

本书第3版荣获“第八届全国高校出版社优秀畅销书一等奖”

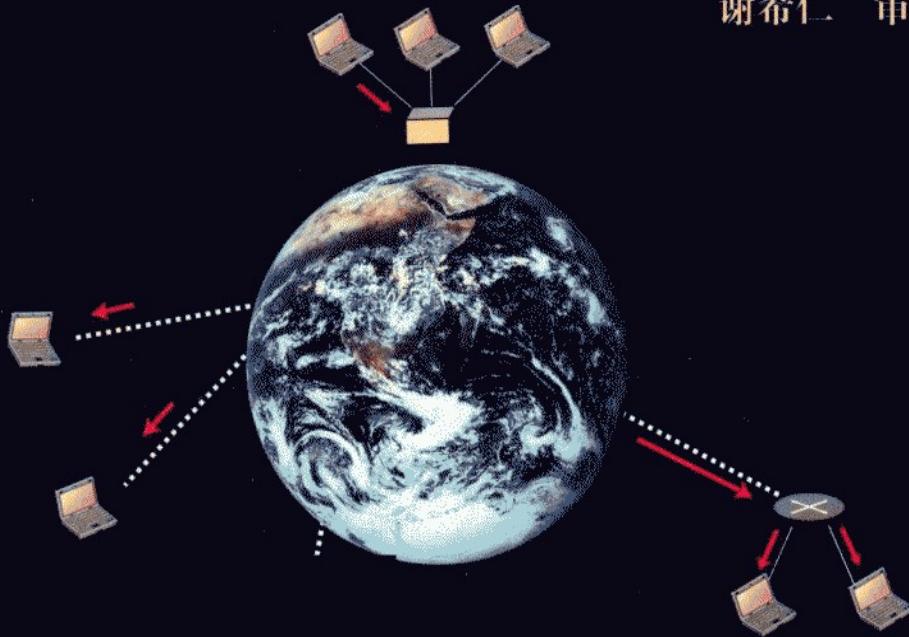
TCP/IP 协议族

(第4版)

Behrouz A. Forouzan 著

王海 张娟 朱晓阳 等译

谢希仁 审校



TCP/IP PROTOCOL SUITE

Fourth Edition

清华大学出版社



TCP/IP 协议族 (第4版)

本书是介绍TCP/IP协议族的经典图书的最新版本。本书自第1版出版以来，就广受读者欢迎。本书的第3版中文翻译版更是获得“第八届全国高校出版社优秀畅销书一等奖”。

本书最新版进行了扩充，以体现计算机网络技术的最新发展，全书含有七大部分共30章和7个附录：第一部分介绍一些基本概念和基础底层技术；第二部分介绍网络层协议；第三部分介绍运输层协议；第四部分介绍应用层协议；第五部分介绍下一代协议，即IPv6协议；第六部分介绍网络安全问题；第七部分给出了7个附录。每章的最后都有实践安排，其中的第一部分是习题，第二部分是研究活动，要求学生或读者再查找以下课外的阅读资料。

本书可作为大学生和研究生的教材，对从事计算机网络的教学和科研人员以及工程技术人员也有很好的参考价值。

世界著名计算机教材精选

- 数字图像处理：Java语言算法描述
- 逻辑设计基础（第3版）
- 计算机网络（第4版）
- TCP/IP协议族（第4版）
- 无线通信与网络（第2版）
- 密码学与网络安全（第2版）
- 网络安全基础：应用与标准（第3版）
- 编译器设计基础
- 数据结构基础（C语言版）（第2版）
- 数据结构基础（C++语言版）（第2版）
- 标准C程序设计（第4版）
- C++面向对象程序设计（第4版）
- Java面向对象程序设计（第2版）
- Java程序设计：一种跨学科的方法
- 计算理论基础（第2版）
- 数值分析与科学计算
- 数据挖掘教程
- Web数据挖掘
- 计算机图形学（OpenGL版）（第3版）
- 计算几何：算法与应用（第3版）
- Java软件结构与数据结构（第3版）
- TCP/IP协议原理与应用（第3版）
- 操作系统原理、设计与应用
- 计算群体智能基础
- 计算智能导论（第2版）
- 程序设计基础（第3版）
- MPI与Open MP并行程序设计：C语言版
- 离散数学与组合数学（第5版）
- 算法设计
- 算法基础
- 数据结构与算法分析（C++语言描述）（第2版）
- 数据结构与算法分析（Java语言描述）（第2版）
- 计算机组装和设计硬件／软件接口（第2版）
- 计算机组织与体系结构：性能设计（第7版）
- 操作系统原理
- 分布式系统原理与范型（第2版）
- 数据库管理系统原理与设计（第3版）
- 数据库系统基础教程
- 数据库设计与开发
- 面向对象系统分析与设计（第2版）
- 面向对象设计UML实践（第2版）
- 软件体系结构
- 面向对象软件工程：使用UML、模式与Java（第3版）
- 数字图像处理：原理与应用
- 程序设计语言概念（第9版）
- 规划算法

ISBN 978-7-302-23239-1



9 787302 232391 >

世界著名计算机教材精选

TCP/IP 协议族（第 4 版）

Behrouz A. Forouzan 著

王 海 张 娟 朱晓阳 等译

谢希仁 审校

清华大学出版社
北京

Behrouz A. Forouzan

TCP/IP Protocol Suite, Fourth Edition

EISBN: 0-07-337604-3

Copyrights © 2010 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 Tsinghua University Press. This editon is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2010 by McGraw-Hill Education (Asia), a division of the Singapore Branch of The McGraw-Hill Companies, Inc. and Tsinghua University 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）教育出版公司和清华大学出版社合作出版。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©2010 由麦格劳-希尔（亚洲）教育出版公司与清华大学出版社所有。

北京市版权局著作权合同登记号 图字：01-2009-4897 号

本书封面贴有 McGraw-Hill 公司防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：**010-62782989 13701121933**

图书在版编目(CIP)数据

TCP/IP 协议族：第 4 版 / (美) 福罗赞 (Forouzan, B. A.) 著；王海等译. —北京：清华大学出版社，2011.1

(世界著名计算机教材精选)

书名原文：TCP/IP Protocol Suite, Fourth Edition

ISBN 978-7-302-23239-1

I . ①T… II . ①福… ②王… III . ①计算机网络—通信协议—教材 IV . ①TP915.04

中国版本图书馆 CIP 数据核字 (2010) 第 144321 号

责任编辑：龙啟铭(longqm@163.com)

责任校对：焦丽丽

责任印制：李红英

出版发行：清华大学出版社

地 址：北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮 编：100084

社 总 机：010-62770175

邮 购：010-62786544

投稿与读者服务：010-62795954, jsjc@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：清华大学印刷厂

经 销：全国新华书店

开 本：185×260 印 张：54 字 数：1298 千字

版 次：2011 年 1 月第 1 版 印 次：2011 年 1 月第 1 次印刷

印 数：1~4000

定 价：99.00 元

产品编号：032606-01

译者序

我很高兴向各位读者推荐福罗赞教授的新书《TCP/IP 协议族（第 4 版）》。

福罗赞教授（Behrouz A. Forouzan）出生于 1944 年，目前任职于 DeAnza 学院的计算机信息系统（CIS）系，他参与了该系计算机信息系统学科的课程设计工作，同时还兼任许多公司的系统开发顾问。福罗赞教授也是一个非常高产的作家，出版了十余部涵盖计算机科学、组网、编程和数据库、安全等领域的热门书籍，其中多部著作反复再版。本书《TCP/IP 协议族》就是他的经典著作之一。

本书的最大特色是将内容庞杂，关联繁多的 TCP/IP 协议体系以有条不紊的组织方式，阐述得非常明晰，充分体现了作者深厚的功力。作为一个教师，我经常体会到，很多时候对于一个思想，理解它并不难，难的是如何能够以通俗浅显的方式准确无误地传达给学生，而福罗赞教授却非常成功地在本书中做到了。他能够化繁入简，虽是一些复杂的概念，往往以实际生活的例子为引线，逐步深入，于细微处见真功。大量的图表是本书的另一特色，对于一些抽象的概念，他往往能够应用一些图表将其具像化，使得整个内容理解起来非常轻松而同时不流于肤浅。

福罗赞教授非常勤奋，本版与第 3 版相比，不少章节几乎全部重写，且新增了很多内容，比如，增加了 MPLS 的介绍，将 IPv6 扩展为三章（第 26、27 和 28 章），将安全内容扩展为两章（第 29 和 30 章）等。在结构方案，他按照 TCP/IP 协议层次关系对章节进行了重组，从中可以看出他对如何更合理地安排这些章节内容所进行的不断探索。同时还在各章中适时补充了很多新的技术和内容，删除了一些过时的协议，还增加了一些新的习题和研究项目等。

该书如果能够与史蒂文斯（Stevens）的 *TCP/IP Illustrated* 一书搭配阅读，可起到概念相互印证，宏观与细节俱备，原理与实现兼收的多重功效。

本书可作为大学生和研究生的教材，也同样适合于自学，对于从事通信网络的教学和科研人员以及工程技术人员也有很好的参考价值。

由于译者水平有限，本书中翻译错误和不当之处在所难免，敬请读者提出宝贵意见。

序　　言

在今天的文明生活中，与网络和连网有关技术的发展变化可能是最快的。很多专家学者以及学生在审阅或学习使用了本书的第 3 版之后都提出建议，希望在本书新一版发行时能够包括这些变化。在第 4 版中，我对本书内容进行了重新整理，不仅融入了许多技术上的发展变化，而且还增加了几个新的章节和附录。

本书第 4 版假定读者并没有关于 TCP/IP 协议族的预备知识，不过读者最好还是预先学习一下数据通信的课程。

内容结构

本书分为七个部分。

- 第一部分（引言和底层技术），包括第 1~3 章，回顾了一些基本概念和基础技术。虽然这部分内容不包含在 TCP/IP 协议中，但 TCP/IP 协议需要它们的支持。
- 第二部分（网络层），包括第 4~12 章，讨论了 IPv4 编址技术、IPv4 协议、所有 IPv4 协议的辅助协议以及单播和多播路由选择协议。
- 第三部分（运输层），包括第 13~16 章，介绍了运输层的总体概念（第 13 章），然后全面讨论了三个运输层协议：UDP、TCP 和 SCTP（第 14、15、16 章）。
- 第四部分（应用层），包括第 17~25 章，介绍了应用层的总体概念，包括客户/服务器模式的编程（第 17 章），然后全面讨论了七个应用层协议（第 18~24 章）。第 25 章专门介绍因特网上的多媒体技术。
- 第五部分（新一代），包括第 26~28 章，介绍了新一代的 IP 协议、IPv6 编址技术（第 26 章）、IPv6 协议（第 27 章）和 ICMPv6（第 28 章）。
- 第六部分（安全性），包括第 29~30 章，讨论了一些不可回避的话题，如加密技术和网络安全（第 29 章）以及因特网安全（第 30 章）。
- 第七部分（附录）一共含有七个附录，在你阅读本书的过程中也许会用得着它们。

特点

为了使学生更容易地学习 TCP/IP，本书的编写具有如下一些特点。

用直观的方法

本书用图文并茂的方法讲述了技术性很强的内容，但并没有使用复杂的公式。大约超过 650 张插图与正文一起为理解这些内容提供了直观的方法。在解释网络的概念时，插图

是特别重要的，因为网络的概念是基于连接和传输的。使用插图要比使用文字更容易地理解这些概念。

突出重点

对一些重要概念反复提示，使读者可迅速找到这些重点并引起注意。

例子和应用

只要合适，我们就会用例子来阐明书中给出的概念。此外，我们在每一章都放进许多实际生活中的应用，以提高读者的兴趣。

协议软件包

虽然我们并不试图给出实现每个协议的详细代码，但在许多章还是包括了一个小节，用来讨论每个协议实现背后的大致想法。这些内容可帮助理解每一种协议的思路和相关问题，不过它们也可作为选读内容。

重要术语

在每一章的最后列出了在这一章中出现的新术语，而这些术语的定义包含在词汇表中。

小结

每一章的结尾部分是对本章内容的小结。小结以重点符号打头，列举了这章中的所有重点内容。

实践项目

每一章都包括一项实践内容，用来巩固重要概念，同时鼓励学生应用它们。实践项目由两部分组成：习题和研究活动。完成习题需要对所学内容真正理解，而研究活动则是为打算更加深入钻研这些内容而安排的。

附录

附录的作用是提供快捷的参考内容，或为理解本书中的概念而需要复习的一些内容。前几版中的附录在这里经过重新修订、整理，同时还增加了一些新的附录。

词汇表和缩写表

本书包含了一个庞大的词汇表和缩写词列表。

教师资源

习题解答、PowerPoint 幻灯片以及学生小测验，这些都可以在本书的网站 www.mhhe.com/forouzan 上找到。

第 4 版改动以及新增的内容

在第 4 版中有很多改动的地方和新增的内容，包括：

- 在每章开头部分增加了学习目标。
- 在每章结尾增加了简要参考列表和相应 RFC 的列表。
- 有几章增加了新的习题和研究活动。
- 有些插图作了修订，以更真实地反映当前实际使用的技术。
- 第 3 章（底层技术）全面改写以囊括新的技术。
- 第 4 章（网络层简介）是全新内容。
- 第 13 章（运输层简介）是全新内容。

- 第 17 章（应用层简介）是全新内容。
- 现在的第 5 章既讨论了分类编址，也讨论了无分类编址（是第 3 版中第 4 章和第 5 章的合并）。
- 第 6 章修订后包括了 MPLS。
- 新一代网际协议（IPv6）的内容扩展到 3 个章节（第 26、27、28 章）。
- 有关安全性的内容扩展到两个章节（第 29、30 章）。
- 一些过时的协议，如 RARP 和 BOOTP 被删掉以腾出空间给新的内容。
- 根据 TCP/IP 协议族中的层次关系对章节进行了重组。
- 附录 A（ASCII 码）被 Unicode 取代。
- 附录 C（差错检测）经过全面修订和扩充。
- 附录 D（检验和）经过全面修订。
- 附录 E（HTML、XHTML、XML 和 XSL）是全新的。
- 附录 F（客户/服务器模式的 Java 编程）是全新的。
- 附录 G（杂项信息）此次综合了前几版中的附录 F、附录 G 和附录 H。

怎样使用本书

本书是为学校学生和专业人员写的。它可作为感兴趣的专业的自学指导书。作为教科书，它可当作一学期的教材使用（对一年两个学期或四个学期的学制都可以）。各章的组织有很大的灵活性。下面是我给出的一些建议：

- 如果学生已经学过数据通信和联网的课程，则可跳过第 1~3 章。
- 第 4~25 章对了解 TCP/IP 是至关重要的。
- 第 26~28 章的决定权在于教授是否认为应该让学生熟悉新一代的 IP。
- 第 29 章和第 30 章为学生学习安全性课程提供预备知识，如果时间有限可以跳过。

致谢

很显然，写这样篇幅的书没有很多人的帮助是不可能的。在前 3 版的序言中，我对许多人的贡献表示了感谢。在此第 4 版中，我想对以下同行审阅人对编写本书的贡献表示感谢，他们包括：

Dale Buchholz, *DePaul University*

Victor Clincy, *Kennesaw State University*

Richard Coppins, *Virginia Commonwealth University*

Zongming Fei, *University of Kentucky*

Guy Hembroff, *Michigan Tech University*

Frank Lin, *San Jose State University*

Tim Lin, *California Polytechnic University-Pomona*

Abdallah Shami, *University of Western Ontario*

Elsa Valeroso, *Eastern Michigan University*

Mark Weiser, *Oklahoma State University*

Ben Zhao, *University of California at Santa Barbara*

我还要感谢 Paul D. Amer 教授, 因为他对初稿提出了许多反馈意见和建议, 对本书做出了宝贵的贡献。

特别要感谢麦格劳-希尔(McGraw-Hill)公司的工作人员。发行人 Raghu Srinivasan 证明了熟练的发行人可以把不可能的事情变为可能的。每当我需要帮助时, 开发编辑 Melinda Bilecki 就会给予帮助。项目经理 Joyce Watters 在出版的过程中一直以极大的热情领导着我们。我还要感谢 Macmillan Publishing Solutions 公司的 Les Chappell 在制作上, Laurie Janssen 在设计上以及 George F. Watson 在原稿编辑上所做的贡献。

Behrouz A. Forouzan

January, 2009

商标

本书中经常会出现一些商标名称。书中并没有在每次提到这些商标名称时都插入一个商标符号, 而是由我在这里确认使用这些商标, 并在此声明绝无对它们进行侵权的意图。其他的产品名称、商标、注册商标等也都是其拥有者的财产。

- Network File System 和 NFS 是 Sun Microsystems 公司的注册商标。
- UNIX 是 UNIX 系统实验室公司 (是诺威公司完全拥有的子公司) 的注册商标。
- Xerox 是商标, 而 Ethernet 是 Xerox 公司的注册商标。

目 录

第一部分 引言和底层技术

第1章 引言	3
1.1 发展简史	3
1.1.1 ARPANET	3
1.1.2 因特网的诞生	4
1.1.3 传输控制协议/网际协议（TCP/IP）	4
1.1.4 MILNET	4
1.1.5 CSNET	5
1.1.6 NSFNET	5
1.1.7 ANSNET	5
1.1.8 今日的因特网	5
1.1.9 主干 ISP	6
1.1.10 地区 ISP	6
1.1.11 本地 ISP	6
1.1.12 大事记	7
1.1.13 因特网的发展	7
1.2 协议和标准	7
1.2.1 协议	7
1.2.2 标准	8
1.3 标准化组织	8
1.3.1 标准创建委员会	9
1.3.2 论坛	10
1.3.3 管理机构	10
1.4 因特网标准	10
1.4.1 成熟度	11
1.4.2 需求级别	11
1.5 因特网的管理机构	12
1.5.1 因特网协会（ISOC）	13
1.5.2 因特网体系结构研究委员会（IAB）	13
1.5.3 因特网工程部（IETF）	13

1.5.4 因特网研究部 (IRTF)	13
1.5.5 因特网赋号管理局和因特网名字与号码指派公司	14
1.5.6 网络信息中心 (NIC)	14
1.6 深入阅读	14
1.6.1 书籍和论文	14
1.6.2 网站	14
1.7 重要术语	14
1.8 本章小结	15
1.9 实践安排	16
1.9.1 习题	16
1.9.2 研究活动	16
第2章 OSI模型和TCP/IP协议族	17
2.1 协议分层	17
2.1.1 分层结构	18
2.1.2 服务	18
2.2 OSI模型	19
2.2.1 分层的体系结构	19
2.2.2 层与层之间的通信	20
2.2.3 封装	21
2.2.4 OSI模型中的各层	21
2.2.5 OSI各层小结	24
2.3 TCP/IP协议族	25
2.3.1 OSI和TCP/IP协议族的比较	25
2.3.2 TCP/IP协议族的分层	26
2.4 编址	30
2.4.1 物理地址	31
2.4.2 逻辑地址	32
2.5 深入阅读	35
2.5.1 参考书	35
2.5.2 RFC	35
2.6 重要术语	35
2.7 本章小结	36
2.8 实践安排	36
2.8.1 习题	36
2.8.2 研究活动	38
第3章 底层技术	39
3.1 有线局域网	39
3.1.1 IEEE标准	40
3.1.2 帧格式	40

3.1.3 编址	42
3.1.4 以太网的发展历程	43
3.1.5 标准以太网	43
3.1.6 快速以太网	47
3.1.7 吉比特以太网	48
3.1.8 10G 以太网	49
3.2 无线局域网	50
3.2.1 IEEE 802.11	50
3.2.2 编址机制	55
3.2.3 蓝牙	57
3.3 点到点广域网	59
3.3.1 56K 调制解调器	59
3.3.2 DSL 技术	60
3.3.3 电缆调制解调器	61
3.3.4 T 线	63
3.3.5 SONET	64
3.3.6 PPP	64
3.4 交换广域网	65
3.4.1 X.25	65
3.4.2 帧中继	66
3.4.3 ATM	66
3.5 连接设备	70
3.5.1 转发器	70
3.5.2 网桥	71
3.5.3 路由器	73
3.6 深入阅读	74
3.7 重要术语	74
3.8 本章小结	75
3.9 实践安排	76
3.9.1 习题	76
3.9.2 研究活动	77
第二部分 网 络 层	
第 4 章 网络层简介	81
4.1 简介	81
4.2 交换	82
4.2.1 电路交换	82
4.2.2 分组交换	83
4.3 网络层的分组交换	83

4.3.1 无连接服务	83
4.3.2 面向连接的服务	85
4.4 网络层的服务	88
4.4.1 一个例子	89
4.4.2 逻辑编址	90
4.4.3 源计算机提供的服务	90
4.4.4 各路由器提供的服务	91
4.4.5 目的计算机提供的服务	92
4.5 其他与网络层相关的问题	93
4.5.1 差错控制	93
4.5.2 流量控制	94
4.5.3 拥塞控制	94
4.6 进一步阅读	96
4.7 重要术语	96
4.8 本章小结	96
4.9 实践安排	97
4.9.1 习题	97
第5章 IPv4地址	98
5.1 引言	98
5.1.1 地址空间	99
5.1.2 记法	99
5.1.3 地址段	101
5.1.4 运算	101
5.2 分类编址	104
5.2.1 分类	104
5.2.2 地址类和地址块	106
5.2.3 两级编址	107
5.2.4 一个例子	110
5.2.5 三级编址：子网划分	112
5.2.6 构造超网	114
5.3 无分类编址	115
5.3.1 可变长度地址块	116
5.3.2 两级编址	116
5.3.3 地址块的分配	120
5.3.4 子网划分	121
5.4 特殊地址	125
5.4.1 特殊地址块	125
5.4.2 每个地址块中的特殊地址	126
5.5 NAT	127

5.5.1 地址转换	127
5.5.2 转换表	128
5.6 深入阅读	129
5.6.1 参考书	129
5.6.2 RFC	130
5.7 重要术语	130
5.8 本章小结	130
5.9 实践安排	131
5.9.1 习题	131
第 6 章 IP 分组的交付和转发	135
6.1 交付	135
6.1.1 直接交付	135
6.1.2 间接交付	136
6.2 转发	136
6.2.1 基于目的地址的转发	136
6.2.2 基于标记的转发	148
6.3 路由器的结构	150
6.3.1 构件	150
6.4 深入阅读	153
6.4.1 参考书	153
6.4.2 RFC	153
6.5 重要术语	153
6.6 本章小结	153
6.7 实践安排	154
6.7.1 习题	154
6.7.2 研究活动	155
第 7 章 网际协议版本 4 (IPv4)	156
7.1 引言	156
7.2 数据报	157
7.3 分片	161
7.3.1 最大传送单元 (MTU)	161
7.3.2 与分片有关的字段	162
7.4 选项	165
7.4.1 格式	165
7.4.2 选项类型	166
7.5 检验和	171
7.5.1 在发送端计算检验和	172
7.5.2 在接收端计算检验和	172
7.5.3 IP 分组中的检验和	173

7.6 IP 在 ATM 上运行	174
7.6.1 ATM 广域网	174
7.6.2 信元的路由选择	175
7.7 安全性	176
7.7.1 安全问题	176
7.7.2 IPSec	176
7.8 IP 软件包	177
7.8.1 首部添加模块	178
7.8.2 处理模块	178
7.8.3 队列	179
7.8.4 路由表	179
7.8.5 转发模块	179
7.8.6 MTU 表	179
7.8.7 分片模块	180
7.8.8 重装表	181
7.8.9 重装模块	181
7.9 深入阅读	182
7.9.1 参考书	182
7.9.2 RFC	182
7.10 重要术语	182
7.11 本章小结	183
7.12 实践安排	184
7.12.1 习题	184
7.12.2 研究活动	185
第 8 章 地址解析协议 (ARP)	186
8.1 地址映射	186
8.1.1 静态映射	187
8.1.2 动态映射	187
8.2 ARP 协议	187
8.2.1 分组格式	189
8.2.2 封装	190
8.2.3 操作	190
8.2.4 代理 ARP	192
8.3 ATMARP	193
8.3.1 分组格式	193
8.3.2 ATMARP 的操作	194
8.3.3 逻辑 IP 子网 (LIS)	197
8.4 ARP 软件包	197
8.4.1 高速缓存表	198

8.4.2 队列	199
8.4.3 输出模块	199
8.4.4 输入模块	200
8.4.5 高速缓存控制模块	201
8.4.6 更多的例子	203
8.5 深入阅读	204
8.5.1 参考书	204
8.5.2 RFC	205
8.6 重要术语	205
8.7 本章小结	205
8.8 实践安排	206
8.8.1 习题	206
第 9 章 网际控制报文协议 (ICMP)	207
9.1 引言	207
9.2 报文	208
9.2.1 报文格式	208
9.2.2 差错报告报文	209
9.2.3 查询	214
9.2.4 检验和	216
9.3 排错工具	217
9.3.1 ping	217
9.3.2 traceroute	219
9.4 ICMP 软件包	221
9.4.1 输入模块	222
9.4.2 输出模块	222
9.5 深入阅读	223
9.5.1 参考书	223
9.5.2 RFC	224
9.6 重要术语	224
9.7 本章小结	224
9.8 实践安排	224
9.8.1 习题	224
9.8.2 研究活动	226
第 10 章 移动 IP	227
10.1 编址	227
10.1.1 固定主机	227
10.1.2 移动主机	228
10.2 代理	228
10.2.1 归属代理	229

10.2.2 外地代理	229
10.3 三个阶段	229
10.3.1 代理发现	230
10.3.2 登记	231
10.3.3 数据传送	233
10.4 移动 IP 的低效率	234
10.4.1 两次穿越	234
10.4.2 三角路由选择	235
10.4.3 解决方法	235
10.5 深入阅读	235
10.5.1 参考书	235
10.5.2 RFC	236
10.6 重要术语	236
10.7 本章小结	236
10.8 实践安排	236
10.8.1 习题	236
10.8.2 研究活动	237
第 11 章 单播路由选择协议（RIP、OSPF 和 BGP）	238
11.1 引言	238
11.1.1 代价或度量	239
11.1.2 静态路由表还是动态路由表	239
11.1.3 路由选择协议	239
11.2 域内和域间路由选择	239
11.3 距离向量路由选择	240
11.3.1 Bellman-Ford 算法	241
11.3.2 距离向量路由选择算法	242
11.3.3 计数到无穷大	246
11.4 RIP	248
11.4.1 RIP 的报文格式	249
11.4.2 请求和响应	250
11.4.3 RIP 的计时器	251
11.4.4 RIP 版本 2	252
11.4.5 封装	253
11.5 链路状态路由选择	253
11.5.1 构造路由表	254
11.6 OSPF	257
11.6.1 区域	257
11.6.2 度量	258
11.6.3 链路的类型	258

11.6.4 图形表示法	260
11.6.5 OSPF 分组	260
11.6.6 链路状态更新分组	261
11.6.7 其他分组	268
11.6.8 封装	270
11.7 路径向量路由选择	270
11.7.1 可达性	271
11.7.2 路由表	272
11.8 BGP	273
11.8.1 自治系统的类型	273
11.8.2 路径属性	274
11.8.3 BGP 会话	274
11.8.4 外部 BGP 和内部 BGP	274
11.8.5 分组的类型	275
11.8.6 分组格式	275
11.8.7 封装	278
11.9 深入阅读	278
11.9.1 参考书	278
11.9.2 RFC	278
11.10 重要术语	279
11.11 本章小结	279
11.12 实践安排	280
11.12.1 习题	280
11.12.2 研究活动	282
第 12 章 多播和多播路由选择协议	283
12.1 引言	283
12.1.1 单播	284
12.1.2 多播	284
12.1.3 广播	286
12.2 多播地址	286
12.2.1 IPv4 中的多播地址	286
12.2.2 选择多播地址	289
12.2.3 数据链路层多播分组的交付	289
12.3 IGMP	291
12.3.1 组管理	291
12.3.2 IGMP 报文	292
12.3.3 在主机上应用 IGMP 协议	294
12.3.4 IGMP 协议应用于路由器	297
12.3.5 IGMP 在转发中的作用	299

12.3.6 变量和计时器	300
12.3.7 封装	301
12.3.8 与老版本之间的兼容	301
12.4 多播路由选择	301
12.4.1 最佳路由选择：最短路径树	302
12.5 路由选择协议	304
12.5.1 多播链路状态路由选择：MOSPF	304
12.5.2 多播距离向量路由选择	305
12.5.3 DVMRP	309
12.5.4 CBT	309
12.5.5 PIM	311
12.6 MBONE	311
12.7 深入阅读	312
12.7.1 参考书	312
12.7.2 RFC	312
12.8 重要术语	313
12.9 本章小结	313
12.10 实践安排	313
12.10.1 习题	313
12.10.2 研究活动	315

第三部分 运 输 层

第 13 章 运输层简介	319
13.1 运输层服务	319
13.1.1 进程到进程的通信	319
13.1.2 编址：端口号	320
13.1.3 封装和解封	322
13.1.4 复用和分用	323
13.1.5 流量控制	323
13.1.6 差错控制	325
13.1.7 流量控制和差错控制的组合	326
13.1.8 拥塞控制	327
13.1.9 无连接的和面向连接的服务	328
13.2 运输层协议	331
13.2.1 简单协议	331
13.2.2 停止等待协议	333
13.2.3 返回 N 协议	336
13.2.4 选择重传协议	342
13.2.5 双向协议：捎带	347

13.3	深入阅读	348
13.4	重要术语	348
13.5	本章小结	349
13.6	实践安排	349
13.6.1	习题	349
13.6.2	研究活动	351
第 14 章	用户数据报协议 (UDP)	352
14.1	引言	352
14.2	用户数据报	353
14.3	UDP 服务	354
14.3.1	进程到进程的通信	355
14.3.2	无连接服务	355
14.3.3	流量控制	355
14.3.4	差错控制	356
14.3.5	拥塞控制	357
14.3.6	封装和解封	357
14.3.7	排队	358
14.3.8	复用和分用	359
14.3.9	UDP 与简单协议的比较	360
14.4	UDP 的应用	360
14.4.1	UDP 的特点	360
14.4.2	典型应用	362
14.5	UDP 软件包	362
14.5.1	控制块表	362
14.5.2	输入队列	362
14.5.3	控制块模块	363
14.5.4	输入模块	363
14.5.5	输出模块	364
14.5.6	举例	364
14.6	深入阅读	366
14.6.1	参考书	366
14.6.2	RFC	366
14.7	重要术语	366
14.8	本章小结	366
14.9	实践安排	367
14.9.1	习题	367
第 15 章	传输控制协议 (TCP)	368
15.1	TCP 服务	368
15.1.1	进程到进程的通信	368

15.1.2 流交付服务	369
15.1.3 全双工通信	371
15.1.4 复用和分用	371
15.1.5 面向连接的服务	371
15.1.6 可靠的服务	372
15.2 TCP 的特点	372
15.2.1 编号系统	372
15.2.2 流量控制	373
15.2.3 差错控制	373
15.2.4 拥塞控制	374
15.3 报文段	374
15.3.1 格式	374
15.3.2 封装	376
15.4 TCP 连接	376
15.4.1 连接建立	376
15.4.2 数据传送	378
15.4.3 连接终止	380
15.4.4 连接复位	382
15.5 状态转换图	382
15.5.1 几种情况	384
15.6 TCP 中的窗口	390
15.6.1 发送窗口	390
15.6.2 接收窗口	391
15.7 流量控制	391
15.7.1 打开和关闭窗口	392
15.7.2 窗口的收缩	394
15.7.3 糊涂窗口综合征	395
15.8 差错控制	396
15.8.1 检验和	397
15.8.2 确认	397
15.8.3 重传	398
15.8.4 失序的报文段	398
15.8.5 TCP 数据传送的 FSM	399
15.8.6 几种情况	400
15.9 拥塞控制	404
15.9.1 拥塞窗口	404
15.9.2 拥塞策略	404
15.10 TCP 的计时器	408
15.10.1 重传计时器	408

15.10.2 持续计时器	411
15.10.3 保活计时器	411
15.10.4 TIME-WAIT 计时器	411
15.11 选项	411
15.12 TCP 软件包	418
15.12.1 传输控制块 (TCB)	418
15.12.2 计时器	419
15.12.3 主模块	419
15.12.4 输入处理模块	423
15.12.5 输出处理模块	424
15.13 深入阅读	424
15.13.1 参考书	424
15.13.2 RFC	424
15.14 重要术语	424
15.15 本章小结	425
15.16 实践安排	426
15.16.1 习题	426
15.16.2 研究活动	429
第 16 章 流控制传输协议 (SCTP)	430
16.1 引言	430
16.2 SCTP 的服务	431
16.2.1 进程到进程的通信	431
16.2.2 多重流	432
16.2.3 多重归属	432
16.2.4 全双工通信	433
16.2.5 面向连接的服务	433
16.2.6 可靠的服务	433
16.3 SCTP 的特点	433
16.3.1 传输序号 (TSN)	434
16.3.2 流标识符 (SI)	434
16.3.3 流序号 (SSN)	434
16.3.4 分组	434
16.3.5 确认号	436
16.3.6 流量控制	436
16.3.7 差错控制	437
16.3.8 拥塞控制	437
16.4 分组格式	437
16.4.1 通用首部	437
16.4.2 块 (chunk)	438

16.5	SCTP 关联	444
16.5.1	关联建立	445
16.5.2	数据传送	447
16.5.3	关联终止	449
16.5.4	关联异常终止	449
16.6	状态转换图	450
16.6.1	几种情况	451
16.6.2	其他情况	453
16.7	流量控制	454
16.7.1	接收方	454
16.7.2	发送方	454
16.7.3	一种情况	455
16.8	差错控制	456
16.8.1	接收方	456
16.8.2	发送方	457
16.8.3	发送数据块	458
16.8.4	生成 SACK 块	458
16.9	拥塞控制	459
16.9.1	拥塞控制和多归属	459
16.9.2	显式拥塞通知	459
16.10	深入阅读	459
16.10.1	参考书	460
16.10.2	RFC	460
16.11	重要术语	460
16.12	本章小结	460
16.13	实践安排	461
16.13.1	习题	461
16.13.2	研究活动	463

第四部分 应用 层

第 17 章	应用层简介	467
17.1	客户-服务器范式	467
17.1.1	服务器	468
17.1.2	客户	468
17.1.3	并发	468
17.1.4	套接字接口	470
17.1.5	使用 UDP 的通信	477
17.1.6	使用 TCP 的通信	480
17.1.7	预先定义的客户-服务器应用	486

17.2	P2P 范式	486
17.3	深入阅读	487
17.4	重要术语	487
17.5	本章小结	487
17.6	实践安排	488
17.6.1	习题	488
第 18 章	主机配置：DHCP	489
18.1	引言	489
18.1.1	曾经使用过的协议	490
18.2	DHCP 操作	490
18.2.1	同一个网络	491
18.2.2	不同的网络	491
18.2.3	UDP 端口	492
18.2.4	使用 TFTP	493
18.2.5	差错控制	493
18.2.6	分组格式	493
18.3	配置	495
18.3.1	静态地址分配	495
18.3.2	动态地址分配	496
18.3.3	转换状态	496
18.3.4	其他	497
18.3.5	交换报文	498
18.4	深入阅读	498
18.4.1	参考书和 RFC	499
18.5	重要术语	499
18.6	本章小结	499
18.7	实践安排	499
18.7.1	习题	499
18.7.2	研究活动	500
第 19 章	域名系统（DNS）	501
19.1	DNS 的必要性	501
19.2	名字空间	502
19.2.1	平面名字空间	503
19.2.2	层次名字空间	503
19.2.3	域名空间	503
19.2.4	域	505
19.2.5	域名空间的分布	505
19.3	因特网中的 DNS	507
19.3.1	类属域	507

19.3.2 国家域	508
19.3.3 反向域	508
19.4 解析	509
19.4.1 解析程序	509
19.4.2 名字到地址的映射	510
19.4.3 地址到名字的映射	510
19.4.4 递归解析	510
19.4.5 迭代解析	511
19.4.6 高速缓存	511
19.5 DNS 报文	512
19.5.1 首部	512
19.6 记录的类型	514
19.6.1 问题记录	514
19.6.2 资源记录	515
19.7 压缩	516
19.8 封装	519
19.9 注册机构	519
19.10 DDNS	520
19.11 DNS 的安全性	520
19.12 深入阅读	521
19.12.1 参考书	521
19.12.2 RFC	521
19.13 重要术语	521
19.14 本章小结	521
19.15 实践安排	522
19.15.1 习题	522
19.15.2 研究活动	523
第 20 章 远程登录：TELNET 与 SSH	525
20.1 TELNET	525
20.1.1 概念	525
20.1.2 分时的环境	526
20.1.3 网络虚拟终端（NVT）	527
20.1.4 嵌入	528
20.1.5 选项	529
20.1.6 对称性	532
20.1.7 子选项协商	532
20.1.8 对服务器进行控制	532
20.1.9 带外信令	533
20.1.10 转义字符	534

20.1.11 操作方式	535
20.1.12 用户接口	536
20.1.13 安全问题	537
20.2 SSH	537
20.2.1 版本	537
20.2.2 组成	537
20.2.3 端口转发	538
20.2.4 SSH 分组格式	539
20.3 深入阅读	539
20.3.1 参考书	539
20.3.2 RFC	539
20.4 重要术语	540
20.5 本章小结	540
20.6 实践安排	541
20.6.1 习题	541
20.6.2 研究活动	541
第 21 章 文件传送: FTP 和 TFTP	543
21.1 文件传送协议 (FTP)	543
21.1.1 连接	544
21.1.2 通信	545
21.1.3 命令处理	547
21.1.4 文件传送	550
21.1.5 匿名 FTP	553
21.2 简单文件传送协议 (TFTP)	554
21.2.1 报文	555
21.2.2 连接	557
21.2.3 数据传送	557
21.2.4 UDP 端口	559
21.2.5 TFTP 举例	560
21.2.6 TFTP 选项	561
21.2.7 安全性	561
21.2.8 应用	561
21.3 深入阅读	561
21.3.1 参考书	562
21.3.2 RFC	562
21.4 重要术语	562
21.5 本章小结	562
21.6 实践安排	563
21.6.1 习题	563

21.6.2	研究活动	564
第 22 章	万维网和 HTTP	565
22.1	体系结构	565
22.1.1	超文本和超媒体	566
22.1.2	Web 客户 (浏览器)	567
22.1.3	Web 服务器	567
22.1.4	统一资源定位符 (URL)	567
22.2	Web 文档	568
22.2.1	静态文档	568
22.2.2	动态文档	568
22.2.3	活动文档	570
22.3	HTTP	571
22.3.1	HTTP 事务	571
22.3.2	有条件请求	577
22.3.3	持续连接	577
22.3.4	Cookie	579
22.3.5	Web 缓存: 代理服务器	581
22.3.6	HTTP 的安全	581
22.4	深入阅读	582
22.4.1	参考书	582
22.4.2	RFC	582
22.5	重要术语	582
22.6	本章小结	583
22.7	实践安排	583
22.7.1	习题	583
22.7.2	研究活动	584
第 23 章	电子邮件: SMTP、POP、IMAP 和 MIME	585
23.1	体系结构	585
23.1.1	第一种情况	586
23.1.2	第二种情况	586
23.1.3	第三种情况	587
23.1.4	第四种情况	587
23.2	用户代理	588
23.2.1	用户代理提供的服务	589
23.2.2	用户代理类型	589
23.2.3	发送邮件	589
23.2.4	接收邮件	590
23.2.5	地址	590
23.2.6	发件清单或分组清单	590

23.3 报文传送代理：SMTP.....	590
23.3.1 命令和响应.....	591
23.3.2 邮件传送阶段.....	594
23.4 报文读取代理：POP 和 IMAP.....	596
23.4.1 POP3.....	597
23.4.2 IMAP4.....	597
23.5 MIME	598
23.5.1 MIME 首部	598
23.6 基于万维网的邮件	602
23.6.1 案例一	602
23.6.2 案例二	603
23.7 电子邮件的安全性	603
23.8 深入阅读	604
23.8.1 参考书	604
23.8.2 RFC	604
23.9 重要术语	604
23.10 本章小结	604
23.11 实践安排	605
23.11.1 习题	605
23.11.2 研究活动	606
第 24 章 网络管理 (SNMP)	607
24.1 概念	607
24.1.1 管理器和代理	608
24.2 管理构件	608
24.2.1 SNMP 的作用	609
24.2.2 SMI 的作用	609
24.2.3 MIB 的作用	609
24.2.4 类比	609
24.2.5 概览	610
24.3 SMI	611
24.3.1 名字	611
24.3.2 类型	612
24.3.3 编码方法	613
24.4 MIB	615
24.4.1 访问 MIB 变量	615
24.4.2 字典式排序	618
24.5 SNMP	618
24.5.1 PDU	618
24.5.2 格式	620

24.5.3 报文	621
24.6 UDP 端口	623
24.7 安全	624
24.8 深入阅读	624
24.8.1 参考书	624
24.8.2 RFC	624
24.9 重要术语	625
24.10 本章小结	625
24.11 实践安排	625
24.11.1 习题	625
24.11.2 研究活动	626
第 25 章 多媒体	627
25.1 引言	627
25.2 数字化音频和视频	628
25.2.1 数字化音频	628
25.2.2 数字化视频	628
25.3 音频和视频压缩	629
25.3.1 音频压缩	629
25.3.2 视频压缩	630
25.4 流式存储音频/视频	633
25.4.1 第一种方法：使用万维网服务器	633
25.4.2 第二种方法：使用具有元文件的万维网服务器	633
25.4.3 第三种方法：使用媒体服务器	634
25.4.4 第四种方法：使用媒体服务器和 RTSP	634
25.5 流式直播音频/视频	635
25.6 实时交互式音频/视频	636
25.6.1 特性	636
25.7 RTP	639
25.7.1 RTP 分组格式	640
25.7.2 UDP 端口	641
25.8 RTCP	641
25.8.1 发送方报告	641
25.8.2 接收方报告	642
25.8.3 源点描述报文	642
25.8.4 再见报文	642
25.8.5 特定应用报文	642
25.8.6 UDP 端口	642
25.9 IP 话音	642
25.9.1 SIP	642

25.9.2 H.323	644
25.10 服务质量	646
25.10.1 流的特性	646
25.10.2 流的分类	647
25.10.3 提高 QoS 的技术	647
25.10.4 资源预留	650
25.10.5 许可控制	650
25.11 综合服务	650
25.11.1 信令	651
25.11.2 流规范	651
25.11.3 许可	651
25.11.4 服务类别	651
25.11.5 RSVP	652
25.11.6 综合服务存在的问题	653
25.12 区分服务	654
25.13 深入阅读	655
25.13.1 参考书	655
25.13.2 RFC	656
25.14 重要术语	656
25.15 本章小结	656
25.16 实践安排	657
25.16.1 习题	657

第五部分 下一代

第 26 章 IPv6 编址	661
26.1 引言	661
26.1.1 记法	661
26.1.2 地址空间	664
26.1.3 三种地址类型	664
26.1.4 广播和多播	665
26.2 地址空间分配	665
26.2.1 指派的和保留的地址块	667
26.3 全球单播地址	670
26.3.1 三级结构	670
26.4 自动配置	672
26.5 重新编号	673
26.6 深入阅读	673
26.6.1 参考书	673
26.6.2 RFC	673

26.7	重要术语	673
26.8	本章小结	674
26.9	实践安排	674
26.9.1	习题	674
第 27 章	IPv6 协议	676
27.1	引言	676
27.1.1	改变的缘由	676
27.1.2	采用进度延缓的原因	677
27.2	分组格式	677
27.2.1	基本首部	677
27.2.2	流标号	678
27.2.3	IPv4 首部和 IPv6 首部的比较	679
27.2.4	扩展首部	680
27.2.5	IPv4 和 IPv6 的比较	684
27.3	从 IPv4 过渡到 IPv6	684
27.3.1	双协议栈	684
27.3.2	隧道技术	685
27.3.3	首部转换	685
27.4	深入阅读	686
27.4.1	参考书	686
27.4.2	RFC	686
27.5	重要术语	686
27.6	本章小结	686
27.7	实践安排	687
27.7.1	习题	687
27.7.2	研究活动	687
第 28 章	ICMPv6	688
28.1	引言	688
28.2	差错报文	689
28.2.1	终点不可达报文	689
28.2.2	分组太大报文	690
28.2.3	超时报文	690
28.2.4	参数问题报文	691
28.3	信息报文	691
28.3.1	回送请求报文	691
28.3.2	回送回答报文	692
28.4	邻站发现报文	692
28.4.1	路由器询问报文	692
28.4.2	路由器通告报文	692

28.4.3 邻站询问报文	693
28.4.4 邻站通告报文	694
28.4.5 改变路由报文	694
28.4.6 反向邻站询问报文	695
28.4.7 反向邻站通告报文	695
28.5 组成员关系报文	695
28.5.1 成员关系查询报文	696
28.5.2 成员关系报告报文	696
28.5.3 功能性	696
28.6 深入阅读	698
28.6.1 参考书	698
28.6.2 RFC	698
28.7 重要术语	698
28.8 本章小结	698
28.9 实践安排	699
28.9.1 习题	699
28.9.2 研究活动	699

第六部分 安 全 性

第 29 章 加密术和网络安全	703
29.1 引言	703
29.1.1 安全的目标	704
29.1.2 攻击	704
29.1.3 服务	706
29.1.4 技术	706
29.2 传统加密方法	706
29.2.1 密钥	707
29.2.2 替代加密方法	708
29.2.3 置换加密方法	710
29.2.4 流和块加密方法	711
29.3 现代加密方法	711
29.3.1 现代块加密方法	711
29.3.2 数据加密标准 (DES)	713
29.3.3 现代流加密方法	714
29.4 不对称密钥加密方法	715
29.4.1 密钥	716
29.4.2 总体思想	716
29.4.3 RSA 加密系统	718
29.4.4 应用	720

29.5	报文完整性	720
29.5.1	报文和报文摘要	720
29.5.2	散列函数	721
29.6	报文鉴别	721
29.6.1	HMAC	722
29.7	数字签名	722
29.7.1	比较	723
29.7.2	过程	723
29.7.3	对摘要的签名	724
29.7.4	服务	725
29.7.5	RSA 数字签名机制	726
29.7.6	数字签名标准	727
29.8	实体鉴别	727
29.8.1	实体鉴别和报文鉴别的比较	727
29.8.2	验证类别	727
29.8.3	口令	728
29.8.4	查问-响应	728
29.9	密钥管理	729
29.9.1	对称密钥的分发	730
29.9.2	对称密钥协商	732
29.9.3	公钥分配	733
29.10	深入阅读	734
29.11	重要术语	734
29.12	本章小结	735
29.13	实践安排	736
29.13.1	习题	736
29.13.2	研究活动	737
第 30 章	因特网安全	739
30.1	网络层安全	739
30.1.1	两种方式	740
30.1.2	两个安全协议	741
30.1.3	IPSec 提供的服务	743
30.1.4	安全关联	744
30.1.5	因特网密钥交换 (IKE)	747
30.1.6	虚拟专用网 (VPN)	747
30.2	运输层安全	748
30.2.1	SSL 的体系结构	748
30.2.2	四个协议	750
30.3	应用层的安全	752

30.3.1 电子邮件的安全	753
30.3.2 相当好的保密 (PGP)	753
30.3.3 密钥环	755
30.3.4 PGP 的证书	756
30.3.5 S/MIME	758
30.3.6 S/MIME 的应用	761
30.4 防火墙	761
30.4.1 分组过滤防火墙	762
30.4.2 代理防火墙	762
30.5 深入阅读	763
30.6 重要术语	763
30.7 本章小结	764
30.8 实践安排	764
30.8.1 习题	764
30.8.2 研究活动	765

第七部分 附录

附录 A Unicode	769
A.1 平面	769
A.1.1 基本多语言平面 (BMP)	770
A.1.2 其他平面	770
A.2 ASCII	770
附录 B 进位制计数系统	773
B.1 不同的系统	773
B.1.1 基 10: 十进制	773
B.1.2 基 2: 二进制	773
B.1.3 基 16: 十六进制	774
B.1.4 基 256: 点分十进制记法	774
B.1.5 比较	775
B.2 转换	775
B.2.1 从任意数制到十进制的转换	775
B.2.2 从十进制到任意数制的转换	776
B.2.3 其他转换	777
附录 C 差错检测码	779
C.1 引言	779
C.1.1 差错的类型	779
C.1.2 冗余	779
C.1.3 检错与纠错的比较	779
C.1.4 编码	780

C.2	块编码.....	780
C.2.1	差错检测.....	781
C.2.2	汉明距离.....	781
C.2.3	最小汉明距离	781
C.3	线性块码.....	782
C.3.1	线性块码的最小距离	782
C.4	循环码.....	783
C.4.1	循环冗余检验	783
C.4.2	循环码的优点	785
C.4.3	其他循环码.....	785
附录 D	检验和	786
D.1	传统的检验和	786
D.1.1	思想	786
D.1.2	因特网的检验和	787
D.2	Fletcher 检验和	788
D.3	Adler 检验和	789
附录 E	HTML、XHTML、XML 和 XSL	791
E.1	HTML.....	791
E.1.1	标签	791
E.1.2	XHTML.....	794
E.2	XML 和 XSL	794
附录 F	Java 中的客户-服务器编程	796
F.1	UDP 程序	796
F.2	TCP 程序	798
附录 G	其他信息	801
G.1	端口号	801
G.2	RFC	802
G.3	联系地址	803
词汇表	804	
参考文献	827	

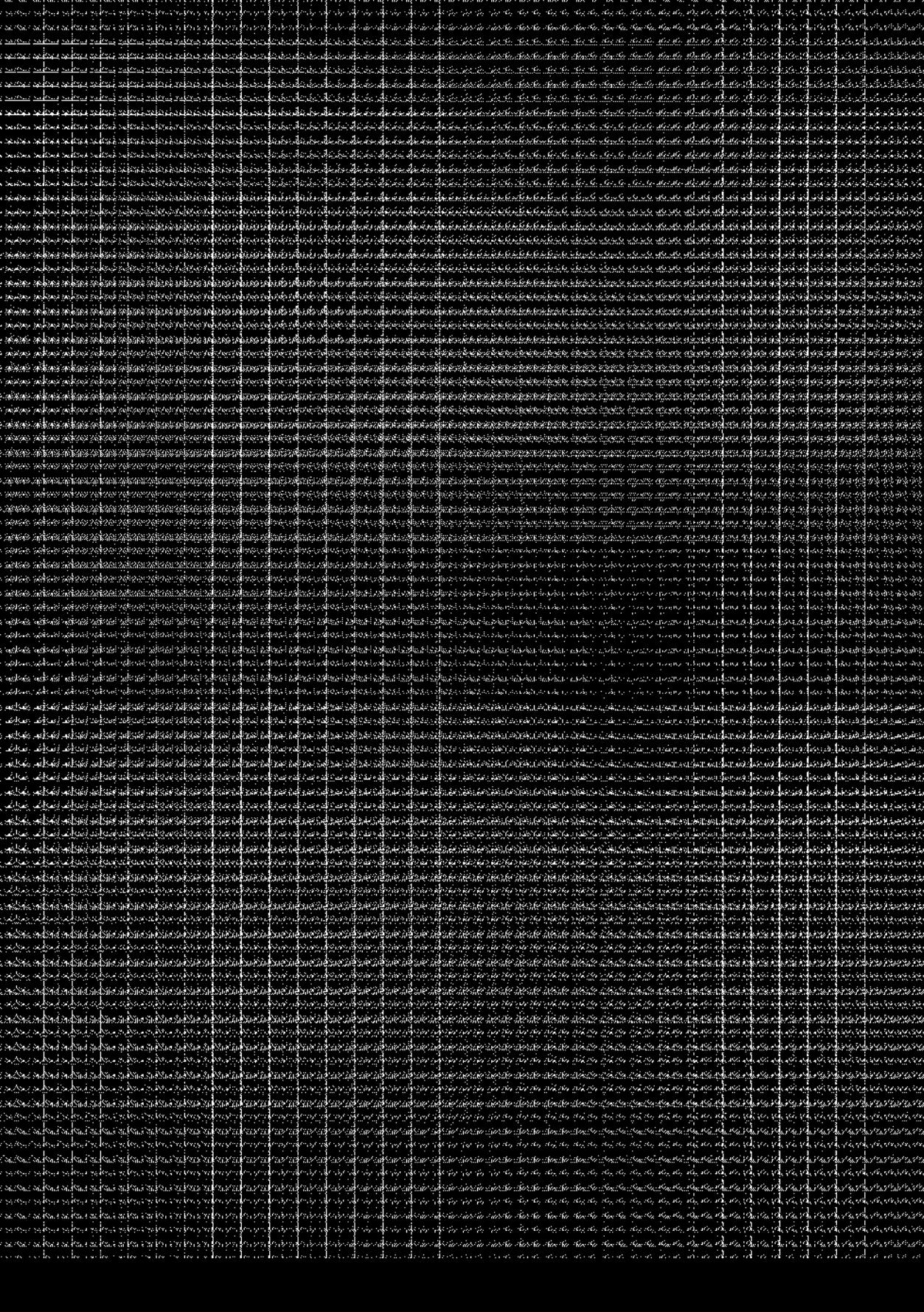
第一部分

引言和底层技术

第 1 章 引言

第 2 章 OSI 模型和 TCP/IP 协议族

第 3 章 底层技术



第1章 引言

因特网是个结构化的、有组织的系统。在讨论因特网是如何工作的，以及它和 TCP/IP 的关系之前，我们先来看看因特网的发展简史。接着我们再定义协议和标准的概念，以及它们彼此之间的关系。我们还要讨论与开发因特网标准有关的各种组织。这些标准并不是由任何一个特定组织开发的，而是来自于因特网用户们的共识。我们将讨论这些标准从发起到成熟的整个机制。当然作为引言，本章还有一些内容用来介绍因特网的管理组织。

目标

本章有以下几个目标：

- 了解因特网的发展简史。
- 定义人们在讨论因特网时经常提到的两个术语：协议和标准。
- 对因特网相关的标准组织进行分类，并简单地逐一介绍。
- 定义因特网标准这个概念，并解释这些标准制定所需要经历的机制。
- 讨论因特网的管理机构并简单介绍每个管理分支机构。

1.1 发展简史

网络（network）是一组互相连接的通信设备，如计算机和打印机。互联网（注意：这是指小写字母 i 开始的 internet）是指两个或更多的可以彼此通信的网络。最著名的互联网就是因特网（大写字母 I 开始的 Internet），它由成千上万个互连的网络所组成。超过 100 个国家和地区的个人以及不同的组织，如政府机关、学校、研究机构、公司以及图书馆等都在使用因特网。因特网的用户数以亿计。然而这个非凡的通信系统在 1969 年才问世。

1.1.1 ARPANET

在 20 世纪 60 年代中期，研究机构拥有的大型计算机都是独立的设备。不同厂家生产的计算机不能彼此通信。美国国防部的远景研究规划局（ARPA）希望找到一种连接计算机的方法，以便使他们资助的研究人员能够共享其研究成果，这样就可以减少费用和避免重复劳动。

1967 年，在一次计算机协会（ACM）的会议上，ARPA 提出了他们连接一些计算机的小网络 **ARPANET** 的概念。这个概念就是每一个主机（不一定是同一个厂家生产的）可以连接到称为接口报文处理机（IMP）的特殊计算机上。这些 IMP 则可以依次地互相连接起来。每一个 IMP 能够和其他的 IMP 通信，也可以和它连接的主机通信。

到了 1969 年，ARPANET 已经成为现实，4 个结点通过 IMP 连接成网络。这 4 个结点是：加州大学洛杉矶分校（UCLA）、加州大学 Santa Barbara 分校（UCSB）、斯坦福研究院（SRI）和犹他（Utah）大学。提供在这些主机之间通信的软件称为网络控制协议（NCP）。

1.1.2 因特网的诞生

1972 年，ARPANET 核心组的成员 Vint Cerf 和 Bob Kahn 共同进行一个研究项目，称为“网络互连项目”。他们希望将不同的网络链接起来，使得一个网络上的主机可以和另一个不同网络上的主机通信。这里有许多问题要解决：不同的分组长度，不同的接口，不同的传输速率，以及不同的可靠性需求。Cerf 和 Kahn 发明了一个概念，就是使用称为网关的设备作为中间的硬件，以便把分组从一个网络传送到另一个网络。

1.1.3 传输控制协议/网际协议（TCP/IP）

Cerf 和 Kahn 在 1973 年发表的里程碑论文阐述了实现分组的端到端交付的协议。这是 NCP 的一个新的版本。这篇关于传输控制协议（TCP）的论文包括的概念有：封装、数据报，以及网关的功能。其中有一个很激进的观点就是把纠错的责任从 IMP 转移到主机上。此时 ARPA 互联网已成为通信领域关注的焦点。大约在这个时期，ARPANET 的管理权被转交给国防通信署（DCA）。

1977 年 10 月，由 3 个不同网络（ARPANET、分组无线电网、分组卫星网）组成的互联网成功地问世了。网络之间的通信成为可能。

不久以后，当局决定将 TCP 划分成两个协议：传输控制协议（Transmission Control Protocol, TCP）和网际协议（Internet Protocol, IP）。IP 处理数据报的路由选择，而 TCP 负责高层的一些功能，如分段、重装和差错检测。这个新的组合被称为 TCP/IP。

1981 年，根据 DARPA 的合同，加州大学伯克利分校修改了 UNIX 操作系统，使它包括了 TCP/IP。这种将网络软件装进流行的操作系统中的做法大大普及了网络的互连。开放的（即非特定厂商的）伯克利 UNIX 为每一个厂家提供了源代码，使得这些厂家能够基于这些代码构造他们自己的产品。

在 1983 年，当局废弃了原来的 ARPANET 协议，而使 TCP/IP 成为 ARPANET 的正式协议。如果想利用因特网来访问不同网络上的主机，就必须运行 TCP/IP。

1.1.4 MILNET

1983 年 ARPANET 拆分成两个网络：军方用户使用的 MILNET 和非军方用户使用的 ARPANET。

1.1.5 CSNET

在因特网历史中的另一个里程碑就是 1981 年创建的 CSNET。CSNET 是美国国家科学基金会 (NSF) 资助的一个网络。这个网络是由一些大学设计的，这些大学和军方 (DARPA) 并没有联系，因此无法加入到 ARPANET。CSNET 是比较便宜的网络，网络中没有冗余链路，而传输速率也较慢。后来 CSNET 连接到 ARPANET 和第一个商用分组交换网 Telenet 上。

到了 20 世纪 80 年代中期，大多数具有计算机科学系的大学都连接到 CSNET 上。其他的一些组织和公司也建立了各自的网络，并使用 TCP/IP 进行互连。因特网这个名词最初是指政府出资连接的一些网络，但现在则是指使用 TCP/IP 协议互相连接在一起的网络。

1.1.6 NSFNET

随着 CSNET 的成功，NSF 于 1986 年出资建造了 NSFNET，一个连接了分布在美国的 5 个超级计算机中心的主干网。许多团体的网络都可以接入到这个主干网，它使用 T1 线路，速率是 1.544 Mbps，因而提供了跨越整个美国的连通性。

1990 年，ARPANET 正式退役并被 NSFNET 所取代。在 1995 年 NSFNET 又回到了它原来的科研网的概念。

1.1.7 ANSNET

1991 年美国政府断定 NSFNET 没有能力支持迅速增长的因特网通信量。IBM、Merit 和 MCI 等三个公司填补了这个空白，组成了称为高级网络和服务 (ANS) 的非盈利机构，共同建造了称为 ANSNET 的新的高速因特网主干网。

1.1.8 今日的因特网

今日的因特网并不是简单的层次结构。它是由许多广域网和局域网通过一些连接设备和交换站连接起来的。我们很难对因特网给出一种准确的表示，因为因特网一直在持续地变化着——新的网络不断加入进来，现有的网络需要更多的地址，已倒闭公司的网络需要从因特网上去除。在今天，绝大多数需要因特网连接的端用户都是使用因特网服务提供者 (ISP) 所提供的服务。有国际因特网服务提供者、国家因特网服务提供者、地区因特网服务提供者和本地因特网服务提供者。今日的因特网是由一些私人公司而不是由政府运营的。图 1.1 给出了因特网在概念上（而不是地理上）的视图。

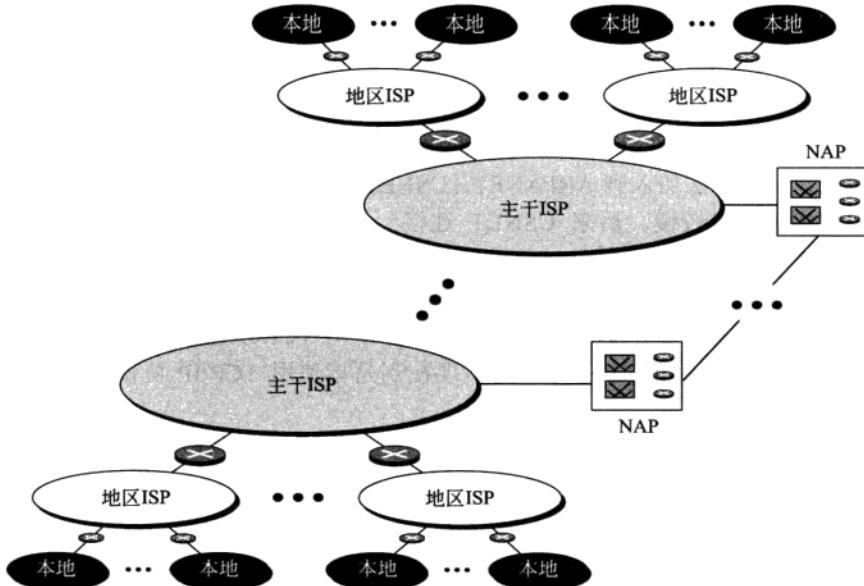


图 1.1 今日的因特网

1.1.9 主干 ISP

主干 ISP 由几个专门的公司创建和维持。在北美有许多个主干 ISP，其中最有名的有 SpringLink、PSINet、UUNet Technology、AGIS 和互联网 MCI。为了给端用户提供连通性，这些主干网通过一些被称为网络接入点（Network Access Point, NAP）的复杂的交换站点（通常是由第三家来运营）相互连接起来。有一些地区性 ISP 网络也通过一些专用交换站（称为对等结点）相互连接。主干 ISP 通常具有较高的数据率（例如 10 Gbps）。

1.1.10 地区 ISP

地区 ISP 是一些小 ISP，通过一个或多个主干 ISP 连接起来。它们位于等级中的第二层，数据率也低一些。

1.1.11 本地 ISP

本地 ISP 给端用户提供直接的服务。本地 ISP 可以连接到地区 ISP，或直接连接到主干 ISP。绝大多数的端用户都是连接到本地 ISP 的。我们应注意到，从这个意义上讲，本地 ISP 可以是一个仅仅提供因特网服务的公司，也可以是一个拥有网络并向自己的雇员提供服务的企业，或者是一个运行自己的网络的非盈利机构（如学院或大学）。以上的每一种都可以与地区 ISP 或主干 ISP 连接。

1.1.12 大事记

下面是按时间先后顺序排列的重要因特网事件。

- 1969 年, 4 个结点的 ARPANET 建立。
- 1970 年, ARPA 的主机实现 NCP。
- 1973 年, TCP/IP 协议族的开发开始。
- 1977 年, 使用 TCP/IP 的互联网经过测试。
- 1978 年, UNIX 分发到各学校/研究站点。
- 1981 年, CSNET 建立。
- 1983 年, TCP/IP 成为 ARPANET 的正式协议。
- 1983 年, MILNET 问世。
- 1986 年, NSFNET 建立。
- 1990 年, ARPANET 退役, 它被 NSFNET 取代。
- 1995 年, NSFNET 又回归为科研网。
- 1995 年, 开始出现因特网服务提供者 (ISP) 这样的公司。

1.1.13 因特网的发展

因特网的发展速度令人吃惊。仅仅在几十年里, 网络数已经从几十万个增长到几百万个。连接在这些网络上的计算机数也从几百台增长到几亿台。因特网仍在不断地发展, 而影响其发展的因素有以下一些:

- 新协议 新的协议需要增加进来, 而陈旧的协议应该去掉。例如, 在许多方面比 IPv4 更加优越的协议已被批准作为标准, 但还没有完全实现 (见第 27 章的 IPv6)。
- 新技术 能增加网络容量和给因特网用户提供更多带宽的许多新技术正在开发之中。
- 多媒体应用的增加 可以预计, 以前仅仅用来共享数据的因特网, 将被越来越多地用于共享多媒体信息 (音频和视频)。

1.2 协议和标准

在本节中, 我们要定义两个广泛使用的名词: 协议和标准。首先, 我们来定义协议, 它是“规则”的同义词。然后我们将讨论标准, 标准是一致同意的规则。

1.2.1 协议

在两个人或两台设备之间进行通信时需要遵守一些协议。协议 (protocol) 就是用以管理通信的一组规则。例如, 在两个人面对面的交流过程中, 每一种文化都存在一些隐性的规则, 以规定两个人如何开始对话, 怎样使对话继续, 怎样结束对话。类似地, 在打电话

的过程中，我们也需要遵守一组规则。如何连接对方（拨打对方的电话号码），如何响应电话呼叫（拿起话筒），如何问候对方，如何交谈流畅（当一方在讲话时，另一方倾听），如何结束交谈（挂机），这些都存在某种规则。

在计算机网络中，通信发生在不同系统的实体之间。实体就是任何一种能够发送或接收信息的东西。但是，两个实体不能简单地将比特流发送给对方，并希望对方能够理解它。要进行通信，这两个实体必须达成一种协议。协议定义了要通信的是什么，怎样进行通信，以及何时进行通信。协议的三个关键要素就是语法、语义和时序。

- **语法** 语法就是数据的结构或格式，也就是指数据呈现的顺序。例如，简单的协议可以规定数据的前 8 位是发送器的地址，第二个 8 位是接收器的地址，而剩下的数据流则是报文自身。数据在存储和传输时的比特顺序也是一种数据顺序。不同的计算机在存储数据时的比特顺序可能不同。当这些计算机相互通信时，必须要解决此类分歧。
- **语义** 语义指的是每一段比特流分别表示什么意思。一个特定的位模式(pattern)应怎样解释？基于这样的解释又该采取什么行动？例如，同样一个地址，它指的是下一步的路由呢？还是报文的终点呢？
- **时序** 时序涉及两个方面：数据应当在何时发送出去以及数据能够以多快的速度发送。例如，如果发送端产生数据的速率是每秒 100 兆比特（Mbps），但接收端只能以 1 Mbps 的速率处理数据，那么这样的传输将使接收端过载并导致数据大量丢失。

1.2.2 标准

在为设备制造商建立并维护一个开放与竞争市场的过程中，标准是必不可少的，同时它保证了数据与电信技术与处理过程在国内和国际上的互操作性。这些标准给制造商、供应商、政府机关以及其他的服务提供者提供了指导，以便在今天的市场中和在国际范围的通信中保证必要的连通性。

数据通信的标准分为两类：事实上的标准（表示“现实的”或“约定俗成”的）与法律上的标准（表示“根据法律”或“根据规章”的）。

- **事实上的标准** 这种标准并没有被哪个组织批准，但却被广泛地采用为事实上的标准（*de facto standard*）。事实上的标准往往最初被一些厂家在试图定义某个新产品或技术的功能时采用。例如 MS Office 和各种各样的 DVD 标准都是事实上的标准。
- **法律上的标准** 法律上的标准（*de jure standard*）是指那些已经被官方认可的组织通过并确定的标准。

1.3 标准化组织

标准的开发是通过一些标准创建委员会、论坛以及政府管理机构之间的合作来完成的。

1.3.1 标准创建委员会

虽然有许多专门负责建立标准的组织，但北美的数据通信主要依赖于以下的一些组织：

- **国际标准化组织（ISO）** 国际标准化组织是一个多国组织，其成员主要是来源于世界各国政府的标准创建委员会。在 1947 年创立的 ISO 是致力于使国际化标准在世界范围达成一致意见的完全志愿性组织。它的成员现在包括许多工业化国家的代表性团体，目标是使国际范围商品和服务的交换更加容易，同时提供一些模型以促进兼容性、质量改进、生产率增长和价格下降。ISO 在科学、技术和经济活动诸领域的合作开发中都是很积极的。本书主要关心的是 ISO 在信息技术领域中的努力，即为网络通信创建了开放系统互连（OSI）模型。美国在 ISO 中的代表是 ANSI。
- **国际电信联盟-电信标准部（ITU-T）** 在 20 世纪 70 年代早期，有不少国家已确立了各自的电信国家标准，但那时在国际范围的兼容性还较差。于是联合国就在国际电信联盟（ITU）下面成立了国际电报电话咨询委员会（Consultative Committee for International Telegraphy and Telephony, CCITT）。这个委员会致力于研究和建立电信的通用标准，特别专注于电话和数据通信系统领域。1993 年 3 月 1 日，CCITT 改名为国际电信联盟-电信标准部（ITU-T）。
- **美国国家标准化局（ANSI）** 尽管名字是这样取的，但美国国家标准化局（ANSI）完全是民间非赢利组织，而不隶属于美国联邦政府。不过，ANSI 一切活动的出发点都以美国国家的和公民的利益为重。ANSI 阐明的目标包括作为美国标准化志愿机构的协调组织，进一步推广标准的应用以加快美国经济的发展，并确保公众利益的分享与保护。ANSI 的成员包括各种专业性协会、工业协会、政府的和管理的团体以及消费者的组织。
- **电气和电子工程师学会（IEEE）** 电气和电子工程师学会（IEEE）是世界上最大的专业性工程师学会。它的范围是国际性的，目标是在电气工程、电子学、无线电以及工程的相关分支领域发展理论、创造性和提高产品质量。作为其中的一个目标，IEEE 对计算机和通信领域标准的开发和推广进行监督。
- **电子工业协会（EIA）** 与 ANSI 并驾齐驱的电子工业协会（EIA）也是一个非赢利组织，它的主要任务是督促电子制造业的发展。它的活动除了开发标准外，还包括对公众的教育培训和促进政府对标准的制定，在定义数据通信的物理连接接口和电子信令规约方面，EIA 都做出了显著的贡献。
- **万维网联盟（W3C）** 这个组织是由 Tim Berners-Lee 在麻省理工学院计算机科学实验室创建的。它的建立是为了向新标准提供工业上的可执行性。W3C 已建立了世界范围的地区办事处。
- **开放移动联盟（OMA）** 标准化组织 OMA 的创建是为了将计算机网络和无线技术方面的各种论坛进行统一管理。它的任务是为应用协议提供统一标准。

1.3.2 论坛

电信技术的发展往往要快于标准委员会对标准的批准。标准委员会是按规章办事的团体，其特点就是行动缓慢。为满足工作的模型和达成一致意见的需要以及加快标准化的过程，许多特殊兴趣的群组就成立了一些论坛，它由感兴趣的一些公司的代表组成。论坛与大学和用户们进行合作来对一些新技术进行测试、评价和标准化。当他们把注意力集中在某个特定技术时，论坛就可以使电信部门加速对这些技术的采纳和使用。论坛把他们结论提交给制定标准的团体。在电信工业界的几个重要的论坛包括：

- **帧中继论坛** 帧中继论坛是由数字设备公司、北方电信、思科公司以及 StrataCom 公司发起建立的，其目的是推进帧中继的应用和实现。今天它已拥有约 40 个成员，代表着北美、欧洲和太平洋周边地区。该论坛研究的问题包括流量控制、封装、转换和多播。论坛得出的结论将提交给 ISO。
- **ATM 论坛** ATM 论坛推进异步传递方式（ATM）技术的接受和使用。ATM 论坛是由用户屋内设备（如小交换机系统）的卖主和电话局（如电话交换机）提供者所组成。它关心服务的标准化以保证互操作性。
- **通用即插即用（UPnP）论坛** UPnP 论坛是一个计算机网络论坛，它通过设计一些零配置即可使用的网络设备来支持并推进网络实现的简单化。一台 UPnP 兼容的设备不需要任何配置即可加入某个网络。

1.3.3 管理机构

在美国，所有的通信技术是由一些政府机构来管理，例如，联邦通信委员会。这些机构的作用是通过对无线电、电视和无线/有线通信的管理来保护公众的利益。

- **联邦通信委员会（FCC）** 联邦通信委员会（FCC）管理与通信有关的州间和国际间的商务。

以上机构的网址参见附录 G。

1.4 因特网标准

因特网标准（Internet standard）是经过充分测试的规约，只要是与因特网打交道，就会用到它们，并要服从于它们。因特网标准是必须被遵守的正式规约。一个规约要成为因特网标准需要经过严格的过程。规约从因特网草案开始。因特网草案（Internet draft）是正在加工的文档（正在进行的工作），没有被官方正式承认，其生存期为 6 个月。一旦被因特网管理机构推荐，该草案就可以作为 RFC（Request for Comment）发布。每一个 RFC 都是经过编辑整理的，并分配有一个 RFC 编号，任何感兴趣的组织都可以得到它。

这些 RFC 的一生要经历几个成熟度，并且根据它们的需求级别进行归类。

1.4.1 成熟度

在其生命期内，一个 RFC 总是属于以下 6 种成熟度（maturity level）之一：建议标准、草案标准、因特网标准、历史的、实验的和提供信息的（见图 1.2）。

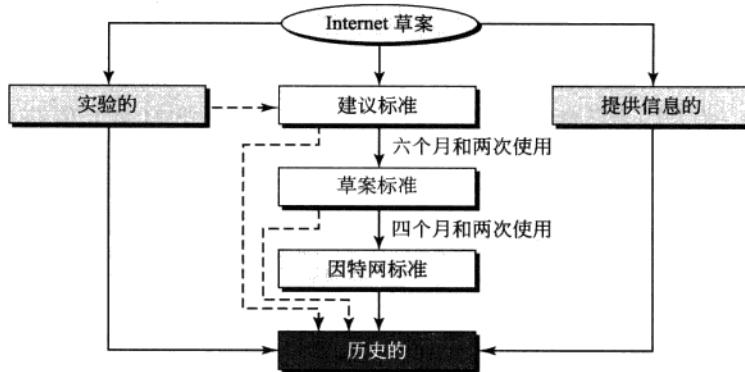


图 1.2 RFC 的成熟度

建议标准

建议标准是稳定的、被广泛了解的，并且是因特网界对其有足够兴趣的规约。在这个成熟程度上的规约通常都已经被几个不同的组测试和实现过。

草案标准

建议标准要上升到草案标准至少要经过两个成功的、独立的和可互操作的实现。通过克服一些困难，在正常的情况下，经过修订（如果遇到了特定问题）的草案标准会成为因特网标准。

因特网标准

草案标准在经过成功实现的证明后就可以成为因特网标准。

历史的

从历史的角度看，历史的 RFC 是很有意义的。这种 RFC 或者被后来的规约所取代，或者是因为没有通过必要的成熟程度而从未成为过因特网标准。

实验的

被列入实验的 RFC 就表示它的工作属于正在实验的情况，但并不影响因特网的运行。这种 RFC 不能够在任何实用的因特网服务中实现。

提供信息的

被划为提供信息的 RFC 包括与因特网有关的一般性、历史性或指导性的信息。这种 RFC 通常是由非因特网的组织中的某个人（例如，设备供应商）撰写。

1.4.2 需求级别

RFC 分为 5 个需求级别 (requirement level): 必需的、推荐的、选用的、限制使用的

和不推荐的（见图 1.3）。

必需的

如果一个 RFC 被标明为必需的，则它必须被所有的因特网系统实现以达到最低限度的一致性。例如，IP（第 7 章）和 ICMP（第 9 章）都是必需的协议。

推荐的

被标明为推荐的 RFC 在最低限度一致性中并不是必须的，但是因为它有用，所以被推荐。例如，FTP（第 21 章）和 TELNET（第 20 章）都是推荐的协议。

选用的

被标明为选用的 RFC 不是必需的，也不是推荐的。但是，某个系统可以因自身的利益而使用它。

限制使用的

被标明为限制使用的 RFC 只能使用在受限的情况下。大多数实验的 RFC 属于这类。

不推荐的

被标明为不推荐的 RFC 对一般的使用都不合适。通常历史的（不赞成的）RFC 就属于这类。

这些 RFC 都可以在 <http://www.rfc-editor.org> 中找到。

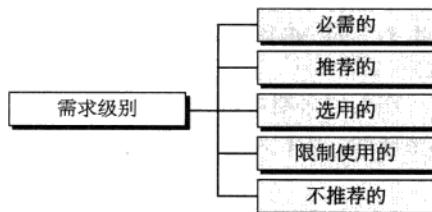


图 1.3 RFC 的几种需求级别

1.5 因特网的管理机构

主要起源于研究领域的因特网现已演进并得到了更广泛的用户基础，吸引了大量的商业活动。协调因特网的各种问题的不同群组共同引领着因特网的增长与发展。附录 G 给出了一些组织的地址、电子邮件地址和电话号码。因特网管理的一般组织方式如图 1.4 所示。

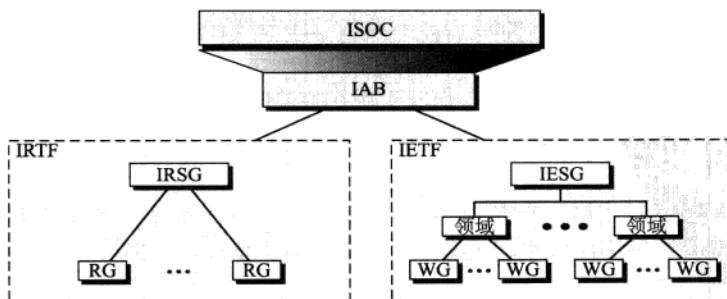


图 1.4 因特网的管理

1.5.1 因特网协会 (ISOC)

因特网协会 (Internet Society, ISOC) 是成立于 1992 年的国际性的、非赢利组织，用来提供对因特网标准化过程的支持。ISOC 通过维护和支持其他的一些因特网管理机构来完成这一任务，如 IAB、IETF、IRTF 以及 IANA（见以下几节）。ISOC 还推进与因特网有关的研究和其他一些学术活动。

1.5.2 因特网体系结构研究委员会 (IAB)

因特网体系结构研究委员会 (Internet Architecture Board, IAB) 是 ISOC 的技术顾问。IAB 的主要任务是监督 TCP/IP 协议族的持续发展，以及用技术咨询能力向因特网界的研究人员提供服务。IAB 通过其下属的两个主要机构，即因特网工程部 (IETF) 和因特网研究部 (IRTF) 来完成此任务。IAB 的另一个责任就是对 RFC 的编辑管理 (RFC 已在本章的前面讲到了)。IAB 还是因特网与其他标准化组织和论坛进行对外联络的机构。

1.5.3 因特网工程部 (IETF)

因特网工程部 (Internet Engineering Task Force, IETF) 是在因特网工程指导小组 (IESG) 领导下的工作组论坛。IETF 负责找出运行中的问题，并对这些问题提出解决的方法。IETF 还开发并评审打算成为因特网标准的一些规约。这些工作组被划分成一些领域，每一个领域集中研究某个特定的题目。目前已经定义了 9 个领域，这个数字并不是固定和一成不变的。这些领域是：

- 应用
- 因特网协议
- 路由选择
- 运行
- 用户服务
- 网络管理
- 运输
- 下一代网际协议 (IPng)
- 安全性

1.5.4 因特网研究部 (IRTF)

因特网研究部 (Internet Research Task Force, IRTF) 是在因特网研究指导小组 (IRSG) 领导下工作组的论坛。IRTF 关注的是有关因特网协议、应用、体系结构和技术的长期研究课题。

1.5.5 因特网赋号管理局和因特网名字与号码指派公司

到 1998 年 10 月之前，由美国政府支持的因特网赋号管理局（Internet Assigned Numbers Authority, IANA）一直负责着因特网域名和地址的管理。而在那之后，由国际委员会管理的，称为因特网名字与号码指派公司（Internet Corporation for Assigned Names and Numbers, ICANN）的民间非赢利公司取代了 IANA 的工作。

1.5.6 网络信息中心（NIC）

网络信息中心（Network Information Center, NIC）负责收集和分发有关 TCP/IP 协议族的信息。

有关因特网组织的地址和网址见附录 G。

1.6 深入阅读

要更详细地了解本章所讨论的内容，我们推荐以下一些书和网站。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

1.6.1 书籍和论文

有一些书和论文对因特网历史的讨论虽然简单但是很完整，包括[Seg 98], [Lei et al. 98], [Kle 04], [Cer 89] 和 [Jen et al. 86]。

1.6.2 网站

以下网站就本章所讨论的话题给出了更多的资料。

ietf.org

IETF 的网站

w3c.org

W3C 标准组织的网站

1.7 重要术语

远景研究规划局（ARPA）

国际电报电话咨询委员会（CCITT）

美国国家标准学会（ANSI）

CSNET

ANSNET

事实上的标准

ARPANET

法律上的标准

ATM 论坛

电子工业协会（EIA）

联邦通信委员会 (FCC)	MILNET
帧中继论坛	网络
电气和电子工程师学会 (IEEE)	网络接入点 (NAP)
国际标准化组织 (ISO)	网络信息中心 (NIC)
国际电信联盟-电信标准部 (ITU-T)	NSFNET
因特网体系结构研究委员会 (IAB)	协议
因特网赋号管理局 (IANA)	开放移动联盟 (OMA)
因特网名字与号码指派公司 (ICANN)	请求评论 (RFC)
因特网草案	需求级别
因特网工程部 (IETF)	语义
因特网研究部 (IRTF)	语法
因特网服务提供者 (ISP)	时序
因特网协会 (ISOC)	传输控制协议 (TCP)
因特网标准	通用即插即用 (UPnP) 论坛
网际协议 (IP)	万维网联盟 (W3C)
成熟度	

1.8 本章小结

- 网络是一组互相连接的通信设备。互联网是指两个或更多的可以彼此通信的网络。最著名的互联网就是因特网，它由成千上万个互连的网络所组成。
- 网络互连的历史可追溯到 20 世纪 60 年代中期出现的 ARPA。因特网的诞生与 Cerf 和 Kahn 的研究工作以及网络连接所使用的网关设备的出现有着密切联系。1977 年，国防通信署 (DCA) 开始对 ARPANET 负责，并使用了 TCP 和 IP 这两个协议来处理各个网络彼此之间的数据报路由选择。MILNET、CSNET、NSFNET 和 ANSNET 都是从 ARPANET 演化出来的。
- 今日的因特网是由许多广域网和局域网通过一些连接设备和交换站连接起来的。在今天，绝大多数需要因特网连接的端用户都是使用因特网服务提供者 (ISP) 所提供的服务。分别有主干 ISP、地区 ISP 和本地 ISP。
- 协议是管理数据通信的一组规则：协议的要素是语法、语义和时序。在计算机网络中，通信发生在不同系统上的实体之间。要进行通信，这两个实体必须达成一种协议。协议定义了要通信的是什么，怎样进行通信，以及何时进行通信。
- 在建立和维护一个开放与竞争市场的过程中，标准必不可少。这些标准给制造商、供应商、政府机关以及其他的服务提供者提供了指导，以便在今天的市场中和在国际范围的通信中保证必要的连通性。数据通信的标准分为两类：事实上的标准与法律上的标准。
- 因特网标准是经过充分测试的规约，只要是与因特网打交道，就会用到它，并要服从于它。因特网草案是正在加工的文档（正在进行的工作），没有被官方正式承认，

其生存期为 6 个月。一旦被因特网管理机构推荐，该草案就可以作为 RFC (Request for Comment) 发布。每一个 RFC 都是经过编辑整理的，并分配有一个 RFC 编号，任何感兴趣的组织都可以得到它。这些 RFC 要经历几个成熟度，并且会根据它们的需求级别进行分类。

- 因特网的管理体系是跟随因特网一起发展起来的。ISOC 推动了与因特网有关的各项研究和学术活动。IAB 是 ISOC 的技术顾问。IETF 是专门负责业务问题的那些工作组的论坛。IRTF 则是关注长期研究课题的工作组的论坛。ICANN 负责管理因特网的域名和地址。NIC 负责收集和分发关于 TCP/IP 协议的信息。

1.9 实践安排

1.9.1 习题

1. 试利用因特网找出 RFC 的数量。
2. 试利用因特网找出 RFC 2418 和 1603 的主题。
3. 试利用因特网找出讨论 IRTF 工作组的指南和规程的 RFC。
4. 试利用因特网找出历史的 RFC 的两个例子。
5. 试利用因特网找出实验的 RFC 的两个例子。
6. 试利用因特网找出提供信息的 RFC 的两个例子。
7. 试利用因特网找出讨论 FTP 应用的 RFC。
8. 试利用因特网找出网际协议 (IP) 的 RFC。
9. 试利用因特网找出传输控制协议 (TCP) 的 RFC。
10. 试利用因特网找出详细阐述因特网标准化过程的 RFC。

1.9.2 研究活动

11. 找出并研究由 ITU-T 开发的 3 个标准。
12. 找出并研究由 ANSI 开发的 3 个标准。
13. EIA 已经开发了一些接口标准。找出并研究由 EIA 开发的两个这种标准。什么是 EIA 232?
14. 找出并研究由 FCC 制定的 3 个涉及到 AM 和 FM 传输的规程。

第 2 章 OSI 模型和 TCP/IP 协议族

1990 年以前，在数据通信和组网文献中占主导地位的分层模型是开放系统互连（Open Systems Interconnection, OSI）模型。当时所有人都认为 OSI 模型将是数据通信的最终标准——然而这种情况并未发生。现在 TCP/IP 协议族成为了占主导地位的商用体系结构，因为它已经在因特网中应用，并且通过了广泛的测试，而 OSI 模型则从来没有被完全实现过。

本章首先简要地讨论一下 OSI 模型，然后再来关注 TCP/IP 协议族。

目标

本章有以下几个目标：

- 讨论数据通信和组网中多层结构的思想以及层与层之间的相互关系。
- 讨论 OSI 模型及其体系结构，并指出层与层之间的接口。
- 简单地讨论一下 OSI 模型中每一层的功能。
- 介绍 TCP/IP 协议族，并将其层次结构与 OSI 模型中相对应的层进行比较。
- 举例说明 TCP/IP 协议中每一层的功能。
- 讨论在 TCP/IP 协议族的一些层中为将报文从源点传递到终点而使用的编址机制。

2.1 协议分层

在第 1 章中我们已经了解到两个实体之间要进行通信就需要有一个协议。而当该通信比较复杂时，就有必要将这个复杂的任务划分为多层。此时我们就需要有多个协议，每一层都有各自的协议。

也许通过一段现实场景的描述可以更好地理解决议分层的作用。下面举两个例子。在第一个例子中，因为通信过程非常简单，所以只需要单层就可以实现，而在第二个例子中则需要有三层。

例 2.1

假设玛丽亚和安是邻居，她们志趣相投，但是玛丽亚只会讲西班牙语而安只会说英语。由于她们俩在儿时都曾学过手语，每星期中有两三天她们会抽时间在咖啡馆见面，并用手语交换彼此的观点和想法。偶尔，她们也会使用双语字典。此时的通信是面对面的，只需单层就可实现，如图 2.1 所示。

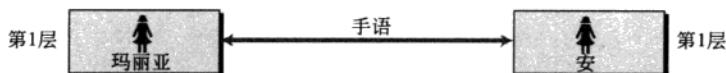


图 2.1 例 2.1

例 2.2

再假设由于工作上的原因，安不得不搬迁到另外一个城市。在搬走之前，两人最后一次在咖啡馆见面。虽然都很伤心，但是玛丽亚给安带来了一个惊喜，她打开一个小包，里面装着两台机器。第一台机器能够扫描用英语写的信，并把信的内容翻译成某种密码，反之也可将密码翻译成英语。另一台机器则能够将西班牙语写的信翻译成密码，也能将密码翻译成西班牙语。安带走了第一台机器，而玛丽亚留下了第二台机器。通过使用密码，这两个好朋友仍然能够互相通信，如图 2.2 所示。

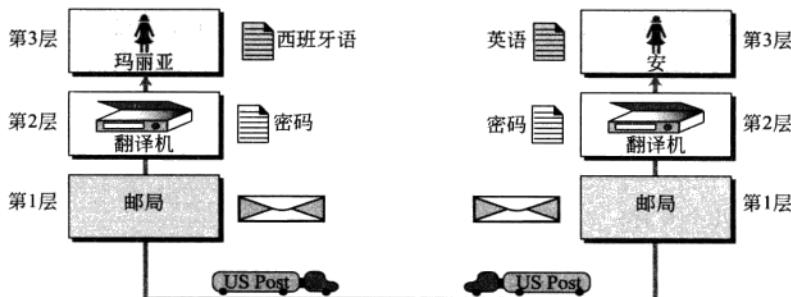


图 2.2 例 2.2

玛丽亚和安之间的通信过程如下。在第三层，玛丽亚用她所擅长的西班牙语写了一封信，然后用那台翻译机扫描信的内容并将其翻译成一封密码信。接着玛丽亚把信放进信箱并投入邮局的信箱。邮车会将信送到安居住的城市的邮局，而邮局又会将信投递到安的住所。安用她的翻译机将信中的密码翻译成英语。对于从安到玛丽亚的通信过程，其处理方式相同，方向相反。不管是哪个方向，信的内容都是用密码来传递的，虽然玛丽亚和安都不懂密码，但是通过分层的通信，她们仍然能够交换彼此的想法。

2.1.1 分层结构

从例 2.2 可知发送方一共完成了三项不同的任务，而接收方也有三项相应的任务。在寄信人和收信人之间传送信件的过程是由邮车完成的。而这些任务必须按照分层结构中指定的顺序执行，这一点可能不那么显而易见。在发送方，这封信必须首先写好，再翻译成密码，然后投到邮箱中，之后再由邮车从邮箱中取出并传递到邮局。在接收方，这封信首先要投递到收信人的信箱中，收信人才能收到信并读出其内容。

2.1.2 服务

发送方的每一层都要使用下一层所提供的服务。位于最高一层的寄信人使用了中间一

层提供的服务，而中间一层使用了最低一层提供的服务，最低一层则用到了邮车服务。

2.2 OSI模型

成立于1947年的国际标准化组织（International Standards Organization, ISO）是一个多国团体，专门就一些国际标准达成世界范围的一致。全世界大约有四分之三的国家派代表参加。一个全方位覆盖网络通信问题的ISO标准就是开放系统互连（OSI）模型，它最早出现在20世纪70年代后期。

ISO是个组织；而OSI是个模型。

一个开放系统（open system）就是一组协议的集合，它使得两个不同的系统之间能够互相通信，而毋须考虑其底层体系结构。OSI模型的作用就是展示两个不同的系统怎样才能做到互相通信，且不需要改变底层的硬件或软件的逻辑。OSI模型不是一个协议，它是一个为了更好地理解并设计出灵活、稳健且可互操作的网络体系结构而存在的模型。为OSI框架中的各种协议的创建提供基础，这才是OSI模型的本意。

OSI模型是一个分层的框架结构，其目的是为了设计出能够让各种类型的计算机系统相互通信的网络系统。它由七个独立且相关的层组成，而每层都定义了信息通过网络传输的完整过程中的一部分（见图2.3）。掌握了OSI模型的基本概念后，就有了学习数据通信的牢固基础。



图2.3 OSI模型

2.2.1 分层的体系结构

OSI模型由七个顺序排列的层组成：物理层（第1层），数据链路层（第2层），网络层（第3层），运输层（第4层），会话层（第5层），表示层（第6层）和应用层（第7层）。图2.4所示为设备A把一个报文发送到设备B的过程中涉及到的层。在报文从设备A到设备B的途中可能会经过多个中间结点。这些中间结点一般只涉及到OSI模型中的下三层。

在开发这个模型时，设计者们首先提取出传输数据过程中最基本的要素，然后进一步判断哪些组网功能需要配合使用，并将这些功能划分为不同的组，从而形成了这些层。每一层都定义了一组关系密切的功能，并且这些功能与其他层的功能有着明显的区别。用这种方式来定义和定位功能集，设计者们就创建出了既有丰富功能又很灵活的体系结构。最重要的是，OSI模型使得各系统完全可互操作，而没有OSI模型这些系统将无法兼容。

处于一台机器上的每一层都要调用紧挨的下一层的服务。例如，第3层使用第2层提供的服务，同时向第4层提供服务。而机器和机器之间，从逻辑上看起来像是一台机器中的第x层与另一台机器中的第x层之间在互相通信。这种通信是由协议来控制的，协议就是事先都同意的一组规则和约定。

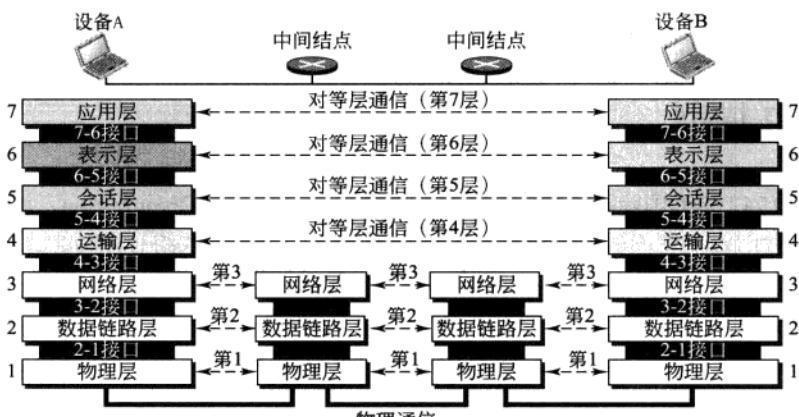


图 2.4 OSI 的各层

2.2.2 层与层之间的通信

在图 2.4 中，设备 A 向设备 B 发送报文（经过中间结点）。从发送方看，报文从第 7 层开始一直向下传送到第 1 层。在第 1 层中完整的数据包被转换为可传送到接收设备的形式。从接收方看，报文又从第 1 层开始向上一直传到第 7 层。

层之间的接口

这些数据和网络信息之所以能够在发送设备中逐层向下传递，同时又在接收设备中能够逐层向上交付，是因为每对相邻的层之间有一个接口（interface）。每个接口都定义了该层必须向它的上层提供什么样的信息和服务。定义清晰明确的接口和功能可以使网络模块化。只要该层向它的上层提供了预期的服务，层功能的具体实现是可以修改或替换的，而不需要对周围的其他层进行改动。

层的组织方式

上述 7 层可看成分属于三个组。第 1、2 和 3 层（物理层、数据链路层和网络层）是网络支撑层，这些层的任务是在物理上把数据从一个设备传送到另一个设备（例如，电气规约、物理连接、物理编址、以及运输的定时和可靠性）。第 5、6 和 7 层（会话层、表示层和应用层）可以看成是用户支撑层，这些层使得一些本来没有关系的软件系统之间有了互操作性。第 4 层（运输层）将这两部分链接起来，使得底层所发送的是高层可使用的形式。OSI 的高层几乎都是用软件来实现的，而低层则是硬件和软件的结合，除了物理层，它的绝大部分是硬件。

图 2.5 给出了 OSI 各层的概貌，D7 数据表示第 7 层的数据单元，D6 数据表示第 6 层的数据单元，如此类推。处理过程从第 7 层（应用层）开始，然后按降序逐层下传。每一层都可以在数据单元上附加一个首部。在第 2 层还要加上尾部。当这样格式化的数据单元通过物理层（第 1 层）时，就转换为电磁信号并沿着一条物理链路传输。

到达终点后，信号首先进入第 1 层，并还原为它的数字形式。这个数据单元向上逐层通过 OSI 各层。每个数据块在到达上一层时，相应的发送层所附加的首部和尾部就被移去，然后由该层进行相应的处理。当数据块到达第 7 层时，这个报文就变为应用层所需要的形式，交给接收者使用。

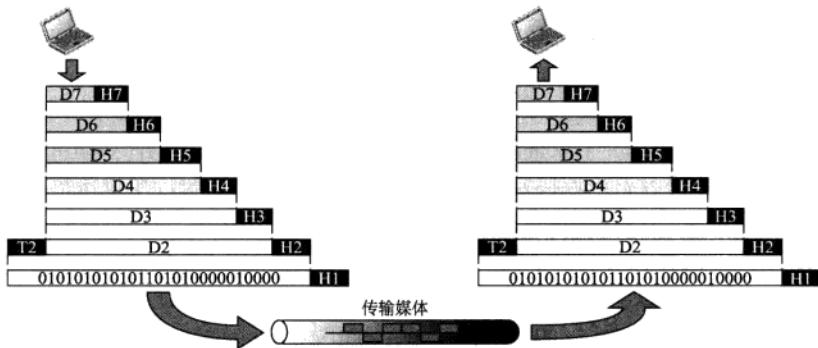


图 2.5 使用 OSI 模型交换数据

2.2.3 封装

图 2.5 揭示了 OSI 模型中数据通信的另一个特点：封装。第 7 层的分组被封装在第 6 层的分组中。整个第 6 层的分组又被封装在第 5 层的分组中，依此类推。

换言之，第 N 层的数据部分就是第 $N+1$ 层的完整分组（数据和开销）。这一概念称为封装，因为第 N 层并不知道封装后的分组中的哪个部分是数据，哪个部分是首部或尾部，所以对第 N 层来说，从第 $N+1$ 层传来的整个分组被看作是一个整体。

2.2.4 OSI 模型中的各层

在本节中，我们将简要地叙述 OSI 模型中各层的功能。

物理层

物理层 (physical layer) 协调通过物理媒体传送比特流时所需要的各种功能。物理层涉及到接口和传输媒体的机械和电气规约。它还定义了这些物理设备及接口为了实现传输必须完成的过程和功能。

物理层负责把逐个的比特从一跳（结点）移动到下一跳。

物理层关心的是以下一些内容：

- **接口和媒体的物理特性** 物理层定义了设备与传输媒体之间的接口特性。它还定义了传输媒体的类型（见第 3 章）。
- **比特的表示** 物理层的数据由一串没有任何解释的比特 (bit) 流 (0 和 1 的序列) 组成。发送时，比特必须经过编码变成信号——电的或光的。物理层对编码 (encoding) 的类型也进行了定义（即 0 和 1 怎样变为信号）。
- **数据率** 传输速率 (transmission rate)，即每秒发送的比特数，也在物理层定义。换言之，物理层定义一个比特的持续时间。
- **比特的同步** 发送设备和接收设备不仅要使用同样的比特率，而且还要在比特级进行同步。换言之，发送设备和接收设备的时钟必须是同步的。

- **线路配置** 物理层要考虑到设备与媒体的连接。点对点配置（point-to-point configuration）时两个设备通过专用链路连接在一起。多点配置（multipoint configuration）时若干个设备共享一条链路。
- **物理拓扑** 物理拓扑定义了设备如何连成一个网络。设备的连接可使用网状拓扑（mesh topology，每一个设备都和其他所有设备连接），星状拓扑（star topology，所有设备都通过中央设备来连接），环状拓扑（ring topology，每一个设备都连接到下一个设备而形成环）或总线拓扑（bus topology，所有设备都在一个公共链路上）等方式。
- **传输方式** 物理层还定义了两个设备之间的传输方向：单工、半双工或全双工。在单工方式（simplex mode）下，只有一个设备可以发送，另一个设备只能接收。单工方式是一种单向通信。在半双工方式（half-duplex mode）下，两个设备都可以发送和接收，但不能在同一时间进行。在全双工方式（full-duplex mode，或简称为双工方式）下，两个设备可在同一时间发送和接收。

数据链路层

数据链路层（data link layer）把物理层（即原始的传输设施）转换为可靠的链路。它使物理层对上层（网络层）看起来好像是无差错的。数据链路层的任务包括：

- **组帧** 数据链路层把从网络层收到的比特流划分成可以处理的数据单元，称之为帧（frame）。
- **物理编址** 如果这些帧需要发送给本网络内的另一个系统，那么数据链路层就要在帧上附加一个首部，指明帧的发送方和/或接收方。如果这个帧要发送给本网络以外的一个系统，则接收方地址应当是连接本网络和下一个网络的连接设备的地址。
- **流量控制** 如果接收方吸收数据的速率小于发送方产生数据的速率，那么数据链路层就应使用流量控制机制来预防接收方因过负荷而无法工作。
- **差错控制** 数据链路层增加了一些措施来检测并重传受损伤的帧或丢失的帧，因而使物理层增加了可靠性。它还采用了某种机制来识别重复的帧。差错控制通常是在一个帧的后面加上一个尾部来实现的。
- **接入控制** 当两个或更多的设备连接到同一条链路时，数据链路层就必须决定任一时刻该由哪一个设备对链路有控制权。

网络层

网络层（network layer）负责把分组从源点交付到终点，这可能要跨越多个网络（链路）。如果说数据链路层监督的是同一个网络（链路）上的两个系统之间的分组交付，那么网络层则要确保每个分组从源点出发并最终抵达目的地。

如果两个系统连接到同一条链路上，则一般来说就不需要网络层了。但是，如果两个系统连接在不同的网络（链路）上，而这些网络（链路）是由一些连接设备连接起来的，那么就需要网络层来完成从源点到终点的交付。数据链路层的任务包括：

- **逻辑编址** 由数据链路层实现的物理编址处理的是本地寻址问题。如果分组穿过了网络的边界，我们就需要另一种编址系统来帮助我们区分源系统和目的系统。网络层给从上层传来的分组附加一个首部，其中包括发送方和接收方的逻辑地址，以及其他一些信息。
- **路由选择** 当多个独立的网络或链路互相连接组成互联网（网络的网络）或组成一个更大的网络时，这些连接设备（称为路由器或交换机）就要为数据分组选路或交

