计 算 机 科 学 丛 书

Education

计算机网络教程

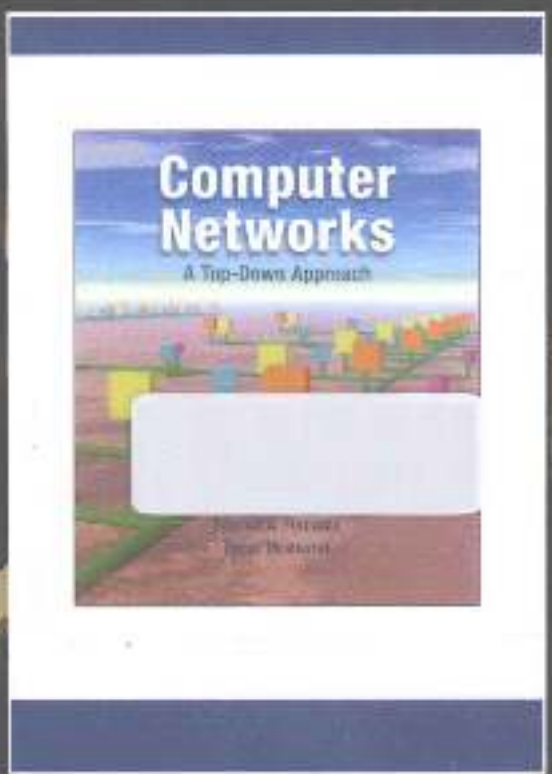
自顶向下方法

（美）Behrouz A. Forouzan Firouz Mosharraf 著

张建忠 靳军 林安华 周立斌 译

Computer Networks

A Top-Down Approach



机械工业出版社
China Machine Press

计算机网络教程 自顶向下方法

Computer Networks A Top-Down Approach

本书是计算机领域知名作者Forouzan按照目前计算机网络教学比较流行的自顶向下方法编写的一部重要教材。它延续了作者一贯的写作风格，以通俗易懂的方式全面阐述了计算机网络原理及其应用，并介绍了一些目前计算机网络发展的新技术。此外，每章都配有丰富的习题集（包括测试题、练习题、思考题），部分章节还包含模拟实验和编程作业，有助于读者巩固所学知识，提高动手实践能力。

本书特色

- 协议分层：本书利用Internet协议分层和TCP/IP协议栈讲授网络原理，强调各层网络理论之间的相互关系。
- 自顶向下：从应用层开始，尽早让读者理解网络设备如何工作，然后讨论其他各层，最后介绍物理层。
- 形象直观：采用图文并茂的方法描述技术性很强的问题，较少涉及复杂的数学公式，便于读者理解相关概念。
- 举例和应用：以丰富的实例阐明相关概念，并添加了一些现实中的应用，激发读者的学习热情。

作者简介

Behrouz A. Forouzan 毕业于加州大学艾尔温分校，现在是迈安那大学教授，从事计算机信息系统专业的课程设置。此外，他还是多家公司的系统并发咨询顾问。除本书外，Forouzan还著有多部成功的计算机科学方面的书，包括《Data Communications and Networking》、《Foundations of Computer Science》、《TCP/IP Protocol Suite》等。



影印版
书号：978-7-111-37490-5
定价：79.00元

上架指导：计算机/计算机网络

ISBN 978-7-111-40088-2

9 787111 400882 >

定价：99.00元

Mc Graw Hill Education

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604

读者信箱：hzjsj@hzbook.com
华章网站：www.hzbook.com
网上购书：www.china-pub.com

计 算 机 科 学 丛 书

计算机网络教程

自顶向下方法

[美] Behrouz A. Forouzan Firouz Mosharraf 著
张建忠 靳军 林安华 周立斌 译

Computer Networks

A Top-Down Approach



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机网络教程：自顶向下方法／（美）福罗赞（Forouzan, B. A.），（美）莫沙拉夫（Mosharraf, F.）著；张建忠等译. —北京：机械工业出版社，2012.10
（计算机科学丛书）
书名原文：Computer Networks: A Top-Down Approach

ISBN 978-7-111-40088-2

Ⅰ. 计… Ⅱ. ①佛… ②莫… ③张… Ⅲ. 计算机网络—教材 Ⅳ. TP393

中国版本图书馆 CIP 数据核字（2012）第 245205 号

版权所有，侵权必究
封底无防伪标均为盗版
本书法律顾问 北京市展达律师事务所

本书版权登记号：图字：01-2011-5060

本书作者 Forouzan 是计算机教育领域的知名专家，他在这本经典著作中，利用 Internet 协议分层和 TCP/IP 协议族，采用自顶向下的方法，首先说明应用层协议是怎样交换信息的，再解释消息是怎样分解成比特和编号并通过 Internet 传输的，以直观易懂的方式讲述了计算机网络原理，帮助学生从系统上理解网络的基础知识，特别是 Internet 的协议。

本书图文并茂，实例丰富，并配有大量的习题集（包括测试题、练习题、思考题）以及模拟实验和编程作业，适合作为本科生、研究生的计算机网络教材，同时也适合计算机网络研究和专业人员阅读。

Behrouz A. Forouzan, Firouz Mosharraf: Computer Networks: A Top-Down Approach (ISBN 978-0-07352326-2).

Copyright © 2011 by The McGraw-Hill Companies, Inc.

All Rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation edition is jointly published by McGraw-Hill Education (Asia) and China Machine Press. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2013 by McGraw-Hill Education (Asia), a division of McGraw-Hill Asian Holdings (Singapore) Pte.Ltd. And China Machine Press.

版权所有，未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔（亚洲）教育出版公司和机械工业出版社合作出版。此版本经授权仅限在中华人民共和国境内（不包括香港特别行政区、澳门特别行政区和台湾）销售。

版权© 2013 由麦格劳-希尔（亚洲）教育出版公司与机械工业出版社所有。

本书封面贴有 McGraw-Hill 公司防伪标签，无标签者不得销售。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：刘立卿
北京瑞德印刷有限公司印刷
2013 年 1 月第 1 版第 1 次印刷
185mm×260mm · 39 印张
标准书号：ISBN 978-7-111-40088-2
定 价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅奠定了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必走之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎各位老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章教育
华章科技图书出版中心

译 者 序

Computer Networks: A Top-Down Approach

近年来，计算机网络技术已经应用到各个领域。人们的学习、工作和生活已经渐渐离不开计算机网络，因此需要了解和掌握计算机网络技术的群体在不断扩大。不论是计算机及相关专业的学生，还是从事相关工作的工程技术人员，都需要较深入地了解计算机网络的基本原理和应用形式。本书作为一本深入浅出且广泛适用的计算机网络教材，能够帮助读者理解和掌握复杂的网络知识。

Behrouz A. Forouzan 是计算机教育领域的知名作家，他出版了《Data Communications and Networking》、《TCP/IP Protocol Suite》、《Cryptography & Network Security》、《Foundations of Computer Science》等多部畅销教材，涉猎范围包括计算机科学领域的多个方面。他与时俱进，作品紧跟计算机技术和计算机教育的步伐。《Computer Network：A Top-Down Approach》是 Forouzan 按照目前计算机网络教学比较流行的自顶向下方法编写的一部重要教材，与《Data Communications and Networking》相比，本书在内容上增加了一些目前计算机网络发展的新技术，编排上采用自底向下的结构，这样既适合学生学习，又适合教师教授。

暑假前夕，出版社的同志和我们联系，希望我们能够翻译这本书。由于要求的翻译周期很短，所以一开始我们非常犹豫。但是，作为教师，我们深知优秀教材在“教”与“学”中的作用。在经过认真思考之后，我们最终欣然接受了这项具有挑战性的任务。接下来的四个月，翻译工作花费了我们大量的时间和精力。

本书共分为 11 章，第 1 章和第 10 章由张建忠翻译，第 2 章、第 3 章和第 4 章由徐敬东翻译，第 5 章、第 6 章和第 11 章由林生佳翻译，第 7 章、第 8 章和第 9 章由周立柱翻译。张建忠负责最后统稿。受译者水平和翻译周期所限，译稿中可能存在不当或错误之处，敬请广大读者批评指正。

译者

2012 年 8 月于南开园

当今社会，网络与互联网技术迅猛发展，其发证就是每年各种新型社交网络应用的不断涌现。人们每天使用 Internet 的频率达到最高，他们利用 Internet 进行科学研究、网络购物、机票预订、查看新闻与天气状况……

在这个面向 Internet 的社会中，需要培养和训练专业技术人员对 Internet、部分 Internet 或者连接到 Internet 的内部网络进行运营和管理。本书的目标是帮助学生总体上理解网络的基本知识，特别是 Internet 使用的协议。

本书特色

本书的主要目标是讲授网络原理，为了讲授这些原理，本书采用了以下方法。

协议分层

本书利用 Internet 协议分层和 TCP/IP 协议簇讲授网络原理。虽然有些网络理论在某些层次上可能有些重复，但每层都会有其特别强调的方面。这些理论在不同层次重复出现，使我们能够更好地理解彼此之间的关系，因此利用协议分层方法讲授网络理论是有益的。例如，虽然寻址（addressing）在 TCP/IP 的 4 个层次中都会遇到，但是各层使用了不同的地址格式以实现各自不同的目标。另外，每层中的寻址范围也有不同。另一个例子是成帧与分组（framing and packetizing），这些内容在几层中也会重复出现，但是各层涉及的程序不同。

自顶向下方法

尽管本书的作者之一曾经编写过几本与网络和 Internet 相关的书籍（《Data Communication and Networking》、《TCP/IP Protocol Suite》、《Cryptography and Network Security》、《Local Area Networks》），但是本书讲授网络的方法不同。它采用自顶向下的方法。

虽然 TCP/IP 协议每一层建立在它下层提供的服务之上，但是学习每一层的知识有两种方法——自底向上或自顶向下。自底向上方法中，我们在学习应用层怎样利用比特传送消息之前，学习比特和信号怎样在物理层移动。自顶向下方法中，我们在学习消息怎样被分解成比特和信号、怎样实际地通过 Internet 传送之前，学习应用层协议怎样交换信息。在本书中，我们采用了自顶向下方法。

面向的读者

本书面向的读者为学术和专业技术人员。感兴趣的专业技术人员也可用本书作为自学指导书。作为教材，它可以用于一个学期或半个学期的课程，适用于大学本科最后一学年或研究生第一年的学习。虽然章节末尾的思考题需要一些概率知识，但是教材的学习只需要大学一年级讲授的基本数学知识。

本书的组织

本书包括 11 章和 5 个附录。

- 第 1 章　概论
- 第 2 章　应用层
- 第 3 章　传输层
- 第 4 章　网络层
- 第 5 章　数据链路层：有线网络
- 第 6 章　无线网络和移动 IP
- 第 7 章　物理层和传输介质
- 第 8 章　多媒体和服务质量
- 第 9 章　网络管理
- 第 10 章　网络安全
- 第 11 章　Java Socket 编程
- 附录　附录 A 到附录 E

写作方法

本书采用的几种写作方法使学生能够很容易地理解计算机网络的基础知识，特别是 Internet 的相关知识。

形象直观

本书采用图文并茂的方式描述技术性很强的问题，而没有采用复杂的数学公式。670 多幅插图与文字讲解，使内容更加直观易懂。在解释网络概念时，插图的作用尤其重要。对于很多学生来说，通过插图理解这些概念比通过文字更容易。

举例和应用

在合适的位置我们加入了一些例子，用于说明书中介绍的相关概念。同时，我们也在每章中添加了一些现实中的应用，用于激励学生学习。

章末资料

每章后包含的相关资料如下：

小结　每章末尾都包含覆盖本章内容的小结。小结将本章的重点内容关联起来，一目了然。

推荐读物　这一部分列出了与本章内容相关的主要参考文献。利用这个参考文献列表，可以在书末尾的“参考文献”部分快速找到相应文献。

习题集

每章都设计有习题集，用于巩固重要的概念，同时鼓励学生应用这些概念。习题集包括 3 部分：测试题、练习题和思考题。

测试题　测试题放置在本书的网站中，用于快速检查概念的掌握情况。学生可以通过完成这些测试题查看自己对内容的理解程度，网站可立即给出测试结果。

练习题　这部分包含与本书讨论内容相关的一些简单问题。题号为奇数的练习题答案放

置于本书的网站中，学生可以查阅。

思考题 这部分内容包括一些较难的问题，需要较为深入地理解本章的内容才能解答。我们强烈推荐学生尝试解决所有这些问题。题号为奇数的思考题答案也放置于本书的网站中，学生可以查阅。

模拟实验

如果能够动手对分组流和分组内容进行分析，那么就能更好地理解这些网络概念。多数章节包含了一部分用于帮助学生进行实验的内容。这一部分内容分为两部分。

Applets Java 小程序（Applets）放置在网站上，是由作者设计的交互式实验。这些小程序一部分用于更好地理解一些问题的解决方案，另一部分用于帮助读者在动手操作中更好地理解网络概念。

实验作业 一些章节包含了使用 Wireshark 模拟软件的实验作业。下载和使用 Wireshark 软件的方法在第 1 章中给出。另外一些章节给出的实验作业用于练习发送和接收分组，同时分析这些分组的内容。

编程作业

一部分章节包含有编程作业。编写一个有关进程或过程的程序能够澄清很多细节，并且能够帮助学生更好地理解隐藏在进程之后的概念。虽然学生可以使用自己熟悉的任意一种计算机语言编写和测试程序，但是本书网站中给出的答案是使用 Java 语言编写的，这些答案供教师使用。

附录

附录的目的是提供快速的资料参考和内容回顾，这些资料和内容可用于理解本书讨论的概念。

术语表和索引

为了更快地检索词汇和缩略语，本书给出了术语表和索引，但因篇幅所限，这些材料不包含在中文版书中，读者可到华章网站 http://www.hzbook.com 查阅。

教辅资源

本书包含的完整教辅资源可以从本书的网站 http://www.mhhe.com/forouzan 中下载[3]。这些资源包括以下内容。

幻灯片

网站给出了一组华美且栩栩如生的 PowerPoint 幻灯片，用于教学使用。

习题集答案

本书网站提供所有练习题和思考题的答案，供教师教学使用。

编程作业答案

本书网站也提供编程作业答案。其中第 2 章的程序采用 C 语言编写，其他章节的程序采用 Java 语言编写。

---

③ 采用该书作教材的教师可向 McGraw-Hill 公司北京代表处联系索取教学课件资料，传真：+8610-62790292；电子邮件：instructorChina@mcgraw-hill.com。

如何使用本书

本书章节的组织提供了很大的灵活性，我们建议如下：

- 第 1 章讨论的大部分内容是理解本书其他章节内容的基础。1.1 节和 1.2 节内容对理解网络分层非常关键，而网络分层是本书内容组织的基础。1.3 节和 1.4 节可以跳过或者安排为自学内容。
- 第 2 章至第 6 章基于 TCP/IP 协议族的顶部 4 层，我们建议按照次序讲授，以保持本书自顶向下的方法。可是，有些部分可以跳过而不会失去连续性，例如第 2 章的客户-服务器 Socket 接口、第 4 章的下一代 IP，第 5 章的其他有线网络。
- 为了使 TCP/IP 协议的讨论更加完整，本书添加了第 7 章物理层。如果教师感觉学生已经熟悉或者已经在相关课程中学习过这些内容，那么这些内容可以跳过。
- 在前 6 章讨论完后，第 8 章、第 9 章和第 10 章可以按任意次序讲授。教师可以全部或部分地讲授这些章节的内容，甚至可以跳过这些内容。
- 第 11 章为 Java 网络编程。该章有两个目的：首先，它给出客户-服务器编程思想，使学生更好地理解 Internet 的整体目标。其次，它可以为网络方面的高级课程做准备。第 2 章中关于 C 语言的内容与本章有一小部分重复，教师既可以使用第 2 章 C 语言部分的内容讲授网络编程基础，也可以使用第 11 章 Java 语言的内容进行讲授。

本书网站

本书网站 http://www.mhhe.com/forouzan 包含以下内容。

测试题

测试题放置于本书网站中，测试结果可以发送给讲授该课程的教师。

学生答案

奇数题号练习题和思考题的答案放置在本书网站中，用于帮助学生检查他们的学习状态。

Applets

学生可以使用为每章设计的小程序，观察实际的网络协议及其问题。

教师答案

所有练习题和思考题的答案放置在本书网站中，供讲授本课程的教师使用。

编程作业

编程作业的代码放置在本书网站中，供讲授本课程的教师使用。

PowerPoint 幻灯片

华美且精细加生的幻灯片放置在本书网站中，供讲授本课程的教师使用。教师可以修改这些幻灯片以适应课程的需要。

致谢

显然，编写如此篇幅的书籍需要很多人的帮助。我们非常感谢匿名评审专家在本书编写过程中做出的贡献。这些评审专家为：

Zongming Fei	肯塔基大学（University of Kentucky）
Randy J. Fortier	温莎大学（University of Windsor）
Seyed H. Hosseini	威斯康星大学米尔沃基分校（University of Wisconsin,Milwaukee）
George Kesidis	宾夕法尼亚州立大学（Pennsylvania State University）
Amin Vahdat	加利福尼亚大学圣地亚哥分校（University of California, San Deigo）
Yannis Viniotis	北卡罗来纳州立大学（North Carolina State University）
Bin Wang	莱特州立大学（Wright State University）
Vincent Wong	英属哥伦比亚大学（University of British Columbia）
Zhi-Li Zhang	明尼苏达大学（University of Minnesota）
Wenbing Zhao	克利夫兰州立大学（Cleveland State University）

特别感谢 McGraw-Hill 出版公司的人员。出版人 Raghu Srinivasan 证明了出版专家可以将不可能的事情变成可能。无论何时，只要有需要，策划编辑 Melinda Bilecki 都会给予帮助。在整个出版过程中，项目经理 Jane Mohr 一直以极大的热情指导我们。我们还要感谢项目经理 Dheeraj Chahal、封面设计人 Brenda A. Rolwes 和文字编辑 Kathryn DiBernardo。

Forouzan 和 Mosharraf
加利福尼亚，洛杉矶

目　录

Computer Networks: A Top-Down Approach

出版者的话
译者序
前言

第 1 章　概论 …… 1
1.1 Internet 概览 …… 1
1.1.1 网络 …… 1
1.1.2 交换 …… 3
1.1.3 Internet …… 5
1.1.4 访问 Internet …… 6
1.1.5 硬件和软件 …… 6
1.2 协议分层 …… 6
1.2.1 场景 …… 7
1.2.2 TCP/IP 协议簇 …… 8
1.2.3 OSI 模型 …… 15
1.3 Internet 发展史 …… 16
1.3.1 早期历史 …… 16
1.3.2 Internet 的诞生 …… 16
1.3.3 今天的 Internet …… 17
1.4 标准和管理 …… 18
1.4.1 Internet 标准 …… 18
1.4.2 Internet 管理 …… 19
1.5 章末资料 …… 20
推荐读物 …… 20
小结 …… 20
1.6 习题集 …… 21
测试题 …… 21
练习题 …… 21
思考题 …… 22
1.7 模拟实验 …… 23
Applets …… 23
实验作业 …… 23

第 2 章　应用层 …… 24
2.1 介绍 …… 24
2.1.1 提供服务 …… 24
2.1.2 应用层模式 …… 26
2.2 客户-服务器模式 …… 28
2.2.1 应用程序接口 …… 28
2.2.2 使用传输层的服务 …… 30
2.3 标准客户-服务器应用 …… 31
2.3.1 万维网和 HTTP …… 32
2.3.2 FTP …… 42
2.3.3 电子邮件 …… 45
2.3.4 TELNET …… 55
2.3.5 安全 Shell …… 57
2.3.6 域名系统 …… 58
2.4 对等模式 …… 66
2.4.1 P2P 网络 …… 66
2.4.2 分布式散列表 …… 68
2.4.3 Chord …… 70
2.4.4 Pastry …… 75
2.4.5 Kademlia …… 79
2.4.6 一种流行的 P2P 网络：BitTorrent …… 81
2.5 套接字接口编程 …… 83
C 的套接字接口 …… 83
2.6 章末资料 …… 94
推荐读物 …… 94
小结 …… 95
2.7 习题集 …… 95
测试题 …… 95
练习题 …… 95
思考题 …… 97
2.8 模拟实验 …… 99
Applets …… 99
实验作业 …… 99
2.9 编程作业 …… 99

第 3 章　传输层 …… 100
3.1 介绍 …… 100
传输层服务 …… 100
3.2 传输层协议 …… 110

3.2.1 简单协议 …… 111
3.2.2 停止-等待协议 …… 111
3.2.3 回退 N 帧协议 …… 115
3.2.4 选择性重复协议 …… 120
3.2.5 双向协议：捎带 …… 123
3.2.6 因特网传输层协议 …… 124
3.3 用户数据报协议 …… 125
3.3.1 用户数据报 …… 126
3.3.2 UDP 服务 …… 126
3.3.3 UDP 应用 …… 128
3.4 传输控制协议 …… 129
3.4.1 TCP 服务 …… 130
3.4.2 TCP 的特点 …… 131
3.4.3 段 …… 133
3.4.4 TCP 连接 …… 134
3.4.5 状态转换图 …… 139
3.4.6 TCP 中的窗口 …… 141
3.4.7 流量控制 …… 143
3.4.8 差错控制 …… 147
3.4.9 TCP 拥塞控制 …… 152
3.4.10 TCP 计时器 …… 159
3.4.11 选项 …… 162
3.5 章末资料 …… 162
推荐读物 …… 162
小结 …… 162
3.6 习题集 …… 163
测试题 …… 163
练习题 …… 163
思考题 …… 165
3.7 模拟实验 …… 169
Applets …… 169
实验作业 …… 169
3.8 编程作业 …… 169

第 4 章 网络层 …… 170
4.1 介绍 …… 170
4.1.1 网络层服务 …… 170
4.1.2 分组交换 …… 173
4.1.3 网络层性能 …… 177
4.1.4 网络层拥塞 …… 179
4.1.5 路由器的结构 …… 182
4.2 网络层协议 …… 183
4.2.1 IPv4 数据报格式 …… 184
4.2.2 IPv4 地址 …… 189
4.2.3 IP 分组的转发 …… 202
4.2.4 ICMPv4 …… 208
4.3 单播路由选择 …… 211
4.3.1 一般思想 …… 212
4.3.2 路由选择算法 …… 213
4.3.3 单播路由选择协议 …… 222
4.4 多播路由选择 …… 237
4.4.1 介绍 …… 237
4.4.2 多播基础 …… 239
4.4.3 域内路由选择协议 …… 243
4.4.4 域间路由选择协议 …… 248
4.5 下一代 IP …… 248
4.5.1 分组格式 …… 249
4.5.2 IPv6 寻址 …… 251
4.5.3 从 IPv4 到 IPv6 的过渡 …… 254
4.5.4 ICMPv6 …… 255
4.6 章末资料 …… 257
推荐读物 …… 257
小结 …… 257
4.7 习题集 …… 258
测试题 …… 258
练习题 …… 258
思考题 …… 260
4.8 模拟实验 …… 264
Applets …… 264
实验作业 …… 264
4.9 编程作业 …… 264

第 5 章 数据链路层：有线网络 …… 265
5.1 介绍 …… 265
5.1.1 结点和链路 …… 265
5.1.2 两类链路 …… 267
5.1.3 两个子层 …… 267
5.2 数据链路控制 …… 267
5.2.1 成帧 …… 267
5.2.2 流量控制和差错控制 …… 269
5.2.3 差错检测和纠错 …… 270
5.2.4 两种 DLC 协议 …… 280
5.3 多路访问协议 …… 285
5.3.1 随机访问 …… 285

5.3.2 受控访问 ····· 294
5.3.3 通道化 ····· 296
5.4 数据链路寻址 ····· 296
5.5 有线局域网：以太网协议 ····· 303
5.5.1 IEEE 项目 802 ····· 304
5.5.2 标准以太网 ····· 304
5.5.3 快速以太网（100 Mbps） ····· 309
5.5.4 千兆以太网 ····· 310
5.5.5 10 千兆以太网 ····· 310
5.5.6 虚拟局域网 ····· 310
5.6 其他有线网络 ····· 313
5.6.1 点对点网络 ····· 313
5.6.2 SONET ····· 317
5.6.3 交换网络：ATM ····· 322
5.7 连接设备 ····· 325
5.7.1 中继器或集线器 ····· 325
5.7.2 链路层交换机 ····· 326
5.7.3 桥接器 ····· 327
5.8 本章资料 ····· 328
推荐读物 ····· 328
小结 ····· 328
5.9 习题集 ····· 329
测试题 ····· 329
练习题 ····· 329
思考题 ····· 331
5.10 模拟实验 ····· 335
Applets ····· 335
实验作业 ····· 335
5.11 编程作业 ····· 335

第 6 章 无线网络和移动 IP ····· 336
6.1 无线局域网 ····· 336
6.1.1 介绍 ····· 336
6.1.2 IEEE 802.11 项目 ····· 339
6.1.3 蓝牙 ····· 347
6.1.4 WiMAX ····· 352
6.2 其他无线网络 ····· 353
6.2.1 通道化 ····· 353
6.2.2 蜂窝电话 ····· 358
6.2.3 卫星网络 ····· 366
6.3 移动 IP ····· 369
6.3.1 寻址 ····· 369
6.3.2 代理 ····· 370
6.3.3 三个阶段 ····· 371
6.3.4 移动 IP 的低效 ····· 374
6.4 章末资料 ····· 375
推荐读物 ····· 375
小结 ····· 376
6.5 习题集 ····· 376
测试题 ····· 376
练习题 ····· 376
思考题 ····· 377
6.6 模拟实验 ····· 380
Applets ····· 380
实验作业 ····· 380
6.7 编程作业 ····· 380

第 7 章 物理层与传输介质 ····· 381
7.1 数据和信号 ····· 381
7.1.1 模拟数据与数字数据 ····· 381
7.1.2 传输损损 ····· 387
7.1.3 数据速率限制 ····· 389
7.1.4 性能 ····· 390
7.2 数字传输 ····· 392
7.2.1 数字到数字转换 ····· 392
7.2.2 模拟到数字转换 ····· 397
7.3 模拟传输 ····· 400
7.3.1 数字到模拟转换 ····· 400
7.3.2 模拟到模拟转换 ····· 404
7.4 带宽利用 ····· 405
7.4.1 多路复用 ····· 405
7.4.2 扩频 ····· 410
7.5 传输介质 ····· 412
7.5.1 有向介质 ····· 412
7.5.2 无向介质：无线 ····· 416
7.6 章末资料 ····· 417
推荐读物 ····· 417
小结 ····· 417
7.7 习题集 ····· 418
测试题 ····· 418
练习题 ····· 418
思考题 ····· 419

第 8 章 多媒体和服务质量 ····· 423
8.1 压缩 ····· 423

8.1.1 无损压缩 …… 425
8.1.2 有损压缩 …… 431
8.2 多媒体数据 …… 435
8.2.1 文本 …… 435
8.2.2 图像 …… 435
8.2.3 视频 …… 436
8.2.4 音频 …… 439
8.3 因特网中的多媒体 …… 440
8.3.1 流式存储音频/视频 …… 440
8.3.2 流式实况音频/视频 …… 442
8.3.3 实时交互式音频/视频 …… 443
8.4 实时交互式协议 …… 447
8.4.1 新协议的基本原理 …… 448
8.4.2 RTP …… 450
8.4.3 RTCP …… 452
8.4.4 会话初始化协议 …… 454
8.4.5 H.323 …… 459
8.4.6 SCTP …… 460
8.5 服务质量 …… 470
8.5.1 数据流量特征 …… 470
8.5.2 流量分类 …… 471
8.5.3 通过流量控制获得 QoS …… 471
8.5.4 综合服务（IntServ） …… 475
8.5.5 区分服务（DiffServ） …… 478
8.6 章末资料 …… 479
推荐读物 …… 479
小结 …… 480
8.7 习题集 …… 480
测试题 …… 480
练习题 …… 480
思考题 …… 482
8.8 模拟实验 …… 487
Applets …… 487
实验作业 …… 487
8.9 编程作业 …… 487

第 9 章 网络管理 …… 488
9.1 介绍 …… 488
9.1.1 配置管理 …… 489
9.1.2 故障管理 …… 490
9.1.3 性能管理 …… 490
9.1.4 安全管理 …… 491
9.1.5 计费管理 …… 491
9.2 SNMP …… 491
9.2.1 管理器和代理 …… 491
9.2.2 管理组件 …… 492
9.2.3 概要 …… 493
9.2.4 SMI …… 493
9.2.5 MIB …… 497
9.2.6 SNMP …… 499
9.3 ASN.1 …… 502
9.3.1 语言的基本要素 …… 503
9.3.2 数据类型 …… 503
9.3.3 编码 …… 505
9.4 章末资料 …… 505
推荐读物 …… 505
小结 …… 506
9.5 习题集 …… 506
测试题 …… 506
练习题 …… 506
思考题 …… 507

第 10 章 网络安全 …… 508
10.1 介绍 …… 508
10.1.1 安全目标 …… 508
10.1.2 攻击 …… 509
10.1.3 服务和技术 …… 510
10.2 机密性 …… 511
10.2.1 对称密钥密码 …… 511
10.2.2 非对称密钥密码 …… 518
10.3 安全的其他方面 …… 522
10.3.1 消息完整性 …… 522
10.3.2 消息认证 …… 523
10.3.3 数字签名 …… 523
10.3.4 实体认证 …… 527
10.3.5 密钥管理 …… 529
10.4 Internet 安全 …… 533
10.4.1 应用层安全 …… 535
10.4.2 传输层安全 …… 540
10.4.3 网络层安全 …… 544
10.5 防火墙 …… 551
10.5.1 分组过滤防火墙 …… 552
10.5.2 代理防火墙 …… 552
10.6 章末资料 …… 553

推荐读物 …… 553
小结 …… 553
10.7 习题集 …… 554
测试题 …… 554
练习题 …… 554
思考题 …… 555
10.8 模拟实验 …… 556
Applets …… 556
实验作业 …… 556
10.9 编程作业 …… 556
第 11 章 Java Socket 编程 …… 559
11.1 介绍 …… 559
11.1.1 地址和端口 …… 559
11.1.2 客户-服务器模式 …… 562
11.2 UDP 编程 …… 563
11.2.1 迭代方法 …… 563
11.2.2 并发方法 …… 571
11.3 TCP 编程 …… 573
11.3.1 迭代方法 …… 573
11.3.2 并发方法 …… 581
11.4 章末资料 …… 583
推荐读物 …… 583
小结 …… 583
11.5 习题集 …… 583
测试题 …… 583
练习题 …… 583
思考题 …… 584
11.6 编程作业 …… 585
附录 A Unicode …… 586
附录 B 按位计数系统 …… 590
附录 C HTML、CSS、XML 和 XSL …… 595
附录 D 其他信息 …… 601
附录 E 8B/6T 编码 …… 603
参考文献 …… 605

第1章

Computer Networks: A Top-Down Approach

概　论

最大的计算机网络——因特网（Internet），拥有超过10亿的用户。利用有线和无线传输介质，Internet连接了大大小小的计算机系统。它允许用户共享包括文本、图像、音频和视频在内的大量信息，允许用户之间相互发送消息。本书的主要目标就是探索这个庞大的系统。在本章，我们有两个目标。第一个目标是对作为一个互联网（网中网）的Internet进行概述，讨论Internet的组成部分。这个目标的部分内容为介绍协议分层和TCP/IP协议簇。换句话说，第一个目标是为本书的其他章节做准备。第二个目标是提供相关的信息，但是这些信息在学习其他章节时不是必需的。

- 1.1节介绍局域网（local area network，LAN）和广域网（wide area network，WAN），给出这两种类型网络的主要定义。我们定义一个局域网和广域网相结合的互联网络——互联网。我们将展示一个组织怎样利用广域网将它的局域网连接起来，从而创建一个私有的互联网。最后，我们介绍由主干、网络提供商和用户网络组成的Internet，它是一个全球性的互联网络。
- 1.2节我们利用协议分层（protocol layering）的概念展示Internet怎样将任务分解成多个小任务。我们将讨论5层协议栈（TCP/IP），介绍每层的任务和每层拥有的协议。我们还将讨论在这种模式下的两个概念：封装/解封装（encapsulation/decapsulation）和多路复用/多路分解（multiplexing/demultiplexing）。
- 1.3节我们为感兴趣的读者介绍Internet的主要发展史。跳过这部分内容不会丧失本书的连续性。
- 1.4节介绍Internet的管理、标准的制定和生命周期。这部分内容仅仅提供相关的信息，对理解本书其他章节内容不是必需的。

1.1　Internet概览

尽管本书的目标是讨论Internet，一个连接世界上数十亿计算机终端的系统，但是我们认为Internet不是一个单一的网络，而是一个互联网络（internetwork），多个网络的组合。所以，我们首先对网络进行定义，然后展示怎样连接多个网络来创建小型的互联网络。最后，我们介绍Internet的结构，进而打开后面10章学习Internet的大门。

1.1.1　网络

网络（network）是由一组具有通信能力的设备相互连接而形成的。在这个定义中，设备可以是主机（host，有时也称为端系统（end system），如大型计算机、桌面计算机、笔记本电脑、工作站、无线电话、安全系统等），也可以是连接设备，如连接网络到其他网络的路由器、将设备连接到一起的交换机、变换数据形式的调制解调器等。在一个网络中，这些设备使用有线或无线的传输介质（如电缆或大气）连接起来。当家庭中利用一台即插即用的路由器连接两台电脑时，我们就组建了一个网络，尽管这个网络很小。

局域网

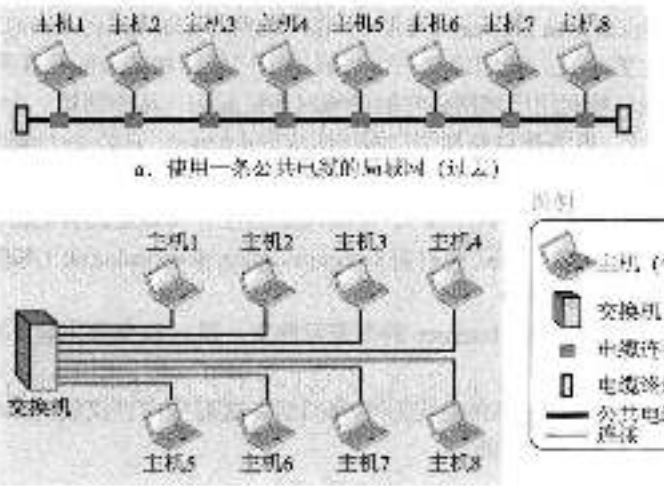
局域网（local area network，LAN）通常是私有的，连接一个办公室、大楼或校园内的一些主机。按照需求的不同，一个局域网既可以简单地由两台电脑和一台打印机组成，用于家庭办公，也

可以贯穿整个公司，包含语音和视频设备。在局域网中的每台主机都具有一个标识符（一个地址），用于在局域网中唯一地定义这台主机。一台主机向另一台主机发送的数据包携带了源主机和目的主机的地址。

在过去，一个网络中的所有主机都连接到一个公共的电缆上，这意味着从一台主机发往另一台主机的数据包可以被所有的主机接收到。目标接收者保存这个数据包，而其他主机丢弃该数据包。现在，多数局域网采用智能连接交换机，它能够识别数据包的目的地址并引导该数据包到达它的目的地而不必将它发送到其他主机。交换机减轻了局域网中的流量，如果不是共同的源主机和目的主机，那么交换机允许同一时刻多对主机之间同时相互通信。注意，上面局域网的定义没有指定局域网中最小或最大的主机数。图 1-1 显示了使用公用电缆和交换机组成的局域网。

第 5 章和第 6 章将对局域网进行详细讨论。

当孤立地使用局域网时（目前已很少见），它们用于主机之间共享资源。我们马上能看到，目前的局域网常常相互连接，同时连接到广域网（下面我们将讨论），以进行更广范围的通信。



a. 使用一条公共电缆的局域网（过去）

b. 使用一台交换机的局域网（现在）

图 1-1 过去和现在的孤立局域网

广域网

广域网（wide area network，WAN）也是由具有通信能力的设备相互连接而形成的。可是，局域网和广域网有一些不同。局域网通常覆盖范围受限，可以覆盖一间办公室、一栋大楼或一个校园；广域网则具有更广的地理覆盖范围，可以覆盖一个城市、一个省、一个国家甚至整个世界。局域网互联主机；广域网互联交换机、路由器、调制解调器等连接设备。局域网通常由使用它的组织拥有；广域网通常由通信公司建设和运营，使用它的组织进行租用。我们看看目前使用的两种典型的广域网：点到点广域网和交换式广域网。

点到点广域网

点到点广域网通过传输介质（电缆或大气）连接两个通信设备。在讨论怎样把一个网络连接到另一个网络时，我们将看到这些广域网的例子。图 1-2 显示了一个点到点广域网的例子。

交换式广域网

交换式广域网具有多个端点。不久我们将看到，交换式广域网目前用作全球通信的主干。我们可以说，交换式广域网是交换机连接几个点到点的广域网而形成的。图 1-3 给出了一个交换式广域

网的例子。



图 1-2 点到点广域网

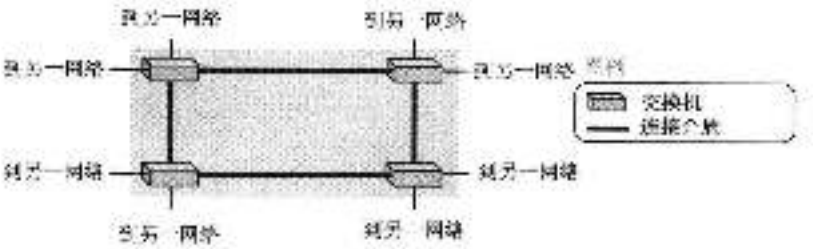


图 1-3 交换式广域网

第 5 章和第 6 章将对广域网进行详细讨论。

互联网络

现在我们很少看到孤立的局域网或广域网，它们都相互连接在一起。当两个或多个网络连接起来，它们就形成了一个互联网络（internetwork），或者互联网（internet）。例如，假如一个机构有两个办公室，一个在东海岸，另一个在西海岸。每一个办公室都拥有一个局域网，允许办公室的员工相互进行通信。为了使不同办公室的员工能够互相通信，管理部门从服务提供商（例如电话公司）租用一个点到点的专用广域网，用来连接两个局域网。现在公司拥有了一个互联网络，或者说一个私有互联网，不同办公室之间的通信变成了可能。图 1-4 显示了这个互联网。

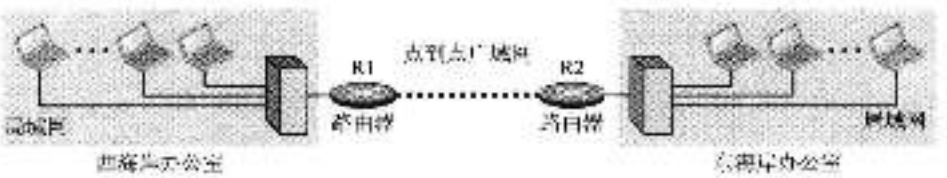


图 1-4 由两个局域网和一个点到点广域网组成的互联网络

当西海岸办公室中的一台主机给同一办公室的另一台主机发送消息时，路由器阻截这条消息，但是交换机指引这条消息到达目的地。另一方面，当西海岸的一台主机给东海岸的一台主机发送消息时，路由器 R1 将数据包路由到路由器 R2，然后数据包到达目的地。

图 1-5 显示了多个局域网和广域网连接形成的另一个互联网。广域网之一为只有 4 个交换机的交换式广域网。

1.1.2 交换

我们已经说过，互联网是由链路和交换机组成的，例如我们前面使用的链路层交换机和路由器。实际上，互联网是一个交换式的网络，其中一台交换机至少将两条链路连接在一起。当需要的时候，交换机需要将数据从一条链路转发到另一条链路。交换式网络最常见的类型为电路交换网络和分组交换网络。下面我们讨论这两种类型的网络。

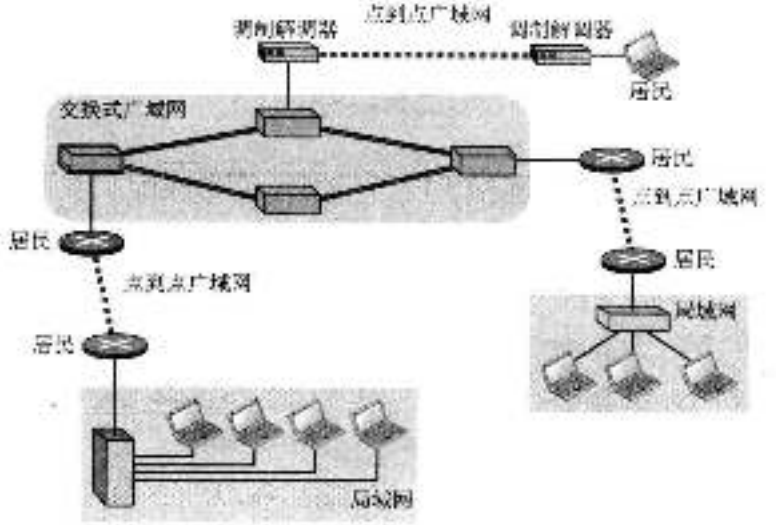


图 1-5 一个由 4 个广域网和 3 个局域网组成的异构网络

电路交换网络

在电路交换网络（circuit-switched network）中，两个端系统之间总是存在一条专用的连接（称为电路），交换机只能使其变成活跃或非活跃状态。图 1-6 显示了一个简单的交换式网络，该网络在每端连接 4 部电话。由于过去电话网络经常采用电路交换，因此我们使用电话机代替计算机作为端系统，尽管目前部分电话系统采用分组交换网络。



图 1-6 电路交换网络

在图 1-6 中，每端的 4 部电话连接到一个交换机。交换机将一端的电话机连接到另一端的电话机。连接两台交换机的粗线是一个高容量的通信线路，它能够同时处理 4 路语音通信，其容量能够被所有电话对之间共享。本例使用的交换机具有转发功能但是没有存储能力。

我们看看以下两种情况。在第一种情况下，所有电话机均处于忙状态：一端的 4 个人正在与另一端的 4 个人进行通话；粗线的容量被完全使用。在第二种情况下，一端只有一部电话机连接到另一端的电话机；粗线容量仅仅四分之一被使用。这意味着仅当占用全部容量时，电路交换网络才具有高效率；在多数时间中，由于工作仅仅占用部分容量，因此它的效率低下。需要将粗线的容量做成每条语音线路容量 4 倍的原因是，当一端的所有电话机想要与另一端所有电话机连接时，我们不希望通信失败。

分组交换网络

在一个计算机网络中，两个端点之间使用被称为分组（packet）的数据块进行通信。也就是说，与正在使用的电话机之间连续通信不同，两台计算机之间交换的是独立的数据分组。由于分组是一个能够被存储和以后发送的独立实体，因此这种机制允许我们实施存储转发的交换功能。图 1-7 显示了一个每端分别连接 4 台计算机的小型分组交换网络。

分组交换网络中的路由器具有能够存储和转发分组的队列。现在假设粗线的容量（即高容量）

仅仅为连接计算机到路由器数据线容量的两倍。如果只有两台计算机（分别在两端）需要相互通信，那么发送的分组不需要等待。但是，如果当链线已经工作在满负荷时分组到达一个路由器，那么应该存储分组并且按照它们到达的次序进行转发。虽然这两个简单的例子显示分组交换网络比电路交换网络效率高，但是分组可能会遇到一些延迟。



图 1-7 分组交换网络

本书我们主要讨论分组交换网络。在第 4 章中，我们将详细讨论分组交换网络，同时讨论这些网络的性能。

1.1.3 Internet

正像我们以前讨论的那样，互联网是由两个或多个能够相互通信的网络组成的。最著名的互联网叫做因特网（Internet）。Internet 由成千上万个相互连接的网络组成。图 1-8 显示了一个 Internet 概念上（而不是物理上）的视图。

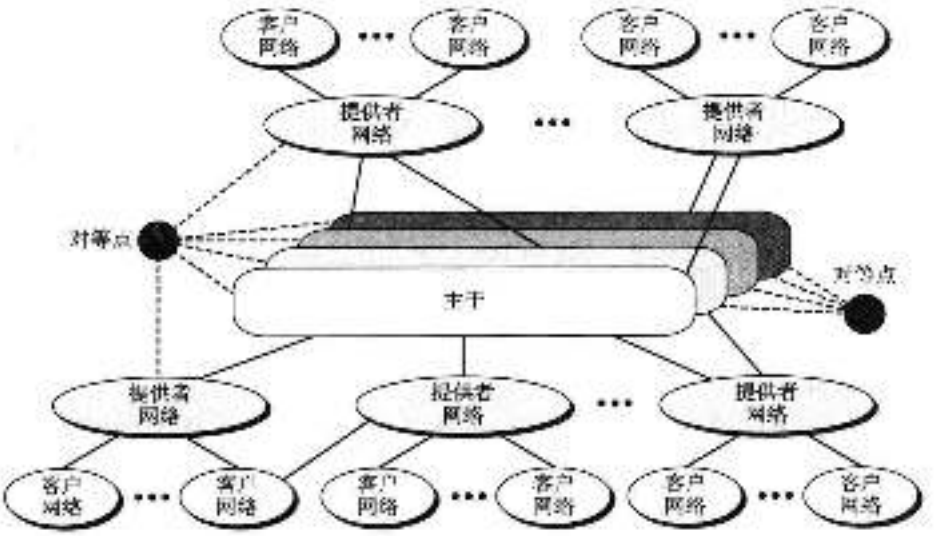


图 1-8 现今的 Internet

从图 1-8 中看到，Internet 由一系列主干、提供者网络和客户网络组成。主干（backbone）处于最高层次，是一些通信公司拥有的大型网络，如 Sprint、Verizon(MCI)、AT&T、NTT。主干网络通过称为对等点（peering point）的复杂交换系统进行连接。一些小些的提供者网络（provider network）处于第二个层次，这些网络通过付费使用主干网络服务。提供者网络连接主干网络，有时提供者网络之间也相互连接。客户网络（customer network）是 Internet 边缘的网络，它们使用 Internet 提供的服务。为了接收服务，客户网络需要向提供者网络付费。

主干和提供者网络也称为 Internet 服务提供商（Internet Service Provider，ISP）。主干常常称为

国际 ISP；提供者网络常常称为国家或区域 ISP。

1.1.4 访问 Internet

今天的 Internet 是一个允许任何用户变成它的一部分的互联网。但是，用户需要物理上连接到一个 ISP。物理连接通常利用一条点到点的广域网实现。本节我们会简单描述怎样进行连接，至于连接的详细技术信息我们将在第 6 章和第 7 章讨论。

使用电话网络

目前大多数居民和小公司具有电话服务，这就意味着他们能够连接到电话网络。由于大多数电话网络自身已经连接到 Internet，因此居民和小公司连接 Internet 的一个选择是把他们和电话中心的语音线路转换成点到点的广域网。这可以用两种方式实现。

- **拨号服务**。第一种解决方法是在电话线路中增加将数据转换成语音的调制解调器。安装在计算机中的软件拨打 ISP 的号码，形成一条电话连接。非常不幸，拨号服务非常慢，同时当线路用于 Internet 连接时，线路就不能进行电话（语音）连接。因此，这种方式只对偶尔访问 Internet 的居民和小公司有效。我们将在第 5 章中讨论拨号服务。
- **DSL 服务**。自从 Internet 出现后，一些电话公司开始升级它们的电话线路，以向居民和小公司提供较高速率的 Internet 服务。DSL 服务允许语音和数据通信同时进行。我们将在第 5 章讨论 DSL。

利用有线电视网络

近 20 年，越来越多的居民开始使用有线电视服务替代天线接收电视广播。有线电视公司已经升级了它们的有线电视网络并连接到 Internet。居民和小公司可以通过这种服务连接到 Internet。虽然这种方法可提供较高速率的连接，但是速率与使用同一电缆的用户数目有关。我们将在第 5 章讨论有线电视网络。

采用无线网络

无线连接最近变得非常流行。住户或小公司可以使用无线和有线连接混合的方法访问 Internet。随着无线广域网接入的发展，住户或小公司能够通过无线广域网连入 Internet。我们将在第 6 章讨论无线接入。

直接连接到 Internet

大机构或大公司自身可以变成一个本地 ISP 并连入 Internet。这种方法要求组织或公司从一个线路提供者那里租用高速广域网并将它连入地区 ISP。例如，具有几个校园的大学可以组建一个互联网，然后连接互联网至 Internet。

1.1.5 硬件和软件

我们已经给出了 Internet 结构的概览，Internet 是由连接设备将大型和小型网络连接起来形成的。但是应该清楚地看到，仅仅连接这些东西是不够的。为了使通信正常进行，我们既需要硬件也需要软件。这就像一个复杂的计算，我们既需要计算机也需要程序。在下一节，我们讨论如何利用协议分层对硬件和软件的组合进行相互协调。

1.2 协议分层

当谈到 Internet 时，我们总能听到的一个词汇就是协议（protocol）。协议定义了发送者、接收者和所有中间设备为了高效通信需要遵循的规则。当通信简单时，我们可能只是需要一个简单的协议；当通信复杂时，我们可能需要把任务划分到不同层，每层需要一个协议，也就是说需要协议分层（protocol layering）。

1.2.1 场景

为了更好地理解为什么需要协议分层，我们来看两种简单的场景。

场景一

在第一个场景中，通信如此简单以至于它能够在一层中实现。假设 Maria 和 Ann 是拥有很多共同想法的邻居。Maria 和 Ann 之间的通信发生在一个层次中，她们面对面并使用相同的语言，如图 1-9 所示。



图 1-9 单一层次的协议

即使在这个简单的场景中，我们也可以看到需要遵循一系列的规则。第一，Maria 和 Ann 了解当她们相遇时应该互相问候。第二，她们明白她们应该限制使用的词汇在她们友谊的层次上。第三，一方知道当另一方讲话时，她应该抑制自己讲话。第四，每一方都知道应该是对话而不是独角戏；双方都应该有机会对某一问题发表看法。第五，当她们分别时，应该交换一些祝福语。

我们可以看到，Maria 和 Ann 使用的协议与课堂上老师和学生的通信不同。第二种情况中，大部分情况下是独角戏；除非学生有问题，老师的谈话会占用大部分时间。在这种情况下，协议应该规定学生应该举手并等待被允许说话。这种情况下的通信通常非常正式，同时限制在讲授的前提下。

场景二

在第二个场景中，我们假设公司提拔了 Ann，但是她需要到距离 Maria 很远的城市上班。由于这两位朋友着手进行一个新的项目以便退休后启动新生意，因此她们希望继续她们的通信、交换她们的想法。她们决定定期通过邮局使用信件继续交换她们的想法。但是，即使信件被拦截，她们也不希望她们的想法被其他人知道。她们一致同意采用一种加密/解密技术。信件的发送者对信件加密，使之对入侵者不可读；信件的接收者对信件解密，从而得到原始的信件。我们将在第 10 章讨论加密/解密方法，但是现在我们假设 Maria 和 Ann 采用了一种技术，该技术在一个人不拥有密钥的情况下很难解密信件。现在我们可以说 Maria 和 Ann 之间的通信在 3 个层次上进行，如图 1-10 所示。我们假设 Ann 和 Maria 每个人拥有 3 台机器（机器人），这 3 台机器分别在每一层执行任务。

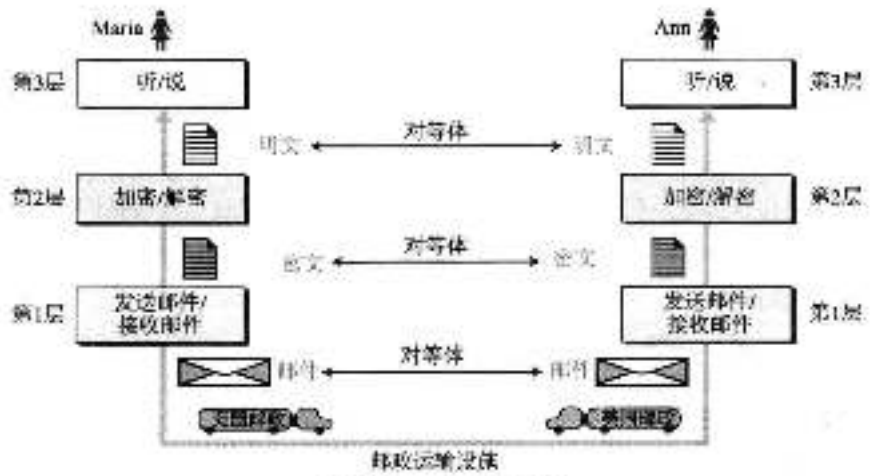


图 1-10 一种三层协议

我们假设 Maria 向 Ann 发送第一封邮件。Maria 在第 3 层对机器谈话，仿佛这台机器就是 Ann，并且在听她谈话。第 3 层机器聆听 Maria 所说的内容并形成了明文（用英文书写的邮件），传递给第 2 层机器。第 2 层机器接收明文，对它进行加密并形成密文，传递给第 1 层机器。第 1 层机器，大概是个机器人，接收密文，把它放入一个信封中，添加发送者和接收者地址，然后进行邮寄。

在 Ann 的一端，第 1 层机器从 Ann 的信箱中取出邮件，通过发送者的地址得知该邮件来自于 Maria。机器从信封中取出密文并将它投递给第 2 层机器。第 2 层机器解密这个信息，形成明文并将明文传递给第 3 层机器。第 3 层机器接收明文并仿佛 Maria 正在说话一样将它读出来。

协议分层允许我们将一个复杂的任务分解成几个较小的、简单的任务。例如，在图 1-10 中，我们可以只使用一台机器完成所有 3 台机器的工作。可是，如果 Maria 和 Ann 判定这台机器所做的加密/解密不足以保护她们的秘密，那么她们需要更换整台机器。在现在的情况下，她们只需要更换第 2 层的机器，另外两台机器能够保持不变。这种方法称为模块化（modularity）。在这个示例中，模块化意味着独立的层次。一层（一个模块）可以定义为一个具有输入和输出的黑盒子，我们不必关心输入如何变成输出。如果给定相同的输入，两台机器提供相同的输出，那么它们可以相互替换。例如，Ann 和 Maria 可以从两个不同的厂商购买第 2 层机器。只要这两台机器能把相同的明文变成相同的密文，相同的密文变成相同的明文，那么它们就可以相互替换。

协议分层的优越性之一是它允许我们将服务从实现中分离出来。一层需要能够接收较低层的一系列服务，同时向较高层提供服务，而我们不关心这一层是如何实现的。例如，Maria 可以决定不为第 1 层购买机器（机器人）；她可以自己做这些工作。只要 Maria 能够在两个方向上完成第 1 层提供的任务，通信系统就可以工作。

协议分层的另一个优越性无法在简单的示例中体现，但是当我们讨论 Internet 中的协议分层时能够展现出来。这个优越性就是通信不只是用于两个端系统，中间系统只需要一些层次而不是所有的层次。如果不使用协议分层，形成的中间系统就不得不像端系统一样复杂，这样就会提高每个系统的造价。

协议分层有不足之处吗？有人说单一层次使工作更加容易。对每一层来说，没有必要向上一层提供服务并使用下一层的服务。例如，Ann 和 Maria 可以寻找或建造一台机器来完成这三种任务。可是，正像前面提到的那样，如果某一天她们发现她们的编码被攻破，那么每人都不得不用新机器替换整个机器，而不是只更换第 2 层的机器。

协议分层原则

让我们讨论一些协议分层原则。第一个原则就是如果想要双向通信，那么我们需要每一层能够实现两个相反的任务，每个方向上一个。例如，第 3 层的任务就是听（在一个方向上）和说（在另一个方向上）。第 2 层需要能够加密和解密。第 1 层需要发送和接收邮件。

在协议分层中我们需要遵循的第二个原则是两端每一层中的两个对象应该相同。例如，两端第 3 层的对象应该为明文信件。两端第 2 层的对象应该为密文信件。两端第 1 层的对象应该为一封邮件。

逻辑连接

在遵循以上两个原则之后，每层之间的逻辑连接如图 1-11 所示。这意味着我们把右层到层的通信。Maria 和 Ann 可以认为每一层有一个逻辑（想象的）连接，通过这个连接她们可以发送那一层创建的对象。我们将看到逻辑连接的概念将帮助我们更好地理解数据通信与网络中遇到的分层任务。

1.2.2 TCP/IP 协议簇

在第 2 个场景中，我们了解了协议分层与层间逻辑连接的概念，现在介绍 TCP/IP（Transmission Control Protocol/Internet Protocol，传输控制协议/互联网协议）。TCP/IP 是目前 Internet 使用的一个协议族（按不同层次组织的协议集）。它是由相互交互的模块组成的一个层次结构协议，每一个模块提供特定的功能。层次意味着较上层次的协议需要得到一个或多个较下层次协议提供的服务支持。初始的 TCP/IP 协议族在硬件基础上定义了 4 个软件层次。但是，目前 TCP/IP 通常是一个 5

层模型。图 1-12 显示了这两种情况。

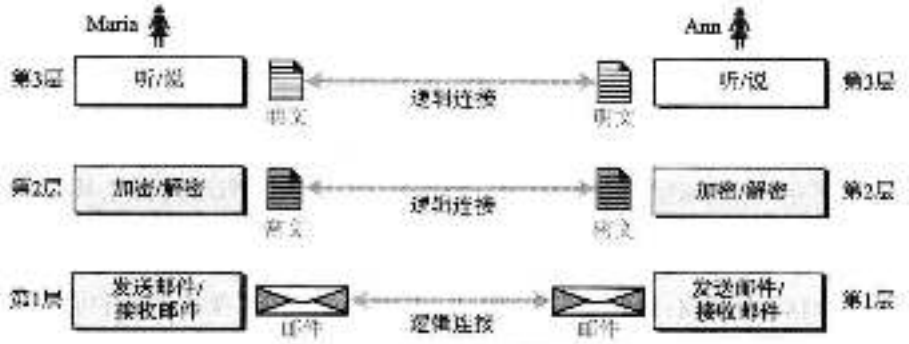


图 1-11 对等层之间的逻辑连接

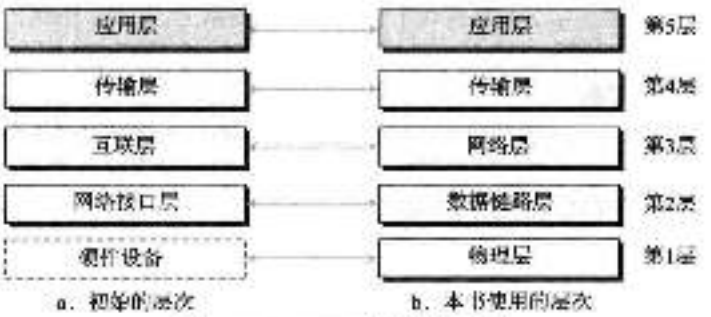


图 1-12 TCP/IP 协议簇的层次

层次化结构

为了展示如何利用 TCP/IP 协议簇的层次在两台主机之间进行通信，我们假设将协议簇用于一个由 3 个局域网（链接）组成的小型互联网。每个局域网都有一个链路层交换机。同时我们假设局域网连接到一个路由器，如图 1-13 所示。

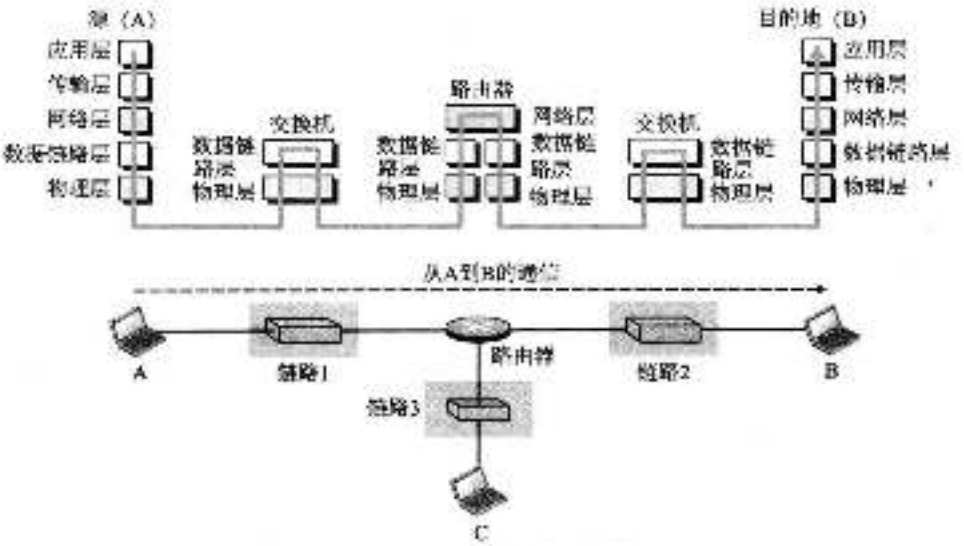


图 1-13 通过一个互联网通信

我们假设计算机 A 与计算机 B 进行通信。正像图 1-13 中显示的那样，在这个通信中涉及 5 个通信设备：源主机（计算机 A）、链路 1 的链路层交换机、路由器、链路 2 的链路层交换机和目的主机（计算机 B）。按照设备在互联网中扮演的角色不同，每一台设备包含了几个层次。两台主机包含了所有 5 个层次；源主机需要在应用层创建一个信息并把它发送到下层，以便把该信息物理上发送到目的主机。目的主机需要在物理层接收这个信息，然后通过其他层投递到应用层。

路由器只涉及 3 层；只要路由器仅仅作为路由选择，在路由器中就没有传输层或应用层。虽然一个路由器总是拥有一个网络层，但是它涉及 n 个数据链路层和物理层的组合，其中 n 为路由器连接的链路的数目。其主要原因是每一个链路可以使用它自己的数据链路或物理层。例如在图 1-13 中，路由器拥有 3 条链接，但是从源 A 发送到目的地 B 的消息涉及两条链接。每一条链接可以使用不同的链路层和物理层协议；路由器需要从基于一对协议的链路 1 接收分组并将它投递到基于另一对协议的链路 2。

可是，在一条链路上的链路层交换机只涉及两个层次：数据链路层和物理层。尽管图 1-13 显示的交换机拥有两个不同的连接，但是这两个连接在同一链路上，它们只使用一个协议集。这意味着链路层交换机与路由器不同，它只涉及一个数据链路层和一个物理层。

TCP/IP 协议簇中的层次

在进行前面的介绍之后，我们来简单讨论 TCP/IP 协议簇中层次的功能和任务。接下来的 6 章将对每一层进行详细讨论。为了更好地理解每一层的任务，我们首先需要知道在层次间存在的逻辑连接。图 1-14 显示了简单互联网的逻辑连接。

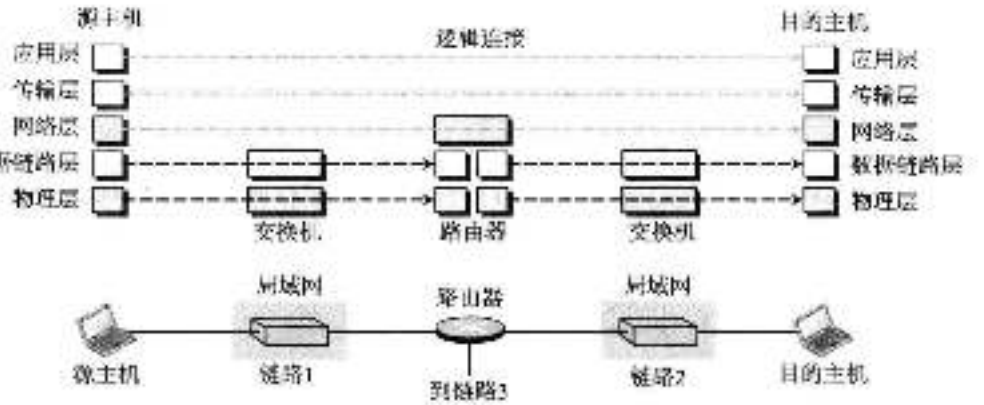


图 1-14 TCP/IP 协议簇中层次间的逻辑连接

采用逻辑连接使我们考虑每一层的任务变得比较容易。如图 1-14 所示，应用层、传输层和网络层的任务是端到端的（end-to-end）。但是，数据链路层和物理层的任务是点到点的（hop-to-hop），其中一个跳步是一个主机或路由器。也就是说，高三层的任务范围是互联网，低两层的任务范围是链路。

另一种理解逻辑连接的方法是考虑每一层创建的数据单元。在高三层，数据单元（分组）不应该被任何路由器或链路层交换机改变。在低两层，主机创建的分组仅仅被路由器改变，链路层交换机不改变它们。

图显示了前面讨论的协议分层的第二个原则。我们看一看与设备相关的每一层之下的对等体。

注意，尽管网络层的逻辑连接在两个主机之间，但是由于一个路由器在网络层对分组进行分片，并且发送的分组比接收的多（见第 4 章分片部分），因此在这种情况下，对等体只存在于两个跳步之间。

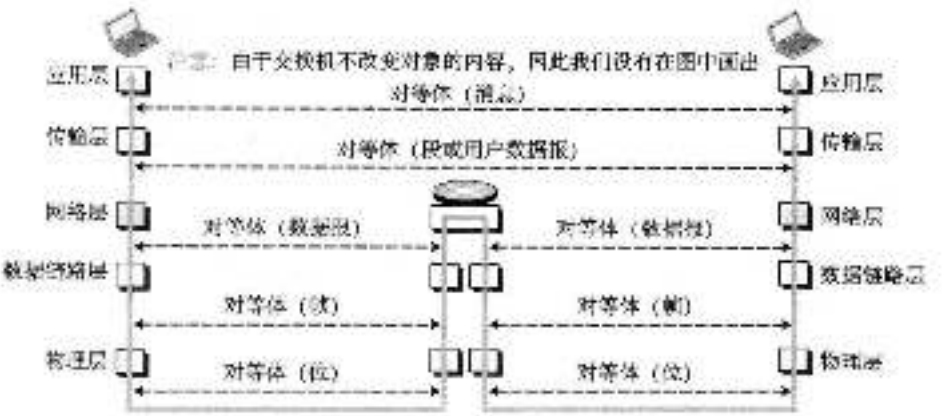


图 1-15 TCP/IP 协议簇中的对等体

TCP/IP 各层描述

理解逻辑通信的概念之后，我们简单讨论各层的主要任务。本章的讨论将非常简单，不过我们会在接下来的 6 章继续讨论各层的功能与任务。

应用层

如图 1-14 所示，两个应用层之间的逻辑连接是端到端的。两个应用层之间仿佛存在一座桥梁一样相互交换消息。可是，我们应该明白通信需要通过所有层次完成。

应用层的通信处于两个进程（该层正在运行的两个程序）之间。为了进行通信，一个进程向另一个进程发送请求，并且接收另一个进程的响应。进程到进程的通信就是应用层的任务。虽然 Internet 的应用层包含了很多预定义的协议，但是也可以在两台主机上运行用户创建的一对进程。我们将在第 2 章研究这种情况。

超级文本传输协议（Hypertext Transfer Protocol，HTTP）是访问万维网（World Wide Web，WWW）的载体。简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）是电子邮件（e-mail）服务的主要协议。文件传输协议（File Transfer Protocol，FTP）用于将文件从一台主机传输到另一台主机。远程登录（Terminal Network，TELNET）和安全外壳（Secure Shell，SSH）用于访问远端的站点。管理员使用简单网络管理协议（Simple Network Management Protocol，SNMP）对 Internet 全局或局部进行管理。域名系统（Domain Name System，DNS）使其他的协议能够查询一台计算机的网络层地址。因特网组管理协议（Internet Group Management Protocol，IGMP）用于管理一个组的成员资格。我们将在第 2 章讨论这些协议的大部分，其他协议在另外一些章节讨论。

传输层

传输层的逻辑连接也是端到端的。源主机的传输层从应用层得到消息，封装成传输层的分组（称为段或用户数据报，不同协议叫法不同），然后进行发送。通过逻辑（想象的）连接，分组到达目的主机的传输层。也就是说，传输层负责向应用层提供服务：从运行于应用层的程序得到信息，并将它投递到目的主机相应的应用程序。我们也许要问为什么我们已经拥有了一个端到端的应用层，还需要端到端的传输层。与我们前面讨论的一样，其主要原因是分割任务与责任。传输层应该独立于应用层。另外，我们将看到传输层有多个协议，这意味着每个应用程序可以使用与它的需求最匹配的协议。

正像我们说的，Internet 中有几个传输层协议，每个都是为一些特定的任务设计的。作为主要的协议，传输控制协议（Transmission Control Protocol，TCP）是一个面向连接的协议，它在传输数据之前，首先在两台主机的传输层之间建立一条逻辑连接。TCP 协议在两个 TCP 层之间创建一

个管道，以便传输字节流。TCP 协议提供流量控制（匹配源主机的发送数据速率与目的主机的接收数据速率，以防止目的主机溢出）、差错控制（保证数据段无差错到达目的地和重新发送受损的数据段）、拥塞控制（减少由于网络拥塞造成的数据段丢失）。另一种常用的协议是用户数据报协议（User Datagram Protocol，UDP）。UDP 是一种无连接协议，它传输用户数据报之前不需要创建逻辑连接。在 UDP 中，每个用户数据报是一个独立的实体，它和前一个或后一个用户数据报没有关系（无连接就是这个意思）。UDP 是一种比较简单的协议，它不提供流量控制、差错控制或拥塞控制。它的简单性（意味着小的额外开销）对某些应用程序具有吸引力，这些应用程序发送较短的消息且不能容忍 TCP 在分组损坏或丢失时使用重发机制。流控制传输协议（Stream Control Transmission Protocol，SCTP）是一种新协议，它是为多媒体出现的新应用设计的。我们将在第 3 章讨论 UDP 和 TCP，第 8 章讨论 SCTP。

网络层

网络层负责在源计算机和目的计算机之间创建一个连接。网络层的通信是主机到主机的。可是，由于从源主机到目的主机可能存在多个路由器，因此路径上的路由器负责为每个分组选择最好的路径。我们可以说网络层负责主机到主机的通信，并且指挥分组通过合适的路由器。我们再次问一问我们自己为什么需要网络层。我们可以在传输层增加路由任务，同时去掉这一层。正像我们前面介绍的，原因之一是在不同的层次之间分割不同的任务。原因之二是路由器不需要应用层和传输层。分割任务允许我们在路由器上加载较少的协议。

Internet 的网络层包括其主要协议：因特网协议（Internet Protocol，IP）。因特网协议定义了在网络层称为数据报的分组格式。IP 同时定义了在这一层使用的地址格式和结构。与此同时，IP 负责从源主机把一个分组路由到目的主机。这种功能主要是通过每个路由器都将数据报转发到路径上的下一个路由器而实现的。

IP 是一个无连接的协议，不提供流量控制、差错控制和拥塞控制服务。这意味着如果一个应用需要这些服务，那么应用需要依赖于传输层协议。网络层也包括单播（一对一）和多播（一对多）路由协议。虽然路由协议不参加路由（路由是 IP 的责任），但是它为路由器创建转发路由表，为转发处理提供帮助。

网络层也包含一些帮助 IP 转发和进行路由工作的辅助协议。在路由一个分组时，因特网控制报文协议（Internet Control Message Protocol，ICMP）帮助 IP 报告遇到的问题。因特网组管理协议（Internet Group Management Protocol，IGMP）协助 IP 进行多任务处理。动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）帮助 IP 获取一台主机的网络层地址。在网络层地址已知时，地址解析协议（Address Resolution Protocol，ARP）帮助 IP 寻找一台主机或一台路由器的链路层地址。我们在第 4 章讨论 ICMP、IGMP 和 DHCP，在第 5 章讨论 ARP。

数据链路层

我们已经知道一个互联网是多个链路（LAN 和 WAN）通过路由器连接而构成的。从主机传输数据报到目的地可能存在多个交叠的链路集，路由器负责选择最好的链路进行传输。可是，当路由器定好需要传输的下一条链路后，数据链路层接管这个数据报并使它穿过这条链路。这条链路可以是一个具有链路层交换机的有线局域网、一个无线局域网、一个有线广域网或者一个无线广域网。对于不同链路类型也存在不同的协议。无论哪种情况，数据链路层都要负责通过链路传输分组。

TCP/IP 没有为数据链路层定义任何特定的协议。它支持所有标准的和私有的协议。能够接管数据报并携带它穿过链路的任何协议都能满足网络层的要求。数据链路层接管一个数据报并将它封装在一个称为帧（frame）的分组中。

每个链路层协议可能提供不同的服务。有些链路层协议提供完整的检查和纠错，有些只提供纠错。我们在第 5 章讨论有线链路，在第 6 章讨论无线链路。

物理层

我们可以说物理层负责携带一个帧中单独的比特穿过链路。尽管物理层位于 TCP/IP 协议簇的最底层，但是由于在物理层之下存在另外一个隐藏的传输介质层，因此两个设备物理层之间的通信仍然是逻辑通信。两个设备通过一种传输介质（电缆或大气）连接。我们需要知道传输介质不携带比特；它携带电或光信号。所以，从数据链路层接收的一个帧的比特需要被变换，然后通过传输介质传输。但是我们可以认为两个设备物理层之间的逻辑单元是一个比特（bit）。将一个比特变换成一个信号存在多种协议。我们将在第 7 章讨论物理层和传输介质时讨论这些内容。

封装和解封装

在 Internet 协议分层中，一个重要的概念是封装/解封装。图 1-16 显示了图 1-13 给出的小型互联网的封装/解封装情况。

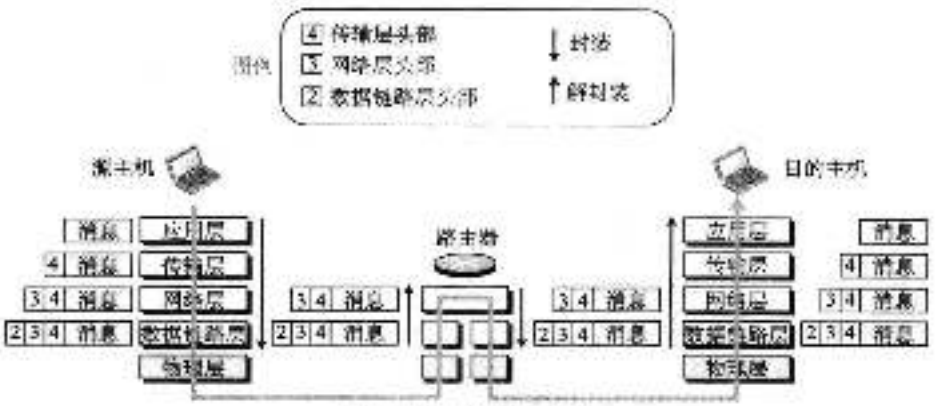


图 1-16 封装/解封装

由于在链路层交换机中没有封装/解封装发生，因此我们没有显示链路层交换机的层次。在图 1-16 中，我们显示了源主机中的封装、目的主机的解封装，以及路由器的封装和解封装。

源主机的封装

在源主机端，我们只进行封装。

1. 在应用层，交换的数据称为消息（message）。消息通常不包含任何头部和尾部，但是即使包含了这些，我们也将其整体称为消息。消息会被传递到传输层。

2. 传输层把这个消息作为有效载荷，该载荷是传输层应该关注的负载。传输层在有效载荷基础上增加传输层头部，其中包括了希望进行通信的源和目的应用程序的标识符和一些投递该消息需要的更多信息，例如进行流量控制、差错控制和拥塞控制需要的信息。其结果为一个传输层分组。该分组在 TCP 中称为段（segment），在 UDP 中称为用户数据报（user datagram）。然后传输层传递该分组到网络层。

3. 网络层把传输层分组作为数据或有效载荷，并且在该有效载荷上添加自己的头部。头部包含源和目的主机的地址，以及用于头部差错检查、分片的信息等其他一些信息。其结果为一个称为数据报（datagram）的网络层分组。然后，网络层传递这个分组到数据链路层。

4. 数据链路层把网络层分组作为数据或有效载荷，并且添加上自己的头部。该头部包含主机或下一跳步（路由器）的链路层地址。其结果为一个称为帧（frame）的链路层分组。该帧被传递到物理层进行传输。

路由器的解封装与封装

由于路由器连接两个或多个链路，因此在路由器中我们既需要进行解封装也需要进行封装。

1. 在比特流被投递到数据链路层后，这一层从帧中解封装出数据报并将它投递到网络层。

2. 网络层只检查数据报头部的源地址和目的地址，查阅它的转发表以寻找该数据报将被投递到的下一跳步。除非数据报太大以至于不能通过下一链路时需要对其进行分片，数据报的内容不应该被网络层改变。然后，数据报被传递到下一链路的数据链路层。

3. 下一链路的数据链路层将数据报封装成一个帧，将其传递到物理层进行传输。

目的主机的解封装

在目的主机端，每层都只解封装接收到的分组，移出有效载荷，并将有效载荷传递至较高一层，直到消息到达应用层。需要说明的是主机中的解封装包含差错检查。

地址

在 Internet 中，与协议分层相关的另一个概念是地址。正像以前讨论的那样，在这种模型中一对层次之间存在逻辑通信。包含两方的任意通信都需要两个地址：源地址和目的地址。尽管看起来我们似乎需要 5 对地址（每层一对），但是由于物理层不需要地址，我们通常只需要 4 对；物理层的数据交换单元是一个比特，它绝对没有地址。

图 1-17 显示了每一层的地址。

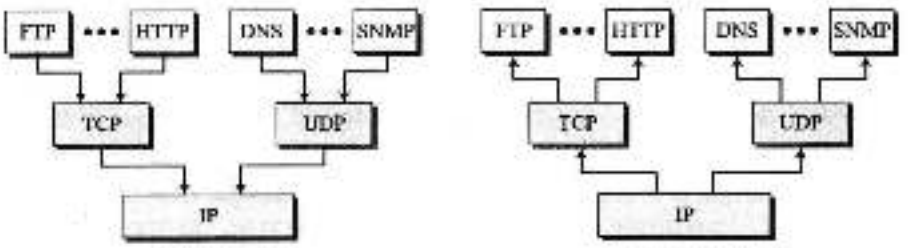


图 1-17 TCP/IP 协议簇中的地址

如图 1-17 所示，层次、地址与分组名之间存在一定的关系。在应用层，我们通常使用一个像 someorg.com 的名字定义提供服务的站点，或者使用像 somebody@coldmail.com 一样的电子邮件地址。在传输层，地址称为端口号，这些端口号指定源和目的地的应用层程序。端口号是本地地址，用于区分同一时间运行的几个程序。网络层地址是全局的，其范围涵盖了整个 Internet。链路层地址有时叫做 MAC 地址（MAC address），是本地定义的地址。每个链路层地址用于在网络（LAN 或 WAN）中定义一个特定的主机或路由器。在后面的章节中，我们将回过头来讨论这些地址。

多路复用与多路分解

由于 TCP/IP 协议簇在一些层次使用多个协议，因此我们在源端需要进行多路复用（multiplexing），在目的端需要进行多路分解（demultiplexing）。在这种情况下，多路复用的意思是一个协议能够封装来自多个上层协议的分组（一次一个）；多路分解的意思是一个协议能够进行解封装，并且将分组投递到多个上层协议（一次一个）。图 1-18 显示了一个高三层的多路复用与多路分解。



a. 源端的多路复用　　b. 目的端的多路分解

图 1-18 多路复用与多路分解

为了进行多路复用和多路分解，协议需要一个用于标识被封装的分组属于哪种协议的头部字段。在传输层，无论 UDP 还是 TCP 都可以接收多个应用层协议的消息。在网络层，IP 既可以接收来自 TCP 的段也可以接收来自 UDP 的用户数据报。同时，IP 也可以接收来自其他协议的分组，如

ICMP 协议、IGMP 协议等等。在数据链路层，数据帧可以携带来自 IP 或 ARP（参见第 5 章）等协议的有效载荷。

1.2.3 OSI 模型

虽然人们说起 Internet 都会谈到 TCP/IP 协议簇，但这个协议簇不是唯一被定义的协议簇。创建于 1947 年的国际标准化组织（International Organization for Standardization，ISO）由多个国家的成员组成，致力于世界范围内国际标准的制定。世界上将近四分之三的国家参加了 ISO 组织。开放系统互连（Open Systems Interconnection，OSI）模型是一个覆盖网络通信所有方面的 ISO 标准。它的第一次提出是在 20 世纪 70 年代末。

ISO 是一个组织；OSI 是一种模型。

开放系统（open system）是一个允许任意两个不同系统进行通信的协议集，无论这两个系统采用何种结构。OSI 模型的主要目标是给出在不改变底层硬件和软件逻辑的情况下，怎样更利于不同系统之间的通信。OSI 模型不是一种协议；它是理解和设计具有可扩展性、健壮性和互操作性网络结构的模型。OSI 模型的目的是作为创建 OSI 协议栈的基础。

OSI 模型是设计网络系统的层次化架构，它允许所有类型计算机系统之间通信。它包含 7 个隔离但又相关的层次，每一层定义通过网络传输信息的一部分工作（如图 1-19 所示）。



图 1-19 OSI 模型

OSI 与 TCP/IP

当比较 OSI 与 TCP/IP 时，我们发现 TCP/IP 协议簇没有会话层和表示层。在 OSI 模型发布后，这两层没有加入 TCP/IP 协议簇中。一般认为 TCP/IP 协议簇的应用层包含了 OSI 模型的 3 层，如图 1-20 所示。

对于这种决定，提到的原因有两点。首先，TCP/IP 存在一种以上的传输层协议。一些会话层的功能在一些传输层协议中已经存在。其次，应用层不只有一个软件，可以在这一层开发很多应用。如果一个特定的应用需要会话层和表示层的一些功能，那么可以在开发那个软件时包含进去。

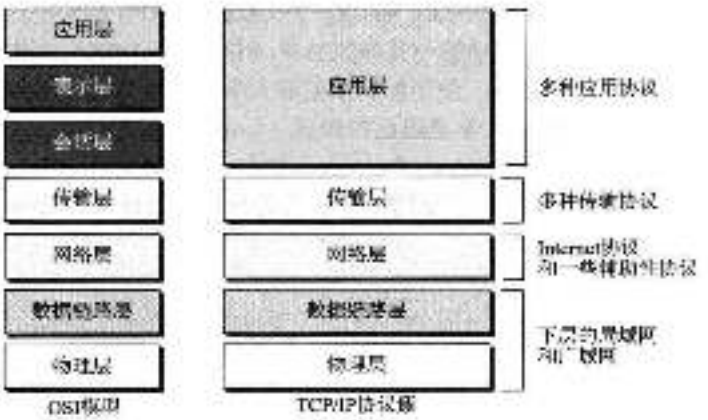


图 1-20 TCP/IP 与 OSI 模型

OSI 模型未成功的原因

OSI 模型在 TCP/IP 协议簇之后出现。多数专家起初都很振奋并认为 TCP/IP 协议将完全被 OSI

模型代替。这种事情没有发生有多种原因，我们只给出该领域所有专家都认同的3种原因。第一，OSI在TCP/IP已经完全部署后才完成，很多时间和金钱已经花费在TCP/IP协议簇上；改变它将化费大量的资金。第二，OSI模型中的一些层次从来没有完整定义过。例如，尽管表示层和会话层提供的服务列于文档中，但是这两层的实际协议既没有完整地定义也没有完整地描述，相应的软件也没有完整地开发出来。第三，当一个组织在不同的应用中实现OSI时，OSI没有表现出足够高的性能，以诱使Internet管理机构从TCP/IP协议簇转向OSI模型。

1.3 Internet发展史

我们已经给出了Internet及其协议的概况，现在简单介绍Internet的发展史。这个简单的历史可以使我们看到在不到40年的时间里，Internet怎样从一个私有网络演变成一个全球的网络。

1.3.1 早期历史

1960年之前存在一些通信网络，如电报网络和电话网络。这些网络适合于那个时代的恒定速率通信，即两个用户之间的连接建立之后，就可以传输编码的信息（电报）或声音（电话）。另一方面，计算机网络应该能够处理突发的数据，这意味着网络应能够在不同的时间按照不同的速率接收数据。整个世界正在等待分组交换网络的出现。

分组交换网络的诞生

1961年，MIT的Leonard Kleinrock首次提出了针对突发流量的分组交换网络。同一时期，兰德研究院（Rand Institute）的Paul Baran和英国国家物理实验室（National Physical Laboratory in England）的Donald Davies也发表了一些关于分组交换网络的文章。

ARPANET

20世纪60年代中期，研究机构中的大型机是独立的设备。不同厂家生产的计算机不能相互通信。美国国防部（DOD）高级研究计划署（Advanced Research Projects Agency，ARPA）对寻找连接计算机的方法产生了兴趣，其目的是使他们资助的研究者能够共享他们的发现，从而减少投资和降低重复劳动。

在1967年美国计算机协会（Association for Computing Machinery，ACM）的一次会议上，ARPA提出了组建一个连接计算机的小型网络——高级研究计划署网络（Advanced Research Projects Agency Network，ARPANET）的想法。按照这种想法，每台计算机（不必是同一生产厂家的计算机）都将附加一台称为接口信息处理机（interface message processor，IMP）的特定计算机。反过来，IMP将被相互连接在一起。每个IMP不但能够与其他的IMP通信，而且能够与它自己连接的主机通信。

1969年，ARPANET变成了现实。位于加利福尼亚大学洛杉矶分校（University of California at Los Angeles，UCLA）、加利福尼亚大学圣巴巴拉校区（University of California at Santa Barbara，UCSB）、斯坦福研究院（Stanford Research Institute，SRI）和犹他大学（University of Utah）的4个结点通过IMP连接在一起，形成了一个网络。一种称为网络控制协议（Network Control Protocol，NCP）的软件提供主机之间的通信。

1.3.2 Internet的诞生

1972年，ARPANET项目组的核心成员Vint Cerf和Bob Kahn开始合作开展所谓的网络互联项目（Internetting Project）。他们希望连接不相同的网络使得一个网络上的主机能够与另一个网络上的主机进行通信。需要克服的问题有很多：不同的分组大小、不同的接口类型、不同的传输速率，以及不同的可靠性需求。Cerf和Kahn提出利用被称为网关（gateway）的一种设备作为中间硬件，进行一个网络到另一个网络的数据传输。

TCP/IP

Cerf 和 Kahn 在 1973 年里程碑式的文章中描绘了实现端到端数据投递的协议。这是一个新版本的 NCP。这篇关于传输控制协议（TCP）的文章包括了封装、数据报、网关的功能等概念。其主要思想是把纠错功能从 IMP 移到了主机。ARPA 的这个 Internet 现在变成了通信领域关注的焦点。大约在这个时期，ARPANET 转交由防御通信署（Defense Communication Agency，DCA）负责。

1977 年 10 月，一个包含了 3 种不同网络（ARPANET、分组无线电网络、分组卫星网络）的互联网成功地展示在人们面前。现在，网络之间的通信变成了可能。

之后不久，官方决定将 TCP 分成两个协议：传输控制协议（Transmission Control Protocol，TCP）和网特网协议（Internet Protocol，IP）。IP 处理数据报的路由选择，TCP 负责高层次的功能，如分段、重组、检错。这个新的联合体变成了人们知道的 TCP/IP。

1981 年，按照与国防部的协议，UC 伯克利（UC Berkeley）将 UNIX 操作系统进行修改使其包含了 TCP/IP。包含有网络软件的流行操作系统对网络的普及起了很大的作用。伯克利 UNIX 的开放性（非厂商相关的）实现使厂商能够获得工作代码，并基于这些代码构建他们的产品。

1983 年，官方废除了初始的 ARPANET 协议，TCP/IP 变成了 ARPANET 的正式协议。如果人们希望使用 Internet 访问处于不同网络的计算机，那么他们将要运行 TCP/IP 协议。

MILNET

1983 年，ARPANET 分裂成两个网络：军队用户使用的军用网络（Military Network，MILNET）和非军队用户使用的 ARPANET。

CSNET

Internet 历史上的另一个里程碑是 1981 年创建的 CSNET。计算机科学网（Computer Science Network，CSNET）是由美国国家科学基金委（National Science Foundation，NSF）资助的一个网络。该网络是为由于缺乏与国防部的联系而不能加入 ARPANET 的大学设计的。CSNET 是一个造价不高的网络，它没有冗余的链路并且传输速率也较低。

20 世纪 80 年代中期，美国大部分具有计算机科学系的大学成为了 CSNET 的一部分。其他一些研究机构和公司也构建了他们自己的网络并利用 TCP/IP 进行互联。起初，Internet 一词与政府资助构建的连接网络相关，现在指利用 TCP/IP 协议连接的网络。

NSFNET

鉴于 CSNET 的成功，NSF 在 1986 年开始资助国家科学基金网络（National Science Foundation Network，NSFNET）。NSFNET 是一个连接美国 5 个超级计算机中心的主干网。由于这个 1.544Mbps 的 T-1 主干网允许社区网络接入，因此可以在全美范围内提供网络连接。1990 年，ARPANET 正式退出历史舞台并被 NSFNET 代替。1995 年 NSFNET 又恢复到它科学研究网络的初始理念。

ANSNET

1991 年，美国政府认为 NSFNET 已不能支持迅速增长的 Internet 流量。IBM、Merit 和 Verizon 三个公司填充了这段真空，组建了一个称为先进网络与服务（Advanced Network & Services，ANS）的非盈利组织，搭建了一个新的、高速 Internet 主干网。该主干网被称为高级网络服务网（Advanced Network Services Network，ANSNET）。

1.3.3 今天的 Internet

今天，我们见证了基础设施和新应用的迅速增长。今天的 Internet 是向全世界提供服务的信息港。使 Internet 变得如此流行的是新应用的发明和出现。

万维网

20 世纪 90 年代看到的 Internet 应用的急速增长得益于万维网（World Wide Web，WWW）的

出现。Web 是由欧洲原子核研究委员会（CERN）的 Tim Berners-Lee 发明的。这个发明增加了 Internet 的商业性应用。

多媒体

多媒体应用最近的发展，如 IP 语音（电话）、IP 视频（Skype）、视频共享（YouTube）和 IP 电视（PPLive）的出现，增加了网络用户的数量和用户在网时间。我们将在第 8 章讨论多媒体。

对等应用

对等（peer-to-peer，P2P）网络也是一个新的、具有很大潜力的通信领域。我们将在第 2 章介绍一些 P2P 应用。

1.4 标准和管理

在对 Internet 和它的协议的讨论中，我们经常看到一些参考标准或管理实体。在本节，我们为不熟悉这些标准和管理实体的读者介绍这些标准和管理实体；熟悉这些内容的读者可以跳过本节。

1.4.1 Internet 标准

Internet 标准是一个彻底通过测试的规范，该规范对从事与联网工作的人员非常有用。Internet 标准是一个必须遵循的正式的规则。经过严格的过程，一个规范才能达到 Internet 标准的某一状态。规范开始于 Internet 草案。一个 Internet 草案（Internet draft）是一个工作文档（该项工作正在进行中），没有官方的状态，只有 6 个月的生命周期。在 Internet 管理机构建议下，草案可以作为请求评论（Request for Comment，RFC）文档发布。每个 RFC 都经过编辑并被分配一个序号，所有感兴趣的团体均可获得。RFC 历经多个成熟阶段，并按照它们要求的级别分类到不同的类别。

成熟阶段

在一个 RFC 生命周期中，它会处于 6 个成熟阶段（maturity levels）之一：建议标准（proposed standard）、草案标准（draft standard）、Internet 标准（Internet standard）、历史的（historic）、实验性的（experimental）和信息性的（informational），如图 1-21 所示。

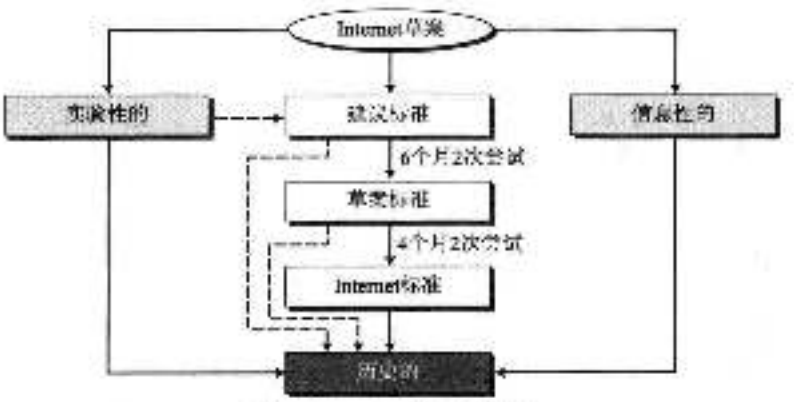


图 1-21 RFC 的成熟阶段

- **建议标准**。建议标准是一个稳定的、得到很好理解的、Internet 社会对其有足够兴趣的规范。在这个阶段，该规范通常被多个不同的工作组进行测试和实现。
- **草案标准**。在至少两个独立的和协作的实现之后，建议标准升级到草案标准。除非遇到困难，一个草案标准在遇到特定的问题进行修改之后，通常会变成 Internet 标准。
- **Internet 标准**。在成功的实现证实之后，一个草案标准就达到了 Internet 标准状态。
- **历史的**。从历史的观点看，历史的 RFC 非常重要。它们或者被后来的规范所取代，或者从

未通过必要的成熟阶段而变成 Internet 标准。

- **实验性的**。归类为实验性的 RFC 描述了与一种实验环境相关的工作，该实验环境不影响 Internet 的运行。这样一个 RFC 不应该在任何实用的 Internet 服务中实现。
- **信息性的**。归类为信息性的 RFC 包含了与 Internet 相关的通用的、历史的或指导性的信息。这类文章通常由非 Internet 组织的人员编写（如厂商）。

要求的级别

RFC 分为 5 个要求的级别（requirement level）：要求的（required）、推荐的（recommended）、可选的（elective）、限制使用的（limited use）和不推荐的（not recommended）。

- **要求的**。如果一个 RFC 必须被所有 Internet 系统实现，以获得最小的一致性，那么该 RFC 被标识为"*要求的*"。例如，IP（见第 4 章）和 ICMP（见第 4 章）都是要求的协议。
- **推荐的**。标识为"推荐的"RFC 不是为了获得最小一致性而必须要求的；推荐它是因为它有用。例如，FTP（见第 2 章）和 TELNET（见第 2 章）都是推荐的协议。
- **可选的**。标识为"可选的"RFC 既不要求必须实现也不是推荐的。但是为了自身的利益，系统可以使用它。
- **限制使用的**。标识为"限制使用的"RFC 仅仅应该在限制的环境下使用。多数实验性的 RFC 归类为这个级别。
- **不推荐的**。标识为"不推荐的"RFC 对通常的应用是不合适的。通常历史性的（遭到反对的）RFC 可能归类到这个级别。

RFC 文档可以在网站 http://www.rfc-editor.org 获得。

1.4.2 Internet 管理

起初以研究为主的 Internet 逐渐演化并赢得了以重要商业活动为主的大量用户。协调 Internet 问题的不同工作组引导了这种增长和发展，附录 D 给出了一些工作组的地址、e-mail 地址和电话号码。图 1-22 显示了 Internet 的管理组织。

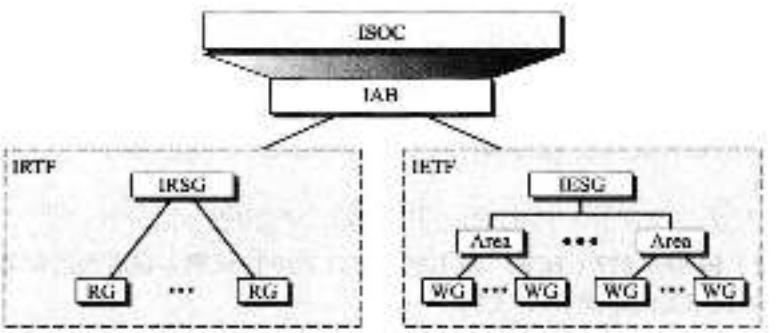


图 1-22 Internet 的管理

ISOC

成立于 1992 年的 Internet 协会（Internet Society，ISOC）是一个国际性、非盈利的组织，主要提供 Internet 标准处理过程的支持。该支持主要通过维护和支持其他 Internet 管理小组实现，如 IAB、IETF、IRTF 和 IANA（见下面的介绍）。ISOC 也负责推进与 Internet 相关的研究与其他学术活动。

IAB

Internet 体系结构委员会（Internet Architecture Board，IAB）是 ISOC 的技术顾问。IAB 的主要目标是关注 TCP/IP 协议簇的持续发展，作为技术顾问向 Internet 社会的研究成员提供服务。IAB

通过它的 Internet 工程任务组（Internet Engineering Task Force，IETF）和 Internet 研究任务组（Internet Research Task Force，IRTF）两个主要部门实现这个目标。IAB 的另一个责任是编辑和管理前面描述的 RFC 文档。IAB 也负责 Internet 与其他标准化组织和论坛的联系。

IETF

Internet 工程任务组（Internet Engineering Task Force，IETF）是一个由 Internet 工程指导组（Internet Engineering Steering Group，IESG）管理的一个工作组论坛。IETF 负责确认运行中的问题并提出解决方案。IETF 也开发和审查计划作为 Internet 标准的规范。工作组分为几个领域，每个领域集中精力于一个特定的内容。目前划分了 9 个领域，这些领域包括了应用、协议、路由、下一代网络（IPng）管理和安全。

IRTF

Internet 研究任务组（Internet Research Task Force，IRTF）是一个由 Internet 工程指导组（Internet Engineering Steering Group，IESG）管理的一个工作组论坛。IRTF 关注于与 Internet 协议、应用、结构和技术相关的、长期的研究课题。

IANA 和 ICANN

直到 1998 年 10 月，美国政府支持的 Internet 号码分配管理局（Internet Assigned Numbers Authority，IANA）负责 Internet 域名和地址的管理。之后，由一个国际委员会管理的非营利机构——Internet 名称与编号分配组织（Internet Corporation for Assigned Names and Numbers，ICANN）——承担了 IANA 的工作。

网络信息中心（NIC）

网络信息中心（Network Information Center，NIC）负责收集和发布有关 TCP/IP 的信息。

Internet 组织的地址和网站列于附录 B 中。

1.5 章末资料

推荐读物

有关本章更详细的内容讨论，我们推荐参阅下列书籍、网站和 RFC 文档。括号中的条目为本书末尾参考文献中的索引号。

书与文章

包括[Seg 98]、[Lei et al. 98]、[Kle 04]、[Cer 89]和[Jen et al. 86]在内的一些书和文章完整地覆盖了 Internet 的历史。

RFC 文档

RFC 791（IP）和 RFC 817（TCP）专门讨论了 TCP/IP 协议簇。在后面的章节中，我们将列出在每层中与每个协议相关的不同 RFC 文档。

小结

网络是一个由通信链路连接起来的设备集合。设备可以是计算机、打印机或其他具有发送和（或）接收网络中另一结点所产生的数据的设备。今天我们谈到的网络主要分为两种类型：局域网和广域网。目前，Internet 由很多广域网和局域网通过连接设备和交换站点连接而成。大部分希望连接 Internet 的最终用户需要利用 ISP 提供的服务。ISP 分为主干 ISP、区域 ISP 和本地 ISP。

协议是管理通信的规则集。在协议分层中，我们需要遵循两个原则以提供双向通信。首先，每一层需要实现两个相反的任务。其次，位于两端每层下的两个对象应该是等同的。TCP/IP 是一个由 5 个层次组成的层次化协议，这 5 层为应用层、传输层、网络层、数据链路层和物理层。

互联网的历史开始于20世纪60年代中期的ARPA网。Internet的诞生与Cerf和Kahn的工作，以及连接网络的网关出现有很大关系。Internet的管理随着Internet的发展不断演化。ISOC促进和发动了相关的研究和活动。IAB是ISOC的技术顾问组。IETF是负责运行问题的工作组论坛，IRTF为关注于长期发展研究课题的工作组论坛。ICANN负责Internet域名和地址的管理。NIC负责收集和发布有关TCP/IP协议的信息。

Internet标准是通过完全测试的规范。Internet草案为非官方的工作文档，具有6个月的生命周期。一个草案可能被作为RFC文档发布。RFC经过成熟阶段并按照要求级别分成不同的类别。

1.6 习题集

测试题

本章的交互式测试题请参见本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q1-1 局域网中利用一条公共电缆进行传输（如图1-1a所示）是不是一种广播（一对多的）传输？请解释。

Q1-2 在一个具有链路层交换机的局域网中（如图1-1b所示），主机1希望向主机3发送消息。由于通信需要通过链路层交换机，这个交换机需要拥有一个地址吗？请解释。

Q1-3 如果每个局域网都能够与其他局域网直接通信，那么连接n个局域网需要多少个点到点的广域网？

Q1-4 当我们使用本地电话与朋友通话时，我们使用的是电路交换网还是分组交换网？

Q1-5 当一个家庭用户利用拨号或DLS服务连接Internet时，电话公司承担什么角色？

Q1-6 为了进行双向通信，我们在这一章讨论的协议分层需要遵循的第一个原则是什么？

Q1-7 链路层交换机包含TCP/IP协议簇的那些层？

Q1-8 一个路由器连接3条链路（网络）。这个路由器包含以下列出的那些层？

a. 物理层　　b. 数据链路层　　c. 网络层

Q1-9 在TCP/IP协议簇中，当我们思考应用层的逻辑连接时，发送方和接收方的对等体是什么？

Q1-10 一台主机利用TCP/IP协议簇与另一台主机通信。在下面列出的层次中发送和接收的数据单元分别是什么？

a. 应用层　　b. 网络层　　c. 数据链路层

Q1-11 下列哪些数据单元被封装在帧中？

a. 用户数据报　　b. 数据报　　c. 段

Q1-12 下列哪些数据单元是从用户数据报中解封装出来的？

a. 数据报　　b. 段　　c. 消息

Q1-13 下列哪些数据单元具有应用层的消息和第4层的头部？

a. 帧　　b. 用户数据报　　c. 比特

Q1-14 列出本章提到的一些应用层协议。

Q1-15 如果一个端口号为16位（2个字节），那么TCP/IP协议簇中传输层的最小头部大小是多少？

Q1-16 下面列出的层次使用的地址（标识符）类型分别是什么？

a. 应用层　　b. 网络层　　c. 数据链路层

Q1-17 当我们说传输层多路复用和多路分解应用层消息时，我们的意思是不是说传输层能够把应用层的多个消息合并到一个数据分组中？请解释。

Q1-18 你能解释我们为什么没有提到应用层的多路复用/多路分解吗？

Q1-19 假设我们要把两台相互隔离的主机连接在一起，以使它们能够相互通信。在两台主机之间需要链路层交换机吗？

Q1-20 如果源主机和目的主机之间有一条单独的通道，在两台主机之间需要路由器吗？

Q1-21 解释Internet草案和建议标准的不同。

Q1-22 解释要求的 RFC 和推荐的 RFC 之间的不同。

Q1-23 解释 IETF 和 IRTF 承担的任务有什么不同。

思考题

P1-1 在图 1-10 中，当 Maria 向 Ann 发送信息时，回答下列问题：

a. 在 Maria 一端，第 1 层向第 2 层提供的服务是什么？

b. 在 Ann 一端，第 1 层向第 2 层提供的服务是什么？

P1-2 在图 1-10 中，当 Maria 向 Ann 发送信息时，回答下列问题：

a. 在 Maria 一端，第 2 层向第 3 层提供的服务是什么？

b. 在 Ann 一端，第 2 层向第 3 层提供的服务是什么？

P1-3 假设 2010 年连接 Internet 的主机数为 5 亿台。如果主机数按照每年 20%的速率增长，那么 2020 年的主机数是多少？

P1-4 假设一个系统采用 5 个协议层次，如果应用程序构建了一个 100 字节的消息，每一个层次（包括第 5 层和第 1 层）向数据单元增加 10 字节的头部，那么这个系统的效率（应用层字节数与传输字节数的比值）是多少？

P1-5 假设我们构建了一个分组交换互联网。我们需要利用 TCP/IP 协议簇传输一个巨大的文件。发送大分组的优势和劣势各是什么？

P1-6 将下列语句匹配到 TCP/IP 协议簇的一层或多层：

a. 路由判定

b. 连接到传输介质

c. 为最终用户提供服务

P1-7 将下列语句匹配到 TCP/IP 协议簇的一层或多层：

a. 构建用户数据报

b. 负责处理相邻结点间的帧

c. 把比特变换为电磁信号

P1-8 在图 1-18 中，当 IP 协议解封传输层分组时，它怎么知道这个分组应该投递到哪个上层协议（UDP 或 TCP）？

P1-9 假设一个私有互联网在数据链路层采用 3 个不同的协议（L1、L2 和 L3）。按照这种假设，重画图 1-18。是否可以这样说，在数据链路层我们在源结点进行多路分解，在目的结点进行多路复用？

P1-10 假设一个私有互联网要求加密/解密应用层消息，以保证其安全。如果我们需要增加一些关于加密/解密处理的信息（例如进行处理的算法），那么是不是意味着我们向 TCP/IP 协议簇添加了一层？如果你认为是这样，重画 TCP/IP 层次（图 1-12 的 b）。

P1-11 协议分层可以在我们生活的很多地方找到。想象你到度假胜地进行一个双程旅行。起飞前，你需要在你本地的机场进行一些处理工作。当你到达度假胜地机场后，你也需要进行一些处理工作。使用行李托运/行李提取、登机/下机、起飞/着陆等层次，对双程旅行进行协议分层。

P1-12 在图 1-4 中，从西海岸的一台主机到东海岸的一台主机仅有一条单一的通路。在这个互联网中，我们为什么需要两台路由器？

P1-13 在今天的 Internet 中，数据表示变得越来越重要。一些人主张 TCP/IP 协议簇应该增加一个新的层次以负责数据表示（参见附录 C）。如果将来增加这个新的层次，那么它应该处于协议簇的什么位置？重画图 1-12 以包含这个层次。

P1-14 在一个互联网中，我们用新的局域网技术替换原有的局域网技术。在 TCP/IP 协议簇中，哪些层次需要改变？

P1-15 假设一个应用层协议利用 UDP 提供的服务进行编写，那么这个应用层协议不改变就可以使用 TCP 服务吗？

P1-16 利用图 1-4 显示的互联网，描述当西海岸的一台主机与东海岸的一台主机交换信息时，TCP/IP 协议簇的层次和数据流。

1.7 模拟实验

Applets

一种查看实际的网络协议和观察一些示例解决方案的方法是利用交互式的动画。我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些协议。

实验作业

进行网络和网络设备相关的实验至少可以采用两种方法。在第一种方法中，我们可以构建一个隔离的网络实验室，利用网络硬件和软件模拟在每章讨论的内容。我们可以构建一个互联网，从一台主机发送信息到另一台主机，从而观察分组流并测量其性能。尽管这种方案比第二种方法更有效、更适宜于教学，但是这种方案实现起来比较昂贵，不是所有的单位都能够投资这样一个独立的实验室。

在第二种方法中，我们可以使用 Internet 世界上最大的网络，作为我们的虚拟实验室。我们可以利用 Internet 发送和接收分组。一些免费的、可以下载的软件允许我们捕获和观察交换的数据分组。我们可以分析这些分组以理解网络的理论概念是如何付诸实现的。尽管第二种方法不能控制和改变分组的路由以观察 Internet 的行为，效果不如第一种方法明显，但是这种方法实现起来比较廉价。它不需要实际的实验室，利用我们的桌面机或笔记本就可以实现。同时，需要的软件也可以免费下载。

很多 Windows 和 UNIX 操作系统下的程序和工具允许我们探测、捕获、跟踪和分析我们的电脑和 Internet 之间交换的数据分组。一些程序和工具，如 WireShark 和 Ping Plotter，具有图形用户接口（GUI）；其他如 traceroute、nslookup、dig、ipconfig 和 ifconfig 为命令行式的网络管理工具。这些工具对网络管理员调试网络、学生学习网络都非常有价值。

在本书中，尽管我们偏爱采用其他工具，但是大部分实验作业使用的是 Wireshark。Wireshark 从一个网络接口捕获活动的分组数据并对详细的协议信息进行显示。但是，Wireshark 是一种被动的分析器。它仅仅能“计量”网络上的东西但不能操纵它们：它既不能在网络上发送数据分组也不能做其他主动的操作。另外，Wireshark 也不是一种入侵检测工具。它不会对网络入侵发出警告。但是，它能帮助网络管理员或者网络安全工程师理解网络内部的状况，帮助他们解决网络故障。Wireshark 除了对网络管理员和安全工程师必不可少之外，它对协议开发人员也非常有价值，协议开发人员可以使用 Wireshark 对实现的协议进行调试。与此同时，Wireshark 也是一个很好的教学工具，学习计算机网络的学生能够使用它实时地观察协议操作的细节。

在本节实验作业中，我们学习如何下载和安装 Wireshark。下载和安装指南在本书网站的第 1 章实验部分给出。在这个文档中，我们还讨论了该软件背后的基本想法、它的窗口格式，以及怎样使用它。这个实验的学习可以使学生为今后完成采用 Wireshark 的实验作业做好准备。

第2章

Computer Networks: A Top-Down Approach

应 用 层

整个因特网、硬件以及软件的设计和开发就是为应用层提供服务。TCP/IP 协议簇的第五层正是这些服务的所在位置。其他四层协议使这些服务成为可能。学习因特网技术的一种方法就是先解释应用层提供的服务，然后再展示其他四层是如何支持这些服务的。由于本书正是依照这种方法，因此应用层是我们首先要讨论的内容。

在因特网的发展历程中，创造和使用了许多应用协议。有些是有特定用途但从未成为标准的，有些已经被弃用，有些被修改或者被新的协议所替换。一些协议得以幸存下来并成为标准应用。新的应用协议被持续不断地加入因特网中。

本章分 5 节讨论应用层。

- 2.1 节将介绍因特网提供的服务的本质以及两个应用类型：传统类型即客户-服务器模式（client-server paradigm），以及新类型即对等模式（peer-to-peer paradigm）。
- 2.2 节讨论客户-服务器模式的概念以及这个模式是如何为因特网用户提供服务的。
- 2.3 节讨论一些客户-服务器模式的预定义和标准应用。我们也会讨论一些流行的应用，比如万维网、文件传输、电子邮件等等。
- 2.4 节讨论对等模式中的概念及协议。我们会介绍一些协议，诸如 Chord、Pastry 和 Kademilia。我们也会提及一些使用这些协议的流行应用。
- 2.5 节我们给出在客户-服务器模式下如何通过用 C 语言编写两个程序创建一个新的应用。这两个程序一个是为客户端编写的，另一个是为服务器编写的。在第 11 章我们将展示如何用 Java 语言编写客户-服务器程序。

2.1 介绍

应用层为用户提供服务，通信是由逻辑连接提供的，这意味着两个应用层假设存在一个假想的直接连接，通过这个连接可以发送和接收报文。图 2-1 展示了这种逻辑连接背后的思想。

图 2-1 展示了这样一幅场景，一个科学家在名为天空研究所的研究公司工作，她需要从网络书商那里订购一本与自己研究相关的书。一条逻辑连接在天空研究所电脑的应用层与科技书作服务器的应用层之间建立了。我们把第一台主机叫 Alice，把第二台主机叫 Bob。应用层的通信是逻辑的，而不是物理的。Alice 和 Bob 认为他们之间有一条双向逻辑信道，他们可以通过这条信道发送接收报文。然而，实际的通信通过了若干设备（Alice、R2、R4、R5、R7 以及 Bob）以及图上所示的若干物理信道。

2.1.1 提供服务

在因特网出现之前就开始运营的所有通信网络都被设计成向网络用户提供服务。然而，这些网络大都原先被设计为提供一种特定服务。比如电话网络原先被设计为提供语音服务，它允许全世界的人们相互交谈。然而之后，这个网络用于了其他服务，比如传真，用户在两个终端加上一些额外的硬件可以收发传真了。

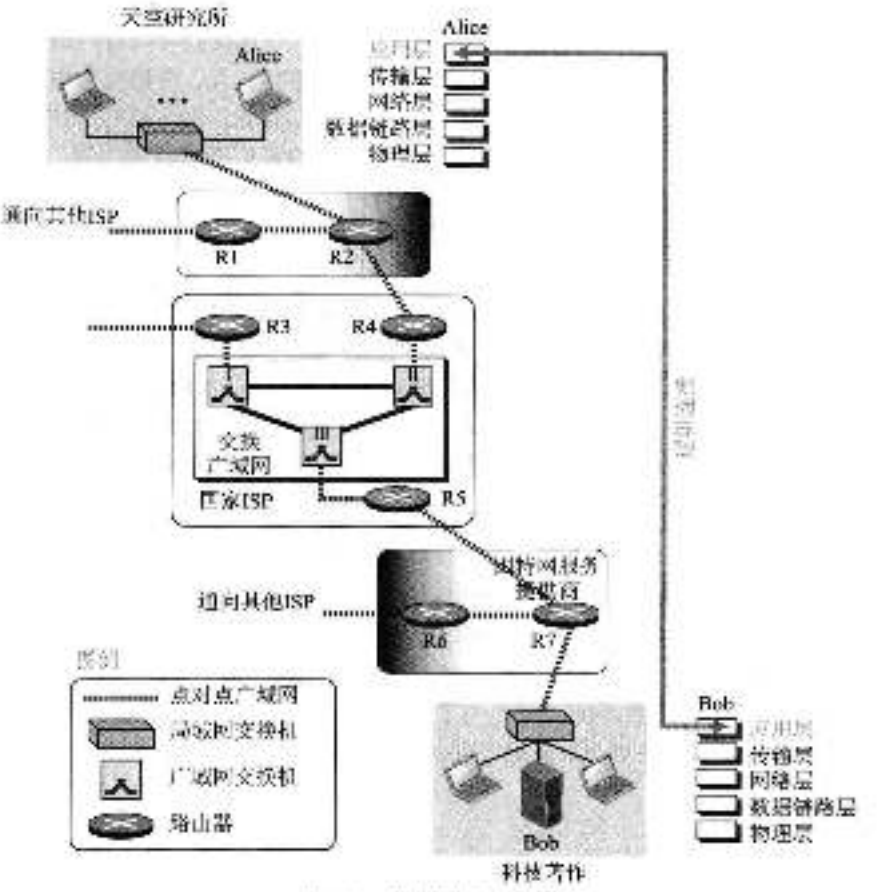


图 2-1 应用层逻辑连接

因特网原先是为同一个目的而设计出来的，即为全世界的用户提供服务。然而 TCP/IP 协议簇的层次结构使得因特网比其他网络更加灵活，诸如邮件网络和电话网络。协议簇的每一层原先由一个或多个协议组成，但是可以加入新的协议，因特网管理机构可以删除或替换某些协议。然而如果将一个协议增加到一层中，那么这个协议应当被设计成使用底层协议提供的服务。如果从一层中去除一个协议，那么应该注意去改写它的上一层协议，高层协议可能使用了它提供的服务。

然而由于应用层是协议簇的最高层，它与其他层有些不同。这层中的协议不为任何其他协议提供服务，它们只接收来自传输层协议的服务。这意味着，可以从这层中轻易地去除协议。只要新的协议可以使用传输层协议提供的服务，那么就可以把它加入到这一层。

由于应用层是唯一向因特网用户提供服务的层次，因此如上所述，应用层的灵活性允许新的应用协议轻松地加入因特网，这一点在因特网的发展历程中不断发生。当因特网被创建时，只有很少的应用协议可以供用户使用。但是现在，我们无法给出这些协议的数目，因为新的协议正在源源不断地被添加进去。

标准和非标准协议

为了使因特网流畅运作，需要标准化和归档 TCP/IP 协议簇前四层所使用的协议。它们通常成为包的一部分，包含在如 Windows 或 UNIX 的操作系统里。然而为了灵活，应用层协议既可以标准化也可以非标准化。

标准应用层协议

有一些应用层协议已经被因特网管理机构标准化和归档，并且我们与因特网的日常交流中正在使用它们。每个标准协议是一对程序，它们与用户和传输层进行交互，传输层为用户提供特定的服务。在本章稍后我们会讨论一些标准应用，某些会在其他章节讨论。关于这些应用协议，我们需要知道它们提供什么服务类型，它们如何工作，以及这些应用给我们哪些选项等等。学习这些协议可以使网络管理员更容易解决使用过程中的问题。对这些协议工作方式的深入理解也将会使我们了解如何创建新的非标准协议。

非标准应用层协议

如果一个程序员能编写两个程序，那么他就可以创建非标准应用层程序，这两个程序通过与传输层交互为用户提供服务。本章的后面我们将给出如何编写这样的程序。如果私人使用的话，创建一个非标准（专利的）协议甚至不需要因特网管理机构的批准，这使得因特网在世界上十分流行。一个私人公司可以创建一种新的定制应用协议，来和遍布全球的办公室进行通信，公司使用 TCP/IP 协议簇前四层提供的服务而不使用任何一个标准应用程序。所需要的就是以一种计算机语言来编写程序，这些程序使用传输层协议提供的服务。

2.1.2 应用层模式

应该弄清楚的是，为了使用因特网，我们需要两个应用程序彼此交互：一个运行在世界某个地方的电脑上，另一个运行在世界其他地方的另一台电脑上。两个程序需要通过因特网基础设施彼此发送报文。然而，我们还没有讨论这两个程序之间的关系。两者都应该能够请求和提供服务吗？抑或应用程序仅仅实现这两种功能中的一个？为了回答这个问题，在因特网的发展历程中开发了两种模式：客户-服务器模式和对等模式。此处，我们简要介绍这两种模式，但是我们会在稍后讨论它们的细节。

传统模式：客户-服务器

传统模式称为**客户-服务器模式**。在几年前它还是最流行的。在这种模式中，服务提供者是一个称为服务进程的应用程序，它不断地运行着，等待另一个称为客户进程的应用程序通过因特网建立连接并请求服务。通常有一些服务进程可以提供特定类型的服务，但是有很多客户向这些服务进程请求服务。服务进程必须一直运行，当需要接受服务时客户进程就被打开。

客户-服务器模式与某些因特网领域外的服务类似。比如，任何地方的电话号码查询中心都可以被看做服务器，一个打电话询问特定电话号码的用户可以被看做客户。电话号码查询中心必须每时每刻准备提供服务，当需要服务时用户可以给中心致电一小段时间。

尽管客户-服务器模式的通信是在两个应用程序之间的，但是每个程序的角色是全然不同的。换言之，我们不能把一个客户端程序当做服务器程序运行，反之亦然。在本章的后面，当谈论在这种模式下的客户-服务器编程时，我们总是要分别为这两种服务类型编写应用程序。图 2-2 展示了一个客户-服务器通信的例子，其中二个客户与一个服务器进行通信，客户接受服务器提供的服务。

这个模式的问题是通信负荷集中在服务器上，这意味着服务器应该是一台强大的计算机。但是，即使是强大的计算机也会难以应对大量客户同时尝试连接。另一个问题是应该存在一个服务提供商，它乐于接受这项花费并创建一台提供特定服务的强大服务器，这意味着服务必须为服务器产生收益，以此便可促进这种安排。

很多传统该服务仍然在使用这种模式，包括万维网（World Wide Web，WWW）以及它的传播媒介：超文本传输协议（HyperText Transfer Protocol，HTTP）、文件传输协议（File Transfer Protocol，FTP）、安全人机界面（Secure Shell，SSH）、电子邮件等等。我们在本章后面讨论这些协议和应用。

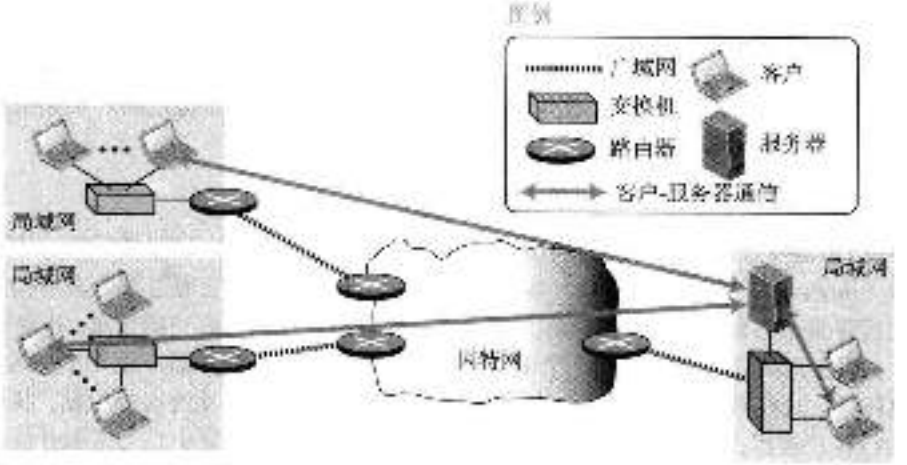


图 2-2 客户/服务器模式示例

新模式：对等

一个称为对等模式（通常简称为 P2P 模式）的新模式已经出现，它适合了新应用的需求。在这种模式下，不需要一个不断运行且等待客户进程连接的服务器进程。责任在对等结点（peer）之间分担。连接到因特网的计算机可以在这一次提供服务却在下一次接受服务。一台计算机甚至可以同时接受和提供服务。图 2-3 展示了这种模式的通信例子。

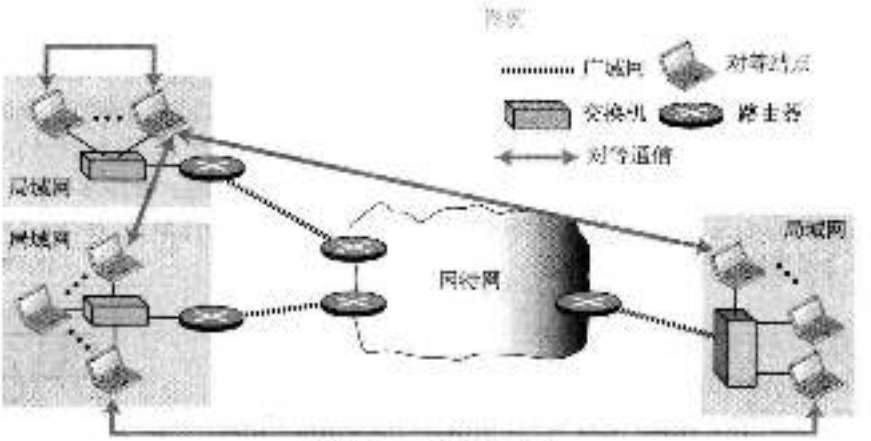


图 2-3 对等模式示例

一个真正符合这个模式的领域是网络电话。电话通信确实是对等活动，没有一方需要不断运行来等待另一方呼叫。当某些计算机有东西要彼此共享而连到因特网上时，也会使用到对等模式。比如，如果某因特网用户有一个可以和其他用户共享的文件，那么这个用户没有必要去建立服务器并一直运行服务器进程等待其他用户连接并获取文件。

由于对等模式无需一台运行和维护昂贵的服务器，它是容易扩展且经济划算的。尽管如此，还是存在一些挑战的。主要挑战就是安全问题，在分布式服务之间创建安全通信比在那些由专用服务器控制的服务之间建立安全通信要更困难。另一个挑战就是适用性，似乎并不是所有的应用都可以使用这个新模式。比如倘若某一天网络可以作为对等服务执行，并不会有非常多的因特网用户准备参与进来。

有一些新的应用使用这种模式，诸如 BitTorrent、Skype、IPTV 以及网络电话。我们将在后面

讨论其中的一些应用，对另一些应用的讨论将放到后续的章节。

混合模式

一个应用可以通过结合这两种模式的优点来把这二者混合起来。比如经层级的客户-服务器通信可以用来寻找可以提供服务的对等结点（peer）的地址。当找到这个地址时，实际服务可以通过使用对等模式从对等结点中获得。

2.2 客户-服务器模式

在客户-服务器模式中，应用层的通信是在两个运行着的应用程序之间进行的，这两个应用程序称为进程（process）：客户和服务器。客户是一个运行着的程序，它通过发送请求初始化通信；另一个应用程序是服务器，它等待来自客户的请求。服务器处理来自客户的请求，准备结果并将其发送给客户。服务器的定义意味着当一个来自客户的请求到达时，服务器必须是正在运行的，但是客户不必这样，它只在必要的时候运行。这意味着如果我们有两台相互连接的电脑，我们可以在一台电脑上运行客户进程，在另一台上运行服务器进程。然而，我们需要小心的是服务器进程要在客户端程序运行前开启。换言之，服务器的生存期是无限的：它应该开启后一直运行，等待客户。客户的生存期是有限的：它通常发送有限的请求给对应的服务器，接收响应然后停止。

2.2.1 应用程序接口

客户进程是如何与服务器进程进行通信的？一个计算机程序通常是由预定义了指令集的计算机语言编写的，这个指令集告诉计算机要做什么。计算机语言有一个数学操作指令集、一个字符串处理指令集、一个输入/输出访问指令集等。如果我们需要一个进程与另一个进程通信，那么我们就需要一个新的指令集告知 TCP/IP 协议簇的低四层打开连接，发送数据，从另一个终端接收数据，以及关闭连接。这样的指令集通常称为应用程序接口（Application Programming Interface，API）。程序中的接口是两个实体之间的指令集。在这种情况下，一个实体是应用层中的进程，另一个是操作系统。操作系统封装了 TCP/IP 协议簇的前四层。换句话说，电脑制造商将协议簇的前四层编写近操作系统中并包含了 API。这样，当通过因特网发送和接收分组时，应用层运行的进程才能够与操作系统通信。有许多通信 API 被设计出来。其中三个是很常见的：套接字接口、传输层接口（Transport Layer Interface，TLI）以及 STREAM。在这一节里，我们仅需要讨论最常见的套接字接口，以给出一个应用层网络通信的宏观概念。

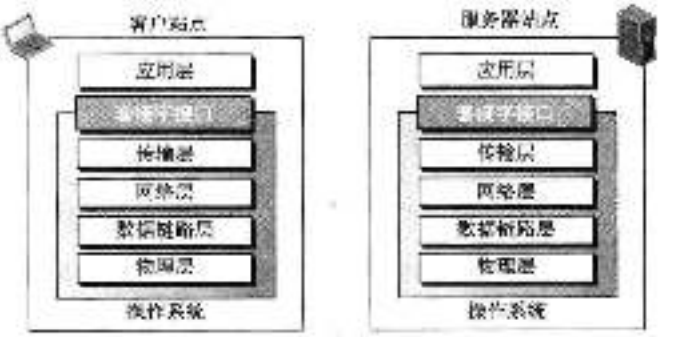


图 2-4 套接字接口的位置

在 20 世纪 80 年代，套接字接口作为 UNIX 环境的一部分出现在加州伯克利大学。如图 2-4 所示，套接字接口是提供应用层和操作系统间通信的指令集，是一个可以被某进程用来与另一个进程进行通信的指令集。

套接字的概念允许我们使用编程语言中对其他信源和信宿设计的所有指令的集合。比如，在绝大多数计算机语言中，如 C、C++以及 Java，已经有很多可以从信源读数据和向信宿写数据的指令。这些信源和信宿有：键盘（信源）、监视器（信宿）以及文件（信源和信宿）。我们可以使用相同的指令从套接字里读或向套接字里写入。换言之，我们只不过在向编程语言中加入新的信源和信宿，而没有改变发送和接收数据的方式。图 2-5 展示了这种思想并将套接字与信源和信宿进行比较。

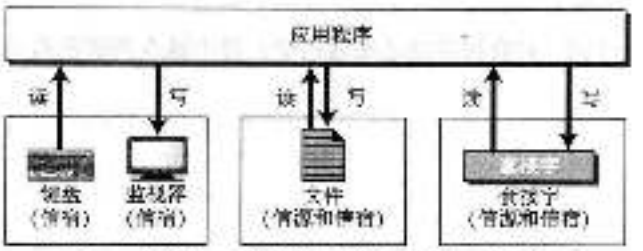


图 2-5 套接字和其他信源和信宿采用相同的方式

套接字

尽管套接字在行为上应该和一个终端或文件类似，但是它不是物理实体，而是一种抽象。套接字是供应用程序创建和使用的数据结构。

我们可以说，就应用层而言，客户进程和服务器进程间的通信是两个套接字间的通信。如图 2-6 所示，在两个终端创建了两者间的通信。客户认为套接字是接收请求和发出响应的实体；服务器认为套接字是发出请求并且需要获得响应的实体。如果我们创建两个套接字，一端创建一个，并且正确定义源端和目的端地址，那么我们就可以使用指令去发送和接收数据了。其余就是操作系统以及嵌入的 TCP/IP 协议的工作了。

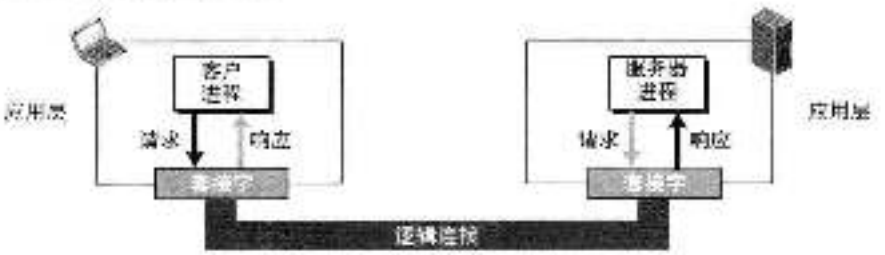


图 2-6 对等通信中套接字的使用

套接字地址

客户和服务器的交互是双向通信。在双向通信中，我们需要一对地址：本地地址（发送端）和远程地址（接收端）。在一个方向上的本地地址对另一个方向来说就是远程地址，反之亦然。由于客户-服务器模式的通信是在套接字之间的，我们需要一对套接字地址（socket address）：一个本地套接字地址和一个远程套接字地址。然而，我们需要以 TCP/IP 协议簇的标识符来定义套接字地址。

一个套接字地址首先定义了一个客户或服务器所在的计算机。正如我们将在第 4 章讨论的，因特网上的一台计算机由 IP 地址唯一确定，IP 地址在现在的因特网版本中是一个 32 位的整数。然而，可能在同一时间同一台计算机上有很多客户或服务器进程运行，这意味着我们需要另一个标识符来定义特定的通信中所涉及的客户或服务器。正如我们将在第 3 章讨论的，一个应用程序可以由端口号定义，它是一个 16 位整数。这意味着套接字地址应该是一个 IP 地址和一个端口号的组合，如图 2-7 所示。

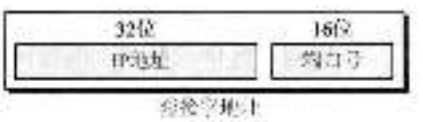


图 2-7 套接字地址

由于套接字定义了通信终端，我们可以说套接字是由一对套接字地址标识的。这一对套接字地址分别是本地套接字地址和远程套接字地址。

例 2.1 在电话通信中我们可以找到两级地址。一个电话号码可以定义一个组织，电话分机号码可以定义组织内的一个特定连接。这样，电话号码就像 IP 地址一样定义了整个组织；分机号码就像端口号，它定义了特定的连接。

寻找套接字地址

客户或服务器如何寻找一对套接字地址来通信呢？每个站点的情况是不同的。

服务器站点

服务器需要一个本地（服务器）和一个远程（客户）套接字地址来通信。

本地套接字地址 本地（服务器）套接字地址由操作系统提供。操作系统知道运行着服务器进程的计算机的 IP 地址。然而服务器进程的端口号需要被分配。如果这个服务器进程是因特网管理结构定义的标准进程，那么端口号就已经分配好了。比如，超文本传输协议（HTTP）被分配的端口号是 80。其他进程就不能再使用了。我们将在第 3 章讨论这些熟知的端口号。如果服务器进程不是标准进程，那么它的设计者就要在规定范围内选择一个端口号，并分配给进程。当服务器开始运行时，它就得知了本地套接字地址。

远程套接字地址 对服务器来说，远程套接字地址是建立连接的客户套接字地址。由于服务器可以给多个用户提供服务，它事先并不知道远程套接字地址。当客户试图连接服务器时，服务器可以知道这个套接字地址。客户套接字地址包含在发送给服务器的请求报文中，它成为远程套接字地址来给客户提供响应。换言之，尽管服务器的本地套接字地址是固定的并且在生存期内一直使用，但是远程套接字地址在服务器与不同客户进行交互时都会改变。

客户站点

客户也需要一个本地（客户）和一个远程（服务器）套接字地址来通信。

本地套接字地址 本地（客户）套接字地址也由操作系统提供。操作系统知道运行着客户进程的计算机的 IP 地址。然而端口号是每次客户进程需要开始通信时分配给客户进程的一个临时 16 位整数。但是端口号需要从一组由因特网管理机构定义的整数中分配，这称为临时端口号，我们将在第 3 章深入探讨。操作系统需要确保新的端口号没有被其他正在运行的客户进程所占用。

远程套接字地址 然而找到远程（服务器）套接字地址需要更多的工作。当一个客户进程开启时，它应该知道自己想要连接到的服务器的套接字地址。这里有两种情况。

- 有时，开启客户进程的用户知道运行着服务器进程的计算机的端口号和 IP 地址。这通常在我们编写客户和服务器应用并进行测试时发生。比如在本章的结尾，我们将编写一个简单的客户-服务器程序，并且我们将采用此种方式进行测试。在这种情况下，当运行客户端程序时程序员可以提供这两条信息。
- 尽管每个标准应用都有一个熟知端口号，但绝大多数情况下我们不知道 IP 地址。这会在如下情景下发生：连接网页、给朋友发送电子邮件以及从一个远程站点拷贝文件等。在这些情况下，服务器有一个名称，一个唯一标识服务器进程的标识符。例如 URL 就是这种标识符，像是 www.xxx.yy 或者电子邮件地址 xxxx@yyyy.com。客户进程现在需要将这个标识符（名称）改成对应的服务器套接字地址。由于端口号应该是一个熟知端口号，因此客户进程通常知道端口号。IP 地址可以通过使用另外一个客户-服务器应用来获得，这个应用叫做域名系统（Domain Name System，DNS）。稍后我们会在本章讨论 DNS，但是我们知道它工作起来就像因特网中的电话簿就够了。将此情景与电话簿相比较。我们想给某个已知姓名的人打电话，但是那个人的电话号码可以从电话簿上得到。电话簿将姓名映射到电话号码；DNS 将服务器名称映射到运行着那个服务器的计算机 IP 地址上。

2.2.2 使用传输层的服务

一对进程间因特网中的用户提供服务，这些用户可以是人，也可以是程序。但是由于应用层没有物理通信，这一对进程需要使用传输层提供的服务来通信。正如我们在第 1 章简要讨论的，在 TCP/IP 协议族中有三个常见的传输层协议：UDP、TCP 以及 SCTP，这些会在第 3 章详细讨论。绝

大多数标准应用被设计来使用这些协议。当我们编写一个新的应用时我们可以决定使用哪个协议。对于传输层协议的选择将严重影响应用进程的性能。在这一节，我们首先讨论每个协议提供的服务，来帮助大家理解为什么一个标准应用会使用它，以及编写一个新应用时需要使用哪个协议。

UDP 协议

UDP 提供了无连接的、不可靠的数据包服务。无连接服务意味着两个交换报文的终端之间没有逻辑连接。每个报文都是独立的实体，它被封装在一个称为数据报（datagram）的分组中。UDP 看不到来自同一个源端并去往同一个目的端的数据报之间的关系（连接）。

UDP 是不可靠的协议。尽管它可能在传输中检查数据是否被破坏，但是它并不要求发送端重传被破坏的或丢失的数据。对于某些应用，UDP 有一个优势，即它是面向报文的。它保留报文边界。

我们可以将无连接、不可靠的服务与邮局提供的常规服务进行对比。两个实体可以在它们之间交换信件，但是邮局并没有看见这些信件之间的任何连接。对于邮局，每个信件都是带有它自己的发送者和接收者的独立实体。尽管邮局是尽力而为的，但是如果一个邮件在发送过程中丢失或被损坏，邮局概不负责。

如果应用程序发送小报文，并且简单性和速度要比可靠性更重要，那么可以将这个应用程序设计成使用 UDP 协议的程序。比如，某些管理和多媒体应用符合这个分类。

TCP 协议

TCP 提供面向连接的可靠的字节流传输。TCP 要求两个终端首先通过交换一些连接建立分组（connection-establishment packet）来建立一个逻辑连接。这个阶段有时称为握手，它设定了两个终端间的某些参数。这些参数包括要交换的数据分组大小、用于保存数据直到整个报文全部到达的缓冲区的大小等等。在握手过程后，两个终端可以向着彼此的方向以报文段形式发送数据块。通过计算交换的字节数，可以检测字节的连续性。比如，如果某些字节丢失或损坏了，接收端可以请求重发这些字节，这使得 TCP 成为一个可靠协议。我们将在第 3 章看到 TCP 也可以提供流量控制和拥塞控制。TCP 协议的一个问题是它不是面向报文的，它不保留报文边界。

我们可以将 TCP 提供的面向连接的可靠的服务与电话公司的服务进行比较，尽管这种比较仅仅是在某种程度上进行的。如果两方决定通过电话而不是邮局通信，他们可以创建一次连接，进行一段时间的通话，电话服务在某种程度上是可靠的，因为如果一个人没听明白或听不清另一方所说的话，他可以要求对方再说一遍。

绝大多数需要发送长报文以及要求可靠性的标准应用都从 TCP 的服务中受益。

SCTP 协议

SCTP 提供了前面两个协议组合的功能。就像 TCP 一样，SCTP 提供了面向连接的可靠的服务，但是它不是面向字节流，它是像 UDP 一样面向报文的。除此之外，SCTP 可以通过提供多媒体网络层连接提供多媒体流服务。

SCTP 通常适用于那些不但需要可靠性，而且即使网络层连接发生错误也需保持连接不断开的应用。

2.3 标准客户–服务器应用

在因特网的发展历程中，很多客户–服务器应用被开发出来。我们没有必要重定义它们，但是我们需要理解它们做了什么，对于每一个应用，我们也需要知道有哪些可用选项。学习这些应用以及了解它们提供不同服务的方式可以帮助我们将来创建定制的应用。

我们在这一节选择了六个标准应用。我们从 HTTP 和万维网开始，因为几乎所有的网络用户都使用它们。然后，我们介绍文件传输和电子邮件应用，这些在因特网上占很大的流量。接下来，我们解释远程登录以及如何通过 TELNET 和 SSH 协议实现远程登录。最后，我们讨论 DNS，所有应

用程序都使用它来将应用层标识符映射到相应的主机 IP 地址上。

其他章节将会适当讨论一些其他的应用，如动态主机配置协议（DHCP）和简单网络管理协议（SNMP）。

2.3.1 万维网和 HTTP

在这一节，我们首先介绍万维网（简称 WWW 或 Web），之后我们讨论超文本传输协议（HTTP），它是与 Web 相关的最常见的客户-服务器应用程序。

万维网

Web 的思想最早由 Tim Berners-Lee 在 1989 年于 CERN[2] 提出的，CERN 是欧洲原子研究中心（European Organization for Nuclear Research）的简称。它允许多个研究者在欧洲的不同地点访问彼此的研究。商业化的 Web 出现在 20 世纪 90 年代早期。

今天的 Web 是信息宝库，其中称为网页的文档在全世界分布，并且相关的文档链接在一起。Web 的流行和成长与前面介绍过的两个术语有关：分布式的（distributed）和链接的（linked）。分布式允许 Web 增长。世界上每个 Web 服务器都可以增加一个新的网页到这个宝库中并向所有因特网用户宣告，而这不会使一些服务器超载。链接使得一个网页与另一个存储在世界某个地方的主机上的网页相互引用。网页的链接通过使用一个称为超文本（hypertext）的概念而实现，这个概念在因特网出现之前很多年就被引入进来了。这个思想是当一个链接出现在文档中时，利用一台机器自动获取存储在系统中的另一个文档。Web 用电子方式实现了这个思想：当用户点击这个链接时允许获取被链接的文档。现在超文本这个术语的含义已经由一开始的被链接的文本文档变成了超媒体（hypermedia），这表示网页可以是文本文件、图片、音频文件或视频文件。

Web 的用途已经超越了简单获取被链接的文件。现在，它用于提供电子购物和游戏。一个用户可以在任何时候使用网络来收听广播节目或收看电视节目，而不必只有在这些节目播出的时候才能收听或收看。

结构

如今 WWW 是一个分布式客户-服务器服务。使用浏览器的用户可以访问一个正在服务器上运行的服务。然而服务是分布在很多称为站点（site）的地点上。每一个站点有一个或多个文档，它们称为网页。但是，每个网页（web page）可以包含一些到其他网页的链接，那些被链接的网页可以在同一个站点也可以在其他站点。换言之，一个网页可以是简单的也可以是复合的。简单网页没有链接；复合网页有一个或多个链接。每个网页是一个有名字和地址的文件。

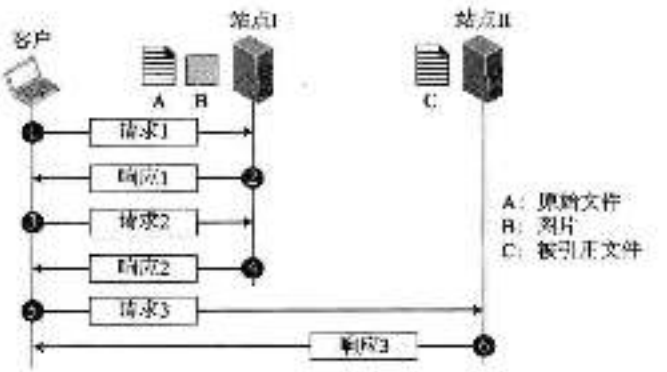


图 2-8 例 2.2

例 2.2 假设我们需要获取一个科学文档，这个文档包含了到另一个文本文件和一幅大图片的引用。图 2-8 说明这个场景。

主文档和图片存储在同一个站点的两个不同文件中（文件 A 和文件 B）；被引用的文本文件被存储在另一个站点（文件 C）。由于我们正在处理三个不同的文件，如果想看到全部文档就需要三项事务。第一项事务（请求/响应）获取主文档的一份拷贝（文件 A），这个文

[2] 法语：Conseil European pour la Recherche Nuclear（欧洲核子研究理事会）。

件有对于第二个和第三个文件的引用（指针）。当获得和浏览主文档的拷贝时，用户可以单击图片的引用来引起第二项事务并获取图片（文件B）的拷贝。如果用户需要看到被引用的文本文件的内容，她可以单击这个链接（指针）引起第三项事务并获取文件C的拷贝。注意，尽管文件A和文件B都存储在站点A上，但是它们是有不同名称和地址的独立文件。需要两项事务才能获取它们。非常重要的一点是我们需要记住，例2.2中的文件A，B和C是独立的网页，每一个都有独立的名称和地址。对于文件B或C的引用尽管包含在文件A中，但这并不意味着不能单独地获取每一个文件。第二个用户可以用一项事务来获取文件B。第三个用户可以用一项事务获取文件C。

网络客户（浏览器） 很多供应商提供可以解释和显示网页的商业浏览器（browser）。它们都使用几乎相同的结构。每个浏览器通常包含三部分：控制程序、客户协议和解释程序（见图2-9）。

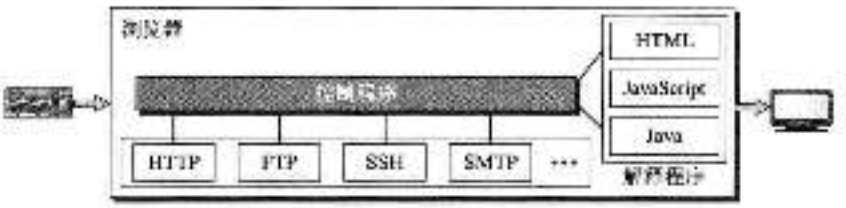


图2-9 浏览器

控制程序接收来自键盘和鼠标的输入并使用客户端程序来访问文档。在文档被访问后，控制程序使用解释程序来在屏幕上显示文档。客户协议可以是后文描述的任何一种协议，比如HTTP或FTP。解释程序可以是HTML、Java或JavaScript，这要取决于文档的类型。商业浏览器包括：Internet Explorer、Netscape Navigator以及Firefox。

网络服务器 网页是存储在服务器上的，每当请求到达时，相应的文档就被发送到客户。为了提高效率，服务器通常将被请求的文件存储在内存的缓存中；访问内存比访问磁盘快。服务器也可以通过多线程或多进程来提高效率。某些流行的网页服务器包含了Apache和微软互联网信息服务器。

统一资源定位符（Uniform Resource Locator，URL）

作为文件，网页需要有一个唯一的标识符来与其他网页区别开。为了定义一个网页，我们需要三个标识符：主机、端口以及路径。然而在定义网页之前，我们需要告知浏览器我们想使用哪个客户-服务器应用，这称为协议。这意味着我们需要四个标识符来定义网页。第一个是用来获取网页的运载工具形式，后三项组合在一起定义了目的对象（网页）。

- **协议。**第一个标识符是我们使用的用来访问网页的客户-服务器程序的缩写。尽管绝大多数情况下的协议是HTTP（超文本传输协议），这个协议我们将简要讨论，但是我们也会使用其他协议，比如FTP（文件传输协议）。
- **主机。**主机标识符可以是服务器的IP地址或主机被给予的唯一名称。IP地址可以按点分隔的十进制数表示法定义（如64.23.56.17），第4章对其进行了描述；名称通常是唯一定义主机的域名，比如forouzan.com。我们将在这一章后面的域名系统（DNS）讨论。
- **端口。**端口是一个16位整数，通常为客户-服务器应用而预定义。例如，如果用HTTP协议来访问网页，那么熟知端口号就是80。然而，如果使用一个不同的端口号，那么这个号码将被显示给出。
- **路径。**路径定义了下层的操作系统中文件的位置和名称。这个标识符的格式通常依赖于操作系统。在UNIX中，路径是后面接有文件名的一组目录名，它们通过斜杠来分隔。比如，/top/next/last/myfile是一个唯一定义myfile文件的路径，这个文件存储在目录last中，last目录是next目录的一部分，而next又在目录top下。换言之，路径自顶向下列出目录，后接文件名。

为了把这四部分组合在一起，使设计出了统一资源定位符（Uniform Resource Locator，URL），它在四个部分之间用三个不同的分隔符，如下所示：

```
protocol://host/path          绝大多数情况使用
protocol://host:port/path     当需要标识服务器端口号时使用
```

例 2.3 URL **http://www.mhhe.com/compsci/forouzan/**定义了与本书作者相关的一个网页。字符串 **www.mhhe.com** 是 McGraw-Hill 公司的计算机名称（**www** 这三个字母是主机名的一部分，它被加到商业主机的前面）。路径是 **compsci/forouzan/**，它定义了在目录 **compsci**（计算机科学）下 Forouzan 的网页。

网上文档

万维网的文档可以分为三大类：静态文档、动态文档和活动文档。

静态文档（Static Document） 静态文档是在服务器中创建和存储的固定内容的文档。客户只能得到一个文档的副本。换言之，文件的内容在创建文件时就决定了，而不是使用时决定的。当然，服务器中的内容是可以改变的，但是用户不能改变它们。当客户访问文档时，一个文档的副本被发送给用户。然后用户可以使用浏览器查看文档。静态文档使用如下语言编写：超文本标记语言（Hypertext Markup Language，HTML）、可扩展标记语言（Extensible Markup Language，XML）、可扩展样式表语言（Extensible Style Language，XSL）以及可扩展超文本标记语言（Extensible Hypertext Markup Language，XHTML）。我们将在附录 C 中讨论这些语言。

动态文档（dynamic document） 当浏览器请求文档时动态文档就被网页服务器创建。当一个请求到达时，网页服务器运行一个应用程序或一个脚本来创建动态文档。服务器返回程序或脚本的结果作为对请求文档的浏览器的响应。由于对每个请求都要创建一个全新的文档，因此动态文档的内容随着请求的不同而不同。一个动态文档的简单例子是从服务器获取时间和日期。时间和日期是动态信息，因为它们不断变化。客户可能要求服务器去运行一个程序，比如 UNIX 中的 data 程序，然后发送程序结果给客户。尽管过去公共网关接口（Common Gateway Interface，CGI）用来获取动态文档，但是如今选择使用脚本语言如 Java Server Pages（JSP），JSP 是使用 Java 语言来编写脚本的，或者 Active Server Pages（ASP），这是使用 Visual Basic 语言编写脚本的微软产品，或者 ColdFusion，它将结构化查询语言（Structured Query Language，SQL）数据库中的查询嵌入到 HTML 文档中。

活动文档（active document） 对很多应用来说，我们需要在客户站点运行一个程序或脚本。这些称为活动文档。比如，假设我们想运行一个在屏幕上创建动画的程序或运行一个与用户交互的程序。程序当然需要在显示了动画或者发生了互动的客户站点运行。当浏览器请求一个活动文档时，服务器发送文档或脚本的一个副本。然后文档在客户站点（浏览器）那里运行。一种创建活动文档的方法是使用 Java applets，这是一种 Java 编写的位于服务器的程序。它被编译并准备运行。文档是字节码（二进制）形式的。另一种方法是使用 JavaScript，但是要在客户站点下载和运行这个脚本。

超文本传输协议（HTTP）

超文本传输协议（HyperText Transfer Protocol，HTTP）是一种用来定义客户服务器程序如何编写和如何从万维网获取网页的协议。一个 HTTP 客户发送一个请求；HTTP 服务器返回响应。服务器使用 80 端口号；客户使用一个临时端口号。HTTP 使用 TCP 服务。在之前讨论过，TCP 是一种面向连接的可靠的协议。这就意味着，在客户和服务器进行任何事务之前，它们之间必须建立连接。在事务之后，连接应当终止。然而，客户和服务器不需要担心交换报文中的差错以及报文的丢失，因为 TCP 是可靠的而且将处理这个问题，我们将在第 3 章看到这一特性。

非持续与持续连接

正如我们在之前讨论的，嵌入到网页中的超文本概念可能需要多个请求和应答。如果网页、这个被获取的对象，位于不同的服务器，那么我们没有其他选择只能每获取一个对象就要创建一个新的 TCP 连接。然而，如果某些对象是位于同一台服务器的，我们可以有两种选择：一是每次使用

一个新的 TCP 连接获取一个对象，二是创建一个 TCP 连接获取全部对象。第一种方法称为非持续连接（nonpersistent connection），第二种称为持续连接（persistent connection）。在 HTTP1.1 版之前指定的是非持续连接，持续连接在 1.1 版中是默认的，但是可以被用户改变。

非持续连接 在非持续连接中，一个 TCP 连接被每一组请求/响应所创建。下面是这个策略的步骤：

1. 客户开启一个 TCP 连接并发送请求。
2. 服务器发送响应并关闭连接。
3. 客户读取数据直到它遇到了文件结束标记，然后关闭连接。

在这种策略中，如果文件包含了 N 个位于不同文件的图片连接（全都位于同一台服务器），那么必须开启和关闭连接 $N+1$ 次。非持续策略给服务器带来了高额开销，因为每次连接被开启时服务器都需要 $N+1$ 个不同的缓冲区。

例 2.4 图 2-10 展示了一个非持续连接的例子。客户需要访问一个包含图片链接的文件，文本文件和图片位于同一台服务器上。这里我们需要两个连接。对于每一个连接，TCP 需要至少三个握手报文来建立连接，但是请求可以和第三个报文一起发送。在连接建立之后，请求对象可以被发送。在接收到一个对象之后，需要另外三次握手报文来结束连接。我们将在第 3 章看到这一点。这意味着客户和服务器参与到这两次连接建立和连接终止之中。如果这项事务包含获取 10 个或 20 个对象，那么这些握手的往返时间总和将会是一个很大的开销。当在本章结尾展示客户-服务器编程时，我们将看到客户和服务器需要为每一个连接分配额外的资源，如缓冲区和变量。这是两个站点的另一个负担，而且对于服务器这边来说负担尤其重。

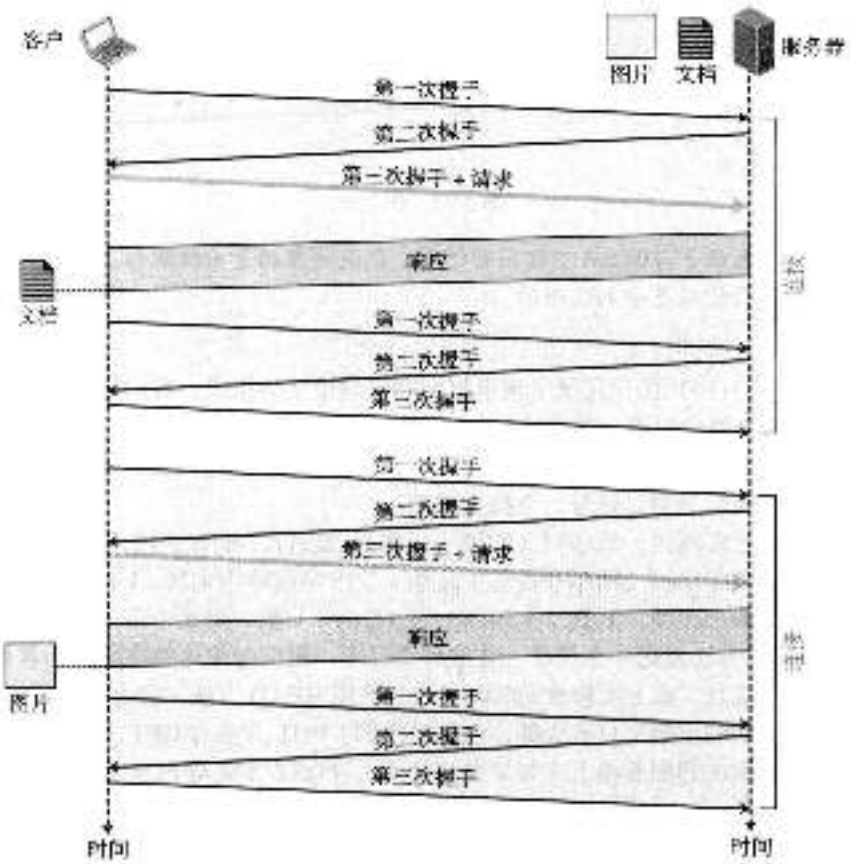


图 2-10 例 2.4

持续连接 HTTP1.1 版默认指定了持续连接。在持续连接中服务器在发送一个响应后，为响应更多的请求而将连接置为打开状态。服务器可以在客户的请求下或者在超时情况下将连接关闭。发送方通常在每次响应中发送数据长度。然而，偶尔情况下发送方不知道数据的长度。这是创建动态文档或活动文档时的情形。在这种情形下，服务器通知客户长度未知并在发送数据后关闭连接。因此客户知道数据已接收完毕。通过使用持续连接，可以节省时间和资源。每个站点只需要为连接设定一组缓冲区和变量。同时节省了连接建立和终止的往返时间。

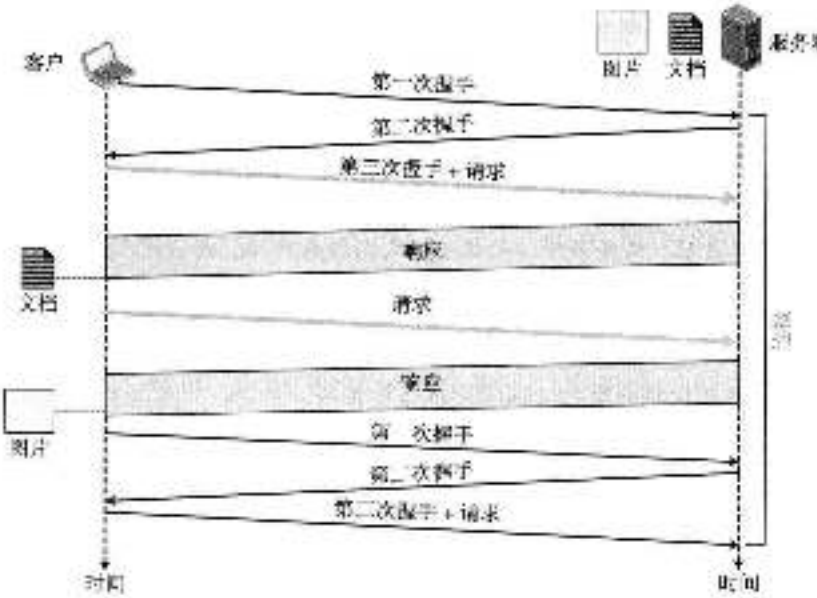


图 2-11 例 2.5

例 2.5 图 2-11 展示了与例 2.4 中相同的情景，但是使用的是持续连接。只有一个连接建立和终止，但是对于图片的请求是分别发送的。

报文格式

如图 2-12 所示，HTTP 协议定义了请求报文和响应报文的格式。我们把两种格式并列以示比较。每一种报文由四个部分组成。请求报文中的第一部分称为请求行；响应报文的第一部分称为状态行。其他三部分在请求报文和响应报文中有相同的名称。然而，这三部分只是名称相似，它们可能含有不同的内容。我们分别讨论每一个报文类型。

请求报文 正如之前所述，请求报文的第一行称为请求行。如图 2-12 所示，这一行有三部分由空格分隔开并且被两个字符（回车和换行）终止。这些字段称为方法、URL 和版本。

方法字段定义了请求类型。如表 2-1 所示，在 HTTP1.1 版中定义了若干种方法。绝大多数情况下，客户使用 GET 方法发送一个请求。在这种情况下，报文的主体是空的。当客户仅需要从服务器获得关于网页的信息，如上次修改的时间，这时使用 HEAD 方法。它也可以用来检测 URL 的有效性。这种情况下的响应报文只有头部；主体是空的。PUT 方法与 GET 方法是相反的；它允许客户将一个新的页面发送到服务器上（如果允许的话）。POST 方法与 PUT 方法类似，但是它用来发送一些信息到服务器上。这些信息被加入网页或用来修改网页。TRACE 方法用来调试；客户要求服务器回送请求来检查服务器是否正在获得请求。如果客户获得许可，DELETE 方法允许客户删除一个服务器上的网页。CONNECT 方法原先作为预留方法；后文会讨论到，这个方法可能被代

据服务器使用。最后，OPTIONS 方法允许客户询问网页属性。

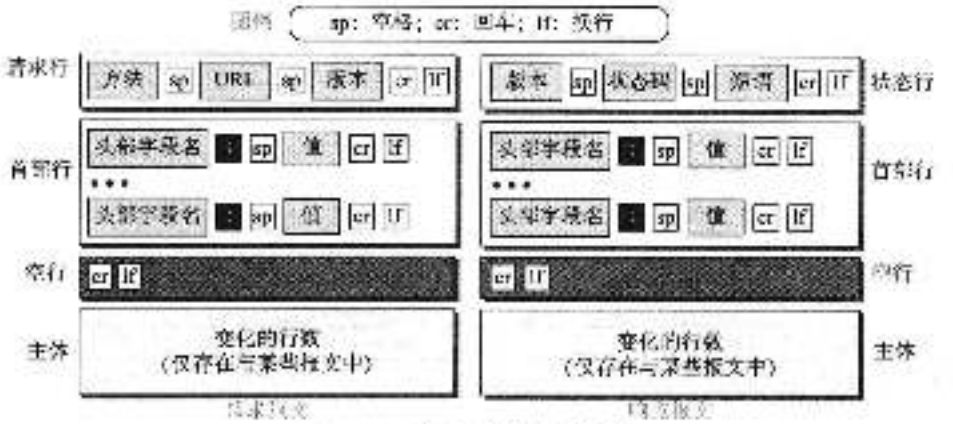


图 2-12 请求和响应报文格式

表 2-1 方法

方 法	动 作	方 法	动 作
GET	向服务器请求文档	TRACE	回送收入的请求
HEAD	请求关于文档的信息，而不是文档本身	DELETE	删除网页
PUT	从客户端向服务器发送文档	CONNECT	预留
POST	从客户端向服务器发送一些信息	OPTIONS	询问有关可用选项的信息

第二个字段：URL，在本章的前面部分已经讨论过了。它定义了相关网页的地址和名称。第三个字段：版本，给出了协议的版本，HTTP 最常用的版本是 1.1。

在请求行之后我们可以有一个或多个请求头部（request header）行。每一个头部行都从客户端向服务器发送额外的信息。例如，客户可以请求以某种特定格式发送文档。每个头部行有头部名字、一个冒号、一个空格和一个头部值（见图 2-12）。表 2-2 列出了一些请求中常用的头部名字。值字段定义了与每个头部名字相关的值。值列表可以在相应的 RFC 中查找到。

主体可以出现在请求报文中。通常，当使用 POST 或 PUT 方法时，它包含要发送的评论或要发布到网站上的文档。

表 2-2 请求头部名称

头 部	说 明	头 部	说 明
User-agent	标识客户端程序	Host	给出主机及客户端的端口号
Accept	给出客户端能够接受的媒体格式	Date	给出当前日期
Accept-charset	给出客户端可以处理的字符集	Upgrade	确定首选的通信协议
Accept-encoding	给出客户端可以处理的编码方案	Cookie	返回 cookie 给服务器（稍后解释）
Accept-language	给出客户端可以接受的语言	If-Modified-Since	如果文档在指定的日期之后被更改，则发送文档
Authorization	给出客户端有哪些许可		

响应报文 图 2-12 给出了响应报文的格式。响应报文包含状态行、头部行并且有时包含主体。响应报文的第一行称为状态行。这一行有三个字段，它们由空格分隔开并且被两个字符（回车和换行）终止。第一个字段定义了 HTTP 协议的版本，通常为 1.1。状态码字段定义了请求的状态。它包含三个数字。在 100 范围内的代码只代表一个报告，在 200 范围内的代码表示这是一个成功的请求。在 300 范围内的代码表示把客户端重定向到另一个 URL，在 400 范围内的代码表示在客户端发生错

误。最后，在500范围内的代码表示错误发生在服务器端。状态短语以文本格式解释了状态码。

在状态行之后，我们可以有一个或多个响应头部行。每一个头部行都从服务器向客户端发送额外的信息。例如，发送方可以发送关于文档的额外信息。每个头部行都有一个头部名称、一个冒号、一个空格和一个头部值。我们将在本节结尾列举一些头部行。表2-3列出了一些常用的头部名称。

表2-3 响应头部名称

头 部	描 述	头 部	描 述
Date	给出当前日期	Content-Length	给出文档长度
Upgrade	规定首选的通信协议	Content-Type	指定媒体类型
Server	给出服务器信息	Location	指明新建或移动后文档的位置
Set-Cookie	服务器要求客户存储 cookie	Accept-Ranges	服务器将会接收的被请求的字节范围
Content-Encoding	指定编码方案	Last-modified	给出上次改变的日期和时间
Content-Language	指定语言		

主体包含了从服务器发送给客户的文档。除非响应是一个错误报文，否则主体是存在的。

例 2.6 这个例子获取了一个文档（见图 2-13）。我们使用 GET 方法来获取一个路径为 **/usr/bin/image1** 的图片。请求行给出了使用的方法（GET）、URL 以及 HTTP 版本（1.1）。头部有两行，它们表示客户可以接收 GIF 或 JPEG 格式的图片。请求是没有主体的。响应报文包含了状态行以及四个头部行。头部行定义了日期、服务器、内容编码（MIME 版本，在电子邮件部分将会描述）以及文档长度。文档主体在头部之后。

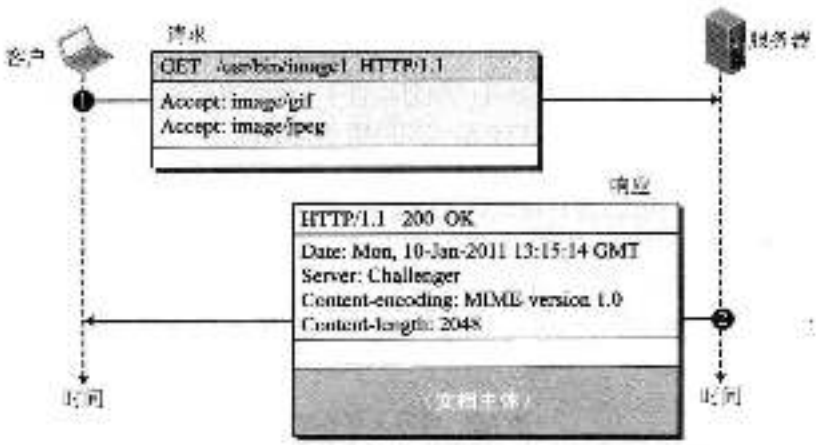


图 2-13 例 2.6

例 2.7 在这个例子中，客户要向服务器发送一个网页。我们使用 PUT 方法。请求行给出方法（PUT）、URL 以及 HTTP 版本（1.1），其头部有四行。请求主体包含要发送的网页。响应报文包含状态行和四个头部行，被创建的文档是一个 CGI 文档，它包含在响应报文的主体中（见图 2-14）。

条件请求

客户可以在请求中加入条件。在这种情况下，如果条件满足，服务器将会发送被请求的网页或者通知用户。客户加入的最常见的一种条件是网页被修改的时间和日期。客户可以在发送请求时附带头部行 If-Modified-Since，这样来告知服务器客户只需要在指定日期之后更新的页面。

例 2.8 以下展示了一个客户在请求中加入了修改日期和时间的条件。

```
GET http://www.commonServer.com/information/file1 HTTP/1.1      请求行
If-Modified-Since: Thu, Sept 04 00:00:00 GMT                    头部行
                                                                空行
```

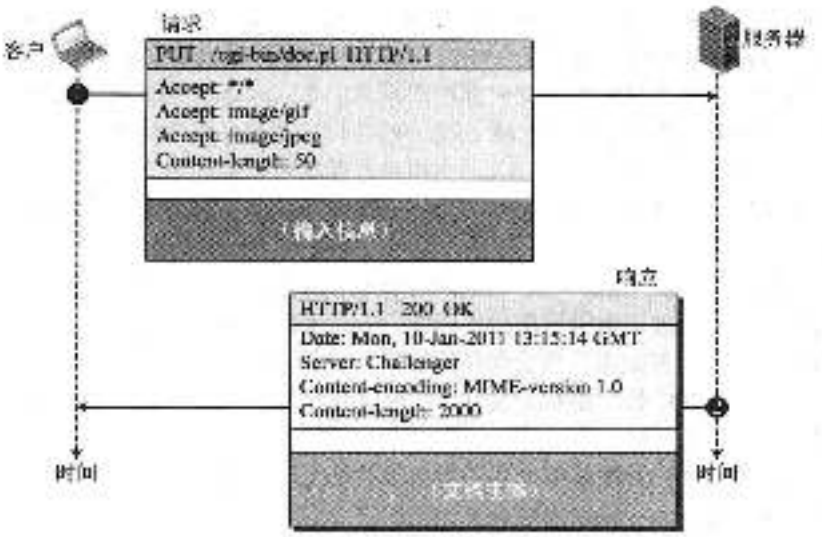


图 2-14 例 2.7

响应中的状态行表示在指定日期之后文档没有修改。响应报文的主体是空的。

```
HTTP/1.1 304 Not Modified                    状态行
Date: Sat, Sept 06 08 16:22:46 GMT           头部第一行
Server: commonServer.com                     头部第二行
                                             空行
(Empty Body)                                 实主体
```

Cookie

万维网起先被设计成无状态实体。客户发送请求；服务器响应。它们之间的关系就结束了。最初设计的 Web，公开检索可用的文档，完全符合这个目标。现在 Web 还有其他功能，如下所示：

- 网站作为电子商店（electronic store），允许客户在商店内浏览、选择需要的商品，把它们放入电子购物车内，最后使用信用卡付费。
- 有些网站只允许注册客户（registered client）访问。
- 有些网站是门户网站（portal），用户可以选择他想看的网页。
- 有些网站仅作为广告（advertising）代理。

为了实现这些目的，cookie 机制就应运而生。

创建和存储 Cookie Cookie 的创建和存储与实现有关，然而它的原理是相同的。

1. 当服务器从客户端接收到请求后，它将客户端的信息存储在文件或字符串中。这些信息可能包含客户端的域名、cookie 内容（服务器收集到的关于客户端的信息，如主机名、注册号等）、时间戳，以及与实现有关的其他信息。

2. 服务器在响应中包含了它发送给客户端的 cookie。

3. 当客户端接收到响应后，浏览器在 cookie 目录中存储 cookie，并根据服务器域名进行分类。

使用 Cookie 当客户向服务器发送请求时，浏览器在 cookie 目录中查询是否有从那个服务器发送过来的 cookie。如果有，则在请求中包含这个 cookie。当服务器接收到这个请求后，它就知道了这是一个老客户，而不是新的。注意，cookie 的内容从来不让浏览器读取或者透露给用户，只由服务器创建并回收 cookie。现在让我们分析如何使用 cookie 来实现上面提到的四个功能。

- 网上电子商店（电子商务）可以为客户端的购物者使用 cookie。当客户端选择商品，并放入购物车中后，包含了这些商品信息（如它的数量、单价）的 cookie 就被发送到浏览器。

如果客户端选择第二个商品，cookie 就被新的选择信息所更新，依此类推。当客户端结束购物并准备付账离开时，就检索最终的 cookie，然后计算出总的费用。

- 当客户端第一次注册时，网站就向客户端发送一个 cookie。网站通过这种方式限制注册用户的访问。只有那些能够发送正确 cookie 的客户才能被允许今后重复访问。
- Web 门户以相同的方式使用 cookie。当用户选择她最喜爱的网页时，就生成一个 cookie 并发送到浏览器。当这个网站再次被访问时，这个 cookie 就发送给服务器说明这个客户端要查找什么页面。
- cookie 也用来作为广告代理。广告代理能够将大字标题广告放置在用户经常访问的网站的主页面上。广告代理仅提供指出大字标题广告地址的 URL，而不是大字标题广告本身。当用户访问网站主页并点击广告公司的图标时，一个请求就发送给了广告代理；广告代理就发送一个大字标题广告，如 GIF 文件，同时也包含了一个含有用户 ID 的 cookie。将来对这个大字标题广告的任何使用都会加入到一个分析用户 Web 行为的数据库中。广告代理已经收集了用户的爱好，并能够将这些信息卖给其他组织。这种 cookie 的使用方法引起很多争议，但愿今后能引入一些新的法规来保护用户的隐私信息。

例 2.9 图 2-15 展示了电子商店可以从 cookie 的使用中受益的场景。

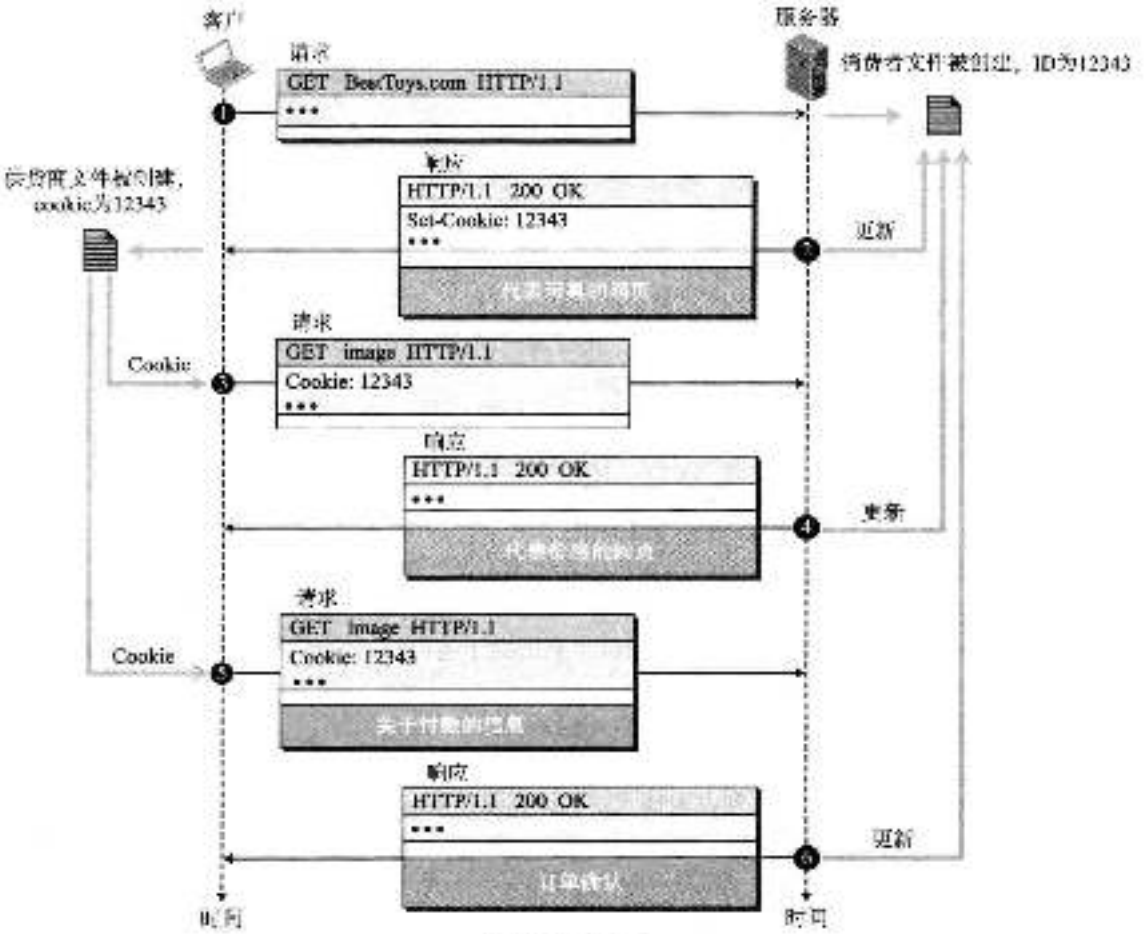


图 2-15 例 2.9

假设一个购物者想要从名叫 BestToys 的电子商店购买一个玩具。购物者浏览器（客户）发送一个请求到 BestToys 服务器。服务器为客户创建了一个空购物车（列表）并给购物车分配了一个 ID（如 12343）。服务器然后发送一个响应报文，它包含了所有可购买的玩具的图片。每个玩具下都有链接。如果单击，这个玩具就被选中。这个响应报文也包含了值为 12343 的 Set-Cookie 头部行。客户显示这些图片并且将 cookie 存储在名为 BestToys 的文件中。cookie 对购物者是不可见的。现在购物者选择一个玩具，单击它。客户发送了一个请求，但是 ID 号 12343 包含在 cookie 头部行。尽管服务器可能很繁忙并且忘记了消费者，但是当它接收到请求并检查头部时会发现 12343 这个值。服务器知道这个消费者不是新顾客；它找寻带有 ID 12343 的购物车。购物车（清单）被打开，然后选中的玩具被输入到清单中。现在服务器发送另一个响应给购物者，告知她价钱并要求其支付款项。消费者提供她信用卡的信息并发送一个 cookie 值为 ID 12343 的新请求。当请求到达服务器时，它再次看到 ID 12343 并且接受订单以及付款，在响应中发送一个确认信息。关于客户的其他信息存储在服务器上。以后如果消费者访问这个商店，客户再次发送 cookie；商店会获取文档并且拥有关于用户的全部信息。

万维网高速缓存：代理服务器

HTTP 支持代理服务器（proxy server）。代理服务器是一台计算机，能够保存最近请求的响应的副本。HTTP 客户端向代理服务器发送请求。代理服务器检查本地高速缓存。如果高速缓存中不存在响应报文，代理服务器就向相应的服务器发送请求。返回的响应会发送到代理服务器中，并且进行存储，以用于其他客户端将来的请求。

代理服务器降低了原服务器的负载，减少了通信量并降低了延迟。但是，为了使用代理服务器，必须配置客户端访问代理服务器而不是目标服务器。

请注意，代理服务器既作为一个服务器又作为一个客户。当它收到客户的请求并有一个要发送给客户的响应时，它作为服务器并且发送响应给客户。当它收到客户的请求但没有要发送给客户的响应时，它首先作为客户然后发送请求给目标服务器。当响应被接受，它又作为服务器并发送响应给客户。

代理服务器位置

通常代理服务器位于客户站点。这意味着我们可以有如下代理服务器的层级：

1. 客户计算机也可以用作小容量代理服务器，它存储着与客户经常调用的请求相对应的响应。
2. 在一个公司，一个代理服务器可能安装在计算机 LAN 中来减少流出 LAN 的负载。
3. 含有很多客户的 ISP 可以安装一台代理服务器来减少进出 ISP 网络的负载。

例 2.10 图 2-16 给出了在诸如校园网或公司网这类的本地网络中使用代理服务器的例子。代理服务器安装在本地网络中。当任意客户（浏览器）创建一个 HTTP 请求，请求首先到达代理服务器。如果代理服务器已经有响应的网页，那么它发送响应给客户。否则代理服务器会作为客户并且发送这个请求给因特网中的网络服务器。当响应返回，代理服务器在将其发送到客户端前，制作一个副本并存储到它的高速缓存中。

缓存更新

一个非常重要的问题是一个响应在被删除或被替换前，应该在代理服务器保持多长时间。很多不同的策略用来达到这个目的。一个解决方法是保存站点列表，这些站点的信息保存一段时间。比如，一个新的代理可能每天早上改变它的新闻页面。这意味着，代理服务器可以在早上获得新闻并保持它直到第二天。另一个建议是加入一些头部来显示信息的最近修改时间。代理服务器可以使用这些头部的信息去猜测它们在多长时间内是有效的。

HTTP 安全

HTTP 本质上并不提供安全。然而，我们在第 10 章指出，HTTP 可以在安全套接层（SSL）上运行。在这种情况下，HTTP 称为 HTTPS。HTTPS 提供保密性、客户和服务器鉴别，以及数据完整性。

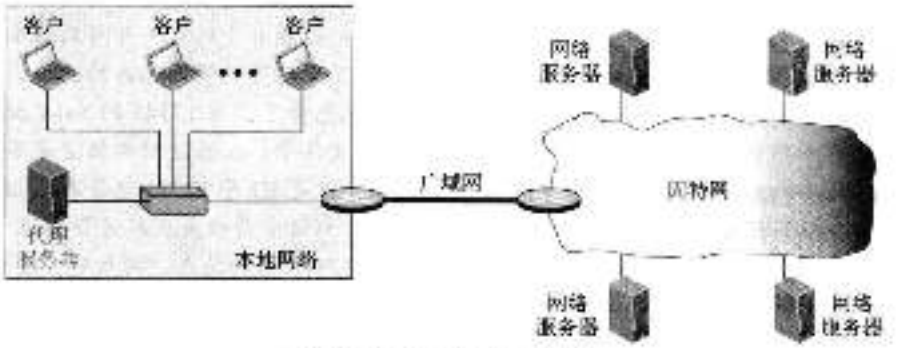


图 2-16 代理服务器的例子

2.3.2 FTP

文件传输协议（File Transfer Protocol，FTP）是 TCP/IP 提供的标准机制，用于将文件从一个主机复制到另一个主机。虽然从一个系统到另一个系统传送文件看起来很简单而且直观，但首先还是要解决一些问题。例如，两个系统可能使用不同的文件名约定。两个系统使用不同的方法来表示文本和数据。两个系统具有不同的目录结构。所有这些问题都已经由 FTP 以一种非常简单而巧妙的方法解决了。尽管我们可以使用 HTTP 传送文件，但是 FTP 是传送大文件或使用不同格式传送文件的更好选择。图 2-17 展示了 FTP 基本模型。客户有三个组件：用户接口、客户控制进程和客户数据传输进程。服务器有两个组件：服务器控制进程和服务器数据传输进程。控制连接是在控制进程之间进行的，而数据连接是在数据传输进程之间进行的。

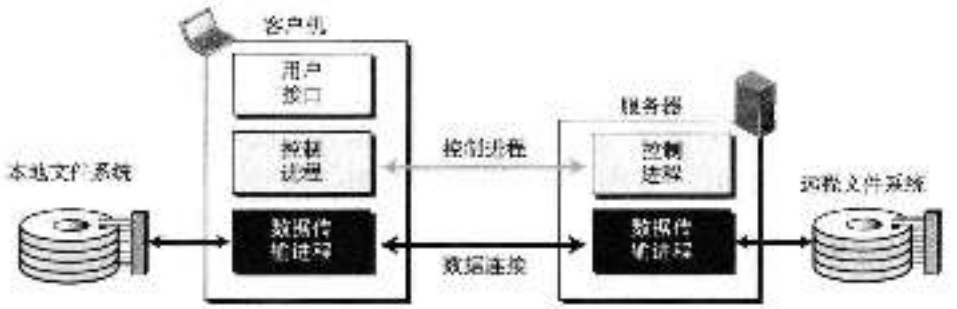


图 2-17 FTP

将命令和数据传输分开使得 FTP 效率更高。控制使用非常简单的通信规则。我们一次需要传输的只是一行命令或者一行响应。另一方面，因为传输的数据类型种类多，数据传输需要更加复杂的规则。

两种连接的寿命

FTP 中的两种连接有不同的寿命。在整个交互的 FTP 会话期间，控制连接始终处于连接状态。数据连接则在每次传输文件时开启然后关闭。每当涉及文件传输的命令被使用时，数据连接就被打开，而当文件传输完毕时连接就关闭。换言之，当用户开始 FTP 会话时，控制连接就被打开。在控制连接处于打开状态期间，如果传输多个文件，那么数据连接可以打开和关闭多次。FTP 使用两个熟知端口：端口 21 用于控制连接，端口 20 用于数据连接。

控制连接

对于控制通信，FTP 使用与 TELNET（后文讨论）相同的方法。它与 TELNET 一样使用 NVT ASCII 字符集。通信是通过命令和响应来完成的。这种简单方法适合控制连接，因为我们一次发送

一条命令（或响应）。每一条命令或响应都是一个短行，因此不必担心它的文件格式或文件结构，每一行结束处是两个字符（回车和换行）的行结束标记。

在控制连接期间，命令从客户端发送到服务器并且响应从服务器发送到客户端。从FTP客户控制进程发送的命令是ASCII大写字母形式的，可能带有也可能不带有参数。一些常见命令如表2-4所示。

表2-4 一些FTP命令

命 令	参 数	说 明
ABOR		放弃之前的行
CDUP		改变到上级父目录
CWD	目录名	改变到另一个目录
DELE	文件名	删除文件
LIST	目录名	列出了目录或文件
MKD	目录名	创建新目录
PASS	用户密码	密码
PASV		服务器选择一个端口
PORT	端口标识符	客户选择一个端口
PWD		显示当前目录名
QUIT		退出登录
RETR	文件名	获取文件：文件从服务器发送到客户
RMD	目录名	删除目录
RNFR	文件名	标识一个将被重命名的文件
RNTO	文件名（新）	重命名文件
STOR	文件名	存储文件：文件从客户发送到服务器
STRU	F、R或P	定义数据组织（F：文件，R：记录，或者P：页面）
TYPE	A、E、I	定义文件类型（A：ASCII，E：EBCDIC，I：图像）
USER	用户ID	用户信息
MODE	S、B或C	定义传输方式（S：流，B：块，C：压缩）

每个FTP命令至少产生一个响应。一个响应有两部分：跟随在文本后的一个三位数字，数字部分定义了编码；文本部分定义了需要的参数或进一步的解释。第一个数字定义了命令状态。第二个数字定义了状态应用的区域。第三个数字提供了额外信息。表2-5给出了一些常见响应。

表2-5 一些FTP响应

编 码	说 明	编 码	说 明
125	数据连接打开	250	请求文件动作成功
150	文件状态良好	331	用户名成功；需要密码
200	命令成功	425	无法打开数据连接
220	服务就绪	450	文件动作未被采用：文件不可用
221	服务关闭	452	动作被放弃：磁盘空间不足
225	数据连接打开	500	语法错：无法识别的命令
226	关闭数据连接	501	参数或变量语法错
230	用户登录成功	530	用户未登录

数据连接

数据连接使用服务器站点的熟知端口20。然而，数据连接的创建和控制连接是不同的。步骤如下：

1. 客户，不是服务器，使用临时端口发起一个被动打开。这必须由客户完成，因为正是客户发出命令要求传输文件的。

2. 客户使用 PORT 命令发送这个端口号到服务器。

3. 服务器接收到端口号，使用熟知端口 20 发出主动打开并且接收临时端口号。

通过数据连接的通信

数据连接的目的和实现与控制连接是不同的。我们通过数据连接来传输文件。客户必须定义传输文件的类型、数据结构以及传输模式。在通过数据连接发送文件之前，我们通过控制连接进行准备。异构性问题可以通过定义三个通信属性来解决：文件类型、数据结构和传输方式。

数据结构 FTP 可以使用下列数据结构中的一种在数据连接上传送文件：文件结构、记录结构和页面结构。文件结构格式（默认使用）没有结构。它是连续的字节流。在记录结构中，把文件划分成一些记录。这只能用于文本文件。在页面结构中，把文件划分成页面，每一个页面有一个页面号和页面头部。页面可以随机地或顺序地存储或访问。

文件类型 FTP 可以在数据连接上传送下列文件中的一种：ASCII 文件、EBCDIC 文件和图像文件。

传输方式 FTP 可以使用下列三种传输方式之一在数据连接上传送文件：流方式、块方式和压缩方式。流方式是默认方式，数据作为连续的字节流从 FTP 传递给 TCP。在块方式中，数据可以按块从 FTP 传递给 TCP。在这种情况下，每一个块前面有一个 3 字节的头部。第一字节称为块描述符，后面两个字节以字节为单位定义块的大小。

文件传输

文件传输是在控制连接上发送用来的命令的控制下，在数据连接上进行的。然而，我们要记住，FTP 的文件传输表示三件事情之一：读取文件（服务器到客户）、存储文件（客户到服务器）和目录列表（服务器到客户）。

例 2.11 图 2-18 给出了使用 FTP 读取文件的一个例子。图中展示了只有一个文件将要传送的情况。控制连接总是保持打开，但是数据连接重复地打开和关闭。我们假设文件以六个部分传输。在所有记录都被传输后，服务器控制进程宣布文件传输完成。由于客户控制进程没有文件要读取，它发出 QUIT 命令，这导致了服务连接被关闭。

例 2.12 下面给出了一个实际的 FTP 会话，它列出目录。灰色的行表示来自服务器控制连接的响应，黑色的行表示用户发送的命令。黑底反白的行表示数据传输。

```
$ ftp voyager.deanza.fhda.edu
Connected to voyager.deanza.fhda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.fhda.edu:forouzan): forouzan
331 Please specify the password.
Password:**********
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.
drwxr-xr-x    2    3027    411     4096 Sep 24  2002  business
drwxr-xr-x    2    3027    411     4096 Sep 24  2002  personal
drwxr-xr-x    2    3027    411     4096 Sep 24  2002  school
226 Directory send OK.
ftp> quit
221 Goodbye.
```

FTP 安全

设计 FTP 协议的时候安全并不是一个大问题。尽管 FTP 要求密码，但是密码还是用明文（未

加密）发送，这意味着它可能被截获并被攻击者使用。数据传输连接也用明文传输数据，这是不安全的。为了安全起见，可以把安全套接层加入到 FTP 应用层和 TCP 层之间。这种情况下，FTP 称为 SSL-FTP。当我们在本章后面讨论 SSH 时，我们也会探索一些安全文件传输应用。

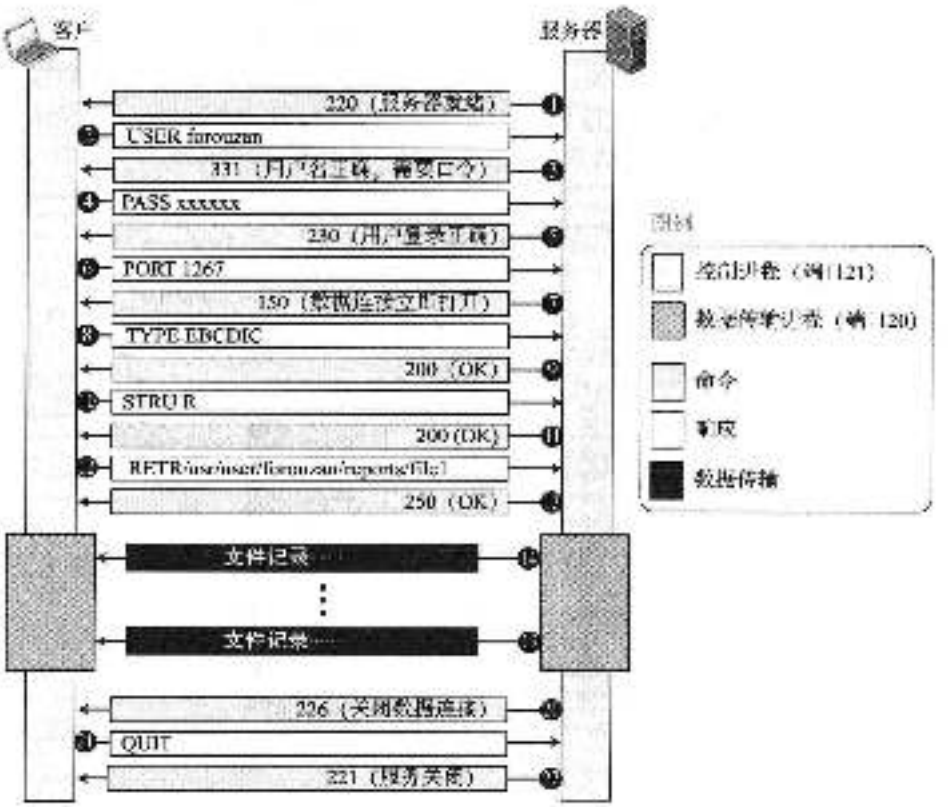


图 2-18 例 2.1

2.3.3 电子邮件

电子邮件（或 e-mail）允许用户交换报文。然而，这种应用的本质与到目前为止讨论的其他应用不同。在诸如 HTTP 或 FTP 应用中，服务器程序不断运行，等待来自客户的请求。当请求到达时，服务器提供服务。这里存在一个请求和一个应答。在电子邮件中，情况不同。首先，电子邮件是一个单向事务。当 Alice 发送一个电子邮件给 Bob，她可能期待着回复，但是这并不是一个命令。Bob 可能响应也可能不响应。如果他响应，这又是一个单向事务。第二，Bob 运行一个服务器程序并且等待别人给他发送电子邮件是不可行的也是不合情理的。当 Bob 不使用计算机时他可能将其关闭。这意味着客户/服务器编程的思想应该按照另一种方式执行：使用介于中间的计算机（服务器）。当用户想要发送邮件的时候他只运行客户程序并且中间服务器提供客户/服务器模式，这就像我们在下一节将要讨论的一样。

架构

为了说明电子邮件的架构，我们给出如图 2-19 所示的常见情景。还有一种可能是 Alice 或 Bob 直接连接到相应的邮件服务器上。那样不要求 LAN 或 WAN 连接。但是这种变化不影响我们的讨论。

在通常情况下，电子邮件的发送者和接收者，分别是 Alice 和 Bob，通过 LAN 或者 WAN 连接到两个邮件服务器上。管理员为每个用户创建了一个邮箱，接收到的报文被存储在邮箱里。邮箱是

服务器硬盘的一部分，是一个带有限制的特殊文件。只有邮箱的拥有者才能访问它。管理员也创建了一个队列（池）来存储等待发送的报文。

如图 2-19 所示，从 Alice 到 Bob 的一封简单的电子邮件需要九个不同步骤。Alice 和 Bob 使用三个不同的代理：**用户代理**（User Agent，UA）、**报文传输代理**（Mail Transfer Agent，MTA）以及**报文访问代理**（Message Access Agent，MAA）。当 Alice 需要给 Bob 发送报文时，她运行客户代理程序准备报文并发送到她的邮件服务器。然而使用 MTA 的话，报文需要从 Alice 的站点穿过因特网发送到 Bob 的站点。这里需要两个报文传输代理：一个客户和一个服务器。像因特网的客户-服务器程序，服务器需要始终运行，因为它不知道用户何时会请求一个连接。另一方面，当队列中有发送报文时，可由系统激活客户程序。Bob 处的客户代理允许 Bob 读取报文。然后 Bob 使用一个 MAA 客户从在第二个服务器上运行的 MAA 服务器中读取报文。

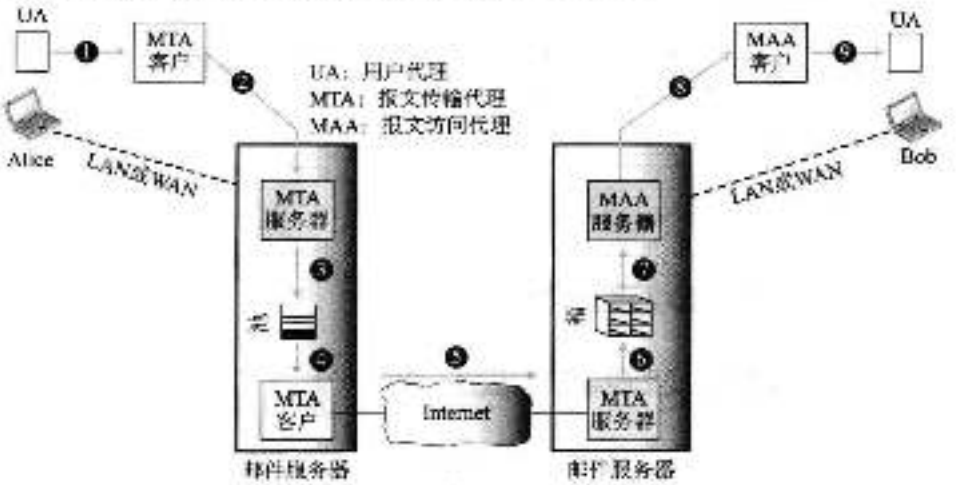


图 2-19 常见情景

我们这里需要强调两个重点。第一，Bob 不能绕开邮件服务器直接使用 MTA 服务器。为了直接使用 MTA 服务器，Bob 应该需要始终运行 MTA 服务器程序，因为他不知道报文何时到达。这意味着如果通过局域网连接他的系统，Bob 必须使计算机保持开机状态。如果他通过广域网连接到他的系统，他必须始终保持连接状态。今天，这两种情形已经都不适用了。

第二点，注意到 Bob 需要另一对客户-服务器程序：报文访问程序。这是因为 MTA 客户-服务器城区是一个推（push）程序。客户需要从服务器拉出报文。我们马上会更多地讨论 MMA。

电子邮件系统需要两个 UA，一对 MTA（客户与服务器）以及一对 MAA（客户与服务器）

用户代理

电子邮件系统的第一个组件是**用户代理**（UA）。它向用户提供服务，使发送和接收一个报文变得容易。用户代理是一个软件包（程序），它由读写、回答和转发报文组成。它也处理用户计算机的本地邮箱。

有两种类型的用户代理：命令驱动型和基于 GUI 的。命令驱动型用户代理属于早期的电子邮件。在服务器中仍然存在这种类型的用户代理。命令驱动型用户代理通常是从键盘接收单个字符的命令以执行某项任务。例如，用户可以在命令提示符输入字符 r 回答报文的发送方，或输入 R 回答发送方和所有的接收者。命令驱动型用户代理的例子有：mail、pine 和 elm。

现在的用户代理都是基于 GUI 型的。它们包含图形用户接口（GUI）组件，该组件允许用户使用键盘和鼠标与软件进行交互。它们还有一些图形组件，如图标、菜单条和使服务更容易访问的

窗口。基于 GUI 的用户代理有 Eudora 和 Outlook。

发送邮件

为了发送邮件，用户通过 UA 创建邮件，邮件看起来像邮政邮件。它有一个信封和一个报文（见图 2-20）。信封通常包含发信人的地址和收件人的地址以及其他信息。报文包含头部和主体。报文头部定义了发信人、收件人、报文的主题以及其他信息。报文的主体包含了收信人要读取的真正信息。

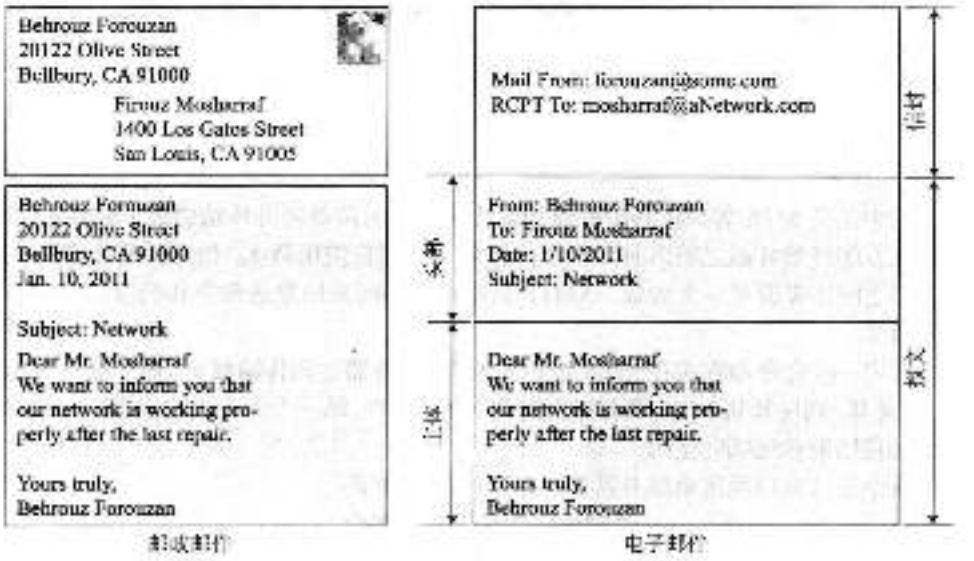


图 2-20 电子邮件格式

接收邮件

用户（或定时器）触发用户代理检查邮箱。如果用户有邮件，UA 就用一个通知来告诉用户。如果用户准备读取邮件，则会显示一个列表，其中列表的每一行包含了邮箱中关于一个特定报文的信息概要。这个概要通常包括发信人的邮件地址、主题以及发送和接收此邮件的时间。用户可以选择任何一个报文，在屏幕上显示它的内容。

地址

为了传递邮件，邮件处理系统必须使用具有唯一地址的寻址系统。在因特网中，地址由两部分构成：本地部分（local part）和域名（domain name），并且用符号@隔开（见图 2-21）。

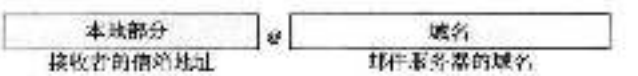


图 2-21 电子邮件地址

本地部分定义了一个特定文件的名字，它称为用户邮箱，在用户邮箱中存储了用户所有接收到的邮件，以便用户代理进行读取。第二部分是域名。一个组织机构通常选择一个或者多个主机用于接收和发送邮件；这些主机有时称为邮件服务器或交换机。分配给每一个邮件交换机的域名或者是来自 DNS 数据库，或者是一个逻辑名字（例如，该组织机构的名字）。

邮件列表或组列表

电子邮件允许用一个名字即别名来表示多个不同的电子邮件地址，这称为邮件列表。每当发送一个报文时，系统就将收信人的名字与别名数据库进行比较；如果所定义的别名有一个邮件列表，那么就将报文分开，每一个对应列表中的一项，然后交给 MTA 处理。

报文传输代理：SMTP

基于普通情景（见图 2-19），我们可以说电子邮件是那些需要使用三组客户-服务器模式完成任务

的众多应用之一。很重要的是，当我们处理电子邮件时我们要区分这三组。图 2-22 给出了这三组客户-服务器应用。我们把第一组和第二组称为报文传输代理（MTA），第三组称为报文访问代理（MAA）。

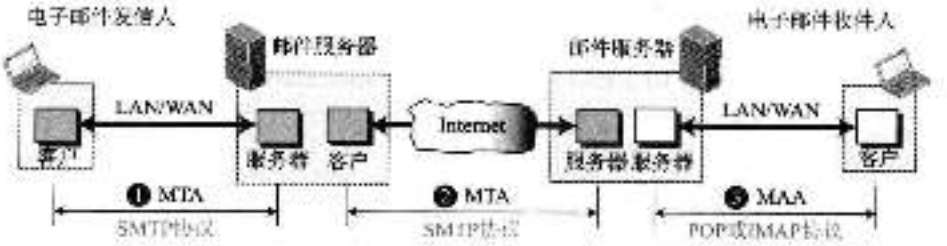


图 2-22 电子邮件中使用的协议

在因特网中定义 MTA 客户机和服务器的常用协议称为**简单邮件传输协议**（SMTP）。SMTP 在发送方和接收方邮件服务器之间以及两个邮件服务器之间被使用两次。稍后我们将看到，在邮件服务器与接收器之间还需要另一个协议。SMTP 只定义了如何来回发送命令和响应。

命令和响应

SMTP 使用一些命令和响应在 MTA 客户和 MTA 服务器之间传输报文。命令是从 MTA 客户发送到 MTA 服务器；响应是从 MTA 服务器发送到 MTA 客户。每一个命令或响应都以一个双字符（回车和换行）的行结束标记来终止的。

命令 命令是从客户发送给服务器的，命令的格式如下：

关键词：变量（可能多个）

它包含一个关键词，后面跟着零个或多个变量。SMTP 定义了 14 个命令，如表 2-6 所示。

表 2-6 SMTP 命令

关 键 词	变 量	说 明
HELO	发送方的主机名	识别自身
MAIL FROM	发信人	识别发信人
RCPT TO	预期的收信人	识别收信人
DATA	邮件主体	发送实际报文
QUIT		结束报文
RSET		放弃当前邮件事务
VRFY	收信人名字	检验收信人地址
NOOP		检测收信人状态
TURN		交换发信人和收信人
EXPN	邮件列表	要求收信人扩展邮件列表
HELP	命令名	要求收信人以多级形式发送关于命令的信息
SEND FROM	预期的收信人	指定邮件只发送到收信人的终端而不是邮箱
SMUL FROM	预期的收信人	指定邮件被发送到收信人的终端或邮箱
SMAL FROM	预期的收信人	指定邮件被发送到收信人的终端和邮箱

响应 响应是服务器发给用户的。响应是一个三位数字码，后面可以跟着附加的文本信息。表 2-7 列出了各种响应。

邮件传输阶段

传输一个邮件共有三个阶段：连接建立、邮件传输和连接终止。

连接建立 在客户创建了到端口 25 的 TCP 连接后，SMTP 服务器开始连接阶段。这个阶段涉

及三个步骤：

1. 服务器发送代码 220（服务就绪）告知客户它准备好接收邮件。如果服务器没有就绪，它发送代码 421（服务不可用）。

表 2-7 响应

代 码	说 明
肯定的完成答复	
211	系统状态或系统回答
214	帮助报文
220	服务就绪
221	服务关闭传输通道
250	请求命令完成
251	用户不是本地的，报文将被转发
肯定中间的服务	
354	开始邮件输入
瞬态否定的完成答复	
421	服务不可用
450	邮箱不可用
451	命令异常终止：本地错误
452	命令异常终止：存储空间不足
永久性否定的完成答复	
500	语法错误，命令无法识别
501	参数或变量中有语法错误
502	命令未执行
503	命令序列不正确
504	命令暂时未执行
550	命令未执行，邮箱不可用
551	用户不是本地的
552	所请求的动作异常终止，超出存储位置
553	所请求动作未发生，不允许使用邮箱名
554	事务失败

2. 客户发送 HELO 报文使用自身域名地址来识别自身。在这一步将客户的域名告知服务器是必要的。

3. 服务器以代码 250（请求命令完成）响应，或者其他依情况而定的代码。

邮件传输 在 SMTP 客户和服务器的连接建立后，单个报文可以在一个发信人和多个收信人之间交换。这一阶段涉及八个步骤。如果有一个以上收信人，那么重复第三步和第四步。

1. 客户发送 MAIL FROM 报文以介绍发信人。它包括发信人的邮件地址（邮箱和域名）。在这一步必须向发信人提供回信地址，回信地址用来返回错误以及报告报文。

2. 服务器用代码 250 或其他适当的代码进行响应。

3. 客户发送 RCPT TO（收信人）报文，包含了收信人的邮箱地址。

4. 服务器用代码 250 或其他适当的代码进行响应。

5. 客户发送 DATA 报文初始化报文传输。

6. 服务器用代码 354（开始邮件输入）或其他适当的代码响应。

7. 客户以连续行的形式发送报文内容。每行用两个行结束标记（回车和换行）终止。报文用

一个只包含一个句点的行终止。

8. 服务器用代码 250（OK）或其他适当的代码进行响应。

连接终止 在报文成功传送后，客户终止连接。这个阶段涉及两步。

1. 客户发送 QUIT 命令。

2. 服务器用代码 221 或其他适当的代码进行响应。

例 2.13 为了给出邮件传输的三个阶段，我们用图 2-23 给出了上文中的所有步骤。图中，在数据传输阶段，我们按照信封、头部以及主体将报文分开。注意图中的步骤在每次电子邮件传输中都重复了一遍：一次是从电子邮件发信人到本地邮件服务器，另一次是从本地邮件服务器到远程邮件服务器。本地邮件服务器在接收整个电子邮件之后可能会进行缓存并在另一个时间发送到远程邮件服务器。

报文访问代理：POP 和 IMAP

邮件传递的第一阶段和第二阶段使用 SMTP。但是，因为 SMTP 是一个推（push）协议，第三阶段不使用 SMTP。它将报文从客户推入服务器。换言之，大量数据（报文）的方向是从客户到服务器。另一方面，第三阶段需要一个拉（pull）协议。客户机必须从服务器拉出报文。大量数据的方向是从服务器到客户。因此，第三阶段使用报文访问代理协议。

目前有两种报文访问代理协议：邮局协议版本 3（POP3）和因特网邮件访问协议版本 4（IMAP4）。图 2-22 显示了这两种协议的位置。

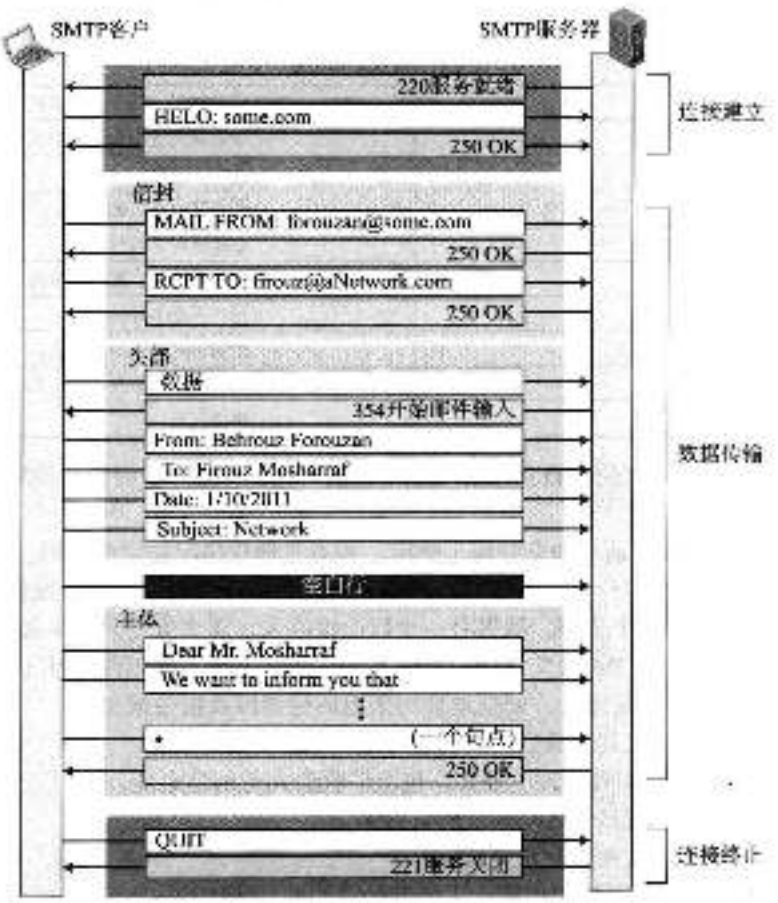


图 2-23 例 2.13

POP3

邮局协议版本 3（Post Office Protocol version 3，POP3）比较简单，但是在功能上受到一定的限制。客户端 POP3 软件安装在收信人的计算机中，服务器 POP3 软件安装在邮件服务器中。

用户需要从邮件服务器的邮箱中下载邮件时，客户端发起邮件访问操作。客户端开启与服务器 110 端口之间的 TCP 连接，然后发送用户名和口令来访问邮箱。用户就可以逐条列出和读取邮件信息了。图 2-24 说明了一个使用 POP3 下载邮件的例子。不像本章其他的图，我们将客户放在了右手边，因为电子邮件收信人（Bob）正在运行客户进程从远程邮件服务器拉报文。

POP3 有两种模式：删除模式和保存模式。在删除模式中，当邮件从邮箱中读取以后就会从邮箱中删除该邮件。在保存模式中，邮件经过读取以后仍然保存在邮箱中。当用户在固定的计算机上工作时，能够在阅读和回复邮件后保存并组织接收到的邮件，此时通常使用删除模式。当用户远离他的主计算机（如在笔记本上）访问邮件时，通常应该使用保存模式。此时可以阅读邮件，但是邮件通常仍然会保存在系统中以备日后读取和组织。

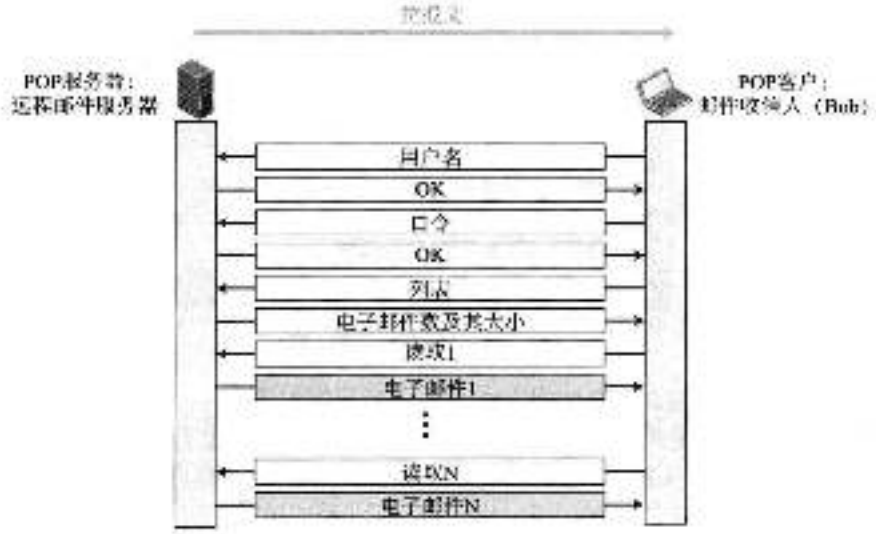


图 2-24 POP3

IMAP4

另外一个邮件访问协议是因特网邮件访问协议版本 4（Internet Mail Access Protocol version 4，IMAP4）。IMAP4 和 POP3 相似，但是有更多特点，功能更强并更复杂。

POP3 在很多方面存在缺点，它不允许用户在服务器上组织邮件；用户在服务器上不能有不同的文件夹。同时，POP3 不允许用户在下载邮件之前部分地查看邮件的内容。IMAP4 提供下列额外的功能：

- 用户在下载电子邮件之前，可以检查电子邮件头部。
- 用户在下载电子邮件之前，可以读取电子邮件内容中的特定字符串。
- 用户可以部分地下载电子邮件。这在带宽受限制而电子邮件包含了需要高带宽的多媒体信息时特别有用。
- 用户可以在邮件服务器上创建或删除邮箱，也可以改变邮箱的名字。
- 用户可以在文件夹中创建邮箱的层次结构以用于邮件存储。

MIME

电子邮件有一个简单的结构。但是，它的简单是有代价的。它只能发送使用 NVT 7 位 ASCII 格式的报文。换言之，它有一些限制。例如，它不能使用其他语言（如法文、德文、希伯来文、俄

文，中文以及日文）。另外，它不能用来发送二进制文件或音频以及视频数据。

多用途因特网邮件扩充（Multipurpose Internet Mail Extensions，MIME）是一个辅助协议，它允许非 ASCII 数据通过电子邮件传送。MIME 在发送方将非 ASCII 数据转换成 NVT ASCII 数据，并将其传递给 MTA 客户机通过因特网发送出去。在接收方再转换到原来的数据。

可以将 MIME 想象为一组软件功能，它能够将非 ASCII 数据转换成 ASCII 数据，以及进行相反的转换。如图 2-25 所示。



图 2-25 MIME

MIME 头部

MIME 定义了五种头部，将 MIME 头加在原来电子邮件的头部以定义转换参数：

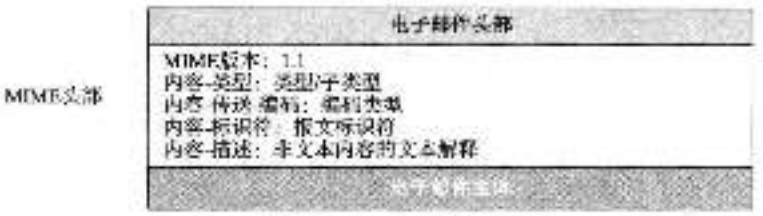


图 2-26 MIME 头部

MIME 版本 这个头部定义 MIME 使用的版本，当前版本是 1.1。

内容-类型 这个头部定义报文主体使用的数据类型。内容类型和内容子类型用一个斜杠分隔开。根据子类型的不同，头部还可以包含其他一些参数。MIME 允许七种不同的数据类型，见表 2-8。

内容-传送-编码 这个头部定义了一些方法，它们用来将传输的报文编码为 0 和 1。表 2-9 列出了五种类型的编码方法。

表 2-8 MIME 中的数据类型和子类型

类 型	子 类 型	说 明
文本	普通	无格式
	HTML	HTML 格式（见附录 C）
多部分	混合	主体包含不同数据类型的有序部分
	并行	向上机无序
	摘要	与混合子类型相似，但默认的是报文/RFC822
	可选择的	相同报文的不同版本部分
报文	RFC822	主体是一个封装的报文
	部分	主体是一个较大报文的一部分
	外部主体	主体是另一个报文的指针
图像	JPEG	JPEG 格式的图像
	GIF	GIF 格式的图像
视频	MPEG	MPEG 格式的视频信号

（续）

类 型	子 类 型	说 明
音频	基本	8kHz 的单声道语音编码
应用	PostScript	Adobe PostScript
	8 位组流	一般的二进制数据（8 位字节）

表 2-9 内容-传送-编码方法

类 型	说 明	类 型	说 明
7 位	NVT ASCII 字符，每行小于 1000 字符	Base64	6 位数据块被编码成 8 位 ASCII 字符
8 位	非 ASCII 字符，每行小于 1000 字符	引用中可打印的	非 ASCII 字符被编码成等号后面跟随一个 ASCII 码
二进制	长度不限的 ASCII 字符		

最后两个编码方法很有趣。在 Base64 编码中，数据作为位字符串，首先被分割为 6 位块，如图 2-27 所示。

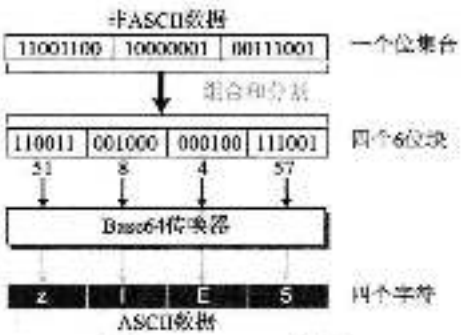


图 2-27 Base64 转换

根据表 2-10 将每个 6 位组转换成一个 ASCII 字符。

表 2-10 Base64 转换表

数值	编码	数值	编码	数值	编码	数值	编码	数值	编码	数值	编码
0	A	11	L	22	W	33	h	44	s	55	3
1	B	12	M	23	X	34	i	45	t	56	4
2	C	13	N	24	Y	35	j	46	u	57	5
3	D	14	O	25	Z	36	k	47	v	58	6
4	E	15	P	26	a	37	l	48	w	59	7
5	F	16	Q	27	b	38	m	49	x	60	8
6	G	17	R	28	c	39	n	50	y	61	9
7	H	18	S	29	d	40	o	51	z	62	+
8	I	19	T	30	e	41	p	52	0	63	/
9	J	20	U	31	f	42	q	53	1		
10	K	21	V	32	g	43	r	54	2		

Base64 是一个冗余编码方案；就是说，每 6 位成为一个 ASCII 字符并且作为 8 位来发送。我们将有 25%的开销。如果数据绝大多数是 ASCII 字符的，一少部分是非 ASCII，我们可以使用引用中可打印的编码。在引用可打印中，如果字符是 ASCII 的，它就原样发送。如果是非 ASCII，它

就作为三个字符发送。第一个字符是等号（=）。后两个字符是该字节的十六进制码。图 2-28 给出了一个例子。在例子中，第三个字符是非 ASCII，因为它是以位 1 开始的。它被解释为两个十六进制数字（$9D_{16}$）。这个字符用三个 ASCII 字符代替（=、9 和 D）。

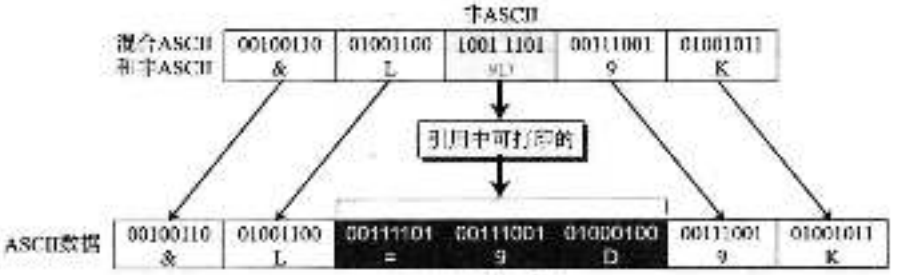


图 2-28 引用中可打印

内容-标识符 这个头部在多报文环境中唯一地标识整个报文。

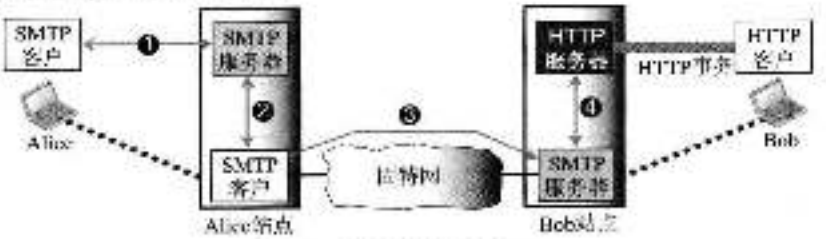
内容-描述 这个头部定义主体是否为图像、音频或视频。

基于 Web 的邮件

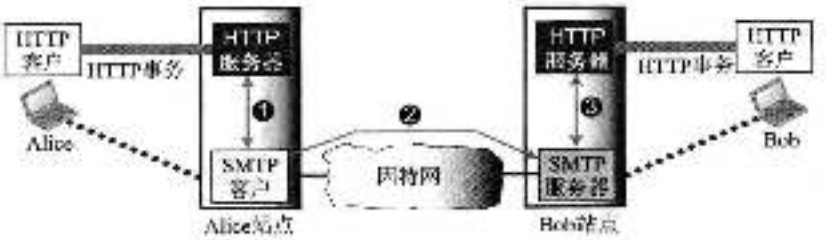
电子邮件是一种很常见的应用。目前有一些 Web 网站对任何访问者都提供服务。两个常用的网站是 Hotmail 和 Yahoo。其思想很简单。图 2-29 给出了两种情况：

情况 1

在第一种情况中，Alice 这名发信人使用传统的邮件服务器；Bob 这名收信人在基于 Web 的服务器上有一个账户。通过 SMTP，邮件从 Alice 的浏览器传输到她的邮件服务器。报文从发送邮件服务器到接收邮件服务器的传输仍然是通过 SMTP 完成的。但是，从接收服务器（Web 服务器）到 Bob 的浏览器是通过 HTTP 完成的。换言之，HTTP 是经常使用的，而不是 POP3 或 IMAP4。当 Bob 需要取回他的电子邮件时，他发送一个 HTTP 请求报文给网站（比如 Hotmail）。网站发送一个由 Bob 填写的表格，它包括了登录名以及口令。如果登录名和口令匹配，电子邮件会从 Web 服务器以 HTML 格式传送到 Bob 的浏览器。



情况1：只有收信人使用HTTP



情况2：发信人和收信人都使用HTTP

图 2-29 基于 Web 的电子邮件，情况 1 和情况 2

情况 2

在第二种情况中，Alice 和 Bob 都采用 Web 服务器，但是不必是同一个服务器。Alice 使用 HTTP 事务发送报文给 Web 服务器。Alice 使用 Bob 的邮箱作为 URL 发送一个 HTTP 请求报文给他的 Web 服务器。Alice 站点的服务器将这个报文传递给 SMTP 客户，并使用 SMTP 协议将其发送给 Bob 站点的服务器。Bob 使用 HTTP 事务接收报文。然而，从 Alice 站点服务器到 Bob 站点服务器的报文仍然使用 SMTP 协议。

电子邮件安全

本章讨论的协议其本身不提供安全。然而，可以使用两种为电子邮件系统特别设计的应用层安全来保护电子邮件的交换。我们讨论基本网络安全之后将在第 10 章讨论良好隐私（Pretty Good Privacy，PGP）以及安全多用途互联网邮件扩展（Secure/Multipurpose Internet Mail Extensions，S/MIME）。

2.3.4 TELNET

一个服务器程序可以为相应的客户程序提供特定的服务。例如，FTP 服务器被设计用来让 FTP 客户存储或获取服务器站点上的文件。然而，我们不可能对每一种所需要的服务都有一个客户-服务器对：服务的数量很快就变得难以处理。这个想法是不可扩展的。另一个解决方法是对一组常见情景使用特定的客户-服务器程序，但是要有一些通用的客户-服务器程序，这些程序允许客户端的用户登录到服务器站点上的计算机并且使用那里的可用服务。例如，如果一个学生需要使用她大学实验室的 Java 编译器服务器。这名学生可以使用一个客户登录程序登录到大学服务器并且使用大学的编译器程序。我们把这种通用客户-服务器程序对称为远程登录（remote login）应用。

原始的远程登录协议之一是 TELNET，这是终端网络（TErminaL NETwork）的缩写。尽管 TELNET 需要登录名和口令，但是它很容易遭到攻击，因为它用明文（非加密）发送所有数据包括口令。一个黑客可以偷听并获得登录名和密码。由于安全问题，TELNET 的使用让位于另一种协议——安全人机界面（Secure Shell，SHH）。我们将在下一节描述这个协议。尽管 TELNET 几乎被 SSH 所替代，但是我们简要讨论 TELNET 的原因如下：

1. TELNET 的简单明文架构允许我们解释事件和一系列与登录相关的挑战，当作为远程登录协议进行服务时，这些也被用在 SSH 上。
2. 网络管理员经常使用 TELNET 进行诊断和调试。

本地与远程登录

我们首先讨论本地和远程登录，如图 2-30 所示。

当一个用户登录到本地系统，这称为本地登录（local login）。用户在终端上键入时或在工作站运行终端仿真程序时，她的击键就被终端驱动程序所接受。终端驱动程序将字符传递给操作系统。操作系统解释字符的组合，并调用所需的应用程序或实用工具。

然而，当用户想访问远程机器上的一个应用程序或实用工具时，她就需要进行远程登录（remote login）。此时就需要使用 TELNET 客户程序和服务器程序。用户将击键发送给终端驱动程序，同时本地操作系统接收这些字符，但并不解释它们。这些字符被发送到 TELNET 客户机，它将这些字符转换成网络虚拟终端（Network Virtual Terminal，NVT）（下文讨论）字符的通用字符集，然后将其传送给本地 TCP/IP 协议堆栈。

采用网络虚拟终端（NVT）形式的命令或文本通过因特网传送到远程机器的 TCP/IP 堆栈，在那里字符传递给操作系统，然后传送给 TELNET 服务器，TELNET 服务器将这些字符转换成远程计算机可以理解的字符。但是，这些字符不能直接传递给操作系统，因为远程的操作系统不能接收来自 TELNET 服务器的字符，它只能接收来自终端驱动程序的字符，解决方法是增加一个称为伪

终端驱动程序（pseudo terminal driver）的软件块，它将这些字符伪装成好像是从一个终端发来的，然后操作系统将这些字符传送给适当的应用程序。

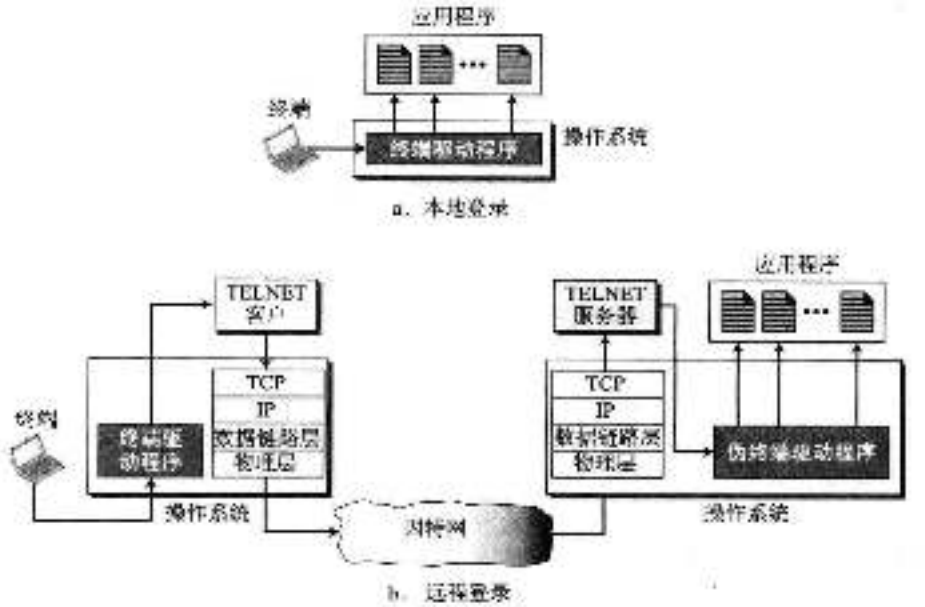


图 2-30 本地和远程登录

网络虚拟终端（NVT）

访问一个远程计算机的机制是复杂的。这是因为每一个计算机以及其操作系统都接收一种特殊的字符组合作为一个标记。例如，在运行 DOS 操作系统的计算机中，文件结束标记是 Ctrl+z，但在 UNIX 操作系统中则是 Ctrl+d。

我们现在是和异构系统打交道。如果我们想要访问世界上任何远程计算机，那么我们必须先知道将要连接的计算机是什么类型，我们还必须安装那个计算机所用的终端仿真程序。TELNET 解决这个问题的方法是定义一个通用的接口，称为网络虚拟终端（Network Virtual Terminal，NVT）字符集。通过这个接口，客户 TELNET 将来自本地终端的字符（数据或命令）转换成 NVT 形式，然后传递给网络。而服务器 TELNET 将来自 NVT 形式的数据或命令转换成远程计算机可接受的形式。图 2-31 表示了这一概念。

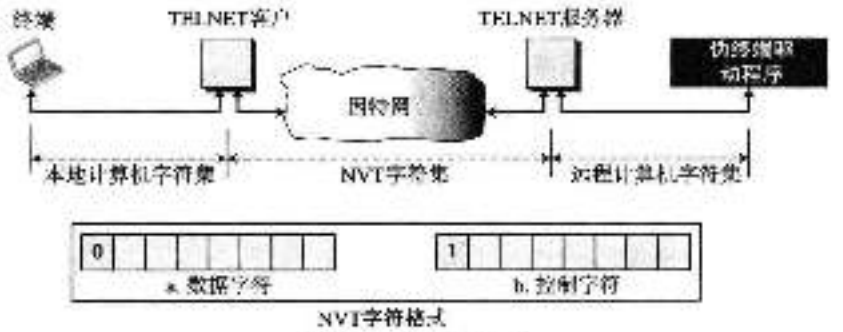


图 2-31 NVT 的概念

NVT 使用两个字符集：一个是数据字符，另一个是控制字符。如图 2-31 所示，两者都是 8 位

的字符集。对于数据，NVT 通常使用称为 NVT ASCII 的字符集。这是一个 8 位字符集，其中 7 个最低位与 ASCII 是一样的，而最高位是 0。在计算机之间发送（从客户到服务器，反之亦然）控制字符，NVT 使用一个 8 位的字符集，其最高位是 1。

选项

TELNET 允许客户与服务器在使用服务之前或使用过程中协商选项。对更加复杂的终端用户，这些选项还提供额外的特性。使用较简单终端的用户可以使用默认的特性。

用户接口

操作系统（比如 UNIX）定义了带有用户友好命令的接口。表 2-11 给出了命令集的一个例子。

表 2-11 接口命令的例子

命 令	含 义	命 令	含 义
open	连接远程计算机	Set	设置操作参数
close	关闭连接	Status	显示状态信息
display	显示操作参数	Send	发送特殊字符
mode	切换至行方式或字符方式	quit	退出 TELNET

2.3.5 安全 Shell

尽管如今的安全 Shell（Secure Shell，SSH）是一个可以用于远程登录和文件传输等多用途的安全应用程序，但是它原先是被设计来替代 TELNET 的。有两个版本的 SSH：SSH-1 和 SSH-2，它们是完全不兼容的。第一个版本 SSH-1 由于安全漏洞现在已被废止。在这一节，我们只讨论 SSH-2。

组件

SSH 是一个有三个组件的应用层协议，如图 2-32 所示。



图 2-32 SSH 组件

SSH 应用层协议（SSH-TRANS）

由于 TCP 不是安全传输层协议，SSH 首先使用在 TCP 顶部创建安全链路的协议。这个新层是一个独立协议，称为 SSH-TRANS。当执行这个协议的程序被调用时，客户和服务器首先使用 TCP 协议建立一个不安全的连接。然后，它们交换几个安全参数在 TCP 顶部建立安全信道。我们将在第 10 章讨论传输层安全，但是，这里我们简要列出这个协议提供的服务：

1. 隐私或交换报文的保密性。
2. 数据完整性，这意味着客户和服务器交换的报文不会被入侵者改变。
3. 服务认证，这意味着客户现在确认服务器正是其所声称的那台服务器。
4. 报文压缩，提高了系统的效率并且使得进攻更难。

SSH 认证协议（SSH-AUTH）

在客户与服务器之间的安全信道建立以及客户端认证服务器之后，SSH 可以调用另外一个流程，它为服务器对客户进行认证。SSH 中的客户认证过程与安全套接层（SSL）的做法非常相似，我们将在第 10 章讨论这个内容。这一层定义了很多与 SSL 中相似的认证工具。认证从客户开始，客户发送一个请求报文给服务器。请求包括用户名、服务器名以及请求数据。服务器或者以一个成功报文进行响应，它确认了客户被认证；或者以一个失败报文进行响应，它表示需要以一个新的请求报文重复过程。

SSH 连接协议（SSH-CONN）

在安全信道建立以及客户和服务器相互认证之后，SSH 可以调用一个执行第三个协议的软件，那就是 SSH-CONN。SSH-CONN 协议提供的一个服务是复用。SSH-CONN 采用前两层协议创建安全信道并

允许客户在其上创建多个逻辑信道。每个信道可以用于各自不同的目的，比如远程登录、文件传输等。

应用

尽管SSH经常被认为是TELNET的替代协议，但实际上SSH是一个提供客户和服务器之间安全连接的通用协议。

SSH用于远程登录

许多免费和商业应用使用SSH来远程登录。其中，我们可以注意到Simon Tatham编写的PuTTy，它是一个可以用来远程登录的SSH客户程序。另外一个应用是Tectia，可以用在多个平台上。

SSH用于文件传输

建立在SSH上用于文件传输的一个应用程序是安全文件传输程序（secure file transfer program，sftp）。sftp应用程序使用SSH提供的信道来传输文件。另一个常见的应用称为安全拷贝（secure copy，scp）。这个应用使用与UNIX拷贝命令cp相同的格式来拷贝文件。

端口转发

SSH协议所提供的一个有趣服务是端口转发（port forward）。我们可以使用SSH中可用的安全信道来访问一个不提供安全服务的应用程序。如TELNET和简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）这类的应用，可以使用SSH端口转发机制的服务，这些应用将在稍后讨论。SSH端口转发机制创建了一个隧道，属于其他协议的报文可以穿过这个隧道。由于这个原因，这个机制有时称为SSH隧道。图2-33给出了用于安全FTP应用的端口转发概念。

FTP客户可以使用本地站点的SSH客户来与远程站点的SSH服务器创建安全连接。任何从FTP客户到FTP服务器的请求都被携带通过SSH客户和服务器提供的隧道。任何从FTP服务器到FTP客户的请求都被携带通过SSH客户和服务器提供的隧道。



图2-33 端口转发

SSH分组的格式

图2-34给出了SSH协议使用的分组格式。

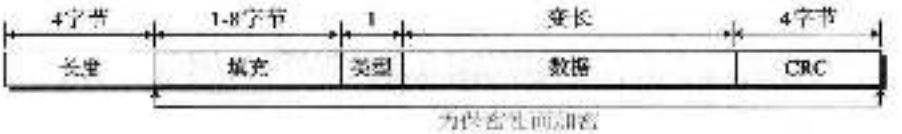


图2-34 SSH分组格式

长度字段定义了分组的长度但是并不包括填充字符。1到8字节的填充字符被加入到分组中，使得攻击更加困难。循环冗余校验字段用于错误检测。类型字段指明在不同SSH协议中使用的分组类型。数据字段是不同协议中由分组传输的数据。

2.3.6 域名系统

我们最后讨论的客户-服务器程序已经被设计来帮助其他应用程序。为了识别一个实体，TCP/IP协议使用IP地址唯一地确定一台主机到因特网的连接。然而，人们更喜欢使用名字而不是数字地址。所以，因特网需要一种能够将名字映射地址的目录系统。这就像电话网络。一个电话网络被设计成使用电话号码，而不是名字。人们可以保留一个私人文件将名字映射到相应的电话上或者可以让电话号码簿来做这件事情。我们将讨论这个因特网中的号码簿系统是如何将名字映射到IP地址的。

由于今天的因特网是如此巨大，一个中心号码簿系统不能承担所有的映射工作。此外，如果一个中心计算机失效，整个通信网络都将瘫痪。一个更好的解决方法是将信息分布在世界上的很多台计算

机中。采用这种方法，需要映射的计算机可寻找到最近一台持有所需信息的计算机。域名系统（Domain Name System，DNS）就是采取这种方法。本章先讨论 DNS 的概念和思想，然后描述 DNS 协议本身。

图 2-35 给出了 TCP/IP 是如何使用 DNS 客户端和服务器来将名字映射到地址的。一个用户想要使用文件传输客户端来访问运行在远程主机上相应的文件传输服务器。用户只知道文件传输服务器的名字，如 afilesource.com。然而，TCP/IP 协议族需要文件传输服务器的 IP 地址来建立连接。以下六步将主机名映射到 IP 地址：

1. 用户将主机名传递给文件传输客户端。
2. 文件传输客户端将主机名传递到 DNS 客户端。
3. 在启动后，每一台电脑都知道 DNS 服务器的地址。DNS 客户端使用已知 DNS 服务器 IP 地址向 DNS 服务器发送附有查询的报文，查询报文使用已知的 DNS 服务器 IP 地址给出文件服务器名。
4. DNS 服务器进行响应，发回客户想要获得的文件传输服务器的 IP 地址。
5. DNS 客户端将 IP 地址传递到文件传输服务器。
6. 现在，文件传输客户端使用接收到的 IP 地址来访问文件传输服务器。

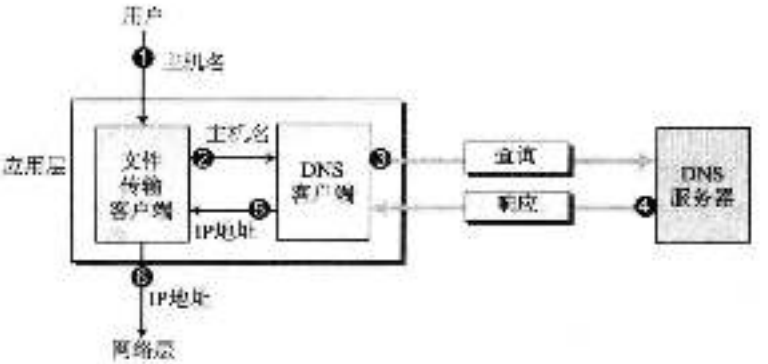


图 2-35 DNS 的用途

请注意，访问因特网的目的就是在文件传输客户端和服务器端建立连接，但是在这之前，需要在 DNS 客户端和 DNS 服务器端之间建立另一个连接。换言之，我们在这种情况下至少需要两个连接。第一个连接用于将名称映射到 IP 地址；第二个连接用于传输文件。稍后我们将看到映射可能需要一个以上连接。

名字空间

为实现无二义性，分配给机器的名字必须从名字空间中仔细选择。该名字空间完全控制对名字和 IP 地址的绑定。换言之，因为地址是唯一的，所以名字也必须是唯一的。名字空间（name space）将每一个地址映射到一个唯一的名字，它可以按两种方式进行组织：平面的和层次的。在平面名字空间（flat name space）中，一个名字分配给一个地址。空间的名字是一个无结构的字符序列。名字之间可能有也可能没有公共部分；即使有公共部分，也没有实际含义。平面名字空间的主要缺点是它必须集中控制才能避免二义性和重复，因而不能用于如因特网这样的大规模系统中。在层次名字空间（hierarchical name space）中，每一个名字由几个部分组成。第一部分可以定义组织的性质，第二部分可以定义一个组织的名字，第三部分可以定义组织的部门，等等。在这种情况下，分配和控制名字空间的机构就可以分散化。中央管理机构可以分配名字的一部分，这部分定义组织的性质和组织的名字。名字其他部分的分配可以交给这个组织自身。这个组织可以给名字加上后缀（或前缀）来定义主机或者其他资源。这个组织机构的管理机构不必担心为一个主机选择的后缀会被其他组织机构所采用，因为即使地址的某一部分相同，整个地址也是不同的。例如，假定两所机构把

他们的计算机都叫做 caesar。第一个机构由中央管理结构分配的名字是 first.com。第二个机构由中央管理结构分配的名字是 second.com。当每个机构在已有的名字上加上名字 caesar 后，得到了两个不同的名字：caesar.first.com 和 caesar.second.com。这些名字是唯一的。

域名空间

为了获得层次名字空间，设计了域名空间（domain name space）。在这种设计方式中，所有的名字由根在顶部的倒置树结构定义。该树最多有 128 级：0 级（根结点）至 127 级（见图 2-36）。

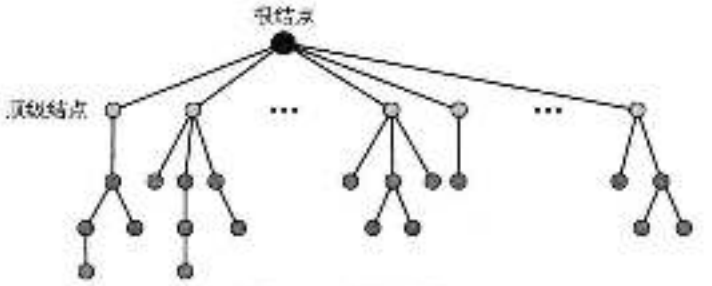


图 2-36 域名空间

标签 树上的每一个结点有一个标签。标签是一个最多为 63 个字符的字符串。根结点的符号是空字符串（空串）。DNS 要求每一个结点的子结点（从同一个结点分支出来的结点）有不同的标签，这样就确保了域名的唯一性。

域名 树上的每一个结点都有一个域名。一个完整的域名（domain name）是用点（.）分隔的标签序列。域名总是从结点向上读到根结点。最后一个标签是根结点的标签（空）。这表示一个完整的域名总是以一个空符号结束，也意味着最后一个字符是一个点（.），因为空字符串表示什么也没有。图 2-37 给出了某些域名。

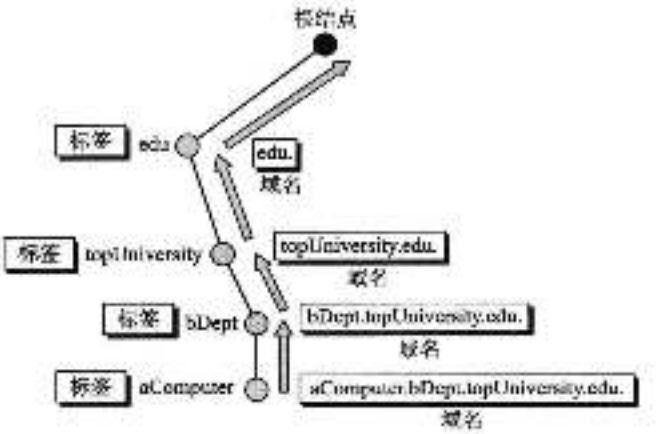


图 2-37 域名和标签

如果一个标签以一个空字符串结束，则它就称为全称域名（fully qualified domain name，FQDN）。名字必须以空标签结束，但是由于空标签表示没有东西，所以这种标签以一个点结束。如果一个标签不是以空字符串结束，则称为部分域名（partially qualified domain name，PQDN）。部分域名起始于一个结点，但没有到达根结点。当这个需要解析的名字属于和客户机相同的站点时使用部分域名。在这种情况下，解析程序能够提供省略的部分，称为后缀（suffix），以创建 FQDN。

域

域（Domain）是域命名空间的子树。域的名称是了树顶端结点的名称。图 2-38

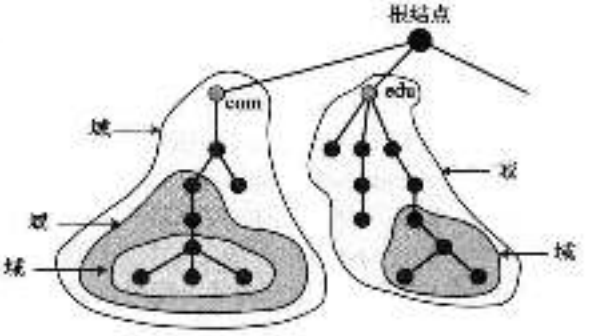


图 2-38 域

给出了一些域。注意，域本身仍可能被分割成多个域。

名字空间的分布

必须将域名空间所包含的信息存储起来。然而，只使用一台计算机存储如此大容量的信息，效率非常低和不可靠。效率低的主要原因是因为响应来自世界各地的请求会给系统造成非常大的负荷。不可靠的原因是因为任何故障将使数据无法访问。

名字服务器的层次结构 解决这些问题的办法是将信息分布在多台称为 DNS 服务器（DNS server）的计算机中。一种方法是将整个空间划分为多个基于第一级的域。换言之，让根结点保持不动，但创建许多与第一级结点一样多的域（子树）。这样创建的域会很大。DNS 允许将域进一步划分成更小的域（子域）。每一台服务器负责（授权的）一个大的域或者较小的域。换言之，与建立名字的层次结构一样，也建立了服务器的层次结构（见图 2-39）。

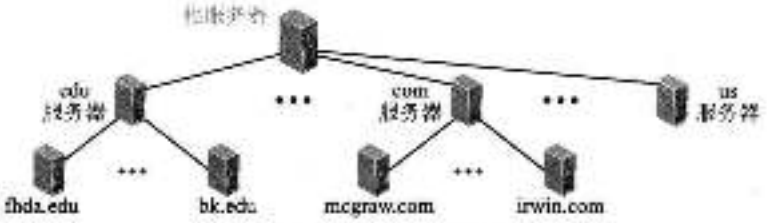


图 2-39 名字服务器的层次结构

区域

因为完整的域名层次结构不能保存在单一的服务器上，所以它被分散在多个服务器上。一个服务器负责或授权的方位称为区域（Zone）。我们可以将一个区域定义为整个树中的一个连续部分。如果服务器负责一个域，而且这个域并没有进一步被划分为更小的子域，此时“域”和“区域”是相同的。服务器有一个数据库，称为区域文件，它保存这个域里所有结点的信息。然而，如果服务器将它的域划分为多个子域，并将其部分授权委托给其他服务器，那么“域”与“区域”就不同了。在子域结点的信息会保存在较低层次的服务器中，原来的服务器则保存到这些较低层次服务器的某种参照。当然，原来的服务器并不是完全不负责任，它仍然拥有一个区域，只是将详细的信息保存在较低层次的服务器上（见图 2-40）。

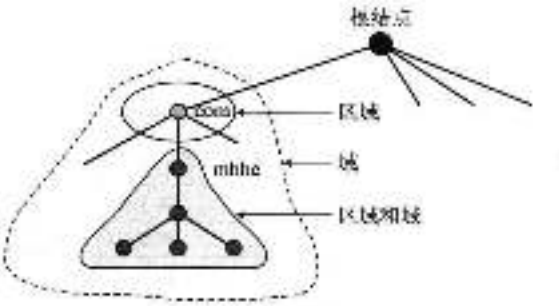


图 2-40 区域

根服务器

根服务器（root server）的区域由整棵树组成。根服务器通常不保存关于域的任何信息，只是将其授权委托给其他服务器，并保持与这些服务器的参照关系。目前有多个根服务器，每一台都覆盖了整个域名空间。这些服务器分布在世界各地。

主服务器和辅助服务器 DNS 定义了两种类型的服务器：主服务器和辅助服务器。主服务器（primary server）是指存储了授权区域有关文件的服务器。它负责创建、维护和更新区域文件，并将区域文件存储在本地磁盘中。

辅助服务器（secondary server）负责从另一个服务器（主服务器或辅助服务器）传输一个区域的全部信息，并将文件存储在它的本地磁盘中。辅助服务器既不创建也不更新区域文件。如果需要更新，则必须由主服务器来完成，由主服务器发送更新的版本到辅助服务器中。

主服务器和辅助服务器对它们所服务的区域都有控制权。这种设计思想并不是将辅助服务器置于一

个较低的授权层次上，而是为了建立数据的冗余备份，这样当一台服务器出现故障时，另一台服务器可以继续为客户机服务。注意，一台服务器可能是某个特定区域的主服务器，同时也是另一个区域的辅助服务器。所以，当提到一个服务器作为主服务器或者辅助服务器时，需要弄清楚所指的是哪个区域。

主服务器生成从域参文件中装载所有信息，辅助服务器从主服务器中装载信息。

因特网中的 DNS

DNS 是一种可以在不同平台上使用的协议。在因特网中，域名空间（树）被划分为三个部分：通用域、国家域和反向域。然而，由于因特网的快速增长，掌握反向域变得极其复杂，它可以用来在给定 IP 地址的情况下找到主机名。反向域现在已经废止（见 RFC 3425）。因此我们集中精力于前两个。

通用域

通用域（generic domain）按照已经注册主机的一般行为对主机进行定义。树中的每一个结点定义一个域，它是到域名空间数据库的一个索引（见图 2-41）。

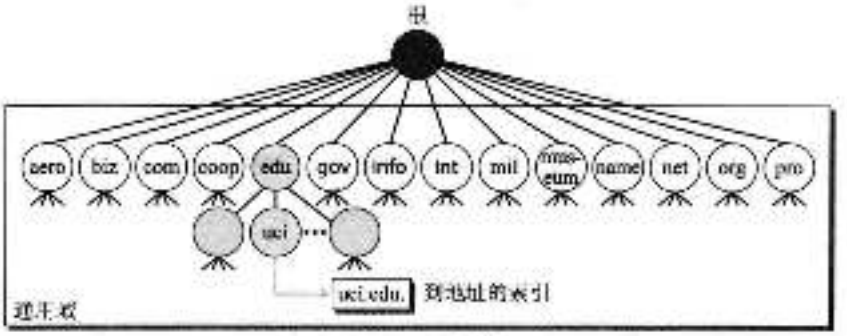


图 2-41 通用域

我们从这棵树可以看出，在通用域的一个层次允许有 14 个可能的标签，这些标签描述了表 2-12 中列出的机构类型。

表 2-12 通用域标签

标 签	描 述	标 签	描 述
aero	航空航天公司	int	国际机构
biz	商业或公司	mil	军事机构
com	商业机构	museum	博物馆
coop	协作商业组织	name	私人姓名（个人的）
edu	教育机构	net	网络支持中心
gov	政府机构	org	非盈利组织
info	信息服务提供商	pro	专业组织

国家域

国家域（country domain）部分使用双字母的国家缩写（例如，us 代表美国），第二级标号可以是机构，或者更具体一些，由各个国家自己制定。例如，美国使用州的缩写作为国家域的子域划分（例如 ca.us）。图 2-42 列出了国家域部分。地址 uci.ca.us 可以理解为美国加州的加州大学欧文分校。

解析

将名字映射为地址或者将地址映射为名字的过程，称为名字-地址解析（name-address resolution）。DNS 是一个客户/服务器应用程序。需要将地址映射为名字或者将名字映射为地址时，主机要调用一个称为**解析程序**（resolver）的 DNS 客户程序。解析程序用一个映射请求访问最近的

一个 DNS 服务器。如果服务器含有该信息，它就满足解析程序的请求；否则，它将解析程序交付给其他的服务器，或者查询其他的服务器来提供这种服务。在解析程序接收到映射后，它解释这一响应，以确定它是一个真正的解析还是一个差错，最后将结果传递给发送这一请求的进程。一个解析可能是递归的或迭代的。

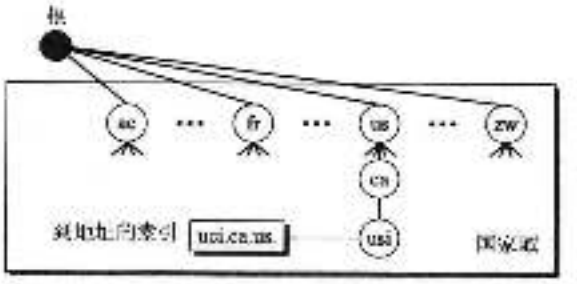


图 2-42 国家域

递归解析

图 2-43 给出了递归解析的一个简单例子。我们假设一个在名为 **some.anet.com** 的主机上运行的程序需要找到另一台名为 **engineering.mcgraw-hill.com** 的主机的 IP 地址从而发送报文。源主机被连接到 Anet ISP；目的主机被连接到 McGraw-Hill 网络。

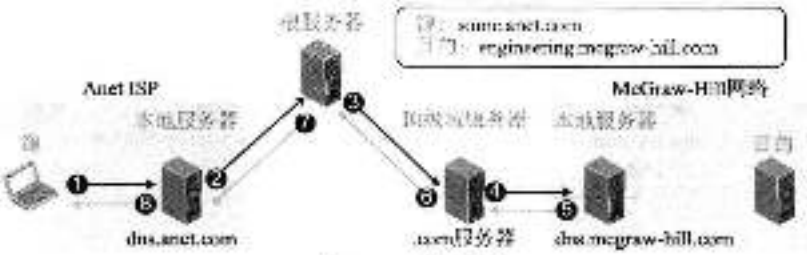


图 2-43 递归解析

源主机上的应用程序调用 DNS 解析程序（客户端）来找到目的主机的 IP 地址。解析程序不知道这个地址，便把请求发送到运行在 Anet ISP 端（事件 1）的本地 DNS 服务器（例如，dns.anet.com）。我们假设这个服务器也不知道目的主机的 IP 地址。它发送一个请求到 DNS 根服务器。本地 DNS 服务器应该知道根服务器的 IP 地址（事件 2）。根服务器通常不保存名字到 IP 地址的映射，但是根服务器至少知道每个顶级域中的一台主机（在此情况下，服务器负责 com 域）。查询被发送到这台顶级域服务器（事件 3）。我们假设这台服务器不知道这个特定目的的名字-地址映射，但是它知道本地 McGraw-Hill 公司的 DNS 服务器的 IP 地址（例如，dns.mcgraw-hill.com）。查询被发送到这台服务器（事件 4），它知道目的主机的 IP 地址。现在 IP 地址被返回给顶层 DNS 服务器（事件 5），然后返回给根服务器（事件 6），然后返回给 ISP DNS 服务器，它可能缓存这个地址以待将来的查询（事件 7），并且最终返回给源主机（事件 8）。

迭代解析

在迭代解析中，每个不知道映射的服务器将下一台服务器的 IP 地址发回到请求查询的主机上。图 2-44 给出了与图 2-43 相同场景下的迭代解析中的信息流。通常迭代解析发生在两台本地服务器之间；原始解析程序从本地服务器得到最终答案。注意事件 2、4 和 6 给出的报文包含了相同的查询。然而，事件 3 给出的报文包含顶层域服务器的 IP 地址。事件 5 给出的报文包含 McGraw-Hill 本地 DNS 服务器的 IP 地址，事件 7 给出的报文包含目的 IP 地址。当 Anet 本地 DNS 服务器接收到目的 IP 地址，它将其发送到解析程序（事件 8）。

高速缓存

每当服务器接收到查询一个不属于自己域的名字时，它需要搜索自己的数据库以查找一台服务器的 IP 地址。缩短这一查询时间能提高效率。DNS 服务器使用一种称为高速缓存（cache）的机制处理这一问题。当一个服务器向另一个服务器请求映射并得到回应时，在将该回应发送给客户端之

前，先将这一信息存储在高速缓存中。如果同一客户端或者另一个客户端请求同一映射时，它会检查其高速缓存并解决这一问题。然而，为了表明客户这一响应来自于高速缓存而不是来自于授权的信息源，该服务器会将这一响应标志为非授权性的。

高速缓存能够加快解析过程，但仍存在问题。如果同一台服务器长时间缓存一个映射，可能会发送给客户端一个过期的映射。为了防止这种情况，使用了两种技术。第一种，授权服务器总是将称为生存时间（TTL）的信息添加在映射上。生存时间定义了接收服务器可以将信息放入高速缓存的时间（以秒计）。超过这一时间，该映射就变为无效，而任何查询必须再次发送到授权服务器。第二种，DNS 要求每一台服务器对每一个映射保留一个 TTL 计数器。高速缓存会定期检查，并清除掉 TTL 已经过期的那些映射。

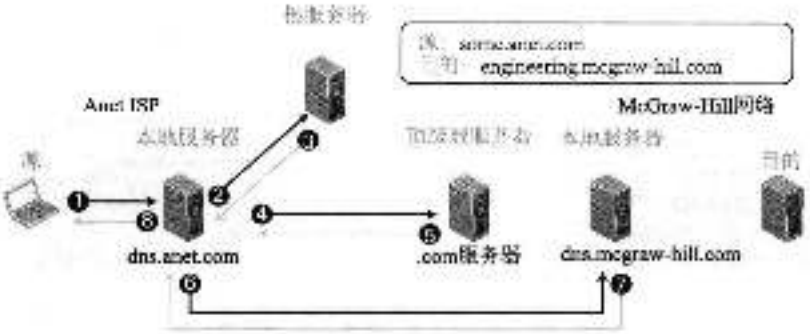


图 2-44　迭代解析

资源记录

与某个服务器相关的区域信息以资源记录（resource record）集的形式实现。换言之，域名服务器存储了资源记录的数据库。资源记录是一个 5 元组结构，如下所示：

（域名，类型，类别，TTL，数值）

域名字段标识资源记录。数值定义了保存的关于域名的信息。TTL 定义了信息有效的时间，以秒为单位。类别定义了网络的类型；我们只对类型 IN（因特网）感兴趣。类型定义了数值应当被如何解释。表 2-13 列出了常见类型以及每种类型的数值解释。

表 2-13　类型

类　型	数值的解释	类　型	数值的解释
A	一个 32 位 IPv4 地址（见第 4 章）	SOA	标记区域的开始
NS	标志区域的授权服务器	MX	转置邮件到邮件服务器
CNAME	为主机的官方名称定义一个别名	AAAA	一个 IPv6 地址（见第 4 章）

DNS 报文

为了获取关于主机的信息，DNS 使用两种类型的报文：查询报文（query）和响应报文（response）。两种类型报文都有相同的格式，如图 2-45 所示。

我们简要地讨论 DNS 报文的字段。标识字段用来匹配对查询的响应。标记定义了报文是查询报文还是响应报文。它也包含差错状态。头部接下来四个字段定义了报文中每个记录类型的数目。查询部分包含在查询报文中并且在响应中被重复，它包含一个或多个问题记录。它在查询和响应报文中都会出现。响应部分包含了一个或多于一个资源记录。它只出现在响应报文中。授权部分给出一个或多个负责查询的授权服务器的信息（域名）。额外信息部分提供了可能帮助解析程序的额外信息。

例 2.14　在 UNIX 和 Windows 系统中，nslookup 命令可以用来获取地址/域名映射。以下给出了当域名被给出时我们如何获得地址。

```
$nslookup www.forouzan.biz
Name:    www.forouzan.biz
Address: 198.170.240.179
```



注：
查询报文仅仅包含查询部分。响应报文包含查询部分、响应部分，也可能包含其他两部分。

图 2-45 DNS 报文

封装

DNS 可以使用 UDP 或者 TCP 协议。在这两种情况下，服务器使用的熟知端口号是 53。当响应报文的长度小于 512 字节时，就使用 UDP。因为大多数 UDP 分组有 512 字节分组大小的限制。如果响应报文的长度大于 512 字节，则必须使用 TCP 连接。在这种情况下，可能会发生以下两种情况：

- 如果解析程序预先知道响应报文超过 512 字节，那么它必须使用 TCP 连接。例如，如果辅助名字服务器（作为客户端）需要从主服务器进行区域传输，那么必须使用 TCP 连接。因为被传输的信息通常是超过 512 字节的。
- 如果解析程序不知道响应报文的大小，那么就可以使用 UDP 端口。但是，如果响应报文超过 512 字节，那么服务器会截断这一报文。此时解析程序会开启 TCP 连接，并重复该请求，从服务器中获得完整的响应。

注册机构

新的域名是怎样加入到 DNS 中呢？这是通过注册机构（registrar）来完成的。一个熟知的商业实体是 ICANN（因特网名称和编号分配组织）。注册机构首先确认询问的域名是唯一的，然后将它输入到 DNS 数据库中。这是需要收费的。现在，有很多的注册机构，它们的名字和地址可以在如下网址中找到：

http://www.intenic.net

为了能够注册，组织机构需要给出域名服务器的主机名和 IP 地址。例如，一个名为 wonderful 的商业机构的域名服务器主机名为 ws，IP 地址为 200.200.200.5，就需要给出以下的信息给注册机构：

域名：ws.wonderful.com

IP 地址：200.200.200.5

DDNS

在设计 DNS 时，没有人预料到地址会有如此多的变化。在 DNS 中，当发生变化时，例如增加一台新主机、移动一台主机或改变一个 IP 地址时，那么这种改变就必然使 DNS 的主文件发生变化。这些类型的变化涉及手工更新。今天的因特网规模已经不允许使用这种手工操作。

DNS 主文件必须能动态更新。动态域名系统（Dynamic Domain Name System，DDNS）就是为满足这种需求而设计的。在 DDNS 中，当名字和地址之间的绑定被确定时，这些信息通常是由 DHCP

（见第4章）发送给主DNS服务器。主服务器更新这一区域。通知辅助服务器的方法可以以主动方式或者以被动方式。在主动通知方式中，主服务器向辅助服务器发送关于区域变化的报文；而在被动通知方式中，辅助服务器定期检查是否有任何变化。无论使用哪一种方式，当得到变化的通知时，辅助服务器会请求整个区域的信息（叫做区域传输）。

为了提供安全性以防止对DNS记录的非授权更改，DDNS可以使用鉴别机制。

DNS安全

DNS是因特网基础设施中最重要的系统之一：它为因特网用户提供重要的服务。很多应用，例如Web访问或电子邮件，都强烈依赖DNS的正常工作。有很多方式可以攻击DNS，这包括：

1. 攻击者可能读取DNS服务器的响应，从而发现用户最常访问的站点的名称。这种信息类型可以被用来找到用户信息。为了防止这种攻击，DNS报文需要保密（见第10章）。

2. 攻击者可能截获DNS服务器的响应并加以改变，或创建一个全新的伪造响应来将用户导向攻击者想要用户访问的站点或域。这类攻击可以使用报文起源鉴别和报文完整性来进行预防（见第10章）。

3. 攻击者可能泛洪攻击淹没DNS服务器，最终使之瘫痪。这类攻击可以采取预防拒绝服务攻击的措施。

为了保护DNS，IETF开发了一种称为DNS安全（DNS Security，DNSSEC）的技术。它使用称为数字签名（digital signature）（见第10章）的安全服务提供报文起源鉴别以及报文完整性。然而，DNSSEC不提供DNS报文机密性。在DNSSEC的说明中，没有针对拒绝服务攻击进行特定的保护。然而，缓存系统在某种程度上保护上层服务器免于这种攻击。

2.4 对等模式

我们在本章前面讨论了客户-服务器模式。我们在之前几节也讨论了一些标准客户-服务器应用。在这一节，我们讨论对等模式。第一个对等文件共享实例要追溯到1987年12月，当时Wayne Bell创建了WWIVnet，这是WWIV (World War Four)公告牌软件的网络组件。在1999年7月，Ian Clarke设计了Freenet，一种分散、抗审查的分布数据存储，目标是通过对等网络提供带有匿名保护的言论自由。

对等随着Napster（1999—2001）变得流行，这是一种在线音乐共享服务，由Shawn Fanning创建。尽管用户自由复制和分发音乐文件导致了针对Napster侵权法律诉讼，并最终导致关闭服务。但是它为之后到来的对等文件分发模型铺平了道路。Gnutella在2000年3月发布了第一个发行版。紧随其后的是FastTrack（被Kazaa使用）、BitTorrent、WinMX以及GNUnet，它们分别于2001年3月、4月、5月和11月出现。

2.4.1 P2P网络

准备共享资源的因特网用户成为对等结点并且组成一个网络。当网络中的一个对等结点有文件（例如，一个音频或视频文件）要共享，那么它使其余对等结点都能获得这个文件。一个感兴趣的对等结点可以连接到存储这个文件的计算机并下载它。在一个对等结点下载后，它能为其他对等结点提供下载。随着越来越多的对等结点加入并下载那个文件，越来越多的文件备份可以被这个组使用。由于对等结点列表可能增长也可能萎缩，因此待解决的问题是，这个模式如何记录忠诚用户以及文件的位置。为了回答这个问题，我们首先需要将P2P网络分成两类：集中式与分散式。

集中式网络

在集中式P2P网络中，目录系统——列出对等结点及它们所提供的内容——使用客户-服务器模式，但是文件存储和下载使用对等模式完成。由于这个原因，一个集中式P2P网络有时称为混合P2P网络。集中式P2P网络的一个例子是Napster。在这种网络中，一个对等结点首先在中心服

务器进行注册。之后对等结点提供自身 IP 地址以及将要分享的文件列表。为了避免系统崩溃，Napster 使用多个服务器来实现这个目的，但是我们在图 2-46 中仅给出一个服务器。

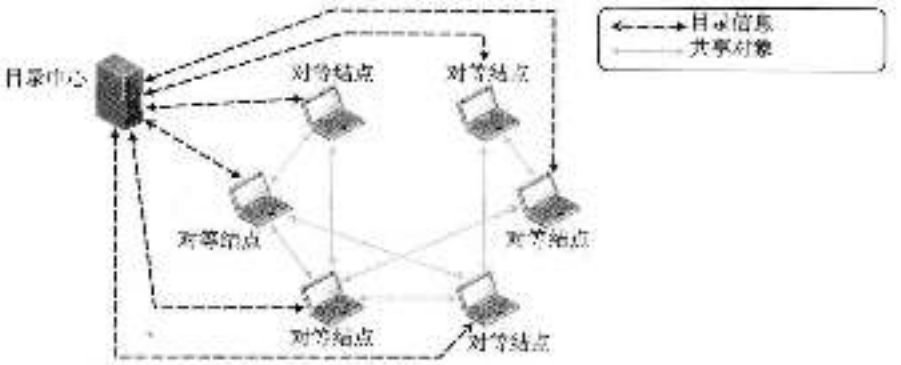


图 2-46 集中式网络

一个寻找特定文件的对等结点向中心服务器发送一个查询。服务器搜索其目录，将含有所需要文件副本的多个结点的 IP 地址作为响应返回对等结点。对等结点与其中一个结点连接并下载文件。当有结点加入或离开时，目录立即进行更新。

集中式网络对目录的维护做得很简单，但是这有很多缺点。访问目录可能产生巨大的流量并减慢系统。中心服务器易受攻击并且如果中心服务器全部失效，那么整个系统就会瘫痪。系统的中央组件是 Napster 在版权诉讼中失败并于 2001 年 7 月关闭的最终原因。Roxio 在 2003 年带来 New Napster；Napster 第二版是一个合法的、付费音乐站点。

分散式网络

分散式 P2P 网络并不依赖中心化目录系统。在这个模式中，对等结点将自身置于覆盖网（overlay network）中。覆盖网是一个逻辑网，在物理网的顶层创建。依照覆盖网中结点连接方式，分散式 P2P 网络分为非结构化和结构化两种。

非结构化网络

在非结构化 P2P 网络中，结点随机连接。在非结构化 P2P 中搜索效率不高，因为对一个文件的查询必须通过网络进行泛洪，这造成了极大的通信量，况且查询可能仍未解决。这类网络的两个例子是 Gnutell 和 Freenet。我们接下来把 Gnutella 作为例子来讨论。

Gnutella Gnutella 网络是分散式非结构化对等网络的一个例子。它是非结构化的，在某种意义上目录是在结点间随机分布的。当结点 A 想要访问一个对象（例如一个文件），它联系它的一个邻居。在这种情况下，邻居是结点 A 知道地址的任意一个结点。结点 A 发送查询报文给它的邻居即结点 W。这个查询包含对象的身份（例如文件名）。如果结点 W 知道结点 X 的地址，结点 X 有这个对象，那么它就发送一个包含结点 X 地址的响应报文。结点 A 现在可以使用如 HTTP 这样的传输协议中定义的命令从结点 X 得到文件的一份副本。如果结点 W 不知道结点 X 的地址，它将请求从 A 泛洪到所有邻居。最终网络中的一个结点响应了这个询问报文，并且结点 A 可以访问结点 X。我们将在第 4 章讨论路由协议时讨论泛洪，然而此处值得注意的是，尽管 Gnutella 中以某种方式防止泛洪产生巨大的通信量负载，但是 Gnutella 不能扩展的一个原因就是泛洪。

