理员可以配置代理过滤掉一些被列入黑名单的网站，或者把代理配置成其他过滤器，对发出的某些请求予以过滤。例如，许多系统管理员对员工在公司上班时间观看 YouTube 视频（或更糟糕的，观看色情）会皱起眉头，因而会相应地设置它们的过滤器。用代理的另一个好处是隐私或匿名，代理作为盾牌向服务器隐匿了用户的身份。

7.5.3 内容分发网络

服务器农场和 Web 代理有助于建立大型网站，并提高 Web 性能，但对于构建一个真正流行的网站，在全球范围内提供内容服务，这些还不够。对于这些网站，需要不同的方法。

内容分发网络（CDN，Content Delivery Network）彻底扭转了传统的 Web 缓存想法。与客户在附近的缓存中寻找请求的页面副本不同，CDN 为客户提供了这样的信息：在一组分布在不同地点的节点中哪个节点拥有所请求页面的副本，并且指示客户端使用附近的节点作为服务器。

图 7-67 给出了一个 CDN 网络中数据分发所遵循的路径例子。这是一棵树。CDN 的源服务器将一份内容副本分发到 CDN 中的其他节点。在这个例子中，这些节点分别位于悉尼、波士顿和阿姆斯特丹，图中用虚线表示。然而，客户端从就近的 CDN 节点获取页面，图中用实线表示。这样，在悉尼的客户获取的都是存储在悉尼的页面副本，他们不会从源服务器处获取页面，源服务器可能在欧洲。

使用树结构有 3 个好处。首先，内容的分发具备可扩展性。随着越来越多的客户需要，可以使用更多的 CDN 节点；而且当在 CDN 节点之间分发变成瓶颈时，可以采用更多的树层次。不管有多少客户，树结构总是有效的。源服务器不会超载，因为它通过 CDN 节点

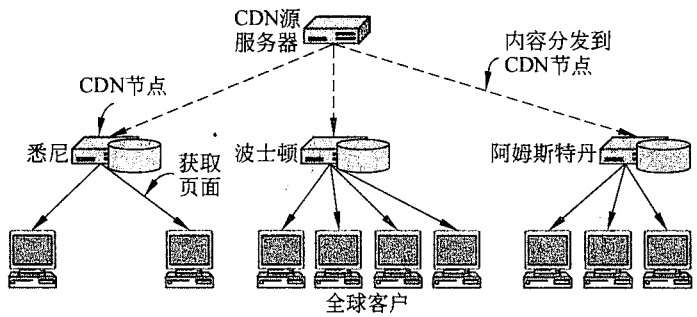


图 7-67 CDN 分发树

树和许多客户交谈；而它自己不需要应答每个页面请求。其次，每个客户都享有不错的性能，因为他是从附近的服务器而不是一个遥远的服务器那里获取页面。这是因为建立连接所需要的往返时间较短，而往返时间越短 TCP 慢启动得更快，而且较短的网络路径经过 Internet 上拥塞区域的可能性较小。最后，放置在网络上的总负荷也保持在最低水平。如果 CDN 节点都放置在恰当的位置，一个给定页面的流量应该恰好通过网络中各个部分一次。这点非常重要，最终总会有人要为网络带宽付费。

采用分发树的想法很直截了当。最简单的是如何组织客户使用这棵树。例如，代理服务器似乎提供了一个解决方案。考查图 7-67，如果每个客户端都配置成使用悉尼、波士顿或阿姆斯特丹 CDN 节点作为缓存的 Web 代理，数据的分发将按照树的结构来进行。然而，这种策略未能在实际中奏效，原因主要有 3 个。第一个原因，网络中位于一个给定区域的客户可能属于不同的组织，所以他们可能会使用不同的 Web 代理。回想一下，大量的客户端共享缓存所获得的利益有限，而且从安全方面的考虑出发，缓存机制通常不能跨组织共享。其次，可能存在多个 CDN，但每个客户端只使用一个代理缓存。客户端应该选择哪个 CDN 作为其代理呢？最后，也许所有问题中最实际的是客户端配置了 Web 代理。他们可能会也可能不会再配置一个 CDN 来受益于内容分发，而且对此 CDN 能做的很少。

支持一棵分发树的另一种简单方法是使用**镜像**（mirroring）。在这种方法中，源服务器跟以前一样将内容复制给 CDN 节点。不同网络区域中的 CDN 节点称为**镜像点**（mirror）。源服务器上的网页包含了指向不同镜像点的显式链接，通常告诉用户这些镜像点的位置。这种设计可以让用户手动选择一个附近的镜像点来下载内容。镜像的典型应用是把一个大的软件包放在不同位置的镜像点上，例如，美国东海岸和西海岸、亚洲和欧洲。镜像站点通常是完全静态的，而且选择的站点可以保持稳定数月或数年。它们是一种经受住考验的技术。然而，它们依靠用户做内容的分发，因为镜像点实际上是不同的 Web 站点，即使它们被链接在一起。

第三种方法克服了前两种方法的困难，使用了 DNS 技术，因而称为 **DNS 重定向**（DNS redirection）。假设一个客户要获取 URL 为 http://www.cdn.com/page.html 的页面。为了提取页面，浏览器将使用 DNS 把 www.cdn.com 解析为一个 IP 地址。DNS 查找过程以通常的方式进行。通过使用 DNS 协议，浏览器了解到 cdn.com 域名服务器的 IP 地址，然后与该域名服务器联系，请求它解析 www.cdn.com。现在到了真正展示聪明才智的地方。该域名服务器同时作为 CDN 节点运行。它不是为每个请求返回相同的 IP 地址，而是根据发出请求的客户端 IP 地址返回不同的查询结果。答案将是最靠近客户端的 CDN 节点的 IP 地址。也

就是说，如果在悉尼的一个客户请求解析 www.cdn.com，域名服务器将返回悉尼 CDN 节点的 IP 地址；但如果阿姆斯特丹的一个客户端发出了同样的要求，域名服务器将返回阿姆斯特丹 CDN 节点的 IP 地址。

根据 DNS 的语义，这种策略是完全合法的。我们之前已经看到过域名服务器可能会返回变化的 IP 地址列表。在域名解析后，悉尼的客户端将直接从悉尼 CDN 节点获取页面。由于 DNS 缓存，对相同“服务器”页面的进一步请求将直接从悉尼 CDN 节点获取。整个一系列序列步骤如图 7-68 所示。

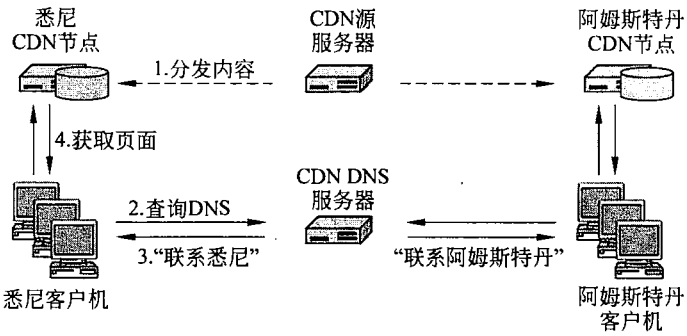


图 7-68 CDN 节点使用 DNS 将客户指向附近的 CDN 节点

上述过程中的一个复杂问题是怎样才算找到最近的 CDN 节点，以及如何到达它。要定义最近的 CDN 节点，它并不是真的和地理位置相关。在将客户端映射到某个 CDN 节点时至少有两个因素需要考虑。一个因素是网络的距离。客户端应该有一条距离短、容量高的网络路径到达 CDN 节点。这样情形下才能做到快速下载。CDN 使用一张它们以前计算的地图，把客户端的 IP 地址转换成它的网络位置。选定的 CDN 节点可能就在作为直线的最短距离上，也有可能不在最短距离上。重要的是，要综合考虑网络路径长度和途中的任何容量限制。第二个因素是 CDN 节点的当前负载。如果 CDN 节点已经超载，它们的传送反应就会缓慢，就像 Web 服务器超载一样，我们应该首先避免使用这样的服务器。因此，均衡整个 CDN 节点的负载非常有必要，应该把一些客户端映射到稍微远一点但负载轻一点的节点。

将 DNS 用作内容分发技术率先由 Akamai 在 1998 年提出的，那时 Web 在早期快速增长的负载下不堪重负。Akamai 是第一个主要的 CDN 提供商，并且成了行业的领头羊。也许比通过 DNS 将客户端连接到附近节点的想法更巧妙的是其经营的激励结构。公司给 Akamai 支付费用，以换取将它们的内容传递给客户，因此它们有客户喜欢使用的响应网站。CDN 节点必须放置在具有良好连通性的网络位置，最初这意味着在 ISP 网络内部。对于 ISP 来说，在它们的网络中放置一个 CDN 节点是有好处的，即 CDN 节点能削减它们所需要的上游网络带宽量（必须付费），就像设置代理缓存一样。此外，CDN 节点能提高 ISP 客户的响应性，使得 ISP 在客户们的眼里变得很好，从而比没有 CDN 节点的 ISP 拥有更大的竞争优势。这些好处（对 ISP 来说没有任何成本）使得安装一个 CDN 节点对 ISP 来说是简直不费吹灰之力。因此，CDN 不仅使得内容提供商、ISP 和客户都受益，而且还能赚钱。自 1998 年以来，其他公司也开始进入这个行业，所以现在 CDN 是一个有多家提供商的竞争产业。

正如这种描述所暗示的那样，多数企业没有建立它们自己的 CDN。相反，它们使用 CDN 提供商的服务来实际传送它们的内容，例如 Akamai。为了让其他公司使用 CDN 服务，我们需要添加最后一个步骤。

当 CDN 与 Web 网站所有者的代表签署内容分发的合同后，网站所有者便向 CDN 提供自己网站的内容。这个内容被推送到 CDN 节点。此外，网站所有者可以重写链接到内容的任何网页。注意，这里不是链接到其网站上的内容页面，而是通过 CDN 与内容链接的页面。为了说明这项方案是如何工作的，请考虑蓬松视频（Fluffy Video）网页的源代码，如图 7-69(a)所示。在预处理后，被转化为图 7-69(b)，并且作为 www.fluffyvideo.com/index.html 被放置在蓬松视频服务器。

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>
<a href="koalas.mpg"> Koalas Today </a> <br>
<a href="kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

(a)

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>
<a href="http://www.cdn.com/fluffyvideo/koalas.mpg"> Koalas Today </a> <br>
<a href="http://www.cdn.com/fluffyvideo/kangaroos.mpg"> Funny Kangaroos </a>
<br>
<a href="http://www.cdn.com/fluffyvideo/wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

(b)

图 7-69

(a) 原始 Web 页面；(b) 与 CDN 链接之后的某些页面

当用户在他的浏览器上输入 URL 为 www.fluffyvideo.com 时，DNS 返回蓬松视频自己网站的 IP 地址，从而允许用户以正常的方式提取主页（HTML）。当用户点击该主页上的任何超链，浏览器会请求 DNS 查询 www.cdn.com。这次查询联系的是 CDN 的 DNS 服务器，它返回的是附近 CDN 节点的 IP 地址。然后浏览器给 CDN 节点发送一个常规 HTTP 请求，例如/fluffyvideo/koalas.mpg。该 URL 标识了要返回的页面，从 fluffyvideo 路径开始，因此 CDN 节点可以区分它为之服务的不同公司的请求。最后，所请求的视频被返回，用户可以看到可爱的毛绒绒动物了。

把托管给 CDN 的内容和内容所有者主管的入口页面分开，隐藏在这背后的策略是让 CDN 移动大量数据的同时给内容所有者以控制权。大多数入口网页微小，只是些 HTML 文本。这些网页通常链接到大文件，比如视频和图像。正是这些大文件由 CDN 来提供服务，即使 CDN 的使用对用户完全透明。网站看起来都一样，但执行得更快。

站点们使用共享 CDN 还有另一个优势。一个 Web 站点的未来需求可能很难预测。通常情况下，潮水汹涌般的需求称为**闪群**（flash crowd）。当发布最新产品、有一场时装秀和发生其他事件，或者该公司卷入新闻时，这种激增的访问需求就可能发生。甚至一个以前未知的、未被访问过的落后网站也可能会突然成为 Internet 的焦点，如果它有新闻价值和被热门网站链接。由于大多数网站没有准备好处理大量激增的流量，其结果就是其中的许多网站当流量激增时立即崩溃掉。

关于这点有一个很好的案例分析。通常情况下，佛罗里达州的网站不是个繁忙的地方，虽然你可以查看有关佛罗里达州公司、公证员、文化事务，以及有关投票和选举等的信息。某种奇怪的原因，2000 年 11 月 7 日（布什与戈尔的美国总统选举日），一大堆人突然对该网站上有关选举结果的页面感兴趣。因此，突然间该网站成了世界上最繁忙的网站之一，结果自然是网站彻底崩溃。如果它早点使用 CDN，或许就会幸免于难。

通过使用 CDN，一个网站能拥有提供超大内容服务的访问能力。最大的 CDN 有数以万计的服务器部署在世界各地的不同国家。由于只有少数网站在任何一个时间（定义的）将会遇到闪群，这些网站可以使用 CDN 的能力来处理负载，直到风暴过去。也就是说，CDN 可以迅速扩大网站的服务能力。

前面的讨论只是针对 Akamai 如何工作的一个简单描述。实际上，还有更多的细节值得重视。在我们例子中描绘的 CDN 节点通常是一堆集群的机器。DNS 的重定向体现在两个层次：一个将客户映射到一个近似的网络位置，另一个将负载遍布在该位置上的节点。可靠性和性能都需要关注的两个方面。为了能够将一个客户从集群中一台机器转移到另一台机器，第二个层次上的 DNS 应答具有很短的 TTL，因此客户端很快就能重复解析。最后，虽然我们将注意力集中在分发静态对象，比如图像和视频，CDN 同样可支持动态页面创建、流媒体以及其他更多的内容。如需更多的有关 CDN 信息，请参阅（Dilley 等，2002）。

7.5.4 对等网络

不是每个人都能在世界各地设立 1000 个节点的 CDN 来分发他们的内容（实际上，租用 1000 台分布在全球各地的虚拟机并不是很难，因为有一个发展良好并且有竞争力的托管行业；但是，建立一个 CDN 仅仅是获得节点的开始）。幸运的是，对于其他人，还有一个替代品可用。它简单易用，并且可以分发大量的内容。它就是**对等网络**（P2P，Peer-to-Peer）。

P2P 网络的异军突起始于 1999 年。第一个广泛的应用是大规模的犯罪行为：50 万 Napster 用户交换没有得到版权拥有人许可的盗版歌曲，直到 Napster 在一片很大的争议声中由法院裁定被关闭。不过，对等技术是那样的有趣，并且有合法的用途。其他 P2P 系统的不断发展，吸引了用户的极大兴趣，因此 P2P 流量迅速崛起使得 Web 流量黯然失色。如今，BT（BitTorrent）是目前最流行的 P2P 协议。它的使用是如此之广泛，大家共享（授权和公共领域）视频以及其他内容，以至于它占有了所有 Internet 流量中的很大一部分。在本节我们将看到这点。

一个 P2P 文件共享网络的基本思路是将多台计算机连到一起并且把它们的资源集中起来形成一个内容分发系统。这些计算机往往是简单的家用电脑。它们并不一定是 Internet 数据中心的机器。这些计算机称为**对等节点**（peer），因为每一个都可交替充当另一个的客

户端，获取其内容；也可作为服务器，提供内容给其他对等节点。使对等系统有趣的是没有专门的基础设施，这点和 CDN 不同。每个人都参与到分发内容的任务中，而且通常还没有中央控制点。

许多人兴奋于 P2P 技术，因为它看起来就像给小家伙授予了很大的自主权。究其原因，不仅在于只有一个大型公司才能运行一个 CDN，而且任何一个人只要有一台计算机都可以加入一个 P2P 网络。正是 P2P 网络具有强大的内容分发能力，因而可以和最大的网站分庭抗礼。

考虑一个 P2P 网络，假设平均由 N 个用户组成，每个用户具有 1 Mbps 的宽带连接。P2P 网络的上载总容量，或者用户可以往 Internet 发送流量的速率是 N Mbps；而下载容量，或用户可以接收流量的速率也是 N Mbps。而且，每个用户可以同时上传和下载，因为他们在每个方向上都有 1 Mbps 链路。

一个并不明显但却是真实的结论是，事实证明所有的容量都可用来高效地分发内容，甚至在所有其他用户共享一个文件副本的情况下。为了看清楚是如何做到这点的，请想象所有的用户组织成一棵二叉树，每个非叶结点的用户给其他两个用户发送。这棵树把文件的单个副本运载到所有的其他用户。为了让尽可能多的用户在任何时候（因此分发大文件的延迟较低）使用尽可能多的上传带宽，我们需要管道化用户的网络活动。试想，该文件被划分成 1000 个小片。每个用户可以从树的上方某个地方接受一个新的小片，并且在同一时间把先前收到的一小片向树的下方发送。这样，一旦管道开始后，在一部分的小片（等于树的深度）被发送出去后，所有的非叶结点用户都将忙于上传文件给其他用户。由于有大约 N/2 个非叶结点的用户，该树的上传带宽是 N/2 Mbps。我们可以重复这一招，通过交换叶结点和非叶结点的角色，创建另一个树来使用其他 N/2 Mbps 的上传带宽。综合起来，这样的结构就使用了全部的容量。

这个说法意味着 P2P 网络能自我调整。它们可用的上传容量随着用户的下载需求而水涨船高。从某种意义上说，它们总是“足够大”，根本不需要任何专门的基础设施。相反，即使一个大型 Web 站点的容量也是固定的，并且要么过大，或者过小。考虑一个站点，只有 100 个集群，每个集群拥有 10 Gbps 容量。这种巨大的容量对于只有少量用户的情况并没多大帮助。该网站不能以高于 N Mbps 的速率给 N 个用户发送信息，因为速率的限制在用户这边而不在网站。而当有超过一百万个 1 Mbps 用户时，该网站又不能足够快地泵出数据让所有用户保持下载。这看起来像一个庞大的用户群，但大型 BitTorrent 网络（例如，Pirate Bay）声称有超过 10 000 000 用户。就我们的例子而言这更像 10 Tbps！

你应该对这些数字的简要分析半信半疑（或者更好，干脆不信），因为它们过于简化了情况。P2P 网络的一个重大挑战在于当面临各种各样的用户，而且用户具有不同的下载和上传容量时，如何更好地使用带宽。尽管如此，这些数字仍然表明了 P2P 的巨大潜力。

P2P 网络之所以如此重要还有另外一个原因。CDN 和其他集中运行服务的系统将提供商置于一个拥有许多用户个人信息宝库的位置，从用户的浏览喜好和网上购物，到人们的地点和电子邮件地址等悉数掌握。这些信息可以被用来提供更好以及更个性化的服务，但也可以用来侵犯人们的隐私。后者可能是故意的行为（比如作为新产品推销的手段），也可能是偶然的披露或妥协。在 P2P 系统中，不可能有单个提供商具备监测整个系统的能力。这并不意味着 P2P 系统一定会提供隐私保护，因为用户在一定程度上是相互信任的。它只

意味着 P2P 可以提供比集中式管理系统形式不同的隐私保护措施。P2P 系统现在正在探索超出文件共享的服务（例如，存储、流媒体），时间会告诉我们这样做的好处是否显著。

P2P 技术的发展遵循两条相关的路径。从更实际的角度出发，每天都有正在使用的系统。这些系统中最著名的是基于 BitTorrent 协议。在学术方面来看，一直存在着对 DHT（分布式哈希表）算法的浓厚研究兴趣，这是 P2P 系统作为整体运行性能良好的核心技术，但不需要依靠任何集中式组件。我们将分别了解这两种技术。

BitTorrent

BitTorrent 协议是 2001 年由 Brahm Cohen 开发的，目的是让一组同行们方便快捷地共享文件。有几十个免费可用的客户端运行这个协议，就好像有许多浏览器和 Web 服务器使用 HTTP 协议通信一样。BitTorrent 协议作为一个开放标准，可从 www.bittorrent.org 获得。

在一个典型的对等系统中，就像使用 BT 形成的系统，每个用户有一些可能对其他用户来说感兴趣的信息。这种信息可能是免费软件、音乐、影片、照片等。在这样的设置中共享内容必须解决 3 个问题：

（1）一个对等用户如何找到具有自己想下载内容的其他对等用户？

（2）对等用户们如何复制内容以便为大家提供高速下载？

（3）对等用户们如何相互鼓励上传内容给他人同时为自己下载内容？

第一个问题确实存在，因为不是所有的对等用户都拥有所有的内容，至少在最初阶段是这样的。BitTorrent 所采取的方法是为每个内容提供商创建一个内容描述，称为**种子文件**（torrent）。种子文件是比内容小很多的数据，被对等用户用来验证它从其他对等用户处下载的数据的完整性。想下载该内容的其他用户必须先取得种子文件，也就是说，必须找到广告内容的 Web 页面。

种子文件只是一种指定格式的文件，它包含了两类关键信息。一类信息是一个**跟踪器**（tracker）名字，这是一个服务器，将对等用户引导到种子文件内容。另一类信息是一个大小相等片或**块**（chunk）的清单，它们组成了内容。不同的种子文件可用不同大小的块，通常块的大小从 64 KB 到 512 KB 不等。种子文件包含每个块的名称，作为块的 160 位 SHA-1 哈希给出。我们将在第 8 章中介绍加密哈希，比如 SHA-1。现在，你可以把哈希想象成一个很长更安全的校验和。给定块的大小和哈希表，种子文件至少比内容小三个数量级，因此它可以被快速传送。

为了下载由种子文件所描述的内容，一个对等用户首先与该种子文件联系。跟踪器是一个服务器，它维护着一个正在主动上传和下载该内容的所有其他对等用户列表。这一组对等用户称为用户**集群**（swarm）。群的成员定期与跟踪器联系并向它报告自己仍然活跃，当他们离开群时也向跟踪器报告。当一个新的对等用户联系跟踪器要加入群时，跟踪器告诉它群内有那些对等用户。获取种子文件并和跟踪器联系是下载内容的前两个步骤，如图 7-70 所示。

第二个问题是如何以一种能够快速下载的方式来共享内容。当一个群首次形成时，有些对等用户必须拥有组成内容的所有块。这些对等用户就是所谓的**种子**（seeder）。要加入该群的其他对等用户没有块；他们是需要下载内容的对等用户。

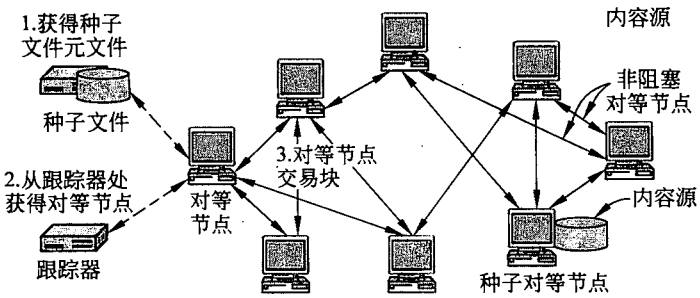


图 7-70 BitTorrent

在一个对等节点参与一个用户集群期间，它同时从其他对等节点处下载缺少的块，并且给其他对等节点上传他拥有但他们需要的块。这种交易显示在图 7-70 内容分发的最后一步。随着时间的推移，该对等节点收集到越来越多的块，直到他下载了内容的所有块。对等节点可以随时离开群（以及以后返回）。通常，一个对等节点在完成自己的下载后会停留一个短时期。随着对等节点的来来往往，用户集群的流失率相当高。

为了上面所述的方法能正常工作，每个块应该被许多对等节点可用。如果每个人都以相同顺序得到块，很可能许多对等节点都依赖于下一块的种子节点。这样容易形成瓶颈。相反，对等节点彼此之间交换各自拥有的块清单；然后，他们选择罕见的很难找到的块下载。我们的想法是下载一个难得的块将生成一个它的副本，这样使得该块更容易为其他对等节点找到并下载。如果所有的对等节点都这样做，那么经过短暂的一段时间后所有的块都将成为广泛可用。

第三个问题也许是最有趣的。CDN 节点是专门为了给用户提供内容而设置的。P2P 节点却不是。它们是用户的计算机，而且用户可能对获得一个电影比帮助其他用户下载更感兴趣。那些只想从系统中获取资源而没有实物贡献的节点称为**搭便车**（free-rider）或**吸血鬼**（leecher）。如果这样的用户太多，那么系统将无法正常工作。早期的 P2P 系统滋生了它们（Saroiu 等，2003），因此 BitTorrent 寻找一切机会来最少化它们。

BitTorrent 客户端采用的方法是奖励那些表现出良好上传行为的对等节点。每个对等节点随机采样其他对等节点，在给它们上传块的同时检索他们的块。该对等节点只与少数提供最高下载性能的对等节点们继续交易块，同时也随机尝试其他对等节点以便发现更好的合作伙伴。随机尝试对等节点们给了那些新加入的对等节点获得初始块的机会，以便他们可以与其他对等节点交易。当前正在与一个节点交换块的对等节点称为**非阻塞**（unchoked）。

随着时间的推移，这种算法趋向于匹配那些相互之间具有可比性上传和下载速率的对等节点。一个对等节点为其他对等节点作出的贡献越多，它预期的回报就越大。使用一组对等节点也有助于饱和一个对等节点的下载带宽，由此提高性能。反之，如果一个对等节点没有给其他对等节点上传块，或者做得非常缓慢，它迟早会被切断或**阻塞**（choked）。这种策略打击了反社会行为，即对等节点在群内搭便车。

抑制算法（choking algorithm）有时被描述成一种实施**针锋相对**（tit-for-tat）的策略，鼓励对等节点们反复交互合作。然而，它并不能在很大程度上防止客户游戏系统（Piatek 等，2007）。不过，注意到这个问题并且使一般用户更难以搭便车的机制有可能对 BitTorrent 的成功作出了贡献。

正如你可以从我们的讨论中看到，BitTorrent带来了丰富的词汇。有种子文件（torrent）、用户集群（swarm）、吸血鬼（leecher），种子节点（seeder）以及冷落（snubbing）、抑制（choking）、潜伏（lurking）等更多其他词汇。欲了解更多信息，请参阅有关BitTorrent的短文章（Cohen，2003）和从www.bittorrent.org开始网站查询。

DHT——分布式哈希表

2000年前后出现的P2P文件共享网络在研究界引发了很大的兴趣。P2P系统的实质是避免了集中管理CDN结构和其他系统结构。这是一个显著的优势。集中管理的组件随着系统增长得非常大将成为瓶颈，而且是一个单故障点。中央组件可被用来作为一个控制点（比如关闭P2P网络）。但是，早期的P2P系统仅仅是部分分散的，否则如果它们完全分散就没有效率了。

我们刚才所描述的BitTorrent传统形式采用了对等传输，并且每个用户集群有一个集中跟踪器（tracker）。正是这个跟踪器成为对等系统实行分散化的最困难部分。关键问题是如何找出哪些对等节点拥有正在寻求的具体内容。例如，每个用户可能有一个或多个数据表项，比如歌曲、照片、程序、文件和其他用户可能想读的东西。其他用户如何找到他们想要的这些东西呢？制作一个索引表，即给出谁拥有什么的清单倒是简单，但它必须是集中的。让每个对等节点保持一份它自己的索引于事无补。真正的它应该是分布式的。然而，它需要做如此之多的工作才能保持所有对等节点的最新索引（因为内容是在系统中移动的），以至于这是不值得的努力。

研究界想解决的问题是在P2P系统中有没有可能建立一个完全分布式的但表现良好的P2P索引。所谓的表现良好，我们的意思是指三件事情。首先，每个节点只需要保留少量的有关其他节点的信息。这个特性意味着它保持最新索引的代价不会很昂贵。其次，每个节点可以快速查看索引中的表项。否则，它就不是个非常有用的索引。最后，每个节点可以同时使用索引，即使其他节点来来去去。这个属性意味着索引的性能随着节点数量的增长而越来越好。

问题的答案是："是的"。在2001年发明了4种不同的解决方案。它们分别是Chord（Stoica等，2001）、CAN（Ratnasamy等，2001）、Pastry（Rowstron和Druschel，2001）和Tapestry（Zhao等，2004）。之后不久，又有许多其他的解决方案被发明出来，包括已经被实际使用的Kademlia（Maymounkov和Mazieres，2002）。该解决方案称为**分布式哈希表**（DHTs，Distributed Hash Tables），因为索引的基本功能将一个关键字映射到一个值。这简直就是一个哈希表，当然，解决方案是分布式的版本。

DHT需要在节点之间的通信采用一个正规结构，正如我们将看到的那样。这种行为有别于传统的P2P网络，在传统的P2P系统中对等节点之间可使用任何连接。出于这个原因，DHT称为**结构化的P2P网络**（structured P2P networks）。传统的P2P协议构建的是**非结构化的P2P网络**（unstructured P2P networks）。

我们将要描述的DHT解决方案是Chord。作为一个场景，考虑如何将BitTorrent中传统使用的集中式跟踪器替换成一个完全分布式的跟踪器。Chord可以解决这个问题。在这种场景下，总索引是所有用户集群的列表，一台计算机可以加入到某个用户集群以便下载内容。查询索引的关键字是内容的种子文件描述，它作为所有内容块的哈希表唯一

地标识了可以下载该内容的一个用户集群。针对每一个关键字存在哈希表中的对应值是组成该用户集群的对等节点列表。这些对等节点是要下载内容的计算机。一个想要下载内容（比如电影）的用户只有该电影的种子文件描述符。在缺乏中央数据库的情况下，DHT 必须回答的问题是如何找出一个对等节点（从 BitTorrent 数以百万计的节点中），从他那里下载电影？

Chord 的 DHT 由 n 个参与节点组成。它们在我们的场景中是运行 BitTorrent 的节点。每个节点都有一个 IP 地址，通过该地址可以联系上它。总索引扩播到这些节点。这隐含着每个节点都存储被其他节点使用的索引位和片。Chord 的关键部分是在一个虚拟空间中使用标识符来导航索引，而不是使用节点的 IP 地址或某个像电影的内容名称。从概念上讲，标识是简单的 m 位数字，它以降序的方式排列成环。

要将一个节点地址变成一个标识符，需要使用一个哈希函数 hash 将该地址映射到一个 m 位的数字。Chord 使用 SHA-1 作为哈希函数 hash。这个函数与我们在描述 BitTorrent 时提到的相同。我们将在第 8 章讨论加密时再来考查它。现在，只要知道它仅仅是一个函数，以一个可变长度的字节串作为参数，并产生一个高度随机的 160 位数字。因此，我们可以使用它把任何 IP 地址转换成一个 160 位的数字，该数字称为**节点标识符**（node identifier）。

在图 7-71(a)中，我们显示了 m = 5 的节点标识符环（暂时忽略中间的弧线）。某些标识符对应到节点，但大多数标识符都没有节点可对应。在这个例子中，具有标识符 1、4、7、12、15、20 和 27 的节点才对应于实际节点，如图中的阴影部分所示；其余的节点都不存在实际节点。

现在，让我们把函数 successor(k)定义为环中紧跟在 k 之后的第一个实际节点的节点标识符。例如，successor(6)=7、successor(8)=12 和 successor(22)= 27。

关键字（key）也可以由哈希函数产生，即用 hash（即 SHA-1）哈希一个内容名称来生成的 160 位数字。在我们的场景中，内容名称为 torrent。因此，为了将 torrent（torrent 描述文件）转换成关键字，我们计算 key=hash（torrent）。这种计算只是一个调用 hash 的本地过程。

为了启动一个新的用户集群，一个节点需要将一对新的“关键字-值”（key-value）插入到索引中，该新的值对由（torrent, my-IP-address）组成。而要做到这一点，节点必须请求 successor（hash（torrent））来存储 my-IP-address。这样，该索引就被随机地分布在节点上。为了容错，p 个不同的哈希函数可以用来在 p 个节点存储该数据，但这里我们将不考虑深入的容错问题。

在 DHT 构造后的某个时间，另一个节点希望找到一个种子文件，以便它可以加入该用户集群和下载内容。节点查找种子文件的过程是这样的：首先通过哈希得到关键字 key；然后使用 successor(key)找到存储了相应值的节点的 IP 地址。该值是一张该用户集群中的对等节点列表；节点可以把它自己的 IP 地址添加到列表中，并联系其他对等节点通过 BitTorrent 协议下载内容。

第一步很容易，第二步就有点困难了。为了使人们有可能找到某个特定关键字对应的节点的 IP 地址，每个节点都需要维护特定的行政管理结构数据。其中之一是沿着节点标识符组成的环，找到该节点后继节点的 IP 地址。例如，在图 7-71 中，节点 4 的后继节点是节点 7，节点 7 的后继节点是节点 12。

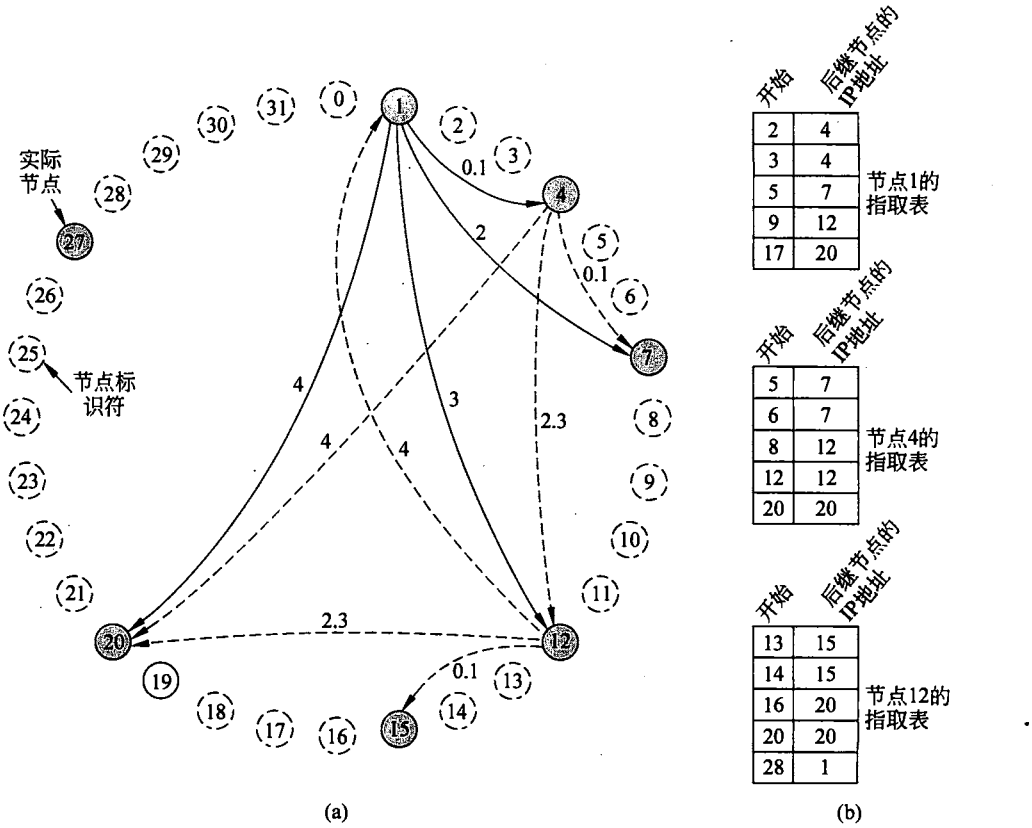


图 7-71

(a) 一组32个节点标识符排列成环，阴影节点对应着实际机器，弧线显示出自节点1、节点4和节点12的指向，弧线的标记是表索引；(b) 指取表实例

现在可以如下进行查找了：请求节点向其后继节点发送一个包，该包含有其IP地址和它正要寻找的关键字。该包沿着环传播，直到它找到正在寻找的节点标识符的继任者。该节点检查是否有和关键字匹配的任何信息，如果有的话，它就将信息直接返回给请求节点，即它的IP地址。

然而，在一个大规模的对等系统中线性搜索所有节点效率非常低，因为每次搜索所需的平均节点数是n/2。为了大大加快搜索速度，每个节点还维持一张表，该表被Chord称为**指取表**（finger table）。指取表有m个表项，索引为从0到m–1，每个指向一个不同的实际节点。每个表项有两个字段：start和successor(start)的IP地址，如在图7-71(b)中所示的三个例子。节点标识符为k的节点的第i个表项的字段值是：

start = $k + 2^i$ （模 2^m）

successor(start[i])的IP地址

请注意，每个节点都存储了相对数量较少的节点的IP地址，而且其中的大部分都相当接近节点标识符。

使用指取表，在节点k上查找关键字key的过程如下所示。如果k介于k和successor(k)之间，那么持有key信息的节点就是successor(k)，并且终止搜索；否则，搜索指取表，找到这样的表项，其start字段最接近key前任。然后，一个请求被直接发送给指取表中该表

项的IP地址，请求它继续搜索。因为它接近key，但又低于它，因此有很好的机会，只需要少量的额外查询就能够返回答案。事实上，因为每次查询都把抵达目标的剩余距离减半，因此可以表明查找的平均次数是$\log_2 n$。

作为第一个例子，考虑在节点1上查询key=3。由于节点1知道节点3位于它和它的继任者4之间，因此所需的节点就是节点4，终止搜索，返回节点4的IP地址。

作为第二个例子，考虑在节点1上查询key=16。因为16不在1和4之间，因此查询指取表。最接近16的前任是9，因此请求被转发到节点9的表项的IP地址，即节点12。节点12自己也不知道答案，因此它查找最接近16的节点，发现是节点14，它产生节点15的IP地址。然后，查询再被转发给节点15。节点15观察到节点16位于它和它的继任者（20）之间，所以它返回节点20的IP地址给调用者，该项工作回到节点1。

由于节点可以随时加入和离开，因此Chord需要一种方法来处理这些操作。我们假设，当系统开始运营时，它足够小到节点之间可以直接交换信息建立第一个环和指取表。之后，需要一个自动化过程。当一个新的节点r想加入时，它必须联系一些已有的节点，并要求它们查找自己的successor(r)的IP地址。接下来，新节点请求successor(r)告知它的前任。然后，新节点同时要求这两个节点在它们之间的环上插入r。例如，如果在图7-71中节点24想加入，它请求任何一个节点查询successor(24)，结果是27。然后，它请求节点27，获知自己的前任（20）。之后它告诉这两个节点有关自己的存在，节点20把节点24作为它的继任者，并且节点27把节点24用作其前任。此外，节点27把21～24范围内的关键字处理移交给节点24，现在属于24节点的管辖范围。至此，节点24才算完全插入到环中。

然而，许多指取表现在是错的。为了予以纠正，每个节点运行一个后台进程，该进程通过调用successor定期重新计算每个指取表。当这些查询结果命中一个新节点时，相应的指取表项被更新。

当一个节点温和地离开用户集群时，它把自己的关键字移交给其继任者，并通知其前任自己要离开了，以便该前任可以链接到离开节点的继任者。当一个节点出现问题崩溃时，就会出现一个问题，因为它的前任不再有一个有效的继任者。为了缓解这个问题，每个节点不仅要跟踪它的直接继任者，而且保持其s个直接继任者，如果灾难发生时，允许其跳过s–1个连续故障节点，并重新连接到环上。

自从DHT被发明出来后，开展了大量针对它的研究。为了让你明白到底这方面的研究有多少，让我们提出一个问题：什么是有史以来引用最多的网络文件？你会发现很难拿出一篇文章被引用的次数超过Chord文章被引用次数的一半（Stoica等，2001）。尽管这方面的研究多得名副其实像座山，但DHT应用才慢慢开始出现。一些BT客户端使用DHT来提供一个完全分布式跟踪器，就是我们所描述的那种。大型商业云服务也纳入了DHT技术，比如亚马逊的Dynamo（DeCandia等，2007）。

7.6 本 章 总 结

ARPANET上的域名始于一种非常简单的方式：一个ASCII文本文件列出了所有主机的名字和它们相应的IP地址。每天晚上，所有机器都下载该文件。但是当ARPANET演变

成 Internet 并且规模发生了爆炸后，需要一个更为复杂和动态的命名方案。现在使用的是一个称为域名系统的分层计划。它将 Internet 上的机器组织成一组树。在顶层，是众所周知的通用域名，包括 com、edu 以及约 200 个国家或地区的顶级域名。DNS 作为一个分布式数据库，其服务器遍布在世界各地。通过查询 DNS 服务器，一个进程可以将一个 Internet 域名映射成 IP 地址，该地址被用来与该域的计算机通信。

电子邮件是 Internet 的原始撒手锏应用。它仍然被广泛地应用，从小孩子到老爷爷，现在每个人都在使用它。世界上绝大多数电子邮件系统都使用由 RFC 5321 和 RFC 5322 定义的协议。邮件都有简单的 ASCII 头，而且通过 MIME 可以发送许多类型的内容。邮件被提交给邮件传输代理，由它作进一步的传递；通过不同的用户代理，用户从邮件传输代理处获取并检索邮件，这些用户代理包括 Web 应用。提交的邮件使用 SMTP 发送，邮件传输的工作通过建立的一条 TCP 连接进行，即要建立一条从发送端邮件传输代理到接收端邮件传输代理的 TCP 连接。

Web 是被大多数人认为等同于 Internet 的应用程序。最初，它只是一个系统，用来无缝连接跨计算机的超文本页面（以 HTML 语言编写）。浏览器与服务器建立一个 TCP 连接并使用 HTTP 下载网页。如今，许多 Web 上的内容是动态生成的，无论是在服务器端（例如，PHP），或在浏览器中（例如，使用 JavaScript）。当与后端数据库相结合时，动态服务器页面催生出这样的 Web 应用程序，比如电子商务和搜索等。动态浏览器页面正在演变为全功能的应用，比如电子邮件，这些全功能应用运行在浏览器内，使用 Web 协议与远程服务器通信。

缓存和持续连接是被广泛用来提高 Web 性能的两种技术。尽管移动电话的带宽和处理能力都有了很大的增长，但在移动设备上使用 Web 独具挑战性。Web 站点通常给带有小屏幕的设备发送页面的裁剪版本，这样的页面具有较小图像和较少复杂的导航。

Web 协议越来越多地被用于机器对机器通信。作为对内容的描述，XML 优于 HTML，因为它更易于机器的处理。SOAP 是一个使用 HTTP 来发送 XML 消息的 RPC 机制。

自 2000 年以来，数字音频和视频已经成为 Internet 的主要驱动力。今天大多数 Internet 流量是视频，其中包含很多通过混合协议（包括 RTP/UDP 和 RTP/HTTP/TCP）来自 Web 站点的媒体流。给很多消费者现场直播媒体流，包括广播所有事件的 Internet 电台和电视台。音频和视频还可用于实时会议。许多呼叫使用了 IP 语音，而不是传统的电话网络，包括视频会议。

盛极一时的网站数量只有一小部分，而数量巨大的网站又没那么流行。为了满足流行的网站的需求，已经部署了内容分发网络。CDN 使用 DNS 把客户指向一个附近的服务器；服务器放置在世界各地的数据中心。另外，P2P 网络让一组机器彼此共享内容，比如电影。它们提供的内容分发能力可随着 P2P 网络的机器数量的不断增大而获得提升，而且可与最大的网站相媲美。

习 题

1. 许多企业的计算机有3个不同的全球唯一标识符。试问它们分别是什么？
2. 在图7-4中，在laserjet后没有终结。试问这是为什么？
3. 考虑这样一种情况，一个网络恐怖主义分子同时破坏了全世界的所有DNS服务器。试问这将如何改变一个人使用Internet的能力？
4. DNS使用UDP而没有采用TCP。如果DNS数据包丢失，并且没有自动恢复机制。试问这会产生问题吗？如果会，它是如何解决呢？
5. John希望有一个原始的域名，并使用一个随机程序来产生他的二级域名。他要在com通用域名下注册这个域名。生成的域名253个字符长。试问com注册处允许注册这个域名吗？
6. 试问只有一个DNS域名的机器可以有多个IP地址吗？这种情况是如何发生的？
7. 近年来，拥有Web站点的公司数量爆炸式增长。结果是，成千上万的公司注册在com域下，从而导致该域的顶级服务器负载极其繁重。请提出一种建议方案来缓解这个问题，同时又不用修改域名方案（也就是说，不引入新的顶级域名）。你的方案可以要求修改客户代码。
8. 有些电子邮件系统支持Content Return:头字段。它指定了当消息未被递交时是否返回邮件体。试问这个字段属于信封还是邮件头？
9. 电子邮件系统需要相应的目录，以便于查找人们的电子邮件地址。为了建立这样的目录，名字应该被分割成标准的组成部分（比如，第一个名字，最后一个名字）才有可能搜索。请讨论为了建立一个可接受的全球标准需要解决哪些问题。
10. 一个大型律师事务所有许多员工，每一位员工有一个电子邮件地址。每个员工的电子邮件地址是<login>@ lawfirm.com。然而，该事务所并没有明确定义登录格式。因此，有些员工使用他们的第一个名字作为自己的登录名，有些员工使用他们最后一个名字，还有些员工使用了名字的缩写，现在该事务所希望制定一个固定的格式，例如：
    firstname.lastname@lawfirm.com
    试问这个格式能用作全体员工的电子邮件地址吗？如何做才不至于造成太多混乱？
11. 一个二进制文件长4560个字节。如果使用基于64的编码技术，而且在每发出110字节以及结尾处插入CR+LF。试问该文件将有多长？
12. 本书没有列出5个MIME类型。请检查你的浏览器或Internet来获取这方面的信息。
13. 假设你想给一个朋友发送一个MP3文件，但是你朋友的ISP限制入境邮件的大小为1 MB，而该MP3文件是4 MB。试问是否有办法通过使用RFC 5322和MIME来处理这种情形？
14. 假设约翰刚刚为他的工作电子邮件地址设置了自动转发机制，该转发机制接收他所有与业务相关的电子邮件，然后转发到他的私人电子邮件地址，他与他的妻子共享该私人邮箱。约翰的妻子不知道这一点，而且在他们的私人账户上触发了一个度假代理。因为约翰转发了他的电子邮件，因而他在他的工作机器上并没有设立度假守护进程。

试问，当约翰的工作电子邮件地址收到一封电子邮件时，会发生什么？

15. 任何一个标准都需要精确的语法，例如RFC 5322，这样不同的实现才能相互协同工作。即使最简单的表项也必须小心定义。SMTP头允许在token之间出现空白。请针对标记符之间的空白，给出两种似是而非的定义。
16. 度假代理是用户代理的一部分还是邮件传输代理的一部分？当然，它是通过用户代理设置的，它是用户代理实际发送的答复吗？请解释你的答案。
17. Chord算法的一个简单版本被用在对等查询中。搜索不使用指取表，相反，它们被线性地绕成一个环，环可取任何一个方向。试问，一个节点能准确地预测它应该在哪个方向搜索吗？讨论你的答案。
18. IMAP允许用户从远端邮箱获取和下载电子邮件。试问，这是否意味着内部邮箱格式必须标准化，因此客户端上的任何IMAP程序可以读取任何邮件服务器上的邮箱？讨论你的答案。
19. 考虑如图7-71所示的Chord环。假设节点18突然上线。试问图中的哪个指取表受到影响？如何受到影响的？
20. Webmail使用了POP3，还是IMAP，或者什么都没有用?如果使用了其中之一，试问为什么会选择这个？如果两者都没用，实质上更接近哪个？
21. 当发送Web页面时，试问，它们会附加上MIME头吗？为什么？
22. 当用户使用Firefox点击一个链接时，一个特定的辅助应用程序就被启动；但在IE浏览器中点击相同的链接时，导致一个完全不同的辅助应用程序被启动，即使在这两种情况下返回的MIME类型是相同的。试问这种情况是否有可能发生？解释你的答案。
23. 虽然在课文中没有提到URL的另一种形式是使用IP地址，而不是它的DNS域名。请用此信息来解释为什么一个DNS域名不能以数字结尾。
24. 试想一下，在斯坦福大学数学系有人刚刚写了一个包括证明过程的新文档，他希望通过FTP将该文档分发给他的同事们审阅。他把程序放在FTP目录ftp/pub/forReview/newProof.pdf。试问这一程序可能的URL是什么？
25. 在图7-22中，www.aportal.com用一个Cookie跟踪记录用户的偏好。这项方案的缺点是Cookie被限制为4 KB，因此如果要扩展偏好，例如，要增加关于许多股票、运动队、新闻报道的类型、多个城市的天气、在众多的产品类别特价商品和更多信息，可能会超出4 KB的限制。请设计另一种方案来保持跟踪用户的偏好但又没有这个问题。
26. 懒银行要为它的客户操作网络银行更加容易，因此，在客户签署并通过密码验证后，银行返回一个Cookie，其中包含客户ID号。这样，客户在将来访问网上银行时可以不必识别自己，或者输入密码。试问，你认为这种想法怎么样？可行吗？它是一个好主意吗？
27. （a）考虑如下的HTML标记：

```
<h1 title="this is the header"> HEADER 1 </h1>
```

在什么情况下，浏览器使用TITLE属性，如何使用的？

（b）TITLE属性与ALT属性有什么不同？

28. 你如何在HTML中生成一个可点击的图像？举例说明。

29. 编写一个 HTML 页面。该页面包括一个邮件地址 username@DomainName.com 的链接。试问，当用户点击该链接时会发生什么？
30. 编写一个大学注册器。能列出多个学生，每个学生有一个姓名、地址和一个 GPA。
31. 对于下列应用程序，告诉它是否（1）有可能（2）最好使用 PHP 脚本或 JavaScript 中的哪一个，以及为什么：
    （a）显示从 1752 年 9 月以来的任何一个月的日历。
    （b）显示从阿姆斯特丹到纽约的航班时间表。
    （c）画出一个用户提供系数的多项式。
32. 写一个 JavaScript 程序，它接受一个大于 2 的整数，并告知它是否为一个素数。请注意，JavaScript 有 if 和 while 语句，这些语句的语法与 C 和 Java 相同。模运算符为%。如果你需要 x 的平方根，使用 Math.sqrt（x）。
33. 一个 HTML 页面如下所示：

```
<html> <body>
<a href="www.info-source.com/welcome.html"> Click here for info </a>
</body> </html>
```

    如果用户单击该超链，就打开了一个 TCP 连接，并且给服务器发送一系列的行。请列出发出的所有行。
34. If-Modified-Since 头可以用来检查一个缓存页面是否仍然有效。请求的页面可以包含图像、声音、视频等，以及 HTML。你认为这种技术用于 JPEG 图像相比 HTML 的效率更好还是更糟？仔细想想“效率”意味着什么，并解释你的答案。
35. 在当天的一个重大体育赛事中，比如一些流行运动的冠军赛，很多人都去访问官方网站。这个闪群与 2000 年佛罗里达州总统选举发生的闪群具有相同的意义吗？为什么是相同的？或者为什么不同呢？
36. 把单个 ISP 建设成 CDN，是否可行？如果可行，它将如何工作？如果不可行，这个想法错在哪里？
37. 假设不对语音 CD 进行压缩。为了播放两个小时的音乐，光盘必须包括多少 MB 的数据？
38. 在图 7-42(c)中，使用 4 位样值来表达 9 个信号值时会引入量化噪声。第一个样值，在 0 点是精确的，但后续几个就不准确了。对于在 1/32、2/32 以及 3/32 处的样值误差百分比分别是多少？
39. 一个伪随机模型能用来减少 Internet 电话所需的带宽吗？如果可以，必须满足什么条件（如果有的话）才能工作？如果不能，为什么？
40. 一个音频流服务器到一个媒体播放器的单向“距离”为 100 毫秒，它以 1 Mbps 速率输出。如果媒体播放器有一个 2 MB 的缓冲区，试问你能说出低水印标记和高水印标记的位置在哪里？
41. 试问 IP 语音遇到防火墙和流式音频遇到防火墙的问题相同吗？讨论你的答案。
42. 试问以每秒 50 帧、每个像素 16 位，传输未经压缩的 1200×800 像素的彩色帧所需要的比特率是多少？

43. 试问 MPEG 帧中 1 位的错误会影响该出错帧以外的其他帧吗？请解释你的答案。
44. 考虑一个有 50 000 个客户的视频服务器，每个客户每个月看三部电影。三分之二的电影是在晚上 9 时需求。服务器在这段时间内一次传输多少部电影？如果每部电影需要 6 Mbps，服务器需要多少个 OC-12 网络连接？
45. 假设齐普夫定律适用于对一个有 10 000 电影的视频服务器的访问。如果服务器在内存中持有 1000 个最流行的电影，在磁盘上保持剩余的 9000 个电影，请给出所有引用都在内存中的比例表达式。写一个小程序，以便从数值上来评估该表达式。
46. 某些域名抢注行为已经注册了一些域名，这些域名是公共企业网站的拼写错误，例如，www.microsfot.com。请列出至少 5 个这样的域名。
47. 许多人注册了这样的 DNS 域名 www.word.com，其中 word 是一种常见字。对于以下每个类别，列出 5 个此类网站，并简要地总结它（例如，www.stomach.com 属于一个长岛的肠胃病）。下面是类别列表：动物、食品、家用物品和身体部位。对于最后一类，请坚持使用腰部以上的身体部位名称。
48. 重写图 6-6 的服务器，使它成为一个真实的使用 HTTP1.1 GET 命令可以访问的 Web 服务器。它也应该接受主机消息。服务器应该保持一份最近从磁盘读取的文件缓存，并且可能时用缓存来服务客户的请求。

第8章 网络安全

在最初几十年中，计算机网络主要被大学的研究人员用于发送电子邮件，以及被公司的员工用于共享打印机。在这样的条件下，安全问题并没有得到足够的关注。但现在，当成千上百万的普通市民利用网络来处理银行事务、网上购物和填报纳税时，出现了越来越多的安全问题，网络安全已成为一个巨大的问题。在本章中，我们将从几个角度来研究网络的安全问题，并指出各种安全陷阱，以及讨论许多能使网络更加安全的算法和协议。

安全性是一个范围很广阔的话题，同时也涉及大量的违法犯罪活动。在最简单的形式中，它关心的是如何保证好事者们不能读取或者（更糟的是）悄悄地修改给其他人的消息。它也涉及有人企图访问未经授权的远程服务。它还要处理这样的情形：如何判断一条自称来自 IRS（美国国税局）的“星期五之前付款”的消息确实来自 IRS，而并非来自黑手党（Mafia）。安全性也涉及合法消息被捕捉及重放的问题，以及人们企图否认自己曾经收到过某些特定的消息。