换以到达它们最终的目的地。网络层的功能之一就是提供这种机制。

运输层

运输层（transport layer）负责完整报文的进程到进程的交付（process-to-process delivery）。进程是运行在主机上的应用程序。网络层管理的是单个分组从源点到终点的交付（source-to-destination delivery），它并不考虑这些分组之间的关系。网络层独立地处理每个分组，就好像每个分组属于独立的报文那样，而不管是否真的如此，运输层则要确保整个报文原封不动地按序到达，它要监督从源点到终点这一级的差错控制和流量控制。运输层的任务包括：

- **服务点编址** 计算机往往在同一时间运行多个程序。因此，从源点到终点的交付并不仅仅是从某个计算机交付到另一个计算机，同时还指从某个计算机上的特定进程（运行着的程序）交付到另一个计算机上的特定进程（运行着的程序）。因此，运输层的首部必须包括一种称为服务点地址（或端口地址）的地址。网络层将各分组送抵正确的计算机，而运输层则将完整的报文递交给该计算机上正确的进程。
- **分段与重装** 一个报文被划分成若干个可传输的报文段，每个报文段应包含一个序号。在报文到达终点时，运输层利用这些序号能够把它们重装起来，同时对在传输时丢失的分组也能够识别并替换为正确的分组。
- **连接控制** 运输层可以是无连接的，也可以是面向连接的。无连接的运输层把每个报文段看成是独立的数据报，并把报文段交付给终点设备上的运输层。面向连接的运输层在发送报文段之前，先要与终点设备上的运输层建立一条连接。当全部数据都传送完毕后，连接就被释放掉。
- **流量控制** 如同数据链路层一样，运输层也要负责流量控制。不同的是运输层的流量控制是端到端的，而不是单条链路上的流量控制。
- **差错控制** 如同数据链路层一样，运输层也要负责差错控制。不同的是运输层的差错控制是端到端的，而不是只限于单条链路上的差错控制。发送端的运输层必须确保整个报文没有差错（即无损伤、无丢失、无重复）地到达接收端的运输层。纠错通常通过重传来完成。

会话层

对某些进程来说，下四层（物理层、数据链路层、网络层和运输层）提供的服务还不够充分。会话层（session layer）就是网络的对话控制器。它用于建立、维持并同步正在通信的系统之间的交互。会话层的具体任务包括：

- **对话控制** 会话层允许两个系统进入对话状态。它允许两个进程之间的通信按半双工（在某个时刻单向通信）或全双工（在某个时刻双向通信）方式进行。
- **同步** 会话层允许进程在数据流中插入若干个检查点（同步点）。例如，如果某系统要发送一个2000页的文件，那么可以在每100页后插入一个检查点，这样就保证了以每100页为一个单元的数据独立地接收和确认。在这种情况下，如果传输第523页时计算机崩溃了，那么系统恢复后只需要重传501~523页。第501页以前的就不需要重传了。

表示层

表示层（presentation layer）考虑的问题是两个系统所交换的信息的语法和语义。表示层的具体任务有以下一些：

- **转换** 分别位于两个系统上的进程（运行着的程序）所交换的信息的形式通常都是

字符串、数字等等。这些信息在传送之前必须变为比特流。由于不同的计算机使用不同的编码系统，表示层的任务就是在这些不同的编码方法之间提供互操作性。发送方的表示层把信息从与发送方相关的格式转换为一种公共的格式，而接收方的表示层把这种公共格式转换为与接收方相关的格式。

- **加密** 为了携带敏感信息，一个系统必须能够做到保密。加密就是发送方把原始信息转换为另一种形式，然后将转换后的报文发送到网络上。解密就是把原过程反过来，把加密信息恢复为原来的形式。
- **压缩** 数据压缩减少了信息中所包含的比特数。在传输多媒体信息（如文本、声音和视像）时，数据压缩就特别重要。

应用层

应用层 (application layer) 让用户（不管是人还是软件）能够接入网络。应用层给用户提供了接口，也提供多种服务支持，如电子邮件、远程文件访问和传送、共享数据库管理，以及其他多种类型的分布式信息服务。应用层提供的特定服务有以下一些：

- **网络虚拟终端** 网络虚拟终端是物理终端的软件版本，用来使用户能够登录到远程主机上。为此，这个应用程序创建一个软件对远程主机的终端进行仿真。用户的计算机先与这个软件终端交谈，然后这个软件终端再和主机交谈，反之亦然。远程主机认为它正在和自己的终端交谈，因此就允许你进行登录。
- **文件传送、存取和管理 (FTAM)** 这个应用程序允许用户访问远程主机中的文件（改变数据或读取数据）、将文件从远程计算机读取到本地计算机上来使用以及在本机上管理和控制远程计算机中的文件。
- **邮件服务** 这个应用程序提供转发和存储电子邮件的基本功能。
- **名录服务** 这个应用程序提供分布式数据库源，以及对全球各种对象和服务信息的存取。

2.2.5 OSI 各层小结

图 2.6 归纳总结了每一层的任务。下一节我们会讲到，有一部分任务是如何在 TCP/IP 协议族中被混合并重新划分为五类。

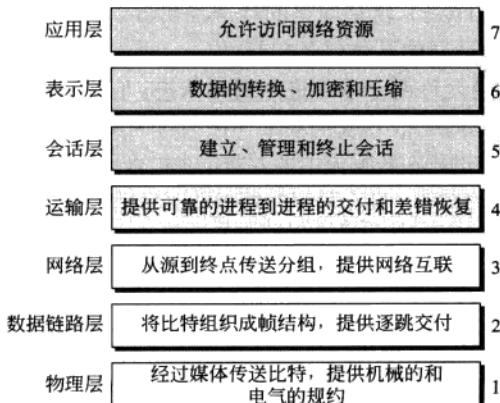


图 2.6 OSI 七层结构小结

2.3 TCP/IP协议族

TCP/IP协议族的开发要比**OSI**模型更早，因此**TCP/IP**协议族的分层结构无法准确地与**OSI**模型一一对应。原始的**TCP/IP**协议族定义为建立在硬件基础上的四个软件层，不过目前**TCP/IP**协议族被认为是一个五层模型，其层的命名类似于**OSI**模型中相应的层。图2.7分别描述了这两种结构。

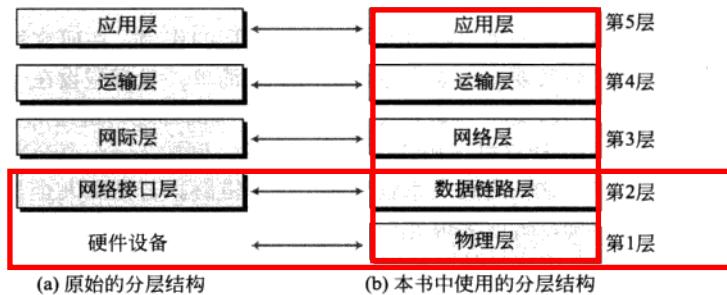


图2.7 TCP/IP协议族的分层结构

2.3.1 OSI和TCP/IP协议族的比较

当我们比较这两个模型时会发现，在**TCP/IP**协议族中没有会话和表示这两层。即使在**OSI**模型发布后，**TCP/IP**协议族也没有因此而增加这两层。在**TCP/IP**协议族中通常认为应用层是**OSI**模型的最高三层的合并，如图2.8所示。

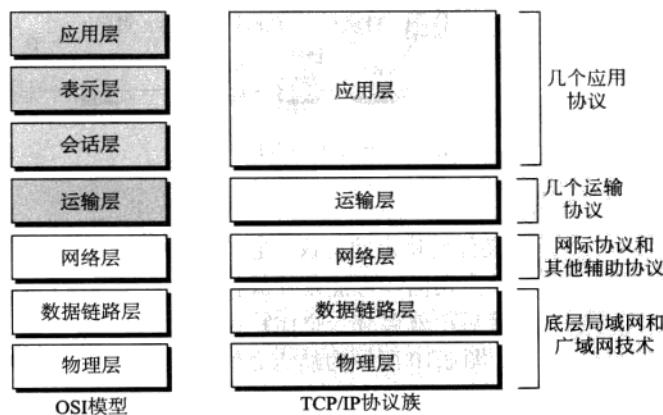


图2.8 TCP/IP和OSI模型

采用这一决定的理由有两个。首先，**TCP/IP**有多个运输层协议，而会话层的某些功能在一些运输层协议中已具备。其次，应用层并不仅仅是一个软件，这一层允许开发的应用

程序有很多。如果特定的应用程序需要用到会话层和表示层中某些相应的功能，那么这些功能也可以包含在该应用软件中进行开发。

TCP/IP 是一种分层协议，它由多个交互的模块构成，每个模块都提供了特定的功能，但是这些模块并不是必须互相依赖的。OSI 模型具体规定了哪一层应该具备哪些功能，而 TCP/IP 协议族的每一层则包含的是一些相对独立的协议，可以根据系统的需要把这些协议混合并重新搭配使用。术语“层次化”指的是每一个上层协议都由一个或多个下层协议来支持。

2.3.2 TCP/IP 协议族的分层

本节我们将简单地讨论一下 TCP/IP 协议族中各层的作用。在研究每一层所起的作用时，考虑一个专用的互联网要比考虑全球因特网更加简单。我们假设在一个小型的专用互联网内使用 TCP/IP 协议族。这种互联网是由若干个小型的称之为链路的网络组成的。一条链路（link）就是允许一组计算机互相通信的一个网络。例如，一个办公室中的所有计算机都通过网线连接在一起，这种连接关系就形成了一条链路。若某私人企业中的若干台计算机通过卫星信道互相连接，此连接也是一条链路。就像我们将在第 3 章中讨论的那样，一条链路有可能是服务小范围区域的 LAN（局域网），也有可能是服务很大范围区域的 WAN（广域网）。我们还要假设不同链路通过称为路由器或者交换机的设备互相连接在一起，这些设备会为数据选路以送抵它们最终的目的地。图 2.9 所示为我们虚构的一个互联网，用以说明 TCP/IP 各层的作用。图中有六条链路和四个路由器（R1 到 R4）。在这幅图中只画出了两台计算机，计算机 A 和计算机 B。

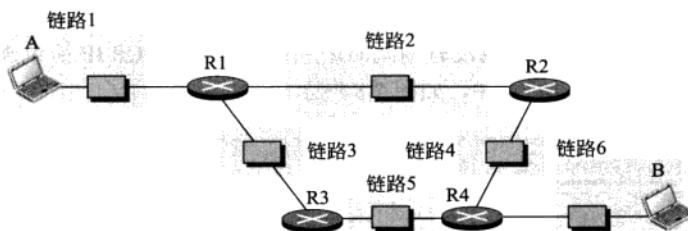


图 2.9 一个专用互联网

物理层

在物理层，TCP/IP 没有定义任何特定的协议。它支持所有标准的和专用的协议。在这一层，通信发生在两跳或两个结点之间，可能是计算机，也可能是路由器。通信以比特为单位。当两个结点之间建立连接后，就会有一个比特流在它们之间流动。但是对于物理层来说每个比特都将被独立对待。图 2.10 所示为结点之间的通信。我们假设此时两台计算机都已知道了与对方通信的最有效路径是经过路由器 R1、R3 和 R4。至于如何知道则属于后面的章节所要讨论的内容。

请注意，如果一个结点与 n 条链路相连，那么它需要 n 个物理层协议，每条链路各需要一个，原因在于不同的链路可能使用不同的物理层协议。图中只显示了此次通信所涉及到的物理层，每台计算机仅涉及一条链路，而每台路由器仅涉及两条链路。如图 2.10 所示，

这些比特在计算机 A 和计算机 B 之间的旅程是由四个独立的短途旅行构成的。计算机 A 以链路 1 所使用的协议格式向路由器 R1 发送各比特。路由器 R1 以链路 3 所使用的协议格式向路由器 R3 发送这些比特，依此类推。路由器 R1 有三个物理层（在我们假设的场景中只显示了两个），连接到链路 1 的物理层根据链路 1 所使用的协议的格式来接收这些比特，连接到链路 3 的物理层根据链路 3 所使用的协议的格式来发送这些比特。此次通信过程中涉及的其他两个路由器的情况也与此相同。

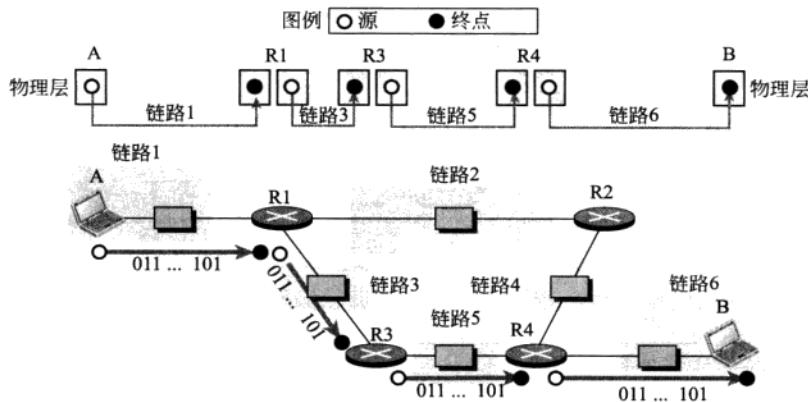


图 2.10 在物理层上的通信

物理层的通信单位是比特。

除了传送比特之外，物理层的其他任务也与 OSI 模型的物理层相对应，且主要取决于提供链路的底层技术。例如，在下一章中我们就会看到局域网和广域网的物理层有很多协议可用。

数据链路层

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专用的协议。在这一层，通信仍然发生在两跳或两个结点之间，不过通信的单位却是称为帧的分组。一个帧就是封装了来自网络层的数据的分组，并为其附加一个首部，有时还要再加一个尾部。在首部中，除了一些在通信时需要用到的信息之外，最主要的是包含了这个帧的源地址和目的地址。目的地址是为了指明正确的接收者，因为连接到该链路上的结点很可能不止一个，而有了源地址才有可能实现某些协议所要求的对帧的响应或确认。图 2.11 显示了数据链层上的通信。

请注意，在计算机 A 和路由器 R1 之间传送的帧有可能不同于路由器 R1 到路由器 R3 之间传送的帧。当路由器 R1 接收到一个帧后，就会将其交给数据链路层协议，如图中 R1 左侧所示。该帧被打开，数据被卸下，然后将这个数据交给 R1 右侧所示的数据链路层协议，从而生成一个将要发送到路由器 R3 的新帧。这样做的原因在于这两条链路（链路 1 和链路 3）可能使用不同的协议且要求不同格式的帧。还要注意，图中没有画出这些帧的物理传输过程，因为物理上的传输只发生在物理层。这两个结点上的数据链路层之间的通信是逻辑上的，而非物理上的。换言之，路由器 R1 上的数据链路层认为帧是从计算机 A 的数据链路层直接发送过来的，而实际上真正从 A 发送到 R1 的是一个比特流，它从 A 的物理层到达 R1 的物理层。因为在计算机 A 上，帧被转换成了一个比特流，而到了路由器 R1，这个比

特流又被转换回帧，所以在两个数据链路层看来，它们之间交换的就是一个帧。

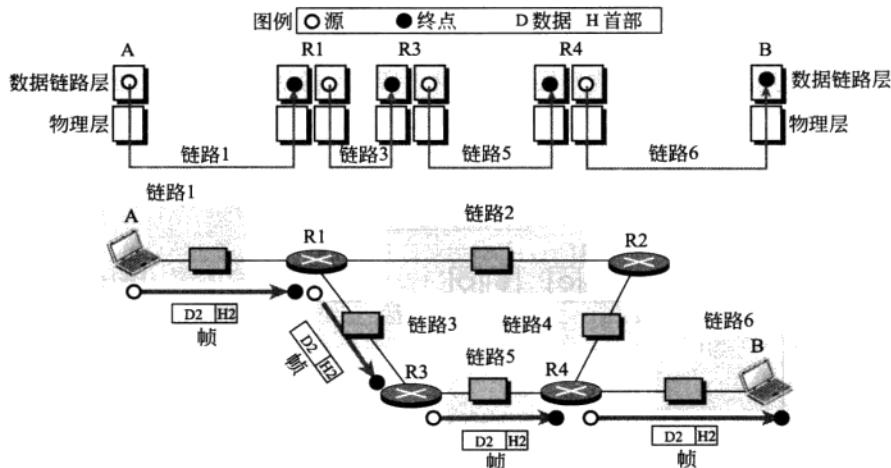


图 2.11 数据链路层上的通信

数据链路层的通信单位是帧。

网络层

在网络层(或者更准确地说是互联网层), TCP/IP 支持的是网际协议。网际协议 (Internet Protocol, IP) 是 TCP/IP 协议使用的传输机制。IP 传输的是称为 **数据报** (datagram) 的分组, 每个数据报独立传输, 不同的数据报可以走不同的路由, 也可能不按顺序地到达, 也可能重复。IP 不会跟踪记录这些数据报经过的路由, 并且在它们到达终点后, IP 也不具有按原顺序重排的能力。图 2.12 所示为在网络层上发生的通信。

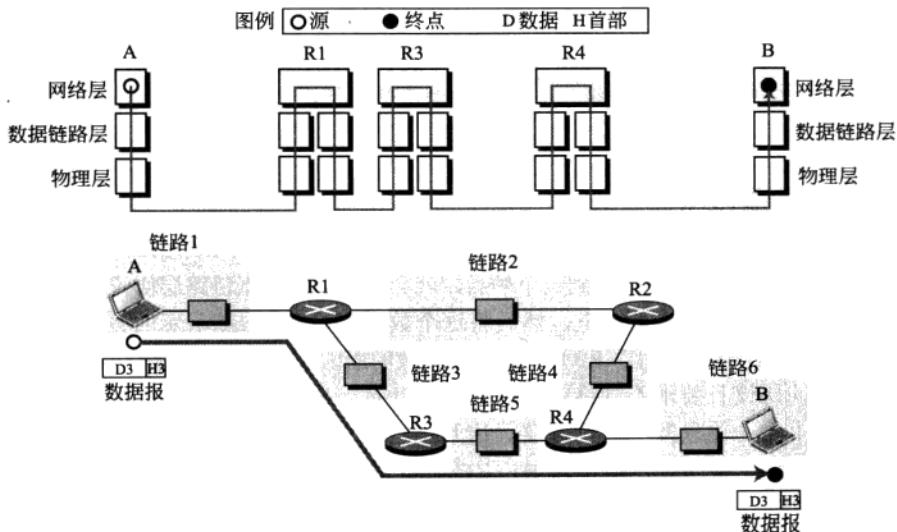


图 2.12 网络层上的通信

需要注意的是网络层的通信与数据链路层或物理层的通信之间有一个很大的区别。网络层上的通信是端到端的通信，而在另外两个层上的通信是结点到结点的。从计算机A出发的数据报就是最后到达计算机B的数据报。路由器上的网络层可以查看分组中的源地址和目的地址以查找最佳路由，但不允许更改分组中的内容。当然，此通信也是逻辑上的，而非物理上的。虽然计算机A和B的网络层认为它们正在发送和接收数据报，但真正的通信过程发生在物理层。

网络层的通信单位是数据报。

运输层

在运输层和网络层之间有一个重要区别，那就是网络中的所有结点都必须具有网络层，但只有两端的计算机上才需要运输层。网络层负责将一个个数据报独立地从计算机A发送到计算机B，而运输层则要负责将完整的报文（也称为报文段、用户数据报或分组）从计算机A交付给计算机B。一个报文段可能包含了几个到几十个数据报。报文段先要被拆分为多个数据报，然后再将这些数据报分别递交给网络层来传输。因为网际协议会为每个数据报指定不同的路由，所以这些数据报有可能会失序到达，也有可能丢失。计算机B的运输层需要等所有数据报全部抵达后进行重装，从而得到报文段。图2.13所示为发生在运输层的通信过程。

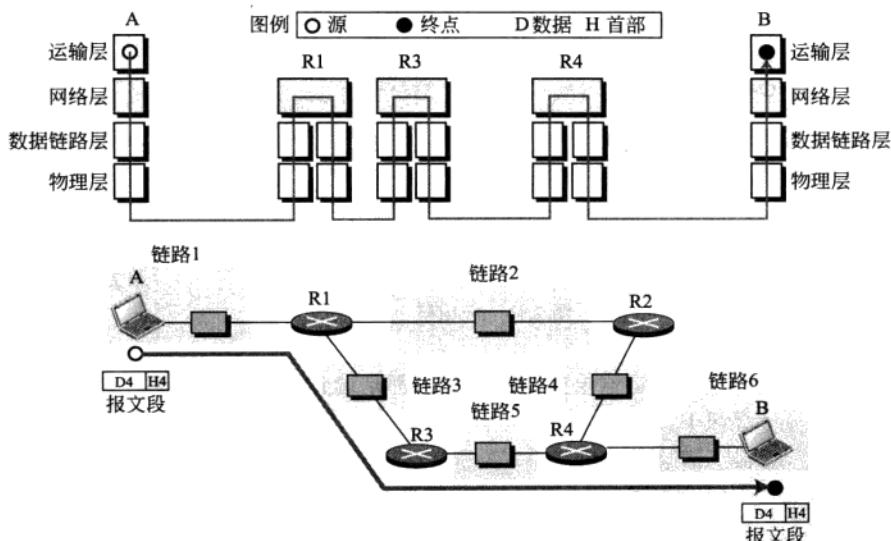


图2.13 运输层上的通信

同理可知，两个运输层会简单地认为它们之间正在用报文段互相通信，而事实上真正的通信发生在物理层，且被交换的是比特。

传统上，TCP/IP协议族中有两个运输层协议：用户数据报协议（User Datagram Protocol, UDP）和传输控制协议（Transmission Control Protocol, TCP）。近几年又出现了一种新的运输层协议称为流控制传输协议（Stream Control Transmission Protocol, SCTP）。

运输层的通信单位可以是报文段、用户数据报或者是分组，取决于运输层使用的具体协议。

应用层

TCP/IP 中的应用层相当于 OSI 模型中会话层、表示层和应用层的组合。应用层使用户能够获得网络所提供的服务，不论是此处所举的专用互联网还是全球因特网。这一层定义了许多协议以提供诸如电子邮件、文件传送以及访问全球万维网的服务。我们将在后面的章节中讨论其中的大部分标准协议。图 2.14 所示为发生在应用层的通信过程。

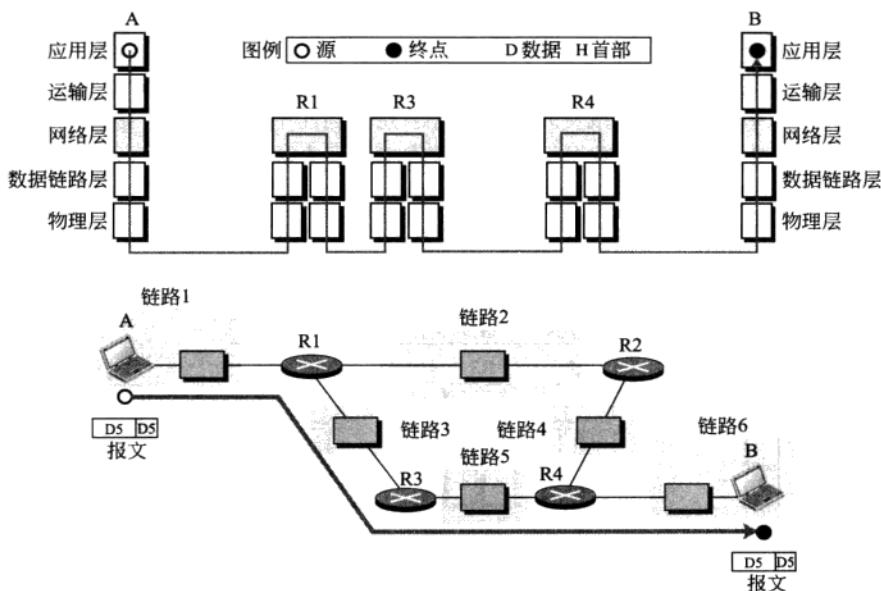


图 2.14 应用层上的通信

注意，应用层的通信与运输层的通信一样，都是端到端的。由计算机 A 生成的报文无修改地被传送到计算机 B。

应用层的通信单位是报文。

2.4 编址

实施了 TCP/IP 协议的互联网需要用到四个级别的地址：物理地址（physical address）、逻辑地址（logical address）、端口地址（port address）和特定应用地址（application-specific address）。每种地址都与 TCP/IP 体系结构中的某一层相关，如图 2.15 所示。

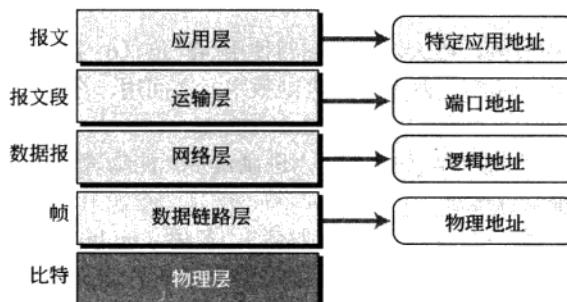


图 2.15 TCP/IP 协议族中的地址

2.4.1 物理地址

物理地址也称为链路地址，是由结点所在的局域网或广域网为该结点指定的地址。物理地址包含在数据链路层所使用的帧中。物理地址是最低一级的地址。物理地址仅对链路（局域网或广域网）有效。这种地址的长度和格式随网络的不同而变化。例如，以太网使用写在网络接口卡（NIC）里的6字节（48位）的物理地址。而LocalTalk（苹果公司）则使用一个字节的动态地址，它在站点每次入网时动态变化。

例 2.3

在图 2.16 中，物理地址为 10 的结点向物理地址为 87 的结点发送了一个帧。这两个结点通过一条链路（局域网）相连接。在数据链路层，帧的首部中包含了这两个物理（链路）地址，它们是此处唯一需要用到的地址。首部的其余部分则包含的是这一层所需要的其他一些信息。帧的尾部通常包含的是用于差错检测的一些附加位。如图所示，物理地址为 10 的计算机是发送方，物理地址为 87 的计算机是接收方。发送方的数据链路层收到来自上一层的数据。通过附加一个首部和一个尾部，它将数据封装成了一个帧。在首部中带有接收方和发送方的物理（链路）地址，以及其他一些信息。请注意，在大多数数据链路协议中目的地址（此处为 87）是放在源地址（此处为 10）前面的。这个帧通过局域网传播。每个物理地址不是 87 的站都会丢弃这个帧，因为帧中的目的地址与自己的物理地址不匹配，而真正的目的计算机会发现该帧中的物理地址与自己的物理地址完全匹配。于是这个帧经过检查后被卸掉首部和尾部进行解封，解封后的数据部分被交付给上一层。

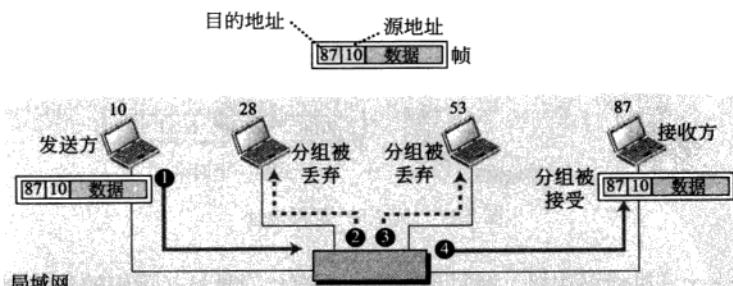


图 2.16 例 2.3: 物理地址

例 2.4

我们将在第 3 章中看到，绝大多数局域网使用 48 位（6 字节）的物理地址，并写成 12 个十六进制数字，每个字节（两个十六进制数字）之间用一个冒号分隔开，如下所示。

07:01:02:01:2C:4B

一个 6 字节（12 个十六进制数字）的物理地址

单播、多播和广播物理地址

物理地址可以是单播（unicast，单个接收者）、多播（multicast，一组接收者）或广播（broadcast，由网络中的所有系统接收）。有些网络支持所有这三种地址。例如以太网（见第 3 章）支持单播物理地址（6 字节）、多播地址和广播地址。有些网络则不支持多播或广播地址，此时如果必须把帧发送给一组接收者或网络中的所有系统时，多播或广播地址必须用单播地址来模拟，也就是说要用单播地址发送多个分组。

2.4.2 逻辑地址

逻辑地址对与底层物理网络无关的全局通信来说是必不可少的。在一个网际互连的环境下，仅有物理地址还是不够的，因为不同的网络可以使用不同的地址格式。我们需要一种全局性的编址系统用以唯一地标志每台主机，做到与底层的物理网络无关。逻辑地址就是为此而设计的。目前因特网上的逻辑地址是一个 32 位地址，可以用来唯一地标志连接在因特网上的每一台主机。在因特网上不存在两台具有相同 IP 地址的公开编址的实体主机。

例 2.5

图 2.17 所示为一个互联网的一部分，它由两台路由器连接三个局域网。

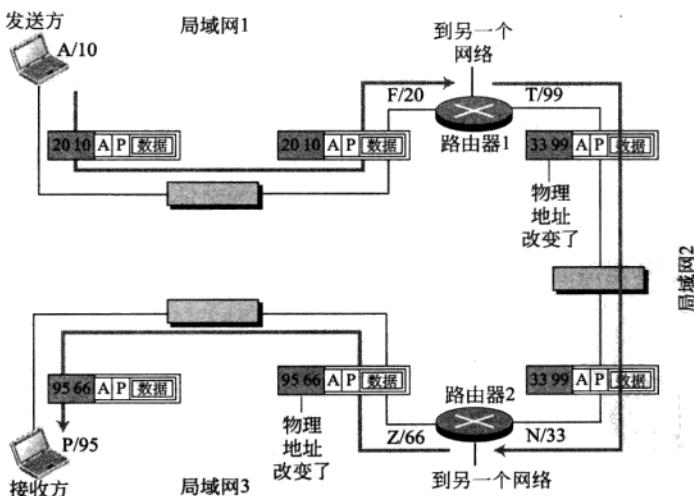


图 2.17 例 2.5：逻辑地址

每个设备（计算机或路由器）在每条连接上都有一对地址（逻辑的和物理的）。在本例中，两台计算机都只连接了一条链路，因而只有一对地址。不过每个路由器都连接到三个

网络上（图中只显示了两个），因此每个路由器都有三对地址，一条连接对应一对地址。虽然很容易理解为什么每个路由器必须为每条连接设置一个独立的物理地址，但是为什么每条连接都需要一个独立的逻辑地址呢？这一点可能不太好理解。我们将在第11章和12章讨论路由选择时再讨论这个问题。

网络地址为A，物理地址为10的计算机需要向网络地址为P，物理地址为95的计算机发送一个分组。这里我们用字母来表示逻辑地址，用数字来表示物理地址，但请注意实际上它们都是数字形式的，我们将在稍后的章节中再详细说明。

发送方在网络层将数据封装入一个分组中，并加入两个逻辑地址（A和P）。注意在大多数协议中，逻辑源地址是出现在逻辑目的地址之前的（与物理地址的顺序正好相反）。网络层必须先要找出下一跳的物理地址才能向下传递这个分组。网络层咨询它的路由表（参见第6章），并找出下一跳（路由器1）的逻辑地址是F。另外还有一个称为地址解析协议(Address Resolution Protocol, ARP)的协议，它会找出与该逻辑地址相对应的是路由器1的物理地址(20)，这个协议我们将在后面的章节中介绍。现在网络层就可以将这个地址传递给数据链路层了，再由数据链路层以物理目的地址20和物理源地址10来封装这个分组。

局域网1上的所有设备都会收到这个帧，但是除了路由器1之外，其他的路由器都会选择丢弃该帧，因为只有路由器1发现该帧中的物理目的地址与自己的物理地址是相匹配的。路由器1将该帧解封后得到数据分组，并读出其逻辑目的地址是P。由于这个逻辑目的地址与路由器1的逻辑地址不匹配，路由器1就知道了还要继续转发这个分组。路由器1也咨询它的路由表，而ARP再次找出下一跳（路由器2）的物理目的地址，然后又创建了一个新的帧将这个分组再次封装起来，并将其发送到路由器2。

请注意这个帧的物理地址。物理源地址从10更新为99。物理目的地址从20（路由器1的物理地址）更新为33（路由器2的物理地址）。逻辑源地址和逻辑目的地址必须保持不变，否则这个分组就会丢失了。

在路由器2我们又看到了类似的场景。物理地址被更新，而一个新的帧被发往目的计算机。当这个帧抵达终点后，分组被解封出来。逻辑目的地址P与该计算机的逻辑地址相匹配。分组解封后得到的数据被交付给上一层。请注意，虽然物理地址逐跳而变，但逻辑地址从源点到终点一直保持不变。这条规则也有一些例外的情况，我们将在本书稍后的地方讨论。

物理地址逐跳而变，逻辑地址保持不变。

单播地址、多播地址和广播地址

逻辑地址可以是单播地址（单个的接收者）、多播地址（一组接收者）和广播地址（网络中所有系统）。广播地址是有限制的。

端口地址

IP地址和物理地址对于将批量数据从源主机发送到目的主机来说是必不可少的，但到达目的主机并不是因特网数据通信的最终目标。一个系统若只是从一台计算机向另一台计算机发送数据，那任务还没有完成。如今的计算机是一种可以在同一时间运行多个进程的设备。因特网通信的最终目标是使一个进程能够和另一个进程通信。例如，计算机A能够与计算机C用TELNET进行通信，与此同时，计算机A还在与计算机B用FTP通信。为了使这些进程能够同时接收数据，我们需要用一种方法对不同的进程打上标号。换言之，

这些进程也需要有地址。在 TCP/IP 体系结构中，给一个进程指派的标号称为端口地址。TCP/IP 中的端口地址长度为 16 位。

例 2.6

图 2.18 所示为通过因特网正在通信的两台计算机。此时，正在发送数据的计算机上运行着三个进程，其端口地址分别为 a、b 和 c，而正在接收数据的计算机上则运行着两个进程，其端口地址分别为 j 和 k。发送方计算机上的进程 a 需要与接收方计算机上的进程 j 通信。注意，虽然两台计算机使用的是相同的应用程序，譬如 FTP，但它们的端口地址却是不同的，因为一个是客户端程序，另一个是服务器程序（参见第 17 章）。为了说明来自进程 a 的数据需要交付给进程 j，而不是进程 k，运输层要将来自应用层的数据封装成一个分组，并加入两个端口地址（a 和 j），源端口地址和目的端口地址。来自运输层的分组到了网络层又与逻辑源地址和逻辑目的地址（A 和 P）一起被封装成一个新的分组。最后，这个分组再次与物理源地址和下一跳的物理目的地址一起被封装成一个帧。在图中没有显示出物理地址，因为这些物理地址在标记为因特网的图示中是逐跳变化的。请注意，虽然物理地址逐跳而变，但逻辑地址和端口地址始终保持不变。这条规则也有一些例外的情况，将在本书稍后的地方讨论。

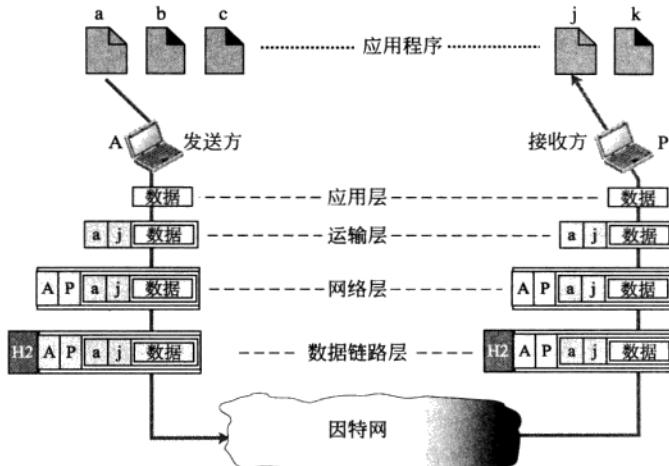


图 2.18 例 2.6：端口号

物理地址逐跳而变，但逻辑地址和端口地址通常保持不变。

例 2.7

我们将在后面的章节中看到，端口地址是由一个十进制数表示的 16 位地址，如下所示。

753

一个 16 位的端口地址被表示为一个十进制数

特定应用地址

有些应用程序具有专门为量身定做的用户友好型地址。这样的例子包括电子邮件地址（如 `forouzan@fhda.edu`）和 URL（如 `www.mhhe.com`）。第一个地址指定了一份电子邮件的接收者，第二个地址用来找到全球万维网中的一份文档。但是就像在后面的章节中将要看到的那样，发送数据的计算机会将这些地址统统转换为相应的端口地址和逻辑地址。

2.5 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

2.5.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]、[Pet & Dav 03]、[Kur & Ros 08]和[Gar & Vid 04]。

2.5.2 RFC

有两个 RFC 专门讨论了 TCP/IP 协议族：RFC 791 (IP) 和 RFC 817 (TCP)。在后面的章节中，我们将罗列与各层中的每一个协议相关的 RFC。

2.6 重要术语

接入控制	半双工方式
地址解析协议 (ARP)	接口
应用层	国际标准化组织 (ISO)
特定应用地址	网际互连
比特	线路配置
广播物理地址	链路
总线拓扑	逻辑地址
压缩	逻辑编址
连接控制	网状拓扑
数据报	多播物理地址
数据链路层	多点配置
对话控制	网络层
名录服务	网络虚拟终端
编码	开放系统
加密	开放系统互连 (OSI)
差错控制	对等进程
文件传送、存取和管理 (FTAM)	物理地址
流量控制	物理层
帧	物理拓扑
全双工方式	点对点配置

端口地址	流控制传输协议（SCTP）
表示层	同步点
进程到进程交付	TCP/IP 协议族
环形拓扑	转换
路由选择	传输控制协议（TCP）
分段	传输方式
服务点编址	传输速率
会话层	运输层
单工方式	单播物理地址
源点到终点交付	用户数据报协议（UDP）
星形拓扑	

2.7 本章小结

- 国际标准化组织（ISO）创建了一个称为开放系统互连（OSI）的模型，它使得不同系统之间可以互相通信。七层 OSI 模型为开发普遍兼容的连网协议提供了指导。物理层、数据链路层和网络层是网络支撑层。会话层、表示层和应用层是用户支撑层。运输层把网络支撑层和用户支撑层链接起来。
- 物理层协调在物理媒体上传送比特流所需的各种功能。数据链路层负责把数据单元无差错地从一个站交付到下一个站。网络层负责跨越多个网络链路的从源点到终点的分组交付。运输层负责从进程到进程的完整报文交付。会话层用于建立、维持并同步正在通信的系统之间的交互。表示层通过把数据转换为双方同意的格式，确保相互通信的设备之间的互操作性。应用层使用户能够接入到网络。
- TCP/IP 是在 OSI 模型之前开发的具有分层体系结构的五层协议族。TCP/IP 的应用层相当于 OSI 模型的会话层、表示层及应用层的合体。
- 使用 TCP/IP 协议的系统需要四种类型的地址：物理地址、网际协议地址（IP 地址）、端口地址和特定应用地址。物理地址又称为链路地址，是由结点所在的局域网或广域网为该结点指定的地址。IP 地址唯一地定义了因特网上的一台主机。端口地址指明主机上的进程。特定应用地址被某些应用程序用来提供用户友好式接入。

2.8 实践安排

2.8.1 习题

1. OSI 和 ISO 有怎样的关系？
2. 试将下述功能与 OSI 模型的一层或几层相对应：
 - a. 确定路由
 - b. 流量控制

- c. 到传输媒体的接口
 - d. 为端用户提供接入
3. 试将下述功能与 OSI 模型的一层或几层相对应:
- a. 可靠的进程到进程的报文交付
 - b. 路由选择
 - c. 定义帧
 - d. 向用户提供服务, 如电子邮件和文件传送
 - e. 在物理媒体上传送比特流
4. 试将下述功能与 OSI 模型的一层或几层相对应:
- a. 直接和用户的应用程序通信
 - b. 纠错和重传
 - c. 机械的、电气的和功能的接口
 - d. 在相邻结点之间运载帧的任务
5. 试将下述功能与 OSI 模型的一个层或几个层相对应:
- a. 格式和代码的转换服务
 - b. 建立、管理和终止会话
 - c. 确保可靠的数据传输
 - d. 登录和注销的过程
 - e. 使数据独立于数据表示的差异
6. 描绘图 2.19 所示的简单专用互联网应用层的通信过程 (参见图 2.14)。

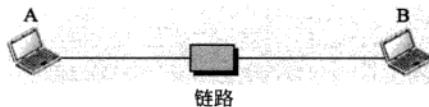


图 2.19 习题 6

7. 描绘图 2.20 所示的简单专用互联网应用层的通信过程 (参见图 2.14)。

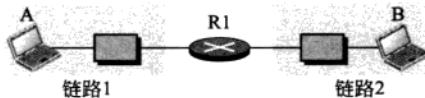


图 2.20 习题 7

8. 使用 TCP/IP 协议族通过专用互联网发送一个 100 字节的报文。如果每一层的协议都为其附加一个 10 字节的首部, 那么这个系统的效率 (有用的字节数占全部字节数的比值) 是多少?
9. 如果端口号为 16 位 (2 字节), 那么 TCP/IP 协议族运输层首部的最小尺寸是多少?
10. 如果逻辑地址为 32 位 (4 字节), 那么 TCP/IP 协议族网络层首部的最小尺寸是多少?
11. 如果物理地址为 48 位 (6 字节), 那么 TCP/IP 协议族数据链路层首部的最小尺寸是多少?
12. 当我们给朋友寄一封普通的信时是否需要将我们的报文封装起来? 当我们在其他国家度假期间给朋友寄明信片时, 是不是也需要封装起来?

13. 你认为为什么在物理层不需要地址？
14. 你认为为什么电台在广播时不需要听众的地址？
15. 你认为为什么在因特网上发送方地址和接收方地址都是必须的？
16. 你认为为什么在因特网上需要有四级地址，但是在电话网中只需要一级地址（电话号码）？

2.8.2 研究活动

17. 域名系统或 DNS（见第 19 章）是 TCP/IP 协议族中的一个应用程序。试进行一些研究并找出在 OSI 模型中和这个协议等效的协议（如果有的话），并对这两个协议进行比较。
18. 文件传送协议 FTP（见第 21 章）是 TCP/IP 协议族中的一个应用程序。试进行一些研究并找出在 OSI 模型中和这个协议等效的协议（如果有的话），并对这两个协议进行比较。
19. 简单文件传送协议 TFTP（见第 21 章）是 TCP/IP 协议族中的一个应用程序。试进行一些研究并找出在 OSI 模型中和这个协议等效的协议（如果有的话），并对这两个协议进行比较。
20. 在 OSI 模型中提出了多个运输层模型的建议。试进行一些研究并找出全部这些模型。说明这些模型之间的区别。
21. 在 OSI 模型中提出了多个网络层模型的建议。试进行一些研究并找出全部这些模型。说明这些模型之间的区别。

第3章 底层技术

我们可以把因特网看成由许多主干网组成，而这些主干网由一些国际的、国家的和地区的 ISP 来运营。主干网通过一些连接设备（如路由器或交换机）相互连接在一起。端用户或者属于本地 ISP 局域网中的一部分，或者是通过点到点网络连接到该局域网上。从概念上讲，因特网就是由交换广域网（主干网）、局域网、点到点广域网，以及连接设备或交换设备共同组成的集合。

虽然 TCP/IP 协议族通常都被表示为一个五层的协议栈，但实际上它只定义了上三层，因为 TCP/IP 只关心网络层、运输层和应用层。这就表示，TCP/IP 假定这些广域网、局域网已经存在，并且连接这些网络的连接设备也已具备。

作为简单的回顾，我们将在本章中提纲挈领地对这些底层技术进行介绍。

目标

本章有以下几个目标：

- 简单地讨论在有线局域网中占主导地位的以太网技术，包括传统以太网，快速以太网，吉比特以太网和 10 G 以太网。
- 简单地讨论无线局域网技术，包括 IEEE 802.11 和蓝牙技术。
- 简单地讨论点到点广域网技术，包括 56 K 调制解调器、DSL、电缆调制解调器、T 线技术和 SONET 技术。
- 简单地讨论交换广域网技术，包括 X.25、帧中继和 ATM。
- 讨论对连接设备的要求和用途，如转发器（或集线器）、网桥（两层交换机）和路由器（三层交换机）。

3.1 有线局域网

局域网（LAN）就是设计在有限地理范围内使用的计算机网络，如一座建筑物内或一所院校内。虽然局域网可以是某组织内部仅仅为了达到共享资源而将计算机连接起来的一个孤立的网络，不过今天的局域网绝大多数都会链接到一个广域网或者因特网上。

在局域网的市场上出现过多种技术，如以太网，令牌环，令牌总线，FDDI 和 ATM 局域网等等。其中有些技术也存在过一段时间，但以太网才是占有绝对优势的技术。

本节我们要简单地讨论一个 IEEE 的 802 号项目标准，它的设计是为了协调不同局域网之间在制造以及互连方面存在的问题。然后我们再集中讨论以太局域网。

在过去的二、三十年中，虽然以太网已经发展演变成了四代，但其基本思想始终如一。以太网的发展变化是为了迎合市场的需求，也是为了充分利用新的技术。

3.1.1 IEEE 标准

1985 年，为了建立一些标准使得来自不同生产厂商制造的设备之间能够互相通信，IEEE 计算机协会启动了一个项目，称为 **802 项目**（Project 802）。802 项目并不想取代 OSI 模型或因特网模型中的任何一部分，相反，它是用来指明主要局域网协议中物理层和数据链路层功能的一种途径。

这个标准被美国国家标准学会（ANSI）采纳。1987 年，国际标准组织（ISO）也批准其作为编号为 ISO 8802 的国际标准。

802 标准与传统的 OSI 模型之间的关系如图 3.1 所示。IEEE 将数据链路层进一步划分为两个子层：**逻辑链路控制（Logical Link Control, LLC）** 和 **媒体接入控制（Media Access Control, MAC）**。IEEE 还为不同的局域网协议建立了多个物理层的标准。

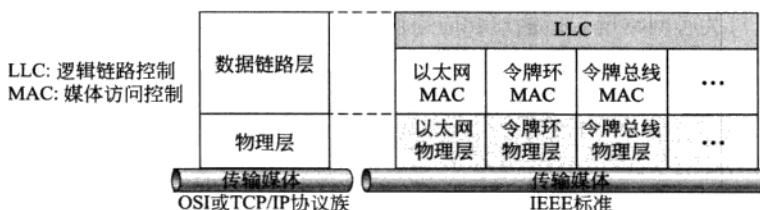


图 3.1 局域网的 IEEE 标准

不过，在本书中我们认为物理层和数据链路层都是底层技术，它们一起支撑了 TCP/IP 协议族中的其他层。有关物理层和数据链路层技术的更具体的细节请参阅 Forouzan, *Data Communications and Networking*, 4th ed., McGraw-Hill, 2007。

3.1.2 帧格式

以太 LAN 网中发送的分组称为帧。本节要讨论几种我们关心的以太网版本中所使用的帧的格式及长度。

帧格式

以太网的帧包含了 7 个字段：前同步码、SFD、DA、SA、数据单元的长度/类型、上层数据以及 CRC。**以太网不提供任何机制来确认收到的帧**，因此以太网是一种被称为不可靠的媒体。MAC 帧的格式如图 3.2 所示。

前同步码: 56 比特交替出现的 1 和 0
SFD: 帧首定界符, 标志 (10101011)

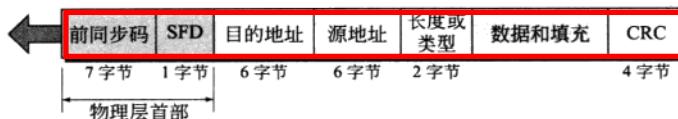


图 3.2 以太网的帧

- **前同步码** 802.3 帧的第一个字段包含的是 7 个字节(56 比特)交替出现的 0 和 1，它的作用就是提醒接收系统有帧到来，并且使它与输入定时同步。此样式仅仅是为了提供一个通知及定时的脉冲。这种 56 比特的样式允许接收站错过帧最前面的几个比特。前同步码实际上是在物理层添加上去的，它并不是(正式的)帧的一部分。
- **帧首定界符 (SFD)** 第二个字段(1 字节: 10101011)作为帧开始的信号。SFD 提醒接收站这是最后一次进行同步的机会。最后两个比特是 11，就是提醒接收方接下来的字段就是目的地址。SFD 也是由物理层添加的。
- **目的地址 (DA)** DA 字段有 6 字节，包含的是目的站或者将要接收该分组的站的物理地址。我们马上会讨论编址问题。
- **源地址 (SA)** SA 字段也是 6 字节，包含的是这个分组的发送设备的物理地址。
- **长度/类型** 长度/类型字段被定义为类型字段或者长度字段。最初的以太网将此字段用作类型字段，以定义使用该 MAC 帧的上层协议。而 IEEE 标准将其作为长度字段，用来指明在数据字段中所包含的字节数目。这两个使用方式目前都很常见。
- **数据** 数据字段携带的是被上层协议封装的数据。它的最小长度是 46 字节，最大长度是 1500 字节，稍后我们再加以解释。
- **CRC** 最后一个字段包含的是差错检测信息，此处使用的是 CRC-32(见附录 C)。

帧长度

以太网对帧的最小长度和最大长度都有严格规定，如图 3.3 所示。

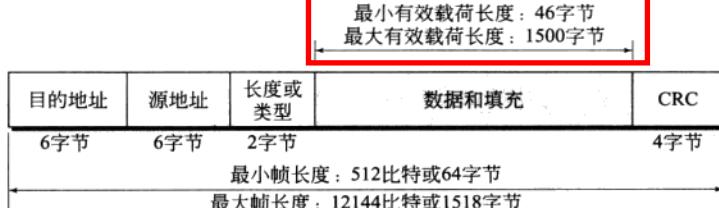


图 3.3 最小和最大长度

限制最小长度是为了使 CSMA/CD 能够正确操作而要求的，稍后我们将会介绍。一个以太网的帧最少需要有 512 比特或 64 字节的长度。这个长度中有一部分是首部和尾部的长度。如果首部和尾部总共算作 18 字节(6 个字节的源地址，6 个字节的目的地址，2 个字节的长度/类型字段，再加上 4 个字节的 CRC)，那么来自上层的数据的最小长度是 $64 - 18 = 46$ 字节。如果上层的分组少于 46 字节，那么就需要用填充来弥补差距。

标准定义一个帧的最大长度(不算前同步码和 SFD 字段)是 1518 个字节。如果减去首部和尾部的 18 个字节，最大有效载荷长度是 1500 个字节。最大长度的限制有两个历史的原因。首先，在最初设计以太网时，内存是非常昂贵的，而限制最大长度有助于减少缓存的大小。其次最大长度的限制可以防止一个站垄断了共享媒体，阻止其他需要发送数据的站发送。

最小长度：64字节(512比特) 最大长度：1518字节(12144比特)。

3.1.3 编址

以太网中的每一个站（如 PC、工作站或打印机）都有自己的网络接口卡（Network Interface Card, NIC）。NIC 通常安装在站的内部，并为该站提供一个 6 字节的物理地址。如图 3.4 所示，以太网的地址为 6 字节（48 比特）长，通常写成十六进制记法（hexadecimal notation），并且用冒号将字节和字节分隔开。这个地址通常也称为数据链路地址、物理地址或 MAC 地址。

例如，以下所示为一个以太网的 MAC 地址：

4A:30:10:21:10:1A

单播、多播和广播地址

源地址始终是单播地址，因为任何帧都只可能来自一个站。而目的地址则有可能是单播、多播或者广播地址。图 3.5 所示为如何区分一个单播地址和一个多播地址。如果目的地址的第一个字节的最低位是 0，那么这个地址就是单播地址。反之则是多播地址。



图 3.5 单播和多播地址

地址字段中第一个字节的最低位指明了该地址的类型。

如果该比特是 0 就是单播地址，反之则为多播地址。

单播地址仅指定了一个接收者，也就是说发送方和接收方之间是一对一的关系。多播地址指定的是一组地址，也就是说发送方和接收方之间是一对多的关系。

广播地址是多播地址的一种特殊形式，它的接收者是该局域网中所有的站。一个广播目的地址就是由 48 个 1 形成的地址。

广播目的地址是多播地址的一种特殊形式，其中所有位都是 1。

例 3.1

指出以下目的地址的类型：

- a. 4A:30:10:21:10:1A
- b. 47:20:1B:2E:08:EE
- c. FF:FF:FF:FF:FF:FF

解

要找出一个地址的类型，我们需要查看的是左手第二个十六进制数字。如果它是偶数，那么这个地址就是单播地址。如果它是奇数，那这个地址就是多播地址。如果所有数字都是 F，那么这个地址就是广播地址。因此我们可以得到：

这里指的是 MAC 地址，
不是 IP 地址那种...

- a. 这是一个单播地址，因为 A 用二进制表示就是 1010（偶数）。
- b. 这是一个多播地址，因为 7 用二进制表示就是 0111（奇数）。
- c. 这是一个广播地址，因为所有数字都是 F。

这些地址在发送到线路上时的方式与它们写成十六进制时的表示方式是不同的。在发送时从左到右逐字节地发送。但是对每一个字节来说，最先发送的是最低位，最后发送的是最高位。也就是说指定地址是单播地址还是多播地址的那一位会最先到达接收方。

例 3.2

描绘出地址 47:20:1B:2E:08:EE 是如何发送到线路上去的。

解

该地址是从左到右，逐字节地发送，而对每个字节来说又是从右到左，逐位发送，如下如示：

←11100010 00000100 11011000 01110100 00010000 01110111

3.1.4 以太网的发展历程

以太网是由施乐公司 Palo Alto 研究中心 (PARC) 于 1976 创建的。自那以后，它已经发展了四代：标准以太网（10 Mbps）、快速以太网（100 Mbps）、吉比特以太网（1 Gbps）和 10 G 以太网（10 Gbps），如图 3.6 所示。我们将简单地讨论所有这四代以太网，当然要从第一代，即标准（或传统）以太网说起。

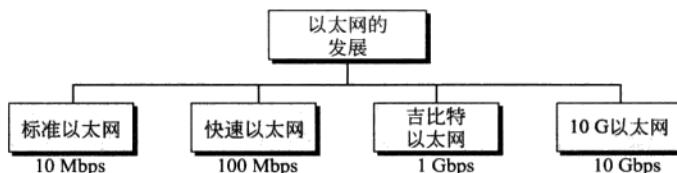


图 3.6 以太网经过了四代的发展

3.1.5 标准以太网

数据率为 10 Mbps 的早期以太网现在已成为历史，但是我们还是要简单地讨论一下它的特点，为理解以太网的其他几个版本铺平道路。

接入方法：CSMA/CD

IEEE 802.3 标准定义了带碰撞检测的载波侦听多点接入（carrier sense multiple access with collision detection, CSMA/CD）作为传统以太网的接入方法。在传统的以太网中，各站在物理上通过总线拓扑或星形拓扑连接在一起，但是其逻辑上的拓扑结构一定是总线的。这句话的意思是说：所有站共享媒体（信道），并且一次只能由一个站使用这个媒体。这也表示由某一个站所发送的帧将被所有站接收（广播方式）。只有真正的目的站才收下这个帧，而其他站都会丢弃。在这种情况下，我们怎样才能保证两个站不会在同一时间使用媒体呢？如果两个站同时使用媒体，它们发送的帧就会发生碰撞。

为了使发生碰撞的机会减至最小，并以此来提高性能，因而开发了 CSMA/CD 方法。如果一个站在试图占用媒体之前先侦听一下它，发生碰撞的机会就会减小。载波侦听多点接入 (carrier sense multiple access, CSMA) 要求每个站在发送之前先要对媒体进行侦听（或检查媒体的状态）。换句话说，CSMA 依据的是原则“传送前先侦听”或“先听后讲”。CSMA 能够降低发生碰撞的可能性，但不能消除碰撞。原因如图 3.7 所示，图中是一个 CSMA 网络的空间和时间模型。所有站都与一条共享信道（通常是专用媒体）相连接。

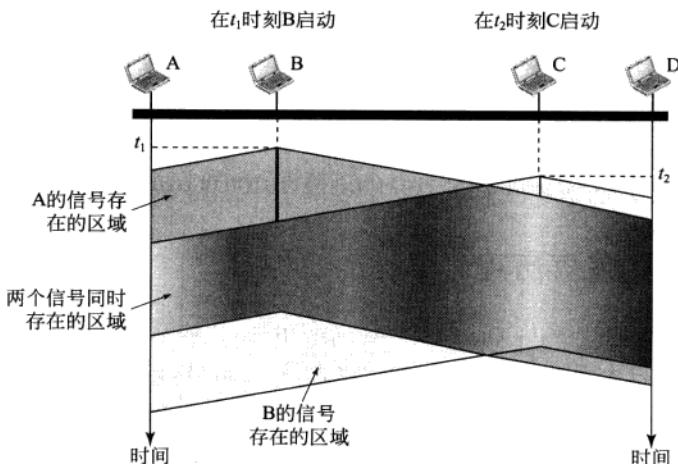


图 3.7 CSMA 中发生碰撞的空间/时间模型

由于传播时延的存在，发生碰撞的可能性依然存在。当某站发送了一个帧之后，该帧的第一个比特要到达所有站并使每个站都侦听到它的存在是需要花时间的。换句话说，虽然一个站在侦听媒体时发现它是空闲的，但很有可能这仅仅是因为另一个站发送的帧的第一个比特还没有被接收到。

在 t_1 时刻，站 B 侦听媒体并发现它是空闲的，因此站 B 就发送了一个帧。在 t_2 时刻 ($t_2 > t_1$)，站 C 侦听媒体并发现它是空闲的，因为此时由站 B 发送的帧的第一个比特还没有到达站 C。于是站 C 也发送了一个帧。两个信号发生碰撞，致使两个帧都被损坏。

带碰撞检测的载波侦听多点接入 (CSMA/CD) 对算法进行了修正以解决碰撞问题。在这种方法下，一个站在发送出去一个帧之后还要继续侦听信道，以确定此次传输是否成功。如果成功了则任务结束，但是如果出现了碰撞，还要再次发送这个帧。为了更好地理解 CSMA/CD，让我们考察一下两个站所传输的帧的前几个比特发生碰撞的情况。虽然这两个站在检测到碰撞之前仍然会不断发送该帧的剩余比特，但是我们只描述前几个比特发生碰撞时的情况。在图 3.8 中，站 A 和站 C 发生了碰撞。

在 t_1 时刻，站 A 开始发送帧，而在 t_2 时刻，站 C 还没有侦听到 A 发送的帧的第一个比特，于是站 C 开始发送它的帧，该帧左右双向传播开来。碰撞发生在 t_2 时刻之后的某个时刻。当站 C 在 t_3 时刻接收到来自站 A 的帧的第一个比特时就检测到了碰撞。站 C 立即（或者短时间内，不过我们假设是立即）放弃传输。当站 A 在 t_4 时刻接收到来自站 C 的帧的第一个比特时也检测到了碰撞，同样站 A 也立即放弃传输。如图所示，我们看到

站 A 传输的持续时间为 $t_4 - t_1$ ，而站 C 传输的持续时间为 $t_3 - t_2$ 。稍后我们将会看到，要使这个协议正常工作，任何一个帧的长度除以比特率都必须大于这两个持续时间的长度。在 t_4 时刻站 A 尚未完成的帧传输被放弃。同样在 t_3 时刻来自站 C 的尚未完成的帧传输被放弃。

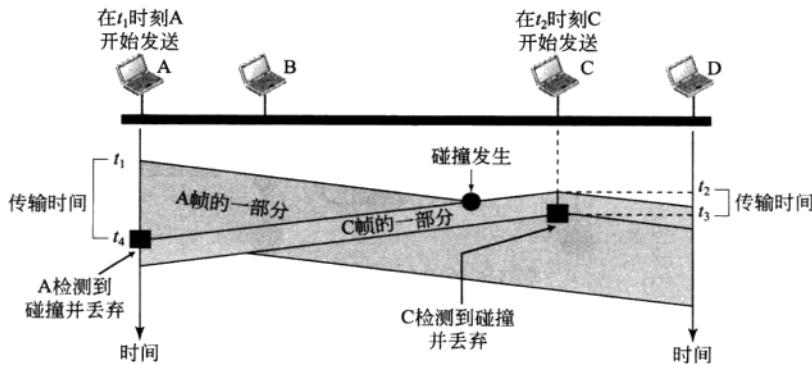


图 3.8 CSMA/CD 中第一个比特发生碰撞的情况

最小帧长

要使 CSMA/CA 正常工作，我们必须要限制帧的长度。如果某次传输发生了碰撞，那么正在发送数据的站必须在发送该帧的最后一比特之前放弃此次传输，因为一旦整个帧都被发送出去，那么该站将不会保留帧的复本，同时也不会继续监视是否发生了碰撞。因此帧的传输时间 T_{fr} 必须至少是最大传播时间 T_p 的两倍。要想了解其原因，让我们来设想一个最糟糕的场景。如果发生碰撞的两个站之间的距离最远，那么来自第一个站的信号需要花费 T_p 时间才能到达第二个站，而碰撞带来的影响又花了另一个 T_p 的时间才到达第一个站。因此，我们的要求就是第一个站必须在 $2T_p$ 长的时间后仍然在传输数据。

例 3.3

在标准以太网中，如果最大传播时间是 $25.6\mu s$ ，那么帧的最小长度是多少？

解

帧的传输时间是 $T_{fr} = 2 \times T_p = 51.2\mu s$ 。也就是说，以最坏的情况下，一个站需要经过长达 $51.2\mu s$ 的传输时间才能检测到碰撞，因此这个帧的最小长度是 $10 \text{ Mbps} \times 51.2\mu s = 512$ 比特或 64 字节。这实际上就是标准以太网中帧的最小长度，之前我们曾提到过。

处理过程

图 3.9 所示为 CSMA/CD 的流程图。在开始发送一个帧之前需要先侦听一下该信道。我们并不是在发送完整个帧之后再检测碰撞，实际上，一个站的发送和接收是连续的且并行的（使用两个不同的端口）。我们用一个循环来表示传输是连续的过程。我们不断地侦听信道以检测到以下两种状态之一：或者是传输完成，或者是检测到碰撞。这两个事件都会使传输停止。当我们跳出循环后，如果发现没有检测到碰撞，那么就说明传输完成了，完整的帧被发送出去，否则就是有碰撞发生了。该流程图中还显示出了一个短暂的干扰信号（jamming signal），它的作用是加强碰撞效果，以防止其他站侦听不到该碰撞。

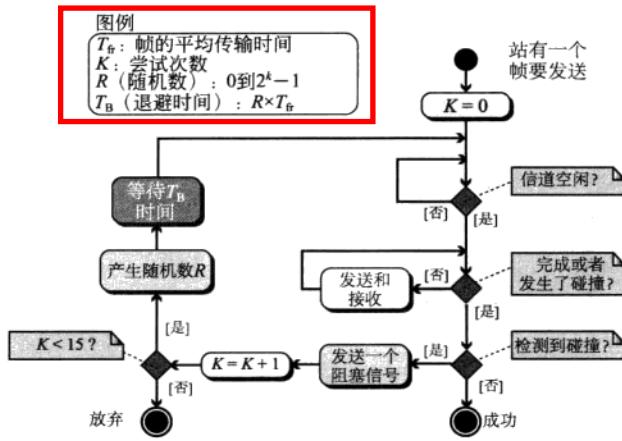


图 3.9 CSMA/CD 流程图

实现

标准以太网定义了多种实现方法，但其中只有 4 种方法在 20 世纪 80 年代流行过一段时间。表 3.1 给出的是标准以太网实现方法的小结。对于 10Base-X 这个命名来说，第一个数字指定了数据率（10 Mbps），术语 Base 代表基带（数字）信号，而 X 则指定了此缆线以 100 米为单位的最大约等于长度（例如，5 表示 500 米；2 表示 185 米），或者表示了此缆线的类型，T 表示无屏蔽双绞线缆（UTP），而 F 表示光缆。

表 3.1 标准以太网实现方法小结

特性	10Base5	10Base2	10Base-T	10Base-F
媒体	粗同轴电缆	细同轴电缆	2 对线 UTP	2 芯光缆
最大长度	500 m	185 m	100 m	2000 m

图 3.10 所示为每种实现方法的简单示意图。

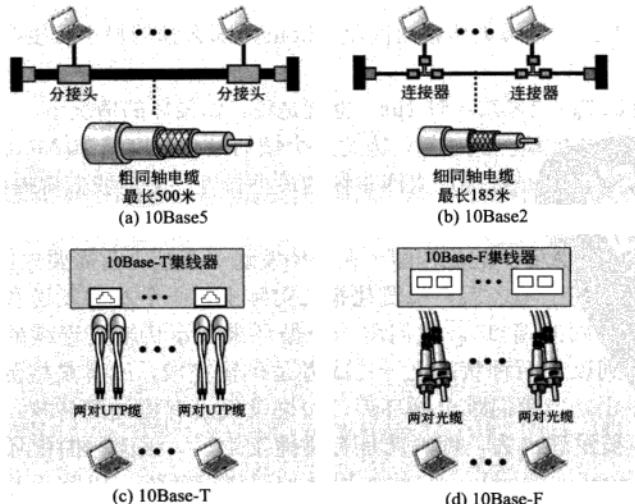


图 3.10 标准以太网的实现

3.1.6 快速以太网

之所以设计快速以太网是为了与其他局域网协议（如 FDDI 或光通道）进行竞争。IEEE 建立的快速以太网命名为 802.3u。快速以太网可向下兼容标准以太网，但它的传输速率要快 10 倍，达到 100 Mbps 的速率。快速以太网的目标总结如下：

1. 数据率升级到 100 Mbps。
2. 使之与标准以太网兼容。
3. 保持相同的 48 比特地址不变。
4. 保持相同的帧格式不变。
5. 保持相同的最小帧长和最大帧长不变。

MAC 子层

在以太网从 10 Mbps 发展到 100 Mbps 的过程中一个最主要的考虑是要原封不动地保留 MAC 子层。不过它还是放弃了总线拓扑结构，而仅保留星形拓扑结构。对于星形拓扑结构，可以有两个选择：半双工和全双工。在半双工方式下，所有站都通过一个集线器连接。而在全双工方式下，连接是通过交换机实现的，且交换机的每个端口都具有缓存（参阅本章最后的 3.5 节，连接设备）。

对于半双工方式来说，接入方法保持不变（CSMA/CD），但全双工快速以太网就不再需要 CSMA/CD 了。不过，为了向下兼容标准以太网，在实现上仍然保留了 CSMA/CD。

自动协商

快速以太网新增的一个特性称为自动协商（autonegotiation）。它允许一个站或一个集线器具有更广泛的性能。自动协商允许两个设备对操作的模式或数据率进行协商。这是针对以下目的而专门设计的：

- 允许不兼容的设备互相连接。例如，一个最大容量为 10 Mbps 的设备能够与最大容量为 100 Mbps 的设备互相通信（但只能工作在较低的那个数据率上）。
- 允许一个设备具有多种能力。
- 允许一个站检查集线器的能力。

实现

快速以太网在物理层的实现可划分为二线和四线两大类。二线实现可以使用屏蔽双绞线电缆 STP (100Base-TX) 或光缆 (100Base-FX)。四线实现的设计只能使用非屏蔽双绞线电缆 UTP (100Base-T4)。表 3.2 为快速以太网实现方法的小结。

表 3.2 快速以太网实现方法小结

特性	100Base-TX	100Base-FX	100Base-T4
媒体	STP	光缆	UTP
线数	2	2	4
最大长度	100 m	100 m	100 m

图 3.11 所示为每种实现方法的简单示意图。

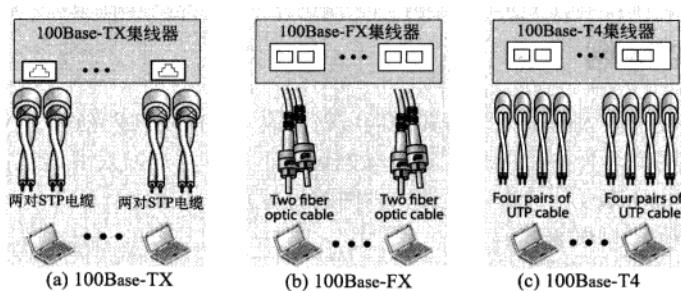


图 3.11 快速以太网的实现

3.1.7 吉比特以太网

当我们需要更高的数据率时就设计出了吉比特以太网协议（1000 Mbps）。IEEE 委员会称之为标准 802.3z。设计吉比特以太网的宗旨总结如下：

1. 数据率升级到 1 Gbps。
 2. 使之与标准以太网和快速以太网兼容。
 3. 使用相同的 48 比特地址。
 4. 使用相同的帧格式。
 5. 保持相同的最小帧长和最大帧长不变。
 6. 支持快速以太网中定义的自动协商功能。

MAC 子层

在以太网的发展过程中一个最主要的考虑是要原封不动地保留 MAC 子层，但是为了达到 1 Gbps 的数据率，这个要求显得不太可能。吉比特以太网有两种截然不同的媒体接入方式：半双工和全双工。几乎所有吉比特以太网都是按全双工方式实现的，但我们还是要简单地讨论一下半双工方式，以说明吉比特网能够与前两代以太网兼容。

全双工模式 在全双工模式下，有一个中央交换机连接到所有计算机和其他交换机。这种模式的每台交换机上的每个端口都具有缓存，数据会保存在这里直至被发送出去。在这种模式下不存在碰撞，也就是说 CSMA/CD 没有用武之地。没有碰撞还暗示着电缆的最大长度是由电缆中的信号衰减来决定的，而不是由碰撞检测过程决定的。

在全双工模式下的吉比特以太网中不存在碰撞：

电缆的最大长度是由电缆中的信号衰减决定的。

半双工模式 吉比特以太网也可以使用半双工模式，虽然非常少见。在这种情况下可以用集线器来代替交换机，它的作用就好像是普通电缆一样，此时就有可能发生碰撞。半双工方式使用 CSMA/CD。不过，正如我们在前面讲过的，这种方式下网络的最大长度完全取决于帧的最小长度。已定义的解决方案有三种：传统的、载波扩充和帧突发。

□ 传统的 在传统方式下，我们保持帧的最小长度与传统以太网一致（512 比特）。但是由于在吉比特以太网中一个比特的长度只有 10 Mbps 以太网中一个比特长度的 1/100，因此这种网络的最大长度就是 25 米。当所有设备都集中在一间屋子里时，

这个长度可能是适合的，但就算只连接一个办公室的所有计算机，它都不够长。

- **载波扩充** 为了使网络可以更长，我们就要增加最小帧长的值。载波扩充定义帧的最小长度为 512 字节（4096 比特）。也就是说最小帧长是原来的 8 倍。这种方法迫使各个站为所有小于 4096 比特的帧附加扩充位（填充）。使用这种方法后，网络的最大长度可以增加到 8 倍，达到 200 米。这就使得从集线器到各站之间的距离可以达到 100 米。
- **帧突发** 如果我们有一连串的短帧要发送，那么载波扩充就显得非常低效，因为每个帧都要运载冗余的数据。为了提高效率，人们提出了帧突发。这种模式不再为每个帧附加扩充位，而是一次发送多个帧。不过要想使多个帧看起来就像一个帧一样，那么在帧和帧之间需要加入填充（与载波扩充模式使用的一样），使得信道不会空闲。换言之，这种方法就是欺骗其他站，让它们相信发送过来的是一个非常大的帧。

实现

表 3.3 为吉比特以太网实现方法的小结。

表 3.3 吉比特以太网实现方法小结

特性	1000Base-SX	1000Base-LX	1000Base-CX	1000Base-T4
媒体	光缆 短波长	光缆 长波长	STP	5 类 UTP
线数	2	2	2	4
最大长度	550 m	5000 m	25 m	100 m

图 3.12 所示为吉比特以太网每种实现方法的简单示意图。

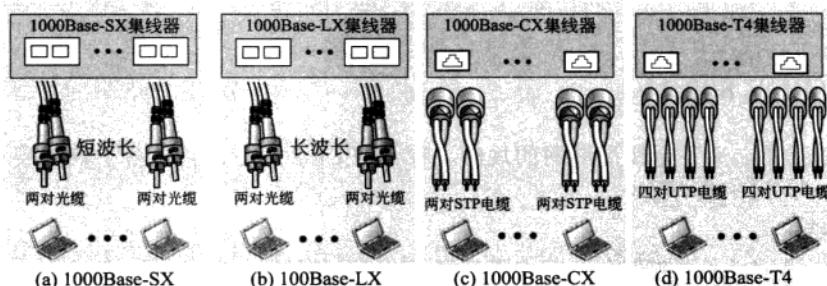


图 3.12 吉比特以太网的实现

3.1.8 10G 以太网

IEEE 建立了 10G 以太网并称之为标准 802.3ae。设计 10G 以太网的宗旨总结如下：

1. 数据率升级到 10 Gbps。
2. 使之与标准以太网和快速以太网和吉比特以太网兼容。

3. 使用相同的 48 比特地址。
4. 使用相同的帧格式。
5. 保持相同的最小帧长和最大帧长不变。
6. 允许现存的局域网能够连接到城域网（MAN）或广域网上。
7. 使以太网能够与像帧中继和 ATM 这样的技术兼容。

实现

10G 以太网只能以全双工模式操作，也就是说不再需要竞争，而且在 10G 以太网中不使用 CSMA/CD。最常见的实现方法有三种：10GBase-S、10GBase-L、10GBase-E。表 3.4 为 10G 以太网实现方法的小结。

表 3.4 10G 以太网实现方法小结

特性	10GBase-S	10GBase-L	10GBase-E
媒体	多模光缆	单模光缆	单模光缆
线数	2	2	2
最大长度	300 m	10 000 m	40 000 m

3.2 无线局域网

无线通信是发展速度最快的技术之一。不使用网线而将设备连接起来这样的需求越来越普遍。在大学校园、办公大楼里及其他许多公共场所都可以找到无线局域网（Wireless LAN）。本节我们重点介绍两种用于局域网的无线技术：IEEE 802.11 无线局域网和蓝牙，IEEE 802.11 有时也被称为无线以太网，而蓝牙则是一种小型无线局域网的技术。

3.2.1 IEEE 802.11

IEEE 已经定义了无线局域网的规约，称为 IEEE 802.11，它包含了物理层和数据链路层。

体系结构

这个标准定义了两类服务：基本服务集（BSS）和扩展的服务集（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件。基本服务集由固定的或移动的无线站以及可选的中央基站构成，中央基站称为接入点（access point, AP）。图 3.13 给出了这个标准中的两种基本服务集。没有接入点的 BSS 是个孤立的网络，不能向其他的 BSS 发送数据。这种体系结构称为自组织体系结构（ad hoc architecture）。在这种体系结构中，几个站就可以构成一个网络，而不需要有接入点。这些站可以互相定位并同意成为一个 BSS 的一部分。具有接入点的 BSS 有时被称为基础结构（infrastructure）网络。



图 3.13 基本服务集 (BSS)

扩展的服务集 一个扩展服务集 (extended service set, ESS) 由两个或更多具有 AP 的 BSS 构成。在这种情况下, 这些 BSS 都连接到一个分配系统 (distribution system), 它通常是一个有线局域网。分配系统把这些 BSS 中的 AP 都连接起来。IEEE 802.11 并没有对分配系统提出什么限制, 它可以是任意的 IEEE 局域网, 譬如以太网。请注意, 扩展服务集使用了两种类型的站: 移动的和固定的。移动站是 BSS 中的一个普通的站, 而固定站就是 AP 站, 它们也是有线局域网的一部分。图 3.14 描绘了一个 ESS。

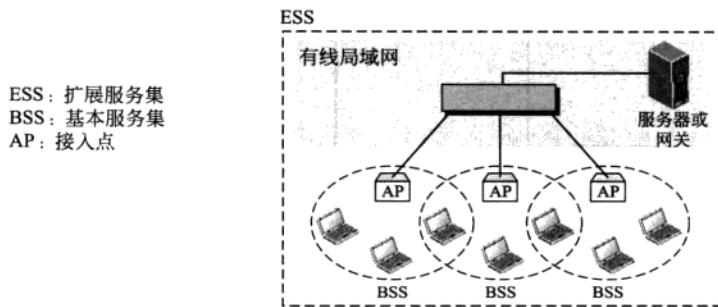


图 3.14 扩展服务集 (ESS)

当 BSS 互相连接起来后, 彼此之间能够直接联系得上的站就可以不用经过 AP 互相通信。但是在两个不同类型的 BSS 中的站之间的通信, 通常要经过两个 AP。

站的类型

IEEE 802.11 根据各站在无线局域网中的移动性定义了三种类型的站: 无切换 (no-transition)、BSS 切换 (BSS-transition)、ESS 切换 (ESS-transition)。具有无切换移动能力的站可能是固定的 (不移动) 或者是只能在一个 BSS 内移动。具有 BSS 切换移动能力的站允许从一个 BSS 移动到另一个 BSS, 但移动范围限制在一个 ESS 内。具有 ESS 切换移动能力的站能够从一个 ESS 移动到另一个 ESS 内。

MAC 子层

在这个协议中有两种不同的 MAC 子层, 不过使用得最多的还是基于 CSMA/CA (碰撞避免的载波监听多点接入) 的 MAC 子层。图 3.15 所示为其流程图。

有三个原因使得无线局域网不能直接应用 CSMA/CD。

1. 一个站如果要进行碰撞检测就必须能够同时发送数据和接收碰撞信号。这就意味着昂贵的费用以及对带宽需求增加。

2. 由于隐藏站的问题可能使碰撞不可检测。稍后我们将在本章讨论这个问题。
3. 站和站之间的距离可能会很远。信号衰减会使得在这一端的站无法听到另一端所发生的碰撞。

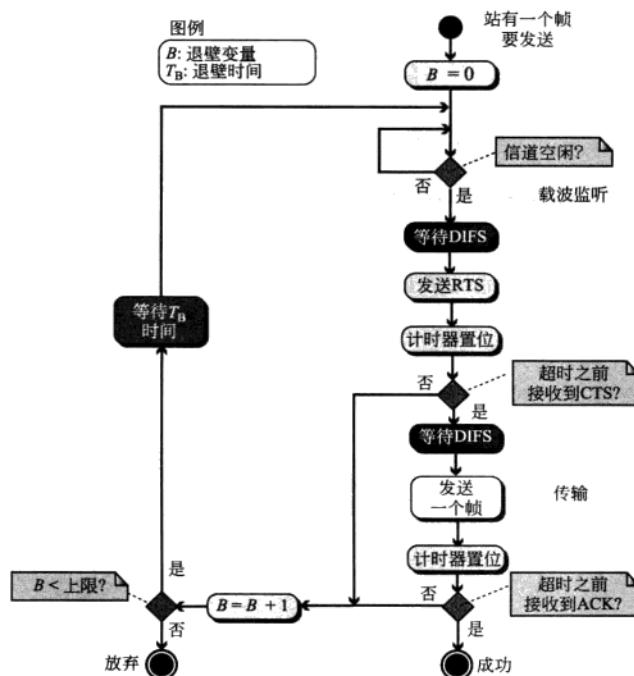


图 3.15 CSMA/CA 流程图

帧交换时序

图 3.16 所示为数据帧和控制帧的交换时序图。

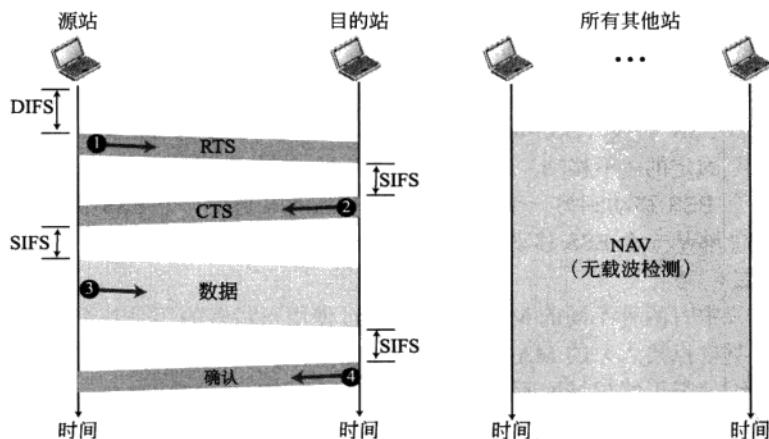


图 3.16 CSMA/CA 和 NAV

1. 源站在发送帧之前首先要检查载波频率上的能量值以检测媒体是否空闲。
 - a. 源站使用带退避的坚持（*persistence*）策略等待信道空闲。
 - b. 源站在发现信道空闲之后先等待一段称为分布帧间距（distributed interframe space, DIFS）的时间，然后再发送一个称为请求发送（RTS）的控制帧。
2. 目的站在接收到这个 RTS 并等待了一段称为短帧间距（Short interframe space, SIFS）的时间之后，向源站发送一个称为允许发送（CTS）的控制帧，这个控制帧表示目的站准备好接收数据。
3. 源站在等待了一段与 SIFS 等长的时间之后开始发送数据。
4. 目的站在等待了与 SIFS 等长的时间之后发送确认帧，以表示该帧已接收。在这个协议中确认帧是很有必要的，因为源站没有任何其他手段可用于检查自己的数据是否成功到达了目的站。从另一方面说，在 CSMA/CD 中没有检测到碰撞这件事本身就是在告诉源站数据已到达目的站。

网络分配向量

如果有一个站已经获得了接入权，那么其他站又如何推迟发送自己的数据呢？换言之，在这个协议中是如何实现碰撞避免的？关键在于一种被称为 NAV 的特性。

当某个站在发送 RTS 帧时，会在该帧中包含它需要占用信道的时间长度。此次传输波及到的所有站都会创建一个定时器，称为网络分配向量（Network Allocation Vector, NAV），它表示网站中的其他站必须等待多长时间才可以检查信道是否空闲。每当有一个站接入系统并发送 RTS 帧后，其他站就必须启动它们的 NAV。换言之，所有站在侦听物理媒体以检查该媒体是否空闲之前，首先需要检查自己的 NAV 是否到期。在图 3.16 中也描绘了 NAV 思想。

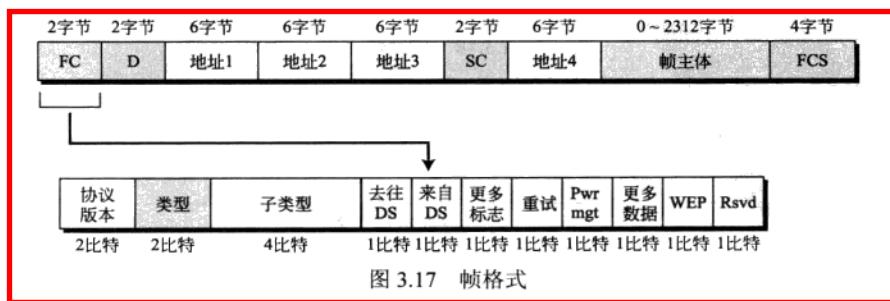
如果在传送 RTS 或 CTS 控制帧期间，也就是通常称为握手期（handshaking period），发生了碰撞，会发生什么事情呢？可能会有两个或更多个站在同时尝试发送 RTS 帧。这些控制帧有可能会碰撞。但是，因为没有任何碰撞检测机制，所以如果发送方没有收到来自接收方的 CTS 帧，它就认为发生了碰撞，此时发送方应用退避策略并再次尝试。

分片

无线环境非常嘈杂，被损坏的帧必须重传，因此，协议推荐使用分片方法，也就是将一个大的帧分割成几个较小的帧。重新发送一个小的帧要比重新发送一个大的帧效率更高。

帧格式

MAC 层的帧包括九个字段，如图 3.17 所示。



□ **帧控制 (FC)** FC 字段为 2 字节且定义了帧的类型以及一些控制信息。表 3.5 描述了这些子字段。稍后我们将在本章讨论帧的每一种类型。

表 3.5 FC 字段中的子字段

字段	解释
版本	当前版本为 0
类型	信息类型：管理 (00)、控制 (01)、数据 (10)
子类型	每种类型的子类型（参见表 3.6）
去往 DS	定义见后
来自 DS	定义见后
更多标志	置 1 时表示还有更多的分段
重试	置 1 时表示是重传的帧
Pwr mgt	置 1 时表示该站处于电源管理模式中
更多数据	置 1 时表示该站还有更多的数据需要发送
WEP	有线等效保密协议（实施加密）
Rsvd	保留的

- **D** 除了一种类型之外，在其他所有类型的帧中这个字段定义的都是传输持续时间，它用于设置 NAV 的值。只有在一种控制帧中，这个字段定义的是该帧的标识号。
- **地址** 共有 4 个地址字段，都是 6 字节长。每个地址字段的意义都取决于去往 DS 和来自 DS 子字段的值，见后面的解释。
- **序号控制 (SC)** 这个字段定义了在流量控制中使用的帧的序号。
- **帧主体** 这个字段长度可以从 0 到 2312 字节，所包含的信息内容取决于定义在 FC 字段中的类型和子类型。
- **FCS** FCS 字段有 4 字节长，含有 CRC-32 差错检测序列。

帧类型

由 IEEE 802.11 定义的无线局域网具有三大类型的帧：管理帧、控制帧和数据帧。

管理帧 管理帧用于站和接入点之间的通信初始化时。

控制帧 控制帧用于信道的接入和帧的确认。图 3.18 所示为其格式。



图 3.18 控制帧

控制帧的类型字段值为 01，而我们之前曾提到的帧的子类型字段值如表 3.6 所示。

表 3.6 控制帧中子类型的值

子类型	含义
1011	请求发送 (RTS)
1100	允许发送 (CTS)
1101	确认 (ACK)

数据帧 数据帧用于携带数据和控制信息。

3.2.2 编址机制

IEEE 802.11 的编址机制定义了四种情况，由 FC 字段中的两个标志去往 DS 和来自 DS 的值决定。每个标志都可能是 0 或者是 1，其结果得到四种不同的情况。对于 MAC 帧中的四个地址（从地址 1 到地址 4）的解释则取决于这些标志的值，如表 3.7 所示。

表 3.7 地址

去往 DS	来自 DS	地址 1	地址 2	地址 3	地址 4
0	0	目的站	源站	BSS ID	不用
0	1	目的站	发送 AP	源站	不用
1	0	接收 AP	源站	目的站	不用
1	1	接收 AP	发送 AP	目的站	源站

我们可以注意到，地址 1 总是下一个设备的地址。地址 2 总是上一个设备的地址。地址 3 是最后的目的站地址，如果它没有被地址 1 定义的话。地址 4 则是最初的源站地址，如果它和地址 2 不一样的话。

隐藏站和暴露站问题

我们在前面的小节中已经提到了隐藏站和暴露站的问题。现在是时候来讨论这两个问题以及它们所带来的影响了。

隐藏站问题 图 3.19 所示为隐藏站问题的一个例子。站 B 的传输范围是左边的椭圆部分（实际空间里是球形的），站 B 传送的任何信号在这个区域里的所有站都能听到。站 C 的传输范围是右边的椭圆部分（实际空间里是球形的），由站 C 传送的任何信号在这个范围内的站都能听到。站 C 在站 B 的传输范围之外，反过来说，站 B 也在站 C 的传输范围之外。但是站 A 既在站 B 的传输范围内，又在站 C 的传输范围内，因此不管是站 B 还是站 C 传送的信号它都能听到。

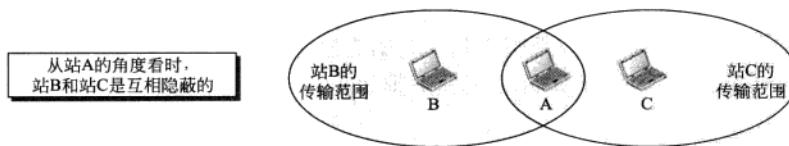


图 3.19 隐藏站问题

假设站 B 正在向站 A 发送数据，而在此发送期间，站 C 也有数据要发送到站 A。但是站 C 不在站 B 的传输范围内，站 B 的传输无法到达站 C，因此站 C 会认为媒体是空闲的。站 C 于是也向站 A 发送自己的数据，这就会导致在站 A 发生碰撞，因为站 A 接收的数据可能来自站 B，也可能来自站 C。在这种情况下，我们说从站 A 的角度看，站 B 和站 C 是互相隐藏的。隐藏站的问题会降低网络容量，因为有可能会发生碰撞。

隐藏站问题的解决方法是使用握手帧（RTS 和 CTS），如我们前面所讨论的。在图 3.20 中，来自站 B 的 RTS 报文抵达了站 A，但是没有抵达站 C。不过，由于站 B 和站 C 都处于站 A 的传输范围内，所以 CTS 报文会到达站 C，而在这个 CTS 报文中含有从站 B 到站 A 的数据传输所需要的时间长度。于是站 C 就知道某个隐藏站正在使用信道，从而抑制传输直至超过该时间长度。

CSMA/CA 握手时的 CTS 帧可以用来防止因隐藏站而带来的碰撞。

暴露站问题 现在要考虑一个与前面正好相反的情况，即暴露站问题。这个问题在于一个站可能会在信道实际上可用时却抑制了发送。在图 3.21 中，站 A 正在向站 B 传送数据，而站 C 有一些数据要发送给站 D，这些数据本来可以在不打断站 A 到站 B 的传输的情况下正常发送，但是因为站 C 暴露在站 A 的传输中，也就是说它能听到站 A 发送的数据，因而抑制了发送。换言之，站 C 因太过保守而浪费了信道容量。

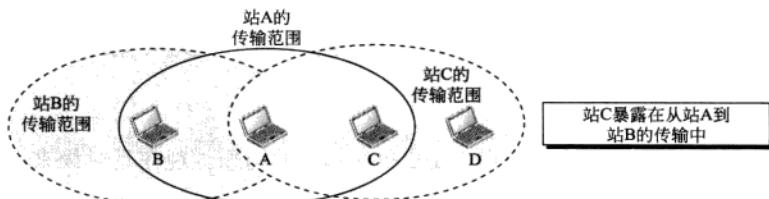


图 3.21 暴露站问题

也许我们希望握手过程中的报文 RTS 和 CTS 会对此有所帮助，遗憾的是它们也无能为力。图 3.22 描绘了这种情况。

站 C 听到了来自站 A 的 RTS，但是听不到来自站 B 的 CTS。站 C 在听到来自站 A 的 RTS 后会等待一段时间，在此期间来自站 B 的 CTS 到达了站 A，然后站 C 向站 D 发送一个 RTS 表示自己需要与站 D 通信。站 D 和站 A 都有可能听到这个 RTS，但是站 A 正在发送状态，而非接收状态。站 D 会用一个 CTS 来响应。问题就在这里。如果站 A 已经开始发送数据了，站 C 就会因为碰撞而听不到来自站 D 的 CTS，于是它就无法将数据发送给 D。站 C 始终处于暴露状态，直至站 A 发送完数据为止。

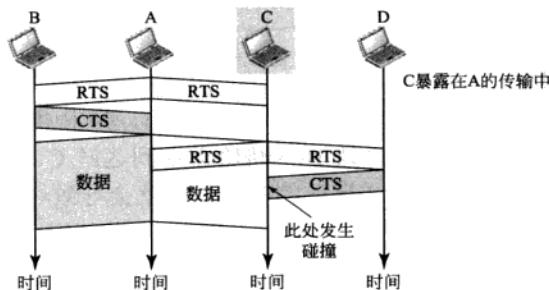


图 3.22 在暴露站问题上使用握手过程

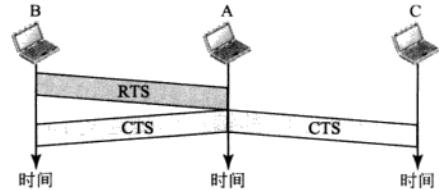


图 3.20 使用握手过程来防止隐藏站问题

3.2.3 蓝牙

蓝牙（bluetooth）是设计用于连接具有不同功能的设备（如电话、笔记本、台式计算机、照相机、打印机、咖啡壶等等）的无线局域网技术。蓝牙局域网是一种自组织网络，也就是说这个网络是自发组成的，这些设备（有时也称为小电器）互相找到对方并形成一个称为微微网的网络。蓝牙局域网甚至能连接到因特网上，只要其中的某个小电器有此能力。蓝牙局域网天生就不可能很大。如果想把很多小家电互相连接起来，那肯定会乱成一团麻。

蓝牙技术有一些典型的应用。像无线鼠标或键盘可以通过这种技术与计算机之间进行通信。在一个小型的医疗保健中心，监控设备也能够用它与传感器互相通信。家用安全设备可以使用这种技术将不同的传感器与主安全控制器相互连接起来。而在会议上，参加会议的人员可以通过这种技术使计算机互相同步。

蓝牙最初是由爱立信公司开发的一个项目。它的命名源自 Harald Blaatand，他是一位丹麦国王（940~981），曾经统一过丹麦和挪威。Blaatand 翻译成英语就是 Bluetooth（蓝牙）。

现在，蓝牙技术是指由 IEEE 802.15 标准定义的协议实现。这个标准定义了一个无线个人局域网（PAN），它可以在一个房间或厅堂大小的空间内工作。

体系结构

蓝牙定义了两种类型的网络：微微网和分散网。

微微网 有一种蓝牙网络称为微微网（piconet）。一个微微网最多可以有 8 个站，其中一个站称为主站（primary），其他站称为从站（secondaries）。所有从站的时钟及跳频都要与主站进行同步。请注意一个微微网只能有一个主站。主站与从站之间的通信可以是一对一的，也可以是一对多的。图 3.23 所示为一个微微网。

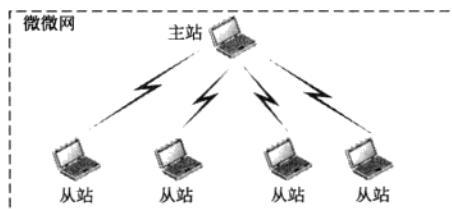


图 3.23 微微网

虽然一个微微网有最多 7 个从站的限制，但还可以有另外 8 个处于停用状态的从站。处于停用状态的从站已经与主站同步过了，但现在还不能参与通信，除非它解除停用状态。因为在一个微微网中只能有 8 个活动的站，所以要激活一个停用状态的站就意味着有一个活动状态的站必须进入停用状态。

分散网 多个微微网可以组合起来形成一个分散网（scatternet）。一个微微网的从站可能是另一个微微网的主站，它可以接收由第一个微微网的主站发送的报文（作为一个从站），然后以主站的身份将这些报文传递给第二个微微网上的从站。也就是说一个站可以成为两个微微网的成员。图 3.24 描绘了一个分散网。

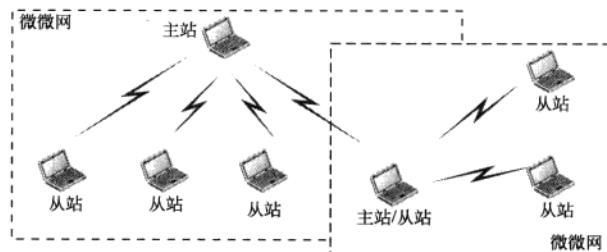


图 3.24 分散网

蓝牙设备

一个蓝牙设备可以是一个内置的短距离无线电波发射器。目前的数据率是 1 Mbps，带宽为 2.4 GHz。这也就意味着在 IEEE 802.11 无线局域网和蓝牙局域网之间有可能会互相干扰。

帧格式

基带层的帧可能是以下三种类型之一：1 个时隙的，3 个时隙的或 5 个时隙的。我们以前曾说过一个时隙是 625 μ s。不过 1 个时隙的帧在交换过程中需要有 259 μ s 用于跳频及控制机制。这也就意味着一个 1 个时隙的帧只能有 366 μ s (625 μ s ~ 259 μ s)，加之 1 MHz 带宽及 1 bit/Hz 的条件，一个 1 个时隙的帧的大小就是 366 位。

一个 3 个时隙的帧占用了 3 个时隙。不过由于有 259 μ s 用于跳频，因此帧的长度是 $3 \times 625 - 259 = 1616 \mu$ s 或 1616 位。使用 3 个时隙帧的设备在三个时隙上都保持着相同的跳频（在相同的载波频率上）。哪怕只使用了一个跳频数也会浪费掉三个跳频数。这就是说每个帧的跳频数就等于该帧第一个时隙的跳频数。一个 5 个时隙的帧同样有 259 μ s 用于跳频，意味着此时帧的长度是 $5 \times 625 - 259 = 2866$ 位。图 3.25 所示为这三种类型的帧格式。

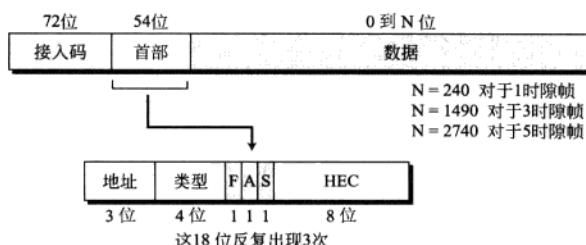


图 3.25 几种类型的帧格式

其中每个字段的描述如下：

- 接入码** 这个 72 位的字段一般情况下包含的是同步位和主站的标识符，以此来区别不同微微网中的帧。
- 首部** 在这个 54 位的字段中反复出现着一个 18 位的固定样式。这个样式由以下子字段组成：
 - a. **地址** 这个 3 位的地址子字段可以定义最多七个从站（从 1 到 7）。如果这个地址为 0，就说明是主站向所有从站发出的广播通信。
 - b. **类型** 这个 4 位的类型子字段定义了上层数据的类型。我们将在后面讨论这些类型。

- c. **F** 这个1位的子字段用于流控制。当置为1时表示这个设备无法接收更多的帧（缓存已满）。
- d. **A** 这个1位的子字段用于确认。蓝牙使用的是停止等待ARQ，且只需1位就足够用于确认了。
- e. **S** 这个1位的子字段包含的是一个序列号。蓝牙使用的是停止等待ARQ，且只需1位的序列号就足够了。
- f. **HEC** 这个8位的首部纠错子字段是一个检验和，用于检测首部中18位为一小节的差错。

首部中有三个完全相同的18位长的段。接收方会逐位比较这三段，如果三次重复完全一致，该位就被接受。如果有不同，则少数服从多数。这是一种前向纠错方法。之所以需要这种双差错控制机制是因为此种通信是通过空气来传输的，本身就非常嘈杂。请注意在此子层是没有重传机制的。

- 数据** 这个子字段的长度从0位到2740位都可以。它包含的是来自上层的数据或控制信息。
 - a. 发送站在检测到媒体空闲后会发送一个称为请求发送(CTS)的特殊帧。在这个报文中，发送方定义了它需要使用媒体的总时间。
 - b. 接收站通过发送一个称为允许发送(CTS)的帧来确认此请求(向所有站广播)。
 - c. 发送站发送数据帧。
 - d. 接收站对接收到的数据进行确认。

3.3 点到点广域网

在因特网中我们遇到的第二种类型的网络是点到点广域网。点到点广域网从公共网(如电话网)中获取一条线路来连接两个远程设备。我们要讨论的是传统的调制解调器技术、DSL线、电缆调制解调器、T线和SONET。

3.3.1 56K 调制解调器

目前我们仍然在使用传统调制解调器将数据上传到因特网，并从因特网下载数据，如图3.26所示。

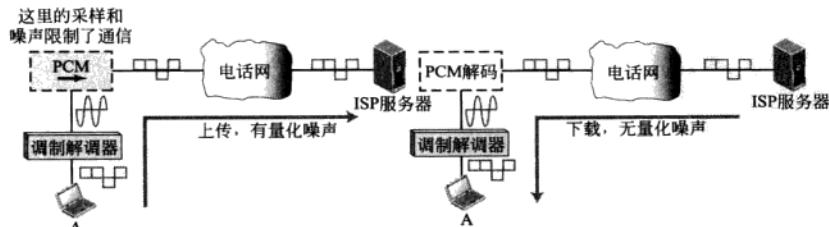


图3.26 56K 调制解调器

上传（uploading）时，模拟信号必须在交换站采样，这表示上传的数据率被限制为 33.6 kbps。但是下载（downloading）时却没有采样问题，信号不受量化噪声的影响，因而也不会受到香农定理的容量限制。上传方向的最高数据率是 33.6 kbps，而下载方向的数据率则可达到 56 kbps。

读者会想为什么是 56 kbps。电话公司对话音信号的采样速率是每秒 8000 个采样，每个采样用 8 位编码。每个采样中需要 1 位用于控制，这就意味着每个采样为 7 位。因此数据率就是 8000×7 ，或 56000 bps，或 56 kbps。

V.90 和 **V.92** 标准调制解调器以 56 kbps 的数据率将主机连接到因特网。

3.3.2 DSL 技术

当传统的调制解调器达到其最高数据率后，电话公司又开发出了另一种技术，即 DSL，它可以提供到因特网的高速接入。数字用户线（Digital Subscriber Line，DSL）技术是使用现有的本地用户线（即电话线）来支持高速数字通信的一种最有前途的技术。DSL 技术是一组技术的统称，其中每种技术以不同的首字母来区分（ADSL、VDSL、HDSL 和 SDSL）。这组技术常记为 xDSL，这里的 x 可以是 A、V、H 或 S。

ADSL

在这组技术中排首位的是非对称数字用户线（Asymmetric DSL，ADSL）。像 56K 调制解调器一样，ADSL 在下行方向（从因特网到用户）可提供比上行方向（从用户到因特网）更高的速率（比特率）。这就是它称为非对称的原因。与 56K 调制解调器的非对称不同的是 ADSL 的设计者特意不平均地分割了本地环路中居民用户的可用带宽。这种服务不适用于企业用户，因为他们在两个方向都需要大的带宽。