有待解决的一个问题是，根据前文表述的过程，结点 A 是否需要知道至少一个邻居的地址。这在引导（bootstrap）时期完成，这个引导时期是结点第一次安装 Gnutella 软件的时候。软件包括一个结点（对等结点）列表，结点 A 可以将它们记作邻居。之后结点 A 可以使用两种报文，称为 ping 和 pong，来检测邻居是否仍然活跃。

正如之前所述，Gnutella 网络的一个问题是由于泛洪的缺乏扩展性。当结点数目增加时，泛洪响应不够良好。为了使得查询更高效，一个新版本的 Gnutella 使用了一种由超结点（ultra node）和叶子结点（leaf）构成的分层系统。一个流入网络的结点是叶子结点，并不负责路由；有路由能力的结点被提升为超结点。这使得查询传播得更远并且提高了效率和扩展性。Gnutella 也采取了很多其他技术，如加入了查询路由协议（Query Routing Protocol，QRP）以及动态查询（Dynamic Querying，DQ）来减少通信量开销并使搜索更高效。

结构化网络

结构化网络采用一组预先确定的规则来连接结点，有效并高效地解决查询。最常用的技术是分布式散列表（Distributed Hash Table，DHT）。DHT 应用于很多应用，包括分布式数据结构（Distributed Data Structure，DDS）、内容分发系统（Content Distributed Systems，CDS）、域名系统（Domain Name System，DNS）以及 P2P 文件共享。一种流行的使用 DHT 的 P2P 文件共享协议是 BitTorrent。我们在下一节讨论 DHT，DHT 是一种在结构化 P2P 网络和其他系统中都使用的技术。

2.4.2 分布式散列表

一个分布式散列表（Distributed Hash Table，DHT）根据预先定义的规则将数据（或引用数据）分发到一组结点上。对于基于 DHT 的网络，每一个对等结点负责一系列数据项。为了避免我们在非结构化 P2P 网络中讨论的泛洪开销，基于 DHT 的网络允许每个对等结点对整个网络做部分了解。这些对网络的了解可用于把对某数据项的查询路由到负责该资源的结点上，这个路由过程使用了我们即将讨论的一系列步骤，这些步骤是有效的并可扩展的。

地址空间

在基于 DHT 的网络中，每个数据项和对等结点都被映射到大小为 2^n 的地址上。地址空间使用模运算进行设计，这意味着我们可以把地址空间的结点想象为圆环上顺时针均匀分布的 2^n 个点（0 到 2^n-1 个），如图 2-47 所示，绝大多数的 DHT 使用 $n=160$。

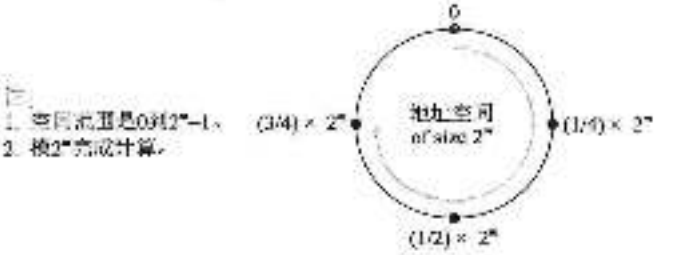


图 2-47 地址空间

散列对等结点标识符

创建 DHT 系统的第一步是将所有对等结点放入地址空间环中。这通常使用散列函数来完成，散列函数将对等结点标识符进行映射。通常是它的 IP 地址，生成一个称为结点 ID 的 n 位整数。

结点 ID = hash (对等结点 IP 地址)

散列函数是一个从输入创建输出的数学函数。然而，DHT 使用了某些加密散列函数，例如安全散列算法（Secure Hash Algorithm，SHA），这些函数是冲突避免的。这意味着两个输入被映射到同一个输出的概率是很低的。我们将在第 10 章讨论散列算法。

散列对象标识符

被共享的对象的名称（例如一个文件）也被散列成相同地址空间上的一个 n 位整数。散列结果在 DHT 术语中称为关键字（key）。

关键字 = hash (对象名)

在 DHT 中，一个对象通常和一组（key，value）相关，其中 key 是对象名的散列值，value 是对象或对象的引用。

存储对象

存储对象有两种策略：直接方法和间接方法。在直接方法中，环中的哪个结点 ID 与对象的关键字最接近（closest），就存储在那个结点上。最接近（closest）这个术语在每个协议中定义不同。这涉及了最可能传输这个对象的电脑，这个电脑最初拥有这个对象。然而由于效率缘故，绝大多数 DHT 系统使用间接方法。拥有对象的对等结点保存对象，但是对象的引用被创建并存储在另一个结点上，这个结点的 ID 最接近关键字。换言之，物理对象以及对其引用被存储在两个不同的地点。在直接策略中，我们创建了存储对象的那个结点的 ID 与对象关键字之间的关系；在间接策略中，我们创建了对象引用（指针）与存储引用的那个结点之间的关系。在这两种情况中，如果给出对象名字就需要利用这个关系才能找到对象。在本节的其余部分中，我们使用间接方法。

例 2.15 尽管 m 通常为 160，但是为了演示，我们令 $m=5$ 来使例子易于处理。在图 2-48 中，我们假设很多对等结点已经加入组。结点 N5 的 IP 地址为 110.34.56.20，它有一个名为 Liberty 的文件想要分享给它的对等结点。结点将文件名 Liberty 进行散列运算，得到 key = 14。由于最接近（closest）结点是 N17，N5 创建了对文件名（关键字）的引用、IP 地址、端口号（以及其他关于文件的信息），并发送这些引用，将其存储在结点 N17。换言之，文件存储在 N5 中，文件的关键字是 k14（DHT 环中的一个点），但是对于文件的引用存储在 N17。我们将在稍后看到其他结点如何找到 N17 并提取引用，然后使用引用来访问文件 Liberty。我们的例子仅仅给出环中的一个关键字，但在实际情况中，环中有成以百万计的关键字。

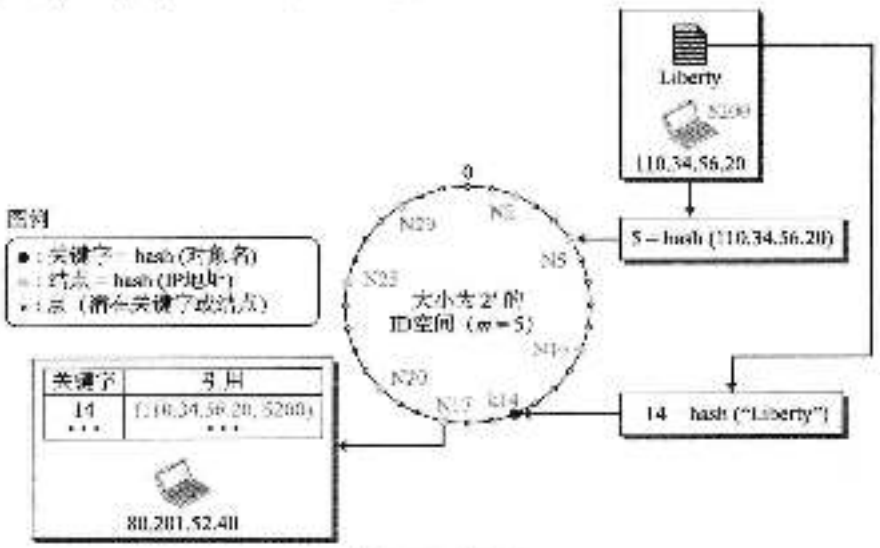


图 2-48 例 2.15

路由

DHT 的主要功能是将一个查询路由到负责存储这个对象引用的结点上。每个 DHT 的具体实现都使用不同的策略来路由，但是都依照一种思想，这种思想是每个结点必须对整个环有部分了解，从而将查询路由到与负责结点最接近的一个结点上。

结点的到达和离开

在 P2P 网络中，每个对等结点可以是一个台式机或一台笔记本电脑，对等结点可以开机或关机。当一个计算机对等结点安装了 DHT 软件，它加入了网络；当计算机关机或者对等结点关闭了软件，它就离开网络。一种 DHT 的实现需要一种清晰、有效的策略来处理结点的到达和离开，并处理对其余结点的影响。绝大多数 DHT 实现将结点失败看做结点离开。

2.4.3 Chord

有很多实现了 DHT 系统的协议。在这一节，我们介绍三种协议：Chord、Pastry 以及 Kademlia。我们选择 Chord 协议是因为它简单并且路由查询方法简约。接下来我们讨论 Pastry 协议，因为它使用与 Chord 不同的方法，并且在路由策略上与 Kademlia 协议非常接近，Kademlia 用于最流行的文件共享网络 BitTorrent。

Chord 由 Stoica 等人在 2001 年发布。我们简要介绍这种算法的主要特性。

标识符空间

Chord 中的数据项和结点是 m 位标识符，这些标识符创建了一个大小为 2^m 个点的标识符空间。这些点按顺时针分布在一个环上。我们将数据项的标识符称为 k（即 key，关键字），对等结点的标识符为 N（即 node，结点）。空间的数学运算是模 2^m 进行的。这意味着标识符号码在 0 到 2^m-1 范围内。尽管有些实现使用了免冲突散列函数，如 SHA1 中令 $m=160$，但是，我们在讨论中令 $m=5$ 使得讨论更简单。最接近 $N \geq k$ 的对等结点称作关键字 k 的后向结点并且拥有数值（k，v），其中 k 是关键字（数据项的散列值），v 是数值（关于拥有对象的对等结点服务器的信息）。换言之，诸如文件这类数据项存储在拥有数据项的对等结点上，但是数据项的散列值 key 以及对等结点的信息 value 被作为一对（k，v）存储在 k 的后向结点上。这意味着存储数据项的对等结点和拥有（k，v）对的结点不必是同一个结点。

指针表

Chord 算法中的结点应该能够解决请求：给出一个关键字，结点应该能够找到负责这个关键字的结点标识符，或者将查询转发给另一个结点。Chord 要求每一个结点维护 m 的后向结点以及一个前向结点的信息。每个结点创建一个称为指针表（finger table）的路由表，如图 2-14 所示。注意到第 i 行的目标关键字是 $N+2^{i-1}$。

表 2-14 指针表

i	目标关键字	目标关键字的后向结点	后向结点的信息
1	$N+1$	$N+1$ 的后向结点	后向结点的 IP 地址和端口
2	$N+2$	$N+2$ 的后向结点	后向结点的 IP 地址和端口
⋮	⋮	⋮	⋮
m	$N+2^{m-1}$	$N+2^{m-1}$ 的后向结点	后向结点的 IP 地址和端口

图 2-49 仅给出了环中的一个后向结点列，这个环仅有少量结点和关键字。请注意，实际上第一行（$i=1$）给出了后向结点。我们也添加了前向结点的 ID，稍后我们会看到这是需要的。

接口

为了进行操作，Chord 需要一组 Chord 接口。在这一节，我们讨论部分操作来给出 Chord 协议背后的思想。

查找

Chord 中最常用的操作可能就是查找。Chord 让对等结点之间共享可用服务。为了找到被共享的对象，对等结点需要知道负责那个对象的结点；存储对象引用的对等结点。在 Chord 中，我们讨论过，环中一组关键字的后继结点就是负责那些关键字的结点。找到负责结点实际上就是找到关键字的后继结点。表 2-15 给出查找操作的代码。

查找函数使用自顶向下方法编写。如果结点负责关键字，它返回自己的 ID；否则，它调用函数 find_successor。find_successor 函数调用 find_predecessor。最终的函数调用 find_closest_predecessor。模块方法允许我们在其他操作中使用这三个函数而不必重定义它们。

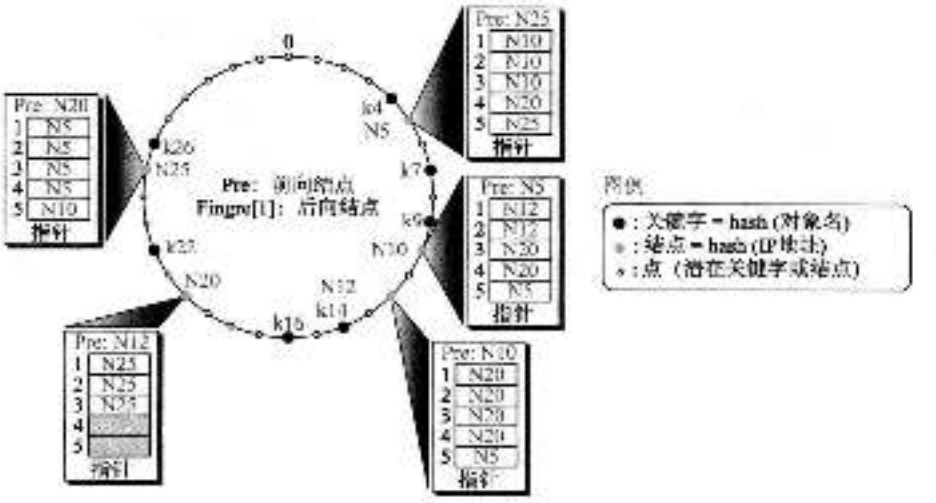


图 2-49 Chord 中环的例子

表 2-15 查找

```
Lookup (key)
{
    if (node is responsible for the key)
        return (node's ID)
    else
        return find_successor (key)
}

find_successor (id)
{
    x = find_ predecessor (id)
    return x.finger[1]
}

find_predecessor (id)
{
    x = N                                          // N 是当前结点
    while (id ∉ (x, x.finger[1]]
    {
        x = x.find_closest_predecessor (id)        // 令 x 向前找
    }
    return x
}

find_closest_predecessor (id)
{
    for (i = m downto 1)
    {
        if (finger [i] ∈ (N, id))                  // N 是当前结点
            return (finger [i])
    }
    return N                                       // 结点 N 是最接近的前驱结点
}
```

让我们来仔细研究 lookup 函数。如果结点不负责关键字 key，lookup 函数调用 find successor 函数来找到这个 ID 的后向结点。这个 ID 是作为参数传递给 find_successor 函数的。如果我们首先找到关键字 key 的后向结点，那么后向结点函数代码可以非常简单。前向结点可以轻易地帮助我们找到环中的下一个结点，因为前向结点的第一个指针（finger[1]）给出了后向结点的 ID。此外，查找关键字前向结点的函数在其他函数中是有用的，这些函数我们稍后会编写。不幸的是，结点自己不能正常地找到前向结点；关键字 key 可能远离结点。正如我们之前讨论的，一个结点仅维护有限个其余结点的信息；指针表最多维护 m 个其他结点（在指针表中有一些重复数据）。由于这个原因，一个结点需要其他结点的帮助来找到关键字 key 的前向结点。这一步可以将 find closest predecessor 函数作为远程程序调用（remote procedure call，RPC）来完成。远程程序调用意味着调用一个在远程结点执行的函数，并且将结果返回到调用结点上。我们在算法中使用表达式 x.procedure，其中 x 是远程结点的标识符，procedure 表示被执行的程序。结点使用这个函数来找到一个更接近前向结点的点。然后，它将寻找前向结点的责任传递给其他结点。换言之，如果结点 A 想要找到结点 X，它找结点 B（最接近前向结点）并且把任务发送给 B。现在结点 B 接管控制并且试图寻找 X，或者 B 将任务发送给另一个结点 C。任务被从一个结点转发到另一个结点，直到一个结点为止，即那个结点拥有所要找的前向结点的信息。

例 2.16 假设图 2-49 的结点 N5 需要寻找负责关键字 k14 的结点。图 2-50 给出了 8 个事件的序列。在事件 4 中，find_closest_predecessor 函数返回了 N10。在事件 4 之后，find predecessor 函数要求 N10 返回它的 finguer[1]，也就是 N12。此时，N5 发现 N10 并不是 k14 的前向结点。之后，结点 N5 要求 N10 找到最接近 k14 的前向结点，这个请求返回了 N12（事件 5 和 6）。现在，结点 N5 请求结点 N12 的 finger[1]，返回结果为 N20。现在结点 N5 进行检查，发现 N12 确实是 k14 的前向结点。这个信息就传送给 find_successor 函数（事件 7）。N5 现在请求结点 N12 的 finguer[1]，返回结果为 N20。搜索终止，N20 是 k14 的后向结点。

稳定化

在我们讨论结点如何加入和离开环之前，我们需要强调的是环中的任何改变（例如，结点、结点组的加入和到达）可能导致环不稳定。Chord 中定义的一个操作称为稳定化，环中的每个点周期性地使用这个操作来验证它的后向结点，并且令后向结点验证它的前向结点信息。结点 N 使用 finguer[1]的值 S 来要求结点 S 返回它的前向结点 P。如果从这个请求中得到的返回值 P 在 N 和 S 之间，这意味着存在一个 ID 等于 P 的结点，它位于 N 与 S 之间。结点 N 使 P 成为它的后向结点，并通知 P 使 N 成为其前向结点。表 2-16 给出了稳定化操作。

表 2-16 稳定化

```
Stabilize ()
{
    P = finger[1].Pre                    // 要求后向结点返回它的前向结点
    If(P∈ (N, finger[1]))  finger[1] = P  // P 是 N 的后向结点
    finger[1].notify (N)                 // 通知 P 改变它的前向结点
}
Notify (x)
{
    if (Pre = null or x ∈ (Pre, N))   Pre = x
}
```

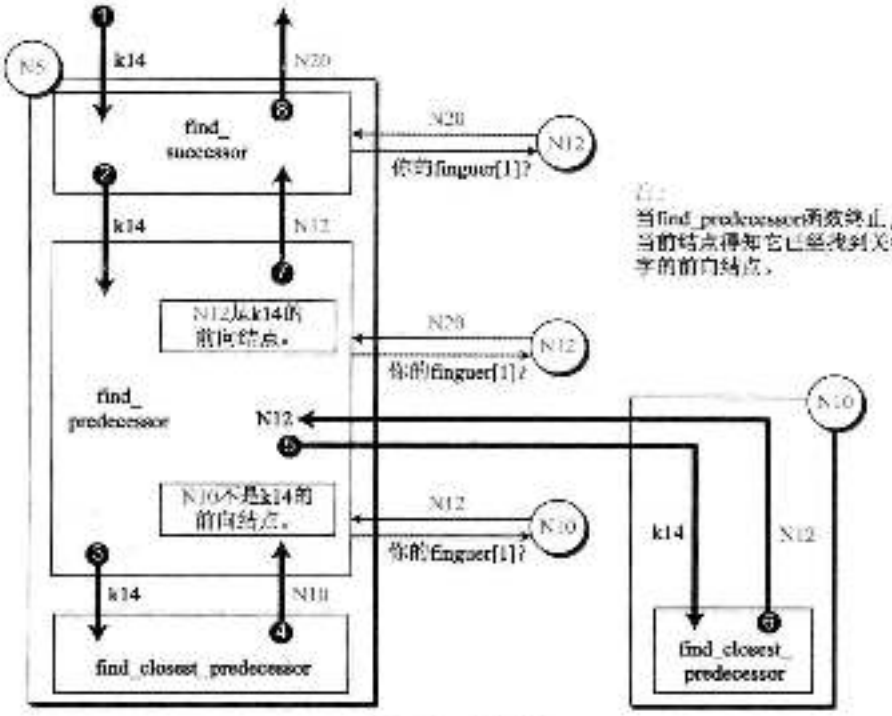


图 2-50 例 2.16

Fix_Finger

不稳定可能最多改变 m 个点的指针表。Chord 中定义的另一个操作是 fix_finger。环中的每个结点必须周期性地调用这个函数来进行指针表更新。为了减少系统的通信量，每个结点必须在每次调用中只更新它的一个指针。这个指针是随机选择的。表 2-17 给出了这个操作的代码。

表 2-17 Fix_Finger

```
Fix_Finger ()
{
    Generate (i ∈ (1, m])                    // 随机产生 i，1<i≤m
    finger[i] = find_successor (N + 2^(i-1))  // 找到 finger[i]关键
}
```

加入

当一个对等结点加入环，它使用 join 操作以及其他对等结点的 ID 来找到它的后向结点，并且将其后向结点置为空。它立即调用稳定化函数来验证它的后向结点。之后，结点要求后向结点调用 move-key 函数来传输新对等结点负责的关键字。表 2-18 给出这个操作的代码。

表 2-18 Join

```
Join (x)
{
    Initialize (x)
    finger[1].Move_Keys (N)
}
```

```
Initialize (x)
{
    Pre = null
    If (x = null) finger[1] = N
    else finger[1] = x.Find_Successor (N)
}

Move_Keys (x)
{
    for (each key k)
    {
        If (x ∈ [k, N)) move (k to node x)                    // N是当前结点
    }
}
```

很明显，在这个操作后加入的结点的指针表是空的，并且最多有 m 个前向结点的指针表是过期的。在这个事件之后，同期生地运行 stablize 以及 fix_finger 操作将逐渐稳定系统。

例 2.17 我们假设图 2-49 中，结点 N17 在 N5 的帮助下加入环。图 2-51 给出了稳定后的环。过程如下：

1. N17 使用 Initialize（5）算法设置其前向结点为空，并设置后向结点（finger[1]指向 N20）。
2. 之后 N17 要求 N20 发送 k14 以及 k16 到 N17，因为 N17 现在负责这些关键字。
3. 在下一次起时，N17 使用 stabilize 操作来验证自己的后向结点（即 N20）并且要求 N20 将其前向结点改为 N17（使用 notify 函数）。
4. 当 N12 使用 stabilize 时，N17 的前向结点更新为 N12。
5. 最终，当某些结点调用 fix-finger 函数时，N17、N10、N5 以及 N12 的指针表被改变。

离开或失效

如果一个对等结点离开环或者失效（不是环失效），环将会中断运行，除非环能稳定化其自身。每个结点与邻居交换 ping 和 pong 报文来侦测邻居是否还活跃。当结点没有收到回应 ping 的 pong 报文时，结点知道邻居失效了。

尽管使用 stabilize 和 fix-finger 操作可以在结点离开或失效后恢复环，但是发现问题的结点可以立即采取这些操作而不用等待超时时间。一个重要问题是当多个结点同时离开或失效时，stabilize 和 fix-finger 操作可能会不起作用。因此，Chord 要求每个结点记录 r 个后续结点（r 的值取决于具体实现）。如果一个后向结点不可用，那么就可以去往下一个后向结点。

这种情况下的另一个问题是由失效或离开的结点管理的那些数据已变得不可用了。Chord 规定只有一个结点负责一组数据和引用，但是 Chord 也规定，在这种情况下，数据和引用应该在别的结点备份。

例 2.18 在图 2-51 中，我们假设结点 N10 离开环。图 2-52 给出了稳定后的环图示。

过程如下：

1. 当结点 N5 收不到 N10 回复的 pong 报文时，N5 发现 N10 已经离开。结点 N5 改变它的后继结点（finger[1]）到 N12（后继结点列表中的第二项）。
2. 结点 N5 立即调用 stabilize 函数并要求 N12 将其前向结点改为 N5。
3. 如果顺利的话，N10 负责的 k7 和 k9 在 N10 离开前已经被复制到 N12。
4. 在若干次 fix-finger 调用之后，如图 2-52 所示，N5 和 N25 更新了它们的指针表。

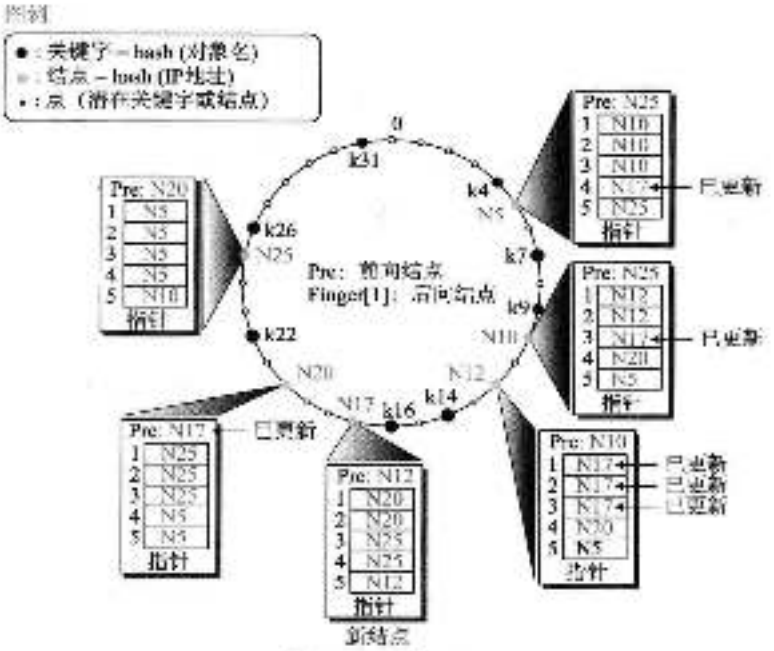


图 2-51 例 2.17

应用

Chord 有很多应用，包括协同文件系统（Collaborative File System，CFS）、ConChord 以及分布式域名系统（Distributive Domain Name System，DDNS）。

2.4.4 Pastry

另一个 P2P 模式中的流行协议是 **Pastry**，它是由 Rowstron 和 Druschel 设计的。如上所述，Pastry 使用 DHT，但是 Pastry 与 Chord 之间在标识符空间和路由过程中有一些根本性的不同，这些不同，我们将在下文介绍。

标识符空间

与 Chord 类似，在 Pastry 中的结点和数据项是一个 m 位的标识符，这些标识符创建了一个由 2^m 个顺时针均匀分布在环上的点组成的标识符空间。m 通常为 128。协议使用 SHA-1 散列算法，其中 $m = 128$。然而，在这个协议中，标识符被视为基于 2^b 的 n 位字符串，其中 b 通常为 4，并且 $n = (m/b)$。换言之，标识符是一个基于 16（十六进制）的 32 位数字。在标识符空间中，关键字存储在结点标识符与其最接近的那个结点上。这个策略与 Chord 不同。在 Chord 中，关键字存储在它的后向结点上；在 Pastry 中，关键字可能存储在数值上最接近关键字的后向或前向结点中。

路由

Pastry 结点应该能够解决查询：给定一个关键字，结点应该能够找到负责那个关键字的结点或者将查询转发到另外一个结点。Pastry 中每个结点使用两个实体来进行路由：路由表和叶子结点集。

路由表

Pastry 要求每个结点维护一个 n 行 2^b 列的路由表。通常，当 $m = 128$ 并且 $b=4$ 时我们有 32（128/4）行 16 列（$2^{128} = 16^{32}$）。换言之，每一行对应标识符的一位，每一列对应十六进制的值（0 到 F）。表 2-19 给出普通情况下的路由表大纲。在结点 N 的路由表中，i 行 j 列的单元格，表[i, j]，给出了结点的标识符（如果存在的话），这个结点最左边 i 位与 N 的标识符相同，第（i + 1）位的值为 j。

第 1 行即 0 行给出了活动结点列表，这些活动结点的标识符与 N 没有相同前缀。第 1 列给出另一个活动结点列表，这些活动结点与结点 N 最左边的 1 位数字是相同的。类似地，第 31 列给出了所有活动结点列表，这些活动结点与结点 N 的左边 31 位数字是相同的；只有最后一位不相同。

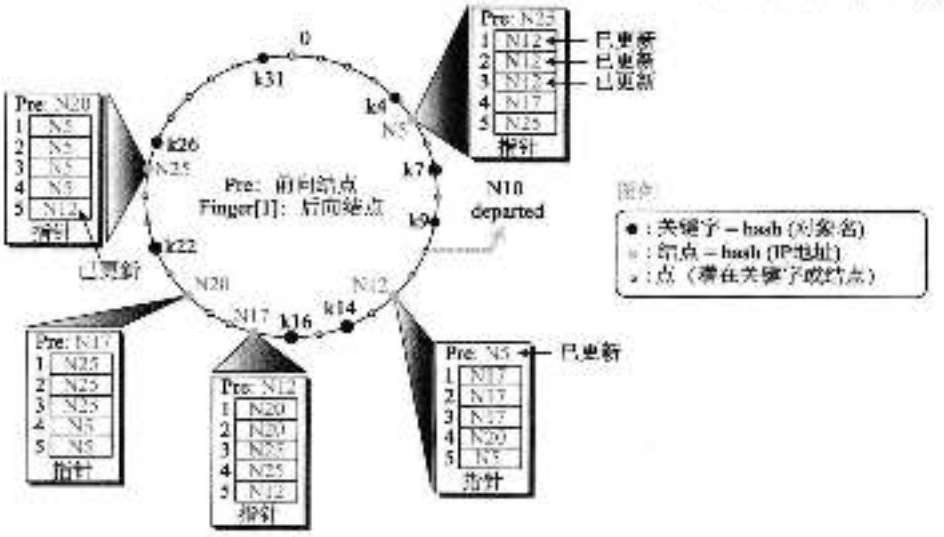


图 2-52 例 2.18

例如，如果 N = (574A234B12E374A2001B23451EEE4BCD)$_{16}$，那么表[2，D]的值可能是结点（57D...）的标识符。注意到最左侧两个数字为 57，与 N 的前两位数字是相同的，但是下一位数字是 D，这个数字与第 D 列相对应。如果有更多的结点带有前缀 57D，根据接近度（proximity metric），最接近的结点被选中并将其标识符插入表格内。接近度是一种由使用网络的具体应用决定的度量。它可能基于结点间的跳数、往返时间或其他度量。

表 2-19 Pastry 中结点的路由表

公共前缀长度	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
31																

叶子结点集

路由中的另一个实体是由 2^b（路由表中列的大小）个标识符组成的集合，它称为叶子结点集（leaf set）。集合中一半的结点标识符在数值上小于当前结点；集合中另一半的结点标识符在数值上大于当前结点。换言之，叶子结点集给出了环上位于当前结点之前的 2^{b-1} 个结点以及位于之后的 2^{b-1} 个结点。

例 2.19 让我们假设 $m = 8$ 位，并且 $b = 2$。这意味着我们有上限为 $2^m = 256$ 个标识符，并且每个标识符基于 $2^b = 4$，共有 $m/b = 4$ 位数字。图 2-53 给出一种情形，其中一些活动结点和关键字映射到这些结点上。关键字 k1213 存储在两个结点上，因为它距离这两个结点等距。这提供了一些冗余，以防其中一个结点失效。图 2-53 也给出了四个结点的路由表和叶子结点集，这些结点在后文例子中会用到。例如，在结点 N0302 的路由表中，根据接近度，我们假设结点 1302 与结点 N0302 最近，所以选定结点 1302 插入表[0，1]。我们对于其他的表项也采取相同的策略。请注意，每个表的每一行中有一个单元格是带阴影的，这是因为它与结点标识符的数字相对应；任何其他结点标

识符都不能插入到这个单元格。有些单元格是空的，这是因为此时网络中没有满足要求的活动结点；当有新结点加入网络，它们就可以被插入这些单元格。

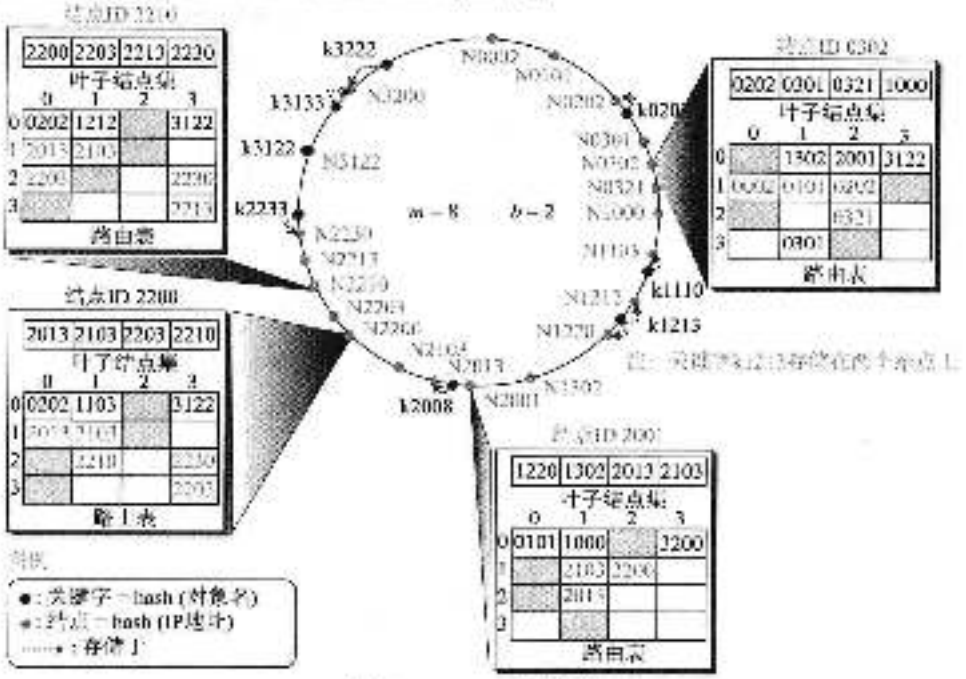


图 2-53 Pastry 环举例

查找

正如我们在 Chord 中讨论的，Pastry 中使用的一种操作是查找：给定一个关键字，我们需要找到存储着关键字相关信息或关键字自身的结点。表 2-20 以伪代码形式给出了查找操作。N 是本地结点的标识符。它是接收报文的结点同时也需要找到存储着报文中关键字的结点。

表 2-20 查找

```
Lookup (key)
{
    if (key is in the range of N's leaf set)
        forward the message to the closest node in the leaf set
    else
        route (key, Table)
}
route (key, Table)
{
    p = length of shared prefix between key and N
    v = value of the digit at position p of the key          // 从 0 开始的位置
    if (Table [p, v] exists)
        forward the message to the node in Table [p, v]
    else
        forward the message to a node sharing a prefix as long as the current node, but
        numerically closer to the key.
}
```

例 2.20 在图 2-53 中，我们假设结点 N2210 接收到一个查询，它寻找负责 key2008 的结点。因为当前结点不负责这个关键字，它首先检查自己的叶子结点集。key2008 不在叶子结点范围内，因此结点需要使用路由表。因为公共前缀的长度是 1，即 p－1。关键字中第 1 位数字的值为 v－0。结点在表[1，0]中检查标识符，得到结果 2013。查询被转发给结点 2013，它实际上负责这个关键字。结点将它的信息发送给请求结点。

例 2.21 在图 2-53 中，我们假设结点 N0302 接收到一个查询，它寻找负责 key0203 的结点。当前结点不负责这个关键字，但是这个关键字在它的叶子结点集范围内。这个集合中与 key 最近的结点是 N0202。查询被发送到这个结点，实际上就是负责这个关键字的结点。结点 N0202 将它的信息发送给请求结点。

加入

加入 Pastry 中的环要比 Chord 中更简单更快速。一个新的结点 X 应该知道至少一个结点 N0，这个结点应该与 X 靠近（基于接近度）；这可以通过运行一个名为附近结点发现的算法来完成。结点 X 发送一个加入报文给结点 N0。在我们的讨论当中，我们假设 N0 的标识符与 X 的标识符没有公共前缀。以下步骤给出了结点 X 如何构造路由表及叶子结点集：

1. 结点 N0 发送第 0 行的内容给结点 X。因为两个结点没有公共前缀，结点 X 使用这个信息的适当部分来构造自己的第 0 行。之后，结点 N0 将加入报文作为查找报文进行处理，假设 X 的标识符为关键字。它向结点 N1 转发加入报文，N1 的标识符更接近 X。

2. 结点 N1 发送第 1 行的内容给结点 X。因为这两个结点有一个相同前缀，结点 X 使用这个信息的适当部分来构造自己的第 1 行。之后，结点 N1 将加入报文作为查找报文进行处理，假设 X 的标识符为关键字。它向结点 N2 转发加入报文，N2 的标识符更接近 X。

3. 这个过程继续，直到 X 的路由表完成。

4. 过程中的最后一个结点，与 X 有最长的公共前缀，它向结点 X 发送叶子结点集，这个集合变成了结点 X 的叶子结点集。

5. 然后结点 X 与它路由表中的结点交换信息和叶子结点集，从而来改善自己的路由信息并允许那些结点更新自身信息。

例 2.22 图 2-54 给出了标识符为 N2212 的新结点 X 是如何加入环的，结点 X 使用图 2-53 中四个结点的信息来创建初始路由表和叶子结点集。请注意，这两个表的内容在更新过程中将更接近它们的理想状态。在这个例子中，我们假设基于接近度，结点 0302 是结点 2212 的接近的结点。

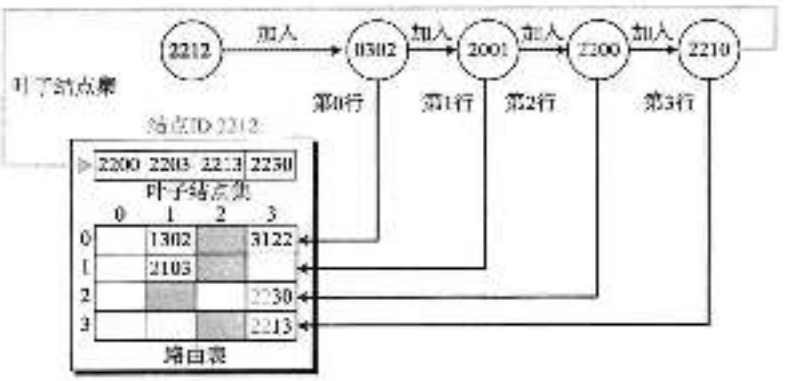


图 2-54 例 2.22

离开或失效

每个 Pastry 结点都将周期性地发送探测报文，来检测叶子结点集和路由表中的结点存活状况。

如果一个本地结点发现叶子结点集中的某个结点对探测报文无响应，它就假设这个结点已经失效或离开。为了替代这个结点，本地结点与叶子结点集中具有最大结点标识符的活动结点联系，并且利用那个结点的叶子结点集信息修复自己叶子结点集信息。由于最近结点的叶子结点集有覆盖，因此这个过程是成功的。

如果本地结点发现路由表中[i，j]结点对探测报文没有响应，它就发送报文给同一行的活动结点并请求表[i，j]的标识符。这个标识符代替了失效或离开的结点。

应用

Pastry 被用于一些应用，比如 PAST，这是一个分布式文件系统，以及 SCRIBE，这是一个分散式发布/订阅系统。

2.4.5 Kademlia

另一种 DHT 对等网络是 **Kademlia**，它由 Maymounkov 和 Mazières 设计。与 Pastry 类似，Kademlia 基于结点距离来路由报文，但是正如下文所述，Kademlia 中的距离度量与 Pastry 中不同。在这个网络中，两个标识符（结点或关键字）之间的距离是通过位异或（XOR）来度量的。换言之，如果 x 和 y 是两个标识符，我们有

$$\text{distance}(x, y) = x \oplus y$$

当我们度量两点间的几何距离时，XOR 有以下四个特性：

$x \oplus x = 0$	点与自身的距离是 0。
$x \oplus y > 0 \text{ if } x \neq y$	两点间距离大于 0。
$x \oplus y = y \oplus x$	x 到 y 的距离与 y 到 x 的距离相等。
$x \oplus z \leq x \oplus y + y \oplus z$	满足三角关系。

标识符空间

在 Kademlia 中，结点和数据项是 m 位标识符，它们创建了含有 2^m 个点的标识符空间。这些点分布在二叉树叶子结点上。此协议使用 SHA-1 散列算法，其中 $m = 160$。

例 2.23 为方便起见，我们假设 $m = 4$。在这个空间中有 16 个分布在二叉树叶子结点上的标识符。图 2-55 给出的情况只有 8 个活动结点以及 5 个关键字。

如图 2-55 所示，由于 $3 \oplus 3 = 0$，因此关键字 k3 存储在结点 N3 上。尽管关键字 k7 看起来与 N6 和 N8 在数字上等距，但是，它却存储在 N6 上，因为 $6 \oplus 7 = 1$ 而 $6 \oplus 8 = 14$。另一个有趣的点是关键字 k12，它与 N11 在数字上更接近，但它却存储在 N15 上，因为 $11 \oplus 12 = 7$ 但是 $15 \oplus 12 = 3$。

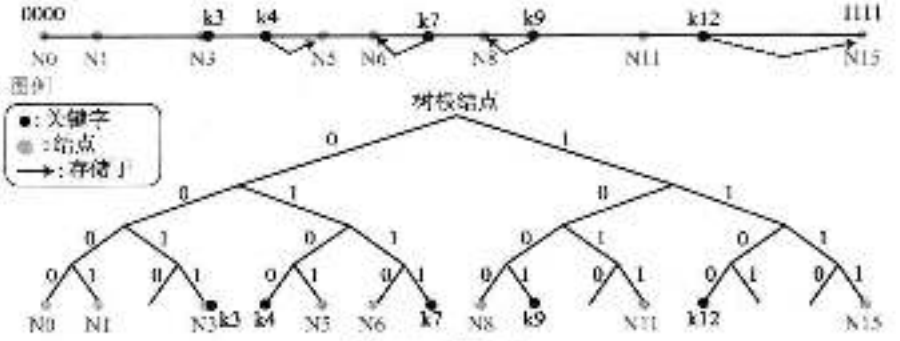


图 2-55 例 2.23

路由表

Kademlia 仅为每一个结点维护一个路由表；且没有叶子结点集。网络中每个结点将二叉树分成 m 棵子树，这些子树并不包含结点自身。子树 i 包含了那些与相应结点共享最左 i 位数字（公共前缀）

的结点。路由表有 m 行但只有 1 列。在我们的讨论中，我们假设每一行存储了相应子树中的一个结点的标识符，但是之后我们展示出 Kademlia 允许每行有多达 k 个结点。这个思想和 Pastry 相同，但是公共前缀的长度是基于比特位的个数，而不是基于 2^b 的数字个数。表 2-21 给出了路由表。

例 2.24 让我们来找例 2.23 中的路由表。为简便起见，我们假设每行仅使用一个标识符。因为 $m=4$，每个结点有四个子树，这些子树对应路由表中的四行。每行的标识符代表在相应子树中与当前结点距离最近的点。图 2-56 给出所有的路由表，但是只有三棵子树。为了使图例更小，我们选择出 8 棵树。

我们以结点 6 举例，使用相应的子树来解释一下如何构造路由表。对于其他结点的解释是类似的。

表 2-21 Kademlia 中的一个结点的路由表

公共前缀长度	标 识 符
0	子树中公共前缀长度为 0 的最接近结点
1	子树中公共前缀长度为 1 的最接近结点
⋮	⋮
$m-1$	子树中公共前缀长度为 $m-1$ 的最接近结点

a. 在第 0 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p=0$ 且距离最近。在这棵子树中有三个结点（N8、N11 以及 N15）。然而，N15 与 N6 最近，因为 N6 ⊕ N8 = 14，N6 ⊕ N11 = 13 并且 N6 ⊕ N15 = 9。N15 就被插入到第 0 行。

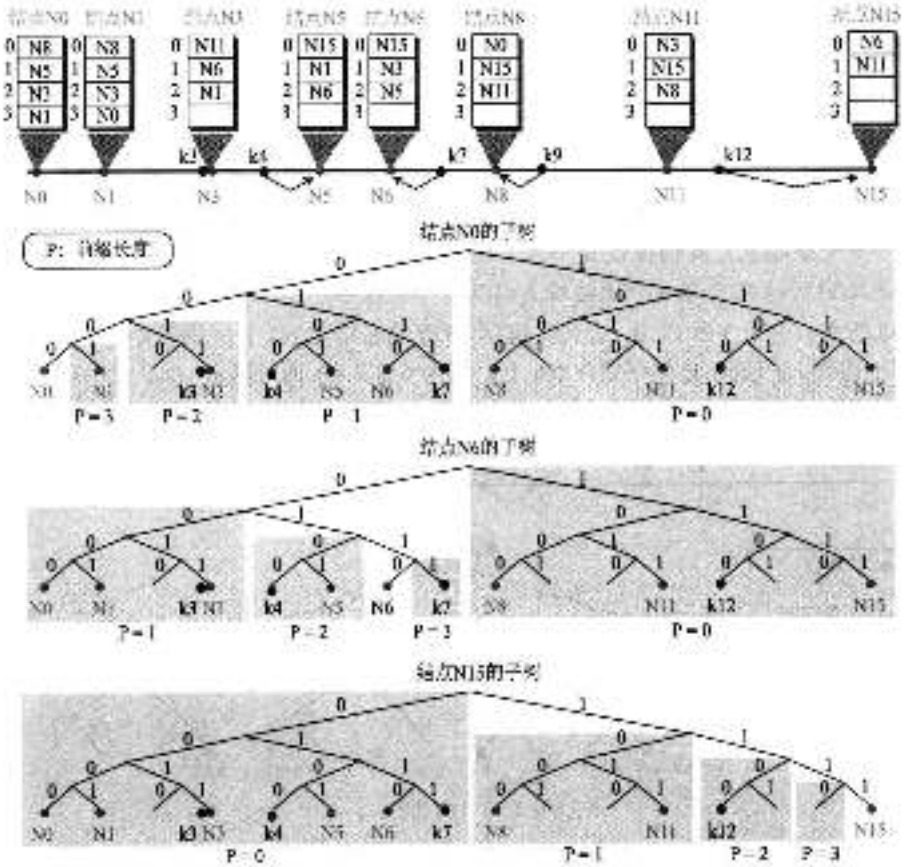


图 2-56 例 2.24

b. 在第 1 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p=1$ 且距离

最近。在这棵子树中有三个结点（N0、N1 以及 N3），然而，N3 与 N6 最近，因为 N6 ⊕ N0 = 6，N6 ⊕ N1 = 7 并且 N6 ⊕ N3 = 5。N15 就被插入到第 1 行。

c. 在第 2 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 2$ 且距离最近。在这棵子树中只有一个结点（N5），它就被插入到那里。

d. 在第 3 行，我们需要插入一个结点标识符，这个结点在子树中公共前缀长度 $p = 3$ 且距离最近。在这棵子树中没有结点，到此该行为空。

例 2.25 在图 2-56 中，我们假设结点 N0（0000）$_2$ 接收到一个查找报文，报文要找负责 k12（1100）$_2$ 的结点。这两个标识符的公共前缀长度为 0。结点 N0 发送报文给路由表第 0 行的结点 N8。现在 N8（1000）$_2$ 需要查找与 k12（1100）$_2$ 最近的结点。这两个标识符的公共前缀长度为 1。结点 N8 将这个报文发送给路由表第 1 行的结点 N15，N15 负责这个关键字 k12。路由过程结束。这个路由是 N0→N8→N15。有趣的是结点 N15（111）$_2$ 以及 k12（1100）$_2$ 公共前缀长度为 2，但是 N15 的第 2 行是空的，这意味着 N15 它自身负责 k12。

例 2.26 在图 2-56 中，我们假设结点 N5（0101）$_2$ 接收到一个查找报文，报文要找负责 k7（0111）$_2$ 的结点。这两个标识符的公共前缀长度为 2。结点 N5 发送报文给路由表第 2 行的结点 N6，这个结点负责 k7。路由过程结束。这个路由是 N5→N6。

例 2.27 在图 2-56 中，我们假设结点 N11（101）$_2$ 接收到一个查找报文，报文要找负责 k4（0100）$_2$ 的结点。这两个标识符的公共前缀长度为 0。结点 N11 发送报文给路由表第 0 行的结点 N3。现在 N3（0011）$_2$ 需要查找与 k4（0100）$_2$ 最近的结点。这两个标识符的公共前缀长度为 1。结点 N3 将这个报文发送给路由表第 1 行的结点 N6。现在结点 N6（0110）$_2$ 需要查找与 k4（0100）$_2$ 最近的结点。这两个标识符的公共前缀长度是 2。结点 N6 发送报文给路由表 2 行的结点 N5，这个结点负责关键字 k12。路由过程结束。这个路由是 N11→N3→N6→N5。

k-桶

在之前的讨论中，我们假设路由表中的一行仅仅列出相应子树中的一个结点。为了提高效率，Kademlia 要求每一行至少维护 k 个来自相应子树的结点。k 的具体数值依赖于系统，但是实际网络中推荐使用 20 左右的数值。因此，路由表中每一行称为一个 k-桶（k-bucket）。在每一行中包含一个以上结点可以允许客户在结点离开网络或失效时使用替换结点。Kademlia 将那些在网络中连接很长时间的结点存储在桶中。已经证明的是，保持连接时间越长的结点越有可能在更长时间内保持连接。

并行查询

由于在 k-桶中有多个结点，Kademlia 允许向 k-桶顶部 α 个结点发送 α 个并行查询。如果一个结点失效或无法回应查询，并行查询将减少延迟。

并发更新

Kademlia 中另一个有趣的特性是并发更新。无论何时，当一个结点收到查询或响应，它都立即更新 k-桶。如果发向一个结点的多个查询没有收到响应，发送查询的结点将从相应的 k-桶中去除这个目的结点。

加入

如 Pastry 中，一个要加入网络的结点需要知道至少一个结点。加入的结点向网络中的结点发送自身标识符，好像这个标识符是要查找的关键字。它接收到的响应允许新结点创建自己的 k-桶。

离开或失效

当一个结点离开网络或失效时，其他结点使用如上所述的并发过程来更新它们的 k-桶。

2.4.6 一种流行的 P2P 网络：BitTorrent

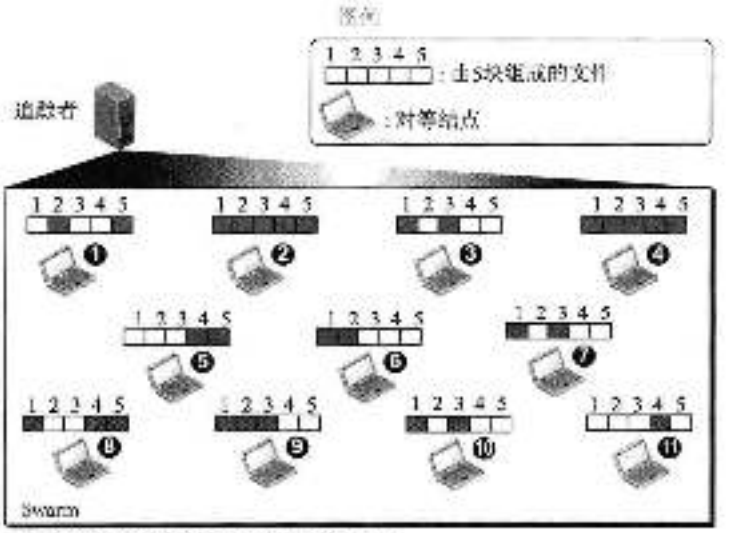
BitTorrent 是 Bram Cohen 设计的一个 P2P 协议，其目的是在一组对等结点间共享一个大文件。

然而，在本文中，共享（sharing）这个术语与其他文件共享协议中是不同的。此处，是一组对等结点参加到为组内所有对等结点提供文件拷贝的过程中，而不是一个对等结点允许另一个对等结点下载整个文件。文件共享是在合作过程中完成的，这称为 torrent。参加到 torrent 中的每个对等结点从另一个有文件的对等结点那里下载人文件的块，同时，它也为其他没有这个文件的结点上传文件块，它有点像孩子们玩的交易游戏以牙还牙（tit-for-tat）。参加 torrent 的所有对等结点的集合称为一个群（swarm）。群中拥有完整文件内容的对等结点称为种子（seed）；只有部分文件并想下载其余部分的对等结点称为寄生虫（leech）。换言之，一个群是种子和寄生虫的组合。BitTorrent 已有多个版本和实现。我们首先描述原始版本，它使用称为追踪者（tracker）的中心结点。之后，我们给出一些新的版本是如何利用 DHT 消除追踪者的。

带有追踪者的 BitTorrent

在原始 BitTorrent 中，在 torrent 中有另一个实体，叫做追踪者。正如其名所示，它追踪群的操作。后文会对其进行描述。图 2-57 给出了一个带有种子、寄生虫和追踪者的 torrent。

在图 2-57 中，文件要被共享。文件内容被分成五个片段（块）。对等结点 2 和 4 已经拥有所有文件片段；另外的对等结点有一些片段。每个对等结点拥有的文件片段标注为阴影。上传和下载文件片段将会继续。某些对等结点可能会离开 torrent；某些可能会加入。



注：对等结点2和4是种子；其他是寄生虫。

图 2-57 torrent 举例

现在假设一个新对等结点想要下载相同的文件内容。新对等结点访问 BitTorrent 服务器查找文件内容的名字。它接收到一个元文件，即 torrent 文件，这个文件包含了文件内容片段的信息以及处理这个 torrent 的追踪者的地址。现在新对等结点访问追踪者并取得 torrent 中一些对等结点的地址，通常称为邻居（neighbor）。现在新对等结点是 torrent 的一部分并且可以下载和上传文件内容的片段。当它拥有所有片段时，它可能会离开 torrent 或留在 torrent 中帮助其他对等结点得到文件内容的所有片段。这些被帮助的对等结点包括在其后加入的新结点。没有什么可以防止一个对等结点在得到所有片段之前就离开 torrent，而且在之后这个结点可以再加入或干脆不再加入 torrent。

尽管加入、共享、离开 torrent 的过程看似简单，BitTorrent 协议应用了一组策略来提供公平性，鼓励对等结点交换片段，防止某个对等结点接收请求过载并且允许对等结点寻找提供更好服务的结点。

为了避免过载并实现公平性，每个结点需要限制它与邻居们的并发连接；通常的数值为4。一个对等结点将其邻居标记为非阻塞或阻塞。它也将它们标记为感兴趣或不感兴趣。

换言之，一个对等结点将他的邻居列表分为两个不同的组：非阻塞（unchoked）和阻塞（choked）。它也将其分为感兴趣（interested）和不感兴趣（uninterested）组。非阻塞组是那些被并发连接到的当前结点组；它从组中持续上传并下载片段。阻塞组是那些当前没有被连接到的结点组，但是将来可能连接。

每隔10秒，当前对等结点尝试连接组中感兴趣但是处于阻塞状态的对等结点，以获得更高的数据速率。如果这个新的对等结点比其他非阻塞对等结点拥有更高的速率，那么这个新对等结点可能变为非阻塞状态，并且非阻塞组中最低数据速率的结点可能移入阻塞组。采取这种方法，非阻塞组中的对等结点总是拥有被探测结点中最高的数据速率。使用这种策略，将邻居分成子组，其中那些有一致数据传输速率的邻居将相互通信。在这种策略中可以看到上文所述的以牙还牙交易策略思想。

为了允许一个尚无片段共享的新加入对等结点也能从其他结点接收片段，每隔30秒，一个对等结点随机地将一个感兴趣结点从阻塞组中选出来，标记为非阻塞，而不管其上传速率是多少。这个动作称为乐观非阻塞（optimistic unchocking）。

BitTorrent协议采用一种称为最少优先（rarest-first）的策略，试图在每个对等结点每一刻拥有的不同片段数之间提供一种平衡。使用这种策略，对等结点首先试图下载邻居中重复得最少的片段。采用这种方法，这些稀少片段会传播得更快。

无追踪者的BitTorrent

在BitTorrent原始的设计中，如果追踪者失效，新对等结点不能连接到网络且更新也中断。有几种BitTorrent实现消除了对中心化追踪者的需要。在此处描述的实现中，协议仍然使用追踪者，但不是一个中心追踪者。追踪的任务分布在网络中的一些结点上。在这一节，我们给出如何用Kademlia DHT实现这个目标，但是我们避免涉及特定的协议细节。

在带有中心追踪者的BitTorrent中，追踪者的任务是当给出原数据文件时，提供对等结点列表。如果我们将元数据的散列函数看做关键字，把群中对等结点列表的散列函数看做数值，我们可以使P2P网络中的某些结点起到追踪者的作用。一个加入torrent的新对等结点将元数据（关键字）的散列函数发送到它所知道的结点。P2P网络使用Kademlia协议来寻找负责关键字的结点。负责结点向加入结点发送数值，这个数值实际上是对应torrent中的对等结点列表。现在加入结点可以使用BitTorrent协议与列表中的对等结点共享内容文件。

2.5 套接字接口编程

在2.2节，我们讨论了客户-服务器模式的原则。在2.3节，我们讨论了使用这种模式的一些标准应用。在这一节，我们给出如何使用过程编程语言C语言来编写简单的客户-服务器程序。我们选择C语言的原因有两个。第一，传统上说套接字编程就是从C语言开始的。第二，C语言的底层特性将更好地揭示这类编程的精妙之处。在第11章，我们用Java扩展这种思想，它提供了一个更加简洁的版本。然而，即使跳过这一节也不会丧失在本书学习中的连续性。

C的套接字接口

我们在2.2节讨论过套接字接口。在本节，我们给出这个接口用C语言是如何实现的。套接字接口的关键问题是理解套接字在通信中的角色。套接字没有存储待发送或待接收数据的缓冲区。它既不能发送也不能接收数据。套接字只起到一个引用或标签的作用。缓冲区和必要的变量在操作系统中创建。

套接字的数据结构

C 语言将套接字定义为一个结构（struct）。套接字结构由五个字段组成：每个套接字地址是一个由五部分构成的结构，如图 2-58 所示。请注意，程序员不该重定义这个结构；它已经在头文件中定义好了。我们能要讨论套接字结构中的五个字段。

- 族。这个字段定义了协议簇（如何解释地址和端口号）。通常值是 PF_INET（用于当前因特网）、PF_INET6（用于下一代因特网）等等。我们在本书使用 PF_INET。
- 类型。这个字段定义了四个套接字类型：SOCK_STREAM（用于 TCP）、SOCK_DGRAM（用于 UDP）、SOCK_SEQPACKET（用于 SCTP），以及 SOCK_RAW（用于直接使用 ISP 服务的应用）。
- 协议。这个字段定义了族中特定协议。对于 TCP/IP 协议簇这个字段设置为 0，因为它是族中唯一的协议。
- 本地套接字地址。这个字段定义了本地套接字地址。一个套接字地址是一个结构，它由长度字段、族字段（对于 TCP/IP 协议簇，它被设置为常量 AF_INET）、端口号字段（定义了进程）以及 IP 地址字段（定义了正在运行的进程所在的主机）构成。它也包含未使用字段。
- 远程套接字地址。这个字段定义了远程套接字地址。它的结构与本地套接字地址相同。

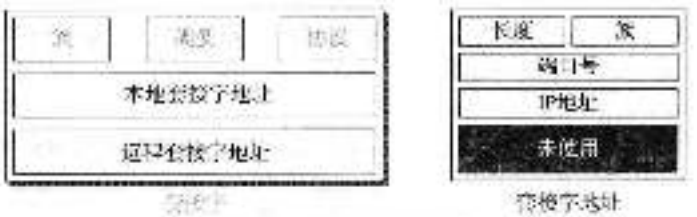


图 2-58 套接字数据结构

头文件

为了能够使用套接字的定义和所有在接口中定义的过程（函数），我们需要一组头文件。我们已经将所有这些头文件收集到了名为 headerFiles.h 的文件里。这个文件需要与程序创建到同一个文件夹中并且它的名字应该包含到所有程序中。

```
// "headerFiles.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/wait.h>
```

使用 UDP 迭代通信

正如我们之前讨论的，UDP 提供无连接服务器，其中客户发送请求，服务器返回响应。

用于 UDP 的套接字

在 UDP 通信中，客户和服务器每一端只使用一个套接字。服务器端创建的套接字永远运行；客户端创建的套接字在客户进程结束时被关闭（销毁）。图 2-59 给出了服务器和客户进程套接字的生存期。换言之，不同的客户使用不同的套接字，但是服务器只创建一个套接字，并且每次当一个新客户

建立连接时只改变远程套接字地址。这是符合逻辑的，因为服务器确实知道自身的套接字地址，但是不知道需要服务的客户端的套接字地址；在填充套接字的这一项之前，它需要等待客户进行连接。

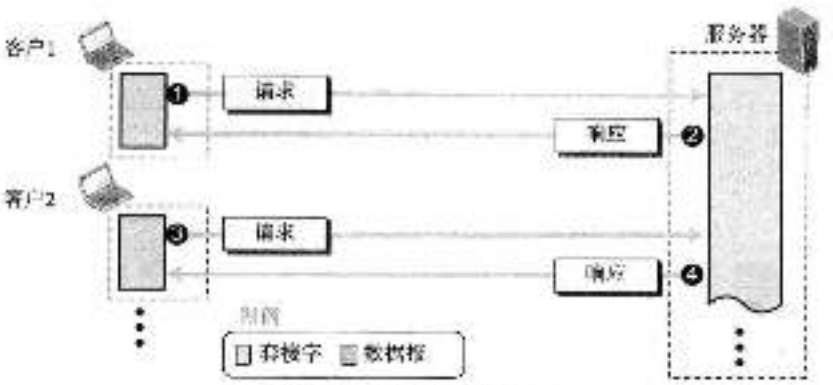


图 2-59 UDP 运行套接字

通信流程图

图 2-60 给出了一个简单的迭代通信的流程图。图中有多个客户，但是只有一个服务器。每个客户在每次循环中得到服务。请注意，这里没有连接建立以及连接终止。每个客户发送一个数据报并接收一个数据报。换言之，如果一个客户想要发送两个数据报，它就被认为是两个客户。第二个数据报需要等待循环轮次。

服务器进程 服务器进行被动开启（passive open），在被动开启中它做好连接准备，但是等待客户进程创建连接。它调用 socket 函数去创建套接字。在这个程序调用中的参数填充了前三个字段，但是本地和远程套接字地址字段仍然未定义。之后，服务器进程调用 bind 函数来填充本地套接字地址字段（信息来自操作系统）。然后，它调用另一个称为 recvform 的函数。然而这个函数阻塞服务器进程直到一个客户数据报到达。当一个数据报到达时，服务器进程解除阻塞并且从数据报中抽出数据报。它也抽取发送套接字地址用来在下一步中使用。在请求被处理之后，响应就准备好了，服务器进程将接收报文中的发送套接字地址填充到远程套接字地址，这样就完成了套接字结构。现在准备发送数据报。这通过调用另一个称为 sendto 的函数来完成。请注意，在服务器进程发送响应之前，套接字中的所有字段都应该被填充。在发送响应之后，服务器进程开始一个新的迭代并且等待其他客户连接。远程套接字地址字段将被再次填充上一个新的客户地址（或是同一个客户，而此处认为是新客户）。服务器进程是一个无限的进程；它永远运行。服务器套接字从不关闭，除非出现了问题且进程需要终止。

客户进程 客户进程进行*主动开启*（active open），换言之，它开启连接。它调用 socket 函数来创建一个套接字并填充前三个字段。尽管某些实现要求客户进程也调用 bind 函数来填充本地套接字，但通常这是由操作系统自动完成的，操作系统为客户选择一个临时端口号。之后，客户进程调用 sendto 函数，并提供远程套接字地址信息。这个套接字地址必须由客户进程的用户提供。套接字在这个时刻完成并且数据报被发送。现在客户进程调用 recvfrom 函数，它阻塞了客户进程，直到响应来自服务器进程。此处没有必要从这个函数中提取远程套接字地址，因为此处没有调用 sendto 函数。换言之，服务器端和客户端的 recvfrom 函数行为不同。在服务器进程中，recvfrom 函数首先被调用，然后调用 sendto，因此 sendto 所使用的远程套接字地址可以从 recvfrom 获得。在客户进程，sendto 在 recvfrom 之前被调用，因此远程地址应该由程序使用者使用，使用者知道她想连接的是哪一台服务器。最终 close 函数被调用以销毁套接字。注意，客户进程是有限的；在响应

被接收之后，客户进程终止。尽管我们可以使用一个循环来设计发送多个数据报的客户，但是每次循环迭代对于服务器都像一个新的客户。

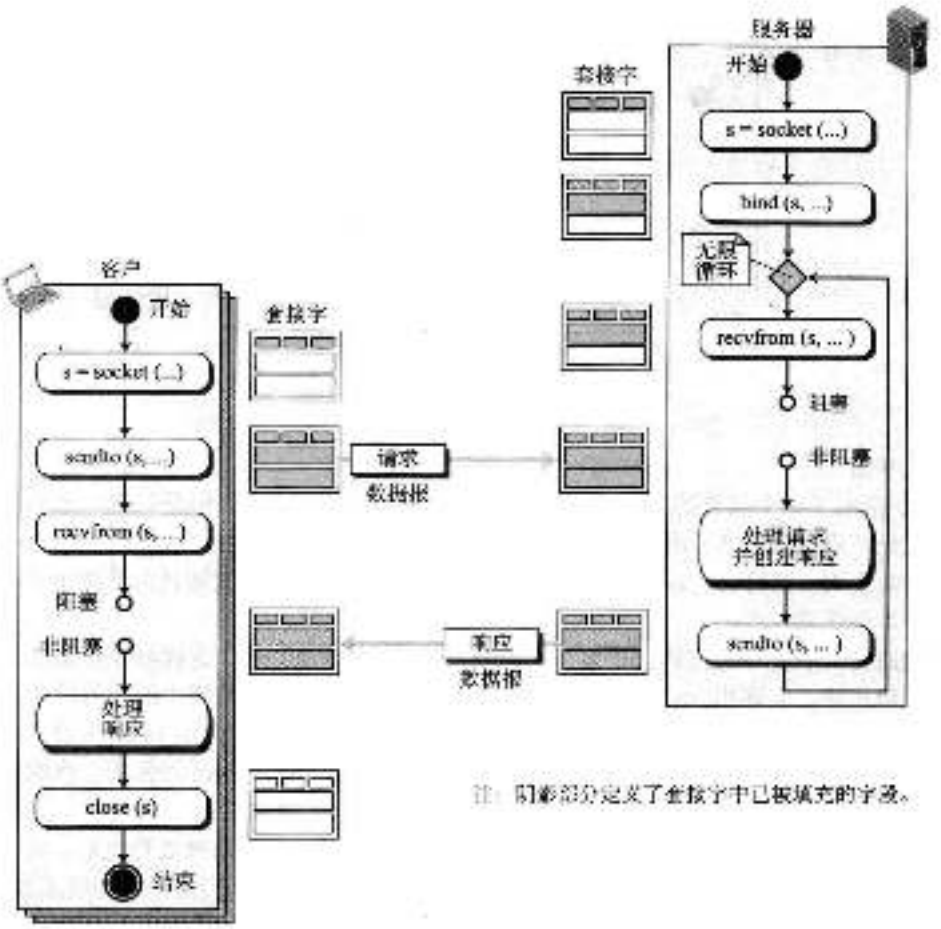


图 2-60 迭代 UDP 通信的流程图

编程举例

在这一节，我们给出如何进行客户和服务器编程来模拟使用 UDP 的标准回送（echo）应用。客户程序向服务器发送一个短的字符串；服务器回送检验这个字符串。标准应用供客户计算机使用，来检测另一台服务器计算机是否活动。我们的程序比标准使用中的程序更简单；为简单起见，我们省略了一些错误检测和调试细节。

回送服务器程序 表 2-22 给出了使用 UDP 的回送服务器编程。程序依照图 2-60 中的流程图。

表 2-22 使用 UDP 的回送服务器程序

```
1  // UDP 回送服务器程序
2  #include "headerFiles.h"
3  int main (void)
4  {
5     // 声明以及定义变量
6     int s;                              // 套接字描述符（引用）
```

```
7   int len;                                    // 要回送的字符串的长度
8   char  buffer [256];                         // 数据缓冲区
9   struct sockaddr_in servAddr;                // 服务器（本地）套接字地址
10  struct sockaddr_in cintAddr;                // 客户（远程）套接字地址
11  int  cintAddrLen;                           // 客户套接字地址的长度
12  // 建立本地（服务器）套接字地址
13  memset (&servAddr, 0, sizeof (servAddr));                   // 分配内存
14  servAddr.sin_family =  AF_INET;                             // 族字段
15  servAddr.sin_port = htons (SERVER_PORT);                    // 默认端口号
16  servAddr.sin_addr.s_addr = htonl (INADDR_ANY);              // 默认 IP 地址
17  // 创建套接字
18  if ((s = socket (PF_INET, SOCK_DGRAM, 0) < 0);
19  {
20      perror ("Error: socket failed!");
21      exit (1);
22  }
23  // 将套接字绑定到本地服务器端口
24  if (bind (s, (struct sockaddr*) &servAddr, sizeof (servAddr)) < 0);
25  {
26      perror ("Error: bind failed!");
27      exit (1);
28  }
29  for ( ; )      // 永远运行
30  {
31      // 获取字符串
32      len = recvfrom (s, buffer, sizeof (buffer), 0,
33              (struct sockaddr*)&cintAddr, &cintAddrLen);
34      // 发送字符串
35      sendto (s, buffer, len, 0, (struct sockaddr*)&cintAddr, sizeof(cintAddr));
36  }// 循环结束
37 }// 回送服务器程序结束
```

第 6 行到第 11 行声明并定义了程序中使用的变量。第 13 行到第 16 行为服务器套接字地址分配内存（使用 memset 函数）并且使用传输层提供的默认值填充了套接字地址字段。为了插入端口号，我们使用 htons（host to network short）函数，它将主机字节序格式的短整型数值转换为网络字节序格式。为了插入 IP 地址，我们使用 htonl（host to network long）函数来做相同的事情。

第 18 行到第 22 行在 if 声明中调用 socket 函数来检查错误。因为如果调用失败，这个函数就返回−1，程序打印错误消息并且退出。perror 函数是 C 中的标准错误函数。类似地，第 24 行到第 28 行调用 bind 函数来将套接字绑定到服务器套接字地址。这个函数也在 if 声明中被调用，从而检查错误。

第 29 行到第 36 行使用无限循环在每次迭代中为客户提供服务。第 32 到第 33 行调用 recvfrom 函数读取客户发送的请求。注意，这个函数是阻塞函数；当它消除阻塞时，它接收请求报文，同时提供客户套接字地址来完成套接字的最后部分。第 35 行调用 sendto 函数来给客户发回（回送）相同的报文，它使用 recvfrom 报文中获得的客户套接字地址。注意这里没有对请求报文进行加工；服务器仅回送它所收到的内容。

回送客户程序 表 2-23 给出了使用 UDP 的回送客户程序。程序依照图 2-60 中的流程图。

表 2-23 使用 UDP 的回送客户程序

```
// UDP回送客户端程序
#include "headerFiles.h"
int main (int argc, char* argv[ ])                    // 三个参数之后的检验
{
    // 声明并定义变量
    int  s;                                           // 套接字描述符
    int  len;                                         // 收回的字符串的长度
    char* servName;                                   // 服务器名
    int  servPort;                                    // 服务器端口
    char* string;                                     // 要回送的字符串
    char buffer[256 + 1];                             // 数据缓冲区
    struct sockaddr_in servAddr;                      // 服务器套接字地址
    // 检测命令行参数个数
    if (argc != 3)
    {
        printf ("Error: three arguments are needed!");
        exit(1);
    }
    servName = argv[1];
    servPort = atoi (argv[2]);
    string = argv[3];
    // 将服务器套接字地址
    memset (&servAddr, 0, sizeof (servAddr));
    servAddr.sin_family = AF_INET;
    inet_pton (AF_INET, servName, &servAddr.sin_addr);
    servAddr.sin_port = htons (servPort);
    // 创建套接字
    if ((s = socket (PF_INET, SOCK_DGRAM, 0)) < 0);
    {
        perror ("Error: Socket failed!");
        exit (1);
    }
    // 发送回送字符串
    len = sendto (s, string, strlen (string), 0, (struct sockaddr)&servAddr, sizeof (servAddr));
    // 接收回送字符串
    recvfrom (s, buffer, len, 0, NULL, NULL);
    // 利用计数回送字符串
    buffer [len] = '\0';
    printf ("Echo string received: "
    fputs (buffer, stdout);
    // 关闭套接字
    close (s);
    // 正常终止
    exit (0);
} // UDP回送客户程序结束
```

第 6 行到第 12 行声明并定义了程序中使用的变量。第 14 行到第 21 行检测并设置了程序运行时提供的参数。头两个参数提供了服务器名与服务器端口号；第三个参数是被回送的字符串。第 23 到第 26 行分配内存、调用函数 inet_pton 将服务器名转换成服务器 IP 地址、将端口号转换为适当的字节序。inet_pton 函数调用 DNS（在本章前面讨论过）。这三部分信息是 sendto 函数所需要的，都被存储在适当的变量中。

第 34 行调用 sendto 函数发送请求。第 36 行调用 recvfrom 函数接收回送报文。请注意报文中的两个参数是 NULL，因为我们不需要取出远程站点的套接字地址；报文已经被发送了。

第 38 行到第 40 行用来在屏幕上显示回送报文，从而达到调试的目的。注意，在第 38 行我们在回送报文尾部加入了一个空字符，使得其他内容可以在下一行显示。最终，第 42 行关闭套接字，第 44 行离开程序。

使用 TCP 通信

正如我们之前描述的，TCP 是面向连接的协议。在发送或接收数据之前，需要在客户端和服务器之间建立连接。在连接建立之后，只要它们有数据要发送或接收，两端就可以彼此发送以及接收数据块。TCP 连接可以是迭代的（一次服务一个客户）也可以是并发的（一次服务多个客户）。在本节，我们只讨论迭代方法。并发方法参见 Java 部分（见第 11 章）。

TCP 中使用的套接字

TCP 服务器使用两个不同的套接字，一个用于连接建立，一个用于数据传输。我们把第一个称为监听套接字（listen socket）并把第二个称为套接字（socket）。设置两种套接字的目的是将建立阶段和数据交换阶段分开。服务器使用监听套接字来监听试图建立连接的新客户，在连接建立之后，服务器创建一个用于和客户交换数据的套接字并且最终终止连接。客户只使用一个套接字用于连接建立以及数据交换（见图 2-61）。

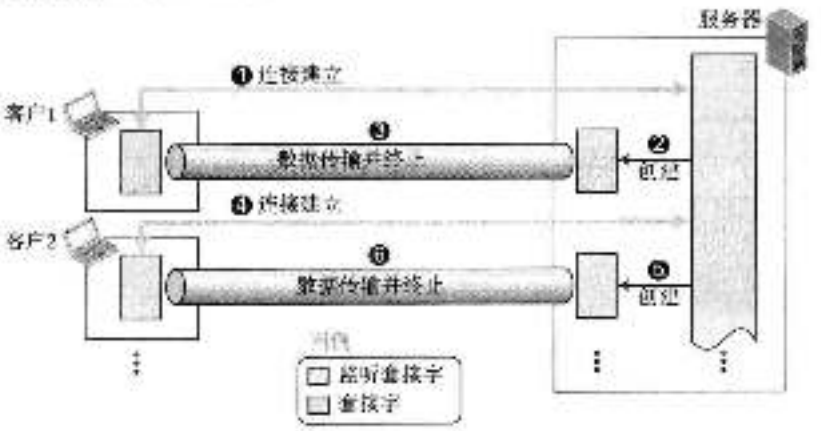


图 2-61 TCP 通信中使用的套接字

通信流程图

图 2-62 给出了简化的迭代通信流程图。图中有多个客户但是只有一个服务器。每个客户在每次循环中被服务。这幅流程图与 UDP 中的流程图类似，但是有一些不同，我们会对每一端进行解释。

服务器进程 在图 2-62 中，和 UDP 服务器进程一样，TCP 服务器进程调用 socket 和 bind 函数，但是这两个函数创建监听套接字，它只在连接建立阶段被使用。之后，服务器进程调用 listen 函数，允许操作系统开始接收客户、完成连接阶段并把他们放入等待被服务的列表。这个函数也定义了被连接的客户等待列表的大小，这依赖于服务器进程的复杂性，但是通常值为 5。

现在，服务器进程开始循环并且逐一对客户进行服务。在每次循环中，服务器进程调用 accept

函数从已连接客户的等待列表中去除一个客户，对其进行服务。如果列表是空的，那么 accept 函数进入阻塞状态直到出现一个客户待服务。当 accept 函数返回，它创建一个新的与监听套接字一样的套接字。监听套接字现在移入后台，并且新的套接字成为活动套接字。服务器进程现在使用连接建立期间获得的客户套接字地址，用它来填充新建套接字的远程套接字地址。

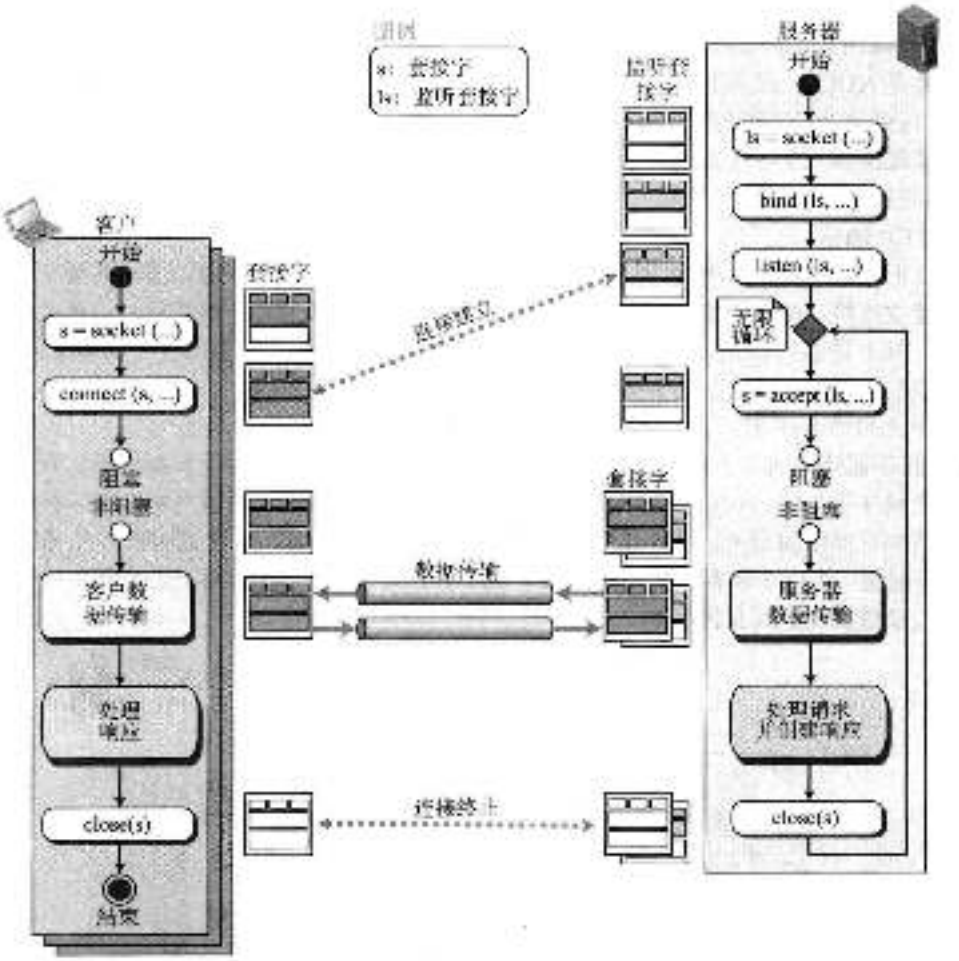


图 2-62 迭代 TCP 通信流程图

此时，客户和服务器可以交换数据。我们没有给出数据传输的特定方式，因为这取决于特定的客户-服务器对。TCP 使用 send 以及 recv 程序在它们之间传输数据字节。这两个函数比 UDP 中使用的 sendto 和 recvfrom 函数更简单，因为它们不提供远程套接字地址；连接已经在客户和服务器之间建立。然而，由于 TCP 用于传输无边界报文，每个应用需要仔细设计数据传输部分。send 和 recv 函数可能被调用多次来处理大量数据传输。可以将图 2-62 中的流程图当作一个通用流程图；如果是特殊用途，需要定义服务器数据传输（sever data-transfer）盒。当我们讨论回送客户-服务器程序时，我们将这样做一个简单的例子。

客户进程 客户流程图与 UDP 版本类似，除了客户数据传输（client data-transfer）盒需要为每个特定情况定义。当稍后我们编写特定程序时，我们会这么做。

编程实例

在这一节，我们给出如何编写客户和服务器程序来模拟使用 TCP 的标准回送应用。客户程序

发送一个短的字符串给服务器；服务器将相同的字符串回送到客户。然而，在我们这样做之前，我们需要为客户和服务器数据传输盒提供流程图，如图 2-63 所示。

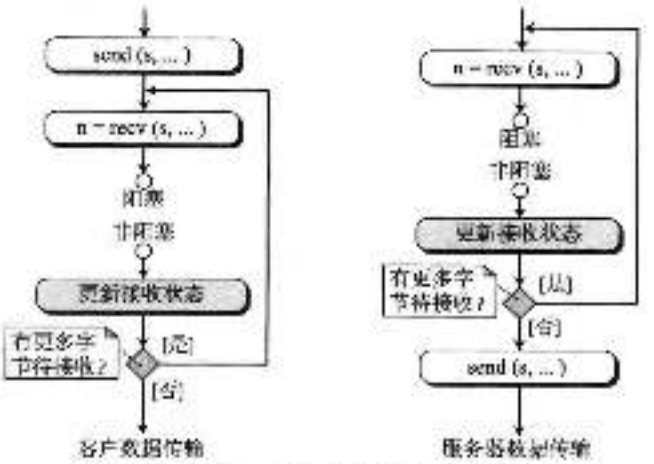


图 2-63 客户和服务器数据传输盒的流程图

对于这个特定的情况，因为待发送的字符串很短（小于几个单词），我们可以在客户端调用 send 函数一次完成。然而，TCP 并不保证把整个报文在一个报文段内发送。因此，我们需要在服务器端调用一组 recv（在一个循环内）来接收每个报文并将它们收集到缓冲区内，从而能一次性发送回去。当服务器向客户发送回送报文时，它也可能使用多个报文段，这意味着客户的 recv 程序需要调用多少次就会被调用多少次。

另一个有待解决的问题是设置缓冲区，缓冲区用于在每个站点接收数据。我们需要控制接收的字节数以及下一个数据块存储的位置。如图 2-64 所示，程序设置了一些变量进行控制。在每次迭代中，指针（ptr）移动指向下一个要接收的字节，接收字节的长度（len）呈增长趋势并且待接收的最大字节数（maxLen）呈减少趋势。

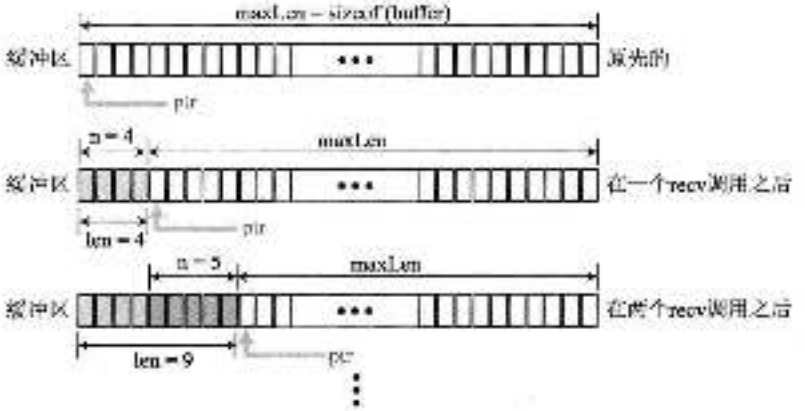


图 2-64 用于接收的缓冲区

考虑到以上两点之后，现在我们编写服务器和客户程序。

回送服务器程序 表 2-24 给出了使用 TCP 的回送服务器程序。程序遵循图 2-62 中的流程图。在图中，阴影中的每一部分对应于一条指令。带斜线部分显示了数据传输部分要做的处理工作。

第 6 行到第 16 行声明并定义了变量。第 18 行到第 21 行分配内存并且按 UDP 情况下所述创建了

本地（服务器）套接字地址。第23行到第27行创建了监听套接字。第29行到第33行将监听套接字绑定到第18行到第21行创建的服务器套接字地址上。第35行到第39行是TCP通信中的新内容。

调用listen函数让操作系统完成连接建立阶段并将客户置入等待列表。第44行到第48行调用accept函数来移除等待列表中的第一个客户并开始为其服务。如果在等待列表中没有客户，那么这个函数处于阻塞状态。第50行到第56行对图2-63中描述的数据传输部分进行编码。最大缓冲区大小与回送字符串长度都和图2-64中所示相同。

回送客户程序 表2-25给出使用TCP的回送客户程序。这个程序遵循图2-62的流程图。在图中，阴影中的每一部分对应于一条指令。背斜线部分显示了数据传输部分要做的处理工作。

表2-24 使用TCP服务的回送服务器程序

```
1  // 回送服务器程序
2  #include "headerFiles.h"
3  int main (void)
4  {
5      // 变量声明
6      int ls;                                  // 监听套接字描述符（引用）
7      int s;                                   // 套接字描述符（引用）
8      char buffer [256];                       // 数据缓冲区
9      char* ptr = buffer;                      // 数据缓冲区
10     int len = 0;                             // 本次调用接收的字节数
11     int maxLen = sizeof (buffer);            // 最大接收字节数
12     int n = 0;                               // 每次调用 recv 接收的字节数
13     int waitSize = 16;                       // 等待客户数量
14     struct sockaddr_in serverAddr;           // 服务器地址
15     struct sockaddr_in clientAddr;           // 客户地址
16     int cliAddrLen;                          // 客户地址长度
17     // 创建地址（服务器）套接字地址
18     memset (&servAddr, 0, sizeof (servAddr));
19     servAddr.sin_family = AF_INET;
20     servAddr.sin_addr.s_addr = htonl (INADDR_ANY);     // 默认 IP 地址
21     servAddr.sin_port = htons (SERV_PORT);             // 默认端口
22     // 创建监听套接字
23     if (ls = socket (PF_INET, SOCK_STREAM, 0) < 0);
24     {
25         perror ("Error: Listen socket failed!");
26         exit (1);
27     }
28     // 将套接字绑定到本地套接字地址
29     if (bind (ls, &servAddr, sizeof (servAddr)) < 0);
30     {
31         perror ("Error: binding failed!");
32         exit (1);
33     }
34     // 监听连接请求
35     if (listen (ls, waitSize) < 0);
36     {
37         perror ("Error: listening failed!");
```

```
        exit (1);
    }
    // 处理连接
    for ( ; ; )                          // 永远运行
    {
        // 接收来自客户的连接
        if (s = accept (ls, &clntAddr, &clntAddrLen) < 0);
        {
            perror ("Error: accepting failed!");
            exit (1);
        }
        // 数据传输部分
        while ((n = recv (s, ptr, maxLen, 0)) > 0)
        {
            ptr + = n;                   // 在缓冲区上移动指针
            maxLen - = n;                // 以整个接收的最大字节数
            len + = n;                   // 更新已接收的字节数
        }
        send (s, buffer, len, 0);        // 发回（回送）所有接收的字节
        // 关闭套接字
        close (s);
    } //整环结束
}// 回送服务器程序结束
```

表 2-25 回送客户程序

```
// TCP 回送客户程序
#include "headerFiles.h"
int main (int argc, char* argv[ ])          // 三个参数之后待检验
{
    // 声明并定义
    int  s;                                 // 套接字程序行
    int  n;                                 // 每次调用 recv 接收的字节数
    char* servName;                         //服务器名
    int servPort;                           //服务器端口号
    char* string;                           //被回送的字符串
    int  len;                               // 被回送的字符串的长度
    char  buffer [256 + 1];                 //缓冲区
    char* ptr = buffer;                     // 在缓冲区上移动指针
    struct sockaddr_in serverAddr;          // 服务器套接字地址
    // 检测并设置参数
    if (argc != 3)
    {
        printf ("Error: three arguments are needed!");
        exit (1);
    }
    servName = arg [1];
    servPort = atoi (arg [2]);
```

```
23      string = arg [3];
24      // 创建进程（服务器）套接字地址
25      memset (&servAddr, 0, sizeof(servAddr));
26      servAddr.sin_family = AF_INET;
27      inet_pton (AF_INET, servName, &serverAddr.sin_addr); // 转换器IP地址
28      serverAddr.sin_port = htons (servPort);                    // 服务器端口号
29      // 创建套接字
30      if ((s = socket (PF_INET, SOCK_STREAM, 0) < 0);
31      {
32          perror ("Error: socket creation failed!");
33          exit (1);
34      }
35      // 连接到服务器
36      if (connect (sd, (struct sockaddr*)&servAddr, sizeof(servAddr)) < 0);
37      {
38          perror ("Error: connection failed!");
39          exit (1);
40      }
41      // 数据传输部分
42      send (s, string, strlen(string), 0);
43      while ((n = recv (s, ptr, maxLen, 0)) > 0)
44      {
45          ptr + = n;                          // 在接收区上移动指针
46          maxLen - = n;                       // 调整待接收的最大字节数
47          len += n;                           // 更新已接收的字节数
48      }// while 循环结束
49      // 为串并验证所返的字符串
50      buffer [len] = '\0';
51      printf ("Echoed string received: ");
52      fputs (buffer, stdout);
53      // 关闭套接字
54      close (s);
55      // 终止程序
56      exit (0);
57  }// 回送客户程序结束
```

TCP 的客户程序与 UDP 的客户程序非常相似，只有些许不同。因为 TCP 是面向连接的协议，第 36 行到第 40 行调用 connect 函数连接服务器。第 42 行到第 48 行使用图 2-63 中的思想完成数据传输。按图 2-64 所示方式完成接收数据的长度调整和指针移动。

2.6 章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。在本书末列出了方括号中的参考资料。

书籍

此处列出一些涵盖本章内容的书籍，它们包括[Com 06]、[Mir 07]、[Ste 94]、[Tan 03]和[Bar et al. 05]。

RFC

HTTP 在 RFC2068 和 2109 中讨论到。FTP 在 RFC 959、2577 和 2585 中讨论。TELNET 在 RFC 854、855、856、1041、1091、1372 和 1572 中讨论。SSH 在 RFC 4250、4251、4252、4253、4254 和 4344 中讨论。DNS 在 RFC1034、1035、1996、2535、3008、3658、3755、3757、3845、3396 以及 3342 中讨论。SMTP 在 RFC2821 和 2822 中讨论。POP3 在 RFC 1939 中有解释。MIME 在 RFC 2046、2047、2048 和 2049 中讨论。

小结

因特网中的应用或使用客户-服务器模式或使用对等模式。在客户-服务器模式中，一个应用程序称为服务器，它提供服务，另一个应用程序称为客户，它接受服务。服务器程序是一个无限程序；客户程序是有限的。在对等模式中，一个对等结点既可以是一个客户也可以是一个服务器。

万维网（WWW）是信息宝库，它把全世界的结点连接在一起。超文本和超媒体文档通过指针互相连接。超文本传输协议（HTTP）是万维网（WWW）上用于访问数据的主要协议。

文件传输协议（FTP）是一个 TCP/IP 客户-服务器应用，它用于从一个结点向另一个结点拷贝件。FTP 要求为数据传输提供两个连接：一个控制连接和一个数据连接。在非近似系统的之间的通信中 FTP 使用 NVT ASCII。

电子邮件是因特网最常见的应用。电子邮件体系结构包含几个部分，比如用户代理（UA）、报文传输代理（MTA）以及报文访问代理（MAA）。实现 MTA 的协议称为简单邮件协议（SMTP）。有两个协议用于实现 MAA：邮局协议版本 3（POP3）和因特网邮件访问协议 4（IMAP4）。

TELNET 是客户-服务器应用，它允许用户登录一台远程计算机，并使得用户能够访问远程系统。当用户通过 TELNET 进程访问远程系统时，这相当于分时环境。

域名系统（DNS）是一个客户-服务器应用，它用唯一的名称标识因特网中的主机。DNS 以层次结构组织名字空间，以此来将命名中所涉及的责任分散。

在对等网络中，准备好分享它们资源的因特网用户成为了对等结点并形成一个网络。对等网络分为中心化和分布式两种。在中心化 P2P 网络中，目录系统使用客户-服务器模式，但是文件存储和下载是通过对等模式完成的。在分布式网络中，目录系统以及文件的存储和下载都通过对等模式完成。

2.7 习题集

测试题

本章的交互式测试题请参见这本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q2-1 假设我们加入一个新的协议到应用层上。我们需要对其他层做什么改动？

Q2-2 请解释在客户-服务器模式中哪个实体提供服务，哪个实体接收服务？

Q2-3 在客户-服务器模式中，请解释为什么服务器应该一直运行而客户可以在需要时运行。

Q2-4 可否在一个只安装了 TCP 作为传输层协议的计算机上编写一个使用 UDP 服务的程序？请解释。

Q2-5 在周末，Alice 经常要从她家的笔记本电脑访问存储在办公室桌面的文件。上个星期，她在办公室桌面上安装了 FTP 服务器进程并在家里的笔记本上安装了 FTP 客户进程。当她周末无法访问文件时，感到很失望。哪里错了吗？

Q2-6 安装在个人计算机上的绝大多数操作系统有很多客户进程，但通常没有服务进程。请解释原因。

Q2-7 要设计一个使用客户-服务器模式的新应用。如果在客户和服务器之间只需要交换小报文而不用担心报文丢失或出错，你推荐使用什么传输层协议？

Q2-8 以下哪项可以是数据源？

a. 键盘　　b. 显示器　　c. 套接字

Q2-9 源套接字地址是 IP 地址和端口号的组合。请解释每部分定义的内容。

Q2-10 请解释客户运程如何得到要插入远程套接字地址中的 IP 地址和端口号。

Q2-11 如果一个 HTTP 请求需要在服务器端运行一个程序并且将结果下载到客户服务器，这个程序是以下的

a. 静态文档　　b. 动态文档　　c. 活动文档

Q2-12 假设我们设计了一个新的客户-服务器应用程序，它需要持续连接。我们能使用 UDP 作为传输层协议吗？

Q2-13 Alice 有一个视频片段，Bob 想得到它；Bob 有另一个视频片段，Alice 想得到它。Bob 创建了一个网页并在 HTTP 服务器上运行。Alice 如何得到 Bob 的视频片段？Bob 能够得到 Alice 的吗？

Q2-14 当 HTTP 服务器接收到来自 HTTP 客户的请求报文时，服务器如何知道所有头部以及跟随其后的报文主体何时到达？

Q2-15 在非持续 HTTP 连接中，HTTP 如何通知 TCP 协议报文已到达结尾？

Q2-16 在日常生活通信中，你能找到与 FTP 的控制和数据连接相似的两个分离的连接吗？

Q2-17 FTP 使用两个不同的熟知端口号用于控制和数据传输。这是否意味着为了交换控制信息和数据而创建了两个分离的 TCP 连接？

Q2-18 FTP 使用 TCP 服务交换控制信息并进行数据传输。FTP 能够为这两种连接使用 UDP 服务吗？请解释。

Q2-19 在 FTP 中，哪个实体（客户或服务器）开启（主动开启）控制连接？哪个实体开启（主动开启）数据传输连接？

Q2-20 如果控制连接在 FTP 会话结束前被中断，你认为会发生什么？它会影响数据连接吗？

Q2-21 在 FTP 中，如果客户需要从服务器站点获取一个文件并且存储一个文件到服务器站点上，这需要多少个控制连接和数据传输连接？

Q2-22 在 FTP 中，服务器可以从客户端获取文件吗？

Q2-23 在 FTP 中，服务器能够从客户端得到文件列表或目录吗？

Q2-24 FTP 可以在两个使用不同操作系统的主机之间传输文件。这是什么原因？

Q2-25 在控制连接期间，FTP 有用于交换命令和响应的报文格式吗？

Q2-26 在文件传输期间，FTP 有用于交换文件或目录/文件列表的报文格式吗？

Q2-27 在 FTP 中能够只有控制连接而没有数据连接吗？请解释。

Q2-28 在 FTP 中能够只有数据连接而没有控制连接吗？请解释。

Q2-29 假设我们需要使用 FTP 下载一个音频。我们可以在命令中指定什么文件类型？

Q2-30 HTTP 和 FTP 都可以从服务器获取文件。当我们下载文件时使用什么协议？

Q2-31 HELO 和 MAIL FROM 命令在 SMTP 中都是必需的吗？为什么？

Q2-32 在图 2-20 的文字中，信封中的 MAIL FROM 与头部的 FROM 有什么区别？

Q2-33 Alice 在长途旅行中，没有查看她的电子邮件。之后她发现丢失了一些朋友们声称已经发给她的电子邮件或附件。这其中的问题可能是什么？

Q2-34 假设一个 TELNET 客户使用 ASCII 表示字符，但是 TELNET 服务器使用 EBCDIC 代表字符。当两者字符表示不同，客户如何登录到服务器上？

Q2-35 TELNET 应用程序中没有像 FTP 或 HTTP 中允许用户传输文件或访问页面的命令。这种应用程序在哪个方面有用？

Q2-36 一台主机能够使用一个 TELNET 客户获得 FTP 或 HTTP 等其他客户-服务器应用所提供的服务吗？

Q2-37 在 DNS 中，以下哪一项是 FQDN，哪一项是 PQDN？

a. xxx　　b. xxx.yyy.net　　c. zzz.yyy.xxx.edu.

Q2-38 在基于 DHT 的网络中，假设 $m = 4$。如果一个结点标识符的散列值为 18，这个结点在 DHT 空间中的位置在哪里？

Q2-39 在基于 DHT 的网络中，假设结点 4 有一个文件的关键字为 18。最接近关键字 18 的结点是 20。这个文件存储在哪里？

a. 按照直接方法　　b. 按照间接方法

Q2-40 在 Chord 网络中，我们有结点 N5 和关键字 k5。N5 是 k5 的前向结点还是后向结点？

Q2-41 在 Kademlia 网络中，标识符空间的大小为 1024。二叉树的高度（树根到叶结点的距离）是多少？每

个结点的子列数量是多少？每个路由表有多少行？

Q2-42 在 Kademlia 中，假设 $m=4$ 并且活动结点是 N4、N7 和 N12。k3 存储在系统的哪个位置？

思考题

P2-1 假设一个服务器的域名是 www.commco.com。

a. 给出一个想要获取文件/usr/users/doc 的 HTTP 请求。客户接收 MIME 1 版、GIF 或 JPEG 图片。但是文档的存在时间不应该超过 4 天。

b. 给出与 a 成功请求对应的 HTTP 响应。

P2-2 在 HTTP 中，画一幅图来给出 cookie 的应用，cookie 的应用场景是服务器只允许注册用户访问服务器。

P2-3 在 HTTP 中，画一幅图来给出网络门户中 cookie 的应用，请使用两个站点来表示。

P2-4 在 HTTP 中，画一幅图来给出 cookie 的应用，cookie 的应用场景是服务器使用 cookie 来做广告，请只用三个站点。

P2-5 请画出给出代理服务器的使用，代理服务器是客户网络的一部分：

a. 给出当响应存储在代理服务器时，客户、代理服务器以及目标服务器之间的事务。

b. 给出当响应不存储在代理服务器时，客户、代理服务器以及目标服务器之间的事务。

P2-6 在第 1 章，我们得到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。HTTP 有表示层特征吗？

P2-7 HTTP 1.1 版将持续连接定义为默认连接。使用 RFC2616 给出客户或服务器如何将这个默认设置改为非持续连接？

P2-8 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。HTTP 有会话层特征吗？

P2-9 在 SMTP 中，一个发送者发送未格式化文本，请给出 MIME 头部。

P2-10 在 SMTP 中：

a. 一个非 ASCII 码 1000 字节的报文使用 base64 编码。编码报文中有多少字节？有多少冗余字节？冗余字节与整个报文的比是多少？

b. 一个非 ASCII 码 1000 字节的报文使用引用可打印编码。报文包含 90%ASCII 字符和 10%非 ASCII 字符。编码报文中有多少字节？有多少冗余字节？冗余字节与整个报文的比是多少？

c. 比较前两个情况的结果。如果报文是 ASCII 和非 ASCII 字符报文的组合，效率提高多少？

P2-11 请用 base64 编码如下报文：

01010111 00001111 11110000

P2-12 请用引用可打印编码如下报文：

01001111 10101111 01110001

P2-13 根据 RFC1939，POP3 会话是以下四种状态之一：关闭、认证、事务或更新。画图给出这四种状态以及 POP3 如何在这些状态之间移动。

P2-14 POP3 协议有一些基本命令（每个客户-服务器需要实现）。使用 RFC1939 中的信息，找出以下基本命令的含义和用法：

a. STAT　　b. LIST　　c. DELE 4

P2-15 POP3 协议有一些可选命令（每个客户-服务器需要实现）。使用 RFC1939 中的信息，找出以下可选命令的含义和用法：

a. UIDL　　b. TOP 1 15　　c. USER　　d. PASS

P2-16 使用 RFC1939，假设 POP3 客户处于下载保持状态。如果服务器从服务器下载两个 192 字节和 300 字节报文，请给出客户和服务器之间的事务。

P2-17 使用 RFC1939，假设 POP3 客户处于下载保持状态。如果服务器从服务器下载两个 230 字节和 400 字节报文，请给出客户和服务器之间的事务。

P2-18 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。SMTP 有表示层特征吗？

P2-19 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。SMTP 或 POP3 有会话层特征吗？

P2-20 在 FTP 中，假设用户名为 John 的客户需要在服务器的目录/top/videos/general 上存储一个名为 video2 的视频片段。如果选择临时端口号 56002，给出服务器和客户之间交换的命令和响应。

P2-21 在 FTP 中，一个用户（Jane）想要使用临时端口 61017 从/usr/users/report 目录下获取一个名为 huge 的 EBCDIC 文件。文件很大，以至于用户想要在传输前压缩它。给出所有命令和响应。

P2-22 在 FTP 中，一个用户（Jan）想要在目录/usr/usrs/letters 下创建一个名为 Jan 的新目录。给出所有命令和响应。

P2-23 在 FTP 中，一个用户（Maria）想要将一个名为 file1 的文件从/usr/users/report 目录移动至/usr/top/letters 目录下。注意，这里有文件被重命名的情况。我们首先需要给出旧文件名然后定义新名称。给出所有命令和响应。

P2-24 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有表示层。但是如果需要，应用层协议可以包含某些表示层定义的特征。FTP 有表示层特征吗？

P2-25 在第 1 章，我们提到 TCP/IP 协议簇与 OSI 模型不同，它没有会话层。但是如果需要，应用层协议可以包含某些会话层定义的特征。FTP 有会话层特征吗？

P2-26 在 Chord 中，假设标识符空间大小为 16。活动结点是 N3、N6、N8 以及 N12。给出 N6 的指针表（只给出目标关键字和后向结点列）。

P2-27 在 Chord 中，假设 N12 的后向结点是 N17。N12 是以下哪个关键字的前向结点：

a. k12　　b. k15　　c. k17　　d. k22

P2-28 在使用 DHT 的 Chord 网络中，$m = 4$，请画出标识符空间，其中放置 4 个对等结点，它们的 ID 是 N3、N8、N11 以及 N13，以及三个关键字 k5、k9 以及 k14。请判断哪个结点负责哪个关键字。为每个结点创建一个指针表。

P2-29 在 Chord 网络中，$m = 4$，结点 N2 指针表的值如下：N4、N7、N10 以及 N12。对于以下每个关键字，首先查找 N2 是否是关键的前向结点。如果不是，应该联系哪个结点（最接近前向结点）来帮助 N2 找到前向结点。

a. k1　　b. k6　　c. k9　　d. k13

P2-30 重复文中的例 2.16，但是假设结点 N12 需要找到负责关键字 k7 的结点。提示：请记住在模 32 运算中需要进行区间检测。

P2-31 重复文中的例 2.16，但是假设结点 N5 需要找到负责关键字 k16 的结点。提示：请记住在模 32 运算中需要进行区间检测。

P2-32 在 Pastry 中，假设地址空间是 16 并且 $b = 2$。地址空间中有多少位？列出一些标识符。

P2-33 在 Pastry 中 $m = 32$ 并且 $b = 4$，路由表以及叶子结点集的大小是多少？

P2-34 Pastry 的地址空间是 16 并且 $b = 2$。给出路由表的大纲。给出结点 N21 路由表中每个单元格的可能表项。

P2-35 在使用 DHT 的 Pastry 网络中，$m = 4$ 且 $b = 2$，请画出含 4 个结点 3 个关键字的标识符空间。4 个结点是 N02、N11、N20 以及 N23，3 个关键字是 k00、k12 以及 k24。请判断哪个结点负责哪个关键字。也请给出每个结点的叶子结点集和路由表。尽管它不是真实的，但是假设每两个结点间的接近度是基于数值接近程度度量的。

P2-36 在之前的问题中，请回答如下问题：

a. 给出结点 N02 如何回应一个寻找负责 k24 结点的请求。

b. 给出结点 N20 如何回应一个寻找负责 k12 结点的请求。

P2-37 使用文中图 2-55 的二叉树，给出结点 N11 的子树。

P2-38 使用文中图 2-56 中的路由表，如果结点 N0 接收到一个查询报文，它寻找负责 k12 的结点，请解释并给出路由。

P2-39 在 Kademlia 网络中 $m = 4$，我们有 5 个活动结点：N2、N3、N7、N10 以及 N12。找出每个活动结点的路由表（只有一列）。

P2-40 如文中图 2-60 所示，服务器如何知道一个客户已经请求了服务？

P2-41 如文中图 2-62 所示，服务器端如何为数据传输创建一个套接字？

P2-42 假设我们想要使表 2-25 中的 TCP 客户程序更具通用性，从而能够发送字符串并处理从服务器接收到的响应，如何做到这一点？

2.8 模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在第 1 章，我们下载并安装了 Wireshark，学习它的基本特性。在本章，我们使用 Wireshark 来捕获并研究一些应用层协议。这里使用 Wireshark 来模拟六种协议：HTTP、FTP、TELNET、SMTP、POP3 以及 DNS。

Lab2-1 在第 1 个实验中，我们使用 HTTP 获取网页。我们使用 Wireshark 来捕获分组进行分析。我们学习最普通的 HTTP 报文。我们也捕获响应报文并分析它们。在实验期间，我们也研究分析一些 HTTP 头部。

Lab2-2 在第 2 个实验中，我们使用 FTP 传输一些文件。我们使用 Wireshark 捕获分组。我们展示出 FTP 使用两个分离的连接：一个控制连接和一个数据传输连接。每次文件传输活动中数据连接都会被打开然后关闭。我们也展示出 FTP 是一个不安全的文件传输协议，因为每项事务都是用明文完成的。

Lab2-3 在第 3 个实验中，我们使用 Wireshark 捕获 TELNET 协议交换的分组。如 FTP 中一样，我们能够在被捕获的分组里观察到会话中的命令和响应。正如 FTP，TELNET 易受攻击，因为它用明文发送包括口令在内的所有数据。

Lab2-4 在第 4 个实验中，我们研究运行中的 SMTP 协议。我们发送一个电子邮件，使用 Wireshark 研究客户和服务器之间交换的 SMTP 分组的内容和格式。我们检查文中讨论的 SMTP 会话中的三个阶段。

Lab2-5 在第 5 个实验中，我们研究 POP3 协议的状态和行为。我们获取存储在 POP3 服务器邮箱中的邮件，通过分析那些经过 Wireshark 的报文来观察并分析 POP3 的状态以及交换报文的类型和内容。

Lab2-6 在第 6 个实验中，我们分析 DNS 协议的行为。除了 Wireshark，许多网络实用工具可用于查找 DNS 服务器中存储的信息。在这个实验中，我们使用 dig 命令（被 nslookup 所代替）。我们也使用 ipconfig 来管理主机中缓存的 DNS 记录。当我们使用这些工具时，我们用 Wireshark 来捕获这些工具发送的分组。

2.9 编程作业

利用你选择的编程语言，编写源代码、编译并测试如下程序：

Prg2-1 编写程序，使表 2-22 中的 UDP 服务器程序更通用：接收请求、处理请求并发回响应。

Prg2-2 编写程序，使表 2-23 中的 UDP 客户程序更通用：能够发送客户程序创建的任何请求。

Prg2-3 编写程序，使表 2-24 中的 TCP 服务器程序更通用：接收请求、处理请求并发回响应。

第3章

Computer Networks: A Top-Down Approach

传 输 层

TCP/IP 协议簇中的传输层位于应用层和网络层之间。它为应用层提供服务，并接收来自网络层的服务。传输层是客户程序和服务器程序之间的联络人，是一个进程到进程的连接。传输层是 TCP/IP 协议簇中的核心；它是因特网上从一点到另一个点传输数据的端到端逻辑传输媒介。这一章有些冗长，因为我们需要讲几个问题并介绍一些新概念。

- 3.1 节介绍通常从传输层获得的一般服务，如进程到进程通信、寻址、多路复用、多路分解、差错、流以及拥塞控制。
- 3.2 节讨论普通的传输层协议，如停止-等待协议、回退 N 帧协议以及选择重复协议。这些协议关注实际传输层协议所提供的传输流和差错控制服务。理解这些协议可以帮助我们更好地理解 UDP 和 TCP 协议的设计，以及其设计背后的逻辑。
- 3.3 节关注 UDP，这是本章讨论的两个协议中较简单的一个。UDP 缺乏许多我们想要从传输层协议得到的服务，但是正如我们所给出的，它的简单性对一些应用很有吸引力。
- 3.4 节讨论 TCP。我们首先列出它的服务和特性。然后我们使用转换图来给出 TCP 如何提供面向连接的服务。最终，我们使用 TCP 中的滑动窗口给出流和差错控制。之后讨论 TCP 中的拥塞控制，这个主题会在第 4 章网络层中重新讨论。我们也会给出一个列表以及 TCP 计时器的功能，它们在 TCP 提供的不同服务中都有所使用。本书网站列出的一些 TCP 常用的选项可用于深入研究。

3.1 介绍

传输层位于应用层和网络层之间。它在两个应用层之间提供进程到进程服务，一个进程在本地主机，另一个在远程主机。使用逻辑连接提供通信，意味着两个应用层可以位于地球上的不同位置，两个应用层假设存在一条想象的直接连接，通过这条连接它们可以发送和接收报文。图 3-1 给出了逻辑连接背后的思想。

这幅图给出的场景和我们在第 2 章里表示应用层的场景（图 2-1）相同。Alice 那台位于 Sky Research 公司的主机在传输层创建了一条逻辑连接，它连接到 Bob 在 Scientific Books 公司的主机。两个公司在传输层通信，好像它们之间有一条真正的连接。图 3-1 展示出只有两个终端系统（Alice 和 Bob 的计算机）使用传输层服务；所有中间路由器只使用前三层。

传输层服务

正如我们在第 1 章讨论的，传输层位于网络层和应用层之间。传输层负责向应用层提供服务；它接收来自网络层的服务。在这一节，我们讨论传输层提供的服务；在下一节，我们讨论几个传输层协议。

进程到进程通信

传输层协议的首要任务是提供进程到进程通信（process-to-process communication）。进程是使用传输层服务的应用层实体（运行着的程序）。我们在讨论进程到进程通信如何实现之前，需要理解主机到主机通信与进程到进程通信的不同之处。

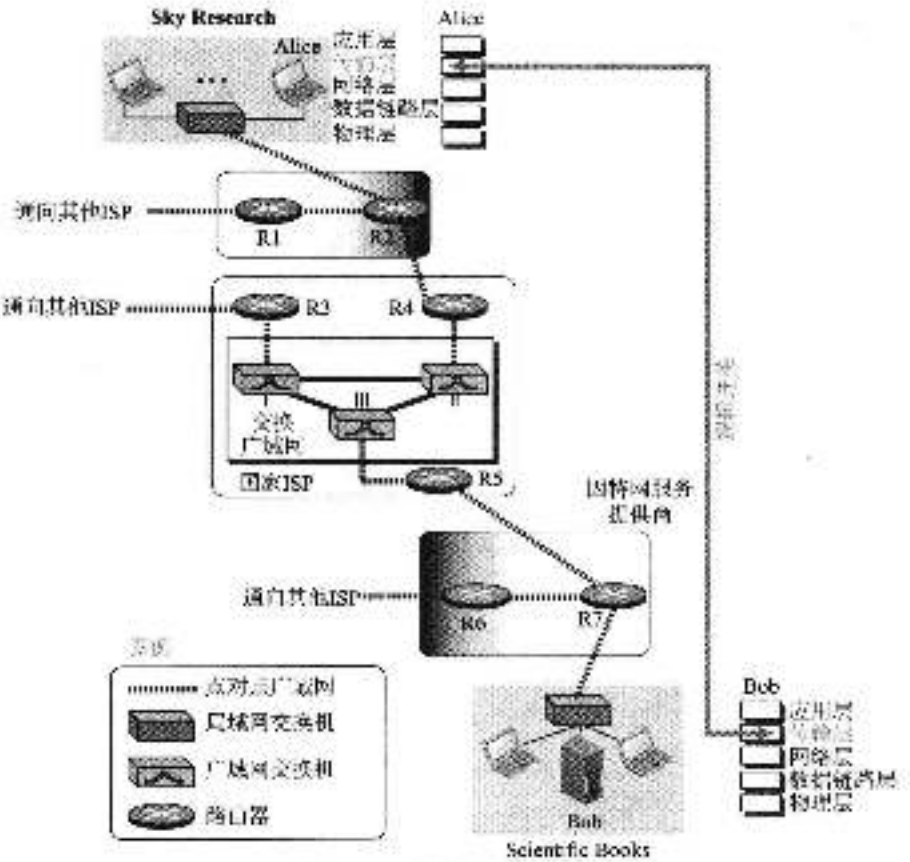


图 3-1 传输层的逻辑连接

网络层（第 4 章讨论）负责计算机层次的通信（主机到主机通信）。网络层协议只把报文传递到目的计算机。然而，这是不完整的传递。报文仍然需要递交给正确的过程。这正是传输层接管的部分。传输层协议负责将报文传输到正确的进程。图 3-2 给出了网络层和传输层的范围。

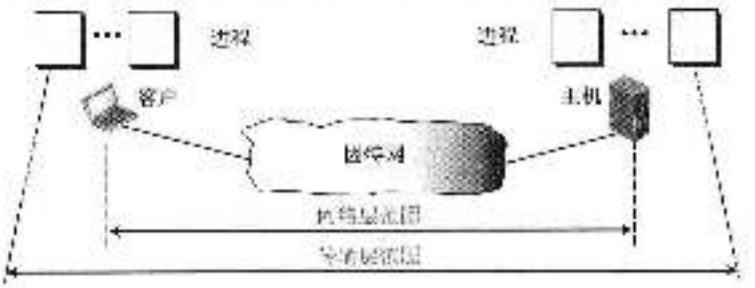


图 3-2 网络层与传输层

寻址：端口号

尽管有一些方法可以实现进程到进程通信，但是最常用的是通过客户-服务器模式（client-server paradigm，见第 2 章）。本地主机上的进程称为客户，它通常需要来自远程主机上的进程提供的服务，这种远程主机称为服务器。

这两个进程（客户和服务器）有相同的名字。例如，如果要从远程机器上获得日期和时间，我们需要在本地主机上运行 Daytime 客户进程并在远程机器上运行 Daytime 服务进程。

然而，目前的操作系统支持多用户和多程序运行的环境。一个远程计算机在同一时间可以运行多个服务器程序，就像许多本地计算机可在同一时间运行一个或多个客户应用程序一样。对通信来说，我们必须定义本地主机、本地进程、远程主机以及远程进程。我们使用 IP 地址来定义本地主机和远程主机（将在第 4 章讨论）。为了定义进程，我们需要第二个标识符，称为端口号（port number）。在 TCP/IP 协议族中，端口号是在 0 到 65 535 之间的 16 位整数。

客户程序用端口号定义它自己，这称为临时端口号（ephemeral port number）。临时这个词表示短暂的（short-lived）。它之所以被使用是因为客户的生命周期通常很短。为了客户-服务器程序能正常工作，临时端口号推荐值为大于 1 023。

服务器进程必须使用一个端口号定义它自己。但是，这个端口号不能随机选择。如果服务器站点的计算机运行一个服务器进程，并随机分配一个数字作为端口号，那么在客户站点的进程想访问该服务器并使用其服务时，将不知道此端口号。当然，一个解决方法是发送一个特殊分组，并请求一个特定服务器的端口号，但是这需要更多的开销。TCP/IP 决定使用全局端口号；它们称为熟知端口号（well-known port number）。但是这条规则也有例外，例如，有一些客户端也被分配了熟知端口号。每一个客户进程都知道相应服务器进程的熟知端口号。例如，前面所讨论的 Daytime 客户进程可以使用临时（暂时）端口号 52 000 来表示自己，但服务器 Daytime 进程必须使用熟知（永久）端口号 13。图 3-3 表示了这一概念。

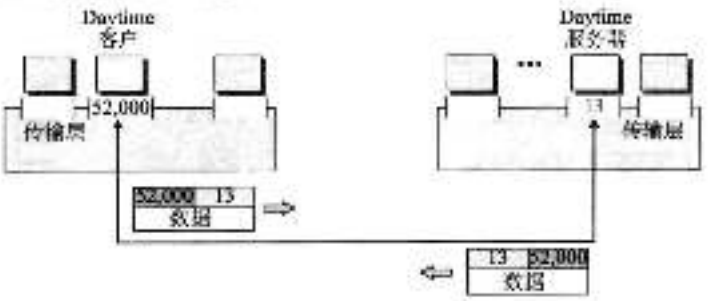


图 3-3 端口号

现在应该搞清楚，在选择数据的最终目的地时，IP 地址和端口号起着不同的作用。目的 IP 地址在世界范围的不同主机中确定一个主机。但主机被选定后，端口号定义了在该特定主机上的多个进程中的一个进程（见图 3-4）。

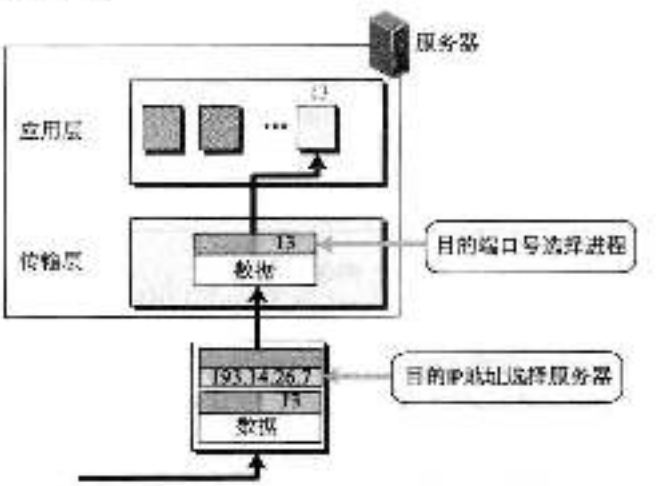


图 3-4 IP 地址和端口号

ICANN 范围

ICANN（见附录 D）已经把端口号编码划分为三种范围：熟知的、注册的和动态的（或私有的），如图 3-5 所示。



图 3-5 ICANN 范围

- 熟知端口。端口号的范围是 0~1023，由 ICANN 分配和控制。这些是熟知端口号。
- 注册端口。端口号的范围是 1024~49151，ICANN 不分配也不控制。它们可在 ICANN 注册以防重复。
- 动态端口。端口号的范围是 49152~65535。这一范围内的端口号既不受控制又不需要注册，可以由任何进程使用。它们是临时或私有端口号。

例 3.1 在 UNIX 中，熟知端口号存储在/etc/services 文件中。这个文件的每行给出服务器名和熟知端口号。我们可用 grep 命令提取该行所对应的应用。下面表示了 TFTP 的端口。注意，TFTP 可使用 UDP 或 TCP 的端口 69。

```
$grep tftp/etc/services
tftp 69/tcp
tftp 69/udp
```

SNMP（见第 9 章）使用两个端口号（161 和 162），每一个用于不同目的。

```
$grep snmp/etc/services
snmp161/tcp#Simple Net Mgmt Proto
snmp161/udp#Simple Net Mgmt Proto
snmptrap162/udp#Traps for SNMP
```

套接字地址

在 TCP 协议簇中的传输层协议需要 IP 地址和端口号，它们各在一端建立一条连接。一个 IP 地址和一个端口号结合起来称为套接字地址（socket address）。客户套接字地址唯一地定义了客户进程，而服务器套接字地址唯一地定义了服务器进程（见图 3-6）。

为了使用因特网中的传输层服务，我们需要一对套接字地址：客户套接字地址和服务器套接字地址。这四条信息是网络层分组头部和传输层分组头部的组成部分。第一个头部包含 IP 地址，而第二个头部包含端口号。

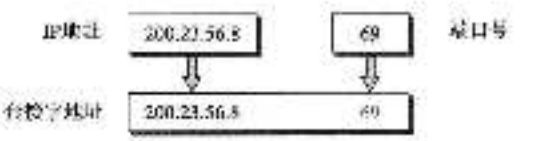


图 3-6 套接字地址

封装与解封装

为了将报文从一个进程发送到另一进程，传输层协议负责封装与解封装报文（见图 3-7）。封装在发送端发生。当进程有报文要发送，它将报文与一组套接字地址和其他信息一起发送到传输层，这依赖于传输层协议。传输层接收数据并加入传输层头部。因特网中传输层的分组称为用户数据报（user datagram）、段（segment）或分组（packet），这取决于我们使用什么传输层协议。在一般讨论中，我们将传输层有效载荷称为分组。

解封装发生在接收端。当报文到达目的传输层，头部被丢弃，传输层将报文传递到应用层运行的进程。如果需要响应接收到的报文，发送方的套接字地址被发送到进程。



图 3-7 封装与解封装

多路复用与多路分解

每当一个实体从一个以上的源接收到数据项时，称为多路复用（multiplexing，多对一）；每当一个实体将数据项传递到一个以上的源时，称为多路分解（demultiplexing，一对多）。源端的传输层执行复用；目的端的传输层执行多路分解（见图 3-8）。

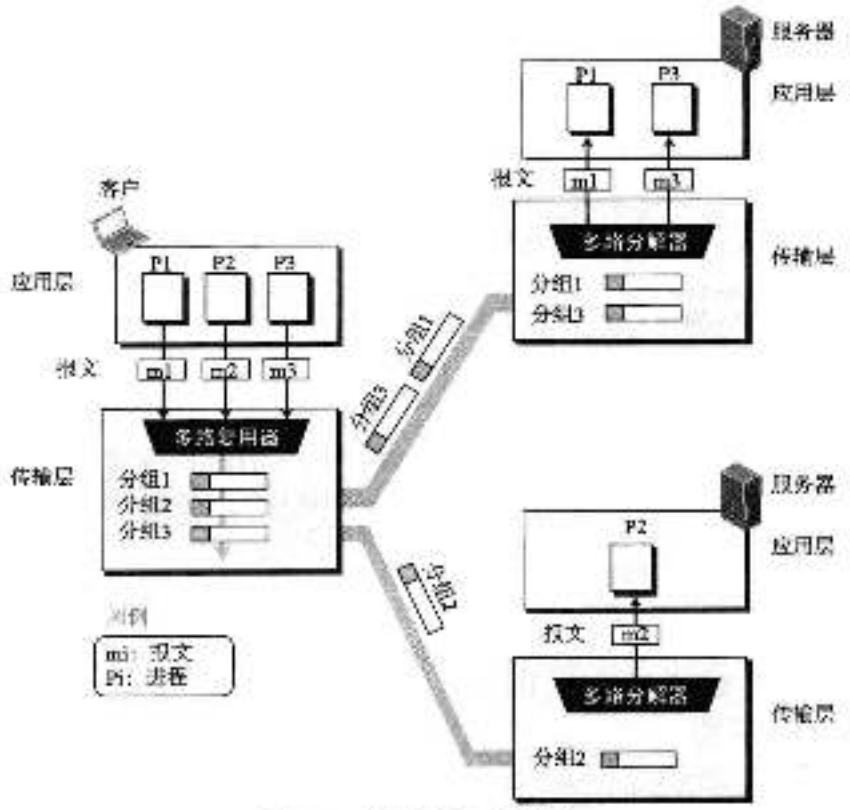


图 3-8 多路复用和多路分解

图 3-8 给出了一个客户和两个服务器之间的通信。客户端运行三个客户进程：P1、P2 和 P3。进程 P1 和 P3 需要将请求发送到对应的服务器进程。客户进程 P2 需要将请求发送到位于另外一个服务器的服务器进程。客户端的传输层接收到来自三个进程的三个报文并创建三个分组。它起到了多路复用器的作用。分组 1 和 3 使用相同的逻辑信道到达第一个服务器的传输层。当它们到达服务器时，传输层起到多路分解器的作用并将报文分发到两个不同的进程。第二个服务器的传输层接收分组 2 并将它传递到相应的进程。注意，尽管只有一个报文，我们仍然用到多路分解。

流量控制

每当一个实体创建数据项并且有另一个实体消耗它们时，就存在生产速率和消费速率的平衡问题。如果数据项生产比消费快，那么消费者可能被淹没并且可能要丢弃一些数据项。如果数据项生产比消费慢，那么消费者必须等待，系统就会变得低效。流量控制与第一种情况相关。我们需要在消费者端防止丢失数据项。

推或拉

从生产者传递数据项到消费者有两种方式：推（push）或拉（pull）。每当发送方生产数据项时，它无须事前获得消费者的请求就会发送它们——这种传递称为推。如果生产者在消费者请求这些数据项之后进行发送，这种传递称为拉。图 3-9 给出了这两种传递类型。

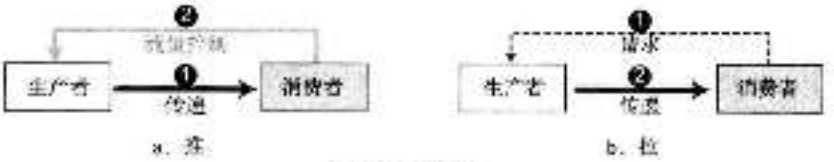


图 3-9 推或拉

当生产者推数据项时，消费者可能被淹没并需要相反方向的流量控制，以此来防止丢弃这些数据项。换言之，消费者需要警告生产者停止传递，并且当消费者再次准备好接收数据时通知生产者。当消费者拉数据项，它会在自身做好准备时进行请求。在这种情况下，不需要流量控制。

传输层流量控制

在传输层通信中，我们需要处理四个实体：发送方进程、发送方传输层、接收方传输层和接收方进程。应用层的发送方进程仅仅是一个生产者。它生产报文块，并把它们推到传输层。发送方传输层有两个作用：它既是消费者也是生产者。它消费生产者推来的报文。它将报文封装进分组并传递到接收方传输层。接收方传输层也有两个作用：它是消费者，消费从发送方那里接收来的分组；它也是生产者，解封装报文并传递到应用层。然而，最后的传递通常是拉传递：传输层等待直到应用层进程请求报文。

图 3-10 展示出，我们至少需要两种流量控制：从发送方传输层到发送方应用层的流量控制和从接收方传输层到发送方传输层的流量控制。

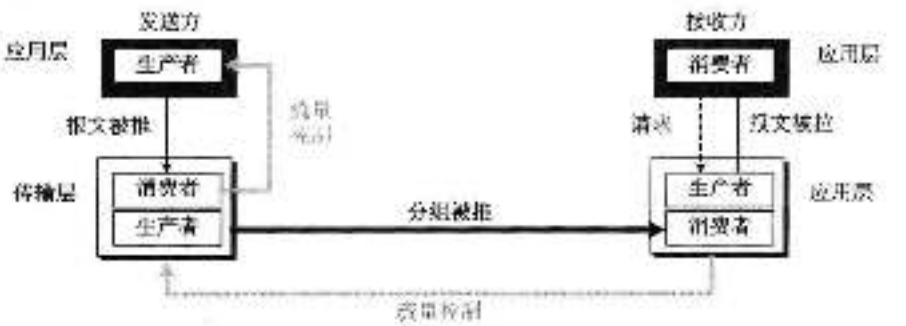


图 3-10 传输层流量控制

缓冲区

尽管流量控制可以用多种方式实现，但通常的方式是使用两个缓冲区：一个位于发送方传输层，另一个位于接收方传输层。缓冲区是一组内存单元，它可以在发送端和接收端存储分组。消费者向生产者发送信号从而进行流量控制通信。

当发送方传输层的缓冲区已满，它就通知应用层停止传输报文块；当有空闲位置时，它通知应用层可以再次传输报文块。

当接收方传输层的缓冲区已满，它就通知发送方传输层停止传输分组；当有空闲位置时，它通知发送方传输层可以再次传输分组。

例 3.2 上述讨论要求消费者与生产者在以下两种情况下进行通信：缓冲区满或缓冲区有空闲位置。如果双方使用的缓冲区只有一个存储单元，那么通信就会变得更简单。假设每个传输层使用一个存储单元来存储分组。当位于传输层的这个存储单元为空时，发送方传输层就会向应用层发送一个通知，要求发送下一个块；当位于接收方传输层的这个存储单元为空时，它向发送方传输层发送一个确认，要求发送下一个分组。然而，我们后文将会看到，这种流量控制在发送方和接收方只使用含有一个存储单元的缓冲区，非常低效。

差错控制

在因特网中，由于网络层（IP）是不可靠的，如果应用层需要可靠性，我们需要使传输层变得可靠。可靠性可以通过在传输层加入差错控制服务来实现。传输层的差错控制负责以下几个方面：

1. 发现并丢弃被破坏的分组。
2. 记录丢失和丢弃的分组并重传它们。
3. 识别重复分组并丢弃它们。
4. 缓冲失序分组直到丢失的分组到达。

差错控制不像流量控制，它仅涉及发送方和接收方传输层。我们假设在应用层和传输层之间交换的报文块是不会产生差错的。图 3-11 给出了发送方和接收方传输层的差错控制。正如传输控制，大多数情况下，接收方传输层管理差错控制，它通过告知发送方传输层存在问题来进行管理。



图 3-11 传输层的差错控制

序号

差错控制需要发送方传输层知道哪个分组要被重传并且接收方传输层需要知道哪个分组是重复的、哪个分组是失序的。如果分组是编号的，这个就可以实现。我们可以在传输层分组中加入一个字段来保存分组的序号（sequence number）。当分组被破坏或丢失，接收方传输层可按某种方式通知发送方传输层去利用序号重传分组。如果两个接收到的分组具有相同的序号，接收方传输层也能发现重复分组。可以通过观察序号的间隔辨别失序分组。

分组一般按序编号。然而，由于我们需要在头部包含每个分组的序号，因此需要设置一个界限。如果分组的头部允许序号最多为 m 比特位，那么序号范围就是 0 到 2^m-1。例如，如果 m 是 4，序号范围是 0 到 15 的闭区间。然而，我们可以回绕。因此，这种情况下，序号为

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

换言之，序号是模 2^m 的。

对于差错控制，序号是模 2^m 的，这里 m 是序号字段的大小，单位是比特。

确认

我们可以发送积极或消极的信号作为差错控制，但是我们只讨论积极信号，这在传输层中最常见。接收方可以为每一组正确到达的分组发送一个确认（ACK）。接收方可以简单地丢弃被破坏的分组。

发送方如果使用计时器，它就可以发现丢失分组。当一个分组被发送，发送方就开启一个计时器。如果 ACK 在计时器超时之前没有到达，那么发送方重发这个分组。重复的分组可以被接收方默默丢弃。失序的分组既可以被丢弃（被发送方当做丢失报文对待），也可以存储直到丢失的那个分组到来。

流量和差错控制的组合

我们已经讨论过，流量控制要求使用两个缓冲区，一个在发送端另一个在接收端。我们也已经讨论过差错控制要求两端均使用序号和确认号。如果我们使用两个带序号的缓冲区：一个位于发送端，一个位于接收端，那么这两个需要可以结合起来。

在发送端，当分组准备发送时，我们使用下一个缓冲区空闲位置号码 x 作为分组的序号。当分组被发送，一个分组的备份存储在内存位置 x，等待来自另一端的确认。当与被发送分组相关的确认到达时，分组被清除，内存位置空闲出来。

在接收端，当带有序号 y 的分组到达时，它被存储在内存位置 y 上，直到应用层准备好接收它。这时发送一个确认表明分组 y 的到达。

滑动窗口

由于序号进行模 2^m 操作，因此一个环可以代表从 0 到 2^m-1 的序号（见图 3-12）。缓冲区由一组片段代表，称为滑动窗口（sliding window），它随时占据环的一部分。在发送端，当一个报文被发送，相应的片段就被标记。当有所片段都被标记时，意味着缓冲区满且不能从应用层进一步读收报文。当确认到达时，相应片段被取消标记。如果从窗口开始处有一些连续的片段没有被标记，那么窗口滑过这些相应序号的范围，允许更多的片段进入窗口尾部。图 3-12 给出发送方的滑动窗口。序号以 16 为模（$m=4$）且窗口大小为 7。请注意滑动窗口仅仅是一个抽象：实际情况是使用计算机变量来保存下一个和最后一个待发送的分组。

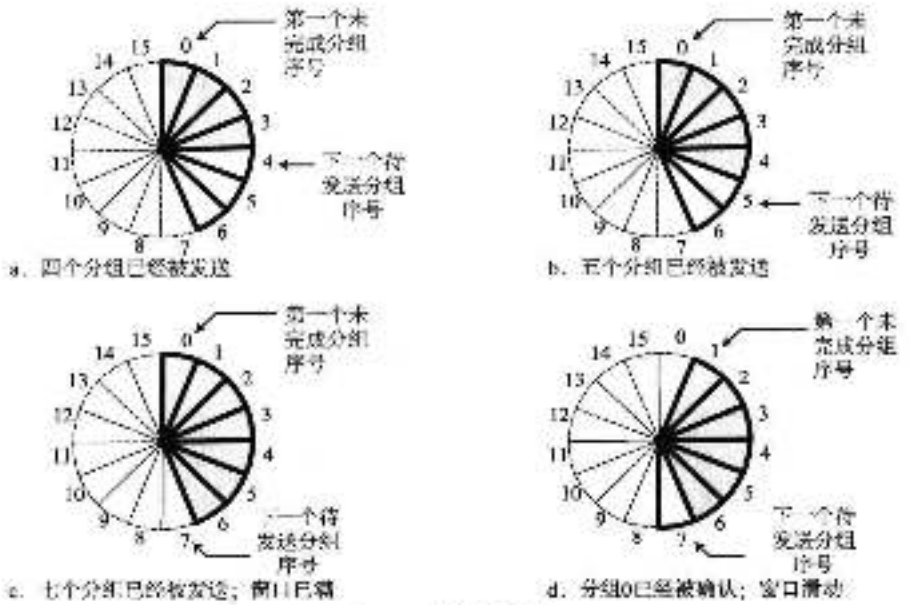


图 3-12 圆形滑动窗口

大多数协议使用线性形式来表示滑动窗口。虽然想法是相同的，但是它通常占用更少的页面空间。图 3-13 给出这种表示方法。这两种表示方法告诉我们相同的事情。如果拿起图 3-13 中每一幅图的两个端点，并且弯曲它们，我们就可以得到与图 3-12 中相同的图。

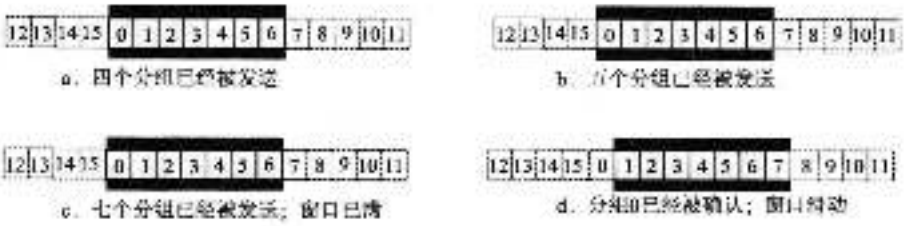


图 3-13 线性形式滑动窗口

拥塞控制

在因特网之类的分组交换网络中存在一个重要问题，这就是**拥塞**（congestion）。如果网络中的负载（load，即发送到网络的分组数）大于网络的容量（网络可以处理的分组数），那么网络就可能发生拥塞。**拥塞控制**（congestion control）指的是一种机制和技术，它控制拥塞并将负载保持在容量以内。

我们可能会问，为什么网络中会有拥塞。在任何系统中发生的拥塞都需要等待。例如，由于车流量不正常，如高峰时段的事故，高速公路上会发生拥塞导致交通堵塞。

在处理前后都存储分组的缓冲区时，由于路由器和交换机中有队列，网络或互联网会发生拥塞。比如，一个路由器，它的每一个接口都有一个输入队列和一个输出队列。如果路由器不能以分组到达的速率进行处理，队列就会超载，拥塞就会发生。传输层的拥塞实际上是网络层拥塞的结果。我们在第 4 章讨论网络层拥塞及其成因。本章后面，假设网络层没有拥塞控制，我们给出 TCP 如何实现它自己的拥塞控制机制。

无连接和面向连接服务

传输层协议就像网络层协议一样，可以提供两种类型的服务：无连接服务和面向连接服务。然而，这些传输层服务的本质与网络层不同。在网络层，无连接服务可能意味着属于同一个报文的不同数据报有不同路径。在传输层，我们不关心分组的物理路径（我们假设两个传输层之间有一条逻辑连接）。传输层的无连接服务意味着分组之间的独立；面向连接服务意味着依赖。接下来让我们仔细研究这两种服务。

无连接服务

在无连接服务中，源进程（应用程序）需要将报文分成传输层可接受大小的数据块，并把它们一个一个地传递到传输层。传输层将每一个数据块看做彼此没有关系的单元。当一个块从应用层到达时，传输层将其封装在分组中并发送。为了展示分组的独立，我们假设客户进程有三个报文块要发送给服务器进程。这些块被按序交给无连接传输协议。然而，由于传输层的这些分组之间没有联系，分组可能失序到达目的地并且被失序传递给服务器进程（见图 3-14）。

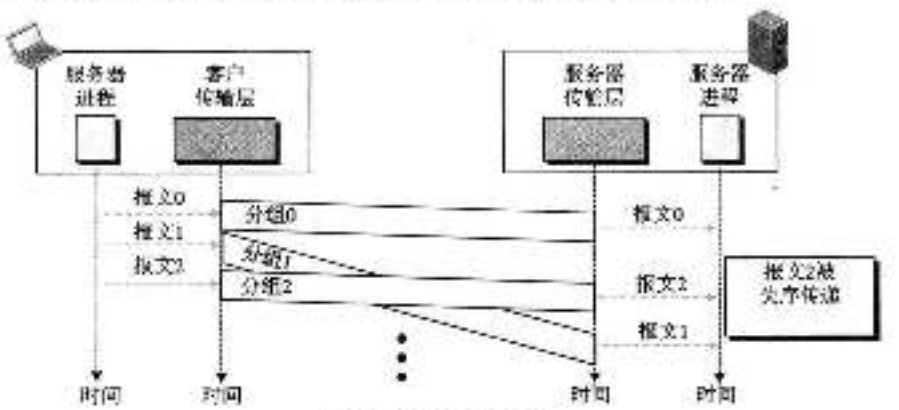


图 3-14 无连接服务

在图 3-14 中，我们使用时间轴给出分组的移动，但是我们假设进程到传输层的传递过程是瞬时的，传输层到进程的传递亦然。如图 3-14 所示，在客户端，三个报文块按序传送给客户传输层（0、1 和 2）。由于第二个分组在传输中的额外延迟，服务器报文的传递失序（0、2 和 1）。如果这三个数据块属于同一个报文，那么服务器进程可能会收到一个奇怪的报文。

如果一个分组丢了情况就更糟糕了。由于分组没有序号，接收方传输层不知道一个报文已经丢失。它仅仅将两个数据块传送到服务器进程。

以上两个问题是由于双方传输层没有互相协调所致。接收方传输层不知道第一个分组将要到来，也不知道所有的分组已经到来。

我们可以说，流量控制、差错控制以及拥塞控制都不能在无连接服务中有效实现。

面向连接服务

在面向连接的服务中，首先需要建立客户和服务器之间的逻辑连接。只有连接建立之后才能进行数据交换。在数据交换之后，连接需要拆除（见图 3-15）。

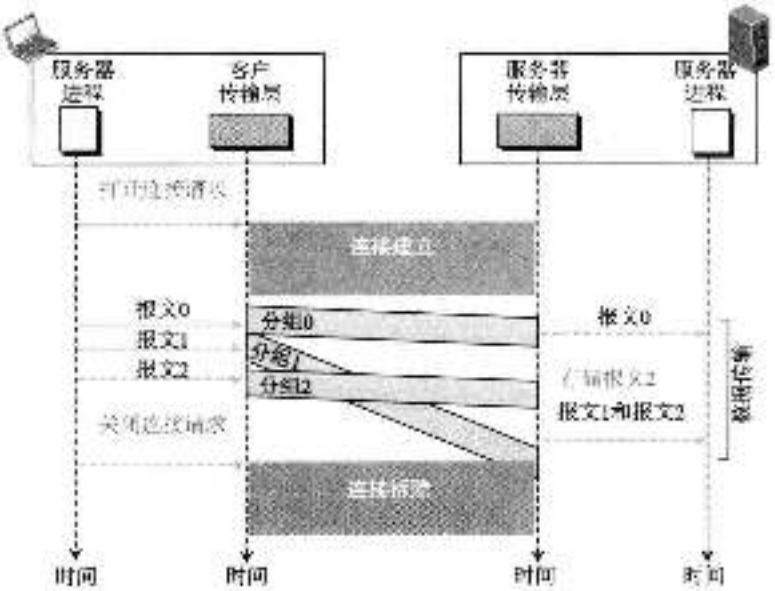


图 3-15 面向连接服务

如前所述，传输层的面向连接服务与网络层的面向连接服务不同。在网络层，面向连接服务意味着两个终端主机以及这之间的所有路由器都进行协调。在传输层，面向连接服务仅仅涉及两个主机；服务是端到端的。这意味着我们能够在传输层建立一个面向连接协议，其下的网络层可以是无连接协议也可以是面向连接协议。图 3-15 给出传输层面向连接服务中的连接建立、数据传输以及拆除阶段。

在面向连接协议中，我们可以实现流量控制、差错控制以及拥塞控制。

有限状态机

当提供无连接或面向连接服务时，传输层协议的行为都可以用有限状态机（finite state machine，FSM）来更好地表示。图 3-16 利用有限状态机给出了一幅传输层的图。使用这个工具，每个传输层（发送方或接收方）都被当做具有有限个状态的状态机来模拟。这个状态机总是处于其中一种状态，直到一个事件（event）发生才改变状态。每个事件与两种反应相关：定义将要执行的动作列表（可能是空的），以及决定下一个状态（可能与当前状态相同）。必须定义一种状态为初始状态，这个状态是状态机开启时的状态。在图 3-16 中，我们使用圆角矩形来表示状态，用带灰度的文本来表示事件，用普通黑色文本来表示动作。尽管我们之后用斜线代替了水平线，但是此处使用水平

线来分割事件和动作。箭头表示移动到下一个状态。

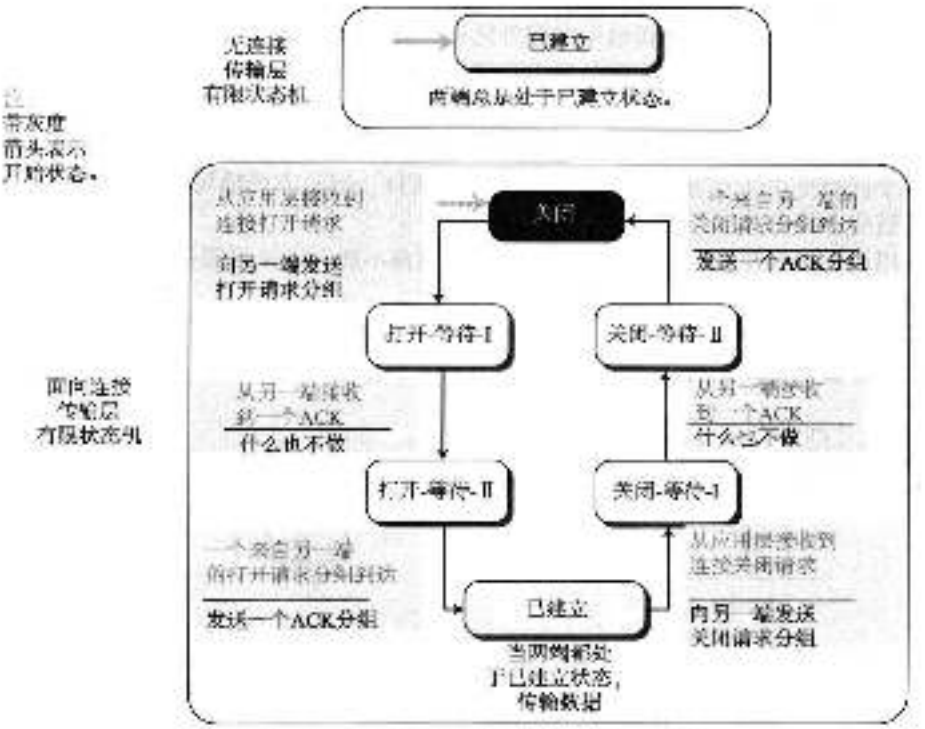


图 3-16 用 FSM 表示的无连接和面向连接服务

我们可以将无连接传输层看做只有一个状态的有限状态机：已建立状态。每一端的状态机（客户和服务器）总是处于已建立状态，准备发送并接收传输层分组。

换言之，在面向连接传输层的有限状态机中，在到达已建立状态之前需要经过三个状态。关闭连接之前状态机也需要经过三个状态。当不存在连接时，状态机处于关闭状态。它保持这个状态直到一个来自本地进程的打开连接的请求到达；状态机向运行传输层发送一个打开请求分组并将状态转移到打开-等待-I。当从另一端接收到确认，本地有限状态机进入打开-等待-II状态。当状态机处于这个状态时，单向连接已经建立，但是如果需要建立双向连接，状态机需要在这个状态继续等待，直到另一端也请求连接。当请求被接收时，状态机发送一个确认并进入已建立状态。

当两个终端都处于已建立状态时，数据和数据确认可以在它们之间交换。然而，我们需要记住，无连接和面向连接传输层中的已建立状态都代表一组数据传输状态，我们将在下一节传输层协议中讨论这个问题。

为了拆除连接，应用层向本地传输层发送一个关闭请求分组。传输层向另一端发送一个关闭请求分组，并进入关闭-等待-I状态。当接收到来自另一端的确认时，状态机进入关闭-等待-II状态，并且等待来自另一端的关闭请求分组。当这个分组到达时，状态机发送一个确认并进入关闭状态。

我们之后会讨论到，面向连接的有限状态机有很多种变化。我们也将看到有限状态机如何被压缩或扩展，以及状态名称如何变化。

3.2 传输层协议

我们通过将上一节描述的一组服务组合起来创建了传输层协议。为了更好地理解这些协议的行为，我们从一个最简单的协议开始并逐步深入。TCP/IP 协议或使用修改的传输层协议，或将几个

协议结合起来使用。我们在这一节讨论这些常见协议，为理解本章剩余的复杂协议铺平道路。为了使讨论简单，我们首先将这些协议作为单向协议（即单工）讨论，在单向协议中，数据分组沿着一个方向移动。在本章结尾部分，我们简要讨论它们如何变成双向协议，在双向协议中数据可以沿两个方向移动（即双工）。

3.2.1 简单协议

我们的第一个协议是一个简单的无连接协议，它既没有流量控制也没有差错控制。我们假设接收方能够立即处理它所收到的任何分组。换言之，接收方永远不会被接收到的分组淹没。图 3-17 给出了这种协议的框架。

发送方的传输层从发送方的应用层接收到报文，从中建立一个分组并发送它。接收方的传输层从网络层接收到这个分组，从分组中提取报文并传递到应用层。发送方和接收方的传输层都为应用层提供传输服务。

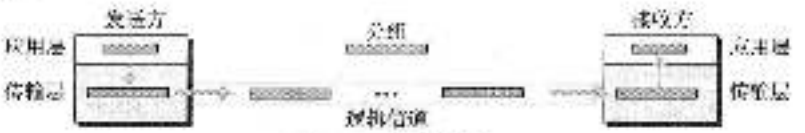


图 3-17 简单协议

有限状态机

直到应用层有报文待发送，发送端才发送分组。直到一个分组到达，接收端才将报文传递到它的应用层。我们可以使用两个有限状态机来表示这些要求。每个有限状态机只有一种状态，即准备状态（ready state）。发送方状态机保持准备状态，直到一个来自应用层进程的请求到来。当这个事件发生时，发送方状态机将报文封装在分组内，并将其发送到接收方状态机。接收方状态机保持准备状态，直到一个来自发送方状态机的分组到来。当这个事件发生时，接收方状态机从分组内解封装报文，并将其发送到应用层进程。图 3-18 给出了简单协议的有限状态机。我们之后会看到，UDP 协议是这个协议的轻微改动版本。

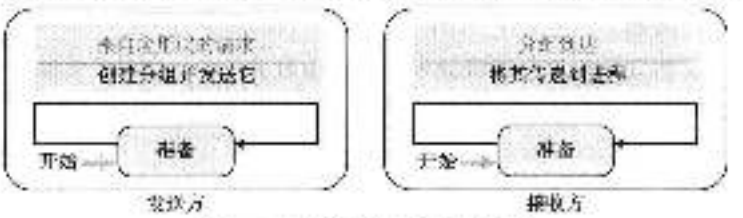


图 3-18 简单协议的有限状态机

例 3.3 图 3-19 给出了使用这种协议的通信示例。它非常简单。发送方一个接一个地发送分组，甚至不用考虑接收方能否承受。

3.2.2 停止-等待协议

我们的第二个面向连接协议称为停止-等待协议（Stop-and-Wait-protocol），它使用流量和差错控制。发送方和接收方都使用大小为 1 的滑动窗口。发送方在某一时刻发送一个分组，并且在发送下一个分组之前等待确认。为了发现被破坏分组，我们需要在每个数据分组中加入校验和。当一个分组到达接收端时，它就被检测。如果校验和不正确，分组就是被破坏的并被悄悄地丢弃。接收方的沉默对发送方来说是一种信号，即那个分组不是被破坏就是丢失了。每当发送方发送一个分组时，它都开启一个计时器。如果在计时器超时之前接收到确认，那么计时器就被关闭并且发送下一个分组（如果它有待发送分组）。如果计时器超时，发送方就认为分组丢失或被破坏，于是重发之前的分组。这意味着在确认到来之前，发送方都需要存储分组的副本。图 3-20 给出了停止-等待协议的框架。注意，信道中每次只能有一个分组或一个确认。

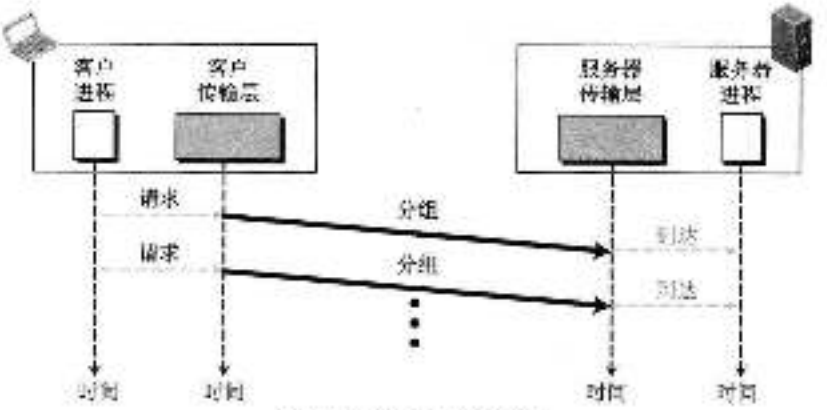


图 3-19 例 3.3 的流程图

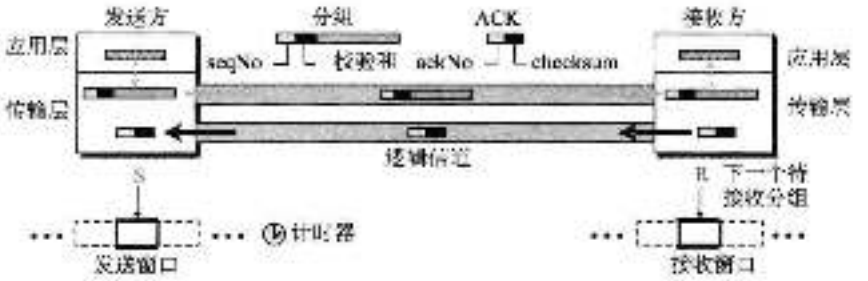


图 3-20 停止-等待协议

停止-等待协议是一个提供流量和差错控制的面向连接协议。

序号

协议使用序号和确认号来防止重复分组。一个字段被加入分组头部来保存那个分组的序号。一件需要着重考虑的事情就是序号的范围。由于想使分组大小最小化，所以我们寻找能提供无歧义通信的最小的序号范围。让我们来讨论一下所需要的序号范围。假设我们使用 x 作为序号；我们只需要在之后使用 $x+1$，不需要 $x+2$。为了表示这种情况，假设发送端已经发送了带有序号 x 的分组。可能发生三件事：

1. 分组安全完整地到达接收端；接收方发送一个确认。确认到达发送端，使发送端发送下一个序号为 $x+1$ 的分组。

2. 分组被破坏或未到达接收端；发送方在超时后重新发送分组（序号 x）。接收方返回一个确认。

3. 分组安全完整到达接收端；接收方发送一个确认，但是确认被破坏或丢失了。发送方在超时后重传分组（序号 x）。注意，这里分组是重复的。接收方可以认出这个事实，因为它等待分组 $x+1$，但是收到了分组 x。

我们可以看到，由于接收方需要区分情况 1 和 3，因此需要序号 x 和 $x+1$。但是不需要一个编号为 $x+2$ 的分组。在情况 1 中，分组可以再次被编号 x，由于分组 x 和 $x+1$ 被确认，两端都不会产生歧义。在情况 2 和 3 中，新的分组是 $x+1$ 而不是 $x+2$。如果仅仅需要 x 和 $x+1$，我们可以令 $x=0$ 且 $x+1=1$。这意味着序号是 0、1、0、1、0，等等。这称为模 2 运算。

确认号

由于序号必须适合于数据分组和确认，因此我们使用这种惯例：确认号总声明接收方预期接收的下一个分组（next packet expected）序号。例如，如果 0 号分组已经安全完整到达，接收方发送一个确认号为 1 的 ACK（意味着 1 号分组是预期接收的下一个分组）。如果 1 号分组已经安全完整

到达，接收方发送一个确认号为0的ACK（意味着0号分组是预期接收的下一个分组）。

在停止-等待协议中，确认号总是以模2运算的方式声明预期接收的下一个分组序号。

发送方有一个控制变量，我们称之为S（sender），它指向发送窗口中唯一的一个槽。接收方有一个控制变量，我们称之为R（receiver），它指向接收窗口中唯一的一个槽。

有限状态机

图3-21给出了停止-等待协议的有限状态机。由于这个协议是面向连接的，因此在交换数据分组之前，两端都处于已建立状态。事实上，这些状态嵌套在已建立状态之中。

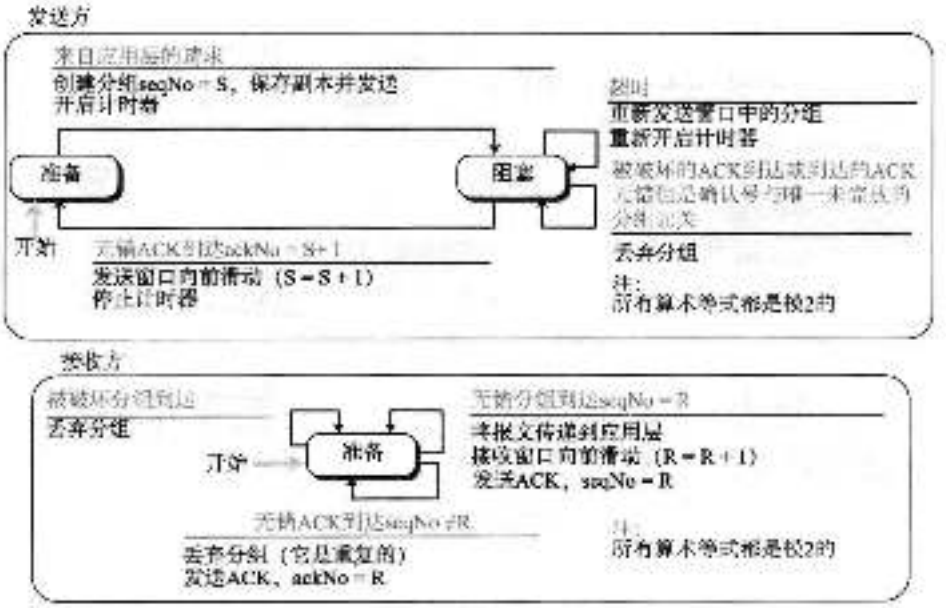


图3-21 停止-等待协议有限状态机

发送方

初始时，发送方处于准备状态，当时它可以在准备状态和阻塞状态之间转换，变量S初始化为0。

- 准备状态。当发送方处于这种状态时，它只等待一个事件发生。如果来自应用层的请求到来，发送方创建一个分组，并将其序号设为S。保存分组的副本，发送分组。之后，发送方开启唯一的计时器。发送方进入阻塞状态。
- 阻塞状态。当发送方处于这个状态时，可能发生三个事件：

a. 如果无错ACK到达，它的确认号与下一个待发送分组相关，这意味着ackNo = (S + 1)modulo 2，然后关闭计时器，窗口滑动S = (S + 1) modulo 2。最终发送方进入准备状态。

b. 如果到达的是被破坏ACK或是ackNo≠(S + 1) modulo 2的无错ACK，那么ACK被丢弃。

c. 如果发生超时，发送方重新发送唯一的未完成分组并重新开启计时器。

接收方

接收方总是处于准备（ready）状态。可能发生三个事件：

a. 如果seqNo = R的无错分组到达，分组中的报文被传递到应用层。之后窗口滑动，R = (R + 1) modulo 2。最终，确认号为ackNo = R的ACK被发送。

b. 如果seqNo≠R的无错分组到达，分组被丢弃，但是确认号为ackNo = R的ACK被发送。

c. 如果一个被破坏的分组到达，分组被丢弃。

例 3.4 图 3-22 给出了停止-等待协议的例子。分组 0 被发送且被确认。分组 1 丢失并在超时后重发。重发分组 1 被确认并且计时器停止。分组 0 被发送且被确认，但是确认丢失了。发送方不知道是分组丢失了还是确认丢失了，因此在超时之后，它重发分组 0，这次被确认了。

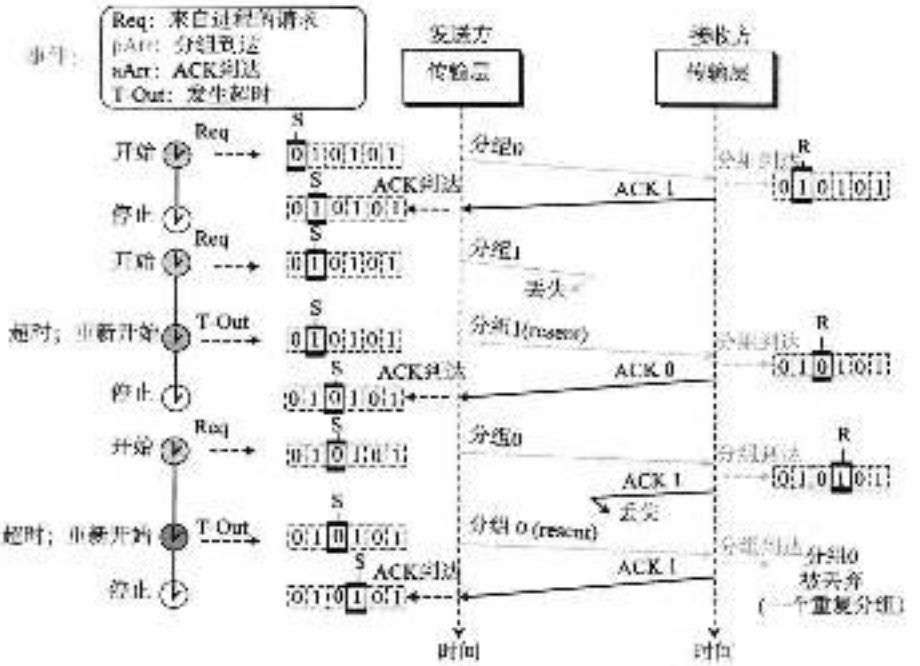


图 3-22 例 3.4 的流程图

效率

如果我们的信道又粗（thick）又长（long），那么停止-等待协议将非常低效。对于粗信道，我们的意思是信道有很大的带宽（高数据速率）；对于长信道，我们的意思是往返时间很长。这两者的乘积称为带宽延迟乘积（bandwidth-delay product）。我们可以把信道看做一条管道。带宽延迟乘积是以比特位为单位的管道容量。管道就在那里。如果不使用它，那么它就无效率。带宽延迟乘积可用来度量在等待来自接收方的确认信息时，发送方能够发送的位数。

例 3.5 假设在停止-等待系统中，线路的带宽是 1Mbps，1 比特需要花 20 毫秒完成往返。带宽延迟乘积是多少？如果系统数据分组长度是 1000 位，链路的利用率是多少？

解答

带宽延迟乘积是 $(1 \times 10^6) \times (20 \times 10^{-3}) = 20\,000$ 位。在数据从发送方到接收方以及确认返回的这段时间内，系统可以发送 20 000 位。然而，系统只发送了 1000 位。我们可以说链路利用率是 1000/20 000，或 5%。因此，在高带宽或大延时的链路中，停止-等待协议会浪费链路容量。

例 3.6 如果我们使用一个协议，在这个协议停止并把心确认之前，可以发送多达 15 个分组，那么例 3.5 中的链路利用百分率是多少？

解答

带宽延迟乘积仍是 20 000 位。系统在往返时间内可以发送多达 15 个分组或 15 000 位。这意味着利用率是 15 000/20 000 或 75%。当然，如果存在损坏的分组，利用率更小，这是因为分组需要重发。

流水线

在网络和其他领域中，在之前的任务结束前经常开始一个新任务。这称为流水线（pipeline）。

在停止-等待协议中没有流水线，因为发送端必须等待分组到达目的地，并且在下一个分组发送前接收到确认。然而，流水线用于我们下面两个协议。因为在发送方接收到关于前一个分组的反馈之前，就可以发送许多分组。如果与带宽延迟乘积相关联处于传输状态的比特位较多，那么流水线方式会提高传输的效率。

3.2.3 回退 N 帧协议

为了提高传输效率（充满管道），当发送端等待确认时，必须传输多个分组。换言之，当发送端等待确认时，我们需要让不止一个分组处于未完成状态，以此确保信道忙碌。在这一节中，我们讨论一个可以实现这个目标的协议；在下一节中，我们讨论第二个协议。第一个协议称为回退 N 帧协议（Go-Back-N，GBN，这个名字的道理之后就会明白）。回退 N 帧的关键是我们在接收到确认之前，可以发送多个分组。但是接收端只能缓冲一个分组。我们保存被发送分组的副本直到确认到达。图 3-23 给出了这个协议的框架。注意，很多数据分组以及确认可以同时处于信道中。

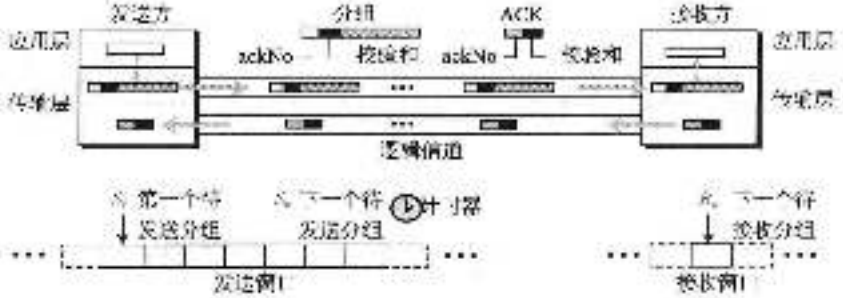


图 3-23 回退 N 帧协议

序号

如前所述，序号是模 2^m 的，这里 m 是序号字段的大小，单位是比特（位）。

确认号

这个协议中的确认号是累积的，并且定义了预期接收的下一个分组序号。例如，如果确认号（ackNo）是 7，这意味着序号在 6 以内的分组都已经安全完整到达，并且接收方等待序号为 7 的分组。

在回退 N 帧协议中，确认号是累积的并且定义了预期接收的下一个分组序号。

发送窗口

发送窗口是一个想象的盒子，它覆盖了处于运送途中的以及可以被发送的数据分组序号。在每个窗口位置，某些序号定义了已经被发送的分组；其他序号定义了可以被发送的分组。窗口最大为 2^m-1，我们之后讨论这其中的原因。在本章，我们令窗口大小固定为最大值，但是我们之后会看到，有些协议的窗口大小可能可以变化。图 3-24 给出回退 N 帧协议中大小为 7 的滑动窗口（$m=3$）。



图 3-24 回退 N 帧发送窗口

在任何时候，发送窗口都可能将序号分成四部分。第一部分，窗口左侧，定义了已经确认的分组的序号。发送方不需要担心这些分组并且不需要保存它们的副本。第二部分，带灰度部分，定义了已经被发送的分组的序号，但是这些分组状态未知。发送方需要等待，从而发现这些分组究竟是已经被

接收还是丢失。我们把这些分组称为未完成（outstanding）分组。第三部分，浅灰部分，定义了可以发送的分组的序号；然而，相应数据还没有从应用层接收到。最后，第四部分，窗口右侧，定义了直到窗口滑动前都不能使用的序号。

窗口本身是一种抽象：三个变量定义了它任何时候的大小和位置。我们将这些变量称为 S_f（发送窗口，第一个未完成分组）、S_n（发送窗口，下一个待发送分组）以及 S_{size}（发送窗口，大小）。变量 S_f 定义了第一个（最旧的）未完成分组序号。变量 S_n 存储的序号将要被分配给下一个待发送的分组。最终，变量 S_{size} 定义了窗口大小，窗口大小在我们的协议中是固定不变的。

发送窗口是一种抽象概念，它定义了一个最大为 $2^m - 1$ 的想象的盒子，其中有三个变量，它们是 S_f、S_n 以及 S_{size}。

图 3-25 给出了当从另一端接收到确认时，发送窗口是如何向右滑动一到多个槽的。在图中，ackNo = 6 的确认到达。这意味着接收方正在等待序号为 6 的分组。

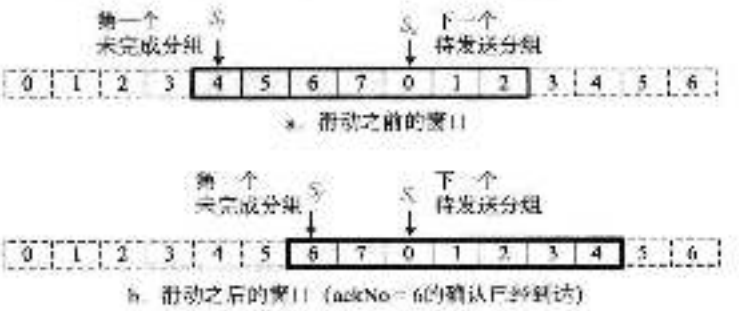


图 3-25 发送窗口的滑动

当 ackNo 大于等于 S_f 且小于 S_n（模运算）的无错 ACK 到达时，发送窗口可以滑动一个或多个槽。

接收窗口

接收窗口确保正确的数据分组被接收，并且确保正确的确认被发送。在回退 N 帧中，接收窗口的大小总是 1。接收方总是寻找特定分组是否到达。任何失序分组到达都会被丢弃并需要被重发。图 3-26 给出了接收窗口。注意，我们只需要一个变量，即 R_n（接收窗口，预期接收的下一个分组），来定义这种抽象窗口。窗口左侧的序号属于已经被接收和确认的分组；窗口右侧的序号定义了不能被接收的分组。任何序号在这两区域中的分组都被丢弃。只有序号符合 R_n 值的分组才能被接收和确认。接收窗口也滑动，但是一次只滑动一个槽。当正确的分组被接收时，窗口滑动 $R_n = (R_n + 1)$ modulo 2^m。

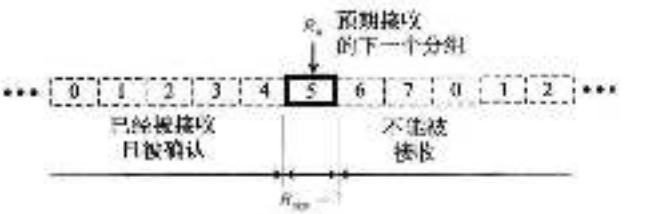


图 3-26 回退 N 帧接收窗口

接收窗口是一个抽象概念，它定义了一个最大为 1 的想象的盒子，其中只有一个变量 R_n。当正确分组到来时，窗口滑动；窗口每次只滑动一个槽。

计时器

尽管每个被发送分组都有计时器，但是在我们的协议中只使用一个计时器。原因是第一个未完成分组的计时器总是最先终止。当这个计时器终止时，我们重发所有未完成分组。

重发分组

当计时器终止时，发送方重发所有未完成分组。例如，假设发送方已经发送了分组 6（$S_s = 7$），但是唯一的计时器终止。如果 $S_f = 3$，这意味着分组 3、4、5 和 6 没有被确认；发送方回退并重发分组 3、4、5 和 6。这就是为什么称为回退 N 帧。一旦超时，机器回退 N 个位置并重发所有分组。

有限状态机

图 3-27 给出了 GBN 协议的有限状态机。

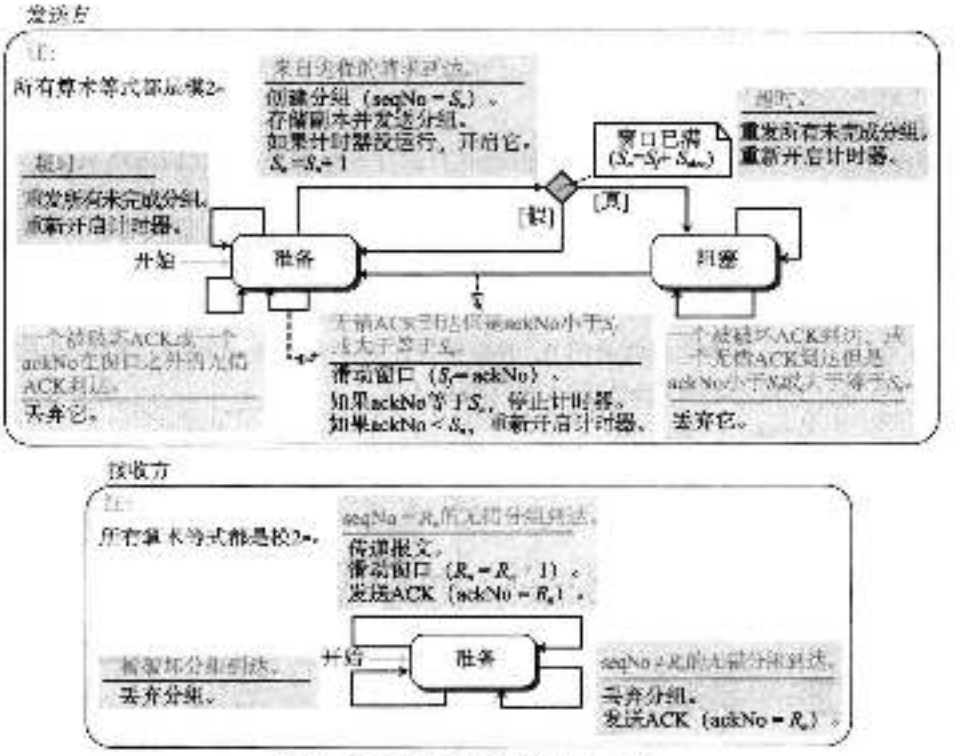


图 3-27 三退 N 帧协议有限状态机

发送方

发送方开始时处于准备状态，但是此后，它可能处于两种状态：准备（ready）或阻塞（blocking）。两个变量被初始化为 0（$S_f = S_s = 0$）。

- 准备状态。当发送方处于准备状态，可能发生四个事件。

a. 如果请求来自应用层，发送方创建一个序号为 S_s 的分组。存储分组的副本，发送分组。如果计时器没有运行，发送方会开启唯一的计时器。S_s 的值增长，（$S_s = S_s + 1$）modulo 2^m。如果窗口已满，$S_s = (S_f + S_{size})$ modulo 2^m，发送方进入阻塞状态。

b. 如果无差错 ACK 到达，其 ackNo 与一个未完成分组有关，那么发送方滑动窗口（令 S_f = ackNo），并且如果所有未完成分组都被确认（ackNo = S_s），那么计时器停止。如果并不是所有未完成分组都被确认，那么计时器重新开启。

c. 如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，它就被丢弃。

d. 如果超时发生，发送方重发所有未完成分组并重新开启计时器。

- 阻塞状态。在这种情况下可能发生三个事件：

a. 如果 ackNo 与一个未完成分组相关的无错 ACK 到达，那么发送方滑动窗口（令 S_f = ackNo），如果所有未完成分组被确认（ackNo = S_s），那么关闭计时器。如果所有未完成分组没有被确认，那

么重新开启计时器。之后，发送方进入准备状态。

b. 如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，那么 ACK 被丢弃。

c. 如果超时发生，发送方发送所有未完成分组并重新开启计时器。

接收方

接收方总是处于准备状态。唯一的变量 R_n 被初始化为 0。可能发生三个事件：

a. 如果 seqNo = R_n 的无错分组到达，分组中的报文被传递到应用层。之后窗口滑动，R_n = (R_n + 1) modulo 2^m。最终 ackNo = R_n 的 ACK 被发送。

b. 如果 seqNo 在窗口之外的无错分组到来，分组被丢弃，但是 ackNo = R_n 的 ACK 被发送。

c. 如果被破坏分组到达，将被丢弃。

发送窗口大小

我们现在可以给出为什么发送窗口大小必须小于 2^m 了。举例来说，我们选择 $m = 2$，这意味着窗口大小可能是 $2^m - 1$ 或 3。图 3-28 将大小为 3 的窗口与大小为 4 的窗口进行对比。如果窗口大小为 3（小于 2^m）并且所有三个确认都丢失，那么终止唯一的计时器并且重发所有三个分组。接收方现在期待分组 3，而不是分组 0。因此重复分组被正确丢弃。另一方面，如果窗口的大小是 4（等于 2^2）并且所有确认都丢失，发送方将会发送一个分组 0 的副本。然而，这次接收窗口期待接收的是分组 0（在下一轮），因此它接收分组 0，并不将其作为一个副本，但是作为下一轮的第一个分组。这是一个错误。这显示出发送窗口的大小必须小于 2^m。

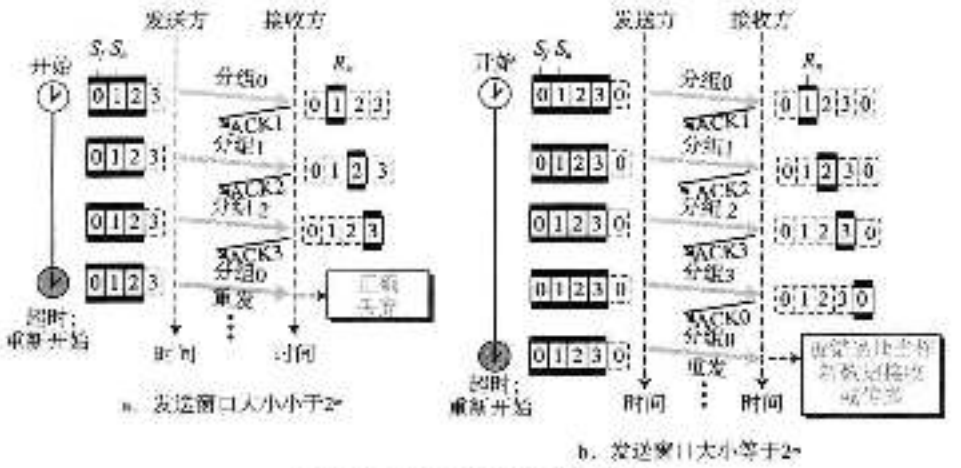


图 3-28 回退 N 帧发送窗口大小

在回退 N 帧协议中，发送窗口大小必须小于 2^m；接收窗口大小总是 1。

例 3.7 图 3-29 给出了回退 N 帧的例子。在这个例子中，给发信道是可靠的，但是反向是不可靠的。虽然没有数据分组丢失，但是一些 ACK 被延迟，还有一个 ACK 丢失。这个例子也给出了当确认延迟或丢失时，积累确认是如何起到帮助作用的。

在初始化之后，有一些发送方事件。请求事件被来自应用层的报文块触发；到达事件被来自网络层的 ACK 触发。这里没有超时事件，因为所有未完成分组在计时器超时之前都被确认。注意，尽管 ACK2 丢失，但是 ACK3 是积累的并且作为 ACK2 和 ACK3 进行服务。在接收端有四个事件。

例 3.8 图 3-30 给出当分组丢失时发生的事情。分组 0、1、2 和 3 被发送。然而，分组 1 丢失。接收方接收分组 2 和 3，但是由于它们是失序的（分组 1 是预期到达的分组），因此被丢弃。当接收方接收到分组 2 和 3 时，它发送 ACK1 来表示它预期接收分组 1。然而，这些 ACK 对发送方是无用的，因为 ackNo 等于 S_f，而不大于 S_f。因此发送方丢弃它们。当超时事件发生时，发送方重新

发送分组1、2和3，它们被确认。

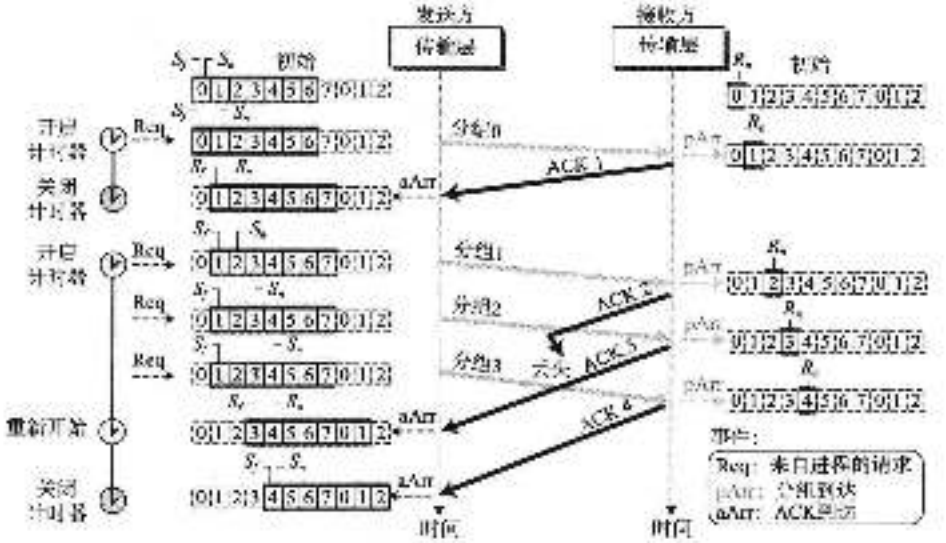


图 3-29 例 3.7 的流程图

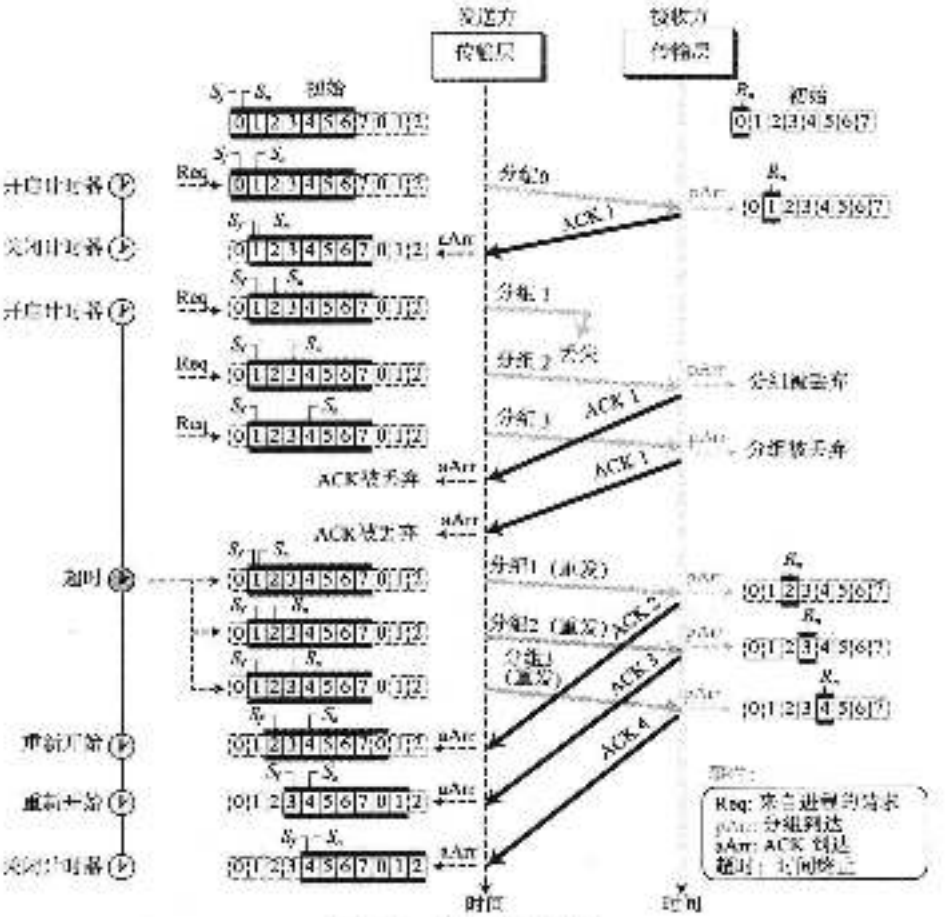


图 3-30 例 3.8 的流程图

回退 N 帧与停止等待

读者可能发现回退 N 帧协议与停止-等待协议有一些相似。停止-等待协议实际上是一种回退 N 帧协议，这种回退 N 帧协议只有两个序号且发送窗口大小为 1。换言之，$m=1$ 且 $2^m-1=1$。在回退 N 帧中，我们说计算是模 2^m 进行的；在停止-等待协议中是模 2，这与 $m=1$ 时的 2^m 相同。

3.2.4 选择性重复协议

回退 N 帧协议简化了接收方的进程。接收方只记录一个变量，没有必要缓冲失序分组；它们被简单地丢弃。然而，如果下层网络层丢失很多分组，那么这个协议是低效的。每当一个分组丢失或被破坏，发送方要重新发送所有未完成分组，即使有些失序分组已经被安全完整地接收了。如果网络层由于网络阻塞，丢失了很多分组，那么重发所有这些未完成分组将会使得阻塞更严重，最终更多的分组丢失。这具有雪崩效应，可能导致网络全部瘫痪。

另一个协议，称为**选择性重复协议**（Selective-Repeat（SR）protocol），已经被设计出来，正如其名字所示，只是选择性重发分组，即那些确实丢失的分组。这个协议的框架如图 3-31 所示。

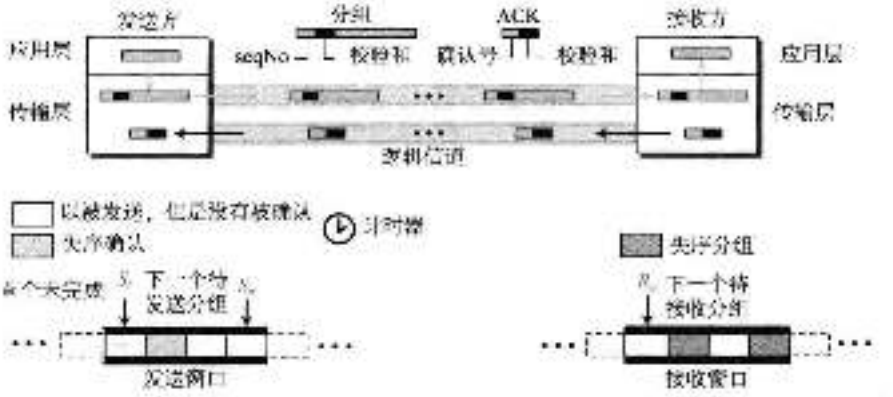


图 3-31 选择性重复框架

窗口

选择性重复协议也使用两个窗口：一个发送窗口和一个接收窗口。然而，这些窗口与回退 N 帧中的不同。首先，发送窗口的最大值更小；它是 2^{m-1}。这里的原因我们稍后讨论。第二，接收窗口和发送窗口大小一致。

发送窗口最大为 2^{m-1}。例如，如果 $m=4$，序号从 0 到 15，但是窗口最大值仅仅是 8（在回退 N 帧协议中是 15）。我们在图 3-32 中给出选择性重复发送窗口来再次强调窗口大小。

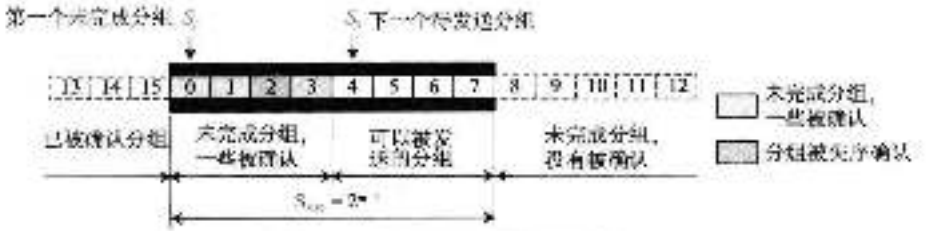


图 3-32 选择性重复协议发送窗口

选择性重复接收窗口与回退 N 帧中的接收窗口完全不同。接收窗口的大小和发送窗口等大（最大 2^{m-1}）。选择性重复协议允许和接收窗口一样多的分组失序到来并被存储，直到有一组连续分组被传递到应用层。因为发送窗口和接收窗口的大小是相同的，在发送窗口的所有分组可以失序到达

并被存储，直到它们可以被传递。然而，我们需要强调的是，在可靠协议中，接收方从不向应用层传递失序分组。图 3-33 给出选择性重复中的接收窗口。那些带阴影的窗口内的槽定义了失序到达的分组，并且在传输到应用层之前正在等待早先发送的分组。



图 3-33 选择性重复协议接收窗口

计时器

理论上讲，选择性重复为每个未完成分组使用一个计时器。当一个计时器终止，只有一个相应分组被发送。换言之，GBN（回退 N 帧）将未完成分组看做一个组；SR（选择性重复）将它们单独处理。然而，绝大多数实现了 SR 的传输层协议只使用一个计时器。出于这个原因，我们只使用一个计时器。

确认

这两个协议之间还有一点不同。在 GBN 中 ackNo 是累积的：它定义了下一个预期分组的序号，确认了之前的分组都安全完整到达。在 SR 中确认的语义是不同的。在 SR 中，ackNo 定义了被安全完整接收的一个分组；对其他分组没有反馈信息。

在选择性重复协议中，确认号定义了已被接收的无错分组的序号。

例 3.9 假设一个发送方发送 6 个分组：分组 0、1、2、3、4 和 5。发送方接收 ackNo = 3 的 ACK。如果系统使用 GBN 或 SR，那么这该如何解释呢？

解答

如果系统使用 GBN，这意味着分组 0、1 和 2 已经被无误接收，并且接收方正在期待分组 3。如果系统正在使用 SR，这意味着分组 3 已经被无误接收；ACK 并没有涉及其他分组的信息。

有限状态机

图 3-34 给出选择性重复协议的有限状态机。它与 GBN 类似，但有一些不同。

发送方

发送方开始时处于准备状态，但是此后，它可能处于两种状态：准备或阻塞。以下列出了事件以及每个状态的相关动作。

- **准备状态**。当发送方处于准备状态，可能发生四个事件：

a. 如果请求来自应用层，发送方创建一个序号为 S_n 的分组。分组的副本被存储，分组被发送。如果计时器没有在运行，发送方开启计时器。现在 S_n 值开始增长，（$S_n = S_n + 1$）modulo 2^m。如果窗口已满，$S_n = (S_f + S_{size})$ modulo 2^m，发送方进入阻塞状态。

b. 如果无差错 ACK 到达，其 ackNo 与一个未完成分组有关，分组被标记为未确认。如果 ackNo = S_f，那么窗口向右方滑动，直到 S_f 指向第一个未确认分组（所有连续的已被确认分组现在处于窗口之外）。如果存在未完成分组，重新开启计时器；否则，停止计时器。

c. 如果一个被破坏分组或 ackNo 与未完成分组无关的无错分组到达，它就被丢弃。

d. 如果超时发生，发送方重发所有未完成分组并重新开启计时器。

- **阻塞状态**。在这种情况下可能发生三个事件：

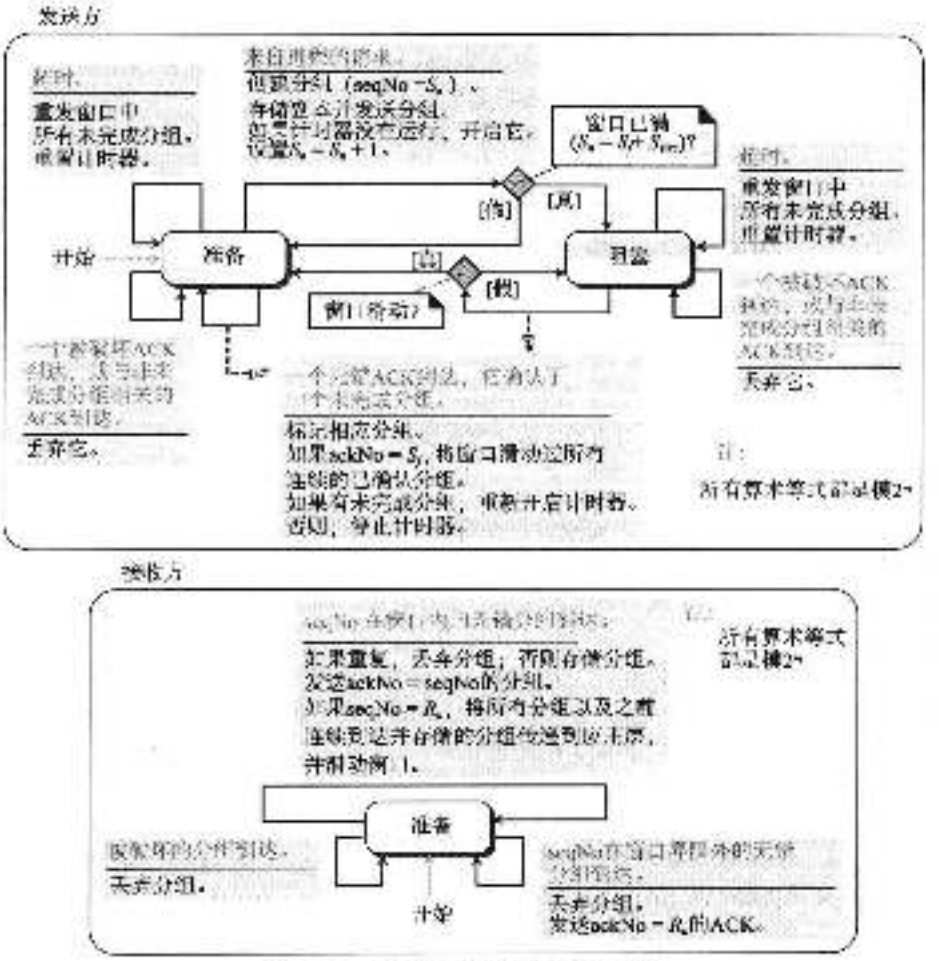


图 3-34 选择性重复协议有限状态机

a. 如果 ackNo 与一个未完成分组相关的无错 ACK 到达，那么这个分组被标记为已确认。此外，如果 ackNo = S_f，那么窗口向右滑动，直到 S_f 指向第一个未确认分组（所有连续的已被确认分组现在处于窗口之外）。如果窗口已经滑动，发送方进入准备状态。

b. 如果一个被破坏 ACK 或 ackNo 与未完成分组无关的无错 ACK 到达，那么 ACK 被丢弃。

c. 如果超时发生，发送方发送所有未完成分组并重新开启计时器。

接收方

接收方总是处于准备状态，可能发生二个事件：

a. 如果 seqNo 在窗口内的无错分组到达，分组被存储并且 ackNo = seqNo 的 ACK 被发送。此外，如果 seqNo = R_n，那么分组以及之前连续到达的分组被传递到应用层，并且窗口滑动使得 R_n 指向第一个空槽。

b. 如果 seqNo 在窗口之外的无错分组到达，分组被丢弃，但是 ackNo = R_n 的 ACK 被返回到发送方。如果那些与 seqNo < R_n 分组相关的 ACK 丢失，那么窗口需要滑动。

c. 如果被破坏分组到达，它被丢弃

例 3.10 这个例子与例 3.8（图 3-30）相似，其中分组 1 丢失。我们给出选择性重复协议在这种情况下如何工作。图 3-35 给出了具体场景。

在发送方，分组0被传输并且被确认。分组1丢失。分组2和3失序到达并被确认。当计时器超时，分组1（唯一未被确认的分组）被重发并被确认。之后，发送窗口滑动。

在接收端我们需要将分组接收和传送分组到应用层区分开。在第二次到达事件中，分组2到达了且被存储、被标记（阴影框），但是它不能被传递，因为分组1丢失了。在下一个到达事件中，分组3到达了且被标记、被存储，但是分组仍然不能被传递。只有在最后一次到达事件中，当最终分组1的副本到达了，分组1、2和3才能被传输到应用层。将分组传输到应用层有两个条件：第一，一组连续的分组必须已经到达。第二，这一组分组开始于窗口的起始端。在最后一次到达事件发生之后，有三个分组并且第一个分组开始于窗口起始端。关键是在于可靠传输层保证按序传输分组。

窗口大小

我们现在给出为什么发送窗口和接收窗口最多为2^m的一半。例如，我们选择$m=2$，这意味着窗口大小为$2^m/2$或$2^{m-1}=2$。图3-36将大小为2的窗口与大小为3的窗口进行比较。

如果窗口大小为2，并且所有确认丢失，那么分组0的计时器超时且分组0被重发。因此这个重复分组被正确地丢弃（序号0不在窗口内）。当窗口大小为3，并且所有确认丢失，那么发送方发送分组0的副本。然而，这次，接收方窗口期待接收分组0（0是窗口的一部分）。因此它接收了分组0，并且不把它看成一个重复分组，但是作为下一个循环中的分组。这明显是一个错误。

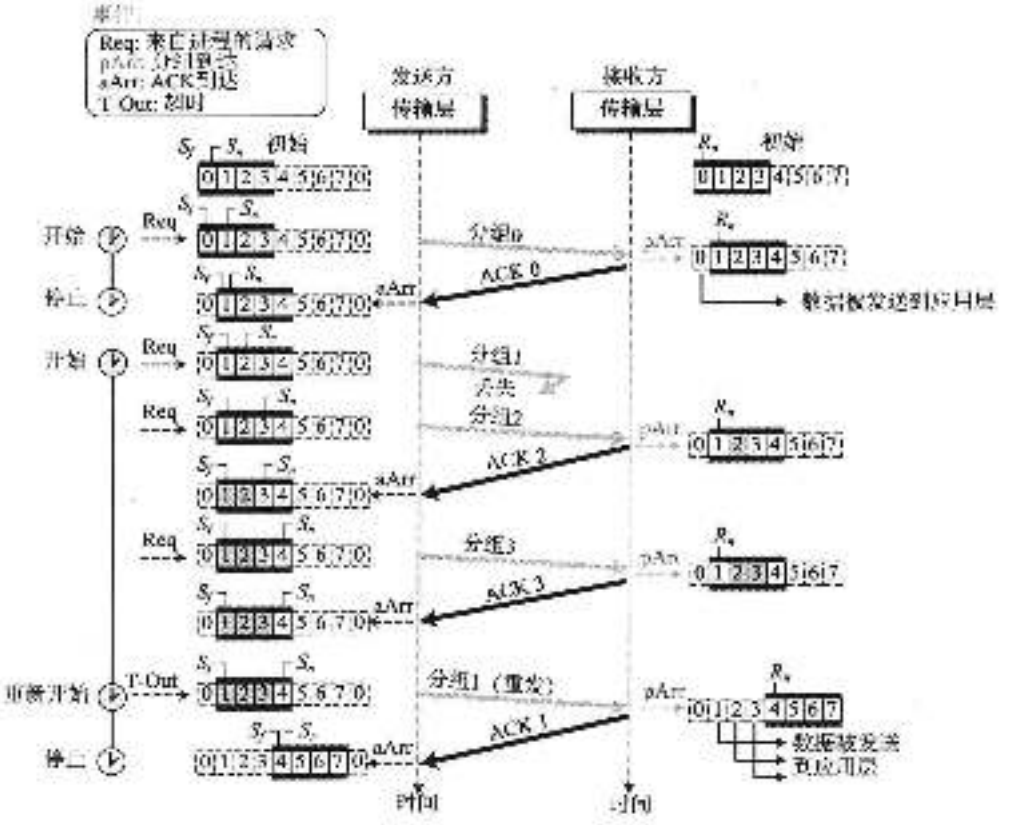


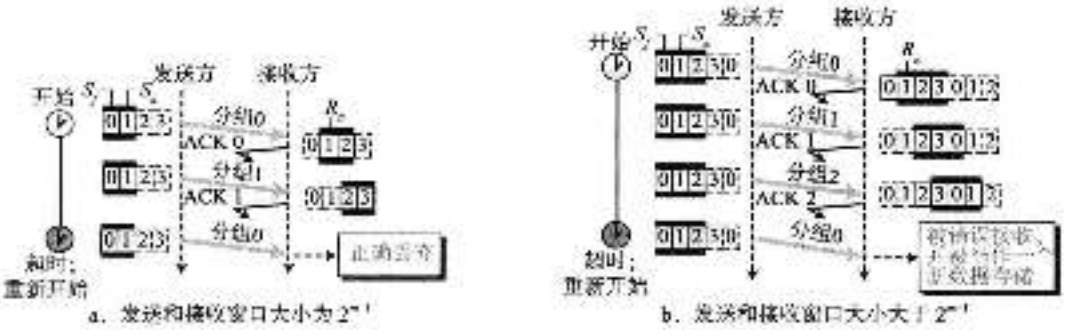
图3-35 例3.10流程图

在选择性重复中，发送方和接收方窗口的大小最多为$2^m/2$。

3.2.5 双向协议：捎带

我们在本节讨论的两个协议都是单向的：数据分组只沿着一个方向流动并且确认也是沿一个方

向传递的。在现实生活中，数据分组通常是双向流动的：从客户到服务器以及从服务器到客户。这意味着确认也需要沿着两个方向流动。一种称为捎带（piggybacking）的技术被用来提高双向协议的效率。当一个分组携带数据从 A 到 B 时，它也携带了确认反馈，这些信息确认了来自 B 的分组已到达；当一个分组携带数据从 B 到 A 时，它也携带了确认反馈，这些信息确认了来自 A 的分组已到达。



a. 发送和接收窗口大小为 2^1

b. 发送和接收窗口大小大于 2^1

图 3-36 选择性重复：窗口大小

图 3-37 给出了使用捎带实现了双向功能的 GBN 协议。客户和服务器各使用两个独立窗口：发送和接收。

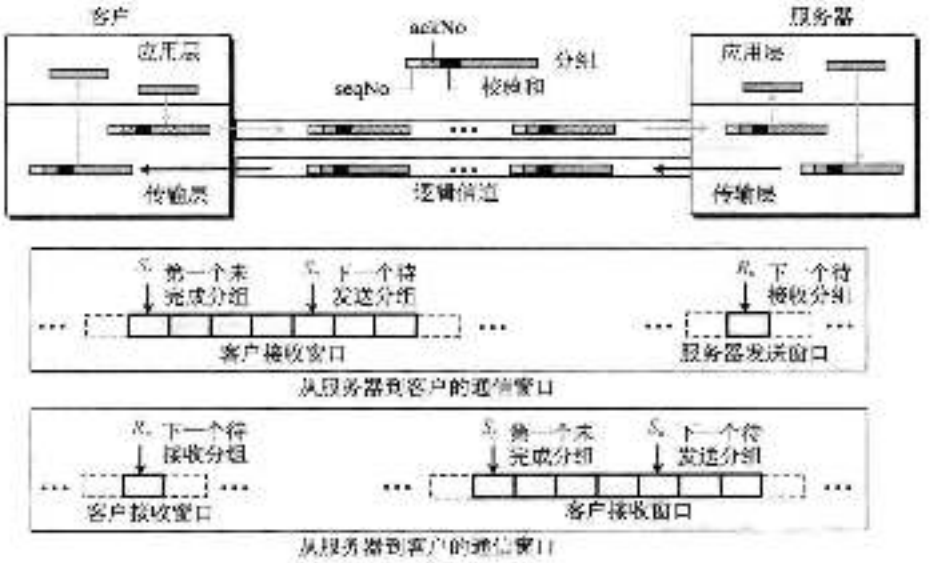


图 3-37 回退 N 帧中的捎带设计

3.2.6 因特网传输层协议

在讨论传输层的一般原则之后，我们在下面两节专注于因特网的传输层协议。尽管因特网使用很多传输层协议，但是我们在本章只讨论两个，如图 3-38 所示。

图 3-38 中给出了 UDP 和 TCP 这两个传输层协议与其他协议的关系，以及 TCP/IP 协议族的层次。这些协议位于应用层和网络层之间，是应用程序和网络操作的中间媒介。

UDP 是不可靠的无连接传输层协议，由于在应用中简单高效而被使用，在那些应用中差错控制由应用层进程提供。TCP 是可靠的面向连接协议，可用于可靠性重要的任何应用。也有其他应

用层协议比如 SCTP，我们会在其他章节讨论它们。

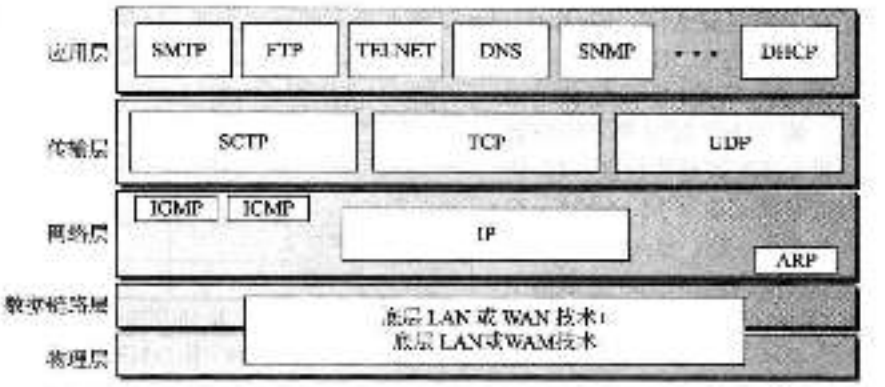


图 3-38 在 TCP/IP 协议族中传输层协议的位置

如前所述，传输层协议通常有很多责任。一个是创建进程到进程通信；这些协议使用端口号来完成这项责任（见表 3-1）。

表 3-1 UDP 和 TCP 使用的熟知端口

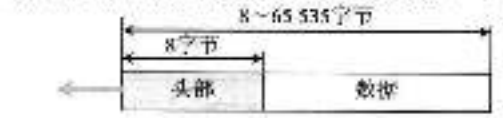
端 口	协 议	UDP	TCP	说 明
7	Echo	√		将接收到的数据报回送到发送方
9	Discard	√		丢弃接收到的任何数据报
11	Users	√	√	活跃的用户
13	Daytime	√	√	返回日期和时间
17	Quote	√	√	返回每日引用
19	Chargen	√	√	返回一个字符串
20,21	FTP		√	文件传输协议
23	TELNET		√	终端网络
25	SMTP		√	简单邮件传输协议
53	DNS	√	√	域名服务
67	DHCP	√	√	动态主机设置协议
69	TFTP	√		简单文件传输协议
80	HTTP		√	超文本传输协议
111	RPC	√	√	远程过程调用
123	NTP	√	√	网络时间协议
161,162	SNMP		√	简单网络管理协议

3.3 用户数据报协议

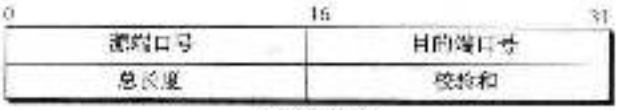
用户数据报协议（User Datagram Protocol，UDP）是无连接不可靠传输层协议。它不提供主机到主机通信，它除了提供进程到进程之间的通信之外，就没有给 IP 服务增加任何东西。此外，它进行非常有限的差错检验。如果 UDP 功能是如此之差，那么为什么进程还要使用它？它有缺点也有优点。UDP 是一个非常简单的协议，开销最小。如果一个进程想发送很短的报文，而且不在意可靠性，就可以使用 UDP。使用 UDP 发送一个很短的报文，在发送方和接收方之间的交互要比使用 TCP 时少得多。我们在本节最后讨论一些使用 UDP 的一些应用。

3.3.1 用户数据报

UDP 分组称为用户数据报（user datagram），有 8 字节的固定头部，这个头部由 4 个字段组成，每个字段 2 字节（16 位）。图 3-39 说明了用户数据报的格式。头两个字段定义了源和目的端口号。第三个字段定义了用户数据报的总长，即头部加数据的长度。16 位可以定义的总长度范围是 0 到 65 535。然而，总长度需要更小一些，这是因为 UDP 数据报存储在总长度为 65 535 的 IP 数据报中。最后一个字段可以携带可选校验和（稍后解释）。



a. UDP用户数据报



b. 头部格式

图 3-39 用户数据报格式

例 3.11 以下是十六进制格式的 UDP 头部内容。

CB840000D001C001C

a. 源端口号是多少？
b. 目的端口号是多少？
c. 用户数据报总长度是多少？
d. 数据长度是多少？
e. 分组是从客户发向服务器的还是相反方向的？
f. 客户进程是什么？

解答

a. 源端口号是头 4 位十六进制数字（CB84）$_{16}$，这意味着源端口号是 52100。
b. 目的端口号是第二组 4 位十六进制数字（000D）$_{16}$，这意味着目的端口号是 13。
c. 第三组 4 位十六进制数字（001C）$_{16}$定义了整个 UDP 分组的长度，长度为 28 字节。
d. 数据的长度是整个分组长度减去头部长度，即 28–8 = 20 字节。
e. 由于目的端口号是 13（熟知端口号），分组是从客户发送到服务器。
f. 客户进程是 Daytime（见表 3-1）。

3.3.2 UDP 服务

我们早先讨论过传输层协议提供的一般服务。在本节，我们讨论 UDP 提供的一般服务。

进程到进程的通信

UDP 使用套接字地址提供进程到进程通信，这是 IP 地址和端口号的组合。

无连接服务

如前所述，UDP 提供无连接服务。这就是表示 UDP 发送出去的每一个用户数据报都是一个独立的数据报。不同的用户数据报之间没有关系，即使它们都是来自相同的源进程并发送到相同的目的程序。用户数据报不进行编号。此外，也没有像 TCP 协议那样的连接建立和连接终止，这就表示每一个用户数据报可以沿着不同的路径传递。

无连接的一个结果就是使用 UDP 的进程不能够向 UDP 发送数据流，并期望它将这个数据流分割成许多不同的相关联的用户数据报。相反，每一个请求必须足够小，使其能够装入用户数据报中，只有那些发送短报文的进程才应当使用 UDP。短报文小于 65 507 字节（65 535 减去 UDP 头部的 8 字节再减去 IP 头部的 20 字节）。

流量控制

UDP 是一个非常简单的协议。它没有流量控制（flow control），因而也没有窗口机制。如果到

来的报文太多时，接收方可能会溢出。缺乏流量控制意味着如果需要的话，使用 UDP 的进程应该提供这个服务。

差错控制

除校验和外，UDP 也没有差错控制（error control）机制，这就表示发送方不知道报文是丢失还是重传。当接收方使用校验和检测出差错时，它就悄悄地将此用户数据报丢弃。缺乏差错控制意味着如果需要的话，使用 UDP 的进程应该提供这个服务。

校验和

我们在第 5 章讨论检验和，以及它的计算。UDP 校验和包含三部分：伪头部、UDP 头部和从应用层来的数据。伪头部（psedoheader）是 IP 分组的头部的一部分（第 4 章讨论），其中有些字段要填入 0，用户数据报分装在 IP 分组中（见图 3-40）。



图 3-40 用于校验和计算的伪头部

如果校验和不包括伪头部，用户数据报也可能是安全完整地到达。但是，如果 IP 头部受到损坏，那么它可能被提交到错误的主机。

增加协议字段可确保这个分组是属于 UDP，而不是属于其他传输层协议。我们在后面将会看到，如果一个进程既可用 UDP 又可用 TCP，则端口号可以是相同的。UDP 的协议字段值是 17。如果在传输过程中这个值改变了，在接收端计算校验和时就可检测出来，UDP 就可丢弃这个分组。这样就不会传递给错误的协议。

可选校验和

UDP 分组的发送方可以选择不计算校验和。这种情况下，在发送前，校验和字段就全填入 0。在发送方决定计算校验和的情况下，如果偶巧结果全是 0，那么在发送前校验和全改为 1。换言之，发送方填充两次校验和。注意，这不会产生混淆，因为校验和的值在正常情况下不会全为 1（见例 3.12）。

例 3.12 在以下假想情况下校验和的数值是多少？

a. 发送方决定不包含校验和。

b. 发送方决定包含校验和，但是数值全为 1。

c. 发送方决定包含校验和，但是数值全为 0。

解答

a. 校验和字段全为 0 表示未计算校验和。

b. 当发送方填充校验和时，结果是全 0；发送方在发送前再次填充。数值为全 1。需要进行第二次填充操作来防止与 a 的情况混淆。

c. 这种情况不会再次发生，因为这时示了计算中涉及的每个项目的数值为 0。这是不可能的；伪头部中的某些字段具有非零值。

拥塞控制

由于 UDP 是无连接协议，它不提供拥塞控制。UDP 假设被发送的分组很小且零星，不会在网络中造成拥塞。今天当 UDP 被用做音频和视频的交互实时传输时，这个假设可能对也可能不对。

封装和解封装

要将报文从一个进程发送到另一个进程时，UDP 协议就要对报文进行封装和解封装。

排队

我们已经讨论过端口，但是没有讨论端口的实际实现。在 UDP 中，队列是与端口联系在一起的。

在客户端，当进程启动时，它从操作系统请求一个端口号。有些实现是创建一个入队列和一个出队列与每一个进程相关联。而有些实现只创建与每一个进程相关的入队列。

多路复用与多路分解

在运行 TCP/IP 协议簇的主机上只有一个 UDP，但可能有多个想使用 UDP 服务的进程。处理这种情况，UDP 采用多路复用和多路分解。

UDP 和通用简单协议比较

我们可以将 UDP 与之前讨论的无连接简单协议进行比较。唯一的区别就是 UDP 提供可选校验和来在接收端发现被破坏分组。如果校验和被加入分组，接收 UDP 可以检测分组，如果分组被破坏可以丢弃它。然而，没有反馈被发向发送方。

> UDP 是我们之前讨论的无连接简单协议的一个例子，区别在于它为差错检测加入了可选校验和。

3.3.3 UDP 应用

尽管 UDP 不满足我们之前讨论的可靠传输层协议标准，但是，UDP 更适合某些应用。原因是其他某些服务可能有副作用，这些副作用或许是不可接受的或许是不称心的。一位应用设计师有时需要折中来得到最佳情况。例如，在日常生活中，我们都知道一日递送包裹比三日递送要贵。尽管时间和代价在递送包裹中都是想要获取的特性，但是它们是彼此矛盾的。我们需要选择最佳值。

在这一节，我们首先讨论 UDP 的一些特性，这些特性是当某人设计应用程序时需要的，然后，我们给出一些典型应用。

UDP 特性

我们简要讨论 UDP 的一些特性以及它们的优势和劣势。

无连接服务

如前所述，UDP 是无连接协议。同一个应用程序发送的 UDP 分组之间是独立的。这个特性可以看做是优势也可以看做是劣势，这要取决于应用要求了。例如，如果一个客户应用需要向服务器发送一个短的请求并接收一个短的响应，那么这就是优势。如果请求和响应各自可以填充进一个数据报，那么无连接服务可能更可取。在这种情况下，建立和关闭连接的开销可能很可观。在面向连接服务中，要达到以上目标，至少需要在客户和服务器之间交换 9 个分组；在无连接服务中只需要交换 2 个分组。无连接服务提供了更小的延迟；面向连接服务造成了更多的延迟。如果延迟是应用的重要问题，那么无连接服务更可取。

例 3.13 一种客户-服务器应用如 DNS（见第 2 章），它使用 UDP 服务，因为客户需要向服务器发送一个短的请求，并从服务器接收快速响应。请求和响应可以填充进一个用户数据报。由于在每个方向上只交换一个报文，因此无连接特性不是问题；客户或服务器不担心报文会失序传送。

例 3.14 一种客户-服务器应用如 SMTP（见第 2 章），它在电子邮件中使用，它不使用 UDP 服务，因为用户可能发送较长的电子邮件报文，邮件可能包含多媒体（图片、音频或视频）。如果

应用使用 UDP 且报文不能填充进一个用户数据报，那么报文必须被应用分割成不同的数据报。在这里，无连接服务可能产生问题。用户数据报可能失序到达并被传送到接收方应用。接收方应用可能无法重拼这些片段。这意味着无连接服务对发送较长报文的应用来说有缺点。在 SMTP 中，当用户发送报文时，用户不期待很快收到响应（有时不需要响应）。这意味着面向连接服务中固有的额外延迟对 SMTP 来说不是至关重要的。

缺乏差错控制

UDP 不提供差错控制；它提供的是不可靠服务。绝大多数应用期待从传输层协议中得到可靠服务。尽管可靠服务是人们想要的，但是它可能有一些副作用。这些副作用对某些应用来说不可接受。当一个传输层提供可靠服务时，如果报文的一部分丢失或被破坏，它就需要被重传。这意味着接收方传输层不能向应用立即传送那一部分；在传向应用层的不同报文部分间会有不一致的延迟。对于某些应用天生就根本注意不到这些不一致的延迟，而对于有些应用这些延迟却是至关重要的。

例 3.15 假设我们正在从因特网下载一个非常大的文本文件。我们肯定需要使用提供可靠服务的传输层。当我们打开文件的时候，我们不想看到部分文件丢失或被破坏。每个报文之间的延迟对我们来说不是首要担心的事情；我们在整个文件构建好之前一直等待，然后查看它。在这种情况下，UDP 不是一个合适的传输层协议。

例 3.16 假设我们正在使用一个实时交互应用，例如 Skype。音频和视频被分割成帧并且一个接一个地发送。如果传输层应该重传某些被破坏或丢失的帧，那么整个传输的同步性就会丢失。观众会突然看到空白屏幕并且需要等待，直到第二个传输到达。这是不可容忍的。然而，如果屏幕的每个小部分都使用一个用户数据报传送，那么接收 UDP 可以轻易地忽略被破坏或丢失的分组，并将其余分组传送到应用程序。屏幕的那部分会空白很短一段时间，而绝大多数观众都不会注意到。

缺乏拥塞控制

UDP 不提供拥塞控制。然而，在倾向于出错的网络中 UDP 没有创建额外的通信量。TCP 可能多次重发一个分组，因此这个行为促使拥塞发生或者使得拥塞状况加重。因此，在某些情况下，当拥塞是一个大问题时，UDP 中缺乏差错控制可以看做是一个优势。

典型应用

下面给出了一些典型应用，与 TCP 服务相比，它们从 UDP 服务中的获益更多。

- UDP 适合于这样的进程：它需要简单的请求-响应通信，而较少考虑流量控制和差错控制。对于需要传送成块数据的进程（如 FTP）则通常不使用 UDP（见第 2 章）。
- UDP 适用于具有内部流量控制和差错控制机制的进程。例如，简单文件传输协议（TFTP）的进程就包含流量控制和差错控制。它可很容易地使用 UDP。
- 对多播来说，UDP 是一个合适的传输协议。多播能力已嵌入到 UDP 软件中，但没有嵌入到 TCP 软件中。
- UDP 可用于管理进程，如 SNMP（见第 9 章）。
- UDP 可用于某些路由选择更新协议，如路由选择信息协议（RIP）（见第 4 章）。
- UDP 通常用于交互实时应用，这些应用不能忍受接收报文之间的不一致延迟（见第 8 章）。

3.4 传输控制协议

传输控制协议（Transmission Control Protocol，TCP）是一个面向连接可靠的协议。TCP 显式定义了连接建立、数据传输以及连接拆除阶段来提供面向连接服务。TCP 使用 GBN 和 SR 协议的组合来提供可靠性。为了实现这个目的，TCP 使用校验和（为差错发现）、丢失或被破坏分组重传、累积和选择确认以及计时器。在这一节，我们首先讨论 TCP 提供的服务；之后我们详细讨论 TCP 的特性。TCP 是因特网中最常见的传输层协议。

3.4.1 TCP服务

在详细讨论TCP之前，让我们解释TCP向应用层提供的服务。

进程到进程的通信

像UDP一样，TCP通过使用端口号来提供进程到进程通信。在表3-1中给出了TCP使用的一些端口号。

流传递服务

与UDP不同，TCP是一个面向流的协议。在UDP中，进程发送一些具有预先规定边界的报文给UDP进行传递。UDP将它自己的头部添加到这些报文中并传递到IP层进行传输。来自进程的每一个报文称为一个用户数据报，最后变成一个IP数据报。IP和UDP都不认识这些数据之间的关系。

另一方面，TCP允许发送进程以字节流形式传递数据，并且接收进程也以字节流形式接收数据。TCP建立一种环境，在这种环境中，两个进程好像由一个假想的"管道"连接，这个管道通过因特网传送这些进程的数据。这种假想的环境如图3-41所示。发送进程产生（写入）字节流，而接收进程消费（读出）这些字节流。



图3-41 字节流传递

发送和接收缓冲区

因为发送和接收进程可能以不同的速度写入和读出数据，所以TCP需要用于存储的缓冲区。每一个方向都存在一个缓冲区：发送缓冲区和接收缓冲区。稍后我们会看到，这些缓冲区也用于TCP流量和差错控制机制。实现缓冲区的一种方法是使用以一字节为存储单元的循环数组，如图3-42所示。为了简化，我们只画出了两个缓冲区，每个缓冲区20个字节。通常情况下，缓冲区是数百甚至数千个字节，这取决于实现方法。这里给出的缓冲区是大小相同的，实际上并非总是如此。

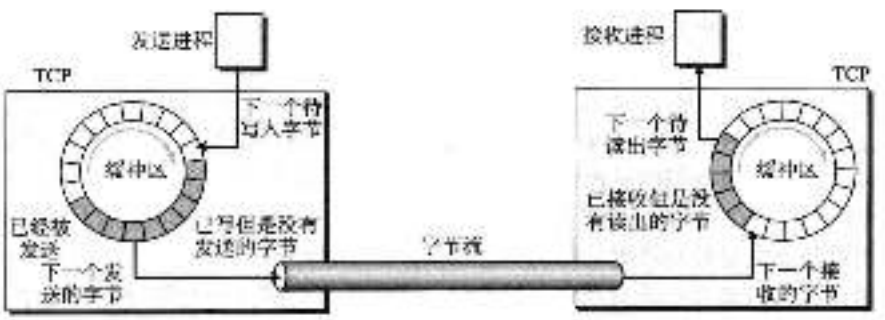


图3-42 发送与接收缓冲区

图3-42表示了在一个方向上数据的移动。在发送端，缓冲区有三种类型的存储单元。白色的部分是空存储单元，可以由发送进程（生产者）填充。灰色的部分用于保存已经发送但还没有得到确认的字节。TCP在缓冲区中保留这些字节，直到收到确认为止。灰色缓冲区是将要由TCP发送的字节。但是，在本章的后面将会看到，TCP可能只发送灰色部分。这可能是由于接收进程缓慢或者网络中可能发生的拥塞造成的。还要注意，灰色存储单元的字节被确认后，这些存储单元可以回收并且对发送进程可用，这就是我们给出一个环形缓冲区的原因。

接收端的缓冲区操作比较简单。环形缓冲区分成两个区域（表示为白色和灰色）。白色区域包含空存储单元，可以由从网络上接收的字节进行填充。灰色区域表示接收到的字节，可以由接收进程读出。当某个字节被接收进程读出以后，这个存储单元可被回收，并加入到空存储单元池中。

效

尽管缓冲能够处理生产进程速度和消费进程速度之间的不相称问题，但在发送数据之前，还需要多个步骤。IP 层作为 TCP 服务的提供者，需要以分组的方式而不是字节流的方式发送数据。在传输层，TCP 将多个字节组合在一起成为一个分组，这个分组称为段（segment）。TCP 给每个段添加头部（为了达到控制目的），并将该段传递给 IP 层。段被封装到 IP 数据报中，然后再进行传输。整个操作对接收进程是透明的。稍后，我们会看到这些段可能被无序接收、丢失，或者损坏和重发。所有这些均由 TCP 处理，接收进程不会察觉到任何操作。图 3-43 表示了在缓冲区中如何从字节生成段。

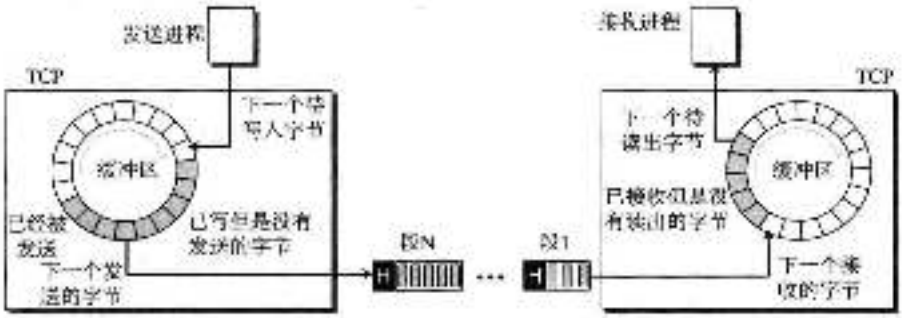


图 3-43 TCP 段

注意，段的大小不必相同。为了简单起见，我们在图中只表示了一个包含 3 个字节的段和另一个包含 5 个字节的段。实际的段可能包含数百（或者数千）个字节。

全双工通信

TCP 提供全双工服务（full-duplex service），即数据可以在同一时间双向流动。每一方向 TCP 都有发送和接收缓冲区，它们能在双向发送和接收段。

多路复用和多路分解

与 UDP 类似，TCP 在发送端执行多路复用，在接收端执行多路分解。然而，由于 TCP 是一个面向连接协议，因此需要为每对进程建立连接。

面向连接的服务

与 UDP 不同，TCP 是一种面向连接的协议。位于站点 A 的一个进程与站点 B 的另外一个进程想要进行数据的发送和接收，步骤如下：

1. 在两个 TCP 之间建立一个连接。
2. 在两个方向交换数据。
3. 连接终止。

注意，这是一个逻辑连接，而不是一个物理连接。TCP 段封装成 IP 数据报，并且可能被无序地发送，或丢失，或被破坏，然后重发。每个段都可以通过不同的路径到达目的端。TCP 建立一种面向字节流的环境，在这种环境中，TCP 能负担按顺序传递这些字节到其他站点的任务。

可靠的服务

TCP 是一种可靠的传输协议。它使用确认机制来检查数据是否安全和完整地到达。我们将在本章的后面进一步讨论差错控制的特点。

3.4.2 TCP 的特点

为了提供上一节所提及的服务，TCP 需要一些特性，本节将针对这些特性进行简要概述并在稍后进行详细讨论。

序号系统

虽然 TCP 软件能够记录发送或接收的段，但是在段的头部没有段序号字段。TCP 在段的头部采用称为序号（sequence number）和确认号（acknowledgment number）的两个字段。这两个字段指的是字节序号，而不是段序号。

字节序号

TCP 为在一个连接中传输的所有数据字节（八位字节）编号。在每个方向上序号都是独立的。当 TCP 接收来自进程的一些数据字节时，TCP 将它们存储在发送缓冲区中并给它们编号。不必从 0 开始编码。TCP 在 0 到 $2^{32}-1$ 之间生成一个随机数作为第一个字节的序号，例如，如果随机数是 1057，并且发送的全部字节个数是 6000，那么这些字节序号是 1057~7056。下面将会看到字节序号是用于流量和差错控制。

在每个连接中传送的字节都由 TCP 编号，序号开始于一个随机产生的数。

序号

字节被编号后，TCP 对发送的每一个段分配一个序号。在每一个方向上的序号定义如下：

1. 第一段的序号是初始序号（initial sequence number，ISN）。这是一个随机数。

2. 其他段的序号是之前段的序号加之前段携带的字节数（实际上的或想象的）。之后，我们将给出一些控制段，它们被认为携带了一个想象字节。

例 3.17 假设一个 TCP 连接正在传送一个 5 000 字节的文件，第一个字节序号是 10 001。如果数据被分成 5 个段，每一个数据段携带 1 000 字节，试问每个段的序号是什么？

解答

每个段的序号如下所示：

段 1 →	序号：10 001	范围：	10 001	到 11 000
段 2 →	序号：11 001	范围：	11 001	到 12 000
段 3 →	序号：12 001	范围：	12 001	到 13 000
段 4 →	序号：13 001	范围：	13 001	到 14 000
段 5 →	序号：14 001	范围：	14 001	到 15 000

一个段的序号字段的值定义了该段包含的第一个字节的序号。

当一个段携带数据和控制信息（捎带）时，它使用一个序号。如果一个段没有携带用户数据，那么它逻辑上不定义序号。虽然字段存在，但是值是无效的。然而，当有些段仅携带控制信息时也需要有一个序号用于接收方的确认。这些段用作连接建立、连接终止或连接废弃。这些段中的每一个好像携带一个字节那样使用一个序号，但都没有实际的数据。当我们讨论连接的时候，我们将自己研究这个问题。

确认号

正如我们前面所讨论过的那样，TCP 中的通信是全双工的；当建立一个连接时，双方同时都能发送和接收数据。每一方为字节编号，每一方经常使用不同的起始字节号。每一方向的序号表明了该段所携带的第一个字节的序号。每一方也使用确认号来确认它已收到的字节。但是，确认号定义了该方预期接收的下一个字节的序号。另外，确认号是累积的，这意味着接收方记下它已安全而且完整地接收到最后一个字节的序号，然后将它加 1，并将这个结果作为确认号进行通告。在这里，术语“累积”指的是，如果一方使用 5 643 作为确认号，则表示它已经接收了所有从开始到序号为 5 642 的字节。但要注意，这并不是指接收方已经接收了 5 642 个字节，因为第一个字节的编号通常并不是从 0 开始的。

段中确认字段的值定义了通信一方预期接收的下一个字节的编号。确认号是累积的。

3.4.3 段

在我们更详细讨论 TCP 之前，让我们讨论 TCP 分组本身。在 TCP 中的分组称为段（segment）。

格式

段的格式如图 3-44 所示。段包含 20 字节到 60 字节的头部，接着是来自应用程序的数据。如果没有选项，那么头部是 20 字节；如果有选项，最多是 60 字节。在本节中，我们将讨论某些头部字段，通过本章的学习可以更加清楚这些字段的意义和目的。



图 3-44 TCP 段格式

- **源端口地址**。这是一个 16 位的字段，它定义了在主机中发送该段的应用程序的端口号。这与 UDP 头部的源端口地址的作用一样。
- **目的端口地址**。这是一个 16 位的字段，它定义了在主机中接收该段的应用程序的端口号。这与 UDP 头部的目的端口地址的作用一样。
- **序号**。这个 32 位的字段定义了一个数，它分配给段中数据的第一个字节。如前所述，TCP 是一种字节流传输协议。为了确保连通性，对要发送的每一个字节都进行编号。序号告诉目的端，在这个序列中那一个字节是该段的第一个字节。在连接建立时，每一方都使用随机数生成器产生一个**初始序号**（initial sequence number，ISN），通常每一个方向的 ISN 都不同。
- **确认号**。这个 32 位的字段定义了段的接收方期望从对方接收的字节号。如果段的接收方成功地接收了对方发来的字节号 x，它就将确认号定义为 $x+1$，确认和数据可捎带一起发送。
- **头部长度**。这个 4 位的字段指明了 TCP 头部中共有多少个 4 字节长的字。头部的长度可以在 20 字节到 60 字节之间。因此，这个字段的值在 5（$5\times4=20$）到 15（$15\times4=60$）之间。
- **控制**。这个字段定义了 6 种不同的控制位或标记，如图 3-45 所示。在同一时间可以设置一位或多位。这些位用在 TCP 的流量控制、连接建立和终止、连接失败和数据传送方式等方面。图 3-45 给出了每个位的说明。本章中，当讨论 TCP 的具体操作时，我们将对它们做深入的讨论。

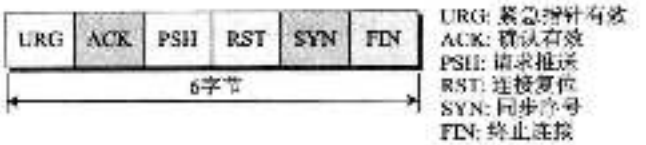


图 3-45 控制字段

- 窗口大小。这个字段定义对方必须维持的窗口的大小（以字节为单位）。注意，这个字段的长度是16位，这意味着窗口的最大长度是65 535字节。这个值通常称为接收窗口（rwnd），它由接收方确定。此时，发送方必须服从接收端的支配。
- 校验和。这个16位的字段包含了校验和。TCP校验和的计算过程与前面描述的UDP所采用的计算过程相同。但是，在UDP数据报中校验和是可选的。然而，对TCP来说，将校验和包含进去是强制的。起相同作用的伪头部被加到段上。对TCP伪头部，协议字段的值是6。如图3-46所示。

在TCP中使用校验和是强制的。

- 紧急指示符。这个16位的字段只有当紧急标志置位时才有效，这个段包含了紧急数据。它定义了一个数，将此数加到序号上就得出此段数据部分中最后一个紧急字节。在本章稍后将会讨论它。
- 选项。在TCP头部中可以有多达40个字节的可选信息。我们将在本节稍后讨论TCP头部中使用的某些选项。



图3-46 加到TCP数据报上的伪头部

封装

一个TCP段封装了来自应用层的数据。TCP段被封装在IP数据报中，IP数据报被封装在数据链路层的帧中。

3.4.4 TCP连接

TCP是一种面向连接的协议。面向连接的传输协议在源端和目的端之间建立一条虚路径。然后，属于一个报文的所有段都沿着这条虚路径发送。整个报文使用单一的虚路径有利于确认处理以及对损坏或丢失帧的重发。读者可能想知道TCP如何使用IP服务，一个无连接协议如何能面向连接。关键就在于TCP的连接是虚连接，不是物理连接。TCP在一个较高层次上操作，TCP使用IP服务向接收方传递独立的段，但它控制连接本身。如果一个段丢失了或损坏了，则重新发送它。与TCP不同，IP不知道这个重新发送过程。如果一个段失序到达，则TCP保存它直到缺少的段到达。IP是不知道这个重新排序过程的。

在TCP中，面向连接的传输需要三个过程：连接建立、数据传输和连接终止。

连接建立

TCP以全双工方式传输数据。当两个机器中的两个TCP建立连接后，它们就能够同时向对方发送段。这就表示，在传输数据之前，每一方都必须对通信进行初始化，并得到对方的认可。

三次握手

在TCP中的连接建立称为三次握手（three-way handshaking）。在我们的例子中，称为客户的应用程序想要与另一个称为服务器的应用程序使用TCP作为传输层协议建立连接。

该过程从服务器开始。服务器程序告诉它的TCP，它已准备好接收一个连接。这就称为被动打开（passive open）。虽然TCP已经准备好接收从世界上任何一个机器发来的连接，但它自己并不能完成这个连接。

客户程序发出请求进行主动打开（active open）。想要与服务器进行连接的客户告诉它的TCP，

它需要连接到特定的服务器。TCP 现在就开始如图 3-47 所示的三次握手过程。

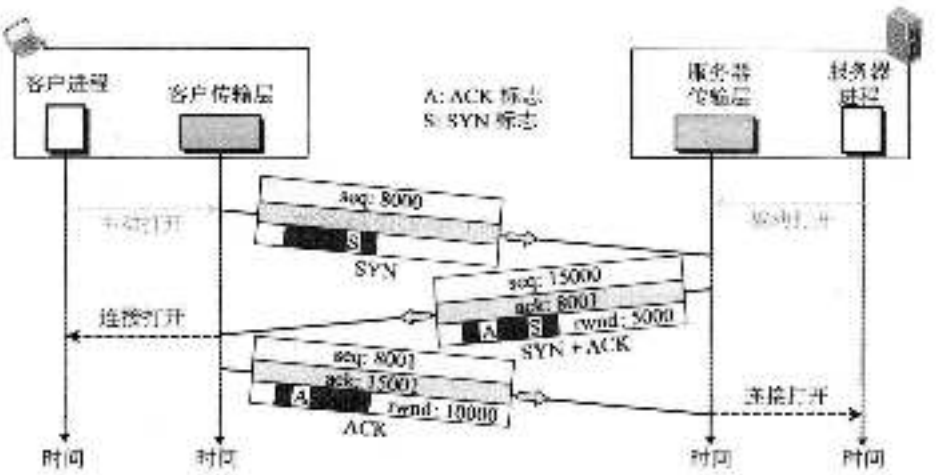


图 3-47 使用三次握手建立连接

为了表示该过程，我们使用了两个时间序列，每端一个。每个段头部的所有字段都有值，或许它的某些选项字段也有。但是，我们每次仅表示少数几个必须要知道的字段，如果序号、确认号、控制标记（仅仅是其中被置位的）和窗口大小等有值，那么我们就表示它们。这个阶段的三个步骤如下。

1. 客户发送的第一个段是 SYN 段。这个段仅有 SYN 标志被置位，它用于序号同步。它占用一个序号。当数据传输开始时，在我们的例子中，客户随机选择一个数字作为初始序号（ISN）。注意，这个段不包含确认号，它也没有定义窗口大小；窗口大小的定义只有当段包含确认号时才有意义。段也能包含一些我们本章稍后讨论的选项。注意，SYN 段是一个控制段并且不携带数据。然而，它消耗一个序号，因为它需要被确认。我们可以说 SYN 段携带了一个假想字节。

SYN 段不携带数据，但它占用一个序号。

2. 服务器发送第二个段，两个标志位 SYN 和 ACK 置位的段，即 SYN -ACK 段。这个段有两个目的。首先，它是另一方向通信的 SYN 段。服务器使用这个段来初始化序号，这个序号用来给从服务器发向客户的字节编号。服务器也通过给 ACK 置位并展示下一个序号来确认接收到来自客户的 SYN 段。这里的下一个序号是服务器预期从客户接收的序号。我们将在介绍流量控制那一节看到，因为它包含确认，它也需要定义接收窗口，即 rwnd（客户使用）。因为这个段起到 SYN 段的作用，它需要被确认。因此，它占用一个序号。

SYN - ACK 段不携带数据，但它占用一个序号。

3. 客户发送第三个段。这个段仅仅是一个 ACK 段。它使用 ACK 标志和确认序号字段来确认收到了第二个段。注意，如果不携带数据，ACK 段没有占用任何序号。但是一些实现允许这第三个段在连接阶段从客户端携带第一块数据。在这种情况下，段消耗的序号与数据字节数相同。

ACK 段，如果不携带数据，则它不占用序号。

SYN 泛洪攻击

在 TCP 中，连接建立过程易遭受到称为 SYN 泛洪攻击（SYN flooding attack）的严重安全问题。一个恶意的攻击者将大量的 SYN 段发送到一个服务器，在数据报中通过伪装源 IP 地址假装这些 SYN 段是来自不同的客户端，此时就是 SYN 泛洪攻击。假定客户机发出主动打开，服务器分配

必要的资源，如生成转换控制块（TCB）和设置计时器等。然后服务器发送 SYN-ACK 段给这些假客户，但这些段都丢失了。然而，当服务器等待第三段握手过程时，许多资源被占用但没有被使用。如果在短时间内，SYN 段的数量很大，服务器最终会耗尽资源而崩溃。这种 SYN 泛洪攻击属于一种称为拒绝服务攻击（denial of service attack）的安全攻击类型，其中，一个攻击者独占系统如此多的服务请求使得系统崩溃，拒绝对每个请求提供服务。

TCP 的某些实现拥有减轻 SYN 攻击影响的策略。有些实现在特定时间周期内对连接请求进行限制，另一些实现过滤掉来自不需要的源地址的数据报。一个新的策略使用 cookie 推迟资源分配直到一个完整的连接建立。新的传输层协议 SCTP 使用这种策略，我们将在第 8 章讨论。

数据传输

连接建立后，可进行双向数据传输，客户端与服务器双方都可发送数据和确认。在本章稍后，我们将学习确认的规则。目前，知道在同一段内携带确认时，在同一方向上也和可以传递数据就够了。这就是数据捎带确认。图 3-48 给出了一个例子。

在这个例子中，在连接建立后，客户端用两个段发送 2000 个字节的数据。然后，服务器用一个段发送 2 000 个字节的数据。客户端发送另一个段。前面三个段携带数据与确认，但是最后一个段仅携带确认，这是因为已没有数据发送了。注意序号与确认号数值，客户端发送的数据段有 PSH（推送）标志，所以服务器 TCP 知道在接收到数据时立刻传递给服务器进程。稍后我们讨论这个标志的用法。另一方面，来自服务器的段没有设置推送标志。大多数 TCP 的实现都有可选标志，可设置或不设置。

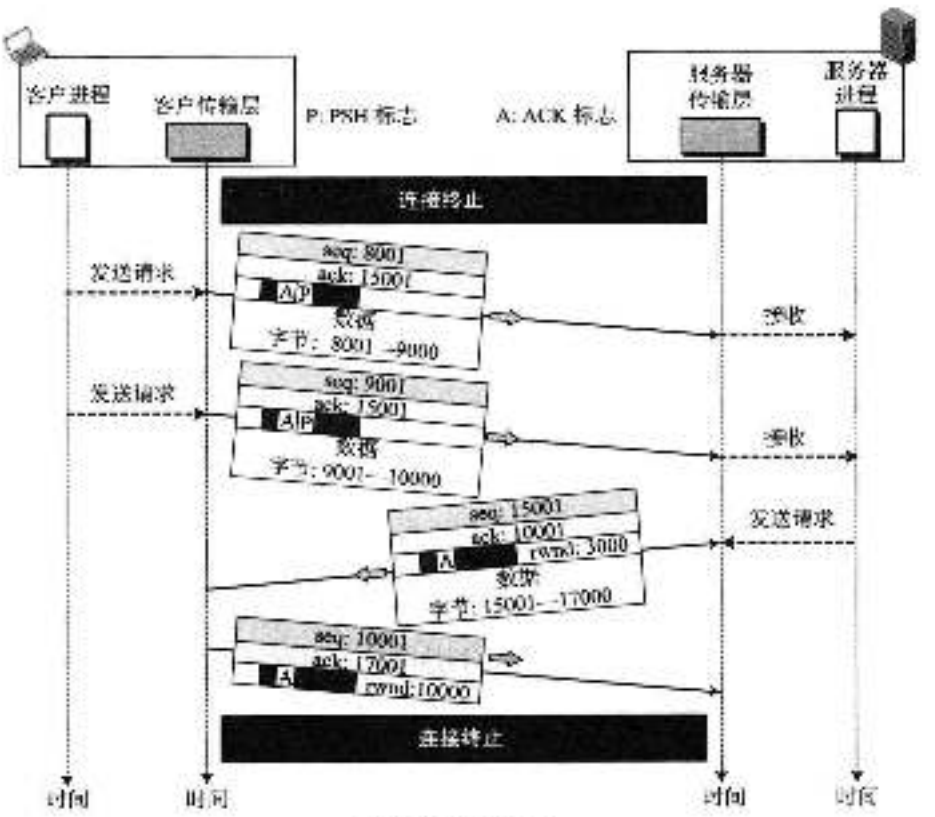


图 3-48 数据传输

推送数据

我们看到发送方的 TCP 使用缓冲区存储来自发送方应用程序的数据流。发送方的 TCP 可以选择段的大小。接收方的 TCP 在数据到达时也将数据进行缓存，并当应用程序准备就绪时或当接收端 TCP 认为方便时将这些数据传递给应用程序。这种灵活性增加了 TCP 的效率。

但是，在有些情况下，应用程序并不需要这种灵活性。例如，应用程序与另一方应用程序进行交互式通信。一方的应用程序打算将其击键发给对方应用程序，并希望接收到立即响应。数据的延迟传输和延迟传递对这个应用程序来说是不可接受的。

TCP 可以处理这种情况。在发送端的应用程序可请求推送操作。这就表示发送端的 TCP 不必等待窗口被填满。它创建一个段就立即将其发送。发送端的 TCP 还必须设置推送位（PSH）以告诉接收端的 TCP，这个段所包含的数据必须尽快地传递给接收应用程序，而不要等待更多数据的到来。这意味着将面向字节的 TCP 改为面向块的 TCP，但是 TCP 可以选择使用或不使用这个特性。

紧急数据

TCP 是面向字节流的协议。这就是说，从应用程序到 TCP 的数据被表示成一串字节流。数据的每一个字节在流中占有一个位置。但是，在某些情况下，应用程序需要发送紧急（urgent）字节，某些字节需要另一端的应用以特殊方式对待。解决方法是发送一个 URG 标志置位的段。发送应用程序告诉发送端的 TCP，这块数据是紧急的。发送端 TCP 创建段，并将紧急数据放在段的开始。段的其余部分可以包括来自缓冲区的普通数据。头部中的紧急指针字段定义了紧急数据的结束（紧急数据的最后一个字节）。例如，如果段序号是 15 000 且紧急指针的值是 200，那么紧急数据的第一字节是字节 15 000 且最后一个字节是 15 200。段中的剩余字节（如果存在的话）是非紧急的。

有一点需要提及，这很重要，那就是 TCP 的紧急数据不是人们所想的优先服务，也不是带外数据服务。相反，TCP 的紧急模式是一种服务，发送端的应用程序通过这种服务将字节流的某些部分标记为需要接收端特别对待的字节流。接收端 TCP 将字节（紧急或非紧急）按序传递到应用程序，但是通知应用程序紧急数据的开始和结束。留给应用程序决定如何处理紧急数据。

连接终止

交换数据双方的任一方（客户或服务器）都可关闭连接，尽管通常情况下是由客户端发起。当前大多数对连接终止的实现有两个方法：三次握手和带有半关闭选项的四次握手。

三次握手

当前对连接终止的绝大多数实现是三次握手（three-way handshaking），如图 3-49 所示。

1. 在正常情况下，在客户进程接收到一个关闭命令后，客户的 TCP 发送第一个段：FIN 段，即其中的 FIN 标志置位。注意，FIN 段可包含客户机要发送的最后数据块，或如图 3-49 所示的只是控制段。如果它只是控制段，它仅占有一个序号因为它需要被确认。

> 如果 FIN 段不携带数据，则该段占用一个序号。

2. 服务器 TCP 接收到 FIN 段后，通知它的进程，并发送第二个段：FIN－ACK 段，证实它接收到来自客户端的 FIN 段，同时通告另一端连接关闭。这个段还可以包含来自服务器的最后数据块。如果它不携带数据，则这个段仅占用一个序号。

> 如果 FIN + ACK 段没有携带数据，则该段仅占用一个序号。

3. 客户端的 TCP 发送最后一段，即 ACK 段，来证实它接收到来自服务器的 FIN 段。这个段包含确认号，它是来自服务器的 FIN 段的序号加 1。这个段不携带数据也不占用序号。

半关闭

在 TCP 中，一端可以停止发送数据后，还可以接续接收数据。这就是所谓的半关闭（half-close）。

虽然任一端都可发出半关闭，但通常都是由客户端发起的。当服务器在开始处理之前需要接收到所有数据，这时就会出现半关闭。例如，排序是一个很好的例子。客户端发送数据给服务器进行排序，在开始排序之前，服务器需要接收到全部数据。这就是说，客户端发送全部数据之后，它在客户到服务器方向可关闭连接，但为了返回存储数据，服务器到客户方向必须保持打开。服务器在接收到数据后还需要时间进行排序；它的向外方向必须保持打开。图 3-50 给出了半关闭的例子。

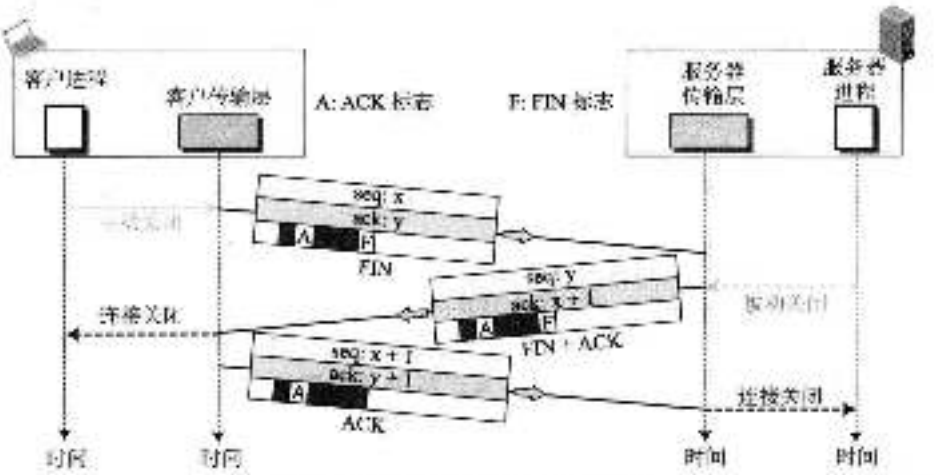


图 3-49 使用三次握手的连接终止

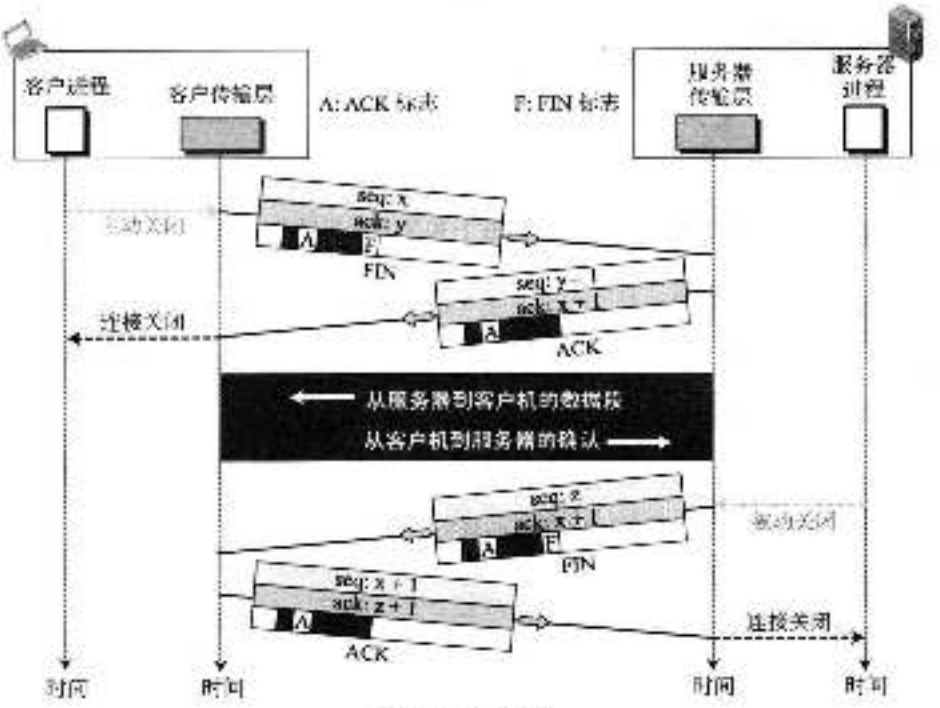


图 3-50 半关闭

从客户到服务器的数据传输停止。客户端通过发送 FIN 段实现半关闭连接。服务器通过发送 ACK 段确认半关闭。然而，服务器还可以发送数据。当服务器已经发送完被处理的数据时，它发

送一个 FIN 段。该 FIN 段由客户端的 ACK 来确认。

连接半关闭后，数据可以从服务器传送给客户端，而确认可以从客户端传送给服务器。客户不能再向服务器发送任何数据。

连接重置

在一端的 TCP 可能拒绝连接请求，可能终止已存在的连接，也可能结束空闲连接。所有这些都通过 RST（重置）标志完成。

3.4.5 状态转换图

为了记录发生在连接建立，连接终止和数据传输阶段发生的事件，如图 3-51 所示，TCP 以有限状态机的形式进行详述。

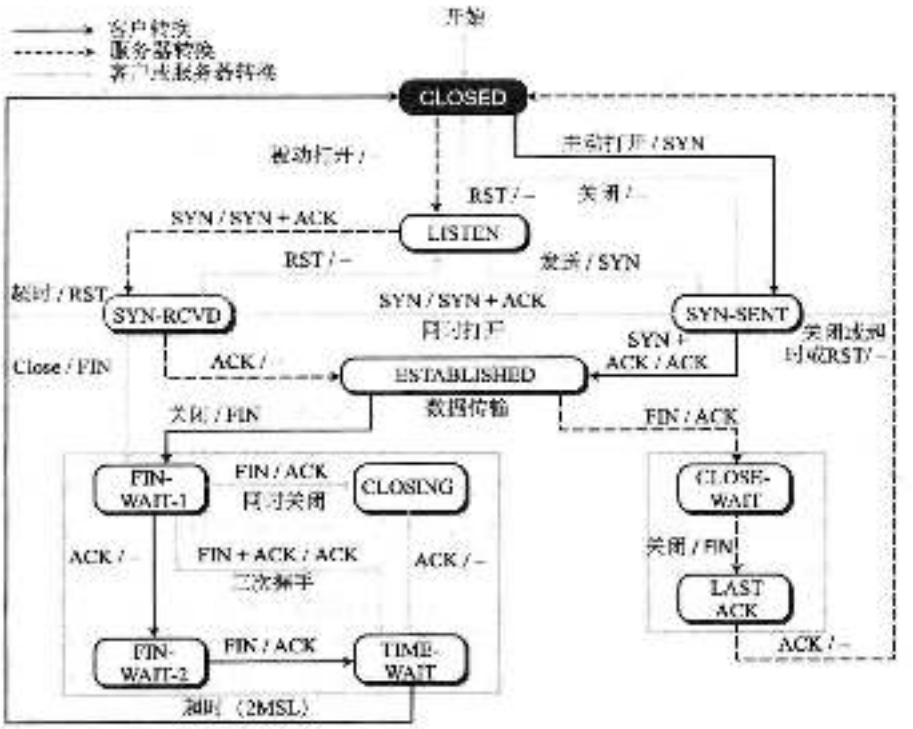


图 3-51 状态转换图

图 3-51 给出了 TCP 客户和服务器使用的两个有限状态机，TCP 客户和服务器被结合到一幅图中。圆角矩形代表状态。从一个状态到另一个状态的转换使用直线表示。每条线有两个字符串，它们用斜线分隔。第一个字符串是输入，即 TCP 所接收的内容。第二个字符串是输出，即 TCP 所发送的内容。图中虚线代表服务器通常经历的转换；黑色实线给出客户通常经历的转换。然而，在某种情况下，服务器沿着实线进行状态转换，客户沿着虚线进行状态转换。灰色线给出特殊情况。注意，被标记为 ESTABLISHED 的圆角矩形实际上是两种状态，一个是客户状态，另一个是服务器状态，它们用于流量和差错控制。本章之后会对其进行解释。我们将会在本章末尾讨论图中提及的计时器，包含 2MSL（2 Maximum Segment Lifetime，2 倍段最大生存时间）计时器。我们使用基于图 3-51 的多个场景并在每个情况中展示图 3-51 的一部分。

在有限状态机中被标记为 ESTABLISHED 状态实际上是两个不同状态，这些状态是客户和服务器进行传输数据经历的。

表 3-2 给出的 TCP 的状态表。

场景

为了理解 TCP 状态机以及转换图，我们在本节考察一种场景。

表 3-2 TCP 状态

状 态	说 明	状 态	说 明
CLOSED	没有连接存在	**FIN-WAIT-2**	首个 FIN 的 ACK 已被接收；等待第二个 FIN
LISTEN	接收到被动打开；等待 SYN	**CLOSE-WAIT**	首个 FIN 被接收，ACK 被发送；等待应用关闭
SYN-SENT	SYN 已被发送；等待 ACK	**TIME-WAIT**	第二个 FIN 被接收，ACK 被发送；等待 2MSL 超时
SYN-RCVD	SYN + ACK 已被发送；等待 ACK	**LAST-ACK**	第二个 FIN 被发送；等待 ACK
ESTABLISHED	连接建立；数据传输正在进行	**CLOSING**	双方决定同时关闭
FIN-WAIT-1	首个 FIN 已被发送；等待 ACK		

一个半关闭场景

图 3-52 给出了这个场景的状态转换图。

客户进程向它的 TCP 发出主动打开命令来请求连接到特定套接字地址。TCP 发送一个 SYN 段并转移到 **SYN-SENT** 状态。在收到 SYN + ACK 段后，TCP 发送了一个 ACK 段并且进入 **ESTABLISHED** 状态。数据被传输，可能是双向的，并且被确认。当客户进程没有数据要发送了，它发出称为主动关闭的命令。TCP 发送 FIN 段并进入 **FIN-WAIT-1** 状态。当它接收到 ACK 段，它进入 **FIN-WAIT-2** 状态。当客户接收到 FIN 段时，它发送一个 ACK 段并进入 **TIME-WAIT** 状态。客户保持这种状态 2MSL 秒（见本章结尾的 TCP 计时器）。当相应计时器超时，客户进入 **CLOSED** 状态。

服务器进程发出被动打开命令。服务器 TCP 进入 **LISTEN** 状态并且保持这种状态直到它接收到一个 SYN 段。TCP 之后发送一个 SYN + ACK 段并且进入 **SYN-RCVD** 状态，等待客户发送 ACK 段。在接收到 ACK 段后，TCP 进入 **ESTABLISHED** 状态，这就开始了数据传输。TCP 保持这种状态直到它接收到一个来自客户的 FIN 段，这表示没有其他数据要被交换且连接可以被关闭。一旦服务器接收到 FIN 段，那么它就向客户发送带有虚拟 EOF 标记的排队中所有的数据，这意味连接必须被关闭。它发送一个 ACK 段且进入 **CLOSE-WAIT** 状态，但是推迟确认来自客户的 FIN 段，直到它接收到来自进程的被动关闭命令。在接收到被动关闭命令后，服务器向客户发送 FIN 段并进入 **LAST-ACK** 状态，等待最终 ACK。当 ACK 段被从客户接收，服务器进入 **CLOSE** 状态。图 3-53 使用时间线上的状态给出相同的场景。

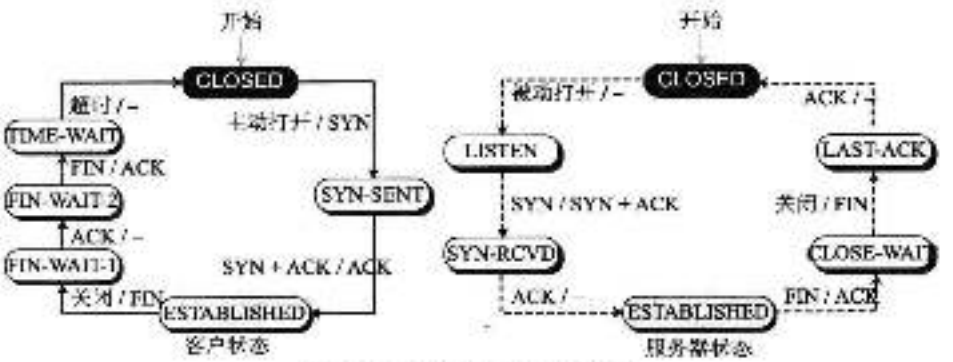


图 3-52 半关闭连接终止转换图

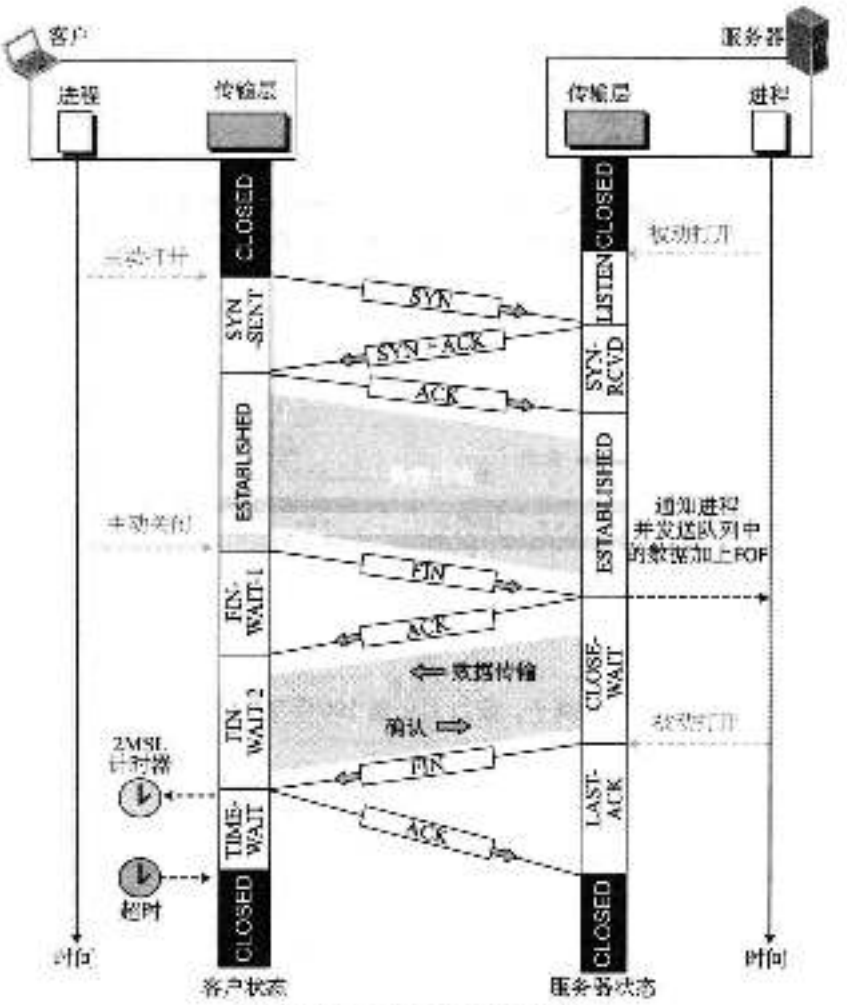


图 3-53 时间线图表示的普通场景

3.4.6 TCP 中的窗口

在讨论 TCP 中的数据传输并提出诸如流量、差错和拥塞控制问题之前，我们首先描述 TCP 中使用到的窗口。TCP 在每个方向的数据传输上使用两个窗口（发送窗口和接收窗口），这意味着双向通信有四个窗口。为了使得讨论简单，我们做一个不实际的假设，即通信只是单向的（比如从客户到服务器）；双向通信可以使用两个带有捎带的单向通信推理出来。

发送窗口

图 3-54 给出了一个发送窗口的例子。窗口大小是 100 字节，但是之后我们会看到发送窗口大小由接收方（流量控制）和底层网络的拥塞程度（拥塞控制）指定。图 3-54 给出了发送窗口如何打开、关闭以及收缩。

TCP 中的发送窗口与选择性重复协议中的发送窗口相似，但是有一些不同：

1. 一个区别是与窗口相关的实体本质不同。在 SR 中窗口的大小是分组的数量，但是 TCP 中的窗口大小是字节的数量。尽管 TCP 中实际传输是一段接一段发生的，但是控制窗口的变量是以字节为单位的。

2. 第二个区别是，在某些实现中，TCP 可以存储来自进程的数据并且在之后发送它们，但是我们假设发送方 TCP 一旦从进程中接收到数据就能够发送数据段。

3. 另一个区别是计时器的数量。理论上，选择性重复协议可能为每个被发送的分组使用多个计时器，而 TCP 只使用一个计时器。



图 3-54 TCP 中的发送窗口

接收窗口

图 3-55 给出了一个接收窗口的例子。窗口大小是 100 字节。图 3-55 也给出了接收窗口如何打开以及关闭；实际上，窗口从不收缩。

TCP 中的接收窗口与 SR 中的接收窗口有两点不同。

1. 第一个区别是 TCP 允许接收进程以自己的速率拉数据。这意味着接收方部分被分配缓冲区可以被已接收且确认的字节占据，但是它们正在等待被接收进程拉过去。如图 3-55 所示，接收窗口大小总是小于或等于缓冲区大小。接收窗口大小决定了接收窗口在被淹没（流量控制）之前可以从发送方接收的字节数量。换言之，接收窗口通常称为 rwnd，可以由下式决定：

$$\text{rwnd} = \text{缓冲区大小} - \text{等待被拉字节数量}$$

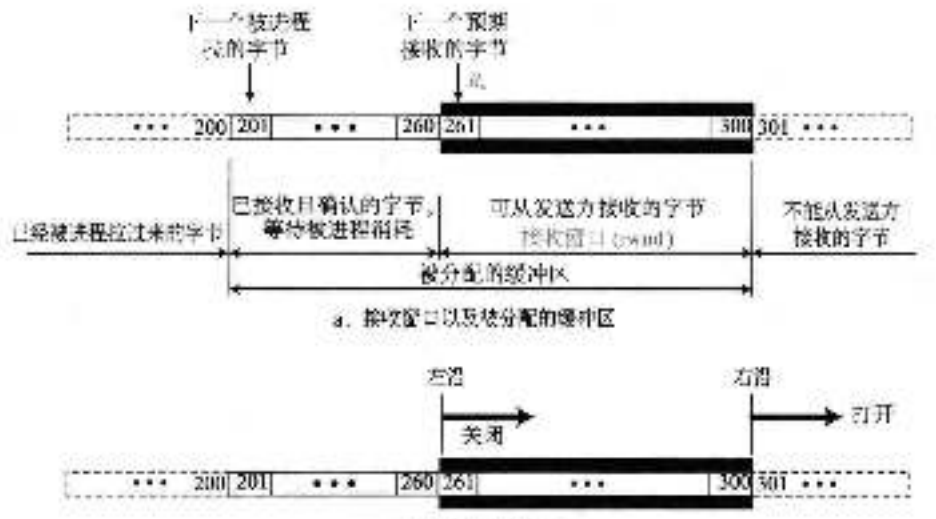


图 3-55 TCP 中的接收窗口

2. 第二个不同是在TCP协议中使用的确认方式不同。请记住，在SR中的确认是选择性的，它定义了已经被接收的分组。TCP中主要确认机制是累积确认，它声明了下一个预期接收字节（我们之前讨论过，按这种方式TCP看起来像GBN）。然而，TCP的新版本使用了累积确认和选择性确认；我们在本书的网站上讨论了这些选项。

3.4.7 流量控制

如之前所讨论的，流量控制平衡了生产者创建数据的速率与消费者使用数据的速率。TCP将流量控制与差错控制分开。在本节，我们讨论流量控制，忽略差错控制。我们假设发送和接收TCP之间的逻辑信道是无错的。

图3-56给出了发送方和接收方之间按的单向数据传输；双向数据传输可以从这个单向进程中推断出来。



图3-56 TCP中数据流和流量控制反馈

图3-56给出了从发送方进程到发送方TCP、从发送方TCP到接收方TCP以及从接收方TCP上升到接收方进程的数据（路径1、2和3）传输过程。然而，流量控制反馈却是从接收方TCP传输到发送方TCP并且从发送方TCP上升到发送方进程（路径4和5）。绝大多数TCP实现不提供从接收方进程到接收方TCP的流量控制反馈；无论何时，当接收方进程准备好了，具体实现就会让接收方进程从接收方TCP中拉数据。换言之，接收方TCP控制发送方TCP；发送方TCP控制发送方进程。

从发送方TCP到发送方进程（路径5）的流量控制反馈的实现方式是，当窗口已满，它就单拒绝发送方TCP的数据。这意味着，我们对于流量控制的讨论集中于由接收方TCP发向发送方TCP的反馈。

打开以及关闭窗口

为了实现流量控制，TCP通使发送方和接收方调整它们的窗口大小，尽管当连接建立时两方的缓冲区大小是固定的。当更多的数据从发送方到来时，接收方窗口关闭（向右移动左沿）；当更多的数据被进程拉过来时，它打开窗口（向右移动右沿）。我们假设它不会收缩（右沿不会向左移动）。

发送窗口的打开、关闭和收缩由接收方控制。当一个新的确认允许发送窗口关闭时，发送窗口关闭（向右移动左沿）。当接收方通知的接收窗口大小（rwnd）允许发送方窗口打开时（新ackNo + 新rwnd > 上一个ackNo + 上一个rwnd），发送窗口打开（向右移动右沿）。这种情况没有发生的事件中，发送窗口缩小。

一种场景

我们给出发送方和接收方窗口如何在连接建立阶段设置大小，并给出它们在数据传输阶段变化的情况。图3-57给出了一个简单的单向数据传输例子（从客户到服务器）。我们暂时忽略差错控制，假设没有段被破坏、丢失、重复或失序到达。注意，我们给出两个单向数据传输的窗口。尽管客户