大多数安全问题都是由于某些恶意的人企图获得某种收益、引起别人关注，或者伤害他人而有意制造的。图 8-1 列出了一些最为常见的攻击者及目的。从这个列表中应该很清楚地看出，要想使一个网络变得更加安全，需要做的事情很多，而不仅仅只是改正编程错误就足够。这通常涉及与一些非常聪明、专一甚至还有足够资助的攻击者进行较量。同时也应该很清楚地看到，一些用来阻止随意攻击者的手段对于那些专业攻击者来说并没有太大的效果。警察的记录显示，多数的破坏性攻击并非是外人窃听电话线而是来自于内部积累的恩怨造成。安全系统在设计上应该考虑到这种情况。

攻击者	目的
学生	乐于窥探他人电子邮件
黑客	测试某人的安全系统；盗取数据
推销员	声称能代表整个欧洲，而不只是安道尔
公司	发现竞争者的策略性市场计划
离职员工	因被解雇而实施报复
会计	挪用公司公款
股票经纪人	拒绝通过E-mail向顾客做过的承诺
身份盗取	出售窃取信用卡号码
政府	了解敌人的军事或工业机密
恐怖分子	窃取生物战秘密

图 8-1 最常见的攻击者及目的

网络安全问题可以被粗略地分成 4 个相互交织的领域：保密、认证、不可否认和完整性控制。**保密**（secrecy）也称为**机密**（confidentiality），它的任务是确保信息不会被未经授权的用户访问。这也正是当人们考虑网络安全时首先想到的内容。**认证**是指当你在展示敏感信息或者进入商务交易之前你必须要确定自己在跟谁通话。**不可否认性**涉及签名：你的

顾客下了一份要购买1000万个左手器具的订单，他事后声称每个器具的单价为69美分，你如何证明他原来定购的价格是89美分呢？或者他事后根本不承认自己曾经下过订单。最后，**完整性控制**是用来确定你收到的信息是真实的而不是被恶意攻击者在传输途中篡改过的伪造信息。

所有这些问题（保密、鉴别、不可否认和完整性控制）同样存在于传统的系统中，但是有一些重大的区别。在传统系统中，利用挂号信和文档上锁的做法可以获得完整性和保密性。现在，抢劫邮政列车比过去Jesse James时代（美国内战期间铁路巨头们为了在中西部铺设铁路而与当地人民发生了激烈的冲突）要困难得多。

而且，人们通常能够辨别出原始的纸质文档和复印件之间的区别，这点对人们来说非常重要。你不妨做一个试验，把手头的一张有效支票复制一份，星期一将支票原件拿到银行去兑现，然后星期二用复印件再试着兑现一次。你能注意到因为这种区别所以银行很容易辨别它。但是若使用电子支票，因为原始支票和复制的支票是一样，银行可能需要花相当一段时间来知道如何识别真伪。

人们通过辨别相貌、声音和笔迹来识别其他人，通过在信笺纸上签字或者盖图章等手段作为签名的证据。篡改的行为通常可以被笔迹、墨水和纸张专家检测出来。但是所有这些手段在电子方式下统统无效，因此我们需要解决问题的其他方法。

在得到各种答案之前，首先值得花点时间来考虑协议栈中网络安全应该归属何方。可能无法把它单独归结为任何单个方面。协议栈中的每一层都对网络安全做出贡献。在物理层中，可以通过将传输线封装在内含高压气体的密封管线中等措施来有效对付那些搭线窃听的攻击手段。若有人企图在管线上钻孔，气体就会泄漏，从而降低导线管中的气压，并触发报警。有些军用系统就使用这种技术。

在数据链路层，通过点对点线路上传输的数据包可以在离开机器前被加密处理，当它们进入另一台机器时再被解密。所有的细节可以都在数据链路层内部处理，上面的网络层对于数据链路层发生的事情一无所知。然而，当数据包必须要穿越多台路由器的时候，这种方法就带来了问题。因为在每台路由器上数据包必须被首先解密，然后才能做进一步的处理，这种处理方式使得数据包容易遭受来自路由器内部的攻击。而且，它也不允许只保护某些会话（比如那些涉及用信用卡在线购物的会话）而不保护其他会话。不过，这种称为**链路加密**（link encryption）的方法正如其名所示，它可以很容易地被应用到任何一个网络中，所以它往往还是有用的。

在网络层，可以通过安装防火墙等措施来过滤好的和坏的数据包，让好的数据包正常通过而阻止有问题的数据包进入。IP安全（IPSec）就运行在这一层上。

在传输层，可以对端到端的整个连接进行加密，即从进程到进程的加密。为了达到最高级别的安全，端到端的安全是必需的。

最后，诸如像用户认证和不可否认性这样的问题只能在应用层上得以解决。

由于安全问题并不完全与某一层吻合，所以它也就不适合在本书前面任何一章中讨论。基于这个原因，我们用专门一章来讨论安全问题。

虽然本章篇幅比较长，技术性很强，而且也比较基础，但是，它相对来说比较独立。现有资料已完整地记载多数银行发生的安全事件。例如，松懈的安全程序、不合格的雇员以及无数的程序错误使系统易于遭受未经授权人的远程破解和所谓的社会工程师的攻击，

或诱使客户透露他们账号的详细信息。所有这些安全问题往往并不是因为聪明的罪犯搭接和窃听电话线，然后解密听到的消息而造成的。如果一个人拿着一张在马路上捡到的ATM卡，随便走进一家银行的分支结构并声称自己忘了PIN码而当场获得新的PIN码（凭着很好的客户关系）的话，那么世界上所有的密码系统都将无法防止这种数据窃用。从这个角度而言，Ross Anderson的书是一本真正让人大开眼界的好书，它记录了在各个行业中曾经发生的数百起安全失败的例子，几乎所有这些例子都是由于商务管理松散或者对安全性掉以轻心造成的（这是客气的说法）（Anderson，2001）。尽管如此，在考虑了所有的那些因素后，构建电子商务的技术基础就是加密法。

除了物理层安全以外，几乎所有的网络安全都基于密码学原理。出于这样的原因，我们将通过比较详细地研讨密码学来开始有关网络安全的学习。在8.1节，我们将介绍一些基本的原理；在8.2～8.5节中，我们将讨论一些密码学的基础算法和数据结构。然后，我们将详细地讨论如何使用这些概念来实现网络中的安全性。最后我们简短地给出有关技术和社会的进一步思考作为小结。

在我们开始之前，有必要提及哪些内容本章没有涉及。我们尽量将焦点集中在与网络有关的问题上，而不是与操作系统和应用系统有关的问题上，当然有时它们的界限难以划分。例如，本章不讨论以下内容：基于生物统计学的用户认证、口令安全、缓冲区溢出攻击、特洛伊木马、登录欺骗、诸如跨点脚本的代码注入、病毒、蠕虫和类似这些内容。所有这些问题都已经在“Modern Operating Systgems”（Tanenbaum，2007）一书的第9章中介绍。对系统安全方面有兴趣的读者可以参考那本书。现在让我们开始网络安全的学习之旅。

8.1 密 码 学

密码学（cryptography）一词来自于希腊语中的短语“secret writing”（秘密书写）。它有着辉煌而又悠久的几千年历史。在这一节中我们只是勾画出其中一些亮点，然后介绍一些后面学习所需的背景知识。有关完整的密码学历史，你可以阅读（Kahn，1995）。要想全面地了解安全和密码算法的最新现状、协议和应用，以及有关的材料，请参考（Kaufman等，2002）。想了解更数学化的方法，请参考（Stinson，2002）；对于那些不那么数学化的方法，可以参考（Burnett 和 Paine，2001）。

专业人员对密码和编码作了明确的区分。**密码**（cipher）是指逐个字符或者逐个位地进行变换，它不涉及信息的语言结构。相反，**编码**（code）则是指用一个词或符号来代替另一个词。尽管编码学有着非常光荣的历史，但现在已经很少被使用了。在历史上设计最为成功的编码系统是二次世界大战中美国军队在太平洋战场上使用的编码。他们的做法很简单，让纳瓦霍印第安人用专门的纳瓦霍语言来交流军事术语，例如，用chay dagahi nail tsaidi（字面意思是“慢行者之克星”）来表示反坦克武器。纳瓦霍语言是一门以音调为主的语言，非常复杂，而且没有书面的形式。日本方面根本没有一个人懂这门语言。

在1945年的9月，圣地亚哥联合会如此描述美国的编码系统：“三年来，无论什么时候海军登陆时，日本人总是听到一连串奇怪的汩汩声，同时夹杂着像是和尚念经的声音和

热水瓶倒空时发出的声音。”日本人始终未能破解这套编码系统，纳瓦霍编码的许多配音者由于突出的表现和勇敢精神而获得了很高的军事荣誉。美国破解了日本的编码，而日本却未能破解纳瓦霍编码，这个事实本身为美国在太平洋战争中赢得胜利起到了至关重要的作用。

8.1.1 密码学概论

历史上，有4类人用到了密码学，并且为之作出了贡献，他们是军事人员、外交人员、写日记者和情侣。在这4组人中，军事人员扮演了重要的角色，而且几个世纪以来他们不断完善着这个领域。在军事组织内部，需要加密的消息通常被交给收入低微的低级译码员来加密和传输。庞大的消息数量不能仰仗少数精英专家来完成。

在计算机出现以前，密码学的一个主要限制在于译码员执行所需变换的能力，尤其在战场上往往缺乏相应的装备。另一个限制是很难从一种密码方法切换到另一种密码方法，因为这需要重新训练一大批人。然而，由于译码员有可能被敌人俘虏，所以，在必要的时候能及时地更换密码方法变得非常关键。这些相互矛盾的需求导致产生了如图8-2所示的模型。

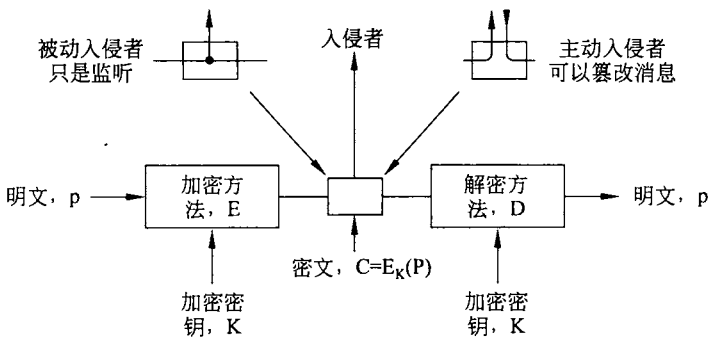


图8-2 加密模型（对称密钥密码）

待加密的消息称为**明文**（plaintext），它经过一个以**密钥**（key）为参数的函数变换，这个过程的结果就是所谓的**密文**（ciphertext），然后通常由无线电或者通信员传送出去。我们假设敌人或者**入侵者**（intruder）可以听到和精确地复制整个密文。然而，与目标接收者不同的是，他不知道解密所用的密钥是什么，所以他们无法很轻易地对密文进行解密。有时候入侵者不仅可以监听通信信道（被动入侵者），而且可以将消息记录下来并且在以后某个时候回放出来，或者插入他自己的消息，或者在合法消息到达接收方之前篡改消息内容（主动入侵者）。破解密码的技术就是所谓的**密码分析学**（cryptanalysis），而它和设计密码的**密码编码学**（cryptography）合起来统称为**密码学**（cryptology）。

用一种合适的表示法将明文、密文和密钥的关系体现出来，往往会非常有用。我们将使用 $C = E_K(P)$表示用密钥K加密明文P得到密文C。类似地，$P=D_K(C)$表示解密密文C得到明文P。由此可以得到：

$$D_K(E_K(P)) = P$$

这种表示法说明E和D只是数学函数，事实上它们也确实是这样。唯一的诀窍是它们

都是带两个参数的函数，并且我们将其中一个参数（密钥）写成下标的形式而不写成实参的形式，从而将它与消息本身区别开来。

密码学的基本规则是假定密码分析者一定知道加密和解密所使用的方法。换句话说，密码分析者知道图 8-2 中加密方法 E 和解密方法 D 的所有细节。当老方法被泄漏或者认为它们已被泄漏后，就需要花费大量的精力来发明、测试和安装新的算法，因此将加密算法本身保持秘密的做法是不现实的。当一个算法已不再保密时而仍然认为它是保密的，这将会带来更大的危害。

这正是密钥的用武之处。密钥由一段相对比较短的字符串构成，它相当于从大量潜在的加密方法中选择了一种加密方法。一般的加密方法可能几年才会发生变化，与密钥可以根据需要频繁地被改变。因此，我们的基本模型是一个稳定的、广泛公开的通用方法，它用一个秘密的、易改变的密钥作为参数。“让密码分析者知道加解密算法，并且把所有的秘密信息全部放在密钥中”这种思想称为**Kerckhoff 原理**（Kerckhoff's principle），这是用佛兰德军事密码学家 Auguste Kerckhoff 的名字来命名的，他在 1883 年第一次提出了这种想法（Kerckhoff，1883）。因此，我们有

Kerckhoff 原则：*所有的算法必须是公开的而密钥是保密的。*

算法的不保密性怎么强调都不过分。企图使算法保持秘密的做法称为**安全模糊**，这种做法永远不会奏效。而且，将算法公开，让密码设计者可以自由地与大量学院派的密码学家进行交流探讨，这些密码学家总是期望能破解密码系统，从而可以发表论文，来证明自己有多聪明。如果一个密码算法被公开 5 年以后，尽管在此期间许多专家试图破解该算法，但是无人能够成功，那么，这个算法应该是非常可靠的。

由于真正的秘密在密钥中，所以它的长度是一个非常重要的设计要素。请考虑一个简单的号码锁，它通常的原理是按照顺序输入数字，这点大家都知道，但是锁的号码（即密钥）却是保密的。如果密钥长度是两个数字，则意味着共有 100 种可能。若密钥长度为三个数字，则意味着共有 1000 种可能；同样地，六位数字的密钥意味着 100 万种可能。密钥越长，则密码分析者要消耗的**工作因子**（work factor）就越高。通过穷举搜索整个密钥空间来破解密码系统的工作因子随着密钥长度增加而呈指数递增。保密性来自于强而牢固的（但是公开的）算法和密钥的长度。为了防止你的孩子阅读你的电子邮件，64 位密钥足够了。对于常规的商业用途，至少应该使用 128 位密钥。对于大规模的政府机构，则至少需要 256 位的密钥，而且需要的话应该越长越好。

从密码分析者的角度来看，密码分析问题有 3 个主要的变种。当他得到了一定量的密文，但是没有对应的明文时，他面对的是**唯密文**（ciphertext only）问题。报纸上猜谜栏目中的密码难题就属于这一类问题。当密码分析者有了一些相匹配的密文和明文时，密码分析问题称为**已知明文**（known plaintext）问题。最后，当密码分析者能够加密某一些他自己选择的明文时，问题就变成了**选择明文**（chosen plaintext）。如果可以问密码分析者诸如“ABCDEFGHIJKL 经过加密之后是什么？”之类的问题，那么破解报纸上的那种密码问题就容易了。

密码编码学领域中的新手通常作这样的假设：如果一个密码能够承受密文攻击，那么它就是安全的。这个假设非常幼稚。在许多情况下，密码分析者可以正确地猜测出明文的某些部分。例如，当你向其他机器发出请求时，许多计算机做的第一件事情是回复“login:”。

一旦有了某些相匹配“明文-密文”后，密码分析的工作就容易得多。为了保证安全起见，密码设计者应该保守一点，并且确信即使对手能够加密任意数量的选择明文，密码系统也不会被攻破。

在历史上，加密方法被分成两大类：置换密码和替代密码。现在我们简要地介绍这两种密码，正好作为现代密码学的背景信息。

8.1.2 置换密码

在**置换密码**（substitution cipher）中，每个字母或者每一组字母被另一个字母或另一组字母取代，从而将原来的字母掩盖起来。最古老的密码之一是**凯撒密码**（Caesar cipher），它因 Julius Caesar 而得名。在这种方法中，a 变成 D，b 变成 E，c 变成 F，…，z 变成 C。例如，attack 变成 DWWDFN。在例子中，明文以小写字母给出，密文则使用大写字母。

凯撒密码的一种略微一般化方案是，允许明文按字母顺序被移动 k 个字母，而并不总是移动 3 个位置。在这种情况下，k 变成了这种循环移动字母表的通用加密方法的一个密钥。凯撒密码也许确实欺骗了庞培（Pompey），但自那以后再也没有骗过别人。

接下来的改进是，让明文中的每个符号（为了简化起见，这里假设为 26 个字母）映射到其他某一个字母上，例如，

明文：a b c d e f g h i j k l m n o p q r s t u v w x y z

密文：Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

采用这种“符号对符号”置换策略的通用系统称为**单字母置换**（monoalphabetic substitution）密码，其密钥是对应于整个字母表的 26 字母串。对于上面的密钥，明文 attack 被变换为密文 QZZQEA。

初看起来，这似乎是一个非常安全的系统，因为虽然密码分析者了解通用的系统（即字母对字母的置换），但是它不知道到底使用了哪一个密钥，而密钥的可能性共有 $26! \approx 4 \times 10^{26}$ 种。与恺撒密码不同的是，要试遍所有这么多种可能的密钥并不是一种可行的做法。即使得到一种方法只需要花费 1 纳秒的时间，但用百万次计算机芯片进行并行计算也需要花费一万年的时间才能试遍所有可能的密钥。

不过，事实上只要知道相对少量的密文，密码就很容易被破解。基本的攻击手段利用了自然语言的统计特性。例如，在英语中，e 是最常见的字母，其次是 t、o、a、n、i 等。最常见的**两字母组合**（或者两字母连字）是 th、in、er 和 an。最常见的**三字母组合**（或者三字母连字）是 the、ing、and 和 ion。

密码分析者为了破解单字母密码，他首先计算所有字母在密文中出现的相对频率，然后他也许会试探将最常见的字母分配给 e，次常见的字母分配给 t。接下来他将查看三字母连字，找到比较常见的形如 tXe 的三字母组合，这强烈地暗示着其中的 X 是 h。类似地，如果模式 thYt 出现得很频繁的话，则 Y 可能代表了 a。有了这些信息以后，他可以查找频繁出现的形如 aZW 的三字母组合，它很有可能是 and。通过猜测常见的字母、两字母连字和三字母连字，并且利用元音和辅音的各种可能组合，密码分析者就可以逐个字母地构造出试探性的明文。

另一种做法是猜测一个可能的单词或者短语，例如，考虑以下这段来自于一家会计事

务所的密文（分成5个字符为一组）：

CTBMN	BYCTC	BTJDS	QXBNS	GSTJC	BTSWX	CTQTZ	CQVUJ
QJSGS	TJQZZ	MNQJS	VLNSX	VSZJU	JDSTS	JQUUS	JUBXJ
DSKSU	JSNTK	BGAQJ	ZBGYQ	TLCTZ	BNYBN	QJSW	

在会计事务所的消息中，一个可能的单词是financial（财经）。我们知道在financial这个单词中有一个重复的字母（i），并且这两个i之间有4个其他的字母，根据这样的知识，我们在密文中查找相隔4个位置的重复字母。我们可以找到12个地方，分别在6、15、27、31、42、48、56、66、70、71、76和82位置上。然而，只有其中两个地方，即31和42，它的下一个字母（对应于明文中的n）重复出现在正确的位置上。而在这两者之中，只有31有正确的a位置（考虑在financial中有两个a），所以，我们知道financial从位置30开始。以此为出发点，利用英语文本的频率统计规律可以很容易地推断出密钥，从而能找出最终的全部单词。

8.1.3 替代密码

置换密码保留了明文符号的顺序，但是将明文伪装起来了。与此相反，**替代密码**（transposition cipher）重新对字母进行排序，但是并不伪装明文。图8-3给出了一个常见的替代密码：列换位。该方案用一个不包含任何重复字母的单词或者短语作为密钥。在这个例子中，密钥是MEGABUCK。密钥的用途是对列顺序进行排列，第1列是密钥字母中最靠近字母表起始位置的那个字母下面那列，以此类推。明文按水平方向的行来书写，如果有必要的话填满整个矩阵。密文则被按列读出，从密钥字母最小的那列开始。

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

明文：
pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

密文：
AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRCXB

图8-3 替代密码

为了破解替代密码，密码分析者首先要明白，自己是在破解一个替代密码。通过查看E、T、A、O、I、N等字母的频率，很容易就可以看出它们是否吻合明文的常规模式。如果是的话，则很显然这是一种替代密码，因为在这样的密码中，每个字母代表的是自己，从而保持了字母出现的频率分布。

接下来要猜测共有多少列。在许多情况下，从环境信息中或许可以猜到一个可能的单词或者短语。例如，假定我们的密码分析者怀疑消息中的某个地方出现了明文短语milliondollars。他观察到在密文中出现的两字母组合MO、IL、LL、LA、IR和OS是因为这个短语回绕的结果。密文字母O跟在密文字母M的后面（即在第4列的垂直方向上它们是相邻的），这可能是因为它们在短语中被一段等于密钥长度的距离所隔开。如果密钥长度为7的话，则两字母组合MD、IO、LL、LL、IA、OR和NS就会出现。实际上，对于

每一个密钥长度，在密文中都会出现一组不相同的两字母组合。通过检查不同的可能性，密码分析者往往很容易就能够确定密钥的长度。

最后的步骤是给列顺序。当列数比较小，假设有k列时，依次检查k（k-1）个列队中的每一个它的出现频率是否和英文明文中相同两字母组出现的频率相同。假定频率最接近的那一对的放置位置是正确的，我们可以尝试让剩下的每一列跟在这一对的后面，两字母组合和三字母组出现频率最匹配的那列假设是正确的。用相同的方法继续找到下一列。整个过程继续下去，直至找出潜在的列顺序。到这时候，明文可以确定是否破解成功了（比如，如果出现milloin的话，很明显就知道错误在哪里了）。

有些替代密码接受一个固定长度的块作为输入，并产生一个固定长度的块作为输出。通过给出一个指明字符输出顺序的列表就可以完整地描述这样的密码。例如，图8-3中的密码可以被看作一个64字符块的密码。它的输出是4、12、20、28、36、44、52、60、5、13，…，62。换句话说，第4个输入字符a首先被输出，然后是第12个字符f，以此类推。

8.1.4 一次性密钥

要想构建一个不可被攻破的密码其实是非常容易的，相应的技术在几十年前就已经知道。首先选择一个随机位串作为密钥，然后将明文转变成一个位串（比如使用明文的ASCII表示法）。最后，逐位计算这两个串的异或（XOR）值，这样得到的密文是不可能被破解的。这是因为在数量足够大的密文样本中，每个字母的出现频率是相等的，两字母组合，三字母组合等的出现频率也都是如此。这种方法称为**一次性密钥**（one-time pad），它可以抵挡现在和将来的攻击，无论入侵者的计算能力有多么强大都是如此。理由源于信息理论：因为任何给定长度的明文都有相同的可能性，所以消息中根本不存在任何可以用来解密的信息。

图8-4给出了一个一次性密钥用法的例子。首先，消息1“I love you.”被转换成7位ASCII码。然后选择一个一次性密钥Pad1，并且与消息1进行异或（XOR）操作得到密文。密码分析者可以试验所有可能的一次密钥，并检查每个密钥所对应的明文。例如，图中列出的一次性密钥Pad2可以被用来做试验，结果得到明文2“Elvis lives”，这个结果有点似是而非（关于这方面的讨论超出了本书的范围）。事实上，对于每一个长度为11个字符的ASCII明文，就有一个生成此明文的一次性密钥。这也正是我们所说的在密文中没有任何信息的原因：你总是可以得到任何一条正确长度的消息。

```
消息1：          1001001 0100000 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110
一次性密钥1：    1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011
密文：           0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101

一次性密钥2：    1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110
明文：           1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011
```

图8-4 利用一次性密钥进行加密，通过其他一次性密钥可从密文获得任何可能明文

在理论上一次性密钥的确很好，但在实际使用中却有若干缺点。首先，一次性密钥无法记忆，所以发送方和接收方必须随身携带书面的密钥副本。如果任何一方被敌人捕获，则显然书面的密钥是一个很大的威胁。除此之外，可被传送的数据总量受到可用密钥数量

的限制。如果一名间谍非常走运，发现了一批极有价值的数据，他可能由于密钥已被用尽而无法将这批数据传送回总部。另一个问题是，这种方法对于丢失字符或者插入字符非常敏感。如果发送方和接收方失去了同步，那么失去同步后面的所有数据都变成了垃圾。

随着计算机的出现，一次性密钥方法对于某些应用可能会变得具有实用价值。密钥源可以是一片特殊的DVD，它包含了几千兆字节的信息，如果它被放在DVD电影箱中运输，并且开头部分加上几分钟电影片段，那么这样的DVD甚至不会招人怀疑。当然，在千兆位速率的网络上，每隔30秒钟就必须插入一片新的DVD可能非常烦人。由于在发送消息之前，必须通过人将DVD从发送方带给接收方，这极大地降低了一次性密钥方法的实际使用性。

量子密码

有趣的是，针对如何在网络上传输一次性密钥问题，可能真的存在一种解决方案。这种方案来源于一种看来不太可能的源：量子机。这个领域现在仍然属于试验性质，但是初始的试验非常成功。如果它能做得完美并且很有效，那么，所有的密码系统都可以利用一次性密钥方法来完成，因为已经证明一次一密的方法是绝对安全的。下面我们简短地介绍这种**量子密码学**（quantum cryptography）的工作原理，尤其是介绍一个称为BB84的协议，该协议的名字来源于其作者的名字和发表年份（Bennet和Brassard，1984）。

假设有一个用户Alice希望与第二个用户Bob建立一个一次性密钥。这里Alice和Bob称为**主角**（principal），他们是我们故事中的主角。例如，Bob是一个银行家，Alice希望与他做一些商务交易。自从Ron Rivest在很多年前引入这两个名字后，几乎所有关于密码学的论文和书籍在叙述的时候都使用Alice和Bob作为虚拟主角。密码学家都喜欢传统，所以，如果我们使用Andy或者Barbara作为主角的话，就没有人会相信这一章的内容了。所以，我们还是沿用传统的做法。

如果Alice和Bob能够建立一个一次性密钥，那么，他们就可以用该密钥来安全地通信了。问题是：如果不像前面提到的那样交换DVD的话他们该如何建立一次性密钥呢？我们可以假定Alice和Bob在一根光纤的两端，通过这根光纤他们可以发送和接收光脉冲。然而，一个超级强大的名叫Trudy入侵者能够割断光纤并插入一个接头，这样Trudy可以读取两个方向上的所有数据，也可以往两个方向上发送假的消息。这种情形对于Alice和Bob来说，似乎已经没有希望进行安全通信了，但是，量子密码学却为他们带来了一线新的曙光。

量子密码学基于这样一个事实：光是以一种极小的称为**光子**（photon）的包的形式传递的，并且光子具有某些偏振特性。而且，光在通过一个偏振滤光镜时可以被调整到一个方向上，戴太阳镜的人或者摄影师都知道这个事。如果将一束光（即一个光子流）通过一个偏振滤光镜，则该光束中的所有光子都将被偏到滤光镜的轴向（比如垂直方向）上。如果现在光束再通过第二个偏振滤光镜，则从第二个滤光镜出来的光的强度将与两轴之间夹角的余弦平方成正比。如果这两个轴相互垂直的话，那么所有的光子都通不过。两个滤光镜的绝对方向并不重要，关键是它们之间的夹角。

为了产生一次性密钥，Alice需要两组偏振滤光镜，第一组偏振滤光镜由一个垂直滤光镜和水平滤光镜组成，这种选择称为**直线基**（rectilinear basis）。这里的一个基只是一个坐

标系统。第二组偏振滤光镜一样，但是旋转了 45°，所以，一个偏振滤光镜的方向是从左下至右上，另一个偏振滤光镜的方向是从左上至右下。这种选择称为**对角基**（diagonal basis）。因此，Alice 有两个基，她可以根据需要快速地将这些基插入到她的光束中。实现时，Alice 并没有 4 个独立的偏振滤光镜，而是一个晶体，该晶体的偏振方向可以通过电子方式以极快的速度切换到 4 个允许方向中的任何一个。Bob 也有一套与 Alice 相同的设备。Alice 和 Bob 两个人都有两组可用的基，这对于量子密码系统是非常关键的。

对于每一组基，Alice 现在将一个方向作为 0，另一个方向作为 1。在下面介绍的例子中，我们假设她选择垂直方向为 0，水平方向为 1。另外，她也选择从左下至右上方向为 0，从左上至右下方向为 1。她通过明文方式将这些选择发送给 Bob。

现在 Alice 选择一个一次性密钥，比如说她利用一个随机数发生器（这本身就是一个非常复杂的主题）来生成该密钥。然后她逐位地将密钥传送给 Bob，在传送每一位的时候她随机地选择其中一个基。为了发送每一位，她的光子枪发射出来的光子已经被正确地偏振到她为这一位所选择使用的基上。例如，她可能选择对角基、直线基、直线基、对角基、直线基等。为了用这些基来发送她的一次性密钥 1001110010100110，她将会发送图 8-5(a)中所示的光子。给定了一次性密钥和基的序列之后，每一位的偏振方向也被唯一地确定下来。像这样一次发送一个光子的数据位称为**量子位**（qubit）。

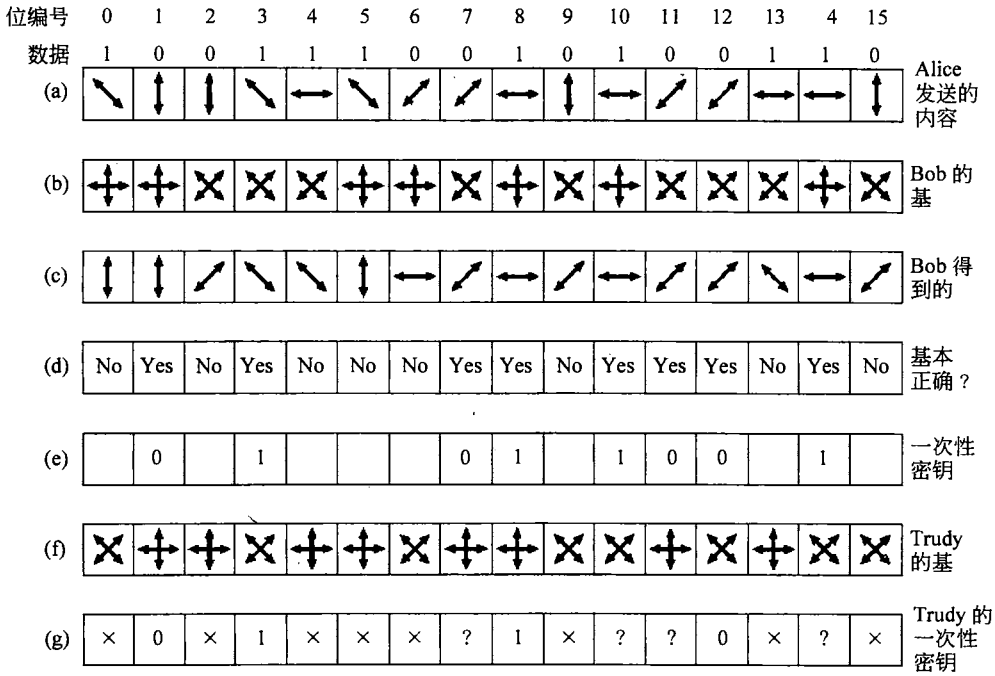


图 8-5 一个量子密码系统示例

Bob 并不知道 Alice 使用了哪些基，所以他随机地为每一个到来的光子选择一个基，并使用这个基，如图 8-5(b)所示。如果他选择了正确的基，则他会得到正确的数据位。如果选择的基不正确，则得到一个随机的位，因为如果一个光子被发射到一个与它自己的偏振方向成 45°角的滤光器上，那么它将会随机地跳到滤光镜的偏振方向上，或者跳到与滤光镜方向垂直的偏振方向上，这两者的概率是相等的。光子的这种性质正是量子机的基础。

因此，有些位是正确的，而有些位是随机的，但是Bob并不知道哪些是正确的、哪些是不正确的。Bob得到的结果如图8-5(c)所示。

那么，Bob如何知道他选择的哪些基是正确的、哪些基是不正确的呢？他简单地告诉Alice，他为明文中的每一位使用了哪个基；然后Alice告诉他，明文中哪些是正确的，哪些是不正确的，如图8-5(d)所示。利用这些信息，双方就可以根据正确的猜测结果获得一个位串，如图8-5(e)所示。平均来说，这个位串的长度将是原始位串的一半，但是由于双方都知道这个位串了，所以他们可以将该位串用作一次性密钥。Alice所要做的事情仅仅是传输一个略微超过所需长度之两倍的位串，于是她和Bob就有了一个所需长度的一次性密钥。问题解决了！

但请稍等。我们忘了还有Trudy呢。假定她对于Alice所说的话非常好奇，所以她割断光纤并插入了她自己的检测器和发射器。对她来说不幸的是她并不知道每个光子使用了哪一个基。她所能够做的就是像Bob一样，随机地为每个光子选择一个基。图8-5(f)显示了她所选择的一个例子。当Bob后来用明文向Alice报告他使用了哪些基，并且Alice也用明文告诉他哪些基是正确的时候，Trudy现在也知道她得到的哪些位是正确的，哪些位是错误的。在图8-5中，她知道以下位是正确的：0、1、2、3、4、6、8、12和13。但是，从图8-5(d)的Alice应答中，她知道只有1、3、7、8、10、11、12和14位才是一次性密钥的组成部分。她猜中了其中的4位（即1、3、8和12位），并且也捕获到了正确的位信息。其他的4位（7、10、11和14位），她猜测错误，所以她不知道真正传输的位是什么。因此，Bob从图8-5(e)中得知，一次性密钥的前8位为01011001，但是，Trudy仅仅得到了01?1??0?，如图8-5(g)所示。

当然，Alice和Bob知道Trudy可能已经捕获了他们的一次性密钥的一部分，所以他们希望进一步减少Trudy拥有的信息。他们只要对这个一次性密钥做一个变换就可以做到这一点。例如，他们可以将一次性密钥分成1024位的块，然后对每一块做平方操作，从而构成2048位数值，再将这些2048位数拼接起来当作一次性密钥。Trudy只有一部分位串信息，所以她无法产生位串的平方，因此她什么也得不到。从原始的一次性密钥变换到另一个削弱了Trudy所知信息，这个变换称为**保密增强**（privacy amplification）。实际上，往往采用这种输出位依赖于所有输入位的复杂变换方法，而不用简单的平方。

可怜的Trudy。她不仅对一次性密钥一无所知，而且她的存在也不再是一个秘密了。毕竟，她必须把接收到的每一位转发给Bob才能欺骗他，让他以为自己是在跟Alice通话。问题在于她Trudy能做的最好事情也只是用她接收用的偏振来传输她收到的每一量子位，而其中大约有一半是错误的，从而导致Bob的一次性密钥中有很多错误。

当Alice最后开始发送数据时，她使用一种重量级的前向纠错码对数据进行编码。从Bob的角度来看，一次性密钥中的1位错误与1位传输错误是相同的，他总是收到错误的位。如果有足够的前向纠错能力，那么不管任何错误他都能够恢复出原始的消息，同时他可以很容易地计算出有多少错误得到了纠正。如果这个数值大大超过了设备的期望错误率，他就能肯定Trudy已经搭接在线路上了，于是可以根据情况做出响应（比如告诉Alice切换到一个无线电信道上，或者打电话给警察局，等等）。如果Trudy有办法复制光子，从而她可以监视其中一个光子，并且将另一个同样的光子发送给Bob，那么，她就可以将自己隐藏起来，避免被Bob发现。但是，迄今为止人们尚未发现有理想的方法可以复制光子。不

过，即使有一天 Trudy 能够复制光子，量子密码学用于建立一次性密钥的价值也不会被减弱。

尽管研究人员已经可以在超过 60 千米距离的光纤上运行量子密码系统，但是装备非常复杂，而且极其昂贵。不过，量子密码学的思想仍然很有前途。有关量子密码学的更多信息，请参考（Mullins，2002）。

8.1.5 两个基本的密码学原则

在本章我们将学习许多不同的密码系统，首先要了解这些密码系统背后的两条基本原则，这对于理解后面的内容非常重要。注意，违反它们就是在冒险。

冗余度

第一条原则：所有被加密的消息必须包含一定的冗余信息，也就是说，这些信息对于理解这条消息来说是不必要的。通过一个例子或许能看清楚为什么需要冗余信息。请考虑一家名为 The Couch Potato（TCP）的邮购公司，它有 60 000 多种产品。想象该公司的效率很高，TCP 公司的程序员决定，每条订购消息应该包含一个 16 字节的顾客名以及一个 3 字节的数据字段（一个字节用于购买数量，另 2 个字节用于产品编号）。最后 3 个字节用一个非常长的密钥进行加密，并且只有顾客和 TCP 公司才知道该密钥。

初看起来这种方案似乎非常安全，从某种意义上来看它确实是安全的，因为被动的入侵者不可能解密消息。不幸的是，它有一个致命的缺陷使得它毫无用处。假定有一个最近刚刚被解雇的员工想要报复 TCP 公司。在离开之前，她带走了顾客的名单。她连夜编写了一个程序来产生伪造的订单，这些订单的客户名是真实的。由于她没有拿到客户们的密钥，所以她在最后的 3 个字节中只是放了一些随机数，然后给 TCP 公司发送了几百份订单。

当这些订购消息到达公司时，TCP 公司计算机利用客户的名字找到对应的密钥，并解密消息中的最后 3 个字节。对于 TCP 公司来说，不幸的是几乎所有的 3 字节消息都是有效的，所以，计算机开始打印发货指令。虽然客户订购 837 套儿童秋千，或者 540 个沙箱，这看起来非常奇怪，但计算机知道的就是这些信息，也许顾客可能正在筹建一个获准特许经营的游乐场。以这种方式，一个主动的入侵者（离职员工）即使并不理解她自己的计算机所产生的消息，她也可以通过这种方式造成大量的麻烦。

实际上，通过在所有的消息中增加一定的冗余度就可以解决这个问题。例如，如果订购消息被扩展到 12 个字节，前 9 个字节必须为 0，那么，这种攻击便不再奏效，因为离职员工不可能产生大量的有效消息。从这个例子得到的启示是：所有的消息必须包含足够大的冗余度，因而主动入侵者发送的随机垃圾消息不可能有机会被解释为有效的消息。

然而，增加冗余度也使得密码分析者更加容易破解消息。假设邮购业务的竞争非常激烈，TCP 公司的主要竞争对手 Sofa Tuber 极力想知道 TCP 公司卖出了多少个沙箱。因此，他们搭接在 TCP 公司的电话线上。在原来的 3 字节消息的方案中，密码分析手段几乎不可能成功，因为在试验了一个密钥之后，分析者无从判断他的猜测是否正确。毕竟，从技术上讲，几乎每一条消息都是合法的。但是，在采用了新的 12 字节的方案以后，分析者可以很容易地区分出有效的消息和无效的消息。因此，我们得到了：

密码学原则 1：消息必须包含一定的冗余度

换句话说，在解密了一条消息以后，接收者必须能够通过简单的检查手段和通过一个简单的计算来判断该消息是否有效。这种冗余是必要的，它可以用来阻止主动入侵者发送垃圾信息，或者防治接收者被诱骗解密垃圾信息，然后处理所谓的“明文”。然而，同样这冗余信息也使得被动入侵者更加容易破解系统，所以这里有点矛盾。而且，冗余信息不应该采用在消息的开头或者末尾加上 n 个 0 的形式，因为在某些密码算法上加解密这样的消息可以得到一些更有预测性的结果，从而使密码分析者的工作更加容易。CRC 多项式编码比加 0 的做法好得多，因为接收者很容易验证消息，而密码分析者却需要更大的工作量。更好的做法是使用密码散列，这个概念我们将在本章后面学习。此刻，我们就认为它是一种更好的 CRC。

再回到稍早前的量子密码上，我们可以看到冗余原则在那里扮演了多么重要的角色：由于 Trudy 截获了光子，所以造成 Bob 的一次性密钥中有些位是错误的。Bob 需要入境消息具备一定的冗余度来判断该消息是否出现了错误。一种非常原始的冗余形式是将消息重复两遍。如果两份副本不相同，那么 Bob 就知道要么光纤上噪声很大，要么有人篡改了传输信息。当然，将所有的信息都发送两份这种做法有点矫枉过正，采用海明码或者里德所罗门编码可以更加有效地实行检错和纠错。但是，应该明确引入冗余度能区分有效消息和无效消息，特别是在面对主动入侵者的情况下。

新鲜度

第二条密码学原则是，必须采取某些措施确保每条接收到的消息可被验证是新鲜的，也就是说，它是最近刚发送出来的。为了预防主动入侵者重放老消息，这种措施是必要的。如果没有相应的措施，那么，我们的例子中的离职员工就可以搭接在 TCP 公司电话线上，然后不停地重复以前发送的有效消息。因此：

密码学原则 2：需要采取某种方法来对抗重放攻击

一种措施是在每条消息中包含一个仅仅在一段时间内（比如 10 秒钟）有效的时间戳。然后，接收方只要保留该消息 10 秒钟，并且将新到达的消息与以前的消息进行比较就可以过滤掉重复的消息。10 秒钟之前到达的消息可以被丢弃，因为凡是在 10 秒钟以后发送的任何重放消息将被当作太老的消息而遭到拒绝。除了时间戳以外的其他措施将在本章后面讨论。

8.2 对称密钥算法

现代密码学使用了与传统密码学相同的思想（替代和置换），但是它的重点有所不同。传统上，密码设计者使用了非常简单的算法。现在，情形完全相反：密码设计的目标是使加密算法尽可能地错综复杂，因而即使密码分析者获得了大量的选择密文，在没有密钥的情况下他也不能够推测出明文。

我们在本章中将要学习的第一类加密算法称为**对称密钥算法**（symmetric key algorithm），因为它们使用同样的密钥来完成加密和解密操作。图 8-2 显示了对称密钥算法的用法。尤其是，我们将焦点集中在**块密码**（block cipher），它接受一个 n 位的明文块作为

输入，并利用密钥将它变换成n位的密文块。

密码算法既可以用硬件来实现（为了速度），也可以用软件来实现（为了灵活）。虽然在本书中我们的主要注意力集中在算法和协议上，而算法和协议本身是独立于它们的具体实现的，但简要提及有关构造密码系统的硬件或许饶有趣味。替代和转置密码可以用简单的电子电路来实现。图8-6(a)显示了一种称为P盒（P-box）的设备（这里的P代表数学中的**排列**（permutation）），P盒的效果相当于对一个8位输入实现一个替代操作。如果这8位从上到下被指定为01234567，则这个特定P盒的输出就是36071245。通过适当的内部连线方式，P盒可以执行任何一个替代操作，而且在实践中可以达到光的速度，因为这中间没有涉及任何计算过程，只是信号传播而已。这种设计符合Kerckhoff原则：攻击者知道对输入的位重新排序的通用方法，但他不知道的哪些位应该出现在哪里。

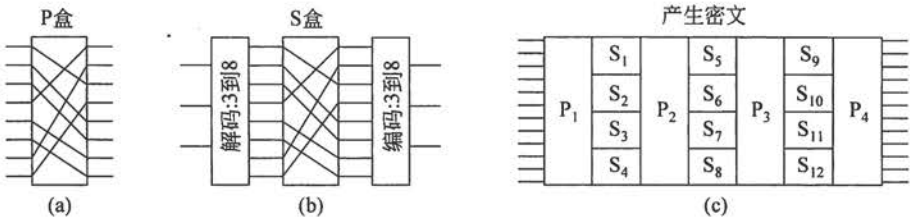


图8-6 乘积密码的基本元素
(a) P盒；(b) S盒；(c) 乘积

置换由**S盒**（S-box）完成，如图8-6(b)所示。在这个例子中，输入是3位明文，输出是3位密文。首先，3位输入从第一阶段的8根输出线中选择其中一根线，并将它设置为1，所有其他的线均为0。第二阶段是一个P盒。第三阶段再将选中的输入线编码成二进制。按照图中显示的连线方式，如果依次输入8个八进制数01234567，则输出序列将是24506713。换句话说，0被2代替，1被4代替，等等。同样地，通过正确地设置S盒内部的P盒，任何一种置换都可以用S盒来完成。而且，这样的设备可以直接用硬件来构造，从而获得极快的速度，因为编码器和解码器只有一个或者两个亚纳秒（subnanosecond）门延迟，而通过P盒的传播延迟可能小于1微微秒（picosec）。

只有当我们将整个系列的盒叠加起来构成**乘积密码**（product cipher）时，这些基本元素的真正威力才显示出来，如图8-6(c)所示。在这个例子中，第一步（P_1）将12根输入线做替代操作（即重新排列）。第二步将输入分成4个3位一组，每一组被独立于其他（S_1～S_4）做置换操作。这样的安排展示了一种通过多个小S盒来近似达到一个大S盒效果的方法。因为小盒子在硬件实现上更现实（例如，一个8位S盒可以实现一个256项的查询表）而大S盒笨重难以制造（例如一个12位的S盒其中间步骤将至少需要2^{12}=4096根交叉线）。虽然这种方法缺乏通用性，但是它仍然非常有用。通过在乘积密码中加入充分大量的处理步骤，获得的输出可以是输入的极其复杂函数。

对k位输入执行一个乘积密码算法以产生一个k位输出，这非常常见。典型情况下，k介于64～256。在硬件实现中，通常至少有10个物理步骤而不像图8-6(c)那样只有7个步骤。而在软件实现中，乘积密码可以被编写成一个至少执行8次迭代的循环，每次迭代都在64～256位数据块的子块上执行类似S盒的置换操作，然后通过一个替代操作将这些S盒的输出全部混合起来。通常在起始处有一个特殊的初始替代操作，在末尾也有一个替代

操作。在许多文献中，这样的迭代称为**轮**（round）。

8.2.1 DES——数据加密标准

1977 年 1 月，美国政府采纳了 IBM 开发的一个乘积密码作为非机密信息的官方标准。这个密码算法，即**数据加密标准**（DES，Data Encryption Standard）已被工业界广泛应用于安全产品中。它最初的形式已经不再安全了，但是它的一个修订形式仍然非常有用。现在我们来学习 DES 的工作原理。

DES 的基本结构如图 8-7(a)所示。明文按 64 位数据块被加密，生成 64 位密文。DES 算法以一个 56 位密钥作为参数，它共有 19 个步骤。第一步是一个与密钥无关的替代操作，它直接作用在 64 位明文上。最后一步是这个替代操作的逆操作。倒数第二步是交换左 32 位和右 32 位。剩下的 16 步在功能上完全相同，但使用了原始密钥的不同函数作为参数。DES 算法的设计允许使用同样的密钥来完成解密过程，这正是任何一个对称密钥算法必须满足的一个特性。在 DES 算法中，解密的步骤只是加密步骤的相反顺序而已。

这些中间步骤的操作如图 8-7(b)所示。每个步骤接受 2 个 32 位输入，并产生两个 32 位输出。左边的输出是右边输入的一份副本。右边的输出是左边输入与一个函数值逐位异或的结果，该函数的输入参数有两个，分别是右边的输入和当前步骤所用的密钥 K_i。所有的复杂性都体现在这个函数中。

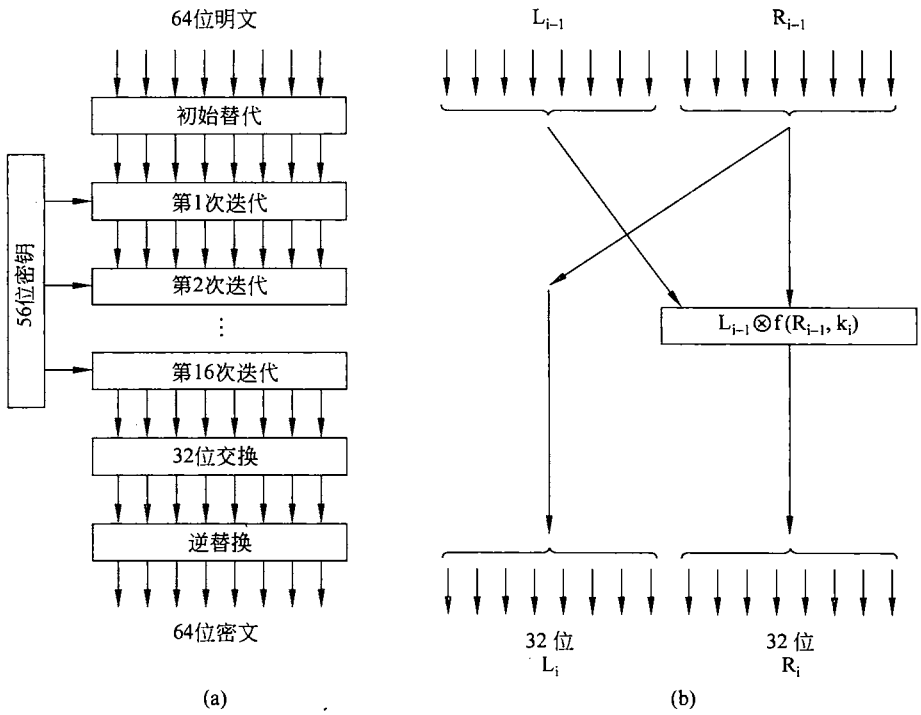


图 8-7 数据加密标准
(a) 一般结构；(b) 一次迭代的细节
⊗ 表示“异或”操作

此函数包含 4 个顺序执行的步骤。第一步，根据一个固定的替代和复制规则，将 32

位 R_{i-1} 扩展成一个 48 位的数字 E。第二步，E 和 K_i 被异或在一起。然后，异或的结果被分成 8 个 6 位组，每个 6 位组被输入到一个不同的 S 盒中。对于每个 S 盒，共有 64 种可能的输入，每种输入被映射到一个 4 位输出上。最后，将这 8×4 位通过一个 P 盒。

这 16 次迭代中的每一次迭代使用了不同的密钥。在算法开始之前，先在 56 位的密钥上执行一个 56 位替代操作。在每一次迭代之前，密钥被分成两个 28 位，每个 28 位向左循环移位，移动的位数取决于当前的迭代次数。移位以后再执行另一个 56 位替代即可导出 K_i。在每一轮上，从这 56 位中提取出不同的 48 位子集并对它进行排列。

有时候可以用一项特殊的技术来加强 DES 的强度，这项技术称为**白化**（whitening）。这项技术的做法是，在将每一个明文数据块送给 DES 之前，先用一个随机 64 位密钥对它执行异或操作，然后在传输之前对 DES 输出的结果密文用第二个 64 位密钥执行异或操作。去掉白化也非常容易，只要运行逆向操作即可（前提条件是接收方也拥有这两个白化密钥）。由于这项技术有效地增加了密钥的长度，所以，它使得用穷举法来搜索密钥空间需要消耗更多的时间。请注意，每个明文数据块使用同样的白化密钥（即这里总共只有一个白化密钥）。

从 DES 被发布的那一天起，有关 DES 的争议就一直不断。DES 建立在一个由 IBM 公司开发并拥有专利的加密算法基础之上，该算法称为 Lucifer，它使用了 128 位密钥，而 DES 采用了 56 位密钥。当美国联邦政府希望为非机密的用途建立一个加密算法标准时，它“邀请”IBM 公司与 NSA（美国政府中专门破解密码的机构）“讨论”有关的事项。NSA 雇佣了一支世界上最强大的数学家和密码学家队伍。NSA 是如此之神秘，以至于在工业界流传着这样一个笑话：

问：NSA 代表什么？

答：代表“不存在的机构”（No Such Agency）。

实际上，NSA 代表了国家安全局（National Security Agency）。

经过讨论之后，IBM 公司将密钥从 128 位缩短到 56 位，并且将 DES 的设计过程加以保密。许多人怀疑缩短密钥长度的原因是为了保证只有 NSA 能够破解 DES，而经费预算较少的其他组织却无法破解 DES。这种不公开设计过程的做法使人们猜测 DES 算法中隐藏了一个后门，NSA 利用这个后门可以很容易地破解 DES。当 NSA 的一个雇员非常谨慎地通知 IEEE 取消一个计划好的密码学会议时，更加重了人们的这种担心。但 NSA 否认这一切。

1977 年，斯坦福大学的两位密码学研究人员 Diffie 和 Helman（1977）设计了一台能破解 DES 的机器，估计它的造价需要两千万美元。给定一小段明文和匹配的密文，这台机器利用穷举法搜索整个密钥空间（共有 2^{56} 个密钥），可以在 1 天内找到密钥。现在，这样一台机器的造价已经大大低于 100 万美元（Kumar 等，2006）。

三重 DES

1979 年初，IBM 公司意识到 DES 的密钥长度太短，于是设计了一种方法，利用三重加密来有效地增加密钥长度（Tuchman，1979）。所选择的方法如图 8-8 所示，该方法后来已被集成到国际标准 8732 中。这种方法使用两个密钥，共包含了 3 个步骤。第 1 步按照常规的方式用密钥 K_1 执行 DES 加密；在第 2 步中，DES 以解密方式运行，并使用 K_2 作为密

钥。最后，再次用 K_1 执行 DES 加密。

这种设计立即引发了两个问题。第一，为什么只使用两个密钥，而不是三个？第二，为什么使用 **EDE（加密-解密-加密）**模式，而不是 **EEE（加密-加密-加密）**呢？使用两个密钥的理由是，即使最为多疑的密码学家也认为，就目前而言，对于常规的商业应用来说 112 位已经足够了（对于密码学家来说，多疑往往被认为是一种个性，而不是缺陷）。将密钥长度增加至 168 位只会带来管理和传输另一个密钥的不必要开销，同时又没有任何实际的益处。

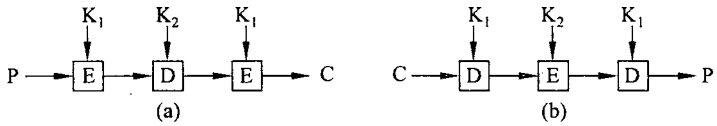


图 8-8

(a) 用 DES 进行三重加密；(b) 解密

采用“加密，解密，再加密”的理由是为了与现有的单密钥 DES 系统保持后向兼容性。DES 的加密和解密函数都是 64 位整数集之间的映射关系。从密码学的角度来看，这两个映射是同等强度的。然而，采用 EDE 而不是 EEE 之后，使用三重 DES 的计算机就可以与使用单密钥 DES 的计算机进行通话，做法很简单，只要设置 $K_1=K_2$ 即可。这种特性使得三重 DES 可以被逐步分阶段地应用到实际系统中，不过，学院派的密码学家对此并不感兴趣，但对于 IBM 公司和它的客户来说却非常重要。