ADSL 是一种为居民用户设计的非对称的通信技术，它不适用于企业。

图 3.27 给出了带宽是如何划分的：

- 话音** 信道 0 为话音通信保留。
- 空闲** 信道 1~5 未使用，以便在话音和数据通信之间留有间隙。
- 上行数据和控制** 信道 6~30（25 个信道）用于上行数据的传送和控制。一个信道用于控制，24 个信道用于数据传送。如果有 24 个信道，且每个信道使用 4 kHz（在 4312 kHz 的可用带宽中），在每赫兹 15 位的条件下，我们在上行方向有 $24 \times 4000 \times 15$ 或者说 1.44 Mbps 带宽。
- 下行数据和控制** 信道 31~255（225 个信道）用于下行数据的传送和控制。一个信道用于控制，224 个信道用于数据传送。如果是全部的 224 个信道，我们就可以达到 $224 \times 4000 \times 15$ 或 13.4 Mbps 的带宽。

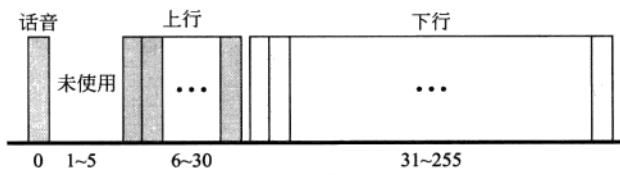


图 3.27 带宽划分

由于信噪比较高，实际的比特率要大大低于以上的数值。实际的比特率如下：

上行：64 kbps~1 Mbps

下行：500 kbps~8 Mbps

图 3.28 描绘的是安装在用户端的一个 ADSL 调制解调器。本地环路连接到一个过滤器，其作用是把话音通信和数据通信分开。ADSL 调制解调器对数据进行调制并建立下行和上行信道。

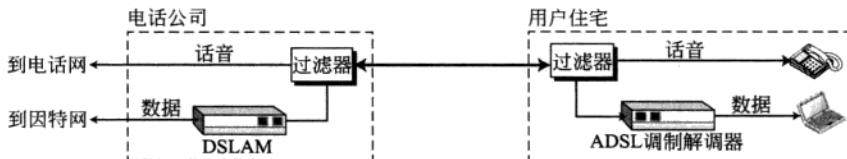


图 3.28 ADSL 和 DSLAM

而在电话公司端的情况就有所不同了。这里不使用 ADSL 调制解调器，而是安装了称为数字用户线接入复用器（Digital Subscriber Line Access Multiplexer, DSLAM）的设备，其功能与 ADSL 调制解调器是相似的。除此之外，它还要把即将发送到因特网的数据打包。图 3.28 给出了这种配置。

其他的 DSL 技术

ADSL 提供了非对称的通信。下行数据率大大高于上行数据率。虽然这一特点能够使绝大多数的居民用户满意，但它并不适合于一些企业单位，因为他们在两个方向都有大量的数据要发送和接收。对称数字用户线（Symmetric DSL, SDSL）就是为这类企业而设计的，它把可用带宽平均分配到下行和上行两个方向。

高数据率数字用户线（High bit rate DSL, HDSL）是设计来取代 T-1 线（1.544 Mbps）的技术。T-1 线技术（后面还要讨论）使用了传号交替反转（AMI）编码，这使得它在高频端对衰减很敏感，因此 T-1 线的长度限制在 1 公里内。对于更远的距离就必须安装转发器，也就意味着增加了成本。

甚高数据率数字用户线（Very high bit rate DSL, VDSL）是类似 ADSL 的另一种技术，它使用同轴电缆、光缆以及双绞线电缆，用于短距离传输（300~1800 m）。其调制技术为离散多音技术（DMT），下行数据率是 50~55 Mbps，上行是 1.5~2.5 Mbps。

3.3.3 电缆调制解调器

目前，有线电视公司正在和电话公司争夺想要高速接入因特网的居民用户。DSL 技术在本地环路上为居民用户提供高数据率的连接。但是，DSL 使用的是现存的无屏蔽双绞线电缆，它很容易受到干扰，这就限制了数据率的上限。另一个解决问题的方法是利用有线电视网。

传统的有线电视网

最初的有线电视（Cable TV）是为了向接收条件较差或无法接收信号的地区分发广播视频信号。它曾经称为共用天线电视（Community Antenna TV, CATV），因为它使用安装在山丘或建筑物顶上的天线接收来自电视台的信号，然后通过同轴电缆把信号分发到居民区。

称为头端 (head end) 的有线电视机房接收来自广播电视台的视频信号, 然后把信号送入同轴电缆。传统的有线电视网端对端地使用同轴电缆。由于信号的衰减, 必须使用大量的放大器, 同时传统网络中的通信是单向的 (一个方向)。视频信号下行传送, 从头端到达用户住宅。

HFC 网络

第二代的有线电视网称为混合光纤同轴 (Hybrid Fiber-Coaxial, HFC) 网络。这种网络同时使用光纤和同轴电缆。从有线电视机房到一个称作光纤结点 (fiber node) 的盒子所使用的传输媒体是光纤, 从光纤结点通过居民区一直到用户住宅所使用的传输媒体依旧是同轴电缆。之所以要把基础结构从传统的转换为混合的, 其中有部分原因是为了把有线电视网变成双向的 (两个方向)。

带宽

即使在 HFC 系统中, 网络的最后一部分 (即从光纤结点到用户住宅) 仍然使用了同轴电缆。这种同轴电缆的带宽大约是 5~750 MHz。有线电视网公司把这个带宽划分为 3 个频带: 视频、下行数据和上行数据, 如图 3.29 所示。

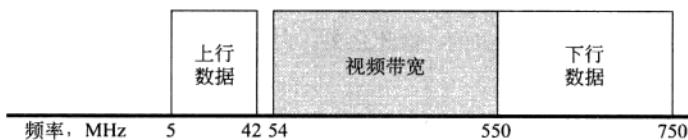


图 3.29 有线带宽

- **视频频带** 仅有下行传输的视频频带 (video band) 范围是 54~550 MHz。由于每个电视频道占据 6 MHz 的带宽, 因此可容纳超过 80 个频道。
- **下行数据频带** 下行数据 (从因特网到用户住宅) 占据较高的 550~750 MHz 的频带。这个频带也被划分成一些 6 MHz 的信道。下行数据能够以 30 Mbps 的速率接收。标准规定的只有 27 Mbps。但是, 电缆调制解调器是通过 10Base-T 电缆连接到计算机的, 这就把数据率限制在了 10 Mbps。
- **上行数据频带** 上行数据 (从用户住宅到因特网) 占据较低的 5~42 MHz 的频带。这个频带同样被划分为一些 6 MHz 的信道。上行数据频带使用了较低的频率, 更加容易受到噪声和干扰的影响。从理论上讲, 上行数据的发送速率可达 12 Mbps ($2 \text{ bit/Hz} \times 6 \text{ MHz}$)。但是, 通常这个数据率小于 12 Mbps。

共享

上行和下行频带都是由用户们共享的。上行数据带宽只有 37 MHz。这就表示, 在上行方向只有 6 个 6 MHz 的信道可用。一个用户就需要使用一个信道在上行方向发送数据。问题是: 这 6 个 6 MHz 的信道是怎样在拥有 1000 或者 2000, 甚至是 10 000 个用户的区域中共享的呢? 解决问题的方法是分时共享。频带被划分为一些信道, 信道必须由在同一个居民区的用户共享。有线电视网提供者静态地或动态地将一个信道分配给一组用户。如果某个用户想发送数据, 她或他就要和其他也想接入网络的用户竞争, 用户必须等到信道可以用时才能发送数据。这种情况类似于以太局域网中讨论过的 CSMA。

下行方向的情况也是类似的。下行频带有 33 个 6 MHz 的信道。有线电视提供者可能有多于 33 个的用户, 因此, 每个信道必须在一组用户中共享。但是, 下行方向的情况与上

行方向不同，在这里我们采取的是多播方式。如果有数据要传送给这个组中的任一位用户，那么就将数据发送到这个信道上。该数据会被发送给组中的每一位用户。由于每个用户在有线电视提供者那儿都有一个注册过的地址，因此该组的电缆调制解调器就会把随数据一起携带过来的地址与有线电视提供者为其指派的地址相比较。若地址匹配，这个数据就保留，否则，就丢弃这些数据。

设备

要使用有线电视网络来传输数据，我们就需要两个关键设备：一个 CM 和一个 CMTS。电缆调制解调器（Cable Modem, CM）安装在用户住宅中，它与 ADSL 调制解调器相似。图 3.30 描绘了它的位置。电缆调制解调器传输系统（Cable Modem Transmission System, CMTS）由有线电视网公司安装在分配集线器里。它接收来自因特网的数据，并把数据传递到组合器，再由组合器传递到用户。CMTS 还要接收来自用户的数据，并把数据传递到因特网。图 3.30 也给出了 CMTS 的位置。

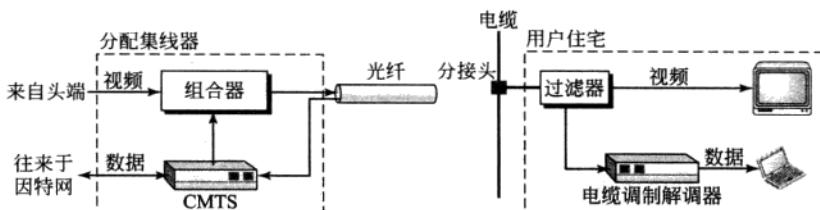


图 3.30 电缆调制解调器的配置

3.3.4 T 线

T 线（T lines）就是标准的数字电话线路，最初用来复用话音信道（在数字化之后）。但现在 T 线已用来把数据从居民小区或从某个组织传送到因特网。T 线还可 在交换广域网的结点之间提供物理链路。T 线市场上的产品有两种数据率：T-1 和 T-3（见表 3.8）。

表 3.8 T 线的速率

线	数据率 (Mbps)
T-1	1.544
T-3	44.736

T-1 线

T-1 线（T-1 line）的数据率是 1.544 Mbps。24 个话路被采样，每个样本数字化后变为 8 位，再加上一位用来同步，这样就使一个帧的长度为 193 位。因为每秒发送 8000 帧，所以得出的数据率为 1.544 Mbps。当我们使用 T-1 线连接因特网时，我们可以使用该线的全部或部分容量来发送数据。

T-3 线

T-3 线（T-3 line）的数据率是 44.736 Mbps。它等效于 28 个 T-1 线。很多用户可能并不需要 T 线的全部容量。为了适应这部分用户，电话公司开发了部分 T 线业务，允许若干个用户通过对传输的复用来共享一个 T 线。

3.3.5 SONET

光缆的高带宽不仅适用于目前数据率最高的技术（如视频会议），还可以同时承载大量较低速率的技术。ANSI 建立了一组标准，称为同步光纤网（Synchronous Optical Network, SONET），用来有效使用光缆。它定义了一种高速数据承载方式。

SONET 首先定义了一组电信号，称为同步运输信号（Synchronous Transport Signals, STS），然后将这些信号转换为相应的光信号，称为光载波（Optical Carriers, OC）。光信号以每秒 8000 帧的速率传输。

表 3.9 给出了 STS 和 OC 的数据率。请注意这里最低一级的数据率是 51.840 Mbps，大于 T-3 的速率（44.736 Mbps）。

表 3.9 SONET 的数据率

STS	OC	数据率 (Mbps)	STS	OC	数据率 (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

3.3.6 PPP

虽然电话公司或有线电视网公司提供了物理链路，但是还需要专门的协议来控制和管理数据的传送。点到点协议（Point-to-Point Protocol, PPP）就是为此而设计的。

PPP 分层

PPP 只有物理层和数据链路层。PPP 没有为物理层定义特定的协议。相反，PPP 让实施者自行选择可用的协议。PPP 支持 ANSI 认可的所有协议。在数据链路层，PPP 定义了帧的格式，以及用来控制链路和传送用户数据的协议。PPP 的帧格式如图 3.31 所示。

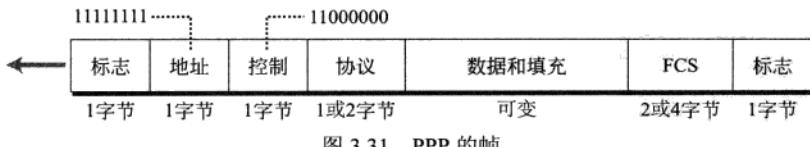


图 3.31 PPP 的帧

下面是各字段的描述：

1. **标志字段。** 标志字段用来标志 PPP 帧的边界。它的值是 01111110。
2. **地址字段。** 因为 PPP 用于点到点连接，所以它使用了绝大多数局域网中使用的广播地址 11111111，这样在协议中就可以避免数据链路层地址。
3. **控制字段。** 控制字段的值是 11000000，表示这个帧不使用序号（如同在绝大多数局域网中一样），每个帧都是独立的。

4. 协议字段。协议字段用来定义在数据字段中携带的数据类型：用户数据或其他信息。
5. 数据字段。这个字段携带的是用户数据或其他信息。
6. FCS。这个帧检验序列字段是简单的 2 字节或 4 字节的 CRC，用来进行差错检测。

链路控制协议（LCP）

链路控制协议（Link Control Protocol, LCP）负责建立、维护和终止链路。当一个帧的数据字段携带的是与这个协议有关的数据时，就表示 PPP 正在处理链路；它不携带数据。

网络控制协议（NCP）

定义网络控制协议（Network Control Protocol, NCP）是为了使 PPP 协议具有灵活性。PPP 可以携带来自不同网络协议（包括 IP）的数据。当链路建立后，PPP 就能在它的数据字段中携带 IP 分组。

PPPoE

PPP 本来是为单个用户通过传统的调制解调器和电话线连接到因特网而设计的。但在今天，DSL、电缆调制解调器和无线技术允许以太局域网上的一组用户通过一条物理线路接入到因特网。换言之，连接在以太网上的多个主机能够共享一条物理线路接入到因特网。以太网上的 PPP（PPP over Ethernet, PPPoE）是一个新的协议，它使用一种发现技术来找出需要连接因特网的主机的以太网地址。在这个地址被发现后，就可以使用正常的 PPP 会话提供连接。

3.4 交换广域网

因特网的主干网可以是交换广域网。交换广域网是覆盖大面积（一个州或国家）的广域网，并能向用户提供多个接入点。在网络内部用网状的点到点网络来连接各交换机。这些交换机都有多端口连接器，可连接多个输入和输出。

交换广域网技术与局域网技术相比有许多不同之处。首先它不使用星形拓扑结构，而是使用交换机产生多条路径。其次局域网技术被认为是一种无连接技术，也就是说发送方向接收方发送的各个分组之间没有直接关系。但交换广域网则不然，它使用面向连接的技术。在发送方发送分组之前，发送方和接收方之间必须先建立一条连接。当连接建立后就被指派了一个标识符（有时称为标记），在传输期间都要使用这个标识符。当传输结束后，连接还必须正式地终止。这种连接标识符代替了局域网技术中的源地址和目的地址。

3.4.1 X.25

在 20 世纪 70 年代问世的 X.25 协议是在欧洲和美国曾经都很流行的第一种交换广域网，主要用作连接单个计算机或局域网的公用网络。X.25 提供端到端的服务。

虽然 X.25 曾经作为广域网用来把 IP 分组从世界的一个地方传送到另一个地方，但在 IP 和 X.25 之间总是有些冲突的。IP 是一个第三层（网络层）的协议。一个 IP 分组应当由第二层（数据链路层）的帧来传送。但在因特网出现之前就已设计的 X.25 是一个三层协议，它有自己的网络层。IP 分组必须封装在 X.25 的网络层的分组中，才能从网络的一端传送到

网络的另一端。这就好比一个人已经在小汽车中，但还要把小汽车装在大卡车中才能从一个地方行驶到另一个地方。

X.25 的另一个问题是在设计它的时候传输媒体还不怎么可靠（没有使用光缆）。因此，X.25 使用了过多的差错控制。这会使传输非常慢，因此无法适应现在对速度需求的增长。

3.4.2 帧中继

帧中继（Frame Relay）协议是一种提供了底层（物理层和数据链路层）服务的交换技术，设计它是用来代替 X.25 的。帧中继与 X.25 相比具有如下的一些优点：

- **高数据率** 虽然帧中继最初的设计是提供 1.544 Mbps 的数据率（相当于 T-1 线），但今天绝大多数的实现可以处理高达 44.736 Mbps 的数据率（相当于 T-3 线）。
- **突发数据** 广域网提供者所提供的某些业务是假定用户需要使用固定的数据率。例如，T-1 线是为始终如一地以 1.544 Mbps 的速率使用线路的用户而设计的。但这种类型的业务对于需要发送**突发数据**（bursty data，即非固定速率数据）的用户来说并不适用。例如，某用户可能希望先以 6 Mbps 发送 2 秒钟的数据，接下来的 7 秒是 0 Mbps（什么也不发送），最后 1 秒是 3.44 Mbps，这样总共花了 10 秒发送了 15.44 Mb 的数据。虽然其平均数据率仍为 1.544 Mbps，但 T-1 线则无法满足这种类型的需求，因为 T-1 线是为固定速率的数据而不是为突发数据设计的。突发数据要求的是**按需带宽**（bandwidth on demand）。用户在不同时候需要不同的带宽。帧中继能够接受突发数据。用户可以保证获得某个平均数据率，并在需要时也可超过这个平均数据率。
- **因传输媒体的改善而减少了开销** 近十几年来，传输媒体的质量得到了极大的改善。传输媒体更加可靠，差错也减少了。因此，已经没有必要让广域网花费时间一而再地反复检查可能出现的差错。X.25 提供了过多的差错检查和流量控制。帧中继不再提供差错检查，并且在数据链路层也不需要确认。相反，帧中继把所有的差错检查留给使用帧中继服务的网络层和运输层协议。

3.4.3 ATM

异步传递方式（Asynchronous Transfer Mode，ATM）是由 ATM 论坛设计，并被 ITU-T 采纳的信元中继协议。

设计目标

在 ATM 的设计者所面临的挑战中，以下六个是其中最主要的。第一也是最重要的是，需要有一种传输系统能够最优化地使用高数据率的传输媒体（特别是光纤）。第二，需要有一种系统能够与现有的各种系统（如不同的分组交换网）接口，并能在现有系统之间提供广域互连，但不会降低它们的性能，也不需要重新更换它们。第三，需要一种实现起来很便宜的设计，从而使得价格不会成为其被采用的障碍。如果 ATM 打算成为国际通信的主干，那么对有意愿使用 ATM 的每一个用户来说，它必须是廉价可用的。第四，新的系统必须能够与现有的电信体系（本地用户线、本地服务提供者、长途网等等）一起工作并提供支持。

第五，新的系统必须是面向连接的，以保证准确和可预期的交付。最后一点同样不可忽视，要把尽可能多的功能转移到硬件上（为了提高速度），同时尽可能地减少其软件功能（这同样是为了提高速度）也是其中的一个目标。

信元网络

ATM 是一种信元网络。一个信元（cell）就是一个很小且长度固定的数据单元，它是信元网络中数据交换的基本单位。在这种类型的网络中，所有的数据都被装载到相同的信元中，而这些信元的传递是完全可预计的和统一的。在信元网络中，一个信元可以与其他信元一起被复用和转发。由于信元的长度都一样，而且都很短，因此不会出现不同长度的分组在复用时遇到的问题。

信元网络使用信元作为基本的数据交换单位。信元定义为固定长度的、小的信息块。

异步 TDM

ATM 使用异步时分复用（asynchronous time-division multiplexing）技术把不同信道上的信元进行复用，这就是为什么称为异步传递方式的原因。它使用和信元长度相同的定长时隙。ATM 复用器把有信元的信道中的信元装入时隙。如果所有信道都没有信元，则这个时隙就处于空闲状态。

图 3.32 所示为来自三个输入的信元进行复用的情况。在时钟的第一个滴答，信道 2 没有信元（输入时隙是空的），因此复用器把第三个信道的信元填入时隙。当所有信道的信元都进入复用器后，输出时隙就变成空闲状态。



图 3.32 ATM 的复用

ATM 的体系结构

ATM 是一个交换网络。称作端点的用户接入设备与网络内部的交换机相连。交换机和交换机之间使用高速通信信道互相连接。图 3.33 给出了 ATM 网络的一个例子。

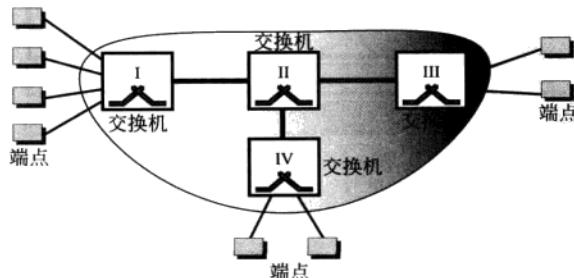


图 3.33 ATM 网络的体系结构

虚连接 两个端点之间的连接是通过传输路径 (TP)、虚通道 (VP) 以及虚电路 (VC) 完成的。传输路径 (Transmission Path, TP) 是端点和交换机之间或两个交换机之间的物理连接 (电线、电缆、卫星等等)。如果把两个交换机视为两座城市, 那么传输路径就是直接连接这两座城市的所有高速公路的集合。

一条传输路径可划分为多个虚通道。虚通道 (Virtual Path, VP) 提供了两个交换机之间的一条或一组连接。可以把虚通道视为连接两个城市的一条高速公路。如果说每条高速公路都是一条虚通道, 那么所有高速公路的集合就是传输路径。

信元网络是基于虚电路 (Virtual Circuits, VC) 的。属于一个报文的所有信元都会沿着一条相同的虚电路传输, 并保持它们原有的发送顺序, 直至到达终点为止。可以把虚电路视为高速公路 (虚通道) 上的车道, 如图 3.34 所示。

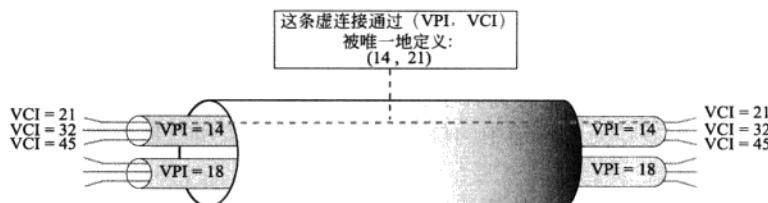


图 3.34 虚电路

图中描绘了传输路径 (物理连接)、虚通道 (是被裹成一束的许多虚电路的组合, 因为这些虚电路有部分路径是相同的) 和虚电路 (从逻辑上把两个点连接起来) 之间的关系。

在虚电路网络中, 要将数据从一个端点传送到另一个端点, 就必须对虚连接进行标识。为此, ATM 的设计者创造了一种分成两级的标识符: 虚通道标识符 (Virtual Path Identifier, VPI) 和虚电路标识符 (Virtual Circuit Identifier, VCI)。VPI 指明了具体的 VP, 而 VCI 指明了该 VP 中的某一条 VC。所有 (逻辑上) 被裹在一个 VP 中的虚连接的 VPI 都是相同的。

一条虚连接是用一对数值来指明的: VPI 和 VCI。

ATM 分层

ATM 标准定义了三层。它们分别是 (从上到下): 应用适配层, ATM 层和物理层, 如图 3.35 所示。

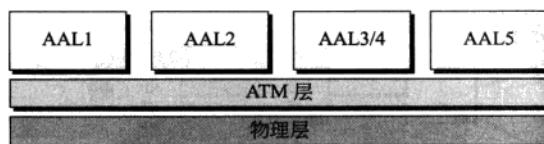


图 3.35 ATM 的层

物理层和 ATM 层既用于网络内部的交换机, 也用于应用了 ATM 业务的端点 (如路由器)。应用适配层 (AAL) 仅用于端点。图 3.36 所示为这几层在 ATM 网络内部和外部的使用情况。

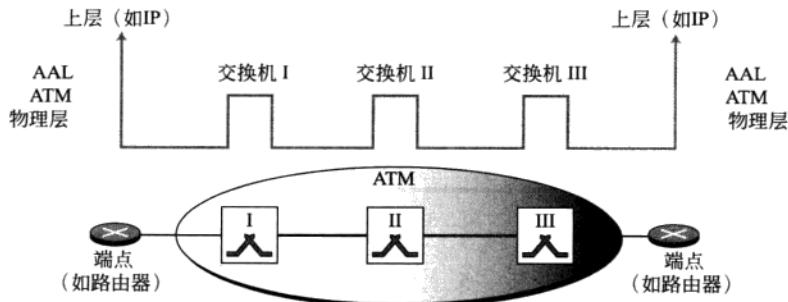


图 3.36 ATM 层的应用情况

AAL 层

应用适配层（Application Adaptation Layer, AAL）允许现有的网络（如分组交换网）与 ATM 设施连接。AAL 协议接受来自上层服务的传输（例如，分组数据），并把它映射为固定长度的 ATM 信元。这些传输可以是任何类型的（话音、数据、音频、视频），同时还可以具有固定的或可变的速率。在接收端，过程正好相反——这些信元被重装成原来的格式，并传递给正在接收的上层服务。虽然已经定义了四种 AAL 层，但我们感兴趣的是 AAL5，因为它用于运载因特网中的 IP 分组。

AAL5 有时也称为简单有效适配器层（Simple and Efficient Adaptation Layer, SEAL），它假设属于一个报文的所有信元都是按序传送的，并且所有控制功能都包括在上层的发送应用程序中。AAL5 是为无连接的分组协议而设计的，这种协议用数据报方式进行路由选择（如 TCP/IP 中的 IP 协议）。

IP 协议用的是 AAL5 子层。

AAL5 接受不超过 65535 字节的 IP 分组，并为其附加一个 8 字节的尾部，同时还有一些填充，这些填充用于保证其尾部正好处在接收设备所预期的位置上（也就是最后一个信元的最后 8 个字节），如图 3.37 所示。一旦填充及尾部准备好，AAL5 就以 48 字节的报文段将报文传递给 ATM 层。

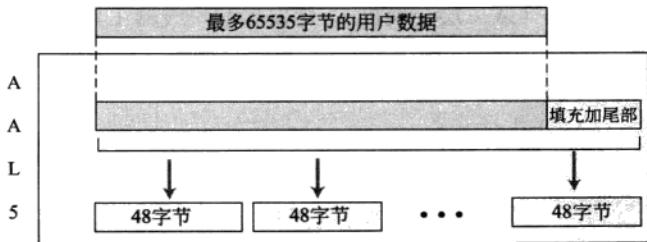


图 3.37 AAL5

ATM 层

ATM 层提供路由选择、通信量管理、交换和复用等服务。它这样处理要发送的通信量：从 AAL 子层接受 48 字节的报文段，再加上 5 个字节的首部后把它转换为 53 字节的信元（如图 3.38 所示）。

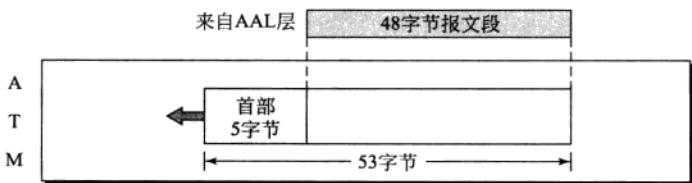


图 3.38 ATM 层

一个信元的长度是 53 字节，其中 5 字节分配给首部，其他 48 字节运载负荷（用户数据可能少于 48 字节）。首部中的大部分被 VPI 和 VCI 占据。图 3.39 描绘的是信元结构。

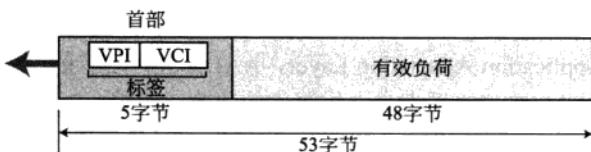


图 3.39 一个 ATM 信元

VPI 和 VCI 的组合可视为一个标记，用于指明某条特定的虚连接。

物理层

物理层定义了传输媒体、比特传输、编码方式以及电信号和光信号的转换。它提供与物理传送协议（如 SONET 和 T-3）的融合，同时也提供将信元流转换成比特流的机制。

3.5 连接设备

局域网和广域网通常都不是孤立工作的。它们或者彼此连接，或者连接到因特网。为了连接局域网和广域网，我们要使用连接设备。连接设备可以在因特网模型中的不同层次上工作。我们要讨论三种不同的连接设备（connecting device）：转发器（或集线器）、网桥（或两层交换机）和路由器（或三层交换机）。转发器和集线器工作在因特网模型中的第一层。网桥和两层交换机工作在前两层。路由器和三层交换机工作在前三层。图 3.40 给出了每一种设备所工作的层次。

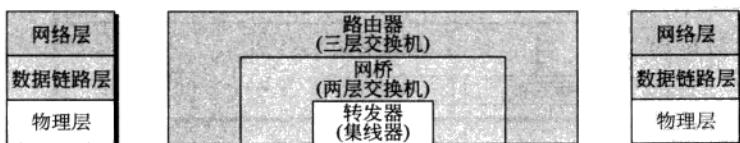


图 3.40 连接设备

3.5.1 转发器

转发器（repeater）是一种仅在物理层工作的设备。携带信息的信号在网络中只能传播

有限的距离，否则衰减会破坏信号的完整性。转发器在信号变得太弱或受到损伤之前接收这个信号，然后再生或重演原来的比特模式。转发器再把刷新后的信号发送出去。在过去，当以太网局域网还使用总线拓扑结构时，转发器的作用是连接一个局域网的两个网段从而克服同轴电缆的长度限制。不过，现在以太局域网使用的是星形拓扑。在星形拓扑结构中，一个转发器就是一个多端口设备，通常称为集线器（hub），它可用作一个连接点，同时又具有转发器的功能。图 3.41 描绘的是当从站 A 到站 B 的一个分组到达集线器时，这个分组的信号被再生，以消除任何可能的破坏性噪声，同时集线器还要将这个分组通过所有输出端口转发给局域网中所有的站。换言之，这个帧被广播了。局域网中的所有站都会收到这个帧，但只有站 B 会保留它，而其他站都会将其丢弃。图 3.41 描绘了一个转发器或一个集线器在交换局域网中所担当的角色。

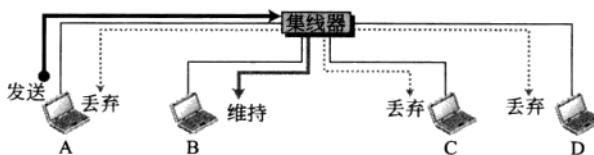


图 3.41 转发器或集线器

从图上可以很清楚地看出集线器不具有过滤功能，它不够聪明无法了解应当发送哪个端口的帧。

转发器转发每一个比特，它没有过滤功能。

集线器或转发器是一种物理层设备。它们本身不具备数据链路地址，同时也不会检查接收到的帧的数据链路地址。它们只是再生损坏的比特并将这些比特从每一个端口发送出去。

3.5.2 网桥

网桥（bridge）工作在物理层和数据链路层。作为一个物理层的设备，它再生接收到的信号；作为一个数据链路层的设备，网桥可以检查包含在帧中的 MAC 地址（源地址和目的地址）。

过滤

读者可能会问，网桥在功能上和转发器有什么区别？网桥有**过滤**（filtering）功能。它能检查帧的目的地址，然后决定应当从哪个出端口将这个帧发送出去。

网桥有一张表，可用于过滤判决。

让我们举一个例子。在图 3.42 中，有一个局域网，它用网桥连接了四个站。如果有一个目的地为 71:2B:13:45:61:42 的帧到达了端口 1，网桥就会查表找出转发的端口。根据这个表，要发送到 71:2B:13:45:61:42 的帧应当仅从端口 2 发送出去，因此没有必要向其他端口转发这个帧。

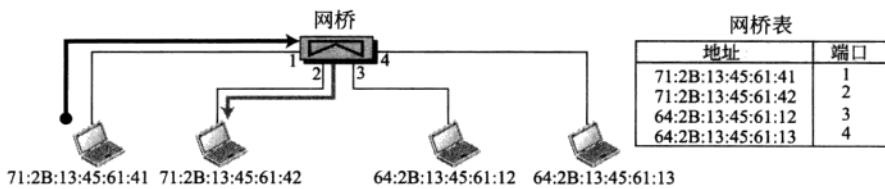


图 3.42 网桥

网桥不改变帧中的物理 (MAC) 地址。

透明网桥

透明网桥 (transparent bridge) 是可以令所有站都完全不知道它存在的一种网桥。如果一个网桥加入到系统中或从系统中删除，所有站都不需要重新配置。根据 IEEE 802.1d 规约，安装了透明网桥的系统必须符合以下三个准则：

1. 帧必须能够从一个站转发到另一个站。
2. 转发表是通过学习网络中帧的移动规律而自动生成的。
3. 系统中必须防止形成环路。

转发 正如前面一节所讨论的，透明网桥必须正确地转发帧。

学习 在最早的网桥中，转发表是静态的。系统管理员在设置网桥时必须人工输入每一个表项。虽然过程很简单，但实际效果不好。如果要加入或删除一个站，那么就必须人工地修改这个表。而且如果一个站的 MAC 地址改变了，那么这个表也必须人工修改，这种情况并不少见，例如，更换一块新的网卡就意味着换了一个新的 MAC 地址。

解决静态转发表问题的一个好办法是采用动态转发表，它能够自动地把地址映射为端口。要动态地构造转发表，我们需要能够从帧的运动规律中逐步学习的网桥。为此，网桥需要检查帧的目的地址和源地址。目的地址是用来进行转发判决 (查表) 的，而源地址则用于增加表项和更新表项的过程。让我们用图 3.43 来说明这个过程。

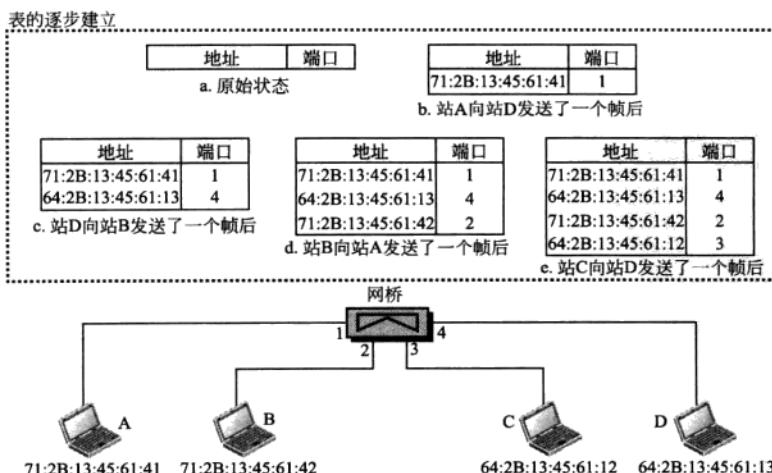


图 3.43 进行学习的网桥

1. 当站 A 向站 D 发送一个帧时，网桥中并没有 D 或 A 的表项。帧从所有三个端口转发出去，也就是用洪泛方式把帧转发到网络。不过，在查看源地址后网桥就知道站 A 一定连接到端口 1。也就是说，今后凡是发送到站 A 的帧一定要从端口 1 转发。网桥就把这个表项加入到表中。现在这个表有了自己的第一个表项。

2. 当站 D 向站 B 发送帧时，网桥没有站 B 的表项，因此再次使用洪泛方式。同时它又向表中增加一个表项。

3. 这个学习过程一直持续下去，直至转发表具有关于每一个端口的信息。

两层交换机

当我们使用交换机这个术语时必须很小心，因为交换机可以指两种不同的东西。我们必须增加这个设备所工作的层次以便使这个术语更明确。我们有两层交换机和三层交换机这两种交换机。两层交换机（two-layer switch）工作在物理层和数据链路层，它是一种具有快速转发能力的复杂网桥。

3.5.3 路由器

路由器（router）是一个三层设备，它工作在物理层、数据链路层和网络层。作为物理层的设备，它把接收到的信号进行再生。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，路由器则要检查网络层地址（在 IP 层中的地址）。请注意，网桥改变了碰撞的范围，而路由器则限制了广播的范围。

路由器是一个三层（物理层、数据链路层和网络层）设备。

路由器可以把多个局域网连接起来，也可以把多个广域网连接起来，还可以把多个局域网与多个广域网连接起来。换言之，路由器就是个网际互连设备，它把一些独立的网络连接起来构成一个互联网。按照这样的定义，两个网络（局域网或广域网）用路由器连接后就成为一个互联网络或互联网。

转发器或网桥连接的是一个局域网的各个网段。

路由器把几个独立的局域网或广域网连接起来，构成了互联网络（互联网）。

路由器与转发器或网桥相比有三个主要的区别：

1. 路由器的每一个接口都有一个物理地址和逻辑（IP）地址。
2. 路由器只在如下的分组到达时才发挥作用，即分组中的物理目的地址与分组抵达时的接口的物理地址相匹配。
3. 路由器在转发分组时要改变分组的物理地址（源地址和目的地址）。

让我们举个例子。在图 3.44 中，假设某单位有两幢独立的大楼，每幢大楼内都安装了一个吉比特以太局域网。该单位在每个局域网内都使用了网桥。这两个局域网可以互相连接起来，用 10G 以太网的技术形成一个更大的局域网，以提高到以太网的连接和到该单位服务器的连接速度。然后再用一个路由器将整个系统连接到因特网。

我们在第二章中已经知道路由器会改变它收到的分组的 MAC 地址，因为 MAC 地址仅在本地具有判决意义。

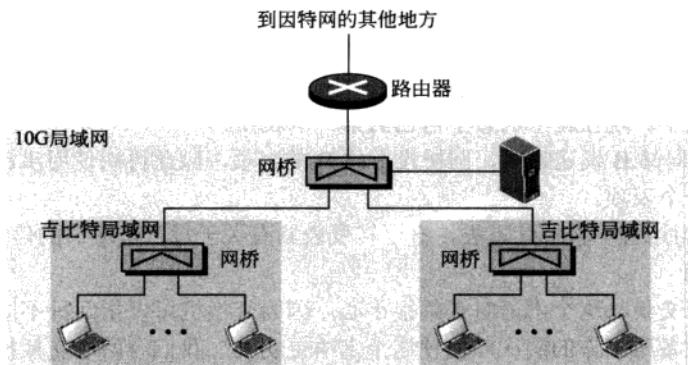


图 3.44 路由选择举例

我们将在讨论完 IP 编址后，在后面的几章将更多地学习路由器和路由选择的问题。

三层交换机

三层交换机 (three-layer switch) 就是路由器，它是一种在设计上有所改进并获得了更好的性能的路由器。三层交换机可以比传统路由器快得多地接收、处理和发送分组，虽然功能都是一样的。在本书中，为了避免混乱，我们使用术语路由器而不使用三层交换机。

路由器会改变分组中的物理地址。

3.6 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下几本书：[For 07]、[For 03]、[Tan 03] 和 [Gar & Wid 04]。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

3.7 重要术语

AAL5

接入点 (AP)

应用适配层 (AAL)

非对称数字用户线 (ADSL)

异步时分复用

异步传递方式 (ATM)

自动协商

按需带宽

基本服务集 (BSS)

蓝牙

网桥

BSS 切换移动性

电缆调制解调器 (CM)

电缆调制解调器传输系统 (CMTS)

有线电视

载波扩充

载波侦听多点接入 (CSMA)

具有碰撞避免的载波侦听多点接入 (CSMA/CA)

具有碰撞检测的载波侦听多点接入 (CSMA/CD)

信元

共用天线电视 (CATV)	以太网上的 PPP (PPPoE)
连接设备	主站
数字用户线 (DSL)	802 项目
数字用户线接入复用器 (DSLAM)	转发器
分布的帧间距 (DIFS)	路由器
下载	分散网
下行数据频带	从站
ESS 切换移动性	短帧间距 (SIFS)
以太网	简单有效适配器 (SEAL)
扩展的服务集 (ESS)	标准以太网
快速以太网	对称数字用户线 (SDSL)
光纤结点	同步数字系列 (SDH)
过滤器	同步光纤网 (SONET)
帧突发	同步传送信号 (STS)
帧中继	T 线
吉位以太网	T-1 线
握手期	T-3 线
头端	10 G 以太网
十六进制记法	三层交换机
高数据率数字用户线 (HDSL)	传输路径 (TP)
集线器	透明网桥
混合光纤同轴 (HFC) 网络	两层交换机
IEEE 802.11	上载
干扰信号	上行数据频带
链路控制协议 (LCP)	V.90
逻辑链路控制 (LLC)	V.92
媒体接入控制 (MAC)	甚高数据率数字用户线 (VDSL)
网络分配向量 (NAV)	视频频带
网络控制协议 (NCP)	虚电路 (VC)
网络接口卡 (NIC)	虚电路标识符 (VCI)
无切换移动性	虚通道 (VP)
光载波 (OC)	虚通道标识符 (VPI)
微微网	无线局域网
点协调功能 (PCF)	X.25
点到点协议 (PPP)	

3.8 本章小结

□ 局域网 (LAN) 就是设计在有限地理范围内使用的计算机网络。在局域网的市场上

出现过多种技术，如以太网、令牌环、令牌总线、FDDI 和 ATM 局域网等等。其中有些技术也存在过一段时间，但以太网才是占有绝对优势的技术。以太网经过了长期的发展过程。目前占绝大多数的以太网版本是吉比特以太网和 10 G 以太网。

- 无线局域网的一个最重要的标准是 IEEE 802.11 定义的标准，有时也被称为无线以太网。另一种比较流行的技术是蓝牙，它是一种设计用于连接具有不同功能的设备（如电话、笔记本、台式计算机、照相机、打印机、煮咖啡壶等等）的无线局域网技术。
- 点到点广域网技术提供了通过常规电话线路和传统调制解调器、DSL 线路、电缆调制解调器、T 线或 SONET 网络对因特网的直接连接。点到点协议（PPP）是为需要用可靠的点到点接入因特网的用户而设计的。PPP 工作在 OSI 模型的物理层和数据链路层。
- 交换广域网技术提供因特网中的主干连接。异步传递方式（ATM）是信元中继的协议，用来支持数据、话音和视像在高速率传输媒体（如光纤）上的传输。
- 连接设备可以把网络的几个网段相互连接起来，也可以把几个网络连接起来形成一个互联网。共有三种类型的连接设备：转发器（和集线器）、网桥（和两层交换机）、路由器（和三层交换机）。转发器在物理层对信号进行再生。集线器是多端口转发器。网桥可以访问站的地址，并在网络中转发或过滤分组，它们工作在物理层和数据链路层。两层交换机是一种复杂的网桥。路由器判断一个分组应当沿什么路径传送，它们工作在物理层、数据链路层和网络层。三层交换机是一种复杂的路由器。

3.9 实践安排

3.9.1 习题

1. 假定 10Base5 的电缆长度是 2500 m。如果信号在粗同轴电缆上的传播速率是 200 000 000 m/s，试问将一个比特从网络的这一端传到另一端需要多少时间？忽略设备中的任何传播时延。

2. 使用习题 1 中的数据，找出侦听到一次碰撞所需的最长时间。最坏的情况发生在：数据从电缆的一端发送而碰撞发生在另一端。请记住，信号需要走一个往返。

3. 10Base5 的数据率是 10 Mbps。产生一个最小的帧需要多少时间？给出你的计算过程。

4. 用习题 2 和 3 中的数据，找出使碰撞检测能正常工作的以太网帧的最小长度。

5. 以太网的 MAC 子层从 LLC 子层收到 42 字节的数据。试问在这个数据上还要填充多少字节？

6. 以太网的 MAC 子层从 LLC 子层收到 1510 字节的数据。这样的数据能够封装成一个帧吗？如果不可以，必须发送多少个帧？每个帧中的数据有多长？

7. 试比较 CSMA/CD 和 CSMA/CA。

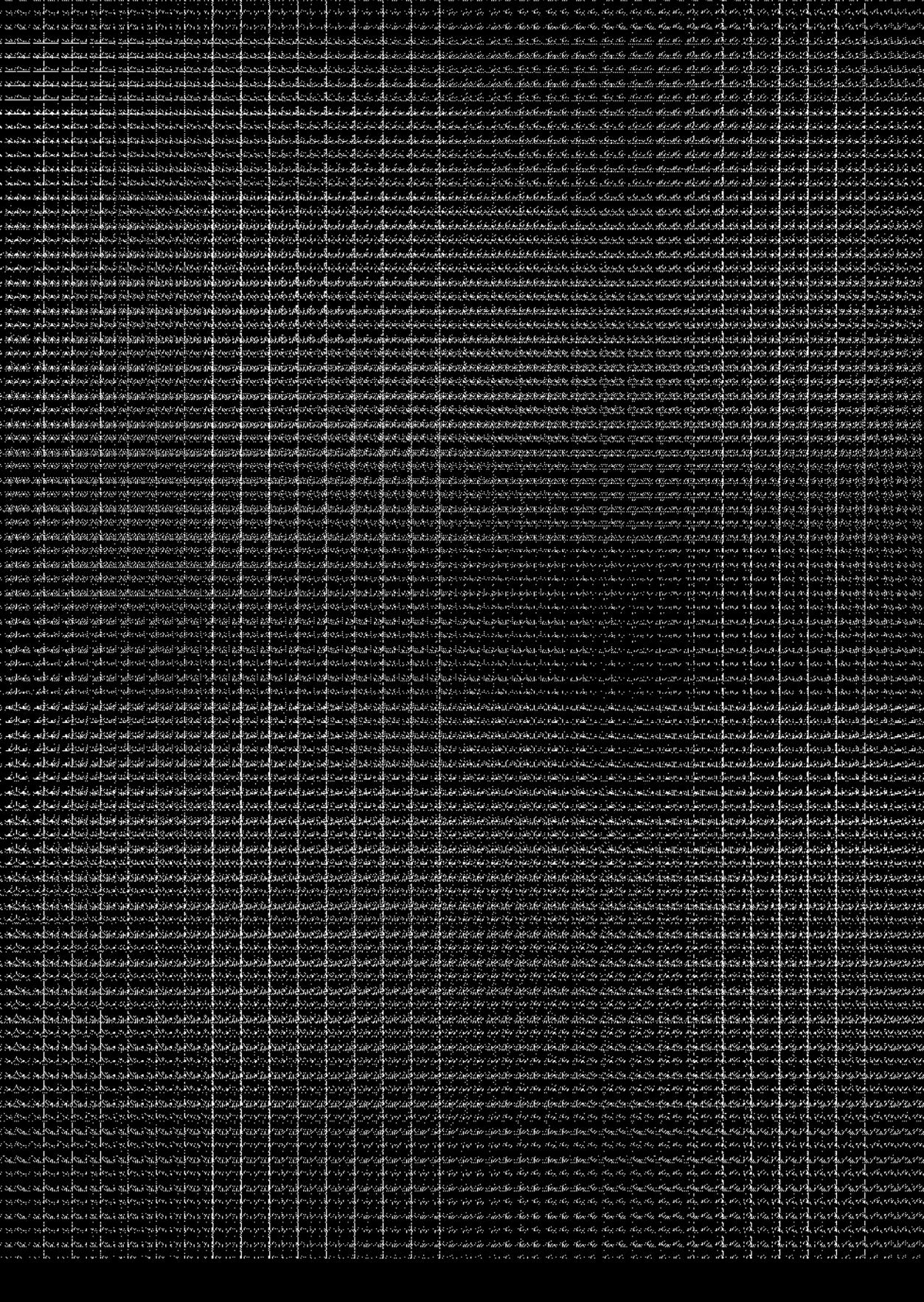
8. 试使用表 3.10 比较 IEEE 802.3 和 IEEE 802.11 中的各字段。

表 3.10 习题 8

字段	IEEE 802.3 字段长度	IEEE 802.11 字段长度
目的地址		
源地址		
地址 1		
地址 2		
地址 3		
地址 4		
FC		
D/ID		
SC		
PDU 长度		
数据和填充		
帧主体		
FCS (CRC)		

3.9.2 研究活动

9. 传统以太网使用了 CSMA/CD 接入方法的一个版本。它称为 1 坚持的 CSMA/CD。试找出关于这个方法的一些信息。
10. DSL 使用一种称为 DMT 的调制技术。试找出关于这种调制技术的一些信息，以及在 DSL 中它是怎样使用的。
11. PPP 要经历几个阶段，这可用状态转换图来表示。试为一条 PPP 连接画出状态转换图。
12. 试找出 LCP 分组（封装在 PPP 帧中）的格式，包括所有的字段、它们的代码和用途。
13. 试找出 NCP 分组（封装在 PPP 帧中）的格式，包括所有的字段、它们的代码和用途。
14. 试找出 ICP 分组（封装在 PPP 帧中）的格式，包括所有的字段、它们的代码和用途。
15. PPP 使用了两个鉴别协议：PAP 和 CHAP。试找出关于这两种协议的一些信息，以及在 PPP 中它们是怎样使用的。
16. 试找出如何使用 AAL5 层将一个 IP 分组封装在 ATM 信元中的信息。
17. 为了避免在使用透明网桥的网络中出现环路，我们要使用支撑树算法。试找出关于这个算法的一些信息，以及它是如何防止环路的。



第二部分

网 络 层

第 4 章 网络层简介

第 5 章 IPv4 地址

第 6 章 IP 分组的交付和转发

第 7 章 网际协议版本 4 (IPv4)

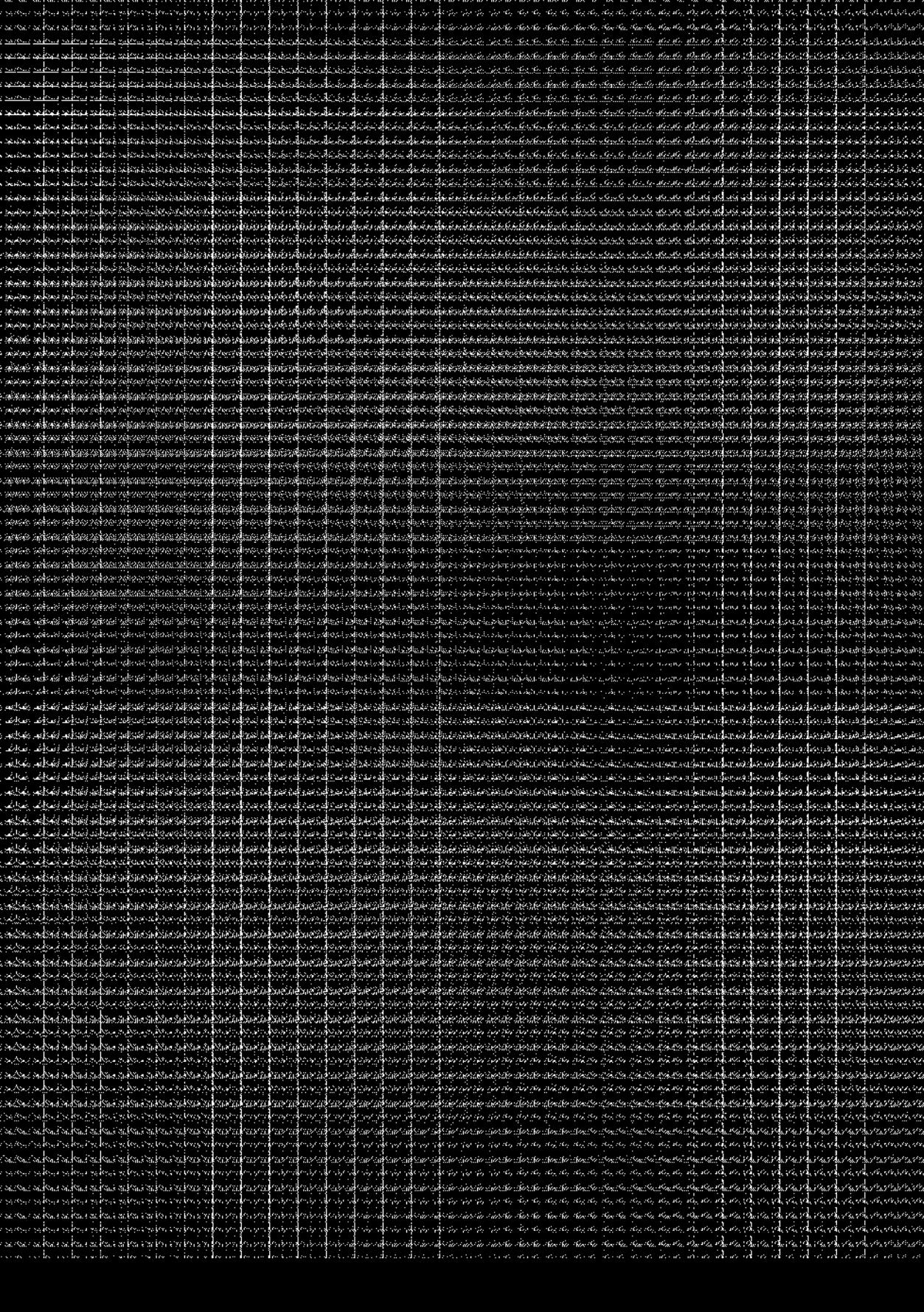
第 8 章 地址解析协议 (ARP)

第 9 章 网际控制报文协议 (ICMP)

第 10 章 移动 IP

第 11 章 单播路由选择协议 (RIP、OSPF 和 BGP)

第 12 章 多播和多播路由选择协议



第4章 网络层简介

为了解决经由多条链路的交付问题，人们设计了网络层（有时也称为互联网层）。网络层负责主机到主机的交付，并且在分组经过路由器时负责为其选择路由。

在本章中，我们将对网络层做简单扼要的介绍，为读者在此之后从第 5 章到第 12 章更全面细致地了解网络层做好准备。本章我们将从原理上解释网络层存在的必要性以及因此而带来的种种问题。不过，如果想要深入透彻地解答这些问题，我们还需要此后八章的内容，甚至我们可能需要在读完全书后才能得到令人满意的答案。

目标

本章有以下几个目标：

- 简单介绍交换技术，特别是作为网络层数据交付机制的分组交换技术。
- 讨论分组交换网提供的两种完全不同类型的服务：无连接服务和面向连接的服务。
- 讨论在无连接的分组交换网中，路由器是如何利用分组中的目的地址以及一张路由表来转发分组的。
- 讨论在面向连接的分组交换网中，路由器是如何利用分组中的标号以及一张路由表来转发分组的。
- 讨论由网络层直接提供的服务，如逻辑地址、源点的交付、各路由器的交付和终点的交付。
- 讨论网络层协议没有直接提供，但可以通过某些辅助协议或因特网后追加的协议来提供的某些内容或服务。

4.1 简介

从概念上看，我们可以认为全球因特网是一个将全世界数以亿计的计算机连接起来的暗箱网络，此时人们关注的仅仅是有一个报文从一台计算机的应用层到达了另一台计算机的应用层。从这个层面上看可以认为计算机 A 和 B 之间的通信如图 4.1 所示。

但实际上因特网不是一个单一的网络，而是由许许多多网络（或链路）通过连接设备互相连接在一起的。换言之，因特网就是一个互联网，是许多局域网和广域网的组合。为了更好地理解网络层（或者说互联网层）的地位和作用，我们需要考虑的不是概念上的那个因特网，而是由这些局域网和广域网组成的因特网。在这里我们不可能显示出所有这些局域网和广域网，因此我们虚构了一个小型互联网，它只有很少几个网络和连接设备，如图 4.2 所示。



图 4.1 视因特网为一个暗箱

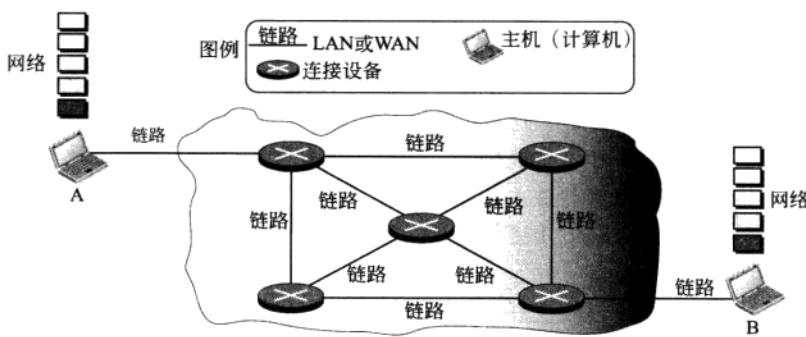


图 4.2 视因特网为多个局域网和广域网互相连接的组合

在这个模型中，像路由器这样的连接设备担当着交换机的作用。当一个分组从某个端口（接口）到达该路由器后，会通过另一个端口转发到下一部交换机（或终点设备）。换言之，我们称之为交换（switching）的处理过程发生在连接设备上。

4.2 交换

根据我们前面的讨论，可以很明显地看出报文在从源点到终点的旅途中要做出许多判决。当一个报文抵达某个连接设备时，必须做出一个判决以选择该分组应当从哪个输出端口发送出去。换言之，连接设备就像交换机那样将一个端口与另一个端口连接起来。

4.2.1 电路交换

有一种交换方式称为电路交换（circuit switching），也就是说在报文传递之前，该报文的源点和终点之间先要建立了一条物理电路（或信道）。在电路已经建立的前提下，报文完整地从源点传送到终点，然后源点就可以通知网络传输已完成，此时网络才能开放所有交换机，并向另一条连接提供链路以及连接设备。电路交换从未在网络层上应用过，它主要用于物理层。

在电路交换中，完整的、没有被分割成分组的报文从源点发送到终点。

例 4.1

电路交换网的一个最典型的例子就是早期的电话系统，当呼叫方拨打了被叫方的电话号码后，在呼叫方和被叫方之间就建立了一条通路。当被叫方接听电话后电路建立。此后语音报文就可以在双方之间双向传递，同时所有的连接设备都在维护着这条电路。当呼叫方或者被叫方挂起电话，电路则终止。时至今日，电话网已经不完全是电路交换网了。

4.2.2 分组交换

第二种交换方式称为分组交换（packet switching）。目前因特网的网络层就是一个分组交换网。在这种类型的网络中，来自上层的报文被分割成便于管理的一个个分组，再通过网络发送这些分组。报文的源点逐个发送分组，而其终点也逐个接收这些分组。等到属于该报文的所有分组都到齐之后，终点才能将报文交付给上层。在分组交换网中，连接设备仍然需要判决如何为这些分组选择路由使之顺利到达终点。目前，分组交换网为分组选择路由的方式有两种：数据报方式和虚电路方式。我们将在下一小节讨论这两种方式。

在分组交换网中，源点在传送之前先要将报文分割成便于管理的分组。这些分组在到达终点后被重新组装。

4.3 网络层的分组交换

网络层被设计为一个分组交换网。这就是说，报文要在源点被分割成便于管理的分组，通常称之为数据报（datagram），然后逐个地将这些数据报从源点传送到终点。终点接收到的数据报经过组装后重新生成原始的报文。因特网的分组交换的网络层在最初设计时只提供无连接服务，不过最近的趋势是向面向连接的服务转变。我们首先要讨论一下占主导地位的前者，然后再简单地介绍一下新面孔。

4.3.1 无连接服务

在因特网诞生之初，网络层被设计为提供无连接服务（connectionless service），此时网络层的协议独立地对待每一个分组，每个分组与其他分组之间没有任何关联。一个报文的所有分组可能会，也可能不会沿着相同的路径抵达终点。在因特网创建时之所以会决定让网络层提供无连接服务就是为了简单。主要是想让网络层只负责从源点到终点的分组交付。图 4.3 描绘了其基本思想。

网络层提供无连接服务时，在因特网中穿梭的每个分组都是一个独立的个体，属于同一个报文的两个分组之间没有任何关联。此类网络中的交换机被称为路由器。属于某个报文的分组之后紧跟着的可能是同一个报文的分组，也可能是另一个报文的分组。一个分组后面跟着的可能是来自相同源点的分组，也可能是来自另一个不同源点的分组。

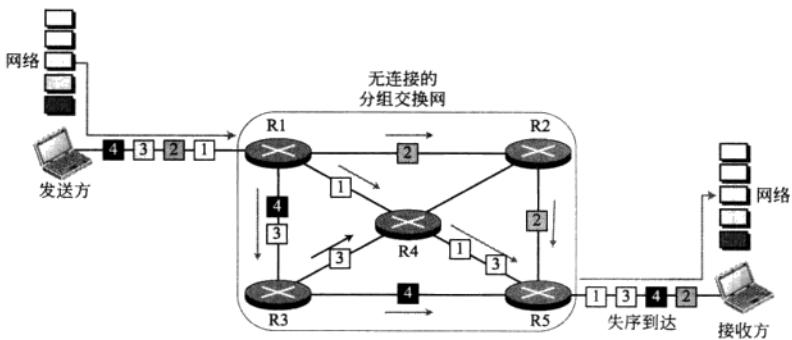


图 4.3 无连接的分组交换网

每个分组在选择路由时都要依据包含在其首部中的信息：源地址和目的地址。目的地址指明了它要到哪去，而源地址则指明了它来自哪里。在这种情况下，路由器仅仅根据其目的地址来为它选择路由。源地址则可用于该报文被丢弃的时候向源点发送差错消息。图 4.4 描绘了在这种情况下路由器采用的转发过程。

在此我们使用了符号地址如 A 和 B。

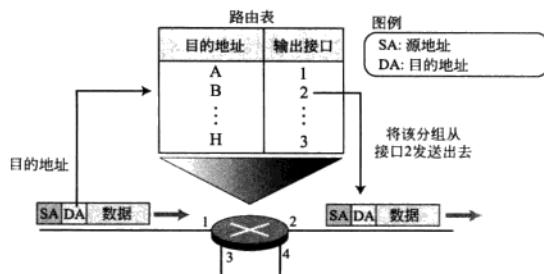


图 4.4 无连接网络中路由器的转发过程

在无连接分组交换网中，转发判决的依据是该分组的目的地址。

无连接网络中的时延

如果我们忽略分组可能会丢失并重传的事实，同时也不计较到了终点可能还需要等待所有分组都被接收的事实，无连接网络中的时延模型如图 4.5 所示。

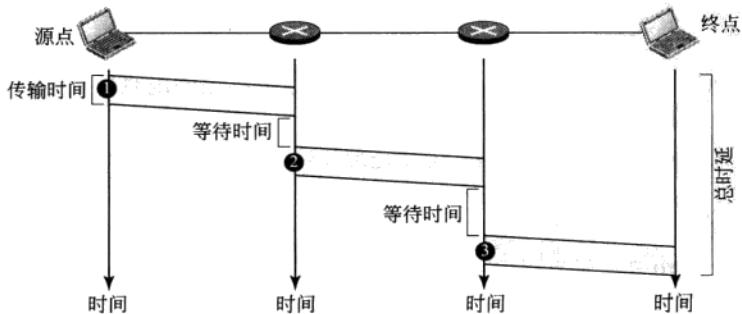


图 4.5 无连接网络中的时延

4.3.2 面向连接的服务

在面向连接的服务（connection-oriented service）中，属于同一个报文的所有分组之间是有关联的。在一个报文的数据报被发送之前，应当首先建立一条虚连接以指定这些数据报通过的路径。在这种类型的服务中，分组不仅要包含源地址和目的地址，同时还要包含一个流标号（flow label），也就是一个虚电路标识符（virtual circuit identifier），用来指定这些分组应当采取的虚路径。稍后我们将会介绍流标号是如何判定的，不过现在我们假设分组中都携带了这个标号。虽然看起来好像有了流标号，源地址和目的地址就没用了，但是由于某些原因使得采用了无连接服务的部分因特网网络层仍然需要保留这两个地址。其中一个原因就是该部分的分组路径可能仍然使用无连接的服务。另一个原因是网络层的协议在设计之时就涉及到了这两个地址，并且要改变这一现状可能还需要一段时间。图 4.6 所示为面向连接服务的概念。

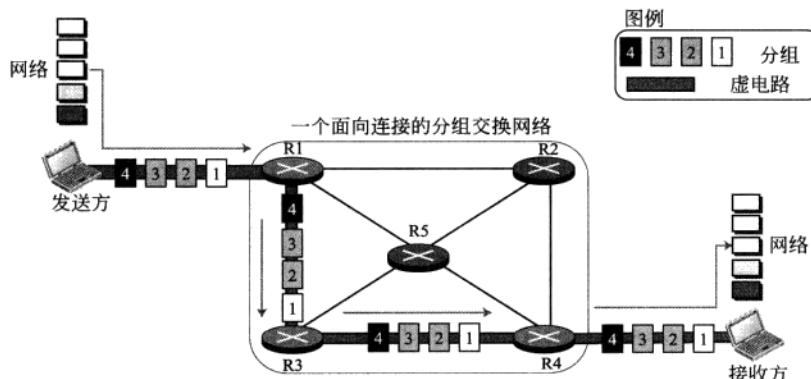


图 4.6 面向连接的分组交换网

各分组的转发依据是包含在分组中的标号。为了更好地理解面向连接的设计理念，先假设分组在到达路由器时都已携带了一个标号，如图 4.7 所示。

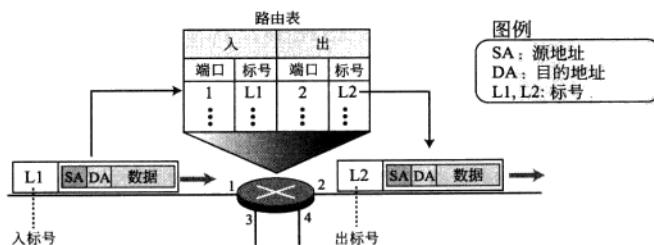


图 4.7 面向连接的网络中路由器的转发过程

在这种情况下，转发判决要依据这个标号（有时也称为虚电路标识符）的值。

要建立面向连接的服务必须经过以下三个阶段的处理过程：建链、数据传送和拆链。在建链阶段，发送方的源地址和接收方的目的地址被用来生成面向连接服务路由表中的一

个表项。在拆链阶段，源点和终点通知路由器删除相应的表项。数据传送则发生在这两个阶段之间。

在面向连接的分组交换网络中，转发判决的依据是该分组的标号。

建链阶段

在建链阶段 (setup phase)，路由器要为一条虚电路创建一个表项。例如，假设源点 A 需要创建一条到达终点 B 的虚电路。在发送方和接收方之间需要交换两个辅助分组：请求分组和确认分组。

请求分组 请求分组从源点发往终点。这个辅助分组携带源地址和目的地址。图 4.8 描述了这一过程。

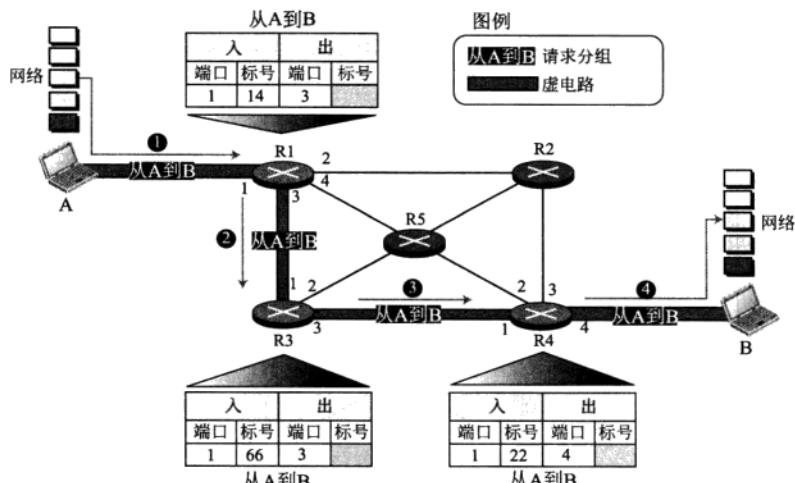


图 4.8 在虚电路网络中发送请求分组

- 源点 A 向路由器 R1 发送一个请求分组。
- 路由器 R1 接收到这个请求分组。它知道从 A 来到 B 去的分组应当通过端口 3 发送出去。至于路由器是如何得到这个信息的，将在后面的章节中再讨论。在这里假设路由器已经知道了正确的输出端口。路由器在自己的路由表中为这条虚电路创建一个表项，但是在这个表项的四列里，它还只能填写三列。路由器为其设置了入端口 (1)，并选择一个有效的入标号 (14)，以及出端口 (3)。在现阶段它还不知道出标号是多少，这要等到确认阶段才能知道。接着路由器通过端口 3 将该分组转发给路由器 R3。
- 路由器 R3 接收到这个建链请求分组。之后发生的情况与在路由器 R1 中完全一样，R3 的路由表中新建了一个表项并填写其中的三列：入端口 (1)、入标号 (66) 和出端口 (3)。
- 路由器 R4 接收到这个建链请求分组。同样，它的路由表中也填写了三列：入端口 (1)、入标号 (22) 和出端口 (4)。
- 终点 B 接收到这个建链请求分组，如果它已经准备好接收来自 A 的分组，那么它就为从 A 发送来的入分组分配一个标号，在这里是 77。这个标号可以让终点分辨出这些分组是来自源点 A，而不是来自其他源点。

确认分组 一个称为确认分组的特殊分组用于完成刚才建立的那些表项。图 4.9 描绘了这一过程。

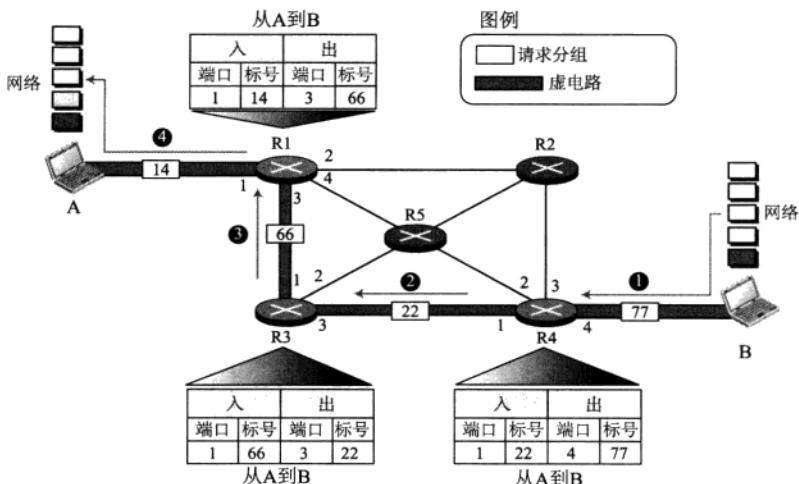


图 4.9 在虚电路网络中的建链确认过程

1. 终点向路由器 R4 发送一个确认分组。这个确认分组携带着全局的源地址和目的地址，有了它们路由器才能知道应当去填写并完成路由表中的哪一个表项。这个分组还携带着标号 77，这是终点为来自 A 的分组所选择的入标号。路由器 R4 就用这个标号来填写路由表相应表项的出标号那一列。请注意，77 在终点 B 上是入标号，而在路由器 R4 上则是出标号。

2. 路由器 R4 向路由器 R3 发送一个确认分组，其中包含了在它的路由表中相对应的入标号，这个入标号是在建链阶段选定的。路由器 R3 就用这个标号作为自己路由表中对应的出标号。

3. 路由器 R3 向路由器 R1 发送一个确认分组，其中包含了在它的路由表中相对应的入标号，这个入标号是在建链阶段选定的。路由器 R1 就用这个标号作为自己路由表中对应的出标号。

4. 最后路由器 R1 向源点 A 发送一个确认分组，其中包含了它的路由表中相对应的入标号，这个入标号也是在建链阶段选定的。

5. 源点利用确认分组中 R1 的入标号作为其发往终点 B 的数据分组的出标号。

数据传送阶段

第二个阶段称为数据传送阶段 (data transfer phase)。当所有路由器都为特定的虚电路创建了各自的路由表项之后，属于同一个报文的所有网络层分组就可以一个接一个地发送出去。在图 4.10 中，我们只描绘了一个分组的流转过程，不过这个过程都是相同的，不管是一个、两个、还是一百个分组。源点计算机利用了标号 14，这个标号就是在建链阶段从路由器 R1 处得到的。路由器 R1 将这个分组转发到路由器 R3，不过它会将这个标号改为 66。路由器 R3 将分组转发到路由器 R4，但是这个标号将被改为 22。最后，路由器 R4 将分组交付到它的终点，此时的标号是 77。该报文的所有分组都遵照相同的标号序列抵达它们的终点。在终点处，分组是按序到达的。

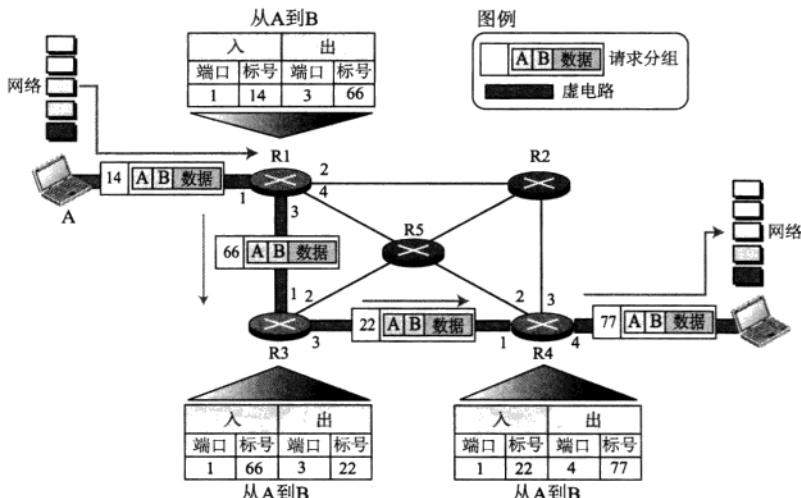


图 4.10 一个分组在已建立的虚电路中的流动过程

拆链阶段

在拆链阶段 (teardown phase)，源点 A 在向终点 B 发送了所有的分组之后就会发送一个称为拆链分组的特殊分组。终点 B 用一个证实分组来响应。所有的路由器从各自的路由表中将对应的表项删除。

面向连接网络中的时延

如果我们忽略分组可能会丢失并被重传的事实，面向连接网络中的时延模型如图 4.11 所示。

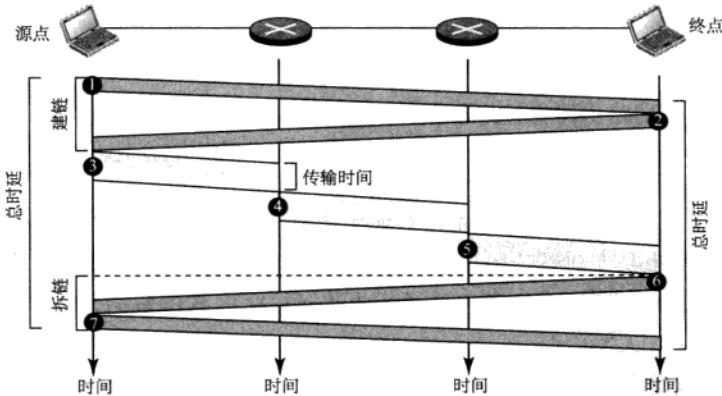


图 4.11 面向连接网络中的时延

4.4 网络层的服务

相比于无连接服务，多了建立连接 和 拆除连接的过程，不过只要进行一次的路由选择，不用每个分组都进行路由选择

在本节我们将简单地讨论一下由网络层提供的服务。我们的讨论绝大部分都是围绕目前在因特网上占主导地位的基于无连接的服务。

4.4.1 一个例子

为了更好地理解网络层的服务，我们先来举一个例子。在图 4.12 中，假设在 Wonderful Publishing 出版公司工作的爱丽丝需要向 Just Flowers 花店的经理鲍勃发送一条消息，通知他花店的广告册已经印好了，随时可以发货。爱丽丝使用电子邮件来发送这条消息。让我们一起跟随想象中该消息从爱丽丝到鲍勃所经过的路线。Wonderful Publishing 出版公司有一个局域网，它通过一个有线广域网连接到称为 BestNet 的本地 ISP。Just Flowers 公司也使用了一个局域网，并通过 DSL 广域网连接到另一个称为 ServeNet 的本地 ISP。这两个本地 ISP 通过高速 SONET 广域网连接到一个全国性的 ISP。爱丽丝发送给鲍勃的消息可能会被分割为多个网络层的分组。稍后我们就会知道这些分组可能是，也可能不是沿着相同的路径前进。为了便于讨论，我们只跟随一个分组从爱丽丝的计算机来到鲍勃的计算机。同时我们还要假设该分组在到达终点之前途经了路由器 R1、R3、R5、R6 和 R8。两台计算机要涉及到 TCP/IP 协议族的五层，而路由器只涉及三层。

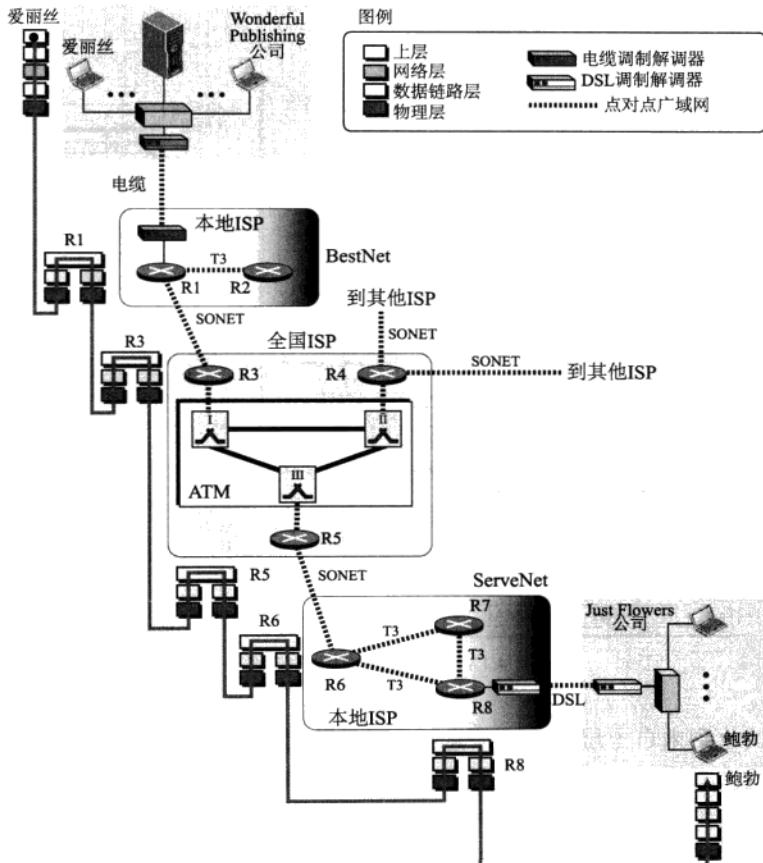


图 4.12 一个想象中的部分因特网

4.4.2 逻辑编址

因为网络层提供了端到端的通信，所以两台想要进行通信的计算机就必须具有全球标识系统，称为网络层地址或逻辑地址。此类标识是通过全球统一的编址机制为网络层提供的。因特网有一个地址空间，每一个要使用因特网的实体都必须从这个地址空间中分配一个唯一的地址。在第 5 章中将会讨论因特网版本 4 的这个地址空间。在第 26 章，将讨论版本 6（版本 5 从未实现过）中新的编址体系。在图 4.12 中，爱丽丝和鲍勃需要有两个网络层的地址才能互相通信。

4.4.3 源计算机提供的服务

源计算机上的网络层提供四种服务：分组化处理、查找下一跳的逻辑地址、查找下一跳的物理（MAC）地址以及对数据报进行必要的分片处理。图 4.13 描绘了这些服务。

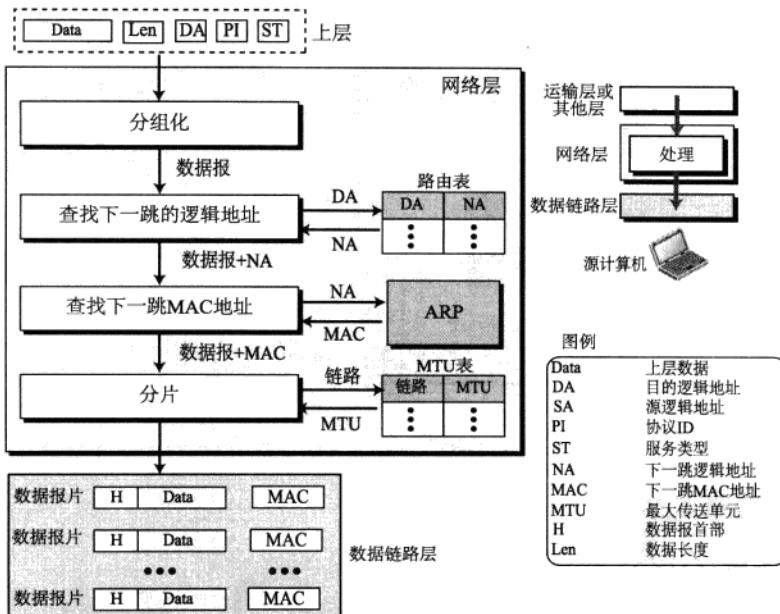


图 4.13 源计算机提供的服务

网络层接收到来自上层的几个信息：数据、数据长度、逻辑目的地址、协议 ID（网络层使用协议的标识号）以及服务类型（稍后再讨论）。网络层对这些信息进行处理后生成了一组数据报片以及下一跳的 MAC 地址，并将它们一起交付给数据链路层。在这里我们只对每种服务进行简单讨论，而本书这一部分的后面几个章节将会更详细地加以解释。

分组化处理

网络层的第一个任务就是要将来自上层的数据封装到一个数据报中。这件事的做法是为数据添加一个首部，其中包含了该分组的逻辑源地址和逻辑目的地址、分片相关的一些信息、请求了此服务的协议的 ID、数据长度，再加上其他一些可能的选项。网络层还要包含一个只计算该数据报首部的检验和。我们将在第 7 章中给出这个数据报的格式以及检验和的计算方法。请注意，上层协议仅提供了逻辑目的地址，而逻辑源地址则来自网络层本身（任何一个主机都要知道它自己的逻辑地址）。

查找下一跳的逻辑地址

准备好的数据报包含了该分组的源地址和目的地址。如前所述，这个数据报可能需要途经多个网络才能到达它的终点。如果目的计算机与源计算机没有连接在同一个网络上，那么数据报就应当交付给下一个路由器。数据报的源地址和目的地址中没有关于下一跳逻辑地址的任何线索。源计算机上的网络层需要咨询路由表并找出下一跳的逻辑地址。

查找下一跳的 MAC 地址

网络层并不是真的将数据报交付给下一跳，而是应该由数据链路层来负责真正的交付。数据链路层需要下一跳的 MAC 地址才能向其交付。为了查找下一跳的 MAC 地址，网络层可以使用另一张表以便将下一跳的逻辑地址映射为 MAC 地址。不过，根据我们在第 8 章中讨论的某些理由，这个任务被分配给了另一个称为地址解析协议（ARP）的辅助协议，它可根据给出的逻辑地址找到相应的 MAC 地址。

分片

到了这一步，数据报可能还是没有准备好交付给数据链路层。在第 3 章中我们已经知道，大多数局域网和广域网都会对一个帧所携带的数据的最大长度(MTU)加以限制，而网络层所准备的数据有可能会超过这个限制。在被传递到数据链路层之前，数据报需要被分割为更小的单元。分片操作必须要保留数据报首部中的信息。换言之，虽然数据被分片了，首部却需要重复多次。另外在这个首部中还需要增加额外的信息，以指明该数据片在整个数据报中的位置。我们将在第 7 章中更详细地讨论分片问题。

4.4.4 各路由器提供的服务

如我们前面曾提到的，一个数据报要涉及到路由器上的两个接口：一个入接口和一个出接口。因此路由器上的网络层需要与两个数据链路层打交道：入接口的数据链路层和出接口的数据链路层。网络层负责接收来自入接口数据链路层的数据报，如有必要需对其进行分片处理，然后将这个数据报片交付给出接口的数据链路层。通常路由器不涉及上层（有一些例外情况将在后面的章节讨论）。图 4.14 描绘了一个路由器的网络层提供的服务。

在这里，三个处理过程（查找下一跳的逻辑地址、查找下一跳的 MAC 地址以及分片）与在源点时提到的后三个处理过程是一样的。不过，在应用这些处理过程之前，路由器首先要通过检验和来检查数据报的有效性（参见第 7 章）。这里的有效性指的是数据报的首部有没有损坏，且数据报是否被交付给正确的路由器。

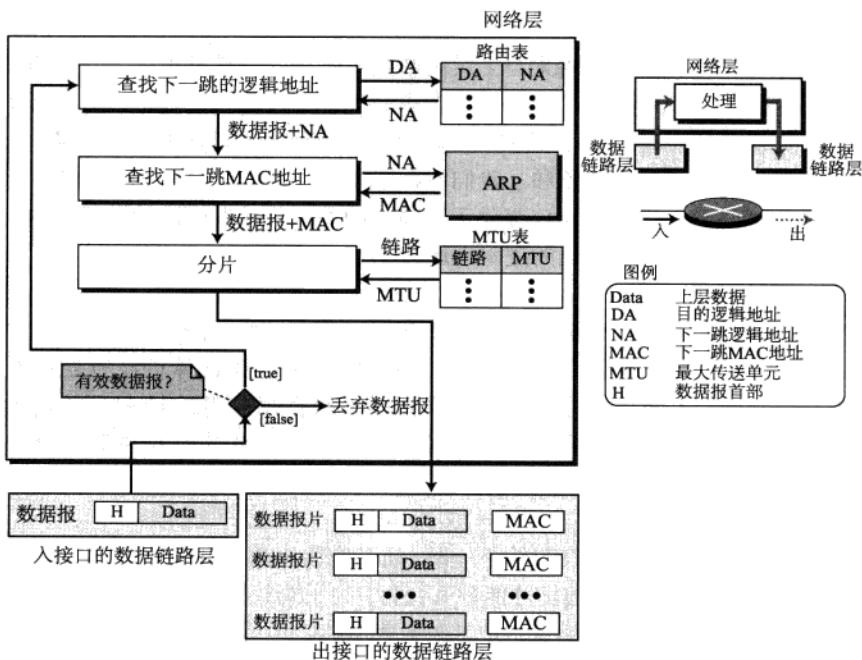


图 4.14 在各路由器上的处理过程

4.4.5 目的计算机提供的服务

目的计算机上的网络层要简单一些，因为不再需要转发了。但是，目的计算机在向终点交付这些数据之前需要重组数据报片。在检查过数据报片的有效性之后，要从中将数据提取出来并加以保存。当所有的数据报片都到齐之后，数据重组并交付给上层。与此同时，网络

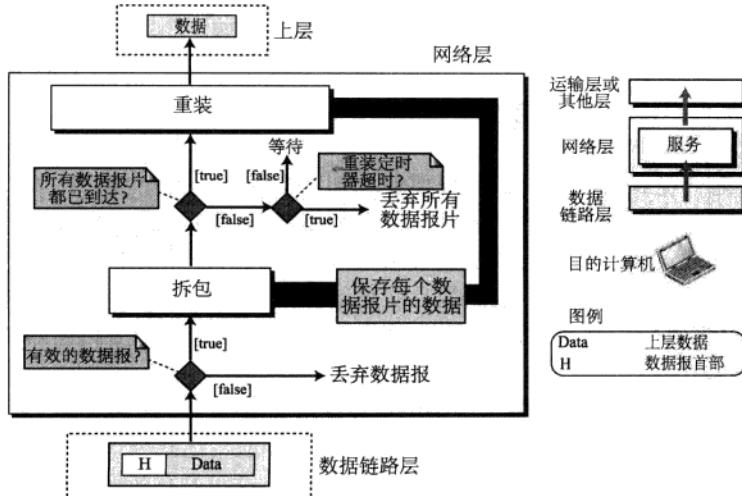


图 4.15 目的计算机上的处理

层还要设置一个重组定时器。如果这个定时器超时，所有的数据报片都会被销毁，并发送一个差错报文，此时所有的数据报片都需要被重新发送。图 4.15 描绘了这一过程。请注意，分片处理对上层来说是完全透明的，因为在所有的数据报片都到达并且被重组之前，网络层不会向上层交付任何不完整的数据。一个数据报可能在源计算机上已经分片过了，但还是可以在任一台路由器上再次（多级）分片，所以重组过程是非常精细和复杂的。我们将在第 7 章中详细讨论这一过程。

4.5 其他与网络层相关的问题

在本节，我们将简单介绍一些与网络层相关的问题。这是一些通常应当在网络层讨论的服务，但实际上它们在网络层只是部分实现，或者根本没有实现。有些服务是由一些辅助协议来提供的，或者是通过因特网上后来附加的协议实现的。其中的大多数内容在后面的章节中还会看到。

4.5.1 差错控制

差错控制（error control）包括对损坏、丢失以及重复的数据报进行检测的机制。差错控制还包括在检测到错误之后的纠错机制。因特网的网络层不提供真正意义上的差错控制机制。从表面上看网络层好像是不需要差错控制的，因为每个数据报在到达终点之前都要穿过多个网络，而控制这些网络（局域网或广域网）行为的数据链路层已经使用了差错控制机制。换言之，既然在数据链路层已经实施了逐跳的差错控制，为什么网络层还需要差错控制呢？虽然逐跳的差错控制在一定程度上能够起到保护数据报的作用，但是它的保护还不够彻底。图 4.16 描绘了在数据报途经的某些地方出现的一些差错没有能够检测出来。在路由器处理数据报时出现的差错，数据链路层是无法检测出来的。

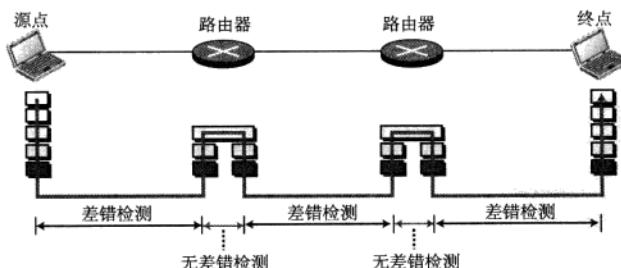


图 4.16 数据链路层上的差错检测

网络层的设计者们希望让这一层的操作既简单又迅速。他们认为如果确实需要更为严格的差错检测，也可以交给调用了网络层服务的上层协议来完成。网络层之所以忽略差错检测的另一个理由与分片有关。由于数据很可能在某些路由器上被分片，因而使部分网络层的数据可能会被改变。如果我们要使用差错控制，就必须在每个路由器上检测是否有分

片。这会使得网络层上的差错检测变得非常低效。

不过，网络层的设计者还是在数据报上增加了一个检验和字段（参见第 7 章），它可以控制在首部中出现的任何损坏，而不是针对整个数据报的。这个检验和可以在两跳之间或从一端到另一端之间防止出现在数据报首部中的任何变动或损坏。例如，它可以防止因目的地址被损坏而将数据报交付到了一个错误的终点。但是，因为在每个路由器上这个首部都可能会有一些变化，所以在源点时需要计算这个检验和，而到了每个路由器上还需要重新计算一遍。

我们还要提到的是，虽然因特网的网络层没有直接提供差错控制，但是因特网使用了另一个协议，ICMP。如果数据报被丢弃或者是首部中含有一些不可知的信息，那么 ICMP 就会提供某种程度的差错控制。我们将在第 9 章中详细讨论 ICMP。

4.5.2 流量控制

流量控制 (flow control) 用于调整源点发送的数据量以免接收方超载。如果源计算机的上层应用产生数据的速度比目的计算机上层应用吸收数据的速度快，则接收方就会被这些数据淹没。为了控制数据的流量，接收方需要向发送方发送某些反馈，以通知发送方自己已被数据淹没了。

但是因特网的网络层并不直接提供任何流量控制。只要数据报准备好了，发送方就会将它们发送出去，而不会去管接收方是否准备好。

在目前版本的因特网中，网络层不提供任何流量控制。

有以下几个理由可以被用来解释为什么在设计网络层时会缺少流量控制。首先，因为在这一层没有差错控制，接收方网络层的工作非常简单，以致很少会出现超载现象。其次，使用网络层服务的上层协议也可以部署一些缓存，只要网络层的数据准备好了就把它们接收下来，这样处理数据的速度就不一定要和接收数据的速度一样快了。第三，绝大多数使用网络层服务的上层协议都会提供流量控制，因此，如果再加一层流量控制会使网络层变得太复杂，并且会降低整个系统的效率。

4.5.3 拥塞控制

网络层协议的另一个话题是**拥塞控制** (congestion control)。网络层的拥塞指的是这样一种状态，有过多的数据报出现在了因特网的某一个区域内。如果网络中的源计算机发送数据报的数量超过了网络或者路由器的容量，则有可能发生拥塞现象。在这种情况下，某些路由器可能会丢弃一些数据报。但是，数据报丢弃得越多，情况就可能变得越糟糕，原因在于上层的差错控制机制，发送方可能会重新发送这些被丢弃的分组的副本。如果拥塞持续下去，在某一时刻这种状况将会达到极点，此时系统崩溃，没有任何数据报能被交付。

无连接网络中的拥塞控制

在无连接网络中有多种方法可以控制拥塞的形成。其中一种方案称为信令。反向信令就是在运动方向与拥塞方向相反的数据报中设置一个比特，以通知发送方拥塞正在形成，发送方应当放慢发送分组的速度。在这种情况下，这个比特可以被设置在对一个分组的响

应或者确认分组中。如果网络层没有使用反馈（确认），但是上层使用了反馈，那么就可以使用前向信令方式。它是在运动方向与拥塞方向一致的分组中设置一个比特，用于向该分组的接收方发出拥塞警告。然后，接收方就可以通知上层协议，再由上层协议来想办法通知源点。在因特网的网络层中既没有使用前向信令，也没有使用反向信令。

无连接网络的拥塞控制还可以通过一个扼流分组（choke packet）来实现，它是在遇到拥塞时由路由器向发送方发送的一个特殊的分组。事实上这正是因特网的网络层所实施的拥塞控制机制。因特网的网络层使用了一个辅助协议，称为 ICMP，我们将在第 9 章中讨论。当一个路由器拥塞了，它就会向源点发送一个 ICMP 分组使其放慢速度。

另一种改善拥塞的方法是将分组按其在整个报文中的重要程度划分等级。例如，可以利用分组首部中的一个字段来定义这个数据报的地位是比较重要的，还是不重要的。当一个路由器拥塞并且需要丢弃一些分组时，标记为不重要的分组将被丢弃。例如，如果有一个报文的内容是一张图片，它可能会被分割为多个分组，那么表示图片边边角角的那些分组的重要性就比不过图片中心位置的分组。如果路由器拥塞了，就会丢弃不重要的分组，同时也不会严重影响整个图片的质量。我们将在第 25 章讨论多媒体通信时再来研究这个问题。

面向连接网络中的拥塞控制

有时候面向连接网络中的拥塞控制要比无连接网络中的拥塞控制更简单。一个最简单的方法就是当某区域发生拥塞后就建立一条额外的虚电路。然而，这样做可能会给某些路由器带来更多的问题。一种更好的解决方法是在建链阶段就进行提前协商。发送方和接收方可以在建立虚电路之时对通信量的级别达成一致意见。这个通信量级别可以由允许建立该虚电路的路由器来指定。换言之，路由器先要检查现有的通信量，并与自己的最大通信相比较，然后才允许建立新的虚电路。

服务质量

随着因特网允许像多媒体通信（特别是音频和视频的实时通信）这样的新应用程序加入进来，通信的服务质量（quality of service, QoS）也变得越来越重要。因特网正全力以赴地提供更好的服务质量以支持此类应用。但是，为了使网络层保持原封不动，对此类应用的支持大多是在上层实现的。由于在使用多媒体通信时会更显得 QoS 很重要，所以我们将在第 25 章讨论多媒体时再来研究这个问题。

路由选择

网络层的一个非常重要的话题就是路由选择（routing），也就是路由器如何建立自己的路由表，以便在无连接网络中帮助数据报的转发，或者在面向连接网络的建链阶段，帮助建立一条虚电路。这个任务是由路由选择协议完成的，它帮助主机和路由器建立自己的路由表，并维护和更新这些路由表。这是一些独立的协议，它们有时会使用网络层的服务，而有时则使用某些运输层协议的服务，以助网络层一臂之力。路由选择协议可以划分为两大类：单播的和多播的。我们将用第 11 章专门讨论单播路由选择，并用第 12 章来专门讨论多播路由选择。在进入第 11 章和第 12 章对这些协议进行讨论之前，我们一直会假设路由器已经建好了它们的路由选择协议。

安全性

另一个与网络层的通信有关的话题是安全性。在最初设计因特网时并没有考虑到安全

性问题，因为它只是被高等院校中的少数用户用来进行研究活动的，其他人根本无法访问因特网。网络层在设计时不提供对安全性的支持。但是时至今日，安全性已经成了人们重点关注的问题。为了在无连接的网络层提供安全性，需要用另外一些虚拟层来把无连接的服务转变为面向连接的服务。这个虚拟层称为 IPSec，在我们讨论了加密的基本原理和安全性问题（第 29 章）之后，会在第 30 章来讨论 IPSec。

4.6 进一步阅读

要更详细地了解本章所讨论的内容，我们推荐以下一些书籍：[Ste 94]，[Tan 03]，[Com 06]，[Gar & Vid 04]以及[Kur & Ros 08]。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

4.7 重要术语

扼流分组	流量控制
电路交换	分组交换
拥塞控制	服务质量（QoS）
无连接服务	路由选择
面向连接服务	建链阶段
数据传送阶段	交换
差错控制	拆链阶段

4.8 本章小结

- 从概念上看，我们可以认为全球因特网是一个暗箱网络。但是因特网并不是单一的网络，它是由许多网络（或者说链路）通过连接设备互相连接在一起形成的。
- 在因特网中，像路由器这样的连接设备起到了交换机的作用。人们在组网时通常会使用两种类型的交换方式：电路交换和分组交换。
- 网络层被设计为一个分组交换网。分组交换网可以提供无连接的服务或者是面向连接的服务。当网络层提供无连接服务时，每个穿行在因特网中的分组都是一个独立的实体，属于同一个报文的分组之间没有任何联系。在面向连接的服务中，属于同一个报文的所有分组之间存在一条虚连接。
- 无连接服务根据分组的目的地址将其转发到下一跳。面向连接的服务根据分组中的标号将其转发到下一跳。
- 在面向连接的网络中，通信过程分为三个阶段：建链、数据传送和拆链。在连接建立后，发送方和接收方之间就建立了一条虚电路，属于同一个报文的所有分组都要通过这条虚电路发送。

- 我们讨论了在因特网的网络层中已经存在的服务，包括寻址、由源计算机提供的服务、由目的计算机提供的服务以及由各路由器提供的服务。
- 我们也讨论了一些与网络层相关的问题，它们是一些通常在讨论网络层时会被讨论的服务，但它们或者只是在网络层部分地实现了，或者就根本没有实现。其中有一些服务，如路由选择和安全性，在因特网上是通过其他协议提供的。

4.9 实践安排

4.9.1 习题

1. 试给出无连接服务的优点和缺点。
2. 试给出面向连接服务的优点和缺点。
3. 在面向连接的服务中，如果一个标号的长度是 n 位，那么同时可以建立几条虚连接？
4. 假设一台终点计算机正在接收来自多台源计算机的报文。它是如何保证来自某个源的分片不会与来自其他源的分片相混淆？
5. 假设一台终点计算机接收来自一个源点的多个分组。它是如何保证属于某个数据报的分片不会与属于另一个数据报的分片相混淆？
6. 你认为为什么图 4.7 中的分组既要有地址又要有标号？
7. 试比较在无连接和面向连接服务中的时延。如果报文比较大，那么哪一种服务产生的时延更小？如果报文比较小，又是哪一种服务产生的时延更小？
8. 在图 4.13 中，为什么分片是最后一项服务？
9. 讨论为什么在每个路由器上都需要有分片服务？
10. 讨论为什么我们要在最后的终点进行重装，而不是在每个路由器重装？
11. 在图 4.15 中，为什么我们需要设置一个定时器，并在定时器超时后就销毁所有分片？你会用什么样的标准来选择这种定时器的时长？

第 5 章 IPv4 地址

在 网络层，我们需要对因特网上的每一个设备进行唯一标识，这样所有的设备之间才能实现全球通信。本章要讨论的是与当前主流的 IPv4 协议相对应的 IPv4 编址机制。因为人们确信 IPv6 总有一天会超越目前的 IPv4 协议，所以我们也需要对 IPv6 编址机制有所了解。我们将在第 26 章到第 28 章的内容中讨论 IPv6 协议及其编址机制。

目标

本章有以下几个目标：

- 引入地址空间的总体概念，重点介绍 IPv4 的地址空间。
- 讨论分类编址体系结构，介绍此模型中的各个类，以及每个类可用的地址块。
- 讨论多级编址的思想以及它是如何在分类编址中实现的。
- 解释分类结构中的子网划分和构造超网，并说明它们如何用来克服分类编址的缺点。
- 讨论无分类编址，这是为了解决在分类编址中遇到的问题（如地址耗尽）而设计的一种新的体系结构。
- 说明从分类编址中借用的某些思想，例如子网划分，如何在无分类编址中很容易地实现。
- 讨论一些特殊的地址块以及各个地址块中存在的一些特殊地址。
- 讨论 NAT 技术，并说明它是如何被用来缓解 IPv4 中地址短缺矛盾的。

5.1 引言

TCP/IP 协议族中，用于在 IP 层识别每一个连接到因特网设备的标识符称为因特网地址或 IP 地址（IP address）。IPv4 地址是一个 32 位的地址，它唯一地且全球地定义了一台主机或路由器与因特网之间的一个连接，也就是说 IP 地址是该接口的地址。