在第二个段将服务器窗口设为 2000，但是我们不给出那个窗口，因为通信是单向的。

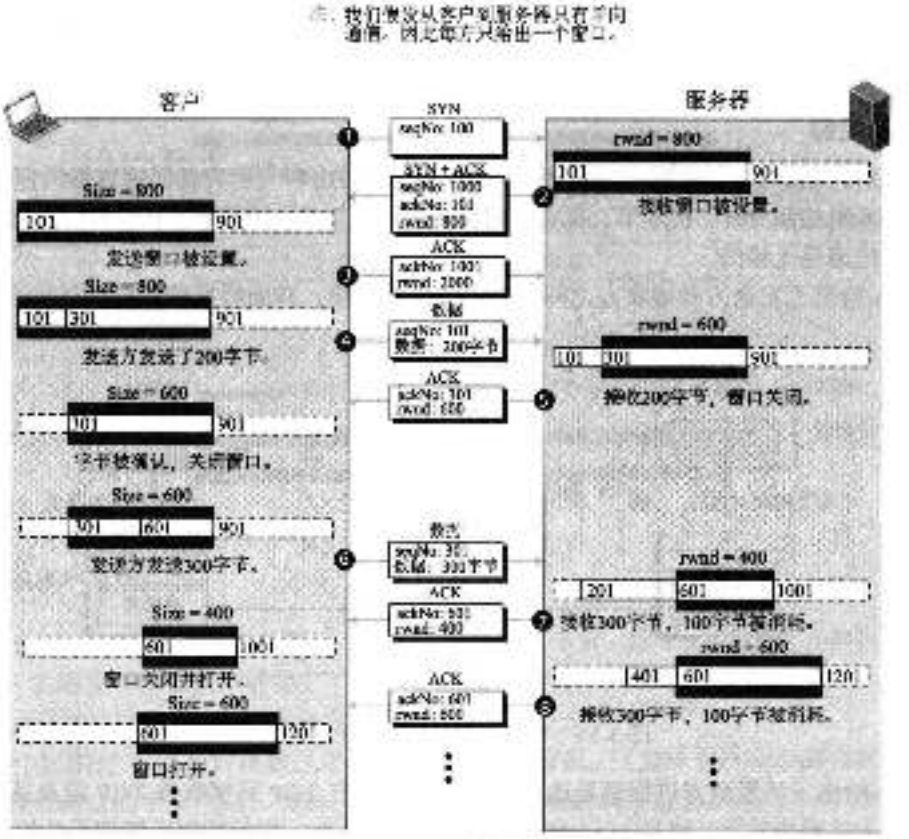


图 3-57 流量控制举例

客户和服务器之间交换了 8 个段。

1. 从客户到服务器的第 1 段（SYN 段）请求连接。这个段声明起始 seqNo = 100。当这个段到达服务器时，它分配了大小为 800 字节的缓冲区（假设）并设置窗口覆盖了全部缓冲区（rwnd = 800）。注意，下一个要到达的字节是 101。

2. 第 2 段是从服务器到客户的。这是 ACK + SYN 段。段使用 ackNo = 101，这表示它期待接收的字节从 101 开始。它也声明了客户可以设置大小为 800 字节的缓冲区。

3. 第 3 段是从客户到服务器的 ACK 段。注意，客户可以定义大小为 2000 的 rwnd，但是，在图 3-57 中我们不使用这个值，因为通信是单向的。

4. 在客户设置了服务器指定的窗口大小（800）之后，进程推送 200 字节数据。TCP 客户给这些字节编号为从 101 到 300。之后，它创建了一个段并发送到服务器。段开始字节数是 101 并且携带了 200 字节。之后客户窗口调整，表示 200 字节数据被发送但是等待确认。当这个段被服务器接收，这些字节被存储，并且接收窗口关闭来表示下一个预期字节是 301；存储字节占据了缓冲区的 200 字节。

5. 第 5 段是从服务器到客户的反馈。服务器确认了多达 300 个字节，含第 300 个字节（预期接收 301 字节）。这个段也携带了减少后的接收窗口大小（600）。在接收这个段之后，客户从它的窗口中清除确认字节并关闭它的窗口，这表示下一个待发送字节是 301。然而，窗口大小减少到 600 字节。尽管分配缓冲区可以存储 800 字节，但是窗口不能打开（向右移动右沿），因为接收方不允许。

6. 在进程又推送了300个字节后，第6段被客户发送。段定义了seqNo为301，并包含了300字节。当这个段到达服务器，服务器存储它们，但是它必须减少自己的窗口大小。在进程读100字节数据后，窗口左边关闭了300字节，但是右侧打开了100字节。结果是窗口大小只减少了200字节。现在接收窗口大小是400字节。

7. 在第7段，服务器确认接收数据，并声明它的窗口大小是400。当这个段到达客户，客户别无选择，只能再次减少它的窗口，并令窗口大小为服务器通告的rwnd = 400。发送窗口从左侧关闭300字节，并从右侧打开100字节。

8. 在进程读另外200字节后，第8段也是从来自服务器的。它的窗口大小增加。现在，新的rwnd的值是600。段告知客户，服务器仍然期待601字节，但是服务器窗口大小增加到600。我们需要提及的是，这个段的发送依赖于具体实现所规定的策略。一些实现可能不允许在这个时候通告rwnd；服务器需要在这样做之前接收一些数据。在这个段到达客户之后，客户将窗口打开200字节而不关闭。结果是窗口增大到600字节。

窗口收缩

如前所述，接收窗口不能收缩。另一方面，如果接收方为rwnd定义了导致窗口收缩的数值，那么发送窗口可以收缩。然而，一些实现不允许收缩发送窗口。这个限制不允许发送窗口的右沿向左移动。换言之，接收方需要保持上一个和新的确认之间以及上一个和新rwnd值之间的如下关系，以防止发送窗口收缩。

$$\text{新 ackNo} + \text{新 rwnd} \geq \text{上一个 ackNo} + \text{上一个 rwnd}$$

不等式左侧表示与序号空间相关的右沿位置；右边给出右沿的旧位置。这个关系表示右沿不能向左移动。不等式是对接收端的命令，它使接收端检查自己的通告。然而，注意，只有当$S_f < S_n$时不等式是有效的；我们需要记住所有计算都是模2^{32}。

例3.18 图3-58给出了这个命令的原因。

图3-58中a部分给出了上次确认和rwnd的值。b部分给出了一种情况，其中发送方已经发送了206到214的字节。206到209字节被确认并被清除。然而，新的通告定义rwnd的新值为4，其中，210 + 4 < 206 + 12。当发送窗口收缩时，它产生了一个问题，已经被发送的214字节处于窗口之外。我们之前讨论的关系迫使接收方保持窗口右边沿与a部分相同，因为接收方不知道210到217之间哪些字节已经被发送。防止这种情况的一种方法是，让接收方推迟反馈，直到在它的窗口中有足够的缓冲区位置。换言之，接收方必须等待，直到更多的字节被它的进程消耗，从而来满足上面描述的关系。

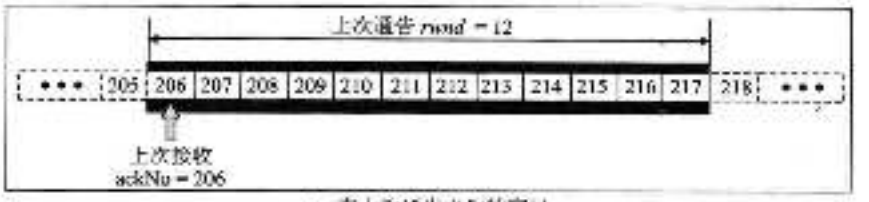
窗口关闭

我们看到，不鼓励将右沿向左移动来收缩发送窗口。然而，有一个例外：接收方可以通过发送rwnd为0来临时关闭窗口。这只会在某些原因下发生，即接收方在一段时间内不想接收来自发送方的任何数据。在这种情况下，发送方并不真的收缩窗口大小，但是它停止发送数据直到新的通告到达。我们将在后面看到，即使当窗口因为来自接收方的命令而关闭了，发送方也总可以发送一个1字节数据的数据段。这称为探测，用来防止死锁（见TCP计时器部分）。

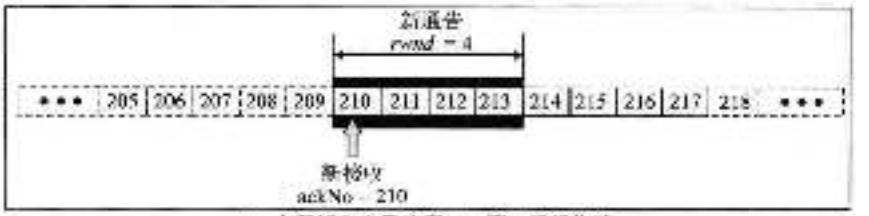
糊涂窗口综合征

当发送方应用程序缓慢创建数据时，或当接收方应用程序缓慢消耗数据时，或者两者同时发生时，在滑动窗口操作中可能发生一个严重的问题。任何这种情况都会导致以很小的段发送数据，这会降低操作的效率。例如，如果TCP发送一个只包含一字节数据的段，这意味着41字节数据报（20字节TCP头部以及20字节IP头部）仅仅传输了1字节用户数据。此时，开销是41/1，这表示我们非常低效地使用网络容量。在考虑到数据链路层和物理层开销后这种低效更为严重。这个问题称为糊涂窗口综合征（silly window syndrome）。对每一个站点，我们首先描述这个问题是如何产生的，

之后给出建议解决方法。



a. 在上次通告之后的窗口



b. 本新通告之后的窗口；窗口已经收缩

图 3.58 例 3.18

发送方产生的综合征

如果发送方 TCP 正在为一个创建数据很缓慢的应用程序服务，那么可能产生糊涂窗口综合征，例如，应用程序每次产生 1 字节数据。应用程序一次将 1 字节写入发送 TCP 的缓冲区。如果发送 TCP 没有任何特定指令，那么它可能创建包含 1 字节数据的段。结果是很多 41 字节的段在网络中传输。

解决方法是防止发送 TCP 一个字节一个字节地发送数据。发送 TCP 必须被迫等待并将数据收集成一个较大的块来发送。发送 TCP 要等待多长时间？如果等待太长，可能会延误进程。如果等待得不够长，可能结果就是发送小的段。Nagle 找到了一个巧妙的解决方法。**Nagle 算法**（Nagle's algorithm）非常简单：

1. 即使从发送方应用程序接收来的第一个数据片段只有 1 字节，发送 TCP 也发送它们。
2. 在发送第一个段之后，发送 TCP 在输出缓冲区积累数据并等待，直到接收 TCP 发送一个确认或直到积累了足够的数据来填充一个最大的段。此时，发送 TCP 可以发送段。
3. 在剩余的传输中重复第二步。如果接收到对第 2 段的确认或者积累了足够的数据填充一个最大段，就立即发送第 3 段。

Nagle 算法的巧妙之处在于它的简单，并且事实上，它考虑了应用程序创建数据的速率以及网络传输数据的速率。如果应用程序比网络快，段就更大（最大段）。如果应用程序比网络慢，段就更小（小于最大段大小）。

接收方产生的综合征

接收方 TCP 可能产生糊涂窗口综合征。如果它正在为一个消耗数据很缓慢的应用程序服务，例如，应用程序每次消耗 1 字节数据。假设发送应用程序以 1 千字节的块来创建数据，但是接收应用程序每次只消耗 1 字节数据。也假设接收 TCP 输入缓冲区是 4 千字节。发送方发送前 4 千字节数据。接收方将其存储在缓冲区中。现在缓冲区满了，它通告一个大小为 0 的窗口。这意味着发送方应该停止发送数据。接收方应用从接收方 TCP 输入缓冲区中读取第一个字节的数据。现在接收缓冲区有 1 字节空间。接收 TCP 声明一个大小为 1 字节的窗口，这意味着迫不及待要发送数据的发送方 TCP 把这个通告看做是一个好消息，并发送只携带 1 字节数据的段。这个流程将继续。1 字节数据被消耗并且携带 1 字节数据的段被发送。我们再一次遇到了效率问题以及糊涂窗口综合征。

此处提出两种解决方法来防止由消耗数据速率低于数据到达速率而产生的糊涂窗口综合征。第

一种方法是 Clark 解决方法（Clark's solution），即数据一到达就发送确认，但是声明一个大小为 0 的窗口，直到有足够的空间容纳最大段，或者至少半个接收缓冲区是空的。第二种方法是延迟发送确认。这意味着当段到达，它不立即被认。接收方延迟确认，直到在确认那些到达段之前接收缓冲区中有适当的空间。延迟确认防止发送 TCP 滑动窗口。在发送 TCP 发送窗口里的数据之后，它停止。这样消除了症状。

延迟确认有另外一个优势：它减少通信量。接收方不必确认每个段。然而，它也有缺点，那就是延迟确认可能导致发送方不必要地重传那些未确认段。

协议平衡了优势和劣势。现在它定义了确认不能延迟超过 500ms。

3.4.8 差错控制

TCP 是一个可靠的传输层协议。这意味着将数据流传递给 TCP 的应用程序依靠 TCP 将整个数据流传递给另一端的应用程序，并且是按序的、无差错的、没有任何一部分丢失或重复的。

TCP 使用差错控制提供可靠性。差错控制包括用于检测并重发损坏段的机制、用于重发丢失的段的机制、用于存储失序的段直到丢失段到达的机制，以及检测并丢弃重复段的机制。TCP 中的差错检测和纠正通过三种简单工具来完成：校验和、确认和超时。

校验和

每个段都包括校验和字段，用来检查损坏的段。如果段被损坏，它将被目的端 TCP 丢弃，并被认为是丢失了。TCP 在每段中强制使用一个 16 位的校验和。在第 5 章我们将会讨论校验和的计算。

确认

TCP 使用确认方法来证实收到了数据段。不携带数据但占用序号的一些控制段也要确认，但 ACK 段是不确认的。

> ACK 段不占序号，它不需要确认。

确认类型

在过去，TCP 只使用一种类型确认：累积确认。现在，一些 TCP 实现也使用选择性确认。

累积确认（ACK） 最初的 TCP 被设计成累积确认接收段。接收方通告下一个预期接收的字节，忽略所有失序段。这有时称为积极累积确认（positive cumulative acknowledgment）或 ACK。积极这个词表示不为丢弃、丢失或被破坏的段提供反馈。TCP 头部的 32 位 ACK 字段用来累积确认，且只有当 ACK 标志位置为 1 时才有效。

选择性确认（SACK） 越来越多的实现加入了另外一种称为选择性确认（selective acknowledgment）或 SACK 的确认类型。SACK 并不替代 ACK，但它向发送方报告额外的信息。SACK 报告失序字节块，也报告重复字节块即接收了一次以上的字节块。然而，因为 TCP 头部没有为加入这种类型的信息做准备，SACK 以一种 TCP 头部未端选项的形式实现。当我们在本书的网站讨论 TCP 选项的时候，我们会讨论这个新特性。

产生确认

接收方什么时候产生确认？在 TCP 的发展历程中，定义了很多规则也使用了很多种实现。我们给出最常见的规则。规则的顺序不代表其重要性。

1. 当终端 A 向终端 B 发送一个数据段时，它必须包含（捎带）一个确认。这个确认给出下一个期待接收的序号。这个规则降低了所需段的数量，因此减少了通信量。

2. 当接收方没有数据要发送并接收到一个有序段（带有预期序号），并且之前的段已经被确认，那么接收方延迟发送 ACK 段直到另一个段到达，或者过一段时间之后（通常 500ms）。换言之，如果只有一个未完成的有序段，那么接收方需要延迟发送 ACK 段。这个规则减少了 ACK 段。

3. 当一个带有接收方预期序号的段到达，且之前一个有序段未被确认，接收方立即发送一个

ACK 段。换言之，任何时候不能多于两个有序的未确认段存在。这防止了不必要的重传，它可能引起网络拥塞。

4. 当一个失序段到达，且它的序号大于预期，那么接收方立即发送一个 ACK 段，声明下一个预期段的序号。这导致了丢失段的快速重传（fast retransmission）（后面会讨论）。

5. 当一个丢失段到达，接收方发送一个 ACK 段，声明下一个预期序号。这通知接收方被报告丢失的段已经到达。

6. 如果重复段到达，接收方丢弃段，但是立即发送一个确认指出下一个预期的有序段。这个方法解决了当 ACK 段丢失时的一些问题。

重传

差错控制机制的核心是段的重传。当一个段被发送，它就被储存在一个队列中直到被确认。当重传计时器超时或当发送方接收到对队列中的第一个段的三次重复 ACK 时，就重传这个段。

RTO 之后重传

发送方 TCP 为每个连接维护一个**重传超时**（retransmission time-out，RTO）。当计时器到时，即超时，TCP 重发队列前面的段（具有最小序号的段）并重启计时器。注意，我们再次假设 $S_f < S_n$。我们将在后面看到 RTO 的数值在 TCP 中是动态的，并根据段的往返时间（round-trip time，RTT）进行更新。RTT 是一个段到达目的端并接收到一个确认所需要的时间。

三次重复 ACK 段之后重传

如果 RTO 的值不是很大，那么之前关于段重传的规则就足够了。当今，大多数实现遵循三次重复 ACK 规则，立即重发缺少的段。这种方法是通过允许发送方不等待超时而重传加速了因特网中的服务。这个特性称为**快速重传**（fast retransmission）。在这个版本中，如果三个对同一个段的重复确认（即一个原始 ACK 加上三个完全相同的副本）到达，那么下一个段就被重传而不必等待超时。我们在本章后面回到这个特性。

失序段

当今的 TCP 实现不丢弃失序段。它们暂时存储这些失序段，并将其标记成失序段直到缺失的段到达。然而，注意，失序段不传递给进程。TCP 确保数据按序传递给进程。

> 数据可以失序到达，并被接收到 TCP 暂时存储。但是 TCP 确保传递给进程的总是非失序的。

TCP 中数据传输的有限状态机

TCP 中的数据传输与选择性重复协议中的数据传输接近，与 GBN 有些许相似。因为 TCP 接收失序段，TCP 可以被认为行为上更像 SR 协议。但是因为原始确认是累积的，它看起来像 GBN。然而，如果 TCP 实现使用 SACK，TCP 最接近 SR。

> 以选择性重复协议为模型对 TCP 进行描述效果最好。

发送方有限状态机

让我们给出一个 TCP 协议发送方的简化有限状态机，它与我们在 SR 协议中讨论的相似，但是对于 TCP 有一些改变。我们假设通信是单向的，且发送的段被 ACK 段确认。我们也暂时忽略选择性确认和拥塞控制。图 3-59 给出了发送方简化的有限状态机。注意有限状态机是初步的；它不包含糊涂窗口综合征（Nagle 算法）或窗口关闭。它定义了单向通信，忽略所有影响双向通信的问题。

图 3-59 中的有限状态机与我们讨论的 SR 协议有一些不同。一个区别是快速重传（三次重复 ACK）。另一个区别是基于 rwnd 数值进行窗口大小调整（暂时忽略拥塞控制）。

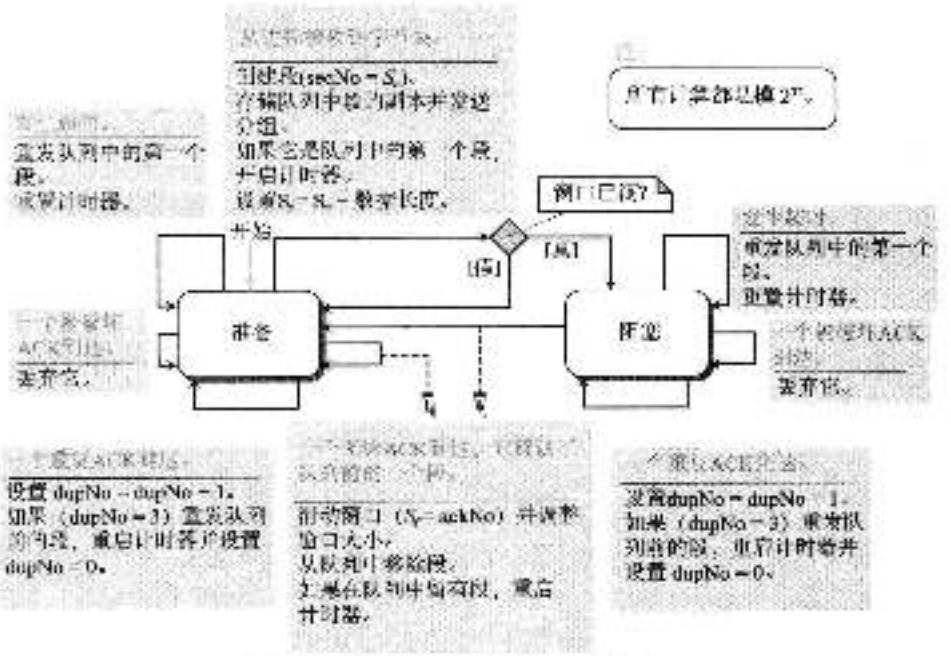


图 3-59 TCP 发送方的简化有限状态机

接收方有限状态机

让我们给出一个 TCP 协议接收方的简化有限状态机。它与我们在 SR 协议中讨论的相似，但是对于 TCP 有一些改变。我们假设通信是单向的，且使用 ACK 段确认段。我们也暂时忽略选择性确认和拥塞控制。图 3-60 给出了发送方简化的有限状态机。注意，我们忽略了一些问题，如糊涂窗口综合征（Clark 解决方法）与窗口关闭。

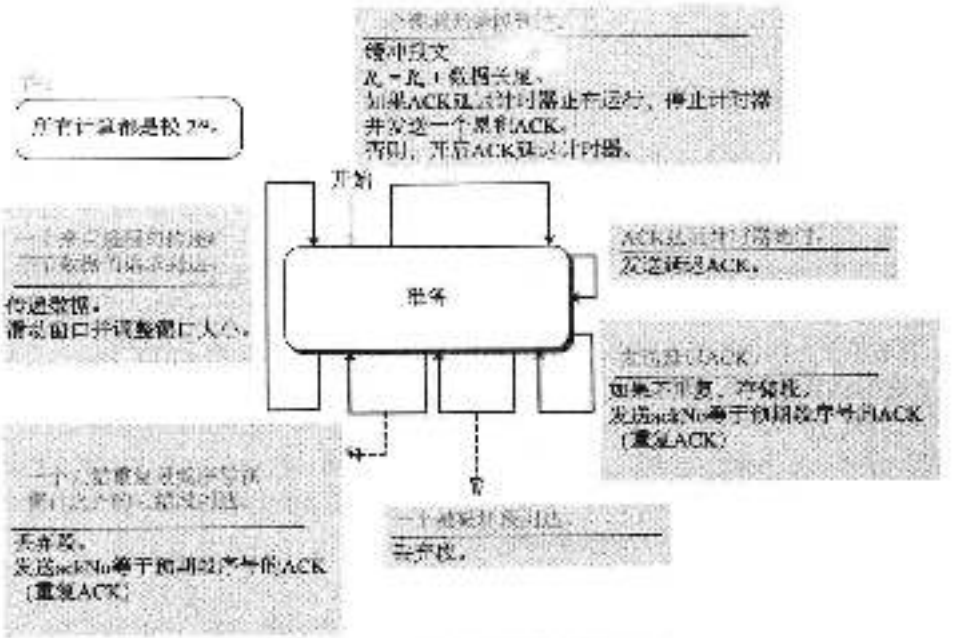


图 3-60 TCP 接收方简化有限状态机

这个有限状态机和我们在 SR 协议中讨论的有一些不同。一个区别是 ACK 在单向通信中延迟。另一个区别是发送重复 ACK 允许发送方实现快速重传策略。

我们也需要强调接收方的双向有限状态机不像 SR 中那么简单；我们需要考虑一些策略，比如，如果接收方有数据返回，那么立即发送一个 ACK。

某些场景

在本节，我们给出 TCP 操作时某些场景的例子。在这些场景中，用长方形表示段。如果段携带数据，我们显示字节序号的范围和确认字段的值。如果段仅是一个确认，我们仅用小方框中的确认号表示。

正常操作

第一种场景表示了两个系统之间的双向数据传输，如图 3-61 所示。客户 TCP 发送一个段，而服务器 TCP 发送三个段。图 3-61 表示了应用于每个确认的规则。在服务器端，只应用规则 1。如果有数据要发送，那么该段显示预期接收的下一个字节序号。当客户端接收到来自服务器的第一个段，而且它没有更多的数据要发送，那么它仅仅需要发送一个 ACK 段。但是，根据规则 2，该确认段需要延迟 500ms 以观察是否还有更多的段到达。当 ACK 延迟计时器到时，它出发一个确认。这样做是因为客户端不知道是否还有其他的段到来；它不能永远延迟确认。当下一个段到达时，启动另一个确认计时器。但是在它超时之前，第三个段到达，第三个段的到达初发另一个基于规则 3 的确认。我们没有给出 RTO 计时器，因为没有段丢失或延迟。我们仅仅假设 RTO 计时器起了作用。

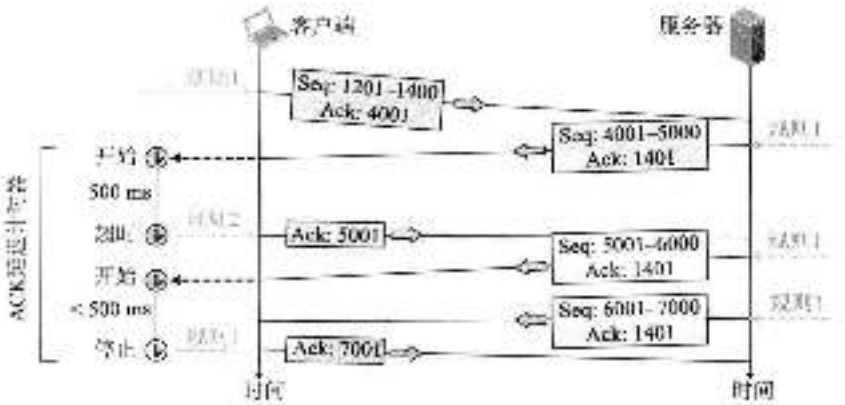


图 3-61 正常操作

丢失的段

在这种场景下，我们说明了当一个段丢失或损坏时将发生什么。接收方对丢失的段和损坏的段用相同的方法处理。丢失的段是在网络中的某处丢失的，损坏的段是被接收方本身丢弃的。图 3-62 给出了一种情况，一个段丢失（或许由于拥塞被在网络中的某个路由器丢弃掉）。

我们假定数据传输是单向的：一方发送，另一方接收。在我们的场景下，发送方发送段 1 和 2，这两个段立即被一个 ACK 确认（规则 3）。但是，段 3 被丢失了，接收方接收到段 4，发生了失序。接收方将数据存储在它的缓冲区的段中，但留出一个间隙指明数据中存在不连续性。接收方立即给发送方发送一个确认，表示了它预期的下一个字节。注意，接收方存储从 801 到 900 的字节，但是在这个间隙被填充之前不将这些字节传递给应用程序。

接收方 TCP 仅将有序的数据传递给进程。

发送方 TCP 在每个连接周期保持一个 RTO 计时器。当第三个段超时，发送方 TCP 重发段 3，

这次段3到达日被确认（规则5）。

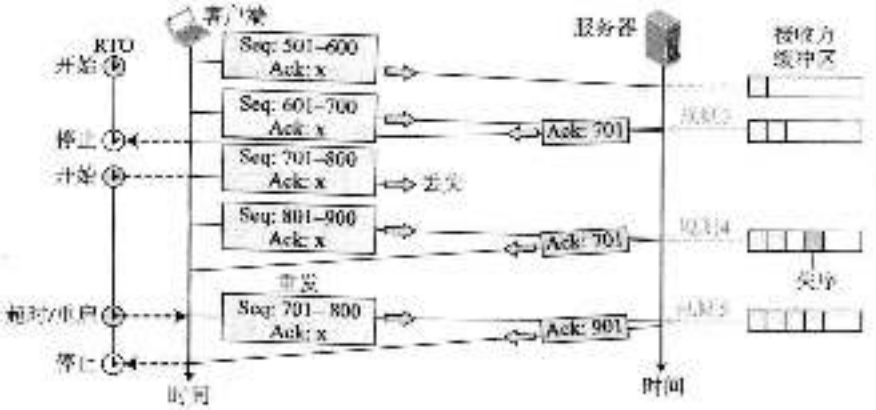


图3-62 丢失的段

快速重传

在这个场景下，我们要给出快速重传。除了RTO具有较大的值外，该情况与第二种情况相同（见图3-63）。

接收方每次接收到一个后续的段，它就触发一个确认（规则4）。发送方接收四个具有相同值的确认（三个是重复的）。尽管计时器没有到时，但是快速重传要求立即重发段3，该段是所有这些确认所预期的。在重发这个段之后，计时器被重启。

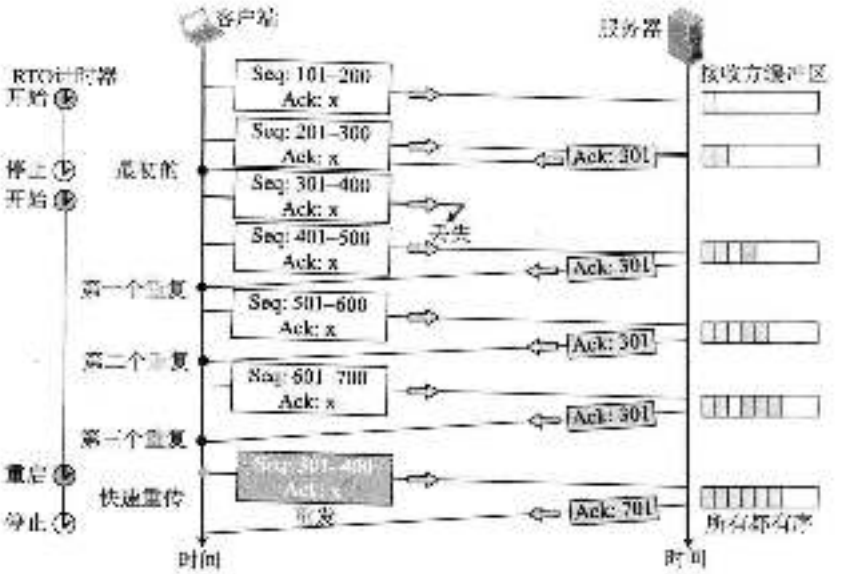


图3-63 快速重传

延迟段

第四个场景描述了一个延迟段。TCP使用IP服务，是一个无连接协议。每个IP数据报封装一个TCP段，它可能通过一条不同的路径以不同的延迟到达最终目的地。因此TCP段可能延迟。延迟段有时可能超时并被重传。如果延迟段在它被重传后到达，那么它就被认为是重复段并被丢弃。

重复段

例如，当一个段延迟并被接收方作为丢失报文时，一个重复段可以被创建。处理重复段对目的 TCP 是一个简单的过程。目的 TCP 预期连续字节流。当段到达，且其序号等于已经被接收且存储的段序号，那么它就被丢弃。ACK 被发送，其中 ackNo 定义了预期段。

自动更正丢失 ACK

这个场景给出一种情况，在丢失确认中的信息被包含在下一个确认中，这是使用累积确认的一个关键优势。图 3-64 给出了数据接收方发送的丢失确认。在 TCP 确认机制中，丢失确认甚至不会被源 TCP 通知。TCP 使用累积确认，我们可以说下一个确认自动更正之前的确认丢失。

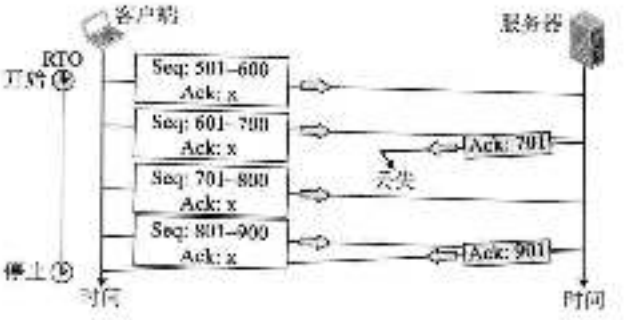


图 3-64 丢失确认

被重发段更正的丢失确认

图 3-65 给出了确认丢失的场景。

如果下一个确认延迟了很长时间或不存在确认（丢失的确认是上一次被发送的），RTO 计时器触发更正。这会造成重复段。当接收方接收到一个重复段，它就立即丢弃并重发上一个 ACK 通知发送方段已经被接收。

注意，尽管两个段没有被确认，但是只有一个段被重传。当发送方接收到重传 ACK，它知道双方是安全和完整的，因为确认是累积的。

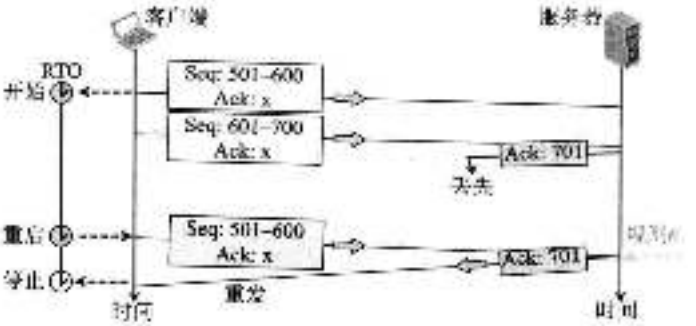


图 3-65 被重发段更正的丢失确认

丢失确认造成的死锁

存在一种情况，其中丢失确认可能导致系统死锁。这种情况中接收方发送 rwnd 设置为 0 的确认，并要求发送方临时关闭它的窗口。在一段时间后，接收方想要去掉限制；然而，如果它没有数据要发送，它就发送一个 ACK 段并以 rwnd 非零值去除限制。如果这个确认丢失了，可能产生一个问题。发送方正在等待声明非零 rwnd 的确认，接收方认为发送方已经接收到了这个确认，而且正在等待数据。这种情况称为死锁（deadlock）：每个终端等待来自另一端的响应，且任何事情都没有发生。重传计时器没有被设置。为了防止死锁，坚持计时器被设计出来，我们在本章稍后会研究到。

如果处理不当，丢失确认可能造成死锁。

3.4.9 TCP 拥塞控制

TCP 使用两种不同的策略来处理网络中的拥塞。我们在本节描述这些策略。

拥塞窗口

当我们讨论 TCP 中的流量控制，我们曾提到过接收方使用 rwnd 的数值来控制发送窗口，它在每个沿相反方向传递的段中被通告。使用这个策略保证了接收窗口不会被接收字节溢出（没有缓冲拥塞）。然而，这不意味着中间缓冲区、路由器中的缓冲区不会变得拥塞。路由器可能从不止一个发送端接收数据。无论路由器的缓冲多大，它都可能被数据淹没，这导致特定 TCP 发送方丢弃某些段。换言之，在另一端不存在拥塞，但是可能在中间存在拥塞。TCP 需要担心中间的拥塞，因为很多丢失段可能导致差错控制。更多的段丢失意味着再次重发相同的段，导致拥塞更严重，并且最终导致通信崩溃。

TCP 是使用 IP 服务的端到端协议。路由器中的拥塞是在 IP 域内，并且应该由 IP 解决。然而，正如我们在第 4 章讨论的，IP 是一个没有拥塞控制的简单协议。TCP 自身需要为这个问题负责。

TCP 不能忽略网络中的拥塞问题；它不能过分激进地向网络中发送段。正如之前提到的，这样激进的结果只能伤害 TCP 自身。TCP 也不能过于保守，每个时间间隔只发送少量的段，因为这意味着没有利用好网络可用带宽。TCP 需要定义当没有拥塞时的加速数据传输策略以及当检测到拥塞时的减速策略。

TCP 使用称为拥塞窗口（congestion window，cwnd）的变量来控制段的发送数量，这个变量的值由网络中的拥塞情况所控制（我们马上就会解释）。cwnd 变量和 rwnd 变量一起定义了 TCP 中的发送窗口大小。第一个变量与中间的拥塞相关（网络）；第二个变量与终端的拥塞相关。实际窗口的大小是这两者中的最小值

$$\text{实际窗口大小} = \text{minimum (rwnd, cwnd)}$$

拥塞检测

在讨论 cwnd 的值如何被设置和改变之前，我们需要描述 TCP 发送方如何检测到网络中可能存在的拥塞。TCP 发送方使用两个事件作为网络中拥塞的标志：超时和接收到三次重复 ACK。

第一个是超时（time-out）。如果一个 TCP 发送方在超时之前没有接收到对于某个段或某些段的 ACK，那么它就假设相应段或相应那些段丢失了，并且丢失是拥塞引起的。

另一个事件是接收到三次重复 ACK（四个带有相同确认号的 ACK）。回忆当 TCP 接收方发送一个重复 ACK，这是段已经被延迟的信号，但是发送三次重复 ACK 是丢失段的标志，这可能是由于网络拥塞造成的。然而，在三次重复 ACK 的情况下拥塞的严重程度低于超时情况。当接收方发送三次重复 ACK 时，这意味着一个段丢失，但是三个段已经被接收到。网络或者轻微拥塞或者从已经从拥塞中恢复。

我们将在稍后给出一个较早版本的 TCP，称为 Taho TCP，它处理两种事件（超时和三次重复 ACK）是相似的，但是之后的 TCP 版本，称为 Reno TCP，处理这两种事件就不同了。

TCP 拥塞中非常有趣的一点是，TCP 发送方只使用一种反馈从另一端来检测拥塞：即 ACK。没有周期性地、及时地接收到 ACK，这导致超时，是严重拥塞的标志；接到三次重复 ACK 是网络中轻微拥塞的标志。

拥塞策略

TCP 处理拥塞的一般策略基于三个算法：慢启动、拥塞避免和快速恢复。在给出在连接中 TCP 如何从一种算法转到另一种算法之前，我们先讨论每一种算法。

慢启动：指数增加

慢启动（slow-start）算法是基于拥塞窗口大小（cwnd）的思想，它以最大段长度（MSS）开始，但是每当一个确认到达时它只增加一个 MSS。如我们之前讨论的，MSS 是连接建立期间由同名的最大段长度选项协商产生的值。

这个算法的名字有些误导：算法启动慢，但它是以指数增长的。为了表示这个思想，让我们看

图 3-66。我们假设 rwnd 比 cwnd 大得多，因此发送窗口大小永远等于 cwnd。我们也假设每个段是同长度的，并携带 MSS 字节。为了简单起见，我们忽略延迟 ACK 策略并假设每个段单独被确认。

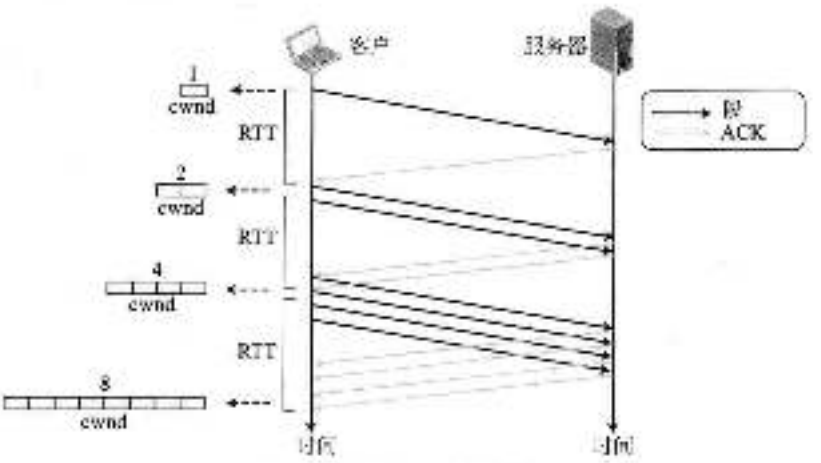


图 3-66 慢启动：指数增加

发送方以 cwnd = 1 开始。这意味着发送方仅能发送一个段。当第一个 ACK 到达后，被确认的段被从窗口中清除，这意味着现在窗口中有一个空段槽。拥塞窗口的大小也增加 1，因为收到确认标志着网络中没有拥塞。窗口的大小现在是 2。在发送两个段并接收到两个独立的确认之后，现在拥塞窗口的大小是 4，依此类推。换言之，在这个算法中拥塞窗口的大小是到达 ACK 数量的函数，可由下式决定。

$$\text{如果一个 ACK 到达，} \quad cwnd = cwnd + 1。$$

如果我们按照往返时间（RTT）观察 cwnd 的大小，那么我们发现其增长速率是指数的，这是一个非常激进的方法：

开始 → $cwnd = 1 \rightarrow 2^0$

第一个 RTT 之后 → $cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$

第二个 RTT 之后 → $cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$

第三个 RTT 之后 → $cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

慢启动不能一直继续下去。肯定存在一个停止该阶段的阈值。发送方保存一个称为 ssthresh（slow-start threshold，慢启动阈值）的变量。当窗口中的字节达到这个阈值时，慢启动停止且下一个阶段开始。

在慢启动算法中，拥塞窗口大小按指数规律增长直到到达阈值。

然而，我们已经提到慢启动策略在延迟确认情况下更慢。请记住，对于每个 ACK，cwnd 值增加 1。因此，如果两个段被累积确认，cwnd 大小只增加 1 不是 2。增长仍是指数的，但是它不是 2 的幂。对于确认了两个段的 ACK，它是 1.5 的幂。

拥塞避免：加性增加

如果我们继续慢启动算法，那么拥塞窗口大小按指数规律增大。为了在拥塞发生之前避免拥塞，必须降低指数增长的速度。TCP 定义了另一个算法，称为拥塞避免（congestion avoidance），这个算法是加性增加 cwnd 而不是指数增加。当拥塞窗口的大小到达慢启动的阈值时，这种情况下 cwnd = 1，慢启动阶段停止且加性增加阶段开始。在这个算法中，每次整个“窗口”的所有段都被确认（一次传输），拥塞窗口才增加 1。窗口是 RTT 期间传输的段的数量。图 3-67 说明了这个概念。

发送方以 cwnd = 4 开始。这意味着发送方只能发送 4 个段。在 4 个 ACK 到达之后，被确认的段被从窗口中清除，这意味着现在窗口中有一个空闲段。拥塞窗口也增加 1。窗口大小现在为 5。在发送 5 个段并接收到 5 个确认之后，拥塞窗口大小变为 6。其余以此类推。换言之，这个算法中拥塞窗口的大小也是到达的 ACK 数量的方程。它由下式决定：

如果一个 ACK 到达，cwnd = cwnd + (1/cwnd)。

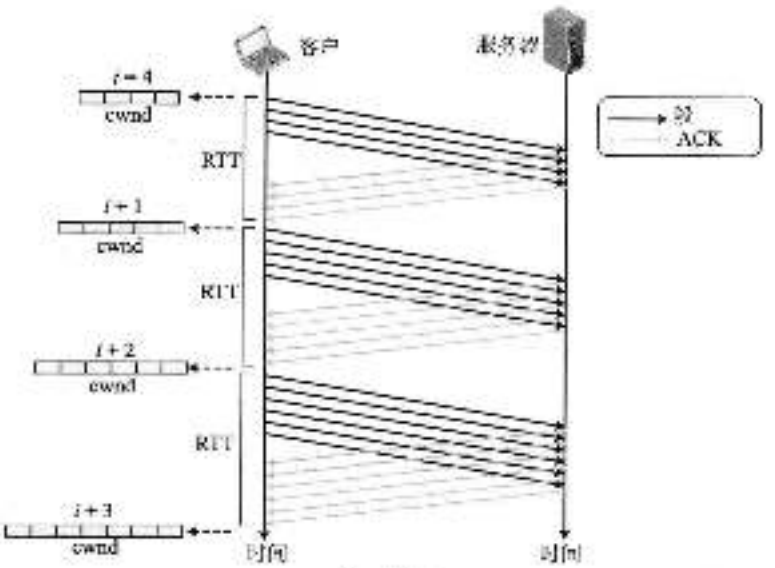


图 3-67

换言之，窗口大小每次只增加 MSS 的 1/cwnd（以字节为单位）。换言之，窗口中所有段都需要被确认，才能使窗口增加 1MSS 字节。

如果我们按照往返时间（RTT）观察 cwnd 的大小，那么我们会发现其增长速率以每次往返时间为单位是线性的，这比慢启动方法保守多了。

开始	→	cwnd = i
第一个 RTT 之后	→	cwnd = i + 1
第二个 RTT 之后	→	cwnd = i + 2
第三个 RTT 之后	→	cwnd = i + 3

在拥塞避免算法中，在检测到拥塞之后，拥塞窗口大小是加法增加的。

快速恢复 快速恢复（fast-recovery）算法在 TCP 中是可选的。旧版本的 TCP 不使用它，但是新版本使用。快速恢复开始于三次重复 ACK 到达。这被解释为网络的轻微阻塞。像拥塞避免一样，这个算法也是加性增加的，但是当一个重复 ACK 到达时（在三次重复 ACK 触发使用这个算法之后），它增加拥塞窗口的大小。我们可以说：

如果一个重复 ACK 到达，cwnd = cwnd + (1/cwnd)。

策略转换

我们讨论了 TCP 中的三种拥塞策略。现在的问题是何时使用这些策略，并且 TCP 何时从一种策略转换到另一种策略。为了回答这些问题，我们需要参照三种 TCP 版本：Taho TCP、Reno TCP 以及新 Reno TCP。

Taho TCP

早期的 TCP 称为 Taho TCP，它只使用两种拥塞策略算法：慢启动和拥塞避免。我们使用图 3-68 给出这个版本 TCP 的有限状态机。然而，需要提及的是，我们已经删除了一些琐碎行为，例如增加和重置重复 ACK 数量，这使得有限状态机不那么臃肿且更简单。

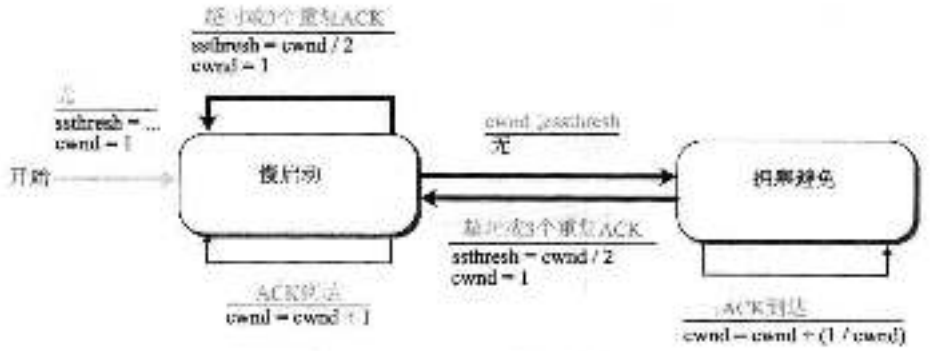


图 3-68 Taho TCP 有限状态机

Taho TCP 用相同的方式对待拥塞检测的两种情况，即超时和三次重复 ACK。在这个版本中，当连接建立，TCP 开始慢启动算法，并将变量 ssthresh 设置为之前协商好的数值（通常是 MSS 的倍数），将 cwnd 设置为 1MSS。在这种状况下，如前所述，每次 ACK 到达，拥塞窗口增加 1。我们知道这个策略很激进并且窗口是指数增加的，这可能导致拥塞。

如果检测到拥塞（发生超时或三次重复 ACK），TCP 立即中断这个激进的增长，并将阈值限定为当前 cwnd 的一半，重置拥塞窗口为 1，从而重启一个新的慢启动。换言之，不仅 TCP 从零开始，而且它还学到了如何调整阈值。如果当达到阈值时没有检测到拥塞，TCP 知道已经到达目标的顶点；它不应该继续以此速度增加。它进入拥塞避免状态并继续这种状态。

在拥塞避免状态，每当 ACK 数目等于当前已接收的窗口大小时，拥塞窗口大小增加 1。例如，如果现在窗口大小是 5MSS，在窗口大小变为 6MSS 之前，需要再接收 5 个 ACK。注意，在这种状态下，没有拥塞窗口大小的上限；除非检测到拥塞，拥塞窗口的保守加性增加将会继续，直到数据传输阶段结束。如果在这个状态下检测到拥塞，TCP 再次将 ssthresh 的值重置为 cwnd 的一半，并进入慢启动状态。

尽管这个版本的 TCP 中，ssthresh 的大小在每次拥塞检测中都在不断调整，但是这并不意味着它必然变得比之前的数值小。例如，当 TCP 处于拥塞避免阶段且 cwnd 是 20 时，如果初始 ssthresh 数值为 8MSS 且检测到拥塞，那么现在的新 ssthresh 的数值为 10，这意味着它增加了。

例 3.19 图 3-69 给出了 Taho TCP 中拥塞控制的例子。TCP 开始数据传输并将 ssthresh 设为初始的 16MSS。TCP 开始慢启动（SS）阶段 cwnd = 1。拥塞窗口指数增长，但是在第三个 RTT 发生超时（在到达阈值之前）。TCP 假设网络中存在拥塞。它立即设置新的 ssthresh = 4MSS（当前 cwnd 的一半，cwnd 为 8）并开始一个新的慢启动（SA）状态，并令 cwnd = 1MSS。拥塞指数增长，直到它到达了新设置的阈值。现在 TCP 进入拥塞避免（CA）状态且拥塞窗口加性增加，直到它到达 cwnd = 12MSS。这时，三次重复 ACK 到达，这是网络拥塞的另一个象征。TCP 再次将 ssthresh 削减一半至 6MSS，并且开始一个新的慢启动（SS）状态。cwnd 的指数增长继续。在 RTT 15 之后，cwnd 的数值为 4MSS。在发送四个段并接收两个 ACK 后，窗口大小到达阈值（6）且 TCP 进入拥塞避免状态。现在数据传输继续处于拥塞避免（CA）状态直到 RTT 20 之后连接终止。

Reno TCP

一个新版 TCP，称为 Reno TCP，在拥塞控制有限状态机中加入了新的状态，即快速恢复状态。这个版本用不同的方法处理拥塞检测的两种情况，即超时和三次重复 ACK。在这个版本中，如果发生超时，TCP 进入慢启动状态（如果它已经处于该状态，则开始新的一轮）；另一方面，如果三

次重复 ACK 到达，则 TCP 进入快速恢复阶段，并且只要有更多的重复 ACK 到达，它就保持这种状态。快速恢复状态是一种介于慢启动和拥塞避免之间的状态。它像慢启动，其中 cwnd 以指数增长，但是 cwnd 以 ssthresh 加 3MSS（而不是 1）开始。当 TCP 进入快速恢复阶段，可能发生三种主要事件。如果重复 ACK 继续到达，那么 TCP 保持这种状态，但是 cwnd 呈指数增长。如果发生超时，TCP 假设网络中有真实的拥塞，并进入慢启动状态。如果一个新的（非重复）ACK 到达，TCP 进入拥塞避免阶段，但是将 cwnd 的大小减小到 ssthresh 值，好像二次重复 ACK 没有发生过一样，并且转换是从慢启动状态到拥塞避免状态。图 3-70 给出一个 Reno TCP 的简化有限状态机。我们再次去除了一些琐碎事件来简化图和讨论。

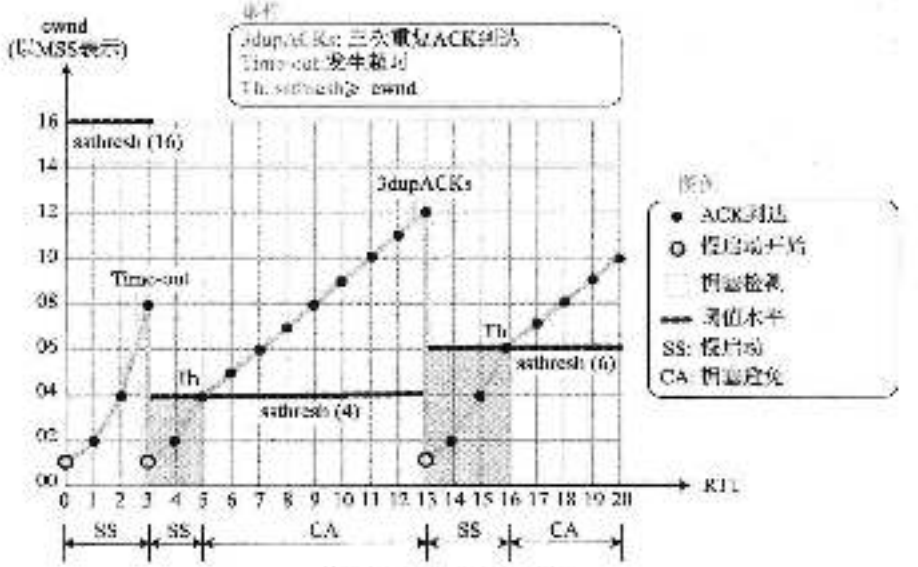


图 3-69 Taho TCP 例子

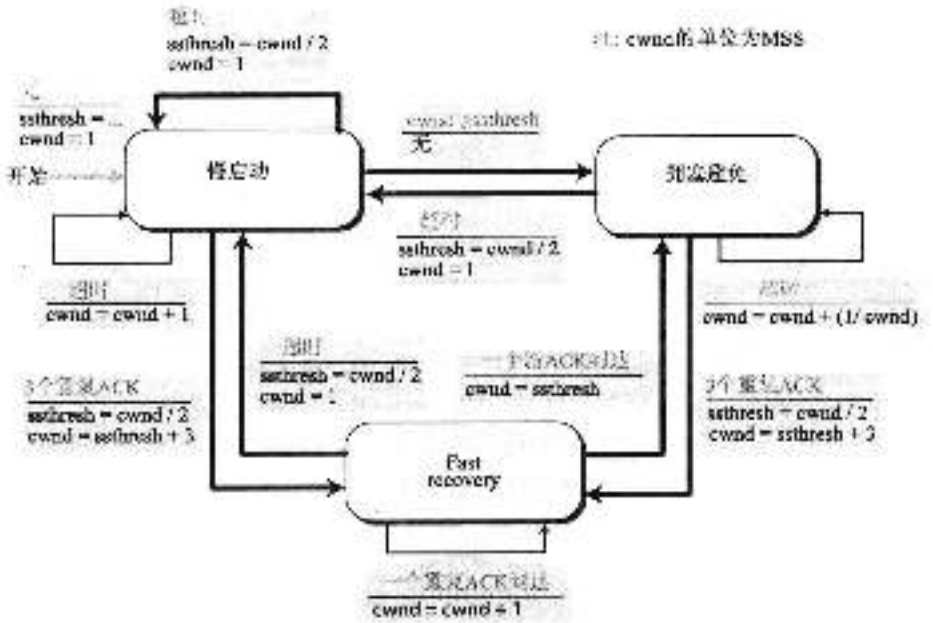


图 3-70 Reno TCP 有限状态机

例 3.20 图 3-71 给出了与图 3-69 相同的情况，但图 3-71 是在 Reno TCP 中。在三次重复 ACK 到达的 RTT 13 之前，拥塞窗口的变化都是相同的。此时，Reno TCP 将 ssthresh 降低到 6MSS（与 Taho TCP 相同），但是它将 cwnd 设置到一个更大的数值（ssthresh + 3 = 9MSS）而不是 1MSS。Reno TCP 现在进入快速恢复状态。我们假设有两个重复 ACK 在 RTT 15 到达，此处 cwnd 呈指数时长。此时，一个新的 ACK（不是重复的）到达，声明接收到丢失段。现在 Reno TCP 进入拥塞避免状态，但是首先将拥塞窗口减少到 6MSS（ssthresh 值），好像忽略整个快速重传状态并移动回之前的状态。

NewReno TCP

一个 TCP 的后期版本称为 NewReno TCP，它在 Reno TCP 基础上进行了额外优化。在这个版本中，当二次重复 ACK 到达时，TCP 检查在当前窗口中是否有一个以上的段丢失。当 TCP 接收到三次重复 ACK 时，它重传丢失段直到一个新的 ACK（非重复）到达。当检测到拥塞时，如果新的 ACK 定义了窗口的末端，TCP 可以确定只有一个段丢失。然而，如果 ACK 数定义了重传段和窗口末端之间的一个位置，那么被 ACK 定义的段也可能丢失了。NewReno TCP 重传这个段以避免接收到关于这个段的越来越多的重复 ACK。

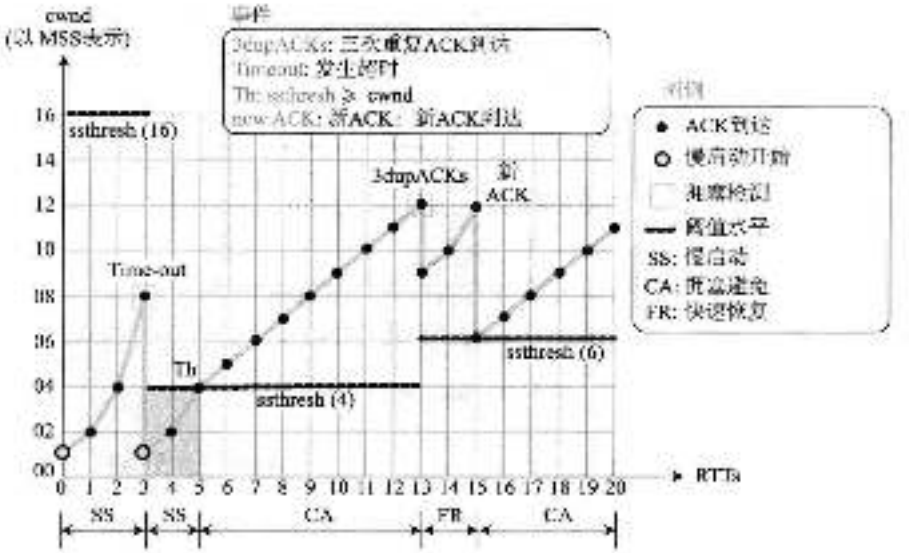


图 3-71 Reno TCP 例子

加性增加，乘性减少

在这三种 TCP 版本中，Reno 版本是如今最普遍的。已经观察到，在这个版本中，绝大多数情况下，通过观察三次重复 ACK，可以检测并处理拥塞。即使有一些超时事件，TCP 也会通过激进的指数增长从中恢复。换言之，在长时间的 TCP 连接中，如果我们忽略在快速恢复期间的慢启动状态和短暂指数增长，那么，当 ACK 到达（拥塞避免）时 TCP 拥塞窗口为 cwnd = cwnd + (1/cwnd)，并且当检测到拥塞时 cwnd = cwnd / 2，好像 SS 不存在且 FR 的长度减小为 0。第一个称为加性增加；第二个称为乘性减少。这意味着在经过初始慢启动状态后，拥塞窗口大小将锯齿样式，称为加性增加，乘性减少（additive increase，multiplicative decrease，AIMD），如图 3-72 所示。

TCP 吞吐量

TCP 的吞吐量是基于拥塞窗口的行为，如果 cwnd 是 RTT 的常数（平直直线）函数，那么吞吐量可以很容易计算出来。这个不实际的假设得出吞吐量 = cwnd / RTT。在这个假设中，在 RTT 时间内，TCP 发送一个 cwnd 字节的数据并接收到对它们的确认。TCP 的行为，如图 3-72 所示，

不是一条平直直线；它像是锯齿，有很多最大值和最小值。如果每个齿都完全相同，我们可以说吞吐量 =[(maximum + minimum) / 2] / RTT。然而，我们知道最大值是最小值的两倍，因为在每次拥塞检测中，cwnd 的数值被设为之前值的一半。因此吞吐量可以计算为：

$$\text{吞吐量} = (0.75)\, W_{max} / RTT$$

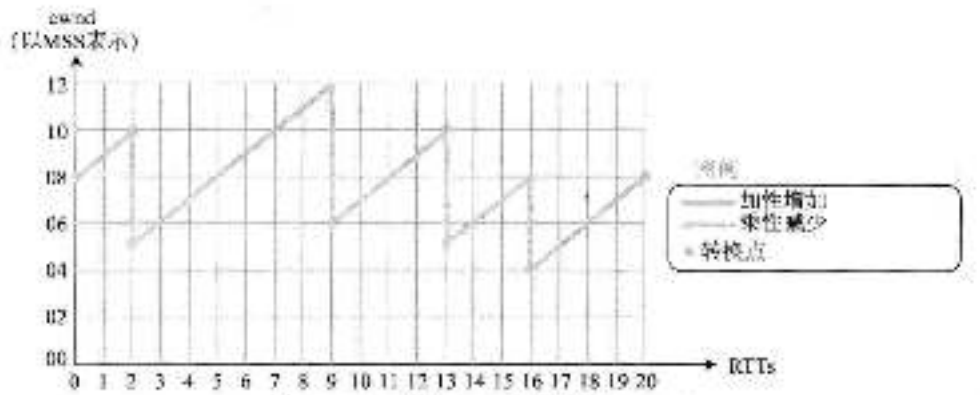


图 3-72 加性增加，乘性减少（AIMD）

其中 W_{max} 是当拥塞发生时窗口大小的平均值。

例 3.21 如果图 3-72 中 MSS = 10KB（千字节）且 RTT = 100ms，我们可以按如下计算吞吐量。

W_{max} = (10 + 12 + 10 + 8 + 8) / 5 = 9.6 MSS

吞吐量 = (0.75 W_{max} / RTT) = 0.75 × 960 kbps / 100 ms = 7.2 Mbps

3.4.10 TCP 计时器

为了更平稳地执行操作，绝大多数 TCP 实现使用至少四种计时器：重传、坚持、保活和时间等待计时器。

重传计时器

为了重传丢失的段，TCP 使用一种重传计时器（在整个连接期间）处理重传超时（RTO），即对一个段的确认等待时间。我们可以为重传计时器定义如下规则：

1. 当 TCP 发送位于发送队列前端中的段时，它开启计时器。
2. 当计时器到时，TCP 重发队列前端的第一个段并且重启计时器。
3. 当一个或多个段被累积确认，那么一个或多个段被从队列中清除。
4. 如果队列是空的，TCP 停止计时器。否则，TCP 重启计时器。

往返时间

为了计算重传超时（RTO），我们首先需要计算往返时间（Round-Trip Time，RTT）。然而，在 TCP 中计算 RTT 是一个复杂的过程，我们用一些例子一步一步进行解释。

- 测量 RTT。我们需要得到发送一个段并接收它的确认所需的时间。这就是测量 RTT。我们需要记住，段以及它们的确认没有一一对应关系；几个段可能被一起确认。一个段的测量往返时间是段到达目的地并被确认所需的时间，尽管确认可能包含其他段。注意，在 TCP 中，任何时候只有一个 RTT 测量可以进行。这就意味着，如果 RTT 测量开始，直到 RTT 数值完成，不能开始其他测量。我们使用符号 RTT_M 表示测量 RTT。

  在 TCP 中，任何时候只能进行一个 RTT 测量。

- 平滑 RTT。测量 RTT，RTT_M，可能在每个往返中改变。如今的因特网中波动很大，以至于

一次测量不能用于重传超时。绝大多数实现使用平滑 RTT，称为 RTT_S，这是 RTT_M 和前一个 RTT_S 的加权平均数，如下所示：

初始 → 无数值

在第一次测量后 → $RTT_S = RTT_M$

在每次测量后 → $RTT_S = (1 - \alpha)\ RTT_S + \alpha \times RTT_M$

α的值依赖于实现，但是它通常被设置为 1/8。换言之，新 RTT_S 被计算成 7/8 的旧 RTT_S 加 1/8 的当前 RTT_M。

- RTT 偏差。绝大多数实现不仅仅使用 RTT_S；它们也计算称为 RTT_D 的 RTT 偏差，计算方法基于 RTT_S 和 RTT_M，使用如下方程（β值也是依赖实现的，但是通常设置为 1/4）：

初始 → 无数值

在第一次测量后 → $RTT_D = RTT_M / 2$

在每次测量后 → $RTT_D = (1 - \beta)\ RTT_D + \beta \times |\ RTT_S - RTT_M\ |$

重传超时（RTO） RTO 的值基于平滑往返时间以及它的偏差。绝大多数实现使用如下公式来计算 RTO：

最初 → 初始数值

在每次测量后 → $RTO = RTT_S + 4 \times RTT_D$

换言之，将运行中 RTT_S 的平滑平均数加上 4 倍运行中 RTT_D 的平滑平均数（通常是一个很小的值）。

例 3.22 让我们来给出一个值超例子。图 3-73 给出连接的部分。图中给出连接建立和部分数据传输阶段。

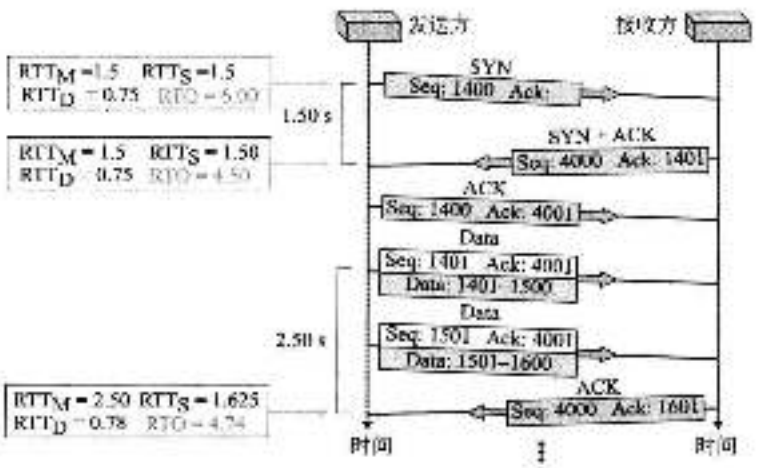


图 3-73 例 3.22

1. 当 SYN 段被发送，RTT_M、RTT_S 和 RTT_D 没有数值。RTO 的数值被设置为 6.00 秒。此时，以下给出了这些变量的数值：

$RTO = 6$

2. 当 SYN + ACK 段到达，RTT_M 被测量且等于 1.5 秒。以下给出这些变量的数值：

$RTT_M = 1.5$

$RTT_S = 1.5$

$RTT_D = (1.5)/2 = 0.75$

$RTO = 1.5 + 4 \times 0.75 = 4.5$

3. 当第一个数据段被发送，一个新 RTT 测量开始。注意，当发送方发送一个 ACK 段，发送方不开始 RTT 测量，因为它并不消耗序号且没有超时。不对第二个数据段进行 RTT 测量，因为测量已经在进行。上一个 ACK 段的到达被用来计算下一个 RTT_M 数值。尽管上一个 ACK 段确认数据段（累积），它的到来完成了第一个段的 RTT_M 数值。这些变量的数值如下所示。

$RTT_M = 2.5$

$RTT_S = (7/8) \times (1.5) + (1/8) \times (2.5) = 1.625$

$RTT_D = (3/4) \times (0.75) + (1/4) \times |1.625 - 2.5| = 0.78$

$RTO = 1.625 + 4 \times (0.78) = 4.74$

Karn 算法

假设在重传超时期间段没有被确认，那么它应该被重传。当发送方 TCP 接收到这个段的确认，它不知道这个确认针对的是原始段的还是重传段。新 RTT 值基于段的离开。然而，如果原始段丢失且确认是针对重传段的，当前 RTT 值必须在段被重传的时候计算。这个歧义被 Karn 解决。Karn 算法很简单。不要考虑 RTT 计算中的重传段的往返时间。直到你发送一个段并接收一个段而不必重传时，才更新 RTT 数值。

TCP 并不考虑新 RTO 计算中重传段的 RTT。

指数后退

如果发生重传，RTO 的值是多少？绝大多数 TCP 实现使用指数退后策略。RTO 的数值再每次重传中翻倍。因此如果段被重传一次，数值是两倍 RTO。如果被重传两次，数值是四倍 RTO。等等。

例 3.23 图 3-74 是之前例子的继续。这里应用了重传和 Karn 算法。

图中第一个段被发送，但是丢失了，RTO 计时器在 4.74 秒后到时。段被重传且计时器被设置为 9.48，是 RTO 数值的两倍。这次 ACK 在超时前被接收。我们等待，直到我们发送一个新的段并在重新计算 RTO 之前接收到针对它的 ACK（Karn 算法）。

坚持计数器

为了处理 0 窗口大小通告，TCP 需要另一个计时器。如果接收 TCP 声明了 0 窗口大小，那么发送 TCP 停止传输段，直到接收 TCP 发送一个 ACK 段声明非零的窗口大小。这个 ACK 段可能丢失。如果这个确认丢失了，接收 TCP 认为它已经完成了工作并等待发送方 TCP 发送更多的段。对于一个只包含确认的段不存在重传计时器。发送方 TCP 没有接收到确认，并且等待另一个 TCP 发送确认通告窗口的大小。两端 TCP 可能继续相互等待直到永远（死锁）。

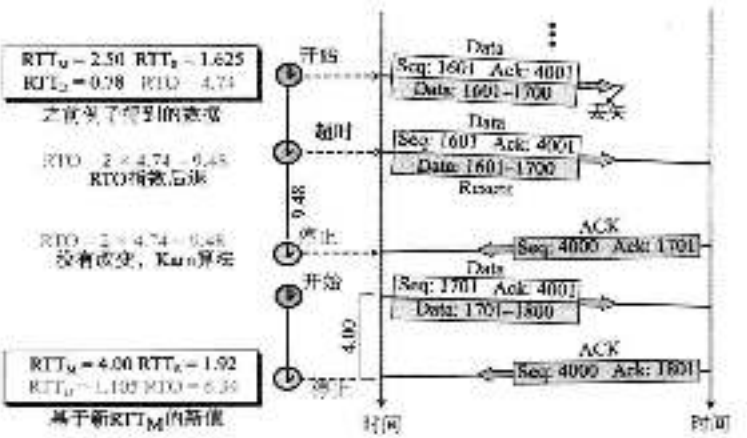


图 3-74 例 3.23

为了更正死锁，TCP 为每个连接使用坚持计时器（persistence timer）。当发送方 TCP 接收一个窗口大小为 0 的确认时，开启坚持计时器。当坚持计时器到时，发送方 TCP 发送一个特殊的段，称为探测（probe）。这个段只包含一字节的新数据。它有一个序号，但是序号从不被确认；在为剩余的数据计算序号时它甚至被忽略。探测报文引发接收方 TCP 重发确认。

坚持计时器的值被设置为重传时间。然而，如果一个响应没有被从接收方接收到，另一个探测段就会被发送且坚持计时器的数值会被加倍重置。发送方继续发送探测段并加倍重置坚持计时器数值，直到这个数值到达阈值（通常为 60s）。这之后，发送方每 60 秒发送一个探测段，直到窗口重新打开。

保活计时器

保活计时器（keepalive timer）通常在某些实现中使用，来防止两个 TCP 之间的长期空闲连接。假设有一个客户与一个服务器建立了连接，传输了一些数据并进入沉默状态。或许客户已经崩溃。在这种情况下，连接会永远保持打开。

为了消除这种情况，绝大多数实现给服务器配备保活计时器。每当服务器从客户收到一次数据，就重置计时器。超时时间通常是 2 小时。如果服务器在 2 小时内没有收到客户数据，那么它发送一个探测段。如果每 75 秒钟发送一个探测段，一共发送 10 个探测段之后仍无客户响应，那么服务器就认为客户端出现了故障，并终止这个连接。

时间等待计时器

时间等待（TIME-WAIT）（2MSL）计时器在连接终止期间使用。最大报文段寿命（maximum segment life time，MSL）是任意报文在被丢弃前在网络中的存在时间。具体实现需要为 MSL 选择数值。通常为 30 秒、1 分钟甚至 2 分钟。当 TCP 执行主动关闭并发送最后一个 ACK 时，使用 2MSL 计时器。连接必须保持 2MSL 的时间，从而允许 TCP 重发最后一个 ACK，以防 ACK 丢失。这要求另一端的 RTO 计时器超时且新 FIN 和 ACK 段被重发。

3.4.11 选项

TCP 头部有多达 40 字节选项信息。选项向目的端传递额外信息或调整其他选项。这些选项包含在本书网站以供未来参考。

CP 选项在本书网站上进行讨论。

3.5 章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。

书籍

一些书籍论述了传输层协议相关内容。在本书末列出了方括号中的参考资料：我们尤其推荐[Com 06]、[Pet & Dav 03]、[Gar & Wid 04]、[Far 04]、[Tan 03]以及[Sta 04]。

RFC

与 UDP 相关的主要 RFC 是 RFC 768。一些 RFC 讨论 TCP 协议，包括 RFC 793、RFC 813、RFC 879、RFC 889、RFC 896、RFC 1122、RFC 1975、RFC 1987、RFC 1988、RFC 1993、RFC 2018、RFC 2581、RFC 3168 以及 RFC 3782。

小结

传输层协议的主要职责是提供进程到进程通信。为了定义进程，我们需要端口号。客户程序使用临时端口号定义自身，服务器使用熟知端口号定义自身。为了从一个进程向另一个进程发送报文，

传输层协议对报文进行封装和解封装。源端的传输层执行多路复用；目的端的传输层执行多路分解。流量控制平衡生产者和消费者之间的数据项交换。传输层协议可以提供两种类型的服务：无连接和面向连接。在无连接服务中，发送方向接收方发送分组，而没有连接建立。在面向连接服务中，客户和服务器首先需要在它们之间建立连接。

在本章中，我们已经讨论过很多常见的传输层协议。停止-等待协议提供了流量和差错控制，但是这是不够的。回退 N 帧协议是停止-等待协议的更高效版本并且使用流水线。选择性重复协议，一个回退 N 帧协议的修改版，更适合于处理分组丢失。所有这些协议都可以使用捎带实现双向通信。

UDP 是一个传输协议，它创建了进程到进程的通信。UDP（大多）是不可靠的无连接协议。它需要很小的开销并提供快速传递。UDP 分组称为用户数据报。

传输控制协议（TCP）是 TCP/IP 协议簇中的另一个传输层协议。TCP 提供进程到进程、全双工的面向连接服务。两个使用 TCP 软件的服务之间的数据传输单位称为段。TCP 连接包含三个阶段：连接建立、数据传输和连接终止。TCP 软件通常以有限状态机（FSM）形式实现。

3.6 习题集

测试题

本章的交互测试题集可以在本书的网站上找到。强烈推荐学生们做这些测试题，这样可以在学生做习题集前检测他们对课程内容的理解。

练习题

Q3-1 假设在一个系统中我们有一组专用计算机，每台都用来执行单独一个任务。我们仍然需要主机到主机和进程到进程的通信以及两级地址吗？

Q3-2 操作系统给何个正在运行的应用程序分配一个进程号。你能解释为什么这些进程号不能代替端口号吗？

Q3-3 假设你需要在你家的两台主机上编写并测试一个客户-服务器应用程序。

a. 你将为客户程序选择什么范围的端口号？

b. 你将为主机程序选择什么范围的端口号？

c. 两个端口号可以相同吗？

Q3-4 假设一个新的组织需要创建一个新的服务器进程并允许它的客户使用这个进程访问组织站点。服务器进程的端口号应该如何选择？

Q3-5 在网络中，接收窗口的大小是 1 个分组。以下哪个协议可以被网络使用？

a. Stop-and-Wait　　b. Go-Back-N　　c. Selective-Repeat

Q3-6 在网络中，发送窗口的大小是 20 个分组。以下哪个协议可以被网络使用？

a. Stop-and-Wait　　b. Go-Back-N　　c. Selective-Repeat

Q3-7 在 m 是定值且 $m > 1$ 的网络中，我们可以使用回退 N 帧或选择性重复协议。请描述这两者的优缺点。在选择这些协议的时候还需要考虑什么其他因素标准？

Q3-8 既然存储分组序号的字段在大小上是有限的，那么协议中的序号就需要回绕，这意味着两个分组可能有相同的序号。在一个序号字段是 m 位的协议中，如果分组具有序号 x，需要发送多少的分组才能看到一个相同的序号 x，假设每个分组都被分配一个序号。

Q3-9 我们在前一个问题中讨论的回绕情况造成了一个网络中的问题吗？

Q3-10 你能解释为什么某些传输层分组可能在因特网中失序接收吗？

Q3-11 你能解释为什么某些传输层分组可能在因特网中丢失吗？

Q3-12 你能解释为什么某些传输层分组可能在因特网中重复吗？

Q3-13 在回退 N 帧协议中，发送窗口的大小可能是 $2^m - 1$，而接收窗口大小只是 1。当发送和接收窗口大小有很大差别时，流量控制是如何实现的？

Q3-14 在选择性重复协议中，发送和接收窗口的大小是相同的。这是否意味着应该没有分组处于运输中呢？

Q3-15 一些应用程序可以使用两种传输层协议（UDP 或 TCP）。当一个分组到达目的端，计算机如何找出它

使用哪种传输层协议?

Q3-16 一个位于 IP 地址为 122.45.12.7 主机的客户发送一个报文到相应服务器，服务器位于 IP 地址为 200.112.45.90 的主机。如果熟知端口号是 161 且临时端口是 51000，通信中使用的套接字地址对是什么?

Q3-17 UDP 是一个面向报文的协议。TCP 是一个面向字节的协议。如果一个应用需要保护报文边界，应该使用什么协议，UDP 还是 TCP?

Q3-18 在第 2 章，我们说我们可以使用不同的应用程序接口（API），如套接字接口、TLI 和 STREAM。这些可以用来提供客户-服务器通信。这是不是意味着我们应该使用支持这些 API 的不同的 UDP 或 TCP 协议？请解释。

Q3-19 假设一个私有网络，主机之间使用点到点通信并且不需要路由，它完全不使用网络层。这个网络仍然能从 UDP 或 TCP 中获益吗？换言之，用户数据报或段能被封装到以太帧中吗?

Q3-20 假设一个私有网络使用与 TCP/IP 完全不同的协议簇。这个网络也能使用 UDP 或 TCP 服务作为端到端报文通信的媒介吗?

Q3-21 你能解释在 TCP 中为什么需要 4（有时是 3 个）段来结束连接吗?

Q3-22 在 TCP 中，一些段类型只能被用来控制；它们不能同时携带数据。你能定义一些这样的段吗?

Q3-23 在 TCP 中，我们如何定义段的序号（在每个方向上）？考虑两种情况：第一个段和其他段。

Q3-24 在 TCP 中，我们有三个连续段。假设第一个段的序号是 101，以下面和情况中，下一个段的序号是多少?

a. 第一个段不消耗序号。

b. 第一个段消耗 10 个序号。

Q3-25 在 TCP 中以下每个段消耗多少序号?

a. SYN　　b. ACK　　c. SYN + ACK　　d. Data

Q3-26 你能解释在 TCP 中为什么 SYN、SYN + ACK 以及 FIN 段，每个段消耗一个序号，但是一个不携带数据的 ACK 段不消耗序号吗?

Q3-27 观察 TCP 头部（见图 3-44），当窗口大小只有 16 位长时，我们发现序号是 32 位长。这是不是意味着 TCP 在这方面更接近回退 N 帧或选择性重复协议?

Q3-28 TCP 中的窗口最大值规划被设计为 64KB（这意味着 64 × 1024 − 65 536 或者实际上为 65 535）。你能说出这里的原因吗?

Q3-29 TCP 头部大小的最大值是多少？TCP 头部大小的最小值是多少?

Q3-30 在 TCP 中，SYN 段是在一个方向上打开连接还是两个方向?

Q3-31 在 TCP 中，FIN 段是在一个方向上关闭连接还是两个方向?

Q3-32 在 TCP 中，什么类型的标志位可以完全关闭双向的通信?

Q3-33 绝大多数的标志位可以在段中一起使用。给出一个由于歧义不能同时使用两种标志位的例子。

Q3-34 假设一个客户向服务器发送一个 SYN 段。当服务器检查数据端口号时，它发现没有端口号所定义的正在的运行进程。这种情况下，服务器应该怎么做?

Q3-35 你能解释为什么 TCP 使用了不可靠 IP 提供的服务，却能够提供可靠通信吗?

Q3-36 我们说 TCP 提供了两个应用程序之间的面向连接服务。在这种情况下，连接需要连接标识符用以相互区分。你认为这种情况下独立的连接标识符是什么?

Q3-37 假设 Alice 使用她的浏览器打开了两个通向 Bob 服务器上运行着的 HTTP 服务器的连接。TCP 如何区分这两个连接?

Q3-38 在讨论使用 TCP 的面向连接通信中，我们使用被动打开和主动打开这两个术语。假设 Alice 和 Bob 之间有电话交谈，由于电话交谈是面向连接通信的一个例子，因此假设 Alice 打电话给 Bob 并且他们在电话上交谈。在这种情况下谁在进行一个被动打开连接？谁在进行一个主动打开连接?

Q3-39 在 TCP 中，发送窗口可以小于、大于或等于接收窗口吗?

Q3-40 你能举出一些由一个 TCP 段或一些 TCP 段组合完成的任务吗?

Q3-41 在 TCP 段中，序号表示什么?

Q3-42 在 TCP 段中，确认号表示什么?

Q3-43 差错控制中使用的校验和在以下协议中是可选的还是强制的?

a. UDP　　b. TCP

Q3-44 假设一个 TCP 服务器期待接收 2001 字节，但是它接收了序号为 2200 的段。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-45 假设一个 TCP 客户期待接收 2001 字节，但是它接收了序号为 1201 的段。TCP 客户对这个事件的反应是什么？你能证明这个反应吗？

Q3-46 假设一个 TCP 服务器期待接收 2001 到 3000 字节，但是它接收了序号为 2001 携带 400 字节的段。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-47 假设一个 TCP 服务器期待接收 2401 字节，但是它接收了序号为 2401 携带 500 字节的段。如果此时服务器没有数据要发送且没有确认之前的段，TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-48 假设一个 TCP 客户期待接收 3001 字节，但是它接收了序号为 3001 携带 400 字节的段。如果此时客户没有数据要发送且没有确认之前的段，TCP 客户对这个事件的反应是什么？你能证明这个反应吗？

Q3-49 假设一个 TCP 服务器期待接收 6001 字节，但是它接收了序号为 6001 携带 2000 字节的段。如果此时服务器要发送 4001 到 5000 字节。TCP 服务器对这个事件的反应是什么？你能证明这个反应吗？

Q3-50 为 TCP 产生 ACK 的第一条规则没有在图 3-59 或图 3-60 中给出。请解释原因。

Q3-51 在我们表述的产生 ACK 的六条规则中，哪一条可以应用于服务器从客户接收到 SYN 段？

Q3-52 在我们表述的产生 ACK 的六条规则中，哪一条可以应用于客户从服务器接收到 SYN + ACK 段？

Q3-53 在我们表述的产生 ACK 的六条规则中，哪一条可以应用于客户或服务器从另一端接收到 FIN 段？

思考题

P3-1 比较 16 位地址的范围（即 0 到 65 535）与 32 位 IP 地址的范围（即 0 到 4 294 967 295）（第 4 章讨论），为什么我们需要这么大范围的 IP 地址，但只需要相对较小的端口号范围？

P3-2 你能解释为什么 ICANN 将端口号分成三组：熟知、注册和动态吗？

P3-3 发送端向同一个目的发送一系列分组，使用 5 位序号。如果序号从 0 开始，第 100 个分组的序号是多少？

P3-4 在如下的每个协议中，在回绕发生前可以有多少分组拥有独立的序号？

a. Stop-and-Wait　　b. Go-Back-N 其中 $m = 8$　　c. Select-Repeat 其中 $m = 8$

P3-5 使用 5 位序号，以下协议中发送和接收窗口最大是多少？

a. Stop-and-Wait　　b. Go-Back-N　　c. Select-Repeat

P3-6 给出一个带有三种状态的假想状态机的有限状态机：A（开始状态），B 和 C；以及四个事件：事件 1、2、3 和 4。以下指定了状态机的行为：

a. 当处于状态 A 时，可能发生两个事件：事件 1 和事件 2。如果事件 1 发生，状态机执行动作 1 并进入状态 B。如果事件 2 发生，状态机进入状态 C（无动作）。

b. 当处于状态 B 时，可能发生两个事件：事件 3 和事件 4。如果事件 3 发生，状态机执行动作 2 并进入状态 B。如果事件 4 发生，状态机进入状态 C。

c. 当处于状态 C 时，状态机永远保持这种状态。

P3-7 假设我们的网络永远不会出现被破坏、丢失或重复分组。我们只考虑流量控制。我们不想让发送方用分组淹没接收方。设计一个有限状态机，当接收方准备好时允许发送方给接收方发送一个分组。如果接收方准备好接收一个分组，它就发送一个 ACK。没有接收到发给发送方的 ACK 就表示接收方没有准备好接收更多的分组。

P3-8 假设我们的网络永远不会出现被破坏、丢失或重复分组。我们也只考虑流量控制。我们不想让发送方用分组淹没接收方。设计一个新协议的有限状态机来实现这些特性。

P3-9 在停止-等待协议中，给出接收方接收到一个重复分组（非是失序的）的情况。提示：考虑延迟 ACK。接收方对这个事件的反应是什么？

P3-10 假设我们想改变停止-等待协议并加入 NAK（消极 ACK）分组到系统中。当一个被破坏分组到达接收方，接收方丢弃这个分组。但是发送一个 NAK，其中 nakNo 定义了被破坏分组的 seqNo。按这种方式，发送方能够重发被破坏分组而不用等待超时。请解释图 3-21 中的有限状态机需要作何改变且用时间线图给出实现这个新协议的例子。

P3-11 重画图 3-19，其中 5 个分组被交换（0、1、2、3 和 4）。假设分组 2 丢失且分组 3 在分组 4 之后到达。

P3-12 创建一个与图 3-22 相似的场景，其中发送方发送三个分组。第一个和第二个分组到达且被确认。第

三个分组延迟并被重发。在对原始分组的确认被发送后，重复分组被接收到。

P3-13 创建一个与图 3-22 相似的场景，其中发送方发送两个分组。第一个分组被接收到且被确认，但是确认丢失。发送方在超时后重发分组。第二个分组丢失并且重发。

P3-14 重画图 3-29，其中发送方发送 5 个分组（0、1、2、3 和 4）。分组 0、1 和 2 被发送且被一个 ACK 确认，这个 ACK 在所有分组被发送后到达发送端。分组 3 被接收且被一个 ACK 确认。分组 4 丢失并被重发。

P3-15 重画图 3-35，其中发送方发送 5 个分组（0、1、2、3 和 4）。分组 0、1 和 2 被按序发送且被一个一个地确认。分组 3 被延迟并在分组 4 之后被接收。

P3-16 回答如下与停止-等待协议有限状态机相关的问题（见图 3-21）：

a. 发送状态机处于准备状态且 $S = 0$。下一个待发送分组的序号是多少？

b. 发送状态机处于阻塞状态且 $S = 1$。如果发生超时下一个待发送分组的序号是多少？

c. 接收状态机处于准备状态且 $R = 1$。序号为 1 的分组到达。回应这个事件的动作是什么？

d. 接收状态机处于准备状态且 $R = 1$。序号为 0 的分组到达。回应这个事件的动作是什么？

P3-17 回答如下与回退 N 帧协议 $m = 6$ 位的有限状态机相关的问题。假设窗口大小是 63（见图 3-27）：

a. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。下一个待发送分组的序号是多少？

b. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。超时发生。有多少分组待重发？它们的序号是多少？

c. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。一个 ackNo = 13 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？

d. 发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。窗口大小是多少？

e. 发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。一个 ackNo = 18 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？发送状态机的状态是什么？

f. 接收方状态机处于准备状态且 $R_n = 16$。序号为 16 的分组到达。下一个 R_n 的值是多少？状态机对这个事件的响应是什么？

P3-18 回答如下与回退 N 帧协议 $m = 7$ 位的有限状态机相关的问题。假设窗口大小是 64（见图 3-34）：

a. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。下一个待发送分组的序号是多少？

b. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。分组 10 的计时器超时。有多少分组待重发？它们的序号是多少？

c. 发送状态机处于准备状态且 $S_f = 10$，$S_n = 15$。一个 ackNo = 13 的 ACK 到达。下一个 S_f 和 S_n 的值是多少？回应这个事件的动作是什么？

d. 发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。窗口大小是多少？

e. 发送状态机处于阻塞状态且 $S_f = 14$，$S_n = 21$。一个 ackNo = 18 的 ACK 到达。分组 15 和 16 已经被确认，下一个 S_f 和 S_n 的值是多少？发送状态机的状态是什么？

f. 接收方状态机处于准备状态且 $R_n = 16$。窗口大小为 8。序号为 16 的分组到达。下一个 R_n 的值是多少？状态机对这个事件的响应是什么？

P3-19 我们将网络中的带宽延迟乘积定义为在往返时间（RTT）内可以处于管道中的分组数。以下每种情况中的带宽延迟乘积是多少？

a. 带宽：1 Mbps，RTT：20 ms，分组大小：1000 位

b. 带宽：10 Mbps，RTT：20 ms，分组大小：2000 位

c. 带宽：1 Gbps，RTT：4 ms，分组大小：10 000 位

P3-20 假设我们需要为一个网络设计回退 N 帧滑动窗口协议，这个网络带宽是 100Mbps 且发送方和接收方之间的平均距离是 10 000km。假设平均分组大小是 100 000 位且介质中的传输速度是 2×10^8m/s。找出发送和接收窗口的最大值，序号字段的位数（m）以及计时器适当的超时值。

P3-21 假设我们需要为一个网络设计选择性重复滑动窗口协议，这个网络带宽是 1Gbps 且发送方和接收方之间的平均距离是 5 000km。假设平均分组大小是 50 000 位且介质中的传输速度是 2×10^8m/s。找出发送和接收窗口的最大值，序号字段的位数（m）以及计时器的适当超时值。

P3-22 回退 N 帧协议中的确认号定义了下一个预期分组，但是选择性重复协议中的确认号定义了被确认的序号。你能解释原因吗？

P3-23 在使用回退 N 帧协议的网络中 $m=3$ 且发送窗口的大小是 7，变量值为 $S_f=62$，$S_n=66$ 且 $R_n=64$。假设网络不复制或重排分组。

a. 正在传输的数据分组的序号是多少？ b. 正在传输的 ACK 分组确认号是多少？

P3-24 在使用选择性重复协议的网络中 $m=4$ 且发送窗口的大小是 8，变量值为 $S_f=62$，$S_n=67$ 且 $R_n=64$。分组 65 已经在发送端被确认；分组 65 和 66 被接收端失序接收。假设网络没有复制分组。

a. 即将来临的数据分组的序号是多少（正在传输、被损坏或丢失）？

b. 即将来临的 ACK 分组的确认号是多少（正在传输、被损坏或丢失）？

P3-25 回答下列问题：

a. UDP 用户数据报的最小是大？

b. UDP 用户数据报的最大是大？

c. 可以被封装到 UDP 用户数据报中的应用层有效载荷数据最小是多大？

d. 可以被封装到 UDP 用户数据报中的应用层有效载荷数据大是多大？

P3-26 一个客户使用 UDP 向服务器发送数据。数据长度是 16 字节。计算 UDP 层的传输效率（有用字节与总字节的比）。

P3-27 以下是 UDP 头部十六进制格式的转储（内容）。

0045DF0000580000

a. 源端口号是多少？

b. 目的端口号是多少？

c. 用户数据报的总长度是多少？

d. 数据长度是多少？

e. 分组是从客户到服务器的吗？还是相反的？

f. 应用层协议是什么？

g. 发送方计算这个分组的校验和了吗？

P3-28 比较 TCP 和 UDP 头部，列出 TCP 头部中 UDP 头部所没有的字段。给出 UDP 头部中没有这些字段的原因。

P3-29 在 TCP 中，如果 HLEN 的值为 0111，被包含进段的选项有多少字节？

P3-30 以下 TCP 段是什么类型的？其中控制字段的值为：

a. 000000 b. 000001 c. 010001

d. 000100 e. 000010 f. 010010

P3-31 TCP 段中的控制字段是 6 位。我们可以有 64 中不同的组合。列出你认为通常使用的组合。

P3-32 以下是 TCP 头部十六进制格式的转储（内容）。

E293 0017 00000001 00000000 5002 07FF...

a. 源端口号是多少？ b. 目的端口号是多少？

c. 序号是多少？ d. 确认号是多少？

e. 头部长度是多少？ f. 段类型是什么？

g. 窗口大小是多少？

P3-33 为了更好地理解三次握手连接建立的必要性，让我们仔细检查场景。Alice 和 Bob 不能打电话或上网（想一下过去的日子）来建立他们下一次会见，这个会见是在离家很远的一个地方。

a. 假设 Alice 发送了一封信给 Bob 并确定了他们会见的日期和时间。Alice 能够到会见地点并确定 Bob 也在吗？

b. 假设 Bob 用信件回复了 Alice 的请求并确认日期和时间。Bob 能够到会见地点并确定 Alice 也在吗？

c. 假设 Alice 回复了 Bob 的请求并确认相同的日期和时间。他们中的一个人能够到会见地点并确定对方也在吗？

P3-34 为了使初始序号是随机数字，绝大多数系统在启动阶段从 1 开启一个计数器并每半秒增加 64 000。计数器回绕一次需要多长时间？

P3-35 在 TCP 连接中，客户端的初始序号是 2171。客户打开连接，发送一个段，第二个段携带 1000 字节数据，并关闭连接。以下客户发出的每个段的序号是多少？

a. SYN段　　b. 数据段　　c. FIN段

P3-36 在连接中，cwnd 的值为 3000 且 rwnd 为 5000。主机已经发送 2000 字节，没有被确认。可以再发送多少字节？

P3-37 一个客户使用 TCP 发送数据给服务器。数据包含 16 字节。计算 TCP 层的传输效率（有用字节与总字节的比）。

P3-38 TCP 正在以每秒 1 兆字节速度发送数据。如果序号从 7000 开始，序号回到 0 需要多长时间？

P3-39 一个 HTTP 客户使用初始序号（ISN）14 534 打开一个 TCP 连接，并且临时端口号是 59 100。服务器打开连接，它的 ISN 为 21 732。如果 rwnd 是 4000 且服务器定义 cwnd 为 5000，请给出连接建立期间的三个 TCP 段。忽略校验和字段的计算。

P3-40 假设在之前问题中的 HTTP 客户发送一个 100 字节的请求。服务器回应了一个 1200 字节的段。给出客户和发送方之间交换的两个段的内容。假设响应的确认将在稍后被客户完成（忽略校验和字段的计算）。

P3-41 假设在之前问题中的 HTTP 客户关闭连接，并且同时对从服务器收到的响应中的字节进行确认。在接收到客户端的 FIN 段之后，服务器也关闭另一个方向的连接。给出连接终止阶段。

P3-42 区分超时事件和三次重复 ACK 事件。哪一个是网络中拥塞较严重的标志？为什么？

P3-43 图 3-52 给出了状态转换图中的客户和服务器，这幅状态转换图描述的是使用四次握手关闭的普通场景。改变这幅图来给出三次握手关闭。

P3-44 Eve，这位入侵者，她使用 Alice 的 IP 地址，向 Bob 这台服务器发送一个 SYN 段。Eve 能够通过假装她是 Alice 来与 Bob 建立 TCP 连接吗？假设 Bob 为每次连接使用一个不同的 ISN。

P3-45 Eve，这位入侵者，她使用 Alice 的 IP 地址，给 Bob 这台服务器发送一个用户数据报。Eve 能够假装她是 Alice 来接收来自 Bob 的响应吗？

P3-46 假设 Alice，这个客户，与 Bob 这台服务器创建了一个连接。他们交换数据并关闭连接。现在 Alice 通过发送一个新的 SYN 段开始一个与 Bob 的新连接，在 Bob 回复这个 SYN 段之前，一个来自 Alice 的旧 SYN 段的副本到达了 Bob 的计算机。这个段之前在网络中慢慢前行，它发起了来自 Bob 的 SYN + ACK。这个段会被 Alice 的计算机误认为是对新 SYN 段的响应吗？请解释。

P3-47 假设 Alice，这个客户，与 Bob 这台服务器建立了 TCP 连接。他们交换数据并关闭连接。现在 Alice 通过发送一个新的 SYN 段开始一个与 Bob 的新连接。服务器响应 SYN + ACK 段。然而，在 Bob 接收来自 Alice 段对这个连接的 ACK 之后，一个重复的旧 ACK 段从 Alice 到达 Bob 端。这个旧的 ACK 会与 Bob 期待从 Alice 端接收的 ACK 混淆吗？

P3-48 使用图 3-56，请解释在 TCP 的发送端如何实现流量控制（从发送 TCP 到发送应用程序）。请画图。

P3-49 使用图 3-56，请解释在 TCP 的接收端如何实现流量控制（从发送 TCP 到发送应用程序）。

P3-50 在 TCP 中，假设一个客户有 100 字节要发送。客户每 10ms 创建 10 字节并将它们传递到应用层。服务器立即确认每个段或计时器在 50ms 超时服务器确认每个段。如果实现使用最大段长度（MSS）是 30 字节的 Nagle 算法，给出段和每个段携带的字节。往返时间是 20ms，但是发送方计时器被设置成 100ms。有携带最大长度的段吗？Nagle 算法真的有效吗？为什么？

P3-51 为了更好地看清 Nagle 算法，让我们重复之前的问题，但是当之前的段（每次一个段）没有被确认或计时器在 60ms 后超时，我们让服务器传输层确认段。给出这个场景的时间线。

P3-52 正如我们在文中解释的，当不使用新 SACK 选项时，TCP 滑动窗口是回退 N 帧和选择性重复协议的组合。请解释 TCP 滑动窗口中哪个方面与回退 N 帧接近，哪个方面与选择性重复协议接近？

P3-53 尽管新 TCP 实现使用 SACK 选项来报告字节的失序和重复范围，但是请解释旧实现如何表示接收段中失序的或重复的字节。

P3-54 我们在本书网站讨论 TCP 的 SACK 新选项，但是暂时假设我们添加了一个 8 字节 NAK 选项到 TCP 段的尾部，TCP 段可以支持 32 位序号。请给出我们如何使用这个 8 字节 NAK 来定义接收到的字节中失序或重复范围。

P3-55 在 TCP 连接中，假设最大段长度（MSS）是 1000 字节。客户进程有 5400 字节要发送到服务器进程。没有字节要响应（单向通信）。TCP 服务器根据我们在文中讨论的规则产生 ACK。给出慢启动阶段事务的时间线，给出开始、结束以及每次变化后的 cwnd 值。假设每个段头部只有 20 字节。

P3-56 Taho TCP 站的 ssthresh 值被设置为 6MSS。现在站处于慢启动状态，cwnd = 4MSS。给出在如下事件

之前和之后的 cwnd，ssthresh 的数值以及站状态。这些事件是：四个连续非重复 ACK 到达之后是一个超时；四个连续非重复 ACK 到达之后是三次非重复 ACK。

P3-57 Reno TCP 站的 ssthresh 值被设置为 8MSS。现在站处于慢启动状态，cwnd = 5MSS 且 ssthresh = 8 MSS。给出在如下事件之后的 cwnd，ssthresh 的数值以及站的当前和下一个状态。这些事件是：三个连续非重复 ACK 到达之后是五个重复 ACK；三个连续非重复 ACK 到达之后是两个非重复 ACK；三个连续非重复 ACK 到达之后是一个超时。

P3-58 在 TCP 连接中，窗口大小在 60 000 字节和 30 000 字节之间波动。如果平均 RTT 是 30ms，连接的存止量是多少？

P3-59 如果初始 RTT_S = 14ms 且α被设置为 0.2，在如下事件之后计算新 RTT_S（对应与事件 1 相关）：

事件 1：00 ms　段 1 被发送。
事件 2：06 ms　段 2 被发送。
事件 3：16 ms　段 1 超时且被重发。
事件 4：21 ms　段 1 被确认。
事件 5：23 ms　段 2 被确认。

P3-60 在连接中，假设旧 RTT_S = 7ms。如果新 RTT_S = 17 且新 RTT_M = 20，新的 RTT_S 是多少？令β= 0.25。

3.7 模拟实验

Applets

我们构建了一些 Java 小程序用于说示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在本章，我们使用 Wireshark 来捕捉并调查一些传输层分组。我们使用 Wireshark 来模拟两个协议：UDP 和 TCP。

Lab3-1 在第 1 个实验中，我们使用 Wireshark 来捕捉运行中的 UDP 分组。我们检查每个字段的值并检查是否校验和被计算。

Lab3-2 在第 2 个实验中，我们使用 Wireshark 来捕捉并仔细研究 TCP 分组中的很多特性。这些特性包括可靠性、拥塞控制以及流量控制。Wireshak 让我们看到 TCP 如何使用段中的序号和确认号来实现可靠数据传输。我们也能观察运行中的 TCP 拥塞算法（慢启动，拥塞避免以及快速恢复）。另一个 TCP 的特性是流量控制。我们观察 TCP 中钻着从接收方到发送方方向的流量控制是通过接收方通告的 cwnd 数值实现的。

3.8 编程作业

利用你选择的编程语言，编写源代码，编译并测试如下程序：

Prg3-1 编写程序来模拟位于发送端的简单协议的有限状态机（见图 3-18）。

Prg3-2 编写程序来模拟位于发送端的停止-等待协议的有限状态机（见图 3-21）。

Prg3-3 编写程序来模拟位于发送端的回退 N 帧协议的有限状态机（见图 3-27）。

Prg3-4 编写程序来模拟位于发送端的选择性重复协议的有限状态机（见图 3-34）。

第4章

Computer Networks: A Top-Down Approach

网 络 层

TCP/IP 协议簇中的网络层负责主机到主机之间的报文传递。这包括很多事件，如本地地址的设计，本地地址唯一定义任意一台连到因特网的主机。网络层也需要一些路由协议来帮助网络层分组找到从源端到目的端的路径。在本章，我们处理所有这些问题并把它们联系起来理解网络层。

- 4.1 节介绍网络层的一般概念和重要问题。我们首先讨论网络层可以提供的服务，包括分组、路由以及转发。之后，我们介绍网络层中使用的分组交换。我们也讨论网络层性能、网络层拥塞以及路由器结构。
- 4.2 节专注于 TCP/IP 协议簇中的网络层，并讨论一些用于第四版的协议，这包括因特网协议第四版（IPv4）以及因特网控制报文协议第四版（ICMPv4）。我们也讨论 IPv4 寻址及相关问题。
- 4.3 节专注于单播路由和单播路由协议。我们给出域内和域间的路由协议是如何在当今的因特网中应用的。
- 4.4 节讨论多播和多播路由协议。并给出域内单播路由协议是如何扩展从而用作多播路由协议。我们也讨论一个新的独立多播协议并简要介绍域间多播路由协议。
- 4.5 节介绍新一代网络层协议，IPv6 和 ICMPv6 以及这一代协议所使用的寻址方法。我们将这些协议的讨论放在最后一部分，从而允许读者跳过它们而不会影响其余的内容。我们相信这些协议的完全实现仍然需要若干年。

4.1 介绍

图 4-1 给出 Alice 和 Bob 之间网络层的通信。这与我们第 2 章应用层和第 3 章传输层中的场景是相同的。

图 4-1 表示因特网由很多通过连接设备连接起来的网络（或链路）组成。换言之，因特网是互联网，多个 LAN 和 WAN 的组合。为了更好地理解网络层的作用（或互联网层），我们需要考虑那些连接了多个 LAN 和多个 WAN 的连接设备（路由器或交换机）。

如图 4-1 所示，网络层与源主机、目的主机和路径上的所有路由器（R2，R4，R5 和 R7）有关联。在源主机（Alice）端，网络层从传输层接收分组，将分组封装进数据报并传递分组到数据链路层。在目的主机（Bob）端，数据报被解封，分组被抽出并被传递到相应的传输层。尽管源主机和目的主机与 TCP/IP 协议簇的所有五层有关，如果路由器只路由分组，那么它使用三层；然而，它们可能需要传输和应用层用来控制。路径中的路由器通常用两个数据链路层和两个物理层表示，因为它从一个网络接收分组并传递到另一个网络。

4.1.1 网络层服务

在讨论当今因特网的网络层之前，让我们简要讨论一下网络层服务，通常网络层协议期待这些服务。

分组

网络层的首要责任一定是分组：在源端将负载（从上层接收的数据）封装进网络层分组并且在目的端从网络层分组中解封负载。换言之，网络层的一个责任是从源端向目的端携带数据而不改变

或使用它。网络层就像邮局一样提供携带服务，邮局负责从发送者向接收者传递分组而不改变或使用里面的内容。

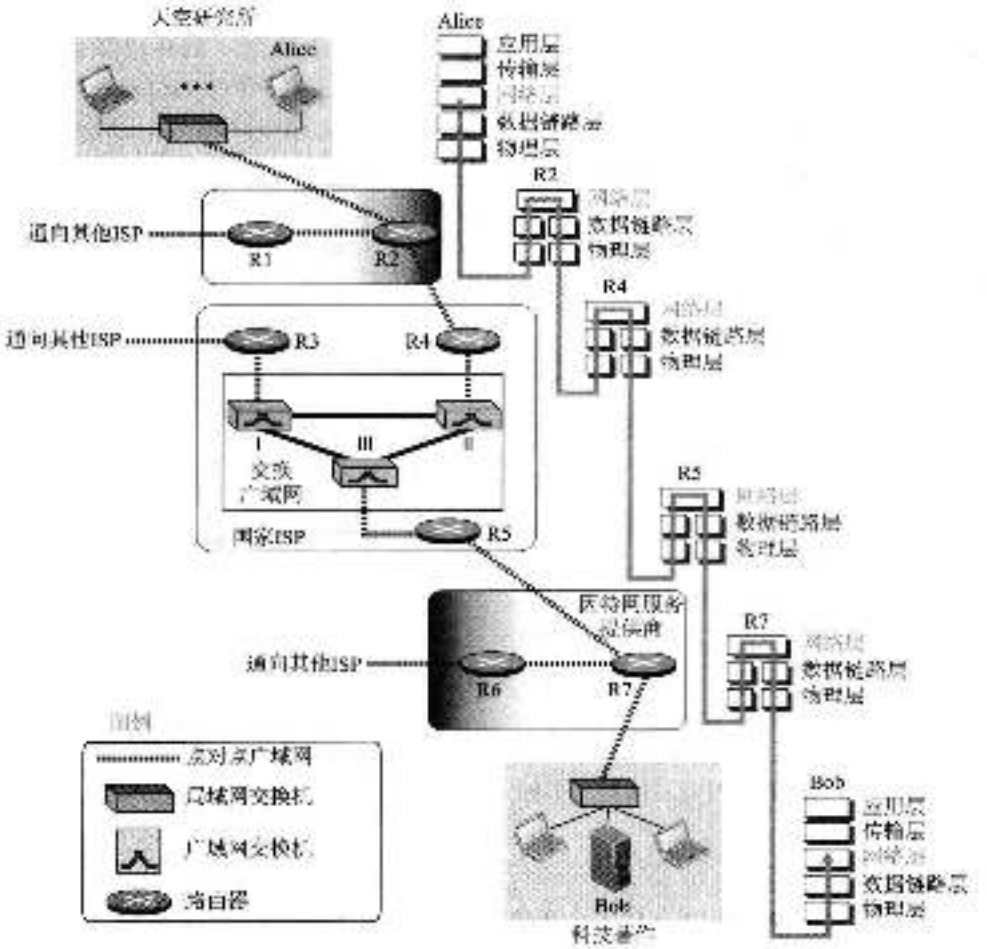


图 4-1 网络层通信

源主机从上层协议接收分组，加入一个头部，头部包含源和目的地址以及其他网络层协议（后文讨论）所需要的信息，并且将其传递到数据链路层。源端不可以改变负载的内容，除非负载过大不能传递才需要分段。

目的主机从数据链路层接收网络层分组，解封分组并将负载传递到相应的上层协议。如果分组在源端或在沿途的路由器被分段，那么网络层负责等待，直到所有分组都到达，它同时负责重组分组并将它们传递到上层协议。

路径上的路由器不可以解封它们接收到的分组，除非它们接收到的分组需要被分段。也不允许路由器改变源端和目的端地址。路由器仅仅可以检查地址，从而将分组转发到路径中的下一个网络。然而，如果分组被分段，分组的头部需要被复制给所有分段并且做一些改变，稍后我们会详细讨论。

路由

网络层的另一个责任与第一个责任同等重要，那就是路由。网络层负责将分组从源端路由到目的端。一个物理网络（多个 LAN 和多个 WAN）是网络和连接它们的路由器的组合。这意味着，从源端到目的端有多个路由器。网络层负责寻找最佳路由。网络层需要一些特定策略来定义最好的路由。在

当今的因特网，通过运行某些路由协议（routing protocol）来帮助路由器协调它们关于邻居的知识，并提出当分组到达时使用的一致性表格。我们在本章稍后讨论路由协议，它应该在通信开始之前运行。

转发

如果路由应用一些策略并运行某些路由协议来为每个路由器创建决策表，那么转发（forwarding）可以定义为当分组到达路由器的一个端口时，路由器所采用的行为。决策表有时候称为转发表（forwarding table），有时称为路由表（routing table），路由器通常用它来实施行为。当路由器从它连接的网络接收到一个分组时，它需要将分组转发到另一个所连接的网络上（在单播路由中）或者转发到多个自己所连接到的网络上（在多播路由中）。为了做出这项决策，路由器使用分组头部的一个信息片段来找到转发表中相应的输出接口号，这个信息片可能是目的地址或标签。图4-2给出了路由器中转发表的思想。

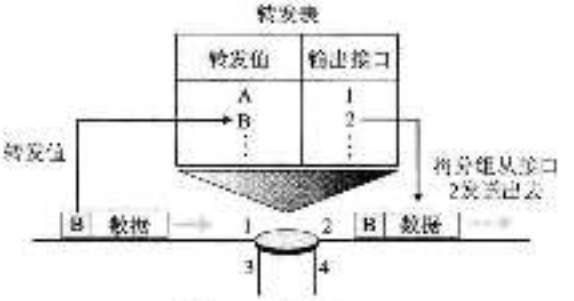


图4-2 转发过程

差错控制

在第3章，我们讨论过传输层的差错控制机制。在第5章我们将讨论数据链路层的差错控制机制。尽管差错控制也可以在网络层实现，但是因特网中网络层的设计者忽略网络层携带的数据中的差错问题。这项决定的一个原因是网络层中的分组可能在每个路由器被分段，这使得这层的差错检测效率很低。

然而，网络层的设计者将校验和字段加到了数据报中来控制头部的损坏而不是控制整个数据报的损坏。这个校验和可能防止两跳之间以及端到端之间的数据报头部的改变或损坏。

我们需要提及的是，尽管因特网中的网络层并没有直接提供差错控制，但是如果数据报被丢弃或在头部中含有未知信息，因特网使用一个辅助协议ICMP来提供某种差错控制。我们在本章稍后讨论ICMP。

流量控制

流量控制规定了源端可以发送但不会淹没接收方的数据量。如果源端计算机上层产生数据的速度比目的端计算机上层消耗数据的速度快，那么接收方会被数据所淹没。为了控制数据流量，接收方需要发送一些反馈给发送方，从而通知后者接收方被数据淹没了。

然而，因特网中的网络层并没有直接提供任何流量控制。当准备好时，数据报被发送方发送，而不考虑接收方是否准备好了。

以下是网络层设计中缺乏流量控制的一些原因。第一，由于这一层没有差错控制，接收端网络层的工作是很简单的，以至于它很少被淹没。第二，当准备好时，使用网络层服务的上层可以实现缓冲区，用它来从网络层接收数据，而无需以接收数据的速度消耗数据。第三，流量控制被提供给绝大多数使用网络层服务的上层协议，因此另一级流量控制使得网络层更复杂且整个系统更低效。

拥塞控制

网络层协议中的另一个事件是拥塞控制。网络层中的拥塞是因特网中某一个区域出现过多数据报的情况。如果源端计算机发送的数据报数量超过了网络或路由器的容量，那么就可能发生拥塞。在这种情况下，一些路由器可能丢弃一些数据报。然而，随着更多的数据报被丢弃，这种情况可能变得更糟糕，这是因为由于上层的差错控制机制，发送方可能发送丢失分组的副本。如果拥塞继续，那么有时这种情况可能导致系统崩溃且没有数据报被传递。

服务质量

随着因特网的发展，它已经产生了如多媒体通信（尤其是视频和音频的实时通信）之类的新应用，通信的服务质量（QoS）已经变得越来越重要。因特网通过提供更好的服务质量支持这些服务实现了自身的连续发展。然而，为了保证网络层不变，这些服务绝大多数在上层实现。由于 QoS 在我们使用多媒体通信时表现最为明显，我们将在第 8 章讨论它。

安全

另一个与网络层通信相关的事件是安全。最初设计因特网时没有考虑安全，因为它只被大学中的一小部分人用于研究活动；其他人不能访问因特网。网络层被设计出来是不提供安全的。然而，如今安全是一个大问题。为了给无连接的网络层提供安全，我们需要另一个虚拟层将无连接服务变成面向连接服务。这个虚拟层称为 IPSec，我们将在第 10 章讨论。

4.1.2 分组交换

从前面的路由和转发讨论中，我们可以推断出交换（switching）发生在网络层。事实上，路由器是一个创建了输入端口和输出端口（或一组输出端口）连接的交换机，就像将输入连接到输出让电流流动的开关。

尽管数据通信交换技术被分为两大类：电路交换和分组交换，但是，网络层只使用分组交换，因为这一层的数据单位是分组。电路交换绝大多数用于物理层；之前提及的电子开关是一种电路交换。

在网络层，来自上层的报文被分割成可管理的分组，每个分组被从网络中发送。报文的源端一个接一个地发送报文；报文的目的端一个接一个地接收分组。目的端在将报文传递到上层之前，等待所有属于同一个报文的分组到达。分组交换网络的连接装置仍然需要决定如何将分组路由到最终的目的端。如今，分组交换网络可以使用两个不同的方法来路由分组：数据报方法（datagram approach）和虚电路方法（virtual circuit approach）。我们接下来讨论这两种方法。

数据报方法：无连接服务

当因特网刚刚开始发展时，为了使其简单，网络层被设计成提供无连接服务，其中网络层协议独立对待每个分组，每个分组和其他分组没有关系。这个思想就是网络层只负责将分组从源端传递到目的端。在这种方法中，报文中的分组可能会也可能不会通过同一条通向目的端的路径。图 4-3 展示了这种思想。

当网络层提供无连接服务时，因特网中每个正在传送的分组都是一个独立的实体；属于同一个报文的多个分组之间没有联系。这种网络中的交换机称为路由器。属于同一个报文的分组可能后面跟着属于同一个报文的分组，也可能跟着一个属于不同报文的分组。一个分组可能后面跟着来自相同或不同源端的分组。

每个分组都基于其头部信息进行路由的：源地址和目的地址。目的地址定义了它应该去哪里；源地址定义了它从哪里来。在这种情况下的路由器仅仅基于目的地址路由分组。如果分组被丢弃，那么源地址可用来发送错误报文。图 4-4 给出此种情况下路由器中的转发过程。我们使用像 A 和 B 这样的符号地址。

在数据报方法中，转发决策基于分组的目的地址。

虚电路方法：面向连接服务

在面向连接服务（也称作虚电路方法）中，属于同一报文的所有分组之间存在联系。在报文中的所有数据报被发送之前，应该建立虚连接从而定义数据报的路径。在连接建立之后，数据报可以沿着相同的路径发送。在这种类型服务中，分组不仅必须包含源和目的地址，也必须包含流标号，这是一个虚电路标识符，它定义了分组应该经过的路径。很快，我们将给出这个流标号是如何确定

的，但目前，我们假设这个分组携带这个标签。尽管看起来标签的使用好像使得源和目的地址在数据传输阶段变得不必要，但是网络层中因特网的很多部分仍然记录这些地址。一个原因是一部分分组路径可能仍然使用无连接服务。另一个原因是网络层的协议被设计成带有这些地址，而且在改变之前要花一些时间。图 4-5 给出面向连接服务的概念。

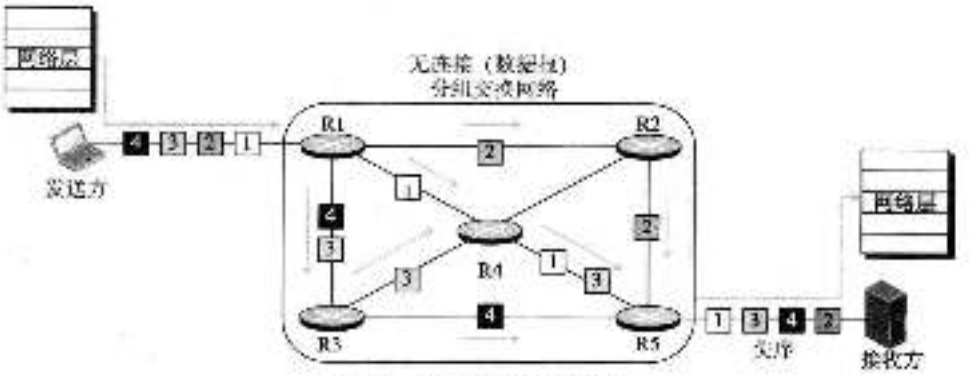


图 4-3 无连接分组交换网络

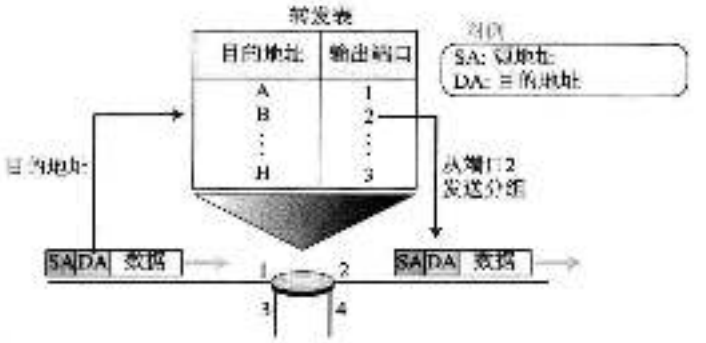


图 4-4 用于无连接网络的路由器中的转发过程

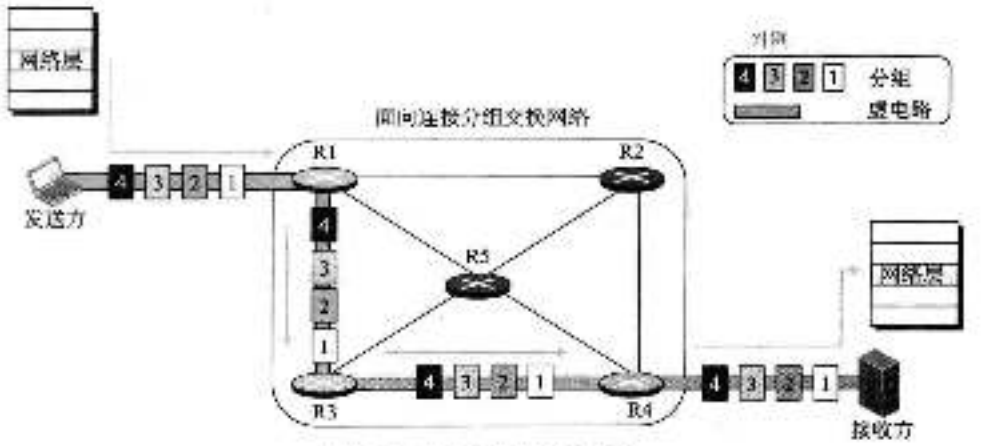


图 4-5 虚电路分组交换网络

每个分组基于分组中的标签被转发。为了研究因特网中使用的面向连接设计的思想，我们假设当分组到达路由器时，它有一个标签。图 4-6 展示了这种思想。在这种情况下，转发决策基于标签的值或基于虚电路标识符，标签有时也称为虚电路标识符。

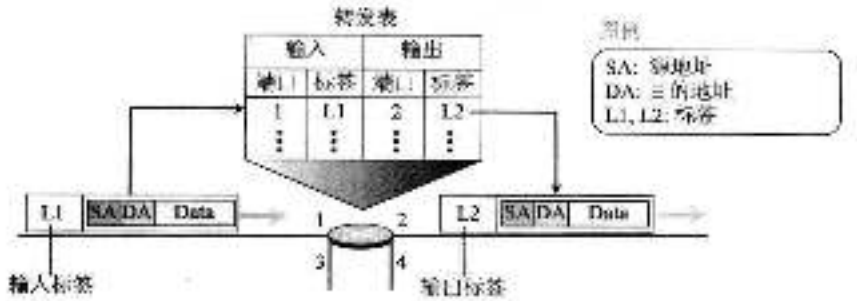


图 4-6 用于虚电路网络的路由器中的转发过程

为了创建面向连接服务，使用一个三阶段过程：建立、数据交换和拆除。在建立阶段，发送方和接收方的源和目的地址用于创建面向连接服务中的表格项。在拆除阶段，源和目的通知路由器删除相应的表格项。在这两个阶段之间发生数据传输。

在虚电路方法中，转发决策基于分组的标签。

建立阶段

在建立阶段，路由器为虚电路创建了一个表格项。例如，假设源 A 需要创建到目的 B 的虚拟电路。两个辅助分组需要在发送方和接收方之间交换：请求分组和确认分组。

请求分组 一个请求分组被从源端发送到目的端。这个辅助分组携带源和目的地址。图 4-7 给出这个过程。

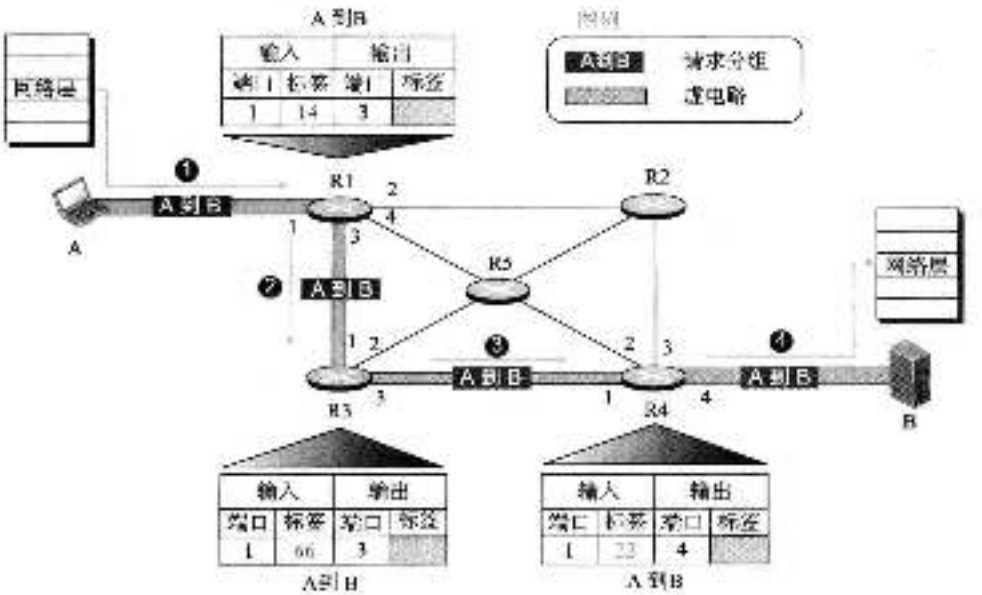


图 4-7 在虚电路网络中发送请求分组

1. 源 A 发送一个请求分组到路由器 R1。

2. 路由器 R1 接收请求分组。它知道从 A 到 B 的分组通过端口 3 发送出去。路由器如何获得这个信息我们稍后再解释。目前，假设它知道端口。路由器为虚电路创建表格中的一个项，但是它只能够填充四个列中的三个。路由器分配输入端口（1）并选择可用的输入标签（14）以及输出端口（3）。

它还不知道输出标签，这在确认步骤将会找到。之后，路由器通过端口 3 将分组转发到路由器 R3。

3. 路由器 R3 接收到建立请求分组。路由器 R1 处也发生了相同的事情；表格的三列完成；在这种情况下是，输入端口（1）、输入标签（66）以及输出端口（3）。

4. 路由器 R4 接收到建立请求分组。又有三列被完成：输入端口（1），输入标签（22）以及输出端口（4）。

5. 目的端 B 接收到建立分组，而且如果它准备好从 A 接收分组，那么它将标签分配给来自 A 的输入分组，此时为 77，如图 4-8 所示。这个标签使目的端知道分组来自 A，而不是来自其他源端。

确认分组　一种特殊的分组，称为确认分组，它完成了交换表中的表格项。图 4-8 展示了这个过程。

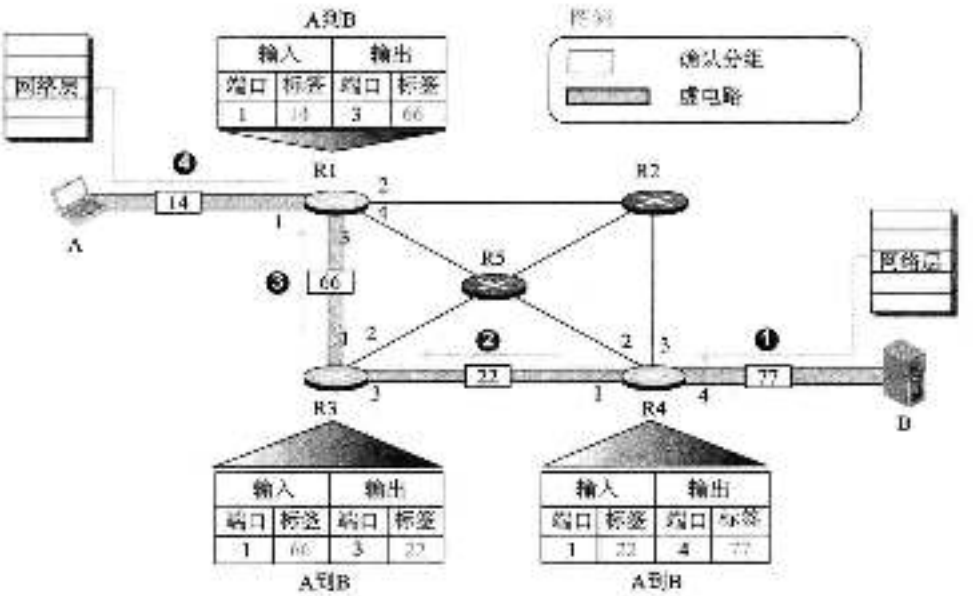


图 4-8　在虚电路网络中发送确认

1. 目的端发送一个确认到路由器 R4，确认携带全局源和目的地址，因此路由器知道表格中的哪个项将被填充完成。分组也携带标签 77，这是目的端为来自 A 的分组选择的，这个标签作为输入标签。路由器 R4 使用这个标签来完成表格项中的输出标签列。注意，77 是目的端 B 的输入标签，而对于 R4 是输出标签。

2. 路由器 R4 发送一个确认到路由器 R3，这个分组包含表格中的输入标签，这是建立阶段选择出来的。路由器 R3 将它用作表格中的输出标签。

3. 路由器 R3 发送一个确认到路由器 R1，这个分组包含表格中的输入标签，这是建立阶段选择出来的。路由器 R1 将它用作表格中的输出标签。

4. 最终，路由器 R1 发送一个确认到源端 A。这个分组包含表格中的输入标签，这是建立阶段选择出来的。

5. 源端将其用作向目的端发送的数据分组的输出标签。

数据传输阶段

第二个阶段称为数据传输阶段。在所有路由器为特定虚电路创建好它们的转发表之后，属于同一个报文的网络层分组可以被一个接一个地发送。在图 4-9 中，我们展示了一个分组的流动情况。但是对于 1、2 或 100 分组来说过程是相同的。源端计算机使用标签 14，这是在建立阶段从路由器 R1 接收来的。路由器 R1 将分组转发到路由器 R3，但是将标签改为 66。路由器 R3 将分组转发到

路由器 R4，但是将标签改为 22。最终，路由器 R4 将分组传递到它最终的目的端，同时将标签设为 77。报文中所有的分组都是按照相同的标签序列，并且分组是有序到达目的端的。

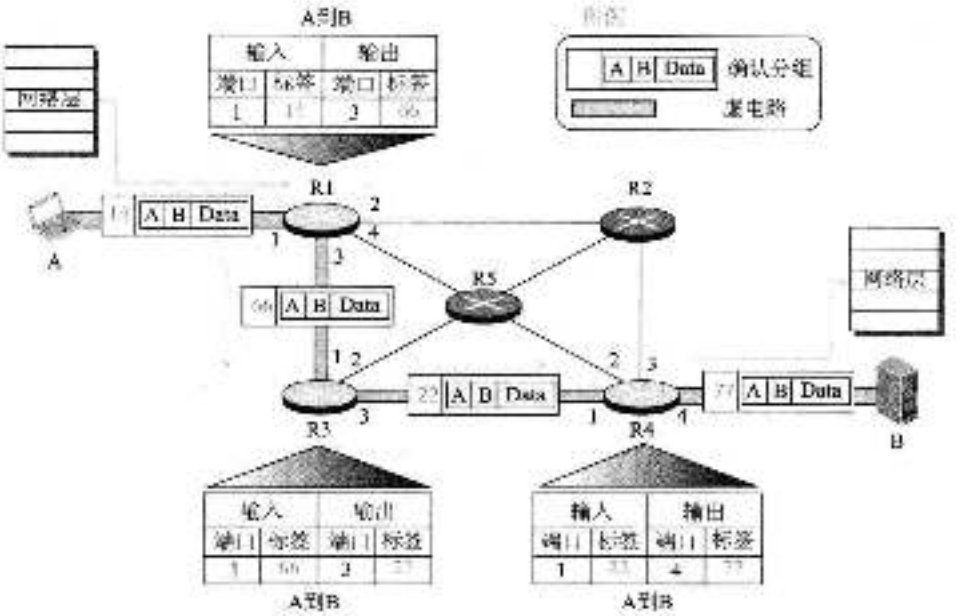


图 4-9 在已建立的虚电路网络中的一个分组的流动情况

拆除阶段

在拆除阶段，在所有分组发送到 B 之后，源 A 发送一个称为拆除分组的特殊分组。目的 B 以确认分组回复。所有路由器从它们的表格中删除相应的项。

4.1.3 网络层性能

使用网络层协议服务的上层协议期待收到理想的服务，但是网络层不是完美的。网络的性能可以通过延迟（delay）、吞吐量（throughput）和分组丢失（packet loss）来度量。在我们讨论它们对于性能的影响之前，首先定义分组交换网络中的这三个术语。

延迟

我们都期待从网络中立即得到响应，但是从源到目的的分组要遭遇延迟。网络中的延迟可以分为四类：发送延迟、传播延迟、处理延迟以及排队延迟。让我们首先来讨论这些延迟类型，之后给出如何计算从源到目的的分组延迟。

发送延迟

一台源主机或路由器不能立即发送分组。发送方需要将分组中的位一个接一个地发到线上。如果分组的第一个位在时间 t_1 发送，最后一个位在时间 t_2 发送，分组的发送延迟就是 $(t_2 - t_1)$。分组越长发送延迟就越长，并且如果发送方更快地发送分组，那么发送延迟就更短。换言之，发送延迟是

$$\text{Delay}_{tr} = (\text{分组长度}) / (\text{发送速度})。$$

比如，在快速以太网 LAN（见第 5 章）中，发送速度是 100Mbps 且分组是 10 000 位，那么它要花(10 000)/(100 000 000)或 100 微秒来将分组中的所有位发送到线上。

传播延迟

传播延迟是在传输介质中一个位从 A 点到 B 点所消耗的时间。分组交换网络的传播延迟取决于每个网络（LAN 或 WAN）的传播延迟。传播延迟取决于介质的传输速度，真空中的速度是 3×10^8

米/秒，并且这个速度通常远远小于有线介质中的速度；传播延迟也取决于链路的距离。换言之，传播延迟是

$$\text{Delay}_{pg} = (\text{距离}) / (\text{传播速度})。$$

例如，如果点对点 WAN 中有线链路的距离是 2000 米，并且线路中信号的传播速度是 2×10^8 米/秒，这样传播延迟是 10 微秒。

处理延迟

处理延迟是路由器或目的主机从它的输入端接收分组、去除头部、执行差错检测程序并将分组传递到输出端口（在路由器情况下）或将分组传送到上层协议（在目的主机情况下）所需要的时间。处理延迟可能对于每个分组不同，但是通常都是计算平均数。

$$\text{Delay}_{pr} = \text{在路由器或目的主机中处理分组所需的时间}$$

排队延迟

排队延迟通常在路由器中发生。如我们在后面讨论的，路由器有一个输入队列，它被连接到路由器的每一个输入端口上，用来存储等待被处理的分组；路由器也有输出队列，它被连接到路由器的每一个输出端口上，用来存储等待被发送的分组。路由器中分组的排队延迟通过分组在输入和输出队列中的等待时间来度量，我们可以将这种情景与繁忙的机场进行比较。一些飞机需要等待降落带（输入延迟）；一些飞机可能需要等待离开带（输出延迟）。

$$\text{Delay}_{qu} = \text{在路由器中输入和输出队列中分组的等待时间}$$

总延迟

假设发送方、路由器和接收端有相同的延迟，如果我们知道整个路径上的路由器的数量，n，那么分组的总延迟（源到目的延迟）可以计算。

$$\text{总延迟} = (n+1)(\text{Delay}_{tr} + \text{Delay}_{pg} + \text{Delay}_{pr}) + (n)(\text{Delay}_{qu})$$

注意，如果我们有 n 个路由器，我们就有（$n+1$）条链路。因此，我们有与 n 个路由器和源端相关的（$n+1$）个发送延迟，与（$n+1$）条链路相关的（$n+1$）个传输延迟，与 n 个路由器和目的端相关的（$n+1$）个处理延迟，与 n 个路由器相关的 n 个排队延迟。

吞吐量

网络中任意点的吞吐量被定义为一秒内通过这个点的位的数量，事实上是那一点的数据发送速率。在从源到目的端的一条路径中，一个分组可能穿过多条链路（网络），每条链路都有不同的发送速率。那么，我们如何确定整个路径的吞吐量？为了看清这种情况，假设我们有三条链路，每一条都有不同发送速率，如图 4-10 所示。

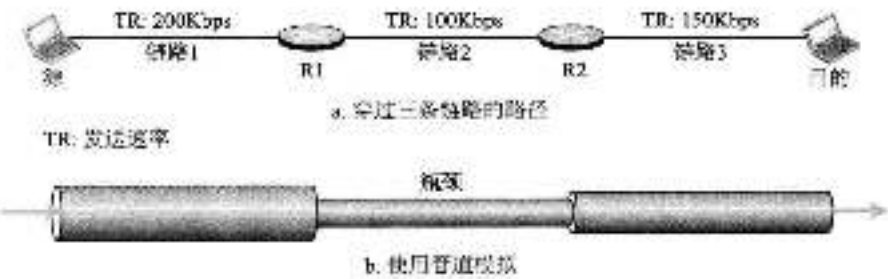


图 4-10 含有三条连续链路的路径的吞吐量

在图 4-10 中，数据可以在链路 1 中以 200kbps 的速度流动。然而，当数据到达路由器 R1 时，它不能以这个速率通过。数据需要在路由器排队并以 100kbps 的速度发送。当数据到达路由器 R2，它能以速度 150kbps 的速度发送，但是没有足够的数据要发送。换言之，在链路 3 的平均数据流也

是 100kbps。我们可以得出结论，这条路径的平均数据速度是 100kbps，这是二个不同数据速率中的最小值。这幅图也表示我们可以用不同大小的管道来模拟每条链路的行为；平均吞吐量由瓶颈决定，即直径最小的管道部分。一般地，在有 n 条连续链路的路径上，我们有

$$\text{吞吐量} = \text{最小值}\{TR_1, TR_2, \cdots TR_n\}。$$

尽管图 4-10 中的情况给出了当数据穿过若干链路时如何计算吞吐量，但是因特网中的实际情况是，数据通常要穿过两个接入网以及骨干网，如图 4-11 所示。

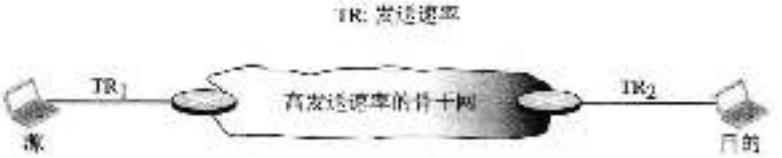


图 4-11 通过骨干网的路径

骨干网有很高的发送速率，大约为每秒十亿位。这意味着吞吐量通常定义为两个接入链路的最小发送速率，这两个接入链路将源和目的连接到骨干网。图 4-11 给出了这种情况，其中吞吐量是 TR_1 和 TR_2 的最小值。例如，如果快速以太网 LAN 的数据速率是 100Mbps，且一台服务器通过它连接到因特网上，但是想要下载文件的用户通过拨号电话线连接到因特网，拨号电话线的速率是 40kbps，那么吞吐量是 40kbps。瓶颈肯定是拨号线路。

我们需要提及另一种考虑吞吐量的情况。两个路由器之间的链路并不总是一个流专用的。路由器可能从若干个源收集流或在若干源分配流。在这种情况下，两个路由器之间链路的发送速率实际上被这些流共享，并且当我们计算吞吐量时，需要考虑到这些。例如，在图 4-12 中，计算吞吐量时主链路的发送速率只有 200kbps，因为链路被三条路径所共享。



图 4-12 共享链路中吞吐量的影响

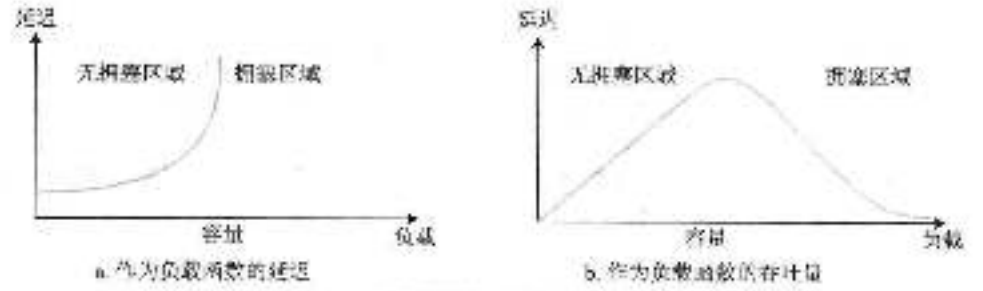
分组丢失

另一个严重影响通信性能的问题是发送期间丢失的分组数量。当路由器处理另一个分组的同时接收到一个分组，那么接收到的分组需要被存储在输入缓冲区等待轮次。然而，一个路由器的缓冲区大小有限。会有一个时刻缓冲区满，且下一个分组需要被丢弃。分组丢失会导致因特网网络层的分组被重发，这反过来可能造成溢出并导致更多的分组丢失。在排队论中已经做过很多理论研究来防止队列溢出并防止分组丢失。

4.1.4 网络层拥塞

在第 3 章，我们讨论过传输层拥塞。尽管网络层的拥塞在因特网模型中没有详述，但是对这一层拥塞的学习可以帮助我们更好地理解传输层拥塞的原因并找到网络层可用的解决方法。网络层的拥塞与两个问题有关：吞吐量和延迟，这一点我们在前一节讨论过。

图 4-13 显示了度量吞吐量和延迟的负载函数



a. 作为负载函数的延迟　　b. 作为负载函数的吞吐量

图 4-13　分组延迟和吞吐量的负载函数

当负载远远小于网络容量时，延迟是最小的。最小的延迟由传播延迟和处理延迟组成，这两者都可忽略。然而，当负载大小达到网络容量时，延迟急剧增加。因为我们现在需要将排队延迟加到总延迟当中。注意，当负载大于网络容量时延迟为无限大。

当负载在网络容量之下时，吞吐量随着负载的大小而增加。我们希望在负载到达容量之后负载保持不变，但是吞吐量会急剧下降。原因在于路由器丢弃分组。当负载超过网络容量时，队列变满且路由器必须丢弃一些分组。丢弃分组不会减少网络中的分组数量，因为当分组没有到达目的端时，源端使用超时机制重发分组。

拥塞控制

正如我们在第 3 章讨论的，拥塞控制指的是在其发生前防止拥塞现象或在其发生后消除拥塞的技术和机制。一般地，我们将拥塞控制机制分为两大类：**开环拥塞控制**（预防）和**闭环拥塞控制**（消除）。

开环拥塞控制

开环拥塞控制中，在发生拥塞之前，应用某种策略来预防拥塞现象的发生。在这些机制中，源端或目的端都可以处理拥塞控制。我们给出预防拥塞策略的简要说明。

重传策略　有时重传是不可避免的。如果发送方认为一个发送分组丢失或损坏，则该分组就需要重发。重传一般在网络上会增加拥塞现象。然而，一个好的重传策略可以预防拥塞。必须优化设计重传策略和重传定时器，使之具有高效率，并用来预防拥塞。

窗口策略　发送方窗口的类型会影响拥塞。对拥塞控制而言，选择性重复窗口要优于回退 N 帧窗口。在回退 N 帧窗口中当一个分组的计时器超时，可以重发多个分组。虽然有一些分组可以安全完整地到达接收方，但是这种重复会使得拥塞更加严重。而选择性重复窗口试图发送那些丢弃或损坏的特定分组。

确认策略　接收方使用的确认策略也可能影响拥塞。如果接收方并不对它所接收的每一个分组进行确认，则它会使发送方放慢发送速度，从而有助于预防拥塞。很多方法用于这种情况。如果接收方有一个要发送的分组或一个特定的计时器到时，接收方才发送一个确认。接收端可以决定每次对 N 个分组仅发送一个确认。我们需要知道确认也是网络负载的一部分，发送确认越少意味着网络的负载越轻。

丢弃策略　路由器使用好的丢弃策略可以预防拥塞，同时不破坏传输的完整性。例如，在音频传输中，如果在可能发生拥塞时丢弃那些不敏感的分组，则仍可保证声音的质量，并且还能预防或减轻拥塞现象的发生。

许可策略　在虚电路网络中，许可策略是一种服务质量机制（在第 8 章讨论），它也能预防拥塞。在允许数据流进网络之前，与数据流通路相连的交换机首先检查其资源需求。如果网络有拥塞或可能出现拥塞，路由器将拒绝建立虚电路。

闭环拥塞控制

在拥塞发生之后，采用闭环拥塞控制可以缓解拥塞状况。不同的协议采用了多种机制。下面对

这几种机制进行介绍。

背压 背压技术是一种控制机制。在这种技术中，一个拥塞点停止接收来自直接上行结点或一些近邻结点的数据。这会引起上行结点或一些近邻结点发生拥塞，它们依次拒绝它们的上行结点或一些近邻结点的数据。依次类推。背压是点到点拥塞控制，它从一点开始，然后传播，数据流反向到达源端。背压技术仅用于虚电路网络。在虚电路网络中，每个结点知道数据的流是来自它的上行结点。图 4-14 表示了背压的思想。

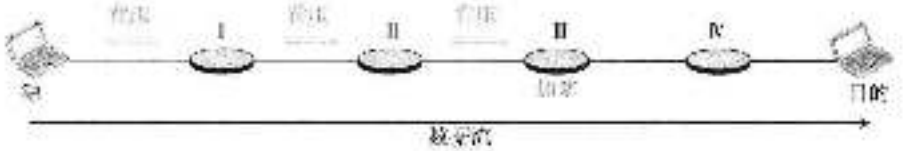


图 4-14 减轻拥塞的背压方法

结点Ⅲ接收到的输入数据超出了它的处理能力。它停止将某些分组放入输入缓冲区中，并通知结点Ⅱ降低传输速率。依次，结点Ⅱ由于降低输出数据流的速率可能会拥塞。如果结点Ⅱ拥塞，则它通知结点Ⅰ降低速率，它也可能会形成拥塞。如果是这样，则结点Ⅰ通知源端降低数据传输速率。这样可以及时减轻拥塞。注意，为了消除拥塞，对结点Ⅲ的压力反向移动到源端。

一件很重要的事情是拥塞控制只能在虚电路中实现。这个技术不能在数据报网络中实现。在这种网络中，结点（路由器）丝毫不了解上行结点。

抑制分组 它是一个分组，该分组由结点发送给源端，通知它发生拥塞的情况。注意，背压和抑制分组方法之间的不同。在背压方法中，尽管警告最后可能到达源端，但是这个警告是从一个结点传递到其上行结点，这样一步步接收警告的。而在抑制分组方法中，警告从已经发生拥塞的路由器直接传到源端，该分组经过的那些中间结点没被警告。我们在 ICMP（本章稍后讨论）中已经看到过这种控制类型。当因特网中一个路由器被大量的 IP 数据报流淹没时，它可能丢弃一些数据报，但它使用一个 ICMP 源端抑制报文通告源主机。警告报文直接发送给源站点，中间的路由器不进行任何处理。图 4-15 表示了抑制分组的思想。

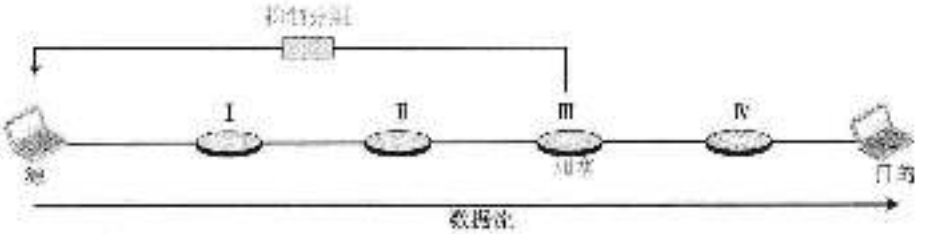


图 4-15 抑制分组

隐含信令 在隐含信令中，阻塞结点与源端之间或结点与源端之间没有通信。源端能从其他有关征兆中察觉出在网络某处有拥塞。例如，当源端发送多个分组，暂时没有收到确认时，一种设想是网络发生了拥塞。在接收确认过程中发生的延迟现象就可以被认为是网络发生了拥塞；源端应该降低发送速率。在第 3 章讨论 TCP 拥塞控制时，我们看到了这种类型的指令。

显式信令 发生拥塞的结点能发送一种显式信令通知源端或目的端发生了拥塞。但是，显式信令方法与抑制分组方法是不同的。在抑制分组方法中，有一个单独的分组用于实现此目的；而在显式信令方法中，信号包含在携带数据的分组中。显式信令可以沿着向前的方向也可以沿着向后的方向。这种拥塞控制可以在 ATM 网络中看到，我们将在第 5 章讨论。

4.1.5 路由器的结构

在我们对于转发和路由的讨论中，我们将路由器表示为一个黑盒，它从一个输入端口（接口）接收输入分组，使用转发表来找到分组从哪个输出端口离开，并且从这个输出端口发出分组。在这一节，我们打开这个黑盒看看里面是什么。然而，我们的讨论不会很细化；有很多书整本地论述路由器。这里我们只是给出路由器的概述。

元件

我们可以说路由器有四个元件：输入端口（input port）、输出端口（output port）、路由处理器（routing processor）以及交换结构（switching fabric），如图 4-16 所示。

输入端口

图 4-17 给出了输入端口的示意图。



图 4-16 路由器元件



图 4-17 输入端口

输入端口发挥路由器的物理层以及链路层功能。比特位从接收信号中获取。分组从帧中解封。检查差错，如果分组被破坏，那么它就被丢弃。之后，分组准备好被网络层处理。除了物理层处理器和链路层处理器，输入端口也有缓冲区（队列），用来在分组被发送到交换网络之前存储它们。

发送端口

发送端口发挥与输入端口相同的功能，但是顺序相反。首先输出分组要排队，每个分组被封装进一个帧，并且最终对帧使用物理层功能来创建一个信号发送到线上。图 4-18 给出输出端口的示意图。



图 4-18 输出端口

路由处理器

路由处理器发挥网络层的功能。目的地址被用来找到下一条的地址以及输出端口号。同时分组从这个输出端口号发送出去。这个行为有时称为表格查询（table lookup），因为路由处理器搜索转发表。在较新的路由器中，这个路由处理器的功能被移动到输入端口，从而辅助并加速这个过程。

交换结构

路由器中最复杂的任务就是将分组从输入队列移动到输出队列。这个任务的速度影响输入/输出队列的大小以及分组传递的总延迟。在过去，当路由器实际上是专用计算机时，计算机的内存或总线被用作交换结构。输入端口将分组存储到内存中；输出端口从内存中获取分组。如今，路由器使用多种交换结构。这里，我们简要讨论一些交换结构。

纵横制交换机 最简单的交换结构类型是纵横制交换机，如图 4-19 所示。在网络中，纵横制交换机（crossbar switch）将 n 个输入连接到 n 个输出上，它在每个纵横交叉点（crosspoint）使用微交换机。

榕树交换机 如图 4-20 所述，比纵横制交换机更注重实际的是榕树交换机（banyan switch 以榕树命名）。

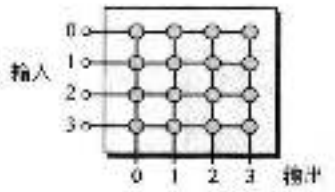


图 4-19 纵横制交换机

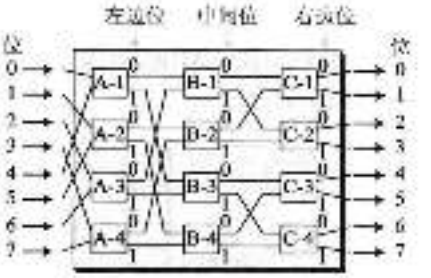


图 4-20 榕树交换机

榕树交换机是多级交换机，在每一级都有一个微开关。它基于以二进制串形式表示的输出端口来路由分组。对于 n 个输入和 n 个输出，我们有 $\log_2(n)$级，每一级有 $n/2$ 个微开关。第一级基于二进制串的最高位顺序来路由分组。第二级基于二进制串的第二高位顺序来路由分组，以此类推。图 4-21 给出八输入八输出的榕树交换机。级数是 $\log_2(8)-3$。假设一个分组到达输入端口 1 并且必须到输出端口 6（二进制为 110）。第一个微开关（A-2）基于第一位（1）路由分组；第二个微开关（B-4）基于第二位（1）路由分组；第三个微开端（C-4）基于第三位（0）路由分组。在 b 部分，分组到达端口 5 并且必须从端口 2 出去（二进制为 010）。第一个微开关（A-2）基于第一位（0）路由分组；第二个微开关（B-2）基于第二位（1）路由分组；第三个微开关（C-2）基于第三位（0）路由分组。

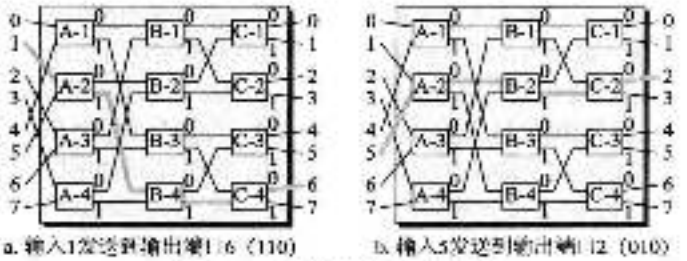


图 4-21 榕树交换机中路由举例

Batcher 榕树交换机 榕树交换机的问题是，即使两个分组不发向同一个输出端口，它们也可能产生内部冲突。我们可以通过基于分组的目的端口排序来解决这个问题。K. E. Batcher 设计了一个交换机，它位于榕树交换机之前并根据分组的最终目的来排序进入分组。这个组合叫做 **Batcher 榕树交换机**（Batcher-banyan switch）（见图 4-22）。

排序交换机使用融合了多种技术的硬件，但是我们不会讨论这些细节。通常地，另一个称为陷阱（trap）的硬件单元被加入到 Batcher 交换机和榕树交换机之间。陷阱单元防止重复分组（有相同输出目的端的分组）同时通过榕树交换机。在每个时钟滴答，每个目的端只能接收一个分组；如果多于一个，它们需要等待下一个时钟滴答。

4.2 网络层协议

在本章的第一节，我们讨论一些与网络层有关的一般问题以及服务。在这一节，我们给出网络层如何在 TCP/IP 协议簇中实现。网络层协议已经历经多个版本；在本节，我们重点论述当前版本（4）。在本章最后一节，我们简要讨论版本 6，这个版本已现端倪，但是没有完全实现。

第四版网络层可以被看做是一个主要协议以及三个辅助协议。主要协议即因特网协议第四版（IPv4）负责网络层的分组、转发以及传递。因特网控制报文协议第四版（Internet Control Message Protocol version 4，ICMPv4）帮助 IPv4 处理一些网络层传递中可能发生的错误。因特网组管理协

议（Internet Group Management Protocol，IGMP）用于帮助 IPv4 多播。在网络层地址映射到链路层地址当中，地址解析协议（ARP）用来将网络和数据链路层联合起来。图 4-23 给出这四个协议在 TCP/IP 协议簇中的位置。

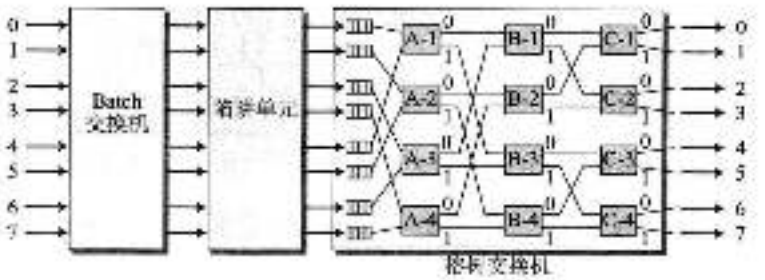


图 4-22 Batcher 蝶树交换机

我们在本节简要讨论 IPv4 和 ICMPv4。当我们讨论路由协议时，将会简要讨论 IGMP。我们将 ARP 的讨论推迟到第 5 章，那时我们将讨论链路层地址。

IPv4 是一个不可靠且无连接的数据报协议——一个尽力而为的传递服务。尽力而为（best-effort）这个词表示 IPv4 分组可能被破坏、丢失、失序到达或延迟，也可能造成网络的拥塞。如果可靠性是重要的，那么 IPv4 必须和可靠传输层协议例如 TCP 搭配。一个更容易理解的尽力而为传送的例子就是邮局。邮局尽力传送平信，但是并不总是成功。如果一个未挂号的信丢失了，这需要发送者或接收者发现丢失并更正问题。邮局自身并不记录每封信件且不通知发送者信件丢失或损坏。

IPv4 也是使用数据报的分组交换网的无连接协议。这意味着，每个数据报被独立处理，并且每个数据报可以沿着不同的路由到达目的端。这意味着被同一个源端发送到的同一个目的端的数据报可能失序到达。某些分组也可能在传输阶段丢失或被破坏。IPv4 依赖高层协议处理所有这些问题。

在本节，我们开始讨论 IPv4 提供的第一个服务，即分组。我们给出 IPv4 是如何定义分组的格式。来自上层协议或其他协议的数据被封装在分组中。我们之后讨论 IPv4 地址。最终，我们讨论 IPv4 的路由和转发以及 ICMP。

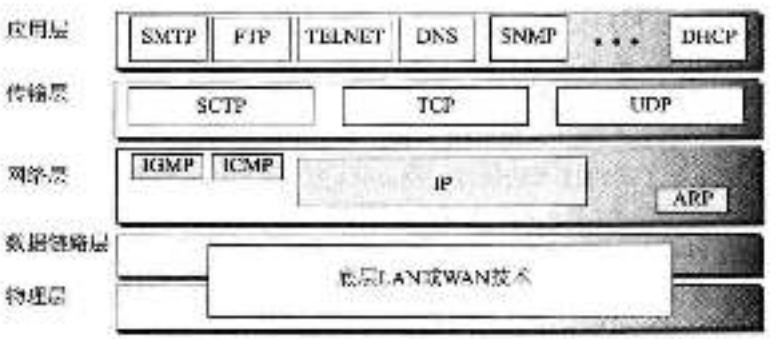


图 4-23 IP 以及其他网络层协议在 TCP/IP 协议簇中的位置

4.2.1 IPv4 数据报格式

IP 使用的分组称为数据报（datagram）。图 4-24 给出 IPv4 数据报格式。数据报是可变长分组，由两部分组成：头部和数据。头部长度可由 20 到 60 个字节组成，包含有与路由选择和传输相关的重要信息。习惯上在 TCP/IP 中都以 4 个字节表示头部。

讨论每个字段存在的意义和理由对于理解 IPv4 的运转十分重要；以下按顺序简要介绍每个字段。

- **版本号**。这4位版本号（VER）字段定义IPv4协议的版本，显然它的值为4。
- **头部长度**。这4位头部长度（HLEN）字段以4字节定义数据报头部的总长度。IPv4数据报有变长头部。当设备接收报文，它需要知道头部在何处停止且被封装在分组中的数据从何处开始。然而，为了使得头部长度值（字节数量）满足4位头部长度，头部的总长度被计算为4字节的字。总长度被4除，且结果值被插入字段。接收端需要将这个字段的数值乘以4来得到总长度。

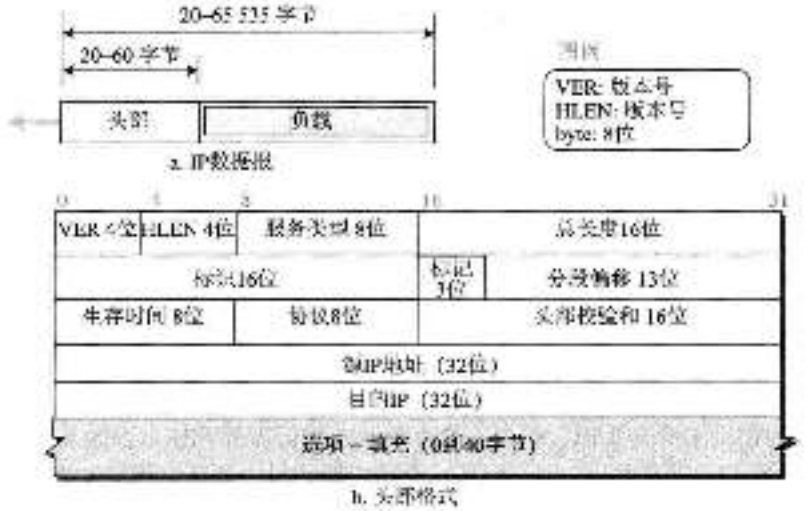


图4-24 IP数据报

- **服务类型**。在IP头部的最初设计中，这个字段指的是服务类型（type of service，TOS）。它定义了数据报如何被处理。在20世纪90年代，IETF将这个字段重定义为提供差分服务（differentiated services，DiffServ）。当我们在第8章讨论差分服务时，我们将更好地定义这个字段中的位。
- **总长**。这个16位字段定义了一个以字节计算的IP数据报总长度（头部加上数据）。一个16位数字可以定义长达65 535的总长度（当所有位都是1时）。然而，数据报的长度通常远远小于这个值。这个字段帮助接收设备知道什么时候分组完全到达。
- **标识、标记以及分段偏移**。当数据报大于底层网络可以携带的大小时，这三个字段和IP数据报的分段有关。当我们在后面讨论分组时，将会讨论这些字段的内容和重要性。
- **生存时间**。由于路由协议的某些故障（稍后讨论），数据报可能在因特网中循环，一遍又一遍地访问某些网络而没有到达目的端。这可能造成因特网中的额外通信量。生存时间（TTL）字段用于控制数据报访问的最大跳数（路由器）。当源主机发送数据报，它在这个字段存储一个数字。这个数值大约是任意两主机之间路由数量最大值的两倍。每个处理数据报的路由器将此数值减1。如果在减1之后，此字段的值为0，路由器就丢弃该数据报。
- **协议**。在TCP/IP中，分组的数据段称为负载，它从另一个协议中携带整个分组。例如，一个数据报可以携带属于任意传输层协议如UDP或TCP的分组。数据报也可以从其他协议携带分组，其他的协议直接使用IP服务，例如某些路由协议或某些辅助协议。因特网机构已经给任何使用IP服务的协议一个唯一的8位数字，它被插入到协议字段。在源端IP当负载被封装到数据报中时，相应协议号被插入这个字段；当数据报到达目的端，这个字段的值帮助定义负载应该被传递到那个协议。换言之，如图4-25所示，这个字段在源端提供多路

复用，在目的端提供多路分解。注意，网络层的协议字段与传输层的端口号起到相同的作用。然而，我们需要在传输层分组中的两个端口号，因为源端和目的端的端口号是不同的，但是我们只需要一个协议字段，因为这个值对于每个协议都是相同的，无论它在源端还是目的端。

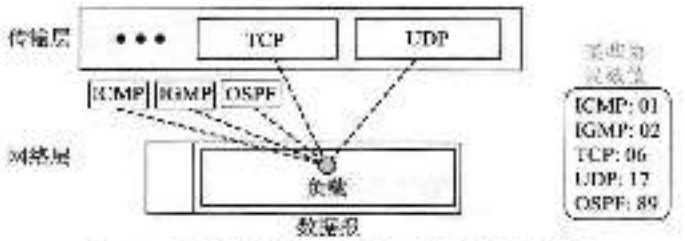


图 4-25 使用协议字段值的多路复用和多路分解

- 头部校验和。IP 不是可靠性协议；在发送期间，它不检查数据报中携带的负载是否被破坏。IP 承担起 UDP 或 TCP 协议中的负载差错检测。然而，数据报头部被 IP 加入，并且它的差错检测是 IP 的责任。IP 头部的差错可能是一个灾难。例如，如果目的 IP 地址被破坏，分组可能被传递到错误的主机。如果协议字段被破坏，分组可能被传递到错误协议。如果与分段相关的字段被破坏，数据报不能在目的端被正确重组，等等。出于这些原因，IP 加入头部校验和字段来检查头部，但是不检查负载。我们需要记住，因为某些字段的值，例如 TTL，与分段和选项有关，它们可能从一个路由器到另一个路由器发生改变，校验和需要在每个路由器重新计算。正如我们在第 3 章讨论过的，因特网中的校验和通常使用一个 16 位字段，这是其他字段使用 1 的补码运算计算得到的和的补码。当我们在第 5 章探讨差错检测时我们将讨论校验和计算。
- 源和目的地址。这个 32 位源和目的地址字段分别定义了源端和目的端的 IP 地址。源主机应该知道它的 IP 地址。目的 IP 地址可能被使用 IP 服务的协议知晓，也可能由如第 2 章描述的 DNS 提供。注意，这些段的数值必须在 IP 数据报从源主机到目的主机的传输过程中保持不变。我们在本章稍后将更多地讨论 IP 地址以及它们的结构。
- 选项。数据报头部可以有多达 40 个字节的选项。选项可以用来进行网络测试和调试。尽管选项不是 IP 头部必需的部分，但是选项处理是 IP 软件必需的。这意味着，如果头部有选项，所有实现必须能够处理它。头部中选项的存在给数据报处理增添了一些负担；某些选项可以被路由器改变，这迫使每个路由器重新计算头部校验和。在本章的额外材料中我们会在本书的网站上详细讨论一字节和多字节选项。
- 负载。负载或数据是创建数据报的主要原因。负载是来自使用 IP 服务的其他协议的分组。将数据报比作邮包的话，负载是包裹的内容；头部是包裹上写的信息。

分段

一个数据报可以通过几个不同的网络进行传输。每一个路由器将它所接收的帧拆封成 IP 数据报，对它进行处理，然后再将它封装成另一个帧。接收到的帧的格式和长度取决于此帧刚刚经过的物理网络所使用的协议。被发送的帧的格式和长度取决于此帧将要经过的物理网络所使用的协议。例如，如果一个路由器将一个 LAN 连接到一个 WAN，那么它以 LAN 格式接收帧，以 WAN 的格式发送帧。

最大传输单元（maximum transfer unit，MTU）

每一个数据链路层协议（见第 5 章）都有其自己的帧格式。每种格式的特征之一是可以封装的

负载的最大长度。换言之，当数据报被封装到帧中，数据报的总大小必须小于这个最大长度，这是根据网络中所使用的硬件和软件给出的限定所定义的（见图 4-26）。



图 4-26 最大传输单元（MTV）

MTU 的值随着物理网络协议的不同而不同。例如，LAN 的值通常为 1500 字节，但是对于 WAN 它可能更大或更小。

为了使得 IP 协议独立于物理网络，设计者决定将 IP 数据报的最大长度限定为 65 535 字节。如我们使用带有这个大小的 MTV 的链路层协议，将使得传输更加有效率。然而，对于其他物理网络，我们必须将数据报进行分割，使其能够通过这些网络。这个过程称为分段（fragmentation）。

当数据报被分段，每个段都有自己的头部。头部中绝大多数字段都是重复的，但是有一些会被改变。一个被分段的数据报如果遇到 MTU 更小的网络它有可能再次被分段。换言之，一个数据报可能在到达最终目的端之前被多次分段。

一个数据报可能被源主机或路径上的任意路由器分段。然而，数据报的重组只能在目的主机上进行，因为每个分段都是一个独立的数据报。由于被分段的数据报可以沿着不同路径传输，因此我们不能控制或保证一个分段的数据报会走哪一条路径，也不能保证所有属于同一个数据报的分段最终都到达目的主机。所以，从逻辑上说，应当在最终的目的端进行重组。一个更强的目的是，在传输期间重组分组将会带来效率的降低。

当我们讨论分段时，我们的意思是 IP 数据报的负载被分段。然而，除了某些选项之外，头部的绝大部分必须拷贝到所有分段。将数据报分段的主机或路由器必须改变这三个字段的值：标记、分段偏移和总长度。剩余字段必须被拷贝。当然，不管是否进行分组，校验和的值必须被重新计算。

与分段相关的字段

我们之前提及过 IP 数据报中的三个与分组相关的字段：标识（identification）、标记（flag）和分段偏移（fragmentation offset）。现在让我们来解释这些字段。

16 位标识字段用于识别一个从源主机发出的数据报。当数据报离开源主机时，这个标识和源 IP 地址的组合必须唯一地定义这个数据报。为了保证唯一性，IP 协议使用一个计数器来标识数据报。当 IP 协议发送数据报时，就将该计数器的当前值复制到标识字段中，并将此计数器的值加 1。只要此计数器保存在主存储器中，唯一性就得到保证。当数据报分段时，标识字段的值就被复制到所有分组中。换言之，所有的分段都有与原始的数据报相同的标识号。标识号有助于在目的端重组数据报。目的端知道应将所有具有相同标识值的分段重组成一个数据报。

3 位标记字段定义了三个标记。最左侧的位是保留位（不使用）。第二位（D 位）称为不分段（do not fragment）位。如果其数值为 1，则机器不能将该数据进行分段。如果无法将此数据报通过任何可用的物理网络进行传递，那么机器就丢弃这个分组，并向源主机发送一个 ICMP 差错报文（稍后讨论）。如果其值为 0，则根据需要对数据报进行分段。第三位（M 位）称为多分段（more fragment）位。如果其值为 1，则表示此数据报不是最后的分段，在该分段后还有更多的分段；如果其值为 0，则表示它是最后一个或唯一的分段。

13 位的分段偏移字段表示这个分段在整个数据报中的相对位置。它是在原始数据报中的数据偏移量，以 8 字节为度量单位。图 4-27 给出了具有 4000 字节的数据报被划分成三个分段。原始数据报的字节编号从 0 到 3999。第一个分组携带的数据是字节 0 到 1399，对于这个数据报，分段的偏移量是 0/8 = 0。第二个分段携带的数据是字节 1400 到 2799；对于这个数据报其偏移量为 1400/8 = 175。最后，第三个分段携带的数据是字节 2800 到 3999，其偏移量为 2800/8 = 350。

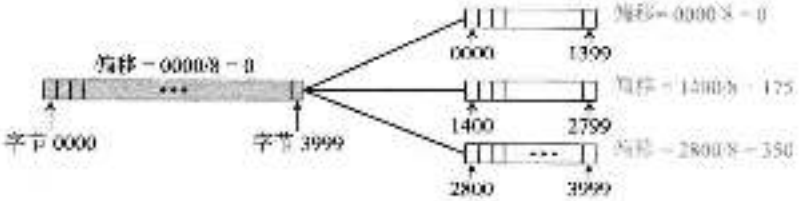


图 4-27 分段示例

请记住，偏移量是以 8 字节为单位的。这样做是因为偏移字段的长度只有 13 位，不能用来表示一个大于 8191 的字节数。因此，将数据报进行分组的主机或路由器必须这样选择每个分段的长度，即第一个字节标号能被 8 整除。

图 4-28 给出了图 4-27 中分段的扩展图。原始分组从客户端开始；分组在服务器端被重组。所有分段中的标识字段数值是相同的，除了最后一个分段外，所有分段的标记字段的值也是相同的。注意，尽管分段失序到达目的端，但是它们可以被正确重组。

这幅图还表示分段本身再进行分段时会发生什么。在这种情况下，分段偏移量永远是相对于原始数据报的。例如，图中第二个分段又划分为两个长度分别为 800 字节和 600 字节的分段，但这些分段的偏移量所表示的位置都相对于原始数据的位置。

显然，即使每一个分段走不同的路径，并在到达时失序，最终目的主机也能用收到的这些分段（如果没有丢失）重组成原始数据报。所使用的策略如下：

- 第一个分段的偏移量字段的值为 0；
- 将第一个分段长度除以 8，其结果就是第二个分段的偏移值；
- 将第一个和第二个分段的总长度除以 8，其结果为第三个分段的偏移值；
- 继续以上的过程，最后一个分组的 M 位即多分段位的值为 0。

IPv4 数据报安全

IPv4 协议以及整个因特网开始于因特网用户彼此信赖的时代。IPv4 协议没有提供任何安全。然而，如今情况不同了；因特网不再安全。尽管我们泛泛讨论网络安全，并且会在第 10 章着重讨论 IP 安全，但是在这里，我们给出 IP 协议中安全问题及其解决方法的简要思想。IP 协议尤其易受这三个安全问题的影响：分组嗅探 packet sniffing）、分组修改 packet modification）以及 IP 欺骗 IP spoofing）。

分组嗅探

入侵者可能劫持一个 IP 分组并制作一份拷贝。分组嗅探是一种被动攻击，其中攻击者并不改变分组的内容。这类型攻击非常难以发现，因为发送者和接收者可能从不知道他们的分组被复制。尽管分组嗅探不能被停止，但是分组的加密可以使得攻击者的企图无效。攻击者可能仍然嗅探分组，但是分组内容无法被发现。

分组修改

第二类攻击是修改分组。攻击者拦截分组，改变其内容并将新的分组发送到接收方。接收方相信分组来自原始发送方。可以使用数据完整性机制来发现这类型攻击。在打开并使用报文的内容之前，接收方可以在发送阶段使用这个机制来确定分组没有被改变。我们在第 10 章讨论分组完整性。

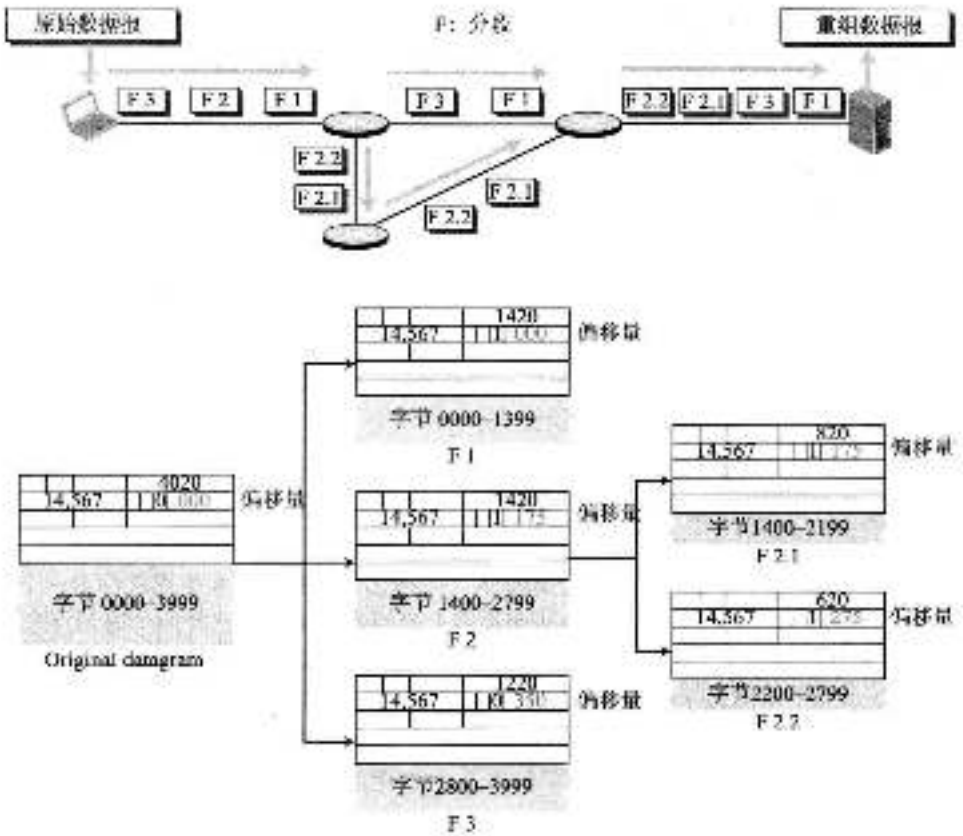


图 4-28 分段的细节示例

IP 欺骗

一个攻击者可以伪装成其他人并创建一个 IP 分组，它携带另一个电脑的源地址。一个攻击者可以发送一个 IP 分组给银行，将它自己伪装成一名客户。这类型的攻击可以使用源鉴别机制来预防（见第 10 章）。

IPSec

如今的 IP 分组可以使用一个称为 IPSec（IP Security，IP 安全）的协议以免遭到之前提到的攻击。这个协议与 IP 协议一起使用，在两个实体之间创建了一个面向连接服务。这两个实体可以交换 IP 分组而不必担心之前提到的三种攻击。我们将在第 10 章详细讨论 IPSec；这里需要提及的是 IPSec 提供如下两种服务：

- **定义算法和密钥**。为了安全起见，想创建安全信道的两个实体可以统一使用一些算法和密钥。
- **分组加密**。为了保护隐私，在双方之间交换的分组可以被加密，加密使用了在第一步中达成一致的加密算法和共享密钥。这使得分组嗅探攻击无效。
- **数据完整性**。数据完整性保证分组在传输阶段没有被修改。如果接收分组没有通过数据完整性测试，它就被丢弃。这防止前面描述的第二种攻击，即分组修改。
- **源鉴别**。IPSec 可以鉴别分组的源，从而来确认分组不是被冒名顶替者伪造的。这个防止上文所述的 IP 欺骗攻击。

4.2.2 IPv4 地址

有一种标识符，它被 TCP/IP 协议簇的 IP 层用来标识连接到因特网的设备，这种标识符称作因

特网地址或 IP 地址。一个 IPv4 地址是 32 位地址，它唯一地并通用地定义了一个连接在因特网上的主机或路由器。IP 地址是连接的地址，不是主机或路由器的地址，因为如果设备移动到另一个网络，IP 地址可能会改变。

IPv4 地址是唯一的，这表示每一个地址定义了一个且唯一一个连接到因特网上的设备。如果某个设备有两个到因特网的连接，那么它就有两个 IP 地址。IPv4 地址是通用的，这表示地址系统必须被任何一个想要连接到因特网上的主机所接收。

地址空间

像 IPv4 这样定义了地址的协议拥有一个地址空间（address space）。地址空间是该系统能够使用地址的总个数。如果协议使用 b 位来定义地址，地址空间是 2^b，因为每一位可能有两个不同值（0 或 1）。IPv4 使用 32 位地址，这意味着地址空间是 2^{32} 或 4 294 967 296（大于 40 亿）。如果没有限制，超过 40 亿个设备可以连接到因特网上。

标记法

IPv4 地址有三种常用的标记法：二进制标记法（基数 2）、点分十进制标记法（基数 256）以及十六进制标记法（基数 16）。在二进制标记法（binary notation）中，IPv4 地址用 32 位表示。为了使这种地址可读性更强，通常在每 8 位之间插入一个或多个空格。每 8 位通常称为一个字节。为了使 IPv4 地址更加简洁和易读，因特网地址通常用十进制形式来书写，并用十进制点来分隔这些字节。这种格式称为点分十进制标记法（dotted-decimal notation）。注意，由于每个字节（8 位字节）只有 8 位，因此点分十进制表示法中的每个数据在 0 和 255 之间。我们有时看到十六进制形式的 IPv4 地址。每个十六进制数字等价于 4 位。这意味着 32 位地址有 8 个十六进制数字。这种表示法通常用于网络编程。图 4-29 以三种标记法给出了一个 IP 地址。

地址层次结构

如电话网络或邮政网络这类涉及传递的网络，地址系统都是有层次结构的。在邮政网络中，邮政地址（信件地址）包含国家、州、城市、街道、门牌号以及邮件接收者姓名。类似地，电话号码也分为国家代码、地区代码、当地交换局代码以及连接。

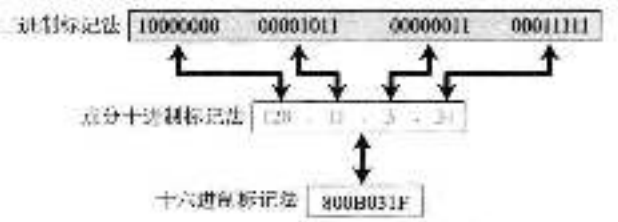


图 4-29　IPv4 地址的三种不同标记法

一个 32 位 IPv4 地址也是有层次结构的，但是它只被分为了两部分。地址的第一部分称为前缀，它定义了网络；地址的第二部分称为后缀，它定义了代码（设备到因特网的连接）。图 4-30 给出一个 32 位 IPv4 地址的前缀和后缀。前缀长度是 n 位，后缀长度是（32- n）位。

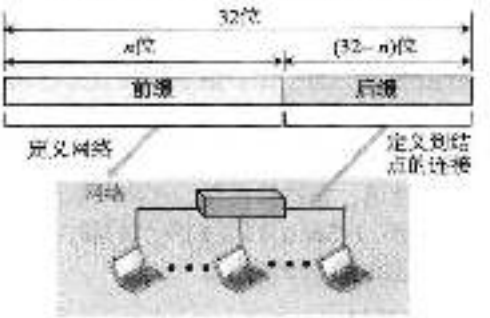


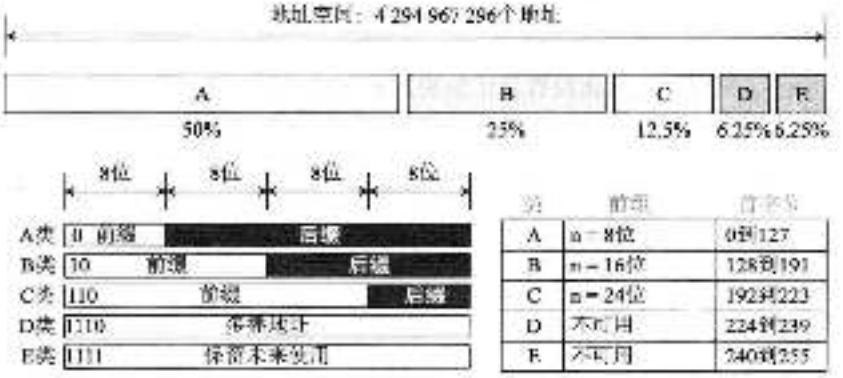
图 4-30　地址的层次结构

前缀可以是固定长度的也可以是变长的。IPv4 中的网络标识符起先被设计为定长前缀。这个方案现在被废止了，它称为分类寻址。一种新的方案称为无类寻址，它使用变长的网络前缀。首先，我们简要讨论分类寻址；之后我们专注讨论无类寻址。

分类寻址

当因特网刚开始发展时，IPv4 地址被

设计成定长前缀的，但是为了满足较小或较大的网络，设计了三种长度的前缀（$n=8$、$n=16$ 以及 $n=24$），而不是只设计了一种。整个地址空间分为五类（A、B、C、D 和 E 类）。如图 4-31 所示。这个方案称为分类寻址（classful addressing）。



类	前缀	首个块
A	$n=8$位	0到127
B	$n=16$位	128到191
C	$n=24$位	192到223
D	不适用	224到239
E	不适用	240到255

图 4-31 分类寻址中地址空间的占用

在 A 类地址中，网络长度是 8 位，但是由于第一位是 0，它定义了类，因此我们只能使用 7 位作为网络标识符。这意味着世界上只能有 2^7-128 个网络可以拥有 A 类地址。

在 B 类地址中，网络长度是 16 位，但是由于前两位即$(10)_2$定义了类，我们只能用 14 位作为网络标识符。这意味着世界上只能有 $2^{14}-16\,384$ 个网络可以拥有 B 类地址。

所有以$(110)_2$开头的地址都属于 C 类地址。在 C 类地址中，网络长度是 24 位，但是因为有 3 位定义了类型，因此我们只有 21 位作为网络标识符。这意味着世界上有 $2^{21}-2\,097\,152$ 个网络可以拥有 C 类地址。

D 类地址并不分为前缀和后缀。它是多播地址。所有以二进制 1111 开头的地址属于 E 类地址。像 D 类地址一样，E 类地址也没有分为前缀和后缀，并且 E 类地址留作将来使用。

地址耗尽

分类寻址被废止的原因就是地址耗尽。因为地址没有被恰当分配，因特网面临地址浪费用光的问题。这导致了需要连接到因特网的组织和个人没有可用的地址。为了理解这个问题，让我们来考虑 A 类地址。这个类仅仅可以分配给世界上的 128 个组织，但是每个组织需要有一个带有 16 777 216 个结点（在这个网络中的计算机）的网络（被世界其余计算机看到）。由于只有很少的组织会如此庞大，在这个类中绝大多数的地址都浪费了（没有被使用）。B 类地址为中等组织设计，但是这一类中多数地址也没有被使用。C 类地址在设计上有一个完全不同的缺陷。每个网络中可以使用的地址数量（256）过小，以至于绝大多数使用 C 类中一大块地址的公司并不感到宽裕。E 类地址几乎从未使用，浪费了整个类。

子网化和超网化

为了减轻地址耗尽，提出了两种策略，并且它们在某种程度上被实施了，那就是子网化和超网化。在子网化中，A 类或 B 类中的一大块地址被分成几个子网。每个子网有一个比原网络更大的前缀长度。例如，如果 A 类网络被分成四个子网，每个子网有一个 $n_{sub}-10$ 的前缀。同时，如果网络中的所有地址没有被使用，子网允许地址被几个组织分割。这个思想并不起作用，因为绝大多数大型组织并不乐于将大块地址进行分割并把未使用的地址给小型组织。

子网化被设计成将大块地址分成小块，而超网化则是将几个 C 类地址块组合成一个较大的块，以此吸引那些需要 256 个以上地址的组织，而一个 C 类地址只有 256 个地址可用。这个思想也不起作用，因为它使得分组路由更困难。

分类寻址的优势

尽管分类寻址有很多问题而且废止，但是它有一个优势：给定一个地址，我们可以轻易地找到地址的类，并且因为每个类的前缀是固定的，因此我们可以立即找到前缀的长度。换言之，分类寻址中的前缀长度是固有的；不必提供额外信息来提取前缀和后缀。

无类寻址

分类寻址中的子网化和超网化没有真正解决地址耗尽问题。随着因特网的发展，很明显，长久的解决办法是需要更大的地址空间。然而，较大的地址空间需要IP地址长度的增加，这意味着IP分组的格式需要改变。尽管长期的解决方法已经提出且被称为IPv6（后文讨论），但是短期的解决方法也被设计出来了，它使用相同的地址空间，但是将地址的分配改成了为每个组织提供平等的分享。短期解决方法仍然使用IPv4地址，但是，它称为无类寻址（classless addressing）。换言之，为了补偿地址耗尽，类特权被从分配中消除。

无类寻址还有另外一个动机。在20世纪90年代，因特网服务提供商（ISP）开始涌现。ISP是为个人、小型企业和中型组织提供因特网连接的组织。那些个人、小型企业和中型组织不想自己建立网站并为自己的员工提供因特网服务（例如电子邮件）。ISP可以提供这些服务。ISP被分配了很大范围的地址并且再将地址细分（1、2、4、8、16个地址一组，等等），它将一定范围的地址给家用或小公司。用户通过拨号调制解调器、DSL或电缆调制解调器连接到ISP。然而，每个用户都需要一些IPv4地址。

在1996年，因特网机构宣布了一个新的体系结构，称为无类寻址。在无类寻址中，使用了不属于任何类的变长块。我们可以使用1个地址的块、2个地址的块、4个地址的块、128个地址的块，等等。

在无类寻址中，整个地址空间被分为变长块。地址的前缀定义了块（网络）；后缀定义了结点（设备）。理论上我们有2^0个地址的块，2^1个地址的块、2^2个地址的块……2^{32}个地址的块。如后文所述，其中一个限制是块内的地址数量需要是2的乘方。一个组织可以赋给一块地址。图4-32给出整个地址空间分成不重叠的块。

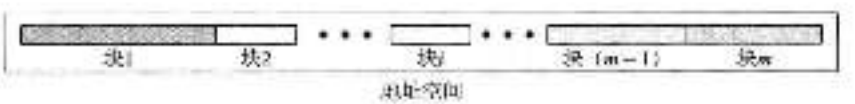


图4-32 无类寻址中的变长块

不像分类寻址，无类寻址中前缀长度是可变的。前缀的长度可以在0到32之间变化。网络的大小与前缀的长度成反比。一个小的前缀意味着较大的网络；一个大的前缀意味着较小的网络。

我们需要强调的是无类寻址思想很容易应用到分类寻址。一个A类地址可以看做是前缀长度为8的无类寻址。一个B类地址可以看做是前缀长度为16的无类寻址，等等。换言之，分类寻址是无类寻址的特殊情况。

前缀长度：斜杠标记法

在无类寻址方面，我们需要回答的第一个问题是，如果给出地址，那么如何找出其前缀的长度。由于地址中的前缀长度不是固定的，我们需要分别给出前缀的长度。在这种情况下，前缀的长度n被加入到地址中，用斜杠来分隔。这个标记法的非正式称呼为斜杠标记法，正式的称呼为无类域间路由（classless interdomain routing）或CIDR（读作cider）策略。无类寻址的地址也可以用图4-33表示。

换言之，从本质上说，无类寻址中的地址定义了地址所属的块或网络；我们也需要给出前缀的长度。

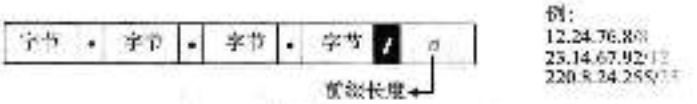


图 4-33 斜杠标记法（CIDR）

从一个地址中抽取信息

给出块中的任意一个地址，我们通常想知道三个地址所属块的信息：地址的数目，块中首地址以及末地址。因为前缀长度 n 已给出，我们可以很轻易地找到这三个信息，如图 4-34 所示。

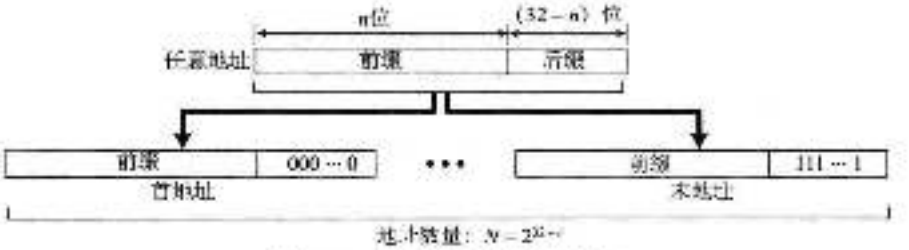


图 4-34 无类地址中的信息抽取

1. 块中地址的数量通过 $N = 2^{32-n}$ 得出。

2. 为了找到首地址，我们保持最左 n 位不变，并将最右侧的（$32 - n$）位全设为 0。

3. 为了找到末地址，我们保持最左 n 位不变，并将最右侧的（$32 - n$）位全设为 1。

例 4.1 一个无类地址为 167.199.170.82/27。如下所示，我们可以找到三个信息。网络中的地址数量是 $2^{32-27} = 2^5 = 32$ 个地址。

首地址可以通过保持前 27 位不变并将剩余位设为 0 得到。

地址：167.199.170.82/27　　10100111 11000111 10101010 01010010

首地址：167.199.170.64/27　10100111 11000111 10101010 01**000000**

末地址可以通过保持前 27 位不变并将剩余位设为 1 得到。

地址：167.199.170.82/27　　10100111 11000111 10101010 01011111

末地址：167.199.170.64/27　10100111 11000111 10101010 01**11111**

地址掩码

另一种找到块中首末地址的方法是使用地址掩码。地址掩码是 32 位数字，其中最左边 n 位都是 1，而最右边(32 − n)位都是 0。计算机可以很容易找到地址掩码，因为它是（$2^{32} - 1$）的补码。用这种方式定义掩码的原因是，计算机程序可以使用三个位操作符 NOT、AND 以及 OR 来从中抽取信息。

1. 块中的地址数量 N = **NOT**（掩码）+ 1。

2. 块中首地址 =（块中任意地址）**AND**（掩码）。

3. 块中末地址 =（块中任意地址）**OR**[**NOT**（掩码）]。

例 4.2 我们使用掩码重复例 4.1。点分十进制表示法中的掩码是 256.256.256.224。如果你使用互联网站上的计算器和 applet，那么可以将 AND、OR 以及 NOT 操作符用于单个字节。

块中地址数量：　N = **NOT**（掩码）+ 1 = 0.0.0.31 + 1 = 32 个地址

首地址：　First =（地址）**AND**（掩码）= 167.199.170. 82

末地址：　Last =（地址）**OR**（**NOT** 掩码）= 167.199.170. 255

例 4.3 在分类寻址中，地址本质上不能定义自己所属的块。例如地址 230.8.24.56 可以属于很多块。以下列出部分块以及其前缀的值。

前缀长度：16 → 块：230.8.0.0　到 230.8.255.255
前缀长度：20 → 块：230.8.16.0　到 230.8.31.255
前缀长度：26 → 块：230.8.24.0　到 230.8.24.63
前缀长度：27 → 块：230.8.24.32　到 230.8.24.63
前缀长度：29 → 块：230.8.24.56　到 230.8.24.63
前缀长度：31 → 块：230.8.24.56　到 230.8.24.57

网络地址

以上的例子表示，给出任意一个地址，我们可以找到关于块的所有信息。首地址即网络地址（network address）是尤其重要的，因为它用于将分组路由到目的端。现在，让我们假设互联网由 m 个网络和带有 m 个接口的路由器组成。当分组从任意源主机到达路由器时，路由器需要知道分组应该被发送到哪个网络；分组应该从哪个接口被发送出去。当分组到达网络时，它使用另一个我们稍后讨论的策略来到达目的地。图 4-35 表示了这种思想。在网络地址被找到后，路由器会查转发表来找到相应的接口，分组从这个接口被发送出去。网络地址实际上是网络的标识符；每个网络被它的网络地址标识。

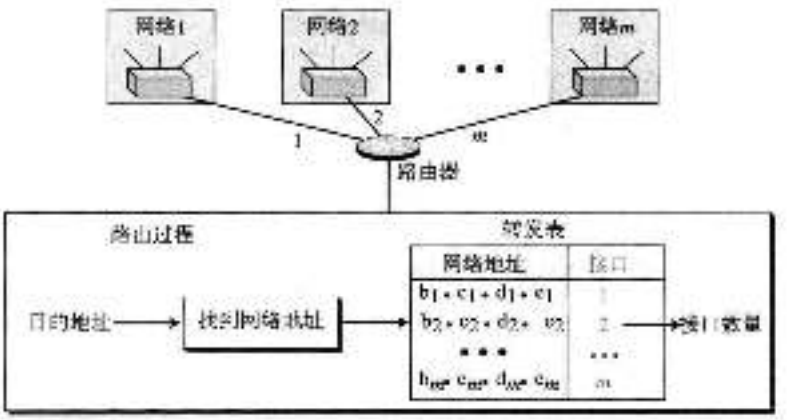


图 4-35　网络地址

块分配

无类寻址的下一个问题是块分配。块如何被分配？块分配的最终责任交给一个全球的机构，称为因特网名称和编号分配组织（Internet Corporation for Assigned Names and Numbers，ICANN）。然而，ICANN 通常不向因特网个人用户分配地址。它给一个 ISP（或一个较大组织，这种情况下认为是一个 ISP）分配一大块地址。为了使 CIDR 正常运行，分配块有两个限制条件。

1. 请求地址的数量 N 必须是 2 的整数次幂。原因是 $N = 2^{32-n}$ 或 $n = 32 - \log_2 N$，如果 N 不是 2 的整数次幂，n 就不是确定值。

2. 请求块需要被分配时，地址空间中要有连续的可用地址。然而，对块中的首地址进行选择时有一条限制。首地址必须可以被块中的地址数量整除。原因是首地址需要是前缀加后面（$32-n$）个 0。首地址的十进制值是：

$$\text{首地址} = (\text{十进制前缀}) \times 2^{32-n} = (\text{十进制前缀}) \times N。$$

例 4.4　一个 ISP 已经请求了 1000 个地址的块。因为 1000 不是 2 的幂，因此被分配了 1024 个地址。前缀长度被计算为 $n = 32 - \log_2 1024 = 22$。一个可用的块，18.14.12.0/22 被分配给 ISP。可以看到十进制的首地址是 302 910 464，它可被 1024 整除。

子网化

使用子网化可以创建更多的层次结构等级。被授予一定范围地址的组织（或一个ISP）可以将地址范围分为几个子范围并将每个子范围分配给一个子网。注意，什么也不能阻止这个组织去创建更多的等级。子网可以被分为多个子子网。一个子子网可以被分为多个子子子网，以此类推。

设计子网 网络中的子网应该被仔细设计，使之能够路由分组。我们假设授予某个组织的地址总数为N，前缀长度为n，被分配给每个子网的地址数为N_{sub}，每个子网的前缀长度为n_{sub}。需要仔细遵从以下步骤从而保证子网的正确运行。

- 每个子网的地址数量应该是2的整数次幂。
- 每个子网的前缀长度应该使用下列公式计算出来：

$$n_{sub} = 32 - \log_2 N_{sub}$$

- 每个子网的起始地址应该可以被子网中的地址数目整除。如果我们首先将地址分配给较大的子网，这个是可以实现的。

寻找关于每个子网的信息 在子网设计之后，关于每个子网的信息，例如首地址和末地址，可以通过使用我们所描述的一个过程来找到，这个过程用于寻找网络中每个网络的信息。

例4.5 一个组织被授予一块地址，首地址是14.24.74.0/24。这个组织需要3个地址子块来在它的3个子网中使用：一个10个地址的子块、一个60个地址的子块以及一个120个地址的子块。请设计这些子块。

解答

这个块中有$2^{32-24} = 256$个地址。首地址是14.24.74.0/24；末地址是14.24.74.255/24。为了满足第三个要求，我们将地址分配给子段，以最大地址开始，以最小地址结束。

a. 最大子块要求120个地址，它不是2的整数次幂。我们分配128个地址。这个子网的子网掩码可以通过$n_1 = 32 - \log_2 128 = 25$得出。这个段的首地址是14.24.74.0/25；末地址是14.24.74.127/25。

b. 第二大子块60个地址，它也不是2的整数次幂。我们分配64个地址。这个子网的子网掩码可以通过$n_2 = 32 - \log_2 64 = 26$得出。这个段的首地址是14.24.74.128/26；末地址是14.24.74.191/26。

c. 最小子块60个地址，它也不是2的整数次幂。我们分配16个地址。这个子网的子网掩码可以通过$n_3 = 32 - \log_2 16 = 28$得出。这个段的首地址是14.24.74.192/28；末地址是14.24.74.207/28。

如果我们将之前子块的所有地址加在一起，结果是208个地址，这意味着剩余48个地址保留。这个范围内的首地址是14.24.74.208，末地址是14.24.74.255。我们还不知道前缀长度。图4-36给出块的配置。我们已经给出每个块的首地址。

地址聚合

CIDR策略的一个优势就是地址聚合（address aggregation）（有时称为地址摘要或路由摘要）。当地址块结合成一个更大的块时，路由可以基于更大块的前缀完成。ICANN给一个ISP分配一个大块地址。每个ISP轮流将它分配的块分割成较小的子块，并将其给予它的用户。

例4.6 图4-37给出四个小地址块如何通过一个ISP分配给四个组织。ISP将这四个块聚合成单独一个块并向世界其余计算机声明这个更大的块。任何想要到达这个较大块的分组应该被发送到这个ISP。将分组转发到恰当的组织是ISP的责任。这与邮政网络中的路由相似。所有来自国家之外的分组首先被发送到首都，然后被分发到相应的目的地。

特殊地址

在结束IPv4地址的话题之前，我们将要提及5种用于特殊目的的地址：本地主机地址、有限广播地址、回送地址、私有地址以及多播地址。

本地主机地址（This-host Address） 块中唯一的地址0.0.0.0/32称为本地主机地址。当主机

需要发送一个 IP 数据报时，需要自己的地址作为源地址，但主机不知道这个地址，这时会使用到本地主机地址。我们将会在下一节看到这个情况的例子。

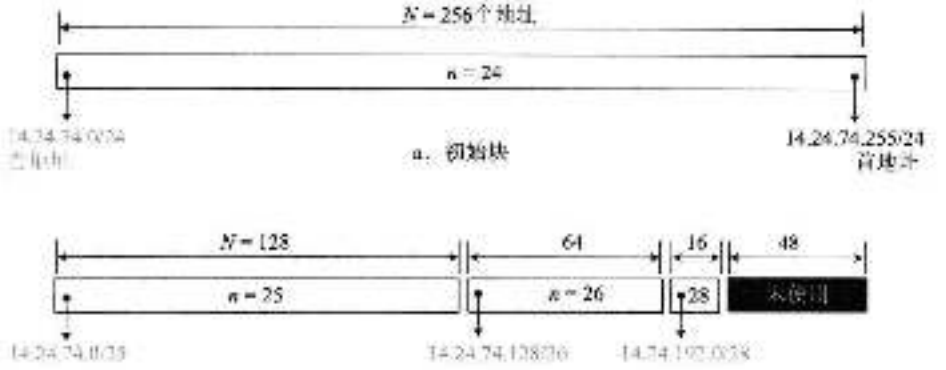


图 4-36 例 4.5 的解决方法

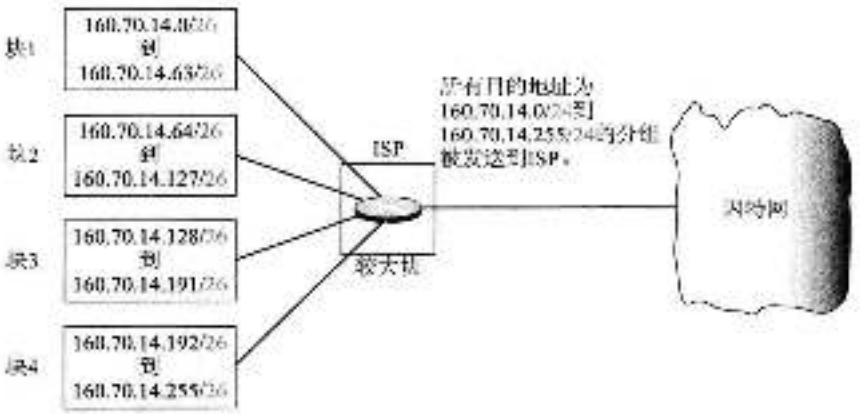


图 4-37 地址聚合的例子

有限广播地址（Limited-broadcast Address） 块中唯一的地址 255.255.255.255/32 称为有限广播地址。当路由器或主机需要向网络中所有设备发送一个数据报时会使用到这个地址。然而，网络中的路由器阻挡目的地址是这个有限广播地址的分组；分组不能传递出网络。

回送地址（Loopback Address） 段 127.0.0.0/8 称为回送地址。如果某分组的目的地址是这个块中某一个地址，那么它不会离开主机；它将会留在主机中。这个块中的地址用于测试机器中的一个软件。例如，我们可以写一个客户和服务器程序，并将一个回送地址作为服务器地址。在将这些程序运行在其他电脑之前，我们可以使用同一个主机来测试程序，看它们是否工作。

私有地址（Private Address） 四个块分配为私有地址：10.0.0.0/8、172.16.0.0/12、192.168.0.0/16 以及 169.254.0.0/16。当在本章稍后讨论 NAT 时，我们将看到这些地址的应用。

多播地址（Multicast Address） 块 224.0.0.0/4 的地址保留为多播地址。我们在本章稍后讨论这些地址。

动态主机配置协议（DHCP）

我们已经看到一个大型组织或一个 ISP 可以直接从 ICANN 获得一个地址块，并且一个小型组织可以从 ISP 获得一个地址块。在地址块被分配给一个组织之后，网管可以手动将地址分配给个人

主机或路由器。然而，一个组织中的地址分配可以使用动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）来完成。DHCP 是一个应用层程序，它使用客户-服务器模式，实际上帮助网络层的 TCP/IP。

DHCP 在因特网中有很广泛的应用，它经常被称为即插即用协议（plug-and-play protocol）。它可以用于多种情境。网络管理员可以配置 DHCP 来将永久 IP 地址分配到主机和路由器上。DHCP 也可以被配置来按需为主机提供临时 IP 地址。第二个功能是为旅行者提供临时 IP 地址，这样旅行者就能在旅店里将笔记本连接到因特网上。假设同时有不多于 1/4 的用户使用因特网，它也允许一个拥有 1000 个地址的 ISP 为 4000 个用户提供服务。

除了 IP 地址，计算机也需要知道网络前缀（或地址掩码）。绝大多数计算机也需要其他两项信息，比如默认路由器地址，这样它就能连接到其他网络，以及域名服务器地址。这正如我们在第 2 章看到的，它就能够使用域名而不使用地址了。换言之，如下四个信息通常是必需的：计算机地址、前缀、路由器地址以及域名服务器 IP 地址。可以用 DHCP 将这些信息提供给主机。

DHCP 报文格式

DHCP 是一个客户-服务器协议，其中客户发送一个请求报文并且服务器返回一个响应报文。在我们讨论 DHCP 的操作之前，图 4-38 给出 DHCP 报文的一般格式。图中解释了绝大多数的字段，但是我们需要讨论选项字段，它在 DHCP 中起到很重要的作用。



字段：
Opcode：操作码，请求（1）或回答（2）
Htype：硬件类型（以太网，……）
HLen：硬件地址的长度
HCount：分组可以传递的最大跳数
事务ID：客户设置的一个整数以标识被服务器重复
运行时间：从客户开始启动电脑后的秒数
标记：第一位定义单播（0）或多播（1）；其他15位未使用
客户IP地址：如果客户不知道则设置为0
你的IP地址：由服务器发送的客户IP地址
服务器IP地址：如果客户不关心则设为广播IP地址
网关IP地址：默认路由器地址
服务器名：一个64字节服务器域名
引导文件名：一个含有额外信息的128字节文件名
选项：一个64字节字段，含有文本中所描述的各种目的

图 4-38 DHCP 报文格式

64 字节选项字段含有双重目的。它可以携带附加信息或一些特定的厂商信息。服务器使用一个称为 **magic cookie** 的号码，它以 IP 地址形式给出，其值为 99.130.83.99。当客户读完报文，它寻找 magic cookie。如果存在，接下来的 60 字节就是选项。一个选项由三个字段组成：一个 1 字节标签字段、一个 1 字节长度字段和一个变长值字段。有很多标签字段，它们绝大多数被厂商使用。如果标签字段是 53，数值字段定义了如图 4-39 所示的 8 个报文中的一个。我们给出 DHCP 使用的报文类型。

1 DHCPDISCOVER　5 DHCPACK
2 DHCPOFFER　6 DHCPNACK
3 DHCPREQUEST　7 DHCPRELEASE
4 DHCPDECLINE　8 DHCPINFORM

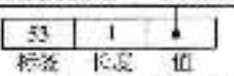


图 4-39 选项格式

DHCP 运行

图 4-40 给出一个简单的情景。

1. 连接主机创建一个 DHCP**DISCOVER** 报文，其中只有事务 ID 字段被设为随机数字。没有其他可以设置的字段，因为主机没有相应信息。这个报文被封装到 UDP 用户数据报当中，其中源端口设为 68，目的端口设为 67。后文我们将讨论使用两个熟知端口号的理由。用户数据报被封装到 IP 数据报中，其中源地址为 **0.0.0.0**（"本机"），目的地址被设为 **255.255.255.255**（广播地址）。

这样设置的理由是连接主机既不知道它自己的地址也不知道服务器的地址。

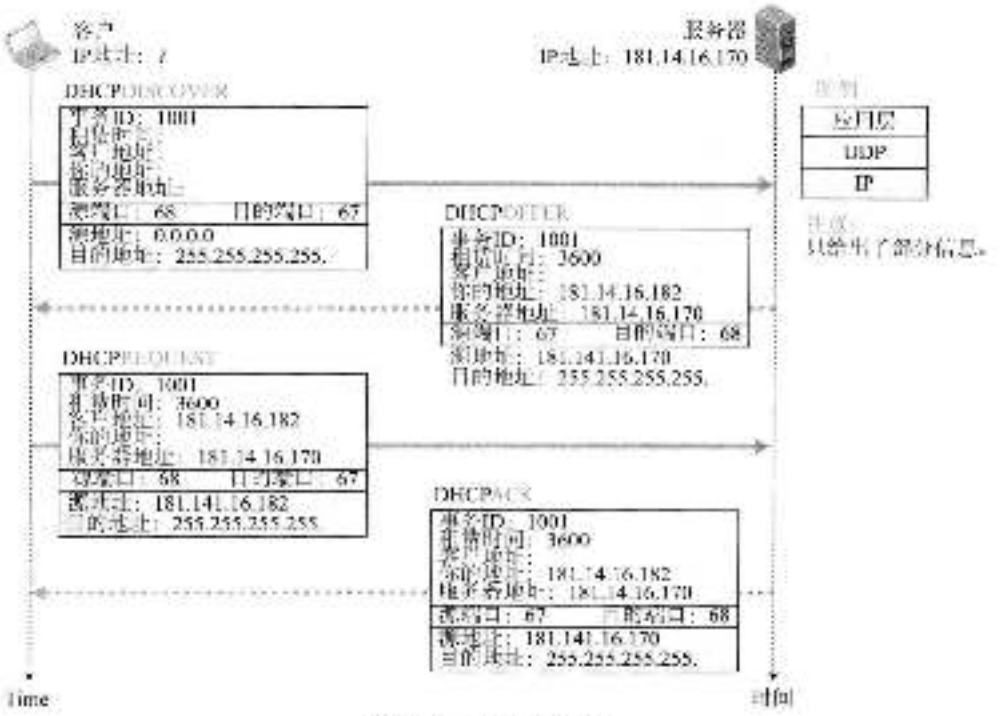


图4-40 DHCP的运行

2. DHCP服务器（如果多于一个）以DHCPOFFER报文响应，其中你的IP地址字段定义了连接主机的IP地址。服务器IP地址字段包含服务器的IP地址。这个报文也包含主机可以拥有这个IP地址的租赁时间。这个报文被封装到一个用户数据报中，它带有相同的端口号，但是顺序相反。用户数据报依次被封装到服务器地址是源IP地址的数据报中，但是目的地址是广播地址。这样服务器允许其他DHCP服务器接收这个提供（offer），如果可以的话它们就提供一个更好的。

3. 连接主机接收到一个或多个提供并从中选择出最好的一个。之后连接主机向给出最佳提供的服务器发送一个DHCPREQUEST报文，带有已知数值的字段被设置。报文被封装在用户数据报中，其中端口号是第一个报文的端口号。用户数据报被封装在IP数据报中，源地址设为新的客户地址，但目的地址仍然是广播地址，这样使得其他服务器得知它们的提供没有被接受。

4. 最后，如果提供的IP地址有效，被选定的服务器以DHCPACK报文响应客户端。如果服务器没有保存这个提供（例如，如果地址被提供给另一个处于中间的主机），那么服务器就会发送一个DHCPNACK报文且客户需要重复这个过程。这个报文也被广播，使得其他服务器知道请求被接收或者被拒绝。

两个熟知端口号

我们说过DHCP使用两个熟知端口号（68和67）而不使用一个熟知端口号和一个临时端口号。选择68这个熟知端口号而没有为客户选择临时端口号的原因是从服务器到客户的响应是广播。请记住，带有有限广播报文的IP数据报被传递到网络的每一个主机上。现在假设有一个DHCP客户和一个DAYTIME客户，例如，它们都等待接收一个来自相应服务器的响应，并且恰好同时使用了相同的临时端口号（例如56017）。两个主机都接收到了来自DHCP服务器的响应报文并将报文传送到它们的客户。DHCP客户处理报文；DAYTIME客户被接收到的奇怪报文完全搞糊涂了。使用

熟知端口号可以避免这个问题的发生。来自 DHCP 服务器的响应报文不会传递给 DAYTIME 客户，因为 DAYTIME 客户运行在端口号 56017 上而不是 68。临时端口号是从另一个范围内选择的，不是从熟知端口号中选择。

好奇的读者可能会问，如果两个 DHCP 客户同时运行会发生什么。这可能在停电或电力恢复后发生。在这种情况下，报文可以通过事务 ID 进行区分，从而将响应区分开。

使用 FTP

服务器并不发送连接网络的客户所需的所有信息。在 DHCPACK 报文中，服务器定义了文件的路径名，其中客户可以找到完整的信息，例如 DNS 服务器地址。客户之后可以使用文件传输协议来获取其余所需信息。

差错控制

DHCP 使用 UDP 服务，这是不可靠的。为了提供差错控制，DHCP 使用两个策略。首先，DHCP 要求 UDP 使用校验和。正如我们在第 3 章所见的，UDP 中校验和是可选的。第二，如果 DHCP 客户没有接收到 DHCP 对请求的回应，那么它使用计时器以及重传策略。然而，为了防止几个主机需要重传请求时（例如，在停电之后）造成的拥塞，DHCP 迫使客户使用随机数来设置它的计时器。

传输状态

我们讨论的前一个 DHCP 的运行情况非常简单。为了提供动态地址分配，DHCP 客户像是一台状态机，执行着从一个状态到另一个的转换，这要取决于它接收或发送的报文。图 4-41 给出带有主要状态的转换图。

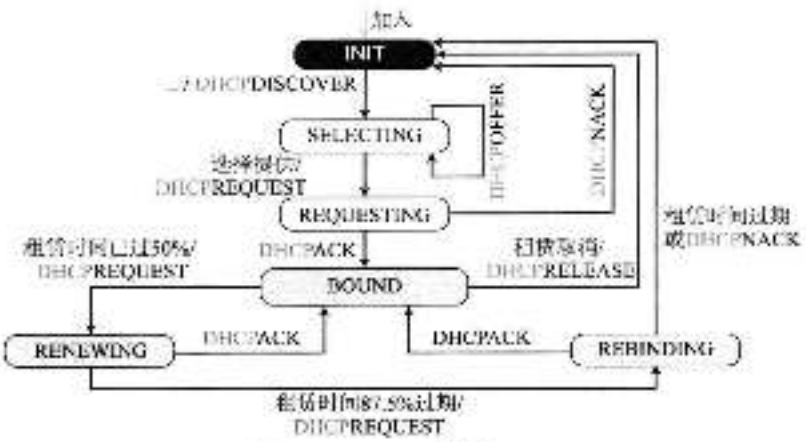


图 4-41 DHCP 客户的 FSM

当 DHCP 客户第一次启动时，它处于 INIT 状态（初始化状态）。客户广播一个发现报文。当它接收到一份提供，客户进入 SELECTING 状态。当它处于这个状态，它可能获得更多的提供。在它选择一个提供之后，它发送一个请求报文并进入 REQUESTING 状态。如果当客户处于这个状态时，一个 ACK 到达，那么它就进入 BOUND 状态并使用 IP 地址。当租赁时间经过了 50%时，客户试图通过进入 RENEWING 状态来更新。如果服务器更新租赁，客户再次进入 BOUND 状态。如果租赁没有被更新且租赁时间经过了 75%，客户将进入 REBINDING 状态。如果服务器同意租赁（ACK 报文到达），客户进入 BOUND 状态并继续使用 IP 地址；否则，客户进入 INIT 状态并请求另一个 IP 地址。注意，仅当客户处于 BOUND、RENEWING 或 REBINDING 状态时，才可以使用 IP 地址。以上流程要求客户使用三个计时器：更新计时器（renewal timer）（设为租赁时间的 50%）、重新绑定计时器（rebinding timer）（设为租赁时间的 75%）以及过期计时器（expiration timer）（设置为租赁时间）。

NAT

通过 ISP 来分配地址造成了一个新的问题。假设一个 ISP 给一个小公司或家庭用户分配了一小范围地址。如果公司成长或家庭用户需要更大的地址范围，ISP 可能无法满足需要。因为这个范围之前或之后的地址可能已经被分配给其他网络了。然而，在多数情况下，一个小网络中只有一小部分计算机同时需要访问因特网。这意味着，分配地址的数量不必非要匹配网络中计算机的数量。例如，假设在小型公司中，最多有 20 台电脑，但是只有 4 台会同时访问因特网。绝大多数计算机或者正在做不需要因特网访问的工作或者正在做不需要相互通信的工作。这个小公司可以使用 TCP/IP 协议作为内部的以及通用的通信协议。这个公司可以将 20（或 25）个私有块（之前讨论过）中的地址用于内部通信；5 个用于通用通信的地址可以由 ISP 分配得到。

一种可以提供私有地址和通用地址之间映射，同时支持虚拟私有网络的技术是网络地址转换（network address translation，NAT）。虚拟私有网络我们在第 10 章会讨论到。这个技术允许一个站点使用一组私有地址以及一组全局因特网地址（至少一个）来与世界上其余的计算机进行通信。这个站点必须有一条与到全局因特网相连接的链路。这条链路穿过了运行着 NAT 软件的支持 NAT 功能的路由器。图 4-42 给出了 NAT 的一个简单实现。

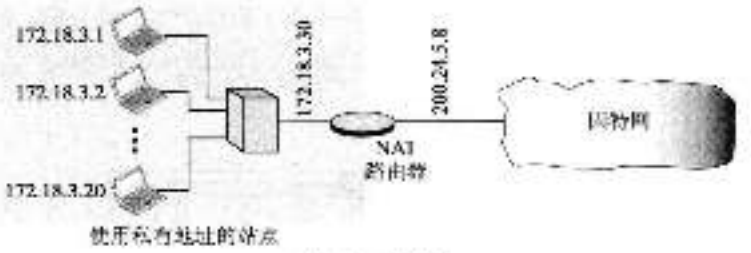


图 4-42 NAT

如图 4-42 所示，私有网络使用私有地址。将这个网络与全局地址连接在一起的路由器使用一个私有地址和一个全局地址。私有网络看不到因特网的其余部分；因特网的其余部分只能看到带有地址 200.24.5.8 的 NAT 路由器。

地址转换

所有通过 NAT 路由器发出去的分组都将分组中的源地址替换为全局 NAT 地址。所有通过 NAT 路由器进入的分组都将目的地址（NAT 路由器的全局地址）替换成适当的私有地址。图 4-43 给出一个地址转换的例子。

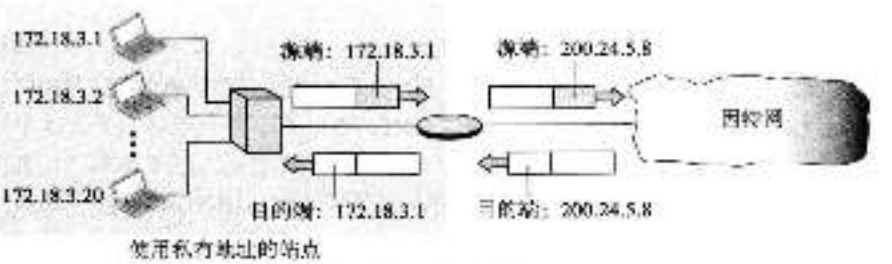


图 4-43 地址转换

转换表

读者可能注意到了，将源地址转换成为外发地址是一件简单的事情。但是，NAT 路由器如何

知道来自因特网的分组的目的地址呢？可能有数十个或者数千个私有 IP 地址，每个都各自属于特定的主机。如果 NAT 路由器有一个转换表，那么这个问题就可以得到解决。

使用一个 IP 地址 在最简单的情况下，一个转换表只有两列：私有地址和外部地址（分组的目的地址）。当路由器转发外发分组的源地址时，它也记录其目的地址——分组去往的地方。当响应从目的端返回时，路由器使用分组的源地址（作为外部地址）来找出分组的私有地址。图 4-44 给出了这种思想。

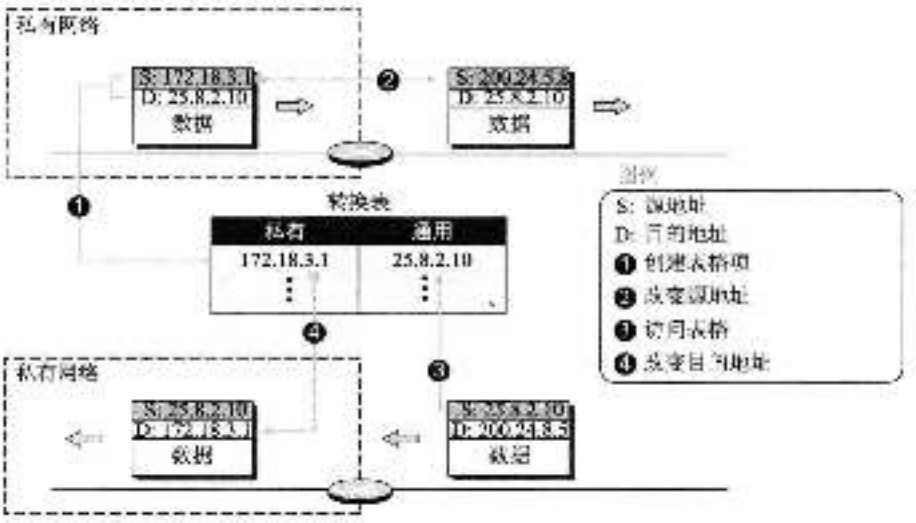


图 4-44 转换

在这个策略中，通信必须由私有网络发起。前面所表述的 NAT 机制要求私有网络发起通信。正如我们所看到的那样，NAT 主要由 ISP 使用，ISP 为每个用户分配一个单一的地址。然而，用户可能是一个具有许多私有地址的私有网络的一员。在这种情况下，与因特网的通信通常总是由用户站点发起的，并使用诸如 HTTP、TELNET 或 FTP 这类的客户端程序来访问相应的服务器程序。例如，当 ISP 的电子邮件服务器接收到由一个来自网络外部站点的电子邮件时，电子邮件就存储在用户帐户中，直到诸如 POP 这类的协议获取这个邮件。

使用一个 IP 地址池 NAT 路由器只使用一个全局地址，从而只允许一个私有网络主机访问一个给定的外部主机。为了去除这个限制，NAT 路由器可以使用一个全局地址池。例如，NAT 路由器可以使用四个地址（200.24.5.8、200.24.5.9、200.24.5.10 以及 200.24.5.11），而不是使用唯一的全局地址（200.24.5.8）。在这种情况下，四个私有网络主机可以同时与相同的外部主机通信，因为每一对地址定义了一个独立的连接。然而，这还是存在一些不足。不能与同样的目的端建立四条以上的连接；没有一台私有网络主机可以同时访问两个外部服务器程序（如 HTTP 和 TELNET）。而且类似地，没有一台私有网络主机能够同时访问两个外部服务器程序（如 HTTP 或 TELNET）。

同时使用 IP 地址和端口号 为了能够在私有网络主机和外部服务器程序之间建立多对多关系，需要获取转换表中更多的信息。例如，假定在一个私有网络中有两台主机，地址分别为 172.18.3.1 和 172.18.3.2，需要访问一台地址为 25.8.3.2 的外部主机上的 HTTP 服务器。如果转换表有 5 列，而不是 2 列，其中包括传输层协议的源和目的端口号，这样就可以消除二义性。表 4-1 给出下表的一个例子。

注意，当来自 HTTP 服务器的响应返回时，目的地址（25.8.3.2）和目的端口号（1401）的组合定义了接收这一响应的内部网络主机。还须注意，要使这种转换能正常运作，临时端口号（1400

和 1401）必须是唯一的。

表 4-1 5 列转换表

私有地址	私有端口	外部地址	外部端口	传输协议
172.18.3.1	1400	25.8.3.2	80	TCP
172.18.3.2	1401	25.8.3.2	80	TCP
⋮	⋮	⋮	⋮	⋮

4.2.3 IP 分组的转发

我们在本章早些时候讨论过网络层转发的概念。在本节，我们将这个概念扩展到 IP 地址在转发中的作用。正如我们之前讨论过的，转发意味着将分组放在通往目的地的路由上。由于如今的因特网由链路（网络）组成，转发意味着将分组传递到下一跳（这可能是最终目的地或中间的连接设备）。尽管 IP 协议原先被设计成无连接协议，但如今网络上的延迟将促使其成为面向连接协议。我们将讨论这两种情况。

当 IP 用作无连接协议时，转发是基于 IP 数据报的目的地址；当 IP 用作面向连接协议时，转发是基于附加到 IP 数据报上的标签。

基于目的地址的转发

我们首先讨论基于目的地址的转发。这是一种传统的方法，今天也很流行。在这种情况下，转发要求主机或路由器有转发表，它查看转发表来找到分组的下一跳。

在无类寻址中，整个地址空间是一个实体；没有类。这意味着转发需要得到每个相关块中的一行信息。这个转发表需要基于网络地址（块中的第一个地址）进行搜索。不幸的是，分组中的目的地址没有给出网络地址的线索。为了解决这个问题，我们需要在表格中包含掩码（/n）。换言之，无类转发表需要包含四项信息：掩码、网络地址、接口号码以及下一个路由器的 IP 地址（正如我们在第 5 章讨论的，需要找到下一跳的链路层地址）。然而，我们经常在文献中看到前两项信息是组合在一起的。例如，如果 n 是 26 并且网络地址是 180.70.65.192，那么一个信息可以与另一个信息组合在一起：180.70.65.192/26。图 4-45 给出一个只有三个接口的路由器的简单转发模块和转发表。

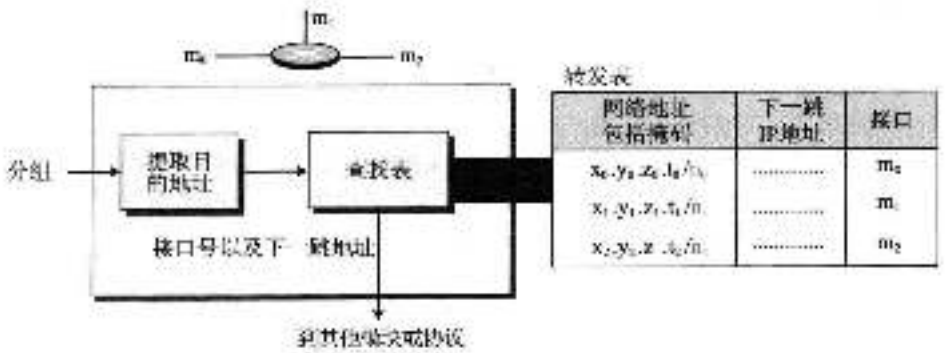


图 4-45 无类地址的简单转发模块

转发模块的工作是一行一行地查找表。在每一行中，目的地址的最左边 n 位（前缀）被记录下来，且其余位（后缀）被设置为 0。如果结果地址（我们称其为网络地址）与第一列的地址匹配，后两列的信息就被抽取出来；否则继续查找表格。通常最后一行在第一列有一个默认值（图中没有给出），这表示所有目的地址都不匹配之前的行。

有时，文献明确地给出最左边 n 位的数值应该与目的地址的最左边 n 位数值相匹配。概念是相

同的，但是表述是不同的。例如，我们不给出 180.70.65.192/26 的网络掩码组合，取而代之的是，我们可以给出如下最左侧 26 位的数值。

10110100 01000110 01000001 11

注意，我们仍然需要使用一种算法来找到前缀并将其与位模式进行比较。换言之，算法还是需要的，但是表述是不同的。当我们使用更小的地址空间做练习时，我们将在转发表单使用这种格式。

例 4.7 使用图 4-46 中 R1 的配置制作转发表。

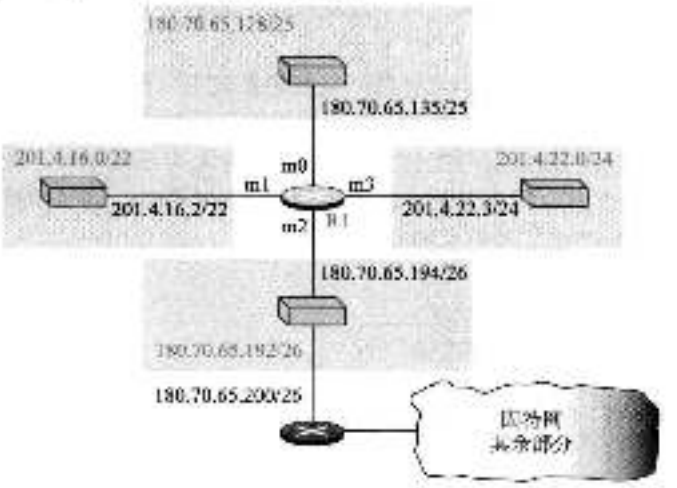


图 4-46 例 4.7 配置

解答

表 4-2 给出了相应的表格。

表 4-2 图 4-46 中路由器 R1 的转发表

网络地址/掩码	下 一 跳	接 口
180.70.65.192/26	—	m2
180.70.65.128/25	—	m0
201.4.22.0/24	—	m3
201.4.16.0/22	—	m1
默认	180.70.65.200	m2

例 4.8 我们可以使用表 4-3 来取代表 4-2，其中网络地址/掩码以位为单位给出。

表 4-3 图 4-46 中使用前缀位的路由器 R1 的转发表

网络地址/掩码	下 一 跳	接 口
10110100 01000110 01000001 11	—	m2
10110100 01000110 01000001 1	—	m0
11001001 00000100 0001110	—	m3
11001001 00000100 000100	—	m2
默认	180.70.65.200	m2

当目的地址最左侧 26 位与第一行匹配的分组到达，这个分组被从接口 m2 发出。当目的地址最左侧 25 位与第二行匹配的分组到达，这个分组被从接口 m0 发出，等等。表格很清楚地给出了第一行有最长的前缀，第四行有最短的前缀。较长前缀意味着一个较小的地址范围；较短前缀意味

若较长的地址范围。

例 4.9 图 4-46 给出分组到达 R1 时的转发过程，其中目的地址是 180.70.65.140。

解答

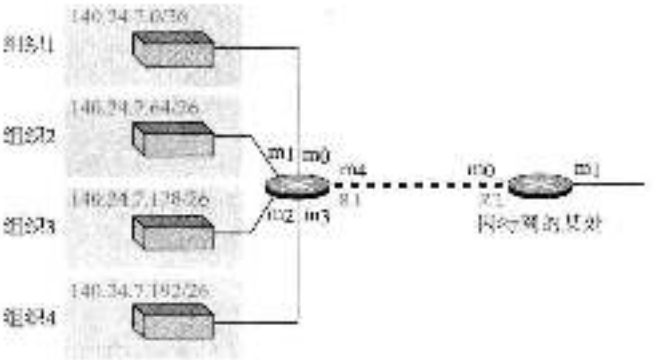
路由器执行如下步骤：

1. 第一个掩码（/26）应用于目的地址。结果是 180.70.65.128，这与相应的网络地址不匹配。

2. 第二个掩码（/25）应用于目的地址。结果是 180.70.65.128，这与相应的目的地址相匹配。下一跳地址以及接口号 m0 被抽取出来，用于转发分组（见第 5 章）。

地址聚合

当我们使用分类寻址时，每个组织之外的站点在转发表中只有一个表项。表项定义了站点，即使站点是被子网化的。当分组到达路由器时，路由器检查相应的表项并根据它来转发分组。当我们使用无类寻址时，转发表项的数量有可能会增加。这是因为无类寻址的目的是将整个地址空间分割成可管理的块。表格大小的增加导致了搜索表格所消耗时间的增加。为了缓解这个问题，地址聚合的思想被提出来。在图 4-47 中，我们有两个路由器。



R1的转发表

网络地址/掩码	下一跳地址	接口
140.24.7.0/26	----------	m0
140.24.7.64/26	----------	m1
140.24.7.128/26	----------	m2
140.24.7.192/26	----------	m3
0.0.0.0/0	address of R2	m4

R2的转发表

网络地址/掩码	下一跳地址	接口
140.24.7.0/24	----------	m0
0.0.0.0/0	default router	m1

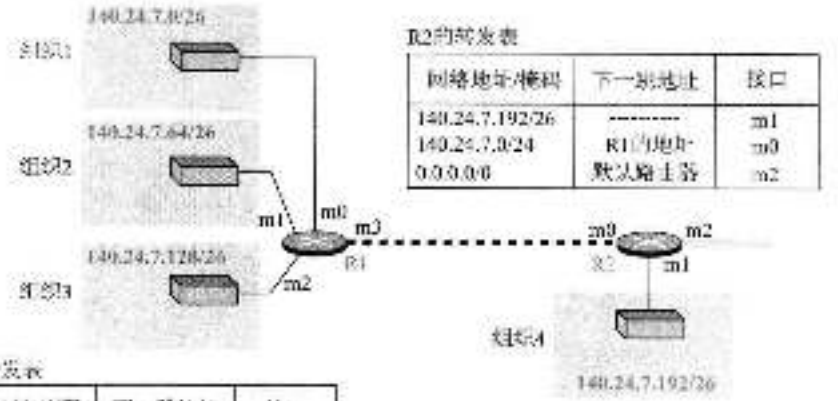
图 4-47 地址聚合

R1 被连接到四个组织上，每个组织使用 64 位地址。R2 位于远离 R1 的某地。R1 有一个较长的转发表，因为每个分组必须被正确地路由到适当的组织。另一方面，R2 可以有一个非常小的转发表。对于 R2，任何目的地址在 140.24.7.0 到 140.24.7.255 之间的分组被从端口 m0 发出，不管组织号是多少。这称为地址聚合，因为四个组织的地址块被聚合成一个更大的块。如果每个组织有一个不能聚合进一个块的地址，那么 R2 将有一个更长的转发表。

最长掩码匹配

如果在图 4-47 中的一个组织离其他三个组织的地理位置很远，那么将发生什么？例如，如果由于相同的原因，组织 4 不能连接到路由器 R1 上，那么我们仍然能够使用地址聚合的思想并将块 140.24.7.192/26 分配到组织 4 吗？回答是肯定的。因为无类寻址的路由使用另外一个原则——最长掩码匹配（longest mask matching）。这个原则表明转发表按照最长掩码到最短掩码的顺序来存储。换言之，如果有三个掩码，/27、/26 和/24，那么掩码/27 必须是第一个表项，并且/24 必须是最后一个表项。让我们来看看这个原则是否可以解决组织 4 与其他三个组织分离的问题。图 4-48 给出

了这种情境。



R2的转发表

网络地址/掩码	下一跳地址	接口
140.24.7.192/26	--------	m1
140.24.7.0/24	R1的地址	m0
0.0.0.0/0	默认路由器	m2

R1的转发表

网络地址/掩码	下一跳地址	接口
140.24.7.0/26	--------	m0
140.24.7.64/26	--------	m1
140.24.7.128/26	--------	m2
0.0.0.0/0	默认路由器	m3

图 4-48 最长掩码示例

假设一个要到达组织 4 的分组到达路由器 R2，它的目的地址为 140.24.7.200。路由器 R2 的第一个掩码被应用，它给出网络地址 140.24.7.192。分组被从端口 m1 正确路由并到达组织 4。然而，如果转发表并不是按照最长前缀优先的顺序存储的，那么应用/24 这个掩码将会把分组错误地路由到路由器 R1。

分层路由（hierarchical routing）

为了解决转发表过大的问题，我们可以在路由表中创建层次结构。在第 1 章，我们提到过现今的因特网具有层次结构。我们说过因特网被分成骨干网和国内 ISP。国内 ISP 又被划分为地区 ISP，地区 ISP 进一步划分为本地 ISP。如果转发表也像因特网那样具有层次结构，那么可以降低转发表的长度。

让我们考虑本地 ISP 的情况。本地 ISP 可以被分配一个但是很大的地址块，而该块具有某一前缀长度。本地 ISP 将这个块划分为不同大小的一些小块。如果分配给本地 ISP 的地址块以 a.b.c.d/n 开始，那么这个 ISP 可以建立以 e.f.g.h/m 开始的一些块，其中 m 大于 n，而对每个客户，m 是可变的。

这如何降低路由表的长度呢？因特网的其余部分不必知道具体划分方案。对因特网其余部分来说，本地 ISP 的所有客户都被定义为 a.b.c.d/n。在这个大块中的每一个分组的目的地址都路由到本地 ISP。对于所有这些客户来说，世界上的每个路由器都只有一项，它们都属于同一组。当然，在本地 ISP 内部，路由器必须识别出子块并路由到目的客户。如果某一客户是一个较大的组织机构，那么它也可以通过子网化和划分它的子块为更小的子块（或子块的子块）建立另一个层次。在无类路由选择中，只要我们遵循无类寻址规则，层次的级是没有限制的。

例 4.10 作为分层路由的例子，让我们考虑图 4-49。区域 ISP 被授予以地址 120.14.64.0 开始的 16 384 个地址。区域 ISP 决定将这个块划分成 4 个子块，每块 4096 个地址，其中的 3 个子块分别指派给 3 个本地 ISP，而第 2 个子块保留做将来使用。注意，由于原始块的掩码是/18，因此每块的掩码是/20。

第一个本地 ISP 将分配的子块分成了 8 个更小的块。每一个指派给一个小的 ISP。每个小的 ISP 对 128 个家庭（11001 到 11128）提供服务，每个家庭使用 4 个地址。注意，现在由于进一步划分成 8 个子块，因此每个小 ISP 的掩码是/23。因为一个家庭只有 4 个地址（$2^{32-30}-4$），所以每个家庭的掩码是/30。第二个本地 ISP 将它的块分成 4 个小块，并将地址分配到 4 个大型组织机构（LOrg01 到 LOrg04）。注意，每个大型组织机构有 1024 个地址，掩码是/22。

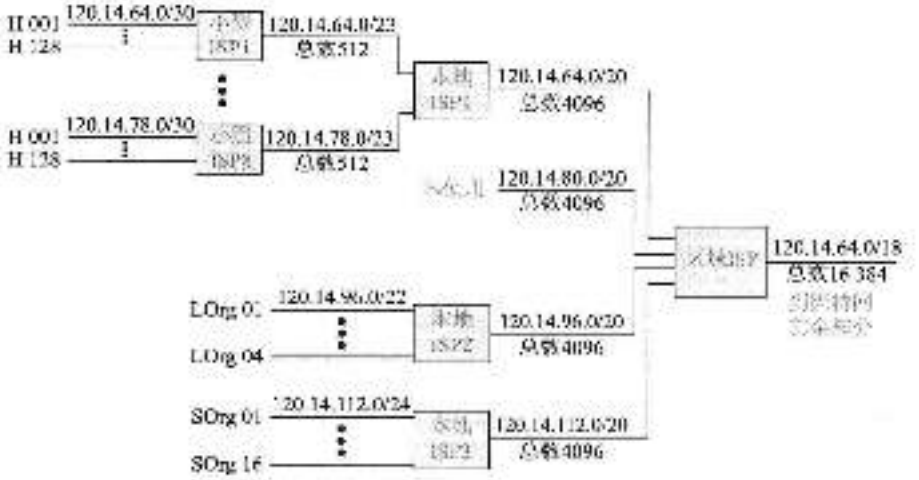


图 4-49 ISP 的分层路由

第三个本地 ISP 将它的块分成了 16 个子块，并将每个子块分配给一个小型的组织机构（标记 SOrg01 到 SOrg16）。每一个小型的组织机构有 256 个地址，掩码是/24。在这种配置下，存在层次结构。在互特网上的所有路由器将带有目的地址属于 120.14.64.0 到 120.14.127.255 之间的分组发送到区域 ISP。区域 ISP 向本地 ISP1 发送目的地址属于 120.14.64.0 到 120.14.79.255 之间的分组。本地 ISP1 向 H001 发送目的地址属于为 120.14.64.0 到 120.14.64.3 之间的分组。

处理路由

为了进一步减少路由表的大小，我们需要扩展分层路由，使其包含宏观路由。我们必须将整个地址空间分成几个大块。我们将一个大块分配给美洲、一个大块给欧洲、一个大块给亚洲、一个大块给非洲等。欧洲以外 ISP 的路由器在它们的路由表中将只有到欧洲的一个项。在美洲以外的 ISP 路由器在它们的路由表中只有到美洲的一个项，等等。

转发表搜索算法

在无类寻址中，目的地址不含有网络信息。一种最简单但不是最高效的搜索算法称为最长前缀匹配（正如我们之前讨论的）。转发表可以被分成多个桶，每个桶一个前缀。路由器首先尝试最长前缀。如果目的地址在这个桶中找到，那么搜索完成。如果地址没有被找到，搜索下一个前缀，等等。很明显这类搜索会花费很长时间。

一种解决方法是改变用于搜索的数据结构。我们可以使用其他数据结构（例如树或二叉树）取代列表。一种候选结构是 trie（一种特殊的树）。然而，这个讨论超出了本书的范围。

基于标签的转发

在 20 世纪 80 年代，人们努力将 IP 改变成为面向连接协议，在面向连接协议中，路由被交换所替代。正如我们在本章前面讨论过的，在无连接网络中（数据报方法），路由器基于分组头部的目的地址转发分组。另一方面，在面向连接网络中（虚电路方法），交换机基于附加在分组上的标签来路由分组。路由通常是基于对表格内容的搜索；而通过使用索引访问表格可以完成交换。换言之，路由包含了搜索；交换包含了访问。

例 4.11 图 4-50 给出一个使用最长掩码算法在转发表中搜索的简单例子。尽管现在有更高效的算法，但是原理是相同的。

当转发表算法得到分组的目的地址时，它需要深入到掩码列。对每一个表项，需要用掩码来找到目的网络地址。之后，需要检查表格中的网络地址直到找到匹配地址。之后路由器取出下一跳地址

址以及接口号，传送到交换结构后。

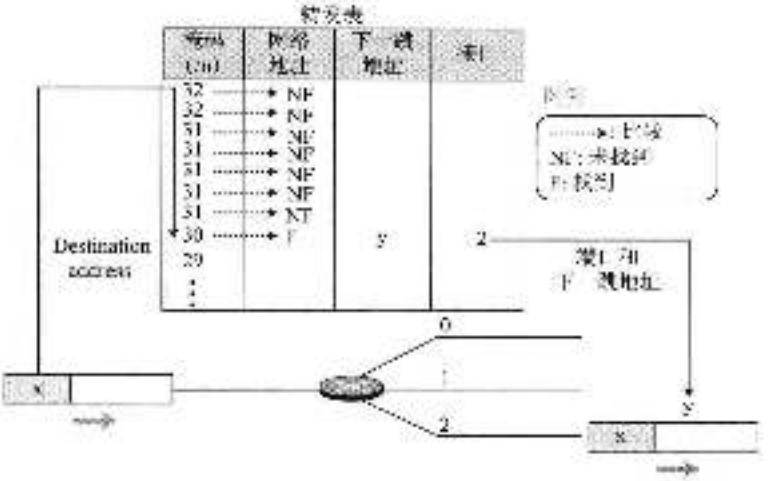


图 4-50 例 4.11：基于目的地址的转发

例 4.12 图 4-51 给出了使用标签访问交换表的简单例子。因为标签被作为表格的索引来使用，因此可以直接寻找表格中的信息。

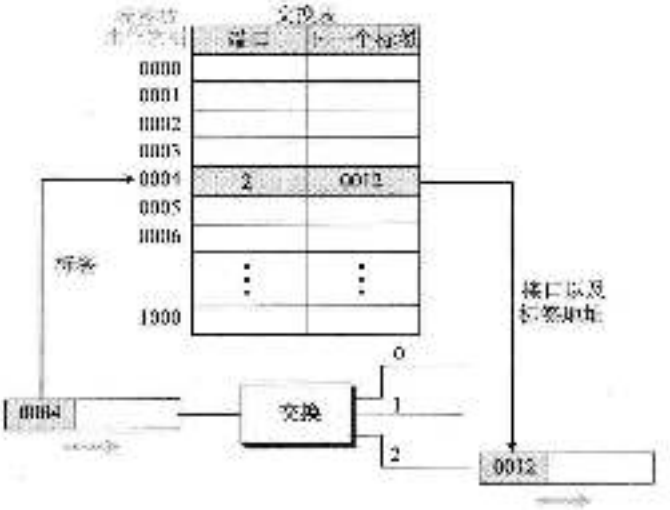


图 4-51 例 4.12：基于标签的转发

多协议标记交换（Multi-Protocol Label Switching，MPLS）

在 20 世纪 80 年代，几个厂商创建了执行交换技术的路由器。之后 IETF 批准了一个标准，称为多协议标记交换。在这项标准中，因特网中的一些传统路由器可以被 MPLS 路由器所替代，它可以表现得像一台路由器和一台交换机的联合体。当它按照一台路由器去工作时，MPLS 可以基于目的地址来转发分组；当它按照一台交换机去工作时，它可以基于标签来转发分组。

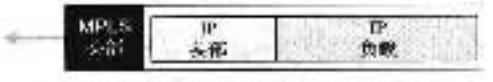


图 4-52 加入 IP 分组的 MPLS 头部

一个新的头部

为了模拟面向连接交换，在使用 IP 协议时，第一件事就是在分组中加一个字段，这个将加入的字段携带着后面将要讨论的标签。IPv4 分组格式不

允许这种扩展（尽管如我们后文会讨论到的，IPv6 分组格式中提供这个字段）。解决方法是将 IPv4 分组封装到 MPLS 分组中（尽管 MPLS 位于数据链路层和网络层之间）。整个 IP 分组作为负载被封装到一个 MPLS 分组中，并且加入一个 MPLS 头部。图 4-52 给出了这种封装。

MPLS 头部实际上是一堆子头部，正如我们马上要讨论到的，它用来进行多级分层交换。图 4-53 给出一个 MPLS 头部的格式，其中每个子头部是 32 位长（4 字节）。

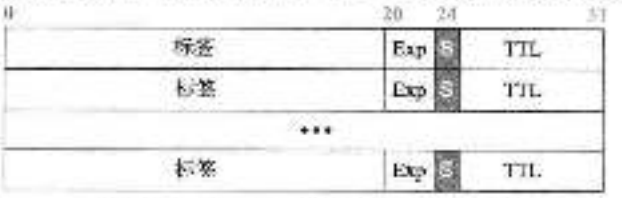


图 4-53 由一堆标签组成的 MPLS 头部

以下是每个字段的简要描述：

- **标签**。这个 20 位字段定义了路由器中索引转发表的标签。
- **Exp**。这个 3 位字段被保留用于实验目的。
- **S**。1 位堆字段定义了堆中子头部的情况。当这个位是 1 时，意味着这个头部是堆中最后一个。
- **TTL**。这个 8 位字段与 IP 数据报中的 TTL 字段类似。每个被访问的路由器减少这个字段的值。当这个字段的值变为 0 时，分组被丢弃以防止形成循环。

分层交换

MPLS 中的一堆标签允许分层交换。这与传统的分层路由选择相似。例如，带有两个标签的分组可以使用顶层标签通过组织之外的交换机转发分组；底层标签可以用来在组织内部将分组路由到目的子网。

4.2.4 ICMPv4

IPv4 没有差错报告或差错更正机制。如果出现某些差错将会发生什么？如果路由器因为找不到通往最终目的地的路径，或者因为生存时间字段是 0 而必须丢弃一个数据报时，将会发生什么？如果最终目的主机在预先设置的时间内不能收到所有的数据报分段，而将一个数据报的全部分段都丢弃时，又会发生什么？这些例子中，都出现了差错但是 IP 协议没有内在机制可以通知发送该数据报的主机。

IP 协议还缺少主机和管理方面的查询机制。主机有时需要确定一个路由器或另一个主机是否处于活跃状态。有时网络管理员需要从另一个主机或路由器得到信息。

因特网控制报文协议（Internet Control Message Protocol version 4，ICMPv4）就是为了弥补上述两个缺点而设计的。它是配合 IP 协议使用的。ICMP 自身是网络层协议。然而，它的报文并不像预期地那样直接传递给数据链路层。在进入较低层之前，报文首先被封装到 IP 数据报中。当一个 IP 数据报封装了 ICMP 报文，IP 数据报中的协议字段值就被设为 1，这表示 IP 负载是一个 ICMP 报文。

报文

ICMPv4 报文分为两大类：差错报告报文（error-reporting message）和查询报文（query message）。差错报告报文向路由器或主机（目的端）报告在处理一个 IP 数据报时可能碰到的一些问题。查询报文是成对出现的，它帮助主机或网络管理员从一个路由器或一个主机得到特定的信息。例如，主机能够发现它们的邻居主机或网络中的路由器，并且路由器可以帮助结点改变报文的路由。

报文格式

ICMPv4 报文有一个 8 字节的头部和一个可变长的数据部分。虽然每一种报文类型的头部的格式都是不同的，但是前面的 4 个字节对所有报文类型来说都是相同的。如图 4-54 所示，第一个字段是 ICMP 类型，它定义报文的类型。代码字段指定了发送此特定报文类型的原因。最后一个共同的字段是校验和字段。头部其余部分对每种报文类型都是特定的。差错报文数据部分所携带的信息

可能出引起差错的原始分组。查询报文的数据部分携带了基于查询类型的额外信息。

差错报告报文

由于 IP 是不可靠协议，ICMP 的主要职责之一就是报告 IP 数据报处理过程中的差错。ICMP 并不更正差错，它仅仅报告差错。差错纠正留给了高层协议。差错报文总是被发给源端，因为在数据报中关于路由唯一可用的信息就是源 IP 地址和目的 IP 地址。ICMP 使用源 IP 地址将差错报文发送给数据报的源端（发送方）。为了简化差错报告，ICMP 遵循报告报文中的一些规则。首先，对于含有多播或特殊地址（例如本机或网络地址）的数据报，不会产生差错报文。其次，关于携带 ICMP 差错报文的数据报，不会再产生差错报文。第三，对于分段的数据报文，如果不是第一个分段则不会产生差错报文。

注意，所有差错报文都包含一个数据部分，它包含了原始数据报的 IP 头部加上数据报中前 8 个字节的数据。原始数据报中的头部被加入是为了向原始的源端给出数据报本身的信息，正是这个源端接收差错报文。附加 8 个字节的数据是因为这前 8 个字节提供了关于端口号（UDP 和 TCP）和序号（TCP）的信息。这些信息是必需的，这样源端可以将差错情况通知给这些协议（TCP 或 UDP）。

使用最广泛的差错报文是目的端不可达报文（destination-unreachable message）（类型 3），这个报文使用不同的代码（0 到 15）来定义差错报文的类型和数据报不能到达目的端的原因。例如代码 0 向源端告知目的端不可达。例如，当我们使用 HTTP 协议来访问网页，但服务器死机时可能出现这个问题。报文“目的主机不可达”被创建并被发回到源端。

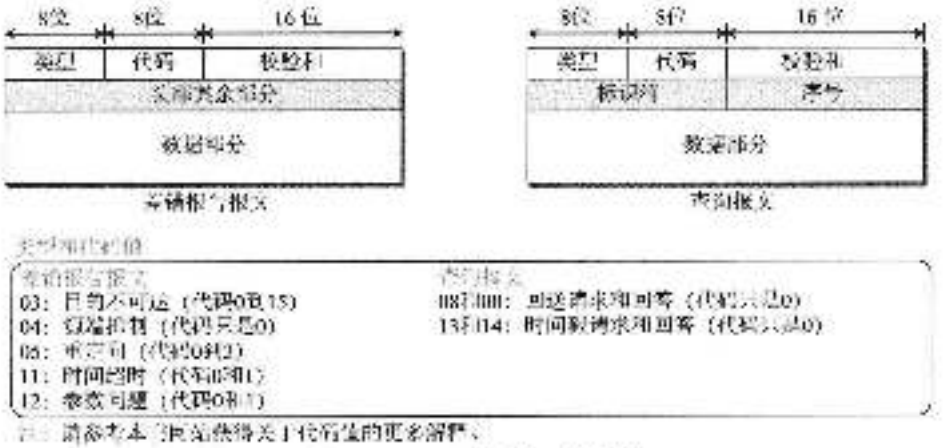


图 4-54 ICMP 报文的一般格式

另一个报文称为源端抑制（source quench）（类型 4）报文，它通知发送端，网络出现拥塞并且数据报被丢弃；源端需要减慢发送速率。换言之，通过使用这种报文，ICMP 将一种流控制机制加入到 IP 协议。

当源端使用一个错误的路由器来发送报时，它会用到重定向（redirection message）报文（类型 5）。路由器将报文重定向到适当的路由器，但是通知源端今其以后改变默认路由器。默认路由器的 IP 地址在这个报文中被发送。

我们讨论过 IP 数据报中生存时间（time-to-live，TTL）的作用，并解释了它可以防止报文在因特网中无目的地循环。当 TTL 值变为 0，数据报被它所访问的路由器丢弃并且一个代码为 0 的超时报文（time exceeded）（类型 11）被发送到源端以告知其情况。当组成一个报文的所有分段不能在某一时限内到达主机时，超时报文（代码为 1）也会被发送。

最终，当数据报头部出现问题（代码 0）或一些选项丢失或不能被解释（代码 1）时，就发送

参数问题（parameter problem）报文（类型 12）。

查询报文

有趣的是，ICMP 中的查询报文可以独立使用而与 IP 数据报无关。当然，查询报文将要作为负载被封装到数据报中。查询报文用来探测或检测互联网中主机或路由器是否处于活跃状态，也用来获取两台设备之间 IP 数据报的单向或往返时间，甚至用于检查两台设备之间的时钟是否同步。很自然地，查询报文成对出现：查询和回答。

回送请求（echo request）（类型 8）以及回送应答（echo reply）（类型 0）报文被主机或路由器使用，目的是检测另一台主机或路由器是否处于活跃状态。一台主机或路由器向另外一台主机或路由器发送一个回送请求报文，如果后者是活跃的，它就以一个回送应答报文回应。我们很快就会看到这对报文在调试工具中的应用：ping 以及 traceroute。

时间戳请求（timestamp request）（类型 13）以及时间戳回答（timestamp reply）（类型 14）这一对报文用来确定两个设备之间的往返时间或用来检查两个设备之间的时钟是否同步。时间戳请求报文发送一个 32 位数字，它定义了报文被发送的时间。时间戳回答重发那个数字，但是也包括一个新 32 位数字以及响应被发送的时间。这个新数字代表了请求被接收时间。如果所有时间戳都使用格林尼治时间，发送方可以计算单向以及往返时间。

例 4.13 主机用来检测另一台主机是否活跃的一种工具是 **ping** 程序。ping 程序利用了 ICMP 回送请求与回送应答报文。一台主机可以发送一个回送请求（类型 8，代码 0）到另一台主机，如果它活跃，它就可以发送回一个回送应答（类型 0，代码 0）报文。通过发送一组请求-回答，ping 程序也可以在一定程度上度量两个主机间路由器的可靠性以及拥塞程度。

以下给出我们如何发送一个 ping 报文到 auniversity.edu 站点。我们设置了回送请求以及回送应答中的标识符字段，并且序号从 0 开始；每当一个新的报文被发送，这个序号就增加 1。注意，ping 可以计算往返时间，它在报文的数据区内插入发送时间。当分组到达时，它用离开时间减去到达时间得到往返时间（round-trip time，rtt）。

```
$ ping auniversity.edu
PING auniversity.edu (152.181.8.3)    56 (84)  bytes of data.
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=0      ttl=62      time=1.91 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=1      ttl=62      time=2.04 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=2      ttl=62      time=1.90 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=3      ttl=62      time=1.97 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=4      ttl=62      time=1.93 ms
64 bytes from auniversity.edu (152.181.8.3): icmp_seq=5      ttl=62      time=2.00 ms
--- auniversity.edu 统计 ---
已发送 6 个分组，6 个分组都被接收到，丢弃分组为 0%
往返时间 最小时间/平均时间/最大时间 = 1.90/1.95/2.04 ms
```

例 4.14 UNIX 中的 **traceroute** 程序或 Windows 中的 **tracert** 程序可用于追踪一个分组从源到目的点所经过的路由。它可以找到沿途访问的所有路由器的 IP 地址。程序通常被设置为最多检查 30 跳（路由器）。因特网中的跳数通常小于这个数值。traceroute 程序与 ping 程序不同。ping 程序从两条查询报文中获得帮助；traceroute 程序从两条差错报告报文中获得帮助：超时报文和目的端不可达报文。traceroute 是应用层程序，但是它只有客户端程序。因为正如我们所见，客户程序永远不会到达目的主机的应用层。换言之，没有 traceroute 服务器程序。traceroute 应用程序被封装到 UDP 用户数据报中，但是 traceroute 有意使用一个目的端不可用的端口号。如果路径上有 n 个路由器，traceroute 程序发送 $(n+1)$ 个报文。前 n 个报文被 n 个路由器丢弃，每个路由器丢弃一个报文；最后一个报文被目的主机丢弃。traceroute 客户程序使用接收到的前 $(n-1)$ 个 ICMP 差错报告报文来找到路由器中间的路径。我们将很快给大家展示出 traceroute 不需要知道 n 的数值；它是自动获得的。图 4-55 给出一个 $n=3$ 的例子。

第一个 traceroute 报文生存时间（TTL）设置为 1；这个报文被第一个路由器丢弃并且超时 ICMP 差错报文被发送，traceroute 程序从这个差错报文中可以找到第一个路由器的 IP 地址（差错报文的源 IP 地址）以及路由器名（报文的数据区）。第二个 traceroute 报文 TTL 设置为 2，它可以找到第二个路由器的 IP 地址和名称。类似地，第三个报文可以得到路由器 3 的信息。然而，第四个报文到达了目的主机。由于另一个原因，这个主机也丢弃这个报文。目的主机不能找到 UDP 用户数据报中指定的端口号，这次 ICMP 发送一个不同的报文，代码为 3 的目的不可达报文表示端口未找到。在接收到这个不同的 ICMP 报文之后，traceroute 程序知道它到达了最终目的端。它使用接收到的目的报文中的信息来获得 IP 地址和最终目的端的名称。

traceroute 程序也设置了一个计时器来得到每个路由器和目的端的往返时间，绝大多数 traceroute 程序向每个设备发送三个报文，每个报文都有相同的 TTL 值，这样做是为了得到更好的往返时间估计值。以下给出 traceroute 程序的一个例子，它对每个设备使用三次探测并得到了三个 RTT。

```
$ traceroute printers.com
traceroute to printers.com (13.1.69.93), 30 hops max, 38 byte packets
 1  route.front.edu      (153.18.31.254)   0.622 ms   0.891 ms   0.875 ms
 2  generic.net          (137.164.32.140)  3.069 ms   2.875 ms   2.930 ms
 3  satire.net           (132.16.132.20)   3.071 ms   2.876 ms   2.929 ms
 4  alpha.printers.com   (13.1.69.93)      5.922 ms   5.048 ms   4.922 ms
```

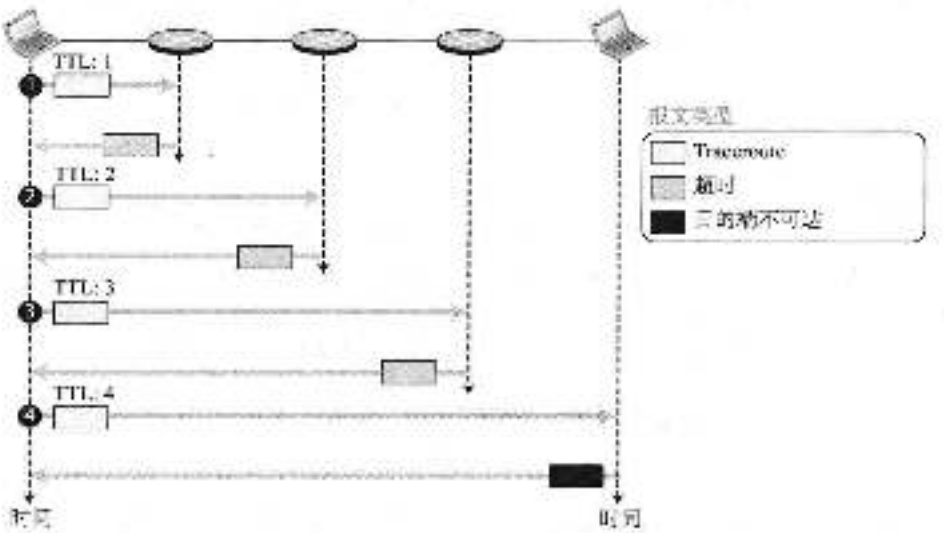


图 4-55 traceroute 程序的例子

源端和目的端之间有三跳。注意，某些往返时间看起来不同寻常。可能是路由器过于繁忙而不能立即处理分组。

4.3 单播路由选择

在互联网中，网络层的目的是从源端向它的一个或多个目的端传送数据报。如果数据报只发向一个目的端（一对一传送），我们使用单播路由选择（unicast routing）[⊖]。如果数据报只发向多个目的端（一对多传送），我们使用多播路由选择（multicast routing）。最终，如果数据报应该传送到互联网中的所有主机（一对多），那么我们使用广播路由选择（broadcast routing）。在本节和下一节，

⊖ routing 一般译为“路由选择”，在不产生歧义的情况下，文中有时也直接译为“路由”。——译者注

我们仅讨论单播路由；多播路由以及广播路由将在本章后面讨论。

在拥有大量路由器和主机的因特网中，仅使用分层路由就可以完成单播路由：在不同步骤使用不同的路由选择算法。在本节，我们首先讨论互联网中的单播路由的一般概念。这个互联网是：由路由器连接的多个网络所构成的互联网络。在理解路由概念和算法之后，我们将给出如何通过分层路由而将其应用到因特网。

4.3.1 一般思想

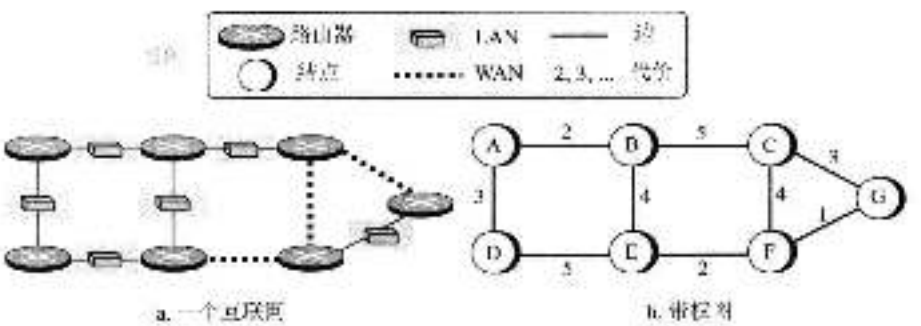
在单播路由中，在转发表的帮助下，分组被一跳一跳地从源端路由到目的端。源端主机不需要转发表，因为它将分组传送到本地网络中的默认路由器。目的主机也不需要转发表。因为它从本地网络的默认路由器中接收分组。这意味着，只有将互联网中网络连接到一起的路由器才需要转发表。有了以上的解释，从源端到目的端路由分组就表示将分组从源路由器（源主机的默认路由器）路由到目的路由器（目的主机的默认路由器）。尽管分组需要访问源和目的路由器，但是关键在于分组需要访问哪些其余的路由器。换言之，分组从源端到目的端有很多条路由；分组必须决定采取哪条路由。

以图的形式表示因特网

为了找到最好的路径，互联网可以用图（graph）来表示。计算机科学中的图是结点（node）和边（edge）（线）的集合。为了使用图表示互联网，我们可以将每个路由器想象成一个结点，将一对路由器之间的网络想象成一条边。实际上，互联网以带权图（weighted graph）表示，其中每条边和一个代价相关。如果带权图用来表示地理区域，结点可以是城市，边可以是连接城市的道路；这种情况下的权重就是城市之间的距离。然而，在路由中，边的代价在不同路由协议中有不同的解释。我们暂且假设代价与对应的边相关。如果结点间没有边，那么代价是无限大的。图 4-56 给出了如何以图模型表示互联网。

最小代价路由

当使用带权图模型表示互联网时，从源路由器到目的路由器的最佳路由就是找到两者之间的最小代价。换言之，源路由器在所有可能的路由中找到代价最小的那一条作为到目的路由器的路由。在图 4-56 中，A 和 E 之间的最佳路由是 A-B-E，代价是 6。这意味着，每个路由器需要找到自身到其余所有路由器之间的最小代价路由，这样才能使用这些条件来路由分组。



a. 一个互联网　　b. 带权图

图 4-56 互联网及其图示

最小代价树

如果互联网中有 N 个路由器，那么从每个路由器到其余任意一个路由器存在$(N-1)$条最小代价路径。这意味着，整个互联网需要有 $N \times (N-1)$条路径。如果互联网中只有 10 个路由器，我们需要 90 条最小代价路由。一种较好的查看所有这些路径的方法就是用最小代价树（least-cost tree）来把它们组合起来。最小代价树以源路由器为根，跨越了整个图（访问了所有其余结点），并且其中

根和其他结点的路径是最短的。按这种方法，每个结点只有一个最短路径树；整个因特网有 N 个最短路径树。本节稍后我们给出如何创建每个结点的最小代价树；现在，图 4-57 给出在图 4-56 中的互联网的 7 个最小代价树。

如果最短路的最小代价树是依照一致的标准创建的，那么它们有以下几个性质：

1. X 树中从 X 到 Y 的最小代价路由与 Y 树中从 Y 到 X 的最小代价路由方向相反；但两个方向的代价是相同的。例如，在图 4-57 中，A 树中从 A 到 F 的路由是（A → B → E → F），但是 F 树中从 F 到 A 的路由是（F → E → B → A）。这与前一条路由相同。但每种情况的代价是都是 8。

2. 从 X 到 Z 除了使用 X 树之外，我们也可以在 X 树中从 X 路由到 Y，然后在 Y 树中从 Y 路由到 Z。例如，在图 4-57 中，从 A 到 G 我们在 A 树中使用路由（A → B → E → F → G）。我们也可以在 A 树中从 A 到 E（A → B → E），并在 E 树中继续路由（E → F → G）。第二种情况中两个路由的结合与第一种情况相同。第一种情况的代价是 9，第二种的代价也是 9（6+3）。

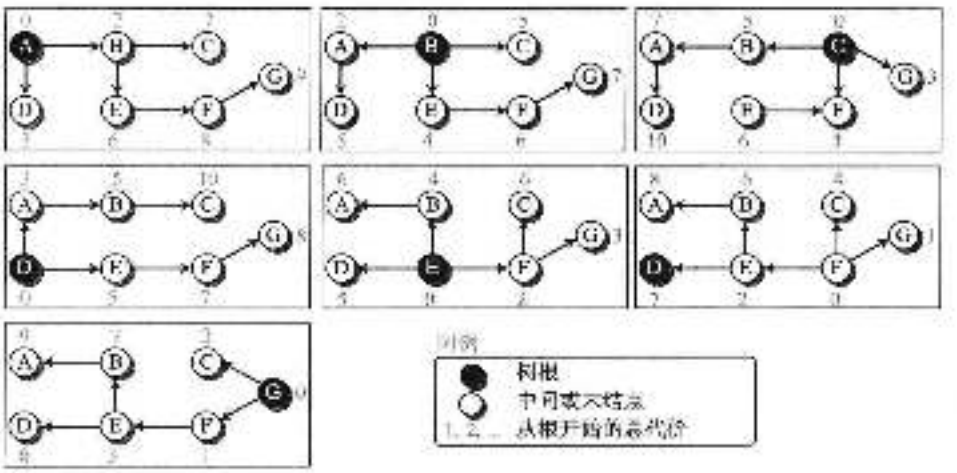


图 4-57 图 4-56 中互联网中结点的最小代价树

4.3.2 路由选择算法

我们讨论了最小代价树的一般思想以及利用这些树创建的转发表。我们现在专注于路由选择算法。过去设计了好几种路由选择算法。这些算法解释最小代价的方法不同，它们为每个结点创建最小代价树的方法也不同。在本节，我们讨论普通算法；之后我们给出因特网中的路由协议如何实现一种这样的算法。

距离向量路由选择

距离向量路由选择（distance-vector（DV）routing）使用我们在介绍中讨论过的那种方法来找到最佳路由，在距离向量路由选择中，每个结点的首要事情是利用自身所拥有的关于临站（immediate neighbor）的初步信息创建最小代价树。这些不完全的树在临站之间交换，使得树越来越完整并且代表整个互联网。可以说，在距离向量路由选择中，路由器不断地告知其临站它所知的关于整个互联网的信息（尽管可能不全面）。

在给出不完全最小代价树如何组合成为完全的最小代价树之前，我们需要讨论两个重要的问题：Bellman-Ford 方程（Bellman-Ford equation）以及距离向量的概念，我们接下来就予以介绍。

Bellman-Ford 方程

距离向量路由选择的核心就是著名的 **Bellman-Ford** 方程。当源结点和中间结点之间的代价以及中间结点和目的结点的代价被给出时，这个方程用来计算源结点 x 到目的结点 y 的最小代价（最

短距离），这条路径穿过了中间结点（a，b，c，……）。以下公式描述了一般情况，D_{xy}是最短距离，c_{xi}是结点x和i之间的代价。

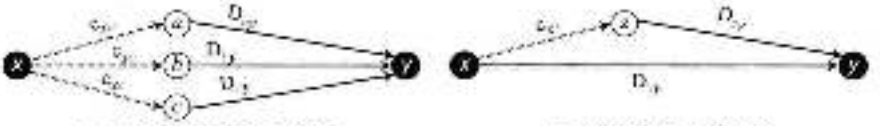
$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \cdots\}$$

在距离向量路由选择中，如果经过一个中间结点（例如z）的最小代价更小的话，我们通常想用它来更新已存在的最小代价。在这种情况下，等式变得更简单了，如下：

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

图 4-58 用图示给出两种情况中包含的思想。

可以说 Bellman-Ford 方程使我们能够从之前建立的最小代价路径中建立一条新的最小代价路径。在图 4-58 中，我们可以把（a—y）、（b—y）和（c—y）看做之前建立的最小代价路径，把（x—y）看做新建立的最小代价路径。如果我们重复使用这个方程，我们甚至可以把这个方程看做从之前建立的最小代价树中建立新最小代价树的构造工具。换言之，在距离向量路由选择中使用这个方程就说明这种方法也使用最小代价树，但是这可能是隐含在背后使用的。



a. 一个中间结点的一般情况　　b. 重新输出更新路径

图 4-58　Bellman-Ford 方程内涵思想的刻示

我们很快将介绍如何使用 Bellman-Ford 方程并介绍距离向量的概念，从而来构建距离向量路由选择中每个结点的最小代价路径。但是首先我们需要讨论距离向量的概念。