8.2.2 AES——高级加密标准

随着 DES 逐渐走到生命的尽头，甚至三重 DES 也难以避免这样的厄运，**美国国家标准及技术委员会**（NIST，National Institute of Standards and Technology）决定美国政府需要一个新的密码标准作为非机密用途，NIST 是美国商务部的代理机构，承担着为美国联邦政府核定标准的重任。NIST 对所有关于 DES 的争议知道得非常清楚，它同时也知道，如果它直接宣布一个新的标准，则任何一个具有丰富密码学知识的人很自然就会设想 NSA 已经在标准算法中内置了一个后门，因而 NSA 可以读取一切用该算法来加密的信息。在这样的情况下，可能没有人会使用此标准，它最终极有可能悄悄死去。

所以，NIST 采取了一种与政府官僚主义截然不同的做法：它发起了一场密码学比赛（竞赛）。1997 年 1 月，世界各地的研究人员接到邀请，希望为一个新的标准提交方案，这个新标准称为**高级加密标准**（AES，Advanced Encryption Standard）。比赛规则如下：

（1）算法必须是一个对称的块密码。

（2）所有的设计必须公开。

（3）必须支持 128、192、256 位密钥长度。

（4）软件实现和硬件实现必须都是可能的。

（5）算法必须是公有的，或者毫无歧视地授权给大众使用。

NIST 共收到了 15 个提案，于是它组织公开的会议展示这些提案，并积极鼓励参加会议的人寻找这些提案中的漏洞。1998 年 8 月，NIST 根据这些算法的安全性、效率、简单

性、灵活性和内存需求（对于嵌入式系统来说是非常重要的），选出5个算法进入最后的决赛。接下来又召开了更多的学术讨论会，同时也发生了更多的激烈论战。

2000年10月，NIST宣布Joan Daemen和Vincent Rijmen提出的Rijndael胜出。名称Rijndael是根据两位作者的姓氏（Rijmem+Daemen）合成的，其发音为Rhine-doll（差不多是这样）。在2001年11月，Rijndael成为AES美国政府标准，发布为FIPS 197（联邦信息处理标准）。由于整个竞争过程绝对的开放，Rijndael的技术特性以及获胜的小组是由两位年轻的比利时密码学家组成的（看上去不太可能为了迎合NSA而在算法内设置个后门），所以，Rijndael已经变成世界上占主导地位的密码学密码。AES的加密和解密现在是一些微处理器指令集的一部分（比如，Intel公司）。

Rijndael支持密钥长度和块的长度可以从128位一直到256位（以32位为间隔逐步向上递增）。密钥长度和块长度可以独立选择。然而，AES规定块长度必须是128位，密钥长度必须是128、192、256位。尽管如此，很怀疑是否有人会使用192位密钥，所以，事实上，AES提供两个变种：(1) 数据块为128位，密钥为128位；(2) 数据块为128位，密钥为256位。

在算法的处理中，我们将仅讨论128/128的情形，因为它有可能会变成商业规范。在128位的密钥空间中有$2^{128} \approx 3 \times 10^{38}$个密钥。即使NSA想办法建造一台内含10亿个并行处理器的机器，并且每个处理器每微微秒就可以评估一个密钥，这样的机器也需要10^{10}年才能搜索完整个密钥空间。到那时候太阳的能量已经消耗殆尽了，因此人们只好点着蜡烛阅读解密结果了。

Rijndael

从数学的角度来看，Rijndael是以迦罗瓦（Galois）域理论为基础的，Galois域理论赋予Rijndael一些可以证明的安全特性。然而，我们也可以用C代码的形式来了解Rijndael算法，而不涉及数学理论。

如同DES一样，Rijndael也使用置换和替代操作，并且它也使用了多轮策略。具体的轮数取决于密钥的长度和块的长度，对于128位密钥和128位块长度的情形，轮数为10；对于更长的密钥或者更长的数据块，轮数可以增加到14。然而，与DES不同的是，所有的操作都牵涉到整个字节，从而允许用硬件和软件高效地实现这些操作。Rijndael的代码结构如图8-9所示。这个代码只是用作说明目的。安全代码的良好实现将遵从其他的实践规则，例如在使用后完全清楚敏感记忆。请参考（Ferguson等，2010）中的例子。

函数rijndael有3个参数。它们分别是：plaintext（明文），一个包含了输入数据的16字节数组；ciphertext（密文），也是一个16字节的数组，它包含的是加密之后返回的输出结果；key（密钥），是16个字节长的密钥。在计算过程中，数据的当前状态被保存在一个字节数组state中，该数组的长度为NROWS×NCOLS。对于128位数据块，该数组为4×4字节。利用这16字节可以存储整个128位数据块。

state（状态）数组首先被初始化为明文数据，然后在计算过程中每一步都要被修改。在有些步骤中，要执行字节对字节的置换；在其他的步骤中，这些字节被执行数组内的替代操作（即在数组内部重新排列顺序）。Rijndael还用到了其他的变换。在算法的最后，state中的内容被作为密文返回。

```
#define LENGTH 16           /* # bytes in data block or key */
#define NROWS4              /* number of rows in state */
#define NCOLS4              /* number of columns in state */
#define ROUNDS 10           /* number of iterations */
typedef unsigned char byte; /* unsigned 8-bit integer */
rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
    int r;                                          /* loop index */
    byte state[NROWS][NCOLS];                       /* current state */
    struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1]; /* round keys */
    expand_key(key, rk);                  /* construct the round keys */
    copy_plaintext_to_state(state, plaintext);  /* init current state */
    xor_roundkey_into_state(state, rk[0]);      /* XOR key into state */
    for (r = 1; r <= ROUNDS; r++) {
        substitute(state);                /* apply S-box to each byte */
        rotate_rows(state);                /* rotate row i by i bytes */
        if (r < ROUNDS) mix_columns(state); /* mix function */
            xor_roundkey_into_state(state, rk[r]); /* XOR key into state */
    }
    copy_state_to_ciphertext(ciphertext, state); /* return result */
}
```

图 8-9 C 代码的 Rijndael 结构

在代码的开始处，首先将密钥扩展到 11 个与 state 同样长度的数组中。它们被保存在 rk 中，这里的 rk 是一个结构数组，其中每个结构元素包含一个状态数组。在这 11 个数组中，有一个被用在计算过程的开始处，其他 10 个被用在 10 轮计算中，每轮一个数组。从加密密钥（即原始密钥）导出每一轮密钥的过程非常复杂，因此我们这里不讨论这个过程。可以简单地这样来描述，轮密钥是通过反复地对密钥中不同的位组进行循环移位和异或操作而生成的。关于这些细节，请参考（Daemen 和 Rijmen，2002）。

接下来的一步是将明文复制到 state 数组中，从而在后续的轮循环中得以处理。复制过程按照列的顺序进行，即前 4 个字节复制到第 0 列中，接下来 4 个字节复制到第 1 列中，以此类推。列和行都从 0 开始编号，但是轮数从 1 开始编号。图 8-10 显示了 12 个 4×4 大小的数组的初始化过程。

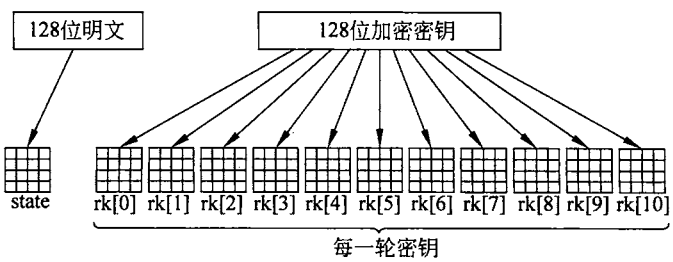


图 8-10 创建状态和 rk 数组

在主要计算过程开始之前还有一步：rk[0]被逐个字节地异或（XOR）到 state 中。换句话说，在 state 数组的 16 个字节中，每个字节都被它与 rk[0]中对应的字节异或之后的结果所取代。

现在该进入主循环了。该循环共执行10次迭代，每轮一次，并且在每一次迭代中都要变换state。每一轮中的内容由4个步骤产生。第1步在state上执行一个逐字节的置换。按照顺序，每个字节被当作一个索引引用到一个S盒中，然后以该S盒中对应项的内容来替换该字节的值。这一步是一个直接的单字母表置换密码。与DES不同，DES有多个S盒，但Rijndael只有一个S盒。

在第2步中，4行中的每一行向左循环移动。第0行移动0字节（即不改变）；第1行左移1个字节；第2行左移2个字节；第3行左移3个字节。这一步起到的作用是将当前的数据内容扩散到整个块中，其效果类似于图8-6中的替代操作。

第3步独立地混合每一列，列与列之间并不相干。这个混合操作是通过矩阵乘法来完成的，在矩阵乘法中，新的列是老的列与一个常量矩阵的乘积，这里的乘法是有限Galois域上的乘法，即GF（2^8）。虽然这听起来好像很复杂，但实际上存在一个算法可以用两个查表操作和三个XOR操作来计算新列的每个元素（Daemen和Rijndael，2002，附录E）。

最后，第4步将这一轮的密钥异或（XOR）到state数组中以便下一轮使用。

由于每一步都是可逆的，所以解密过程也很简单，只要反过来运行这个算法的每一步即可。然而，通过使用技巧，也可以用该加密算法来完成解密，只是用到的数据表不相同。

该算法的设计不仅具有极高的安全性，同时也达到极快的速度。在2 GHz的机器上，一个良好的软件实现应该可以达到700 Mbps的加密速率，这个速度足以实时地加密100部MPEG-2视频。当然，用硬件实现更快。

8.2.3 密码模式

尽管AES算法（或者DES，或者其他任何块密码算法）如此复杂，但归根到底，它们本质上只是一种单字母置换密码算法，只不过使用了极大的字符（对于AES为128位字符，对于DES为64位字符）。任何时候当同样的明文块进入到算法前端时，同样的密文块就从后端输出。如果你用同样的DES密钥加密明文abcdefgh 100次，那么你就会得到同样的密文100次。入侵者可以充分发掘这种特性来协助攻破整个密码系统。

电码本模式

为了看清楚如何利用这种“单字母置换密码算法”的特性来局部破解密码算法，我们将使用（三重）DES作为例子，因为描述64位数据块比128位数据块更容易，而且AES也有同样的问题。为了使用DES来加密一长段明文，最简单的做法是将它分割成连续的8字节（64位）数据块，然后用同样的密钥逐个加密这些数据块。如果有必要的话，将最后一段明文填补至64位。这项技术称为**电码本**（ECB，Electronic Code Book）模式，它类似于老式代码簿。在老式的代码簿中，每个明文单词的旁边列出了它的密文（通常是5个十进制数字）。

如图8-11所示，我们从一个计算机文件开始，该文件列出了某一家公司已确定的奖励给雇员的年终奖金。文件由连续的32字节长的记录构成，每个雇员一条记录，正如图中所显示的，其格式为：16字节的名字、8字节的职位和8字节的奖金。现在利用（三重）DES将图中显示的16个8字节块（从0到15编号）进行加密。

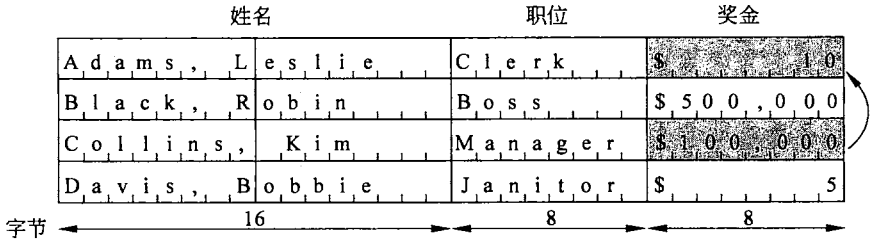


图 8-11 一个文件的明文被加密成 16 个 DES 块

Leslie 刚刚与老板有过冲突，因此他不期望会得到很多奖金。相反，Kim 则深得老板的喜爱，而且每个人都知道这一点。在这个文件被加密之后和它被送给银行之前，Leslie 有机会访问该文件。请问，在仅仅拿到加密文件的情况下，Leslie 有可能扭转这种不公平的局面吗？

这完全没有问题。Leslie 所需要做的事情是，将第 12 个密文块（这是 Kim 的奖金）做一份副本，然后用它替换掉第 4 个密文块（这是 Leslie 的奖金）。虽然 Leslie 并不知道第 12 个密文块到底包含了什么内容，但是他可以期望今年有一个愉快的圣诞节（复制第 8 个密文块也是可以的，但是极有可能被检测出来；另外，Leslie 并不是一个很贪婪的人）。

密码块链模式

为了对抗这种类型的攻击，所有的块密码算法可以按各种不同的方式链接起来。那样，Leslie 所做的密文块替换方法将导致在解密之后，从被替换的块开始的明文变成垃圾数据。一种链接方法是**密码块链**（cipher block chaining）。在这种方法中，如图 8-12 所示，每个明文块在被加密之前，先与上一个密文块执行一次异或操作。因此，同样的明文块不再被映射到同样的密文块上，加密操作也不再是一个大的单字母置换密码了。第一个块与一个随机选取的**初始矢量**（IV，Initialization vector）执行异或操作，该初始向量（以明文方式）随着密文一起被传输。

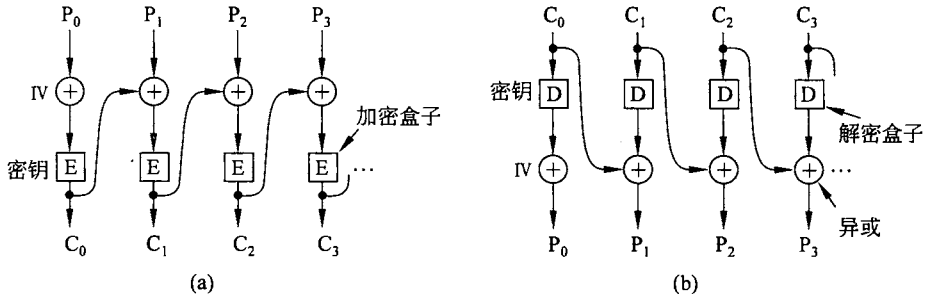


图 8-12 密码链接块

(a) 加密；(b) 解密

通过查看图 8-12 中的例子，我们可以看出密码块链模式的工作原理。首先我们计算 $C_0 = E(P_0 \text{ XOR } IV)$，然后计算 $C_1 = E(P_1 \text{ XOR } C_0)$，以此类推。解密过程使用 XOR 逆转整个过程，有 $P_0 = IV \text{ XOR } D(C_0)$，以此类推。请注意，第 i 块的加密操作是从 0 到 i-1 块所有明文的一个函数，所以同样的明文根据它所出现的位置的不同将生成不同的密文。像 Leslie 所做的变换将导致从 Leslie 的奖金域开始的两个块变得没有任何意义。对于精明的安全检查官来说，这种特点可能正好暗示了应该从哪里开始进行深入的调查。

密码块链模式还有一个好处，即同样的明文块并不导致同样的密文块，这使得密码分析更加困难。实际上，这也正是它被广泛采用的原因。

密码反馈模式

然而，密码块链模式也有一个缺点，即只有当整个64位数据块到达以后才可以开始解密。对于逐字节的加密应用，可以使用如图 8-13 所示的**密码反馈模式**（cipher feedback mode），它使用了（三重）DES。对于 AES，基本思想完全相同，只要使用 128 位移位寄存器。在这个图中，加密机正好处于“字节 0 至 9 已经被加密并发送出去”之后的状态。当明文字节 10 到来时，如图 8-13(a)所示，DES 算法被作用在 64 位移位寄存器上，并产生一个 64 位密文。该密文最左边的字节被提取出来，并且与 P_{10} 进行异或。然后该字节被传送到输出线路上。并且，移位寄存器左移 8 位，使得 C_2 被从最左边移走，而 C_{10} 被插入到 C_9 右边空出来的位置中。

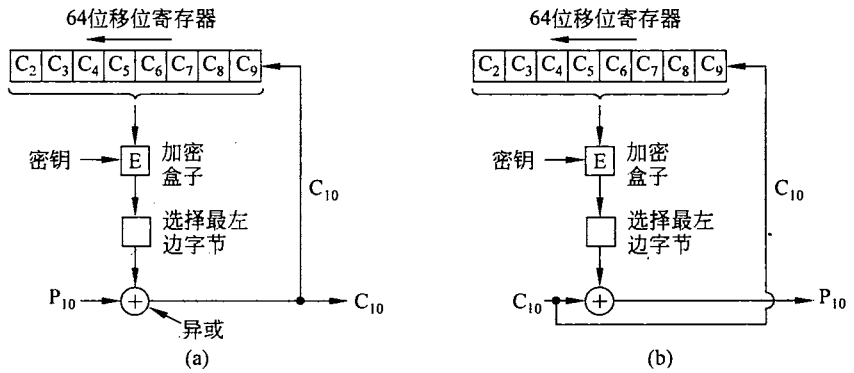


图 8-13 密码反馈模式
(a) 加密；(b) 解密

请注意，移位寄存器的内容依赖于明文的全部以前的历史，所以，若一个模式在明文中重复多次，则经过加密以后，在密文中每次出现都不相同。如同密码块链模式一样，密码反馈模式也需要一个初始向量来启动整个加密过程。

密码反馈模式的解密过程如同加密过程一样。尤其是，移位寄存器的内容是被加密，而不是被解密，所以，为了生成 P_{10} 而被选出来与 C_{10} 做异或的那个字节，与在加密过程中为了生成 C_{10} 而与 P_{10} 做异或的字节是同一个字节。只要两个移位寄存器保持不变，解密过程就不会错。图 8-13(b)显示了这一过程。

密码反馈模式的一个问题是，如果密文在传输过程中只有一位意外发生翻转，则在解密过程中，当坏字节位于移位寄存器中时所波及的 8 个字节都将遭到破坏。一旦坏字节移出了移位寄存器以后，后面产生的明文仍然是正确的。因此，单位翻转错误只影响相对局部的区域，而不会破坏消息中其余剩下的部分，但是其影响的位数与移位寄存器的宽度相等。

流密码模式

然而对于一些应用来说，1 位传输错误会捣乱 64 位明文，这种影响还是太大了。对于这些应用来说，它们可以选择第四种方案，即**流密码模式**（stream cipher mode）。它的工作

方式是用一个密钥来加密一个初始向量以便生成一个输出块；然后用同样的密钥对这个输出块进行加密得到第二个输出块；再对这一块进行加密以生成第三块，以此类推而进行加密。输出块的序列（可以任意长）称为**密钥流**（keystream），它就像一次性密钥那样与明文做异或操作来生成密文，如图 8-14(a)所示。请注意，初始矢量 IV 仅仅被用在第一步中；在此以后，输出数据块被加密。同时也要注意，密钥流与数据是独立的，所以，如果有必要的话，可以将密钥流提前计算出来，而且，它对于传输错误完全不敏感。解密过程如图 8-14(b)所示。

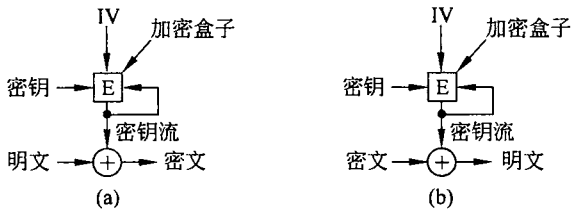


图 8-14 流密码模式
(a) 加密；(b) 解密

解密过程发生在接收端生成相同的密钥流。由于密钥流仅仅依赖于初始矢量 IV 和密钥，所以，它不会受到密文中传输错误的影响。因此，密文中的 1 位传输错误仅仅导致被解密出来的明文中产生同样的 1 位错误。

很关键的一点是，对于一个流密码，永远不要两次使用同样的（密钥，IV）对，因为一旦这样做以后，则每次都会生成同样的密钥流。两次使用同样的密钥流将导致密文受到**密钥流重用攻击**（keystream reuse attack）。假定明文块 P_0 经过一个密钥流加密之后得到 P_0 XOR K_0。后来，第二个明文块 Q_0 也用同样的密钥流加密之后得到 Q_0 XOR K_0。一个入侵者捕捉到这两个密文块以后，他只需将它们异或一下就可以得到 P_0 XOR Q_0，从而消掉了密钥。现在入侵者有了两个明文块的 XOR 结果。如果其中一个明文被入侵者知道或者猜中的话，另一个明文自然也暴露无遗。无论如何，利用消息的统计特性，对两个明文流的 XOR 结果实施破解的可能性更大。例如，对于英语文本，在流中最常见的特征可能是两个空格的 XOR，其次是空格和字母“e”的 XOR，等等。简而言之，有了两个明文的 XOR 信息以后，密码分析者就有更多的机会来推断出这两个明文。

计数器模式

除了电码本模式以外其他所有模式都存在这样一个问题：要想随机访问加密之后的数据是不可能的。例如，假设通过网络传输一个文件，然后以密文的形式将文件存储在磁盘上。如果接收端计算机是一台有可能被偷走的笔记本电脑，那么这可能是一种非常合理的工作方式。使所有关键的文件都以加密的形式来保存可以极大地降低潜在的危险性，因为即使计算机落入敌手之后秘密信息也不易泄漏出去。

然而，磁盘文件通常是以非顺序的方式来访问的，特别是数据库中的文件。如果用密码块链模式来加密一个文件，那么为了访问一个随机的块，要求解密在它之前的所有数据块，而这是一个非常昂贵的操作。出于这个原因，后来又发明了另一种模式，即**计数器模式**（counter mode），如图 8-15 所示。在这里，明文并不直接被加密，相反，被加密的是初

始矢量加上一个常量的值，结果得到的密文再与明文做异或。通过“为每个新的数据块使初始向量递增 1”这种办法，在文件中任何地方的块都可以直接解密，而无须解密所有在它之前的数据块。

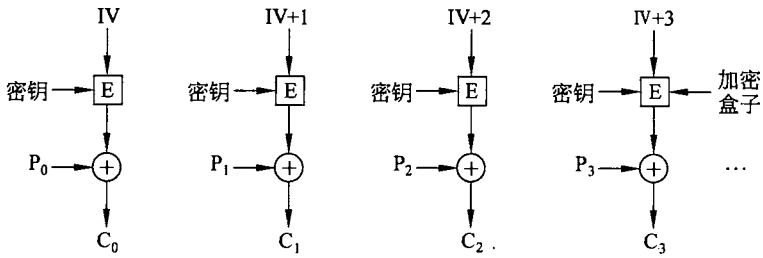


图 8-15 使用计数器模式的加密过程

尽管计数器模式非常有用，但它有一个弱点值得指出。假设同样的密钥 K 在将来又被使用了（明文不同，但 IV 相同），并且攻击者获得了这两次运行中所有的密文。这两种情形下的密钥流是相同的，从而使密码算法遭受密钥流重用攻击，我们在讨论流密码模式时曾经提到过这种类型的攻击。密码分析者只要将两个密文 XOR 在一起，就可以消除所有的密码保护，并获得明文的 XOR 结果。这个弱点并不意味着计数器模式的想法不好，而仅仅说明密钥和初始向量应该独立地随机选取。即使同样的密钥偶尔被用到了两次，如果这两次的 IV 不相同的话，则明文仍然是安全的。

8.2.4 其他密码模式

如果仅考虑可靠性，那么 AES（Rijndael）和 DES 是众所周知的对称密钥密码学算法和业界选择的标准（如果你使用 AES 的产品而 AES 被破解，那么没有人会抱怨；但是如果你选用一个非标准密码而密码被破解，那么一定会有人怪你）。然而，值得说明的是已经有些其他对称密钥算法被实际使用。它们中的一些被嵌入到不同的产品。图 8-16 列出了少数几个比较常见的其他对称密钥。也可以使用组合密钥，例如，在使用 Twofish 加密后再用 AES 加密，必须破解两个密码才能恢复数据。

密码算法	作者	密钥长度	注释
DES	IBM	56位	太脆弱，现已不用
RC4	Ronald Rivest	1~2048位	小心：某些密钥很脆弱
RC5	Ronald Rivest	128~256位	好，但是专利
AES(Rijndael)	Daemen和Rijmen	128~256位	最好的选择
Serpent	Anderson、Biham、Knudsen	128~256位	很强大
Triple DES	IBM	168位	好，但年代久了
Twofish	Bruce Scnneier	128~256位	很强大，被广泛使用

图 8-16 一些常见的对称密钥密码学算法

8.2.5 密码分析

在结束关于对称密码学主题的讨论之前，至少应该提及密码分析领域中的 4 个进展。

第一个进展是**区分密码分析**（differential cryptanalysis）（Biham 和 Shamir，1993）。这项技术可以被用来攻击任何一种块密码算法。它的工作原理是：首先用一对仅在少量的位上有差异的明文块来执行加密，然后仔细观察加密过程中内部每一次迭代所发生的情况。在许多情况下，有些位模式可能比其他的位模式更经常出现，这种现象可以导致概率攻击。

第二个值得提及的密码分析进展是**线性密码分析**（linear cryptanalysis）（Matsui，1994）。它只需用 2^{43} 个已知明文就可以破解 DES。它的工作原理是：将明文和密文中特定的位 XOR 在一起，然后检查结果。当重复这样的过程后，应该一半的位为 0，另一半的位为 1。然而，通常情况下密码算法会在某一个方向上引入偏差，不管这个偏差有多小，它总可以用来降低工作量。有关细节过程，请参考 Matsui 的论文。

第三个进展是通过对电子功率消耗的分析来发现秘密密钥。计算机通常使用 3 伏代表“1”位，用 0 伏来代表“0”位。因此，处理“1”比处理“0”需要更多的电能。如果一个密码算法由一个循环来完成的，并且这个循环按照顺序逐位地处理密钥中的位，那么，攻击者用一个慢的时钟（比如 100 Hz）来代替 n GHz 的主时钟，并且用弹簧夹夹住 CPU 的电源和地线就可以精确地监视每条机器指令所消耗的功率。利用这些数据来推断出密钥的做法竟然出奇地容易。这种密码分析方法很容易被挫败，只要用汇编语言小心地编写算法，以确保功率消耗与密钥独立，也与每一轮的密钥独立。

第四个进展是时间分析。密码算法往往充满了对轮密钥中的位进行测试的 if 语句。如果 then 和 else 部分的执行需要不同长度的时间，则只要减慢时钟，并且观察每一步所花的时间，就有可能推断出轮密钥。一旦获知了所有的轮密钥，原始的密钥就很容易被计算出来。还可以同时使用功率分析法和时间分析法，从而使破解工作更加容易。虽然功率分析和时间分析看起来好像是外来的（即不针对算法本身的特点），实际上它们的威力非常强大，足以破解任何一个未经专门设计来对抗这两种解密方法的密码算法。

8.3 公开密钥算法

在历史上，分发密钥往往是绝大多数密码系统中最薄弱的环节。不管一个密码系统有多强，如果入侵者能够偷取到密钥，则整个系统就变得毫无价值。密码学家总是认为加密密钥和解密密钥是一样的（或者很容易从一个推导出另一个）。但是，这个密钥必须要被分发给一个系统的所有用户。因此，看起来这里存在一个固有的问题：密钥必须被保护起来，以防被偷，但是它们又必须要被分发出去，所以，它们不可能仅仅被锁在银行的保险柜中。

1976 年，斯坦福大学的两位研究人员 Diffie 和 Hellman 提出了一种全新的密码系统，在这种密码系统中，加密密钥和解密密钥并不相同，而且不可能很轻易地从加密密钥推导出解密密钥。在他们的提案中，（受密钥控制的）加密算法 E 和解密算法 D 必须满足 3 个要求。这 3 个要求可以简述如下：

（1）D(E(P)) = P。

（2）从 E 推断出 D 极其困难。

（3）E 不可能**被选择明文**（chosen plaintext）攻击破解。

第一个需求意味着，如果我们将 D 作用在一条被加密的消息 E(P)上，则可以恢复原来

的明文消息 P。如果没有这个特性，则合法的接收者就无法解密密文了。第二个需求不言自明。第三个需求是必要的，因为正如我们马上将会看到的那样，入侵者有可能用此算法对他们的核心内容进行试验。具备了上述这些条件以后，加密密钥就有理由可以被公开了。

这种方法的工作过程如下所述。有一个人，比如说 Alice，希望接收秘密消息，她首先设计了两个满足以上要求的算法。然后，加密算法和 Alice 的密钥都被公开，因此该系统被命名为**公开密钥密码系统**（public key cryptography）。例如，Alice 可以把公开密钥放在她的 Web 主页上。我们将使用标记 E_A 来表示以 Alice 的公开密钥作为参数的加密算法。类似地，以 Alice 的私有密钥作为参数的（秘密）解密算法被记为 D_A。Bob 也做同样的事情，即公开 E_B，但是 D_B 保密。

现在，我们来看是否能够在 Alice 和 Bob（以前两者从来没有联系过）之间建立一个安全的通道。假设 Alice 的加密密钥 E_A 和 Bob 的加密密钥 E_B 位于某些公开可读的文件中。现在，Alice 取出她的第一个消息 P，并计算 $E_B(P)$，然后将它发送给 Bob。Bob 接收到消息以后，利用他的秘密密钥 D_B 对消息进行解密（即计算 $D_B(E_B(P)) = P$）。其他没有人能够读取已被加密的消息 $E_B(P)$，因为假设加密系统足够强，并且从公开已知的 E_B 推导出 D_B 是极其困难的。为了发送一个应答消息 R，Bob 传输 $E_A(R)$。现在 Alice 和 Bob 就可以安全地进行通信了。

这里提一下关于术语的说明。公开密钥密码系统要求每个用户拥有两把密钥：一个公开密钥（简称公钥）和一个私有密钥（简称私钥）。当其他人给某个用户发送加密消息时，他们使用该用户的公钥；当这个用户需要解密消息时，他（她）使用自己的私钥。我们在下文中将统一地把这两个密钥分别称为公钥和私钥，从而与传统的对称密钥密码系统中用到的秘密密钥区分开。

8.3.1 RSA

现在唯一的问题是，我们需要找到满足上述所有 3 个要求的算法。由于公开密钥密码系统的潜在优势，许多研究人员不遗余力地工作，陆续发表了一些满足要求的算法。一种比较好的方法是由 MIT 的一个小组发现的（Rivest 等，1978），该方法的名字来源于 3 个发现者名字的首字母（Rivest、Shamir、Adleman）：RSA。尽管 30 多年来，有大量的工作企图破解该方法，但它都抵抗住了，所以它被认为是一个非常强的公开密钥算法。实际上有大量的安全保障都建立在它的基础之上。正是这个原因，Rivest、Shamir 和 Adleman 共同被授予了 2002 年的 ACM 图灵奖。它的主要缺点是要想达到好的安全性，密钥至少需要 1024 位（相比之下，对称密钥算法只需 128 位），这也使得它的运算速度非常慢。

RSA 方法基于数论中的一些原理。我们现在简要地说明如何使用该方法，有关详细的信息请参考 Rivest 等人的论文。

（1）选择两个大的素数 p 和 q（典型情况下为 1024 位）。

（2）计算 $n=p\times q$ 和 $z=(p-1)\times(q-1)$。

（3）选择一个与 z 互素的数，将它称为 d。

（4）找到 e，使其满足 $e\times d=1 \bmod z$。

提前计算出这些参数以后，我们就可以开始执行加密了。首先将明文（可以看作一个

位串）分成块，使得每个明文消息 P 落在间隔 $0 \leqslant P < n$ 中。为了做到这一点，只要将明文划分成 k 位的块即可，这里 k 是满足 $2^k < n$ 的最大整数。

为了加密消息 P，只要计算 $C=P^e \pmod n$即可。为了解密 C，只要计算 $P= C^d \pmod n$即可。可以证明，对于指定范围内的所有 P，加密和解密函数互为反函数。为了执行加密，你需要 e 和 n；为了执行解密，你需要 d 和 n。因此，公钥是由(e, n)对组成的，而私钥是由(d, n)对组成的。

这种方法的安全性建立在分解大数的难度基础之上。如果密码分析者能够分解 n（公开已知的），那么他就可以找到 p 和 q，从而得到 z。有了 z 和 e 之后，只要使用欧几里得算法就能找到 d。幸运的是，数学家们探索大数分解法至少有 300 多年了，几百年积累起来的经验表明这确实是一个极其困难的问题。

根据 Rivest 和同事们的研究工作，若使用穷举法，分解一个 500 位十进制数需要 10^{25} 年的时间。在两种情况下，他们假设使用了已知的最佳算法，并且计算机的指令时间为 1 微秒。使用 100 万个处理器并行计算，每个指令时间为 1 纳秒，仍然需要 10^{16} 年的时间。即使计算机的速度继续加快，以每 10 年一个数量级的速度增长，则仍然需要许多年才能使分解一个 500 位十进制数的方法变得实际可行，到那个时候，我们的后代们只要选择更大的 p 和 q 就可以了。

图 8-17 给出了一个简单的教学示范例子，它说明了 RSA 算法如何工作。在这个例子中，我们选择 p=3 和 q=11，因此 n=33 和 z=20。由于 7 和 20 没有公因子，所以 d 值取 d=7 是合适的。有了这些选择以后，通过解方程 $7e=1 \pmod{20}$可以找到 e，结果是 e=3。针对明文消息 P 的密文 C 可以通过 $C=P^3 \pmod{33}$求得。接收方在解密密文时，只要利用 $P=C^7 \pmod{33}$规则即可。图中的例子显示了明文“SUZANNE”的加密过程。

明文(P)			密文(C)		解密后	
符号	数值	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	符号
S	19	6859	28	13 492 928 512	19	S
U	21	9261	21	1 801 088 541	21	U
Z	26	17 576	20	1 280 000 000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78 125	14	N
N	14	2744	5	78 125	14	N
E	05	125	26	8 031 810 176	05	E
发送方计算				接收方计算		

图 8-17 RSA 算法的一个例子

因为这个例子中选择的素数太小，以至于 P 必须小于 33，因此每个明文块只能包含一个字符。其结果是一个单字母置换，让人觉得该方法并无稀奇之处。然而，如果我们选择 p 和 $q \approx 2^{512}$，则我们有 $n \approx 2^{1024}$，每个块就可以达到 1024 位，或者 128 个 8 位字节，相比之下，DES 的数据块是 8 个字符，AES 的块是 16 个字节。

应该指出，像以上描述的 RSA 用法有点类似于按 ECB 模式使用对称密钥算法的做法——即，同样的输入块得到同样的输出块。因此，我们需要使用某种链接形式来加密数据。然而，实际上大多数基于 RSA 的系统主要利用公开密钥算法来分发一次性会话密钥，再将这些会话密钥用于某一个对称密钥算法，比如 AES 或者三重 DES。若真要用 RSA 来

加密大量的数据，则运算速度太慢，因此RSA被广泛用于密钥的分发。

8.3.2 其他公开密钥算法

虽然RSA已经得到广泛使用，但它并不是唯一的已知公开密钥算法。第一个公开密钥算法是**背包算法**（Merkle和Hellman，1978）。背包算法的思想是：某个人拥有大量的物品，每个物品的重量各不相同。他秘密地选择其中一组物品并将这些物品放到一个背包中来编码一个消息。背包中物品的总重量被公开，所有可能的物品和它们对应的重量也被公开。但是，背包中物品的明细则是保密的。在特定的附加限制条件下，“根据给定的总重量找出可能的物品明细列表”这个问题被认为是一个计算上不可行的问题，从而构成了此公开密钥算法的基础。

算法的发明者Ralph Merkle非常确信这个算法不可能被攻破，所以他悬赏100美元奖金给破解算法的人。Adi Shamir（即RSA中的“S”）迅速地破解了算法，并领走了奖金。Merkle并没有气馁，他又加强了算法，并悬赏1000美元奖金给破解新算法的人。Ronald Rivest（即RSA中的“R”）也迅速地破解了新算法，并领取了奖金。Merkle不敢再为下一个版本悬赏10 000美元奖金了，所以“A”（Leonard Adleman）就无法那样幸运也有机会破解它。不管怎么样，背包算法已经不再被认为是安全的，实际上也没有被采用。

其他一些公开密钥方案建立在计算离散对数的难度基础之上。（El Gamal，1985）和（Schnorr，1991）发明了使用这种原理的公开密钥算法。

另外还存在一些方案，比如基于椭圆曲线的公开密钥算法（Menezes和Vanstone，1993），但是最主要的两大类算法是：建立在“分解大数的难度”基础上的算法，以及建立在“以大素数为模来计算离散对数的难度”基础上的算法。这些问题被认为的确很难解决，因为数学家们已经为之钻研了许多年而未能有所突破。

8.4 数字签名

许多法律的、金融的和其他文档的真实性是依据是否具有某一个授权人物的手写签名。影印件是无效的。为了能够用计算机化的消息系统来代替纸和笔墨文档的物理传输系统，必须要找到一种方法对文档进行签名，并且文档的签名不可被伪造。

设计一个代替手写签名的方案非常困难。基本上，我们需要的是这样一个系统，其中一方向另一方发送的签名消息必须满足以下条件：

（1）接收方可以验证发送方所声称的身份。

（2）发送方以后不能否认该消息的内容。

（3）接收方不可能自己编造这样的消息。

第一个要求是必需的，例如在金融系统中，当顾客的计算机向银行的计算机订购一吨黄金时，银行的计算机必须要确信发出订单的计算机真正属于付款账号的所属公司。换句话说，银行必须要认证顾客的身份（顾客也要认证银行的身份）。

第二个要求也是必需的，它可以保护银行不被欺骗。假设银行买进了一吨黄金，之后

黄金的价格立刻跌了下来。一个不诚实的顾客可以起诉银行，宣称自己从来没有发出过购买黄金的订单。当银行在法庭上出示这条消息的时候，顾客否认自己曾经发送过此消息。这种“契约的任何一方以后不能否认自己曾经签过名”的特性称为**不可否认性**（nonrepudiation）。我们现在将要学习的数字签名方案可以用来提供这样的特性。

第三个要求也是必需的，它可以保护顾客的利益不受损害，例如，当黄金的价格暴涨时，银行企图构造一个“顾客要求购买一条黄金（而不是一吨黄金）”的签名消息。在这种欺骗的情形下，银行就可以保留剩余的黄金。

8.4.1 对称密钥签名

数字签名的一种做法是设立一个人人都信任并且又熟知一切的中心权威机构，比如Big Brother（简称BB）。每个用户选择一个秘密密钥，并且亲手将它送到BB的办公室。因此，只有Alice和BB才知道Alice的秘密密钥K_A，如此等等。

当Alice想要给她的银行业务员Bob发送一条签名的明文消息P时，她生成$K_A(B, R_A, t, P)$，这里B是Bob的标识，R_A是Alice选择的一个随机数，t是一个时间戳（可用来保证该消息是最新的），$K_A(B, R_A, t, P)$表示用她的密钥K_A加密之后的消息。然后，她按照图8-18所示的方式将该消息发送出去。BB看到该消息来自Alice，于是解密该消息，并按图中所示给Bob发送一条消息。给Bob的消息包含了Alice的原始消息的明文P和一条经过签名的消息$K_{BB}(A, t, P)$。Bob现在执行Alice的请求。

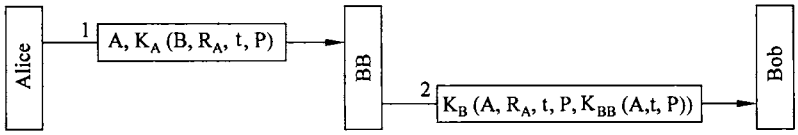


图8-18 使用BB的数字签名

如果Alice后来拒绝承认自己曾经发送过该消息，那该怎么办呢？首先，人人都可以起诉别人（至少在美国是这样的）。最后，当案子到了法庭上，Alice顽固地否认自己曾经给Bob发送过这条有争议的消息时，法官会问Bob，他如何能保证这条有争议的消息确实来自Alice而不是Trudy。Bob首先指出，BB在看到来自Alice的消息时，除非该消息是用K_A加密的，否则BB是不会接受该消息的，所以，Trudy向BB发送一条谎称来自Alice的消息却没有被BB检测到，这是不可能的。

然后Bob在众目睽睽下出示了证据A：$K_{BB}(A, t, P)$。Bob说，这是一条由BB签名的消息，它可以证明Alice曾经发送P给Bob。然后，法官请BB（人人都信任BB）解密证据A。当BB证实了Bob说的是真话时，法官做出有利于Bob的判决。案子最终结束。

在图8-18的签名协议中，有个潜在的问题，那就是Trudy可能会重放其中某一条消息。为了使这个问题的危险性降低到最小，可以在每一个环节上用时间戳进行控制。而且，Bob可以检查所有最近接收到的消息，查看这些消息是否使用过R_A。如果确有消息用到了R_A，则他可以将新收到的消息当作重放消息而丢弃。请注意，利用时间戳机制，Bob将拒绝接收那些很老的消息。为了对付短时间内的重放攻击，Bob只要检查每条入境消息中的R_A，以此判断在过去的一小时内是否曾经收到过来自Alice的这条消息。如果没有，则Bob就

可以安全地认为这是一个新的请求。

8.4.2 公开密钥签名

利用对称密钥密码技术来实现数字签名存在一个结构性问题：每个人都必须同意 BB 是可相信的。而且，BB 能够解读所有签名的消息。从逻辑来看，最有可能运行 BB 服务器的候选机构是政府、银行、会计事务所和律师事务所。不幸的是，老百姓对所有这些组织并不能寄予足够的信任。因此，若能在签名文档的时候不要求通过一个可信的权威机构就好了。

幸运的是，公开密钥密码学为这个领域做出了重要的贡献。我们假设公开密钥的加密算法和解密算法除了具有常规的 D(E(P)) = P 属性以外，还具有 E(D(P)) = P 属性（RSA 有这样的属性，所以这个假设并非不合理）。假设这种情况是成立的，那么 Alice 可以通过传输 $E_B(D_A(P))$向 Bob 发送一条签名的明文消息 P。请注意，Alice 知道她自己的（私有）密钥 D_A 以及 Bob 的公开密钥 E_B，所以，Alice 能够这样做。

当 Bob 收到这条消息时，他像往常一样利用自己的私钥对消息做替代，从而得到 $D_A(P)$，如图 8-19 所示。他将这份信息放在一个安全的地方，然后通过使用 E_A 可得到原始的明文。

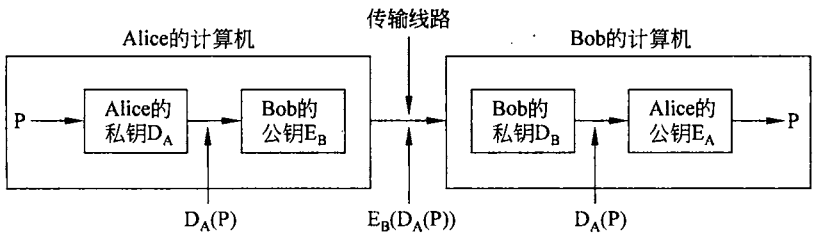


图 8-19 使用公开密钥密码数字签名

为了看清这种签名特性的工作原理，不妨假设 Alice 后来否认自己曾经给 Bob 发送过消息 P。当这个案子被提到法庭上时，Bob 可以同时出示 P 和 D_A (P)。法官很容易验证 Bob 是否真的拥有一条由 D_A 加密的消息，他只需简单地在消息上应用 E_A 即可。由于 Bob 并不知道 Alice 的私钥是什么，所以，Bob 能获得由 Alice 私钥加密的消息的唯一途径是 Alice 给他发送了这样的消息。当 Alice 因为作伪证和欺骗罪而被监禁起来时，她有足够的时间来设计新的有趣的公开密钥算法。

尽管用公开密钥密码技术实现数字签名是一种非常优雅的方案，但是，这里还存在一些涉及运行环境的问题，与基本算法无关。首先，只要 D_A 仍然是保密的，Bob 就可以证明一条消息确实是 Alice 发送的。如果 Alice 泄漏了她的密钥，那么这个论点就不再成立，因为任何人都可以发送这样的消息，包括 Bob 自己在内。

这时候就可能出现这样的问题，如果 Bob 是 Alice 的股票经纪人，假设 Alice 告诉 Bob 购买一支特定的股票或者国库券。很快地，股票价格急剧下降。Alice 为了否认她曾经给 Bob 发送过消息，她跑到警察局声称她家遭到抢劫，保存密钥的个人电脑被偷了。根据她所在州或者国家的法律，她可能需要，也可能不需要承担法律责任，尤其是如果她声称是在下班以后回到家里时才发现被抢了（即发生抢劫几小时以后才发现）。

与这种签名方案有关的另一个问题是，如果 Alice 决定要改变她的密钥，那该怎么办

呢？很显然，这样做是合法的，而且定期改变密钥可能是一种很好的做法。如果后来发生了法律纠纷，就像上面描述的那样，那么，法官将当前的 E_A 作用在 $D_A(P)$上，发现其结果并不等于 P。这时候 Bob 就会很难堪。

原则上，任何一种公开密钥算法都可以用作数字签名。事实上的业界标准是 RSA 算法。许多安全产品都使用了 RSA 算法。然而，1991 年，NIST 建议使用 El Gamal 公开密钥算法的一个变种作为它们新的**数字签名标准**（DSS，Digital Signature Standard）。El Gamal 算法的安全性建立在计算离散对数的难度基础之上，而并非建立在分解大数的困难度基础上。

通常情况下，当政府试图推行一个密码学标准时，它总会招致各种非议。DSS 也受到了下述批评：

（1）太神秘（这是 NSA 设计的协议，它使用了 El Gamal 算法）。

（2）太慢（在检查签名时比 RSA 慢了 10～40 倍）。

（3）太新（El Gamal 算法还没有被完全分析透彻）。

（4）太不安全（固定的 512 位密钥）。

在后来的修订版本中，允许密钥长度增加到 1024 位，因此第（4）点已不成问题，然而前两点批评还是没有改变。

8.4.3 消息摘要

对签名方法的一个批评是，它们通常将两种不相同的功能耦合在一起：认证和保密。通常情况下，认证是必要的，但是保密性并不一定是必需的。而且，如果一个系统只提供认证而没有提供保密性的话，往往更加容易申请出口许可。下面介绍具备认证不要求加密整条消息的方案。

这个方案以单向散列函数思想为基础，这里的单向散列函数接受一个任意长度的明文作为输入，并且根据此明文计算出一个固定长度的位串。这个散列函数 MD 通常称为**消息摘要**（message digest），它有 4 个重要的特性：

（1）给定 P，很容易计算 MD(P)。

（2）给定 MD（P），要想有效地找到 P 是不可能的。

（3）在给定 P 的情况下，没有人能够找到满足 MD (P′) = MD(P)的 P′。

（4）在输入明文中即使只有 1 位的变化，也会导致完全不同的输出。

为了满足第 3 条，散列结果应该至少 128 位，最好还能更长一些。为了满足第 4 条，散列结果必须彻底弄乱明文中的位，和我们在对称密钥加密算法中看到的没有什么不同。

从一段明文计算出一个消息摘要必须比用公开密钥算法来加密这段明文要快得多，所以消息摘要可以被用来加速数字签名算法。为了看清楚这个工作过程，请再次考虑图 8-18 的签名协议。BB 现在不再利用 $K_{BB}(A, t, P)$作为 P 的签名，而是计算消息摘要，他将 MD 作用在 P 上，得到 MD(P)。然后 BB 用 $K_{BB}(A, t, MD(P))$代替原来的 $K_{BB}(A, t, P)$，作为他发送给 Bob 的消息（被 K_B 加密）中的第 5 项。

如果发生纠纷，Bob 可以出示 P 和 $K_{BB}(A, t, MD(P))$。当 BB 根据法官的要求将后者解密出来后，Bob 有了 MD(P)，以及他所宣称的 P。这里 MD(P)保证是真实的。然而，由于 Bob 不可能有效地找到另一条具有相同散列结果的消息，所以法官很容易就可以确信 Bob

讲的是事实。按照这种方式来使用消息摘要可以节省加密时间和消息传输的开销。

消息摘要也可以用在公开密钥密码系统中，如图 8-20 所示。在这里，Alice 首先计算明文的消息摘要，然后她针对消息摘要进行签名，并且将签名之后的摘要与明文本身一起发送给 Bob。如果 Trudy 中途偷换掉 P，那么当 Bob 计算 MD(P)时就可以发现这一点。

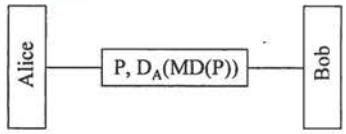


图 8-20 使用消息摘要的数字签名

SHA-1 和 SHA-2

各种消息摘要函数被提了出来。一个被广泛应用的消息摘要函数是**安全散列算法 1**（SHA-1，Secure Hash Algorithm 1）（NIST，1993）。像所有的消息摘要一样，它使用非常复杂的方法来弄乱位，以至于它的每个输出位都受到输入位的影响。SHA-1 由 NSA 开发并得到 NIST 的赏识，并被标准化在 FIPS 180-1 中。它按照 512 位的块大小来处理输入数据，并且生成一个 160 位的消息摘要。图 8-21 所示为一种典型的使用方式，Alice 用来发给 Bob 非保密性、但有她签名的消息。在此，她的明文消息被送入到 SHA-1 算法中，以便得到一个 160 位的 SHA-1 散列值。然后，她用自己的 RSA 私钥对散列值进行签名，并且将明文消息和签过名的散列值发送给 Bob。

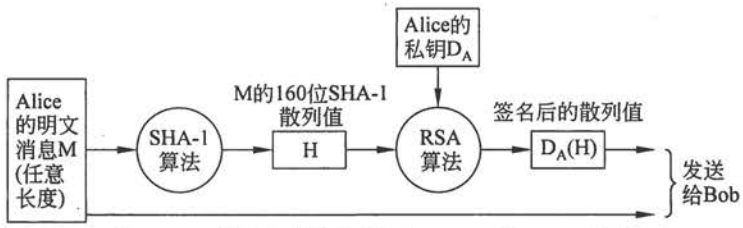


图 8-21 非保密性消息使用 SHA-1 和 RSA 签名

Bob 在接收到消息以后，他自己计算 SHA-1 散列值，并且利用 Alice 的公钥来解密签过名的散列值，从而得到原始的散列值 H。如果这两者一致的话，则认为该消息是有效的。因为对于 Trudy 来说，她无法在传输过程中既要修改（明文）消息，也要使修改之后的消息散列到同一个 H 上，所以，Bob 可以很容易地检测到 Trudy 对于消息 P 所做的任何改变。对于那些完整性很重要但内容并不需要保密的消息，广泛使用了图 8-21 所示的方案。那些计算量相对较小的计算，采用这种方案保证了在传输过程中对明文消息所做的任何修改都将（以极高的概率）被检测到。

现在我们大致看一下 SHA-1 的工作原理。首先它在消息的末尾填补一个 1 位，然后跟上一定数量的 0 位，0 位的个数必须保证填补之后的消息长度为 512 位的倍数。然后构造一个包含了填补之前消息长度的 64 位整数，并将它与消息的低 64 位进行或（OR）操作。在图 8-22 中，填补的位被显示在消息的右侧，因为英语文本和图片按照从左至右的顺序来书写或绘制（即右下角往往被认为是图片的尾部）。对于计算机来说，这种方向特性对应于 big-endian 字节序的机器，比如 SPARC，但是，SHA-1 总是填补在消息的末尾，而不管当前机器使用了哪一种字节序。

在计算过程中，SHA-1 维护了 5 个 32 位变量，从 H_0 至 H_4，最后散列的结果就放在这些变量中。图 8-22(b)显示了这些变量。它们的初始常量由标准定义。

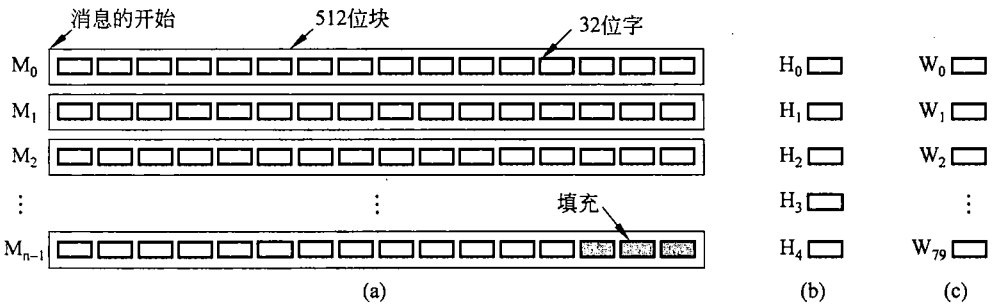


图 8-22

(a) 消息被填补成 512 位的倍数；(b) 输出变量；(c) 字数组

现在按照顺序逐个处理数据块 M_0 至 M_{n-1}。对于当前的块，首先将 16 个字复制到一个如图 8-22(c)所示的 80 字的辅助数组 W 的前面部分。然后，利用下面的公式来填充 W 中其他的 64 个字：

$$W_i = S^1(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \quad (16 \leqslant i \leqslant 79)$$

在这里 S^b (W)代表了 32 位字 W 循环左移 b 位。现在，利用 H_0 至 H_4 分别初始化 5 个临时变量（从 A 至 E）。

实际的计算过程可以用下面的伪 C 代码来表达：

```
for (i=0;i<80; i++){
   temp=S^5(A)+f_i(B,C,D)+E+W_i+K_i;
   E=D; D=C; C=S^30(B); B=A; A=temp;
}
```

在此常量 K_i 由标准定义，而混合函数 f_i 的定义如下：

f_i(B, C, D)=(B AND C) OR (NOT B AND D)　(0≤i≤19)

f_i(B, C, D)=B XOR C XOR D　(20≤i≤39)

f_i(B, C, D)=(B AND C) OR (B AND D) OR (C AND D)　(40≤i≤59)

f_i(B, C, D)=B XOR C XOR D　(60≤i≤79)

当循环中的 80 次迭代完成以后，A 至 E 被分别加到 H_0 至 H_4 上。

现在第一个 512 位的块处理完毕，下一个块又接着开始处理。W 数组又被按照新的块进行初始化，但是，H 数组仍然不变。当这一块完成以后，下一块又开始，如此周而复始，直到所有的 512 位消息块都被处理完毕。当最后一块被处理完以后，H 数组中的 5 个 32 位字即被输出作为 160 位的密码学散列值。RFC 3174 给出了 SHA-1 算法的完整 C 代码。

一些生成 256、384 和 512 位散列值的 SHA-1 新版本已经被开发出来。这些版本统称为 SHA-2。新版本不仅散列值比 SHA-1 散列值长，而且摘要函数也得到修改，弥补了一些 SHA-1 的潜在弱点。目前 SHA-2 还没有被广泛使用，但是它未来的前景很被看好。

MD5

为了完整性，我们介绍另一个很普及的消息摘要算法 **MD5**（Rivest，1992）。它由 Ronald Rivest 设计，在他设计的一系列消息摘要算法中排在第五位。简单地说，消息被填补到一个 448 位长（modulo512），然后附加 64 位整数的原始消息长度，使得总输入长度是 512

位的倍数。每一轮的计算都用一个正在运行的128位缓冲区，完全地混合一个512位的输入块。为何便于测量，混合时使用了一个用正弦函数构建的表。使用已知函数的目的是为了避免让人怀疑设计师留下一个巧妙但只有他们可以进入的后门。这个过程一直进行到所有的输入块都已经被用完。128位缓冲区里的内容就构成了消息摘要。

经过了超过10年的实际使用和研究后，MD5的弱点导致人们能够找到具有相同散列的冲突或不同的消息（Sotirov 等，2008）。这对于消息摘要来说是个丧钟，因为它意味着摘要不能安全地用来代表消息。因此，安全社团认为MD5将被破解，在可能的情况下应该换掉它，而且新的系统在设计时不应该再使用它。不过，你可能会看到一些已有系统仍然在使用MD5。

8.4.4 生日攻击

在密码学领域，没有一件事情会像表面上看到的那么简单。有人可能会认为，为了攻破一个m位的消息摘要，将需要2^m数量级的操作次数。事实上，（Yuval，1979）在他的论文“How to Swindle Rabin”中发表了一种生日攻击方法，用这种方法通常只需要$2^{m/2}$量级的操作次数，现在他的这篇论文已经成为经典。

这种攻击的思想来源于数学教授们通常在他们的概率课堂上采用的一种技术。所提的问题是：一个班级中有多少个学生时，使得这个班级中两个学生具有相同生日的概率超过1/2？大多数学生都认为答案将会超过100。事实上，根据概率论，答案只是23。如果不给出严密的分析，仅仅从直觉上，我们可以看到，在有23个学生的情况下，我们可以组成(23×22)/2=253对不同的组合，每一对组合有1/365的命中概率。照这样看来，这个答案实际上就不那么令人惊奇了。

推广至更为一般的情形，如果在n个输入（人、消息等）和k个可能的输出（生日、消息摘要等）之间存在某一种从输入到输出的映射关系，那么就有n(n-1)/2个输入对。若n(n-1)/2>k，则至少有一个匹配的机会是非常大的。因此，近似地，存在一个匹配的条件是$n>\sqrt{k}$。这个结果意味着只要生成大约2^{32}条消息，然后在这些消息中寻找具有相同消息摘要的两条消息就有可能攻破一个64位的消息摘要。

现在我们来看一个实际的例子。州立大学的计算机科学系有一个终生教员的职位，并且有两个候选人：Tom和Dick。由于Tom比Dick早雇用两年，所以他是首先被考察的对象。如果他通过了，则Dick就没有机会。Tom知道系行政主管Marilyn非常欣赏他的工作，所以他请Marilyn为他写一封推荐信给系主任，因为最终的决定权在系主任手中。一旦发出去之后，所有的信件就都是保密的。

Marilyn告诉她的秘书Ellen给系主任写封信，并大略地描述了她在信中要表达的内容。当Ellen写完信后，Marilyn将会检查一遍，并计算和签署64位摘要，然后发送给系主任。Ellen可以稍后通过电子邮件发送这封信。

对于Tom来说很不幸的是，Ellen正迷恋着Dick，因而她想要陷害Tom，于是，她写了下面这封具有32个用括号括起选项的信：

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Prof. Wilson for [*about* | *almost*] six years. He is an [*outstanding* | *excellent*] researcher of great [*talent* | *ability*] known [*worldwide* | *internationally*] for his [*brilliant* | *creative*] insights into [*many* | *a wide variety of*] [*difficult* | *challenging*] problems.