IPv4 地址的长度是 32 位。

IPv4 地址是唯一的。它们的唯一性表现在每个地址仅能定义一个到因特网的连接。因特网上的两个设备不能同时具有相同的地址。如果一个设备有两个到因特网的连接，那么这个设备就有两个 IPv4 地址。IPv4 地址的全球性表现在它是任何希望连接到因特网的主机都必须采纳的地址系统。

IP 地址是唯一的且全球统一的。

5.1.1 地址空间

像 IPv4 这种定义了地址的协议都有一个地址空间 (address space)。地址空间就是协议所使用的地址的总数。如果一个协议用 b 位来定义地址，那么它的地址空间就是 2^b ，这是因为每一位都可以有两个不同的取值 (1 或 0)。IPv4 使用 32 位地址，这就表示它的地址空间是 2^{32} ，或 4 294 967 296（超过 40 亿）。从理论上讲，如果没有其他限制，可以有超过 40 亿个设备连接到因特网。

IPv4 的地址空间是 2^{32} 或 4294967296。

5.1.2 记法

有三种常用的记法来表示 IPv4 地址：二进制记法（基 2）、点分十进制记法（基 256）和十六进制记法（基 16）。不过目前最流行的还是基 256 的点分十进制记法。这几种计数法在附录 B 中都有定义。我们还在该附录中说明了如何把一个数从某种计数法转换成另一种计数法。我们建议在继续阅读本章内容之前，最好先复习一下该附录。

基 2、基 16 和基 256 中的数字在附录 B 中讨论。

二进制记法：基 2

在二进制记法 (binary notation) 中，IPv4 地址表现为 32 位。为了使这个地址有更好的可读性，通常在每个八位组 (8 位) 之间要加上一个或多个空格。一个八位组也称为一个字节。因此经常会听到人们称 IPv4 地址为 32 位地址，或 4 个八位组的地址，或 4 字节的地址。下面是 IPv4 地址二进制记法的一个例子

01110101 10010101 00011101 11101010

点分十进制记法：基 256

为了使 IPv4 地址更简洁，更易于阅读，人们通常将其写成用小数点把各字节分隔开的十进制数字格式。这种格式称为点分十进制记法 (dotted-decimal notation)。图 5.1 所示为一个点分十进制记法的 IPv4 地址。请注意，因为每个字节 (八位组) 只有 8 位，所以点分十进制记法中的数值一定在 0~255 之间。

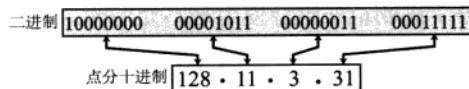


图 5.1 点分十进制记法

例 5.1

试把下列 IPv4 地址从二进制记法转换为点分十进制记法。

- a. 10000001 00001011 00001011 11101111

- b. 11000001 10000011 00011011 11111111
- c. 11100111 11011011 10001011 01101111
- d. 11111001 10011011 11111011 00001111

解

我们把每 8 位一组转换成等值的十进制数（见附录 B），并增加分隔的点，得到：

- a. 129.11.11.239
- b. 193.131.27.255
- c. 231.219.139.111
- d. 249.155.251.15

例 5.2

试把下列 IPv4 地址从点分十进制记法转换为二进制记法。

- a. 111.56.45.78
- b. 221.34.7.82
- c. 241.8.56.12
- d. 75.45.34.78

解

我们把每个十进制数转换成等值的二进制数（见附录 B）：

- a. 01101111 00111000 00101101 01001110
- b. 11011101 00100010 00000111 01010010
- c. 11110001 00001000 00111000 00001100
- d. 01001011 00101101 00100010 01001110

例 5.3

下列 IPv4 地址是否有错误，请指出：

- a. 111.56.045.78
- b. 221.34.7.8.20
- c. 75.45.301.14
- d. 11100010.23.14.67

解

- a. 在点分十进制记法中不应当有以 0 开头的数（045）。
- b. IPv4 地址不能超过 4 个字节。
- c. 每个字节必须小于或等于 255，而 301 超过了这个范围。
- d. 二进制记法和点分十进制记法混合使用是不允许的。

十六进制记法：基 16

有时我们也会遇到十六进制记法（hexadecimal notation）的 IPv4 地址。一个十六进制数字等效于一个 4 位的二进制数字。这就是说，一个 32 位的地址要用 8 个十六进制数字来表示。这种记法常用在网络编程中。

例 5.4

试将下列 IPv4 地址从二进制记法转换为十六进制记法。

- a. 10000001 00001011 00001011 11101111

b. 11000001 10000011 00011011 11111111

解

我们把每4位一组转换为等值的十六进制数（见附录B）。需要注意的是，十六进制记法中通常不需要加入空格或点，但要在开头加上0X（或0x），或在其尾部附加下标16，以表示这个数是十六进制的。

a. 0X810B0BEF 或 810B0BEF₁₆

b. 0XC1831BFF 或 C1831BFF₁₆

5.1.3 地址段

我们经常需要处理一段范围内的地址，而不是一个地址。有时候，在给定了一段地址的首地址和末地址的条件下，我们需要求出这个地址段共有多少个地址，有时候，我们又要在给定了一段地址的首地址及其地址数量的条件下算出该地址段的末地址是什么。此时我们可以使用相应计数法（基2、基256或基16）的减法或加法运算。还有一种方法，我们可以先把地址转换成十进制数（基10），然后再进行相应的加减法运算。

例 5.5

假设一段地址的首地址为146.102.29.0，末地址为146.102.32.255，求这个地址段的地址数。

解

我们可以使用基256计数法的减法运算，让末地址减去首地址（参见附录B）。在基256计数法下得到的结果是0.0.3.255。为了求出这个地址段的地址数（以十进制数表示），我们将这个数转换为基10计数法，结果再加上1：

$$\text{地址的数目} = (0 \times 256^3 + 0 \times 256^2 + 3 \times 256^1 + 255 \times 256^0) + 1 = 1024$$

这个想法要记住啊！！

例 5.6

某地址段的首地址为14.11.45.96。假设这个地址段的地址数为32个，那么它的末地址是什么？

解

我们把地址数减去1后再转换为基256计数法表示，也就是0.0.0.31。然后用这个值加上首地址就得到末地址，此加法为基256的加法。

$$\text{末地址} = (14.11.45.95 + 0.0.0.31)_{256} = 14.11.45.127$$

5.1.4 运算

我们经常需要对一个二进制或点分十进制的32位数执行某种运算，这个数所代表的要么是IPv4地址，要么是与IPv4地址相关的某些实体（如掩码，稍后再做讨论）。这一节我们要介绍本章稍后会用到的三种运算：非(NOT)、与(AND)、或(OR)。

位非运算

位非(bitwise NOT)运算是一种单操作数运算，它的输入只有一个。当我们对某个数做非运算时，通常是指对这个数求反。而在二进制下对一个32位的数进行非运算就是反转

它的每一位。每个 1 都变成 0，而每个 0 都变成 1。图 5.2 描绘了这种非运算。

对于一个 32 位的二进制数我们可以直接使用非运算，但是如果这个数被表示成四个字节的点分十进制记法时，我们需要使用一种快捷算法：分别用 255 减去每一个字节。

例 5.7

从下面可以看出我们如何对一个 32 位的二进制数执行非运算。

原数: 00010001 01111001 00001110 00100011

反码: 11101110 10000110 11110001 11011100

我们可以用快捷算法对该数的点分十进制形式执行同样的非运算。

原数: 17 . 121 . 14 . 35

反码: 238 . 134 . 241 . 220

位与运算

位与 (bitwise AND) 运算是一种双操作数运算，它的输入有两个。与运算就是将两个输入的每一个对应位分别进行比较，并选择出较小的（如果相等，则任选一个）。图 5.3 描绘了位与运算。

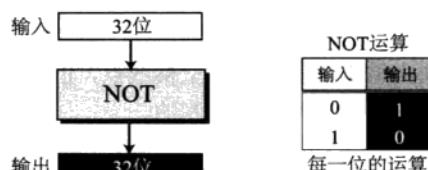


图 5.2 位非运算

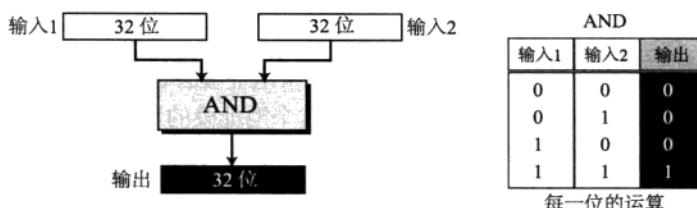


图 5.3 位与运算

对于两个用二进制表示的 32 位数我们可以直接进行与运算，但如果这两个数被表示为点分十进制记法时，我们需要使用以下两种快捷算法：

1. 如果两个字节中至少有一个是 0 或 255，那么就选择较小的那个字节（如果相等则任选一个）。
2. 如果两个字节都不等于 0 或 255，我们可以把这两个字节分别写成具有八个项的多项式之和，且每一项都是 2 的乘方，然后我们选择各对应的项中较小的（如果相等则任选一个），最后再把选出的项累加起来就得到了结果。

例 5.8

从下面可以看出我们如何对两个 32 位的二进制数执行与运算。

第一个数: 00010001 01111001 00001110 00100011

第二个数: 11111111 11111111 10001100 00000000

结果: 00010001 01111001 00001100 00000000

我们可以用快捷算法对点分十进制的形式执行相同的与运算。

第一个数: 17 . 121 . 14 . 35

第二个数: 255 . 255 . 140 . 0
 结果: 17 . 121 . 12 . 0

对于第一、第二和第四个字节我们采用了第一种快捷算法, 而对第三个字节则采用了第二种快捷算法。我们把 14 和 140 分别写成多项式, 并选择每一对相应项中较小的项, 最后得到结果如下:

$$\begin{array}{cccccccccc}
 \text{乘方} & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 \text{字节 (14)} & 0 + 0 + 0 + 0 + 8 + 4 + 2 + 0 \\
 \text{字节 (140)} & 128 + 0 + 0 + 0 + 8 + 4 + 0 + 0 \\
 \text{结果 (12)} & 0 + 0 + 0 + 0 + 8 + 4 + 0 + 0
 \end{array}$$

位或运算

位或 (bitwise OR) 运算也是一种双操作数运算, 它的输入有两个。或运算就是分别比较两个数相对应的每一位, 并选择较大的 (如果相等, 则任选一个)。图 5.4 描绘了位或运算。

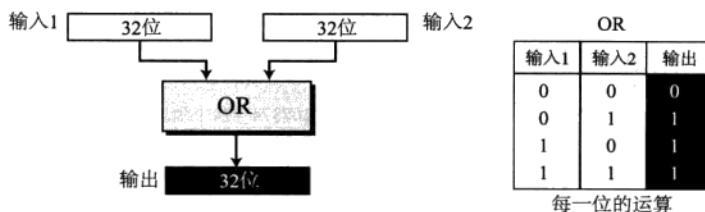


图 5.4 位或运算

虽然对于两个 32 位的二进制表示法我们可以直接进行或运算, 但如果这两个数被表示成点分十进制记法时, 我们需要使用以下两种快捷算法:

1. 如果两个字节中至少有一个字节是 0 或 255, 那么或运算就选择较大的那个字节 (如果相等则任选一个)。
2. 如果两个字节都不等于 0 或 255, 我们可以把每个字节写成八个项的多项式之和, 其中每项都是 2 的乘方。然后我们选择各个对应项中较大的 (如果相等则任选一个), 最后再把这些项加起来就得到或运算的结果。

例 5.9

从下面可以看出我们如何对两个 32 位的二进制数执行或运算。

第一个数: 00010001 01111001 00001110 00100011
 第二个数: 11111111 11111111 10001100 00000000
 结果: 11111111 11111111 10001110 00100011

我们可以用快捷算法对点分十进制的形式执行同样的或运算。

第一个数: 17 . 121 . 14 . 35
 第二个数: 255 . 255 . 140 . 0
 结果: 255 . 255 . 142 . 35

我们对第一、第二和第四个字节采用了第一种快捷算法, 而对第三个字节采用了第二种快捷算法。

5.2 分类编址

在数十年前 IP 地址诞生之初就使用了分类的概念。这种体系结构称为 **分类编址** (classful addressing)。到了 20 世纪 90 年代中期，一种称为 **无分类编址** (classless addressing) 的新体系出现，并且比原有体系结构更具优势。在这一小节中我们要先介绍分类编址，因为这样做可以为更好地理解无分类编址打下基础，并了解为什么要迁移到新的体系结构的理由。无分类编址将在下一节进行讨论。

5.2.1 分类

在分类编址时，IP 地址空间被分为五类 (class): A、B、C、D 和 E。每一类占用整个地址空间中的一部分。图 5.5 给出了每一类地址在地址空间的占用情况。

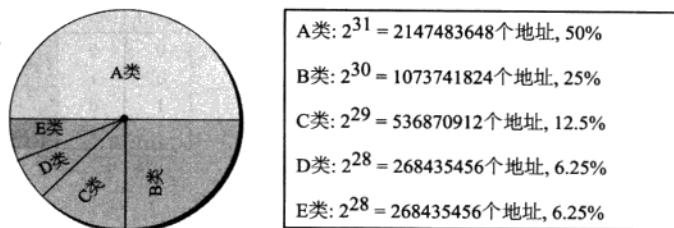


图 5.5 地址空间的占用情况

在分类编址中，地址空间共分为五类：A、B、C、D 和 E。

辨认类别

当给出了一个用二进制记法或点分十进制记法表示的地址时，我们能够找出这个地址的类别。以二进制记法表示时，该地址的前几位就可以立即告诉我们它的类别，而以点分十进制记法表示时，通过第一个字节的值就可以确定地址的类别（参见图 5.6）。

	第一个 八位组	第二个 八位组	第三个 八位组	第四个 八位组		第一字节	第二字节	第三字节	第四字节
A类	0.....				二进制记法	0~127			
B类	10.....					128~191			
C类	110....					192~223			
D类	1110....					224~239			
E类	1111....					240~255			
					点分十进制记法				

图 5.6 找出地址的类别

应当注意，某些特殊地址落入 A 类或 E 类的范围。我们要强调指出，这些特殊地址是分类的例外情况，我们将在本章的后面讨论。

计算机以二进制的形式来存储IPv4地址。在这种情况下，很方便就可以写出一种算法，通过一个连续检查过程来找出该地址的类别，如图5.7所示。

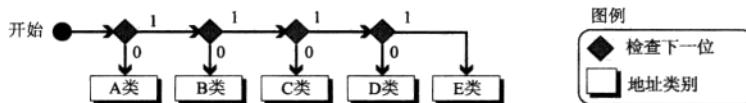


图5.7 通过连续检查来找出地址的类别

例5.10

求下列每个地址的类别。

- 00000001 00001011 00001011 11101111
- 11000001 10000011 00011011 11111111
- 10100111 11011011 10001011 01101111
- 11110011 10011011 11111011 00001111

解

参考图5.7所示的过程。

- 第一位是0。这是A类地址。
- 前两位是1，第三位是0。这是C类地址。
- 第一位是1，第二位是0。这是B类地址。
- 前四位是1。这是E类地址。

例5.11

求下列每个地址的类别。

- 227.12.14.87
- 193.14.56.22
- 14.23.120.8
- 252.5.15.111

解

- 第一个字节是227（在224~239之间），是D类。
- 第一个字节是193（在192~223之间），是C类。
- 第一个字节是14（在0~127之间），是A类。
- 第一个字节是252（在240~255之间），是E类。

网络标识和主机标识

在分类编址中，对于A、B、C类地址来说，IP地址都可划分为网络标识（net-id）和主机标识（host-id）两个部分。这两个部分的长度随地址类别的不同而变化。图5.8给出了网络标识和主机标识所占的字节。应当注意的是，D类和E类地址不划分网络标识和主机标识，我们稍后再讨论其原因。

对于A类地址，前一个字节用于定义网络标识，后三个字节用于定义主机标识；对于B类地址，前两个字节用于定义网络标识，后两个字节用于定义主机标识；对于C类地址，前三个字节用于定义网络标识，后一个字节用于定义主机标识。

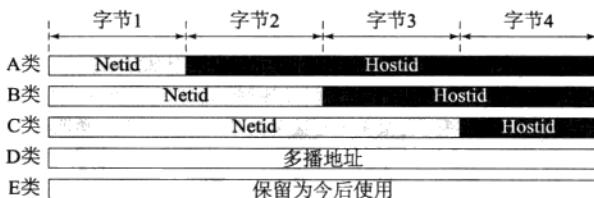


图 5.8 网络标识和主机标识

5.2.2 地址类和地址块

分类编址存在的一个问题就是每一类地址都被划分为固定数目的地址块，而每一个地址块的大小也都是固定的。让我们先来了解一下各类地址。

A类

在 A 类地址中，只有一个字节用来定义网络标识，而且它的最左一位应当始终为 0，只有后七位可变，由此可得 A 类地址中地址块的数目。A 类地址共有 $2^7 = 128$ 个地址块，可以被指派给 128 个机构组织（实际数目还要少一些，因为其中有几个特殊的地址块是保留的）。但是，这类地址的每个地址块都包含有 16777216 个地址，这表明要使用所有这些地址的机构应该是个非常庞大的机构。在这类地址中有大量的地址被浪费了。图 5.9 所示为 A 类地址的地址块。

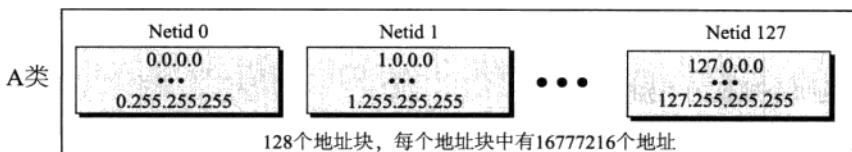


图 5.9 A 类地址的地址块

数以百万计的 A 类地址都被浪费了。

B类

在 B 类地址中，用了两个字节来定义网络标识，而且它的最左边的两位应当为 10（固定），剩下的 14 位可变，由此可得知 B 类地址中地址块的数目。B 类地址共有 $2^{14} = 16384$ 个地址块，可指派给 16384 个机构组织（实际数目还要少一些，因为有几个特殊的地址块是保留的）。但是，这类地址的每个地址块都包含有 65536 个地址。需要使用这么多地址的机构也不多见。在此类地址中也有很多地址被浪费了。图 5.10 所示为 B 类地址的地址块。

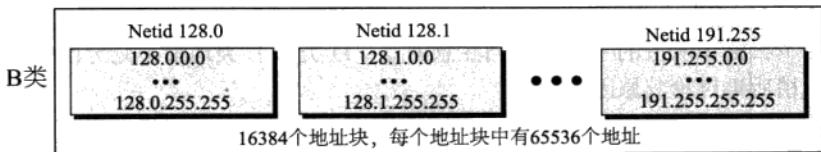


图 5.10 B 类地址的地址块

许多B类地址被浪费了。

C类

在C类地址中，用了三个字节来定义网络标识，而且它的最左边的三位应当为110（固定），剩下的21位可变，由此可得知C类地址中地址块的数目。C类地址共划分为 $2^{21} = 2097152$ 个地址块，每个地址块有256个地址，可指派给2097152个机构组织（实际数目还要少一些，因为有几个特殊的地址块是保留的）。一个C类地址块只有256个地址，小到一个C类地址块就能满足的机构也不多见。图5.11所示为C类地址的地址块。

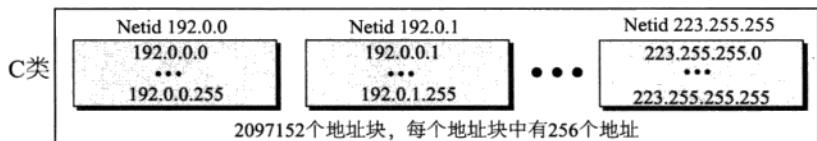


图5.11 C类地址的地址块

C类地址中的地址数对大多数机构来说是不够用的。

D类

D类地址只有一个地址块。它用来进行多播（我们将在以后讨论）。这类地址中的每一个地址都被用来定义因特网上的一组主机。当某个组被指派了一个D类地址时，该组中的每一个成员主机都会在正常地址（单播地址）的基础上增加一个多播地址。图5.12描绘了这种地址块。

D类地址用来进行多播；这类地址只有一个地址块。

E类

E类地址只有一个地址块。它被设计为保留地址，如图5.13所示。

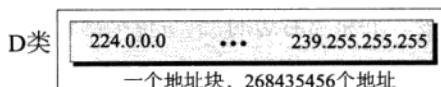


图5.12 D类地址的唯一地址块

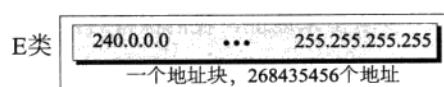


图5.13 E类地址的唯一地址块

E类地址仅有的一个地址块为将来使用而保留。

5.2.3 两级编址

总的说来，IPv4地址的作用就是为因特网中的一个分组指明其终点（在网络层）。在设计分类地址时，人们认为整个因特网可划分为很多网络，而每个网络连接多个主机。换言之，因特网被视为一个由网络组成的网络。通常是由希望连接到因特网的组织来创建各个网络。因特网权威机构则向该组织指派一个地址块（A类、B类或C类的）。

分类编址时，指派给一个组织的地址段是A、B或C类地址的一个地址块。

分类编址时，同一个网络中的所有的地址都属于同一个地址块，且每个地址都包含两部分：网络标识和主机标识。网络标识指明了网络，而主机标识则指明了连接到该网络上的一台特定的主机。图 5.14 描绘了分类编址中的一个 IPv4 地址。如果某一类地址用了 n 位来定义网络，那么就有 $32 - n$ 位定义了主机。不过这个 n 的值取决于地址块所属的类。 n 的值可以是 8、16 或 24，它们分别对应 A 类、B 类和 C 类地址。

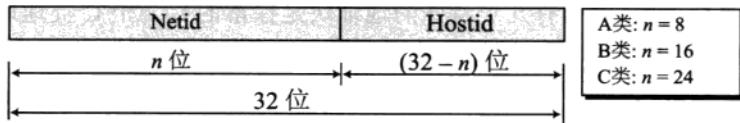


图 5.14 分类地址中的两级编址

例 5.12

在其他通信系统中也可以看到两级编址的情况。例如，美国国内的一个电话系统可以被看成是由两部分组成的：区号和本地号。区号指明了区域，而本地号则指明了该区域中的某个具体的电话用户。

(626)3581301

它的区号是 626，与网络标识相对应；本地号 3581301，与主机标识相对应。

提取一个地址块中的信息

一个地址块就是一段地址。给定该地址块中的任意一个地址，我们通常希望了解的是该地址块的三个信息：地址数、首地址和末地址。在提取这三个信息之前，我们首先要要知道这个地址属于哪一类，这一点我们在前面已经知道该如何做。在找出了地址的类别之后，我们就知道了 n 的值以及网络标识的长度(以位为单位)。现在我们可以求出这三个信息了，如图 5.15 所示。

1. 该地址块的地址数 N 可以用 $N = 2^{32-n}$ 求出。
2. 要求出首地址，我们保持最左边的 n 位不变，并将靠右边的 $32-n$ 位全部置 0。
3. 要求出末地址，我们保持最左边的 n 位不变，并将靠右边的 $32-n$ 位全部置 1。

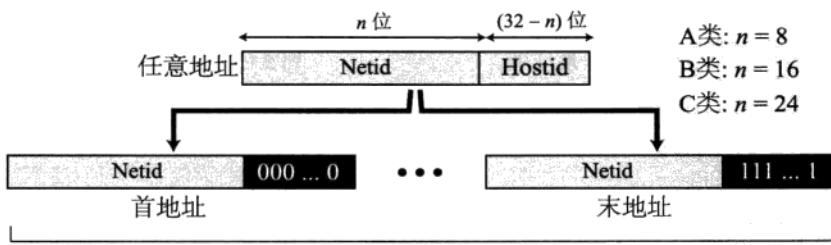


图 5.15 分类地址中的信息提取

例 5.13

给出某地址块中的一个地址为 73.22.17.25。求该地址块的地址数及其首地址和末地址。

解

因为 73 在 0~127 之间，所以这个地址是 A 类地址。A 类地址的 n 等于 8。图 5.16 描绘了使用此类地址的一种可能的网络配置。请注意，它的网络地址在一个斜杠后面给出了 n 的值。虽然在分类地址中这种写法并不常用，但是它有助于我们养成无分类地址（下一节）书写的习惯。

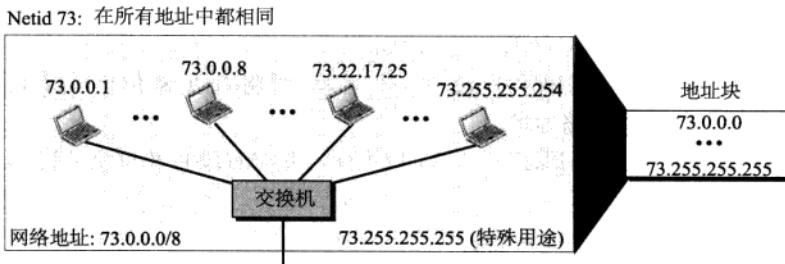


图 5.16 例 5.13 的解

- 该地址块的地址数为 $N = 2^{32-n} = 2^{24} = 16777216$ 。
- 为了求出首地址，我们保持左边 8 位不变，并将右边 24 位全部置 0。因此首地址是 73.0.0.0/8，其中 8 就是 n 的值。首地址被称为网络地址，它不会指派给任何主机，而是用来定义这个网络。
- 为了求出末地址，我们保持左边 8 位不变，并将右边的 24 位全部置 1。末地址是 73.255.255.255。这个末地址通常有特殊的用途，本章稍候将讨论这一用途。

例 5.14

给出某地址块中的一个地址为 180.8.17.9。求该地址块的地址数及其首地址和末地址。

解

因为 180 在 128~191 之间，所以这个地址是 B 类地址。B 类地址的 n 等于 16。图 5.17 描绘了使用此类地址的一种可能的网络配置。

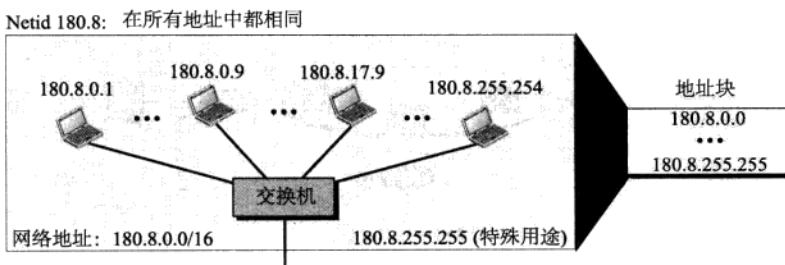


图 5.17 例 5.14 的解

- 该地址块中的地址数为 $N = 2^{32-n} = 2^{16} = 65536$ 。
- 为了求出首地址，我们保持左边 16 位不变，并将右边 16 位全部置 0。首地址（网络地址）是 180.8.0.0/16，其中 16 是 n 的值。
- 为了求出末地址，我们保持左边 16 位不变，并将右边的 16 位全部置 1。末地址是

180.8.255.255。

例 5.15

给出某地址块中的一个地址为 200.11.8.45。求该地址块的地址数及其首地址和末地址。

解

因为 200 在 192~223 之间，所以这个地址是 C 类地址。C 类地址的 n 等于 24。图 5.18 描绘了使用此类地址的一种可能的网络配置。

1. 该地址块的地址数为 $N = 2^{32-n} = 2^8 = 256$ 。
2. 为了求出首地址，我们保持左边 24 位不变，并将右边 8 位全部置 0。首地址是 200.11.8.0/24。首地址称为网络地址。
3. 为了求出末地址，我们保持左边 24 位不变，并将右边的 8 位全部置 1。末地址是 200.11.8.255。

Netid 200.11.8: 在所有地址中都相同

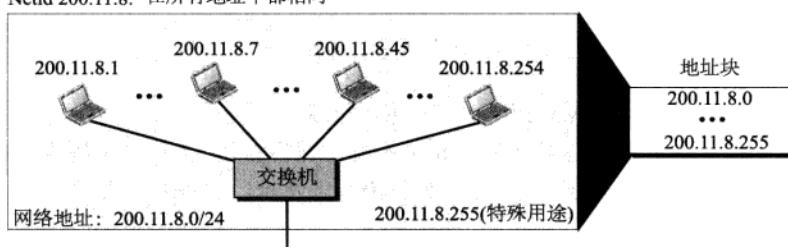


图 5.18 例 5.15 的解

5.2.4 一个例子

图 5.19 描绘了一个假想的互联网的一部分，它由三个网络组成。

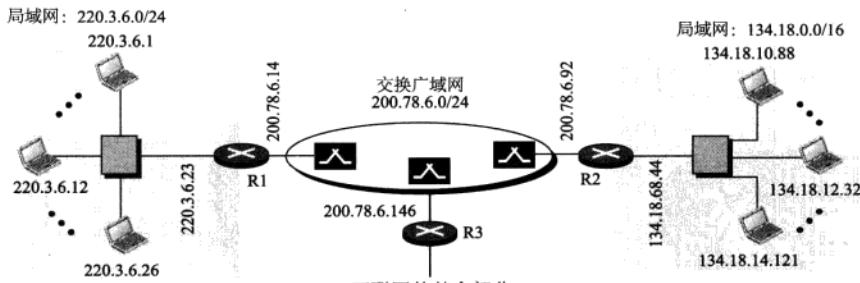


图 5.19 互联网的例子

我们有：

1. 一个局域网，其网络地址为 220.3.6.0 (C 类)。
2. 一个局域网，其网络地址为 134.18.0.0 (B 类)。
3. 一个交换广域网 (C 类)，如帧中继或 ATM，它可以连接很多个路由器。在这里只描绘了三个路由器，一个连接到左边的局域网，另一个连接到右边的局域网，还有一个连

接到广域网，该广域网通往互联网其他部分。

网络地址

从前面三个例子可以看出，给定任意一个地址就可以求出有关该地址块的所有信息。其中首地址，也就是**网络地址**（network address）尤其重要，因为它被用来为分组选择路由，使其能够到达目的网络。此刻让我们假设一个互联网由 m 个网络组成，且一个路由器有 m 个接口。当来自任意源主机的一个分组到达路由器时，路由器就要知道应当将这个分组发往哪个网络，也就是说该路由器需要知道这个分组应当从哪个接口发送出去。当分组到达目的网络后会采用另一种策略抵达它的目的主机，我们将在后面的章节中再讨论。图 5.20 描绘了这一思想。在得知了网络地址之后，路由器咨询它的路由表以找出相应的接口将分组转发出去。网络地址实际上就是网络的标识，所有网络都要通过自己的网络地址来识别。

网络地址就是对一个网络的标识。

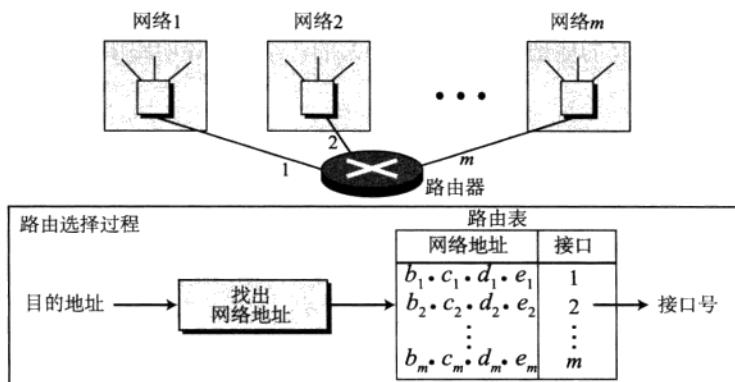


图 5.20 网络地址

网络掩码

前面我们所讨论的如何提取网络地址的方法都是为了从概念上来理解这个问题。因特网上的路由器通常会用一种算法从分组的目的地址中提取其网络地址。要做到这一点，我们需要有一个**网络掩码**（network mask）或者叫**默认掩码**（default mask）。分类地址中的**网络掩码**（network mask）或者叫**默认掩码**（default mask）是一个 32 位的数，这个数的左边 n 位全部置 1，右边 $32-n$ 位全部置 0。因为在分类地址中，不同的类别有不同的 n 值，所以我们一共有三种网络掩码，如图 5.21 所示。

A类掩码	8位		24位		255.0.0.0
	11111111	00000000	00000000	00000000	
B类掩码	16位		16位		255.255.0.0
	11111111	11111111	00000000	00000000	
C类掩码	24位		8位		255.255.255.0
	11111111	11111111	11111111	00000000	

图 5.21 网络掩码

要从一个分组的目的地址中提取其网络地址，路由器使用了与运算（如前所述）。在用默认掩码对目的地址（或地址块中的任一地址）进行与运算后，得到的结果就是网络地址（图 5.22）。路由器是用二进制（或十六进制表示法）的地址及掩码进行与运算的，不过我们在举例时用的是前面讨论的快捷算法对点分十进制记法的掩码进行与运算。这个默认掩码也可用来求该地址块中的地址数及其末地址，我们将在无分类编址中讨论此类应用。

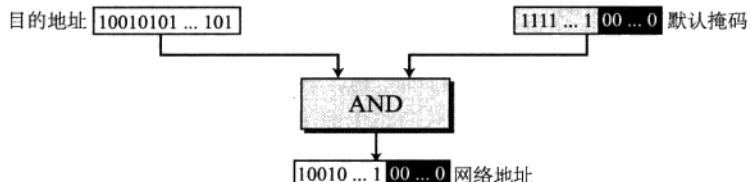


图 5.22 使用默认掩码求网络地址

例 5.16

路由器接收到一个目的地址为 201.24.67.32 的分组。请说明路由器如何才能找出这个分组的目的网络地址。

解

我们假设路由器首先要找出目的地址的类别，然后用相应的默认掩码对该地址进行处理。但是有一点我们必须清楚，实际上路由器使用的是另外一种策略，这将在下一章中讨论。因为地址的类别是 B，我们假设对应 B 类地址的默认掩码是 255.255.0.0，路由器用这个掩码求得目的网络地址。

目的地址	→	201	.	24	.	67	.	32
默认掩码	→	255	.	255	.	0	.	0
网络地址	→	201	.	24	.	0	.	0

我们使用了上一节介绍过的快捷算法，网络地址正如我们所预期的是 201.24.0.0。

5.2.5 三级编址：子网划分

如我们前面所讨论的，IP 地址最初的设计为两级编址。也就是说，要想抵达因特网中的一台主机，首先必须抵达其网络，然后才能找到主机。很快人们就发现像这样的两级编址还不够，原因有以下两点。首先，一个被授权使用 A 类或 B 类地址块的组织，出于安全性和管理方便的考虑，有必要将自己的大网络进一步划分为若干个子网。其次，因为 A 类和 B 类地址块已近耗尽，而 C 类地址块又太小，无法满足大多数组织的需要，所以已经拥有 A 类或 B 类地址块使用权的组织可以将其地址块划分为多个较小的子地址块，并与其他组织一起分享。将一个地址块分割为若干个较小地址块的思想称为子网划分。在子网划分（subnetting）时，一个网络被划分为若干个较小的子网络（子网），其中每个子网都有自己的子网地址。

例 5.17

如果我们把电话号码中的本地号码看成是由交换局号和用户号组成的，那么在电话系统中存在着三级编址：

答案错误：这是一个C类地址，所以答案应该为 201.24.67.0

(626) 358 - 1301

其中 626 是区号，358 指的是交换局，而 1301 则指的是用户连接。

例 5.18

图 5.23 所示为子网划分之前的一个网络，它使用的是 B 类地址。我们有大约 2^{16} 台主机，却只有一个网络。整个网络仅通过一条连接与因特网中的一个路由器进行连接。请注意，在网络地址中出现的/16 表示的是网络标识的长度（B 类）。

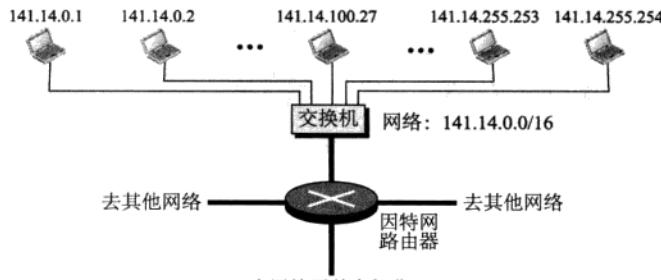


图 5.23 例 5.18

例 5.19

图 5.24 所示为图 5.23 中的网络在子网划分之后的情况。整个网络仍然是通过相同的路由器连接到因特网。但是，网络使用了一个专用路由器将它划分为四个子网。其他因特网上的设备看到的只是一个网络，但这个网络内部实际上是由四个子网组成的。此时每个子网都可以拥有 2^{14} 台主机。这样的一个网络可以属于一所具有四个独立校区（或大楼）的大学。在子网划分后，每个校区都有自己的子网，但是对其余因特网来说，整个大学是一个网络。请注意，/16 和 /18 分别表示的是网络标识长度和子网标识长度。

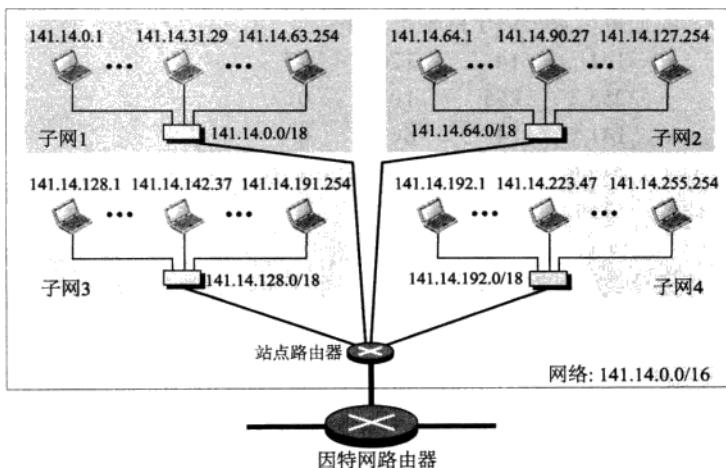


图 5.24 例 5.19

子网掩码

我们在前面讨论过网络掩码（默认掩码）。当一个网络没有子网划分时，使用的是网络掩码。而当我们把一个网络划分为若干个子网后，就要为每个子网建立一个子网掩码。一个子网由子网标识和主机标识两部分组成，如图 5.25 所示。

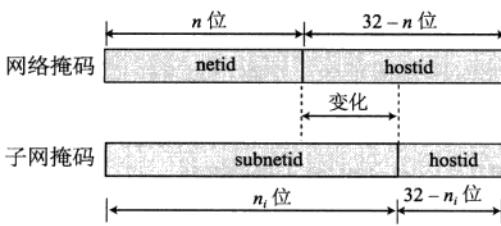


图 5.25 网络掩码和子网掩码

子网划分增加了网络标识的长度，同时也减少了主机标识的长度。当我们把一个网络划分为 s 个子网且每个子网的主机数相同时，我们可以如下计算每个子网的子网标识：

$$n_{\text{sub}} = n + \log_2 s$$

其中， n 是网络标识的长度， n_{sub} 是各子网标识的长度，而 s 是子网的数目，它必须是 2 的乘方。

例 5.20

在例 5.19 中，我们把一个 B 类网络划分为四个子网，因此 n 的值为 16，而 $n_1 = n_2 = n_3 = n_4 = 16 + \log_2 4 = 18$ 。这就意味着该子网掩码有 18 个 1 和 14 个 0 组成。换言之，该子网掩码就是 255.255.192.0，它不同于 B 类的网络掩码（255.255.0.0）。

子网地址

当一个网络划分子网后，子网的首地址就是它的子网标识符，也是路由器在为分组选择路由使之到达正确的子网时要用到的。给定某个子网中的一个任意地址，路由器就可以找出其子网掩码，其过程与前面讨论的如何找出网络掩码的过程相同：用子网掩码对给定的地址做与运算。前面讨论的快捷算法同样适用于此处求子网地址。

例 5.21

在例 5.19 中，我们描绘了如何把一个网络划分为四个子网。已知子网 2 中有一个地址为 141.14.120.77，我们可以求出该子网地址如下：

地址	→	141	.	14	.	120	.	77
掩码	→	255	.	255	.	192	.	0
子网地址	→	141	.	14	.	64	.	0

第一、二和四个字节的值使用了与运算的第一种快捷算法得到。第三个字节使用了与运算的第二种快捷算法得到。

设计子网

我们将在讨论无分类编址时讨论如何设计一个子网。因为分类编址只是无分类编址的一种特殊情况，所以后面的讨论将同样适用于此处的分类编址。

5.2.6 构造超网

子网划分并不能完全解决分类编址中地址耗尽的问题，因为大多数机构组织并不愿意和别人共享自己的地址块。C 类地址目前还能申请得到，但是 C 类地址空间无法满足那些希望加入到因特网中来的新的组织的需要，因此，另一个解决办法就是构造超网。

(supernetting)。在构成超网时，一个组织可把若干个C类地址块合并成为一个更大的地址段。换言之，多个网络合并成为一个超网。使用这种方法时，一个组织可申请多个C类地址块，而不只一个。例如，如果一个组织需要1000个地址，那么它就可以申请获取四个C类地址。

超网掩码

超网掩码(supernet mask)和子网掩码正好相反。C类子网掩码中1的个数要比C类默认掩码中1的个数多，而C类超网掩码中1的个数要比默认掩码中1的个数少。

图5.26描绘了子网掩码、默认掩码和超网掩码的区别。将一个地址块划分为8个子地址块的子网掩码中，1的个数比默认掩码中的要多3个($2^3 = 8$)，而将8个地址块合并为一个超地址块的超网掩码中，1的个数则比默认掩码中的要少3个。

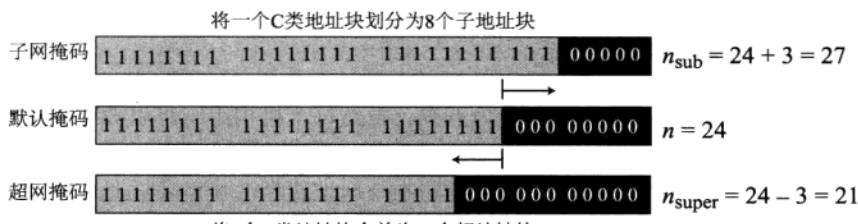


图5.26 子网掩码、默认掩码和超网掩码之比较

在构造超网时，可以被合并成一个超网的C类地址块的数目必须是2的乘方。超网标识的长度可以通过以下公式求得：

$$n_{\text{super}} = n - \log_2 c$$

其中 n_{super} 指明了超网标识的长度(以位为单位)，而 c 指的是有多少个C类地址块被合并。

遗憾的是，构造超网带来了两个新的问题：第一，合并为超网的地址块的数目必须是2的乘方，也就是说虽然一个机构组织只需要7个地址块，但是它还是要被赋予至少8个地址块(地址浪费)。第二，构造超网和子网划分会使因特网分组的路由选择变得相当复杂。

5.3 无分类编址

在分类编址中划分子网和构造超网并没有从实际上解决地址耗尽的问题，还使得地址分配和路由选择变得更加困难。随着因特网的不断扩大，很明显我们需要一个更大的地址空间来作为一种长期解决办法。但是，更大的地址空间就需要把IP的地址长度增加，也就意味着IP分组的格式需要改变。虽然这种长期的解决方案已经被设计出来，称为IPv6(参见第26章~第28章)，但是人们还设计了一种临时的解决方案，它使用原有的地址空间不变，但是改变了地址的分配方法，以便向每个机构组织提供一种更公正共享的办法。这种临时解决方案仍然使用IPv4地址，但是称为无分类编址。换言之，在分配地址时，类别的特权被取消了，以此来补偿地址耗尽问题。

使用无分类编址还有另一个原因。在 20 世纪 90 年代，因特网服务提供者（ISP）开始流行。ISP 是这样的一种组织，它可以让个人、小公司以及中等规模的组织提供到因特网的接入，这些中等规模的组织不想创建自己的因特网网站，也不想忙于为自己的雇员提供因特网服务（如电子邮件服务），而 ISP 则能够提供这样的一些服务。一个 ISP 可分配得到一个较大的地址段，然后再把这些地址进一步地划分为若干个组（每个组具有 1、2、4、8 或 16 个地址等等），并把其中的一组分配给一个家庭或公司用户。这些用户通过拨号调制解调器、DSL 或电缆调制解调器连接到 ISP。不过，每个用户都需要多个 IPv4 地址。

1996 年，因特网管理机构宣布了一种新的体系结构，称为无分类编址（classless addressing）。无分类编址时使用的是可变长度的地址块，这些地址块不属于任何类。每个地址块可以有 1 个地址、2 个地址、4 个地址、128 个地址等等。

5.3.1 可变长度地址块

在分类编址时整个地址空间被划分为五类。虽然每个机构组织可以被授权使用 A 类、B 类或 C 类中的任一种地址块，但是这些地址块的长度是预先定义的，也就是说这些机构只能从三种尺寸的地址块中选择一个。D 类的唯一地址块和 E 类的唯一地址块保留用作特殊用途。在无分类编址时整个地址空间被划分为许多不同大小的地址块。从理论上讲，这些地址块可以有 2^0 ， 2^1 ， $2^2 \dots$ ， 2^{32} 个地址。唯一的限制是一个地址块中的地址数必须是 2 的乘方，我们稍后还要讨论。一个组织可以分配一个地址块。图 5.27 描绘了如何将整个地址空间划分为不重叠的地址块。

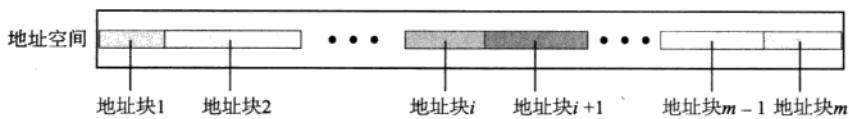


图 5.27 无分类编址中可变长度的地址块

5.3.2 两级编址

在分类编址时，通过将一个地址划分为网络标识和主机标识两部分来提供两级编址。网络标识指明了网络，而主机标识则指明了网络中的一台主机。这个思想同样也可应用于无分类编址中。当一个组织被授权使用一个地址块，这个地址块实际上也可划分为两部分：前缀（prefix）和后缀（suffix）。前缀的作用与网络标识一样，而后缀的作用与主机标识一致。某个地址块中的所有地址都具有相同的前缀，且每个地址的后缀各不相同。图 5.28 描绘了无分类地址块中的前缀和后缀。

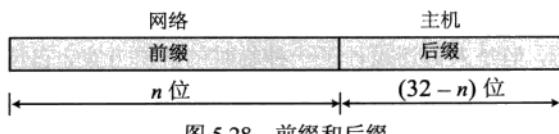


图 5.28 前缀和后缀

在无分类编址中，前缀指明了网络，而后缀指明了主机。

在分类编址中，网络标识的长度 n 取决于该地址的类别，它的值只能是 8、16 或 24 三者之一。在无分类编址中，前缀的长度 n 取决于地址块的尺寸，它可以是 0, 1, 2, 3, …, 32。无分类编址时 n 的值称为前缀长度 (prefix length)，而 $32 - n$ 的值称为后缀长度 (suffix length)。

在无分类编址中，前缀长度可以在 1 到 32 之间。

例 5.22

如果将整个因特网视为具有 4294967296 个地址的一个地址块，那么它的前缀长度和后缀长度各是多少？

解

在这种情况下，前缀的长度为 0，后缀的长度为 32。所有这 32 位的变化组合一共定义了此地址块中的 $2^{32} = 4294967296$ 台主机。

例 5.23

如果将因特网划分为 4294967296 个地址块，且每个地址块中只有一个地址，那么它的前缀长度和后缀长度分别是多少？

解

在这种情况下，每个地址块的前缀的长度为 32，而后缀长度为 0。所有这 32 位都要被用来定义 $2^{32} = 4294967296$ 个地址块。每个地址块中仅有一个定义了地址块自己的地址。

例 5.24

地址块中的地址数目与前缀长度的值 n 成反比。 n 的值越小意味着地址块越大， n 的值越大则地址块越小。

斜线记法

当我们给定了某个地址块中的一个地址，并希望从中提取出有关该地址块的信息时，网络标识的长度（分类编址），或者说前缀的长度（无分类编址），担当着非常重要的作用。不过此时的分类编址和无分类编址要区别对待。

- 分类编址时，网络标识的长度是地址所固有的。给定一个地址，我们就知道这个地址的类别，于是就可以求出网络标识的长度（8、16 或 24）。
- 无分类编址时，如果我们仅仅给出地址块中的一个地址是无法求出其前缀长度的。这个给定的地址可以属于任意前缀长度的地址块。

无分类编址时，如果我们要找出该地址所属的地址块，就必须在每个地址中包含其前缀长度。在这种情况下，前缀长度 n 被附加在地址后面，并用斜线分隔开。这种表示法的非正式叫法是斜线记法 (slash notation)。于是，无分类编址中的一个地址就可以表示为如右所示（图 5.29）。

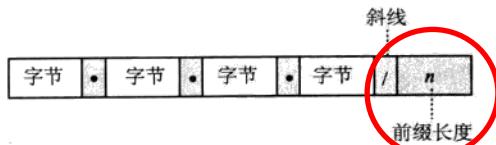


图 5.29 斜线记法

斜线记法的正式名称是无分类域间路由选择 (classless interdomain routing) 或 CIDR (发音为 cider) 记法。

在无分类编址中，我们需要知道地址块中的一个地址，以及定义该块的前缀长度。

例 5.25

无分类编址时，仅凭一个地址还不足以定义它所属的地址块。例如，地址 230.8.24.56 就可能属于多个地址块。下面列出了其中一些地址块，并给出了每个地址块的前缀长度。

前缀长度: 16	→ 地址块: 230.8.0.0	到	230.8.255.255
前缀长度: 20	→ 地址块: 230.8.16.0	到	230.8.31.255
前缀长度: 26	→ 地址块: 230.8.24.0	到	230.8.24.63
前缀长度: 27	→ 地址块: 230.8.24.32	到	230.8.24.63
前缀长度: 29	→ 地址块: 230.8.24.56	到	230.8.24.63
前缀长度: 31	→ 地址块: 230.8.24.56	到	230.8.24.57

网络掩码

在无分类编址中，网络掩码的思想与分类编址是一样的。一个网络掩码就是一个 32 位的数，这个数的左边 n 位全部置 1，而其余位全部置 0。

例 5.26

以下是一些用斜线记法定义的地址。

- a. 在地址 12.23.24.78/8 中，网络掩码是 255.0.0.0。这个网络掩码有 8 个 1 和 24 个 0。它的前缀长度是 8，后缀长度是 24。
- b. 在地址 130.11.232.156/16 中，网络掩码是 255.255.0.0。这个网络掩码有 16 个 1 和 16 个 0。它的前缀长度是 16，后缀长度是 16。
- c. 在地址 167.199.170.82/27 中，网络掩码是 255.255.255.224。这个网络掩码有 27 个 1 和 5 个 0。它的前缀长度是 27，后缀长度是 5。

提取地址块的信息

以斜线记法 (CIDR) 表示的地址中包含了我们需要知道的有关该地址块的所有信息：首地址（网络地址）、地址数及其末地址。这三个信息可以用如下方法求得：

- 该地址块的地址数可以用以下方法求得：

$$N = 2^{32-n}$$

其中 n 是前缀长度， N 是地址块的地址数。

- 地址块的首地址可通过用网络掩码对这个地址进行与运算求出：

$$\text{首地址} = (\text{任意地址}) \text{AND} (\text{网络掩码})$$

另外，我们还可以保留该地址块中任意地址的左边 n 位不变，并把其余的 $32 - n$ 位置为 0，即可得到首地址。

- 我们有两种方法可以找出一个地址块的末地址，或者用地址块中的地址数加上首地址，或者直接用网络掩码的反码对该地址进行或运算：

$$\text{末地址} = (\text{任意地址}) \text{OR} [\text{NOT}(\text{网络掩码})]$$

另外，我们还可以保留该地址块中任意地址的左边 n 位不变，并把其余的 $32 - n$ 位置为 1，即可得到末地址。

例 5.27

如果地址块中的一个地址是 167.199.170.82/27，求这个地址块的地址数、首地址以及末地址各是多少？

解

n 的值是27，它的网络掩码就是由27个1和5个0组成，也就是255.255.255.240。

- 该网络的地址数为 $2^{32-n} = 2^{32-27} = 2^5 = 32$ 。
- 我们可以用AND运算来求出它的首地址（网络地址）。首地址是167.199.170.64/27。

二进制表示的地址: 10100111 11000111 10101010 01010010

网络掩码: 11111111 11111111 11111111 11100000

首地址: 10100111 11000111 10101010 01000000

c. 要找到末地址，我们首先求出网络掩码的反码，然后再用它和给定的地址进行OR运算，得到的末地址为167.199.170.95/27。

二进制表示的地址: 10100111 11000111 10101010 01010010

网络掩码的反码: 00000000 00000000 00000000 00011111

末地址: 10100111 11000111 10101010 01011111

例 5.28

某地址块中有一个地址为17.63.110.114/24。求这个地址块的地址数、首地址以及末地址各是多少？

解

它的网络掩码为255.255.255.0。

- 该网络中的地址数为 $2^{32-24} = 256$ 。
 - 要找出首地址，我们使用本章前面讨论过的快捷算法：
- | | | | | |
|------------|-------|-------|-------|-----|
| 地址: | 17 . | 63 . | 110 . | 114 |
| 网络掩码: | 255 . | 255 . | 255 . | 0 |
| 首地址 (AND): | 17 . | 63 . | 110 . | 0 |
- 因此它的首地址为17.63.110.0/24。
- 为了求出末地址，我们要使用网络掩码的反码，再用前面讨论过的第一种快捷算法。求出的末地址为17.63.110.255/24。

地址: 17 . 63 . 110 . 114

掩码的反码 (NOT): 0 . 0 . 0 . 255

末地址 (OR): 17 . 63 . 110 . 255

例 5.29

某地址块中有一个地址为110.23.120.14/20。求这个地址块的地址数、首地址以及末地址各是多少？

解

它的网络掩码为255.255.240.0。

- 该网络的地址数为 $2^{32-20} = 4096$ 。
- 要找出首地址，我们对第一、二和四个字节使用第一种快捷算法，对第三个字节使用第二种快捷算法，最后求出的首地址是110.23.112.0/20。

地址: 110 . 23 . 120 . 14

网络掩码: 255 . 255 . 240 . 0

首地址 (AND): 110 . 23 . 112 . 0

c. 为了求出末地址，对第一、二和四个字节使用第一种快捷算法，对第三个字节使用第二种快捷算法，并用网络掩码的反码来做或运算，求出的末地址为 110.23.127.255/20。

地址:	110 .	23 .	120 .	14
掩码的反码 (NOT):	0 .	0 .	15 .	255
首地址 (OR):	110 .	23 .	127 .	255

5.3.3 地址块的分配

无分类编址的下一个问题是地址块的分配问题。这些地址块是如何分配的？分配地址块的最终责任被交给了一个全球管理机构，称为因特网名字与号码指派公司（Internet Corporation for Assigned Names and Numbers, ICANN）。不过 ICANN 通常不会直接向因特网的个人用户分配地址。它将大块的地址指派给某一个 ISP（或被视为 ISP 的大型机构）。为了使 CIDR 能够正常运转，在分配地址块时需要遵守三个原则：

1. 申请的地址数 N 必须是 2 的乘方。这样做是为了使前缀长度的值 n 是一个整数（参见第二个限制），因此地址数可以是 1 个、2 个、4 个、8 个、16 个等等。
2. 对于一个地址块来说，根据地址数就可以求出其前缀长度的值。因为 $N = 2^{32-n}$ ，所以 $n = \log_2(2^{32} / N) = 32 - \log_2 N$ 。这就是为什么 N 必须是 2 的乘方的原因。
3. 必须是地址空间中连续的未分配地址才能被分配给申请的地址块。但是，如何选择该地址块的起始地址是有限制的。起始地址必须能够被地址块的地址数整除。要理解这一限制，我们可以用 $X \times 2^{32-n}$ 这个公式来计算起始地址，其中 X 是前缀的十进制值。换言之，起始地址就是 $X \times N$ 。

例 5.30

某 ISP 申请一个有 1000 个地址的地址块。以下地址块被授权：

- a. 因为 1000 不是 2 的乘方，所以应当授权 1024 个地址 ($1024 = 2^{10}$)。
- b. 地址块前缀长度的计算方法为： $n = 32 - \log_2 1024 = 22$ 。
- c. 起始地址选择为 18.14.12.0（它可以被 1024 整除）。

被授权的地址块是 18.14.12.0/22。首地址是 18.14.12.0/22，末地址是 18.14.15.255/22。

与分类编址之间的关系

在讨论无分类编址时涉及到的所有内容都可被应用于分类编址。事实上，分类编址是无分类编址的一个特例，其中 A 类、B 类和 C 类地址块的前缀长度分别为 $n_A = 8$, $n_B = 16$, $n_C = 24$ 。通过对照表 5.1 中定义的前缀长度，分类编址的地址块很容易就可以转换成无分类编址的地址块。

表 5.1 分类编址的前缀长度

类别	前缀长度	类别	前缀长度
A	/8	D	/4
B	/16	E	/4
C	/24		

例 5.31

假设某组织曾经分配得到了一个 A 类的地址块 73.0.0.0。如果这个地址块一直没有被管理机构征用，那么无分类的体系结构就认为这个组织拥有的是一个网络地址为 73.0.0.0/8 的无分类地址块。

5.3.4 子网划分

通过子网划分就可以建立一个三级结构。授权使用某段地址的一个组织（或 ISP）可以将该地址段进一步划分为若干个子段，并为每个子网络（或子网）指派一个子地址段。请注意，该组织可以随意地继续往下分为更多级，没有任何限制。一个子网可以被划分为若干个子子网（sub-subnetwork）。一个子子网还可以被划分为若干个子子子网（sub-sub-subnetwork），依此类推。

子网设计

为了能够顺利地为分组选择路由，在为一个网络设计子网络时应当加倍小心。我们假设某组织被授权使用的地址数为 N ，前缀长度为 n ，分配下去的每个子网的地址数为 N_{sub} ，每个子网的前缀长度为 n_{sub} ，总的子网数为 s 。那么，必须要严格遵守以下原则才能确保每个子网络能够正常运转。

1. 每个子网络的地址数应当是 2 的乘方。
2. 每个子网络的前缀长度应当用以下公式求得：

$$n_{\text{sub}} = n + \log_2(N/N_{\text{sub}})$$

3. 每个子网络的起始地址应当能够被它的地址数整除。要做到这一点，我们应该先为较大的子网络指派地址。

为子网络分配地址时受到的限制与为网络分配地址时受到的限制是一致的。

找出每个子网络的信息

在子网络设计完成之后，有关每个子网络的信息，如首地址和末地址，可以套用我们之前描述的如何找出因特网中各网络信息的过程。

例 5.32

某组织被授权使用地址块 130.34.12.64/26。该组织需要四个子网络，且每个子网络拥有的主机数相同。请为该组织设计子网络，并求得每个网络的相关信息。

解

整个网络的地址数为 $N = 2^{32-26} = 64$ 。使用上节描绘的过程可求得此网络的首地址为 130.34.12.64/26，且末地址为 130.34.12.127/26。现在我们来设计子网络：

1. 我们为每个子网络分配 16 个地址，这就满足了第一个要求（64/16 是 2 的乘方）。
2. 每个子网络的子网掩码（subnetwork mask）是：

$$n_1 = n_2 = n_3 = n_4 = n + \log_2(N/N_i) = 26 + \log_2 4 = 28$$

3. 我们为每个子网分配 16 个地址，且从第一个可用地址开始。图 5.30 描绘了每个子网的子地址块。请注意，每个子网的起始地址都能被子网络的地址数整除。

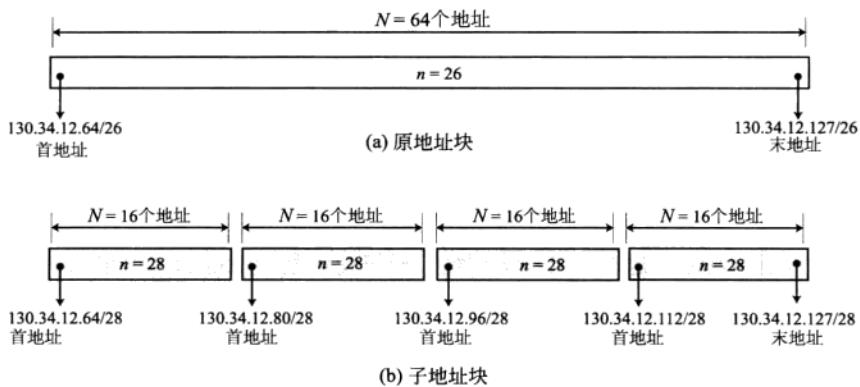


图 5.30 例 5.32 的解

例 5.33

某组织被授权使用一个起始地址为 14.24.74.0/24 的地址块。该组织需要用到三个子网，它们的三个子地址块的具体要求分别如下所示：

- 一个子地址块有 120 个地址；
- 一个子地址块有 60 个地址；
- 一个子地址块有 10 个地址。

解

在这个地址块中共有 $2^{32-24} = 256$ 个地址。其中首地址为 14.24.74.0/24，末地址为 14.24.74.255/24。

a. 第一个子地址块的地址数不是 2 的乘方。我们需要分配给它 128 个地址。第一个地址可用作网络地址，而最后一个地址作为一个特殊地址，仍然有 126 个地址可用。这个子网的前缀长度为 $n_1 = 24 + \log_2(256/128) = 25$ ，它的首地址是 14.24.74.0/25，末地址是 14.24.74.127/25。

b. 第二个子地址块的地址数也不是 2 的乘方。我们需要分配给它 64 个地址。第一个地址可用作网络地址，而最后一个地址作为一个特殊地址，仍然有 62 个地址可用。这个子网的前缀长度为 $n_2 = 24 + \log_2(256/64) = 26$ ，它的首地址是 14.24.74.128/26，末地址是 14.24.74.191/26。

c. 第三个子地址块的地址数也不是 2 的乘方。我们要分配给它 16 个地址。第一个地址可用作网络地址，而最后一个地址作为一个特殊地址。仍然有 14 个地址可用。这个子网的前缀长度为 $n_3 = 24 + \log_2(256/16) = 28$ ，它的首地址是 14.24.74.192/28，末地址是 14.24.74.207/28。

d. 如果我们将上面这几个子地址块的地址数全部相加，一共得到 128 个地址，也就是说还有 48 个地址剩余未用。这个剩余地址段的首地址是 14.24.74.209，末地址是 14.24.74.255。我们还不知道它的前缀长度是多少。

- e. 图 5.31 所示为这些地址块的分配情况。图中显示了每个地址块的首地址。

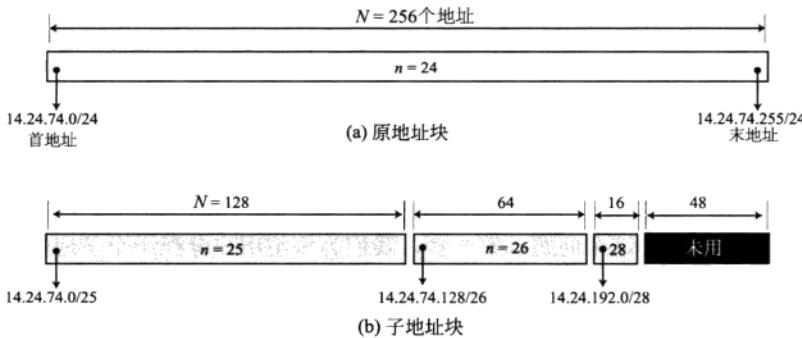


图 5.31 例 5.33 的解

例 5.34

假设某公司有三个办事处：中心办事处、东部办事处和西部办事处。中心办事处通过一个专用的点对点广域网线路连接到东部和西部办事处。这个公司授权使用的地址块共有 64 个地址，且起始地址为 70.12.100.128/26。管理部门决定给中心办事处分配 32 个地址，剩下的地址由其余两个办事处平均分配。

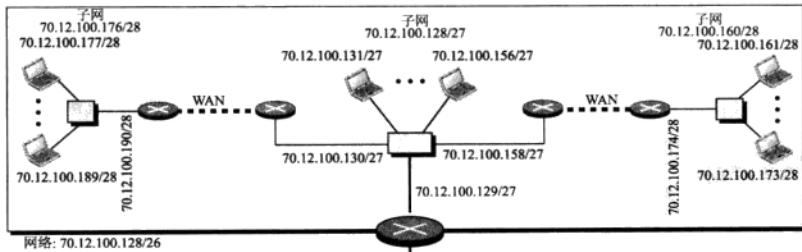
1. 地址数的分配情况如下：

$$\text{中心办事处 } N_c = 32 \quad \text{东部办事处 } N_e = 16 \quad \text{西部办事处 } N_w = 16$$

2. 我们求得每个子网的前缀长度分别如下：

$$n_c = 26 + \log_2(64/32) = 27 \quad n_e = 26 + \log_2(64/16) = 28 \quad n_w = 26 + \log_2(64/16) = 28$$

3. 图 5.32 描绘了由管理部门设计的配置图。中心办事处使用从 70.12.100.128/27 到 70.12.100.159/27 的地址。公司在这些地址中选了三个地址用于路由器，且该地址块的末地址保留。东部办事处使用从 70.12.100.160/28 到 70.12.100.175/28 的地址。其中有一个地址用于路由器，且其末地址保留不用。西部办事处使用从 70.12.100.176/28 到 70.12.100.191/28 的地址。其中有一个地址用于路由器，且其末地址保留不用。公司没有为点对点的广域网连接使用任何地址。



目的地址为 70.12.100.128 到 70.12.100.191 的分组都被交付到这个网络

图 5.32 例 5.34 的解

地址聚合

CIDR 体系的一个优点就是地址聚合。ICANN 把大块的地址段指派给 ISP。ISP 把得到的地址块再划分为较小的子地址块，然后把这样的子地址块授权给它的客户使用。也就是说许多地址块聚合成一个大块，然后将其授权给一个 ISP 使用。

例 5.35

某 ISP 授权得到一个地址块，起始地址是 190.100.0.0/16（共有 65 536 个地址）。这个

ISP 需要把这些地址分配给如下的三组用户：

- 第一组有 64 个用户，每个用户大约需要 256 个地址。
- 第二组有 128 个用户，每个用户大约需要 128 个地址。
- 第三组有 128 个用户，每个用户大约需要 64 个地址。

试设计这些子地址块，并求出在这些地址分配后，还有多少地址可以使用？

解

让我们分两步来解决这个问题。第一步，我们为每组用户分配一个子地址块。分配给每个组的总地址数以及每个子地址块的前缀长度如下求得：

$$\text{第1组: } 64 \times 256 = 16384$$

$$n_1 = 16 + \log_2(65536/16384) = 18$$

$$\text{第2组: } 128 \times 128 = 16384$$

$$n_2 = 16 + \log_2(65536/16384) = 18$$

$$\text{第3组: } 128 \times 64 = 8192$$

$$n_3 = 16 + \log_2(65536/8192) = 19$$

图 5.33 描绘了上述第一级设计的情况。

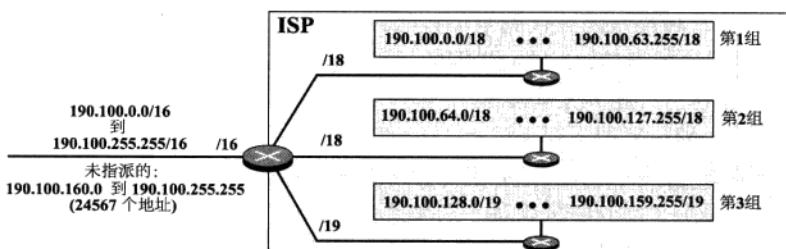


图 5.33 例 5.35 的解：第一步

现在我们来考虑每一个组的情况。在每一组中，子网络的前缀长度是变化的，且取决于每个子网络使用的地址数目。图 5.34 描绘了第二级的结构。请注意，我们以每个用户的首地址作为该子网的地址，并将其末地址保留不用。

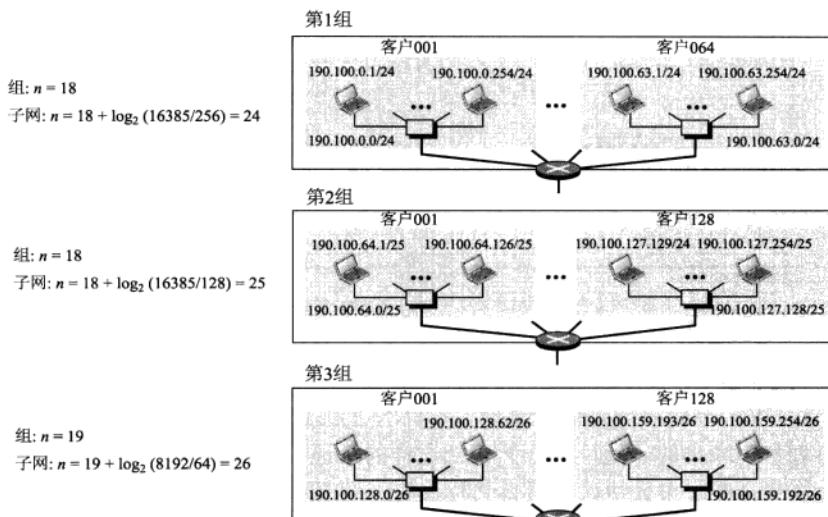


图 5.34 例 5.35 的解：第二步

5.4 特殊地址

在分类编址中有些地址因特殊用途而被保留。无分类编址也从分类编址中继承了部分的特殊地址的策略。

5.4.1 特殊地址块

有些地址块是保留作特殊用途的。

全 0 地址

地址块 0.0.0.0/32 仅含有一个地址，它被保留用于某主机需要发送一个 IPv4 分组，但又不知道自己的地址的情况下。通常用在主机正在启动，尚不知道自己的 IPv4 地址时。主机为了找出自己的地址，就向引导服务器（称为 DHCP 服务器，将在第 18 章中讨论）发送一个 IPv4 分组，并以这种全 0 的地址作为源地址，而用受限广播地址（limited broadcast address）作为目的地址（参见图 5.35）。

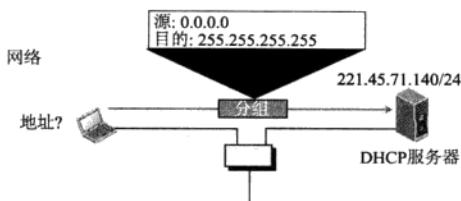


图 5.35 使用全 0 地址的例子

全 1 地址：受限广播地址

地址块 255.255.255.255/32 仅含有一个地址，它被保留作为当前网络的受限广播地址。一个主机若想把报文发送给网络中其他所有主机，就可用这个地址作为 IPv4 分组中的目的地址。但是，路由器会把具有这种类型地址的分组阻挡住，这样一来广播只能局限在本地网络。在图 5.36 中，主机发送了一个目的 IPv4 地址全部由 1 组成的数据报，该网络中的所有设备都会接收并处理这个数据报。

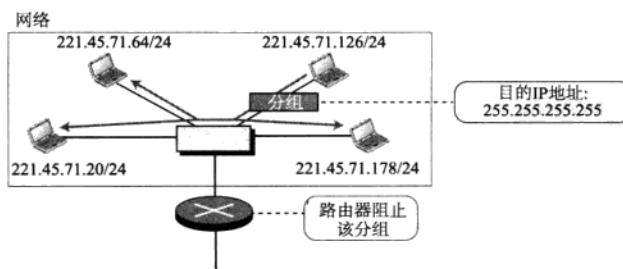


图 5.36 受限广播地址的例子

环回地址

地址块 127.0.0.0/8 被用作环回地址（loopback address），这个地址用来测试机器上的软件。在使用这个地址时，分组从来没有离开过机器，它只是简单地由协议软件返回。这个地址可用于测试 IPv4 软件。例如，像“ping”这样的应用程序，可以发送以环回地址作为

目的地址的分组，以便测试 IPv4 软件能否接收和处理分组。另一个例子就是客户进程（一个运行着的程序）可以用环回地址来向本机上的服务器进程发送一个报文。应当注意，这种地址在 IPv4 分组中只能用作目的地址。（参见图 5.37）。

专用地址

有一些地址块被指派为专用地址。它们不会在全球被识别。这些地址块在表 5.2 中描述。这些地址或者用于隔绝其他网络的情况下，或者用于具有网络地址转换技术的连接（参见本章稍后的有关 NAT 的小节）。

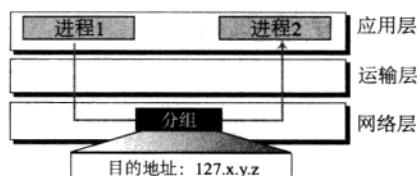


图 5.37 环回地址的例子

表 5.2 专用网络地址

地址块	地址数	地址块	地址数
10.0.0.0/8	16777216	192.168.0.0/16	65536
172.16.0.0/12	1047584	169.254.0.0/16	65536

多播地址

地址块 224.0.0.0/4 为多播通信保留。我们将在第 12 章详细讨论多播问题。

5.4.2 每个地址块中的特殊地址

各地址块中有一些地址有着特殊的用途，这只是推荐的，而不是强制的。原则上这些地址不会被指派给任何主机，但是如果一个地址块（或子块）太小了，再让它拿出部分地址来做特殊用途是它所负担不起的。

网络地址

我们已经讨论过网络地址。一个地址块的首地址（即后缀全部为 0）定义为网络地址。实际上，这个地址指的就是网络本身（缆线连接），而不是网络中的哪一台主机。当然，子网络中的首地址称为子网络地址，其作用也是一样的。

直接广播地址

一个地址块的末地址（即后缀全部为 1）可用作直接广播地址（direct broadcast address）。通常，路由器会用这个地址把一个分组发送给某个特定网络上的所有主机。所有主机都会

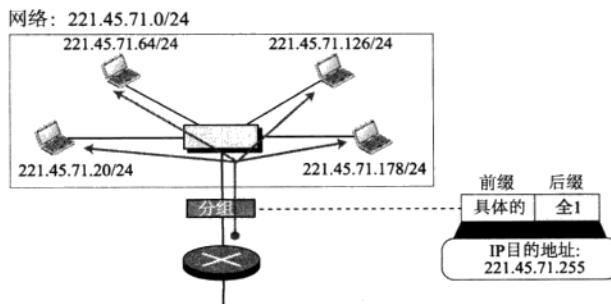


图 5.38 一个直接广播地址的例子

接受具有这种类型的目的地址的分组。要注意到，这个地址在IPv4分组中只能被用作目的地址。在图5.38中，路由器发送的数据报中含有一个后缀全部为1的IPv4目的地址。这个网络上的所有设备都要接收和处理这个数据报。

5.5 NAT

通过ISP来分发地址又带来了一个新的问题。假设一个ISP向某个小企业或家庭用户授予了一小段地址的使用权。如果这个企业或家庭发展壮大了，需要一个较大的地址段，那么该ISP可能无法满足新增地址段的授权需求，因为该地址段的前后地址都已经分配给其他网络了。不过在一个小型网络中，绝大多数情况下仅有一部分计算机需要同时接入因特网。这就意味着分配的地址数不一定要完全匹配网络中的计算机数目。例如，假设一个小企业拥有20台计算机，而同时接入因特网的计算机数最多只有5台。大多数计算机或者在做一些不需要接入因特网的工作，或者正在互相通信。这个小企业可以用TCP/IP协议来实现企业内部的以及全球的通信，它可以为内部通信使用20（或25）个地址，这些地址来自我们之前讨论过的专用地址块，另一方面，5个用于全球通信的地址则由ISP来指派。

有一种技术称为网络地址转换（network address translation, NAT），它可用于提供在专用地址和全球地址之间的互相映射，同时也支持虚拟专用网络（将在第30章讨论）。这个技术允许一个站点的内部通信使用一组专用地址，而与世界其他地方进行通信时又使用另一组（至少有一个）全球因特网地址。这个站点必须仅有一条到全球因特网的连接，而且这条连接通过了一个运行NAT软件的具有NAT功能的路由器。图5.39描绘了一个简单的NAT实现方案。

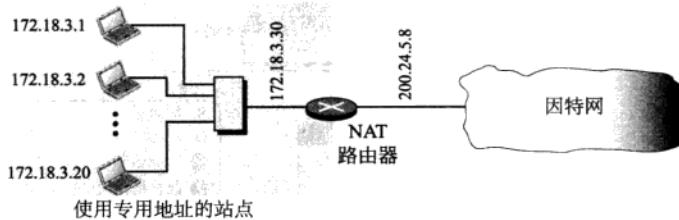


图5.39 NAT

如图所示，这个专用网内部使用的是专用地址。连接该网络到全球因特网的路由器使用了一个专用地址和一个全球地址。对因特网的其他地方来说这个专用网络是透明的，它们看到的只是地址为200.24.5.8的那个NAT路由器。

5.5.1 地址转换

所有外出的分组都要通过这台NAT路由器，它将分组中的源地址替换为全球NAT地址。所有进入的分组也都要通过这台NAT路由器，它又将这些分组中的目的地址（NAT路由器的全球地址）替换为相应的内部地址。图5.40描绘了地址转换的一个例子。

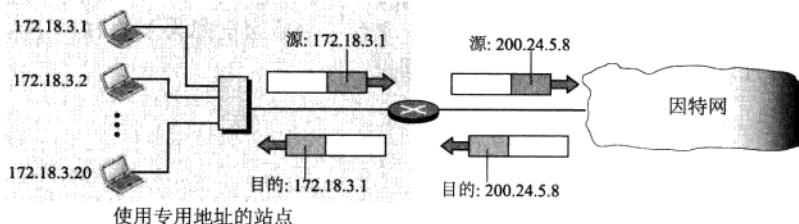


图 5.40 地址转换

5.5.2 转换表

读者可能已经注意到，外出分组中源地址的转换过程是很容易理解的。但是 NAT 路由器又是怎么知道来自因特网的每个分组的目的地址是什么呢？在这个网络内部可能有几十个或几百个专用 IP 地址，每个地址属于一台特定的主机。如果这个 NAT 路由器有一张转换表（translation table），这个问题就可以解决了。

使用一个 IP 地址

一张最简单的转换表只有两个列：一个专用地址和一个外部地址（分组的目的地址）。在路由器为外出分组转换源地址时，也注意到了该分组的目的地址，即这个分组要到哪里去。当终点返回的响应到达时，路由器就利用返回分组的源地址作为外部地址，查找相对应的专用地址。图 5.41 描绘了这一思想。

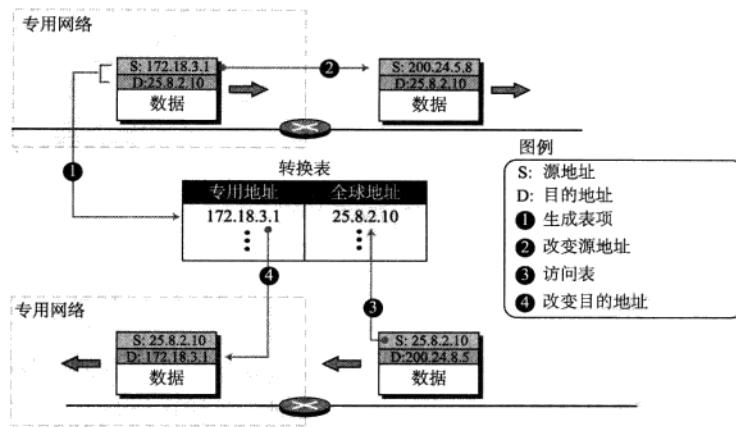


图 5.41 转换

在这个策略里，通信必须总是由专用网络发起的。前面所描述的 NAT 机制要求由专用网络发起通信。如我们将看到的，ISP 在使用 NAT 时，绝大多数都给一个用户只分配一个地址。而该用户可能是一个专用网络的成员，这个专用网络有很多专用地址。在这种情况下，与因特网的通信几乎都是由用户所在的站点发起的，它们使用诸如 HTTP、TELNET 或 FTP 这样的客户端程序来访问相应的服务器程序。例如，从用户所在站点之外的地方发

来的一封电子邮件被 ISP 的邮件服务器接收后，这个邮件被保存在该用户的邮箱里，直至用户主动使用像 POP 这样的协议来读取。

如果一个专用网络使用了 NAT 技术，那么它就不能运行服务器程序来为网络之外的客户端提供服务。

使用 IP 地址池

如果 NAT 路由器仅仅使用一个全球地址，那么在专用网络中，一次只能允许一台主机访问某个外部主机。为了消除这个限制，NAT 路由器可以使用全球地址池。例如，某 NAT 路由器使用的不仅仅是一个全球地址（200.24.5.8），而是四个全球地址（200.24.5.8、200.24.5.9、200.24.5.10 和 200.24.5.11）。在这种情况下，专用网络上的四台主机可以同时与某个相同的外部主机通信，因为一对地址指明一个连接。但这样做还是有些问题。同时访问相同主机的连接不能超过四条。另外，在专用网络中，一台主机不能同时访问两个外部服务器程序（如 HTTP 和 TELNET）。类似地，两台主机也不能同时访问一个相同的外部服务器程序（如 HTTP 或 TELNET）。

使用 IP 地址和端口地址

为了允许专用网络内的主机和外部服务器程序之间多对多的关系，我们就需要在转换表中加入更多的信息。例如，假设在一个专用网络内部有两台地址分别为 172.18.3.1 和 172.18.3.2 的主机，它们需要访问地址为 25.8.3.2 的外部主机上的 HTTP 服务器程序。如果转换表不仅仅有两列，而是有五列，包括源端口地址和目的端口地址以及运输层协议，那就没有什么不清楚的了。表 5.3 所示为一张这样的表的例子。

表 5.3 五列的转换表

专用地址	专用端口	外部地址	外部端口	运输层协议
172.18.3.1	1400	25.8.3.2	80	TCP
172.18.3.1	1401	25.8.3.2	80	TCP
:	:	:	:	:

请注意，当来自 HTTP 的响应返回时，源地址（25.8.3.2）与目的端口地址（1400）组合在一起指明了应当将这个响应分组交付给专用网络中的哪一台主机。同时还要注意到如果要使这张转换表正确工作，这些一次性使用的端口地址（1400 和 1401）必须是唯一的。

5.6 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

5.6.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]、[Koz 05] 和[Ste 95]。

5.6.2 RFC

有几份 RFC 专门讨论了 IPv4 地址，包括：RFC 917、RFC 927、RFC 930、RFC 932、RFC 940、RFC1122 和 RFC 1519。

5.7 重要术语

地址聚合	受限广播地址
地址空间	环回地址
二进制记法	网络标识
地址块	网络地址
A 类地址	网络地址转换 (NAT)
B 类地址	网络掩码
C 类地址	前缀
D 类地址	前缀长度
E 类地址	斜线记法
分类编址	子网
无分类编址	子网掩码
无分类域间路由选择	子网划分
默认掩码	子网络
直接广播地址	后缀
点分十进制记法	后缀长度
十六进制记法	超网掩码
主机标识	构造超网
IP 地址	地址转换表

5.8 本章小结

- 在 TCP/IP 协议族的 IP 层使用的标识符称为因特网地址或 IP 地址。IPv4 地址是一个 32 位的地址。地址空间就是协议所使用的地址的总数量。IPv4 的地址空间为 2^{32} 或 4294967296。
- 在分类编址中，IPv4 地址空间共分为五类：A、B、C、D 和 E。一个机构组织可以被授权使用 A、B 或 C 这三种类型中的一种地址块。D 类和 E 类为特殊用途而保留。A 类、B 类和 C 类的 IP 地址可以被分成网络标识和主机标识两部分。
- 在分类编址中，地址块的首地址称为网络地址。它指明了该地址所属的网络。网络地址用来为分组选择路由，使其能够到达目的网络。

- 分类地址中的网络掩码或默认掩码是一个32位的数，这个数的左边n位全部置1，而右边 $32-n$ 位全部置0。路由器通过它就可以从一个分组的目的地址中求出其网络地址。
- 将一个网络划分为若干个较小的子网络的概念称为子网划分。与网络掩码一样，子网掩码用于在给定一个IP目的地址的情况下找出其子网地址。在构造超网时，某个组织可以将若干个C类地址块合并起来，以创建一个较大的地址段。
- 1996年，因特网管理机构宣布了一种新的体系结构称为无分类编址或CIDR，它使得一个组织能够拥有任意长度的地址块，只要这个长度是2的乘方。
- 无分类编址的地址也被划分为两部分：前缀和后缀。前缀的作用与网络标识一样，而后缀的作用与主机标识一致。一个地址块中的所有地址都具有相同的前缀，且每个地址的后缀各不相同。
- 在IPv4中有些地址块是为特殊用途而保留的。另外，一个地址块中的某些地址通常被用作特殊地址。这些地址不会指派给任何主机。
- 为了改善地址分发的问题，人们创建了NAT技术以便将一个网络中的专用地址与它在因特网上的全球地址分隔开。通过一张地址转换表，来自具有此功能的专用网络中的专用地址就可以被转换为全球地址。这张转换表不仅把IP地址，同时还要把口号从专用的映射成为全球的地址，反之亦然。

5.9 实践安排

5.9.1 习题

1. 下列各系统的地址空间分别是多少?
 - a. 使用8位地址的系统
 - b. 使用16位地址的系统
 - c. 使用64位地址的系统
2. 某地址空间的地址总数为1024。那么要表示一个地址需要多少位?
3. 某地址空间使用了三种符号(0、1和2)来表示地址。若每个地址由10个符号组成，那么在这个系统中的可用地址数是多少?
4. 试把以下的IP地址从点分十进制记法转换为二进制记法。
 - a. 114.34.2.8
 - b. 129.14.6.8
 - c. 208.34.54.12
 - d. 238.34.2.1
5. 试把以下的IP地址从点分十进制记法转换为十六进制记法。
 - a. 114.34.2.8
 - b. 129.14.6.8
 - c. 208.34.54.12

- d. 238.34.2.1
6. 试把以下的 IP 地址从十六进制记法转换为二进制记法。
- a. 0x1347FEAB
 - b. 0xAB234102
 - c. 0x0123A2BE
 - d. 0x00001111
7. 在以下各类地址中，网络标识需要用到多少个十六进制的数字？
- a. A 类
 - b. B 类
 - c. C 类
8. 试把以下的 IP 地址从二进制记法转换为点分十进制记法。
- a. 01111111 11110000 01100111 01111101
 - b. 10101111 11000000 11110000 00011101
 - c. 11011111 10110000 00011111 01011101
 - d. 11101111 11110111 11000111 00011101
9. 求以下 IP 地址的类别：
- a. 208.34.54.12
 - b. 238.34.2.1
 - c. 242.34.2.8
 - d. 129.14.6.8
10. 求以下 IP 地址的类别：
- a. 11110111 11110011 10000111 11011101
 - b. 10101111 11000000 11110000 00011101
 - c. 11011111 10110000 00011111 01011101
 - d. 11101111 11110111 11000111 00011101
11. 求以下 IP 地址的网络标识和主机标识：
- a. 114.34.2.8
 - b. 132.56.8.6
 - c. 208.34.54.12
 - d. 251.34.98.5
12. 如果一个地址段的首地址为 14.7.24.0，末地址为 14.14.34.255，求这个地址段的地址数目。
13. 如果一个地址段的首地址为 122.12.7.0，且这个地址段一共有 2048 个地址，那么它的末地址是什么？
14. 计算下列运算结果：
- a. NOT(22.14.70.34)
 - b. NOT(145.36.12.20)
 - c. NOT(200.7.2.0)
 - d. NOT(11.20.255.255)

15. 计算下列运算结果:

- a. (22.14.70.34) AND (255.255.0.0)
- b. (12.11.60.12) AND (255.0.0.0)
- c. (14.110.160.12) AND (255.200.140.0)
- d. (28.14.40.100) AND (255.128.100.0)

16. 计算下列运算结果:

- a. (22.14.70.34) OR(255.255.0.0)
- b. (12.11.60.12) OR (255.0.0.0)
- c. (14.110.160.12) OR (255.200.140.0)
- d. (28.14.40.100) OR (255.128.100.0)

17. 在 A 类子网中, 我们知道其中一台主机的 IP 地址以及子网掩码如下:

IP 地址: 25.34.12.56 子网掩码: 255.255.0.0

那么它的首地址 (子网地址) 是什么? 末地址又是什么?

18. 在 B 类子网中, 我们知道其中一台主机的 IP 地址以及子网掩码如下:

IP 地址: 131.134.112.66 子网掩码: 255.255.224.0

那么它的首地址 (子网地址) 是什么? 末地址又是什么?

19. 在 C 类子网中, 我们知道其中一台主机的 IP 地址以及子网掩码如下:

IP 地址: 202.44.82.16 子网掩码: 255.255.255.192

那么它的首地址 (子网地址) 是什么? 末地址又是什么?

20. 求下列情况的子网掩码:

- a. A 类地址中的 1024 个子网
- b. B 类地址中的 256 个子网
- c. C 类地址中的 32 个子网
- d. C 类地址中的 4 个子网

21. 在一个地址块中, 我们知道有一个主机的 IP 地址是 25.34.12.56/16。试问这个地址块的首地址 (网络地址) 和末地址 (受限的广播地址) 各是什么?

22. 在一个地址块中, 我们知道有一个主机的 IP 地址是 182.44.82.16/26。试问这个地址块的首地址 (网络地址) 和末地址 (受限的广播地址) 各是什么?