距离向量

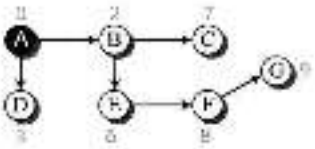
距离向量（distance vector）的概念是距离向量路由选择的命名依据。最小代价路由树是从根结点到所有目的结点的最小代价路径的组合。这些路径在图上连接在一起形成树。距离向量路由选择拆开这些路径并形成距离向量，即这是代表树的一维数组。图 4-59 给出图 4-56 中互联网内结点 A 的离和响应的距离向量。

注意，距离向量的名称（name）定义了根结点，索引（index）定义了目的结点，每个单元格的值（value）定义了从根到目的结点的最小代价。距离向量并不像最小代价树一样给出到目的结点的路径；它仅仅给出到目的结点的最小代价。之后，我们会展示如何将距离向量转换成转发表，但是我们首先需要找到互联网中的所有距离向量。

我们知道距离向量可以代表最小代价树中的最小代价路径，但是问题在于互联网中的每个结点最初如何创建相应的向量。当互联网中的结点启动时，它创建一个非常基本的距离向量，这个向量包含了可从临站获得的最少信息。结点从接口发送一些问候报文并发现临站的身份以及接口到临站的距离。之后，它把获得的距离插入相应的单元格，从而建立一个简单的距离向量，并且将其他单元格的值设为无限。这些距离向量能代表最小代价路径吗？是的，考虑到结点所拥有的有限信息，它们代表最小代价路径。当我们仅仅知道两个结点之间的一种距离时，它就是最小代价。图 4-60 给出互联网中所有的距离。然而，我们需要提及的是，所有向量都是当相应结点启动时异步生成的；图中所有向量都存在并不意味着它们是同步生成的。

这些基本的向量不能帮助互联网有效地转发分组。例如，结点 A 认为它没有连接到结点 G，因为相应单元格给出最小代价为无限大。为了改进这些向量，互联网的结点需要通过交换信息相互帮助。在每个结点创建它的向量之后，它向所有临站发送向量的副本。在一个结点从临站接收到一个距离向量后，这个结点使用 Bellman-Ford 方程（第二种情况）来更新距离向量。然而，需要明白的是，我们需要更新的不只是一条最小代价而是 N 条，这其中的 N 是互联网中的结点数目。如

果我们正在使用一个程序，我们可以使用循环去做；如果我们给出纸面上的概念，我们可以给出整个向量而不是给出 N 个独立的方程。在图 4-61 中我们给出每次更新中的完整向量而不是 7 个方程。这幅图展示了两个异步事件，一个事件在另一个之后发生，中间间隔了一段时间。在第一个事件中，结点 A 向结点 B 发送一个向量。结点 B 使用代价 $c_{BA}=2$ 更新向量。在第二个事件中，结点 E 发送向量到结点 B。结点 B 使用代价 $c_{EA}=4$ 更新向量。



a. 结点A的树



b. 结点A的距离向量

图 4-59 与树相关的距离向量

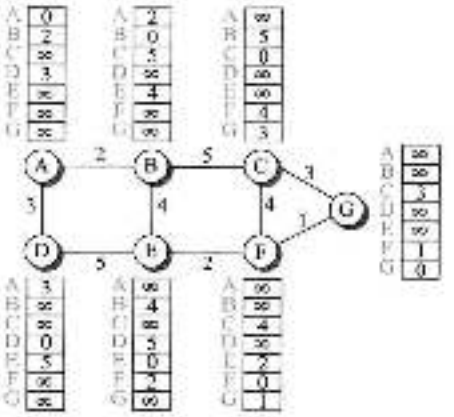
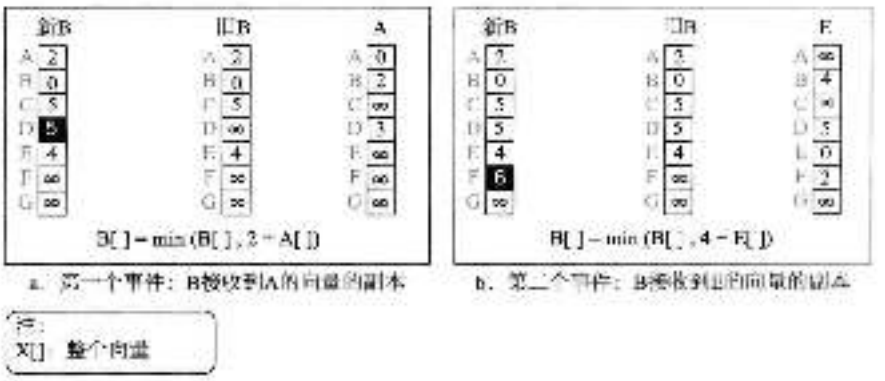


图 4-60 长时间中第一个距离向量



a. 第一个事件：B接收到A的向量的副本

b. 第二个事件：B接收到E的向量的副本

注：
X[] 整个向量

图 4-61 更新距离向量

在第一个事件之后，结点 B 在它的向量中有改进：它到结点 D 的最小代价从无限变为了 5（通过结点 A）。在第二个事件之后，结点 B 在它的向量中有更大改进：它到结点 F 的最小代价从无限变为了 6（通过结点 E）。我们希望已经使读者确信交换向量最终会稳定系统并且允许所有结点找到自身与任意其他结点的最小代价。我们需要记住的是，在更新结点之后，它立即向所有临站发送更新过的向量。即使它的临站已经接收到了前一个向量，更新过的向量仍然可能较为有用。

距离向量路由选择算法

现在我们给出距离向量路由选择算法的简化伪代码。如表 4-4 所示。算法被结点单独异步地运行。

第 4 行到第 11 行初始化结点的向量。第 14 行到第 23 行给出从临站接收到向量后结点如何被更新。第 17 行到第 20 行的 for 循环允许向量中的所有表格项（单元格）在接收到一个新的向量后进行更新。注意，在第 12 行，结点在初始化后发送它的向量；在第 22 行，结点在更新后发送它的向量。

表 4-4 结点 A 的距离向量路由选择算法

```
Distance_Vector_Routing ( )
{
    // 初始化（创建结点的初始向量）
    D[myself] = 0
    for (y = 1 to N)
    {
        if (y is a neighbor)
            D[y] = c[myself][y]
        else
            D[y] = ∞
    }
    send vector {D[1], D[2], ..., D[N]} to all neighbors
    // 更新（根据从邻居接收来的可能发生改变向量）
    repeat (forever)
    {
        wait (for a vector D_w from a neighbor w or any change in the link)
        for (y = 1 to N)
        {
            D[y] = min [D[y], [c[myself][w] + D_w[y]]] // Bellman-Ford 方程
        }
        if (any change in the vector)
            send vector {D[1], D[2], ..., D[N]} to all neighbors
    }
} // 算法可能结束
```

计数到无穷

距离向量路由的问题是任何代价的减少（好消息）传播得快，代价的增加（坏消息）传播得慢。为了使路由协议正常工作，如果链路损坏（代价变为无穷），其余每个路由器应该立即意识到这件事，但是在距离向量路由选择中，这要花一些时间。这个问题称为计数到无穷（count to infinity）。有时在损坏链路被所有路由器记录为无穷之前，需要好几个更新。

两结点循环

计数到无穷的一个例子就是两结点循环。为了理解这个问题，让我们来看一看图 4-62 中描述的场景。

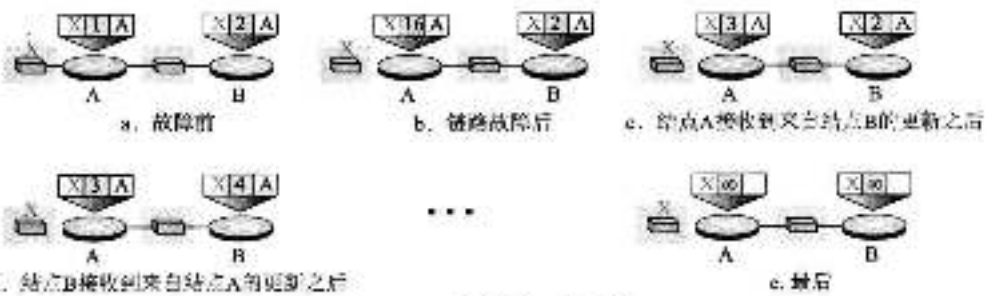


图 4-62 两结点的不稳定性

图 4-62 给出了带有三个结点的系统。我们只给出讨论所需的一部分转发表。开始时，两个结

点 A 和 B 都知道如何到达结点 X。但是突然 A 与 X 断开，结点 A 改变它的转发表。如果 A 立即将其转发表发送到结点 B，一切正常。然而，在 B 接收到 A 的转发表之前，B 已经发送了它的转发表，该系统变得不稳定。结点 A 接收到来自 B 的转发表，它假定 B 已经找到了到达 X 的路径，立即更新它的路由表。现在 A 向 B 发送它刚刚更新的转发表。现在 B 认为 A 的周围环境已经改变，于是 B 更新它自身的转发表。到达 X 的代价逐渐地增加直到无穷大。这时，A 和 B 两者都知道 X 不可达。但是，在这个过程中，系统是不稳定的。结点 A 认为通过 B 有路径到达 X，而结点 B 认为通过 A 有路径到达 X。如果 A 接收到目的为 X 的分组，这个分组发向 B，然后回到 A。同样地，如果 B 接收到目的地为 X 的分组，这个分组发向 A，然后回到 B。这样分组在 A 和 B 之间来回往返产生两结点循环问题。对于这类不稳定性已经提出了若干个解决办法。

水平分割 一种解决方法称为水平分割（split horizon）。在这种策略中，不通过每一个接口发送整个表，而是每一个结点通过每一个接口仅发送它的表的一部分。根据转发表，如果结点 B 认为通过 A 到达 X 是最佳路径，则结点 B 不需要向 A 通知这一部分信息，因为这一信息来自 A（A 已知道），而把来自结点 A 的信息修改后，再发送回结点 A 可能会产生混淆。在前面图 4-62 的场景中，结点 B 在向结点 A 发送它的表之前，删除转发表的最后一行，结点 A 仍然保持到 X 的距离为无穷大。稍后，当结点 A 发送它的转发表到结点 B 时，结点 B 也修改它的路由表。在第一次更新后，系统成为稳定的，两个结点 A 和 B 都知道 X 是不可达的。

毒性逆转 水平分割策略有一个缺点，即距离向量协议经常使用定时器。如果长时间没有关于一条路径的新信息，那么结点就从它的表中删除该路径。在前面的场景中，当结点 B 从发向 A 的通知中删除到 X 的路径时，结点 A 不能推断这是因为水平分割策略（信息源是 A），还是因为结点 B 最近没有接收到关于 X 信息。水平分割策略可与毒性逆转（poison reverse）策略结合起来。结点 B 依旧可以通知到 X 的值，但如果信息源是结点 A，那么它用无穷大表示距离作为警告："不要使用这个值，我知道这个路径来自你。"

三结点不稳定性

两结点不稳定性可以通过使用水平分割策略与毒性逆转策略结合起来避免。但是，如果不稳定性存在于三个结点之间，则稳定性不能得到保证。

链路状态路由选择

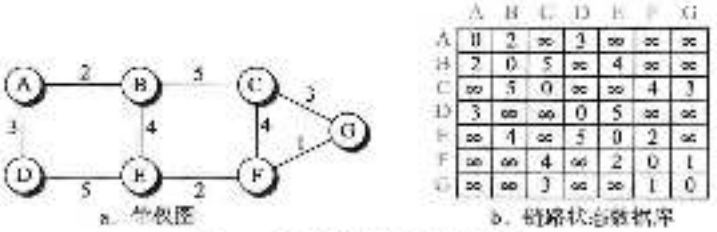
符合我们对创建最小代价树和转发表讨论的一种路由协议是链路状态路由选择（link-state（LS）routing）。这个方法使用术语链路状态（link-state）来定义链路（一条边）的特征，链路代表了互联网中的网络。在这种算法中，与边相关的代价定义了链路的状态。代价低的链路比代价高的链路更好；如果链路的代价无限大，这意味着链路不存在或已经被损坏。

链路状态数据库

为了使用这个方法创建最小代价树，每个结点需要网络的完全图，这意味着它需要知道每条链路的状态。所有链路状态的集合称为链路状态数据库（link-state database（LSDB））。整个互联网只有一个 LSDB；每个结点需要一个副本才能创建最小代价树。图 4-63 给出一个 LSDB 的例子，它表述的是图 4-56 中的互联网图。LSDB 可以使用二维数组（矩阵）表示，其中每个单元格的值定义了相应链路的代价。

现在，问题在于每个结点如何创建这个 LSDB，它包含了整个互联网的信息。这可以通过称为泛洪（flooding）的过程完成。每个结点可以给所有临站（与其直接相连的结点）发送问候报文，来收集每个临站的两条信息：结点的标识以及链路的代价。这两个信息的组合称为 LS 分组（LS packet（LSP））；如图 4-64 所示 LSP 从每个接口中发出，这幅图表述的是图 4-56 中的互联网。当结点从一个接口接收到 LSP 时，它将 LSP 与已经拥有的副本进行比较。如果新到达的 LSP 比现有

的副本旧（通过检查序号判断），它就丢弃 LSP。如果较新或者这是接收到的第一个 LSP，那么结点丢弃旧的 LSP（如果存在的话）并保存接收到的 LSP。之后结点从每个端口中发送一份副本，不包含分组到达的端口。这保证了泛洪会在网络中某处停止（结点只有一个接口）。我们需要使自己确信，在接收到所有新的 LSP 之后每个结点都创建了如图 4-64 所示的 LSDB。这个 LSDB 对于每个结点都是相同的，它也给出了互联网的整个图。换言之，如果需要的话结点可以使用这个 LSDB 创建整个图。



a. 节点图

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. 链路状态数据库

图 4-63 链路状态数据库的例子

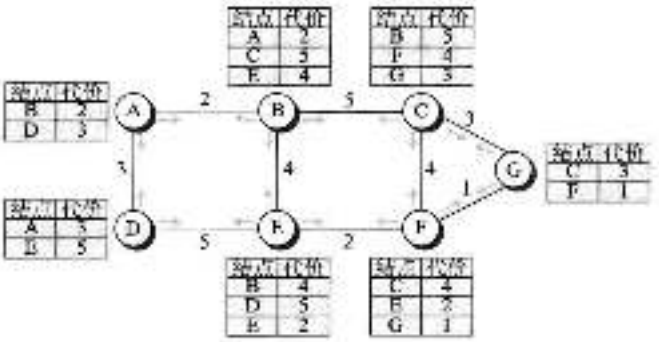


图 4-64 每个结点创建并发送 LSP 来创建 LSDB

我们可以将链路状态路由选择算法与距离向量路由选择算法进行比较。在距离向量路由选择算法中，每个路由器向它的邻居告知它所知道的关于整个互联网的信息；在链路状态路由选择算法中，每个路由器向整个互联网告知它所知道的关于邻居的信息。

最小代价树的形成

为了创建最小代价树，使用共享 LSDB 的每个结点需要运行著名的 **Dijkstra 算法**（Dijkstra Algorithm）。这个迭代算法使用如下步骤：

1. 结点将自身作为树根，创建只有一个结点的树，并根据 LSDB 中的信息设置每个结点的总代价。

2. 结点从所有不在树中且与根最近的结点中选择一个，加入树。在这之后所有不在树中的结点的代价需要更新，因为路径可能有变化。

3. 结点重复第二步，直到所有结点加入树。

我们需要确信的就是，在以上三步完成之后，最终创建了最小代价树。表 4-5 给出一个 Dijkstra 算法的简化版本。

第 4 行到第 13 行执行算法中的第一步。第 16 行到第 23 行执行了算法中的第 2 步。第 2 步被重复直到所有结点加入树中。

图 4-65 使用 Dijkstra 算法给出图 4-63 中互联网图的最小代价树信息。我们需要经过初始化步骤和六次迭代来得到最小代价树。

表 4-5 Dijkstra 算法

```
Dijkstra's Algorithm ( )
{
    // 初始化
    Tree = {root}                    // 树仅由根节点组成
    for (y = 1 to N)                 // N 是结点的个数
    {
        if (y is the root)
            D[y] = 0                 // D[y]是从根到结点 y 的最短距离
        else if (y is a neighbor)
            D[y] = c[root][y]        // c[x][y]是 LSDB 中结点 x 和 y 之间的代价
        else
            D[y] = ∞
    }
    // 计算
    repeat
    {
        find a node w, with D[w] minimum among all nodes not in the Tree
        Tree = Tree ∪ {w}            // 将 w 加入到树中
        // 更新 w 所有邻居的距离
        for (every node x, which is neighbor of w and not in the Tree)
        {
            D[x] = min{D[x], (D[w] + c[w][x])}
        }
    } until (all nodes included in the Tree)
} // Dijkstra 结束
```

路径向量路由选择

链路状态和距离向量路由选择都是基于最小代价目标的。然而，在一些例子中这个目标不是优先考虑的事情。例如，假设互联网中有一些路由器，一个发送端不想通过这些路由器发送它的分组。比如说，属于某个组织的路由器可能不够安全，或者属于发送者的某个商业对手的路由器可能检查分组来获取信息。最小代价路由并不阻止分组经过某个最小代价路径中的区域。换言之，LS 或 DV 路由选择所使用的最小代价目标不允许发送端在路由分组时采用特定的策略。除了安全和保密，如后面所讨论的，有时路由的唯一目的就是到达：允许分组更高效地到达目的端，而不给路由指派代价。

为了满足这些要求，称为路径向量路由选择（path-vector（PV）routing）的第三种路由选择算法被设计出来。路径向量路由选择没有上文讨论的 LS 或 DV 路由选择的缺点，因为它不基于最小代价路由。最佳路由由源端使用的策略决定。换言之，源端可以控制路径。尽管实际上，路径向量路由选择没有在互联网上应用，而是主要为 ISP 之间路由分组而设计，但是我们在本节还是要讨论这个方法的主要原则，就像它用在了互联网上一样。在下一节，我们给出如何将其应用到因特网。

生成树

在路径向量路由选择中，从源端到目的端的路径由又由最佳生成树决定。然而，最佳生成树不是最小代价树；当执行自己的策略时，最佳生成树是由源端决定的树。如果到目的端有不止一条路由，那么路由器可以选择与这个策略最匹配的路由。源端可能同时应用几种策略。常见的一

个策略是使得被访问的结点数量最小（有点类似最小代价）。另一个常见策略是避免路由中的某些中间结点。

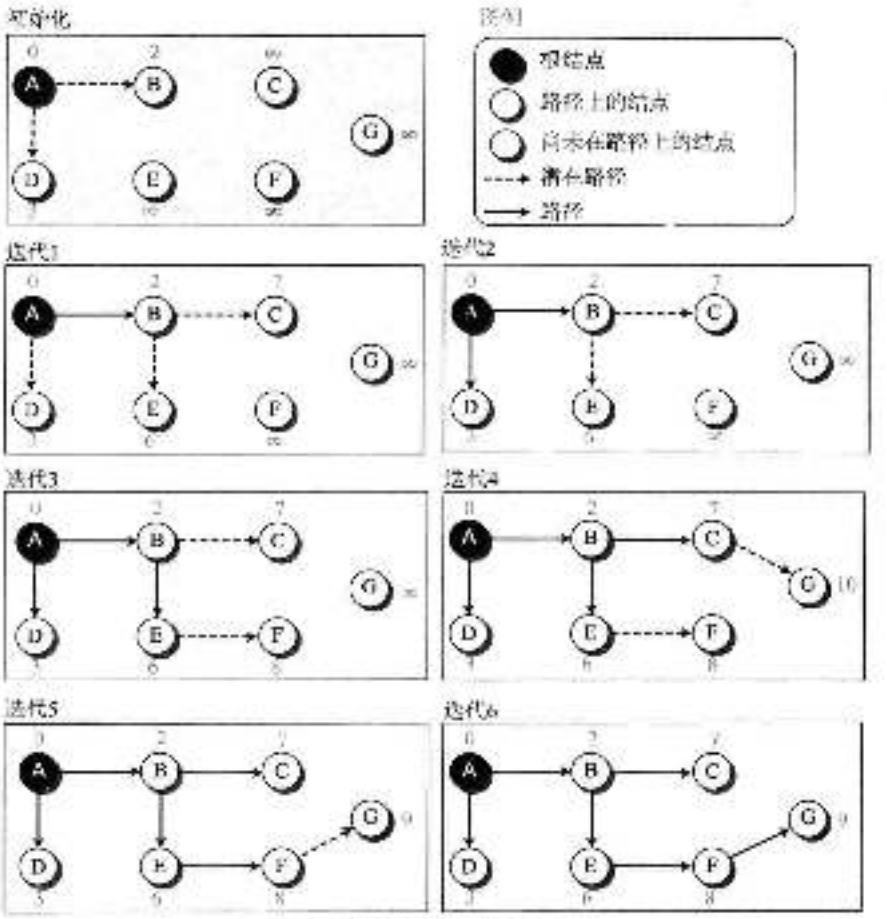


图 4-65 最小代价树

图 4-66 给出一个只有五个结点的小型互联网。每个源端创建满足自身策略的生成树。所有源端使用的策略是使用最少的结点到达目的端。由 A 和 E 选择的生成树就是符合这个策略的，它将 D 作为中间结点。类似地，由 B 选择的生成树不将 C 作为中间结点。

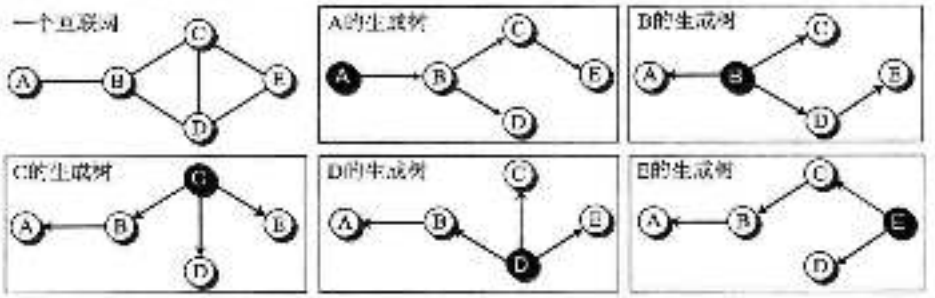


图 4-66 路径向量路由选择中的生成树

生成树的创建

路径向量路由选择与距离向量路由选择类似，是一种异步、分布式路由选择算法。生成树是每

个结点逐步地、异步地创建起来的。当结点启动时，它基于自身得到的关于临站的信息来创建一个路径向量。结点向它的临站发送问候报文来收集信息。图 4-67 给出了如图 4-66 所示的互联网的所有路径向量。然而，我们并不是说所有这些表格都是同时创建的；它们是当各个结点启动时分别被创建的。这幅图也给出这些路径向量在创建后是如何被发送到临站的（箭头）。

在创建初始路径向量之后，每个结点将其发送给所有临站。当从邻居接收到路径向量时，每个结点都使用与 Bellman-Ford 类似的方程式来更新路径向量，但是结点应用自己的策略而不是寻找最小代价。我们可以将这个方程式定义为

$$Path(x, y) = best\ \{Path(x, y), [x + Path(v, y)]\} \qquad 对于互联网中所有\ v。$$

在这个方程式中运算符（+）意味着将 x 加到路径的开始。我们也将要小心避免将结点加到空路径上，因为空路径意味着路径不存在。

这个策略由选择最佳的多重路径所定义。路径向量路由选择也在方程式中加入了一个条件：如果路径（v，y）包含 x，那么路径被丢弃以防路径中出现循环。换言之，当 x 选择到 y 的路径时，它不想访问到自身。

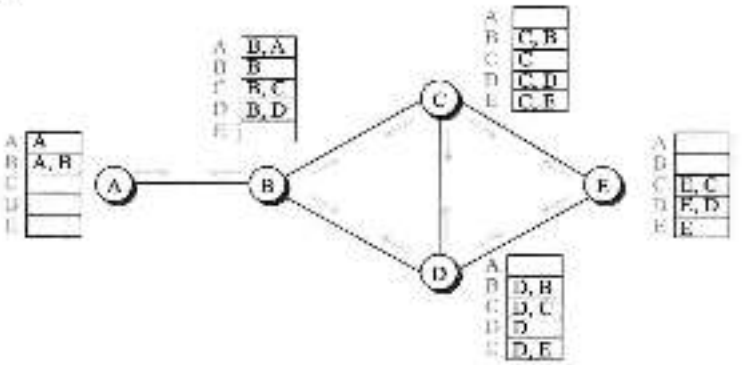


图 4-67 启动时刻创建的路径向量

图 4-68 给出了在两个事件之后结点 C 的路径向量。在第一个事件中，结点 C 接收到 B 的向量的副本，这改善了结点 C 的向量：现在结点 C 知道如何到达结点 A 了。在第二个事件中，结点 C 接收到 D 的向量的副本，这没有改变结点 C 的向量。事实上，在第一个事件之后，结点 C 的向量就稳定下来并作为转发表。

路径向量算法

基于初始化过程以及从邻居接收到路径向量后更新每个转发表所用到的方程式，我们可以写出路径向量算法的一个简化版本，如表 4-6 所示。

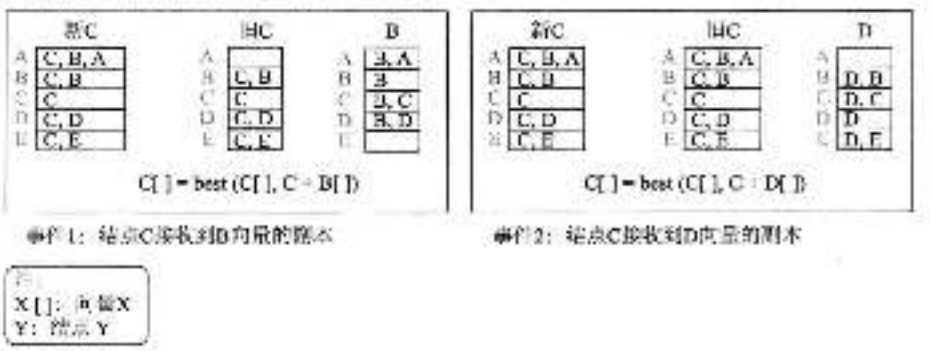


图 4-68 更新路径向量

表 4-6 结点的路径向量算法

```
1   Path_Vector_Routing ( )
2   {
3       // 为初化
4       for (y = 1 to N)
5       {
6           if (y is myself)
7               Path[y] = myself
8           else if (y is a neighbor)
9               Path[y] = myself + neighbor node
10          else
11              Path[y] = empty
12      }
13      Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
14      // 更新
15      repeat (forever)
16      {
17          wait (for a vector Path_w from a neighbor w)
18          for (y = 1 to N)
19          {
20              if (Path_w includes myself)
21                  discard the path                    // 避免环路
22              else
23                  Path[y] = best {Path[y], (myself + Path_w[y])}
24          }
25          If (there is a change in the vector)
26              Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
27      }
28  } // 路径向量结束
```

第 4 行到第 12 行给出结点的初始化。第 17 行到第 24 行给出结点从邻居接收到向量后，它如何更新自己的向量。更新过程永远重复，我们可以看出这个算法和 DV 算法之间的相似之处。

4.3.3 单播路由选择协议

在前一节，我们讨论单播路由选择算法；在本节，我们讨论因特网中使用的单播路由选择协议。尽管我们此处讨论的三个协议基于我们之前讨论的三个相应算法，但是协议比算法丰富得多。协议需要定义操作的域、交换报文、路由器之间的通信以及与其他域内协议的相互作用。在介绍之后，我们讨论因特网中三个常用协议：基于距离向量算法的路由选择信息协议（Routing Information Protocol，RIP）、基于链路状态算法的开放最短路径优先（Open Shortest Path First，OSPF）以及基于路径向量算法的边界网关协议（Border Gateway Protocol，BGP）。

互联网结构

在讨论单播路由选择协议之前，我们需要理解现今因特网的结构。因特网已经从只有一个骨干网的树状结构转变成了由不同私人公司运营的多骨干结构。尽管很难给出当今因特网的全貌，但是

我们可以说因特网的结构与图 4-69 类似。

有很多私人通信公司运营的骨干网，它们提供全球连接。骨干网通过一些对等点连接在一起。在较低层次有一些供应商网络使用骨干网来进行全球连接，但是它们向因特网用户提供服务。最终是一些使用提供商网络所提供服务的用户网络。这三种实体（骨干网、供应商网络或用户网络）都可以称为因特网服务提供商（Internet Service Provider）或 ISP。它们提供服务，但是处于不同的层次。

分层路由选择

当今的因特网由大量网络和起连接作用的路由器组成。很明显，因特网中的路由选择不能仅使用一种协议，理由有两点：扩展性问题以及管理问题。扩展性问题（scalability problem）意味着转发表会变得巨大，在其中寻找目的端会相当耗时，对其更新会产生巨大的网络流量。管理问题（administrative issue）与图 4-69 中描述的因特网结构有关。如图所示，每个 ISP 都由一个管理机构运营。管理者需要在系统中进行控制。组织机构必须能够使用它所需要的了网和路由器，它可能需要来自特定生产商的路由器，也可能希望运行特定的路由算法来满足组织机构的需要，也可能想要在通过 ISP 的通信流上实行某些策略。

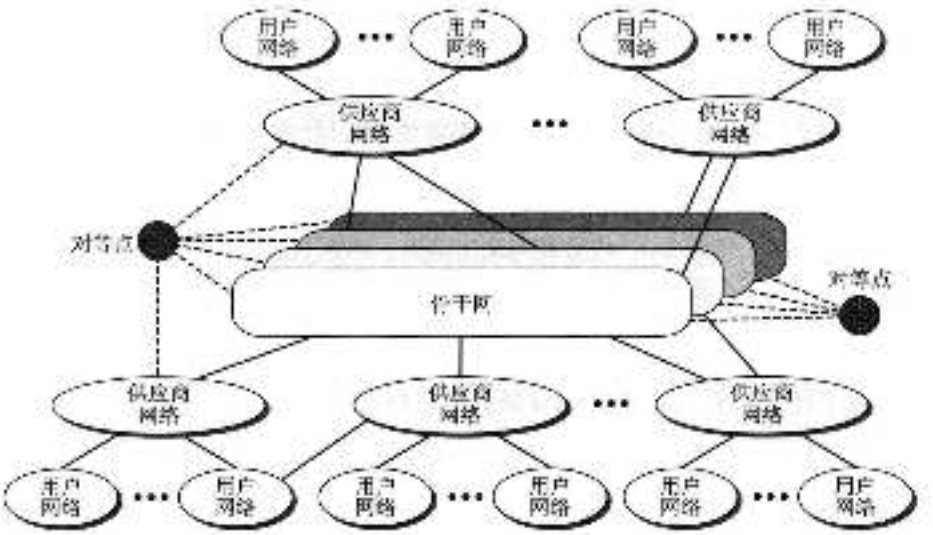


图 4-69 因特网结构

分层路由意味着将每个 ISP 看做一个自治系统（autonomous system，AS）。每个 AS 可以运行满足需要的路由协议，但是全局因特网只运行一个全局协议来将所有 AS 连接到一起。每个 AS 中运行的路由协议称为 AS 内部路由选择协议（intra-AS routing protocol）、域内路由协议或内部网关协议（interior gateway protocol，IGP）；全局路由选择协议称为 AS 间路由选择协议（inter-AS routing protocol）、域间路由选择协议（interdomain routing protocol）或外部网关协议（exterior gateway protocol，EGP）。我们可以有多个域内路由选择协议，并且每个 AS 可以自由选择，但是应该清楚的是我们应该只有一种域间协议来处理这些实体之间的路由选择。现在，两种常见域间路由选择协议是 RIP 和 OSPF；一种域间路由选择协议是 BGP。当我们转到 IPv6 时情况会发生转变。

自治系统

正如我们之前所述，当 ISP 管理多个网络和多个路由器时，每个 ISP 就是一个自治系统。尽管我们可能拥有小型、中型的以及大型 AS，每个 AS 都由 ICANN 分配一个自治号（ASN）。每个 ASN 都是一个 16 位长的无符号整数，它唯一地定义了一个 AS。然而，自治系统不是根据自身大小来分类的；而是根据它们与其他 AS 的连接方式来分类的。我们有残桩 AS、多宿主 AS 以及过渡 AS。

相互会看到，这些类型会影响到与 AS 相关的域间路由协议的运行。

- 残桩 AS（Stub AS）。残桩 AS 与其他 AS 只有一个连接。数据通信可以在残桩 AS 内初始化或结束；数据不能从中穿过。一个很好的残桩 AS 例子就是客户网络，它是数据的源端或接收端。
- 多宿主 AS（Multihomed AS）。一个多宿主 AS 可以与其他 AS 有一个以上的连接，但是不允许数据通信穿过自身。一个很好的多宿主 AS 例子就是客户 AS，它们可能使用多个提供商网络，但是它们的策略不允许数据穿过自身。
- 过渡 AS（Transient AS）。过渡 AS 与一个以上其他 AS 进行连接，而且也允许数据通信流经自身。提供商网络以及骨干网就是一个例子。

路由选择信息协议（RIP）

路由选择信息协议（Routing Information Protocol，RIP）是一个在自治系统内部使用的域内路由选择协议，它基于我们之前描述的距离向量路由选择算法。一开始，RIP 是作为施乐公司网络系统（Xerox Network System，XNS）的一部分，但它是 UNIX 的加州大学伯克利分校软件（Berkly Software Distribution，BSD），正是这一点使得 RIP 被广泛使用。

跳数（hop count）

这个协议中的路由器从根本上讲实现的是如表 4-4 所示的距离向量路由选择算法。然而，如下所述，算法被修改了。首先，由于 AS 中的路由器需要知道如何将分组转发到 AS 中的另一个网络（子网），RIP 路由器通告到达不同网络的代价，而不通告到达理论图中结点的代价。换言之，代价被定义在路由器和目的主机所在的网络之间。第二，为了使代价的实现简化（独立于路由器和链路的性能参数，例如延迟、带宽等），代价被定义为跳数，跳数表示分组从源路由器到最终目的主机所需要穿过的网络（子网）的数量。注意，源端主机所连接的网络在这个算法中不计，因为源端主机不使用转发表；分组被传递到默认路由器。图 4-70 给出跳数的概念，这个跳数由三个路由器从源端主机通告给目的端主机。在 RIP 中路径的最大代价可能为 15，16 则意味着无限（无连接）。因此 RIP 只能用于自治系统，其中 AS 的直径不大于 15 跳。

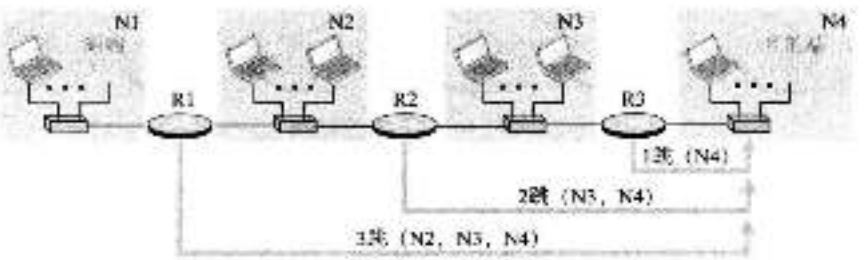


图 4-70　RIP 中的跳数

转发表

尽管我们前一节讨论的距离向量算法涉及邻居结点间交换距离向量，但是自治系统中的路由器需要记录转发表来将分组转发到它们的目的网络。RIP 中的转发表是一个三列表格，第一列是目的网络的地址，第二列是分组应该被转发到的下一跳路由器的地址，第三列是到达目的网络的代价（跳数）。图 4-71 给出图 4-70 中路由器的三个转发表。注意，第一列和第三列与距离向量表达的概念相同，但是代价表示到目的网络的跳数。

尽管 RIP 中转发表在第二列定义了下一跳路由器，但是基于之前讨论的最小代价树的第二个特性，它给出了关于整个最小代价树的信息。例如，R1 将 R2 定义为通往 N4 的下一跳路由器；R2 将 R3 定义为通往 N4 的下一跳路由器；R3 定义了这条路径上没有下一跳路由器。那么这棵树就是

R1 → R2 → R3 → N4。

R1的转发表

目的网络	下一跳路由器	下一跳代价
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

R2的转发表

目的网络	下一跳路由器	下一跳代价
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

R3的转发表

目的网络	下一跳路由器	下一跳代价
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

图 4-71 转发表

一个经常被问及的关于转发表的问题就是第三列的作用是什么。转发分组时不需要第三列，但是当路由中有变化而需要改变转发表时会用到这一列，我们稍后会看到。

RIP 实现

RIP 是使用 UDP 服务和熟知端口号 520 实现的一种进程。在 BSD 中，RIP 是一个称为 routed（路由守护进程（route daemon）的缩写）的守护进程（后台运行的一个进程）。这意味着尽管 RIP 是路由选择协议，但是它帮助 IP 对通过 AS 的分组进行路由，但是 RIP 报文封装在 UDP 用户数据报之内，之后又封装到 IP 数据报中。换言之，RIP 运行在应用层上，但是为网络中的 IP 创建转发表。

RIP 经历了两个版本：RIP-1 和 RIP-2。第二版向后兼容第一版；它允许在 RIP 报文中使用更多的信息，这些信息在第一版中都设置为 0。我们在本节只讨论 RIP-2。

RIP 报文 一个客户进程和一个服务器进程，这两个 RIP 进程像其他进程一样需要交换报文。如图 4-72 所示，RIP-2 定义了报文格式。报文中称为表项的那一部分可以按需重复。每个表项携带着与转发表里某一行相关的信息，这个转发表位于发送报文的路由器中。



图 4-72 RIP 报文格式

RIP 有两种类型报文：请求和响应。请求报文被创建它的路由器发送。请求报文可以询问特定的表项或全部表项。响应（或更新）报文可以是请求型的或非请求型。请求型的响应报文仅仅用于回答请求报文。它包含了相应请求报文中指定的目的端的信息。另一方面，非请求型的响应报文被周期性地发送，周期是 30 秒，或者当转发表有变化时便进行发送。

RIP 算法 RIP 实现的算法与我们之前讨论的距离向量路由选择算法相同。然而，为了使路由器能够更新它的转发表，需要做一些改变：

- 路由器需要在响应报文中发送转发表的全部内容而不是仅仅发送距离向量。
- 接收端在每个代价上加入一跳，并且将下一跳路由器字段改为发送路由器的地址。我们将被修改的转发表中每一条路由称为接收路由（received route），将旧的转发表中的路由称为旧路由（old route）。接收路由器将旧路由选为新路由，除了以下三种情况：

1. 如果接收路由在旧转发表中不存在，它应该被加入路由。

2. 如果接收路由的代价小于旧路由的代价，那么接收路由应该被选为一个新的路由。

3. 如果接收路由的代价大于旧路由的代价，但是下一跳路由器的值与两条路由相同，接收路由应该被选为新路由。当路由被与原先相同的那个路由器通告，但现在情况发生变化时，会发生这种情况。例如，假设原先邻居结点通告通往目的端的路由代价为3，但现在这个邻居结点到目的端之间没有路径了。邻居通告这个目的端代价值为无限大（RIP 中是 16）。接收路由器必须忽略这个值，即使它到同一个目的端的旧路由有较少的代价。

- 新转发表需要根据目的路由被存储（通常使用最长前缀优先）。

例 4.15 图 4-73 给出自治系统中一个更实际的 RIP 运行例子。首先，这幅图给出所有路由器启动后的所有转发表。之后，当更新报文交换后，我们给出表中的一些变化。最后，当没有改变时，我们给出稳定化的转发表。

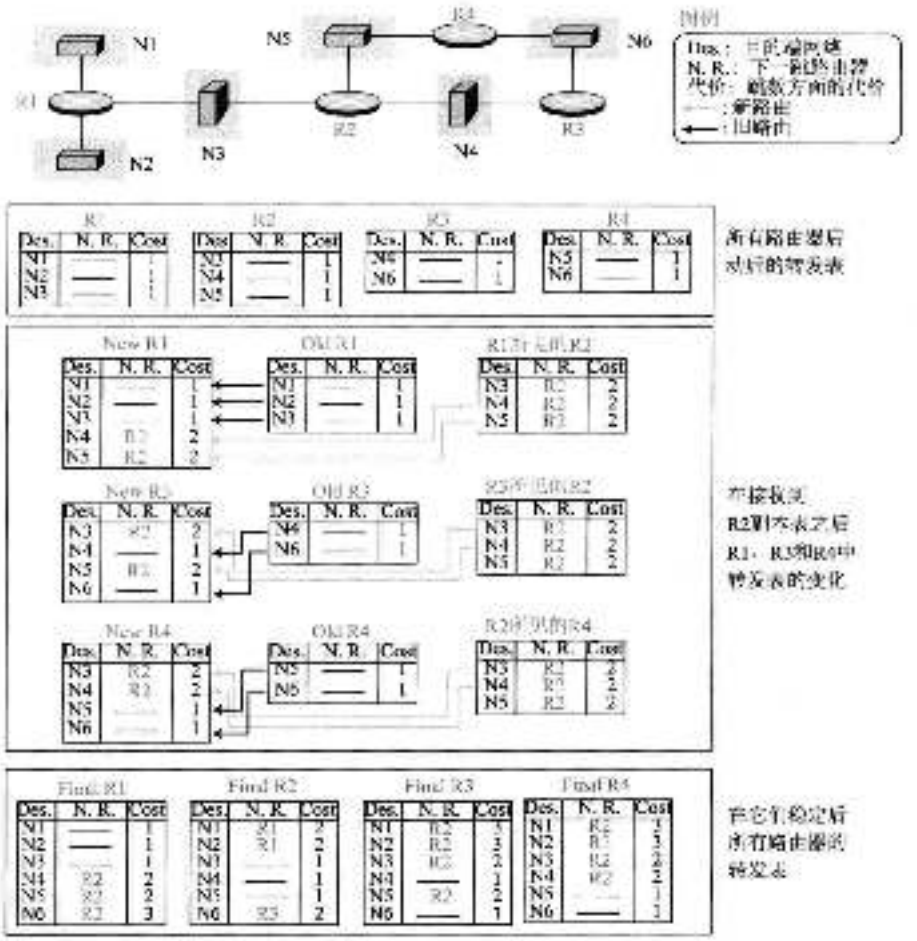


图 4-73 使用 RIP 的自治系统的例子

RIP 中的计时器 RIP 使用三个计时器来支持它的运行。周期计时器（periodic timer）控制常规更新报文的通告行为。每个路由器有一个周期计时器，它将超时时间设为 25 到 35 秒之间的随机值（为了防止所有路由器同时发送报文并造成过量通信）。计时器倒计时；当倒数到 0 时，更新报文被发送且再次随机设置计时器。超时计时器（expiration timer）控制路由的有效性。当路由器接收到路由的更新信息时，那条路由的超时计时器就被设置为 180 秒。每当那条路由的新的更新到达

时，计时器就被重置。如果互联网出现问题并且在分配的 180 秒内没有收到更新，那么路由被认为超时，并且路由的跳数被设置为 16，这意味着目的端不可达。每个路由都有自己的超时计时器。垃圾收集计时器（garbage collection timer）用来从转发表中清除路由。当关于路由的信息无效时，路由器不会立即将路由从表中清除。它继续将路由代价通告为 16。同时，那条路由的垃圾收集计时器被设置为 120 秒。当计时到 0 时，路由被从表中清除。这个计时器允许邻居在清除路由前，注意到路由的无效性。

性能

在结束这一节之前，让我们简要讨论 RIP 的性能：

- **更新报文**。RIP 中的更新报文有一个很简单的格式且仅发送给邻居；这些更新报文是局部的。它们通常并不造成过大的网络通信量，因为路由器试图避免同时发送它们。
- **转发表的收敛**。RIP 使用距离向量算法，如果域过大那么收敛就慢，但是，由于 RIP 只允许域中 15 跳（16 被认为无穷），通常收敛中不会存在问题。唯一可能减慢收敛的问题是计数到无穷以及在域内的循环；被加入到 RIP 扩展中的毒性逆转和水平分割策略可以缓解这个问题。
- **健壮性**。正如我们之前所述，距离向量路由选择基于一种概念，即每个路由器将它对每个域的了解发送给它的邻居。这意味着转发表的计算取决于从临站接收到的信息，这些邻居依次从它们的临站接收信息。如果一个路由器失效或出错，这个问题可能被传播给所有路由器并且每个路由器的转发表都会受到影响。

开放最短路径优先（OSPF）

开放最短路径优先（Open Shortest Path First，OSPF）是和 RIP 类似的域内路由选择协议，但是它基于我们本章之前讨论的链路状态。OSPF 是一个开放协议，这意味着算法说明是一个公开文档。

度量

像 RIP 一样，在 OSPF 中从主机到目的端的代价被计算为从源路由器到目的网络的代价。然而，每个链路（网络）可以被分配一个基于流量、往返时间和可靠性等方面的权值。管理者也可以决定将跳数作为代价。有趣的是，对于 OSPF 中的代价，不同服务类型（TOS）可以有不同的权值。图 4-74 给出路由器到目的主机网络的代价的概念。我们可以将图 4-70 中的数字与 RIP 进行比较。

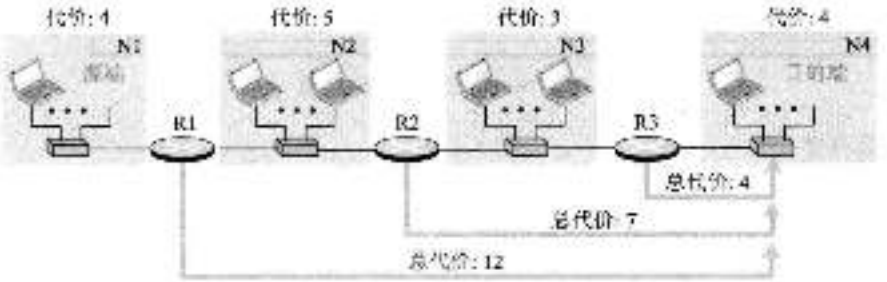


图 4-74 OSPF 中的度量

转发表

正如本章之前描述的，每个 OSPF 路由器都可以在利用 Dijsktra 算法找到自身和目的端之间的最短路径数后创建一个转发表。图 4-75 给出图 4-74 中简单 AS 的转发表。通过比较同一个 AS 中的 OSPF 和 RIP 我们可以发现唯一的不同就在于代价值。换言之，如果我们在 OSPF 中使用跳数，那么转发表将会是完全相同的。原因是这两个协议都使用最短路径树来定义从源端到目的端的最佳路由。

区域

与 RIP 相比，OSPF 用来处理小型或大型自治系统中的路由选择，而 RIP 通常在小型 AS 中使

用。然而，最短路径树的形成需要所有路由器在整个 AS 范围内泛洪链路状态分组（LSP），以此来创建全局链路状态数据库（LSDB）。尽管在小型 AS 中这样做不会产生问题，但是在大型 AS 中会产生巨大的通信量。为了防止这个问题，AS 需要分为几个称为区域（area）的部分。每个区域在泛洪 LSP 中相当于小型独立域。换言之，OSPF 使用路由选择中的另一个层次：第一层是自治系统，第二层是区域。

R1的转发表

目的网络	下一跳路由器	代价
N1	——	4
N2	——	5
N3	R2	8
N4	R3	12

R2的转发表

目的网络	下一跳路由器	代价
N1	R1	9
N2	——	5
N3	——	5
N4	R3	7

R3的转发表

目的网络	下一跳路由器	代价
N1	R2	12
N2	R2	8
N3	——	5
N4	——	4

图 4-75 OSPF 中的转发表

然而，区域中的每个路由器需要知道本区域和其他区域的链路状态的信息。因此，AS 中的一个区域被指派为主干区域（backbone area），它负责将其他区域连接到一起。主干区域的路由器负责将从每个区域收集来的信息传递到其他区域。这样，区域中的路由器就可以接收到其他区域产生的所有 LSP。为了通信，每个区域都有一个区域标识。主干网的区域标识为 0。图 4-76 给出自治系统和它的区域。

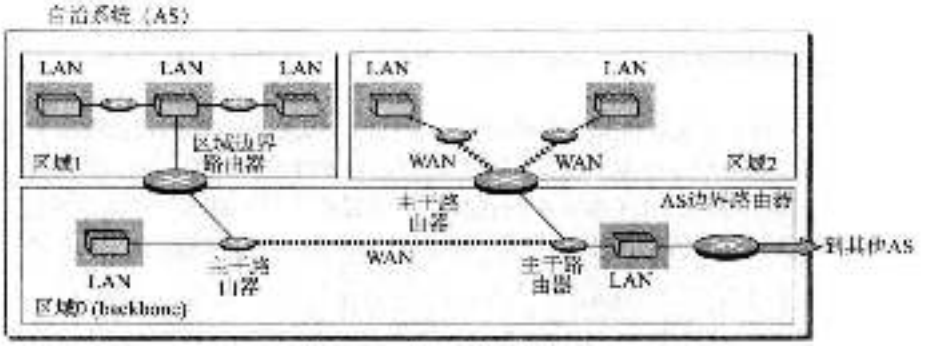


图 4-76 自治系统中的区域

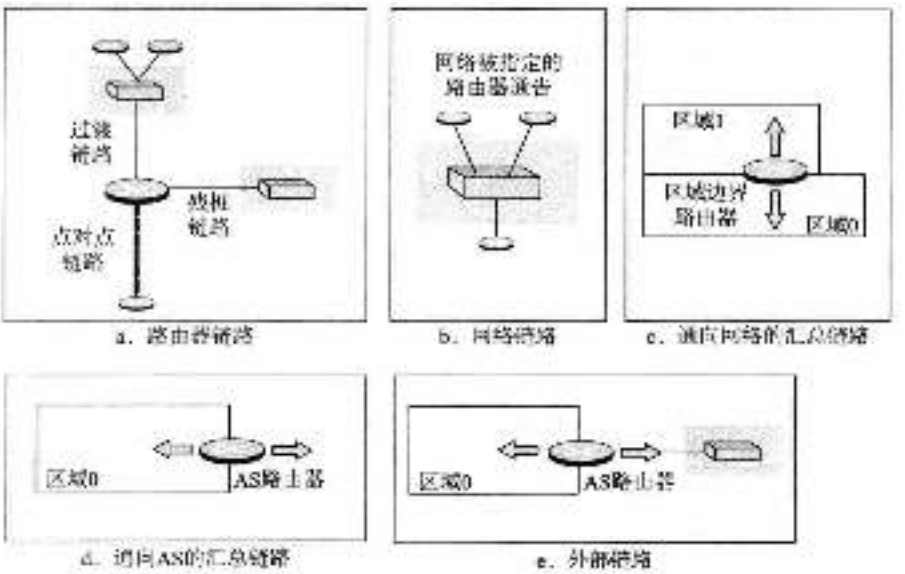
链路状态通告

OSPF 基于链路状态路由选择算法，这要求路由器向所有邻居通告其链路状态，以此来形成 LSDB。当我们讨论链路状态算法时，我们使用图论且将每个路由器假设为一个结点并将两个路由器之间的网络假设为边。在真实世界中情况有所不同，我们需要通告不同实体的存在例如结点，需要通告不同的连接类型，是它们将每个结点连接到它的邻居结点上，也需要通告与每条链路相关的代价。这意味着我们需要不同的通告类型，每种类型能够通告不同的情况。我们有五种链路状态通告类型：路由器链路（router link）、网络链路（network link）、通向网络的汇总链路（summary link to network）、通向 AS 边界路由器的汇总链路（summary link to AS border router）以及外部链路（external link）。图 4-77 给出了五种通告以及它们的用法。

- **路由器链路。**路由器链路将路由器以结点的形式进行通告。除了给出所声明路由器的地址外，这种类型的通告还可以定义多种链路类型，这些链路将路由器连接到其他实体。过渡链路声明了通向过渡网络的链路，这个网络通过一个或多个路由器连接到网络的剩余部分。这种通告类型应该定义过渡网络的地址以及链路的代价。残桩链路通告通往残桩网络的链

路，这种网络不完全算是贯穿型网络（through network）。与前一个链路类似，通告再次定义网络地址以及代价。点对点链路应该定义点对点连线末端的路由器的地址以及到达它所需要花费的代价。

- **网络链路**。网络链路将网络作为一个结点进行通告。然而，由于网络不能通告自身（它是一个被动实体），因此其中的一个路由器被指派为指定路由器，并且由它进行通告。除了指定路由器的地址，这类 LSP 声明了所有路由器的 IP 地址（将指定路由器包含进去，但是把它作为路由器，而不是网络的代言结点（speaker））。但是通告不包含代价，因为当路由器发送路由器链路时，每个路由器声明了到达网络的代价。



a. 路由器链路　b. 网络链路　c. 通向网络的汇总链路　d. 通向AS的汇总链路　e. 外部链路

图 4-77　五种不同的 LSP

- **通向网络的汇总链路**。这通过区域边界路由器完成；它通告了由主干收集的通往一个区域的汇总链路，或者通告区域收集的通往主干的链路汇总。正如我们之前讨论的，将区域连接到一起的话就需要这种信息交换。
- **通向 AS 的汇总链路**。这通过一个 AS 路由器完成，它通告了从其他 AS 到当前 AS 主干区域的汇总链路，这些信息可以散播到区域中，这样它们就会了解其他 AS 中的网络。当我们讨论 AS 间路由选择（BGP）时将会更深入体会到交换这种信息的必要性。
- **外部链路**。这也是通过 AS 路由器来完成的。AS 路由器向主干区域声明 AS 之外存在一个单独网络。这个信息被散播到多个区域中。

OSPF 实现

OSPF 被以网络层程序的形式实现，它使用了 IP 服务来进行传播。IP 数据报携带来自 OSPF 的报文，并将协议字段的值设置为 89。这意味着，尽管 OSPF 是帮助 IP 在 AS 内部路由数据报的路由选择协议，但是 OSPF 报文被封装在数据报中。OSPF 经历了两个版本。绝大多数实现采用版本 2。

OSPF 报文　OSPF 是一个非常复杂的协议；它使用五个不同类型的报文。在图 4-78 中，我们首先给出 OSPF 公共头部的格式（用于所有报文）与链路状态通用头部（用于部分报文）。之后，我们给出 OSPF 中用到的五种报文类型的纲要。问候报文（hello message）（类型 1）被路由器用来向邻居介绍自己并声明它所知道的所有邻居。数据库描述报文（database description message）（类型 2）通常响应问候报文，允许一个新加入的路由器获取全部 LSDB。链路状态请求报文（link-state

request message）（类型 3）被需要特定 LS 的路由器发送。链路状态更新报文（link-state update message）（类型 4）是用于创建 LSDB 的主要 OSPF 报文。事实上，正如我们之前讨论的，这个报文有五种不同的版本（路由器链路、网络链路、面向网络的汇总链路、面向 AS 边界路由器的汇总链路以及外部链路）。链路状态确认报文（link-state acknowledgement message）（类型 5）用来保证 OSPF 中的可靠性；每个接收链路状态更新报文的路由器都需要确认它。

鉴定 如图 4-78 所示，OSPF 公共头部为报文发送器提供鉴定。正如我们在第 10 章将讨论的，这可以防止恶意实体向路由器发送 OSPF 报文并导致路由器成为某个路由选择系统的一部分，而实际上这个路由器不属于这个路由选择系统。



图 4-78 OSPF 报文格式

OSPF 算法 OSPF 执行了我们在前一节讨论的链路状态路由选择算法。然而需要在算法中加入一些改变和鉴定。

- 在每个路由器创建最短路径树之后，算法需要用它来创建响应的路由选择算法。
- 这个算法需要增强，使之能够发送和接收五种类型的报文。

性能

在结束本节之前，让我们来简要讨论 OSPF 的性能：

- **更新报文。** OSPF 中的链路状态报文有些复杂。它们也被泛洪到整个区域。如果区域很大，这些报文可能会造成很大的通信量并占用很大的带宽。
- **转发表的收敛。** 当 LSP 泛洪完成，每个路由器都可以创建它自己的最短路径树和转发表；收敛比较快。然而，每个路由器需要运行 Dijkstra 算法，这会花费一些时间。

- 健壮性。OSPF 协议比 RIP 协议更健壮。因为在接收到完整的 LSDB 之后，每个路由器是独立的并且不依赖于区域中的其他路由器。一个路由器的差错或失效不会像 RIP 中那样产生地影响其他路由器。

边界网关协议第四版（BGP4）

边界网关协议第四版（Border Gateway Protocol Version 4，BGPv4）是当今因特网中唯一的域间路由选择协议。BGP4 基于我们之前描述的路径向量算法，但是此处这个算法被修改了，使之能够提供信息，用以表示因特网中的网络能否到达。

介绍

BGP，尤其是 BGP4 是一个复杂的协议。在本节，我们介绍 BGP 的基本概念及其与域内路由选择协议（RIP 或 OSPF）的关系。图 4-79 给出一个带有四个自治系统的互联网例子。AS2、AS3 和 AS4 是残桩自治系统；AS1 是过渡自治系统。在我们的例子中，在 AS2、AS3 和 AS4 之间交换的数据应该穿过 AS1。

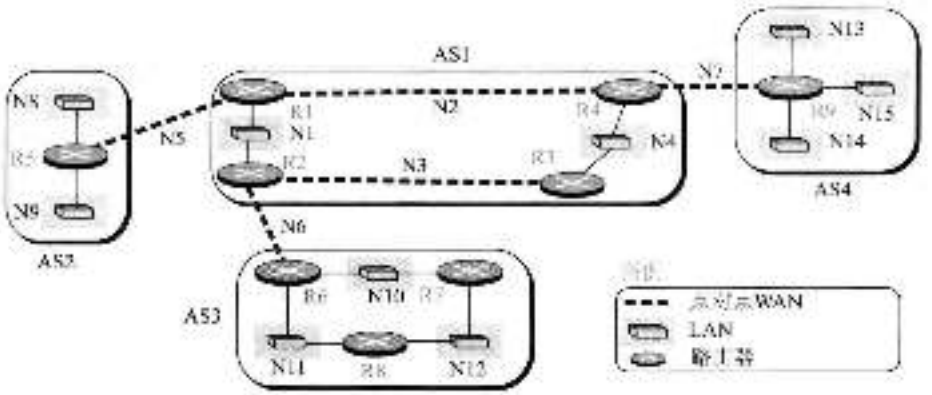


图 4-79 带有四个 AS 的互联网样例

图中每个自治系统使用一个公共域内协议，即 RIP 或 OSPF。每个 AS 中的路由器都知道如何到达自身 AS 中的网络，但是它并不知道如何到达另一个 AS 中的网络。

为了使每个路由器能够转分组路由到互联网中的任意网络中，我们首先在每个边界路由器（它位于每个 AS 的一条边上，这条边连接到另一个 AS 的路由器上）上安装 BGP4 的一种变体，称为外部 BGP（external BGP，eBGP）。之后，我们在所有路由器上安装 BGP 的第二种变体，称为内部 BGP（internal BGP，iBGP）。这意味着边界路由器将会运行三种路由选择协议（域内，eBGP 以及 iBGP），但是其他路由器运行两种协议（域内以及 iBGP）。我们分别讨论每个 BGP 变体的作用。

外部 BGP（eBGP）的运行 我们将 BGP 协议称为一种点对点协议。当软件运行在两个路由器上时，它们试图使用熟知端口号 179 创建一个 TCP 连接。换言之，一对客户进程和服务器进程持续进行通信来交换报文。运行 BGP 进程的两个路由器称为 BGP 对等结点（BGP peers）或 BGP 代言结点（BGP speaker）。我们将讨论两个对等结点之间所交换的不同报文类型，但是，目前我们只对更新报文（稍后讨论）感兴趣，这个报文声明了每个 AS 中网络的可达性。

这个 BGP 的 eBGP 变体允许两个通过物理连接到一起的边界路由器形成一对 eBGP 代言结点并交换报文，这两个边界路由器位于两个不同 AS 中。图 4-79 的例子中形成了三对符合条件的路由器：R1-R5、R2-R6 以及 R4-R9。这些对之间的连接建立在三个物理 WAN 之上（N5，N6 以及 N7）。然而，我们需要在物理连接上创建逻辑 TCP 连接以实现信息交换。在 BGP 语言中，逻辑连接称为会话（session）。这意味着我们的例子需要三个会话，如图 4-80 所示。

这幅图也给出了eBGP会话中由路由器发送的简化更新报文。带圈数字定义了每种情况下的发送路由器。例如1号报文被路由器R1发送并告知路由器R5以下信息：N1、N2、N3以及N4可以通过路由器R1到达（R1从相应的域内转发表中得到这个信息）。路由器R5现在可以将这些信息加入到它的转发表中。当R5接收到任何前往这四个网络的分组时，它可以使用这个转发表并找到下一跳路由器是R1。

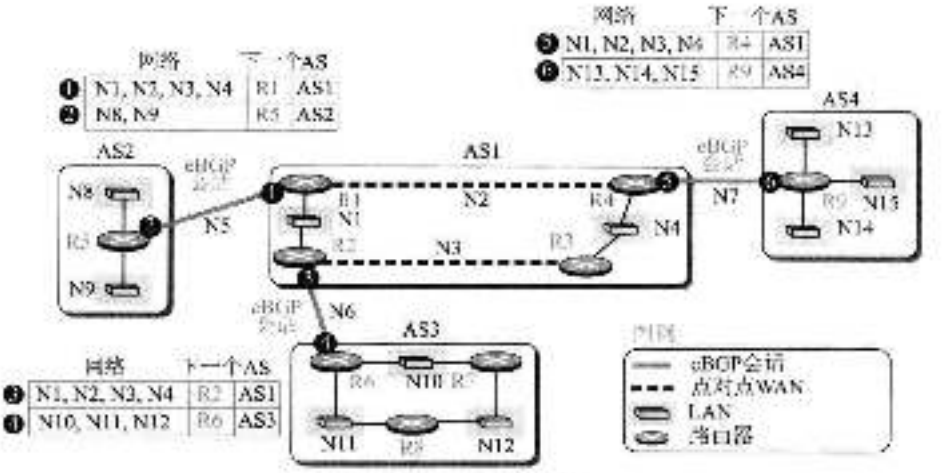


图4-80 eBGP的运行

读者可能已经注意到，在三个eBGP会话期间交换的报文帮助某些路由器了解如何将分组路由到互联网中的某些网络，但是可达性信息并不完整。有两个问题需要处理：

1. 某些边界路由器不知道如何路由一个去往非邻居AS的分组。例如，R5不知道如何路由去往AS3和AS4的分组。路由器R6和R9与R5的情况相同：R6不知道AS4中的网络；R9不知道AS3中的网络。

2. 没有一个非边界路由器知道如何路由一个去往其他AS中任意网络的分组。

为了处理以上两个问题，我们需要允许所有路由器对（边界或非边界）运行BGP协议的第二个变体，即iBGP。

内部BGP（iBGP）的运行 iBGP协议与eBGP协议类似，因为它在熟知端口179上使用TCP服务，但是它在自治系统中可能成对的路由器之间创建一个会话。然而，需要澄清几点。首先，如果一个AS只有一个路由器，就不会存在iBGP会话。例如，在互联网中，我们不能在AS2或AS4内部创建一个iBGP会话。第二，如果自治系统中有n个路由器，那么在那个自治系统（全连接网格）中应该存在$[n \times (n-1)/2]$个会话，用以防止系统中的循环。换言之，每个路由器需要在会话中向对等结点通告它自身的可达性，而不是将它在另一个会话中从另一个对等结点接收到的信息进行泛洪。图4-81给出了在我们的互联网中eBGP和iBGP会话的组合。

注意，我们没有给出AS内部的物理网络，因为会话是在覆盖网（TCP连接）完成的，可能覆盖了多个物理网络。这一点是由路由器决定的，而同时这个路由器由域内路由选择协议指定。还应注意，在这个阶段只交换四个报文。第一个报文（编号1）被R1发送，它声明网络N8和N9通过路径AS1-AS2是可达的，但是下一跳路由器是R1。这个报文通过一个独立的会话发送到R2、R3以及R4。路由器R2、R4和R6完成相同的事情，但是将不同的报文发送到不同的目的地。有趣的一点是，在这个阶段R3、R7和R8与它们的对等结点创建会话，但是它们实际上没有报文要发送。

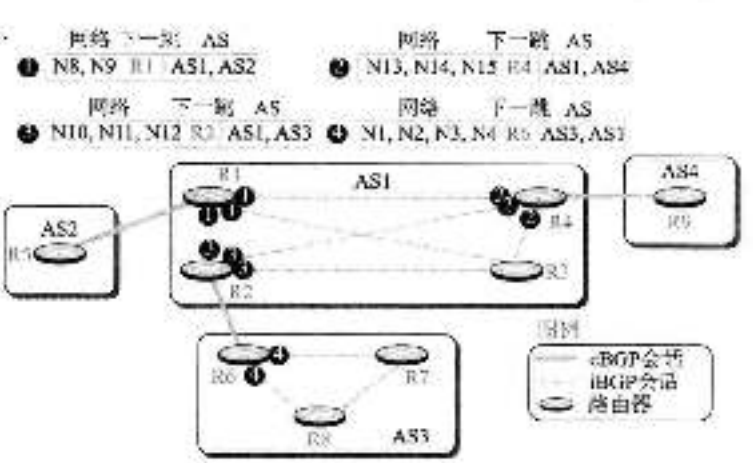


图 4-81 在我们的互联网中 eBGP 和 iBGP 会话的组合

这里更新过程不会停止。例如，在 R1 接收到来自 R2 的更新报文后，它将 AS3 的可达信息与已知的 AS1 的可达信息联合在一起，并且向 R5 发送一个新的更新消息。现在 R5 知道如何到达 AS1 和 AS3 中的网络了。当 R1 接收到来自 R4 的更新报文时这个过程将继续。关键在于我们需要确定在某个时间点，之前的更新没有变化了并且所有信息都被传播到了全部 AS。此时，每个路由器把从 eBGP 和 iBGP 接收到的信息结合起来，利用寻找最佳路径的标准创建一个称为路径表（path table）的东西，这些标准包括稍后会讨论的路由策略。为了便于展示，我们在图 4-82 中给出图 4-79 中路由器的路径表。例如路由器 R1 现在知道任何去往 N8 或 N9 的分组应该穿过 AS1 和 AS2，并且下一个传递分组的路由器是 R5。类似地，路由器 R4 知道任何去往 N10、N11 或 N12 的分组应该穿过 AS1 和 AS3，并且下一个传递分组的路由器是 R1，等等。

网络	下一跳	路径
N8, N9	R5	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

R1的路径表

网络	下一跳	路径
N8, N9	R1	AS1, AS2
N10, N11, N12	R6	AS1, AS3
N13, N14, N15	R1	AS1, AS4

R2的路径表

网络	下一跳	路径
N8, N9	R2	AS1, AS2
N10, N11, N12	R3	AS1, AS3
N13, N14, N15	R4	AS1, AS4

R3的路径表

网络	下一跳	路径
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

R4的路径表

网络	下一跳	路径
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

R5的路径表

网络	下一跳	路径
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

R6的路径表

网络	下一跳	路径
N1, N2, N3, N4	R5	AS3, AS1
N8, N9	R5	AS3, AS1, AS2
N13, N14, N15	R5	AS3, AS1, AS4

Path table for R7

网络	下一跳	路径
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

R8的路径表

网络	下一跳	路径
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

R9的路径表

图 4-82 最终化 BGP 路径表

域内路由选择信息插入 域间路由选择协议如 BGP，它的职责是帮助 AS 内部的路由器增加路由选择信息。换言之，路径表被 BPG 收集和组织，本质上说它不用来路由分组；它被插入到用于路由分组的域内转发表（RIP 或 OSPF）中。这可以用多种方式完成，取决于 AS 的类型。

在残桩 AS 情况下，只有区域边界路由器才在它的转发表末尾加入一个默认表项，并将下一跳路由器定义为位于 eBGP 连接末端的代言路由器。在图 4-79 中，AS2 中的 R5 将 R1 定义为除了 N8 和 N9 之外所有网络的默认路由器。AS4 中路由器 R9 的情况也是类似的，它将默认路由器定义为 R4。在 AS3 中，R6 将默认路由器设定为 R2，但是 R7 和 R8 将默认路由器设定为 R6。这些设

完与我们在图 4-82 中描述的各个路由器的路径表一致。换言之，通过加入一个默认表项，路径表被插入到域内转发表中。

在过渡 AS 情况下，情况更为复杂。AS1 中的 R1 需要将图 4-82 中路径表的全部内容插入到它的域内转发表中。R2，R3 和 R4 也面临相同的情况。

需要解决的一个问题是代价值。我们知道 RIP 和 OSPF 使用不同的度量。一种非常常见的方法就是将到达外部网络的代价设置为达路径中第一个 AS 的代价值。例如 R5 到达另一个 AS 中所有网络的代价就是其到达 N5 的代价。R1 到达 N10 再到 N12 的代价就是其到达 N6 的代价，等等。代价从域间转发表（RIP 或 OSPF）中获得。

图 4-83 给出域间转发表。为了简化，我们假设所有 AS 都使用 RIP 作为域内路由选择协议。阴影部分是 BGP 协议插入的增加部分；默认目的地用 0 表示。



图 4-83 从 BGP 插入之后的转发表

地址聚合 读者可能已经认识到 BGP4 协议所获得的域间转发表可能在全局因特网的情况下变得非常巨大，因为很多目的网络可能被包含进转发表中。幸运的是，正如我们本章之前讨论的，BGP4 使用前缀作为目的标识符并允许聚合这些前缀。例如前缀 14.18.20.0/26、14.18.20.64/26、14.18.20.128/26 以及 14.18.20.192/26 可以组合成 14.18.20.0/24，如果所有四个子网可以通过一条路径到达。即使一个或两个聚合前缀需要一个单独的路径，我们之前讨论的最长前缀原则也允许我们这样做。

路径属性

在域内路由选择协议（RIP 或 OSPF）中，目的端通常与两条信息有关：下一跳和代价。第一个信息给出传送分组的下一跳路由器的地址；第二个信息定义到达目的地的代价。域间路由选择更复杂，而且自然需要更多关于如何到达目的端的信息。在 BGP 中，这些信息称作路径属性（path attribute）。BGP 允许目的端与多达 7 个路径属性相关。路径属性分为两大类：熟知和可选。熟知属性必须被所有路由器识别；而可选属性不是这样。熟知属性可以是强制的，这意味着它必须在 BGP 的任意一个更新报文中出现；它也可以是任意的，这意味着它不必在 BGP 的任意一个更新报文中出现。可选属性可以是可迁移的，这意味着它可以传递到另一个 AS；也可以是不可迁移的，这意味着它不能传送到另一个 AS。所有属性都被插入到更新报文（后文会讨论）中相应目的端前缀的后面。属性的格式在图 4-84 中给出。



图 4-84 路径属性格式

每个属性的第一字节定义了四个属性标记（如图 4-84 所示）。下一位定义了 ICANN 分配的属性类型（如下面的解释，只分配了 7 种类型）。属性值长度定义了属性值字段的长度（不是整个属性部分的长度）。以下给出每个属性的简要描述。

- ORIGIN（类型 1）。这是熟知的强制属性，它定义了路由选择信息的源端。这个属性可以定义为以下三种值：1、2 和 3。数值 1 表示路径的信息从域内协议（RIP 或 OSPF）获得。数值 2 表示这个信息从 BGP 获得。数值 3 表示它来自未知源端。
- AS-PATH（类型 2）。这是熟知的强制属性，它定义了自治系统清单，穿过这些自治系统可以到达目的端。我们在例子中已经使用过这个属性了。正如我们在前面路径向量路由选择算法中描述的，AS-PATH 属性帮助防止循环。当一条将当前 AS 列为路径的更新报文到达路由器时，这个路由器丢弃那条路径。AS-PATH 也可以在路由选择中使用。
- NEXT-HOP（类型 3）。这是熟知的强制属性，它定义了数据分组应该被转发到的下一跳路由器。我们在例子中也使用了这个属性。正如我们所看到的，这个属性帮助将 eBGP 和 iBGP 运行中收集来的信息插入到域内路由选择协议中，如 RIP 或 OSPF。
- MULT-EXIT-DISC（类型 4）。多出口鉴别器（multiple-exit discriminator）是一个可选的不可迁移属性，它鉴别通向一个目的端的多个出口路径。这个属性的数值通常被相应域内协议的度量标准定义（一个四字节无符号整型的属性数值）。例如，如果一个路由器有多条通向目的端的路径，每条路径的这些属性值不同，那么具有最小值的路径被选出来。注意，这个属性是不可迁移的，这意味着它不能通一个 AS 传播到另一个。
- LOCAL-PREF（类型 5）。本地偏好（local preference）属性是熟知的可选属性，它通常由管理员设定，它基于组织机构的策略。管理员偏好的路由被给予较高的本地偏好数值（一个四字节无符号整型的属性数值）。例如，在带有 5 个 AS 的互联网中，AS1 的管理员可以设置路径 AS1-AS2-AS5 的本地偏好数值为 400，AS1-AS3-AS5 为 300 以及 AS1-AS4-AS5 为 50。这意味着，与第二条路径相比管理员偏好第一条，并且与第三条路径相比管理员偏好第二条。这种情况可能是，对于 AS1 的管理员来说 AS2 最安全并且 AS4 最不安全。如果前两条路径不可用，则选择最后一条路径。
- ATOMIC-AGGREGATE（类型 6）。这是熟知的可选属性，它不以聚合方式定义目的端前缀；它只定义一个目的端网络。这个属性没有数值字段，这意味着长度字段的值为 0。
- AGGREGATOR（类型 7）。这是可选的可迁移属性，它通告目的端前缀是聚合的。属性数值给出进行聚合的上一个 AS 数量，在这之后是进行聚合的路由器的 IP 地址。

路由选择

到本节为止，我们一直不讨论路由如何被 BGP 路由器选择，主要因为上文的简单例子中只有一条通向目的端的路由。在通向一个目的端的多个路由被接收到的情况下，BGP 需要从中选择一个。BGP 的路由选择过程不像基于最短路径树的域内路由选择协议中那样简单。BGP 的路由有一些附加其上的属性，并且它可能来自 eBGP 会话或 iBGP 会话。图 4-85 给出通常实现所使用的流程图。

路由器取出满足每步标准的路由。如果只抽出一条路由，它就被选定并且过程结束；否则，进行下一步，过程继续。注意，第一个选择与 LOCAL-PREF 属性相关，这反映了管理对路由实施的策略。

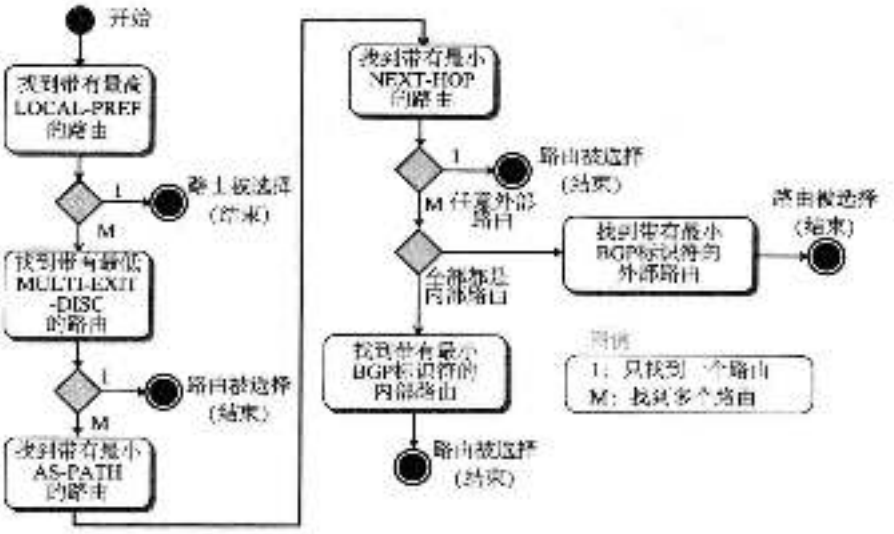


图 4-85 路由选择流程图

报文

BGP 使用四类报文用于 AS 间和 AS 内部 BGP 代言结点的通信：打开报文、更新报文、保活报文以及通知报文（见图 4-86）。所有 BGP 分组分享相同的公共头部。

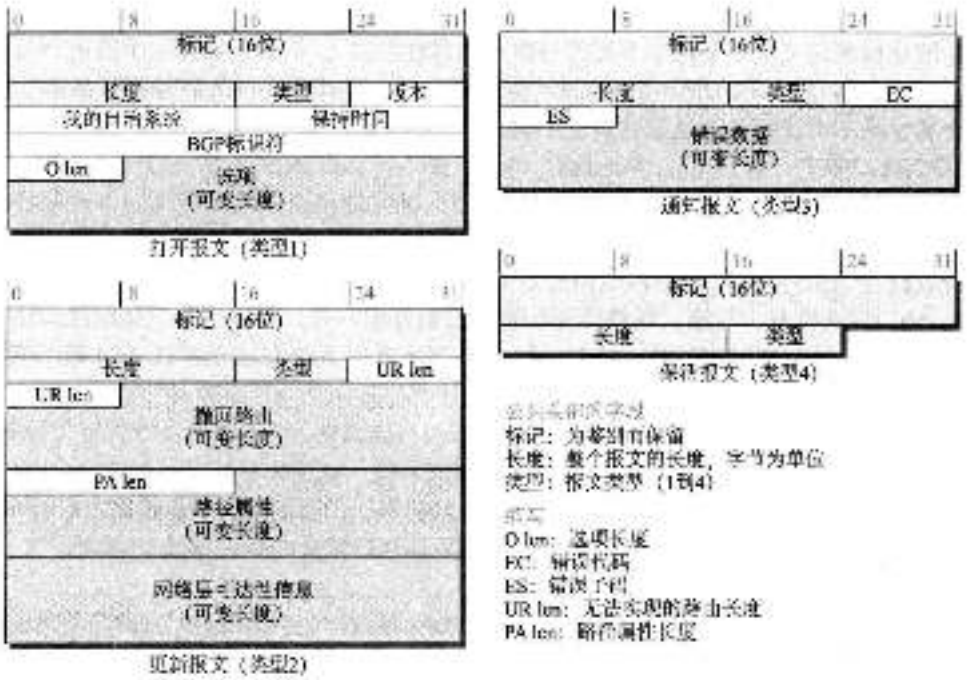


图 4-86 BGP 报文

- **打开报文**（open message）。为了创建邻居关系，运行 BGP 的路由器打开一个与邻居的 TCP

连接并发送一个打开报文。

- **更新报文**（update message）。更新报文是 BGP 协议的核心。它被路由器用来取出之前通知的目的端，以便声明通向新的目的端的路由，或者两者都做。注意，BGP 可以取出几个之前声明的目的端，但是，在一个更新报文中它只能声明一个新的目的端（或多个都有相同路径属性的目的端）。
- **保活报文**（keepalive message）。处于运行状态的 BGP 对等结点周期性交换保活报文（在它们的保持时间到达前）来彼此告知它们处于活跃状态。
- **通知**（notification）。当检查到错误情况或路由器想关闭会话时，路由器发送通知报文。

性能

BGP 性能可以和 RIP 比较。BGP 代言结点交换很多报文来创建转发表，但是 BGP 没有循环和计数到无穷。我们之前提过的 RIP 中关于传播失效和出错的缺点在 BGP 中也存在。

4.4 多播路由选择

如今因特网中的通信不只是单播；多播通信正在快速增长。在本节，我们首先讨论单播、多播以及广播背后的一般思想。之后我们讨论多播路由选择的一些基本问题。最终，我们讨论因特网中的多播路由选择协议。

4.4.1 介绍

前面我们已经学到路由器转发数据报通常是基于数据报中目的端地址的前缀，它定义了目的端主机连接到的网络。明白了以上的转发原则，我们现在可以定义单播、多播以及广播。让我们搞清楚这些与因特网有关的术语。

单播

在单播（unicasting）中，只有一个源端和一个目的端网络。源端和目的端网络的关系是一对一的。数据报路径中的每个路由器试图将分组转发到唯一一个端口上。图 4-87 给出一个小型互联网，其中单播分组需要被从源端计算机传递到连接到 N6 的目的端计算机。路由器 R1 负责通过接口 3 转发分组；路由器 R4 负责通过接口 2 转发分组。当分组到达 N6，传递的任务就落在了网络的层上；它或者向所有主机广播或者以太网交换机只将其传递到目的端主机。

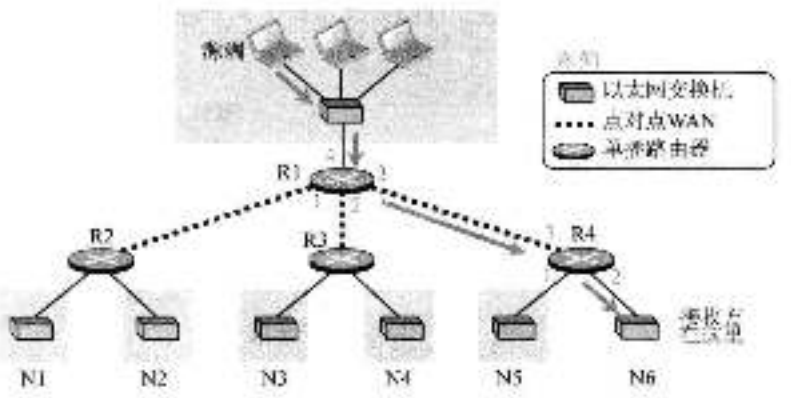


图 4-87 单播

多播

在多播（multicasting）中，存在一个源端和一组目的端，其关系是一对多的。在这类通信中，源地址是一个单播地址，而目的地址是一组地址，其中存在至少一个有兴趣接收多播数据报的组成

员，组地址定义组成员。图 4-88 给出图 4-87 中的一个小型互联网，但是路由器已经改成多播路由器（或者之前的路由器已经被配置成为能完成两种工作的路由器）

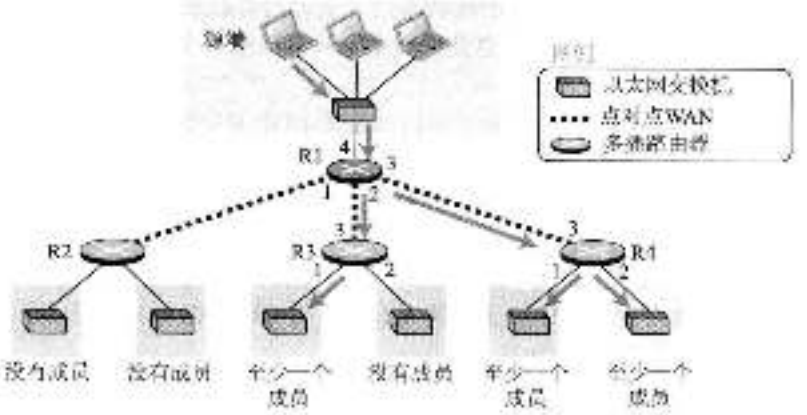


图 4-88 多播

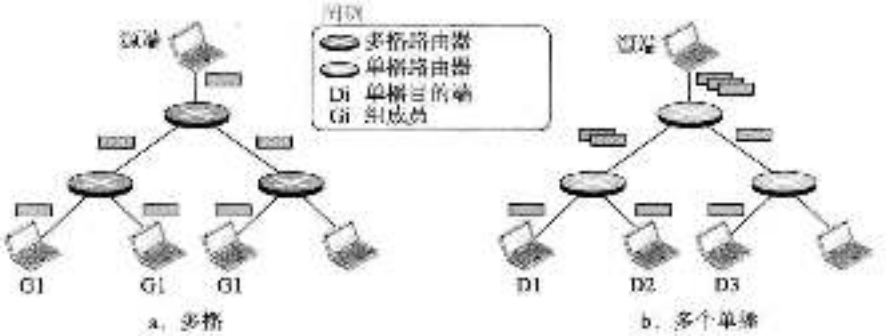
在多播中，多播路由器可能需要通过它的多个端口将相同数据报的副本发送出去。在图 4-88 中，路由器 R1 需要从端口 2 和 3 发送数据报。类似地，路由器 R4 需要从两个端口中发送数据报。然而，路由器 R3 知道没有成员属于端口 2 所连接的子域的组；它仅仅从接口 1 发送出数据报。

多播与多个单播

我们需要区分多播和多个单播。图 4-89 说明了这两个概念。

多播从源地址开始时是单个分组，这个分组被路由器复制。每个分组中的目的地址对所有的副本都是相同的。注意，在任何两个路由器之间只有分组的一份副本。

在多个单播（multiple unicasting）中，从源端发出多个分组。例如，如果有 3 个目的端，源端发送三个分组，每个都有不同的单播目的地址。注意，在两个路由器之间可能有多个副本在传递。当一个人发送一封电子邮件给一组人，这就是多个单播。电子邮件软件创建报文的多个副本，其中每一个具有不同的目的地址并逐一发送它们。这不是多播，而是多个单播。



a. 多播　　b. 多个单播

图 4-89 多播与多个单播

用单播仿真多播

我们可能会问当多播可用单播仿真时，为什么还要设立这样一个单独的机制。对于这个问题有两点明显的理由：

1. 多播比多个单播效率更高。在图 4-89 中我们看到多播比多个单播要求带宽小，在多个单播中，某些链路必须处理多个副本。

2. 在多个单播中，由源端生成一些分组，这些分组之间有相对的延迟。如果有1000个目的地址，第一个分组与最后一个分组之间的延迟可能是不可接受的。如果在多播中，因为源端仅生成一个分组，因此没有延迟。

多播应用

当今多播有很多应用，如访问分布式数据库（distributed database）、信息发布、电话会议和远程学习。

- **访问分布式数据库。**当前数据库大多是分布式的，即信息通常在生成时存储在多个地方。需要访问数据库的用户不知道信息的地址。用户的请求是向所有数据库多播，而有该信息的地方响应。
- **信息发布。**商业机构时常需要向它们的客户发送信息。如果对每个客户来说信息都是相同的，那么它可以多播。采用这种方式，一个商业机构可向多个客户发送一个报文。例如，可向购买某个零售软件包的所有客户发送一个软件更新。类似地，可以容易地通过多播发布新闻。
- **电话会议。**电话会议（teleconferencing）包含多播。所有出席会议的人都在同一时间接收到相同的信息。为此，可构成临时组或永久组。
- **远程学习。**多播使用中一个正在成长的领域是远程学习（distance learning）。某一教授讲的课可被一个特定组的学生接收到。这特别适用于那些不能到大学课堂听课的学生。

广播

广播意味着一对多通信：一个主机向互联网中的所有主机发送分组。因特网层次上没有提供这种广播，显然这是因为它可能造成很大的通信量并占用大量带宽。然而，因特网中完成了部分广播。例如，某些对等结点应用（peer-to-peer application）可能使用广播来访问所有对等结点。受控广播也可能在域内（区域或自治系统）实现，这作为实现多播的一个步骤。当我们讨论广播协议时，我们讨论这些受控广播类型。

4.4.2 多播基础

在讨论因特网多播路由选择协议之前，我们需要讨论某些多播基础：多播地址、收集多播组信息以及多播最优树。

多播地址

当我们向目的端发送一个单播分组时，分组的源地址定义了发送端，分组目的地址定义了分组接收端。在多播通信中，发送端只有一个，但是接收方有多个，有时成千个或上百万个接收方分布在世界上。应该清楚的是，我们不能包含分组中所有接收者的地址。正如在因特网协议（Internet Protocol，IP）中描述的，分组目的端地址应该只有一个。因此，我们需要多播地址。一个多播地址定义了一组接收者，而不是一个。换言之，多播地址是多播组的一个标识符。如果一个新的多播组由一些活跃成员组成，权威机构可以向这个组分配一个唯一的多播地址来唯一地定义它。这意味着分组通信的源地址可以是唯一定义发送方的单播地址，而目的地址可以是定义一个多播组的多播地址。在这种方式下，如果一台主机是 n 个多播组的成员，那么它事实上有（$n+1$）个地址：一个单播地址，它用作单播通信源地址或目的地址，以及 n 个多播地址，它仅用作目的地址来接收发送到这个组的报文。图4-90给出了概念。

IPv4中的多播地址

路由器或目的主机需要区分单播和多播数据报。IPv4和IPv6都出于此目的分配了一个地址块。在本节中，我们仅讨论IPv4多播地址；IPv6多播地址在本章之后讨论。在IPv4中多播地址属于一

大块地址，它们按此目的而特别设计。在分类寻址中，所有 D 类地址由这些地址组成；无类寻址使用同一个块，但是它称为块 224.0.0.0/4（从 224.0.0.0 到 239.255.255.255）。图 4-91 给出二进制形式的块。前四个比特位定义了块；其余比特位用来作组的标识符。

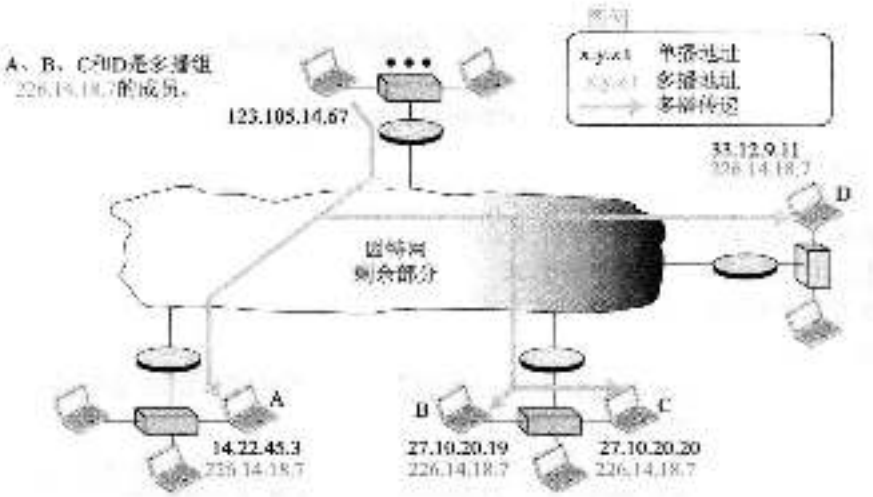


图 4-90 需要多播地址



图 4-91 二进制形式的多播地址

多播地址块中的地址数目是十分巨大的（2^{28}）。我们肯定不会有如此众多的多播组。然而，一个块划分为几个子块，每个子块用于特定多播应用。以下给出一些公共子块：

- **本地网络控制块**（Local Network Control Block）。子块 224.0.0.0/24 被分配给用于网络内部的多播路由选择协议。这意味着带有这个范围内目的地址的分组不能被路由器转发。在这个子块内，地址 224.0.0.0 被保留，地址 224.0.0.1 用于将数据报发送到网络内的所有主机和路由器，地址 224.0.0.2 用于将数据报发送到网络内所有路由器。正如我们之后讨论的，剩余地址被分配给某些多播协议用于通信。
- **因特网控制块**（Internetwork Control Block）。224.0.1.0/24 子块被分配给用于整个互联网的多播路由选择协议。这意味着带有这个范围内目的地址的分组可以被路由器转发。
- **特定源多播块**（Source-Specific Multicast (SSM) Block）。232.0.0.0/8 块用于特定源多播协议。当我们本章之后讨论 IGMP 协议时，我们讨论 SSM 路由选择。
- **GLOP 块**（GLOP Block）。233.0.0.0/8 块称为 GLOP 块（GLOP 不是首字母缩略词或缩写词）。这个块定义了可用于自治系统（AS）内部的地址范围。正如我们之前学到的，每个自治系统都被分配一个 16 位号码。我们可以将 AS 号码作为一个块的中间两个八位字节，从而创建 256 个多播地址（233.x.y.0 到 233.x.y.255），其中 x.y 是 AS 号码。
- **管理范围块**（Administratively Scoped Block）。239.0.0.0/8 块称为管理范围块。这个块的地址用于因特网的特定区域。如果分组的目的地址属于这个范围，那么它不应该离开这个区域。换言之，这个块中的地址被限制在一个组织机构中。

选择多播地址

选择一个分配给组的多播地址不是一件容易的工作。地址选择取决于应用类型。让我们讨论一

此情况。

有限组 管理者可以使用AS号码 $(x.y)_{256}$ 并选择239.x.y.0到239.x.y.255（管理范围块）之间的地址作为特定组的多播地址，而其他组不使用这些地址。举例来说，假设大学教授们为了和学生交流需要创建多播组地址。如果大学所属的AS号码是23452，这可以写成 $(91.156)_{256}$，这给大学256个地址：233.91.156.0到233.91.156.255。大学管理机构可以给每个教授分配一个此范围内的地址。之后，该便成为多播组地址，教授用它来向学生们发送多播通信。然而，分组不能超越大学AS范围。

较大组 如果组扩展到AS范围之外，之前的解决方法不起作用。多播组需要从SSM块（232.0.0.8）中选择一个地址。使用这个块中的地址不需要获得许可，因为特定源多播中的分组基于多播组和源地址被路由；它们是唯一的。

收集关于组的信息

创建单播和多播路由选择的转发表需要两步：

1. 路由器需要知道它连接到哪些目的端。

2. 每个路由器需要向所有其他结点广播第一步获得的信息。这样每个路由器都知道其他路由器连接到哪个目的端。

在单播路由选择中，第一步的信息收集是自动的；每个路由器都知道它连接到哪个网络上，网络前缀（在CIDR中）是路由器所需要的。我们在之前章节描述的路由选择协议（距离向量或链路状态）负责自动向互联网中的其他路由器传播收集到的信息片段。

在多播路由选择中，第一步的信息收集不是自动的，有两个原因。第一，路由器不知道所连网络中哪台主机是特定多播组的成员；多播组的成员身份与网络前缀没有关系。我们已经给出，如果一台主机是一个多播组的成员，它拥有与组有关的单独多播地址。第二，成员身份不是主机的固定属性；即使在短时间内，主机也可能加入一些新的组并离开其他组。因此，路由器需要帮助才能找到每个接口中哪个组是活跃的。在收集这些信息之后，正如我们之后将要讨论的，路由器可以向其他路由器传播成员身份。图4-92给出单播和多播通告在第一步的不同点。对于单播，路由器不需要帮助；对于多播，它需要另一个协议的帮助，我们稍后讨论。

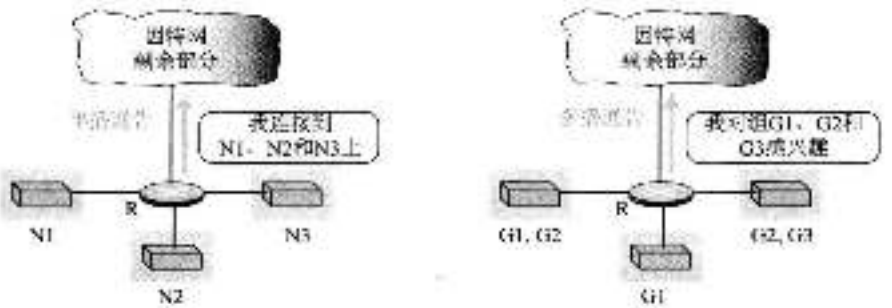


图4-92 单播与多播通告

在单播情况下，路由器R知道带有前缀N1、N2和N3的主机连接到它的接口上；它向因特网剩余部分传播这个信息。在多播情况下，路由器R需要知道在连接到它的端口的网络中，存在至少带有一个位于组G1、G2和G3中的忠诚成员的主机。总之，为了单播路由选择，我们仅需要每个域中的路由选择协议来传播关于路由器链路的信息；在多播中，我们需要两个协议：一个收集这些信息，另一个传播它们。在我们讨论第二步涉及的协议之前，我们需要讨论第一步所涉及的协议。

因特网组管理协议（IGMP）

如今这个用于收集组成员信息的协议是**因特网组管理协议**（Internet Group Management Protocol，IGMP）。IGMP 是在网络层定义的协议；它是一个辅助协议，就像 ICMP 一样被认为是 IP 的一部分。IGMP 报文，像 ICMP 报文一样被封装在 IP 数据报中。IGMP 第三版中只有两种报文类型：查询和报告报文，如图 4-93 所示。

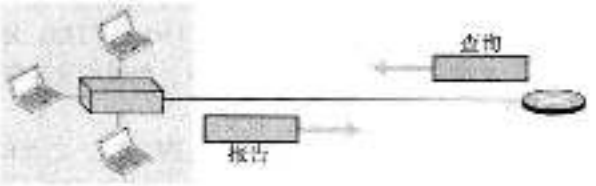


图 4-93 IGMP 运行

- **查询报文**（query message）。查询报文周期性地由路由器发送给所有连接其上的主机，让各主机报告它们对于组内成员身份的兴趣。在 IGMPv3 中，查询报文可以采取以下三种形式之一：

a. 查询任意组中成员身份的普通（general）查询报文被发送。它被封装在数据报中，目的端地址是 244.0.0.1（所有主机和路由器）。注意，所有连接到同一个网络的路由器都接收到这个报文，这样它们就知晓这个报文已经被发送并且应该抑制重发。

b. 特定组（group-specific）查询报文被从路由器发送，查询与特定组相关的成员。当路由器没有接收到特定组的响应且想确定网络中哪个组没有活跃成员时，就发送这个报文。组标识符（多播地址）在这个报文中被提及。报文被封装在数据报中，其中目的地址被设置为相应的多播地址。尽管所有主机都接收这个报文，但是不感兴趣的可以丢弃它。

c. 当报文来自一个或多个特定源端时，特定源和目的（source-and-group-specific）查询报文被从路由器发送，以此来寻找与特定组相关的成员。当路由器没有收到与一个或多个特定的主机相关的特定组时，这个报文被重次发送。这个报文被封装在数据报中，目的地址被设置为相应多播地址。尽管所有主机都接收这个报文，但是不感兴趣的可以丢弃它。

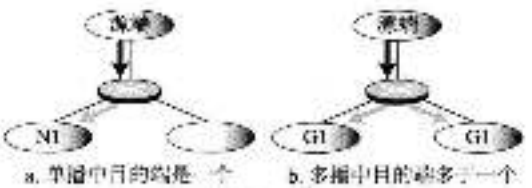
- **报告报文**（report message）。主机发出报告报文时，把它作为查询报文的响应。报文包含记录列表，其中每条记录给出相应组的标识符（多播地址）以及所有主机感兴趣的源地址。主机想从这些源地址接收报文（包含）记录也提及一些主机不感兴趣的源地址，主机不想从它们那里接收组报文（排除）。报文被封装在数据报中，多播地址是 224.0.0.22（分配给 IGMPv3 的多播地址）。

在 IGMPv3 中，如果主机需要加入一个组，那么它要等待，直到它接收到一个查询报文并发送一个报告报文。如果主机需要离开组，它不响应查询报文。如果没有其他主机响应相关报文，组就被从路由器数据库中清除。

多播转发

多播中另一个重要问题是路由器需要作出转发多播分组的决定。单播和多播通信中的转发在两方面有所不同。

1. 在单播通信中，分组的目的地址只定义了一个目的端，分组仅需从其中几个端口发出，端口是用最小代价到达目的端的最短路径树的一个分支。在多播通信中，分组的目的端定义了一个组，但是那个组可能在互联网中有多个成员。为了到达所有目的端，路由器可能必须从一个以上的端口发出分组。图 4-94 给出这个概念。在单播中，目的网络 N1 不能处于互联网的多个部分；在多播中，组 G1 可能拥有多个成员，它们位于互联网多个部分。



a. 单播中目的端是一个　b. 多播中目的端多于一个

图 4-94 单播和多播的目的端

2. 单播通信中的转发决定取决于分组的目的地址。多播通信中的转发决定取决于分组的目的

地址和源地址。换言之，在单播中，转发是基于分组应该去哪里；在多播中，转发是基于分组应该去哪里以及分组从哪里来。图 4-95 给出了这个概念。在 a 中，源端处于互联网中没有组成员的部分。a 中路由器需要从两个接口发出分组；在 b 中，路由器只需要从一个接口发出分组，从而避免从接收端口再发出分组。换言之，在图中 b 部分，组 G1 的多个成员在分组到达路由器时就已经接收到了分组的副本；而朝着源端方向发送分组是没有帮助的，反而会增加通信量。这展示出多播通信中的转发取决于源地址和目的地址。

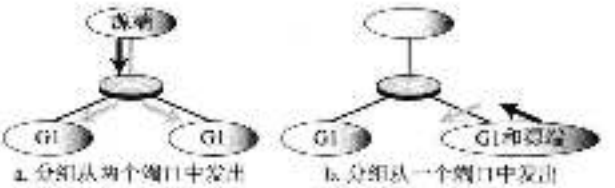


图 4-95 依赖目的端和源端的转发

两种多播方式

像单播路由选择一样，在多播路由选择中，我们需要创建路由选择树来最优地选择分组从源端到目的端的路由。然而，正如我们之前讨论的，每个路由器所做出多播路由选择决定不仅仅取决于分组的目的端，也取决于分组的源端。路由选择过程中涉及源端使得多播路由选择比单播路由选择更加复杂。因此，多播路由选择中发展出了两种不同的方法：使用基于源树路由选择以及使用组共享树路由选择。

基于源树方法

在基于源树（sourced-based tree）方法中，每个路由器需要为每个源端组的组合创建一个单独的树。换言之，如果互联网中有 m 个组和 n 个源端，那么路由器需要创建 $(m \times n)$ 个路由选择树。在每棵树中，相应源端是根结点，组的成员是叶子结点，路由器自身位于树的某处位置。我们可以将这种情况与单播路由选择比较，后者中的路由器仅需一棵树，其中自身为根结点，所有互联网中的网络为叶子结点。尽管看起来，可能每个路由器需要创建并存储大量关于这些树的信息。但是如今的因特网中有两种协议使用这个方法，我们后面会讨论到。这些协议使用某些策略来减轻这个情况。

组共享树方法

在组共享树（group-shared tree）方法中，我们将一个路由器指派为每个组的伪源端（phony source）。被指派的路由器称为核心路由器（core router）或者会合点路由器（rendezvous-point router），它代表这个组。任何源端如果需要将分组发送到那个组，那么它直接将分组发送给中央核心（单播通信）。中央核心负责多播。中央核心创建一个路由选择树，它自身是树的根结点并且任何带有这个组中活动成员的路由器都是叶子结点。在这个方法中，存在 m 个核心路由器（每组一个），每个核心路由器有一个路由选择树，共有 m 棵树。这意味着，路由选择树的数目从基于源树方法的（$m \times n$）个减少到这个方法中的 m 个。读者可能注意到，我们将从源端到所有组成员的多播传递分成了两部分。第一部分是从源端到核心路由器的单播传递；第二部分是从核心路由器到所有组成员的传递。注意，传递的第一部分需要使用管道完成。源端创建的多播分组需要封装在单播分组中，并且被发送到核心路由器。核心路由器解封单播分组，抽取多播分组然后将它发送给组成员。尽管很吸引人的是这个方法中涉及的数量减少了，但是这个方法有自己的开销：使用算法从所有路由器中选择一个作为组的核心路由器。

4.4.3 域内路由选择协议

在过去的几十年内，出现了很多域内多播路由选择协议。在本节，我们讨论其中的三种。两种是单播路由选择协议（RIP 和 OSPF）的扩展，它使用基于源树方法；第三种是独立协议，它变得越来越流行。域内多播路由选择协议可以按两种模式使用，采用基于源树方法或者采用组共享树方法。

多播距离向量（DVMRP）

距离向量多播路由选择协议（Distance Vector Multicast Routing Protocol，DVMRP）是用于单播路由选择的路由选择信息协议（RIP）的扩展。它使用基于源树方法来多播。值得注意的是，这

个协议中，路由器接收广播要被路式转发的多播分组，它按以下三步创建一个基于源端的多播树：

1. 路由器使用一种称为逆路径转发（reverse path forwarding，RPF）的算法来仿真创建源端和自身之间的一部分最优基于源树。

2. 路由器使用一种称为逆路径广播（reverse path broadcasting，RPB）的算法创建广播（生成）树，根结点是路由器自身，叶结点了是互联网中的所有网络。

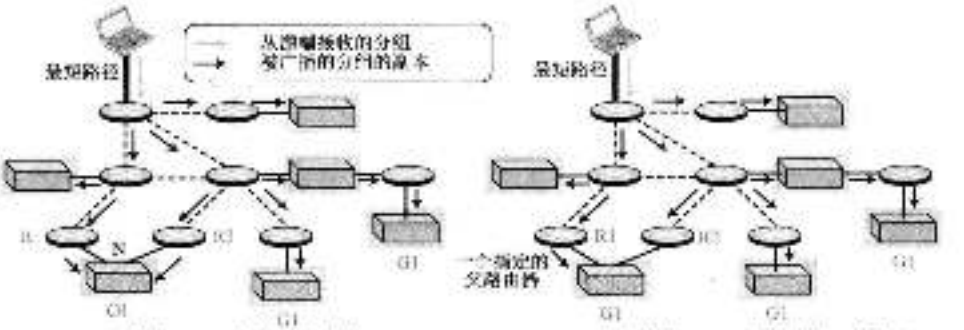
3. 路由器使用一种称为逆路径多播（reverse path multicasting，RPM）的算法，通过减去树中以组中没有成员的网络结束的枝干来创建多播树。

逆路径转发（RPF）

第一种算法逆路径转发（RPF）迫使路由器从一个特定的接口转发多播分组；这个接口穿过最短路径从源端到达路由器。如果没有以源端为根的最短路径树，路由器如何知道那个接口处在这样的最短路径中？路由器使用我们在单播路由选择中讨论过的最短路径树的第一条性质，即从 A 到 B 的最短路径也是从 B 到 A 的最短路径。路由器不知道从源端到自身的最短路径，但是它可以找到从自身到源端的最短路径中下一跳路由器（逆路径）。路由器仅需要查询它的单播转发表，假装它想要发送一个分组到源端；转发表给出下一跳路由器以及沿着逆向分组应该从那个接口发出。路由器使用这个信息来接收多播分组，仅当分组来自这个接口时才会接收。这是为了避免循环。在多播中，分组可能到达了已经转发过这个分组的路由器。如果路由器没有丢弃除了第一个以外所有的到达分组，那么多个分组的副本将会在互联网中循环。当然，当分组第一次到达时，路由器可以加入标签，并且丢弃带有相同标签的分组，但是 RPF 策略更简单。

逆路径广播（RPB）

RPF 算法帮助路由器仅仅转发一个来自源端的副本并丢弃其余副本。然而，当我们考虑第二步中的广播时，我们需要记住，目的端是互联网中的所有网络（LAN）。为了高效，我们需要防止每个网络接收一个以上分组副本。如果一个网络连接到多个的路由器上，它可能从每个路由器接收到一个分组的副本。RPF 在这里无法帮忙，因为网络没有实施 RPF 算法的智能；我们需要只允许一个连接到网络的路由器将分组传递到网络。一种方法是只指派一个路由器作为与特定源相关的网络的父路由器（parent）。当非父路由器接收到多播分组时，它丢弃这个分组。有很多方式可以选择网络的父路由器；一种方法是选择一个拥有到源端最短路径的路由器（使用单播转发表，再次沿着逆向）。如果这种情况中有并列关系，那么选择 IP 地址最小的。读者可能会注意到实际上 RPB 从 RPF 算法创建的图中创建广播树。RPB 减枝以免造成循环。如果我们使用最短路径标准来选择父路由器，实际上我们已经创建了最短路径广播树。换言之，在这步之后，我们有了一棵源端作为根结点、所有网络（LAN）作叶子结点的最短路径树。每个从源端出发的分组，穿过最短路径到达互联网中所有 LAN。图 4-96 给出网络中 RPB 如何通过为网络 N 分配指定的父路由器 R1 来避免重复接收。



a. 使用RPF，N接收两个副本　　b. 使用RPB，N只接收一个副本

图 4-96　RPF 与 RPB

逆路径多播（RPM）

正如你可能注意到的，RPB 对分组进行广播而不是多播。这样做并不高效。为了提高效率，多播分组必须只抵达那些拥有特定组中活跃成员的网络。这被称为逆路径多播（RPM）。为了将广播最短路径树变成多播最短路径树，每个路由器需要修剪（使之不活跃）一些端口，那些端口无法到达拥有源端组中活跃成员的网络。这一步可以通过自底向上即从叶结点到根结点的方式完成。在叶结点层，连接到网络上的路由器使用前面讨论过的 IGMP 协议收集成员信息。之后，网络中的父路由器使用逆最短路径树从路由器向源端传播消息。距离向量报文从一个邻居发送到另一个邻居。当路由器接收到所有这些有关成员的报文后，它便知道那个接口应该被修剪。当然由于这些分组是被周期性传播的，如果一个新成员加入网络，所有路由器都被通知到，并且可以相应地改变它们的接口状态。加入和离开不断进行。图 4-97 描述了 RPM 中的修剪如何仅仅使带有组成员的网络接收到分组的副本，除非那些网络位于通往带有成员网络的路径上。

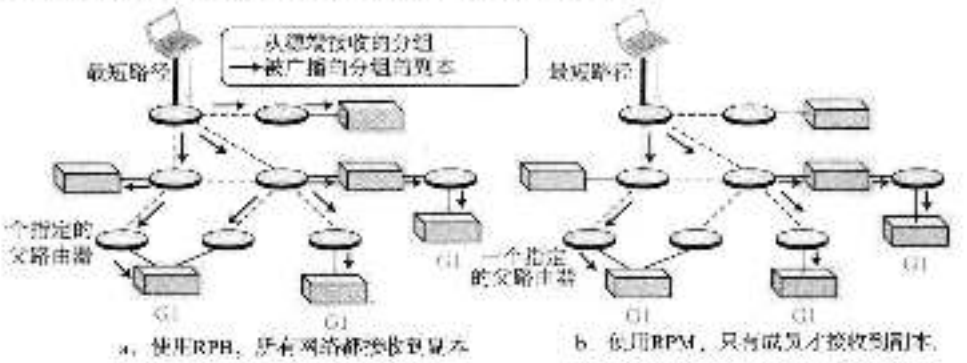


图 4-97 RPB 与 RPM

多播链路状态（MOSPF）

多播开放最短路径优先（Multicast Open Shortest Path First，MOSPF）是开放最短路径优先（OSPF）协议的扩展，也用于单播路由选择。它也使用基于源树方法来进行多播。如果互联网运行单播链路状态路由选择算法，这个思想可以被扩展来提供多播链路状态路由选择算法。请回忆，在单播链路状态路由选择中，互联网中的每一个路由器都有一个链路状态数据库（LSDB），路由器用它来创建最短路径树。为了将单播扩展到多播，正如单播距离向量路由选择中一样，每个路由器需要用另外一种数据库，以给出哪个接口拥有特定组中的活跃成员。现在，路由器经过以下步骤转发从源端 S 接收到的多播分组，并且这个分组发往目的端 G（一组接收者）：

1. 路由器使用 Dijkstra 算法创建一个最短路径树，S 是根结点并且互联网的目的端是叶子结点。注意，这个最短路径树不同于路由器通常使用的单播转发树，在后者中，树的根结点是路由器自身。在这种情况下，树的根结点是分组源地址定义的分组的源端。路由器能够创建这棵树，因为它有 LSDB，即互联网的整个拓扑；无论哪个路由器使用它，Dijkstra 算法都可以用来创建任何根结点的树。我们需要记住的要点是，这种方式创建的最短路径树取决于特定的源端。对于每个源端我们需要创建一个不同的树。

2. 路由器在第一步创建的最短路径树中找到自身。换言之，路由器创建了最短路径子树，自己作为子树的根结点。

3. 最短路径子树事实上是一个广播子树，根结点是路由器，所有网络都是叶子结点。路由器使用与 DVMRP 中描述的类似策略来修剪广播树，使其成为一棵多播树。IGMP 协议用来在叶子结点层级搜寻信息。MOSPF 加入了一个新的链路状态更新报文，它将成员身份泛洪到所有路由器。路由器可以使用以此方式接收到的信息，并且修剪广播树从而创建多播树。

4. 路由器现在可以只从那些相当于多播树分支的端口中转发接收到的分组。我们需要确保多播分组的副本能够到达所有含有活跃成员的网络，并且确保分组的副本不到达那些不含有活跃成员的网络。

图 4-98 给出一个例子，按照以上步骤利用一张拓扑图生成一棵多播树。简便起见，我们没有给出网络，但是我们将组连接到每个路由器上。图 4-98 给出了以源端为根结点的基于源树是如何创建的，以及如何将其改变成一个根结点是当前路由器的多播子树。

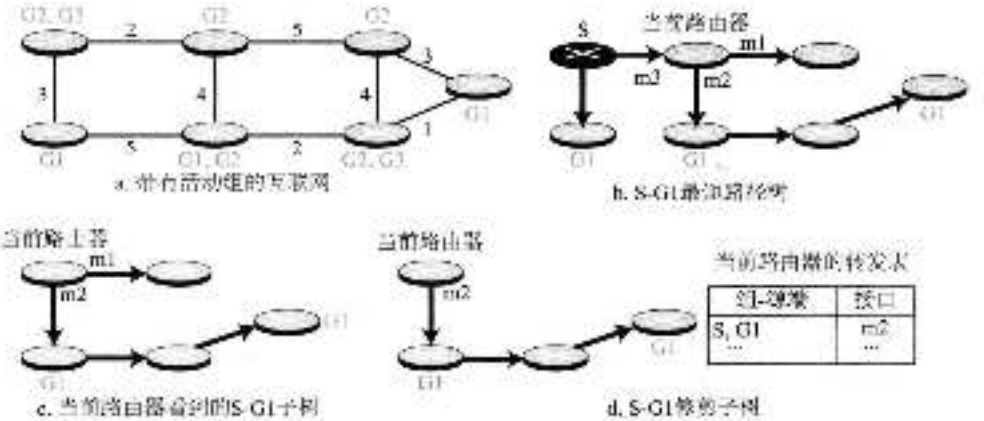


图 4-98 MOSPF 树信息的例子

协议独立多播（PIM）

协议独立多播（Protocol Independent Multicast，PIM）是对一种公用协议的称呼，这种协议将要单播路由选择协议，但是单播协议可以是距离向量协议或者链路状态协议。换言之，PIM 需要使用单播路由选择协议的转发表来找到去往目的端的下一跳路由器，但是不管转发表是如何创建的。PIM 还有其他有趣的特性，它可以按两种模式工作：密集（dense）模式以及稀疏（sparse）模式。这里，密集这个词的意思是互联网中组内活跃成员数量巨大；路由器拥有组内成员的可能性高。例如，这种情况可能发生在拥有很多成员的受大众欢迎的电话会议上。另一方面，稀疏的意思是互联网中只有少量路由器拥有组内活跃成员；路由器拥有组内成员的可能性低。例如，这种情况可能发生在非常专业的电话会议上，会议成员分组在互联网的某些位置。当协议在密集模式下工作时，它称为 PIM-DM；当协议在稀疏模式下工作时，它称为 PIM-SM。我们接下来解释这两个协议。

协议独立多播-密集模式（PIM-DM）

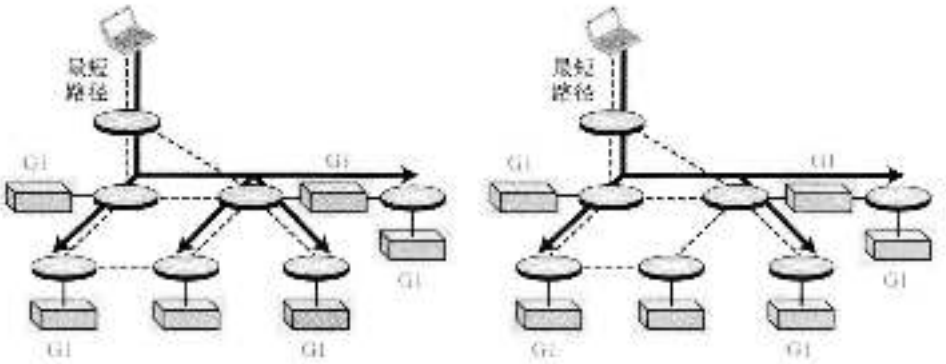
与互联网中路由器的数目相比，当带有组成成员的路由器的数目巨大时，PIM 工作在密集模式，称为 PIM-DM。在这种模式中，协议使用基于源树方法，并且与 DVMRP 类似，但是更简单。PIM-DM 只使用 DVMRP 中描述的两个策略：RPF 和 RPM。但是不像 DVMRP，等待第一个子树修剪的时候，分组转发不会暂停。让我们来解释 PIM-DM 中使用的两个步骤来弄清这个问题。

1. 某路由器接收到从源端 S 到组 G 的分组，它首先使用 RPF 策略避免接收重复的分组。如果它想要发送报文到源 S（沿逆向），它询问底层的单播协议的转发表来找到下一跳路由。如果没有来自逆向的下一跳路由器的分组到达，那么它丢弃分组并沿那个方向发送一个修剪报文，防止接收到与（S，G）相关的后续分组。

2. 如果第一步的分组来自逆向的下一跳路由器，接收路由器将分组从所有的接口进行转发，这些接口不包含分组到达的接口以及已经接收到与（S，G）相关的修剪报文的接口。注意，如果分组是从源端 S 到组 G 的第一个分组，这实际上是广播而不是多播。然而，每个收到不想要分组的下游路由器都向上游发送一个修剪报文，最终广播被改为多播。注意，DVMRP 有所不同：在通过非修剪端口发送任何报文之前，它要求修剪报文（DV 分组的一部分）先到达并且树被修剪。

PIM-DM 不关心这个预防措施，因为它假设绝大多数路由器对组感兴趣（泛洪模式的思想）。

图 4-99 给出 PIM-DM 背后的思想。第一个分组被广播到所有网络中，这些网络有或没有成员。当来自无成员的路由器的修剪报文到达之后，第二个分组就只是多播了。



a. 第一个分组被广播　　b. 第二个分组被多播

图 4-99　PIM-DM 背后的思想

协议独立多播-稀疏模式（PIM-SM）

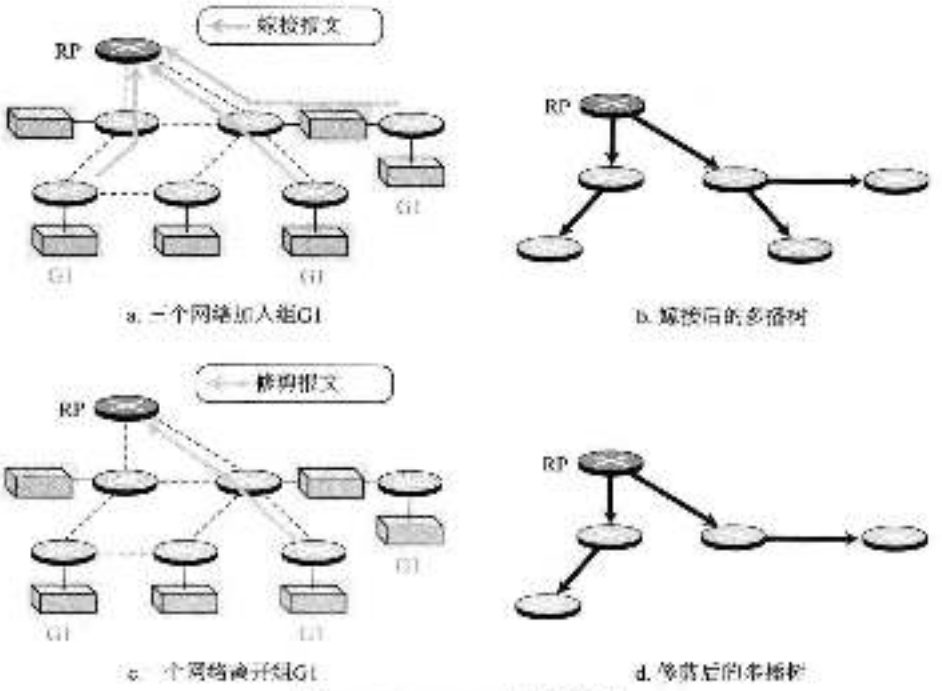
与互联网中路由器的数目相比，当带有组成员的路由器的数目较小时，PIM 工作在稀疏模式，称为 PIM-SM。在这种环境中，使用广播分组直到树被修剪的协议是不正确的；PIM-SM 使用组共享树方法来进行多播。PIM-SM 核心路由器称为会合点（rendezvous point，RP）。多播通信通过两步完成。任一路由器，如果它有一个要发往一组目的站的多播分组，那么它首先将多播分组封装在单播分组中（隧道），并且将其发送给 RP。之后 RP 剥开单播分组并发送多播分组到它的目的站。

PIM-SM 使用一个复杂的算法从互联网中选择一个路由器作为特定组的 RP。尽管路由器可能服务多个组，但是这意味着如果我们有 m 个活跃组，那么就需要 m 个 RP。在每个组的 RP 选择好后，每个路由器创建一个数据库并存储组标识符以及 RP 的 IP 地址，用于将多播分组通过隧道方式传送给它。

PIM-SM 使用生成多播树，树的根结点为 RP，叶子结点为指定路由器，这些路由器连接到每个拥有活跃成员的网络上。PIM-SM 中的重点是组的多播树的形成。其中的思想是，每个路由器帮助创建树。路由器应该知晓一个单一端口，即从那个端口可以接收发往某个组的多播分组（通过 DVMRP 中的 RPF 实现）。路由器也应该知晓一个或多个端口，即从那些端口发送去往某个组的多播分组（通过 DVMRP 中的 RPM 实现）。为了避免通过多个路由器向网络传送多个相同的分组副本（通过 DVMRP 中的 RPB 实现），正如我们将很快看到的，PIM-SM 要求指定路由器只发送 PIM-SM 报文。

为了创建根结点为 RP 的多播树，PIM-SM 使用嫁接（join）和修剪（prune）报文。图 4-100 给出 PIM-SM 中嫁接和修剪报文的操作。首先，三个网络嫁接到经 G1 上并形成多播树。之后，一个网络离开组并且树被修剪。

嫁接报文用来将可能的新分支加入树；修剪报文用来修剪不需要的分支。当指定路由器（通过 IGMP）发现网络在对应组中有一个新成员，它向 RP 发送一个嫁接报文，这个报文在单播分组中。路径上任何路由器将接收并转发这个报文，但同时，路由器在自身多播转发表中加入两条信息。嫁接报文到达的那个接口的接口号被标记下来（如果没有被标记），未来去往组的多播分组应该从这个接口发送出去。正如我们马上讨论的，路由器也加入一个计数器来计算已经被接收的嫁接报文。被发送到 RP 的嫁接报文穿过某个接口，这个接口号被标记下来（如果没有被标记），发往相同组的多播分组应该从这个接口被发送出去。按照这种方式，第一个由指定路由器发送的嫁接分组创建了一条从 RP 到带有组成员的网络的路径。



a. 一个网络加入组G1　　b. 嫁接后的多播树

c. 一个网络离开组G1　　d. 修剪后的多播树

图 4-100　PIM-SM 背后的思想

为了避免向没有成员的网络发送多播分组，PIM-SM 使用修剪报文。每个（通过 IGMP）发现网络中没有活跃成员的指定路由器，都向 RP 发送修剪报文。当路由器接收到修剪报文时，它减少报文到达的接口的嫁接计数器，并将其转发到下一个路由器。当一个接口的嫁接计数器到达 0，接口不再是多播树的一部分。

4.4.4　域间路由选择协议

我们讨论的三种多播路由选择协议：DVMRP、MOSPF 以及 PIM 都用来为自治系统内部提供多播通信。当组的成员分布在不同域（AS）时，我们需要域间多播路由选择协议。

一个常见的域间路由选择协议称为多播边界网关协议（Multicast Border Gateway Protocol，MBGP），它是 BGP 协议的扩展，BGP 是我们讨论的域间单播协议。MBGP 提供 AS 间的两条路径：一条用来单播，一条用来多播。多播信息在不同 AS 的边界路由器之间交换。MBGP 是组共享多播路由选择协议，在每个 AS 中选定一个路由器作为会合点（RP）。

MBGP 协议的问题是它很难向 RP 通知另一个 AS 中的组的源端。多播源发现协议（Multicast Source Discovery Protocol，MSDP）是新引入的协议，它在每个 AS 中指定一个源端代表路由器，以此来通知所有 RP 在那个 AS 中存在源端。

另一个被认为是 MGBP 替代品的协议是边界网关多播协议（Border Gateway Multicast Protocol，BGMP），它允许在一个 AS 中建立带有一个根的组共享树。换言之，对于每个组，只有一个组共享树，叶结点在其他 AS 中，但是根结点位于其中一个 AS 中。当然，这里存在两个问题，即如何将一个 AS 指定为树的根，以及如何向所有源端通知根结点位置，所有源端将多播分组通过隧道发送到根结点。

4.5　下一代 IP

在本章的最后一节我们讨论新一代 IP，IPv6，IPv4 地址耗尽以及这个协议的其他缺点在 20 世

纪90年代诞生了IP协议新版本。新版本称为因特网协议第六版（Internet Protocol version 6，IPv6）或者新一代IP（IP new generation，IPng），它在增加了IPv4地址空间的同时重新设计了IP分组的格式并修改了一些辅助协议，例如ICMP。有趣的是，IPv5曾是一个提议，它基于OSI模型，但是没有成为现实。以下给出了IPv6协议的主要变化：

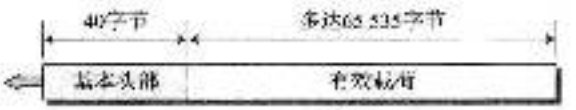
- 更大的地址空间。IPv6地址是128位长。与32位长的IPv4地址相比，其地址空间增加了很多（2^{96}倍）。
- 更好的头部格式。IPv6使用了新的头部格式，其选项与基本头部分开，如果需要，可将选项插入到基本头部与上层数据之间。这就简化和加速了路由选择过程，因为大多数选项不需要由路由器检查。
- 新的选项。IPv6有一些新的选项来实现附加的功能。
- 允许扩展。如果新的技术或应用需要的话，IPv6允许协议进行扩展。
- 支持资源分配。在IPv6中，服务类型字段被取消了，但增加了一种机制（称为流标号）使得源端可以请求对分组进行特殊的处理。这种机制可用来支持像实时音频和视频的通信量。
- 支持更多的安全性。在IPv6中的加密和鉴别选项提供了分组的保密性和完整性。

IPv6的发展势头已经减缓。原因在于它的发展最初的动机是IPv4地址耗尽，地址耗尽问题已经被短期策略缓解了：无类寻址、为动态地址分配使用DHCP和NAT。然而，因特网快速发展和新的服务的出现，如移动IP、IP电话和IP移动电话最终要求用IPv6全部替代IPv4。过去预期世界上的所有主机将在2010年使用IPv6，但是这个事情没有发生。最近的预期结果是2020年。

本节，我们首先讨论IPv6分组格式。之后讨论IPv6寻址，它遵循与IPv4不同的架构。之后我们给出，如何使用一些非常策略来完成从旧版本转换到新版本的艰巨任务。最终，我们讨论ICMPv6，网络层唯一的辅助协议，它替代了IPv4中的很多协议。

4.5.1 分组格式

IPv6分组格式如图4-101所示。每个分组由基本头部和紧跟其后的有效载荷组成。基本头部占40字节，但是有效载荷可以包含多达65 535字节信息。字段描述如下。



a. IPv6分组



b. 基本头部

图4-101 IPv6数据报

- 版本（version）。4位字段定义了IP版本号。对于IPv6，其值为6。
- 流量类（traffic class）。8位流量类字段用来区分不同传递要求的不同有效载荷。它代替了IPv4中的服务类型字段。
- 流标号（flow label）。流标号是一个20字节的字段，它用来对特殊的数据流提供专门处理。我们将在稍后讨论这个字段。

- **有效载荷长度**（payload length）。这个2字节的有效载荷长度字段定义了不包括基本头部的IP数据报的总长度。注意，IPv4定义两个与长度相关的字段：报头长度以及总长度。在IPv6中，基本头部的长度是固定的（40字节）；因此只有有效载荷的长度需要被定义。
- **下一个头部**（next header）。下一个头部是一个8位字段，它定义了第一个扩展头部的类型（如果存在）或是数据报中跟随在基本头部之后的头部。这个字段与IPv4中的协议字段类似，但是我们将在讨论有效载荷时进行更深入的讨论。
- **跳数限制**（hop limit）。这个8位的跳数限制字段与IPv4中的TTL字段所起的作用是一样的。
- **源地址和目的地址**（source and destination address）。源地址字段是16字节（128位）的因特网地址，它用来识别数据报的原始源端。目的地址字段是16字节（128位）的因特网地址，它用来识别数据报的目的端。
- **有效载荷**（payload）。与IPv4相比，IPv6中的有效载荷字段有不同的格式和含义，如图4-102所示。

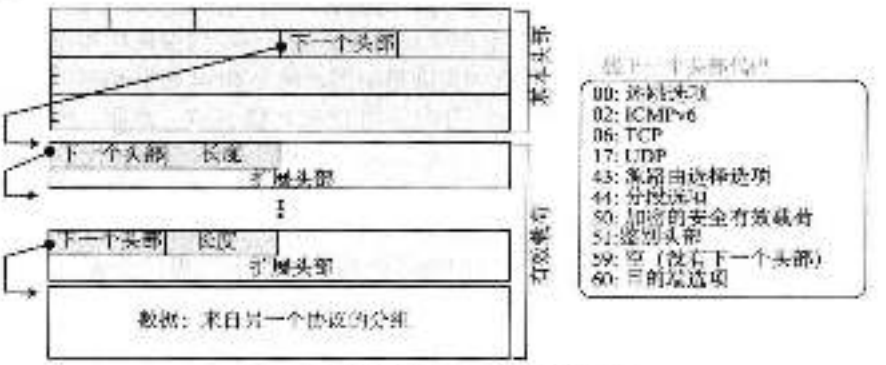


图4-102 IPv6数据报中的有效载荷

IPv6中的有效载荷意味着0个或多个扩展头部（选项）的组合，紧跟其后的是来自其他协议（UDP、TCP等）的数据。在IPv6中，IPv4中部分头部的选项被设计为扩展头部。有效载荷可以根据情况有多个扩展头部。每个扩展头部有两个强制字段，下一个头部和长度，紧跟其后的是与特定选项相关的信息。注意，每个下一个头部字段值（代码）定义了下一个头部的类型（逐跳选项，源路由选择选项……）；最后的下一个头部字段定义了协议（UDP、TCP……），它由数据报携带。

IPv6中流和优先级的概念

IP协议最初被设计成无连接协议。然而，现今的趋势是将IP协议用作面向连接协议。之前描述的MPLS技术允许我们将IPv4分组封装到一个使用标签字段的MPLS头部中。在版本6中，流标号被直接加入到IPv6数据报中，允许我们将IPv6用作面向连接协议。

对于路由器，流是共享某些特性的分组序列，例如，经过相同路径，使用相同的资源，具有相同安全性等。支持流标号处理的路由器有一个流标号表，这个表为每一个活动的流标号设置一个项目，每一个项目定义相应流标号所需的服务。当路由器收到一个分组时，它就从其流标号表中找出在分组中的流标号值所对应的项目。但请注意，流标号本身并不给流标号表项提供信息，信息是由其他方法提供的，如逐跳选项或其他协议。

在最简单的形式中，流标号可用来加速路由器对分组的处理。当路由器接收到一个分组时，它不需要查找路由表也不需要通过路由选择算法确定下一跳地址，取而代之的是可以很容易地在流标号表中找到下一跳的地址。

在更复杂的形式中，流标号可用来支持实时音频和视频的传输。特别是数字形式的实时音频或

视频，它们需要高带宽、大缓存、长处理时间等资源。进程可以事先对这些资源进行预留，以保证实时数据不会因资源不够而被延迟。使用实时数据和预留资源除了 IPv6 需要一些其他协议，如实时传输协议（RTP）以及资源预留协议（RSVP）（见第 8 章）。

分段和重组

IPv6 协议中仍然需要分段和重组，但是在这方面存在很大不同。IPv6 数据报仅仅在源端才分段，而不是在路由器；重组发生在目的端。不允许在路由器对分组进行分段，以此来提高路由器中分组的处理速度。路由器中分组的分段需要很多处理。分组需要被分段，与分段相关的所有字段需要被重新计算。在 IPv6 中，源可以检查分组的大小，并决定分组是否被分段。当路由器接收分组时，它可以检查分组的大小，如果大于前方网络允许的 MTU 则丢弃它。之后，路由器发送分组过长 ICMPv6 错误报文（稍后讨论）来通知源端。

扩展头部

扩展头部在 IPv6 中是一个必要部分，它起到重要作用。尤其是三个扩展头部——分段、鉴别以及扩展的加密安全有效载荷——存在于一些分组中。为了减少章节的长度，我们不讨论扩展头部，但是作为第 4 章的扩展资料它包含在本书的网站上。

4.5.2 IPv6 寻址

从 IPv4 迁移到 IPv6 的主要原因是 IPv4 地址空间小。在本节，给出 IPv6 巨大的地址空间如何防止未来的地址耗尽。我们也讨论新地址如何解决 IPv4 寻址机制中的问题。IPv6 地址是 128 位或 16 字节（8 位字节）长，是 IPv4 地址长度的四倍。

一台计算机通常按二进制存储地址，但是很明显，普通人无法轻易处理 128 位。当人们处理 IPv6 地址时，提出了一些标记法以代表 IPv6 地址。以下给出这些标记法中的两个：二进制和冒号十六进制。

```
二进制          1111111011110110  ...  1111111100000000
冒号十六进制    FEF6:BA98:7654:3210:ADEF:BBFF:2922:FF00
```

当地址被存储在计算机中，使用的是二进制标记法。冒号十六进制标记法（colon hexadecimal notation）（或简称 colon hex）将地址分为 8 个部分，每个部分由 4 个十六进制数字组成，每 4 个数字用一个冒号分隔开。

即使使用十六进制格式，IPv6 地址也非常长，但是很多数字是 0。在这种情况下我们可以将地址缩短。地址某部分中开始的一些 0 可以省略。使用这种缩短（abbreviation）形式，0074 可以写成 74，000F 可以写成 F，而 0000 可以写成 0。注意，3210 就不能缩短。进一步的缩短，通常称为 0 压缩。如果有连续的部分仅仅包含 0，则可以使用 0 压缩。我们可以将所有的 0 移除，而用两个冒号来代替 0。

FDEC:0:0:0:0:BBFF:0:FFFF → FDEC::BBFF:0:FFFF

注意，这种缩短方法对一个地址只能使用一次。如果有多串 0 的部分，只能有其中的一部分进行缩短。

有时我们会到 IPv6 地址的混合表示法：冒号十六进制与点分十进制表示法的结合。在过渡阶段这个方法是合适的，在这个阶段 IPv4 地址被嵌入 IPv6 地址中（作为最右边的 32 位）。我们可以对最左边的六个部分使用冒号十六进制表示法以及四字节点分十进制表示法，而不是对最右边的两个部分使用这些表示法。然而，当 IPv6 地址最左侧个部或绝大多数为 0 时才可以这样做。例如地址（**::130.24.24.18**）是 IPv6 中的合法地址，其中 0 压缩方法让我们看到左侧 96 个位是 0。

正如我们马上看到的，IPv6 使用层次寻址。因此，IPv6 允许斜杠或 CIDR 表示法。例如，以下给出我们如何使用 CIDR 定义前缀 60 位。我们将稍后给出 IPv6 地址如何分为前缀和后缀。

FDEC::BBFF:0:FFFF/60

地址空间

IPv6 有 2^{128} 个地址，这个地址空间是 IPv4 地址的 2^{96} 倍数——肯定没有地址耗尽——如下所示，空间大小是

340 282 366 920 938 463 374 607 431 768 211 456

为了理解这个数字，我们假设只有 1/64（几乎 2%）的地址被分配给地球上的人，其余地址留作特殊目的。我们也假设地球的人口很快到达 2^{34}（超过 160 亿）。每个人可以有 2^{88} 个地址。这个版本的地址耗尽是不可能的。

三种地址类型

在 IPv6 中，目的地址可以属于以下三种中的一种：单播、任播以及多播。单播地址定义了一个接口（计算机或路由器）。被发送到单播地址的分组将被路由到计划中的接收者那里。

任播地址定义了一组计算机，这组计算机共享一个地址。带有任播地址的分组只被传递到组中的一个成员，即那个最容易到达的成员。例如，当很多服务器响应一个查询时，就使用任播通信。请求被发送到最容易到达的服务器上。硬件和软件只产生请求的一个副本；副本只到达路由器中的一个。IPv6 不为任播指派块；地址被从单播块中分配。

多播地址也定义了一组计算机。然而，任播和多播有所不同。在任播中，分组只有一个副本被发送给组的成员；在多播中每个组的成员都接收一个副本。我们将马上看到，IPv6 为多播指派一个块。从这个块中为组成员分配相同的地址。

有趣的是，IPv6 没有定义广播，即使是限制版本的广播也没有定义。我们将看到 IPv6 将广播看做多播的特殊情况。

地址空间分配

像 IPv4 地址空间一样，IPv6 地址空间被分成若干不同大小的块，每个块分别起到不同的作用。绝大多数块是未指派的，它们并被留出以待将来使用。表 4-7 只给出指派了的块。在这个表中，最后一列给出每个块在整个地址空间所占的份额。

表 4-7 已被分配的 IPv6 地址的前缀

块 前 缀	CIDR	从 任 务	份 额
0000 0000	0000::/8	特殊地址	1/256
001	2000::/3	全局单播	1/8
1111 110	FC00::/7	唯一本地单播	1/128
1111 1110 10	FE80::/10	链路本地地址	1/1024
1111 1111	FF00::/8	多播地址	1/256

全局单播地址

地址空间中用来进行互联网中两个主机之间单播（一对一）通信的块称为全局单播地址块。这个块的 CIDR 是 2000::/3，这意味着，这个块中所有地址的最左边三位是相同的（001）。这个块的大小是 2^{125} 位，可以支撑因特网未来几年的扩展。这个地址被分为三部分：全局路由选择前缀（n 位）、子网标识符（m 位）以及接口标识符（q 位）。如图 4-103 所示。图中也给出了每个部分的推荐长度。

全局路由选择前缀用来将分组从因特网路由到组织机构站点，例如拥有块的 ISP。由于这个部分的前三位是固定的（001），剩余 45 位可以定义多达 2^{45} 个站点（私有组织机构或 ISP）。因特网中全局路由器基于 n 的值将分组路由到目的站点。接下来 m 位（推荐 16 位）定义组织机构中的子网。这意味着，一个组织机构可以有多达 $2^{16}-65\,536$ 个子网，这个数量足够了。

最后 q 位（推荐 64 位）定义了接口标识符。接口标识符与 IPv4 寻址中 hostid 类似，但是前者

是一个更好的选择，因为我们之前讨论过，主机标识符实际上定义了接口而不是主机。如果主机从一个接口移动到另一个接口，它的 IP 地址需要改动。

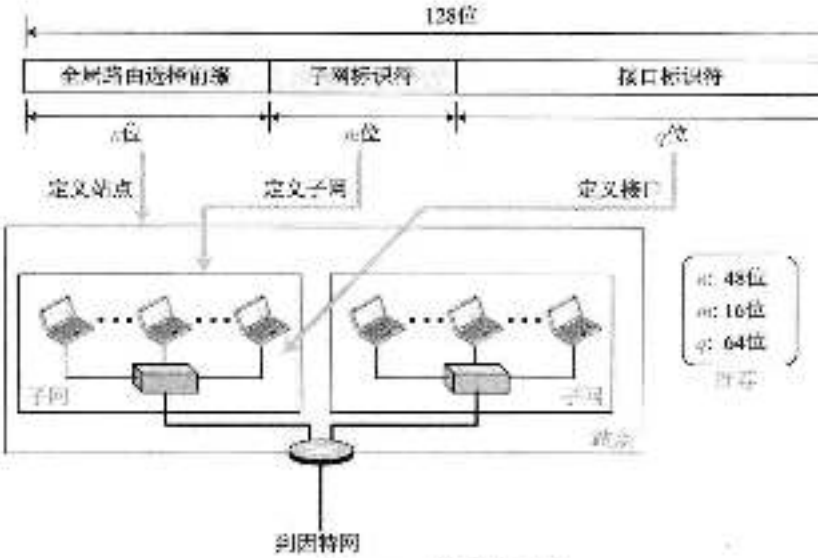


图 4-103 全局单播地址

在 IPv4 寻址中，hostid（在 IP 层）和链路层地址（在数据链路层）之间没有特定关系，因为链路层地址通常更长。IPv6 寻址允许这两者存在一定关系。链路层地址的长度小于 64 位，它可以作为接口标识符或作为其一部分，这样就排除了映射过程。有两种常见的链路层寻址方案以实现这个目的：IEEE 定义的 64 位扩展唯一标识符（64-bit extendedunique identifier，EUI-64）以及以太网定义的 48 位链路层地址。

特殊地址

在讨论全局单播块之后，让我们来讨论表 4-7 中第一列已被指定和保留的块的特点及其作用。使用前缀（**0000::/8**）的地址被保留，但是这个块的一部分用于定义某些特殊地址。图 4-104 给出这个块中已指定的地址。

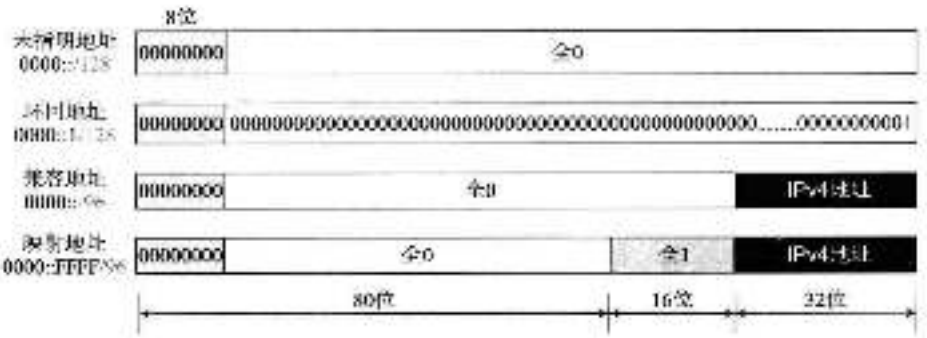


图 4-104 特殊地址

未指明地址（unspecified address）是包含一个地址的子块，在启动时使用，那时主机不知道自己的地址并想发送查询找出其地址（见 DHCP 节）。

环回地址（loopback address）也只包含一个地址。我们之前讨论过 IPv4 的环回地址。在 IPv4

中，块由一系列地址组成；在 IPv6 中，块仅仅含有一个地址。

正如我们之后将要看到的，从 IPv4 到 IPv6 的过渡中，主机可以使用嵌入 IPv6 地址中的 IPv4 地址。为这个目的，设计出了两种格式：兼容地址以及映射地址。兼容地址（compatible address）是含有 96 位 0 后面紧跟 32 位 IP 地址的一种地址。当使用 IPv6 的计算机想要发送报文到另一个 IPv6 计算机时使用这种格式。当一台已经过渡到 IPv6 的计算机想要发送报文到仍然使用 IPv4 的计算机时，它使用映射地址（mapped address）。关于映射地址和兼容地址很有趣的一点是，当计算校验和时，它们可以使用嵌入地址或整个地址，因为 16 位数个 0 或 1 对于校验和计算没有影响。这对使用伪头部来计算校验和的 UDP 和 TCP 非常重要，因为如果路由器将分组的地址从 IPv6 转到 IPv4，校验和不会受影响。

其他特定块

IPv6 将两个大块用于私有寻址，一个大块用于多播，如图 4-105 所示。唯一本地单播块（unique local unicast block）的子块可以是私有创建的并被一个站点使用。携带这类地址作为目的地址的分组不应该被路由。这类地址有标识符 1111 1110，下一位可能是 0 或 1，它用来定义地址如何被选择（本地或被权威机构选择）。接下来 40 位被站点用随机数方式选定。这意味着一共有 48 位定义了子块，它看起来像全局单播地址。40 位随机数字使得地址重复的概率极其小。请注意这些地址与全局单播地址的类似之处。第二个块被指派用于私有地址，是本地链路块（link local block）。这个块中的子块可以用于网络的私有地址。这类地址有块标识符 1111111010。接下来 54 为被设置为 0。最后 64 位可以被改变，用于为每个计算机定义接口。注意这些地址格式与全局单播地址的相似之处。

我们在本章早些时候讨论过 IPv4 的多播地址。多播地址用于定义一组主机而不是一个主机。在 IPv6 中一大块地址用于多播。所有这些地址使用前缀 11111111。第二个字段是标记，它将组地址定义成永久或临时的。永久组地址由因特网权威机构定义，并可以随时访问。另一方面，临时组地址仅仅临时使用。例如参加电话会议的系统可以使用临时组地址。第三个字段定义了组地址的范围。如图 4-105 所示，很多不同范围已经被定义。

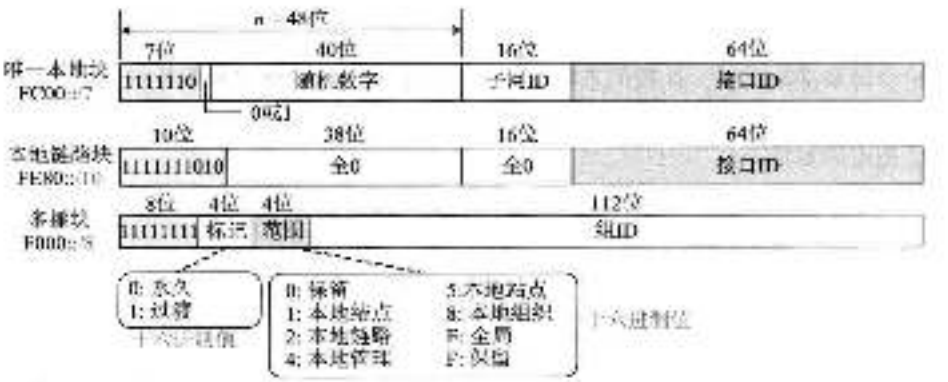


图 4-105　唯一本地单播块

4.5.3　从 IPv4 到 IPv6 的过渡

尽管我们已经有了新版本的 IP 协议，但是我们如何停止使用 IPv4 而开始使用 IPv6 呢？我们想到的第一个解决方法是定义过渡日期，从那一天起，每个主机或路由器应该停止使用旧版开始使用新版。然而，这不现实；因为因特网中系统的数量巨大。从 IPv4 到 IPv6 的过渡不可能突然发生。需要花费相当长的时间。过渡必须平缓，以防止在 IPv4 和 IPv6 系统之间出现问题。IETF 已经提出三种策略来帮助过渡：双协议栈、隧道以及头部转换。

双协议栈

IETF 推荐所有的主机在完全过渡到第六版之前，使用一个双协议栈（dual stack）。换言之，一个站应同时运行 IPv4 和 IPv6，直到整个因特网使用 IPv6。图 4-106 给出了双协议栈的配置示意图。

当把分组发送到目的地时，为了确定使用哪个版本，主机要向 DNS 进行查询。如果 DNS 返回一个 IPv4 地址，那么源主机就发送一个 IPv4 分组；如果返回一个 IPv6 地址，就发送一个 IPv6 分组。



图 4-106 双协议栈

隧道技术

当两台使用 IPv6 的计算机要进行通信，但其分组要通过使用 IPv4 的区域时，就要使用隧道技术（tunneling）这种策略。因此，当进入这种区域时，IPv6 分组要封装成 IPv4 分组，而当分组离开该区域时，再去掉这个封装。这就好像 IPv6 分组进入隧道一般，而在另一端流出来。为了更清楚地说明利用 IPv4 分组捎带 IPv6 分组，其协议的值设为 41。隧道技术如图 4-107 所示。

头部转换

当因特网中绝大部分已经过渡到 IPv6，但一些系统仍然使用 IPv4 时，就需要使用头部转换（header translation）。发送方想使用 IPv6，但接收方不能识别 IPv6。这种情况下使用隧道技术无法工作，因为分组必须是 IPv4 的格式才能被接收方识别。在此情况下，头部格式必须通过头部转换而彻底改变，IPv6 的头部就转变成 IPv4 的头部（见图 4-108）。



图 4-107 隧道策略



图 4-108 头部转换策略

4.5.4 ICMPv6

另一个在 TCP/IP 协议簇第六版中被修改的协议是 ICMP。这是一个新版本，因特网控制报文协议第六版（Internet Control Message Protocol version 6，ICMPv6）。它与第四版具有相同的策略和目的。然而，ICMPv6 比 ICMPv4 更复杂：在版本 4 中的一些独立的协议现在成为 ICMPv6 的一部分，并且加入了一些新的报文来使其更实用。图 4-109 比较了版本 4 和版本 6 的网络层。ICMP、

ARP（第 5 章讨论）以及 IGMP 协议第 4 版被合并到一个协议即 ICMPv6 中。

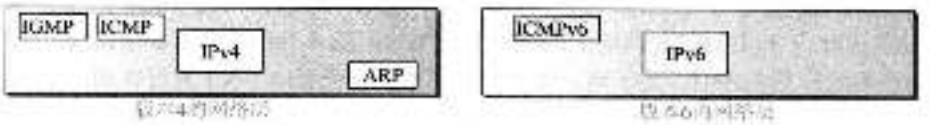


图 4-109 版本 4 和版本 6 的网络层比较

我们可以将 ICMPv6 中的报文分为四组：差错报告报文、信息报文、邻居发现报文以及组成员关系报文。

差错报告报文采用了与版本 4 中相同的策略。这组中有四种报文类型：目的端不可达（destination-unreachable）、分组过大（packet-too-big）、时间超时（time-exceeded）以及参数问题（parameter-problem）。版本 4 中的源端抑制报文被取消，因为它不常用。重定向报文被移动到另一组。一种新的报文即分组过大报文被加入，因为在版本 6 中路由器不允许分段。如果报文过大，路由器将会丢弃它并发送一个分组过大错误报文给源端（见图 4-110）。



图 4-110 ICMPv6 报文

版本 6 中的信息报文与版本 4 中的请求报文作用相同。在这组中只有一对报文：回送请求（echo-request）和回送应答（echo-reply）报文。将这组信息报文与第 4 版进行比较，我们看到两个报文（即时间戳请求和时间戳回答）已经被取消，因为它们很少用到。

一种称为邻居发现报文（neighbor-discovery message）的新组被加入到第 6 版。这组报文可以找到邻居中的路由器，找到邻居中其他主机的链路层地址。换言之，这个组可以被分为四个子组。

路由器请求（router-solicitation）以及路由器通告（router-advertisement）报文可以帮助主机找到默认路由器，主机将分组转发到这个路由器。第4版中DHCP协议负责这个任务。当给出IPv6地址，邻居请求（neighbor-solicitation）和邻居通告（neighbor-advertisement）报文试图发现主机或路由器的链路层地址，正如我们在第5章讨论的，这个任务被第4版的地址解析协议（ARP）完成。重定向（redirection）报文与第4版完成的工作相同。当给出链路层地址，逆向邻居请求（inverse-neigh-bor-solicitation）以及逆向邻居通告（inverse-neigh-bor-advertisement）报文用来找出主机或路由器的IPv6地址。

另一个新组，组成员关系（group-membership）报文，拥有两种报文：成员关系请求（membership-query）报文以及成员关系报告（membership-report）报文，它们用于多播。它们完成本章早些时候讨论的IGMP报文所完成的工作。

更多于ICMPv6相关讨论可以在本书网站的第4章额外资料中找到。

4.6 章末资料

推荐读物

书籍

此处列出一些涵盖本章内容的书籍，我们推荐[Com 06]、[Tan 03]、[Koz 05]、[Ste 95]、[G & W 04]、[Per 00]、[Kes 02]、[Moy 98]、[W & Z 01]以及[Los 04]。

RFC

IPv4寻址在RFC 917、927、930、932、940、950、1122和1519中讨论到。转发在RFC 1812、1971和1980中讨论。MPLS在RFC 3031、3032、3036以及3212中讨论。IPv4协议在RFC 791、815、894、1122、2474以及2475中讨论。ICMP在RFC 792、950、956、957、1016、1122、1256、1305以及1987中讨论。RIP在RFC 1058和2453中讨论。OSPF在RFC 1583和2328中讨论。BGP在RFC 1654、1771、1773、1997、2439、2918以及3392中讨论。多播在RFC 1075、1585、2189、2362以及3376中讨论。IPv6寻址在RFC 2375、2526、3513、3587、3789以及4291中讨论。IPv6协议在RFC2460、2461和2462中讨论。ICMPv6在RFC 2461、2894、3122、3810、4443以及4620中讨论。

小结

因特网由很多网络（或链路）组成。它通过连接设备相互连接，每个连接设备都起到路由器或交换机的作用。传统上在网络中使用两种交换：电路交换和分组交换。网络层被设计成分组交换网络。

网络层通过底层物理网络监管分组的处理。分组的传递可以是点到点的也可以是间接的。这里定义了两类转发：基于IP数据报目的地址的转发以及基于IP数据报附加标签的转发。

IPv4是一个不可靠无连接协议，它负责源端到目的端传递。IP层中的分组称为数据报。TCP/IP协议簇的IP层所使用的标识符称为IP地址。一个IPv4地址32位长，分为两部分：前缀和后缀。块中所有地址都有相同的前缀；每个地址有一个不同的后缀。

因特网控制报文协议（ICMP）支持不可靠无连接因特网协议（IP）。

为了能够路由分组，路由器需要转发表。路由选择协议是特定的应用程序，它负责更新转发表。

多播是将相同的报文同时发送给一个以上接收者。收集本地成员关系组信息涉及了因特网组管理协议（IGMP）。

IPv6，因特网协议的最新版本，有 128 位地址空间。IPv6 使用十六进制冒号表示法，可以使用缩短形式。

4.7 习题集

测试题

本章的交互测验题集可以在本书的网站上找到。强烈建议学生们做这些测验题，这样可以在学生做习题集前实检测他/她对课程资料的理解。

练习题

Q4-1 为什么网络层协议需要向传输层提供分组服务？为什么不把段封装到数据中传输层就不能直接发送段？

Q4-2 为什么网络层的责任是路由选择？换言之，为什么不能在传输层或数据链路层完成路由选择？

Q4-3 区分这两个过程：将分组从源端路由到目的端以及在每个路由器处转发分组。

Q4-4 在以下交换方法中，转发决定是基于分组中哪条信息做出的？

a. 数据报方法　　b. 虚电路方法

Q4-5 如果面向连接服务的标签是 8 位，同时可以建立多少条虚电路？

Q4-6 请列出虚电路交换方法中的三个阶段。

Q4-7 TCP/IP 网络层有以下哪个服务？如果没有，为什么？

a. 流量控制　　b. 差错控制　　c. 拥塞控制

Q4-8 列出分组交换网络的四种延迟。

Q4-9 在图 4-10 中，假设 R1 和 R2 之间的链路升级到 170kbps，源主机和 R1 之间的链路下降到 140kbps。改变之后的源端和目的端之间的吞吐量是多少？现在瓶颈是哪段路的链路？

Q4-10 IPv4 分组中的头部长度字段可以小于 5 吗？什么时候正好等于 5？

Q4-11 一台主机正在向另一台发送 100 个数据报。如果第一个数据报的标识号是 1024，最后一个是多少？

Q4-12 一个偏移值为 100 的 IP 分段到达。在这个分段的数据之前，源端发送多少字节的数据？

Q4-13 在无类寻址中，我们知道块中的第一个和最后一个地址。我们可以找到前缀长度吗？如果回答是肯定的，请给出过程。

Q4-14 在无类寻址中，我们知道块中的第一个和地址数。我们可以找到前缀长度吗？如果回答是肯定的，请给出过程。

Q4-15 在无类寻址中，两个不同的块可以有相同的前缀长度吗？请解释。

Q4-16 在无类寻址中，我们知道块中的第一个和块中的另一个地址（不一定是最后一个）。我们可以找到前缀长度吗？请解释。

Q4-17 一个 ISP 有一个拥有 1024 个地址的块。它需要将地址分给 1024 客户。它需要子网吗？解释你的答案。

Q4-18 说出三个 TCP/IP 协议簇网络层中为帮助 IPv4 协议而设计的辅助协议。

Q4-19 在 IPv4 数据报中，头部长度（HLEN）字段的值是$(6)_{16}$。可以在分组中加入多少字节的选项？

Q4-20 数据报中 TTL 值可以是以下哪种情况？解释你的答案。

a. 23　　b. 0　　c. 1　　d. 301

Q4-21 将网络层协议字段和传输层端口号进行对比。它们的共同目的是什么？为什么我们需要两个端口号字段但只需要一个协议字段？为什么协议字段的大小只是每个端口号大小的一半？

Q4-22 数据报中哪些字段负责将所有分段组合成原始数据报？

Q4-23 假设目的端计算机从源端接收到几个分组。它如何确定属于某个数据报的分段没有混入属于其他数据报的分段？

Q4-24 请解释为什么使用 MPLS 意味着向 TCP/IP 协议中加入一个新层？这个层在哪里？

Q4-25 请解释为什么因特网不创建一个报告报文，用来汇报携带了 ICMP 报文的 IP 数据报中的差错？

Q4-26 在携带由路由器报告的 ICMP 报文的数据报中，源端和目的端 IP 地址是多少？

Q4-27 在图中，如果我们知道从结点 A 到结点 G 的最短路径是 {A → B → E → G}，从 G 到 A 的最短路径是什么？

Q4-28 假设图中从结点 A 到结点 H 的最短路径是 A → B→ H。假设从结点 H 到结点 N 的最短路径是 H → G → N。从结点 A 到结点 N 的最短路径是什么？

Q4-29 请解释为什么使用链路状态路由选择的路由器需要在创建并使用转发表前接收整个 LSDB。换言之，路由器为什么不能使用接收到的部分 LSDB 创建转发表？

Q4-30 路径向量路由选择算法与以下哪种算法更接近：距离向量路由选择算法还是链路状态路由选择算法？请解释。

Q4-31 列出文中描述的三种类型的自治系统（AS），并做一比较。

Q4-32 解释 RIP 中跳数的概念。你能解释为什么图 4-70 中 N1 和 R1 之间没有跳数吗？

Q4-33 假设我们有运行着 RIP 的独立 AS。我们可以说在这个 AS 中至少有两种不同的数据报流量。第一种携带两个主机之间交换的分组；第二种携带基于 RIP 的报文。当我们考虑源和目的 IP 地址时，这两种流量有不同是什么？这是不是表明路由器也需要 IP 地址？

Q4-34 路由器 A 发送两个 RIP 报文到两个近邻路由器，B 和 C。这两个数据报携带相同的源 IP 地址吗？两个数据报有相同的目的 IP 地址吗？

Q4-35 任何时候，RIP 报文都可到达运行着 RIP 路由选择协议的路由器。这是否意味着 RIP 进程应该不停地运行？

Q4-36 你知口 RIP 为什么使用 UDP 而不使用 TCP 吗？

Q4-37 我们说 OSPF 是一个分层的域内协议，但是 RIP 不是。这样说的原因是什么？

Q4-38 在一个很小的使用 OSPF 的 AS 中，使用一个区域（主干）高效还是使用多个区域高效？

Q4-39 为什么只需要一种 RIP 更新报文而需要多种 OSPF 更新报文？

Q4-40 OSPF 报文在路由器之间交换。这是否意味着我们需要 OSPF 进程不断运行才能在报文到达时接收到它？

Q4-41 OSPF 报文以及 ICMP 报文被直接封装在 IP 数据报中。如果我们捕获一个 IP 数据报，我们如何区分负载是属于 OSPF 还是 ICMP？

Q4-42 请解释如下情况通知哪种 OSPF 链路状态类型？

a. 路由器需要通告点对点连接的末尾存在一个路由器。

b. 路由器需要通告两个或更多网络和一个过渡网络的存在。

c. 被指派路由器将网络通告为一个结点。

Q4-43 路由器能够在一个链路状态更新中包含链路和网络的通告吗？

Q4-44 请解释为什么不同的 AS 有不同的域内路由选择协议，但是整个因特网中只有一种域间路由选择协议？

Q4-45 你能解释为什么 BGP 使用 TCP 服务而不使用 UDP 吗？

Q4-46 请解释为什么策略路由选择可以在域间路由选择中实现，但是不能在域内路由选择中实现？

Q4-47 请解释以下属性何时在 BGP 中使用：

a. LOCAL-PREF　　b. AS-PATH　　c. NEXT-HOP

Q4-48 区分多播和多个单播。

Q4-49 当我们向多个收件人发送电子邮件时，我们使用多播还是多个单播？给出原因。

Q4-50 以下哪个地址是多播地址：

a. 224.8.70.14　　b. 226.17.3.53　　c. 240.3.6.25

Q4-51 以下多播地址属于哪个组？（本地网络控制块、互联网控制块、SSM 块、团块或管理范围块）：

a. 224.0.1.7　　b. 232.7.14.8　　c. 239.14.10.12

Q4-52 一个主机可以有一个以上多播地址吗？请解释。

Q4-53 一个多播路由器连接到 4 个网络上。每个网络的兴趣如下所示，路由器应该通告的组列表是什么？

a. N1: {G1, G2, G3}　　b. N2: {G1, G3}　　c. N3: {G1, G4}　　d. N4: {G1, G3}

Q4-54 假设我们需要为单播和多播提供生成树。每种情况下有多少树的叶结点参与到传输中？

a. 单播传输　　b. 多播传输

Q4-55 假设在小型 AS 中有 20 个主机。这个 AS 中只有 4 个组。找出以下方式中的生成树数量。

a. 基于源树　　b. 组共享树

Q4-56 在 DVMRP 中我们说路由器按需创建最短路径树。这是什么意思？按需创建的优势在哪里？

Q4-57 列出 DVMRP 路由器创建基于源树的三个步骤。哪个步骤负责创建从源到当前路由器的树？哪个步骤负责创建路由器为根的广播树？哪个步骤负责将广播树改变为多播树？

Q4-58 请解释为什么 MOSPF 路由器可以用一步创建源为根结点的最短路径树，但是 DVMRP 需要二步才能完成。

Q4-59 解释为什么 PIM 称为协议独立多播（protocol independent multicast）。

Q4-60 第个版本的 PIM 使用 DVMRP 的第一步和第三步？这两步是什么？

Q4-61 请解释为什么在 PIM-DM 中广播第一个或头几个报文不重要，而在 PIM-SM 中就重要。

Q4-62 请解释与 IPv4 相比 IPv6 的优势。

Q4-63 请解释 IPv6 中流字段的作用。这个字段的潜在应用是什么？

Q4-64 区分兼容地址和映射地址，并解释它们的应用。

Q4-65 列出 IPv4 网络层的三个协议，它们在 IPv6 中组合为一个协议。

Q4-66 差错报告 ICMP 报文将 IP 头部和数据报数据前 8 个字节包含进去的目的是什么？

思考题

P4-1 在 IPv4 数据报中，总长度字段的值是$(00A0)_{16}$，并且头部长度（HLEN）的值是$(5)_{16}$。数据报携带的有效负载是多少字节？这个数据报的效率是多少（有效负载长度与总长度的比）？

P4-2 一个 IP 数据报到达。它的头部含有如下信息（十六进制）：

45000054 00030000 2006...

a. 头部大小是多少？ b. 分组中有选项吗？

c. 数据大小是多少？ d. 分组被分段了吗？

e. 分组可以再经过多少路由器？

f. 分组携带的有效载荷的协议号是多少？

P4-3 IPv4 头部的哪个字段可能随着路由器变化而变化？

P4-4 判断带有以下信息的数据报是第一个分段还是中间分段还是最后一个分段，或唯一一个分段（没有分段）：

a. M 位被设为 1，且偏移值为 0。 b. M 位被设为 1，且偏移值为非 0。

P4-5 简要描述我们如何应对如下安全攻击：

a. 分组嗅探 b. 分组修改 c. IP 欺骗

P4-6 以下系统中地址空间的大小是多少？

a. 每个地址仅是 16 位的系统。

b. 每个地址由 6 个十六进制数字构成的系统。

c. 每个地址由 4 个 8 进制数字构成的系统。

P4-7 使用二进制表示法重写以下 IP 地址：

a. 130.11.5.88 b. 12.74.16.18 c. 201.24.44.32

P4-8 使用点分十进制表示法重写以下 IP 地址：

a. 01011110 10110000 01110101 00010101

b. 10201031 10201110 11010000 00110201

c. 01010111 10000100 00110111 00001111

P4-9 找出以下分类 IP 地址的类：

a. 130.34.54.12 b. 200.34.2.1 c. 245.34.2.8

P4-10 找出以下分类 IP 地址的类：

a. 01110111 11110011 10000111 11011101

b. 11101111 11000000 11110000 00011101

c. 11011111 10110000 00011111 01011101

P4-11 在无类寻址中，使用 CIDR 表示法将整个地址空间作为一个块表示出来。

P4-12 在无类寻址中，如果前缀长度（n）如下，块（N）的大小是多少？

a. $n=0$ b. $n=14$ c. $n=32$

P4-13 在无类寻址中，如果块（N）的大小如下，前缀长度（n）是多少？

a. $N=1$ b. $N=1024$ c. $N=2^{32}$

P4-14 用点分十进制表示法将如下前缀长度改为掩码：

a. $n=0$ b. $n=14$ c. $n=30$

P4-15 将如下掩码改成前缀长度：

a. 255.224.0.0 b. 255.240.0.0 c. 255.255.255.128

P4-16 以下哪个不是 CIDR 中的掩码？

a. 255.225.0.0　　b. 255.192.0.0　　c. 255.255.255.6

P4-17 以下每个地址属于一个块。找出块中的第一个和最后一个地址。

a. 14.12.72.8/24　　b. 200.107.16.17/18　　c. 70.110.19.17/16

P4-18 给出以下网络地址/掩码的最左 n 位，这 n 位可以用于转发表中（见图 4-45）。

a. 170.40.11.0/24　　b. 110.40.240.0/22　　c. 70.14.0.0/18

P4-19 请解释当分配给一个组织机构的块小于组织机构的主机数目时，DHCP 如何使用。

P4-20 对比 NAT 和 DHCP。两者都可以解决组织机构中地址短缺，但是使用不同的策略。

P4-21 假设我们有一个 8 位地址空间的互联网。地址在 4 个网络（N_0 到 N_3）间均分。互联网通信通过一个带有 4 个接口（m_0 到 m_3）的路由器完成。给出互联网略图和路由器的转发表（含有两列：二进制形式前缀和接口号）。这个路由器是连接网络的唯一一个路由器。给每个网络分配一个地址。

P4-22 假设我们有一个 12 位地址空间的互联网。地址在 8 个网络（N_0 到 N_7）间均分。互联网通信通过一个带有 8 个接口（m_0 到 m_7）的路由器完成。给出互联网略图和路由器的转发表（含有两列：二进制形式前缀和接口号）。这个路由器是连接网络的唯一一个路由器。给每个网络分配一个地址。

P4-23 假设我们有一个 9 位地址空间的互联网。地址在 3 个网络（N_0 到 N_2）间分配，分别有 64、192 和 256 个地址。互联网通信通过一个带有 3 个接口（m_0 到 m_2）的路由器完成。给出互联网略图和路由器的转发表（含有两列：二进制形式前缀和接口号）。这个路由器是连接网络的唯一一个路由器。给每个网络分配一个地址。

P4-24 将如下三个地址块组合成一个块：

a. 16.27.24.0/26　　b. 16.27.24.64/26　　c. 16.27.24.128/25

P4-25 一个带有一大块地址（12.44.184.0/21）的大型组织被分为一个使用块地址（12.44.184.0/22）的中型公司和两个小型组织机构。如果第一个小公司使用块（12.44.188.0/23），第二个小公司可以使用的剩余块是什么？请解释如果两个小公司的地址块仍然是原来公司的一部分，那么关于这两个小公司的数据报如何被正确路由？

P4-26 一个 ISP 被分配块 16.12.64.0/20。ISP 需要为 8 个组织机构分配地址，每个分配 256 个地址。

a. 找出 ISP 块中地址的数量和范围。

b. 找出每个组织机构的地址范围和未分配的地址范围。

c. 给出地址分布概述以及转发表。

P4-27 一个 ISP 被分配块 80.70.56.0/21。ISP 地址分配如下：2 个组织机构，每个分配 500 个地址；2 个组织分配地址，每个 250 个地址；以及 3 个组织机构每个 50 个地址。

a. 找出 ISP 块中地址的数量和范围。

b. 找出每个组织机构的地址范围和未分配的地址范围。

c. 给出地址分组略图以及转发表。

P4-28 一个组织被分配块 130.56.0.0/16。管理者想要创建 1024 个子网。

a. 找出每个子网的地址数量。

b. 找出每个子网前缀。

c. 找出第一个子网的首地址和末地址。

d. 找出最后一个子网的首地址和末地址。

P4-29 图 4-48 中路由器 R1 能否接收目的地址为 140.24.7.194 的分组？如果这件事情发生了分组会怎么样？

P4-30 假设图 4-48 中路由器 R2 接收目的地址为 140.24.7.42 的分组。这个分组如何被路由到它的最终目的地？

P4-31 假设结点 a、b、c 和 d 到结点 y 的最短路径和从结点 x 到结点 a、b、c 和 d 的代价如下：

$D_{ay}=5$　　$D_{by}=6$　　$D_{cy}=4$　　$D_{dy}=3$

$c_{xa}=2$　　$c_{xb}=1$　　$c_{xc}=3$　　$c_{xd}=1$

根据 Bellman-Ford 方程，结点 x 和结点 y 之间的最短距离 D_{xy} 是多少？

P4-32 假设在时间 t_1 使用 RIP 的路由器转发表里有 10 个实体。其中 6 个在时间 t_2 仍然有效。其中 4 个在时间 t_2 之前过期了 70、90、110 和 210 秒。找出时间 t_1 和时间 t_2 中周期计时器、超时计时器以及垃圾收集计时器的数量。

P4-33 OSPF 路由器何时发送以下报文？

a. 问候报文　　b. 数据描述报文　　c. 链路状态请求　　d. 链路状态更新

e. 链路状态确认

P4-34 为了理解表 4-4 中距离向量算法的是如何工作的，让我们将它应用在 4 结点互联网上，如图 4-111 所示。

假设所有结点首先被初始化。也假设每次对于一个结点（A、B、C、D）分别使用算法。请展示出这个过程会收敛，并且所有结点最终将拥有稳定的距离向量。

P4-35 在距离向量路由选择中，好消息（链路代价下降）将会传播得快。换言之，如果链路距离下降，所有结点会很快了解到这一点并更新它们的向量。在图 4-112 中，我们假设四结点互联网是稳定的，但是突然结点 A 和 D 之间的距离由 6 下降到了 1（可能是由于链路质量的某些提高）。请给出这个好消息是如何传播的，并且找出稳定化后每个结点的新距离向量。

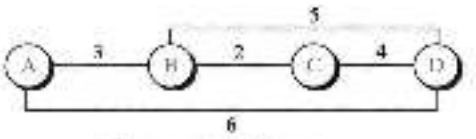


图 4-111　思考题 P4-34

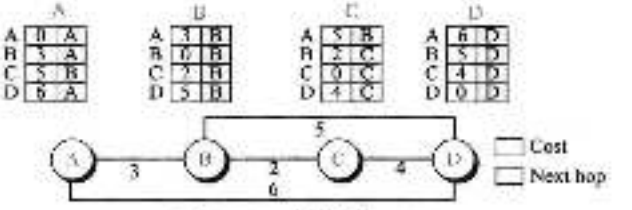


图 4-112　思考题 P4-35

P4-36 在距离向量路由选择中，坏消息（链路代价的增加）将会传播得慢。换言之，如果链路距离增加，有时要花很长时间让所有结点了解到这个坏消息。在图 4-112 中（见前一个思考题），我们假设四结点互联网是稳定的，但是突然结点 B 和 C 之间的距离由 2 增加到了无穷（链路失效）。请给出这个坏消息是如何传播的，并且找出稳定化后每个结点的新距离向量。假设这个实现使用周期计时器来触发对于邻居的更新（当存在变化时不再触发更新）。也假设如果结点从同一个邻居接收到更高的代价，它将使用新的代价，因为这意味着旧通告不再有效。为了使得稳定化更迅速，当下一跳不可达时，这个实现将毒害路由。

P4-37 在计算机科学中，当我们谈到算法时，我们经常需要问算法的复杂度（我们需要花多大计算量来完成这个算法）。为了找到距离向量算法的复杂度，当它从邻居接收向量时，请找出结点需要完成的操作数。

P4-38 假设图 4-113 中网络使用距离向量路由选择算法，每个结点的转发表如图所示。

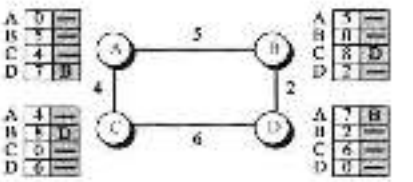


图 4-113　思考题 P4-38

如果每个结点周期性使用毒性逆转向邻居声明它们的向量，那么在以下选项情况下，大约一个周期内通告的距离向量是什么？

a. 从 A 到 B　　b. 从 C 到 D　　c. 从 D 到 B　　d. 从 C 到 A

P4-39 假设图 4-113 中（前一道思考题）网络使用距离向量路由选择算法，每个结点的转发表如图所示。如果每个结点周期性使用水平分裂向邻居声明它们的向量，那么在以下选项情况下，大约一个周期内通告的距离向量是什么？

a. 从 A 到 B　　b. 从 C 到 D　　c. 从 D 到 B　　d. 从 C 到 A

P4-40 假设图 4-113 中（思考题 P4-38）网络使用距离向量路由选择算法，每个结点的转发表如图所示。如果结点 E 被加入网络，它到结点 D 的链路代价是 1，不使用距离向量算法，你能为每个结点找到新的转发表吗？

P4-41 为图 4-65 中的结点 A 创建转发表。

P4-42 为图 4-63 中结点 G 创建最短路径树以及转发表。

P4-43 为图 4-63 中结点 B 创建最短路径树以及转发表。

P4-44 使用 Dijkstra 算法（见表格 4-4）为图 4-114 中结点 A 找出最短路径树以及转发表。

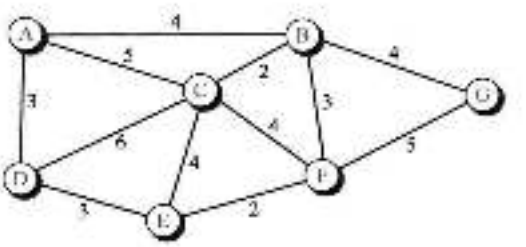


图 4-114　思考题 P4-44

P4-45 在计算机科学中，当我们遇到算法时，我们经常需要问算法的复杂度（我们需要花多大计算量来完成这个算法）。为了得到 Dijkstra 算法的复杂度，当结点数为 n 时，请找出为一个结点创建最短路径需要完成的搜索数量。

P4-46 假设图 4-115 中的 A、B、C、D 和 E 是自治系统（AS）。使用表 4-6 中的算法找出每个 AS 的路径向量。假设这种情况下最佳路径是穿过最少 AS 的路径。也假设算法首先初始化每个 AS，然后每次在一个系统上运行应用（A、B、C、D、E）。请展示出这个过程会收敛，并且所有 AS 最终将拥有稳定的距离向量。

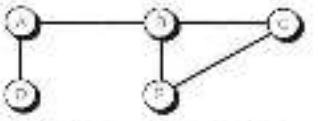


图 4-115 思考题 P4-46

P4-47 在图 4-79 中，假设 AS1 使用的域内路由选择协议是 OSPF，但 AS2 使用的是 RIP。请解释 R5 如何将分组路由到 N4。

P4-48 在图 4-79 中，假设 AS4 和 AS3 使用的域内路由选择协议是 RIP。请解释 R8 如何将分组路由到 N13。

P4-49 在图 4-116 中，假设路由器 R3 通过接口 m0、m1 和 m2 已经接收到来自源端 S 的同一个分组的三个副本。如果路由器 R3 使用 RPF 策略，那个报文应该从接口 m3 转发给剩余网络？

P4-50 假设 m 远小于 n 并且路由器 R 被连接到 n 个网络上，其中只有 m 个网络对与组 G 相关的分组感兴趣。路由器 R 如何仅仅向那些对组 G 感兴趣的网络发送分组的副本？

P4-51 在图 4-117 的网络中，请找出以下两种情况中路由器 R 的最短路径树：首先，如果网络使用 OSPF；第二，如果网络使用 MOSPF 且源端被连接到标记为 S 的路由器上。假设所有路由器都对相应的广播组感兴趣。

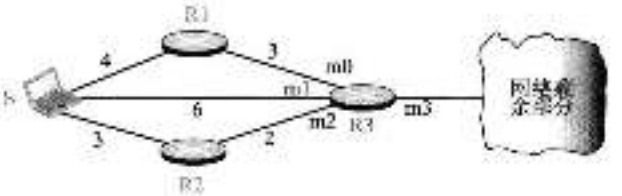


图 4-116 思考题 P4-49

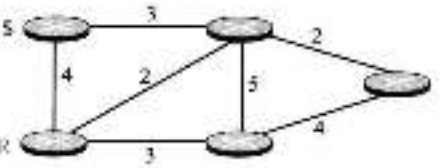


图 4-117 思考题 P4-51

P4-52 只通告一个网络的 RIP 报文多大？通告 n 个网络的 RIP 报文多大？

P4-53 一个使用 RIP 的路由器有如下转发表。请给出这个路由器发送的 RIP 更新报文。

目 的 端	代 价	下一跳路由器
Net1	4	B
Net2	2	C
NetG	1	F
Net4	5	G

P4-54 对比 IPv4 头部和 IPv6 头部。列表比较每个字段。

P4-55 用未缩短的冒号十六进制表示法给出如下 IPv6 地址：

a. 带有 64 个 0 紧接其后是 32 个二比特位（01）的地址。

b. 带有 64 个 0 紧接其后是 32 个二比特位（10）的地址。

c. 带有 64 个二比特位（01）的地址。

d. 带有 32 个四比特位（0111）的地址。

P4-56 给出以下地址的缩略形式：

a. 0000:aaaa:aaaa:0000:0000:0000:0000:0000

b. 1234:2346:3456:0000:0000:0000:0000:FFFF

c. 0000:0001:0000:0000:0000:aaaa:1200:1000

d. 0000:0000:0000:0000:FFFF:FFFF:24.123.12.6

P4-57 解压缩如下地址并给出完整的未缩短 IPv6 地址：

a. ::2222　　b. 1111::　　c. B:A:CC::1234:A

P4-58 给出如下 IPv6 地址的原型（未缩短）：

a. ::2　　b. 0:23::0　　c. 0:A::3

P4-59 与以下基于表 4-7 的 IPv6 地址相关的块或子块是什么？

a. FE80::12/10　　b. FD23::/7　　c. 32::/3

P4-60 一个组织机构被分配块 2000:1234:1423/48。这个组织机构中第一个和第二个子网中块的 CIDR 是什么？

4.8 模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在本章，我们使用 Wireshark 来捕获并研究一些网络层交换的分组。我们使用 Wireshark 以及其他计算机网络管理实用程序。请参见本书网站获得实验作业的完整细节。

Lab4-1 在第一个实验中，我们通过捕获并研究 IP 数据报来研究 IP 协议。

Lab4-2 在第二个实验中，我们捕获并研究其他实用程序如 ping 或 traceroute 产生 ICMP 分组。

4.9 编程作业

利用你选择的编程语言，编写源代码，编译并测试如下程序：

Prg4-1 编写一个程序来模拟距离向量算法（见表 4-4）。

Prg4-2 编写一个程序来模拟链路状态算法（见表 4-5）。

Prg4-3 编写一个程序来模拟路径向量算法（见表 4-6）。

数据链路层：有线网络

在第 2 章到第 4 章中，我们讨论了应用层、传输层和网络层。本章和下一章中，我们讨论数据链路层。TCP/IP 协议簇在数据链路层和物理层没有定义任何协议。这两层是网络的边界，当连接时它们补充了因特网。正如我们在第 1 章中讨论的，这些有线或者无线网络为 TCP/IP 协议簇中上面三层提供了服务。这可能给我们一个线索，即在现在的市场中有几个标准的协议。为此，我们在两章中讨论链路层：本章中我们给出数据链路层的一般概念，并讨论有线网络；在下一章中，我们讨论无线网络。

- 5.1 节介绍结点和链路的概念以及链路的类型，并说明数据链路层实际中如何分为两个子层：数据链路控制和介质访问控制。
- 5.2 节讨论数据链路层的数据链路控制（DLC）并解释由这一层提供的服务，如成帧、流和差错控制以及差错检测。
- 5.3 节讨论数据链路层的介质访问控制子层（MAC）。我们解释不同的访问方法，如随机访问、受控访问和通道化。
- 5.4 节讨论链路层寻址以及如何使用地址解析协议（ARP）来获取一个结点的链路层地址。
- 5.5 节介绍有线局域网，特别是当今主要的局域网协议——以太网。我们通过不同时代的以太网，说明它是如何演变的。
- 5.6 节讨论其他的在当今因特网中存在的有线网络，例如点对点网络和交换网络。
- 5.7 节讨论 TCP/IP 协议下面 3 层的连接设备，例如集线器、链路层交换机和路由器。

5.1 介绍

在第 4 章中，我们知道在网络层的通信是主机到主机的。尽管可以片段和重组，但是数据报是一个数据单元，它从世界某个地方的一台主机发送到世界某个地方的另一台主机。然而因特网是通过连接设备（路由器或者交换机）连接在一起的网络的组合。如果数据报要从一台主机到达另一台主机，它就需要通过这些网络。

图 5-1 展示了在 Alice 和 Bob 间的通信，我们使用前三章中相同的情节。然而，数据链路层的通信由路径上数据链路层之间的 5 个独立逻辑连接组成。

Alice 的计算机的数据链路层与路由器 R2 的数据链路层通信。路由器 R2 的数据链路层与路由器 R4 的数据链路层通信，等等。最后，路由器 R7 的数据链路层与 Bob 的计算机的数据链路层通信。在源主机或者目的主机只涉及一个数据链路层，但是在每一个路由器涉及两个数据链路层。这是因为 Alice 和 Bob 的计算机只与一个单一网络连接；而每一个路由器从一个网络获取输入，将输出发送至另一个网络。

5.1.1 结点和链路

尽管在应用层、传输层和网络层的通信是端到端的，但是数据链路层的通信是结点到结点的。正如我们在前面章节中学习的，来自于因特网一个点的数据单元需要通过很多网络（局域网和广域网）才能到达另一个点。这些局域网和广域网通过路由器相连。通常习惯上，将涉及的两个端主机

和路由器视为结点，将它们之间的网络视为链路。下面是当数据单元路径上只有 6 个结点时，链路和结点的一种简单表示法。

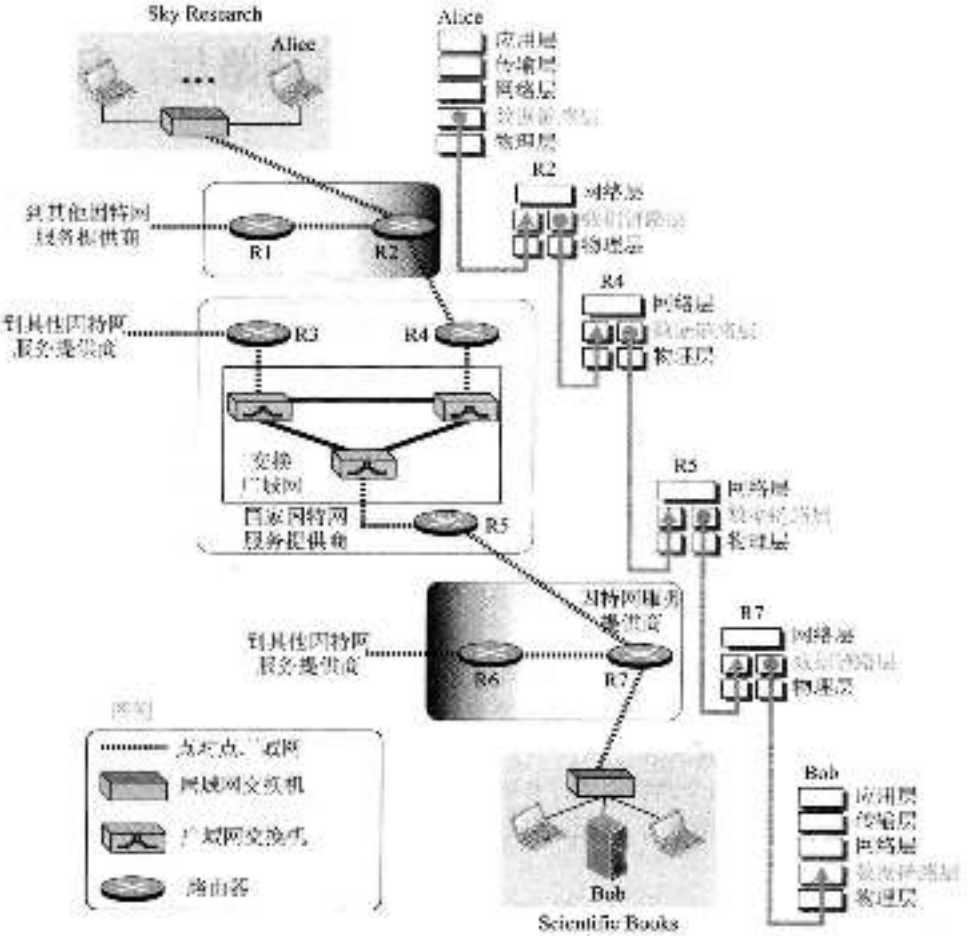


图 5-1　数据链路层的通信



a. 因特网的一小部分



b. 结点和链路

图 5-2　结点和链路

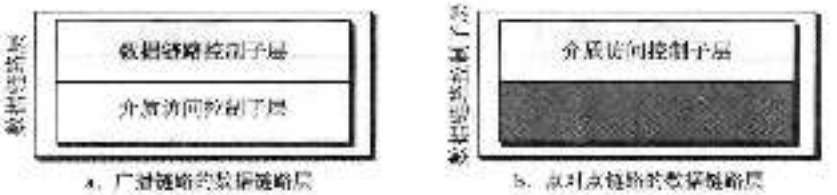
第一个结点是源主机；最后一个结点是目的主机。其他的 4 个结点是 4 个路由器。第 1 条、第 3 条和第 5 条链路代表了 3 个局域网；第 2 条和第 4 条链路代表了 2 个广域网。

5.1.2　两类链路

尽管两个结点通过诸如电缆或者空气等传输介质物理地连接起来，我们需要记住数据链路层控制了如何使用介质。我们可以使数据链路层使用介质的全部容量，也可以使数据链路层只使用链路的部分容量。换言之，我们可以有点对点链路或者广播链路。在点对点链路中，链路专供给两个设备使用；在广播链路中，链路在几对设备之间共享。例如，当两个朋友使用传统的家庭电话聊天时，他们使用点对点链路；当他们使用蜂窝电话时，他们使用广播链路（空气由很多蜂窝电话用户共享）。

5.1.3　两个子层

为了更好地理解链路层的功能以及它提供的服务，我们将数据链路层分为两个子层：数据链路控制（DLC）和介质访问控制（MAC）。正如我们在本章和下一章将要看到的，这是不寻常的，局域网协议实际上使用相同的策略。数据链路控制子层处理点对点链路和广播链路的共有的问题；介质访问控制子层只处理广播链路的特定问题。如图 5-3 所示，我们将数据链路层的两类链路分离。



a. 广播链路的数据链路层　　b. 点对点链路的数据链路层

图 5-3　将数据链路层分为两个子层

本章中，我们首先讨论数据链路控制子层，该层为两类链路共有。然后我们讨论介质访问控制子层，该层只在广播链路中使用。在讨论这两个子层后，我们讨论每个分类的协议。

5.2　数据链路控制

数据链路控制处理两个邻近结点的通信过程，即结点到结点的通信，无论该链路是专用的还是广播的。数据链路控制（Data Link Control，DLC）的功能包括成帧、流量控制和差错控制，以及差错检测和差错纠正。本节中，我们首先讨论成帧，或者说如何组织物理层所承载的位。然后我们讨论流量控制和差错控制。差错检测技术将在本节末尾讨论。

5.2.1　成帧

物理层的数据传输是指以信号的形式将位从源主机移动到目的主机。物理层提供了位同步以保证发送方和接收方使用相同的位周期和时序。我们在第 7 章讨论物理层。

另一方面，数据链路层需要将位组合成帧，以便使帧区别于另一个。我们的邮局系统实践了一种成帧类型。将一封信放入一个信封的简单动作就将一段信息与另一段信息分离开来；信封充当了分界符。另外每一个信封定义了发送方和接收方的地址，这是有必要的，因为邮局系统是一种多对多的传输设施。

在数据链路层中，通过添加发送方和接收方的地址，成帧将从源端到目的端的报文分离开来。目的地址定义了分组的去向；发送地址帮助接收方确认接收。

尽管整个报文能够打包为一个帧，但是通常不那么做。一个原因是一个帧可能非常大，这就使得流量控制和差错控制非常低效。当一个报文在非常大的帧中承载时，甚至一个单个位差错将会要求将个帧的重传。当一个报文分为多个小的帧时，一个单个位差错只影响那个小帧。

帧长度

帧的大小可以是固定的，也可以是可变的。在固定长度成帧中，没有必要定义帧的边界；长度

本身可以作为分界符。这种成帧的一个例子就是 ATM 广域网，它使用称为信元的固定大小的帧。本章稍后我们讨论 ATM。

本章中我们主要讨论的是可变长度成帧，它在局域网中非常流行。在可变长度成帧中，我们需要一种方式来定义一个帧的结束和下一个帧的开始。过去，主要使用两种方法来达到这一目的：面向字符的方法和面向位的方法。

面向字符成帧

在面向字符（或者面向字节）成帧中，传输的数据是来自于诸如 ASCII（参见附录 A）编码系统的 8 位的字符。头部通常携带源地址和目的地址以及其他的控制信息，而尾部携带差错检测冗余位，也是 8 位的倍数。为了将一个帧和下一个帧分离，一个 8 位（1 字节）的标记（flag）在帧的开始和结尾处添加。标记由与协议相关的特殊字符组合而成，标明一个帧的开始和结束。图 5-4 说明了面向字符协议中帧的格式。

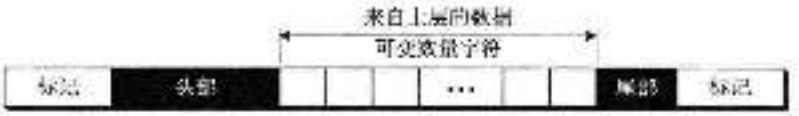


图 5-4　面向字符协议中的帧

当数据链路层只有文本交换时，面向字符成帧是很流行的。标记可以选择文本通信中不使用的任意字符。然而现在，我们发送其他类型的信息，如图像、音频、视频等信息，标记所使用的任何模式可能也是这些信息的一部分。如果这种情况发生，当接收方在数据中间遇到这种模式时，它就认为已经到达帧结束处。为了解决这个问题，在面向字符成帧中使用了字节填充策略。在字节填充（或者字符填充）中，如果数据中存在与标记相同模式的字符，那么一个特定的字符将会添加到帧的数据部分。数据部分被填入一个额外的字节。这个字节通常称为转义字符（escape character，ESC），它有预先定义的位模式。当接收方遇到转义字符时，它将转义字符从数据部分移除，并仍将下一个字符作为数据处理，而非当做分界标记。

通过转义字符来进行字节填充允许帧的数据段中存在标记，但是又产生了另一个问题。如果文本中包含一个或者多个转义字符，转义字符后面又存在与标记相同模式的一个字节该怎么办？接收方移除转义字符，但是保留下一个字节，该字节就会被解释为帧的末端。为了解决这个问题，作为文本一部分的转义字符必须由另一个转义字符标记出来。换言之，如果转义字符是文本的一部分，那么要增加额外的转义字符来表明第二个字符是文本的一部分。图 5-5 说明了这一情况。

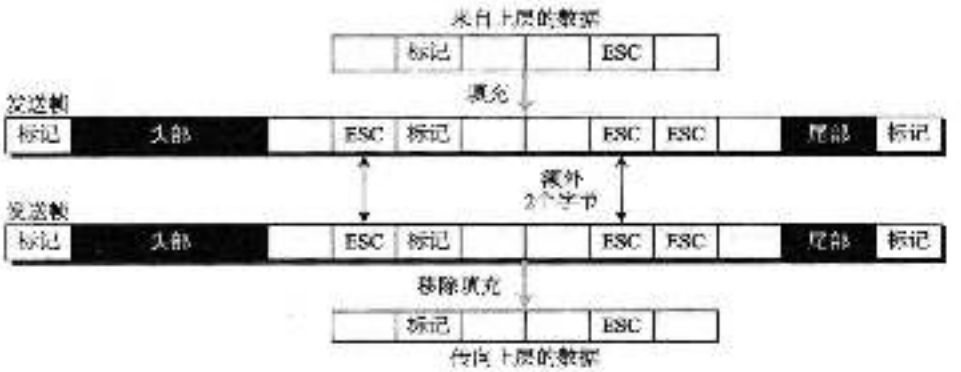


图 5-5　字节填充和移除

字节填充是当文本中存在一个标记或者转义字符时，添加一个额外字节的过程。

面向字符的协议在数据通信中产生了另一个问题。如今广泛使用的编码系统，例如 Unicode 有 16 位和 32 位字符，就会与 8 位字符产生冲突。所以我们可以说，下面讨论的面向位的协议将是大势所趋。

面向位成熟

在面向位成帧中，一个帧的数据段是一个位序列，它将由上层翻译为文本、图形、音频、视频等等。然而除了头部（也可能是尾部）之外，我们仍然需要分界符区分不同帧。大部分的协议使用特殊的 8 位模式标记 01111110 作为分界符来标明帧的开始和结束，如图 5-6 所示。

该标记也会产生与面向字符协议中相同的问题。也就是说，如果标记模式在数据中出现，我们需要以某种方法来通知接收方这不是帧的结束。我们通过填充 1 个单个位（而不是一个字节）来使得该模式区别于标记。这种策略称为**位填充**（bit stuffing）。在位填充中，当遇到 1 个 0 和 5 个连续的 1 时，添加一个额外的 0。这个额外的填充最终被接收方移除。

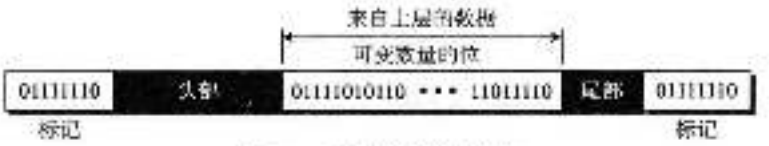


图 5-6　面向位协议中的帧

注意，在 1 个 0 和 5 个连续的 1 后面添加 1 个额外的位而不管接下来那个位是什么。这就保证了标志位序列不会出现在帧中。

> 位填充是指在数据中出现 1 个 0 和其后 5 个 1 时添加一个额外的 0 的过程，这样接收方就不会误认 011111C 为一个标示。

图 5-7 说明了发送方填充位和接收方移除位的过程。注意即使我们在 5 个 1 后有 1 个 0，我们仍然填充 1 个 0。这个 0 将会由接收方移除。

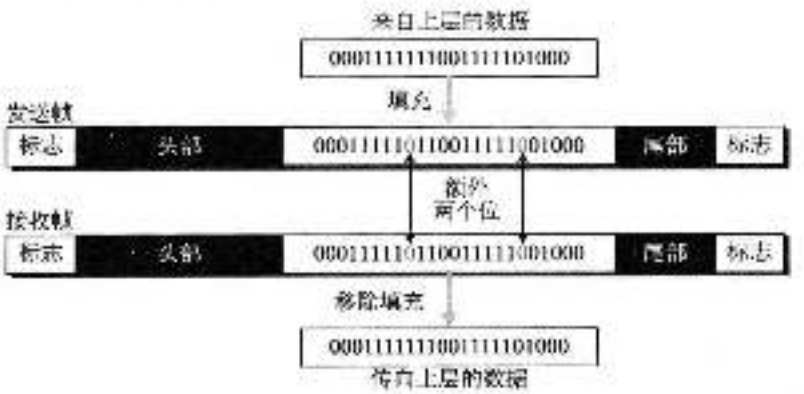


图 5-7　位填充和移除

这就意味着如果在数据中出现标记的模式 01111110，它将会变成 011111010（填充后），不会被接收方误认为是一个标记。真正的标记 01111110 是不会被发送方填充的，故而能被接收方正确确认。

5.2.2　流量控制和差错控制

我们在第 3 章定义了流量控制和差错控制。数据链路控制子层的一个功能就是在数据链路层进行流量控制和差错控制。

流量控制

正如我们在传输层（第 3 章）讨论的，流量控制在接收确认前两端能够被发送的数据数量。在数据链路层，流量控制是数据链路控制子层的一项职能。数据链路层流量控制的思想和我们讨论的传输层的思想遵循相同的原则。尽管传输层流量控制是端到端的（主机到主机），而数据链路层的流量控制是通过链路的，结点到结点的。

差错控制

差错控制包括差错检测和差错纠正。它允许接收方通知发送方在传输过程中有帧的丢失或者被坏，并协调发送方重新传输这些帧。在数据链路层，术语差错控制通常指差错检测和重传的方法。在数据链路层中，差错控制一般容易实现：在交换中的任意时刻检测到一个差错，就要重传这个出错的帧。然而，我们需要强调在传输层的差错控制是端到端的，但是数据链路层的差错控制是通过链路的，结点到结点的。换言之，每一次一个帧通过链路，我们就需要确认这个帧没有出错。

5.2.3　差错检测和纠错

在数据链路层，如果在两个结点间的一个帧被破坏，在它继续传播到其他结点之前就需要纠正该差错帧。但是大多数的链路层协议只是简单地丢弃了这个帧，让上层协议处理这个帧的重传。但是，一些无线协议试图纠正这个被破坏的帧。

介绍

让我们首先直接或间接地讨论有关差错检测和纠错的问题。

差错类型

无论何时，当位流从一点流动到另一点时，由于干扰（interference）的存在，都可能经受不可预测的变化。这些干扰可能会改变信号的波形。术语**单个位差错**（single-bit error）是指在给定的数据单元（如一个字节、字符或者分组）中，只有一个位从 1 变为 0 或者从 0 变为 1。术语**突发性差错**（burst error）是指在数据单元中，2 个或者更多的位从 1 变为 0 或者从 0 变为 1。图 5-8 说明了在数据单元中单个位差错和突发性差错的影响。

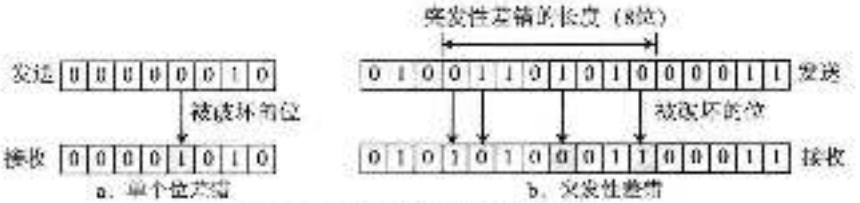


图 5-8　单个位差错和突发性差错

突发性差错比单个位差错更容易发生，因为噪声信号的持续时间要比 1 个位的持续时间长，这就意味着当噪声影响数据时，它要影响很多位。受影响的位的数量取决于数据速率和噪声持续时间。例如，如果我们以 1kbps 的速率发送数据，一个 1/100 秒的噪声就会影响 10 位；如果我们以 1Mbps 的速率发送数据，同样的噪声就会影响 10 000 位。

冗余

检错或纠错的核心概念是冗余。为了能够检测或者纠正，我们除了发送数据外，还需要发送一些额外的位。这些冗余位由发送方添加，由接收方移除。它们的存在允许接收方检测或者纠正被破坏的位。

检错和纠错

差错的纠正比检测更难。在差错检测中，我们只需要查看差错是否发生。答案只是简单的是或者否。我们甚至对发生差错的位的个数不感兴趣。单个位差错和突发性差错对我们来说是一样的。

在差错纠正中，我们需要知道被破坏的位的精确数目，更重要的是，知道它们在报文中的位置。差错的个数和报文的长度是重要的因素。如果我们需要纠正 8 位数据单元中的单个位差错，我们需要考虑 8 个可能的差错位置；如果我们需要纠正相同长度数据单元中的 2 个差错，我们需要考虑 28（8 中选 2 的组合）种可能性。可以想象，接收方在 1000 位的数据单元里查找 10 个差错的困难性。我们集中精力于差错检测；差错纠正更难，但是我们将在第 8 章中简略地讨论一下。

编码

通过各种编码方案实现冗余。发送方通过某种方法建立冗余位和真实数据之间的某种关系，以此来增加冗余位。接收方检测这两者之间的关系来检错。冗余位和数据位比率以及方法的健壮性都是任意编码方案中的重要因素。

我们可以将编码方案分为两大类：块编码（block coding）和卷积编码（convolution coding）。本书中我们只专注于块编码，而卷积编码更加复杂，超出了本书的范围。

块编码

在块编码中，我们将报文分块，每块 k 位，称为**数据字**（dataword）。我们为每一块增加 r 个冗余位，使得长度 $n = k + r$。这样形成的 n 位的块称为**代码字**（codeword）。这额外的 r 位如何选择或者计算将在后面讨论。目前，重要的是知道我们有一组长度为 k 的数据字和一组长度为 n 的代码字。数据字的长度为 k 位，我们就有 2^k 个数据字组合；代码字长度 n 位，我们就有 2^n 个代码字的组合。既然 $n>k$，可能的代码字数目就比可能的数据字大。块编码处理是 1 对 1 的：相同的数据字总是被编码为相同的代码字。这意味着有 2^n-2^k 个代码字没有使用。我们称这些代码字为无效的或者非法的。差错检测中的欺骗就是指这些无效码的存在，我们后面会讨论。如果接收方收到了无效代码字，这表明数据在传输过程中被破坏了。

差错检测

通过使用块编码，如何检测差错？如果满足下面两个条件，接收方就能检测出原始代码字的变化。

1. 接收方（或者能够查到）有一张有效代码字表。
2. 原始代码字已经变为无效的。

图 5-9 说明了差错检测中块编码的任务。

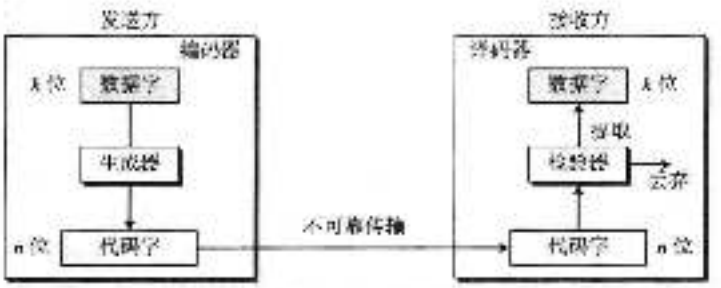


图 5-9　块编码中差错检测过程

发送方通过生成器根据数据字创建代码字。生成器是使用编码规则和过程（稍后讨论）的程序。每一个发送到接收方的代码字在传输过程可能改变。如果接收到的代码字和有效代码字中的一个相同，那么接收方接收该码字；相应的数据字被提取出来使用。如果接收到的代码字不是有效的，接收方丢弃该代码字。然而，如果代码字在传输过程中被破坏了，但是接收到的代码字能够和一个合法的代码字匹配，那么该差错仍然不能检测到。

例 5.1　假设 k=2，n=3。表 5-1 给出了数据字和代码字的列表。稍后，我们将会看到如何从数

据字生成代码字。

表 5-1　例 5.1 中用于差错检测的一种编码

数据字	代码字	数据字	代码字
00	000	10	101
01	011	11	110

假设发送方将数据字 01 编码为 011，并将其发送给接收方。考虑以下情况：

1. 接收方接收到 011。这是一个有效的代码字。接收方从中提取数据字 01。

2. 在传输过程中，代码字被破坏了，接收方收到 111（最左边的位被破坏了）。这不是一个有效的代码字，接收方丢弃。

3. 代码字在传输过程中破坏了，接收方收到 000（右边的两个位被破坏了）。这是一个有效代码字。接收方错误地提取了数据字 00。两个破坏的位使得差错无法检测。

> 差错检测编码只能检测那些它设计时假设的差错的类型；其他类型的差错可能仍然无法检测。

汉明距离

差错控制编码中一个核心概念就是汉明距离（Hamming distance）。两个字（相同长度）之间的汉明距离是它们之间对应的位的值不同的数量。我们以 $d(x, y)$ 表示两个字 x 和 y 之间的汉明距离。我们也许很好奇，为什么汉明距离对于差错检测很重要。原因是接收到的代码字和发送的代码字之间的汉明距离就是在传输过程中被破坏的位的数目。例如，如果发送的代码字为 00000，而接收的代码字为 01101，有 3 个位有差错，并且这两个代码字之间的汉明距离 $d(00000,01101)=3$。换言之，如果发送的和接收的代码字的汉明距离不是 0，那么该代码字在传输过程中就被破坏了。

如果我们对这两个字进行异或（⊕）运算，然后计算结果中 1 的个数，就可以很简单地得出汉明距离。注意汉明距离大于或者等于 0。

> 两个字之间的汉明距离是它们对应位的值不同的数量。

例 5.2　让我们来计算两组字的汉明距离。

1. 汉明距离 $d(000,011)$ 是 2，因为 $(000 \oplus 011)$ 是 011（2 个 1）。

2. 汉明距离 $d(10101,11110)$ 是 3，因为 $(10101 \oplus 11110)$ 是 01011（3 个 1）。

差错检测的最小汉明距离

在一组代码字中，最小汉明距离是指所有可能代码字对中最小的汉明距离。如果我们想要检测多达 s 个差错，我们需要计算编码中的最小汉明距离。如果在传输过程中有 s 个差错发生，那么发送的代码字和接收的代码字间的汉明距离就是 s。如果我们的系统能够检测多达 s 个差错，那么有效编码之间的最小距离就必须是 $(s+1)$，这样接收的代码字就不会匹配一个有效代码字。换言之，如果所有的有效代码字之间的最小距离是 $(s+1)$，接收到的代码字就不会被误认为另一个正确代码字。差错将会被检测出来。在这里我们需要澄清一点：尽管 $d_{min}=s+1$ 的编码可能在某些特定情况下检测到多于 s 个差错，但是只有 s 个或者更少的差错可以保证被检测到。

> 为了保证检测出所有情况下多达 s 个差错，表块编码中的最小汉明距离必须是 $d_{min}=s+1$。

我们可以用几何学的观点看待这个条件。假设发送的代码字 x 是一个圆的圆心，半径为 s。所有接收的有 0 到 s 个差错的代码字是该圆内的点或者圆上的点。所有的其他的有效代码字必须是圆外的，如图 5-10 所示。这就意味着 d_{min} 一定是比 s 大的整数，或者 $d_{min}=s+1$。



图 5-10　差错检测中 d_{min} 的几何意义

例 5.3　我们第一个编码方案（表 5-1）中的最小汉明距离是 2。这个编码保证 1 个差错的检测。

例如，如果第 3 个代码字（101）被发送了，并且一个差错发生了，那么接收的代码字不会匹配任何有效的代码字。然而，如果两个差错发生了，接收到的代码字就可能匹配一个有效代码字，差错就无法检测出来。

例 5.4　一种编码方案中有汉明距离 $d_{min}=4$。该编码保证了多达 3 个差错（$d=s+1$ 或者 $s=3$）的检测。

线性块编码

现在使用的几乎所有的块编码都属于一个称为线性块编码（linear block codes）的子集。差错检测和纠错中非线性块编码的应用不广泛，因为它们的结构使得理论分析和实现都很困难。因此我们专注于线性块编码。线性块编码的正式定义要求抽象代数（特别是 Galois 领域）的知识，这超出了本书的范围。因此我们给出一个非正式定义。线性块编码是一种编码，其中由两个有效代码字执行异或（模 2 加法）运算产生另一个有效代码字。

例 5.5　表 5-1 中的编码是一种线性块编码，因为任意代码字和其他代码字的异或结果是一个有效代码字。例如，第 2 个和第 3 个代码字的异或产生了第 4 个代码字。

线性块编码的最小距离

求解线性块编码的最小汉明距离很简单。最小汉明距离是具有最小 1 的个数的非 0 有效代码字中 1 的个数。

例 5.6　在我们第一个编码（表 5-1）中，非零代码字中 1 的个数是 2、2 和 2。所以最小汉明距离 $d_{min}=2$。

奇偶校验编码

可能最常见的差错检测编码是奇偶校验编码（parity-check code）。这种编码是一种线性块编码。在这种编码中，一个 k 位的数据字变成 n 位的代码字，其中 $n=k+1$。额外的位称为奇偶校验位（parity bit），用来使代码字中 1 的总个数是偶数。尽管一些实现指定了 1 的个数是奇数，但是我们只讨论偶数的情况。这种编码的最小汉明距离是 $d_{min}=2$。这意味着这种编码是单个位差错检测编码。我们的第一个编码方案（表 5-1）是一个奇偶校验编码（$k=2$，$n=3$）。在表 5-2 中的编码也是奇偶校验编码，其中 $k=4$，$n=5$。

表 5-2　简单奇偶校验编码 C（5,4）

数据字	代码字	数据字	代码字
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

图 5-11 说明了一种编码器（发送方）和一种译码器（接收方）的可能结构。

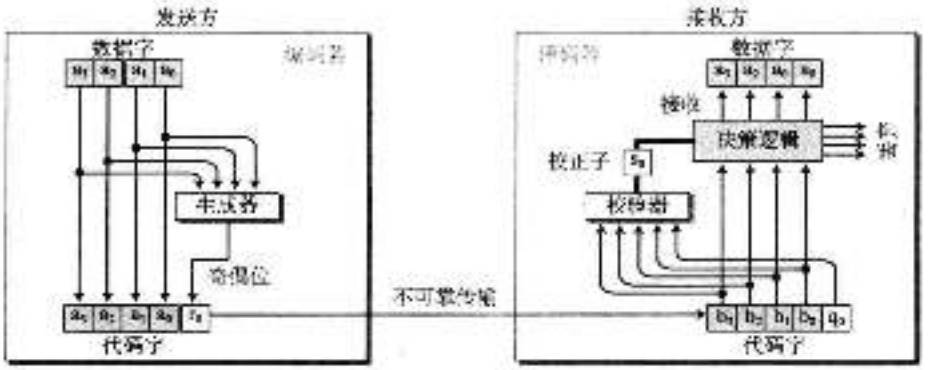


图 5-11　简单奇偶校验编码的编码器和译码器

编码器使用生成器获取一个 4 位数据字（a_0、a_1、a_2 和 a_3）的副本，并产生一个奇偶位 p_0。数据字的各个位和奇偶位（parity bit）产生一个 5 位代码字。添加的奇偶位使得代码字中 1 的个数为偶数。这通常通过将数据字的 4 位相加实现（模 2 运算），结果是奇偶位的值。换言之，

$$p_0 = a_3 + a_2 + a_1 + a_0 \text{(模2运算)}$$

如果 1 的个数是偶数，结果是 0；如果 1 的个数是奇数，结果是 1。这两种情况下，代码字中 1 的总数总是偶数。

发送方发送一个代码字，该代码字可能在传输过程中被破坏。接收方接收一个 5 位的字。接收方的校验器和发送方的生成器做同样的事情，不同之处在于，校验器要将所有的 5 位都要相加。我们将结果称为**校正子**（syndrome）。它只有 1 位。当收到的代码字中 1 的个数是偶数时，校正子为 0，否则为 1。

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \text{(模2运算)}$$

该校正子传递给决策逻辑分析器。如果校正子是 0，那么在接收到的代码字中，没有检测出差错。接收代码字中的数据部分作为数据字接收；如果校正子是 1，接收代码字的数据部分被丢弃。没有数据字生成。

例 5.7　让我们看一些传输的情形。假设发送方发送数据字 1011。由此数据字产生的代码字为 10111，该代码字发送给接收方。我们查看 5 种情况：

1. 没有差错发生；接收的代码字为 10111。校正子为 0。生成数据字 1011。
2. 一个单个位差错改变了 a_1。接收到的代码字是 10011。校正子为 1，没有数据字生成。
3. 一个单个位差错改变了 q_0。接收到的代码字是 10110。校正子为 1，没有数据字生成。注意，尽管数据字中没有位被破坏，但是因为编码不够复杂不能说明被破坏位的位置，因此仍然没有数据字生成。
4. 一个差错改变了 a_0，第二个差错改变了 a_3。接收到的代码字为 00110。校正子是 0。在接收方生成 0011。注意，这里由于校正子的值生成了错误的数据字。简单的奇偶校验译码器不能检测出偶数个差错。差错互相抵消使得校正子为 0。
5. 三个位 a_3、a_2 和 a_1 被破坏了。接收的代码字为 01011。校验子是 1。没有数据字生成。这说明简单奇偶校验码除了能保证检测出单个位差错外，还可以检测出任意奇数个差错。

奇偶校验码可以检测出奇数个差错。

循环编码

循环编码是有一个附加性质的特殊线性块编码。在**循环编码**（cyclic code）中，如果代码字循环移位，其结果为另一个代码字。例如，如果 1011000 是一个代码字，我们循环左移，然后 0110001 也是一个代码字。这种情况下，如果我们称第一个代码字中的位为 a_0 到 a_6，称第二个代码字中的位为 b_0 到 b_6，我们可以通过如下方式进行移位：

$$b_1 = a_0 \; b_2 = a_1 \; b_3 = a_2 \; b_4 = a_3 \; b_5 = a_4 \; b_6 = a_5 \; b_0 = a_6$$

在最右边的等式中，第一个代码字中的最后一个位变成了第二个代码字中的第一位。

循环冗余校验

我们能够生成循环编码来纠错。但是，所需的理论背景超出了本书的范围。本节中，我们简单地讨论循环编码的一个子集——**循环冗余校验**（cyclic redundancy check，CRC），它用于诸如局域网和广域网等网络中。

表 5-3 说明了一个 CRC 编码的例子。我们可以看到这种编码的线性和循环性。

图 5-12 说明了编码器和译码器的一种可能设计方案。

在编码器中，数据字有 k 位（这里 $k=4$）；代码字有 n 位（这里 $n=7$）。数据字的长度通过在该字的右边添加 $n-k$（这里是 3）个 0 来增加。n 位的结果传给生成器。生成器使用 $n-k+1$ 位（这里是

4）的除数，该除数是预定义的并经双方同意的。生成器用除数除增加后数据字（模 2 除法）。除法的商被丢弃；余数（$r_2r_1r_0$）被系加到数据字上产生代码字。

译码器接收到代码字（可能在传输中被破坏）。所有的 n 位的副本被传递给校验器。它是生成器的复制品。由校验器产生的余数是 $n-k$ 位（这里是 3 位）的校正子，它被传给决策逻辑分析器。该分析器有一个简单函数。如果校正子都是 0，该代码字最左边的 4 位被接收为数据字（视为无差错）；否则，这 4 位被丢弃（有差错）。

表 5-3　C(7,4)的 CRC 编码

数据字	代码字	数据字	代码字
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

编码器　让我们仔细考察编码器。编码器获取数据字，并用 $n-k$ 个 0 将其扩充。然后用除数去除增加后的数据字，如图 5-13 所示。

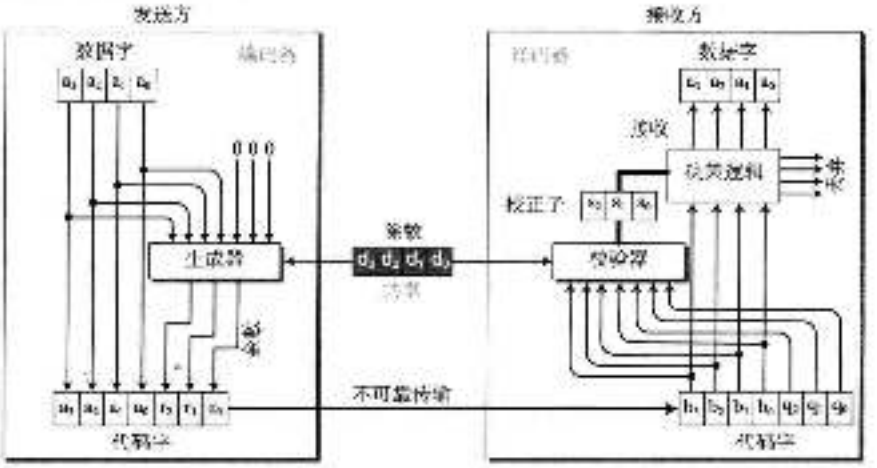


图 5-12　CRC 编码器和译码器

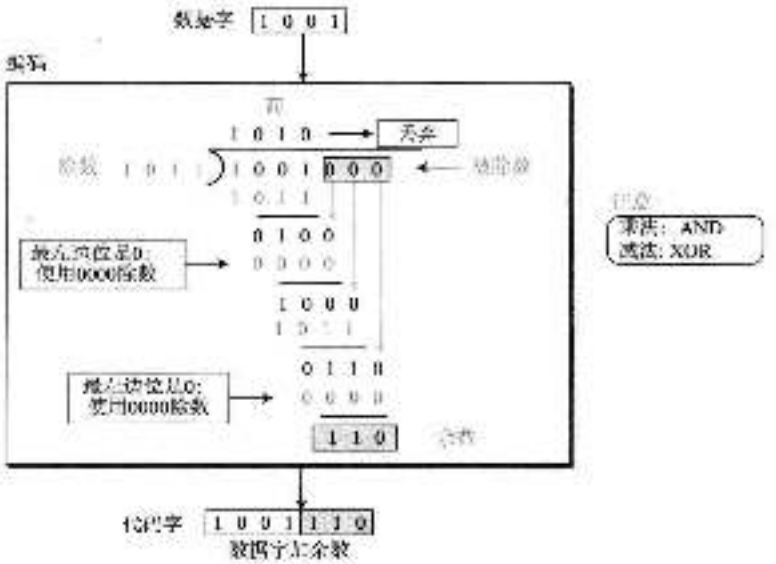


图 5-13　CRC 编码器中的除法

模 2 二进制除法的过程与熟悉的用于十进制数字的除法的过程相同。但是，在这种情况下的加法和减法是一样的：我们使用异或操作来实现这两种运算。

和十进制除法一样，处理过程是逐步的。每一步中，除数的副本和被除数的 4 位进行异或运算。异或操作的结果（余数）是 3 位的（这种情况下），一个额外的位移下来添加至该结果，使其长度为 4 位，然后用于下一步。在这种类型的除法中，我们需要记住重要的一点：如果被除数最左边（或者每一步中使用的部分）的位是 0，该步中不能使用常规的除数；我们需要使用全 0 的除数。

当没有可以移下来的位时，我们就得到了结果。3 位的余数形成了校验位（r_2、r_1 和 r_0）。它们附加到数据字上形成了代码字。

译码器　代码字在传输过程中可能变化。译码器执行和编码器相同的除法过程。除法的余数是校正子。如果校正子全是 0，就没有差错；数据字从接收的代码字中分离出来并被接受。否则，整个代码字被丢弃。图 5-14 说明了两种情况：左边的图说明了当没有差错发生时校正子的值；该校正子为 000。右边的图说明了单个差错发生的情况。该校正子不全是 0（此处为 011）。

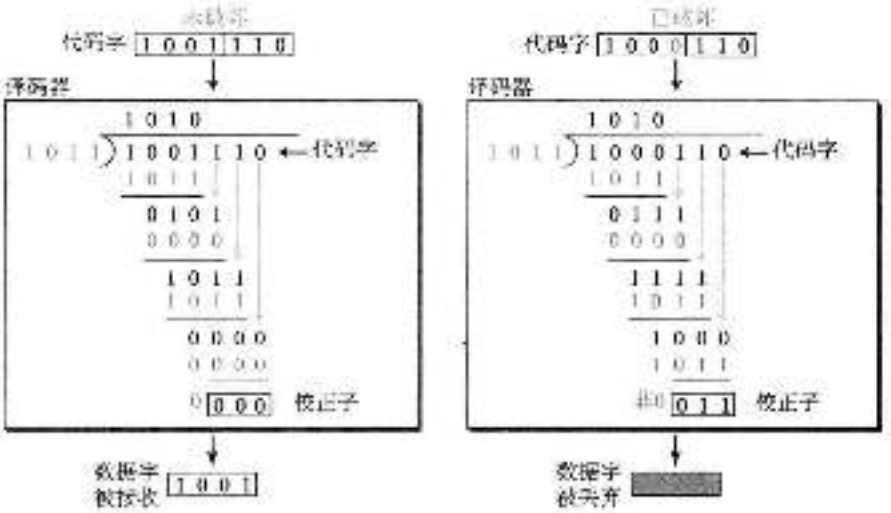


图 5-14　两种情况下 CRC 译码器中的除法

除数　我们可能想知道如何选择除数 1011。这依赖于对编码的预期。我们在本书的网站上讨论了该标准。在网络中使用的一些标准除数如表 5-4 所示。在除数名称中的数字（例如 CRC-32）代表除数多项式次数（最高次幂）。位的数目总是比多项式的次数多 1。例如，CRC-8 有 9 位，CRC-32 有 33 位。

多项式

理解循环编码以及如何分析它们的更好的方式是将它们表示为多项式。我们在本书网站上讨论了多项式的相关情况，感兴趣的读者可以参考。

表 5-4　标准多项式

名　称	二进制数	应　用
CRC-8	100000111	ATM 头部
CRC-10	11000110101	ATM AAL
CRC-16	10001000000100001	HDLC
CRC-32	100000100110000010001110110110111	LAN

要求

我们可以从数学上证明位模式被认为是生成器（除数）需要至少两个特性：

1. 该模式应该至少有两位。
2. 最右边和最左边的位都是 1。

性能

我们可以从数学上证明 CRC 编码的性能如下所示。

- **单个位差错**：所有的合格的生成器（如前所述）能够检测任意的单个位差错。

- **奇数个差错**：如果使用模 2 二进制除法，生成器能够均匀地被(11)除，那么所有的合格的生成器能够检测任意奇数个差错；否则，只有部分奇数个差错能被检测出来。
- **突发性差错**：如果我们假设突发性差错的长度是 L 位，余数长度为 r 位（r 是生成器长度减 1；它也是代表生成器多项式的最高次幂的值）：

a. 长度 $L \leq r$ 的所有的突发性差错能够被检测。

b. 长度 $L = r + 1$ 的所有的突发性差错能够以 $1 - (0.5)^{r-1}$ 的概率被检测。

c. 长度 $L > r + 1$ 的所有的突发性差错能够以 $1 - (0.5)^{r}$ 的概率被检测。

循环编码的优点

循环编码可以很容易地由硬件和软件实现。由硬件实现时它们非常快。这就使得循环编码是很多网络的很好的候选编码。在本书网站，我们说明了除法是如何通过移位寄存器实现的，该移位寄存器就包含在结点的硬件中。

校验和

校验和（checksum）是一种差错检测技术，它能够被应用于任意长度的报文中。在因特网中，校验和技术大部分用在网络层和传输层中，而非在数据链路层。然而，为了使我们关于差错检测技术的讨论完整起来，我们在本章中讨论校验和。

在源端，报文首先被分成 m 位的单元。然后生成器生成一个称为校验和的额外的 m 位单元，该单元随报文一起发送。在目的端，校验器由报文和发送的校验和生成一个新的校验和。如果新的校验和全是 0，报文就被接收了；否则，该报文被丢弃（见图 5-15）。注意在实际实现时，校验和单元没有必要添加到报文的末尾；它可以被添加到报文的中间。

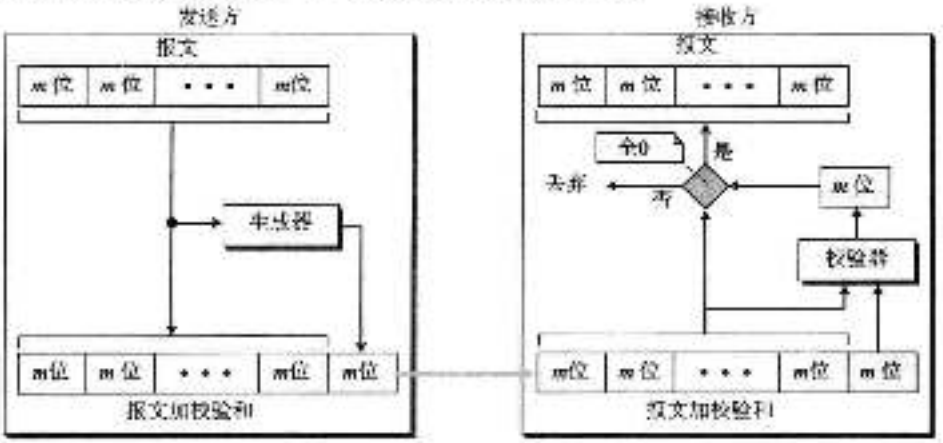


图 5-15　校验和

概念

传统的校验和的思想是很简单的。我们使用例子来说明这一点。

例 5.8　假设我们要发送到目的端的报文是 5 个 4 位数字的列表。除了发送这些数字之外，我们还发送这些数字的和。例如，如果这些数字的集合是(7, 11, 12, 0, 6)，我们发送(7, 11, 12, 0, 6, 36)，其中 36 是原始数字的和。接收方将 5 个数字相加，将其结果和此和相比较。如果两者相同，接收方就认为没有差错，接收 5 个数字并丢弃和。否则，就认为有差错，该报文就不被接收。

反码加法　前一个例子有一个主要的缺点。除了校验和外，每一个数字都能被写成一个 4 位的字（每一个数字都小于 15）。一种解决方案是使用反码（one's complement）算法。在该算法中，我们只使用 m 位来表示 0 到 2^m-1 之间的无符号数字。如果数字多于 m 位，那么最左边的额外的位需要加到最右边的 m 位（包装）。

例 5.9　在前面的例子中，十进制数字 36 的二进制表示为$(100100)_2$。为了将其变为 4 位数字，我们将最左边的额外的位加到右边的 4 位，如下所示。

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

不是发送 36 作为其校验和，而是发送 6 来代替，即(7, 11, 12, 0, 6, 6)。在反码算法中，接收方将前面的 5 个数字相加。如果结果是 6，这些数字被接收；否则，它们被拒绝。

校验和　如果我们发送校验和的反码，我们可以使得接收方的工作更加简单。在反码算法中，一个数字的反码通过将所有的位取反来实现（所有的 1 变为 0，所有的 0 变为 1）。这与从 $2^m - 1$ 减去这个数字相同。在反码算法中，我们有两个 0：一个正的和一个负的，它们互为反码。正 0 所有的 m 位全是 0；负 0 所有的位是 1（即 $2^m - 1$）。如果我们将一个数字和它的反码相加，我们会得到负 0（一个所有位全是 1 的数）。当接收方将所有 5 个数字（包括校验和）相加时，就得到一个负 0。接收方可以对结果求反来获得一个正 0。

例 5.10　让我们使用例 5.9 中校验和的思想。发送方将所有的以反码形式表示的 5 个数字相加得到和 sum = 6。发送方然后对结果求反得到检验和 checksum = 9（即 15-6）。注意 $6 = (0110)_2$，$9 = (1001)_2$；它们互为反码。发送方发送 5 个数据数字和校验和，即(7, 11, 12, 0, 6, 9)。如果传输过程中没有被破坏，接收方会接收到(7, 11, 12, 0, 6, 9)，并将它们按反码算法相加得到 15。

发送方对 15 求反得到 0。这就说明数据没有被破坏。图 5-16 说明了这一过程。

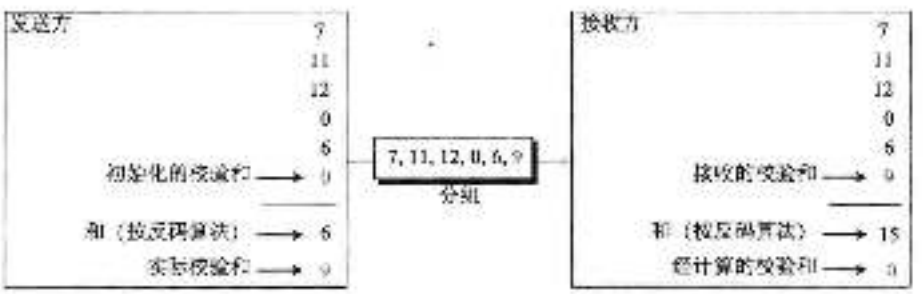


图 5-16　例 5.10

因特网校验和

传统上，因特网使用 16 位校验和。发送方和接收方遵循表 5-5 中所描述的步骤。发送方或者接收方使用 5 个步骤。

表 5-5　计算传统校验和的步骤

发　送　方	接　收　方
1. 报文被划分为 16 位的字	1. 报文和校验和接收
2. 校验和的值初始化为 0	2. 报文被分为 16 位的字
3. 包含校验和在内的所有字使用反码运算相加	3. 所有字使用反码运算相加
4. 对和求反，成为校验和	4. 对和求反，成为新的校验和
5. 校验和和数据一起发送	5. 如果校验和的值为 0，报文被接收；否则被拒绝

算法

我们可以使用图 5-17 的流程图来说明计算校验和的算法。基于该算法的使用任意语言编写的程序都可以很简单的实现。注意，第一个循环仅仅计算以补码表示的数据单元的和；第二个循环包含了由补码运算计算出的额外的位来模拟反码运算。因为现在几乎所有的计算机均做补码运算，因此这一点是必需的。

性能　传统的校验和使用较少的位（16）来检测任意长度的报文（有时数千位）中的差错。然

面，在差错检测能力上它没有 CRC 强。例如，如果一个字的值增加，另一个字减少相同的值，这两个差错就无法被检测，因为和以及校验和不变。同样，如果多个字的值增加，但是和以及校验和没有改变，这些差错也无法检测。Fletcher 和 Adler 建议使用一些带权值的校验和来解决第一个问题。但是因特网的趋势，特别是在设计新的协议时是使用 CRC 代替校验和的。

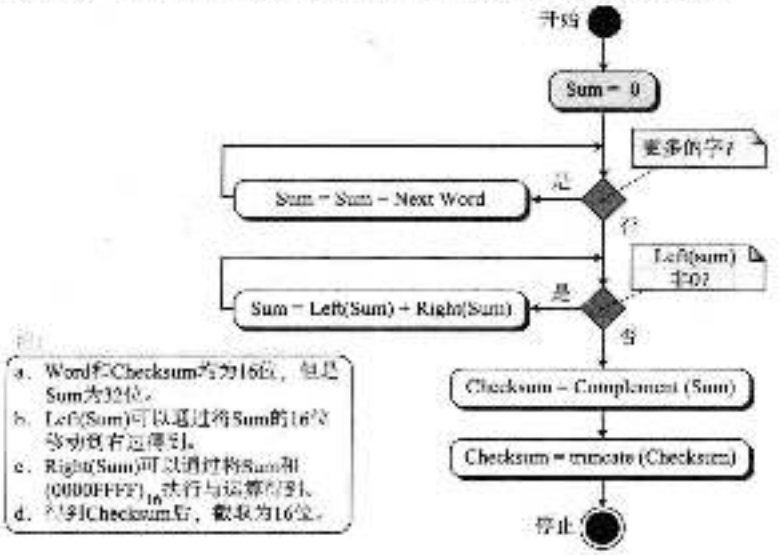


图 5-17　计算传统校验和的算法

计算校验和的其他方法

如前所述，传统的校验和计算有一个主要的问题。如果两个 16 位的项在传输过程中调换位置，则校验和就无法发现该差错。原因是传统的校验和是不含权值的：它公平对待每一个数据项。换言之，数据项的顺序对于计算来说是不重要的。好几种方法已经用来防止这一问题。这里我们讨论两种方法：Fletcher 和 Adler。

Fletcher 校验和　Fletcher 校验和根据数据项的位置对其加权。Fletcher 已经提出了两个算法：8 位和 16 位。8 位 Fletcher 根据 8 位的数据项计算生成 16 位的校验和。16 位的 Fletcher 根据 16 位的数据项计算生成 32 位的校验和。

8 位 Fletcher 是通过计算 8 位数据位组生成 16 位校验和。该计算是模 256（2^8）的运算，这意味着中间结果被 256 除，保留余数。该算法使用了两个累加器 L 和 R。首先简单地将数据项加在一起；然后对计算结果加权。8 位 Fletcher 算法有很多变种。我们在图 5-18 中说明了一个简单的算法。

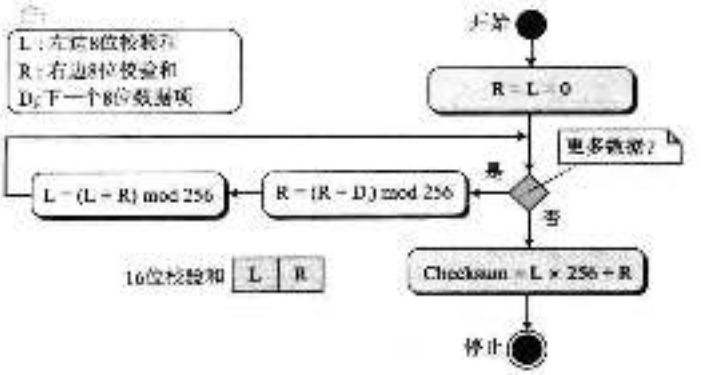


图 5-18　计算 8 位 Fletcher 校验和的算法

16 位 Fletcher 校验和与 8 位 Fletcher 校验和相似，但是它是通过计算 16 位数据项生成 32 位校验和。该计算是模 65 536 的。

Adler 校验和　Adler 校验和是 32 位校验和。图 5-19 以流程图的形式说明了一个简单的算法。它与 16 位 Fletcher 相似，但有 3 点不同。第一，计算对象为单字节而非一次两个字节。第二，模数是素数（65 521）而非 65 536。第三，L 被初始化为 1 而非 0。我们已经证明，一个素数基数在一些数据组合上有更好的检测能力。

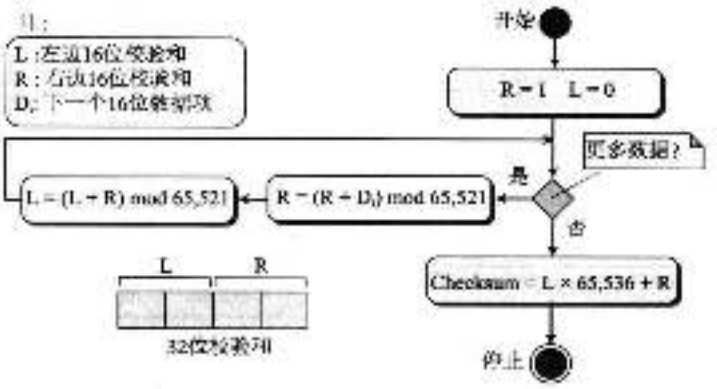


图 5-19　计算 Adler 校验和的算法

5.2.4　两种 DLC 协议

在结束与 DLC 子层相关的问题之后，我们讨论两种 DLC 协议，它们在实际中实现了那些概念。第一个是 HDLC，它是很多为局域网设计的协议的基础。第二个是 PPP，它来源于 HDLC，用于点对点链路。

HDLC

高级数据链路控制（High-level Data Link Control，HDLC）是一种面向位的协议，它用于点到点和多点链路的通信。它实现了我们在第 3 章中提到的停止等待（Stop-and-Wait）协议。

配置和传输模式

HDLC 提供了两种通用的传输模式，它们可以在不同的配置中使用：正常响应模式（normal response mode，NRM）和异步平衡模式（asynchronous balanced mode，ABM）。在正常响应模式中（NRM），站点配置是不平衡的。我们有一个主站（primary station）和多个从站（secondary stations）。主站可以发送指令；从站只能响应命令。NRM 可以在点到点和多点链路中使用，如图 5-20 所示。

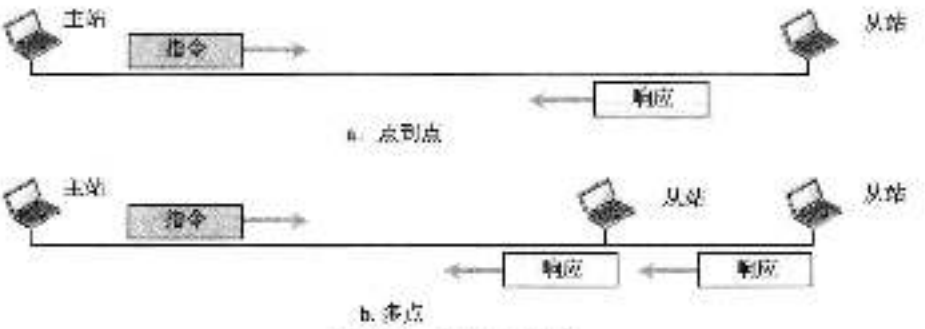


图 5-20　正常响应模式

在 ABM 中，配置是平衡的。链路是点到点的，每一个站点可以行使主站和从站的功能（作为对等点），如图 5-21 所示。这是现在的通用模式。



图 5-21　异步平衡模式

或

为了提供必要的灵活性来支持在刚刚描述的模式和配置中的所有可能的选项，HDLC 定义了 3 种类型的帧：信息帧（information frames，I-frames），管理帧（supervisory frames，S-frames）以及无编号帧（unnumbered frames，U-frames）。每一种类型的帧为不同类型的报文传输充当信封的角色。信息帧用来传输用户数据和与用户数据相关的控制信息（捎带）。管理帧只用来传输控制信息。无编号帧是为系统管理保留的。由无编号帧携带的信息用来管理链路本身。HDLC 中的每一个帧都可以包含多达 6 个域，如图 5-22 所示：开始标记域、地址域、控制域、信息域、帧校验序列域（FCS）和结束标记域。在多帧传输中，一个帧的结束标记可以充当下一个帧的开始标记。



图 5-22　HDLC 帧

现在让我们讨论各域以及它们在不同类型帧中的使用。

- 标记域。该域包含同步模式 01111110，它指明了一个帧的开始和结束。
- 地址域。该域包含从站的地址。如果主站创建该帧，它包含“去往”的地址。如果从站创建该帧，它包含一个“来自”的地址。该地址域可以是一个或者多个字节长度，这依赖于网络的需要。
- 控制域。控制域为一个或者两个字节长度，用来进行流量控制和差错控制。各个位的意义稍后讨论。
- 信息域。信息域包含了来自网络层的用户数据或者管理信息。它的长度因网络而异。
- 帧校验序列域。帧校验序列域是 HDLC 的差错检测域，它可以包含 2 字节或者 4 字节的 CRC。

控制域决定帧的类型，定义了它的功能。所以让我们详细地讨论一下该域的格式。该格式对于每种类型的帧都是特定的，如图 5-23 所示。



图 5-23　不同帧类型的控制域格式

信息帧的控制域　信息帧设计用来运载来自于网络层的用户数据。另外，它们可以包含流量控制和差错控制信息（捎带）。控制域中的子域用来定义这些功能。第一位定义了类型。如果控制域的第一位为 0，就意味着该帧是信息帧。接下来的 3 位称为 N（S），它们定义了该帧的序号。注意使用 3 位，我们可以定义在 0 和 7 之间的序号。最后 3 位称为 N（R），当捎带被使用时相当于确认号。

在 N（S）和 N（R）之间的单个位称为 P/F 位。P/F 域是带有双重用途的单个位。只有当它被设定时（位=1）才有含义，意味着轮询或终止。当帧由主站向从站发送时（此时地址域包含接收方的地址），它意味着轮询。当帧由从站向主站发送时（此时地址域包含发送方地址），它意味着终止。

管理帧的控制域　当捎带是不可能的、不恰当的时候，管理帧用来进行流量控制和差错控制。管理帧没有信息域。如果控制域的前两位是 10，意味着该帧是管理帧。最后 3 位称为 N(R)，相当于确认号（ACK）或者否定应答号（NAK），这依赖于管理帧的类型。有两位称为编码（code），用来定义管理帧本身类型。使用 2 位，我们可以有 4 种类型的管理帧，如下所述：

- 准备接收（RR）　如果编码子域的值为 00，该帧为 RR 管理帧，这种帧确认收到一个或者一组安全接收的帧。这时，N(R)域定义了确认号。
- 不准备接收（RNR）　如果编码子域的值为 10，该帧为 RNR 管理帧。这种帧是具有额外功能的 RR 帧。它确认收到一个或者一组安全接收的帧，并宣布接收方已忙不能接收更多的帧。它通过要求发送方降低发送速度来实现到压控制机制。N(R)的值是确认号。
- 拒收（REJ）　如果编码子域的值为 01，该帧为 REJ 管理帧。这是个 NAK 帧，但是与选择性重复 ARQ 中的不同。它是可以在回退 N 帧 ARQ 中使用的 NAK，能在发送方计时器到时前，通知发送方最后的帧丢失或是破坏了，以此改进了程序的效率。N(R)的值是否定确认号。
- 选择性拒收（SREJ）　如果编码子域的值是 11，该帧为 SREJ 管理帧。它是个 NAK 帧，用于选择性重复 ARQ 中。注意 HDLC 协议使用了术语选择性拒收（selective reject）而非选择性重复（selective repeat）。N(R)的值为否定确认号。

无编号帧的控制域　无编号帧用来在互联的设备之间交换会话管理和控制信息。不像管理帧，无编号帧包含信息域，但是它用来承载系统管理信息，而非用户数据。虽然与管理帧有相似之处，但是无编号帧交换的多数信息包含在控制域的编码中。无编号帧编码被分为两部分：P/F 位前的一个 2 位前缀和 P/F 位后的 3 位后缀。总之，这两部分（5 位）能够定义多达 32 种不同类型的无编号帧。

点到点协议（PPP）

点到点协议是最通用的协议之一是点到点协议（Point-to-Point Protocol，PPP）。现在成千上万的因特网用户需要使用 PPP 将他们的家庭电脑与因特网服务提供商的服务器连接起来。这些用户的大部分有一个传统的调制解调器，它们通过电话线连接到因特网，该电话线提供物理层的服务。但是为了控制和管理数据传输，需要在数据链路层有点对点的协议。PPP 是目前为止最为通用的协议。

服务

PPP 的设计者已经在 PPP 中包含了数项服务以使它适合点对点协议，但是忽略了一些传统的服务，这样会使它更加简单。

PPP 提供的服务　PPP 定义了在设备之间交换的帧的格式。它也定义了两台设备之间如何协商链路的建立和数据的转发。PPP 可以接收来自于多个网络层（不只 IP）的负载。协议中也提供了认证，但是这是可选的。PPP 的新版本称为多链路 PPP（Multilink PPP），提供多条链路的连接。PPP 的一个有趣的特点是它提供网络地址配置，当一个家庭用户需要一个临时网络地址连接到因特网时，这一点特别有用。

PPP 不提供的服务　PPP 不提供流量控制。发送方可以一个接一个地发送帧，而无需考虑会淹没接收方。PPP 对于差错控制有很简单的机制。CRC 域用来检测差错。如果帧被破坏，它便被丢弃；上层协议需要注意这个问题。缺少差错控制和序号编号可能导致数据包被乱序接收。在多点配置中，PPP 无法提供高级寻址机制来处理帧。

成帧

PPP 使用面向字符（或者面向字节）的帧。图 5-24 说明了一个 PPP 帧的格式。每个域的说明如下：

- 标记。一个 PPP 帧以位模式 01111110 的 1 字节标记开始和结束。

- 地址。协议中的地址域是一个常量，设置为 11111111（广播地址）。

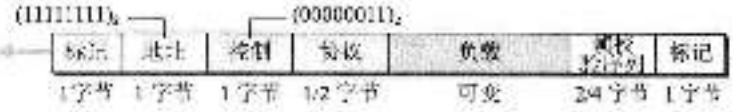


图 5-24　PPP 帧格式

- 控制。该域被设置为常量值 00000011（模仿 HDLC 中的无编号帧）。正如我们后面要讨论的，PPP 不提供任何的流量控制。差错控制仅限于差错检测。
- 协议。协议域定义了数据域中要承载的内容：或者是用户数据或者是其他信息。该域默认为 2 字节长，但是两端可以协商只使用 1 字节。
- 负载域。该域承载着用户数据或其他信息，我们将简短地讨论一下。数据域是一系列的字节，默认最大值为 1500 字节；但是通过协商可以改变。如果标记字节模式出现在数据域中，那么数据域是字节填充的。因为没有域定义数据域的长度，所以如果数据域长度小于数据域长度默认最大值或是协商的最大值，就需要数据填补。
- 帧校验序列。帧校验序列（FCS）是简单的 2 字节或者 4 字节的标准 CRC。

字节填充　因为 PPP 是面向字节的协议，PPP 中的标记是一个字节，而且只要它出现在帧的数据部分，这就需要被转义。转义字节为 01111101，这意味着类似标记模式的字节出现在数据部分，都要填充该额外字节来告诉接收方下一个字节不是标记。显然，转义字节本身应该使用另一个转义字节来填充。

转换阶段

一个 PPP 连接所经历的阶段，可以通过转换阶段（transition phase）图来说明（见图 5-25）。该转换图以闲置状态开始。在该状态时，没有活动的载体（在物理层），线路是静默的。当两个结点中的一个开启通信，连接进入建立状态。在该状态时，双方就选项进行协商，如果双方同意认证，系统就进入认证状态；否则，系统进入联网状态。稍后讨论的链路控制协议分组就用于此目的。一些分组可以在此交换。数据传输发生于打开状态。当一个连接进入此状态，数据分组的交换就可以开始了。该链路一直处于此状态，直到一个终端想要终止该连接为止。这种情况下，系统进入终止状态。系统一直停留在该状态，直至载体（物理层信号）释放为止，然后系统又进入闲置状态。

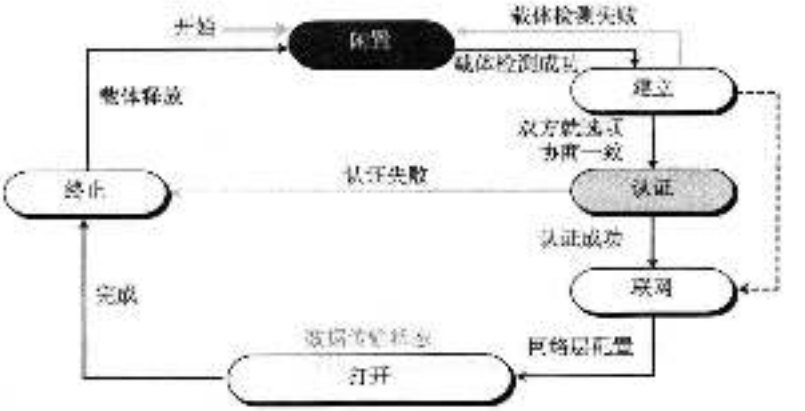


图 5-25　转换阶段

多路复用

尽管 PPP 是链路层协议，它使用另一组协议来建立链路，认证涉及的双方，承载网络层数据。

三组协议使得 PPP 更为强大：链路控制协议（Link Control Protocol，LCP），2 个认证协议（Authentication Protocol，AP）和几个网络控制协议（Network Control Protocol，NCP）。任意时刻，一个 PPP 分组能够在数据域里承载来自这些协议中的数据，如图 5-26 所示。注意，其中有一个 LCP、两个 AP 和几个 NCP。数据可能来自不同的网络层。

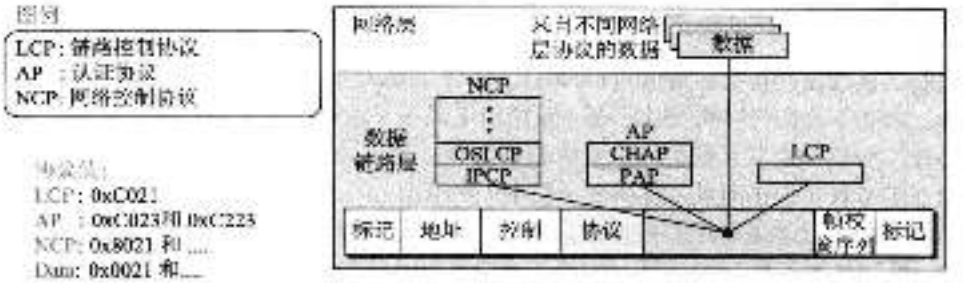


图 5-26　PPP 中的多路复用

链路控制协议　链路控制协议（Link Control Protocol，LCP）负责建立、维护、配置和终止链路。它也提供了协商机制以在两终端间设置选项。在链路建立之前，该链路的两个终端必须就选项达成一致。

认证协议　认证在 PPP 中扮演很重要的角色，这是因为 PPP 用于拨号链路中，该链路中用户身份的验证是十分必要的。用户需要访问一系列的资源，认证（Authentication）就意味着确认用户的身份。PPP 创建了两种认证协议：口令认证协议（Password Authentication Protocol）和查询握手认证协议（Challenge Handshake Authentication Protocol）。注意这些协议在认证阶段使用。

- PAP。口令认证协议（PAP）是有两个步骤的简单认证过程：

a. 想要访问系统的用户发送认证身份证明（通常是用户名）和口令。

b. 系统检查身份证明和口令的有效性，接受连接或者拒绝连接。

- CHAP。查询握手认证协议（CHAP）是一个三步握手认证协议，比 PAP 更具安全性。在这种方式下，口令是保密的，它从不在线上发送。

a. 系统向用户发送一个包含查询值（通常是一些字节）的查询分组。

b. 用户使用预定义的函数，根据查询值和用户口令生成一个结果。用户在发往系统的响应分组中发送结果。

c. 系统做相同的操作。它使用同样的函数，根据用户口令（系统已知）和查询值来创建结果。如果该结果和响应分组中发送的结果一样，允许访问；否则，禁止访问。CHAP 比 PAP 更加安全，特别如果系统不断地改变查询值时。甚至当入侵者得知该查询值和结果时，口令仍然是保密的。

网络控制协议　PPP 是多网络层协议。它可以承载来自因特网定义的协议（如 OSI、Xerox、DeCnet、AppleTalk、Novel 等）中的网络层数据。为此，PPP 为每一种网络层协议定义了一个特定的网络控制协议。例如，因特网协议控制协议（Internet Protocol Control Protocol，IPCP）用来配置承载 IP 数据分组的链路，Xerox CP 用来配置 Xerox 协议数据分组的链路，等等。注意没有一个 NCP 分组承载网络层数据；它们仅仅是在网络层为到来的数据配置链路。NCP 协议之一是因特网协议控制协议（Internet Protocol Control Protocol，IPCP）。该协议用来配置承载因特网中 IP 分组的链路。我们对 IPCP 特别感兴趣。

来自网络层的数据　由 NCP 协议中的一个完成网络层配置后，用户可以交换来自网络层的数据分组。再次重申，不同的网络层有不同的协议域。例如，如果 PPP 承载来自 IP 网络层的数据，那么该域值是$(0021)_{16}$。如果 PPP 承载来自于 OSI 网络层的数据，该协议域的值是$(0023)_{16}$，等等。

多链路 PPP

PPP 最初是为单通道点对点物理层链路设计的。单一点对点链路中多通道的可用性促进了多链路 PPP 的发展。在这种情况下，一个逻辑的 PPP 帧分为多个实际的 PPP 帧。该逻辑帧的一段在一个实际的 PPP 帧的负载中承载。如图 5-27 所示。为了说明实际的 PPP 帧正承载着一个逻辑 PPP 帧的一段，该协议域被设置为$(003d)_{16}$。这种新发展增加了复杂性。例如，需要一个序号添加到实际的 PPP 帧中来说明该分段在逻辑帧的位置。

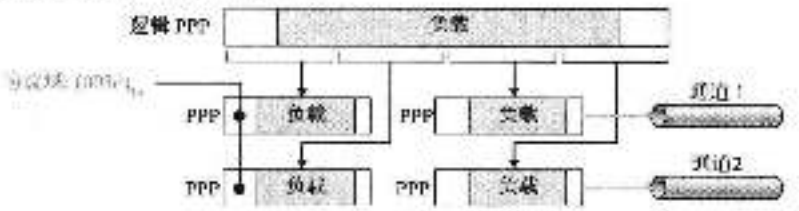


图 5-27　多链路 PPP

5.3　多路访问协议

我们曾经说过数据链路层被分为两个子层：数据链路控制（DLC）和介质访问控制（MAC）。我们在前一节中讨论了 DLC；我们在本节中讨论 MAC。当我们使用专用链路时，如拨号电话线，我们只需要诸如点对点协议（PPP）的数据链路控制协议。它们在两端间管理数据传输。另一方面，如果我们和其他用户共享有线的或者无线的介质，我们首先需要一个协议来管理共享过程，然后进行数据传输。例如，如果我们使用蜂窝电话来连接另一个蜂窝电话，该信道（分配给营方公司的波段）不是专用的。几步之外的人可能使用相同的波段和她的朋友聊天。

当结点或者站点使用称为多点链路或者广播链路的公共链路相互连接时，我们需要使用多路访问协议来协调链路访问。控制访问介质的问题和在集会上发言的规则相似。规则保证发言的权利并确认两个人不会同时发言，发言者不会打扰到彼此，不会独占发言权等。该情形和多点网络相似。我们需要确保每个结点能够获得链路访问。首要目标是阻止结点间的冲突。如果由于某种原因冲突发生了，第二个目标是处理该冲突。

很多协议已经设计用来处理访问共享链路。我们将其分为 3 组。属于每一组的协议如图 5-28 所示。



图 5-28　本章中讨论的多路访问协议的分类

5.3.1　随机访问

在随机访问（random-access）或者竞争（contention）访问方法中，没有站点优于其他站点，也没有站点能控制其他站点。每一次有数据要发送的站点使用由协议定义的程序来决定是否要发送。该决定依赖于介质的状态（空闲或忙碌）。换言之，只要遵循预定义的程序，包括介质状态的检测，满足条件的每一个站点都能传输数据。

该方法之所以如此命名是因为它的两个特性。第一，每一个站点的传输没有特定的时间表。站

点的传输是随机的。这就是该方法称为随机访问的原因。第二，没有规则来规定下一个将要发送的站点是哪一个。为了访问介质，站点展开竞争。这就是该方法称为竞争访问的原因。

在随机访问中，每一个站点都有权访问介质且不受控于其他站点。然而，如果多于一个站点尝试发送时，便会产生访问冲突碰撞（access conflict collision），那么便会有帧被破坏或修改。为了避免访问冲突或是在访问冲突发生时解决这个问题，每个站点遵循一个程序，该程序回答了以下问题：

- 站点何时能访问介质？
- 如果介质忙碌，站点能做什么？
- 站点如何确定传输的成功或者失败？
- 如果有访问冲突，站点能做什么？

本章中我们学习的随机访问方法是由一个非常有趣的称为 ALOHA 的协议发展来的，ALOHA 使用一个非常简单的称为多路访问（multiple access，MA）的程序。该方法是外加一个程序（就是站点在传输之前对介质进行侦听）改进而来的。这称为载波侦听多路访问（carrier sense multiple access，CSMA）。该方法后来演化成两个并行的方法：带冲突检测的载波侦听多路访问（carrier sense multiple access with collision detection，CSMA/CD），它告诉站点当检测到冲突该如何处理；带避免冲突的载波侦听多路访问（carrier sense multiple access with collision avoidance，CSMA/CA），它尝试避免冲突。

ALOHA

ALOHA 是最早的随机访问方法，在 20 世纪 70 年初由夏威夷大学开发出来。它用于无线电（无线）局域网，但是它可以用于任何的共享介质上。

很明显，在这种安排中存在潜在的冲突。介质由各站点共享。当一个站点发送数据时，另一个站点在同一时间也可能尝试发送。来自两个站点的数据就发生碰撞并相互混淆。

纯 ALOHA

原始的 ALOHA 协议称为纯 ALOHA。这是一个有效但是优雅的协议。它的思想是只要有站点有帧要发送，它就发送该帧（多路访问）。然而，只有一条通道共享，来自于不同站点的帧就有可能冲突碰撞。图 5-29 说明了在纯 ALOHA 中帧冲突的例子。

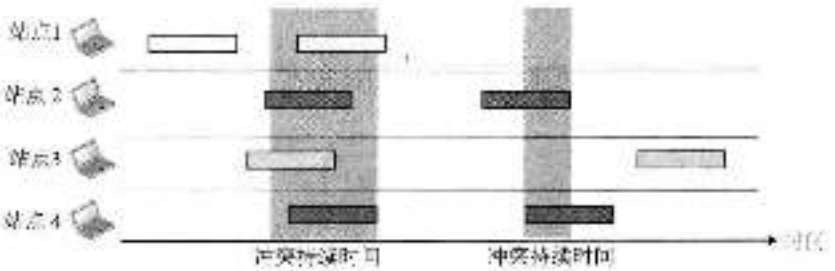


图 5-29　纯 ALOHA 网络中的帧

图中有 4 个站点（不现实的假设）为访问共享介质而竞争。图中每个站点发送两个帧；在共享介质上共有 8 个帧。因为多个帧为共享通道展开竞争，某些帧冲突了。图 5-29 中只有两个帧是成功的：一个帧来自于站点 1，一个帧来自于站点 3。必须提醒的是，即使一个帧的一位和另一个帧的一位同时存在于一个通道中，也会发生冲突且两个帧都会被破坏。很明显我们需要重发那些在传输过程中已经被破坏的帧。

纯 ALOHA 协议依赖于接收方的确认。当一个站点发送一个帧时，它期望接收方发送一个确认。如果超时后确认还未到达，站点就认为帧（或者确认）已经被破坏了，于是重发该帧。

一个冲突涉及两个或者更多的站点。如果超时后所有的这些站点都尝试重发帧，那么这些帧将会再次发生冲突。纯 ALOHA 指定超时后，每个站点在重发帧之前都会随机等待一段时间。该随机

件有助于避免更多的冲突。我们称这个时间为补偿时间 T_B（back-off time T_B）。

纯 ALOHA 有第二种方法来阻止重发帧造成的通道拥堵。一个站点在经过最大数目（K_{max}）的重传尝试后，它必须放弃重传并在以后再尝试。图 5-30 描述了基于上述策略的纯 ALOHA 进程。

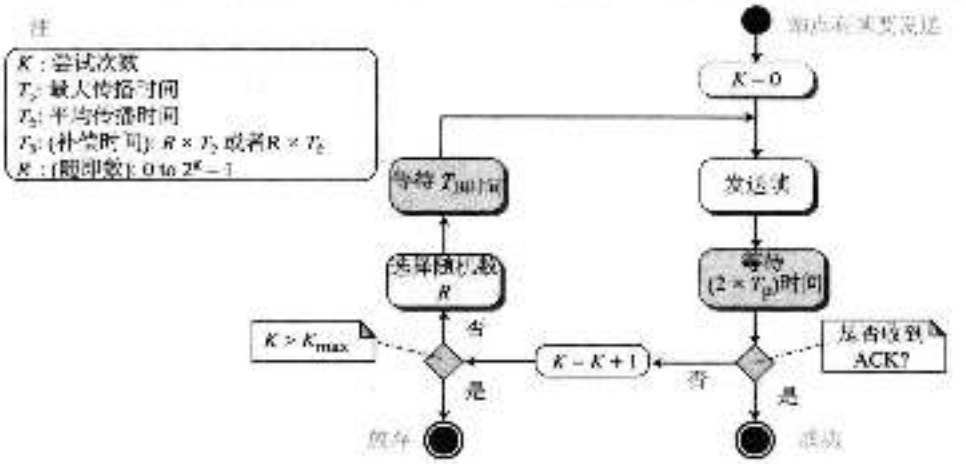


图 5-30　纯 ALOHA 协议进程

超时周期等于往返传输延迟的最大可能值，也就是两个最远距离站点之间发送一个帧所需时间的两倍（$2 \times T_p$）。补偿时间 T_B 是一个随机值，主要取决于 K（未成功的传输尝试次数）。T_B 的公式取决于实现。一个通用公式是二进制指数回送（binary exponential back-off）。在这种方法中，每一次重传随机选择乘数 R（0 到 $2^K - 1$），然后乘以 T_p（最大传播时间）或者 T_t（一个帧的平均传输时间）以得到 T_B。注意在这个步骤中，每一次冲突后随机数的范围便扩大了。K_{max} 的常用值为 15。

例 5.11　一个无线 ALOHA 网络的站点之间最远距离是 600 公里。如果我们假设信号传播速度为 3×10^8 m/s，那么 $T_p = (600 \times 10^3) / (3 \times 10^8) = 2$ ms。对于 $K = 2$，R 的取值范围是 {0, 1, 2, 3}。这意味着 T_B 可能是 0、2、4 或者 6 ms，这取决于随机变量 R 的结果。

脆弱时间　我们计算一下脆弱时间（vulnerable time）的长度，即可能发生冲突的时间长度。我们假设站点发送固定长度的帧，且每个帧的发送时间是 T_t 秒。图 5-31 显示了站点 B 的脆弱时间。

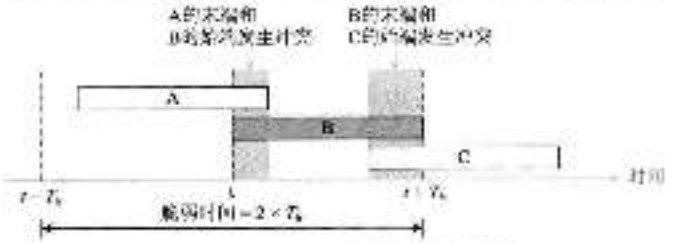


图 5-31　纯 ALOHA 协议的脆弱时间

站点 B 在时间 t 开始发送一个帧。现在设想站点 A 已经在 $t - T_t$ 时刻后开始发送帧。这就导致了来自站点 B 和站点 A 的帧发生冲突。另一方面，假如站点 C 开始在时间 $t - T_t$ 前开始发送帧。站点 B 和站点 C 之间的帧同样发生冲突。

在图 5-31 中，我们发现在纯 ALOHA 中冲突可能发生的脆弱时间是 2 乘以帧传输时间。

$$\text{纯 ALOHA 脆弱时间} = 2 \times T_t$$

例 5.12　一个纯 ALOHA 网络在一个传输速率为 200kbps 的共享通道上传输 200 位的帧。使这个帧免于冲突需要满足什么条件？

解答

平均帧传输时间 T_b 是 200 bit/200 kbps，即 1 ms。脆弱时间为 2 × 1 ms = 2 ms。这就意味着在该站点开始传输前 1ms，没有站点会开始传输，并且当该站点正在传输的 1ms 期间也没有站点会开始传输。

吞吐量　我们称 G 为一个帧传输时间内系统产生帧的平均数目。能证明纯 ALOHA 中成功传输帧的平均数目是 $S = G \times e^{-2G}$。当 $G = 1/2$ 时，最大吞吐量 S_{max} 是 0.184（我们可以通过计算 G 变化到 0 的过程中，S 的一系列值来得到该最大值，见练习题）。换言之，如果在一个帧的传输时间内产生半个帧（两个帧传输时间中产生一个帧），那么 18.4%的帧可以成功到达目的地。因为脆弱时间是帧传输时间的 2 倍，故我们可以预期 $G = 1/2$。因此，如果一个站点在脆弱时间内只产生一个帧（并且没有其他站点在此时间内产生帧），该帧将会成功到达目的地。

纯 ALOHA 的吞吐量是 $S = G \times e^{-2G}$。

当 $G = (1/2)$ 时，最大吞吐量 $S_{max} = 1/2e = 0.184$。

例 5.13　一个纯 ALOHA 网络每传输速度为 200 kbps 的共享通道上传输 200 位的帧。如果系统（包括所有站点）产生以下数量的帧，吞吐量是多少？

a. 每秒 1000 个帧。

b. 每秒 500 个帧。

c. 每秒 250 个帧。

解答

帧传输时间为 200/200 kbps，即 1 ms。

a. 如果系统每秒产生 1000 个帧，也就是说 1ms 产生 1 个帧，则 $G = 1$。这种情况下 $S = G \times e^{-2G} = 0.135$ (13.5%)。这意味着吞吐量是 1000 × 0.135 = 135 帧。1000 个帧中只有 135 个帧可能成功。

b. 如果系统每秒产生 500 个帧，即 1ms 产生 1/2 帧，即 $G = 1/2$。这种情况下 $S = G \times e^{-2G} = 0.184$ (18.4%)。这意味着吞吐量是 500 × 0.184 = 92，500 个帧中只有 92 个帧才可能成功。注意这就是最大吞吐量百分比的例子。

c. 如果系统每秒产生 250 个帧，即 1ms 产生 1/4 帧，则 $G = 1/4$。这种情况下 $S = G \times e^{-2G} = 0.152$ (15.2%)。这意味着吞吐量是 250 × 0.152 = 38，250 个帧中只有 38 个帧可能成功。

时隙 ALOHA

纯 ALOHA 的脆弱时间为 $2 \times T_b$。这是因为没有规则来定义何时站点可以发送帧。一个站点可能在另一个站点已经开始发送不久后发送，也可能在另一个站点完成发送之前发送。时隙 ALOHA 用来提高纯 ALOHA 的效率。

在时隙 ALOHA（slotted ALOHA）中，我们将时间分为 T_b 秒的时隙，并且强制站点在时隙开始之时才能发送。图 5-32 描述了时隙 ALOHA 中帧冲突的例子。

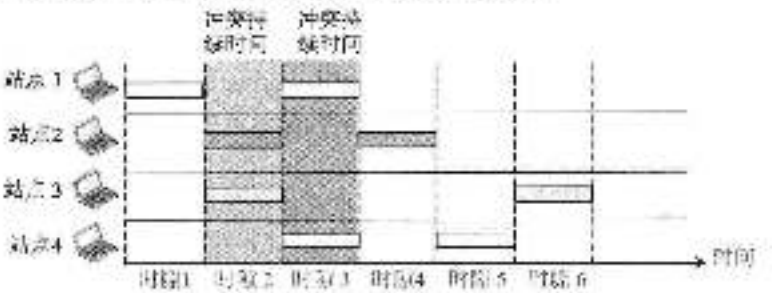


图 5-32　时隙 ALOHA 网络中的帧

因为站点只允许在同步时隙的开始处发送帧，如果一个站点错过了这个时刻，它必须等到下一个时隙的开始。这就意味着在该时隙开始之时发送帧的站点就已经完成了发送。当然如果两个站点在同一时隙的开始尝试发送帧，仍然会有冲突的可能性。然而，脆弱时间减少了一半，和 T_b 相等。图 5-33 说明了这种情况。