He is also a [*highly* | *greatly*] [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*rave* | *spectacular*] reviews. He is [*our* | *the Department's*] [*most popular* | *bestloved*] [*teacher* | *instructor*].

[*In addition* | *Additionally*] Prof. Wilson is a [*gifted* | *effective*] fund raiser. His [*grants* | *contracts*] have brought a [*large* | *substantial*] amount of money into [*the* | *our*] Department. [*This money has* | *These funds have*] [*enabled* | *permitted*] us to [*pursue* | *carry out*] many [*special* | *important*] programs, [*such as* | *for example*] your State 2000 program. Without these funds we would [*be unable* | *not be able*] to continue this program, which is so [*important* | *essential*] to both of us.

I strongly urge you to grant him tenure.

对于 Tom 来说，不幸的是 Ellen 撰写并输入这封信后，她还马上写了第二封信：

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Tom for [*about* | *almost*] six years. He is a [*poor* | *weak*] researcher not well known in his [*field* | *area*]. His research [*hardly ever* | *rarely*] shows [*insight in* | *understanding of*] the [*key* | *major*] problems of [*the* | *our*] day.

Furthermore, he is not a [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*poor* | *bad*] reviews. He is [*our* | *the Department's*] least popular [*teacher* | *instructor*], known [*mostly* | *primarily*] within [*the* | *our*] Department for his [*tendency* | *propensity*] to [*ridicule* | *embarrass*] students [*foolish* | *imprudent*] enough to ask questions in his classes.

[*In addition* | *Additionally*] Tom is a [*poor* | *marginal*] fund raiser. His [*grants* | *contracts*] have brought only a [*meager* | *insignificant*] amount of money into [*the* | *our*] Department. Unless new [*money is* | *funds are*] quickly located, we may have to cancel some essential programs, such as your State 2000 program. Unfortunately, under these [*conditions* | *circumstances*] I cannot in good [*conscience* | *faith*] recommend him to you for [*tenure* | *a permanent position*].

现在 Ellen 利用她的计算机彻夜地计算出每封信的 2^{32} 个消息摘要。第一封信的消息摘要有可能跟第二封信的消息摘要匹配。如果没有的话，她可以增加一些选项然后今晚再尝

试一次。现在假设她找到了一个匹配。我们将“好的”信称为A，“坏的”信称为B。

Ellen现在以电子邮件的方式将信A发送给Marilyn审核。同时将信B完全保密，不让任何人看到。Marilyn当然批准了它，计算她的64位消息摘要，并且签了摘要，然后将签过名的摘要发送给系主任Smith。而Ellen则单独地将信B以电子邮件方式寄送给系主任（请注意，她寄送的并不是Marilyn所想象的信A）。

系主任在得到了信和签过名的消息摘要后，他针对信B运行消息摘要算法，发现与Marilyn送给他的摘要一致，于是解雇了Tom。系主任并没有意识到Ellen生成了两封具有相同消息摘要的信，并且发送给他的信与Marilyn看到并批准的信不是同一封（一个选择性的结尾是：Ellen将自己所做的一切全部告诉了Dick。Dick非常震惊，并与她断绝了关系。Ellen也非常气愤，向Marilyn坦白了一切。Marilyn打电话告诉了系主任，最终Tom获得了职位）。使用SHA-1，生日攻击很难实施，因为即使每秒钟产生10亿个摘要，那也需要花32 000年来计算两封信的所有2^{80}个摘要（每封信有80种变化形式），而且即使这样还不能保证总会存在一个匹配。当然，如果使用1 000 000个处理器的云处理，则计算时间由32 000年变成了2个星期。

8.5 公钥的管理

公钥密码学使得人们有可能在不共享公共密钥的情况下仍然可以安全地进行通信。它也使人们有可能在不存在可信第三方的情况下为消息做签名。最后，利用签过名的消息摘要，接收方能够很容易地验证自己所接收到的消息的完整性。

然而，请等一等，我们这里还掩盖了一个问题：如果Alice和Bob彼此之间并不认识，那么，他们如何获得对方的公钥来开始通信呢？一种很容易想到的解决方案是将公钥放在自己的Web站点上，但是这种方案并不能工作，理由如下所述。假设Alice想要在Bob的Web站点寻找他的公钥，那么她该怎么办呢？首先，通过Bob的URL，她用浏览器找到Bob主页的DNS地址，然后向该地址发送一个GET请求，如图8-23所示。不幸的是，Trudy截获了这个请求，并且将一个伪造的主页发回给Alice作为应答，这个伪造的主页可能是Bob主页内容的一份副本，只不过其中Bob的公钥被替换成Trudy的公钥。当Alice现在用E_T加密她的第一条消息时，Trudy解密并阅读该消息，然后用Bob的公钥重新进行加密，再发送给Bob，而Bob根本不知道Trudy已经阅读了他所接收到的消息。更糟的是，Trudy在为Bob重新加密消息之前还可以对消息做一些修改。很明显需要有某种机制来确保公钥交换的安全性。

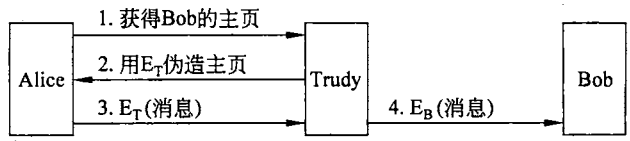


图8-23 Trudy颠覆公钥加密的一种方式

8.5.1 证书

为了安全地分发公钥，第一种尝试的方法是我们想象有一个**密钥分发中心**（KDC，key distribution center）一天 24 小时地根据客户的需求提供在线公钥服务。这种方案有很多问题，其中一个问题是它的扩展性差，密钥分发中心将很快成为瓶颈。而且，如果它停止工作，则 Internet 安全性也将突然停顿下来。

由于这些原因，人们已经开发出了另一种不同的方案，这种方案并不要求密钥分发中心始终不停地保持在线。事实上，它并不需要时时在线。相反，它所要做的事情是证明每个公钥属于个人、公司或者其他组织。这种证明公钥所属权的组织现在称为**认证中心**（CA，Certification Authority）。

作为一个例子，假设 Bob 希望 Alice 和其他不认识的人能够安全地与他进行通信。他可以带上护照或者驾驶证到 CA 那里，请求认证他的公钥。然后，CA 给他颁发一个证书，其中的内容类似于图 8-24 所示的证书，而且 CA 用自己的私钥对证书的 SHA-1 散列值进行签名。然后，Bob 付给 CA 一定的费用，并获得一张包含了证书和签过名的散列值的光盘。

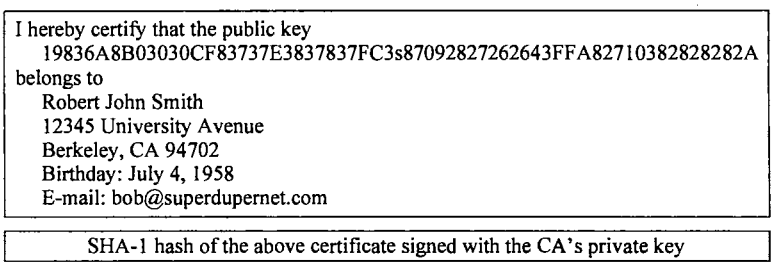


图 8-24 一个可能的证书和它的签名散列

证书的基本任务是将一个公钥与安全个体（个人、公司等）的名字绑定在一起。证书本身并不是保密的，也没有被保护。例如，Bob 可能决定将他的新证书放到 Web 站点上，他的做法是在主页上使用这样一个链接："点击这里可得到我的公钥证书"。当用户单击该链接之后，就可以得到 Bob 的证书和签名块（即该证书经过签名之后的 SHA-1 散列值）。

现在我们再来看图 8-23 的场景。当 Trudy 截获了 Alice 请求 Bob 主页的命令时，她该怎么办呢？她可以把她自己的证书和签名块放到伪造的页面上，但是当 Alice 读取证书时，她立即就会发现那不是来自于 Bob，因为 Bob 的名字不在证书中。Trudy 可以临时修改 Bob 的主页，用她自己的公钥来替换 Bob 的公钥。然而，当 Alice 对证书运行 SHA-1 算法时，她得到的散列值与她将 CA 的公钥（这是众所皆知的）应用在签名块上所得到的散列值不一致。由于 Trudy 拿不到 CA 的私钥，所以她无法生成一个包含她公钥的签名块，也就无法构造出让 Alice 相信她是 Bob 的 Web 页面。通过这种方式，Alice 可以确信她得到的是 Bob 的公钥，而不是 Trudy 或者其他人的公钥。正如前面我们所承诺的，这种方案不要求 CA 一直提供在线验证服务，从而消除了潜在的瓶颈问题。

虽然证书的标准功能是将一个公钥绑定到一个安全个体上，但证书也可以被用来将一个公钥绑定到一个**属性**（attribute）上。例如，一个公钥可以表达这样的含义：该公钥属于

某一个年龄超过18岁的人。这样的公钥可以被用来证明私钥的所有者是个成年人，从而被允许访问少儿不宜的资料等而不需要泄漏所有者的身份。典型情况下，持有证书的人往往会把这个证书发送给网站、其他的安全个体，或者对年龄较为敏感的进程等。网站、安全个体或进程将生成一个随机数，并用证书中的公钥对随机数进行加密。如果所有者能够解密随机数并发送回来，那将证明所有者确实具备了该证书中所声明的属性。另外，此随机数也可以被用来生成一个会话密钥，以保证后续通话的安全。

在证书中可能包含属性的另一个例子是用在面向对象的分布式系统中。每个对象通常有多个方法。对象的所有者可以向每个顾客提供一个证书，在证书中给出一个位图来表明允许该顾客调用哪些方法，并且用一个签过名的证书将位图与一个公钥绑定在一起。同样地，如果证书持有者可以证明自己确实拥有相应的私钥，那么，他将能够执行位图中指定的方法。这个方法的特点是它不需要暴露拥有者的身份。这在个人隐私很重要的情形下非常有用。

8.5.2 X.509

如果每一个想要为某些资料做签名的人都到CA那里申请各种不同类型的证书，那么，管理所有不同格式的证书很快就成为一个问题。为了解决这个问题，ITU已经设计并批准了一个专门针对证书格式的标准。该标准称为**X.509**，现在它已经广泛应用于Internet上。自从1988年首次被标准化以来，它经历了三个版本。下面我们将讨论第3版（V3）。

X.509深受OSI的影响，它借用了OSI领域中某些非常糟糕的特性（比如命名和编码）。然而，令人惊奇的是，在几乎所有其他的领域中，从机器地址到传输协议，再到电子邮件格式，IETF一般都忽略OSI，而努力按照自己的方式来做得更好，但是在证书格式上，IETF却采纳了X.509。X.509的IETF版本由RFC 5280来描述。

X.509的核心是提供一种描述证书的格式。图8-25列出了证书中的主要字段。这里给出的描述仅仅说明了每个字段的一般性用途。有关更多的信息，请参考标准本身或者RFC 2459。

字段	含义
Version	X.509的哪个版本
Serial number	此数值加上CA的名字唯一标识了证书
Signature algorithm	用来签名证书的算法
Issuer	CA的X.500名字
Validity period	有效期的开始和终止时间
Subject name	哪个实体的密钥得到了验证
Public key	主题的公钥和所用算法的ID
Issuer ID	可选的ID，唯一标识了证书颁发者
Subject ID	可选的ID，唯一标识了证书主题
Extensions	已经定义了多少扩展字段
Signature	证书的签名（用CA的私钥加密）

图8-25 X.509证书的基本字段

例如，如果Bob在Money银行的信贷部门工作，那么，他的X.500地址可能是这样的：

/C=US/O=MoneyBank/OU=Loan/CN=Bob/

其中C代表国家，O代表组织，OU代表组织单位，CN是一个普通名字。CA和其他

的实体也用类似的方式来命名。关于 X.500 名字的一个实质性问题是，如果 Alice 试图通过 bob@moneybank.com 与 Bob 联系，而她拿到的证书中却是 X.500 名字，那么，对于 Alice 来说，该证书是否引用了她想要的那个 Bob 却并不那么显而易见。幸运的是，从第 3 版开始，它也允许使用 DNS 名字来代替 X.500 名字，所以，这个问题最终可能被消除了。

证书的编码使用了 OSI 的**抽象语法标记 1**（ASN.1，Abstract Syntax Notation 1），你可以将这种编码想象成与 C 语言结构类似的描述形式，除此以外它也包含一些非常特殊和细微的标记。有关 X.509 更多的信息，请参考（Ford 和 Baum，2000）。

8.5.3 公钥基础设施

由一个 CA 来颁发全世界所有的证书显然是不切实际的。它将会不堪重负，并且成为一个中心故障点。一种可能的解决方案是使用多个 CA，并且所有的 CA 由同一个组织来运行，它们使用同一个私钥来签名证书。虽然这样可以解决负载太重和单故障点的问题，但是它又引入了新的问题：密钥泄漏。如果有几十台服务器遍布在世界各地，并且所有的服务器都持有 CA 的私钥，那么私钥被偷窃或者泄漏的机会就会明显地增加。由于该私钥的暴露将会影响到全世界的电子安全基础设施，所以，采用一个中心 CA 的做法是非常危险的。

另外，由哪一个组织负责运行这个 CA 呢？很难想象有哪一个权威机构能在全球范围内被普遍接受和信任。在有些国家中，人们坚持这个组织应该是一个政府机构，而在其他的国家中，人们又坚持这个组织不应该属于一个政府机构。

基于这些理由，人们又发展了另一种证明公钥身份的方法。它的通用名称是**公开密钥基础设施**（PKI，Public Key Infrastructure）。在这一小节中，我们将一般性地概述一下 PKI 的工作原理，不过，由于有许多关于 PKI 的提案和建议，所以有些细节将来可能会发生变化。

一个 PKI 有多个组件，包括用户、CA、证书和目录。PKI 所做的事情是提供一种方法将这些组件有序地组织起来，并且定义了各种文档和协议的标准。一种特别简单的 PKI 形式是 CA 层次，如图 8-26 所示。在这个例子中，我们显示了三个层次，但实际上，层次的数目可能更少，也可能更多。最顶级的 CA，即层次的根，它的责任是证明第二级的 CA，这种 CA 我们称为**区域管理机构**（RA，Regional Authority），因为它们可能覆盖某一个地理区域，比如一个国家或者一个洲。然而，这个术语不是标准的；事实上，PKI 并没有定义标准的术语来表达树结构的不同层次。这些区域管理机构又依次证明下一级 CA 的真实性；在图中所示的 PKI 结构中，这些下级 CA 才真正为组织和个人颁发 X.509 证书。当根 CA 授权一个新的 RA 时，它生成一个 X.509 证书，并且在证书中声明它已经批准了这个 RA，同时将新 RA 的公钥也包含在这个证书中；根 CA 为证书签过名以后，将它交给该 RA。类似地，当 RA 批准一个新 CA 时，它产生一个证书并为其签名；该证书声明了它已经批准该 CA，并包含 CA 的公钥。

我们的 PKI 按照如下方式工作。假设 Alice 需要 Bob 的公钥以便与他通信，她搜寻并找到了一个包含 Bob 公钥的证书，该证书已由 CA 5 签过名。但是，Alice 从来没有听说过 CA 5。她所知道的是 CA 5 也许是 Bob 的 10 岁的女儿。她可以到 CA 5 那里说：请证明你

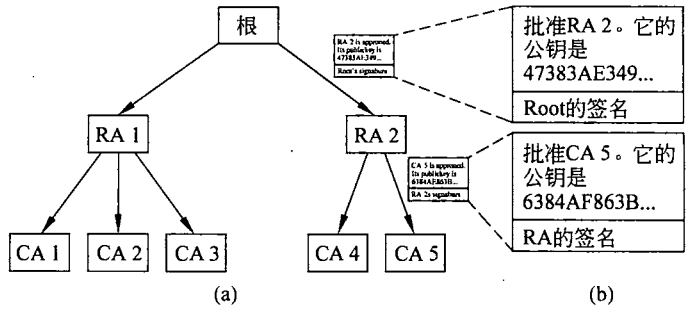


图 8-26
(a) 一个层次 PKI；(b) 一个证书链

的合法性。CA 5 用它从 RA 2 那里获得的证书作为响应，该证书包含了 CA 5 的公钥。现在 Alice 拿到了 CA 5 的公钥，所以她可以验证 Bob 的证书确实是由 CA 5 签名的，因此是合法的。

除非 RA 2 是 Bob 的 12 岁儿子，否则接下来的步骤是，Alice 请 RA 2 证明它的合法性。RA 2 对此的应答是一个由根 CA 签名的证书，其中包含了 RA 2 的公钥。现在 Alice 可以确信她真正拿到了 Bob 的公钥。

但是，Alice 如何找到根 CA 的公钥呢？这正是 PKI 的奥妙所在。它假设每个人都知道根 CA 的公钥。例如，她的浏览器可能在发行的时候已经预装了根的公钥。

Bob 是一个非常友好的合作伙伴，他不想给 Alice 增加太多额外的工作。他知道她将要检查 CA 5 和 RA 2 的合法性，所以，为了省去她的麻烦，他把两个必要的证书收集起来，并将这两个证书连同他自己的证书一起交给 Alice。现在，Alice 可以使用她自己所了解到的根证书来验证顶级证书和其中所包含的公钥，进而再验证第二级证书。按照这种方法，Alice 不用跟任何人联系就可以完成验证工作。因为证书全都是经过签名的，所以她可以检测出所有对证书内容的篡改行为。像这样由底向上回溯到根的证书链有时候称为**信任链**（chain of trust），或者**证书路径**（certification path）。这项技术已经得到广泛的实际应用。

当然，我们还有另一个问题，即谁来运行根 CA 呢？解决方案是，并非只有一个根，而是有多个根，每个根有它自己的 RA 和 CA。实际上，现代浏览器在安装的时候预先加载了 100 多个根的公钥，有时候这些根称为**信任锚**（trust anchor）。通过这种方式，就可以避免在全球范围内使用单一的可信权威机构。

但是，现在又引出了另一个问题，即浏览器厂商如何确定哪些声称的信任锚是可靠的，哪些是不可靠的。对于用户来说，归根到底只能信任浏览器厂商会做出明智的选择，相信它们不会简单地把那些愿意支付包含费的根 CA 全部变成信任锚。大多数浏览器允许用户检查根密钥（通常以证书的形式出现，并且包含根自身的签名），也允许用户删除掉那些看起来不太让人放心的根密钥。

目录

对于任何一个 PKI，另一个问题是在哪里存放证书（以及能回溯到某一个知名信任锚的证书链）。一种可能的方案是让每个用户保存他（她）自己的证书。虽然这样做很安全（即

用户不可能在篡改了签过名的证书以后不会被检测到），但是并不方便。提出来的另一种方案是将 DNS 做成一个证书目录。Alice 在与 Bob 联系之前，她或许要通过 DNS 来查找 Bob 的 IP 地址，所以，为什么不让 DNS 在返回 IP 地址的时候也返回 Bob 的完整证书链呢？

有些人认为这是正确的解决之道，但是，其他人则希望使用专门的目录服务器来管理 X.509 证书。这样的目录允许用户利用 X.500 名字的属性来提交查询任务。例如，在理论上，这样的目录服务可以回答诸如这样的问题："将所有在美国或者加拿大销售部门工作的名为 Alice 的人的名单交给我。"

撤销

现实世界中也充满了各种各样的证书，比如护照和驾驶证。有时候这些证书可以被撤销，例如，对于酒后驾车或者其他的违章驾驶行为，警察局有权吊销驾驶员的驾驶证。在数字世界中也有同样的问题：一个证书的颁发者可能因为证书持有者以某种方式滥用他的证书而决定撤销他的证书。如果拥有者的私钥已被泄漏，或者更糟糕的情况下，CA 的私钥被泄漏，那么，这时候相关的证书都要被撤销。因此，任何一个 PKI 都需要处理有关证书撤销的事宜。撤销的可能性加剧了事情处理的复杂化。

处理的第一步是让每个 CA 定期地发布一个**证书撤销列表**（CRL，Certificate Revocation List），该 CRL 列出了所有已被撤销的证书序列号。由于证书包含了过期时间，所以 CRL 只需包含那些尚未过期的证书的序列号。一旦证书的过期时间已经过去，则该证书自动失效，所以，在那些已经过期的证书与真正被撤销的证书之间并不需要做专门的区分。无论哪一种情形，这些证书都不能再被使用了。

不幸的是，引入 CRL 意味着一个用户在使用证书之前必须先要获得 CA 的 CRL，以便确定该证书是否已被撤销。如果它确实已被撤销，则用户就不应该再使用该证书。然而，即使该证书不在 CRL 列表中，它也有可能在 CA 发布 CRL 列表之后刚刚被撤销。因此，真正有保证的唯一途径是询问 CA。而且，当下次再使用这个证书时，用户必须再次询问 CA，因为该证书有可能在几秒钟之前刚刚被撤销掉。

另一个复杂之处在于一个已撤销的证书也可以被恢复为有效证书，例如，如果撤销的原因仅仅是因为用户没有支付必要的费用。这种必须处理证书的撤销（可能也要处理恢复）要求同时也消除了证书的最佳特性之一，即用户在使用证书的时候可以不必与 CA 联系。

那么，CRL 应该被保存在哪里呢？一个理想的选择是证书本身所在的地方。一种策略是，让 CA 主动地定期推出 CRL，然后由各个目录服务器对 CRL 进行处理，它们只需删除那些被撤销的证书即可。如果目录服务器本身不保存证书，那么可以将 CRL 缓存在网络中的各个便利之处。由于 CRL 本身也是一个被签过名的文档，所以，如果它被篡改的话，用户很容易检测出来。

如果证书的有效期很长，那么 CRL 也将很长。例如，如果信用卡的有效期为 5 年，则尚在有效期内而被撤销的信用卡的数量，一定比每隔 3 个月发行新卡的情形要多得多。处理这种长 CRL 的一种标准方法是每过一段（较长）时间发行一个全列表，但在这期间频繁地发行 CRL 更新消息。这样做可以减少因分发 CRL 而需要的带宽。

8.6 通信安全

现在我们已经结束了对各种交易工具的学习。大多数重要的技术和协议也都已经涉及了。本章剩余部分将讨论如何把这些技术应用到实践中来提供网络安全性，同时在本章末尾还将提到一些有关安全性之社会因素的思考。

在接下来的4小节中，我们将讨论通信安全，也就是说，如何将数据位秘密地、未被篡改地从源端传送到接收方，以及如何将有害的数据位排除在外。这些问题绝不是网络环境中唯一的安全问题，但它们无疑是最重要的，所以，通信安全是一个很好的学习起点。

8.6.1 IPSec

多年以来IETF一直很清楚，Internet缺乏安全性。要在Internet上增加安全性并不容易，因为曾经爆发过一场关于在哪里加入安全机制的争论。绝大多数安全专家相信，为了真正做到安全，加密和完整性检查必须是端到端的（即位于应用层上）。这就是说，源进程对数据进行加密，以及/或者实施完整性保护，然后将数据发送给目标进程；在目标进程中，数据被解密，以及/或者被验证其完整性。若企图在这两个进程之间（包括在操作系统内部）篡改数据，则必定可以被检测到。这种做法的麻烦在于，它要求改变所有的应用系统，以便它们能够感知到安全特性的存在。从这个角度来说，次好的方法是将加密功能放到传输层上，或者放到应用层与传输层之间的一个新层上，这样仍然可以做到端到端的安全，但是不要求改变应用系统。

另一种相反的观点是，用户并不理解安全性，他们不具备正确地使用它的能力，并且没有人愿意以任何一种方式来修改已有的程序，所以网络层应该认证和/或者加密数据包，而不应该让用户卷入进来。在经过多年的激烈争论之后，这种观点赢得了足够的支持，因此IETF定义了一个网络层的安全标准。这么做的部分理由是，在网络层上实施加密操作，并没有妨碍那些了解安全性的用户正确地使用这些安全特性，而且多多少少可以帮助那些对安全性一无所知的用户。

这场争战的结果是提出了一个称为**IP安全**（IPSec，IP security）的设计方案，它由RFC 2401、2402和2406等一些RFC文档描述。然而，并不是所有的用户都想要加密功能（因为加密操作的计算代价较高）。虽然IPSec并没有将加密功能做成可选项，相反，它总是要求加密功能，但它允许使用**空算法**（null algorithm）。空算法因其简单、易于实现和极高的速度而获得了高度赞赏，RFC 2410描述了空算法。

完整的IPSec设计方案是一个多服务、多算法和多粒度的框架。IPSec支持多种服务的原因在于考虑了并不是每个人都愿意为所有的服务、所有的时间支付费用，所以，IPSec按照点菜的方式来提供这些服务。最主要的服务是保密性、数据完整性，以及针对重放攻击（即入侵者重放一次会话过程）的保护。所有这些服务都建立在对称密钥密码学的基础之上，因为高性能在这里非常关键。

IPSec支持多算法的考虑在于任何一个现在被认为安全的算法在将来某个时候可能会

被攻破。将 IPSec 设计成与算法无关的好处是，即使某个特殊的算法将来被攻破了，这个框架仍然可以幸存下来。

IPSec 支持多粒度的理由是，这样可使得它既能够保护单个 TCP 连接，也能够保护一对主机之间的所有流量，或者一对安全路由器之间的所有流量，以及其他一些可能性。

关于 IPSec，一个略微使人惊讶的方面在于，尽管它落在 IP 层，但它是面向连接的。实际上，这也无须惊讶，因为为了做到任何一种安全性，建立一个密钥并且在一定长的时间内使用该密钥——在本质上，这就是一种连接，只是用了不同的名称。而且，在面向连接的环境中，建立连接的开销可以被分摊到许多数据包。在 IPSec 环境中，一个“连接”称为一个**安全关联**（SA，security association）。SA 是两个端点之间的单纯连接，它有一个与之关联的安全标识符。如果两个方向上都需要安全通信的话，则要求使用两个安全关联。数据包携带着安全标识符通过这些安全连接，而标识符是用来查询密钥和其他有关安全信息。

技术上，IPSec 有两个主要部分。第一部分描述了两个新的头，这两个新的头可以被加入到数据包中以便携带安全标识符、完整性控制数据和其他的信息。另一部分是 Internet **安全关联及密钥管理协议**（ISAKMP，Internet Security Association and Key Management Protocol），处理建立密钥。ISAKMP 是个框架，用来开展工作的主要协议是 Internet **密钥交换**（IKE，Internet Key Exchange）。由于早期版本的 IKE 存在严重的缺陷，（Perlman 和 Kaufman，2000）指出应该使用在 RFC 4306 中描述的 IKE 第 2 版本。

IPSec 有两种使用模式。在**传输模式**（transport mode）中，IPSec 头被直接插在 IP 头的后面。IP 头中的 Protocol 字段要做相应的修改，以表明有一个 IPSec 头紧跟在普通 IP 头的后面（但是在 TCP 头的前面）。IPSec 头包含了安全信息，主要是 SA 标识符、一个新的序号，可能还包括有效载荷数据的完整性检查信息。

在**隧道模式**（tunnel mode）中，整个 IP 数据包，连同头和所有的数据一起被封装到一个新的 IP 数据包的数据部分，并且这个 IP 数据包有一个全新的 IP 头。当隧道的终点并不是最终的目标节点时，隧道模式将非常有用。在有些情况，隧道的终点是一台安全网关机器，例如，公司的一个防火墙。在这种模式中，当数据包通过防火墙时，防火墙负责封装或者拆封数据包。由于隧道终止于这台安全的机器，所以公司 LAN 上的机器不必知晓 IPSec 的存在。只有防火墙必须要知道 IPSec。

当一束 TCP 连接被聚合起来，作为一个加密的流被处理的时候，隧道模式也非常有用，这种用法的好处是，入侵者将无法看到谁给谁发送了多少数据包。有时候，仅仅知道有多少流量发送到哪里，这本身就是很有价值的信息。例如，如果在一次军事危机中，五角大楼和白宫之间的网络流量突然减少了许多，但是五角大楼与科罗拉多州洛矶山脉（Colorado Rocky Mountains）某个军事基地之间的流量却增加了同等的数量，那么入侵者就有可能从这些数据中推断出某些有用的信息。研究数据包的流模式（即使是加密的数据流）称为**流量分析**（traffic analysis）。IPSec 的隧道模式在某种程度上提供了一种应对的方法。隧道模式的缺点是，它加入了一个额外的 IP 头，因此实实在在地增加了数据包的长度。与此相反，IPsec 的传输模式不会明显影响数据包的长度。

第一个新的头是**认证头**（AH，Authentication Header）。它提供了完整性检查和防止重放攻击的安全性，而没有提供保密性（即没有加密数据）。在传输模式中 AH 的用法如

图 8-27 所示。在 IPv4 中，AH 被插在 IP 头（包括可能有的所有选项）和 TCP 头之间；在 IPv6 中，它仅仅是另一个扩展头而已，就如同一般的扩展头那样。实际上，AH 的格式与标准的 IPv6 扩展头的格式非常接近。有效载荷部分有可能必须被填充到某个特殊的长度以适应认证算法的要求，如图 8-27 所示。

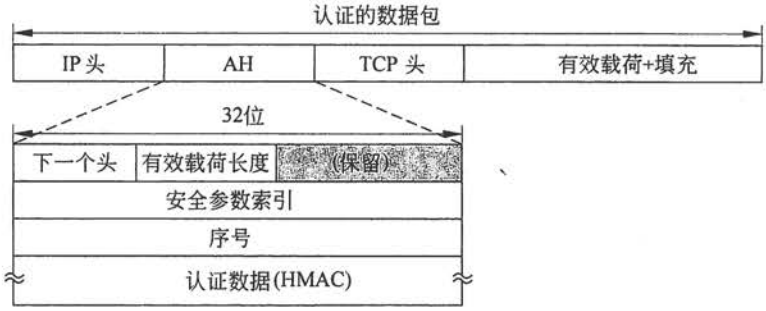


图 8-27 IPv4 传输模式下的 IPSec 认证头

现在我们来考查 AH 头。下一个头（Next header）字段被用来保存原始的 IP 包的协议（Protocol）字段，该字段后来被替换成 51 以表明后面紧跟着一个 AH 头。在绝大多数情况下，这里出现的是 TCP 协议代码（6）。有效载荷长度（Payload length）字段等于 AH 头部中 32 位字的个数减 2。

安全参数索引（Security parameters index）是连接标识符。它由发送方插入，用来指定接收方数据库中一条特定的记录。该记录包含了这个连接上所使用的共享密钥，以及有关该连接的其他信息。如果这个协议是由 ITU 而不是 IETF 发明的，则这个字段称为**虚拟电路号**（virtual circuit number）。

序号（Sequence number）字段用来对通过一个 SA 发送的所有数据包进行编号。每个数据包都有一个唯一的编号，即使重传的数据包也有唯一的编号。换句话说，重传的数据包与原始的数据包具有不同的序号（尽管它们的 TCP 序号是相同的）。这个字段的目的是为了检测重放攻击。这些序号不会发生回绕的现象，因为如果 2^{32} 个序列号全部用完以后；IPSec 必须建立一个新的 SA 以便继续进行通信。

最后是 Authentication data（**认证数据**）字段，它是一个可变长度字段，包含了有效载荷的数字签名。当 SA 被建立时，双方协商将使用哪个签名算法。通常情况下，这里不使用公开密钥密码学，因为数据包的处理速度必须极快，而所有已知的公钥算法都太慢。由于 IPSec 建立在对称密钥的基础上，而且发送方和接收方在建立 SA 之前协商了一个共享密钥，所以，在计算签名过程中可以利用这个共享密钥。一种简单的办法是在数据包和共享密钥合起来的位串上计算散列值。当然，这个共享的密钥并不在网络上传输。像这样的方案称为**散列的消息认证码**（HMAC，Hashed Message Authentication Code）。它的计算速度比先运行 SHA-1 然后在其结果上再运行 RSA 要快得多。

AH 头并不支持数据加密功能。因此，只有在需要完整性检查但不需要保密性的场合，AH 头才十分有用。AH 头有一个值得注意的特性，那就是它的完整性检查也覆盖了 IP 头中的某些字段，即当数据包在路由器之间被转发时不随路由器而变化的那些域。例如，TTL（Time to live）字段在每一跳上都要改变，所以它不能被包含在完整性检查的范围内。然而，

IP 源地址被包含在检查范围内，这就使入侵者无法伪造数据包的来源信息。

另一个 IPSec 头是**封装的安全有效载荷**（ESP，Encapsulating Security Payload）。图 8-28 显示了 ESP 在传输模式和隧道模式下的使用情形。

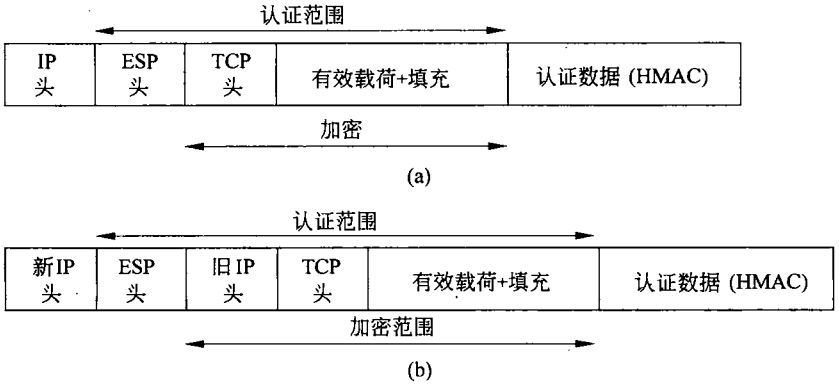


图 8-28

(a) 传输模式中的 ESP；(b) 隧道模式中的 ESP

ESP 头由两个 32 位字组成。它们是安全参数索引（Security parameters index）和序号（Sequence number）字段，我们在 AH 头中已经看到过这两个字段了。通常跟在这两个字后面的第三个字是用于数据加密的初始向量（Initialization vector），如果使用空加密算法，则可以省略这个字段。

如同 AH 一样，ESP 也提供了 HMAC 完整性检查，但是完整性检查的范围并不包含头部，而是在有效载荷之后，如图 8-28 所示。将 HMAC 放在末尾对于硬件实现是有好处的。在数据位通过网络接口卡往外发送的过程中，同时可以计算出 HMAC，然后再追加到尾部。这也正是为什么以太网和其他的 LAN 将它们的 CRC（循环冗余校验码）放在尾部而不是头部的原因。若使用 AH，则数据包必须被缓冲，并且在发送数据包之前首先要计算出签名，这样有可能会降低每秒钟发送的数据包的数量。

既然 ESP 可以完成 AH 所能做的任何事情，甚至还能做得更多，而且效率也更高，那么，一个很自然产生的问题是：为什么还要引入 AH 呢？这主要是由于历史的原因。最初，AH 只处理完整性，而 ESP 只处理保密性。后来，完整性被加入到 ESP 中，但是，那些设计 AH 的人不希望在 AH 已经实用之后让它死掉。然而，他们唯一提出的真正论据是，AH 检查 IP 头的一部分，而 ESP 不检查 IP 头，当然这并没有多少说服力。另一个也没有什么说服力的原因是，如果一个产品支持 AH 但不支持 ESP，那么它或许可以省却有关出口许可的诸多麻烦，因为它不能够做加密操作。AH 在将来有可能被慢慢淘汰掉。

8.6.2 防火墙

能够将任何地方的任何一台计算机与另外一个地方的另一台计算机连接起来，这种能力令人喜忧参半。对于家庭个人用户来说，在 Internet 上漫游充满了各种乐趣。但对于公司的安全管理员来说，这是一种梦魇。大多数公司有大量的保密信息放在网络上，例如交易的秘密、产品开发计划、市场策略、财务分析等。如果这些信息被泄露给竞争对手将会

带来可怕的后果。

除了信息往外泄漏的危险性以外，同时还存在着信息渗透进来的危险性。尤其是，病毒、蠕虫和其他的数字有害物可能会破坏安全性、销毁有用的数据，管理员必须消耗大量精力才能清理它们留下的混乱局面。这些有害物通常是由于粗心的员工在玩各种新奇的游戏时引入的。

因此，公司的网络需要有一些机制允许“好的”数据位进来，而阻止“坏的”数据位出去。一种方法是使用 IPSec。这种方法可以保护安全站点之间传送的数据。然而，IPSec 并不能阻止数字有害物和入侵者进入到公司的 LAN 中。为了看清楚如何实现这个目标，我们需要考察防火墙。

防火墙（firewall）只不过是现代版的古代中世纪安全设施，即在城堡周围挖一条很深的护城河来保护城堡。在古代，这种设计强迫每个进出城堡的人都要经过一座吊桥，而他们在通过吊桥时，哨兵可以对他们进行检查。在网络环境中，也可以采用相同的方法。公司可以按照任意的方式将多个 LAN 连接起来，但是强迫所有进出公司的流量必须通过一座电子吊桥（即防火墙），如图 8-29 所示。除此之外，没有其他路由可走。

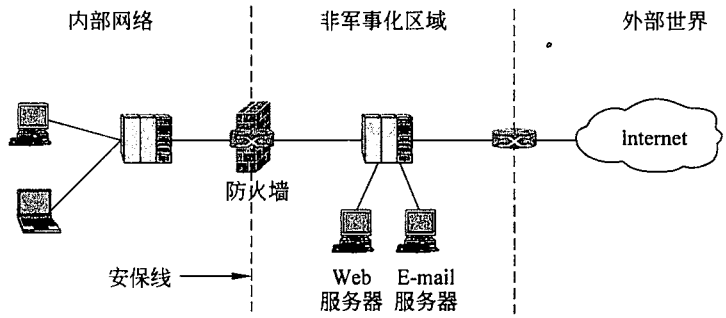


图 8-29 保护内部网络的防火墙

防火墙的作用就像一个**包过滤器**（packet filter）。它检查进出的每个数据包。通常只有那些符合网络管理员所定规则和条件的数据包才被放行，而那些不符合条件的数据包则被丢弃。

典型的过滤标准以规则或者表格列出，它通常含有可接受的源点和目标点，被阻塞的源点和目标点，以及含有关于往来其他机器的数据包的处理规范。在通常的 TCP/IP 设置中，源或者目标点也许都含有一个 IP 地址和一个端口。端口号说明了哪些服务是需要的。例如，在 TCP 中，端口 25 用于电子邮件，端口 80 用于 HTTP。可以简单地通过阻塞某些端口来过滤数据包。例如，一个公司可以阻塞所有 IP 地址上带有 TCP 端口 79 的数据包。曾经很普遍的一个做法是，让 Finger 服务搜寻人们的电子邮件地址，但是这种方法现今已经很少采用。

其他的端口并不是那样容易被封杀。困难在于网络管理人员既要保证网络安全，但又不能切断与外界的通信。要设置得让网络比较安全并不难，但是要使得没有人抱怨并不容易。这就是图 8-29 中**非军事化区域**（DMZ，DeMilitarized Zone）的设置派上用场的地方。DMZ 是公司网络的一部分，但是它被放置在安全装置（例如防火墙）的外部，是进入网络的入口。放置一个诸如 Web 服务器那样的机器在 DMZ 中，Internet 上的计算机可以联系它

来浏览公司的网页。现在可以设置防火墙使之阻塞所有想通过端口 80 的入境 TCP 流量，从而 Internet 上的计算机无法通过这个端口来攻击在公司内部网的计算机。为了允许对 Web 服务器实施管理，可以在防火墙上制定一个允许 Web 服务器和内部机器连接的规则。

通过一段时期跟攻击者的竞争，防火墙已经变得非常复杂。刚开始时，防火墙只是将一组规则独立地用在每个数据包，但是实践已经证明很难制定一些规则，它既能让有用的信息通过而又能封杀所有不想要的流量。**状态防火墙**（Stateful firewalls）将数据包映射到连接，并且使用 TCP/IP 的头字段来保持跟踪连接。这样的设置倒是可以制定出一些规范，例如允许一个外部的 Web 服务器给内部机器发送数据包，但是仅在内部机器先跟外部 Web 服务器建立一个连接之后。这样一个规则不可能用于无状态设计，因为它只能使所有的来自外部 Web 服务器的数据包要么全部通过要么全部丢弃。

来自状态处理的另一层复杂性是为防火墙实现**应用层的网关**（application-level gateway）。这个处理涉及防火墙要检查数据包内部（甚至至 TCP 头后部分）来查看应用正在做什么。具备了这样的能力，防火墙就有可能区别用于网页浏览的 HTTP 流量和用于对等文件分享的 HTTP 流量。管理员可以制定规则避免公司被用于对等文件共享，但同时又保证对公司运行至关重要的网页浏览。所有这些方法，不仅要检查出境流量还要检查入境流量，例如，为了防止公司内部的敏感信息通过电子邮件方式泄露出去。

经过上述讨论，我们应该很清楚防火墙违反了协议的标准分层。它们是网络层设备，但是它们偷窥传输层和应用层内容来做过滤。这使它们变得脆弱。例如，防火墙往往依赖于标准端口号的约定来判断数据包含有哪类流量。虽然标准端口常被按约定使用，但是并非所有的应用都使用标准端口。一些对等应用动态地选用标准端口来避免被轻易地认出（和阻塞）。使用 IPSEC 或者其他加密手段使得防火墙无法摄取上层信息。最后，当计算机相互交谈通过防火墙时，防火墙无法告诉它们使用的规则，以及为何要撤销他们的连接。它只能简单地假装线路不通。基于所有这些原因，网络纯粹主义者认为防火墙是 Internet 体系结构上的一个污点。然而，对计算机来说 Internet 可能是一个危险的地方，防火墙能帮助它们避免遭受攻击，所以还是不会被淘汰。

即使防火墙的配置极其完美，网络中仍然会存在大量的安全问题。例如，如果一个防火墙被配置成只允许接收来自某些特定网络的数据包（比如，来自公司其他分部的流量），那么，防火墙外面的入侵者可以在数据包中填上假的源地址，从而绕过防火墙的检查机制。如果一个内部人员想要偷取保密文档，他可以加密这些文档，或者先把文档拍摄下来然后以 JPEG 文件的形式将图片转运出来，这样可以绕过任何电子邮件过滤器。我们一直没有提到，事实上，在所有的攻击事件中，虽然有四分之三来自防火墙外部，但是来自防火墙内部的攻击通常损害最大，例如来自心怀不满的员工报复（Verizon，2009）。

防火墙的另一个不同问题是它提供了单道防线来阻挡攻击，如果它被攻破就全盘皆输。基于这个原因，通常都使用多层防火墙来防范攻击。例如，一个防火墙可以放置在内部网入口，而且每个计算机可能也要运行自己的防火墙。那些认为只需要一道安全关口就够了的读者，显然近来没有搭乘过定时国际航班。

另外，还存在许多其他类型的攻击是防火墙无法处理的。防火墙的基本思想是阻止入侵者进入，或者避免保密数据被转运出去。不幸的是，还有一些人的手段更加低劣，他们不会别的，只会想方设法把目标站点搞瘫痪。他们的做法是，向目标机器发送大量合法的

包，直至目标机器不堪重负而崩溃。例如，为了削弱一个 Web 站点，入侵者可以发送一个 TCP SYN 包来请求建立一个连接。然后 Web 站点为这个连接分配一个内部表项，并且发送一个 SYN+ACK 包作为应答。如果入侵者不再响应的话，则这个表项将被占用一定长的时间（比如几秒钟），直至它超时。如果入侵者在短时间内发送数千个连接请求，则所有的表项都将被占用，从而 Web 站点无法再建立起正常合法的连接。在这种攻击中，入侵者的意图是使目标机器停止服务，而不是偷取数据，因此这种攻击称为**拒绝服务攻击**（DoS，Denial of Service）。一般情况下，请求包中的源地址是伪造的，所以入侵者不太容易被追踪到。DoS 攻击 Web 站点的现象在 Internet 很常见。

一个更加糟糕的变种是，入侵者已经攻破了遍布世界各地的数百台计算机，然后他命令所有这些机器同时向同一个目标发起攻击。这种方法不仅增加了入侵者的威力，而且也减小了他被检测到的几率，因为那些攻击包来自大量的机器，并且这些机器属于普通的可信任用户。这样的攻击称为**分布式拒绝服务**（DDoS，Distributed Denial of Service）。这种攻击很难防护。即使被攻击的机器能够快速地识别出伪造的请求，它也必须花一定的时间来处理该请求，然后再将它丢弃；如果每秒钟到达的请求足够多，CPU 将不得不花费所有的时间来处理这种请求。

8.6.3 虚拟专用网络

许多公司的办事处和部门分散在多个城市中，有时候甚至分布在多个国家。过去，在公共数据网络出现以前，对于这些公司来说，最常见的做法是从电话公司租用一些线路，将各场所或者全部场所两两连接起来。现在有些公司仍然这样做。利用公司的计算机和租用的电话线路而建立起来的网络称为**专用网络**（private network）。

专用网络工作得很好，而且也非常安全。如果可用的线路仅仅是租用线路的话，则所有的流量不会泄漏到公司各工作场所以外，入侵者必须通过物理搭线的方法才能闯入，而这种物理搭线的方法并不容易做到。但是，这种专用网络存在的问题是，租用一条 T1 线路的开销是每个月几千美元，而租用一条 T3 线路可能需要几倍的价格。当公共数据网络以及后来的 Internet 出现以后，许多公司希望将它们的数据流量（可能还有语音流量）转移到公共网络上，但是又不希望损失专用网络的安全性。

这种需求很快导致了**虚拟专用网络**（VPN，Virtual Private Networks）的诞生。VPN 是建立在公共网络之上的层叠网络，但是具有专用网络的绝大多数特性。它们之所以称为“虚拟的”，是因为它们仅仅是一个假想的网络，就好像虚电路并不是真正的电路、虚拟内存并不是真正的内存一样。

一个流行的做法是直接在 Internet 上构建 VPN。一个很常见的设计是在每个办事处都设置一个防火墙然后在所有办事处之间创建一条通过 Internet 的隧道，如图 8-30(a)所示。使用 Internet 连接的进一步优势是隧道可以根据需要而设置，比如包含一个员工在家里的计算机，或者他正在旅行时用的笔记本电脑，只要他能够有 Internet 连接就可以。这种灵活性远远优于使用租用的线路。然而站在 VPN 的角度来看，拓扑结构看起来就像专用网络一样，如图 8-30(b) 所示。当系统启动时，每一对防火墙必须协商它们的 SA 参数，包括服务、模式、算法和密钥。如果隧道采用了 IPSec，那么很有可能要聚合两个办事处之间的所

有流量组成一个经过认证的加密 SA，从而提供完整性控制、保密性，甚至可考虑流量分析的防护措施。许多防火墙有内置的 VPN 功能。一些常规路由器也有这功能。但是因为防火墙主要用在安全方面，很自然防火墙上有隧道的起点和隧道的终点，它在公司和 Internet 之间提供了一条清楚的分界线。因此，防火墙、VNP 和具有 ESP 的 IPSec（隧道模式）是很自然的一种组合，并且在实践中得到了广泛应用。

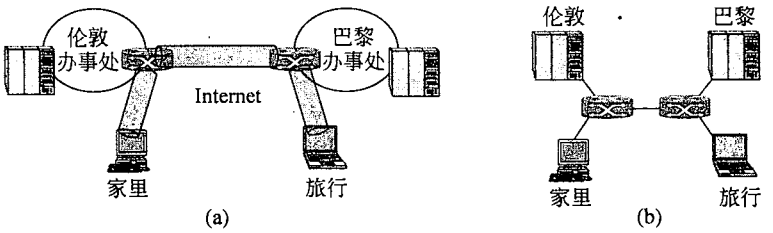


图 8-30

(a) 虚拟专用网络；(b) 从内部看的拓扑结构

一旦建立起 SA，数据包流量就可以开始流动了。对于 Internet 上的一台路由器而言，沿着 VPN 隧道传递的一个数据包仅仅是一个普通的数据包而已。这个数据包的唯一不同寻常之处在于 IPSec 头出现在 IP 头的后面，但是，由于这个额外的头不影响转发过程，所以，路由器并不关心这个额外的 IPSec 头。

另一个逐渐流行的方法是让 ISP 来设置 VPN。使用 MPLS（正如第 5 章讨论的），VPN 流量的路径甚至可以穿过公司办事处之间的 ISP 网络。这些路径保持使得 VPN 流量独立于 Internet 上其他流量，并且可以获得一定数量的带宽或者其他的服务质量保障。

VPN 的一个关键好处是它对于所有的用户软件都是透明的。防火墙建立并管理 SA。唯一知道这种配置方案的人是系统管理员（他必须配置和管理安全网关），或者是 ISP 管理员（他必须配置 MPLS 路径）。对于其他人来说，这样的网络与租用线路的专用网络并无区别。有关 VPN 的更多信息，请参考（Lewis，2006）。

8.6.4 无线安全性

利用 VPN 和防火墙来设计一个逻辑上绝对安全的系统是出奇地容易，但是，实际上，这样的系统仍然有可能像筛子一样漏洞百出。例如，如果有些机器是无线的，它们使用无线电波进行通信，这样就会在进入和出去两个方向上绕过防火墙。802.11 网络的范围通常是几百米距离，所以，如果一个人想要暗中监视一家公司，那么他只需早晨开车到该公司的停车场，并把一台支持 802.11 协议的笔记本电脑留在汽车里以便记录下所有听到的内容，然后他自己离开停车场。到下午晚些时候，该计算机的硬盘上就充满了各种有价值的信息。理论上，这种泄漏是不应该发生的。同样地，理论上，人们是不应该去抢劫银行的。

大多数安全问题的根源在于无线基站（接入点）的生产商，它们总是试图使自己的产品对用户极为友好。通常情况下，如果用户将设备从包装盒中取出来并插到电源插口上，它立即就开始运行了——它几乎没有任何安全性，所有秘密都暴露给无线电波范围内的任何一个人。如果它后来又被插入到一个以太网中，则所有的以太网流量也突然出现在停车场内。无线连接使得偷窥者的梦想成为现实：无须做任何工作就可以免费获得数据。因此，

毋庸多说，安全性对于无线系统比有线系统更加重要。在这一小节中，我们将考查无线网络处理安全性的几种方法。要想获得更多的信息，请参考（Nichols 和 Lekkas，2002）。

802.11 安全性

部分 802.11 标准，最初称为 802.11i 规定了一个无线数据链路层安全协议，用来防止一个无线节点阅读或干扰另一对无线节点之间发送的消息。它还有个商标名称为 **WiFi 保护接入 2**（WPA2，WiFi Protected Access 2）。纯 WPA 是一个临时方案，它实现了 802.11i 的一个子集。应该避免使用它，而使用 WPA2。

我们马上将介绍 802.11i，在那之前我们首先应该知道它是**有线等效保密**（WEP，Wired Equivalent Privacy）的替代品，这是第一代 802.11 安全协议。WEP 由一个网络标准委员会设计，这个过程与 NIST 选取 AES 设计的过程完全不同。结果是灾难性的。问题究竟出在何处？从安全角度来看，它几乎每个方面都有问题。例如，WEP 加密数据使用了和流密码输出一样的异或操作。不幸的是，弱密钥分配意味着输出经常被重复使用。由此导致了通过不断尝试就可以攻破它的保护的直接后果。另一方面，它的完整性检查基于 32 位的 CRC。这种方式对于检测传输错误是足够的，但它并不是一个可用来防范攻击的很强保密机制。