23. 在固定长度子网划分中, 如果需要划分的子网数如下所示, 试问分别需要在掩码上再增加几个 1。

- a. 2
- b. 62
- c. 122
- d. 250

24. 某组织被授权使用地址块 16.0.0.0/8。管理员想创建 500 个固定长度的子网。

- a. 求子网掩码。
- b. 求每个子网的地址数。
- c. 求第一个子网的首地址和末地址。
- d. 求最后一个子网的首地址和末地址 (子网 500)。

25. 某组织被授权使用地址块 130.56.0.0/16。管理员想创建 1024 个子网。
- 求子网掩码。
 - 求每个子网的地址数。
 - 求第一个子网的首地址和末地址。
 - 求最后一个子网的首地址和末地址（子网 1024）。
26. 某组织被授权使用地址块 211.17.180.0/24。管理员想创建 32 个子网。
- 求子网掩码。
 - 求每个子网的地址数。
 - 求第一个子网的首地址和末地址。
 - 求最后一个子网的首地址和末地址（子网 32）。
27. 把下列掩码写成斜线记法 (/n) 格式：
- 255.255.255.0
 - 255.0.0.0
 - 255.255.224.0
 - 255.255.240.0
28. 求以下地址块的地址范围：
- 123.56.77.32/29
 - 200.17.21.128/27
 - 17.34.16.0/23
 - 180.34.64.64/30
29. 在无分类编址中，如果我们已知一个地址块的首地址和末地址，能否求出其前缀长度？如果答案是肯定的，请给出计算过程并举例说明。
30. 在无分类编址中，如果我们已知一个地址块的首地址及其地址数，能否求出其前缀长度？如果答案是肯定的，请给出计算过程并举例说明。
31. 在无分类编址中，两个不同的地址块能否具有相同的前缀长度？为什么？
32. 在无分类编址中，我们已知一个地址块的首地址以及其中的一个地址（不一定是末地址），能否求出其前缀长度？为什么？
33. 某 ISP 被授权使用一个起始地址为 150.80.0.0/16 的地址块。这个 ISP 希望将地址块分发给 2600 个用户，具体情况如下：
- 第一组有 200 个中等大小的企业用户，每个用户大约需要 128 个地址。
 - 第二组有 400 个小型企业用户，每个用户大约需要 16 个地址。
 - 第三组有 2000 个住宅用户，每个用户需要 4 个地址。
- 请为他们设计子地址块并给出每个子地址块的斜线记法。求这些地址分配出去以后还剩下多少地址可用？
34. 某 ISP 被授权使用一个起始地址为 120.60.4.0/20 的地址块。这个 ISP 希望将地址块分发给 100 个组织，且每个组织仅分得 8 个地址。请为他们设计子地址块并给出每个子地址块的斜线记法。求这些地址分配出去以后还剩下多少地址可用？
35. 某 ISP 拥有一个有 1024 个地址的地址块。它希望将这些地址分给 1024 个用户。在这种情况下是否需要子网划分？为什么？

第6章 IP分组的交付和转发

本章描述IP分组的交付和转发。交付（delivery）是指在网络层的控制下，底层各网络对分组的处理方式。像直接交付和间接交付这样概念会被讨论。转发（forwarding）指的是把分组交付到下一站的方式。我们要讨论两种不同形式的转发：基于分组目的地址的转发和基于附加在分组上的标记的转发。

目标

本章有以下几个目标：

- 讨论分组在网络层的交付以及直接交付和间接交付之间的区别。
- 讨论分组在网络层的转发以及基于目的地址的转发和基于标记的转发之间的区别。
- 讨论不同的转发技巧，包括下一跳方法、特定网络方法、特定主机方法以及默认方法。
- 分别讨论分类编址和无分类编址时路由表的内容，以及一些搜索路由表的算法。
- 介绍MPLS技术，并说明如何用它来实现基于标记的转发。
- 列出路由器的组成构件，并解释各个构件的作用以及与其他构件之间的关系。

6.1 交付

网络层监视底层物理网络对分组的处理过程，我们把这种处理定义为分组的交付。要完成把一个分组交付给它的最后终点的任务，需要使用两种不同的交付方法：直接交付和间接交付。

6.1.1 直接交付

直接交付（direct delivery）时，分组的终点是一台与交付者连接在同一个网络上的主机。直接交付发生在两种情况下，或者是分组的源点和终点都在同一个物理网络上，或者是在最后一个路由器与目的主机之间进行交付时（见图6.1）。

发送方很容易判断交付是否为直接的。发送方可以提取终点的网络地址（用掩码），然后与自己所连接的网络的地址相比较。若匹

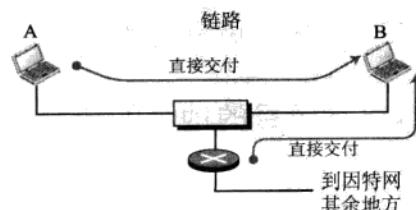


图6.1 直接交付

配，交付就是直接的。

在直接交付时，发送方通过目的 IP 地址找出目的物理地址，然后 IP 软件把目的 IP 地址和目的物理地址一起交付给数据链路层用于实际的交付。这个过程称为把 IP 地址映射到物理地址。虽然这种映射可以通过在地址表中寻找匹配来完成，但我们在第 8 章中看到，地址解析协议（ARP）动态地把 IP 地址映射为相应的物理地址。

6.1.2 间接交付

如果目的主机与交付者不在同一个网络上，分组就要间接地交付。在间接交付（indirect delivery）时，分组经过了一个又一个路由器，最后到达与终点连接在同一个网络上的路由器。图 6.2 所示为间接交付的概念。

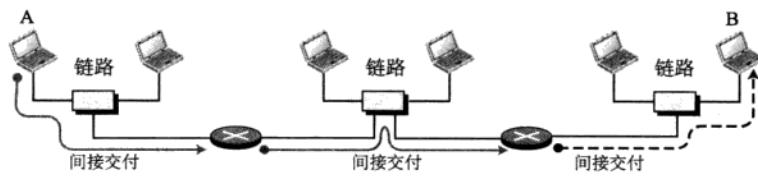


图 6.2 间接交付

间接交付时，发送方通过分组的目的 IP 地址和路由表来查找该分组应当被交付的下一个路由器的 IP 地址，然后发送方再用 ARP 协议（参见第 8 章）找出下一个路由器的物理地址。应当注意，直接交付是在终点的 IP 地址和终点的物理地址之间进行的地址映射，而间接交付是在下一个路由器的 IP 地址与下一个路由器的物理地址之间进行的地址映射。还应注意到，交付总是包括一个直接交付以及零个或多个间接交付。另外，最后的交付总是直接交付。

6.2 转发

转发意味着让分组踏上通往终点的路途。由于如今的因特网是由许多链路（网络）组合构成的，所以转发也就是将分组交付给下一跳（它可能是最后的终点，也可能只是一个中间的连接设备）。虽然 IP 协议在最初设计时是一个无连接的协议，但是将 IP 作为面向连接的协议来使用也是当前的一种趋势。

当 IP 作为无连接的协议时，转发的基础是 IP 数据报的目的地址，而当 IP 作为面向连接的协议时，转发的基础则是附加在 IP 数据报上的标记。我们首先要讨论基于目的地址的转发，然后再讨论基于标记的转发。

6.2.1 基于目的地址的转发

我们首先要讨论基于目的地址的转发。这是一种传统方式，并且目前仍然是主流。这种情况要求主机或路由器具有一张路由表才能进行转发。当主机有分组要发送，或者路由

器收到分组要进行转发时，就要搜索路由表，以便找出到达最后终点的路由。但是，这种简单的方法在今天看来，对于像因特网这样的互联网是十分低效的，因为路由表的表项数量过多将导致路由表的查找效率非常低。

转发技术

使用一些技术可以使路由表的规模变为可管理的，同时还能够处理一些如安全性这样的问题。我们将在这里简单地讨论这些方法。

下一跳方法 有一种称为下一跳方法（next hop method）的技术可以减少路由表中的内容。这种技术就是在路由表中只保留下一跳的地址，而不是保留完整路由的信息。此时路由表中的表项必须保持一致。图 6.3 描绘了如何使用这种技术来简化路由表。

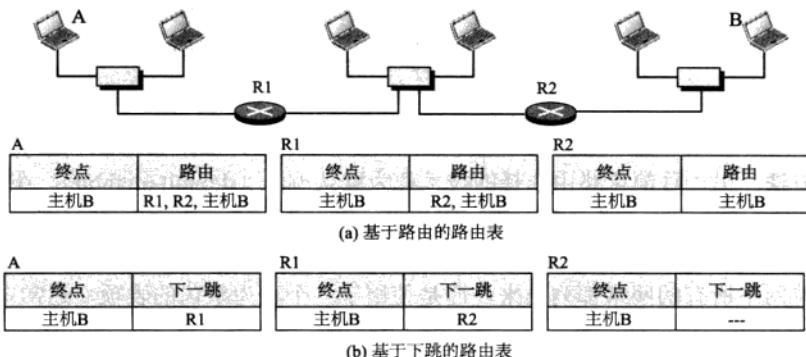


图 6.3 下一跳方法

特定网络方法 能够使路由表长度变小并简化查找过程的第二种技术称为特定网络方法（network-specific method）。此时，路由表不是对连接在同一个物理网络上的每一台主机都设置一个表项，而是只用一个表项来定义目的网络本身的地址。换言之，我们把连接在同一个网络上的所有主机看成是一个表项。例如，假定有 1000 台主机连接在同一个网络上，那么在路由表中就仅需要一个表项而不是 1000 个表项。图 6.4 描绘了这种概念。

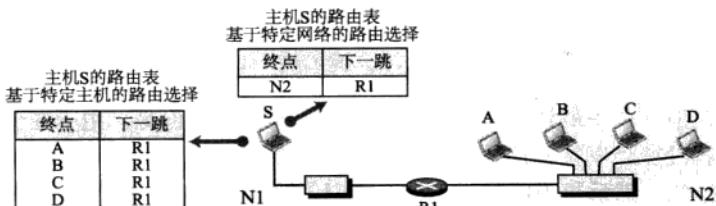


图 6.4 特定网络方法

特定主机方法 在使用特定主机方法（host-specific method）时，目的主机的地址在路由表中要给出。这种方法背后的思想与特定网络方法正好相反。这是用牺牲效率来换取其他一些优点，虽然把主机地址放在路由表中会降低效率，但有时管理人员还是想对路由选择有更多的控制。例如，在图 6.5 中如果管理人员希望所有到达主机 B 的分组都经

过路由器 R3 而不是 R1，那么在主机 A 的路由表中为 B 单独建立一个表项就能显式地定义这条路由。

特定主机的路由选择主要用在像检查路由或提供安全措施这样的特殊情况下。

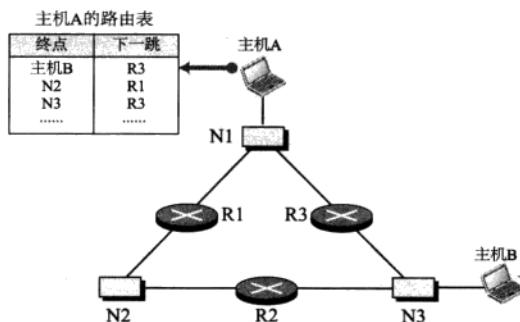


图 6.5 特定主机的路由选择

默认方法 另一种简化路由选择的技术称为默认方法 (default method)。在图 6.6 中，主机 A 所连接的网络上有两个路由器。通过路由器 R1 可以把分组转发到连接在网络 N2 上的主机。但是，对因特网的其他部分，则要使用路由器 R2。因此，主机 A 的路由表不必把整个因特网中所有的网络都列出来，而是使用了一个称为默认的表项（通常定义网络地址为 0.0.0.0）。

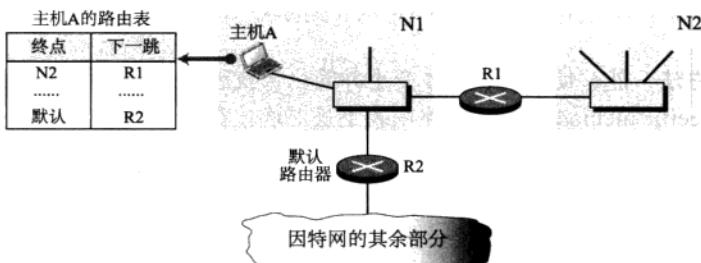


图 6.6 默认路由选择

使用分类编址时的转发

正如我们在前几章中所讨论的，分类编址有一些缺点。但是，由于在分类地址中存在默认掩码，这就使得转发过程比较简单。这一节，我们首先给出在不做子网划分时路由表的内容和转发模块，然后再说明如果划分了子网，这个模块应如何改变。

无子网划分的转发 在使用分类编址时，全球因特网中的绝大多数路由器都没有涉及子网划分。子网划分是在一个组织内部进行的。在这种情况下，一个典型的转发模块要使用三张表，每个单播类别 (A、B 和 C) 对应一张表。如果路由器支持多播，那么还要增加一张处理 D 类地址的表。有了这三张不同的表就使得搜索的效率更高。每张路由表至少要有下面三列：

- 目的网络的网络地址，它告诉我们目的主机的位置。请注意，我们使用的是特定网

络转发，而不是很少使用的特定主机转发。

2. 下一跳地址，在间接交付时它告诉我们分组应当交付到哪个路由器。在直接交付时这一列是空的。

3. 接口号，它定义了分组应当从哪一个输出端口发送出去。一个路由器通常都会连接到多个网络。每一个连接就有一个不同的端口号或接口，我们将其表示为 m₀, m₁ 等等。

图 6.7 描绘了一个简化的模块。

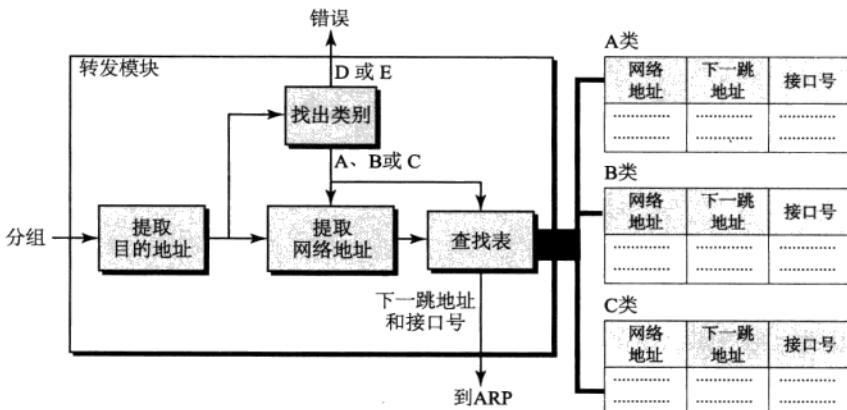


图 6.7 在无子网划分的无分类编址中简化的转发模块

在最简单的情况下，转发模块按照下列步骤工作：

1. 提取出分组的目的地址。

2. 目的地址的一个副本用来查找地址的类别。具体的做法是将地址的副本右移 28 位。结果得到一个在 0~15 之间的 4 位数。如果这个数是：

- a. 0~7, 为 A 类。
- b. 8~11, 为 B 类。
- c. 12 或 13, 为 C 类。
- d. 14, 为 D 类。
- e. 15, 为 E 类。

3. 用步骤 2 得到的结果（A 类、B 类或 C 类），以及目的地址，就可以用来提取目的网络地址。具体做法是：根据不同的类别，把目的地址最右边的 24 位、16 位或 8 位分别掩蔽掉（就是变为 0）。

4. 将地址的类别和目的网络地址一起用于查找下一跳的信息。地址类别决定了要搜索哪一张路由表。转发模块搜索这张路由表中的网络地址。如果发现有匹配，就从表中提取出相应的下一跳地址和输出端的接口号。如果没有发现匹配，就使用默认的。

5. ARP 模块（见第 8 章）用下一跳地址和接口号来找出下一个路由器的物理地址，然后请求数据链路层把分组交付到下一跳。

例 6.1

图 6.8 给出了一个假想因特网中的某一部分。试给出路由器 R1 的路由表。

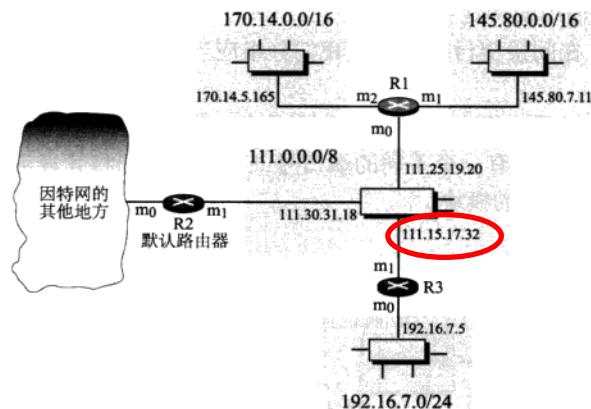


图 6.8 例 6.1 中路由选择的配置

解

图 6.9 给出了路由器 R1 使用的三张路由表。请注意，在下一跳地址这一列中有些项目是空的，因为此时终点已经在路由器所连接的同一个网络上（直接交付）。正如我们在第 8 章将要讨论的，在这种情况下 ARP 使用的下一跳地址就是分组的目的地址。

例 6.2

图 6.8 中的路由器 R1 收到一个分组，其目的地址是 192.16.7.14。试说明该分组将怎样被转发。

A类			B类		
网络地址	下一跳地址	接口	网络地址	下一跳地址	接口
111.0.0.0	-----	m0	145.80.0.0	-----	m1
			170.14.0.0	-----	m2
C类			默认的: 111.30.31.18, m0		
192.16.7.0	111.15.17.32	m0			

图 6.9 例 6.1 中的路由表

解

这个目的地址用二进制表示就是 11000000 00010000 00000111 00001110。地址的副本右移 28 位，结果得到 00000000 00000000 00000000 00001100 或 12。目的网络是 C 类。把目的地址的最右边的 8 位掩蔽掉就可提取出网络地址，结果是 192.16.7.0。搜索 C 类路由表。在第一行找到了这个网络地址。下一跳的地址 111.15.17.32 以及接口号 m0 被传递给 ARP(参见第 8 章)。

例 6.3

图 6.8 中的路由器 R1 收到一个分组，其目的地址是 167.24.160.5。试说明该分组将怎样被转发。

解

这个目的地址用二进制表示就是 10100111 00011000 10100000 00000101。将地址的副

本右移28位。结果得到00000000 00000000 0000000 0000**1010**或10。目的网络是B类。把目的地址最右边的16位掩蔽掉就可找出网络地址，结果是167.24.0.0。搜索B类路由表。因为找不到匹配的网络地址，这个分组必须转发给默认路由器（这个网络会是在因特网中其他的某个地方）。下一跳的地址111.30.31.18以及接口号m0被传递给ARP。

有子网划分的转发 在使用分类编址时，子网划分发生在组织的内部。处理子网划分的路由器不是在该组织站点的边界上，就是在站点边界的里面。如果这个组织使用了可变长度的子网划分，那么我们就需要多张路由表，否则只需要一张就够了。图6.10给出了固定长度子网划分的简化模块。

1. 模块提取分组的目的地址。
2. 如果这个目的地址与路由表中任意一个特定主机表项相匹配，那么就从表中提取出下一跳地址和接口号。
3. 通过使用目的地址和掩码来提取子网地址。
4. 使用子网地址来搜索路由表，查找下一跳地址和接口号。如果找不到匹配，就使用默认的。
5. 把下一跳地址和接口号传送给ARP（参见第8章）。

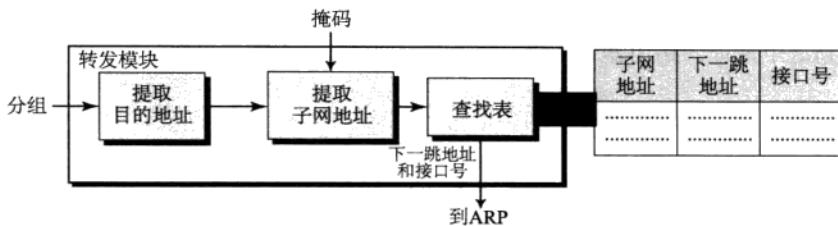


图6.10 有子网划分时分类编址的简化转发模块

例6.4

图6.11给出了连接四个子网的一台路由器。

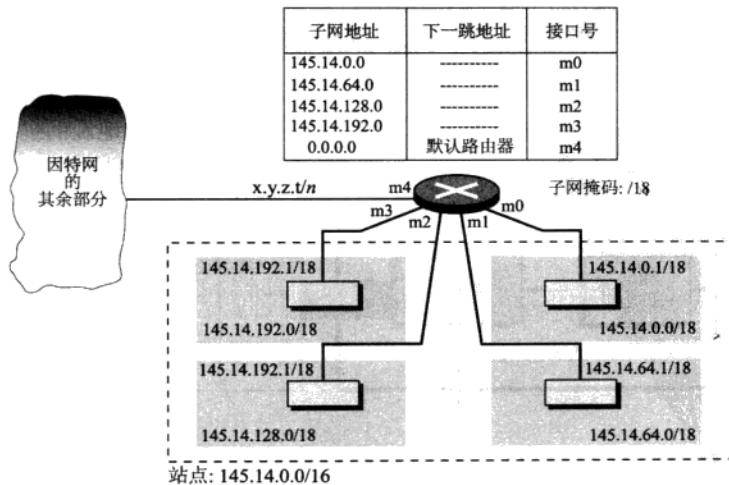


图6.11 例6.4的配置

我们应当注意以下几点。首先，这个站点的地址是 145.14.0.0/16（一个 B 类地址）。目的地址在 145.14.0.0 到 145.14.255.255 之间的每一个分组都要交付到接口 m4，并由这个路由器分发到最后的目的子网。第二，我们用地址 $x.y.z.t/n$ 表示接口 m4，因为我们不知道这个路由器连接到哪个网络。第三，这张表有一个默认表项，用于要从这个站点发送出去的分组。这个路由器被配置为对任何目的地址都使用子网掩码/18。

例 6.5

图 6.11 中的路由器收到一个目的地址为 145.14.32.78 的分组。试说明该分组是怎样被转发的。

解

掩码是/18。在应用这个掩码后，得出的子网地址是 145.14.0.0。然后把这个分组连同下一跳地址 145.14.32.78 和外出接口 m0 一起传递给 ARP（参见第 8 章）。

例 6.6

在图 6.11 中，网络 145.14.0.0 上有一台主机要把一个分组发送给地址为 7.22.67.91 的主机。试说明该分组是怎样被转发的。

解

路由器接收这个分组，并应用掩码 (/18)，得到网络地址是 7.22.64.0。搜索路由表，但没有找到这个地址。路由器使用默认路由器的地址（没有标注在图中）将分组发往那个路由器。

使用无分类编址时的转发

在使用无分类编址时，整个地址空间是完整的，没有划分类别。这就表示在转发时要求对每个涉及到的地址块都必须在路由表中有一行相应的信息，并根据网络地址（地址块的第一个地址）对这张路由表进行查找。不幸的是，从分组的目的地址中得不到有关网络地址的线索（而在分类编址时就可以找到）。

为了解决这个问题，我们需要在路由表中包含掩码 ($/n$)。我们需要有另外一列，它包含对应的地址块的掩码。换言之，虽然一个分类编址的路由表可以设计为三列，但无分类编址的路由表至少需要四列。

分类编址的路由表可以设计为三列，但无分类编址的路由表至少需要四列。

图 6.12 描绘了无分类编址使用的简化转发模块。请注意，网络地址的提取是在查表的同时完成的，因为目的地址本身没有什么可用来提取网络地址的信息。

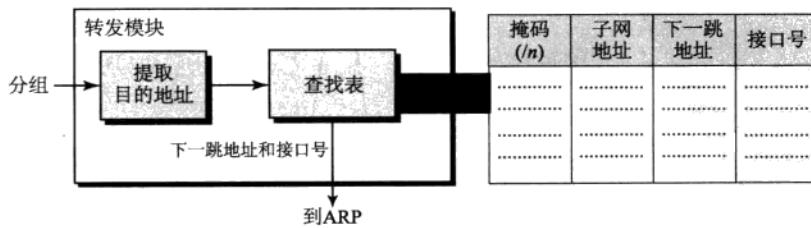


图 6.12 无分类编址简化的转发模块

例 6.7

试用图 6.13 的配置为路由器 R1 制作一张路由表。

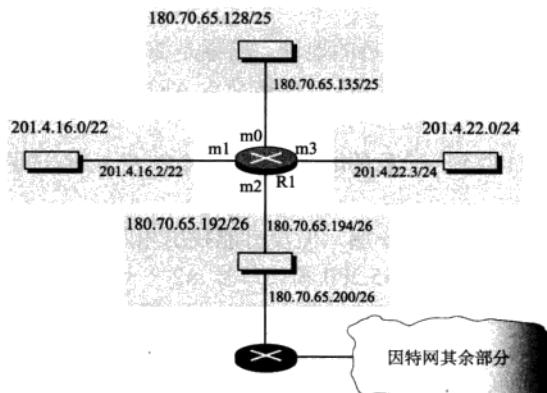


图 6.13 例 6.7 的配置

解

表 6.1 给出了相应的路由表。

表 6.1 图 6.13 中的路由器 R1 的路由表

掩码	网络地址	下一跳地址	接口
/26	180.70.65.192	-	m2
/25	180.70.65.128	-	m0
/24	201.4.22.0	-	m3
/22	201.4.16.0	...	m1
默认	默认	180.70.65.200	m2

例 6.8

图 6.13 中的路由器 R1 收到一个目的地址是 180.70.65.140 的分组，试说明该分组的转发过程。

解

路由器执行以下的步骤：

- 把第一个掩码 (/26) 应用到目的地址，结果得到 180.70.65.128，它与对应的网络地址不匹配。
- 把第二个掩码 (/25) 应用到目的地址，结果得到 180.70.65.128，它与对应的网络地址匹配，于是将下一跳地址（本例中就是分组的目的地址）和口号 m0 一起传递给 ARP（参见第 8 章）做进一步处理。

例 6.9

图 6.13 中路由器 R1 收到一个目的地址是 201.4.22.35 的分组。试说明该分组的转发过程。

解

路由器执行以下的步骤：

- 把第一个掩码 (/26) 应用到目的地址，结果是 201.4.22.0，它与对应的网络地址（第

一行) 不匹配。

2. 把第二个掩码 (/25) 应用到目的地址, 结果是 201.4.22.0, 它与对应的网络地址 (第二行) 不匹配。

3. 把第三个掩码 (/24) 应用到目的地址, 结果是 201.4.22.0, 它与对应的网络地址相匹配。这个分组的目的地址和接口号 m3 一起被交给 ARP。

例 6.10

图 6.13 中的路由器 R1 收到一个目的地址是 18.24.32.78 的分组。试说明该分组的转发过程。

解

这一次把所有的掩码都应用了一遍, 但没有能找出相匹配的网络地址。在搜索到表的末尾时, 模块将下一跳地址 180.70.65.200 和接口号 m2 一起递交给 ARP。这或许是一个需要通过默认路由器向外传送到因特网其他某个地方的数据包。

例 6.11

现在让我们举一个另一种类型的例子。如果我们只知道一个路由器的路由表, 能否得出它的配置? 表 6.2 给出了路由器 R1 的路由表。我们能够画出它的拓扑图吗?

表 6.2 例 6.11 的路由表

掩码	网络地址	下一跳	接口
/26	140.6.12.64	180.14.2.5	m2
/24	130.4.8.0	190.17.6.2	m1
/16	110.70.0.0	...	m0
/16	180.14.0.0	...	m2
/16	190.17.0.0	...	m1
默认	默认	110.70.4.6	m0

解

我们知道一些事实, 但还不足以给出一个完整且确定的拓扑图。我们知道路由器 R1 有三个接口: m0、m1 和 m2。我们也知道有三个网络直接连接到路由器 R1。我们还知道有两个网络间接地连接到路由器 R1。在这里至少还有另外三个路由器被涉及到 (见下一跳这一列)。通过查看这些路由器的 IP 地址, 我们可以知道它们分别连接到哪些网络。这样, 我们就可以把这些路由器放到适当的位置上。我们知道有一个默认路由器连接到因特网的其余部分。但是还缺少一些信息。我们不知道网络 130.4.8.0 是否直接连接到路由器 R2, 还是经过了点对点的网络 (广域网) 和其他的路由器连接到 R2。我们不知道网络 140.6.12.64 是否直接连接到路由器 R3, 还是通过了点对点网络 (广域网) 和其他的路由器连接到 R3。点对点网络通常在路由表中没有表项, 因为没有主机连接到它们。图 6.14 给出了猜测的拓扑。

地址聚合 当我们使用分类编址时, 一个组织之外的每一个站点在路由表中只有一个表项。这个表项定义了一个对应的站点, 而不管该站点是否又划分了子网。当分组到达路由器时, 路由器检查相应的表项并转发这个分组。当我们使用无分类编址时, 路由表的表项数量很可能会增加, 这是因为无分类编址的意图就是把整个的地址空间划分为很多个可管理的地址块。路由表的规模变大就会导致搜索路由表的时间变长。为了解决这个问题, 人们设计了地址聚合 (address aggregation) 的概念。在图 6.15 中, 我们有两个路由器。

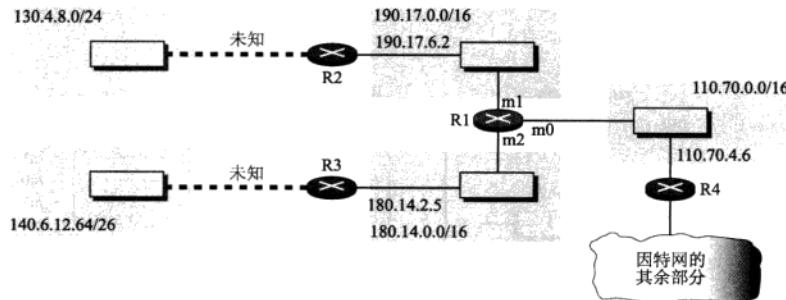
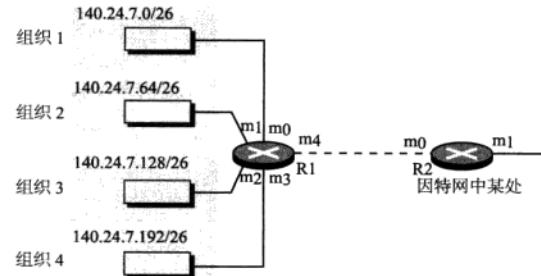


图 6.14 例 6.11 中猜测出的拓扑



掩码	网络地址	下一跳地址	接口号
/26	140.24.7.0	-----	m0
/26	140.24.7.64	-----	m1
/26	140.24.7.128	-----	m2
/26	140.24.7.192	-----	m3
/0	0.0.0.0	默认路由器	m4

掩码	网络地址	下一跳地址	接口号
/24	140.24.7.0	-----	m0
/0	0.0.0.0	默认路由器	m1

R2的路由表

图 6.15 地址聚合

R1 连接了四个组织的网络，每个网络使用 64 个地址。R2 在远离 R1 的某处。R1 的路由表比较长，因为每一个分组必须被正确地转发到相应的组织。但是另一方面，R2 的路由表就很短。对于 R2 来说，任何目的地址在 140.24.7.0~140.24.7.255 之间的分组，都要从接口 m0 发送出去，不管发送给哪个组织的，这就称为地址聚合，因为分属于四个组织的地址块被聚合成为一个更大的地址块。如果每个组织的地址不能被聚合到一个地址块中，那么 R2 就需要一个更长的路由表。

请注意，虽然地址聚合的概念和子网划分的概念相似，但这里我们并没有一个共同的站点，也就是说每一个组织的网络是独立的。另外，我们还可以有多级聚合。

最长掩码匹配 如果在前面的配置图中有一个组织的地理位置距离其他三个组织比较远，那么会出现什么情况呢？例如，由于某些原因使组织 4 不能连接到路由器 R1，那么我们是否还能使用地址聚合的概念，是否还能把地址块 140.24.7.192/26 指派给组织 4？答案是肯定的，因为无分类编址的路由选择使用了另一个原则，即**最长掩码匹配 (longest mask matching)**。这个原则指出，路由表要按照从最长掩码到最短掩码来排序。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，而/24 必须是最后一个表项。

让我们来看看，这个原则是否可以用来解决组织 4 与其他三个组织分隔所带来的问题。图 6.16 描绘了这种情况。

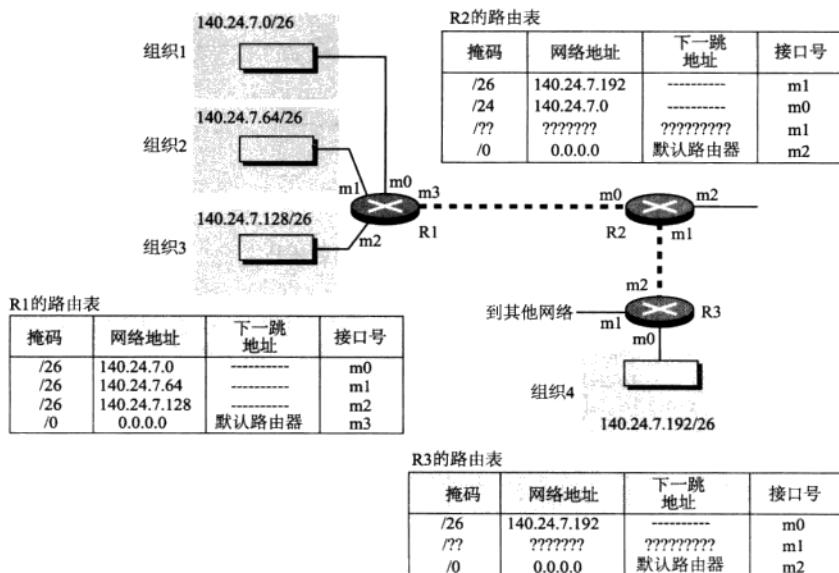


图 6.16 最长掩码匹配

假定有一个目的地址为 140.24.7.200 的分组要去往组织 4。在路由器 R2 上应用第一个掩码，得出网络地址为 140.24.7.192。这个分组通过接口 m1 正确地转发到达组织 4。但是，如果路由表不是把最长前缀放在最前面，那么使用掩码/24 就会导致把分组不正确地转发到路由器 R1。

多级路由选择 要解决路由表过于庞大的问题，我们可以在路由表中建立分等级的概念。在第 1 章中曾提到过，如今的因特网可被视为一种多级结构。我们说因特网被划分为主干 ISP、地区 ISP 和本地 ISP。如果路由表具有像因特网这样的多级结构，那么路由表的长度将会大大缩短。

让我们以一个本地 ISP 为例来说明。本地 ISP 可以被指派一个比较大的，具有一定前缀长度的地址块。再由本地 ISP 把这个地址块划分为若干个大小不同的较小的地址块，并把它们指派给个人用户和组织（有大的也有小的）。如果指派给本地 ISP 的地址块从 a.b.c.d/n 开始，那么该 ISP 就可以创建多个以 e.f.g.h/m 开始的地址块，这里的 m 大于 n，且随不同用户而变化。

这样做为什么会缩短路由表的长度呢？因为因特网的其余部分根本不了解这么具体的划分。对因特网的其余部分来说，这个本地 ISP 的所有用户都被定义为 a.b.c.d/n。发送给这个大地址块中某个地址段的每一个分组，都是要先传送到本地 ISP。对世界上其他地方的每一台路由器来说，这些用户合起来只能有一个表项，它们同属于一个组。当然，到了本地 ISP 内部，路由器必须能够识别这些子块，并能够把这些分组传送到目的用户。如果某个用户是个比较大的组织，那么它还可以用子网划分的方法再创建下一级，并把它的子块划分成为更小的子块（或子子块）。对于无分类编址的路由选择，只要我们遵循无分类编

址的规则，等级的划分是没有限制的。

例 6.12

作为多级路由选择的例子，让我们来讨论一下图 6.17。某地区 ISP 被授权使用从 120.14.64.0 开始的 16 384 个地址。这个地区 ISP 决定把这个地址块划分为四个子块，各有 4096 个地址。其中的三个地址子块指派给三个本地 ISP，并把第二个子块保留为今后使用。请注意，每个子块的掩码都是/20，因为原来的掩码为/18 的地址块被划分成了 4 个子块。

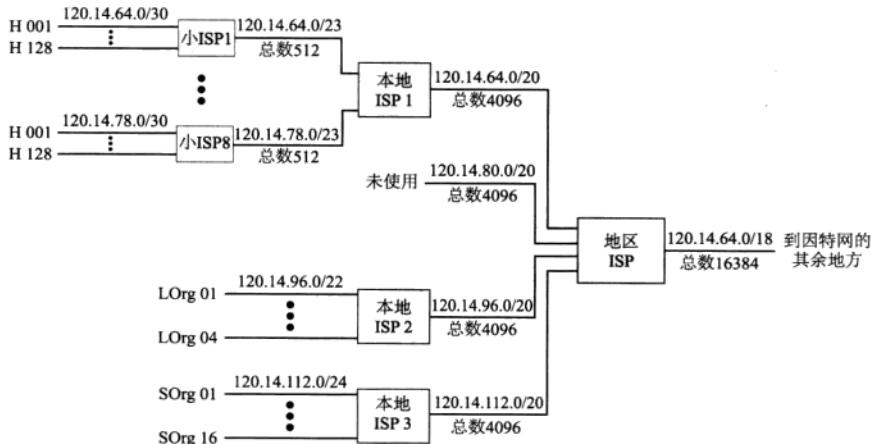


图 6.17 ISP 的分层路由选择

第一个本地 ISP 把它分到的子块再划分为 8 个更小的地址块，并把它们分别指派给 8 个小 ISP。每个小 ISP 可以为 128 个家庭 (H001~H128) 提供服务，每个家庭使用 4 个地址。请注意，此时各个小 ISP 的掩码是/23，因为每个子块被进一步划分成了 8 个块。每个家庭用户的掩码是/30，因为一个家庭用户只有 4 个地址 (2^{32-30} 是 4)。第二个本地 ISP 把分到的子块再划分为 4 个地址块，并把它们分别指派给 4 个较大的组织 (LOrg01~LOrg04)。请注意，每个组织有 1024 个地址，掩码为/22。

第三个本地 ISP 把它分到的子块划分为 16 个地址块，并把它们分别指派给 16 个较小的组织 (SOrg01~SOrg16)。每个较小的组织有 256 个地址，掩码为/24。这个配置里就有分级的概念。因特网的所有路由器都会将目的地址在 120.14.64.0~120.14.127.255 之间的每一个分组发送到这个地区 ISP。而这个地区 ISP 则把目的地址在 120.14.64.0~120.14.79.255 之间的每一个分组都发送到本地 ISP1。本地 ISP1 把目的地址在 120.14.64.0~120.14.64.3 之间的每一个分组都发送到 H001。

地理区域化路由选择 为了进一步减小路由表的长度，我们需要把多级路由选择进一步延伸，以包括地理区域化的路由选择。我们必须首先把整个地址空间划分为很少的几个地址块，并把其中的一个地址块指派给美洲，一块给欧洲，一块给亚洲，一块给非洲，等等。对欧洲以外的路由器来说，所有要发往欧洲的分组在路由表中只需要有一个表项。对美洲以外的路由器来说，所有要发送到北美的分组在路由表中也需要一个表项，依此类推。

路由表的查找算法

分类编址搜索路由表的算法必须做适当的调整，才能使无分类编址的路由选择更加有

效。这里面也包括路由表更新的算法。我们将在第 11 章中讨论路由表的更新问题。

使用分类编址时的查找 在使用分类编址时，路由表被组织成列表的形式。但是，为了使搜索更加高效，路由表被划分为三张表（有时也称为存储桶），每张表分别对应于一个类别。当分组到达时，路由器就应用默认掩码（默认掩码是地址本身所固有的）找出对应的存储桶（A 类、B 类或 C 类）。然后，路由器对相应的存储桶进行搜索，而不是搜索完整的路由表。有些路由器甚至还根据类别搜索过程的结果，为 A 类指派 8 个存储桶，为 B 类指派 4 个存储桶，为 C 类指派 2 个存储桶。

在无分类编址中的查找 在使用无分类编址时，目的地址中没有网络的信息。最简单的，但不是最有效的搜索方法称为最长前缀匹配（如我们在前面讨论的）。路由表可以划分为若干个存储桶，每个存储桶对应于一个前缀。路由器首先尝试在最长前缀的存储桶中搜索。如果在这个存储桶中找到目的地址，查找过程就完成了。如果这个地址没有找到，就搜索下一个前缀的存储桶，依此类推。显然，这种类型的查找需要较长的时间。

一种解决办法是改变查找的数据结构。可以不使用列表，而是使用其他的数据结构（例如，树或二叉树）。有一种可选的结构叫线索（trie，一种特殊的树）。不过关于线索的讨论已经超出了本书的范围。

6.2.2 基于标记的转发

在 20 世纪 80 年代，有一股力量开始或多或少地要将 IP 朝着面向连接的协议方向努力，其中的路由选择被交换所取代。正如我们在第 4 章中讨论的，在一个无连接的网络（数据报方式）中，路由器根据分组首部的目的地址来转发该分组。另一方面，在一个面向连接的网络（虚电路方式）中，交换机则根据附加在分组上的标记来转发该分组。路由选择通常基于对路由表内容的搜索，而交换则可以用一个索引访问交换表来完成。换言之，路由选择涉及到搜索过程，而交换则涉及直接的访问。

例 6.13

图 6.18 所示为一个用最长匹配算法来查找路由表的简单例子。虽然现在有些算法更加高效，但基本原理都是一样的。

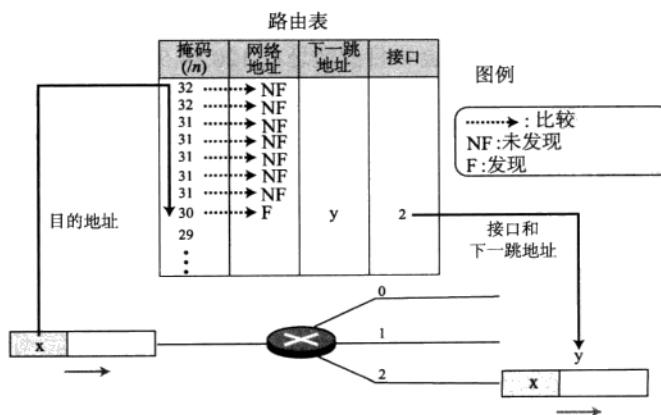


图 6.18 例 6.13 基于目的地址的转发

当转发算法得到分组的目的地址后，它就需要埋头在掩码列中开始搜索。对于每一个表项，它都要应用掩码来求出分组的目的网络地址，然后再查看路由表中的网络地址，直到找到相匹配的网络地址。然后路由器提取出要传递给 ARP 协议的下一跳地址和接口号，以便将该分组交付到下一跳。

例 6.14

图 6.19 所示为使用标记来访问一张交换表的简单例子。因为标记被当作是交换表的索引，所以可以立刻在表中找到相应的信息。

MPLS

在 20 世纪 80 年代期间，有几个生产商新研制的路由器应用了交换技术。后来，IETF 批准了一个标准，称为多协议标记交换（Multi-Protocol Label Switching，MPLS）。在这个标准中，因特网的有些传统路由器可以被 MPLS 路由器所取代，这些 MPLS 路由器既可用作路由器，也可用作交换机。当 MPLS 被当作路由器使用时，它可以根据目的地址来转发分组，而当它被当作交换机使用时，又能够基于标记来转发分组。

一个新的首部

要使用像 IP 这样的协议来模仿面向连接的交换，首先我们要做的事情就是为分组增加一个字段，以便携带标记（在第 4 章中讨论过）。IPv4 分组格式不允许此类扩展（虽然在 IPv6 分组格式中已提供了这个字段，我们将在第 27 章讨论）。解决办法就是将 IPv4 分组封装在一个 MPLS 分组中（就好像 MPLS 是位于数据链路层和网络层之间的又一个层）。完整的 IP 分组作为净负荷被封装到一个 MPLS 分组中，并附加上 MPLS 首部。图 6.20 所示为此封装过程。

这个 MPLS 首部实际上就是一个用于多级交换的子首部的堆栈，我们稍后会讨论。图 6.21 描绘了一个 MPLS 首部的格式，其中每个子首部是 32 位（4 字节）长。

以下是对每个字段的简单描述：

- **标记** 这个 20 位的字段定义的就是标记，它在路由器中被用作路由表的索引。
- **Exp** 这个 3 位的字段被保留作为实验用。
- **S** 这个 1 位的堆栈字段定义了堆栈中的子首部的状态。当它是 1 时，表示这个子首部是堆栈中的最后一个。
- **TTL** 这个 8 位的字段与 IP 数据报中的 TTL 字段类似（参见第 7 章）。每经过一个路由器这个字段的值就会递减 1，当它的值减至 0 时，这个分组就被丢弃，以防止出现循环。

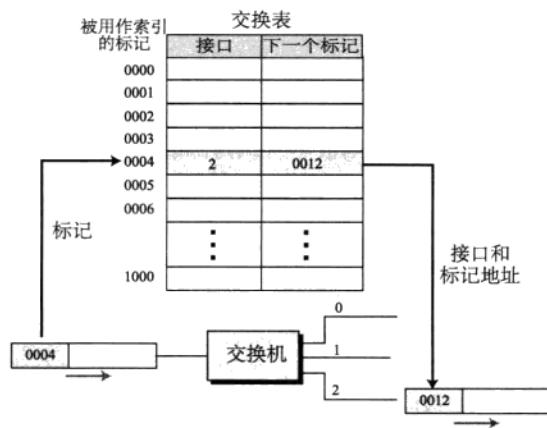


图 6.19 例 6.14 基于标记的转发

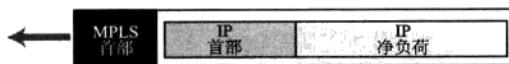


图 6.20 在一个 IP 分组上添加 MPLS 首部

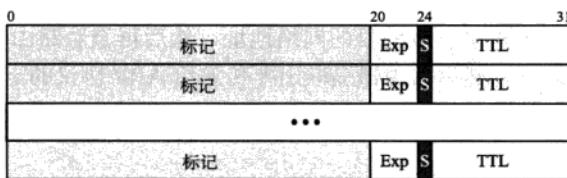


图 6.21 MPLS 的首部由标记的堆栈组成

多级交换

MPLS 中的标记堆栈使得多级交换成为可能。这与传统的多级路由选择类似。例如，有一个分组携带了两个标记，上面的那个标记可在本组织之外的交换机上作为转发时的依据，而下面的标记则被本组织内部的路由器用来为这个分组选择路由，以到达它的目的子网。

6.3 路由器的结构

当我们讨论转发和路由选择时，我们把路由器画成了一个黑盒子，它从一个输入端口（接口）接受进入的分组，再通过路由表找出分组离开的输出端口，然后把分组从输出端口发送出去。在这一节，我们将要打开这个黑盒子，并观察其内部。不过，我们的讨论不会很详细，有几本书是专门讨论路由器的，而我们仅仅是给读者一个大体的介绍。

6.3.1 构件

我们可以说，一个路由器有四个构件：输入端口（input ports）、输出端口（output ports）、路由选择处理器（routing processor）以及交换结构（switching fabric），如图 6.22 所示。

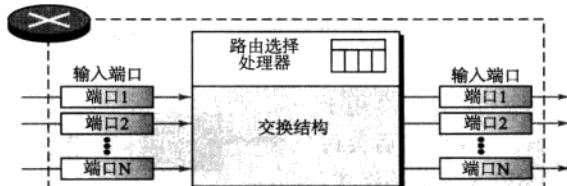


图 6.22 路由器的构件

输入端口

图 6.23 所示为输入端口的概要图。

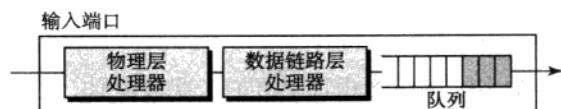


图 6.23 输入端口

输入端口执行路由器的物理层和数据链路层的功能。它从接收到的信号中得到比特流，并把帧拆装后得到分组，同时还要进行差错的检测和纠正。分组准备就绪后，就可以通过网络层进行转发。除了物理层处理器和数据链路层处理器之外，输入端口还有一些缓存（队列），用来在分组被传送到交换结构之前进行暂存。

输出端口

输出端口执行的功能与输入端口一样，但是顺序相反。首先，向外发送的分组要进行排队，然后将分组封装成帧，最后，对这些帧使用物理层的功能，把它们变成发送到线路上去的信号。图 6.24 所示为输出端口的概要图。

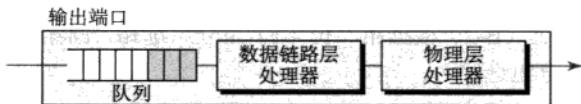


图 6.24 输出端口

路由选择处理器

路由选择处理器执行网络层的功能。它用目的地址来找出下一跳地址，与此同时，还要找出将分组发送出去的口号。这个动作有时称为查表，因为路由选择处理器需要对路由表进行搜索。在一些较新的路由器中，路由选择处理器的这个功能被转移到输入端口，为的是使这个过程能够更容易和更快地进行。

交换结构

在路由器中最困难的任务就是把分组从输入队列搬到输出队列中。完成这件事的速度直接影响到输入/输出队列的大小和分组交付的总延时。在过去，当路由器实际上是一个专用的计算机时，这台计算机的存储器或总线被用作交换结构。输入端口把分组放到存储器中，输出端口从存储器中得到分组。如今的路由器使用了多种多样的交换结构。这里我们要简单地讨论其中一些结构。

纵横交换结构 最简单的交换结构就是如图 6.25 所示的纵横交换结构。纵横交换结构（crossbar switch）把 n 个输入和 n 个输出连接成一个栅格，在每一个交叉点（crosspoint）使用了电子微开关。

榕树交换结构 比纵横交换结构更现实一些的是榕树交换结构（banyan switch，以榕树命名），如图 6.26 所示。

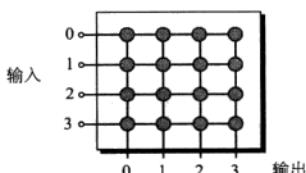


图 6.25 纵横交换结构

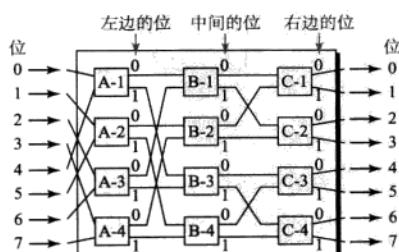


图 6.26 榕树交换结构

榕树交换结构是一种多级交换结构，每一级都有许多微交换单元，可以根据以二进制字符串表示的输出端口号来为分组选路。对于 n 个输入和 n 个输出，我们共有 $\log_2(n)$ 级，每个级有 $n/2$ 个微交换单元。第一级要根据二进制字符串的最高位为分组选路。第二级根据第二个最高位为分组选路，依此类推。图 6.27 所示为一个具有 8 个输入和 8 个输出的榕树交换结构。它的级数是 $\log_2(8) = 3$ 。假设有一个分组从输入端口 1 到达，并且要从输出端口 6（二进制的 110）发送出去，那么第一个微交换单元（A-2）根据第一位（1）来为分组选路，第二个微交换单元（B-4）根据第二位（1）为分组选路，第三个微交换单元（C-4）根据第三位（0）为分组选路。在图 6.27 (b) 中，一个分组从输入端口 5 到达，并且要从输出端口 2（二进制的 010）发送出去。第一个微交换单元（A-2）根据第一位（0）为分组选路，第二个微交换单元（B-2）根据第二位（1）为分组选路，而第三个微交换单元（C-2）则根据第三位（0）为分组选路。

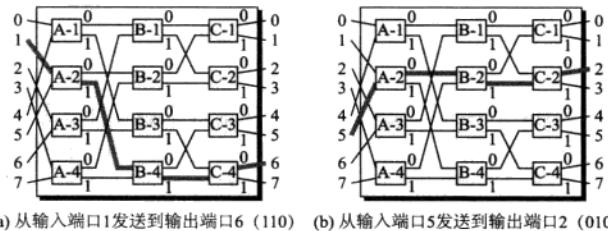


图 6.27 在榕树交换结构里选路的例子

Batcher 榕树交换结构 榕树交换结构存在的问题是有可能会出现内部碰撞，哪怕两个分组并不是要到相同的输出端口去。解决这个问题的方法是对到达的分组根据它们的目的端口进行排序。K. E. Batcher 设计的交换结构被放置在榕树交换结构的前端，它的作用就是根据入口分组的目的地址对这些分组进行排序。这种组合称为 **Batcher 榕树交换结构** (Batcher-banyan switch)，如图 6.28 所示。

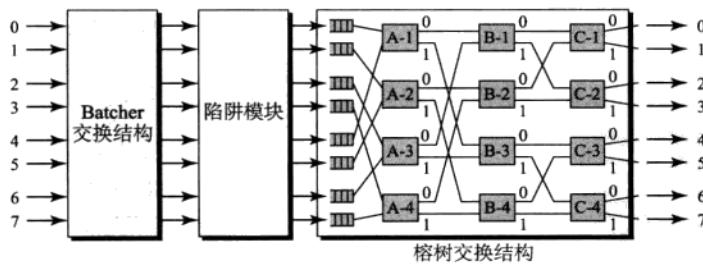


图 6.28 Batcher 榕树交换结构

这种排序交换结构使用了硬件融合技术，不过我们在此不详细讨论它。通常，另外另一个称为陷阱的硬件模块被加在 Batcher 交换结构和榕树交换结构之间。陷阱模块用来防止重复的分组（具有相同输出终点的分组）同时通过榕树交换结构。在每一个滴答 (tick)，对每一个终点只允许有一个分组发送。如果发送分组超过一个，那它们就必须等到下一个滴答。

6.4 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

6.4.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]、[Kur & Ros 08]。

6.4.2 RFC

讨论转发的有 RFC 1812、RFC 1971 和 RFC 1980。讨论 MPLS 的有 RFC 3031、RFC 3032、RFC 3036 和 RFC 3212。

6.5 重要术语

榕树交换结构	间接交付
Batcher 榕树交换结构	输入端口
纵横交换结构	最长掩码匹配
交叉点	最长前缀匹配
默认方法	特定网络方法
交付	下一跳地址
直接交付	下一跳方法
转发	输出端口
特定主机方法	路由选择处理器

6.6 本章小结

- 网络层监视底层各物理网络对分组的处理过程，我们把这种处理定义为分组的交付。如果分组的交付者（主机或路由器）与终点连接在同一个网络上，则分组的交付就称为直接的。如果分组的交付者（主机或路由器）与终点连接在不同的网络上，则分组的交付就称为间接的。
- 转发是指将分组交付到下一跳。本章讨论了两种类型的转发：基于 IP 数据报目的地址的转发和基于附加在 IP 数据报上的标记的转发。第一种类型为了转发分组必须搜索路由表，而第二种类型在转发分组时会把标记当作交换表的索引。

- 我们讨论了几种基于目的地址的转发方法，包括特定主机方法、下一跳方法、特定网络方法和默认方法。
- 在基于目的地址的转发中，分类编址时的路由表可以只有三列，而无分类编址时的路由表则至少需要四列。地址聚合简化了无分类编址时的转发过程，并且在无分类编址时要求用最长掩码匹配。
- 基于标记的转发使用的是交换表而不是路由表。多协议标记交换（MPLS）是经过 IETF 批准的标准，它把 IP 分组封装到 MPLS 分组中，从而给 TCP/IP 协议族增加了一个伪层。
- 路由器通常由四个构件组成：输入端口、输出端口、路由选择处理器和交换结构。

6.7 实践安排

6.7.1 习题

1. IP 地址为 137.23.56.23/16 的主机把分组发送给 IP 地址为 137.23.67.9/16 的主机。试问这是直接交付还是间接交付？假定无子网划分。
2. IP 地址为 137.23.56.23/16 的主机把分组发送给 IP 地址为 142.3.6.9/24 的主机。试问这是直接交付还是间接交付？假定无子网划分。
3. 试写出图 6.8 中路由器 R2 的路由表。
4. 试写出图 6.8 中路由器 R3 的路由表。
5. 在图 6.8 中，目的地址为 192.16.7.42 的分组到达路由器 R1。试说明该分组将怎样被转发。
6. 在图 6.8 中，目的地址为 145.80.14.26 的分组到达路由器 R1。试说明该分组将怎样被转发。
7. 在图 6.8 中，目的地址为 147.26.50.30 的分组到达路由器 R1。试说明该分组将怎样被转发。
8. 在图 6.11 中，目的地址为 145.14.192.71 的分组到达路由器。试说明该分组将怎样被转发。
9. 在图 6.11 中，目的地址为 135.11.80.21 的分组到达路由器。试说明该分组将怎样被转发。
10. 在图 6.13 中，目的地址为 201.4.16.70 的分组到达路由器 R1。试说明该分组将怎样被转发。
11. 在图 6.13 中，目的地址为 202.70.20.30 的分组到达路由器 R1。试说明该分组将怎样被转发。
12. 试给出一个完全孤立的主机的路由表。
13. 试给出一个连接到局域网但没有连接到因特网的主机的路由表。
14. 如果表 6.3 是路由器 R1 的路由表，试给出该网络的拓扑图。

表 6.3 习题 14 的路由表

掩码	网络地址	下一跳地址	接口
/27	202.14.17.224	---	m1
/18	145.23.192.0	---	m0
默认	默认	130.56.12.4	m2

15. 图 6.16 中的路由器 R1 能够收到目的地址为 140.24.7.194 的分组吗？为什么？
16. 图 6.16 中的路由器 R1 能够收到目的地址为 140.24.7.42 的分组吗？为什么？
17. 试给出图 6.17 中的地区 ISP 的路由表。
18. 试给出图 6.17 中的本地 ISP1 的路由表。
19. 试给出图 6.17 中的本地 ISP2 的路由表。
20. 试给出图 6.17 中的本地 ISP3 的路由表。
21. 试给出图 6.17 中的小 ISP1 的路由表。

6.7.2 研究活动

22. 说明一个 MPLS 分组是如何被封装到一个帧中的。如果使用的是以太网协议，找出此时类型字段的值应当是什么。
23. 把 Cisco Systems 公司使用的多层交换机（Multi-Layer Switching, MLS）与我们在这里描述的 MPLS 技术进行比较。
24. 有些人认为 MPLS 应当被称为 TCP/IP 协议族的第 2.5 层。你同意吗？为什么？
25. 你的 ISP 是如何使用地址聚合和最长掩码匹配的？
26. 你的 IP 地址是否也是地理区域化地址分配中的一部分？
27. 如果你使用了一台路由器，试找出其路由表中的列数和每一列的名称。
28. Cisco 是路由器的主流生产厂家之一。试找出由这个公司生产的不同类型路由器的有关信息。

第 7 章 网际协议版本 4 (IPv4)

在讨论了 IP 编址机制和 IP 分组的交付与转发之后，我们在这一章要讨论的是 IP 分组的格式。我们将说明 IP 分组如何由一个基本首部和一些选项构成，这些选项有时用于促进和控制分组的交付。

目标

本章有以下几个目标：

- 解释 IP 协议背后的总体思想，并说明 IP 与 TCP/IP 协议族中其他协议之间的位置关系。
- 说明 IPv4 数据报的一般格式，并列出首部中的各个字段。
- 讨论数据报的分片和重装，以及如何从数据报的分片中恢复原始的数据报。
- 讨论 IPv4 数据报中可能存在的一些选项以及它们的应用。
- 说明发送方站点如何为 IPv4 数据报的首部计算检验和，以及接收方站点又是如何检查这个检验和的。
- 讨论 IP 在 ATM 上运行，并与建立在局域网或点对点广域网之上的 IP 进行比较。
- 描述一个简化版本的 IP 软件包，并给出某些模块的伪码。

7.1 引言

网际协议（Internet Protocol, IP）是 TCP/IP 协议族在网络层使用的传输机制。图 7.1 给出了 IP 在这个协议族中的位置。

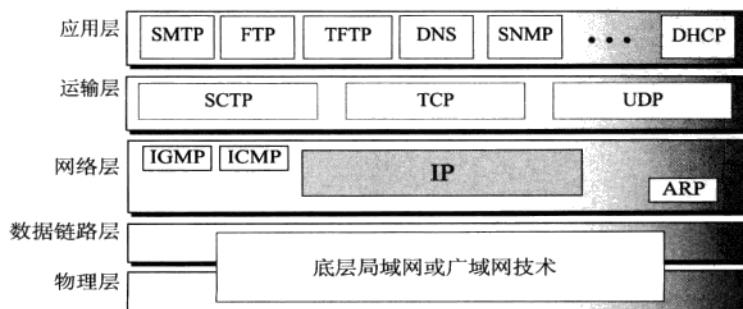


图 7.1 IP 在 TCP/IP 协议族中的位置

IP是一种不可靠的无连接数据报协议——一种尽最大努力交付(best-effort delivery)的服务。尽最大努力一词的意思是IP分组有可能损坏、丢失、失序或延迟到达，并且有可能会给网络带来拥塞。

如果可靠性很重要，那么IP就必须与可靠的协议（如TCP）配合起来使用。理解尽最大努力服务的一个日常生活中的例子就是邮政局。邮政局尽最大努力交付邮件，但并不总是能够成功。如果一封非挂号信丢失了，那只能由发信人或预期的收信人来发现信件的丢失，并采用补救措施。邮政局本身并不对每一封信进行跟踪，也不通知发信人有关信件的丢失或损坏情况。

IP也是分组交换网络中使用数据报方式（见第4章）的一种无连接协议。这就意味着每个数据报被独立处理，且各数据报可能沿不同的路由传送到终点。同时也暗示如果一个源点向同一个终点发送了多个数据报，那么这些数据报有可能不按顺序到达。同时，有些数据报在传输过程中可能会受到损伤或丢失。再次强调，IP依靠更高层的协议来解决所有这些问题。

7.2 数据报

网络（互联网）层的分组称为数据报(datagram)。图7.2给出了IP数据报的格式。数据报是一个可变长度的分组，它由两部分组成：首部和数据。首部的长度可以是20~60字节，包含有关路由选择和交付的重要信息。习惯上，在TCP/IP中都是以4个字节为一段来表示首部。下面按顺序简单地介绍各个字段。

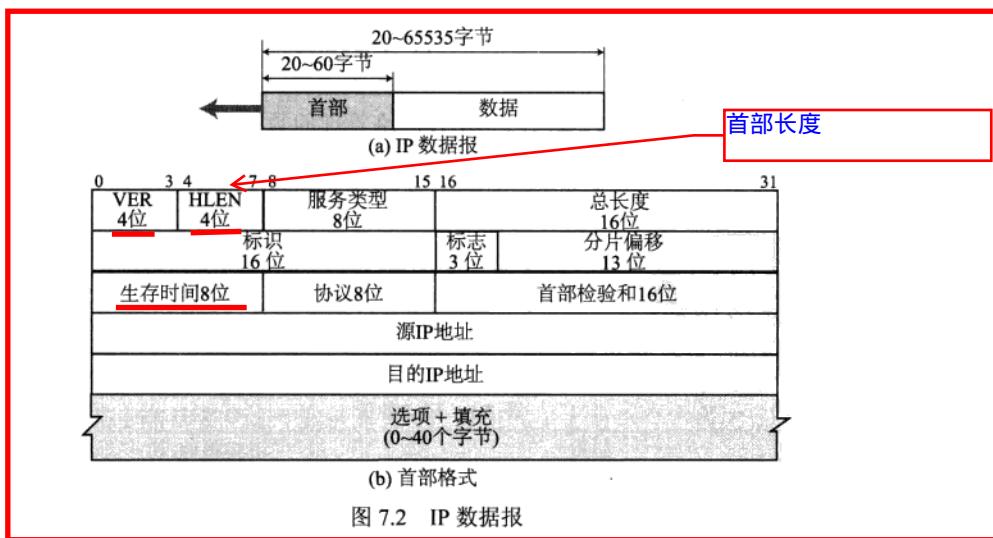


图7.2 IP数据报

- 版本(VER) 这个4位字段定义了IP协议的版本。目前的版本是4。但是将来版本6(或IPng)很有可能会取代版本4。这个字段向处理机上运行的IP软件指出该IP数据报使用的是版本4的格式。所有字段都要按照版本4协议所规定的来解释。如果机器使用其他版本的IP，那么这个数据报就会被丢弃，而不是错误地进行解释。

- **首部长度 (HLEN)** 这个 4 位字段定义了数据报首部的总长度，以 4 字节的字为单位计算。这个字段是必要的，因为首部的长度是可变的（在 20~60 字节之间）。在没有选项时，首部长度 (header length) 是 20 字节，且这个字段的值是 5 ($5 \times 4 = 20$)。当选项字段为最大值时，这个字段的值是 15 ($15 \times 4 = 60$)。
- **服务类型** 在最初设计 IP 首部时，这个字段称为服务类型 (type of service, TOS)，它指明了应当如何处理数据报。这个字段中有一部分用于定义数据报的优先级，剩下的一部分定义了服务类型 (低时延，高吞吐量等等)。IETF 已经改变了这个 8 位字段的解释，现在它定义了一组区分服务 (differentiated services)。我们在图 7.3 中给出了最新的解释。

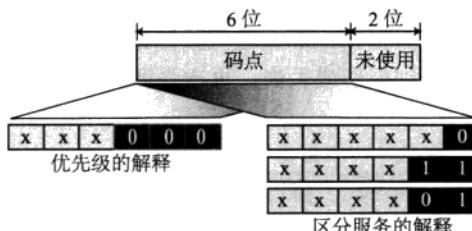


图 7.3 服务类型

在这种解释中，前 6 位构成了码点 (codepoint) 子字段，而最后两个位未使用。码点子字段可用于两种不同的方式。

- 当最右边的 3 位为 0 时，对最左边 3 位的解释就如同服务类型解释中的优先级位一样。换言之，就是与原先的解释兼容。这个优先级定义了在出现一些问题（如拥塞）时数据报的优先级。当路由器因出现拥塞而必须丢弃一些数据报时，具有最低优先值的数据报将首先被丢弃。在因特网中有一些数据报比其他一些更为重要。例如，用来进行网络管理的数据报要比向一组人发送可选信息的一般数据报紧迫得多和重要得多。
- 当最右边的 3 位并非全都为 0 时，这 6 位就定义了 54 ($64 - 8$) 种服务，它们是由因特网或本地管理机构根据表 7.1 所列出的指派优先级而指派的。第一类包含 24 种服务，第二类和第三类各包含 16 种服务。第一类由因特网管理机构 (IETF) 指派。第二类可以被本地管理机构 (组织) 使用。第三类是临时性的，可用于试验目的。请注意，这些指派还没有最后完成。

表 7.1 码点的值

类别	码点	指派机构
1	XXXXX0	因特网
2	XXXX11	本地
3	XXXX01	临时的或试验的

- **总长度** 这个 16 位字段定义了以字节为单位的数据报总长度 (首部加上数据)。要找出从上层传送来的数据的长度，可以用总长度减去首部长度。把 HLEN 字段的值

乘以4就可得出首部长度。

$$\text{数据长度} = \text{总长度} - \text{首部长度}$$

因为总长度字段是16位，因此IP数据报长度限制在65535（即 $2^{16}-1$ ）字节，其中首部占20~60字节，剩下的是从上层传送来的数据。

总长度字段定义了包括首部在内的**数据报总长度**。

虽然65535字节的长度看起来好像很长，但是，随着底层技术使得更大的吞吐量（更高带宽）成为可能，IP数据报的长度在不久的将来可能会大大增加。当我们在下一节讨论分片时，我们将会看到某些物理网络不能把65535字节的数据报封装成它们的帧。要通过这些网络就必须把这样的数据报进行分片。

读者可能会问为什么我们需要这个字段。当一个机器（路由器或主机）接收到一个帧时，它剥去首部和尾部后就剩下这个数据报了，那么为什么还要使用一个并不需要的字段呢？答案是：在许多情况下，我们的确不需要这个字段的值。但在有些情况下，封装在一个帧里的并不仅仅是数据报，还可能附加了一些填充。例如，以太网协议对能够封装在一个帧里的数据有最小值和最大值的限制（46~1500字节）。当一个IP数据报的长度小于46字节时，必须增加一些填充字节才能满足这个要求。在这种情况下，当机器对这个数据报进行拆装时，必须检查总长度字段，以便确定数据的真正长度和填充字节的长度（见图7.4）。

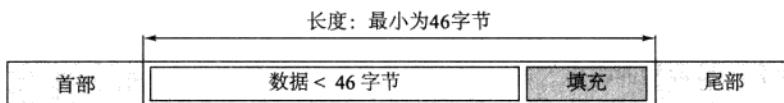


图7.4 把一个小的数据报封装在以太网帧中

- 标识** 这个字段用于分片（在下一节讨论）。
- 标志** 这个字段用于分片（在下一节讨论）。
- 分片偏移** 这个字段用于分片（在下一节讨论）。
- 生存时间** 数据报在互联网中的旅程是有生存时间限制的。这个字段的最初设计是存放一个时间戳，并由经过的每一个路由器进行递减。当时间戳的值变为0时就丢弃这个数据报。但是要做到这点，所有的机器必须是同步的，而且还必须知道数据报从一个机器到另一个机器所花费的时间。现在，这个字段绝大多数用来控制数据报所经过的最大跳数（路由器）。当源主机发送数据报时，它在这个字段存入一个数值。这个数值大约是任意两台主机之间的最大路由器数的两倍。每一个处理此数据报的路由器都要把这个数值递减1。若在减1之后这个字段的值变成了0，路由器就丢弃这个数据报。

这个字段之所以是必要的，是因为因特网中的路由表可能会损坏。一个数据报可能在两个或多个路由器之间逗留了很长时间，也没有能够交付到目的主机。这个字段限制了数据报的生存时间。

这个字段的另一个用途就是源主机想故意限制这个分组的行程。例如，如果源主机打算把分组限制在局域网的范围内，就可以把这个字段的值设置为1。当分组到达第一个路由器时，这个值就减为0，因而数据报就被丢弃了。

□ 协议 这个 8 位字段定义了使用此 IP 层服务的高层协议。有许多高层协议（诸如 TCP, UDP, ICMP 和 IGMP 等）的数据都能够被封装到 IP 数据报中。这个字段指明 IP 数据报必须交付给哪个最终目的协议。换言之，因为 IP 协议要对来自不同高层协议的数据进行复用和分用，所以当数据报到达最后的终点时，要用到这个字段的值来完成分用过程（见图 7.5）。

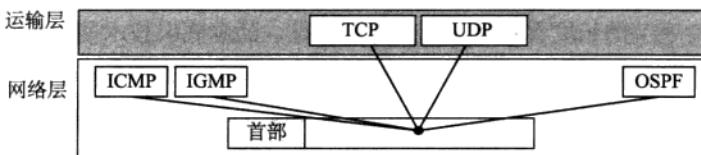


图 7.5 复用

对不同的高层协议，这个字段的一些值如表 7.2 所示。

表 7.2 协议

值	协议	值	协议
1	ICMP	17	UDP
2	IGMP	89	OSPF
6	TCP		

- 检验和** 检验和的概念及其计算将在本章的后面部分讨论。
- 源地址** 这个 32 位字段定义了源点的 IP 地址。在 IP 数据报从源主机发送到目的主机的过程中，这个字段始终保持不变。
- 目的地址** 这个 32 位字段定义了终点的 IP 地址。在 IP 数据报从源主机发送到目的主机的过程中，这个字段始终保持不变。

例 7.1

有一个到达分组的前 8 位如下所示：

01000010

接收者丢弃了这个分组。为什么？

解

在这个分组中出现了差错。**最左边的 4 位 (0100) 给出了版本**，这是正确的。接下去的 4 位 (0010) 给出了一个错误的**首部长度 ($2 \times 4 = 8$)**，因为首部的最小字节数是 20。这个分组在传输中已经受到损伤。

例 7.2

在 IP 分组中，HLEN 的值是二进制的 1000。试问这个分组携带了多少个字节的选项？

解

HLEN 的值 8，这表明首部总的字节数是 8×4 或 32 字节。前 20 个字节是基本首部，剩下的 12 个字节就是选项部分。

例 7.3

在 IP 分组中，HLEN 的值是 5_{16} ，总长度字段的值是 0028_{16} 。试问这个分组携带了多少

HLEN 是以 4 字节为单位的...

0028 (16) 等于 $8 + 2 \times 16 = 8 + 32 = 40 (10)$

字节的数据？

解

HLEN 的值是 5，这表明首部长度的字节数是 5×4 或 20 字节（无选项）。总长度是 40 字节，这表明分组携带了 20 字节的数据（ $40 - 20$ ）。

例 7.4

有一个到达的分组的前几个十六进制数字如下所示：

45000028000100000102……

这个分组在被丢弃之前还能传送多少跳？这个数据属于什么上层协议？

解

要找出生存时间字段，我们必须跳过 8 个字节（16 个十六进制数字）。生存时间字段是第 9 个字节，它的值是 01。这表明分组只能再走一跳。协议字段是下一个字节（02），这表明上层协议是 IGMP（见表 7.2）。

7.3 分片

数据报可以穿越不同的网络。每个路由器都会从收到的数据帧中拆解出 IP 数据报，并对它进行处理，然后再将它封装成另一个帧。接收到的帧的格式与长度取决于这个帧刚刚经过的物理网络所使用的协议，而发送出去的帧格式与长度则取决于这个帧将要经过的物理网络所使用的协议。例如，如果某台路由器将一个以太网连接到一个广域网，那么它收到的帧是以太网格式的，而发送的帧是广域网格式的。

7.3.1 最大传送单元（MTU）

每个数据链路层协议都有自己的帧格式。在这个格式中有一个字段是“数据字段最大长度”。换言之，当数据报被封装成帧时，数据报的总长度必须小于这个最大长度，它是由网络使用的硬件和软件带来的限制确定的（见图 7.6）。

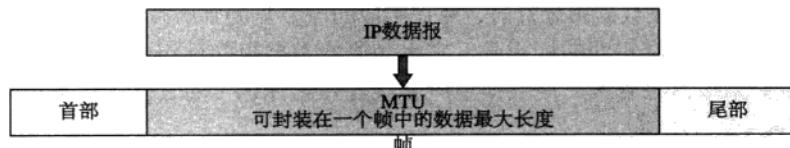


图 7.6 MTU

对于不同的物理网络协议，MTU 的值是不同的。例如以太网局域网的这个值是 1500 字节，FDDI 局域网的是 4352 字节，而 PPP 的这个值则是 296 字节。

为了使 IP 协议与物理网络无关，协议设计者们决定让 IP 数据报的最大长度等于 65535 字节。如果我们使用的协议的 MTU 正好等于这个数值，那么传输的效率会很高。但是，对于其他一些物理网络，我们不得不分割数据报，使它们能够通过这些网络。这就称为分片（fragmentation）。

源点通常不会对 IP 分组进行分片。因为运输层会把数据划分成 IP 与源点使用的数据链路层可接纳的大小。

当数据报被分片时，每一个数据报片都有自己的首部，其中大部分的字段是重复的，但有些是变化的。如果已经分片的数据报遇到具有更小 MTU 的网络，那么这些已经分片的数据报还可再进行分片。换言之，数据报在到达最后终点之前可以经过多次分片。

数据报可以被源主机或途中的任何路由器分片，但是数据报的重装却只能在目的主机上进行，因为每一个分片都变成了独立的数据报。一方面分片的数据报可以各走不同的路由，我们永远无法控制或保证分片的数据报应当走哪一条路径；另一方面，属于同一个数据报的所有数据报片最终总是会到达它们的目的主机，所以从逻辑上讲应当在最后的终点进行重装。另外还有一个更有力的反对在传输期间进行重装的理由，那就是这样做会严重影响效率。

当数据报被分片时，首部中一些必要的部分必须被复制到所有的分片中。选项字段可以被复制，也可不被复制，如我们将要在下一节中所看到的。对数据报进行分片的主机或路由器必须改变三个字段的值：标志、分片偏移和总长度。其余的各字段必须被复制。当然，不管是否进行分片，检验和的值总是要重新计算的。

只有数据报中的数据是分片的。

7.3.2 与分片有关的字段

与一个数据报的分片与重装有关的字段是：标识、标志和分片偏移。

□ **标识 (identification)** 这个 16 位字段标志了从源主机发出的一个数据报。当数据报离开源主机后，此标识与源 IP 地址的组合必须唯一地确定这个数据报。为了保证唯一性，IP 协议使用了一个计数器来为数据报生成标号。这个计数器的初始值是一个正整数。当 IP 协议每发送一个数据报时，就把这个计数器的当前值复制到标识字段中，并把计数器的值加 1。只要这个计数器保存在主存储器中，唯一性就能得到了保证。在数据报分片时，标识字段的值要复制到所有的分片中。换言之，所有的分片都具有相同的标识号，它也是原始数据报的标识号。这个标识号在终点重装数据报时很有用。终点知道所有具有相同标识号的分片必须被组装成一个数据报。

□ **标志 (flag)** 这是 3 位的字段。第一位保留（未用）。第二位称为“不分片”位。如果这个值是 1，机器就不能对该数据报进行分片。若无法通过任何可用的物理网络把这个数据报转发出去，就丢弃这个数据报，并向源主机发送 ICMP 差错报文（见第 9 章）。如果这个值为 0，则可在必要时对这个数据报进行分片。第三位是“还有分片”位。若这个值是 1，则表示这个数据报不是最后的分片，在这个分片后面还有更多的分片。
若这个值是 0，则表示这已是最后的或唯一的分片（见图 7.7）。

D: 不分片
M: 还有分片 

图 7.7 标志字段

□ 分片偏移 这个13位字段表示的是分片在整个数据报中的相对位置。这是数据在原始数据报中的偏移量，以8字节为度量单位。图7.8所示为一个有4000字节数据的数据报被划分为三个分片。在原始数据报中，数据字节从0到3999编号。第一个分片携带的是字节0~1399。对于这个数据报，分片偏移是 $0/8=0$ 。第二个分片携带的是字节1400~2799。对于这个数据报，偏移值是 $1400/8=175$ 。最后，第三个分片携带的是字节2800~3999。对于这个数据报，偏移值是 $2800/8=350$ 。

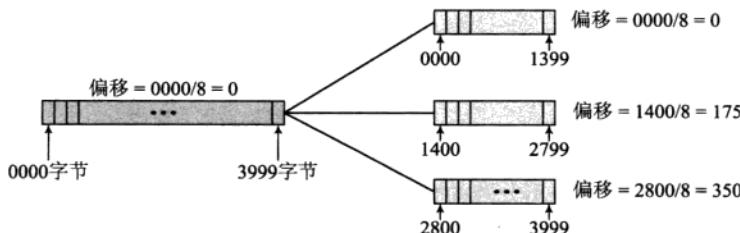


图7.8 分片举例

要记住偏移值是以8字节为单位。这样做是因为分片偏移字段的长度只有13位，它不能表示超过8191个字节。这就迫使对数据报进行分片的主机或路由器在选择每一个分片的长度时必须使第一个分片的字节数量能够被8除尽。

图7.9所示为图7.8中分片的详细扩展图。应注意的是，所有分片的标识字段值都是一样的。还应注意，除最后一个分片外，所有分片的标志字段中的“还有分片”位均被置1。此外图中还展示了每个分片的分片偏移字段的值。

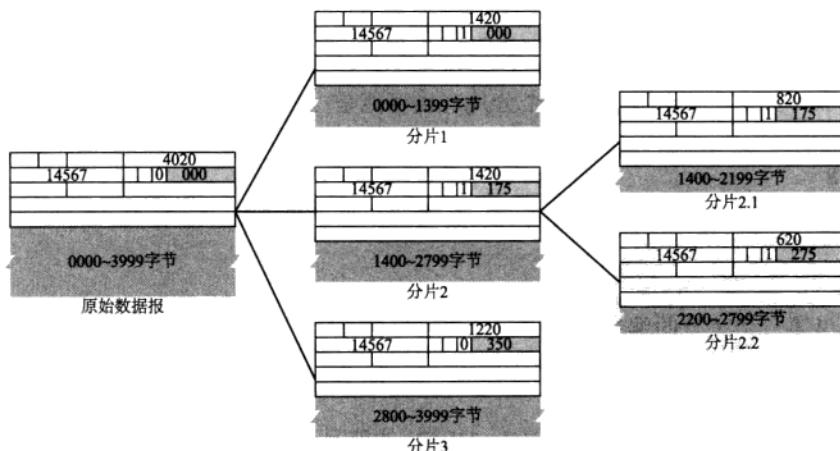


图7.9 详细的分片举例

这个图还描绘了如果分片本身再一次进行分片时会发生什么。在这种情况下，分片偏移值永远是相对于原始数据报的。例如，在本图中第二个分片后来又被划分为两个分片，长度分别为800字节和600字节，但它们的偏移量所表示的位置都相对于原

始数据的。

很显然即使各个分片采取不同的路径并失序到达，最终的目的主机也能够用收到的分片（假定没有一个丢失）重装成原始的数据报，使用策略如下：

- 第一个分片的分片偏移字段值为 0。
- 把第一个分片的长度除以 8。这个结果就是第二个分片偏移值。
- 把第一个和第二个分片的总长度除以 8。这个结果就是第三个分片偏移值。
- 继续以上过程。最后一个分片的“还有分片”位的值是 0。

例 7.5

一个到达的分组 M 位是 0。这是第一个分片，还是最后一个分片，或者是中间的分片？我们能否知道这个分组有没有被分片？

解

若 M 位是 0，就表示后面再没有分片了，这个分片就是最后一个。但是，我们不知道原始分组是否被分片了。一个不分片的分组也被认为是最后一个分片。

例 7.6

一个到达的分组 M 位是 1。这是第一个分片，还是最后一个分片，或者是中间的分片？我们能否知道这个分组有没有被分片？

解

若 M 位是 1，就表示至少有一个以上的分片。这个分片可能是第一个，也可能是中间的一个，但不会是最后一个。我们不知道这是第一个还是中间的一个，我们还需要有更多的信息（分片偏移值）。见后面的例子。

例 7.7

一个到达的分组 M 位是 1，而分片偏移值是 0。这是第一个分片，还是最后一个分片，或者是中间的分片？

解

因为 M 位是 1，它可能是第一个分片，也可能是一个中间分片。又因为分片偏移值是 0，所以这是第一个分片。

例 7.8

一个到达分组的分片偏移值是 100。它的第一个字节的编号是多少？我们能否知道其最后一个字节的编号？

解

要找出第一个字节的编号，我们把分片偏移值乘以 8。这就表明它的第一个字节的编号是 800。我们不能确定最后一个字节的编号，除非我们知道数据的长度。

例 7.9

一个到达分组的分片偏移值是 100，HLEN 的值是 5，而总长度字段的值是 100。它的第一个字节和最后一个字节的编号是多少？

解

第一个字节的编号是 $100 \times 8 = 800$ 。总长度是 100 字节，首部长度是 20 字节 (5×4)，这表明在这个数据报中共有 80 字节。若第一个字节的编号是 800，则最后一个字节的编号必定是 879。

7.4 选项

IP数据报的首部由两部分组成：固定部分和可变部分。固定部分的长度是20字节，我们已在前一节中讨论过了。可变部分由一些选项组成，它最多可长达40字节。

选项这个词的字面意思就是说它们对每个数据报来说并不是必需的。这些选项可用于网络的测试和排错。虽然选项并非IP首部中的必要项目，但对选项的处理却是IP软件的必要部分。这就意味着只要这些选项出现在首部中，任何实现都必须能够处理它们。

7.4.1 格式

图7.10给出了选项的格式。它的组成是：一个字节的类型字段，一个字节的长度字段，以及可变长度的值字段。这三个字段经常被称为TLV (type-length-value)。

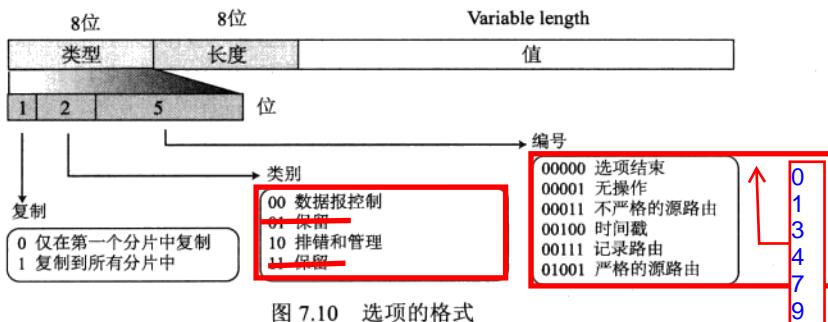


图7.10 选项的格式

类型

类型字段(type field)的长度为8位，包括了三个子字段：复制、类别和编号。

- 复制** 这个1位子字段控制了选项在分片中的出现。若这个值是0，就表示选项必须仅仅复制到第一个分片。若这个值是1，则表示选项必须复制到所有分片中。
- 类别** 这个2位子字段定义了该选项的一般用途。若这个值是00，就表示选项是用作数据报的控制。若这个值是10，则表示选项是用作排错和管理。其他两个可能的值(01和11)目前尚未定义。
- 编号** 这个5位子字段定义了选项的类型。虽然5位共可定义32种不同类型，但目前仅使用了6种类型。我们将在下一小节讨论它们。

长度

长度字段(length field)定义选项的总长度，包括类型字段和长度字段本身。这个字段并不是在所有类型的选项中都会出现的。

值

值字段(value field)包含的是某些特定选项所需的数据。像长度字段一样，这个字段并不是在所有类型的选项中都会出现的。

7.4.2 选项类型

如以上所述，目前仅使用了六种选项。其中的两种是单字节选项，它们不需要长度字段和值字段。剩下的四种是多字节选项，它们需要长度字段和值字段（见图 7.11）。

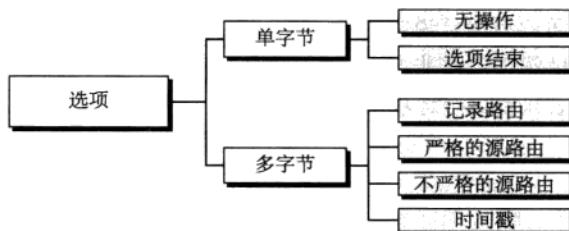


图 7.11 选项的种类

无操作选项

无操作选项（no-operation option）是个 1 字节的选项，用作选项和选项之间的填充符。例如，它可被用作使下一个选项在 16 位或 32 位边界上对齐（见图 7.12）。

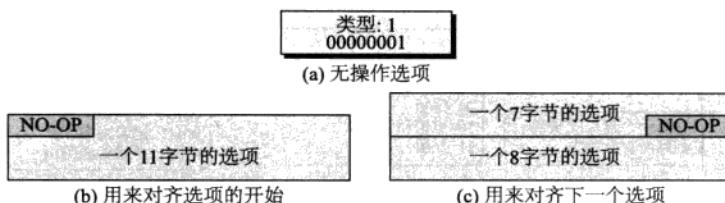


图 7.12 无操作选项

选项结束选项

选项结束选项（end-of-option option）也是个 1 字节的选项，用于选项字段结束时的填充。但是，它只能用作最后一个选项，并且只能使用一次。在这个选项之后，接收器就开始寻找净荷数据。这就表示，如果对齐选项需要超过一个字节，那么必须使用一些无操作选项，然后再跟随一个选项结束选项（见图 7.13）。

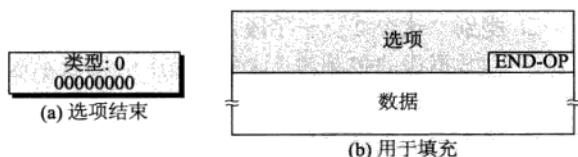


图 7.13 选项结束选项

记录路由选项

记录路由选项（record-route option）是用来记录处理数据报的因特网路由器。它可以列出最多九个路由器的 IP 地址，因为首部的最大长度是 60 字节，其中还包括了 20 个字节的基本首部，这就意味着只剩下 40 字节留给选项部分。源点会在选项中创建一些预留位置字段，并由经过的路由器来填写。图 7.14 描绘了记录路由选项的格式。

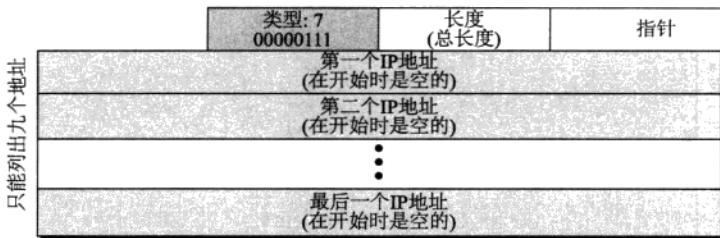


图 7.14 记录路由选项

类型字段和长度字段在前面都已讨论过了。指针字段 (pointer field) 是一个偏移量的整数字段，它包含的是第一个空项的字节号。换言之，它指向第一个可用的空项。

源点在选项的值字段中创建了一些用于填写 IP 地址的空字段。当数据报离开源点时，所有这些字段都是空的。指针字段的值是 4，指向第一个空字段。

在数据报的前进过程中，处理这个数据报的每一个路由器都要比较指针字段的值和长度字段的值。若指针值大于长度值，则选项已满，不做任何改变。但是，如果指针值小于长度值，路由器就要在下一个空字段中插入自己的出口 IP 地址（谨记路由器的 IP 地址不只一个）。在这种情况下，路由器把数据报离开的那个接口的 IP 地址加入到选项中，然后路由器再把指针的值递加 4。图 7.15 描绘了一个数据报自左向右地从一个路由器到另一个路由器转发时，它的选项的内容。

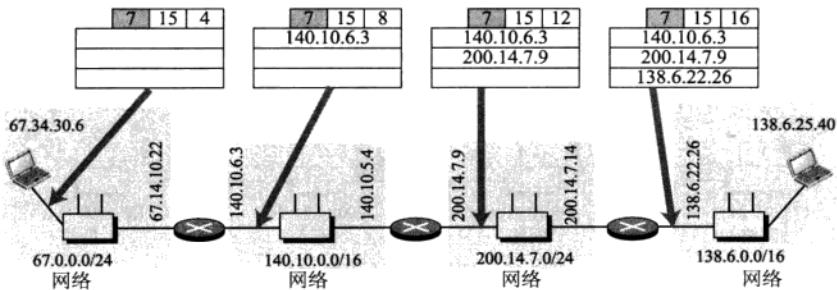


图 7.15 记录路由的概念

严格源路由选项

严格源路由选项 (strict-source-route option) 被源点用来预先指定数据报在因特网中传送时的路由。对某些用途来说，由源点指定路由是很有用的。发送方可以选择具有特定服务类型的路由，如最小延时或最大吞吐量。此外，发送方出于自己的考虑，还可以选择一条更加安全或更加可靠的路由。例如，发送方可以选择某条路由使得数据报不经过竞争对手的网络。

如果数据报指定了严格的源路由，那么这个数据报就必须经过在选项中定义的所有路由器。若某路由器的 IP 地址未被列入到数据报中，则这个数据报就一定不能通过这个路由器。若数据报通过了某个未被列入的路由器，则这个路由器将丢弃该数据报并发出差错报文。若数据报到达终点时仍有某些列入的路由器未曾经过，则终点将把这个数据报丢弃并发出差错报文。

然而因特网上的普通用户通常并不知道因特网的物理拓扑。因此，严格的源路由并不是大多数用户的选择。图 7.16 所示为严格的源路由选项的格式。

类型: 137 10001001	长度 (总长度)	指针
第一个IP地址 (在开始时填入)		
第二个IP地址 (在开始时填入)		
•		
最后一个IP地址 (在开始时填入)		

图 7.16 严格的源路由选项

这种格式类似于记录路由选项的格式，其中的区别在于所有的 IP 地址都是由发送方输入的。

在数据报的前进过程中，处理这个数据报的每一个路由器都要对指针字段的值和长度字段的值相比较。若指针值大于长度值，则这个数据报已经通过了所有预设定的路由器，这个数据报不能再向前转发了，必须丢弃它并发出差错报文。若指针值小于长度值，则路由器就比较该帧的目的 IP 地址是否与自己的入口 IP 地址一致，如果两者相等，则处理这个数据报，并用输出帧的目的 IP 地址来替换指针所指向的 IP 地址，并把指针值递增 4，然后转发这个数据报。如果两者不相等，就丢弃这个数据报，并发出差错报文。图 7.17 描绘了一个数据报从源点向终点前进途中经过的路由器所采取的动作。

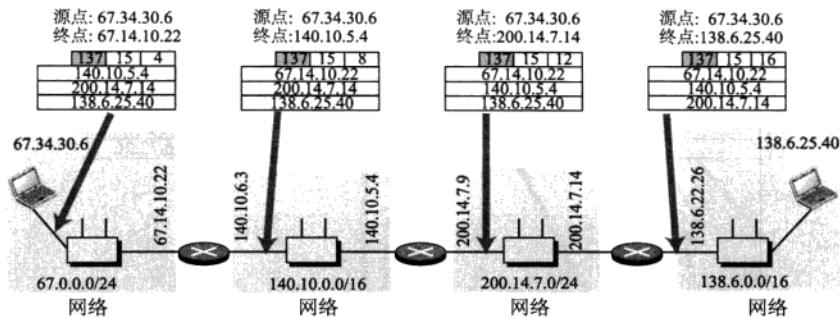


图 7.17 严格的源路由的概念

不严格源路由选项

不严格源路由选项（loose-source-route option）与严格源路由选项相似，但条件要放宽一些。表中列出的路由器必须通过，但数据报还可以访问其他的路由器。图 7.18 所示为不严格源路由选项的格式

类型: 131 10000011	长度 (总长度)	指针
第一个IP地址 (在开始时填入)		
第二个IP地址 (在开始时填入)		
•		
最后一个IP地址 (在开始时填入)		

图 7.18 不严格的源路由选项

时间戳

时间戳选项 (timestamp option) 用来记录路由器处理数据报的时间。时间是从午夜开始以毫秒计的全球通用时间。知道数据报的处理时间有助于用户和管理者对因特网上的各路由器的行为进行跟踪。我们能够估计数据报从一个路由器到另一个路由器所需的时间。我们用估计是因为虽然所有的路由器都使用全球通用时间，但它们的本地时钟可能并没有同步。

然而因特网上的普通用户通常并不知道因特网的物理拓扑。因此，时间戳选项并不是大多数用户的选择。图 7.19 所示为时间戳选项的格式。

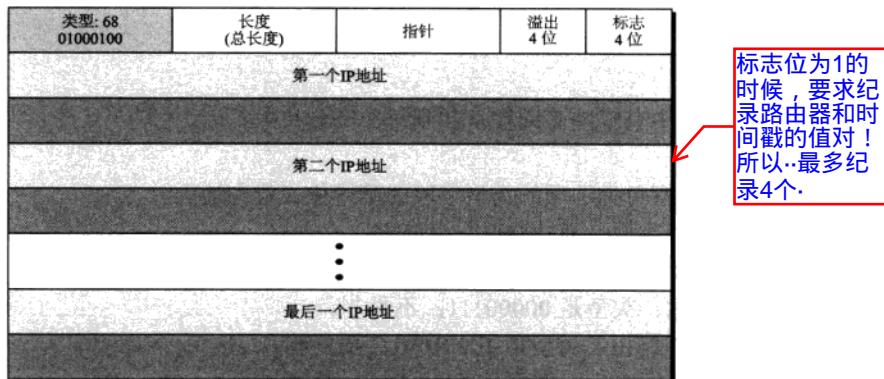


图 7.19 时间戳选项

在图中类型字段与长度字段的定义和以前一样。溢出字段记录了因缺少可用字段而无法将其时间戳加上去的路由器的数量。标志字段规定了被访问路由器的责任。若这个标志是 0，则每一个路由器仅把时间戳加到这个字段上。若标志值是 1，则每一个路由器必须增加其出口 IP 地址和时间戳。若这个值为 3，则 IP 地址是预先给出的，每一个路由器必须用自己的入口 IP 地址来检查给定的 IP 地址。若匹配，则路由器用自己的出口 IP 地址覆盖该 IP 地址，并加入时间戳（见图 7.20）。

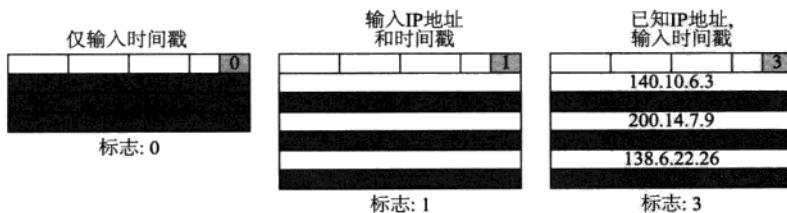


图 7.20 时间戳选项中标志的使用

图 7.21 描绘了一个数据报从源点向终点前进途中经过的路由器所采取的动作。图中假设标志的值是 1。

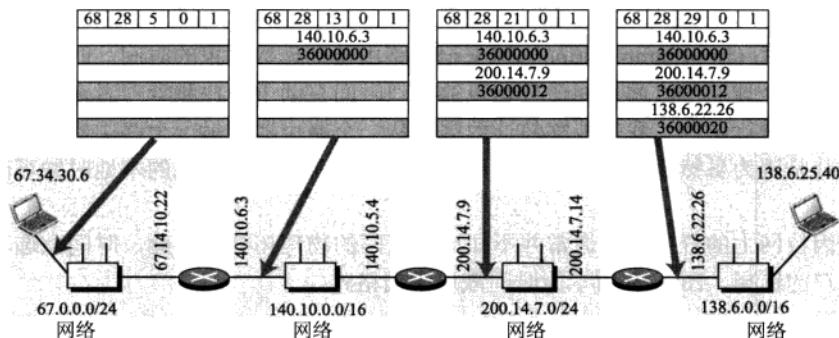


图 7.21 时间戳的概念

例 7.10

在六个选项中有哪些必须被复制到每一个分片中？

解

我们可以观察每种选项的类型字段的第一位（最左边的）。

- 无操作：类型是 00000001；不复制。
- 选项结束：类型是 00000000；不复制。
- 记录路由：类型是 00000111；不复制。
- 严格的源路由：类型是 10001001；要复制。
- 不严格的源路由：类型是 10000011；要复制。
- 时间戳：类型是 01000100；不复制。

例 7.11

在六个选项中哪些用于数据报控制，哪些用于排错和管理？

解

我们观察类型字段的第二位和第三位（从左边数）。

- 无操作：类型是 00000001；数据报控制。
- 选项结束：类型是 00000000；数据报控制。
- 记录路由：类型是 00000111；数据报控制。
- 严格源路由：类型是 10001001；数据报控制。
- 不严格源路由：类型是 10000011；数据报控制。
- 时间戳：类型是 01000100；排错和管理控制。

例 7.12

UNIX 中可用来检查 IP 分组的传送过程的实用程序之一是 ping。在下一章我们将更详细地讨论 ping 程序。在本例中，我们要说明怎样使用这个程序来了解一个主机是否可用。我们 ping 一个在 De Anza 学院名叫 fhda.edu 的服务器。结果显示，这台主机的 IP 地址是 153.18.8.1。从结果中还可看出使用的字节数。

```
$ ping fhda.edu
PING fhda.edu (153.18.8.1) 56(84) bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 0 ttl=62 time=1.87 ms
...
```

例 7.13

我们还可以使用具有-R 选项的 ping 实用程序来实现记录路由选项。这个结果给出了接口和 IP 地址。

```
$ ping -R fhda.edu
PING fhda.edu (153.18.8.1) 56(124) bytes of data.
64 bytes from tiptoe.fhda.edu
(153.18.8.1): icmp_seq = 0 ttl=62 time=2.70 ms
RR: voyager.deanza.fhda.edu (153.18.17.11)
    Dcore_G0_3-69.fhda.edu (153.18.251.3)
    Dbackup_V13.fhda.edu (153.18.191.249)
    tiptoe.fhda.edu (153.18.8.1)
    Dbackup_V62.fhda.edu (153.18.251.34)
    Dcore_G0_1-6.fhda.edu (153.18.31.254)
voyager.deanza.fhda.edu (153.18.17.11)
```

例 7.14

实用程序 traceroute 可用来跟踪一个分组的路由。这个结果给出了经过的三个路由器。

```
$ traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 37 byte packets
1 Dcore_G0_1-6.fhda.edu (153.18.31.254) 0.972 ms 0.902 ms 0.881 ms
2 Dbackup_V69.fhda.edu (153.18.251.4) 2.113 ms 1.996 ms 2.059 ms
3 tiptoe.fhda.edu (153.18.8.1) 1.791 ms 1.741 ms 1.751 ms
```

例 7.15

程序 traceroute 可用来实现不严格的源路由。选项-g 允许我们定义从源点到终点所要经过的路由器。以下所示为我们怎样发送一个分组到 fhda.edu 服务器，并要求这个分组经过路由器 153.18.251.4。

```
$ traceroute -g 153.18.251.4 fhda.edu
Traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
1 Dcore_G0_1-6.fhda.edu (153.18.31.254) 0.976 ms 0.906 ms 0.889 ms
2 Dbackup_V69.fhda.edu (153.18.251.4) 2.167 ms 2.148 ms 2.037 ms
```

例 7.16

程序 traceroute 还可用来实现严格的源路由。选项-G 强制分组经过命令行所定义的路由器。以下所示为我们怎样发送一个分组到 fhda.edu 服务器，并强制这个分组只经过路由器 153.18.251.4。

```
$ traceroute -G 153.18.251.4 fhda.edu
Traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
1 Dbackup_V69.fhda.edu (153.18.251.4) 2.168 ms 2.148 ms 2.037 ms
```

7.5 检验和

绝大多数 TCP/IP 协议采用的差错检测方法称为检验和 (checksum)。检验和能够防止分组在传输期间出现的损坏。检验和是附加在分组上的冗余信息。

发送端计算出检验和，并把得到的结果与分组一起发送出去。接收端对包括检验和在内的整个分组重复同样的计算。若得到了满意的结果（稍后再说明）则接受这个分组，否则就把它丢弃。

7.5.1 在发送端计算检验和

在发送端，分组首部被划分为 n 位的段 (n 通常取值为 16)。把这些段用反码算术运算（见附录 D）相加，得到的和的长度也是 n 位。再把这个和取反码（把所有的 0 变为 1 以及把所有的 1 变为 0），就得出了检验和。

发送端按以下步骤产生检验和：

- 把分组划分为 k 段，每段的长度都是 n 位。
- 用反码算术运算把所有这些段相加。
- 把最终结果取反码就得出检验和。

7.5.2 在接收端计算检验和

接收端把收到的分组划分为 k 段，并把所有这些段相加，然后把得到的和取反码。若最后结果为 0，则接受这个分组，否则就拒绝这个分组。图 7.22 用图解的方法给出了在发送端和接收端所发生的过程。

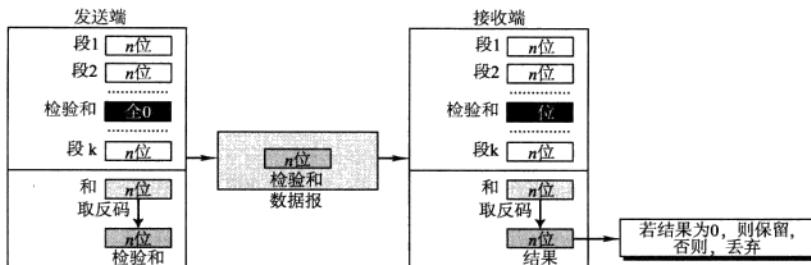


图 7.22 检验和的概念

我们说过，当接收端把所有段相加并把结果取反码，如果在数据传输或处理过程中未出现差错，则得到的结果必定为零。根据反码算术运算的规则，这个说法是正确的。

假设我们在发送端将所有段相加后得到的数称为 T 。当我们用反码算术运算把这个数取反码时，我们就得到这个数的负值。这就表明，若所有段之和是 T ，则检验和就是 $-T$ 。

当接收端收到这个分组时，把所有的段相加。这实际上就是把 T 和 $-T$ 相加，在二进制反码中得出 -0 （负的零）。把这个结果取反码时， -0 又变为 0。因此，若最终结果为 0，则接受这个分组，否则就丢弃这个分组（见图 7.23）。



图 7.23 用反码算术运算计算检验和

7.5.3 IP分组中的检验和

在IP分组中，检验和的实现与上面所讨论的原理一样。首先，把“检验和”字段值置为0。然后把整个首部划分为16位的段，再将各段相加。把结果（即和）取反码，并插入到检验和字段中。

IP分组中的检验和仅覆盖首部，而不管数据。这有两个原因。首先，将数据封装在IP数据报中的所有高层协议，都有一个涉及整个分组的检验和字段。因此，IP数据报的检验和就不必再检验所封装的数据部分。其次，每经过一个路由器，IP数据报的首部就要有所变化，但数据部分保持不变。因此检验和只对发生变化的部分进行检验。若检验包含数据部分，则每一个路由器必须重复计算整个分组的检验和，这也意味着要花费更多的处理时间。

IP中的检验和只覆盖首部，不包括数据。

例 7.17

图7.24描绘了一个没有选项的IP首部计算检验和的例子。首部被划分为16位的段。把所有这些段相加，再把得到的和取反码。最后把结果插入到检验和字段中。

4、5和0	→	01000101 00000000	4	5	0	28
28	→	00000000 00011100		1	0	0
1	→	00000000 00000001			0	0
0和0	→	00000000 00000000				0
4和17	→	00000100 00010001	4	17		
0	→	00000000 00000000				0
10.12	→	00001010 00001100				10.12.14.5
14.5	→	00001110 00000101				12.6.7.9
12.6	→	00001100 00000110				
7.9	→	00000111 00001001				
和	→	01110100 01001110				
检验和	→	10001011 10110001				去替换0

图 7.24 在发送端计算检验和的例子

例 7.18

图7.25所示为接收端(或中间路由器)对检验和的检查计算，假设首部没有出现差错。首部被划分为16位的段。把所有这些段相加，再把得到的和取反码。因为得到的结果是16个0，所以此分组被接受。

4、5和0	→	01000101 00000000	4	5	0	28
28	→	00000000 00011100		1	0	0
1	→	00000000 00000001			0	0
0和0	→	00000000 00000000				0
4和17	→	00000100 00010001	4	17		35761
检验和	→	10001011 10110001				
10.12	→	00001010 00001100				
14.5	→	00001110 00000101				
12.6	→	00001100 00000110				
7.9	→	00000111 00001001				
和	→	1111 1111 1111 1111				
检验和	→	0000 0000 0000 0000				

图 7.25 在接收端计算检验和的例子

在附录D中给出了有关计算检验和的算法。

7.6 IP 在 ATM 上运行

在前几节中，我们都假设支持 IP 数据报在其上传送的底层网络是局域网或者是点对点的广域网。在本小节，我们希望了解 IP 数据报如何通过像 ATM 这样的交换广域网来传送。我们会看到它们有相同的地方，也有些不一样的地方。IP 分组被封装在信元中（不只一个）。对于一个设备的物理地址，ATM 网络有它自己的定义。在 IP 地址和物理地址之间的绑定是通过一个称为 ATMARP 的协议获得的（参见第 8 章）。

7.6.1 ATM 广域网

我们在第 3 章中已经讨论过 ATM 广域网。作为信元交换网络的 ATM 可以成为 IP 数据报的高速公路。图 7.26 所示为一个 ATM 网络在因特网中是如何被利用的。

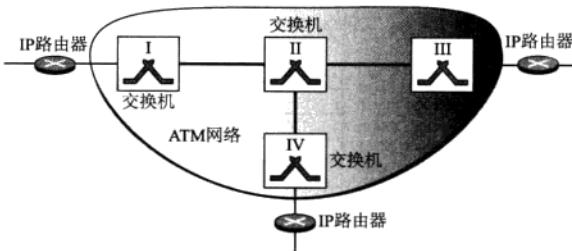


图 7.26 因特网中的一个 ATM 广域网

AAL 层

在第 3 章中，我们讨论了不同的 AAL 层以及它们的应用。能被因特网使用的 AAL 只有 AAL5，有时它也称为简单有效的适配层（simple and efficient adaptation layer，SEAL）。AAL5 假设从 IP 数据报中创建出来的所有信元都属于同一个报文。因此，AAL5 不提供地址、序列或其他的首部信息。实际上，只有一些填充和一个四字段的尾部被附加在 IP 分组上。

AAL5 接受不多于 65 536 个字节的 IP 分组，并添加 8 字节的尾部以及一些必要的填充，以保证该尾部出现的位置正好在接收设备预期的地方（最后一个信元的最后 8 个字节）。一旦填充和尾部准备就绪，AAL5 就把这个报文以 48 字节的报文段的形式递交给 ATM 层。

IP 协议使用的 AAL 层是 AAL5。

为什么使用 AAL5

一个经常被提到的问题是：为什么我们要用 AAL5？为什么我们不简单地把一个 IP 分组封装在一个信元中？答案就是使用 AAL5 的效率更高。如果将一个 IP 分组封装在一个信元中，那么 IP 层的数据必须是 $53 - 5 - 20 = 27$ 字节，因为 IP 首部至少需要 20 个字节，且 ATM 首部还需要 5 个字节，这样做的效率只有 $27/53$ ，也就是 51%。通过让 IP 数据报跨越多个信元，我们就可以让这些信元来共同分担 IP 的额外开销（20 字节），从而提高了效率。

7.6.2 信元的路由选择

ATM 网络在两个路由器之间建立一条路由。我们把这两个路由器称为进入点(entering-point)路由器和离去点(exiting-point)路由器。如图 7.27 所示，信元从进入点路由器处产生，到离去点路由器结束。

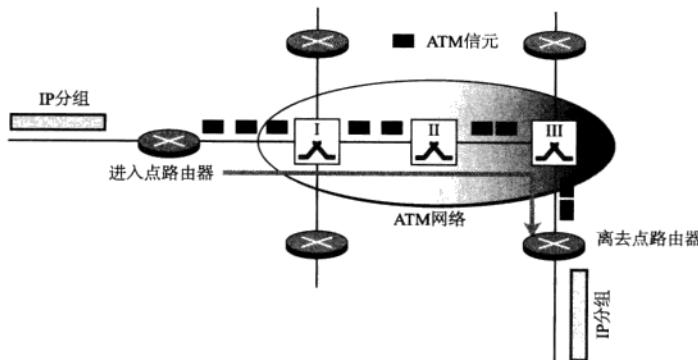


图 7.27 进入点和离去点路由器

地址

把信元从一个特定的进入路由器转发到一个特定的离去路由器，需要 3 种类型的地址：IP 地址、物理地址和虚电路标识符。

IP 地址 每个连接到 ATM 网络上的路由器都有一个 IP 地址。在后面我们会看到这些地址的前缀可能相同，也可能不同。IP 地址在 IP 层指明了特定的路由器。但是它在 ATM 网络中没有任何作用。

物理地址 每个连接到 ATM 网络上的路由器（或任何其他设备）都有一个物理地址。这个物理地址与 ATM 网络相关，但和因特网没有什么关系。ATM 论坛为 ATM 网络定义了 20 字节的地址。在一个网络中每个地址都是唯一的，且由网络管理员来指定。ATM 网络中物理地址的作用与局域网中 MAC 地址的作用是一样的。物理地址连接建立时使用。

虚电路标识符 正如我们在第 3 章中讨论的，ATM 网络内部的交换机根据虚电路标识符（VPI 和 VCI）来为信元选路。这个虚电路标识符在数据传送时使用。

地址绑定

ATM 网络需要用虚电路标识符为信元选路。在 IP 数据报中只包含了源 IP 地址和目的 IP 地址。必须根据目的 IP 地址来判断其虚电路标识符。图 7.28 描绘了如何做到这一点。以下是采取的步骤：

1. 进入点路由器收到一个 IP 数据报。它用 IP 数据报的目的地址和自己的路由表来找出下一个路由器（即离去点路由器）的 IP 地址。这一步与数据报通过一个局域网时是完全一致的。
2. 进入点路由器使用了一个称为 ATMARP 的协议服务找出离去点路由器的物理地址。ATMARP 与 ARP 类似（在第 8 章讨论）。
3. 正如在第 3 章中讨论的，虚电路标识符被绑定到相应的物理地址。

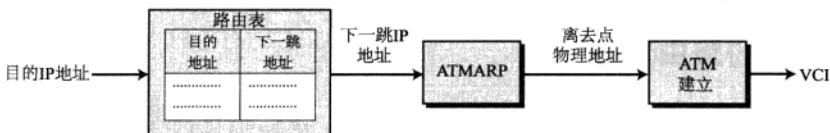


图 7.28 IP 在 ATM 上运行的地址绑定

7.7 安全性

与整个因特网一样，IPv4 协议在诞生之初，因特网上的用户彼此之间都是互相信任的。IPv4 没有提供任何安全措施。但是，目前情况又有所不同，因特网已不再是安全的。虽然我们将在第 29 章和第 30 章中讨论普遍意义上的网络安全性，并会专门研究 IP 协议的安全问题，但是在这里我们要简单介绍一下有关 IP 协议中的安全问题及解决方法。

7.7.1 安全问题

有三个安全问题是特别应用于 IP 协议的：分组窃取 (packet sniffing)、分组篡改 (packet modification) 和 IP 伪装 (IP spoofing)。