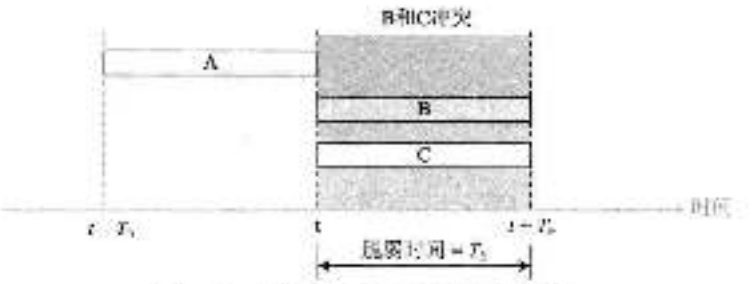


图 5-33　时隙 ALOHA 协议的脆弱时间

吞吐量　可以证明，时隙 ALOHA 成功传输帧的平均数量是 $S = G \times e^{-G}$。当 G=1 时，最大吞吐量 S_{max} 是 0.368。换言之，如果在一个帧的传输时间内产生一个帧，36.8%的帧会成功到达它们的目的地。因为脆弱时间和一个帧的传输时间相等，故 $G = 1$ 产生最大吞吐量是可以预知的。因此，如果一个站点在脆弱时间里只产生一个帧（并且没有其他站点在此时间内产生帧），该帧就会成功到达它的目的地。

对纯 ALOHA 的吞吐量是 $S = G \times e^{-2G}$。
当 G=1 时，最大吞吐量 $S_{max} = 0.368$。

例 5.14　一个时隙 ALOHA 网络使用一个带宽为 200kbps 的共享通道传输 200 位的帧。如果系统（包括所有站点）产生以下数量的帧，吞吐量将是多少？

a. 每秒 1000 个帧。

b. 每秒 500 个帧。

c. 每秒 250 个帧。

解答

除了使用时隙 ALOHA 代替纯 ALOHA 外，该练习和前一个习题的情况相似。帧传输时间是 200/200 kbps，即 1 ms。

a. 此例中 G=1，因此 $S = G \times e^{-G} = 0.368$ (36.8%)。这就意味着吞吐量为 $1000 \times 0.368 = 368$ 帧。1000 个帧中只有 368 个可能成功。注意这是最大吞吐量百分比的例子。

b. 此例中 G=1/2，这种情况下 $S = G \times e^{-G} = 0.303$ (30.3%)。这就意味着吞吐量是 $500 \times 0.303 = 151$。500 个帧中只有 151 个可能成功。

c. 此例中 G=1/4，这种情况下 $S = G \times e^{-G} = 0.195$ (19.5%)。这就意味着吞吐量是 $250 \times 0.195 = 49$。250 个帧中只有 49 个可能成功。

载波侦听多路访问（CSMA）

为了降低冲突发生的概率从而提高性能，采用 CSMA。如果一个站点在尝试发送前检测介质，冲突的概率就会降低。载波侦听多路访问（Carrier sense multiple access，CSMA）要求每一个站点在发送前首先要监听介质（或者检测介质状态）。换言之，CSMA 基于“传输前先监听”或者“说话前先听”的原则。

CSMA 能够降低冲突的概率，但是不能消除冲突。其原因如图 5-34 所示，一个 CSMA 网络的

空间和时间模型。站点通过共享通道（通常是专用介质）相互连接。

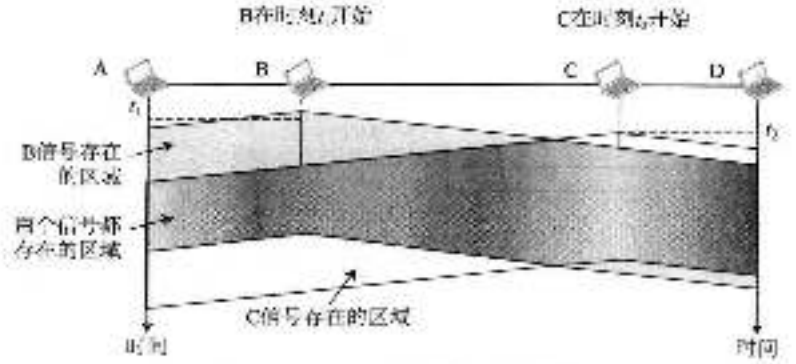


图 5-34　CSMA 中的空间/时间模型

由于广播延迟的存在，冲突的可能性仍然存在：当一个站点发送一个帧时，它的第一个位到达每一个站点和每个站点检查到它的信号需要一定时间（尽管时间很短）。换言之，仅仅因为一个站点发送的第一位还没有到达，另一个站点就可能检测介质并发现它处于空闲状态。

在时刻 t_1，站点 B 检查介质，发现它空闲，所以 B 发送一个帧。在时刻 t_2（$t_2>t_1$），站点 C 检查介质，发现它处于空闲状态。因为此时来自于 B 的第一个位还没有到达站点 C。站点 C 也发送一个帧。这两个信号发生冲突，两个帧都被破坏了。

脆弱时间

CSMA 的脆弱时间是传播时间 T_p。这是一个信号从介质的末端到另一端所需的传播时间。当一个站点发送一个帧而其他站点在这段时间中试着发送帧时，冲突将会发生。但是如果该帧的第一位到达介质的末端，每一个站点都能侦听到这个位，并停止发送。图 5-35 显示了最坏的情况。最左边的站点 A 在时刻 t_1 发送一个帧，它在时刻 t_1+T_p 到达最右边的站点 D。深灰色区域显示了时间和空间上的脆弱区域。

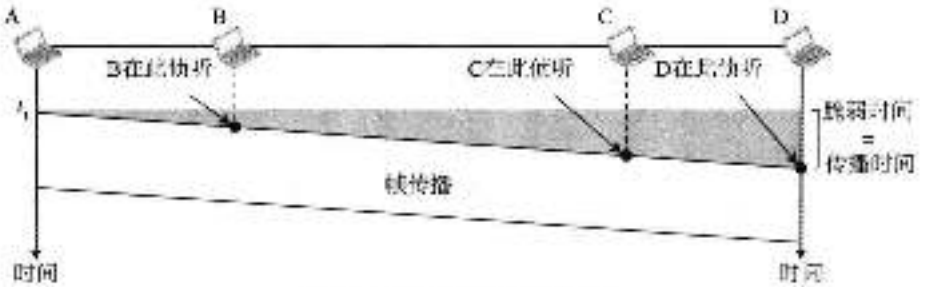


图 5-35　CSMA 中的脆弱时间

持续方法

如果信道繁忙，站点将如何处理？如果通道空闲，站点又将做什么？用三种方法来回答这些问题：**1-持续方法**、**非持续方法**和 **p-持续方法**。图 5-36 描述了当站点发现通道繁忙时三种持续方法的表现。

1-持续　1-持续方法（1-persistent method）简单直接。在该方法中，在站点发现线路空闲后，它立即发送帧（概率是 1）。该方法发生冲突的概率最高，因为两个或者更多的站点可能发现线路空闲并立即发送它们的帧。我们稍后将看到以太网使用这种方法。

非持续　在非持续方法（nonpersistent method）中，要发送帧的站点监听线路。如果线路是空

闲的，它立即发送。如果线路不是空闲的，它等待一个随机时间，然后再一次侦听线路。非持续方法降低了冲突的概率，因为两个或者更多的站点等待相同的时间，然后同时重新发送是不太可能的。然而该方法降低了网络的效率，因为当有站点要发送帧时，介质仍然可能是空闲的。

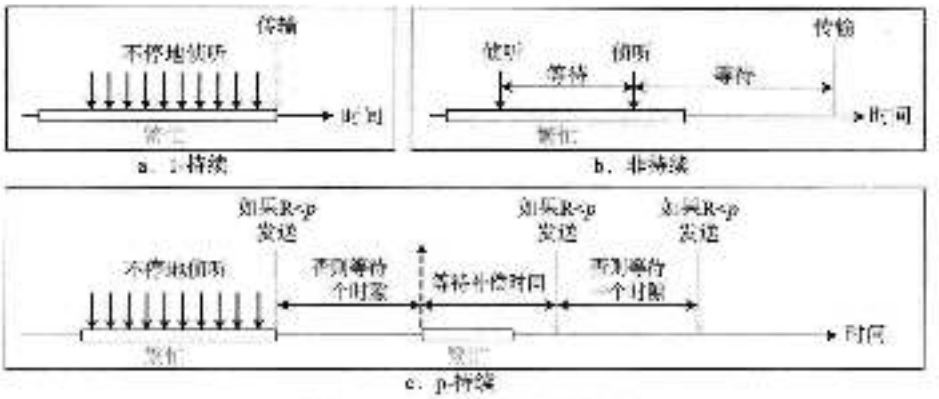


图 5-36　三种持续方法的表现

图 5-37 描述了这些方法的流程图。

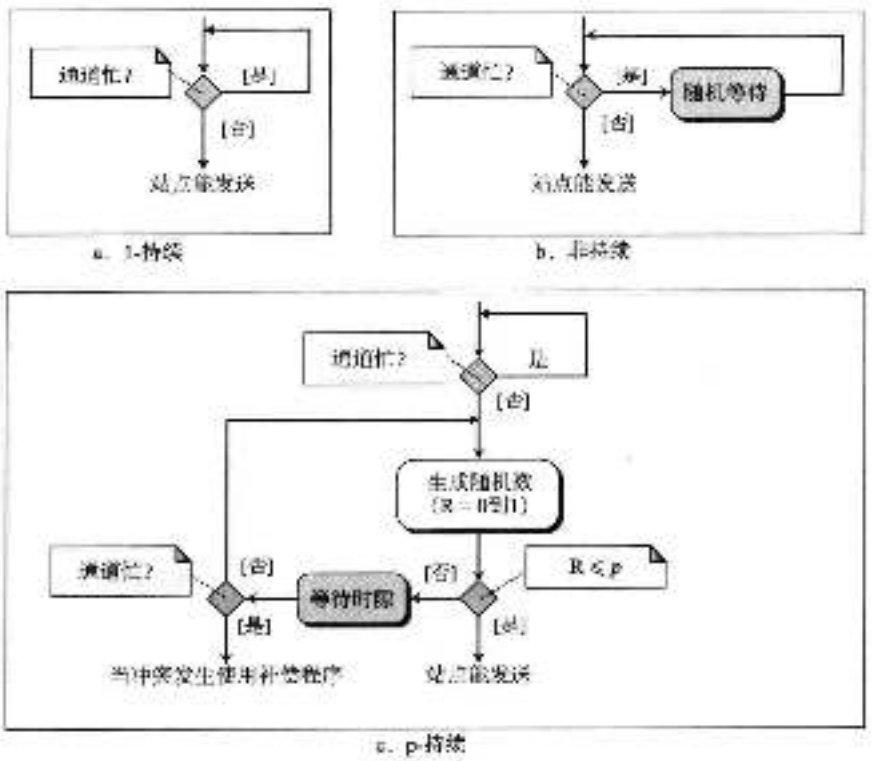


图 5-37　三种持续方法流程图

p-持续　如果通道有时隙且时隙同期等于或者大于最大传播时间，使用 p-持续方法。p-持续方法结合了其他两种策略的优点。它降低了冲突的概率，提高了效率。这种方法中，站点发现线路空闲时，它采取以下步骤：

1. 站点以概率 p 发送帧。
2. 站点以概率 $q=1-p$ 等待下一个时隙的开始并再一次检查线路。

a. 如果线路空闲，转步骤 1。

b. 如果线路忙，它会当做一个冲突已经发生并使用补偿程序。

CSMA/CD

CSMA 方法没有规定冲突之后的处理程序。带冲突检测的载波监听多路访问（carrier sense multiple access with collision detection，CSMA/CD）提出了处理冲突的算法。

在该方法中，站点在发送帧后监视介质来查看传输是否成功。如果成功，站点完成发送。否则，说明存在冲突，需要重新发送此帧。

为了更好地理解 CSMA/CD，我们查看在冲突中涉及的两个站点传输的第一个位的情况。尽管每个站点在检测到冲突之前继续发送帧中的其他位，我们说明当第一个位冲突时会发生什么。在图 5-38 中，站点 A 和 C 发生冲突。

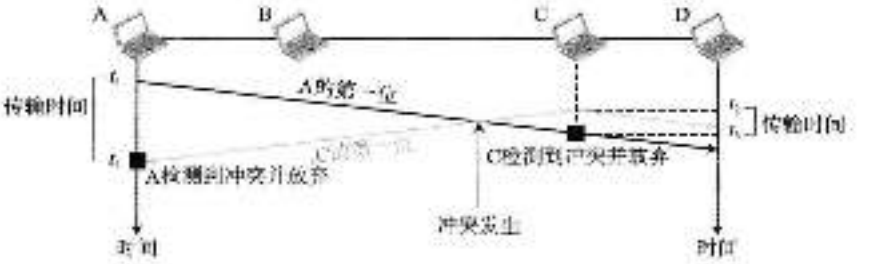


图 5-38　CSMA/CD 中第一个位冲突的情况

在时刻 t_1，站点 A 执行它的持续程序并且开始发送帧。在时刻 t_2，站点 C 没有侦听到 A 发送的第一个位。站点 C 执行它的持续程序并开始发送帧，该帧即向左传播也向右传播。在时刻 t_2 后的某个时刻冲突发生了。在时刻 t_3，当站点 C 接收到 A 发送帧的第一个位时，它检测到冲突。站点 C 立即（或者稍后，但是我们假设立即）放弃传输。在时刻 t_4，当站点 A 接收到 C 发送帧的第一个位时，它检测到冲突；它也立即放弃传输。由图可见，站点 A 在 t_4-t_1 时间段内传输；C 在 t_3-t_2 时间段内传输。

现在我们知道两个传输过程的持续时间。在图 5-39 中展示了更加完整的图。

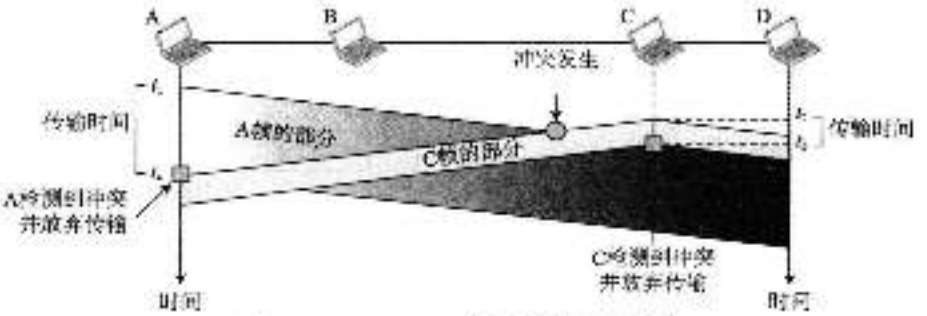


图 5-39　CSMA/CD 中的冲突和放弃传输

帧最小长度

为了实现 CSMA/CD 工作，我们需要限制帧长度。在发送帧的最后一个位之前，发送站点必须检测冲突，如有任何冲突就需要放弃传输。这是因为一旦整个的帧发送以后，站点不会保留该帧的副本，不会侦听线路检测冲突。因此，帧传输时间 T_f 必须至少是最大传播时间 T_p 的两倍。为了理

解该原因，让我们考虑最坏情况。如果冲突涉及的两个站点是距离最远的，来自第一个站点的信号用了 T_p 时间到达第二个，冲突的作用使得到达第一个站点还需要花费时间 T_p。所以要求第一个站点必须在 $2T_p$ 时间后传输。

例 5.15　使用 CSMA/CD 的一个网络，其带宽为 10Mbps。如果最大传播时间（包括设备延迟，忽略发送一个干扰信号所需的时间）是 25.6 μs，那么帧的最小长度是多少？

解答

最小帧传输时间是 $T_b = 2 \times T_p = 51.2$ μs。这意味着，在最坏情况下，一个站点需要 51.2 μs 的时间才能检测到冲突。帧的最小长度是 10 Mbps × 51.2 μs = 512 位或者 64 字节。这是标准以太网中帧的最小长度，我们将会在后面章节中介绍。

程序

现在看一下图 5-40 中 CSMA/CD 的流程图。它和 ALOHA 协议中的流程图类似，但存在着不同。

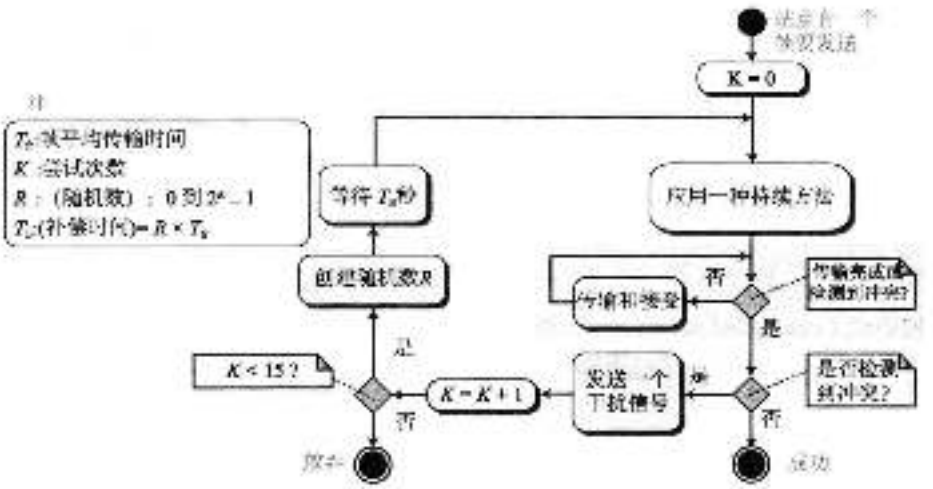


图 5-40　CSMA/CD 的流程图

第一个不同点是增加了持续过程。在我们开始发送帧之前，需要使用我们先前讨论的持续程序来侦听通道（非持续、1-持续或 p-持续）。图 5-37 中相关的图形将被一种持续程序代替。

第二个不同点是帧的传输。在 ALOHA 中，我们首先传输整个帧，然后等待确认。在 CSMA/CD 中，传输和冲突检测是一个连续过程。我们并非先发送整个帧，然后才检测冲突。站点的传输和接收是连续的和同步的（使用两个不同的端口或双向端口）。我们使用循环来描述传输这个连续过程。为了检测出以下两个条件之一，我们不断地监听：传输完成或者冲突被检测到。这两个事件都可以终止传输。当我们跳出循环时，如果冲突没有被检测到，就意味着传输已经完成了；整个帧被传输，否则，便有冲突发生了。

第三个不同是发送一个短的干扰信号（jamming signal）以确定所有的其他站点发觉该冲突。

能量级别

我们可以认为一个通道的能量级别有三种值：0、正常和不正常。在 0 级别，通道是空闲的。在正常级别，站点成功地获取通道并发送帧。在不正常级别发生冲突，能量的级别是正常的两倍。要发送帧的站点或正发送帧的站点需要监听能量级别以决定通道是空闲、忙碌还是冲突状态。图 5-41 说明了这种情况。

吞吐量

CSMA/CD 的吞吐量比纯 ALOHA 或者时隙 ALOHA 的要大。基于不同的持续方法及 p-持续中

不同的 p 值，不同的 G 值将有不同的最大吞吐量。对于 1-持续方法，当 $G=1$ 时，最大吞吐量在 50% 左右。对于非持续方法，当 G 在 3 到 8 之间时，最大吞吐量可以达到 90%。

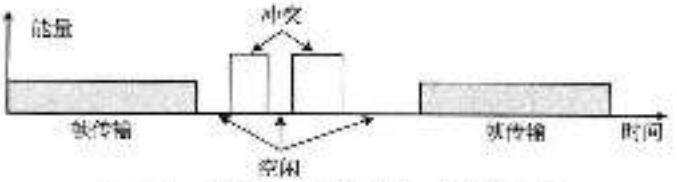


图 5-41　在传输、空闲或冲突时的能量级别

传统以太网

局域网协议中使用 CSMA/CD 技术的协议之一是传统以太网，它的数据传输率为 10Mbps。我们稍后在本章中讨论以太网局域网，传统局域网是一种广播局域网，它使用 1-持续方法来控制公共介质的访问，明确这些对我们有很多好处。以太网的后续版本试着放弃 CSMA/CD 访问方法，原因我们将在讨论局域网时讨论。

CSMA/CA

CSMA 方法的变种是带冲突避免的载波侦听多路访问（carrier sense multiple access with collision avoidance，CSMA/CA），它在无线局域网中使用。我们将其推迟到第 6 章讲解，该章主要讨论这一主题。

5.3.2　受控访问

在受控访问（controlled access）中，站点之间相互协商以确定哪一个站点有权发送帧。如果一个站点没有其他站点授权，该站点不能发送帧。我们讨论 3 种受控访问方法。

预约

在预约方法中，站点在发送数据前需要先预约。时间被分为时隙。在每个时隙中，在数据帧之前先发送一个预约帧。

如果系统中有 N 个站点，在预约帧中就恰有 N 个预约子时隙。每个子时隙属于一个站点。当站点需要发送数据帧时，它在自己的子时隙中预约。已经预定的站点在预约帧后发送数据帧。

图 5-42 描述了有 5 个站点和 5 个子时隙的预约帧情形。在第一个间隙中，只有站点 1，3，4 做了预约。在第二个间隙中，只有站点 1 做了预约。

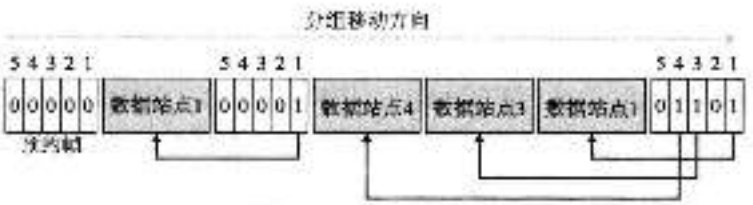


图 5-42　预约访问方法

轮询方法

轮询（polling）方法的拓扑结构是一个设备做主站（primary station），其他的设备是从站（secondary station）。所有的数据交换必须通过主站设备，即便最终目的地是从站。主站设备控制连接；从站遵循它的指令。由主站决定哪一个设备在给定的时间内使用通道。因此主站设备总是会话的发起者（如图 5-43 所示）。该方法使用轮询和选择来预防冲突。但是，缺点是如果主站出现问题，系统就崩溃了。

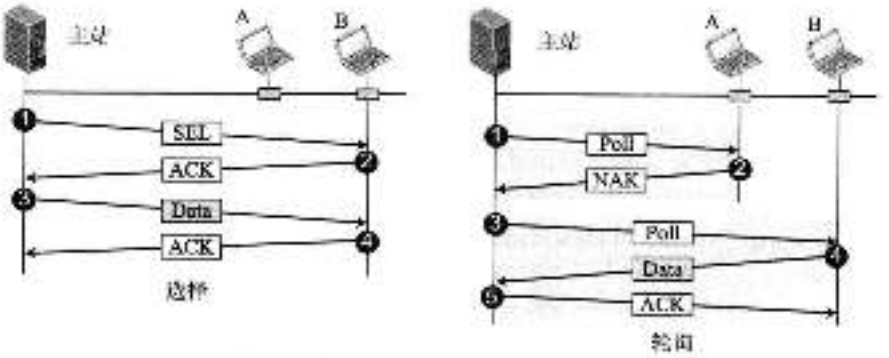


图 5-43　轮询访问方法中选择和轮询功能

选择

当主站设备要发送数据时使用选择（select）功能。记住主站设备控制链路。如果主站既不发送也不接收数据，那么它知道链路是可用的。如果它要发送数据，主站设备就可以发送。它不知道的是目标设备是否准备好接收。所以主站必须通知从站即将来临的传输，并等待从站设备准备状态的确认。在发送数据之前，主站创建和传输一个选择帧（SEL 帧），该帧的一个域包含了目标从站设备的地址。

轮询

主站设备使用轮询功能来向从站设备请求传输。当主站准备接收数据时，它必须依次询问每个设备是否要发送数据。当询问到达第一个从站时，如果它没有数据要发送，就用一个 NAK 帧来回应，如果有，就用数据回应（以数据帧的形式）。如果回应是否认（一个 NAK 帧），然后主站用同样方法轮询下一个从站，直至找到有数据发送的从站。当回应是肯定的（一个数据帧），基站就读取该帧，并返回一个确认（ACK 帧），表示已经接受。

令牌传递

在令牌传递（token-passing）方法中，网络中的站点组织为一个逻辑环。换言之，对每个站点，都有一个前驱（predecessor）和一个后继（successor）。前驱是环中逻辑上在该站点之前的站点；后继是环中该站点之后的站点。当前的站点正在访问通道。访问权由前驱结点传递给当前站点。当前站点没有数据发送时，访问权传递给后继结点。

访问通道的权利是如何从一个站点传递到另一个站点的？在该方法中，一个称为令牌（token）的特殊的信息分组在环中循环。拥有令牌的站点有权力访问通道和发送数据。当站点要发送数据时，它等到从前驱获取令牌为止。然后它持有令牌发送数据。当站点没有数据要发送了，它释放该令牌，将其传递给环中下一个逻辑站点。该站点直到在下一次循环中接收到令牌才能再次发送数据。在这一过程中，当一个站点接收到令牌，但没有数据要发送时，它就将令牌传递给下一个站点。

在这种访问方法中，需要进行令牌管理。站点获得令牌的时间必须是受限制的。令牌必须被监控以确保它没有丢失或者被破坏。例如，如果获得令牌的结点失败了，令牌就会从网络中消失。令牌管理的另一个功能是指定站点的优先权以及被传输的数据类型。最后，令牌管理需要使低优先级的站点将令牌传递给高优先级的站点。

逻辑环

在令牌传递网络中，站点不必物理上连接成环；该环可以是逻辑的。图 5-44 描述了 4 种不同的能创建逻辑环的物理拓扑结构。

在物理环拓扑结构中，当一个站点发送令牌到后继站点时，该令牌不能被其他站点看到；后继

站点是线路上的下一个。这意味着令牌不用获取下一个后继的地址。该拓扑的问题是如果链路中的一个（两个临近站点间的介质）失效，整个系统就失效了。

二重环拓扑使用一个二级（辅助）环，该二级环的运行方向和主环正好相反。二级环只用于紧急情况（就像汽车备用胎）。如果主环中链路之一失效了，系统自动将两个环合并成一个临时环。当失效的链路恢复后，辅助环又进入空闲状态。注意这种拓扑要工作，每个站点需要有两个传输端口和两个接收端口。称为光纤分布式数据接口（Fiber Distributed Data Interface，FDDI）和铜线分布式数据接口（Copper Distributed Data Interface，CDDI）的高速令牌环网络就使用这种拓扑结构。

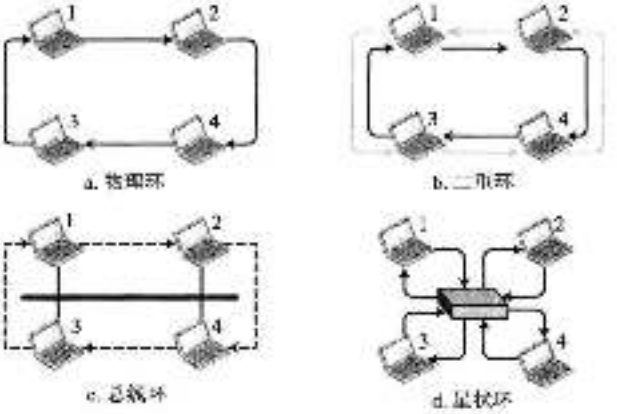


图 5-44　令牌传递访问方法中的逻辑环和物理拓扑

总线环拓扑也称作令牌总线。在这种拓扑中，站点连接到称为总线的单根光纤连接。然而，它们也组成了一个逻辑环，因为每一个站点知道它的后继（为了令牌管理的目的，它也知道前驱的地址）的地址。当站点结束发送数据时，它释放令牌并将后继的地址插入令牌中。只有和令牌目的地址相匹配的站点才能访问共享介质。IEEE 标准化的令牌总线局域网就使用这种拓扑结构。

在星状环拓扑结构中，物理拓扑结构是星状的。有一个集线器充当连接器。集线器中线路形成了环；站点通过两根电线连接到环。这种拓扑结构使网络不易发生故障，因为如果一条链路失效了，集线器可以忽略它，剩余的站点还可以运行。而且从环中增加或者移除站点很简单。IBM 设计的令牌环局域网仍在使用这种拓扑结构。

5.3.3　通道化

通道化（channelization）有时也称为通道划分（channel partition），是一种多路访问方法，在该方法中，不同的站点之间可以通过时间、频率或编码共享链路的可用带宽。由于这些方法通常在无线网络中使用，我们将其推迟到第 6 章讨论。

5.4　链路层寻址

下一个我们要讨论的关于链路层的主题是链路层地址。在第 4 章中，我们讨论过 IP 地址是网络层的标识符，它准确地定义了源主机和目的主机互联的因特网中的一点。但是，在无连接的像因特网这样的互联网络中，我们无法只使用 IP 地址就使数据报到达目的地。原因是因特网中来自同一个源主机到达同一个目的主机的数据报可能经过不同的路径。源端 IP 地址和目的端 IP 地址定义了两个终端但是没有定义数据报应该经过哪条链路。

我们需要记住数据报中的 IP 地址不能改变。如果数据报中的目的 IP 地址改变了，分组就无法到达目的地。如果数据报中源 IP 地址改变，那么当有响应需要回复或者有差错需要向源端主机报告时（见第 4 章中 ICMP），目的端主机或者路由器就无法与源端主机交互。

上面的讨论说明在无连接的互联网络中我们需要另一种寻址机制：两个结点的链路层地址。链路层地址（link-layer address）有时称为链路地址（link address），有时称为物理层地址（physical address），有时称为 MAC 地址。本书中我们交替地使用这些术语。

既然链路在数据链路层控制，那么其地址也属于数据链路层。当数据报从网络层传递到数据链路层时，数据报被封装为帧，两个数据链路层地址添加到帧头部。每当帧从一个链路移动到另一个时，这两个地址就要发生变化。图 5-45 在一个小型互联网络中说明了这一点。

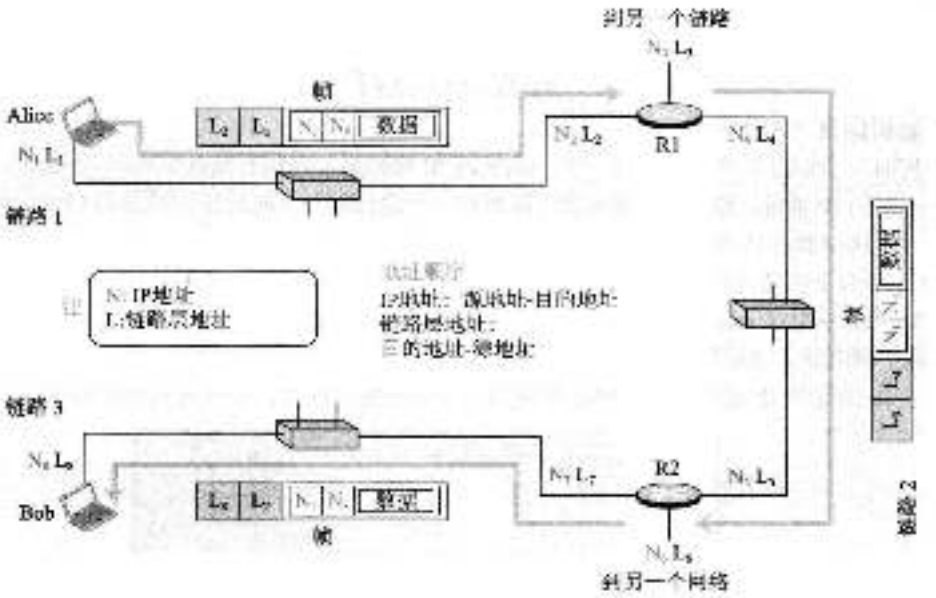


图 5-45　小型互联网络中的 IP 地址和链路层地址

在图 5-45 的互联网络中，我们有 3 条链路和两个路由器。我们也只展示了两个主机：Alice（源端主机）和 Bob（目的端主机）。对于每一个主机，我们展示两种地址，IP 地址（N）和链路层地址（L）。注意路由器地址对的数目与它连接的链路的数目相等。我们描述了 3 个帧，每个链路一个帧。每一个帧运载相同的数据报，它们有相同的源地址和目的地址（N_1 和 N_8），但是帧的链路层地址随着链路变化而变化。在链路 1，链路层地址是 L_1 和 L_2。在链路 2，是 L_4 和 L_5。在链路 3，是 L_7 和 L_8。注意 IP 地址和链路层地址顺序是不同的。对于 IP 地址，源地址在目的地址之前；对于链路层地址，目的地址在源地址之前。数据报到帧被设计成这种方式，我们要遵循这种设计。我们可以提几个问题：

- 如果路由器的 IP 地址在从源端到目的端的任意数据报中都没有出现，为什么我们需要给路由器指定 IP 地址？答案是在很多协议中，路由器扮演数据报发送方或者接收方的角色。例如，我们在第 4 章中讨论的路由协议，路由器是信息的发送方或者接收方。这些协议中的通信发生在路由器之间。
- 为什么在一个路由器中需要多个 IP 地址，每个接口一个？答案是接口是路由器到链路的连接。我们曾经说一个 IP 地址定义了因特网中的一点，该点与设备相连。有 n 个接口的路由器与因特网中 n 点相连。这就如同在角角的房子有两个门一样；每个门都有与对应街道相应的地址。
- 分组中的源端 IP 地址和目的端 IP 地址是如何决定的？答案是主机知道它自己的 IP 地址，该 IP 地址即分组中源 IP 地址。正如我们在第 2 章中讨论的，应用层使用 DNS 的服务来寻找分组的目的地址，并将它传递给网络层，然后插入到分组中。
- 对于每条链路，源端链路层地址和目的端链路层地址是如何决定的？每一跳（路由器或者

主机）应该知道自己的链路层地址，正如本章中我们稍后讨论的。目的链路层地址通过地址解析协议（Address Resolution Protocol，ARP）决定，我们稍后将讨论。

- 链路层地址的长度是多少？答案是这依赖于链路层使用的协议。尽管对于整个因特网我们只有 IP 协议，但是我们可以在链路层使用不同的数据链路层协议。这就意味着我们讨论不同的链路层协议时，可以定义地址的长度。对于有线网络来说，本章将讲述，而无线网络相关的，将在第 6 章讲述。

例 5.16　正如我们本章稍后讨论的，在最通用的局域网，以太网中，链路层地址是 48 位（6 个字节），它用由冒号分开的 12 个十六进制数字表示。例如，下面是一台计算机的链路层地址。

A2:34:45:11:92:F1

地址解析协议（ARP）

无论何时一个结点要向链路上另一个结点发送 IP 数据报，它都有接收结点的 IP 地址。源主机知道默认路由器的 IP 地址。路径上的路由器（除最后一个路由器外）通过使用转发表（forwarding table）来获取下一个路由器的 IP 地址（在下一跳列中）。最后的路由器知道目的主机的 IP 地址。然而下一个结点的 IP 地址对于帧通过链路移动时没有帮助；我们需要下一个结点的链路层地址。此时 ARP 协议（Address Resolution Protocol，ARP）变得非常有用。ARP 是定义在网络层的辅助协议之一，如图 5-46 所示。它属于网络层，但是我们将其推迟到现在才讲述是因为它将 IP 地址映射为逻辑链路地址。ARP 接收来自于 IP 协议的 IP 地址，将该地址映射为对应的链路层地址，并将其传递给数据链路层。

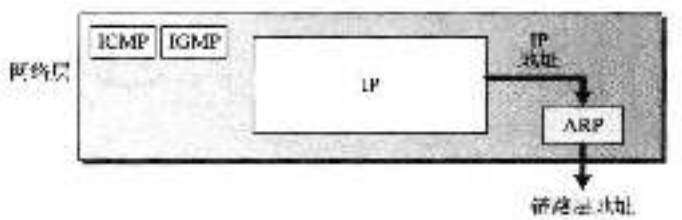


图 5-46　TCP/IP 协议簇中 ARP 的位置

无论何时一台主机或者路由器需要查询在其网络中另一台主机或者路由器的链路层地址时，它发送 ARP 请求分组。该分组包含发送方的链路层地址、发送方的 IP 地址和接收方的 IP 地址。因为发送方不知道接收方的链路层地址，该查询使用链路层广播地址在链路上广播查询，我们将在稍后讨论每一种协议的情况（见图 5-47）。

网络上的每一台主机或路由器接收和处理 ARP 请求分组，但是只有预定的接收方识别它的 IP 地址，并发回一个 ARP 响应分组。该响应分组包含接收方的 IP 地址和链路层地址。该分组是单播的，直接发给发送请求分组的结点。

在图 5-47a 中，左边的系统（A）有一个分组需要发送到给 IP 地址为 N2 的另一个系统（B）。为了发送，系统 A 需要将分组传递给链路层，但是它不知道接收方的物理层地址。它使用 ARP 的服务，要求 ARP 协议发送一个广播 ARP 请求分组来请求 IP 地址为 N2 的系统的物理地址。

该分组被物理网络上的每一个系统接收，但是只有系统 B 会应答，如图 5-47b 所示。系统 B 发送一个包含它物理地址的 ARP 应答分组。现在系统 A 可以使用它接收到的物理地址来发送到此目的地的所有分组。

分组格式

图 5-48 说明了一个 ARP 分组的格式。各个域的名字是自解释的。硬件类型域（hardware type）定义了链路层协议的类型；类型值为 1 是以太网。协议类型域（protocol type）定义了网络层协议：IPv4 协议是$(0800)_{16}$。源硬件地址域和源协议地址域是可变长度的域，它们定义了发送方的链路层

地址和网络层地址。目的硬件地址域和目的协议地址域定义了接收方的链路层地址和网络层地址。ARP 分组直接封装成一个数据链路层帧。该帧需要一个域来描述属于 ARP 但不属于网络层数据报的有效负载。

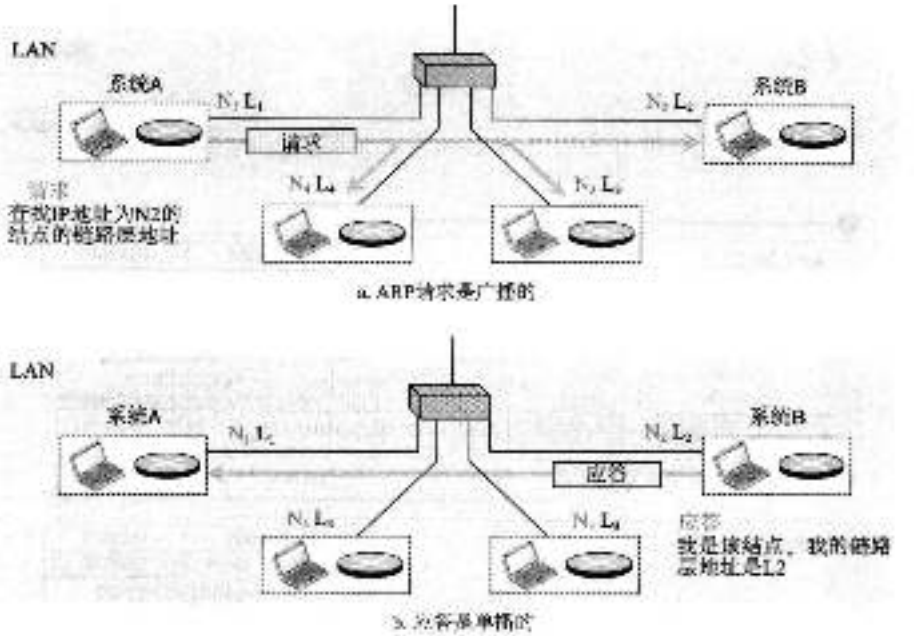


图 5-47　ARP 操作过程



图 5-48　ARP 分组

例 5.17　IP 地址为 N1，MAC 地址为 L1 的主机要向 IP 地址为 N2，MAC 地址为 L2（对于第一个主机来知）的另一个主机发送一个分组。这两个主机在相同的网络上。描述在以太网帧中封装的 ARP 请求和应答分组（见图 5-55）。

解答

图 5-49 描述了该 ARP 请求和应答分组。注意，在这种情况下 ARP 数据域是 28 字节，个别的地址不适合 4 字节的范围。这就是为什么我们没有描述这些地址是 4 个字节的原因。还需要注意 IP 地址以十六进制数表示。

一个例子

我们已经讨论了 TCP/IP 协议族中上面 4 层的寻址。在物理层没有寻址，因为该层中通信单元是位，它不能携带地址；在物理层位被发送，并由与发送方相连接的任意接收方接收（广播）。现

在是时候在一个例子中来描述这些地址，并强调它们的关系了。我们也说明了所有的这些地址如何由系统创建的。对于该例中只需要知道用户需要联系的站点的名字。图 5-50 说明了 Alice 和 Bob 相互连接的一个互联网络。Alice 想要访问由 Bob 提供的产品列表。Bob 有一个称为 Bob.biz 的小型商业站点，在该站点上，产品列表以及它们的规格说明存储在称为 products 的文档中。

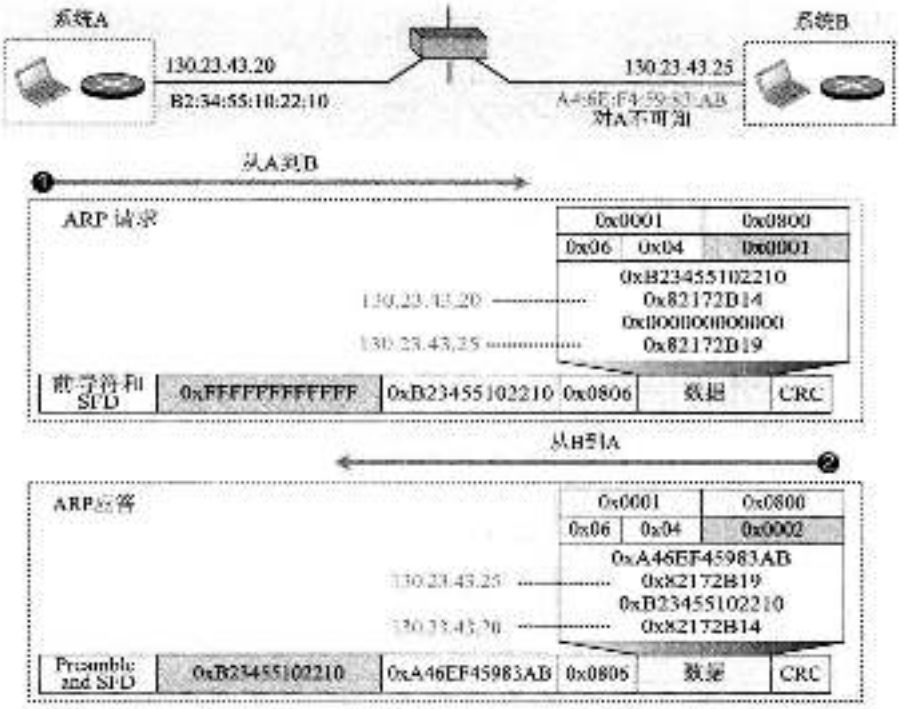


图 5-49　例 5.17

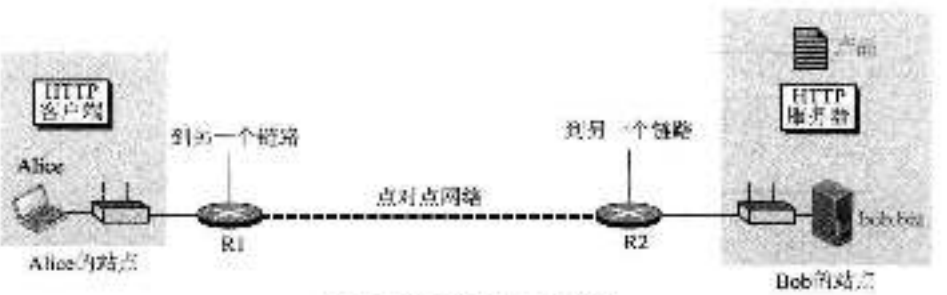


图 5-50　例子中的互联网络

Alice 需要知道的唯一标识符是站点的 URL：http://bob.biz/product。地址的剩余部分（源地址和目的地址）都由系统产生，如例子中所示。

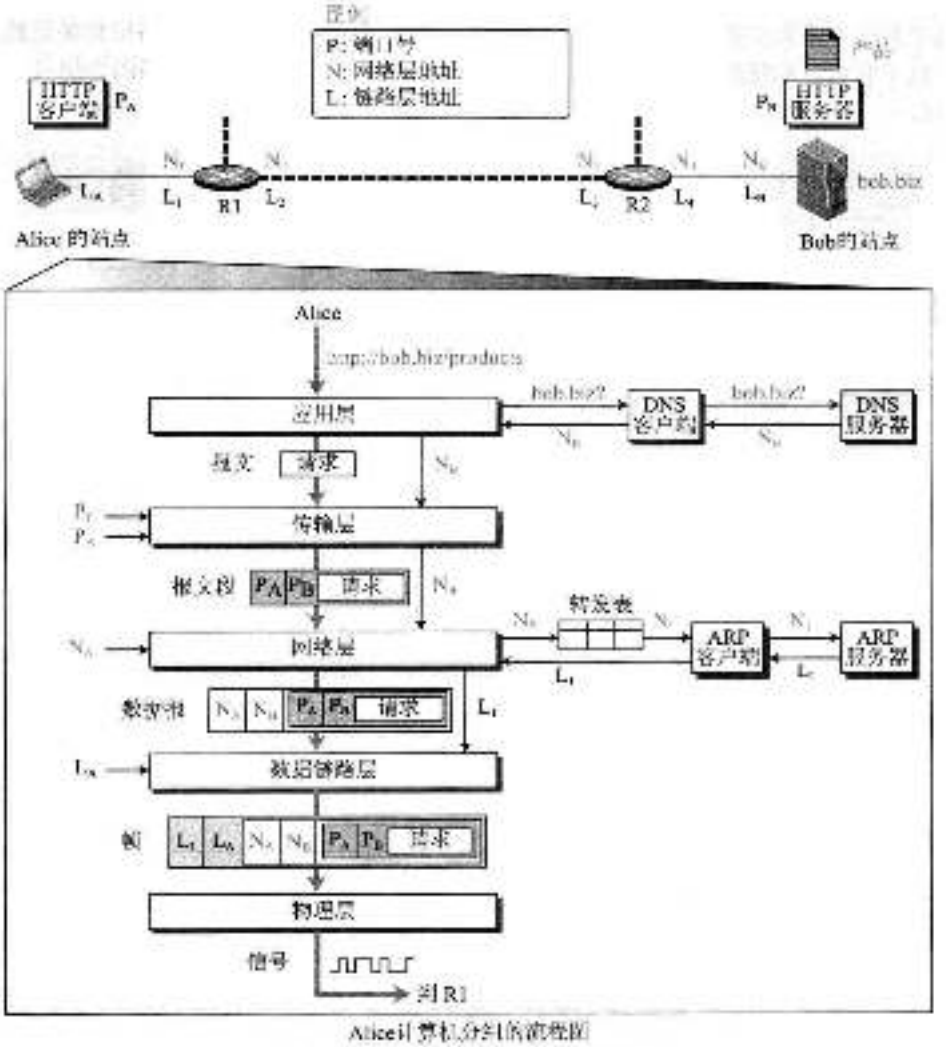
Alice 站点的活动

我们假设 Alice 连接到一个局域网，她的计算机知道自己的 IP 地址和链路层地址。Bob 的服务器也连接到一个局域网，他的计算机也知道它的 IP 地址和链路层地址。互联网络中的路由器也知道它们的 IP 地址和链路层地址。我们将使用符号地址以使说明更易读。然而，Alice 不需要知道这些地址；当事务开始时，它们自动被创建。

图 5-51 说明了 Alice 的站点发生了什么。Alice 使用浏览器并键入 Bob 站点的 URL

（http://bob.biz/product）。Alice 的计算机使用 HTTP 客户端来应答 Alice 的请求。但是 HTTP 客户端不知道 Bob 的 IP 地址。该客户端调用称为 DNS 的客户端寻求帮助。DNS 客户端向 DNS 服务器发送请求并查询 Bob 计算机的 IP 地址（N_B），该地址传递给传输层。

传输层使用熟知的 HTTP 服务器端口（P_B=80）和由操作系统为 HTTP 客户端分配的临时端口号（P_A）创建了一个报文段。传输层可以构造报文段并将该报文段以及先前获取的 Bob 计算机的 IP 地址（N_B）传递给网络层。当然，我们没有描述发生的所有的握手过程；我们假设连接在此建立并且只交换一个报文段。



Alice计算机分组的流程图

图 5-51　Alice 计算机分组的流程图

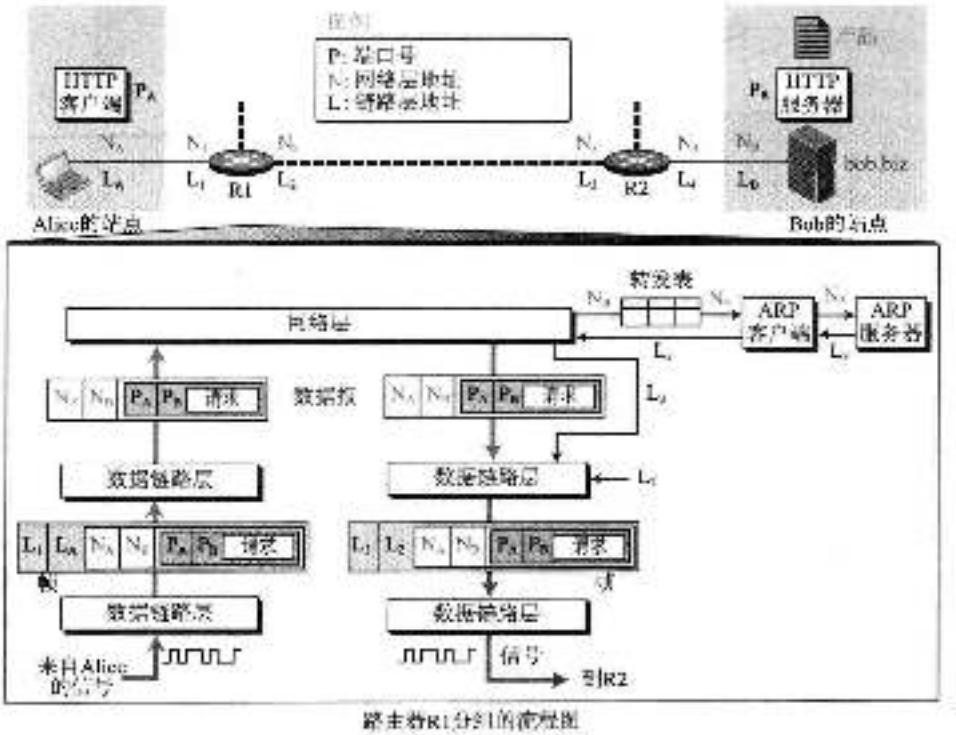
网络层接收报文段和 N_B，但是它需要查询链路层地址。网络层查询它的路由表，并试图查询到目的地 N_B 的下一个路由器（在这种情况下是默认路由器）。路由表给出 N_1，但是网络层需要查询路由器 R_1 的链路层地址。它使用 ARP 客户端来查询链路层地址 L_1。网络层现在能够将数据报和链路层地址传递给数据链路层。

数据链路层知道它自己的链路层地址 L_A。它创建帧并将帧传递给物理层，在物理层该地址转

类为信号（我们将在第 7 章学习），并且通过介质发送。

路由器 R1 的活动

现在让我们来查看路由器 R1 发生了什么。如我们所知，路由器 R1 具有下面三层。接收到的分组需要经过这三层。图 5-52 说明了该活动。帧到达后，左边链路的物理层创建该帧并将其传递给数据链路层。数据链路层拆封获得数据报然后将其传递给网络层。网络层检查数据报的网络层地址，发现数据报需要发送给 IP 地址为 N_B 的设备。网络层查阅它的路由表，发现到 N_B 的路径上下一个结点（路由器）是哪一个。该转发表返回 N_2。路由器 R2 的 IP 地址和 R1 的在同一个链路上。网络层现在使用 ARP 客户端和服务器来查找该路由器的链路层地址，即 L_1。网络层将数据报和 L_1 传递给属于右边（原书左边，怀疑有误。——译者注）的数据链路层。该链路层封装该数据报，添加 L_3 和 L_2（它自己的链路层地址）。然后将帧传递给物理层。物理层将位编码为信号，通过介质发送给 R2。



路由器R1分组的流程图

图 5-52　路由器 R1 的活动流程图

路由器 R2 的活动

除了一些地址变化外，该路由器的活动和路由器 R1 的相同。所以我们不在这里重复了。

Bob 站点的活动

现在让我们看一下 Bob 的站点发生了什么。图 5-53 说明了在 Bob 站点信号如何转变成 HTTP 服务器程序的报文。在 Bob 的站点不需要更多的地址或者映射。从链路层接收的信号转变为帧。该帧被传递给数据链路层，该层拆封该帧获取数据报，并将其传递给网络层。网络层拆封该数据报获取报文，并将其传递给传输层。传输层拆封报文段，并将其传递给端口 P_B(80)。HTTP 服务器接收到报文，准备一份 products 副本，将其格式转化为 HTML 文档，使用相同的流程图将其发送给 Alice，但是该图要倒序。返回的报文可能会经过相同的路由器 R2 和 R1，然后到达 Alice。

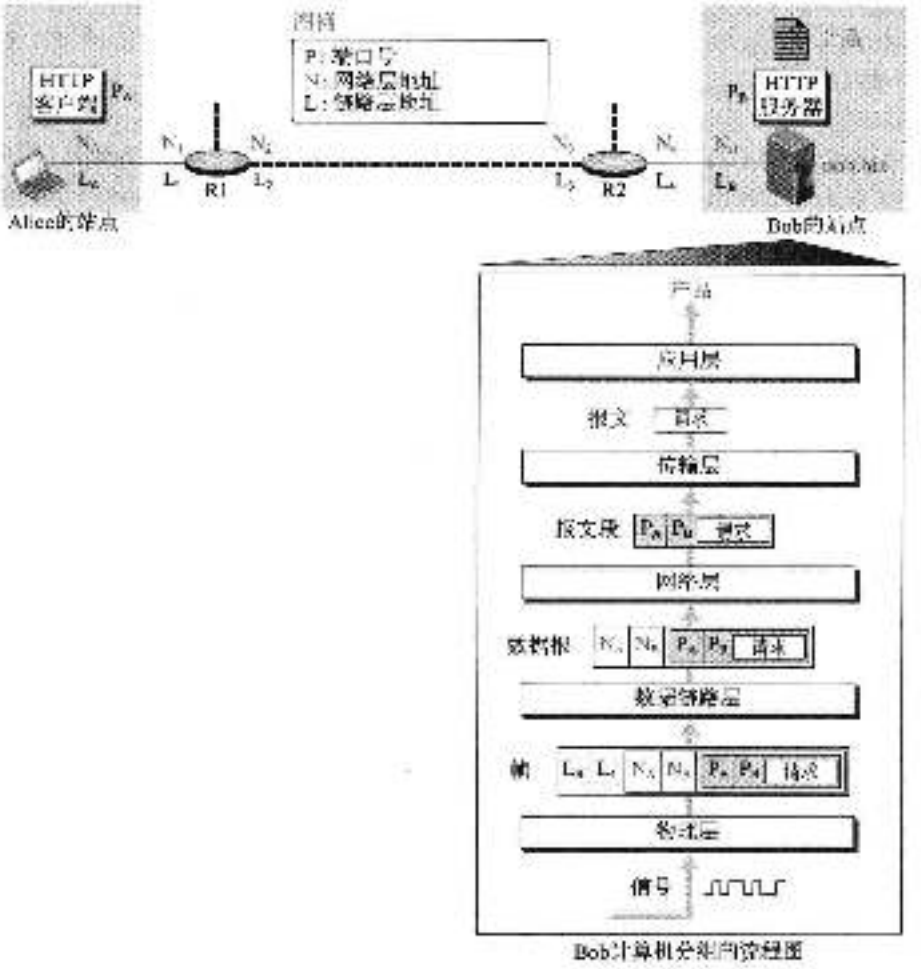


图 5-53　Bob 站点的活动

5.5　有线局域网：以太网协议

在第 1 章中，我们提到过 TCP/IP 协议簇没有为数据链路层和物理层定义任何协议。换言之，TCP/IP 接收这两层中能够向网络层提供服务的任意协议。数据链路层和物理层实际上属于本地网络和广域网络的范围。这意味着当我们讨论这两层时，我们在讨论使用它们的网络。正如我们在本章和第 6 章所见，我们可以有有线网络和无线网络。我们在本章讨论有线网络，将无线网络的讨论推迟到第 6 章。

在第 1 章中，我们知道本地局域网（LAN）是一个计算机网络，该网络是为有限地理区域而设计的，如一座建筑或者一所学校。尽管只为了资源共享的目的，局域网可以用作一个孤立的网络来连接组织中的计算机，但是现在大部分的局域网也连接到一个广域网（WAN）或者因特网。

在 20 世纪 80 年代至 90 年代，几种不同类型的局域网在使用。所有的局域网使用一种介质访问方法来解决共享介质的问题。以太网使用 CSMA/CD 方法。令牌环、令牌总线和光纤分布式数据接口（Fiber Distribution Data Interface，FDDI）使用令牌传递方法。在此期间，另一种局域网技术（ATM LAN）在市场上出现，该技术使用了高速广域网技术（ATM）。

几乎除以太网之外的其他局域网都从市场上消失了，因为以太网能够及时更新来满足时代的需要。在文献中已经提到了几个它成功的原因，但是我们相信以太网协议被这样设计以便它能够演变

以适应高传输速率的要求。一个组织过去使用以太网局域网，现在它需要更高的数据速率，因此它更新到新一代以太网，而不是转变到新的技术。后者花费可能会更高，这是很自然的。这意味着我们将有线局域网的讨论限制到以太网的讨论。

以太网局域网在 20 世纪 70 年代由 Robert Metcalfe 和 David Boggs 开发。从那时起，它已经经历了 4 代：**标准以太网**（Standard Ethernet）（10Mbps）、**快速以太网**（Fast Ethernet）（100Mbps）、**千兆以太网**（Gigabit Ethernet）（1Gbps）和 **10 千兆以太网**（10 Gigabit Ethernet）（10Gbps）。

5.5.1　IEEE 项目 802

在我们开始讨论以太网协议和它所有的世代之前，我们需要简要地讨论一下 IEEE 标准，我们经常在课本或现实生活中遇到它们。在 1985 年，计算机团体 IEEE 开启一个项目，称为**项目 802**（Project 802），以设定标准使得不同厂商生产的设备之间相互通信。项目 802 不是试图代替 OSI 模型或是 TCP/IP 协议簇的内容。相反，它说明了大部分 LAN 协议的物理层和数据链路层的功能。

图 5-54 说明了 802 标准和 TCP/IP 协议簇的关系。IEEE 将数据链路层细分为两个子层：**逻辑链路控制**（logical link control，LLC）和**介质访问控制**（media access control，MAC）。IEEE 也为不同的局域网协议创建了一些物理层标准。



图 5-54　局域网 IEEE 标准

逻辑链路控制（LLC）

先前我们讨论了数据链路控制。我们说数据链路控制用来处理帧、流量控制和差错控制。在 IEEE 项目 802 中，流量控制、差错控制和部分成帧职能都被集中到一个称为逻辑链路控制（LLC）的子层中。成帧在 LLC 子层和 MAC 子层处理。

LLC 为所有的 IEEE 局域网提供一个单一的链路层控制协议。这就意味着 LLC 协议能够在不同局域网之间提供相互连接性，因为它使 MAC 子层变得透明。

介质访问控制（MAC）

先前我们讨论的多路访问方法包括随机访问、受控访问和通道化。IEEE 项目 802 已经创造了称为介质访问控制的子层，为每个局域网定义了特定的访问方法。例如，它定义了 CSMA/CD 作为以太网局域网的介质访问方法，定义了令牌传递方法作为令牌环和令牌总线局域网的介质访问方法。正如我们前几节所提到的，部分成帧功能也由 MAC 层处理。

5.5.2　标准以太网

我们将原始的 10Mbps 数据速率的以太网技术称为标准以太网。尽管在以太网演化过程中，大部分实现已经迁移到其他技术，但是一些标准以太网的特点在演化过程中仍未改变。我们讨论标准版本为理解其他三种技术打好基础。

帧格式

以太网帧包含 7 个域，如图 5-55 所示。

- 前导符。该域包含 7 个字节（56 位），其中 0 和 1 交替出现，以此通知接收系统有帧到来，并使其同步它的时钟。该模式只提供一个通知和时钟脉冲。56 位的模式允许站点丢失帧开始处的一些位。该前导符实际在物理层添加，并非帧的（正式的）一部分。



图 5-55　以太网帧

- 起始帧分界符（SFD）。该域（1 个字节：10101011）表示帧的开始。SFD 通知站点这是最后一次同步的机会。最后两位是(11)$_2$，通知接收方下一个域是目的地址。该域实际上是一个标志，它定义了帧的开始。我们需要记住一个以太网帧是一个可变长度的帧。它需要一个标记来定义这个帧的开始。SFD 域也是在物理层添加的。
- 目的地址（DA）。该域是 6 个字节（48 位），包含目的站点或接收分组站点的链路层地址。我们稍后讨论寻址。当接收方看到自己的链路层地址，或是看到接收方所属组的广播地址，或是看到广播地址时，它解封来自帧的数据，并将数据传递给由类型域定义的上层协议。
- 源地址（SA）。该域也是 6 个字节，并包含分组发送方的链路层地址。我们稍后将讨论寻址。
- 类型。该域定义了帧中封装的分组所属的上层协议。该协议可以是 IP、ARP、OSPF 等等。换言之，它和数据报中的协议域、报文段或用户数据报中的端口号起到相同的作用。它用于多路复用和多路分解。
- 数据。该域运载来自上层协议封装的数据。它最小长度是 46 字节，最大长度是 1500 字节。我们稍后讨论最小值和最大值的原因。如果来自上层的数据多于 1500 字节，它就应该被分段并且被封装为多个帧。如果它小于 46 字节，就需要使用额外的 0 来填充。被填充的数据帧原封不动地交付给上层协议（不会去除填充）。这意味着添加或者删除填充是上层的任务。上层协议需要知道它的数据长度。例如，一个数据报有一个域来定义数据的长度。
- CRC。最后一个域包含差错检测信息，在此情况下为 CRC-32。CRC 由地址、类型和数据域计算而来。如果接收方计算 CRC 并发现它非 0（在传输过程中损坏），它丢弃该帧。

无连接服务和不可靠服务

以太网提供无连接服务，这意味着发送的每一个帧独立于前一个帧和后一个帧。以太网没有连接建立和连接中止阶段。当发送方要发送帧时，它就发送；接收方可能准备好接收也可能没准备好接收。发送方可能会淹没接收方，这就导致丢帧。如果帧被丢掉，发送方不知道它丢掉了。因为使用以太网服务的 IP 协议也是无连接的，所以它也无法知道发生丢帧。如果传输层也是无连接的协议，如 UDP，帧就丢掉了，补救措施只可能来自于应用层。但是，如果传输层是 TCP，发送方 TCP 没有接收到它的报文段的确认，就再次发送该报文段。

以太网像 IP 和 UDP 一样也是不可靠的。如果帧在传输过程中被破坏了，接收方发现了帧被破坏，该现象因为 CRC-32 的存在而有较高的可能性发生，接收方就会丢弃掉帧。发现丢帧是高层协议的任务。

帧长度

以太网对帧长度的最大值和最小值加以限制。正如我们稍后所见，最小长度限制对于 CSMA/CD 的正确操作是必需的。一个以太网帧需要的最小长度为 512 位或者 64 字节。这个长度

的一部分是头部和尾部。如果我们数出 18 字节的头部和尾部（6 字节源地址，6 字节目的地址，2 字节长度或类型，4 字节 CRC），然后来自于上层的数据最小长度为 64 – 18 – 46 字节。如果上层协议分组小于 46 字节，填充来补足差额。

标准定义了一个帧的最大长度（没有前导符和 SFD 域）是 1518 字节。如果我们减去 18 字节的头部和尾部，负载的最大长度为 1500 字节。最大长度限制有两个历史原因。首先，当设计以太网时内存很昂贵，最大长度限制帮助减少了缓冲区的长度。其次，最大长度限制阻止了一个站点独占共享介质，阻塞其他站点发送数据。

最小帧长度：64 字节　最小数据长度：46 字节
最大帧长度：1518 字节　最大数据长度：1500 字节

寻址

以太网上的每一个站点（例如 PC、工作站或者打印机）都有自己的网络接口卡（network interface card，NIC）。NIC 安装在站点内部，提供给站点链路层地址。以太网地址是 6 字节（48 位），通常以十六进制表示，在字节之间有冒号。例如，下面展示了一个以太网 MAC 地址：

4A:30:10:21:10:1A

地址在传输

地址在线发送的方式不同于它们以十六进制表示的方式。传输是自左向右，一个字节接一个字节的；但是，对于每一个字节，最低有效位首先发送，最高有效位最后发送。这意味着接收方定义单播或多播地址的位首先到达。这就帮助接收方即刻知道分组是单播的还是多播的。

例 5.18　说明地址 47:20:1B:2E:08:EE 如何在线发送。

解答

地址自左向右发送，一个字节接一个字节的；对于每个字节，它自右向左发送，一位接一位的，如下：

十六进制	47	20	1B	2E	08	EE
二进制	01000111	00100000	00011011	00101110	00001000	11101110
传输	←11100010	00000100	11011000	01110100	00010000	01110111

单播地址、多播地址和广播地址

源地址总是单播地址，帧只来自于一个站点。但是目的地址可以是单播的，多播的或者广播的。图 5-56 说明了如何区分单播地址和多播地址。如果目的地址第一个字节的最低有效位是 0，地址是单播的；否则，是多播的。

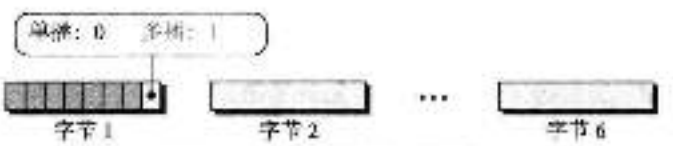


图 5-56　单播地址和多播地址

注意，由于位的传输方式，单播/多播位是传输或者接收的第一位。广播地址是多播地址的一个特例：接收方是局域网上的所有站点。广播地址是 48 位 1。

例 5.19　定义下面目的地址的类型：

a. 4A:30:10:21:10:1A
b. 47:20:1B:2E:08:EE
c. FF:FF:FF:FF:FF:FF

解答

为了明确地址的类型，我们需要查看左边开始的第二个十六进制数字。如果它是偶数，该地址是单播地址。如果它是奇数，该地址是多播地址。如果所有的位都是 F，该地址是广播地址。因此，我们得到：

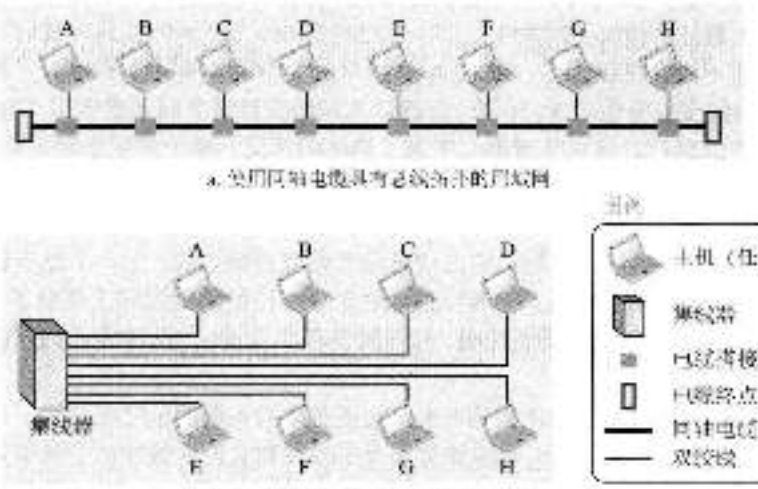
a. 这是单播地址，因为 A 的二进制表示是 1010（偶数）。

b. 这是多播地址，因为 7 的二进制表示是 0111（奇数）。

c. 这是广播地址，因为所有的数字都是十六进制数字 F。

单播传输、多播传输和广播传输的区别

标准以太网使用同轴电缆（总线拓扑）或是带集线器的一组双绞线（星状拓扑），如图 5-57 所示。



a. 使用同轴电缆具有总线拓扑的以太网

b. 使用集线器具有星状拓扑的以太网

图 5-57　标准以太网的实现

我们需要知道标准以太网的传输总是广播的，不管目的地址是单播、多播还是广播。在总线拓扑中，当站点 A 向站点 B 发送帧时，所有的站点都会接收。在星状拓扑中，当站点 A 向站点 B 发送时，集线器将会接收。因为集线器是被动设备，它不会检查帧的目的地址；它恢复位（如果它们变弱），然后向除 A 外的所有的站点发送。事实上，它使用该帧向网络泛洪。

问题是如何区别实际的单播、多播和广播传输。答案在于帧被保留或者丢弃的方式。

- 单播传输中，所有的站点将会接收帧，接收方保留并处理该帧；其他的丢弃该帧。
- 在多播传输中，所有的站点接收该帧，组中的成员站点保留并处理该帧；其他的丢弃该帧。
- 在广播传输中，所有的站点（除了发送方）将会接收该帧，所有的站点（除了发送方）保留并处理该帧。

访问方法

因为使用标准以太网协议的网络是广播网络，我们需要使用访问方法来控制共享介质的访问。标准以太网选择带有 1-持续方法的 CSMA/CD，前面在图 5-37 到图 5-40 讨论过。让我们用例子来说明该方法如何在以太网工作。

- 假设图 5-57 中站点 A 要向站点 D 发送帧。站点 A 首先检查是否有其他站点在发送（载波侦听）。站点 A 测量介质上的能量水平（很短的时间周期，一般少于 100μs）。如果介质上没有信号能量，这就意味着没有站点在发送（或者信号还没有到达站点 A），站点 A 将这种情况视为介质空闲。它开始发送帧。另一方面如果信号能量水平不是 0，这就意味着介质正在

被另一个站点使用。站点 A 不断地监测介质直至 100μs 后介质变为空闲。然后它开始发送帧。但是站点 A 需要在缓冲区中保留该帧的副本直至它确定介质上没有冲突为止。站点 A 何时确定没有冲突是我们下面要讨论的。

- 站点 A 开始发送帧后，介质侦听并没有停止。站点 A 需要持续的发送和接收。可能发生两种情况：

a. 站点 A 已经发送了 512 位，并且没有侦听到冲突（能量水平没有超过正常能量水平），然后该站点确定该帧将会通过，并且停止侦听介质。数字 512 位如何得来？如果我们认为以太网的传输速率是 10Mbps，这就意味着发送 512 位花费了站点 512/（10Mbps）= 51.2μs 的时间。依据电缆中的传播速度（2×10^8 米），第 1 位可能已经经过 10 240 米（单程）或者只有 5 120 米（往返），和电缆上的最后一个站点的一位冲突了，并且已经返回。换言之，如果冲突要发生，它会等到发送方已经发送完 512 位（最坏的情况）时发生，第 1 位已经完成 5 120 米的往返。我们知道如果冲突在电缆的中部发生，而不是在终点发生，站点 A 会更早的侦听到冲突并放弃传输。我们也需要谈到另一个问题。上面的假设中电缆是 5 120 米。标准以太网的设计者实际上设置了 2 500 米的限制，因为我们需要考虑途中的延迟。这意味着他们考虑了最坏的情况。整个思想是如果站点 A 在发送 512 位之前没有侦听到冲突，这里一定没有冲突，因为这段时间中，第 1 位已经到达了线路的终点，所有其他的站点知道有一个站点正在发送帧，延迟自己发送帧。换言之，问题发生在当另一个站点（例如最后的站点）在站点 A 的第一位到达它之前已经开始发送帧之时。另一个站点错误地认为线路是空闲的，因为第一位还没有到达它。读者应该注意到 512 位的限制实际上帮助了发送站点：发送站点确认如果在前面 512 位没有侦听到冲突，那么就没有冲突发生，这样发送站点就可以丢弃缓冲区中该帧的副本。

b. 在发送 512 位之前站点 A 已经侦听到冲突。这就意味着先前的位已经和另一个站点发送的位发生了冲突。在这种情况下，两个站点都应该抑制发送帧并在它们的缓冲区中保留帧以等待线路可用时重发。但是，为了通知其他站点网络中发生冲突，该站点会发送一个 48 位的堵塞信号。该堵塞信号是要创建允足的信号（即使冲突在一些位后发生）来通知其他站点发生了冲突。在发送堵塞信号后，站点需要增加 K（尝试次数）的值。如果增大至 K=15，经验说明网络太忙；站点需要放弃它的努力并重试。如果 K < 15，站点需要等待一段补偿时间（见图 5-40 中 T_B），然后重启进程。如图 5-40 所示，站点创建一个 0 到 2^K-1 的随机数字，这就意味着每一次冲突发生，随机数的范围以指数方式增加。第一次冲突（K = 1）后，随机数在范围（0,1）；在第二次冲突（K = 2）后，在范围（0,1,2,3）中。在第三次冲突（K=3）后，在范围（0,1,2,3,4,5,6,7）中。这样每经过一次冲突，补偿时间变长的可能性就增加了。如果第三次或第四次尝试后冲突发生，这就意味着网络真的很忙；就需要更长的补偿时间。

标准以太网的效率

以太网的效率定义为站点发送数据的时间与站点占有介质的时间之比。标准以太网的实际效率由以下公式衡量：

$$效率 = 1 / (1 + 6.4 \times a)$$

其中参数 a 是介质上帧的数目。它可以由如下公式计算得到：a =（传播延迟/传输延迟），因为传输延迟是一个平均大小的帧被发送所需的时间，传播延迟是到达介质末端的时间。注意随着参数 a 减少，效率增加。这意味着如果介质的长度更短或者帧更大，效率就会增加。理想的情况是，a = 0，效率为 1。我们在本章末尾的问题中会要求计算该效率。

例 5.20　在传输速率为 10Mbps 的标准以太网中，我们假设介质的长度为 2500m，帧的长度为 512 位。电缆中信号的传播速率正常为 2×10^8 m/s。

传播延迟 = $2500/(2 \times 10^8) = 12.5$ μs　传输延迟 = $512/(10^7) = 51.2$ μs

$a = 12.5/51.2 = 0.24$　　效率 = 39%。

在这例子中 $a = 0.24$，意味着这种情况下只有 0.24 个帧占据整个介质。效率为 39%，算是中等的；意味着只有 61% 的时间介质被占据但是没有被站点使用。

实现

标准以太网定义了一些实现，但是只有 4 个在 20 世纪 80 年代变得流行。表 5-6 总结了标准以太网的一些实现。

表 5-6　标准以太网实现总结

实　现	介　质	介质长度	编　码
10Base5	Thick coax	500m	Manchester
10Base2	Thin coax	185m	Manchester
10Base-T	2UTP	100m	Manchester
10Base-F	2Fiber	2000	Manchester

在 10BaseX 命名法中，数字定义了数据速率（10Mbps），术语 Base 是指基带（数字）信号，X 要么近似定义了以 100 米为单位的电缆的最大长度（例如 5 代表 500 米，2 代表 185 米），或者定义了电缆的类型，T 代表无屏蔽双绞线（unshielded twisted pair cable，UTP），F 代表光缆。标准以太网使用基带信号，这就说明位转换为数字信号直接发送到线路上。所有的实现都使用 Manchester 编码，我们将在第 7 章中详细讨论该编码。

5.5.3　快速以太网（100 Mbps）

在 20 世纪 90 年代，一些传输速率高于 10 Mbps 的局域网技术在市场上出现，如 FDDI 和光纤通道。如果标准以太网想要继续存在，就不得不与这些技术竞争。以太网通过将传输速率增加到 100Mbps 实现了大的跨进。新一代的以太网称作快速以太网。快速以太网的设计者需要考虑与标准以太网兼容。MAC 子层保留下来，没做改变，这就意味着帧格式、帧最大长度和最小长度也不用改变。通过增加传输速率，标准以太网的那些依赖于传输速率、访问方法和实现的特点就不得不重新设计。

访问方法

我们记得 CAMA/CD 的正确操作依赖于传输速率、帧最小长度和最大网络长度。如果我们想要保留帧最小长度，那么网络最大长度就应该改变。换言之，如果最小帧长度仍然是 512 位，传输速度快了 10 倍，冲突检测需要快 10 倍，这就是说网络最大长度缩短 10 倍（传播速率不变）。所以快速以太网有两个解决方案（二选其一它就可以工作）：

1. 第一个解决方案是完全放弃总线拓扑，使用被动集线器和星状拓扑，但是使得网络最大长度是 250 米而非标准以太网的 2500 米。这种方法保留了与标准以太网的兼容性。

2. 第二个解决方案是使用带有缓存的链路层交换机（本章稍后讨论）来存储帧以及到每个主机的全双工连接，使得传输介质对于每台主机来说都是专有的。在这种情况下，不需要 CSMA/CD，因为主机间不存在相互竞争。链路层交换机从源主机接收帧并将其存储在缓冲区（队列）中等待处理。然后它检查目的地址，将帧发送至相应的接口。因为到交换机的连接是全双工的，目的地址也能够在接收帧的同时向另一个站点发送帧。换言之，共享介质转变为很多点对点介质，并且没有必要建立连接。

自动协商

快速以太网上增加的新特性是自动协商（autonegotiation），它允许一个站点或是集线器有一定的能力范围。自动协商允许两台设备协商它们的运行模式或是传输速率。这种设计使得不相容的设备能够相互连接。例如，最大数据率为 10Mbps 的设备可以与 100Mbps 数据速率的设备（可以在低速率下工作）通信。

实现

快速以太网在物理层的实现方式可以分为两线或是四线的。两线的实现可以是屏蔽双绞线（shielded twisted pair，STP），称为 100Base-TX，也可以是光纤，称为 100Base-FX。而四线实现则

只为非屏蔽双绞线（UTP）设计，称为 100Base-T4。表 5-7 总结了快速以太网的实现。我们将在第 7 章讨论编码。

表 5-7　快速以太网实现总结

实　现	介　质	介质长度	线路数量	编　码
100Base-TX	STP	100m	2	4B5B-LT3
100Base-FX	Fiber	185m	2	4B5B-NZ-I
100Base-T4	UTP	100m	4	Two 8B/6T

5.5.4　千兆以太网

对传输速率更高的需求使得千兆以太网（1000 Mbps）应运而生。IEEE 委员会称之为标准 802.3z。千兆以太网的目标是将数据速率提高到 1Gbps，同时保留地址长度、帧格式以及最大帧和最小帧的长度。

MAC 子层

以太网进化过程中的主要思想是保持 MAC 子层不变。但是为了达到数据率为 1Gbps，这就不可能了。千兆以太网在介质访问方面有两个独特方法：半双工和全双工。几乎千兆以太网的所有实现都采用了全双工方法，所以我们忽略半双工模式。在全双工模式中，有一个中央交换机与所有的计算机或是其他交换机相连。在这种模式中，对于每一个输入端口，每一个交换机都有缓冲区，数据在传输之前存储在缓冲区中。因为交换机使用帧的目的地址，并将帧发送到与特定目的主机相连的端口上，就不会有冲突发生。这就意味着 CSMA/CD 不会使用。没有冲突意味着电缆的最大长度由电缆中信号衰减决定而不是由冲突检测进程决定。

实现

表 5-8 总结了千兆以太网实现。S-W 和 L-W 分别意味着短波和长波。我们在第 7 章中讨论编码。

表 5-8　千兆以太网实现总结

实　现	介　质	介质长度	线路数量	编　码
1000Base-SX	Fiber S-W	550m	2	8B/10B+NRZ
1000Base-LX	Fiber L-W	5000m	2	8B/10B+NRZ
1000Base-CX	STP	25m	2	8B/10B+NRZ
1000Base-T4	UTP	100m	4	4D-PAM5

5.5.5　10 千兆以太网

近些年来，对于大城市的以太网使用存在另一种看法。该想法是扩展技术、数据率和覆盖范围以使以太网作为局域网和城域网（metropolitan area network，MAN）使用。IEEE 委员会创建了 10 千兆以太网并称它为标准 802.3ae。10 千兆以太网的设计目标可以总结为提升数据速率到 10Gbps，同时保持相同的帧大小和帧格式，允许局域网、城域网和广域网的互联。现在这样的数据速率只有光纤技术才可能达到。该标准定义了两类的物理层：LAN PHY 和 WAN PHY。第一个设计用来支持现在的局域网；第二个实际上定义了一个链路通过 SONET OC-192（稍后讨论）相连的广域网。

实现

10 千兆以太网只在全双工模式下工作，这就意味着不需要连接；CSMA/CD 在 10 千兆以太网中也没有使用。最常见的 4 种实现是：10GBase-SR、10GBase-LR、10GBase-EW、和 10GBase-X4。

表 5-9 总结了 10 千兆以太网实现。我们在第 7 章讨论编码。

表 5-9　10 千兆以太网实现总结

实　现	介　质	介质长度	线路数量	编　码
10GBase-SR	Fiber 850m	300m	2	64B66B
10GBase-LR	Fiber 1310m	10Km	2	64B66B
10GBase-EW	Fiber 1350m	40Km	2	SONET
10GBase-X4	Fiber 1310m	300m to 10Km	2	8B10B

5.5.6　虚拟局域网

如果站点物理上属于一个局域网，那么它被认为是那个局域网的一部分。成员的标准是地理上决定的。如果我们需要在两个属于不同物理局域网的站点之间的虚拟连接该怎么办？我们可以简单的定义一个**虚拟本地局域网**（virtual local area network，VLAN）作为由软件配置的本地局域网，而不是物理连接的。

让我们使用一个例子来详细说明该定义。图 5-58 是一个工程公司的交换局域网，10 个站点被

分为 3 组局域网，它们通过一个交换机相连。

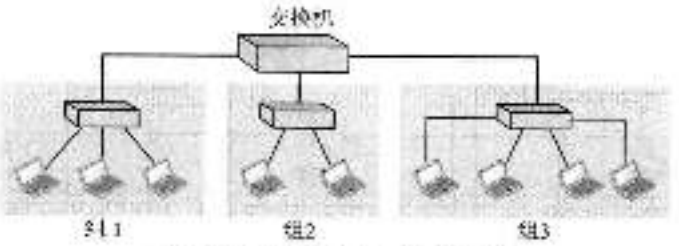


图 5-58　连接 3 个局域网的交换机

前面的 3 个工程师作为第一组一起工作，后面的两个工程师作为第二组一起工作，最后的 4 个工程师作为第三组一起工作。局域网配置为允许这种部署。

但是，如果为了加速第三组正在进行的项目，管理员需要将第一组的两个工程师移动到第三组该怎么办？局域网的配置就需要改变。网络技术人员必须重新布线。如果某个早期，两个工程师又移回他们先前的组，问题又会重复。在一个交换局域网中，工作组的变化意味着网络配置中物理的变化。

图 5-59 是相同的交换局域网，但是被分为 VLAN。VLAN 技术的核心思想是将局域网分为逻辑上的部分，而非物理上的。一个局域网可以被分为几个逻辑上的局域网，称为 VLAN。每一个 VLAN 是组织中的一个工作组。如果一个人从一个工作组移动到另一个，就没有必要改变物理配置。VLAN 的组成资格是由软件定义的而非硬件定义的。任何站点可以逻辑的移动到另一个 VLAN。属于一个 VLAN 的所有成员可以接收发送到该特定 VLAN 的广播信息。这意味着如果一个站点从 VLAN 1 移动到 VLAN 2，它接收发送到 VLAN 2 的广播信息，但是不在接收 VLAN 1 的广播信息。

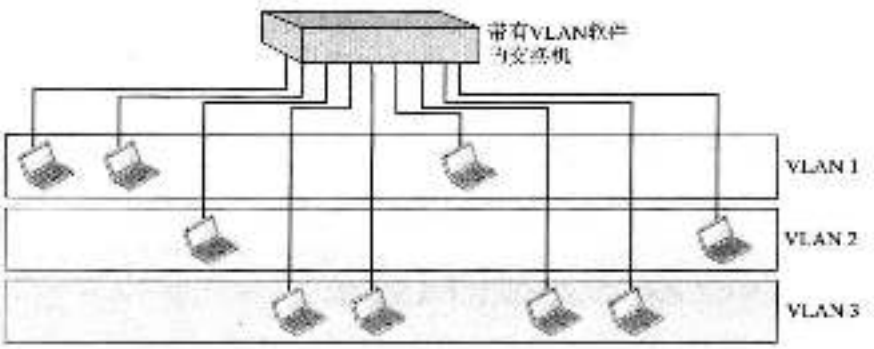


图 5-59　使用 VLAN 软件的交换机

通过使用 VLAN，我们先前例子中遇到的问题可以轻松的解决，这是很显然的。将工程师从一个组移动到另一个组，通过软件方法要比改变物理网络配置简单得多。

VLAN 技术甚至允许将一个 VLAN 中连接到不同交换机的站点分组。图 5-60 是有 2 个交换机和 3 个 VLAN 的骨干局域网。来自交换机 A 和 B 的站点属于每个 VLAN。

对于有两栋建筑的公司来说这是很好的配置。每一栋建筑有它自己的交换局域网，该局域网连接至骨干网。在第一栋建筑的人和第二栋建筑的人可以在相同的工作组中，即使他们连接到不同的物理 LAN。

从这二个例子我们可以发现一个 VLAN 定义了广播域。VLAN 将属于一个或多个物理局域网的站点分为多个广播域。在 VLAN 中的站点彼此通信，就好像它们属于一个物理分段一样。

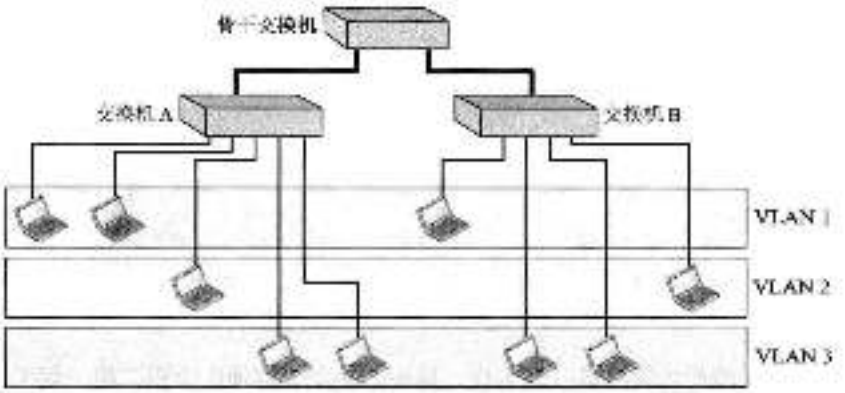


图 5-60　使用 VLAN 软件的骨干网中的两个交换机

成员

VLAN 中什么特征可以用来将站点分组？服务商使用不同的特征，例如接口号、端口号、MAC 地址、IP 地址、IP 广播地址或者它们中几个的组合。

接口号

一些 VLAN 供应商使用交换机接口号作为成员特征。例如，管理员可以定义连接到 1、2、3 和 7 号接口的站点属于 VLAN 1，连接到 4、10 和 12 的站点属于 VLAN 2，等等。

MAC 地址

一些 VLAN 供应商使用 48 位 MAC 地址作为成员特征。例如，管理员可以规定 MAC 地址为 E2:13:42:A1:23:34 和 F2:A1:23:BC:D3:41 的站点属于 VLAN 1。

IP 地址

一些供应商使用 32 位的 IP 地址（见第 4 章）作为成员特征。例如，管理员可以规定 IP 地址为 181.34.23.67、181.34.23.72、181.34.23.98 和 181.34.23.112 的站点属于 VLAN 1。

多播 IP 地址

一些 VLAN 供应商使用多播 IP 地址（见第 4 章）作为成员特征。在 IP 层的多播转变成数据链路层的多播。

组合

最近，一些来自供应商的可用软件允许所有的这些特征相互组合。管理员在安装软件时可以选择一种或多种特征。另外，软件可以重新配置来改变设置。

配置

这些站点是如何分为不同 VLAN 的？站点的配置方式有以下三种：手动、半自动和自动。

手动配置

在手动配置（manual configuration）中，网络管理员在设置时使用 VLAN 软件手动将站点分配至不同的 VLAN。从一个 VLAN 到另一个的迁移也是手动完成的。注意这不是物理层配置，而是逻辑的配置。术语手动这里的意思是管理员使用 VLAN 软件输入端口号、IP 地址或者其他特征。

自动配置

在自动配置（automatic configuration）中，站点使用由管理员定义的标准自动地与一个 VLAN 连接或者断开。例如，管理员可以定义项目编号作为组成员的标准。当一个用户改变了项目，他（她）自动地迁移到一个新的 VLAN。

半自动配置

半自动配置（semiautomatic configuration）处于手动配置和自动配置之间。通常初始化手动完

成，迁移自动完成。

交换机间通信

在一个多路交换骨干网中，每一个交换机必须知道哪个站点属于哪个 VLAN，以及连接到其他交换机的站点的成员。例如，在图 5-60 中，交换机 A 必须知道连接到交换机 B 的站点的成员状态，交换机 B 必须知道 A 的相同的情况。为此有三种方法：表维护（table maintenance）、帧标记（frame tagging）和时分多路复用（time-division multiplexing）。

表维护

这种方法中，当一个站点向它的组成员发送广播帧时，交换机在表里创建一个实体，记录站点成员。交换机彼此间发送表，并定期更新。

帧标记

在该方法中，当帧在交换机间移动时，额外的头部增加到 MAC 帧来定义目的 VLAN。接收交换机使用帧标记来决定要接收广播信息的 VLAN。

时分多路复用（TDM）

在该方法中，交换机间的连接（主干）分时使用通道（见第 7 章 TDM）。例如，如果骨干网中 VLAN 的总数是 5 个，每一个主干分为 5 个通道。去往 VLAN 1 的流量流向通道 1，去往 VLAN 2 的流量流向通道 2，等等。接收交换机通过检查帧到达的通道决定目的 VLAN。

IEEE 标准

在 1996 年，IEEE 802.1 附属委员会通过了称为 802.1Q 的标准，它定义了帧标记的格式。标准也定义了多路交换骨干网中使用的格式，使得在 VLAN 中可以使用多个供应商的设备。IEEE 802.1Q 开启了与 VLAN 相关的问题进一步标准化之路。大部分厂商已经接受了该标准。

优势

使用 VLAN 有以下优势：

降低费用和节省时间

VLAN 可以减少站点从一个组移动到另一个组的花费。物理上的重新配置花费时间并且代价很高，与物理上将一个站点移动到另一个分段或是将站点移动到另一个交换机相比，使用软件方法更加简单快速。

创建虚拟工作组

VLAN 可以用来创建虚拟工作组。例如，在校园环境中，进行相同工作的教授可以彼此发送广播信息，而不用属于相同的部门。如果使用 IP 的多播功能，这就可以减少流量。

安全

VLAN 提供额外的安全度量。属于相同组的人可以发送有保证的广播信息，这样其他组的用户就不会接收到这些信息。

5.6　其他有线网络

正如我们第 1 章中讨论的，在因特网中我们遇到的网络是一般是局域网和广域网。但是有些技术仍在争论中。例如，一些接入网络，如拨号连接和电缆连接，一些人称为广域网，而另一些人称为城域网（Metropolitan Area Networks，MAN）。在本节中，我们讨论这些在因特网中点接或间接使用的网络时，不会称其为广域网或城域网，而是称它们为网络。

5.6.1　点对点网络

一些点对点网络，如拨号、DSL 和电缆用来为因特网用户提供互联网络接入。由于这些网络在两个设备之间使用一个专用链接，因此它们不用使用介质访问控制（MAC）。正如我们先前讨论

的，只需要 PPP 协议即可。

拨号

拨号网络或连接使用电话网络提供的服务来传输数据。电话网络在 19 世纪末开始出现。整个网络最初是一个传输声音的模拟系统，称为普通电话业务（plain old telephone system，POTS）。随着计算机时代的出现，在 20 世纪 80 年代，该网络除了声音外也开始运载数据。在过去的 10 年里，电话网络已经经历了很多的技术改变。事实上，现在大部分的电话网络是数字的。只有连接用户接入电话网络的那部分线路是模拟的。传输数字数据的需求导致了拨号调制解调器的发明。

术语调制解调器（modem）是引用了两个组成该设备的功能实体的合成词：信号调制器和信号解调器。调制器由数字数据创建一个模拟信号。解调器把调制信号恢复成数字数据。只有一方使用数字信号（如通过因特网提供商）时，这些连接才能使用。其中下载速率（由因特网服务提供商流向 PC 的数据流）最大为 56kbps，而上传速率（由 PC 流向因特网服务提供商的数据流）最大值为 33.6kbps，它们是非对称的，如图 5-61 所示。

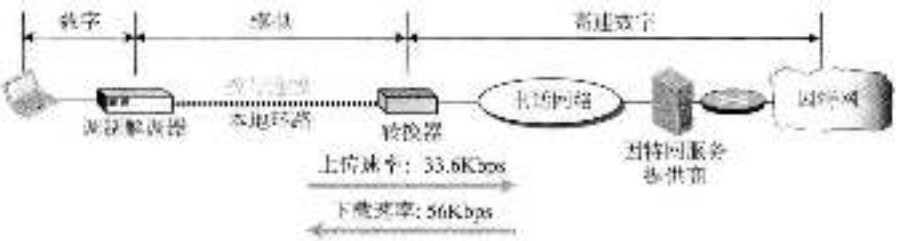


图 5-61　提供因特网接入的拨号网络

在上传过程中，模拟信号在进入高速数字电话网络之前必须被取样。在该方向上，量化噪声（见第 7 章）引入了信号之中，它将速率降至 33.6kbps。但是，在下载过程中没有采样发生。该信号不受量化噪声影响。对于正常接入因特网的个体用户来说，需要更高的下载速率，而非上传速率。例如，这些用户想要下载一个大文件往往比他们上传一个大文件更频繁，因此不对称的速率通常被忽视。我们需要提醒的是，当电话线路用于因特网接入时，不能同时用于言语通信。

有人可能想知道我们如何到达 56kbps 的速率。电话公司每秒钟采样 8000 次，每个样本 8 位。每一个样本中的一位用于控制，即每一个样本 7 位。速率因此是 8000 × 7，即 56 000 bps 或是 56 kbps。

数字用户线路（DSL）

在传统的调制解调器到达它们最高的数据速率后，电话公司开发了另一种技术 DSL 来提供高速率的因特网接入。**数字用户线路**（Digital subscriber line，DSL）技术是最有前途的技术之一，它通过已经存在的电话来支持高速数字通信。DSL 技术是一组技术，每一种的第一个字母都不同（ADSL、VDSL、HDSL 和 SDSL）。该集合经常称为 xDSL，x 可以由 A、V、H 或 S 代替。我们只讨论第一种 ADSL。集合中的第一种技术是非对称 DSL（ADSL）。像 56k 的调制解调器一样，ADSL 在下载方向（由因特网向用户）比上传方向（由用户向因特网）提供更高的速率（位速率）。这就是称其为非对称的原因。不像 56K 调制解调器的非对称，ADSL 的设计者特别为居民客户将不均衡的本地环路的可用带宽分离了。对于需要更大双向带宽的商业客户来说，这种服务是不合适的。

使用存在的本地环路

非常有趣的一点是 ADSL 使用已经存在的电话线路（本地环路）。但是 ADSL 如何达到传统调制解调器没有达到的速率？答案是电话线路中使用的双绞线实际上有能力处理达到 1.1MHz 的带宽，但是在电话公司的末端办公室（本地环路终止于此），它安装的过滤器将带宽限制在 4kHz（言频通信足够了）。如果将过滤器移除，那么整个 1.1MHz 就可用于数据和音频通信。典型地，1.104MHz

的可用带宽被分为一个音频通道、一个上行通道和一个下行通道，如图 5-62 所示。

ADSL 允许用户同时使用音频通道和数据通道。上行速率可以达到 1.44Mbps。但是由于通道中高噪声的影响，数据速率通常低于 500kbps。下行数据速率可以达到 13.4Mbps。但是由于通道中噪声的影响，数据速率通常低于 8Mbps。非常有趣的一点是这种情况下电话公司充当 ISP 的角色，因此像 email 或互特网接入的服务由电话公司自己提供。

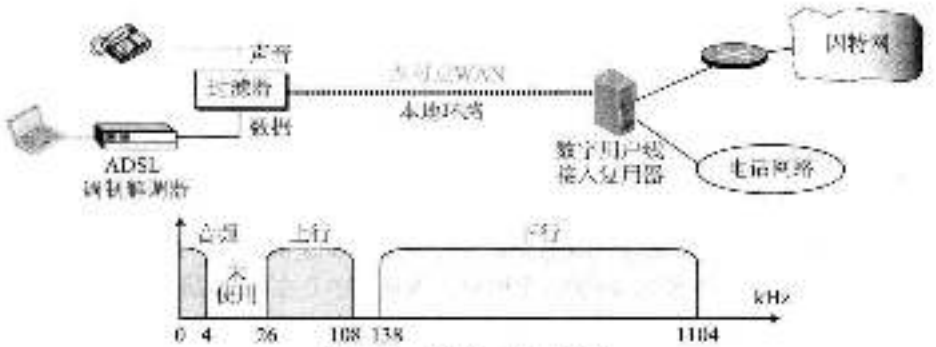


图 5-62　ASDL 点对点网络

电缆

有线网络最初是用来为那些因山脉等天然障碍物而无法接收电视信号的用户提供电视节目接入服务的。后来有线网络广受那些想要更好信号的用户的喜爱。另外，有线网络能够通过微波连接访问远程广播电台。有线电视通过使用最初为视频设计的通道也在互特网接入服务方面找到了很好的市场。在讨论有线网络的基本结构之后，我们讨论有线调制解调器如何提供到因特网的高速连接。

传统有线网络

在 20 世纪 40 年代，有线电视开始部署广播电视信号到那些信号弱或者没有信号的地区。这称为**有线电视**（community antenna TV，CATV），因为在高山或是建筑物的顶部有天线接收来自于电视台的信号，并且通过同轴电缆向各社区发布信号。图 5-63 显示了传统有线电视网络的原理图。

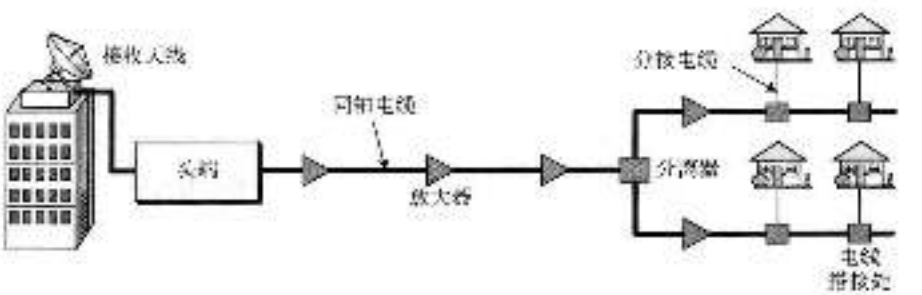


图 5-63　传统有线电视网

有线电视办公室称为头端，接收广播站的视频信号，并将信号提供给同轴电缆。随着距离变长信号开始变弱，所以放大器被引入到网络各处来更新信号。在头端和用户之间可能有多达 35 个放大器。在另一端分离器分开电缆，电线搭接处和分接电缆负责到用户住所的连接。

传统有线电视系统使用端到端的同轴电缆。由于信号的衰减和大量放大器的使用，传统网络中的通信是单向的。视频信号从头端向用户处所传输。

混合光纤同轴网络

有线网络的第二代称为**混合光纤同轴网络**（hybrid fiber-coaxial network，HFC）。该网络将光纤

和同轴电缆相结合。从有线电视办公室到称为光纤结点（fiber node）的盒子的传输介质是光纤；从光纤结点通过住宅区到房屋的传输介质仍是同轴电缆。图 5-64 是 HFC 网络的原理图。

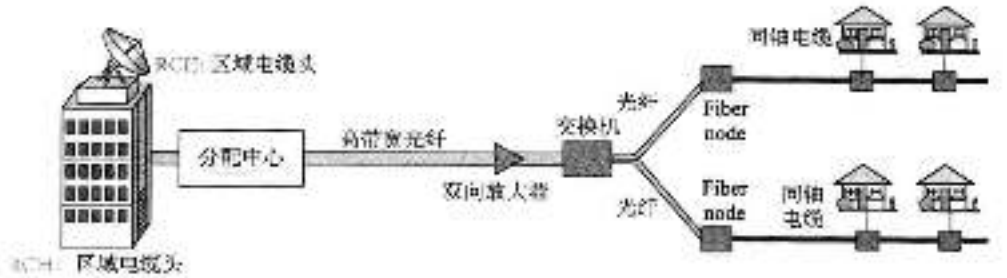


图 5-64　混合光纤同轴网络

区域电缆头（regional cable head，RCH）正常可以服务多达 400 000 个用户。RCH 供给分配中心，每一个分配中心可以服务多达 40 000 个用户。该分配中心在新的基础设施中扮演很重要的角色。信号的调制和分配在这里完成；然后信号通过光纤电缆供给光纤结点。光纤结点将模拟信号分离，这样相同的信号就被发送至各自同轴电缆。每一个同轴电缆可以为多达 1000 个用户提供服务。光纤电缆的使用将放大器的数目降至 8 个或者更少。

从传统的基础设施迁移到混合基础设施的一个原因是使得电缆网络变为双向的。

用于数据传输的有线电视

有线电视公司正与电话公司竞争那些想要获取高速数据传输的住宅用户。DSL 技术在本地环路上为住宅用户提供了高速数据连接。但是，DSL 使用现存的无屏蔽双绞线，这种线路易受干扰。这就限制了数据速率的上限。一种解决方法是使用有线电视网络。本节，我们简单地讨论该技术。

在 HFC 系统中，网络的最后一部分（从光纤结点到住宅用户）仍然是使用同轴电缆。这种同轴电缆的带宽从 5MHz 到 750MHz（近似值）。为了提供因特网接入服务，有线电视公司将该带宽分为三个波段：视频、下行数据和上行数据，如图 5-65 所示。

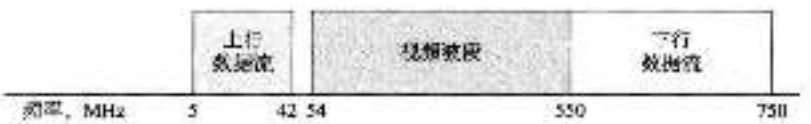


图 5-65　有线电视同轴电缆波段的划分

视频波段占据从 54MHz 到 550MHz 的频率。由于每个电视频道占据 6MHz，所以该波段可以容纳 80 多个频道。下行数据流（从因特网到用户住所）占据上面的波段，从 550MHz 到 750MHz。该波段也被分为 6-MHz 的通道。

上行数据流（从用户住所到因特网）占据下面的带宽，从 5MHz 到 42MHz。该波段也被分为 6-MHz 的通道。在 QPSK 中为 2 位/波特。该标准为每波特指定 1Hz；这意味着，理论上上行数据流可以以 12Mbps（2 bits/Hz × 6 MHz）发送数据。但是，数据速率通常少于 12Mbps。

共享

上行流波段和下行流波段由用户共享。上行数据流带宽为 37MHz。这意味着在上行方向，只有 6 个 6-MHz 通道可用。在上行方向，用户需要使用一个通道来发送数据。问题是，6 个通道如何被一个区域内的 1000、2000 或者甚至是 100 000 个用户共享？解决该方法是分时共享。波段被分为通道；这些通道必须由处于同一街区的用户共享。有线网络供应者静态地或是动态地为一组用户指定一个通道。如果一个用户想要发送数据，他（她）和那些想要访问的其他人竞争通道；用户

必须等待，直到通道可用。

我们在下行方向有相似的情况。下行波段有 33 个 6MHz 的通道。有线网络提供者的用户可能多于 33 个；因此，每个通道必须由一组用户共享。但是，和下行方向的情况是不同的：此处我们存在多点广播的情况。如果一组中的任意一个用户有数据，该数据就会发送至该通道。数据发送至每个用户。但是由于每一个用户在网络提供者处也有一个地址；该组的有线调制解调器将数据中携带的地址与提供者指定的地址匹配。如果地址匹配成功，数据就保留；否则，就被丢弃。

CM 和 CMTS

为了使用有线网络用于数据传输，我们需要两种关键设备：**有线调制解调器**（cable modem，CM）和**有线调制解调器传输系统**（cable modem transmission system，CMTS）。有线调制解调器安装在用户住所。有线调制解调器传输系统安装在有线电视公司。它从因特网接收数据并将其发送至用户。CMTS 也从用户接收数据并将其发送至因特网。它与 ADSL 调制解调器相似。图 5-66 说明了这两种设备的位置。像 DSL 技术，有线电视公司需要实现 ISP，给用户提供因特网服务。在用户住所，CM 将数据和视频分离，将其发送至电视或是计算机。

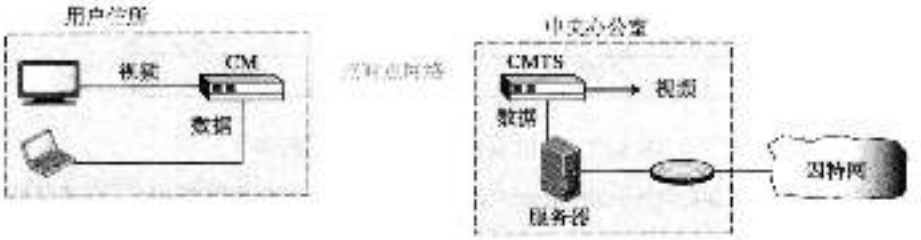


图 5-66　有线调制解调器传输系统（CMTS）

5.6.2　SONET

本节中，我们介绍一种高速网络，SONET，它用来承载其他网络的数据。我们首先讨论 SONET 的协议，然后我们说明如何从协议定义的标准构建 SONET 网络。

光纤电缆的高带宽适用于现在的高数据速率技术（如视频会议）和低速下同时承载大量数据。因此，光纤电缆和要求高速据速率或者高带宽传输的技术共同发展。继而有了标准化的需要，因此，美国组织（ANSI）和欧洲组织（ITU-T）定义了相应的标准，虽然它们是独立的，但是基本功能相似并且最终是兼容的。ANSI 标准称为同步光纤网络（Synchronous Optical Network，SONET），我们在此讨论。

体系结构

SONET 是使用同步 TDM 多路复用技术的同步网络。系统中所有的时钟都锁定于主时钟。我们首先介绍 SONET 系统的体系结构：信号、设备和连接。

信号

SONET 定义了称为**同步传输信号**（synchronous transport signal，STS）的层次电子信号。每一个 STS 等级（从 STS-1 到 STS-192）支持特定的数据速率，以每秒兆位规定，参见表 5-10。相应的光信号称为**光载波**（optical carrier，OC）。

表 5-10　SONET 速率

STS	OC	速率（Mbps）	STS	OC	速率（Mbps）
STS-1	OC-1	51.840	STS-24	OC-24	1244.160
STS-3	OC-3	155.520	STS-36	OC-36	1866.230
STS-9	OC-9	466.560	STS-48	OC-48	2488.320
STS-12	OC-12	622.080	STS-96	OC-96	4976.640
STS-18	OC-18	933.120	STS-192	OC-192	9953.280

通过观察表 5-10 可以看出一些有趣的地方。首先，该体系中最低等级的数据传输速率为 51.840Mbps，比 DS-3 服务（44.736Mbps，在第 7 章中解释）的速率大。事实上，STS-1 是设计用来调整数据速率以等同于 DS-3 的速率。提供容量上的差异来处理光系统中的开销需要。其次，

STS-3 速率是 STS-1 速率的三倍；STS-9 速率是 STS-18 速率的一半。这些关系意味着 18 个 STS-1 通道能复用成一个 STS-18，6 个 STS-3 通道能够复用成一个 STS-18 通道，依此类推。

SONET 设备

图 5-67 显示了使用 SONET 设备的简单链路。SONET 传输依靠三种基本设备：STS 多路复用器/分离器、再生器、分插复用器和终端。

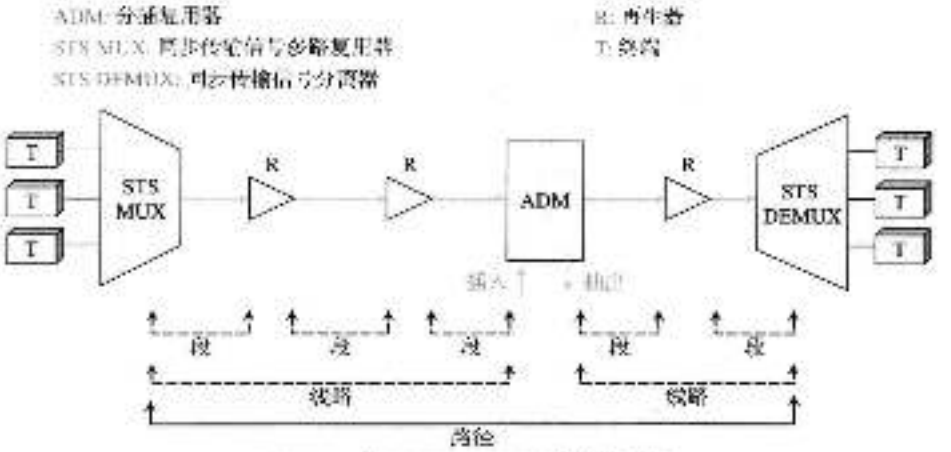


图 5-67　使用 SONET 设备的简单网络

STS 多路复用器/分离器（STS multiplexer/demultiplexer）　STS 多路复用器/分离器标记 SONET 链路的开始点和结束点。它们提供了电子支路网络和光网络之间的接口。STS 多路复用器（STS multiplexer）复用来自多个电子源的信号，并且产生相应的 OC 信号。STS 分离器（STS demultiplexer）将光 OC 信号分解成相应的电子信号。

再生器　再生器扩展了链路的长度。再生器（regenerator）是一个中继器（repeater），它把接收到的光信号（OC-n）解调成相应的电信号（STS-n），再生成电信号，最后将电信号调制成相应的 OC-n 信号。SONET 再生器用新的信息替换一些现有的开销信息（头部信息）。

分插复用器　分插复用器（add/drop Multiplexer，ADM）允许信号的插入和抽出。分插复用器能够将来自于不同源的 STSs 增加至指定路径或者从一条路径中除去并重定向所需的信号而不用分解整个信号。分插复用器使用诸如地址和指针（本节后面描述）的头部信息来确定单个流，而不是依靠时间和位位置。

终端　一台终端是使用 SONET 网络服务的设备。例如，在因特网中，一台终端可以是一台路由器，它需要向 SONET 网络中另一边的路由器发送分组。

连接　在前一节定义的设备使用段、线路和路径进行连接。

段

段（section）是连接两个相邻设备（多路复用器到多路复用器、多路复用器到再生器或是再生器到再生器）的光链路。

线路

线路（line）是两个多路复用器之间的网络部分。

路径

路径（path）是两个 STS 多路复用器之间的端到端网络部分。在两个 STS 多路复用器直接互联的简单 SONET 中，段、线路和路径相同。

SONET 层

SONET 标准包括四个功能层：光子层、段层、线路层和路径层。它们对应于物理层和数据链路

层。在各层都添加帧的头部，稍后在本章讨论。图 5-68 总层、层之间的关系以及前面描述的设备。

路径层

路径层（path layer）负责将信号从它的光信源移动到光信宿。在光信源，信号从电形式转变为光形式，再与其他信号复用在一起，封装成帧。在光信宿，接收到的帧被多路分解，单个的光信号再转换回电形式。本层增加了路径层开销。STS 多路复用提供了路径层功能。

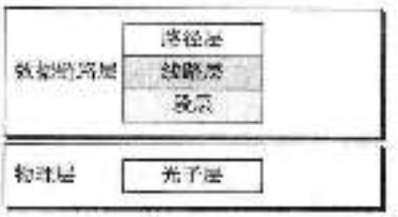


图 5-68　SONET 层与 OSI 或因特网层的对比

线路层

线路层（line layer）负责通过物理线路的信号移动。本层帧增加了线路层开销。STS 多路复用器和分插复用器提供了线路层功能。

段层

段层（section layer）负责信号通过物理段的移动。它处理成帧、串扰和差错控制。在本层帧增加了段层开销。

光子层

光子层（photonic layer）对应于物理层。它包括光纤通道、接收器敏感度、多路复用功能等的物理规范说明书。SONET 使用 NRZ 编码（见第 7 章），光存在为 1，不存在为 0。

SONET 帧

每一个同步传输信号 STS-n 由 8000 个帧组成，每一个帧是个 9 行 90 × n 列的二维矩阵。例如，STS-1 帧有 9 行、90 列（810 字节），一个 STS-3 是 9 行、270 列（2430 字节）。图 5-69 说明了 STS-1 帧以及 STS-n 帧的一般格式。

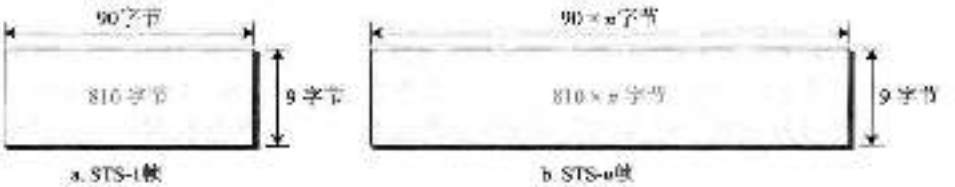


图 5-69　STS-1 和 STS-n 帧

帧、字节和位传输

SONET 的一个有趣的地方是每一个 STS-n 信号以每秒 8000 个帧的固定速率传输。这是数字化语音的速率（见第 7 章）。对于每一个帧，自左向右、自上向下的传输字节。对于每个字节，从高位到低位（从左向右）传输位。图 5-70 显示了帧和字节的传输顺序。

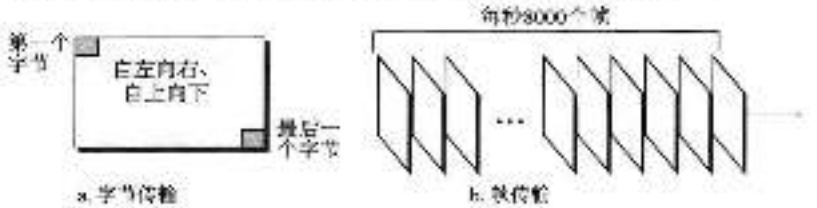


图 5-70　传输中的 STS-1 帧

SONET 中 STS-n 信号以每秒 8000 个帧传输。

如果我们对声音信号进行采样，每个采样使用 8 位（1 字节），我们可以说 SONET 帧中的每一个字节可以携带来自数字化语音通道中的信息。换言之，STS-1 信号能够同时携带 774 个语音通道（810 减去用于开销的 36）。

SONET 帧中的每个字节能够承载一个数字化语音通道。

注意，在 SONET 中不同的 STS 信号数据速率之间存在准确的关系。我们可以通过 STS-1 的数据速率得到 STS-3 的数据速率（前者乘以 3 即可）。

在 SONET 中，STS-n 信号的数据速率是 STS-1 信号数据速率的 n 倍。

STS-1 帧格式

STS-1 帧的基本格式如图 5-71 所示。如我们先前所述，SONET 帧是一个 9 行的矩阵，每行 90 个字节，总共 810 个字节。



图 5-71　STS-1 帧开销

STS 多路复用

在 SONET 中，较低速率的帧会同步地时分复用至一个更高速率的帧中。例如，三个 STS-1 信号（通道）可以组合为一个 STS-3 信号（通道），四个 STS-3 可以多路复用为一个 STS-12，等等。

多路复用是同步 TDM，网络中所有的时钟都锁定一个主时钟来实现同步。

在 SONET 中，网络的所有时钟都锁定在一个主时钟。

我们需要提到的是多路复用技术在更高数据速率时也可以进行。例如，4 路 STS-3 信号可以复用为一路 STS-12 信号。但是，STS-3 信号需要首先多路分解为 12 路 STS-1 信号，然后这 12 路信号复用为 1 路 STS-12 信号。在我们讨论完字节交叉后，做这项额外工作的原因就会清晰了。

分插多路复用器

几路 STS-1 信号复用为 1 路 STS-n 信号在 STS 多路复用器完成（在路径层）。STS-n 信号多路分解为 STS-1 部分在 STS 多路分解器完成。但是，在两者之间，SONET 使用分插复用器，它可以使用一个信号来代替另一个。我们需要知道在传统意义上这不是多路分解/复用。分插复用器在线路层运行。分插复用器不创建段、线路和路径开销。它几乎充当一个开关：它移走一路 STS-1 信号，添加另一路。在分插复用器的输入端和输出端，信号的类型是相同的（例如，都是 STS-3 或者 STS-12）。分插复用器（ADM）只是移除相应的字节并且用新的字节代替它们（包含在段和线路开销中的字节）。

SONET 网络

使用 SONET 设备，我们可以创建 SONET 网络，可以用作高速骨干网，从诸如 ATM 或是 IP 网络中运载负荷。我们可以粗略地将 SONET 网络分为三类：线状、环状和网状网络。

线状网络

线状网络通常由 STS 多路复用器和分解器，一些再生器和一些分插多路复用器组成，如图 5-72 所示。

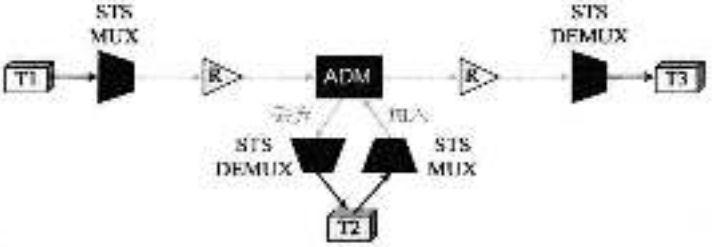


图 5-72　线状 SONET 网络

环状网络

ADM 使得 SONET 环状网络成为可能。SONET 环可以用于单向或者双向配置。在每种情况下，我们可以添加额外的环使得网络能自我修复线路故障。图 5-73 是一个环状网络。

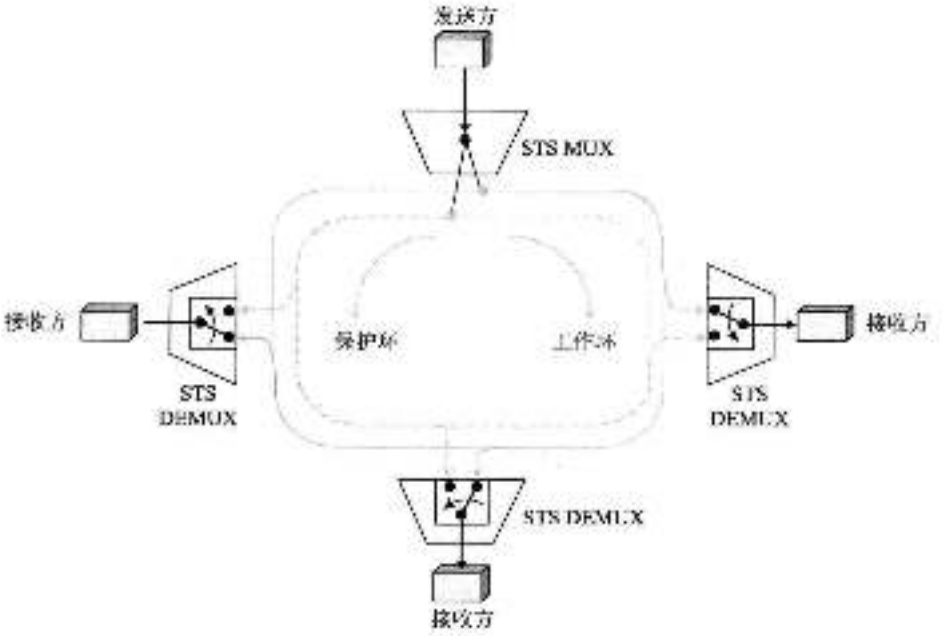
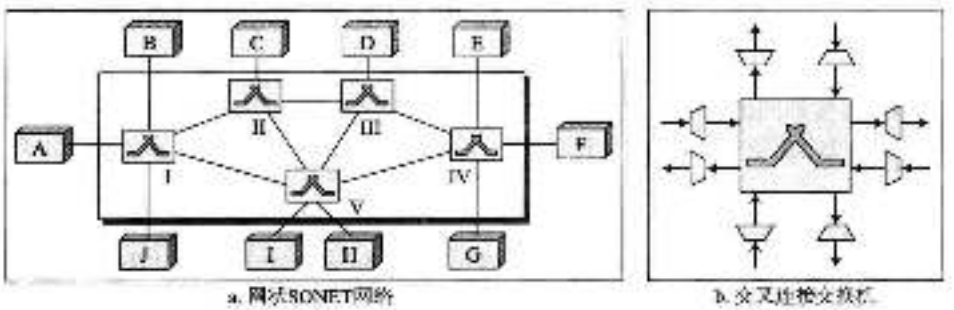


图 5-73　单向路径交换网

尽管在图中我们已经选择了一个发送方和二个接收方，但是可以有很多其他配置。发送方使用两路连接同时向两个环发送数据；接收方使用选择交换机来选择信号质量更好的环。我们使用 1 个 STS 多路复用器和 3 个 STS 分解器来强调工作在路径层的结点。

网状网络

环状网络的一个问题是缺乏可扩展性。当环中的流量增加时，我们不只需要升级线路也需要升级 ADM。在这种情况下，有多个交换机的网状网络有可能提供更好的性能。在网状网络中的交换机称为交叉连接，像其他我们见到的交换机一样，交叉连接有输入和输出端口。在输入端口，交换机接收 OC-n 信号，将其转换为 STS-n 信号，把它分解为相应的 STS-1 信号，并将每个 STS-1 信号发送至正确的输出端口。输出端口使用来自于不同输入端口的 STS-1 信号，将它们复用为 STS-n 信号，并生成 OC-n 信号传输。图 5-74 显示了一个网状 SONET 网络以及交换机的结构。



a. 网状SONET网络　　b. 交叉连接交换机

图 5-74　网状 SONET 网络

虚拟支路

SONET 用来承载宽带载荷。但是现在的数字体系的数据速率（DS-1 到 DS-3）比 STS-1 低。为了使 SONET 向后兼容当前的体系，它的帧设计包括了虚拟支路（virtual tributary，VT）系统（见图 5-75）。虚拟支路是能够插入到 STS-1 的部分载荷，并与其他部分载荷组合装入一个帧。对于来自于一个源的数据，我们将 SPE 分割开来，并称每一部分为 VT，而不是使用 STS-1 帧的所有的 86 个有效载荷列。

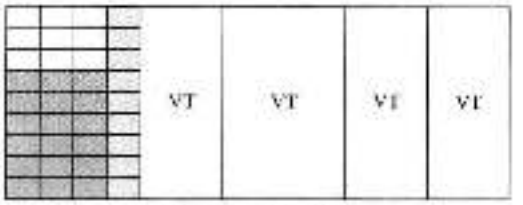


图 5-75　虚拟支路

已经定义了四种 VT 类型来适应现在的数字体系。注意，允许每类 VT 的列数与倍乘类型标识号绑定（VT1.5 有 3 列，VT2 有 4 列）。VT1.5 适应 U.S. DS-1 服务（1.544Mbps），VT2 适应欧洲 CEPT-1 服务（2.048Mbps）。VT3 适应 DS-1C 服务（部分 DS-1.3.152Mbps），VT6 适应 DS-2 服务（6.312Mbps）。

当两个或更多支路插入单个 STS-1 帧时，它们逐列交织。SONET 提供了确定每个 VT 并把它们分开而无需分解每个流的机制。这些机制的讨论以及其后的控制问题超出了本书的范畴。

5.6.3　交换网络：ATM

异步传输模式（Asynchronous Transfer Mode，ATM）是交换广域网，它基于由 ATM 论坛设计的信元中继（cell relay）协议，并被 ITU-T 采纳。ATM 和 SONET 的结合允许全世界网络的高速互联。事实上，ATM 可以被认为是信息超高速公路上的高速公路。

ATM 使用统计（异步）时分多路复用来处理来自不同通道的信元，这就是为什么它被称为异步传输模式（Asynchronous Transfer Mode）的原因。它使用固定大小的时隙（一个信元的大小）。ATM 多路复用器使用来自任意输入通道的信元来填充一个时隙；如果通道没有信元要发送，则时隙为空。图 5-76 显示了来自三个输入的信元如何进行多路复用的。在第一个时钟节拍，通道 2 没有信元（空的输入时隙），因此多路复用器使用来自第三个通道的信元填充时隙。当所有通道的所有时隙都被复用后，输出时隙为空。



图 5-76　ATM 多路复用

体系结构

ATM 是一个信元交换网络。用户访问设备称为端点，它们通过用户到网络接口（user-to-network interface，UNI）连接到网络内部交换机。交换机通过网络到网络接口（network-to-network interface，NNI）互联。图 5-77 是 ATM 网络的一个示例。

虚拟连接

在两个端点之间连接是通过传输路径（transmission path，TP）、虚拟路径（virtual path，VP）和虚电路（virtual circuit，VC）完成的。传输路径（TP）是一个端点和一台交换机之间或者是两台交换机之间的物理连接（有线、电缆、卫星等等）。将两台交换机看做是两个城市。传输路径是直接连接两个城市的所有高速公路的集合。

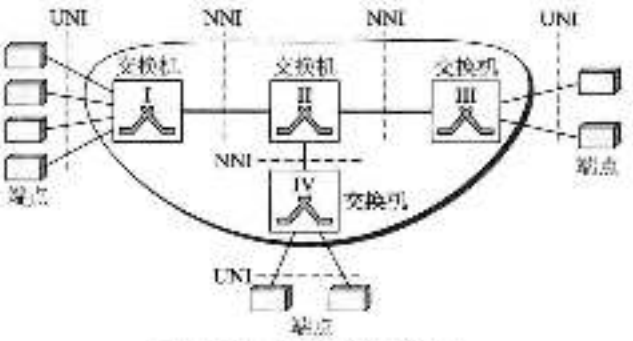


图 5-77　ATM 网络体系结构

传输路径划分成几个虚拟路径。一个虚路径（virtual path，VP）提供两个交换机之间的一条连接或一组连接。可以认为虚路径是连接两个城市的一条高速公路。每一条高速公路是一个虚路径；所有高速公路的集合是传输路径。

信元网络基于虚电路（VC）。属于同一报文的所有信元沿着同一条虚电路传输，并且保持它们的原始次序直到到达目的地。可以认为虚电路是高速公路（虚拟路径）的车道。图 5-78 显示了传输路径（物理连接）、虚路径（绑在一起的虚电路的集合，因为它们的部分路径是相同的）和虚电路（逻辑上连接两个端点）之间的关系。

标识符　在虚电路网络中，为了将数据从一个端点路由到另一个，就需要标识虚连接。为此，ATM 的设计者创建了一个两级的层次标识：虚路径标识符（virtual-path identifier，VPI）和虚电路标识符（virtual-circuit identifier，VCI）。VPI 定义了特定的 VP，而 VCI 定义了 VP 中的特定的 VC。VPI 对于所有绑定到（逻辑上）一个 VP 中的所有虚连接都是相同的。

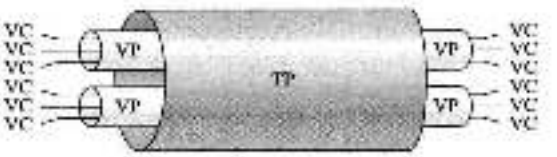


图 5-78　TP、VP 和 VC

UNI 和 NNI 的 VPI 的长度是不同的。在 UNI 中，VPI 是 8 位，然而在 NNI 中，VPI 是 12 位。在两种接口中 VCI 的长度是相同的（16 位）。因此我们可以说，在 UNI 中，虚连接由 24 位确定，而在 NNI 中由 28 位确定（见图 5-79）。

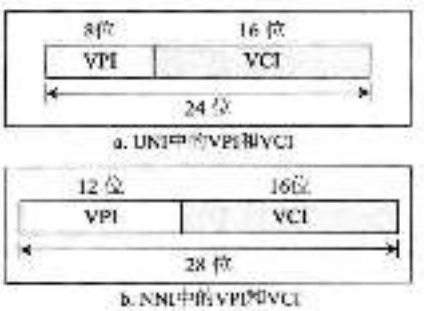


图 5-79　UNI 和 NNI 中的虚连接标识符

将虚电路标识符分割成两部分，其背后的整个思想是要允许层次路由。在典型 ATM 网络中，大部分交换机使用 VPI 来路由。而处于网络边界的交换机，它们直接和端点设备交互，使用 VPI 和 VCI 来路由。

信元　ATM 网络中基本数据单元称为信元。信元只有 53 字节长，5 个字节为头部，48 个字节携带有效负载（用户数据可能少于 48 字节）。大部分的头部被 VPI 和 VCI 占有，它们定义了虚连接。通过该虚连接，信元从一个端点传送到一个交换机，或者从一个交换机传送到另一个交换机。图 5-80 显示了信元的结构。

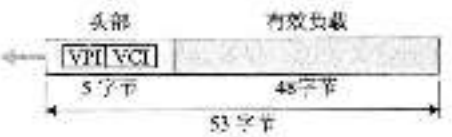


图 5-80　ATM 信元

连接建立与释放

ATM 使用两类连接：PVC 和 SVC。

PVC　永久虚电路连接（permanent virtual-circuit connection，PVC）由网络提供商在两个端点之间建立。VPI 和 VCI 是为了永久连接而定义的，它们的值输入到每一个交换机的表中。

SVC　在一个交换虚电路连接（switched virtual-circuit connection，SVC）中，每当一个端点想要和另一个端点连接时，就需要建立一条新的虚电路。ATM 不能独自完成该工作，需要网络层地址和另一个协议（如 IP）的服务。该协议的信令机制使用两个端点的网络层地址做出一个连接请求。实际的机制依赖于网络层协议。

交换

ATM 使用交换机将信元从源端点路由到目的端点。交换机使用 VPI 和 VCI 来路由信元。路由要求整个标识符。图 5-81 显示了 PVC 交换机如何路由信元。VPI 为 153 和 VCI 为 67 的信元到达交换机接口（端口）1。交换机检查它的交换表，该表每行存储六项信息：到达接口号，输入 VPI，输入 VCI 以及相应的输出接口号、新的 VPI 和新的 VCI。交换机找到接口号为 1、VPI 为 153、VCI 为 67 的实体，发现相应的输出接口为 3，VPI 为 153，VCI 为 92。它将头部的 VPI 和 VCI 改为 140 和 92，并通过接口 3 将信元发送出去。

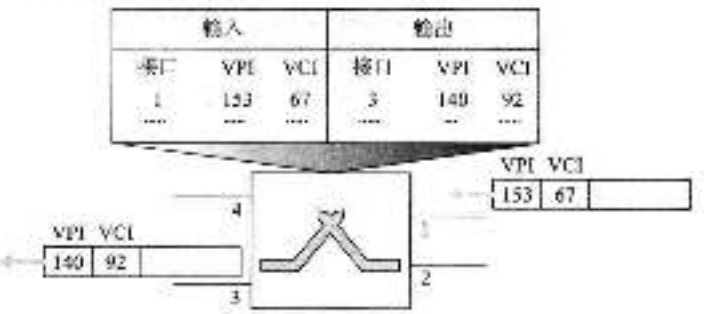


图 5-81　交换机的路由

ATM 标准定义了三层。自顶向下，它们依次是应用适配层、ATM 层和物理层（见图 5-82）。端点使用所有的三层，而交换机只使用两个底层。



图 5-82　ATM 层

AAL 层　应用适配层（application adaptation layer，AAL）用来支持两个 ATM 概念。首先，ATM 必须接收任意类型的负载，不论是数据帧还是位流。一个数据帧可以来自上层协议，这些协议创建了清晰地定义了的，要发送给承载网络（如 ATM）的帧。一个很好的例子是因特网。ATM 也必须承载多媒体有效负载。它可以接受连续位流并将它们分成块，然后在 ATM 层封装成信元。AAL 使用两个子层来完成这些任务。

无论数据是数据帧还是位流，负载必须被分段为 48 字节的报文段以由信元运载。在目的端，这些分段需要重组以创建原始负载。AAL 定义了一个称为分割与重组（segmentation and reassembly，

SAR）的子层来完成这个工作。在远端拆分；在目的端重组。

在 SAR 将数据拆分之前，必须保证数据的完整性。这由称为会聚子层（convergence sublayer，CS）的子层完成。

ATM 定义了四个版本的 AAL：AAL1、AAL2、AAL3/4 和 AAL5。我们这里只讨论 AAL5，它在现在的因特网中使用。

AAL5 子层为因特网应用而设计。它也称为简单有效适配层（simple and efficient adaptation layer，SEAL）。AAL5 假设属于单个报文的所有信元顺序地发送，并且控制功能已经包含在发送应用的上层中了。图 5-83 显示了 AAL5 子层。

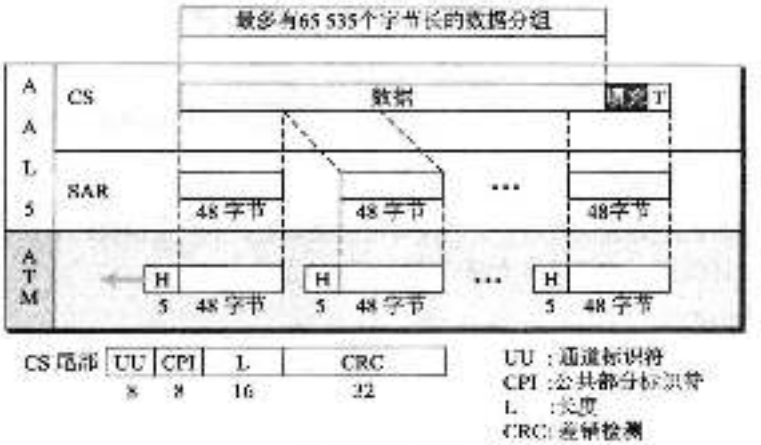


图 5-83　AAL5

在 CS 层的分组使用带有四个域的尾部。UU 是用户到用户标识符。CPI 是公共部分标识符。L 域定义了原始数据的长度。CRC 域是用于对整个数据单元差错检测的 4 个字节。

ATM 层　ATM 层提供路由、通信量管理、交换和多路复用服务。它按照如下方式处理输出信号：从 AAL 子层接收 48 字节分段，然后通过添加 5 字节的头部将它们转换成 53 字节信元。

物理层　像以太网和无线局域网，ATM 信元可以被任意物理层载体运载。

拥塞控制和服务质量

ATM 有非常成熟的拥塞控制和服务质量机制。

5.7　连接设备

主机和网络通常不是孤立的运行。我们使用连接设备将主机连接起来组成网络或者将网络连接起来组成互联网络。连接设备可以运行在因特网模型的不同分层。我们讨论三种连接设备：中继器（或集线器）、链路层交换机（或两层交换机）和路由器（或三层交换机）。中继器和集线器运行在因特网模型的第一层。链路层交换机和两层交换机运行在前两层。路由器和三层交换机运行在前三层。

5.7.1　中继器或集线器

中继器是只工作在物理层的设备。在一个网络中携带信息的信号在衰减到危及数据完整性之前，可以传输一段固定距离。中继器接收信号，在信号变得很弱或是被破坏之前，重新生成以及重新定时原始位模式。然后中继器发送新生成的信号。在过去，当以太网使用总线拓扑时，中继器用来连接局域网的两段来克服同轴电缆长度的限制。但是现在以太网使用星状拓扑。在星状拓扑中，中继器是多端口设备，经常称为集线器，可以作为连接点提供服务，同时有中继器的功能。图 5-84 说明当站点 A 到站点 B 的分组到达集线器时，代表该帧的信号重新生成以移除任何可能的破坏模

声，但是除了信号接收的端口，集线器从所有输出端口转发该分组。换言之，帧被广播了。局域网中所有的站点接收帧，但是只有站点 B 保留该帧。其余的站点丢弃该帧。图 5-84 说明了在交换局域网中中继器的角色。

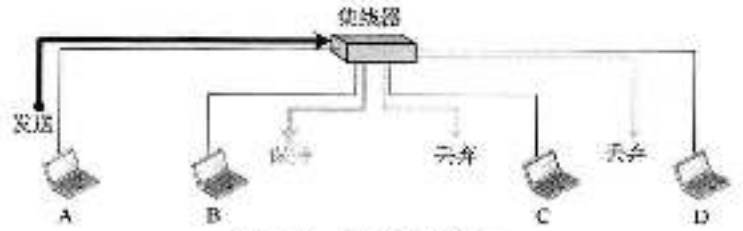


图 5-84　中继器或集线器

该图清楚地说明集线器没有过滤能力；它没有能力发现该帧应该从那个端口发送出去。

中继器没有过滤能力。

集线器或是中继器是物理层设备。它们没有链路层地址，不检查接收到的帧链路层地址。它们只是重新生成被破坏的位，并且从每个端口将它们发送出去。

5.7.2　链路层交换机

链路层交换机运行在物理层和数据链路层。作为物理层设备，它重新生成它接收到的信号。作为链路层设备，链路层交换机能够检查包含在帧中的 MAC 地址（源地址和目的地址）。

过滤

有人可能问链路层交换机和集线器之间有什么区别。链路层交换机有过滤能力。它可以检查帧的目的地址，决定帧该从哪个端口发送。

链路层交换机有一张过滤决策中使用的表。

让我们给出一个例子。在图 5-85 中，我们的局域网有四个站点，它们连接至一个链路层交换机。如果去往站点 71:2B:13:45:61:42 的帧到达端口 1，链路层交换机查询它的表来寻找离开端口。

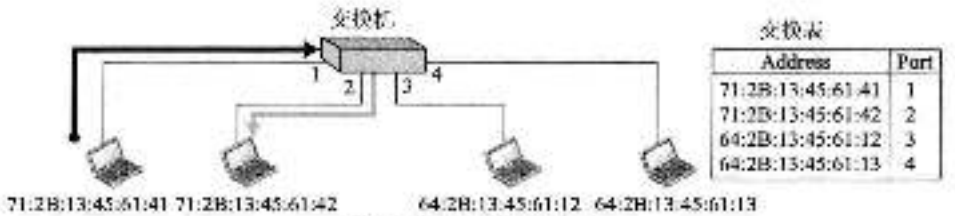


图 5-85　链路层交换机

根据它的表，到 71:2B:13:45:61:42 的帧应该只通过端口 2 发送出去；因此，没有必要通过其他端口转发帧。

链路层交换机不会改变帧中的链路层地址（MAC）。

透明交换机

一个透明交换机（transparent switch）是与其连接的站点完全意识不到其存在的交换机。如果在系统中添加或者移除一个交换机，站点的重新配置是没有必要的。根据 IEEE 802.1d 规范，安装有透明交换机的系统必须满足以下三个标准：

- 帧必须从一个站点转发至另一个站点。
- 通过学习网络中帧的传输，转发表可以自动建立。
- 必须避免系统内循环。

转发

如前一节讨论的那样，透明交换机必须正确地转发帧。

学习

最早的交换机的转发表是静态的。在配置交换机时，系统管理员需要手动地输入每个表的条目。尽管过程简单，但是并不实用。如果增加或移除了一个站点，该表就必须手动地修改。如果站点的 MAC 地址改变了，也是如此，这并不少见。例如，安装一个新网卡就意味着一个新的 MAC 地址。

比静态表更好的解决方法是使用自动映射地址到端口（接口）的动态表。为了动态生成转发表，我们需要交换机从帧传输中逐渐学习。要做到这一点，交换机要检查目的地址和源地址。目的地址用来做转发决定（查找表）；源地址用于向表中添加条目和更新目的。让我们使用图 5-86 来详细说明该过程。

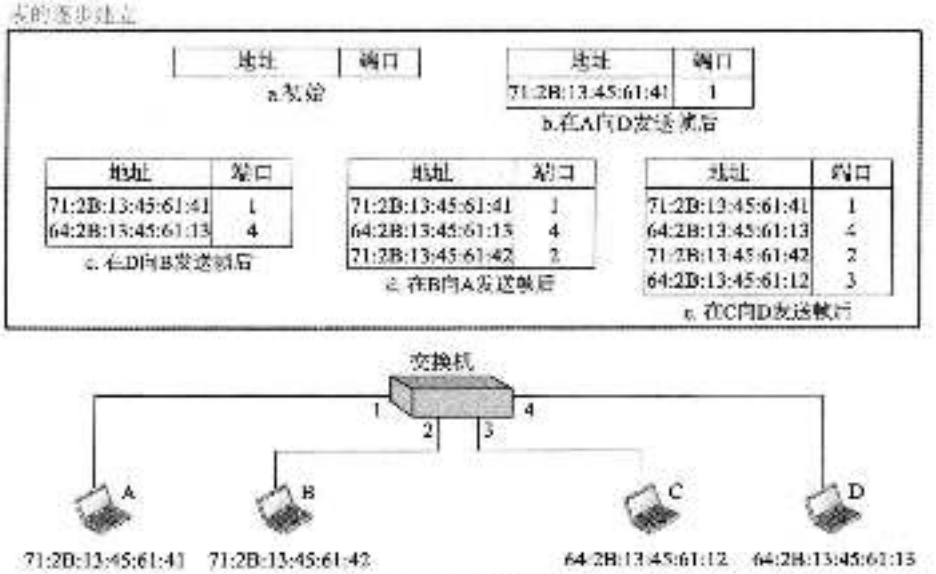


图 5-86　学习交换机

1. 当站点 A 向站点 D 发送帧时，交换机还没有关于 D 或 A 的表的条目。帧从所有的三个端口转发出去；帧在网络中泛洪。但是，通过查看源地址，交换机知道站点 A 一定与端口 1 相连。这就意味着去往站点 A 的帧将来必须通过端口 1 发送出去。交换机将该条目添加到它的表中，该表现在有了第一个条目。

2. 当站点 D 向站点 B 发送帧时，交换机没有 B 的条目，因此它再次泛洪。但是它再在表中增加关于站点 D 的条目。

3. 学习过程一直持续直至表中有关于每个端口的信息。

但是，注意学习过程可能花费很长时间。例如，如果站点不发送帧（很少见的情况），该站点永远不会在转发表中有条目。

5.7.3　路由器

我们在第 4 章中讨论过路由器（router）。在本章中，我们将路由器与两层交换机和集线器作比较。路由器是三层设备；它工作在物理层、数据链路层和网络层。作为物理层设备，它重新生成它

接收到的信号。作为数据链路层设备，路由器检查包含在分组中的物理地址（源地址和目的地址）。作为网络层设备，路由器检查网络层地址。

路由器是三层（物理层、数据链路层和网络层）设备。

路由器能够连接网络。换言之，路由器是网络互连设备；它将独立的网络连接以形成互联网络。根据该定义，由路由器连接的两个网络变为一个互联网络或是一个互联网。

在路由器和中继器或是交换机之间有三个主要的不同点：

1. 路由器每个端口都有一个物理地址和一个逻辑地址（IP）。
2. 路由器只对分组中链路层目的地址与分组到达的接口地址相匹配的那些分组起作用。
3. 当路由器转发分组时，它改变分组的链路层地址（源地址和目的地址）。

让我们给出一个示例。在图 5-87 中，假设一个组织有两栋独立的建筑，每栋建筑中安装了千兆以太网。该组织在每个局域网中使用交换机。这两个局域网可以相互连接，以形成更大的使用 10 千兆以太网技术的局域网，这加速了到以太网和该组织服务器的连接。然后一个路由器将整个系统连接至因特网。

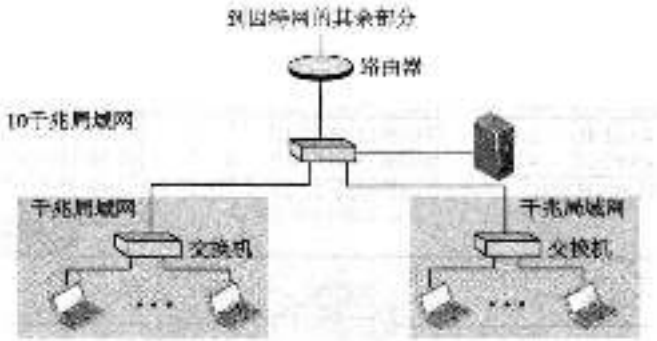


图 5-87　路由器示例

正如我们在第 4 章中所见，路由器将会改变它接收到的分组的 MAC 地址，因为 MAC 地址只有本地有效性。

路由器改变分组中的链路层地址。

5.8　章末资料

推荐读物

要获取本章中讨论的主题的更多详细信息，我们推荐下面的书。在方括号[...]中的项目涉及课文结尾的参考文献列表。

书籍

一些极好的书讨论了链路层问题。我们推荐[Ham 80]、[Zar 02]、[Ror 96]、[Tan 03]、[GW 02]、[For 03]、[KMK 04]、[Sta 04]、[Kes 02]、[PD 03]、[Kei 02]、[Spu 00]、[KCK 98]、[Sau 98]、[Izz 00]、[Per 00]和[WV 00]。

RFC

关于因特网中检验码的使用可以在 RFC1141 中找到。

小结

我们可以将数据链路层看做两个子层。上面子层负责数据链路控制，下面子层负责解决共享介

质的访问。数据链路控制（DLC）处理两个临近结点之间通信的设计和处理：结点到结点通信。该子层负责成帧和差错控制。差错控制处理传输中的数据损坏。我们在本章中讨论了两种链路层协议：HDLC 和 PPP。高级数据链路控制（HDLC）是点对点和多点链路通信中使用的面向位的协议。但是，点对点访问中最通用的协议是点对点协议（PPP），它是面向字节的协议。

很多正式的协议用来处理访问共享链路。我们将它们分为 3 类：随机访问协议、受控访问协议和通道化协议。在随机访问或是竞争方法中，没有站点优于其他站点，没有站点受另一个站点控制。在受控访问中，站点彼此查询以发现哪个站点有权发送帧。通道化是一种多路访问方法，其不同站点之间链路的可用带宽通过时间、频率或编码来分享。

在数据链路层，我们使用链路层寻址。系统通常使用地址解析协议（ARP）来获取下一个结点的链路层地址。

以太网是使用范围最广的本地局域网协议。以太网的数据链路层由 LLC 子层和 MAC 子层构成。MAC 子层负责 CSMA/CD 访问方法的操作以及成帧。以太网上的每一个站点有一个唯一的 48 位的地址，该地址刻印在站点的网络接口卡上（NIC）。虚拟局域网（VLAN）是由软件配置的而非物理线路配置的。VLAN 中的成员资格可以基于端口号、MAC 地址、IP 地址、IP 多播地址或是这些特征的组合。VLANS 是花费低、效率高的，它能够减少网络流量，提供额外的安全措施。

数字数据通信的需求导致了数号调制解调器的发明。因为高速下载和上传输率的需求，电话公司增加了一种新的技术，即数字用户线路（DSL）。有线网络最初是为更好的访问电视节目而创建的。通过使用一些最初为视频设计的通道，有线电视也在因特网访问供应中找到了很好的市场。同步光纤网络（SONET）是由 ANSI 设计推广的光纤网络标准。异步传输模式（ATM）是一种信元中继协议，和 SONET 组合允许高速连接。信元是一个很小的、固定长度的信息块。ATM 数据分组由 53 字节组成的信元。ATM 标准定义了三层：应用适配层（AAL）、ATM 层和物理层。

中继器是工作在因特网模型中物理层的互连设备。交换机是工作在因特网模型中物理层和数据链路层的设备。传输层交换机可以转发和过滤帧，并且自动地形成自己的转发表。路由器是工作在 TCP/IP 协议族中前三层的互连设备。

5.9　习题集

测试题

本章的交互测验题集可以在本书的网站上找到。强烈推荐学生们做这些测验题，这样可以在学生做习题集前来检测他们对课程资料的理解。

练习题

Q5-1　区分网络层通信和数据链路层通信。

Q5-2　区分点对点链路和广播链路。

Q5-3　解释当我们使用可变长度帧时，为什么需要标记。

Q5-4　解释为什么在面向字节的成帧中，我们无法使用位填充来区分出现在文本中的标记字节。

Q5-5　单个位差错和突发性差错有什么不同？

Q5-6　线性块编码的定义是什么？

Q5-7　在块编码中，一个数据字是 20 位，相应的代码字是 25 位。根据文中的定义，k、r、n 的值是多少？多少冗余位要添加到该数据字？

Q5-8　在代码字中，我们为每个 8 位数据字添加 2 位冗余位。计算下面要求的数目：

a. 有效代码字。　　b. 无效代码字。

Q5-9　什么是最小汉明距离？

Q5-10　如果我们想要能够检测 2 位差错，最小汉明距离应是多少？

Q5-11　一类差错检测（纠错）码称为汉明码。其中 $d_{min}=3$。这种编码能够检测多达 2 个差错（或是纠正一个

单个位差错）。在这种编码中，n、k、r 之间关系为：$n = 2^r - 1$，$k = n - r$。如果 $r = 3$，计算数据字和代码字的位数。

Q5-12 在 CRC 中，如果数据字是 5 位，代码字是 8 位，需要在数据字中添加多少个 0 来形成被除数？余数的长度是多少？除数的长度是多少？

Q5-13 在 CRC 中，下列哪个生成器（除数）可以保证检测到一个单个位差错？

a. 101　　b. 100　　c. 1

Q5-14 在 CRC 中，下列哪个生成器（除数）可以保证检测到一个奇数位差错？

a. 10111　　b. 101101　　c. 111

Q5-15 在 CRC 中，我们已经选择了生成器为 1100101，突发性差错长度为以下值时，能够被检测出来的概率是多少？

a. 5?　　b. 7?　　c. 10?

Q5-16 假设我们正在发送长为 16 位的数据项。如果在传输过程中两个数据项交换了位置，传统的校验码能否检测到这个差错？解释其原因。

Q5-17 传统校验和的值能否全是 0（二进制）？解释原因。

Q5-18 解释 Fletcher 算法（见图 5-18）在计算校验和时，如何给数据项添加权值？

Q5-19 解释 HDLC 帧中为何只有一个地址域（而非两个地址域）？

Q5-20 下列哪个是随机访问协议？

a. CSMA/CD　　b. Polling　　c. TDMA

Q5-21 纯 Aloha 网络中的站点以 1Mbps 的速率发送长度为 1000 位的帧。该网络的脆弱时间是多少？

Q5-22 $G = 1/2$ 的纯 Aloha 网络中，吞吐量在以下情况中受到什么影响？

a. G 增加到 1　　b. 减少到 1/4

Q5-23 为了理解图 5-40 中 K 的使用，在以下情况中，计算一个站点能够立即发送的可能性。

a. 失败一次后　　b. 失败 3 次后

Q5-24 基于图 5-30，我们如何解释 Aloha 网络中的成功？

Q5-25 假设广播网络中的广播延迟为 5μs，帧传输时间为 10 μs。

a. 第一位到达目的地需要多长时间？

b. 第一位到达后，最后一位到达目的地需要多长时间？

c. 网络中与该帧相关的时间是多少（易发生冲突）？

Q5-26 假设广播网络中传播延迟是 3 μs，帧传输时间是 5 μs。无论冲突在哪里发生，它是否能被检测到？

Q5-27 不同网络中的两个主机能否有相同的链路层地址？

Q5-28 解释为何冲突在随机访问网络中是一个问题，但是在受控访问或者通道化协议中却不是。

Q5-29 当我们使用由电话公司提供的 DSL 服务来访问因特网时，我们是否需要多路访问协议？为什么？

Q5-30 ARP 分组的长度是否是固定的？解释原因。

Q5-31 当协议是 IPv4，硬件地址是以太网地址时，ARP 分组的长度是多少？

Q5-32 承载 28 字节的 ARP 分组的以太网帧的长度是多少？

Q5-33 以太网帧中，前导域如何与 SFD 域区分？

Q5-34 NIC 的意义是什么？

Q5-35 为何全双工以太网局域网中，不需要 CSMA/CD？

Q5-36 比较标准以太网、快速以太网、千兆以太网和 10 千兆以太网的数据速率。

Q5-37 适用标准以太网的实现是什么？

Q5-38 适用千兆以太网的实现是什么？

Q5-39 拨号调制解调器技术是什么？列举一些本章中讨论过的适用调制解调器标准，给出它们的速率。

Q5-40 比较传统有线网络和混合光纤同轴网络。

Q5-41 一个传统以太网中，4 个站点连接至一个集线器，站点与集线器之间的距离分别为：300 m、400 m、500 m 和 700 m。当我们计算 T_p 时，网络的长度是多少？

Q5-42 当我们说链路层交换机能够过滤流量时，意味着什么？为什么过滤很重要？

Q5-43 VLAN 如何节省公司的时间和金钱？

Q5-44　VLAN 如何减少网络流量？

Q5-45　为什么 SONET 称为同步网络？

Q5-46　讨论每一个 SONET 分层的功能。

Q5-47　在 ATM 中，TP、VP 和 VC 之间是什么关系？

思考题

P5-1　对下面的帧负载进行字节填充，其中 E 是转义字节，F 是标记字节，D 是非转义字节和标记字节的数据字节。

D	E	D	D	F	D	D	E	E	D	F	D

P5-2　对下面的帧负载进行位填充：

00011111110011111010001111111111000011

P5-3　以下面的速率传输数据时，2-ms 的突发噪声的最大影响是什么？

a．1500 bps　　b．12 kbps　　c．100 kbps　　d．100 Mbps

P5-4　假设数据单元中一位在传输过程中被破坏的概率为 p，计算以下情况中，n 位数据单元中 x 位被破坏的概率：

a．$n = 8, x = 1, p = 0.2$　　b．$n = 16, x = 3, p = 0.3$　　c．$n = 32, x = 10, p = 0.4$

P5-5　"异或"是计算代码字中最常用的操作之一。对下面的模式对执行异或操作，解释结果。

a．(10001) ⊕ (10001)　　b．(11100) ⊕ (00000)　　c．(10011) ⊕ (11111)

P5-6　表 5-1 中，发送方发送数据字 10，一个 3 位的突发差错破坏了代码字。接收方是否能够检测到该差错？解释原因。

P5-7　使用表 5-2 中的编码，如果接收到下列代码字，数据字分别是什么？

a．01011　　b．11111　　c．00000　　d．11011

P5-8　证明以下代码字代表的编码不是线性的，你需要发现一种违反线性的情况。

{(00000), (01011), (10111), (11111)}

P5-9　下面代码字的汉明距离是多少？

a．d(10000,00000)　　b．d(10101,10000)　　c．d(00000,11111)　　d．d(00000,00000)

P5-10　尽管可以正式证明表 5-3 中的编码是线性的、循环的，但是我们只用两个测试来部分地证明该事实：

a．通过代码字 0101100 测试循环性。　　b．通过代码字 0010110 和 1111111 测试线性。

P5-11　通过表 5-4 中的 CRC-8，回答以下问题：

a．它是否检测一个单个位差错？解析原因。　　b．它是否检测长度为 6 的突发性差错？解释原因。

c．检测到长度为 9 的突发性差错的概率是多少？　　d．检测到长度为 15 的突发性差错的概率是多少？

P5-12　假设偶校验码，计算下列数据单元的奇偶校验位。

a．1001011　　b．0001100

c．1000000　　d．1110111

P5-13　简单的奇偶校验检查通常添加到字的末尾（将一个 7 位的 ASCII 字符转变为一个字节），它无法检测偶数个差错。例如 2、4、6、8 个差错就无法通过这种方式检测。更好的方法是以表的形式组织字符，创建行奇偶校验位和列奇偶校验位。行奇偶校验位随着字节发送，列奇偶校验位作为额外的字节发送（见图 5-88）。

	C1	C2	C3	C4	C5	C6	C7	
R1	1	1	0	0	1	1	1	1
R2	1	0	1	1	1	0	1	1
R3	0	1	1	1	0	0	1	0
R4	0	1	0	1	0	0	1	1
	0	1	0	1	0	1	0	1

行奇偶校验位

列奇偶校验位

Rn: 第 n 行
Cm: 第 m 列

图 5-88　思考题 P5-13

说明下面的差错如何被检测：

a．(R3, C3)发生一个差错。

b．(R3, C4)和 (R3, C6)发生两个差错。

c．(R2, C4)、(R2, C5)和 (R3, C4)发生三个差错。

d．(R1, C2)、(R1, C6)、(R3, C2)和(R3, C6)发生4个差错。

P5-14　给定数据字 1010011111，除数 10111，给出发送端站点（使用二进制除法）CRC 代码字的生成器。

P5-15　假设一个分组由 4 个 16 位字 $(A7A2)_{16}$、$(CABF)_{16}$、$(903A)_{16}$ 和 $(A123)_{16}$ 组成。手动模拟图 5-17 的算法来计算校验和。

P5-16　传统校验和计算需要以反码运算完成。现在的计算机和计算器设计成以补码做运算。一种计算传统校验和的方式是以补码运算将数相加，将结果除以 2^{16} 获取商和余数，将商和余数相加获取反码表示的和。校验和可以通过用 $2^{16}-1$ 减去上述和获得。使用上面的方法来获取以下 4 个数字的校验和：43 689、64 463、45 112 和 59 683。

P5-17　本题目展示了校验和处理中的一种特殊情况。一个发送方有两个数据项要发送：$(4567)_{16}$ 和 $(BA98)_{16}$。校验和的值是多少？

P5-18　手动模拟 Fletcher 算法（见图 5-18）来计算以下字节的校验和：$(2B)_{16}$、$(3F)_{16}$、$(6A)_{16}$ 和 $(AE)_{16}$。也要说明该结果是加权校验和。

P5-19　手动模拟 Adler 算法（见图 5-19）来计算以下字的校验和：$(FBFF)_{16}$ 和 $(EFAA)_{16}$。也要说明该结果是加权校验和。

P5-20　作为一个实际的例子，计算图 5-89 中数据报的校验和域的值。由于 IP 数据报的校验和只计算头部，所以我们只给出这些域的值。

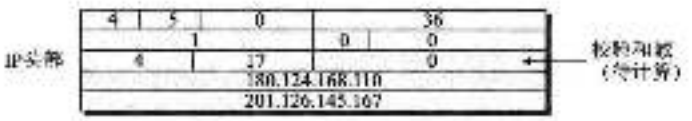


图 5-89　思考题 P5-20

P5-21　假设带有如图 5-90 所示头部的数据报已经到达。校验和的值是 65 207。检验传输过程中头部是否被破坏。

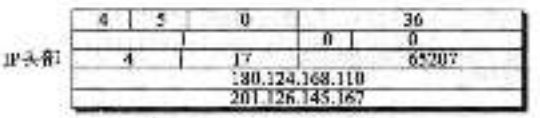


图 5-90　思考题 P5-21

P5-22　一个加权校验和的例子是 ISBN-10 编码，我们可以在一些书的背面见到。在 ISBN-10 中，有 9 个十进制数字定义了国家、出版商和书。第 10 个数字（最右边的）是一个校验数字。编码 $D_1D_2D_3D_4D_5D_6D_7D_8D_9C$，满足以下条件：

$$[(10 \times D_1) + (9 \times D_2) + (8 \times D_3) + \cdots + (2 \times D_9) + (1 \times C)] \bmod 11 = 0$$

换言之，权值是 10、9、…、1。如果 C 的计算值是 10，就使用字母 X 代替。通过使用权值 w 以 11 为模（11-w）的补码代替权值 w，可以得到校验数字可以按以下方式计算：

$$C = [(1 \times D_1) + (2 \times D_2) + (3 \times D_3) + \ldots + (9 \times D_9)] \bmod 11$$

计算 ISBN-10: 0-07-296775-C 中的校验数字。

P5-23　ISBN-13 编码是 ISBN-10 的新的版本，也是 13 个数字的加权校验和的另一个例子，其中有 12 个十进制数字定义了书的情况。最后的数字是校验数字。编码 $D_1D_2D_3D_4D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}C$ 满足以下条件：

$$[(1 \times D_1) + (3 \times D_2) + (1 \times D_3) + \cdots + (3 \times D_{12}) + (1 \times C)] \bmod 10 = 0$$

换言之，权值是 1 和 3 之一。利用上面的描述，计算 ISBN-13: 978-0-07-296775-C 的校验数字。

P5-24　为了规定多路访问网络的性能指标，我们需要一个数学模型。当网络中站点的数目非常多时，使用泊松分布 $p[x] = (e^{-\lambda} \times \lambda^x)/x!$。在该公式中，$p[x]$ 是一段时间内生成 x 个帧的概率，λ 是相同时间内生成帧的平均数目。使用泊松分布：

a. 计算纯 Aloha 网络中，在脆弱时间内生成 x 个帧的概率。注意该网络中的脆弱时间是帧传输时间（T_f）的两倍。

b. 计算在时隙 Aloha 网络中，在脆弱时间内生成 x 个帧的概率。注意该网络中的脆弱时间与帧传输时间（T_f）相等。

P5-25　在前一个题目中，我们使用泊松分布 $p[x] = (e^{-x} \times x^x)/x!$ 来计算纯 Aloha 或是时隙 Aloha 网络中一段确定时间内生成 x 个帧的概率。本题中，我们想要计算在这样的网络中一个帧不和其他帧冲突就到达目的地的概率。为此，认为有 G 个站点，每个站点在帧传输时间内平均发送 1 个帧（而非有 N 个帧，每个站点在相同时间内平均发送 G/N 个帧）。然后，一个站点成功的概率就是在脆弱时间内没有其他站点发送帧的概率。

a. 计算在纯 Aloha 网络中，在脆弱时间内一个站点能够成功发送帧的概率。

b. 计算在时隙 Aloha 网络中，在脆弱时间内一个站点能够成功发送帧的概率。

P5-26　在前一个题目中，我们计算了一个站点（G 个站点的网络中）在脆弱时间内成功发送一个帧的概率，其在纯 Aloha 网络中为 $P = e^{-2G}$，在时隙 Aloha 中为 $P = e^{-G}$。本题中，我们想要计算这些网络的吞吐量，即任意站点（G 个站点之外）在脆弱时间内能够成功发送帧的概率。

a. 计算纯 Aloha 网络中的吞吐量。

b. 计算时隙 Aloha 网络中的吞吐量。

P5-27　在前一个题目中，我们说明纯 Aloha 网络的吞吐量 $S = Ge^{-2G}$，时隙 Aloha 网络的吞吐量 $S = Ge^{-G}$。本题中我们想要计算使得吞吐量最大的 G 的值，并计算该最大吞吐量。如果我们计算 S 对 G 的导数并设定导数为 0，那么很容易完成该工作。

a. 计算使吞吐量最大的 G 的值，并且计算纯 Aloha 网络的最大吞吐量的值。

b. 计算使吞吐量最大的 G 的值，并且计算时隙 Aloha 网络的最大吞吐量的值。

P5-28　有大量站点的多路访问网络可以使用泊松分布来分析。当网络中有有限个站点时，我们需要使用另一种方法来分析。在有 N 个站点的网络中，我们假设每个站点以概率 p 在传输时间（T_b）内发送一个帧。在这样的网络中，如果一个站点在脆弱时间内要发送帧，而没有其他站点在此段时间内要发送帧，那么该站点能成功发送该帧。

a. 计算纯 Aloha 网络中，一个站点在脆弱时间内能够成功发送帧的概率。

b. 计算时隙 Aloha 网络中，一个站点在脆弱时间内能够成功发送帧的概率。

P5-29　在前一个题目中，我们计算了一个站点在脆弱时间内成功发送帧的概率。有限个站点网络的吞吐量是任意站点（除 N 个站点）成功发送帧的概率。换言之，吞吐量是 N 个成功概率之和。

a. 计算纯 Aloha 网络的吞吐量。

b. 计算时隙 Aloha 网络的吞吐量。

P5-30　在前一个题目中，我们计算了纯 Aloha 网络的吞吐量为 $S = Np(1-p)^{2(N-1)}$，时隙 Aloha 网络的吞吐量 $S = Np(1-p)^{(N-1)}$。在本题中，我们想要计算关于 p 的最大吞吐量。

a. 计算使得纯 Aloha 网络吞吐量最大的 p 的值，并计算当 N 非常大时该最大吞吐量。

b. 计算使得时隙 Aloha 网络吞吐量最大的 p 的值，并计算当 N 非常大时该最大吞吐量。

P5-31　时隙 Aloha 网络中只有三个有效站点：A、B 和 C。每一个站点以相应的概率在一个时隙内生成一个帧，概率分别为 $p_A = 0.2$、$p_B = 0.3$ 和 $p_C = 0.4$。

a. 每个站点的吞吐量是多少？

b. 网络的吞吐量是多少？

P5-32　时隙 Aloha 网络中只有三个有效站点：A、B 和 C。每一个站点以相应的概率在一个时隙内生成一个帧，概率分别为 $p_A = 0.2$，$p_B = 0.3$ 和 $p_C = 0.4$。

a. 任意站点在第一个时隙发送帧的概率是多少？

b. 站点 A 在第二个时隙第一次能够成功发送帧的概率是多少？

c. 站点 C 在第三个时隙第一次能够成功发送帧的概率是多少？

P5-33　一个时隙网络以最大吞吐量工作。

a. 一个时隙是空的概率是多少？　　b. 在得到一个空时隙之前平均要经过多少个时隙？

P5-34　局域网中的一个有用的参数是 1m 介质的位数（$n_{bit/m}$）。如果数据速率是 100Mbps，介质广播速率是 2×10^8 m/s，计算 $n_{bit/m}$ 的值。

P5-35　局域网中另一个有用的参数是介质的位长度（L_b），其定义了任意时间介质可以持有的位数。如果数据速率是 100Mbps，两个站点之间通信的介质长度（L_m）是 200m，计算局域网的位长度。假设介质的广播速率为 2×10^8 m/s。

P5-36　我们已经定义了参数 a 作为能两个站点之间介质的帧的数目，$a = (T_p)/(T_t)$。另一种定义该参数的方法是 $a = L_p/F_b$，其中 L_b 是介质的位长度，F_b 是介质的帧长度。说明这两种定义是等价的。

P5-37　在数据速率为 10Mbps 的总线 CSMD 网络中，在帧的第一位离开发送站点后 20μs 发生了一次冲突。该帧的长度是多少，才能使得发送站点检测到该冲突？

P5-38　假设在总线 CSMA/CD 网络中只有两个站点 A 和 B，两个站点之间的距离是 2000m，广播速率是 2×10^8 m/s。如果站点 A 在 t_1 时刻开始传输：

a. 协议是否允许站点 B 在时刻 $t_1 + 8$ μs 开始传输？如果答案是允许，将会发生什么？

b. 协议是否允许站点 B 在时刻 $t_1 + 11$ μs 开始传输？如果答案是允许，将会发生什么？

P5-39　在总线 1-持续 CSMA/CD 网络中只有两个站点 A 和 B，其中 $T_p = 25.6$ μs，$T_t = 51.2$ μs。站点 A 要向站点 B 发送一个帧。该帧失败了两次，第三次尝试时成功了。画一个该问题时序图。假设 R 分别为 1 和 2，忽略发送阻塞信号的时间（见图 5-40）。

P5-40　为了理解在 CDMA/CD 网络中，为何我们需要最小帧长度 $T_t = 2 \times T_p$。我们假设在只有两个站点 A 和 B 的总线网络中，$T_p = 40$ μs，$T_t = 25$ μs。站点 A 在时刻 t = 0.0 μs 开始发送一个帧，站点 B 在时刻 $t = 23.0$μs 开始发送一个帧。回答下面问题：

a. 帧是否会冲突？

b. 如果 a 的答案是会，站点 A 是否检测到冲突？

c. 如果 a 的答案是会，站点 B 是否检测到冲突？

P5-41　在总线 1-持续 CSMA/CD 中，$T_p = 50$ μs，$T_t = 120$ μs，有两个站点 A 和 B。这两个站点同时向彼此发送帧。由于帧冲突了，每个站点尝试重传。站点 A 开始时 R = 0，站点 B 开始时 R = 1。忽略包含发送阻塞信号在内的其他延迟。这些帧会再次发生冲突吗？画时序图来证明你的说法。这种情况下，随机数的产生能否帮助避免冲突？

P5-42　随机变量 R（见图 5-40）用来在冲突发生时，给不同的站点不同的延迟。为了减少冲突，我们期望不同的站点产生不同的 R 值。为了说明这一点，计算以下情况发生后，R 的值相同的概率：

a. 第一次冲突发生后　　b. 第二次冲突发生后

P5-43　假设我们有一个对称 CSMA/CD 网络。该网络中的每一个站点使用一个竞争周期，该周期中站点在能够发送帧之前，竞争访问共享介质。我们假设竞争周期由竞争时隙组成。在每个时隙的开始，站点检测通道。如果通道是空闲的，站点发送它的帧；如果通道时忙碌的，站点抑制发送并等到下一个时隙的开始。换言之，站点在发送帧之前，平均等待 k 个时隙，如图 5-91 所示。注意通道或者处于竞争状态、传输状态，或者处于空闲状态（当没有站点要发送帧时）。但是，如果 N 是非常大的数字，空闲状态实际消失了。

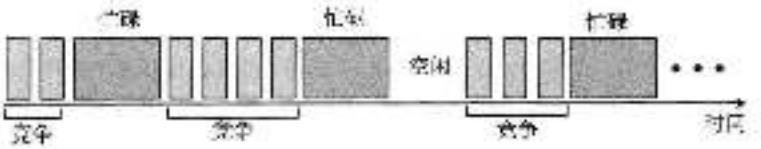


图 5-91　思考题 P5-43

a. 如果站点的数目是 N，每个站点以概率 p 发送帧，空闲时隙（P_{idle}）的概率是多少？

b. 当 N 是非常大的数时，概率的最大值是多少？

c. 第 j 个时隙空闲的概率是多少？

d. 站点在得到空闲时隙前，平均要等待多少个时隙（k）？

e. 当 N（站点数目）非常大时，k 的值是多少？

P5-44　尽管 CDMA/CD 的吞吐量计算是很复杂的，但是使用前一个题目中我们描述的，我们能够计算时隙 CDMA/CD 的最大吞吐量。我们发现一个站点需要等待的竞争时隙的平均数目 $k = e$。根据该假设，时隙 CDMA/CD 的吞吐量为

$$S = (T_f)/(通道为帧忙碌的时间)$$

通道为帧忙碌的时间是指等待空闲时隙的时间、传输帧的时间以及接收到没有冲突发生的好消息的广播延迟。假设竞争时隙的持续时间是 $2 \times (T_p)$，$a = (T_p)/(T_t)$。注意参数 a 是占据传输介质的帧的数目。根据参数 a 计算时隙 CDMA/CD 的吞吐量。

P5-45　我们有一个数据速率为 10Mbps 的纯 Aloha 网络。通过该网络成功发送 1000 位帧的最大数目是多少？

P5-46　在数据速率为 10Mbps 的 CDMA/CD 网络中，为了冲突检测进程的正确工作，最小帧长度为 512 位。如果我们保持该网络常数，但是增加数据速率至以下情况时，最小帧长度是多少？

a．100 Mbps?　　b．1 Gbps?　　c．10 Gbps?

P5-47　下列以太网地址的十六进制等价形式是什么？

01011010 00010001 01010101 00011000 10101010 00001111

P5-48　在线路上以太网地址 1A:2B:3C:4D:5E:6F 如何以二进制形式出现？

P5-49　如果以太网目的地址是 07:01:02:03:04:05，该地址是什么类型的（单播、多播或广播）？

P5-50　在 CSMA/CD 网络中有两个站点 A 和 B。假设站点 A 和 B 分别以概率 p_1 和概率 p_2 在冲突间隔($2 \times t_p$)内发送帧。如果 p_1 为 0.3，p_2 为 0.4，回答以下问题：

a．站点 A 成功的概率是多少？　　b．站点 B 成功的概率是多少？　c．一个帧成功的概率是多少？

P5-51　IPv4 地址为 125.45.23.12，以太网地址为 23:45:AB:4F:67:CD 的一个路由器已经接收到目的 IP 地址为 125.11.78.10 的分组。说明由路由器发送的 ARP 请求分组中的条目。并且要将该分组封装为一个以太网帧。

P5-52　在图 5-30 中，说明当帧从 Bob 发送至 Alice 时，Bob 站点的活动。

P5-53　一个以太网 MAC 子层从上层接收到 42 字节的数据。必须向该数据填充多少字节？

P5-54　最小以太网帧中有效数据占整个分组的比率是多少？

P5-55　假设 10Base5 电缆的长度是 2500m。如果粗同轴电缆中广播的速率是 200 000 000 m/s，一位从网络的开始到网络的终点需要经过多长时间？假设设备里有 10 μs 的延迟。

P5-56　当一个区域的顾客使用 DSL 调制解调器来进行数据传输时，使用什么拓扑结构？解释使其原因。

P5-57　一个链路层交换机使用一个过滤表；一个路由器使用一个转发表。你能解释它们的不同点吗？

5.10　模拟实验

Applets

我们已经创建了一些 Java 小程序来展示本章讨论的主要概念。强烈建议学生们激活本书网站上的这些 applet，仔细研究这些协议的运作方式。

实验作业

本书中，我们使用 Wireshark 来模拟两个协议：Ethernet 和 ARP。这些试验的完整说明参见本书网站。

Lab5-1　在本实验中，我们需要检查由数据链路层发送的帧的内容。我们想要查找不同域的值，如目的 MAC、源 MAC、CRC 值、描述帧运载负载类型的协议域的值等。

Lab5-2　在本实验中，我们需要检查 ARP 分组的内容。我们想要抽获 ARP 请求和 ARP 应答分组。要检查的有趣的域是那些说明分组中使用的源地址和目的地址类型的域。

5.11　编程作业

使用你熟悉的编程语言实现下面的几个作业。

Prg5-1　编写并测试模拟图 5-5 中所示的字节填充和字节恢复的程序。

Prg5-2　编写并测试模拟图 5-7 中所示的位填充和位恢复的程序。

Prg5-3　编写并测试模拟图 5-17 中所示流程图的程序。

Prg5-4　编写并测试模拟图 5-18 中所示流程图的程序。

Prg5-5　编写并测试模拟图 5-19 中所示流程图的程序。

第 6 章

Computer Networks: A Top-Down Approach

无线网络和移动 IP

在第 5 章中我们讨论了有线网络。本章我们介绍无线网络。我们将讨论几项无线技术，包括无线局域网以及其他无线网络，它们跨越了蜂窝电话网、卫星和无线接入网。

正如在本章中我们所见到的，无线局域网的性质不同于有线局域网。因特网中其他的无线技术的使用正在逐渐增加，例如蜂窝电话网和卫星。作为无线技术的组成部分，我们讨论一下蜂窝电话网和卫星。我们也将在本章中用一节来介绍移动 IP：移动主机 IP 协议的使用。

无线技术事实上覆盖了 TCP/IP 协议簇中的数据链路层和物理层。我们主要关注与数据链路层相关的问题，将与物理层相关的问题推迟到第 7 章。

- 6.1 节介绍无线局域网，并将其与我们第 5 章中讨论的有线局域网相比较。然后讨论无线局域网中的主要标准 IEEE 项目 802.11。接下来我们讨论蓝牙局域网，它在很多应用程序中作为独立的局域网来使用。最后我们讨论一下 WiMAX 技术，它是诸如 DSL 电缆等最后一英里有线网络的竞争对手。
- 6.2 节讨论其他的无线网络，这些无线网络可以分为无线广域网和无线宽带网。我们首先讨论用于蜂窝电话的信道访问方法，然后我们讨论蜂窝电话。我们也会涉及和因特网连接相结合的卫星。
- 6.3 节涵盖了移动 IP，它提供了移动接入互联网的方法。我们首先讨论寻址，这是移动网络中的一个很大的议题。然后我们讨论移动接入的三个阶段。最后我们讨论移动 IP 中的低效率问题。

6.1 无线局域网

无线通信是增长最快的技术之一。不使用电缆来连接设备的需求正到处增长。无线局域网可以在大学校园中、在办公大楼里以及很多公共场所找到。

6.1.1 介绍

在讨论与无线局域网相关的特定协议之前，让我们先大体上谈论一下它们。

架构比较

让我们首先比较一下有线局域网和无线局域网的架构，以明确当我们研究无线局域网时需要了解的一些概念。

介质

我们发现，有线局域网和无线局域网的第一个不同点是介质。在有线局域网中，我们使用电缆来连接主机。在第 5 章中，我们看到，通过各代以太网，我们从多路访问迁移到点对点访问。在一个交换局域网中，使用链路层交换机，主机之间的通信是点对点的和全双工的（双向的）。在一个无线局域网中，介质是空气，信号通常是广播的。当无线局域网中的主机彼此通信时，它们共享相同的介质（多路访问）。极少数情况下，我们可以通过使用非常有限的带宽和双向天线来创建两个无线主机之间的点对点通信。但是，本章中我们的讨论是关于多路访问介质的，这意味着我们需要使用 MAC 协议。

主机

在有线局域网中，一个主机总是通过与其网络接口卡（NIC）相关的固定的链路层地址与它的网络在某一点相连。当然，一台主机可以从因特网中的一点移动到另一点。这种情况下，它的链路层地址仍然是相同的，但是它的网络层地址将会改变，如我们将后所见（见移动 IP 一节）。但是，在主机能够使用因特网的服务之前，它需要物理地连接至因特网。在无线局域网中，主机不必物理地连接至网络；它可以自由地移动（如我们将要看到的）并且能够使用网络提供的服务。因此，有线网络和无线网络中的移动性是完全不同的事情。本章中我们将尝试阐明这一点。

隔离局域网

有线隔离局域网的概念也不同于无线隔离局域网。有线隔离局域网是通过链路层交换机相互连接的一组主机（在近代的以太网中）。无线隔离局域网在无线局域网技术中称为自组织网络（ad hoc network），是彼此自由通信的一组主机。在无线局域网中不存在链路层交换机的概念。图 6-1 展示了两个隔离局域网，一个有线的和一个无线的。

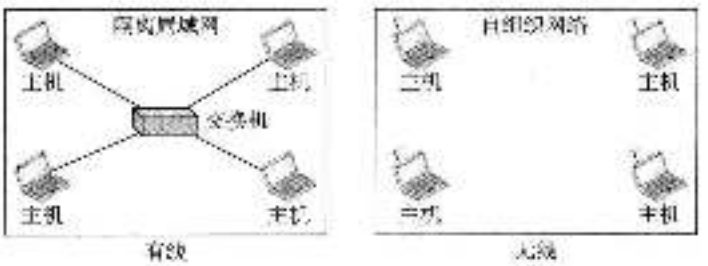


图 6-1　隔离局域网：有线与无线

连接至其他网络

使用路由器，一个有线局域网可以连接至另一个网络或是像因特网一样的一个互联网络。一个无线局域网可以连接至一个有线基础设施（infrastructure）网络、无线基础设施网络或是另一个无线局域网。第一种情况是我们本节要讨论的；无线局域网到有线基础设施网络的连接。图 6-2 展示了这两种情况。

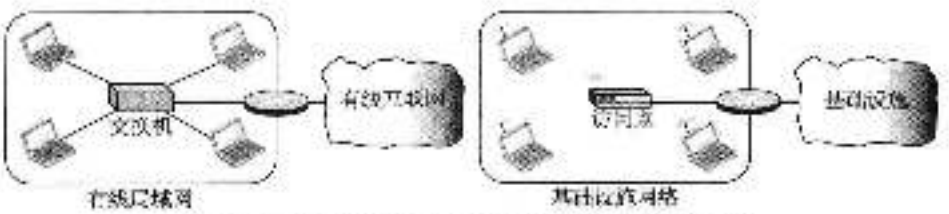


图 6-2　有线局域网及无线局域网到其他网络的连接

这种情况下，无线局域网称为基础设施网络（infrastructure network），到诸如因特网等有线基础设施网络的连接通过称为访问点（access point，AP）的设备完成。注意访问点的角色完全不同于有线环境中链路层交换机的角色。一个访问点将两种不同的环境黏合在一起：一边是有线环境，一边是无线环境。AP 和无线主机之间的通信发生在无线环境中；AP 和基础设施之间的通信发生在有线环境中。

环境之间的移动

以上讨论又回到了前一章中我们学到的：有线局域网或是无线局域网只是运行在 TCP/IP 协议栈中的下面两层。这意味着如果我们在一条连接中有一个有线局域网，该有线局域网通过路由器或是

调制解调器连接至因特网，为了从有线环境移动至无线环境，所有我们需要做的就是将为有线网络环境设计的网络接口卡换为为无线环境设计的，并用访问点代替链路层交换机。在这种变化中，链路层地址将会改变（因为改变了 NIC），但是网络层地址（IP 地址）仍然相同；我们正从有线链路移动至无线链路。

特性

无线局域网有几个特性，这些特性不适用于有线局域网，或是可以忽略。这里我们讨论一些特性来为讨论无线局域网协议做准备。

衰减

因为电磁信号分散在所有的方向，因此电磁信号的强度会快速降低；只有很少的一部分到达接收方。对于使用电池并且通常很少有能量供给的移动发送方来说，情况会变得更糟。

干扰

另一个问题是接收方可能不止从预期的发送方那里接收信号，如果其他发送方也使用相同的频带，那么也可能从它们那里接收信号。

多条路径广播

接收方可能从相同的发送方处接收到不止一个信号，因为电磁波可以从墙壁、地面或其他物体处反射回来。结果造成接收方在不同的时刻接收到一些不同的信号（因为它们经过不同的路径）。这就使得信号不易辨认。

差错

由于无线网络的上述特性，我们可以预料无线网络中的差错和差错检测是比有线网络中更加严重的问题。如果将差错水平考虑为测量信噪比（signal to noise ratio ，SNR），我们就可以更好地理解为什么无线网络中的差错检测、差错纠正以及重传更加重要。我们将在第 7 章中更详细地讨论 SNR，但是足以说它测量了好坏东西（信号噪声）的比率。如果 SNR 高，就意味着信号比噪声（无用信号）要强，这样我们就能够将信号转换为实际数据。另一方面，当 SNR 低时，就意味着信号被噪声破坏，数据不可恢复。

访问控制

也许无线局域网中我们需要讨论的最重要的问题是访问控制——无线主机如何访问共享介质（空气）。我们在第 5 章中讨论标准以太网使用 CSMA/CD 算法。在该方法中，每一个主机竞争访问介质，并在发现介质空闲时发送它的帧。如果冲突发生了，它被检测到，然后帧再次被发送。CSMA/CD 中的冲突检测有两个目的。如果检测到了冲突，意味着帧还没有被接收，需要重发。如果没有检测到冲突，这是帧被接收的一种确认。CSMA/CD 算法在无线网络中因三个原因不发挥作用：

1. 为了检测到冲突，主机需要同时发送和接收（发送帧和接收冲突信号）。这意味着主机需要以全双工模式工作。无线主机没有足够的电量这样做（由电池供电）。它们只能在同一时间发送或接收。

2. 由于隐藏站点的问题，冲突可能发生了但是没有检测到。隐藏站点问题是指一个站点可能由于一些障碍物或是范围问题而不知道另一个站点的传输。图 6-3 是隐藏站点问题的一个例子。站点 B 有一个传输范围，在左边椭圆中显示（在球体空间中）；该范围中的每一个站点可以听到由站点 B 传输的任意信号；站点 C 有一个传播范围，在右边椭圆中显示（在球体空间中）；处于该范围中的每一个站点能够听到站点 C 传输的任意信号。站点 C 在站点 B 的传输范围之外；同样地，站点 B 在站点 C 的传输范围之外。但是，站点 A 在 B 和 C 都覆盖的范围中；它可以听到 B 或 C 传输的任意信号。该图也说明隐藏站点问题可能由于障碍物而发生。

假设站点 B 正在向站点 A 发送数据。在该传输中间，站点 C 也有数据要向站点 A 发送。但是站点 C 在 B 的范围之外，来自 B 的传输不能到达 C。因此 C 认为介质是空闲的。站点 C 向 A 发送它的数据，因站点 A 正接收来自于 B 和 C 的数据，这就导致在 A 发生冲突。在这种情况下，我们

说站点 B 和 C 对于 A 来说是相互隐藏的。因为冲突的可能性，隐藏站点降低了网络的能力。

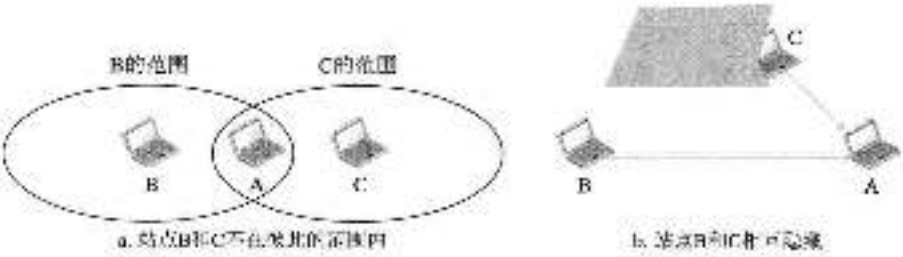


图 6-3　隐藏站点问题

3. 站点之间的距离可以是很大的。信号衰减可以阻止一端的站点听到另一端的冲突。

为了克服上述三个问题，为无线局域网发明了带有冲突避免的载波侦听多路访问（Carrier Sense Multiple Access with Collision Avoidance，CSMA/CA）。我们稍后讨论。

6.1.2　IEEE 802.11 项目

IEEE 已经定义了无线局域网的规格说明，称为 IEEE 802.11，它包含了物理层和数据链路层。在包含美国在内的一些国家，人们使用术语 WiFi（无线保真的缩称）作为无线局域网的同义词。然而，WiFi 是一个无线局域网，它由 WiFi 联盟授权。WiFi 联盟是一个全球性的非营利的工业联盟，有超过 300 家公司成员致力于无线局域网的发展。

体系结构

标准定义了两种服务类型：基本服务集（BSS）和扩展服务集（ESS）。

基本服务集

IEEE 802.11 将**基本服务集**（Basic Service Set，BSS）定义为无线局域网的构建块。一个基本服务集由固定的或是移动的无线站点和一个可选的称为访问点（Access Point，AP）的中央基站组成。图 6-4 说明了这一标准的两个集合。

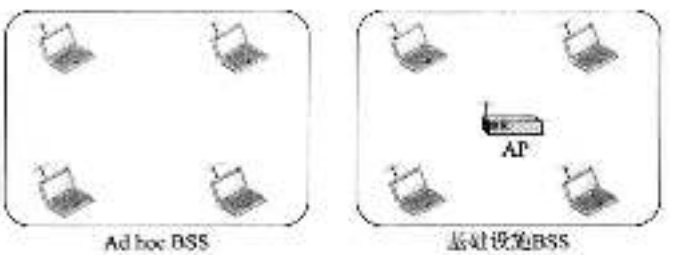


图 6-4　基本服务集（BSS）

不带 AP 的 BSS 是一个独立的网络，不能向其他 BSS 发送数据。它称为自组织结构（ad hoc architecture）。在这种体系结构中，站点不需要 AP 就能形成一个网络；它们能够相互定位并允许是 BSS 的一部分。带 AP 的 BSS 有时称为基础设施（infrastructure）BSS。

扩展服务集

扩展服务集（Extended Service Set，ESS）由两个或是更多个带有 AP 的 BSS 组成。在这种情况下，BSS 通过一个分布式系统连接，该系统通常为一个有线网络或是无线网络。分布式系统连接 BSS 中的 AP。IEEE 802.11 没有严格限制分布式系统；它可以是任意的 IEEE 局域网，如以太网等。注意，扩展服务集使用两种类型的站点：移动的和固定的。移动站点是 BSS 内的普通站点。固定站点则是作为有线局域网一部分的 AP 站点。图 6-5 展示了一个 ESS。

当 BSS 相互连接时，互相可达的站点之间可以相互通信而不必使用 AP。但是，BSS 内部站点和 BSS 外部的站点之间通过 AP 通信。如果我们将每一个 BSS 看做一个信元，将每个 AP 看做一个基站，那么这一概念与蜂窝网络中的通信类似（稍后讨论）。注意，一个移动站点可以同时属于多个 BSS。

站点类型

根据站点在无线局域网中的移动性，IEEE 802.11 定义了三种类型的站点：不迁移（no-transition）、BSS 迁移（BSS-transition）和 ESS 迁移（ESS-transition）。具有不迁移移动性的站点或者是固定的（不移动的），或者是只在 BSS 内部移动的站点。具有 BSS 迁移移动性的站点可以从一个 BSS 移动至另一个，但是其移动被限制在一个 ESS 内。具有 ESS 迁移移动性的站点可以从一个 ESS 移动至另一个 ESS。但是，IEEE 802.11 不能保证通信在移动中是连续的。

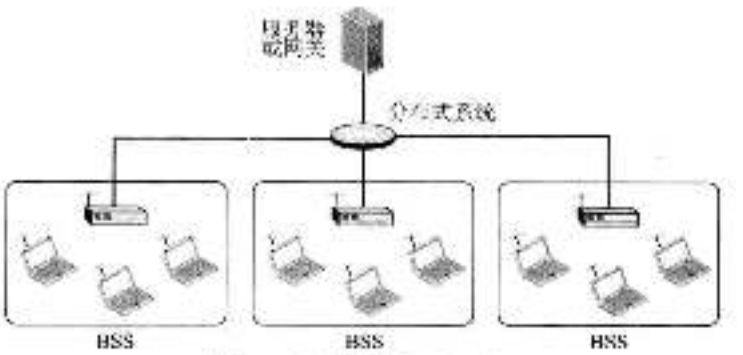


图 6-5　扩展服务集（ESS）

MAC 子层

IEEE 802.11 定义了两个 MAC 子层：分布式协调功能（distributed coordination function，DCF）和点协调功能（point coordination function，PCF）。图 6-6 显示了两个 MAC 子层、LLC 子层和物理层之间的关系。我们稍后在本章中讨论物理层，现在集中讨论 MAC 子层。



图 6-6　IEEE 802.11 标准中的 MAC 层

分布式协调功能

IEEE 在 MAC 子层定义的两种协议之一称为**分布式协调功能**（distributed coordination function，DCF）。DCF 使用 CSMA/CA 作为访问方法。

CSMA/CA　在无线网络中，因为冲突不能被检测到，因此我们需要避免冲突，带有冲突避免的载波侦听多路访问即为这种网络而发明。通过 CSMA/CA 的三种策略：帧间隔、竞争窗口和确认避免了冲突，如图 6-7 所示。我们稍后讨论 RTS 和 CTS 帧。

- **帧间隔（IFS）**。首先，即使发现通道是空闲时也推迟传输，这样冲突可以避免。当一个空闲通道被发现时，站点不立即发送。它等待称为帧间隔（interframe space，IFS）的一段时间。即使当通道被侦听时它是空闲的，一个远处的站点也可能已经开始传输。远处站点的信号还没有到达该站点。IFS 时间允许由远处站点传输的信号的前端到达该站点。在等待 IFS

时间后，如果通道仍是空闲的，站点可以发送了，但是仍然需要等待一段竞争时间（下面描述）。IFS 变量也可以用于给站点或者帧类型按优先顺序排队。例如，被指定较短 IFS 的站点有较高优先级。

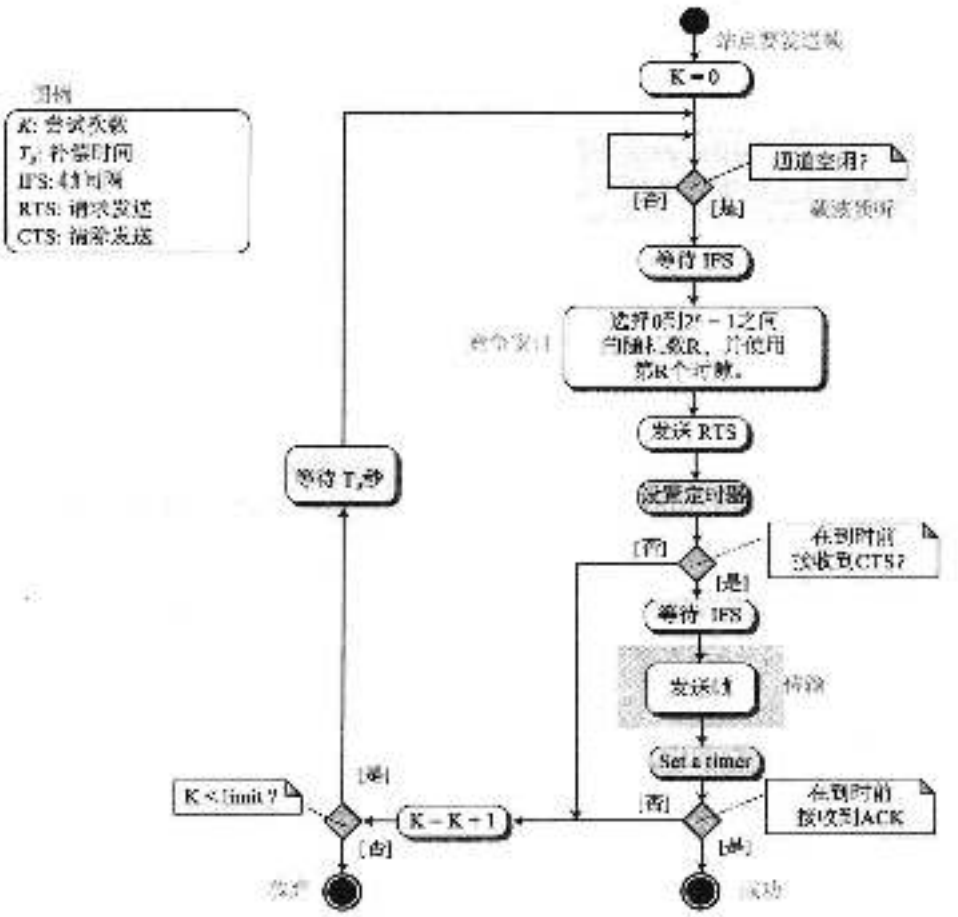


图 6-7　CSMA/CA 的流程图

- **竞争窗口**。竞争窗口是时间分为时隙（slot）的数量。准备发送的站点选择随机数目的时隙数作为它的等待时间。窗口中的时隙数根据二进制指数后退策略发生变化。这意味着第一次时它被设置为一个时隙，然后每当站点在 IFS 时间后没检测到空闲通道时就翻倍。除了由等待的站点随机定义的时隙数之外，其他的和 p-持续方法很像。竞争窗口中很有趣的一点是每一个时隙后站点都必要检测通道。但是，如果站点发现通道忙，它不重新启动该进程；它只是停止定时器，当检测到通道空闲时重启定时器。这给予了等待最长时间的站点优先权。见图 6-8。

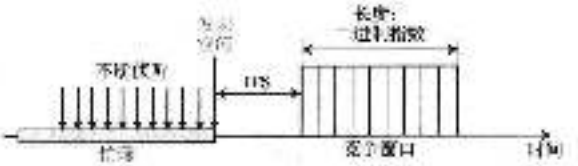


图 6-8　竞争窗口

- **确认**。尽管使用了所有的这些措施，仍然有可能发生冲突，导致数据破坏。另外，数据有可能在传输过程中被破坏。正式确认和超时定时器能够帮助保证接收方已经接受了该帧。

帧交换时序　图 6-9 说明了数据和超时控制帧的交换。

1. 在发送帧前，源主机通过检测载波频率的能量侦听介质。

a. 在通道空闲之前，通道使用带有补偿的持续策略。

b. 站点发现通道空闲之后，站点等待一段称为分布式帧间间隔（DCF interframe space，DIFS）的时间周期，然后发送一个称为请求发送（RTS）的控制帧。

2. 接收到 RTS 后，等待一段称为短帧间间隔（short interframe space，SIFS）的短暂时间后，目的地站点就向源站点发送一个称为清除发送（CTS）的控制帧。该控制帧表明目的站点准备接收数据。

3. 源主机在等待了 SIFS 相等的时间后发送数据。

4. 目的地站点在等待与 SIFS 相等的时间后，发送一个确认来说明帧已经收到了。在这个协议中，确认是需要的，因为站点没有任何方法来检查数据是否已经成功到达目的地站点。另一方面，在 CSMA/CD 中，没有冲突对源站点来说就意味着数据已经到达了。

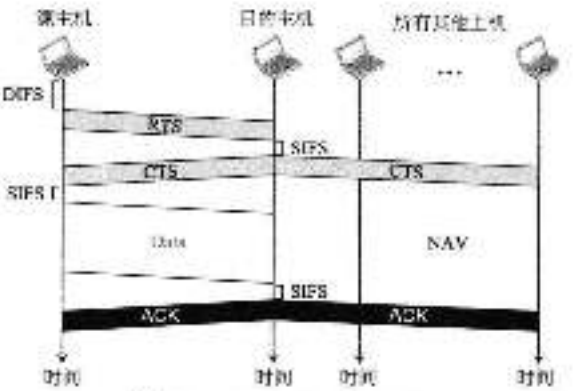


图 6-9　CSMA/CA 和 NAV

网络分配矢量　如果一个站点获得访问，那么其他站点如何推迟发送它们的数据？换言之，协议中的冲突避免是如何实现的？关键是称为 NAV 的特性。

当站点发送一个 RTS 帧时，它包含需要占用通道的时间。受该传输影响的站点创建一个称为**网络分配矢量**（network allocation vector，NAV）的定时器，该定时器指出在这些站点检测通道是否空闲之前还必须要经过多长时间。每次站点访问系统并且发送 RTS 帧时，其他站点开启它们的 NAV。换言之，每个站点在检查物理介质是否空闲之前，首先要检查它的 NAV 是否过期。图 6-9 说明了 NAV 的概念。

握手时的冲突　如果 RTS 或是 CTS 控制帧在传输时（这一段时间通常称为握手周期（handshaking period））发生了冲突，会发生什么情况？两个或更多的站点可能会同时发送 RTS 帧。这些控制帧有可能发生冲突。但是，由于没有冲突检测的机制，发送方认为如果它没有收到接收方的 CTS 帧就发生了冲突。补偿策略就被使用，发送方再次尝试。

隐藏站点问题　隐藏站点问题的解决方法是使用握手帧（RTS 和 CTS）。图 6-3 也说明了 B 的 RTS 信息到达 A，但是没有到达 C。但是，因为 B 和 C 在 A 的范围内，包含从 B 到 A 的数据传输持续时间的 CTS 信息到达 C。站点 C 知道某个隐藏站点正使用该通道，并在持续周期结束之前抑制传输。

点协调功能（PCF）

点协调功能（point coordination function，PCF）是一种可以在基础设施网络中（不在 ad hoc 网络中）实现的可选的访问方式。它在 DCF 上实现，主要用于对时间敏感的传输。

PCF 有一个集中式的、无竞争的轮询访问方式，我们曾在第 5 章中讨论过。AP 对那些可以被轮询的站点进行轮询。站点依次被轮询，将它们的任意数据发送给 AP。

为了给予 PCF 高于 DCF 的优先级，定义了另一种帧间间隔 PIFS。PIFS（PCF IFS）比 DIFS 短。这意味着，如果一个站点想要只使用 DCF，而 AP 想要使用 PCF，那么 AP 有优先权。

由于 PCF 的优先级高于 DCF，只使用 DCF 的站点可能得不到对介质的访问。为了避免这种情况，设计了重复间隔来覆盖无竞争 PCF 和基于竞争的 DCF 的通信。重复间隔（repetition interval）不断地重复，并始于一个称为**信号帧**（beacon frame）的特殊控制帧。当站点侦听到信号帧时，在重复间隔的无竞争周期内，它们开始它们的 NAV。图 6-10 展示了重复间隔的一个例子。

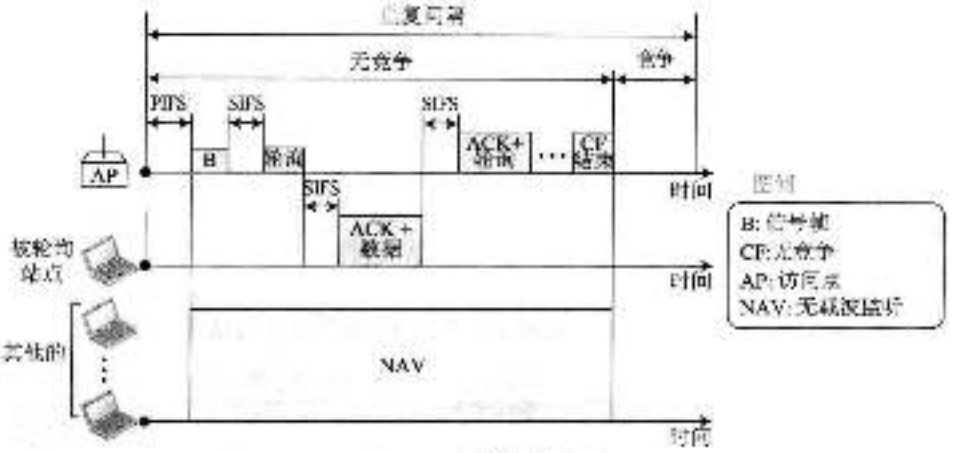


图 6-10　重复间隔例子

在重复间隔中，PC（点控制方）能够发送轮询帧、接收数据、发送 ACK、接收 ACK，或者做任何这些动作的组合（802.11 使用捎带）。在无竞争周期结束时，PC 发送 CF 结束帧（无竞争结束），以允许基于竞争的站点使用介质。

分段

无线环境噪声很多，因此帧经常被破坏。被破坏的帧不得不被重传。因此协议要求分段，即将一个很大的帧分为多个更小的帧，重传很小的帧比更大的帧更加有效。

帧格式

MAC 层由 9 个域组成，如图 6-11 所示。

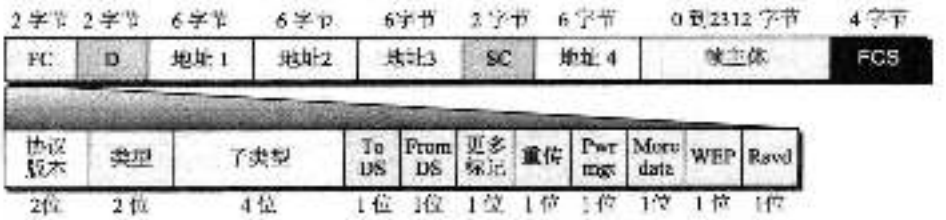


图 6-11　帧格式

- **帧控制**（FC）。FC 域 2 字节长，定义帧的类型和一些控制信息。表 6-1 描述了它的子域。我们稍后在本章中讨论每一种帧类型。
- **D**。该域定义了用于设置 NAV 值的传输间隔时间。在一个控制帧中，它定义了帧的 ID。
- **地址**。有四个地址域，每个 6 字节长。每个地址域的意义依赖于 To DS 和 From DS 子域的值，稍后讨论。
- **序列控制**。该域经常称为 SC 域，定义了一个 16 位的值。前 4 位定义了片段序号；最后的 12 位定义了序列号。在所有片段中序列号相同。
- **帧主体**。该域长度可以在 0 到 2312 字节之间，包含了根据 FC 域中定义的类型和子类型的信息。

• **FCS**。FCS 域长度为 4 字节，包含一个 CRC-32 的差错检测序列。

表 6-1　FC 域的子域

域	解　释	域	解　释
版本	当前版本是 0	重传	置 1 意味着重传帧
类型	信息类型：管理（00）、控制（01）或数据（10）	Pwr mgt	置 1 意味着处于电源管理模式
子类型	每个类型的子类型（见表 6-2）	更多数据	置 1 意味着站点有更多的数据被发送
To DS	稍后定义	WEP	有线对等保密（实现加密）
From DS	稍后定义	Rsvd	保留
更多标记	置 1 意味着更多分段		

帧类型

由 IEEE 802.11 定义的无线局域网有三种类型的帧：管理帧、控制帧和数据帧。

管理帧　管理帧用于站点和接入点之间的初始通信。

控制帧　控制帧用于访问通道和对帧的确认。图 6-12 说明了它的格式。



图 6-12　控制帧

对于控制帧类型域的值是 01；帧的子类型域的值如表 6-2 所示。

数据帧　数据帧用于携带数据和控制信息。

表 6-2　控制帧中子类型域的值

子 类 型	含　义
1011	请求发送（RTS）
1100	请求发送（CTS）
1101	确认（ACK）

寻址机制

IEEE 802.11 寻址机制说明了四种情况，由 FC 域中的两个标记值（To DS 和 From DS）定义。每个标记可以是 0 或 1，这样会有四种情况出现。MAC 帧中的 4 个地址（地址 1 到地址 4）的解释取决于这些标记的值，如表 6-3 中所示。

表 6-3　地址

To DS	From DS	地址 1	地址 2	地址 3	地址 4
0	0	目的地址	源地址	BSS ID	N/A
0	1	目的地址	发送 AP	源地址	N/A
1	0	接收 AP	源地址	目的地址	N/A
1	1	接收 AP	发送 AP	目的地址	源地址

注意，地址 1 总是帧将访问的下一个设备的地址。地址 2 总是帧离开的前一个设备的地址。如果地址 1 没有定义最后的目的地址，地址 3 是最后的目的站点的地址；如果地址 2 没有定义原始源地址，地址 3 是原始源站点地址。当分布式系统也是无线的，地址 4 是原始源地址。

情况 1：00　这种情况下，To DS = 0，From DS = 0。这意味着帧既不是发往一个分布式系统（To DS = 0）也不是来自一个分布式系统（From DS = 0）。该帧来自于 BSS 中的一个站点，去往另一个站点，而不用通过分布式系统。地址如图 6-13 所示。

情况 2：01　这种情况下，To DS = 0，From DS = 1。这意味着该帧来自于一个分布式系统（From DS = 1）。帧来自于一个 AP，去往一个站点。地址如图 6-13 所示。注意，地址 3 包含帧的原始发送方地址（在另一个 BSS 中）。

情况 3：10　这种情况下，To DS = 1，From DS = 0。这意味着该帧要去往一个分布式系统（To DS = 1）。该帧来自于一个站点，去往一个 AP。ACK 发送至原始站点。地址见图 6-13。注意，地址 3 包含分布式系统中帧的最终目的地址。

情况 4：11　这种情况下，To DS = 1，From DS = 1。此时，分布式系统也是无线的。在无线分布式系统中，该帧来自于一个 AP 去往另一个 AP。这里，我们需要 4 个地址来定义原始发送方、最终目的地和两个中间 AP。图 6-13 说明了这种情况。

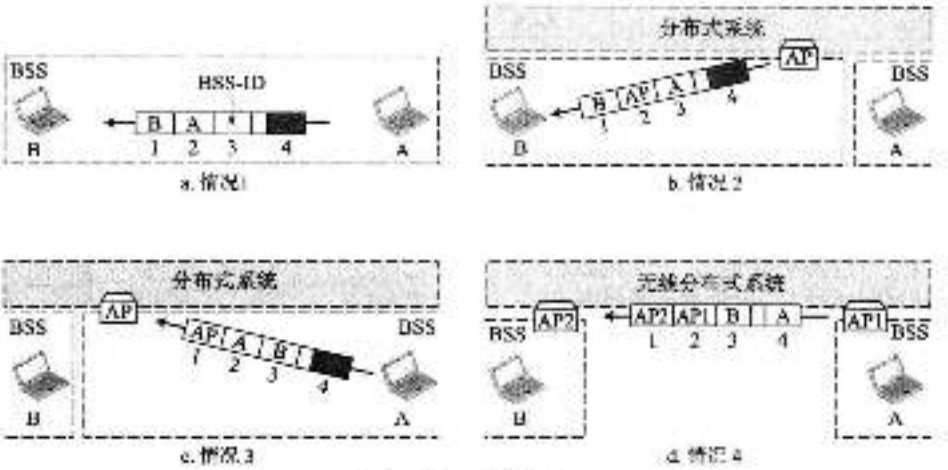


图 6-13　寻址机制

暴露站点问题

我们讨论了如何解决隐藏站点问题。与之相似的一个问题是暴露站点问题。该问题中，当通道可用时，站点也限制使用通道。在图 6-14 中，站点 A 向站点 B 发送。站点 C 有些数据要向站点 D 发送，可以直接发送而不会干扰到 A 到 B 的传输。但是，站点 C 暴露给了 A 的传输；它侦听到 A 正在发送内容，因此抑制发送。换言之，C 太保守，浪费了通道的能力。握手信息 RTS 和 CTS 在这种情况下起不了作用。站点 C 侦听来自于 A 的 RTS，抑制发送，即使 C 和 D 之间的通信不会导致 A 和 C 之间区域的冲突；站点 C 无法知道站点 A 的传输不会影响 C 和 D 之间的地区。

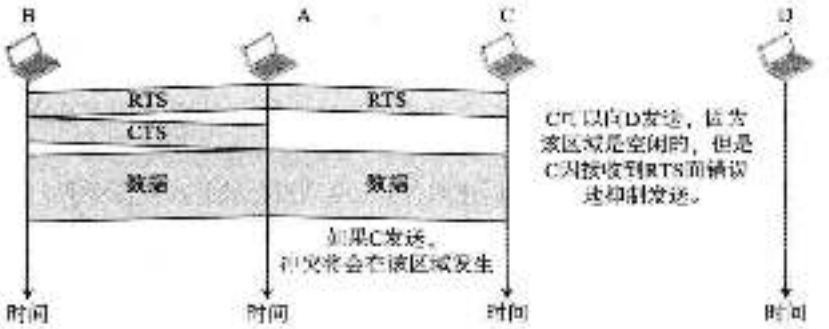


图 6-14　暴露站点问题

物理层

我们讨论了 6 种规范，如表 6-4 所示。除了红外线外，所有的实现都运行在工业、科学和医学频带，定义了三个范围中的三个许可（原书为无许可怀疑有误。——译者注）频带：902 ~ 928 MHz，2.400 ~ 4.835 GHz 和 5.725 ~ 5.850 GHz。

表 6-4　规范

IEEE	技　术	频　带	调　制	速率（Mbps）
802.11	FHSS	2.400 ~ 4.835 GHz	FSK	1 和 2
	DSSS	2.400 ~ 4.835 GHz	PSK	1 和 2
	无	红外线	PPM	1 和 2
802.11a	OFDM	5.725 ~ 5.850 GHz	PSK 或 QAM	6 到 54
802.11b	DSSS	2.400 ~ 4.835 GHz	PSK	5、5 和 11
802.11g	OFDM	2.400 ~ 4.835 GHz	不同	22 和 54
802.11n	OFDM	5.725 ~ 5.850 GHz	不同	600

IEEE 802.11 FHSS

IEEE 802.11 FHSS 使用跳频扩频（Frequency-Hopping Spread Spectrum，FHSS）方法，如我们在第 7 章所讨论的。FHSS 使用 2.400 ~ 4.835 GHz ISM 频带。该频带分为 79 个 1MHz 的子频带（以及一些防护频带）。一个伪随机数产生器选择跳频序列。该规范中的调制技术可以是二电平 FSK，也可以是四电平 FSK（1 或 2 位/波特），这使得数据速率为 1 或 2Mbps，如图 6-15 所示。

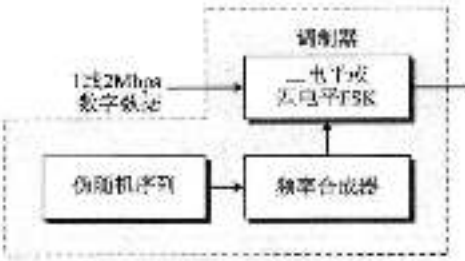


图 6-15　IEEE 802.11 FHSS 的物理层

IEEE 802.11 DSSS

IEEE 802.11 DSSS 使用直接序列扩频（Direct Sequence Spread Spectrum，DSSS）方法，如第 7 章讨论的。DSSS 使用 2.400 ~ 4.835GHz ISM 频带。该规范中的调制技术是 1Mbaud/s 的 PSK。系统允许 1 或 2 位/波特（BPSK 或 QPSK），这使得数据速率为 1 或 2Mbps，如图 6-16 所示。

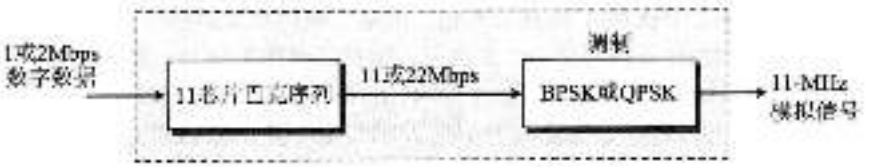


图 6-16　IEEE 802.11 DSSS 的物理层

IEEE 802.11 红外线

IEEE802.11 红外线使用 800nm 到 950nm 范围中的红外线。调制技术称为脉冲相位调制（Pulse Position Modulation，PPM）。为了 1Mbps 的数据速率，4 位序列首先映射成 16 位序列，其中只有 1 位置 1，其余的均置 0。为了 2Mbps 的数据速率，2 位序列首先映射成 4 位序列，其中只有 1 位置 1，其余均置 0。映射序列然后转换为光信号：光存在为 1，光不存在为 0。见图 6-17。

IEEE 802.11a OFDM

IEEE 802.11a OFDM 描述了在 5.725 ~ 5.850 GHz ISM 频带中用于生成信号的正交频分多路复用（orthogonal frequency-division multiplexing，OFDM）方法。OFDM 与第 7 章中讨论的 FDM 类似，主要有一个区别：在给定时间，所有的子波段由一个源使用。各个源在数据链路层访问时互相竞争。频带被分为 52 个子频带，其中 48 个子频带每次发送 48 组位，4 个子频带用于控制信息。将频带分为子频带减少了干扰的影响。如果随机地使用子频带，安全性就可以增加。OFDM 使用 PSK 和 QAM 进行调制。适用的数据速率是 18Mbps（PSK）和 54Mbps（QAM）。

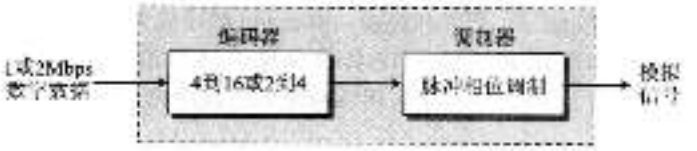


图 6-17　IEEE 802.11 红外线物理层

IEEE 802.11b DSSS

IEEE 802.11b DSSS 描述了在 2.400～4.835 GHz ISM 频带中用于生成信号的高速率直接序列扩频（high-rate direct-sequence spread spectrum，HR-DSSS）方法。HR-DSSS 和 DSSS 类似，区别在于它们的编码方法。HR-DSSS 的编码方法叫做补码键控（complementary code keying，CCK），CCK 编码将 4 位或是 8 位编码成一个 CCK 符号。为了向后兼容 DSSS，HR-DSSS 定义了 4 种数据速率：1、2、5.5 和 11Mbps。前两种使用和 DSSS 相同的调制技术。5.5Mbps 版本使用 BPSK，其传输速率为 1.375Mbps（baud/s），带有 4 位 CCK 编码，而 11Mbps 版本使用 QPSK，其传输速率为 1.375Mbps，带有 8 位 CCK 编码。图 6-18 说明了该标准的调制技术。

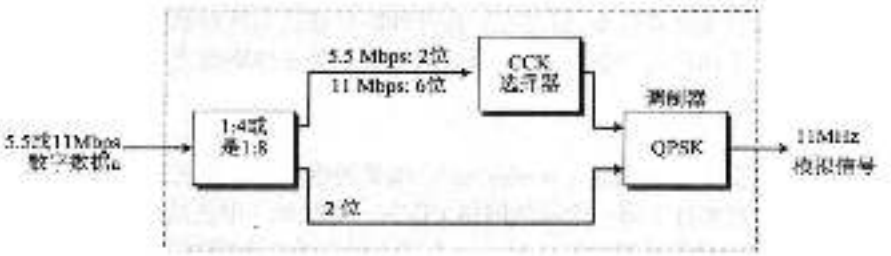


图 6-18　IEEE 802.11b 的物理层

IEEE 802.11g

这个新规范定义了前向纠错和使用 2.400～4.835 GHz ISM 频带的 OFDM。调制技术可以达到 22 或是 54Mbps 的数据速率。它向后兼容 802.11b，但调制技术是 OFDM。

IEEE 802.11n

802.11 项目的升级称为 802.11n（下一代无线局域网）。目标是增加 802.11 无线局域网的吞吐量。新标准不只强调更高的位速率也要消除一些不必要的开销。标准使用称为多输入多输出（multiple-input multiple-output，MIMO）的技术来克服无线局域网中的噪声问题。该想法是如果我们能够发送多路输出信号并且接收多路输入信号，我们能更好地处理噪声。该项目的一些实现已经达到 600Mbps 的数据速率。

6.1.3　蓝牙

蓝牙（bluetooth）是一种无线局域网技术，用来当设备彼此相距较近时，连接不同功能的设备，如电话、笔记本、电脑（台式机和便携式笔记本）、相机、打印机和咖啡壶等。蓝牙局域网是一种自组织网络，即网络是自发形成的；设备有时称为小设备（gadgets），它们之间相互发现并形成称为微微网络的网络。蓝牙局域网甚至能够连接至因特网，此时微微网中的一个必须有这种能力。蓝牙局域网就其本性来说规模不能很大。如果有很多小设备要连接时，就会造成混乱。

蓝牙技术有一些应用。诸如无线鼠标或键盘等外围设备可以通过该技术与计算机通信。在小规模卫生保健中心，监控设备也能够和传感器设备相互通信。家庭安全设备能够使用该技术来将不同传感器连接至主安全控制器。会议出席者可以在会议上同步他们的便携式笔记本。

蓝牙开始于爱立信公司 Ericsson Company 的一个项目，最初称为 Harald Blaatand。Harald Blaatand

是丹麦的国王（940—981），他统一了丹麦和挪威。Blaatand 翻译成为英语中的 bluetooth（蓝牙）。

今天，IEEE 802.15 标准定义了蓝牙技术的实现。该标准定义了一个无线个人域网络（personal-area network，PAN），可以在一个大房间或是大厅范围内运行。

体系结构

蓝牙定义了两种网络：微微网络和散射网络。

微微网络

蓝牙网络称为微微网络（piconet）或小网络。微微网络可以包含多达 8 个站点，其中之一称为主站（primary）；剩下的称为从站（secondary）。所有的从站点的时钟和跳频序列都要和主站点同步。注意一个微微网络只能有一个主站点。在主站点和从站点之间的通信可以是一对一的，也可以是一对多的。图 6-19 显示了一个微微网络。

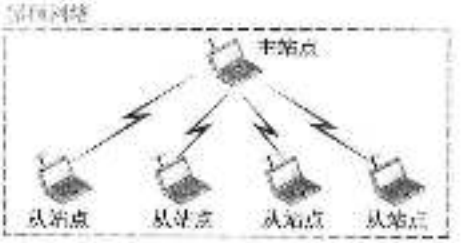


图 6-19　微微网络

尽管微微网络最多可以有 7 个从站点，额外的从站点可以处于休眠状态（parked state）。处于休眠状态的从站点可以和主站点同步，但是在它离开休眠状态进入活动状态之前，它不能参加通信。因为微微网络中只有 8 个站点处于活动状态，所以激活一个处于休眠状态的站点就意味着一个活动站点必须进入休眠状态。

散射网络

微微网络可以组合形成散射网络（scatternet）。微微网络中的一个从站可以是另一个微微网络的主站。该站点可以接收来自于第一个微微网络（作为一个从站）中主站的信息，并且作为第二个微微网络中的主站将信息发送给其中的从站。一个站点可以是两个微微网络中的成员。图 6-20 显示了一个散射网络。

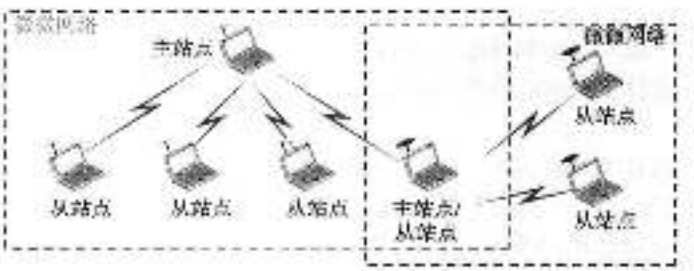


图 6-20　散射网

蓝牙设备

蓝牙设备有一个内置的短程无线电发射机。当前的数据速率是在 2.4GHz 带宽下的 1Mbps。这意味着在 IEEE 802.11b 无线局域网和蓝牙局域网之间有冲突干扰的可能性。

蓝牙层

蓝牙使用与本书所定义的因特网模型不完全匹配的几层。图 6-21 显示了这几层。

L2CAP

逻辑连接控制和自适应协议（Logical Link Control and Adaptation Protocol，L2CAP，这里的 L2 意思是 LL）基本上与局域网中的 LLC 子层类似。它是在 ACL 链路上用于数据交换。SCO 通道不使用 L2CAP。图 6-22 显示了这一级的数据分组的格式。

16 位的长度域以字节为单位定义了来自上层的数据的长度。数据可以多达 65 535 字节。通道 ID（CID）定义了在本级创建的虚拟通道的唯一标识符（后面有述）。



图 6-21　蓝牙层

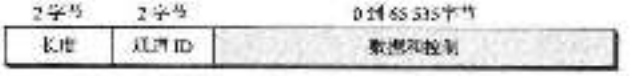


图 6-22　L2CAP 数据分组格式

L2CAP 有几个特殊的任务：多路复用、分段和重装、服务质量（QoS）和组管理。

多路复用　L2CAP 可以实现多路复用。在发送方站点，它接收一个来自于上层协议的数据，组织成帧，将帧传递到基带层以备发送。在接收方站点，它接收来自于基带层的帧，提取数据并将其传递给相应的协议。它创建某种类型的虚拟通道，这将在以后有关高层协议的章节中讨论。

分段和重组　基带层中负载域的最大长度是 2774 位或是 343 字节。其中包含 4 个字节来定义分组和分组长度。因此，能够从上层到达的分组的长度仅为 339 字节。但是，应用层有时需要发送的数据长度可达 65 535 字节（例如一个因特网分组）。L2CAP 将这些大的分组分割成段并附加额外的信息来定义该段在原始分组中的位置。L2CAP 在源站点将分组分成段并在目的站点重组它们。

QoS　蓝牙允许站点来定义服务质量等级。我们在第 8 章中讨论服务质量。暂时可以这样理解，如果不定义服务质量等级，蓝牙就默认尽力服务（best-effort service）；这种情况下，它将尽力提供好的服务质量。

组管理　L2CAP 的另一个功能是允许在设备之间创建一种逻辑寻址类型。这和多播类似。例如，2 个或 3 个从设备可以是多播分组中的部分来从主站接收数据。

基带层

基带层大体上相当于局域网中的 MAC 子层。访问方法是 TDMA（稍后讨论）。主设备和从设备使用时隙彼此通信。时隙的长度恰好与驻留时间相等，为 625μs。这意味着在此期间主设备向从设备发送帧或是从设备向主设备发送帧都使用一个频率。注意通信仅在主设备和从设备之间进行；从设备不能直接相互通信。

TDMA　蓝牙使用的 TDMA 的格式称为时分双工 **TDMA**（Time-Division Duplex TDMA，TDD-TDMA）。TDD-TDMA 是一种半双工通信，在这种类型中，发送方和接收方发送和接收数据，但是不能同时进行（半双工）；每一个方向的通信使用不同的跳频。这与使用不同载波频率的无绳电话类似。

- **单一从设备通信**　如果微微网络仅有一个从设备，TDMA 运行是很简单的。时间被划分为多个 625μs 的时隙。主设备使用偶数时隙（0，2，4，…）；从设备使用奇数时隙（1，3，5，…）。TDD-TDMA 允许主设备和从设备在半双工模式下通信。在时隙 0，主设备发送，从设备接收；在时隙 1，从设备发送，主设备接收。该周期重复。图 6-23 说明了该概念。
- **多个从设备通信**　如果在微微网络中有多个从设备，该过程会有点复杂。和前面一样，主设备使用偶数时隙，但只有在前面时隙的分组寻址到的那个从设备才在下一个奇数时隙内发送。所有的从设备都在偶数时隙期间侦听，但是只有一个从设备在奇数时隙发送。图 6-24 说明了这一情况。

让我们详细说明该图。

1. 在时隙 0，主设备向从设备 1 发送一个帧。

2. 在时隙 1，只有从设备 1 向主设备发送帧，因为前一个帧是向从设备 1 寻址的；其他的从设备处于静止状态。

3. 在时隙 2，主设备向从设备 2 发送帧。

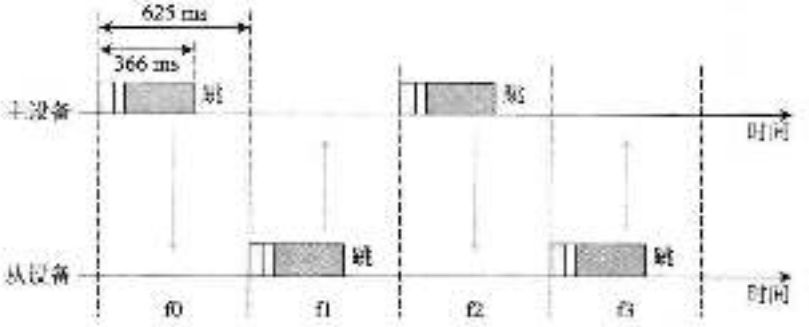


图 6-23　单个从设备通信

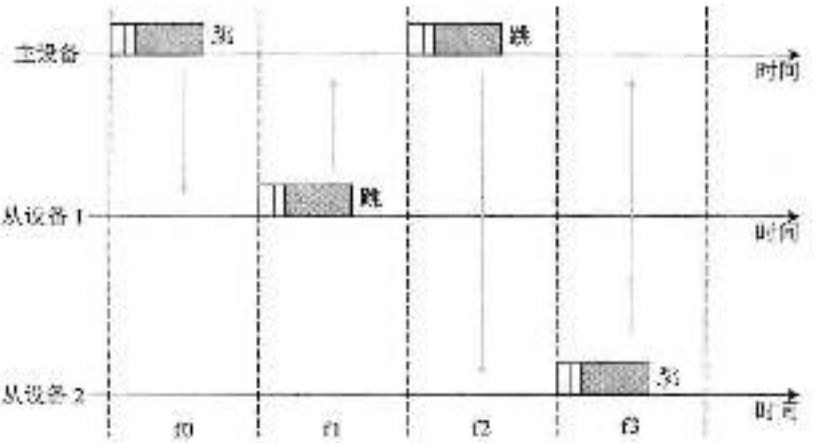


图 6-24　多个从设备通信

4. 在时隙 3，只有从设备 2 向主设备发送一个帧，因为前一个帧是向从设备 2 寻址的；其他从设备处于静止状态。

5. 循环继续。

我们可以说，这一访问方法类似于有保留地轮询/选择操作。当主设备选择一个从设备时，它也轮询该从设备。保留下一个时隙为被轮询到的站点发送它的帧。如果被轮询的从设备没有帧要发送，那么通道就处于静止状态。

链路　在主设备和从设备之间可以创建两种类型的链路：SCO 链路和 ACL 链路。

- **SCO**　当避免延迟（数据传递中的时间延迟）比保证完整性（无错传递）更重要时，使用同步的面向连接（synchronous connection-oriented，SCI）的链路。在 SCO 链路中，通过每隔一段时间保留特定时隙来建立主设备和从设备之间的物理层链路。连接的基本单元是两个时隙，每个方向一个。如果分组被破坏，它从不被重传。SCO 用于实时音频传输，这种场合下避免延迟是最重要的。从设备可以和主设备之间创建多达 3 个 SCO 链路，每条链路发送 64kbps 的数字化的音频信号（PCM）。
- **ACL**　当数据完整性比避免延迟更重要时，使用异步无连接链路（asynchronous connectionless link，ACL）。这种类型的链路中，如果封装在帧中的有效负载被破坏，就要重传。如果前一个时隙已经寻址到一个从设备，该从设备就会在可用奇数时隙内返回一个 ACL 帧。ACL 可以使用一个、三个或更多的时隙并能达到 721kbps 的最大数据速率。

帧格式　基带层中的帧可以是三种类型之一：一个时隙、三个时隙或五个时隙。如我们前面所述，一个时隙是 625 μs。但是，在一个时隙的帧交换中，有 259μs 用于跳频和控制机制。这就意味着一个时隙的帧只能持续（625 − 259）μs，即 366μs。对于 1MHz 的带宽和 1bit/Hz 来说，一个时隙的帧是 366 位。

一个三个时隙的帧占据三个时隙的时间。但是，由于有 259μs 用于跳频，该帧的长度是 $3 \times 625 - 259 = 1616$ μs，或是 1616 位。使用三个时隙帧的设备在同一跳频中（在相同的载波频率下）保持三个时隙。即使只使用一个跳频，也要消耗三个跳频数。这意味着每一帧的跳频数等于帧的第一个时隙。

一个五时隙帧也使用 259 位实现跳频，也就是说该帧的长度是 $5 \times 625 - 259 = 2866$ 位。图 6-25 显示了三种帧类型的格式。

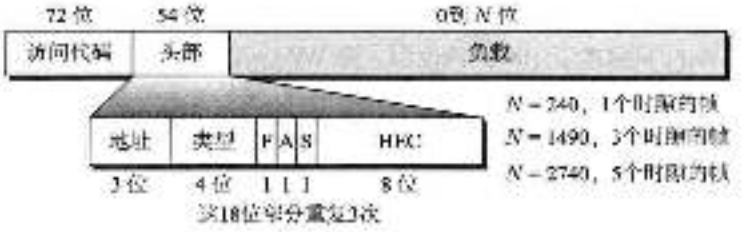


图 6-25　帧格式类型

下面描述各个域：

- 访问代码。这个 72 位的域通常包含同步位和主设备的标识符，用于将一个微微网络的帧与其他微微网络的帧区分开。
- 头部。这个 54 位的域是重复的 18 位模式。每一种模式有下面的子域：

a. 地址。3 位的地址子域能够定义多达 7 个从设备（1 到 7）。如果地址是 0，用于广播主设备到所有从设备的通信。

b. 类型。4 位的类型子域定义了来自于上层的数据类型。我们稍后讨论这些类型。

c. F。1 位的子域用于流量控制。当置为 1 时，表示设备不能接收更多的帧（缓存区已满）。

d. A。1 位的子域用于确认。蓝牙使用停止等待 ARQ；1 位就能满足确认的要求。

e. S。1 位的子域用来保持序列号。蓝牙使用停止-等待 ARQ；1 位就能满足序列编码的要求。

f. **HEC**。8 位的头部差错纠正子域是一个校验和，用于检测每一个 18 位头部分段的差错。头部有三个完全相同的 18 位分段。接收方逐位比较这三个分段。如果每一个对应的位都相同，该位被接收；如果不同，就接收两个相同的位。这是前向差错校正的一种形式（只对于头部）。由于使用空气作为通信的介质，噪音大，所以需要双重的差错控制。注意这一子层中没有重传机制。

- 有效载荷。该子域可以是 0 到 2740 位长。它包含来自上层的数据或是控制信息。

无线电层

无线电层大体上与因特网模型中的物理层相当。蓝牙设备是低功率的，其范围是 10m。

频带　蓝牙使用 2.4GHz ISM 频带，并将其分为 79 个 1MHz 的通道。

FHSS　蓝牙在物理层使用跳频扩频（frequency-hopping spread spectrum，FHSS）方法来避免来自于其他设备或网络的干扰。蓝牙每秒跳频 1600 次，即每个设备每秒改变它的调制频率达 1600 次。一个设备在它跳转到另一个频率之前，使用一个频率的时间为 625μs（1/1600s）；驻留时间是 625μs。

调制　为了将位转变为信号，蓝牙使用 FSK 的一个复杂版本，叫做 GFSK（带有高斯带宽过滤的 FSK，对该问题的讨论超出了本书的范围）。GFSK 有一个载波频率。位 1 由一个在载波频率之上的频率偏移来表示；位 0 由一个在载波频率之下的频率偏移来表示。以兆赫兹为单位的频率，

对每个通道而言根据以下公式定义：

$$f_c = 2402 + n \text{ MHz} \qquad n = 0, 1, 2, 3, \cdots, 78$$

例如，第一个通道使用载波频率 2402MHz（2.402GHz），第二个通道使用载波频率 2403MHz（2.403 GHz）。

6.1.4　WiMAX

全球互通微波存取（Worldwide Interoperability for Microwave Access，WiMAX）是 IEEE 标准 802.16（为固定的无线）和 802.16e（为移动无线）。其目标是提供“最后一公里”宽带无线接入，可以替代有线调制解调器和电话 DSL 的服务。WiMAX 为视线（line-of-sight，LOS）用户提供了到基站的最佳的范围和吞吐量（通过非屏蔽的路径），并为接近或是非视线（non-line-of-sight，NLOS）用户提供了到基站的可接受的范围和吞吐量。

很多用户将 WiMAX 比作 WiFi。像 WiFi 一样，WiMAX 有一个基站基础设施，但是它比 WiFi 提供的范围更多。WiFi 只覆盖了 100 码的范围，而 WiMAX 覆盖了 6 英里的范围。与 WiFi 相比，WiMAX 提供了更好的安全性、可靠性、服务质量和吞吐量。

体系结构

本节中我们简单地讨论 WiMAX 体系结构。

基站

一个 WiMAX 基站的基本单元是无线电广播设备和天线。每一个 WiMAX 无线电广播设备都有一个发射机和一个接收器，传输频率为 2 – 11GHz 的信号。WiMAX 使用软件定义无线电（Software Defined Radio，SDR）系统。

在 WiMAX 中使用三种不同类型的天线（全方位、扇形和平板型）来为给定的应用优化性能。WiMAX 使用光束偏转自适应天线系统（adaptive antenna system，AAS）。当传输时，AAS 天线可以将其传输能量集中于接收方的方向；当接收时，它可以集中于传输设备的方向。

WiMAX 中使用的其他的干扰避免措施是使用 OFDMA 和 MIMO 天线系统。OFDMA 是一种多路访问方法，它允许同时向用户传输和从用户传输，并且有效地与 AAS 和 MIMO 一起工作，能显著地增加吞吐量、增加链路范围以及减少干扰。

用户站点

客户端设备（customer premises equipment，CPE）也称作用户单元（subscriber unit）。它在室内版本或室外版本都是有效的、可用的。室内单元是电缆或是 DSL 调制解调器的规格，并且是自安装的，但是由于无线电损耗，要求用户离基站更近一些。室外版本是卫星电视的规格，必须经专业安装。

便携单元

随着移动 WiMAX 的潜力挖掘出来，大家越来越关注便携式单元，包括手机、PC 外围设备、笔记本电脑中的嵌入式设备以及用户电子设备（比如游戏终端、MP3 播放器等）。

数据链路层

WiFi 的 MAC 使用竞争访问。这可能导致远离 AP 的用户站点时常被更近的站点中断。WiMAX 中的 MAC 使用调度算法。用户站点只需要为初始进入网络竞争一次。访问时隙然后分派给该用户。

物理层

802.16e-2005 规定了 2GHz 到 11GHz 的范围，可扩展 **OFDMA**（scalable OFDMA，SOFDMA）、MIMO 天线和完全移动性支持的能力。

应用

WiMAX 的目标是对一些现存的通信基础设施提供成本效益好的可替代方案，现存的基础设施

包括电话公司的铜线、蜂窝网络以及有线电视的同轴电缆基础设施。

6.2　其他无线网络

本节中，我们专注于其他无线网络。我们首先讨论普遍存在的蜂窝电话。然后讨论卫星网络。在讨论上述无线网络之前，让我们讨论从第 5 章中推迟到此的一种访问方式：通道化，它在蜂窝网络和其他无线网络中使用。

6.2.1　通道化

通道化（channelization，有时称为通道划分）是一种多路访问方法，其中链路的可用带宽在不同的站点之间通过时间、频率或编码来共享。本节中，我们讨论三种通道化协议：FDMA、TDMA 和 CDMA。

频分多址（FDMA）

在**频分多址**（Frequency-Division Multiple Access，FDMA）中，可用带宽被分为频带。每一个站点分配一个频带来发送数据。换言之，每个频带为一个指定站点保留，它一直属于该站点。每个站点也使用一个带通滤波器来限制发射机效率。为了预防站点干扰，分配的频带通过小的防护频带彼此分离。图 6-26 说明了 FDMA 的概念。

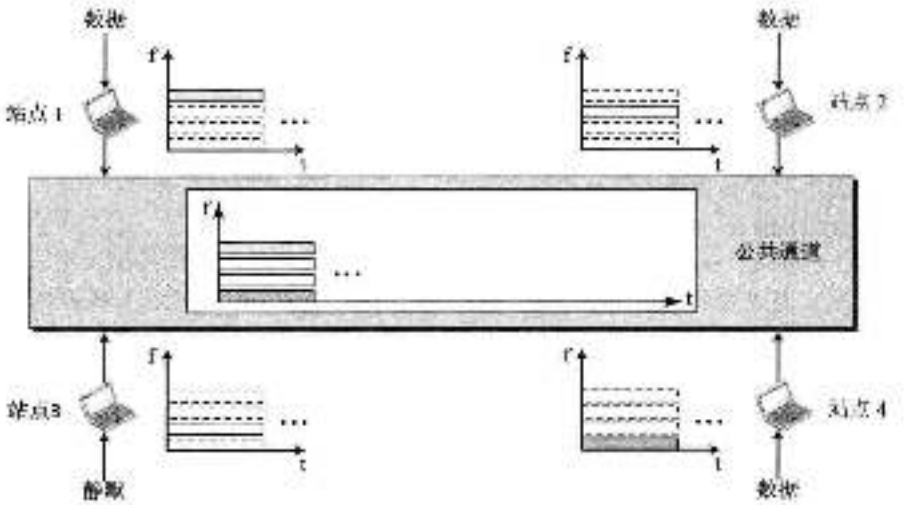


图 6-26　频分多址（FDMA）

FDMA 为整个通信周期指定一个预定频率的频带。这意味着 FDMA 可以方便地使用流数据（不能分组的连续的数据流）。我们稍后将会看到蜂窝电话系统如何使用这一特性。

我们需要强调 FDMA 是链路层协议；它不应该和我们第 7 章中讨论的复用过程频分复用（FDM）相混淆。FDM 实际上是 FDMA 在物理层的实现。

时分多址（TDMA）

在**时分多址**（Time-Division Multiple Access，TDMA）中，站点在时间上共享通道的带宽。每个站点分配一个时隙，在这一时隙内它能够发送数据。每个站点在它指定的时隙内传输数据。图 6-27 说明了 TDMA 的概念。

TDMA 的主要问题在于在不同站点之间达到同步。每一个站点需要知道自身时隙的开始和所处的位置。如果站点分布在一个广泛的区域内，那么系统产生的广播延迟会给前述问题带来困难。为了使延迟得到补偿，我们插入了保护时间（guard time）。通过每一个时隙开始处的一些同步位（通

常称为前导位）来完成同步化。

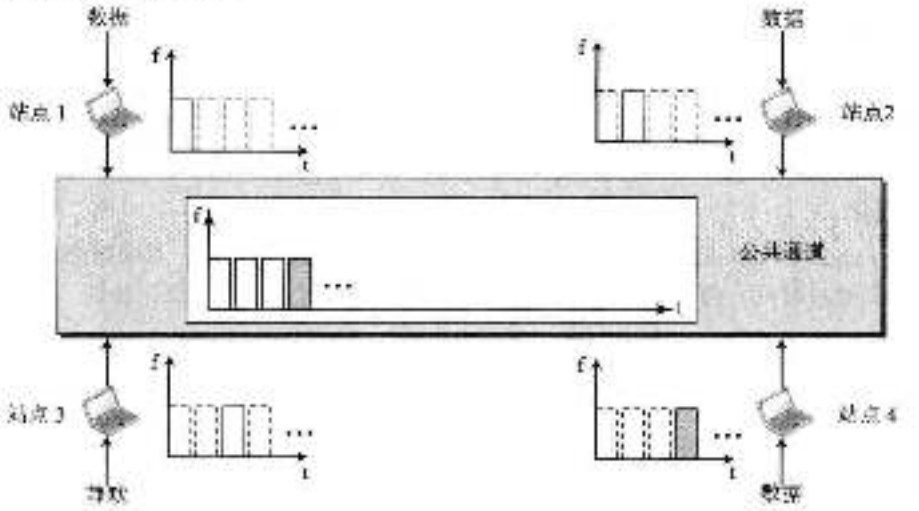


图 6-27　时分多址（TDMA）

我们也需要强调尽管 TDMA 和 TDM（时分复用）在概念上相似，但是它们之间存在不同。我们第 7 章中介绍的 TDM 是物理层技术，它将来自于低速通道的数据组合起来，通过使用高速通道传输它们。该过程使用物理多路复用器从每个通道交叉存取数据单元。

另一方面，TDMA 是链路层的访问方法。每个站点的数据链路层告诉其物理层来使用指定的时隙。在数据链路层没有物理多路复用器。

码分多址（CDMA）

码分多址（Code-Division Muluple Access，CDMA）在几十年前就已经提出来了。电子技术的新发展使得它的实现成为可能。与 FDMA 不同，CDMA 仅有一个通道占据链路的整个带宽。与 TDMA 不同，所有的站点都可以同时发送数据，没有时间的共享问题。

类比

我们首先给出一个类比。CDMA 是指使用不同的编码来通信。例如，在一个很大的房间里有很多人，如果其他人都不懂英语，两人就可以用英语交流。另两个人可以用汉语交流，如果只有他们懂汉语，依此类推。换言之，公共通道可以允许不同对之间的通信，但是使用不同语言（编码）。

思想

我们假设有四个站点，1、2、3 和 4，它们连接至相同的通道。来自于站点 1 的通道是 d_1，来自于站点 2 的是 d_2，依此类推。指定给第一个站点的编码是 c_1，第二个站点是 c_2，依此类推。我们假设被分配的编码有两个特性。

1. 如果我们将两个编码相乘，结果为 0。
2. 如果我们将编码自身相乘，结果为 4（站点的数目）。

通过这两个假想的特性，我们查看上述 4 个站点如何使用公共通道发送数据，如图 6-28 所示。

站点 1 将其数据和它的编码相乘（如我们所看到的，这是一种特殊的乘法）得到 $d_1 \cdot c_1$。站点 2 将其数据和它的编码相乘得到 $d_2 \cdot c_2$，依此类推。通道上的数据是所有这些结果的和，如图中方框部分所示。任何想从其他三个站点之一接收数据的站点将通道上的数据和发送方的编码相乘。例如，假设站点 1 和 2 在彼此通话。站点 2 想要听站点 1 在说什么。它将通道上的数据与站点 1 的编码 c_1 相乘。

因为 $(c_1 \cdot c_1)$ 是 4，但是 $(c_2 \cdot c_1)$、$(c_3 \cdot c_1)$ 和 $(c_4 \cdot c_1)$ 全是 0，站点 2 将结果除以 4 以获取来自

于站点 1 的数据。

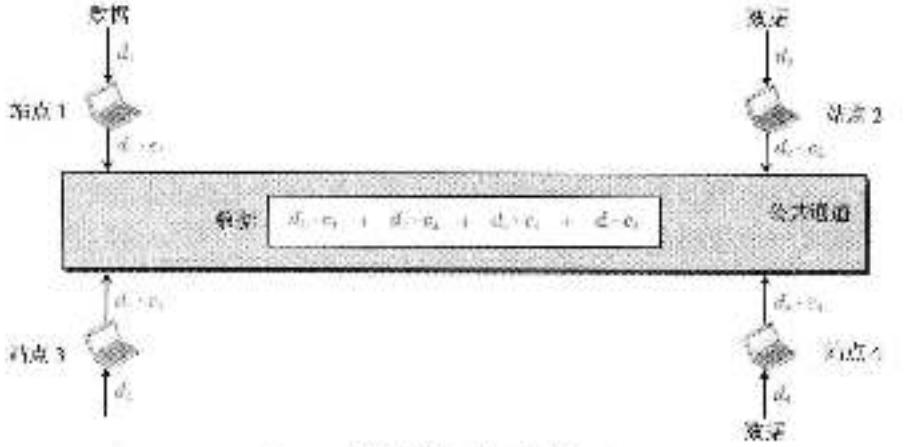


图 6-28　使用编码通信的简单思想

数据 = $[(d_1 \cdot c_1 + d_2 \cdot c_2 + d_3 \cdot c_3 + d_4 \cdot c_4) \cdot c_1] / 4 = [d_1 \cdot c_1 \cdot c_1 - d_2 \cdot c_2 \cdot c_1 + d_3 \cdot c_3 \cdot c_1 - d_4 \cdot c_4 \cdot c_1] / 4$
$= (4 \times d_1) / 4 = d_1$

码片

CDMA 基于编码理论。每一个站点指定一种编码，编码是称为码片（chip）的数字序列。图 6-29 显示了前一个例子的码片。

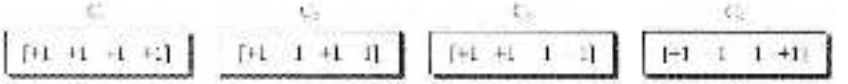


图 6-29　码片序列

不幸的是我们没有编码如何选择。现在需要知道我们不是随机选择序列而是谨慎地选择的。它们称为正交序列（orthogonal sequences），有以下特性：

1. 每一个序列由 N 个元素组成，N 是站点的数目，需要是 2 的幂。

2. 如果我们将序列乘以一个数，序列中的每一个元素都乘以该数。这称为序列与标量的乘积。例如，

$$2 \cdot [+1 +1 -1 -1] = [+2 +2 -2 -2]$$

3. 如果我们将两个相同的序列的元素逐个相乘，然后将结果相加，我们得到 N，N 是每个序列中元素的数目。这称为两个相同序列的内积。例如：

$$[-1 +1 -1 -1] \cdot [-1 -1 -1 -1] = 1 - 1 - 1 + 1 = 4$$

4. 如果我们将两个不同的序列的元素逐个相乘，并将结果相加，结果为 0。这称为两个不同序列的内积。例如，

$$[+1 -1 -1 -1] \cdot [+1 -1 -1 +1] = 1 - 1 - 1 - 1 = 0$$

5. 将两个序列相加意味着将其对应元素相加。结果是另一个序列。例如：

$$[+1 -1 -1 -1] + [+1 -1 +1 +1] = [+2 -2 0 0]$$

数据表示法

我们遵循编码规则：如果站点需要发送位 0，它将其编码为−1；如果要发送位 1，将其编码为+1。当站点空闲时，它不发送信号，可以看做是 0。这些规则如图 6-30 所示。

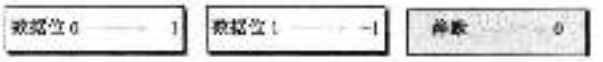


图 6-30　CDMA 中的数据表示

编码和解码

作为一个例子，我们说明 4 个站点如何在 1 位间隔中共享链路。过程很容易为附加的间隔重复。我们假设站点 1 和 2 正在发送 0 位，站点 4 正在发送 1 位。站点 3 是静默的。发送站点的数据依次编码为−1、−1、0 和+1。每一个站点将相应的数和它的码片序列（它的正交序列）相乘，该码片对每个站点来说是唯一的。结果是一个新的序列，被发送至通道。为了简化，我们假设所有的站点同时发送结果序列。如前面定义的，通道上的序列是所有四个序列之和。图 6-31 显示了这种情况。

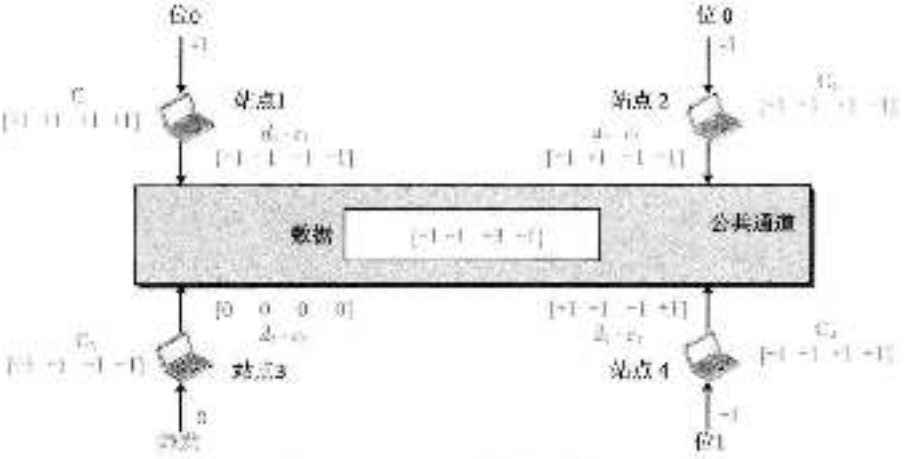


图 6-31　CDMA 中的共享通道

现在假设静默的站点 3 正在侦听站点 2。站点 3 将通道上的总数据和站点 2 的编码（[+1 −1 +1 −1]）相乘得到：

$$[-1\ -1\ -3\ -1] \cdot [+1\ -1\ +1\ -1] = 4$$

$$-4/4 = -1 \rightarrow \text{bit } 0.$$

信号水平

如果我们说明了每个站点产生的数字信号和在目的站点恢复的数据，该过程会更好理解（如图 6-32 所示）。该图说明了每个站点相应的信号（NRZ-L 信号，见第 7 章）和公共通道上的信号。

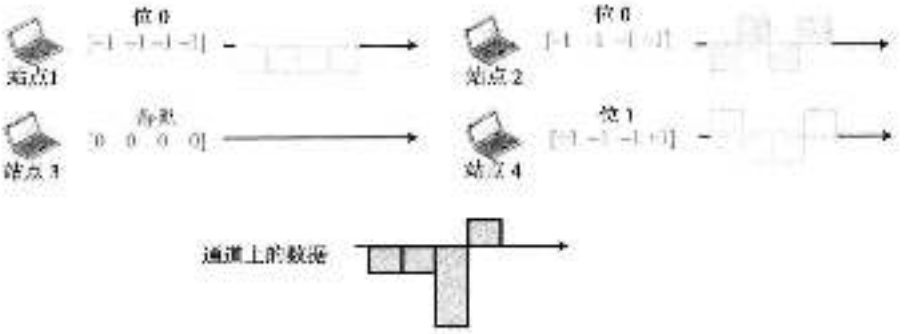


图 6-32　CDMA 中由四个站点产生的数字信号

图 6-33 说明了通过使用站点 2 的编码，站点 3 如何检测由站点 2 发送的数据。通道上的总数

据和代表站点 2 的码片编码的信号相乘（内积运算）产生一个新的信号。然后站点将信号下的领域整合相加，得到值-4，被 4 整除结果为-1，解释为位 0。

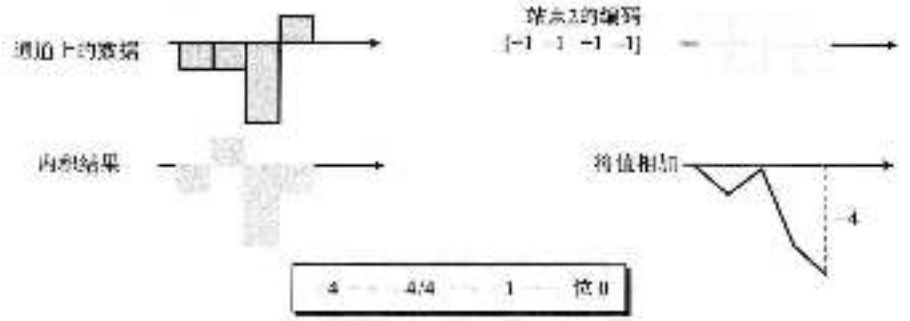


图 6-33　CDMA 中的合成信号的解码

序列产生

为了产生码片序列，我们使用 **Walsh 表**，它是一张二维表，而且行数和列数相等，如图 6-34 所示。

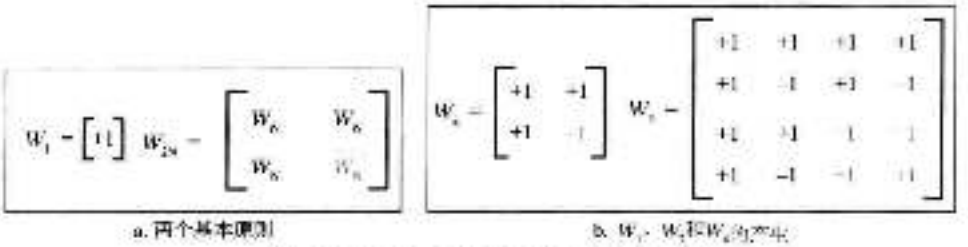


图 6-34　创建 Walsh 表的基本原则和示例

在 Walsh 表中，每一行是一个码片序列。一个码片序列 W_1 有一行和一列。我们可以为这个表的码片选择-1 或是+1（我们选择+1）。根据 Walsh，如果知道 N 序列的表 W_N，我们可以为 $2N$ 序列创建一个表 W_{2N}，如图 6-34 所示。骨上标线的 W_N 代表 W_N 的反码，其中+1 变为-1，反之亦然。图 6-34 也说明了如何由 W_1 创建 W_2 和 W_4。我们选择 W_1 之后，W_2 可以由 4 个 W_1 组成，其中最后一个是 W_1 的反码。在 W_2 产生后，W_4 可以由 4 个 W_2 组成，其中最后一个是 W_2 的反码。当然，W_8 由 4 个 W_4 组成，依此类推。注意，在产生 W_N 后，会给每一个站点分配一行相应的码片。

我们需要强调的是序列 N 的个数需要是 2 的幂。换言之，我们需要 $N = 2^m$。

例 6.1　计算以下网络的码片

a. 两个站点

b. 四个站点

解答

我们可以使用图 6-34 中的 W_2 和 W_4 的行：

a. 对于两个站点网络来说，我们有[+1 +1]和 [+1 -1]。

b. 对于四个站点网络来说，我们有[+1 +1 +1 +1]、[+1 -1 +1 -1]、[+1 +1 -1 -1]和[+1 -1 -1 +1]。

例 6.2　如果我们的网络中有 90 个站点，序列个数是多少？

解答

序列的个数需要是 2^m。我们需要选择 $m = 7$，$N = 2^7$，即 128。然后使用 90 个序列作为码片。

例 6.3　证明，如果用通道上的整个数据和指定发送方的芯片编码相乘，然后将结果除以站点

的个数，接收站点就可以获得指定发送方发送的数据。

解答

我们使用前面 4 个站点的例子来为第一个站点证明这一点。我们可以说通过上的数据 $D = (d_1 \cdot c_1 + d_2 \cdot c_2 - d_3 \cdot c_3 + d_4 \cdot c_4)$。想要获取站点 1 发送数据的接收方将这些数据乘以 c_1：

$$\begin{aligned}[D \cdot c_1] / 4 &= [(d_1 \cdot c_1 + d_2 \cdot c_2 + d_3 \cdot c_3 + d_4 \cdot c_4) \cdot c_1] / 4 \\ &\quad - [d_1 \cdot c_1 \cdot c_1 + d_2 \cdot c_2 \cdot c_1 + d_3 \cdot c_3 \cdot c_1 + d_4 \cdot c_4 \cdot c_1] / 4 \\ &\quad - [d_1 \times 4 + d_2 \times 0 + d_3 \times 0 + d_4 \times 0] / 4 = [d_1 \times 4] / 4 = d_1\end{aligned}$$

6.2.2　蜂窝电话

蜂窝电话（cellular telephony）用来为两个称为移动站点（Mobile Station，MS）的移动单元之间提供通信，或是在一个移动单元和一个静止单元（通常称为地面单元，land unit）之间提供通信。服务提供商必须能够定位并追踪主叫方，为呼叫分配一个通道，并且当主叫方移动到范围之外时，将通道从一个基站切换到另一个基站。

为了使这种追踪成为可能，每一个蜂窝服务区域分成称为信元（cell）的小区域。每个信元包含一个天线并且被一个小的称为基站（Base Station，BS）的由太阳能或是 AC 供能的网络站点控制。每一个基站控制由一个称为移动交换中心（Mobile Switching Center，MSC）的交换局控制。MSC 协调所有基站和电话中心局之间的通信。它是一个计算中心，负责呼叫连接，记录呼叫信息和计费（见图 6-35）。

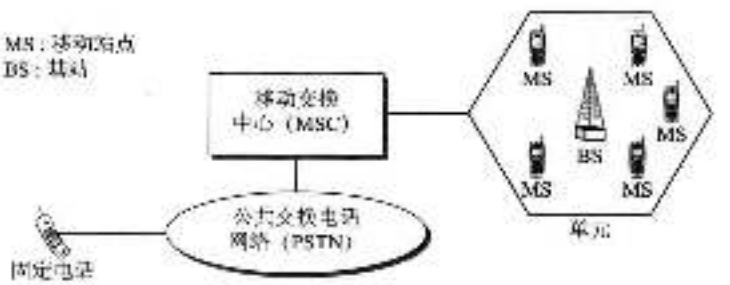


图 6-35　蜂窝系统

信元大小不是固定的，并能依据该地区人口数增大或减少。一个信元的典型范围是 1 到 12 英里。与低密度地区相比，高密度地区要求更多的地理范围更小的信元来满足流量需求。信元大小一旦确定，应该是阻止邻近信元信号相互干扰的最佳大小。每个信元的传输能量都保持在低水平来阻止信号被其他信元干扰。

频率复用原理

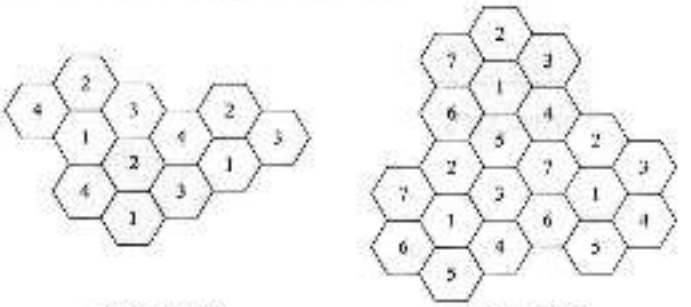
通常，相邻的信元不能使用同一套频率来通信，因为可能给位于信元边界的用户产生干扰。然而，可用的频率集是有限的，所以频率需要复用。频率复用的模式是 N 个信元的配置，N 是复用因子（reuse factor），其中每一个信元使用唯一的频率集。当模式重复时，频率就可以复用。有几种不同的模式。图 6-36 是其中的两种。

信元中的数字定义了模式。模式中相同数字的信元可以使用相同的频率集。我们称这些信元为复用信元（reusing cell）。如图 6-36 所示，复用因子为 4 的模式中，仅用一个信元分开使用相同频率集的信元。在复用因子为 7 的模式中，用两个信元分开复用信元。

传输

为了放置来自移动站点的呼叫，主叫方输入 7 位或 10 位数字的编码（电话号码），然后按发送按钮。移动站点然后扫描频带，寻找一个信号强的设置通道，然后使用该通道向最近的站点发送数

据（电话号码）。基站向 MSC 转发数据。MSC 接着向电话中心局发送数据。如果被叫方空闲，连接建立并且将结果返回给 MSC。这时候，MSC 就为此呼叫指定一个未使用的声音通道，连接建立。移动站点根据通道自动调整频率，通信就可以开始了。



a. 使用 7 子区　　b. 使用 3 子区

图 6.36　频率复用模式

接收

当移动手机被呼叫时，电话中心局便将此号码发送到 MSC。MSC 通过向所有信元发送查询信号来搜索移动站点的位置，此过程称为寻呼（paging）。一旦找到移动站点，MSC 就发送一个铃声信号。当移动站点应答时，就为此呼叫指定一个声音通道，允许语音通信开始。

切换

在会话中，移动站点可能从一个信元移动到另一个信元。当这种情况发生时，信号可能变弱。为了解决该问题，MSC 每隔几秒就监测一次信号强度。如果信号的强度减小，MSC 就寻找一个新的信元，以便更好地进行通信。然后 MSC 改变承载该呼叫的通道（将信号从旧的通道切换到新的通道）。

硬切换　早期的系统使用硬切换（hard handoff）。在硬切换中，移动站点只和一个基站通信。当 MS 从一个信元切换到另一个时，和前一个基站的通信必须首先切断，然后再和新的基站建立通信。这可能创建一个粗转换。

软切换　新的系统使用软切换（soft handoff）。这种情况下，移动站点能够和两个基站同时通信。这意味着，在切换过程中，移动站点可能在切断和旧基站的联系之前，就可以使用新的站点继续通话。

漫游

蜂窝网络的一个特性称为漫游（roaming）。漫游意味着原则上一个用户只要在覆盖范围内，用户都能发出通信或是被呼叫。服务商通常提供有限的范围。邻近的服务提供商能够通过漫游合约提供扩展的覆盖范围。该情况类似于国家之间邮寄信件。两个国家之间信件的交付费用可以由这两个国家通过协商来分担。

第一代（1G）

蜂窝电话网络现在处于第四代。第一代设计使用模拟信号来进行语音通信。我们讨论在北美使用的一种第一代移动系统，即 AMPS。

AMPS

高级移动电话系统（Advanced Mobile Phone System，AMPS）是北美先进的模拟移动电话系统之一。它使用 FDMA（见第 5 章）来分隔链路中的通道。

AMPS 是一种使用 FDMA 的模拟移动电话系统。

频段 AMPS 运行在 ISM 800MHz 的频带。系统使用两个独立的模拟通道，一个用于前向通信

（基站到移动站点），一个用于反向通信（移动站点到基站）。在 824MHz 和 849MHz 之间的频带用于反向通信；在 869MHz 到 894MHz 之间的频带用于前向通信（见图 6-37）。

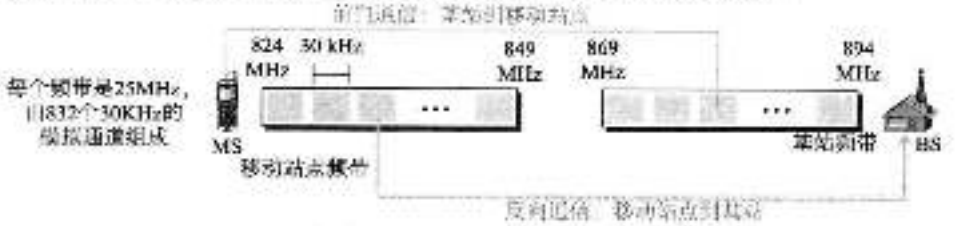


图 6-37　AMPS 的蜂窝频带

每一个频带分为 832 个通道。但是，两个服务提供商能共享一个地区，这意味着每个提供商的信元只有 416 个通道。其中 21 个通道用于控制，剩余 395 个通道。AMPS 的频率复用因子为 7；这意味着一个信元中仅有 395 个流量通道的 1/7 可以使用。

传输　AMPS 使用 FM 和 FSK 调制信号。图 6-38 显示了反向的传输。语音通道使用 FM 调制，控制通道使用 FSK 来创建 30KHz 的模拟信号。AMPS 使用 FDMA 将 25MHz 的频带分为 30KHz 的通道。

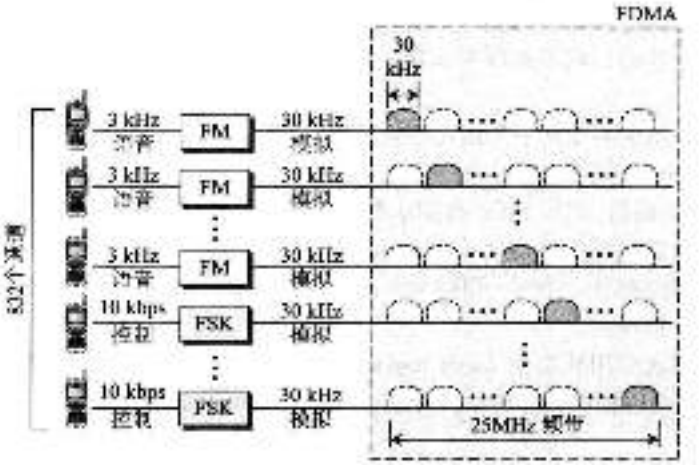


图 6-38　AMPS 反向通信频带

第二代（2G）

为了提供高质量的（更少的噪声）移动语音通信，蜂窝电话网络的下一代发展起来。第一代用于模拟语音通信，而第二代主要用于数字语音通信。第二代中有三个主要系统：D-AMPS、GSM 和 CDMA。

D-AMPS

模拟 AMPS 演化成数字系统的产品是数字 **AMPS**（Digital AMPS，D-AMPS）。D-AMPS 设计为向后兼容 AMPS。这意味着，在一个信元里，一个电话可以使用 AMPS，另一个可以使用 D-AMPS。D-AMPS 最初由 IS-54（过渡标准 54）定义，后来在 IS-136 中修订。

频带　D-AMPS 使用和 AMPS 一样的频带和通道。

传输　每一个语音通道使用非常复杂的 PCM 和压缩技术来进行数字化。语音通道数字化为 7.95kbps。三个 7.95kbps 数字语音通道使用 TDMA 组合在一起。结果是 48.6kbps 的数字数据；这里有很多的额外开销。如图 6-39 所示，系统每秒发送 25 帧，每帧 1944 位。每帧持续 40ms（1/25），并且分为六个时隙，由三个数字通道共享；每个通道分配两个时隙。