这些以及其他设计上的缺陷使 WEP 很容易被攻破。第一个攻破 WEP 的实际演示由 Adam Stubblefield 在 AT&T 当实习生时完成（Stubblefield 等，2002），他能够在一个星期内编写和测试 Fluhrer 等描述的攻击，而且他把多数时间花在说服管理人员买一个实验用的无线 WiFi 卡。现在一分钟内破解 WEP 的软件唾手可得并且是免费的，因此极不倡导使用 WEP。虽然它可以防止那些随意的访问，但它并不能提供任何形式的真正安全保证。了解到 WEP 被严重破解后，802.11i 组仓促上马，于 2004 年 6 月提出了一个正式的标准。

现在我们来介绍 802.11i，如果适当地设置和使用，它能够提供真正的安全防护。WPA2 通常用在两种常见的场合下。第一种情形是企业设置，在这种情况下公司有一台单独的认证服务器，该服务器上有一个用户名和口令数据库，根据数据库来确定一个无线用户是否允许访问网络。在这种设置中，客户端使用标准的协议向网络验证他们自己。主要的标准是 802.1X，接入点用该协议让客户端和认证服务器对话并观察其结果，而**可扩展的身份验证协议**（EAP，Extensible Authentication Protocol）（RFC 3748）则告诉了客户端和认证服务器是如何互动的。事实上，EAP 是个框架，其他标准定义了协议消息。然而，我们不会深入到交互的很多细节，因为这些内容对于概述并不重要。

第二种情形是在家庭设置，此时没有认证服务器。相反，只有一个被客户端用来访问无线网络的共享密码。这种设置比起认证服务器要简单得多，这也是为何它通常被用在家庭环境或者小型公司。但相比之下它的安全性也弱了一些。两种设置的主要区别在于有认证服务器时，每个用户都拿到一个用来加密流量的密钥，而其他用户对此并不知晓。使用单个共享密码时，虽然它为每个客户端推导出不同的密钥，但所有的客户端具有相同的密码，只要愿意，客户端们也能推导出其他客户端的密钥。

用来加密流量的密钥是在身份验证的握手过程中计算出来的。握手过程发生在客户和一个无线网络关联，并且与一个认证服务器认证后产生的。在握手开始时，客户端要么有一个共享的网络密码，要么有一个认证服务器的密码。这个密码被用来推导出一个主密钥。然而，该主密钥并没有直接被用来加密数据包。实际上应该采用标准的加密方法来推导出

每次使用期限内的会话密钥、修改不同会话的密钥并尽可能少地暴露主密钥。

如图 8-31 所示，会话密钥是通过四次包（four-Packet）握手计算出来的。首先，接入点（AP）发出一个随机数用作识别。像这样在安全协议中仅使用一次的随机数称为**临时值**（nonce），它或多或少是“number used once”（一次性数字）的一个简写。客户可以选取它自己的临时值，然后它用临时值、它的 MAC 地址和 AP 的 MAC 地址以及主密钥作为参数计算出一个会话密钥 K_S。会话密钥由几部分组成，每一部分都有不同的用途，但我们忽略这个细节。现在客户有会话密钥，但接入点（AP）还没有。因此客户发送它的临时值给 AP，AP 执行相同的计算推导出同样的会话密钥。临时值可以明文发送，因为在缺乏额外机密信息的情况下，会话密钥无法从临时值中推导出来。从客户发出的消息用**消息完整性检查**（MIC，Message Integrity Check）来加以完整性保护。AP 在它计算会话密钥之后检查 MIC 是否正确，以便确定消息的确来自于该客户。就像 HMAC 一样，MIC 是消息认证码的另一个名字。因为很有可能跟 MAC（Medium Access Control）地址混淆，MIC 术语通常用来替代网络协议。

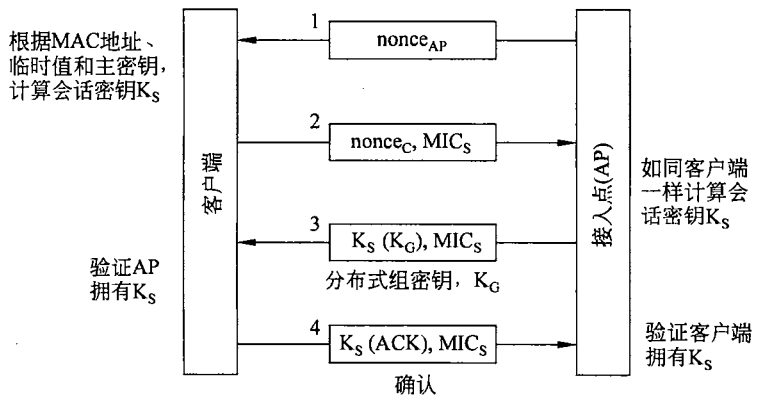


图 8-31 802.11i 密钥建立的握手过程

在最后两个消息中，AP 分发一个组密钥 K_G 给客户，并且客户要确认该消息。这些消息的接收让客户验证 AP 有正确的会话密钥，反之，AP 也知道客户端有正确的密钥。组密钥用在 802.11 局域网中广播和组播流量。因为四次包握手的结果是每个客户都有自己的加密密钥，因此 AP 无法使用这些密钥中的任何一个给所有的无线客户广播数据包；AP 必须为每个客户发送一个单独的副本，并且使用该客户的密钥。相反，给所有的无线客户分发一个共享密钥可以使得广播流量只发送一次，并且被所有的客户端接收。当有客户离开或者加入网络时，该共享密钥必须被更新。

最后，我们讨论如何实际使用密钥来提供安全性。802.11i 可使用两个协议来提供消息的保密性、完整性和身份验证。其中一个协议称为**临时密钥完整性协议**（TKIP，Temporary Key Integrity Protocol），它像 WPA 一样，也是一个临时解决方案。它被设计用来改进老和慢的 802.11 网卡安全性，因此在某些安全性上面至少比 WEP 更好，它可以被当成固件升级。然而，它现在也被破解了，所以你最好用另一个推荐的协议 CCMP。究竟什么是 CCMP？CCMP 是一个蔚为壮观的名称缩写，它的全称是具有密码块链的**计数器模式消息认证码协议**（Counter mode with Cipher block chaining Message authentication code Protocol），我们就简称它为 CCMP，你可以随意地称呼它。CCMP 的工作原理很简单。它用 AES 加密，使用

的密钥和块大小都是 128 位。它的密钥来自于会话密钥。为了提供保密性，消息用计数器模式的 AES 加密。记得我们曾在 8.2.3 节讨论过**密码模式**（Cipher mode），这些模式防止了每次用相同的位来加密相同的消息。计数器模式在加密中混合了计数值。为了提供完整性，包括头字段在内的消息都用密码块链模式进行了加密，并且最后长度为 128 位的块作为 MIC 被保存起来。然后，消息（用计数器模式加密后）和 MIC 同时被发送出去。在接收到一个无线数据包时，客户和 AP 各自执行加密操作，或者验证这个加密消息。对于广播或者组播的消息，使用共享的组密钥执行相同的处理步骤。

蓝牙安全性

蓝牙（Bluetooth）的无线范围比 802.11 要短得多，所以它不容易受到来自停车场的攻击，但是它的安全性仍然是个问题。例如，想象 Alice 的计算机配备了一个无线蓝牙键盘。在缺少安全防护的情况下，如果 Trudy 碰巧在邻近的办公室，那么她就可以读取到 Alice 敲入的所有信息，包括她往外发送的所有电子邮件。她也可以捕获到 Alice 计算机发送给不远处蓝牙打印机的所有信息（比如 Alice 接收到的电子邮件或者机密报告）。幸运的是，蓝牙有一个非常精致的安全方案，它试图让全世界 Trudy 们的攻击无法得逞。现在我们概要地描述这个安全方案的主要特性。

蓝牙版本 2.1 和更新的版本有 4 种安全模式，从“没有任何安全性”到“完全的数据加密和完整性控制”。如同 802.11 一样，如果安全功能被禁止（默认模式），则没有任何安全性可言。大多数用户关闭了安全功能；直到发生了严重的安全事件以后他们才将安全功能打开。在农业社会中，这种做法相当于等到马跑掉以后，才想起来要锁上马房的门。

蓝牙在多个层上提供了安全性。在物理层，跳频机制提供了一定程度的安全性，但是由于任何一个蓝牙设备在进入到一个微网时，它必须要被告知跳频序列，所以这个序列很显然不是一个秘密。当一个新到来的从节点向主节点请求一个信道时，真正的安全性开始了。在蓝牙 2.1 之前，假设两个设备分享一个预设置的密钥。在有些情况下，两个设备是由同一个生产商搭配在一起的（比如，头戴耳机和移动电话配套出售）。在其他情况下，一个设备（比如头戴耳机）有一个硬置的密钥，用户必须将这个密钥（十进制数）输入到另一个设备中（比如移动电话）。这些共享的密钥称为**万能密钥**（passkey）。不幸的是，万能密钥通常被预先设置为“1234”或者另一个可以猜测到的值，总之是仅允许有 10^4 种选择的 4 位数字。在蓝牙 2.1 中，使用简单的安全配对，设备选取 6 位数字。虽然它让万能密钥更难被猜到，但是距离安全还是太遥远。

为了建立一条信道，从节点和主节点都要检查对方，看它是否知道万能密钥。如果知道，则它们之间进行协商以便确定信道是否要加密、是否要进行完整性控制，或者两者都要。然后它们选择一个随机的 128 位会话密钥，其中有些位可能是公开的。之所以允许弱化这个密钥，是为了遵从不同国家的政府限制，从而避免出口或者使用超出政府破解能力的密钥长度。

蓝牙的加密操作使用了一个称为 E_0 的流密码；完整性控制使用 SAFER+。两者都是传统的对称密钥块密码算法。SAFER+曾经参加了 AES 算法竞赛，但是第一轮就被淘汰了，因为它比其他候选算法都要慢。蓝牙在 AES 密码被选定之前就已经确定选择 SAFER+，否则它极有可能会使用 Rijndael。

使用流密码的实际加密过程如图 8-14 所示，其中明文与密钥流做 XOR 操作以便生成密文。不幸的是，E_0 本身（就像 RC4 一样）可能有致命的弱点（Jakobsson 和 Wetzel，2001）。虽然在本书撰写时它还没有被攻破，但是它与 A5/1 密码之间的相似之处引起了人们的关注（Biryukov 等，2000），A5/1 的惨败危及所有的 GSM 电话流量。有时候人们（包括我在内）真的很惊讶，在密码编码学家和密码分析家之间长期的“猫与老鼠”的游戏中，密码分析家总是赢家。

另一个安全问题是蓝牙只认证设备，而不认证用户，所以蓝牙设备被盗之后，窃贼有可能访问到用户的金融账户或者其他账户。然而，蓝牙在上层实现了安全性，所以即使链路层的安全性被突破，其他的安全性仍然存在，尤其是对于那些要求从某种键盘上手工敲入 PIN 码才能完成交易的应用来说更是如此。

8.7 认证协议

认证（Authentication）是这样的一项技术：一个进程通过认证过程来验证它的通信对方是否是它所期望的实体而不是假冒者。在面对一个恶意的主动入侵者时，要验证远程进程的身份是极其困难的，它要求使用一些基于密码学理论的复杂协议。在本节中，我们将学习许多认证协议，这些协议可被用在不安全的计算机网络中。

顺便提一下，有些人可以会混淆**授权**（authorization）和**认证**（authentication）这两个概念。认证所针对的问题是，你是否真的在跟一个特定的进程通信；而授权关注的是允许这个进程做什么样的事情。例如，一个客户进程与文件服务器建立联系，并且说：我是 Scott 的进程，我想删除文件 cookbook.old。从文件服务器的角度来看，有两个问题是它必须要回答的：

（1）这真的是 Scott 的进程吗（认证）？

（2）Scott 是否有权删除 cookbook.old（授权）？

只有当这两个问题都得到明确肯定的答复之后，客户所请求的动作才可以执行。前一个问题实际上是一个非常关键的问题。一旦服务器知道了它在跟谁通话，那么，检查授权就非常容易了，它只要在本地的数据表或者数据库中查找一些记录即可。由于这个原因，我们在本节中把注意力集中在认证上。

基本上所有的认证协议都使用一个通用的模型，如下所述。Alice 首先发起认证过程，她给 Bob 或者可信的**密钥分发中心**（KDC，Key Distribution Center）发送一条消息。假设这里的 KDC 是诚实可靠的。接下来在不同方向上继续交换其他一些消息。当这些消息被发送出去以后，Trudy 可能截取、修改或者重放这些消息以便欺骗 Alice 和 Bob，或者纯粹捣乱他们的工作。

然而，当协议完成的时候，Alice 确信她是在跟 Bob 通话，而 Bob 也确信他是在跟 Alice 通话。而且，在绝大多数协议中，两者还将建立起一个秘密的**会话密钥**（session key），用在接下来的会话过程中。实际上，尽管公开密钥密码算法被广泛应用于认证协议本身，同时也被用于建立会话密钥，但出于性能上的考虑，所有的数据流量都是用对称密钥密码算法（通常为 DES 和 AES）加密的。

为每个新连接选择一个新的随机选取的会话密钥，这样做的好处是最小化用户密钥或者公钥来发送的流量，从而减少了入侵者可能得到的密文数量，而且，如果进程崩溃或者核心转储落入敌手也可将损失降到最小。期望达到的效果是发生意外事件后唯一被暴露的密钥是会话密钥。当会话建立起来以后，所有的永久密钥应该谨慎地退出通信过程。

8.7.1 基于共享密钥的认证

在第一个认证协议中，我们假设 Alice 和 Bob 已经共享了一个密钥 K_{AB}。这个密钥有可能是通过电话或者面对面地建立起来的，但无论如何，肯定不是在（不安全的）网络上建立的。

这个协议以下面一个在许多其他的认证协议中都出现的原理为基础：一方给另一方发送一个随机数，后者将这个随机数做一个特殊的替代，再把结果返回给前者。这样的协议称为**质询-回应**（challenge-response）协议。在本小节的协议以及后续的认证协议中，我们将使用下面的记号：

A，B 是 Alice 和 Bob 的标识。

R_i 是质询，其中下标指明了发起挑战的一方。

K_i 是密钥，这里 i 代表密钥的所有者。

K_S 是会话密钥。

我们的第一个共享密钥认证协议中的消息序列如图 8-32 所示。在消息 1 中，Alice 以一种 Bob 能理解的方式将她的标识发送给 Bob。当然，Bob 无法判断这条消息来自 Alice 还是 Trudy，所以他选择发送一个质询，即一个大的随机数 R_B，并且它以明文方式发给 Alice，如图中的消息 2。然后 Alice 用她与 Bob 共享的密钥加密此消息，并且在消息 3 中将密文 $K_{AB}(R_B)$发送回来。当 Bob 看到这条消息时，他立即知道这条消息来自于 Alice，因为 Trudy 并不知道 K_{AB}，因此她不可能生成此消息。而且，由于 R_B 是在一个很大的空间中被随机选取的（比如说是一个 128 位的随机数），所以 Trudy 不太可能曾经在以前的会话中看到过 R_B 和相应的回应。同样地，她也不可能猜测到任何一个质询随机数所对应的正确回应。

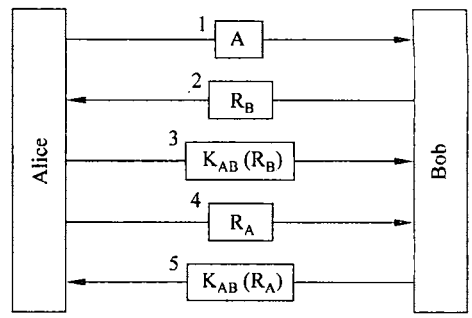


图 8-32 使用“质询-回应”的双向认证

这时候，Bob 已经确信与他通话的是 Alice，但是 Alice 还不能确定对方的身份。Alice 知道，Trudy 可能已经截取了消息 1 并送回了 R_B 作为回应。也许 Bob 昨天晚上已经死了。为了查清楚与她通话的到底是谁，Alice 选取了一个随机数 R_A，并以明文方式送给 Bob，见图中的消息 4。当 Bob 以 $K_{AB}(R_A)$作为回应的时候，Alice 知道她在跟 Bob 通话。如果他们现在希望建立一个会话密钥，则 Alice 可以选取一个 Ks，并且用 K_{AB} 加密之后发送给 Bob 即可。

图 8-32 中的协议包含了 5 条消息，我们来看是否可以让它更为精简一些，以便省下几条消息。图 8-33 显示了一种做法。在这里，Alice 主动发起了质询-回应协议，而不是等 Bob

发起。类似地，当 Bob 回应 Alice 的质询时，他也发送自己的质询消息。整个协议可以减少为 3 条消息，而不再是 5 条消息。

这个新的协议是否比原来的协议有所改进呢？从某种意义上讲，确实是因为新协议更短了。不幸的是，它也是错误的。在特定的条件下，Trudy 利用一种称为**反射攻击**（reflection attack）的技术可以使该协议失败。尤其是，如果 Trudy 能够与 Bob 一次打开多个会话，则 Trudy 就可以攻破这个协议。这种条件是有可能成立的，比如，Bob 是一家银行，他随时准备接受来自柜台机器的多个并发连接。

Trudy 的反射攻击如图 8-34 所示。首先，Trudy 声称她是 Alice，并且发送 R_T，然后 Bob 如同往常一样，用他自己的质询 R_B 作为回应。现在 Trudy 被定住了，她该怎么办呢？她并不知道 $K_{AB}(R_B)$。

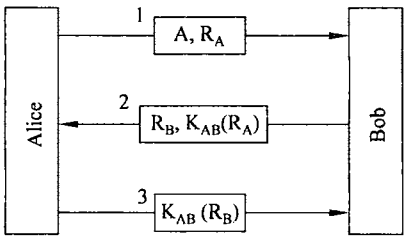


图 8-33 一个缩短的双向认证协议

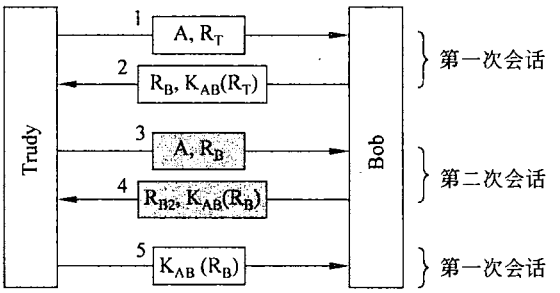


图 8-34 反射攻击

她可以利用消息 3 打开第二个会话，并且用消息 2 中的 R_B 作为她的质询。Bob 平静地对 R_B 进行加密，并在消息 4 中送回 $K_{AB}(R_B)$。我们为第二个会话上的消息加了阴影，以便突出显示它们。现在，Trudy 有了缺少的信息，所以她可以完成第一个会话，并放弃第二个会话。Bob 现在相信 Trudy 就是 Alice，所以当她请求查询 Alice 的银行账户余额时，Bob 毫无怀疑地告诉了她。然后，她请求 Bob 将所有的余额转到瑞士银行的一个秘密账户中，他也毫不犹豫地照办不误。

这个故事的寓意是：

设计一个正确的认证协议要比表面看上去的复杂得多。

下面 4 条通用规则通常有助于协议设计者避免陷入常见的陷阱：

（1）让发起方首先证明自己是谁，然后再轮到应答方。在上面的例子中，在 Trudy 给出证据来证明自己是谁之前 Bob 先给出了有价值的信息。

（2）让发起方和应答方使用不同的密钥作为证明，不过，这意味着要使用两个共享密钥 K_{AB} 和 K'_{AB}。

（3）让发起方和应答方从不同的集合中选取质询随机数。例如，发起方必须使用偶数，而应答方必须使用奇数。

（4）使协议能够抵抗这种涉及第二个并行会话的攻击，比如像上面给出的反射攻击那样，在一个会话中得到的信息可以用在另一个会话中。

只要违反了以上规则中的任何一条，则协议通常就会被攻破。在前面的例子中，所有 4 条都违反了，所以带来了灾难性的后果。

现在我们回去再仔细看图 8-32。这个协议真的不会遭受反射攻击吗？难说，那要看情

况了。情况很微妙。Trudy 能够用反射攻击来挫败我们的协议是因为她有可能打开第二个与 Bob 之间的会话，然后诱使他回答自己的问题。如果 Alice 是一台能够接受多个会话的通用计算机，而不是计算机前面的一个人，那会怎么样呢？现在我们来看 Trudy 能做些什么。

为了看清楚 Trudy 的攻击是如何实施的，请参考图 8-35。Alice 首先在消息 1 中宣告自己的标识。Trudy 截取了这条消息，并开始她自己的会话，声称自己是 Bob，如图中的消息 2 所示。同样地，我们用阴影来表示第二个会话的消息。Alice 在消息 3 中回应了消息 2，她这样说：你是 Bob？请证明这一点。这时候，Trudy 被定住了，因为她无法证明自己是 Bob。

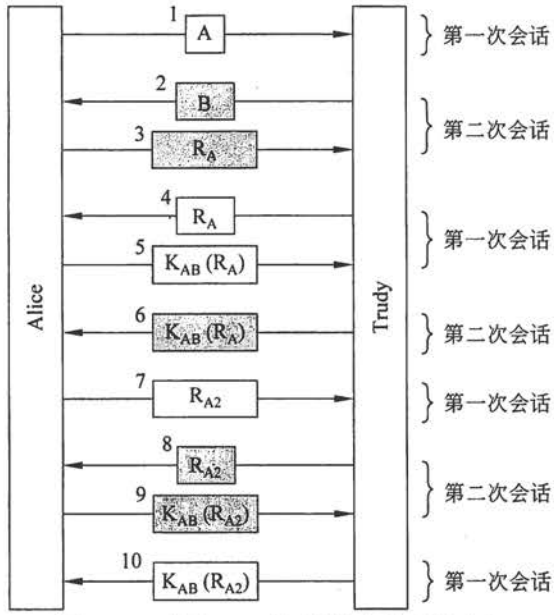


图 8-35　对图 8-32 所示的协议的反射攻击

现在 Trudy 该怎么办呢？她又回到第一个会话中，在这里，该轮到她发送一个质询，于是，她把在消息 3 中得到的 R_A 发送出去。Alice 非常友好地做了回应，即消息 5，从而给 Trudy 提供了她所急需的信息，所以，Trudy 发送了消息 6，此消息属于会话 2。此时，Trudy 基本上畅行无阻了，因为在会话 2 中，她已经成功地回应了 Alice 的质询。现在她可以取消会话 1 了，对于会话 2 的剩余部分，她发送任何一个老的数值，因此，她获得了一个与 Alice 之间经过认证的会话，即会话 2。

但是 Trudy 比我们想象的还要恶劣，她想彻底地攻进去并且不留下任何异常。她并不是马上发送一个老的数值来完成会话 2，而是等待 Alice 发送消息 7，即会话 1 中 Alice 的质询随机数。当然，Trudy 不知道该如何回应，所以她再次利用反射攻击，送回 R_{A2}，即消息 8。Alice 很自然地加密了 R_{A2}，并通过消息 9 送给 Trudy。Trudy 现在切换到会话 1，并且在消息 10 中把 Alice 想要的加密结果送给她，实际上她只是将 Alice 在消息 9 中发送过来的加密结果复制回去而已。到这时候，Trudy 与 Alice 之间有了两个完全经过认证的会话。

与图 8-34 中显示的针对三次消息协议的攻击相比较，上面的攻击有一个略微不同的结

果。这一次，Trudy 与 Alice 之间有了两个经过认证的连接。而在前一个例子中，她与 Bob 之间只有一个经过认证的连接。同样在这里，如果我们应用了前面讨论过的全部通用认证协议规则，那么，这种攻击就可以被制止。有关这些攻击以及如何阻止这些攻击的详细讨论，请参考（Bird 等，1993）。他们还证明了：完全有可能用系统化的方法来构造一个可被证明是正确的协议。然而，对于这样的协议，即使是最简单的形式也较为复杂，所以，我们现在来介绍他们提出的另外一类同样可以工作的协议。

新的认证协议如图 8-36 所示（Bird 等，1993）。它使用了 HMAC，我们曾经在学习 IPSec 时了解过它。Alice 首先给 Bob 发送一个临时值（nonce）R_A 作为消息 1。Bob 的回应是：选择他自己的临时值 R_B，并连同一个 HMAC 一起发送回去。这里的 HMAC 是这样形成的：首先建立一个数据结构，其中包含了 Alice 的临时值、Bob 的临时值、他们的标识，以及共享密钥 K_{AB}；然后将这个数据结构做**散列**（hash）运算，比如使用 SHA-1 算法，散列的结果即为 HMAC。当 Alice 接收到消息 2 时，她现在拥有 R_A（这是她自己选取的）、R_B（这是以明文形式传来的）、双方的标识，以及密钥 K_{AB}（这是她一直知道的），所以她自己也可以计算出 HMAC。如果她计算的 HMAC 与消息中的 HMAC 一致，那么她知道她在与 Bob 通话，因为 Trudy 不知道 K_{AB}，因此 Trudy 不可能计算出该发送哪一个 HMAC。Alice 回应给 Bob 的也是一个 HMAC，但是此 HMAC 仅仅包含两个临时值和 K_{AB}。

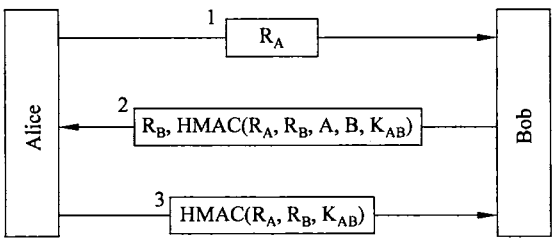


图 8-36 使用 HMCA 的认证

那么，Trudy 能够破坏这个协议吗？不能，因为她不可能像图 8-34 和图 8-35 中那样，强迫任何一方按照她的选择来加密或者散列一个值。这里的两个 HMAC 中包含了双方选择的值，其中部分值是 Trudy 所无法控制的。

采用 HMAC 并不是体现这种思想的唯一做法。除了在一组数据项上计算 HMAC 这种做法以外，另一种常用的方案是利用**密码块链**（cipher block chaining）模式顺序地加密所有的数据项。

8.7.2 建立共享密钥：Diffie-Hellman 密钥交换

到现在为止，我们一直假定 Alice 和 Bob 共享一个密钥。现在假设他们之间并没有共享密钥（因为至今还没有一个被普遍接受的 PKI 可用于签名和分发证书）。他们如何才能建立一个共享密钥呢？一种办法是，Alice 打电话给 Bob，通过电话将她的密钥告诉 Bob，但是 Bob 可能一上来就这样说：我怎么知道你是 Alice 而不是 Trudy 呢？他们可能会试着安排一次会面，每个人都带上护照、驾驶证和三个主要的信用卡，但如果他们都很忙碌的话，他们有可能几个月都找不到一个双方都能接受的会面日期。幸运的是，虽然听起来有

点难以置信，但是存在一种办法可以让完全陌生的人在完全公开的情况下建立起一个共享的密钥，即使 Trudy 小心地记录下每一条消息也无妨。

用在陌生人之间建立共享密钥的协议称为 **Diffie-Hellman 密钥交换协议**（Diffie-Hellman key exchange）（Diffie 和 Hellman，1976），其工作过程如下所述。Alice 和 Bob 必须就两个大数 n 和 g 达成一致，这里 n 是一个素数，(n-1)/2 也是一个素数，并且 g 需要满足一些特殊的条件。这些数可以是公开的，所以，他们两人中的任何一个选取 n 和 g，并且通过公开的方式告诉另一个人。现在 Alice 选择一个大数 x （比如说 1024 位），并将它保密；类似地，Bob 也秘密地选择一个大数 y。

Alice 发起密钥交换协议，她给 Bob 发送一条消息，其中包含了$(n, g, g^x \bmod n)$，如图 8-37 所示。Bob 给 Alice 发送一条包含了$(g^y \bmod n)$的消息作为应答。现在，Alice 把 Bob 发送给自己的数计算 x 次乘方并且以 n 为模，得到$(g^y \bmod n)^x \bmod n$。Bob 也执行类似的计算，得到$(g^x \bmod n)^y \bmod n$。根据模算术定理，双方的计算结果都是$g^{xy} \bmod n$。你瞧，就像变魔术一样，Alice 和 Bob 突然之间共享了一个密钥，即$g^{xy} \bmod n$。

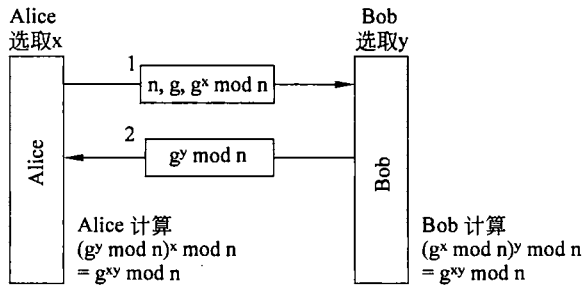


图 8-37 Diffie-Hellman 密钥交换

Trudy 当然也看到了这两条消息。她从消息 1 中知道了 g 和 n。如果她能够计算出 x 和 y，那么，她就可以得到密钥。问题在于，仅仅知道了$g^x \bmod n$，她不可能找到 x。在以一个非常大的素数为模的情况下，计算离散对数的实用算法目前尚未发现。

为了使上面的例子更加具体化，我们假设 n=47 和 g=3（这些值完全是不切实际的）。Alice 选取 x=8，Bob 选取 y=10。这两个数值都是保密的。Alice 发给 Bob 的消息是(47, 3, 28)，因为$3^8 \bmod 47$是 28。Box 发给 Alice 的消息是（17）。Alice 计算$17^8 \bmod 47$，等于 4。Bob 计算$28^{10} \bmod 47$，也等于 4。所以，Alice 和 Bob 现在单独地确定了密钥为 4。为了找到密钥，Trudy 必须要解方程式：$3^x \bmod 47=28$，对于像这里给出的小数值，利用穷举搜索可以解出 x 的值，但是当所有这些数值都是几百位长时，这种做法就行不通了。所有当前已知的算法都需要太长的时间才能解出 x，即使在大型的并行超级计算机上也是如此。

尽管 Diffie-Hellman 算法非常优美，但是它也存在一个问题：当 Bob 得到了三元组（47, 3, 28）时，他怎么知道该消息来自 Alice 而不是 Trudy？他没有办法判断这一点。不幸的是，Trudy 正好可以利用这个事实来同时欺骗 Alice 和 Bob，过程如图 8-38 所示。这里，在 Alice 和 Bob 分别选择了 x 和 y 时，Trudy 也选取了她自己的随机数 z。Alice 发送消息 1 给 Bob，但是 Trudy 截获了此消息，并且发送消息 2 给 Bob；在消息 2 中，Trudy 使用了正确的 g 和 n（不管怎么样，这两个数都是公开的），但是她用自己的 z 来代替 x。她还将消息 3 送回给 Alice。后来，当 Bob 给 Alice 发送消息 4 时，Trudy 再次截获此消息，并保留

不转发。

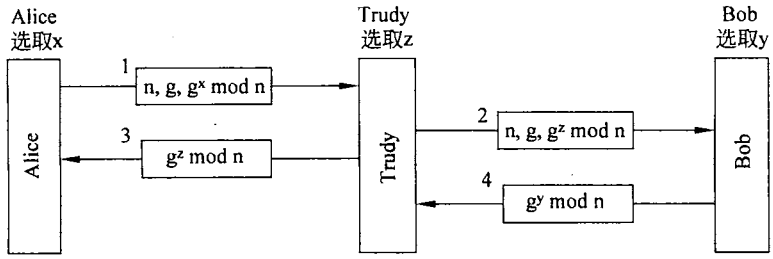


图 8-38 中间人攻击

现在，每个人都做模算术运算。Alice 计算出密钥为 g^{xz} mod n，Trudy 也一样（针对所有发送给 Alice 的消息）。Bob 计算出 g^{yz} mod n，Trudy 也做同样的计算（针对所有发送给 Bob 的消息）。Alice 认为她在跟 Bob 通话，所以她建立一个会话密钥（实际上，与 Trudy 共享此会话密钥）。Bob 也如此。Alice 在此加密会话上发送的所有消息都被 Trudy 捕获到，并存储下来，如果愿意的话做适当修改，然后（可选地）传递给 Bob。在另一个方向上也类似。Trudy 看到了所有的秘密，而且可以随意地改动所有的消息，而 Alice 和 Bob 都错误地认为自己与对方有一条安全的信道。基于这个原因，这种攻击也称为**中间人攻击**（man-in-the-middle attack），或者称为**水桶队列攻击**（bucket brigade attack），因为它有点类似于以前的义务消防队的做法，他们从救火车到着火点排成一队传递水桶。

8.7.3 使用密钥分发中心的认证

与陌生人建立共享密钥差不多可以工作了，但是并不完全解决问题。另外，有可能从一开始就不值得这么做（酸葡萄攻击）。若通过这种方式与 n 个人进行通话，那么你将需要 n 个密钥。对于普通人来说，密钥管理将变成一份实实在在的负担，尤其当每个密钥都必须被存储在一个单独的塑料芯片卡上的时候。

另一种不同的方法是引入一个可信的密钥分发中心（KDC）。在这种模型中，每个用户与 KDC 共享一个密钥。现在认证和会话密钥管理都通过 KDC 来完成。图 8-39 显示了目前已知的最简单的 KDC 认证协议，它涉及两方和一个可信的 KDC。

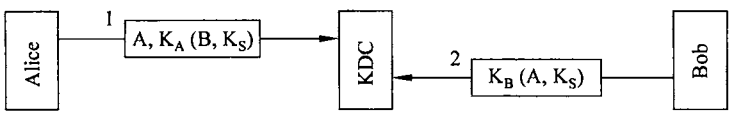


图 8-39 在认证协议中首次使用 KDC 的尝试

这个协议背后的思想非常简单：Alice 选取一个会话密钥 K_S，并且告诉 KDC 她希望利用 K_S 与 Bob 进行通话。这条消息用 Alice 与 KDC 之间的共享密钥 K_A（只有 Alice 和 KDC 才知道）加密。KDC 解密此消息，并提取出 Bob 的标识和会话密钥。然后它构造一条新的消息，其中包含了 Alice 的标识和会话密钥，并且它将这条消息发送给 Bob。这次用 Bob 与 KDC 之间的共享密钥 K_B 加密。当 Bob 解密出该消息时，他就知道 Alice 想与他通话，并且她希望使用 K_S 作为会话密钥。

这里的认证是自然而然的。KDC 知道消息 1 一定来自 Alice，因为其他人不可能用 Alice

的密钥来加密消息。类似地，Bob 知道消息 2 一定来自于 KDC（而 Bob 对 KDC 是信任的），因为其他没有人知道他的密钥。

不幸的是，这个协议有一个严重的缺陷。Trudy 需要一些钱，所以她想出一些自己可以帮助 Alice 处理的合法业务，向 Alice 提出优厚的条件，最后得到了这份工作。在完成了工作以后，Trudy 礼貌地请求 Alice 通过银行转账来支付费用。然后 Alice 与她的银行服务员 Bob 建立一个会话密钥。之后她给 Bob 发送一条消息，请求将一笔钱转到 Trudy 的账户中。

同时，Trudy 又故伎重演，她在网络上窃听流量，并且将图 8-39 中的消息 2 和随后的转账请求消息复制下来。稍后，她又把这两条消息重新发送给 Bob。Bob 收到这两条重放消息后，他会这样想：Alice 一定又雇用了 Trudy，很显然 Trudy 干得不错。于是，Bob 又一次将同样数额的一笔钱从 Alice 的账户转到 Trudy 的账户上。经过了 50 次这样的消息重放以后，Bob 走出办公室并找到 Trudy，表示愿意为她提供一大笔贷款，以便 Trudy 能够拓展她那看起来显然很成功的业务。这个问题称为**重放攻击**（replay attack）。

针对重放攻击，有几种可能的解决方案。第一种方案是在每一条消息中包含一个时间戳。之后，如果有人接收到一条过期的消息，丢弃此消息即可。这种做法的麻烦之处在于，网络上的时钟从来都不是精确同步的，所以，总是存在某一个时间间隔使得在此期间时间戳是有效的。Trudy 就可以在这段时间间隔中重放消息，逃脱时间戳的检查。

第二种方案是在每条消息之中放置一个临时值。然后，每一方必须要记住所有此前已经出现过的临时值，如果一条消息包含了以前用过的临时值，那么拒绝该消息。但是，这些临时值必须被永久地记录下来，以免 Trudy 可能会试图重放 5 年前的消息。而且，如果某台机器崩溃之后丢失了所有的临时值信息，那么它又会再次遭受重放攻击。我们可以将时间戳和临时值结合起来，以便缩短需要记录临时值的时间范围，但是很显然这将会导致协议变得非常复杂。

一种较为复杂的实现双向认证的做法是使用多路径的质询-回应协议。这种协议的一个著名的例子是 Needham-Schroeder 认证协议（Needham 和 Schroeder，1978），图 8-40 显示了它的一种变化形式。

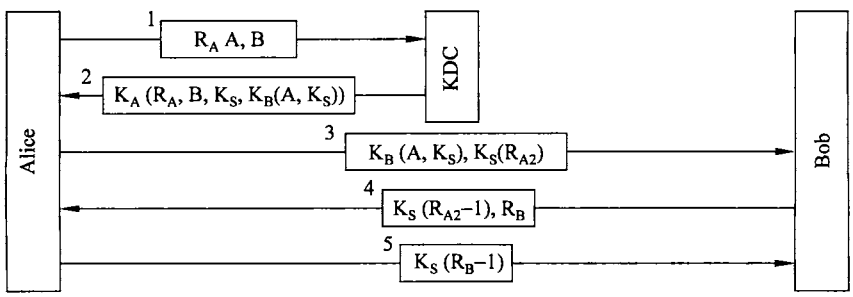


图 8-40 Needham-Schroeder 认证协议

协议开始时，Alice 首先告诉 KDC 她想与 Bob 通话。这条消息包含一个大的随机数 R_A 作为临时值。KDC 发回消息 2，其中包含了 Alice 的随机数、一个会话密钥，以及一个她将来可以发送给 Bob 的票据。这里使用随机数 R_A 是为了使 Alice 相信消息 2 是最新的，而不是重放消息。Bob 的标识也被包含在消息中，这是为了防止 Trudy 用她自己的标识来

替换消息 1 中的 B，从而导致 KDC 用 K_T 而不是 K_B 来加密消息 2 末尾的票据；经 K_B 加密的票据也被包含在加密的消息中，这是为了防止在返回到 Alice 的途中，Trudy 用其他的数据来偷换此票据。

Alice 现在将票据以及一个用会话密钥 K_S 加密的新随机数 R_{A2} 发送给 Bob。在消息 4 中，Bob 送回 $K_S(R_{A2}-1)$，以此向 Alice 证明，她在跟真正的 Bob 通话。如果直接送回 $K_S(R_{A2})$ 是没有用的，因为 Trudy 可以直接从消息 3 中偷到 $K_S(R_{A2})$。

Alice 在接收到消息 4 以后，她现在确信自己正在跟 Bob 通话，而且到现在为止没有任何重放消息。毕竟她仅仅在几毫秒之前才产生了 R_{A2}。消息 5 的用途是为了使 Bob 确信，现在与他通话的是真正的 Alice，而且这里也没有出现任何重放消息。在这个协议中，每一方都生成一个质询，并且回应一个质询，因而消除了任何一种重放攻击的可能性。

尽管这个协议看起来似乎非常坚固，但它确实有一个微小的弱点。如果 Trudy 曾经设法获得过一个老的会话密钥的明文，那么她可以向 Bob 发起一个新的会话，具体做法是把对应于泄漏密钥的消息 3 重放给 Bob，使 Bob 相信她是真正的 Alice（Denning 和 Sacco，1981）。现在，她甚至一次都不用为 Alice 提供合法业务就可以抢劫她的银行账户了。

Needham 和 Schroeder 后来发表的一个协议改正了这个问题（Needham 和 Schroeder，1987）。在同一份杂志的同一期上，（Otway 和 Rees，1987）也发表了一个同样解决这个问题的协议，而且更加简短。图 8-41 显示了一个略作修改之后的 Otway-Rees 协议。

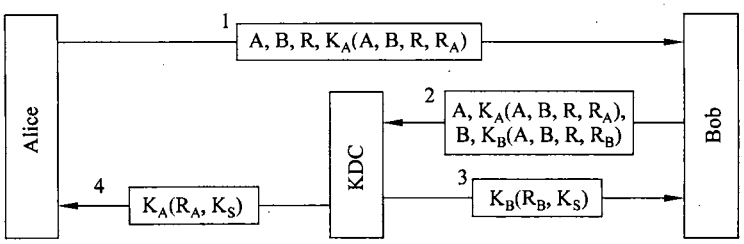


图 8-41 稍微被简化的 Otway-Rees 认证协议

在 Otway-Rees 协议中，Alice 首先生成一对随机数 R 和 R_A，这里 R 被用作一个公共的标识符，而 R_A 则是 Alice 质询 Bob 的随机数。当 Bob 得到了这条消息时，他根据 Alice 消息中的加密部分构造一个新的消息，以及一条用他自己的信息构造出来的类似消息。用 K_A 和 K_B 加密的部分标识了 Alice 和 Bob，同时也包含了公共的标识符和一个质询随机数。

KDC 对这两部分中的 R 进行检查，看它们是否相同。它们有可能不相同，比如 Trudy 篡改了消息 1 中的 R，或者替换了消息 2 中的部分数据。如果两个 R 一致，则 KDC 相信 Bob 的请求消息是有效的。然后它生成一个会话密钥，并且对它加密两次，一次为 Alice，另一次为 Bob。在它发送给 Alice 和 Bob 的两条消息中，每条消息都包含了接收方的随机数，以此来证明这是 KDC 而不是 Trudy 生成的消息。这时候 Alice 和 Bob 拥有了同一个会话密钥，于是他们便可以开始通信了。当他们第一次交换数据消息时，每一方都看到对方有一个完全相同的 K_S，所以认证过程便完成了。

8.7.4 使用 Kerberos 的身份认证

许多实际系统（包括 Windows 2000 以及后续版本）使用的认证协议是 Kerberos，它以

Needham-Schroeder 协议的一种变化形式为基础。Kerberos 得名于古希腊神话中一头专门守卫地狱大门的多头狗（大概它也被禁止离开地狱）。Kerberos 由 MIT 设计，设计目标是允许工作站用户以一种安全的方式访问网络资源。它与 Needham-Schroeder 协议的最主要不同之处在于，它假设所有的时钟都取得良好同步。Kerberos 协议经历了几次改版，其中第5版已被广泛应用于工业界，并且标准化为 RFC 4120。较早的第4版在出现严重问题后最终退役（Yu 等，2004）。第5版在第4版的基础上，对协议和功能方面做了很多小的改变，例如它再也不基于现在已经过时的 DES。有关更多的信息，请参看（Neuman 和 Ts'o，1994）。

除了 Alice（一个客户工作站）以外，Kerberos 还涉及3个服务器：

（1）认证服务器（AS）：在用户登录过程中验证用户的身份。

（2）票据发放服务器（TGS，Ticket-Granting Server）：发放“身份票据的证明”。

（3）Bob 服务器：真正完成 Alice 所请求的工作。

AS 与 KDC 非常类似，它与每个用户共享一个秘密口令。TGS 的任务是发放票据，这个票据可使真正的服务器相信：TGS 票据的持有者确实是他或她所声称的那一位。

为了开始一个会话，Alice 在任意一台公共的工作站前面坐下来，输入她的名字。该工作站将她的名字和 TGS 名字以明文方式发送给 AS，如图 8-42 中所示的消息 1。AS 发送回来的是一个会话密钥和一个替代 TGS 的票据 $K_{TGS}(A, K_S, t)$。会话密钥使用 Alice 的密钥加密，因此唯有 Alice 才能解密此消息。只有当消息 2 到达后，工作站才会请求 Alice 输入口令。然后工作站利用该口令生成 K_A，来解密消息 2 获得会话密钥。

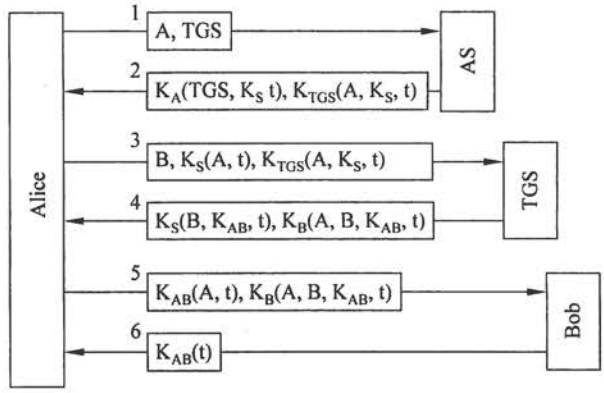


图 8-42 Kerberos 第5版的操作

这时候，工作站覆盖掉 Alice 的口令，以确保它只会在工作站里最多停留几毫秒钟的时间。如果 Trudy 试图以 Alice 的身份登录，那么，她输入的密码将是错误的，工作站将会检测到这种错误，因为消息 2 的标准部分是不正确的。

当 Alice 登录后，她可能告诉工作站，她想要联系 Bob 文件服务器。然后工作站给 TGS 发送消息3，请求一个使用 Bob 文件服务器的票据。这个请求中的关键要素是 $K_{TGS}(A, K_S, t)$，它用 TGS 的密钥加密的，它的用途是证明发送方真的是 Alice。TGS 的回应是，创建并返回一个会话密钥 K_{AB}，供 Alice 和 Bob 通信时使用。该会话密钥的两个版本都被发送回来，第一个版本仅仅用 K_S 加密，所以 Alice 可以读取它；第二个版本是另一个票据，用 Bob 的密钥 K_B 加密，所以 Bob 可以读取它。

Trudy 可以复制消息3，并再次发送此消息，但是，她将会失败，因为连同消息3一起

发送的还有一个加密时间戳 t。Trudy 不可能用一个更新的时间戳来代替 t，因为她不知道 K_S，即 Alice 用来与 TGS 通话的会话密钥。即使 Trudy 极为快速地重发了消息 3，她所得到的也只不过是消息 4 的另一份副本而已，既然第一次她不能解密消息 4，那么第二次她同样也不能解密。

现在 Alice 可以通过该新票据将 K_{AB} 发送给 Bob 以便与他建立一个会话（消息 5）。这次消息交换也使用了时间戳。可选的 Bob 的响应（消息 6）向 Alice 证明了，她确实在与 Bob 而不是 Trudy 通话。

经过这一系列的消息交换以后，Alice 就可以在 K_{AB} 的保护下与 Bob 进行通信了。如果她后来决定要跟另一个服务器 Carol 进行通话，那么她只要向 TGS 再次发送消息 3，只不过这一次指定 C 而不是 B。TGS 将迅速地回送一个用 K_C 加密的票据，以后 Alice 可以将此票据发送给 Carol，而 Carol 则凭此票据相信她收到的消息确实来自 Alice。

所有这些工作的要点在于现在 Alice 可以以一种安全的方式来访问网络上所有的服务器了，而且她的口令永远都不会出现在网络上。事实上，即使在她自己的工作站上，口令也只是停留了几毫秒而已。然而，请注意，每个服务器都有它自己的授权规则。当 Alice 向 Bob 出示她的票据时，此票据仅仅向 Bob 证明了这是谁发送过来的。至于到底允许 Alice 做什么事情，则要取决于 Bob。

由于 Kerberos 的设计者们并不期望全世界都信任同一个认证服务器，所以他们提供了对多个域（realm）的支持，每个域有自己的 AS 和 TGS。为了得到一个针对远程域中某个服务器的票据，Alice 需要向她自己所在域的 TGS 请求一个票据，该票据必须能被远程域 TGS 接受。如果远程 TGS 已经向本地的 TGS 注册过了（其做法就如同本地服务器的注册方法一样），则本地 TGS 将给 Alice 一个可用于远程 TGS 的有效票据。然后她就可以在那边（即远程域）完成各种事务，比如获取针对该域中各个服务器的票据。然而，请注意，为了让两个域中的多个参与方共同完成一些工作，每个域必须信任另一个域的 TGS。否则，它们将无法工作。

8.7.5 使用公开密钥密码学的认证

可以利用公开密钥密码学来完成双向认证。首先，Alice 需要得到 Bob 的公钥。如果有一个 PKI 目录服务器可以提供公钥证书的查询服务，那么 Alice 可以请求 Bob 的公钥证书，如图 8-43 中的消息 1。消息 2 中的应答是一个 X.509 证书，其中包含了 Bob 的公钥。当 Alice 验证了证书中的签名无误后，她给 Bob 发送一条消息，该消息包含了她的标识和一个临时值。

当 Bob 接收到这条消息时，他无从判断这条消息到底来自于 Alice 还是 Trudy，但是他可以等待一下，并且向目录服务器请求 Alice 的公钥（消息 4），很快他便获得了 Alice 的公钥（消息 5）。然后，他给 Alice 发送一条消息，即图中的消息 6，其中包含了 Alice 的 R_A、Bob 自己的临时值 R_B 和一个建议的会话密钥 K_S。

当 Alice 得到了消息 6 以后，她用自己的私钥将它解密。她看到了里边的 R_A，这让她有一种很温暖的感觉。这条消息一定是 Bob 发送过来的，因为 Trudy 无法确定 R_A。而且，它也一定是最新的，而不是一条重放消息，因为她刚刚把 R_A 发送给 Bob。Alice 发回消息

7 表示她同意这次会话。当 Bob 看到了 R_B 是用他刚刚生成的会话密钥来加密时，他知道 Alice 已经得到了消息 6 并且验证了 R_A。现在，Bob 心满意足了。

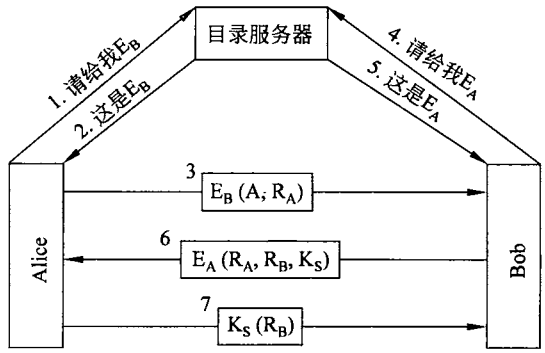


图 8-43 使用公开密钥密码学的双向认证

Trudy 怎么样才能破坏这个协议呢？她可以虚构消息 3，以此来引诱 Bob 向 Alice 发送消息 6，但是 Alice 将会发现这个 R_A 不是她发送的，所以不会再进行下去。Trudy 不可能伪造出 Alice 发送回给 Bob 的消息 7，因为她不知道 R_B 或者 K_S，在没有 Alice 私钥的情况下，她是不可能确定 R_B 和 K_S 的。因此，面对这个协议时她的运气就太不好了。

8.8 电子邮件安全性

当电子邮件消息在两个远程站点之间传递时，一般情况下该消息沿途要经过几十台机器。这些机器中的任何一台都可以阅读和记录下该消息，以备将来可能之用。实际上，不管人们怎么想，隐私是不存在的。然而，许多人希望自己发送的电子邮件只有目标接收者才能阅读，其他人都无法阅读，即使他们的老板或者他们的政府也不能阅读。这种愿望刺激了一些人和组织将前面学习过的密码学原理应用到电子邮件上，从而形成安全的电子邮件。在本节中，我们将首先学习一个已经得到广泛应用的安全电子邮件系统：PGP，然后再简短介绍另外一个安全电子邮件系统 S/MIME。有关安全电子邮件的更多信息，请参看（Kaufman 等，2002）和（Schneier，1995）。

8.8.1 PGP——良好的隐私性

我们的第一个例子**良好的隐私性**（PGP，Pretty Good Privacy）本质上是 Zimmermann 一个人的心血结晶（Zimmermann，1995a，1995b）。他是一个极其崇尚隐私的人，他的人生格言是："如果隐私是非法的，那么只有违法者才会有隐私"。PGP 发布于 1991 年，它是一个完整的电子邮件安全软件包，提供了私密性、认证、数字签名和压缩功能，而且所有这些功能都非常易于使用。此外，完整的软件包中也包含了所有的源代码，并通过 Internet 免费分发。由于 PGP 的质量、价格（0），以及在 UNIX、Linux、Windows 和 Mac OS 平台上的易用性，今天 PGP 已得到了广泛的使用。

PGP 使用一个称为**国际数据加密算法**（IDEA，International Data Encryption Algorithm）

的块密码算法来加密数据。该算法使用128位密钥，它在瑞士被设计出来，正值DES被认为不够安全而AES尚未发明之际。概念上讲，IDEA非常类似于DES和AES：它也运行许多轮，在每一轮中将数据位尽可能地混合起来，但是，它的混合函数与DES和AES的不同。PGP的密钥管理使用RSA，数据完整性使用MD5，这些话题我们前面都已经讨论过了。

PGP自从诞生以来，就被卷入到一场持久的争论之中（Levy，1993）。因为Zimmermann并没有阻止其他人把PGP放到Internet上，全世界的任何人都可以得到PGP，因此，美国政府声称Zimmermann违反了美国的军用武器出口法。美国政府对Zimmermann进行了长达5年的调查，但是最终放弃了，这里可能有两个原因。第一，Zimmermann自己并没有把PGP放到Internet上，所以他的律师声称他没有出口任何东西（至于是否创建了一个出口物品的Web站点则是一个小问题了）。第二，政府最终意识到，要想赢得这个案子，意味着要使陪审团相信：一个可下载隐私程序的Web站点也应该被列入军火交易法的适用范围内，因而对Web站点的限制就如同禁止出口战争武器，比如坦克、潜艇、军用飞机和核武器。而且多年来的反面宣传可能没有多大帮助。

顺便提一下，将出口法应用在PGP上是非常不恰当的。美国政府认为把代码放在一个Web站点上是一种非法的出口，并为此侵扰了Zimmermann达5年之久。另外，如果某个人以一本书的形式发表PGP完整的C代码（采用比较大的字体，并且每一页带一个校验和，因此扫描识别非常容易），然后出口这本书，那么，政府认为这没有什么关系，因为图书并不是军品。剑利于笔，至少山姆大叔是这样认为的。