分组窃取

入侵者可能会截取并复制一个 IP 分组。分组窃取是一种被动攻击方式，此时攻击者并没有改变分组的内容。这种类型的攻击是非常难以检测到的，因为发送方和接收方可能永远都无法知道分组已经被复制过了。虽然分组窃取没有办法阻止，但是对分组进行加密能够令攻击者无功而返。攻击者仍然能够窃取分组，但是没有办法知道分组中的内容。

分组篡改

第二种攻击类型是分组的篡改。攻击者先截取分组，再改变其中的内容，然后把新的分组发送给接收方，而接收方以为这个分组来自原始的发送方。这种类型的攻击可以被数据完整性机制检测到。在接收方打开并使用报文的内容之前，可以先用这个机制来确保收到的分组在传输期间不曾被修改过。

IP 伪装

攻击者伪装成另一个人，创建一个携带另一台计算机的源地址的 IP 分组。攻击者可以发送 IP 分组到某银行，并假装这个分组来自该银行的一位客户。这种类型的攻击可以通过起源鉴别机制来加以防范。

7.7.2 IPSec

目前我们可以用一种称为 IPSec (IP 安全性) 的协议来保护 IP 分组不受前面所提到的各种攻击。这个协议是与 IP 协议结合在一起使用的，它在两个实体之间创建了一种面向连接的服务，使这两个实体在交换 IP 分组时不用担心会受到之前讨论的三种攻击。我们将在第 30 章中详细讨论 IPSec，而此处我们只需要了解 IPSec 能够提供以下四种服务：

定义算法和密钥

如果两个实体希望在彼此之间创建一条安全信道，它们可以就一些可用的算法和密钥达成一致意见，以提供安全性。

分组加密

为了保密，可以使用一种加密算法和共享密钥对两个实体之间交换的分组进行加密，至于加密算法和共享密钥的选择，双方需要在第一阶段达成一致意见。这样做使得分组窃取攻击无法得逞。

数据完整性

数据的完整性用于保证分组在传输期间不被篡改。如果接收到的分组不能通过数据完整性测试，那么这个分组就被丢弃。这样做可以防范前面所描述的第二种攻击：分组篡改。

起源鉴别

IPSec 可以鉴别分组的起源，以保证该分组不是由一个冒名顶替的人创建的。这样做可以防范前面所描述的 IP 伪装攻击。

7.8 IP 软件包

本节我们将给出一个假想的简化 IP 软件包的例子，目的是为了展示本章所讨论的各种不同概念之间的相互关系。图 7.29 描绘了 IP 软件包的八个构件以及它们之间的交互。

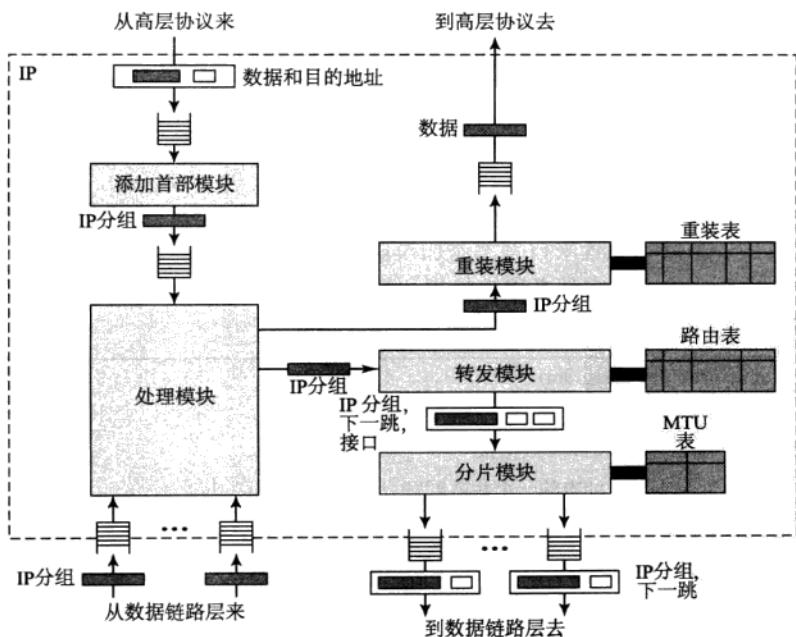


图 7.29 IP 的构件

虽然 IP 支持多种选项，但在我们的软件包中省略了选项的处理，以便使它在目前的水平上更加容易理解。此外，为了简单性我们还牺牲了效率。

可以认为 IP 软件包包含了八个构件：首部添加模块、处理模块、转发模块、分片模块、重装模块、路由表、MTU 表和重装表。此外，在软件包中还包括一些输入和输出队列。

这个软件包接收来自数据链路层或高层协议的分组。如果分组是从高层协议来的，那么它必须将其交付给数据链路层进行传输（除非是在使用环回地址 127.X.Y.Z 时）。如果分组来自数据链路层，那么它或者将其交付给数据链路层进行转发（在路由器中的情况），或者交付给高层协议（如果分组的目的 IP 地址与本站的地址一致）。请注意，在这里使用了多个送往与取来自于数据链路层的队列，因为路由器是连接多个网络的。

7.8.1 首部添加模块

首部添加模块（header-adding module）（表 7.3）接收来自高层协议的数据及其目的 IP 地址。它通过添加 IP 首部，把数据封装在一个 IP 数据报中。

表 7.3 首部添加模块

```

1 IP_Adding_Module (data, destination_address)
2 {
3     把数据封装成为 IP 数据报
4     计算检验和，把它插入到检验和字段
5     把数据发送到相应的输入队列
6     返回
7 }
```

7.8.2 处理模块

处理模块（processing module）（表 7.4）是 IP 软件包的核心。在我们的软件包中，处理模块接收来自一个接口或者来自首部添加模块的数据报。这两种情况的处理方式是一致的。不管数据报从何处来，它都必须被处理和转发。

表 7.4 处理模块

```

1 IP_Processing_Module (Datagram)
2 {
3     从一个输入队列中取出一个数据报
4     If(目的地址与本地地址中的一个相匹配)
5     {
6         把数据报发送到重装模块
7         返回
8     }
9     If(本机是路由器)
```

```

10  {
11      把 TTL 减 1
12  }
13  If(TTL 小于或等于零)
14  {
15      丢弃这个数据报
16      发送 ICMP 差错报文
17      返回
18  }
19  把数据报发送到转发模块
20  返回
21 }
```

处理模块首先检查这数据报是否已到达最后终点。如果是这样，分组要传送到重装模块。

如果这个结点是路由器，它就把生存时间 (TTL) 字段的值减 1。如果这个值小于或等于零，就丢弃这个数据报，并向发送端发送 ICMP 差错报文（见第 9 章）。如果 TTL 值在减 1 后还大于零，处理模块把这个数据报传递给转发模块。

7.8.3 队列

我们的软件包使用了两种类型的队列：输入队列和输出队列。输入队列 (input queues) 存放的是来自数据链路层或高层协议的数据报。输出队列 (output queues) 存放的是将要发送到数据链路层或高层协议的数据。处理模块从输入队列中取出数据报。分片和重装模块则向输出队列中添加数据报。

7.8.4 路由表

我们曾在第 6 章中讨论了路由表。转发模块利用路由表来确定分组的下一跳地址。

7.8.5 转发模块

我们曾在第 6 章中讨论过转发模块 (forwarding module)。转发模块接收来自处理模块的 IP 分组。如果分组需要被转发，就会把该分组传递给这个模块。转发模块找出下一站的 IP 地址以及发送该分组时应当经过的接口号。然后，转发模块把分组连同这些信息一起传递给分片模块。

7.8.6 MTU 表

分片模块使用 MTU 表找出特定接口的最大传送单元 (maximum transfer unit, MTU)。

MTU 表可以仅含两列：接口和 MTU。

7.8.7 分片模块

在我们的软件包中，分片模块（fragmentation module）（表 7.5）接收来自转发模块的 IP 数据报。转发模块给出了 IP 数据报、下一站（或者是直接交付中的最后终点，或者间接交付中的下一个路由器）的 IP 地址，以及发送这个数据报所必须通过的接口号。

分片模块咨询 MTU 表以找出这个特定接口号的 MTU。若数据报的长度大于 MTU，则分片模块要对数据报进行分片，为每一个分片添加首部，并把它们发送给 ARP 软件包（见第 8 章）以便进行地址解析和交付。

表 7.5 分片模块

```

1  IP_Fragmentation_Module (Datagram)
2  {
3      提取数据报的长度
4      If(长度 > 相应网络的 MTU)
5      {
6          If(D 位被置 1)
7          {
8              丢弃这个数据报
9              发送 ICMP 差错报文
10             返回
11         }
12     Else
13     {
14         计算最大长度
15         把数据报划分为分片
16         给每一个分片添加首部
17         给每一个分片添加需要的选项
18         发送分片
19         返回
20     }
21 }
22 Else
23 {
24     发送这个数据报
25 }
26 返回
27 }
```

7.8.8 重装表

重装表 (reassemble table) 是重装模块使用的。在我们的软件包中，重装表有5个字段：状态、源IP地址、数据报标识符、超时以及分片（见图7.30）。

St.: 状态
 S. A.: 源地址
 D. I.: 数据报标识
 T. O.: 超时
 F.: 分片

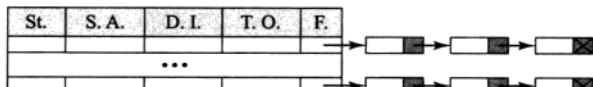


图 7.30 重装表

状态字段的值可以是FREE（空闲）或IN-USE（使用中）。源IP地址字段定义了数据报的源IP地址。数据报标识符是一个数字，它唯一地定义了一个数据报以及属于该数据报的所有分片。超时是一个预定义的时间，在这段时间内所有的分片都必须到达。最后，分片字段是指向分片链表的指针。

7.8.9 重装模块

重装模块 (reassemble module) (表7.6) 接收来自处理模块的，且已到达最终目的地的数据报分片。在我们的软件包中，重装模块把未分片的数据报也看成数据报分片，只不过这个数据报仅有一个分片。

因为IP协议是无连接协议，因而不能保证所有分片都按序到达。此外，属于一个数据报的分片可能与另一个数据报的分片混杂在一起。为了分清楚这些状况，重装模块使用了重装表及其相关联的链表，如我们在前面所讨论的。

重装模块的工作就是找出一个分片是属于哪个数据报的，将属于同一个数据报的各分片进行排序，并且当所有的分片都到达时把它们重新组装成一个数据报。如果预先设定的超时已到期，但还有分片没有到位，那么重装模块就把这些分片全部丢弃。

表 7.6 重装模块

```

1 IP_Reassembly_Module (Datagram)
2 {
3   If(分片偏移值是 0 且 M 位也是 0)
4   {
5     把数据报发送到适当的队列
6     返回
7   }
8   查找重装表中的相应表项
9   If(相应表项未找到)
10  {
11    创建一个新的项目

```

```

12    }
13    在链表的适当地方插入这个分片
14    If(所有的分片都已到达)
15    {
16        重装这些分片
17        把数据报交付给相应的高层协议
18        返回
19    }
20    Else
21    {
22        If(超时)
23        {
24            丢弃所有的分片
25            发送 ICMP 差错报文
26        }
27    }
28    返回
29 }

```

7.9 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

7.9.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]、[Kur & Ros 08]和[Gar & Vid 04]。

7.9.2 RFC

讨论 IPv4 协议的 RFC 包括：RFC 791、RFC 815、RFC 894、RFC 1122、RFC 2474 和 RFC2475。

7.10 重要术语

尽最大努力交付

检验和

码点	输出队列
数据报	ping
目的地址	指针字段
区分服务	优先级
选项结束选项	处理模块
进入点路由器	重装模块
转发模块	重装表
分片	记录路由选项
分片模块	服务类型
分片偏移	源地址
首部长度	严格的源路由选项
首部添加模块	生存时间
输入队列	时间戳选项
网际协议 (IP)	Traceroute
长度字段	类型字段
不严格的源路由选项	服务类型 (TOS)
最大传送单元 (MTU)	值字段
无操作选项	

7.11 本章小结

- IP 是不可靠的无连接协议，负责源点到终点的交付。在 IP 层的分组称为数据报。
- MTU 是数据链路协议能够封装的最大字节数。MTU 随着协议的不同而不同。分片就是把一个数据报划分为若干个更小的单元，以便能够适应数据链路层协议的 MTU。
- IP 数据报首部包括 20 字节的固定部分和最大长度为 40 字节的可变的选项部分。IP 首部中的选项部分用于网络测试和排错。六个 IP 选项各有其特定功能。
- IP 使用的差错检测方法是检验和。但是这个检验和只覆盖分组的首部，而不管数据部分。检验和使用反码算术运算把 IP 首部的相同长度的各段相加。相加结果取反码后存储在检验和字段中。接收端也使用反码算术运算来检查首部。
- IP 在 ATM 上运行使用了 ATM 网络的 AAL5 层。ATM 网络在进入点路由器和离去点路由器之间建立一条路由。通过 ATMARP，IP 分组的下一跳地址能够被映射为离去点路由器的物理地址。
- IP 软件包由以下一些构件组成：首部添加模块、处理模块、转发模块、分片模块、重装模块、路由表、MTU 表以及重装表。

7.12 实践安排

7.12.1 习题

1. IP 首部中有哪些字段在经过一个又一个路由器时不断变化着？
2. 若总长度为 1200 字节而其中的 1176 字节是来自高层的数据，试计算 HLEN 的值。
3. MTU 的范围从 296 到 65 535 不等。使用大的 MTU 有什么好处？使用小的 MTU 又有什么好处？
4. 已知一被分片的数据报的分片偏移为 120，你如何确定第一个和最后一个字节的编号？
5. 一个 IP 数据报必须经过路由器 128.46.10.5，而对经过的其他路由器则没有限制。试画出其 IP 选项，并给出它们的值。
6. 若时间戳选项的标志值是 1，试问能够被记录的路由器数的最大值是多少？为什么？
7. IP 分组首部长度的值能否小于 5？什么时候它正好等于 5？
8. 有一个 IP 数据报中的 HLEN 值为 7。试问其中有多少个选项字节？
9. 有一个 IP 数据报的选项字段长度为 20 字节。它的 HLEN 值是多少？用二进制来表示是多少？
10. 有一个 IP 数据报的总长度字段值为 36，首部长度字段值为 5。试问该分组携带了多少字节的数据？
11. 有一个数据报共携带了 1024 字节的数据。若没有选项信息，则首部长度字段值是多少？总长度字段值是多少？
12. 有一台主机正在向另一台主机发送 100 个数据报。若第一个数据报的标识号为 1024，那么最后一个数据报的标识号是多少？
13. 一个 IP 数据报在到达时其分片偏移是 0 且 M 位（还有分片）也是 0。这是第一个分片，还是中间的分片，或者是最后一个分片？
14. 一个 IP 数据报在到达时其分片偏移为 100。在这个分片的数据之前，源点已经发送了多少字节的数据？
15. 一个数据报在到达时其首部有如下的信息（十六进制表示）：

 a. 有无任何选项字段？
 b. 这个分组被分片了吗？
 c. 数据的长度是多少？
 d. 有没有使用检验和？
 e. 这个分组还能经过多少个路由器？
 f. 这个分组的标识号是多少？
 g. 服务类型是什么？

16. 在一个数据报中， M 位是0，HLEN是5，总长度值是200，分片偏移值是200。试问这个数据报的第一个字节的编号和最后一个字节的编号是多少？这是最后一个分片，还是第一个分片，或者是中间的分片？

7.12.2 研究活动

17. 试利用带有-R选项的实用程序 ping 检查一个分组到某个终点的路由选择。解释所得的结果。

18. 试利用带有-g选项的实用程序 traceroute 实现不严格的源路由选项。选择在源点和终点之间的一些路由器。解释所得的结果，并找出是否所有已指明的路由器都已经访问过。

19. 试利用带有-G选项的实用程序 traceroute 实现严格的源路由选项。选择在源点和终点之间的一些路由器。解释所得的结果，并找出是否所有已指明的路由器都已经访问过，同时没有访问过未指明的路由器。

第 8 章 地址解析协议 (ARP)

在 IP 协议能够把一个分组从源主机交付到目的主机之前，它首先要知道的是如何将这个分组交付给下一跳。如第 6 章所述，一个 IP 分组可以通过咨询路由表找出下一跳的 IP 地址。但是，既然 IP 使用的是数据链路层的服务，它就需要知道下一跳的物理地址。**通过一个称为地址解析协议 (ARP) 的协议就可以做到这一点，这也正是我们在本章要讨论的内容。**

目标

本章有以下几个目标：

- 区分用于网络层的逻辑地址（IP 地址）和用于数据链路层的物理地址（MAC 地址）。
- 描述如何静态地或动态地将逻辑地址映射为物理地址。
- 说明如何使用地址解析协议 (ARP) 来动态地将逻辑地址映射为物理地址。
- 说明如何使用代理 ARP 来产生子网划分的效果。
- 讨论 ATMARP，当底层网络是一个 ATM 广域网时就要用它来映射 IP 地址。
- 展示一个 ARP 软件包可以由五个构件组成：高速缓存表、队列、输出模块、输入模块和高速缓存控制模块。
- 给出 ARP 软件包中每个模块的伪码。

8.1 地址映射

互联网是由许多物理网络和一些像路由器这样的连网设备组成的。从源主机出发的分组在最终到达目的主机之前，可能要经过多个不同的物理网络。

在网络这一级，主机和路由器是通过它们的逻辑地址来识别的。逻辑地址就是互联网地址。它的管辖范围是全局的。逻辑地址在全局上是唯一的。之所以称为逻辑地址是因为它通常在软件中实现。每一个与互联网打交道的协议都需要用到逻辑地址。在 TCP/IP 协议族中，逻辑地址也称为 **IP 地址**，它的长度是 32 位。

但是，分组必须通过物理网络才能到达主机和路由器。在物理这一级，主机和路由器是用它们的物理地址来识别的。**物理地址**是一个本地地址。它的管辖范围是本地网络的。物理地址在本地范围内必须唯一，但在全局上没有此要求。之所以称为物理地址是因为物理地址通常（并非总是）在硬件上实现。物理地址的一个例子是以太网协议中 48 位的 MAC

地址，它被写入安装在主机或路由器中的网络接口卡(NIC)中。

物理地址和逻辑地址是两种不同的标识符。这两个地址都需要，因为一个物理网络(如以太网)可以同时为使用两种不同的协议的网络层提供服务，如IP和IPX(Novell公司)。同样地，网络层的分组(如IP)也可以通过不同的物理网络，如以太网和LocalTalk(Apple公司)网。

这就意味着把一个分组交付到主机或路由器需要用到两级地址：逻辑地址和物理地址。我们要把逻辑地址映射为相应的物理地址，反之亦然。这样的映射可通过静态映射或动态映射来完成。

8.1.1 静态映射

静态映射(static mapping)就是说要创建一张表，把逻辑地址与物理地址关联起来。这个表存储在网络中的每一台机器上。例如，知道其他机器的IP地址但却不知道其物理地址的机器就可以通过查表找出对应的物理地址。这样做有一些局限性，因为物理地址可能会因以下原因而发生变化：

1. 一台机器可能会更换其NIC，结果变成了一个新的物理地址。
2. 在某些局域网中，如LocalTalk，每当计算机加电时，它的物理地址都要改变一次。
3. 移动电脑可以从一个物理网络转移到另一个物理网络，这就会引起物理地址的改变。

要完成这些变化，静态映射表必须定期更新，而此类开销会影响网络的性能。

8.1.2 动态映射

动态映射(dynamic mapping)时，每次只要机器知道另一台机器的逻辑地址，就可以使用协议找出相应的物理地址。已设计出的实现了动态映射的协议有两个：**地址解析协议(ARP)**和**逆地址解析协议(RARP)**。ARP把逻辑地址映射为物理地址，RARP把物理地址映射为逻辑地址。因为RARP已经被另一个协议所取代而失去了价值，所以在本章我们只讨论ARP协议。

8.2 ARP协议

在任何时候，只要一台主机或路由器有IP数据报要发送给另一台主机或路由器，它就要知道接收方的逻辑(IP)地址。但是IP数据报必须封装成帧才能通过物理网络。这就意味着发送方还需要有接收方的物理地址，因而需要从逻辑地址到物理地址的映射。图8.1所示为ARP在TCP/IP协议族中的位置。ARP接受来自IP协议的逻辑地址，将其映射为相应的物理地址，然后再把这个物理地址递交给数据链路层。

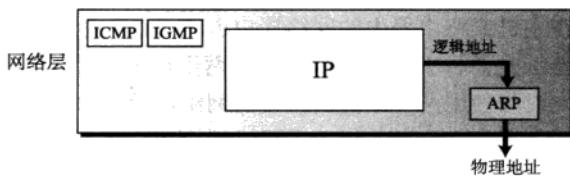


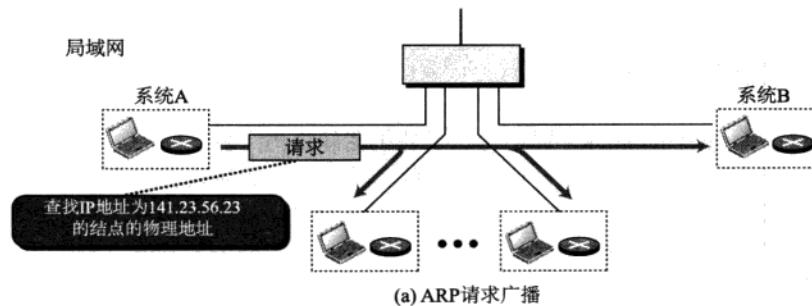
图 8.1 ARP 在 TCP/IP 协议族中的位置

ARP 将 IP 地址与其物理地址关联起来。在像局域网这样的一个典型的物理网络中，链路上的每一个设备通常是用写入 NIC 中的物理地址或站地址来标记的。

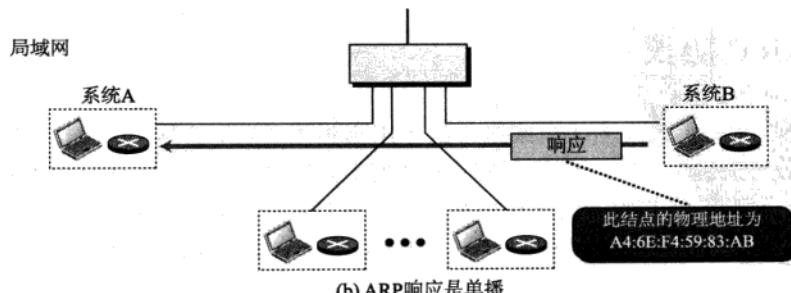
任何时候，当主机或路由器需要找出这个网络上的另一个主机或路由器的物理地址时，它就可以发送一个 ARP 查询分组。这个分组包括了发送方的物理地址和 IP 地址以及接收方的 IP 地址。因为发送方不知道接收方的物理地址，所以这个查询分组会在网络上进行广播（见图 8.2）。

网络上的每一台主机或路由器都会接收并处理这个 ARP 查询分组，但只有期待的接收方才能认出是自己的 IP 地址，并返回一个 ARP 响应分组。这个响应分组包含有接收方的 IP 地址和物理地址。这个分组利用收到的查询分组中的物理地址以单播方式直接发送给查询者。

在图 8.2 (a) 中，左边的系统 (A) 有一个分组需要交付给 IP 地址为 141.23.56.23 的另一个系统 (B)。系统 A 需要将分组传递给它的数据链路层进行实际的交付，但它却不知道接收方的物理地址。系统 A 使用 ARP 服务，请求 ARP 协议广播发送 ARP 请求分组，以查询 IP 地址为 141.23.56.23 的系统的物理地址。



(a) ARP请求广播



(b) ARP响应是单播

图 8.2 ARP 的操作

在这个物理网络上的每一个系统都会收到这个分组，但只有系统B才会回答，如图8.2(b)所示。系统B发送一个包含了它的物理地址的ARP回答分组。现在系统A就可以用收到的物理地址来发送所有给系统B的分组。

8.2.1 分组格式

图8.3给出了ARP分组的格式。其中各个字段的含义如下所示：

- **硬件类型** 这是一个16位字段，用来定义运行ARP的网络的类型。每一个局域网基于其类型被指派给一个整数。例如，以太网是类型1。ARP可用在任何物理网络上。
- **协议类型** 这是一个16位字段，用来定义使用的协议。例如，对IPv4协议，这个字段的值是 0800_{16} 。ARP可用于任何高层协议。
- **硬件长度** 这是一个8位字段，用来定义物理地址的长度，以字节为单位。例如，对于以太网这个值是6。
- **协议长度** 这是一个8位字段，用来定义逻辑地址的长度，以字节为单位。例如，对于IPv4协议这个值是4。

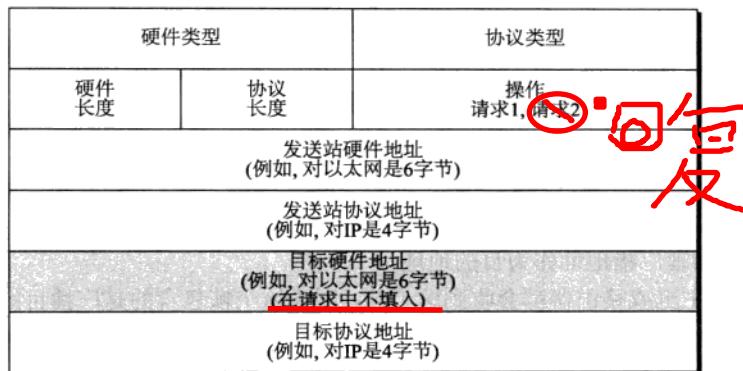


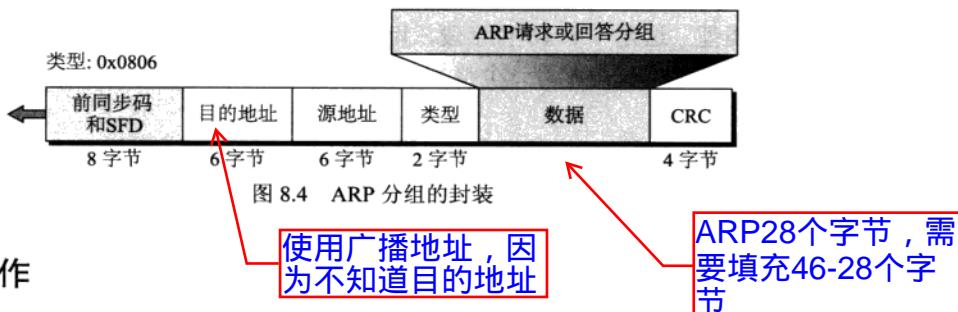
图8.3 ARP分组

- **操作** 这是一个16位字段，用来定义分组的类型。已定义的分组类型有两种：ARP请求(1)，ARP回答(2)。
- **发送方硬件地址** 这是一个可变长度字段，用来定义发送方的物理地址。例如，对于以太网这个字段的长度是6字节。
- **发送方协议地址** 这是一个可变长度字段，用来定义发送方的逻辑(例如，IP)地址。对于IP协议，这个字段的长度是4字节。
- **目标硬件地址** 这是一个可变长度字段，用来定义目标的物理地址。例如，对以太网来说这个字段是6字节长。对于ARP请求报文，这个字段是全0，因为发送方并不知道目标的物理地址。
- **目标协议地址** 这是一个可变长度字段，用来定义目标的逻辑地址(例如，IP地址)。

对于 IPv4 协议，这个字段的长度是 4 字节。

8.2.2 封装

ARP 分组直接封装在数据链路帧中。例如，在图 8.4 中，ARP 分组被封装在以太网的帧中。请注意，帧中的类型字段指出此帧所携带的数据是 ARP 分组。



8.2.3 操作

让我们来讨论在一个典型的互联网中 ARP 是怎样工作的。首先我们要描述其中的几个步骤，然后再讨论主机或路由器需要使用 ARP 的四种情况。

几个步骤

ARP 处理过程需要以下 7 个步骤：

1. 发送方知道目标的 IP 地址。
2. IP 请求 ARP 创建一个 ARP 请求报文，填入发送方的物理地址、发送方的 IP 地址以及目标 IP 地址。对于目标物理地址字段则全部填入 0。
3. 这个报文被递交给数据链路层。在这一层它被封装成帧，并以发送方的物理地址作为源地址，以物理广播地址作为目的地址。
4. 每一个主机或路由器都会收到这个帧，因为这个帧包含的是广播目的地址。所有站点都会取走这个报文并把它交给 ARP。除了目标机器之外，其他所有机器都会丢弃这个分组。目标机器认识这个 IP 地址。
5. 目标机器用 ARP 回答报文进行回答。回答报文中包含了它的物理地址。这个报文使用的是单播方式。
6. 发送方收到这个回答报文。现在发送方就知道目标机器的物理地址了。
7. 携带有给目标机器数据的 IP 数据报现在可以封装成帧，并用单播方式发送到终点。

ARP 请求采用广播发送； ARP 回答采用单播发送。

四种不同情况

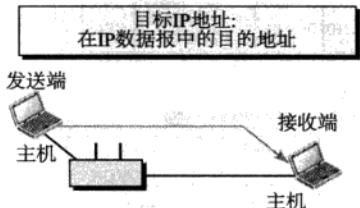
下面是可以使用 ARP 服务的四种不同情况（见图 8.5）。

情况 1 发送方是一台主机，它希望把分组发送给同一个网络上的另一台主机。

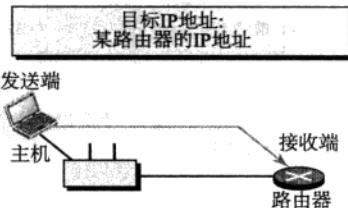
在这种情况下，必须被映射为物理地址的逻辑地址就是数据报首部中的目的 IP 地址。

- **情况2** 发送方是一台主机, 它希望把分组发送给另一个网络上的一台主机。在这种情况下, 这个主机要查找它的路由表, 找出到达这个终点的下一跳(路由器)的IP地址。如果它没有路由表, 就找出默认路由器的IP地址。这个路由器的IP地址就是必须被映射为物理地址的逻辑地址。
- **情况3** 发送方是路由器, 它收到了要发送给另一个网络上的主机的数据报。路由器先检查自己的路由表, 找出下一个路由器的IP地址。下一个路由器的IP地址就是必须被映射为物理地址的逻辑地址。
- **情况4** 发送方是路由器, 它收到了要发送给连接在同一个网络上的主机的数据报。数据报的目的IP地址就是必须被映射为物理地址的逻辑地址。

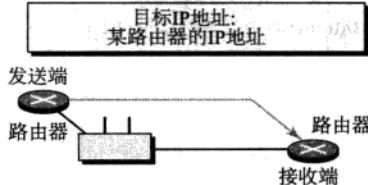
情况1: 主机有分组要发送给同一个网络上的另一台主机。



情况2: 主机有分组要发送给另一个网络上的一台主机。



情况3: 路由器有分组要发送给另一个网络上的一台主机。



情况4: 路由器有分组要发送给同一个网络上的一台主机。

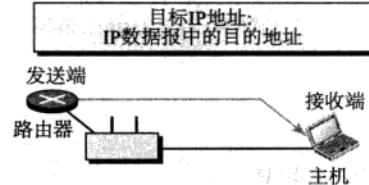


图 8.5 使用 ARP 的四种情况

例 8.1

IP 地址为 130.23.43.20 且物理地址为 B2:34:55:10:22:10 的主机有一个分组要发送给另一台主机, 这台主机的 IP 地址为 130.23.43.25, 物理地址为 A4:6E:F4:59:83:AB (第一个主机不知道这个物理地址)。这两台主机连接在同一个以太网上。试给出封装在以太网帧中的 ARP 请求和回答分组。

解

图 8.6 给出了 ARP 请求和回答分组。请注意, 在这种情况下 ARP 的数据字段是 28 字节, 而且其中各地址字段无法按每 4 字节为一段进行分隔, 这就是为什么我们没有使用通常的 4 字节边界来显示这些地址。还应注意到, IP 地址是用十六进制表示的。关于二进制和十六进制记法的更多信息请参见附录 B。

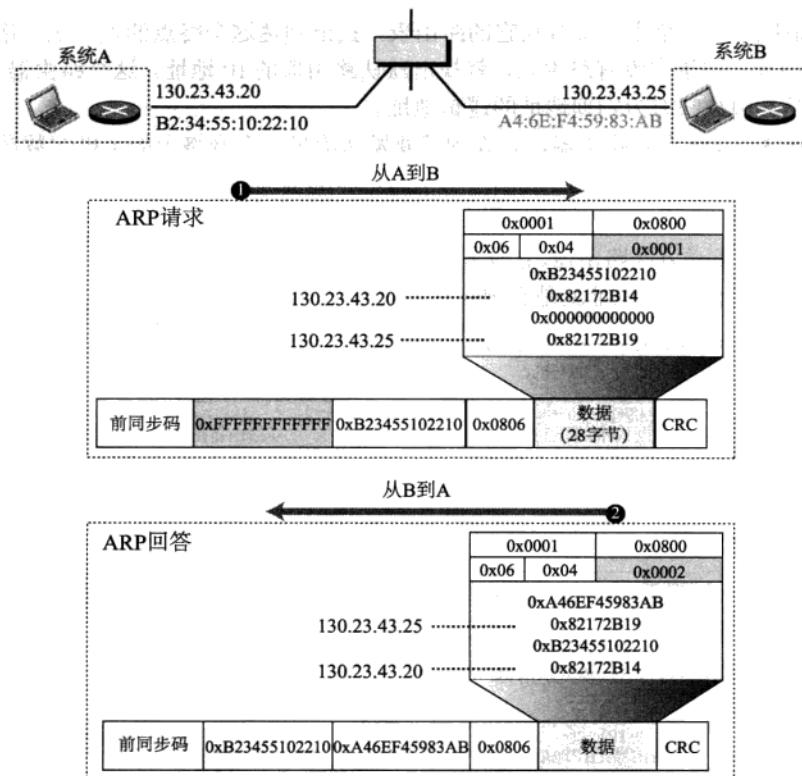


图 8.6 例 8.1 图

8.2.4 代理 ARP

有一种技术称为代理 ARP，可用于产生一种子网划分的效果。代理 ARP (proxy ARP) 是代表了一组主机的 ARP。当运行代理 ARP 的路由器收到一个 ARP 请求，希望找出这些主机中的某一台主机的物理地址时，路由器就返回一个宣布了它自己的硬件（物理）地址的 ARP 回答分组。在这个路由器收到真正的 IP 分组后，它会把这些分组发送给相应的主机或路由器。

让我们来举一个例子。在图 8.7 中，安装在右边主机上的 ARP 只能回答对目标 IP 地址为 141.23.56.23 的 ARP 请求。

但是，管理员有可能需要创建一个子网，并且要求在不改变整个系统的条件下能够识别出这个子网中的地址。解决这个问题的一种方法就是增加一个运行了代理 ARP 的路由器。在这种情况下，该路由器就代表了所有安装在子网上的主机。当它收到一个目标 IP 地址与它所代表的主机之一的地址相匹配（141.23.56.21、141.23.56.22 和 141.23.56.23）的 ARP 请求时，就返回一个 ARP 回答，声称它自己的硬件地址就是目标硬件地址。此后当路由器收到 IP 分组时，它会把分组发送到相应的主机。

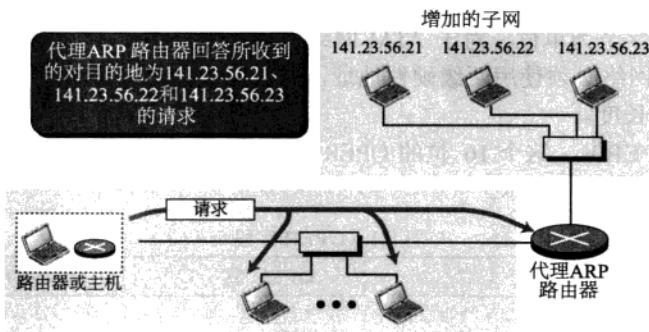


图 8.7 代理 ARP

8.3 ATMARP

我们在第7章中讨论了IP在ATM上运行。当IP分组要通过一个ATM广域网时，我们需要一种协议来找出（映射）ATM广域网中给定IP地址的离去点路由器的物理地址。这和我们在局域网上使用ARP所完成的任务一样。但是，局域网和ATM网络有一些区别。局域网是个广播网络（在数据链路层），ARP利用了局域网的广播能力来发送ARP请求。ATM网络不是一个广播网络，因此要处理这个任务需要其他的解决方法。

8.3.1 分组格式

ATMARP分组的格式和ARP分组的相似，如图8.8所示。这些字段如下：

- 硬件类型 (HTYPE)** 这个16位的HTYPE字段定义物理网络的类型。对于ATM网络，这个值是 0013_{16} 。
- 协议类型 (PTYPE)** 这个16位的PTYPE字段定义协议的类型。对于IPv4协议，这个值是 0800_{16} 。



图 8.8 ATMARP 分组

- **发送方硬件长度 (SHLEN)** 这个 8 位的 SHLEN 字段定义了发送方物理地址的长度, 以字节为单位。对于 ATM 网络, 这个值是 20。请注意, 如果绑定要跨越的 ATM 网络必须使用两级硬件地址, 那么旁边的 8 位保留字段可用来定义第二个地址的长度。
- **操作 (OPER)** 这个 16 位的 OPER 字段定义了分组的类型。已定义的分组类型有五种, 如表 8.1 所示。

表 8.1 OPER 字段

报文	OPER 值
请求	1
回答	2
反向请求	8
反向回答	9
NACK	10

- **发送方协议长度 (SPLLEN)** 这个 8 位的 SPLLEN 字段定义了发送方协议地址的长度, 以字节为单位。对于 IPv4 这个值是 4。
- **目标硬件长度 (TLEN)** 这个 8 位的 TLEN 字段定义了接收方物理地址的长度, 以字节为单位。对于 ATM 网络这个值是 20。请注意, 如果绑定要跨越的 ATM 网络必须使用两级硬件地址, 那么旁边的 8 位保留字段可用来定义第二个地址的长度。
- **目标协议长度 (TPLEN)** 这个 8 位的 TPLEN 字段定义了接收方协议地址的长度, 以字节为单位。对于 IPv4 这个值是 4。
- **发送方硬件地址 (SHA)** 这个可变长度的 SHA 字段定义了发送方的物理地址。对于 ATM 网络, ATM 论坛把这个值定义为 20 字节。
- **发送方协议地址 (SPA)** 这个可变长度的 SPA 字段定义了发送方的协议地址。对于 IPv4, 这个字段的长度是 4 字节。
- **目标硬件地址 (THA)** 这个可变长度的 THA 字段定义了接收方的物理地址。对于 ATM 网络, ATM 论坛把这个值定义为 20 字节。在请求报文中这个字段是空的, 在回答报文和 NACK 报文中, 这个字段被填入值。
- **目标协议地址 (TPA)** 这个可变长度的 TPA 字段定义了接收方的协议地址。对于 IPv4, 这个字段的长度是 4 字节。

8.3.2 ATMARP 的操作

有两种方法可以连接 ATM 网络上的两个路由器: 通过永久虚电路 (PVC) 或通过交换虚电路 (SVC)。ATMARP 的操作取决于连接方法。

PVC 连接

永久虚电路 (PVC) 连接是由网络提供者在两个端点之间建立的。这些永久连接都指定了 VPI 和 VCI, 且它们的数值被保存在每一个交换机的路由表中。

如果永久虚电路是在两个路由器之间建立的，那么就不需要 ATMARP 服务器。但是，路由器必须能够把物理地址和 IP 地址绑定起来。在进行绑定时要使用反向请求报文(inverse request message)和反向回答报文(inverse reply message)。在为一个路由器建立 PVC 时，该路由器要发送一个反向请求报文。位于连接的另一端的路由器接收到这个报文(这个报文含有发送方的 IP 地址和物理地址)，然后返回一个反向回答报文(这个报文包含了它自己的物理地址和 IP 地址)。

交换过报文之后，这两个路由器都增加了一个表项，将物理地址映射到 PVC。现在，当路由器收到 IP 数据报时，这张表就能够提供信息，使路由器可以用虚电路标识符对这个数据报进行封装。图 8.9 所示为两个路由器之间的报文交换情况。

在 PVC 的情况下，反向请求报文和反向回答报文可以把物理地址和 IP 地址绑定起来。

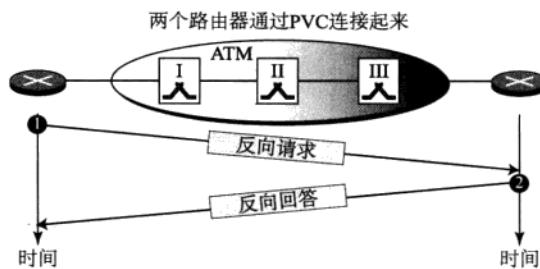


图 8.9 PVC 的绑定过程

SVC 连接

在使用交换虚电路(SVC)连接的情况下，路由器每一次想和另一个路由器(或任何主机)建立连接时，都必须建立一条新的虚电路。但是，仅当进入点路由器知道离去点路由器的物理地址时，这条虚电路才能建立(ATM 不认识 IP 地址)。

要把 IP 地址映射为物理地址，每一个路由器必须运行客户 ATMARP 程序，但只有一个计算机运行 ATMARP 服务器程序。要了解 ARP 和 ATMARP 的区别，我们应当首先想到 ARP 运行在局域网上，而局域网是广播网络。ARP 客户程序会广播 ARP 请求报文，网络上的每一个路由器都能收到它，而只有目标路由器会响应。ATM 则是非广播网络，一个 ATMARP 请求报文不可能到达这个网络上的所有路由器。

建立虚连接的过程需要 3 个步骤：连接服务器、收到物理地址、建立连接。图 8.10 描绘了这些步骤。

连接服务器 通常，每一个路由器和服务器之间都建立了一条永久虚电路。如果在路由器和服务器之间没有 PVC 连接，那么为了交换 ATMARP 请求和回答报文，服务器必须至少知道要建立 SVC 连接的路由器的物理地址。

收到物理地址 如果进入点路由器和服务器之间有一条连接，那么路由器就向服务器发送 ATMARP 请求报文。如果服务器能够找到相应的物理地址，就返回 ATMARP 回答报文，否则就返回 ATMARP NACK 报文。如果进入点路由器收到 NACK 报文，就丢弃这个数据报。

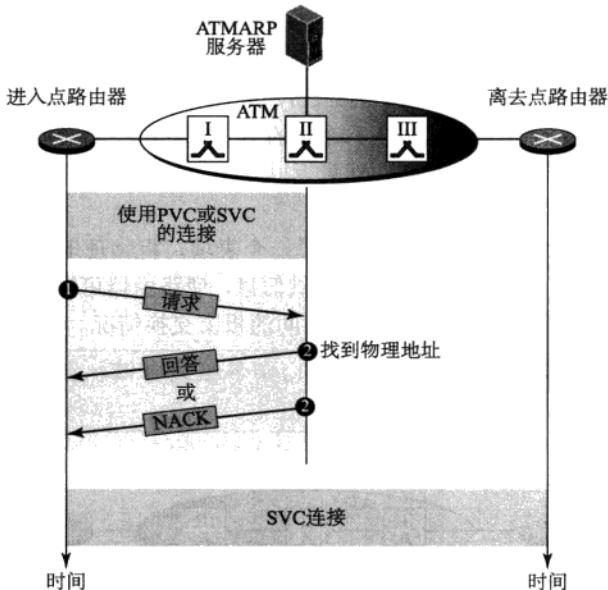


图 8.10 用 ATMARP 进行绑定

建立虚电路 在进入点路由器收到离去点路由器的物理地址后，它就能够请求在自己和离去点路由器之间建立一条 SVC。ATM 网络利用这两个物理地址建立虚电路，这条虚电路一直持续到进入点路由器要求断开连接时为止。在这一步，网络中的每一个交换机在它们的路由表中都要增加一个表项，以便能够转发携带 IP 数据报的信元。

在 SVC 的情况下，用请求报文和回答报文可以将物理地址和 IP 地址绑定。

建立映射表

ATM 服务器怎样建立它的映射表呢？也是通过使用 ATMARP 以及两个反向报文（反向请求和反向回答）。当路由器第一次连接到 ATM 网络上时，在路由器和服务器之间就建立起一条永久虚电路，服务器向该路由器发送一个反向请求报文，路由器则返回一个反向回答报文，并且在这个反向回答报文中包含了自身的 IP 地址和物理地址。有了这两个地址，服务器就能在它的路由表中建立一个表项，以便今后当这个路由器作为离去点路由器时能够用得上。图 8.11 给出了 ATMARP 的反向操作。

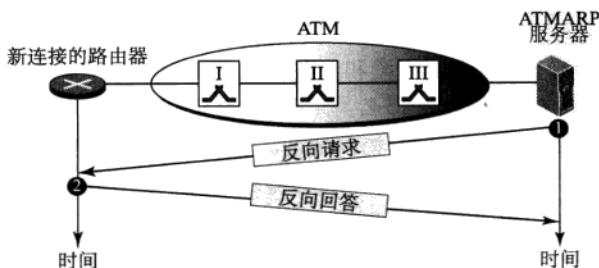


图 8.11 建立映射表

反向请求报文和反向回答报文也可用来构建服务器的映射表。

8.3.3 逻辑 IP 子网 (LIS)

在我们离开 IP 在 ATM 上运行这个题目之前，还要讨论一个称为逻辑 IP 子网 (logical IP subnet, LIS) 的概念。我们知道，一个大的局域网可以划分为若干个子网，基于同样的原因，一个 ATM 网络也可以划分为若干个逻辑上的（而不是物理上的）子网。这样做是为了方便 ATMARP 以及其他一些需要在 ATM 网络上模拟广播操作的协议（如 IGMP）。

连接在 ATM 网络上的路由器可以属于一个或多个逻辑子网，如图 8.12 所示。在这个图中，路由器 B、C 和 D 都属于一个逻辑子网（用虚线方框表示），而路由器 F、G 和 H 则属于另一个逻辑子网（用有阴影的方框表示）。路由器 A 和 E 则属于这两个逻辑子网。一个路由器可以直接和在同一个子网中的另一个路由器通信，并直接把 IP 分组发送过去，但是，如果它要向属于另一个子网中的路由器发送分组，就必须先把这个分组发送到属于这两个子网的路由器。例如，路由器 B 可以直接把分组发送给路由器 C 和 D，但是从 B 发送到 F 的分组就要先通过 A 或 E。

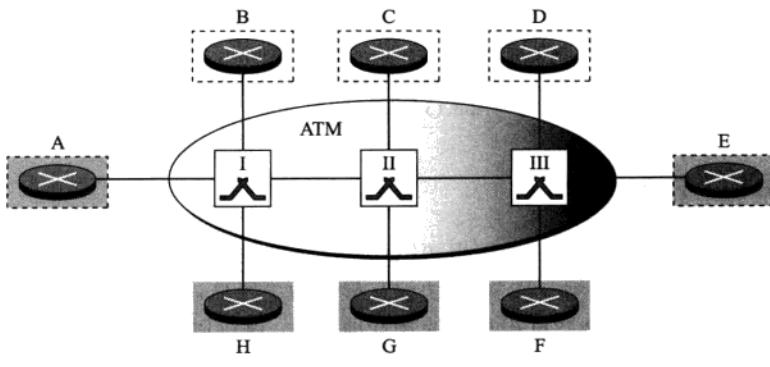


图 8.12 LIS

请注意，属于同一个逻辑子网的路由器共享相同的前缀和子网掩码，而不同子网的路由器的前缀各不相同。

要使用 ATMARP，每一个子网必须有各自的 ATMARP 服务器。例如，在图 8.11 中，我们需要两个 ATMARP 服务器，每个子网一个。

LIS 允许把 ATM 网络划分成若干个逻辑子网。要使用 ATMARP，各个子网中都需要一个独立的服务器。

8.4 ARP 软件包

在本节我们给出一个简化的 ARP 软件包的例子，目的是展示一个假想的 ARP 软件包

中所包含的构件以及这些构件之间的关系。图 8.13 所示为这些构件和它们之间的交互关系。

我们可以认为 ARP 软件包由以下五个构件组成：高速缓存表（cache table）、队列、输出模块、输入模块和高速缓存控制模块。这个软件包接收的是即将被封装成帧而需要一个目的物理（硬件）地址的 IP 数据报。如果 ARP 软件包找到了这个物理地址，它就把这个 IP 分组和物理地址一起交付给数据链路层以便传输。

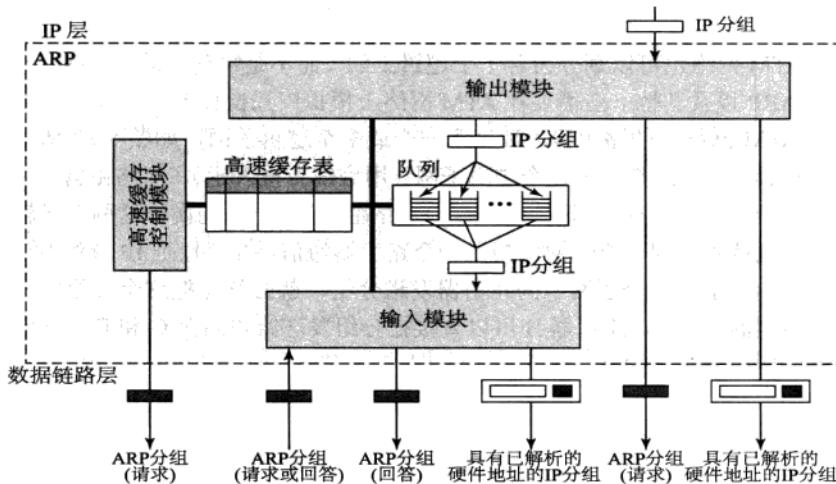


图 8.13 ARP 的构件

8.4.1 高速缓存表

发送方往往有一个以上的 IP 数据报要发送到同一个终点。如果对发送到同一个主机或路由器的每一个数据报都使用一次 ARP 协议，显然效率很低。解决方法就是高速缓存表。当主机或路由器收到一个 IP 数据报相应的物理地址时，就可以把这个物理地址存储在高速缓存表中。在之后的几分钟内，发往相同目的地的数据报都可以使用这个地址。但是，由于高速缓存表的空间非常有限，表中的映射关系不能无时间限制地保存。

高速缓存表以表项数组的形式实现。在我们的软件包中，每个表项包含以下一些字段：

- **状态** 这一列显示的是表项的状态。它可以是以下三个值之一：FREE、PENDING 或 RESOLVED。FREE 状态表示这个项目的生存时间已到期。这个空间可给新的表项使用。PENDING 状态表示为这个表项的请求已经发出，但回答还未收到。RESOLVED 状态表示这个表项已完成映射，它现在已经有了终点的物理（硬件）地址，等待发送到这个终点的分组可以使用这个表项的信息。
- **硬件类型** 这一列与 ARP 分组中的相应字段相同。
- **协议类型** 这一列与 ARP 分组中的相应字段相同。
- **硬件长度** 这一列与 ARP 分组中的相应字段相同。
- **协议长度** 这一列与 ARP 分组中的相应字段相同。
- **接口号** 一个路由器（或连接多个网络的主机）能够连接到多个不同的网络，每个

连接都有不同的接口号。每个网络还可能有不同的硬件类型和协议类型。

- **队列号** ARP 使用带编号的队列把等待地址解析的分组进行排队。发往同一个终点的分组通常放在同一个队列中。
- **尝试** 这一列表示已经为此表项发送了多少次的 ARP 请求。
- **超时** 这一列表示此表项的生存时间，以秒为单位。
- **硬件地址** 这一列表示目的硬件地址。它一直是空的，直到被 ARP 回答解析出来为止。
- **协议地址** 这一列表示目的 IP 地址。

8.4.2 队列

我们的 ARP 软件包维持着一组队列，每个队列对应于一个终点，用来在 ARP 尝试解析硬件地址时保留 IP 分组。输出模块把未解析的分组发送到相应的队列（queue）。输入模块从队列中取出分组，并连同解析出的物理地址一起发送给数据链路层传输。

8.4.3 输出模块

表 8.2 以伪码的形式展示了输出模块的实现。

表 8.2 输出模块

```

1  ARP_Output_Module()
2  {
3      睡眠，直至收到来自 IP 软件的 IP 分组
4      检查高速缓存表，寻找这个 IP 分组终点对应的表项
5      If(找到)
6      {
7          If(状态是 RESOLVED)
8          {
9              从该表项中提取出硬件地址的值
10             把分组连同硬件地址一起发送到数据链路层
11             返回
12         } //end if
13         If(状态是 PENDING)
14         {
15             把分组放入相应的队列
16             返回
17         } //end if
18     } //end if
19     If(未找到)

```

```

20    {
21        创建一个高速缓存表项，状态置为 PENDING，ATTEMPTS 置 1
22        创建一个队列
23        把分组放入队列
24        发送一个 ARP 请求
25        返回
26    } //end if
27 } //end module

```

输出模块等待来自 IP 软件的 IP 分组。输出模块检查高速缓存表，查找是否有某个表项对应于这个分组的目的 IP 地址。这个 IP 分组的目的 IP 地址必须与这个表项的协议地址相匹配。

如果找到了这样的表项，且其状态是 RESOLVED，则把这个分组连同目的硬件地址一起传递给数据链路层去传输。

如果找到了这样的表项，且其状态是 PENDING，则这个分组必须等待，直到获得目的硬件地址。因为状态为 PENDING，所以这个终点已经创建了一个队列，这个模块把分组放入相应的队列中。

如果找不到表项，输出模块就创建一个队列，并把这个分组放入队列中。这时要为这个终点创建一个状态为 PENDING 的表项，并把 ATTEMPTS 字段的值置为 1，然后用广播方式发送 ARP 请求分组。

8.4.4 输入模块

表 8.3 以伪码的形式展示了输入模块的实现。

表 8.3 输入模块

```

1 ARP_Input_Module()
2 {
3     睡眠，直至一个 ARP 分组（请求或回答）到达
4     检查高速缓存表，寻找相应的表项
5     If(找到)
6     {
7         更新这个表项
8         If(状态是 PENDING)
9         {
10            While (相应的队列非空)
11            {
12                从队列中取出一个分组
13                将该分组连同硬件地址一起发送

```

```

14          } //end if
15      } //end if
16  } //end if
17  If(未找到)
18  {
19      创建一个表项
20      把新建的表项加入到表中
21  } //end if
22  If(这个分组是请求分组)
23  {
24      发送 ARP 回答
25  } //end if
26  返回
27 } //end module

```

输入模块等待，直到有 ARP 分组（请求或回答）到达。输入模块检查高速缓存表，寻找对应于这个 ARP 分组的表项，其目标协议地址应当与表项的协议地址相匹配。

如果找到这样的表项，且其状态是 PENDING，则模块对这个表项进行更新，即把分组中的目标硬件地址复制到表项中的硬件地址字段，并把状态改变为 RESOLVED。输入模块还要设置这个表项的超时时间值 TIME-OUT。然后，它从相应的队列中把分组逐个取出，连同其硬件地址一起交付给数据链路层去传输。

如果找到这样的表项，且其状态是 RESOLVED，输入模块还是要更新该表项。这是因为目标硬件地址可能已经改变了。TIME-OUT 字段的值也要重置。

如果没有找到相应的表项，输入模块就创建一个新的表项，并把它加入到表中。协议要求任何收到的信息都要添加到表中以供今后使用。此时状态应置为 RESOLVED，且 TIME-OUT 字段的值也要设置。

现在，输入模块要查看到达的 ARP 分组是不是请求分组。如果是，它就立即创建一个 ARP 回答分组并发送给对方。ARP 回答分组的建立是通过将请求分组中的操作字段从请求改为回答，并填入目标硬件地址来实现的。

8.4.5 高速缓存控制模块

高速缓存控制模块（cache-control module）负责维护高速缓存表。它周期性地（例如，每隔 5 秒钟）逐项检查高速缓存表。如果表项的状态为 FREE，就继续查看下一个表项。如果表项的状态为 PENDING，则把尝试字段的值加 1，然后再检查尝试字段的值。如果这个值大于允许尝试的最大数，则把状态改为 FREE，并销毁相应的队列。但是，如果尝试值小于最大数，则要创建并再次发送一个 ARP 请求分组。

如果表项的状态是 RESOLVED，则模块把超时字段的值减去自上次检查以来所经过的时间。若这个数值小于或等于零，则把状态改变为 FREE，并销毁相应的队列。表 8.4 以伪

码的形式展示了高速缓存控制模块的实现。

表 8.4 高速缓存控制模块

```

1 ARP_Cache_Control_Module( )
2 {
3     睡眠, 直至计时器到时
4     Repeat for 高速缓存中的每一个表项
5     {
6         If(状态为 FREE)
7         {
8             继续
9         } //end if
10        If(状态是 PENDING)
11        {
12            把尝试值加 1
13            If(尝试大于最大数)
14            {
15                把状态改变为 FREE
16                撤销相应的队列
17            } //end if
18            Else
19            {
20                发送一个 ARP 请求
21            } //end else
22            继续
23        } //end if
24        If(状态为 RESOLVED)
25        {
26            递减超时字段的值
27            If(超时字段的值小于或等于零)
28            {
29                把状态改变为 FREE
30                撤销相应的队列
31            } //end if
32        } //end if
33    } //end repeat
34    返回
35 } //end module

```

8.4.6 更多的例子

在本节，我们要给出一些 ARP 的操作以及高速缓存表的内容如何变化的例子。表 8.5 所示为我们的例子刚开始时高速缓存表中的一些字段。

表 8.5 用在例子中的原始高速缓存表

状态	队列	尝试 (Attempt)	超时 (Time-Out)	协议地址	硬件地址
R	5		900	180.3.6.1	ACAE32457342
P	2	2		129.34.4.8	
P	14	5		201.11.56.7	
R	8		450	114.5.7.89	457342ACAE32
P	12	1		220.55.5.7	
F					
R	9		60	19.1.7.82	4573E3242ACA
P	18	3		188.11.8.71	

例 8.2

ARP 输出模块收到一个 IP 数据报（来自 IP 层），其目的地址为 114.5.7.89。输出模块检查高速缓存表，发现存在与这个终点相匹配的表项，且状态为 RESOLVED（表中的 R）。输出模块提取硬件地址（457342ACAE32），并把这个分组和地址一起传递给数据链路层传输。高速缓存表保持不变。

例 8.3

20 秒后，ARP 输出模块又收到一个 IP 数据报（来自 IP 层），其目的地址为 116.1.7.22。输出模块检查高速缓存表，在表中未发现这个终点。输出模块在表中增加一个表项，状态是 PENDING，尝试值是 1。输出模块为这个终点创建一个新的队列，并把分组放入这个队列中，然后它就为这个终点向数据链路层传递一个 ARP 请求。新的高速缓存表见表 8.6 所示。

表 8.6 例 8.3 中已更新的高速缓存表

状态	队列	尝试	超时	协议地址	硬件地址
R	5		900	180.3.6.1	ACAE32457342
P	2	2		129.34.4.8	
P	14	5		201.11.56.7	
R	8		450	114.5.7.89	457342ACAE32
P	12	1		220.55.5.7	
P	23	1		116.1.7.22	
R	9		60	19.1.7.82	4573E3242ACA
P	18	3		188.11.8.71	

例 8.4

过了 15 秒后，ARP 输入模块收到一个 ARP 分组，其目标协议（IP）地址为 188.11.8.71。

输入模块检查高速缓存表，找到了这个地址。它把该表项的状态改变为 RESOLVED，并把超时值置为 900。然后，输入模块把目标地址（E34573242ACA）添加到该表项中。现在它可以访问队列 18 并将队列中的所有分组逐个地发送到数据链路层。新的高速缓存表见表 8.7 所示。

表 8.7 例 8.4 更新后的高速缓存表

状态	队列	尝试	超时	协议地址	硬件地址
R	5		900	180.3.6.1	ACAE32457342
P	2	2		129.34.4.8	
P	14	5		201.11.56.7	
R	8		450	114.5.7.89	457342ACAE32
P	12	1		220.55.5.7	
P	23	1		116.1.7.22	
R	9		60	19.1.7.82	4573E3242ACA
R	18		900	188.11.8.71	E34573242ACA

例 8.5

又过了 25 秒后，高速缓存控制模块开始更新每个表项。前三个已解析的表项中超时值减去 60。最后一个已解析的表项中超时值减去 25。倒数第二个表项的状态改为 FREE，因为超时时间等于零。对于三个等待中的表项，每一个的尝试字段的值加 1。在加 1 之后，有一个表项（其 IP 协议地址为 201.11.56.7）的尝试值超过了最大值，它的状态改为 FREE，队列被删除，同时向原来的终点发送一个 ICMP 报文（见第 9 章），参见表 8.8。

表 8.8 例 8.5 更新后的高速缓存表

状态	队列	尝试	超时	协议地址	硬件地址
R	5		840	180.3.6.1	ACAE32457342
P	2	3		129.34.4.8	
F					
R	8		390	114.5.7.89	457342ACAE32
P	12	2		220.55.5.7	
P	23	2		116.1.7.22	
F					
R	18		875	188.11.8.71	E34573242ACA

8.5 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

8.5.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]和[Ste 94]。

8.5.2 RFC

专门讨论 ARP 的 RFC 包括：RFC 826、RFC 1029、RFC 1166 和 RFC1981。

8.6 重要术语

地址解析协议 (ARP)	逻辑 IP 子网 (LIS)
高速缓存表	物理地址
高速缓存控制模块	代理 ARP
动态映射	队列
封装	保留字段
反向回答报文	逆地址解析协议 (RARP)
反向请求报文	静态映射
IP 地址	

8.7 本章小结

- 把分组交付给主机或路由器需要有两级地址：逻辑地址和物理地址。逻辑地址在网络级标记每一台主机或路由器。TCP/IP 称这种逻辑地址为 IP 地址。物理地址在物理级标记每一台主机或路由器。
- 逻辑地址到物理地址的映射可以是静态的，也可以是动态的。静态映射涉及到一张映射逻辑地址与物理地址的列表，维护这种表需要很大的开销。
- 地址解析协议 (ARP) 是一种动态映射方法，它为给定的逻辑地址找出对应的物理地址。ARP 请求用广播方式发送给网络上的所有设备。ARP 回答用单播方式发送给请求映射的主机。
- 使用代理 ARP 时，路由器代表了一组主机。当 ARP 请求寻找该组中任意主机的物理地址时，这个路由器就将自己的物理地址发送过去。这样做可以产生子网划分的效果。
- ATMARP 是在 ATM 网络上使用的协议，它用来把物理地址和 IP 地址进行绑定。构建 ATMARP 服务器的映射表需要使用反向请求报文和反向回答报文。一个 ATM 网络可以划分为若干个逻辑子网，以便于 ATMARP 和其他协议的操作。
- ARP 软件包由五个构件组成：高速缓存表、队列、输出模块、输入模块以及高速缓存控制模块。高速缓存表有一个表项数组，由 ARP 报文使用和更新。在一个队列中包含的是要发送到相同终点去的分组。输出模块从 IP 层得到分组，然后或者把它发送到数据链路层，或者发送到队列中。输入模块使用 ARP 分组来更新高速缓存表。输入模块也可以发送 ARP 回答。高速缓存控制模块通过更新表项字段来维护高速缓存表。

8.8 实践安排

8.8.1 习题

1. ARP 分组的长度是固定的吗？试加以解释。
2. 当协议是 IP 且硬件是以太网时，ARP 分组的长度是多少？
3. 携带 ARP 分组的以太网帧的长度是多少？
4. 以太网的广播地址是什么？
5. 某路由器的 IP 地址是 125.45.23.12 且所在的以太网物理地址是 23:45:AB:4F:67:CD，它收到了一个分组，分组的目的 IP 地址是 125.11.78.10，对应的以太网物理地址是 AA:BB:A2:4F:67:CD。
 - a. 试给出这个路由器发出的 ARP 请求分组的各个表项。假定无子网划分。
 - b. 试给出对 a 题中的请求分组进行响应的 ARP 回答分组的各个表项。
 - c. 将 a 题中的分组封装到数据链路帧中，请填写所有的字段。
 - d. 将 b 题中的分组封装到数据链路帧中，请填写所有的字段。
6. 某路由器的 IP 地址是 195.5.2.12 且所在的以太网物理地址是 AA:25:AB:1F:67:CD，它收到了目的 IP 地址是 185.11.78.10 的分组。当路由器检查自己的路由表时，发现分组应交付给 IP 地址为 195.5.2.6 且以太网物理地址是 AD:34:5D:4F:67:CD 的路由器。
 - a. 试给出由这个路由器发出的 ARP 请求分组中的各个表项。假定无子网划分。
 - b. 试给出对 a 题中的请求分组进行响应的 ARP 回答分组的各个表项。
 - c. 将 a 题中的分组封装到数据链路帧中，请填写所有的字段。
 - d. 将 b 题中的分组封装到数据链路帧中，请填写所有的字段。
7. 两个路由器之间有一条 PVC 连接，试给出在这两个路由器之间交换的 ATMARP 反向分组的内容。已知它们的 IP 地址分别是 172.14.20.16/16 和 180.25.23.14/24。任意选择两个 20 字节的物理地址。在填入这些字段时使用十六进制。
8. 试给出在路由器和服务器之间交换的 ATMARP 分组（请求和回答）的内容。路由器的 IP 地址是 14.56.12.8/16，而服务器的 IP 地址是 200.23.54.8/24。任意选择两个 20 字节的物理地址。在填入这些字段时使用十六进制。
9. 试给图 8.12 中的路由器加上 IP 地址。应当注意，各个 LIS 中的前缀必须是相同的，但两个不同的 LIS 其前缀则必须是不同的。还应当注意，属于两个 LIS 的路由器必须有两个 IP 地址。
10. ATMARP 分组必须由信元来运送。在本章中讨论的 ATMARP 分组需要多少个信元来运送？

第 9 章 网际控制报文协议 (ICMP)

如 第 7 章所讨论的，IPv4 提供了不可靠的和无连接的数据报交付。当初这样设计是为了有效地利用网络资源。IP 协议是尽最大努力的服务，它把数据报从最初的源点交付到最后的终点。但是，它有两个缺点：缺少差错控制和缺少辅助机制。ICMPv4 就是为了补偿这两个缺点而设计的。

目标

本章有以下几个目标：

- 讨论 ICMP 存在的理由。
- 说明 ICMP 报文如何划分为两大类：差错报告和查询报文。
- 讨论差错报告的作用和格式。
- 讨论查询报文的作用和格式。
- 说明 ICMP 报文中的检验和是如何计算的。
- 说明一些排错工具是如何利用 ICMP 协议的。
- 说明实现 ICMP 的一个简单的软件包是怎样组织的。

9.1 引言

IP 协议没有差错报告或差错纠正机制。如果出了一些差错将会发生什么问题呢？如果路由器因找不到可到达最后终点的路由器，或者因生存时间字段的值为零而必须丢弃数据报时，该怎么办呢？如果最终目的主机在预先设定的时间内没有收到一个数据报的所有分片，因而必须把已接收到的分片全部丢弃时，又该怎么办呢？这些都是因出现了差错而 IP 协议却没有内建的机制可以通知发出该数据报的主机的例子。

IP 协议还缺少主机和管理查询所需要的机制。主机有时需要判断某个路由器或者是对方主机是否活跃。有时网络管理员也需要来自其他主机或路由器的信息。

网际控制报文协议 (ICMP) 是设计来弥补上述两个缺憾的，它是 IP 协议的伴侣。图 9.1 给出了 ICMP 在网络层的位置，以及它与 IP 及其他协议之间的关系。

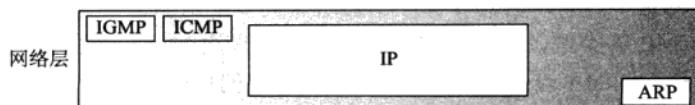


图 9.1 ICMP 在网络层中的位置

ICMP 本身是一个网络层协议。但是，它的报文并不是如预期的那样直接传递给数据链路层。实际上，ICMP 报文首先要封装成 IP 数据报，然后才被传递到下一层（见图 9.2）。

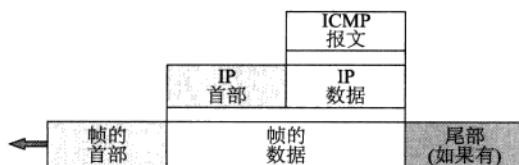


图 9.2 ICMP 的封装

在一个 IP 数据报中，如果协议字段值是 1，就表示 IP 数据是 ICMP 报文。

9.2 报文

ICMP 报文可划分为两大类：差错报告报文(error-reporting messages)和查询报文(query messages)。差错报告报文报告了路由器或主机(终点)在处理 IP 数据报时可能遇到的问题。查询报文总是成双成对地出现，它帮助主机或网络管理员从某个路由器或对方主机那里获取特定的信息。例如，结点可以去发现它们的邻站。同样，主机也可以发现和了解它们所在的网络上的一些路由器的情况，而路由器又能够帮助一个结点改变报文的路由。表 9.1 列出了每一类 ICMP 报文。

表 9.1 ICMP 报文

种类	类型	报文
差错报告报文	3	终点不可达
	4	源点抑制
	11	超时
	12	参数问题
	5	改变路由
	8 或 0	回送请求或回答
	13 或 14	时间戳请求或回答

9.2.1 报文格式

ICMP 报文由一个 8 字节的首部和可变长度的数据部分组成。虽然对每一种报文类型，首部的一般格式都是不同的，但前 4 个字节对所有类型来说都是相同的。如图 9.3 所示，第一个字段是 ICMP 的类型，它定义了报文类型。代码字段指明了发送这个特定报文类型的原因。最后一个共同的字段是检验和字段（将在本章的后面讨论）。首部的其余部分对于每一种类型的报文都是特有的。

在差错报文中，数据部分携带的是用于找出引起差错的原始分组的信息。在查询报文中，数据部分携带的是基于查询类型的额外信息。

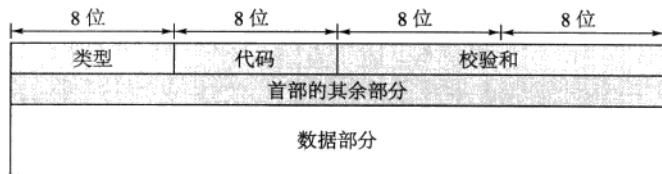


图 9.3 ICMP 报文的一般格式

9.2.2 差错报告报文

ICMP 的主要责任之一就是报告差错。虽然现代技术已经制造出很可靠的传输媒体，但差错仍然存在且必须被处理。如在第 7 章中讨论的，IP 是一个不可靠的协议。这就表明 IP 并不考虑差错检验和差错控制。之所以设计 ICMP 的部分原因就是为了弥补这个缺点。不过，ICMP 不能纠正差错，它只是简单地报告差错。差错纠正留给高层协议去做。差错报文总是发送给最初的数据源，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 利用了源 IP 地址把差错报文发送给数据报的源点（发出者）。

ICMP 总是把差错报文报告给最初的数据源。

一共有 5 种类型的差错要处理：终点不可达、源点抑制、超时、参数问题以及改变路由（见图 9.4）。



图 9.4 差错报告报文

以下是关于 ICMP 差错报文的一些要点：

- 对于携带 ICMP 差错报文的数据报，不再产生 ICMP 差错报文。
- 对于分片的数据报，如果不是第一个分片，则不产生 ICMP 差错报文。
- 对于具有多播地址的数据报，不产生 ICMP 差错报文。
- 对于具有特殊地址（如 127.0.0.0 或 0.0.0.0）的数据报，不产生 ICMP 差错报文。

请注意，所有的差错报文都包含了一个数据部分，它包括的是原始数据报的 IP 首部再加上该数据报数据的前 8 个字节。加上原始数据报的首部是为了向接收差错报文的原始信源给出关于数据报本身的信息。之所以要包含数据的前 8 个字节是因为这前 8 个字节提供了关于端口号（UDP 和 TCP）和序号（TCP）的信息，这些我们将在第 14 和 15 章讨论 UDP 和 TCP 协议时看到。为了让源点能够通知这些协议（TCP 或 UDP）就需要这些信息。ICMP 生成一个差错分组，然后再被封装成 IP 数据报（见图 9.5）。

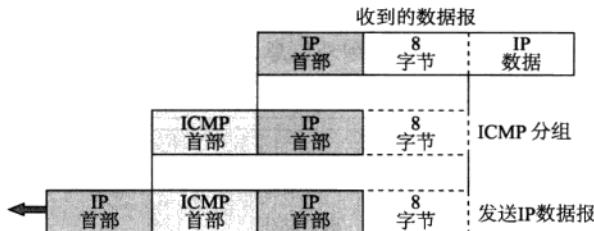


图 9.5 差错报文的数据字段的内容

终点不可达

当路由器无法为一个数据报找到路由，或者主机无法交付一个数据报时，该数据报被丢弃，然后由路由器或主机向发出这个数据报的源主机返回一个终点不可达报文（destination-unreachable message）。图 9.6 给出了终点不可达报文的格式。这种类型的代码字段指明了丢弃该数据报的原因：

类型: 3	代码: 0 ~ 15	检验和 未使用(全0)
收到的IP数据报的一部分, 包括IP首部 以及数据报数据的前8个字节		

图 9.6 终点不可达报文的格式

- **代码 0** 网络不可达。可能是硬件发生故障导致的。
- **代码 1** 主机不可达。这也可能是硬件发生故障导致的。
- **代码 2** 协议不可达。IP 数据报可以携带属于某个高层协议的数据，如 UDP、TCP 和 OSPF。例如，目的主机收到一个数据报必须交付给 TCP 协议，但这时 TCP 协议并未运行，那么就要发送一个代码 2 的报文。
- **代码 3** 端口不可达。数据报要交付的目标应用程序（进程）此时未运行。
- **代码 4** 需要进行分片，但这个数据报的 DF（不分片）字段已被设置。换言之，数据报的发送方已经指明这个数据报不能分片，但不进行分片就不可能为其进行路由选择。
- **代码 5** 源路由不能完成。换言之，在源路由选项中定义的一个或多个路由器无法通过。
- **代码 6** 目的网络未知。这与代码 0 不同。对于代码 0，路由器知道目的网络是存在的，只是当时无法到达。对于代码 6，路由器根本没有关于这个目的网络的信息。
- **代码 7** 目的主机未知。这与代码 1 不同。对于代码 1，路由器知道目的主机是存在的，只是当时无法到达。对于代码 7，路由器根本不知道目的主机的存在。
- **代码 8** 源主机被隔离了。
- **代码 9** 从管理上禁止与目的网络通信。
- **代码 10** 从管理上禁止与目的主机通信。
- **代码 11** 对指明的服务类型，网络不可达。这与代码 0 不同。如果源点请求的是一

个可用的服务类型，那么路由器就可以为数据报选择路由。

- **代码 12** 对指明的服务类型，主机不可达。这与代码 1 不同。如果源点请求的是一个可用的服务类型，那么路由器就可以为数据报选择路由。
- **代码 13** 主机不可达，因为管理员已经在这个主机上放置了过滤器。
- **代码 14** 主机不可达，因为主机违反了优先级策略。这种报文由路由器发出，指出所请求的优先级在该目的主机是不允许的。
- **代码 15** 主机不可达，因为它的优先级被截止。当网络操作员为网络的操作设定了最小的优先级限制，但发送的数据报的优先级低于此限制时，就会产生这个报文。

请注意，终点不可达报文可以由路由器或目的主机产生。代码 2 或 3 的报文只能由目的主机产生，其余的几个报文只能由路由器产生。

代码 2 或 3 的终点不可达报文只能由目的主机创建。其余的终点不可达报文只能由路由器创建。

还应注意，即使路由器没有发送终点不可达报文，也不一定表示数据报已经被交付了。例如，如果数据报通过以太网网络，那么路由器就无法知道这个数据报是否已交付给目的主机或下一个路由器，因为以太网不提供任何确认机制。

路由器无法检测出导致分组没有交付的所有问题。

源点抑制

IP 协议是无连接协议。在产生数据报的源主机和转发数据报的路由器以及处理数据报的目的主机之间没有任何通信联系。这种缺乏通信所引起的一个问题是缺乏流量控制和拥塞控制。

在 IP 协议中没有流量控制或拥塞控制机制。

ICMP 的源点抑制报文 (source-quench message) 就是为了给 IP 协议增加某种程度的流量控制和拥塞控制而设计的。当路由器或主机因拥塞而丢弃数据报时，它就向该数据报的发送方发送一个源点抑制报文。这个报文有两个作用。第一，它通知源点，数据报已被丢弃。第二，它警告源点，在路径中的某处出现了拥塞，因而源点必须放慢（抑制）发送过程。源点抑制的格式如图 9.7 所示。

类型: 4	代码: 0	检验和
未使用 (全0)		
收到的IP数据报的一部分,包括IP首部 以及数据报数据的前8个字节		

图 9.7 源点抑制报文的格式

源点抑制报文通知源点，由于路由器或目的主机的拥塞，数据报已被丢弃。源点必须放慢数据报的发送，直到拥塞得到缓解为止。

有几点值得做更多的解释。首先，遭遇拥塞的路由器或目的主机必须为每一个丢弃的数据报向源主机发送一个源点抑制报文。第二，没有一种机制可以告诉源点，拥塞已得到缓解，可以按照原来的速率发送数据报了。源点只是不断地降低发送速率，直到不再收到

更多的源点抑制报文为止。第三，在一对一的通信或多对一的通信中都有可能产生拥塞。在一一对 一的通信中，一台高速主机生成数据报的速度可能比路由器或目的主机处理数据报的速率要快得多。在这种情况下，源点抑制报文就很有用。这些报文告诉源点要放慢速率。在多对一的通信中，多个源点产生的数据报都必须由一个路由器或目的主机来处理。在这种情况下，每一个源点以不同的速率发送数据报，有的发送速率较低，而另一些发送速率则较高。在这种情况下，源点抑制报文就不一定有用。因为路由器或目的主机并不知道哪一个源点应该对拥塞负责。它可能丢弃了来自低速率的源点的数据报，而没有丢弃真正产生拥塞的源点所发来的数据报。

对每一个因拥塞而被丢弃的数据报，都应当发送一个源点抑制报文。

超时

超时报文 (time-exceeded message) 在以下两种情况下产生：

- 首先，如我们在第 6 章中所看到的，路由器使用路由表找出必须接收这个分组的下一跳（下一个路由器）。如果在一个或多个路由表中出现了错误，那么一个分组就很可能进入回环或循环状态，无休止地从一个路由器到下一个路由器，或无休止地通过一系列的路由器。正如在第 7 章中所见到的，每个数据报都有一个称为生存时间的字段来控制这种情况。当数据报通过路由器时，这个字段的值就减 1。当路由器发现这个字段的值在减 1 之后变为 0，就丢弃这个数据报。但是，在丢弃这样的数据报时，路由器需要向源点发送一个超时报文。

一旦路由器将数据报的生存时间字段值递减后变成了零，就丢弃这个数据报，并向源点发送超时报文。

- 第二，当组成一个报文的所有分片未能在某一时限内全部到达目的主机，也要产生超时报文。当第一个分片到达时，目的主机就启动一个计时器。如果计时器到时，却没有收齐所有的分片，目的主机就将已收到的分片全部丢弃，并向源点发送一个超时报文。

当最后的终点在规定的时间内没有收到所有的分片时，它就丢弃已收到的分片，并向源点发送超时报文。

图 9.8 给出了超时报文的格式。当数据报的生存时间字段值为零而被路由器丢弃时，就使用代码 0。因为在规定的时限内一个数据报的某些分片未能到达而导致已到达的分片被丢弃时，就使用代码 1。

类型: 11	代码: 0或1	检验和
		未使用(全0)
收到的IP数据报的一部分，包括IP首部 以及数据报数据的前8个字节		

图 9.8 超时报文的格式

在超时报文中，代码 0 仅供路由器使用，说明数据报的生存时间字段值为零。代码 1 仅供目的主机使用，说明不是所有的分片都按时到达了。

参数问题

当数据报经过因特网传输时，其首部中出现的任何二义性都可能会产生严重的问题。如果路由器或目的主机发现了这种二义性，或者在数据报的某个字段中缺少某个值，就会丢弃这个数据报，并向源点返回一个参数问题报文。

路由器或目的主机都可以产生参数问题报文。

图 9.9 给出了参数问题报文（parameter-problem message）的格式。此时的代码字段指明了丢弃数据报的原因：

- **代码 0** 首部的某个字段中有差错或二义性。在这种情况下，指针字段值指向有问题的字节。例如，若这个值为零，那么第一个字节是无效字段。
- **代码 1** 表示缺少所需的选项部分。在这种情况下不使用指针。

类型: 12	代码: 0或1	检验和
指针	未使用 (全0)	
收到的IP数据报的一部分，包括IP首部 以及数据报数据的前8个字节		

图 9.9 参数问题报文的格式

改变路由

当路由器要将分组转发到另一个网络时，它必须知道下一个适当的路由器的 IP 地址。如果发送方是一台主机，情况也是这样的，因此路由器和主机都必须有路由表，以便找出路由器或下一个路由器的地址。正如我们将在第 11 章所看到的，路由器要参与路由选择的更新过程，并且这种更新是经常进行的。路由选择是动态的。

但是为了提高效率，主机不参与路由选择更新过程，因为在因特网上的主机数量比路由器要多出很多。动态地更新主机的路由表会产生无法容纳的通信量。主机通常使用静态路由选择。刚开始时，主机的路由表中表项数目有限。它通常只知道默认路由器这一个路由器的 IP 地址。因此，主机可能会把发往另一个网络的数据报发送给一个错误的路由器。在这种情况下，收到这个数据报的路由器会把它转发给正确的路由器。但是，为了更新主机中的路由表，路由器还要向主机发送一个改变路由报文。改变路由的概念如图 9.10 所示。主机 A 打算向主机 B 发送一个数据报。路由器 R2 显然是最有效的路由选择，但是主机 A 没有选择路由器 R2。数据报被发送到路由器 R1 而不是 R2。R1 在查找路由表后发现分组应当给 R2。它把分组发送到 R2，同时向主机 A 发送一个改变路由报文。现在主机 A 的路由表就可以更新了。

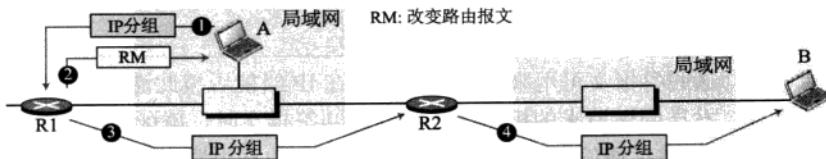


图 9.10 改变路由的概念

主机在刚开始工作时只有一张很小的路由表，这个路由表逐渐增大和更新。完成这项任务的工具之一就是改变路由报文。

图 9.11 给出了改变路由报文 (redirection message) 的格式。请注意：合适的目标 IP 地址在第二行中给出。

类型:5	代码: 0~3	检验和
目标路由器的IP地址		
收到的IP数据报的一部分，包括IP首部 以及数据报数据的前8个字节		

图 9.11 改变路由报文的格式

虽然改变路由报文被认为是一种差错报告报文，但它与其他差错报文不同。在这种情况下路由器不会丢弃数据报，而是将数据报发送给合适的路由器。改变路由报文的代码字段缩小了改变路由的范围：

- 代码 0** 对特定网络路由的改变。
- 代码 1** 对特定主机路由的改变。
- 代码 2** 基于指定服务类型的对特定网络路由的改变。
- 代码 3** 基于指定服务类型的对特定主机路由的改变。

改变路由报文是由路由器向同一个本地网络上的主机发送的。

9.2.3 查询

除差错报告外，ICMP 还能对某些网络问题进行诊断。这是通过查询报文来完成的。为了这个目的，人们已设计了一组五对不同的报文，但其中有三对如今已过时，我们将在本节稍后再讨论。目前只有两对还在使用：**回送请求与回答和时间戳请求与回答**。在使用这种类型的 ICMP 报文时，先由一个结点先发送一个报文，再由目的结点以指明的格式进行回答。

回送请求与回答

回送请求 (echo-request) 与回送回答 (echo-reply) 报文是为了诊断而设计的。网络管理员和用户可以使用这对报文来发现网络的问题。回送请求和回送回答组合起来确定了两个系统（主机或路由器）之间能否彼此通信。

一个主机或路由器可以向另一个主机或路由器发送回送请求报文。收到回送请求报文的主机或路由器产生回送回答报文，并将其返回给源发送方。

回送请求报文可以由主机或路由器发送。收到回送请求报文的主机或路由器发送回送回答报文。

回送请求和回送回答报文可用来确定两台机器在 IP 级能否彼此通信。因为 ICMP 报文被封装成数据报，发送回送请求的机器在收到回送回答报文时，就证明了发送方和接收方之间能够用 IP 数据报进行通信。此外，这还证明了中间的路由器也能够接收、处理和转发数据报。

回送请求和回送回答报文可被网络管理员用来检查 IP 协议的工作情况。

回送请求和回送回答报文还可被主机用来检查另一个主机是否可达。在用户级，调用

分组因特网搜寻器（ping）命令可以做到这点。今天，大多数的系统都提供 ping 命令的某个版本，它可以产生一连串（而不是仅仅一个）回送请求或回送回答报文，以提供统计信息。在本章结尾处我们将看到这个程序是如何使用的。

用回送请求和回送回答报文可测试某个主机的可达性。通常是调用 ping 命令来实现的。

回送请求和回送回答一起可用来验证某个结点是否工作正常。可以向被测试的结点发送回送请求报文，并且在可选的数据字段中包含了一个报文，这个报文必须由回答的结点在回送回答报文中一模一样地重复一遍。图 9.12 给出了回送请求和回送回答报文的格式。标识符和序号字段在协议中没有正式定义，可以由发送方任意使用。标识符通常与发起请求的进程的 ID 是一致的。

8: 回送请求	类型: 8 或 0	代码: 0	检验和
0: 回送回答	标识符	序号	
可选数据 由请求报文发送，被回答报文重复			

图 9.12 回送请求和回送回答报文

时间戳请求和回答

两个机器（主机或路由器）可以使用时间戳请求（timestamp-request）和时间戳回答（timestamp-reply）报文来确定 IP 数据报在这两个机器之间来回所需的往返时间。它也可用于同步两个机器的时钟。这两个报文的格式如图 9.13 所示。

13: 请求	类型: 13 或 14	代码: 0	检验和
14: 回答	标识符	序号	
原始时间戳			
接收时间戳			
发送时间戳			

图 9.13 时间戳请求和时间戳回答报文的格式

其中的三个时间戳字段都是 32 位长。每个字段可保存一个数，代表以通用时间（以前称为格林威治标准时间）的午夜起测量出的时间，以毫秒为单位。请注意：32 位可表示从 0~4294967295 的数，但在这种情况下，一个时间戳不会超过 $86400000 (= 24 \times 60 \times 60 \times 1000)$ 。

源点生成时间戳请求报文。源点在报文离开前要把它的时钟所显示的通用时间填入原始时间戳字段中，其他两个时间戳字段则都填入零。

终点生成时间戳回答报文。终点把请求报文中的原始时间戳值复制到回答报文的相同字段中。然后，在接收时间戳字段中填入收到这个请求时其时钟所显示的通用时间。最后，终点在回答报文离开前在发送时间戳字段中填入其时钟所显示的通用时间。

时间戳请求和时间戳回答报文可用来计算数据报从源点到终点所需的单向时间以及再返回到源点所需的往返时间。所用的公式是：

$$\text{发送时间} = \text{接收时间戳} - \text{原始时间戳}$$

$$\text{接收时间} = \text{分组返回的时间} - \text{发送时间戳}$$

$$\text{往返时间} = \text{发送时间} + \text{接收时间}$$

请注意，只有当源点和终点机器的时钟是同步的，发送时间和接收时间的计算才是准确的。但是，即使这两个时钟没有同步，往返时间的计算还是准确的，因为每一个时钟在往返时间的计算中出现了两次，它们在同步上的差别被抵消了。

时间戳请求和时间戳回答报文可用来计算源点和终点机器之间的往返时间，哪怕它们的时钟没有同步。

例如，给出以下的信息：

原始时间戳值：46

接收时间戳值：59

发送时间戳值：60

返回时间：67

我们可计算出往返时间是 20 ms：

$$\text{发送时间} = 59 - 46 = 13 \text{ ms}$$

$$\text{接收时间} = 67 - 60 = 7 \text{ ms}$$

$$\text{往返时间} = 13 + 7 = 20 \text{ ms}$$

若给出实际的单向时间，则时间戳请求和时间戳回答报文还可用来对两个机器的时钟进行同步，使用公式如下所示：

$$\text{时间差} = \text{接收时间戳} - (\text{原始时间戳字段} + \text{单向经历时间})$$

要得到单向经历时间，可以将往返经历的时间除以 2（如果我们确信发送时间和接收时间是一样的），或者也可以使用其他方法。例如，我们可得出在上面的例子中，两个机器时钟的差别是 3 毫秒，因为

$$\text{时间差} = 59 - (46 + 10) = 3 \text{ ms}$$

如果已知准确的单向经历时间，则时间戳请求和时间戳回答报文可用来同步两个机器的时钟。

过时的报文

有三对报文已被 IETF 宣布为过时失效：

1. 信息请求和回答报文目前已不再使用，因为它们的任务已交给第 8 章讨论的地址解析协议来完成。

2. 地址掩码请求和回答报文目前已不再使用，因为它们的任务已交给第 18 章讨论的动态主机配置协议（DHCP）来完成。

3. 路由器询问和通告报文目前已不再使用，因为它们的任务已交给第 18 章讨论的动态主机配置协议（DHCP）来完成。

9.2.4 检验和

在第 7 章我们学习了有关检验和的概念及思想。在 ICMP 中，检验和的计算覆盖了整个报文（首部和数据）。

检验和的计算

发送方按以下步骤使用反码算术运算：

1. 把检验和字段置为零。
2. 计算所有 16 位字（首部和数据）之和。
3. 把得到的和求反码，得到检验和。
4. 把检验和存储在检验和字段中。

检验和的检测

接收方按以下步骤使用反码算术运算：

1. 计算所有 16 位字（首部和数据）之和。
2. 把得到的和求反码。
3. 若步骤 2 得到的结果是 16 个 0，则接受这个报文，否则就拒绝这个报文。

例 9.1

图 9.14 给出了一个简单的回送请求报文（见图 9.12）的检验和计算的例子。我们随机地选择标识符为 1 和序号为 9。这个报文被划分成 16 位（2 字节）的字。把这些字相加，然后把得到的和求反码。现在发送方就可将此值放入检验和字段。

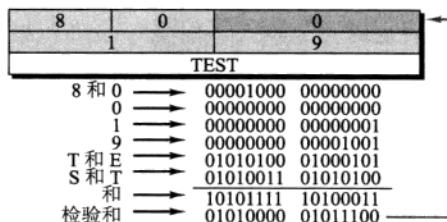


图 9.14 计算检验和的例子

9.3 排错工具

在因特网中用于排错的工具有多种。我们能够查出某个主机或路由器是否已加电并正在运行。我们也可以跟踪一个分组的路由。这里要介绍两种使用 ICMP 进行排错的工具：ping 和 traceroute。在以后的章节中讨论相应的协议时我们还要介绍更多的工具。

9.3.1 ping

我们可以使用程序 ping 来查出某个主机是否已加电并能够响应。我们在第 7 章中曾使用程序 ping 来模拟记录路由选项。现在我们要更详细地讨论 ping，研究它是怎样使用 ICMP 分组的。

源主机发送 ICMP 回送请求报文（类型：8，代码：0）。如果终点已加电，就会返回 ICMP 回送回答报文。程序 ping 在回送请求和回送回答报文中设置了标识符字段，并使序号从 0 开始，每发送一个新报文序号就递增 1。请注意，ping 能够计算往返时间。它在报文的数据部分插入了发送时间。当分组到达时，它就用到达时间减去出发时间得出往返时间（round-trip time, RTT）。

例 9.2

我们使用程序 ping 测试服务器 fhda.edu。结果如下：

```
$ ping fhda.edu
PING fhda.edu (153.18.8.1) 56(84) bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 0 ttl=62 time=1.91 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 1 ttl=62 time=2.04 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 2 ttl=62 time=1.90 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 3 ttl=62 time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 4 ttl=62 time=1.93 ms

64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 5 ttl=62 time=2.00 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 6 ttl=62 time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 7 ttl=62 time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 8 ttl=62 time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 9 ttl=62 time=1.89 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq = 10 ttl=62 time=1.98 ms

--- fhda.edu ping statistics ---
11 packets transmitted. 11 received, 0% packet loss, time 10103ms
rtt min/avg/max = 1.899/1.955/2.041 ms
```

程序 ping 发送了序号从 0 开始的报文。对每一次探测，给出一个往返时间 RTT。封装了 ICMP 报文的 IP 数据报的 TTL（生存时间）字段值被设置为 62，这表示分组不能传送超过 62 跳。在开始时，ping 定义了数据的字节数为 56，字节总数是 84。显然，如果我们在 56 的上面加上 8 字节的 ICMP 首部和 20 字节的 IP 首部，结果就是 84。但是应当注意，每一个探测的 ping 包定义的字节数是 64，这是 ICMP 分组的字节总数 (56 + 8)。如果我们不使用中断键（例如，ctrl + c）停止它，程序 ping 会不断地发送报文。当它被中断后，就打印出探测的统计值。这个统计值告诉我们发送的分组数、接收到的分组数、总时间、RTT 的最小值、最大值和平均值。

例 9.3

第二个例子是，我们希望知道，邮件服务器 adelphia.net 是否加电并正在运行。结果如下：

```
$ ping mail.adelphia.net
PING mail.adelphia.net (68.168.78.100) 56(84) bytes of data.
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 0 ttl=48 time=85.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 1 ttl=48 time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 2 ttl=48 time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 3 ttl=48 time=84.3 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 4 ttl=48 time=84.5 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 5 ttl=48 time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 6 ttl=48 time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 7 ttl=48 time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 8 ttl=48 time=84.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 9 ttl=48 time=84.2 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 10 ttl=48 time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 11 ttl=48 time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq = 12 ttl=48 time=84.5 ms

--- mail.adelphia.net ping statistics ---
```

```
14 packets transmitted, 13 received, 7% packet loss, time 13129ms
rtt min/avg/max = 84.207/84.694/85.469 ms
```

请注意，在这种情况下，我们共发送了 14 个分组，但只返回了 13 个。我们可能在最后一个分组（序号为 13）返回之前就中断了这个程序。

9.3.2 traceroute

在 UNIX 中的程序 traceroute 或在 Windows 中的程序 tracert 可以用来跟踪一个分组从源点到终点的路径。让我们用图 9.15 来说明程序 traceroute 的思想。

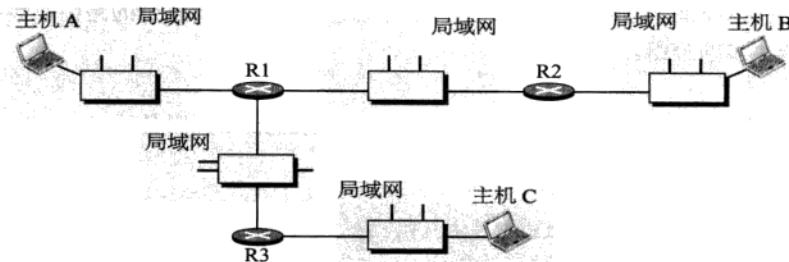


图 9.15 程序 traceroute 的工作过程

我们已经在前面的章节中看过用程序 traceroute 模拟一个 IP 数据报的不严格源路由和严格源路由选项。在本章中，我们把这个程序和 ICMP 分组一起使用。这个程序很巧妙地使用了两个 ICMP 报文—超时报文和终点不可达报文来找出一个分组的路由。这是一个应用级的程序，它使用 UDP 服务（见第 14 章）。

给定了图 9.15 所示的拓扑，我们就知道从主机 A 到主机 B 的分组要经过路由器 R1 和 R2。但是，在大多数情况下，我们并不知道这个拓扑。在主机 A 到主机 B 之间可能会有很多的路由器。程序 traceroute 使用 ICMP 报文和 IP 分组中的 TTL（生存时间）字段就能找出这个路由。

1. 程序 traceroute 使用以下步骤找出路由器 R1 的地址和主机 A 到路由器 R1 之间的往返时间。程序 traceroute 重复步骤 a 到 c 三次，以便得到更好的往返时间平均值。第一个往返时间可能会比第二或第三个大得多，因为用 ARP 找出路由器 R1 的物理地址需要花费时间。对于第二和第三个往返时间，ARP 可使用高速缓存中的地址。

a. 主机 A 上的程序 traceroute 使用 UDP 向终点 B 发送一个分组。这个报文被封装成 IP 分组，其 TTL 值为 1。这个程序记下发送该分组的时间。

b. 路由器 R1 收到这个分组，把 TTL 值减到 0。路由器 R1 就丢弃这个分组（因为 TTL 为 0）。同时路由器 R1 要发送一个超时 ICMP 报文（类型：11，代码：0），表示因 TTL 值为 0 而丢弃该分组。

c. 主机 A 的程序 traceroute 收到这个 ICMP 报文，利用封装 ICMP 的 IP 分组的源地址找出路由器 R1 的地址。这个程序还记下该分组的到达时间。步骤 a 的时间与这个时间之差就是往返时间。

2. 程序 traceroute 重复前面的步骤找出路由器 R2 的地址，以及主机 A 和路由器 R2

之间的往返时间。不过，在这一步 TTL 值设置为 2。因此路由器 R1 会转发这个报文，而路由器 R2 会丢弃它，并发送超时 ICMP 报文。

3. 程序 traceroute 重复前面的步骤找出主机 B 的地址，以及主机 A 和主机 B 之间的往返时间。当主机 B 收到这个分组时，就把 TTL 值减 1，但并不丢弃这个报文，因为现在报文已经到达了它最后的终点。那么 ICMP 报文怎样发回给主机 A 呢？程序 traceroute 使用另一种策略。UDP 的目的端口被设置为 UDP 协议不支持的一个端口。当主机 B 收到这个分组时，它找不到可以接受交付的应用程序。于是它就丢弃这个分组，并向主机 A 发送一个 ICMP 终点不可达报文（类型：3，代码：3）。请注意，这种情况在路由器 R1 和 R2 并没有发生，因为路由器不检查 UDP 首部。程序 traceroute 记录了到达的 IP 数据报的目的地址，并记下了往返时间。收到代码为 3 的终点不可达报文就表示整个的路由已经找出，没有必要再发送更多的分组。

例 9.4

我们利用程序 traceroute 找出从计算机 voyager.deanza.edu 到服务器 fhda.edu 的路由。结果如下：

```
$ traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 38 byte packets.
1. Dcore.fhda.edu (153.18.31.254) 0.995 ms 0.889 ms 0.878 ms
2. Dbackup.fhda.edu (153.18.251.4) 1.039 ms 1.064 ms 1.083 ms
3. tiptoe.fhda.edu (153.18.8.1) 1.797 ms 1.642 ms 1.757 ms
```

紧跟在命令行后面的无编号行指出终点是 153.18.8.1，TTL 值是 30 跳，分组有 38 字节（20 字节的 IP 首部，8 字节的 UDP 首部，10 字节的应用数据）。程序 traceroute 利用这些应用数据跟踪分组。第一行给出了访问的第一个路由器。这个路由器的名字是 Dcore.fhda.edu，其 IP 地址是 153.18.31.254。第一个往返时间是 0.995 ms，第二个是 0.889 ms，第三个是 0.878 ms。第二行给出了访问的第二个路由器。这个路由器的名字是 Dbackup.fhda.edu，其 IP 地址是 153.18.251.4。同样也给出了三个往返时间。第三行给出目的主机。我们知道这就是目的主机，因为没有更多的行了。目的主机就是服务器 fhda.edu，但它的名字是 tiptoe.fhda.edu，IP 地址是 153.18.8.1。同样也给出了三个往返时间。

例 9.5

在这个例子中，我们跟踪了一个更长的路由，到 xerox.com 的路由。

```
$ traceroute xerox.com
traceroute to xerox.com (13.1.64.93), 30 hops max, 38 byte packets.
1. Dcore.fhda.edu (153.18.31.254) 0.622 ms 0.891 ms 0.875 ms
2. Ddmz.fhda.edu (153.18.251.40) 2.132 ms 2.266 ms 2.094 ms
3. Cinic.fhda.edu (153.18.253.126) 2.110 ms 2.145 ms 1.763 ms
4. cenic.net (137.164.32.140) 3.069 ms 2.875 ms 2.930 ms
5. cenic.net (137.164.22.31) 4.205 ms 4.870 ms 4.197 ms
6. cenic.net (137.164.22.167) 4.250 ms 4.159 ms 4.078 ms
7. cogentco.com (38.112.6.225) 5.062 ms 4.825 ms 5.020 ms
8. cogentco.com (66.28.4.69) 6.070 ms 6.207 ms 5.653 ms
9. cogentco.com (66.28.4.94) 6.070 ms 5.928 ms 5.499 ms
10. cogentco.com (154.54.2.226) 6.545 ms 6.399 ms 6.535 ms
11. sbcglobal.net (151.164.89.241) 6.379 ms 6.370 ms 6.210 ms
12. sbcglobal.net (64.161.1.45) 6.908 ms 6.748 ms 7.359 ms
```

13. sbcglobal.net	(64.161.1.29)	7.023 ms	7.040 ms	6.734 ms
14. snfc21.pbi.net	(151.164.191.49)	7.656 ms	7.129 ms	6.866 ms
15. sbcglobal.net	(151.164.243.58)	7.844 ms	7.545 ms	7.353 ms
16. pacbell.net	(209.232.138.114)	9.857 ms	9.535 ms	9.603 ms
17. 209.233.48.223	(209.233.48.223)	10.634 ms	10.771 ms	10.592ms
18. alpha.Xerox.COM	(13.1.64.93)	10.922ms	11.048 ms	10.922ms