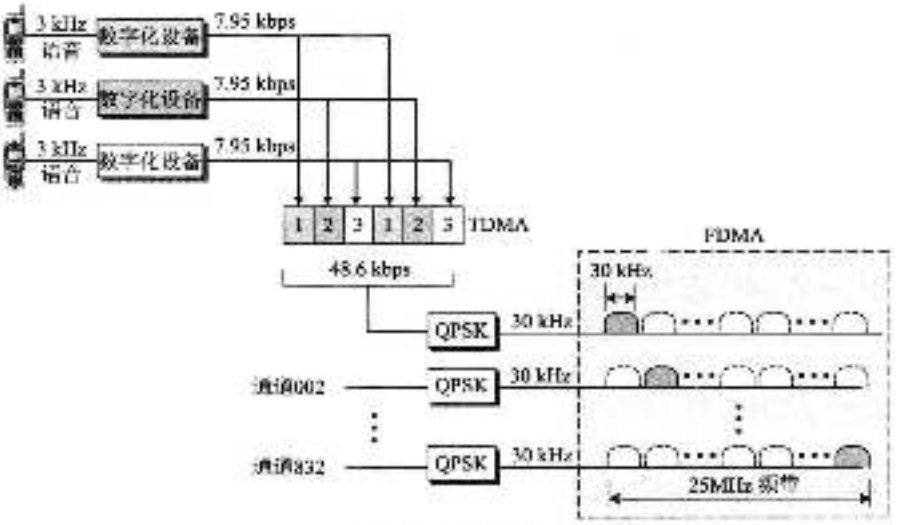


图 6-39　D-AMPS

每个时隙容纳 324 位。但是，只有 159 位来自数字语音；64 位用于控制，101 位用于差错纠正。换言之，在分配给它的两个通道中，每个通道都携带 159 位的数据。系统增加 64 位控制位和 101 位纠错位。

使用 QPSK 将 48.6kpbs 的数字信号调制成载波信号；结果是 30kHz 的模拟信号。最后，30kHz 模拟信号共享 25MHz 的频带（FDMA）。D-AMPS 的频率复用因子是 7。

D-AMPS 或 IS-136 是使用 TDMA 和 FDMA 的数字蜂窝电话系统。

GSM

全球移动通信系统（Global System for Mobile communication，GSM）是为所有欧洲国家提供通用的第二代移动通信技术而研发的欧洲标准。目标是取代很多不兼容的第一代移动通信技术。

频带　GSM 使用两个频段以实现双工通信。每一个频带宽度为 25MHz，向 900MHz 方向推移，如图 6-40 所示。每一个频带分为 200kHz 的 124 个通道，这些通道被防护频带隔开。



图 6-40　GSM 频带

传输　图 6-41 显示了一个 GSM 系统。每一个语音通道被数字化，并压缩为 13kbps 的数字信号。每一个时隙携带 156.25 位。8 个时隙共享一个帧（TDMA）。26 个帧共享一个复帧（TDMA）。我们可以按以下公式计算每个通道的位数率。

$$\text{通道数据速率} = (1/120\ \text{ms}) \times 26 \times 8 \times 156.25 = 270.8\ \text{kbps}$$

每一个 270.8kbps 的数字通道使用 GMSK（主要在欧洲系统中使用的一种 FSK 形式）调制成载波；结果是一个 200kHz 的模拟信号。最后 124 个 200kHz 的模拟通道使用 FDMA 组合在一起。结果是 25MHz 的频带。图 6-42 显示了复帧中的用户数据和额外开销。

读者可能已经注意到 TDMA 中有大量的开销。每个时隙用户数据只有 65 位。系统添加用于纠错的额外的位使得每个时隙 114 位。就这样，加上控制位使得每个时隙达到 156.25 位。8 个时隙封装成一帧。24 个流量帧和 2 个附加的控制帧组成一个复帧。一个复帧的持续时间为 120ms。但是，这个结构定义了不附加任何开销的超帧和超大帧；这里我们不讨论它们。

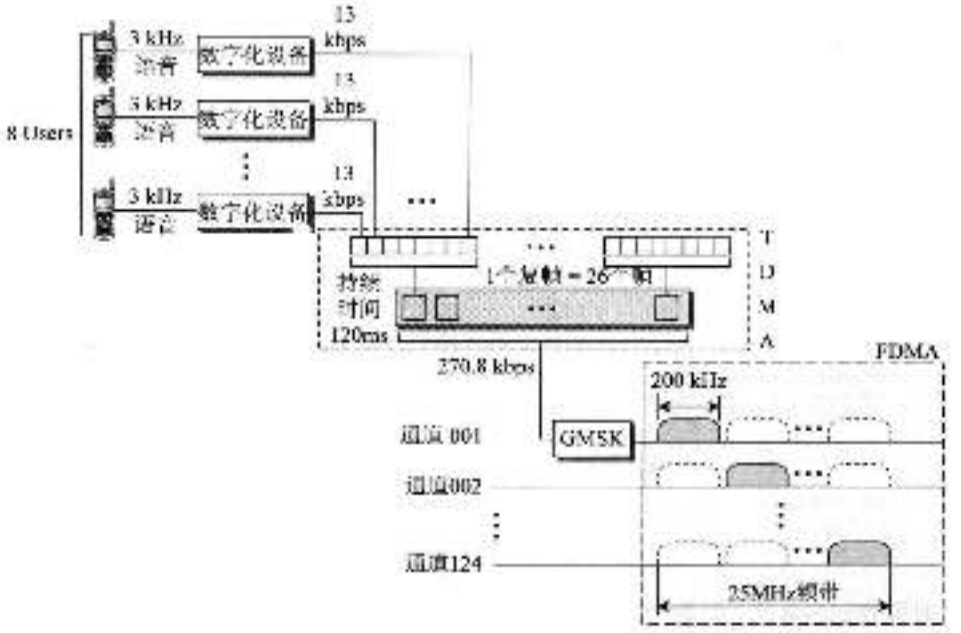


图 6-41　GSM

复用因子　由于复杂的差错控制机制，GSM 允许复用因子最小为 3。

GSM 是使用 TDMA 和 FDMA 的数字蜂窝电话系统。

IS-95

在北美占优势的第二代标准之一是过渡标准 95（Interim Standard 95，IS - 95）。它基于 CDMA 和 DSSS。

频带和通道　IS-95 使用两个频带用于全双工通信。频带可以是传统的 ISM 800MHz 的频带，或是 ISM 1900MHz 的频带。每一个频带分为 1.228MHz 的 20 个通道，并且被保护频带分隔开。每一个服务提供商分配 10 个通道。IS-95 可以用来和 AMPS 并行工作。每一个 IS-95 通道相当于 41 个 AMPS 通道（41 × 30 kHz = 1.23 MHz）。

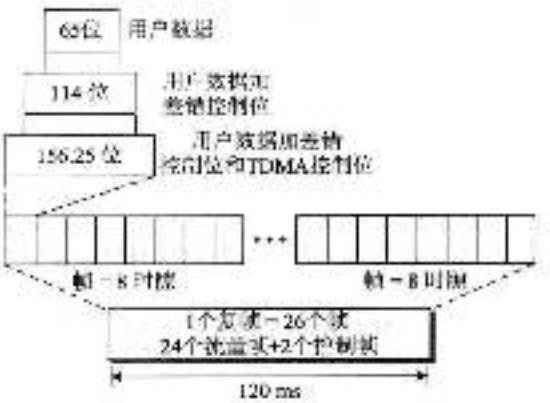


图 6-42　反向组成

同步　所有使用 CDMA 的基本通道都需要同步。为了提供同步，基站使用 GPS（全球定位系统）服务。GPS 是一个卫星系统，我们在下一节讨论。

前向传输　IS-95 有两种不同的传输技术：一个用于前向（基站向移动站点）方向，另一个用于反向（移动站点向基站）方向。在前向方向，基站和所有的移动站点之间的通信都是同步的；基站向所有的移动站点发送同步数据。图 6-43 显示了前向方向的一个简化图示。

每一个语音通道都是数字化的，以 9.6kbps 的基本速率产生数据。在添加纠错、重复位和交织位后，结果是 19.2ksps（千信号每秒）的信号。现在输出结果就是用 19.2ksps 的信号进行扰频处理的。该扰频信号由一个长代码生成器来生成，该长代码生成器使用移动站点的电子序列号（ESN），并生成 2^{42} 个伪随机码片，每个码片 42 位。注意这些码片是伪随机生成的，不是随机的，因为样

本重复使用。长代码生成器的输出供应给抽取器，抽取器选择 64 位中的 1 位。抽取器的输出用于扰频。扰频用于保密；每个站点的 ESN 是唯一的。

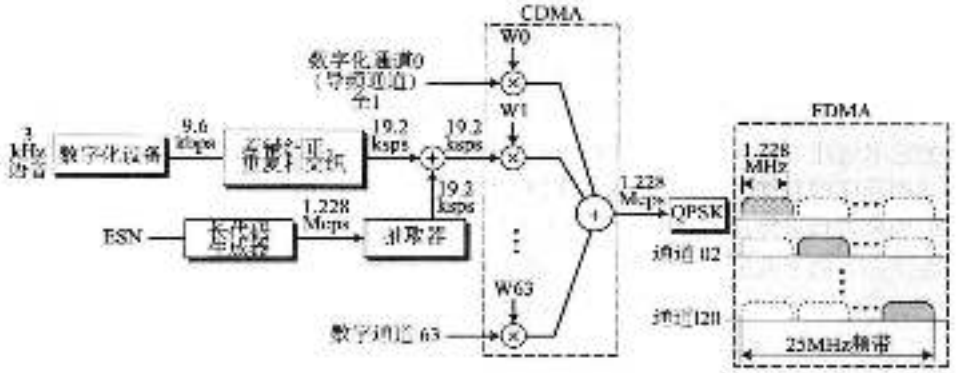


图 6-43　IS-95 前向传输

扰频的结果使用 CDMA 合成。对于每个流量通道，选择一个 Walsh 64 × 64 行码片，结果是 1.228Mcps 的信号（兆码片每秒）。

$$19.2\ \text{ksps} \times 64\ \text{cps} = 1.228\ \text{Mcps}$$

信号提供给 QPSK 调制器生成 1.228MHz 的信号。使用 FDMA，产生的带宽也相应移动。一个模拟通道创建 64 个数字通道，其中 55 个是流量通道（携带数字语音）。9 个通道用于控制和同步。

a. 通道 0 是一个导频通道。该通道向所有的移动站点发送 1 的数据流。该数据流提供了位同步，也可以作为解调的相位参考，并允许移动站点来与临近基站的信号强度相比较以做出切换决定。

b. 通道 32 向移动站点给出关于系统的信息。

c. 通道 1 到 7 用于调度，向一个或更多的移动站点发送信息。

d. 通道 8 到 31 和 33 到 63 是携带从基站到相应移动站点的数字语音的流量通道。

反向传输　前向方向中 CDMA 的使用是可行的，因为导频通道不间断地发送 1 的序列来同步传输。在反向传输中不能使用同步，因为我们需要一个实体来做这些，但这是不可行的。反向通道不使用 CDMA 而使用 DSSS（直接序列扩频），这在第 7 章中讨论。图 6-44 显示了反向传输的简化图。

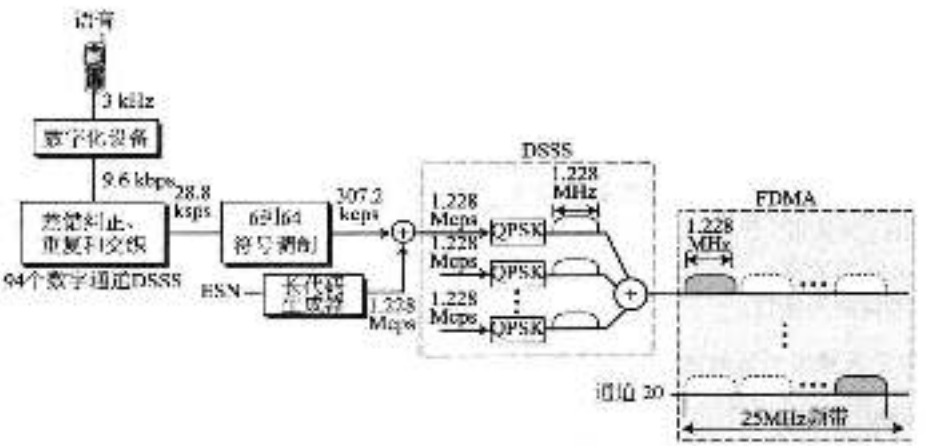


图 6-44　IS-95 反向传输

每一个语音通道都被数字化，以 9.6kbps 的速率生成数据。然而，在添加纠错、重复位和交织

位后，结果是 28.8kbps 的信号。这个输出结果经过一个 6/64 符号调制器。这些符号被分为六符号的大块，每一大块用一个二进制数字来说明（从 0 到 63）。这个二进制数字用作选择码片行的 64 × 64 Walsh 矩阵的索引。注意该过程不是 CDMA；每位没有被行中的码片相乘。每个六符号的块由一个 64 码片代码代替。这样做是为了提供一种正交性，它与从不同移动站点发出的码片就有区别。结果生成了 307.2kcps 或是(28.8/6) × 64 的信号。

下一步是扩频：每一个芯片被扩充为 4 个。另外，移动站点的 ESN 以 1.228Mcps（307.2Kcps 的 4 倍）的速率创建一个 42 位的长代码。扩频后，每一个信号使用 QPSK 调制，这里的 QPSK 和在前向方向中使用的有细微差别；在此我们不讨论细节。注意这里没有多路访问机制；所有的反向通道将它们的模拟信号发送至空气，但是由于扩频，基站只接收正确的码片。

尽管在反向方向上我们创建 $2^{42} - 1$ 个数字通道（因为长代码生成器），但通常只使用 94 个通道；62 个是流量通道，32 个通道用于和基站接入。

IS-95 是使用 CDMA/DSSS 和 FDMA 的数字蜂窝电话系统。

两套数据速率集　IS-95 定义了两套数据速率集，每套中有 4 个不同的速率。第一套定义了 9600、4800、2400 和 1200bps。例如，如果选定的速率是 1200bps，每一位重复 8 次以提供 9600bps 的速率。第二套定义了 14 400、7200、3600 和 1800bps。通过减少用于纠错的位数，这是可行的。比特率与通道的活动状态有关。如果通道是静默的，只有 1200 位能够传输，通过每位重复 8 次来改善扩频。

频率复用因子　在 IS-95 系统中，频率复用因子通常是 1，因为邻近的信元不会影响 CDMA 或 DSSS 传输。

软切换　每个基站不断地使用导频通道广播信号。这意味着移动站点可以从所在信元和邻近信元检测导频信号。相对硬切换而言，这可以使移动站点做软切换。

第三代（3G）

蜂窝电话的第三代是指提供数字数据和语音通信技术的组合。使用小的便携式设备，人们能够和世界上任何人通话，并且通话质量和现有的固定电话网络类似。个人可以下载观看一部电影，下载收听一首音乐，网上冲浪或者玩游戏，开视频会议等等。第三代系统的有趣的特性之一是便携式设备总是连接的，不需要拨号即可连接至因特网。

第三代概念开始于 1992 年，当时 ITU 颁布了一个称为**因特网移动通信-2000**（Internet Mobile Communication 2000，IMT-2000）的蓝图。该蓝图定义了 3G 技术的一些原则，如下所述。

a. 语音质量可以和现有的公共电话网络相比。

b. 移动交通工具（汽车）的接入速率为 144kbps，当用户行走（步行者）时接入速率为 384kbps，固定用户（办公室或家里）的接入速率为 2Mbps。

c. 支持分组交换和电路交换的数据服务。

d. 2GHz 的频带。

e. 2MHz 的带宽。

f. 到因特网的接口。

第三代蜂窝电话的主要目标是提供普通的个人通信。

IMT-2000 无线电接口

图 6-45 显示了 IMT-2000 采用的无线电接口（无线标准）。所有的这五个接口都是从第二代技术发展而来的。前两个是从 CDMA 技术演化而来。第三个由 CDMA 和 TDMA 的组合演化而来。第四个由 TDMA 演化而来，最后的由 FDMA 和 TDMA 演化而来。

IMT-DS　该方法使用 CDMA 的一个称为宽带 CDMA 或是 W-CDMA 版本。W-CDMA 使用 5MHz 的带宽。它在欧洲发展起来，并和在 IS-95 中使用的 CDMA 兼容。

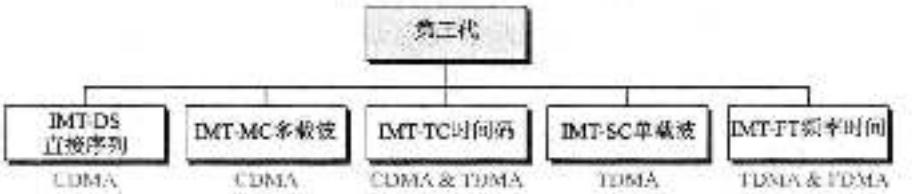


图 6-45　IMT-2000 无线电接口

IMT-MC　该方法在北美发展起来，并以 CDMA 2000 而闻名。它是 IS-95 通道中使用的 CDMA 技术的演化。它联合使用新的宽带（15MHz）扩频和 IS-95 中的窄带（1.25MHz）CDMA。它向后兼容 IS-95，允许多路 1.25MHz 的通道（1、3、6、9、12 倍），最高可达 15MHz。更宽通道的使用使其可以达到第三代标准定义的 2Mbps 的数据速率。

IMT-TC　该标准结合使用 W-CDMA 和 TDMA。该标准通过将 TDMA 多路技术加入到 W-CDMA 中而努力达到 IMT-2000 的目标。

IMT-SC　该标准只使用 TDMA。

IMT-FT　该标准联合使用 FDMA 和 TDMA。

第四代（4G）

蜂窝电话的第四代预期是无线通信中的一个完整解决方案，由 4G 工作组定义的一些目标如下。

a. 一个非常有效的系统。

b. 高网络容量。

c. 移动的汽车中接入速率为 100Mbit/s，静止用户的接入速率为 1Gbit/s。

d. 世界上任意两点之间的数据速率至少为 100Mbit/s。

e. 异构网络的平滑切换。

f. 无缝的连通性及通过多个网络全球漫游。

g. 为下一代多媒体提供高质量的服务（服务质量将在第 8 章中讨论）。

h. 与现有无线标准互通。

i. 全 IP 分组交换网络。

第四代是只基于分组的（不像 3G），并且支持 IPv6。这提供了更好的广播、安全性和路由最佳化的能力。

访问模式

为了提高效率，容量和可扩展性，正在为 4G 考虑新的访问技术。例如，正在为下一代通用移动通信系统（Universal Mobile Telecommunications System，UMTS）的下行链路和上行链路而分别考虑正交 FDMA（Orthogonal FDMA，OFDMA）和交织 FDMA（interleaved FDMA，IFDMA）。类似地，多重载波码分多址（Multi-Carrier Code Division Multiple Access，MC-CDMA）是为 IEEE 802.20 标准而提出的。

调制

更加有效的正交调幅（64-QAM）为长期演进（Long Term Evolution，LTE）标准的使用而提出。

无线电系统

第四代使用软件定义的无线电（Software Defined Radio，SDR）系统。不像普通的无线电要使用硬件，SDR 的组件是很多软件，因此很灵活。SDR 可以改变它的程序来转换它的频率以减少频率干扰。

天线

多输入多输出（multiple-input multiple-output，MIMO）和多用户 MIMO（Multi-User MIMO，MU-MIMO）天线系统是智能天线的一个分支，为 4G 而提出。使用这种天线系统以及特别的多路技术，4G 允许同时从所有的天线传输独立的流以成倍增加数据速率。当干扰发生时，MIMO 也允许发射器和接收器协调以移动到一个开放频率。

应用

以目前 15～30Mbit/s 的速率，4G 能够为用户提供高清晰度电视流，以 100Mbit/s 的速率，DVD-5 的内容可以在 5 分钟时间内下载以用于离线访问。

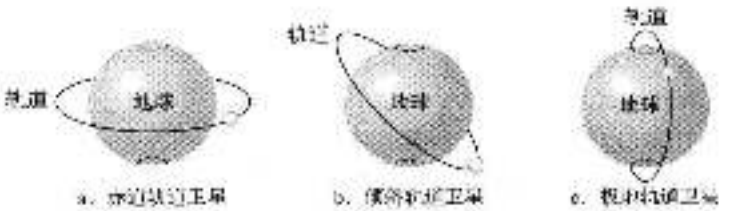
6.2.3　卫星网络

卫星网络（satellite network）是多个结点的联合体，其中一些是卫星，提供从地球上一点到另一点的通信。网络中的结点可以是卫星、地面站、终端用户或电话。尽管天然卫星，例如月球，可以作为该网络中的中继结点，但是倾向于使用人造卫星，因为我们可以在卫星上安装电子设备来生成在传输过程中损失能量的信号。使用天然卫星的另一个限制是它们离地球的距离，这将在通信过程中产生很大的延迟。

卫星网络像蜂窝网络一样，它们将地球划分成信元。卫星能够提供到达或者来自地球上任意地点的传输能力，而不管多远。这种优势使得不需要大量的地面基础设施的投资而提供与世界上不发达地区的高质量通信成为可能。

轨道

人造卫星需要有一个轨道（orbit），围绕地球旋转的路径。轨道可以是赤道轨道、倾斜轨道或极地轨道，如图 6-46 所示。



a. 赤道轨道卫星　　b. 倾斜轨道卫星　　c. 极地轨道卫星

图 6-46　卫星轨道

卫星的周期，即卫星绕地球旋转一周所需的时间，由开普勒定律决定，该定律定义了周期是一个卫星距地球中心的距离的函数。

例 6.4　根据开普勒定律，月球的周期是多少？

$$\text{周期} = C \times \text{距离}^{1.5}$$

这里，C 是一个常数，约等于 1/100。周期的单位是 s，距离的单位 km。

解答

月球距离地面大约 384 000km。地球的半径是 6378km。应用上述公式，我们得到：

$$\text{周期} = (1/100) \times (384\,000 - 6378)^{1.5} = 2\,439\,090\text{ s} = 1\text{ 个月}$$

例 6.5　根据开普勒定律，一个定位在距离地面大约 35 786km 的轨道的卫星的周期是多少？

解答　应用开普勒公式，我们得到：

$$\text{周期} = (1/100) \times (35\,786 + 6378)^{1.5} = 86\,579\text{ s} = 24\text{ h}$$

这意味着一个定位于 35 786km 的卫星的周期是 24h，这和地球自转的周期相同。这样的卫星相对于地球是静止的。这样的轨道，如我们所见，称为地球同步轨道（geostationary orbit）。

覆盖区域

卫星使用双向天线（在视线范围内）进行微波通信。因此，来自于卫星的信号通常只瞄准于一个特定的区域，该区域称为覆盖区域（footprint）。在覆盖区域中心的信号的能量是最大的。当我们远离覆盖区域中心时，能量减弱。覆盖区域的边界是那些信号能量达到预先定义的极限值的地区。

三种类型的人造卫星

根据轨道的位置，卫星可以分为三类：地球同步卫星（Geostationary Earth Orbit，GEO）、近地卫星（Low-Earth-Orbit，LEO）和中地卫星（Medium-Earth-Orbit，MEO）。

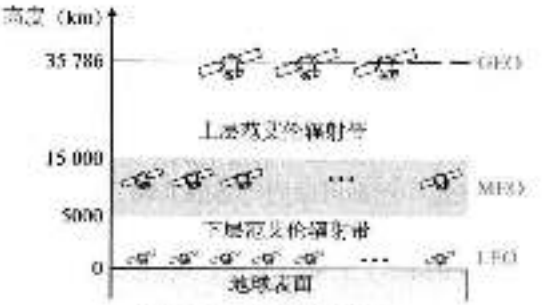


图 6-47　卫星轨道高度

图 6-47 显示了卫星距离地球表面的高度。对于 GEO 卫星来说，只有一个高度在 35 786 的轨道。MEO 卫星定位在高度介于 5000km 到 15 000km 之间。LEO 卫星通常在低于 2000km 的高度。

拥有不同轨道的一个原因是由于两个范艾伦辐射带（Van Allen belts）的存在。范艾伦辐射带是包含带点粒子的层。在这两个带中环绕的卫星将被充满能量的带电粒子完全破坏。MEO 轨道定位在这两个带之间。

卫星通信频带

为卫星微波通信保留的频率处于千兆赫兹（GHz）范围内。每颗卫星通过两个以上不同的频带发送和接收。从地球到卫星的传输称为上行链路（uplink）。从卫星到地球的传输称为下行链路（downlink）。表 6-5 给出了频带名字和每个范围的频率。

表 6-5　卫星频带

频　带	下行链路/GHz	上行链路/GHz	带宽/MHz
L	1.5	1.6	15
S	1.9	2.2	70
C	4.0	6.0	500
Ku	11.0	14.0	500
Ka	20.0	30.0	3500

GEO 卫星

视线传播要求发送和接收天线能够随时地锁定彼此的位置（一个天线必须在另一个天线的视线之内）。为此，比地球自转快或慢的卫星只有在短时间内可以使用。为了确保稳定的通信，卫星必须以与地球同样的速度移动，这样看起来好像在一个特定点保持固定一样。这种卫星称为地球同步轨道（geostationary）卫星。

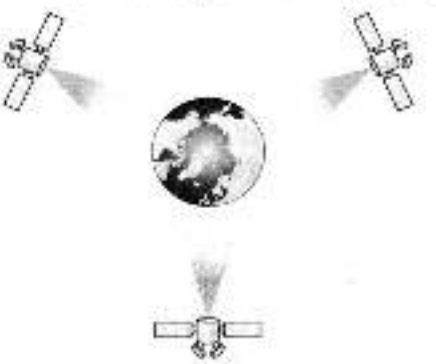


图 6-48　在地球同步轨道上的卫星

因为轨道速度与距地球的距离有关，因此只有一个轨道可以是同步的。这个轨道在地球赤道平面上，距地球表面约 22 000 英里。

但是，一个同步卫星不能覆盖整个地球。尽管一颗在轨道上的卫星可以和视线之内很大数量的地面站保持联系，但是地面的曲率仍然使地球的很多部分处于视线之外。至少需要在地球同步轨道（geostationary Earth orbit，GEO）上等距的三颗卫星来实现全球传输。图 6-48 显示了这三颗卫星，在围绕赤道的同步轨道上，

每个卫星相隔 120°。视点在北极。

MEO 卫星

中地轨道（Medium-Earth-orbit，MEO）卫星定位在两个范艾伦辐射带之间。这个轨道的卫星大约需要 6-8 小时环绕地球一圈。

全球定位系统（GPS）

MEO 卫星系统的一个例子是全球定位系统（Global Positioning System，GPS），轨道位于地球上空 18 000km（11 000 英里）的高度，该系统由美国国防部制定和操作。系统由 24 颗卫星组成，用于地面、海洋和航空导航，为车辆和舰船提供时间和定位服务。GPS 使用 6 个轨道的 24 颗卫星，如图 6-49 所示。每个轨道上的卫星的轨道和位置是基于如下原则设计的，即在任何时间，在地球上的任意一点都可以同时看到 4 颗卫星。每个 GPS 接收器都有天文年日历来得知卫星的当前位置。

三边测量　GPS 基于称为三边测量的原理。术语三边测量（trilateration）和三角测量（triangulation）通常可互换。我们使用三边测量而不用三角测量，意味着使用三个距离而不使用三个角度。在平面上，如果知道与三个点的距离，我们就能准确地知道我们在哪里。假设我们距点 A 10 英里，距点 B 12 英里，距点 C 15 英里。如果我们以 A、B 和 C 点为圆心画三个圆，我们一定在圆 A 的某处、圆 B 的某处及圆 C 的某处。这三个圆相交于一点（如果我们的距离是正确的）；这就是我们的位置。图 6-50a 说明了该概念。

在三维空间中，情况就不同了。三个球体相交于两点，如图 6-50b 所示。我们至少需要需要 4 个球体才能知道我们在空间中的精确位置（经度、纬度和高度）。但是，如果我们有与位置相关的其他信息（例如，我们知道我们不在海洋或空间的其他地方），三个球体就足够了，因为三个球体相交的两个点中，如果选择了另一个，这一个就毫无疑问是不可能的了。

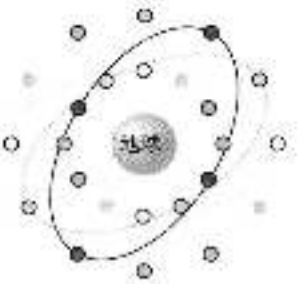
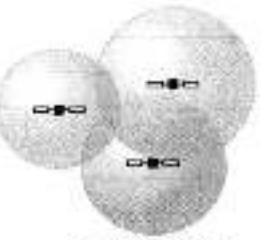


图 6-49　全球定位系统（GPS）卫星的轨道



a. 二维三边测量



b. 三维三边测量

图 6-50　平面三边测量

测距　如果我们知道距三个卫星的距离以及每个卫星的位置，那么就可以使用三边测量原理来找出我们在地球上的位置。每个卫星的位置可以通过 GPS 接收器计算出来（使用卫星预先确定的路径）。然后 GPS 接收器需要获得至少距三个卫星的距离（球心）。测距是通过单向测距的原理实现的。这时，我们假设所有的 GPS 卫星和地面上的接收器是同步的。24 个卫星的每一个同步地传输一个复杂的信号，每个卫星的信号有一个唯一的模式。接收器上的计算机测量来自卫星的信号和信号副本之间的延迟，以确定到各个卫星的距离。

同步　前面的讨论基于一个假设，即卫星的时钟彼此同步，且和接收器时钟也同步。卫星使用精确的原子钟，能够相互同步地工作。但是，接收器的时钟是一般的石英钟（原子钟比石英钟要贵 50000 美元），没有办法与卫星时钟同步。卫星时钟和接收器时钟之间有个未知的偏移，使得测距时有了相应的偏移量。因为这个偏移量，测得距离称为伪距。

GPS 使用一种优秀的解决方案来解决时钟偏移问题。通过分析得知偏移值对所有的卫星都是一样的。未知的计算变成求解四个未知量：x_r、y_r、z_r（接收器的坐标）以及通用时钟偏移 dr。为

了求解这四个变量，我们至少需要四个方程。这意味我们需要测量到四个卫星而非到三个卫星的伪距。如果我们称四个测得的伪距为 PR_1、PR_2、PR_3 和 PR_4，每个卫星的坐标为 x_i、y_i 和 z_i（i 从 1 到 4），我们就能够使用下面的四个方程来求出前面提到的未知量（四个未知量用黑体标出）。

$$PR_1 = [(x_1 - \mathbf{x})^2 + (y_1 - \mathbf{y})^2 + (z_1 - \mathbf{z})^2]^{1/2} + c \times \mathbf{dt}$$

$$PR_2 = [(x_2 - \mathbf{x})^2 + (y_2 - \mathbf{y})^2 + (z_2 - \mathbf{z})^2]^{1/2} + c \times \mathbf{dt}$$

$$PR_3 = [(x_3 - \mathbf{x})^2 + (y_3 - \mathbf{y})^2 + (z_3 - \mathbf{z})^2]^{1/2} + c \times \mathbf{dt}$$

$$PR_4 = [(x_4 - \mathbf{x})^2 + (y_4 - \mathbf{y})^2 + (z_4 - \mathbf{z})^2]^{1/2} + c \times \mathbf{dt}$$

上面的公式使用的坐标在地球中心地球固定（Earth-Centered Earth-Fixed，ECEF）的参考帧中，这表示坐标空间的原点是地球球心，坐标空间绕地球旋转。这说明地球表面固定点的 ECEF 坐标不会改变。

应用　军队也使用 GPS。例如，在海湾战争中，步兵、车辆和直升飞机使用了数以千计的便携式 GPS 接收器。GPS 的另一个应用是导航。汽车驾驶员可以找出汽车的位置。然后驾驶员查阅汽车上存储的数据库，以指引到目的地。换言之，GPS 给出汽车的位置，数据库使用该信息找出到目的地的一条路线。正如我们前面提到的，IS-95 蜂窝电话系统使用 GPS 来实现基站间同步。

LEO 卫星

近地卫星有极地轨道，高度在 500 到 2000km 之间，转动周期为 90 到 120 分钟。卫星的速度为 20 000 到 25 000km/h。一个 LEO 系统通常有移动类型的访问，和蜂窝电话系统类似。覆盖范围通常是直径 8000km 区域。因为 LEO 卫星离地球很近，来回传播的时间延迟通常少于 20ms，这对于视频通信来说是可以接受的。

LEO 系统由像网络一样一起工作的一群卫星组成；每一个卫星扮演交换机的角色。相互邻近的卫星通过卫星间链路（ISL）连接在一起。移动系统通过用户移动链路（UML）和卫星通信。卫星也可以通过网关链路（GWL）和地面站（网关）通信。图 6-51 显示了一个典型的 LEO 卫星网络。

LEO 卫星可以分为三类：小型 LEO，大型 LEO 和宽带 LEO。小型 LEO 运行频率小于 1GHz。它们通常用于低速率数据通信。大型 LEO 运行频率在 1GHz 到 3GHz 之间。全球星（globalstar）是大型 LEO 卫星系统的实例之一。它在 6 个极地轨道上使用 48 个卫星，每个轨道 8 颗卫星。轨道定于大约 1400km 的高度。铱星（Iridium）系统有 66 个卫星，分为 6 个轨道，每个轨道 11 个卫星。轨道在 750km 的高度。每个轨道的卫星大约经纬度 32° 相互隔开。宽带 LEO 提供类似于光纤网络的通信。第一个宽带 LEO 系统是 Teledesic。Teledesic 是一个提供类似光纤通信（宽带通道、低差错率和低延迟）的卫星系统。它的主要目标是为全球用户提供宽带因特网接入服务。有时它被称为“空中因特网”。项目在 1990 年由 Craig McCaw 和 Bill Gates 开启；稍后，其他的投资者加入。项目计划在不久的将来实现全部功能。

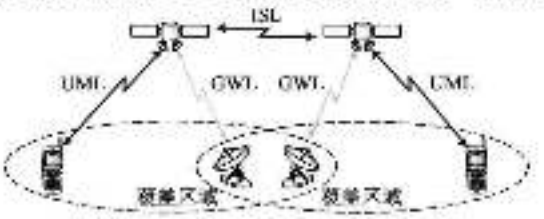


图 6-51　LEO 卫星系统

6.3　移动 IP

随着膝笔记本之类的移动个人电脑逐渐流行起来，我们需要考虑移动 IP，它是 IP 协议的扩展，允许移动计算机在连接可用的任意地点连接至因特网。本节中，我们讨论这个问题。

6.3.1　寻址

使用 IP 协议提供移动通信中必须解决的主要问题是寻址。

静止主机

最初的 IP 寻址基于以下假设：主机是静止的，并附属于一个特定的网络。路由器使用 IP 地址

来路由 IP 数据报。如我们第 5 章学习的，IP 地址有两部分：前缀和后缀。前缀将一个主机和一个网络联系起来。例如，IP 地址 10.3.4.24/8 定义了一台属于网络 10.0.0.0/8 的主机。这意味着因特网中的一台主机没有这样的地址，它可以随身携带从一个地方到另一个地方。该地址只有当主机附属于一个网络时才是有效的。如果网络改变了，地址就不再有效。路由器使用这种想法来路由一个分组；它们使用前缀将分组交付给主机所附属的网络。静止主机这种方案工作得很好。

> 三域址设计用来和静止站点一起工作，因为地址的部分定义了主机附属的网络。

移动主机

当主机从一个网络移动到另一个时，IP 地址结构需要修改。一些解决方案已经提出来了。

改变地址

一个简单的解决方案是当移动主机去往一个新的网络时，让移动站点改变它的地址。主机能够使用 DHCP（见第 4 章）来获取一个新的地址将其和新的网络关联起来。这种方法有几个缺点。第一，配置文件需要更改。第二，每当计算机从一个网络移动到另一个时，必须重启。第三，DNS 表（见第 2 章）需要修改，这样因特网中的每一个其他主机知道这个变化。第四，如果主机在传输过程中从一个网络漫游到另一个，数据交换将会被中断。这是因为客户端和服务器的端口和 IP 地址在整个连接周期中必须是不变的。

两个地址

更加可行的方法是使用两个地址。主机有一个称为归属地址（home address）的原始地址和一个称为转交地址（care-of address）的临时地址。归属地址是永久不变的；它将主机与它的归属网络联系起来，该网络是主机永久不变归属地。转交地址是临时的。当主机从一个网络移动到另一个时，转交地址就会改变；它与外地网络（foreign network）相关联，该网络为主机移动到的网络。图 6-52 说明了这个概念。

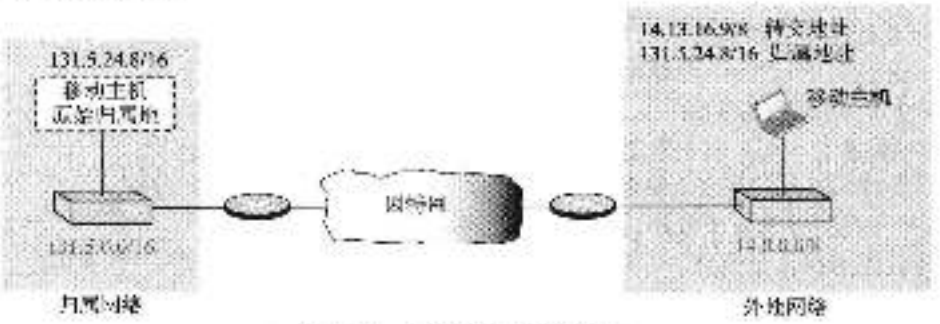


图 6-52　归属地址和转交地址

> 移动 IP 中一个移动主机有两个地址：一个归属地址和一个转交地址。归属地址是永久的；转交地址随着移动主机从一个网络移动到另一个网络而改变。

当一个移动主机访问外地网络时，它在代理发现和注册阶段接收到它的转交地址，稍后描述。

6.3.2　代理

为了使地址的改变对因特网中其他的部分保持透明，就需要一个归属代理（home agent）和一个外地代理（foreign agent）。图 6-53 显示了与归属网络相关联的归属代理的位置和与外部网络相关联的外部代理的位置。

我们已经显示归属代理和外地代理为路由器，但是我们需要强调的是它们作为代理的特定的功能是在应用层完成的。换言之，它们是路由器和主机。



图 6-53　归属代理和外部代理

归属代理

归属代理（home agent）通常是附属于移动主机归属网络的路由器。当远程主机向移动主机发送分组时，归属代理代表移动主机。归属代理接收该分组，将其发送至外地代理。

外地代理

外地代理（foreign agent）通常是附属于外地网络的一个路由器。外地代理接收归属代理发送的分组，并向移动主机发送该分组。

移动主机也可以扮演外地代理的角色。换言之，移动主机和外地代理可以是相同的。但是，移动主机必须能够接收它自己的转交地址，该地址可以通过使用 DHCP 来获取。另外，移动主机需要必要的软件来允许它和归属代理通信，并允许它有两个地址：它的归属地址和转交地址。这种双重的寻址对应用层必须是透明的。

当移动主机作为一个外地代理时，转交地址就称为**配置转交地址**（collocated care-of address）。

当移动主机和外地代理相同时，转交地址称为配置转交地址。

使用配置转交地址的优势是移动主机能够移动到任意的网络，而不用担心外部代理的可用性。其缺点是移动主机需要额外的软件来供当它自己的外部代理。

6.3.3　三个阶段

为了和远程主机通信，移动主机需要通过三个阶段：代理发现、注册和数据传输，如图 6-54 所示。

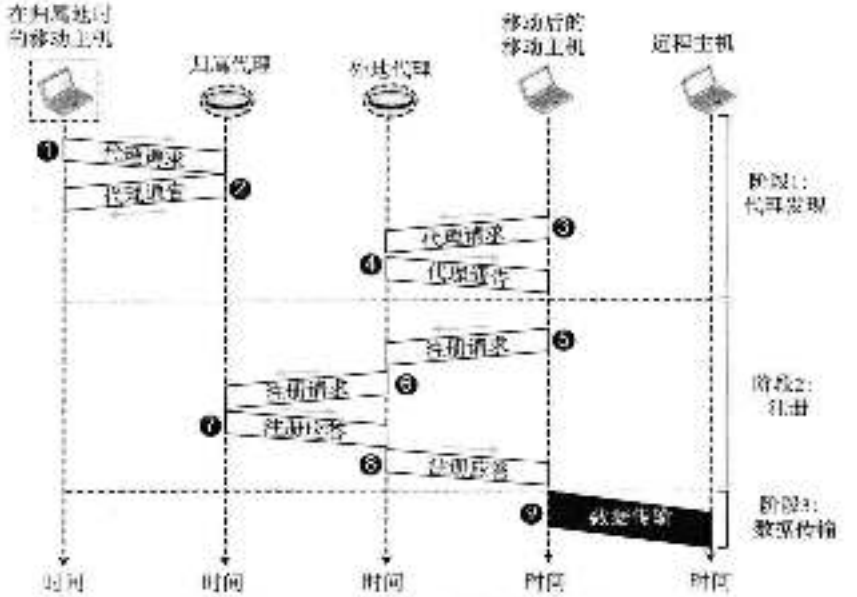


图 6-54　远程主机和移动主机通信

第一个阶段，代理发现，涉及移动主机、外部代理和归属代理。第二个阶段注册，也涉及移动主机和两个代理。最后，第三个阶段还涉及移动主机。我们分别讨论每个阶段。

代理发现

移动通信中的第一个阶段，即代理发现，包含两个子阶段。移动主机必须在它离开其归属网络前发现（知道其地址）归属代理。移动主机必须在它移动到外地网络后也发现外地代理。该发现包含学习转交地址和外地代理地址。发现包含两类信息：通告和请求。



图 6-55　代理通告

代理通告

当路由器使用 ICMP 路由通告通知网络中它的存在时，如果它担当代理，它可以将代理通告添加至分组。图 6-55 显示了代理通告是如何装载在路由器通告分组中的。

移动 IP 不为代理通告使用新的分组；它使用 ICMP 的路由器通告分组，并且附加代理通告信息。

域描述如下：

- **类型**。8 位类型域被设置为 16。
- **长度**。8 位长度域定义了扩展信息的总长度（不是 ICMP 通告信息的长度）。
- **序列号**。16 位的序列号域保存信息编号。接收方可以使用序列号来决定信息是否丢失。
- **生命期**。生命期域定义了代理将会接收请求的秒数。如果该值是一串 1，生命期是无限的。
- **编码**。编码域是一个 8 位的标志，其中每一位是设定的（1）或是未设定的（0）。位的意义如表 6-6 所示。

表 6-6　编码位

位	意　义	位	意　义
0	需要注册。没有配置转交地址	4	代理使用最小封装
1	代理忙并且此刻不接受注册	5	代理使用通用路由封装（GRE）
2	代理支持归属代理	6	代理支持头部压缩
3	代理支持外地代理	7	未使用（0）

- **转交地址**。该域包含代转交地址使用的可用地址列表。移动主机能够选择这些地址中的一个。转交地址的选择在注册请求中宣布。注意该域只供外地代理使用。

代理请求

当一个移动主机已经移动到一个新的网络，并且没有接收到代理通告时，它可以开始代理请求。它可以使用 ICMP 请求信息来通知一个代理它需要帮助。

移动 IP 不为代理请求使用新的分组类型；它使用 ICMP 的路由器请求分组。

注册

移动通信的第二个阶段是注册。移动主机移动到一个外地网络并发现外地代理之后，它必须注册。注册有四个方面：

1. 移动主机必须在外地代理上注册。
2. 移动主机必须在归属代理上注册。这通常由外地代理代表移动主机完成。

3. 如果移动主机已经过期，它必须更新注册。

4. 当移动主机回到归属地时，它必须取消登记（撤销登记）。

请求和应答

为了在外地代理和归属代理上注册，移动主机使用注册请求和注册应答，如图 6-54 所示。

注册请求　注册请求由移动主机向外地代理发送来注册它的转交地址，并且宣布它的归属地址和归属代理地址。外地代理在接收和注册该请求后，将信息中继至归属代理。注意，归属代理要在知道外地代理的地址，因为用于中继的 IP 分组有外地代理的 IP 地址作为其源地址。图 6-56 显示了注册请求的格式。



图 6-56　注册请求格式

域描述如下：

- 类型。8 位的类型域定义了信息的类型。对于请求信息来说，该域的值为 1。
- 标识。8 位的标识域定义了转发信息。每一位的值可以设置也可以不设置。每一位的意义在表 6-7 中给出。

表 6-7　注册请求标识域位

位	意　义	位	意　义
0	移动主机请求归属代理保存它先前的转交地址	4	移动主机请求通用路由封装（GRE）
1	移动主机请求归属代理隧道任意广播信息	5	移动主机请求头部压缩
2	移动主机正在使用配置转交地址	6-7	保留位
3	移动主机请求归属代理使用最小封装		

- 生命期。该域定义了注册有效的秒数。如果该域是一串 0，请求信息正在取消登记。如果该域是一串 1，生命期是无限的。
- 归属地址。该域包含移动主机的永久地址（主要的）。
- 归属代理地址。该域包含了归属代理的地址。
- 转交地址。该域是移动主机的临时地址（次要的）。
- 鉴定。该域包含一个 64 位的数，由移动主机等其插入请求中，在应答信息中重复。它匹配请求和应答。
- 扩展。可变长度的扩展用于认证。它们允许归属代理来认证移动代理。我们在第 10 章讨论认证。

注册应答　注册应答从归属代理发送至外地代理，然后中继至移动主机。该应答确认或是拒绝注册请求。图 6-57 显示了注册请求的格式。



图 6-57　注册请求格式

这些域和注册请求中的那些类似，除了以下例外。类型域是 3。编码域代替了标识域，说明了注册请求的结果（接受或是拒绝）。不再需要转交地址域。

封装

注册信息封装在一个 UDP 用户数据报中。代理使用熟知的端口 434；移动主机使用临时端口。

注册请求或应答使用熟知的端口 434 通过 UDP 发送。

数据传输

在代理发现和注册后，移动主机能够和一个远程主机通信。图 6-58 说明了该概念。

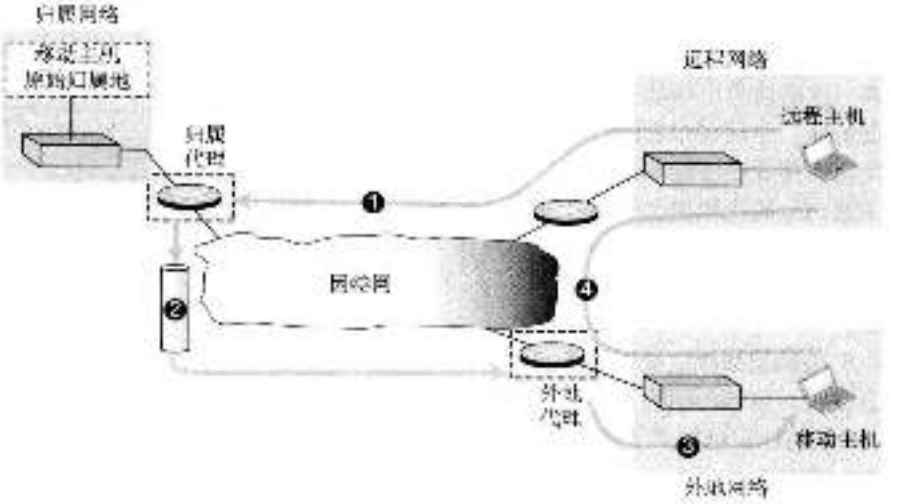


图 6-58　数据传输

从远程主机到归属代理

当一个远程主机想要向移动主机发送分组时，它使用它的地址作为源地址，移动主机的归属地址作为目的地址。换言之，远程主机发送分组好像移动主机在它的归属网络一样。但是，分组被归属代理拦截，它假装它就是移动主机。这通过使用代理 ARP 技术完成，这在第 5 章中讨论过。图 6-58 中的路径 1 显示了该步骤。

从归属代理到外地代理

在接收到分组后，归属代理向外地代理发送分组，其中使用第 4 章中讨论的隧道概念。归属代理将整个 IP 分组封装到另一个 IP 分组中，使用它的地址作为源地址，外地代理的地址作为目的地址。图 6-58 中的路径 2 显示了该步骤。

从外地代理到移动主机

当外地代理接收到分组时，它移除原始分组。但是，由于目的地址是移动主机的归属地址，外地代理查阅一个注册表来查找移动主机的转交地址。（否则，分组将只发回至归属网络。）然后分组被发送至转交地址。图 6-58 的路径 3 显示了该步骤。

从移动主机到远程主机

当移动主机想要向远程主机发送分组时（例如，对它接收到的分组的响应），它如同正常发送一样。移动主机准备分组，用它的归属地址作为源地址，远程主机的地址作为目的地址。尽管分组来自于外地网络，它包含了移动主机的归属地址。图 6-58 中的路径 4 显示了该步骤。

透明

在数据传输过程中，远程主机不知道移动主机的任何移动。远程主机使用移动主机的归属地址作为目的地址发送分组；它接收具有移动主机归属地址作为源地址的分组。移动完全是透明的。因特网的剩余部分不知道移动主机的移动性。

移动主机的运动对于因特网的其余部分是透明的。

6.3.4　移动 IP 的低效

包含移动 IP 的通信的效率很低。低效性可能是严重的或是适度的。严重的情况称为双重交换

（double crossing）或是 2X。适度的情况称为三角路由（triangle routing）或是迂回路由（dog-leg routing）。

双重交换

当远程主机和一个移动到与远程主机相同的网络（站点）的移动主机通信时，双重交换（double crossing）就会发生（见图 6-59）。

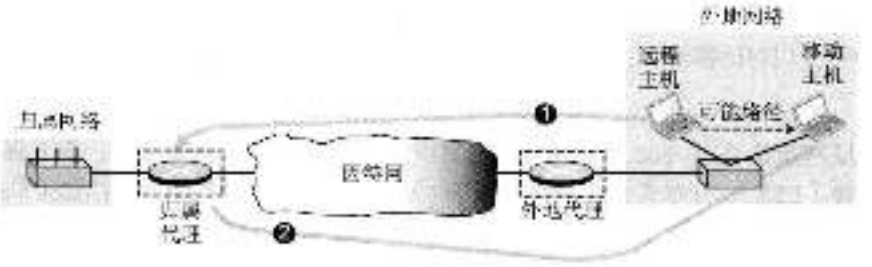


图 6-59　双重交换

当移动主机向远程主机发送分组时，没有低效性；通信是本地的。但是，当远程主机向移动主机发送分组时，分组通过因特网两次。

由于计算机通常和其他的本地计算机（本地性原则）通信，由于双重交换造成的低效性是有意义的。

三角路由

三角路由（triangle routing）是不严重的情况，当远程主机和不附属于移动主机相同网络（站点）的移动主机通信时，会发生三角路由。当移动主机向远程主机发送分组时，没有低效性。但是，当远程主机向移动主机发送分组时，分组从远程主机去往归属代理，然后去往移动主机。分组经过三角形的两条边，而非只经过一条边（见图 6-60）。

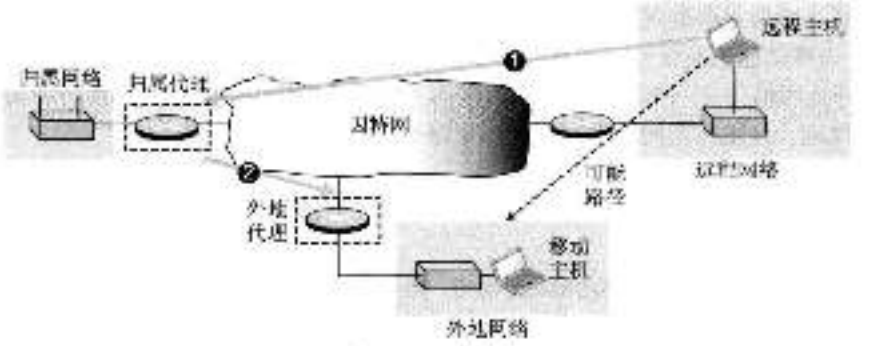


图 6-60　三角路由

解决方案

一种解决低效性的方案是远程主机将转交地址和移动主机的归属地址绑定。例如，当一个归属代理接收到去往移动主机的第一个分组时，它将分组转发至外地代理；它也可以向远程主机发送一个更新绑定分组（update binding packet），这样以后去往该主机的分组就直接发送至转交地址。远程主机可以在缓存中保留该信息。

使用这种策略的问题是当移动主机移动时，缓存项就过时了。这种情况下，归属代理需要向远程主机发送一个警告分组（warning packet）以通知远程主机该变化。

6.4　章末资料

推荐读物

要获取本章中讨论的主题的更多详细信息，我们推荐下面的书籍和 RFC。在方括号中的项目

涉及本书结尾的参考文献。

书籍

很多书籍涵盖了本章中讨论的材料。包括[Sch 03]、[Gas 02]、[For 03]、[Sta 04]、[Sta 02]、[Kei 02]、[Jam 03]、[AZ 03]、[Tan 03]、[Cou 01]、[Com 06]、[G & W 04]和[PD 03]。

RFC

一些 RFC 专门讨论移动 IP：RFC 1701、RFC 2003、RFC 2004、RFC 3024、RFC 3344 和 RFC 3775。

小结

无线局域网随着 IEEE 802.11 标准变得正式化，该标准定义了两种服务：基本服务集（BSS）和扩展服务集（ESS）。分布式协调功能（DCF）MAC 子层中使用的访问方法是 CSMA/CA。点协调功能（PCF）MAC 子层中使用的访问方法是轮询。蓝牙是一种无线局域网技术，它将小范围内的设备（称为小设备）连接起来。蓝牙网络称为微微网络。WiMAX 是一个无线访问网络，将来可能代替 DSL 和电缆。

蜂窝电话提供两个设备之间的通信。一个或两个可能是移动的。蜂窝服务区域划分为信元。高级移动电话系统（AMPS）是第一代蜂窝电话系统。数字 AMPS（D-AMPS）是第二代蜂窝电话系统，它是 AMPS 的数字版本。全球移动通信系统（GSM）是在欧洲使用的第二代蜂窝电话系统。过渡标准 95（IS-95）是基于 CDMA 和 DSSS 的第二代蜂窝电话系统。第三代蜂窝电话系统提供普遍的个人通信。第四代是新一代的蜂窝电话，正在变得流行。

卫星网络使用卫星来提供地球上任意点间的通信。地球同步卫星（GEO）在赤道平面上并与地球同步旋转。全球定位系统（GPS）卫星是中地卫星（MEO），为车辆和轮船提供时间和位置信息。铱星卫星是近地卫星（LEO），为手持终端设备提供直接普遍的语音和数据通信。Teledesic 卫星是近地卫星，能够提供适用的宽带因特网接入服务。

移动 IP 为移动通信而设计，是 IP 协议的增强版本。移动主机在它的归属网络中有一个归属地址，在它的外地网络中有一个转交地址。当移动主机在外地网络时，归属代理中继信息（为移动主机）至外地代理。外地代理向移动主机发送中继的信息。

6.5　习题集

测试题

本章的交互测验题集可以在本书的网站上找到。强烈推荐学生做这些测验题，这样可以在做练习题前检测他对课程资料的理解。

练习题

Q6-1　比较现在通信环境中有线局域网的介质和无线局域网的介质。

Q6-2　解释为什么无线局域网中的 MAC 协议比有线局域网中更重要。

Q6-3　解释为什么无线局域网比有线局域网会有更多的差错，忽略噪声和干扰。

Q6-4　为什么无线局域网中的 SNR 通常比有线局域网中的低？

Q6-5　多路径广播是什么？它对无线网络有什么影响？

Q6-6　无线局域网中不能使用 CSMA/CD 的原因？

Q6-7　列举 CSMA/CA 中用于避免冲突的一些策略。

Q6-8　在无线局域网中，指定站点 A 的 IFS = 5ms，站点 B 的 IFS = 7ms。哪一个站点有更高的优先级？解释原因。

Q6-9　CSMA/CD 机制中没有确认机制，但是我们在 CSMA/CA 中需要这种机制。解释原因。

Q6-10　CSMA/CA 中 NAV 的目的是什么？

Q6-11　解释为什么无线局域网中推荐分段？

Q6-12　解释为什么在有线局域网中我们只有一种帧类型，但是无线局域网中有 4 种？

Q6-13　802.3（有线以太网）中使用的 MAC 地址和 802.11（无线以太网）中使用的 MAC 地址是否属于两个不同的地址空间？

Q6-14　一个 AP 可以有无线网络和有线网络相连。这种情况下 AP 是否需要两个 MAC 地址？

Q6-15　无线网络中的 AP 和有线网络中的链路层交换机扮演相同的角色。但是，链路层交换机没有 MAC 地址，而 AP 通常需要一个 MAC 地址。解释其原因。

Q6-16　蓝牙通常称为无线个人区域网（wireless personal area network，WPAN），而不被为无线局域网（wireless local area network，WLAN）的原因是什么？

Q6-17　比较蓝牙体系中的微微网和散射网。

Q6-18　该微网是否可以多于 8 个的站点？解释原因。

Q6-19　蓝牙网络中用于通信的实际带宽是多少？

Q6-20　蓝牙中的无线电层协议什么角色？

Q6-21　填空。蓝牙中的 83.5MHz 被分为____通道，每个____MHz。

Q6-22　蓝牙中使用的扩频技术是什么？

Q6-23　蓝牙无线电层中的调制技术是什么？换言之，数字数据（位）是如何变成模拟信号的（无线电波）？

Q6-24　蓝牙中基带层使用的 MAC 协议是什么？

Q6-25　蓝牙中 L2CAP 层的角色是什么？

Q6-26　什么通道化方法中，站点按时间共享可用带宽？

a. FDMA　　b. TDMA　　c. CDMA

Q6-27　什么通道化方法中，站点按频率共享可用带宽？

a. FDMA　　b. TDMA　　c. CDMA

Q6-28　什么通道化方法中，站点指定不同的编码？

a. FDMA　　b. TDMA　　c. CDMA

Q6-29　假设两个蜂窝电话用户使用 FDMA 通信。如果每个站点指定不同的频率频带，两个站点如何相互通信？

Q6-30　TDMA 中，如果每个站点指定不同的时隙，两个站点如何相互通信？

Q6-31　CDMA 中，如果每个站点指定不同的编码，两个站点如何相互通信？

Q6-32　假设系统中有 8 个站点使用 CDMA。下面编码的内积是多少？

a. $c_1 \cdot c_1$　　b. $c_1 \cdot c_6$　　c. $c_6 \cdot c_1$

Q6-33　CDMA 中，对于下面指定站点个数的系统，其序列数是多少？

a. 8 个站点　　b. 12 个站点　　c. 28 个站点

Q6-34　长度为 4 的编码可以创建多少？这些编码中有多少是正交的？

Q6-35　下面的蜂窝电话系统分别属于哪一代？

a. AMPS　　b. D-AMPS　　c. IS-95

Q6-36　在移动 IP 中，如果移动主机担当外地代理，是否需要注册？解释其原因。

Q6-37　讨论 ICMP 路由请求消息如何用于代理请求。

Q6-38　解释为什么注册请求和应答不是直接分装在 IP 数据报中。为什么需要 UDP 用户数据报？

思考题

P6-1　在 802.11 中，下面情况中，给出地址 1 域的值（左边位定义 To DS，右边位定义 From DS）。

a. 00　　b. 01　　c. 10　　d. 11

P6-2　在 802.11 中，下面情况中，给出地址 2 域的值（左边位定义 To DS，右边位定义 From DS）。

a. 00　　b. 01　　c. 10　　d. 11

P6-3　在 802.11 中，下面情况中，给出地址 3 域的值（左边位定义 To DS，右边位定义 From DS）。

a. 00　　b. 01　　c. 10　　d. 11

P6-4　在 802.11 中，下面情况中，给出地址 4 域的值（左边位定义 To DS，右边位定义 From DS）。

a. 00　　b. 01　　c. 10　　d. 11

P6-5　在没有 AP 的 BSS 中（自组织网络）中，我们有 5 个站点：A、B、C、D 和 E。站点 A 需要向站点 B 发送一个信息。网络使用 DCF 协议的情况下，回答下面问题：

a. 在交换帧中，To DS 和 From DS 位的值是多少？

b. 哪个站点发送 RTS 帧？该帧中地址域的值是多少？
c. 哪个站点发送 CTS 帧？该帧中地址域的值是多少？
d. 哪个站点发送数据帧？该帧中地址域的值是多少？
e. 哪个站点发送 ACK 帧？该帧中地址域的值是多少？

P6-6 在图 6-61 中，两个无线网络 BSS1 和 BSS2 通过有线分布式系统（DS）（一个以太局域网）相互连接。假设 BSS1 中的站点 A 需要向 BSS2 中的站点 C 发送数据帧。说明以下三个阶段中 802.11 和 802.3 帧中的地址的值：从站点 A 到 AP1，从 AP1 到 AP2，从 AP2 到站点 C。注意 AP1 和 AP2 之间的通信发生于有线环境中。

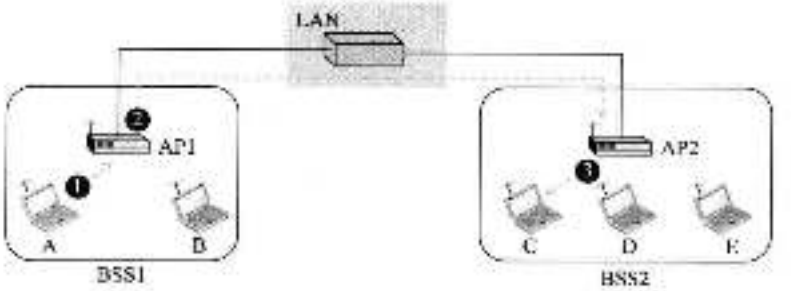


图 6-61　思考题 P6-6

P6-7 重复上一题（见图 6-61），但是假设分布式系统也是无线的。AP1 通过无线通道连接至 AP2。说明所有通信阶段中地址的值：从 A 到 AP1，从 AP1 到 AP2，从 AP2 到站点 C。

P6-8 假设来自于使用 802.3 协议的有线网络的一个帧向使用 802.11 协议的无线网络移动。说明 802.11 帧中的域如何使用 802.3 帧的值来填充。假设转换发生在 AP，该 AP 是两个网络之间的边界。

P6-9 假设来自于使用 802.11 协议的无线网络的一个帧向使用 802.3 协议的有线网络移动。说明 802.3 帧中的域如何使用 802.11 帧的值来填充。假设转换发生在 AP，该 AP 是两个网络之间的边界。

P6-10 假设两个 802.11 无线网络通过一个路由器连接至因特网的其余部分，如图 6-62 所示。路由器接收到一个 IP 数据报，其目的 IP 地址为 24.12.7.1，并且需要将其发送至相应的无线主机。解释该过程，并描述这种情况下，地址 1、地址 2、地址 3 和地址 4（见图 6-62）的值是如何决定的。

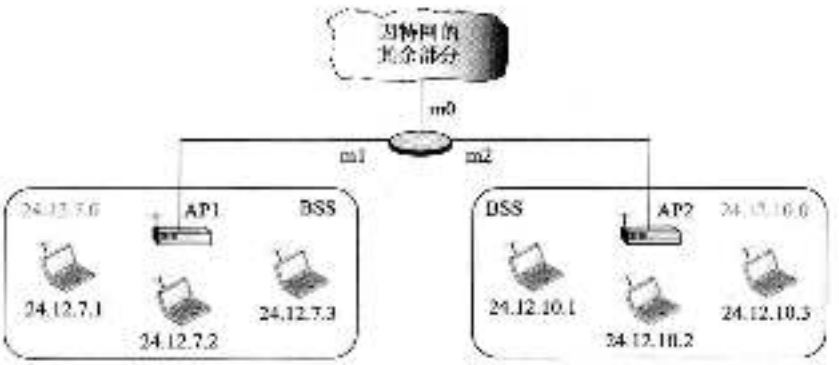


图 6-62　思考题 6-10

P6-11 在图 6-62 中（上一个问题），假设 IP 地址为 24.12.10.3 的主机需要向世界其他的 IP 地址为 128.41.23.12 的主机（未在图中显示）发送一个 IP 数据报。解释该过程，并描述这种情况下，地址 1、地址 2、地址 3 和地址 4（见图 6-62）的值是如何决定的。

P6-12 一个 BSS ID（BSSID）是 802.11 网络中指定给一个 BSS 的 48 位地址。做一些研究并查找在自组织网络和基础设施网络中 BSSID 起什么作用，BSSID 是如何指定的。

P6-13 做一些研究并查找使用 DCF MAC 子层的 802.11 网络中流量控制和差错控制是如何完成的。

P6-14　在 802.11 通信中，负载的长度（帧主体）是 1200 字节。站点决定将帧分为三段，每个 400 负载字节。回答以下问题：

a. 如果没有做分段，数据帧的长度是多少？

b. 分段后每个帧的长度是多少？

c. 分段后，总共要发送多少字节（忽略额外的控制帧）？

d. 由于分段，要发送多少额外的字节（再一次忽略额外的控制帧）？

P6-15　IP 协议和 802.11 项目都要将分组分段。IP 在网络层将数据报分段；802.11 在数据链路层将帧分段。使用每个协议中不同的域和子域来对比两种分段方案。

P6-16　在一个 802.11 网络中，假设站点 A 要向站点 B 发送 4 个分段。如果第一个分段序列号被选为 3273，更多的分段标识、分段号和序列号是多少？

P6-17　在一个 802.11 网络中，站点 A 向站点 B 发送一个数据帧（不分段）。在以下帧中为了 NAV，需要将 D 域的值设置为多少（以微秒为单位）：RTS、CTS、数据和 ACK？假设 RTS、CTS 和 ACK 的传输时间是 4 μs。数据帧的传输时间是 40 μs，SIFS 持续时间被设置为 1 μs。忽略传播时间。注意每个帧需要设置 NAV 的持续时间，以便为介质完成事务保留时间。

P6-18　在 802.11 网络中，站点 A 要向站点 B 发送两个数据分段。在以下帧中为了 NAV，需要将 D 域的值设置为多少（以微秒为单位）：RTS、CTS、数据和 ACK？假设 RTS、CTS 和 ACK 的传输时间是 4 μs。每个分段的传输时间是 20 μs，SIFS 持续时间被设置为 1 μs。忽略传播时间。注意每个帧需要设置 NAV 的持续时间，以便为介质完成事务保留时间。

P6-19　在 802.11 网络中，三个站点（A、B 和 C）正竞争访问介质。每个站点的竞争窗口有 31 个时隙。站点 A 随机选取第 1 个时隙；站点 B 选择第 5 个时隙；站点 C 选择第 21 个时隙。说明每个站点应该遵循的过程。

P6-20　在一个 802.11 网络中，有三个站点 A、B 和 C。站点 C 相对于 A 隐藏，但是可以被 B 看见（电子地）。现在假设站点 A 需要向站点 B 发送数据。由于站点 C 相对于 A 隐藏，RTS 帧不能到达 C。解释站点 C 如何发现通道被 A 预留，它应该抑制传输。

P6-21　一个 802.11 网络可能使用 4 个不同的帧间隔（IFS）来在不同的情况下延迟帧的传输。这允许低优先级的流量在通道空闲时，等待高优先级的流量。正常地，4 个不同的 IFS 用于不同的实现，如图 6-63 所示。解释这些 IFS 的目的（你可能需要使用因特网来做一些研究）。

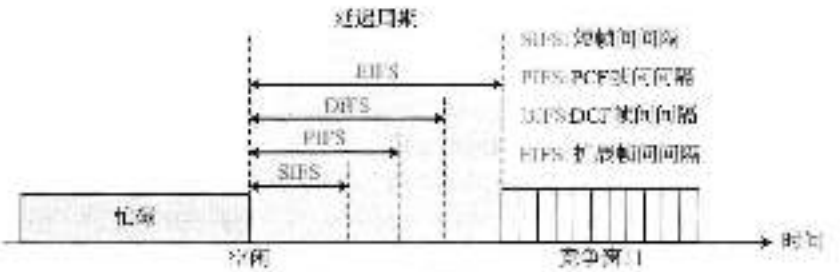


图 6-63　思考题 6-21

P6-22　尽管一个 RTS 帧定义了 NAV 为会话剩余部分有效的时间，为什么 802.11 项目定义，会话中使用的其他帧应该重新定义 NAV 的剩余部分？

P6-23　图 6-24 显示了蓝牙基带层的帧的格式（802.15）。基于这种格式，回答下面的问题：

a. 蓝牙网络中的地址域的范围是什么？

b. 基于图 6-24 中的信息的微微网络中，同时可以有多少个站点是活跃的？

P6-24　查看以下码片集是否属于一个正交系统：

$$[-1, +1] \text{ 和 } [+1, -1]$$

P6-25　查看以下码片集是否属于一个正交系统：

$$[+1, -1, +1, +1] \text{ 、 } [-1, -1, -1, +1] \text{ 、 } [-1, +1, -1, -1] \text{ 和 } [+1, -1, -1, +1]$$

P6-26　Alice 和 Bob 正在使用 W_2 Walsh 表进行 CDMA 试验（见图 6.34）。Alice 使用编码[−1, +1]，Bob 使用编码[+1, −1]。假设他们同时向彼此发送十六进制数字。Alice 发送（6）$_{16}$，Bob 发送(B)$_{16}$。说明他们

如何发现对方发送的内容。

P6-27　画复用因子为 5 的信元模式。

P6-28　依据每兆赫兹带宽同时通话，计算 AMPS 协议的效率。换言之，计算 1MHz 带宽分配中，可以进行的通话的数目。

P6-29　依据每兆赫兹带宽同时通话，计算 D-AMPS 协议的效率。换言之，计算 1MHz 带宽分配中，可以进行的通话的数目。

P6-30　依据每兆赫兹带宽同时通话，计算 GSM 协议的效率。换言之，计算 1MHz 带宽分配中，可以进行的通话的数目。

P6-31　依据每兆赫兹带宽同时通话，计算 IS-95 协议的效率。换言之，计算 1MHz 带宽分配中，可以进行的通话的数目。

P6-32　使用开普勒定律来检查给定周期和高度的 GPS 卫星的准确性。

P6-33　使用开普勒定律来检查给定周期和高度的全球星卫星的准确性。

P6-34　当移动主机和当外地代理时，重画图 6-58。

P6-35　使用 1456 作为序列号，生命期为 3 小时，来创建归属代理通告信息。选择你自己的值用于编码域中的位。计算并插入长度域的值。

P6-36　使用 1672 作为序列号，生命期为 4 小时，来创建外地代理通告信息。选择你自己的值用于编码域中的位。使用至少三个你自己选择的转交地址。计算并插入长度域的值。

P6-37　我们有以下信息。说明从远程主机发送至归属代理的 IP 数据报头部的内容。

移动主机归属代理：130.45.6.7/16
移动主机转交地址：14.56.8.9/8
远程主机地址：200.4.7.14/24
归属代理地址：130.45.10.20/16
外地代理地址：14.67.34.6/8

6.6　模拟实验

Applets

我们已经创建了一些 Java applet 来展示本章讨论的主要概念。我们建议学生激活本书网站上的这些 applet，仔细研究这些协议的运作方式。

实验作业

通过本书的网站来查看如何使用无线局域网模拟器。

Lab6-1　本实验中，我们捕获并研究在无线主机和访问点之间交换的无线帧。在本书网站上查看本实验的详细描述。

6.7　编程作业

使用你所选择的编程语言编写、编译并测试以下程序。

Prg6-1　编写一个程序来模拟图 6-7 中 CSMA/CA 的流程图。

第 7 章

Computer Networks: A Top-Down Approach

物理层与传输介质

如果没有对物理层和能携带由物理层产生的信号的传输介质的讨论，那么我们关于 TCP/IP 协议族的讨论就是不完整的。但是，如果读者有一些关于物理层的认识可以部分或者整个跳过这一章。我们应该说物理层的作用就是传送从数据链路层接收的位并把这些位转化成传输用的电磁信号。本章前四节讨论这个问题。在位转化为信号后被转交给传输介质，这个传输介质就是我们最后一节讨论的问题。

- 7.1 节首先讨论数据和信号的关系，然后说明如何将数据和信号模拟化和数字化。在此基础上，讨论了模拟信号和数字信号及其特点。我们还讨论了一些涉及物理信道容量和性能的问题。
- 7.2 节主要讨论数字传输。首先说明了如何将数字数据转化为数字信号，然后说明如何将模拟数据转化为数字信号。
- 7.3 节主要讨论模拟传输。首先说明了如何将数字数据转化为模拟信号，然后说明了如何将模拟数据转化为模拟信号。
- 7.4 节说明了多路复用（multiplexing）技术，可以将多个低带宽信号合并成一个高带宽信号，这种技术允许一个传输介质传输多条信道。然后我们说明如何应用扩频技术来保证信号抗干扰。
- 7.5 节我们向下进入物理层，讨论数据通信中使用的传输介质。介绍了有线介质（电线和电缆）和无线介质（空气）。

7.1 数据和信号

在物理层，通信是点对点的，不过结点间交换的是电磁信号。图 7-1 用了与前 4 章相同的方案，不过这个通信是在物理层上的。

物理层的一个主要功能是在结点间转发位。然而，位以两种形式的值存储在结点（主机、路由器或者交换机）的内存中，不能直接发送到传输介质（有线或者无线）。位需要转换为电磁信号进行传输，所以物理层的主要任务是将这些位高效地转换为电磁信号。我们首先需要了解数据的性质，然后看到信号的类型，看看我们如何能使这个转换高效。

7.1.1 模拟数据与数字数据

数据可分为模拟数据和数字数据。**模拟数据**（analog data）是指连续状态的信息。当人说话时，会在空气中形成模拟形式的声波。能通过麦克风采集到这个声波并转换成模拟信号，或者对这个声波进行采样转换成数字信号。

数字数据采用离散值。例如，数据以 0 和 1 的形式存储在计算机内存中。它们可以转换成数字信号或者调制成模拟信号通过介质进行传输。

就像它们表述的一样，信号（signal）可以是模拟信号也可以是数字信号。**模拟信号**（analog signal）在一段时间内有无穷多个强度等级。当波形振幅由值 A 变为值 B 时，沿着它变化的过程有无穷多个值。另一方面，**数字信号**（digital signal）只能有有限个已定义的数值，通常情况下是简

单的 0 和 1。表示信号的最简单的方法是将它们绘制在直角坐标系中。垂直坐标轴表示信号的值或者强度。水平坐标轴表示时间。图 7-2 说明了一个模拟信号和一个数字信号。

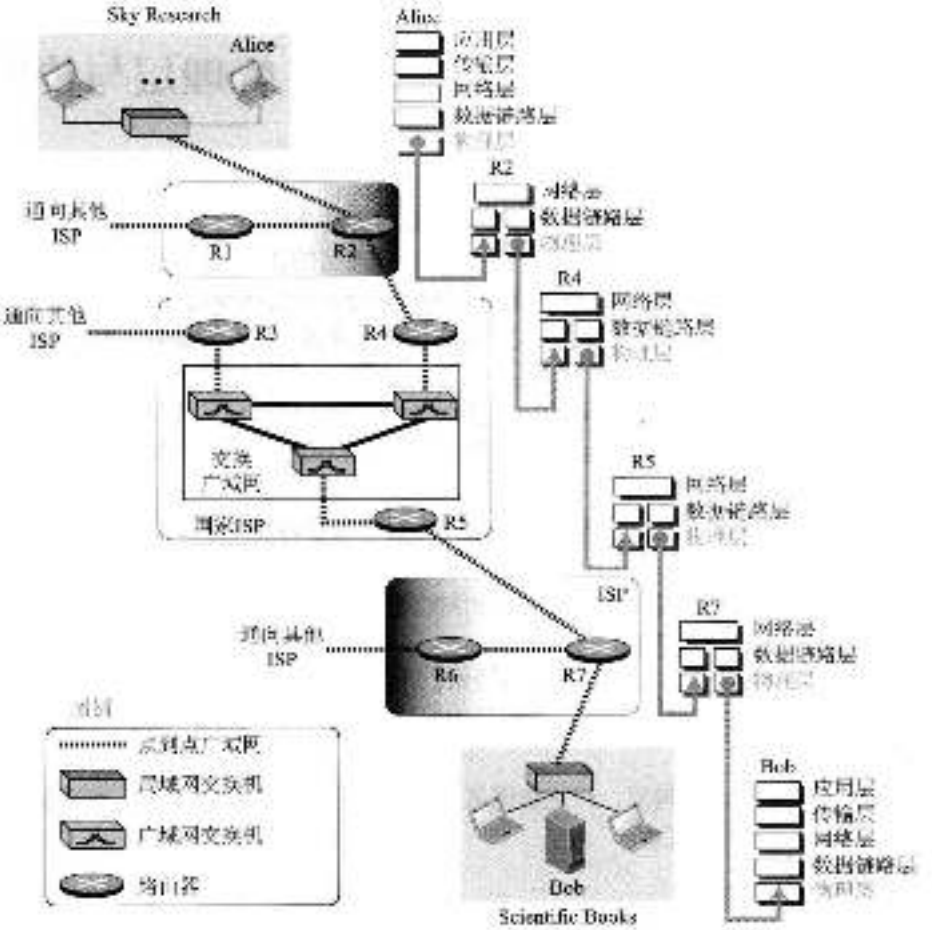


图 7-1　物理层上的通信

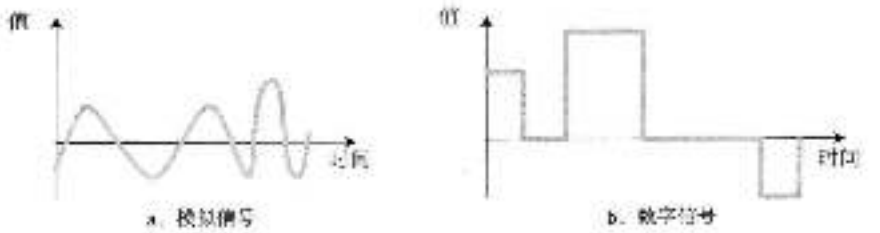


图 7-2　模拟与数字信号的比较

模拟信号

让我们首先主要讨论模拟信号。模拟信号可以以两种形式之一出现：周期信号或者非周期信号。周期模拟信号（periodic analog signal）在一个可测量的时间范围内（称为一个周期）完成一种模式，并且在后续的相同时间内重复这一模式。一个完整模式的实现称为一个循环（cycle）。而非周期模拟信号（nonperiodic analog signal）的变化则不会随着时间的变化而出现重复的模式或循环。在数据传输中，我们一般使用的是周期模拟信号。

周期模拟信号可以分为简单类型或复合类型两种。简单周期模拟信号，即正弦波（sine wave），不能再分解为更简单的信号。而复合周期模拟信号则是由多个正弦波信号组成的。正弦波是周期模拟信号的最基本形式。可以看做是一条简单的震荡曲线，在一个周期内的变化是平滑的、一致的、连续的、起伏的曲线。图 7-3 表示的就是一条正弦波。每个循环由时间轴上方的单弧构成。

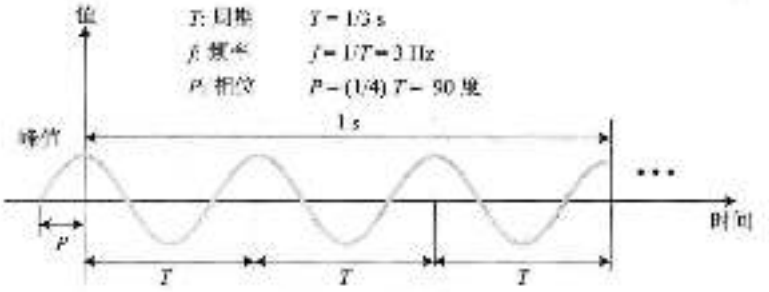


图 7-3　正弦波

一个正弦波可以由三个参数表示：峰值振幅、频率和相位。这三个参数完全决定正弦波。信号的峰值振幅（peak amplitude）是其最高强度的绝对值，与其携带的能量成比例。对于电信号，峰值振幅通常以伏特为单位来计量。图 7-4 表示了两个信号和它们的峰值振幅。周期（period）是信号完成一个循环所需要的时间，以秒为单位。频率（frequency）是指 1 秒内的周期数，单位为赫兹（Hz）。注意周期与频率是按两种方式定义的同一特性。周期和频率互为倒数（$f=1/T$）。

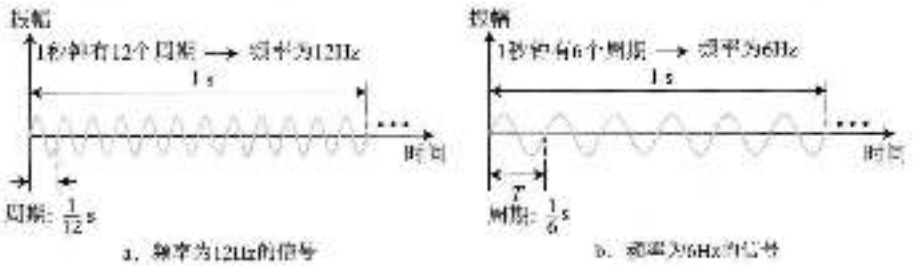


图 7-4　周期和波长

术语相位（phase）描述波形相对于时间零的位置。如果我们将波形想象为能够沿着时间轴向后或者向前平移，那么相位描述的就是这种偏移的位置。它表明第一个循环的状态。相位使用角度或者弧度进行计量（360°等于 2π弧度）。

波长（wavelength）是信号通过传输介质传播的另一个特性。波长是简单类型信号一个周期能传播的距离。波长将简单正弦波的周期或频率与介质的传播速度结合在一起（见图 7-4）。

信号的频率与介质无关，但波长取决于频率和介质。波长是任何信号的一个属性。在数据通信中，我们一般使用波长来描述光纤中的光传输。

如果给定了介质中的传播速度（光速）和信号频率，就可以计算出波长。如果我们用λ表示波长，用 c（光速）表示传播速度，用 f 表示频率，那么我们可以得到：

$$\lambda = c / f = c \cdot T$$

时域和频域

一个正弦波可以通过它的振幅、频率和相位得到完整的定义。前面已经使用时域图（time-domain plot）来表示正弦波。时域图显示了信号振幅随时间的变化情况。为了表示振幅和频率的关系，可以使用频域图（frequency-domain plot）。图 7-5 显示了信号的时域图和频域图。

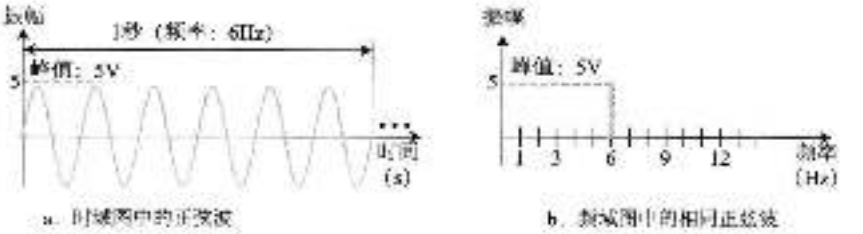


图 7-5　正弦波的时域图和频域图

在频域图中，一个正弦波通过一个尖峰表示。尖峰的位置表示频率，尖峰的高度表示峰值振幅。

例 7.1　当我们处理多个正弦波时，频域更简洁更有用。例如，图 7-6 显示了三个不同振幅和频率的正弦波。它们都能通过频域中的三个尖峰表示。

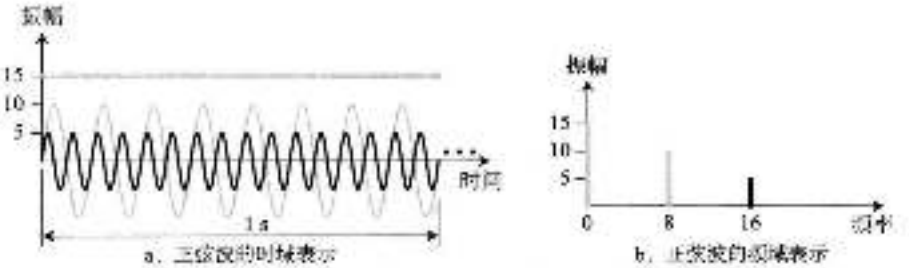


图 7-6　三个正弦波的时域和频域

复合信号

到目前为止，我们主要讨论的是简单类型正弦波。在日常生活中简单类型正弦波有很多应用，比如将能量从一个位置传送到另一个位置。但是，如果使用单一正弦波来传输电话的谈话，就毫无意义，因为它不会携带任何信息。我们只会听到嗡嗡声。我们需要通过发送复合信号来进行数据通信。**复合信号**（composite signal）由许多简单正弦波构成。

在 20 世纪早期，法国数学家傅里叶证明了任何复合信号都是由具有不同频率、相位和振幅的正弦波信号组合而成的。

复合信号可以是周期性的也可以是非周期性的。周期复合信号可以分解为一系列具有离散频率（取整数值 1，2，3 等的频率）的简单正弦波。非周期复合信号可以分解成具有连续频率（取实数值的频率）的无穷简单正弦波的组合。

带宽

复合信号分组包含的频率范围称为带宽（bandwidth）。带宽通常是两个数的差值。例如，如果一个复合信号分组含 1000 到 5000 的频率，它的带宽就是 5000−1000=4000Hz。

> 复合信号的带宽是信号最高频率与最低频率的差值。

图 7-7 说明了带宽的概念。图中描述了两个复合信号，一个是周期信号而另一个是非周期信号。周期信号的带宽分组包含许多为基频整数倍的离散频率，非周期信号的带宽有相同的范围，但频率是连续的。

数字信号

数据除了用模拟信号表示以外，还可以使用数字信号表示。例如，1 可以编码为正电平，0 可以编码为 0 电平。一个数字信号可以多于两个电平。在这种情况下，每个电平就可以发送多个位。图 7-8 表示了两个信号，一个信号有两个电平而另一个信号有四个电平。

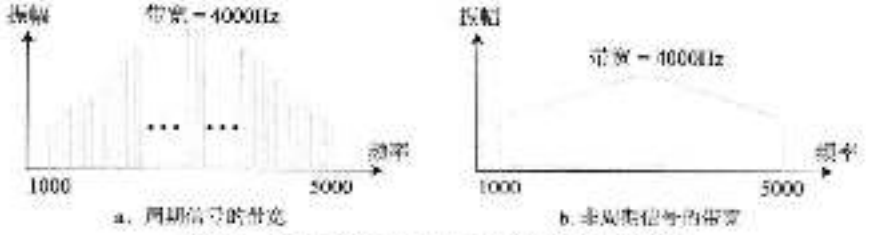


图 7-7　周期复合信号和非周期复合信号的带宽

在图 7-8a 中，每个电平可以发送 1 位，而图 7-8b 中每个电平可以发送 2 位。一般，如果信号有 L 个电平，则每个电平需要 $\log_2 L$ 个位。

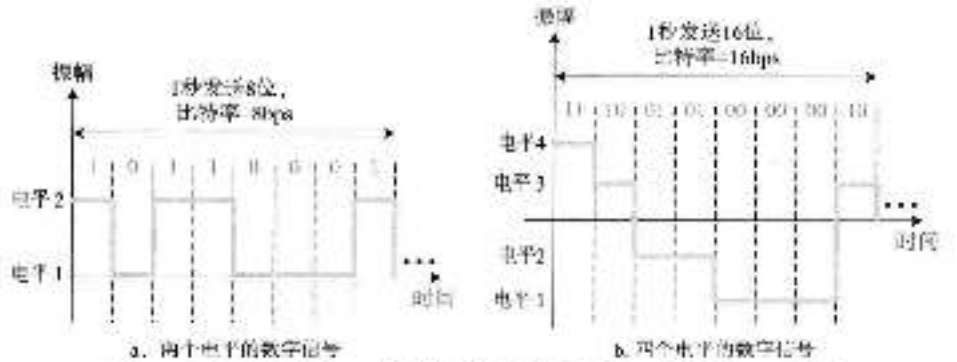


图 7-8　两个数字信号：一个信号有两个电平而另一个信号有四个电平

比特率

大多数数字信号是非周期性的，而这样周期和频率就不是适合的属性。另一个术语——比特率（而不是频率）用来描述数字信号。比特率（bit rate）是指 1 秒中发送的位数，以位每秒（bps）表示。

例 7.2　假定我们需要以每分钟 100 页的速率下载文本文档。所需通道的比特率是多少？一页平均 24 行，每一行平均 80 个字符。如果我们假定每个字符需要 8 位，比特率为

$$100 \times 24 \times 80 \times 8 = 1\,536\,000 \text{ bps} = 1.536 \text{ Mbps}$$

位长

我们已经讨论过模拟信号的波长概念：一个周期在传输介质上的距离。我们可以为数字信号定义相似的概念：位长。位长（bit length）是一个位在传输介质上的距离。

$$\text{位长} = \text{传播速度} \times \text{位持续时间}$$

数字信号是一种复合模拟信号

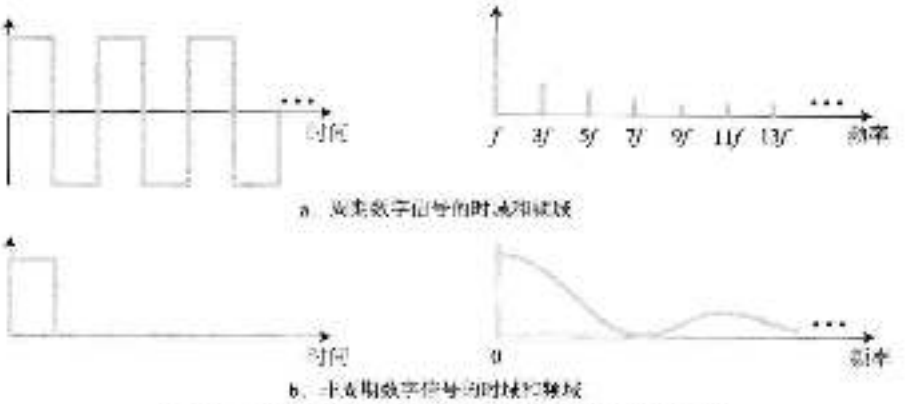
基于傅里叶分析，数字信号是复合模拟信号。正如你猜想的，带宽是无穷大的。当我们考虑一个数字信号时，可以直观地提出这个概念。在时域中，数字信号由连续的垂直和水平线段组成。时域中的垂直线表示为无穷大的频率（随时间突变）；时域中的水平线表示为 0 的频率（不随时间变化）。从频率 0 变为频率无穷大（以及反之）暗示两者间的所有频率都是域的一部分。

傅里叶分析可以用来分解数字信号。如果数字信号是周期性的（在数据通信中较少见），分解后的信号可以表示为无穷大带宽和离散频率的频域。如果数字信号是非周期性的，分解后的信号仍为无穷大的带宽，但频率是连续的。图 7-9 显示了周期数字信号和非周期数字信号以及它们的带宽。

数字信号的传输

前面的讨论声称数字信号（周期性或非周期性）是由零到无穷大范围内的频率组合成的复合模拟信号。下面所讨论的情况，我们假定是非周期数字信号，类似于我们在数据通信中碰到的信号。

基本问题是我们如何将数字信号从 A 点发送到 B 点？我们使用两种不同的方法传输数字信号：基带传输和宽带传输（使用调制）。



a. 周期数字信号的时域和频域

b. 非周期数字信号的时域和频域

图 7-9　周期数字信号和非周期数字信号的时域和频域

基带传输

基带传输是指通过通道发送数字信号，该数字信号不转换成模拟信号。图 7-10 表示了基带传输（baseband transmission）。

基带传输需要一个低通通道（low-pass channel），该通道的带宽是从 0 开始的。如果我们的带宽只组成一条通道的专用介质，就是这种情况。例如，连接两台计算机的串行的整个带宽就是一条单通道。另一个例子，我们可以连接多个计算机到一条总线上，但是不允许同一时刻超过两个站进行通信。如果有低通通道，我们就可以用它来进行基带传输。



图 7-10　基带传输

例 7.3　专用通道（介质的整个带宽用于一条单通道）的一个例子是 LAN。现在几乎每一个无线 LAN 都使用专用通道进行两个站点的相互通信。

宽带传输

宽带传输或调制是指把数字信号转换成模拟信号进行传输。调制允许我们使用带通通道（band-pass channel），即该通道的带宽不是从 0 开始的。这种通道比低通通道更有用。图 7-11 说明了带通通道。

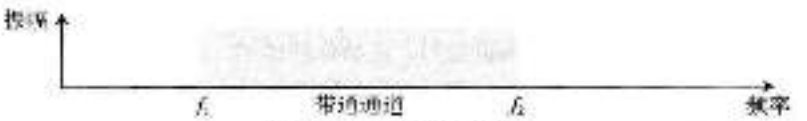


图 7-11　带通通道的带宽

注意低通通道可以理解为从 0 开始的较低频率的带通通道。图 7-12 显示了数字信号的调制过程。在该图中，数字信号转换成复合模拟信号。我们使用单频模拟信号（称为载波），已经改变的载波振幅看起来像数字信号。但是结果不是单频率信号，正如我们以后会看到的，它是复合信号。在接收端将接收到的模拟信号转换为数字信号，结果是被发送的信号的复制品。

例 7.4　使用调制的宽带传输的一个例子是通过电话用户线传输计算机数据，电话用户线连接住所到中心电话局。这些线在许多年前就安装了，用有限带宽（频率为 0～4Hz）承载语音（模拟

信号）。虽然这个通道能用低通通道，但是通常就看成是带通通道。一个原因是带宽太小（4kHz）以至于如果我们把它当作低通通道用于基带传输，那么最大的比特率就只有 8kbps（稍后解释）。解决方案是把它看做带通通道，从计算机把数字信号转换成模拟信号，然后发送模拟信号。我们要安装两个转换器用来在发送端将数字信号转换成模拟信号以及在接收端将模拟信号转换成数字信号。在这种情况下，转换器称为调制解调器（调制器/解调器），这个我们在第 5 章讨论过。

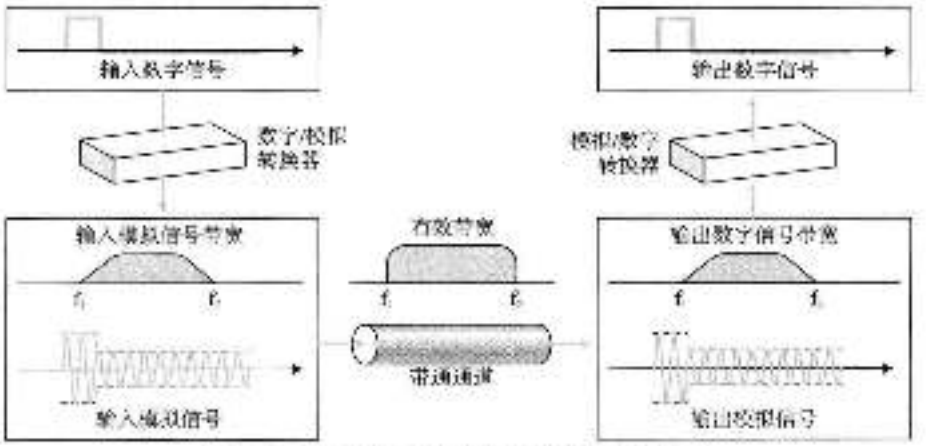


图 7-12　数字信号在带通通道传输的调制过程

例 7.5　第二个例子是数字蜂窝电话。为了更好地接收信号，数字蜂窝电话把声音的模拟信号转换成数字信号。尽管分配给公司用来提供数字蜂窝电话服务的带宽很高，但是我们仍然不能发送不经过转换的数字信号。原因是我们在主叫方和被叫方之间只有一条可用的带通通道。例如，如果有效带宽是 W，并且我们允许 1000 对用户同时交谈，这意味着有效带宽是 W/1000，只是整个带宽的一部分。我们需要在发送前把数字信号转换成复合模拟信号。

7.1.2　传输减损

信号通过介质进行传输，但是传输并非是完美无缺的。有缺陷的地方会导致信号减损。这意味着信号在介质的开始一端和结束一端是不相同的。发送的信号并非就是接收到的信号。通常会有三种类型的减损：衰减、失真和噪声。

衰减

衰减（attenuation）是指能量的损失。当某种简单或者复合的信号通过某种介质传输时，它会为了克服介质的阻抗而失去一些能量。这就是传输电信号的电缆会变热的原因，如果不热，过一段时间也会发热。信号中的一部分电能转换为热能。为了补偿这种损失，放大器被用来放大信号。图 7-13 说明了衰减和放大的效果。

为了说明信号损失或放大的强度，工程上使用了分贝的概念。分贝（decibel，dB）用来测量两个信号之间或者一个信号在不同的两个位置之间的相对强度。如果信号衰减了，则分贝为负值；如果信号放大了，则分贝为正值。

$$dB = 10 \log_{10}(P_2/P_1)$$

变量 P_1 和 P_2 分别是信号在位置 1 和位置 2 的功率。

例 7.6　假设信号通过一种传输介质传输后，它的功率降低了一半。这可以表示为 $P_2 = 0.5\ P_1$。这种情况下衰减（损失的能量）可以计算为：

$$10 \log_{10} P_2/P_1 = 10 \log_{10} (0.5\ P_1)\ /P_1 = 10 \log_{10} 0.5 = 10\ \times\ (-0.3) = -3\ dB$$

3dB 的衰减（-3dB）等价于功率损失了一半。

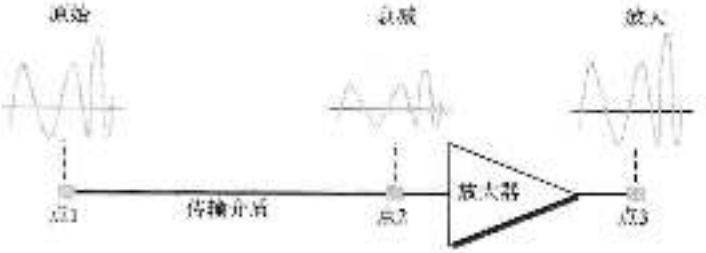


图 7-13　衰减和放大

失真

失真（distortion）是指信号改变形态或形状。失真产生在由不同频率组成的复合信号当中。每一种信号成分在通过介质时有自己的传播速度，所以到达最终目的的结点时有各自的延迟。如果延迟与周期时间不完全一致，那么延迟的差异就会产生相位的不同。换言之，接收方的信号成分与发送方的信号成分存在相位差异。因此复合信号的形状会不一样。图 7-14 说明了失真对复合信号的影响。

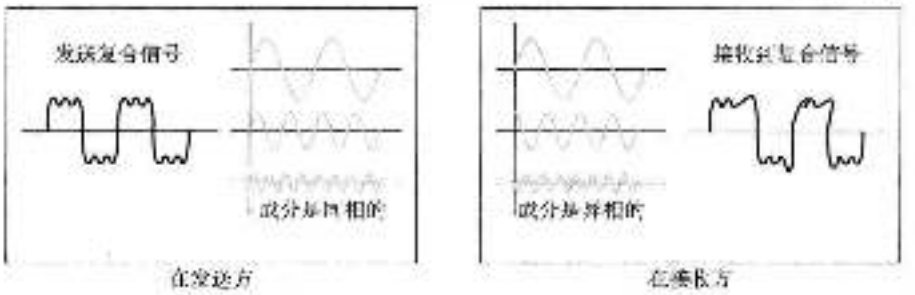


图 7-14　失真

噪声

噪声（noise）是衰减的另一个原因。有几种类型的噪声，如热噪声、感应噪声、串扰和脉冲噪声，都会损害信号。热噪声是电缆中的电子随机移动而产生的额外信号，而不是信号发送装置最初发送的。感应噪声的来源是电动机和设备。这些设备相当于发射天线，而传输介质成为接收天线。**串扰**（crosstalk）则是指两根电缆之间的相互影响。一根电缆作为发射天线而另一根电缆作为接收天线。脉冲噪声是一种尖峰信号（在非常短的时间内有很高能量的一种信号），来自电线、闪电等。图 7-15 说明了噪声对信号的影响。我们在第 5 章讨论了如何去检测和控制这些类型的差错。

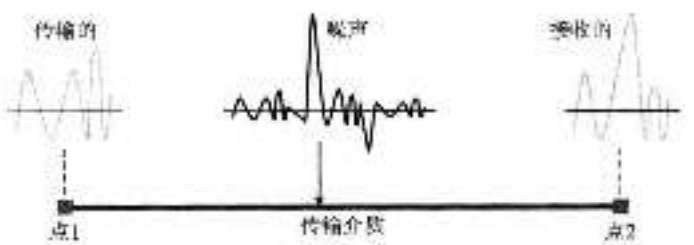


图 7-15　噪声

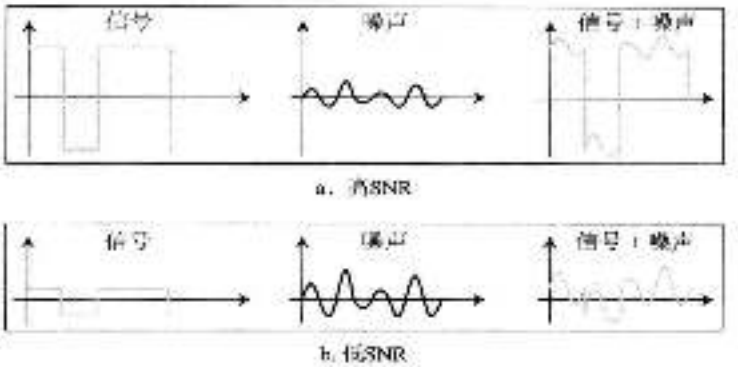
信噪比（SNR）

我们以后会看到，为了找到理论上的比特率限制，我们需要知道信号功率和噪声功率的比率。信噪比（signal-to-noise）的定义如下：

信噪比（SNR）= 平均信号功率/平均噪声功率

我们需要考虑平均信号功率和平均噪声功率，因为这些会随时间变化。图 7-16 说明了 SNR 的概念。高信噪比是指信号较少被噪声破坏；低信噪比是指信号较多被噪声破坏。因为 SNR 是两个功率的比率，所以一般以分贝为单位描述。SNR_{dB} 定义如下。

$$SNR_{dB} = 10 \log_{10} SNR$$



a. 高SNR

b. 低SNR

图 7-16　SNR 的两种情形：高 SNR 和低 SNR

7.1.3　数据速率限制

数据通信中一个非常重要的问题是：在一个通道中能够以多快的速率发送数据，即每秒钟的比特数。数据速率取决于三种因素：

1. 有效带宽；
2. 使用的信号电平数；
3. 通道的质量（噪声电平）。

有两个理论上的公式用来计算数据速率：一个由奈奎斯特（Nyquist）提出用于无噪声通道，另一个由香农（Shannon）提出用于噪声通道。

无噪声信道：奈奎斯特比特率

对于无噪声通道，奈奎斯比特率（Nyquist bit rate）公式定义了理论上的最大比特率：

$$比特率 = 2 \times B \times \log_2 L$$

公式中，B 是通道的带宽，L 是用于表示数据的信号电平数量，比特率是指每秒钟比特数。

根据这个公式，我们可以认为：给定一个带宽，就可以通过增加信号电平数获得任何想要的比特率。虽然这个思想在理论上是正确的，但是实际上有局限性。当我们增加信号电平数时，也增加了接收端的负担。如果信号的电平数仅仅为 2，接收方可以轻易区分 0 和 1。如果电平数为 64，接收方必须很复杂才能区分 64 个不同的电平。换言之，增加信号电平数减弱了系统的可靠性。

例 7.7　我们需要通过带宽为 20kHz 的无噪声通道发送 265kbps 。我们需要多少个信号电平数？使用奈奎斯特公式如下：

$$265\,000 = 2 \times 20\,000 \times \log_2 L \rightarrow \log_2 L = 6.625$$

$$L = 2^{6.625} = 98.7 \text{ 个电平}$$

因为结果不是 2 的幂，所以我们需要增加电平数或者减少比特率。如果我们有 128 个电平，则比特率为 280kbps。如果我们有 64 个电平，则比特率为 240kbps。

噪声通道：香农容量定理

实际上，不可能存在无噪声通道，通道总是有噪声的。1944 年，香农（Shannon）引进了一个公式，称为香农容量定理（Shannon capacity），能够确定噪声通道理论上的最高数据速率。

$$C = B \times \log_2 (1 + SNR)$$

公式中，B 是指通道的带宽，SNR 是信噪比，C 是指通道的传输容量（每秒比特数）。注意，香农公式中没有指出信号电平，这意味着无论使用多少个电平，都不可能获得比通道容量更高的数据速率。换言之，公式定义了通道的特性，而不是传输方式。容量定义了通道比特率的上限。

例 7.8　考虑到极端的噪声通道，其信噪比接近于 0。换句话说，噪声太强了以至于信号很弱。对于这个通道，它的容量 C 计算如下：

$$C = B \log_2(1 + \text{SNR}) = B \log_2(1 + 0) = B \log_2 1 = B \times 0 = 0$$

这就是说通道的容量为 0，与带宽无关。换言之，该通道不能接收到任何数据。

例 7.9　我们能够计算理论上一条常规电话线的最高比特率。通常情况下，电话线中用于数据通信的带宽为 3000Hz（300Hz～3300Hz）。信噪比通常为 3162。对于这一通道，它的容量计算为如下：

$$C = B \log_2(1 + \text{SNR}) = 3000 \log_2(1 + 3162) = 34\,881 \text{ bps}$$

这就是说电话线的最高比特率为 34 881kbps。如果要使数据发送速率比这更快，则需要增大线路的带宽或者提高信噪比。

使用两种限制条件

在实际应用中，两种方法都要使用，以确定所需要的带宽以及信号电平数。下面通过一个实例来说明这一点。

例 7.10　有一个 1MHz 带宽的通道。这个通道的信噪比为 63，合适的比特率以及信号电平数是多少？

解答

首先，使用香农公式确定上限。

$$C = B \log_2(1 + \text{SNR}) = 10^6 \log_2(1 + 63) = 10^6 \log_2 64 = 6 \text{ Mbps}$$

尽管香农公式计算的结果是 6Mbps，但这是上限。为了获得更好的性能，可以选择较低的值，例如 4Mbps。然后使用奈奎斯特公式计算信号电平的数量。

$$4 \text{ Mbps} = 2 \times 1 \text{ MHz} \times \log_2 L \quad \rightarrow \log_2 L = 2 \quad \rightarrow L = 4$$

香农容量定理给出数据速率的上限；奈奎斯特公式给出所需的信号电平数。

7.1.4　性能

直到目前为止，我们已经讨论了在网络上传输数据（信号）的工具以及如何传输数据。网络中的一个重要问题是网络的性能——网络有多么好？我们会在第 8 章更详细地讨论服务质量、网络性能的整体平衡。

带宽

衡量网络性能的一个特性就是带宽。但是，这个术语在两种不同情况下使用不同的衡量值：以赫兹衡量的带宽和以每秒位数衡量的带宽。

以赫兹衡量的带宽

我们已经讨论过这个概念。以赫兹衡量的带宽是复合信号分组包含的频率范围或者通道可以通过的频率范围。例如，我们可以说用户电话线的带宽是 4kHz。

以每秒位数衡量的带宽

带宽这个术语还可以指通道、链路或网络每秒能发送的位数。例如，我们可以说快速以太网（或者这种网络中的链路）的带宽最高为 100Mbps。这意味着这种网络能发送 100Mbps。

两者关系

以赫兹衡量的带宽和以每秒位数衡量的带宽之间有明显的关系。基本上，以赫兹衡量的带宽的增长意味着以每秒位数衡量的带宽的增长。它们的关系取决于使用的是基带传输还是宽带传输。

例 7.11　用于语音或者数据传输的用户线带宽是 4KHz。使用复杂的调制解调器把数字信号转

换成模拟信号，这种线路用于数据传输的带宽可以达到 56kbps。如果电话公司提高线路的质量，把带宽增加到 8KHz，用于数据传输的带宽就可以达到 128kbps。

吞吐量

吞吐量（throughput）用于衡量通过网络发送数据的快慢。尽管第一感觉上，以每秒位数衡量的带宽和吞吐量看来一样，但它们是不同的。一条链路可以有 Bbps 的带宽，但是我们只能通过这条链路发送 7bps，7 永远小于 B。换句话说，带宽是链路的潜在衡量值；而吞吐量是发送速度快慢的实际衡量值。例如，我们有一条带宽为 1Mbps 的链路，但是连接到链路末端的设备只能处理的带宽为 200kbps。这意味着我们不能通过这条链路发送高于 200kbps 吞吐量的数据。

想象一下，一条高速公路设计为从一点到另一点每分钟通行 1000 辆汽车。但是，如果道路上有拥塞，这个数字会降低到每分钟 100 辆汽车。带宽是每分钟 1000 辆汽车，而吞吐量是每分钟 100 辆汽车。

延迟

延迟（latency，或 delay）定义了第一个位从源开始发出到整个报文完全到达目标所经历的时间。我们在第 4 章讨论过分组的延迟。我们说一般有四种类型的延迟：**传播延迟**（propagation delay）、**传输延迟**（transmission delay）、**排队延迟**（queuing delay）和**处理延迟**（processing delay）。延迟或总的时延为：

$$\text{延迟} = \text{传播延迟} + \text{传输延迟} + \text{排队延迟} + \text{处理延迟}$$

带宽与延迟的乘积

带宽与延迟是链路的两个性能标准。但是，数据通信中非常重要的是带宽与延迟两者的乘积。让我们通过两个假想情况作为例子来详细阐述（见图 7-17）。

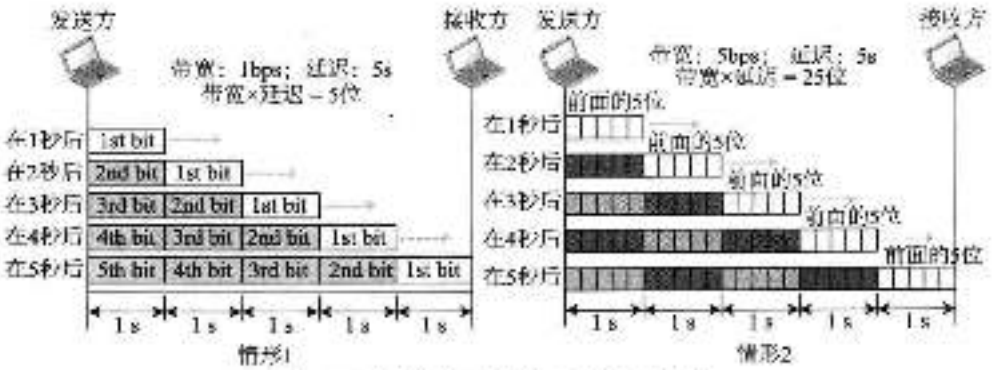


图 7-17　在情形 1 和情形 2 下位充满链路

- **情形 1**：假定我们有一条带宽为 1bps 的链路（显然不现实，但是利于说明目的）。还假定链路的延迟是 5s（也是不现实的）。我们想看这个情形中带宽延迟乘积意味着什么。从图 7-17 中可知，我们可以说带宽延迟乘积（1×5）是能充满链路的最大位数。链路上任何时刻都不可能多于 5 位。
- **情形 2**：现在假定带宽是 5bps，延迟是 5s。图 7-17 说明了链路上最多可以有 5×5=25 位。原因是每一秒钟链路上有 5 位，每位的持续时间是 0.2s。

上述两种情形说明带宽与延迟的乘积是能充满链路的位个数。如果我们需要以脉冲发送数据并在下一个脉冲前等待每个脉冲的确认，这个标准十分重要。为了使用链路的最大容量，我们需要脉冲长度是带宽与延迟乘积的 2 倍，需要充满全双工通道（两个方向）。发送方应该发送一个（2×带宽×延迟）位的数据脉冲。然后发送方在发送下一个脉冲前等待这个脉冲部分的接收方的确认。

数量 2×带宽×延迟是任何时刻转换的位数。

带宽延迟乘积定义了能充满线路的位数。

例 7.12　我们可以把两点间的线路看做管道。管道的横截面表示带宽，管道的长度表示延迟。我们可以说管道的容量定义了带宽延迟乘积，如图 7-18 所示。

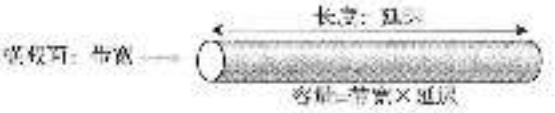


图 7-18　带宽延迟乘积的概念

抖动

另一个与延迟有关的性能问题是抖动（jitter）。我们可以大致认为抖动是这样一个问题，如果数据的不同分组碰到不同的延迟并且应用的在接收站使用的数据是时间敏感的（例如音频和视频数据）时候就会出现。例如，如果第一个分组的延迟是 20ms，第二个延迟是 45ms，第三个是 40ms，那么使用这些分组的实时应用就会遭遇抖动。我们会在第 8 章更详细地讨论抖动。

7.2　数字传输

计算机网络用于在网络结点之间传送信息。信息需要转换为数字信号或模拟信号进行传输。本节中我们讨论第一种选择，转换为数字信号。在下一节我们会讨论第二种选择，转换为模拟信号。

如果数据是数字的，我们将要使用**数字到数字转换**（digital-to-digital conversion）技术，这种技术将数字数据转换为数字信号。如果数据是模拟的，需要使用**模拟到数字转换**（analog-to-digital conversion）技术，这种技术将模拟信号转换为数字信号。

7.2.1　数字到数字转换

我们讨论过数据和信号。已经说过数据可以是数字的也可以是模拟的。也说过数据的信号可以是数字的也可以是模拟的。本节中，会看到如何使用数字信号表示数字数据。转换涉及三种技术：线路编码，块编码和扰动。线路编码总是需要的，块编码和扰动可能需要也可能不需要。

线路编码

线路编码（line coding）是将数字数据转换为数字信号的过程。假设文本、数字、图形图像、音频或者视频形式的数据在计算机内存中是以位序列形式保存的。线路编码将为序列转换为数字信号。在发送端，数字数据被编码为数字信号；在接收端，数字信号通过解码重新生成原数字数据。图 7-19 说明了这个过程。

尽管在文献中使用更多的分类，但是我们把公用线路编码机制分为三类，单极性、双极性和多电平。

在讨论每个分类之前，让我们先定义线路编码中使用的一些术语。定义 N 为比特率（每秒钟传送的位的数量）。定义 r 为信号电平每次变化时承载的位的数量。定义 S 为波特，即每秒钟信号电平变化的次数。我们通常对 S_{avg} 感兴趣，能用如下公式计算出信号变化的平均数。

$$S_{avg} = c \cdot N \cdot (1/r)$$

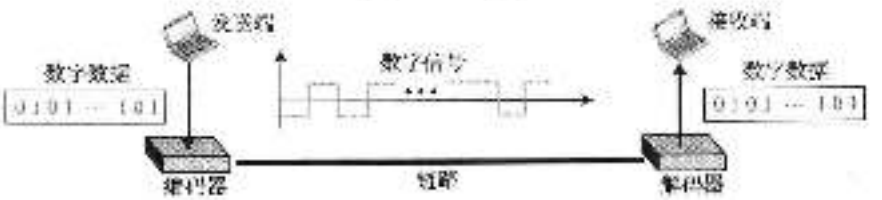


图 7-19　线路编码和解码

上式中，c 是一个因子，它与如何计算平均数有关。如果数据全由 1 组成或者全由 0 组成，当位改变时（从 0 变成 1 或从 1 变成 0）信号电平发生改变，那么 c 的值就为 0。通常情况下就是 1/2。

单极性方案

在单极性方案中，尽管电压电平可能保持在正电平与负电平两者间的零电平，但是电压电平在正电平和负电平间振荡。图 7-20 说明了这类中的几种方案。图中也说明了带宽（相对于频率的信号能量），但是频率是标准化的（fN）。

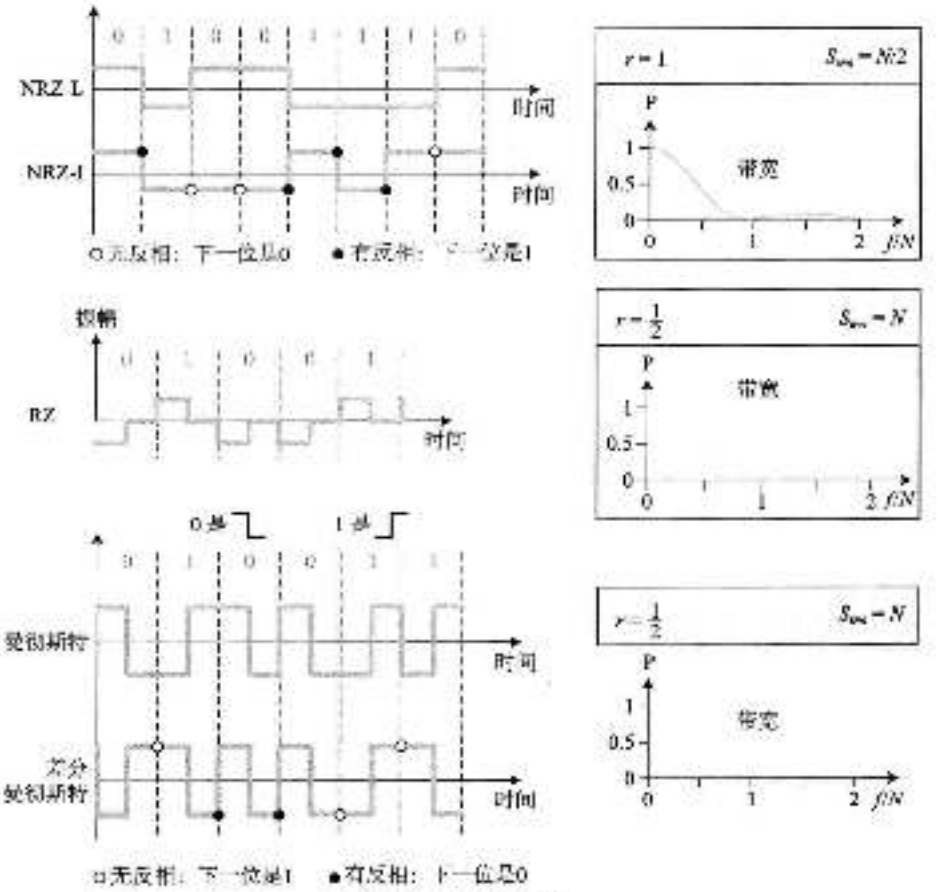


图 7-20　单极性方案

在不归零编码（non-return-to-zero，NRZ）方案中，信号的振幅使用两个电平（非零电平）。极性 NRZ 有两种常见形式：NRZ-I 和 NRZ-L。在 NRZ-L（NRZ-Level）中，电平等级决定了位值；在 NRZ-I（NRZ 反相编码，NRZ-Invert）中，信号电平是否反相转或跳变决定了位值。如果没有跳变，位值是 0；如果有跳变，位值是 1。如果有一个全 0 或全 1 的长序列，平均信号功率会发生偏移。但是，NRZ-L 中的偏移比 NRZ-I 更严重，前者是后者的 2 倍。接收端辨别位值就会很困难。而在 NRZ-I 中这个问题只发生在全 0 的长序列中。

当发送端和接收端始终不同步时，NRZ 编码的主要问题就发生了。接收方不知道一个位什么时候结束，下一个位什么时候开始。一种解决方法是归零编码（return-to-zero，RZ）方案，它使用三个值：正值、负值和零。在 RZ 中，信号不在两个位之间变化而是在位中变化。RZ 编码的主要缺点是它需要两个信号变化来编码一个位，因此占用更大的带宽。

RZ 的思想（位中间跳变）和 NRZ-L 的思想共同组合成了曼彻斯特（Manchester）编码方案。在曼彻斯特编码中，位的持续时间被二等分。在前部分电平保持一个值，后半部分电平变成另一个值。位中间的跳变提供了同步。另一方面，差分曼彻斯特（Differential Manchester）组合了 RZ 和 NRZ-I 的思想。在位中间有一个跳变，但是位值在位开始时确定。如果下一个位是 0，就有一个跳变；如果下一个位是 1，则没有跳变。

双极性方案

在双极（bipolar）编码（也称为多电平二进制，multilever binary）中，有三个电平：正电平、负电平和零。一个数据元素的电平是 0，而另一个数据元素的电平在正电平和负电平间交替。图 7-21 显示了双极编码的两种编码：AMI 和伪三元编码。一个常用的双极编码方案称为双极交替传号反转（alternate mark inversion，AMI）。在术语交替传号反转中，传号这个词来源于电报，它的值是 1。所以 AMI 表示的含义是交替的 1 的反转。中间的 0 电平表示二进制数 0，而二进制 1 由交替正负电平表示。AMI 编码的一个变型是**伪三元编码**（pseudoternary），位 1 被编码成 0 电平，而位 0 编码成交替正负电平。

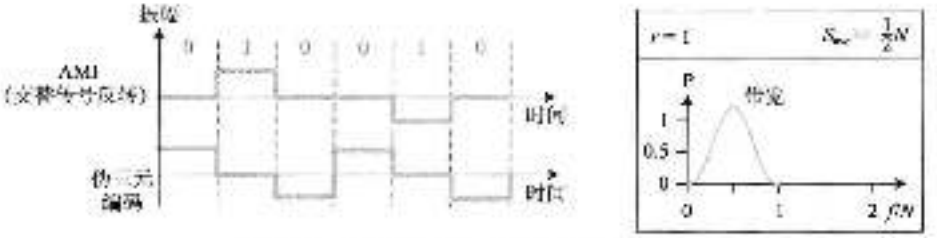


图 7-21　双极性方案：AMI 和伪三元编码

多电平方案

增加数据速度或者降低所需带宽的需求导致了很多方案的产生。目标是通过把 m 个数据元素的模式编码成 n 个信号元素的模式时增加每波特的位数。我们只有两种数据元素（0 和 1），这表示有 m 个数据元素的组可以产生 2^m 个数据模式组合。编码设计者以 $mBnL$ 区分这些编码类型。这里 m 是二进制模式的长度，B 表示二进制数据，n 是信号模式的长度，L 是信号中的电平数。一般用字母 L 替换：例如，$L=2$ 时用 B 表示（二元）、$L=3$ 时用 T 表示（三元）、$L=4$ 时用 Q 表示（四元）。注意前两个字母定义了数据模式，后两个字母定义了信号模式。尽管多电平有很多方案，我们在图 7-22 只显示了其中的两种。

两个二元、一个四元（two binary,one quaternary，2B1Q）方案使用长度为 2 的 2 位数据模式编码成一个 4 电平信号元素。注意，在 2B1Q 中是把 11，3，−1，−3 编码为模式 00,01,10,11。然而，如果先前的电平是负的，信号的值就要被反转为（−1，3，−1，+3）。这是图中模式 01 的情形，因为先前的信号电平是负的，由+3 反转为-3。一个很有趣的方案是八个二元、六个三元（eight binary,six ternary，8B6T）。正如我们在第 5 章看到的，这个方案用于 100BASE-4T 电缆。这个方案的思想是把 8 位模式编码成 6 个信号元素模式，每个信号有 3 个电平（三元的）。我们可以有 $2^8=256$ 个不同的数据模式和 $3^6=729$ 个不同的信号模式。有 $729-256=473$ 个冗余信号用来提供同步和差错检测。注意，反转第二个方案是为了避免一个 DC 成分，非零电压平均值。

块编码

我们需要某种冗余以确保同步，并提供一些内在的差错检测。块编码可以提供这种冗余并提高线路编码的性能。一般，块编码（block coding）把 m 位的块变成 n 位的块，其中 n 大于 m。块编码称为 mB/nB 编码技术。块编码中的斜线（例如，4B/5B）用来区别多电平编码（例如，8B6T），

多电平编码中没有斜线。块编码一般涉及三个步骤：分组、置换和合并。在分组步骤中，位序列被划分成许多个 m 位的组。例如，在 4B/5B 编码中，原始的位序列划分成 4 位的组。块编码的核心是置换步骤。在这个步骤，用 n 位的组置换 m 位的组。例如，在 4B/5B 编码中，用 5 位组置换 4 位组。最后的 n 位组组合形成流。新的流比原始的流有更多的位，图 7-23 说明了这个过程。

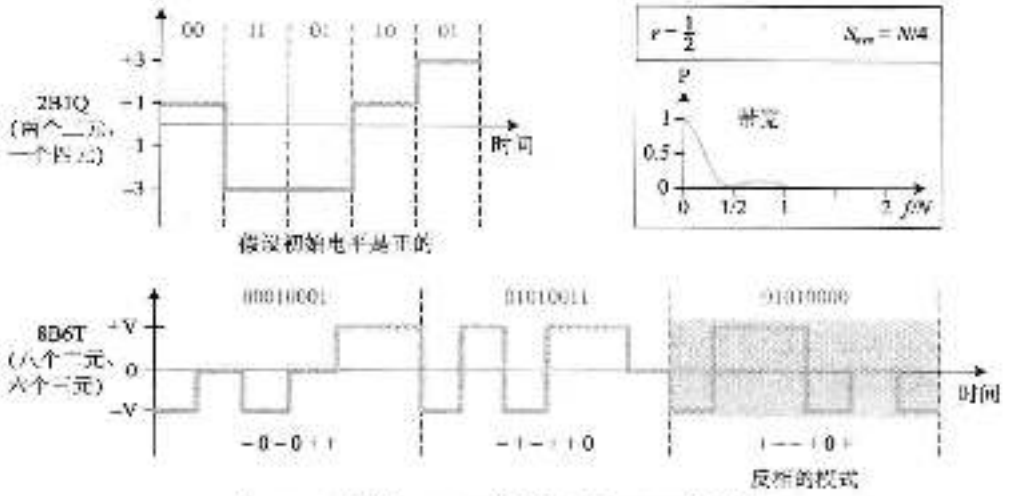


图 7-22　多电平：2B1Q 编码方案和 8B6T 编码方案

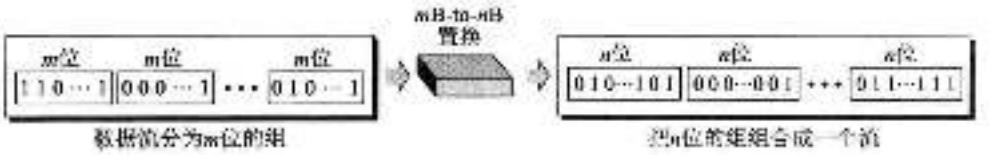


图 7-23　块编码概念

4B/5B 编码

4 个二元/5 个二元（four binary/five binary，4B/5B）编码方案设计出来与线路编码结合使用，比如 NRZ-I，这种方案中全 0 的长序列可以导致接收方的时钟失去同步。一种解决方法是在使用 NRZ-I 编码前改变位流，以至于不会是一个个 0 的位流。4B/5B 方案实现了这个目的。经过块编码的流不会有多于三个连续的 0。在接收端，NRZ-I 编码后的数字信号线解码成位流，然后去掉冗余。图 7-24 说明了这个思想。

在 4B/5B 编码中，将 4 位输入置换为 5 位输出包含不超过一位的前导 0（左边的位）和不超过两位的后缀 0（右边的位）。因此当不同的组组合形成新的序列时，最多只会有三个连续的 0。

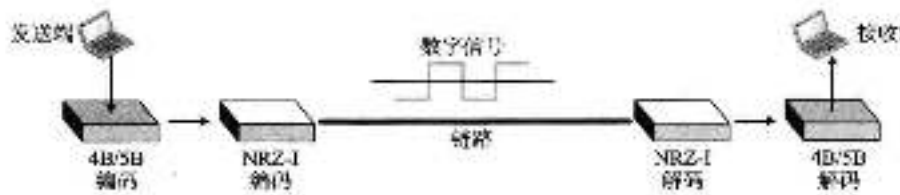


图 7-24　使用块编码 4B/5B 和线路编码 NRZ-I 的组合方案

8B/10B 编码

8 个二元/10 个二元（eight binary/ten binary，8B/10B）编码类似于 4B/5B 编码方案，除了它是

8 位数据组被置换成 10 位编码外。它提供了比 4B/5B 更强的差错检测能力。8B/10B 块编码实际上是 5B/6B 和 3B/4B 编码的组合，如图 7-25 所示。

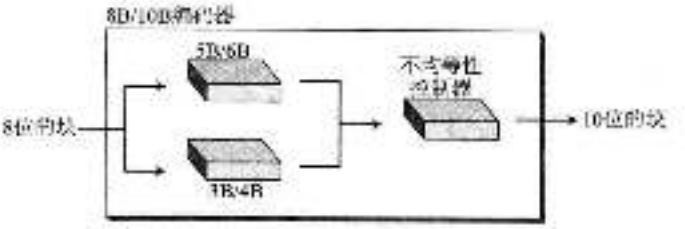


图 7-25　8B/10B 编码

10 位块中的高 5 位被送入 5B/6B 编码器；低 5 位被送入 3B/4B 编码器。这样做为了简化映射表。为了防止连续 0 或 1 的长串，编码使用不均等性控制器观察 0 比 1 多（或 1 比 0 多）。如果当前块中的位产生了不均等性，而该不均等性对上一块的不均等性有影响（任一方向），那么当前块的每一位取反（0 变成 1 而 1 变成 0）。这种编码有 $2^{10} - 2^8 = 768$ 个冗余组，可以用于不均等性校验和差错检测。一般来说，这种技术优于 4B/5B 是因为它有更好的内置差错检测能力和同步能力。

扰动

双相编码方案适用于 LAN 中站点间的专用链路，而并不适用于长距离通信（由于带宽需求）。由于有 DC 成分（信号频率为 0）。块编码和 NRZ 线路编码的组合也不适用于长距离通信。另一方面，双极 AMI 编码有窄带宽而且不会产生 DC 成分。但是，连续 0 的长序列会不同步。如果能找到避免初始流中连续 0 的长序列的方法，那么双极 AMI 编码可以用于长距离通信。我们正寻找一种不会增长位数且提供同步的技术。换句话说，我们正寻找一种把长的 0 电平脉冲置换成其他电平组合来提供同步的解决方法。一种方法称为扰动（scrambling）。如图 7-26 所示，我们修改部分 AMI 规则引入了扰动。注意，相对于块编码，扰动与编码同时完成。系统需要基于定义好的扰动规则插入所需脉冲。两种适用脉冲技术是 B8ZS 和 HDB3。



图 7-26　使用扰动的 AMI

B8ZS 编码

8-零置换的双极编码方案（bipolar with 8-zero substitution，B8ZS）通常用于北美地区。在这种技术中，8 个连续的 0 电平会被替换成序列 **000VB0VB**。这里的 V 表示违反；这个非零电平违反了 AMI 编码规则（前一极的相反极性）。序列中的 B 表示双极性，表示与 AMI 规则一致的非零电平。如图 7-27 所示，有两种情形。

注意，这里的扰动不会改变比特率。而且这个技术平衡了正电平和负电平（两个正电平和两个负电平），这意味着维持了 DC 平衡。注意，置换可能会改变 1 的极性，因为置换后 AMI 需要遵守它的规则。

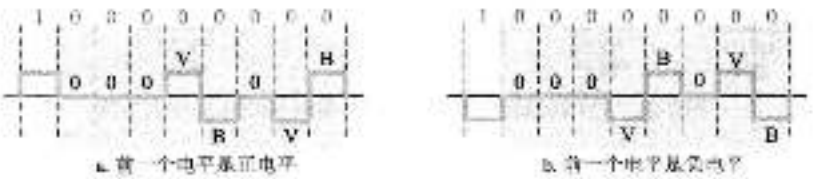


图 7-27　B8ZS 扰动技术的两种情形

还要多到一点。这里的字母 V（违反）或 B（双极）是相对的。V 表示与前一个非零脉冲极性相同的极性；B 表示与前一个非零脉冲极性相反的极性。

HDB3 编码

高密度双极 3-零方案（high-density bipolar 3-zero，HDB3）通常用于北美以外的地区。这种技术比 B8ZS 方案更加保守，四个连续的零电平被置换成 000V 或 B00V。有两种不同置换的原因是为了维持每次置换后非零脉冲数为偶数。这两个规则可以定义如下：如果最后一次置换后的非零脉冲数为奇数，则置换模式是 000V，这样使得非零脉冲总数为偶数。如果最后一次置换后的非零脉冲数为偶数，则置换模式是 B00V，这样使得非零脉冲总数为偶数。图 7-28 给出了一个实例。

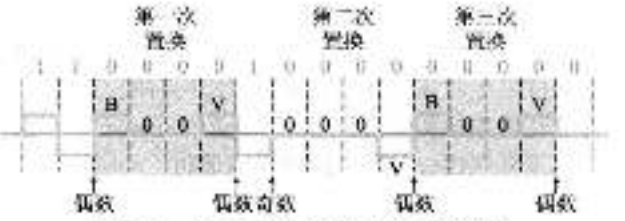


图 7-28　HDB3 抗攻技术中的不同情形

这里还需要提到几点。首先，在第一次置换之前，非零脉冲总数为偶数，所以第一次置换是 B00V。这次置换后，因为每次置换后必须遵循 AMI 方案自己的规则，所以位 1 的极性改变。在这个位后，因为在最后一次置换后只有一个非零脉冲（奇数），所以我们需要另一次置换，这次置换是 000V。第三次置换是 B00V，因为在第二次置换后（偶数）没有非零脉冲。

7.2.2　模拟到数字转换

7.2.1 节中描述的技术用于把数字数据转换成数字信号。然而，有时我们只有模拟信号，诸如麦克风或摄影机产生的信号。因为数字信号比模拟信号受噪声影响较小，现在的趋势是把模拟信号转换成数字信号。本章将描述两种技术，脉冲码调制和 delta 调制。在数字数据产生（数字化）后，我们使用前面章节描述的一种技术把数字数据转换成数字信号。

脉冲码调制（PCM）

把模拟信号转换成数字信号（数字化）最适用的技术称为脉冲码调制（pulse code modulation，PCM）。如图 7-29 所示，PCM 编码器有三个过程。

三个过程如下：

1. 对模拟信号进行采样；
2. 对采样后的信号进行量化；
3. 量化后的值编码成位流。

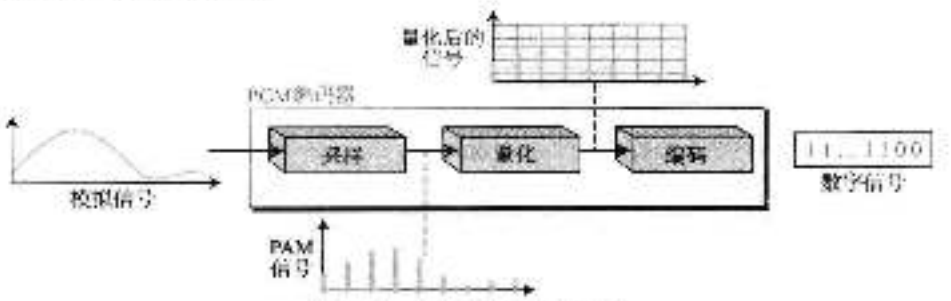


图 7-29　PCM 编码器的组成

采样

PCM 的第一个步骤是采样。每隔 T_s 秒对模拟信号进行采样，这里 T_s 秒是样本间隔或周期。采样间隔的倒数称为**采样率**（sampling rate）或**采样频率**（sampling frequency），定义为 f_s。这里 f_s =

$1/T_s$。采样方法有三种：理想采样、自然采样和方顶采样，如图 7-30 所示。

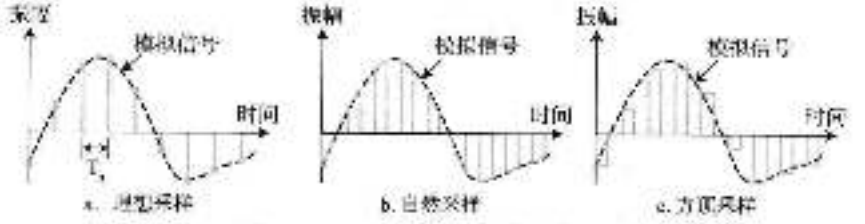


图 7-30　PCM 的三种不同采样方法

理想采样对来自模拟信号的脉冲进行采样。这是理想采样方法，不容易实现。在自然采样中，当采样发生时高速开关只开启很短的时间。结果是样本序列保持了模拟信号的形状。最常用的采样方法称为**采样和保持技术**（sample and hold），然而，它通过使用电路产生方顶样本。

采样过程有时称为**脉冲振幅调制**（pulse amplitude modulation，PAM）。但是我们需要记住，结果仍然是不完整的模拟信号。

采样速率　一个重要的考虑是采样速率或采样频率。对 T_s 的限制是什么？这个问题由奈奎斯特给出完善的回答。根据**奈奎斯特定理**（Nyquist theorem），为了再生原始模拟信号，一个必要条件是采样速率（sampling rate）至少是原始信号中最高频率的两倍。

我们需要在这里详细说明这个定理。首先，只有信号带宽是有限时我们才能对它进行采样。换言之，我们无法对无限带宽的信号进行采样。其次，采样速率必须至少是信号最高频率而不是带宽的两倍。如果模拟信号是低通的，带宽和最高频率是相同的值。如果模拟信号是带通的，带宽值小于最高频率值。图 7-31 说明了这两种信号的采样速率。

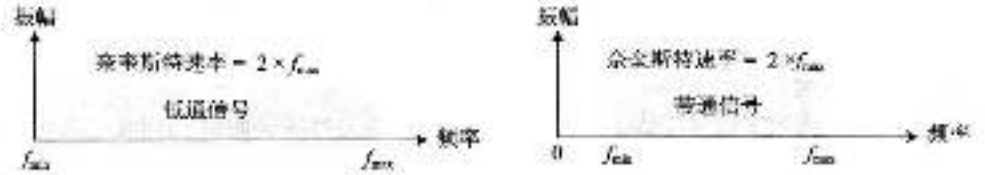


图 7-31　低通和带通信号的奈奎斯特采样速率

量化

采样的结果是一系列振幅在信号最大振幅和信号最小振幅之间的脉冲。振幅集可能是无限个介于最大振幅和最小振幅之间的非整数值。这些值不能用于编码过程。以下是量化的步骤：

1. 我们假设原始模拟信号有介于 V_{max} 和 V_{min} 之间的振幅。
2. 我们把范围分成 L 个区间，每个区间高度为Δ (delta)。

$$\Delta = (V_{max} - V_{min}) / L$$

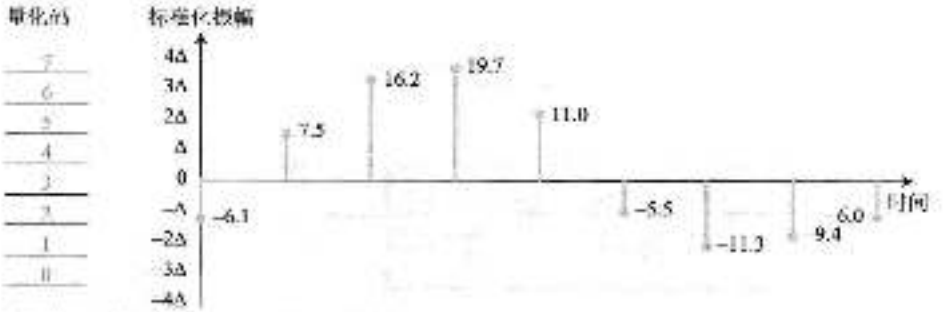
3. 我们给每个区间的中点分配 0 到 L-1 的量化值。
4. 样本振幅值近似于量化值。

举一个简单的例子，假设我们有一个采样信号，并且样本振幅在–20V 到 20V 范围内。我们决定有 8 个等级(L = 8)。这意味着Δ= 5V。图 7-32 说明了这个例子。

我们只展示了使用理想采样的 9 个样本（为简单起见）。图中每个样本顶端的值给出了实际振幅。在表中，第一行是每个样本（实际振幅/A）标准化后的值。量化进程从每个区间的中央选择量化值。这表示标准化量化值（第二行）不同于标准化振幅。两者之差称为标准化误差（第三行）。第四行是基于图左边量化等级的每个样本的量化码。编码码字（第五行）是转换的最后结果。

量化等级　在前面的例子中，我们给出了 8 个量化等级。L（等级数）的选择取决于模拟信号

振幅的范围以及我们需要恢复信号的准确程度。如果信号的振幅只在两个值间波动，我们只需要两个电平等级；如果信号像语音一样有多个等级，我们需要更多的量化等级。在音频数字化中，L 通常为 256；在视频中，L 通常是几千。如果信号的振动变化很大，较低的 L 值就会增加量化误差。



标准化PAM值	−1.22	1.50	3.24	3.94	2.20	−1.10	−2.26	−1.88	−1.20
标准化量化值	−1.50	1.50	3.50	3.50	2.50	−1.50	−2.50	−1.50	−1.50
标准化错误	−0.28	0	−0.26	−0.44	+0.30	−0.40	−0.24	+0.38	−0.30
量化码	2	5	7	7	6	2	1	2	2
编码码字	010	101	111	111	110	010	001	010	010

图 7-32　采样信号的量化和编码

量化误差　一个重要的问题是量化过程中产生的误差。量化是一个近似的过程。量化器的输入值是实际值；输出值是近似值。每个输出值为每个区间的中点值。如果输入值也是区间的中点值，则没有量化误差；否则就有误差。在前面的例子中，第二个例子的标准化振幅是 3.24，但是标准化量化值是 3.50。这表示误差为−0.26。任一样本的误差值都小于 Δ/2。也就是说，$-\Delta/2 \leq$ 误差 $\leq \Delta/2$。

量化误差改变了信号的信噪比，根据香农定理，这反过来就减少了上限容量。

可以证明量化误差（quantization error）对信号 SNR_{dB} 的影响取决于量化等级 L 或每个样本位数 n_b，如下面的公式所示。

$$\mathbf{SNR_{dB} = 6.02}n_b \mathbf{+ 1.76\ dB}$$

编码

PCM 中的最后一个步骤是编码。在每个样本量化并且每个样本的位数确定后，每个样本可以转换成 n_b 位的码字。图 7-32 中编码后的码字显示在最后一行。2 的量化码被编码为 010；5 被编码为 101 等。注意，每个样本的位数取决于量化等级数。如果量化等级数为 L，则位数就是 $n_b=\log_2 L$。在我们的例子中，L 是 8，所以 n_b 是 3。比特率可以通过如下公式获得：

$$\text{比特率} = \text{样本速率} \times \text{每个样本位数} = f_s \times n_b$$

例 7.13　要将人的话音数字化。假设每个样本有 8 位，那么比特率是多少？

解答

人的语音通常包含 0 到 4000Hz 的频率。所以样本速率和比特率可以由下式计算：

$$\text{样本速率} = 4000 \times 2 = 8000\ \text{样本/秒}$$

$$\text{比特率} = 8000 \times 8 = 64\,000\ \text{bps} = 64\ \text{kbps}$$

原始信号恢复

原始信号的恢复需要 PCM 解码器。解码器先使用电路把码字转换成脉冲，这个脉冲在下个脉冲前保持振幅。当阶梯信号完成后，它经过一个低通过滤器把阶梯信号平滑成模拟信号。过滤器有

与发送端原始信号一样的截断频率。如果信号是以奈奎斯特定理采样速率采样，并且有足够的量化等级，那么就可以重新生成原始信号。注意，通过使用放大器可以得到原来信号的最大值和最小值。图 7-33 说明了简化的过程。

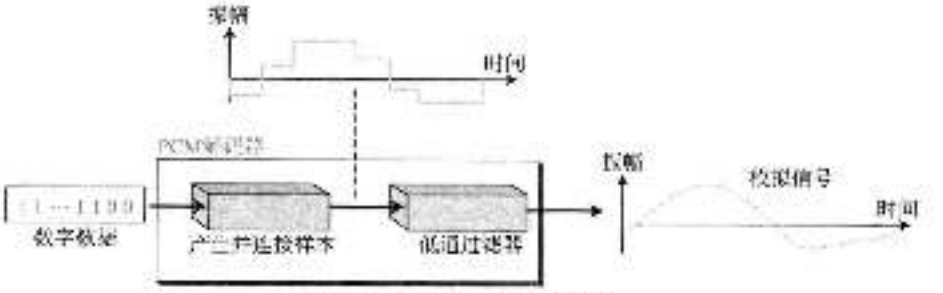


图 7-33　PCM 解码器的组成

PCM 带宽

可以证明数字信号的最小带宽可由下式表示。

$$B_{\min} = n_b \times B_{\text{analog}}$$

这表示数字信号的最小带宽是模拟信号的带宽的 n_b 倍。这就是数字化的代价。

delta 调制（DM）

PCM 是一种非常复杂的技术。已经开发出其他技术用来减少 PCM 的复杂性。最简单的一种是 delta 调制（delta modulation，DM）。PCM 得到每个样本的信号振幅值；DM 从前一个样本中得到变化。图 7-34 说明了这个过程。注意，这里没有任何码字，位一个接一个被发送。

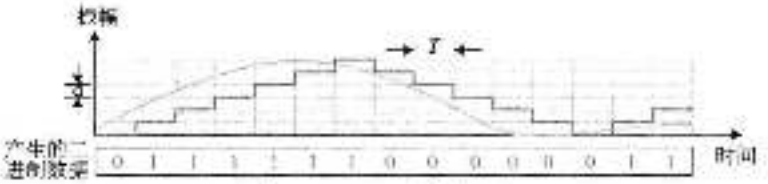


图 7-34　delta 调制过程

7.3　模拟传输

虽然数字传输有很多好处，但是需要低通通道。如果是带通通道，则只能选择模拟传输。将数字数据转换为带通模拟信号习惯上称为数字到模拟转换。将低通模拟信号转换为带通信号习惯上称为模拟到模拟转换。本节中，我们讨论这两种转换。

7.3.1　数字到模拟转换

数字到模拟转换（digital-to-analog conversion）是指根据数字数据中的信息而改变模拟信号的某种特性的过程。图 7-35 说明了数字信息、数字到模拟调制过程和最终模拟信号之间的关系。

如前面所述，一个正弦波可通过三个特性定义：振幅、频率和相位。当我们改变其中任意一个，就产生了波的另一种形式。所以，通过改变简单正弦波的某一特性，就可以用来表示数字数据。波的三个特性中的任意一个都可以用这种方式改变，从而使我们至少有三种机制将数字数据调制成模拟信号：**幅移键控**（amplitude shift keying，ASK）、**频移键控**（frequency shift keying，FSK）和**相移键控**（phase shift keying，PSK）。另外，还有第四种（更好的）机制，这种机制将振幅和相位的变化结合起来，称为**正交振幅调制**（quadrature amplitude modulation，QAM）。其中正交振幅调制是效率最高的，也是目前调制解调器中普遍采用的机制。



图 7-35　数字到模拟转换

幅移键控

在幅移键控方式中，通过改变载波信号的振幅来生成信号元素。只有振幅变化而频率和相位都保持不变。

二进制 ASK（BASK）

ASK 通常只使用两个电平。这称为二进制幅移键控或开关键控（on-off keying，OOK）。一个信号电平的振幅峰值为 0；另一个和载波频率振幅一样。图 7-36 给出了二进制 ASK 的概念图。图 7-36 也显示了 ASK 的带宽。虽然载波信号是一个简单正弦波，但调制处理后产生一个非周期性复合信号。正如前面讨论的，这个信号有一个连续频谱。正如我们期望的，带宽和信号速率（波特率）成正比。然而，通常涉及另一个因子称为 d，它取决于调制和过滤过程。d 的值在 0 和 1 之间。这意味着带宽能由如下表示，其中 S 是信号速率而 B 是带宽。

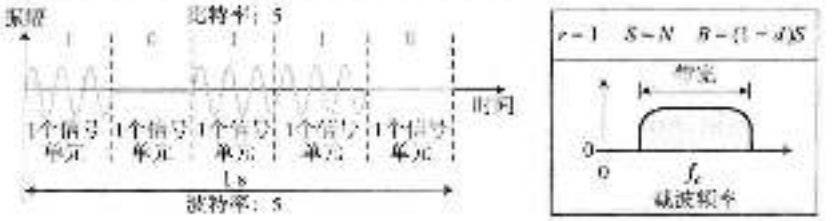


图 7-36　二进制幅移键控

公式中，$B = (1+d)S$，说明所需带宽的最小值是 S，最大值是 $2S$。最重要的一点是带宽的位置。载波频率 f_c 位于带宽的中间。这意味着如果有可用的带通通道，就可以选择 f_c 使得调制后的带宽可以占用该带宽。实际上这是数字到模拟转换的最重要优点，我们可以改变带宽结果来匹配可用带宽。

多电平 ASK

上面的讨论只使用两个振幅电平。也可以有多于两个电平，比如信号可以使用 4 个、8 个、16 个或更多的振幅电平，对应每次使用 2、3、4 或更多位来调制数据。在这些情况中，$r=2$，$r=3$，$r=4$，等等。这不是用单纯的 ASK 实现，而是使用 QAM（后面会见到）实现。

频移键控

在频移键控方式中，通过改变载波信号的频率来表示数据。调制后信号的频率在一个信号元素持续期间是不变的，但是如果数据元素改变则下一个信号元素也会改变，而所有元素振幅峰值和相位保持不变。

二进制 FSK（BFSK）

理解二进制 FSK（或 BFSK）的一个方法是考虑两个载波频率。在图 7-37 中，已经选择了两个载波频率 f_1 和 f_2。如果数据元素是 0 则使用第一个载波，如果数据元素是 1 则使用第二个载波。但是，注意这只是为了演示而不是实际的例子。通常这两个载波频率都很高，而且它们的差很小。

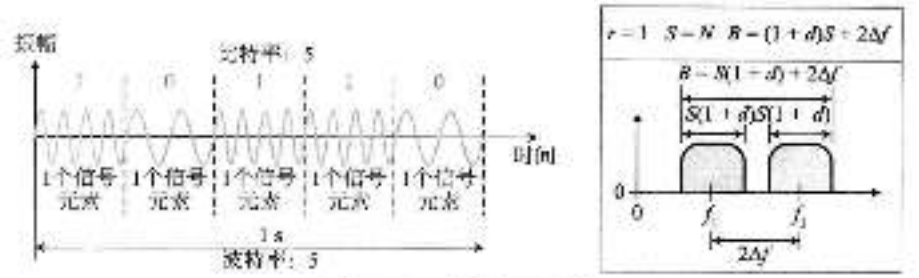


图 7-37　二进制频移键控

如图 7-37 所示，一个带宽中点的频率是 f_1，而另一个带宽中点的频率是 f_2。f_1 和 f_2 离这两个带的中点距离都是 Δf，则两个频率之差是 $2\Delta f$。

图 7-37 也给出了 FSK 的带宽。两个载波信号还是简单正弦波，但是调制产生一个具有连续频谱的非周期复合信号。我们可以把 FSK 看做是两个 ASK 信号，每一个都有自己的载波频率（f_1 或 f_2）。如果两个频率的差为 $2\Delta f$，那么所需带宽为 $B = (1 + d) \cdot S + 2\Delta f$。

$2\Delta f$ 的最小值应该是什么？在图 7-37 中，我们已选择了一个大于 $(1+d)S$ 的值。对于正确的调制解调操作，显然最小值至少应该是 S。

多电平 FSK

在 FSK 方式中，多电平 FSK（MFSK）是常见的。我们可以使用多于两个频率。例如，可以使用 4 个不同的频率：f_1、f_2、f_3 和 f_4 每次发送两个位。而每次发送 3 个位时，可以使用 8 个频率，以此类推。但是，需要记住相邻频率需要相隔 $2\Delta f$。对于正确的调制解调操作，显然 $2\Delta f$ 的最小值需要是 S。我们可以得出 $d=0$ 的带宽是：

$$B = (1 + d) \cdot S + (L - 1)\, 2\Delta f \quad \rightarrow B = L \cdot S$$

相移键控

在相移键控中，通过改变载波的相位来表示两个或更多个不同的信号元素。当相位改变时，峰值振幅和频率保持不变。现在，PSK 比 ASK 或 FSK 更通用。我们很快会看到 QAM（组合了 ASK 和 PSK），在数字到模拟转换中占主导的方法。

二进制 PSK（BPSK）

最简单的 PSK 是 BPSK，它只用两个相位元素，一个相位是 0°，另一个相位是 180°。图 7-38 给出了 PSK 的概念描述。二进制 PSK 和二进制 ASK 一样简单，但是 PSK 比起 ASK 有一个大的优点，即不易受噪声影响。在 ASK 中，位检测的标准是信号振幅；在 PSK 中则是相位。噪声改变振幅比噪声改变相位容易得多。PSK 优于 FSK 是因为我们不需要两个载波信号。

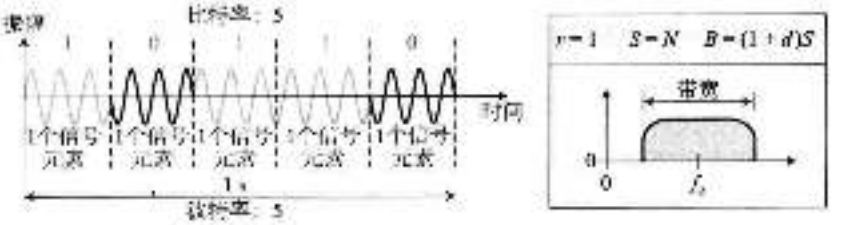


图 7-38　二进制相移键控

图 7-38 还说明了 BPSK 的带宽。这个带宽和二进制 ASK 的一样，但比 BFSK 的少。没有浪费带宽来分离成两个载波信号。

正交 PSK（QPSK）

BPSK 的简单性促使设计者在每个信号元素中一次使用 2 位，因此减少了波特率而最终减少了

所需带宽。这个方案称为正交 PSK 或 QPSK，因为它使用两个独立的 BPSK 调制：一个是同相的，另一个是正交的（异相）。进入的 2 个位先经过串行到并行的转换，它发送一个位到调制器，发送下一个位给另一个调制器。如果进入信号中的每个位的持续时间是 T，发送相应 BPSK 信号的每个位持续时间就是 $2T$。这意味着每个 BPSK 信号中位的频率是原始信号的一半。

星座图

星座图（constellation diagram）可以帮助我们定义信号元素的振幅和相位，尤其是使用两个载波（一个同相，而另一个正交）时。当处理多电平 ASK、PSK 或 QAM（见下一节）时，星座图很有用。在星座图中，一个信号元素类型用一个点表示。它携带的位或位集合一般写在它的旁边。

星座图有两根轴。水平 X 轴与同相载波相关；垂直 Y 轴与正交载波相关。针对图中每个点，可以推断出 4 条信息。点在 X 轴的投影定义了同相成分的峰值振幅。点在 Y 轴的投影定义了正交成分的峰值振幅。点到原点的连线（向量）长度是该信号元素的峰值振幅（X 成分和 Y 成分的组合）。连线与 X 轴的夹角是信号元素的相位。所有所需信息都可以在星座图中轻易得到。图 7-39 表示了三个星座图的概念，分别是 BASK 信号、BPSK 信号和 QPSK 信号。

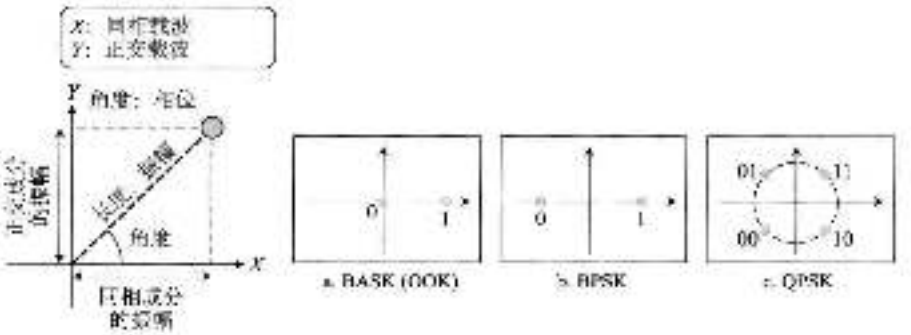


图 7-39　星座图概念

正交振幅调制

PSK 受到设备辨别相位细小差别能力的限制。这个原因限制了其潜在的比特率。到目前为止，我们每次只改变正弦波三种特性中的一种特性，那么改变两种特性会怎么样呢？为什么不将 ASK 与 PSK 结合在一起呢？即使用两个载波，一个同相而另一个正交，而且每个载波都用不同的振幅电平，这就是正交振幅调制（quadrature amplitude modulation，QAM）的概念。

QAM 可能的变化有无穷多个。图 7-40 给出了其他一些方案。图 7-40a 给出了最简单的 4-QAM 方案（4 个不同信号元素类型），它使用单极 NRZ 信号来调制每个载波。这与 BASK（OOK）的限制一样。图 7-40b 给出了另一个使用极性 NRZ 的 4-QAM 方案，但它是完全跟 QPSK 一样。图 7-40c 给出了另一个 4-QAM 方案，这个方案使用带两个正电平的信号来调制两个载波。最后，图 7-40d 给出了带有 8 个电平（4 个正 4 个负）信号的 16-QAM 星座图。

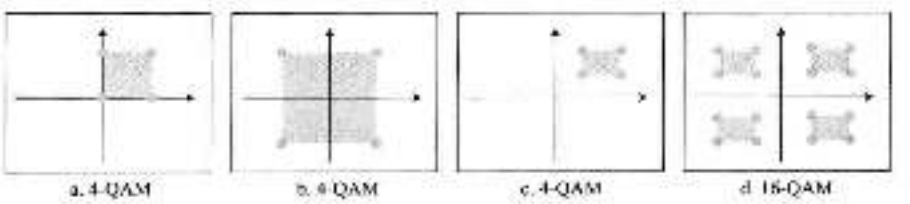


图 7-40　一些 QAM 方案的星座图

QAM 的带宽

QAM 传输所需的最小带宽与 ASK 和 PSK 传输所需的最小带宽相同。相对于 ASK 来说，QAM

与 PSK 具有同样的优点。

7.3.2　模拟到模拟转换

模拟到模拟转换（analog-to-analog conversion），或者叫模拟调制，是通过模拟信号来表示模拟信息的。人们可能会问，既然信号已经是模拟的，为什么还要调制模拟信号呢？答案是，如果介质具有带通性或者只有带通宽带可用，那么模拟信号就需要进行调制。政府为每个无线电基站分配基带带宽，每个基站生成的模拟信号都是低通信号，都在同一频率范围内。为了能够收听不同电台，需要将低通信号平移，使每一个信号对应不同的频率范围。

模拟到模拟转换可以通过三种方法实现：调幅（amplitude modulation，AM）、调频（frequency modulation，FM）和调相（phase modulation，PM）。FM 和 PM 通常归在一起。

调幅

在调幅传输中，对载波信号进行调制，使其振幅随着调制信号的振幅变化而改变。载波的频率和相位保持不变。只有振幅随着信息改变。图 7-41 说明了其工作原理。调制信号是载波信号的包层。

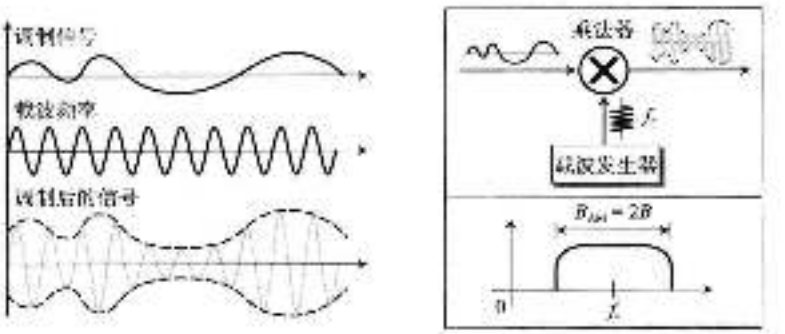


图 7-41　调幅

如图 7-41 所示，AM 通常使用简单乘法器实现，因为载波信号的振幅需要根据调制信号的振幅而变化。图 7-41 还表示了 AM 信号的带宽，是调制信号带宽的两倍，并且覆盖了以载波信号频率为中心的频率范围。但是，载波频率上边带和下边带的信号成分携带了完全相同的信息。基于这个原因，一些实现丢弃了一半信号，削减了一半的带宽。

调频

在调频传输中，载波信号的频率随着调制信号电平（振幅）的改变而调整。载波信号的振幅峰值和相位保持不变，但是当调制信号的振幅改变时，载波信号的频率相应地改变。图 7-42 说明了调制信号、载波信号和最终得到的 FM 信号的关系。

如图 7-42 所示，FM 通常使用用于 FSK 的压控振荡器实现。振荡器的频率根据输入电压（即调制信号的振幅）而变化。图 7-42 还说明了 FM 信号的带宽。实际带宽很难完全确定，但是可以从经验上得出是 $B_{FM} = 2(1 + \beta)B$，其中 β 是取决于调制技术的因子，一般为 4。

调相

在调相传输中，载波信号的相位随着调制信号的电平（振幅）变化而改变。载波信号的振幅峰值和频率保持不变，但是当调制信号的振幅改变时，载波的相位也相应地变化。可以从数学上证明 PM 和 FM 除一点不同外其余都相同。在 FM 中，载波频率的瞬时变化与调制信号的振幅成正比；在 PM 中，载波频率的瞬时变化与调制信号振幅的导数成正比。图 7-43 说明了调制信号、载波信号和最终得到的 PM 信号的关系。

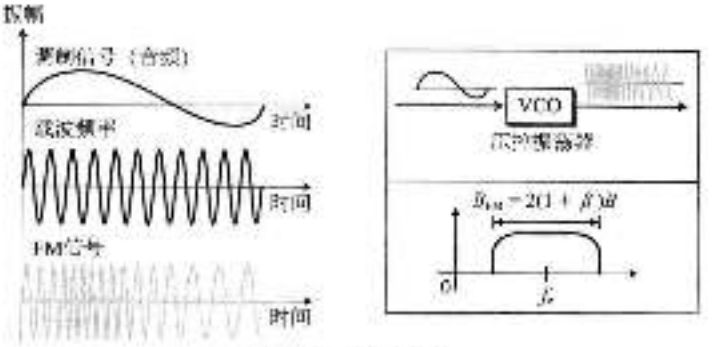


图 7.42　调制频率

如图 7-43 所示，PM 通常使用压控振荡器实现。振荡器的频率根据输入电平（调制信号的振幅）的导数而变化。

图 7-43 还说明了 PM 信号的带宽。实际带宽很难准确确定，但是从经验上可以得出是模拟信号带宽的若干倍。虽然公式显示 FM 和 PM 的带宽相同，但是 PM 中β较小（窄带大约是 1，宽带大约是 3）。

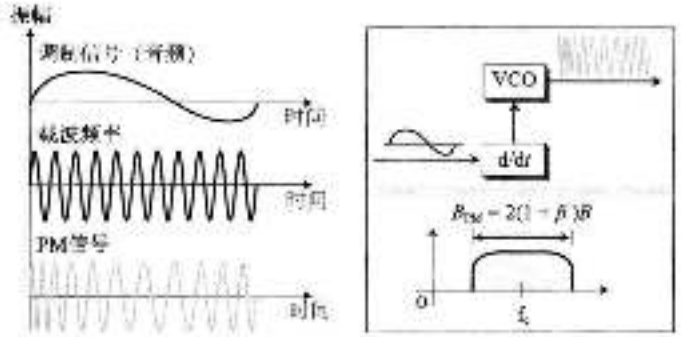


图 7.43　调制相位

7.4　带宽利用

实际生活中，链路的带宽都是有限的。合理运用带宽将是电路通信有待解决的主要问题。然而，所谓合理是与应用有关的。有时候我们需要将一些低带宽通道整合在一起形成一条高带宽通道，有时候为达到目的又需要扩展通道带宽，如保密、抗干扰等。在本节中，我们将系统地讨论带宽利用的两个主要类型：多路复用和扩频。在多路复用中，目标是效率，我们将一些通道合并成一个通道；在扩频中，目标是保密和抗干扰，我们插入冗余扩展通道带宽，这对于完成这些目标是必要的。

7.4.1　多路复用

只要连接两台设备的介质带宽比设备间传输所要求的带宽高时，该链路就可以被共享。多路复用（multiplexing）就是允许同时通过一条数据链路传输多个信号的一组技术。随着数据和通信应用的增加，通信量也不断增加。我们可以通过每需要一条新的通道时就建立一条单独链路的方式来满足这种增长，也可以安装更高带宽的链路并且在这些链路上使用多路复用技术。目前的技术包括诸如光纤、地面微波和卫星微波等高带宽介质。每一种介质都具有远远超过平均传输需求的承载能力。如果一条链路的带宽比连接在它上面的设备所需的带宽要大，那么多余的带宽就被浪费。一个高效的系统可以最大限度地使用所有设备的资源，而带宽是数据通信中最为珍贵的资源之一。

在多路复用的系统中，n 条线路共享一条链路的带宽。图 7-44 说明了多路复用系统的基本形式。左侧的线路将它们的传输流送到多路复用器（mutiplexer），多路复用器将这些流组成一个单独的流（多对一）。在接收端，这个传输流被分离器（demultiplexer）接收，并分解成原来几个独立的传输流（一对多），并直接发送到对应的线路上。图 7-44 中的线路（link）一词指的是物理通路。通道（channel）一词是指给定一对设备之间传送传输信号的链路的那部分。一个链路可以有多个（n）通道。



图 7-44　链路划分为通道

有三种基本的多路复用技术：**频分多路复用**（frequency-division multiplexing，FDM）、**波分多路复用**（wavelength-division multiplexing，WDM）和**时分多路复用**（time-division multiplexing，TDM）。前两种技术用于模拟信号，而第三种用于数字信号。

有些书中也把术语 FDM 称为 FDMA（或者把 TDM 称为 TDMA）。我们认为 FDMA 和 TDMA 是数据链路层定义的访问协议，而 FDM 和 TDM 服务是物理层的。我们在第 6 章中讨论 FDMA 和 TDMA。虽然有些教科书将码分多路访问（code division multiple access，CDMA）看做是第四种技术，但我们还是把它当做数据链路层的访问方法讨论，因为它在物理层没有相应的服务。

频分多路复用

频分多路复用（frequency-division multiplexing，FDM）是一种模拟技术，在链路带宽（以 Hz 为单位）大于要传输的信号的带宽之和时采用。在 FDM 中，每个发送设备生成的信号用于调制不同的载波频率。调制后的信号再被合并为一个可以通过链路传输的复合信号。为了适应调制信号，载波频率被分离为足够的带宽。这些带宽的范围就是不同信号通过的通道。通道之间有狭长的未使用的带宽，即防护频带（guard band）进行分离，以防止信号重叠。另外，载波频率必须不会影响原始的数据频率。

图 7-45 给出了 FDM 的概念描述。在这个图中，传输通路分为三个部分，每一部分都是传输某一个流的通道。

我们认为 FDM 是模拟多路复用技术，但这不是说 FDM 不能把源端的发送数字信号组合在一起。在 FDM 使用多路复用之前，数字信号可以转换成模拟信号。



图 7-45　频分多路复用

例 7.14　假设一个语音通道占用的带宽是 4kHz。要将三个语音通道合并到一条带宽为 12kHz（20-32kHz）的链路，使用频线图表示这一配置过程，这里假定不使用防护频带。

解答

将三个语音通道平移（调制）到不同的带宽，如图 7-46 所示。第一个通道使用 20-24kHz 的

带宽，第二个通道使用 24~28kHz 的带宽，第三个通道使用 28~32kHz 的带宽。然后，如图 7-46 所示将它们合并。在接收端，每个通道接收到完整信号后，使用滤波器将自己的信号分离出来。第一个通道使用能够通过 20~24kHz 频率成分的滤波器，并过滤（丢弃）任何其他频率成分。第二个通道使用能够通过 24~28kHz 频率成分的通过滤器，而第三个通道使用能够通过 28~32kHz 频率成分的滤波器。最后，每一个通道将频率平移到从 0 开始的起点。

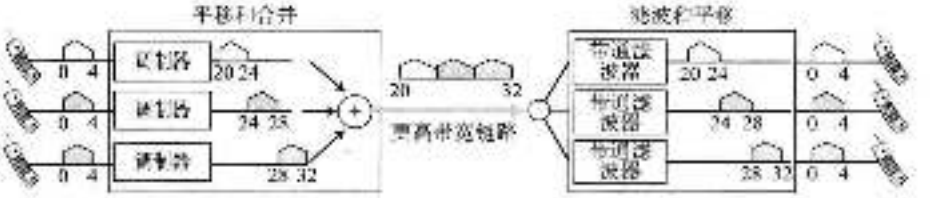


图 7-46　例 7.14

波分多路复用

波分多路复用（wavelength-division multiplexing，WDM）用于具有高数据速率传输能力的光纤电缆。光纤电缆的数据速率高于金属传输介质的数据速率，将光缆用作单一线路会浪费可用带宽。多路复用允许将多条线路合并成为一条。

除了多路复用和多路分离包括通过光纤通道传输的光信号以外，WDM 在概念上与 FDM 一样。其原理是一样的：都是将不同频率的不同信号合并。但其差别是这些频率非常高。

图 7-47 给出了 WDM 的多路复用器和分离器的示意图。来自不同源带的不同波长的光合并成一种波长较宽的光。在接收器端，信号通过信号分离器进行分离。

WDM 的一个应用是同步光纤网（SONET），其中多条光纤线路进行多路复用和信号分离。我们在第 5 章讨论过 SONET。

时分多路复用

时分多路复用（time-division multiplexing，TDM）是一个数字化过程，它允许多个连接共享一条高带宽链接。与 FDM 共享一部分带宽不同的是，TDM 是在时间上共享。每个链接占用链路的一个时间段。图 7-48 给出了 TDM 的示意图。注意，同一条链路都用于 FDM，但是这里表示的链路分割是在时间上的而不是在频率上的。在图中，信号 1、2、3 和 4 依次占用链路。

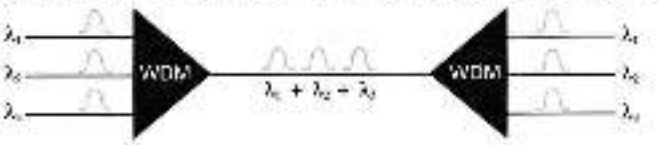


图 7-47　波分多路复用



图 7-48　TDM

注意，在图 7-48 中只涉及多路复用而不是交换。这意味着来自源 1 的报文中的所有数据总有一个特定的目的地，它可能是 1、2、3 或 4。传送到固定目的地是不变的，这与交换是不一样的。

我们可将 TDM 划分为两种方案：同步的和统计的。首先讨论同步时分多路复用（synchronous TDM），然后说明它与统计时分多路复用（statistical TDM）有哪些不同。在同步 TDM 中，即使没有发送数据，每个输入端也允许连接输出器。

同步 TDM

在同步 TDM 中，每个输入连接的数据流被划分为多个单元，其中每个输入占用一个输入时间间隙。一个单元可以是一位、一个字节或者是一个数据块。每个输入单元成为一个输出单元，占用一个输出时隙，但是每个输出单元的持续时间是输入单元持续时间的 n 分之一。如果输入时间间隙是 T 秒，则输出时间间隙是 T/n 秒，这里的 n 是连接数。换言之，输出连接单元具有较短的持续时隙，输出速率更快。图 7-49 显示了同步 TDM 的一个实例，其中 n 等于 3。

在同步 TDM 中，每个输入连接的全部数据单元组成一个帧（不久我们将会看到其理由）。如果有 n 条连接，则一帧则分为 n 个时隙，一个时隙分配给每个单元。如果输入单元持续 T 秒，则每个时隙持续时间为 T/n 秒，而每个帧的持续时间是 T（除非帧携带其他信息，这在不久后我们将会看到）。

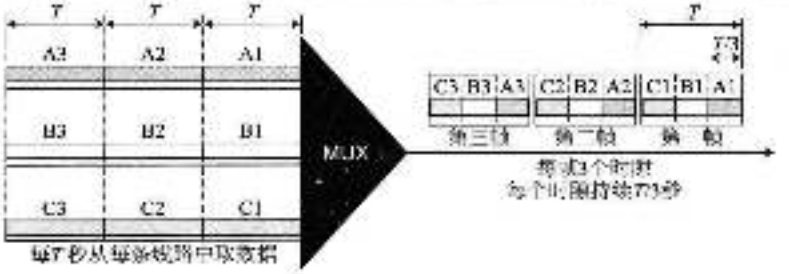


图 7-49　同步时分多路复用

输出链路的数据速率必须是单个连接的数据速率的 n 倍，以确保数据流动。在图 7-49 中，链路数据速率是一条连接的数据速率的 3 倍。同样，一条连接中的一个单元的持续时间是时隙（链路中一个单元的持续时间）的 3 倍。图中将多路复用之前的数据表示为多路复用后数据大小的 3 倍。这只是表达一种思想，即每个单元在多路复用之前的持续时间是多路复用后的 3 倍。

时隙分组为多帧。帧由多个时隙组成的一个完整的循环构成，每个时隙专用于每一个发送设备。在具有 n 条输入线路的系统中，每个帧有 n 个时隙，分配每个时隙用于传送来自特定输入线路的数据。

例 7.15　图 7-50 表示了四个输入数据流和一个输出数据流的同步 TDM。数据单元是 1 位。试求：（a）输入位的持续时间；（b）输出位的持续时间；（c）输出比特率；（d）输出帧的速率。

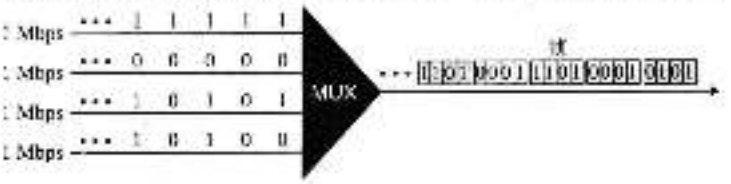


图 7-50　例 7.15

解答

这些问题回答如下：

1. 输入位的持续时间是比特率的倒数：1/1Mbps = 1μs；
2. 输出位持续时间是输入位持续时间的四分之一，即 1/4μs；
3. 输出比特率是输出位持续时间 1/4μs 的倒数，即 4Mbps。这也可以从输出速率比输入速率

块 4 倍这一点推出，即输出速率 = 4 × 1Mbps = 4Mbps；

4. 帧速率需与任一输入速率相同，因此帧速率是每秒 10 000 帧。因为我们发送每帧 4 位，所以我们可用多路复用证明前一结论，用每帧位的个数证明帧速率。

例 7.16　电话公司通过一种数字信号的层次结构实现 TDM，称为**数字信号**（digital signal，DS）服务或数字层次结构（digital hierarchy）。图 7-51 说明了每一级支持的数据速率。这些服务的商业实现称为 **T 线路**（T lines）。

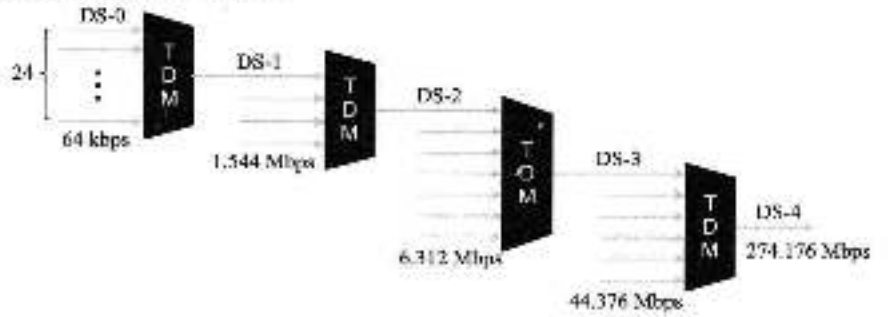


图 7-51　数字层次结构

- **DS-0** 服务是 64kbps 的单一数字通道。
- **DS-1** 是一种 1.544Mbps 的服务。1.544Mbps 是 24 × 64kbps 加上 8kbps 的开销得到的。可以用做单一的 1.544Mbps 传输服务，或者作为 24 个 64kbps 通道的多路复用，或者用户需要时还可以用于适合于 1.544Mbps 容量的其他服务类型的组合形式。
- **DS-2** 是一种 6.312Mbps 服务。6.312Mbps 是 96 × 64kbps 加上 168kbps 的开销得到的。可以用做单一的 6.312Mbps 传输服务，或者用做 4 个 DS-1 通道的多路复用，或者用做 96 个 DS-0 通道，或者用做这些服务类型的组合形式。
- **DS-3** 是一种 44.376Mbps 服务。44.376Mbps 是 672 × 64kbps 加上 1.368Mbps 开销得到的。可以用做单一的 44.376Mbps 传输服务，或者用做 7 个 DS-2 通道的多路复用，或者用做 28 个 DS-1 通道，或者用做 672 个 DS-0 通道，或者用做这些服务类型的组合形式。
- **DS-4** 是一种 274.176Mbps 服务，274.176Mbps 是 4032 × 64kbps 加上 16.128Mbps 开销得到的。可以用做 6 个 DS-3 通道的多路复用，或者用做 42 个 DS-2 通道，或者用做 168 个 DS-1 通道，或者用做 4032 个 DS-0 通道，或者用做这些服务类型的组合形式。

统计时分多路复用

我们在上一节看到，在同步 TDM 中每个输入在输出帧中都占有一个时隙。如果有些输入线没有数据发送，那么效率就不高。在统计时分多路复用中，时隙是动态分配的，这样有利于提高带宽的效率。仅当输入线有发送数据，时隙才有意义并在输出帧中给予一个时隙。在统计多路复用中，每个帧中时隙的个数小于输入线的条数。多路复用器循环顺序地检测每条输入线，如果输入线有数据发送，则对输入线分配一个时隙，否则跳过这条线检测下一条线。图 7-52 表示了一个同步和一个统计 TDM 的例子。前一个例子由于对应的线没有数据发送，某些时隙是空的。然后在后一个例子中，只要有一条输入线有数据要发送，那么就没有空的时隙。

图 7-52 还表明了同步 TDM 和统计 TDM 时隙之间的主要差别。在同步 TDM 中，输出时隙全部由数据占用，而在统计 TDM 中，输出时隙需要携带数据和目的地址。在同步 TDM 中不需要寻址。作为输入和输出的地址之间关系是同步的和指定的。例如，我们知道输入线 1 后转向输出线 2。如果多路复用器和分离器是同步的，这是可以保证的。在统计 TDM 中，由于输入和输出之间不存在指定或预定的时隙，它们之间没有固定的关系，需要在每个时隙中包含接收方地址以表明将要传

送的地方。寻址最简单形式可用定义 N 个不同输出线的 n 位表示，其中 $n = \log_2 N$。例如，对于 8 条输出线，需要 3 位地址。

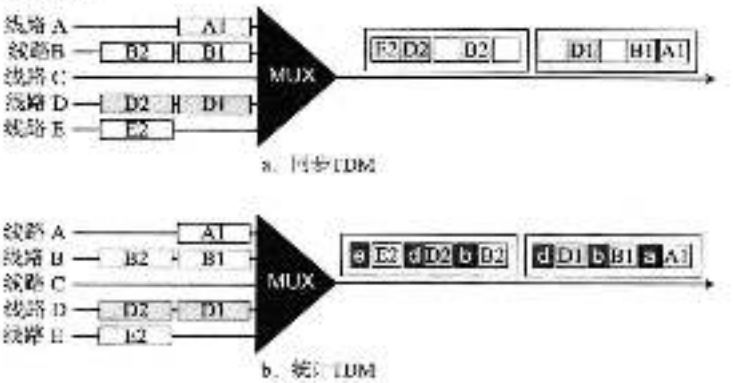


图 7-52　TDM 时隙比较

7.4.2　扩频

多路复用是把某些源端的信号组合在一起获得带宽的效率，链路的可用带宽在各个源端之间划分。在扩频（spread spectrum，SS）中，也是把来自某些源端的信号组合在一起形成一个更宽的带宽，可是目的略有不同。扩频是为无线应用而设计的（无线 LAN 和 WAN）。在这些类型应用中，我们所关注的比带宽效率更为重要。在无线应用中，所有基站都以空气（或真空）作为通信传输介质。基站共享这个介质需要没有窃听者拦截，也需要没有恶意的人侵者的干扰（例如军事行动）。

为了达到这个目的，扩频技术增加了冗余部分，扩展原始信号的频带以满足每个基站的需要。如果每个基站要求的带宽是 B，扩频将带宽扩展到 B_{ss}，这里 $B_{ss} >> B$。扩大的带宽允许源端用有防护的封装将它的报文进行更安全的传输，类似发送精美的礼品，把它放在特制礼盒中防止在运输过程中受到损坏，可用一种优级传送服务保证包装安全。

图 7-53 表示了扩频的思想。扩频通过两个原则达到它的目的：

1. 对每个站点需要分配的带宽显然要比它所需要的带宽更大；

2. 原来的带宽 B 扩大到 B_{ss} 必须由一个与原来的信号无关的过程来做。换言之，信号由源端生成后，扩频过程才发生。

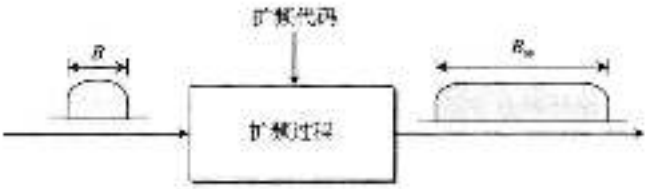


图 7-53　扩频

信号由源端生成后，扩频过程利用扩频代码并扩大带宽。图中表示了原来的带宽 B 和扩大的带宽 B_{ss}。扩频代码是伪随机二进制数据流，但它实际是一种模式。

扩频的技术有两种：跳频扩频（FHSS）和直接序列扩频（DSSS）。

跳频扩频

跳频扩频（frequency hopping spread spectrum，FHSS）使用源信号时制 M 个不同的载波频率。在某一时刻用信号调制 1 个载波频率。在下一时刻信号调制另一个频率。虽然调制是一次使用一个频率，但在最终用了 M 个频率。源信号扩展后占用的带宽是 $B_{FHSS} >> B$。图 7-54 表明了 FHSS 总的设计情况。

伪随机代码生成器（pseudorandom code generator）称为伪随机噪声（pseudorandom noise，PN），对每个跳周期 T_h 生成一个 k 位模式。频率表使用这个模式查找频率作为这个跳周期的频率，而且

通过它传送到频率合成器。频率合成器生成该频率的载波信号，同时源信号调制这个载波。假定我们决定采用 8 个跳频率。实际上这是最简单的例子，但恰好说明问题。在这个情形中，M 是 8，k 是 3。伪随机代码生成器将生成 8 个不同的 3 位模式，这些映射到频率表中 8 个不同的频率。

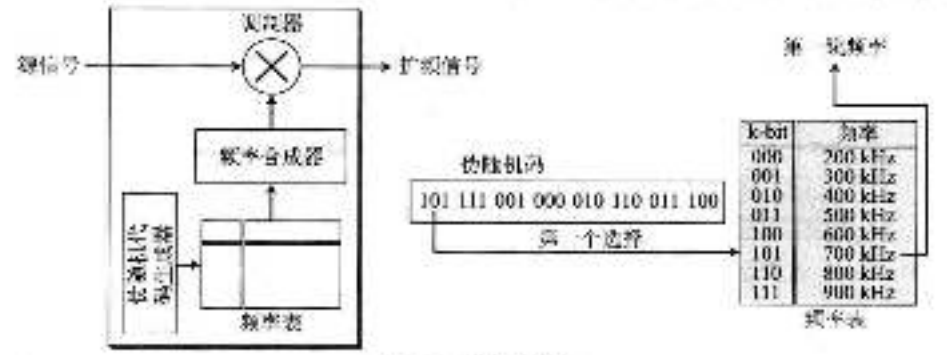


图 7-54　跳频扩频

这个基站的模式是 101，111，001，000，010，110，011，100。注意，模式是伪随机的，它在 8 次跳后重复循环。这就是说，在跳周期 1，模式是 101，所选的频率是 700kHz，源信号调制这个载波频率。所选第 2 个模式是 111，它选得频率是 900kHz。第 8 个模式是 100，所选的频率是 600kHz。8 次跳频后，模式重复，再从 101 开始。图 7-55 表示了信号如何从载波跳到载波的循环，并假定源信号要求的带宽是 100kHz。

它可以表示该方案能实现前面提到的目的。如果有许多个 k 位的模式而跳周期是短的，那么发送端和接收端可具有保密性。如果入侵者企图窃听传输信号，他仅能获取很少一部分数据，因为他不知道扩频序列，不能很快地使他能适应下一跳变。这个方案也有抗干扰的效果。一个恶意的发送者可能会将噪声发送到一个跳周期中（随机地）干扰信号，但不是全部周期。

带宽共享

如果跳频数是 M，那么我们可将 M 个通道多路复用为使用同一带宽 B_{ss} 的一个通道。可能是由于一个站点在每跳周期中只用一个频率，其他 $M-1$ 个站点可使用另外的 $M-1$ 个频率。也就是说，如果使用适当的调制技术，比如多 FSK（MFSK），M 个不同站点可用同一带宽 B_{ss}。FHSS 与 FDM 相似，如图 7-56 所示。

图 7-56 表示了使用 FDM 的 4 个通道与使用 FHSS 的 4 通道的例子。在 FDM 中，每个站点用带宽的 $1/M$，但是固定的分配；在 FHSS 中，每个站点占用的带宽的 $1/M$，但跳到跳改变分配。

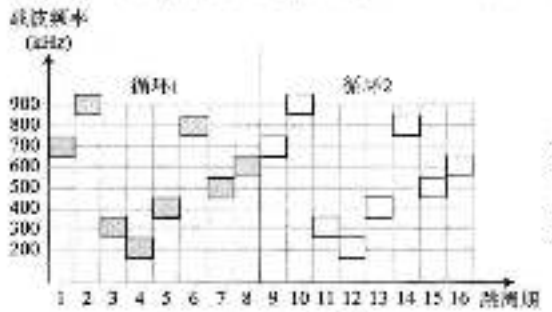


图 7-55　FHSS 循环

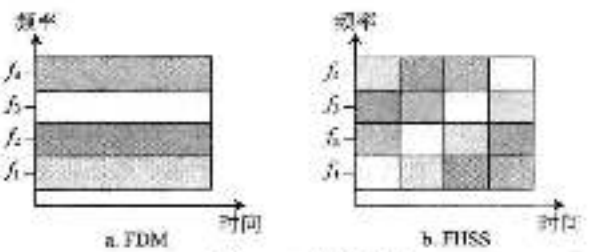


图 7-56　带宽共享

直接序列扩频

直接序列扩频（direct sequence spread spectrum，DSSS）技术也是扩大源信号的带宽，但方法

不同。在 DSSS 中，每个数据位用扩展编码的 n 位代替。也就是说，每一位被编码为 n 个码片，此处码片的速率是数据比特率的 n 倍。图 7-57 表示了 DSSS 的概念。

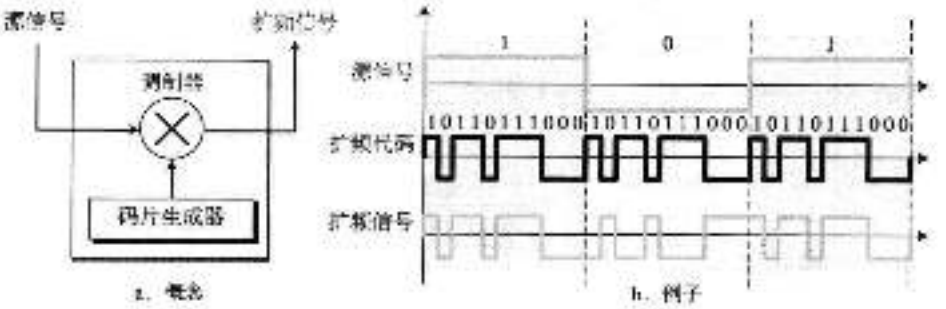


图 7-57　DSSS

作为实例，让我们考察无线局域网中所有的直接序列扩频，著名的巴克序列（Barker sequence），此时 n 为 11。假定源信号和码片生成器中的码片使用极性 NRZ 编码。图 7-57 表示了码片和用码片多路复用源信号得到扩频信号的结果。

在图 7-57 中，扩频码是模式 10110111000（该例子中）的 11 位码片。如果源信号的速率是 N，则扩频信号的速率是 $11N$。这就是说，扩频信号要求的带宽比源信号带宽大 11 倍。如果入侵者不知道该编码，扩频信号可提供保密。如果每个站点使用不同的编码，它可提供抗干扰的能力。

7.5　传输介质

在这章中，我们讨论了许多与物理层相关的问题。本节讨论传输介质。传输介质实际上位于物理层以下并直接由物理层控制。可以说传输介质属于 0 层。图 7-58 说明了传输介质相对于物理层的位置关系。

传输介质（transmission medium）可广义地定义为可以从源端传送信息到目的端的任何东西。例如，用餐的两个人交谈时，传输介质是空气。空气也能用烟信号或旗语传递信息。对于一封书信，传输介质可能是邮递员、卡车或飞机。



图 7-58　传输介质和物理层

在数据通信中，信息和传输介质的定义更加具体。传输介质通常是自由的空间、金属或光纤。信息通常是一种信号，是来自另一种形式的数据转换的结果。

电磁能是电场和磁场相互振荡的组合形式，它包括电力、无线电波、红外线、可见光、紫外线、X 射线、伽马射线和宇宙射线。每一种都对应于电磁频谱（electromagnetic spectrum）的一部分。但是目前，并非所有频谱都可用于电子通信。这些可利用频谱的介质也只限于有限的几类。

在通信中，传输介质可以分为两大类：有向的和无向的。有向介质包括双绞线、同轴电缆和光缆。无向介质通常是空气。

7.5.1　有向介质

有向介质（guided media）是指那些在设备之间提供通路的介质，包括双绞线（twisted-pair cable）、同轴电缆（coaxial cable）和光缆（fiber-optic cable）。沿着这类介质传输的信号，其传输方

向和传播范围受介质的物理边界限制。双绞线和同轴电缆使用金属（铜）导体接收和传输电流形式的信号。光纤（optical fiber）是一种玻璃线缆，接收和传输光波形式的信号。

双绞线

双绞线由两根导线（通常是铜线）构成，其中的每一根导线都有自己的塑料绝缘层，两者绞在一起，如图 7-59 所示。

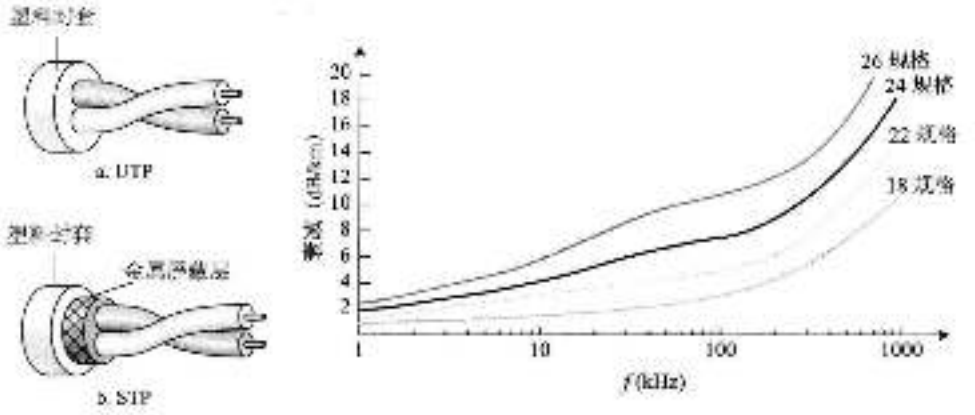


图 7-59　双绞线

线路中的一条用于将信号传输给接收端，另一条仅作为接地参考点。接收端使用两条线路的电平之间的差值。

除了发送端在线路上发送的信号之外，干扰（噪声）和串扰也可能会影响到两条线路并产生有害的信号。如果两根导线是平行的，这些有害信号的影响在两条线路中就会不同，因为它们相对于噪声或者串扰所处的位置是不同的。这会在接收方产生差值。通过将两根线绞合在一起，可以维持平衡。通信中使用的最常见的双绞线是指非屏蔽双绞线（unshielded twisted-pair，UTP）。IBM 还生产了一种自己使用的双绞线类型，称为屏蔽双绞线（shielded twisted-pair，STP）。STP 电缆有一层金属薄片或者网状的包覆材料将每一对带有绝缘层的导体包围起来。尽管金属包装层可以通过防止噪声或者串扰而提高电缆的质量，但是它更加笨重和昂贵。

性能

测试双绞线性能的一个方法是对比频率和距离的衰减曲线。双绞线允许通过的频率范围比较宽。但是，图 7-59 说明随着频率的升高，以每公里的分贝数（dB/km）进行测量，在频率超过 100kHz 时衰减数值会突然升高。注意，规格（gauge）是电线粗细的标准计量单位。

应用

双绞线用于电话线路，提供语音和数据的通道。本地环路，即连接用户到中心电话机房的线路，最常用的是非屏蔽双绞线。电话公司使用的 DSL 线路提供高数据速率连接，这种线路也使用具有高带宽容量的非屏蔽双绞线。局域网，如 10Base-T 和 100Base-T，也使用双绞线。我们在第 5 章讨论过这些网络。

同轴电缆

同轴电缆（或称为同轴）与双绞线相比，可以传输更高频率范围的信号，部分原因是这两种介质的构造有很大的不同。同轴电缆不使用两根电线，而是使用一根位于中央的实心或者多股绞合的核心金属丝导体（通常是铜的），导体封装在绝缘护套中，然后再把它封装在金属箔、金属网或者两者组成的外部导体中。外部金属包装层可以屏蔽噪声，又可以作为第二导体，构成回路。外部导体的外面再由绝缘护套封装，最后每条电缆由一层塑料外套保护（见图 7-60）。

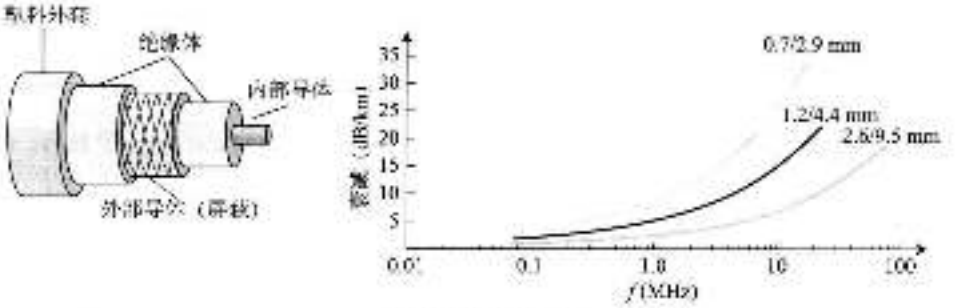


图 7-60　同轴电缆

性能

同双绞线一样，我们也能测试同轴电缆的性能。从图 7-60 中可以看出，同轴电缆的衰减要高于双绞线，也就是说，尽管同轴电缆有更高的带宽，但是信号减弱很快因而经常需要使用中继器。

应用

同轴电缆最初是在模拟电话网络中使用的，单个的同轴电缆网络能够传送 10 000 个语音信号。后来它用于数字电话网络，其中单个的同轴电缆能够以高达 600Mbps 的速率传送数据。但是，电话网络中的同轴电缆目前大部分已经被光缆取代了。

有线电视网络也使用同轴电缆。在传统的有线电视网络中，整个网络都使用同轴电缆。但是，后来有线电视提供商用光缆取代了网络中的大部分同轴电缆，混合网络只是在网络边界、靠近消费者房屋的地方使用同轴电缆。有线电视使用 RG-59 同轴电缆。

另一种常见的应用是在传统的以太局域网中（见第 5 章）。因为它的带宽高以及由此产生的高速率，因此在早期的以太局域网中选用同轴电缆进行数字传输。粗缆以太网有专用的连接器。

光缆

光缆是由玻璃或者塑料构成，能够传输光信号。为了理解光缆，我们首先需要研究光的几个本质特性。

光在通过单一物质时，以直线传播。如果光线从一种物质突然进入另一种物质（密度更高或更低），则会改变方向。图 7-61 说明了光线从高密度物质进入低密度物质时是如何改变方向的。

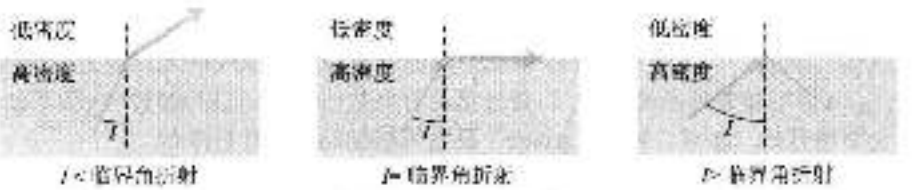


图 7-61　光线的折射

如图 7-61 所示，如果入射角（angle of incidence）（入射光线与界面上入射点发现的夹角小于临界角）小于临界角（critical angle），光线会沿着较接近于表面的方向发生折射（refract）。如果入射角等于临界角，光线则会沿着表面弯曲。如果入射角大于临界角，光线则会反射（reflect，发生转向）并再次在高密度物质中传输。注意，临界角是物质的一种特性，不同物质的临界角的值是不同的。

光线使用反射来引导光通过通道。玻璃心或者塑料外面环绕着低密度的玻璃的或塑料的包层（cladding）。这两种材料的密度差值必须达到如下条件，即通过纤芯传播的一束光必须完全被包层反射，而不发生折射。见图 7-62。

传播模式

目前的技术支持两种模式（多模和单模）。通过这两种模式光线沿着光通道传播，每一种模式

与要光纤具有不同的物理特性。多模传输有两种实现形式：阶跃折射率模式和渐变折射率模式，见图 7-63。

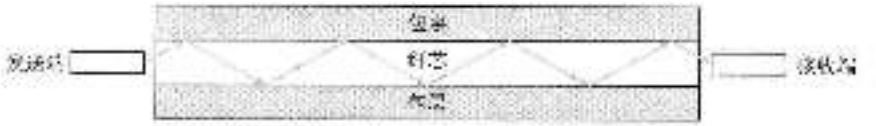


图 7-62　光纤

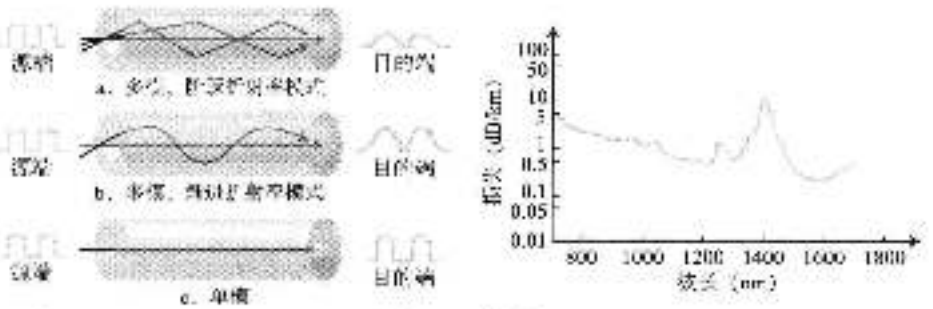


图 7-63　模式

在多模阶跃折射率光纤（multimode step-index fiber）中，纤芯的密度从中心到边缘保持不变。光束到达纤芯与包层交界处之前，以直线的形式通过等密度的纤芯，在交界处，突然遇到低密度的包层会引起光束传输角度的变化。阶跃折射率（step index）这个术语是指这种变化的突然性。

第二种类型的光纤，称为多模渐变折射率光纤（multimode graded-index fiber），通过这种光缆的信号可以减少失真。指数（index）这个词是指折射率，正如上面所看到的，折射率与密度有关。所以，渐变折射率光纤是一种具有不同密度的光纤。密度在纤芯的中心处最高，从中心到边缘逐渐降低，在边缘处最低。

单模使用阶跃折射率光纤并且使用聚焦效果更好的光源。这样可以把光束限制在更小的角度范围内，几乎接近于水平。单模光纤（single-mode fiber）在生产时使用直径比多模光纤要小得多且密度（折射率）非常低的光纤缆。密度的降低使临界角几乎接近 90°，这样光束的传播基本上是水平的。在这种情况下，不同光束的传播几乎相同，可以忽略延迟。所有的光束同时到达目的端，并可以重组为失真很小的信号。

性能

在图 7-63 中，衰减对于波长的变化曲线说明了光缆中的一种非常值得注意的现象，衰减曲线比双绞线和同轴电缆的衰减曲线要平坦，其性能如此之好，以至于在使用光纤时，只需要很少的中继器（实际上少于 1/10）。

应用

光纤通常用于主干网络中，因为它提供了很高的带宽，所以其成本是合算的。目前，对于 WDM，能以 1600Gbps 的速率传播数据。我们在第 5 章讨论过的 SONET 网络就提供了这样的主干网。

一些有线电视公司结合使用光纤和同轴电缆，可以构建混合网络。光纤提供主干结构，而同轴电缆则提供到用户住所的连接。这是一种性能成本合算的配置方案，因为用户端所需的带宽，使用光纤在经济上不合算。

局域网，如 100Base-FX 网（快速以太网）和 1000Base-X 网也使用光缆。10 千兆以太网也使用光缆。

7.5.2　无向介质：无线

无向介质（unguided media）不使用物理导体传输电磁波。这种类型的通信通常是指无线通信（wireless communication）。信号通常通过空气广播，能够被任何人接收，只要他拥有一台接收信号的设备。

图 7-64 表示了用于无线通信的部分电磁频谱，范围是 3kHz～900THz。

无向信号能通过多种途径从信号源传输到目的地。其方法有：地表传输、天空传输和视线传播。

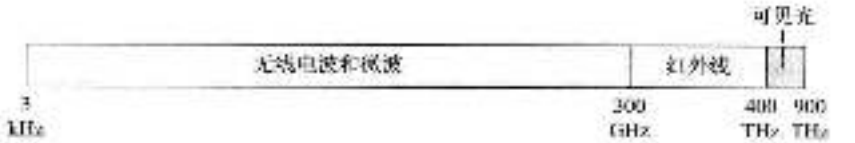


图 7-64　无线通信使用的电磁频谱

在地表传输（ground propagation）中，无线电波通过地球的大气层的最低部分紧挨地球传播。这些低频率信号通过天线沿着地球表面向各个方向发生。传输距离取决于信号功率的大小。功率越强，传输的距离越远。**在天空传播**（sky propagation）中，较高频率的无线电波向上传播到电离层（大气中的一层，该层中粒子以离子的形式存在），从电离层再反射回地球。这种类型的传输可以使用较低的传输功率，传输更远的距离。**在视线传播**（line-of-sight propagation）中，频率非常高的信号沿着直线在天线之间传输。天线必须是有向的、互相正对的，而且必须足够高或者距离足够近，以免受到地球曲率的影响。视线传播是很难处理的，因为无线电传输不能完全聚焦。

电磁频谱中，定义为无线电波和微波的部分被分成八个区域，称为波段（band），每一个波段都受政府机构的管制。这些波段从甚低频率（very low frequency，VLF）到极高频率（extremely high frequency，EHF）。表 7-1 列出了这些波段及其频率范围、传播方法和应用。

我们可以把无线传输分为三大组：无线电波、微波和红外波。

表 7-1　波段

波　段	范　围	传　播	应　用
VLF（甚低频）	3～30 kHz	地面	远程无线电
LF（低频）	30～300 kHz	地面	无线电导航台
MF（中频）	300 kHz～3 MHz	天空	AM 广播
HF（高频）	3～30 MHz	天空	民用波段（CB）和航海/航空通信
VHF（甚高频）	30～300 MHz	天空和视线	VHF 电视波段和 FM 广播
UHF（超高频）	300 MHz～3 GHz	视线	UHF 电视波段、移动电话、寻呼、卫星
SHF（特高频）	3～30 GHz	视线	卫星通信
EHF（极高频）	30～300 GHz	视线	雷达和卫星

无线电波

尽管无线电波和微波之间没有明确的界限，但是通常将频率范围在 3kHz～1GHz 之间的电磁波称为无线电波（radio wave），频率范围在 1GHz～300GHz 的电磁波称为微波（microwave）。但是，作为分类的标准，使用波的特性比使用频率的特性要好。大部分无线电波是全方向的。在天线发射无线电波时，它们会向各个方向传播。这意味着发射天线和接收天线不必对准。发射天线发射的无线电波，可以被任何接收天线接收。全方向特性也有一个缺点：一只天线发射的无线电波，容易受到另一只使用相同频率或波段的天线所发射的信号的干扰。

无线电波，尤其是那些以天空模式传播的电波，可以传输很长的距离。这使得无线电波成为诸

如 AM 广播等远距离广播的首选。

无线电波，特别是那些低频率、中频率的电波，可以穿透墙体。这种特性既有优点也有缺点。它是有优点的，例如，AM 收音机可以在建筑物内接收信号。它也是有缺点的，因为不能隔离建筑物内部和外部的通信。无线电波的波段与微波相比较窄，仅在 1GHz 以下。当这一波段被划分为多个子波段时，子频带也很窄，导致在数据通信中数据速率较低。

几乎整个波段都受政府机构（比如，在美国是 FCC）的管制。使用波段的任何部分必须得到这些机构的许可。

微波

频率范围为 1GHz ~ 300GHz 的电磁波称为微波。微波是单向的。在天线发射微波时，它们可以聚集在很窄的范围内。这意味着发射天线和接收天线需要对准。单向特性具有一个很明显的优势，即一对对准的天线不受另一对对准天线的干扰。下面描述了微波传播的一些特性：

- 微波传播属于视线传播。安装天线的塔必须相互可见。如果相距很远，天线塔就必须很高。地球的曲率和其他障碍物不允许两座很低的天线塔使用微波进行通信。长距离通信时，通常需要中继站。
- 甚高频微波不能穿透墙体。在接收机位于建筑物内部时，这种特性就是一种缺点。
- 微波波段相对较宽，接近于 299GHz。所以，它能够分配更高的子波段而且可以获得更高的数据速率。
- 要使用波段中的某个部分，就必须得到政府机构的许可。

应用

由于微波具有单向性，因为在发送端和接收端之间需要单播（一对一）通信时非常有用。微波应用于移动电话、卫星网络和无线局域网。

红外线

红外信号（infrared wave）的频率范围为 300GHz ~ 400THz（波长为 1mm ~ 770nm），可以用于短距离通信。红外信号的频率很高，不能穿透墙体。这种优点可以防止系统之间相互干扰，在一个房间内的短距离通信不会受到相邻房间的另一个系统的影响。在使用红外遥控器时，不会干扰邻居红外遥控器的使用。但是，这种特性也使得红外信号无法进行长距离通信。另外，在建筑物外面不能使用红外波，因为太阳射线中包含可能干扰通信的红外波。

7.6　章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。在本书末列出了方括号中的参考资料。

书籍

覆盖多媒体与服务质量的一些书籍包括：[Pea 92]、[Cou 01]、[Ber 96]、[Hsu 03]、[Spi 74]、[Sta 04]、[Tan 03]、[G & W 04]、[SSS 05]、[BEL 01]和[Max 99]。

小结

数据必须转换为电磁信号进行传输。模拟信号是连续状态，取连续的值；数字信号具有离散状态，取离散的值。模拟信号可以在一个范围内有无穷个值；而数字信号具有有限个值。在数据通信中，我们通常使用周期模拟信号和非周期数字信号。

数字到数字转换有二种技术：线路编码、块编码和扰动。线路编码是将数据转换成数字信号的处理过程。块编码提供冗余以保证同步和内部差错检测。将模拟信号转换成数字数据（数字化）最

常用的技术是脉冲码调制（PCM）。

数字到模拟转换是指根据数字数据中信息而改变模拟信号的某种特性的过程。数字到模拟转换使用下列方法完成：幅移键控（ASK）、频移键控（FSK）、相移键控（PSK）以及 ASK 和 PSK 结合的正交振幅调制（QAM）。模拟到模拟转换可使用三种方法：调幅（AM）、调频（FM）和调相（PM）。

带宽的利用就是使用有用的带宽达到特殊的目的。多路复用可达到特高效率的目的；扩频可达到保密和抗干扰的目的。

传输介质位于物理层以下。有向介质在设备之间提供一条物理通道。双绞线、同轴电缆和光纤是最常用的有向介质。无向介质（通常指空气）不能使用物理导体传输电磁波。

7.7　习题集

测试题

本章的交互式测试题请参见这本书的网站，在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q7-1　给定正弦波的频率，怎样计算相应的周期？

Q7-2　下列哪个测量值是信号在任何时间都存在的？

a. 振幅　　b. 频率　　c. 相位

Q7-3　是否可用时域图观察出一个信号是周期的还是非周期的？为什么？

Q7-4　语音信号的频域图是离散的还是连续的？

Q7-5　下列哪个导致传输受损？

a. 衰减　　b. 滤波　　c. 噪音

Q7-6　下列哪个是低通通道的特性？

a. 带宽从 0 开始的通道。　　b. 带宽不从 0 开始的通道。

Q7-7　语音信号从麦克风到录音机的发送是基带传输还是宽带传输？

Q7-8　在局域网中，一个数字信号从一个站到另一个站的发送是基带传输还是宽带传输？

Q7-9　下列哪个是基带传输的定义？

a. 使用低通通道发送不经过调制的数字或模拟信号。

b. 使用基带通道调制数字或模拟信号。

Q7-10　周期复合信号怎样分解成单独的频率成分？

Q7-11　下列哪个定义了无噪声通道理论上的最大比特率？

a. 奈奎斯特定理　　b. 香农容量原理

Q7-12　下列哪个技术是数字到数字转换的例子？

a. 线路编码　　b. 块编码　　c. 脉冲调制

Q7-13　块编码的定义和目的是什么？

Q7-14　描述 PCM。

Q7-15　定义模拟传输。

Q7-16　下列哪个是在数字到模拟转换机制中，模拟信号转变成数字信号的特性？

a. ASK　　b. PSK

Q7-17　下列哪种数字到模拟转换技术更易受噪声影响？

a. ASK　　b. PSK

Q7-18　星座图中横轴表示信号的哪种成分？

Q7-19　下列哪个是模拟到模拟转换中，模拟信号转换成低频模拟信号的特性？

a. FM　　b. PM

Q7-20　描述多路复用技术。

Q7-21　多路复用技术中链路的定义。

Q7-22　三种多路复用技术中，哪种是用来组合模拟信号的？
Q7-23　同步 TDM 的定义与统计 TDM 比较有什么不同？
Q7-24　定义 FHSS，解释它是如何实现带宽扩展的。
Q7-25　传输介质在 TCP/IP 协议簇中的位置是什么？
Q7-26　传输介质的两大类是什么？
Q7-27　有向介质包括哪二类？
Q7-28　光纤中包层的作用是什么？
Q7-29　描述波是如何通过天空从源端传输到目的端的。
Q7-30　描述全方向波是如何传播的。

思考题

P7-1　给定下列频率，计算相应的周期。
a. 24Hz　　b. 8MHz　　c. 140kHz

P7-2　下列各波的相移是多少？
a. 在时间 0 具有最大振幅的正弦波
b. 经过 1/4 周期达到最大振幅的正弦波
c. 经过 3/4 周期，上升段振幅达到 0 值的正弦波

P7-3　一个信号被分解成 5 个正弦波，它们频率分别为 0、20、50、100 和 200Hz，该信号的带宽是多少？它们的峰值振幅都相同，试画出频域中的信号并给出其带宽。

P7-4　一个带宽为 2000Hz 的复合信号由两个正弦波组成，其中一个频率为 100Hz，最大振幅为 20V，另一个最大振幅为 5V，画出信号并给出其带宽。

P7-5　一个正弦波频率为 100Hz，另一个正弦波频率为 200Hz。试问哪一个正弦波带宽较大？

P7-6　下列信号的比特率各是多少？
a. 一个位持续 0.001s　　b. 一个位持续 2ms　　c. 一个为持续 20μs

P7-7　一个设备以 1000bps 发送数据
a. 发送 10 位需要多长时间？　　b. 发送一个字符（8 位）需要多长时间？
c. 发送 100 000 个字符的文件需要多长时间？

P7-8　图 7-65 中的信号比特率是多少？

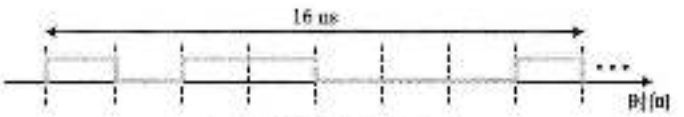


图 7-65　思考题 P7-8

P7-9　图 7-66 中的信号频率是多少？

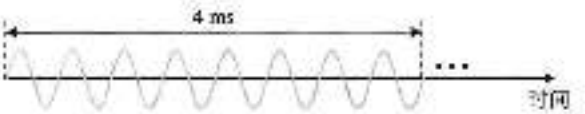


图 7-66　思考题 P7-9

P7-10　图 7-67 中的复合信号的带宽是多少？



图 7-67　思考题 P7-10

P7-11　周期复合信号包含 10kHz 到 30kHz 的频率成分，每一个成分的振幅是 10V，试画出其频谱。

P7-12　非周期复合信号包含 10kHz 到 30kHz 的频率成分，信号的最低频率成分和最高频率成分的振幅是 10V，20kHz 信号的振幅是 30V。假定振幅由最小值到最大值是渐变的，试画出其频谱。

P7-13　信号从 A 点到达 B 点，在 A 点时功率是 100W，在 B 点时功率是 90W。问衰减了多少分贝？

P7-14　信号衰减是 10dB，如果信号的初始功率是 5W，问衰减后的功率是多少？

P7-15　信号通过三级放大器，每级的增益是 4dB，问总增益是多少？信号放大了多少倍？

P7-16　如果通道的带宽是 5kbps，那么从设备发送 100 000 位的帧需要多长时间？

P7-17　信号在空气中的波长是 10m，经过 1000 个周期后的前端能传输多远？

P7-18　一条线路的信噪比是 1000，带宽是 4kHz，该线路能支持的最大数据速率是多少？

P7-19　要测试电话线路（带宽是 4kHz），信号是 10V，噪声是 10mV。该电话线支持的最大数据速率是多少？

P7-20　一个 2M 字节的文件，用 56kbps 通道（拨号上网）下载该文件需要多少时间？

P7-21　一台计算机显示器的分辨率是 1200 × 1000 个像素，如果每个像素使用 1024 个颜色，发送一个完整屏幕需要多少位？

P7-22　具有 200 毫瓦功率的信号通过 10 个设备，每个设备平均噪声是 2 毫瓦，问 SNR 是多少？SNR_{dB} 是多少？

P7-23　如果信号峰值电压值是噪声峰值的 20 倍，问 SNR 是多少？SNR_{dB} 是多少？

P7-24　需要提高通道的带宽，问题是：

a．如果想提高带宽两倍，如何改进速率？　　b．如果提高 SNR 两倍，如何改进速率？

P7-25　如果一个分组的长度是 1M 字节（1 字节等于 8 位），而通道的带宽是 200kbps，一个站发送一个分组需要多少时间？

P7-26　通道的传播速率是 2×10^8m/s，如果通道速率如下：

a．1Mbps　　b．10Mbps

问通道内的一位的长度是多少？

P7-27　如果链路具有 2ms 的延迟，在带宽如下：

a．1Mbps　　b．10Mbps

问在线路上填满有多少位？

P7-28　一条链路有 10 个路由器，每个路由器排队的时间是 2μs，处理时间是 1μs，在其上发送具有 5M 位的一个帧，问总的延迟是多少？链路长为 2000km，链路中内的光速为 2×10^8m/s，带宽是 5Mbps。决定总延迟的是哪些成分？哪些成分可以忽略不计？

P7-29　在数字传输中，发送端时钟比接收端时钟快 0.2%。如果发送端以 1Mbps 速率发送，问发送端每秒发送多少额外的位？

P7-30　使用下列数据流画出每个数据流的 NRZ-I 方案图，假定最近一个信号电平是正的，根据图猜测这种方案的带宽。

a．00000000　　b．11111111　　c．01010101　　d．00110011

P7-31　使用下列数据流画出每个数据流的曼彻斯特方案图，假定最近一个信号电平是正的，根据图猜测这种方案的带宽。

a．00000000　　b．11111111　　c．01010101　　d．00110011

P7-32　在 5B/6B 编码方案中，有多少无效的（不使用的）代码序列？

P7-33　使用B8ZS扰乱技术对序列 111000000000000 进行扰动，会引起什么结果？假定最近非零信号电平是正的。

P7-34　对下列每个信号奈奎斯特采样速率是多少？

a．带宽为 200kHz 的低通信号　　b．带宽为 200kHz，最低频率是 100kHz 的带通信号

P7-35　对带宽 200kHz 的低通信号采用 1024 级采样：

a．计算数字化信号的速率　　b．计算这个信号的 SNR_{dB}　　c．计算这个信号 PCM 的带宽

P7-36　如果使用 4 级数字信号，带宽为 200kHz 的通道最大速率是多少？

P7-37　一个模拟信号带宽为 4kHz。如果采样该信号并在 30kbps 的通道上发送，问 SNR_{dB} 是多少？

P7-38　带宽为 1MHz 的基带通道，按下列线路编码：

a．NRZ-L　　b．曼彻斯特

问该通道数据速率是多少？

P7-39 已给定比特率和调制类型，试计算波特率：

a. 200bps，FSK　　b. 4000bps，ASK　　c. 36000bps，64-QAM

P7-40 已给定波特率和调制类型，试计算比特率：

a. 1000 波特，FSK　　b. 1000 波特，ASK　　c. 1000 波特，16-QAM

P7-41 下列技术每波特有多少位？

a. 具有 8 个不同频率的 FSK　　b. 128 点星座图的 QAM

P7-42 试画出下列情况的星座图

a. 峰值振幅为 1 和 3 的 ASK 调制

b. 具有两个分别为 1 和 3 的峰值振幅以及 4 个相位的 8-QAM

P7-43 如果星座图具有下列数据点的个数，试问每个波特发送多少位？

a. 2　　b. 4　　c. 16　　d. 1024

P7-44 如果发送速率为 4000bps，试问下列情况要求带宽是多少？设 $d=1$。

a. ASK　　b. FSK（$2\Delta f = 4$ kHz）　　c. 16-QAM

P7-45 某公司拥有 1MHz 带宽（低通）的介质，需要建立 10 个独立的通道，每个通道至少能发送 10Mbps 的能力，决定用 QAM 调制技术。试问对每个通道，每波特最小位的个数是多少？每个通道星座图点数是多少？设 $d=0$。

P7-46 如果需要调制带宽为 5kHz 的语音信号，分别对下列情形求出其带宽：

a. AM　　b. FM（$\beta = 5$）　　c. PM（$\beta = 1$）

P7-47 关于奈奎斯特定理的一个例子，以三种不同的取样率对一个简单的正弦波进行取样：$f_s = 4f$（奈奎斯特速率的 2 倍），$f_s = 2f$（奈奎斯特速率）和 $f_s = f$（奈奎斯特速率的一半）。说明如何恢复此波？

P7-48 假定语音通道带宽是 4kHz，需要用 FDM 多路复用 10 个语音通道，而防护频带为 500Hz，试计算所需要的带宽。

P7-49 需要用一个 20kHz 带通通道传输 100 个数字语音通道，假定没有防护频带，每赫兹位的比率应该是多少？

P7-50 有 20 个数字信号源使用同步 TDM 实现多路复用，每个信号源的速率是 100kbps，每个输出时隙携带来自每个数字信号源的 1 位，但帧需要增加 1 位用于同步。回答下面的问题：

a. 以位为单位的输出帧的长度是多少？　　b. 输出帧速率是多少？

c. 输出帧持续时间是多长？　　d. 输出速率是多少？

e. 系统效率是多少（有用位与所有位之比）？

P7-51 有 14 个数字信号源，每个信号源每秒生成 500 个字符（8 位）。因为当前只有一些数字信号源是活跃的，所以使用字符交替的统计 TDM 实现多路复用。每帧每次携带 6 个时隙，但帧对每个时隙需要增加 4 位地址。试回答下列问题：

a. 以位为单位的输出帧的长度是多少？　　b. 输出帧速率是多少？

c. 输出帧持续时间是多长？　　d. 输出速率是多少？

P7-52 使用同步 TDM 实现多路复用，其 4 个信号源按下列发送字符（注意：字符按它们键入的顺序发送，第二个源端不发送数据）。试表示出 5 个输出帧的内容：

a. 信号源 1 信息：HELLO　b. 信号源 2 信息：HI　　c. 信号源 3 信息：　d. 信号源 4 信息：BYE

P7-53 图 7-68 表了使用同步 TDM 的一个多路复用器。如果输出时帧长度只有 10 位（每个输入取 3 位加上 1 个帧指示位），那么输出流是什么？到达多路复用器的位按照箭头所示。



图 7-68　思考题 P7-53

P7-54 某个 FHSS 系统使用 4 位 PN 序列。如果 PN 的比特率是 64bps，试回答下列问题：

a. 可能的跳的总数是什么？

b. 一个完整的 PN 循环需要多长时间？

P7-55 一个数据速率为 10Mbps 的数字介质，如果用巴克序列的 DSSS，该介质能携带多少个 64kbps 的语音通道？

P7-56 假设一个无噪声信号的能量是 10mW，这个信号经过扩大器的 5 个阶段，每个阶段将信号的能量扩大为原来的 2 倍，但同时也增加了 10μw 的噪声信号。当信号离开最后一个阶段时，SNR_{dB} 是多少？

P7-57 每个样本 8 位，数字转换人类语音时 SNR_{dB} 是多少？

P7-58 有 5kHz 基带通带，需要使用这个通道以 40kbps 的速率发送数据。

a. 可能的级的总数是什么？

b. 应该如何提高信道（SNR）质量来实现这个速率？

P7-59 STP 电缆在频率为 10kHz 时每 1km 损失 1dB。我们想要在一个 10 公里的链路上使用该电缆。如果要求信号在目的端有 10mW 能量，那么源端信号的能量应该是多少？

第 8 章

Computer Networks: A Top-Down Approach

多媒体和服务质量

多媒体涉及很多不同的离散媒体，例如文本、图像、音频和视频，它们以数字形式产生、存储和传输，并能够交互式的被访问。现在的多媒体是一个广泛的主题，甚至无法在一个章节充分地讨论。在本章中，我们只对多媒体做一个概述，涉及的内容直接或间接与多媒体有关，比如压缩或服务质量。

- 8.1 节讨论压缩的基本概念。尽管压缩与多媒体这个主题不直接相关，但是如果不压缩数据而直接传输多媒体则是不可能的。
- 8.2 节讨论多媒体的元素：文本、图像、视频和音频。说明了如何表示、编码和压缩（用到的技术在 8.1 节讨论过）这些元素的。
- 8.3 节讨论多媒体在因特网中的分类，它分为流式存储音频/视频、流式实况音频/视频和实时交互式音频/视频。简要描述了各自的特点和功能，并给出一些例子。
- 8.4 节主要讨论实时交互式的多媒体。介绍了用来处理这类型通信的两种协议，SIP 和 H.323。这两种协议应用在 IP 语音，也可以用在未来应用的通信协议中。通过讨论用于多媒体应用的传输层协议，总结出传统的传输层协议（UDP 和 TCP）不完全适用于多媒体。我们介绍了一种新的传输层协议 RTP，以及其控制元件 RTCP。还讨论了一种新的传输层协议 SCTP，这个协议在第 3 章提到过。
- 8.5 节讨论服务质量。多媒体通信与单独使用文本的通信相比，更需要服务质量。这一节只介绍了一个非常有趣而且有争议的主题。

8.1 压缩

我们在这节讨论压缩。由于多媒体通信中存在大量的数据交换，所以压缩起着一个至关重要的作用。通过压缩使交换的数据量减少。压缩分为两大类：无损压缩（loss compression）和有损压缩（lossy compression）。我们简单地讨论下两类中共同的技术。这节为不熟悉压缩技术的读者讲述必要的背景知识。如果读者熟悉压缩技术，这节可以跳过。

8.1.1 无损压缩

无损压缩（lossless compression）可以保存完整的数据，因为压缩算法和解压缩算法相互间是完全相反的：在压缩或解压缩的过程中没有数据丢失。无损压缩技术通常在不允许数据丢失的情况下使用。例如，在压缩一个文本文件或应用程序时不允许丢失数据。为了更进一步地减少数据量，无损压缩也应用在一些有损压缩过程的最后一步。

在这节中讨论四种无损压缩的技术：行程长度编码、词典编码、哈夫曼编码和算术编码。

行程长度编码

行程长度编码（run-length coding）有时也称为行程编码（run-length encoding，RLE），是最简单的消除冗余的技术。它能够将数据压缩为一些符号的组合。这种技术把重复的数据单元替换成两种字符体：字符重复出现的次数和字符本身。例如，下面说明了如何将 17 字节的字符串压缩成 10 字节的字符串。

AAABBBBCDDDDDDEEE　→　3A4B1C6D3E

这种技术的一种改进版本可以用在只包含两种字符的模式。如一个由 0 和 1 组成二进制数据。在这种情况下，我们只用记录发生在每两个字符之间的另一个字符的数量。图 8-1 说明了一个二进制模式，其中 0 比 1 多。我们只是显示在 1 和 1 之间的 0 的数量。

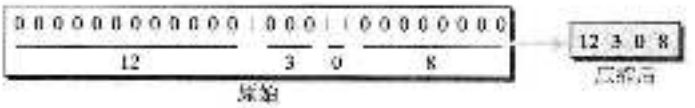


图 8-1　压缩二进制模式的行程长度编码版本

通过固定的位数表示每个数字，将压缩数据编码为二进制模式。假设每个数字用 4 位表示，压缩数据可以编码为 1100 0011 0000 1000，这个例子中压缩比是 26/16，接近 1.62。

词典编码

有一组压缩技术是基于创建词典用来转换文本中的字符串。这个概念是对共同的字符序列进行编码，而不是对每个字符分别进行编码。报文被扫描时创建词典，如果报文中发现字符序列是字典的入口，那么发送该入口代码（索引）而不是发送这个字符序列。我们在此讨论的词典编码是由 Lempel 和 Ziv 发明并由 Welch 将此技术实用化，称为 Lempel-Ziv-Welch 技术（LZW）。有趣的一点是，这种编码技术动态创建词典，这个词典是由发送端和接收端在编码和解码过程中创建的，而且它不被发送端发送到接收端。

编码

以下是编码的过程：

1. 词典被初始化为报文（字母表）中每个可能字符的条目（entry）。与此同时，我们称为字符串的缓存区被初始化为报文中的第一个字符。字符串保存了到目前为止发现最大的可编程序列。在初始化阶段，只有报文的第一个字符是可编程的。

2. 流程扫描报文并且获得报文中的下一个字符。

a. 如果串联在一起的字符串以及被扫描的字符存在于词典中，那么这个字符串不是最大的可编程序列。流程将两者连接起来之后更新字符串，然后等待下一个循环。

b. 如果串联在一起的字符串以及被扫描的字符不在词典中，那么最大可编程序列就是这个字符串，而不是两者串联起来的字符串。有三种情况：第一，流程把两者串联起来作为词典新的条目；第二，流程将字符串编码；第三，用扫描到的字符为下一个循环重新设置字符串。

3. 当报文中有更多的字符时，重复第二个步骤。

表 8-1 给出了编码过程的伪代码。为简单起见，我们把下一个字符和字符串都用 S 表示。

表 8-1　LZW 编码

```
LZWEncoding (message)
{
    Initialize (Dictionary)
    Char = Input (first character)
    S = char                                    // S 是可编程序列
    while (more characters in message)
    {
        char = Input (next character);
        if ((S + char) is in Dictionary)        // S 不是可编程序列
        {
            S = S + char;
```

```
        }
        else                                    // S 上已经有序列
        {
        addToDictionary (S + char);
        Output (index of S in Dictionary);
        S = char;
    }
  }
  Output (index of S in Dictionary);
}
```

例 8.1　我们给出一个关于 LZW 编码的例子，它使用文本报文，其字母表由两个字符（A 和 B）组成（见图 8-2）。

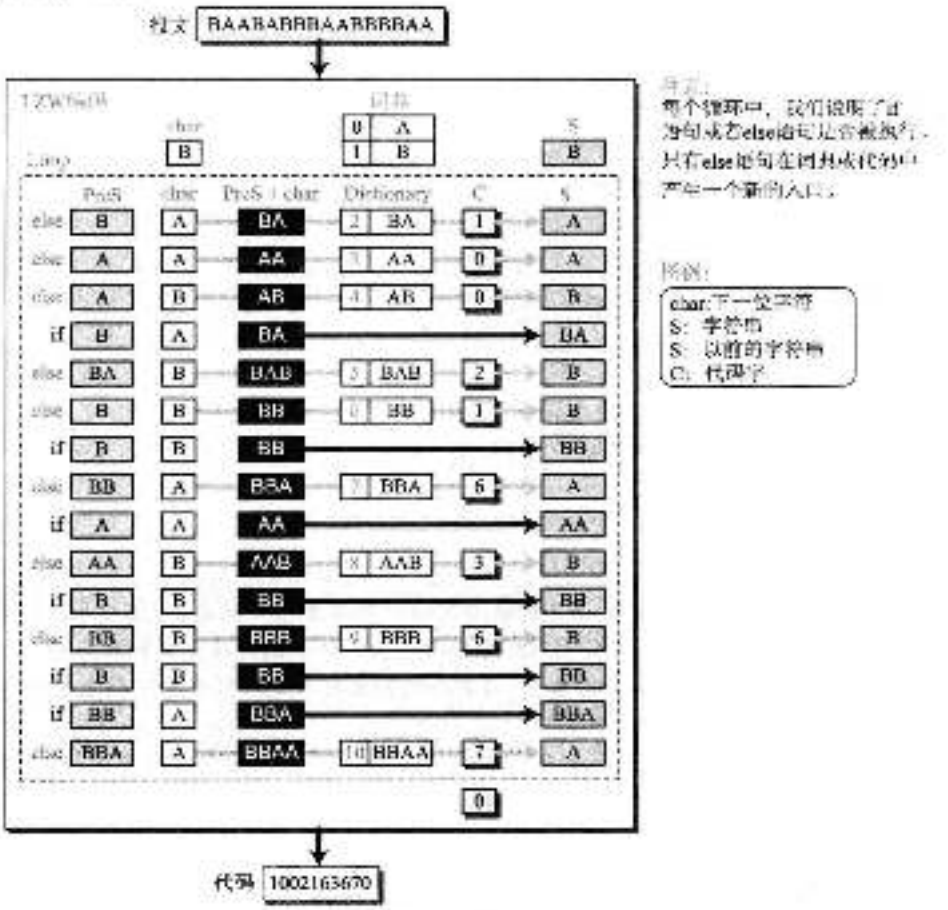


图 8-2　例 8.1

图中显示了文本“BAABABBBAABBBBAA”如何编码为 1002163670。注意缓冲区的 PreS 中保存的是更新之前的来自前一个迭代的字符串。

解码

以下是解码的过程：

1. 在编码的声明过程中初始化词典。扫描第一个码字，并使用字典，输出报文中的第一个字符。
2. 然后这个过程中创建一个字符串，并将其设置到之前扫描的码字中。现在它扫描一个新的

码字。

a. 如果码字在词典中，那么在这个过程中字典增加一个新的条目，这是连接来自与新的码字有关的条目中的第一个字符的字符串。它还输出与新的条目相关的码字。

b. 如果码字不在词典中（偶尔可能发生），这个过程中字符与字符串中的第一个字符连接，并将其存储在词典里。仍然输出连接的结果。

3. 当代码中有更多的码字出现时便重复第 2 步。

表 8-2 显示了 LZW 解码的简化算法。我们用于 C 表示码字，用 S 表示字符串。

表 8-2　LZW 解码

```
LZWDecoding (code)
{
    Initialize (Dictionary);
    C = Input (first codeword);
    Output (Dictionary [C]);
    while (more codewords in code)
    {
        S = Dictionary[C];
        C = Input (next codeword);
        if (C is in Dictionary)                    // 正常情况
        {
            addToDictionary (S + firstSymbolOf Dictionary[C]);
            Output (Dictionary [C]);
        }
        else                                       // 特殊情况
        {
            addToDictionary (S + firstSymbolOf (S));
            Output (S + firstSymbolOf (S));

        }
    }
}
```

例 8.2　让我们说明例 8.1 中的代码如何被解码和如何恢复原始报文的（见图 8-3），所用的 PreC 保存的是前一次迭代的码字，这个在伪代码中是不需要的，但在这里需要更好地显示过程。请注意在这个例子中只有特殊情况码字不在词典中。词典中的新条目需要字符串和字符串中的第一个字符合并。输出也和新条目相同。

哈夫曼编码

当我们以二进制模式对数据进行编码时，我们通常为每个符号位设定一个固定数量的比特位。为了压缩数据，我们可以考虑报文中符号的频率和它们出现的概率。哈夫曼编码（Huffman coding）为那些频率发生频繁的符号分配较短的代码，为那些频率发生较少的符号分配较长的代码。例如，假设我们有一个文本文件，只使用五个字符（A、B、C、D、E），发生的频率分别为（20、10、10、30、30）。

哈夫曼树

要使用哈夫曼编码，我们首先需要建立哈夫曼树。在哈夫曼树中，树的叶子代表符号。这样设计使最频繁的符号最接近的树的根（相对于根使用最少数量的结点），出现次数最少的符号距树的根距离最远。图 8-4 显示了这个过程。

1. 我们把整个字符集排列成一个序列。现在每个字符都是树的最底层的一个结点。

2. 我们选择出现最少频率的两个结点，把它们连接起来形成一个新的结点，形成一个简单的

两层的树。新结点的频率是原有的两个结点的频率的结合。这个结点，位于叶子结点上层，有结合其他结点的资格。

3. 我们重复第 2 步，直到所有的结点合并成一个单一的树。

4. 树建立之后，我们为每个分支分配位值。由于哈夫曼树是一个二叉树，每个结点最多有两个子结点。

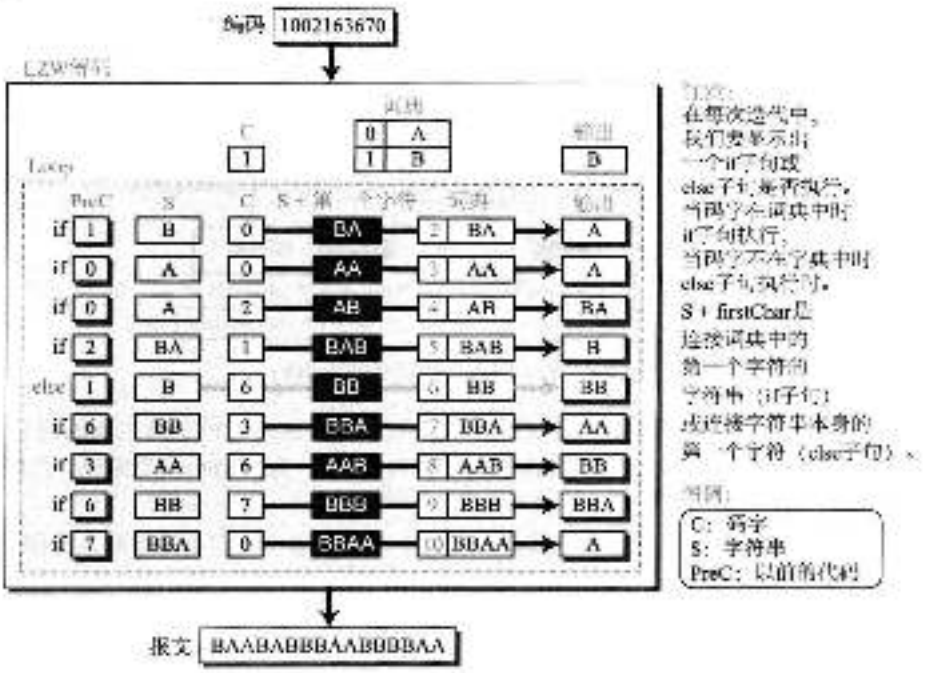


图 8-3　例 8.2

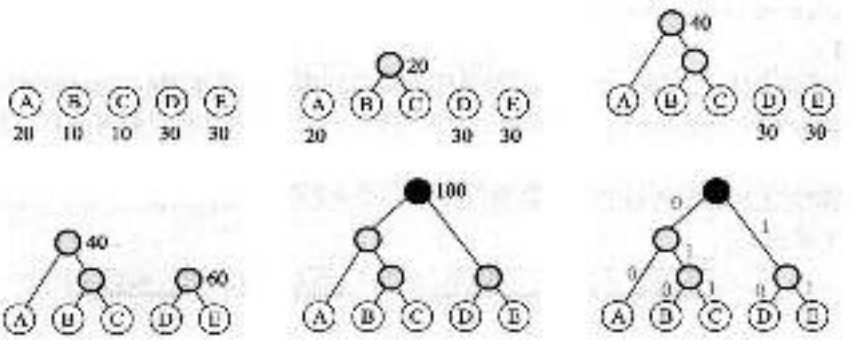


图 8-4　哈夫曼树

编码表

树建立之后，我们可以创建一个表，用来显示如何对每个字符进行编码和解码。可以从树的根沿着通向字符的分支得到每个字符的编码。代码本身是依次取得路径上的每个分支的位值。我们举个简单的例子如表 8-3 显示的字符代码。

关于编码注意以下几点：首先，出现频率较高的字符（A、D 和 E），比出现频率较低的字符（B 和 C）接收一个较短的编码，然后为相同频率的每个字符分配相同位的编码。其次，在这种编码系统没有任何代码是另一个代码的前缀。2 位代码，00、10 和 11 不是任何其他两个代码（010

和 011）的前缀。换言之，我们没有以 00，10 或 11 开头的 3 位编码。此属性使哈夫曼编码可以即时编码（instantaneous code）。我们将在下一节解释此属性。

表 8-3　编码表

符　号	编　码	符　号	编　码	符　号	编　码
A	00	C	011	E	11
B	010	D	10		

编码和解码

图 8-5 说明了我们如何使用哈夫曼编码进行编码和解码。

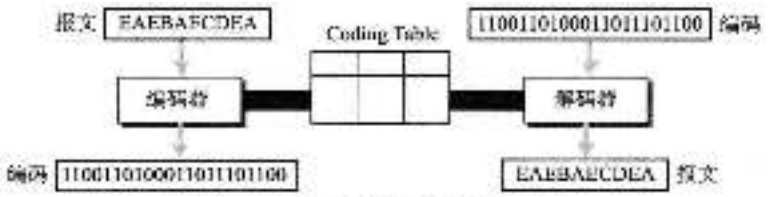


图 8-5　哈夫曼编码中的编码和解码

注意，我们已经完成了一个小报文的压缩。如果我们想发送由五种字母组成的固定长度编码，每个字符需要 $\log_2 5 = 2.32$ 或 3 位，或者说整个报文需要 30 位，而使用哈夫曼编码，我们只需要 22 位。压缩比是 30/22 或 1.36。

在哈夫曼编码中，没有代码是另一个代码的前缀。这意味着，我们不需要插入分隔符分隔一个字符编码和下一个字符编码。哈夫曼编码还有即时解码的属性：当解码器有两个位 00，它可以立即解码为一个字符，而不需要看到更多的数据位。

哈夫曼编码的一个缺点是，编码器和解码器都需要使用相同的编码表。换言之，就像词典在 LZW 编码中一样，哈夫曼树不能被动态创建。但是，如果编码器和解码器一直使用的是相同的符号集，树可以被创建和共享一次。否则，表需要由编码器建立，并给予接收器。

算术编码

在以前的压缩方法中，每一个符号或符号序列被单独编码。**算术编码**（arithmetic coding），在 1981 年由 Rissanen 和 Langdon 引入将整个报文映射到一个较小的时间间隔内[0,1）。然后将这个小的时间间隔以二进制模式进行编码。算术编码建立在我们在半开区间[0,1）有大量小的时间间隔这样一个事实基础之上。其中的每个小间隔可以代表一个我们使用有限集的符号可以表示的报文。图 8-6 说明了这个思想。

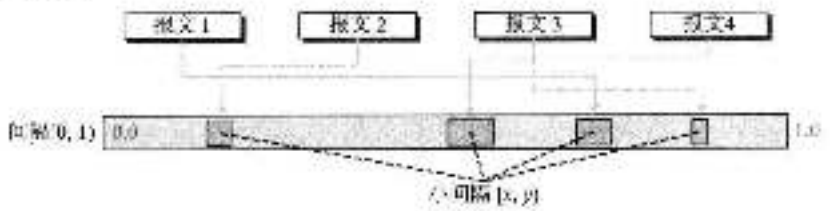


图 8-6　算术编码

编码

为了将一个报文以算术编码方式编码，我们首先需要分配每个符号发生的概率。如果我们在字母表中有 M 个符号（包括我们需要进行解码的终止符号，我们稍后将看到），概率 P_1、P_2、…、P_M，$P_1 + P_2 + \cdots + P_M = 1.0$。表 8-4 显示了这种编码算法。

在循环的每次迭代中，我们将现有的间隔划分成 M 子区间间隔，每个子区间的长度与相应的符号的概率成正比。这样做是为了均匀分散在区间[0,1）的报文。我们在每次迭代中保存新的时间间隔中的符号顺序。

表 8-4　算术编码

```
ArithmeticEncoding (message)
{
    currentInterval = [0,1);
    while (more symbols in the message)
    {
        s = Input (next symbol);
        divide currentInterval into subintervals
        subInt = subinterval related to s
        currentInterval = subInt
    }
    Output (bits related to the currentInterval)
}
```

选定位输出的选择取决于实现。一些实现使用位，其用来表示开始时间间隔内的小数部分。

例 8.3　为了简单起见，让我们假设，我们的符号集是 S={A，B，*}，其中*是终止符号。我们分配每个符号发生的概率如下：

$$P_A = 0.4 \qquad P_B = 0.5 \qquad P_* = 0.1$$

图 8-7 显示了我们如何找到短报文“BBAB*”中相关的间隔和代码。

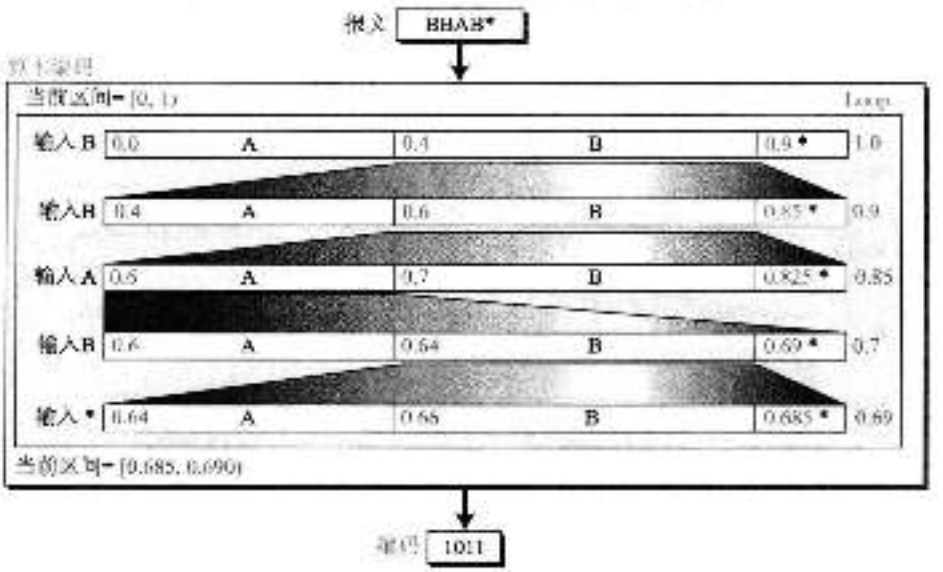


图 8-7　例 8.3

我们初始化当前间隔[0,1）。在循环的每次迭代中，我们根据每个符号发生的概率将当前间隔划分为三个子区间。然后我们读第一个符号，并选择相应的子区间。然后将当前间隔设置为所选间隔。重复这个过程，直到所有符号输入完成。读每一个符号，直到当前间隔减小到[0.685，0.690）。我们以二进制的 0.685 为编码下限，大约是 $(0.1011)_2$，但我们只保留小数部分 $(1011)_2$ 作为编码。注意，当我们将一个 0 和 1 之间实数转换成二进制时，可以得到一个无限的数字位。我们需要保持足够的位来恢复原始消息。超过足够的位不是有效编码，而少于足够的位可能会导致错误的解码。例

如，如果我们只用三个位（101）代表真正的值 0.625，它在最后间隔区间[0.685 0.690），之外。

解码

解码与编码类似，但当终止符号输出时退出循环。这就在原始报文中我们需要终止符号的原因。表 8-5 显示了解码算法。

表 8-5　算术解码

```
ArithmeticDecoding (code)
{
    c = Input (code)
    num = find real number related to code
    currentInterval = [0,1);
    while (true)
    {
        divide the currentInterval into subintervals;
        subInt = subinterval related to num;
        Output (symbol related to subInt);
        if (symbol is the terminating symbol)   return;
        currentInterval = subInt;
    }
}
```

例 8.4　图 8-8 显示了在例 8.3 中我们是如何使用解码进行解码报文的。注意，手指的地方说明了在相应的时间间隔里数字的位置。

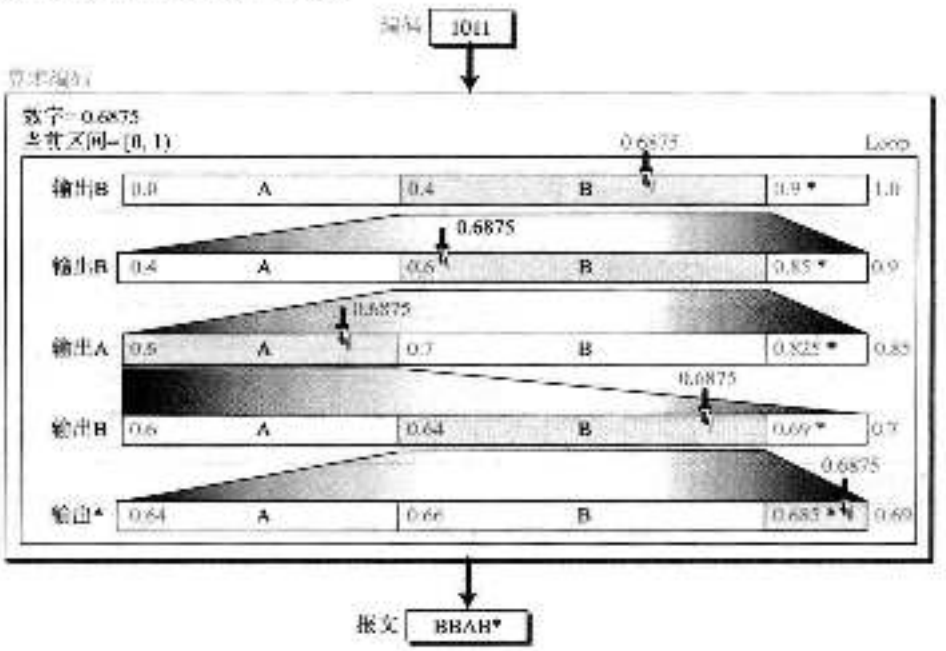


图 8-8　例 8.4

静态与动态的算术编码

介绍算术编码的两个版本：静态编码（有时也称为纯编码）和动态编码（有时也称为间隔编码）。我们在本节讨论的版本是第一种，静态编码。静态算术编码存在两个问题。首先，如果当前的间隔非常小，我们需要一个精度非常高的算术对报文进行编码，这导致在代码中出现很多 0。其次，在

报文的所有符号全部输入之后才可以进行编码，这就是我们在解码时需要一个终止符号的原因。新版本，即动态算术编码，通过在读取每一个符号之后立即输出二进制位的程序解决了这两个问题。

8.1.2　有损压缩

无损压缩有压缩量的限制。但是在某些情况下，我们可以牺牲一些准确性，以提高压缩率。尽管我们不能接受在文本压缩时丢失信息，但是可以接受在压缩图像、视频和音频时丢失部分信息。例如，人类视觉无法察觉由于图像有损压缩导致的一些小扭曲。在这节中，我们讨论有损压缩中的一些想法。在下一节中，我们说明在图像、视频和音频压缩过程中这些想法是如何实现的。

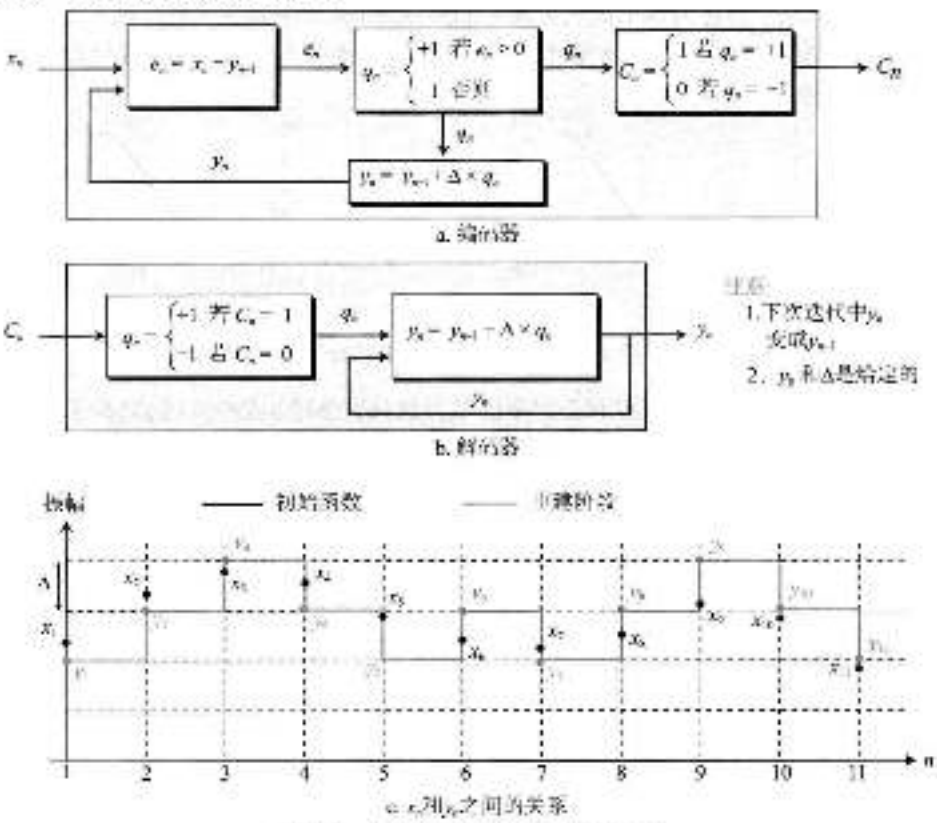
预知编码

我们数字化一个模拟信号时使用预知编码。在第 7 章，我们讨论了使用采样把模拟信号转换成数字信号的一种技术，脉冲编码调制（PCM）。取样后，每个样品需要量化创建二进制值。可以在量化阶段使用预知编码实现压缩。

在 PCM 机制中，样本分别被量化。然而，邻近的量化样本有密切的关系，并有相似的值。在预知编码（predictive coding）中，我们就使用这种相似性。不是将每个样品分别量化，而只是将不同的样品进行量化。由于不同的样品数量比实际样品的数量少，因此需要较少的位数。许多算法都是基于这个原则的。我们先从最简单的开始然后过渡到更复杂的。

delta 调制

在预知编码中最简单的方法称为 delta 调制（delta modulation）。让 x_n 代表采样间隔 n 的初始函数值，y_n 是 x_n 的重建值。图 8-9 显示了在增量调制中编码和解码过程。在 PCM 机制中发送编量化样本（x_n），并将它们传送到接收端。



c. x_n和y_n之间的关系

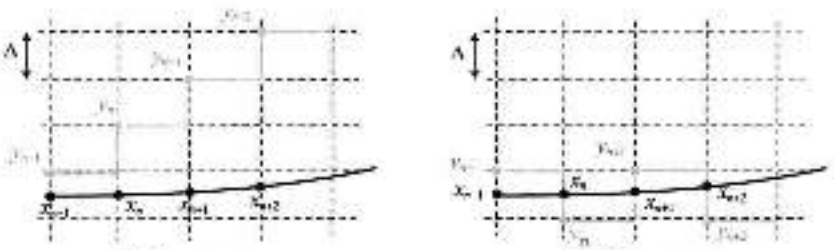
图 8-9　delta 调制中编码和解码过程

在 delta 调制中，发送者量化 e_n，e_n是每个样品（x_n）和前面的重建值（y_{n-1}）的差。

然后发送端发送 C_n。接收端将收到的 C_n重建成样品值 y_n。

注意，每个样品 PCM 需要传输多个位。例如，如果最大的量化值是 7（见第 7 章），那么它需要传输每个样品 3 位。由于它为每个样品传送一个比特（1 或 0），所以 DM 减少了传输的比特数。

我们可能会有疑问，为什么 DM 量化 $x_n - y_{n-1}$ 而不是 $x_n - x_{n-1}$。原因是如果 x 是一个缓慢变化的函数，第二种选择使得 y 的值变化比 x 快得多。量化 $x_n - y_{n-1}$ 是为了自动调整增长缓慢或缓慢下降的 x。图 8-10 比较了 $x_n - x_{n-1}$ 与 $x_n - y_{n-1}$ 缓慢增长函数的量化重建梯度。（对于一个缓慢下降的函数，这个概念是类似的。）



a. 量化 $q_n = x_n - x_{n-1}$　　b. 量化 $q_n = x_n - y_{n-1}$

图 8-10　对比 $x_n - x_{n-1}$ 与 $x_n - y_{n-1}$ 的量化重建

自适应 DM（ADM）

图 8-11 说明了 delta 调制中量化器Δ的作用。在这个范围内，Δ小于初始函数中的斜率，重建梯度不如初始函数，其结果是称为斜率过载失真（slope overload distortion）的错误。另一方面，在这个范围内Δ比初始函数的斜率大，重建梯度在初始函数周围连续大幅度振荡，并导致称为颗粒状噪音（granular noise）的错误。



a. 斜率过载　　b. 颗粒状噪音

图 8-11　斜率过载和颗粒状噪音

由于大部分函数都有大斜率区域和小斜率区域，选择大值的Δ或小值的Δ减少了错误的一个类型，但也增加了错误的另一个类型。自适应 DM（adaptive DM，ADM）是用来解决这个问题的。在 ADM 中，Δ值从一步跳到下一步并且按如下所示计算：

$$\Delta_n = M_n \Delta_{n-1}$$

其中 M_n是步长乘数（step-size multiplier）。它由前几位的 q_n值计算出来。有许多不同的计算 M_n的算法。一个简单的算法是如果 q_n保持不变则按一定比例增加 M_n，如果 q_n变化则按一定比例减少 M_n。通过延迟编码过程进一步改善自适应过程，延迟过程用来在计算 M_n过程中包含一些未来样品的信息。

差分 PCM（DPCM）

差分 PCM（differential PCM，DPCM）是 delta 调制的推广。在 delta 调制中一个预先重建样值 y_{n-1}称为预测值（predictor），因为它用于预测当前值。在 DPCM 中，不止一个预先重建样值用于预测。在这种情况下，差分被认为如下所示：

$$e_n = x_n - \sum_{i=1}^{N} a_i y_i - 1$$

和是预测值，a_i是预测值的系数（或权值），N是预测序号。对于 DM 来说，预测序号是 1，a_i=1。差分像 DM 中那样被量化并发送到接收端。接收端重建当前值，如下所示。

$$y_n = \sum_{i=1}^{N} a_i y_{i-1} + \Delta q_n$$

通过最小化预测值和实际值之间的累计误并建立预测值系数，优化使用方差方法（method of square error），这超出了本书的范围。

自适应 DPCM（ADPCM）

进一步压缩可以通过对不同区域的样品使用不同系数，或通过一步步调整量化器（Δ），或二者结合实现。这是自适应 DPCM（adaptive DPCM，ADPCM）的原则。

线性预知编码

线性预知编码（linear predictive coding，LPC），不是发送量化的差异信号，而是分析信号并确定其特征。其特性包括在敏感频率范围的频率，每个频率的功率、每个信号的持续时间。源端量化信息并把它传送到接收端。接收端把信息送入信号合成器来将信号模拟成与原信号类似的信号。LPC 可以实现高度压缩。但是，这种方法通常用于军事上压缩语音。在这种情况下合成的语音虽然是能理解的，但是缺乏鉴定说话者的自然性和音质。

变换编码

在变换编码中，输入信号应用数学转换产生输出信号。这个转换需要是可逆的，允许恢复原信号。这个转换从一个域到另一个域（例如时域到频域）改变信号表示法，结果导致编码数位减少。

我们需要强调的是，在多媒体使用的转换技术本身是无损的。然而，为了完成压缩目标，另一个步骤量化被添加到操作中，这使得整个过程有损。

离散余弦变换（DCT）

在多媒体使用的流行的变换之一称为**离散余弦变换**（discrete cosine transform，DCT）。尽管我们在多媒体压缩中使用二维 DCT，但是为了方便理解我们首先要讨论一维 DCT。

一维 DCT　在一维 DCT 中，变换是方形矩阵 T（DCT 系数）和列矩阵 p（源数据）得到的乘法矩阵。其结果是列矩阵 M（转换后的数据）。由于代表 DCT 系数的方阵是一个正交矩阵（逆矩阵和转置矩阵是相同的），逆变换可以通过方形矩阵 T（DCT 系数）和列矩阵 p（源数据）得到的乘法矩阵得到。图 8-12 显示了矩阵转换，其中 N 是矩阵 T 的大小，T^T 是 T 的转置矩阵。

$$\begin{bmatrix} M \end{bmatrix} = \begin{bmatrix} & T & \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \qquad \begin{bmatrix} p \end{bmatrix} = \begin{bmatrix} & T^T & \end{bmatrix} \times \begin{bmatrix} M \end{bmatrix}$$

a. 变换　　b. 逆变换

$$T(m,n) = C(m)\cos\left[\frac{\pi n(2m+1)}{2N}\right] \qquad C(m) = \begin{cases} \sqrt{\dfrac{1}{N}} & m = 0 \\ \sqrt{\dfrac{2}{N}} & m > 0 \end{cases}$$

其中，$m = 0 \sim N-1$

其中，$n = 0 \sim N-1$

图 8-12　一维 DCT

我们相信变换用矩阵表示法更容易理解，也可以使用两个公式做到，如图 8-13 所示。

例 8.5　图 8-14 显示为 $N=4$ 的变换矩阵。如图所示，第一行有四个相等的值，其他行有交替的正负值。当源数据矩阵乘以每一行数据时，如果源数据项是彼此接近的我们预期正值和负值导致最终值接近 0。这是我们从变换中期望得到的；这表明只有源数据中的一些值是重要的而大多数值是多余的。

$$M(m) = \sum_{n=0}^{N-1} C(n) \cos\left[\pi n(2m+1)/(2N)\right] \times p(n) \qquad \text{其中，} m = 0, \ldots, N-1$$

$$p(n) = \sum_{m=0}^{N-1} C(n) \cos\left[\pi n(2m+1)/(2N)\right] \times M(m) \qquad \text{其中，} n = 0, \ldots, N-1$$

$$C(i) = \begin{cases} \sqrt{\frac{1}{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & i > 0 \end{cases}$$

图 8-13　一维前向和逆变换公式

$$\begin{bmatrix} 203 \\ -2.22 \\ 0.00 \\ -0.16 \end{bmatrix} = \begin{bmatrix} 0.50 & 0.50 & 0.50 & 0.50 \\ 0.65 & 0.27 & -0.27 & -0.65 \\ 0.50 & -0.50 & -0.50 & 0.50 \\ 0.27 & -0.65 & 0.65 & -0.27 \end{bmatrix} \times \begin{bmatrix} 100 \\ 101 \\ 102 \\ 103 \end{bmatrix}$$

M　　T　　p

a. 变换

$$\begin{bmatrix} 100 \\ 101 \\ 102 \\ 103 \end{bmatrix} = \begin{bmatrix} 0.50 & 0.65 & 0.50 & 0.27 \\ 0.50 & 0.27 & -0.50 & -0.65 \\ 0.50 & -0.27 & -0.50 & 0.65 \\ 0.50 & -0.65 & 0.50 & -0.27 \end{bmatrix} \times \begin{bmatrix} 203 \\ -2.22 \\ 0.00 \\ -0.16 \end{bmatrix}$$

p　　T^{T}　　M

b. 逆变换

图 8-14　例 8.5

这个例子说明了我们如何变换数字序列，从（100，101，102，103）到另一个数字序列（203，-2.22，0.00，-0.16）。为了更好地解释关于 DCT 变换的属性，我们有几点要提：首先，变换是可逆的。其次，变换矩阵是正交（$T^{-1}=T^{\mathrm{T}}$）的，这意味着我们计算逆转换时不需要使用逆矩阵，可以使用转置矩阵（更快的计算）。第三，M 矩阵的第一行始终是 p 矩阵的加权平均。第四，矩阵中的其他行的值是非常小的值（正或负），在这种情况下可以忽略不计。关于这三个值很重要的一点是，如果我们改变 p 矩阵的四个值为同样的值，但保持它们之间相同的相关性。如果我们更改源数据 $p=(7, 8, 9, 10)$，可以证明转换后的数据是：$M=(17、-2.22、0.00、-0.16)$；因为平均值已更改，第一个值将发生变化。因为数据项之间的关系并未改变，其余的值没有改变。这是我们期望从变换中得到的结果。它删除了冗余数据。p 矩阵中的最后三个值是冗余的，它们与第一个值有非常密切的关系。

二维 DCT　二维 DCT 是我们压缩图像、音频和视频时需要的。除源数据和转换后的数据是二维的方阵之外，原理与一维 DCT 是一样的。为了实现与一维 DCT 提到的属性相同的变换，我们需要使用 T 矩阵两次（T 和 T^{T}）。逆变换也使用两次 T 矩阵，但以相反的顺序。图 8-15 显示以二维 DCT 的格式矩阵。图 8-16 使用两个公式显示了同样的想法。

$$\begin{bmatrix} M \end{bmatrix} = \begin{bmatrix} T \end{bmatrix} \times \begin{bmatrix} p \end{bmatrix} \times \begin{bmatrix} T^{\mathrm{T}} \end{bmatrix} \qquad T(m,n) = C(m,n)\cos\left[\frac{m\pi(2n-1)}{2N}\right]$$

其中，$m = 0 \sim N-1$

其中，$n = 0 \sim N-1$

a. 变换

$$\begin{bmatrix} p \end{bmatrix} = \begin{bmatrix} T^{\mathrm{T}} \end{bmatrix} \times \begin{bmatrix} M \end{bmatrix} \times \begin{bmatrix} T \end{bmatrix} \qquad C(m,n) = \begin{cases} \sqrt{\frac{1}{N}} & m = 0 \\ \sqrt{\frac{2}{N}} & m > 0 \end{cases}$$

b. 逆变换

图 8-15　二维 DCT

$$M(m,n) = \frac{2}{N} C(m)\,C(n) \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} p(k,l) \cos\left[\frac{m\pi(2k+1)}{2N}\right] \cos\left[\frac{n\pi(2l+1)}{2N}\right] \qquad \text{其中 } m = 0, \ldots, N-1 \quad \text{其中 } n = 0, \ldots, N-1$$

$$p(k,l) = \frac{2}{N} C(l)\,C(k) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} M(m,n) \cos\left[\frac{k\pi(2m+1)}{2N}\right] \cos\left[\frac{l\pi(2n+1)}{2N}\right] \qquad \text{其中 } k = 0, \ldots, N-1 \quad \text{其中 } l = 0, \ldots, N-1$$

$$C(u) = \begin{cases} \sqrt{\frac{1}{2}} & u = 0 \\ 1 & u > 0 \end{cases}$$

图 8-16　二维前向和逆变换公式

8.2　多媒体数据

目前，多媒体数据由文本、图像、视频和音频组成，但是多媒体的定义更改为包括未来的媒体类型。

8.2.1　文本

因特网中存储着大量的可以下载和使用的文本。经常在文本数据中以线性形式引用明文，非线性形式引用超文本。文本以字符集的形式存储在因特网中，如 Unicode，表示基本语言中的字符。为了存储大量的文本数据，文本可以使用我们之前讨论过的无损压缩技术压缩。注意，需要使用无损压缩是因为我们在解压时不允许丢失任何数据。

8.2.2　图像

按多媒体的说法，图像（经常称为静止影像）用来表示一张照片、一份传真或者电影中的一个独立画面。

数字图像

要使用一个图片，首先必须将其数字化。这种情况下的数字化是指把一个图片表示成为点的二维数组，这个点称为像素。每个像素可以表示成若干个比特位，称为位深度（bit depth）。如果是黑白图片（如传真页），位深度为 1，每个像素可以被表示为二进制数 0（黑色）或 1（白色）。如果是灰度图片，每个像素可以用 8 位的整数（256 级）来表示。如果是彩色图片，图片通常分成三个通道，每个通道代表红、绿、蓝（RGB）三原色之一。在这种情况下，位深度为 24（每种颜色 8 位）。而某些特殊的表现需要用单独的通道去表示背景，这种通道称为 alpha（α）通道。在黑白图片中，这导致产生了两种通道。在彩色图片中，这导致产生了五种通道。每个像素可以用 24 位（3×8 位）表示，红、绿、蓝（RGB）分别用 8 位表示。

显而易见，在因特网中传输图片，随着黑白图片到灰度图片再到彩色图片，传输信息的消耗是非常惊人的。这意味着我们需要压缩图片来节省传输时间。

例 8.6　下面说明了在传输速率为 100kbps 的情况下，传输一张 1280 像素 × 720 像素的图片所需要的时间。

a. 传输黑白图片，位深度为 1，我们需要：

$$\text{传输时间} = (1280 \times 720 \times 1) / 100\,000 \approx 9\text{s}$$

b. 传输灰度图片，位深度为 8，我们需要：

$$\text{传输时间} = (1280 \times 720 \times 8) / 100\,000 \approx 74\text{s}$$

c. 传输彩色图片，位深度为 24，我们需要：

$$\text{传输时间} = (1280 \times 700 \times 24) / 100\,000 \approx 215\text{s}$$

图像压缩：JPEG

对于图像压缩，包括无损压缩和有损压缩两种算法，在这节我们讨论叫做 JPEG 的有损压缩。**联合图像专家组**（Joint Photographic Experts Group，JPEG）标准提供的有损压缩在大多数压缩中使用。JPEG 标准不但适用于灰度图像，而且还适用于彩色图像。为了简单，我们只关注灰度图像，该方法可适用于彩色图像的每个阶段。在 JEGP 中，灰度图片被分割为 8×8 的像素块。压缩和解压都要分别经过三个步骤，如图 8-17 所示。