PGP的另一个问题是它牵涉到了专利侵犯案。拥有RSA专利的公司是RSA Security公司，它宣称PGP使用RSA算法侵犯了它的专利，但是2.6以后的版本就不存在这个问题了。此外，PGP还使用了另一个受专利保护的加密算法，即IDEA，起初使用也曾经引起了一些问题。

由于PGP是源码开放的，所有的人和组织都可以对它进行修改，再产生各种版本。其中有些版本的设计目标是为了回避军品出口法，而其他有些版本则是为了避免使用专利算法，另外还有一些版本则是为了将它变成一个源码封闭的商业产品。虽然军品出口法现在已有所放宽（否则，使用AES的产品将不可能从美国出口），而且，RSA的专利也已于2000年9月到期，但是，所有这些问题遗留下来的结果是有许多不兼容的PGP版本流传在各处，而且名字也不尽相同。下面的讨论将集中在经典的PGP上，这是最老的也是最简单的版本。另一个流行的版本是OpenPGP，由RFC 2440描述。还有一个版本是GNU Privacy Guard。

PGP特意使用已有的密码学算法，而不是发明新的算法。它主要基于那些已经经受住了广泛审查并且未受任何政府机构影响（政府机构总是试图削弱它们）的算法，对于那些不太信任政府的人来说，这个特性是一个很大的吸引点。

PGP支持正文压缩、私密性和数字签名，也提供了可扩展的密钥管理工具，但奇怪的是，它没有提供电子邮件工具。它就像一个预处理器，接受明文输入，并产生签过名的密文作为输出，其输出格式为Base64。然后这个输出被电子邮件程序发送出去。有些PGP实现的最后一步调用用户代理，以便真正将邮件发送出去。

为了看清楚PGP是如何工作的，我们考虑图8-44中的例子。这里，Alice想要通过一

种安全的方式给 Bob 发送一个签名的明文邮件 P。Alice 和 Bob 都有私有（D_X）和公开（E_X）RSA 密钥。我们假设每一方都知道对方的公钥；我们将扼要地介绍 PGP 的密钥管理机制。

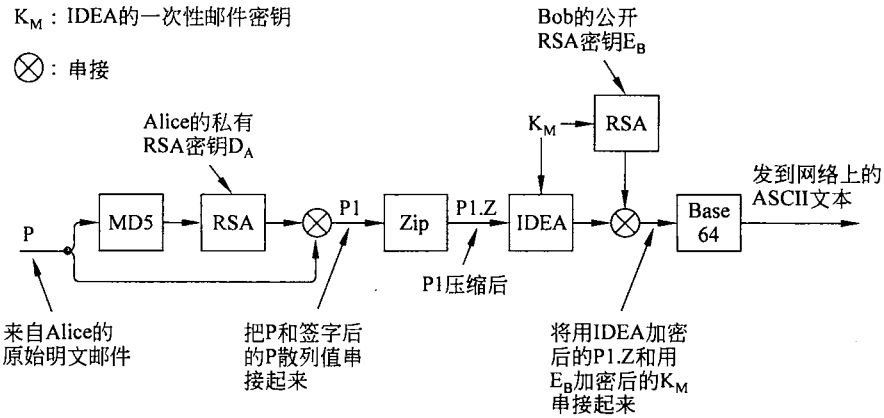


图 8-44 在发送消息的操作中的 PGP

作为开始，Alice 调用她自己机器上的 PGP 程序。PGP 首先使用 MD5 算法对她的消息 P 做散列运算，然后用 Alice 的私有 RSA 密钥 D_A 加密此结果散列值。当 Bob 最终得到该消息时，他可以用 Alice 的公钥解密此散列值，并且验证散列值是否正确。即使此时其他人（比如 Trudy）也获得这个散列值，并且也可以用 Alice 的公钥对它解密，但 MD5 的强度保证了要产生另一个具有相同 MD5 散列值的消息在计算上是不可行的。

经过加密的散列值和原始的消息现在被串接成单条消息 P1 中，并且使用 ZIP 程序进行压缩。ZIP 程序使用了 Ziv-Lempel 算法（Ziv 和 Lempel，1977）。我们将这一步的输出结果称为 P1.Z。

接下来，PGP 提示 Alice 输入一些随机的信息。Alice 输入的内容和敲键的速度被用来生成一个 128 位的 IDEA 邮件密钥 K_M（在 PGP 的文献中该密钥称为会话密钥，但是，这实际上并不太恰当，因为这里没有会话）。现在按照密码反馈模式，利用 IDEA 算法和 K_M 来加密 P1.Z。另外，使用 Bob 的公钥 E_B 来加密 K_M。然后，这两部分内容被串接起来，并转换成 Base64 编码，正如我们在第 7 章中讨论 MIME 时所介绍的那样。结果邮件只包含字母、数字和符号+、/和=，这意味着该邮件可以被放到一个 RFC 822 邮件体中，并且有望未被修改地到达另一方。

当 Bob 得到此邮件时，他做一个逆向的 Base64 编码，并且利用他自己的私有 RSA 密钥解密出 IDEA 密钥。利用这个密钥，他解密此邮件得到 P1.Z；再经过解压缩，Bob 将明文与加密的散列值分离开，并且使用 Alice 的公钥解密此散列值。如果得到的明文散列值与他自己计算的 MD5 散列值一致，那么，他知道 P 是正确的邮件，而且它确实来自于 Alice。

值得注意的是，在这里只有两个地方用到了 RSA：为了加密 128 位的 MD5 散列值，以及加密 128 位 IDEA 密钥。虽然 RSA 非常慢，但是它只需加密 256 位数据，而不是大量的数据。而且，所有这 256 位明文数据是高度随机的，因此 Trudy 需要做相当大量的工作才能确定一个猜测出来的密钥是否正确。在 PGP 的操作过程中，繁重的加密工作是由 IDEA 来完成的，而 IDEA 比 RSA 快了几个数量级。因此，PGP 不仅提供了安全性、压缩和数字

签名的功能，而且它完成这些功能的效率比图 8-19 中显示的方案要高得多。

PGP 支持 4 种 RSA 密钥长度。它可以由用户选择一种最为合适的长度。这些长度选项是：

（1）临时的（384 位）：今天很容易被破解。

（2）商用的（512 位）：可被三字母组织破解。

（3）军用的（1024 位）：地球上的任何人都无法破解。

（4）星际的（2048 位）：其他行星上的任何人也无法破解。

由于 RSA 仅仅被用在两个小的计算过程中，所以，每个人在任何时候都应该使用星际强度的密钥。

经典 PGP 消息的格式如图 8-45 所示。在实际中还使用了其他许多格式。图中显示的消息有 3 个部分，分别包含了 IDEA 密钥、签名和消息体。密钥部分包含的不仅仅是密钥，还有一个密钥标识符，因为 PGP 允许每个用户有多个公钥。

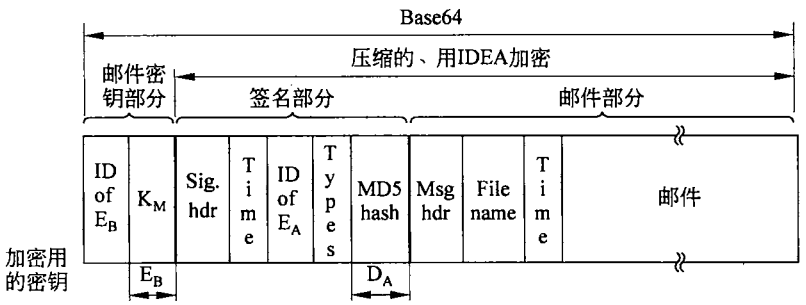


图 8-45 一个 PGP 邮件

签名部分包含一个头，这里我们并不关心这个头。头的后面紧跟一个时间戳、发送方公钥的标识符、某些类型信息以及加密的散列值本身。其中，公钥标识符指定了用来解密签名散列值的公钥，而类型信息则标识了所使用的算法（当 MD6 和 RSA2 发明出来时可以直接支持它们）。

邮件部分也包含一个头，以及一个默认的文件名，如果接收方要将文件写到磁盘中的话，则可以使用此默认的文件名。另外，邮件部分还包含了一个用来标识该邮件被创建时刻的时间戳，最后是邮件本身。

在 PGP 中，密钥管理部分最为受到关注，因为它是所有安全系统的要害所在。PGP 的密钥管理模型如下所述。每个用户在本地维护两个数据结构：一个私钥环和一个公钥环。**私钥环**（private key ring）包含一个或者多个本人的“公钥/私钥”对。为每个用户支持多对“公钥/私钥”的原因是允许用户定期地改变他们的公钥，或者当用户认为某一个私钥已被泄漏时，他无须将当前已准备好或者正在传送过程中的邮件统统作废就可以改变公钥。每一对“公钥/私钥”都有一个标识符与其关联，所以邮件发送方可以告诉接收者自己是使用哪一个公钥来对它加密的。该标识符由公钥的低 64 位构成的。避免公钥标识符发生冲突是用户的责任。磁盘上的私钥必须使用特殊的口令（任意长）来加密，以避免它们被偷袭。

公钥环（public key ring）包含了与当前用户通信的其他用户的公钥。为了加密与每个邮件相关联的邮件密钥，这些公钥是必要的。公钥环上的每个条目不仅包含了公钥，而且

也包含它的64位标识符，以及一个代表用户信任此密钥的强烈程度的指示值。

这里需要解决的问题如下所述。假设公钥被存放在公告板上。Trudy为了阅读Bob的秘密电子邮件，一种办法是攻击公告板，并且用她自己选择的公钥来代替Bob的公钥。当Alice后来获得了这个自称属于Bob的公钥时，Trudy就可以对Bob实施水桶队列攻击。

为了防止这样的攻击，或者至少将攻击的后果减到最小，Alice需要知道在她的公钥环中，她对这个自称是“Bob的公钥”有多大的信任程度。如果她知道该公钥来自于Bob亲自交给她的软盘，那么她就可以将信任值设置到最大。正是这种非中心化的、由用户控制的公钥管理方法使得PGP无须采用中心化的PKI方案。

然而，有时候人们还是要通过查询一个可信的密钥服务器来获得其他用户的公钥。由于这个原因，在X.509被标准化以后，PGP除了支持传统的PGP公钥环机制以外，也支持X.509证书。所有当前的PGP版本都支持X.509。

8.8.2 S/MIME

IETF冒险进军电子邮件安全领域的结果是**安全的MIME**（S/MIME，Secure/MIME），该协议由RFC 2632～2643描述。它提供了认证、数据完整性、保密性和不可否认性。而且，它还非常灵活，支持大量的密码学算法。无须惊奇的是既然使用了这样的名字，那么它一定与MIME集成得很好，从而可以保护各种类型的邮件。此外，它也定义了许多新的MIME头，比如，用来存放数字签名的MIME头。

S/MIME并没有一个严格的、从单个根开始的证书层次结构，因为这种结构早已成为早期系统**增强保密邮件**（PEM，Privacy Enhanced Mail）的政治问题之一。相反，用户可以有多个信任锚。只要一个证书能够被回溯到当前用户所相信的某个信任锚，它就被认为是有效的。S/MIME使用了标准的算法和协议，关于这些算法和协议，我们前面都已经学习过了，所以这里不再进一步讨论它们。有关S/MIME的细节，请参考相应的RFC。

8.9 Web安全性

我们前面已经学习了两个非常需要安全性的重要领域：通信和电子邮件。你可以将它们想象成饭前汤和开胃小吃。现在该上主菜了：即Web安全。Web是如今的大多数Trudy们的主要工作场所，正是在Web上他们做着卑劣的事情。在本节，我们将讨论与Web安全有关的一些问题和事项。

我们可以粗略地将Web安全分成3个部分。第一，如何安全地命名对象和资源？第二，如何建立安全的、真实的连接？第三，当一个Web站点向客户发送一段可执行代码时，会发生什么？下面我们首先看一些潜在的威胁，之后我们再详细地讨论每一部分。

8.9.1 威胁

人们几乎每个星期都可以在报纸上看到有关Web站点安全的问题。这种状况确实非常严酷。我们现在看一些已经发生过的例子。首先，许多组织的主页受到过攻击，被代之以

骇客（cracker）选择的新主页（流行的出版物将这些侵入别人计算机的人称为**黑客**（hacker），但是许多程序员把“黑客”这个术语留给了那些超级程序员。本书中我们将这些人称为“骇客”）。曾经遭受过这种破坏的站点包括 Yahoo、美国军方、CIA、NASA 和纽约时报。在绝大多数情况下，骇客只是放置一些稀奇古怪或者调侃的文字，因而遭受破坏的站点可以在几小时内得到修复。

现在我们看一些更加严重的案例。有许多站点曾经因为遭受拒绝服务攻击而停止运行，在这些攻击中，骇客用大量的流量淹没掉 Web 站点，使它不能对合法的请求做出响应。通常这种攻击来自大量已被骇客攻破（DDoS 攻击）的机器。这样的攻击非常常见，所以它们甚至已经不再是新闻了，但是，它们却可以使被攻击的站点造成成千上万美元的损失。

1999 年，一名瑞典的骇客侵入了 Microsoft 的 Hotmail Web 站点，并创建了一个镜像站点，该镜像站点允许任何人输入 Hotmail 用户的名字，然后读取该用户当前的和已经存档的所有电子邮件。

在另一个案例中，一个名叫 Maxim 的 19 岁俄罗斯骇客侵入了一个电子商务 Web 站点，并且偷到了 300 000 个信用卡号码。然后，他联系上站点的所有者们，并告诉他们如果他们不付给他 100 000 美元，他将把所有的信用卡号码张贴到 Internet 上。他们并没有屈从于他的敲诈勒索，结果他真的张贴了信用卡号码，极大地损害了许多无辜者的利益。

在另一个不同类型的事件中，一名 23 岁的加利福尼亚学生以电子邮件的形式给一家通讯社发了一则新闻稿，谎称 Emulex 公司将要发布有关季度亏损的消息，而且 CEO 也要立即辞职。在几小时之内，该公司的股票暴跌了 60%，从而导致股票持有者损失了 20 亿美元以上。而作案者在发送公告之前不久卖出了他的股票，他赚了 25 万美元。虽然这起事件不属于 Web 站点入侵，但是很显然，这样的公告放在任何一家大公司的主页上，也会有类似的效果。

我们可以用许多页的篇幅来介绍类似这样的事件（这真的很不幸）。但现在该介绍与 Web 安全相关的技术要素了。有关各种安全问题的更多信息，请参考（Anderson，2008）、（Stuttard 和 Pinto，2007）以及（Schneier，2004）。你也可以在 Internet 上搜索到大量的实际案例。

8.9.2 安全命名

我们首先从一些最基本的问题入手：Alice 希望访问 Bob 的 Web 站点。她在她的浏览器中敲入 Bob 的 URL，几秒钟以后，一个 Web 页面出现了。但是，这真的是 Bob 的主页吗？可能是，也可能不是。Trudy 可能又重施故技了。例如，她可能截获了 Alice 发出来的所有数据包，并查看了这些数据包的内容。当她捕捉到有一个 HTTP GET 请求指向 Bob 的 Web 站点时，她可以到 Bob 的 Web 站点上将页面取过来，然后根据她的意愿进行修改，再将修改之后的假页面返回给 Alice。Alice 对此一无所知。更糟的是，Trudy 可以大幅降低 Bob 电子商场中的货物价格，以便使他的货物看起来非常有诱惑力，从而诱使 Alice 将她的信用卡号码发送给 Bob 以购买商品。

这种经典的中间人攻击的一个缺点是，Trudy 必须位于某一个能够截取到 Alice 的出境

流量以及能够伪造她入境流量的位置上。实际上，她必须搭接在 Alice 的电话线上，或者 Bob 的电话线上，因为搭接到光纤骨干线路上相当困难。虽然主动搭线这种方法肯定行得通，但是它需要一定的工作量；另外，虽然 Trudy 非常聪明，但同时她也非常懒惰。除了这种做法以外，还有其他更加容易的方法在等着 Alice 去尝试。

DNS 欺骗

Trudy 能够攻破 DNS 系统的一种方式或许是 Alice 的 ISP 上的 DNS 缓存，Trudy 用她自己的 IP 地址（比如说 42.9.9.9）来替换 Bob 的 IP 地址（比如说 36.1.2.3）。这将会导致下述的攻击发生。图 8-46(a)显示了设想中的工作过程。这里，（1）Alice 向 DNS 请求 Bob 的 IP 地址；（2）得到了地址；（3）向 Bob 请求他的主页；（4）得到了 Bob 的主页。当 Trudy 将 Bob 的 DNS 记录修改成包含她自己的 IP 地址而不再是 Bob 的地址时，我们得到了如图 8-46(b)所示的情形。这里，当 Alice 查询 Bob 的 IP 地址时，她得到了 Trudy 的 IP 地址，所以她期望发送给 Bob 的流量全部转向 Trudy。现在 Trudy 就可以实施中间人攻击，而无须陷入到搭接任何电话线的麻烦之中。不过，她必须要侵入一台 DNS 服务器，并改变其中一条记录，这可能会容易得多。

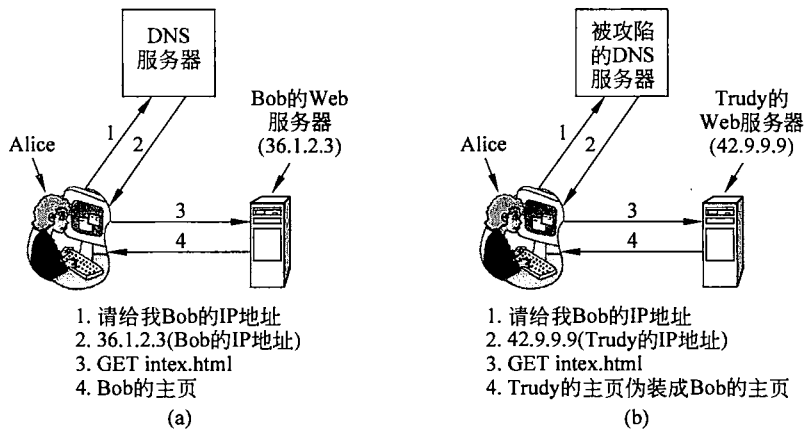


图 8-46

(a) 正常情况下；(b) 一个基于攻入 DNS 服务器和修改 Bob 记录的攻击

Trudy 如何欺骗 DNS 呢？这实际上比较简单。简而言之，Trudy 可以诱使 Alice 的 ISP 的 DNS 服务器发出一个查询请求，请求查找 Bob 的 IP 地址。由于 DNS 使用的是 UDP，所以对于 DNS 服务器来说，非常不幸的是，它实际上并没有办法来检查到底谁提供了答案。Trudy 可以充分利用这种特性，她的做法是，伪造一个预设的应答数据包，从而将一个假的 IP 地址插入到 DNS 服务器的缓存中。为了简化起见，我们假设 Alice 的 ISP 初始时并不包含与 Bob 的 Web 站点 bob.com 相对应的 DNS 表项。如果已经存在的话，则 Trudy 可以等待一段时间，直到该表项超时然后再重试（或者使用其他的技巧）。

Trudy 发起攻击的做法：她向 Alice 的 ISP 发送一个查询请求，请求 bob.com 的 IP 地址。由于不存在与这个 DNS 名字相对应的表项，所以缓存服务器询问顶级 com 域服务器，以便得到此 DNS 名字的 IP 地址。然而，Trudy 抢在 com 服务器之前先发送回一个假的应答："bob.com 是 42.9.9.9"，这里的 IP 地址正是她自己的地址。如果她的假应答首先到达

Alice 的 ISP 处，则她的 IP 地址将被缓存起来，而真正的应答则变成一个不请自来的应答而遭到拒绝。诱使 DNS 服务器安装一个假 IP 地址的技术称为 **DNS 欺骗**（DNS spoofing）。像这样存放了一个刻意的假 IP 地址的缓存则称为**染毒缓存**（poisoned cache）。

实际上，事情并没有这么简单。首先，Alice 的 ISP 要检查该应答是否包含顶级服务器的正确 IP 源地址。但是，由于 Trudy 可以在这个 IP 域中设置任何数值，所以她很容易挫败这项测试，因为顶级服务器的 IP 地址是公开的。

其次，为了允许 DNS 服务器辨别哪个应答对应于哪个请求，所有的请求都携带一个序号。为了欺骗 Alice 的 ISP，Trudy 必须知道它的正确序号。要想了解当前的序号，对于 Trudy 来说，最简单的办法是自己注册一个域，比如说 trudy-the-intruder.com。我们假定它的 IP 地址也是 42.9.9.9。她为这个新谋划的域创建一个 DNS 服务器：dns.trudy-the-intruder.com。而且，该服务器仍然使用 Trudy 的 IP 地址 42.9.9.9，因为 Trudy 只有一台计算机。现在，她必须让 Alice 的 ISP 知道她的 DNS 服务器。这很容易做到，她只需向 Alice 的 ISP 请求 foobar.trudy-the-intruder.com 即可，这会导致 Alice 的 ISP 询问顶级 com 服务器，谁在为 Trudy 的新域提供 DNS 服务。

随着 dns.trudy-the-intruder.com 安全地进入到 Alice 的 ISP 的缓存中，真正的攻击开始了。现在 Trudy 向 Alice 的 ISP 请求查询 www.trudy-the-intruder.com。该 ISP 很自然地向 Trudy 的 DNS 服务器发送一个同样的查询请求。该请求携带的序号正是 Trudy 所努力寻找的。Trudy 高兴得跳了起来，她立即请求 Alice 的 ISP 查询 Bob 的地址，并迅速地发送一个伪造的应答来回答自己的问题，而且谎称该应答来自顶级 com 服务器，该应答的内容是："bob.com 是 42.9.9.9"。这个伪造的应答中所携带的序号，比她刚刚接收到的请求中的序号大 1。虽然她已经算得很准确了，但是她也可以再发送第二个伪造的应答，其中序号比刚才接收到的大 2，甚至可能发送几十个伪造的应答，其中的序号逐渐增加。这些应答中总有一个会匹配到 Alice 的 ISP 所发送的 DNS 查询请求，剩下的将被丢弃。当这个伪造的应答到达时，它被缓存起来；而当真正的应答稍后到达时，却遭到拒绝，因为那时已经没有正在等待回应的查询了。

现在，当 Alice 查询 bob.com 时，他被告知应该使用 42.9.9.9，即 Trudy 的地址。Trudy 在她自己舒适的客厅里成功地实施了中间人攻击。图 8-47 显示了这种攻击的各个步骤。这种特定的攻击可以通过让 DNS 服务器在发出的查询中使用随机 ID 而不是仅仅计数值来挫败，但是看起来每当补上一个漏洞，另一个又出现。特别是，ID 仅是 16 位，所以用计算机很容易尝试所有的可能数字。

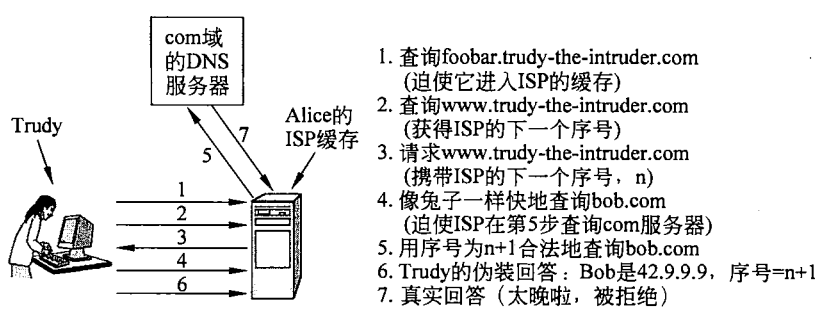


图 8-47 Trudy 如何欺骗 Alice 的 ISP

安全的 DNS

前面介绍的攻击很容易被挫败，只要让 DNS 服务器在它们发出的查询请求中使用随机的 ID 而不是按顺序递增的 ID。但事情总是这样，每当补上一个漏洞时，另一个漏洞又冒了出来。真正的问题在于，在 DNS 被设计时，Internet 只是一个仅供几百所大学使用的研究工具，而不是让 Alice、Bob 或者 Trudy 都加入进来的聚会场所。那时候安全并不是一个问题，使 Internet 能真正工作起来才最为重要。几十年来，网络环境已经有了很大的变化，所以，1994 年 IETF 成立了一个工作组致力于使 DNS 从本质上变得更加安全。这个（正在进行中）项目称为 DNS **安全**（DNSsec，DNS Security）；它的第一项成果发表在 RFC 2535。不幸的是，DNSsec 尚未得到完全的部署，所以，目前有大量的 DNS 服务器仍然会遭受各种欺骗攻击。

从概念来看，DNSsec 极其简单。它建立在公开密钥密码学的基础之上。每一个 DNS 区域（按照图 7-5 中的含义）有一个“公钥/私钥”对。DNS 服务器发送的所有信息都经过签名，签名所用的私钥是发起区域的私钥，所以接收方可以验证它的真实性。

DNSsec 提供了 3 个基本服务：

（1）证明数据是从哪里发出来的。

（2）公钥的分发。

（3）事务和请求的认证。

主要的服务是第一个，它证实了返回的数据确实经过了区域所有者的同意。第二个服务对于安全地保存和获取公钥非常有用。第三个服务对于预防重放和欺骗攻击非常必要。请注意，DNSsec 并没有提供保密性服务，因为 DNS 中的所有信息都应该是公开的。按照预定的期望，DNSsec 需要几年时间才能被完全部署，所以，具有安全功能的 DNS 服务器与不支持安全功能的服务器之间的协同工作能力是最基本的，这意味着 DNSsec 不能改变协议。现在我们来考察其中的一些细节。

DNS 的记录被组织成一些称为**资源记录集**（RRSet，Resource Record Sets）的集合，所有具有同样名字、类别和类型的记录被集中到一个集合中。一个 RRSet 可能包含多个 A 记录，例如，倘若一个 DNS 名字被解析到一个主要的 IP 地址和一个次要的 IP 地址。RRSet 也允许扩展新的记录类型（后面将讨论到）。每个 RRSet 都有一个密码学意义上的散列值（比如使用 MD5 或者 SHA-1），然后又利用该区域的私钥对散列值进行签名（比如使用 RSA 算法）。发送给客户的传输单元是经过签名的 RRSet。当客户接收到一个签过名的 RRSet 时，它可以验证此 RRSet 是否被发起区域签过名。如果签名没有问题，则可以接受 RRSet 中的数据。由于每个 RRSet 都包含它自己的签名，所以 RRSet 可以被缓存在任何地方，即使被缓存在不可信的服务器上也不会危及安全性。

DNSsec 引入了几种新的记录类型。第一种是 KEY 记录。这种记录包含了某一个区域、用户、主机或者其他主要的公钥，以及用于签名的密码算法、所使用的传输协议，还有其他一些位。这里的公钥是裸存的，DNSsec 没有使用 X.509 证书，因为它太庞大。若算法字段被设置为 1，则代表了签名方案使用 MD5/RSA 算法组合（首选的方案），其他的算法组合使用其他的值。协议字段表明了使用 IPSec 或者其他的安全协议（如果有的话）。

第二种新的记录类型是 SIG 记录。它存放的是经过签名的散列值，其中所用的算法由

KEY 记录来指定。这个签名作用在该 RRSet 中的所有记录上（即签名的范围），包括所有出现的 KEY 记录，但是不包括 SIG 记录自身。SIG 记录也包含了该签名有效周期的开始时间和失效时间、签名者的名字以及其他一些信息。

在 DNSsec 的设计方案中，每个区域的私钥可以一直保持离线状态。每天一次或者两次将一个区域的数据库通过手工方式（比如通过 CD-ROM）传输到一台未联网的机器上，这台机器上保存着私钥。所有的 RRSet 在那里被签名，由此而产生的 SIG 记录被传回到该区域的主服务器中。通过这种方式，私钥可以被保存在 CD-ROM 上，除了每天签名新的 RRSet 时需要将 CD-ROM 插入到未联网的机器中以外，其他时间该 CD-ROM 被锁在保险箱里。当签名完成以后，内存中和磁盘上所有的私钥副本都被擦除掉，并且 CD-ROM 又被放回保险箱里。这个过程实际上把电子安全还原为物理安全，人们对于物理安全总是比较容易理解和处理。

这种预先为 RRSet 做签名的方法大大加速了 DNS 服务器响应客户查询的过程，因为这样就不需要在处理过程中做密码学运算了。所付出的代价是，在 DNS 数据库中需要大量的磁盘空间来存储所有的密钥和签名。由于引入了签名，有些记录的长度将增加十倍以上。

当客户进程获得了一个签过名的 RRSet 时，它必须利用发起区域的公钥来解密散列值，同时自己计算一遍散列值，然后比较这两个值。如果它们相等，则认为此 RRSet 有效。然而，这个过程又引出了另一个问题，即客户如何得到一个区域的公钥。一种方法是利用一个安全的连接（比如使用 IPSec）通过一台可信的服务器来获得区域的公钥。

然而，实际上，DNSsec 假定客户已经预先配置了所有顶级域的公钥。如果 Alice 现在想要访问 Bob 的 Web 站点，则她可以向 DNS 请求 bob.com 的 RRSet，该 RRSet 中包含了 Bob 的 IP 地址和一个 KEY 记录（其中包含了 Bob 的公钥）。此 RRSet 已被 com 顶级域签过名，所以 Alice 很容易验证它的有效性。图 8-48 中显示了这个 RRSet 可能包含的内容。

域名	生存期	类别	类型	值
bob.com.	86 400	IN	A	36.1.2.3
bob.com.	86 400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86 400	IN	SIG	86947503A8B848F5272E53930C...

图 8-48 bob.com 的 RRSet 例子
KEY 记录是 Bob 的公钥。SIG 记录是顶级 com 服务器签过名的 A 和 KEY 记录的散列值，以便验证它们的真实性

现在 Alice 拥有了 Bob 公钥的一份副本，并且已经通过了验证，于是她向 Bob 的 DNS 服务器（由 Bob 运行）请求 www.bob.com 的 IP 地址。Bob 返回的 RRSet 将用 Bob 的私钥做签名，所以 Alice 可以验证这个 RRSet 的签名。如果 Trudy 设法将一个假的 RRSet 插入到任何一个缓存中，则 Alice 很容易就可以检测出它是不真实的，因为 RRSet 中的 SIG 记录将是不正确的。

然而，DNSsec 还提供了一种密码学机制把应答和特定的查询绑定起来，从而防止像图 8-47 中 Trudy 所做的那种欺骗。这种（可选的）反欺骗手段的做法是，应答方利用自己的私钥对查询消息的散列值做签名，然后把签名后的散列值也加入到应答消息中。由于 Trudy 不知道顶级 com 服务器的私钥，所以她不可能为 Alice 的 ISP 所发送的查询伪造一个应答。她当然可以让她的应答先返回到 Alice 的 ISP 处，但是它将遭到拒绝，因为她对查

询的散列值所做的签名是无效的。

DNSsec 也支持其他一些记录类型。例如，CERT 记录可以被用来保存证书（比如 X.509 证书）。之所以提供这个记录是因为有些人希望将 DNS 变成一个 PKI。这种变化是否真的会发生还有待于进一步观察。我们对于 DNSsec 的讨论到此为止，有关更多的细节，请参考 RFC 2535。

8.9.3 SSL——安全套接层

安全的命名机制是一个很好的起点，但是，对于 Web 安全而言还需要做得更多。下一步是安全的连接。我们现在考察如何实现安全的连接。安全涉及的事情绝不简单，这不是非此即彼。

当 Web 最初出现在公众面前时，它仅仅被用来发布静态页面而已。然而，不久之后，有些公司就想到了将 Web 用于金融交易，比如用信用卡购物、在线银行和电子股票交易。这些应用创造了一种新的需求，即迫切需要安全的连接。1995 年，Netscape 公司作为当时占主导地位的浏览器厂商，对此迅速做出了响应，他们引入了一个称为**安全套接字层**（SSL，Secure Sockets Layer）的安全软件包来迎合这种需求。这个软件包和它的协议现在已被广泛使用，甚至也为 Firefox、Safari 和 Internet Explorer 所采用，因此值得我们考察它的某些细节。

SSL 在两个套接字之间建立一个安全的连接，其中包括以下功能：

（1）客户与服务器之间的参数协商。

（2）客户和服务器的双向认证。

（3）保密的通信。

（4）数据完整性保护。

以前我们曾经看到过这些功能的确切含义，所以这里不再细致地加以阐述。

图 8-49 显示了 SSL 在通常协议栈中的位置。实际上，它位于应用层和传输层之间的一个新层，它接受来自浏览器的请求，再将请求传递给 TCP 以便传输到服务器上。一旦安全的连接已经被建立起来，则 SSL 的主要任务是处理压缩和加密。在 SSL 之上使用的 HTTP 尽管也是标准的 HTTP 协议，但是它称为**安全的** HTTP（HTTPS，Secure HTTP）。有时候它使用一个新的端口（443），而不是标准的端口（80）。顺便提一下，尽管 SSL 并不仅限于被用在 Web 浏览器中，但这仍然是它最常见的应用。它也可以提供双向认证。

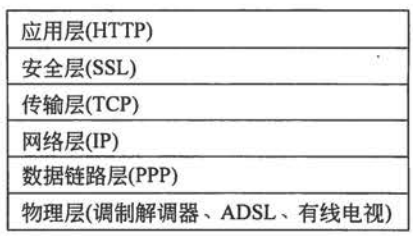


图 8-49 家庭用户使用 SSL 层（和协议） 浏览网页

SSL 协议经历了几个版本。下面我们只讨论第 3 版，这是最为广泛使用的版本。SSL 支持许多种不同的算法和选项。这些选项包括是否使用压缩功能、使用哪些密码学算法，以及一些与密码产品出口限制有关的事项。最后一项的意图是为了确保高强度的密码学技

术仅用于连接双方都在美国国内的情形。在其他情况下，密钥被限制在40位，许多密码学家对此非常不屑一顾。为了从美国政府这里拿到出口许可，Netscape被迫加入了这项限制。

SSL由两个子协议组成，一个用来建立安全的连接，另一个使用安全的连接。我们首先来看如何建立安全连接。图8-50显示了连接建立子协议。子协议从消息1开始，此时Alice向Bob发送一个建立连接的请求。该请求指定了Alice所用的SSL版本，以及她优先选择的压缩算法和密码学算法。它也包含一个临时值R_A，后面将会用到此值。

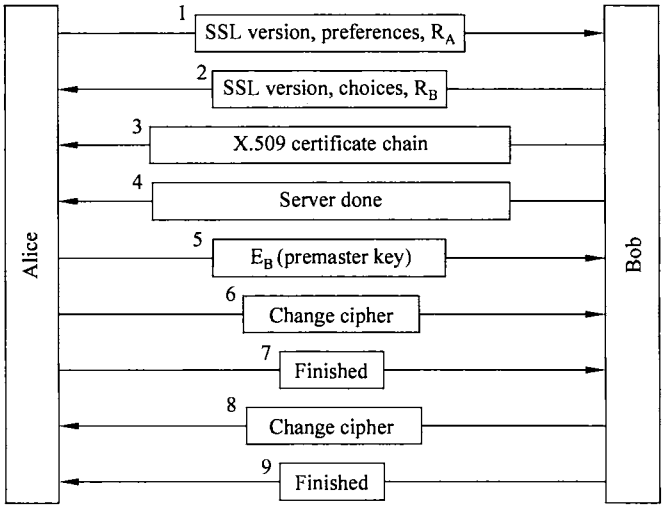


图8-50 简化版的一个SSL连接建立子协议

现在轮到Bob了。在消息2中，Bob在Alice可能支持的各种算法之中做出选择，并且将他的临时值R_B发送给Alice。然后，在消息3中，他发送一个证书，其中包含了他的公钥。如果这个证书并没有被某一个知名的权威机构签过名，那么，他还发送一个证书链，通过这条证书链可以回溯到某一个由权威机构签名的证书。所有的浏览器，包括Alice的浏览器，都预装了大约100多个公钥，所以，如果Bob能够建立起一条证书链，并且此证书链以这100多个证书中的某一个作为信任锚，那么，Alice将能够验证Bob的公钥。这时Bob可以发送其他一些消息（比如请求Alice的公钥证书）。当Bob完成以后，他发送消息4告诉Alice该轮到她了。

Alice的响应是，选择一个随机的384位**预设主密钥**（premaster key），并且用Bob的公钥加密之后发送给Bob（消息5）。用于加密数据的实际会话密钥是从这个预设主密钥和两个临时值，通过一种复杂的方法推导得来的。当Bob接收到消息5以后，Alice和Bob都能够计算会话密钥。由于这个原因，Alice告诉Bob，现在请切换到新的密码（消息6），并且告诉他她已经完成了建立连接子协议（消息7）。然后，Bob对这两条消息分别进行确认（消息8和9）。

然而，虽然Alice知道了Bob是谁，但是，Bob并不知道Alice是谁（除非Alice有一个公钥和相应的证书，而这对于普通个人用户来说是不太可能的）。因此，Bob的第一条消息极有可能是一个登录请求，请求Alice用一个预先建立的名字和口令登录进来。然而，此登录协议完全超出了SSL的范围。一旦SSL的安全连接已经建立起来，不管怎么样，数据传输就可以开始了。

正如上面所提到的，SSL 支持多种密码学算法。最强的一种方案使用三个独立密钥的三重 DES 来加密数据，通过 SHA-1 保护数据完整性。这种做法相对比较慢，所以它主要用于银行和其他一些对安全性要求比较高的应用。对于普通的电子商务应用，则使用 128 位密钥的 RC4 算法完成数据加密，通过 MD5 实现消息认证。RC4 将这个 128 位的密钥用作一个种子，并且将它扩展成一个非常大的数以便内部使用。然后，它利用这个内部数来生成一个密钥流。该密钥流与明文做异或（XOR）操作，从而提供了一个经典的流密码算法，正如我们在图 8-14 中看到的那样。出口版本也使用了 128 位密钥的 RC4 算法，但是其中 88 位是公开的，以使该密码算法易于被破解。

为了实际传输数据，我们需要使用第二个子协议，如图 8-51 所示。来自浏览器的消息首先被分割成最多 16 KB 的单元。如果当前连接支持压缩功能的话，则每个单元被单独压缩。之后，根据两个临时值和预设主密钥推导出来的一个密钥被串接到压缩之后的文本中，然后再利用已经协商好的散列算法（通常是 MD5）对串接之后的结果做散列运算。这个散列值被附加在每一个分段的尾部，作为它的消息认证码（MAC）。然后，再使用协商好的对称加密算法（通常是与 RC4 的密钥流进行 XOR 运算）对压缩之后的分段和 MAC 进行加密。最后，每个分段被附上一个分段头，再通过 TCP 连接被传送出去。

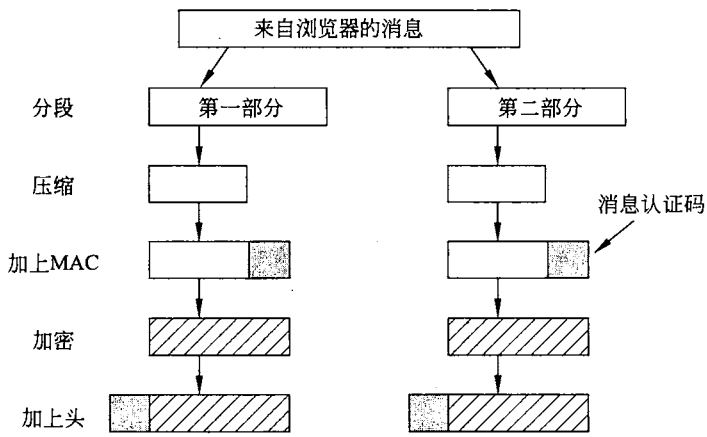


图 8-51 使用 SSL 的数据传输

然而，在这里有个忠告值得一提。由于研究人员已经证明了 RC4 算法存在一些弱密钥，即很容易被分析出密钥，所以使用 RC4 的 SSL 其安全性将是不可靠的（Fluhrer 等，2001）。如果浏览器允许用户选择密码算法组合，则用户应该将它配置成使用 168 位密钥的三重 *DES* 和 SHA-1，尽管这种组合比 RC4 和 MD5 要慢一些。

SSL 存在一个问题，即主角可能没有证书，或者即使他们有证书，他们也并不总是验证所用的密钥是否与证书相匹配。

1996 年，Netscape 公司将 SSL 移交给 IETF 进行标准化。最终的结果是**传输层安全**（TLS，Transport Layer Security）。RFC 5246 描述了 TLS。

TLS 基于 SSL 的第 3 个版本。尽管 IETF 对 SSL 所做的修改非常小，但即使这些小小的变化也足以使 SSL 版本 3 和 TLS 无法实现互操作。例如，从预设主密钥和临时值推导出会话密钥的方法被改变了，从而使得密钥更强大（即难以进行密码分析）。由于这种不兼容性，大多数浏览器都实现了两个协议，需要时可以协商将 TLS 回退到 SSL，这就是所谓的

SSL/TLS。第一个 TLS 实现出现在 1999 年，其版本 1.2 在 2008 年 8 月得到定义，它包括更强大的密码套件（特别是 AES）。SSL 在市场上一直保持着强势，虽然 TLS 有可能会逐渐取代它。

8.9.4 移动代码安全性

命名和连接是与 Web 安全有关的两个主要关注领域。但是，除此以外还有其他一些领域也值得关注。在早期，Web 页面只是一些静态的 HTML 文件，它们不包含任何可执行代码。现在，Web 页面通常包含小程序，例如 Java Applet、ActiveX 控件和 JavaScript 等。下载并执行这样的**移动代码**（mobile code）很显然存在极大的安全风险，所以人们已经设计了各种方法来尽可能地降低这种风险。现在我们来快速地了解某些因为移动代码而引起的问题，以及处理这些问题的一些方法。

Java 小程序安全性

Java 小程序（Java Applet）是小型的 Java 程序，它们已被编译成一种面向栈的机器语言，即 **Java 虚拟机**（JVM，Java Virtual Machine）。Java 小程序可以被放到 Web 页面，连同 Web 页面一起被下载到用户机器上。当 Web 页面被加载到浏览器中以后，这些 Java 小程序也被插入到浏览器内部的 JVM 解释器中，如图 8-52 所示。

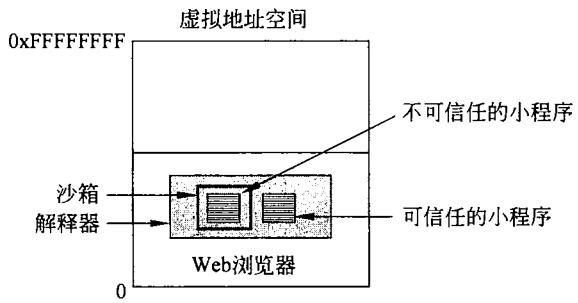


图 8-52 可被 Web 浏览器解释的 Java 小程序

在经过编译的代码上运行解释性代码的好处是，解释器在执行每一条指令之前可以先对指令进行检查。这使解释器有机会检查每条指令的地址是否有效。另外，系统调用也可以被捕获到，并且得到解释。如何处理这些调用是一件与安全策略有关的事情。例如，如果一个 Java 小程序是可信任的（比如它来自本地硬盘），则解释器可以毫不犹豫地执行它的系统调用。然而，如果一个 Java 小程序是不可信任的（比如它通过 Internet 进入本地机器），则解释器可能要将它封装到一个**沙箱**（sandbox）中以便限制它的行为，并且捕捉住它使用系统资源的企图。

当一个 Java 小程序试图要使用某一个系统资源时，它的调用被传递给一个安全监视器进行审核。监视器根据本地的安全策略对该调用进行检查，然后做出允许或者拒绝的决定。通过这种方式，就有可能让小程序访问某些资源，但不是所有的资源。不幸的是，在现实中这种安全模型工作得非常差，它总是不停地出现各种错误。

ActiveX

ActiveX 控件是可以被嵌入到 Web 页面中的 x86 二进制程序。当浏览器遇到一个

ActiveX 控件时，它就要对该控件进行检查，看它是否应该被执行，如果通过了测试，则执行该控件。ActiveX 控件并不是解释执行的，也没有以任何方式被封装在沙箱中，所以，它具有跟其他用户程序一样的能力，而且有可能造成更大的危害。因此，所有的安全性仅在于一项简单的决定上，即决定是否运行 ActiveX 控件。现在回想起来，整个构思是个巨大的安全漏洞。

Microsoft 做出这项决策的方法建立在**代码签名**（code signing）的基础上。每个 ActiveX 控件都伴随一个数字签名——由它的创建者利用公开密钥密码技术对代码的散列值进行签名。当浏览器看到一个 ActiveX 控件时，它首先验证该控件的签名，以确定它是否在传输途中被篡改了。如果签名是正确的，于是浏览器检查它的内部表，看该程序的创建者是否可信，或者是否存在一条信任链可以回溯到某一个可信的创建者。如果该创建者是可信的，则程序被执行；否则它不予执行。Microsoft 用来验证 ActiveX 控件的系统称为**认证码**（Authenticode）。

将 Jave 和 ActiveX 这两种方法做一下对比非常有用。若采用 Java 方法，则用户无法确定一个 Java 小程序是谁写的；但是，运行时的解释器可以保证这个 Java 小程序不会做出该机器主人明确指明不允许做的事情。与此相反，若采用代码签名的方法，则浏览器不可能监视移动代码运行时的行为。如果移动代码来自于一个可信源，并且在传输途中没有被修改过，那么它直接运行即可，这时不必查看这份代码是否是恶意的。如果原始程序员有意编写这段代码来格式化硬盘，并且擦除闪存（flash ROM）中的内容，从而使计算机无法再重新启动，而且如果程序员已经被证明是可信的，那么这段代码将被运行，从而毁坏计算机（除非浏览器已经禁用 ActiveX 控件）。

许多人觉得，信任一个不熟悉的软件公司是一件可怕的事情。为了演示这个问题，西雅图的一名程序员组建了一个软件公司，并且使它通过鉴定成为一个可信任的公司，这是不难做到的。然后，他编写了一个 ActiveX 控件，该控件所做的事情是彻底关闭机器，然后他广泛发布此控件。它关闭了许多机器，但是这些机器只需重新启动即可，所以没有造成其他任何危害。这名程序员只是想把这个问题暴露给全世界。官方的回应是撤销了这个 ActiveX 控件所使用的证书，从而结束了这一短暂的窘迫事件；但是，根本的问题仍然存在，并且有可能被恶意的程序员所利用（Garfinkel 和 Spafford，2002）。成千上万的软件公司都有可能编写移动代码，要想监管所有这些公司是不可能的，所以，代码签名技术实际上是一场等待发生的灾难。

JavaScript

JavaScript 没有任何正式的安全模型，但是却有很长的安全泄漏历史。每个厂商处理安全问题的方式都不相同。例如，Netscape Navigator 第 2 版使用了类似于 Java 模型的方式，但是到第 4 版时，它又放弃了原来的模型，改而使用代码签名模型。

基本的问题在于让外部代码在你的机器上运行，这等于在自找麻烦。从安全的角度来看，这就类似于邀请一个窃贼到你家里，然后企图谨慎地看守住他，以便不让他从厨房溜到客厅里。如果一旦发生意料之外的事情，你稍不留神坏事就有可能发生。这里的安全压力在于移动代码允许各种闪烁的图形和快速的交互过程，而且许多 Web 设计者认为，这些比安全性更加重要，尤其是当处于危险境地的是别人的机器而不是自家机器的时候。

浏览器扩展

除了用代码扩展 Web 页面外，在浏览器扩展、外接附件和插件方面还有一个蓬勃发展的市场。它们是用来扩展 Web 浏览器的计算机程序。插件通常提供了解释或者显示一定类型内容的能力，例如 PDF 或者 Flash 动画。扩展和外接附件提供新的浏览器功能，例如更好的口令管理，或者与页面更好的交互方式，使网上购某些方面的物品变得更加容易。

安装一个浏览器扩展、外接附件或插件，就像遇到你在浏览时和跟踪链接来安装程序一样容易。这个动作将使代码穿过 Internet 被下载到本地和安装在浏览器上。所有的这些程序都写在一个框架上，这个框架不尽相同，具体取决于正被增强的浏览器。然而，近似地说它们已经变成浏览器可信计算基础的一部分。也就是说，如果安装的代码有问题，整个浏览器都将受到损害。

有其他两个很明显的故障模式。首先是程序可能的恶意行为，如收集个人信息然后发给其他远程服务器。对于所有的浏览器而言，它们为用户安装扩展程序正是为了这个目的。第二个问题是插件让浏览器有能力去解释新类型的内容。通常这内容本身是一个完全成熟的编程语言。PDF 和 Flash 就是很好的例子。当用户看那些具有 PDF 和 Flash 内容的网页时，浏览器中的插件就执行 PDF 和 Flash 代码。对于所有这些原因，外插附件和插件应该只在需要时才安装，并且只安装那些来自值得信任的供应商的附件和插件。

病毒

病毒是另一种形式的移动代码。与上述移动代码例子唯一不同的是病毒并不是被邀请进来的。病毒与普通移动代码之间的区别在于病毒具有繁殖能力，它可以不断地复制自己。当一个病毒不管是通过 Web 页面、电子邮件的附件或者其他某种方式入境时，它通常首先会感染硬盘上的可执行程序。当任何一个被感染的程序运行起来时，控制权被传递给了病毒，它通常试图将自己传播到其他的机器，例如，通过电子邮件将自身的副本寄送给受害者的电子邮件地址簿中的每一个人。有些病毒还会感染硬盘的启动扇区，当机器启动时，病毒就有机会运行。病毒已经变成了 Internet 上的一个大问题，而且导致了数十亿美元的损失。关于病毒并没有显而易见的解决方案。可能一个全新的操作系统将有助于抑制病毒，这个新的操作系统将建立在安全的微内核基础之上，并且严格地区分用户、进程和资源。

8.10 社会问题

Internet 和它的安全技术是一个交汇着社会问题、公共政策和技术交汇的广阔领域，这种交汇往往带来重要的后果。下面我们将简短地讨论 3 个领域：隐私、言论自由和版权。毋庸多说，我们这里只能涉猎表面而已。有关另外的读物，请参看（Anderson，2001）、（Garfinkel 和 Spafford，2002）以及（Schneier，2000）。Internet 本身也充满了各种材料。你只需在任何一个搜索引擎中输入诸如“privacy”（隐私）、“censorship”（审查）和“copyright”（版权）之类的关键字即可。另外，在本书的 Web 站点上你也可以看到一些链接，网站 URL 为 http://www.pearsonhighered.com/tanenbaum。

8.10.1 隐私

人们有隐私权吗？这是个很好的问题。美国宪法的第4次修订版本中禁止政府在毫无恰当理由的情况下搜查人们的房屋、文件和私人财产，并且限制了搜查许可证的发放条件。因此，隐私被提到公开的议程中已经有200多年的历史了，至少在美国是如此。

在过去10多年中所发生的变化有两个方面：政府监视老百姓更加容易了，而老百姓预防这种监视也更加容易了。在18世纪，政府为了搜查一个公民的文件，它必须派出一名骑警到该公民的农场去查看某些文档。这是一个非常烦琐的过程。现在，只要出示搜查许可证，电话公司和Internet供应商就会提供窃听装置。这使得警察们的工作更加容易了，而且也不用再冒着从马上掉下来的危险了。

密码学技术的发展改变了这一切。任何人只要下载并安装了PGP，而且使用星际强度的密钥，他就可以保证宇宙间的任何一个人都不可能读取他的电子邮件，不管他有没有搜查许可证。政府很清楚这一点，并且不喜欢这样的局面。对于政府来说，真正的隐私意味着它们将很难监视各种各样的罪犯，而且也很难监视新闻记者和各种持不同政见者。因此，有些政府限制或者禁止密码技术的使用或出口。例如，在法国，1999年以前所有的密码系统都是禁止的，除非把密钥交给政府。

法国并不是唯一一个禁止密码技术的国家。1993年4月，美国政府宣布它计划要制造一种硬件密码处理器，称为**加密芯片**（clipper chip），将作为所有网络通信的标准。据说用这种方式就可以保证公民的隐私。该计划同时也提到了这种芯片采用一种称为**密钥托管**（key escrow）的方案（它允许政府访问所有的密钥），从而为政府提供了解密所有流量的能力。然而，该计划也承诺，只有当拥有合法的搜查许可证时政府才可以使用这种能力。无须多说，随之爆发了激烈的抗议，隐私倡导者们公开指责整个计划，而法律执行官则极力赞美它。最终，政府取消了这个计划，并放弃了这种想法。

在电子前沿基金会（Electronic Frontier Foundation）的Web站点www.eff.org上，有大量关于电子隐私的信息可以参考。

匿名邮件转发器

PGP、SSL和其他的一些技术使得任何两方之间有可能建立起安全的、真实的通信，从而避免受到第三方的监视和干扰。然而，有时候，隐私的真正含义是不要认证，也就是说使通信变成匿名。对于点到点的消息、新闻组或者两者兼而有之的应用场合，人们可能非常期望这种匿名性。

现在我们来看一些例子。第一，生存在独裁政治体制下的持不同政见者通常希望进行匿名的通信，以此来防止被监禁或谋杀。第二，企业、教育、政府以及其他机构中的不正当行为通常被一些勇敢的正义人士揭发出来，而这些揭发者通常希望能保持匿名，以免遭到打击报复。第三，持有非主流的社会、政治或宗教观点的人可能希望在跟其他人通过电子邮件或者新闻组进行通信时不暴露自己的身份。第四，人们可能希望在新闻组中匿名地讨论酗酒、精神病、性骚扰、虐待儿童，或者受迫害的少数派的成员信息。当然，除此以外，还有许多其他例子。

我们来考虑一个特殊的例子。在20世纪90年代，一个非传统宗教组织的某些批评人士利用一个**匿名邮件转发器**（anonymous remailer），将他们的观点张贴到一个USENET新闻组中。这个服务器允许用户创建假的名字，并且给服务器发送邮件；然后，该服务器使用这个假名字将邮件寄送或者张贴出去，所以没有人能够知道消息的确切来源。他们的有些贴子泄露了宗教组织所宣称的内部机密以及受版权保护的文档。该宗教组织对此做了回应，它告诉本地权力机构，它的内部机密被暴露，并且它的版权受到了侵犯，这两者都是转发服务器所在地的犯罪行为。这起事件被闹到法庭上，最终，服务器运行商被迫将那些隐含了发贴人真实身份的历史记录信息交出来（顺便提一下，这并不是第一起因为有人泄露秘密而惹怒一个宗教的事件：1536年William Tyndale因为将圣经翻译成英文而被绑在树桩上活活烧死）。