这一次从源点到终点共有 17 跳。请注意，某些往返时间看起来不大正常。可能是有个路由器太忙，不能及时地处理分组。

例 9.6

一件很有趣的事就是一个主机可以给它自己发送 traceroute 分组。只要指明这个主机为终点即可。正如我们所期待的，这个分组应当传送给环回地址。

```
$ traceroute voyager.deanza.edu
traceroute to voyager.deanza.edu (127.0.0.1), 30 hops max, 38 byte packets.
1. voyager      (127.0.0.1)   0.178 ms   0.086 ms   0.055 ms
```

例 9.7

最后，我们使用程序 traceroute 找出从 fhda.edu 到 mhhe.com (McGraw-Hill 的服务器) 之间的路由。我们注意到，我们不能找出完整的路由。当 traceroute 在 5 秒之内收不到响应时，它就打印出一个星号表示有问题（在本例中不是这样的），然后就试下一跳。

```
$ traceroute mhhe.com
traceroute to mhhe.com (198.45.24.104), 30 hops max, 38 byte packets.
1. Dcore.fhda.edu    (153.18.31.254)    1.025 ms    0.892 ms    0.880 ms
2. Ddmz.fhda.edu    (153.18.251.40)    2.141 ms    2.159 ms    2.103 ms
3. Cinic.fhda.edu    (153.18.253.126)    2.159 ms    2.050 ms    1.992 ms
4. cenic.net        (137.164.32.140)    3.220 ms    2.929 ms    2.943 ms
5. cenic.net        (137.164.22.59)     3.217 ms    2.998 ms    2.755 ms
6. SanJose1.net     (209.247.159.109)    10.653 ms   10.639 ms   10.618ms
7. SanJose2.net     (64.159.2.1)       10.804 ms   10.798 ms   10.634ms
8. Denver1.Level3.net (64.159.1.114)    43.404 ms   43.367 ms   43.414ms
9. Denver2.Level3.net (4.68.112.162)    43.533 ms   43.290 ms   43.347ms
10. unknown          (64.156.40.134)    55.509 ms   55.462 ms   55.647ms
11. mcleodusa1.net   (64.198.100.2)    60.961 ms   55.681 ms   55.461ms
12. mcleodusa2.net   (64.198.101.202)    55.692 ms   55.617 ms   55.505ms
13. mcleodusa3.net   (64.198.101.142)    56.059 ms   55.623 ms   56.333ms
14. mcleodusa4.net   (209.253.101.178)    297.199 ms  192.790 ms  250.594ms
15. cppg.com         (198.45.24.246)    71.213 ms   70.536 ms   70.663ms
16. ...              ...           ...           ...           ...
```

9.4 ICMP 软件包

为了让大家了解 ICMP 是如何处理发送和接收 ICMP 报文的，这里展示了一种我们自己的 ICMP 软件包版本，它由两个模块组成：输入模块和输出模块。图 9.16 给出了这两个模块。

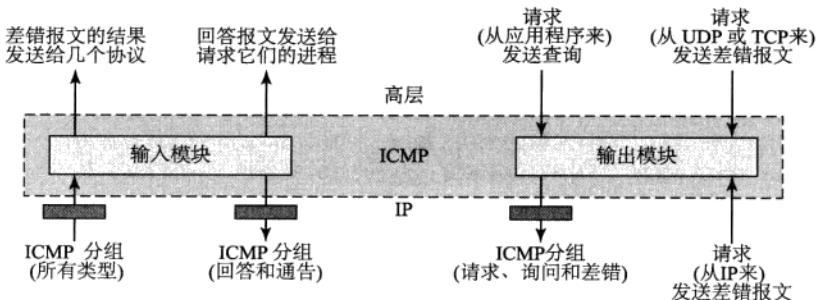


图 9.16 ICMP 软件包

9.4.1 输入模块

输入模块处理所有收到的 ICMP 报文。当一个来自 IP 层来的 ICMP 分组要交付给它时，它就被激活了。如果收到的分组是请求报文，输入模块就产生一个回答报文并把它发送出去。如果收到的分组是改变路由报文，输入模块就使用其中的信息更新路由表。如果收到的分组是差错报文，输入模块就通知相关协议有关差错的情况。表 9.2 所示的是伪码。

表 9.2 输入模块

```

1  ICMP_Input_Module(ICMP_Packet )
2  {
3      If(类型为请求)
4      {
5          产生回答报文
6          发送这个回答报文
7      }
8      If(类型指定的是一个改变路由报文)
9      {
10         修改路由表
11     }
12     If(类型指定的是其他差错报文)
13     {
14         把情况通知适当的源协议
15     }
16     返回
17 }
```

9.4.2 输出模块

输出模块负责根据高层协议或 IP 协议的要求创建请求报文、询问报文或差错报文。输

出模块接收来自 IP、UDP 或 TCP 的请求发送一个 ICMP 差错报文。若是来自 IP，则输出模块必须首先检查这个请求是否允许。应当记住，有四种情况是不允许创建 ICMP 报文的：携带 ICMP 差错报文的 IP 分组、被分片的 IP 分组、多播 IP 分组，以及 IP 地址为 0.0.0.0 或 127.X.Y.Z 的 IP 分组。输出模块还可从应用程序处收到发送 ICMP 某个请求报文的要求。表 9.3 所示的是伪码：

表 9.3 输出模块

```

1  ICMP_Output_Module(要求)
2  {
3      If(要求定义的是一个差错报文)
4      {
5          If(要求是从 IP 来的并且是禁止的)
6          {
7              返回
8          }
9          If(要求是一个有效的改变路由报文)
10         {
11             返回
12         }
13         产生一个差错报文
14     }
15     If(要求定义的是一个请求)
16     {
17         产生一个请求报文
18     }
19     发送这个报文
20     返回
21 }
```

9.5 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

9.5.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]和[Ste 94]。

9.5.2 RFC

几个讨论了 ICMP 的 RFC 包括：RFC 792、RFC 950、RFC 956、RFC 957、RFC 1016、RFC 1122、RFC 1256、RFC 1305 和 RFC1987。

9.6 重要术语

终点不可达报文	改变路由报文
回送回答报文	往返时间（RTT）
差错报告报文	源点抑制报文
回送请求报文	超时报文
网际控制报文协议（ICMP）	时间戳回答报文
参数问题报文	时间戳请求报文
ping	traceroute
查询报文	

9.7 本章小结

- 网际控制报文协议（ICMP）支持不可靠的和无连接的网际协议（IP）。
- ICMP 被封装在 IP 数据报中。ICMP 报文有两种类型：差错报告报文和查询报文。
差错报告报文报告了路由器或主机（终点）在处理 IP 数据报时可能遇到的问题。
查询报文总是成对出现的，它帮助主机或网络管理员从某个路由器或对方主机那里获取特定的信息。
- 计算 ICMP 的检验和要用到 ICMP 报文的首部及其数据字段。
- 在因特网中用于排错的工具有多种。我们能够查出某个主机或路由器是否已加电并正在运行。其中的两个工具是 ping 和 traceroute。
- 一个简单的 ICMP 设计由输入模块和输出模块组成，输入模块处理到来的 ICMP 分组，而输出模块处理对 ICMP 服务的要求。

9.8 实践安排

9.8.1 习题

1. 主机 A 向主机 B 发送了一个时间戳请求报文，但却没有收到任何回答。试讨论三种可能的原因和相应的对策。

2. 为什么要限制对失败的 ICMP 差错报文再产生一个 ICMP 报文？
3. 主机 A 向主机 B 发送了一个数据报。主机 B 未能收到该数据报，而主机 A 也没有收到任何失败的通知。试给出两种可能发生的情况。
4. 在 ICMP 差错报告报文其中包括 IP 首部和数据报数据前 8 个字节的目的是什么？
5. 在参数问题报文中，指针字段的最大值是多少？
6. 试举例说明主机可能永远收不到改变路由报文的情况。
7. 试用表格说明哪些 ICMP 报文是由路由器发送出的，哪些是由非目的主机发送出的，哪些又是由目的主机发送出的？
8. 计算出的发送时间、接收时间或往返时间是否会出现负值？为什么？请举例说明。
9. 为什么分组的单向时间不是简单地用往返时间除以 2？
10. ICMP 分组的最小长度是多少？最大长度又是多少？
11. 携带 ICMP 分组的 IP 分组的最小长度是多少？最大长度又是多少？
12. 如果 IP 分组携带的是 ICMP 分组，那么携带这个 IP 分组的以太网帧的最小长度是多少？最大长度又是多少？
13. 我们如何能够确定 IP 分组携带的是不是 ICMP 分组？
14. 试计算下述 ICMP 分组的检验和：
类型：回送请求 标识符：123 序号：25 报文：Hello
15. 某路由器收到一 IP 分组，其源 IP 地址为 130.45.3.3 且目的 IP 地址为 201.23.4.6。这个路由器无法在其路由表中找到该目的 IP 地址。试填写发出的 ICMP 报文的各个字段（你能写出多少就写多少）。
16. TCP 接收到目的端口为 234 的一个报文段。TCP 检查后发现无法打开这个目的端口。试填写发出的 ICMP 报文的各个字段。
17. 一个到达的 ICMP 报文的首部如下（十六进制表示）：
03 03 10 20 00 00 00 00
这个报文的类型是什么？代码是什么？这个报文的目的又是什么？
18. 一个到达的 ICMP 报文的首部如下（十六进制表示）：
05 00 11 12 11 0B 03 02
这个报文的类型是什么？代码是什么？这个报文的目的是什么？最后 4 字节的值是什么？最后这几个字节是什么意思？
19. 某计算机发送了时间戳请求。若它的时钟显示的是 5:20:30_{A.M.}（通用时间），试给出这个报文中的各个表项。
20. 若时间为 3:40:30_{P.M.}（通用时间），重复习题 19。
21. 某计算机在 2:34:20_{P.M.}收到从另一计算机发来的时间戳请求。原始时间戳值为 52453000。若发送方的时钟慢了 5 毫秒，试问单向时延是多少？
22. 某计算机向另一台计算机发送了一个时间戳请求。它在 3:46:07_{A.M.}收到相应的时
间戳回答。原始时间戳、接收时间戳以及发送时间戳的值分别为 13560000、13562000 和 13564300。试问发送路途用了多少时间？返回路途用了多少时间？往返时间是多少？发送方与接收方的时钟之差是多少？
23. 若两个计算机相距 5000 英里，报文从一个计算机到另一个计算机所用的最小时

间是多少？

9.8.2 研究活动

24. 试使用程序 ping 测试你自己的计算机（环回）。
25. 试使用程序 ping 测试一台国内的计算机。
26. 试使用程序 ping 测试一台国外的计算机。
27. 试使用程序 traceroute（或 tracert）找出从你的计算机到一个学院或大学的计算机的路由。
28. 你怎样能够找出在习题 27 的两个路由器之间的 RTT？

第 10 章 移 动 IP

在过去的十几年中，移动通信受到了人们极大的关注。人们对因特网上的移动通信感兴趣就意味着原来为固定设备的通信而设计的 IP 协议必须要增强，使之能够适用于从一个网络移动到另一个网络的移动计算机。

目标

本章有以下几个目标：

- 讨论与移动主机相关的编址问题以及转交地址的必要性。
- 讨论在移动 IP 通信中出现的两个代理：归属代理和外地代理，以及它们相互之间的通信问题。
- 解释在移动主机和远程主机之间通信的三个阶段：代理发现、登记和数据传送。
- 说明在两次穿越和三角路由选择这两种情况下移动 IP 的低效率问题以及可能的解决方案。

10.1 编址

在使用 IP 协议提供移动通信时必须解决的主要问题就是编址。

10.1.1 固定主机

最初 IP 编址基于这样的假定：主机是固定的，并且连接到某个特定的网络。路由器利用 IP 地址为一个 IP 数据报选择路由。正如我们在第 5 章中看到的，IP 地址由两部分组成：前缀和后缀。前缀使得主机与某个网络相关联。例如，IP 地址 10.3.4.24/8 定义了连接在网络 10.0.0.0/8 上的一台主机。这也暗示了因特网上的主机没有一个能够被它携带着从一个地方走到另一个地方而固定不变的地址。只有当主机连接到特定的网络上时，这个地址才有效。如果网络改变了，这个地址就不再有效。路由器利用这种关联来为分组选择路由，也就是根据前缀把分组交付到这个主机所连接的网络。这种方案对于固定主机（stationary host）来说是很有效的。

IP 地址是为固定主机的操作而设计的，因为这个地址的一部分定义了主机所连接的网络。

10.1.2 移动主机

当主机从一个网络移动到另一个网络时，IP 编址结构就需要进行修改。为实现这一目标，已提出了几种解决方案。

改变地址

一种简单的解决方法就是让移动主机（mobile host）在移动到新的网络时改变它的地址。主机可以使用 DHCP（参见第 18 章）获得一个新的地址，把它和新的网络关联起来。但这种做法有几个缺点。首先，配置文件可能需要改变。其次，每当计算机从一个网络移动到另一个网络时，它就必须重新启动。第三，DNS 表（参见第 19 章）必须更新，以便因特网上的其他主机都能知道这种变化。第四，如果这个主机正在传输时从一个网络漫游到另一个网络，数据交换就会中断。这是因为在连接持续期间客户和服务器的端口和 IP 地址都必须保持不变。

两个地址

使用两个地址是一种更可行的解决方法。主机有它的原始地址称为归属地址（home address），还有一个临时地址称为转交地址（care-of address）。归属地址是永久的，它使得主机和它的归属网络（home network，即这个主机的永久归属）相关联。转交地址是临时的。当主机从一个网络移动到另一个网络时，转交地址就发生变化，使之与外地网络（foreign network，即这个主机移动到的网络）相关联。图 10.1 给出了这个概念。

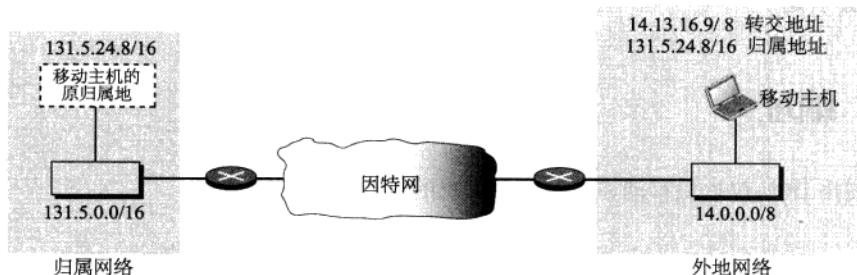


图 10.1 归属地址和转交地址

移动 IP 要给移动主机两个地址：一个归属地址，一个转交地址。归属地址是永久的，而当移动主机从一个网络移动到另一个网络时，转交地址就改变。

当移动主机接入一个外地网络时，它就会在代理发现阶段和登记阶段收到自己的转交地址，这两个阶段我们将在下面讨论。

10.2 代理

为了让地址的改变对因特网的其余部分是透明的，就需要一个归属代理（home agent）和一个外地代理（foreign agent）。图 10.2 给出了归属代理相对于归属网络的位置，以及外

地代理相对于外地网络的位置。



图 10.2 归属代理和外地代理

在图中我们将归属代理和外地代理表示为路由器，但是需要强调的是它们作为代理的特定功能是在应用层完成的。换言之，它们既是路由器，也是主机。

10.2.1 归属代理

归属代理通常是连在移动主机的归属网络上的路由器。当远程主机向移动主机发送分组时，归属代理就充当该移动主机。先由归属代理接收分组，然后再把分组发送给外地代理。

10.2.2 外地代理

外地代理通常是连接在外地网络上的路由器。外地代理接收归属代理发送过来的分组，并把这些分组交付给移动主机。

移动主机也可充当外地代理。换言之，移动主机和外地代理可以是同一个设备。但是，要这样做，移动主机必须能够收到自己的转交地址，这可以通过 DHCP 来实现。此外，移动主机还需要具有一些必要的软件，使之能够与归属代理通信，并且能够拥有两个地址：它的归属地址和它的转交地址。对应用程序来说，这两个地址必须是透明的。

当移动主机充当外地代理时，这个转交地址称为同址转交地址(colocated care-of address)。

当移动主机和外地代理是同一个设备时，它的转交地址称为同址转交地址。

使用同址转交地址的好处是移动主机可以移动到任何网络，而不必担心外地代理的可用性。但缺点是移动主机需要额外的软件使之能够充当自己的外地代理。

10.3 三个阶段

要与远程主机通信，移动主机需经过三个阶段：代理发现、登记和数据传送，如图 10.3 所示。

第一个阶段是代理发现，它涉及到移动主机、外地代理和归属代理。第二个阶段是登记，也涉及到移动主机和这两个代理。最后，第三个阶段另外还要涉及到远程主机。我们

将分别讨论每一个阶段。

10.3.1 代理发现

移动通信的第一个阶段是代理发现 (agent discovery)，它包括两个子阶段。移动主机在离开它的归属网络之前必须发现归属代理，也就是掌握归属代理的地址。移动主机还要在它移动到外地网络后发现外地代理。这里的“发现”是指掌握外地代理地址以及转交地址。这个发现过程涉及到两种类型的报文：通告和询问。

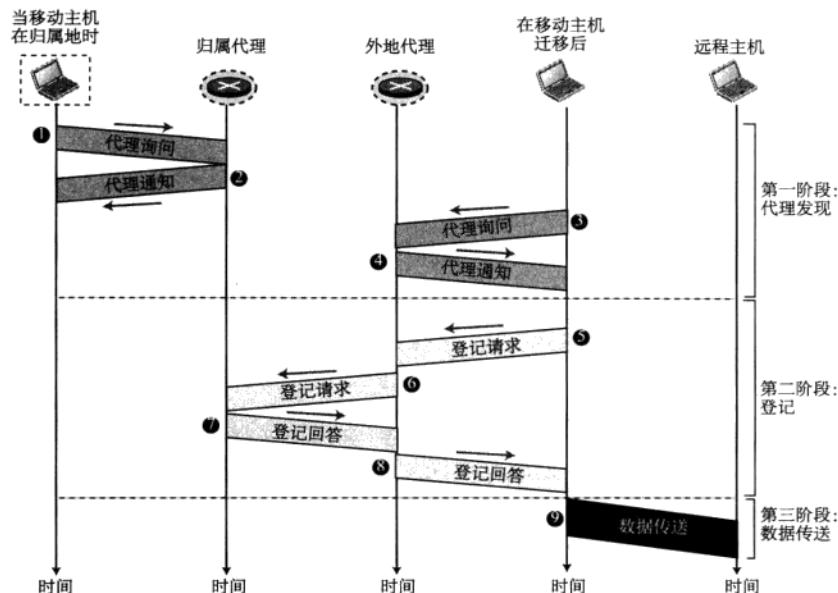


图 10.3 远程主机和移动主机通信

代理通告

如果一个路由器充当了代理的角色，那么在路由器使用 ICMP 路由器通告报文宣布自己连接到某个网络时，就可以在这个分组上再附带一个代理通告 (agent advertisement)。图 10.4 给出了路由器通告分组是怎样捎带代理通告的。



图 10.4 代理通告

移动 IP 没有使用新的分组类型来进行代理通告，它使用了 ICMP 的路由器通告分组，并在后面附带了代理通告报文。

这些字段的描述如下：

- 类型** 这个 8 位字段设置为 16。
- 长度** 这个 8 位字段定义了扩充报文的总长度（不是 ICMP 通告报文的长度）。
- 序号** 这个 16 位序号字段保存的是报文编号。接收者使用这个编号确定是否有报文丢失。
- 生存时间** 生存时间字段定义了代理接受请求的时间长度，以秒为单位。如果这个字段是一串 1，那么生存时间就是无穷大。
- 代码** 代码字段是一个 8 位的标志，它的每一位都可以置 1 或置 0。每一位的含义如表 10.1 所示。

表 10.1 代码位

位	意义
0	需要登记。没有同址转交地址
1	代理忙，现在不接受登记
2	代理充当归属代理
3	代理充当外地代理
4	代理使用最小的封装
5	代理使用通用路由选择封装 (GRE)
6	代理支持首部压缩
7	未使用 (0)

- 转交地址** 这个字段包含的是可用作转交地址的地址列表。移动主机可选择其中的一个地址。转交地址的选择在登记请求中宣布。应当注意，这个字段仅为外地代理使用。

代理询问

当移动主机已经移动到新的网络而没有收到代理通告时，它可以发起一个代理询问 (agent solicitation)。它可以使用 ICMP 询问报文通知代理，让代理知道它需要帮助。

移动 IP 没有为代理询问使用新的分组类型，它使用 ICMP 的路由器询问分组。

10.3.2 登记

移动通信的第二个阶段是登记 (registration)。在移动主机移动到外地网络并已经发现了外地代理后，它必须登记。登记的四个要素如下：

1. 移动主机必须向外地代理登记。
2. 移动主机必须向它的归属代理登记。这通常是由外地代理以移动主机的身份来完成的。
3. 如果截止期到了，那么移动主机必须更新登记。
4. 如果移动主机回到归属网络，它就必须取消登记。

请求和回答

为了向外地代理和归属代理登记，移动主机要使用登记请求（registration request）和登记回答（registration reply），如图 10.3 所示。

登记请求 移动主机把登记请求发送给外地代理，以便登记它的转交地址，同时也告知它的归属地址和归属代理地址。外地代理收到并登记这个请求之后，把该报文转发给归属代理。请注意，现在归属代理就知道了外地代理的地址，因为转发时使用的 IP 分组把外地代理的 IP 地址作为源地址。图 10.5 给出了登记请求的格式。

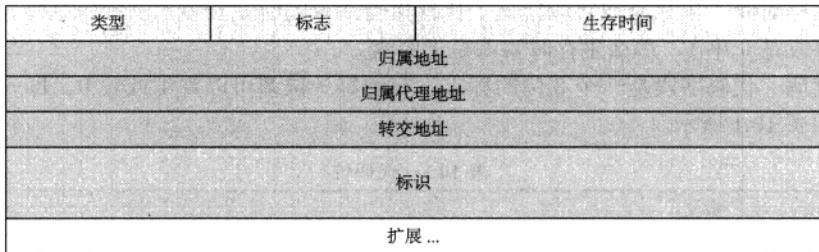


图 10.5 登记请求的格式

各字段的描述如下：

- **类型** 这个 8 位字段定义了报文的类型。对于请求报文，这个字段的值是 1。
- **标志** 这个 8 位字段定义了转发信息。每一位都可以置 1 或置 0。每一位的意义如表 10.2 所示。

表 10.2 登记请求标志字段的各位

位	意义
0	移动主机请求归属代理保留它以前的转交地址
1	移动主机请求归属代理把任何广播报文用隧道技术发送出去
2	移动主机使用同址转交地址
3	移动主机请求归属代理使用最小封装
4	移动主机请求使用通用路由选择封装（GRE）
5	移动主机请求首部压缩
6-7	保留位

- **生存时间** 这个 16 位字段定义了登记的合法时间长度，以秒为单位。如果这个字段是一串 0，就表示这个请求报文是要求取消登记。如果这个字段是一串 1，生存时间就是无穷大。
- **归属地址** 这个字段包含移动主机的永久（第一个）地址。
- **归属代理地址** 这个字段包含归属代理的地址。
- **转交地址** 这个字段是移动主机的临时（第二个）地址。
- **标识** 这个字段包含一个 64 位的数，它被移动主机插入到请求中，而在回答报文中要重复这个数。它使得请求和回答相互匹配。

□ 扩展 可变长度的扩展被用于鉴别。它们使归属代理能够鉴别移动主机。我们将在第 29 章讨论鉴别。

登记回答 登记回答由归属代理发送给外地代理，然后再被转发给移动主机。这个回答用于确认或拒绝登记请求。图 10.6 给出了登记回答的格式。

类型	代码	生存时间
	归属地址	
	归属代理地址	
	标识	
	扩展 ...	

图 10.6 登记回答的格式

其中的字段和登记请求相似，只有以下几个例外。类型字段的值是 3。代码字段代替了标志字段，并给出登记请求的结果（接受或拒绝）。这里不需要转交地址。

封装

登记报文被封装成 **UDP 用户数据报**。代理使用熟知端口 434，移动主机使用一个临时端口。

登记请求或回答通过 UDP 采用熟知端口 434 发送。

10.3.3 数据传送

在代理发现和登记后，移动主机就能够和远程主机进行通信。图 10.7 描绘了其中的思想。
从远程主机到归属代理

当远程主机要向移动主机发送分组时，它使用自己的地址作为源地址，而移动主机的归属地址作为目的地址。换言之，远程主机发送分组时还是认为移动主机连接在它的归属网络上。但是，这个分组被归属代理截获了。归属代理假装自己就是这个移动主机。这里使用了第 8 章中讨论的代理 ARP 技术。图 10.7 中的路径 1 表示这个步骤。

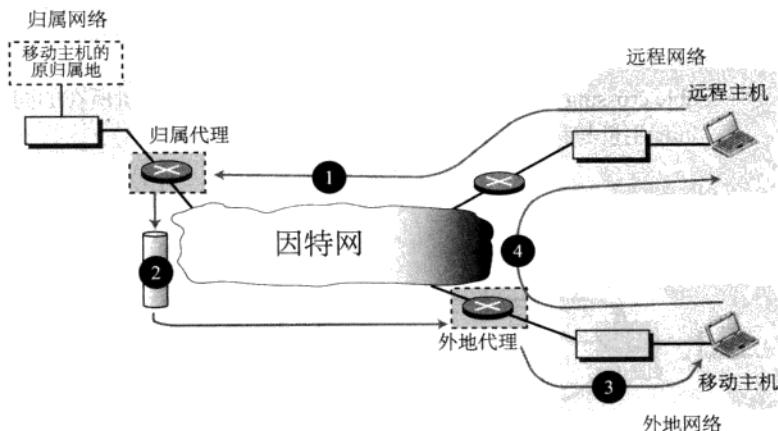


图 10.7 数据传送

从归属代理到外地代理

归属代理在收到这个分组后，就使用第 30 章要讨论的隧道技术，把分组发送给外地代理。归属代理把整个的 IP 分组封装在另一个 IP 分组中，并用自己的地址作为源地址而把外地代理的地址作为目的地址。图 10.7 中的路径 2 表示这个步骤。

从外地代理到移动主机

当外地代理收到这个分组后，从中取出原来的分组。但是，因为原来分组的目的地址是移动主机的归属地址，外地代理需要咨询登记表，找出移动主机的转交地址（否则这个分组就会又发回到归属网络）。然后，这个分组被发送到转交地址。图 10.7 的路径 3 表示这个步骤。

从移动主机到远程主机

当移动主机要发送分组给远程主机时（例如，对收到的分组的响应），它就像在正常情况下一样发送。移动主机准备分组，用它的归属地址作为源地址，用远程主机的地址作为目的地址。虽然这个分组是从外地网络发来的，但它使用了移动主机的归属地址。图 10.7 的路径 4 表示这个步骤。

透明性

在这个数据传送阶段，远程主机并不知道移动主机的任何移动。远程主机发送分组时使用的是移动主机的归属地址作为目的地址，它收到的分组也是用移动主机的归属地址作为源地址。移动是完全透明的。因特网的其余部分都不知道这个移动主机的移动性。

移动主机的移动性对因特网的其余部分是透明的。

10.4 移动 IP 的低效率

涉及到移动 IP 的通信效率较低。这种低效率可能很严重或者中等严重。很严重的情况称为两次穿越或 2X。中等严重的情况称为三角路由选择或狗腿路由选择。

10.4.1 两次穿越

两次穿越 (double crossing) 发生在移动主机已经移动到与远程主机所在的同一个网络 (或站点) 的情况下，远程主机与移动主机互相通信之时（见图 10.8）。

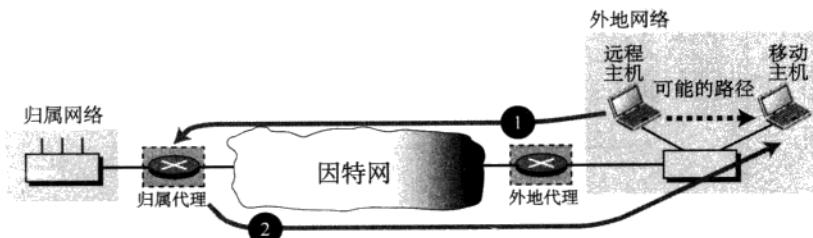


图 10.8 两次穿越

当移动主机向远程主机发送分组时，不存在低效率问题，因为通信就在本地进行。但是，当远程主机向移动主机发送分组时，这个分组就要穿越因特网两次。

由于计算机通常都是和另一个本地计算机进行通信（本地性原则），因此两次穿越的低效率问题是十分严重的。

10.4.2 三角路由选择

三角路由选择（triangle routing）的严重情况较轻，它发生在远程主机和移动主机进行通信时，它和移动主机并不连接在同一个网络（地点）上。当移动主机向远程主机发送分组时，没有低效率问题。但是，当远程主机向移动主机发送分组时，这个分组就要从远程主机到归属代理，然后再回到移动主机。分组走了一个三角形的两个边，而不仅仅是一个边（见图 10.9）。

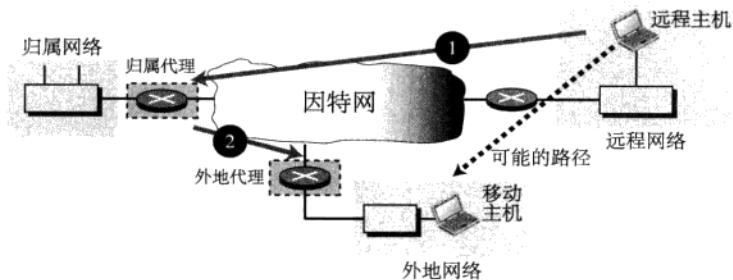


图 10.9 三角路由选择

10.4.3 解决方法

解决低效率问题的一个方法就是让远程主机把移动主机的转交地址和归属地址绑定起来。例如，当归属代理收到要发给移动主机的第一个分组时，它要把这个分组转发给外地代理，同时它还可以向远程主机发送一个更新绑定分组（update binding packet），使之以后发送给该移动主机的分组都发送到这个转交地址。远程主机可以把这个信息存放在高速缓存中。

这种策略的问题是，一旦移动主机又移动了，高速缓存中的这一项就变成过时的。在这种情况下，归属代理需要向远程主机发送告警分组（warning packet）以通知这种改变。

10.5 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

10.5.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]、[Kur & Ros 08]、[Gar & Vid 04]和[Pet & Dev 03]。

10.5.2 RFC

几个专门讨论了移动 IP 的 RFC 包括：RFC 1701、RFC 2003、RFC 2004、RFC 3024、RFC 3344 和 RFC 3775。

10.6 重要术语

代理通告	代理发现
代理询问	移动主机
转交地址	登记
同址转交地址	登记回答
两次穿越	登记请求
外地代理	固定主机
外地网络	三角路由选择
归属地址	更新绑定分组
归属代理	告警分组
归属网络	

10.7 本章小结

- 为移动通信而设计的移动 IP 是网际协议（IP）的增强版本。移动主机有一个归属地址和一个转交地址，归属地址是它在归属网络上的地址，而转交地址是它在外地网络上的地址。当移动主机连接到外地网络上时，归属代理把给移动主机的报文转发给外地代理。外地代理再把转发过来的报文发送给移动主机。
- 移动主机在它的归属网络上要通过称为代理发现的过程掌握归属代理的地址。移动主机在外地网络上时要通过代理发现或代理询问来掌握外地代理的地址。
- 当移动主机连接到外地网络上时，必须向归属代理和外地代理登记自己。
- 从远程主机发往移动主机的报文要先从远程主机到归属代理，再到外地代理，然后到移动主机。
- 移动通信可能是低效率的，因为报文必须走一段额外的距离。两次穿越和三角路由选择是低效率路由选择的两个例子。

10.8 实践安排

10.8.1 习题

1. 如果移动主机充当外地代理，那么还需要登记吗？请解释。

2. 如果移动主机充当外地代理，试重画图 10.7。
3. 创建一个归属代理通告报文，其序号为 1456，寿命为 3 小时。你自己选择代码字段每一位的值，并计算和插入长度字段的值。
4. 创建一个外地代理通告报文，其序号为 1672，寿命为 4 小时。你自己选择代码字段每一位的值。至少要有三个你选择的转交地址。计算并插入长度字段值。
5. 讨论为什么路由器询问报文也能够用于代理询问？为什么不需要额外的字段？
6. 在哪一个协议里有代理通告和询问报文？
7. 试给出习题 3 中的通告报文在 IP 数据报中的封装。协议字段值是多少？
8. 试解释为什么登记请求和回答不直接封装成 IP 数据报？为什么需要使用 UDP 用户数据报？

9. 我们有以下的信息：

移动主机归属地址：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

试给出从远程主机发送给归属代理的 IP 数据报首部的内容。

10. 根据习题 9 中的信息，给出归属代理发送给外地代理的 IP 数据报内容。使用隧道技术。
11. 根据习题 9 中的信息，给出外地代理发送给移动主机的 IP 数据报内容。
12. 根据习题 9 中的信息，给出移动主机发送给远程主机的 IP 数据报内容。
13. 在习题 9 中具有哪种类型的低效率？请解释。

10.8.2 研究活动

14. 我们讲过登记报文被封装在 UDP 中。找出为什么选择 UDP 而不是 TCP 的理由？
15. 试找出代理通告报文发送的频繁程度？
16. 试找出移动 IP 所需的不同的鉴别类型。
17. 试找出移动 IP 中多播的作用。

第 11 章 单播路由选择协议（RIP、OSPF 和 BGP）

正如我们在前面几章中讨论过的，单播通信是指在一个发送者和一个接收者之间的通信，也就是一对一的通信。在本章，我们将讨论路由器如何建立路由表以支持单播通信。我们将说明因特网如何被划分为一些称为自治系统的管理区域，这样做是为了更有效地处理路由信息。然后我们将解释在自治系统内部使用的两个主流路由选择协议，以及一个用于自治系统之间交换路由信息的协议。

目标

本章有以下几个目标：

- 介绍自治系统（AS）的思想，也就是为了便于交换路由信息而将因特网划分为若干个较小的可管理区域的思想。
- 讨论距离向量路由选择的思想，它是我们要讨论的第一个 AS 内部使用的路由选择方法，以及它是如何使用 Bellman–Ford 算法来更新路由表的。
- 讨论**路由信息协议（RIP）**如何被用于实现因特网中的距离向量路由选择的思想。
- 讨论链路状态路由选择的思想，它是我们要讨论的第二个 AS 内部使用的路由选择方法，以及它是如何使用 Dijkstra 算法来更新路由表的。
- 讨论**开放最短路径优先协议（OSPF）**如何被用于实现因特网中的链路状态路由选择的思想。
- 讨论作为主流的用于 AS 间路由选择方法的路径向量路由选择协议，并解释策略路由选择的概念。
- 讨论**边界网关协议（BGP）**如何被用于实现因特网中的路径向量路由选择的思想。

11.1 引言

互联网是由路由器连接起来的许多网络共同组成的。在数据报从源点到终点的过程中，它可能要通过多个路由器，直至抵达连接到目的网络的路由器为止。

11.1.1 代价或度量

路由器接收来自一个网络分组，并把这个分组转发到另一个网络。一个路由器通常会和多个网络相连。当路由器收到分组时，它应当将分组转发到哪一个网络呢？这个决定要基于最优化原则：哪一个可用的路径是最佳路径？术语最佳的定义又是什么？

有一种方法就是为通过的每一个网络指派一个代价（cost）。我们称这个代价为度量（metric）。代价高可以被认为是不好的，而代价低则被认为是好的。例如，如果我们希望让网络的吞吐量最大化，那么高吞吐量的就表示为低代价的，而低吞吐量的则表示为高代价的。再举一个例子，如果我们希望让网络的时延最小化，那么时延小的就是代价低的，时延大的就是代价高的。

11.1.2 静态路由表还是动态路由表

路由表可以是静态的或动态的。静态路由表是人工设置表项的路由表，而动态路由表则在互联网中某处有变化时就自动地进行更新。如今的互联网需要的是动态路由表。只要互联网中有了变化，路由表就应当尽快地更新。例如，当一条链路出了故障时，只要能够找到一条更好的路由，就应当更新路由表。

11.1.3 路由选择协议

由于需要动态的路由表，因此就产生了路由选择协议。路由选择协议是一些规则和过程的集合，使得在互联网中的各个路由器能够彼此互相通知变化信息。路由选择协议使路由器能够共享它们所知道的有关互联网或邻站的情况。这种信息的共享使得在旧金山的路由器能够知道得克萨斯的网络故障情况。路由选择协议还包括一些过程，用来合并从其他路由器收到的信息。

路由选择协议可以是内部协议或者是外部协议。内部协议处理域内路由选择，外部协议处理域间路由选择。我们就从对这几个术语的定义开始进入下一节。

11.2 域内和域间路由选择

今天的互联网可以是非常庞大的，以至于仅使用一种路由选择协议还无法处理更新所有路由器的路由表的任务。为此，互联网需要划分为多个自治系统。一个自治系统（autonomous system, AS）就是在管理机构管辖下的一组网络和路由器。在自治系统内部的路由选择称为域内路由选择。在自治系统之间的路由选择称为域间路由选择。每一个自治系统可以选择一个或多个域内路由选择协议来处理本自治系统内部的路由选择。但是，处理自治系统之间的路由选择只能使用一种域间路由选择协议，见图 11.1。

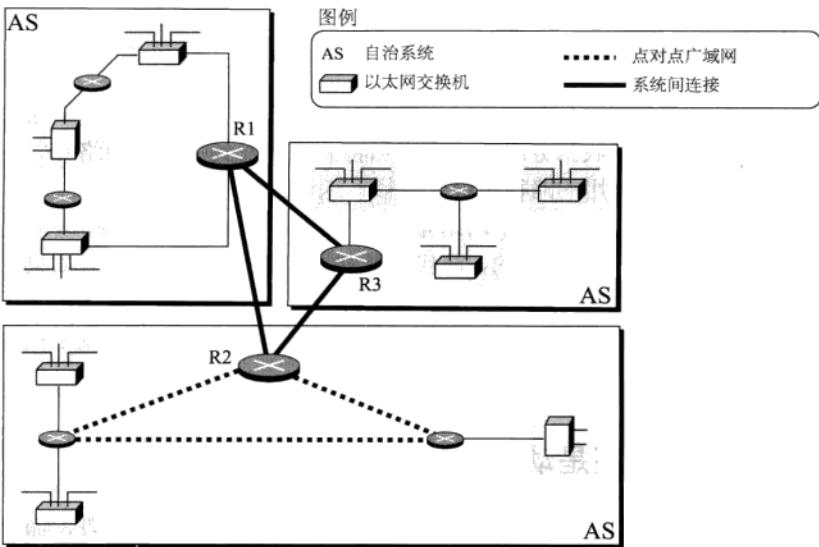


图 11.1 自治系统

已经有几个域内的以及域间的路由选择协议得到了应用。在本章，我们只讨论几个最流行的。我们要讨论两种域内路由选择协议：距离向量和链路状态。我们还要介绍一种域间路由选择协议：路径向量（见图 11.2）。

路由信息协议（RIP）是对距离向量协议的实现。开放最短路径优先（OSPF）协议是对链路状态协议的实现。边界网关协议（BGP）是对路径向量协议的实现。RIP 和 OSPF 是内部路由选择协议，而 BGP 是外部路由选择协议。

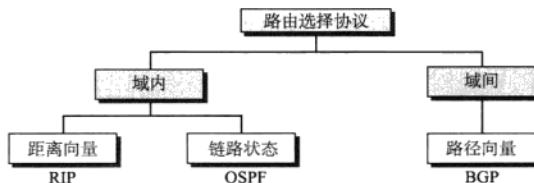


图 11.2 流行的路由选择协议

11.3 距离向量路由选择

我们首先来讨论距离向量路由选择 (distance vector routing)。这种方法视 AS 及其所有路由器和网络如同一张由结点以及连接结点的线 (边) 的集合构成的图。一个路由器通常表示为一个结点，而一个网络通常表示为连接两个结点之间的一条链路，虽然其他的表示方法也是可以的。目前，如果给定了结点和结点之间的距离，就可以使用图论中的一种称为 Bellman-Ford 的算法（有时也叫 Ford-Fulkerson 算法）来找出图中任意两个结点之间的

最短路径。我们首先来讨论一下这种算法，然后再来了解如何对这个算法做些修改就可以在距离向量路由选择中用它来更新路由表。

11.3.1 Bellman-Ford 算法

让我们先简单地讨论一下 Bellman-Ford 算法。这个算法可用于图论的很多应用中。如果我们知道每一对结点之间的代价，那么就可以用这个算法找出任意两个结点之间的最小代价（最短路径）。图 11.3 所示为一张由结点和链路组成的地图。在线的上方给出了每条线的代价，这个算法可以求出任意两个结点之间的最小代价。例如，如果这些结点代表的是城市，而线代表的是连接城市的道路，那么通过这张地图就能求出任意两个城市之间的最短距离。

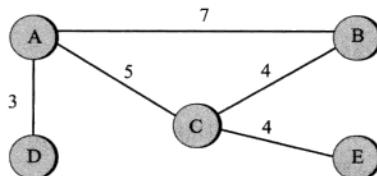


图 11.3 Bellman-Ford 算法的图

这个算法基于这样一个事实，如果结点 i 的所有邻站都知道到结点 j 的最短距离，那么求结点 i 和结点 j 之间的最短距离就可以用结点 i 到每个邻站之间的距离分别加上该邻站到结点 j 的最短距离，然后再从得数中选择最小的一个，如图 11.4 所示。

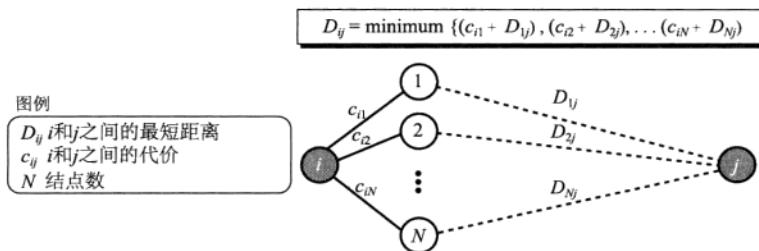


图 11.4 Bellman-Ford 算法

虽然 Bellman-Ford 算法的原理非常简单且容易理解，但是算法本身看起来却是循环的。那些邻站又是怎样计算出它们与终点之间的最短路径的呢？为了解决这个问题，要使用循环语句。我们用以下步骤为每个结点创建一个最短距离表（向量）：

1. 结点和它自己之间的最短距离和代价被初始化为 0。
2. 一个结点和任何其他结点之间的最短距离被设置为无穷大。一个结点和任何其他结点之间的代价应当给定（如果两个结点之间没有直接连接，可设置为无穷大）。
3. 图 11.4 中给出的算法不断循环，直至最短距离向量不再发生改变为止。

表 11.1 所示为这个算法的伪码。

表 11.1 Bellman-Ford 算法

```

1  Bellman_Ford()
2  {
3      //初始化
4      for (i = 1 to N; for j = 1 to N)
5      {
6          if (i == j)    Dij = 0    Cij = 0
7          Else        Dij = ∞    Cij = i 和 j 之间的代价
8      }
9      //更新
10     repeat
11     {
12         For (i = 1 to N;  for j = 1 to N)
13         {
14             Dij ← minimum[(Cil + Dlj)  ... (CiN + DNj)]
15         } //结束 for 循环
16     } until (上面循环不再导致 Dij 发生变化)
17 } //结束 Bellman-Ford 算法

```

11.3.2 距离向量路由选择算法

Bellman-Ford 算法非常适用于由城市和道路组成的交通地图，因为我们具有每个结点的所有初始信息，并且保持不变。我们可以将这些信息输入到计算机中，让计算机通过保存中间计算结果，产生每个结点最终的距离向量并输出。换言之，这个算法的设计思想是要同步地产生结果。如果我们希望利用这个算法为 AS 中的路由器创建路由表，那么就需要对这个算法做一些修改：

1. 在距离向量路由选择中，代价通常就是跳数（即在到达终点之前通过了多少个网络）。因此任意两个邻站之间的代价被设置为 1。
2. 每当路由器从它的邻站那里接收到一些信息时，它就要异步地更新自己的路由表。换言之，每个路由器只执行了整个 Bellman-Ford 算法中的一部分。这个处理过程是分布式的。
3. 在路由器更新了自己的路由表之后，应当将结果发送给它的所有邻站，以便这些邻站也能更新它们的路由表。
4. 每个路由器至少应当保存每条路由的三个信息：目的网络、代价和下一跳。我们称完整的路由表为 Table，表中的第 i 行为 Table_i ，第 i 行的三个列分别为 $\text{Table}_i.\text{dest}$, $\text{Table}_i.\text{cost}$ 和 $\text{Table}_i.\text{next}$ 。
5. 我们把来自邻站的一条路由信息称为一个 R（记录），它只包含了两个信息： $\text{R}.\text{dest}$ 和 $\text{R}.\text{cost}$ 。在收到的记录中不包含下一跳信息，因为下一跳就是发送方的源地址。

表 11.2 所示为这个算法的伪码。

表 11.2 在各个路由器上应用的距离向量算法

```

1 Distance_Vector_Algorithm()
2 {
3     //开始时
4     for (i = 1 to N)           //N 是端口数
5     {
6         Tablei.dest = 相连网络的地址
7         Tablei.cost = 1
8         Tablei.next = —      //表示就是它本身
9         将每一行的情况向每一个邻站发送一个记录 R
10    }   //结束循环
11
12 //更新
13 repeat (永远)           //N 是端口数
14 {
15     等待从邻站处来一个记录 R
16     Update (R, T)          //调用更新模块
17     For (i = 1 to N)       //N 是当前表的大小
18     {
19         将每一行的情况向每一个邻站发送一个记录 R
20     }
21 }   //结束重复
22
23 }   //结束距离向量算法
24 Update (R, T)           //更新模块
25 {
26     搜索表 T 找出与 R 中的 dest 相匹配的表项
27     if(在第 i 行找到了匹配的 dest)
28     {
29         if (R.cost + 1 < Ti.cost or R.next == Ti.next)
30         {
31             Ti.cost = R.cost + 1
32             Ti.next = 发送路由器的地址
33         }
34         else    丢弃该记录    //不需要做任何改变
35     }
36     else
37         //插入这个新的路由
38     {
39         TN+1.dest = R.dest

```

```

40   TN+1.cost = R.cost + 1
41   TN+1.next = 发送路由器的地址
42   根据目的地址对表进行排序
43   }
44 } //结束更新模块

```

从第 4 行到第 10 行是开始时的初始化过程。路由器创建一个初始的路由表，它只能用于将分组转发到与它的接口直接相连的网络。对于路由表中的每一个表项，它都要向每一个邻站发送一个记录，这个记录中只有两个字段：目的地址和代价。

每当路由器收到来自邻站的一个记录，就要更新自己的路由表。在更新之后，路由器又要将路由表中的每一个表项发送给它的每一个邻站，以便这些邻站各自完成更新。

从第 24 行到 44 行给出了详细的更新过程。当一个记录到达时，路由器搜索路由表以找到相应的目的地址。

- 如果相应的表项找到了，需要分两种情况检查并修改路由信息。

- 如果这个记录的代价加上 1 之后小于路由表中的相应代价，那就意味着该邻站发现了一条更好的路由可以到达此终点。

- 如果下一跳是相同的，则意味着某一部分网络发生过一些变化。例如，如果某邻站原先通告说到某终点的代价是 3，但是现在该邻站和终点之间没有路径了。该邻站通告说到这个终点的代价是无穷大。接收路由器绝对不能忽略这个数值，哪怕原先表项的代价值更小些。旧的路由不复存在了。

- 如果路由表中没有相应的表项，路由器就将其添加到路由表中，并根据目的地址对路由表进行排序。

例 11.1

图 11.5 所示为一个 AS 的初始路由表。请注意，这幅图并不是说所有的路由表都是在同一时间创建的，每个路由器会在启动时各建各自的路由表。

考试?????

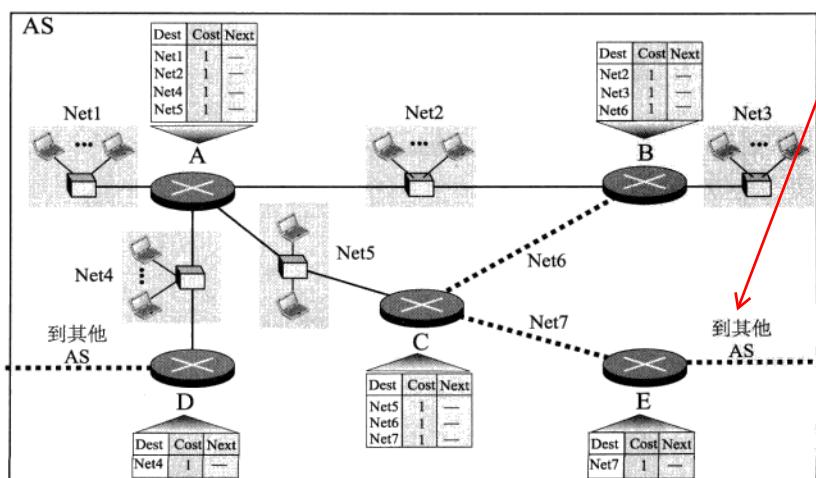


图 11.5 例 11.1

例 11.2

现在假设路由器 A 发送了四个记录到它的邻站：路由器 B、D 和 C。图 11.6 所示为路由器 B 在接收到这些记录后它的路由表的变化情况。我们把其他两个邻站的路由表的变化情况留作练习。

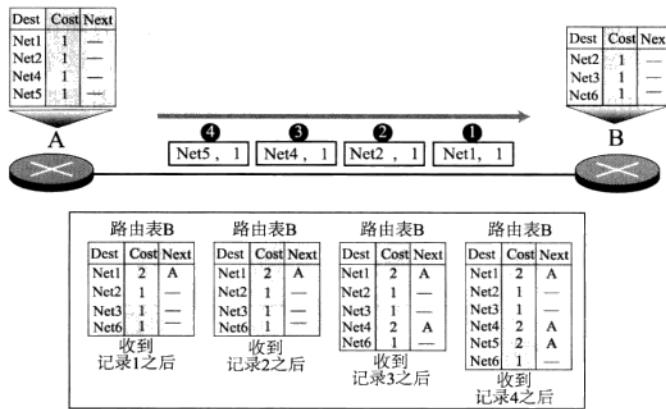


图 11.6 例 11.2

a. 当路由器 B 接收到记录 1 后，它搜索自己的路由表寻找 Net1 的路由，但是没有找到，所以它就将这个记录的代价再增加一跳（即 B 和 A 之间的距离），然后把记录添加到路由表中，并且设置下一跳为 A。

b. 当路由器 B 接收到记录 2 后，它搜索自己的路由表并在其中找到了终点 Net2 的表项。但是，因为记录 2 中的代价加 1 后大于路由表中相应的代价，所以记录 2 被弃之不理。

c. 当路由器 B 接收到记录 3 后，它搜索自己的路由表，因为没有找到 Net4，所以就在路由表中增加一个表项。

d. 当路由器 B 接收到记录 4 后，它搜索自己的路由表，因为没有找到 Net5，所以就在路由表中增加一个表项。

现在路由器 B 有了更多的信息，但还不完全。路由器 B 甚至还不知道 Net7 的存在，它还需要更进一步的更新。

例 11.3

图 11.7 所示为图 11.5 中各路由器的最终路由表。

A			B			C			D			E		
Dest	Cost	Next												
Net1	1	—	Net1	2	A	Net1	2	A	Net1	2	A	Net1	3	C
Net2	1	—	Net2	1	—	Net2	2	A	Net2	2	A	Net2	3	C
Net3	2	B	Net3	2	—	Net3	2	B	Net3	3	A	Net3	3	C
Net4	1	—	Net4	2	A	Net4	2	A	Net4	1	—	Net4	3	C
Net5	1	—	Net5	2	A	Net5	1	—	Net5	1	A	Net5	2	C
Net6	2	C	Net6	1	—	Net6	1	—	Net6	3	A	Net6	2	C
Net7	2	C	Net7	2	C	Net7	1	—	Net7	3	A	Net7	1	—

图 11.7 例 11.3

这个应该和各个路由器发送自己路由信息的时间有关系…

11.3.3 计数到无穷大

距离向量路由选择存在的一个问题就是任何有关代价下降的消息（好消息）都扩散得非常快，但是任何有关代价上升的消息（坏消息）则扩散得很慢。要想让路由选择协议能够正常工作，如果一条链路中断了（代价变为无穷大），那么其他所有路由器都应当立刻获知这一情况，但是在距离向量路由选择中，这是要花费一些时间的。这个问题就称为计数到无穷大（count to infinity）。需要经过多次更新才能使所有的路由器都把这条中断链路的代价记录为无穷大。

二结点循环

计数到无穷大的一个例子就是二结点循环问题。要了解这个问题，让我们来看看图 11.8 描绘的场景。

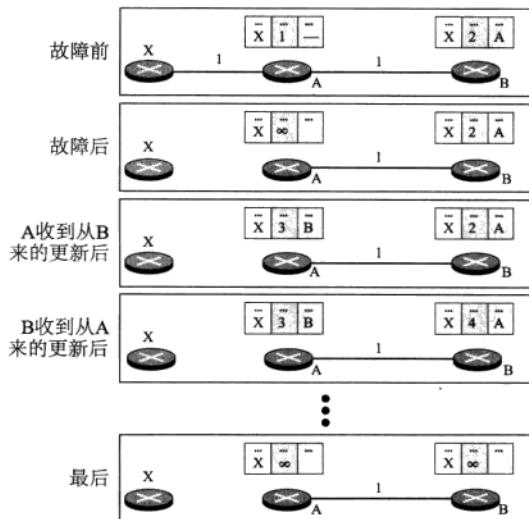


图 11.8 二结点的不稳定性

这个图描绘了一个有三个结点的系统。我们只给出了对我们的讨论有用的那部分路由表。在开始时，结点 A 和 B 都知道怎样到达结点 X。但是，在 A 和 X 之间的链路突然出了故障。结点 A 改变其路由表。如果 A 能够立即把它的路由表发送给 B，那么一切问题都没有了。但是，如果 B 在收到 A 的路由表之前已经向 A 先发送了它自己的路由表，那么这个系统就会变得不稳定。结点 A 收到这个更新，它认为 B 已经找到了到达 X 的路，于是立即对自己的路由表更新。然后 A 又将此次更新发送给 B。而此时 B 就认为 A 周围发生了一些变化，因而更新其路由表。到达 X 的代价在逐渐增大，直到增加到无穷大。此时，A 和 B 都知道了 X 是不可达的。但是在此期间，系统是不稳定的。结点 A 认为到 X 的路由要经过 B，而结点 B 认为到 X 的路由要经过 A。如果 A 收到了一个要发往 X 的分组，它就被转发到 B，然后又被转发到 A。同样，如果 B 收

到了一个要发往 X 的分组，它就被转发到 A，然后又被转发到 B。分组会在 A 和 B 之间来回传送，这就产生了二结点循环问题。为了解决此类不稳定性，已经提出了好几种方法。

定义无穷大 第一种最明显的解决方法是重新定义无穷大，把它定义为一个较小的数值，例如 16。对我们前面描述的场景中，该系统会在较少的几次更新内就达到稳定。事实上，距离向量协议的大多数实现都把 16 定义为无穷大。但是，这也意味着距离向量不能用于大系统中。在各个方向上，网络的大小都不能超过 15 跳。

分割范围 另一种解决方法称为分割范围 (split horizons)。这种策略就是不让每个结点通过所有接口用洪泛方法发送更新，而是只发送它的路由表的一部分。如果结点 B 根据其路由表认为到达 X 的最佳路由要经过 A，那么它就不需要再把这个信息通告给 A 了，因为这个信息就是从 A 来的 (A 已经知道了)。从结点 A 得到信息，修改后再发回给 A，这就是产生混乱的根源。在我们所描述的场景中，结点 B 在发送路由表给 A 之前要删除路由表的最后一行。在这种情况下，结点 A 保留到 X 的距离为无穷大。在此之后，当结点 A 将其路由表发送给 B 时，结点 B 也就更正了它的路由表。系统在第一次更新后就变稳定了，因为结点 A 和 B 都知道了 X 是不可达的。

分割范围和毒性逆转 使用分割范围策略有一个缺点。通常，距离向量协议使用一个计时器，若长时间没有关于某个路由的消息，就要从路由表中删除这个路由。在前面描述的场景中，当结点 B 在它给 A 的通告中删除了到 X 的路由时，结点 A 并不能猜出这是由于分割范围策略（因为信息的来源是 A），还是因为 B 最近一直都没有收到有关 X 的任何消息。分割范围策略可以与毒性逆转 (poison reverse) 策略组合起来使用。(译者注：这里的“毒性”表示度量为 16，表示网络中出现了故障。) 结点 B 可以仍然通知关于 X 的数值，但如果信息源是 A，就把距离换成为无穷大作为一种警告：“不要使用这个数值，我所知道的关于这条路由的信息来自于你。”

三结点的不稳定性

二结点的不稳定性可以用分割范围加上毒性逆转的方法来避免。但是，如果不稳定性发生在三个结点之间，稳定性就不能保证。图 11.9 描绘了此类场景。

假定在发现 X 不可达之后，结点 A 发送分组把这个情况通知 B 和 C。结点 B 立即更新其路由表，但发送给 C 的分组在网络中丢失了，因而永远不能到达 C。结点 C 完全不了解情况，仍然以为有一条距离为 2 的路由能够经过 A 到达 X。过了一会，结点 C 把它含有到 X 的路由的路由表发送给 B。此时结点 B 完全被愚弄了，它收到的信息是说有一条路由可以经过 C 到达 X。结点 B 根据算法更新了自己的路由表，指出有一条代价为 3 的路由可以经过 C 到达 X。这个信息来自 C 而不是 A，因此过一会结点 B 可能会把这个路由通知 A。现在连 A 也被愚弄了，它更新了自己的路由表，指出从 A 出发可以经过 B 到达 X，代价是 4。当然，这个循环会继续下去。现在 A 又把到 X 的路由的代价增加后通知给 C，但是不会给 B。C 又把增加了代价的路由通知给 B。B 向 A 又做了相同的事，如此循环往复。当每一个结点的代价都达到无穷大时，这个循环就结束了。

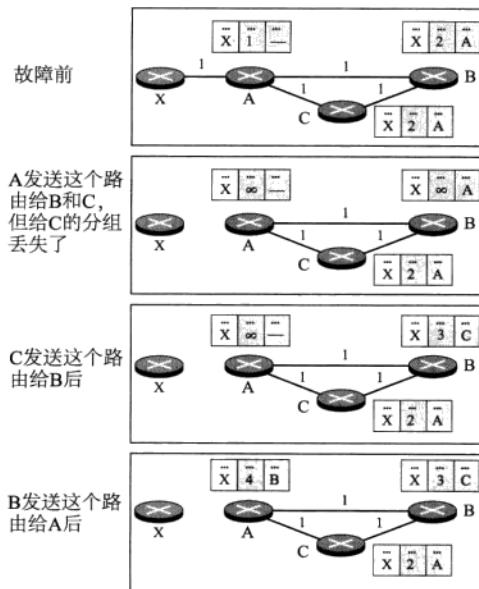


图 11.9 三结点的不稳定性

11.4 RIP

路由信息协议 (Routing Information Protocol, RIP) 是在一个自治系统中使用的域内 (内部) 路由选择协议。它是基于距离向量路由选择的非常简单的协议。RIP 直接实现距离向量路由选择，并有如下的一些考虑：

1. 在一个自治系统中，我们打交道的是路由器和网络（链路），它们被表示为结点。
2. 在路由表中，终点是一个网络，也就是说路由表的第一列定义的是一个网络地址。
3. RIP 使用的度量非常简单。距离定义为到达终点所需通过的链路（网络）数。正是由于这个原因，RIP 的度量称为跳数 (hop count)。
4. 无穷大被定义为 16，这表示在一个使用 RIP 的自治系统中，任何路由都不能超过 15 跳。
5. “下一个结点”这一列定义分组为了到达终点而要发往的路由器的地址。

图 11.10 描绘的是一个具有七个网络和四个路由器的自治系统。每一个路由器的路由表如图所示。让我们来观察 R1 的路由表。这个路由表有七个表项，用来指出在这个自治系统中应该怎样到达每一个网络。路由器 R1 直接连接到网络 130.10.0.0 和 130.11.0.0，也就是说这两个网络的表项中没有下一个结点。要向最左边的三个网络之一发送分组，路由器 R1 就需要先把这个分组交给 R2。因此这三个网络的表项的下一个结点就是路由器 R2 的接口，其 IP 地址是 130.10.0.1。要向最右边的两个网络之一发送分组，路由器 R1 就需要先把这个分组交给 R4 的接口，其 IP 地址是 130.11.0.1。其他的路由表可用类似的方法解释。

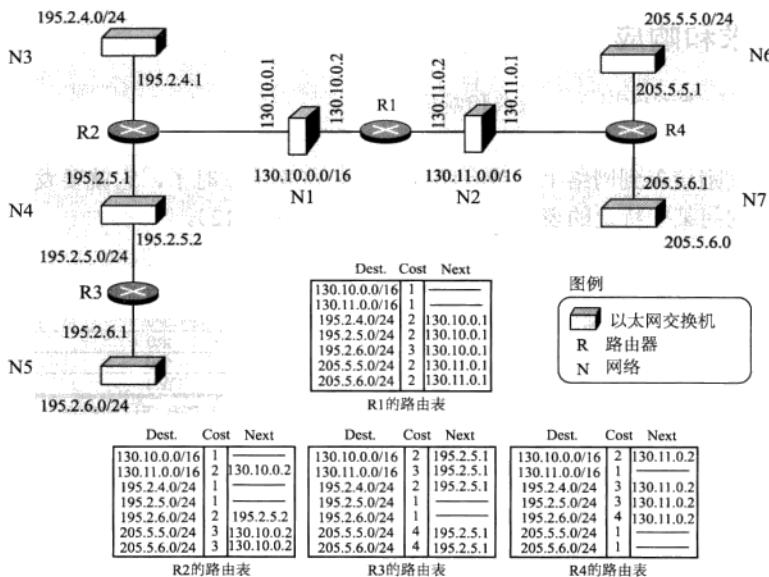


图 11.10 使用 RIP 的域的例子

11.4.1 RIP 的报文格式

RIP 报文的格式如图 11.11 所示。

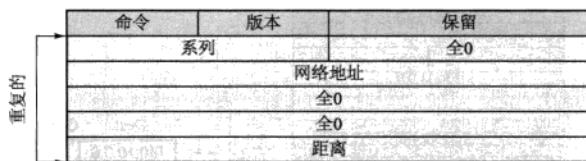


图 11.11 RIP 报文的格式

- **命令** 这个 8 位字段指明报文的类型：请求（1）或响应（2）。
- **版本** 这个 8 位字段定义了版本。在本书中我们使用版本 1，但在本节的最后，我们将给出版本 2 的某些新特点。
- **系列** 这个 16 位字段定义了所使用的协议系列。对于 TCP/IP 这个值是 2。
- **网络地址** 这个地址字段定义了目的网络的地址。RIP 为这个字段分配了 14 个字节，可用于任何协议。但是，IP 目前只使用 4 个字节。地址的其余部分全部填入 0。
- **距离** 这个 32 位字段定义了从发出通告的路由器一直到目的网络所经过的跳数。请注意，在这个报文中有一部分是重复出现的，每一个目的网络出现一次，我们把这部分称为表项（entry）。

11.4.2 请求和响应

RIP 使用两种类型的报文：请求和响应。

请求

当路由器刚刚接入到网络上，或者路由器有一些表项超时了，它就要发送请求报文。请求报文可以询问某些特定的表项或者所有表项（见图 11.12）。

命令: 1	版本	保留
系列	全0	
网络地址		
全0	全0	全0
全0	全0	全0
全0	全0	全0

(a) 对某个项目的请求

命令: 1	版本	保留
系列	全0	全0
网络地址		
全0	全0	全0
全0	全0	全0
全0	全0	全0

(b) 对所有项目的请求

图 11.12 请求报文

响应

响应可以是询问的或非询问的。询问的响应仅在回答请求时才发出，它包含了在对应的请求中指明的终点的信息。而非询问的响应则是定期发送，如每隔 30 秒或当路由表中有变化时。这种响应有时称为更新分组。

例 11.4

图 11.13 给出了从路由器 R1 发送到路由器 R2 的更新报文。报文从接口 130.10.0.2 发出。

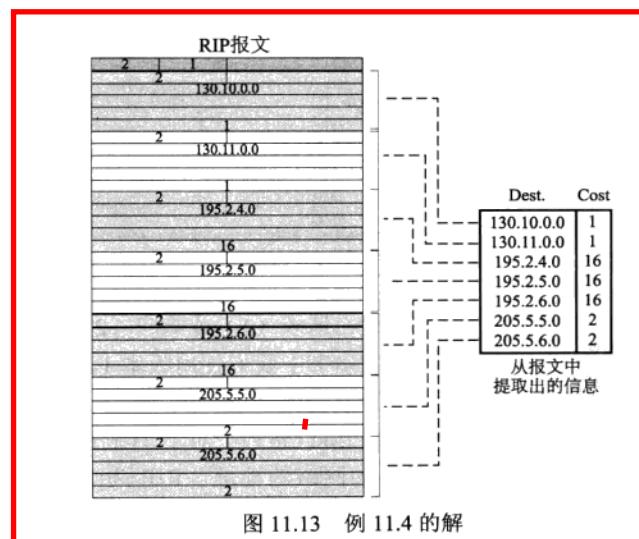


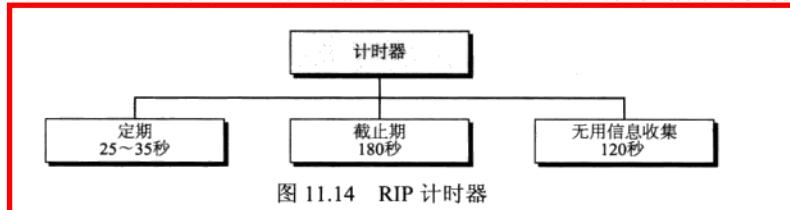
图 11.13 例 11.4 的解

在准备报文时要记得分割范围和毒性逆转策略的组合。路由器 R1 已经从路由器 R2 那里得到了三个网络的信息：195.2.4.0、195.2.5.0 和 195.2.6.0。当 R1 发送更新报文给 R2 时，它把对应于这三个网络的跳数的实际数值用 16（无穷大）替代，以防止 R2 产生任何混乱。这个图还给出了从这个报文中提取出的表。路由器 R2 使用 IP 数据报的源地址（130.10.0.2）

作为下一跳的地址，这个 IP 数据报携带了来自 R1 的 RIP 报文。路由器 R2 还要给每个跳数加上 1，因为报文中的值是相对于 R1 而不是 R2 的。

11.4.3 RIP 的计时器

RIP 使用了三个计时器来支持它的操作（见图 11.14）。定期计时器控制报文的发送，截止期计时器管理路由的有效性，而无用信息收集计时器则通知某个路由出了故障。



定期计时器

定期计时器 (periodic timer) 控制更新报文的定期发送。虽然协议指明这个计时器必须设置为 30 秒，但得到应用的模型使用的是 25~35 秒之间的一个随机数。这是为了避免路由器因同时更新而可能会引起的同步操作，从而导致互联网过载。

每个路由器都有一个定期计时器，设置为 25~35 秒之间的一个随机数。它是一个倒数计时器，当计时到零时就发送出更新报文，然后把计时器再随机地设置一次。

截止期计时器

截止期计时器 (expiration timer) 管理路由的有效性。当路由器收到路由更新信息时，截止期计时器就为这个特定的路由设置到 180 秒。每当收到该路由的一个新的更新时，截止期计时器就要复位。在正常情况下，每隔 30 秒发生一次复位。但是，如果互联网中出了问题，并且在分配的 180 秒内没有收到任何更新报文，那么这个路由就被认为是过期了，而路由的跳数就被设置为 16，这表示终点不可达。每一条路由有它自己的截止期。

无用信息收集计时器

当某一条路由的信息变成无效时，路由器并不立即在路由表中清除这条路由。相反，路由器继续通告这个路由的度量为 16。与此同时，一个称为无用信息收集计时器 (garbage collection timer) 的计时器就为该路由设置为 120 秒。当计数倒数到零时，相应的路由就从路由表中清除掉。这个计时器使得邻站在某个路由被清除之前能够了解该路由是无效的。

例 11.5

某路由表有 20 个表项。有 5 条路由在 200 秒的时间内都没有收到过它们的信息。这时共有多少个计时器在运行？

解

共有如下所示的 21 个计时器：

定期计时器：1

截止期计时器： $20 - 5 = 15$

无用信息收集计时器：5

11.4.4 RIP 版本 2

设计 RIP 版本 2 是为了克服版本 1 的某些缺点。版本 2 的设计者没有增加报文中每个表项的长度。他们只是把版本 1 中那些对 TCP/IP 协议填入 0 的那些字段改为一些新的字段。

报文格式

图 11.15 给出了 RIP 版本 2 报文的格式。这个报文的几个新的字段如下：

- 路由标记** 这个字段携带了如自治系统号这样的信息。它可用来使 RIP 能够从域间路由选择协议中接收信息。
- 子网掩码** 这个 4 字节的字段携带的是子网掩码（或前缀）。也就是说 RIPv2 可支持无分类编址和 CIDR。
- 下一跳地址** 这个字段表示下一跳地址。若两个自治系统共享一个网络（例如，主干网），则这个字段就特别有用。这个报文就可以定义分组必须发往的路由器，不管这个路由器是在同样的自治系统或在另一个自治系统。

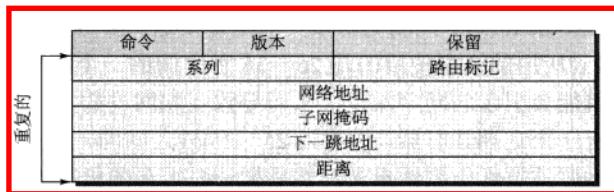


图 11.15 RIP 版本 2 的格式

无分类编址

RIP 的这两个版本之间的最重要的区别，可能是分类编址还是无分类编址。RIPv1 使用分类编址。报文格式中只有一个表项用于网络地址（掩码是默认的）。RIPv2 为子网掩码增加了一个字段，它可用来定义网络前缀的长度。这就表示在这个版本中，我们可以使用无分类编址。一组网络可以合并为一个前缀，也可以合起来一并通知，如我们在第 5 和第 6 章中看到的那样。

鉴别

增加鉴别是为了保护报文防止未授权的通告。分组没有增加新的字段，但是报文的第一个表项被用于鉴别信息。为了指出这个表项是鉴别信息而不是路由选择信息，在系列字段中放入了 $FFFF_{16}$ （见图 11.16）。第二个字段是鉴别类型，定义了鉴别所使用的协议，而第三个字段则包含真正的鉴别数据。

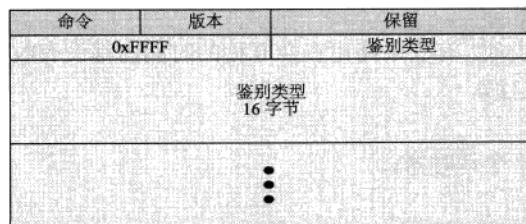
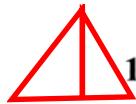


图 11.16 鉴别

多播

RIP 版本 1 使用广播方式把 RIP 报文发送给每一个邻站。使用这种方法时，不仅网络上的所有路由器都会接收到这些分组，而且所有的主机也都会接收这些分组。但 RIP 版本 2 使用了全路由器（all-router）多播地址把 RIP 报文仅发送给这个网络上的 RIP 路由器。



11.4.5 封装

RIP 报文被封装在 UDP 用户数据报中。RIP 报文不包括指示报文长度的字段。这可以从 UDP 分组来确定。在 UDP 中，指派给 RIP 的是熟知端口 520。

RIP 在熟知端口 520 使用 UDP 的服务。

11.5 链路状态路由选择

链路状态路由选择的原理与距离向量路由选择的原理不同。使用链路状态路由选择时，如果某个域的每一个结点都有这个域的完整拓扑，也就是说具有这个域的结点和链路的列表并知道它们是怎样连接起来的，包括类型、代价（度量）和链路的状态（正常工作或故障），那么这个结点就能够使用 Dijkstra 算法构造一个路由表。图 11.17 给出了这个概念。

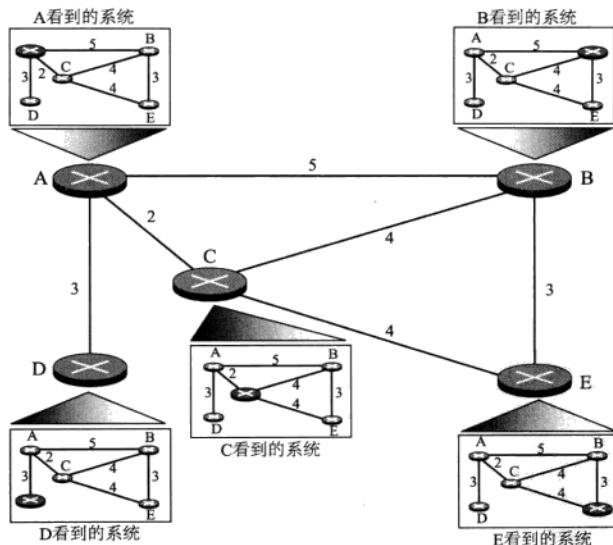


图 11.17 链路状态路由选择的概念

图中描绘了一个简单的有五个结点的域。每个结点都是利用相同的拓扑来创建路由表，但是每个结点的路由表却是独一无二的，因为路由表的计算是基于对这个拓扑的不同解释。这和一个城市的地图相似。居住在不同城市的两个人可以有相同的地图，但每个人需要走不同的路径才能到达目的地。

这个拓扑必须是动态的，它代表了每一个结点和每一条链路的最新状况。如果网络中的某处出现了变化（例如，一条链路出故障），那么每一个结点所知的拓扑必须更新。

一个公共的拓扑怎样才能成为动态的，并且被存储在每一个结点中？没有哪个结点能够在一开始就知道这个拓扑，或者在网络中的某处发生变化后就立即知道。链路状态路由选择是基于这样的假定：虽然没有这个拓扑的全局知识，但每一个结点知道其中的一部分。每个结点都知道它自己的链路的状态（类型、状态及代价）。换言之，整体拓扑可以从每一个结点的部分知识汇集而成。图 11.18 中描绘的是与前图一样的域，图中指出了属于每一个结点的部分知识。

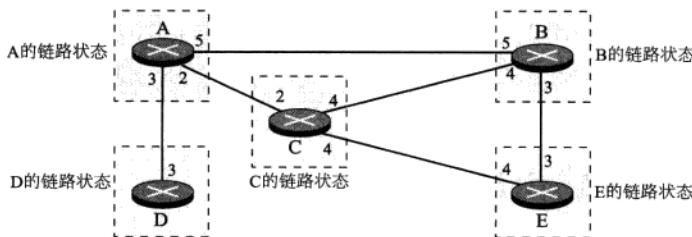


图 11.18 链路状态的知识

结点 A 知道它以度量 5 连接到结点 B，以度量 2 连接到 C，以度量 3 连接到 D。结点 C 知道它以度量 2 连接到结点 A，以度量 4 连接到 B，以度量 4 连接到 E。结点 D 知道它仅以度量 3 连接到结点 A。依此类推。虽然这些知识有重叠部分，但这种重叠能够保证产生一个公共的拓扑：给每一个结点提供一个完整域的图。

11.5.1 构造路由表

在链路状态路由选择 (link state routing) 中，需要四组动作来确保每一个结点的路由表能给出到达其他各结点的最小代价结点。

1. 每个结点先创建一个有关链路状态的分组，称为链路状态分组或 LSP (Link State Packet)。
2. 以可靠和有效的方法向其他各个结点散发 LSP，称之为洪泛 (flooding)。
3. 为每个结点形成一个最短路径树。
4. 基于这个最短路径树计算路由表。

链路状态分组的创建

链路状态分组 (LSP) 可以携带大量的信息。但目前我们假定它携带了最少量的数据：结点标识、链路列表、一个序号以及寿命。其中前两个（结点标识和链路列表）是为构造拓扑所必需的。第三个（序号）是洪泛要用到的，并且可以把新的 LSP 和旧的 LSP 区分开。第四个（寿命）用来防止旧的 LSP 长时间在域中逗留。以下两种场合会产生 LSP。

1. 当这个域的拓扑发生变化时。触发 LSP 的散发是快速通知域中所有结点更新其拓扑的主要方法。
2. 基于定期更新。这种情况下的定时周期要比距离向量路由选择大得多。实际上并不需要用它来散发 LSP。这样做只是为了确保旧的信息可以从这个域中删除。对于定期散发

计时器通常设置为 60 分钟或 2 小时，视不同的具体实现而定。设置这么长的周期是为了确保洪泛不会在网络上产生太大的通信量。

LSP 的洪泛

在结点准备好一个 LSP 后，它必须向其他所有结点散发，而不仅仅是向其邻站散发。这个过程称为洪泛，它基于以下步骤：

1. 产生 LSP 的结点把 LSP 的副本从它的每一个接口发送出去。
2. 结点把收到的 LSP 和可能已经有的副本进行比较。如果新到达的 LSP 比原有的还旧（检查序号即可知道），就丢弃这个 LSP。如果它比较新，这个结点就进行以下操作。
 - a. 丢弃旧的 LSP，保留新的。
 - b. 除了这个分组到达的那个接口外，从其他所有接口发送这个 LSP 副本。这就保证了洪泛会在这个区域的某处（只有一个接口的结点）停止。

形成最短路径树：Dijkstra 算法

在收到所有的 LSP 后，各个结点就会有一个完整拓扑的副本。但是，仅仅用这个拓扑还不足以找到到其他所有结点的最短路径，我们还需要使用最短路径树（shortest path tree）。

树就是由结点和链路构成的图，其中有一个结点称为根。可以从这个根经过仅一条路径到达所有的其他结点。最短路径树就是这样的树，从根到每一个其他结点之间的路径都是最短的。Dijkstra 算法可用来从一个给定的图产生一个最短路径树。这个算法使用了以下步骤。

1. **初始化：**选择作为树的根的结点，并把它加到路径中。为根的所有邻站设置从根到这些邻站之间的最短距离。设计根到它自己的最短距离为零。
2. **循环：**重复下面两个步骤，直至所有结点都被加入到路径中。
 - a. **加入下一个结点到路径中：**搜索不在路径中的结点。选择一个具有最小的最短距离的结点，把它加入到路径中。
 - b. **更新：**用刚才在第 2 步中移到路径中的结点来更新所有剩余结点的最短距离。

$$D_j = \min(D_j, D_i + c_{ij}) \quad \text{对所有剩余结点}$$

表 11.3 所示为这个算法的一个简单版本。

表 11.3 Dijkstra 算法

```

1  Dijkstra ()
2  {
3    //初始化
4    路径 = {s}                      //s 表示自己
5    for (i = 1 to N)
6    {
7      if (i 是 s 的邻站，且 i ≠ s)   Di = csi
8      if (i 不是 s 的邻站)        Di = ∞
9    }
10   Ds = 0
11
12 }    //Dijkstra
13 //循环
14 Repeat

```

```

15  {
16      //找出要加入的下一个结点
17      路径 = 路径 ∪ i    如果  $D_i$  是所有剩余结点中最小的
18
19      //更新其他结点的最短距离
20      For (j = 1 to M)      //M 是剩余结点数
21      {
22          {
23               $D_j = \min(D_j, D_j + c_{ij})$ 
24          }
25      } until (所有结点都被包括到路径中, M 等于 0)

```

图 11.19 描绘了为具有七个结点的图构造最短路径树的过程。图中的所有结点都具有相同的拓扑，但每个结点会以自己为树根创建不同的最短路径树。我们所描绘的树是由结点 A 创建的。为了得到这个最短路径树，我们需要经过一个初始化步骤和六次循环。

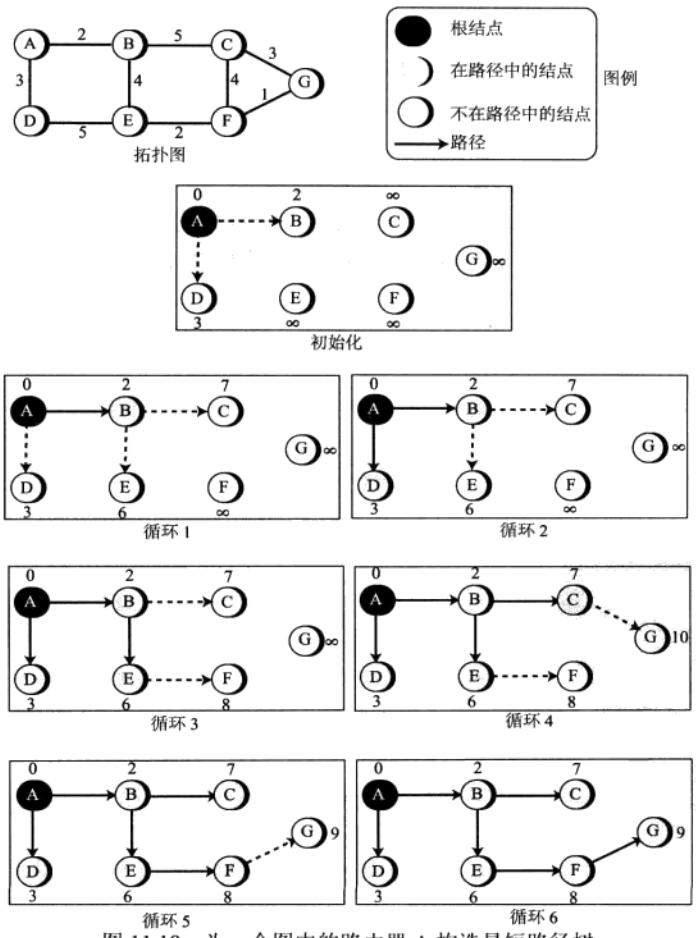


图 11.19 为一个图中的路由器 A 构造最短路径树

在初始化阶段，结点 A 选择自己作为树的根。然后它为拓扑中的每个结点指派最短距离。那些不是 A 的邻站的结点被分配到的最短距离值是无穷大。

在每次循环时，具有最小距离的结点被选做下一个结点并加入到路径中。然后剩余所有结点的最短距离都要根据刚才入选的结点进行更新。例如，在第一个循环中，结点 B 被选中并加入到路径中，最短距离在考虑到结点 B 的情况下进行更新（C 和 E 的最短距离改变了，但其他结点保持不变）。在经过六次循环之后，结点 A 的最短路径树就建好了。请注意，在第四次循环时发现到 G 的最短路径要经过 C，但是在第五次循环时又发现了一个新的最短路由（经过 F），于是前面的路径被取消，换成新的路径。

例 11.6

为了说明每一个结点的最短路径树都是不同的，我们找出了从结点 C 的角度得到的最短路径树（图 11.20）。具体细节留作练习。

从最短路径树计算路由表

每一个结点使用我们在前面所讨论的最短路径树来构造其路由表。路由表给出从根到每一个结点的代价。表 11.4 给出了图 11.19 中结点 A 使用最短路径树构造的路由表。

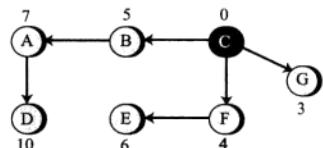


图 11.20 例 11.6

表 11.4 结点 A 的路由表

终点	代价	下一路由器
A	0	—
B	2	—
C	7	B
D	3	—
E	6	B
F	8	B
G	9	B

11.6 OSPF

开放最短路径优先（Open Shortest Path First, OSPF）协议是一种基于链路状态路由选择的域内路由选择协议。它的域也就是一个自治系统。

11.6.1 区域

为了能够更加有效迅速地处理路由选择问题，OSPF 把一个自治系统划分为若干区域。一个区域（area）就是包含在自治系统中的一些网络、主机和路由器的集合。一个自治系统可划分为若干个不同的区域。在一个区域里的所有网络必须是互相连接的。

在一个区域内的路由器使用洪泛法传送路由选择信息。在一个区域的边界，有一些特

殊的路由器称为区域边界路由器 (area border router)，它们把有关本区域的信息汇总起来发送到其他区域。在自治系统中还有一个特殊的区域称为主干。在自治系统中的其他所有区域必须连接到主干上。换言之，主干相当于一级区域，而其他的区域相当于二级区域。但这并不表示各区域内部的路由器不能互相连接。在主干中的路由器称为主干路由器 (backbone router)。请注意，主干路由器同时也可以是一个区域边界路由器。

如果由于某些问题，在主干和区域之间的连通性被破坏了，那么管理员必须在路由器之间创建一条虚链路 (virtual link)，以使得作为一级区域的主干的各种功能能够继续下去。

每个区域都有一个区域标识。主干的区域标识是零。图 11.21 给出了一个自治系统以及它的各个区域。

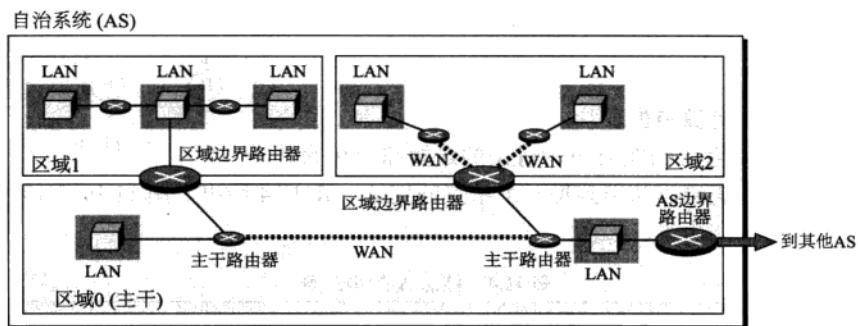


图 11.21 自治系统中的区域

11.6.2 度量

OSPF 协议允许管理员给每条路由指派一个代价，称之为度量 (metric)。度量可以基于服务的类型（最小延时、最大吞吐量，等等）。事实上，一个路由器可以有多个路由表，而每一个路由表基于不同的服务类型。

11.6.3 链路的类型

在 OSPF 术语中，一个连接称为一条链路。已定义了四种类型的链路：点对点链路、穿越链路、残桩 (stub) 链路和虚拟链路（见图 11.22）。

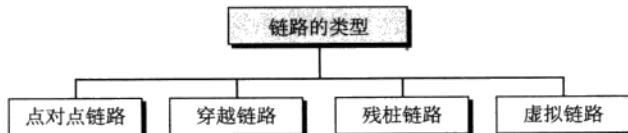


图 11.22 链路的类型

点对点链路

点对点链路 (point-to-point link) 直接连接两个路由器，中间没有任何其他的主机或路由器。换言之，这条链路（网络）的目的仅仅是为了连接这两个路由器。这种类型链路的例子

就是两个路由器用一条电话线或一条 T 级电信专线 (美国电信运营商提供。译者) 连接起来。没有必要给这种类型的链路指派一个网络地址。用图表示时, 路由器用结点表示, 而链路用一条连接两个结点的双向边来表示, 度量写在两端, 各表示每一个方向的度量, 通常这两个度量是一样的。换言之, 每一个路由器只有一个邻站在链路的另一端 (见图 11.23)。



图 11.23 点对点链路

穿越链路

穿越链路 (transient link) 是连接了若干个路由器的一个网络。数据可以从任何一个路由器进入网络, 并从任何一个路由器离开网络。所有具有两个或更多路由器的局域网和一些这样的广域网都属于此种类型。在这种情况下, 每个路由器有多个邻站。例如考虑图 11.24 (a) 中的以太网。路由器 A 的邻站是路由器 B、C、D 和 E。路由器 B 的邻站是 A、C、D 和 E。如果我们要表示这种情况下的邻站关系, 就使用图 11.24 (b) 来表示。

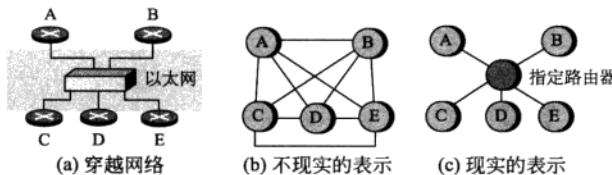


图 11.24 穿越链路

但是这样做既低效又不现实。说它低效是因为每一个路由器都需要向另外四个路由器做邻站通告, 总共就有 20 个通告。说它不现实是因为实际上不会在每一对路由器之间都有一个单独的网络 (链路), 而是有一个网络为五个路由器提供像十字路口一样的服务。

为了表示路由器通过一个网络连接到其他各个路由器, 我们要把这个网络表示成一个结点。但是, 网络并不是某一种机器, 它不能像路由器那样工作。因此网络中的一个路由器就要担负起这个责任。这个路由器被分配了双重任务, 它既是一个真正的路由器, 又是一个指定的路由器。我们用图 11.24 (c) 中所示的拓扑来表示穿越网络的连接。

图中每个路由器都只有一个邻站, 即指定路由器 (网络)。从另一方面看, 这个指定路由器 (即网络) 有五个邻站。我们可以看到, 邻站通告的数量从 20 减少到 10, 而链路仍然用连接结点的双向边来表示。虽然从每个结点到指定路由器都有一个度量, 但从指定路由器到任何其他结点都没有度量。原因就是指定路由器代表了网络。我们只能为分组通过网络指派一次代价, 不能为它计算两次。当分组进入网络时, 我们指派一个代价, 当分组离开网络到其他的路由器时就没有代价。

残桩链路

残桩链路 (stub link) 是仅连接了一个路由器的网络。数据分组通过这个路由器进入网络, 而离开网络也是通过相同的路由器。这是穿越网络的一种特例。对于这种情况, 我们可以用一个结点来表示这个路由器, 并用指定路由器代表网络。但是, 这条链路是单向的,

即只能从路由器到网络（见图 11.25）。



图 11.25 残桩链路

虚拟链路

当两个路由器之间的链路断开时，管理员就要在它们之间用一条更长的路径来创建一条虚拟链路（virtual link），这可能要经过好几个路由器。

11.6.4 图形表示法

让我们来看看怎样用图形表示一个自治系统。图 11.26 画的是具有 7 个网络和 6 个路由器的小型自治系统。其中有两个网络是点对点网络。我们使用像 N1 和 N2 这样的符号来代表穿越网络和残桩网络，但没有必要给点对点网络指派一个号。这个图还给出了 OSPF 对自治系统的图形表示。

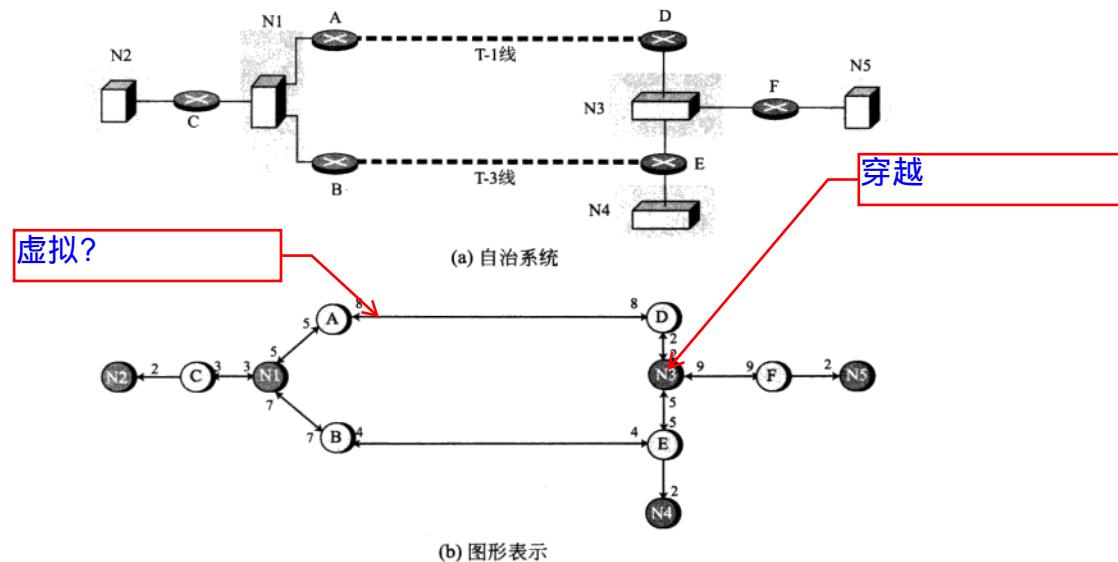


图 11.26 AS 的一个例子和它在 OSPF 中的图形表示

我们使用浅灰色结点表示路由器，用深灰色结点表示网络（由指定路由器代表的网络）。不过 OSPF 认为它们都是结点。请注意，这里我们共有 3 个残桩网络。

11.6.5 OSPF 分组

OSPF 使用五种不同类型的分组：问候分组、数据库描述分组、链路状态请求分组、链

路状态更新分组以及链路状态确认分组（见图 11.27）。最重要的是链路状态更新分组，它本身就有五个不同的种类。



图 11.27 OSPF 分组的类型

公共首部

所有的 OSPF 分组都有相同的公共首部（见图 11.28）。在学习不同类型的分组之前，让我们先讨论一下这个公共首部。

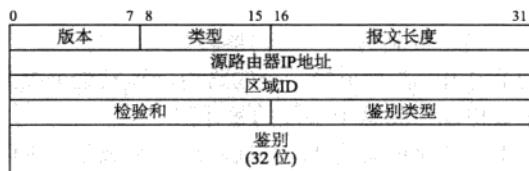


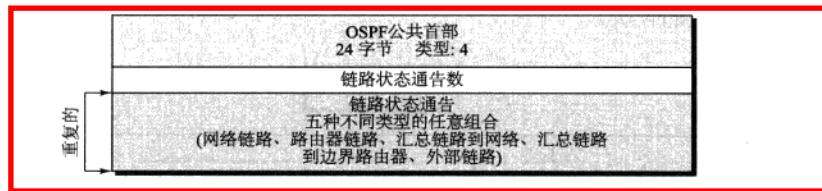
图 11.28 OSPF 的公共首部

- 版本** 这个 8 位字段定义了 OSPF 协议的版本。它目前是版本 2。
- 类型** 这个 8 位字段定义了分组的类型。如前所述，我们共有五种类型，分别用值 1 至 5 来定义这些类型。
- 报文长度** 这个位字段定义了包括首部在内的总报文长度。
- 源路由器 IP 地址** 这个 32 位字段定义了发送这个分组的路由器的 IP 地址。
- 区域标识** 这个 32 位字段定义了进行路由选择的区域。
- 检验和** 这个字段用来对整个分组进行差错检测，但不包括鉴别类型字段和鉴别数据字段。
- 鉴别类型** 这个 16 位字段定义了在这个区域内使用的鉴别协议。目前已定义了两种类型的鉴别：0 表示没有鉴别，1 表示口令。
- 鉴别** 这个 64 位字段放的是鉴别数据真正的值。今后当定义出更多类型的鉴别时，这个字段将包含鉴别计算的结果。在目前，若鉴别类型是 0，则这个字段就填入 0。若类型是 1，这个字段就携带 8 字符的口令。

11.6.6 链路状态更新分组

我们首先要讨论链路状态更新分组（link state update packet），它是 OSPF 运行的核心。

路由器用它来通告自己的链路状态。链路状态更新分组的通用格式如图 11.29 所示。



每个更新分组可包含几个不同的 LSA（链路状态通告）。所有五种 LSA 具有相同的通用首部。这个通用首部如图 11.30 所示，描述如下。

链路状态寿命	保留	E	T	链路状态类型
链路状态ID				
发送通告的路由器				
链路状态序号				
链路状态检验和	长度			

图 11.30 LSA 的通用首部

- **链路状态寿命** 这个字段指出从这个报文第一次产生后所经历的秒数。回顾这种类型的报文是从一个路由器到另一个路由器转发的（洪泛）。当路由器创建这个报文时，这个字段的值是 0。在一个接一个的路由器转发这个报文时，就要估计传送的时间，并把这个数值加入到这个字段的累计值中。
- **E 标志** 若这个 1 位的字段置为 1，则表示这个区域是残桩区域。残桩区域是只有一条路径连接到主干区域的一种区域。
- **T 标志** 若这个 1 位的字段置为 1，则表示这个路由器能够处理多种类型的服务。
- **链路状态类型** 这个字段定义 LSA 的类型。如我们前面曾提到，共有五种不同的通告类型：路由器链路（1）、网络链路（2）、汇总链路到网络（3）、汇总链路到 AS 边界路由器（4）和外部链路（5）。
- **链路状态标识符** 这个字段的值取决于链路的类型。对于类型 1（路由器链路），它就是路由器的 IP 地址。对于类型 2（网络链路），它就是指定路由器的 IP 地址。对于类型 3（汇总链路到网络），它是网络的 IP 地址。对于类型 4（汇总链路到 AS 边界路由器），它是 AS 边界路由器的 IP 地址。对于类型 5（外部链路），它是外部网络的 IP 地址。
- **发送通告的路由器** 这是发送这个报文的路由器的 IP 地址。
- **链路状态序号** 这是指派给每个链路状态更新报文的序号。
- **链路状态检验和** 这不是常见的检验和字段。它使用一种特殊的检验和计算方法，称为 Fletcher 检验和（见附录 C），它的值取决于除寿命字段外的整个分组。
- **长度** 这个字段定义了整个分组的长度，以字节为单位。

路由器链路 LSA

路由器链路定义的是连接真路由器的一条链路。真路由器使用这个通告来宣布它的链路的相关信息，以及链路的另一端是什么（邻站）。图 11.31 描绘了一条路由器链路。

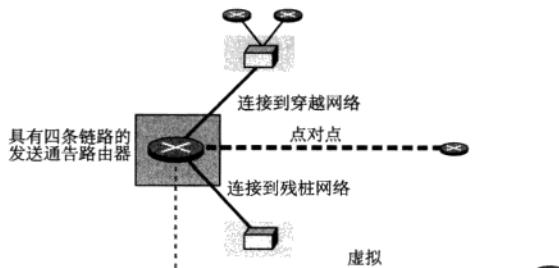


图 11.31 路由器链路

路由器链路 LSA 通告了路由器（真路由器）的所有链路。路由器链路分组的格式如图 11.32 所示。

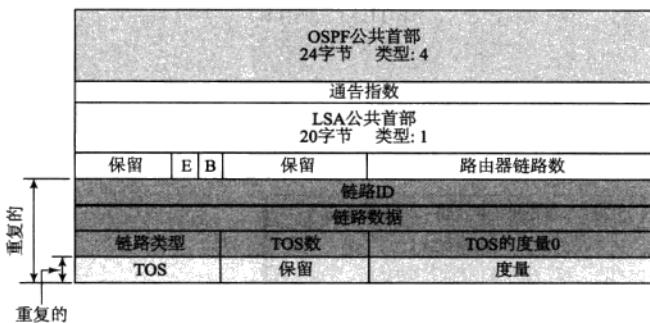


图 11.32 路由器链路 LSA

路由器链路 LSA 的字段如下：

- **链路标识** 这个字段的值取决于链路的类型。表 11.5 给出了基于链路类型的不同链路标识。
- **链路数据** 这个字段给出关于该链路的附加信息。同样，它的值也取决于这条链路的类型（见表 11.5）。

表 11.5 链路类型、链路标识和链路数据

链路类型	链路标识	链路数据
类型 1：点对点	邻站路由器地址	接口数
类型 2：穿越	指定路由器地址	路由器地址
类型 3：残桩	网络地址	网络掩码
类型 4：虚拟	邻站路由器地址	路由器地址

- **链路类型** 基于这个路由器连接到的网络的类型，共定义了四种不同的链路类型（见表 11.5）。
- **服务类型 (TOS) 数** 这个字段定义每一条链路所宣布的服务类型数。
- **TOS 0 的度量** 这个字段定义默认的服务类型 (TOS 0) 的度量。

TOS 这个字段定义服务类型。

度量 这个字段定义相应的 TOS 的度量。

例 11.7

试给出图 11.33 中由路由器 10.24.7.9 发出的路由器链路 LSA。

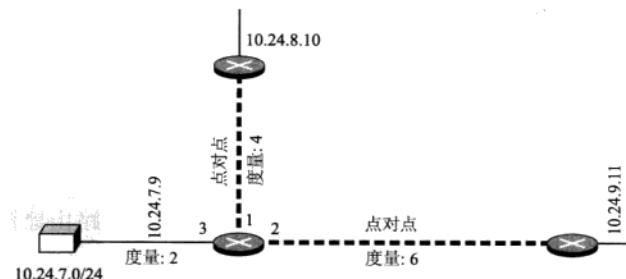


图 11.33 例 11.7

解

这个路由器有三条链路：两条是类型 1（点对点），一条是类型 3（残桩网络）。图 11.34 给出了这个路由器链路 LSA。

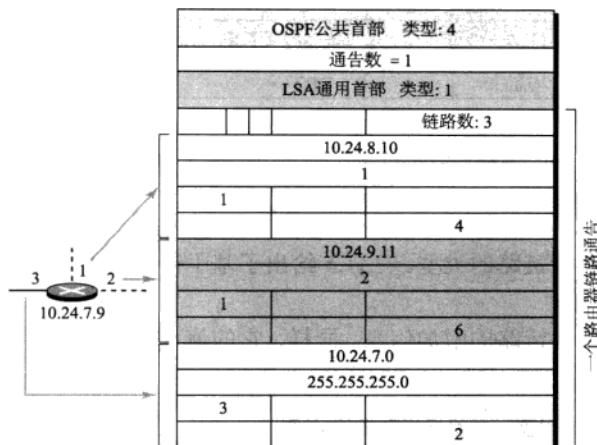


图 11.34 例 11.7 的解

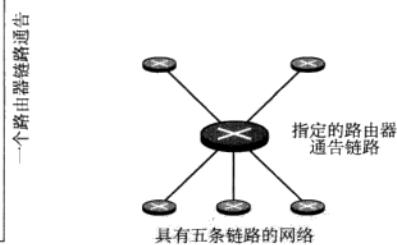


图 11.35 网络链路

网络链路 LSA

网络链路定义的是连接网络的一条链路。一个指定的路由器代表穿越网络来发布这种类型的 LSP。这个分组宣布了连接到本网络上的所有路由器的存在（见图 11.35）。网络链路通告的格式如图 11.36 所示。网络链路 LSA 的各字段如下：

网络掩码 这个字段定义了网络掩码。

连接的路由器 这个重复出现的字段定义了所有相连路由器的 IP 地址。

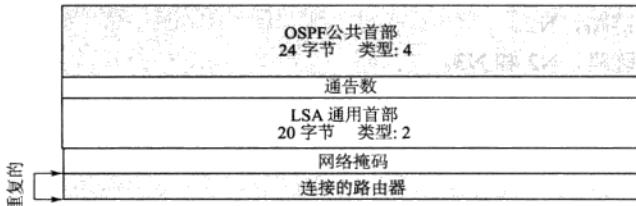


图 11.36 网络链路通告格式

例 11.8

试给出图 11.37 中的网络链路 LSA。

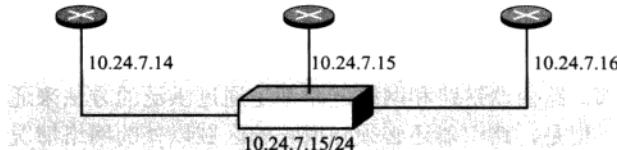


图 11.37 例 11.8

解

发出网络链路通告的网络共连接了三个路由器。LSA 给出其掩码和路由器地址。图 11.38 给出了这个网络链路 LSA。

OSPF公共首部 类型: 4
通告数: 1
LSA通用首部 类型: 2
255.255.255.0
10.24.7.14
10.24.7.15
10.24.7.16

图 11.38 例 11.8 的解

例 11.9

在图 11.39 中，哪一个（或哪些）路由器要发送路由器链路 LSA？

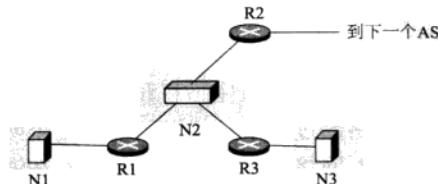


图 11.39 例 11.9 和例 11.10

解

所有的路由器都要发送路由器链路 LSA。

- a. R1 有两条链路，N1 和 N2。

- b. R2 有一个链路, N2。
- c. R3 有两条链路, N2 和 N3。

例 11.10

在图 11.39 中, 哪一个(或哪些)路由器要发送网络链路 LSA?