将图片分割为块的目的是为了减少计算量，因为如二维 DCT 中表示的，每一张图片的数据计算次数是单元数目的平方。

转换

JPEG 通常在压缩的第一阶段使用 DCT，而在解压的最后一阶段使用反 DCT。转换和反转换应用在 8×8 的块中。我们在前一节中讨论过 DCT。

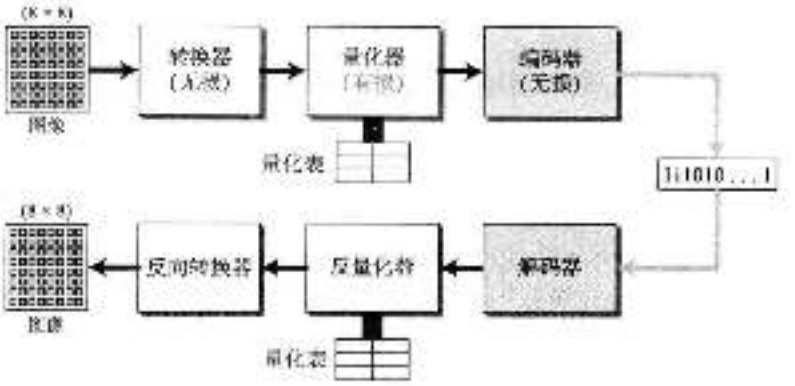


图 8-17　JPEG 的三个阶段

量化

DCT 转换的输出是一个实数矩阵。对于这些实数进行精确的编码需要大量的比特位。JPEG 在量化阶段不仅仅是四舍五入矩阵中的实数值，而且还将一些值置为零。为了完成高速率压缩，这些零在编码阶段被丢弃。我们之前讨论过，DCT 转换的结果定义了源矩阵中不同频率的权重。高频是指像素值发生突变。这些像素可以丢弃因为人的视力分辨不出这些突变。

量化阶段创建了一个新的矩阵，矩阵里的每个元素（$C(m, n)$）定义如下：

$$C(m, n) = \text{round}\,[M(m, n) / Q(m, n)]$$

其中 $M(m, n)$ 是转换矩阵的元素，而 $Q(m, n)$ 是量化矩阵的元素。四舍五入功能使实数值缩短为一个整数。这意味着值 3.7 会四舍五入为 4，而值 3.2 则四舍五入为 3。

JPEG 定义了 100 个量化矩阵，Q1 到 Q100。其中 Q1 代表最差的图像质量和最高的压缩比例，而 Q100 代表最高的图像质量和最低的压缩比例。这应由实现选择使用哪个矩阵。图 8-18 说明了其中的一些矩阵。

80	60	50	80	120	200	255	255
55	60	70	95	130	255	255	255
70	65	80	120	200	255	255	255
70	85	110	145	255	255	255	255
90	110	185	255	255	255	255	255
120	175	255	255	255	255	255	255
245	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Q10

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	110	103	99

Q50

3	2	2	3	5	8	10	12
2	2	3	4	5	12	12	11
3	3	3	5	8	11	14	11
3	3	4	6	10	17	16	12
4	4	7	11	14	22	21	15
5	7	11	13	16	12	23	18
10	13	16	17	21	24	24	21
14	18	19	20	22	20	20	20

Q90

图 8-18　三种不同的量化矩阵

注意，处理过程中唯一不可逆的阶段是量化阶段。在这里会丢弃某些信息，而且是不可恢复的。事实上，JPEG 称为有损压缩的唯一原因是因为存在该量化阶段。

编码

在量化后，按"之"字形序列对矩阵的值进行重排，之后将其输入到编码器中。进行量化值"之"字形排序的目的是使与低频相关的值在与高频相关的值之前进入编码器。由于绝大部分高频值为零，这意味着非零值在零值之前被提供给编码器。图 8-19 说明了这个过程。

这种情况下的编码是一种无损压缩。它或者使用了行程长度编码，或者使用了算术编码。

例 8.7　为了说明 JPEG 压缩的思想，我们有一个单一灰度的块，其中每个像素的位深度都是 20。我们已经用 Java 程序转换、量化并且把值重排成"之"字形。图 8-20 说明了这个编码过程。

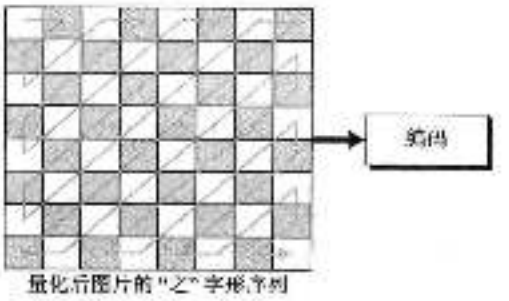


图 8-19　表的读取方式

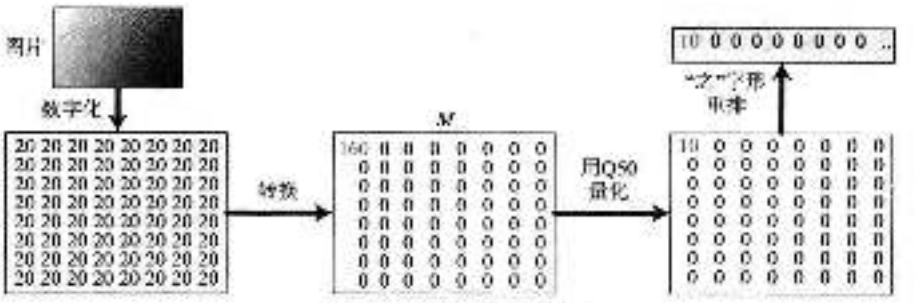


图 8-20　例 8.7 相同灰度

例 8.8　第二个例子，我们有一个逐渐改变的块，也就是说相邻像素间的值没有突变。如图 8-21 所示，我们已经获得许多零值，如图 8-21 所示。

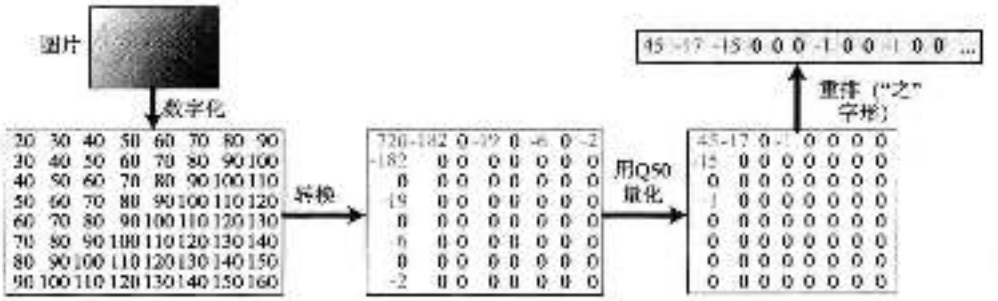


图 8-21　例 8.8：渐变灰度等级

图像压缩：GIF

JPEG 标准使用的图像中的每个像素表示为 24 个比特（每 8 位表示一种原色）。这意味着每个像素可以表示 2^{24}（16 777 216）种颜色。例如，洋红色是由红色和蓝色组成的（不包含绿色成分），可以表示为十六进制整型 FF00FF。

大多数简单的图像并不包含如此大范围的颜色。**可交换的图像文件**（Graphic Interchange Format，GIF）使用更小的调色板（索引表）颜色，通常有 2^8=256 种颜色。换言之，GIF 把真彩映射成调色板颜色。

例如，如果第 226 个是调色板颜色，则洋红色像素可以表示为十六进制整型 E2。这意味着，与 JPEG 相比 GIF 将图片大小减小到原来的 1/3。

为一个特定的图片创建调色板之后，每个像素可以表示为 256 个字符中的一个（例如，两位十

六进制数表示调色板索引）。现在为了更进一步压缩图像，我们可以使用无损压缩方法，如词典编码或者算术编码。

8.2.3　视频

视频由多帧构成，每个帧都是一个图片。这意味着视频文件需要一个快的传输速率。

数字化视频

视频是由一系列帧构成。如果帧在屏幕上显示足够快，我们就会有运动的感觉。原因是人的眼睛不能够分辨出快速闪过的单帧图像。每秒的帧数没有统一的标准。在北美，常见的是每秒 25 帧。但是为了避免闪烁的单帧图像，需要不断重复刷新帧。电视需要每一帧刷新两次，也就是说需要发送 50 帧。换言之，如果在发送端有存储器的话，发送的 25 帧中每一帧要从存储器中提取重画两次。

例 8.9　计算一些视频标准的传输速率：

a. 彩色广播电视提供的画面为每帧 720 × 480 个像素，每秒 30 个帧，每个颜色 24 位。不压缩时的传输速率如下：

$$720 \times 480 \times 30 \times 24 = 248\,832\,000 \text{ bps} = 249 \text{ Mbps}$$

b. 高清彩色广播电视提供的画面为每帧 1920 × 1080 个像素，每秒 30 个帧，每个颜色 24 位。不压缩时的传输速率如下：

$$1920 \times 1080 \times 30 \times 24 = 1\,492\,992\,000 \text{ bps} = 1.5 \text{ Gbps}$$

视频压缩：MPEG

运动图像专家组（Moving Picture Experts Group，MPEG）是一种用于压缩视频的技术。从原理上讲，运动图像是按顺序快速移动的一组帧，每一帧都是一幅图像。换言之，帧是像素的空间组合，而视频是顺序发送的帧的时间组合。压缩视频，其含义是在空间上压缩每一帧，而在时间上压缩一组帧。

空间压缩

每一帧的空间压缩（spatial compression）通过 JPEG 完成（或者 JPEG 的修改版）。每一帧是一幅图像，可以单独进行压缩。

时间压缩

在时间压缩（temporal compression）中会去掉冗余帧。我们看电视时，每秒接收 50 帧。但是大多数的连续帧几乎是相同的。例如，假设某个人在说话，除了帧中嘴唇部分外，大多数帧与前一帧是相同的。而嘴唇这一部分在每一帧中都可能会变化。

在时间域上压缩数据，MPEG 方法首先将帧分为三类：I-帧、P-帧和 B-帧。图 8-22 说明了一个帧的集合（图中是 7 个帧）是如何被压缩而生成其他的帧集合的。

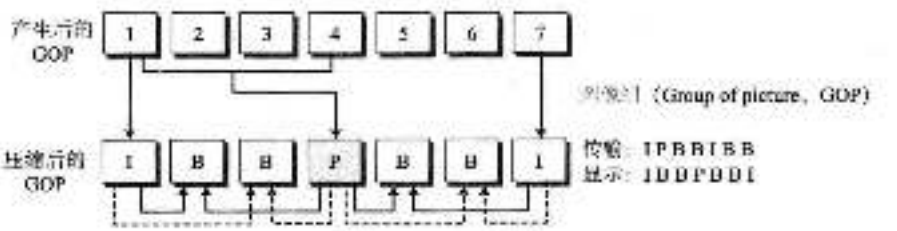


图 8-22　MPEG 帧

- I-帧。内部编码帧（intracoded frame，I-frame）是一个独立帧，与其他任何帧没有关系（与相邻前后发送的帧都没有关系）。它们在规定的时间间隔内出现。I-帧必须定期出现，用于处理帧中的突然变化。由于这种突然变化，使得前后帧无法显示。另外，在视频播出时，

观看者可能会随时间调整它的接收机。如果只在广播的开始时有一个 I-帧，那么观众随后调谐其接收机时，可能会接收不到完整的画面。I-帧独立于其他帧，不能从其他帧中构造。

- P-帧。预测帧（predicted frame，P-帧），与前一个 I-帧或 P-帧有关。换言之，每一个 P-帧只包含前一帧中变化的部分。P-帧只能够从前一个 I-帧或者 P-帧来构造。P-帧比其他类型的帧携带的信息要少得多，压缩后占用的位数很少。
- B-帧。双向帧（bidirectional fram，B-帧）与前一个或者后一个 I-帧或 P-帧有关。换言之，每一个 B-帧与过去和将来都有关。注意，一个 B-帧与另一个 B-帧没有任何关系。

8.2.4　音频

音频（语音）信号是需要介质传输的模拟信号，而在真空中不能传播。声音在空气中的传播速度大约是 330m/s（740mph）。从正常人的听觉来看，可以听到的声频范围是大约 20Hz 到 20kHz，最大可听到 3300Hz 左右。

数字化音频

为了能够提供压缩，音频模拟信号通过模拟到数字转换被数字化。模拟到数字转换由两个过程组成：采样和量化。数字化过程被认为是脉冲编码调制（pulse code modulation），这个在第 7 章讨论过。过程包含对模拟信号采样，量化采样和将量化后的值编码成比特流。语音信号被以每秒 8000 个样本、每个样本 8 位采样，也就是 8 000 × 8 = 64 kbps。音乐是被以每秒 44 100 个样本、每个样本 16 位采样，也就是说，单声道是 44 100 × 16 = 705.6 kbps，而立体声是 1.411Mbps。

音频压缩

无损压缩和有损压缩都适用于音频压缩。无损音频压缩允许保存音频文件的精确副本，压缩比较小（大约是 2），它大多数是用于档案和可编辑的目的。有损算法提供更大的压缩比（5 到 20），用于主流用户设备。虽然有损算法牺牲了一点点质量，但是在实质上减少了空间和带宽需求。例如，一张 CD 可以播放一小时的高保真音乐，也可以播放两小时无损压缩音乐，或者八小时使用有损技术压缩的音乐。

用于语音和音乐的压缩技术有不同的需求。用于语音的压缩技术必须是低延迟，因为在电话中有效延迟降低通信的质量。用于音乐的压缩技术必须能用更少比特位的数量录制高品质的声音。音频压缩的两类技术是：预知编码和感知编码。

预知编码

预知编码（predictive coding）技术由于低延迟性而广泛应用于电话的语音编码中（有效延迟会降低通信的质量）。在这章开始的时候我们集中讨论过预知编码方法：DM、ADM、DPCM、ADPCM 和 LPC。

感知编码

即使是最好的感知编码（perceptual coding）方法也不能为多媒体应用充分地压缩一个 CD 品质的音频。用于创建 CD 品质音频的最常用的压缩技术是基于感知编码的技术，其应用了声音心理学的科学知识。感知编码中使用的算法首先将数据从时域转换为频域，然后在频域执行数据操作。因此，这种技术也称为频域-时域方法。

声音心理学是研究人们如何感知声音的。感知编码利用了人类听觉系统的缺陷，人类可闻度的下限是 0dB。大约 2.5 到 5kHz 才是唯一真正的声音频率。如图 8-23a 所示，频率在这两个频率间时下限较小而在这两个频率间以外下限则变高了。频率的分贝值在这条曲线以下时，我们就听不到，因此，没有必要对这样的频率编码。

例如，通过忽略频率低于 100Hz，能量低于 20dB 的声音，我们可以无损地保存位。

我们能用频率遮掩（frequency masking）和暂时遮掩（temporal masking）概念保存更多。对于

频率遮掩，某一频率范围内的较强声音能够部分或者全部遮掩另一个频率范围内较弱的声音。例如，如果在一个房间里，一个重金属乐队正在演奏，我们很难听到舞伴在说什么。在图 8-23b 中，大约 700Hz 的大声遮蔽音调，把频率在大约 250 到 1500Hz 间的可听见曲线的临界值提高了。对于暂时遮蔽，很强的声音可以使耳朵在它停止后的短时间内失去听觉。

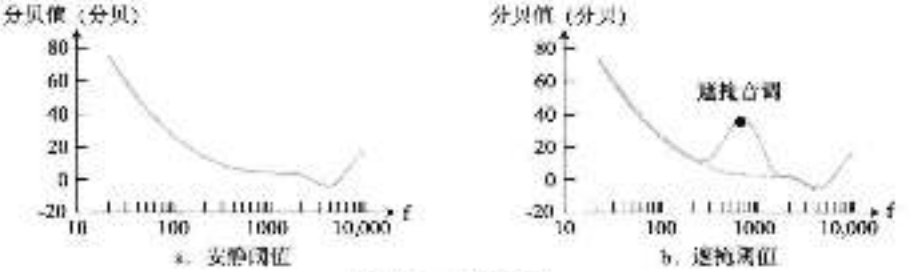


图 8.23　听觉临界

感知编码的基本方法是供给音频 PCM 同时输入编码器的两个独立的单元。第一个单元由一排称为分析滤波器组（analysis filter bank）的数字旁路滤波器组成。通过数学工具（如离散傅里叶转换（Discrete Fourier Transform，DFT）），滤波器通过时域转换输入到等距频率的子频带上。通过相同的或者类似的数学工具（如快速傅里叶转换（Fast Fourier Transform，FFT）），第二个单元将时域转换然后输入频域并且确定每个子频带的遮蔽频率。屈时按照每个子频带的遮蔽属性分配可用位：完全遮蔽后的子频带不分配位，部分遮蔽的子频带分配少量的位，而没有遮蔽的子频带分配绝大多数的位。得到的位再进一步编码以完成更进一步的压缩。

MP3

一个使用预知编码的标准是 MP3（MPEG 音频层面 3（MPEG audio layer 3））。

8.3　因特网中的多媒体

我们可以把音频和视频服务分为三大类：*流式存储音频/视频*（streaming stored audio/video）、*流式实况音频/视频*（streaming live audio/video）和*实时交互式音频/视频*（interactive audio/video）。流式是指用户可以在文件下载开始后就可以收听（或者观看）。

8.3.1　流式存储音频/视频

第一类，流式存储音频/视频，文件被压缩并存储在服务器上。用户通过因特网下载文件。有时候也称为*按需分配音频/视频*（on-demand audio/video）。音频文件的存储如歌曲、交响曲、磁带书籍和著名的演讲。视频文件的存储如电影、电视节目和音乐视频剪辑。我们可以说，流式存储音频/视频是指按需请求压缩的音频/视频文件。

第一种方法：使用 Web 服务器

压缩的音频/视频文件可以像文本文件一样下载。客户端（浏览器）使用 HTTP 服务，发送 GET 报文下载文件。Web 服务器可以将压缩的文件发送到浏览器中。然后浏览器借助于一个通常称为媒体播放器（media player）的应用程序播放文件。图 8-24 说明了这一方法。

这个方法很简单，还没有涉及流。但是，它存在缺陷。一个音频/视频文件即使压缩以后通常仍然很大。音频文件可能包含几十兆字节，而视频文件可能包含几百兆文字。在这种实现方法中，文件在播放之前需要完整地下载下来。使用目前的数据传输率，用户在文件播放之前需要几秒或几十秒的时间来下载文件。

第二种方法：使用带有元文件的 Web 服务器

在这种方法中，媒体播放器直接连接到 Web 服务器并下载音频/视频文件。Web 服务器存储这两种文件，实际的音频/视频文件和一个元文件（metafile），元文件中包含了音频/视频文件的有关信息。图 8-25 说明了这种方法的步骤。

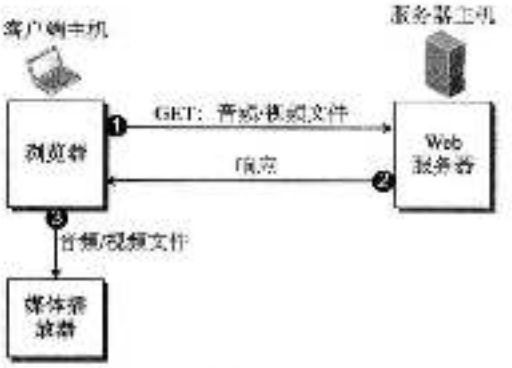


图 8-24　使用 Web 服务器



图 8-25　使用带有元文件的 Web 服务器

1. HTTP 客户端使用 GET 报文访问 Web 服务器。
2. 响应中包含元文件的有关信息。
3. 元文件传输到媒体播放器。
4. 媒体播放器使用元文件中指定的 URL 来访问音频/视频文件。
5. Web 服务器做出响应。

第三种方法：使用媒体服务器

第二种方法中存在的问题是，浏览器和媒体播放器都使用 HTTP 服务。HTTP 是运行在 TCP 之上的。它适合于检索文件，但是不适合检索音频/视频文件。原因在于 TCP 会重发丢失或者损坏的分段，这刚好和流的原理相反。所以我们不需要使用 TCP 和它的纠错控制，而是应该使用 UDP。但 HTTP 用于访问 Web 服务器，而 Web 服务器则是为 TCP 设计的，所以需要另外一个服务器，即**媒体服务器**（media server）。图 8-26 说明了这一概念。

1. HTTP 客户端使用 GET 报文访问 Web 服务器。
2. 响应中含有元文件的有关信息。
3. 元文件传送到媒体播放器。
4. 媒体播放器使用元文件中的 URL 访问媒体服务器以下载文件。通过任何使用 UDP 的协议都可以下载。
5. 媒体服务器做出响应。

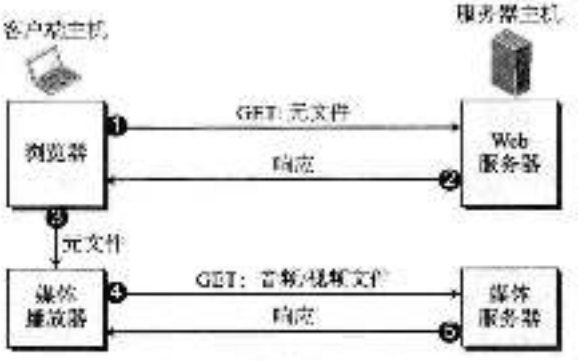


图 8-26　使用媒体服务器

第四种方法：使用媒体服务器和 RTSP

实时流协议（Real-Time Streaming Protocol，RTSP）是一种控制协议，用来为流的处理增加更多的功能。使用 RTSP 可以控制音频/视频的播放。RTSP 是一种带外控制协议，与 FTP 中的第二种连接很相似。图 8-27 说明了媒体服务器和 RTSP。

1. HTTP 客户使用 GET 报文访问 Web 服务器。
2. 响应中含有元文件的有关信息。

3. 元文件传送到媒体服务器。

4. 媒体播放器发送一条 SETUP 报文，建立与媒体服务器的连接。

5. 媒体服务器响应。

6. 媒体播放器发送一条 PLAY 报文以启动播放（下载）。

7. 使用一种运行在 UDP 之上的协议下载音频/视频文件。

8. 使用 TEARDOWN 报文断开连接。

9. 媒体服务器响应。

媒体播放器能够发送其他类型的报文。例如，一条 PAUSE 报文，能够暂时中止下载过程，通过 PLAY 报文可以重新开始下载。

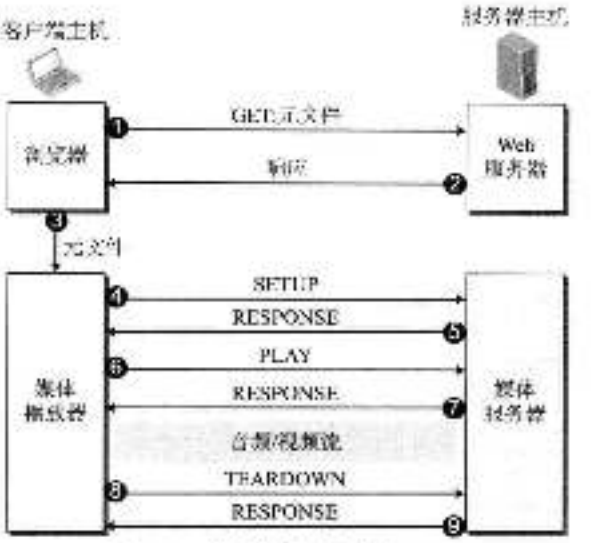


图 8-27　使用媒体服务器和 RTSP

例子：视频点播

视频点播（Video On Demand，VOD）允许用户从大量的存储视频中选择一个并且交互式地（暂停、倒回、快进等）观看。用户可能实时地观看视频，也可能先将视频下载到电脑、手提媒体播放器或者像数字录像机（digital video recorder，DVR）之类的设备上，以后再观看。有线电视、卫星电视和网络电视供应商提供按次收费和免费两种视频点播流。一些其他公司，比如 Amazon video 和如 Blockbuster video 之类的公司也提供视频点播。因特网电视在视频点播中日益流行。

8.3.2　流式实况音频/视频

第二类，流式实况音频/视频，类似于广播电台和电视台播放的音频和视频节目。区别在于电台通过因特网而不是无线电波播出。流式存储音频/视频与流式实况音频/视频之间有一些相似性，它们都对延迟很敏感，并且都不接受重传。但也有一些区别，在第一种应用情况下，通信是单播或点播的，在第二种情况下，通信是多播和实时的。实时流更适合于 IP 的多播服务和使用诸如 UDP 和 RTP（后面将要讨论）这样的协议。但是目前，实时流仍然使用 TCP 和多个单播代替多播。在这一领域，仍然有许多需要改进的地方。

例子：因特网广播

因特网广播（Internet radio）或者 Web 广播是一种通过因特网提供新闻、体育、会话和音乐的音频广播服务的网络广播。它包含可以从世界任何地方接入的流媒体。通过因特网提供 Web 广播但与传统广播媒体相似：它是交互式的，不能像交互式服务那样暂停和重播。一些最大的网络广播供应商包括现存的无线电台，其同时在传统广播和因特网上播出。也有一些只在因特网上播出的无线电台。Web 广播中，音频经常被 MP3 之类的软件压缩，然后封装成 TCP 或 UDP 包传输。为了防止抖动，在用户端位被重组和使用之前，要缓存和延迟几秒钟。

例子：因特网电视

因特网电视（Internet television，ITV）允许用户从节目表中选择想看的节目。ITV 的主要模型是流因特网电视或因特网某处中可选择的视频。

例子：IPTV

因特网协议电视（Internet protocol television，IPTV）是实时交互电视的下一代技术。IPTV 信号代替 TV 信号，前者在因特网上被传输，而后者通过卫星、电缆或者陆地线路被传输。注意，IPTV

与 ITV 不同：ITV 由服务供应商建立和管理，供应商不能控制最终传输。ITV 通过开放因特网中现有的基础设置分发；另一方面，IPTV 通过严格地管理在复杂和收费的网络上提供有保证的服务质量。IPTV 网络被设计为确保为用户有效地发送大量的多播视频流量和 HDTV 内容。

基于 IP 的平台提供了重要的优势，包括将其他基于 IP 的服务（比如高速因特网访问和 VoIP）与电视整合的能力。IPTV 与有线电视和卫星电视在操作上不同的一点是，在典型的有线电视或卫星电视中所有的内容是持续不断地从基站发送到用户。用户使用机顶盒（与电视相连的设备）从内容中选择。在 IPTV 中，内容仍然在网络中，只有用户选择的内容被发送。这种服务的优势是 IPTV 需要明显更少的带宽，所以允许交付更多的内容和更多的功能。缺点是用户的隐私可能遭到侵犯，因为 IPTV 的服务提供者可以准确地跟踪每个用户观看的每个节目。

8.3.3　实时交互式音频/视频

第三类，**交互式音频/视频**（interactive audio/video），人们相互之间进行实时通信。因特网电话或者 IP 语音是这种应用的实例。视频会话是另一个例子，人们可以进行可视化通信和交谈。

特性

在讨论这类应用使用的协议之前，先讨论一下实时音频/视频通信的一些特性。

时间关系

分组交换网络中的实时数据，需要保留一次会话期内各分组之间的时间关系。例如，我们假定有一台实时视频服务器创建了实时视频图像，并在线发送。视频被数字化并打包分组。只有三个分组，每个分组保存 10 秒的视频信息。第一个分组从 00:00:00 开始，第二个分组从 00:00:10 开始，第三个分组从 00:00:20 开始。假定每个分组花费 1 秒的时间到达目的地（为了简化，夸张了一些）。接收方可以在 00:00:01 播放第一个分组，在 00:00:11 播放第二个分组，在 00:00:21 播放第三个分组。尽管在服务器发送和客户端在屏幕上所看到的之间有 1 秒的时间差，但动作基本是实时的，保留了分组之间的时间关系，而 1 秒的时间延迟并不重要。图 8-28 说明了这种思想。

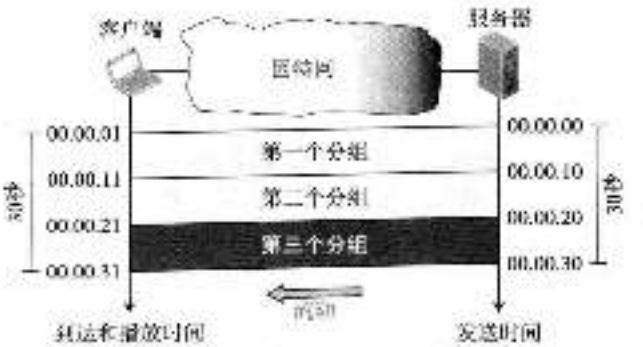


图 8-28　时间关系

但是，如果这些分组有不同的时间延迟，到达后会发生什么情况呢？例如，第一个分组在 00:00:01 到达（1 秒的延迟），第二个分组在 00:00:15 时刻到达（5 秒的延迟），第三个分组在 00:00:27 到达（7 秒的延迟）。如果接收方在 00:00:01 时刻播放第一个分组，则会在 00:00:11 时刻结束播放。但是下一个分组还没有到达，要在 4 秒之后才会到达。远方站点观看这一视频时，在第一个分组和第二个分组之间会出现间断，同样第二个分组和第三个分组之间也有时间间断。这种现象称为抖动（jitter）。图 8-29 说明了这种情况。

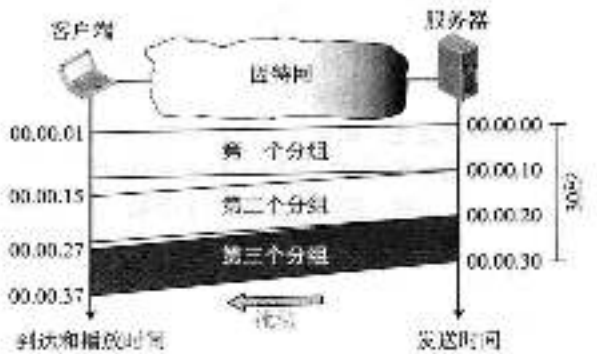


图 8-29　抖动

时间戳

解决抖动的一种方案是使用时间戳（timestamp）。如果每一个分组都有一个时间戳，标明它的生成时间与第一个（或者前一个）分组的时间间隔，那么接收方就能够将它的播放开始时间加上这一段时间。换句话说，接收方能够确定每一个分组的播放起始时间。在上面的例子中，假设第一个分组的时间戳是 0，第二个分组的时间戳是 10，第三个分组的时间戳是 20。如果接收方在 00:00:08 时刻播放第一个分组，则第二个分组会在 00:00:18 时刻开始播放，第三个分组在在 00:00:28 时刻开始播放。这样在分组之间就没有间断了。图 8-30 说明了这种情况。

为了防止抖动，可以为分组加上时间戳，将播放时间与到达时间进行分离。

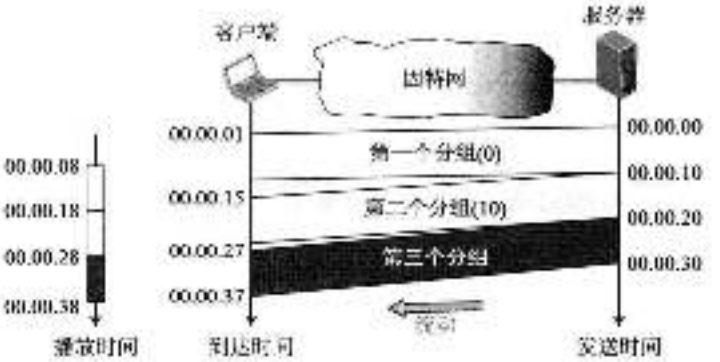


图 8-30　时间戳

回放缓冲区

为了将到达时间与播放时间分离开，需要一个缓冲区存储数据直到回放开始。缓冲区是指回放缓冲区（playback buffer）。当会话开始（第一个分组的第一个到达）时，接收方会延迟播放数据，直到到达一个阈值。在上一个例子中，第一个分组的第一份在 00:00:01 到达，阈值是 7 秒，回放时刻是 00:00:08。阈值通过数据的时间单元计算。取新播放会一直等到数据的时间单元数等于阈值时才会开始。

数据可能会以变化的传输速率存储在缓冲区内，但是它们是以固定速率抽取和回放的。注意，缓冲区内数据的数量可以缩小或扩大，但是只要延迟小于播放阈值时间，就不会产生抖动。图 8-31 说明了该例在不同时刻的缓冲区情况。

为了理解回放缓冲区如何能防止抖动，需要将回放缓冲区看做一个在每个分组中引入更多延迟的工具。如果对于每个分组将两种延迟时间的长度计算在一起，那么这就使得每个分组的总延迟（网络延迟和缓冲延迟）相等，然后好像没有延迟一样平稳回放分组。图 8-32 说明了对七个分组使用时间线的思想。注意，我们选择缓冲区中第一个分组的回放缓冲区，用这种方法是为了使右边的两个锯齿状曲线不重叠。

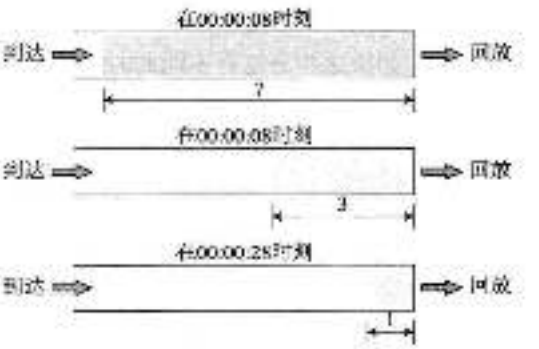


图 8-31　回放缓冲区

如图所示，如果第一个分组选择的是恰当的延迟时间，那么所有分组的总延迟应该是相等的。有较长传输延迟的分组应该有较短的缓冲延迟，反之亦然。



图 8-32　分组的时间线

排序

除了时间关系信息和用于实时通信量（real-time traffic）的时间戳以外，还需要更多的特性。每一个分组需要一个序列号。只靠时间戳是不能通知接收端是否有分组丢失的。例如，假定时间戳 0、10 和 20。如果第二个分组丢失了，那么接收端只收到两个分组，时间戳是 0 和 20。接收方假定时间戳为 20 的分组是第二个分组，也就是第一个分组 20 秒钟以后产生的。接收端没有办法确定第二个分组已经丢失。要处理这种情况，需要一个序列号对分组进行排序。

多播

多媒体播放在音频和视频会话中发挥着重要作用。通信量负载可能很重，数据流过使用多播（multicasting）的方式进行分发。会话在接收端和发送端之间需要双向通信。

转换

有时实时通信量需要转换（translation）。转换器是一台计算机，它能够将高带宽视频信号的格式转换为低质量的窄带信号。这种转换是必要的，例如，源端以 5Mbps 的速率创建了一种高质量的视频信号，而是通过低于 1Mbps 的带宽发送给接收端。要接收到这种信号，就需要转换器将信号解码，然后再次编码成只需要低带宽的低质量视频信号。

混合

如果有多个源方能够同时发送数据（如在视频或者音频会话中），通信量由多个数据流组成。要将通信量减少为一个数据流，就需要将来自多个源方的数据混合为一个数据流。混合器（mixer）通过数学算法将来自多个源方的信号相加，建立单一信号。

前向纠错

我们在第 5 章讨论过差错检测和重传。但是，重传损坏的和丢失的分组对于实时多媒体传输是没有用的，因为在复制过程中会产生一个不能接受的延迟；我们需要等待直到丢失或者损坏的分组传送到。我们需要立即纠错或者复制分组。设计了好几个方案适用于这种情况，这些方案统称为前向纠错（forward error correction，FEC）技术。在此我们对这种技术几种方法做简短的讨论。

使用汉明距离的纠错

在第 5 章我们讨论过用于差错检测的汉明距离。检测 s 位的差错，最小汉明距离应该是 $d_{min} = s + 1$。为了纠正错误，当然需要更多的汉明距离。可以表示为纠正 t 位差错，我们需要 $d_{min} = 2t + 1$。换言之，如果要纠正分组中的 10 个位，需要使最小汉明距离为 21 位，这意味着大量冗余位需要与数据一起发送。举一个例子，考虑著名的 BCH 编码。在这种编码中，如果数据是 99 位，检测仅仅 23

位可能的位错误需要发送 255 位（额外 156 位）。大多数情况下不能负担得起这样一个冗余。我们在习题集中给出一些如何计算需要多少比特位的例子。

使用 XOR 的纠错

另一个介绍的是使用异或操作特性，如下所示。

$$R = P_1 \oplus P_2 \oplus \ldots \oplus P_i \oplus \ldots \oplus P_N \quad \rightarrow \quad P_i = P_1 \oplus P_2 \oplus \ldots \oplus R \oplus \ldots \oplus P_N$$

换言之，如果在 N 个数据项（P_1 到 P_N）应用异或操作。这意味着可以将一个分组分成 N 块，创建所有块的异或并发送 $N+1$ 个块。如果块丢失或者损坏了，它可以在接收站被创建。现在问题是 N 的值应该是多少。如果 $N=4$，意味着需要发送 25%的额外数据，而且只有当每四个块中有一块丢失时才能纠正数据。

块交错

多媒体中实现 FEC 的另一种方式是允许一些小块在接收方丢失。我们不能接受属于相同分组的所有块都丢失，但是可以接受每个分组的一个块丢失。图 8-33 说明我们可以将每个分组分为 5 个块（实际上块数要更大一些）。然后我们可以（横向）逐块创建数据，但是需要纵向将块合并成分组。在这种情况中，每个发送的分组携带来自多个源分组的块。如果分组丢失，只是丢失每个分组的一个块，这通常在多媒体通信中是可以接受的。



图 8-33　交错

结合汉明距离和交错

可以将汉明距离和交错结合。首先创建 n 位的分组，其可以纠正 t 位错误。然后将 m 行交错，逐列地发送比特位。按照这种方法，自动纠错的位数会增加为 $m \times t$ 位。

混合高分辨率和低分辨率的分组

然而，另一个解决问题的办法是为每一个分组创建一个低分辨率冗余（low-resolution redundancy）的副本，并且将冗余版本（redundant version）与下一个分组合并。例如，我们可以从五个高分辨率分组中创建四个低分辨率冗余分组，并且如图 8-34 所示发送。如果一个分组丢失，可以使用下一分组的低分辨率版本。注意，一个分组的低分辨率部分是空的。这种方法中，如果最后一个分组丢失，则不能恢复。但是如果丢失的分组不是最后一个，我们就可以使用分组里的低分辨率版本。音频和视频的副本有不同的质量，但是缺失的质量在绝大多数时间内是察觉不到的。

实时应用例子：Skype

Skype（经典项目 Sky peer-to-peer 的缩写）是一种 P2P 的 VoIP 应用软件，最初由 Ahti Heinla，Priit Kasesalu 和 Jaan Tallinn 开发，他们也开发了 Kazaa（一种 P2P 文件共享应用软件）。这个应用允许持有音频输入和输出设备的注册用户在因特网上通过他们的电脑对其他注册用户进行免费的

PC-to-PC 语音呼叫。Skype 包括其很普及的特性，如即时通信（instant messaging，IM）、短信息服务（short message service，SMS）组通信、文件传输、视频会话和 SkypeIn 与 SkypeOut 服务。使用 SkypeIn 与 SkyOut 服务，可以使注册用户用较少的花费与传统的路线电话和移动电话进行通信。Skype 被称为引领全球互联网通信的公司之一，在 2009 年第三季度大约占国际通话分钟数总量的 8%。

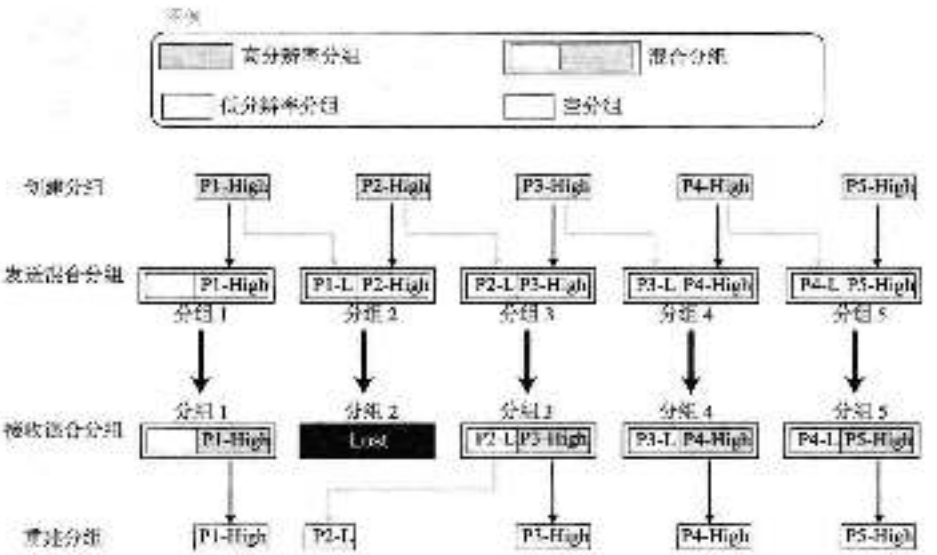


图 8-34　混合高分辨率和低分辨率分组

Skype 对于 PC-to-PC 呼叫是免费的，但是涉及 PSTN 或者手机时提供基于费用的服务。在涉及 PSTN 或手机服务的 Skype 会话有两种模式：SkypeIn 和 SkyOut。

为世界上的许多国家提供 Skype 服务，并且范围不断扩大。无论什么情况都提供服务，允许 Skype 注册用户在接入因特网的电脑上接收来自 PSTN 或手机的呼叫。要使用 SkypeIn，用户需要支付月租认购在线号码。用户可以使用 PSTN 或者手机呼叫这个号码，支付的费用与呼叫同一电话地区号内的其他 PSTN 或手机号码的标准价格相同。在线号码通过因特网将呼叫转发给 SkypeIn 注册用户。除了为在线号码需要付费外，注册用户接收到的呼叫是免费的。为了减轻呼叫方呼叫长途电话价格，注册用户可以认购多个在线号码。例如，注册用户可以在美国认购一个在线号码，在法国认购另一个在线号码。SkypeIn 服务附带免费的语音邮件服务。

SkypeOut 允许注册用户按当地的费率付费，从自己的电脑上电话呼叫世界上任何地方的任意一部 PSTN 电话或手机。要使用 SkypeOut 呼叫，用户需要购买月租或者 Skype 信誉分钟数。Skype 用户使用电脑可以拨打 PSTN 或者手机的电话号码。Skype 通过 SkypeOut 呼叫网关，网关再直接呼叫 PSTN 或手机服务。除了支付月租和 Skype 信誉分钟数费用，Skype 用户还需要支付少量的全球服务费和本地服务费。如果认购了在线号码，那么他们可以将来电转移到 PSTN 或手机上。需要强调的是，Skype 不能替代电话服务，Skype 也不能用于紧急呼叫。例如，在美国不能用 Skype 拨打 911。

8.4　实时交互式协议

在讨论过多媒体在因特网中使用的三种方法之后，我们现在集中讨论最后一个，也是最有趣和最复杂的：实时交互式多媒体（real-time interactive multimedia）。这个应用在因特网社区中已经引起广泛的关注，并且已经设计了多个应用层协议来处理这个应用。在我们讨论这种类型应用的需求

和原理之前，先给出一个示意图，如图 8-35 所示。



图 8-35　实时多媒体系统的示意图

尽管图中只有一个麦克风和一个音频播放器，但是现在的交互式实时应用通常由多个麦克风和多个摄像头组成。音频和视频信息（模拟信号）被转换成数字数据。数字数据通常由不同源的数据混合和封装而成。分组被发送到分组交换网络。在目的端接收分组有不同的延迟（抖动），并且有一些分组可能已经损坏或丢失。然后被发送到数字到模拟转换器中重新生成音频和视频信号。音频被发送给扬声器，而视频被发送给显示器。

源端每个麦克风或摄像头称为参与者，被赋予一个 32 位的标识符，这个标识符称为参与源标识符（contributing source identifier，CSRC）。混合器也称为同步设置，也被赋予另一个标识符，这个标识符称为同步源标识符（synchronizing source identifier，SSRC）。我们稍后将会使用分组中的这些标识符。

8.4.1　新协议的基本原理

从第 2 章到第 7 章，我们讨论了一般网络应用的协议栈。在这一节要说明为什么需要一些新的协议来处理交互式实时多媒体应用，如音频和视频会话。

显然我们不需要修改 TCP/IP 协议簇的前三层（物理层、数据链路层和网络层），因为这三层协议可以携带任意类型的数据。物理层为数据链路层提供服务，而不关心帧中比特位的具体内容。数据链路层保证网络层点对点的交付，而不关心分组的具体内容。尽管我们需要为多媒体应用提供更好服务质量的网络层，但是网络层也要保证主机对主机的数据报交付而不关心数据报的具体内容。我们将在下一节讨论这个问题。

看起来我们似乎只需要担心应用层和传输层。一些应用层协议在编码和压缩多媒体数据时需要考虑包括质量、带宽需求和编码与压缩时数学操作复杂度的平衡。简单地说，原来应用层协议允许在传输层处理多媒体的一些要求，而不只是在每个应用层协议处理。

应用层

显然我们需要开发一些用于实时多媒体的应用层协议，因为音频会话（audio conferencing）和视频会话（video conferencing）的本质与其他一些应用不同，如文件传输和电子邮件（在第 2 章讨论过）。许多具体的应用已经被专门的部门开发出来，而且市场上每天出现越来越多的应用。这些应用中的一些，如 MPEG 音频和 MPEG 视频，使用一些标准定义音频和视频数据传输。没有被所有应用都使用的具体标准，也没有每个应用都可以使用的具体应用协议。

传输层

缺少单一标准，多媒体应用（在 8.3.3 节讨论过）的综合特性提出了一些关于被所有多媒体应用使用的传输层协议的问题。两个公用的传输层协议 UDP 和 TCP，被开发出来时多媒体还没有在因特网使用。我们能使用 UDP 或 TCP 作为实时多媒体应用的通用传输层协议吗？为了回答这个问

题．首先需要考虑这个类型的多媒体应用的要求，然后看 UDP 或 TCP 是否可以满足这些要求。

交互式实时多媒体的传输层要求

首先让我们简略地为这类应用构成一套要求。

- 发送端-接收端协商。第一个要求与缺少音频或视频的单一标准有关。语音会话或视频会话有多个标准，这些标准对应着不同的编码或压缩方法。如果发送端使用一种编码方法而接收端使用另一种，那么通信是不可能的。应用程序需要在编码和压缩之前协商音频/视频上使用的标准，这样才可以传输成功。
- 创建分组流。我们在第 3 章讨论 UDP 和 TCP 时，提到 UDP 允许在向 UDP 发送报文之前用明显的边界将报文封装。另一方面，TCP 能处理字节流而不需要应用程序在数据块上设定明显的边界。换言之，UDP 适用于需要传输有明显边界的报文应用，而 TCP 适用于传输连续字节流的应用。就实时多媒体而言，上述两种协议都需要。实时多媒体是一种有帧界线的帧流或有明确块大小的块流。显然不论 UDP 还是 TCP 都不适用于处理这种情况下的帧流。UDP 不能提供帧之间的关系；虽然 TCP 提供字节之间的关系，但字节比一个多媒体帧或块更小一些。
- 源同步。如果一个应用使用多个源（音频和视频都有），此时就需要源之间同步。例如，使用音频和视频的电话会话（如 Skype），音频和视频可能使用不同的编码和压缩，其速率也不同。显然这两种类型的应用应该同步，否则我们会先听到声音后看到画面，反之亦然。也有可能有多个音频源或视频源（使用多个麦克风或多个摄像头）。源同步通常使用混合器完成。
- 纠错控制。我们已经讨论过处理差错（分组损坏或分组丢失）需要在实时多媒体应用中特别注意。我们不能接受将损坏的或丢失的分组重传。我们需要在数据中增加额外的冗余用于复制损坏的或丢失的分组（而不用请求重发分组的方法）。这意味着基于 TCP 协议的特性，TCP 不适用于实时多媒体应用。
- 拥塞控制。与其他应用类似，我们需要在多媒体中或多或少提供一些拥塞控制。如果决定在多媒体中使用 TCP（由于重发的问题），那么我们应该在系统中以某种方法实现拥塞控制。
- 防止抖动。我们在 8.3.3 节讨论过实时多媒体应用的一个问题是由接收端造成的抖动，因为因特网提供的分组转发服务对一个流中不同分组可能造成不均匀的延迟。在过去，电话网络提供音频会话，这个网络最初设计为电路交换网络，而电路交换网络是不存在抖动的。

如果我们逐步将所有的应用移植到因特网上，就需要处理抖动。我们在 8.3.3 节中提过减轻抖动的一个方法是使用回放缓冲区和时间戳。在接收端，回放缓冲区在应用层实现，但是传输层应该能够为应用层提供时间戳和排序。

- 发送端认证。如其他应用，在多媒体应用中一个微妙的问题是在应用层认证发送端。在我们使用因特网时，接入点通过 IP 地址认证。但是，我们需要将 IP 地址采用友好的方式进行映射，就像我们在 HTTP 协议或电子邮件里做的那样。

UDP 或 TCP 处理实时多媒体的能力

在讨论过实时多媒体的要求之后，让我们讨论一下 UDP 或者 TCP 是否有处理这些要求的能力。表 8-6 基于这些要求比较了 UDP 和 TCP。

表 8-6 中的第一行显示了一个很有趣的事实：UDP 和 TCP 都不支持所有的要求。然后，我们需要记住的是，需要一个传输层协议实现客户-服务器套接字。我们不能让应用层做传输层的工作。这意味着我们可能有三种选择：

1．我们可以使用一个新的传输层协议（如 SCTP，在这章结尾会讨论到），它综合了 UDP 和 TCP 的特性（尤其是流封装和多流）。这个选择可能是最好的，因为 SCTP 将 UDP 和 TCP 的特性

和其自身额外的特性综合在一起。

表 8-6　UDP 或 TCP 处理实时多媒体的能力

要　　求	UDP	TCP
1. 发送端/接收端协商选择编码类型	NO	NO
2. 创建分组流	NO	NO
3. 混合多个不同源的源同步	NO	NO
4. 纠结控制	NO	YES
5. 拥塞控制	NO	YES
6. 防止抖动	NO	NO
7. 发送端认证	NO	NO

2. 我们可以使用 TCP，将它与其他传输层协议合并用来弥补 TCP 不能提供的要求。然而，这个选择有点困难，因为 TCP 使用重发机制，而这是在实时应用程序中不被接受的。另一个问题是 TCP 不支持多播。一个 TCP 连接只是两个点连接，但是在实时交互式通信中我们需要多个点连接。

3. 我们可以使用 UDP，将它与其他传输层协议合并用来弥补 UDP 不能提供的要求。换言之，我们使用 UDP 提供客户端-服务器套接字接口，但是使用另一个协议，这个协议运行在 UDP 之上。这是当前多媒体应用的选择。这个传输层协议是实时传输协议（Real-time Transport Protocol RTP），我们稍后讨论。

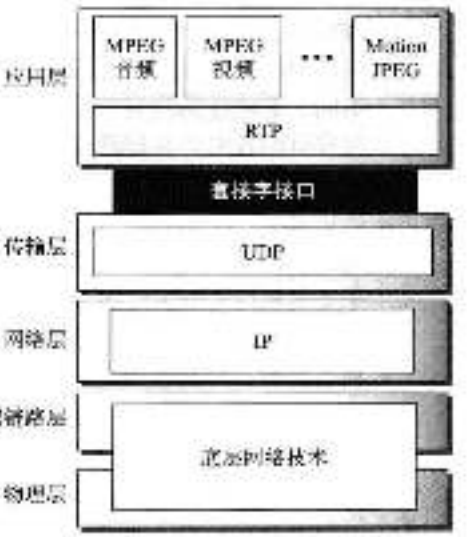


图 8-36　RTP 位于 TCP/IP 协议簇中的位置

8.4.2　RTP

实时传输协议（Real-Time Transport Protocol，RTP）是用来处理因特网上实时通信的协议。RTP 没有分发机制（多播、端口号等），它必须与 UDP 一起使用。RTP 位于 UDP 和应用程序之间。文献和标准中将 RTP 视为传输协议（不是传输层协议），可以看做其位于应用层（见图 8-36）。在 RTP 中封装来自多媒体应用程序的数据，然后依次发送到传输层。换言之，套接字接口位于 RTP 和 UDP 之间，这意味着我们在为每个多媒体应用程序开发的客户端-服务器程序中应该包含 RTP 功能。然而，一些编程语言提供一些工具，可以使编程工作更加简单。例如，为了使编程工作更简单，C 语言提供一个 RTP 库而 Java 语言提供了一个 RTP 类。如果使用 RTP 库或者 RTP 类，可以认为我们将应用程序从 RTP 中分离出来，而 RTP 则变成了传输层的一部分。

RTP 分组格式

在讨论 RTP 如何服务多媒体应用程序之前，让我们先讨论其分组格式。可以将 RTP 的功能与我们上一节讨论过的要求关联起来。图 8-37 说明了 RTP 分组头部的格式。这个格式非常简单而且通用，适合于所有的实时应用。如果应用需要更多的信息，则可以加入到头部开始的负载中。

下面是每个字段的描述说明：

- **版本**（Ver）。这是一个 2 位的字段，定义了版本号。现在的版本是 2。
- **填充位**（P）。这是一个 1 位的字段。如果值是 1，分组末有填充，此时填充中最后一个字节的值为填充的长度。如果分组是加密的，则填充是标准的。如果值是 0，则没有填充。这个字段的使用消除了对 RTP 数据长度的需求，因为如果没有填充数据的长度就是 UDP 数据

的长度减去 RTP 头部。否则，减去填充长度就是 RTP 数据长度。



图 8-37　RTP 分组头部格式

- **扩展标识位（X）**。这是一个 1 位的字段。如果值为 1，则说明在基础头部与数据之间有额外的扩展头部；如果值是 0，则没有额外的扩展头部。
- **贡献者数（Contributor count）**。这是一个 4 位的字段，用于表明贡献源的数目。注意，我们最多可以有 15 个贡献者，因为一个 4 位字段只允许 0 到 15 的数字。在音频或视频会话中，每个主动源（发送数据而不只接收数据的源）称为贡献者（contributor）。
- **标志位（M）**。这 1 位的字段是由应用指示的标志，例如，数据的结束。上述的流媒体应用是一个有结束帧标记的块流或帧流。如果这些位是 RTP 分组，则意味着 RTP 分组携带了这个标记。
- **有效载荷类型（Payload type）**。这是一个 7 位的字段，说明了有效载荷的类型。目前为止已经定义了许多有效载荷类型。在表 8-7 中列出了一些常用的应用。对这些类型的讨论超出了本书的范围。

表 8-7　有效载荷类型

类　型	应　用	类　型	应　用	类　型	应　用
0	PCMu 音频	7	LPC 音频	15	G728 音频
1	1016	8	PCMA 音频	26	Motion JPEG
2	G721 音频	9	G722 音频	31	H.261
3	GSM 音频	10-11	L16 音频	32	MPEG1 视频
5-6	DVI4 音频	14	MPEG 音频	33	MPEG2 视频

- **序列号（Sequence number）**。这是 16 位长的字段，用来给 RTP 分组进行编号。第一个分组的序列号是随机选择的；对后续的每一个分组则加 1。接收者用序列号检测丢失或者乱序的分组。
- **时间戳（Timestamp）**。这是 32 位长的字段，用来说明各分组之间的时间关系。第一个分组的时间戳是一个随机数；对随后的每一个分组，它的时间戳的值为前一分组的值与第一个字节产生的时间之和（取样值）。时钟计时单元的值取决于应用。例如，音频应用通常产生 160 字节的大块，那么这个应用的时钟计时单元就是 160。这个应用的时间戳的值就是每一个 RTP 分组累加 160。
- **同步源标识符（Synchronization source（SCRC）identifier）**。如果只有一个源，那么这 32 位就定义了源。然而，如果存在多个源，那么混合器是同步源，其他源都是贡献源。源标识符的值是由源选取的一个随机数。协议提供对突情况下的一个策略（两个源开始具有相同的序列号）。

- 贡献源标识符（Contributing source（CSRC）identifier）。每个 32 位的标识符（最大值是 15）定义一个源，当会话中有多个源时，混合器是同步源而其余的源都是贡献源。

UDP 端口

尽管 RTP 本身是一种传输层协议，但 RTP 分组并非直接封装为 IP 数据报。相反，RTP 被当作应用程序对待，被封装成 UDP 用户数据报。但是，与其他应用程序不同，没有为 RTP 分配熟知端口。端口可以按需要选择，这里只有一个限制：端口号必须是偶数。下一个端口号（奇数）由 RTP 的伴随协议（实时传输控制协议（RTCP））使用。

RTP 使用偶数编号的 UDP 端口。

8.4.3　RTCP

RTP 只支持一种类型的报文，该报文负责将数据从源方传送到目的方。为了真正地控制会话，我们在会话中还需要更多参与者之间的通信。既然这样，控制通信被设计为一种单独的协议，称为**实时传输控制协议**（Real-time Transport Control Protocol，RTCP）。我们需要强调的是 RTP 分组中不能携带 RTCP 有效载荷，RTCP 事实上是一种 RTP 的姐妹版协议。这意味着作为实际的传输协议，UDP 可以携带 RTP 有效负载，也可以携带属于上层应用的 RTCP 有效负载。

RTCP 分组创建了一个带外控制流，这个流在多媒体流的发送端和接收端之间提供反馈信息。特别是 RTCP 提供以下功能：

1. 接收端或者多媒体流的接收端从 RTCP 中获知网络性能，其直接与网络中的拥塞相关。多媒体应用使用 UDP（代替 TCP）以后，在传输层就没有办法控制网络拥塞。意味着如果必须控制拥塞，那么这应该在应用层完成。如果拥塞被发现并由 RTCP 报告，应用程序可以使用更加积极的压缩方法减少分组的数量从而减少拥塞（为了权衡性能）。换言之，如果没有发现拥塞，应用程序就是用比较消极的压缩方法提供更好的服务质量。

2. RTCP 分组中携带的信息可以用来同步与同一源相关的不同流。一个源可以使用两个不同的源收集音频或视频数据。此外，音频数据可能从多个不同的麦克风中收集，而视频数据可能从多个不同的摄像头中收集。总之，为了实现同步需要两条信息：

a. 每个发送端需要一个 ID。尽管每个源可能有不同的 SSRC，RTCP 为每个源提供单一的 ID 称为*规范名字*（canoical name，CNAME）。CNAME 可以用来关联不同的源并且允许接收端合并来自相同源中的不同源。例如，一个电话会话可能有 n 个发送端参与，但是我们可能有 m 个贡献流的源（$m > n$）。在这个系统中，我们只有 n 个 CNAME，m 个 SSRC。CNAME 的格式如下。如果是用户，CNAME 通常以用户的注册名字形式命名；如果是主机，则以主机域名形式命名。

user@host

b. 规范名字本身不能提供同步。为了同步源，我们需要知道流的绝对时间，除此之外还要知道在每个 RTP 分组中由时间戳提供的相对时间。每个分组中的时间戳举出分组中比特位与流开始时比特位之间的相对时间关系，它不能使一个流与另一个流有某种联系。绝对时间（有时也叫挂钟时间（wall clock））需要通过 RTCP 分组发送为了确保同步。

3. RTCP 分组可以携带关于发送端的额外信息，而这对接收端是有用的，如发送端的名字（规范名字之外）或视频的标题。



图 8-38　RTCP 分组类型

RTCP 分组

在讨论完 PTCP 的主要功能和用途之后，让我们来讨论它的分组。图 8-38 说明

了 RTCP 的五种分组类型。紧跟在每个方框后面的数字定义了每个分组的数值。我们需要提到多个 RTCP 分组可以被封装为一个单一的 UDP 有效载荷，因为 RTCP 分组比 RTP 分组小。

格式和每个字段的准确定义之间的关系是非常复杂的，这些超出了本书的范围。我们只简单地讨论每个分组的用途，并讨论与前面所描述的功能之间的关系。

发送端分组报告

发送端分组报告由会话中的主动发送端定期发送，用来报告在一定的时间间隔内所有 RTP 分组的发送和接收的统计数据。发送方分组格式包括以下信息：

- RTP 流的 SSRC。
- 绝对时间戳，由相对时间戳和挂钟时间组成的从 1970 年 1 月 1 日午夜算起的秒数。相对时间戳，允许接收端同步不同的 RTP 分组。
- RTP 分组数和会话（session）开始时发送的字节数。

接收端报告分组

接收端报告是由被动参与者发送的，这些被动参与者不发送 RTP 分组。报告通知发送端和接收端有关服务质量的信息。反馈信息可以用来在发送端控制拥塞。接收端报告包括以下信息：

- RTP 流（在此产生接收端报告）的 SSRC。
- 分组的丢失比例。
- 最后一个序列号。
- 间隔抖动。

源描述分组

源端周期性发送源描述报文，提供关于自身的附加信息。分组包括：

- SSRC。
- 发送方的规范名字（CNAME）。
- 其他信息，如真实名字、电子邮件地址、电话号码。
- 源描述分组可能也包含额外的数据，如视频上使用的字幕。

关闭分组

源端发送关闭分组来关闭流。它允许源端宣布要离开会话。尽管其他源端能够检测到一个源端的缺席，但是这个分组可以直接宣布。它对混合器也是非常有用的。

特定应用分组

特定应用分组是希望使用新的应用（没有在标准中定义）的一个应用分组。它允许定义新的分组类型。

UDP 端口

与 RTP 类似，RTCP 没有使用熟知的 UDP 端口。它使用临时端口。选择的 UDP 端口号必须是紧随 UDP 为 RTP 选定的端口号的那个数，它必须是一个编号为奇数的端口。

RTCP 使用奇数的 UDP 端口号，该端口号必须是 RTP 选定的端口号的下一个数。

带宽使用

RTCP 分组不仅可以通过主动发送端发送，也可以通过被动发送端发送，后者的端口号通常比前者的端口号大。这意味着，如果 RTCP 流量没有限制时，很可能失控。为了控制这种情况，RTCP 使用一个控制机制将会话中使用的流量（RTP 和 RTCP）限制为占很小的比例（通常是 5%）。这个比例中，大部分（x%）被分配给由被动接收端产生的 RTCP 分组，而剩余部分（$1-x$%）被分配给由主动发送端产生的 RTCP 分组。RTCP 协议使用一个基于被动接收和主动发送的比率的机制定义了值 x。

例 8.10　假设分配给一个会话的总带宽是 1Mbps。RTCP 流量仅占这个带宽的 5%，也就是 50kbps。如果只有 2 个主动发送方而有 8 个被动接收方，虽然每个发送方或者接收方的速率只为 5kbps。如果 RTCP 分组的平均大小是 5Kb，那么每个接收方或接收方每秒只能发送 1 个 RTCP 分组。注意，我们需要在数据链路层考虑分组大小。

实现需求

正如前面提到的，让我们讨论 RTP 和 RTCP 的组合是如何响应交互式实时多媒体应用的。在数字音频或视频流中，比特序列被分成多个块（有时也叫块或帧）。每个块都有预先定义的边界，这个边界用于区分前一个块或下一个块。块被封装成 RTP 分组，其定义特定的编码（有效载荷类型）、一个序列号、一个时间戳、一个同步源（SSRC）标识符和一个或多个贡献源（CSRC）标识符。

1. 第一个需求：发送端-接收端协商（sender-receiver negotiation）不能通过 RTP/RTCP 协议的组合实现。应该通过一些其他方法实现。稍后我们将会讨论另一个协议（SIP），这个协议提供用于协调 RTP/RTCP 的能力。

2. 第二个需求：创建块流（creation of a stream of chunk）通过将每个块封装成 RTP 分组并为每一块定义一个序列号实现。

3. 第三个需求：同步源（synchronization of source）通过对每个源通过 32 位的标识符认证并且在 RTP 分组中使用相对时间戳而在 RTCP 分组中使用绝对时间戳实现。

4. 第四个需求：差错控制（error control）通过使用 RTP 分组的序列号和前向纠错方法（FEC，在这章前面讨论过）重新生成丢失的分组实现。

5. 第五个需求：拥塞控制（congestion control）通过来自接收端的反馈实现，这个反馈是接收端使用接收端报告分组（RTCP 中）通知接收端丢失分组的序列号。然后接收端使用更加积极的压缩技术来减少发送分组的数量，从而减少阻塞。

6. 第六个需求：防止抖动（jitter removal）通过在数据回放缓冲区中使用的由每个 RTP 分组提供的时间戳和排序实现。

7. 第七个需求：源认证（identification of source）通过由发送端发送的包含源描述分组（RTCP 中）的规范名字实现。

8.4.4　会话初始化协议

我们以前讨论过如何使用因特网进行语音-视频会话。尽管 RTP 和 RTCP 可以用于提供这些服务，但是它忽略了一个组件：用于呼叫参与者的信号系统。

为了理解这个问题，让我们追溯到使用传统电话系统（交换电话网络或 PSTN）的传统电话会话（两个或多个人之间）时代。一个电话呼叫需要两个电话号码，一个呼叫方和一个被呼叫方。我们需要拨打被呼叫方的电话号码然后等待他的响应。电话会话在被呼叫方响应后开始。简言之，规定电话通信包括四个阶段：信令阶段和音频通信阶段。

电话网络中的**信令阶段**（signaling phase）由称为信令系统 7（Signaling System 7，SS7）的协议提供。SS7 协议从语音通信系统完全分离出来。例如，传统电话系统在电路交换网络上使用模拟信号携带语音，而 SS7 协议使用电脉冲，每拨打一个电话号码会引起一系列脉冲的变化。现在 SS7 不但提供呼叫服务，而且还提供其他的一些服务，如电话呼叫转移和错误报告。

我们在前一节中讨论的 RTP/RTCP 协议的合并相当于 PSTN 提供的语音通信，为了在因特网上模仿这个系统，我们需要一个信令系统。我们的追求需要我们更进一步。我们不只想能够在语音或视频会话中使用电脑（PC）呼叫对方，而且也想能够使用电话机、手机、PDA 等等。我们也需要在参与人不在其办公桌前时找到他。这就需要在混合设备间通信。

会话初始化协议（Session Initiation Protocol，SIP）由 IETF 设计。它是一种应用层协议，能够

建立、管理和终止多媒体会话（呼叫）。它可以用来创建双方、多方或者多播会话。SIP 设计上是独立于下面的传输层的，它能够运行在 UDP、TCP 或者 SCTP 之上，使用的端口号是 5060。SIP 可以提供以下几种服务：

- 在连接入因特网上的用户间建立会话。
- 在因特网上寻找用户的位置（IP 地址），因为用户的 IP 地址可能会改变（考虑移动 IP 和 DHCP）。
- 判断用户是否能够或者想进行会话呼叫。
- 按照多媒体用途和编码类型决定用户权限。
- 建立会话，设置自定义参数，如使用的端口号（记住 RTP 和 RTCP 使用的端口号）。
- 提供会话管理功能，如保持呼叫、电话呼叫转移、接听新的呼叫和改变会话参数。

通信双方

我们需要注意的是，交互式实时通信应用程序和其他应用程序相比不同的一点是通信双方。在一个音频或视频会话中，通信双方是人，而不是设备。例如，在 HTTP 或 FTP 中，客户需要在通信前查询服务器的 IP 地址（利用 DNS）。不需要在通信前查询人。在 SMTP 中，发送者向接收方的邮箱发送一封电子邮件，而不用考虑邮件什么时候被打开。在语音或视频会话中，呼叫方需要查询被呼叫方。被呼叫方可以坐在他的办公室，也可以在街上散步，或者任何位置。使得通讯更为复杂的原因就是：参与者在某一时间所使用的设备与另一时间使用的设备在功能上有所不同。SIP 协议需要查询被呼叫方的位置，并同时与参与者使用的设备能力进行协商。

地址

在常规的电话通信中，一个电话号码可用于识别发送方，而另一个电话号码用于识别接收方。SIP 非常灵活。在 SIP 中，电子邮件地址、IP 地址、电话号码和其他类型的地址都能用于识别发送方和接收方。但是，地址必须使用 SIP 格式（也就是呼叫方案）。图 8-39 列出了一些常用的格式。

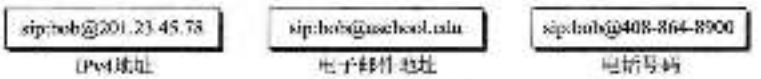


图 8-39　SIP 格式

我们已经注意到，SIP 地址与在第 2 章中见过的 URL 很相似。实际上，SIP 地址就是包含在潜在被呼叫方网页中的 URL。例如，Bob 可以包含上述地址中的一个作为他的 SIP 地址，如果有人点击它，则 SIP 协议被唤醒并呼叫 Bob。其他地址也是可能的，比如那些使用姓在名前面的地址，但是所有地址需要以这样的格式，sip:user@address。

报文

SIP 是一种类似 HTTP 的基于文本的协议。与 HTTP 类似，SIP 也使用报文。SIP 中的报文分为两大类：请求和响应。这两大类的格式如下所示（注意与图 2-12 所示的 HTTP 报文的相似性）：

Request Messages	请求报文	Response Messages	响应报文
Start line	起始行	Status line	状态行
Header	头部	Header	头部
Blank line	空行	Blank line	空行
Body	正文	Body	正文
one or more lines	一行或多行		

请求报文

IETF 最初定义了六种请求报文，但是一些新的请求报文已经被计划用于扩展 SIP 功能。我们

只提及最初定义的六种报文，如下：

- INVITE。呼叫方使用 INVITE 请求报文用于初始化会话。使用这个请求报文，呼叫方邀请一个或多个呼叫方参与这个会话。
- ACK。呼叫方发送 ACK 报文用于确认这个会话初始化已经完成。
- OPTIONS。OPTIONS 报文查询一台机器的功能。
- CANCEL。CANCEL 报文取消已经启动的初始化进程，但是不能终止呼叫。新的初始化可能在 CANCEL 报文之后开始。
- REGISTER。REGISTER 报文会在被呼叫方不可用时建立连接。
- BYE。BYE 报文用于终止会话。BYE 报文与 CANCEL 报文相比，BYE 报文由呼叫方或被呼叫方发起，用于终止整个会话。

响应报文

IETF 也定义了六种响应报文，可以发送给请求报文，但是注意请求报文和响应报文之间没有关系。一个响应报文可以被发送给任意的请求报文。与其他基于文本的应用协议类似，使用三位数定义响应报文。简单地描述响应报文如下：

- **消息性质的响应**（Informational Responses）。这个响应的格式是 SIP 1xx（通常有：100 正在处理中（trying）、180 振铃（ringing）、181 呼叫转移（call forwarded）、182 排队（queue）和 183 会话进度（session progress））。
- **成功响应**（Successful Responses）。这个响应的格式是 SIP 2xx（通常有：200 处理成功（OK））。
- **转发响应**（Redirection Responses）。这个响应的格式是 SIP 3xx（通常有：301 永久转发（moved permanently）、302 临时转发（moved temporarily）、380 替代服务（alternative service））。
- **客户端故障响应**（Client Failure Responses）。这个响应的格式是 SIP 4xx（通常有：400 语法错误（Bad Request）、401 认证失败（unauthorized）、403 拒绝执行（forbidden）、404 不存在（not found）、405 不允许应用此方法（method not allowed）、406 无法接受（not acceptable）、415 不支持的媒体类型（unsupported media type）、420 不支持扩展（bad extension）、486 对方忙（busy here））。
- **服务器故障响应**（Sever Failure Responses）。这个响应的格式是 SIP 5xx（通常有：500 服务器内部故障（sever internal error）、501 没有实现（not implemented）、503 服务器不可用（sever unavailable）、504 超时（timeout）、505 SIP 版本不支持（SIP version not supported））。
- **总体故障响应**（Global Failure Responses）。这个响应的格式是 SIP 6xx（通常有：600 对方忙（busy everywhere）、603 拒绝（decline）、604 不存在（doesn't exist）和 606 不支持（not acceptable））。

第一种方案：简单会话

在第一种方案中，假设 Alice 需要呼叫 Bob，在通信中使用 Alice 的 IP 地址和 Bob 的 SIP 地址。我们可以把这个通信分为三个模块：建立、通信和终止。图 8-40 说明了一个使用 SIP 的简单会话。

建立会话

建立一次会话需要三次握手。呼叫方使用 UDP、TCP 或者 SCTP 发送 INVITE 报文开始通信。如果 Bob 愿意启动会话，他会发送一条响应报文（200 OK）。为了确认已经接收到的回复代码，Alice 会在开始音频通信前发送一条 ACK 报文进行确认。建立阶段使用两种请求报文（INVITE 和 ACK）和一种响应报文（200 OK）。稍后我们更详细地讨论报文的内容，但现在需要说明 INVITE 报文起始行定义接收方的 IP 地址和 SIP 版本。在头部没有写入任何行，但稍后会做的。头部的正文用另一种协议，会话描述协议（Session Description Protocol，SDP），其定义了语法（格式）和语义（指每行）。稍后我们简短地讨论这个协议。我们刚刚提到正文的第一行定义了报文的发送方，

第二行定义了 Alice 呼叫 Bob 的用于 RTP 的端口号。响应报文定义了媒体（音频）和 Bob 到 Alice 方向用于 RTP 的端口号。在 Alice 确认会话建立并发送 ACK 报文请求（不需要响应）时，建立会话完成并且开始通信。

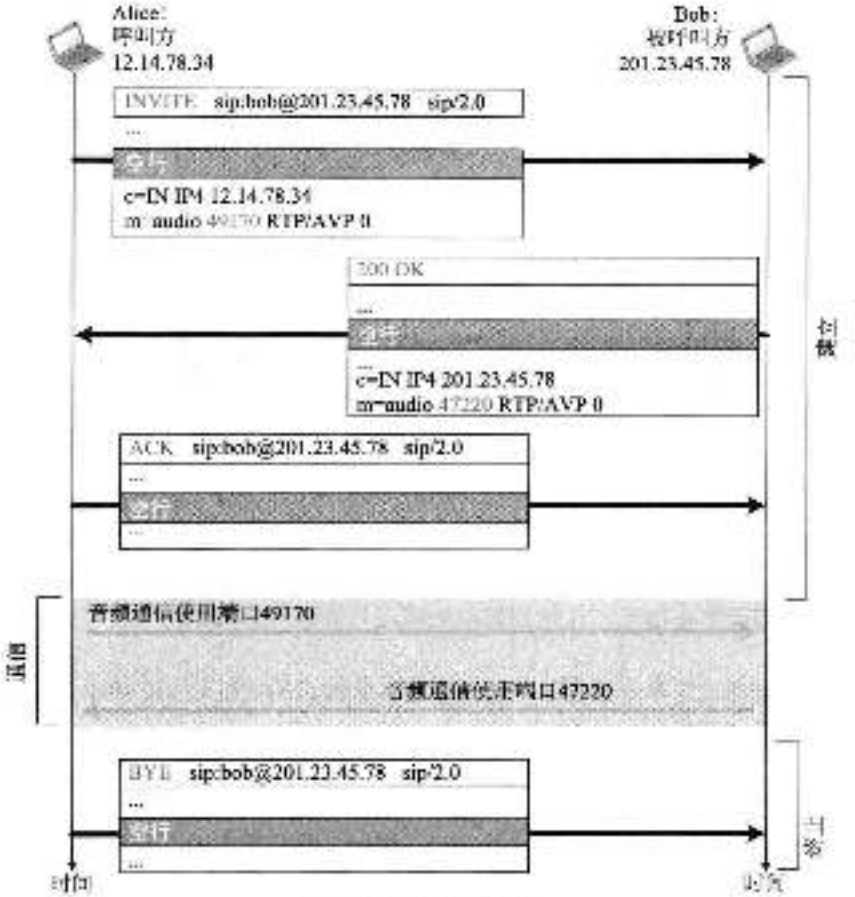


图 8-40　SIP 简单会话

通信

会话建立之后，Alice 和 Bob 能够通过两个临时端口进行通信。RTP 使用端口号为偶数的端口，而 RTCP 使用端口号为奇数的端口（我们在图 8-40 中仅说明了 RTP 使用的偶数端口号）。

终止会话

任何一方发送一条 BYE 报文，会话就会结束。在图中我们假设 Alice 终止了会话。

第二种方案：跟踪被呼叫方

如果 Bob 没有坐在终端前面，会发送什么情况呢？他可能离开了自己的系统，或者在另一台终端前面。如果使用 DHCP，他甚至不会有固定的 IP 地址。SIP 使用一种机制（与 DNS 中的一种类似），能够查找到被呼叫方所处的终端的 IP 地址。要执行跟踪，SIP 使用注册的概念。SIP 规定了一些服务器作为注册服务器（register server）。任何时刻，用户至少注册到一台服务器上，这台服务器就能知道被呼叫方的 IP 地址。

当 Alice 需要与 Bob 通信时，Alice 可以用电子邮件代替在 INVITE 报文中的 IP 地址。报文会传送到代理服务器中。代理服务器发送一条查询报文（不属于 SIP 的一部分）到某台 Bob 注册的

服务器。当代理服务器从注册服务器接收到回复报文时，代理服务器取出 Alice 的 INVITE 报文，插入新发现的 Bob 的 IP 地址，然后将这条报文发送到被呼叫方。图 8-41 说明了这一过程。

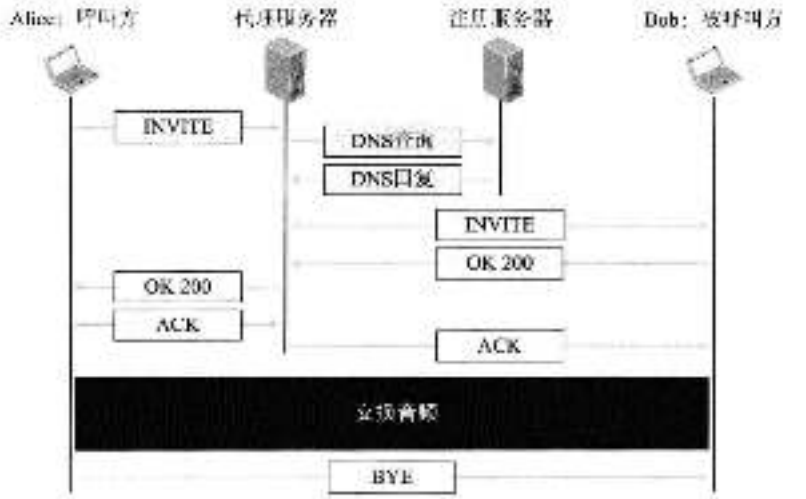


图 8-41　跟踪被呼叫方

SIP 报文格式和 SDP 协议

正如我们之前讨论过的，SIP 请求和响应报文被分为四部分：起始或状态行、头部、空行和正文。因为空行不再需要更多信息，让我们简单地描述其他部分的格式。

起始行

起始行是以报文请求名字开始的单一行，紧跟着接收者的地址和 SIP 版本。例如，INVITE 报文请求起始行的格式如下所示：

INVITE sip:forouzan@roadrunner.com

状态行

状态行是以三位数的响应代码开始的单一行。例如，响应报文 200 的格式如下所示：

200 OK

头部

在请求或响应报文中，头部能够使用许多行，每一行以名字、冒号、空格和后面跟着的值开始。一些类型的头部行为：Via、From、To、Call-ID、Content-Type（内容类型）、Content-Length（内容长度）和 Expired（结束）。Via 头部定义了 SIP 设备，报文就是通过这个 SIP 设备发送出去，包括发送方。From 头部定义了发送方，而 To 头部定义了接收方。Call-ID 头部是一个随机数，定义了会话。Content-Type 定义了报文中主体的类型，通常为 SPD，我们稍后简短地描述它。Content-Length 定义了报文中主体以字节为单位的长度。Expired 头部通常在 REGISTER 报文中用于定义主体中信息的结束。下面有一个 INVITE 报文头部的例子。

```
Via: SIP/2.0/UDP 145.23.76.80
From: sip:alice@roadrunner.com
To: sip:bob@arrowhead.net
Call-ID: 23a345@roadrunner.com
Content-Type: application/spd
Content-Length: 600
```

主体

报文的主体是应用（如 HTTP）和 SIP 之间主要的区别。SIP 使用其他的协议，称为会话描述

协议（Session Description Protocol，SDP），用来定义注意。主体中的每行由 SDP 代码接着是等号，然后接着是值组成。代码是决定代码用途的单一特性。我们可以将主体分为下面几个部分。

主体的第一部分通常是基本信息。在这部分使用的代码为：v（表示 SDP 的版本）和 o（表示报文的源）。

主体的第二部分通常为接收方提供消息用于决定是否参与会话。在这部分使用的代码为：s（主题）、i（主题的消息）、u（表示会话 URL）和 e（会话负责者的电子邮箱地址）。

主体的第三部分提供使会话可能的技术细节。在这部分使用的代码为：c（用户参与会话需要连接到的单播或多播 IP 地址）、t（会话的起始时间和结束时间，编码为标数）、m（关于媒体的信息，如音频、视频、端口号、使用的协议）。

下面的一个例子说明了 INVITE 请求报文的主体。

```
v=0
o=forouzan 64.23.45.8
s=computer classes
i=what to offer next semester
u=http://www.uni.edu
e=forouzan@roadrunner.com
c=IN IP4 64.23.45.8
t=2923721854 2923725454
```

将这些部分组合起来

让我们把报文请求的四个部分如下组合起来。第一行是起始行，接下来的六行组成头部。接下来的一行（空行）将主体和头部分开，最后八行是报文的主体。我们结束关于 SIP 协议和辅助协议 SPD 的讨论，SIP 使用辅助协议 SPD 定义主体。

```
INVITE sip:forouzan@roadrunner.com
Via: SIP/2.0/UDP 145.23.76.80
From: sip:alice@roadrunner.com
To: sip:bob@arrowhead.net
Call-ID: 23a345@roadrunner.com
Content-Type: application/spd
Content-Length: 600
// Blank line
v=0
o=forouzan 64.23.45.8
s=computer classes
i=what to offer next semester
u=http://www.uni.edu
e=forouzan@roadrunner.com
c=IN IP4 64.23.45.8
t=2923721854 2923725454
```

8.4.5　H.323

H.323 是 ITU 制定的一个标准，它允许公共电话网络上的电话与连接到因特网上的计算机（在 H.323 中称为终端）进行通话。图 8-42 说明了音频上 H.323 的总体结构，而且也可以用于视频。

网关（gateway）将因特网连接到电话网络中。通常，网关是一种五层设备，能够把报文从一个协议栈转换到另一个协议栈。此处的网关实际上做同样的事情。它将电话网络报文转换为因特网报文。本地局域网的关守（gatekeeper）承担着注册服务器的角色，正如在 SIP 协议中讨论的。

协议

H.323 使用多种协议建立和维护音频（或视频）通信。图 8-43 列出了这些协议。H.323 使用 G.71 或 G.723.1 进行压缩。使用一个名为 H.245 的协议实现通信方之间压缩方法的协商。Q.931 协议用于建立和终止连接。H.225 或 RAS（Registration/Administration/Status，RAS）协议用于与关守的注册。

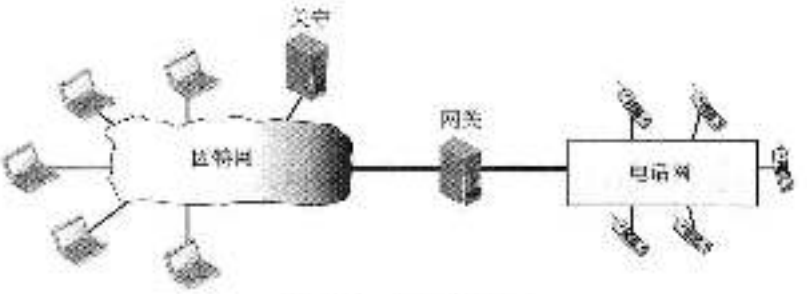


图 8-42　H.323 结构

需要提及的是，不能与 SIP 比较，H.323 是一组完整协议，而 SIP 只是一个信令协议，通常由 RTP 和 RTCP 组合创建一组多用于交互式实时多媒体应用的协议，但是它不能与其他协议一起使用。换言之，H.323 是授权使用 RTP 和 RTCP 的一组完整协议。

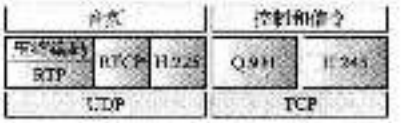


图 8-43　H.323 协议

操作

下面用一个简单的例子说明使用 H.323 协议进行电话通信的操作步骤。图 8-44 说明了一台计算机终端与一部电话之间通信的步骤。

1. 终端向关守发送广播报文。关守用它自己的 IP 地址响应。
2. 终端与关守通信，使用 H.225 协商带宽。
3. 终端、关守、网关和电话使用 Q.931 通信，以建立连接。
4. 终端、关守、网关和电话通信，使用 H.245 协商压缩方法。
5. 终端、网关和电话使用 RTP 在 RTCP 的管理下交换音频。
6. 终端、网关和电话使用 Q.931 终止通信。

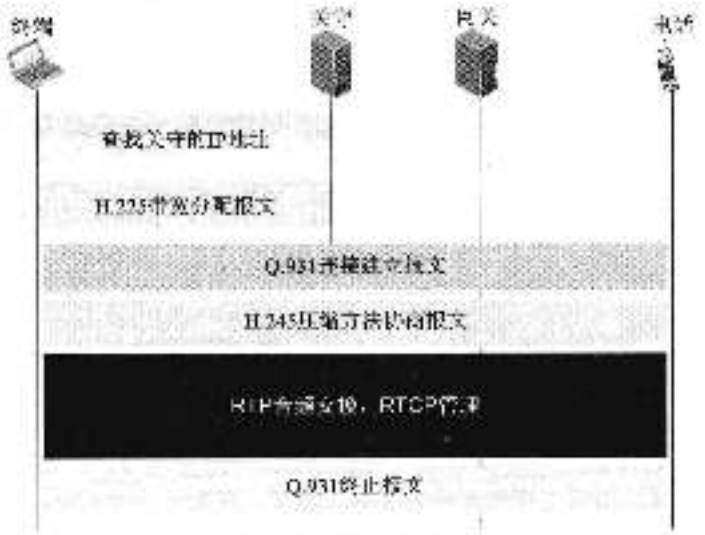


图 8-44　H.323 实例

8.4.6　SCTP

流控制传输协议（Stream Control Transmission Protocol，SCTP）是一种新的传输层协议，用来合并 UDP 和 TCP 的一些特性为多媒体通信创建一种更好的协议。

SCTP 服务

在讨论 SCTP 的操作之前，让我们先解释 SCTP 为应用层进程提供的服务。

进程到进程的通信

与 UDP 和 TCP 类似，SCTP 提供进程到进程的通信。

多流

我们之前学过，TCP 是一种基于流的协议。TCP 客户端和 TCP 服务器之间的每个连接包含一个单一的流。这种方法的问题是流中任意点的丢失都会阻塞其余数据的发送。在传输文本时这个方法可以被接受，但是不适用于发送如音频或视频之类的实时数据。SCTP 允许每个连接中多流服务（multistream service），在 SCTP 术语中称为关联（association）。如果流中的一个被堵塞了，其他的流仍然可以发送数据。图 8-45 说明了多流发送的思想。

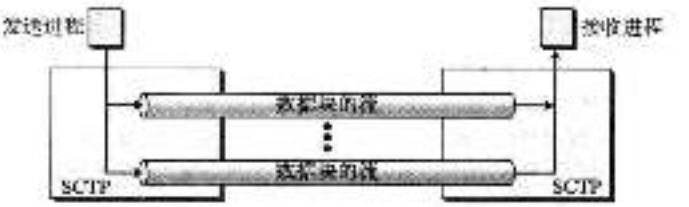


图 8-45　多流概念

多接口

一个 TCP 连接包含一个源 IP 地址和一个目的 IP 地址。这意味着，即使发送方或接收方是一个多接口主机（连接到多个物理地址，每个物理地址有多个 IP 地址），每端在连接中也只能使用这些 IP 地址中的一个。另一方面，SCTP 关联支持多接口服务（multihoming service）。发送主机和接收主机可以定义关联中每端的多 IP 地址。在这个容错方法中，一条路径失败时其他接口可以用来发送数据而不需要中断。在发送或接收实时有效载荷（如因特网电话）时容错特性是非常有用的。图 8-46 说明了多接口这个概念。

在图中，客户端使用两个 IP 地址连接在两个本地网络上。服务器也使用两个 IP 地址连接在两个本地网络上。客户端和服务器可以使用四对不同的 IP 地址获得一个关联。但是要注意的是，在当前实现的 SCTP 中，正常的通信只能选择一对 IP 地址，其他的作为替代在这条通信失败时使用。换言之，目前 SCTP 不允许不同的路径之间负载共享。



图 8-46　多接口概念

全双工通信

与 TCP 类似，SCTP 提供全双工服务，同一时刻在两端都可以发送和接收数据。每个 SCTP 都有一个发送缓存和一个接收缓存，用于在两端同时发送和接收分组。数据可以同时双向流动。

面向连接的服务

与 TCP 类似，SCTP 是一种面向连接的协议。但是在 SCTP 中的连接被称为关联（association）。

可靠的服务

与 TCP 类似，SCTP 是一种可靠的传输协议。它使用确认机制来检查数据是否安全到达。我们将在差错控制那一节中更进一步地讨论这个特性。

SCTP 特性

SCTP 的基本特性如下。

传输序列号（TSN）

SCTP 中的数据单元是数据块，由于分段会使得进程中的多个报文不一定有对应关系。在 SCTP 中数据传输由数据块的序列号控制。SCTP 使用传输序列号（Transmission Sequence Number，TSN）为数据块编号。换言之，SCTP 中 TSN 的规则与 TCP 中序列号的规则类似。TSN 是 32 位长并被初始化为 0 到 2^{32}-1 之间的一个随机数。每个数据块必须在它的头部携带相应的 TSN。

流标识符（SI）

在 SCTP 中，一个关联可以有多个流。SCTP 中的每个流需要用流标识符（Stream Identifier，SI）来标识。每个块必须在其头部携带 SI，以至于在到达终点时可以准确的与来自其他流中的块区分开来。SI 是一个从 0 开始的 16 位数。

流序列号（SSN）

在 SCTP 中当数据块到达终点时，它被以适当的顺序交付给适当的流。这意味着，除了 SI 外，SCTP 为每个流中的数据块定义了流序列号（Stream Sequence Number，SSN）。

分组

在 TCP 中，分段携带数据和控制信息。数据是以字节集的形式被携带，而控制信息在头部被定义为六种控制标记位。而 STCP 的设计则是完全不同的：数据的形式是数据块，而控制信息的形式是控制块。多个控制块和数据块封装在一起形成一个分组。SCTP 中分组的规则与 TCP 中分段的规则相同。图 8-47 比较了 TCP 中的分段和 SCTP 中的分组。我们将在下一节讨论 SCTP 分组的格式。

在 SCTP 中有数据块、流和分组。一个关联可能发送多个分组，每个分组可能包含多个块，每个块可能属于不同的流。为了清楚地定义这些术语，让我们假设进程 A 需要通过 3 个流向进程 B 发送 11 个报文。前四个报文在第一个流中，接下来的四个报文在第二个流中，最后四个报文在第三个流中。虽然如果一个报文很长，可以由多个数据块携带，但我们假设每个报文只由一个数据块携带。所以，在这三个流中有 11 个数据块。

应用进程交付 11 个报文给 SCTP，在其中每个报文被标记为对应的流。虽然进程需要交付第一个流中的一个报文然后交付第二个流中的另一个报文，但是我们假设首先交付第一个流中的所有报文，然后交付第二个流中的所有报文，最后交付第三个流中的。

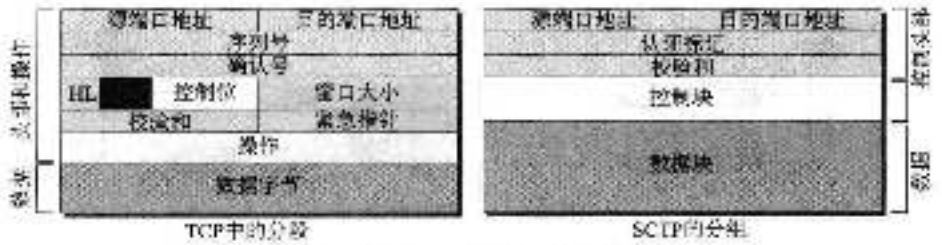


图 8-47　TCP 分段和 SCTP 分组的比较

我们假设网络中每个分组只允许包含三个数据块，这意味着我们需要 4 个分组，如图 8-48 所示。流 0 中的数据块由第一个分组和第二个分组中的部分携带，流 1 中的数据块由第二个分组和第三个分组携带，流 2 中的数据块由第三个分组和第四个分组携带。

注意，每个数据块需要三个标识符：TSN、SI 和 SSN。TSN 是使用过的累积数，用于流量控制和差错控制（稍后我们会讲到）。SI 定义了包含块的那个流。SSN 定义了块在特定流中的序列。在这个例子中，SSN 在每个流中从 0 开始。

确认号

TCP 确认号是基于字节的，称为序列号。而 SCTP 确认号是基于块的，称为 TSN。TCP 确认号

和 SCTP 确认号的第二个不同点是控制信息。这个信息是 TCP 中分组头部的一部分。为了确认只携带控制信息的分段，TCP 使用序列号和确认号（例如，SYN 分组需要通过 ACK 分段确认）。但是，在 SCTP 中控制信息是由控制块携带，而控制块不需要 TSN。这些控制块通过相应类型的另一个控制块确认（一些不需要确认）。例如，INIT 控制块通过 INIT-ACK 块确认，而不需要序列号或确认号。

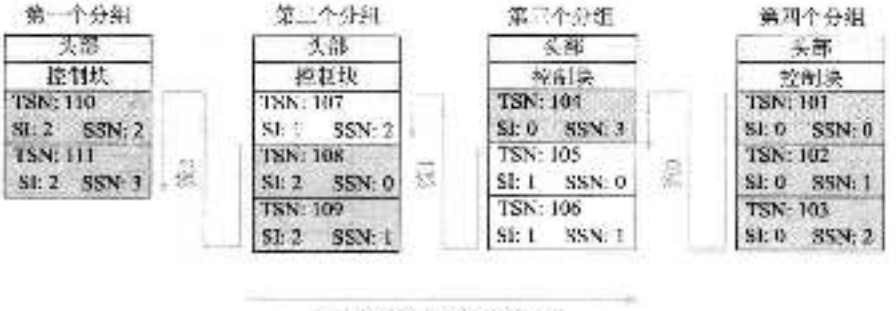


图 8-48　分组、数据块和流

分组格式

SCTP 分组有一个强制的通用头部（general header）和被称为数据块（chunk）的一组块（block）有两种类型的块：控制块和数据块。控制块控制和保持关联；数据块携带用户数据。在分组中，控制块于数据块之前到达。图 8-49 说明了 SCTP 分组的一般格式。



图 8-49　SCTP 分组格式

通用头部

通用头部（分组头部）定义了每个分组所属关联的结束点，也保证分组属于一个特定的关联，并且保存包括头部在内的分组内容的完整性。通用头部的格式如图 8-50 所示。

通用头部中有 4 个字段。UDP 和 TCP 中的源端口号与目的端口号相同。认证标记是一个 32 位的字段，用来将分组匹配到关联上。这个字段防止将来自前一个关联的分组误认为是来自当前关联的分组。对关联而言，它起到标识符的作用。下一个字段是校验和。然而，为了支持 CRC-32 校验，SCTP 中的校验和大小从 16 位增加到 32 位。

块

块携带控制信息或用户数据。如图 8-51 所示，块有一个公用的布局。前三个字段被所有的块公用，信息字段取决于块的类型。类型字段可以定义 256 种类型的块。到目前为止只有很少字段被使用，而剩余字段保留着为以后使用。标记字段定义了具体块可能需要的具体标记。每一位都有取决于块类型的不同意义。长度字段定义了块的总大小（以字节为单位），包括类型、标记和长度字段。由于信息区域大小取决于块的类型，我们需要定义块的边界。如果块不携带信息，长度字段的值为 4（4 字节）。注意，如果有填充字段，计算长度字段的值不包括填充的长度。这可以帮助接收端找出一个块中携带了多少有用的字节。如果值不是 4 的倍数，那么接收端则知道有填充字段。

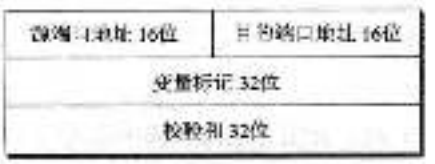


图 8-50　通用头部



图 8-51　块公用的布局

块的类型　SCTP 定义了多种类型的块，如表 8-8 所示。

表 8-8　块

类　型	块	描　述
0	DATA	用户数据
1	INIT	建立一个关联
2	INIT ACK	确认 INIT 块
3	SACK	选择性的确认
4	HEARTBEAT	探测对等站点是否处于活跃状态
5	HEARTBEAT ACK	确认 HEARTBEAT 块
6	ABORT	关联中止失败
7	SHUTDOWN	终止关联
8	SHUTDOWN ACK	确认 SHUTDOWN 块
9	ERROR	报告协议出不关键
10	COOKIE ECHO	关联建立中的第三个分组
11	COOKIE ACK	确认 COOKIE ECHO 块
14	SHUTDOWN COMPLETE	关联结束中的第一个分组
192	FORWARD TSN	用于调整累积 TSN

SCTP 关联

SCTP 与 TCP 类似，是一种面向连接的协议。但是，为了强调多接口，SCTP 中的连接称为关联。

SCTP 中的连接称为关联。

关联建立

在 SCTP 中的关联建立（association establishment）需要 4 次握手。在这个过程中，一个进程（通常为客户端，想要与另一个进程（通常为服务器）建立一个关联，需要使用传输层协议中的 SCTP。类似于 TCP，SCTP 服务器需要准备好接收任意关联（被动打开）。但是，建立关联通过客户端（主动打开）初始化。图 8-52 说明了 SCTP 关联的建立。

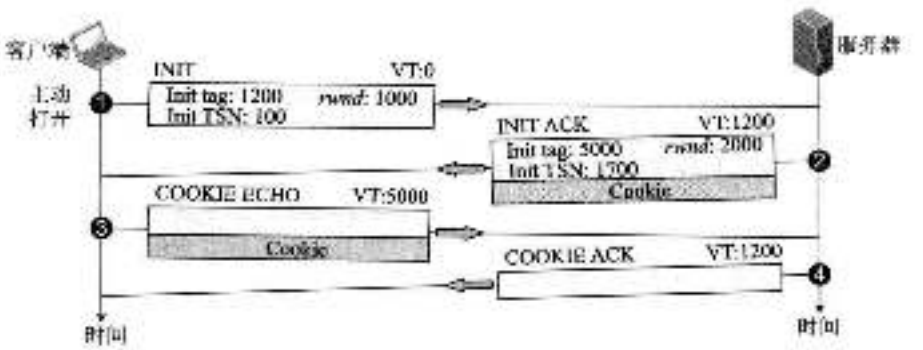


图 8-52　四次握手

通常情况下，四次握手如下：

1. 客户端发送第一个分组，这个分组包含 INIT 块。分组（通用头部中定义的）的认证标记（verification tag，VT）是 0，因为在这个方向（客户端到服务器）还没有定义认证标记。而在其他

方向（服务器到客户端）分组中使用的 INIT 标记包含一个初始标记。块也为这个方向定义了初始的 TSN 并以 rwnd 表示。rwnd 值通常在 SACK 块中赋值，是因为 SCTP 允许在第三个和第四个分组里包含数据块。服务器必须知道可用客户端的缓存大小。注意其他的块不可以由第一分组发送。

2. 服务器发送第二个分组，这个分组包含 INIT ACK 块。认证标记是 INIT 块中初始标记字段的值。这个块初始化了用于另一个方向的标记，为从服务器到客户端的数据流定义了初始 TSN，并且设置了服务器的 rwnd。定义 rwnd 的值用来允许客户端在第三个分组中发送数据块。INIT ACK 也发送了 cookie，这个 cookie 定义了此刻服务器的状态。稍后我们将简单地讨论 cookie。

3. 客户端发送第三个分组，这个分组包含 COOKIE ECHO 块。这是一个很简单的块，它不加改变地回显了服务器发送的 cookie。SCTP 允许在这个分组中包含数据块。

4. 服务器发送第四个分组。这个分组包含 COOKIE ACK 块，用于确认收到的 COOKIE ECHO 块。SCTP 允许在这个分组中包含数据块。

数据传输

关联的总体目的是在两个终端间传输数据。在关联建立之后，就可以进行双向数据传输。客户端和服务器都可以传输数据。与 TCP 类似，SCTP 支持捎带。

但是，TCP 和 SCTP 中的数据传输有很大的不同。TCP 以字节流的形式从进程中接收报文，但不能识别它们之间的边界。进程可能插入一些边界为它的对等结点使用。但 TCP 将这些标记看为文本的一部分。总之，TCP 获取每个报文并将它添加到缓存区中。一个段可以携带两个不同报文的多个部分。TCP 强制使用的唯一顺序系统是字节号。

另一方面，SCTP 识别并维护边界。进程中的每个报文被当作一个单元插入到数据块中，除非这个单元被分段（稍后讨论）。在这个意义上，SCTP 与 TCP 类似，有一个很大的优势：数据块之间是关联的。

从进程收到的一个报文成为了一个 DATA 块，如果有分段的话，会向报文中添加一个 DATA 块头部形成多个数据块。每个报文或报文的分段所形成的 DATA 块有一个 TSN。我们需要记住的是，只有 DATA 块使用 TSN，并且只有 DATA 块由 SACK 块确认。

多接口数据传输　我们之前讨论过 SCTP 中多接口的性能，它是在 UDP 和 TCP 中区分 SCTP 的特征。多接口允许在发送端和接收端定义多个通信 IP 地址。但是，这些地址中只有一个可以定义为初始地址，剩余的都是备用的地址。在关联建立的时候定义初始地址。有趣的一点是，一个端的地址由另一端决定。换言之，源端定义了目的端的初始地址。

数据传输默认使用目的端的初始地址。如果初始地址不可用，则使用备用地址。但是，进程经常可以重新定义初始地址，也可以通过向备用地址发送一个报文来完成明确的请求。进程也可以明确地改变当前关联的初始地址。

接着提出一个问题，那就是这个 SACK 被发送到哪里去？SCTP 命令将 SACK 发送到相应的发起 SCTP 分组的地址。

多流传递　SCTP 的一个有趣的特点是它可以区别数据传输和数据传递。SCTP 使用 TSN 号处理数据传输，即源端到目的端的数据块移动。数据块的传递则由 SI 和 SSN 控制。SCTP 支持多流。多流是指发送方进程可以定义不同的流，并且一个报文可以属于这些流中的一个。每个流都被分配一个流标识符（SI），它用唯一号定义流。但是，SCTP 在流中支持两种数据传递：有序（默认）和无序。在有序数据传递中，流中数据块使用流序列号（SSN）定义其在流中的顺序。当有序数据块到达目的端时，SCTP 负责报文按照 SSN 中定义的块顺序传送。因为一些块可能不按序到达而导致传递延迟。在无序数据传递中，流中的数据块有 U 标记集，但是它们的 SSN 字段值是被忽略的，不能使用 SSN。当无序数据块到达目的端时，SCTP 就将携带着块的报文传递给应用程序，而不等待其他的报文。大多数情况下，应用程序使用有序传递服务，但是偶尔一些应用程序需要发送紧急

数据，而必须不能按照顺序传递（回忆紧急数据和 TCP 中的紧急指针功能）。在这些情况下，应用程序可以定义无序传递。

分段　数据传输中另一个问题是分段（fragmentation）。尽管 SCTP 和 IP 中都使用这个术语（见第 4 章），但是 IP 中的分段和 SCTP 中的分段分别属于不同的层：前者属于网络层，而后者属于传输层。

SCTP 在从报文大小（封装 IP 数据报时）不超过 MTU（见第 4 章）的报文中创建数据块时保存从一个进程到另一进程的报文边界。IP 数据报根据增加的报文大小决定携带报文的大小（以字节为单位），有四种常用开销：数据块头部、必要 SACK 块、SCTP 通用头部和 IP 头部。如果总大小超过 MTU，那么报文需要被分段。

分段通过下列步骤在源 SCTP 进行：

1. 报文被分成满足需求大小的更小的分段。

2. DATA 块头部被加到每个携带不同 TSN 的分段中。TSN 需要存在于序列中。

3. 所有头部块携带相同的流标识符（SI）、相同的序列号（SSN）、相同的有效负载协议标识符和相同的 U 标记。

4. 合并 B 和 E 如下所示：

a）第一个分段：10。

b）中间的分段：00。

c）最后一个分段：01。

分段在目的端被重新合并。如果数据块到达时 B/E 位等于 1/1，那么它不是分段。接收端知道如何通过相同的 SI 和 SSN 重新合并所有的块。分段的数量取决于第一个分组和最后一个分组中的 TSN 号。

关联终止

与 TCP 类似，在 SCTP 中交换数据（服务器或客户端）的双方都可以关闭连接。但是与 TCP 不同的是，SCTP 不允许“半关闭”关联。如果一端关闭关联，另一端必须停止发送新的数据。如果终止请求后在发送队列中还剩余一些数据，则在将这些数据发送完后关闭关联。如图 8-53 所示，关联终止使用三个分组。注意，图中说明的情况是终止请求由客户端发起，但是终止请求也可以由服务端发起。

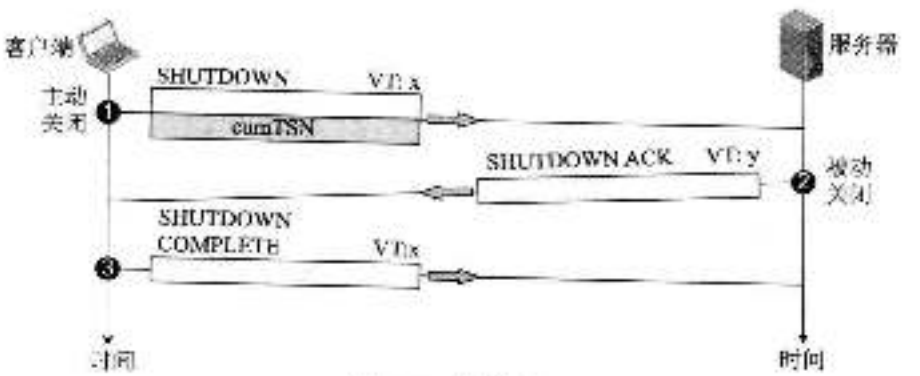


图 8-53　关联终止

流量控制

SCTP 中的流量控制（flow control）与 TCP 中的流量控制相似。在 TCP 中，我们只需要处理一种单位的数据，即字节。而在 SCTP 中需要处理两种单位的数据，即字节和块。rwnd 和 cwnd 的

值用字节表示，而 TSN 和确认用块表示。为了说明这个概念，我们做了一些不切实际的假设。假设网络中没有到达并且网络中没有差错。换言之，我们假设 cwnd 是无穷大并且没有分组丢失、延迟或者乱序到达。也假设数据传输是单向的。在以后的章节中纠正这个不切实际的假设。现在 SCTP 实现流量控制仍使用面向字节的窗口。但是为了更容易地理解这个概念，我们说明了块的一个术语缓冲区。

接收站

接收端有一个 buffer（序列）和三个变量。序列保存已接收的但还没有被进程读取的数据块。第一个变量保存最近接收的 TSN，即 cumTSN。第二个变量保存可用 buffer 大小，即 winSize。第三个变量保存最新的确认，即 lastACK。图 8-54 说明了在接收站的序列和变量。

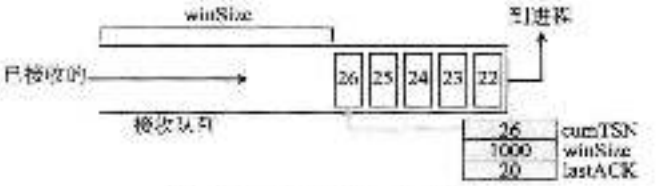


图 8-54　流量控制，接收站

1. 当站接收到一个数据块时，接收站将数据块保存在 buffer（序列）的末端并且在 winSize 中减去这个块的大小。这个块的 TSN 号被保存在 cumTSN 变量中。

2. 当进程读取一个块时，将读取的数据块从序列中移出并且在 winSize 中加上移出的数据块的大小（回收）。

3. 当接收端决定发送一个 SACK 时，接收端检查 lastACK 的值。如果小于 cumTSN，它则发送 TSN 号等于 cumTSN 的 SACK。发送的 SACK 也包括建议的窗口大小（值为 winSize）。然后 lastACK 的值被更新以保存 cumTSN 的值。

发送站

发送端有一个 buffer（序列）和三个变量：curTSN、rwnd 和 inTransit，如图 8-55 所示。我们假设每个块的长度是 100 字节。

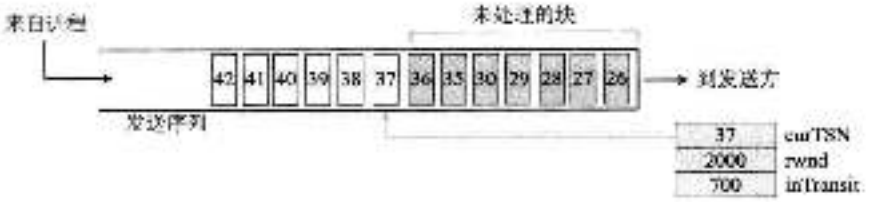


图 8-55　流量控制，接收站

buffer 保存进程产生的数据块，包括被发送的和准备发送的。第一个变量 curTSN，指的是下一个要发送的块。序列中的块的 TSN 小于已经发送但没有确认的值，则这些块是未处理的。第二个变量 rwnd，保存由接收端建议的最接近的值（字节）。第三个变量 inTransit，保存正在传输的字节数，字节已经发送但没有收到确认。发送端使用的过程如下：

1. 如果数据的大小小于或等于一个数（rwnd−inTransit），那么由 curTSN 指向的块就可以被发送。在发送块之后，curTSN 的值被加 1 并且指向下一个要发送的块。inTransit 的值增加了已发送块数据的大小。

2. 当接收到一个 SACK 时，如果块的 TSN 小于或等于 SACK 中累积的 TSN，则将这个块从序列中移出并且丢弃。发送端不必考虑更多。丢弃块的总大小是多少，inTransit 的值就减少多少。rwnd 的值随着 SACK 中建议窗口的值变化而更新。

差错控制

与 TCP 类似，SCTP 是一种可靠的传输层协议。它使用 SACK 块向发送端报告接收端 buffer

的状态。对于接收站和发送站，每一种实现使用不同的实体集和计时器。我们用一个非常简单的设计为读者说明这个概念。

接收站

在我们这个设计中，接收端保存所有已经到达的块，其中也包括乱序的块。但是，接收端为丢失的块预留空间，接收端丢弃重复的报文并且通过报告与发送端保持联系。图 8-56 说明了一个接收站的典型设计和在特定时间点上接收序列的状态。

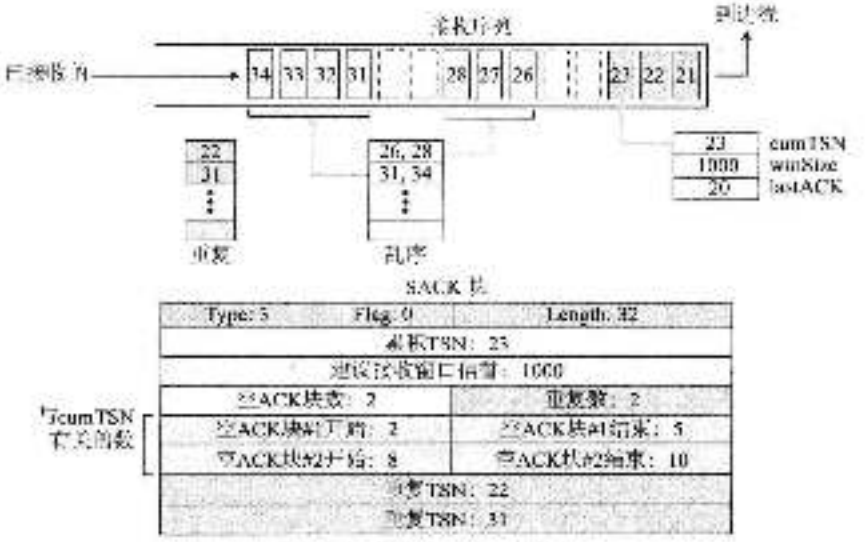


图 8-56　差错控制，接收站

最后的发送确认是数据块 20。可用窗口大小是 1000 字节。块 21 到块 23 被有序接收。第一个乱序块包含块 26 到块 28。第二个乱序块包含块 31 到块 34。一个变量保存 cumTSN 的值，一个变量队列保存乱序块的起点和终点，一个变量队列保存接收到的重复块。注意，不需要在序列中保存重复块。这些重复块将被丢弃。图也说明了 SACK 块，它向发送端发送用来报告接收端状态。无序块的 TSN 号与累积 TSN 有关（偏移）。

发送站

在发送站，我们的设计要求有两个 buffer（序列）：发送序列和重传序列，也使用三个变量：rwnd、inTransit 和 curTSN，和前面描述的相同。图 8-57 说明了一个典型设计。发送序列保存块 23 到块 40。块 23 到块 36 已经被发送，但是没有收到确认，这些块则是未处理的块。curTSN 指向下一个发送的块（37）。假设每个块是 100 字节。这意味着 1400 字节的数据（块 23 到块 36）在传输中。发送端在此刻有一个重传序列。当分组被发送时，重传计时器开始计时。一些实现中每个实体关联都使用一个单独的计时器，但简单起见我们仍然在每个分组只使用一个计时器。当一个分组在重传计时器溢出或有三个说明分组丢失的 SACK 到达时（在 TCP 中讨论过快速重传），分组中的块被放在重传序列上重新发送。这些块被认为是丢失的，而不是未处理的。这些块在重传队列中的优先级是最高的。换言之，下一次发送端发送块时，重传序列中发送的块将是块 21。

让我们看看发送端的状态是如何改变的。假设图 8-56 中的 SACK 到达图 8-57 的接收站。图 8-58 说明了新的状态。

1. 将 TSN 小于或等于 SACK 中 cumTSN 的块从发送队列或重传队列中移除。这些块不再是未处理的或被标记为重传。块 21 和块 22 从重传序列中移除，而块 23 从发送序列中移除。

2. 我们的设计中也移除了在间隙块中声明的发送队列中的块。但是在一些保守的实现中块和

应的 cumTSN 在到达之前都保存着。这是为了预防极少的偶然情况，如当接收端发现了一些乱序块的问题时。我们忽略了这些偶然的情况，所以块 26 到块 28，块 31 到块 34 从发送序列中移除。

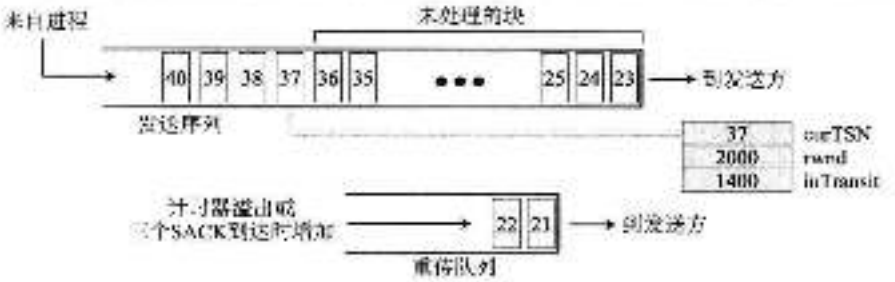


图 8-57　差错控制，发送站

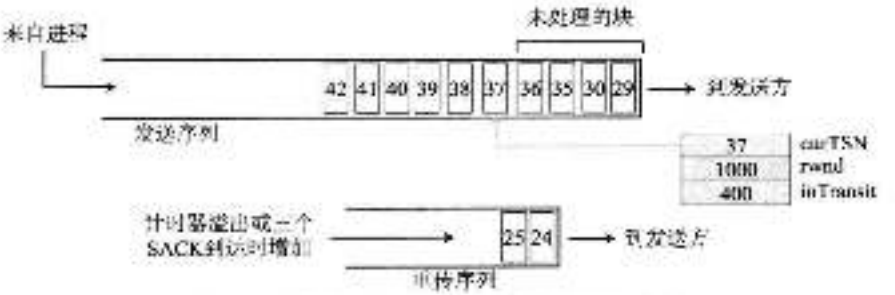


图 8-58　接收到 SACK 块之后发送站的新状态

3．重传块列表没有任何作用。

4．rwnd 的值被改变为 1000，如 SACK 块中建议的一样。

5．我们假设携带块 24 和块 25 的传输计时器溢出。这些块被放在重传序列中，新的重传计时器设置遵循 TCP 中讨论过的指数退避规则（exponential backoff rule）。

6．因为此时只有 4 个块在传输中，inTransit 的值变为 400。重传序列中的块不计数，因为假设这些块丢失了，不在传输中。

发送数据块

如果数据分组中的数据块在发送序列中的 TSN 大于或等于 curTSN，或数据块存在于重传序列，一个端可以在任何时刻发送该数据分组。传输序列有优先级。但是，包含在分组中的数据块或块的总大小不能超过值（rwnd – inTransit）。前的总大小不能超过 MTU，我们在之前讨论过。在我们之前的方案中，如果假设分组可以有三个块（由于 MTU 限制），那么重传序列中的块 24 和块 25，以及发送序列中准备发送的下一个块（块 37）可以被发送。注意，发送队列中未处理的块不能被发送。而重传队列中发送的块也可能因为超时而再一次重传。块 24、块 25 和块 37 被新的计时器控制。我们需要在这里提到，一些实现可能不允许将来自重传序列和发送序列的块混合（mixing）。在这个例子中，只有块 24 和块 25 可以在分组中被发送（数据块的格式在这本书的网站上）。

重传　为了控制丢失或丢弃的块，SCTP 与 TCP 类似，有两种策略：使用重传计时器和接收三次相同丢失块的 SACK。

- **重传**（Retransmission）。SCTP 使用重传计时器，计时器控制重传时间、分段确认的等待时间。SCTP 中计算 RTO 和 RTT 的过程与之前我们讨论过的 TCP 中的相同。SCTP 使用 RTT 样本（RTTM）、平滑 RTT（RTTS）、RTT 的偏差（RTTD）计算 RTO。SCTP 也使用 Karn 算法来避免确认不明确。注意，如果一个主机使用多个 IP 地址（多接口），必须计算单独 RTO 并且为每个路径保存。

- 四次丢失报告（Four Missing Reports）。无论什么时候发送端接收到四次空 ACK 信息的 SACK，其表明有一个或多个具体的数据块丢失，发送端需要将这些块当做丢失块并且将其加载到重传序列中。这个方法与 TCP 中的快速重传相类似。

产生 SACK 块

差错控制中的另一个问题是产生 SACK 块。产生 SCTP SACK 块的规则与 TCP 中 ACK 标记确认的规则相类似。我将规则总结如下：

1. 当一端向另一端发送一个 DATA 块时，必须包含一个 SACK 块，这个 SACK 块用来显示没有收到确认的 DATA 块。

2. 当一端接收到一个包含数据的分组但是并没有数据要发送时，它需要在一个特定的时间内（通常 500ms）确认接收到的分组。

3. 一端在接收到每个其他的分组后必须要发送至少一个 SACK。这个规则重复了第二条。

4. 当分组以乱序数据块方式到达时，接收端需要立即向发送端发送一个 SACK 块报告这个情况。

5. 一端接收的分组中有重复数据块而且没有新的数据块时，必须立刻通过发送一个 SACK 块报告这些重复数据块。

拥塞控制

与 TCP 类似，SCTP 是一种在网络中含有分组阻塞的传输层协议。SCTP 的设计者们使用的拥塞控制机制与 TCP 中使用的相同。

8.5　服务质量

因特网最初设计为尽力而为的服务，并且保证可预测的性能。尽力而为的服务通常适用于对延迟不敏感的通信中，如文件传输和电子邮件。通信量称为弹皮基（elastic），因为它在延迟条件下弹性工作。也称为有效比特率，因为应用程序可以根据有效比特率加速或者减速。

一些多媒体应用产生实时流量，而实时流量对延迟很敏感，所以需要有保证的和可预测性的性能。

服务质量（Quality of service，QoS）是一个网际交换的问题，这个问题指的是一系列保证网络向应用程序交付可预测性服务的性能的技术和机制。

8.5.1　数据流量特征

如果我们想为因特网应用提供服务质量，首先需要明确每个应用的需求。对象上认为流量的特征包括这四种：可靠性、延迟、抖动和带宽。让我们首先明确这几个特征，然后研究每种应用类型的需求。

定义

我们给出了上述四种特征通常的定义：

- **可靠性**（Reliability）。可靠性是指流量需要以将分组准确交付给目的端为目的的特征。缺少可靠性是指丢失分组或确认，其可以引起重传。然而，不同的应用程序对可靠性的敏感度不同。例如，电子邮件、文件传输和因特网访问与电话或语音会话相比较，可靠传输对于前者更重要。
- **延迟**（Delay）。流量的另一个特征是源端到目的端的延迟。再提一遍，不同的应用允许延迟的程度不同。在上个例子中，电话、语音会话、视频会话和远程登录需要最小的延迟，而延迟对于文件传输或电子邮件来说则不那么重要。
- **抖动**（Jitter）。分组从源端发送到目标端会发生不一样的延迟，这样的延迟变动就是抖动。例如，如果四个分组的发送时间分别是 0、1、2、3，而到达时间分别是 20、21、22、23。分组的延迟都一样，为 20。另一方面，如果上述四个分组到达的时间分别为 21、23、24 和 28，它们的延迟则不同。对于音频或视频类的应用，第一种情况是完全可以接受的，但是

第二种情况则不可以接受。如果所有分组或长或短的延迟都相等，这些应用对于这些延迟都无所谓。这种类型的应用不允许互动。

- **带宽**（Bandwidth）。不同的应用需要的带宽也不同。在视频会话中，我们需要每秒发送百万比特来刷新影屏，而在电子邮件中直到结束可能仅仅需要百万比特。

应用程序的敏感度

现在让我们讨论不同的应用对于什么样的流量特征敏感度高。表 8-9 给出了关于应用类型和相应敏感度的总结。

表 8-9　流量特征应用的敏感度

应　用	可 靠 性	延　迟	抖　动	带　宽
FTP	高	低	低	中
HTTP	高	中	低	中
音频点播	低	低	高	中
视频点播	低	低	高	高
网络电话	低	高	高	低
网络视频	低	高	高	高

对可靠性有高灵敏度的应用程序，我们需要进行差错检测并且丢弃损坏的分组；对延迟有高灵敏度的应用，我们需要确保分组在传输中低优先序列；对抖动有高灵敏度的应用，我们需要确保属于相同应用的分组经过网络传输后有相同的时延；最后，对于需要高带宽的应用，我们需要分配足够的带宽以确保分组不丢失。

8.5.2　流量分类

基于流量特征，我们可以对流量归类成组。每组都包括每个特征需要的等级。因特网社区还没有定义如此正式的分类。例如，如 FTP 之类的协议需要高等级的可靠性和中等级的带宽，但延迟和抖动的等级对于这类协议可能不太重要。

例 8.11　尽管因特网没有正式定义流量分类，但是 ATM 协议定义了。按照 ATM 规定，定义了五类服务。

a. 恒定比特率（Constant Bit Rate，CBR）。这类用来仿效电路交换。CBR 应用对 cell-delay 变化比较敏感。CBR 的例子有：电话业务、视频会话和电视。

b. 非实时可变比特率（Variable Bit Rate-Non Real Time，VBR-NRT）。这类中的用户以一个速率发送通信量，这个速率基于可用的用户信息并且随着时间而变化。一个例子是多媒体电子邮件。

c. 实时可变比特率（Variable Bit Rate-Real Time，VBR-RT）。这类与 VBT-NRT 类似，被设计为如交互式压缩视频这类对 cell-delay 变化敏感的应用使用。

d. 可用比特率（Available Bit Rate，ABR）。这类 ATM 服务提供基于速率的流量控制并且针对突发流量，如文件传输和电子邮件。

e. 未指定比特率（Unspecified Bit Rate，UBR）。这类包括其他的类，现在在 TCP/IP 中广泛应用。

8.5.3　通过流量控制提高 QoS

尽管在因特网中没有对流量正式的分类，IP 数据报中的 ToS 字段可以非正式地定义服务类型需要应用程序发送的数据报集。如果为某一应用类型的一个等级分配需要的服务，那么我们可以定义这个服务等级的一些规定。这些可以通过几种机制实现。

调度

在因特网中主要是在路由器中基于服务等级需求处理分组（数据报）。在路由器中，分组可能到

迟、遭受抖动、丢失或被分配所需带宽。好的调度技术以公平和适当的方式处理不同的流量。一些调度技术用来提高服务质量。在此我们讨论其中的三种：FIFO 队列、优先级队列和加权公平队列。

FIFO 队列

在先入先出队列（first-in, first-out（FIFO）queuing）中，分组在缓存（队列）中等待，直到结点（路由器）处理这个分组。如果平均到达时间大于平均处理时间，队列将被填满而新的分组在到达后将被丢弃。FIFO 队列就像我们必须在公交车站等待一辆公交车。图 8-59 是 FIFO 队列的概念图。图中说明了分组到达时间和离开时间之间的关系。来自不同的应用（大小不同）的分组到达队列后，经过处理，然后离开队列。大的分组当然可能需要更长的处理时间。图中，分组 1 和分组 2 需要三个单位的处理时间，但是分组 3 比较小只需要两个单位的处理时间。这意味着，分组到达队列时有一定的延迟，但是离开队列时又有不同的延迟。如果这些分组来自同一个应用，那么就会发生抖动。如果这些分组来自不同的应用，那么对于每个应用也会发生抖动。

FIFO 队列是因特网中默认的调度算法。这种队列可以保证的唯一一点是：分组以到达队列时的顺序离开队列。FIFO 队列可以区别分组分类吗？答案当然是不能。这种队列在因特网中不提供区分来自不同源分组的服务。FIFO 队列中都是在分组交换网络处理所有分组，不管分组属于 FTP、网络电话或电子邮件报文，也不管分组是否丢失、延迟或抖动。带宽的分配取决于一段时间内到达路由器的分组数。如果需要为不同的分组提供不同的服务，我们需要其他的调度机制。

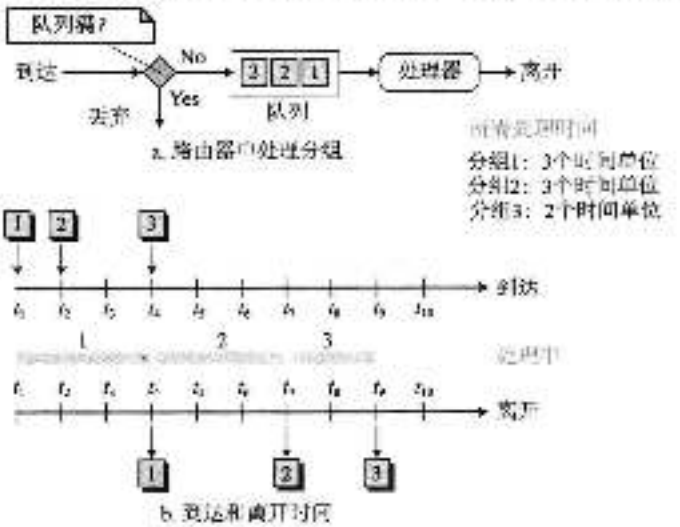


图 8-59　FIFO 队列

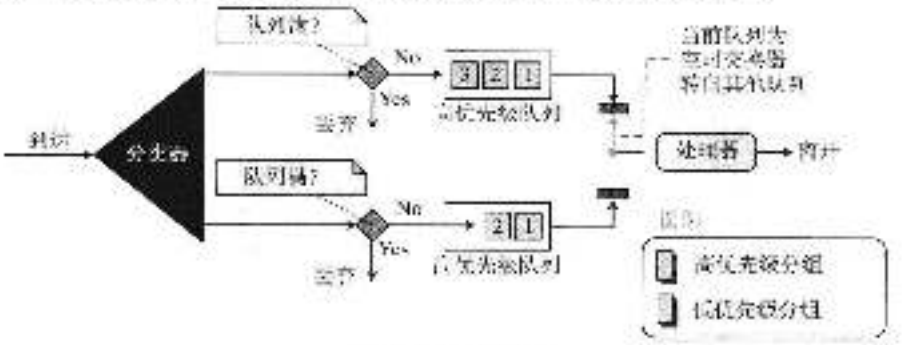
优先级队列

FIFO 队列的延迟通常会降低网络的服务质量。携带实时分组的帧可能要在携带一个小文件的帧后面等待很长的时间。为了解决这个问题，我们使用了多重队列和优先级队列。

在优先级队列（priority queuing）中，分组首先被分配优先级。每种优先级都有自己的队列。最高优先级队列中的分组最先被处理，而最低优先级队列中的分组最后被处理。注意，这个系统直到队列为空时才会停止服务。分组优先级由分组头部的特定字段决定：IPv4 头部的 ToS 字段，IPv6 中的优先级字段，分配给目的地址的优先号或分配给应用程序（目的端口号）的优先号等等。

图 8-60 说明了优先级队列的两个优先级（为了简单）。因为高优先级流量，优先级队列可以提供比 FIFO 队列更好的服务质量，例如通过优先级队列可以使多媒体到达目的端的延迟更小。但是，也有潜在的缺点。如果高优先级队列有一个连续不断的流量，那么低优先级队列中的分组将永远没

有机会被处理。这种情况叫做饿死（starvation）。严重的饿死可能导致低优先级队列中的一些分组丢失。在图中，高优先级队列中的分组在低优先级队列中的分组之前被发送。



a. 两组分组到达一个



b. 到达时间和离开时间

图 8-60　优先级队列

加权公平队列

一个更好的调度算法是加权公平队列（weighted fair queuing）。在这种技术中，分组仍然被分配给不同的类别并且加入到不同的队列中。然而，队列根据队列的优先级赋权值，高优先级意味着权值高。这个系统以循环方式处理每个队列中的分组，每个队列选取的分组数由每个队列中分组相应的权值决定。例如，如果权值有 3、2 和 1 三个，处理的分组中有三个分组来自第一个队列，两个来自第二个队列，一个来自第三个队列。这种方式就是带权值的公平队列。图 8-61 说明了这三类分组使用的技术。在加权公平队列中每类分组在每个时间周期获得一部分时间。换言之，分配部分时间用于服务每类分组，但是这部分时间由这类的优先级决定。例如在图中，如果路由器的吞吐量为 R，则高优先级的那类分组可能获得的吞吐量为 R/2，中优先级的分组可能获得的吞吐量为 R/3，而低优先级的分组可能获得的吞吐量为 R/6。这种情况在三类分组的分组大小相等时是正确的，但这是不可能发生的。分组大小的不可可能在为不同类分组分配相应时间部分时造成不平衡。

流量整形或监控

流量数量和速率的控制称为流量整形（traffic shaping）或流量监控（traffic policing）。第一个术语在流量离开网络时使用，而第二个术语在数据进入网络时使用。有两种技术可以整形或监控流量：漏桶算法和令牌桶算法。

漏桶算法

如果桶的底部有一个小孔，桶中的水就会以一个恒定的速率从孔中漏出。水漏出的速率不取决于水流入的速率，除非桶是空的。如果桶是满的，水就会溢出。流入速率可以变化，但是漏出速率

保持不变。网络中也与此类似，称为漏桶算法（leaky bucket）的技术可以解决突发性流量。突发性的数据块被保存在桶中并以平均速率发送。图 8-62 说明了漏桶算法及此算法的效果。

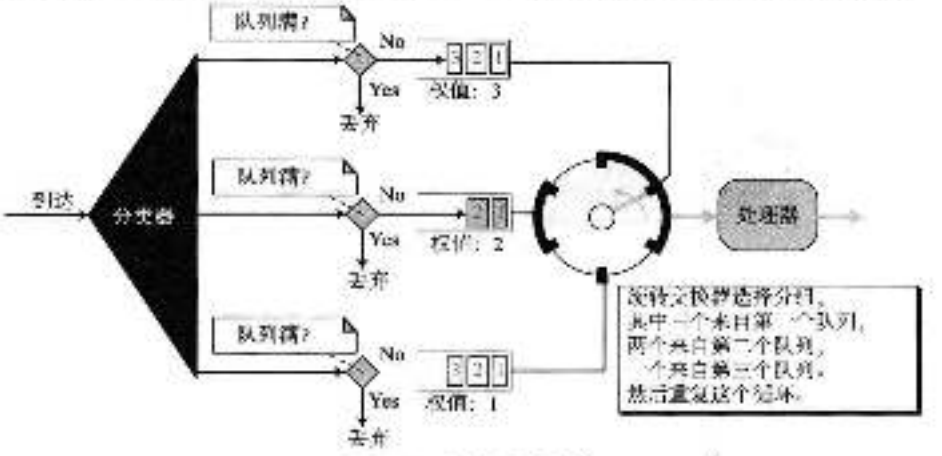


图 8-61　加权公平队列

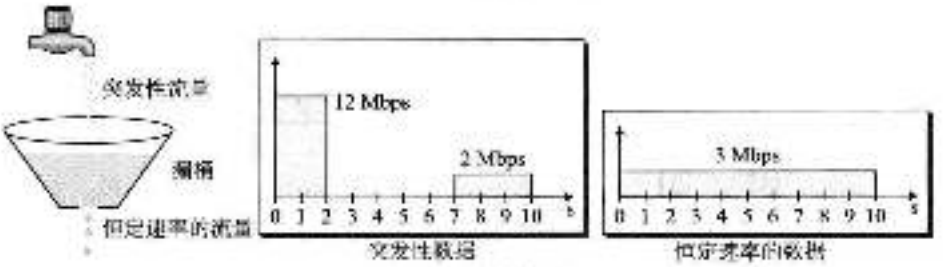


图 8-62　漏桶算法

在图中我们假设网络为主机提供 3Mbps 的带宽。漏桶算法使输入流量按照带宽定形。图 8-62 中前两秒主机以 12Mbps 的速率发送突发性数据，总共 24Mb 数据。接下来 5 秒主机不发送，然后以 2Mbps 的速率发送 3 秒，总共 6Mb 数据。最后主机 10 秒合计发送 30Mb 数据。漏桶算法这样处理这些流量，在相同的 10 秒内以 3Mbps 的速率发送数据。如果不使用漏桶算法，在一开始可能会消耗的带宽比为主机预留的更多，从而破坏网络。我们也可以认为漏桶算法可以预防拥塞。

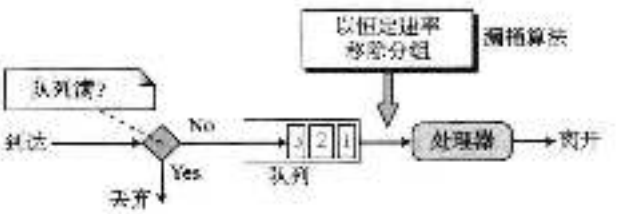


图 8-63　漏桶实现

图 8-63 说明了一种简单的漏桶算法实现。分组以 FIFO 队列发送。如果流量由恒定大小的分组组成，每个时钟进程从这个队列中移除固定数量的分组。如果流量由长度可变的分组组成，那么固定输出速率必须基于字节或比特的数量。

长度可变分组算法如下：

1. 为时钟初始化一个计数器 n。

2. 如果 n 大于分组的大小，则发送分组并且将计数器减少为分组的大小。重复这个步骤直到计数器的值小于分组大小。

3. 重新设置计时器的值 n，转到第一步。

漏桶算法将突发性流量整形为恒定的速率（平均数据传输率），如果桶满则可生丢弃分组。

令牌桶算法

漏桶算法非常有局限性。它不适用于空闲的主机。例如，如果主机在一段时间内不发送数据，那么它的“漏桶”就变为空。如果主机有突发性数据，漏桶却只允许以平均速率发送。主机空闲的时间不能被计算为发送时间。换言之，令牌桶算法（token bucket）允许空闲主机以令牌的形式计算未来的信誉。

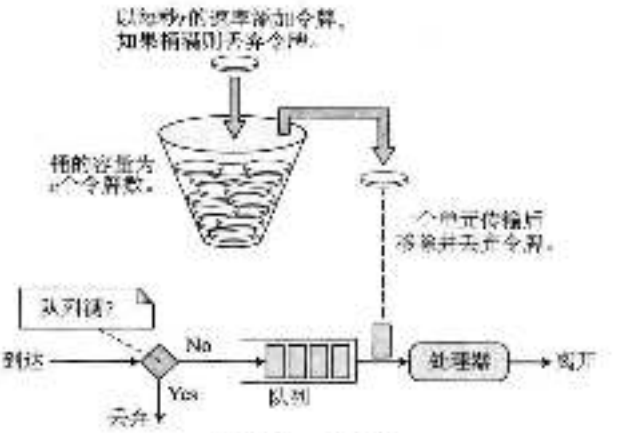


图 8-64　令牌桶

假设桶的容量为 c 个令牌并且以每秒 r 个令牌的速率进入桶。每发送一个数据单元系统就移除一个令牌。任意时间间隔长度 t 进入网络的单元最大数如下所示：

$$\text{最大分组数} = rt + c$$

令牌桶算法的最大平均速率如下所示：

$$\text{最大平均速率} = (rt + c)/t \text{ 分组/秒}$$

这意味着，令牌桶限制网络中分组的平均速率。图 8-64 说明了这个思想。

例 8.12　假设桶的容量为 10 000 个令牌并且以每秒 1000 个令牌的速率加入。如果系统空闲 10 秒（或更久），桶被装满 10 000 个令牌。桶满后再加入的令牌将会丢失。最大平均速率如下所示：

$$\text{最大平均速率} = (1000t + 10\,000)/t$$

令牌桶可以简单地通过计数器实现。计数器被初始化为 0。每加入一个令牌，计数器加一。每发送一个令牌，计数器减一。当计数器为 0 时，主机不能发送数据。

令牌桶允许规定突发最大速率的突发性流量。

合并漏桶和令牌桶算法

这两种技术可以结合用来信任空闲的主机同时控制流量。漏桶算法在令牌桶算法之后应用，漏桶的速率需要快于令牌桶中丢失的速率。

资源预留

数据流需要资源，如缓存、带宽、CPU 时间等等。如果事先预留这些资源，就会提高服务质量。下面我们讨论一种称为综合服务的 QoS 模型，其非常依赖通过资源预留提高服务质量。

进入许可控制

进入许可控制指的是一种机制，这种机制通过使用路由器或交换机接收或拒绝基于称为流规格（flow specifications）的预先定义参数的流量。在路由器接收处理流量之前，先检查流规格并判断其能力是否可以处理新的流。它考虑到要提供和之前别的流量一样的带宽、缓存大小、CPU 速度等。在 ATM 网络中，进入许可控制称为是连接允许控制（Connection Admission Control，CAC），它是控制拥塞策略中的主要部分。

8.5.4　综合服务（IntServ）

不管用户需要什么服务，传统因特网只为所有用户提供尽力而为的交付服务。但是一些应用的功能需要较小的带宽数（如实时音频和视频）。为了给不同的应用提供不同的 QoS，IETF（第 1 章讨论过）开发了综合服务（integrated services，IntServ）模型。这种模型是基于流的体系结构，要为给定的数据流明确预留资源，如带宽。换言之，不管是哪种应用类型（数据传输、网络电话或视频点播），

这个模型都被认为是一个特定情况下应用的特定需求。重要的是应用需要什么资源，而不是做什么。

这种类型基于三种方案：

1. 首先根据分组需求对服务将其分类。

2. 模型根据流特征调度前向分组。

3. 设备（如路由器）通过进入许可控制判断设备是否有能力（处理流的可用资源），然后做出承诺。例如，如果一个应用程序需要非常高的数据传输率，但这个链接的路由器不能提供如此高的速率，那么路由器拒绝其进入。

在讨论这种模型之前，我们需要强调这个模型是基于流的，满足所有的需求之后才可以开始流。这意味着在网络层我们需要面向连接的服务。在连接建立阶段需要向所有路由器报告需求并且获得路由器的许可（进入许可控制）。但是，因为 IP 是一种普通的无连接协议，我们需要在使用这个模型之前运行另一种在 IP 上层的面向连接的协议。这种协议称为资源预留协议（Resource Reservation Protocol，RSVP），稍后讨论。

综合服务是为 IP 设计的基于流的 QoS 类型。

路由器根据流量特征标记模型中的分组。

流规格

综合服务是基于流的。为了定义特定的流量，源端需要定义流规格，其由两部分组成。

1. 资源规格（resource specification，Rspec）。Rspec 定义流需要预留的资源（缓存、带宽等）。

2. 通信量规格（traffic specification，Tspec）。Tspec 定义流中通信量的特性描述，之前讨论过。

准入控制

路由器接收来自应用程序的流量特性描述之后决定准许或拒绝服务。这个决定基于路由器先前的协商和当前的可用资源。

服务类型

综合服务定义了两种服务：确保服务和控制负载服务。

确保服务类型

这种类型的服务设计是为了不容忍的实时应用设计的，为提供限制端到端的分组延迟小于某个固定值。端到端的延迟是路由器中的延迟、媒介中的传输延迟和工作机制延迟的总和。只有第一种延迟——路由器中的延迟是有保证的。这种类型的服务保证分组在某一传递时间内到达，并且如果流量保存在 Tspec 边界之内就不会被丢弃。我们可以认为确保服务是定量服务，其中端到端的延迟和数据传输率必须由应用程序确定。通常确保服务是实时应用程序（网络电话）必需的。

控制负载服务类型

这种类型的服务为容错型的自适应实时多媒体应用设计，用来为允许延迟的应用提供服务，这种应用对超负荷网络很敏感并且有丢失分组的危险性。关于这种应用的例子有文件传输、电子邮件和因特网访问。控制负载服务是一种定性服务，这种服务的应用请求的分组绝大部分被成功地传递给接收端。没有传递的分组的比特率和传输媒体的基本丢失率相当。

资源预留协议（RSVP）

我们之前讨论过综合服务模型需要面向连接的网络层。因为 IP 是无连接的协议，所以设计了一个运行在 IP 上层的面向连接新的协议。我们在第 4 章讨论过，面向连接协议需要连接建立阶段和连接终止阶段。在讨论 RSVP 之前我们需要强调，RSVP 是从综合服务模型中分离出来的独立协议，将来它可能在其他的模型中使用。

多播树

RSVP 与其他面向连接的协议的不同之处是它基于多播通信。但是，RSVP 也可以用于单播，

因为单播只是多播组中只有唯一一个成员的一个特例。如此设计的原因是为了使 RSVP 能够为所有种类的流提供资源预留，包括经常使用多播的多媒体。

基于接收端的预留

RSVP 中是由接收端发起的预留而不是发送端。这个策略匹配其他的多播协议。例如，在多播路由协议中是接收端决定加入或者离开多播组，而不是发送端。

RSVP 报文

RSVP 有几种类型的报文。但是，我们只讨论其中的两种：Path 和 Resv。

- Path 报文。之前讨论过 RSVP 中流是由接收端发起预留。路径需要预留，但是在预留成功之前接收端不知道分组传输的路径。为了解决这个问题，RSVP 使用了 Path 报文。Path 报文包括从发送端到所有接收端途径的多播路径。顺便说一下，Path 报文为接收端保存了必要的信息。Path 报文在多播环境下发送，路径偏离时产生新的报文。图 8-65 说明了 Path 报文。

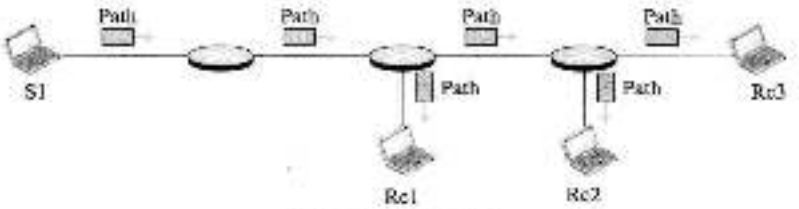


图 8-65　Path 报文

- Resv 报文。接收端接收到 Path 报文后向发送端发送一个 Resv 报文（上行），路径中的 RSVP 路由器在接收到该 Resv 报文后通过准入控制模块检测是否有足够的资源。如果路径中的路由器不支持 RSVP，路由分组基于我们之前讨论过的尽力而为传递方法。图 8-66 说明了 Resv 报文。

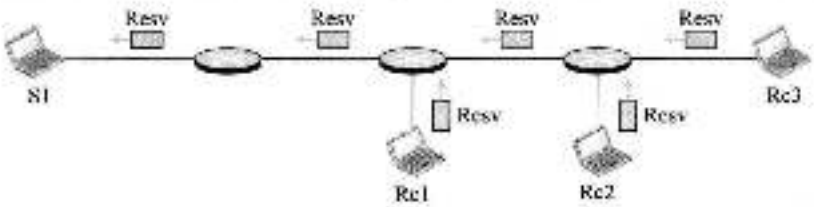


图 8-66　Resv 报文

预留合并　在 RSVP 中，流中资源不是为每个接收端单独保留，而是合并之后预留。在图 8-67 中，Rc3 请求 2Mbps 的带宽，而 Rc2 请求 1Mbps 的带宽。需要进行资源预留的路由器 R3 合并了两种请求。预留的资源是两种请求中较大的 2Mbps，因为 2Mbps 的预留资源满足前面任何一种请求。相同的情况适用于 R2。可能有读者会问为什么都属于一个单流的 Rc2 和 Rc3 请求的带宽值会不同。答案是，在多媒体环境中不同的接收端可能处理的质量等级不同。例如，Rc2 可能会仅仅以 1Mbps 接收视频（低质量），而 Rc3 则以 2Mbps 接收视频（高质量）。

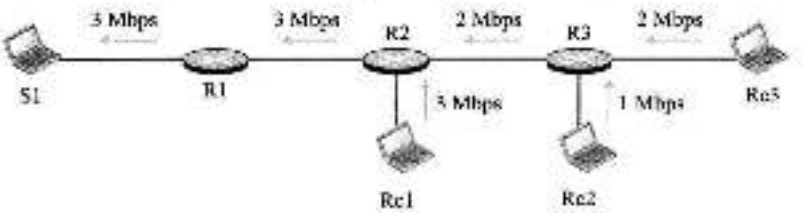


图 8-67　预留合并

预留方式　当有多个流时，路由器需要为每个流预留相应的资源。RSVP 定义三种类型的预留方式：通配预留方式（WF）、固定预留方式（FF）和共享显示预留方式（SE）。

- 通配预留方式（Wild Card Filter Style）。这种方式中，路由器为所有的发送端预留一个单一的资源。预留根据最大的请求。这种方式在来自不同发送端的流不同时使用。
- 固定预留方式（Fixed Filter Style）。这种方式中，路由器为每个流都预留相应的带宽。也就是说如果有 n 个流，就需要发起 n 个不同的预留。这种方式在同一时间流来自多个发送端时使用。
- 共享显示预留方式（Shared Explicit Style）。这种方式中，显示指定相对于哪些发送端来预留资源。

软状态　预留信息（状态）保存在每个结点中，并且需要周期性刷新。这称为软状态（soft state），相对于其他虚拟电路协议中使用的硬状态（如 ATM），软状态中维持流的信息直到它被清除。默认刷新间隔（软状态预留）一般为 30 秒。

综合服务的问题

综合服务中至少有两个问题可能阻碍它在因特网中全面实现：可扩展性和服务类型限制。

可扩展性

综合服务模型需要每个路由器为每个流保留信息。因特网还在不断壮大，这就是个严重的问题。对于中心路由器来说保留信息是特别麻烦的，因为中心路由器设计首先用来高速转发分组而不是处理信息。

服务类型限制

综合服务模型只提供两种类型的服务：确保服务和控制负载服务。那些反对这一模型的认为应用可能需要比这两种类型更多的服务。

8.5.5　区分服务（DiffServ）

这个模型也称为 DiffServ，应用把分组按照其优先级标记成不同的类别。使用各种队列策略的路由器和交换机转发分组。这种模型由 IETF（因特网工程任务部）引进用于处理综合服务中的问题。两点根本变化如下：

1. 三要处理过程从网络中心移动到网络边缘上，这解决了扩展性问题。路由器不用必须保存流信息。应用或主机在每次发送分组时定义需要的服务类型。

2. 基于流的服务更改为基于类别的服务。路由器基于分组中定义的服务类别转发分组，而不是基于流。这解决了服务类型限制问题。我们可以基于应用需求定义不同类别的类型。

区分服务是设计为 IP 使用的基于类别的 QoS 模型。
在这个模型中把分组按照其优先级标记。

DS 字段

在 DiffServ 中，每个分组都包含一个字段称为 DS 字段。这个字段的值由网络中的主机或第一个路由器指定的边界路由器设置。IETF 打算将当前 IPv4 中的 ToS（服务类型）字段或 IPv6 中的优先级类别字段替换为 DS 字段，如图 8-68 所示。

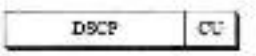


图 8-68　DS 字段

DS 字段包括两个子字段：DSCP 和 CU。其中 DSCP（被区分的服务编码点，Differentiated Services Code Point）是 6 位的子字段，定义了逐跳行为（per-hop behavior，PHB）。2 位的 CU（当前不使用）子字段不在当前使用。

DiffServ 可用结点（路由器）使用 6 位的 DSCP 作为索引，指向定义了当前被处理分组的分组处理机制的表格。

逐跳行为

DiffServ 模型为接收分组的每个结点定义了 PHB。但目前为止定义了三种 PHB：DE PHB、EF PHB 和 AF PHB。

- DE PHB。默认（Default，DE）PHB 与尽力而为交付相同，兼容 ToS。
- EF PHB。加速转发（Expedited Forwarding，EF）PHB 提供以下服务：

a. 低丢失率。

b. 低延迟。

c. 确保带宽。

这相当于在源端和目的端之间建立一条虚拟连接。

- AF PHB。确保转发（Assured Forwarding，AF）PHB 在类别负载不超过结点负载属性时发送很有保证的分组。网络中的用户需要注意到一些分组可能会丢失。

负载调节器

为了实现 DiffServ，DS 结点使用了负载调节器，如度量器、标记器、整形器和丢失器，如图 8-69 所示。

- 度量器。度量器用来测量输入的流是否符合预先协商的负载特性。度量器也向其他组件发送这个结果。度量器可以使用多种工具测量这个特性，如令牌桶。
- 标记器。标记器标记分组，可以标记为尽力而为交付（DSCP:000000），也可以基于从度量器接收的信息向下标记。向下标记（降低流的类别）在流不符合属性时使用。标记器不能向上标记分组（提升类别）。
- 整形器。当分组不符合预先协商的特性时，整形器通过从度量器接收到的信息重新定形负载。
- 丢失器。当分组不符合预先协商的特性时丢失该分组，丢失器可以看成是一个缓冲区大小为 0 的整形器。

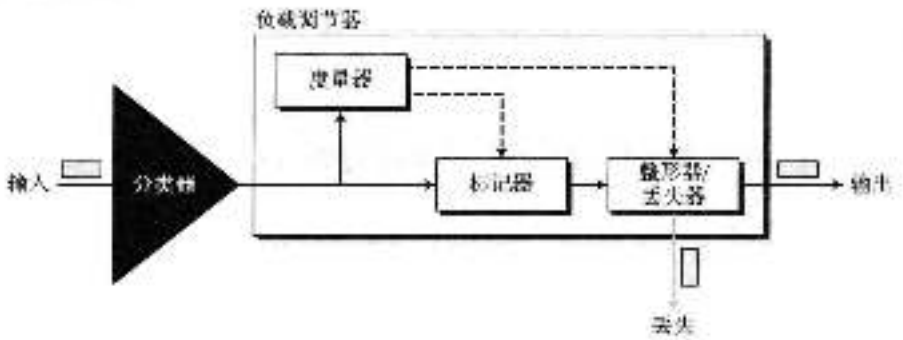


图 8-69　负载调节器

8.6　章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。在本书末列出了方括号中的参考资料。

书籍

覆盖多媒体与服务质量的一些书籍包括：[Com 06]、[Tan 03]和[G & W 04]。

RFC

一些 RFC 列出了与本章讨论主题有关的最新知识，它们包括 RFC 2198、RFC 2250、RFC 2326、

RFC 2475、RFC 3246、RFC 3550 和 RFC 3551。

小结

我们将压缩分为两大类：无损压缩和有损压缩。在无损压缩中保存了数据的完整性，因为压缩和解压缩算法之间是完全相反的，在这个过程中没有数据丢失。有损压缩不能保证数据的精确度，但是优点是我们可以进一步减少压缩后数据的大小。

音频/视频文件可以下载下来为以后使用（流式存储音频/视频），也可以通过因特网向客户广播（流式实况音频/视频）。因特网可以用于实时音频/视频交互。音频和视频通过因特网发送之前需要数字化。我们可以用一个 Web 服务器，或一个 Web 服务器的元文件，或一个媒体服务器，或一个媒体服务器和 RTSP 下载流式音频/视频文件。

分组交换网络中实时数据需要保存会话中分组间的时间关系。接收端连续分组之间的缝隙引起的现象称为抖动。抖动可以通过使用时间戳和选择正确的回放时间控制。

网络电话是一个实时互动式的音频/视频应用。会话初始化协议（Session Initiation Protocol，SIP）是一种建立、管理和终止多媒体会话的应用层协议。H.323 是一种 ITU 标准，它允许接入公共网络的电话与接入因特网的电脑通信。

实时多媒体负载不仅需要 UDP，而且也需要实时传输控制协议（Real-Time Transport Protocol，RTP）。RTP 处理时间戳、排序和混合。实时传输控制协议（Real-Time Transport Control Protocol，RTCP）提供流控制、数据质量控制和向源端回馈。新的传输层协议 SCTP 设计用来处理多媒体应用，如网络电话。

调度、流量整形、资源预留和准入控制这几种技术用来提高服务质量（quality of service，QoS）。调度的三种技术是 FIFO 队列、优先级队列和加权公平队列。流量整形的两种技术是漏桶算法和令牌桶算法。综合服务是为 IP 设计的基于流的 QoS 模型。资源预留协议是帮助 IP 创建流并且发起资源预留的一个独立的协议。区分服务是为 IP 设计的基于分类的 QoS 模型。

8.7　习题集

测试题

本章的交互式测试题请参见这本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q8-1　词典编码中，如果报文中有 60 个字符，问压缩算法的迭代次数是几次？请解释。

Q8-2　词典编码中，在进程中建立的所有字典入口应该用于编码还是解码？

Q8-3　字母表中有 20 个字符，哈夫曼树中的叶子结点数有几个？

Q8-4　下列代码是即时代码（instantaneous code）吗？请解释。

00　01　10　11　001　011　111

Q8-5　假设报文由四个字符（A、B、C 和 D）组成，其出现概率相同。这个报文的哈夫曼编码表是什么？这个编码会使发送的位数最少吗？

Q8-6　算术编码中，两种不同的报文可以以相同间隔编码吗？请解释。

Q8-7　预知编码中，DM 和 ADM 之间的区别是什么？

Q8-8　预知编码中，DPCM 和 ADPCM 之间的区别是什么？

Q8-9　如果最大的量化值如下，比较每个 PCM 样本和 DM 样本变换的位数。

a. 12　　b. 30　　c. 50

Q8-10　回答下列关于预知编码的问题：

a. DM 编码中的斜率过载失真和颗粒噪音失真是什么？

b. 解释在 ADM 编码中怎么解决上述问题。

Q8-11 DM 和 DPCM 的不同点是什么？

Q8-12 用 LPC 方法压缩一个语音信号的问题是什么？

Q8-13 变换编码中，当发送端向接收端发送 M 矩阵时，需要发送用于计算的 T 矩阵吗？试解释原因。

Q8-14 JPEG 中，如果图像只有三原色中的一种或两种，那么每个像素需要的大小等比 24 位小吗？试解释原因。

Q8-15 JPEG 中，为什么量化矩阵中 Q（m，n）的值不相同？也就是说，为什么 M（m，n）中的每个元素不是除以固定的值，而是除以不同的值？

Q8-16 试解释，与使用 Q90 相比，为什么使用 Q10 会有更好的压缩速率和相对较差的图像质量（见图 8-18）。

Q8-17 试解释在 JPEG 中的量化阶段，为什么需要对除后的结果进行四舍五入。

Q8-18 在量化阶段我们将 M 矩阵的每个元素除以 Q 矩阵的相应元素，而在反量化阶段又将矩阵乘以相同的值。试解释为什么即便这样也认为 JPRG 中量化和反量化阶段是一个有损过程。

Q8-19 多媒体通信中，假设发送端使用 JPEG 将一个图像进行编码，但是潜在的接收端只能解码以 GIF 编码的图像。两端可以交换多媒体数据吗？

Q8-20 流式存储音频/视频中，第一种方法（图 8-24）和第二种方法（图 8-25）相比有什么不同？

Q8-21 流式存储音频/视频中，第二种方法（图 8-25）和第一种方法（图 8-26）相比有什么不同？

Q8-22 流式存储音频/视频的第四种方法中 RTSP 的规则是什么？

Q8-23 实时音频/视频和实时交互式音频/视频的主要区别是什么？

Q8-24 当我们点播音频/视频时，会使用哪种多媒体通信：流式存储音频/视频、流式实况音频/视频还是实时交互式音频/视频？

Q8-25 图 8-26 中的 Web 服务器和多媒体服务器可以运行在不同的主机上吗？

Q8-26 在实时交互式音频/视频中，如果分组在预定的回放时间之后到达接收站时会发生什么？

Q8-27 假设我们需要发送 2 位的数据字。下列码字可以纠正单独一位的错误吗？

00000　　01011　　10101　　11110

Q8-28 如果有多个分组丢失，图 8-33 中的交错还运行吗？

Q8-29 如果有多个分组丢失，图 8-34 中的机制仍然运行吗？

Q8-30 假设我们设计一种协议的分组足够大，一个分组可以携带所有的实况数据块或实时多媒体数据流。我们还需要块的序列号或时间戳吗？试解释原因。

Q8-31 试解释为什么 RTP 不能当传输层协议使用，除非运行在另一种传输层协议的上层，比如 UDP。

Q8-32 TCP 和 RTP 中都使用序列号。这两种协议中序列号的作用一样吗？试解释原因。

Q8-33 如果 RTP 不提供相应的服务，UDP 可以在实时交互式多媒体应用程序中使用吗？

Q8-34 如果捕获的 RTP 分组中大部分的 RTP 头都大小为 12 字节，试解释原因。

Q8-35 多媒体数据的编码和解码也是通过 RTP 完成的吗？试解释原因。

Q8-36 假设一个图像通过 10 个 RTP 分组从发送端发送到目的端，可以将前五个分组用 JPEG 编码而后五个分组用 GIF 编码吗？

Q8-37 UDP 不建立连接，怎样把携带在不同 RTP 分组中的不同数据块重组在一起？

Q8-38 假设一个应用程序在一个 RTP 会话中使用单独的音频和视频流，则每个 RTP 分组中使用的 SSRC 和 CSRC 有多少？

Q8-39 我们可以认为 UDP 加 RTP 就是 TCP 吗？

Q8-40 为什么 RTP 需要其他协议的服务（RTCP），而 TCP 则不需要？

Q8-41 SIP 需要使用 RTP 服务吗？试解释原因。

Q8-42 我们提到过，SIP 是用来在呼叫者和被呼叫者之间提供信令机制的一种应用层程序。这个通信中哪一方是客户端，哪一方是服务器端？

Q8-43 本章中讨论过音频中使用 SIP。将其用在视频上会有哪些缺点？

Q8-44 假设双方需要使用 RTP 服务建立 IP 电话业务。如何定义每个方向上 RTP 使用的两个临时端口号？

Q8-45 在单播会话或多播会话情况下，哪种发送端接收来自会话的 RTCP 分组后处理反馈时更容易一些？

Q8-46 在无线环境中可以将 RTP/RTCP 和 SIP 合并使用吗？试解释原因。

Q8-47 做一些研究，并找出 SIP 是否可以通过现代电话装置提供下列服务。

a. 来电显示　　b. 通话保持　　c. 多方通话

Q8-48 试解释在因特网电话中，Bob 不在办公室也不在家（不知道 Bob 在哪里）时 Alice 如何可以呼叫到 Bob？

Q8-49 你认为 H.323 实际上与 SIP 一样吗？它们之间的区别是什么？对它们做一个比较。

Q8-50 H.323 也可以用于视频吗？

Q8-51 在一个银行支行中有两个柜员，一个柜员专门服务商业客户，另一个柜员服务老顾客。这个例子是优先级队列吗？试解释原因。

Q8-52 在一个银行支行中，为了使顾客等候的队伍更短一些，管理者让顾客分二列等候。每个柜员只可以服务每个队列中的一个顾客。问这是队列机制中的哪种类型？

Q8-53 在一个银行支行中只有一个柜员，但是有两列顾客，一列是商业顾客，另一列是老顾客。只有当商业顾客队伍中没有人时柜员才服务老顾客。问这是队列机制中的哪种类型？试解释原因。

Q8-54 在一个银行支行中只有一个柜员，但是有两列顾客，一列是商业顾客，另一列是老顾客。柜员服务两名商业客户然后服务一名老客户。问这是队列机制中的哪种类型？

思考题

P8-1 给出下列报文。给出使用行程长度编码的压缩数据。

AAACCCCCCBCCCCDDDDDAAAABBB

P8-2 给出下列报文。给出使用第二个版本的行程长度编码的压缩数据，这种编码将数表示为 4 位的二进制数。

10000001000001000000000000010000001

P8-3 在词典编码中，报文如下。你可以很容易得找到编码吗？（报文字母表只有一个字符）

a. "A"　　b. "AA"　　c. "AAA"

d. "AAAA"　　e. "AAAAA"　　f. "AAAAAA"

P8-4 在 LZW 编码中，给出报文 "AACCCBCCDDAB"。

a. 将这个报文编码（见图8-2）。

b. 如果压缩使用8位表示一个字符，4位表示一个十六进制数，计算压缩比是多少。

P8-5 在 LZW 编码中，给出编码 "0026163301"。假设字母表由 "A"、"B"、"C"、"D" 四种字符组成，这个编码解码后是什么。（见图 8-3）。

P8-6 给出报文 "AACCCBCCDDAB"，其中符号的概率是 P(A) = 0.50，P(B) = 0.25，P(C) = 0.125 和 P(D) = 0.125。

a. 使用哈夫曼编码将报文进行编码。

b. 如果编码时每个字符由8位表示，计算其压缩比。

P8-7 哈夫曼编码中，给出编码表如下：

A → 0　　B → 10　　C→110　　D→111

收到的编码为 "00110110011110111111010"，请给出其原始报文。

P8-8 给出报文 "ACCBCAAB*"，其中字符的概率是 P(A) = 0.4，P(B) = 0.3，P(C) = 0.2 和 P(*) = 0.1。

a. 使用精度为10个二进制数的算术编码，问压缩后的数据是什么。

b. 如果报文中的字符由8位表示，计算其压缩比。

P8-9 算术编码中，假设我们接收到的编码是 100110011。如果字母表由四种字符组成，并且其概率是 P(A) = 0.4，P(B) = 0.3，P(C) = 0.2 和 P(*) = 0.1，问原始数据是什么。

P8-10 在预知编码中，假设给定样本（x_n）如下：

n	1	2	3	4	5	6	7	8	9	10	11
x_n	15	24	46	60	45	32	30	40	30	27	20

a. 如果使用DM，说明发送的编码报文是什么。其中 $y_0 = 10$ 且 $\Delta = 8$。

b. 通过计算 q_n 的值，对于A可以得到什么结论？

P8-11 预知编码中，给定下列编码。如果使用 DM，说明如何计算每个样本的重建值 y_n。其中 $y_0 = 8$，$\Delta = 6$。

n	1	2	3	4	5	6	7	8	9	10	11
C_n	1	0	0	1	0	1	1	0	0	1	1

P8-12 预知编码中，假设给定样本 x_n 如下。如果使用 ADM，说明发送的编码报文是什么。其中 $y_0 = 10$，$A_1 =$

4, $M_1 = 1$。如果 $q_n = q_{n-1}$（q_n 没有变化），则 $M_n = 1.5 \times M_{n-1}$，否则 $M_n = 0.5 \times M_{n-1}$。

n	1	2	3	4	5	6	7	8	9	10	11
s_n	15	15	15	17	20	20	18	16	16	17	18

P8-13 假设给定编码如下。如果使用 ADM，说明如何计算每个样本的重建值 y_n。其中 $y_1 = 20$, $\Delta_1 = 4$, $M_1 = 1$。如果 $q_n = q_{n-1}$（q_n 没有变化），则 $M_n = 1.5 \times M_{n-1}$，否则 $M_n = 0.5 \times M_{n-1}$。

n	1	2	3	4	5	6	7	8	9	10	11
C_n	1	1	1	0	0	1	1	0	0	1	1

P8-14 一维 DCT 中，如果 $N = 1$，矩阵变换变为简单的乘法。也就是说，$M = T \times p$。其中 T、p 和 M 是替代矩阵的数（标量值）。这个例子中 T 的值是多少？

P8-15 DCT 中，变换编码中 $T(m, n)$ 的值经常在 -1 和 1 之间。试解释原因。

P8-16 说明在变换编码中接收端如何将接收的 M 矩阵变换为原始的 p 矩阵。

P8-17 分别计算 $N=1$、$N=2$、$N=4$ 和 $N=8$ 时用于 DCT 的 T 矩阵。

P8-18 使用一维 DCT 编码，通过下列三个 p 矩阵计算 M 矩阵。（虽然给出的是一行，但是要理解为是一列矩阵）。给出结果。

$p_1 = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$　$p_2 = [1\ 3\ 5\ 7\ 9\ 11\ 13\ 15]$　$p_3 = [1\ 6\ 11\ 16\ 21\ 26\ 31\ 36]$

P8-19 假设一个图像使用 JPEG，其调色板大小为 8（GIF 也用这种策略，只不过调色板大小为 256）。下列颜色是由两种表示为强度等级的颜色组成。

红色：0和7　　蓝色：0和5　　绿色：0和4

说明这种情况下的调色板并且回答以下问题：

a. 发送每个像素需要发送多少位？

b. 发送下列颜色的像素需要发送多少位？红色、蓝色、绿色、黑色、白色和洋红色（红和蓝组成，不含绿色）。

P8-20 在第一种方法流式存储音频/视频（图 8-24）中，假设我们需要收听 4MB（标准情况）的压缩歌曲。如果您通过一个 56kbps 的调制解调器连接因特网，在歌曲播放之前我们需要等待多长时间（下载时间）？

P8-21 在交错中的 FEC 方法中，假设每个分组包含从音乐块中取样的 10 个样本，而不是第一个分组加载前十个样本，第二个分组加载接下来的十个样本，依此类推。而是发送将在第一个分组中加载前二十个样本中的奇数样本，第二个分组中加载前二个样本中的偶数分组，依此类推。接收端将这些样本重排序并播放分组。假设在传输中第二个分组丢失，那么在接收端丢失的将是什么？

P8-22 假设我们需要通过基于汉明距离的 FEC 发送位的数据字。说明下列数据字编码字在传输中如何自动纠正一位差错。

00 → 00000 01→ 01011 10 → 10101 11 → 11110

P8-23 假设我们需要创建可以自动纠正一位差错的码字。数据字中的位数（k）给出的冗余位数（r）应该是多少？需要记住的是编码字需要 $n = k + r$ 位，或为 $C(n, k)$。在找出它们之间的关系之后，计算当 $k = 1$、2、5、50 或 1000 时，r 中的位数是多少。

P8-24 在前面的问题中我们通过向数据字中增加位数来纠正单一的一位差错。如果我们需要纠正多位差错，则就需要增加冗余位数。为了自动纠正大小为 k 的数据字中的一位或两位差错（不一定发生），需要的冗余位数（r）是多少？在找出它们之间的关系之后，计算当 $k = 1$、2、5、50 或 1000 时，r 中的位数是多少？

P8-25 结合前两个问题，我们可以创建一个通用公式，这个公式可以用来为大小为 n 的编码字纠正任意位的差错（m）。试写出这样的一个公式。

P8-26 到 8-34 中，假设有 100 个分组。我们分别创建高分辨率和低分辨率两个分组集。每个高分辨率分组平均携带 700 位，而每个低分辨率分组平均携带 400 位。在 FEC 机制中需要发送多少额外的比特？占日常开销的百分率是多少？

P8-27 到 8-31 中，下列时间回放缓存中的数据量是多少？

a. 00:00:17　　b. 00:00:20　　c. 00:00:25　　d. 00:00:30

P8-28　图 8-70 说明了 10 个音频分组产生和到达时间。回答下列问题：

a. 如果我们在时刻t_6开始播放音频，哪个分组不能使用？

b. 如果我们在时刻t_9开始播放音频，哪个分组不能使用？

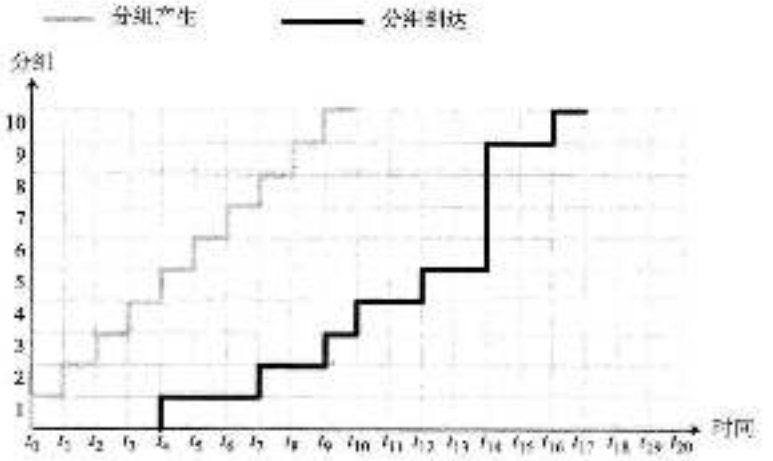


图 8-70　思考题 P8-28

P8-29　给定一个 RTP 分组，RTP 分组的前 8 个十六进制数为$(86032132)_{16}$，回答下列问题：

a. RTP协议的版本是什么？

b. 存在用于安全的填充吗？

c. 有扩展头部吗？

d. 分组中定义了多少贡献者？

e. RTP分组携带的负载类型是什么？

f. 头部总大小是多少？以字节为单位。

P8-30　在实时多媒体通信中，假设有一个发送端和十个接收端。如果发送端以 1Mbps 发送多媒体数据，一秒内每个接收端可以收到多少 RTCP 分组？假设为接收端分配 80%的 RTCP 带宽，而为发送端分配 20%的 RTCP 带宽。平均每个 RTCP 分组的大小为 1000 比特。

P8-31　解释为什么 TCP 作为一种面向字节流的协议，不适用于如实况或实时多媒体流之类的应用。

P8-32　图 8-71 显示了一个在输入端使用 FIFO 队列的路由器：

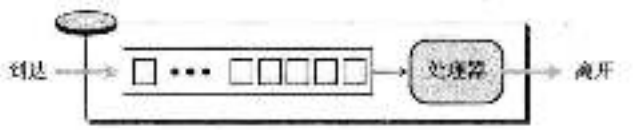


图 8-71　思考题 P8-32

七个分组的到达时间和所需服务时间如下所示。t_i是指分组到达时间或者参照时间t ms之后离开。所需服务时间的值也以ms为单位。假设传输时间可以忽略。

分组	1	2	3	4	5	6	7
达到时间	t_0	t_1	t_2	t_3	t_4	t_5	t_7
所需服务时间	1	1	3	2	2	3	1

a. 通过时间行，说出每个分组的到达时间、处理持续时间和离开时间，并说出每一毫秒开始时队列的内容。

b. 根据之间离开的分组计算出每个分组在路由器消耗的时间和离开时刻。

c. 如果所有的分组属于同一个应用程序，判断路由器对分组是否产生抖动。

P8-33　图 8-72 显示了一个在输入端使用优先级队列的路由器。

有10个分组的到达时间和所需服务时间（传输时间忽略）如下所示。t_i是指分组到达时间或者参照时间t ms之后离开。所需服务时间的值也以ms为单位。高优先级的分组有分组1、2、3、4、7和9（彩色表示），其他分组的优先级为低。



图 8-72　思考题 P8-33

分组	1	2	3	4	5	6	7	8	9	10
到达时间	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
所需服务时间	2	2	2	2	1	1	2	1	2	1

a. 通过时间行，说出每个分组的到达时间、处理持续时间和离开时间，并说出每一毫秒优先队列（Q1）的内容和非优先队列（Q2）的内容。

b. 根据之前离开的分组计算优先队列中每个分组在路由器中消耗的时间和离开时延。判断路由器对这类分组是否产生抖动。

c. 根据之前离开的分组计算非优先队列的每个分组在路由器中消耗的时间和离开时延。判断路由器对这类分组是否产生抖动。

P8-34　为了控制输出流，路由器在输出端设置了一个有三个队列的加权队列机制。分组在传输前被分类并保存在这三个队列中。队列的权值分配为：w = 3，w = 2，w = 1（3/6、2/6、1/6）。T_0时刻每个队列的内容如图 8-73 所示。假设所有分组大小相同并且每个分组的传输时间为 1μs。

a. 通过时间行，给出每个分组的离开时间。

b. 给出5、10、15和120μs后队列的内容。

c. 在w = 3的队列中，根据之前的分组计算每个分组的离开时延。这个队列会产生抖动吗？

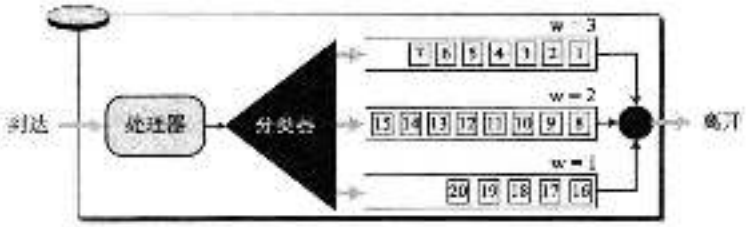


图 8-73　思考题 P8-34

d. 在w = 2的队列中，根据之前的分组计算每个分组的离开时延。这个队列会产生抖动吗？

e. 在w = 1的队列中，根据之前的分组计算每个分组的离开时延。这个队列会产生抖动吗？

P8-35　图 8-61 中，假设每类的权值为 4、2 和 1。在最高权值队列中的分组标记为 A，中等权值队列中的标记为 B，低权值队列中的标记为 C。给出下列各种情况时传输的分组列表。

a. 每个队列都有大量的分组。

b. 从权值高到权值低，三个队列的分组数依次是：10，4和10。

c. 从权值高到权值低，三个队列的分组数依次是：0，5和10。

P8-36　漏桶用来控制液体流量。在一分钟内，桶在前 12 秒的流入速率为 100 gal/min，后 48 秒为 0。如果桶中流出速率是 5 gal/min，问总共有多少液体流出？

P8-37　假设每秒有三个固定大小的分组到达路由器。说明路由器如何使用“漏桶”算法使其每秒只发送两个分组。使用这种方法有什么问题？

P8-38　假设路由器每 100ms 接收 400 比特分组，也就是该数据传输率为 4kbps。说明如何使用“漏桶”算法使其输出数据传输率小于 1kbps。

P8-39　在使用“令牌桶”算法的交换机中，桶中加入令牌的速率为 $r = 5$ 令牌/秒。令牌桶的容量为 $c = 10$。交换机的缓存只可以保存 8 个分组（举例假设）。分组到达交换机的速率为 R 分组/秒。假设分组大小相同且所需处理时间相同。如果 0 时刻桶为空，说明下列各种情况时桶和队列的内容，并解释原因。

a. $R=5$　　b. $R=3$　　c. $R=7$

P8-40　为了理解令牌桶算法如何授予发送端信许，此发送端暂时不使用它的速率分配但之后将使用，让我们重复之前的问题，其中 $r=3$、$c=10$。假设发送端的速率是可变的，如图 8-74 所示。发送端在前两秒内每秒发送三个分组，接下来的两秒内不发送分组，而最后三秒内每秒发送七个分组。发送端可以每秒发送五个分组，但是因为发送端在前四秒不能充分使用它的带宽，它只能在接下来的三秒内发送更多的分组。说明每秒令牌桶和缓存中的内容以证明这个情况。



图 8-74　思考题 P8-40

P8-41　交换机的接口被设计为使用速率为 8000 字节/秒的“漏桶”算法。如果队列中收到下列帧，试判每秒发送的是哪个帧。

- 帧 1、2、3、4：每个 4000 字节
- 帧 5、6、7：每个 3200 字节
- 帧 8、9：每个 400 字节
- 帧 10、11、12：每个 2000 字节

P8-42　假设一个 ISP 使用三个“漏桶”来控制接收三个客户向因特网上传的数据。客户发送固定大小的分组（单元）。ISP 每秒为每个客户发送 10 个分组。而其最大速率为每秒 20 个分组。每个“漏桶”由 FIFO 队列和计时器实现，每 1/10 秒从队列里发送一个分组（见图 8-75）。

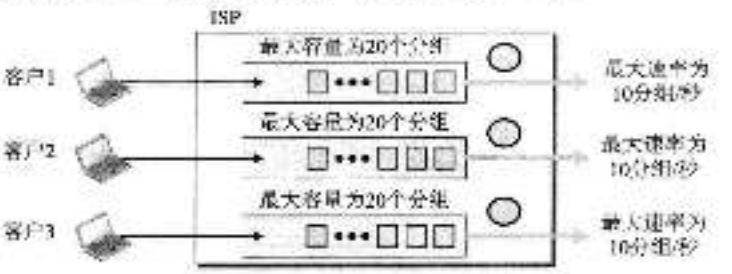


图 8-75　思考题 P8-42

a. 第一个客户前7秒每秒发送5个分组，而后9秒每秒发送15个分组，请给出这个客户的速率和队列内容。

b. 第二个客户前4秒每秒发送15个分组，而后14秒每秒发送5个分组，请给出这个客户的速率和队列内容。

c. 第三个客户前2秒不发送分组，接下来2秒每秒发送20个分组，然后重复四次，请给出这个客户的速率和队列内容。

P8-43　假设 ISP 在上一个问题中决定使用“令牌桶”（容量 $c=20$，速率 $r=10$）来代替“漏桶”，又允许客户一段时间内不发送分组，而稍后可能爆发式的发送。每个令牌桶由被相应客户使用的一个很大的队列实现（没有分组丢失），桶保存令牌和计时器，其中计时器用来控制桶中漏下的令牌（见图 8-76）。

a. 第一个客户前7秒每秒发送5个分组，后9秒每秒发送15个分组，请给出这个客户的速率和队列内容。

b. 第二个客户前4秒每秒发送15个分组，后14秒每秒发送5个分组，请给出这个客户的速率和队列内容。

c. 第三个客户前2秒不发送分组，接下来的2秒每秒发送20个分组，然后重复4次，请给出这个客户的速率和队列内容。

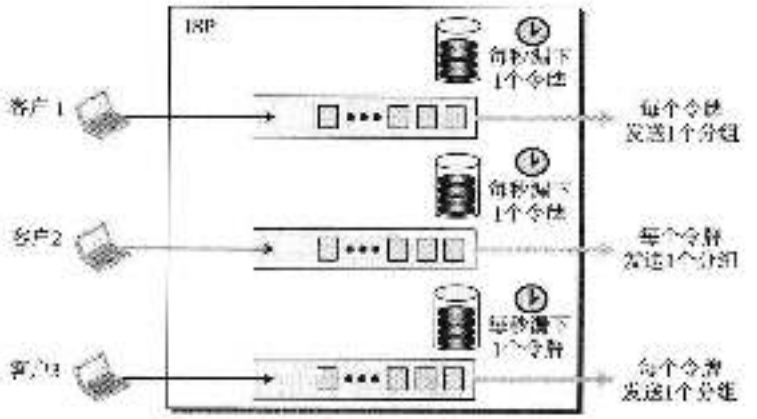


图 8-76　思考题 8-43

8.8　模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在这一部分，我们使用 Wireshark 仿真两种协议：安全外壳（Secure shell，SSH）和安全超文本传输协议（HyperText Transfer Protocol Secure，HTTPS）。这些实验作业的完整细节参见本书网站。

Lab8-1　在第一个实验中，我们需要检验多媒体分组的内容以及用发送端使用的编码。

Lab8-2　在第二个实验中，我们需要检验多媒体分组的内容以及通信中所使用的传输层协议。

8.9　编程作业

利用你选择的编程语言，编写源代码，编译并测试如下程序：

Prg8-1　编写程序，将第 2 章中的一段 UDP 客户端-服务器程序（C 语言）或第 11 章中的程序（Java）修改为包含 RTP 的。C 语言的程序需要使用 RTP 库接收和发送 JPEG 图像。而 Java 程序需要使用 RTP Java 类。

Prg8-2　编写程序，模拟“漏桶”。

Prg8-3　编写程序，模拟“令牌桶”。

Prg8-4　编写程序，编码和解码第一版的行程长度压缩算法。

Prg8-5　编写程序，编码和解码第二版的行程长度压缩算法。

Prg8-6　编写程序，模拟表 8-1（LZW 编码）。

Prg8-7　编写程序，模拟表 8-2（LZW 解码）。

Prg8-8　编写程序，模拟表 8-4（算术编码）。

Prg8-9　编写程序，模拟表 8-5（算术解码）。

Prg8-10　编写程序，读取一个 $N \times N$ 的二维矩阵，然后以本章描述的“之”字形序列输出。

Prg8-11　编写计算一维矩阵 DCT 变换的程序，实现矩阵相乘。

Prg8-12　编写计算二维矩阵 DCT 变换的程序，实现矩阵相乘。

第9章

Computer Networks: A Top-Down Approach

网络管理

尽管网络管理已经在 TCP/IP 协议簇的应用层实现，但是直到现在才讨论这个问题是为了更详细地讨论。随着因特网规模变得越来越庞大，网络管理发挥着一个重要的作用。一台设备的故障可能中断这个设备与因特网中其他设备的通信。在这章中，我们首先讨论网络管理的分类，然后讨论每一类在 TCP/IP 簇中的应用层是如何实现的。

- 9.1 节介绍了网络管理的概念，并且讨论了网络管理的 5 大类：配置管理、故障管理、性能管理、安全管理和计费管理。配置管理（configuration management）与每个实体的状态及其与其他实体的关系有关。故障管理（fault management）是处理系统中与中断有关的问题的网络管理。性能管理（performance management）试图监视和控制网络，确保网络尽可能有效地运行。安全管理（security management）根据预定的负责策略监视对网络的访问。计费管理（accounting management）是通过费用控制用户访问网络的资源。
- 9.2 节讨论了**简单网络协议**（Simple Network Management Protocol，SNMP），它是使用 TCP/IP 协议簇对互联网上的设备进行管理的一个框架，并且说明了管理器（通常是一台主机）如何运行 SNMP 客户端和代理（通常是路由器或主机），以及如何运行一个服务程序。我们定义了因特网中管理协议的三个组成部分。也定义了**管理信息结构**（Structure of Management Information，SMI），指定 SNMP 中数据类型和对象如何被识别的语言。然后介绍了**管理信息库**（Management Information Base，MIB），它根据 SMI 定义的规则来指定 SNMP 中被管理的对象。
- 9.3 节针对一个标准进行了一个简短的讨论，这个标准提供了定义数据和对象的方法和规则。这一节非常简短，只介绍了这个主题。它的一部分在 9.2 节的 SMI 中使用。

9.1 介绍

我们将网络管理（network management）定义为监视、检测、配置和故障发现及修理的网络组件，以满足组织机构的一系列需求。这些要求包括网络能够顺利地、高效地运行，为用户提供预定的服务质量。为了完成这个任务，网络管理系统使用软件、硬件以及人工参与来实现。

国际标准化组织（International Organization for Standardization，ISO）将网络管理的功能划分为 5 大类：配置管理、故障管理、性能管理、安全管理和计费管理。如图 9-1 所示。



图 9-1 网络管理的功能

尽管有些组织认为包括其他分类，比如成本管理，但是我们认为 ISO 分类法是专门针对网络管理的。例如，成本管理是一个通用管理方法，面向任何管理系统而不仅仅对网络管理。

9.1.1 配置管理

大型网络通常是由上千个物理地或逻辑地连接在一起的实体组成。当网络建立时这些实体有一个初始配置，但是能随着时间改变。台式计算机可能由其他计算机取代，应用软件可能更新为新的版本，用户也可能由一组移动到另一组。任何时候，配置管理（configuration management）可划分为两个子系统：重新配置和文档资料。

重新配置

重新配置在大型网络中每天都可能发生。重新配置有 3 种类型：硬件重新配置（hardware reconfiguration）、软件重新配置（software reconfiguration）和用户账号重新配置（user-account reconfiguration）。

硬件重新配置

硬件重新配置包括硬件的所有变化。例如，台式计算机可能被其他计算机取代，一个路由器可能需要从网络的一个部分移动到另一个部分，某一个子网可能增加到网络上或从网络上删除。所有这些都需要网络管理及时关注。在大型网络中，必须有专职技术人员快速和高效地进行硬件重新配置。令人遗憾的是，这种类型的重新配置不是自动的，而需按情况手动处理。

软件重新配置

软件重新配置包括软件的所有变化。比如，新软件可能需要安装在服务器或客户机上，操作系统需要更新。幸好，大多数的软件重新配置都是自动的。例如，更新某些或所有客户机上的一个应用程序可以从服务器上下载电子版本。

用户账号重新配置

用户账号重新配置不是简单的在系统上增加或删除用户，还必须考虑用户作为个人和作为一个组的成员的用户权限。例如，某个用户可以对有些文件有读和写的权限，但对另一些文件仅有读的权限。在某种程度上，用户账号重新配置是自动的。例如，在学院或大学中，每一季度或每一学期将新生增加到系统中。根据新生所选择的课程或从事的专业，通常将他们分组。每个组的成员有具体的权限，计算机科学专业的学生可能需要接入的服务器提供不同的计算机语言工具，而工程专业的学生则可能需要接入的服务器提供计算机辅助设计软件（比如 CAD）。

文档资料

原始网络配置和每次变化都必须有详细的记录。也就是说要有硬件文档、软件文档和用户账号文档。

硬件文档

硬件文档（hardware documentation）通常包含两组文档：映射和说明。

映射 映射（map）追踪硬件的每一部件及其与网络的连接。也可有一个总的映射说明每个子网之间的逻辑关系。还可有第二个总映射，它说明每个子网的物理位置。然后，对于每个子网有一个或多个说明所有设备的映射，这些映射使用的标准能易于当前的或以后的用户阅读和理解。

说明 仅有映射还是不够的，硬件的每一部件也需要有文档。连接网络的每一部件必须有一组说明（specification），这些说明包含硬件类型、序列号、制造厂商（地址和电话号码）、购买日期和保修卡等信息。

软件文档

所有软件也需要文档。软件文档（software documentation）包含的信息有软件型号、版本、安装日期和许可证。

用户账号文档

大多数操作系统有一个允许进入用户账号文档（user account documentation）的程序。管理人员必须确保对这些文件进行更新与维护。有些操作系统在两个文档中记录访问权限：一个表示所有

文件和每一个用户的访问类型；另一个表示访问一个具体文件的用户列表。

9.1.2 故障管理

当前复杂的网络是由数百个有时是数千个组件组成的，网络的正确运行取决于各个组件的正确运行以及相互之间的正确运行。故障管理（fault management）是处理这类问题的网络管理。一个有效的故障管理系统有两个子系统：再生的故障管理和主动故障管理。

再生的故障管理

再生的故障管理系统（reactive fault management system）负责故障的检测、隔离、纠正和记录。它处理对故障的短期解决办法。

检测故障

再生的故障管理系统所采取的第一步是检测故障准确的位置。故障被定义为是系统中的异常状态。当一种故障发生的时候，系统或完全停止工作或产生过多的错误。当一种故障发生的时候，系统或适当停止工作或产生过多的错误。一个很好的关于故障的例子是通信介质被损坏，故障可能中断通信或产生过多的差错。

隔离故障

再生的故障管理系统所采取的第二步是隔离故障。如果故障被隔离，则通常仅有少数用户受影响。隔离后，立即通知受影响的用户并给出一个纠正错误的大致时间。

纠正故障

第三步是纠正故障。这可能包括替换和修理故障的部分。

记录故障

故障纠正后，必须被记录在文档中。这个记录必须表明故障的准确位置，可能引起的原因或修复故障的措施、成本和所耗花费的时间。文档极其重要，有下列几点理由：

- 故障可能重复发生。文档有助于帮助网络管理人员解决同样的或者类似的问题。
- 同一类故障的频率是系统中较重要问题的指示。如果某一故障在一个组件中经常发生，就应该用另一个组件去更换它，或者改变整个系统不要使用那种组件。
- 统计是对网络管理和性能管理有帮助的另一个部分。

主动的故障管理

主动的故障管理（proactive fault management）试图防止故障发生。尽管通常这是不可能的，但有些故障还是可预知的和可避免的。例如，制造商指定一个部件或一个部件中某部分的有效期，在到达有效期之前，更换它是一种好的策略。再例如，如果在网络某一特定点经常有故障发生，那么在重新配置网络时注意防止故障再一次发生也是明智的。

9.1.3 性能管理

与故障管理有着密切联系的是性能管理（performance management），它试图监视和控制网络，确保网络尽可能有效地运行。性能管理试图通过一些可测量的量（如能力、通信量、吞吐量和响应时间等）来量化性能。在本章中讨论一些使用在性能管理中的协议（如 SNMP）。

能力

性能管理系统必须监视的一个要素是网络的能力（capacity）。每个网络的能力都是有限的，性能管理系统必须确保使用不能超过这个能力。例如，如果一个局域网设计为 100 个站点的平均数据速率是 2Mbps，则连接到这个网络 200 个站点就不能正常地运行，数据速率将会下降甚至还可能发生拥塞。

通信量

通信量（traffic）的测量方法有两种：内部的和外部的。内部通信量是用网络内部传送分组（或

字节）的个数来度量；外部通信量是用网络与外部交换分组（或字节）的个数来度量。系统在负载重的峰值期间，如果出现过多的通信量，拥塞就会发生。

吞吐量

我们可以测量单独一台设备（如路由器）或部分网络的吞吐量（throughput）。性能管理监视吞吐量确保它不降低于不可接受的水平。

响应时间

通常将从用户请求服务到服务被准许的时间称为响应时间。其他一些因素如能力和通信量会影响响应时间。性能管理监视平均响应时间和峰值时刻响应时间。响应时间的增加是一个非常严重的情况，它表示网络正朝着超过它的能力方向工作。

9.1.4 安全管理

安全管理（security management）是根据预定的负责策略监视对网络的访问。在第10章将讨论安全工具，诸如加密和认证。加密允许保护用户的隐私；认证迫使用户对自己认证。

9.1.5 计费管理

计费管理（accounting management）是通过费用控制用户访问网络的资源。根据计费管理，个人用户、部门、公司，甚至工程等都需要对从网络上得到的服务付费。付费不必现金支付，为预算目的它可以记入部门或公司的账单上。今天，组织机构使用计费系统有下列原因：

- 它防止用户独占有限的网络资源；
- 它防止用户无效地使用系统；
- 网络管理员可以按需对网络使用做出短期或长期的计划。

9.2 SNMP

最近的数十年间，多个网络管理标准被设计出来。最重要的一个是简单网络管理协议（Simple Network Management Protocol，SNMP），它使用在因特网中。我们在这节中讨论这个标准。简单网络管理协议是使用TCP/IP协议簇对互联网上的设备进行管理的一个框架，它提供一组基本的操作来监控和维护因特网。SNMP使用管理器和代理器的概念。也就是说，管理器通常是一台主机，控制和监视一组代理。代理通常是路由器（如图9-2所示）。

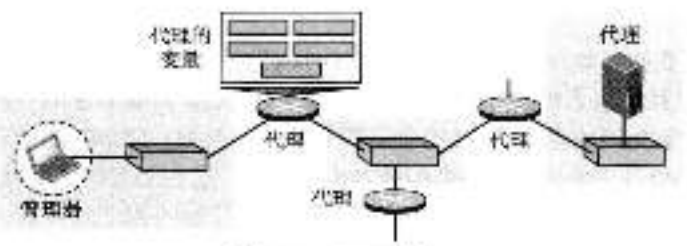


图9-2 SNMP概念

SNMP是应用层协议，它用少数几个管理器控制一组代理。该协议设计在应用层，因此它能监控不同厂商制造的设备，而这些设备可安装在不同的物理网络上。换言之，SNMP使管理任务与被管理设备的物理特性和联网技术没有关系。它可以使用由不同厂商制造的路由器相互连接在一起的不同局域网和广域网组成的异构互联网中。

9.2.1 管理器和代理

称为管理器（manager）的管理站是运行SNMP客户程序的主机。称为代理（agent）的被管

理站是运行 SNMP 服务器程序的路由器（或主机）。管理是通过管理站与代理之间的简单交互来实现的。

代理在数据库中保存性能信息，管理器使用该数据库中的值。例如，路由器可将接收到的和转发的分组数存储成适当的变量。管理器可读取和比较这两个变量，以便确定路由器是否拥塞。

管理器也可使路由器完成某些动作。例如，路由器定期检查一个重新启动计数器的值，看它何时应当重新启动自己。例如，当计数器的值为 0 时就应该引导自己。管理器可使用这个特性在任何时候从远程重新引导这个代理，它只要发送一个分组，就迫使这个计数器的值为 0。

代理也可参加到管理过程中。在代理上运行的服务器程序可检查环境，如果发现有异常现象可向管理器发送一个称为陷阱（trap）的报警报文。

换言之，SNMP 管理是基于三个基本思想：

1. 管理器检查代理的方法是发出请求得到能反映代理行为的信息；
2. 管理器可用重新设置在代理数据库中的某些值来强迫代理完成一个任务；
3. 代理管理过程的方法是向管理器发送异常情况的报警。

9.2.2 管理组件

为了完成任务，SNMP 还使用另外两个协议：管理信息结构（Structure of Management Information，SMI）和管理信息库（Management Information Base，MIB）。换言之，在因特网上的管理是通过 SNMP、SMI、MIB 三个协议共同协作完成的，如图 9-3 所示。

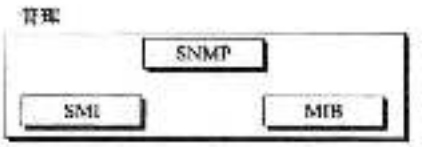


图 9-3 因特网上网络管理组件

让我们详细阐述三个协议之间的交互。

SNMP 的作用

SNMP 在网络管理中起着一些很特殊的作用，它定义从管理器到代理和从代理到管理器发送分组的格式。还说明了产生的结果并创建统计表（经常借助其他的管理软件）。交换的分组包括对象（变量）名及它们的状态（值）。SNMP 负责读取和改变这些值。

> SNMP 定义管理器与代理之间交换分组的格式，读取和改变 SNMP 分组中对象（变量）的对象值。

SMI 的作用

要使用 SNMP，我们需要有一些命名对象的规则。因为在 SNMP 中的对象是有层次结构的（对象可能有一个父对象和多个子对象），所以命名规则特别重要。对象名字的一部分可继承父对象的名字。还需要定义对象类型的规则。SNMP 能处理什么样的类型？SNMP 能处理简单类型还是结构类型？有多少简单类型可供使用？这些类型的大小是多少？这些类型的定义域是什么？此外，这些类型如何编码？

> SMI 定义对象命名、数据类型（包括长度和定义域）以及编码方法和取值的一般规则，并且说明了如何对对象和值进行编码。

因为我们不知道发送、接收或存储这些值的计算机的结构，我们必须要有通用的规则。例如，发送计算机是一个大型计算机，能将一个整数存储为 8 字节数据，而接收计算机是一个小型的计算机，仅能将一个整数存储为 4 字节数据。

SMI 是定义这些规则的协议。但我们必须明白 SMI 仅是定义规则，它不能定义一个实体管理多少个对象或对象使用什么类型。SMI 是给对象命名和定义对象数据类型的规则集合。对象与数

强类型的关联不由 SMI 来做出。

MIB 的作用

我们期望清晰地了解另一个所需要的协议。对于每个被管理的实体，这个协议必须按照 SMI 规定的规则定义对象的个数和给对象命名，以及每个对象与类型的关联。这就是 MIB 协议。与数据库（在数据库中，大多数元数据未命名并且没有数据）类似，MIB 对每个实体创建一组规定的对象。

MIB 创建了一个被管理的实体中所有对象命名和类型，以及它们之间的相互关系。

类比

在更为详细讨论这三个协议之前，我们先进行一个类比。这三个网络管理组件类似于为求解某个问题用计算机语言所编写的程序。图 9-4 给出了这个类比。



图 9-4 比较计算机程序设计和网络管理

语法：SMI

在编写程序前，必须预先定义程序设计语言的语法（如 C 语言或 Java 语言），语言还定义变量的结构（简单的、结构的、指针等）以及如何给变量命名。例如，一个变量名字长度需要 1 到 n 个字符，以一个字母开始后跟数字、字母字符。语言还定义所用的数据类型（整数型、浮点数型、字符串等）。在程序设计中，规则是由语言来定义的。而在网络管理中，规则是由 SMI 来定义的。

对象声明和定义：MIB

大多数程序设计语言要求在每个具体的程序中声明变量，其内容包括变量的名字及其类型。例如，如果某程序有两个变量：一个名字为 counter，数据类型是整数型；另一个名字为 grades，数据类型是字符型，那么在程序开始时这两个变量必须声明：

```
int counter;
char grades[40];
```

MIB 在网络管理中负责类似的任务，MIB 给每个对象命名并定义其类型。因为由 SMI 定义类型，所以 SNMP 知道其定义域和大小。

程序编写：SNMP

在编写程序的声明之后，程序要编写语句存储变量的值，如有需要，还要更新其值。SNMP 在网络管理中负责类似的任务，SNMP 按 SMI 定义规则对 MIB 已说明的对象值进行存储、更新和解释。

9.2.3 概要

在详细地讨论每个组件之前，我们用一个简单的方案说明每个组件涉及的内容。这就是我们在本章末将展开详述的概要的内容。管理器站点（SNMP 客户端）想要发送一个报文给代理站点（SNMP 服务器）用来查询代理接收到 UDP 用户数据报的个数。图 9-5 表示每步的概要。

MIB 负责查找保存接收到的 UDP 用户数据报个数的对象。在另一个嵌入式协议的帮助下，SMI 负责给对象名编码。SNMP 负责创建一个称为 GetRequest 的报文，并封装已编码的报文。当然，实际上比这个简单概述复杂得多，但是我们首先需要对每个协议有更详细的叙述。

9.2.4 SMI

管理信息结构第二版（SMIv2）是网络管理中的一个组件。SMI 是 SNMP 的一个引导。它强调三个属性来处理一个对象：名字、数据类型和编码方式。它的功能是：

- 给对象命名；

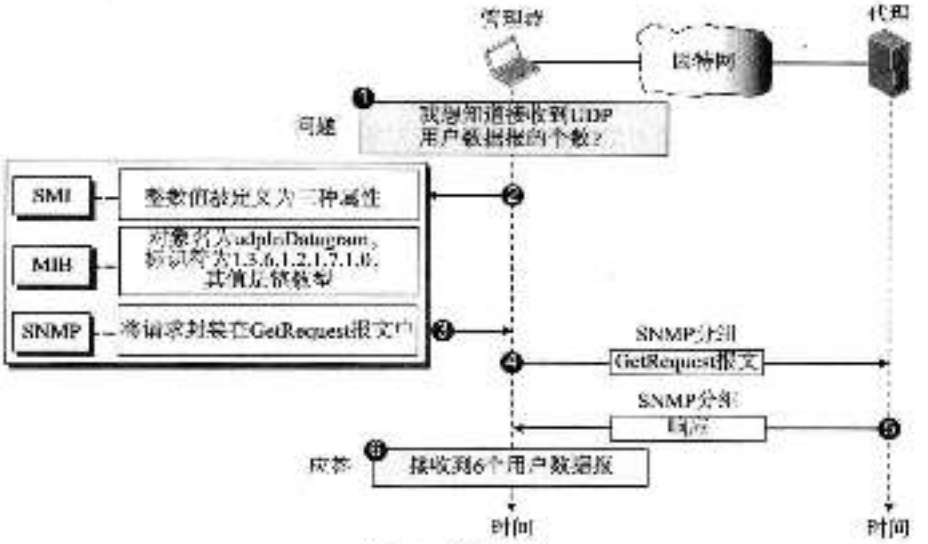


图 9-5 管理框架

- 定义可在对象中存储的数据类型；
- 给出如何对网络上传输的数据流进行编码的方法。

名字

SMI 要求每一个被管理对象（如一个路由器、一个路由器中的变量、一个值等等）具有一个唯一的名字。为了在全局给对象命名，SMI 使用对象标识符（object identifier），它是基于树结构的分层次标识符（如图 9-6 所示）。

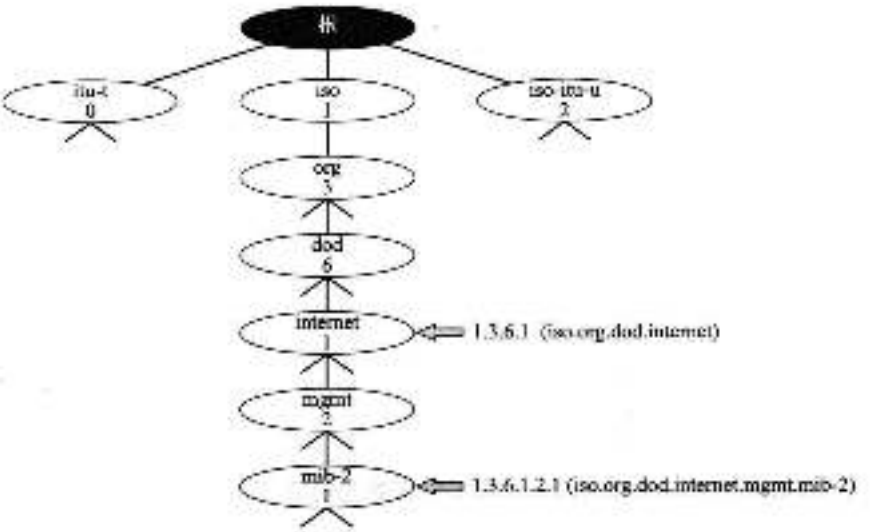


图 9-6 SMI 中的对象标识符

树结构从一个未命名的根开始，每一个对象可用一个点分隔开的整数序列来定义，树结构也可用一个点分隔开名字的文本序列来定义。整数点的表示法是 SNMP 使用的一种方法，人们使用名字点表示法。例如，下面是同一对象两种不同的表示法。

iso.org.dod.internet.mgmt.mib-2 ↔ 1.3.6.1.2.1

在 SNMP 中使用的对象位于 mib-2 对象的下面，所以它们标识符都从 1.3.6.1.2.1 开始。

类型

对象的第二个属性是在这个对象中存储的数据类型。为了定义数据类型，SMI 使用抽象语法表示法 1（Abstract Syntax Notation One，ASN.1）的一些基本定义，并且增加了几个新的定义。换言之，SMI 既是 ASN.1 的一个子集，又是 ASN.1 的一个超集（我们在本章的最后一节讨论 ASN.1）。

SMI 使用两大类数据类型：简单的和结构化的。我们先定义简单类型，然后讨论如何从简单类型构造出结构化类型。

简单类型

简单数据类型（simple data type）是原子数据类型。这些类型中的一些是直接取自 ASN.1，另一些是 SMI 增加的。表 9-1 给出了最重要的一些类型。前五个取自 ASN.1，后 7 个是 SMI 定义的。

表 9-1 数据类型

类 型	大 小	说 明
INTEGER	4 字节	在 -2^{31} 到 $2^{31}-1$ 之间的一个整数
Integer32	4 字节	与 INTEGER 类型相同
Unsigned32	4 字节	在 0 到 $2^{32}-1$ 之间的一个无符号数
OCTET STRING	可变	长度最多达 65535 的字节串
OBJECT IDENTIFIER	可变	对象标识符
IPAddress	4 字节	由 4 个整数组成的 IP 地址
Counter32	4 字节	一个整数，其值由 0 增加到 2^{32}，当它达到最大值时就返回 0
Counter64	8 字节	64 位计数器
Gauge32	4 字节	与 Counter32 同，但当它达到最大值时不返回，而保持最大值直到复位为止
TimeTicks	4 字节	记录时间的计数值，以 1/100s 为单位
BITS		位串
Opaque	可变	未解释的串

结构化类型

将简单的和结构化的数据类型结合起来，就可构成新的结构化数据类型。SMI 定义了两种**结构化数据类型**（structured data type）：sequence 和 sequence of。

- sequence。一个 sequence 数据类型是一些简单数据类型的组合，不必都是相同类型。它与 C 语言编程中使用的 struct 和 record 的概念相似。
- sequence of。一个 sequence of 数据类型是所有相同类型的简单数据类型的组合，或所有相同类型的 sequence 数据类型的组合。它与 C 语言编程中使用的 array 概念相似。

图 9-7 说明了数据类型。

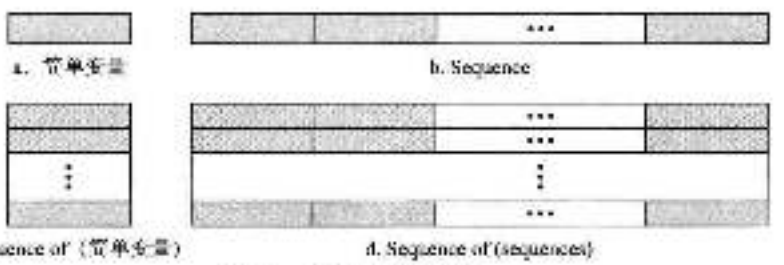


图 9-7 概念性的数据类型

编码方法

SMI 使用另一个标准，即基本编码规则（Basic Encoding Rules，BER），对数据进行编码并将编码后的数据在网络上传输。BER 规定每一块数据都要被编码成三元组的格式：标签、长度和值（TLV），如图 9-8 所示。

标签是一个字节的字段，它定义了数据类型。表 9-2 说明了我们在这章中使用的数据类型，其标签是十六进制数。字段长度是 1 个或多个字节。如果是 1 个字节，最高有效位必须是 0，其他 7 位定义数据的长度。如果是多个字节，最高有效位必须是 1，第一个字节的另外 7 位指定需要定义长度的字节数目。值字段按照在 BER 中定义的规则将数据的值进行编码。



图 9-8 编码格式

表 9-2 数据类型的代码

数据类型	标签（十六进制）	数据类型	标签（十六进制）
INTEGER	02	IPAddress	40
OCTET STRING	04	Counter	41
OOBJECT IDENTIFIER	06	Gauge	42
NULL	05	Time Ticks	43
SEQUENCE,SEQUENCE OF	30	Opaque	44

例 9.1 图 9-9 表示了如何定义整数（INTEGER）14。长度字段的大小来自表 9-1。

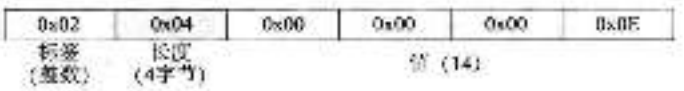


图 9-9 例 9.1：整数 14

例 9.2 图 9-10 表示了如何定义串类型（OCTET STRING）"HI"。

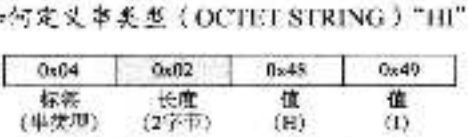


图 9-10 例 9.2：串类型 "HI"

例 9.3 图 9-11 表示了如何定义对象标识符 1.3.6.1（iso.org.dod.internet）。

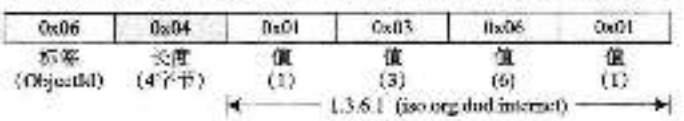


图 9-11 例 9.3：对象标识符 1.3.6.1

例 9.4 图 9-12 表示了如何定义 IP 地址 131.21.14.8。

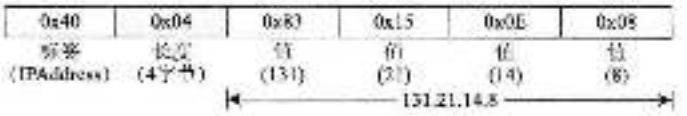


图 9-12 例 9.4：IP 地址 131.21.14.8

9.2.5 MIB

管理信息数据库版本 2（MIB2）是网络管理中的第二个组件。每个代理都有它自己的 MIB2，这是管理器能够管理的所有对象的集合（如图 9-13 所示）。

在 MIB2 中的对象分组如下：system、interface、address translation、ip、icmp、tcp、udp、egp、transmission 和 snmp（注意第九组已经废弃）。这些组都在对象标识符树中的 mib-2 对象的下面。每一个组定义一些变量和（或）表。

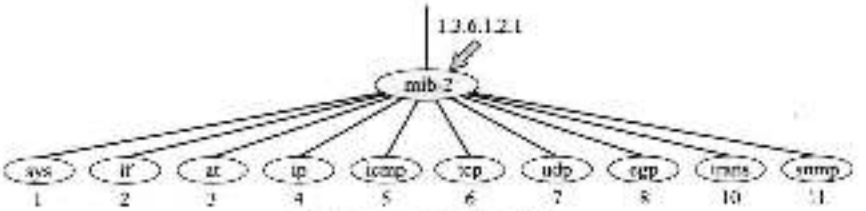


图 9-13 mib-2 的一些组

下面是一些对象的简要说明：

- sys　这个对象（system）定义了关于整个结点（系统）的通用信息，如名字、位置和生命周期。
- if　这个对象（interface）定义了关于整个结点所有接口的信息，如接口数、物理地址和 IP 地址。
- at　这个对象（address translation）定义了关于 ARP 表的信息。
- ip　这个对象定义了有关 IP 的信息，如路由表和 IP 地址。
- icmp　这个对象定义了有关 ICMP 的信息，如已发送和已接收的分组个数和产生的总差错数。
- tcp　这个对象定义了有关 TCP 的信息，如连接表、超时值、端口数及已发送和已接收到的分组个数和产生的总差错数。
- udp　这个对象定义了有关 UDP 的信息，如端口数及已发送和已接收到分组的个数。
- egp　与 EGP 操作有关的对象。
- trans　与传输具体方式有关的对象（将来使用）。
- snmp　这个对象定义了有关 SNMP 本身的信息。

访问 MIB 变量

我们用 udp 组作为例子，说明如何访问不同的变量。在 udp 组中有 4 个简单变量和一个记录（表）序列。图 9-14 给出这些变量和表。我们将说明如何访问每一个实体。

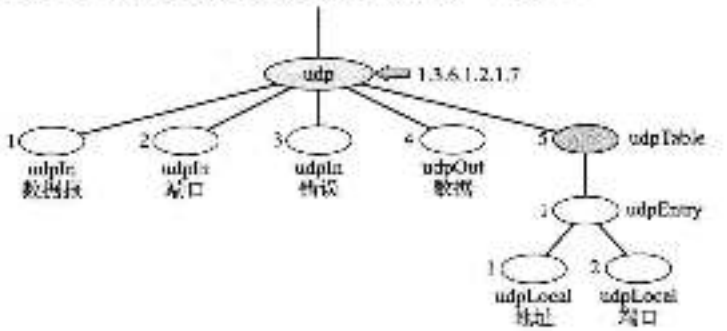


图 9-14 udp 组

简单变量

要访问任何简单变量，我们用组（1.3.6.2.1.7）的 id 后面跟着该变量的 id。下面表示如何访问每一个变量。

```
udpInDatagrams    →    1.3.6.1.2.1.7.1
udpNoPorts        →    1.3.6.1.2.1.7.2
udpInErrors       →    1.3.6.1.2.1.7.3
udpOutDatagrams   →    1.3.6.1.2.1.7.4
```

但是，这些对象标识符定义的是变量而不是实例（内容）。要给出每一个变量的实例或内容，必须增加一个实例后缀。一个简单变量的实例后缀是一个 0。换言之，要给出以上变量的实例，使用下列方法：

```
udpInDatagrams.0  →    1.3.6.1.2.1.7.1.0
udpNoPorts.0      →    1.3.6.1.2.1.7.2.0
udpInErrors.0     →    1.3.6.1.2.1.7.3.0
udpOutDatagrams.0 →    1.3.6.1.2.1.7.4.0
```

表

要标识一个表，首先使用表 id。如图 9-15 所示，udp 组只有一个表（id 为 5）。因此，要访问这个表，使用下面形式：

```
udpTable    →    1.3.6.1.2.1.7.5
```

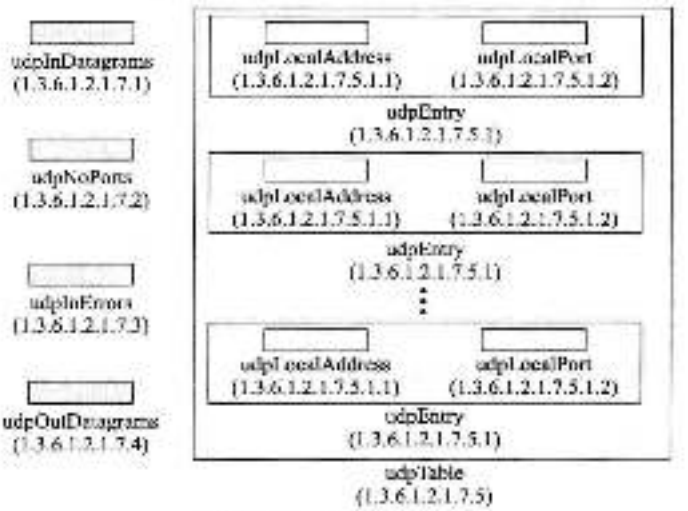


图 9-15 udp 的变量和表

但是，这个表不是树结构的叶子级，我们不能访问这个表；因此需要定义这个项目中的每一个实体（字段）。

```
UdpEntry    →    1.3.6.1.2.1.7.5.1
```

这个项目也不是一个叶子级，我们不能访问它，因此需要定义这个项目中的每一个实体（字段）。

```
UdpLocalAddress  →    1.3.6.1.2.1.7.5.1.1
udpLocalPort     →    1.3.6.1.2.1.7.5.1.2
```

这两个变量在树的叶子上。虽然能访问它们的实例，但需要定义是哪一个实例。在任何时候，这个表对每一个本地地址/本地端口对都可以有多个值。要访问表中的一个特定实例（行），应当给以上 id 加上索引。在 MIB 中，数组的索引不是整数（像大多数的编程语言那样）。这些索引是基于在这些项目中的一个或多个字段的值。在我们的例子中，udpTable 的索引是基于本地地址和本地端口号的。例如，图 9-16 显示了一个具有 4 行的表和每一个字段的值。每一行索引是这两个值的组合。要访问第一行本地地址的实例，使用标识符加上实例的索引：



图 9-16 udpTable 的索引

9.2.6 SNMP

SNMP 在因特网的网络管理中使用 SMI 和 MIB。SNMP 是一个应用程序，它允许：

- 管理器读取代理定义的一个对象的值；
- 管理器将一个值存储在代理定义的一个对象中；
- 代理将关于异常情况的告警报文发送给管理器。

PDU

SNMP 版本 3 定义了 8 种类型的协议数据单元（或 PDU）：GetRequest、GetNextRequest、GetBulkRequest、SetRequest、Response、Trap、InformRequest 和 Report（如图 9-17 所示）。

GetRequest（请求读取）

GetRequest PDU 报文是由管理器（客户机）发送给代理（服务器），用来读取一个变量或一组变量。

GetNextRequest（读取下一个请求）

PDU 报文是由管理器发送给代理，用来读取一个变量的值。所要读取的值是 PDU 中定义的 ObjectId 后面对象的值，它主要是用来读取一个表中的项目的值。如果管理器不知道该项目的索引，它就不能读这个值。但是，它可以使用 GetNextRequest 并定义表的 ObjectId。因为第一个项目的 ObjectId 是紧跟在表的 ObjectId 后面，因此第一个项目的值立即返回了。管理器可使用这个 ObjectId 来得到下一个项目的值，等等。

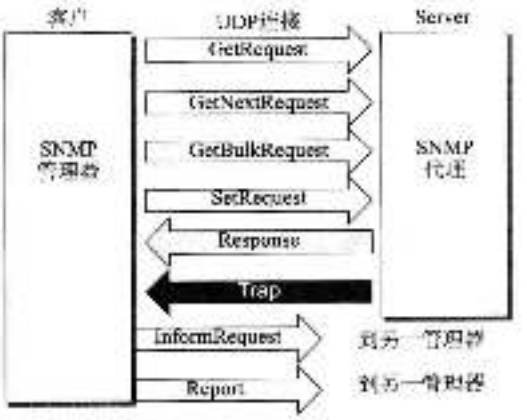


图 9-17 SNMP PDU

GetBulkRequest（读取大量请求）

PDU 报文是由管理器发送给代理，用来读取大量数据。它可以用来取代多个 GetRequest 和 GetNextRequest PDU。

SetRequest（设置请求）

PDU 报文是由管理器发送给代理，用来设置（存储）一个变量的值。

Response（响应）

PDU 报文是由代理发送给管理器以响应 GetResponse 和 GetNextRequest，它包含管理器所请求的变量的值。

Trap（陷阱）

也称为 SNMP 版本 2 的 Trap，以区别 SNMP 版本 1 的 Trap。PDU 报文是由代理发送给管理器，

用来报告一个事件。例如，如果代理重新启动，它通知管理器并报告重新启动的时间。

InformRequest（请求答复）

PDU 报文是由一个管理器发送给另一个远程管理器，用来获得远程管理器所控制的代理的某些变量的值。远程管理器通过发送一个 Response PDU 响应。

Report（报告）

Report PDU 用来报告管理器之间错误类型。它还没有被使用。

格式

8 种 SNMP PDU 的格式如图 9-18 所示。在图 9-18 中，除 GetBulkRequest 有两处不同外，其他 PDU 都相同。

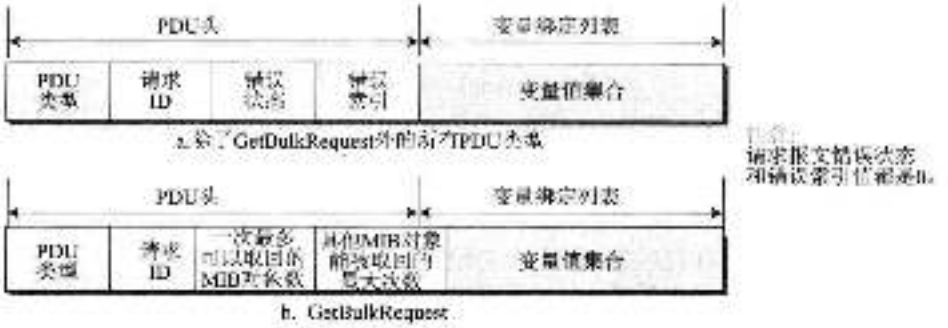


图 9-18 SNMP PDU 格式

各字段列出如下：

- PDU 类型（PDU type）。这个字段定义 PDU 类型（如表 9-3 所示）。
- 请求标识（Request ID）。这个字段是序列号，由管理器在 Request PDU 中使用，而代理在响应中重复它，它用来使响应和请求相匹配。
- 错误状态（Error status）。这是只用于 Response PDU 中的一个参数，它给出代理报告的错误类型。在 Request PDU 中它的值是 0，表 9-4 列出了可能出现的错误类型。

表 9-3 PDU 类型

类 型	标识（十六进制）	类 型	标识（十六进制）
GetRequest	A0	GetBulkRequest	A5
GetNextRequest	A1	InformRequest	A6
Response	A2	Trap(SNMPv2)	A7
SetRequest	A3	Report	A8

表 9-4 错误类型

状 态	名 字	说 明
0	noError	无错误
1	tooBig	响应太大无法放入一个报文中
2	noSuchName	变量不存在
3	badValue	要存储的值无效
4	readOnly	这个值不能修改
5	genErr	其他错误

- 一次最多可以取回的 MIB 对象数（Non-repeaters）。这个字段只存在于 GetBulkRequest PDU 中，代替错误状态字段。在请求 PDU 中它的值是空。

- 错误索引（Error index）。错误索引是一个偏移量，它告诉管理器哪一个变量引起这个错误。
- 其他 MIB 对象能被取回的最大次数（Max-repetition）。这个字段值存在于 GetBulkRequest PDU 中，该字段定义了表中迭代检索次数以读取所有重复的对象。
- 变量绑定列表（Variable-value pair list）。这是管理器希望读取或设置的一组具有相应值的变量。在请求 PDU 中它是空的。

报文

SNMP 不能只发送 PDU，而是将 PDU 嵌入到一个报文中发送。报文由报文头部和后面相应的 PDU 组成，如图 9-19 所示。报文头部的格式依赖于版本和安全规定，统在图中表示。我们在此不详细讨论，如想了解可以查看一些具体的文档。

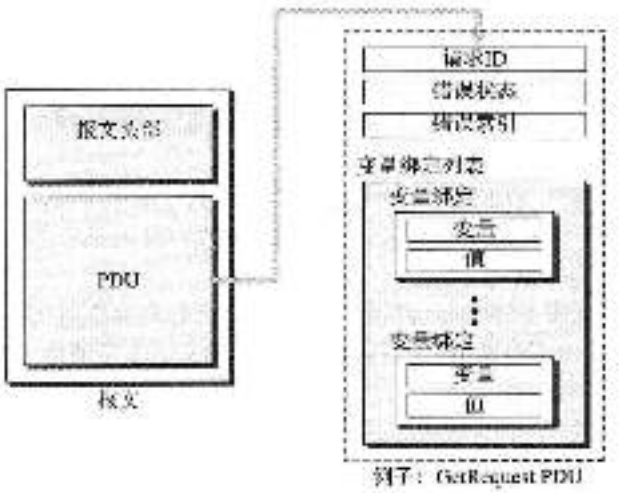


图 9-19 SNMP 报文

例 9.5 在本例中，管理器站点（SNMP 客户机）使用 GetRequest 报文读取路由器返回到的一些 UDP 差错报（见图 9-20）。

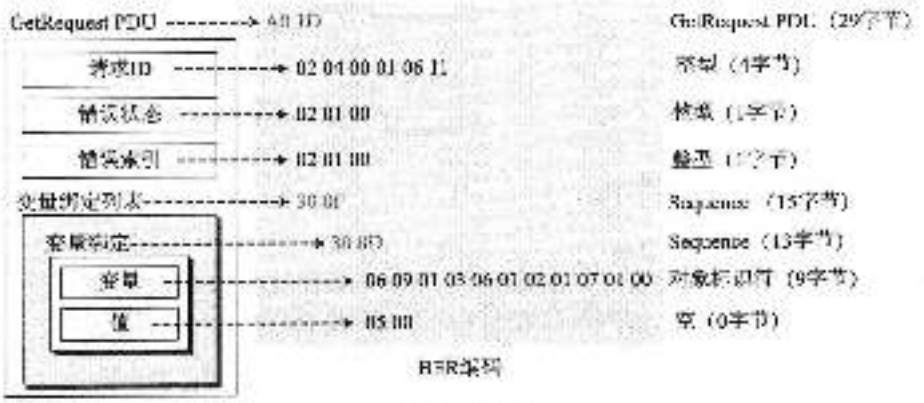


图 9-20 例 9.5

仅有一个变量绑定表项，与这个信息相应的 MIB 变量是 udpInDatagram，其对象标识符是 1.3.6.1.2.1.7.1.0。管理器打算获取一个值（而不是存储一个值），因此值定义为 0 的空实体。发送的字节用十六进制表示。

变量绑定表只有一个绑定变量。变量类型 06，长度为 09。变量类型 05，长度为 00。整个变量绑定是 sequence，长度为 0D（13）。变量绑定表也是 sequence，长度是 0F（15）。GetRequest PDU 长度为 1D（29）。

注意，我们试图用字节来表示包含简单数据类型的 sequence 或者包含 sequences 和简单数据类型的较大的 sequences。PDU 本身就像一个 sequence，但是它的标识用十六进制表示是 A0。

图 9-21 表示发送实际的报文。我们假设报文头部是 10 字节（实际上报文头部可能会不同）。我们表示报文为每行 4 个字节，字节与字节之间用横杠连接用来标记报文头部。

UDP 端口

SNMP 在两个熟知端口 161 和 162 上使用 UDP 的服务，熟知端口 161 由服务器（代理）使用，而熟知端口 162 由客户端（管理器）使用。

代理（服务器）在端口 161 发出一个被动打开。然后它就等待从管理器（客户端）来的连接。管理器（客户端）使用临时端口发出一个主动打开。客户端向服务器发送请求报文，使用临时端口作为源端口而熟知端口 161 作为目的端口。服务器向客户端发送响应报文，使熟知端口 161 作为源端口而临时端口作为目的端口。

管理器（客户端）在端口 162 发送一个被动打开，然后它就等待从代理（服务器）来连接。代理（服务器）只要有一个 Trap 报文要发送，就使用临时端口发出一个主动打开。这个连接是单向的，从服务器到客户端（如图 9-22 所示）。

在 SNMP 中的客户端/服务器机制与其他协议不同。此处的客户端和服务器使用熟知端口。此外，客户端和服务器都必须不停地运行。其原因是请求报文是由管理器（客户）发出的，但 Trap 报文是由代理（服务器）发出的。

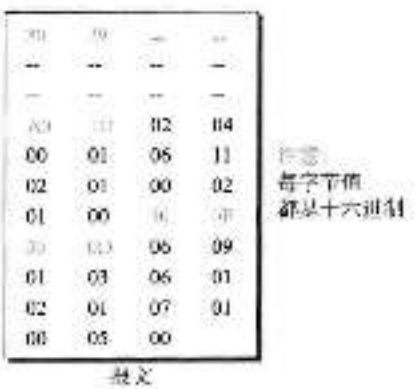


图 9-21 例 9.5 中实际报文发送

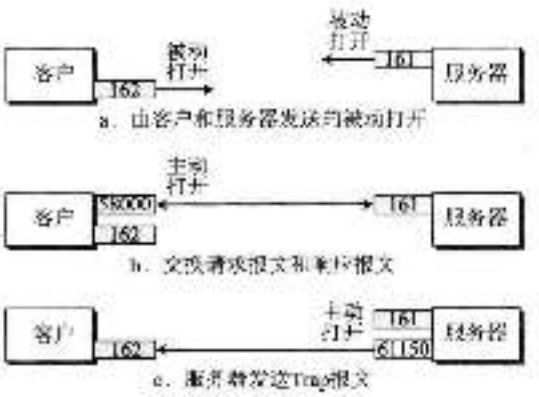


图 9-22 SNMP 端口号

安全性

SNMPv3 较以前的版本多了两个新的特性：安全性和远程管理。SNMPv3 允许管理器访问代理时选择一个或多个安全等级。管理器能修改不同方面的安全配置，允许报文鉴别，保密性和完整性。

SNMPv3 也允许管理员远程修改安全配置，也就是说不需要管理员实际位于设备所在的位置。

9.3 ASN.1

在数据通信中，当我们向目的端发送连续的比特流时，由于某种原因需要定义数据的格式。如果在一个单独的报文中发送一个名字（name）和一个数字（number），我们需要告诉目的端：例如前 12 个比特定义名字而接下来的 8 位定义数字。当我们发送如数组或记录之类的复杂数据时会更

加困难。例如，在数组例子中，如果我们发送 2000 比特的报文，我们需要告诉接收端这是一个 200 个数的数组，每个数 10 位，或者是一个 10 个数的数组，每个数 200 位。

一个问题是从网络传输的位序列中将数据类型的定义分离出来。这已经通过一种抽象语言解决了。这种语言使用了一些符号、关键字和原子数据类型，并且用这些简单类型生成一种新的数据类型。这种语言称为抽象语法表示法 1（Abstract Syntax Notation One，ASN.1）。注意，ASN.1 是一种应用在计算机科学中多个不同领域的非常复杂的语言，而在本节，我们只介绍在 SNMP 协议中需要的部分。

9.3.1 语言的基本要素

在说明如何定义对象和相关的值之前，让我们先讨论一下这种语言它本身。这种语言使用一些符号和一些关键字定义一些原始数据类型。如我们之前提到的，SMI 在我们的语言中使用这些字符实体的子集。

符号

这种语言使用如表 9-5 所示的符号集。这些符号中有一些是单个字符，而有一些是成对的字符。

表 9-5 ASN.1 中使用的符号

符 号	意 义	符 号	意 义
::=	定义为或赋值	..	范围
|	或、二选一或选项	()	组的开始和结束
,	逗号	[]	标记的开始和结束
--	接下来是注释	{}	子类型的开始和结束

关键字

这种语言有一个有限的关键字集供其使用。在这种语言中只有对已经定义的才可以使用这些关键字。这些关键字都应该大写（见表 9-6）。

表 9-6 ASN.1 中的关键字

关 键 字	描 述	关 键 字	描 述
BEGIN	一个模块的开始	OBJECT	与 IDENTIFIER 一起使用，定义对象
CHOICE	可供选择的表	OCTET	八位为一组的数据
DEFINITIONS	数据类型或对象的定义	OF	与 SEQUENCE 或 SET 一起使用
END	一个模块的结束	SEQUENCE	有序表
EXPORTS	可以被导出到别的模块的数据类型	SEQUENCE OF	相同类型数据数组
IDENTIFIER	认证为一个对象的非负数字列	SET	无序表
IMPORTS	在外部模块和嵌入中定义的数据类型	SET OF	无序表数组
INTEGER	包括负数、零和整数	STRING	数据串
NULL	空值		

9.3.2 数据类型

讨论过在这种语言中使用的符号和关键字之后，该定义它的数据类型了。这个思想类似于如 C、C++或 Java 之类的计算机语言中的定义。在 ASN.1 中，我们有几种简单数据类型，如 integer、float、boolean、char 等。我们可以合并这些旧数据类型组成一个新的简单数据类型（用一个不同的名字）或定义一个结构化数据类型，如 array 或 struct。我们首先定义 ASN.1 中的简单数据类型，然后说明如何用这些数据类型生成一个新的数据类型。

简单数据类型

ASN.1 定义了简单（原子的）数据类型集。每种数据类型被赋予一个通用的标签和一个值集，

如表 9-7 所示。这种思想相当于计算机语言中的基本数据类型，其有预定义的值区间。例如，在 C 语言中有数据类型 int。它可以表示一定范围内的值。注意，表中的标签实际上是表 9-2 中定义的标签的最右边五位。

表 9-7　一些简单的 ASN.1built-in 类型

标　签	类　型	值　集
Universal 1	BOOLEAN	TRUE 或 FALSE
Universal 2	INTEGER	整数（负数、0、正数）
Universal 3	BIT STRING	二进制数的有序集
Universal 4	OCTET STRING	OCTET 串或空集
Universal 5	NULL	Null，单值
Universal 6	OBJECT IDENTIFIER	值集，其定义一个对象
Universal 7	Object Descriptor	用人们可读的文本描述一个对象
Universal 8	EXTERNAL	不在标准中的类型
Universal 9	REAL	科学计数法中的实数
Universal 10	ENUMERATED	值枚举
Universal 16	SEQUENCE，SEQUENCE OF	有序表的类型
Universal 17	SET，SET OF	无序表的类型
Universal 18	Numeric String	数字 0～9 和空格
Universal 19	Printable String	可打印字性
Universal 26	Visible String	ISO646String
Universal 27	General String	通用字符串
Universal 30	CHARACTER STRING	字符集

新数据类型

ASN.1 使用巴科斯范式（Backus Naur Form，BNF）语法定义新数据类型，这些新数据类型来自 built-in 数据类型或预定义的数据类型，如下所示：

<new type> ::= <type>

其中 new type 必须以大写字母开始。

例 9.6　下面的例子中的新类型使用来自表 9-7 中的 built-in 类型。

```
Married::=BOOLEAN
MaritalStatus::=ENUMERATED{single,married,widowed,divorced}
DayOfWeek::=ENUMERATED{sun, mon tue, wed, thu, fri, sat}
Age::=INTEGER
```

新子类型

ASN.1 其至允许我们创建一个子类型，其区域是 built-in 数据类型或预定义的数据类型的子区域。

例 9.7　下列说明了我们如何创建三个子类型。第一个的区域是 INTEGER 的子集，第二个的区域是 REAL 的子集，第三个的区域是 DayOfWeek 的子集（我们在例 9.6 中定义过）。注意，我们用（..）定义区域并且用（|）定义选择。

```
NumberOfStudents::=INTEGER(15..40)          —An integer with the range 15 to 40
Grade::=REAL(1.0..5.0)                      —A real number with the range 1.0 to 5.0
Weekend::=DayOfWeek(sun|sat)                —A day that can be sun or sat
```

简单变量

在编程语言中，我们可以创建一个特定类型的变量并且给它赋（保存）一个值。在 ASN.1 中，术语值名字（Value Name）用来替代变量，但是我们使用术语变量（variable），因为程序员更习惯这个。我们可以创建一个特定类型的变量并且赋一个属于这个类型定义范围内的值。下面说明了其语法：

<variable> <type> ::= <value>

为了与 type 区别，variable 应该以小写字母开头。

例 9.8 下面是定义变量并且从相应类型范围中分配相应值的一些例子。注意，第一个变量和第二个变量是 built-in 类型，第二个的变量是在例 9.6 中定义的类型，最后一个变量是在例 9.7 中定义的子类型。

```
number OfCoputers INTEGER::=2
married Married ::=FALSE
herAge INTEGER::=35
clAssSize NumberOfStudents::=22
```

结构化类型

ASN.1 使用关键字 SEQUENCE 定义结构化数据类型，这与 C 语言或 C++中的 struct（record）相似。SEQUENCE 类型是一种有序表的变量类型。下面说明了一种新类型 StudentAccount，它是由三个变量组成的序列：username、password 和 accountNumber。

```
StudentAccount::=SEQUENCE
{
        username VisibleString,
        password VisibleString,
        accountNumber INTEGER
}
```

结构化变量

在定义了新类型之后，我们可以在新类型中创建变量并且赋值，如下所示：

```
johnNewton StudentAccount
{
        userName"john",
        password"120007",
        accountNumber 25579
}
```

图 9-23 从类型定义和赋值上说明创建的 record。

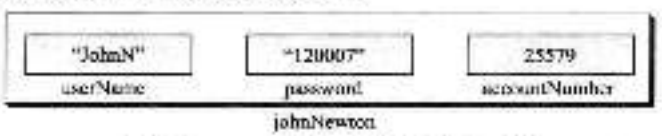


图 9-23 record 表现了类型定义和赋值

我们用关键字 SEQUENCE OF 定义新类型，这与 C 或 C++中的数组相似，数组是一种所有组件都相同的复合类型。例如，我们可以将路由器中的转发表定义为 SEQUENCE OF 行，其中每行都是由多个变量组成的序列。

9.3.3 编码

在数据定义和值与变量关联起来之后，ASN.1 使用一种编码规则进行编码然后发送。我们已经在之前的章节中讨论过基本编码规则。

9.4 章末资料

推荐读物

想要得到本章讨论主题的更多细节，我们推荐如下书籍和 RFC。在本书末列出了方括号中的参考资料。

书籍

覆盖多媒体与服务质量的一些书籍包括：[Com 06]、[Ste 94]、[Tan 03]和[MS 01]。

RFC

一些 RFC 列出了与本章讨论主题有关的最新知识，它们包括 RFC 3410、RFC 3412、RFC 3415 和 RFC 3418。关于 MIB 的更多信息可以在 RFC 2578、RFC 2579 和 RFC 2580 中找到。

小结

组成网络管理的 5 个部分是配置管理、故障管理、性能管理、计费管理和安全管理。配置管理涉及网络实体的物理与逻辑的改变。故障管理涉及每个网络部件的正确操作。性能管理涉及监视和控制网络，确保网络尽可能有效地运行。安全管理涉及控制对网络的访问。计费管理涉及通过付费控制用户对网络资源的访问。

简单网络管理（SNMP）是使用 TCP/IP 协议簇对互联网中的设备进行管理。管理器通常是主机。它控制和监视一组代理，而代理通常是路由器。SNMP 使用 SMI 和 MIB 服务。SMI 给对象命名，定义可在对象中存储的数据类型，以及对数据进行编码。MIB 是能够被 SNMP 管理的对象组的集合。MIB 使用字典排序来管理其变量。

抽象语法标记法 1（ASN.1）是定义数据语法和语义的语言。它使用一些字符、关键字、简单的和结构化数据类型。ASN.1 的部分是由 SMI 使用用来定义对象格式和网络管理中使用的值。

9.5 习题集

测试题

本章的交互式测试题请参见这本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q9-1 下列哪个不是 ISO 定义的网络管理的 5 大类之一？

a. 故障　　b. 性能　　c. 个人

Q9-2 下列哪个不是配置管理的一部分？

a. 重新配置　　b. 加密　　c. 文档资料

Q9-3 网络管理者决定将连接到特网的旧路由器换个更好的。网络管理中的哪些类与此有关？

Q9-4 网络管理者决定将一个重要软件更新成新的版本。网络管理中的哪些类与此有关？

Q9-5 区别声生的故障管理和主动故障管理。

Q9-6 如果组织生命周期过期且网络管理器不能替换它，网络管理的哪些类被忽略？

Q9-7 区别网络内部通信量和外部通信量。

Q9-8 如果一个学生在学校里多次访问一个软件，导致其他学生等待很长时间才能访问这个软件，这时网络管理中的哪个类没有发挥作用？

Q9-9 下列哪个设备不能成为 SNMP 中的管理器站点？

a. 路由器　　b. 主机　　c. 交换机

Q9-10 SNMP 管理器可以运行客户端 SNMP 程序或者服务端 SNMP 程序吗？

Q9-11 说明在 SMI 中是如何将原义命名（textual name）"iso.org.dod" 数字编码的。

Q9-12 在 SMI 中可能有一个原义命名（textual name）是 "iso.org.internet" 吗？解释原因。

Q9-13 指出在下列对象在 SMI 中的类型（simple、sequence、sequence of）。

a. 无符号整型　　b. IP 地址　　c. 对象名字

d. 一个列表的整数　　e. 定义一个对象名字、一个 IP 地址和一个整型的记录

f. 每条记录都是计数后面跟着对象名字的列表

Q9-14 下面 BER 编码中值字段的长度是什么？

01 09 48 65 6C 6C ...