Internet相当一部分地区被这种侵犯隐私的行为彻底激怒了。每个人得出的结论是那些保存了从真实邮件地址到假名字之间映射关系的匿名邮件中继器（现在称为第一类邮件转发器）不再有任何价值。这种情形刺激了很多人开始设计可抵抗法庭传票攻击的匿名邮件转发器。

这些新的邮件转发器通常称为**加密朋克邮件转发器**（cypherpunk remailer），其工作过程如下所述。用户产生一个电子邮件消息，其中含有完整的RFC 822头（当然除了“From:”以外），然后用邮件转发器的公钥对它加密，并发送给该邮件转发器。在邮件转发器上，外部的RFC 822头被剥掉，内容被解密，然后邮件被重新发送出去。邮件转发器没有任何账号，也不维护日志，所以，即使邮件转发服务器事后被查抄，它也不会保留任何从它这里经过的邮件的痕迹。

许多希望发送匿名消息的用户可以使他们的请求穿越多个匿名邮件转发器，如图8-53所示。这里，Alice想要给Bob发送一张完完全全匿名的情人节卡片，所以她使用3个邮件转发器。她写好一个邮件M，把邮件放在一个含有Bob邮件地址的RFC 822头中。然后，她用3号邮件转发器的公钥E_3对整个邮件进行加密（如图中水平阴影线所示）。此时她又在整个邮件的外面套上一个明文头，其中包含3号邮件服务器的邮件地址。这是图中2号和3号邮件转发器之间所显示的邮件。

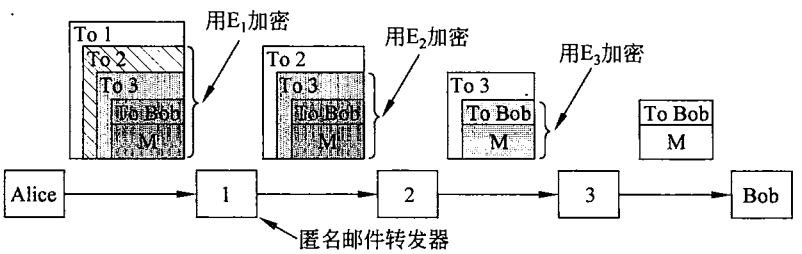


图8-53 Alice如何使用3个邮件转发器给Bob发送邮件

然后，她用2号邮件转发器的公钥E_2来加密这个邮件（如图中垂直阴影线所示），并且在外面套上一个含有2号转发器邮件地址的明文头。这是图8-53中1号和2号邮件转发器之间所显示的消息。最后，她再用1号邮件转发器的公钥E_1来加密整个邮件，并且套上一个含有1号转发器邮件地址的明文头。这是在图8-53中Alice右侧所显示的邮件，而且这也是她真正传送的消息。

当邮件到达1号邮件转发器时，首先外部的头被剥掉。邮件体被解密，然后将其中的邮件发送给2号邮件转发器。在其他两个邮件转发器上也执行类似的步骤。

任何人要想从最终邮件追踪到Alice将非常困难，尽管如此，许多邮件转发器还采取了附加的安全预防措施。例如，它们可能将邮件保留一段随机的时间，在邮件的末尾增加或者删除一些无关紧要的信息，对邮件重新排序；它们所做的这一切使得辨别邮件转发器的输出邮件对应于哪一个输入邮件变得更加困难，以此来抵抗流量分析攻击。有关此类邮件转发器的更多描述，请参考（Mazières和Kaashoek，1998）。

匿名需求并不仅限于电子邮件。同样的服务需求是匿名Web冲浪，使用相同的分层路径形式，其中一个节点只知道链中的下一个节点。这个方法称为**洋葱路由**（onion routing），因为每一个节点剥离了洋葱的另一层才能确定下一步应该将数据包转发到哪里。用户把他的浏览器配置成使用作为代理的匿名用户服务。Tor是此类系统的一个众所周知例子（Dingledine等，2004）。今后，所有的HTTP请求通过匿名用户网络，请求网页和将网页发回。Web网站把该匿名用户网络而不是真正的用户当作发出请求的节点。只要匿名用户网络不保留日志，事后没有人可以确定究竟是谁请求了那个网页。

8.10.2 言论自由

隐私涉及的内容是每个人希望限制其他人能够观察到的信息。第二个关键的社会问题是言论自由，以及它的对立面，即审查。所谓审查是指政府希望限制每个公民所能阅读和发表的内容。由于Web包含了数千万以上的页面，所以Web成了审查员的天堂。根据政治体制和意识形态的不同，Web站点如果包含了以下内容则可能会被禁止：

（1）少儿不宜的材料。

（2）对各种种族、宗教、性别或者其他组织有歧视倾向的言论。

（3）有关民主和民主价值的信息。

（4）与政府的版本有抵触的历史事件报道。

（5）介绍撬锁、制造武器、加密消息等技术的参考材料。

通常采取的相应措施是禁止这样的坏站点。

有时候结果是不可预测的。例如，有些公开的图书馆在它们的计算机上安装了Web过滤器，并且阻塞色情站点，以便允许儿童使用。这些过滤器一方面禁止黑名单中的所有站点，另一方面，在显示页面之前还要检查是否包含了脏字。在弗吉尼亚州Loudoun县的一个案例中，当一名赞助人通过Web搜索有关乳房癌的信息时，过滤器阻止了这一搜索请求，因为它看到了“乳房”一词。这名赞助人控告了Loudoun县。然而，在加利福尼亚州的Livermore，一名家长控告公共图书馆的理由是他们没有安装过滤器，所以她那12岁的儿子在那里浏览色情内容时被抓。图书馆该怎么办呢？

许多人都知道，万维网（World Wide Web）是一个全球范围的Web，它覆盖了整个世界。然而，对于哪些内容可以出现在Web上，并不是所有的国家都有统一的规定。例如，在2000年11月，法国的法院命令美国加利福尼亚州的Yahoo公司阻止法国的用户观看Yahoo的Web站点上有关纳粹纪念物的拍卖活动，因为按照法国的法律，拥有这样的物品是违法的。Yahoo向美国的法院提出上诉，而美国法院没有这样的禁令，因此问题演变成

谁的法律可以适用哪里，这个问题远远没有得到解决。

你可以继续想象。如果美国犹他州的一个法院命令法国阻塞所有做酒生意的Web站点，因为犹他州关于酒的法律非常严格，这些站点不符合它的法律，那么会怎么样呢？假设某国要求所有涉及民主的Web站点都被禁止，因为它们不符合国家利益。伊朗关于宗教的法律适用于更加自由的瑞典吗？沙特阿拉伯能够阻塞所有涉及妇女权利的Web站点吗？整个问题将变成一个名副其实的潘多拉盒子。

一条来自John Gilmore的很中肯的评论是："网络将审查行为看作一种破坏，并且设法绕过它。"要想了解一个具体的实现，请考虑**永恒服务**（eternity service）（Anderson，1996）。它的目标是保证已被发表的信息不可能被取消或改写。为了使用该服务，用户需要指定自己的材料将被保留多久，并且支付与材料长度和保留时间成正比的费用，然后上载材料。之后，没有人可以移除或者编辑这些材料，即使上载者也不例外。

这样的服务应该如何来实现呢？最简单的模型是使用对等系统，所存放的文档被存储在几十台参与到对等系统中的服务器上，并且每台服务器获得一部分费用，以此来鼓励服务器加入到对等系统中。系统应该使这些服务器尽可能地遍布到许多法律的管辖范围。10台随机选择的服务器的列表被安全地保存在多个地方，所以即使有些地方被攻破了，其他的仍然还在。如果当局下决心要毁掉某一个文档，那么它可能永远也无法确定自己是否找到了该文档所有的副本。该系统可以被做成自修复的，也就是说，如果它知道有些副本已经被毁坏，则剩下的站点将设法找到新的藏宝处，以便替换那些被毁坏的副本。

永恒服务是第一个抗审查系统的方案。自那以后，还有其他一些系统也被提出来了，而且有几个已经实现了。各种新的特性也被加入进来，比如加密、匿名性和容错性。保存在系统中的文件通常被分割成多个片段，每个分段被保存在许多台服务器上。这样的系统有Freenet（Clarke等，2002）、PASIS（Wylie等，2002）和Publius（Waldman等，2000）。有关其他工作的报告请参考（Serjantov，2002）。

渐渐地，越来越多国家试图对无形作品的出口进行管制，所谓无形作品通常包括Web站点、软件、科技论文、电子邮件、电话服务台等。即使像英国这样有几个世纪之久言论自由传统的国家，现在也在考虑制定更加严格的法律，例如，剑桥大学的英国教授和他的外国博士学生之间的技术讨论也被定义为需要得到政府许可的、受管制的出口行为（Anderson，2002）。无须多言，这样的政策是极有争议的。

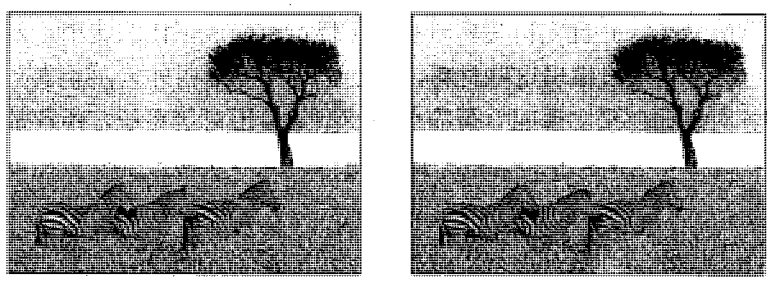
信息隐藏学

在审查制度比较严格的国家中，持不同政见者通常企图使用技术手段来躲避审查。密码学技术使得秘密消息仍然可以被发送出去（不过这样做可能不合法），但是如果政府认为Alice是一个坏人，那么仅仅凭着Bob与她进行通信这一事实就可能使他也被归到坏人这一类别中，因为即使高压政府中缺乏数学家，他们也会理解传递闭包的概念。匿名邮件转发器也许有所帮助，但如果它们被禁止只能在国内通信，并且通向国外转发器的邮件都要求得到政府的出口许可，那么它们也无济于事。但是Web仍然可以。

希望进行保密通信的人总是试图隐藏所有的通信痕迹，包括发生了通信本身这一事实。隐藏消息的学科称为**信息隐藏学**（steganography），它来源于希腊语中的词"covered

writing”。事实上，古希腊人自己也使用这种技术。希罗多德（Herodotus，希腊历史学家）写到了这样一个故事：一名将军剃光了一个信使的头，再将消息刺在他的头皮上，等到他的头发长出来之后再送他出去。现代技术在概念上是一样的，只不过它们有更高的带宽和更低的延迟。

作为一个例子，请考虑图 8-54(a)。这幅照片是本书作者在肯尼亚拍摄的，图中有 3 匹斑马凝望着 1 棵橡胶树。图 8-54(b)看起来也有同样的 3 匹斑马和 1 棵橡胶树，但是它加入了额外有趣的东西。它内嵌了 5 部莎士比亚戏剧的完整文本：Hamlet（哈姆雷特）、King Lear（李尔王）、Macbeth（麦克白）、The Merchant of Venice（威尼斯商人）和 Julius Caesar（凯撒大帝）。这些喜剧合起来总共超过了 700 KB 的文本。



(a) (b)

图 8-54

(a) 3 匹斑马和 1 棵树；(b) 3 匹斑马、1 棵树和 5 个莎士比亚完整文本

这种隐藏信道是如何做到的呢？原始的彩色图像有 1024×768 个像素。每个像素由三个 8 位整数构成，分别对应于该像素的红、绿和蓝 3 种颜色的强度。像素的颜色由这 3 种颜色线性叠加而形成。这里的信息隐藏编码方法使用了每个 RGB 颜色值的最低位作为隐蔽信道。因此，每个像素有 3 位秘密信息空间：红色值 1 位、绿色值 1 位以及蓝色值 1 位。对于像例子中大小的一幅图像，它可以存储至多 1024×768×3 位或者 294 912 字节的秘密信息。

这 5 部戏剧的完整剧本以及一份简短的评论累加起来总共有 734 891 字节。首先使用一个标准压缩算法将所有这些文本信息压缩到大约 274 KB 大小。然后利用 IDEA 算法对压缩之后的输出进行加密，接下来再将密文插入到每个颜色值的最低位中。当这幅图像被显示时，隐藏信息的存在是完全不可见的（实际上，插入的信息是不可能看得到的）。在一幅大的、全彩色的图像中，隐藏信息的存在同样是不可见的。肉眼不可能轻易区分出 21 位彩色和 24 位彩色。

在低分辨率下观看两幅黑白图像并不能体现出这项技术是多么有效。为了更好地感觉信息隐藏技术的效果，本书作者专门准备了一个演示，其中包括图 8-54(b)中内嵌了 5 部戏剧的全彩色高分辨率图像。你可以从本书的 Web 站点上找到这个演示，其中包括用来在图像中插入和提取文本的工具。

为了将信息隐藏技术用于未能被发现的通信中，持不同政见者可以创建一个 Web 站点，并在 Web 站点上放置各种无任何政治问题的图片，比如领导人的照片、本地的体育赛事、电影、电视明星等。当然，这些图片中包含有隐藏的消息。如果这些消息首先被压缩，然后再被加密的话，那么即使有人怀疑存在这些消息，他们也很难将这些消息与白噪声区分

开。当然，这些图像应该是重新扫描的；如果直接从 Internet 上复制一个图片，再改变其中的一些位，这将会泄漏其中的机密。

隐藏消息的载体并不仅限于图像。音频文件也可以工作得很好。在语音 IP 呼叫中，通过操纵包延迟、音频失真甚至是头字段都可以用来携带隐藏的信息（Lubacz 等，2010）。即使 HTML 文件中的布局和标签顺序也可以携带信息。

虽然我们是在讨论言论自由的上下文中介绍信息隐藏学，但实际上它还有许多其他用途。一种常见的用途是图像所有者编制一些秘密消息来表达他们的版权所有声明。如果这样的图像被别人偷走并放在一个 Web 站点上，则合法的所有者可以在法庭上出示隐藏的消息，以此来证明到底谁拥有这幅图像。这项技术称为**数字水印**（watermarking），在（Piva 等，2002）中进行了讨论。

有关信息隐藏学的更多信息，请参看（Wayner，2008）。

8.10.3 版权

隐私和审查恰好是技术符合公共政策的两个领域。第三个领域是版权。**版权**（copyright）是对**知识产权**（IP，Intellectual Property）创造者的一种授权，这些创造者包括作家、画家、作曲家、音乐家、摄影师、电影摄影师、舞蹈动作设计师等。这种授权是排他性的，允许他们在一定长的时间里充分使用他们的 IP，典型的时间长度是作者的一生再加上 50 年，如果是法人所有权的话，则要加上 75 年。当一件作品的版权过期以后，它就变成公有财产，任何人都可以随意地使用或者出售。例如，Gutenberg 工程（www.promo.net/pg）在 Web 上放置了数千件公有作品（比如莎士比亚、马克·吐温、狄更斯等人的作品）。1998 年，美国国会接受好莱坞的请求，将作品在美国的版权延长了 20 年，好莱坞声称如果不延长的话，则没有人再愿意制作任何作品了。与此形成鲜明对照的是，专利只持续 20 年，但人们仍然在发明各种事物。

当 Napster 这个交换音乐服务拥有 5000 万成员时，版权便成了引人注目的焦点。虽然 Napster 并没有实际复制任何音乐，但是法庭认为它拥有的中心数据库记录了谁有哪一首歌，这有助于侵犯版权，也就是说，他们帮助其他人侵犯版权。虽然没有人正式声称版权是一个坏概念（不过，许多人声称这个术语太长了，它更偏向于大公司而不是公众），但是，下一代音乐共享技术已经引发了一些主要的道德问题。

例如，考虑在一个对等网络中，人们共享合法的文件（公有的音乐、家庭视频片段、非机密的宗教宣传册等），可能还有一些受版权保护的内容。假设每个人通过 ADSL 或者有线电视全天候地保持在线连接。每台机器都对自己硬盘上的内容做了索引，同时还有一个其他成员的列表。当有人要查找某一特定作品时，他可以随机选择一个成员，看它是否有这件作品。如果没有，他可以检查此人列表中的所有成员，以及他们的列表中的所有成员，以此类推下去。计算机很擅长做这种工作。一旦找到了作品之后，请求者只要复制过来即可。

如果这件作品是受版权保护的，则请求者便侵犯了版权（不过，对于跨国家的传输，到底该遵循谁的法律，这个问题尚不清楚）。但是，对于提供作品的人又怎么样呢？你付钱购买了音乐并且将它合法地下载到自己的硬盘上，但别人有可能在你的硬盘上找到这件音

乐作品，那么，你的所作所为算是一种犯罪行为吗？如果在乡下你有一间没锁门的小屋，一个IP小偷带着一台笔记本电脑和一台扫描仪，偷偷地溜进来复制了一本版权书存到他笔记本的硬盘上，然后又溜出去了，那么，你是否犯了未保护他人版权的罪行呢？

但是，在版权方面还有更多潜在的麻烦。好莱坞和计算机工业界之间正在进行着一场巨大的斗争。前者希望对所有的知识产权都进行严格的保护，而后者并不愿意成为好莱坞的警察。1998年10月，美国国会通过了**数字千禧年版权法案**（DMCA，Digital Millennium Copyright Act），这使得破解或绕开一件版权作品中的任何保护机制，或者告诉别人怎样破解或绕开的行为也是一种犯罪。同样地，欧盟也制定了类似的法律。虽然几乎没有人认为远东的盗版商复制那些受版权保护的作品应该被允许，但是，许多人认为DMCA完全改变了版权所有者利益和公众利益之间的平衡。

举一个相关的例子。2000年9月，一个音乐界团体负责建立了一个不可破解的系统，用于在线销售音乐。它同时发起了一场竞赛，邀请人们来破解这个系统（对于任何一个新的安全系统来说，这种做法是相当正确的）。由普林斯顿大学的Edward Felten教授领导的一个小组聚集了来自多所大学的安全研究人员，他们接受了这个挑战，并且最终攻破了系统。然后他们写了一篇论文介绍他们的研究结果，并且将论文提交给USENIX的安全会议，经过同行评议之后，这篇论文被接受了。在论文发表前夕，Felten收到了来自美国唱片业协会的一封信，信中威胁他，如果他们坚持要发表这篇论文的话，则唱片业协会将根据DMCA控告论文的作者。

他们的回应是起草一份诉讼文件请求联邦法庭来裁定发表关于安全研究的科技论文是否仍然合法。慑于联邦法庭可能会做出不利的裁定，唱片业协会撤回了它的威胁，法庭也就驳回了Felten的上诉。毫无疑问，唱片工业界肯定被它的案例中暴露出来的弱点所刺激了：他们邀请人们来攻击他们的系统，然后又威胁要起诉某些接受了挑战的人。由于威胁被撤回了，所以论文得以发表（Craver等，2001）。因此，新一轮的对峙已是必定无疑了。

其间，盗版音乐和电影已经引发了对等网络的大规模增长。这让版权拥有者很不高兴，他们已经使用DMCA来采取行动。现在已经有一些搜寻对等网络的自动系统，它们在发现侵犯版权材料时会给网络操作者和那些侵犯版权的用户发出警告。在美国，这些警告是大家都知道的**DMCA删除通告**（DMCA takedown notices）。这场搜索是一个军备竞赛，因为很难可靠地扑捉到侵权者。即使你的打印机也可能会被误认为是罪魁祸首（Piatek等，2008）。

一个相关的问题是**公平使用原则**（fair use doctrine）的度，所谓原则是在各个国家中由法庭裁决建立起来的。这条原则说明了一件版权作品的购买者拥有一定的、受限制的权利来复制这件作品，包括引用部分内容作为科学研究之用、用作学校或学院的教材，以及在某些情况下当原始介质不能使用时可以做一些副本以供自己使用。合理使用的判断准则包括（1）是否出于商业目的；（2）被复制部分所占的百分比；（3）这种复制对于该作品销售的影响。由于DMCA和欧盟内部类似的法律禁止破解或者绕开作品的复制保护方案，所以这些法律也禁止合法的合理使用。实际上，DMCA将一些历史上原本属于用户的权利给了内容销售商，从而赋予销售商更多的权利。一场大的冲突已经不可避免。

尽管DMCA改变了版权所有者和用户之间的平衡，正如**可信计算组**（TCG，Trusted Computing Group）倡导的那样，**可信计算**（trusted computing）方面的发展使得DMCA相

形见绌。诸如 TCG 这样的行业机构由 Intel 和 Microsoft 这样的公司领导。TCG 的思想是在操作系统更低的层次提供谨慎监视用户各种行为（比如听盗版音乐）的支撑，以便禁止有害的行为。这是通过一个小芯片实现的，这种小芯片称为**可信平台模块**（TPM，Trusted Platform Module），它很难被篡改。现在销售的多数个人电脑销售时都配有 TPM。该系统允许内容所有者编写的软件操纵用户个人电脑，甚至以用户无法改变的方式进行。这就提出了一个问题：在可信计算中究竟谁是可信的。显然，它不是用户。无须多说，这种方案的社会影响极为广泛。工业界最终注意到了安全性，这是好事，但令人惋惜的是它把所有的焦点都瞄准在加强版权法上，而不是对付病毒、破坏者、入侵者和其他一些大多数人都很关心的安全问题上。

简而言之，在接下来的几年中，立法者和律师们将不停地忙于平衡版权所有者的经济利益与公众利益。网络空间与社会空间并没有什么不同：它经常挑起一群人与另一群人之间的竞争，从而导致权力斗争、诉讼，以及（希望）最终某种形式的决议案，至少在某一新的破坏技术出现之前是这样的。

8.11 本章总结

密码学是一个可被用来保密信息以及确保信息完整性和真实性的工具。所有现代的密码系统都建立在 Kerckhoff 原则的基础上，即算法是公开的，但密钥是保密的。许多密码学算法使用了内含置换和替代的复杂变换，从而将明文变换成密文。然而，如果量子密码学变得切实可行，那么使用一次性密钥的方法也许真的可以提供牢不可破的密码系统。

密码算法可以分成对称密钥算法和公开密钥算法。对称密钥算法将数据位通过一系列用密钥作为参数的轮变换，将明文变成密文。AES（Rijndael）和三重 DES 是目前最为常见的对称密钥算法。这些算法可被用在电码本模式、密码块链模式、流密码模式、计数器模式以及其他一些模式中。

公开密钥算法具有加密和解密使用不同密钥的特性，并且从加密密钥不可能推导出解密密钥。这些特性使得公开一个密钥（即公钥）成为可能。主要的公开密钥算法是 RSA，它的强度建立在分解大整数非常困难的基础之上。

许多法律的、商业的和其他文档需要有签名。因此，人们设计了许多种数字签名方案，有的使用对称密钥算法，有的使用公开密钥算法。通常，需要被签名的消息首先使用像 SHA-1 这样的算法来计算散列值，然后对此散列值进行签名，而不是对原始的消息进行签名。

公钥管理可以通过证书来完成，所谓证书是指将一个主角与一个公钥绑定到一起的文档。证书需要由可信的权威机构签名，或者由某一个得到可信权威机构批准的实体签名（可以如此递归下去）。信任链的根必须事先提前获取，但是浏览器通常已经内置了许多根证书。

这些密码学工具可以被用来保护网络流量。IPSec 运行在网络层上，它加密主机之间的数据包流。防火墙可以屏蔽进出一个组织的流量，屏蔽的依据通常是所用的协议和端口。虚拟专用网可以模拟老式的租用线路网络，从而提供某些期望的安全特性。最后，无线网络需要良好的安全性，以免每个人都能读取所有的消息，像 802-11i 协议就提供了这种安

全性。

当两方建立一个会话时，他们必须相互验证对方的身份；如果有必要，还要建立一个共享的会话密钥。现在已经有了许多认证协议，包括某些使用了可信的第三方，还有一些使用了 Diffie-Hellman、Kerberos 和公开密钥密码学。

电子邮件的安全性可以通过组合本章介绍的技术来加以实现。例如，首先用 PGP 压缩邮件，然后用密钥对邮件进行加密，并发送用接收方公钥加密的这个密钥。此外，它也计算邮件的散列值，并且将签过名的散列值发送出去以便验证邮件的完整性。

Web 安全也是一个重要的话题，起点是安全的命名机制。DNSsec 提供了一种预防 DNS 欺骗的方法。大多数电子商务 Web 站点使用 SSL/TLS 在客户和服务器之间建立安全和经过认证的会话。另外，有许多技术被用来处理移动代码，特别是沙箱和代码签名技术。

Internet 引发了许多导致技术与公共政策强烈交汇的问题。这样的领域包括隐私、言论自由和版权。

习　题

1. 请破解下面的单字母置换密码。明文仅由字母组成，它是从 Lewis Carroll 的诗中摘录下来的名句：

   mvyy bek mnyx n yvjjyr snijrh invq n muvjvdt je n idnvy

   jurhri n fehfevir pyeir oruvdq ki ndq uri jhrnqvdt ed zb jnvy

   Irr uem rntrhyb jur yeoijrhi ndq jur jkhjyri nyy nqlndpr

   Jurb nhr mnvjvdt ed jur iuvdtyr mvyy bek pezr ndq wevd jur qndpr

   mvyy bek, medj bek, mvyy bek, medj bek, mvyy bek wevd jur qndpr

   mvyy bek, medj bek, mvyy bek, medj bek, medj bek wevd jur qndpr

2. 仿射密码是单字母置换密码的一种版本，其中 m 个字母中的字母首先被映射成 0 到 m–1 的整数。随后，表示每个明文字母的整数被转换成一个表示对应密文的字母的整数。单个字母的加密函数是 E(x) = (ax + b) mod m，其中 m 是字母表的尺寸，a 和 b 是密文的密钥，并且它们是共素数（co-prime）。Trudy 发现 Bob 使用仿射密码来产生密文。她得到一份密文的副本，并且发现在密文中出现频率最大的字母是“R”，出现频率第二高的字母是“K”。请显示 Trudy 如何可以破解密文并得到明文。

3. 破解下面柱形替代密码。明文是从一本流行的计算机教材中摘录的，所以“computer”是一个可能的单词。明文全部由字母组成（没有空格）。为了方便阅读，密文被分割成 5 个字符的块。

   aauan cvlre rurnn dltme aeepb ytust iceat npmey iicgo gorch srsoc

   nntii imiha oofpa gsivt tpsit lbolr otoex

4. Alice 使用替代密码来加密她给 Bob 的消息。为了增强安全性，她又使用置换密码来加密替代密码的密钥，并且在她的机器上保留加密密码。Trudy 设法得到了加密后的替代密码的密钥。请问，Trudy 是否能解密 Alice 给 Bob 的消息？并解释为什么能？或者为什么不能？

5. 从图 8-4 中的密文中找到用来产生“Hello World”文本的 77 位一次性密钥。
6. 假设你是个密探，为了方便起见，假设你有一个具有无限数量书的图书馆任由你处置。你的操作员在他的处置库里也有这样一个图书馆。你们已经事先使用指环王（Lord of the Rings）作为一次性密钥。请解释你们如何能够使用这样的资产来产生一个无限长的一次性密钥。
7. 量子密码学需要一个能根据需要激发单个光子（携带 1 位信息）的光子枪。在这个习题中，请计算在一条 250 Gbps 的光纤链路上，一位携带多少个光子。假设光子的长度等于它的波长，波长为 1 微米。光纤中的光速为 20 厘米/纳秒。
8. 假设一个系统使用了量子密码技术，如果 Trudy 能够捕获并重新生成光子，那么，它将会得到一些错误的位，从而导致在 Bob 的一次性密钥中出现错误。试问，平均 Bob 的一次性密钥中错误的位占多大的比例？
9. 一条基本的密码学原则规定所有的消息必须有冗余度。但是，我们也知道，消息中的冗余可以帮助一个入侵者来识别一个猜测的密钥是否正确。请考虑两种冗余形式。第一，明文的前 n 位包含一种已知的模式；第二，消息的最后 n 位包含了该消息的一个散列值。从安全的角度来看，这两种形式是等价的吗？请解释你的答案。
10. 在图 8-6 中，P 盒和 S 盒交替出现。尽管这种安排令人审美愉悦，但是相比先通过所有的 P 盒再通过所有的 S 盒形式，它会更加安全吗？
11. 假设 DES 的明文中只包含大写 ASCII 字母、空格、逗号、句号、冒号、回车符和换行符，请根据这样的知识设计一种破解 DES 的方法。没有人知道有关明文的奇偶校验位。
12. 在正文中我们计算过，假设一个处理器每 1 纳秒能够分析一个密钥，那么，一台具有 100 万个处理器的密码破译机将需要 10^{16} 年时间才能破解 128 位版本的 AES。试问需要计算多久才能把这个破译时间下降到 1 年，当然这个时间还是太长。为了达到这个目标，我们需要计算机快 10^{16} 倍。如果摩尔定律持续有效（即每 18 个月计算能力翻一番），则需要多少年才能制造出能将破译时间下降到一年的并行机器？
13. AES 支持 256 位的密钥。试问 AES-256 有多少个密钥？看看你是否可以在物理、化学或者天文学中找到同样规模的数值。你可以使用 Internet 来帮助查找大的数值。根据你的调查研究，请给出一个结论。
14. 假设一条消息已被使用计数器模式的 DES 进行了加密。块 C_i 中的一位密文在传输过程中被偶然地从 0 变成了 1。试问结果将有多少明文被弄乱？
15. 现在再来考虑密码块链模式。这次不再是一个 0 位被转换成了 1 位，而是一个额外的 0 位被插入到块 C_i 之后的密文流中。同样地，试问结果将有多少明文被弄乱？
16. 请从传输一个大文件所需要的加密操作次数的角度，来比较一下密码块链模式与密码反馈模式。试问哪个效率更高？高多少？
17. 使用 RSA 公开密钥密码，并有 a = 1, b = 2, …, y = 25, z = 26：
    （1）如果 p = 5 和 q = 13，列出 d 的 5 个合法值。
    （2）如果 p = 5，q = 31 和 d = 37，找出 e。
    （3）使用 p = 3，q = 11 和 d = 9，找出 e 和加密“hello”。
18. Alice 和 Bob 在通信中使用 RSA 公开密钥加密。Trudy 发现了 Alice 和 Bob 共享一个用

来确定公钥对数目 n 的素数。换句话说，Trudy 发现 $n_a = p_a \times q$ 和 $n_b = p_b \times q$。试问，Trudy 如何使用这信息来破解 Alice 的代码？

19. 考虑使用计数器模式，如图 8-15 所示，但是让 IV = 0。试问，使用 0 是否通常会威胁到密文的安全性？
20. 在图 8-20 中，我们看到 Alice 如何给 Bob 发送一个签名消息。如果 Trudy 替换了 P，Bob 就会检测出来。但是如果 Trudy 将 P 和签名都换了，试问结果会如何呢？
21. 数字签名有一些由于懒惰的用户而引起的潜在弱点。在电子商务交易中，有可能会形成一份合同，并要求用户对它的 SHA-1 散列值进行签名。如果用户没有真正对这份合同和对应的散列值进行验证，那么他有可能无意中签了另一份完全不同的合同。假设黑手党试图利用这个弱点来套一笔钱。他们建立了一个付费的 Web 站点（比如色情或者赌博站点，等等），并且要求新的顾客提供信用卡号码。然后，他们发送了一份合同，声称顾客希望使用他们的服务并通过信用卡来支付费用，而且他们要求顾客对这份合同进行签名，实际上，他们知道大多数顾客根本不检查合同和散列值的一致性，就会对散列值进行签名。请说明黑手党如何从一家合法的 Internet 珠宝商处购买一批钻石，并且将费用记到那些轻信的顾客身上。
22. 一个数学班有 20 名学生。假设所有的学生都出生在上半年——1 月 1 日到 6 月 30 日。试问，至少两名学生具有相同生日的概率为多少？假设这个班上没有人在闰日（即 2 月 29 日）出生，所以，总共有 181 种可能的生日。
23. 当 Ellen 向 Marilyn 坦白了自己在 Tom 的终身职位事件中所做的欺骗行为以后，Marilyn 决定将想要发送的邮件内容口授下来，记录在一台录音机里，然后让她的新秘书仅仅完成键盘输入工作，以免再次发生同样的问题。然后 Marilyn 打算在邮件被输入以后，她在自己的终端上检查这些邮件，由此确保邮件中的内容是她的本意。试问，新秘书仍然可能使用生日攻击来伪造邮件吗？如果可能，她该怎么办？（提示：她能够。）
24. 考虑图 8-23 中 Alice 未能获得 Bob 公开密钥的情形。假设 Bob 和 Alice 已经共享了一个秘密密钥，但是 Alice 仍然想要 Bob 的公钥。试问，现在 Alice 有办法安全地获得 Bob 的公钥吗？如果可以的话，她该怎么做？
25. Alice 想要利用公开密钥密码技术与 Bob 进行通信。她与某个人建立了一个连接，她希望这个人就是 Bob。她请他出示他的公钥，然后他以明文方式将公钥和一个由根 CA 签名的 X.509 证书一起发送给 Alice。Alice 早就有根 CA 的公钥。为了证明她的确在跟真正的 Bob 进行通信，Alice 需要执行哪些步骤来验证这点？假设 Bob 并不关心他在跟谁通话（比如，Bob 是某一种公共服务）。
26. 假设系统使用了一个基于树状 CA 层次结构的 PKI。Alice 希望与 Bob 进行通信，在与 Bob 建立起通信信道以后，她收到 Bob 发送过来的一个证书，该证书的签名者是一个 CA，我们称为 X。假设 Alice 从来没有听说过 X。试问，Alice 该执行哪些步骤来验证自己确实在跟 Bob 通话？
27. 假设一台机器位于 NAT 盒子之后。试问，IPSec 还能使用传输模式的 AH 吗？请解释你的答案。
28. Alice 想用 SHA-1 散列值给 Bob 发送一个消息。她请教你关于可用的合适的签名算法。请问，你将给她什么建议？

29. 请说出一个理由，说明为什么可以把防火墙配置成对入境流量进行检查。再说出一个理由，说明为什么也可以对防火墙进行配置，以便检查出境流量。试问，你认为这两种检查可能成功吗？
30. 假设一个组织使用了 VPN，以便通过 Internet 将它的多个站点安全地连接起来。对于这个组织中的一个用户 Jim 来说，使用 VPN 与他的老板 Mary 通信。请描述 Jim 和 Mary 之间可采纳的一种通信类型，在通信过程中不需要使用加密或者其他安全机制。请再描述另一种通信类型，在通信中需要加密或者其他的安全机制。请解释你的答案。
31. 请对图 8-34 所示的协议中的一条消息做修改，使得它能够抵抗反射攻击。请解释你的修改是如何发挥作用的。
32. Diffie-Hellman 密钥交换协议可以被用在 Alice 和 Bob 之间建立一个秘密密钥。Alice 发送给 Bob 的是(227, 5, 82)。Bob 的响应（125）。Alice 的秘密数 x 是 12，Bob 的秘密数 y 是 3。试问，Alice 和 Bob 是如何计算出秘密密钥的？
33. 两个用户可以建立一个使用 Diffie-Hellman 算法的共享密钥，即使他们从来没有见过面、没有共享过密钥以及没有证书。
    （1）解释这算法为什么容易受到中间人攻击。
    （2）如果 n 或 g 是秘密的，该算法的易受攻击性会发生变化吗？
34. 在图 8-39 的协议中，试问为什么 A 在明文中发送加密的会话密钥？
35. 在 Needham-Schroeder 协议中，Alice 产生两个临时值 R_A 和 R_{A2}。这看起来有点多余。试问，一个就无法胜任工作吗？
36. 假设一个组织使用 Kerberos 来认证。根据安全性和服务的可用性，如果 AS 或者 TGS 关闭会产生什么效果？
37. Alice 正在使用图 8-43 中的公钥认证协议来认证与 Bob 的交谈。然而，在发送消息 7 时，Alice 忘了加密 R_B。Trudy 现在知道了 R_B 的值。试问，现在 Alice 和 Bob 是否需要使用新的参数重复认证步骤以便保证交谈是安全的？请解释你的回答。
38. 如图 8-43 所示的公钥认证协议，在消息 7 中 R_B 被用 K_S 加密。试问，这种加密是否必要？以明文发送回来是否足够安全？请解释你的回答。
39. 使用磁卡和 PIN 码的销售点终端有一个致命的缺陷：一个恶意的商家可以修改他的读卡器，以便将用户磁卡上的所有信息以及 PIN 码捕捉并保存下来，从而将来可以寄送另外的（伪造的）交易。下一代销售点终端使用的卡将配备完整的 CPU、键盘和微显示器。请为这种系统设计一个协议，使得恶意的商家无法攻破系统。
40. 试问，组播一个 PGP 消息是否有可能？这将受到什么限制？
41. 假设 Internet 上的每个人都使用 PGP。试问，可以将一个 PGP 消息发送给一个任意的 Internet 地址，并正确解码获得所有信息吗？请讨论你的答案。
42. 如图 8-47 所示的攻击中遗漏了一步。这一步对于欺骗工作是不需要的，但包括了这一步可以减少事后的潜在可疑性。试问，缺少的步骤是什么？
43. SSL 数据传输协议涉及两个临时值和一个预设主密钥。试问，如果有的话，使用的临时值为多少？
44. 考虑 2048×512 像素的图像。假如你想加密的文件大小为 2.5 MB。试问，利用这个图像你能加密的文件比例多少？如果你将文件压缩到其原始大小的四分之一，那么能加

密的文件比例又为多少？请说明你的计算过程。

45. 如图 8-54(b)所示的图像包含了莎士比亚的 5 部戏剧的 ASCII 文本。试问，有可能在斑马之间隐藏音乐而不是文本吗？如果可能，试问它将如何工作，并且可以在这幅画中隐藏多少音乐呢？如果不行，请解释为什么不行？
46. 给定一个大小为 60 MB 的文本文件，采用信息隐藏学的方法利用一个图像文件中每个颜色的低序位来进行加密。试问，为了加密整个文件，图像的尺寸应该多大？如果文件事先被压缩到原来文件的 1/3 大小，试问图像的尺寸又需要多大？请用像素来说明你的计算过程。假设图像的宽高比是 3∶2，像素为 3000×2000。
47. Alice 是一个大量使用第一类匿名邮件转发器的用户。她在自己喜欢的新闻组 alt.fanclub.alice 上张贴了许多消息，每个人都知道这些消息来自 Alice，因为它们有同样的笔名。假设邮件转发器的工作一切正常，Trudy 不可能模仿 Alice。在第一类邮件转发器全部停机以后，Alice 切换到一台加密朋克邮件转发器上，并且在她的新闻组中开始一个新的讨论话题。请为她设计一种方法可用它来阻止 Trudy 模拟 Alice 在新闻组中张贴新的消息。
48. 请搜索 Internet，找到一个涉及隐私的有趣案例，并写下一页纸的报告。
49. 请搜索 Internet，找到一些涉及版权与公平使用的法院案例，并写下一页纸的报告总结你的调查结果。
50. 编写一个程序，它用一个密钥流通过 XOR 操作来加密输入数据。寻找或者编写一个尽可能好的随机数生成器，你可用它来生成密钥流。这个程序应该起到一个过滤器的作用，它接受来自标准输入的明文，并在标准输出上产生密文（或者反之）。该程序应该有一个参数，即作为随机数生成器种子的密钥。
51. 写出一个计算数据块 SHA-1 散列值的过程。该过程应该含有两个参数：一个指向输入缓冲区的指针，另一个指向大小为 20 字节输出缓冲区的指针。若要了解确切的 SHA-1 规范，可以在 Internet 搜索 FIPS 180-1，它是完整的规范说明。
52. 写一个函数，该函数接受一个 ASCII 字符流，并用密码块链模式的置换密码加密这个输入流。块的大小是 8 字节。 程序应该从标准输入获取明文，并从标准输出打印密文。本习题允许你选择任何合理方法来确定输入流的结束以及/或者何时应该填充以便达到一个完整的块。你也可以选取任何的输出格式，只要它是无二义性的。程序应该接受两个参数：

    （1）一个初始向量的指针。

    （2）一个表示置换密码漂移数的 k，表示每一个 ASCII 字母都用字母表中其前第 k 个字母加密。

    例如，如果 x = 3，然后 A 用 D 加密，B 用 E 加密等。做出关于到达 ASCII 字符集最后一个字母的合理假设。确保在你的代码里清楚地记录任何有关输入和加密算法的假设。
53. 这个练习的目的是让你更好地了解 RSA 机制。写一个函数，它接收作为参数的素数 p 和 q，使用这些参数计算公共和私人 RSA 密钥，并且把 n、z、d 和 e 作为打印输出发送到标准输出。这个函数还应该接受一个 ASCII 字符流，并且使用计算出来的 RSA 密

钥加密该输入。程序应该从标准输入接收明文，并在标准输出打印出密文。加密应该基于字符进行，即从输入得到每个字符并加密它，而且加密过程独立于其他的字符。对于本习题，你可以选取任何合理的系统来确定到达了输入流的末尾，你也可以选取任何的输出格式，只要它含义清晰。确保在你的代码里清楚地记录任何有关输入和加密算法的假设。

第 9 章　阅读清单和参考书目

现在我们已经完成了计算机网络的学习，但这仅仅是个开始。由于篇幅有限，许多有趣的话题无法尽可能多地展示细节，其他一些有趣的话题则被完全省略了。在本章，我们为那些希望继续学习计算机网络的读者，给出了一些进一步阅读的建议和一些参考书目。

9.1　进一步阅读的建议

计算机网络的各个方面都有广泛的文献资料。这个领域发表科研论文的两个有影响力的期刊分别是 *IEEE/ACM Transactions on Networking* 和 *IEEE Journal on Selected Areas in Communications*。

ACM 数据通信特殊兴趣组（SIGCOMM）和移动系统用户数据和计算特殊兴趣组（SIGMOBILE）定期发表许多论文，并且特别关注新出现的主题。它们是 *Computer Communication Review* 和 *Mobile Computing and Communications Review*。

IEEE 也定期出版 3 个期刊 *IEEE Internet Computing*、*IEEE Network Magazine* 和 *IEEE Communications Magazine*——包含与网络有关的综述、教程和案例研究。前两个杂志强调体系结构标准和软件，后一个倾向于通信技术（光纤、卫星等）。

还有一些每年或每两年召开的会议，吸引了众多的网络论文。特别是，一些 SIGCOMM 会议，包括网络系统设计与实现研讨会（NSDI，Symposium on Networked Systems Design and Implementation）、移动系统应用及服务会议（MobiSys，Conference on Mobile Systems, Applications, and Services）、操作系统原理研讨会（SOSP，Symposium on Operating Systems Principles）和操作系统设计和实现研讨会（OSDI，Symposium on Operating Systems Design and Implementation）。

下面我们按照本书的章节列出了一些补充阅读的建议。这里给出的许多建议正是本书有关章节采纳的素材，它们给出了一些教程和综述。9.2 节给出了全部的引用列表。

9.1.1　概论与综合论著

Comer, *The Internet Book*, 4th ed.

任何一个希望快速进入 Internet 大门的读者都应该阅读这本书。Comer 以初学者能理解的方式介绍了 Internet 的历史、成长、技术、协议和服务，由于该书覆盖了如此多的素材，所以即使对于技术员来说，该书阅读起来也饶有趣味。

***Computer Communication Review*, 25th Anniversary Issue, Jan. 1995**

想获得 Internet 如何发展第一手材料的读者，这期刊物收集到了 1995 年以前的重要文件，包括有关 TCP、组播、DNS、以太网以及整体体系结构发展的文章。

Crovella and Krishnamurthy, *Internet Measurement*

我们怎样才能知道 Internet 是否工作良好？这个问题并不容易回答，因为没有人在负责 Internet 的运行。这本书介绍了已开发的网络测量技术，这些技术可用来测量 Internet 的运行状况，从底层网络基础设施到上层应用均可测量。

***IEEE Internet Computing*, Jan.–Feb. 2000**

在新千禧年发布的 *IEEE Internet Computing* 第一个问题正是你所期望的：它要求在上一个千禧年帮助创建了 Internet 的人们推测它在下一个千禧年会发展成什么样。这些专家包括 Paul Baran、Lawrence Roberts、Leonard Kleinrock、Stephen Crocker、Danny Cohen、Bob Metcalfe、Bill Gates、Bill Joy 和其他专家。了解他们预测的 Internet 在这超过十年的进展是否吻合。

Kipnis, "Beating the System: Abuses of the Standards Adoption Process"

标准委员会在工作中尽量做到公平，并在各供应商之间保持中立。但不幸的是，总有一些公司试图滥用系统。例如，这样的事件屡屡发生，一个公司帮助开发了一个标准，然后在该项标准被批准后，公司就宣布该标准基于它所拥有的一项专利。它给喜欢的公司发放许可，而那些它不喜欢的公司就得不到许可，至于发放许可的价格则由它单独决定。若想了解标准化的黑暗那一面，这篇文章是一个良好的敲门砖。

Hafner and Lyon, *Where Wizards Stay Up Late*

Naughton, *A Brief History of the Future*

究竟是谁发明了 Internet？很多人把功劳归为自己。这么说也正确，因为很多人以不同的方式援手了 Internet 的发展。其中包括撰写了一份描述包交换报告的 Paul Baran、各大学中参与了 ARPANET 体系结构设计的学者、编程实现了第一个 IMP 的 BBN 研究人员，以及发明了 TCP/IP 的 Bob Kahn 和 Vint Cerf 等。这些书诉说了有关 Internet 的故事，内容至少涵盖到 2000 年，其间充满了许多奇闻逸事。

9.1.2 The Physical Layer

Bellamy, *Digital Telephony*, 3rd ed.

如果想回头看另一个重要网络——电话网络，那么这本权威的书包含了你曾经想知道的一切，甚至更多，特别有趣的是关于传输和多路复用、数字交换、光纤、移动电话和 DSL 章节。

Hu and Li, "Satellite-Based Internet: A Tutorial"

通过卫星访问 Internet 与使用地面线路访问 Internet 完全不同。这里不仅存在延迟的问题，路由和交换也不尽相同。在本文中，作者探讨了利用卫星接入 Internet 的相关问题。

Joel, "Telecommunications and the IEEE Communications Society"

有关电信的历史，从电报开始到 802.11 结束，这篇文章给出了紧凑但出人意料全面的描述，值得一看。它还涵盖了无线电、电话、模拟和数字交换机、海底电缆、数字传输、

广播电视、卫星、有线电视、光纤通信、移动电话、包交换、ARPANET 以及 Internet 等内容。

Palais, *Fiber Optic Communication*, 5th ed.

有关光纤技术的书籍往往针对专业技术人员，但这本书面向大多数读者。它包括波导、光源、光探测器、耦合器、调制、噪声和许多其他议题。

Su, *The UMTS Air Interface in RF Engineering*

这本书提供了一个主要 3G 蜂窝系统的详细介绍。它侧重于空中接口，或移动电话和网络基础设施之间所用的无线协议。

Want, *RFID Explained*

这本书是一本容易阅读的入门书，描述了不寻常的 RFID 物理层技术如何工作。它涵盖了 RFID 的各个方面，包括其潜在的应用。本书还包括了一些取自真实世界的 RFID 部署实例和从中获得的经验。

9.1.3 数据链路层

Kasim, *Delivering Carrier Ethernet*

如今，以太网已不仅仅是一个局域网技术。作为运营商级的以太网已俨然成为长距离链路的新贵。这本书汇集了这方面的详细资料。

Lin and Costello, *Error Control Coding*, 2nd ed.

检错和纠错编码是可靠计算机网络的核心。这本流行的教科书解释了一些最重要的编码技术，从简单的线性海明码到更复杂的低密度奇偶校验码。它试图把涉及的必要代数压到最小，但仍然有很多内容。

Stallings, *Data and Computer Communications*, 9th ed.

第二部分包括了数字数据传输和各种不同的链路，包括差错检测、带有重传机制的差错控制和流量控制。

9.1.4 介质访问控制子层

Andrews et al., *Fundamentals of WiMAX*

这本书给出了有关 WiMAX 技术全面的深入讨论，从无线宽带的想法到采用 OFDM 和多天线的无线技术，以及多路访问系统。针对这个包含大量技术的议题，你将发现本书教程形式的叙述是最容易接受的。

Gast, *802.11 Wireless Networks*, 2nd ed.

对于 802.11 技术和协议的可读性介绍，本书是一个良好的开端。它从 MAC 子层开始，然后介绍了不同的物理层技术，包括安全性。然而，第 2 版对 802.11n 没有给出足够的更新内容。

Perlman, *Interconnections*, 2nd ed.

对于通用网桥、路由器和路由器的权威而非一般性的讨论，Perlman 的书值得一看。作者设计的算法被用在了 IEEE 802 生成树网桥，而且她还是网络领域不同方面的世界级领袖权威人物之一。

9.1.5 网络层

Comer, *Internetworking with TCP/IP*, Vol. 1, 5th ed.

Comer 已经撰写了有关 TCP / IP 协议套件如何工作的书籍，现在已经到了第 5 版。本书前一半主要关注网络层的 IP 和相关协议，其他各章主要涉及更高层次，也值得一读。

Grayson et al., *IP Design for Mobile Networks*

传统的电话网络和 Internet 进入到了一个全面碰撞的历程中，移动电话网络正在其内部实施 IP。这本书讲述了如何用 IP 协议设计一个支持移动电话服务的网络。

Huitema, *Routing in the Internet*, 2nd ed.

如果你想深入了解路由协议，这是一本很好的书。无论是可发音的算法（例如，RIP 和 CIDR）还是无法发音的算法（例如 OSPF、IGRP 和 BGP），本书都对此做了非常详细的讨论。因为这是一个比较旧的书，因此没有包括较新的发展状况，但是对于本书涵盖的内容则做了很好的解释。

Koodli and Perkins, *Mobile Inter-networking with IPv6*

在一卷中提出了网络层的两个重要发展：IPv6 和移动 IP。本书对这两个主题做了全面的描述，Perkins 是移动 IP 背后的驱动人物之一。

Nucci and Papagiannaki, *Design, Measurement and Management of Large-Scale IP Networks*

我们谈了许多关于网络如何工作，但没有谈多少关于如何设计、部署和管理一个网络。如果你是 ISP，那么这本书填补了这方面的缺陷。本书考查了流量工程中的一些现代方法，以及 ISP 如何利用网络为客户提供服务的方法。

Perlman, *Interconnections*, 2nd ed.

从第 12～15 章，Perlman 介绍了许多单播和组播路由算法设计所涉及的问题，这些算法同时考虑了广域网和局域网。但到目前为止，这本书最好的部分是第 18 章，作者将她在网络协议方面的多年经验提炼为一个集知识性和趣味性为一体的篇章。这是协议设计者的必读之书。

Stevens, *TCP/IP Illustrated*, Vol. 1

第 3～10 章提供了关于 IP 和相关协议（ARP、RARP 和 ICMP）的综合讨论，并通过实例加以说明。

Varghese, *Network Algorithmics*

我们花了很多时间来讨论路由器和其他网络元素是如何彼此交互的。这本书与众不同：它关注如何实际设计路由器，使其以惊人的速度转发数据包。对于内幕消息和相关问题的讨论，这是一本很好的阅读书。在实践中如何通过巧妙的算法以软硬件方式实现高速网络元素，作者是这方面的权威。

9.1.6 传输层

Comer, *Internetworking with TCP/IP*, Vol. 1, 5th ed.

如上所述，Comer 已经撰写了 TCP / IP 协议族如何工作。这本书的后半部分是关于 UDP 和 TCP。

Farrell and Cahill, *Delay- and Disruption-Tolerant Networking*

这个简短的书是针对“挑战网络”体系结构、协议和应用的深入探讨，这种网络必须在苛刻的连接条件下运行。作者作为 IETF 的 DTN 研究小组成员曾参与了 DTN 的开发。

Stevens, *TCP/IP Illustrated*, Vol. 1

第 17～24 章结合实例综合讨论了 TCP。

9.1.7 应用层

Berners-Lee et al., “The World Wide Web”

时光倒流，作为 Web 发明人及其他在欧洲核子研究中心的一些同事们，他们是如何看待 Web 的。本文重点介绍了 Web 体系结构、URL、HTTP 和 HTML，以及未来发展方向，并将 Web 和其他分布式信息系统进行了比较。

Held, *A Practical Guide to Content Delivery Networks*, 2nd ed.

这本书给出了 CDN 如何工作的详实描述，重点强调在设计和运行一个工作良好 CDN 时该有的实际考虑。

Hunter et al., *Beginning XML*, 4th ed.

有很多很多关于 HTML、XML 和 Web 服务的书籍。这本有 1000 页的书涵盖了你想了解的大多数内容。它不仅解释了如何编写 XML 和 XHTML，还描述了如何开发 Web 服务，这些服务使用 AJAX、SOAP 和其他实际常用的技术来产生和处理 XML。

Krishnamurthy and Rexford, *Web Protocols and Practice*

这本书包括了 Web 的所有方面，很难找到一本比这本更全面的书。正如你所期待的，它涵盖了客户端、服务器、代理和缓存。还有些章节涉及 Web 流量和测量，也包括一些章节给出了有关 Web 的当前研究和改进状况。

Simpson, *Video Over IP*, 2nd ed.

作者广泛考查了如何运用 IP 技术通过网络移动视频，同时针对 Internet 和专门设计用

来运载视频的专用网络做了详细的描述。有趣的是，这本书面向学习网络知识的视频专业人员，而不是其他方式的指导书。

Wittenburg, *Understanding Voice Over IP Technology*

这本书介绍了语音如何通过 IP 传递，从采用 IP 协议携带音频数据以及服务质量问题，到 SIP 和 H.323 协议族。这些都是必须的详细材料，但被分解成易于消化吸收的单位。

9.1.8　网络安全

Anderson, *Security Engineering*, 2nd. ed.

这本书提出了一个安全技术的奇妙组合，呈现在人们面前帮助大家进一步理解人们是如何使用和滥用这些技术的。这本书的技术性比 *Secrets and Lies* 更强，但是比 *Network Security* 一书（见下面）弱。该书在简单地介绍了基本的安全技术以后，所有的章节都用在介绍各种各样的应用，包括银行系统、核命令和控制、安全打印、生物测量学、物理安全性、电子战争、电信安全、电子商务和版权保护。

Ferguson et al., *Cryptography Engineering*

许多书籍告诉你流行的加密算法是如何工作的。这本书告诉你如何使用加密技术——为什么加密协议是这样设计的，以及如何把这些技术应用到一个系统中来满足你的安全目标。这是一本相当紧凑的书，是任何人设计依赖于加密技术系统的必备读物。

Fridrich, *Steganography in Digital Media*

信息隐藏术可以追溯到古希腊，蜡被熔化后形成一空白片，因此秘密信息可以在蜡被溶化覆盖前写到衬底的木片上。如今，视频、音频和 Internet 上的其他内容提供了传送秘密消息的不同载体。本书讨论了通过图像隐藏和寻找信息的各种现代技术。

Kaufman et al., *Network Security*, 2nd ed.