Q9-15 区别 SMI 和 MIB。

Q9-16 正对象在 MIB 中的定义是什么？为什么这个对象需要被管理？

Q9-17 假设 MIB 中的一个对象标识符有三个简单变量。如果这个对象标识符是 x，那么标识符的每个变量是什么？

Q9-18 SNMP 可以引用表中一个完整的行吗？SNMP 可以读取或改变表中完整行的值吗？请解释原因。

Q9-19 SNMP 报文可以引用 MIB 树的叶子结点吗？请解释原因。

Q9-20 假设管理的对象只有三个简单变量。问这个对象的 MIB 树有多少个叶子结点？

Q9-21 假设管理的对象有一个二列的表。问这个对象的 MIB 树有多少个叶子结点。

Q9-22 区别请求读取 PDU 和设置请求 PDU。

Q9-23 SNMP 中，下列哪个 PDU 是由客户端 SNMP 发送到服务器 SNMP？

a. 请求读取　　b. 响应　　c. 陷阱

Q9-24 当 SNMP 报文携带下列 PDU 时，源端口号和目的端口号是多少？

a. 请求报文　　b. 响应　　c. 陷阱　　d. 报告

思考题

P9-1 假设对象 x 有两个简单变量：一个整型和一个 IP 地址。问每个变量的标识符是什么？

P9-2 假设对象 x 只有一个三列的表。问每列的标识符是什么？

P9-3 假设对象 x 有两个简单变量和一个两列的表。每个变量和每列的标识符是什么？假设变量是表中的元素。

P9-4 假设对象 x 有一个简单变量和分别有两列与三列的两个表。表的每列和每个变量的标识符是什么？假设变量是表中的元素。

P9-5 对象 x 有两个简单变量。如何使 SNMP 能够用于每个变量？

P9-6 对象 x 有一个两列的表，表的三行内容如下所示。如果表的索引根据第一列的值建立，说明 SNMP 如何能访问每个元素。

Object x

a	aa
b	bb
c	cc

Table

P9-7 可以被管理的对象（组）中的一个，字是对象标识符为（1.3.6.1.2.1.4）的 ip 组。其中 1.3.6.1.2.1 是 MIB-2 标识符，4 定义 ip 组。在代理中，这个对象有 20 个简单变量和三个表，其中一个表是标识符为（1.3.6.1.2.1.4.21）的路由（转发）表。这个表有 11 列，第一列名字为 ipRouteDest（意思是目的 IP 地址）。假设索引是根据第一列建立的。表在此时有四行目的 IP 地址，分别为（201.14.67.0），（123.16.0.0），（11.0.0.0）和（0.0.0.0）。说明 SNMP 是如何访问第二列（名字为 ipRouteIndex，定义了接口号。IP 应该从这个接口号被发出）的每个元素。

P9-8 试给出 INTEGER（整数）1456 的编码。

P9-9 试给出 OCTET STRING（串）"Hello World" 的编码。

P9-10 试给出使用 BER，IP 地址 112.56.23.78 的编码。

P9-11 试给出使用 BER，对象标识符 1.3.6.1.2.1.7.1（udp 组中的 udpInDatagram 变量）是的编码。

P9-12 试说明使用 BER，结构化数据是如何编码的。这个数据由一个 INTEGER 2371），一个 OCTET STRING（"Computer"），一个 IP 地址（185.32.1.5）组成，如下所示：

```
SEQUENCE
{
    INTEGER 2371
    OCTET STRING "Computer"
    IP Address 185.32.1.5
}
```

P9-13 试说明使用 BER，由一个 INTEGER（151），另一个结构（由值为 24.70.6.14 的 IP 地址组成）和一个 OCTET STRING（"UDP"）组成的数据结构是如何编码的。

P9-14 试使用 BER 对代码 02040000C738 解码。

P9-15 试使用 BER 对代码 300C020400000998060040A05030E 解码。

P9-16 试使用 BER 对代码 300D040240D6F300706030103060500 解码。

P9-17 假设管理器需要知道代理发送的用户数据报的数量（标识符为 1.3.6.1.2.1.7.4 的 udpOutDatagrams 计数器）。给出在 GetRequest 报文中被发送的变量绑定的代码，并且当此刻计数器的值是 15 时，给出代理将要在 Response 报文中发送的代码。

P9-18 试使用 ASN.1 中结构化数据类型的语法定义一个 SNMP 报文（见图 9-19）。

P9-19 试使用 ASN.1 中结构化数据类型的语法定义一个请求读取 PDU（见图 9-18）。

P9-20 试使用 ASN.1 中结构化数据类型定义一个响应 PDU（见图 9-18）。

P9-21 试使用 ASN.1 中结构化数据类型定义一个变量绑定列表（见图 9-19）。

第 10 章

Computer Networks: A Top-Down Approach

网 络 安 全

网络安全内容很广，同时还涉及数论等一些特定的数学领域。在这一章中，我们尝试对这些内容进行简单的介绍，以作为深入学习的基础。

- 10.1 节介绍网络安全。我们将讨论安全目标、攻击类型、网络安全提供的服务。同时，我们还将提及两个安全技术：密码系统和信息隐藏。
- 10.2 节介绍安全的第一个目标——机密性。我们将讨论对称密钥密码、非对称密钥密码和它们的应用。我们将让大家看到对称密钥密码可用于长消息加密，而非对称密钥密码可用于短消息加密。目前，这两种密码我们都需要。
- 10.3 节讨论安全的其他特征：消息完整性、消息认证、数字签名、实体认证和密钥管理。今天，安全的这些特征与机密性相辅相成。我们将看到除非增加一个或多个这样的特征，有时保证机密性是不可能的。
- 10.4 节将已经学习的前 3 节内容应用于 Internet。安全可以应用于应用层、传输层和网络层。我们首先讨论应用层安全：我们给出两个用于安全电子邮件的安全协议：良好隐私协议和 S/MIME。然后我们讨论能够加强任意应用层协议安全的传输层安全：SSL 和 TLS。最后，我们讨论网络层安全 IPSec。IPSec 用于对使用传输层的应用或直接使用网络层服务的应用提供安全服务。
- 10.5 节讨论防火墙。我们将展示怎样防止有害的消息进入一个系统。我们首先讨论分组过滤防火墙，这些防火墙运行于网络层和传输层。然后我们讨论代理防火墙，这些防火墙运行于应用层。

10.1 介绍

我们生活在信息时代。我们需要管理生活中各个方面的信息。也就是说，像其他的财富一样，信息也是一种有价值的财富。作为一种财富，我们应该担保信息不受到攻击。为了使信息安全，我们应该隐藏信息以防止未授权的访问（机密性），应该保护信息以防止未授权的修改（完整性），并且当需要时应该保证信息对授权实体可用（可用性）。

在最近 30 年中，计算机网络在信息利用方面发生了一场革命。现在，信息是分布式的，被授权的人可以利用计算机网络从远程发送和检索信息。尽管上面提到的 3 个需求——机密性、完整性和可用性——没有改变，但是它们现在拥有一些新的特点。信息不仅在存储的时候需要加密，而且在从一台计算机到另一台计算机传输时也应该保证它的机密性。

在这一部分中，我们首先讨论信息安全的 3 个主要目标。然后，我们看一看攻击怎样威胁这 3 个目标。而后，我们讨论与这些安全目标相关的安全服务。最后，我们介绍用于实现安全目标且能防护攻击的两种技术。

10.1.1 安全目标

我们首先讨论安全的 3 个目标：机密性、完整性和可用性。

机密性

机密性（confidentiality）可能是信息安全最公共的特征。我们需要保护机密性的信息。一个组织需要防范那些危害它的信息机密性的恶意行为。机密性不仅适用于信息的存储，而且也适用于信息的传输。当发送一条消息到远程计算机进行存储或者当我们从远程计算机检索一条信息时，我们需要在传输中将它隐藏起来。

完整性

信息需要不断地变化。在银行，当一个客户存取资金时，他账户的结余需要改变。完整性（integrity）意味着改变需要由授权的实体并通过授权的机器进行。完整性侵害不一定是由恶意行为造成的：系统的中断，如电涌，也可以对一些信息造成不希望的改变。

可用性

信息安全的第 3 个目标为可用性（availability）。一个组织创建和存储的信息应该对授权实体可用。如果不可用，那么信息就是无用的。信息需要不断地改变，这意味着信息必须可以被授权实体访问。与缺乏机密性和完整性一样，信息不可用也会对组织造成伤害。可以想象如果客户在交易时不能访问他们的账户，银行将会发生什么情况。

10.1.2　攻击

我们的 3 个安全目标——机密性、完整性和可用性——可能受到安全攻击的威胁。尽管文字上可以按照不同的方法对攻击进行分类，但是我们将它们划分为与安全目标相关的 3 组。图 10-1 给出了这种分类。

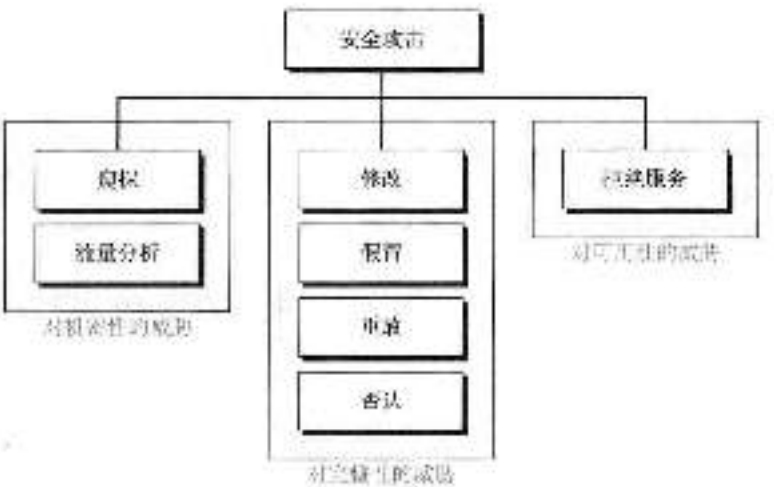


图 10-1　按照与安全目标的相关性对攻击进行分类

对机密性的威胁

通常，两种类型的攻击威胁着信息的机密性：窥探（snooping）和流量分析（traffic analysis）。

窥探

窥探指的是无授权的访问或数据窃听。例如，通过 Internet 传输的一个文件可能包含着机密信息。一个未授权的实体可能对传输进行窃听并利用这些窃听的内容为她自己的私益服务。为了防止窥探，可以利用以后讨论的加密技术将数据变成窃听者不可理解的东西。

流量分析

尽管数据加密可以将信息变成窃听者不可理解的东西，但是她可以通过监听在线流量获取一些其他的类型信息。例如，她能够发现发送者或者接收者的电子地址（如电子邮件地址），她能够收

集请求与应答信息对，以帮助她猜测交易的性质。

对完整性的攻击

信息的完整性可能受到的几种攻击包括：修改（modification）、假冒（masquerading）、重放（replaying）和拒绝（repudiation）。

修改

在窃听或访问到信息后，攻击者可以按照有利于她自己利益的方式修改信息。例如，一个客户向银行发送一条信息以开始一些交易。攻击者截获这个消息并改变交易类型以使自己获利。注意，有时候攻击者简单地删除或延迟这些信息就可以破坏系统或从中获利。

假冒

当攻击者模仿其他人时，假冒（masquerading）或欺骗（spoofing）就发生了。例如，一个攻击者可能偷窃一个银行客户的银行卡和个人标识码（PIN），然后假装她就是那个客户。有时，攻击者假冒成接收实体。例如，一个用户试图联系银行，但是另一端就称它就是银行，从而获取用户的信息。

重放

重放是另一种攻击类型。攻击者获取了一个用户发送消息的拷贝，然后尝试重放它。例如，一个人向银行发送一个要求付款的请求，已做好准备的攻击者截获这个消息，并且再次发送该消息以获得银行的另一次付款。

拒绝

这种类型的攻击与其他攻击不同，因为它是由通信双方的一方（发送方或接收方）实施的。消息的发送方以后可能否认她曾经发送过消息；消息的接收方以后也可能否认她曾经接收过消息。一个发送方否认的例子是一个银行客户要求银行给第三方发送一些资金，但是她随后否认进行过这种请求。一个接收方否认的例子是一个人从一厂家购买了产品并且使用电子方式进行了付款，但是厂家随后否认已经接收了付款并要求再次付款。

对可用性的攻击

我们仅仅提及一种对可用性的攻击：拒绝服务。

拒绝服务

拒绝服务（denial of service，DoS）是一种常见的攻击。它可以减缓或完全中断一个系统的服务。攻击者可以采用多种策略达到这种目的。她可能向一台服务器发送过多的虚假请求，使服务器因为负载过重而崩溃。攻击者可能截获并删除一台服务器向一个用户的响应，致使用户相信服务器没有进行响应。攻击者也可以截获用户的请求，致使用户发送多次请求并使系统过载。

10.1.3 服务和技术

ITU-T定义了一些安全服务以实现安全目标并防止攻击。每个服务都设计为防止一种或多种攻击，以达到维护安全目标的目的。真实地实现安全目标需要多种技术。在现在流行的两种技术中，一种非常通用（密码系统），一种比较特定（信息隐藏）。

密码系统

有些服务可以使用密码系统实现。密码系统（**cryptography**）一词源于希腊语，意思是“秘密书写”。但是，我们用这个词语来指变换消息的科学与艺术，通过变换使这些消息变得安全并免受攻击。尽管在过去，密码系统仅仅指利用密钥对消息进行加密（**encryption**）和解密（**decryption**），但是现在它包含3个清晰的机制：对称密钥密码学，非对称密钥密码学和散列法。我们将在稍后讨论这些机制。

信息隐藏

尽管本章和后面的章节以密码系统作为一种实现安全机制的技术，但是另一种过去用于秘密通信的技术现在也浮出水面：信息隐藏。**信息隐藏**（steganography）一词出于希腊语，意思是指“掩盖书写”，与密码系统的“秘密书写”相对应。密码系统的意思是指通过译成密码将消息的内容进行隐藏；信息隐藏的意思是指通过其他的东西将其覆盖，从而隐藏信息本身。我们将信息隐藏的讨论留给专门讨论这个内容的书籍。

10.2　机密性

现在，我们看看安全的第一个目标，即机密性。机密性可以利用密码实现。密码可以分为两大类：对称密钥和非对称密钥。

10.2.1　对称密钥密码

对称密钥密码（symmetric-key cipher）的加密和解密使用相同的密钥，并且密钥可以用于双向通信。这就是为什么把它叫做对称的原因。图 10-2 显示了对称密钥密码的基本思想。



图 10-2　对称密钥密码的基本思想

对称密钥密码也称为秘密密钥密码。

在图 10-2 中，实体 Alice 可以通过一个不安全的信道向另一个实体 Bob 发送一个消息，并假设对手 Eve 通过简单地窃听信道不能理解信息的内容。

从 Alice 到 Bob 的原始消息称为**明文**（**plaintext**）；通过通道发送的消息称为**密文**（**ciphertext**）。为了从明文构建一个密文，Alice 使用了一个**加密算法**（**encryption algorithm**）和一个共享的秘密密钥（*shared secret key*）。

为了从密文构建一个明文，Bob 使用了一个**解密算法**（decryption algorithm）和一个同样的秘密密钥。我们将加密和解密算法称为**密码**。一个密钥就是一个密码算法操作的数值（或数字）集。

注意，对称密钥密码的加密和解密使用一个单一的密钥（密钥本身可以是一个数值集）。另外，加密和解密算法是互逆的。假设 P 为明文，C 为密文，K 为密钥，加密算法 $E_K(x)$从明文构建密文；解密算法 $D_K(x)$从密文构建明文。我们假设 $E_K(x)$和 $D_K(x)$相逆：如果对于同样的输入，在使用一个之后使用另一个，它们的影响相互抵消。我们得到

$$\textbf{Encryption: } C = E_K(P) \qquad\qquad \textbf{Decryption: } P = D_K(C)$$

其中，$D_K(E_K(x)) = E_K(D_K(x)) = x$。我们需要强调的是最好公开加密和解密算法而秘密保存共享密钥。这意味着 Alice 和 Bob 需要另外一个安全的通道以交换秘密密钥。Alice 和 Bob 可以见一次面，亲自交换密钥。这里的安全通道就是面对面的密钥交换。他们也可以相信一个第三方，第三方给出他们相同的密钥。他们可以利用另外一种密码——非对称密钥密码——创建一个临时的密钥，这种方法将在后面讨论。

加密可以认为是把信息锁入一个盒子中；解密可以认为是对盒子解锁。在对称密钥密码中，使用同一把钥匙进行加锁和解锁操作，如图 10-3 所示。后面部分将看到非对称密钥密码需要两个密钥，一个用于加锁，一个用于解锁。



图 10-3　对称密钥密码加锁和解锁使用相同的密钥

对称密钥密码可以分为传统密码和现代密码。传统密码是简单的、面向字符型的密码。按照今天的标准，传统密码不安全。在另一方面，现代密码是复杂的、面向比特型的密码，这种密码更安全。我们简单讨论传统密码，以便为讨论更加复杂的现代密码打好基础。

传统对称密钥密码

传统密码属于过去。但是，由于它们被认为是现代密码的组成部分，因此我们在这里简单对它们进行讨论。更精确地，我们可以将传统密码分为替换密码和换位密码。

替换密码

替换密码（**substitution cipher**）将一个符号替换成另一个符号。如果明文中的符号是字母表中的字符，那么我们用一个字符替换另一个字符。例如，我们可以用字母 D 替换字母 A，字母 Z 替换字母 T。如果符号为数字（0 到 9），那么我们可以用 7 替代 3，用 6 替代 2。

替换密码使用一个符号替换另一个符号

替代密码可以分为单字母密码和多字母密码。

单字母密码　在单字母密码（monoalphabetic cipher）中，明文中的一个字符（或符号）总是变换成密文中的同一个字符（或符号），而不考虑它在文本中的位置。例如，如果算法规定明文中的字母 A 变换成字母 D，那么每个字母 A 都变换成字母 D。也就是说，明文中的字母与密文中的字母之间的关系是一对一的。

最简单的单字母密码是加性密码（additive cipher，或叫做移位密码（shift cipher））。假设明文由小写字母（a 到 z）组成，密文由大写字母（A 到 Z）组成。为了能够在明文和密文上运用数学运算，我们为每一个字母（小写的或大写的）分配一个数值，如图 10-4 所示。

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	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
值	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

图 10-4　明文和密文字符在模 26 中的表示

在图 10-4 中，每个字符（小写的或大写的）都分配了模 26 中的一个整数。Alice 和 Bob 之间的秘密密钥也是一个模 26 的整数。加密算法将明文字符与密钥相加；解密算法使用密文字符减去密钥。所有操作都在模 26 下进行。

在本性密码中，明文、密文和密钥都是模 26 的整数。

加性密码在历史上也被叫做移位密码，这是因为加密算法可以解释为“向下移动密钥个字符”，解密算法可以解释为“向上移动密钥个字符”。Julius Caesar 使用密钥 key 为 3 的加性密码与他的办公室通信。由于这个原因，加性密码有时被称为恺撒密码（Caesar Cipher）。

例 10.1　使用密钥 key=15 的加性密码对消息“hello”进行加密。

解答　我们对明文逐个字符使用加密算法：

```
明文: h → 07        加密: (07 + 15) mod 26        密文: 22 → W
明文: e → 04        加密: (04 + 15) mod 26        密文: 19 → T
明文: l → 11        加密: (11 + 15) mod 26        密文: 00 → A
```

明文: l → 11	加密: (11 + 15) mod 26	密文: 00 → A
明文: o → 14	加密: (14 + 15) mod 26	密文: 03 → D

结果为"WTAAD"。注意由于两个相同的明文字符（*l*）被加密成相同的字符（*A*），因此这个密码是一个单字母密码。

例 10.2 利用密钥 key=15 的加性密码对消息"WTAAD"进行解密。

解答 我们对密文这个字符使用解密算法：

密文: W → 22	解密: (22 − 15) mod 26	明文: 07 → h
密文: T → 19	解密: (19 − 15) mod 26	明文: 04 → e
密文: A → 00	解密: (00 − 15) mod 26	明文: 11 → l
密文: A → 00	解密: (00 − 15) mod 26	明文: 11 → l
密文: D → 03	解密: (03 − 15) mod 26	明文: 14 → o

结果为"hello"。注意运算是模 26 的，这意味着结果为负数时需要加上 26（例如-15 变为 11）。

加性密码对于使用穷尽密钥搜索的攻击（野蛮攻击）是脆弱的。加性密码的密钥域非常小，只有 26 个密钥。可是，密钥 0 是无用的（密文与明文相同），可能的密钥只留下 25 个。Eve 可以很容易地对密文进行一次野蛮攻击。一种较好的解决方案是在每个明文字符与相应的密文字符之间构建一个映射。Alice 和 Bob 可以使用商定好的字符映射表。图 10-5 显示了这样映射的一个例子。

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	N	O	A	T	R	B	E	C	F	U	X	D	Q	G	Y	L	K	H	V	I	J	M	P	Z	S	W

图 10-5 一个单字母替代密码的密钥示例

例 10.3 我们使用图 10-5 给出的密钥加密信息。

明文:	this message is easy to encrypt but hard to find the key
密文:	ICFVQRVVNERFVRNVSIYRGAHSLIOJICNHTIYHFGTICRXRS

多字母密码 在多字母密码（**polyalphabetic cipher**）中，一个字符的每次出现可能使用不同的替代。明文中一个字符与密文中的一个字符的对应关系为一对多。例如，在文本的开始，"a"可能被译为密码"D"，但是在中间可能被译为密码"N"。多字母密码具有对一种语言隐藏字母出现频率的优越性。Eva 不能使用单个字母频率分析攻破密文。

为了构建一个多字母密码，我们需要使密文中的每个字符既依赖于相应的明文字符又依赖于明文字符在消息中的位置。这意味着我们的密钥应该是一个子密钥流，其中每个子密钥依赖于使用该密钥进行加密的明文字符的位置。也就是说，我们需要一个密钥流 $k=(k_1, k_2, k_3, \cdots)$，其中 k_i 用于加密明文中的第 i 个字符，以创建密文的第 i 个字符。

为了描清密钥的位置依赖，我们讨论一种简单的、被称为自动密钥密码（auto key cipher）的多字母密码。在这种密码中，密钥是一个子密钥流，其中每个子密钥用于加密明文中相应的字符。第 1 个子密钥是由 Alice 和 Bob 一致同意的、预先定义好的一个秘密值。第 2 个子密钥是第 1 个明文字符的值（0 到 25 之间）。第 3 个子密钥是第 2 个明文字符的值，等等。

$$P = P_1P_2P_3\ldots \qquad C = C_1C_2C_3\ldots \qquad k = (k_1, P_1, P_2, \ldots)$$

$$\text{加密: } C_i = (P_i + k_i) \bmod 26 \qquad \text{解密: } P_i = (C_i - k_i) \bmod 26$$

自动密钥这个名字隐含着子密钥是在加密过程中通过明文字符自动创建的。

例 10.4 假设 Alice 和 Bob 同意使用初始密钥值 k_1=12 的自动密钥密码。现在 Alice 希望向 Bob 发送消息"Attack is today"。加密译码采用逐一字符进行。明文中的每个字符首先被替换成它的整数值。创建第 1 个密文字符时加上第 1 个密钥。读取明文字符时构建其他的密钥。由于"a"在明文中的 3 次出现被加密成不同的密文，因此这种密码是一种多字母密码。"t"的三次出现也被加密成不同的密文。

明文:	a	t	t	a	c	k	i	s	t	o	d	a	y
明文的值:	00	19	19	00	02	10	08	18	19	14	03	00	24
密钥流:	12	99	*19*	*19*	*00*	*02*	*10*	*08*	*18*	*19*	*14*	*03*	*00*

密文的值:	12	19	12	19	02	12	18	00	11	7	17	03	24
密文:	M	T	M	T	C	M	S	A	L	H	R	D	Y

换位密码

换位密码（transposition cipher）不是用一个符号替换另一个符号，而是改变符号的位置。一个位于明文第 1 个位置的符号可能出现在密文的第 10 个位置。一个位于明文第 8 个位置的符号可能出现在密文的第 1 个位置。也就是说，换位密码对符号重新排序（即倒次序）。

换位密码是排符号的顺序。

假设 Alice 希望向 Bob 秘密地发送消息 "Enemy attacks tonight"。加密和解密显示在图 10-6 中。注意我们向消息的末尾添加了一个字符（z），以使字符的数目为 5 的倍数。

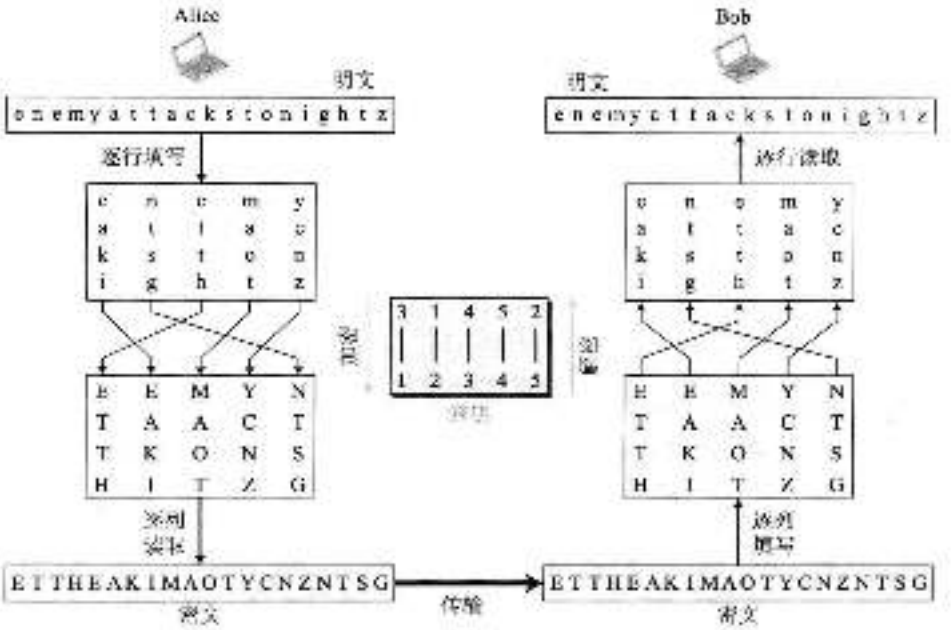


图 10-6　换位密码

Alice 逐行填写明文创建了第 1 个表。利用密钥对列进行排序。通过对第 2 个表格进行逐列读取构建密文。Bob 按照相反的顺序完成这 3 个步骤。他逐行将密文写入第 1 个表格，对列进行重新排列，然后逐行读取第 2 个表格。注意虽然加密和解密使用了相同的密钥，但是算法按照相反的次序使用密钥。

流密码和块密码

文字上可以将对称密码分为两个大类别：流密码和块密码。

流密码　在流密码（**stream cipher**）中，每次加密和解密一个符号（例如一个字符或一个比特）。我们拥有一个明文流、一个密文流和一个密钥流。明文流称为 P，密文流称为 C，密钥流称为 K。

$$P = P_1P_2P_3 \ldots \qquad C = C_1C_2C_3 \ldots \qquad K = (k_1, k_2, k_3 \ldots)$$
$$C_1 = E_{k1}(P_1) \qquad C2 = E_{k2}(P_2) \qquad C3 = E_{k3}(P_3) \ldots$$

块密码　在块密码（**block cipher**）中，长度为 m（$m>1$）的一组明文符号一起被加密，从而创建一组同样长度的密文。基于这个定义，在块密码中，即使密钥由多个值组成也要使用单一的密钥加密整个块。在块密码中，一个密文块依赖于整个明文块。

组合　在实际应用中，虽然多个明文块被单独地加密，但是它们使用密钥流一块一块地加密整个

消息。也就是说，当我们看待数据块时，密码采用的是块密码，但是当我们看待整个消息时，每个数据块作为一个单一的单元，它是一个流密码。每一块使用加密前或加密过程中生成的不同的密钥。

现代对称密钥密码

直到现在，我们研究的传统对称密钥密码是面向字符的密码。随着计算机的出现，我们需要面向比特的密码。这是因为需要加密的信息不只包含文本；它可能也包含数字、图像、声音和视频数据。将这些类型的数据变成比特流并且加密比特流，然后发送加密后的比特流会比较方便的。另外，在比特级别看待文本时，每个字符被替换成 8 个（或 16 个）比特，这意味着数字符号变大了 8（或 16）倍，混合大量的符号增加了安全性。现代密码既可以是块密码也可以是流密码。

现代块密码

对称密钥的现代块密码（modern block cipher）加密一个 n 比特的明文块或解密一个 n 比特的密文块。加密和解密算法使用一个 k 比特的密钥。解密算法必须与加密算法相反，并且加密和解密操作必须使用相同的秘密密钥以保证 Bob 能够恢复 Alice 发送的消息。图 10-7 显示了一个现代块密码加密和解密的基本思想。

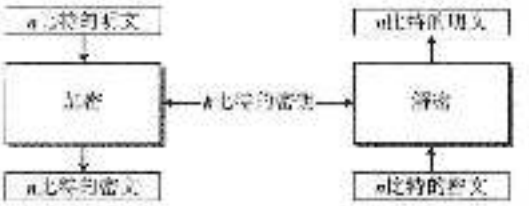


图 10-7　现代块密码

如果消息少于 n 个比特，那么必须进行填充以使它变成一个 n 比特的块。如果消息多于 n 个比特，那么应该把它分成多个 n 比特的块，并且如果需要就在最后一块进行填充。常用的 n 值为 64 比特、128 比特、256 比特和 512 比特。

现代块密码的组成　把块作为整体看，现代块密码为替换密码。可是，现代块密码没有被设计为一个单一的单元。为了提供抗攻击的密码，现代块密码由排列单元（有时称为 P-盒）、替换单元（有时称为 S-盒）、异或（XOR）操作、移位元素、交换元素、分裂元素和组合元素组成。图 10-8 显示了现代块密码的组成部分。

P-盒（P-box，排列盒）与传统的字符换位密码相似，但它对比特进行换位。在现代块密码中，我们可以看到 3 种类型的 P-盒：直接 P-盒、扩展 P-盒和压缩 P-盒。**S-盒**（S-box，替换盒）可以认为是一个小型的替换密码，但是它对比特进行替换。与传统的替换密码不同，S-盒输入和输出数可以不同。在多数块密码中，一个重要的组成单元是异或（exclusive-OR）操作，在这种操作中如果两个输入相同则输出 0，如果两个输入不同则输出 1。在现代块密码中，我们使用 n 个异或操作将 n 比特的数据片与 n 比特的密钥进行混合。在通常情况下，异或操作通常是应用密钥时使用的唯一操作。其他组成部分一般基于预定义的函数。

在一些现代块密码中可以看到的另一个组成单元是循环移位操作。移位可以向左移或者向右移。循环左移操作将一个 n 位的字向左移动 k 位；左面最高 k 位移去变成最右边的 k 位。交换操作是循环移位操作的一个特例，在这种情况下移动的位数 $k=n/2$。

在一些块密码中可以看到的另外两种操作是分裂和组合。分裂操作（split operation）将一个 n 位的字从中间分开，从而构建两个等长的字。组合操作（combine operation）正常情况下连接两个长度为 $n/2$ 的等长字，从而构建一个 n 位的字。

数据加密标准（DES）

作为现代块密码的一个实例，我们讨论数据加密标准（Data Encryption Standard，DES）。图 10-9 显示了加密端 DES 密码的元素。

在加密端，DES 输入一个 64 位的明文，构建一个 64 位的密文；在解密端，DES 输入一个 64 位的密文，构建一个 64 位的明文。加密和解密使用相同的 56 位密钥。

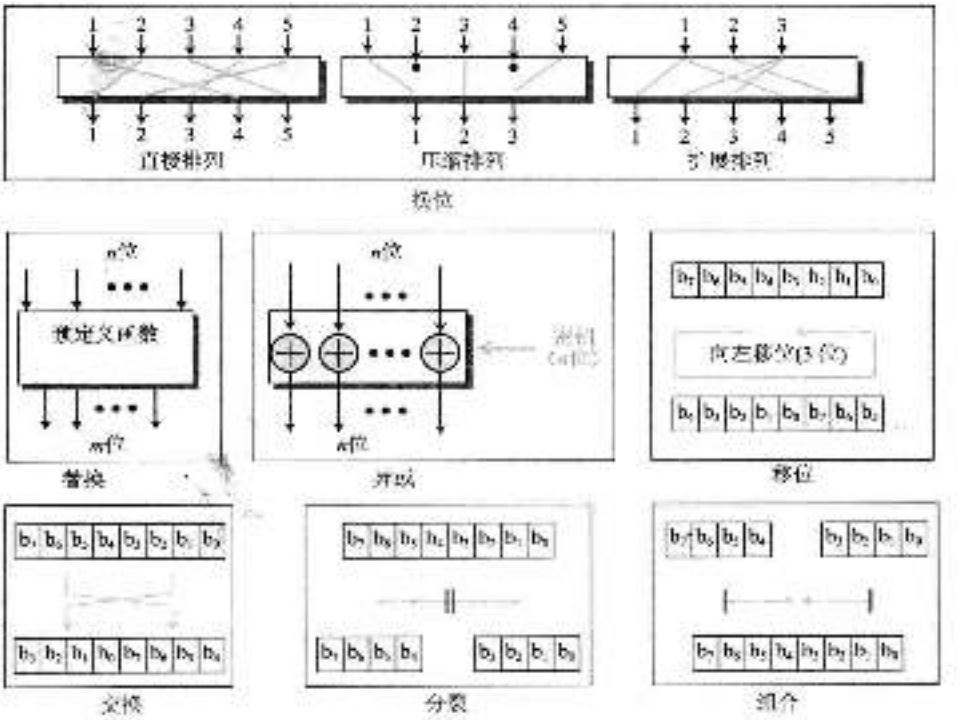


图 10-8 现代块密码的组成

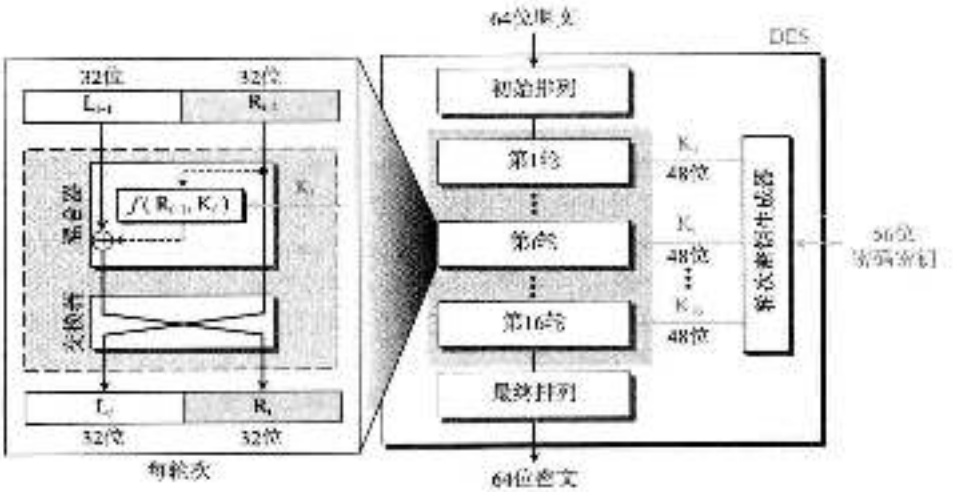


图 10-9 DES 的一般结构

初始排列将 64 位的输入块按照预先定义的规则进行排列。最终排列与初始排列相反。这两个排列抵消相互的影响。也就是说，如果忽略掉轮次结构，那么明文和密文是一样的。

轮次 DES 使用 16 个轮次。DES 的每一轮都是一个可逆的（Feistel）变换，如图 10-9 所示。每个轮次以上个轮次（或初始排列盒）的 L_{i-1} 和 R_{i-1} 作为输入，构建一个传输到下一轮次（或最终排列盒）的 L_i 和 R_i。每个轮次可以有两种密码元素（混合器和交换器）。这些元素都是可逆的。很明显，交换器是可逆的。它将文本的左半部分和右半部分进行交换。由于使用了 XOR 操作，混合器也是可逆的。所有不可逆的元素都集中于函数 $f(R_{i-1}, K_i)$。

DES 函数　DES 的中心是 DES 函数。DES 函数对最右边的 32 位（R_{i-1}）应用一个 48 位的密钥，从而生成一个 32 位的输出。该函数由 4 部分组成：一个扩展 P-盒、一个白化器（它将密钥加上）、一组 S-盒和一个直接 P-盒，如图 10-10 所示。

由于 R_{i-1} 是一个 32 位的输入，而 K_i 是一个 48 位的密钥，因此我们首先需要将 R_{i-1} 扩展到 48 位。这个扩展排列遵循预先定义的规则。

在扩展排列之后，DES 对扩展后的右半部分和轮次密钥进行 XOR 操作。S-盒实施真正的混合。DES 使用 8 个 S-盒，每个都具有 6 位的输入 4 位输出。DES 的最后一个操作是 32 位输入到 32 位输出的直接排列。

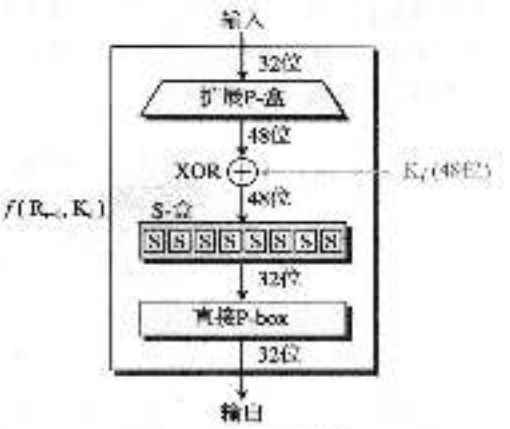


图 10-10　DES 函数

密钥生成　轮次密钥生成器从一个 56 位的密钥构造 16 个 48 位的密钥。可是，密码密钥通常按照 64 位给出，其中 8 个额外的比特为奇偶校验位。在实际密钥生成之前，需要去掉这 8 位，如图 10-11 所示。

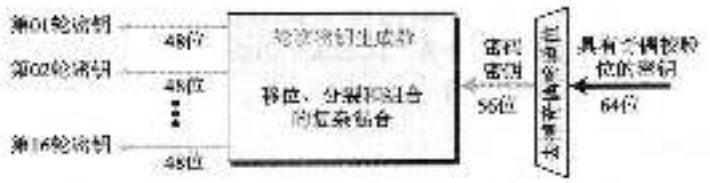


图 10-11　密钥生成

例 10.5　我们选择了一个任意的明文块和一个任意的密钥，生成（利用程序）的密文块（全部使用十六进制）如下所示。

明文：	密钥：	密文：
123456ABCD132536	AABB09182736CCDD	C0B7A8D05F3A829C

例 10.6　为了查看当一个输入位发生变化时 DES 的有效性，我们使用只有一位不同的明文（利用一个程序）。即使不改变密钥，两个密文也完全不相同。

尽管两个明文块只在右面的最高位不同，密文块出现了 29 位的不同。

明文：	密钥：	密文：
0000000000000000	22334512987AB823	4789FD476B82A5F1
明文：	密钥：	密文：
0000000000000001	22334512987AB823	0A4ED5C15A63FEA3

现代流密码

除了现代块密码外，我们也可能使用现代流密码。现代流密码和现代块密码之间的不同与传统流密码和块密码相似，这些我们已经在前面的部分进行了解释。在现代流密码（modern stream cipher）中，加密和解密一次处理 r 位。我们拥有一个明文比特流 $P = p_n \ldots p_2 p_1$，一个密文比特流 $C = c_n \ldots c_2 c_1$ 和一个密钥流 $K = k_n \ldots k_2 k_1$，其中 p_i、c_i 和 k_i 为 r 位的字。加密为 $c_i = E(k_i, p_i)$，解密为 $p_i = D(k_i, c_i)$。流密码速度快于块密码，其硬件实现也比较简单。当我们需要加密二进制流并且按照常数速率传输它们时，流密码是一种较好的选择。同时，流密码具有更好的抵抗传输过程中比特的毁坏能力。

最简单、最安全的同步流加密称为一次一密（one-time pad），Gilbert Vernam 享有这种加密的

发明权和专利权。一次一密密码利用一个随机选择的密钥流进行每次加密。加密和解密都采用单一的异或操作。基于异或操作的性质，加密算法和解密算法互逆。需要特别注意的是异或操作每次处理一个比特。另外还需要注意，必须存在一个安全通道，以便 Alice 可以向 Bob 发送密钥流序列（如图 10-12 所示）。



图 10-12　一次一密

一次一密是一种理想的密码，它非常完美。对于不能猜到密钥或明文和密文的统计特性。明文和密文之间也没有关系。也就是说，即使明文存在一些格式，密文也是一个真正随机的比特流。除非 Eve 尝试所有可能的随机密钥流，否则她不能破译这个密码。如果密文的长度为 n 位，那么所有可能的随机密钥流将是 2^n。但是，这里有一个问题。发送者和接收者每次通信时怎么共享一次一密的密钥？所以，这种完美的、理想的密码实现非常困难。不过有一些可行的、安全性稍差的版本。一个常用的替代版本是反馈移位寄存器（feedback shift register，FSR），但是我们将这种有趣密码的讨论留给专门讨论安全内容的书籍。

10.2.2　非对称密钥密码

在上一部分，我们讨论了对称密钥密码。在这一部分，我们开始讨论非对称密钥密码（**asymmetric-key ciphers**）。对称和非对称密码将同时存在并继续服务社会。我们相信它们相互补充；一种密码的优越性可以弥补另一种密码的缺陷。

两个系统概念上的不同基于这些系统怎样保持秘密信息。在对称密钥密码系统中，秘密信息必须在两个人之间共享。在非对称密钥密码系统中，秘密信息是私有的（不共享）；每个人创建并持有他或她自己的秘密信息。

在一个拥有 n 个人的团体中，对称密钥密码系统需要 n（n-1）个共享的秘密信息，而非对称密钥密码系统只需要 n 个私有的秘密信息。对于拥有 1 百万成员的团体，对称密钥密码系统将要求 5 亿个共享的秘密信息，而非对称密钥系统加密要求 1 百万个私有的秘密信息。

对称密钥密码系统基于共享的秘密；
非对称密钥密码系统基于私有的秘密。

除了加密之外，还有其他一些安全方面需要非对称密钥密码系统。这些方面包括认证和数字签名。无论何时一个应用基于一个私有的秘密信息，我们就需要使用非对称密钥密码系统。

对称密钥密码系统基于符号（字符或比特）的替换和排列，而非对称密钥密码系统基于数学函数的运用。在对称密钥密码系统中，明文和密文被当作一个符号组合；加密和解密排列这些符号或用一个替换另一个。在非对称密钥密码系统中，明文和密文被当作一个数字；加密和解密是数学函数对这些数字进行运算从而构造出另一个数字。

对称密钥密码系统对符号进行排列和替换；
非对称密钥密码系统对数字进行运算。

非对称密钥密码系统使用两个分离的密钥：一个私钥和一个公钥。如果加密和解密被认为是用

钥匙加锁和解锁，那么使用公钥加锁的锁头只能够使用对应的私钥解锁。如果 Alice 使用 Bob 的公钥对锁头加锁，那么只有 Bob 的私钥可以打开它，如图 10-13 所示。



图 10-13 非对称密钥密码系统中的加锁和解锁

从图中可以看到，与对称密钥密码系统不同，在非对称密钥密码系统中存在不同的密钥：一个私钥（private key）和一个公钥（public key）。尽管有些书使用秘密密钥（secret key）代替私钥，但是我们仅仅在对称密钥密码系统中使用秘密密钥，在非对称密钥密码系统中我们使用私钥和公钥。我们甚至使用不同的符号表示这三种密钥。也就是说，我们不希望秘密密钥与私钥互换使用，它们是两种不同类型的秘密信息。

非对称密钥密码系统也称为公钥密码。

基本思想

图 10-14 显示了一个用于加密的非对称密钥密码系统的基本思想。

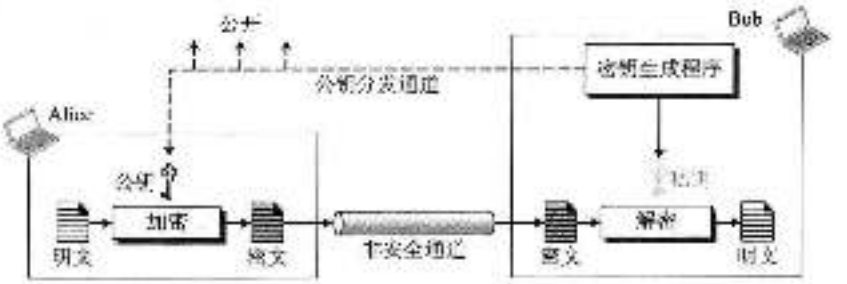


图 10-14 非对称密钥密码系统的基本思想

图 10-14 显示了几个重要概念。首先，它强调了密码系统的非对称特性。提供安全性的职责主要由接收方承担（在这个例子中为 Bob）。Bob 需要构建两个密钥：一个私钥和一个公钥。Bob 负责向团体发布这个公钥，发布过程可以通过一个公钥发布通道实现。尽管这个通道不要求是秘密的，但是它必须进行认证并保证完整性。Eve 应该不能用她的公钥假冒 Bob 的公钥向团体发布。

其次，非对称密钥密码系统意味着 Bob 和 Alice 不能使用相同的密钥集进行双向通信。团体中的每个实体应该构建它自己的私钥和公钥。图 10-14 显示了 Alice 怎样利用 Bob 的公钥向 Bob 发送加密的消息。如果 Bob 希望进行响应，那么 Alice 需要构建她自己的私钥和公钥。

再次，非对称密钥密码系统意味着 Bob 只需要一个私钥接收来自团体中任何人的所有响应，而 Alice 需要 n 个公钥与团体中的 n 个实体进行通信，每个实体一个公钥。也就是说，Alice 需要一个公钥环。

明文/密文

与对称密钥密码系统不同，在非对称密钥密码系统中，明文和密文被当作整数对待。在加密前，

消息必须编码为一个整数（或一个整数集）；在解密后，整数（或整数集）必须解码为消息。非对称密钥密码系统通常用于加密或解密小块的信息，例如一个对称密钥密码系统中的密码密钥。也就是说，非对称密钥密码系统通常用于辅助功能而不是消息加密。可是，这些辅助性功能在今天的密码系统中扮演着非常重要的角色。

非对称密钥密码系统通常用于加密或解密小块的信息。

加密/解密

在非对称密钥密码系统中，加密和解密是应用于数字的数学函数。这些数字代表着明文和密文。密文可以认为是 $C = f(K_{public}, P)$；明文可以认为是 $P = g(K_{private}, C)$。加密函数 f 只用于加密，解密函数 g 只用于解密。

二者兼有

一个非常重要但有时被误解的事实是：非对称密钥（公钥）密码系统的出现不排除人们对对称密钥（秘密密钥）密码系统的需要。其原因是非对称密钥密码系统采用数学函数进行加密和解密，其速度比对称密钥密码系统慢很多。大块信息的加密仍然需要对称密钥密码系统。在另一方面，对称密钥密码系统的速度也不排除人们对非对称密钥密码系统的需要。认证、数字签名和秘密密钥交换仍然需要非对称密钥密码系统。这意味着，为了能够实现目前安全的所有特征，我们既需要对称密钥密码系统也需要非对称密钥密码系统。两者相辅相成。

RSA 密码系统

尽管存在几种非对称密钥密码系统，但是常用的公钥算法之一为 RSA 密码系统（RSA cryptosystem）。该系统是根据发明者（Rivest、Shamir 和 Adleman）的名字命名的。RSA 使用两个指数 e 和 d，其中 e 为公钥，d 为私钥。假设 P 为明文，C 为密文。Alice 利用 $C = P^e \bmod n$ 由明文 P 创建密文 C；Bob 利用 $P = C^d \bmod n$ 恢复由 Alice 发送的明文。模 n 为一个非常大的数，是在密钥生成过程中构建的。

处理过程

图 10-15 显示了 RSA 处理过程背后的基本思想。

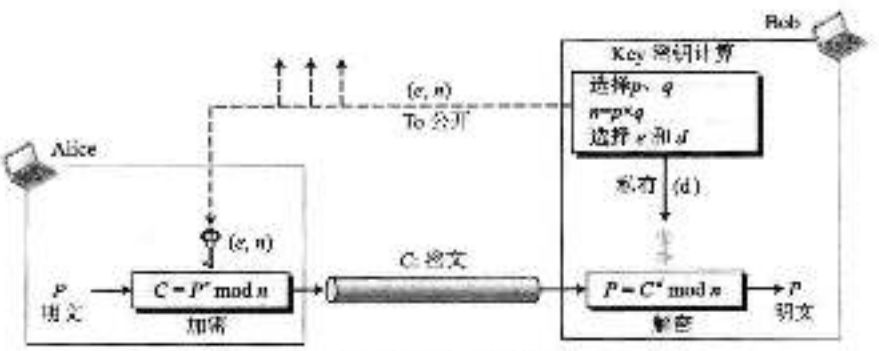


图 10-15　RSA 的加密、解密和密钥生成

Bob 选择两个大数 p 和 q，并计算 $n = p \times q$ 和 $\phi = (p-1) \times (q-1)$。然后 Bob 选择 e 和 d，并使 $(e \times d) \bmod \phi = 1$。作为公钥，Bob 向他所在的团体公开 e 和 n；作为私钥，Bob 保存 d。包括 Alice 在内的任何人可以利用 $C = (P^e) \bmod n$ 加密消息并将密文发送给 Bob；只有 Bob 能够利用 $P = (C^d) \bmod n$ 解密这个消息。如果 p 和 q 为非常大的数字（Eve 不知道 d），那么类似 Eve 一样的入侵者不能解密这个消息。

例 10.7 为了说明问题，Bob选择7和11作为p和q，同时计算$n=7\times11=77$。$\phi(n)$的值为$\phi(n)=(7-1)(11-1)=60$。如果他选择e为13，那么d就是37。注意$e\times d \bmod 60=1$。现在想象Alice希望向Bob发送明文5。她使用公共的指数13对5加密。由于p和q很小，因此该系统不安全。

明文: 5	密文: 26
$C=5^{13}=26 \bmod 77$	$P=26^{37}=5 \bmod 77$
密文: 26	明文: 5

例 10.8 这是一个利用Java程序计算的更加实际的一个例子。我们选择一个512位的p和q，同时计算n和$\phi(n)$。然后我们选择e并计算d。最后我们显示了加密和解密的结果。整数p由一个159个数字组成。

```
p =  9613034531358350457419158128061542790930984559499621582258315087964794045505647063849125716018034750312098666606492420191808780667421096063354219926661209
```

整数q由160个数字组成。

```
q =  12060191957221446918276794204450896001555925054637033936061798321731482148483764659215389453209175225273226830107120695604602513887145524969000359660045617
```

模数$n=p\times q$，它有309个数字。

```
n =  115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656772727460097082714127730434960500556347274566628060099924037102991424472292215772798521727033839381334692684137327622000966676671831831088373420823444370953
```

$\phi(n)=(p-1)(q-1)$ 有309个数字。

```
φ(n) =  115935041739676149688925098646158875237714573754541447754855261376147885408326350817276878815968325168468849300625485764111250162414552339182927162507656751054233608492916752034482627988117554787657013923444405716989581728196098226361075467211864612171359107358640614008885170265377277264467341066243857664128
```

Bob选择e=35535（理想值为65537）。然后他寻找d。

```
e =  35535
d =  5800830286003776393609366128967791759466906208965096218042286611138059385282235873170628691003002171085904433840217072986908760061152062025249598844480475682409662470814858171304632406440777048331340108509473852956450719367740611973265574242372176176746207763716420760033708533328853214470885955136670294831
```

Alice希望发送消息"THIS IS A TEST"，这个消息可以利用00-26编码方案变换成一个数字值（26代表空格字符）。

```
P =  1907081826081826002619041819
```

Alice计算的密文为$C=P^e$，其结果如下所示。

```
C =  47530912364622682720636555061054518094237179607049171652323924305445296061319932856661784341835911415119741125200568297979457173603610127821884789274156609048002350719071527718591497518846588863210114835410336165789846796838676373376577746562507928052114814184404814184430812773059004692874248559166462108656
```

Bob可以利用$P=C^d$从密文恢复明文，其结果如下所示。

```
P =  1907081826081826002619041819
```

解码之后，恢复出的明文为"THIS IS A TEST"。

应用

尽管RSA能够用来加密和解密实际的消息，但是如果消息较长，那么它显得非常慢。因此，RSA对短消息是有用的。特别地，我们将看到RSA用于数字签名和其他密码系统中，它们常常需要加密一个短消息而不希望访问对称密钥。在本章的后面我们将会看到，RSA也用于认证。

10.3　安全的其他方面

直到现在，我们讨论的密码系统都是提供机密性。可是在现代通信中，我们需要考虑安全的其他方面，例如完整性、消息和实体的认证、不可否认性和密钥管理。这一部分我们简单讨论这些问题。

10.3.1　消息完整性

有的场合我们不需要保密而必须保证完整性：消息应该保持未被修改状态。例如，Alice 可能写了一个在她去世后财产分配的遗嘱。这个遗嘱不需要加密。在她去世后，任何人都可以查对该遗嘱。可是，遗嘱的完整性需要保护。Alice 不希望遗嘱的内容被修改。

消息和消息摘要

保护文档完整性的一种方法是使用指纹（fingerprint）。如果 Alice 需要确保她的文档的内容不被修改，那么她可以在文档的底部按上她的指纹。由于 Eve 不能伪造 Alice 的指纹，因此她不能修改这份文档的内容或构造一份假的文档。为了确保文档没有被修改过，文档上 Alice 的指纹可以与文件上 Alice 的指纹相比较。如果它们不同，那么这份文档就不是出自 Alice 的。与文档和指纹对相对应的电子等价物是消息和摘要对。为了保证一个消息的完整性，这个消息需通过一个称为加密散列函数（**cryptographic hash function**）算法的运算。这个函数构建一个称为摘要（digest）的消息压缩映像，该映像可以像指纹一样使用。为了检查一个消息或文档的完整性，Bob 再次运行加密散列函数，并且将新的摘要与原来的摘要相比较。如果两者相同，Bob 相信原始的消息没有被修改过。图 10-16 显示了这种思想。

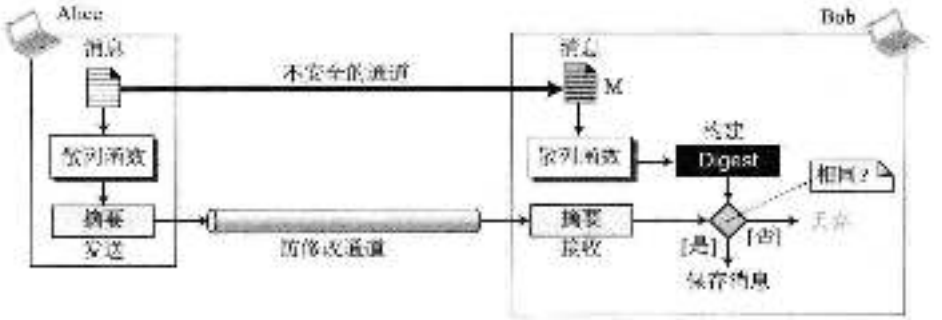


图 10-16　消息和摘要

文档/指纹和消息/消息摘要这两对信息是相似的，但也有一些不同。文档与指纹物理上链接在一起。消息和消息摘要可以不链接在一起（或者说可以分开发送），另外更重要的是需要保护消息摘要的安全，以免遭到修改。

> 需要保护消息摘要的安全，以免遭到修改。

散列函数

加密散列函数以任意长度的消息作为输入，创建一个固定长度的消息摘要。所有的加密散列函数都需要将可变长度的消息变成一个固定长度的摘要。构建这样一个函数最好使用迭代实现。我们不是使用一个以可变长度作为输入的散列函数，而是构建一个以固定长度作为输入的函数并重复使用该函数数次。这个固定长度输入函数叫做压缩函数（compression function）。它将一个 n 位的串压缩成一个 m 位的串，这里 n 通常大于 m。这种方案称为迭代加密散列函数（iterated cryptographic hash function）。

Ron Rivest 设计了几个散列函数。这些函数称为 MD2、MD4 和 MD5，这里 MD 代表消息摘要

(Message Digest)。最新的版本 MD5 是 MD4 的加强版，它把消息分成多个 512 位的块，创建一个 128 位的摘要。但是，它生成的 128 位的消息摘要比较小以至于不能抵抗攻击。

作为对不安全 MD 散列算法的反应，安全散列算法出现了。**安全散列算法**(Secure Hash Algorithm, SHA)是美国国家标准技术研究所(National Institute of Standards and Technology, NIST)开发的标准。SHA 已经经历了几个版本。

10.3.2 消息认证

摘要能够用于检查消息的完整性——即消息没有被修改过。为了确保消息的完整性和数据源认证——即 Alice 是消息的创建人而不是其他人——我们需要在处理过程中包含一个由 Alice 和 Bob 共享的秘密，需要创建一个**消息认证码**(message authentication code, MAC)。图 10-17 显示了这种思想。

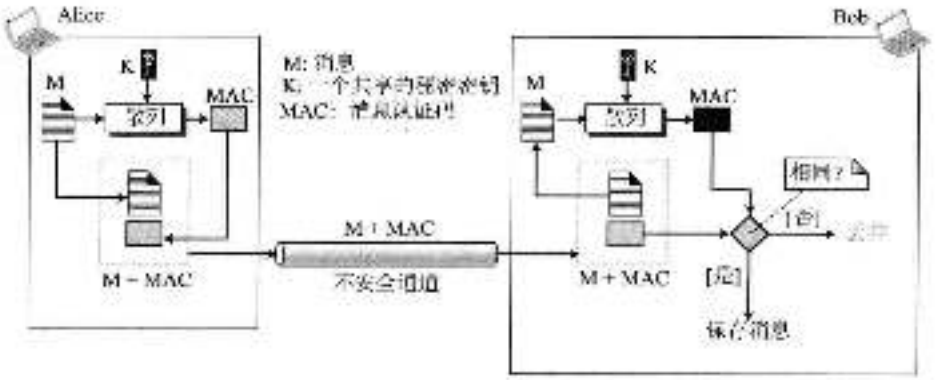


图 10-17 消息认证码

Alice 将密钥和消息串联后作为一个散列函数的输入，即 h(K+M)，从而创建一个 MAC 值。她通过非安全的通道向 Bob 发送消息和 MAC。Bob 将消息和 MAC 分离，然后通过串联消息和秘密密钥形成一个新的 MAC。之后，Bob 将新形成的 MAC 与接收的 MAC 进行比较。如果两个 MAC 相匹配，那么消息就是真实的并且没有被修改过。

注意，在这种情况下没有必要使用两个通道。消息和 MAC 都可以在相同的、不安全的通道中发送。Eve 可以看到消息，但是由于 Eve 没有 Alice 和 Bob 之间共享的密钥，因此她不能伪造一个新消息进行替代。她也不能创建与 Alice 形成的 MAC 相同的 MAC。

利用散列函数和秘密密钥的组合，一个 MAC 能够提供消息的完整性和消息认证。

HMAC

美国国家标准技术研究所(NIST)发布了一个标准式 MAC 标准，它通常被称为 HMAC(散列消息认证码，hashed MAC)。实现 HMAC 比实现简单 MAC 复杂得多，本书不需要这些内容。

10.3.3 数字签名

提供消息完整性、消息认证(和一些更安全的服务，我们稍后将会看到)的另一种方法是数字签名。MAC 利用一个秘密密钥保护摘要；而数字签名使用了一个私钥-公钥对。

一个数字签名使用一个私钥-公钥对。

我们都很熟悉签名的概念。一个人签署一份文档以表示这份文档源自于她或由她批准。签名对接收者来说就是文档来自于正确实体的证据。当一个顾客签署了一张支票，银行需要确信这张支票

是由那个客户而不是其他人发出了。也就是说，文档上的签名在验证后就是一个鉴定的标记——文档是真实的。考虑由一位艺术家签署的一幅画。艺术品上的签名如果是真的，那么意味着这幅画可能也是真的。

当 Alice 向 Bob 发送一个消息时，Bob 需要检查发送者的真实性；他需要确信这个消息来自于 Alice 而不是 Eve。Bob 可以要求 Alice 电子签署该消息。也就是说，一个电子签名可以证明 Alice 作为消息发送者的真实性。我们把这种类型的签名叫做**数字签名**（**digital signature**）。

比较

我们通过观察传统签名和数字签名的不同来开始这一部分。

包含

传统签名包含在文档中；它是文档的一部分。当我们写一张支票时，签名就在支票上；它不是一个分离的文档。但是当我们数字地签署一个文档时，我们将签名作为一个分离的文档进行发送。

验证方法

这两种签名的第二个不同之处是签名的验证方法。对于传统签名，当接收者收到一份文档时，她比较文档上的签名与文件上的签名。如果它们相同，那么这份文档就是真实的。接收者需要拥有一份文件上签名的副本，以便进行比较。对于数字签名，接收者接收消息和签名。签名的拷贝不会存储在任何地方。接收者需要应用一种验证技术来组合消息和签名以证明其真实性。

关系

对于一个传统的签名，签名和文档之间正常是一对多的关系。一个人使用同样的签名签署多份文档。对于一个数字签名，签名和消息之间是一对一的关系。每个消息有它自己的签名，一个消息的签名不能用于其他消息。如果 Bob 接收到两条来自于 Alice 的消息，一条在另一条之后，那么他不能利用第一条消息的签名验证第二条消息。每条消息都需要一个新的签名。

二重性

两种签名类型的另一个不同之处称为二重性。对于传统的签名，被签署文档的副本区分于原始文件上的签名。对于数字签名，除非在文档上有时间因素（如时间戳），否则就没有这样的区分。例如，假设 Alice 发送了一份指示 Bob 向 Eve 付款的文档。如果 Eve 截获了这个文档和签名，那么她可以稍后重新发送它并再次从 Bob 那里获得付款。

处理过程

图 10-18 显示了数字签名的处理过程。发送者使用一个签名算法（signing algorithm）来签署消息。消息和签名被送往接收者。接收者接收消息和签名并对其组合应用验证算法（verifying algorithm）。如果结果正确，那么接受这个消息；否则，拒绝它。

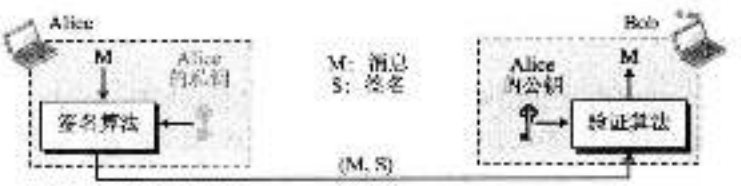


图 10-18 数字签名处理过程

传统签名就像一把属于文档签署者的私有“钥匙”。签署者使用这把“钥匙”签署文档；没有其他人拥有这个签名。文件上签名的副本就像一把公开的“钥匙”；任何人可以使用它对文档进行验证，将它与原始签名进行比较。

在数字签名中，签署者将她的私钥应用于签名算法，进而签署文档。在另一方面，验证者将签署者的公钥应用于验证算法，进而验证文档。

注意，当一份文档被签署后，由于每个人都可以访问 Alice 的公钥，因此任何人（包括 Bob 在内）都可以对它进行验证。Alice 一定不能使用她的公钥签署文档，因为那样的话任何人都可以伪造她的签名。

我们能否使用一个秘密（对称）密钥进行签名和验证签名呢？由于一些原因，答案是否定的。首先，仅仅两个实体（例如 Alice 和 Bob）知道秘密密钥。所以如果 Alice 需要签署另一份文档并将它发送给 Ted，那么她需要使用另一个秘密密钥。其次，正如我们将看到的，为一个会话构建一个秘密密钥涉及认证，认证又要使用数字签名。我们进入了一个无休止的循环。再次，Bob 能够使用他自己和 Alice 之间的秘密密钥签署一份文档，并且把它发送给 Ted，假装这份文档来自于 Alice。

数字签名需要一个公钥系统。

签署者使用他的私钥进行签名；验证者使用签署者的公钥进行验证。

我们应该区分清楚数字签名中使用的私钥和公钥与密码系统中保证机密性时使用的公钥与私钥。后者在处理过程中使用接收者的公钥和私钥。发送者使用接收者的公钥进行加密；接收者使用他自己的私钥进行解密。数字签名使用发送者的私钥和公钥。发送者使用她的私钥；接收者使用发送者的公钥。

密码系统使用接收者的私钥和公钥；

数字签名使用发送者的私钥和公钥。

签署摘要

我们之前说过，非对称密钥密码系统在处理长信息时非常低效。在数字签名系统中，虽然消息通常都比较长，但是我们又不得不使用非对称密钥方案。其解决方法是签署一个消息的摘要，该摘要比消息短很多。一个精心选取的消息摘要具有与消息一对一的关系。发送者可以签署消息摘要，接收者可以对消息摘要进行验证，其效果是相同的。图 10-19 显示了数字签名系统中签署一个摘要的方法。



图 10-19 签署摘要

在 Alice 一端，通过消息形成一个摘要。然后利用 Alice 的私钥对该摘要进行签名。之后 Alice 向 Bob 发送消息和签名。

在 Bob 一端，首先利用相同的公开散列函数通过接收到的消息构建摘要。然后应用验证处理过程。如果是可信的，接收消息；否则拒绝它。

服务

在本章的开始，我们讨论了几种安全服务。这些服务包括消息机密性、消息认证、消息完整性和不可否认。数字签名能够直接提供后三种服务；对于消息机密性，我们仍然需要加密/解密。

消息认证

像安全的传统签名（不容易被复制的签名）一样，一个安全的数字签名方案能够提供消息认证（也称为数据源认证）。由于在验证中使用 Alice 的公钥，因此 Bob 能够验证这条消息由 Alice 发送。

Alice 的公钥不能验证由 Eve 的私钥签署的签名。

消息完整性

因为如果改变消息的任何部分，我们都不能得到同样的摘要，所以如果我们对消息或消息的摘要进行签名，那么就能保证消息的完整性。今天的数字签名方案在签名算法和验证算法中使用散列函数，用于保护消息的完整性。

不可否认

如果 Alice 签署了一条消息而后又否认它，那么 Bob 以后能够证明 Alice 确实签署过它吗？例如，如果 Alice 向银行（Bob）发送了一条消息，要求从她的账户转 10 000 美金到 Ted 的账户，那么以后 Alice 能够否认她发送过这条消息吗？对于我们直到现在给出的方案，Bob 可能会遇到问题。Bob 必须把签名保存在文件中，以后可以利用 Alice 的公钥构建原始消息，以证明文件中的消息与新构建的消息相同。由于 Alice 在这期间可能已经改变了她的私钥和公钥，因此这种方法是不可能的。她也可以声明文件中包含的签名是不可信的。

一种解决方案是可信任的第三方。人们可以在他们之间构建一个都可以信任的机构。在本章的后面，我们将看到一个可信任的机构可以解决很多与安全服务和密钥交换相关的其他问题。图 10-20 显示了一个可信任机构怎样能够防止 Alice 否认她发送过信息。

Alice 通过她的消息构建一个签名（S_A），然后向中心发送这个消息、她的身份标识、Bob 的身份标识和生成的签名。在检查 Alice 的公钥有效后，中心通过 Alice 的公钥验证消息来自于 Alice。然后，中心将该消息的副本、发送者标识、接收者标识和时间戳进行归档。中心利用它的私钥为这个消息构建另一个签名（S_T）。之后，中心向 Bob 发送这个消息、新的签名、Alice 的标识符和 Bob 的标识符。Bob 使用可信中心的公钥验证这个消息。

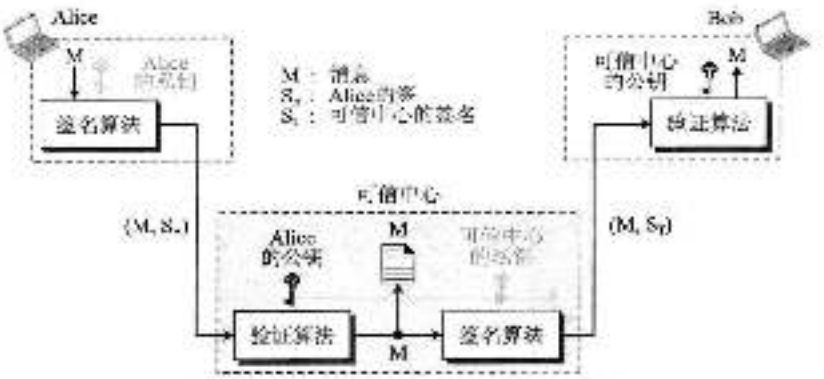


图 10-20　利用可信中心提供不可否认服务

如果将来 Alice 否认她发送过消息，中心能够展示一个它保存信息的副本。如果 Bob 的消息与中心保存的消息相同，那么 Alice 将失去争辩的权利。为了保证所有信息的机密性，需要向方案中增加一定级别的加密/解密。这些内容我们将在下一部分讨论。

机密性

数字签名不能提供机密性的通信。如果要求机密性，那么消息和签名必须利用对称密钥或非对称密钥进行加密。

RSA 数字签名方案

近几十年中，一些数字签名方案不断发展和演化，其中一些已经实现。在这一部分，我们主要展示其中之一，即 RSA。在上一部分，我们讨论了怎样利用 RSA 密码系统提供隐私保护。RSA 的思想也能用于消息的签名和验证。在这方面，它叫做 RSA 数字签名方案（RSA digital signature

scheme）。数字签名方案变换了私钥和公钥的角色。首先，它使用了发送者而不是接收者的私钥和公钥。其次，发送者使用她自己的私钥签署文档；接收者使用发送者的公钥验证它。如果我们将这个方案与传统的签名方法进行比较，我们可以看到私钥扮演着发送者自己签名的角色，发送者的公钥扮演着签名副本的角色，该签名副本可以向公众公开。显然，Alice 不能使用 Bob 的公钥对消息进行签名，这是因为任何其他的人都可以这样做。虽然签名方和验证方使用同样的函数，但是它们使用不同的参数。验证者比较消息与函数输出是否按模数运算后相等。如果结果为真，那么接受消息。图 10-21 显示了这种方案。其中由于公钥密码系统对于长消息效率不高，因此签名和验证采用了消息摘要而不是消息本身；摘要比消息本身小得多。

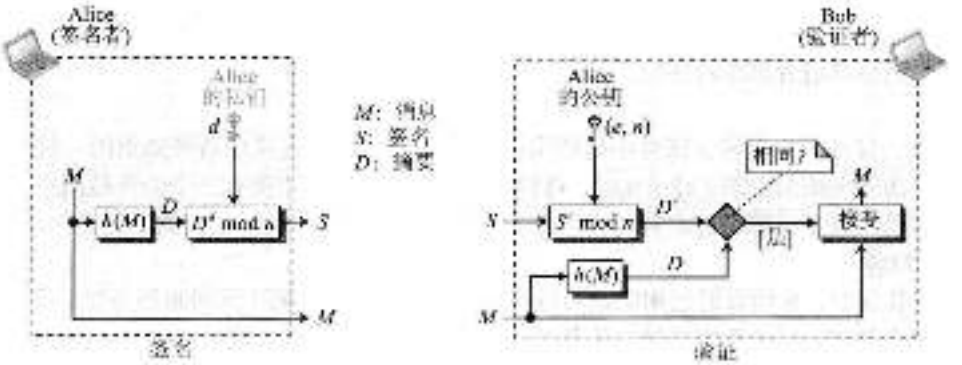


图 10-21 对消息摘要进行 RSA 签名

签名者 Alice 首先利用商定好的散列函数为消息构建一个摘要，$D=h(M)$。然后，她对这个摘要签名，$S=D^d \bmod n$。这个消息和签名被发送给 Bob。验证者 Bob 接收消息和签名。他首先利用 Alice 的公开指数对摘要进行恢复，$D'=S^e \bmod n$。然后，他对接收到的消息实施散列算法，得到 $D=h(M)$。现在 Bob 比较这两个摘要，D 和 D'。如果它们相等（按照模数运算），那么他接受这个消息。

数字签名标准（DSS）

数字签名标准（Digital Signature Standard，DSS）于 1994 年被 NIST 采纳。DSS 是一个复杂的、更加安全的数字签名方案。

10.3.4 实体认证

实体认证这种技术用来让一方验证另一方的身份。一个实体（entity）可以是一个人、一个进程、一个客户或一个服务器。需要被证明身份的实体称为申请者（claimant）；试图验证申请者身份的一方称为验证者（verifier）。

实体认证与消息认证

实体认证与消息认证（数据源认证）存在两点不同。

1. 消息认证（或数据源认证）可能不是实时发生的；而实体认证是实时的。对于前者，Alice 向 Bob 发送了一条消息。当 Bob 认证消息时，Alice 既可以出现在现场也可以不出现在现场。在另一方面，当 Alice 请求实体认证时，Alice 被 Bob 认证完毕才会发生实实在在的消息通信。Alice 需要在线并参与处理过程。只有在她被认证后，消息才能在 Alice 与 Bob 之间进行传输。当 Alice 向 Bob 发送一封电子邮件时，需要进行数据源认证。当 Alice 从一台自动取款机提取现金时，需要进行实体认证。

2. 消息认证简单地认证一条消息；对于每一条新消息，处理过程将要重复执行。实体认证用来在整个会话期间认证申请者的身份。

验证分类

在实体认证中，申请者必须向验证者标识她自己。标识自己可以通过三类证据之一实现：所知的（something known）、所有的（something possessed）或固有的（something inherent）。

- 所知的。这是只有申请者知道的秘密，该秘密可以被验证者验查。例如密码、PIN 码、秘密密钥和私钥。
- 所有的。这是能够证明申请者身份的一些信息。例如护照、驾驶执照、身份证、信用卡和智能卡。
- 固有的。这是申请者固有的特征。例如传统的签名、指纹、声音、面部特征、视网膜和笔迹。

在这一部分，我们只讨论第一种证据类型，即所知的。它通常用于远程（在线）实体认证。其他两种通常是在申请者出席时使用。

密码

最简单、最古老的实体认证方法是使用密码（password）。这是申请者知道的一些信息。当一个用户需要访问系统的资源（登录）时，可以使用密码。每个用户拥有一个公开的用户标识符和一个私有的密码。可是，密码很容易受到攻击。密码可能被窃取、截获、猜测等等。

挑战–响应

在密码认证中，申请者通过阐明她知道一个秘密（也就是密码）证明她的身份。可是，由于申请者发送这个秘密，因此很容易被对手截获。在挑战–响应认证（challenge-response authentication）中，申请者证明她知道一个秘密但不用发送它。也就是说，申请者不向验证者发送秘密；验证者或者拥有它或者能找到它。

在挑战–响应认证中，申请者证明她知道一个秘密但不必将它发送给验证者。

挑战（challenge）是一个随时间变化的值，如一个随机数或者时间戳。该数值由验证者发送。申请者对这个挑战应用一个函数并向验证者发送结果。这称为响应（response）。响应表示申请者知道这个秘密。

使用对称密钥密码

有几种挑战–响应认证采用对称密钥加密。这里的秘密就是申请者和验证者共享的秘密密钥。函数就是应用于挑战的加密算法。尽管这种方案可以使用几种方法实现，我们只展示最简单的一种以介绍其思想。图 10-22 显示了第一种方法。

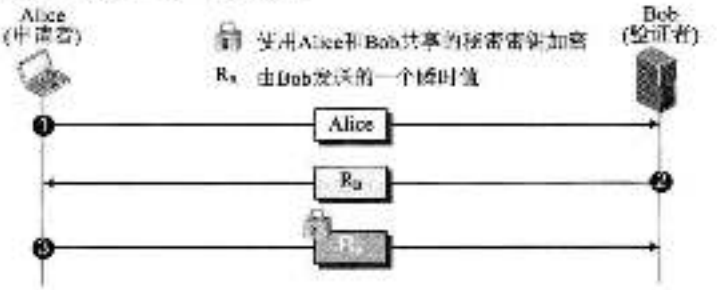


图 10-22　单向、对称密钥认证

第一条消息不是挑战–响应的一部分，它仅仅通知验证者申请者希望被挑战。第二条信息是挑战。R_B 为验证者（Bob）随机选取的瞬时值（nonce，number once 的缩写），用来挑战申请者。申请者使用只有申请者和验证者知道的共享秘密密钥加密这个瞬时值，然后向验证者发送结果。验证

者解密这条信息。如果解密获得的瞬时值与验证者发送的瞬时值相同，那么 Alice 被授权访问。

注意，在这个过程中，申请者和验证者需要保证处理过程中使用的对称密钥的秘密性。在响应返回之前，验证者还必须保存用于申请者收到的瞬时值。

使用非对称密钥密码

图 10-23 显示了这种方法。



图 10-23　单向、非对称密钥认证

除了对称密钥密码，我们可以使用非对称密钥密码进行实体认证。这里的秘密必须是申请者的私钥。申请者必须展示她拥有一个与大家都知道的公钥相关的私钥。这意味着验证者必须使用申请者的公钥对挑战进行加密；申请者使用她的私钥解密这个消息。对挑战的响应就是这个解密后的消息。

使用数字签名

实体认证也可以利用数字签名实现。当数字签名被用于实体认证时，申请者使用她的私钥进行签名。在第一种方法中，如图 10-24 所示，Bob 使用明文挑战，Alice 对响应签名。



图 10-24　数字签名、单向认证

10.3.5　密钥管理

在上一部分，我们讨论了对称密钥和非对称密钥密码系统。但是，我们还没有讨论对称密钥密码系统中的秘密密钥和非对称密钥密码系统中的公钥如何发布和管理。这一部分触及这两个问题。

对称密钥发布

对于加密大块的消息，对称密钥密码系统比非对称密钥密码系统效率更高。可是，对称密钥密码系统需要一个双方之间共享的秘密密钥。

如果 Alice 需要与 N 个人交换机密的消息，那么她需要 N 个不同的密钥。如果 N 个人相互之间进行通信需要多少密钥？如果我们要求两个人双向通信使用两个密钥，那么总共需要 $N(N-1)$个密钥；如果我们允许两个方向使用一个密钥，那么只需要 $N(N-1)/2$ 个密钥。这意味着如果一百万个人之间需要相互进行通信，那么每个人差不多都有一百万个不同的密钥；系统中总共需要 5 千亿个密钥。由于 N 个实体要求的密钥数接近于 N^2，因此这个问题通常也被称为 N^2 问题。

密钥数目不是唯一的问题；密钥发布是另外一个问题。如果 Alice 和 Bob 希望进行通信，那么他们需要一种交换秘密密钥的方法；如果 Alice 希望和一百万人通信，那么她如何与一百万人交换一百万个密钥？利用 Internet 绝对不是一个安全的方法。很明显，我们需要一种管理和发布秘密密钥的有效方法。

密钥分发中心：KDC

一种实用的解决方案是利用称为密钥分发中心（**key distribution center，KDC**）的可信任的第三方。为了减少密钥的数量，每个人与 KDC 建立一个共享的秘密密钥。KDC 与每个成员之间存在一个秘密密钥。现在的问题是 Alice 如何向 Bob 发送机密消息。其处理过程如下：

1. Alice 向 KDC 发送一个请求，表明她需要一个她和 Bob 之间进行会话（暂时）的秘密密钥。
2. KDC 将 Alice 的请求通知 Bob。
3. 如果 Bob 同意通信，那么两者之间创建一个会话密钥。

使用 KDC 在 Alice 和 Bob 之间创建的秘密密钥用于 KDC 认证 Alice 和 Bob，同时防止 Eve 假冒他们中的任何一个。

多重 KDC 当使用 KDC 的人数增加时，系统变得不可管理，并且也会产生瓶颈。为了解决这个问题，我们需要多重 KDC。我们可以把世界划分成多个域。每个域可以拥有一个或多个 KDC（在故障时作为备份使用）。现在我们假设 Alice 希望向 Bob 发送一个机密消息，而 Bob 属于另一个域。Alice 联系她的 KDC，这个 KDC 反过来联系 Bob 所在域中的 KDC。这两个 KDC 能够构建一个 Alice 和 Bob 之间的秘密密钥。KDC 可能为本地 KDC、国家 KDC 和国际 KDC。当 Alice 需要与 Bob 通信，而 Bob 生活在另一个国家，她向一个本地 KDC 发送请求；本地 KDC 中继这个请求到一个国家 KDC；国家 KDC 中继这个请求到一个国际 KDC。然后，这个请求沿着向下的路径被中继到 Bob 所在的本地 KDC。图 10-25 显示了一个层次化的多重 KDC 结构。

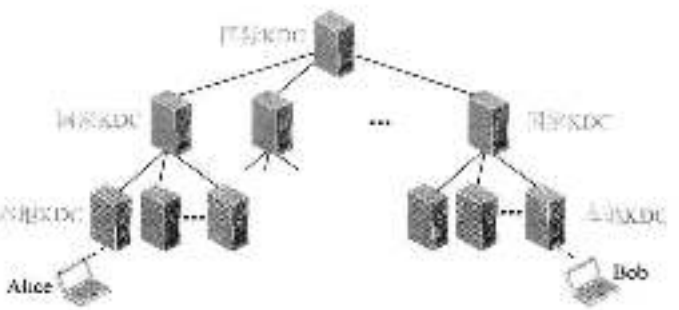


图 10-25 多重 KDC

会话密钥 KDC 为每个成员构建一个秘密密钥。这个秘密密钥只能够用于成员与 KDC 之间，而不是两个成员之间。如果 Alice 需要与 Bob 秘密地通信，她需要一个她自己与 Bob 之间的秘密密钥。利用 Alice 和 Bob 与中心的秘密密钥，KDC 可以在 Alice 和 Bob 之间创建一个会话密钥（session key）。Alice 和 Bob 的密钥用于会话密钥建立之前中心认证 Alice 和 Bob，以及他们之间相互认证。通信结束以后，会话密钥将不再有用。

> 双方之间的一个会话对称密钥只使用一次。

利用上面讨论的思想，人们提出了几种不同的方法来构建会话密钥。我们在图 10-26 中给出了最简单的方法。尽管这种方法非常基本，但是它能帮助理解文献中更加复杂的方法。

1. Alice 向 KDC 发送明文消息以获得 Bob 和她自己之间的对称会话密钥。这个消息包含了她的注册标识（图中的 Alice）和 Bob 的标识（图中的 Bob）。该消息没有加密，它是公开的。KDC

并不关心这些。

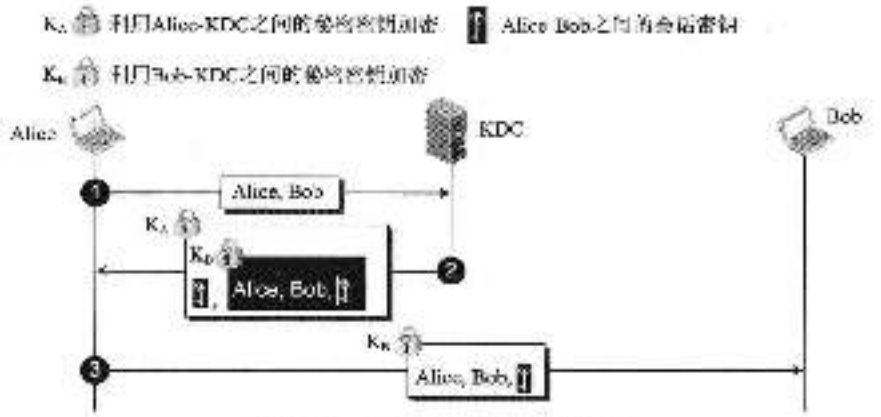


图 10-26 利用 KDC 创建会话密钥

2. KDC 接收这条消息，生成所需的票据（ticket）。该票据使用 Bob 的密钥（K_B）加密。票据包含了 Alice 和 Bob 的标识符，以及会话密钥。带有会话密钥副本的票据发送给 Alice。Alice 接收该消息，将其解密，然后提取出会话密钥。她无法解密 Bob 的票据；这个票据是给 Bob 的，不是给 Alice 的。注意这条消息是双重加密的——票据是加密的，整个消息也是加密的。在第二条消息中，因为只有 Alice 能够利用她与 KDC 之间的秘密密钥打开这个消息，所以 KDC 实际上认证了 Alice。

3. Alice 向 Bob 发送票据。Bob 打开票据，获知 Alice 希望利用会话密钥向他发送消息。注意在这条消息中，由于只有 Bob 能够打开票据，因此 KDC 认证了 Bob。因为 KDC 认证了 Bob，所以信任 KDC 的 Alice 也认证了 Bob。同样，因为 Bob 信任 KDC，而 KDC 发送给 Bob 的票据包含有 Alice 的标识符，所以 Bob 也认证了 Alice。

对称密钥协商

Alice 和 Bob 可以创建一个他们之间的会话密钥但不使用 KDC。这种会话密钥创建方法称为对称密钥协商（symmetric-key agreement）。尽管实现对称密钥协商存在几种不同的方法，但是我们只讨论 Diffie-Hellman 这一种。这种方法展示了在更加精密（不易受到攻击的）的方法中使用的基本思想。

Diffie-hellman 密钥协商

在 Diffie-Hellman 协议（Diffie-Hellman protocol）中，双方不需要 KDC 就可创建一个对称会话密钥。在建立对称密钥之前，双方需要选择两个数 p 和 g。这两个数拥有的一些特性通常在数论中讨论。但是这个讨论超出了本书的范围。这两个数不需要保密。它们可以通过 Internet 发送，并且可以公开。图 10-27 显示了这个过程。

具体步骤如下：

1. Alice 选择一个大随机数 x，使 $0 \leq x \leq p-1$ 并计算 $R_1 = g^x \bmod p$。Bob 选择另一个大随机数 y，使 $0 \leq y \leq p-1$ 并计算 $R_2 = g^y \bmod p$。

2. Alice 向 Bob 发送 R_1。注意 Alice 不发送 x 的值；她只发送 R_1。

3. Bob 向 Alice 发送 R_2。再次注意 Bob 不发送 y 的值；他只发送 R_2。

4. Alice 计算 $K = (R_2)^x \bmod p$。Bob 也计算 $K = (R_1)^y \bmod p$。

K 就是这个会话的对称密钥。

$$K = (g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$$

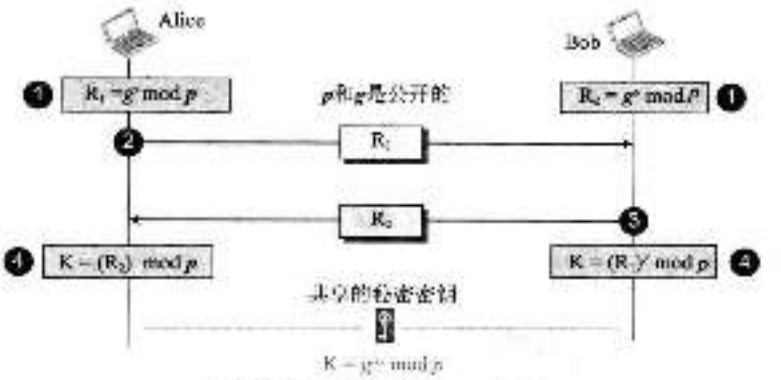


图 10-27 Diffie-Hellman 方法

Bob 计算 $K = (R_1)^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$。Alice 计算 $K = (R_2)^x \bmod p = (g^y \bmod p)^x \bmod = g^{xy} \bmod p$。在 Bob 不知道 x 的值和 Alice 不知道 y 的值的情况下，两者得到了相同的值。

在 Diffie-Hellman 方法中，对称（共享）密钥为 $K = g^{xy} \bmod p$。

例 10.9 为了清楚地说明这个过程，我们给出一个小示例。我们的例子使用小数字，但是请记住在真实的环境中，数字非常大。假设 $g=7$、$p=23$。具体步骤如下：

1. Alice 选择 $x=3$ 并计算 $R_1 = 7^3 \bmod 23 = 21$。Bob 选择 $y = 6$ 并计算 $R_2 = 7^6 \bmod 23 = 4$。
2. Alice 向 Bob 发送数字 21。
3. Bob 向 Alice 发送数字 4。
4. Alice 计算对称密钥 $K = 4^3 \bmod 23 = 18$。Bob 计算对称密钥 $K = 21^6 \bmod 23 = 18$。

对于 Alice 和 Bob，K 的值是相同的：$g^{xy} \bmod p = 7^{18} \bmod 23 = 18$。

公钥发布

在非对称密钥密码系统中，人们不需要知道对称的共享密钥。如果 Alice 想向 Bob 发送消息，她只需要知道 Bob 的公钥，该公钥是公开的，每个人都可以知道。如果 Bob 需要向 Alice 发送消息，他只需要知道 Alice 的公钥，该公钥每个人也可以知道。在公钥密码系统中，每个人都对私钥进行隐藏，而将公钥进行公告。

在公钥密码系统中，每个人都可以访问其他人的公钥；公钥是公开的。

为了实用，公钥像秘密密钥一样也需要发布。我们简单讨论一下公钥发布的方法。

公共通告

一种简单的方法是公开对公钥进行通告。Bob 可以把他的公钥放在他的网站上或者在本地或全国的报纸上公布。当 Alice 需要向 Bob 发送保密信息时，她可以从他的网站或报纸上获得 Bob 的公钥，甚至可以发送消息进行询问。可是，这种方法不安全，会遭受到假冒。例如，Eve 可能进行了这样一次公共通告。在 Bob 反应过来之前，已经造成了损害。Eve 可以欺骗 Alice 向她发送消息，该消息本来是想给 Bob 的。Eve 也可能使用相应的伪造的私钥签署一份文档，使每个人相信它是由 Bob 签署的。如果 Alice 直接请求 Bob 的公钥，这种方法也是脆弱的。Eve 可以截获 Bob 的响应，用她自己伪造的公钥替代 Bob 的公钥。

认证中心

发布公钥常用的方法是构建公钥证书（public-key certificate）。Bob 思考两件事情：他希望人们知道他的公钥；他希望没有人能接收到以他的名义伪造的公钥。Bob 可以利用认证中心（certification authority，CA）。认证中心是一个联邦或政府组织，它将公钥与实体进行绑定并发布证书。图 10-28

显示了这个概念。

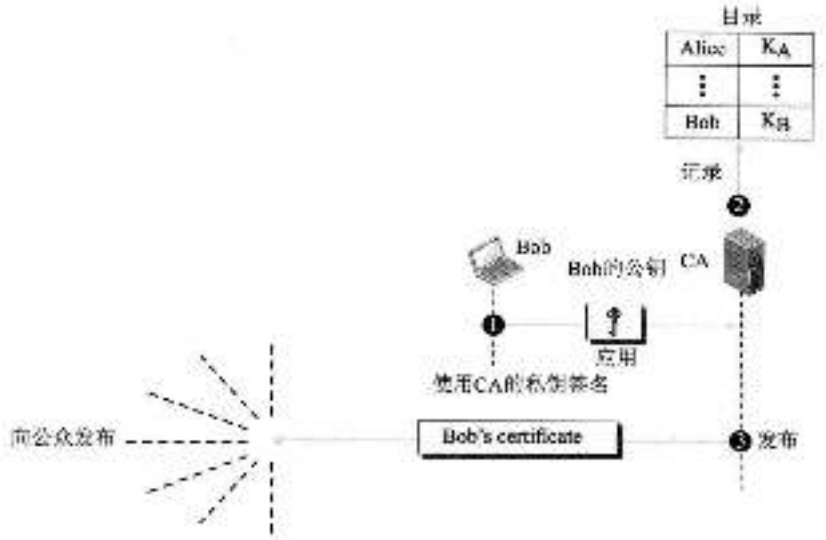


图 10-28　认证中心

CA 中心有一个众所周知的、无法伪造的公钥。CA 中心核实 Bob 的身份（采用图片 ID 和其他证据），并需要 Bob 的公钥并将它写在证书上。为了防止证书本身被假冒，CA 中心使用它的私钥对证书签名。现在 Bob 可以上传这个签名的证书。需要 Bob 公钥的任何人可以下载这个被签名的证书，并且利用认证中心的公钥提取 Bob 的公钥。

X.509

尽管 CA 中心解决了公钥欺诈问题，但是它也有负面影响。每份证书可能存在不同的格式。如果 Alice 需要使用程序自动下载属于不同人的不同证书和摘要，那么程序可能无法做到这一点。一份证书的公钥可能是一种格式，另一份证书可能是另一种格式。在第一份证书中，公钥可能出现在第一行，在另一份中可能出现在第三行。任何需要普及的事物必须拥有统一的格式。为了消除这种负面影响，ITU 开发了 X.509 标准。进行了一些修改后，X.509 已经被 Internet 采纳。X.509 按照结构化方法描述证书。它采用知名的 ASN.1 协议（已在第 9 章讨论）。ANS.1 协议定义的字段程序员非常熟悉。

10.4　Internet 安全

在这一部分，我们讨论密码原理怎样应用于 Internet。我们讨论应用层、传输层和网络层的安全。数据链路层的安全比较特殊，通常由 LAN 或 WAN 的设计者实现。

10.4.1　应用层安全

这一部分讨论两种对电子邮件提供安全服务的协议：良好隐私（Pretty Good Privacy，PGP）协议和安全多用途因特网邮件扩展（Secure/Multipurpose Internet Mail Extension，S/MIME）协议。

电子邮件安全

发送电子邮件是一次性的活动。这种活动的本质不同于我们将在后两部分看到的 SSL 和 IPSec。在 SSL 和 IPSec 中，我们假设双方在它们之间创建了一个会话，并且在两个方向上交换数据。电子邮件中不存在会话，Alice 和 Bob 不可能创建会话。Alice 向 Bob 发送了一个消息；一段时间以后，Bob 阅读这个消息，他既可以回复也可以不回复。由于 Alice 向 Bob 发送的内容独立于 Bob

向 Alice 发送的内容，因此，我们讨论单向消息的安全性。

密码算法

如果电子邮件是一种一次性的活动，那么发送方和接收方怎样为电子邮件协商密码算法呢？如果没有会话，没有握手信号来协商加密/解密和散列使用的算法，接收方怎么能知道发送方为各种目的选择的算法呢？

为了解决这个问题，协议为每一种操作定义了一个算法集，用户可以在他/她的系统中使用这个算法集。Alice 在邮件中包含她使用算法的名字（或标识符）。例如，Alice 可以选择 DES 作为加密/解密算法，MD5 作为散列算法。当 Alice 向 Bob 发送消息时，她在她的消息中包含 DES 和 MD5 相应的标识符。Bob 接收这个消息后首先提取这些标识符。之后他就能够知道哪种算法用于解密，哪种算法用于散列了。

在电子邮件安全中，消息的发送方需要在消息中包含算法的名字或标识符。

密码系统的秘密信息

密码算法的问题在密码秘密信息（密钥）中也同样存在。如果没有协商，那么双方怎么在他们之间建立秘密信息？今天的电子邮件安全协议要求加密/解密使用对称密钥算法，并且一次性秘密密钥随消息一起发送。Alice 可以构建一个秘密密钥并且随她发送给 Bob 的消息一起发送。为了防止秘密密钥被 Eve 截获，秘密密钥使用 Bob 的公钥加密。换言之，秘密密钥本身是加密的。

在电子邮件安全中，加密/解密使用对称密钥算法。但是，秘密密钥使用接收者的公钥加密并且随消息一起发送。

证书

在我们讨论电子邮件安全协议之前，还有一个问题需要特别注意。显然，电子邮件安全必须使用一些公钥算法。例如，我们需要加密秘密密钥或对消息签名。为了加密秘密密钥，Alice 需要 Bob 的公钥；为了验证一个签名后的消息，Bob 需要 Alice 的公钥。因此，为了发送一个小的被认证和加密的消息，需要两把公钥。Alice 怎样才能确信 Bob 的公钥？Bob 怎样才能确信 Alice 的公钥？每个电子邮件安全协议拥有不同的证明密钥的方法。

良好隐私（PGP）

这一部分讨论的第一种协议称为良好隐私（Pretty Good Privacy，PGP）。PGP 是由 Phil Zimmermann 设计的，它对电子邮件提供保密性和完整性保护，并对电子邮件进行认证。PGP 可用于构建安全的电子邮件消息。

场景

我们首先讨论 PGP 的基本思想，逐渐从简单的场景过渡到复杂的场景。我们将处理之前的消息用数据一词表示。

明文 最简单的场景是使用明文发送电子邮件，如图 10-29 所示。在这种场景下，还谈不上消息的完整性或保密性。

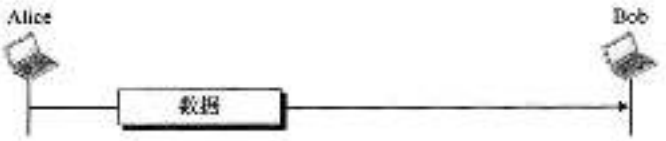


图 10-29 明文消息

消息完整性 下一个可能的改进是让 Alice 签署信息。Alice 创建一个消息摘要并且用她的私钥对其签名，如图 10-30 所示。



图 10-30 认证的消息

当 Bob 接收到消息，他使用 Alice 的公钥进行验证。这个场景需要两个密钥。Alice 需要知道她的总钥；Bob 需要知道 Alice 的公钥。

压缩 再一个改进就是压缩信息，以使报文更加紧凑。这种改进没有安全的意义，但能减轻流量。如图 10-31 所示。



图 10-31 压缩的消息

利用一次性会话密钥加密 图 10-32 显示了这种情形。正像我们以前讨论的那样，电子邮件的加密可以采用一次性会话密钥的常规密钥加密方法实现。Alice 可以创建一个会话密钥，使用这个会话密钥加密消息和摘要，并将密钥本身与消息一同发送。可是，为了保护会话密钥，Alice 使用 Bob 的公钥对其加密。

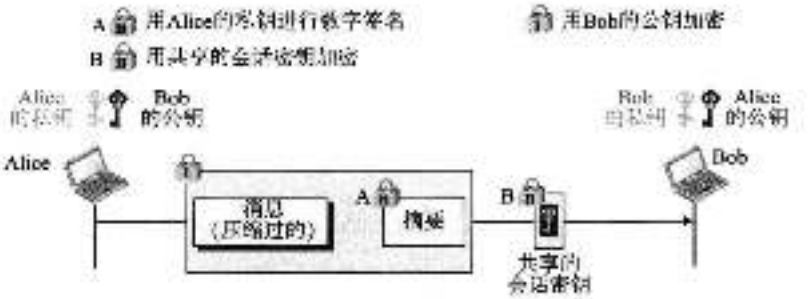


图 10-32 加密的消息

当 Bob 收到报文，他首先利用他的私钥解密会话密钥。然后他利用会话密钥解密其余的消息。在对其余的消息解压之后，Bob 生成一个消息摘要并检查它是否与 Alice 发送的摘要相等。如果是这样，那么这个消息就是可信的。

代码变换 PGP 提供的另一个服务是代码变换。多数电子邮件系统只允许消息中包含 ASCII 字符。为了变换非 ASCII 字符集中的字符，PGP 采用了基数 64 编码（见第 2 章）。

分段

在进行基数 64 编码之后，PGP 允许消息分段以使每个传输单元与电子邮件协议允许的大小一致。

密钥环

在前面所有的场景中，我们假设 Alice 只向 Bob 发送消息。这种假设不总是成立的。Alice 可能需要向很多人发送消息，她需要密钥环（key ring）。在这种情况下，Alice 需要一个公钥环，其中每个密钥属于 Alice 需要通信的（发送或接收消息的）每个人。另外，PGP 的设计者还指定了一个私钥/公钥对环。原因之一是 Alice 可能希望经常改变她的密钥对。原因之二是 Alice 可能需要与

不同组的人（朋友、同事等）进行通信。Alice 希望对每组人使用不同的密钥对。所以，每个用户需要拥有两个环：一个私钥环和一个其他人的公钥环。图 10-33 显示了一个具有 3 个人的团体，每个人拥有一个私钥/公钥对环以及一个属于团体中其他人的公钥环。



图 10-33 PGP 中的密钥环

例如，Alice 拥有多个属于她的私钥/公钥对，以及属于其他人的公钥。注意每个人可能拥有多个公钥。两种情况可能出现。

1. Alice 需要向团体中的其他人发送消息。
   a. 她使用她的私钥对摘要进行签名。
   b. 她使用接收方的公钥对新生成的会话密钥进行加密。
   c. 她使用创建的会话密钥对消息和签署的摘要进行加密。
2. Alice 接收团体中其他人的消息。
   a. 她使用她的私钥对会话密钥进行解密。
   b. 她使用会话密钥对消息和摘要进行解密。
   c. 她使用发送方的公钥对摘要进行验证。

PGP 算法

PGP 定义了一系列的对称密钥与非对称密钥算法、加密散列函数以及压缩方法。我们将这些算法的详细内容留给专门讨论 PGP 的书籍。当 Alice 向 Bob 发送电子邮件时，她指定她为实现每个目标所使用的算法。

PGP 证书和信任模型

像其他我们至今看到的协议一样，PGP 也使用证书认证公钥。但是，正像下面要解释的那样，处理过程完全不同。

PGP 证书 PGP 不需要认证中心（CA）；环中的任何人都能为环中的其他人签署证书。Bob 能够为 Ted、John、Anne 等签署证书。PGP 中没有信任层次，没有树形结构。由于层次结构的缺乏，Ted 可以拥有 Bob 签署的一个证书和 Liz 签署的另一个证书。如果 Alice 想要给 Ted 的证书路线走，那么有两条路径：一条开始于 Bob，一条开始于 Liz。一个有趣的问题是 Alice 可能完全信任 Bob 但只部分地信任 Liz。在从一个完全或部分信任中心到一个证书的连线中，可能存在多条路径。在 PGP 中，证书的发布者通常称为介绍者（*introducer*）。

在 PGP 中，从完全或部分信任中心到任何主体都可能存在多条路径。

- **信任与合法性。** PGP 的全部操作基于介绍人的信任、证书的信任和公钥合法性的认可程度。
- **介绍人信任级别。** 在缺乏中心权威机构的情况下，显然如果每个用户必须完全信任其他所有用户，那么环不可能很大。（即使在实际生活中，我们也不能完全信任我们认识的所有人。）为了解决这个问题，PGP 允许不同的信任级别。级别数大多与实现有关，但是为了简单，我们为介绍人分配 3 个信任级别：不信任、部分信任和完全信任。介绍人信任级别具体指定介绍人为环中其他人颁发证书的信任级别。例如，Alice 可能完全信任 Bob、部分地信任 Anne、完全不信任 John。PGP 没有机制决定怎样判定介绍人的信任程度，它依靠用户做出决定。
- **证书信任级别。** 当 Alice 收到一个介绍者的证书，她将其存储在主体（经认证的实体）名下。

他为这张证书分配一个信任级别。证书信任级别通常与颁发证书的介绍人级别相同。假设Alice完全信任Bob、部分信任Anne和Janette、不信任John，那么会发生如下情况。

1. Bob颁发了两张证书，一张是Linda的证书（带有公钥K1），另一张是Lesley的证书（带有公钥K2）。Alice将Linda的公钥和证书存储在Linda名下并为其分配一个完全信任级别。Alice也将Lesley的公钥和证书存储在Lesley名下并为其分配一个完全信任级别。

2. Anne为John颁发了一张证书（带有公钥K3）。Alice将这张证书和公钥存储在John的名下，但为其分配一个部分信任级别。

3. Janette颁发了两张证书，一张John的证书（带有公钥K3），另一张是Lee的证书（带有公钥K4）。Alice将John和Lee的证书分别存储在他们的名下并为他们分别分配部分信任级别。注意，John现在拥有两张证书，一张来自于Anne，另一张来自于Janette。这两张证书都具有部分信任级别。

4. John为Liz颁发了一张证书。Alice可以丢弃该证书，或者存储该证书但标注信任级别为不信任。

- **密钥合法性。**使用介绍人信任和证书信任的目的是决定一个公钥的合法性。Alice需要知道Bob、John、Liz、Anne等人的公钥是否合法。PGP定义了一个非常清晰的过程来判定密钥的合法性。一个用户密钥合法性级别是带有权重的该用户的信任级别。例如，假设我们为证书信任级别分配如下权重：

1. 为不信任的证书分配权重0。
2. 为部分信任的证书分配权重1/2。
3. 为完全信任的证书分配权重1。

之后，为了完全相信一个实体，Alice需要那个实体一张完全信任的证书或两张部分信任的证书。例如，在前面的场景中，由于Anne和Janett都为John颁发了证书，每个证书的信任级别为1/2，因此Alice可以使用John的公钥。注意属于一个实体的公钥的合法性与对那个人的信任级别没有关系。尽管Bob可以使用John的公钥向他发送消息，但是对于Alice，由于John的信任级别为不信任，因此Alice不能接受由John发布的任何证书。

PGP中的信任模型 正像Zimmermann提出的那样，我们可以为环中的任意用户创建一个信任模型，该模型以用户作为活动的中心。这样的模型看上去如图10-34所示。该图显示了某一时刻为Alice创建的信任模型。

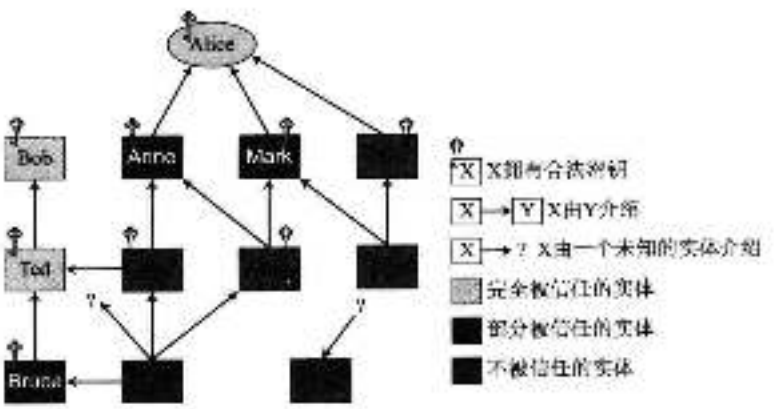


图10-34 信任模型

现在我们对图进行详细说明。在图10-34中，Alice的环中有3个实体为完全信任（Alice自己、Bob和Ted），3个实体为部分信任（Anne、Mark和Bruce），6个实体为不信任。图10-34还显示，9个实体拥有合法的密钥。Alice可以对这些实体的信息进行加密或对来自这些实体的签名进行验

证（在这种模型中，不使用 Alice 的密钥）。另外，对 Alice 来说，还有 3 个实体没有合法的密钥。

通过邮件发送密钥并且通过电话验证这些密钥的指纹，Bob、Anne 和 Mark 使他们的密钥变成了合法的密钥。在另一方面，因为 Helen 不被 Alice 信任并且无法通过电话验证，因此她发送了一张来自 CA 的证书。尽管 Ted 被完全信任，但是他送给了 Alice 一张由 Bob 签署的证书。John 发送给 Alice 两张证书，一张由 Ted 签署，另一张由 Anne 签署。Kevin 向 Alice 发送了两张证书，一张由 Anne 签署，另一张由 Mark 签署。由于这两张证书都给予 Kevin 一半的合法性，因此 Kevin 的密钥是合法的。

Duc 向 Alice 发送了两张证书，一张由 Mark 签署，另一张由 Helen 签署。因为 Mark 为半信任，Helen 为不信任，所以 Duc 没有合法的密钥。Jenny 发送了 4 张证书，一张由半信任的实体签署，两张由不信任的实体签署，一张由未知的实体签署。Jenny 没能获得足够的点数支持她密钥的合法性。Luise 发送了一张由未知实体签署的证书。注意 Alice 可以在表中保存 Luise 的名字以备将来 Luise 证书的到来。

- **信任网络**。PGP 能够在一组人之间形成一个信任网络（web of trust）。如果每个实体向其他实体介绍更多的实体，那么每个实体的公钥环变得越来越大。同时，环中的实体就可以相互发送安全的电子邮件。
- **密钥撤销**。一个实体可能需要从环中撤销他或她的公钥。如果密钥的拥有者感觉该密钥已不安全（例如被偷），或者感觉密钥太老以至于不安全，那么这种情况就可能发生。为了撤销证书，拥有者可以发送一个由她自己签署的撤销证书。撤销证书必须由旧密钥签名并分发给环中使用该公钥的所有人。

PGP 分组

在 PGP 中，一个消息包含一个或多个分组。随着 PGP 的演化，分组类型的格式和数目也有一些变化。这里我们不再讨论这些分组的格式。

PGP 应用

PGP 已经在个人电子邮件中大量使用，并可能继续维持这种状态。

S/MIME

另一个为电子邮件设计的服务是安全多用途因特网邮件扩展（Secure/Multipurpose Internet Mail Extension，S/MIME）协议。该协议是第 2 章讨论的多用途因特网邮件扩展（MIME）协议的扩展。

安全消息语法（CMS）

为了定义怎样在 MIME 内容类型中增加安全服务（如保密性或完整性服务），S/MIME 定义了安全消息语法（Cryptographic Message Syntax，CMS）。语法在各种情况下为每个内容类型定义精确的编码方案。下面讨论消息的类型和由这些消息生成的子类型。至于详细内容，读者可以参阅 RFC 3369 和 RFC 3370。

Data 内容类型 这是一个任意字符串。创建的对象称为 Data（数据）。

Singned-Data 内容类型 这种类型只提供数据的完整性。它包含任意数据类型加上 0 个或多个签名值。编码后的结果是一个称为 signedData（被签名数据）的对象。图 10-35 显示了创建这种类型对象的过程。处理步骤如下：

1. 每个签署者利用他自己选择的特定散列算法对内容生成一个消息摘要。
2. 利用签署者的私钥对每个消息摘要签名。
3. 然后，将内容、签名值、证书和算法收集起来，形成 signedData 对象。

注意，在这种情况下，内容不必是一个私有的消息。它可以是一个完整性需要保护的文档。发送者可以收集这些签名并将它们与消息一起发送（或存储）。

Enveloped-Data 内容类型 这种类型用于提供消息的保密性。它包含任意消息类型加上 0 个或多个被加密的密钥和证书。编码后的结果是一个称为 envelopedData（被封装的数据）的对象。图 10-36 显示了创建这种类型对象的过程。

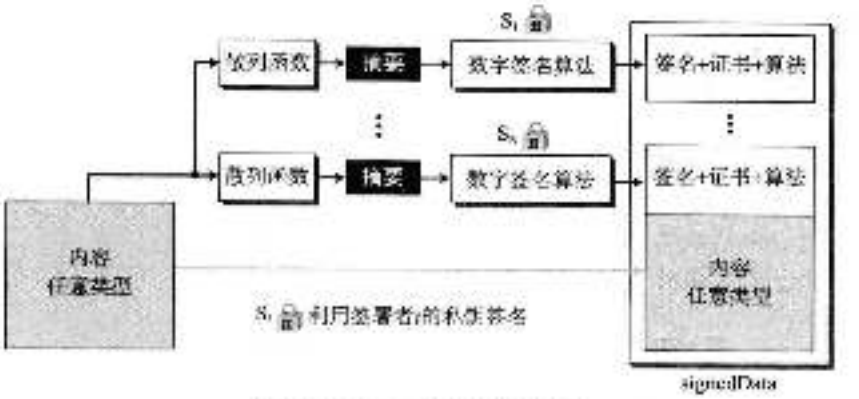


图 10-35 signed-Data 内容类型

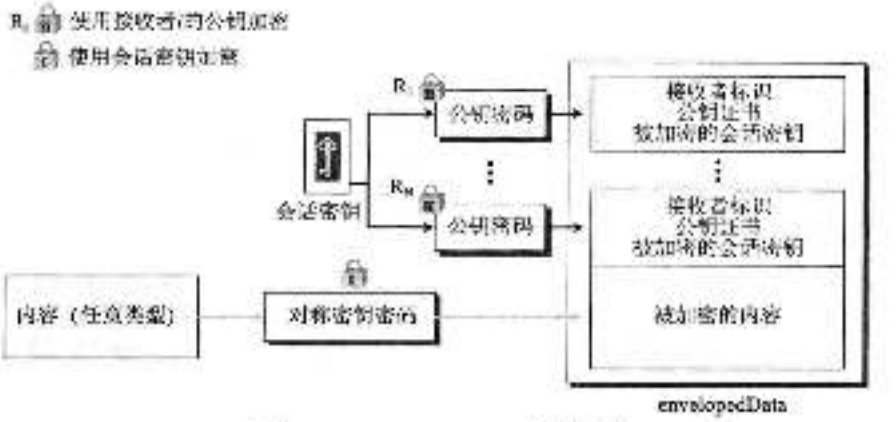


图 10-36 enveloped-Data 内容类型

1. 为了使用对称密钥算法，生成一个伪随机的会话密钥。
2. 对于每个接收者，利用那个接收者的公钥加密一个会话密钥的副本。
3. 利用指定的算法和生成的会话密钥对内容进行加密。
4. 使用基数 64 编码对被加密的内容、被加密的会话密钥、使用的算法和证书进行编码。

注意，在这种情况下，我们可以有一个或多个接收者。

Digested-Data 内容类型 这种类型用于提供消息的完整性，其结果通常作为 enveloped-data 内容类型的内容。编码后的结果是一个被称为 digestedData（摘要后的数据）的对象。图 10-37 显示了创建这种对象类型的过程。



图 10-37 Digested-data 内容类型

1. 从内容计算消息摘要。
2. 由消息摘要、算法和内容共同构成 digestedData 对象。

Encrypted-Data 内容类型　这种类型用于创建任意内容类型的加密版本。尽管这看上去像 enveloped-data 内容类型，但是 encrypted-data 内容类型没有接收者。它用于存储加密的数据而不是传送它。处理过程非常简单：用户使用任意密钥（通常源自于密码）和任意算法对内容进行加密。存储被加密的内容不包括密钥或算法。创建的对象称为 encryptedData。

Authenticated-Data 内容类型　这种类型用于提供数据的认证，其对象称为 authenticatedData。其处理过程如图 10-38 所示。

1. 利用一个伪随机数生成器，对每个接收者生成一个 MAC 密钥。
2. 利用接收者的公钥对 MAC 密钥进行加密。
3. 为内容创建一个 MAC。
4. 内容，MAC，算法和其他信息一起构成 authenticatedData 对象。

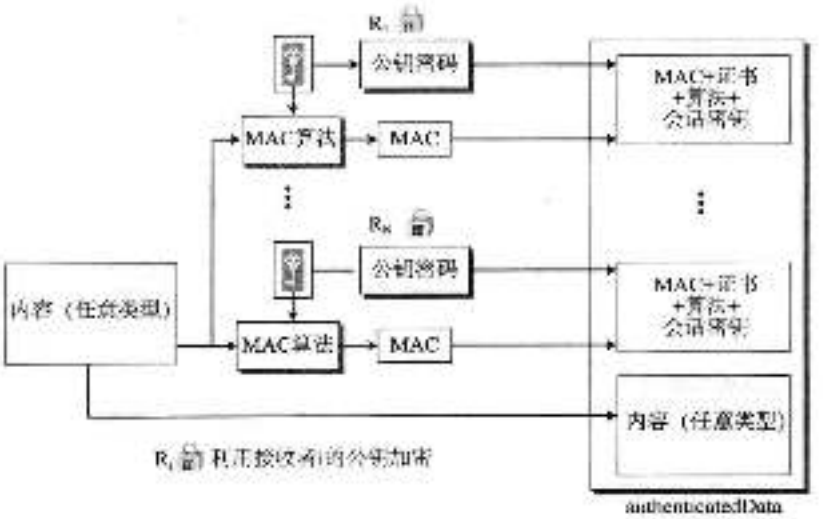


图 10-38　authenticated-data 内容类型

密钥管理

S/MIME 中的密钥管理方法是 X.509 密钥管理方法和 PGP 密钥管理方法的结合。S/MIME 使用 X.509 定义的认证中心签署的公钥证书。但是用户负责维护信任网络，以便像 PGP 一样验证签名。

安全算法

S/MIME 定义了几种安全算法。我们将这些算法的详细内容留给专门讨论互联网安全的书籍。

例 10.10　下面显示了一个 enveloped-data 的例子，其中利用 3DES 对一个小信息进行了加密。

```
Content-Type: application/pkcs7-mime; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
name="report.txt";
ck52u6764b6jHL21c487ey5i0287hmcklsgI3o5Y8c65SGh0Ai1b5+3mjc6eH23VjBrmN
ybedkzjhgfdyIcCe7JKjk34XiuD678Es16sd9y76jRuyTMDcbmIkgIEdtuyu6783+3m0n3rG
34un12P2454Hoi87e2ry80H2MjN6KuyrsqFDoY5975923jljk1301XiuD6yh78EsUyT23y
```

S/MIME 应用

预计 S/MIME 将变成工业界的选择，用于对商业电子邮件提供安全性。

10.4.2　传输层安全

今天，在传输层提供安全的协议主要有两个：**安全套接层（Secure Sockets Layer，SSL）协议**和**传输层安全（Transport Layer Security，TLS）协议**。后者实际上就是前者的 IETF 版本。我们在这一部分讨论 SSL；TLS 非常相似。图 10-39 显示了 SSL 和 TLS 在 Internet 模型中的位置。

这些协议的目标之一是提供服务器和客户之间的认证、数据机密性和数据完整性。利用 TCP 服务的应用层的客户/服务器程序，如 HHTP（见第 2 章），可以将它们的数据封装在 SSL 报文（HTTPS）中。如果服务器和客户机具有运行 SSL（或 TLS）程序的能力，那么客户机可以使用 URL https://...代替 http://...，以允许 HTTP 消息封装在 SSL（或 TLS）报文中。例如，对于在线购物者，信用卡号可以经 Internet 安全传输。



图 10-39 SSL 和 TLS 在 Internet 模型中的位置

SSL 结构

SSL 被设计成为应用层产生的数据提供安全和压缩服务。SSL 可以接收任何应用层协议的数据，但是典型的是 HTTP 协议。从应用层接收到的数据被 SSL 压缩（可选）、签名和加密。之后，数据被传递到一个可靠的传输层协议，如 TCP。Netscape 于 1994 年开发了 SSL 协议，版本 2 和版本 3 于 1995 年发布。在这一部分，我们讨论 SSLv3。

服务

SSL 为从应用层接收到的数据提供多种服务。

- 分段。首先，SSL 把数据划分为等于或小于 2^{14} 个字节的块。
- 压缩。客户和服务器之间协商一种无损压缩方法，同时利用该方法对数据的每个分段压缩。这种服务是可选的。
- 消息完整性。为了保护数据的完整性，SSL 使用一种密钥散列函数构建一个 MAC。
- 保密性。为了提供保密性，原始数据和 MAC 值利用对称密钥密码进行加密。
- 成帧。给被加密的有效载荷增加一个头部，然后将它传递给一个可靠的传输层协议。

密钥交换算法

为了交换一个认证的和保密的消息，客户和服务器都需要一个安全秘密集。但是，为了创建这些秘密，双方之间必须建立一个预主密码（pre-master secret）。SSL 为建立这种预主密码定义了几种密钥交换方法。

加密/解密算法

客户和服务器也需要就加密和解密算法集达成一致。

散列算法

SSL 利用散列算法提供消息的完整性（消息认证）。为了这个目的，SSL 定义了几种散列算法。

密码组

对于每个 SSL 会话，密钥交换、散列和加密算法的组合定义了一个密码组（cipher suite）。

压缩算法

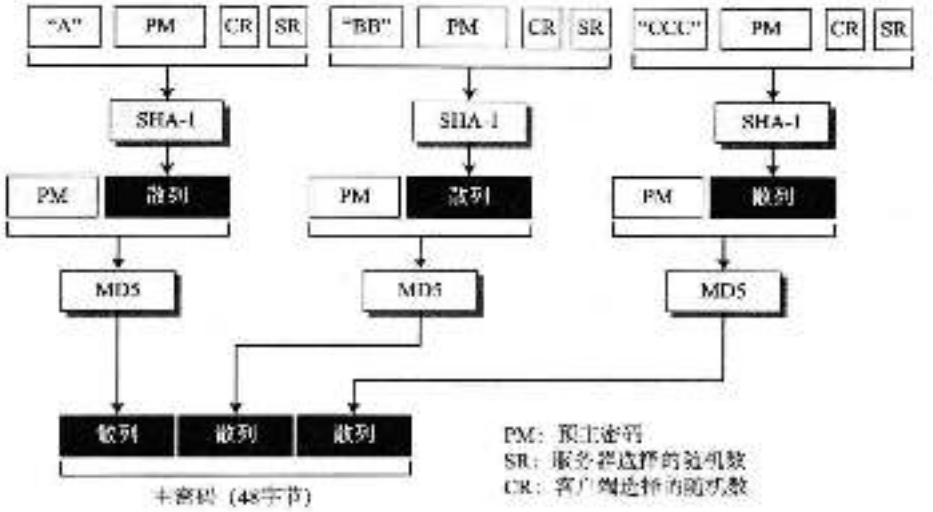
在 SSL 中，压缩是可选的，且没有定义具体的压缩算法。因此，系统可以使用它希望的任何压缩算法。

密码参数生成

为了实现消息的完整性和保密性，SSL 需要 6 个密码秘密信息：4 个密钥和两个 IV（初始化向量）。客户需要一个密钥进行消息认证，一个密钥进行加密，一个 IV 在计算中作为初始块。服务器与客户端相同。SSL 要求一个方向上的密钥要不同于另一个方向上的密钥。如果在一个方向上存在一个攻击，另一个方向不会受到影响。这些参数使用下列过程生成：

1. 客户和服务器交换两个随机数：一个由客户端生成，另一个由服务器端生成。
2. 客户和服务器利用一个预先定义的密钥交换算法交换一个预主密码。
3. 对预主密码应用两个散列函数（SHA-1 和 MD5）创建一个 48 字节的主密码，如图 10-40 所示。

4. 通过应用相同的散列函数集和增加不同的常数，主密码可用于构建可变长度的密料（key material），如图 10-41 所示。该模块不断重复直到构建出足够长度的密料。



注意："A"、"BB"和"CCC"为填充后的明文本。

图 10-40 由预主密码计算主密码

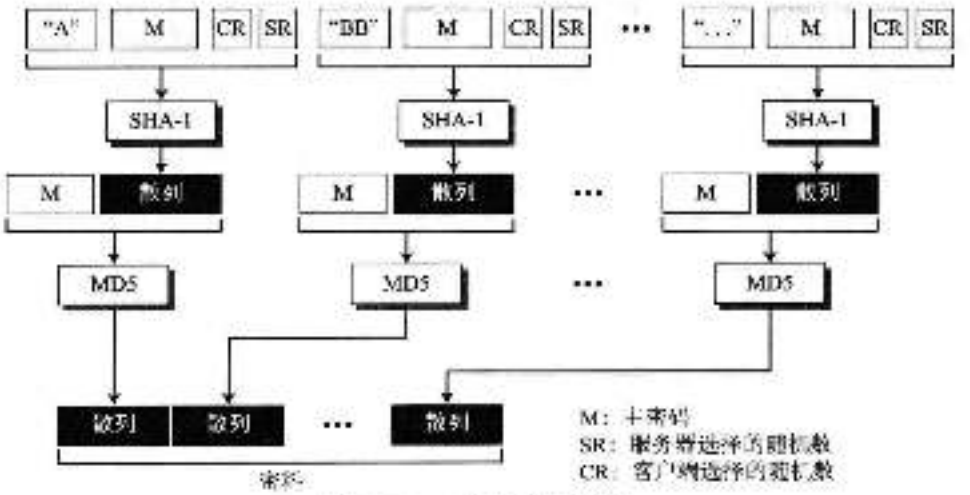


图 10-41 由主密码计算密料

注意密料块的长度与选择的密码组和这个密码组需要的密钥长度有关。

5. 从密料块取出 6 个不同的秘密信息，如图 10-42 所示。

会话和连接

SSL 中的会话和连接不同。会话是服务器和客户机之间的一个关联。在一个会话建立之后，双方就具有了公共的信息，如会话标识符、相互认证的证书（如果需要）、压缩方法（如果需要）、密码组和用于生成密钥进行认证加密的主密码。

对于两个交换数据的实体来说，会话的建立是必需的，但这并不足够；它们需要在它们自己之间创建一个连接。两个实体交换两个随机数，利用主密码创建密钥及其参数。这些密钥和参数用于对交换的消息进行认证和隐私保护。

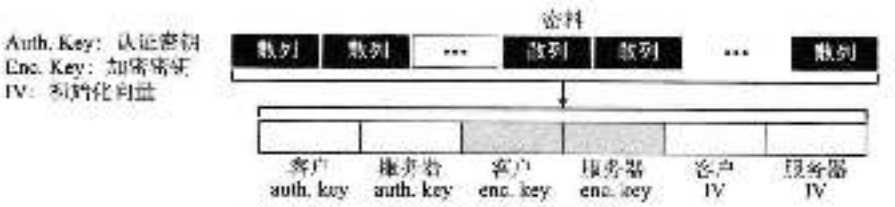


图 10-42　从密料提取密码和密信息

一个会话可能包含多个连接。双方之间的一个连接可以被终止或在同一个会话中被重新建立。当一个连接被终止时，双方也可以终止这个会话，但这不是强制性的。一个会话可以被挂起或恢复。

四个协议

我们讨论了 SSL 的思想，但并没有说明 SSL 如何完成这些任务。SSL 在两层中定义了 4 个协议，如图 10-43 所示。

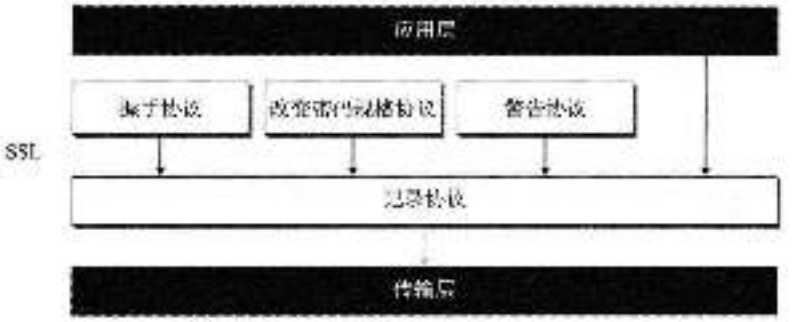


图 10-43　四个 SSL 协议

记录协议是承载者。它携带来自于其他 3 个协议的消息，也携带来自于应用层的数据。对于传输层（通常是 TCP）来说，来自记录协议的消息是它的有效载荷。握手协议为记录协议提供安全参数。它建立一个密码集，并提供密钥和安全参数。如果需要，它也实施客户机认证服务器和服务器认证客户机。改变密码规格协议用于示意安全密码信息准备就绪。警告协议用于报告非正常状态。在这一部分，我们将简单讨论这些协议。

握手协议

握手协议（Handshake Protocol）利用消息来协商密码组，如果需要，它也实施客户机认证服务和服务器认证客户机。同时，它也交换一些信息以构建安全密码。握手协议分为 4 个阶段，如图 10-44 所示。

第 I 阶段：建立安全能力　在第 I 阶段，客户机和服务器通告它们的安全能力并选出两者都支持的安全方法。这个阶段建立会话 ID 并选择密码组。同时双方协商一个特定的压缩方法。最后，客户机和服务器各选择一个随机数，用于生成我们前面谈到的主密码。在阶段 I 之后，客户机和服务器知道了 SSL 的版本、安全算法、压缩算法和密钥生成使用的两个随机数。

第 II 阶段：服务器认证和密钥交换　在第 II 阶段，如果需要，服务器则认证它自己。发送方可以发送它的证书、它的公钥，并从客户机请求证书。在第 II 阶段之后，客户机认证了服务器。如果需要，客户机能够知道服务器的公钥。

第III阶段：客户机认证和密钥交换　第III阶段被设计成认证客户机。在第III阶段之后，服务器认证了客户机，客户机和服务器两者都知道了预主密码。

第IV阶段：终止和结束　在第IV阶段，客户和服务器发送消息，以改变密码规格并完成握手协议。



图 10-44 握手协议

改变密码规格协议

我们已经看到，握手协议进行了密码组协商并逐渐生成了安全密码。现在的问题是：双方什么时候开始使用这些参数和密码？SSL 规定，直到发送和收到改变密码规格（ChangeCipherSpec）这个特殊的消息，双方才能使用这些参数或密码。改变密码规格这个消息在握手协议中进行交换，在改变密码规格协议（ChangeCipherSpec Protocol）中进行定义。其原因是这个问题不只是发送和接收一个消息。发送方和接收方需要两个状态而不是一个。状态之一，挂起态，跟踪参数和密码。状态之二，激活态，利用记录协议使用的参数和密码签署/验证或加密/解密信息。另外，每个状态拥有两个数值集合：读（入）和写（出）。

警告协议

SSL 使用警告协议（Alert Protocol）报告错误和非正常情况。它只使用一个消息描述问题和问题的级别（警告或致命）。

记录协议

记录协议（Record Protocol）承载来自于上层（握手协议、改变密码规格协议、警告协议或应用层）的消息。该消息被分段和有选择地压缩，并将利用协商好的散列算法生成的 MAC 添加到压缩信息。之后，利用协商好的加密算法对压缩的分段和 MAC 进行加密。最后，将 SSL 头部添加到加密后的消息前。图 10-45 显示了发送方的处理过程，接收方的处理过程与此相反。

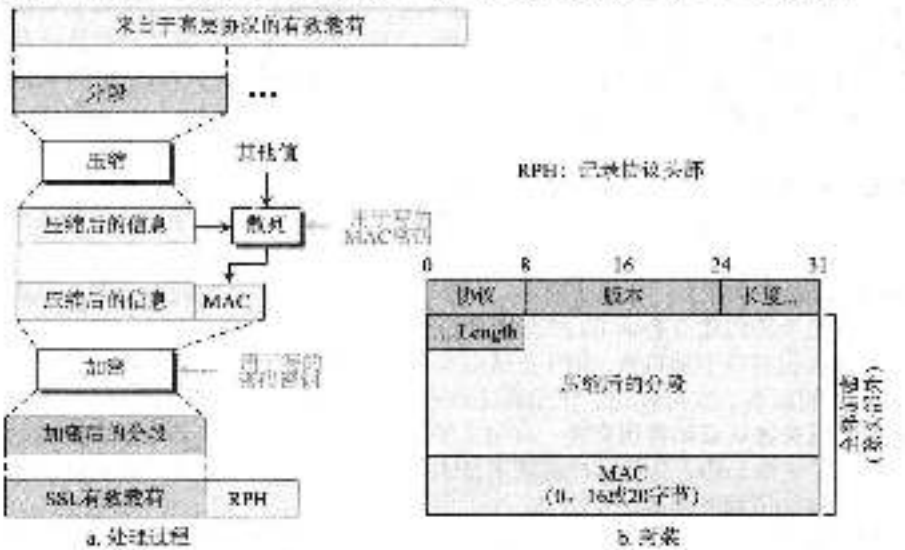


图 10-45 记录协议的处理过程

10.4.3 网络层安全

我们在这一部分开始讨论网络层安全。尽管我们在前面讨论了应用层和传输层的安全，但是，

有三种理由表明，我们也需要网络层的安全。第一，不是所有的客户/服务器程序都在应用层进行防护。第二，不是所有的应用层客户/服务器程序都使用 TCP 服务，以至于能够被我们前面讨论的传输层安全所保护；有些程序使用了 UDP 服务。第三，路由协议等很多应用程序直接使用 IP 服务；它们需要 IP 层的安全服务。

IP 层安全（IP Security，IPSec）是一个由 Internet 工程任务组（IETF）设计的一组协议，用来为网络层分组提供安全。IPSec 帮助生成经过认证和加密的 IP 层分组。

两种模式

IPSec 运行两种不同的模式之一：传输模式和隧道模式。

传输模式

在**传输模式（transport mode）**中，IPSec 保护传输层传递到网络层的内容。换言之，传输模式保护被封装在网络层中的有效载荷，如图 10-46 所示。



图 10-46　IPSec 的传输模式

注意，传输模式不保护 IP 头部。也就是说，传输模式不保护整个 IP 分组；它只保护从传输层传下来的内容（IP 层的有效载荷）。在这种模式中，需要在来自传输层的内容上添加 IPSec 头部（和尾部）。之后，再添加 IP 头部。

IPSec 的传输模式不保护 IP 头部；
它只保护来自于传输层的有效载荷。

传输模式通常在需要进行主机到主机（端到端）数据保护时使用。发送主机利用 IPSec 认证和/或加密来自于传输层的有效载荷。接收主机利用 IPSec 验证认证和/或解密 IP 分组，并将它投递给传输层。图 10-47 显示了这种概念。



图 10-47　传输模式

隧道模式

在**隧道模式（tunnel mode）**中，IPSec 保护整个 IP 分组。它处理包括头部在内的 IP 分组，并对整个分组实施 IPSec 安全方法，之后再增加一个新的 IP 头部，如图 10-48 所示。

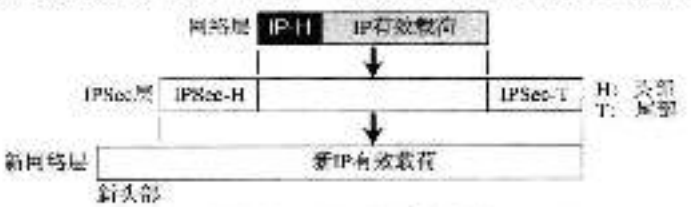


图 10-48　IPSec 的隧道模式

正像我们将看到的，这个新的 IP 头部与原有的 IP 头部不同。隧道模式通常用于两台路由器之间、一台主机和一台路由器之间或一台路由器和一台主机之间，如图 10-49 所示。在发送方和接收方之间，整个原有分组被保护以免受到入侵，仿佛整个分组穿过了一个想象的隧道。



图 10-49　隧道模式

IPSec 的隧道模式保护原有的 IP 头部。

比较

在传输模式中，IPSec 层位于传输层和网络层之间。在隧道模式中，数据流从网络层到 IPSec 层，然后再回到网络层。图 10-50 比较了这两种模式。

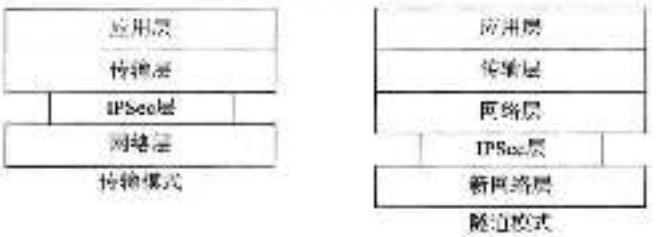


图 10-50　传输模式与隧道模式

两种安全协议

IPSec 定义了两种协议：认证报头（Authentication Header，AH）协议和封装安全载荷（Encapsulating Security Payload，ESP）协议。这两种协议用于在 IP 层对分组提供认证和/或加密。

认证报头（AH）

认证报头（Authentication Header，AH）协议用于认证源主机，并保证 IP 分组所携带的有效载荷的完整性。该协议使用一个散列函数和一个对称（秘密的）密钥生成一个消息摘要，该摘要被插入到认证报头中（见 MAC）。然后，按照使用的模式（传输或隧道），这个 AH 被放置在合适的位置上。图 10-51 显示了传输模式中认证报头的字段和位置。

当一个 IP 数据报携带一个认证报头，IP 头部协议字段的原有值被替代为 51。认证报头（下一个头部字段）的一个字段保存原有协议字段的值（IP 数据报携带的有效载荷的类型）。增加一个认证报头的步骤如下：

1. 在有效载荷前添加一个认证报头，其认证数据字段设置为 0。
2. 为了使总长度适合一个特定的散列算法，可能需要进行填充。
3. 基于整个分组进行散列运算。但是，在消息摘要生成过程中，只计算那些在传输过程中不发生改变的字段。
4. 将认证数据插入到认证报头之中。
5. 在将协议字段修改为 51 之后，将 IP 头部加入。

每个字段的简要描述如下：

- 下一个头部。8 位的下一个头部字段定义 IP 数据报携带的有效载荷的类型（如 TCP、UDP、ICMP 或 OSPF）。



图 10-51　认证报头（AH）协议

- **有效载荷长度**。这个 8 位字段的名称容易使人误解。它不是定义有效载荷的长度，而是定义认证报头的长度。该长度按照 4 字节为单位进行计数，但不包含前 8 个字节。
- **安全参数索引**。32 位的安全参数索引（security parameter index，SPI）字段扮演着虚拟电路标识符的角色。在一个称为安全关联（下面讨论）的连接里，所有发送分组使用相同的 SPI。
- **序列号**。32 位的序列号为数据报队列提供排序信息。序列号用于防止重放。注意即使一个分组被重传，序列号也不重复。当序列号达到 2^{32} 时，它也不返回来重复计数。这时必须建立一个新连接。
- **认证数据**。最后，认证数据字段是对整个 IP 数据报进行散列运算后的结果，散列计算时不包括传输中可能发生改变的字段（如生存时间）。

AH 协议提供源认证和数据完整性，但不提供机密性。

封装安全载荷（ESP）

AH 协议不提供机密性，只提供源认证和数据完整性。不久，IPSec 定义了一种可选择的称为封装安全载荷（Encapsulating Security Payload，ESP）的协议。该协议提供源认证、数据完整性和机密性。ESP 增加头部和尾部。注意 ESP 的认证数据添加在分组的尾部，这种计算方法使计算更加容易。图 10-52 显示了 ESP 的头部和尾部。

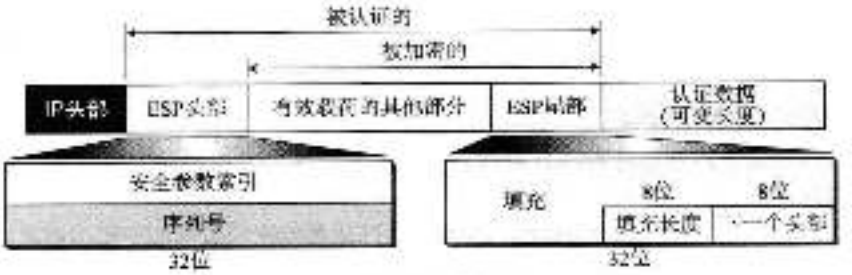


图 10-52　封装安全载荷（ESP）

当一个 IP 数据报携带 ESP 头部和尾部时，IP 报头协议字段的值为 50。ESP 尾部的一个字段（下一个头部字段）保存协议字段的原有值（IP 数据报携带的有效载荷的类型，如 TCP 或 UDP）。ESP 的处理步骤如下：

1. 将一个 ESP 尾部添加到有效载荷中。
2. 加密有效载荷和尾部。
3. 添加 ESP 头部。

4. 利用 ESP 头部、有效载荷和 ESP 尾部创建认证数据。
5. 将认证数据添加到 ESP 尾部的后面。
6. 把协议字段的值修改为 50 后，添加 IP 头部字段。

头部和尾部字段描述如下：

- **安全参数索引**。32 位的安全参数索引字段的定义与 AH 协议类似。
- **序列号**。32 位的序列号字段的定义与 AH 协议类似。
- **填充**。可变长的全 0 字段（0 – 255 字节）作为填充使用。
- **填充长度**。8 位的填充长度字段定义填充的字节数。该值在 0 到 255 之间，最大值很少出现。
- **下一个头部**。下一个头部字段的定义与 AH 协议类似。它的作用与封装前 IP 头部协议字段的作用相同。
- **认证数据**。最后，认证数据字段是对部分数据报实施认证方案的结果。注意，AH 和 ESP 认证数据不同。在 AH 中，IP 头部被包含在认证数据的计算之中。但是在 ESP 中，不包含 IP 头部。

IPv4 和 IPv6

IPv4 和 IPv6 都支持 IPSec。可是在 IPv6 中，AH 和 ESP 为扩展头部的一部分。

AH 与 ESP

ESP 协议是在 AH 协议已经使用之后设计的。ESP 除了能提供 AH 所提供的功能外，还能提供额外的功能（机密性）。我们实际上不需要 AH。可是，AH 的实现已经包含在一些商业产品中。这意味着直到淘汰这些产品后，AH 才将退出 Internet。

IPSec 提供的服务

AH 和 ESP 这两种协议能够在网络层为分组提供多种安全服务。表 10-1 显示了每种协议可以提供服务的列表。

表 10-1 IPSec 服务

服 务	AH	ESP
访问控制	是	是
消息认证（消息完整性）	是	是
实体认证（数据源认证）	是	是
保密性	否	是
重放攻击保护	是	是

访问控制

利用我们在下一部分将要看到的安全关联数据库（SAD），IPSec 非直接地提供访问控制。当一个分组到达目的地，并且没有为该分组建立安全关联，这个分组将被抛弃。

消息完整性

AH 和 ESP 都可以保护消息的完整性。由发送方创建和发送的数据摘要可以由接收方进行验证。

实体认证

在 AH 和 ESP 中，安全关联和由发送方发送的密钥散列数据摘要可以对数据报的发送方进行认证。

保密性

ESP 通过对消息加密提供保密性。但是，AH 不提供保密性。如果需要保密，那么应该使用 ESP 而不是 AH。

重放攻击防护

两个协议都利用序列号和一个滑动的接收窗口对重放攻击进行防护。在安全关联建立时，每个 IP 头部都包含了一个唯一的序列号。该序列号从 0 开始增加，直到 $2^{32}-1$。当序列号达到最大值后，

它被取置为 0。与此同时，删除旧的安全关联（见下一部分），建立一个新的安全关联。为了防止处理重复的分组，IPSec 在接收端强制使用固定大小的窗口。窗口的大小由接收端决定，其默认值为 64。

安全关联

安全关联是 IPSec 非常重要的特征。IPSec 要求两台主机之间建立一种称为安全关联（Security Association，SA）的逻辑关系。这一部分首先讨论安全关联的基本思想，然后介绍 IPSec 怎样使用安全关联。

安全关联的思想

安全关联就是双方之间的一个协议，该关联创建一个双方之间的通道。我们假设 Alice 需要与 Bob 进行单向通信。如果 Alice 和 Bob 只对安全中的保密性感兴趣，那么他们可以在他们之间形成一个共享的秘密密钥。我们可以说在 Alice 和 Bob 之间存在两个 SA：一个出 SA，一个入 SA。它们每个都将密钥的值存储在一个变量中，加密/解密算法的名字存储在另一个变量中。Alice 使用该算法和密钥加密发给 Bob 的消息；当 Bob 需要解密从 Alice 那里接收的消息时，它也使用同样的算法和密钥。图 10-53 显示了一个简单的 SA。

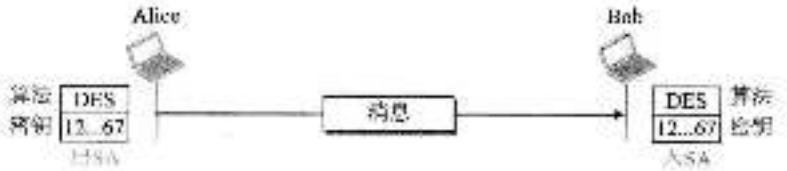


图 10-53 简单的 SA

如果双方需要消息的完整性和进行认证，那么安全关联可以更多。每个关联需要另外的一些数据，如消息完整性的算法、密钥和另外的参数。如果参与的各方需要对不同的协议（如 IPSec AH 或 IPSec ESP）使用特定的算法和特定的参数，那么安全关联可能更加复杂。

安全关联数据库（SAD）

一个安全关联可能非常复杂。当 Alice 希望向多个人发送消息，而 Bob 需要接收多个人的消息时更是如此。另外，为了允许双向的通信，每端既需要入 SA 也需要出 SA。也就是说，我们需要一个 SA 集合。这些集合可以形成一个数据库，这个数据库就称为**安全关联数据库**（Security Association Database，SAD）。数据库可以看做一个两维的表格，其中每一行定义一个单一的 SA。一般情况下存在两个 SA，一个入 SA 和一个出 SA。图 10-54 显示了一个实体的入和出 SAD 的概念。

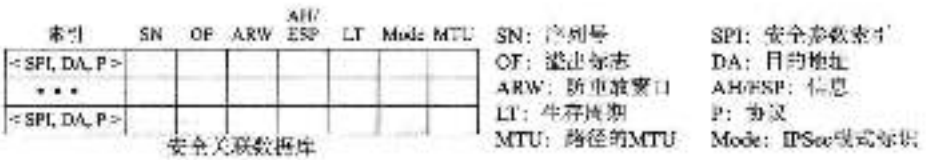


图 10-54 SAD

当一台主机需要发送一个必须携带 IPSec 头部的分组时，主机需要在出 SAD 中找到相应的条目，以便获得信息对分组实施安全策略。类似地，当一台主机接收到一个携带 IPSec 头部的分组时，主机需要在入 SAD 中找到相应的条目，以获得信息验证分组的安全性。由于接收主机需要确保使用正确的信息处理分组，因此这种搜索必须是具体的。在一个入 SAD 中，每个条目使用一个三元索引进行选择：安全参数索引（定义目的 SA 的一个 32 位数字）、目的地址和协议（AH 或 ESP）。

安全策略

IPSec 的另一个重要特征是**安全策略**（Security Policy，SP），它定义了分组发送或到达时采用的安全类型。在使用前一部分讨论的 SAD 之前，一台主机必须确定这个分组预先定义的策略。

安全策略数据库

采用 IPSec 协议的每台主机需要保存一个安全策略数据库（Security Policy Database，SPD）。同样，存在入 SPD 和出 SPD 两种 SPD。SPD 的每一项可以利用一个六元索引进行访问：源地址、目的地址、名字、协议、源端口和目的端口，如图 10-55 所示。

索引	策略
< SA, DA, Name, P, SPort, DPort >	
• • •	
< SA, DA, Name, P, SPort, DPort >	

SA：源地址　　SPort：源端口
DA：目的地址　　DPort：目的端口
P：协议

图 10-55　SPD

出 SPD　当发送一个分组时，需要查看出 SPD。图 10-56 显示了发送方处理一个分组的过程。出 SPD 的输入是一个六元组，输出为下列三种情况之一：丢弃（分组不能被发送）、绕过（绕过安全头部）、应用（按照 SAD 应用安全策略；如没有 SAD 则创建一个）。

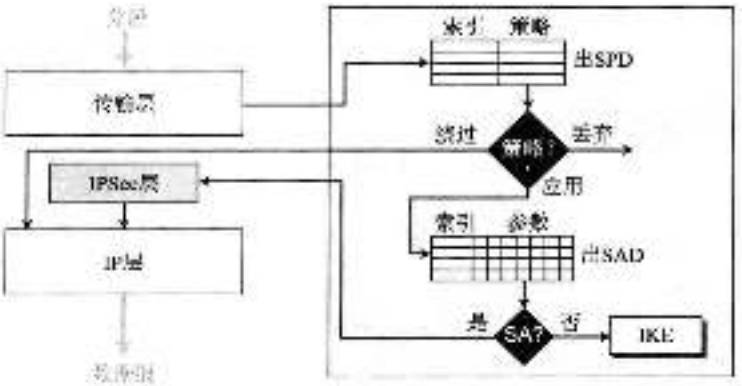


图 10-56　出 SPD 处理

入 SPD　当一个分组到达时，需要查看入 SPD。入 SPD 的每一项也利用同样的六元组索引进行访问。图 10-57 显示了接收方处理一个分组的过程。

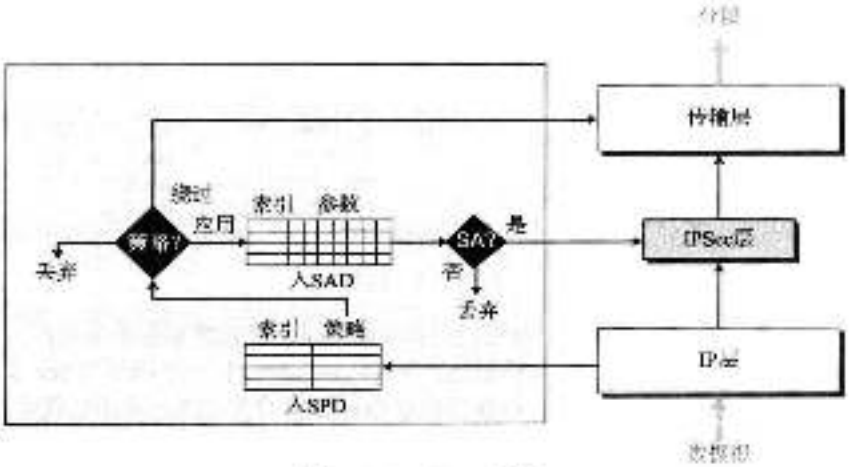


图 10-57　入 SPD 处理

入 SPD 的输入是一个六元组索引，输出为下列三种情况之一：丢弃（抛弃分组）、绕过（绕过安全策略，将分组投递到传输层）、应用（按照 SAD 实施安全策略）。

因特网密钥交换（IKE）

因特网密钥交换（Internet Key Exchange，IKE）用于创建人和出安全关联。正像我们前面讨论的那样，当一方需要发送一个 IP 分组时，它需要通过安全策略数据库（SPD）查看是否存在那种流量类型的 SA。如果没有 SA，则调用 IKE 创建一个。

IKE 为 IPSec 创建 SA。

IKE 是一个复杂的协议，它基于其他二个协议：Oakley、SKEME 和 ISAKMP，如图 10-58 所示。Oakley 是由 Hilarie Orman 设计的一个密钥生成协议。而 SKEME 由 Hugo Krawcyzk 设计，是另一种密钥交换协议。密钥交换协议使用公钥加密对实体进行认证。

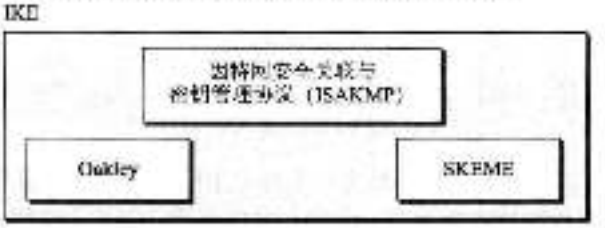


图 10-58 IKE 的组成单元

因特网安全关联与密钥管理协议（Internet Security Association and Key Management Protocol，ISAKMP）是由国家安全局（NSA）设计的一个协议，它具体实现 IKE 中定义的交换。它规定了几种分组、协议和参数，以允许 IKE 交换标准的、格式化的消息来创建 SA。我们将这三个协议讨论留给专门讨论安全的书籍。

虚拟专用网（VPN）

IPSec 的一种应用是用于虚拟专用网。虚拟专用网（virtual private network，VPN）这种技术在大型组织中应用非常广泛。这些大型组织使用个球 Internet 进行组织内部和组织之间的通信，但需要保护组织内部通信的隐私。VPN 是一个专用网络，但这种专用是虚拟的。它是专用的，因为它能够保证组织内部的隐私。它是虚拟的，因为它不使用真正的专用 WAN，网络物理上是公用的但实际上是专用的。图 10-59 给出了虚拟专用网的思想。路由器 R1 和 R2 采用 VPN 技术保证组织的隐私。VPN 技术使用 IPSec 隧道模式的 ESP 协议。一个私有数据报（包括头部）被封装在一个 ESP 分组中。发送方边界路由器在新数据报中使用它自己的 IP 地址和目的端路由器的地址。公共网络（Internet）负责将分组从 R1 传递到 R2。局外人不能解码分组的内容或源和目的地址。解码发生在 R2，它找到分组的目的地址并进行投递。

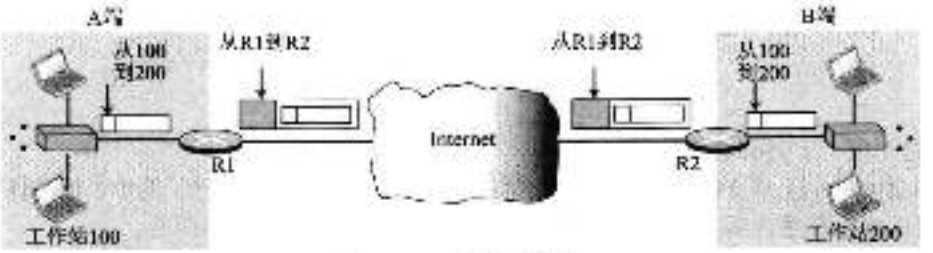


图 10-59 虚拟专用网

10.5 防火墙

前面介绍的所有安全方法不能阻止 Eve 向系统中发送有害的消息。我们需要使用防火墙进行系统访问控制。防火墙（firewall）是一种安装在组织机构内部网络和其他 Internet 网络之间的一种设备（通常是路由器或计算机），用于转发一些分组而过滤掉（不转发）另一些分组。图 10-60 给

出了一个防火墙的示意图。

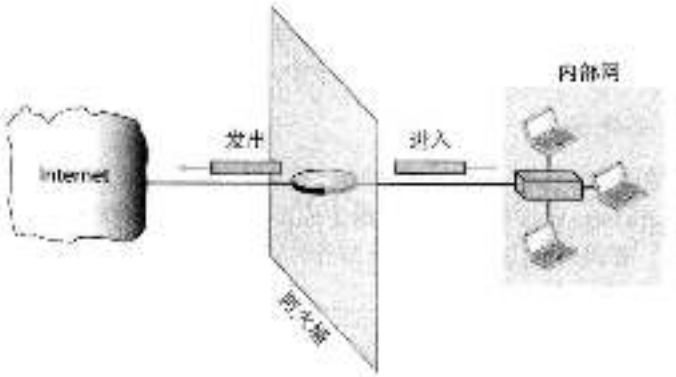
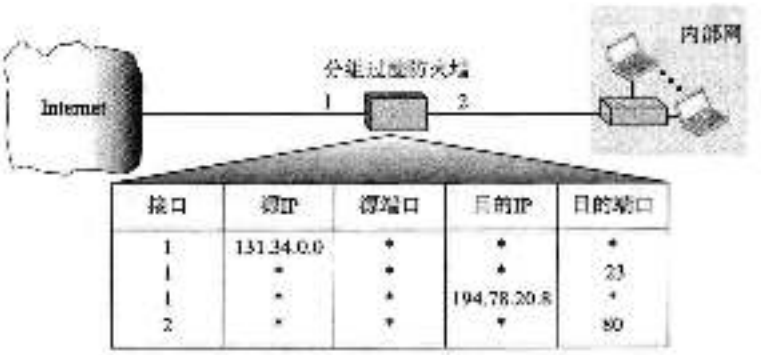


图 10-60　防火墙

例如，一个防火墙可以过滤所有目的地为一个特定主机或一个特定服务器（如 HTTP）的分组。一个防火墙可以用来在组织机构中拒绝对一个特定主机或特定服务的访问。防火墙通常可以分为分组过滤分火墙（packet-filter firewall）或基于代理的防火墙（proxy-based firewall）。

10.5.1　分组过滤防火墙

防火墙可以用作分组过滤器。它可以基于网络层或传输层头部的信息转发或阻止分组。这些头部信息可以为源和目的 IP 地址、源和目的端口地址、协议类型（TCP 或 UDP）。一个分组过滤防火墙就是一个利用过滤表决定哪些分组必须丢弃（不转发）的路由器。图 10-61 显示了这类防火墙的一个过滤表。



接口	源IP	源端口	目的IP	目的端口
1	131.34.0.0	*	*	*
1	*	*	*	23
1	*	*	194.78.20.8	*
2	*	*	*	80

图 10-61　分组过滤防火墙

按照这个图，下列分组将被过滤：

1. 源自网络 131.34.0.0 的进入分组将被阻止（安全预警）。注意，*（星号）指“任何一个”。
2. 目的地为任意内部 TELNET 服务器（端口 23）的进入分组将被阻止。
3. 目的地为内部 194.78.20.8 的主机将被阻止。组织机构希望该主机仅供内部使用。
4. 目的地为 HTTP 服务器（端口 80）的发出分组将被阻止。该组织机构不希望员工浏览 Internet。

分组过滤防火墙在网络层和传输层进行过滤。

10.5.2　代理防火墙

分组过滤防火墙基于网络层和传输层头部（IP 和 TCP/UPD）的可用信息进行过滤工作。但是，

有时我们需要基于消息本身（处于应用层）的可用信息进行过滤。例如，假设一个组织机构希望实现下列与 Web 页面相关的策略：已经和公司建立商务关系的那些 Internet 用户可以拥有访问权，同时阻止其他用户的访问。在这种情况下，因为分组过滤防火墙不能区分到达 TCP 80 端口（HTTP）分组之间的不同，所以应用分组过滤防火墙是不可行的。检查必须在应用层（利用 URL）完成。

一种解决方案是安装一个代理计算机（有时称为应用网关，application gateway），它位于客户计算机和公司计算机之间。当用户的客户进程发送一个消息时，应用网关运行一个服务器进程接收请求。服务器在应用层打开分组并查看该分组是否合法。如果合法，那么服务器扮演一个客户进程并向公司真正的服务器发送消息。如果不合法，丢弃该消息并向外部用户发送一个错误信息。通过这种方式，外部用户的请求在应用层进行了基于内容的过滤。图 10-62 显示了应用层网关，该网关实现 HTTP 协议。

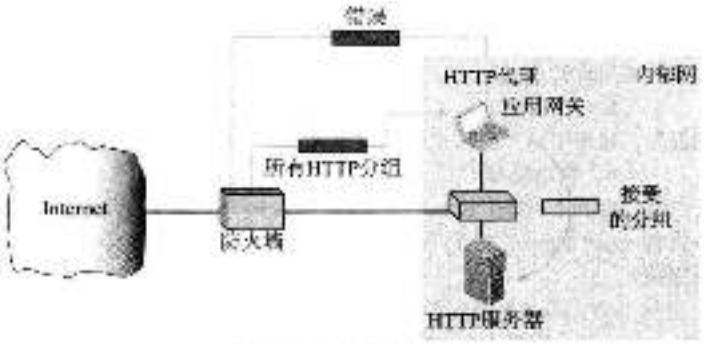


图 10-62 代理防火墙

代理防火墙在应用层进行过滤。

10.6 章末资料

推荐读物

覆盖密码学和网络安全的一些书籍包括：[For 08]、[Sta 06]、[Bis 05]、[Mao 04]、[Sti 06]、[Res 01]、[Tho 00]、[DH 03]和[Gar 95]。

小结

安全攻击可能威胁安全的三个目标。人们使用两种机制保护信息免受攻击：加密系统和信息隐藏。

在对称密钥密码中，加密和解密使用相同的密钥，同时该密钥用于双向通信。我们可以将传统的对称密钥密码分成两大类：替换密码和换位密码。

非对称密钥密码系统中存在两个分离的密钥：一个私钥和一个公钥。非对称密钥密码系统意味着 Bob 和 Alice 不能使用相同的密钥集进行双向通信。

安全的其他方面包括完整性、消息认证、实体认证和密钥管理。

由 Phil Zimmermann 提出的良好隐私（PGP）协议提供电子邮件的保密性、完整性，同时能对其进行认证。为电子邮件设计的另一种安全服务为安全多用途因特网邮件扩展（S/MIME）。

传输层安全协议为应用层提供端到端的安全服务，该应用层使用可靠的传输层服务（如 TCP）。目前，在传输层提供安全的协议主要有两个：安全套接层（SSL）协议和传输层安全（TLS）协议。

IP 层安全（IPSec）协议是由 IETF 设计的一个协议集，用于在网络层为分组提供安全。IPSec 操作模式包括传输模式和隧道模式。IPSec 定义了两种协议：认证报头（AH）协议和封装安全载荷（ESP）协议。

防火墙是安装在组织机构内部网络与其他 Internet 网络之间的一种设备（通常是路由器或计算机）。它能够转发一些分组，过滤另一些分组。防火墙通常分为分组过滤防火墙或代理防火墙。

10.7 习题集

测试题

本章的交互式测试题请参见这本书的网站。在进行其他练习之前，强烈建议学生完成这些测试题以检查对这些内容的理解程度。

练习题

Q10-1 下列哪一种攻击是对保密性的攻击？

a. 窥探　　b. 假冒　　c. 否认

Q10-2 下列哪一种攻击是对完整性的攻击？

a. 修改　　b. 重放　　c. 拒绝服务

Q10-3 下列哪种攻击是对可用性的攻击？

a. 否认　　b. 拒绝服务　　c. 修改

Q10-4 下列哪个词意思为“秘密书写”？哪个词为“隐蔽书写”？

a. 密码系统　　b. 信息隐藏

Q10-5 一封封装起来的信件由 Alice 发往 Bob，这里对信息保密采用的是密码系统还是信息隐藏？

Q10-6 当一封由 Bob 发往 Alice 的信件采用只有两个人能够理解的语言书写，这里对信息保密采用的是密码系统还是信息隐藏？

Q10-7 Alice 找到了一种给 Bob 书写秘密信件的方法。她每次采用一个新的文本（如报纸上的一篇文章），但是在单词之间增加一个或两个空格。一个单一的空格意思为二进制数字 0；双空格表示二进制数字 1。Bob 提取这些二进制数字，利用 ASCII 码对它们进行解释。这种方法属于密码系统还是信息隐藏？请解释。

Q10-8 Alice 和 Bob 交换机密消息。他们共享一个非常大的数字作为两个方向上的加密和解密密钥。这种方法属于对称密钥密码还是非对称密钥密码？请解释。

Q10-9 当 Alice 加密发送给 Bob 的消息和解密来自 Bob 的消息时使用相同的密钥。这种方法属于对称密钥密码还是非对称密钥密码？请解释。

Q10-10 对替换密码和换位密码进行区分。

Q10-11 在一种密码中，明文中的所有 A 变成了密文中的 D，明文中的所有 D 变成了密文中的 H。这是一种单字母还是多字母替换密码？请解释。

Q10-12 单字母密码和多字母密码哪个更容易被攻破？

Q10-13 假设 Alice 和 Bob 使用一种模 26 运算的加性密码。如果攻击者 Eve 希望通过尝试所有的密钥破解该密码（野蛮攻击），那么她平均需要尝试多少个密钥？

Q10-14 如果说例 10.1 和例 10.2 有一个单一的整数密钥，那么例 10.3 有多少个整数密钥？

Q10-15 假如我们有一个 1000 字符的明文，那么在下列每种密码中，我们加密或解密消息需要多少个密钥？

a. 加性密码　　b. 单字母密码　　c. 自动密钥

Q10-16 按照流密码和块密码的定义，找出下列密码哪个是流密码。

a. 加性密码　　b. 单字母密码　　c. 自动密钥

Q10-17 在现代块密码中，一个排列块（P-盒）有 5 个输入和 5 个输出，这是一种____排列。

a. 直接　　b. 压缩　　c. 扩展

Q10-18 在现代块密码中，一个排列块（P-盒）就是无密钥换位密码的一个示例。这句话的意思是什么？（参考图 10-8。）

Q10-19 在现代密码中，我们经常在解密密码中使用的一个部件就是加密密码中使用的一个相反部件。下列部件的相反部件是什么？

a. 交换　　b. 向右移位　　c. 组合

Q10-20 DES 每轮中都包含了图 10-8 中定义的部件。哪些部件使用密钥？哪些不用？

Q10-21 在图 10-10 中，为什么我们需要一个扩展 P-盒？为什么我们不能使用一个直接或压缩 P-盒？

Q10-22 图10-9显示DES生成了16个不同的48位密钥，每轮使用一个。为什么我们需要16个不同的密钥？为什么我们不能在每轮中使用相同的密钥？

Q10-23 如果说一次一密密码是最简单、最安全的密码，那么为什么不总是使用它？

Q10-24 如果Alice和Bob希望利用非对称密钥密码系统进行通信，那么他们需要多少个密钥？谁需要生成这些密钥？

Q10-25 你为什么认为非对称密钥密码系统只能用于处理长度短的消息？

Q10-26 在一个非对称公钥密码中，哪个密钥用于加密？哪个密钥用于解密？

a. 公钥　　b. 私钥

Q10-27 在RSA中，为什么Bob不能选择1作为公钥e？

Q10-28 在图10-17（MAC）中，加到散列函数中的秘密密钥的作用是什么？请解释。

Q10-29 区分消息认证和实体认证的不同。

Q10-30 Alice签署她发送给Bob的消息以证明她是消息的发送者。Alice需要使用下列哪个密钥？

a. Alice的公钥　　b. Alice的私钥

Q10-31 Alice需要向拥有50个人的一个组发送一个消息。如果Alice需要使用消息认证，那么你推荐使用下列哪种方案？

a. MAC　　b. 数字签名

Q10-32 数字签名不提供下列哪种服务？

a. 消息认证　　b. 保密性　　c. 不可否认

Q10-33 假设Alice需要向100个人发送一个加密的、签名的文档。如果Alice使用非对称密钥加密方法，那么她需要使用多少个密钥来准备这100份文档？请解释。

Q10-34 在一个拥有50个会员的俱乐部中，如果允许任何一对会员之间相互交换秘密消息，那么需要多少个秘密密钥？

Q10-35 密钥分发中心（KDC）用于解决分发____密钥的问题。

a. 秘密　　b. 公开　　c. 私有

Q10-36 认证中心（CA）用于解决分发____密钥的问题。

a. 秘密　　b. 公开　　c. 私有

思考题

P10-1 指出下列各种情况下的攻击类型：

a. 一个学生闯入教授的办公室，获得了下次考试的复印件。

b. 一个学生用10美元的支票购买了一本有用的书。之后，该支票被兑现了100美元。

c. 一个学生使用虚假的邮件返回地址，每天向学校发送几百封电子邮件。

P10-2 使用$k=10$的加性密码加密明文"book"。之后，对消息进行解密，得到原始的明文。

P10-3 使用$k=20$的加性密码加密消息"this is an exercise"，忽略单词之间的空格。解密这个消息得到原始的明文。

P10-4 Atbash是一种在圣经书写者之间非常流行的密码。在Atbash中，"A"被加密为"Z"，"B"被加密为"Y"，等等。同样，"Z"被加密为"A"，"Y"被加密为"B"，等等。假设字母表被分成两部分，第一部分的字母被加密成第二部分的字母，反之亦然。请指出这种密码类型和密钥。使用Atbash密码加密明文"an exercise"。

P10-5 替换密码不都是字符到字符的变换。在棋盘密码（Polybius cipher）中，明文中的每个字母被加密成两个整数。密钥为一个5×5的字符矩阵。明文是矩阵中的字符，密文为表示行数和列数的两个整数（每个在1到5之间）。利用下列密钥和Polybius密码，加密消息"An exercise"：

	1	2	3	4	5
1	z	a	p	l	e
2	y	r	o	g	d
3	x	s	n	h	c
4	w	t	m	i/j	b
5	v	u	l	k	a

P10-6　Alice 在她的计算机上只使用加性密码向一个朋友发送消息。她认为如果将消息加密两次，每次使用不同的密钥，那么消息会更安全。她的想法正确吗？说明你的理由。

P10-7　入侵者能够对简单的密码（如加性密码）实施的一种攻击叫做密文攻击。在这种类型的攻击中，入侵者截获密文，试图找到密钥并最终获得明文。密文攻击的一种方法是*野蛮*（*brute-force*）攻击。在这种方法中，入侵者尝试多个密钥解密消息，直到消息变成可以理解的东西。假设入侵者截获的密文为"UVACLYZLJBYL"。从密钥 1 开始尝试解密这个消息，直到有意义的明文出现。

P10-8　密文攻击（见上题）使用的另一种方法叫做统计（statistical）攻击，其中入侵者截获一个长密文并尝试分析密文中字符的统计特性。简单的密码（如加性密码）由于加密是一对一的，因此不能改变字符的加密特性。假设入侵者截获了如下密文，并决定明文中最常出现的字符为字符"e"，利用该知识寻找密码的密钥并解密密文。

P10-9　在换位密码中，加密和解密密钥经常表示为两个一维的表（数组），密码表示为一段软件（一个程序）。

a. 给出图 10-6 的加密密钥数组。提示：每个元素的值可以表示输入列号；索引可以表示输出列号。

b. 给出图 10-6 的解密密钥数组。

c. 已知加密密钥，解释如何找到解密密钥。

P10-10　循环移位操作是现代块密码的组成部分之一。

a. 对一个字（10011011）$_2$进行 3 位循环左移，给出其结果。

b. 对 a 的结果进行 3 为循环右移，给出其结果。

c. 比较 b 的结果与 a 的原始字，看看右移和左移操作是否为相反操作。

P10-11　交换操作是现代块密码的组成部分之一。

a. 交换字（10011011）$_2$。

b. 交换 a 产生的结果。

c. 比较 a 和 b 的结果，看看交换操作是否为自逆操作。

P10-12　在块密码中，经常使用的操作为 XOR 操作。给出如下操作的结果并对其进行解释。

a.（01001101）⊙（01001101）

b.（01001101）⊕（00000000）

P10-13　假设你想写一个程序来模拟图 10-8 中的排列盒。

a. 给出你怎样将一个盒表示为一个表。

b. 给出表示每一个反向盒的表。

P10-14　假设我们有一个无密钥置换盒（S-盒），该盒具有 3 个输入（x_1，x_2和x_3）和两个输出（y_1和 y_2）。输入和输出的关系定义如下（⊕表示 XOR）：

$y_1=x_1 \odot x_2 \odot x_3$　　$y_2=x_1$

如果输入为（110），那么输出是多少？如果输入为（001），那么输出为多少？

P10-15　在块密码中，每轮应该是可逆的，以使使整个块是可逆的。现代密码使用两种方法实现这个目标。在第一种方法中，每个组成单元是可逆的；在第二种方法中，一些组成单元是不可逆的，但是利用 Feistel 密码使整轮是可逆的。本章介绍的 DES 中使用了这种方法。Feistel 密码的技巧是将 XOR 操作作为组成单元之一。为了看到这一点，假设一轮由一个非可逆的单元 NI 和一个 XOR 操作组成，如图 10-63 所示。证明其整轮是可逆的（即由密文能够恢复明文）。提示：利用 XOR 的特性（$x \oplus x=0$ 和 $x \oplus 0=x$）。

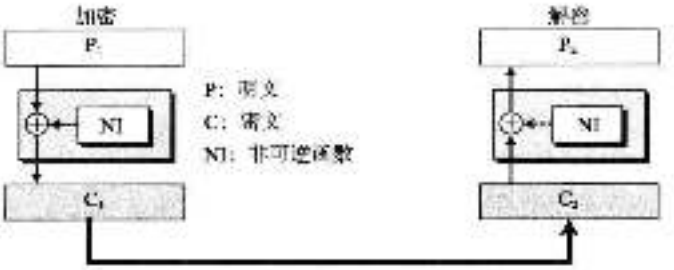


图 10-63　思考题 P10-15

P10-16　在图 10-9 中，每轮有一个交换器。这个交换器的功能是什么？

P10-17　在图 10-9 中有两个直接排列操作：初始排列（initial permutaion）和最终排列（final permutation）。专家相信这两个操作无用，不能使密码更安全。你能说说这种观点的理由吗？

P10-18　DES 的密钥为 56 位。假设入侵者 Eve 试图利用野蛮攻击（逐个尝试所有的密钥）寻找密钥。如果她（利用一个功能强大的计算机）每秒尝试一百万个密钥（约 2^{20}），那么破解密码需要多长时间？

P10-19　假设 Bob 使用 RSA 密码系统，他选择 p=11、q=13、d=7，下列哪个值可能是公钥 e？

a. 11　　b. 105　　c. 19

P10-20　在 RSA 中，已知 p=107、q=113、e=13、d=3653。利用 00 到 26（A：00，空格：26）作为编码方案，加密消息“THIS IS TOUGH”。解密密文得到原始的消息。

P10-21　加密散列函数需要具有抗二次原像攻击（*second preimage resistant*）能力。这就意味着已知消息 M 和消息摘要 d，我们不应该能够找到任何其他的消息 M′，它的消息摘要也是 d。换言之，两个不同的消息不能具有相同的摘要。基于这种需求，请说明 Internet 中传统的校验码不能作为散列函数使用。

P10-22　解释为什么不能使用私钥/公钥构建 MAC？

P10-23　图 10-22 中的瞬时值用于防止第三个消息的重放攻击。由于 Bob 接收到响应后，R_b 的值不再有效，因此 Eve 不能重放第三个消息并假装它是来自 Alice 的一个新的请求。这意味着如果我们在方案中增加一个时间戳，那么我们也可以排除第一个和第二个消息。绘出图 10-22 使用时间戳的新版本。

P10-24　解释图 10-23 的第二个消息（从 Bob 到 Alice）为什么使用加密，但是图 10-24 的第二个消息（从 Alice 到 Bob）使用签名。

P10-25　图 10-22 显示了一个为 Bob 验证 Alice 的单向认证。修改该图以提供双向认证：为 Bob 认证 Alice，为 Alice 认证 Bob。

P10-26　修改图 10-23 以允许双向认证。为 Bob 认证 Alice，为 Alice 认证 Bob。

P10-27　修改图 10-24 以允许双向认证。为 Bob 认证 Alice，为 Alice 认证 Bob。

P10-28　你可能已经注意到图 10-26 有一个缺陷。入侵者 Eve 可能重放第三个消息，并且如果她能够以某种方式获得了会话密钥，那么她能够假冒 Alice 并与 Bob 交换信息。如果 Alice 和 Bob 使用两个瞬时值，那么这个问题就可以避免。注意，瞬时值有生命周期，它们的主要目的是防止重放。修改图 10-26，增加两个瞬时值。

P10-29　假设我们有一个非常简单的消息摘要。我们这个不现实的消息摘要为 0 和 25 之间的一个数字。该摘要初始设置为 0。加密散列函数将摘要当前的值与字符当前的值（在 0 到 25 之间）相加。加法按模 26 进行。如果消息为“HELLO”，那么摘要值为多少？为什么这种摘要不安全？

P10-30　为了理解秘密密钥分发的概念，假设一个小型私人俱乐部只有 100 个会员（不包括主席）。请回答下列问题：

a. 如果俱乐部所有会员需要相互之间发送秘密消息，那么需要多少个秘密密钥？

b. 如果每个会员都信任俱乐部主席，那么需要多少个秘密密钥？如果一个会员需要向另一个会员发送消息，那么她首先将消息发送给主席，然后主席将消息发送给另一个会员。

c. 如果主席决定需要通信的两个会员应该首先与他联系，然后主席创建一个暂时的密钥用于他们两个之间的通信，该临时密钥加密后发送给这两个会员，那么需要多少个秘密密钥？

P10-31　我们为电子邮件定义了两种安全服务（PGP 和 S/MIME）。解释为什么电子邮件不能使用 SSL/TLS 服务而需要使用 PGP 或 S/MIME。

P10-32　假设 Alice 需要向 Bob 发送一封电子邮件。解释怎样利用 PGP 实现电子邮件的完整性保护。

P10-33　假设 Alice 需要向 Bob 发送一封电子邮件。解释怎样利用 PGP 实现电子邮件的机密性保护。

P10-34　假设 Alice 需要向 Bob 发送一封电子邮件。解释怎样利用 S/MIME 实现电子邮件的完整性保护。

P10-35　假设 Alice 需要向 Bob 发送一封电子邮件。解释怎样利用 S/MIME 实现对电子邮件的认证。

P10-36　假设 Alice 需要向 Bob 发送一封电子邮件。解释怎样利用 S/MIME 实现电子邮件的机密性保护。

P10-37　当我们谈到 SSL 中的认证时，我们的意思是指消息认证（message authentication）还是实体认证（entity authentication）？请解释。

P10-38　当我们谈到 PGP（或 S/MIME）中的认证时，我们的意思是指消息认证（message authentication）还是实体认证（entity authentication）？请解释。

P10-39　如果 PGP 或 S/MIME 中的密码算法不能协商，那么电子邮件的接收方怎样知道发送方使用的哪种算法？

P10-40　UDP 能够使用 SSL 吗？请解释。

P10-41　为什么 SSL 中不需要安全关联？

P10-42　比较和对比 PGP 和 S/MIME，它们各自的优势和劣势是什么？

P10-43　SSL 的握手应该发生在 TCP 的三次握手之前还是之后？它们能够合并在一起吗？请解释。

P10-44　主机 A 和主机 B 使用传输模式的 IPSec。我们可以说两台主机在它们之间需要创建一个虚拟的面向连接的服务吗？请解释。

P10-45　当我们谈论 IPSec 中的认证时，我们的意思是指消息认证（message authentication）还是实体认证（entity authentication）？请解释。

P10-46　如果 Alice 和 Bob 相互之间连续发送消息，他们可以一次性地创建一个安全关联并且利用该关联交换两个分组吗？请解释。

10.8　模拟实验

Applets

我们构建了一些 Java 小程序用于展示本章讨论的一些主要概念。强烈推荐学生激活本书网站中的这些小程序，仔细观察这些实际的协议。

实验作业

在这一部分，我们使用 Wireshark 仿真两种协议：安全外壳（secure shell，SSH）和安全超文本传输协议（HyperText Transfer Protocol Secure，HTTPS）。这些实验作业的完整描述参见本书网站。

Lab10-1　在第 2 章，我们学习了 FTP 和 TELNET。利用 FTP 传输文件、利用 TELNET 登录一个系统并不安全。在第 2 章中，我们也了解到可以使用 SSH 仿真 FTP 和 TELNET。在这个实验中，我们需要使用 SSH 并利用 Wireshark 捕获分组，以便学习本章介绍的 Internet 安全协议（SSL/TLS）怎样进行安全的文件传输和安全的登录。

Lab10-2　在第 2 章，我们学习了在 Internet 中访问 Web 页面的 HTTP 协议。HTTP 协议不提供安全性。但是，我们可以融合 HTTP 和 SSL/TLS，以增强 HTTP 的安全。这个新协议被称为安全超文本传输协议（HyperText Transfer Protocol Secure，HTTPS）。在这个实验中，我们需要使用 HTTPS 并利用 Wireshark 捕获分组，以查看使用 HTTPS 时的 SSL/TLS 分组内容。

Lab10-3　在第 4 章，我们学习了 IP 协议。在这个实验中，我们需要使用 IPSec 在两个端点之间创建一个安全的 IP 连接。

10.9　编程作业

利用你选择的编程语言，编写源代码，编译并调试如下程序：

Prg10-1　实现加性密码（加密和解密）的通用程序。程序的输入为需要进行加密或解密的标志、对称密钥、明文或密文。输出根据输入的标志为密文或明文。

Prg10-2　实现换位密码（加密和解密）的通用程序。程序的输入为需要进行加密或解密的标志、对称密钥、明文或密文。输出根据输入的标志为密文或明文。

Prg10-3　实现 RSA 加密系统的通用程序。程序的输入为需要进行加密或解密的标志、p 和 q 的值、e 的值、明文或密文。输出根据输入的标志为密文或明文。

第 11 章

Computer Networks: A Top-Down Approach

Java Socket 编程

在第 2 章中，我们讨论了使用 C 语言来编写客户-服务器程序。在本章中，我们使用 Java 来做相同的事情，来展示在 C 语言里定义的实体如何在面向对象的语言里来新定义。我们之所以选择 Java 语言，是因为编程的很多方面通过使用 Java 中的有效的强大的类能够很简单的表现出来。我们使用传统的 Socket 接口 API 来处理编程方面的主要问题，但是它们可以容易地扩展到网络编程的其他领域。我们假设读者熟悉 Java 程序的基本要素。

- 11.1 节展示诸如 IP 地址、端口、套接字地址等实体如何使用 Java 中相应的类来代替。我们使用一些程序来展示如何使用这些实体。我们也会回顾第 2 章中讨论的客户-服务器模式的概念。
- 11.2 节首先介绍 UDP 编程中使用的 Java 类。然后我们展示如何使用迭代的方法书写简单的客户-服务器程序。接下来，我们展示如何使用并行的方法来修改服务器端的程序。我们也会谈及本节中书写的通用程序如何在一些具体例子中提供服务。
- 11.3 节首先介绍 TCP 编程中使用的 Java 类。然后我们展示如何使用迭代的方法来书写简单的客户-服务器程序。最后，我们展示如何使用并行的方法来修改服务器端的程序。我们也会谈及本节中书写的通用程序如何在一些具体例子中提供服务。

11.1 介绍

本节中我们讨论在第 2 章中提到的 C 网络编程的通用思想如何在 Java 网络编程中使用。

11.1.1 地址和端口

任何语言中的网络编程都明确的需要处理 IP 地址和端口号。我们简要介绍 Java 中地址和端口是如何表示的。我们建议读者比较一下 C 和 Java 中这两种实体的表示方法。

IP 地址

正如我们在第 4 章中讨论的，因特网中有两种 IP 地址：IPv4 地址（32 位）和 IPv6 地址（128 位）。在 Java 中，一个 IP 地址被定义为一个对象，是 InetAddress 类的一个实例。这个类最初被定义为 final 类，也就是说它不能够被继承。后来 Java 修改了这个类，定义了从这个类继承的两个子类：Inet4Address 和 Inet6Address。然而大部分时间我们只使用 InetAddress 来创建 IPv4 和 IPv6 地址。表 11-1 显示了一些方法的签名。

表 11-1 InetAddress 类摘要

public class java.net.**InetAddress** extends java.lang.Object implements Serializable
// 静态方法
public static InetAddress [] **getAllByName** (String host) throws UnknownHostException
public static InetAddress **getByName** (String host) throws UnknownHostException
public static InetAddress **getLocalHost** () throws UnknownHostException

```
// 实例方法
public byte [] getAddress ()
public String toString ()
public String getHostAddress ()
public String getHostName ()
public String getCanonicalHostName ()
public boolean isAnyLocalAddress ()
public boolean isLinkLocalAddress ()
public boolean isLoopbackAddress ()
public boolean isMulticastAddress ()
public boolean isMCGlobal ()                    // MC 的全局广播
public boolean isMCLinkLocal ()
public boolean isMCNodeLocal ()
public boolean isMCOrgLocal ()                  // Org 的全局广播
public boolean isMCSiteLocal ()
public boolean isReachable (int timeout)
public boolean isReachable (NetworkInterface interface, int ttl, int timeout)
```

在 InetAddress 类中没有公有构造器，但是我们可以使用这个类中任意的静态方法来返回 InetAddress 的一个实例。这个类中也有一些实例方法，可以用来改变地址对象的格式或者获取此对象的一些信息。我们只使用几个方法，但是参考文档可以用来帮助做一些练习。

例 11.1 本例中，我们展示如何使用第二个和第三个静态方法来获取一个站点和本地主机的 InetAddress（见表 11-2）。

表 11-2 例 11.1

```
1  import java.net.*;
2  import java.io.*;
3  public class GetIPAddress
4  {
5      public static void main (String [] args) throws IOException, UnknownHostException
6      {
7          InetAddress mysite = InetAddress.getByName ("forouzan.biz");
8          InetAddress local = InetAddress.getLocalHost ();
9          InetAddress addr = InetAddress.getByName ("23.12.71.8");
10
11         System.out.println (mysite);
12         System.out.println (local);
13         System.out.println (addr);
14
15         System.out.println (mysite.getHostAddress ());
16         System.out.println (local.getHostName ());
17     } // Main 结束
18
19 } // 类结束
```

```
结果：
forouzan.biz/204.200.156.162
Behrouz/64.183.101.114
/23.12.71.8
204.200.156.162
Behrouz
```

在第 7 行，我们使用了第 2 个静态方法来获取站点"forouzan.biz"的 IP 地址。程序实际上使用了 DNS 来查找此站点的 IP 地址。在第 8 行，我们使用了第 3 个静态方法来获取我们使用的本地主机的 IP 地址。在第 9 行，我们将一个 IP 地址作为字符串传递给 getByName 方法，来将其转变为一个 InetAddress 对象。

需要注意的是一个 InetAddress 对象并不是一个字符串，但它是可序列化的。第 11 行到 13 行打印了上面存储在 InetAddress 对象中的地址：主机名后面跟着一个斜线，斜线后面跟着以点分十进制表示法表示的地址。然而，由于在第 9 行中获取的地址没有主机名，所以主机部分是空的。

第 15 行中我们可以使用 getHostAddress 方法以字符串的形式来获取 InetAddress 对象的地址部分。第 16 行中，我们使用 getHostName 方法来获取一个已知地址的主机的名字（再一次使用 DNS）。

端口号

TCP/IP 协议簇中的端口号是一个无符号 16 位整数。然而，由于 Java 没有定义无符号数值数据类型，在 Java 中的端口号被定义为整数数据类型（32 位的 int 类型），其中左边的 16 位被设置为 0。这样就防止一个很大的端口号被误认为是负数。

例 11.2 表 11-3 中的程序展示了如果我们使用 short 类型的变量来存储端口号，我们就有可能得到一个负值。*integer* 类型的变量返回给我们一个正确的值。

表 11-3 例 11.2

```
import java.io.*;
public class Ports
{
    public static void main (String [] args) throws IOException
    {
        short shortPort = (short) 0xFFF0;
        System.out.println (shortPort);

        int intPort = 0xFFF0;
        System.out.println (intPort);
    } // Main 结束

} // 类结束
```

结果：

```
-16
65520
```

InetSocketAddress

套接字地址是由 IP 地址和端口号联合组成的。在 Java 中，有一个 SocketAddress 的抽象类，但是在 Java 网络编程中使用的类却是 InetSocketAddress，该类继承自 SocketAddress 类。表 11-4 展示了此类中使用的方法摘要。

表 11-4 InetSocketAddress 类摘要

```
public class java.net.InetSocketAddress extends java.lang.SocketAddress

// 构造器
InetSocketAddress (InetAddress addr, int port)
InetSocketAddress (int port)
InetSocketAddress (String hostName, int port)
```

```
// 其他方法
public InetAddress getAddress ()
public String getHostName ()
public int getPort ()
public boolean isUnresolved ()
```

例 11.3 在表 11-5 中的程序展示了我们如何创建一个套接字地址。注意在这种表示法中端口号和 InetAddress 用冒号隔开。

表 11-5 例 11.3

```
import java.io.*;
public class SocketAddresses
{
    public static void main (String [] args) throws IOException
    {
        InetAddress local = InetAddress.getLocalHost ();
        int port = 65000;
        InetSocketAddress sockAddr = new InetSocketAddress (local, port);
        System.out.println (sockAddr);
    } // Main 结束

} // 类结束
```

```
输出:
Behrouz/84.185.101.114:65000
```

11.1.2 客户-服务器模式

在第 2 章中，我们已经讨论了客户-服务器模式（client-server paradigm）。我们说过，一个客户端（client）是一个有限的程序，它从服务器请求服务。我们也说过，一个服务器（server）是一个无限的程序，它为客户端提供服务。在客户-服务器模式中的服务器可以设计为一个迭代服务器（iterative server），也可以是一个并发服务器（concurrent server）。迭代服务器一个接一个处理客户端。当服务器在服务一个客户端时，其他的客户端必须等待。这和一个小型银行类似，里面只有一个员工为顾客（客户端）提供服务。当员工在服务一个顾客时，其他的顾客需要排队等候（队列）。并发服务器能够同时为计算机资源允许的多个客户端服务。一个不现实的类比就是，一个很大的银行，有足够的员工可以同时服务所有的顾客。当一个顾客来了，就指定一个员工为他服务。

正如我们在第 3 章中见到的，传输层协议可以是无连接的（connectionless）或者是面向连接的（connection-oriented）。它们的差别影响了在应用层中客户端和服务器程序的设计方式。使用诸如 UDP 的无连接传输层服务的客户-服务器对，应该被设计为无连接的程序。客户端需要将它的请求作为一个单一的数据块交给传输层，从传输层接收单一数据块形式的响应。服务器也需要遵循相同的过程。

使用诸如 TCP 的面向连接传输层服务的客户-服务器对，应该被设计为面向连接的程序。客户端需要在字节流中将它的请求交给传输层，从传输层接收以字节流形式的响应。服务器也需要遵循相同的过程。

客户端和服务器程序

正如我们在第2章中讨论的，TCP/IP协议簇中位于应用层下方的那些层知道如何向因特网发送分组，如何从因特网接收分组。应用层的情形是不同的。我们有两个客户-服务器程序。第一套包含有标准的客户-服务器模式，对于这些程序，它们被书写然后编译成为我们正在使用的计算机的机器语言。第二套包含有客户端和服务器的程序，这些程序需要我们为特定的目的自己编写。本章

大部分的就是关于这些类型的程序，但是为了学习，我们试着模仿一些标准的程序。

Java 中的套接字接口

在第2章（第5节）中，我们讨论过这样的观点，尽管有一些方法来书写一个客户端或者服务器应用程序，但我们专注于套接字接口。在我们专注于Java网络编程之前，我们建议回顾一下那一节。

11.2　UDP 编程

为了与第 2 章中 socket 编程那一节相一致，我们首先讨论使用无连接的 UDP 服务的网络编程。我们首先谈论迭代方法，接下来谈论并行方法。

11.2.1　迭代方法

UDP 提供了一个无连接的服务，使用称作用户数据报的数据块来完成通信。在迭代方法中，服务器一次服务一个数据报。剩下的已到达的数据报需要等待，无论它们是来自于相同的客户端还是其他的客户端。

用于 UDP 的套接字

图 11-1 展示了客户端和服务器两端的套接字的生存期。UDP 的 Java 实现使用只有一种类型的套接字对象：DatagramSocket 类的实例。注意我们正在讨论迭代的通信，也就意味着当一个客户端与服务器连接时，其他客户端不能连接（它们需要等待）。在这种类型的通信中，一个客户端发送一个数据报分组，即 DatagramSocket 类的一个实例，并且接收一个数据报分组。

类

尽管 UDP 使用只有一种类型的套接字对象，但是我们仍然需要数据报对象。在我们编写简单的客户端和服务器代码之前，让我们说明一下这两个类中使用的方法的概要。

DatagramSocket 类

DatagramSocket 类用于在客户端和服务器端创建套接字。它也提供了发送数据报、接收数据报、关闭套接字的方法。

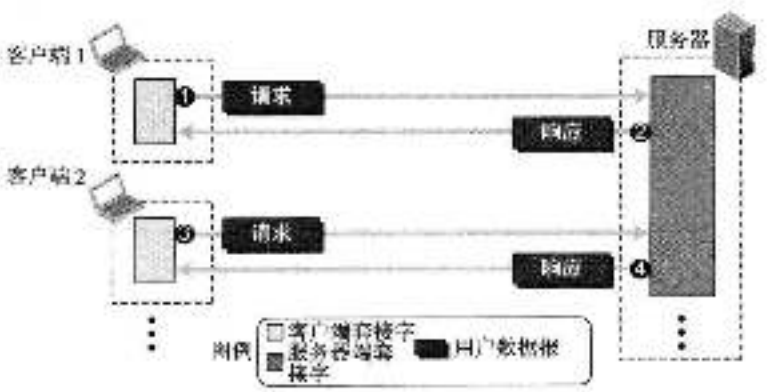


图 11-1　UDP 通信中的套接字

表 11-6 展示了这个类里面一些方法的签名。

表 11-6　Datagram Socket 类中的一些方法

public class java.net.**DatagramSocket** extends java.lang.Object
// 构造器
public **DatagramSocket** ()
public **DatagramSocket** (int localPort)
public **DatagramSocket** (int localPort, InetAddress localAddr)

```
// 实例方法
public void send (DatagramPacket sendPacket)
public void receive (DatagramPacket recvPacket)
public void close ()
```

DatagramPacket 类

DatagramPacket 类用来创建数据报分组。表 11-7 展示了这个类里面一些方法的签名。

表 11-7 DatagramPacket 类中的一些方法

```
public final class java.net.DatagramPacket extends java.lang.Object

// 构造器
public DatagramPacket (byte [] data, int length)
public DatagramPacket (byte [] data, int length, InetAddress remoteAddr, int remotePort)

// 实例方法
public InetAddress getAddress ()
public int getPort ()
public byte [] getData ()
public int getLength ()
```

UDP 客户端设计

图 11-2 展示了一些对象的设计，以及我们要编写的客户端程序中它们的关系。在我们解释这种设计之前，我们需要声明这种设计只适合于我们的客户端程序，我们的程序是很简单的。

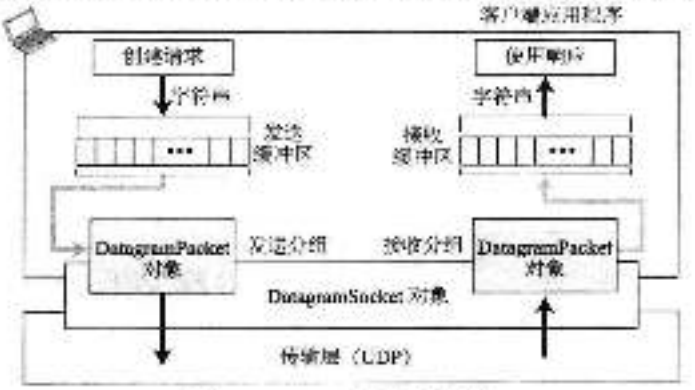


图 11-2 UDP 客户端的设计

在设计中，我们有一个客户端套接字对象（类型为 DatagramSocket），该对象由客户端程序创建用来提供客户端和传输层（UDP）间的连接。由于应用程序将数据作为块来传输，我们需要一个 DatagramPacket 类型的分组对象将数据块发送给套接字。我们也需要相同类型的分组对象从套接字接收数据块。

我们需要以字节的形式发送数据给数据报分组或是从数据报分组获取数据。因此，我们使用了发送缓冲区和接收缓冲区这两个字节数组，我们将这些字节发送给数据报之前首先将它们存储起来。为简单起见，我们的程序假设我们有创建字符串请求的方法，也有获取字符串响应的方法。在这种设计中，我们需要将请求字符串转换为字节数组格式，将字节数组转换为响应字符串的形式。

客户端程序

表 11-8 展示了一个简单的遵循图 11-2 中设计的客户端程序。我们将这个程序设计为 UDPClient 类，该类具有构造器和实例方法。第 6 行到 11 行创建常量和数据域。在运行程序之前，我们需要

定义缓冲区的大小。我们在程序之后简要地解释一下代码。

表 11-8　一个简单的 UDP 客户端程序

```
import java.net.*;
import java.io.*;

public class UDPClient
{
    final int bufSize = ...;                    // 定义缓冲区大小
    DatagramSocket sock;
    String request;
    String response;
    InetAddress servAddr;
    int servPort;

    UDPClient (DatagramSocket s, String sName, int sPort)
                                    throws UnknownHostException
    {
        sock = s;
        servAddr = InetAddress.getByName (sName);
        servPort = sPort;
    }

    void makeRequest ()
    {
        // 创建请求字符串的代码在此添加
    }

    void sendRequest ()
    {
        try
        {
            byte [] sendBuff = new byte [bufSize];
            sendBuff = request.getBytes ();
            DatagramPacket sendPacket = new DatagramPacket (sendBuff,
                                        sendBuff.length, servAddr, servPort);
            sock.send(sendPacket);
        }
        catch (SocketException ex)
        {
            System.err.println ("SocketException in getRequest");
        }
    }

    void getResponse ()
    {
        try
        {
```

```
                byte [] recvBuff = new byte [buffSize];
                DatagramPacket recvPacket = new DatagramPacket (recvBuff, buffSize);
                sock.receive (recvPacket);
                recvBuff = recvPacket.getData ();
                response = new String (recvBuff, 0, recvBuff.length);
            }
        '   catch (SocketException ex)
            {
                System.err.println ("SocketException in getRequest");
            }
        }

        void useResponse ()
        {
            // 使用 [illegible] 字符串的 [illegible] 在此结束
        }

        void close ()
        {
            sock.close ();
        }

        public static void main (String [] args) throws IOException, SocketException
        {
            final int servPort = ...;                    // 添加服务器端口号
            final String servName = ...;                 // 添加服务器名字
            DatagramSocket sock = new DatagramSocket ();
            UDPClient client = new UDPClient (sock, servName, servPort);
            client.makeRequest ();
            client.sendRequest ();
            client.getResponse ();
            client.useResponse ();
            client.close ();
        } // main 结束
    } // UDPClient 类结束
```

main 方法

程序从 main 方法（68 到 79 行）开始执行。使用者需要提供服务器端口号（整数）和服务器名称（字符串）。程序然后创建 DatagramSocket 类（72 行）的一个实例以及 UDPClient 类（73 行）的一个实例，后者负责创建请求，发送请求，接收响应，使用响应，关闭套接字。74 行到 78 行调用 UDPClient 类的合适方法来完成这项工作。

UDPClient 类中的方法

随后的部分将介绍 UDPClient 类中的方法。.

构造器　构造器（13 行到 19 行）非常简单。它获取套接字的引用，服务器的名字和服务器的端口号。它使用服务器名字来查询服务器的 IP 地址。注意这里不需要客户端的端口号和 IP 地址；它由操作系统提供。

makeRequest 方法　在我们的设计中该方法（21 行到 24 行）是空的。需要由程序的使用者来

填充创建请求字符串。我们在例子中展示一些情况。

sendRequest 方法　该方法（26 行到 40 行）负责以下几个任务：

1. 创建空的输出缓冲区（30 行）。
2. 使用由makeRequest方法创建的请求字符串填充输出缓冲区（31行）。
3. 创建一个数据报分组，将其添加至发送缓冲区、服务器地址和服务器端口（32 行到 33 行）。
4. 使用在 DatagramSocket 类中定义的发送方法发送分组（34 行）。

getResponse方法　该方法（42行到56行）负责以下几个任务：

1. 创建空的接收缓冲区（46 行）。
2. 创建接收数据报，将其附加至接收缓冲区（47 行）。
3. 使用 DatagramSocket 中的接收方法来接收服务器的响应，使用该响应来填充数据报（48 行）。
4. 提取接收数据分组中的数据，将其存储在接收缓冲区中（49 行）。
5. 创建响应字符串供 useResponse 方法使用（50 行）。

useResponse 方法　该方法（58 行到 61 行）在我们的设计中是空的。需要由程序的使用者用服务器端的响应来填充。我们在一些例子中展示一些情况。

close 方法　该方法（63 行到 66 行）关闭套接字。

UDP 服务器

图 11-3 展示了一些对象的设计，以及我们要编写的服务器程序中它们的关系。在我们解释这种设计之前，我们再一次地提醒此设计只适用于我们的服务器程序，该程序非常简单；这不是一种通用设计，通用的设计会更加复杂。在设计中，我们使用了一个 DatagramSocket 对象和两个 DatagramPacket 对象。

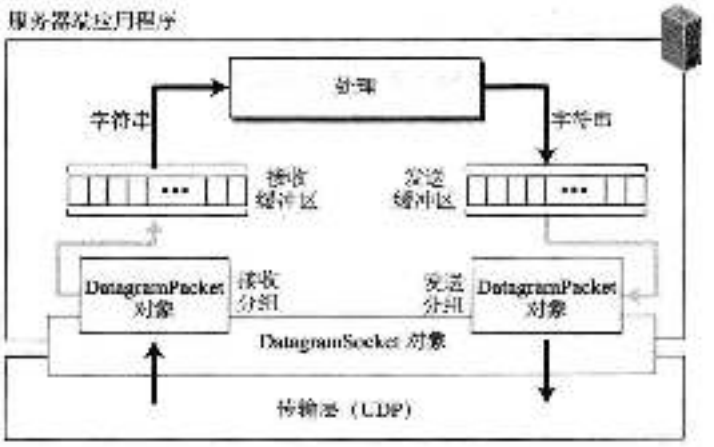


图 11-3　UDP 服务器设计

服务器程序

表 11-9 展示了一个简单的遵循图 11-3 设计的服务器程序。随后为其简要的说明。

表 11-9　简单的 UDP 服务器程序

```
1  import java.net.*;
2  import java.io.*;
3
4  public class UDPServer
5  {
6      final int buffSize = ...;                    // 定义缓冲区大小
```

```
DatagramSocket sock;
String request;
String response;
InetAddress clientAddr;
int clientPort;

UDPServer (DatagramSocket s)
{
    sock = s;
}

void getRequest ()
{
    try
    {
        byte [] recvBuff = new byte [buffSize];
        DatagramPacket recvPacket = new DatagramPacket (recvBuff, buffSize);
        sock.receive (recvPacket);
        recvBuff = recvPacket.getData ();
        request = new String (recvBuff, 0, recvBuff.length);
        clientAddr = recvPacket.getAddress ();
        clientPort = recvPacket.getPort ();
    }
    catch (SocketException ex)
    {
        System.err.println ("SocketException in getRequest");
    }
    catch (IOException ex)
    {
        System.err.println ("IOException in getRequest");
    }
}

void process ()
{
    // 处理收到的客户请求，创建响应
}

void sendResponse()
{
    try
    {
        byte [] sendBuff = new byte [buffSize];
        sendBuff = response.getBytes ();
        DatagramPacket sendPacket = new DatagramPacket (sendBuff,
                                        sendBuff.length, clientAddr, clientPort);
        sock.send(sendPacket);
    }
```

```
            catch (SocketException ex)
            {
                System.err.println ("SocketException in sendResponse");
            }
            catch (IOException ex)
            {
                System.err.println ("IOException in sendResponse");
            }
        }

        public static void main (String [] args) throws IOException, SocketException
        {
            final int port = ...;                    // 设置服务器端口号
            DatagramSocket sock  = new DatagramSocket (port);
            while (true)
            {
                UDPServer server = new UDPServer (sock);
                server.getRequest ();
                server.process ();
                server.sendResponse ();
            }
        } // main 结束
} // UDPServer 类结束
```

main 方法

程序（65 行到 76 行）由 main 方法开始执行。用户需要提供服务器侦听的端口号（整数）（67 行）。没有必要提供主机的地址或者名字，它由操作系统提供。程序然后使用指定的端口号创建 DatagramSocket 类的一个实例（68 行）。程序然后无限循环（69 行），其中在循环的一次迭代中，通过创建 UDPServer 类的一个实例以及调用它的三个实例方法来为每一个客户端服务。

UDPServer 类中的方法

随后的部分将介绍 UDPServer 类中的方法。

构造器　构造器（13 行到 16 行）很简单。它获取客户端套接字引用，将其存储在 sock 变量中。

getRequest 方法　该方法负责以下几个任务（18 行到 38 行）：

1. 创建接收缓冲区（22 行）。
2. 创建数据报分组，将其附加至缓冲区（23 行）。
3. 接收数据报内容（24 行）。
4. 提取数据报中的数据部分，将其存储至缓冲区（25 行）。
5. 将接收缓冲区中的字节转换为请求字符串（26 行）。
6. 获取发送分组的客户端的 IP 地址（27 行）。
7. 提取发送请求的客户端的端口号（28 行）。

process 方法　该方法（40 行到 43 行）在我们的设计中是空的。需要由程序的使用者来处理请求字符串、创建响应字符串。我们在一些例子中展示一些情况。

sendResponse 方法　该方法（45 行到 63 行）负责以下几个任务：

1. 创建空的输出缓冲区（49 行）。
2. 将响应字符串转换为字节，将这些字节存储在发送缓冲区中（50 行）。

3. 创建新的数据报，使用缓冲区中的数据填充（51 行、52 行）。

4. 发送数据报分组（53 行）。

例 11.4　最简单的例子就是模仿标准的 echo 客户端/服务器。该程序用来检查服务器是否在线，由客户端发送一个短的报文，该报文被精确地回显。尽管标准程序使用熟知的 7 号端口，为了模仿它，我们使用 52007 端口作为服务器端口。

1. 在客户端程序中，我们设置服务器端口为 52007，服务器名字为计算机的名字或者计算机地址（x.y.z.t）。我们也将 makeRequest 和 useResponse 方法做如下修改：

```
void makeRequest ()
{
    request = "Hello";
}

void useResponse ()
{
    System.out.println (response);
}
```

2. 在服务器程序中，我们设置服务器端口为 52007。我们将服务器程序中的 *process* 方法做如下修改：

```
void process ()
{
    response = request;
}
```

3. 我们让服务器程序在一台主机上运行，然后在另一台主机上运行客户端程序。如果服务器程序在后台运行，我们也可以在同一台机器上运行这两个程序。

例 11.5　在本例中，我们将服务器修改为一个简单的日期/时间服务器。它返回服务器运行机器上的日期和时间。

1. 在客户端程序中，我们设置服务器端口为 40013，设置服务器名字为计算机的名字或者计算机地址（"x.y.z.t"）。我们也将使用如下代码替换 makeReques 和 useResponse 方法。

```
void makeRequest ()
{
    request = "Send me data and time please.";
}

void use Response ()
{
    System.out.println (response);
}
```

2. 在服务器程序中（见表 11-9），我们在程序的开始部分增加声明来使用 Calendar 和 Date 类（import java.util.*;），我们设置服务器端口为 40013。我们也将使用如下代码来替换服务器程序中的 process 方法。

```
void process ()
{
    Date date = Calendar.getInstance ().getTime ();
    response = date.toString ();
}
```

process 方法使用 Calendar 类来获取时间（包括日期），然后将日期转换为字符串，存储在响应变量中。

3. 我们让服务器程序在一台主机上运行，然后在另一台主机上运行客户端程序。如果服务器程序在后台运行，我们也可以在同一台机器上运行这两个程序。

例 11.6　在这个例子中，我们需要使用简单的客户-服务器程序来测量从客户端发送一个报文

到服务器端的时间（以毫秒为单位）。

1. 在客户端程序中，我们在程序的开始部分添加声明来使用 Date 类（import java.util.*;），我们设置服务器端口为 40013，设置服务器名字为计算机名字或者计算机地址（"x.y.z.t"）。我们也使用如下代码来替代 makeRequest 和 useResponse 方法。

```
void makeRequest ()
{
    Date date = new Date ();
    long time = date.getTime ();
    request = String.valueOf (time);
}

void useResponse ()
{
    Date date = new Date ();
    long now = date.getTime ();
    long elapsedTime = now - Long.parse(response));
    System.out.println ("Elapsed time = " + elapsedTime + " milliseconds.");
}
```

注意尽管我们不需要发送时间值到服务器，但是我们那样做是为了不去修改客户端程序的结构。

2. 在服务器程序中，我们设置服务器端口为 40013。我们也使用如下代码来替换服务器中的 process 方法。

```
void process ()
{
    response = request;
}
```

3. 我们让服务器程序在一台主机上运行，然后在另一台主机上运行客户端程序。如果服务器程序在后台运行，我们也可以在同一台机器上运行这两个程序。

11.2.2 并发方法

对于大多数应用来说，UDP 服务器的迭代处理已经足够好了，因为在处理和发送一个数据报以后，服务器已准备好去服务其他的客户端。然而，如果一个数据报的处理持续很长时间，一个客户端可能独占服务器。并发的服务器程序就是用来解决这个问题的，它使用多线程技术。

服务器程序

表 11-10 展示了并行的服务器程序。它几乎和迭代版本（见表 11-9）相同，除了服务器类实现了 Runnable 接口，这样就允许我们重写类中的 run（）方法，在该方法（18 行到 23 行）中基本上包含先前的方法。在 main 方法中，我们创建 Thread 类的一个实例，让线程对象包含了服务器类的一个实例。这种情况下，每一个客户端在一个独立的线程中被服务。将这个版本和迭代版本（见表 11-9）相比较，我们可以看出 3 个主要的方法（request, process, response）不是在 main 方法中完成任务，而是在 run 方法中完成的。

表 11-10 一个简单的并发 UDP 服务器程序

```
1  import java.net.*;
2  import java.io.*;
3
4  public class ConcurUDPServer implements Runnable
5  {
6      final int buffSize = ...;                    // 设置缓冲区大小
7      DatagramSocket servSock;
8      String request;
```

```
    String response;
    InetAddress clientAddr;
    int clientPort;

    ConcurUDPServer (DatagramSocket s)
    {
        servSock = s;
    }

    public void run ()
    {
        getRequest ();
        process ();
        sendResponse ();
    }

    void getRequest ()
    {
        try
        {
            byte [] recvBuff = new byte [buffSize];
            DatagramPacket recvPacket = new DatagramPacket (recvBuff, buffSize);
            sock.receive (recvPacket);
            recvBuff = recvPacket.getData ();
            request = new String (recvBuff, 0, recvBuff.length);
            clientAddr = recvPacket.getAddress ();
            clientPort = recvPacket.getPort ();
        }
        catch (SocketException ex)
        {
            System.err.println ("SocketExcepion in getRequest");
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in getRequest");
        }
    }

    void process ()
    {
        // 为 process 功能保留
    }

    void sendResponse ()
    {
        try
        {
            byte [] sendBuff = new byte [buffSize];
```

```
            sendBuff = response.getBytes ();
            DatagramPacket sendPacket = new DatagramPacket (sendBuff,
                                                    sendBuff.length, clientAddr, clientPort);
            sock.send(sendPacket);
        }
        catch (SocketException ex)
        {
            System.err.println ("SocketException in sendResponse");
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in sendResponse");
        }
    }

    public static void main (String [] args) throws IOException, SocketException
    {
        final int port = ...;                    // 类相属口号
        DatagramSocket sock = new DatagramSocket (port);
        while (true)
        {
            ConcurUDPServer server = new ConcurUDPServer (sock);
            Thread thread = new Thread (server);
            thread.start ();
        }
    } // main 结束

} // UDPServer 类结束
```

例 11.7　我们使用并发方法重做例 11.5。我们需要几个计算机来同时地发送请求，获取应答。

例 11.8　我们使用并发方法来重做例 11.6。我们需要几个计算机来同时地发送请求，获取应答。

11.3　TCP 编程

我们现在准备好讨论使用 TCP 面向连接的服务进行网络编程。我们首先讨论如何使用迭代方法编写客户端和服务器程序。然后我们展示如何修改服务器程序使它可以并发执行。

11.3.1　迭代方法

尽管 TCP 编程中迭代方法很少见，但是它是并发方法的基础。在该方法中，服务器一个接一个地处理处理客户端。当服务器开始服务一个客户端时，其他的客户端需要等待。

两类套接字

TCP 的 Java 实现使用了两类的套接字对象：ServerSocket 和 Socket。正如我们在第 3 章中学习的，使用 TCP 进行通信分 3 个阶段：连接建立、数据传输、终止连接。客户端仅使用一个 Socket 对象；服务器在通信建立时使用一个 ServerSocket 对象，在其他两个阶段使用一个 ServerSocket 对象。图 11-4 展示了客户端和服务器中套接字的生命期。注意我们正在讨论的是迭代通信，也就是说当一个客户端与服务器连接时，其他的客户端不能够连接服务器（它们需要等待）。

有两类套接字对象的基本原理是分离连接建立阶段和数据传输阶段。对于迭代通信，能够说

ServerSocket（有时也叫做被动套接字和侦听套接字）只负责建立连接。在和客户端的连接建立以后，ServerSocket 对象创建一个 Socket 对象来处理客户端，然后它将休眠，知道客户端关闭连接。

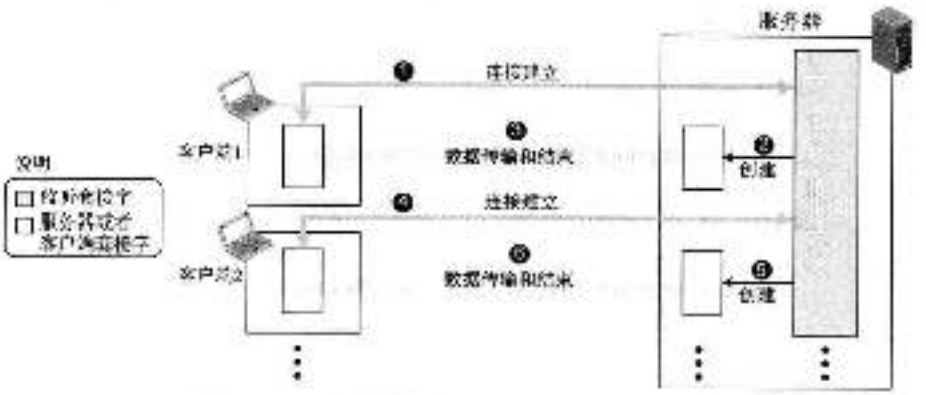


图 11-4　TCP 通信中的 ServerSocket 和 Socket 对象

类

在我们编写简单的客户端和服务器代码之前，让我们看一下两个类中使用的方法摘要。

ServerSocket 类

ServerSocket 类用来创建侦听套接字，该套接字在 TCP 中用来建立通信连接（握手）。表 11-11 展示了该类中一些方法的签名。储备定义了可排队的连接请求的数目，它们在等待连接。

表 11-11　ServerSocket 类摘要

```
public class java.net.ServerSocket extends java.lang.Object

// 构造器
ServerSocket ()
ServerSocket (int localPort)
ServerSocket (int localPort, int backlog)
ServerSocket (int localPort, int backlog, InetAddress bindAddr)

// 实例方法
public Socket accept ()
public void bind (int localPort, int backlog)
public InetAddress getInetAddress ()
public SocketAddress getLocalSocketAddress ()
```

Socket 类

在 TCP 中 Socket 类用来数据传输。表 11-12 展示了该类中一些方法的签名。

表 11-12　Socket 类摘要

```
public class java.net.Socket extends java.lang.Object

// 构造器
Socket ()
Socket (String remoteHost, int remotePort)
Socket (InetAddress remoteAddr, int remotePort)
Socket (String remoteHost, int remotePort, InetAddress localAddr, int localPort)
Socket (InetAddress remoteAddr, int remotePort, InetAddress localAddr, int localPort)
```

```
// 实例方法
public void connect (SocketAddress destination)
public void connect (SocketAddress destination, int timeout)
public InetAddress getInetAddress ()
public int getPort ()
public InetAddress getLocalAddress ()
public int getLocalPort ()
public SocketAddress getRemoteSocketAddress ()
public SocketAddress getLocalSocketAddress ()
public InputStream getInputStream ()
public OutputStream getOutputStream ()
public void shutdownInput ()
public void shutdownOutput ()
public void close ()
```

TCP 客户端设计

图 11-5 展示了一些对象的设计，以及我们要编写的客户端程序中它们的关系。在我们解释这种设计之前，我们需要声明这种设计只适合于我们的客户端程序，我们的程序是很简单的。它不是为诸如 HTTP 这些标准应用程序设计的，这些会更加复杂。

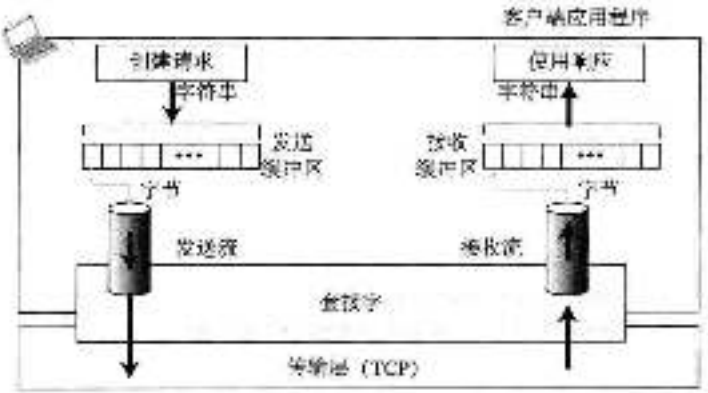


图 11-5　TCP 客户端的设计

在这种设计中，我们有一个由客户端程序创建的套接字对象（Socket 类型），它来提供客户端和传输层（TCP）之间的连接。我们需要字节流来向套接字发送字节，我们也需要字节流从套接字接收字节。在 Java 中，使用 Socket 类（见表 11-12）提供的 getOutputStream 和 getInputStream 这两个方法来创建这两个字节流。在程序中，我们也需要两个字节数组：发送缓冲区和接收缓冲区，它们分别存储发送至发送流之前的字节和接收流接收到之后的字节。为了简化，我们的程序假设我们已经有一种方法产生字符串请求，有一种方法使用字符串请求。

在这种设计中，我们需要将请求从字符串转换为字节数组，将字节数组转换为响应。然而在其他的设计中我们最好使用其他的策略。

TCP 客户端程序

表 11-13 展示了一个遵循图 11-5 设计的简单的客户端程序。我们已经设计将整个程序作为一个叫做 TCPClient 的类，该类具有构造器和实例方法。在表 11-13 后面有该程序的简要表示。

表 11-13　一个简单的 TCP 客户端程序

```
1  import java.net.*;
2  import java.io.*;
```

```
public class TCPClient
{
    Socket sock;
    OutputStream sendStream;
    InputStream recvStream;
    String request;
    String response;

    TCPClient (String server, int port) throws IOException, UnknownHostException
    {
        sock = new Socket (server, port);
        sendStream = sock.getOutputStream ();
        recvStream = sock.getInputStream ();
    }

    void makeRequest ()
    {
        // 译者代码产生请求字符串。
    }

    void sendRequest ()
    {
        try
        {
            byte [] sendBuff = new byte [request.length ()];
            sendBuff = request.getBytes ();
            sendStream.write (sendBuff, 0, sendBuff.length);
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in sendRequest");
        }
    }

    void getResponse ()
    {
        try
        {
            int dataSize;
            while ((dataSize = recvStream.available ()) == 0);
            byte [] recvBuff = new byte [dataSize];
            recvStream.read (recvBuff, 0, dataSize);
            response = new String (recvBuff, 0, dataSize);
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in getResponse");
```

```
        }
    }

    void useResponse ()
    {
        // 添加代码使用响应字符串。
    }

    void close ()
    {
        try
        {
            sendStream.close ();
            recvStream.close ();
            sock.close ();
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in close");
        }
    }

    public static void main (String [] args) throws IOException
    {
        final int servPort = ...;                    // 提供服务器端口
        final String servName = "...";               // 提供服务器名字
        TCPClient client = new TCPClient (servName, servPort);
        client.makeRequest ();
        client.sendRequest ();
        client.getResponse ();
        client.useResponse ();
        client.close ();
    }// main 结束
}// TCPClient 类结束
```

main 方法

程序从 main 方法（73 行到 83 行）开始执行。使用者需要提供服务器端口号（整数）和服务器名字（字符串）。然后程序创建 TCPClient 类的一个实例。该实例负责创建请求，发送请求，接收响应，使用响应，关闭套接字和流。78 行到 82 行调用 TCPClient 类中恰当的方法来完成该任务。

TCPClient 类中的方法

随后的部分描述了 TCPClient 类中的方法。

构造器　构造器（12 行到 17 行）很简单。它获得服务器名字和服务器端口号的引用。它使用服务器名字来查询服务器的 IP 地址。构造器也创建输出和输入流来发送和接收数据。

makeRequest 方法　在我们的设计中该方法（19 行到 22 行）是空的。需要由程序的使用者来填充创建请求字符串。我们在例子中展示一些情况。

sendRequest 方法　该方法（24 行到 36 行）负责以下几个任务：

1. 创建空的发送缓冲区（28 行）。

2. 使用 makeRequest 方法中创建的请求字符串填充发送缓冲区（29 行）。

3. 使用输出流中的写入方法写入发送缓冲区中的内容（30 行）。

getResponse 方法 该方法（38 行到 53 行）负责以下几个任务：

1. 在一个空循环中连续不断地寻找可用的字节数（43 行）。

2. 创建适当大小的缓冲区（44 行）。

3. 从输入流中读取数据，将其存储至接收缓冲区（45 行）。

4. 将接收缓冲区的字节转换为 useResponse 方法中使用的响应字符串（46 行）。

useResponse 方法 在我们的设计中该方法（54 行到 57 行）是空的。需要由程序的使用者来填充使用服务器发送的响应。我们在例子中展示一些情况。

close 方法 该方法（59 行到 71 行）首先关闭流，然后关闭套接字。

TCP 服务器

图 11-6 展示了一些对象的设计，以及我们要编写的服务器程序中它们的关系。在我们解释这种设计之前，我们需要声明这种设计只适合于我们的服务器端程序，我们的程序是很简单的。它不代表一种通用设计，通用设计会更加复杂。

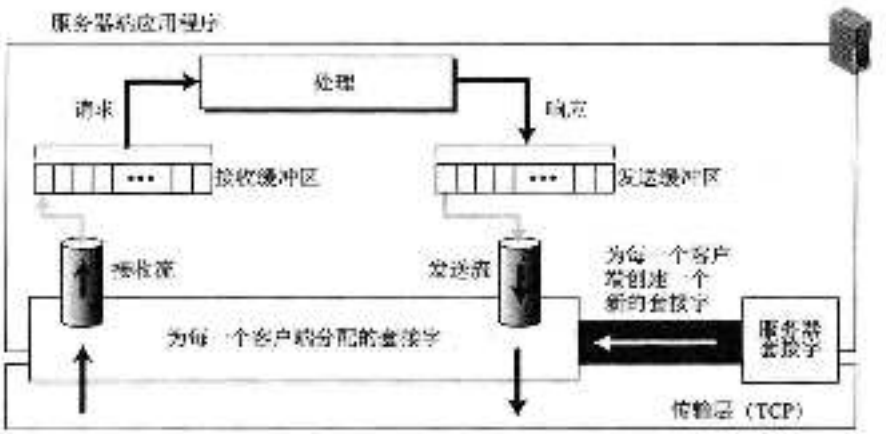


图 11-6 为每个客户端连接提供服务的 TCP 服务器的设计

在设计中，我们有一个由服务器程序创建的，用来侦听来自客户端的连接请求的套接字（ServerSocket 类型）。该侦听套接字使用 ServerSocket 类中定义的 accept 方法，迭代的为每一个客户端创建一个套接字。

服务器程序

表 11-14 展示了一个遵循图 11-6 设计的简单的服务器程序。程序的后面有一个简要的说明。我们假设请求和响应是小块的数据。

表 11-14 一个简单的 TCP 服务器程序

```
1  import java.net.*;
2  import java.io.*;
3
4  public class TCPServer
5  {
6      Socket sock;
7      InputStream recvStream;
8      OutputStream sendStream;
```

```
String request;
String response;

TCPServer (Socket s) throws IOException, UnknownHostException
{
    sock = s;
    recvStream = sock.getInputStream ();
    sendStream = sock.getOutputStream ();
}

void getRequest ()
{
    try
    {
        int dataSize;
        while ((dataSize = recvStream.available ()) == 0);
        byte [] recvBuff = new byte [dataSize];
        recvStream.read (recvBuff, 0, dataSize);
        request = new String (recvBuff, 0, dataSize);
    }
    catch (IOException ex)
    {
        System.err.println ("IOException in getRequest");
    }
}

void process()
{
    // 添加代碼來处理所求字符串，引用 [illegible]。
}

void sendResponse ()
{
    try
    {
        byte [] sendBuff = new byte [response.length ()];
        sendBuff = response.getBytes ();
        sendStream.write (sendBuff, 0, sendBuff.length);
    }
    catch (IOException ex)
    {
        System.err.println ("IOException in sendResponse");
    }
}

void close ()
{
    try
```

```
57          {
58              recvStream.close ();
59              sendStream.close ();
60              sock.close ();
61          }
62          catch (IOException ex)
63          {
64              System.err.println ("IOException in close");
65          }
66      }
67
68      public static void main (String [] args) throws IOException
69      {
70          final int port = ...;                    // 指定端口号
71          ServerSocket listenSock = new ServerSocket (port);
72          while (true)
73          {
74              TCPServer server = new TCPServer (listenSock.accept ());
75              server.getRequest ();
76              server.process ();
77              server.sendResponse ();
78              server.close ();
79          }
80      } // main 结束
81  } // TCPClient 类结束
```

main 方法

程序从 main 方法（68 行）开始执行。使用者需要提供服务器侦听的端口号（整数）。没有必要提供主机的地址或者名字；它由操作系统提供。程序现在使用定义的端口号创建 ServerSocket 类（71 行）的一个实例。然后程序运行一个无限循环（72 行）。其中在循环的每次迭代中通过创建 TCPServer 类的一个实例以及调用它的实例方法来为一个客户端服务。

TCPServer 类中的方法

随后描述该类中的方法。

构造器　构造器（12 行到 17 行）很简单。它获取套接字的引用。它也创建接收流和发送流。

getRequest 方法　该方法（19 行到 33 行）负责以下任务：

1. 连续不断地寻找可用的字节大小（24 行）。
2. 创建合适大小缓冲区（25 行）。
3. 从输入流中读取数据，将其存储至接收缓冲区（26 行）。
4. 将接收缓冲区内的字节转换为 process 方法使用的响应字符串（27 行）。

process 方法　在我们的设计中该方法（35 行到 38 行）是空的。需要由程序的使用者来填充处理请求字符串以及创建响应字符串。我们在例子中展示一些情况。

sendResponse 方法　该方法（40 行到 52 行）负责以下几个任务：

1. 创建一个空的发送缓冲区（44 行）。
2. 将响应字符串转换为字节，使用这些字节填充发送缓冲区（45 行）。
3. 将字节写入发送流（46 行）。

close 方法　该方法（54 行到 66 行）负责关闭为每一个客户端创建的流和套接字。

11.3.2 并发方法

TCP 服务器程序的迭代方法允许一个客户端独占服务器，不允许服务器关注其他客户端的请求。并发服务器程序用来解决这个问题。在过去，当并发服务器使用 UNIX 环境的 C 语言编写时，一个服务器使用多个进程来服务多个客户端。在 Java 中，该任务使用多线程来完成。

服务器程序

表 11-15 展示了并发服务器程序。它几乎和迭代版本相同，除了服务器类实现了 Runnable 接口，这就允许我们覆盖该类中的 run ( ) 方法，在该方法（19 行到 26 行）中基本上包含所有先前的方法。在 main 方法中，我们创建了一个 Thread 类，让这个线程对象包含了服务器类的一个实例。这种情况下，每一个客户端在一个单独的类中接受服务。

表 11-15 一个简单的并发 TCP 服务器程序

```
1  import java.net.*;
2  import java.io.*;
3
4  public class ConcurTCPServer implements Runnable
5  {
6      Socket sock;
7      InputStream recvStream;
8      OutputStream sendStream;
9      String request;
10     String response;
11
12     ConcurTCPServer (Socket s) throws IOException
13     {
14         sock = s;
15         recvStream = sock.getInputStream ();
16         sendStream = sock.getOutputStream ();
17     }
18
19     public void run()
20     {
21         getRequest ();
22         process ();
23         sendResponse ();
24         close ();
25     }
26
27     void getRequest ()
28     {
29         try
30         {
31             int dataSize;
32             while ((dataSize = recvStream.available ()) == 0);
33             byte [] recvBuff = new byte [dataSize];
34             recvStream.read (recvBuff, 0, dataSize);
35             request = new String (recvBuff, 0, dataSize);
```

```
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in getRequest");
        }
    }

    void process ()
    {
        // 添加代码来使用请求字符串，提供响应字符串。
    }

    void sendResponse ()
    {
        try
        {
            byte [] sendBuff = new byte [response.length ()];
            sendBuff = response.getBytes ();
            sendStream.write (sendBuff, 0, sendBuff.length);
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in sendResponse");
        }
    }

    void close ()
    {
        try
        {
            recvStream.close ();
            sendStream.close ();
            sock.close ();
        }
        catch (IOException ex)
        {
            System.err.println ("IOException in close");
        }
    }

    public static void main (String [] args) throws IOException
    {
        final int port = ...;                    // 提供服务器端口
        ServerSocket listenSock = new ServerSocket (port);
        while (true)
        {
            ConcurTCPServer server = new ConcurTCPServer (listenSock.accept ());
```

```
            Thread thread = new Thread (server);
            thread.start ();
        }
    } // main 结束
} // ConcurTCPServer 类结束
```

11.4 章末资料

推荐读物

一些书籍详细地讲述了 Java 网络编程，包括：[Cul & Don 08]、[Pit 06]和[Har 05]。

小结

网络编程肯定需要处理 IP 地址和端口号。在 Java 中，一个 IP 地址是 InetAddress 类的一个实例。Inet4Address 和 Inet6Address 这两个子类明确地用来代表 IPv4 地址和 IPv6 地址。一个端口号用一个整数表示。在 Java 中，一个套接字地址是 IP 地址和端口号的联合表示。它通过 SocketAddress 类来表示，但是 SocketAddress 类的子类 InetSocketAddress 类通常在 Java 编程中使用。

在客户-服务器模式中，通信发生在两个应用程序之间，一个客户端和一个服务器。客户端是一个有限的程序，它请求来自于服务器的服务。客户-服务器模式中的服务器可以被设计为一个迭代服务器，也可以被设计为一个并发服务器。迭代服务器一个接一个的处理客户端。并发服务器可以同时服务计算机资源允许的尽可能多的客户端。使用诸如 UDP 的无连接传输层服务的客户-服务器对应该被设计为无连接程序。使用诸如 TCP 的面向连接的传输层服务的客户-服务器对应该被设计为面向连接的程序。

尽管有一些方式来编写客户端或者服务器应用程序，但是我们只讨论套接字接口的方式。整体思路就是在操作系统和应用程序之间创建一个新的抽象层。

基于 UDP 的应用程序的 Java 实现使用了两个类：DatagramSocket 和 DatagramPacket。第一个是用来创建套接字对象的；第二个是用来创建交换数据报的。基于 TCP 的应用程序的 Java 实现使用了两个类：ServerSocket 和 Socket。第一个只在连接建立阶段使用；第二个在余下的通信阶段使用。Java 网络编程中的并行方法使用多个线程来允许服务器在一个单独线程中服务一个客户端。

11.5 习题集

测试题

本章的交互测验题集可以在本书的网站上找到。强烈推荐学生们做这些测验题，这样可以在学生做习题集前来检测其对课程资料的理解。

练习题

Q11-1 在 Java 中一个 IP 地址如何表示？

Q11-2 Java 如何区分 IPv4 和 IPv6 地址？

Q11-3 在 TCP/IP 协议簇中一个端口号是一个无符号 16 位整数。我们在 Java 中如何使用一个 32 位整数来表示一个端口号？

Q11-4 你认为 Java 为什么使用一个类的一个实例来表示 IP 地址，而不用一个整数来表示？

Q11-5 你认为为什么 Java 没有提供 InetAddress 类的构造器？

Q11-6 你知道一台计算机的域名是"aBusiness.com"，使用 Java 写一段程序来创建一个与这台计算机相关联的 InetAddress 对象。

Q11-7 你知道一台计算机的 IP 地址是"23.14.76.44"，使用 Java 写一段程序来创建一个与该地址相关联的 InetAddress 对象。

Q11-8　你认为一台域名为 "aCollege.edu" 的计算机有几个 IP 地址。使用 Java 写一段程序来创建与该主机相关联的 InetAddress 对象数组。

Q11-9　你认为一台 IP 地址为 "14.26.89.101" 的计算机还有更多的 IP 地址。使用 Java 写一段程序来创建与该主机相关联的 InetAddress 对象数组。

Q11-10　你想要写一段程序，在该程序中你需要使用你工作的计算机的 InetAddress 对象。使用 Java 写一段程序来创建相应的对象。

Q11-11　你想要创建与你工作的计算机相关联的所有的 InetAddress 对象。使用 Java 写一段程序来完成该工作。

Q11-12　使用 Java 写一段程序，将端口号 62230 存储在 Java 的变量中，并保证该端口号以无符号整数来存储。

Q11-13　一个套接字地址是 IP 地址和端口号的联合表示，端口号定义了运行在主机上的一个应用程序。我们能不能使用一个未分配给任何主机的 IP 地址来创建一个 InetSocketAddress 类的实例。

Q11-14　使用 Java 写一段程序来创建一个与本地主机和 HTTP 服务器进程绑定的套接字地址。

Q11-15　使用 Java 写一段程序来创建一个与本地主机和临时端口号 56000 绑定的套接字地址。

Q11-16　使用 Java 写一段程序来创建一个与域名为 "some.com" 的主机和一个端口号为 51000 的客户端进程绑定的套接字地址。

Q11-17　使用 Java 写一段程序来创建一个与 InetSocketAddress 对象 addr 和 TELNET 服务器进程绑定的套接字地址。

Q11-18　使用 Java 写一段程序来从 InetSocketAddress 对象 sockAd 中提取 InetAddress。

Q11-19　使用 Java 写一段程序来从 InetSocketAddress 对象 sockAd 中提取端口号。

Q11-20　Java 中 DatagramSocket 类和 Socket 类有什么不同？

Q11-21　Java 中 ServerSocket 类和 Socket 类有什么不同？

Q11-22　一种说法是在网络编程中一个套接字应该至少和一个本地套接字地址绑定。DatagramSocket 类（见表 11-6）的第一个构造器没有参数。你能否解释当它在客户端站点使用时，它是如何和本地套接字地址绑定的？

Q11-23　DatagramPacket 类有两个构造器（见表 11-7）。哪一个构造器可以用来发送分组？

Q11-24　DatagramPacket 类有两个构造器（见表 11-7）。哪一个构造器可以用来接收分组？

Q11-25　在图 11-2 中，如果客户端需要发送一个报文而非字符串（例如一张图片），那么需要什么变化？

Q11-26　在图 11-3 中，假设请求是要检索的图片的 URL，该 URL 如何在接收缓冲区中存储？

Q11-27　解释一个 UDP 客户端程序（表 11-8）在接收到服务器的相应之前如何休眠？

Q11-28　TCP 客户端（图 11-5）中如何创建和销毁一个套接字对象？

Q11-29　在 Java 中我们有输入流类。你能否解释为什么 TCP 客户端程序不直接使用这些类来创建输入流？

Q11-30　在图 11-5 中，客户端如何创建输入输出流来与服务器端交互？

思考题

P11-1　使用 Java 写一个方法来接受以 "x.y.z.t" 形式表示的 IP 地址的字符串，将其转换为一个无符号整数。

P11-2　使用 Java 写一个方法将一个 32 位整数转换为以 "x.y.z.t" 形式表示的 IP 地址的字符串。

P11-3　给定以 "x.y.z.t/n" 形式表示的 CIDR 表示法的字符串，使用 Java 写一个方法来提取一个地址（作为一个整数）的前缀。

P11-4　使用 Java 写一个方法从 CIDR 表示法（"x.y.z.t/n"）表示的字符串中提取 IP 地址（没有前缀）作为以点分十进制表示的字符串。

P11-5　使用 Java 写一个方法在一个 IP 地址的末尾添加一个给定的前缀（作为一个整数）来创建以 CIDR 表示法（"x.y.z.t/n"）表示的字符串。

P11-6　使用 Java 写一个方法将一个表示前缀的整数转换为一个表示数值掩码的无符号 32 位整数。

P11-7　使用 Java 写一个方法将一个表示掩码的 32 位整数转换为一个表示前缀（/n）的整数。

P11-8　当以 CIDR 表示法的形式给定块中的一个地址时，使用 Java 写一个方法来查找该块中第一个地址（网络地址）。

P11-9　当以 CIDR 表示法的形式给定块中的一个地址时，使用 Java 写一个方法来查找该块中最后一个地址。

P11-10　当给定块（以 CIDR 表示法表示）中的一个地址，使用 Java 来查找该块的大小。

P11-11　当给定开始地址和结束地址时，使用 Java 写一个方法来查找地址的范围。

P11-12 使用并行方法重做例 11.4。

P11-13 使用 TCP 的服务重做例 11.4。

P11-14 使用并行方法重做习题 11.13。

11.6 编程作业

Prg11-1 修改、编译、测试表 11-8 中的客户端程序和表 11-9 中的服务器程序来完成以下要求：客户端程序需要客户端从文件中读取请求字符串，将响应字符串存储在另一个文件中。文件的名字需要作为参数传递给客户端程序的 main 方法。服务器程序需要接受请求字符串，将所有的小写字母转换为大写字母，返回结果。

Prg11-2 修改、编译、测试表 11-13 中的客户端程序和表 11-14 中的服务器程序来允许客户端提供存储在服务器主机上的短文件的路径名。服务器需要以字符串的形式发送该短文件的内容。客户端在客户端主机上存储该文件。这就意味着模拟一个简单的文件传输协议。

Prg11-3 修改、编译、测试表 11-13 中的客户端程序和表 11-14 中的服务器程序来模拟一个本地 DNS 客户端和服务器。该服务器有一个很短的表，该表有两列组成，分别为域名和 IP 地址。客户端可以发送两类请求：正向和反向。正向请求是“N:domain name”格式的字符串；反向请求是“R:IP address”格式的字符串。服务器使用 IP 地址、域名或者报文“Not found.”来响应。

Prg11-4 只使用非持续连接，编写并发的 TCP 客户-服务器程序来模拟一个简单版本的 HTTP 协议（第 2 章）。该客户端发送一个 HTTP 报文；服务器使用请求文件来响应。只使用两类方法-GET 和 PUT。只使用简单的头部。注意在测试以后，你应该可以在 Web 浏览器中测试你的程序。

Prg11-5 编写并发的 TCP 客户-服务器程序来模拟一个简单版本的 POP 协议（第 2 章）。客户端发送请求来接收邮箱里的邮件；服务器使用邮件来响应。

Unicode

计算机使用数字。它们通过给每个字符分配一个数字来存储字符。最初的编码系统称为美国信息交换标准编码(American Standard Code for Information Interchange，ASCII)，有 128 个字符（0 到 127），每一个存为 7 位的数字。ASCII 能够令人满意地处理小写字母、大写字母、数字、标点符号和一些控制字符。过去尝试扩展 ASCII 字符集至 8 位。新的编码称为扩展 ASCII 码（Extended ASCII），从来没有国际性的标准化过。

为了克服 ASCII 和扩展 ASCII 固有的一些问题，Unicode 协会（很多多语种的软件制造商组成的组织）创建了通用编码系统来提供广泛的字符集，称为 Unicode。

Unicode 最初是 2 个字节的字符集。然而，Unicode 版本 3 是一个 4 字节的编码并且全面兼容 ASCII 和扩展 ASCII。ASCII 集现在称为 Basic Latin，它是 Unicode 将最高 25 位有效位置 0 得到的。扩展 ASCII 现在称为 Latin-1，它是 Unicode 最高的 24 位有效位置 0 得到的。图 A-1 显示了不同的系统如何兼容的。

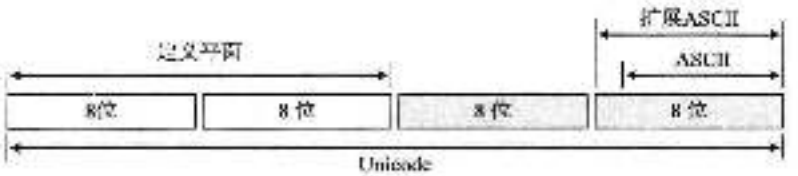


图 A-1 Unicode 字节

这种编码中的每一个字符或是符号由一个 32 位数字定义。编码能够定义多达 2^{32}（4 294 967 296）个字符或符号。这种表示法使用十六进制数字按以下格式表示。

U-XXXXXXXX

每一个 X 是一个十六进制数字。因此，编号从 U-00000000 到 U-FFFFFFFF。

A.1 平面

Unicode 将可用代码空间划分为平面。最高有效的 16 位定义了平面，这意味着我们有 65 536 个平面。每个平面能够定义多达 65 536 个字符或是符号。图 A-2 显示了 Unicode 空间的结构和平面。

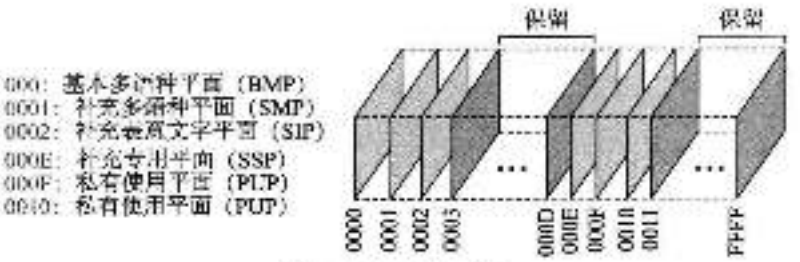


图 A-2 Unicode 平面

A.1.1 基本多语种平面（BMP）

平面$(0000)_{16}$，即基本多语种平面（BMP），用来兼容先前的 16 位 Unicode。该平面中最高有效的 16 位全是 0。编码通常以 U+XXXX 表示，其意思是 XXXX 只定义了最低有效的 16 位。该平面主要定义了不同语言中的字符集，除了一些用于控制和其他特殊字符的编码之外。

A.1.2 其他平面

有一些其他平面（非保留的），我们简述如下：

补充多语种平面（SMP）

平面$(0001)_{16}$，即补充多语种平面（SMP），用来为那些没有包含在 BMP 中的多语言字符提供更多的编码。

补充表意文字平面(SIP)

平面 $(0002)_{16}$，即补充表意文字平面（SIP），用来为表意符号（与表示声音（或发音）相对照，主要用来表示想法（或是意义）的符号）提供编码。

补充专用平面（SSP）

平面$(000E)_{16}$，即补充专用平面（SSP），用于表示特定字符。

私有使用平面(PUP)

平面$(000F)$和$(0010)_{16}$，即私有使用平面（PUP），用于私有使用。

A.2 ASCII

美国信息交换标准编码（American Standard Code for Information Interchange，ASCII）是一个 7 位的编码，用来为 128 个符号（大部分是美式英语中的）提供编码。现在 ASCII 或是 Basic Latin 是 Unicode 的一部分，它占据着 Unicode 中前面的 128 个编码（00000000 到 0000007F）。表 A-1 包含十六进制和图形编码（符号）。十六进制编码只定义了 Unicode 中两个最低有效数字。要查找真正的编码，我们在编码前预加以十六进制表示的 000000。

表 A-1 ASCII 编码

符　号	十六进制	符　号	十六进制	符　号	十六进制	符　号	十六进制
NULL	00	SP	20	@	40	`	60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	'	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	-	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F

（续）

符　号	十六进制	符　号	十六进制	符　号	十六进制	符　号	十六进制
DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B	{	7B
FS	1C	<	3C	\	5C	|	7C
GS	1D	=	3D	]	5D	}	7D
RS	1E	>	3E	^	5E	~	7E
US	1F	?	3F	_	5F	DEL	7F

ASCII 的一些性质

ASCII 有一些有趣的性质，在此我们简单提一下。

1. 空格符号字符$(20)_{16}$，是一个可打印的字符。它打印一个空白区。

2. 大写字母从$(41)_{16}$开始。小写字母从$(61)_{16}$开始。当比较时，大写字母数值上比小写字母小。这意味着在基于 ASCII 值排序的列表中，大写字母出现在小写字母前面。

3. 在 7 位编码中，大写字母和小写字母只有一位不同。例如，字符 *A* 是$(1000001)_2$，字符 *a* 是$(1100001)_2$。第六位不同，在大写字母中是 0，在小写字母中是 1。如果我们知道一种情况的编码，就可以很容易地通过加或是减$(20)_{16}$计算出另一种情况的编码，或是我们简单地替换第六位。

4. 小写字母不紧跟在大写字母后面，在中间有很多的标点字符。

5. 数字（0 到 9）从$(30)_{16}$开始。这意味着如果你想要将一个数字字符变化成整数面值，只需要用它减去$(30)_{16}$ − 48 即可。

6. 前面的 32 个字符，$(00)_{16}$到$(1F)_{16}$，以及后面的字符$(7F)_{16}$，是非打印字符。字符$(00)_{16}$是简单的用于定义一个字符串结尾的定界符。字符$(7F)_{16}$是一个删除字符，一些编程语言使用它来删除前一个字符。剩余的非打印字符称为控制字符（control characters），在数据通信中使用。表 A-2 给出了这些字符的说明。

表 A-2　ASCII 编码

符　号	解　释	符　号	解　释
SOH	报头开始	DC1	设备控制 1
STX	文本开始	DC2	设备控制 2
ETX	文本结束	DC3	设备控制 3
EOT	传输结束	DC4	设备控制 4
ENQ	询问	NAK	否定信号
ACK	确认	SYN	同步空闲字符
BEL	响铃	ETB	组的传输结束

（续）

符　号	解　释	符　号	解　释
BS	退格	CAN	取消
HT	水平制表符	EM	媒体结束符
LF	换行	SUB	替代
VT	垂直制表符	ESC	退出符
FF	换页	FS	文件分隔符
CR	回车	GS	组分隔符
SO	移出	RS	记录分隔符
SI	移入	US	单元分隔符
DLE	数据传送换码		

按位计数系统

按位计数系统使用一组符号。但是，每个符号代表的值依赖于它的面值和位值，该值与其在数字中占据的位置有关。换言之，我们有以下几点。

$$\text{符号值} = \text{面值} \times \text{位值}$$

$$\text{数字值} = \text{符号值之和}$$

本附录中，我们只讨论整数，即没有小数部分的数字；实数，即有小数部分的数字，其讨论与之类似。

B.1 不同的系统

我们首先显示整数在4个不同系统中是如何表示的：base 10、base 2、base 16 和 base 256。

B.1.1 Base 10：十进制

我们讨论的第一个位值系统称为**十进制系统**。术语十进制来源于拉丁词根 decem（意为 10）。十进制系统使用 10 个符号(0, 1, 2, 3, 4, 5, 6, 7, 8 和 9)，它们和符号本身一样拥有相同的面值。十进制数字系统中的位值是 10 的幂。图 B-1 显示了整数 4 782 的位值和符号值。

十进制系统使用 10 个符号，其中位值是 10 的幂。

B.1.2 Base 2:二进制

我们要讨论的第二个位值系统称为**二进制系统**。术语二进制来源于拉丁词根 bi（意为两个两个的）。二进制系统使用两个符号（0 和 1），它们和符号本身一样拥有相同的面值。二进制数字系统中的位值是 2 的幂。图 B-2 显示了二进制数$(1101)_2$中的位值和符号值。注意我们使用下标 2 来说明该数字是二进制的。

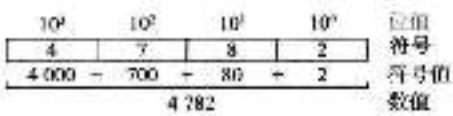


图 B-1 十进制数字的例子

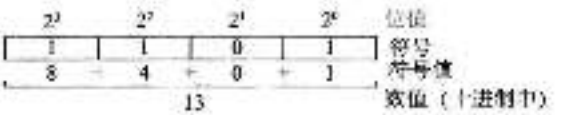


图 B-2 二进制数字的例子

二进制系统使用 2 个符号，其中位值是 2 的幂。

B.1.3 Base 16: 十六进制

我们要讨论的第三个位值系统称为**十六进制系统**。术语十六进制来源于希腊词根 hex（意为 6）和拉丁词根 decem（意为 10）。十六进制系统使用 16 个符号(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E 和 F)。前十个符号的面值和符号一样拥有相同的面值，但是符号 A 到 F 的面值分别为 10 到 15。十六进制数字系统中的位值是 16 的幂。图 B-3 显示了十六进制数$(A20E)_{16}$中的位值和符号值。注意我们使用下标 16 来说明该数字是十六进制的。

十六进制系统使用 16 个符号，其中位值是 16 的幂。

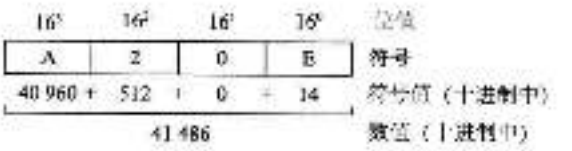


图 B-3 十六进制数字的例子

B.1.4 Base 256: 点分十进制表示法

我们要讨论的第四个位值系统是 base 256，称为点分十进制表示法。该系统用于表示 IPv4 地址。该系统中的位值是 256 的幂。但是，由于使用 256 个符号几乎是不可能，该系统中的符号是 0 到 255 之间十进制数字，和符号一样拥有相同的面值。为了将这些数字彼此分开，系统使用点，如我们第 4 章中讨论的一样。图 B-4 显示了地址（14.18.111.252）的位值和符号值。注意我们在 IPv4 地址中使用多于 4 个符号。

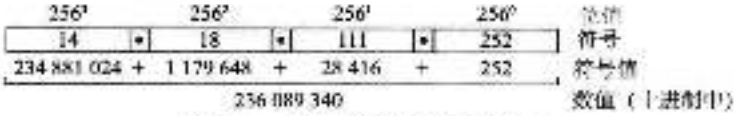


图 B-4 点分十进制表示法的例子

点分十进制表示法使用十进制数字（0 到 255）作为符号，但是在每一个符号之间插入一个点。

B.1.5 对比

表 B-1 显示了三个不同的系统如何表示十进制数字 0 到 15。例如，十进制数字 13 等价于二进制$(1101)_2$ 十六进制 D。

表 B-1 三个系统的比较

十 进 制	二 进 制	十 六 进 制	十 进 制	二 进 制	十 六 进 制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

B.2 转换

我们需要知道如何将一个系统中一个数字转换为另一个系统的等价数字。

B.2.1 从任意基转换为十进制

图 B-2 到图 B-4 实际上说明了我们如何手动地将以任意基表示的一个数转换为十进制的。但是，使用图 B-5 中的算法更加简单。该算法基于这一事实，即下一个位值是前一个位置乘以基（2，16 或 256）。该算法是通用的，能够用来将给定基表示的一串符号转换为一个十进制数字。算法中唯一不同的一段是如何获取字符串中的下一个符号并且计算其面值。在基为 2 时，很简单；面值可以通过将符号转换为数字值即可。在基为 16 时，我们需要考虑以下符号的面值：符号 A 的面值为 10，符号 B 的面值为 11，依此类推。在基为 256 时，我们需要提取由点号分界的每个字符串，将字符

串转换为其数值。

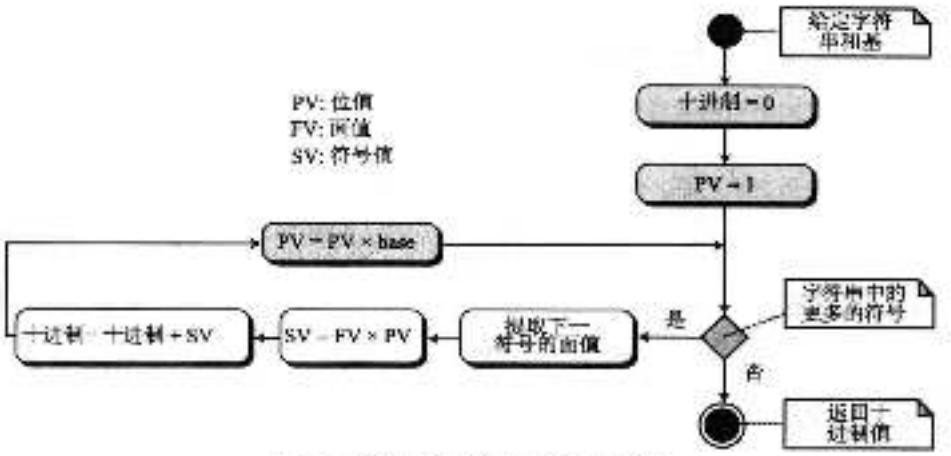


图 B-5 从任意基转换为十进制的算法

小数字的上述算法的手动实现可以在一些例子中说明。

例 B.1 计算二进制数字$(11100111)_2$的十进制等价值。

解答

我们按如下所示遵循该算法。

128		64		32		16		8		4		2		1	位值
1		1		1		0		0		1		1		1	面值
128		64		32		0		0		4		2		1	符号值
231		103		39		7		7		7		3		1	十进制 = 0

十进制的值最初设置为 0。当循环终止时，十进制的值是 231。

例 B.2 说明 IPv4 地址 12.14.67.24 的十进制等价值。

解答

我们按如下所示遵循该算法。

16 777 216		65 536		256		1	位值
12	*	14	*	67	*	24	面值
201 326 592		917 504		17 152		24	符号值
202 261 272		934 680		17 176		24	十进制 = 0

十进制的值最初设置为 0。当循环终止时，十进制的值是 202 261 272。

B.2.2 将十进制值转换为任意基

如果我们不断地将十进制数字除以基来获取余数和商，就可以将十进制数值转换为任意基的数。余数是下一个符号面值；商是下一次循环使用的十进制值。作为相反转换的情况，我们需要一个独立的算法来将相应基中一个符号的面值变换为实际的符号，将其插入代表转换后数字的字符串中。图 B-6 展示了从十进制转换为任意基的过程。

我们可以在一些例子中说明如何手动地遵循该算法。

例 B.3 将十进制数字 25 转换为其二进制等价值。

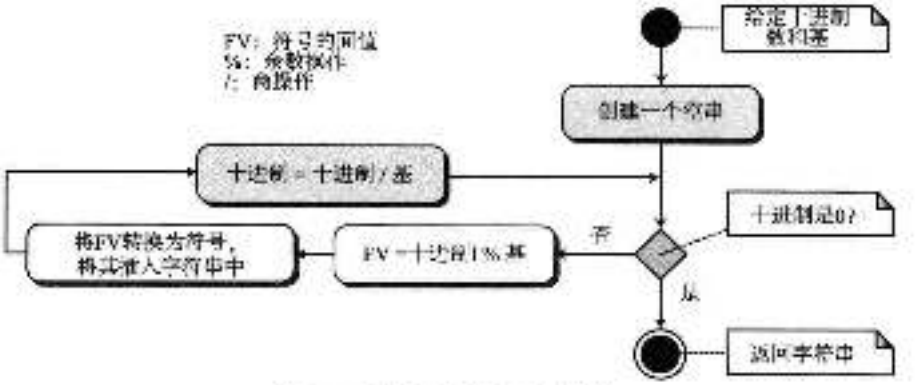


图 B-6 从十进制转换为任意基

解答

我们不断地将十进制值除以 2（二进制系统的基）直至商为 0。在每次除法中，我们将余数的值作为下一个要插入二进制字符串的符号。向下箭头显示了余数；向左箭头显示了商。当十进制值变为 0 时，我们停止。结果是二进制字符串$(11001)_2$。

0	←	1	←	3	←	6	←	12	←	25	十进制
		↓		↓		↓		↓		↓	
		1		1		0		0		1	二进制

例 B.4 将十进制数字 21 432 转换为其十六进制等价值。

解答

我们不断地将十进制值除以 16（十六进制系统的基）直至商为 0。在每次除法中，我们将余数的值作为下一个要插入十六进制字符串的符号。结果是十六进制字符串$(53B8)_{16}$。

0	←	5	←	83	←	1339	←	21432	十进制
		↓		↓		↓		↓	
		5		3		B		8	十六进制

例 B.5 将十进制数字 73 234 122 转换为 base 256 的(IPv4 地址)。

解答

我们不断地将十进制值除以 256（基）直至商为 0。在每次除法中，我们将余数的值作为下一个要插入 IPv4 地址的符号。我们在点分十进制表示法中也需要插入点。结果是 IPv4 地址 4.93.118.202。

0	←	4	·	1 217	·	286 070	·	73 234 122	十进制
		·		↓		↓		↓	
		4	·	93	·	118	·	202	IPv4 地址

B.2.3 其他转换

从非十进制系统向另一个非十进制系统转换通常更简单。通过将一组 4 位转换为一个十六进制数字，我们能简单地将一个二进制数字转换为十六进制数字。我们也能够将一个十六进制数字转换为一组 4 位数字。我们给出一些例子来说明该过程。

例 B.6 将二进制数字$(1001111101)_2$转换为其十六进制等价值。

解答

我们自右边开始创建 4 位的组。然后我们使用每一组的等价十六进制数字代替该组。注意我们

需要在最后一组添加两个额外的 0。

0010	0111	1101	二进制
↓	↓	↓	
2	7	D	十六进制

结果是$(27D)_{16}$。

例 B.7 将十六进制数字$(3A2B)_{16}$转换为其二进制等价值。

解答

我们将每个十六进制数字转换为 4 位二进制等价形式。

3	A	2	B	十六进制
↓	↓	↓	↓	
0011	1010	0010	1011	二进制

结果是$(0011\ 1010\ 0010\ 1011)_2$。

例 B.8 将 IPv4 地址 112.23.78.201 转换为二进制形式。

解答

我们将每个符号转换为它的等价 8 位二进制形式。

112	.	23	.	78	.	201	IPv4 地址
↓		↓		↓		↓	
01110000		00010111		01001110		11001001	二进制

结果是$(01110000\ 00010111\ 01001110\ 11001001)_2$。

HTML、CSS、XML 和 XSL

本附录是对两种标记语言及其相应风格的简介。本附录想要为本书读者给出这些语言的高级简介。它们不是要教如何使用这些语言书写文档，这需要更详细的文本。

C.1 HTML

超文本标记语言（HyperText Markup Language，HTML）是为创建 Web 页面的标记语言。本文中，当我们提到 HTML 时，除非另有说明，我们意为 HTML 或是 XHTML。这两者之间的不同点稍后将会叙述。术语标记语言（Markup Language）来自于书籍出版业：在书籍排版和印刷之前，一个文本编辑阅读手稿，在其上做标记。这些标记告诉印刷工人怎样格式化文本。例如，如果文本编辑想要以黑体印刷一行的某一段，她在那一段下面画一条波浪线。同样地，Web 页面上的文本和其他信息由 HTML 标记，由浏览器来解释和显示。为了便于阅读，我们已经以颜色设置文档的标记章节。

C.1.1 HTML 文档

为了在浏览器中显示文档，我们需要创建一个 HTML 文档。文档应该是非格式化的文本（在标准 ASCII 中）。大部分的简单文本编辑器，如 Windows Notepad 或是 Macintosh TextEdit，创建无格式文本，但是如果我们使用文字处理软件，应该将结果保存为纯文本档案。我们将文件保存为以 html 为扩展名的文件，例如 fileName.html。然后就可以使用任意的 Web 浏览器打开该文件。

C.1.2 标签

标签是 HTML 的基本元素。标签是被隐藏的命令，它们告诉 Web 浏览器如何解释和显示 Web 页面的文本和其他内容。大部分的标签成对出现：开始标签和结束标签。

```
<tagName> ... </tagName >
```

注意标签名字是小写的，并且包含在尖括弧中；结束标签有额外的斜线。开始标签和结束标签之间的是内容。例如，和对是黑体标签：

```
<b> This text will be displayed in bold. </b>
```

某些标签不是成对出现的。例如，我们只将
标签放入到我们想要分行的地方。这样的标签称为空标签，并在标签名字的右边有一条斜线。

```
<tagName/>
```

大部分的标签有一组可选属性及其相应的值包含在开始标签中：

```
<tagName attribute = value attribute = value ... > content </tagName>
```

文档类型

文档类型声明是版本信息，出现在任意 HTML 文档第一行。它使用已知的文档类型定义（Document Type Definition，DTD），DTD 提供了 HTML 文档的标签和属性。在任意的 Web 页面中，点击右键，选择查看源文件，就会看到文档类型声明出现在源代码的第一行，看起来如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

以下解释代码中不同的节：

!DOCTYPE	文档类型声明(大写)
PUBLIC	DTD 是公共资源
W3C	DTD 的监护人
HTML 4.01:	标记语言和版本
EN	英语
http://www.w3.org/TR/html4/loose.dtd	DTD 位置的 URL

结构化的标签

头部和标题

<head>标签通常在文档类型声明之后。它后面跟着重要的文档信息。文档信息中最主要的片段之一为文档标题，出现在<title>和</title>标签之间。标题只有信息值，不被浏览器显示，但是它被显示在浏览器标题栏中。</head>标签结束文档的头部节。下面是一个例子。

```
<head>
    <title> Title of document goes here. </title>
    Other document information
    ...
</head>
```

主体

HTML 文档的实际主体在<body>和</body>标签之间。一个 HTML 文档通常将 head、title 和 body 标签组织如下：

```
<html>
    <head>
        <title> Title of document goes here. </title>
        Other document information
        ...
    </head>

    <body>
        The content of the document goes here.
        ...
    </body>
</html>
```

标题

标签<hn>和</hn>，其中 n = 1, 2, … 6,用于描述 HTML 中 6 个标题等级，其中 h1 是最大的，h6 是最小的。浏览器在结束标签后应用一个换行符。

段落

标签 用于开始一个新的段落；相应的结束 用于结束一个段落。浏览器在结束标签后应用一个换行符。

换行符

标签出现在一行的末尾时，强制换行。

中央标签

标签 和 用于将一行文本定于中心。浏览器在结束标签后应用一个换行符。

引用

<blockquote>和</blockquote>标签用于标记一块文本引用于某人或是某原文件。通常默认的，包含在这些标签中的文本，显示时左右缩进。浏览器在结束标签后应用一个换行符。

保留区

如果不通过特定的标签指定，浏览器只会保留第一个空格，并且忽略其他空白，如回车和制表符等，知道这一点是很重要的。例如，营养结构表键入如下

```
Total fat       5 g
Sodium          15 g
Protein         0 g
```

在浏览器中，会呈现出

Total fat 5g Sodium 15 g Protein 0 g

然而，我们可以使用<pre>和</pre>标签来保留文档中的白色空白（如空格、制表符和换行符）。例如我们可以通过在表前面和表后面放置保留标签来使得表呈现的样子和输入的一致。

```
<pre>
        Total fat       5 g
        Sodium          15 g
        Protein         0 g
</pre>
```

列表

我们可以定义两种类型的列表：排序的和非排序的。
和
标签用于排序的列表； 和
标签用于非排序列表。每种类型的列表中的每个项都在 和
标签中。浏览器在
标签后应用一个换行符。排序列表中的项是编号的，而非排序列表中的项通常是项目列表：

HTML text	Appearance in a web browser
`<ol>`	
`<li> CIS 20 </li>`	1. CIS 20
`<li> CIS 30 </li>`	2. CIS 30
`<li> CIS 40 </li>`	3. CIS 40
`</ol>`	

锚

HTML 的一个特性是超文本链接。HTML 文档中的连接允许用户从一个文档导航至另一个文档。<a> 和标签称为锚标签或是链接标签，用于创建到另一个 Web 页面的链接。和<a>标签一起使用的属性之一是 href（超链接引用），它的值是 URL，表明链接的目的地址。例如，下面的 HTML 行创建了指向 McGraw-Hill Publisher 的 Behrouz Forouzan Web 页面的链接。

```
<a href = "http://www.mhhe.com/forouzan" > Behrouz Forouzan </a>
```

图像

我们在文档中也要包含图像。图像标签有很多属性。下面的显示了 3 个属性。

```
<img src = "http://www..." alt = "family picture" align = middle />
```

src（源）属性给出了图像固定的位置（URL）。alt（替代）属性定义了代替图像的文本，如果由于某种原因，图像不能被显示，就显示该替代文本。align 属性定义了图像如何与文本文档相关的相对齐。图像不是直接的嵌入到文档中；使用上面的属性，浏览器发现图像，将其放置于标签所处的位置。

文本格式

粗体斜体与重点强调

最常用的文本格式标签是粗体标签和，斜体标签和。这些标签的使用减少了两个功能相近的标签的使用：strong 标签，和以及 emphasis 标签，和 。就像粗体和斜体标签一样，这些标签使得文本呈现出粗体和或是斜体，但是也给了被包含文本语义。不像粗体和斜体标签，strong 和 emphasis 标签指明了这些标签相应的字标如何由读者说出来。

小字和大字

为了将字号增大或减小一号，我们使用<big>和</big>或者<small>和</small>标签。例如，HTML 文本

```
This is <small> smaller </small> but that one is <big> bigger </big>
```

会呈现出"This is smaller but that one is bigger"。

其他格式化

一些其他的通用文本格式化标签在下表中列出。

删线 <strike>	在文本中画线
下标 <sub>	将文本移动到字符的上半部
上标 <sup>	将文本移动到字符的下半部
下划线<u>	在文本下方画线

可以将<sub>和<sup>标签和 small 标签一起使用：

```
H <sub> <small> 2 </small> </sub> O  呈现出 H2O
```

咨询标签

4 个重要的咨询标签是缩写标签（<abbr> and </abbr>），首字母缩略标签(<acronym> 和</acronym>)，定义标签 (<def> 和</def>)和引用标签 (<cite> 和</cite>)。所有的 4 个标签都有标题属性和相似的格式：

```
<tagName title = "string"> </tagName>
```

例如，下面的缩写标签显示了 DTD 是 Document Type Definition 的缩写。

```
<abbr title = "Document Type Definition"> DTD </abbr>
```

在浏览器中，当我们将光标移动到被包含的文字中（我们的例子是 DTD），标题（Document Type Definition）就显示出来（通常在工具提示中）。

嵌套

我们可以以嵌套的形式使用两个或更多的标签。例如，我们可以在粗体标签中嵌套使用斜体标签：

```
<b> <i> This text is in italic bold </i> </b>
```

务必正确嵌套。下面的就是错误的格式。

```
<b> <i> Wrong Format </b> </i>
```

嵌套的顺序可以改变文本的外观。例如，比较这两个嵌套的表达。

HTML 文本	在浏览器中的显示
First Second	***First* Second**
First Second	***First Second***

C.1.3 XHTML

可扩展超文本标记语言（Extensible HyperText Markup Language，XHTML）几乎和 HTML 4.01 是完全相同的，但是它也遵循 XML 受限制的语法。这种服从使得 XHTML 成为结构化的标记语言。使用 XHTML 标记的文档将会是"格式良好的"文档，并且，因此，浏览器理解和显示的方式是作者想要的。XHTML 的一些重要的要求是：

- 元素必须被适当地嵌套。
- 元素必须总是闭合的：标准的标签，如段落标签 和 ，必须有开始标签和结束标签；空标签，如换行符，必须写作
而不是
，br 后面的斜线指明元素的关闭。
- 元素和属性必须是小写的。
- 属性值必须带双引号。

- 文档必须有三部分：文档类型声明、头部节和主体节。

C.2　CSS

逻辑上，一个简单的 Web 文档由两层组成：内容和表现。尽管两层可以在一起，但是将它们分离增加了灵活性，减少了重复，并提高了效率。层叠样式表（Cascading Style Sheets，CSS）的创建用来将文档内容和文档表现分离。我们可以有三种方式对一个 HTML 文档的元素应用样式表：嵌入式、内部和外部。

C.2.1　嵌入式样式表

我们可以给 HTML 文档的单个元素指定样式表。例如，下面的例子使得被包含段落的字体大小为 90%，字体颜色为蓝色。

```
<p style = "font-size: 90%; color: blue" >
The size of the font is 90% and blue
</p>
```

C.2.2　内部样式表

如果我们想要指定一个样式规则应用于一个单独的 HTML 文档，我们可以将样式脚本放于<style>和</style>标签之间，该标签在 HTML 文档的头部节（如 body、h1 等等）。样式脚本的规则的一般格式为：

```
HTML content {attribute: value; attribute: value; ...}
```

注意每个属性通过冒号和其值分开。属性通过分号分开，整个的属性块放置于波形括号{}之中。例如，下面的内部样式脚本将规则应用于头部 1 和文档主体。

```
<head>
    <title> Internal Style sheet </title>
        <style type = "text / css" >
        h1{font-family: mono space; color: green}
        body {font-family: cursive; color: red}
        </style>
</head>
<body>
...
</body>
```

C.2.3　外部样式表

为了创建一个外部样式表，我们创建一个文本文档，并将那个文档中 HTML 内容每一部分的所有需求的样式规则放入文件，将文档保存为以 css 为扩展名的文件：fileName.css。下面是这样的一个文档的例子：

```
body {font-size: 10 pt; font-family: Times New Roman; color:
    black; margin-left: 12 pc; margin-right: 12 pt; line-height: 14 pt}

p {margin-left: 24 pt; margin-right: 24 pt}
h1 {font-size: 24 pt; font-family: Book Antiqua; color: red}
h2 {font-size: 22 pt; font-family: Book Antiqua; color: red}
...
h6 {font-size: 12 pc; font-family: Book Antiqua; color: red}
...
a: link {color: red}
a: visited {color: blue}
...
```

下面通过在 HTML 文档的头部节包含<link/>标签，我们将该样式脚本链接至任意的 HTML 文档：

```
<link rel = "style sheet" type = "text/css" href = "URL" />
```

rel（关系）属性说明引用文档是一个样式脚本。type 属性说明了链接的资源的 MIME 类型（text/css），href 属性给出了那个 css 文件的 URL 地址。

C.3　XML

扩展标记语言（Extensible Markup Language，XML）是一种语言，它允许用户定义数据的表现形式或是数据的结构，并指定结构中每个域的值。换言之，XML 是一种语言，它允许我们来定义标记元素（我们自己的标签和我们自己的文档结构），并且创建自定义的标记语言。唯一的限制是我们需要遵循 XML 中定义的规则。例如，下面显示了我们如何定义一个学生的记录，有三个域：name、id 和 birthday。

```
<?xml version = "1.0"?>
    <student>
        <name> George Brown </name>
        <id> 2345 </id>
        <birthday> 12 -08 - 82 </birthday>
    </student>
```

这和 C、C++或是 Java 等语言中的结构或类类似。

C.4　XSL

在 XML 文档中定义的数据和初始化的值需要另一种语言，一种样式语言来指定文档如何被显示。扩展样式语言（Extensible Style Language，XSL）是一种 XML 的样式语言，就像 CSS 是 HTML 或是 XHTML 的样式语言一样。

其他信息

D.1 端口号

表 D-1 列出在本书中我们提到的所有的端口号。

表 D-1 按端口号排序的端口

端口号	UDP 或 TCP	协议	端口号	UDP 或 TCP	协议
7	*TCP/UDP*	ECHO	70	*TCP*	GOPHER
13	*UDP/TCP*	DAYTIME	79	*TCP*	FINGER
19	*UDP/TCP*	CHARACTER GENERATOR	80	*TCP*	HTTP
20	*TCP*	FTP-DATA	110	*TCP*	POP-3
21	*TCP*	FTP-CONTROL	111	*UDP/TCP*	RPC
22	*TCP*	SSH	143	*TCP*	IMAP
23	*TCP*	TELNET	161	*UDP*	SNMP
25	*TCP*	SMTP	162	*UDP*	SNMP-TRAP
37	*UDP/TCP*	TIME	179	*TCP*	BGP
67	*UDP*	DHCP SERVER	443	*TCP*	HTTPS
68	*UDP*	DHCP-CLIENT	520	*UDP*	RIP
69	*UDP*	TFTP			

D.2 RFC

在表 D-2 中，我们列出与本书中材料直接相关的 RFC。要获取更多的信息，访问网站 http://www.rfc-editor.org。

表 D-2 每个协议的 RFC

协议	RFC	协议	RFC
ARP	826, 1029, 1166, and 1981	MPLS	3031, 3032, 3036, and 3212
BGP	1654, 1771, 1773, 1997, 2439, 2918, and 5392	Multicast Routing	1584, 1585, 2117, and 2362
DHCP	3342 and 3396	Multimedia	2198, 2250, 2326, 2475, 3240, 3550, and 3551
DNS	1034, 1035, 1996, 2535, 3008, 3658, 3755, 3757, and 3845	OSPF	1583 and 2328
Forwarding	1812, 1971, and 1980	POP3	1939
FTP	959, 2577, and 2585	RIP	1058 and 2453
HTTP	2068 and 2109	SCTP	4820, 4895, 4960, 5043, 5061, and 5062
ICMP	792, 950, 956, 957, 1016, 1122, 1256, 1305, and 1987	SMTP	2821 and 2822

（续）

协 议	RFC	协 议	RFC
IPMPv6	2461, 2894, 3122, 3810, 4443, and 4620	SNMP	3410, 3412, 3413, and 3418
IPv4 Addressing	917, 932, 936, 932, 940, 950, 1122, and 1519	SSH	4250, 4251, 4252, 4253, 4254, and 4344
IPv4	760, 781, 791, 815, 1025, 1063, 1071, 1141, 1190, 1191, 1624, and 2113	TCP	793, 813, 879, 889, 896, 1122, 1975, 1987, 1988, 1993, 2018, 2581, 3168, and 3782
IPv6	1365, 1550, 1678, 1680, 1682, 1683, 1686, 1688, 1726, 1752, 1826, 1883, 1884, 1886, 1887, 1955, 2080, 2373, 2452, 2460, 2461, 2462, 2463, 2465, 2466, 2473, 2492, 2545, and 2590	TELNET	854, 855, 856, 1041, 1091, 1372, and 1572
IPv6 Addressing	2375, 2526, 3513, 3587, 3789, and 4291	TFTP	906, 1350, 2347, 2348, and 2349
MIB	2578, 2579, and 2580	UDP	768
MIME	2046, 2047, 2048, and 2049	WWW	1614, 1630, 1737, and 1738
Mobile IP	1701, 2003, 2004, 3024, 3344, and 3775		

D.3 联系地址

表 D-3 显示了本书中我们讨论的组织的联系方式。

表 D-3 联系地址

ATM Forum Presidio of San Francisco P.O. Box 29920 572B Ruger Street San Francisco, CA 94129-0920 www.atmforum.com	**International Telecommunication Union** Place des Nations CH-1211 Geneva 20 Switzerland www.itu.int/home
Federal Communications Commission 445 12th Street S.W. Washington, DC 20554 www.fcc.gov	**Internet Corporation for Assigned Names and Numbers (ICANN)** 4676 Admiralty Way, Suite 330 Marina del Rey, CA 90292-6601 www.icann.org
Institute of Electrical and Electronics Engineers (IEEE) Operations Center 445 Hoes Lane Piscataway, NJ 08855 www.ieee.org	**Internet Engineering Task Force (IETF)** E-mail: ietf-info@ietf.org www.ietf.org
International Organization for Standardization (ISO) 1, rue de Varembé Case postale 56 CH-1211 Geneva 20 Switzerland www.iso.org	**Internet Society (ISOC)** 1775 Wiehle Avenue, Suite 201 Reston, VA 20190-5108 www.isoc.org

8B/6T 编码

本附录是 8B/6T 编码对表。8 位数据以十六进制形式表示。6T 编码使用+（正信号）、−（负信号）和 0（无信号）符号表示。

表 E-1 8B/6T 编码

数据	编码	数据	编码	数据	编码	数据	编码
00	−00+−	20	0+−00+	40	−00+0+	60	0−+0−0
01	0−−−+0	21	−00−+−	41	0−00++	61	−0+−00
02	0−+0−−	22	−−0−++	42	0−0+0+	62	−0+0−0
03	0−+−0−	23	+−0−++	43	0−0++0	63	−0+00−
04	−−0−0−	24	−−0+00	44	−00++0	64	0+−00−
05	+0−−−0	25	−+0+00	45	00−0−+	65	−+0−00
06	+0−0−+	26	−+00−0	46	00−+0+	66	−+00−0
07	+0−+0−	27	−++−−−	47	00−++0	67	−+000−
08	−+00+−	28	0+−+0−	48	00+000	68	0+1−+−
09	0−++−0	29	+0+0−−	49	++−000	69	+0++−−
0A	0−+0+−	2A	+0+−0−	4A	+−+000	6A	+0+−+−
0B	0−+−0+	2B	+0+−−0	4B	−++000	6B	+0+−−+
0C	−+0−0+	2C	0++−−0	4C	0+−000	6C	0−+−−+
0D	+0−+−0	2D	++00−−	4D	+0−000	6D	−+0+−−
0E	−+0−0−	2E	++0−0−	4E	0−+000	6E	−+0+−+
0F	−+0−0+	2F	++0−−0	4F	−0+000	6F	−+0−−+
10	0−+0+1	30	+−00−+	50	+−−+0+	70	000+−−
11	−0−0−+	31	0+−−+0	51	−+0+−+	71	000+−+
12	−0−+0+	32	0+−0−+	52	−+−+0−	72	000+−0
13	−0−+0−	33	0+−+0−	53	−+−++0	73	000+00
14	0−−+0−	34	+−0+0−	54	+−−+−0	74	000+0−
15	−−00+−	35	−0+−+0	55	−−+0+−	75	000+−0
16	−−0−0+	36	−0−0−+	56	−−+−0+	76	000−0+
17	−−0−+0	37	−0−+0−	57	−−+−−0	77	000−−0
18	−−0−+0	38	−−00+−	58	−−0+++	78	+−−−−0
19	+−0−+0	39	0−−+−0	59	−0−+++	79	+−−−0−
1A	−+−−+0	3A	0−−0+−	5A	0−−+++	7A	+−−0+−
1B	−00−+0	3B	0−−0−+	5B	0−−0++	7B	0+−0+−
1C	+00+−0	3C	+−0−0+	5C	+−−0++	7C	−00−+−
1D	−+−+−0	3D	0+−+0	5D	−00−+1	7D	−00−00
1E	+−0+−0	3E	−0+0+−	5E	0+−+−−	7E	+−−−+
1F	−+0−−0	3F	−0+−0−	5F	0−+−00	7F	+−−−00

（续）

数　码	编　码	数　据	编　码	数　据	编　码	数　据	编　码
80	−00++−	A0	−++0−0	C0	−−0+−+	E0	−+0−+
81	0−0−++	A1	+−+−00	C1	0−−−++	E1	−+−+0
82	0−0+−−	A2	+−+0−0	C2	0−−−−+	E2	−+−0−+
83	0−0++−	A3	+−−00−	C3	0−+−+−	E3	+−+−0
84	−00++−	A4	−+−00−	C4	−−0−−−	E4	−++−0−
85	00−−++	A5	+−−−00	C5	+0−−−+	E5	−+−−0
86	00−−−+	A6	+−−0−0	C6	+0−−−+	E6	−+−0−+
87	00−−+−	A7	+−−00−	C7	+0−+−	E7	−+−+0−
88	−000+0	A8	−−+−−−	C8	−−00−0	E8	−−+0+−
89	0−0+00	A9	−−++−−	C9	0−+−00	E9	−−++−0
8A	0−00−0	AA	+−+−+	CA	0−+0−0	EA	−−+0+−
8B	0−000−	AB	+−+−+	CB	0−+00+	EB	+−+−0+
8C	−0000+	AC	−+−+	CC	−−000+	EC	−−−−0+
8D	00−−00	AD	+−−+−−	CD	+0−−00	ED	−−−+−0
8E	00−0−0	AE	+−−−+−	CE	+0−0+0	EE	+−−0−−
8F	00−00+	AF	+−−−−+	CF	+0−00+	EF	+−−−0+
90	+−−−−+	B0	+000−0	D0	−−0+−+	F0	−000−+
91	−+−−−+	B1	0−0−00	D1	0+−−++	F1	0+0−+0
92	−+−−−+	B2	0−00−0	D2	0+−+−+	F2	0+00−+
93	−+−−−−	B3	0−000−	D3	0+−++−	F3	0+0+0−
94	−−−−−−	B4	+0000−	D4	−−0−+−	F4	+00+0−
95	−−+−−+	B5	00−−00	D5	−0−−++	F5	00+−+0
96	−−++−−	B6	00+0−0	D6	−0+−−+	F6	00+0−+
97	−−+++−	B7	00+00−	D7	−0+−+−	F7	00++0−
98	+−−0+0	B8	+00−+−	D8	−−00+0	F8	+000+−
99	−+−−00	B9	0+0+−−	D9	0−−+00	F9	0+0+−0
9A	−+−0+0	BA	0+0−+−	DA	0−−0−0	FA	0+00+−
9B	−+−00+	BB	0+0−−+	DB	0−−00−	FB	0+0−0+
9C	+−−00+	BC	+00−−+	DC	−−000−	FC	+00−0+
9D	−−++00	BD	00−+−−	DD	−0−+00	FD	00++−0
9E	−−+0+0	BE	00−−+−	DE	−0−0−0	FE	00+0+−
9F	−−+00+	BF	00−−−+	DF	−0−00+	FF	00+−0+

[AL 98] Albitz, P., and Liu, C. *DNS and BIND*. 3rd ed. Sebastopol, CA: O'Reilly, 1998.

[AZ 03] Agrawal, D., and Zeng, Q. *Introduction to Wireless and Mobile Systems*. Pacific Grove, CA: Brooks/Cole Thomson Learning, 2003.

[Barr et al. 05] Barrett, Daniel J., Silverman, Richard E., and Byrnes, Robert G. *SSH: The Secure Shell: The Definitive Guide*. Sebastopol, CA: O'Reilly, 2005.

[BEL 01] Bellamy, J. *Digital Telephony*. New York, NY: Wiley, 2001.

[Ber 96] Bergman, J. *Digital Baseband Transmission and Recording*. Boston, MA: Kluwer, 1996.

[Bis 03] Bishop, D. *Introduction to Cryptography with Java Applets*. Sebastopol, CA: O'Reilly, 2003.

[Bis 05] Bishop, Matt. *Introduction to Computer Security*. Reading, MA: Addison-Wesley, 2005.

[Bla 00] Black, U. *QOS in Wide Area Networks*. Upper Saddle River, NJ: Prentice Hall, 2000.

[Bla 00] Black, U. *PPP and L2TP: Remote Access Communication*. Upper Saddle River, NJ: Prentice Hall, 2000.

[BYL 09] Buford, J. F., Yu, H., and Lua, E. K. *P2P Networking and Applications*. San Francisco: Morgan Kaufmann, 2009.

[CD 08] Calvert, Kenneth L., and Donahoo, Michael J. *TCP/IP Sockets in Java*. San Francisco, CA: Morgan Kaufmann, 2008.

[CHW 99] Crowcroft, J., Handley, M., Wakeman, I. *Internetworking Multimedia*. San Francisco, CA: Morgan Kaufmann, 1999.

[Com 06] Comer, Douglas E. *Internetworking with TCP/IP*, vol. 1. Upper Saddle River, NJ: Prentice Hall, 2006.

[Cou 01] Couch, L. *Digital and Analog Communication Systems*. Upper Saddle River, NJ: Prentice Hall, 2001.

[DC 01] Donahoo, Michael J., and Calvert, Kenneth L. *TCP/IP Sockets: C version*. San Francisco, CA: Morgan Kaufmann, 2001.

[DH 03] Doraswamy, N., and Harkins, D. *IPSec*. Upper Saddle River, NJ: Prentice Hall, 2003.

[Dro 02] Drozdek A. *Elements of Data Compression*. Pacific Grove, CA: Brooks/Cole (Thomson Learning), 2002.

[Far 04] Farrel, A. *The Internet and Its Protocols*. San Francisco: Morgan Kaufmann, 2004.

[FH 98] Ferguson, P., and Huston, G. *Quality of Service*. New York: John Wiley and Sons, Inc., 1998.

[For 03] Forouzan, B. *Local Area Networks*. New York: McGraw-Hill, 2003.

[For 07] Forouzan, B. *Introduction to Data Communication and Networking*. New York: McGraw-Hill, 2007.

[For 08] Forouzan, B., *Cryptography and Network Security*. New York: McGraw-Hill, 2008.

[For 08] Forouzan, B., *TCP/IP Protocol Suite*. New York: McGraw-Hill, 2010.

[Fra 01] Frankkel, S. *Demystifying the IPSec Puzzle*. Norwood, MA: Artech House, 2001.

[GW 04] Garcia, A., and Widjaja, I. *Communication Networks*. New York, NY: McGraw-Hill, 2004.

[Gar 01] Garret, P. *Making, Breaking Codes*. Upper Saddle River, NJ: Prentice Hall, 2001.

[Gar 95] Garfinkel, S. *PGP: Pretty Good Privacy*. Sebastopol, CA: O'Reilly, 1995.

[Gas 02] Gast, M. *802.11 Wireless Networks*. Sebastopol, CA: O'Reilly, 2002.

[GGLLB 98] Gibson, J. D., Gerger, T., Lookabaugh, T., LindBerg, D., and Baker, R. L. *Digital Compression for Multimedia*. San Francisco: Morgan Kaufmann, 1998.

[Hal 01] Halsall, F. *Multimedia Communication*. Reading, MA: Addison-Wesley, 2001.

[Ham 80] Hamming, R. *Coding and Information Theory*. Upper Saddle River, NJ: Prentice Hall, 1980.

[Har 05] Harol, Elliot R. *Java Network Programming*. Sebastopol, CA: O'Reilly, 2005.

[HM 10] Havaldar, P., and Medioni, G. *Multimedia Systems: Algorithms, Standards, and Industry Practices*. Boston: Course Technology (Cengage Learning), 2010.

[Hsu 03] Hsu, H. *Analog and Digital Communications*. New York, NY: McGraw-Hill, 2003.

[Hui 00] Huitema, C. *Routing in the Internet*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2000.

[Izz 00] Izzo, P. *Gigabit Networks*. New York, NY: Wiley, 2000.

[Jam 03] Jamalipour, A. *Wireless Mobile Internet*. New York, NY: Wiley, 2003.

[Jen *et al.* 86] Jennings, D. M., Landweber, L. M., Fuchs, I. H., Farber, D. H., and Adrion, W. R. "Computer Networking for Scientists and Engineers," *Science* 231, no. 4741 (1986): 943–950.

[KCK 98] Kadambi, J., Crayford, I., and Kalkunte, M. *Gigabit Ethernet*. Upper Saddle River, NJ: Prentice Hall, 1998.

[Kei 02] Keiser, G. *Local Area Networks*. New York, NY: McGraw-Hill, 2002.

[Kes 02] Keshav, S. *An Engineering Approach to Computer Networking*. Reading, MA: Addison-Wesley, 2002.

[Kle 04] Kleinrock, L. *The Birth of the Internet*.

[KMK 04] Kumar, A., Manjunath, D., and Kuri, J. *Communication Network: An Analytical Approach*. San Francisco: Morgan Kaufmann, 2004.

[Koz 05] Kozierock, Charles M. *The TCP/IP Guide*. San Francisco: No Starch Press, 2005.

[Lei *et al.* 98] Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., Lynch, D., Postel, J., Roberts, L., and Wolff, S., *A Brief History of the Internet*. http://www.isoc.org/internet/history/brief.shtml.

[Los 04] Loshin, Pete. *IPv6: Theory, Protocol, and Practice*. San Francisco: Morgan Kaufmann, 2004.

[Mao 04] Mao, W. *Modern Cryptography*. Upper Saddle River, NJ: Prentice Hall, 2004.

[Max 99] Maxwell, K. *Residential Broadband*. New York, NY: Wiley, 1999.

[Mir 07] Mir, Nader F. *Computer and Communication Networks*. Upper Saddle River, NJ: Prentice Hall, 2007.

[Moy 98] Moy, John. *OSPF*. Reading, MA: Addison-Wesley, 1998.

[MS 01] Mauro, D., and Schmidt, K. *Essential SNMP*. Sebastopol, CA: O'Reilly, 2001.

[PD 03] Peterson, Larry L., and Davie, Bruce S. *Computer Networks*, 3rd ed. San Francisco: Morgan Kaufmann, 2003.

[Pea 92] Pearson, J. *Basic Communication Theory*. Upper Saddle River, NJ: Prentice Hall, 1992.

[Per 00] Perlman, Radia. *Interconnections*, 2nd ed. Reading, MA: Addison-Wesley, 2000.

[Pit 06] Pitt, E. *Fundamental Networking in Java*. Berlin: Springer-Verlag, 2006.

[PKA 08] Poo, D., Kiong, D., Ashok, S. *Object-Oriented Programming and Java*. Berlin: Springer-Verlag, 2008.

[Res 01] Rescorla, E. *SSL and TLS*. Reading, MA: Addison Wesley, 2001.

[Ror 96] Rorabaugh, C. *Error Coding Cookbook*. New York, NY: McGraw-Hill, 1996.

[RR 96] Robbins, Kay A., and Robbins, Steven. *Practical UNIX Programming*. Upper Saddle River, NJ: Prentice Hall, 1996.

[San 98] Sanders, S. *Gigabit Ethernet Handbook*. New York, NY: McGraw-Hill, 1998.

[Sch 03] Schiller, J. *Mobile Communications*. Reading, MA: Addison-Wesley, 2003.

[Seg 98] Segaller, S. Nerds 2.0.1: *A Brief History of the Internet*. New York: TV Books, 1998.

[Sna 00] Snader, J. C. *Effective TCP/IP Programming*. Reading, MA: Addison-Wesley, 2000.

[Spi 74] Spiegel, M. *Fourier Analysis*. New York, NY: McGraw-Hill, 1974.

[Spu 00] Spurgeon, C. *Ethernet*. Sebastopol, CA: O'Reilly, 2000.

[Sta 02] Stallings, W. *Wireless Communications and Networks*. Upper Saddle River, NJ: Prentice Hall, 2002.

[Sta 04] Stallings, W. *Data and Computer Communications*. 7th ed. Upper Saddle River, NJ: Prentice Hall, 2004.

[Sta 06] Stallings, W. *Cryptography and Network Security*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2006.

[Ste 94] Stevens, W. Richard. *TCP/IP Illustrated*, vol. 1. Reading, MA: Addison-Wesley, 1994.

[Ste 95] Stevens, W. Richard. *TCP/IP Illustrated*, vol. 2. Reading, MA: Addison-Wesley, 1995.

[Ste 99] Stewart, John W. III. *BGP4: Inter-Domain Routing in the Internet*. Reading, MA: Addison-Wesley, 1999.

[SX 02] Stewart, Randall R., and Xie, Qiaobing. *Stream Control Transmission Protocol (STCP)*. Reading, MA: Addison-Wesley, 2002.

[Ste *et al.* 04] Stevens, W. Richard, Fenner, Bill, and Rudoff, Andrew. M. *UNIX Network Programming: The Sockets Networking API*. Reading, MA: Addison-Wesley, 2004.

[Sti 06] Stinson, D. *Cryptography: Theory and Practice*. New York: Chapman & Hall/CRC, 2006.

[SW 05] Steinmetz, R., and Wehrle, K. *Peer-to-Peer Systems and Applications*. Berlin: Springer-Verlag, 2005.

[Tan 03] Tanenbaum, Andrew S. *Computer Networks*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[Tho 00] Thomas, S. *SSL and TLS Essentials*. New York: John Wiley & Sons, 2000.

[WV 00] Warland, J., and Varaiya, P. *High Performance Communication Networks*. San Francisco, CA: Morgan, Kaufmans, 2000.

[WZ 01] Wittmann, R., and Zitterbart, M. *Multicast Communication*. San Francisco: Morgan Kaufmann, 2001.

[Zar 02] Zaragoza, R. *The Art of Error Correcting Coding*. Reading, MA: Addison-Wesley, 2002.

华章精品奉献

享受智慧盛宴，品味思想精华



写给评价

"我从为本书之所以被先解说，好一无二，是源于其对相关的注重于对历史的关注。书中介绍了计算机网络的背景知识，并提供了解决不断增长的网络问题的各种方法。本书一直在不断努力以获得明确的答案和探索创新的问题域，对于致力于完善和保护互联网这样庞大的探究解决问题方法的工程师，本书提供的无疑将是宝贵的。作者对当今互联网技术的全面掌握和透彻分析是值得称赞的。"

——Vint Cerf，互联网先驱

读者评论

"本书第1版在1994年出版以来，深受读者欢迎，仅过时三个月，第1版的内容就已经比较陈旧，为适应当前发展IPv6，现在，这部世界领先的TCP/IP权威著作已经流行近十年。反映了新一代基于TCP/IP的互联技术。这本书仍保留了Stevens幽默的写作风格，简明、清晰，并且可以快速找到要点。这本书虽然近千页，但是不乏让人兴奋之处，每一章都解一个疑问或概念，复杂的TCP协议分散到各章，读起这套书的一个好方法是每章都描述了已有的和可能的攻击方法，如果你想成为已发现攻击者，并且不希望自己守门人一样潜伏习惯的攻击，这套书是非常有用。这本书是日常工作中经常和TCP/IP打交道或进行网络软件开发的人的宝书，即使你工作并不基于IP协议，这本书仍然包含很多你可以用到的方法。"

——摘自Amazon读者评论

TCP/IP详解 卷1：协议（英文版·第2版）
作者：Kevin R. Fall, W. Richard Stevens
ISBN：978-7-111-38228-7
定价：129.00元
中文版预计2012年1月出版

推荐阅读



计算机网络：系统方法（英文版·第5版）
作者：Larry L. Peterson
ISBN：978-7-111-32722-6
定价：138.00元



计算机网络（英文版·第5版）
作者：Andrew S. Tanenbaum
Davis J. Wetherall
ISBN：978-7-111-30925-8
定价：89.00元



计算机网络自顶向下方法（原书第5版）
作者：James F. Kurose
Keith W. Ross
ISBN：978-7-111-32805-6
定价：99.00元



计算机网络与因特网（原书第5版）
作者：Comer, D.E.
ISBN：978-7-111-39601-4
定价：55.00元

教师服务登记表

尊敬的老师：

您好！感谢您购买我们出版的 _____ 教材。

机械工业出版社华章公司为了进一步加强与高校教师的联系与沟通，更好地为高校教师服务，特制此表，请您填妥后发回给我们，我们将定期向您寄送华章公司最新的图书出版信息！感谢合作！

个人资料（请用正楷完整填写）

教师姓名		□先生 □女士	出生年月		职务		职称：□教授 □副教授 □讲师 □助教 □其他
学校			学院			系别	
联系电话	办公： 宅电： 移动：			联系地址及邮编			
				E-mail			
学历		毕业院校		国外进修及讲学经历			
研究领域							

主讲课程	现用教材名	作者及出版社	共同授课教师	教材满意度
课程： □专 □本 □研 人数： 学期：□春 □秋				□满意 □一般 □不满意 □希望更换
课程： □专 □本 □研 人数： 学期：□春 □秋				□满意 □一般 □不满意 □希望更换

样书申请			
已出版著作		已出版译作	
是否愿意从事翻译/著作工作 □是 □否		方向	
意见和建议			

填妥后请选择以下任何一种方式将此表返回：（如方便请赐名片）

地 址：北京市西城区百万庄南街1号 华章公司营销中心 邮编：100037

电 话：(010) 68353079 88378995 传真：(010)68995260

E-mail:hzedu@hzbook.com marketing@hzbook.com 图书详情可登录http://www.hzbook.com网站查询