这是一本有关网络安全算法和协议的大全，它提供了更多的技术信息，并且兼具权威性和诙谐性于一体。秘密和公开密钥算法和协议、消息散列、认证、Kerberos、PKI、IPSec、SSL/TLS 和邮件安全都得到了细致的解释，并具备相当长的篇幅，给出许多例子说明。第 26 章，关于安全传说则是一个真正的宝库。在安全性方面，麻烦体现在细节上。任何人在规划设计一个安全系统时，应该学习本章给出的来源于现实世界的建议，尤其是当设计的系统将要被实际使用时，更应该好好学习本章内容。

Schneier, *Secrets and Lies*

如果你从头到尾阅读 *Cryptography Engineering* 一书，你将会了解到有关加密算法的所有方面。如果你再逐章阅读 *Secrets and Lies* 一书（少花不少时间），你将了解到加密算法不是故事的全部。大多数的安全弱点并不在于算法的失败或者甚至密钥太短，而是所在安全环境中存在的缺陷。对于在最广泛意义上的计算机安全，这本书给出了非技术性的和引人入胜的讨论，是一本很好阅读的书。

Skoudis and Liston, *Counter Hack Reloaded*, 2nd ed.

阻止黑客的最好办法是以黑客的思维来思考问题。这本书展示了黑客如何看待网络的，并且提出了一个观点：安全应该是整个网络设计的功能，而不是基于某个特定技术的事后功能。它涵盖了几乎所有常见的攻击，其中包括“社会工程”类型的攻击。此类攻击利用了对计算机安全性措施不熟悉的用户来实施攻击。

9.2 按字母顺序参考书目

ABRAMSON, N.: “Internet Access Using VSATs,” *IEEE Commun. Magazine*, vol. 38, pp. 60–68, July 2000.

AHMADI, S.: “An Overview of Next-Generation Mobile WiMAX Technology,” *IEEE Commun. Magazine*, vol. 47, pp. 84–88, June 2009.

ALLMAN, M., and PAXSON, V.: “On Estimating End-to-End Network Path Properties,” *Proc. SIGCOMM '99 Conf.*, ACM, pp. 263–274, 1999.

ANDERSON, C.: *The Long Tail: Why the Future of Business is Selling Less of More*, rev. upd. ed., New York: Hyperion, 2008a.

ANDERSON, R.J.: *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed., New York: John Wiley & Sons, 2008b.

ANDERSON, R.J.: “Free Speech Online and Offline,” *IEEE Computer*, vol. 25, pp. 28–30, June 2002.

ANDERSON, R.J.: “The Eternity Service,” *Proc. Pragocrypt Conf.*, CTU Publishing House, pp. 242–252, 1996.

ANDREWS, J., GHOSH, A., and MUHAMED, R.: *Fundamentals of WiMAX: Under- standing Broadband Wireless Networking*, Upper Saddle River, NJ: Pearson Education, 2007.

ASTELY, D., DAHLMAN, E., FURUSKAR, A., JADING, Y., LINDSTROM, M., and PARKVALL, S.: “LTE: The Evolution of Mobile Broadband,” *IEEE Commun. Magazine*, vol. 47, pp. 44–51, Apr. 2009.

BALLARDIE, T., FRANCIS, P., and CROWCROFT, J.: “Core Based Trees (CBT),” *Proc. SIGCOMM '93 Conf.*, ACM, pp. 85–95, 1993.

BARAN, P.: “On Distributed Communications: I. Introduction to Distributed Communication Networks,” *Memorandum RM-420-PR*, Rand Corporation, Aug. 1964.

BELLAMY, J.: *Digital Telephony*, 3rd ed., New York: John Wiley & Sons, 2000.

BELLMAN, R.E.: *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.

BELLOVIN, S.: “The Security Flag in the IPv4 Header,” RFC 3514, Apr. 2003.

BELSNES, D.: “Flow Control in the Packet Switching Networks,” *Communications Net- works*, Uxbridge, England: Online, pp. 349–361, 1975.

BENNET, C.H., and BRASSARD, G.: “Quantum Cryptography: Public Key Distribution and Coin Tossing,” *Int'l Conf. on Computer Systems and Signal Processing*, pp.175–179, 1984.

BERESFORD, A., and STAJANO, F.: “Location Privacy in Pervasive Computing,” *IEEE Pervasive Computing*, vol. 2, pp. 46–55, Jan. 2003.

BERGHEL, H.L.: “Cyber Privacy in the New Millennium,” *IEEE Computer*, vol. 34, pp.132–134, Jan. 2001.

BERNERS-LEE, T., CAILLIAU, A., LOUTONEN, A., NIELSEN, H.F., and SECRET, A.: “The World Wide Web,” *Commun. of the ACM*, vol. 37, pp. 76–82, Aug. 1994.

BERTSEKAS, D., and GALLAGER, R.: *Data Networks*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.

BHATTI, S.N., and CROWCROFT, J.: "QoS Sensitive Flows: Issues in IP Packet Handling," *IEEE Internet Computing*, vol. 4, pp. 48–57, July–Aug. 2000.

BIHAM, E., and SHAMIR, A.: "Differential Fault Analysis of Secret Key Cryptosystems," *Proc. 17th Ann. Int'l Cryptology Conf.*, Berlin: Springer-Verlag LNCS 1294, pp.513–525, 1997.

BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P.A., KUTTEN, S., MOLVA, R., and YUNG, M.: "Systematic Design of a Family of Attack-Resistant Authentication Proto- cols," *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 679–693, June 1993.

BIRRELL, A.D., and NELSON, B.J.: "Implementing Remote Procedure Calls," *ACM Trans. on Computer Systems*, vol. 2, pp. 39–59, Feb. 1984.

BIRYUKOV, A., SHAMIR, A., and WAGNER, D.: "Real Time Cryptanalysis of A5/1 on a PC," *Proc. Seventh Int'l Workshop on Fast Software Encryption*, Berlin: Springer-Verlag LNCS 1978, pp. 1–8, 2000.

BLAZE, M., and BELLOVIN, S.: "Tapping on My Network Door," *Commun. of the ACM*, vol. 43, p. 136, Oct. 2000.

BOGGS, D., MOGUL, J., and KENT, C.: "Measured Capacity of an Ethernet: Myths and Reality," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 222–234, 1988.

BORISOV, N., GOLDBERG, I., and WAGNER, D.: "Intercepting Mobile Communications: The Insecurity of 802.11," *Seventh Int'l Conf. on Mobile Computing and Networking*, ACM, pp. 180–188, 2001.

BRADEN, R.: "Requirements for Internet Hosts—Communication Layers," RFC 1122, Oct. 1989.

BRADEN, R., BORMAN, D., and PARTRIDGE, C.: "Computing the Internet Checksum," RFC 1071, Sept. 1988.

BRANDENBURG, K.: "MP3 and AAC Explained," *Proc. 17th Intl. Conf.: High-Quality Audio Coding*, Audio Engineering Society, pp. 99–110, Aug. 1999.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C., MALER, E., YERGEAU, F., and COWAN, J.: "Extensible Markup Language (XML) 1.1 (Second Edition)," W3C Recommendation, Sept. 2006.

BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., and SHENKER, S.: "Web Caching and Zipflike Distributions: Evidence and Implications," *Proc. INFOCOM Conf.*, IEEE, pp. 126–134, 1999.

BURLEIGH, S., HOOKE, A., TORGERSON, L., FALL, K., CERF, V., DURST, B., SCOTT, K., and WEISS, H.: "Delay-Tolerant Networking: An Approach to Interplanetary Internet," *IEEE Commun. Magazine*, vol. 41, pp. 128–136, June 2003.

BURNETT, S., and PAINE, S.: *RSA Security's Official Guide to Cryptography*, Berkeley, CA: Osborne/ McGraw-Hill, 2001.

BUSH, V.: "As We May Think," *Atlantic Monthly*, vol. 176, pp. 101–108, July 1945.

CAPETANAKIS, J.I.: "Tree Algorithms for Packet Broadcast Channels," *IEEE Trans. on Information Theory*, vol. IT–5, pp. 505–515, Sept. 1979.

CASTAGNOLI, G., BRAUER, S., and HERRMANN, M.: "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits," *IEEE Trans. on Commun.*, vol. 41, pp. 883–892, June 1993.

CERF, V., and KAHN, R.: "A Protocol for Packet Network Interconnection," *IEEE Trans. on Commun.*, vol. COM–2, pp. 637–648, May 1974.

CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., and GRUBER, R.: "Bigtable: A Distributed Storage System for Structured Data," *Proc. OSDI 2006 Symp.*, USENIX, pp. 15–29, 2006.

CHASE, J.S., GALLATIN, A.J., and YOCUM, K.G.: "End System Optimizations for High- Speed TCP," *IEEE Commun. Magazine*, vol. 39, pp. 68–75, Apr. 2001.

CHEN, S., and NAHRSTEDT, K.: "An Overview of QoS Routing for Next-Generation Net- works," *IEEE Network Magazine*, vol. 12, pp. 64–69, Nov./Dec. 1998.

CHIU, D., and JAIN, R.: "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Comput. Netw. ISDN Syst.*, vol. 17, pp. 1–4, June 1989.

CISCO: "Cisco Visual Networking Index: Forecast and Methodology, 2009–2014," Cisco Systems Inc., June 2010.

CLARK, D.D.: "The Design Philosophy of the DARPA Internet Protocols," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 106–114, 1988.

CLARK, D.D.: "Window and Acknowledgement Strategy in TCP," RFC 813, July 1982.

CLARK, D.D., JACOBSON, V., ROMKEY, J., and SALWEN, H.: "An Analysis of TCP Processing Overhead," *IEEE Commun. Magazine*, vol. 27, pp. 23–29, June 1989.

CLARK, D.D., SHENKER, S., and ZHANG, L.: "Supporting Real-Time Applications in an Integrated Services Packet Network," *Proc. SIGCOMM '92 Conf.*, ACM, pp. 14–26,1992.

CLARKE, A.C.: "Extra-Terrestrial Relays," *Wireless World*, 1945.

CLARKE, I., MILLER, S.G., HONG, T.W., SANDBERG, O., and WILEY, B.: "Protecting Free Expression Online with Freenet," *IEEE Internet Computing*, vol. 6, pp. 40–49, Jan.–Feb. 2002.

COHEN, B.: "Incentives Build Robustness in BitTorrent," *Proc. First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

COMER, D.E.: *The Internet Book*, 4th ed., Englewood Cliffs, NJ: Prentice Hall, 2007.

COMER, D.E.: *Internetworking with TCP/IP*, vol. 1, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2005.

CRAVER, S.A., WU, M., LIU, B., STUBBLEFIELD, A., SWARTZLANDER, B., WALLACH, D.W., DEAN, D., and FELTEN, E.W.: "Reading Between the Lines: Lessons from the SDMI Challenge," *Proc. 10th USENIX Security Symp.*, USENIX, 2001.

CROVELLA, M., and KRISHNAMURTHY, B.: *Internet Measurement*, New York: John Wiley & Sons, 2006.

DAEMEN, J., and RIJMEN, V.: *The Design of Rijndael*, Berlin: Springer-Verlag, 2002.

DALAL, Y., and METCLFE, R.: "Reverse Path Forwarding of Broadcast Packets," *Commun. of the ACM*, vol. 21, pp. 1040–1048, Dec. 1978.

DAVIE, B., and FARREL, A.: *MPLS: Next Steps*, San Francisco: Morgan Kaufmann, 2008.

DAVIE, B., and REKHTER, Y.: *MPLS Technology and Applications*, San Francisco: Morgan Kaufmann, 2000.

DAVIES, J.: *Understanding IPv6*, 2nd ed., Redmond, WA: Microsoft Press, 2008.

DAY, J.D.: "The (Un)Revised OSI Reference Model," *Computer Commun. Rev.*, vol. 25, pp. 39–55, Oct. 1995.

DAY, J.D., and ZIMMERMANN, H.: "The OSI Reference Model," *Proc. of the IEEE*, vol.71, pp. 1334–1340, Dec. 1983.

DECANDIA, G., HASTORIN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., and VOGELS, W.: "Dynamo: Amazon's Highly Available Key-value Store," *Proc. 19th Symp. on Operating Systems Prin.*, ACM, pp. 205–220, Dec. 2007.

DEERING, S.E.: "SIP: Simple Internet Protocol," *IEEE Network Magazine*, vol. 7, pp.16–28, May/June 1993.

DEERING, S., and CHERITON, D.: "Multicast Routing in Datagram Networks and Ex- tended LANs," *ACM Trans. on Computer Systems*, vol. 8, pp. 85–110, May 1990.

DEMERS, A., KESHAV, S., and SHENKER, S.: "Analysis and Simulation of a Fair Queueing Algorithm,"

Internetwork: Research and Experience, vol. 1, pp. 3–26, Sept. 1990.

DENNING, **D.E., and SACCO, G.M.:** "Timestamps in Key Distribution Protocols," *Commun. of the ACM*, vol. 24, pp. 533–536, Aug. 1981.

DEVARAPALLI, **V., WAKIKAWA, R., PETRESCU, A., and THUBERT, P.:** "Network Mobility (NEMO) Basic Support Protocol," RFC 3963, Jan. 2005.

DIFFIE, **W., and HELLMAN, M.E.:** "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *IEEE Computer*, vol. 10, pp. 74–84, June 1977.

DIFFIE, **W., and HELLMAN, M.E.:** "New Directions in Cryptography," *IEEE Trans. on Information Theory*, vol. IT–2, pp. 644–654, Nov. 1976.

DIJKSTRA, **E.W.:** "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, vol. 1, pp. 269–271, Oct. 1959.

DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., and WHEIL, B.: "Globally Distributed Content Delivery," *IEEE Internet Computing*, vol. 6, pp.50–58, 2002.

DINGLEDINE, **R., MATHEWSON, N., SYVERSON, P.:** "Tor: The Second-Generation Onion Router," *Proc. 13th USENIX Security Symp.*, USENIX, pp. 303–320, Aug.2004.

DONAHOO, **M., and CALVERT, K.:** *TCP/IP Sockets in C*, 2nd ed., San Francisco: Morgan Kaufmann, 2009.

DONAHOO, **M., and CALVERT, K.:** *TCP/IP Sockets in Java*, 2nd ed., San Francisco: Morgan Kaufmann, 2008.

DONALDSON, **G., and JONES, D.:** "Cable Television Broadband Network Architectures," *IEEE Commun. Magazine*, vol. 39, pp. 122–126, June 2001.

DORFMAN, **R.:** "Detection of Defective Members of a Large Population," *Annals Math. Statistics*, vol. 14, pp. 436–440, 1943.

DUTCHER, **B.:** *The NAT Handbook*, New York: John Wiley & Sons, 2001.

DUTTA-**ROY, A.:** "An Overview of Cable Modem Technology and Market Perspectives," *IEEE Commun. Magazine*, vol. 39, pp. 81–88, June 2001.

EDELMAN, **B., OSTROVSKY, M., and SCHWARZ, M.:** "Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords," *American Economic Review*, vol. 97, pp. 242–259, Mar. 2007.

EL GAMAL, **T.:** "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans. on Information Theory*, vol. IT–1, pp. 469–472, July 1985.

EPCGLOBAL: *EPC Radio-Frequency Identity Protocols Class– Generation– UHF RFID Protocol for Communication at 860-MHz to 960-MHz Version 1.2.0*, Brussels: EPC global Inc., Oct. 2008.

FALL, **K.:** "A Delay-Tolerant Network Architecture for Challenged Internets," *Proc. SIGCOMM 2003 Conf.*, ACM, pp. 27–34, Aug. 2003.

FALOUTSOS, **M., FALOUTSOS, P., and FALOUTSOS, C.:** "On Power-Law Relationships of the Internet Topology," *Proc. SIGCOMM '99 Conf.*, ACM, pp. 251–262, 1999.

FARRELL, **S., and CAHILL, V.:** *Delay- and Disruption-Tolerant Networking*, London: Artech House, 2007.

FELLOWS, **D., and JONES, D.:** "DOCSIS Cable Modem Technology," *IEEE Commun. Magazine*, vol. 39, pp. 202–209, Mar. 2001.

FENNER, **B., HANDLEY, M., HOLBROOK, H., and KOUVELAS, I.:** "Protocol Independent Multicast-Sparse Mode (PIM-SM)," RFC 4601, Aug. 2006.

FERGUSON, **N., SCHNEIER, B., and KOHNO, T.:** *Cryptography Engineering: Design Principles and Practical Applications*, New York: John Wiley & Sons, 2010.

FLANAGAN, **D.:** *JavaScript: The Definitive Guide*, 6th ed., Sebastopol, CA: O'Reilly, 2010.

FLETCHER, **J.:** "An Arithmetic Checksum for Serial Transmissions," *IEEE Trans. on Commun.*, vol. COM–0, pp. 247–252, Jan. 1982.

FLOYD, S., HANDLEY, **M., PADHYE, J., and WIDMER, J.:** "Equation-Based Congestion Control for Unicast Applications," *Proc. SIGCOMM 2000 Conf.*, ACM, pp. 43–56, Aug. 2000.

FLOYD, **S., and JACOBSON, V.:** "Random Early Detection for Congestion Avoidance," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 397–413, Aug. 1993.

FLUHRER, S., MANTIN, **I., and SHAMIR, A.:** "Weakness in the Key Scheduling Algorithm of RC4," *Proc. Eighth Ann. Workshop on Selected Areas in Cryptography*, Berlin: Springer-Verlag LNCS 2259, pp. 1–24, 2001.

FORD, **B.:** "Structured Streams: A New Transport Abstraction," *Proc. SIGCOMM 2007 Conf.*, ACM, pp. 361–372, 2007.

FORD, L.R., Jr., and FULKERSON, D.R.: *Flows in Networks*, Princeton, NJ: Princeton University Press, 1962.

FORD, W., and BAUM, M.S.: *Secure Electronic Commerce*, Upper Saddle River, NJ: Prentice Hall, 2000.

FORNEY, **G.D.:** "The Viterbi Algorithm," *Proc. of the IEEE*, vol. 61, pp. 268–278, Mar. 1973.

FOULI, **K., and MALER, M.:** "The Road to Carrier-Grade Ethernet," *IEEE Commun Magazine*, vol. 47, pp. S30–S38, Mar. 2009.

FOX, A., GRIBBLE, S., BREWER, E., and AMIR, E.: "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *SIGOPS Oper. Syst. Rev.*, vol. 30, pp. 160–170, Dec. 1996.

FRANCIS, **P.:** "A Near-Term Architecture for Deploying Pip," *IEEE Network Magazine*, vol. 7, pp. 30–37, May/June 1993.

FRASER, **A.G.:** "Towards a Universal Data Transport System," *IEEE J. on Selected Areas in Commun.*, vol. 5, pp. 803–816, Nov. 1983.

FRIDRICH, **J.:** *Steganography in Digital Media: Principles, Algorithms, and Applications*, Cambridge: Cambridge University Press, 2009.

FULLER, **V., and LI, T.:** "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," RFC 4632, Aug. 2006.

GALLAGHER, **R.G.:** "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. on Commun.*, vol. COM–5, pp. 73–85, Jan. 1977.

GALLAGHER, **R.G.:** "Low-Density Parity Check Codes," *IRE Trans. on Information Theory*, vol. 8, pp. 21–28, Jan. 1962.

GARFINKEL, **S., with SPAFFORD, G.:** *Web Security, Privacy, and Commerce*, Sebastopol, CA: O'Reilly, 2002.

GAST, M.: *802.11 Wireless Networks: The Definitive Guide*, 2nd ed., Sebastopol, CA: O'Reilly, 2005.

GERSHENFELD, **N., and KRIKORIAN, R., and COHEN, D.:** "The Internet of Things," *Scientific American*, vol. 291, pp. 76–81, Oct. 2004.

GILDER, **G.:** "Metcalfe's Law and Legacy," *Forbes ASAP*, Sepy. 13, 1993.

GOODE, **B.:** "Voice over Internet Protocol," *Proc. of the IEEE*, vol. 90, pp. 1495–1517, Sept. 2002.

GORALSKI, **W.J.:** *SONET*, 2nd ed., New York: McGraw-Hill, 2002.

GRAYSON, **M., SHATZKAMER, K., and WAINNER, S.:** *IP Design for Mobile Networks*, Indianapolis, IN: Cisco Press, 2009.

GROBE, **K., and ELBERS, J.:** "PON in Adolescence: From TDMA to WDM-PON," *IEEE Commun.*

Magazine, vol. 46, pp. 26–34, Jan. 2008.

GROSS, **G., KAYCEE, M., LIN, A., MALIS, A., and STEPHENS, J.:** "The PPP Over AAL5," RFC 2364, July 1998.

HA, S., RHEE, **I., and LISONG, X.:** "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, June 2008.

HAFNER, **K., and LYON, M.:** *Where Wizards Stay Up Late*, New York: Simon & Schuster, 1998.

HALPERIN, **D., HEYDT-BENJAMIN, T., RANSFORD, B., CLARK, S., DEFEND, B., MOR- GAN, W., FU, K., KOHNO, T., and** MAISEL, **W.:** "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses," *IEEE Symp. On Security and Privacy*, pp. 129–142, May 2008.

HALPERIN, **D., HU, W., SHETH, A., and WETHERALL, D.:** "802.11 with Multiple Antennas for Dummies," *Computer Commun. Rev.*, vol. 40, pp. 19–25, Jan. 2010.

HAMMING, **R.W.:** "Error Detecting and Error Correcting Codes," *Bell System Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

HARTE, **L., KELLOGG, S., DREHER, R., and SCHAFFNIT, T.:** *The Comprehensive Guide to Wireless Technology*, Fuquay-Varina, NC: APDG Publishing, 2000.

HAWLEY, **G.T.:** "Historical Perspectives on the U.S. Telephone Loop," *IEEE Commun. Magazine*, vol. 29, pp. 24–28, Mar. 1991.

HECHT, **J.:** *Understanding Fiber Optics*, Upper Saddle River, NJ: Prentice Hall, 2005.

HELD, **G.:** *A Practical Guide to Content Delivery Networks*, 2nd ed., Boca Raton, FL: CRC Press, 2010.

HEUSSE, **M., ROUSSEAU, F., BERGER-SABBATEL, G., DUDA, A.:** "Performance Anomaly of 802.11b," *Proc. INFOCOM Conf.*, IEEE, pp. 836–843, 2003.

HIERTZ, **G., DENTENEER, D., STIBOR, L., ZANG, Y., COSTA, X., and WALKE, B.:** "The IEEE 802.11 Universe," *IEEE Commun. Magazine*, vol. 48, pp. 62–70, Jan. 2010.

HOE, **J.:** "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *Proc. SIGCOMM '96 Conf.*, ACM, pp. 270–280, 1996.

HU, Y., and LI, V.O.K.: "Satellite-Based Internet: A Tutorial," *IEEE Commun. Magazine*, vol. 30, pp. 154–162, Mar. 2001.

HUITEMA, **C.:** *Routing in the Internet*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1999.

HULL, B., BYCHKOVSKY, V., CHEN, K., GORACZKO, M., MIU, A., SHIH, E., ZHANG, Y., BALAKRISHNAN, H., and MADDEN, S.: "CarTel: A Distributed Mobile Sensor Computing System," *Proc. Sensys 2006 Conf.*, ACM, pp. 125–138, Nov. 2006.

HUNTER, **D., RAFTER, J., FAWCETT, J., VAN DER LIST, E., AYERS, D., DUCKETT, J., WATT, A., and MCKINNON, L.:** *Beginning XML*, 4th ed., New Jersey: Wrox, 2007.

IRMER, **T.:** "Shaping Future Telecommunications: The Challenge of Global Standardization," *IEEE Commun. Magazine*, vol. 32, pp. 20–28, Jan. 1994.

ITU (INTERNATIONAL **TELECOMMUNICATION UNION):** *ITU Internet Reports 2005: The Internet of Things*, Geneva: ITU, Nov. 2005.

ITU (INTERNATIONAL **TELECOMMUNICATION UNION):** *Measuring the Information Society: The ICT Development Index*, Geneva: ITU, Mar. 2009.

JACOBSON, **V.:** "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, Feb. 1990.

JACOBSON, **V.:** "Congestion Avoidance and Control," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 314–329, 1988.

JAIN, **R., and ROUTHIER, S.:** "Packet Trains—Measurements and a New Model for Computer Network Traffic," *IEEE J. on Selected Areas in Commun.*, vol. 6, pp. 986–995, Sept. 1986.

JAKOBSSON, **M., and WETZEL, S.:** "Security Weaknesses in Bluetooth," *Topics in Cryptology: CT-RSA 2001*, Berlin: Springer-Verlag LNCS 2020, pp. 176–191, 2001.

JOEL, **A.:** "Telecommunications and the IEEE Communications Society," *IEEE Commun. Magazine*, 50th Anniversary Issue, pp. 6–14 and 162–167, May 2002.

JOHNSON, **D., PERKINS, C., and ARKKO, J.:** "Mobility Support in IPv6," RFC 3775, June 2004.

JOHNSON, **D.B., MALTZ, D., and BROCH, J.:** "DSR: The Dynamic Source Routing Proto- col for Multi-Hop Wireless Ad Hoc Networks," *Ad Hoc Networking*, Boston: Addison-Wesley, pp. 139–172, 2001.

JUANG, **P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L., and RUBENSTEIN, D.:** "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 96–107, Oct. 2002.

KAHN, **D.:** *The Code breakers*, 2nd ed., New York: Macmillan, 1995.

KAMOUN, **F., and KLEINROCK, L.:** "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks," *Computer Networks*, vol. 3, pp. 337–353, Nov. 1979.

KARN, **P.:** "MACA—A New Channel Access Protocol for Packet Radio," *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.*, pp. 134–140, 1990.

KARN, **P., and PARTRIDGE, C.:** "Improving Round-Trip Estimates in Reliable Transport Protocols," *Proc. SIGCOMM '87 Conf.*, ACM, pp. 2–7, 1987.

KARP, **B., and KUNG, H.T.:** "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. MOBICOM 2000 Conf.*, ACM, pp. 243–254, 2000.

KASIM, **A.:** *Delivering Carrier Ethernet*, New York: McGraw-Hill, 2007.

KATABI, **D., HANDLEY, M., and ROHRS, C.:** "Internet Congestion Control for Future High Bandwidth-Delay Product Environments," *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 89–102, 2002.

KATZ, **D., and FORD, P.S.:** "TUBA: Replacing IP with CLNP," *IEEE Network Magazine*, vol. 7, pp. 38–47, May/June 1993.

KAUFMAN, **C., PERLMAN, R., and SPECINER, M.:** *Network Security*, 2nd ed., Engle- wood Cliffs, NJ: Prentice Hall, 2002.

KENT, **C., and MOGUL, J.:** "Fragmentation Considered Harmful," *Proc. SIGCOMM '87 Conf.*, ACM, pp. 390–401, 1987.

KERCKHOFF, **A.:** "La Cryptographie Militaire," *J. des Sciences Militaires*, vol. 9, pp. 5–38, Jan. 1883 and pp. 161–191, Feb. 1883.

KHANNA, **A., and ZINKY, J.:** "The Revised ARPANET Routing Metric," *Proc. SIGCOMM '89 Conf.*, ACM, pp. 45–56, 1989.

KIPNIS, **J.:** "Beating the System: Abuses of the Standards Adoption Process," *IEEE Commun. Magazine*, vol. 38, pp. 102–105, July 2000.

KLEINROCK, **L.:** "Power and Other Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," *Proc. Intl. Conf. on Commun.*, pp. 43.1.1–43.1.10, June 1979.

KLEINROCK, **L., and TOBAGI, F.:** "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," *Proc. Nat. Computer Conf.*, pp. 187–201, 1975.

KOHLER, **E., HANDLEY, H., and FLOYD, S.:** "Designing DCCP: Congestion Control without Reliability," *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 27–38, 2006.

KOODLI, **R., and PERKINS, C.E.:** *Mobile Inter-networking with IPv6*, New York: John Wiley & Sons, 2007.

KOOPMAN, **P.:** "32-Bit Cyclic Redundancy Codes for Internet Applications," *Proc. Intl. Conf. on Dependable Systems and Networks.*, IEEE, pp. 459–472, 2002.

KRISHNAMURTHY, **B., and REXFORD, J.:** *Web Protocols and Practice*, Boston: Addison-Wesley, 2001.

KUMAR, **S., PAAR, C., PELZL, J., PFEIFFER, G., and SCHIMMLER, M.:** "Breaking Ciphers with COPACOBANA: A Cost-Optimized Parallel Code Breaker," *Proc. 8th Cryptographic Hardware and Embedded Systems Wksp.*, IACR, pp. 101–118, Oct. 2006.

LABOVITZ, **C., AHUJA, A., BOSE, A., and JAHANIAN, F.:** "Delayed Internet Routing Convergence," *IEEE/ACM Trans. on Networking*, vol. 9, pp. 293–306, June 2001.

LAM, C.K.M., and TAN, B.C.Y.: "The Internet Is Changing the Music Industry," *Commun. of the ACM*, vol. 44, pp. 62–66, Aug. 2001.

LAOUTARIS, **N., SMARAGDAKIS, G., RODRIGUEZ, P., and SUNDARAM, R.:** "Delay Tolerant Bulk Data Transfers on the Internet," *Proc. SIGMETRICS 2009 Conf.*, ACM, pp. 229–238, June 2009.

LARMO, **A., LINDSTROM, M., MEYER, M., PELLETIER, G., TORSNER, J., and WIEMANN, H.:** "The LTE Link-Layer Design," *IEEE Commun. Magazine*, vol. 47, pp. 52–59, Apr. 2009.

LEE, J.S., and MILLER, L.E.: *CDMA Systems Engineering Handbook*, London: Artech House, 1998.

LELAND, **W., TAQQU, M., WILLINGER, W., and WILSON, D.:** "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Trans. on Networking*, vol. 2, pp. 1–15, Feb. 1994.

LEMON, **J.:** "Resisting SYN Flood DOS Attacks with a SYN Cache," *Proc. BSDCon Conf.*, USENIX, pp. 88–98, 2002.

LEVY, **S.:** "Crypto Rebels," *Wired*, pp. 54–61, May/June 1993.

LEWIS, **M.:** *Comparing, Designing, and Deploying VPNs*, Indianapolis, IN: Cisco Press, 2006.

LI, M., AGRAWAL, **D., GANESAN, D., and VENKATARAMANI, A.:** "Block-Switched Networks: A New Paradigm for Wireless Transport," *Proc. NSDI 2009 Conf.*, USENIX, pp. 423–436, 2009.

LIN, S., and **COSTELLO, D.:** *Error Control Coding*, 2nd ed., Upper Saddle River, NJ: Pearson Education, 2004.

LUBACZ, J., MAZURCZYK, W., and SZCZYPIORSKI, K.: "Vice over IP," *IEEE Spec- trum*, pp. 42–47, Feb. 2010.

MACEDONIA, **M.R.:** "Distributed File Sharing," *IEEE Computer*, vol. 33, pp. 99–101, 2000.

MADHAVAN, **J., KO, D., LOT, L., GANGPATHY, V., RASMUSSEN, A., and HALEVY, A.:** "Google's Deep Web Crawl," *Proc. VLDB 2008 Conf.*, VLDB Endowment, pp.1241–1252, 2008.

MAHAJAN, **R., RODRIG, M., WETHERALL, D., and ZAHORJAN, J.:** "Analyzing the MAC-Level Behavior of Wireless Networks in the Wild," *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 75–86, 2006.

MALIS, **A., and SIMPSON, W.:** "PPP over SONET/SDH," RFC 2615, June 1999.

MASSEY, **J.L.:** "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. on Information Theory*, vol. IT–5, pp. 122–127, Jan. 1969.

MATSUI, **M.:** "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology— Eurocrypt 1993 Proceedings*, Berlin: Springer-Verlag LNCS 765, pp. 386–397, 1994.

MAUFER, **T.A.:** *IP Fundamentals*, Upper Saddle River, NJ: Prentice Hall, 1999.

MAYMOUNKOV, **P., and MAZIERES, D.:** "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. First Intl. Wksp. on Peer-to-Peer Systems*, Berlin: Springer-Verlag LNCS 2429, pp. 53–65, 2002.

MAZIERES, **D., and KAASHOEK, M.F.:** "The Design, Implementation, and Operation of an Email Pseudonym Server," *Proc. Fifth Conf. on Computer and Commun. Security*, ACM, pp. 27–36, 1998.

MCAFEE **LABS:** *McAfee Threat Reports: First Quarter 2010*, McAfee Inc., 2010.

MENEZES, **A.J., and VANSTONE, S.A.:** "Elliptic Curve Cryptosystems and Their Implementation," *Journal of Cryptology*, vol. 6, pp. 209–224, 1993.

MERKLE, **R.C., and HELLMAN, M.:** "Hiding and Signatures in Trapdoor Knapsacks," *IEEE Trans. on Information Theory*, vol. IT–4, pp. 525–530, Sept. 1978.

METCALFE, **R.M.:** "Computer/Network Interface Design: Lessons from Arpanet and Ethernet," *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 173–179, Feb. 1993.

METCALFE, **R.M., and BOGGS, D.R.:** "Ethernet: Distributed Packet Switching for Local Computer Networks," *Commun. of the ACM*, vol. 19, pp. 395–404, July 1976.

METZ, **C:** "Interconnecting ISP Networks," *IEEE Internet Computing*, vol. 5, pp. 74–80, Mar.–Apr. 2001.

MISHRA, **P.P., KANAKIA, H., and TRIPATHI, S.:** "On Hop by Hop Rate-Based Congestion Control," *IEEE/ACM Trans. on Networking*, vol. 4, pp. 224–239, Apr. 1996.

MOGUL, **J.C.:** "IP Network Performance," in *Internet System Handbook*, D.C. Lynch and M.Y. Rose (eds.), Boston: Addison-Wesley, pp. 575–575, 1993.

MOGUL, **J., and DEERING, S.:** "Path MTU Discovery," RFC 1191, Nov. 1990.

MOGUL, **J., and MINSHALL, G.:** "Rethinking the Nagle Algorithm," *Comput. Commun. Rev.*, vol. 31, pp. 6–20, Jan. 2001.

MOY, **J.:** "Multicast Routing Extensions for OSPF," *Commun. of the ACM*, vol. 37, pp. 61–66, Aug. 1994.

MULLINS, **J.:** "Making Unbreakable Code," *IEEE Spectrum*, pp. 40–45, May 2002.

NAGLE, **J.:** "On Packet Switches with Infinite Storage," *IEEE Trans. on Commun.*, vol. COM–5, pp. 435–438, Apr. 1987.

NAGLE, **J.:** "Congestion Control in TCP/IP Internetworks," *Computer Commun. Rev.*, vol. 14, pp. 11–17, Oct. 1984.

NAUGHTON, **J.:** *A Brief History of the Future*, Woodstock, NY: Overlook Press, 2000.

NEEDHAM, **R.M., and SCHROEDER, M.D.:** "Using Encryption for Authentication in Large Networks of Computers," *Commun. of the ACM*, vol. 21, pp. 993–999, Dec. 1978.

NEEDHAM, **R.M., and SCHROEDER, M.D.:** "Authentication Revisited," *Operating Systems Rev.*, vol. 21, p. 7, Jan. 1987.

NELAKUDITI, **S., and ZHANG, Z.-L.:** "A Localized Adaptive Proportioning Approach to QoS Routing," *IEEE Commun. Magazine* vol. 40, pp. 66–71, June 2002.

NEUMAN, **C., and TS'O, T.:** "Kerberos: An Authentication Service for Computer Net- works," *IEEE Commun. Mag.*, vol. 32, pp. 33–38, Sept. 1994.

NICHOLS, **R.K., and LEKKAS, P.C.:** *Wireless Security*, New York: McGraw-Hill, 2002.

NIST: "Secure Hash Algorithm," U.S. Government Federal Information Processing Stan- dard 180, 1993.

NONNENMACHER, **J., BIERSACK, E., and TOWSLEY, D.:** "Parity-Based Loss Recovery for Reliable Multicast Transmission," *Proc. SIGCOMM '97 Conf.*, ACM, pp. 289–300, 1997.

NUCCI, **A., and PAPAGIANNAKI, D.:** *Design, Measurement and Management of Large- Scale IP Networks*, Cambridge: Cambridge University Press, 2008.

NUGENT, **R., MUNAKANA, R., CHIN, A., COELHO, R., and PUIG-SUARI, J.:** "The CubeSat: The PicoSatellite Standard for Research and Education," *Proc. SPACE 2008Conf.*, AIAA, 2008.

ORAN, **D.:** "OSI IS-IS Intra-domain Routing Protocol," RFC 1142, Feb. 1990.

OTWAY, **D., and REES, O.:** "Efficient and Timely Mutual Authentication," *Operating Systems Rev.*, pp. 8–10, Jan. 1987.

PADHYE, J., FIROIU, V., TOWSLEY, D., and KUROSE, J.: "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *Proc. SIGCOMM '98 Conf.*, ACM, pp. 303–314, 1998.

PALAIS, J.C.: *Fiber Optic Commun.*, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2004.

PARAMESWARAN, M., SUSARLA, A., and WHINSTON, A.B.: "P2P Networking: An Information-Sharing Alternative," *IEEE Computer*, vol. 34, pp. 31–38, July 2001.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-Node Case," *IEEE/ACM Trans. on Networking*, vol. 2, pp. 137–150, Apr. 1994.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 344–357, June 1993.

PARTRIDGE, C., HUGHES, J., and STONE, J.: "Performance of Checksums and CRCs over Real Data," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 68–76, 1995.

PARTRIDGE, C., MENDEZ, T., and MILLIKEN, W.: "Host Anycasting Service," RFC 1546, Nov. 1993.

PAXSON, V., and FLOYD, S.: "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. on Networking*, vol. 3, pp. 226–244, June 1995.

PERKINS, C.: "IP Mobility Support for IPv4," RFC 3344, Aug. 2002.

PERKINS, C.E.: *RTP: Audio and Video for the Internet*, Boston: Addison-Wesley, 2003.

PERKINS, C.E. (ed.): *Ad Hoc Networking*, Boston: Addison-Wesley, 2001.

PERKINS, C.E.: *Mobile IP Design Principles and Practices*, Upper Saddle River, NJ: Prentice Hall, 1998.

PERKINS, C.E., and ROYER, E.: "The Ad Hoc On-Demand Distance-Vector Protocol," in *Ad Hoc Networking*, edited by C. Perkins, Boston: Addison-Wesley, 2001.

PERLMAN, R.: *Interconnections*, 2nd ed., Boston: Addison-Wesley, 2000.

PERLMAN, R.: *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, M.I.T., 1988.

PERLMAN, R.: "An Algorithm for the Distributed Computation of a Spanning Tree in an Extended LAN," *Proc. SIGCOMM '85 Conf.*, ACM, pp. 44–53, 1985.

PERLMAN, R., and KAUFMAN, C.: "Key Exchange in IPsec," *IEEE Internet Computing*, vol. 4, pp. 50–56, Nov.–Dec. 2000.

PETERSON, W.W., and BROWN, D.T.: "Cyclic Codes for Error Detection," *Proc. IRE*, vol. 49, pp. 228–235, Jan. 1961.

PIATEK, M., KOHNO, T., and KRISHNAMURTHY, A.: "Challenges and Directions for Monitoring P2P File Sharing Networks—or Why My Printer Received a DMCA Takedown Notice," *3rd Workshop on Hot Topics in Security*, USENIX, July 2008.

PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., and VENKA- TARAMANI, V.: "Do Incentives Build Robustness in BitTorrent?," *Proc. NSDI 2007 Conf.*, USENIX, pp. 1–14, 2007.

PISCITELLO, D.M., and CHAPIN, A.L.: *Open Systems Networking: TCP/IP and OSI*, Boston: Addison-Wesley, 1993.

PIVA, A., BARTOLINI, F., and BARNI, M.: "Managing Copyrights in Open Networks," *IEEE Internet Computing*, vol. 6, pp. 18–26, May– 2002.

POSTEL, J.: "Internet Control Message Protocols," RFC 792, Sept. 1981.

RABIN, J., and MCCATHIENEVILE, C.: "Mobile Web Best Practices 1.0," W3C Recommendation, July 2008.

RAMAKRISHNAM, K.K., FLOYD, S., and BLACK, D.: "The Addition of Explicit Congestion Notification

(ECN) to IP," RFC 3168, Sept. 2001.

RAMAKRISHNAN, K.K., and JAIN, R.: "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 303–313, 1988.

RAMASWAMI, R., KUMAR, S., and SASAKI, G.: *Optical Networks: A Practical Perspective*, 3rd ed., San Francisco: Morgan Kaufmann, 2009.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., and SHENKER, S.: "A Scalable Content-Addressable Network," *Proc. SIGCOMM 2001 Conf.*, ACM, pp. 161–172, 2001.

RIEBACK, M., CRISPO, B., and TANENBAUM, A.: "Is Your Cat Infected with a Computer Virus?," *Proc. IEEE Percom*, pp. 169–179, Mar. 2006.

RIVEST, R.L.: "The MD5 Message-Digest Algorithm," RFC 1320, Apr. 1992.

RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: "On a Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Commun. of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

ROBERTS, L.G.: "Extensions of Packet Communication Technology to a Hand Held Personal Terminal," *Proc. Spring Joint Computer Conf.*, AFIPS, pp. 295–298, 1972.

ROBERTS, L.G.: "Multiple Computer Networks and Intercomputer Communication," *Proc. First Symp. on Operating Systems Prin.*, ACM, pp. 3.1–3.6, 1967.

ROSE, M.T.: *The Simple Book*, Englewood Cliffs, NJ: Prentice Hall, 1994. **ROSE, M.T.:** *The Internet Message*, Englewood Cliffs, NJ: Prentice Hall, 1993.

ROWSTRON, A., and DRUSCHEL, P.: "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Storage Utility," *Proc. 18th Int'l Conf. on Distributed Systems Platforms*, London: Springer-Verlag LNCS 2218, pp. 329–350, 2001.

RUIZ-SANCHEZ, M.A., BIERSACK, E.W., and DABBOUS, W.: "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network Magazine*, vol. 15, pp. 8–23, Mar.–Apr. 2001.

SALTZER, J.H., REED, D.P., and CLARK, D.D.: "End-to-End Arguments in System Design," *ACM Trans. on Computer Systems*, vol. 2, pp. 277–288, Nov. 1984.

SAMPLE, A., YEAGER, D., POWLEDGE, P., MAMISHEV, A., and SMITH, J.: "Design of an RFID-Based Battery-Free Programmable Sensing Platform," *IEEE Trans. on Instrumentation and Measurement*, vol. 57, pp. 2608–2615, Nov. 2008.

SAROIU, S., GUMMADI, K., and GRIBBLE, S.: "Measuring and Analyzing the Characteristics of Napster & Gnutella Hosts," *Multim. Syst.*, vol. 9,, pp. 170–184, Aug. 2003.

SCHALLER, R.: "Moore's Law: Past, Present and Future," *IEEE Spectrum*, vol. 34, pp. 52–59, June 1997.

SCHNEIER, B.: *Secrets and Lies*, New York: John Wiley & Sons, 2004.

SCHNEIER, B.: *E-Mail Security*, New York: John Wiley & Sons, 1995.

SCHNORR, C.P.: "Efficient Signature Generation for Smart Cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

SCHOLTZ, R.A.: "The Origins of Spread-Spectrum Communications," *IEEE Trans. on Commun.*, vol. COM–0, pp. 822–854, May 1982.

SCHWARTZ, M., and ABRAMSON, N.: "The AlohaNet: Surfing for Wireless Data," *IEEE Commun. Magazine*, vol. 47, pp. 21–25, Dec. 2009.

SEIFERT, R., and EDWARDS, J.: *The All-New Switch Book*, NY: John Wiley, 2008.

SENN, J.A.: "The Emergence of M-Commerce," *IEEE Computer*, vol. 33, pp. 148–150, Dec. 2000.

SERJANTOV, A.: "Anonymizing Censorship Resistant Systems," *Proc. First Int'Workshop on Peer-to-Peer Systems*, London: Springer-Verlag LNCS 2429, pp. 111–120, 2002.

SHACHAM, N., and MCKENNY, P.: "Packet Recovery in High-Speed Networks Using Coding and Buffer Management," *Proc. INFOCOM Conf.*, IEEE, pp. 124–131, June 1990.

SHAIKH, A., REXFORD, J., and SHIN, K.: "Load-Sensitive Routing of Long-Lived IP Flows," *Proc. SIGCOMM '99 Conf.*, ACM, pp. 215–226, Sept. 1999.

SHALUNOV, S., and CARLSON, R.: "Detecting Duplex Mismatch on Ethernet," *Passive and Active Network Measurement*, Berlin: Springer-Verlag LNCS 3431, pp. 3135–3148, 2005.

SHANNON, C.: "A Mathematical Theory of Communication," *Bell System Tech. J.*, vol. 27, pp. 379–423, July 1948; and pp. 623–656, Oct. 1948.

SHEPARD, S.: *SONET/SDH Demystified*, New York: McGraw-Hill, 2001.

SHREEDHAR, M., and VARGHESE, G.: "Efficient Fair Queueing Using Deficit Round Robin," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 231–243, 1995.

SIMPSON, W.: *Video Over IP*, 2nd ed., Burlington, MA: Focal Press, 2008.

SIMPSON, W.: "PPP in HDLC-like Framing," RFC 1662, July 1994b.

SIMPSON, W.: "The Point-to-Point Protocol (PPP)," RFC 1661, July 1994a.

SIU, K., and JAIN, R.: "A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic," *ACM Computer Communications Review*, vol. 25, pp. 6–20, Apr. 1995.

SKOUDIS, E., and LISTON, T.: *Counter Hack Reloaded*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2006.

SMITH, D.K., and ALEXANDER, R.C.: *Fumbling the Future*, New York: William Morrow, 1988.

SNOEREN, A.C., and BALAKRISHNAN, H.: "An End-to-End Approach to Host Mobility," *Int'l Conf. on Mobile Computing and Networking* , ACM, pp. 155–166, 2000.

SOBEL, D.L.: "Will Carnivore Devour Online Privacy," *IEEE Computer*, vol. 34, pp. 87–88, May 2001.

SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D., OSVIK, D., and DE WEGER, B.: "MD5 Considered Harmful Today," *Proc. 25th Chaos Communication Congress*, Verlag Art d'Ameublement, 2008.

SOUTHEY, R.: *The Doctors*, London: Longman, Brown, Green and Longmans, 1848. **SPURGEON, C.E.:** *Ethernet: The Definitive Guide*, Sebastopol, CA: O'Reilly, 2000.

STALLINGS, W.: *Data and Computer Communications*, 9th ed., Upper Saddle River, NJ: Pearson Education, 2010.

STARR, T., SORBARA, M., COIFFI, J., and SILVERMAN, P.: "DSL Advances," Upper Saddle River, NJ: Prentice Hall, 2003.

STEVENS, W.R.: *TCP/IP Illustrated: The Protocols*, Boston: Addison Wesley, 1994.

STINSON, D.R.: *Cryptography Theory and Practice*, 2nd ed., Boca Raton, FL: CRC Press, 2002.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M.F., and BALAKRISHNAN, H.: "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM 2001 Conf.*, ACM, pp. 149–160, 2001.

STUBBLEFIELD, A., IOANNIDIS, J., and RUBIN, A.D.: "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," *Proc. Network and Distributed Systems Security Symp.*, ISOC, pp. 1–11, 2002.

STUTTARD, D., and PINTO, M.: *The Web Application Hacker's Handbook*, New York: John Wiley & Sons, 2007.

SU, S.: *The UMTS Air Interface in RF Engineering*, New York: McGraw-Hill, 2007.

SULLIVAN, G., and WIEGAND, T.: "Tree Algorithms for Packet Broadcast Channels," *Proc. of the IEEE*, vol. 93, pp. 18–31, Jan. 2005.

SUNSHINE, C.A., and DALAL, Y.K.: "Connection Management in Transport Protocols," *Computer Networks*, vol. 2, pp. 454–473, 1978.

TAN, K., SONG, J., ZHANG, Q., and SRIDHARN, M.: "A Compound TCP Approach for High-Speed and Long Distance Networks," *Proc. INFOCOM Conf.*, IEEE, pp. 1–12, 2006.

TANENBAUM, A.S.: *Modern Operating Systems*, 3rd ed., Upper Saddle River, NJ: Prentice Hall, 2007.

TANENBAUM, A.S., and VAN STEEN, M.: *Distributed Systems: Principles and Paradigms*, Upper Saddle River, NJ: Prentice Hall, 2007.

TOMLINSON, R.S.: "Selecting Sequence Numbers," *Proc. SIGCOMM/SIGOPS Interprocess Commun. Workshop*, ACM, pp. 11–23, 1975.

TUCHMAN, W.: "Hellman Presents No Shortcut Solutions to DES," *IEEE Spectrum*, vol. 16, pp. 40–41, July 1979.

TURNER, J.S.: "New Directions in Communications (or Which Way to the Information Age)," *IEEE Commun. Magazine*, vol. 24, pp. 8–15, Oct. 1986.

UNGERBOECK, G.: "Trellis-Coded Modulation with Redundant Signal Sets Part I: Introduction," *IEEE Commun. Magazine*, vol. 25, pp. 5–11, Feb. 1987.

VALADE, J.: *PHP & MySQL for Dummies*, 5th ed., New York: John Wiley & Sons, 2009.

VARGHESE, G.: *Network Algorithmics*, San Francisco: Morgan Kaufmann, 2004.

VARGHESE, G., and LAUCK, T.: "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility," *Proc. 11th Symp. on Operating Systems Prin.*, ACM, pp. 25–38, 1987.

VERIZON BUSINESS: *2009 Data Breach Investigations Report*, Verizon, 2009.

VITERBI, A.: *CDMA: Principles of Spread Spectrum Communication*, Englewood Cliffs, NJ: Prentice Hall, 1995.

VON AHN, L., BLUM, B., and LANGFORD, J.: "Telling Humans and Computers Apart Automatically," *Commun. of the ACM*, vol. 47, pp. 56–60, Feb. 2004.

WAITZMAN, D., PARTRIDGE, C., and DEERING, S.: "Distance Vector Multicast Routing Protocol," RFC 1075, Nov. 1988.

WALDMAN, M., RUBIN, A.D., and CRANOR, L.F.: "Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System," *Proc. Ninth USENIX Security Symp.*, USENIX, pp. 59–72, 2000.

WANG, Z., and CROWCROFT, J.: "SEAL Detects Cell Misordering," *IEEE Network Magazine*, vol. 6, pp. 8–9, July 1992.

WANT, R.: *RFID Explained*, San Rafael, CA: Morgan Claypool, 2006.

WARNEKE, B., LAST, M., LIEBOWITZ, B., and PISTER, K.S.J.: "Smart Dust: Communicating with a Cubic Millimeter **Computer**," *IEEE Computer*, vol. 34, pp. 44–51, Jan. 2001.

WAYNER, P.: *Disappearing Cryptography: Information Hiding, Steganography, and Watermarking*, 3rd ed., San Francisco: Morgan Kaufmann, 2008.

WEI, D., CHENG, J., LOW, S., and HEGDE, S.: "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Trans. on Networking*, vol. 14, pp. 1246–1259, Dec. 2006.

WEISER, M.: "The Computer for the Twenty-First Century," *Scientific American*, vol. 265, pp. 94–104, Sept. 1991.

WELBOURNE, E., BATTLE, L., COLE, G., GOULD, K., RECTOR, K., RAYMER, S., BALAZINSKA, M., and BORRIELLO, G.: "Building the Internet of Things Using RFID," *IEEE Internet Computing*, vol.

13, pp. 48–55, May 2009.

WITTENBURG, N.: *Understanding Voice Over IP Technology*, Clifton Park, NY: Delmar Cengage Learning, 2009.

WOLMAN, A., VOELKER, G., SHARMA, N., CARDWELL, N., KARLIN, A., and LEVY, H.: "On the Scale and Performance of Cooperative Web Proxy Caching," *Proc. 17th Symp. on Operating Systems Prin.*, ACM, pp. 16–31, 1999.

WOOD, L., IVANCIC, W., EDDY, W., STEWART, D., NORTHAM, J., JACKSON, C., and DA SILVA CURIEL, A.: "Use of the Delay-Tolerant Networking Bundle Protocol from Space," *Proc. 59th Int'l Astronautical Congress*, Int'l Astronautical Federation, pp. 3123–3133, 2008.

WU, T.:" Network Neutrality, Broadband Discrimination," *Journal on Telecom. and High-Tech. Law*, vol. 2, pp. 141–179, 2003.

WYLIE, J., BIGRIGG, M.W., STRUNK, J.D., GANGER, G.R., KILICCOTE, H., and KHOSLA, P.K.: "Survivable Information Storage Systems," *IEEE Computer*, vol. 33, pp. 61–68, Aug. 2000.

YU, T., HARTMAN, S., and RAEBURN, K.: "The Perils of Unauthenticated Encryption: Kerberos Version 4," *Proc. NDSS Symposium*, Internet Society, Feb. 2004.

YUVAL, G.: "How to Swindle Rabin," *Cryptologia*, vol. 3, pp. 187–190, July 1979.

ZACKS, M.: "Antiterrorist Legislation Expands Electronic Snooping," *IEEE Internet Computing*, vol. 5, pp. 8–9, Nov.–Dec. 2001.

ZHANG, Y., BRESLAU, L., PAXSON, V., and SHENKER, S.: "On the Characteristics and Origins of Internet Flow Rates," *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 309–322, 2002.

ZHAO, B., LING, H., STRIBLING, J., RHEA, S., JOSEPH, A., and KUBIATOWICZ, J.: "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. on Selected Areas in Commun.*, vol. 22, pp. 41–53, Jan. 2004.

ZIMMERMANN, P.R.: *The Official PGP User's Guide*, Cambridge, MA: M.I.T. Press, 1995a.

ZIMMERMANN, P.R.: *PGP: Source Code and Internals*, Cambridge, MA: M.I.T. Press, 1995b.

ZIPF, G.K.: *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Boston: Addison-Wesley, 1949.

ZIV, J., and LEMPEL, Z.: "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT–3, pp. 337–343, May 1977.

索　引

数字

A

B

C

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z