解

所有三个网络都必须通告网络链路:

- a. R1 为 N1 发送通告, 因为它是唯一连接在这个网络上的路由器, 因此它就是指定路由器。
- b. R1、R2 或 R3 这三者之一为 N2 发送通告, 这取决于哪一个被选为指定路由器。
- c. R3 为 N3 发送通告, 因为它是唯一连接在这个网络上的路由器, 因此它是个指定路由器。

汇总链路到网络 LSA

在一个区域内部, 路由器链路和网络链路都是通过洪泛的方法来通告有关路由器链路和网络链路的信息。但是, 路由器还必须知道它的区域以外的网络情况, 而区域边界路由器则能够提供这个信息。区域边界路由器是活跃在多个区域的路由器。它接收路由器链路和网络链路通告, 并为每一个区域创建一个路由表(我们稍后将会了解)。例如, 在图 11.40 中, 路由器 R1 是一个区域边界路由器。它有两个路由表, 一个由区域 1 使用, 另一个由区域 0 使用。R1 用洪泛方法在区域 1 中传播如何到达区域 0 中的网络的信息。同样地, R2 用洪泛方法在区域 2 中传播如何到达区域 0 中的网络的信息。

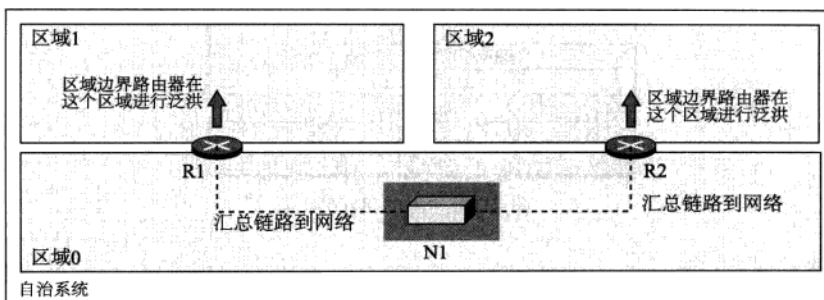


图 11.40 汇总链路到网络

区域边界路由器使用汇总链路到网络 LSA 来宣布在这个区域以外的其他网络的存在。汇总链路到网络通告非常简单。它包括网络掩码和对每一种服务类型的度量。请注意, 每一个通告只宣布一个网络。若网络超过一个, 就必须对每一个网络发出单独的通告。读者会问, 为什么只通告网络的掩码? 网络地址本身又是什么? 在链路状态通告的首部中宣布了发送通告的路由器的 IP 地址, 根据此信息以及掩码, 就可得出网络地址。这个通告的格式如图 11.41 所示。汇总链路到网络 LSA 的各字段如下:

- 网络掩码** 这个字段定义了网络掩码。
- TOS** 这个字段定义了服务类型。
- 度量** 这个字段定义了在 TOS 字段中定义的服务类型的度量。



图 11.41 汇总链路到网络 LSA

汇总链路到 AS 边界路由器 LSA

前面所讨论的那些通告可以让每一个路由器知道到达这个自治系统内的所有网络的代价。但这个自治系统以外的网络怎么办呢？如果在一个区域内的路由器要向这个自治系统外发送一个分组，它必须首先知道如何到达自治系统的一个边界路由器的路由，汇总链路到 AS 边界路由器通告则可以提供这种信息。区域边界路由器用洪泛方法传播此信息（见图 11.42）。这个分组被用来宣布到达 AS 边界路由器的路由。它的格式与前面的汇总链路一样。这个分组仅定义 AS 边界路由器所连接到的网络。如果一个报文能够到达这个网络，它就能够被 AS 边界路由器收到。这个分组的格式如图 11.43 所示。它的一些字段与汇总链路到网络通告报文的字段是一样的。

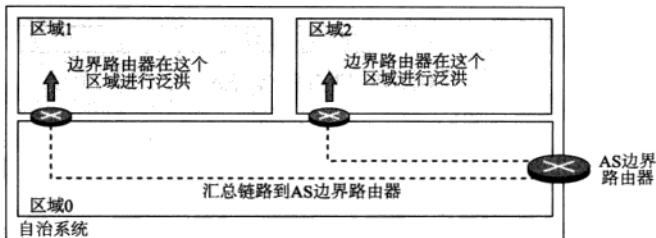


图 11.42 汇总链路到 AS 边界路由器

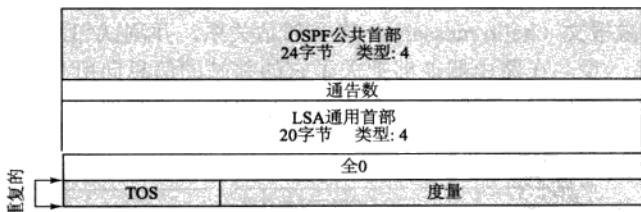


图 11.43 汇总链路到 AS 边界路由器 LSA

外部链路 LSA

虽然前面的通告让每一个路由器都知道了到达 AS 边界路由器的路由，但这些信息还是不够的。在自治系统内的路由器希望知道在自治系统外的哪一个网络是可用的，外部链路通告提供了这种信息。AS 边界路由器把到达这个自治系统外的每一个网络的代价，在自治系统内进行洪泛，它使用了域间路由选择协议产生的路由表。每一个通告宣布一个网络。如果网络数超过一个，就分别宣布。图 11.44 描述了一个外部链路。它被用来宣布在 AS 外

部的所有网络。这个 LSA 的格式与汇总链路到 AS 边界路由器 LSA 相似，但增加了两个字段。AS 边界路由器可以定义能够提供一条更好路由到达终点的转发路由器。这个分组还可以包括由其他协议（而不是 OSPF）使用的外部路由标记。这个分组的格式如图 11.45 所示。

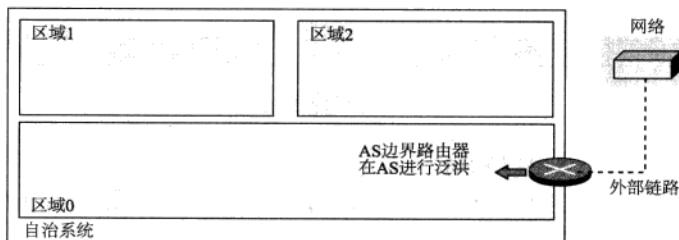


图 11.44 外部链路

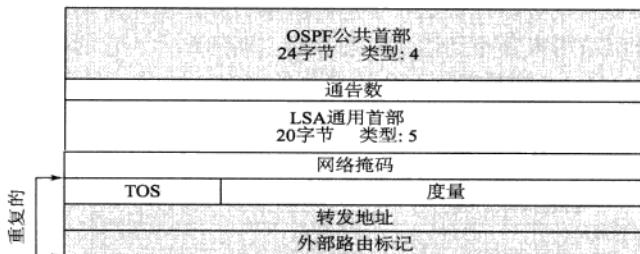


图 11.45 外部链路 LSA

11.6.7 其他分组

现在我们要讨论其他四种分组类型（见图 11.27）。它们并没有被当作 LSA 使用，但在 OSPF 的运行中却是很重要的。

问候报文

OSPF 使用问候报文（hello message）建立邻站关系，并测试邻站的可达性。这是链路状态路由选择的第一步。在路由器能够把关于它的邻站的信息向所有其他路由器进行洪泛之前，必须首先和它的邻站打招呼。这个路由器必须知道这些邻站是否都在工作，它们是否都是可达的（见图 11.46）。

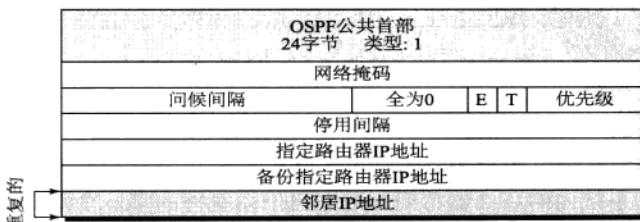


图 11.46 问候分组

- 网络掩码 这个 32 位字段定义了发送问候报文的网络的网络掩码。

- **问候间隔** 这个 16 位字段定义了问候报文和问候报文之间间隔的秒数。
- **E 标志** 这是 1 位标志。当它置 1 时，表示这个区域是残桩区域。
- **T 标志** 这是 1 位标志。当它置 1 时，表示这个路由器支持多种度量。
- **优先级** 这个字段定义了路由器的优先级。优先级用来选择指定路由器。在所有的邻站都宣布了它们的优先级后，具有最高优先级的路由器就被选择为指定路由器。具有第二高优先级的路由器则被选为备份指定路由器。若这个字段的值是 0，则表示这个路由器从来不想成为指定路由器或备份指定路由器。
- **停用间隔** 这个 32 位字段定义了在路由器认为某个邻站是停用的之前必须等候的时间，以秒为单位。
- **指定路由器 IP 地址** 这个 32 位字段是发送这个报文的网络的指定路由器的 IP 地址。
- **备份指定路由器 IP 地址** 这个 32 位字段是发送这个报文的网络的备份指定路由器的 IP 地址。
- **邻站 IP 地址** 这是一个可重复出现的 32 位字段，它定义了已经同意成为这个发送路由器的邻站。换言之，这是目前的所有邻站列表，说明发送路由器已经收到了从这些邻站发来的问候报文。

数据库描述报文

当路由器第一次或经过了一次故障后连接到系统上时，马上就需要有一个完整的链路状态数据库。它不能等收到来自所有其他路由器的所有的链路状态更新分组之后才去构造自己的数据库并计算路由表。因此，在路由器连接到系统之后，它就发送问候报文，向自己的邻站打招呼。若这些邻站是第一次收到这个路由器的信息，它们会发送数据库描述报文 (database description message)。数据库描述分组并不包含完整的数据库信息，它只给出了概要，即数据库中每一行的标题。新连接上的路由器检查这些标题，并找出哪些行的信息它还没有，然后再发送一个或多个链路状态请求报文，以便得到这个特定链路的完整信息。当两个路由器想交换数据库描述分组时，其中的一个起到主路由器的作用，而另一个则是从路由器。因为这个报文可能会太长了，所以数据库的内容可以分割为多个报文。数据库描述分组的格式如图 11.47 所示。它的一些字段如下：

- **E 标志** 若发送通告的路由器是自治边界路由器，则这个 1 位的标志就置为 1 (E 代表外部)。
- **B 标志** 若发送通告的路由器是区域边界路由器，则这个 1 位的标志就置为 1。
- **I 标志** 若该报文是第一个报文，则这个 1 位的初始化标志字段就置为 1。
- **M 标志** 若该报文不是最后一个报文，则这个 1 位的更多标志字段就置为 1。
- **M/S 标志** 这个 1 位字段是主/从位，它指出分组的来源：主 (M/S = 1) 或从 (M/S = 0)。

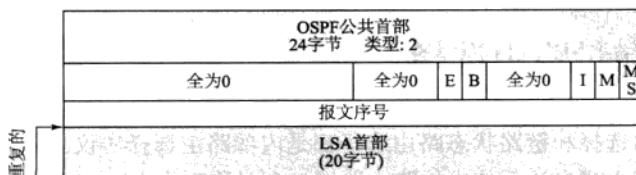


图 11.47 数据库描述分组

- **报文序号** 这个 32 位字段包含了报文的序号，用来使响应与请求相匹配。
- **LSA 首部** 这个 20 字节的字段在每一个 LSA 中使用。这个首部的格式已在链路状态更新报文一节中讨论过。首部中给出了每一条链路的概要，没有细节。它是可重复出现的字段，为链路状态数据库中的每一条链路重复一次。

链路状态请求分组

链路状态请求分组 (link state request packet) 的格式如图 11.48 所示。这是由需要一条或多条特定路由信息的路由器所发送出的分组。对它的回答则是链路状态更新分组。新接入的路由器在收到数据库描述分组后，可以使用它来请求关于某些路由的更多信息。这里出现的三个字段是 LSA 首部的一部分，我们已经讨论过了。三个字段为一组就是一个的 LSA 请求。如果需要一个以上的通告，就重复此组合。

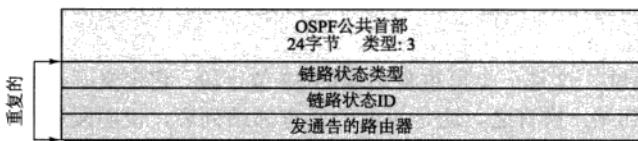


图 11.48 链路状态请求分组

链路状态确认分组

OSPF 强制路由器对所收到的每一个链路状态更新分组进行确认，这就使得路由选择更加可靠。链路状态确认分组 (link state acknowledgment packet) 的格式如图 11.49 所示。它具有公共的 OSPF 首部和通用的 LSA 首部。用这两个部分对分组进行确认是足够的。

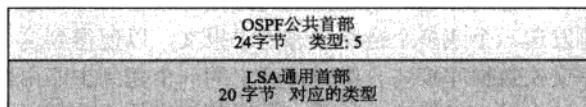


图 11.49 链路状态确认分组

11.6.8 封装

OSPF 分组被封装成 IP 数据报。这些数据报包括确认机制以实现流量控制和差错控制。它们不需要通过运输层协议来提供这些服务。

OSPF 分组被封装成 IP 数据报。

11.7 路径向量路由选择

距离向量路由选择和链路状态路由选择都是内部路由选择协议。它们可以在一个自治系统内部被用作域内或自治系统内的路由选择（有时称呼为 Intra-AS 路由选择）协议，但它们不能被用于自治系统之间。当运行区域变得很大时，这两种协议都会变得不可操作。

如果运行区域的跳数比较多，距离向量路由选择就会变得不稳定。而链路状态路由选择则需要大量的资源用于计算路由表。由于使用了洪泛方法，会产生很大的通信量。因此需要有第三种路由选择协议，我们称之为路径向量路由选择（path vector routing）。

路径向量路由选择是外部路由选择协议，并且已被证明在域间路由选择（有时称为 Inter-AS 路由选择）中是很有用的。在距离向量路由选择中，路由器拥有一个网络列表，列出了某一个 AS 中的能够到达的所有网络及其相应的代价（跳数）。在路径向量路由选择中，路由器也有一个网络的列表，列出了到达每个网络的路径（途经 AS 的列表）。换言之，距离向量路由选择的运行区域是单一的 AS，而路径向量路由选择的运行区域则是整个因特网。距离向量路由选择告诉我们到达每个网络的距离，而路径向量路由选择则告诉我们到达每个网络的路径。

例 11.11

距离向量路由选择和路径向量路由选择之间的区别可以被看成是国家地图与世界地图之间的区别。国家地图能告诉我们到达每个城市的路线，如果选择了某条特定的路线我们还能知道行程的距离。而世界地图则只能告诉我们每个国家有哪些城市，以及要到达某个城市需经过哪些国家。

11.7.1 可达性

为了能够向其他 AS 提供信息，每个 AS 必须至少有一个收集了其内部所有网络可达性信息的路径向量路由表。在这种情况下收集到的信息仅仅表示了哪个网络是存在的（在此 AS 中可达），这些网络通过它们的网络地址（CIDR 前缀）来标识。换言之，AS 需要有一个列表，列出它内部存在的所有网络。图 11.50 描绘了三个 AS。每个距离向量（外部路由器）建立了一个列表，列出该 AS 内部可达的网络。

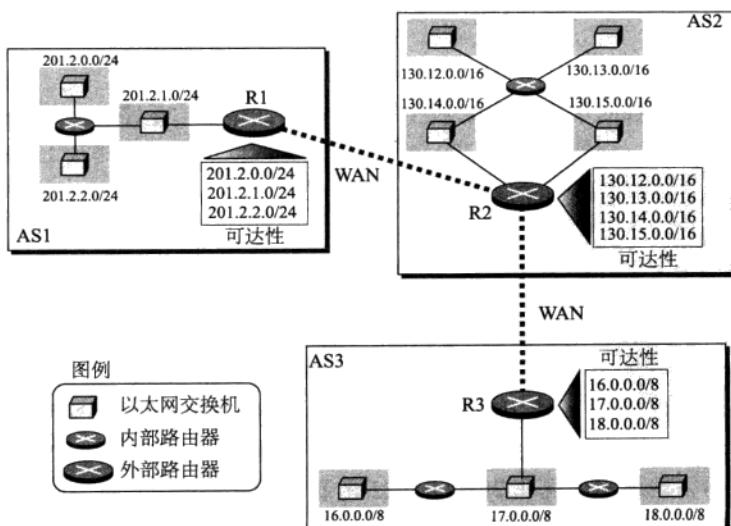


图 11.50 可达性

11.7.2 路由表

如果 AS 之间能够彼此共享它们的可达性列表，那么每个路由器就能够建立一张路径向量路由表。在图 11.50 中，AS1 中的路由器 R1 可以向路由器 R2 发送它的可达性列表。路由器 R2 把收到的信息合并到自己的可达性列表后，就能够将合并后的结果发送给 R1 和 R3。路由器 R3 也可以向 R2 发送它的可达性列表，这样做又加强了 R2 的可达性路由表，依此类推。图 11.51 所示为这三个路由器都更新过它们的路由表之后每个路由器的路由表。路由器 R1 知道如果有一个分组要到达网络 201.2.2.0/24，那么这个网络应该是在 AS1 内部的（在本地），但是如果一个分组要到达网络 130.14.0.0/16，那么这个分组应当从 AS1 转发到 AS2 才能到达它的目的网络。另一方面，如果路由器 R2 收到一个目的网络为 22.0.0.0/8 的分组，R2 就知道这个分组应当从 AS2 转发到 AS3 才能到达它的终点。我们可以将这种路由表与距离向量路由表进行对比，看看有什么不同。

R1		R2		R3	
网络	路径	网络	路径	网络	路径
201.2.0.0/24	AS1 (This AS)	201.2.0.0/24	AS2, AS1	201.2.0.0/24	AS3, AS2, AS1
201.2.1.0/24	AS1 (This AS)	201.2.1.0/24	AS2, AS1	201.2.1.0/24	AS3, AS2, AS1
201.2.2.0/24	AS1 (This AS)	201.2.2.0/24	AS2, AS1	201.2.2.0/24	AS3, AS2, AS1
130.12.0.0/16	AS1, AS2	130.12.0.0/16	AS2 (This AS)	130.12.0.0/16	AS3, AS2
130.13.0.0/16	AS1, AS2	130.13.0.0/16	AS2 (This AS)	130.13.0.0/16	AS3, AS2
130.14.0.0/16	AS1, AS2	130.14.0.0/16	AS2 (This AS)	130.14.0.0/16	AS3, AS2
130.15.0.0/16	AS1, AS2	130.15.0.0/16	AS2 (This AS)	130.15.0.0/16	AS3, AS2
16.0.0.0/8	AS1, AS2, AS3	16.0.0.0/8	AS2, AS3	16.0.0.0/8	AS3 (This AS)
17.0.0.0/8	AS1, AS2, AS3	17.0.0.0/8	AS2, AS3	17.0.0.0/8	AS3 (This AS)
18.0.0.0/8	AS1, AS2, AS3	18.0.0.0/8	AS2, AS3	18.0.0.0/8	AS3 (This AS)

图 11.51 三个自治系统的稳定的路由表

防止环路

距离向量路由选择的不稳定性和环路的产生可以在路径向量路由选择中避免。当路由器收到可达性信息后，它就检查看它的自治系统是否在到达终点的路径上。如果是，就说明会出现环路，于是就丢弃这个报文。

聚合

路径向量路由选择协议通常支持 CIDR 记法和地址聚合（如果可能）。这有助于使路径向量路由表更加简短，且路由器之间的信息交换更快。例如，图 11.51 中的路径向量路由表就可以通过聚合创建出更简短的路由表（图 11.52）。请注意，一个地址段中也可能包含了对应 AS 中没有的地址块。例如，地址段 201.2.0.0/22 中也包含了地址段 201.2.0.3/24，而后者不是 AS1 中任何网络的网络地址。但是，如果这个网络在其他 AS 中存在，那么它最终会成为这个路由表中的一部分。根据我们在第 6 章中讨论的最长前缀原则，这个网络地址将出现在 201.2.0.0/22 之前，并且先被搜索到，所以能够实现正确的路由选择。



图 11.52 聚合后的路由表

策略路由选择

策略路由选择 (policy routing) 可以很容易地通过路径向量路由选择来实现。当路由器收到一个报文，就检查其路径。如果路径中列出的一个自治系统与策略相违背，就忽略这条路径和这个终点。路由器不会在路由表中对这条路径进行更新，也不会把这个报文发送给它的邻站。

11.8 BGP

边界网关协议 (Border Gateway Protocol, BGP) 是一个使用路径向量路由选择的域间路由选择协议。它于 1989 年首次出现，目前已经有四个版本。

11.8.1 自治系统的类型

如我们以前讲过的，因特网被划分为多个分级的区域，称为自治系统 (AS)。例如，一个大企业管理着自己的网络，它对该网络有完全的控制权，就构成了一个自治系统。向本地用户提供服务的一个 ISP 也是一个自治系统。请注意，一个组织可能会由于地理上的分布、服务提供者 (ISP) 不同、甚或是由于本地存在着某些障碍而选择拥有多个 AS。

我们可以把自治系统划分为三类：残桩 AS、多归属 AS 和转接 AS。

残桩 AS

残桩 AS 只有一个连接可到达另一个 AS。残桩 AS 的域间通信量要么是由这个 AS 产生的，要么就是将在这个 AS 中终止的。残桩 AS 中的主机可以向其他 AS 发送数据，也可以接收来自其他 AS 中的主机的数据。但是，数据通信量不可能途经一个残桩 AS。残桩 AS 要么是一个信源，要么是一个信宿。残桩 AS 的一个典型例子就是一家小公司或一个本地 ISP。

多归属 AS

多归属 AS 有多条连接到达其他多个 AS，但它仍然只能是数据通信量的信源或者信宿。它可以接收来自多个 AS 的数据通信量，也可以向多个 AS 发送数据通信量，但是没有穿越的通信量会经过它。它不允许从一个 AS 到另一个 AS 的数据通信量在途中经过它。多归属 AS 的一个典型例子就是一家大公司，它可连接到多个地区 AS 或国家级 AS，但不允许穿越的通信量。

转接 AS

转接 AS 是一种允许穿越通信量的多归属 AS。转接 AS 的一个典型例子就是国家的和

国际的 ISP (因特网主干网)。

CIDR

BGP 使用无分类域间路由选择编址。换言之，BGP 使用前缀（第 5 章讨论过）来定义一个目的地址。这个地址和位数（前缀长度）被用在更新报文中。

11.8.2 路径属性

在前面的例子中，我们讨论了到达目的网络的路径。这条路径用自治系统的列表来表示，但实际上它是用路径的属性列表来表示的。每一个属性给出了关于该路径的一些信息。在应用策略时，属性列表可帮助接收信息的路由器做出更好的决定。

属性可分为两大类：熟知的和可选的。熟知属性（well-known attribute）是每一个 BGP 路由器必须掌握的。可选属性（optional attribute）则不是每一个路由器都必须知道的。

熟知属性本身又可划分为两类：强制的和自选的。熟知强制属性是在路由的描述中必须出现的。熟知自选属性是每一个路由器必须知道的，但不一定要包括在每一个更新报文中。有一个熟知强制属性是 ORIGIN，它定义了路由选择信息的来源（RIP、OSPF 等等）。另一个熟知强制属性是 AS_PATH，它定义了一个自治系统列表，通过这些自治系统可以到达终点。还有一个熟知强制属性是 NEXT_HOP，它定义了分组必须发送到的下一个路由器。

可选属性也可划分为两类：传递的和非传递的。可选传递属性是没有实现这个属性的路由器必须传递给下一个路由器的。可选非传递属性则说明如果接收路由器没有实现这个属性，它就必须丢弃这个属性。

11.8.3 BGP 会话

两个路由器之间使用 BGP 互相交换路由选择信息的行为发生在一次会话中。会话就是建立在两个 BGP 路由器之间的一个连接，且该连接仅用于交换路由选择信息。为了建立一种可靠的环境，**BGP 使用 TCP 的服务**。换言之，作为一个应用程序，在 BGP 级的会话是在 TCP 级的连接上进行的。但是，为 BGP 建立的 TCP 连接和为其他应用程序建立的 TCP 连接有一点点不同。如果 TCP 连接是为 BGP 而产生的，那么它就能持续很长的时间，直到发生了某些异常情况。因此，BGP 会话有时也称为半永久的连接。

11.8.4 外部 BGP 和内部 BGP

与 RIP 的区别 ==
跳数区别？
和 TCP 与 UDP

如果我们希望再精确一些，那么 BGP 可以有两种类型的会话：外部 BGP (E-BGP) 会话和内部 BGP (I-BGP) 会话。E-BGP 会话用于两个不同的自治系统内的发言人结点之间互相交换信息。与此相反，I-BGP 会话用于一个自治系统内的两个路由器之间互相交换路由选择信息。图 11.53 描绘了这个概念。

在 AS1 和 AS2 之间建立的会话是 E-BGP 会话。这两个发言人路由器交换的信息是它们知道的关于因特网中的一些网络的信息。但是，这两个路由器需要在各自的自治系统中从其他路由器收集信息。这就要使用 I-BGP 会话。

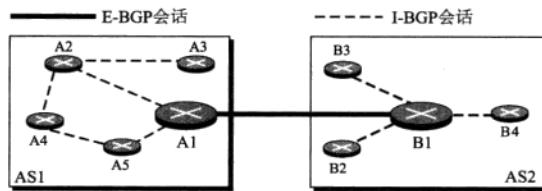


图 11.53 内部和外部 BGP 会话

11.8.5 分组的类型

BGP 使用四种不同类型的报文：打开、更新、保活和通知（见图 11.54）。



图 11.54 BGP 报文的类型

11.8.6 分组格式

所有的 BGP 分组共享相同的公共首部。在学习不同类型的分组之前，我们先来讨论公共首部（见图 11.55）。这个首部的各字段如下：

- **标记** 这个 16 字节的标记字段保留给鉴别用。
- **长度** 这个 2 字节的字段定义了包括首部在内的报文总长度。
- **类型** 这个 1 字节的字段定义了分组的类型。如以前讲过的，我们共有四种类型，用数值 1 至 4 定义这些类型。

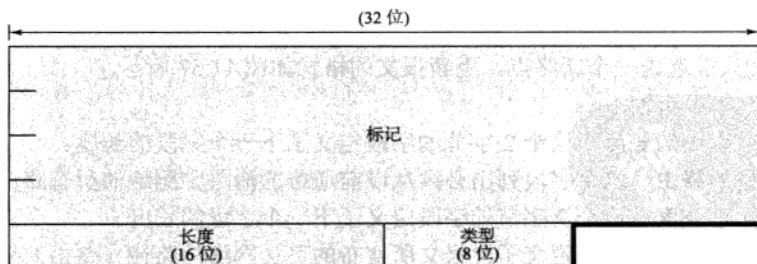


图 11.55 BGP 的分组首部

打开报文

为了建立邻站关系，运行 BGP 的路由器需要打开与邻站的 TCP 连接，因而要发送打开报文（open message）。若对方接受这种邻站关系，就用保活报文（keepalive message）来响应，表示在这两个路由器之间已经建立了关系。图 11.56 是打开报文格式的描述。

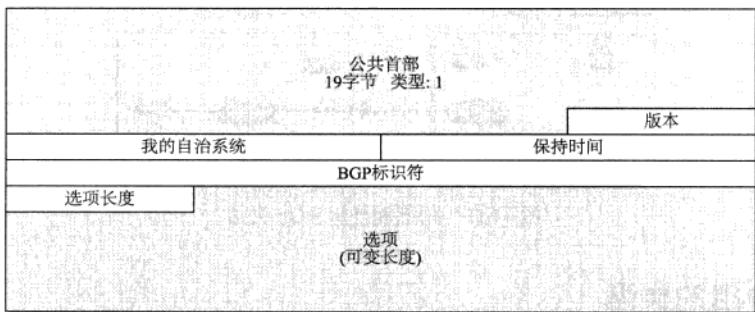


图 11.56 打开报文

打开报文的各字段如下：

- **版本** 这个 1 字节的字段定义了 BGP 的版本。当前的版本是 4。
- **我的自治系统** 这个 2 字节的字段定义了自治系统号。
- **保持时间** 这个 2 字节的字段定义了一方从另一方收到保活报文或更新报文之前所经过的最大秒数。若路由器在保持时间内没有收到任何一个此类报文，就认为对方是不工作的。
- **BGP 标识符** 这个 4 字节的字段定义了发送打开报文的路由器。为此，这个路由器通常要使用它的一个 IP 地址作为 BGP 标识符（因为 IP 地址是唯一的）。
- **选项参数长度** 打开报文还可以包含某些选项参数。如果包含了，则这个 1 字节的字段定义了选项参数总长度。若没有选项参数，则这个字段的值为 0。
- **选项参数** 若选项参数长度的值不是零，则表示有某些选项参数。每一个选项参数本身又有两个子字段：参数长度和参数值。目前为止已定义的唯一的选项参数是鉴别。

更新报文

更新报文是 BGP 协议的核心。它被路由器用来撤销以前曾通告过的终点，或宣布到一个新终点的路由，或者两者皆有。请注意，BGP 可以撤销多个以前曾通告过的终点，但一个的更新报文只能通告一个新终点。更新报文的格式如图 11.57 所示。

更新报文的各字段如下：

- **不可行路由的长度** 这个 2 字节的字段定义了下一个字段的长度。
- **被撤销的路由** 这个字段列出必须从以前通告的清单中删除的所有路由。
- **路径属性长度** 这个 2 字节的字段定义了下一个字段的长度。
- **路径属性** 这个字段定义了此报文所宣布的可达网络的路径（路由）属性。
- **网络层可达性信息（NLRI）** 这个字段定义了此报文真正通告的网络。它有一个长度字段和一个 IP 地址前缀。长度定义前缀的位数。前缀定义的是网络地址的共同部分。例如，若这个网络是 153.18.7.0/24，则网络前缀是 24，而前缀是 153.18.7。BGP4 支持无分类编址和 CIDR。

BGP 支持无分类编址和 CIDR。

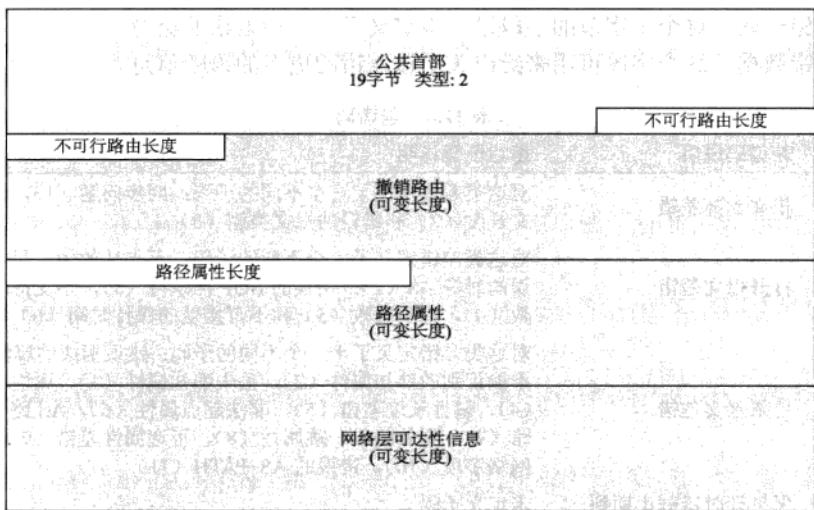


图 11.57 更新报文

保活报文

运行 BGP 协议的路由器（用 BGP 的术语称为对等路由器）定期地互相交换保活报文（在其保持时间截止之前），用来告诉对方自己是工作的。保活报文只包括图 11.58 所示的公共首部。

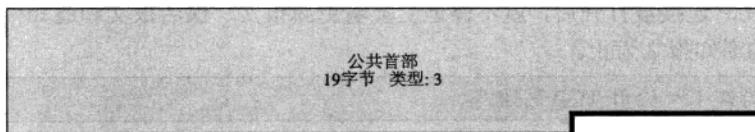


图 11.58 保活报文

通知报文

当检测出差错状态或路由器打算关闭连接时，路由器就发送通知报文。这个报文的格式如图 11.59 所示。组成通知报文的各字段如下：

- 差错码** 这个 1 字节的字段定义了差错的种类，见表 11.6。

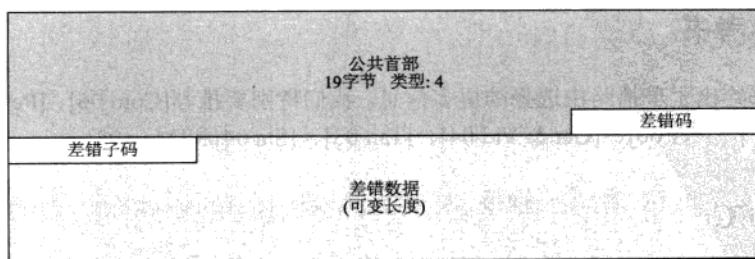


图 11.59 通知报文

- **差错子码** 这个1字节的字段进一步定义了每一种差错的类型。
- **差错数据** 这个字段可用来给出关于该差错的更多的诊断信息。

表 11.6 差错码

差错码	差错码说明	差错子码说明
1	报文首部差错	对这类差错定义了三个不同的子码：同步问题（1），错误的报文长度（2）和错误的报文类型（3）
2	打开报文差错	对这类差错定义了六个不同的子码：不支持的版本号（1），错误的对等 AS（2），错误的 BGP 标识符（3），不支持的可选参数（4），鉴别失败（5）和不可接受的保持时间（6）
3	更新报文差错	对这类差错定义了十一个不同的子码：错误形成的属性表（1），不能识别的熟知属性（2），丢失熟知属性（3），属性标志差错（4），属性长度差错（5），非法起点属性（6），AS 路由选择环路（7），无效的下一跳属性（8），可选属性差错（9），无效的网络字段（10），错误的 AS_PATH（11）
4	保持计时器截止期到	未定义子码
5	有限状态机差错	定义处理过程中的差错。未定义子码
6	停止	未定义子码

11.8.7 封装

BGP 报文封装成 TCP 报文段，并使用熟知端口 179。这表示不需要使用差错控制和流量控制。在 TCP 连接被打开后，就不停地交换着更新报文、保活报文和通知报文，直到发出停止类型的通知报文为止。

BGP 在端口 179 使用 TCP 的服务。

11.9 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

11.9.1 参考书

有几本书给出了单播路由选择的更多信息。我们特别要推荐[Com 06]、[Pet & Dav 03]、[Kur & Ros 08]、[Per 00]、[Gar & Vid 04]、[Tan 03]、[Sta 04]和[Moy 98]。

11.9.2 RFC

有一些 RFC 与我们本章所讨论的内容相关。RIP 在 RFC 1058 和 RFC 2453 中讨论。OSPF 在 RFC 1583 和 RFC 2328 中讨论。BGP 在 RFC 1654、RFC 1771、RFC 1997、RFC 2439、

RFC 2918 和 RFC 3392 中讨论。

11.10 重要术语

区域	保活报文
区域边界路由器	链路状态请求分组
自治系统（AS）	链路状态路由选择
自治系统边界路由器	链路状态更新分组
主干路由器	度量
Bellman-Ford 算法	通知报文
边界网关协议（BGP）	打开报文
代价	开放最短路径优先（OSPF）
计数到无穷大	可选属性
Dijkstra 算法	路径向量路由选择
距离向量路由选择	定期计时器
截止期计时器	点对点传输
洪泛	毒性逆转
无用信息收集计时器	路由信息协议（RIP）
问候间隔	分割范围
问候报文	残桩链路
跳数	穿越链路
域间路由选择	更新报文
域内路由选择	虚拟链路
熟知属性	

RIP 使用 UDP 520 端口
 OSPF 直接使用 IP
 BGP 使用 TCP 179 端口

11.11 本章小结

- 度量是对分组途经的网络通路指派的一个代价。路由器通过咨询其路由表来为分组决定一条最佳路径。
- 自治系统（AS）是在一个管理机构管辖之下的一组网络和路由器。RIP 和 OSPF 都是流行的域内路由选择协议（也称为内部路由选择协议），用来在自治系统内部更新路由表。RIP 基于距离向量路由选择，即每一个路由器定期与它的邻站共享关于整个自治系统的知识。OSPF 则把 AS 划分为一些区域，区域的定义是一些网络、主机和路由器的集合。OSPF 基于链路状态路由选择，即每一个路由器向区域内的其他所有路由器发送其邻站的状态。
- BGP 是用来更新路由表的域间路由选择协议（也称为外部路由选择协议）。BGP 所基于的路由选择方法称为路径向量路由选择。在这个协议中，分组必须经过的一些

自治系统应当显式列出。路径向量路由选择没有距离向量路由选择的不稳定性，也没有环路问题。共有四种类型的 BGP 报文：打开、更新、保活以及通知。

11.12 实践安排

11.12.1 习题

- 在 RIP 中，为什么截止期计时器的值是定期计时器的值的 6 倍？
- 跳数限制如何缓解了 RIP 的问题？
- 试对照和比较距离向量路由选择和链路状态路由选择。
- 为什么 OSPF 报文传播得比 RIP 报文快？
- 只通告一个网络的 RIP 报文的长度是多少？通告 N 个网络的 RIP 报文的长度是多少？试设计一个公式，给出通告的网络数与 RIP 报文长度之间的关系。
- 一个运行 RIP 的路由器的路由表有 20 个表项。试问处理这个表共需要多少个定期计时器，多少个截止期计时器以及多少个无用信息收集计时器？
- 一个运行 RIP 的路由器其路由表如表 11.7 所示。

表 11.7 习题 7 的路由表

终点	代价	下一个路由器
Net1	4	B
Net2	2	C
Net3	1	F
Net4	5	G

试给出这个路由器发送的 RIP 响应报文。

- 习题 7 中的路由器收到一个来自路由器 C 的 RIP 报文，该报文具有如下的四个记录：

(Net1, 2), (Net2, 1), (Net3, 3), (Net4, 7)

试给出更新后的路由表。

- 在通告了 N 个网络的 RIP 报文中中有多少个字节是空的？

- 参考图 11.26，回答以下问题：

- 给出路由器 A 的链路状态更新/路由器链路通告。
- 给出路由器 D 的链路状态更新/路由器链路通告。
- 给出路由器 E 的链路状态更新/路由器链路通告。
- 网络 N2 的链路状态更新/网络链路通告。
- 网络 N4 的链路状态更新/网络链路通告。
- 网络 N5 的链路状态更新/网络链路通告。

- 在图 11.26 中：

- 假定网络 N1 的指定路由器是路由器 A。试给出这个网络的链路状态更新/网络

- 链路通告。
- 假定网络 N3 的指定路由器是路由器 D。试给出这个网络的链路状态更新/网络链路通告。
12. 为图 11.26 中的网络和路由器指派 IP 地址，并回答以下问题：
- 给出路由器 C 发送的 OSPF 问候报文。
 - 给出路由器 C 发送的 OSPF 数据库描述报文。
 - 给出路由器 C 发送的 OSPF 链路状态请求报文。
13. 试给出具有以下规约的自治系统：
- 共有八个网络 (N1~N8)。
 - 共有八个路由器 (R1~R8)。
 - N1、N2、N3、N4、N5 和 N6 是以太网网络。
 - N7 和 N8 是点对点网络。
 - R1 连接 N1 和 N2。
 - R2 连接 N1 和 N7。
 - R3 连接 N2 和 N8。
 - R4 连接 N7 和 N6。
 - R5 连接 N6 和 N3。
 - R6 连接 N6 和 N4。
 - R7 连接 N6 和 N5。
 - R8 连接 N8 和 N5。

请用图形方式从 OSPF 的角度表示这个自治系统。哪些网络是穿越网络？哪些是残桩网络？

14. 在图 11.50 中：
- 试给出路由器 R1 的 BGP 打开报文。
 - 试给出路由器 R1 的 BGP 更新报文。
 - 试给出路由器 R1 的 BGP 保活报文。
 - 试给出路由器 R1 的 BGP 通知报文。
15. 在图 11.5 中，假设路由器 A 和路由器 B 之间的链路出现故障（中断了）。给出图 11.7 所示的路由表的变化情况。
16. 在图 11.5 中，假设路由器 C 和路由器 D 之间的链路出现故障（中断了）。给出图 11.7 所示的路由表的变化情况。
17. 用 Bellman-Ford 算法（表 11.1）求出图 11.60 所示图中所有结点的最短距离。
18. 用 Dijkstra 算法（表 11.3）求出图 11.61 所示图中所有结点的最短路径。

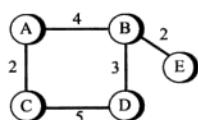


图 11.60 习题 17

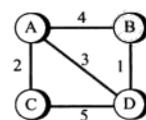


图 11.61 习题 18

19. 试求出图 11.19 中结点 B 的最短路径树。
20. 试求出图 11.19 中结点 E 的最短路径树。
21. 试求出图 11.19 中结点 G 的最短路径树。

11.12.2 研究活动

22. 在 BGP 之前，有一个协议称为 EGP。试找出关于这个协议的一些信息以及为什么这个协议不能生存下来的原因。
23. 在 UNIX 中，有一些程序使用共同的名字 `daemon`。试找出能够处理路由选择协议的 `daemon`。
24. 如果你可以使用 UNIX 系统，试找出有关 `routed` 程序的一些信息。这个程序怎样帮助你跟踪 RIP 交换的报文？`routed` 程序是否还支持本章中讨论的其他路由选择协议？
25. 如果你可以使用 UNIX 系统，试找出有关 `gated` 程序的一些信息。`gated` 程序支持了本章讨论过的哪些路由选择协议？
26. 有一个路由选择协议称为 HELLO，它没有在本章中讨论。试找出有关这个协议的一些信息。

第 12 章 多播和多播路由选择协议

本章我们要讨论多播和多播路由选择协议。虽然人们对多播应用的需求与日俱增，但是正如我们将要看到的，多播路由选择比第 11 章讨论的单播路由选择要困难得多，多播路由器要负责给相应组的所有成员都发送一个多播分组的副本。

目标

本章有以下几个目标：

- 对照比较单播、多播和广播通信。
- 定义 IPv4 的多播地址空间，并说明该空间如何被划分成若干个地址块。
- 讨论 IGMP 协议，它负责收集一个网络中的组成员信息。
- 讨论多播路由选择协议以及这些协议根据不同的最短路径树的创建方式被划分为两大类所隐含的总体思想。
- 讨论多播链路状态路由选择的一般概念，以及它在因特网上的具体实现：一个称为 MOSPF 的协议。
- 讨论多播距离向量路由选择的一般概念，以及它在因特网上的具体实现：一个称为 DVMRP 的协议。
- 讨论核心基干树协议（CBT）并简单介绍两个独立的多播协议 PIM-DM 和 PIM-SM。
- 讨论多播主干网（MBONE）并说明当多播报文需要通过一个不含多播路由器的区域时应当如何创建一个隧道。

12.1 引言

从本书这一部分的前几章内容中，我们已经了解到路由器在转发数据报时通常要根据数据报的目的地址前缀，它指明了目的主机连接到哪个网络上。当然，地址聚合机制也可以把多个数据报合并起来交付给一个 ISP（可以被认为是把几个网络聚合成一个大网络），然后再由这个 ISP 将数据报分别交付到它们各自的目的网络，但是，其中的基本原理并没有什么不同。聚合只不过是减少了前缀的长度，但分别交付时又会增加前缀的长度。

理解了以上这些转发原理，现在就让我们来定义单播、多播和广播。让我们结合因特网来弄清楚这几个术语。

12.1.1 单播

在单播通信中，只有一个源点网络和一个终点网络。源点网络和终点网络的关系是一对一的。数据报途经的每一个路由器都要将这个分组仅从一个接口转发出去。图 12.1 描绘了一个小型的互联网，其中有一个单播分组需要从源计算机交付到一台连接在网络 N6 上的目的计算机。路由器 R1 负责将这个分组仅通过接口 3 转发出去，而路由器 R4 负责将这个分组仅通过接口 2 转发出去。当分组到达 N6 后，就由该网络负责将其交付给目的主机，可以通过广播传送给网络中的所有主机，也可以由智能以太网交换机仅交付给它的目的主机。

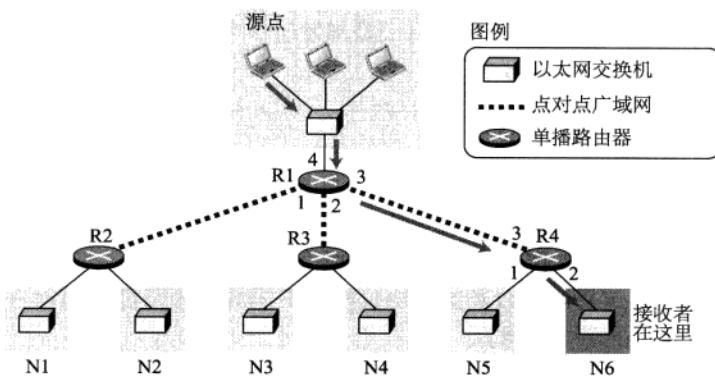


图 12.1 单播

正如我们在第 11 章中看到的，在单播时，有一张基于最佳路径的路由表，它为每个数据报定义了一个唯一的输出端口。

在单播通信中，路由器仅从它的一个接口转发收到的分组。

12.1.2 多播

在多播通信中，有一个源点和一组终点。这是一对多的关系。在这种类型的通信中，源地址是一个单播地址，而目的地址则是一个组地址，在这个组中包含了一个或多个目的网络，且在这些目的网络中至少含有一个有兴趣接收该多播数据报的组成员。组地址定义了组的成员。图 12.2 描绘的是与图 12.1 一样的小型互联网，但是路由器都被换为多播路由器了（或者是前图中的路由器被设置为既能单播也能多播）。

在多播通信中，多播路由器有可能需要将同一个数据报的多个副本通过多个接口发送出去。在图 12.2 中，路由器 R1 需要通过接口 2 和 3 发送该数据报。同样地，路由器 R4 也需要从两个接口发送该数据报。但是，路由器 R3 知道通过接口 2 到达的区域内没有属于这个组的成员，所以它只通过接口 1 发送该数据报。

在多播通信中，路由器可以通过它的多个接口转发收到的分组。

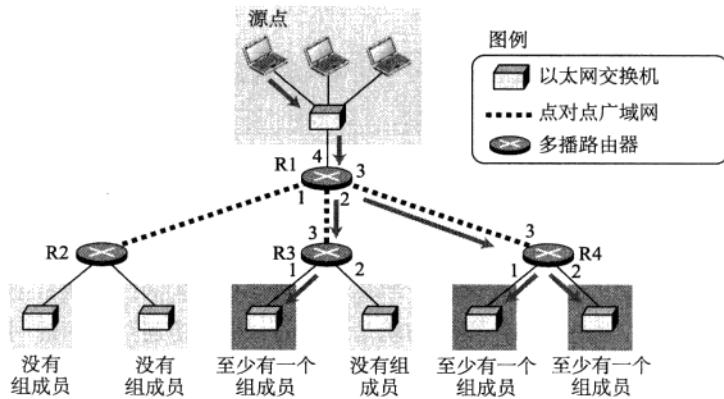


图 12.2 多播

多播和多个单播的比较

我们还需要区分多播和多个单播。图 12.3 描绘了这两个概念。

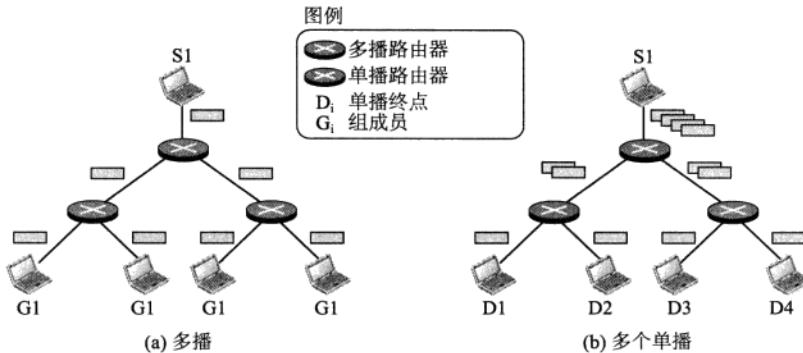


图 12.3 多播和多个单播

多播是由源点发送单个分组，然后一路上由各个路由器复制这个分组。所有分组副本的目的地址都是一样的。请注意，在两个路由器之间只有一个分组副本在传送。

在多个单播中，从源点开始就发出多个分组。例如，如果有四个终点，那么源点就发出四个分组，且每个分组具有不同的单播终点。请注意，在两个路由器之间可能会有多个副本在传送。例如，有人要向一组人发送一份电子邮件报文时，就是使用了多个单播。电子邮件软件把报文复制多份，给每一份写入不同的目的地址，然后逐个发送。

用单播对多播进行仿真

你可能会奇怪，既然可以用单播来模拟多播，我们为什么还需要一个单独的多播机制。这样做有几个理由，其中有两个理由是很明显的：

1. 多播比多个单播更加有效。在图 12.3 中，我们可看出为什么多播所需的带宽要小于多个单播。在多个单播中，有些链路必须要处理多个副本。
2. 在多个单播中，源点在产生这些分组时彼此之间会存在时延。如果有 1000 个终点，那么第一个分组和最后一个分组之间的时延可能就是不可接受的。在多播中，因为源点仅

产生一个分组，所以不存在这种时延。

用多个单播来模拟多播是低效率的，并且可能会因此而产生很大的时延，特别是对于很大的多播组。

多播应用

目前多播有着多种多样的应用，如访问分布式数据库、信息传播、电视会议以及远程学习。

访问分布式数据库 如今的大多数大型数据库都是分布式的。也就是说信息被存储在多个场所，而且通常在这些信息产生之初就是这样的。需要访问数据库的用户并不知道信息的所在地。用户的请求以多播方式传送到所有的数据库场所，而存放有这个信息的场所就给出响应。

信息传播 一些公司常常需要向其顾客发送信息。如果这个信息对每一个顾客都是相同的，那么就可以使用多播。使用这种方法时，公司只需发送一个报文就能让许多顾客都收到。例如，可以发送一个软件的更新给所有购买了这个特定软件包的人。

传播新闻 新闻也可以用类似的方法很方便地通过多播进行传播。一个报文可以发送给对这个特殊话题感兴趣的人群。例如，高中篮球锦标赛的冠军统计数据可以发送给多家报纸的体育编辑。

电视会议 电视会议涉及到了多播。所有参加电视会议的人员都需要在同一时间接收到同一信息。为此，可以形成临时性的或永久性的组。例如，在每星期一上午举行会议的某个工程组可以是一个永久性的组，而筹划节日晚会的组可以是一个临时性的组。

远程学习 远程学习是一个方兴未艾的多播领域。某个特定组的所有学生可以接收某一位教授的讲课。这对到校园教室来听课有困难的学生特别方便。

12.1.3 广播

在广播通信中，源点和终点的关系是一对所有。源点只有一个，但其他所有的主机都是终点。因特网明确地不支持广播，因为这会产生非常巨大的通信量，同时它所需的带宽也是非常巨大的。如果有一个人要向连接在因特网上的每个人发送消息，那么可以想象这将产生多巨大的通信量。

12.2 多播地址

多播地址是参加了多播组的一群主机的目的地址。用多播地址作为目的地址的分组能够被该组的所有成员都收到，除非接收方有某种过滤限制。

12.2.1 IPv4 中的多播地址

虽然数据链路层和网络层都有多播地址，但在本章我们只讨论网络层的多播地址，特

别是在 IPv4 中使用的多播地址。在 IPv6 中使用的多播地址我们将在第 26 章中讨论。在使用分类编址的情况下，多播地址占据了 D 类唯一的地址块。在无分类编址中也将相同的地址块用于多播。换言之，指派给多播的地址块是 224.0.0.0/4，同时也意味着这个地址块有 $2^{28} = 268435456$ 个地址（从 224.0.0.0 到 239.255.255.255）。根据 RFC 3171，这个有时被称为多播地址空间的大地址块被划分为若干个较小的地址段，如表 12.1 所示，它们将来或许还会有所变化。不过，我们无法为每一个分配的地址段指派一个 CIDR（斜线记法），因为主地址块在划分为小地址块（或子地址块）时并没有遵守我们在第 5 章中定义的规则。将来，这些地址段有可能会进一步划分为更小的地址段以遵守规则。

表 12.1 多播地址段

CIDR	地址段	分配
224.0.0.0/24	224.0.0.0 → 224.0.0.255	本地网络控制地址块
224.0.1.0/24	224.0.1.0 → 224.0.1.255	网际互连控制地址块
	224.0.2.0 → 224.0.255.255	AD HOC 地址块
224.1.0.0/16	224.1.0.0 → 224.1.255.255	流多播组地址块
224.2.0.0/16	224.2.0.0 → 224.2.255.255	SDP/SAP 地址块
	224.3.0.0 → 231.255.255.255	保留
232.0.0.0/8	232.0.0.0 → 232.255.255.255	特定源多播 (SSM)
233.0.0.0/8	233.0.0.0 → 233.255.255.255	GLOP 地址块
	234.0.0.0 → 238.255.255.255	保留
239.0.0.0/8	239.0.0.0 → 239.255.255.255	管理范围的地址块

本地网络控制地址块

第一个地址块 224.0.0.1/24 称为本地网络控制地址块 (Local Network Control Block)。这个地址块中的地址是由协议控制通信量来使用的。换言之，它们并不用于普通的多播通信。某些多播的或与多播相关的协议需要使用这些地址。目的地址落在这个范围内的 IP 分组必须将自己的 TTL 值置为 1，也就是说不允许路由器转发这些分组。这些分组被拦在一个网络中，就好像是这个网络私有的，这也意味着两个或多个网络可以同时使用相同的地址。表 12.2 所示为其中某些地址的分配情况。

网际互连控制地址块

地址块 224.0.1.0/24 称为网际互连控制地址块 (Internetwork Control Block)。这个地址块中的地址也是由协议控制通信量使用的，但是以这些地址为目的地址的 IP 分组能够被路由器转发，因而可以在整个因特网内畅通无阻。例如，地址 224.0.1.1 被 NTP 协议使用。

表 12.2 本地网络控制地址块中的一些地址

地址	分配
224.0.0.0	基地址 (保留)
224.0.0.1	此网络中的所有系统 (主机或路由器)
224.0.0.2	此网络中的所有路由器

续表

地址	分配
224.0.0.4	DMVRP 路由器
224.0.0.5	OSPF 路由器
224.0.0.7	ST (流) 路由器
224.0.0.8	ST (流) 主机
224.0.0.9	RIPv2 路由器
224.0.0.10	IGRP 路由器
224.0.0.11	移动代理
224.0.0.12	DHCP 服务器
224.0.0.13	PIM 路由器
224.0.0.14	RSVP 封装
224.0.0.15	CBT 路由器
224.0.0.22	IGMPv3

AD-HOC 地址块

从 224.0.2.0 到 224.0.255.255 的地址块被 IANA 称为专用(AD-HOC)地址块。传统上，这个地址块被分配给那些不适合于前面讨论两种地址块的应用。例如，地址块 224.0.18.0/24 被分配给了 Dow Jones。请注意，这个地址块的范围有两个与众不同的特点。第一，它的末地址是 224.0.255.255。第二，整个地址块无法用 CIDR 记法来表示（这个地址块的分配没有遵守我们在第 5 章中讨论过的无分类编址原则）。

流多播组地址块

地址块 224.1.0.0/16 称为流多播组地址块 (Stream Multicast Group Block)，它是为流媒体而分配的地址块。

SAP/SDP 地址块

地址块 224.2.0.0/16 用于会话宣布协议 (Session Announcement Protocol) 和会话目录协议 (Session Directory Protocol) (RFC 2974)。

SSM 地址块

地址块 232.0.0.0/8 用于特定源多播 (Source Specific Multicasting, SSM)。我们将在本章稍后介绍 IGMPv3 时讨论 SSM。

GLOP 地址块

地址块 233.0.0.0/8 称为 GLOP 地址块 (它既不是首字母缩写，也不是简写)。这个地址块定义的是可以在一个自治系统 (AS) 内使用的全球分配的地址段。正如我们在第 11 章中所了解的，每个自治系统都分配有一个 16 位的编号。在这个地址块的中间两个八位组上插入 AS 号，就可以产生一个具有 256 个多播地址的地址段 (从 233.x.y.0 到 233.x.y.255)，其中 x.y 是 AS 号。

管理范围的地址块

地址块 239.0.0.0/8 称为管理范围的地址块 (Administratively Scoped Block)。这个地址

块中的地址被用于因特网中的某个特定区域内。目的地址属于这个地址块的分组不应离开该区域。换言之，这个地址块中的地址被限制在一个组织内。

netstat 实用程序

netstat 实用程序可查找出某个接口支持的多播地址。

例 12.1

我们使用的 netstat 有三个选项：-n，-r，-a。选项-n 要求给出 IP 地址的版本号。选项-r 要求给出路由表，而选项-a 则要求给出所有的地址（单播和多播）。请注意，我们仅显示了与我们的讨论相关的字段。

\$ netstat -nra

Kernel IP routing table

Destination	Gateway	Mask	Flags	Iface
153.18.16.0	0.0.0.0	255.255.240.0	U	eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	10
224.0.0.0	0.0.0.0	224.0.0.0	U	eth0
0.0.0.0	153.18.31	0.0.0.0	UG	eth0

请注意，多播地址用粗体显示。多播地址在 224.0.0.0~239.255.255.255 之间的任何分组都被进行掩码操作，并交付到以太网接口。

12.2.2 选择多播地址

要选择出分配给某个组的多播地址并不是一个简单的任务。应当根据应用的类型来选择合适的地址。下面让我们来讨论其中的几种情况。

有限的组

系统管理员可以利用 AS 号 (x.y) 并选择在 233.x.y.0~233.x.y.255 之间的一个还没有被其他组使用的地址。例如，假设某位大学教授希望创建一个组用于她和学生之间的通信。如果该大学所属的 AS 号是 23452，可写为(91.156)₂₅₆，那么它就有这样一个具有 256 个地址的地址段：233.91.156.0~233.91.156.255。大学的系统管理员可以授权她使用该地址段中一个尚未被别人使用的地址，例如 233.91.156.47。这个地址就成为她用来给学生发送多播分组的组地址。但是这些分组不能超出该大学 AS 的管辖范围。

更大的组

如果这个组要扩展到一个 AS 管辖范围之外，那么前面的方法就不起作用了。这样的组需要从 SSM 地址块 (232.0.0.0/8) 中选择一个地址。这个地址块中的地址不需要经过允许就可以使用，因为在特定源多播中，分组的路由选择要根据分组的组地址和源地址，而它们的组合是唯一的。

12.2.3 数据链路层多播分组的交付

由于 IP 分组使用的是 IP 多播地址，ARP 协议无法找出相应的 MAC（物理）地址，但

是在数据链路层转发这个分组时需要用到 MAC 地址。下面该怎样办则取决于底层的数据链路层是否支持物理多播地址。

支持多播的网络

绝大多数局域网都支持物理多播地址。以太网就是其中之一。以太网地址（MAC 地址）的长度是六个八位组（48 位）。如果一个以太网地址的前 25 位是 00000001 00000000 01011110 0，就说明这是一个 TCP/IP 协议的物理多播地址。剩下的 23 位可用来定义组。要将一个 IP 多播地址转换为一个以太网地址，多播路由器就要截取 IP 多播地址中的最低 23 位，并将它们插入到以太网物理多播地址中（见图 12.4）。

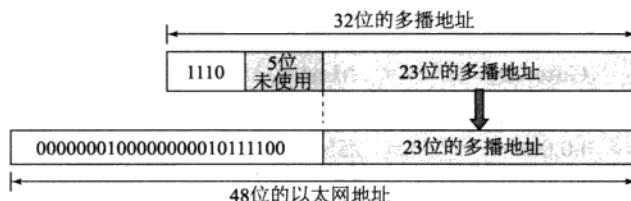


图 12.4 将 D 类地址映射为以太网物理地址

但在 IPv4 地址中，多播地址块的组标识长度是 28 位，也就是说还有 5 位没有使用到。这就意味着 IP 级的每 32 (2^5) 个多播地址会映射到同一个物理地址。换言之，这个映射不是一对一的，而是多对一的。如果多播地址中组标识的最左边 5 位不是全 0，那么接收到分组的主机就有可能并不是真正属于与该分组相关的组。为此，主机必须检查分组的 IP 地址，并丢弃任何不属于它的分组。

以太网物理多播地址的范围是：01:00:5E:00:00:00 ~ 01:00:5E:FF:FF:FF

例 12.2

请将 IP 多播地址 232.43.14.7 改写成一个以太网物理多播地址。

解

我们可以分两步来完成：

a. 先将 IP 地址的最右边 23 位写成十六进制。可以这样做：先把最右边的 3 个字节转换为十六进制，然后，如果这个十六进制数中最左边一位大于或等于 8，就把这个数字减去 8。在我们的例子中得到的结果就是 2B:0E:07。

b. 把上一步计算出的结果加上起始以太网多播地址，也就是 01:00:5E:00:00:00，最后得到的结果就是

01:00:5E:2B:0E:07

例 12.3

请将 IP 多播地址 238.212.24.9 改写成一个以太网物理多播地址。

解

a. 最右边的 3 个字节转换为十六进制是 D4:18:09，我们需要把这个数的最左边一位减去 8，得到的结果就是 54:18:09。

b. 把上一步计算出的结果加到起始以太网多播地址上，最后得到的结果就是

01:00:5E:54:18:09

不支持多播的网络

绝大多数广域网都不支持物理多播地址。要通过这样的网络发送一个多播分组就需要使用称之为隧道的处理过程。在使用隧道时，多播分组被封装成单播分组并通过网络传送，而当它出现在隧道的另一端时再转换为一个多播分组（见图 12.5）。

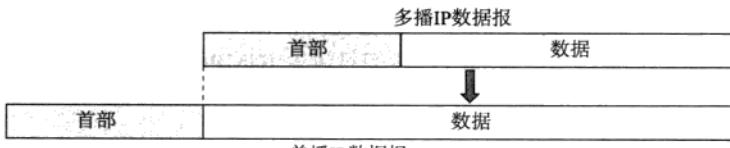


图 12.5 隧道

12.3 IGMP

多播通信表示由一个发送者给一组接收者发送报文，而这些接收者都是同一个组的成员。由于发送者只发送报文的一个副本，路由器需要对这个报文进行复制和转发，所以多播路由器必须掌握一张多播组的列表，这些组中至少有一个成员与它的某个接口相关。也就是说，多播路由器需要收集组成员的相关信息并与其它多播路由器共享。此类信息的收集工作分为两级：本地的和全球的。连接着某一个网络的多播路由器负责在本地收集此类信息，而收集到的信息可以传播到全球的其他路由器。前一个任务是通过 IGMP 协议完成的，而后一个任务则由多播路由选择协议完成。我们首先在本节讨论 IGMP。

网际组管理协议（Internet Group Management Protocol, IGMP）负责收集和解释一个网络中的组成员信息。它是为此目的而设计的 IP 层协议之一。图 12.6 给出了 IGMP 协议在网络层中的位置以及与其他协议之间的关系。

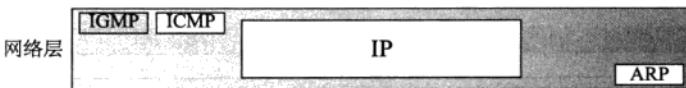


图 12.6 IGMP 在网络层中的位置

12.3.1 组管理

IGMP 不是一个多播路由选择协议，它是管理组成员关系的协议。在任何网络中总有一个或几个多播路由器负责把多播分组分发给主机或其他路由器。IGMP 协议把连接到网络上的主机（路由器）的成员关系状态信息交给多播路由器。

一个多播路由器每天都有可能接收成千上万个属于不同多播组的多播分组。如果路由器不知道这些主机的成员关系状态，那么它就要将收到的所有分组都进行广播，这会产生很大的通信量并消耗大量带宽。解决这个问题的一个较好的方法是保存一张有关网络中的多播组的列表，在这些组中至少有一个忠实成员(loyal member)。IGMP 帮助多播路由器创

建和更新这个表。

IGMP 是个组管理协议。它帮助多播路由器创建和更新与每一个路由器接口有关的忠实成员的列表。

IGMP 经历了三个版本的发展。版本 1 和版本 2 提供的是任意源多播 (any-source multicast, ASM)，也就是说，不管是从哪来的多播报文，组成员都会接收。IGMP 版本 3 则提供特定源多播 (source-specific multicast, SSM)，就是说接收者可以选择只接收来自预先定义好的源列表中的某个源的多播报文。在本节，我们只讨论 IGMPv3。

12.3.2 IGMP 报文

IGMPv3 有两种类型的报文：成员关系查询报文和成员关系报告报文。第一种类型有三种不同的格式：一般的格式、特定组的格式以及特定组和源的格式，如图 12.7 所示。

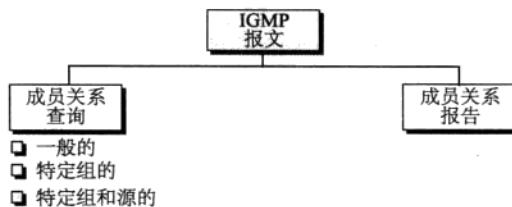


图 12.7 IGMP 报文

成员关系查询报文格式

成员关系查询报文是路由器为了找出网络中活跃的组成员而发送的报文。图 12.8 描绘了这种报文的格式。

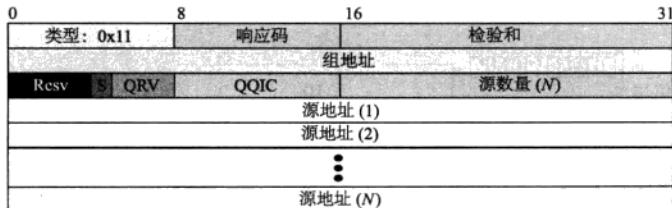


图 12.8 成员关系查询报文格式

其中每个字段的简要说明如下：

- **类型** 这个 8 位字段定义了报文的类型。值 0x11 表示成员关系查询报文。
- **最大响应码** 这个 8 位字段定义了在接收到这个查询后必须在多长时间内做出响应，稍后我们将会进一步解释。
- **检验和** 这个 16 位字段保存的是检验和。这个检验和是对整个报文做的计算。
- **组地址** 这个 32 位字段在一般的查询报文中置 0。而在发送特定组或特定组和源的查询报文时，这个字段设置为被查询的 IP 多播组地址。

- **Resv** 这个 4 位字段目前未用，留待将来使用。
- **S** 这是一个 1 位的抑制标志。当这个字段置 1 时表示此查询报文的接收者应当抑制正常的计时器更新。
- **QRV** 这个 3 位字段称为询问者的健壮性变量。它用于监视网络的健壮性。
- **QQIC** 这个 8 位字段称为询问者的查询间隔码。它用于计算询问者的查询间隔时间 (QQI)，稍后我们再来说明。
- **源数量 (N)** 这个 16 位字段定义了依附在此查询上的 32 位单播源地址的数量。对于一般的查询和特定组的查询来说，这个字段的值是零，而对于特定组和源的查询来说，这个字段的值非零。
- **源地址** 这些 32 位的字段列出了 N 个源地址，也就是多播报文的源点。 N 的值在前一个字段中定义。

查询报文的三种格式

在前面我们曾提到查询报文有三种格式：一般的查询格式、特定组的查询格式以及特定组和源的查询格式。每种格式都有不同的作用。图 12.9 所示为这三种类型的报文对比举例。

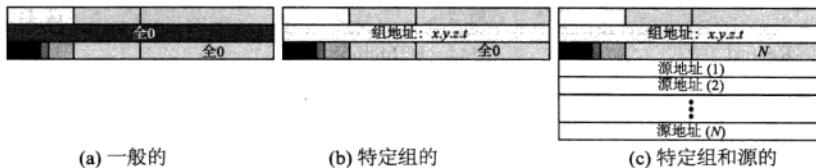


图 12.9 查询报文的三种格式

- 在一般的查询报文中，发起查询的路由器探询每个邻站，使之报告组成员关系的完整列表（对任何多播组都感兴趣）。
- 在特定组的查询报文中，发起查询的路由器探询每个邻站，使之报告是否仍然对某个特定的多播组感兴趣。这个多播组的地址在查询的组地址字段中用 $x.y.z.t$ 来指明。
- 在特定组和源的查询报文中，发起查询的路由器探询每个邻站，使之报告是否仍然对来自 N 个源之一的且到特定多播组 $x.y.z.t$ 的多播分组感兴趣，这些源的单播地址在分组中指明。

成员关系报告报文格式

图 12.10 所示为 IGMP 成员关系报告报文格式。

- **类型** 这个 8 位字段的值是 0x22，它定义了此报文的类型。
 - **检验和** 这个 16 位字段保存的是检验和。这个检验和是对整个 IGMP 报文做的计算。
 - **组记录数量 (M)** 这个 16 位字段定义了分组携带的组记录的数量。
 - **组记录** 在这个分组中可以有零个或多个可变长度的组记录。每个组记录中的字段将在下面加以解释。
- 每个组记录包含了响应者在某个多播组中的成员关系信息。下面是对各个字段的简要说明：
- **记录类型** 目前一共有六种记录类型，如表 12.3 所示。我们稍后将会了解它们的应用。

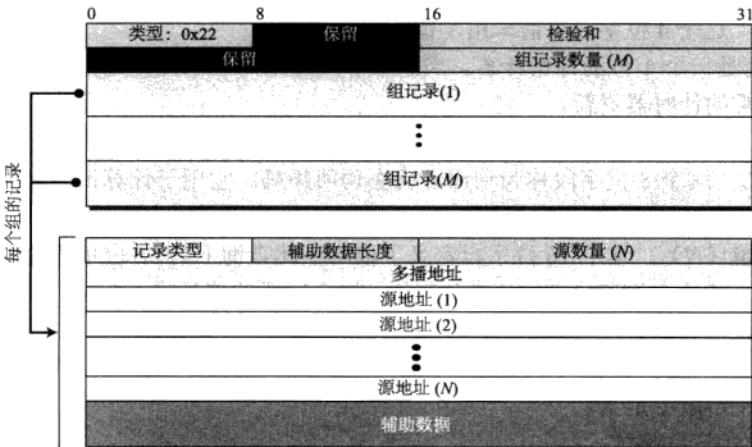


图 12.10 成员关系报告报文格式

表 12.3 记录类型

类别	类型	类型值
Current-State-Record (当前状态记录)	Mode_Is_Include	1
	Mode_Is_Exclude	2
Filter-Mode-Change-Record (过滤模式改变记录)	Change_To_Include_Mode	3
	Change_To_Exclude_Mode	4
Source-List-Change-Record (源列表改变记录)	Allow_New_Sources	5
	Block_Old_Sources	6

- 辅助数据长度 这个 8 位字段定义了每个组记录中包含的辅助数据的长度。如果这个字段的值为零，就表示在这个分组中不含辅助数据。如果这个值非零，它就指明了以 32 位字为单位的辅助数据的长度。
- 源数量 (N) 这个 16 位字段定义了依附在这个报告上的 32 位源多播地址的数量。
- 源地址 这些 32 位字段一共列出了 N 个源地址，其中 N 的值在前一个字段中定义。
- 辅助数据 这个字段包含的是任何需要被包含在这个报告报文中的辅助数据。IGMP 目前还没有定义任何辅助数据，但是将来可以在协议中加入辅助数据。

12.3.3 在主机上应用 IGMP 协议

本小节我们要讨论如何在一个主机上应用 IGMP 协议。

套接字状态

对组的管理从主机上的进程（正在运行的应用程序）开始，进程与接口相连。正如我们将在第 17 章中看到的，一个进程与一个套接字关联，对于套接字希望从哪个多播组接收多播报文，该进程都有一个记录。这个记录有两种显示模式：include 模式或 exclude 模式。如果记录是 include 模式的，那么它给出的是组报文可被套接字接受的单播源地址列表。如

果记录是 exclude 模式，那么它给出的是组报文不能被套接字接受的单播源地址列表。换言之，include 模式表示“仅...”，而 exclude 模式表示“所有的，除了...”。这些状态可以组成一张表，表中的每一行定义了一个记录。如果套接字希望接收多个多播组的多播报文，那么它就可能有多个记录。

例 12.4

图 12.11 所示为一台主机上的三个进程：S1，S2 和 S3。第一个进程只有一个记录，第二和第三个进程各有两个记录。我们用小写字母来表示源地址。



图例 S: 套接字
a, b, ... 源地址

状态表			
套接字	多播组	过滤器	源地址
S1	226.14.5.2	include	a, b, d, e
S2	226.14.5.2	exclude	a, b, c
S2	228.24.21.4	include	b, c, f
S3	226.14.5.2	exclude	b, c, g
S3	228.24.21.4	include	d, e, f

图 12.11 套接字状态

接口状态

如图 12.11 所示，套接字的记录中可能有重叠信息。两个或多个套接字可能具有同一个多播组的记录。为了提高效率，组管理要求将主机连接到网络的接口也要保存一个接口状态。最初这个接口状态是空的，当套接字记录发生变化时（此时创建也是一种变化），接口状态就开始建立。但是，接口状态只为每一个多播组保存一个记录。如果为相同的多播组又创建了一个新的套接字记录，那么需要修改相应的接口记录以反映最新的变化。例如，对应图 12.11 的套接字状态的接口状态仅含有两个记录，而不是五个记录，因为它仅涉及到两个多播组。不过，接口状态需要为整个状态设置一个接口计时器，此外还要对每个记录也设置一个计时器。我们稍后再讨论这些计时器的应用。合并记录存在的唯一问题就是对源列表的处理。如果同一个多播组的记录具有两个或多个源列表，那么在合并源列表时需要遵守以下两个规则：

1. 如果被合并的记录之中有某个记录具有 exclude 过滤模式，那么结果得到的接口记录也将具有 exclude 过滤模式，且源地址列表的生成方法如下：
 - 对用 exclude 过滤的所有地址列表进行交集操作。
 - 把 a 部分得出的结果与用 include 过滤的所有地址列表进行差集操作。
2. 如果被合并的所有记录都是 include 过滤模式，那么结果得到的接口记录也是 include 过滤模式，且新的源地址列表就是对所有地址列表的并集操作。

当任何一个套接字记录发生变化时，接口状态就要采用上述规则进行修改。

例 12.5

我们使用前面给出的两条规则来为例 12.4 中的主机建立接口状态。首先我们要找出每个多播组的源地址列表。

- a. 多播组 226.14.5.2 有两个 exclude 列表和一个 include 列表。结果得到的是如下计算

的 exclude 列表。我们用点号表示交集操作，用减号表示差集操作。

$$\text{exclude 源列表} = \{a, b, c\} \cdot \{b, c, g\} - \{a, b, d, c\} = \{c\}$$

b. 多播组 228.24.21.4 有两个 include 列表。结果得到的是如下计算的 include 列表。我们用加号来表示并集操作。

$$\text{include 源列表} = \{b, c, f\} + \{d, e, f\} = \{b, c, d, e, f\}$$

图 12.12 给出了这个接口的状态。从图中还可以看出，整个接口有一个计时器，而与每个多播组相关的各个状态也有各自的计时器。

接口状态			
多播组	组计时器	过滤器	源地址
226.14.5.2	计时器图标	exclude	c
228.24.21.4	计时器图标	include	b, c, d, e, f

图 12.12 接口状态

发送改变状态的报告

不管接口状态有任何变化，主机都需要立即用相应的组记录为该多播组发送一个成员关系报告报文。如图 12.13 所示，根据旧的状态过滤器和新的状态过滤器的组合，可能产生四种不同的变化。我们描绘的仅仅是组记录，而不是整个报告。

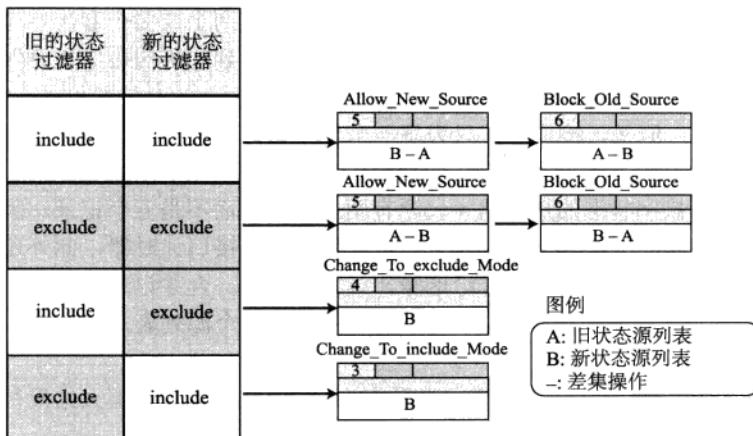


图 12.13 发送改变状态报告

如图所示，在前两种情况下，这个报告含有两个组记录，而在后两种情况下，这个报告仅含有一个组记录。

收到查询报告

当主机接收到一个查询时并不马上做出响应，它要延迟一段随机的时间后再响应，这个随机时间是根据最大响应码字段的值计算得到的（稍后再说明）。主机的响应动作取决于收到的查询的类型，如下所示：

- 如果收到的查询是一般的查询，主机就会用计算得到的时延值复位接口计时器（参见图 12.12）。这就意味着如果在此之前还有任何延迟的响应，都会被取消。

2. 如果收到的查询是特定组的查询，那么相应的组计时器（参见图 12.12）被复位为计时器剩余时间和计算后得到的时延中较短的那个时间。如果计时器未运行，它的剩余时间就被认为是无穷大的。

3. 如果收到的查询是特定组和源的查询，那么主机采取的动作与前一个动作相同。另外，这个延迟的响应中要记录源列表。

计时器到期

当计时器到期时，主机就发送成员报告报文。不过，这个报文中包含的组记录的类型和数量都取决于到期的计时器。

1. 如果到期的计时器是在收到一般的查询后设置的接口计时器，那么主机发送的成员关系报告中要为接口状态的每个组包含一个对应的 Current-State-Record 记录。这些记录的类型或者是 Mode-Is-Include（类型 1），或者是 Mode-Is-Exclude（类型 2），取决于该组的过滤模式。不过，如果需要发送的所有记录在一个报告中装不下，也可以分成几个报告来发送。

2. 如果到期的计时器是在收到特定组查询后设置的组计时器（也就是说接口没有为这个组记录源列表），当且仅当该组仍然是活跃的，则主机发送的成员关系报告中只包含这个特定组的一个 Current-State-Record 记录。包含在报告中的这个唯一的记录的类型或者是 Mode-Is-Include（类型 1），或者是 Mode-Is_Exclude（类型 2）。

3. 如果到期的计时器是在收到特定组和源查询后设置的组计时器（也就是说接口有这个组的源列表记录），当且仅当该组仍然是活跃的，则主机发送一个成员关系报告，其中只含有这个特定组的一个 Current-State-Record 记录。包含在报告中的这个唯一的记录的类型和源列表取决于该组的过滤器模式。

a. 如果该组的过滤器是 include，那么这个记录的类型是 Mode-Is-Include（类型 1），并且源列表是 $(A \cdot B)$ ，其中 A 是该组的源列表，B 是接收到的源列表，点号表示交集操作。

b. 如果该组的过滤器是 exclude，那么这个记录的类型是 Mode-Is-Exclude（类型 2），并且源列表是 $(B - A)$ ，其中 A 是该组的源列表，B 是接收到的源列表，减号表示差集操作。

报告抑制

在 IGMP 前面的两个版本中，如果一台主机接收到了由另一台主机发送的报告，它就会取消接口状态中对应的计时器，也就是说抑制了正在等待发送的报告。而在 IGMPv3 中，基于某些在 RFC 3376 中描述的原因，这个机制被从协议中被删除了。

12.3.4 IGMP 协议应用于路由器

在本小节中，我们将讨论一个路由器是如何应用 IGMP 协议的。在 IGMPv3 中经常被称为查询者的多播路由器比起前两个版本来说要复杂得多。查询者需要处理成员关系报告中包含的六种类型的组记录。

查询者的状态

多播路由器需要维持与每一个网络接口关联的每一个多播组的状态信息。一个网络接

口的状态信息是一张表，表中的每一行与一个多播组相关。每个多播组的信息由多播地址、组计时器、过滤模式和源记录组成。而且，每个源记录又包含了源的地址和相应的计时器。图 12.14 所示的例子为一个多播路由器与它的两张状态表，其中一张状态表与接口 m1 相关，另一张状态表与接口 m2 相关。



图 12.14 路由器状态

收到成员关系报告后采取的动作

多播路由器发送查询并接收报告。在这一小节，我们要说明当路由器收到一个报告后它的状态变化情况。

收到的报告是对一般的查询的响应 当路由器发送了一个一般的查询后，它就等待着接收一个或多个报告。这种类型的报告通常包含的是 Current-State-Record(类型 1 和类型 2) 记录。当此类报告到达时，路由器如图 12.15 所示的那样改变自己的状态。图中给出了根据当前状态和到达的报告中的记录，路由器中每一个记录的下一个状态。换言之，它是一张状态转换图。当路由器从到达的报告中提取出一个记录时，就要改变自身相应记录（具有相同的组地址）的状态并调用某些动作。我们分别给出了四组操作，一种状态改变对应一组操作。

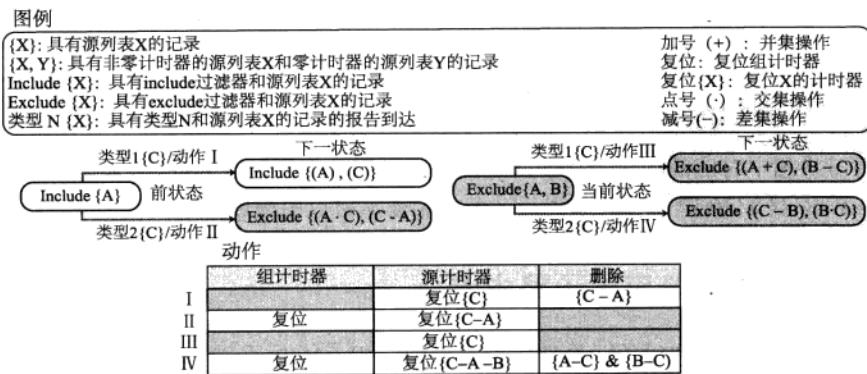


图 12.15 与一般的查询报告相关的变化状态

收到的报告是对其他查询的响应 当路由器发送了一个特定组或特定组和源的查询后就等待着接收一个或多个报告。这种类型的报告通常包含的是 Filter_Mode_Change_Record

(类型 3 和 4) 或 Source_List_Change_Record (类型 5 和 6) 的记录。当此类报告到达后, 路由器如图 12.16 所示的那样改变自己的状态。图中给出了根据当前状态以及到达的报告中所包含的记录类型, 路由器中每一条记录的下一状态。换言之, 它是一张状态转换图。我们一共有八种不同的情况。

图例



图 12.16 与特定组的以及特定组和源的查询报告相关的变化状态

12.3.5 IGMP 在转发中的作用

正如我们前面所讨论的, IGMP 是一种管理协议。它的作用只是通过收集各种信息来帮助连接网络的路由器, 使之能够为来自特定源并且以某个多播组为终点的分组做出转发或者不转发的决定。在前两个版本的 IGMP 中, 转发建议仅仅依据分组的多播目的地址, 而在 IGMPv3 中, 这个转发决定要依据目的地址和源地址。对于应用了 IGMPv3 的 IP 多播路由器, 当它在决定一个接口是否要转发一个分组时, RFC 3376 给出了六种不同情况下的建议。这些建议的基础是如图 12.14 所示的路由器接口状态表。如果一个分组的多播目的地址在这张路由器的状态表中不存在, 那么显然会建议不要转发该分组。如果这个多播目的地址存在于路由器的状态表中, 那么根据过滤器模式和源地址, 建议如表 12.4 所示。

表 12.4 IGMPv3 转发建议

过滤模式	源地址	源计时值	建议
include	在列表中	大于 0	转发
include	不在列表中	0	不转发

续表

过滤模式	源地址	源计时值	建议
include	不在列表中		不转发
exclude	在列表中	大于 0	转发
exclude	在列表中	0	不转发
exclude	不在列表中		转发

从表中可以看出有三种情况是建议转发 IP 多播数据报的。第一种情况（第一行）很明显，源地址在列表中，也就是说该网络中至少有一台主机希望接收这种类型的分组。第三种情况（第 6 行）也很清楚，过滤模式是 exclude 而这个源地址没有被排除，也就是说该网络中至少有一台主机需要接收来自这个源的组报文。不是很明显的是第二种情况（第 4 行）。过滤器模式是 exclude，而源地址也在这个列表中，也就是说这个源地址应当被排除，因此一般说来这个分组不应该转发。但在这种情况下之所以要转发这个分组的理由在于这个源的计时器还没有超时。只有当计时器超时了，这个源才被排除。

12.3.6 变量和计时器

在前面几个小节中，我们用到了一些变量和计时器，在这里需要进一步阐明它们的含义。

最大响应时间

最大响应时间是在响应一个查询而发送报告报文之前所允许的最大时间。它是根据查询分组的 8 位最大响应码字段计算得来的。换言之，这个最大响应时间是由查询来定义的。最大响应时间的计算过程如图 12.17 所示。如果最大响应码的值小于 128（最高位为零），那么这个最大响应时间是一个整数。如果最大响应码的值大于或等于 128（最高位为 1），那么这个最大响应时间是一个浮点数。

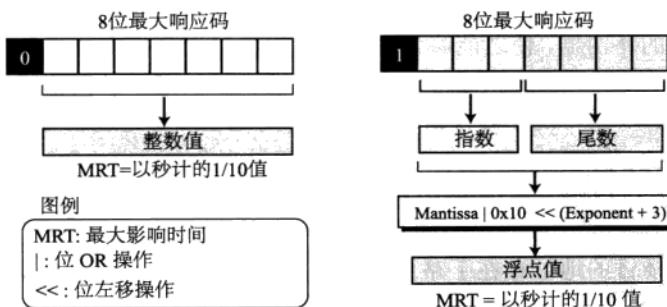


图 12.17 最大响应时间的计算

查询者的健壮性变量 (QRV)

IGMP 监视网络中分组的丢失情况（通过测量接收到的报告的时延），并调整 QRV 的值。这个变量的值指明了一个查询的响应报文应当被发送多少次。如果网络丢失分组的情

况比较严重，路由器就为 QRV 设置一个较高的值。主机应当为一个查询发送响应的次数是 QRV - 1。默认值是 2，也就是说主机至少要发送一次响应。如果路由器指明这个值为 5，就表示该主机需要发送四次响应。请注意，QRV 的值永远不会是 0 或 1，如果是则被看成是默认值 2。

查询者的查询间隔

这是发送一般的查询之间的时间间隔。默认值是 125。这个默认值可被管理员修改，以控制网络通信量。

其他变量和计时器

在 RFC 3376 中还定义了其他几个变量和计时器，管理员可能对它们更感兴趣。更多详情我们建议阅读 RFC 3376。

12.3.7 封装

IGMP 报文封装在 IP 数据报中，其协议字段的值设置为 2，TTL 字段置 1。不过这个数据报的目的 IP 地址取决于这个报文的类型，如表 12.5 所示。

表 12.5 目的 IP 地址

报文类型	IP 地址
一般的查询	224.0.0.1
其他查询	组地址
报告	224.0.0.22

12.3.8 与老版本之间的兼容

为了兼容版本 1 和版本 2，IGMPv3 软件被设计为可以接受在版本 1 和版本 2 中定义的报文。表 12.6 给出了版本 1 和版本 2 中类型字段的值和相应的报文类型。

表 12.6 版本 1 和版本 2 中的报文

版本	类型值	报文类型
1	0x11	查询
	0x12	成员关系报告
2	0x11	查询
	0x16	成员关系报告
	0x17	退出组

12.4 多播路由选择

现在我们可以来说明由 IGMP 收集到的信息如何用多播路由选择协议传播给其他路由

器。不过，我们首先要讨论最佳路由选择的概念，这对所有多播协议都是一样的。然后再给出多播路由选择协议的概述。

12.4.1 最佳路由选择：最短路径树

最佳域间路由选择过程最终的结果就是找出一个**最短路径树** (shortest path tree)。这个树的根是源点，而树叶是所有可能的终点。从根到每一个终点的路径都是最短路径。但是，在单播和多播路由选择中，树的数目和构成是不同的。让我们来分别讨论。

单播路由选择

在单播的情况下，当路由器收到一个分组要转发时，它需要找出到达这个分组终点的最短路径。路由器从路由表中找出到达这个特定终点的信息。对应于这个终点的下一跳是最短路径的起点。路由器知道到达每一个终点的最短路径，这表示路由器有一个最短路径树使得到达所有终点的路径都是最佳的。换言之，路由表的每一行都是最短路径，而整个路由表就是一个最短路径树。在单播路由选择的情况下，路由器在转发分组时只需要一个最短路径树，不过，每个路由器有它自己的最短路径树。图 12.18 描绘了这种情况。

这个图给出了路由表以及路由器 R1 的最短路径树的详细情况。路由表中的每一行对应于从根到对应网络的一条路径。整个路由表则给出了一个最短路径树。

在单播路由选择的情况下，域中的每个路由器都有一张定义了到达可能终点的最短路径的路由表。

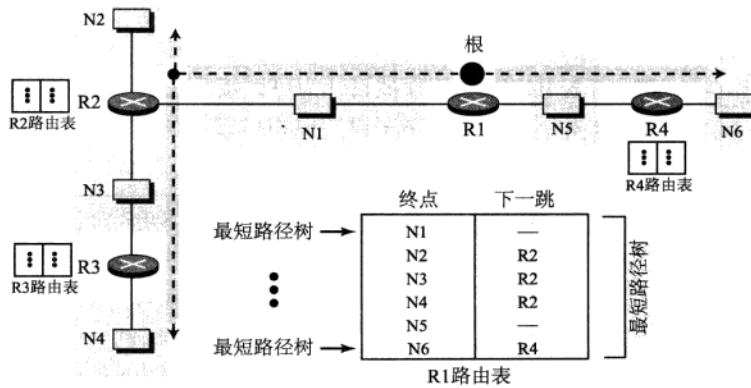


图 12.18 单播路由选择时的最短路径树

多播路由选择

当路由器收到一个多播分组时，情况就有所不同了。一个多播分组的终点可能在多个网络中。把一个分组转发给一组成员需要一个最短路径树。如果我们有 n 个组，那么就需要有 n 个最短路径树。我们可以想象得出多播路由选择的复杂性。有两种办法可以解决这

个问题：源点基准树和组共享树。

在多播路由选择时，每一个相关的路由器需要对每一个组构建一个最短路径树。

源点基准树 使用源点基准树（source-based tree）的方法时，每一个路由器需要对每一个组有一个最短路径树。对一个组的最短路径树定义了到每一个网络（在这个网络中有该组的忠实成员）的下一跳。在图 12.19 中，我们假定域中只有 5 个组：G1、G2、G3、G4 和 G5。现在 G1 在 4 个网络中有忠实成员，G2 在 3 个网络中有忠实成员，G3 在 2 个网络中有忠实成员，G4 在 2 个网络中有忠实成员，而 G5 也在 2 个网络中有忠实成员。我们还在每个网络上标出了有忠实成员的组的名称。这个图还给出了路由器 R1 的多播路由表。每个组有一个最短路径树，因此 5 个组就有 5 个最短路径树。如果路由器 R1 收到一个终点地址为 G1 的分组，它就必须把分组的一个副本发送给连接的网络，再把一个副本发送给路由器 R2，还有一个副本发送给路由器 R4，这样做 G1 的所有成员才能都收到一个副本。

使用这种方法，如果组的数目是 m ，每一个路由器就需要有 m 个最短路径树，每个组对应一个树。如果有几百个或几千个组，那么我们可以想象路由表会有多么复杂。但是，后面将会看到不同的协议是如何试图缓解这种状况的。

使用源点基准树的方法，每个路由器需要对每个组都有一个最短路径树。

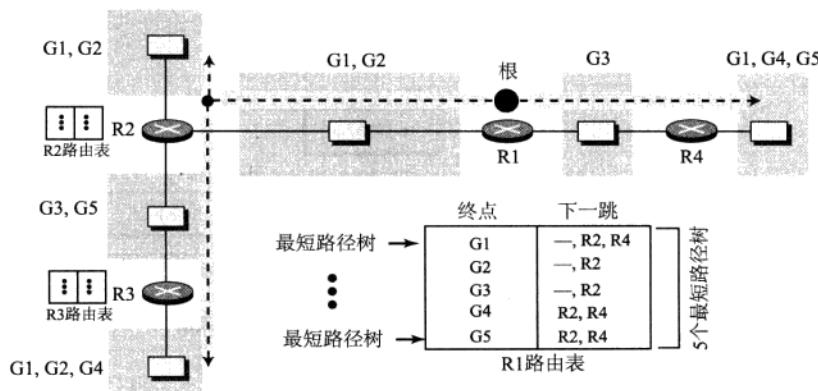


图 12.19 源点基准树

组共享树 使用组共享树（group-shared tree）的方法时，不是每个路由器都要有 m 个最短路径树，而是只有一个称为中心核心路由器或汇集点路由器的特定路由器负责多播通信量的传播。核心路由器在其路由表中有 m 个最短路径树。域中的其他路由器则一个都没有。如果某个路由器收到一个多播分组，它就把这个分组封装成为一个单播分组，然后发送给核心路由器。核心路由器拆开封装，取出多播分组，查找路由表为这个分组选择路由。图 12.20 给出了这个概念。

使用组共享树的方法，只有拥有每个组的最短路径树的核心路由器才涉及到多播。

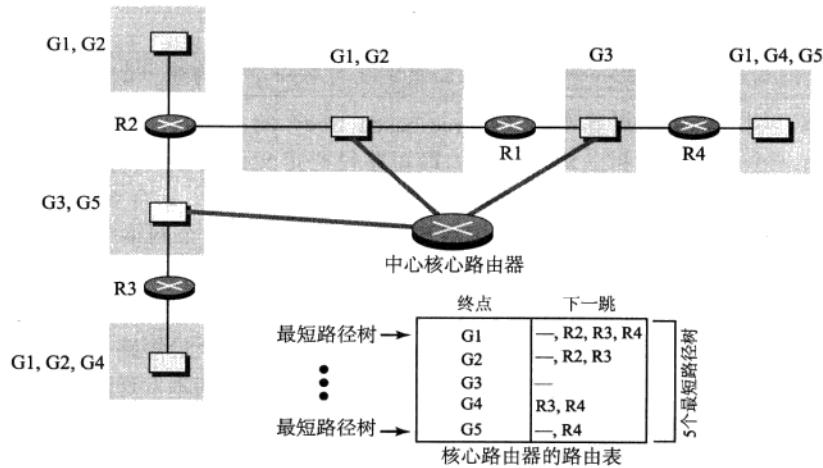


图 12.20 组共享树方法

12.5 路由选择协议

在过去的几十年中出现过好几种多播路由选择协议。其中有一些协议是单播路由选择协议的扩展，而有些则是全新的。我们将在本章的剩下部分来讨论这些协议。图 12.21 给出这些协议的分类。

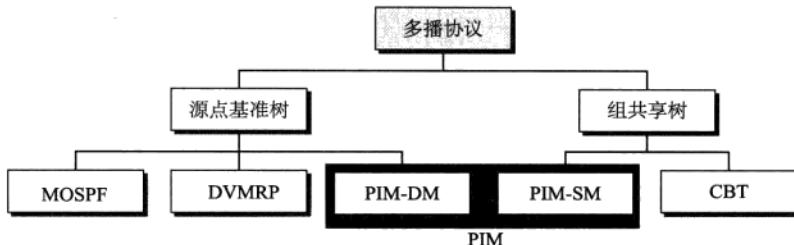


图 12.21 常用多播路由选择协议的分类

12.5.1 多播链路状态路由选择：MOSPF

在本小节，我们要简单地讨论多播链路状态路由选择及其在因特网中的实现，即 MOSPF。

多播链路状态路由选择

我们已经在第 11 章中讨论过单播链路状态路由选择。我们讲过，每一个路由器使用 Dijkstra 算法创建一个最短路径树。路由表就是对这个最短路径树的转换。多播链路状态路由选择是单播链路状态路由选择的直接扩展，使用的是源点基准树的方法。虽然单播路由选择涉及到很多方面，但它在多播路由选择上的扩展却是比较简单和直截了当的。

多播链路状态路由选择使用源点基准树的方法。

回忆在单播路由选择的情况下，每一个结点需要通告它的链路状态。对于多播路由选择来说，结点需要修正对状态的解释。结点要通告的是在这条链路上有任何忠实成员的每一个组。在这里状态的含义是指“哪些组在这个链路上是活跃的”。关于组的信息来自 IGMP（见本章前面的讨论）。每个运行了 IGMP 的路由器询问链路上的主机以找出成员关系状态。

当一个路由器收到所有这些 LSP 时，它就生成了 n (n 是组的数目) 个拓扑，根据这些拓扑，使用 Dijkstra 算法就可以得出 n 个最短路径树。因此，每一个路由器的路由表显示出和组数一样多的最短路径树。

这个协议唯一的问题就是创建这样多的最短路径树所需的空间和时间。解决问题的方法是仅当必要时才创建树。当一个路由器收到一个具有多播地址的分组时，它就运行 Dijkstra 算法为这个组计算最短路径树。其结果会保存在高速缓存中，以便其他的也要到这个终点的分组到达时就可以直接使用这个结果。

MOSPF

多播开放最短通路优先 (Multicast Open Shortest Path First, MOSPF) 协议是 OSPF 协议的扩展，它使用多播链路状态路由选择来创建源点基准树。这个协议需要一个新的链路状态更新分组，把主机的单播地址和组地址或主机负责的地址联系起来，这个分组就称为组成员关系 LSA。用这种方法，我们可以在树中只包含属于这个组的主机（使用它们的单播地址）。换言之，我们构建了包含属于一个组的所有主机，但是我们在计算中使用了主机的单播地址。为了提高效率，路由器在需要时才计算最短路径树（当这个路由器收到第一个多播分组时）。此外，这个树可以保存在高速缓存中，以便以后有同样源点/组地址对的分组可以使用它。MOSPF 是数据驱动 (data-driven) 的协议，也就是说当 MOSPF 路由器第一次收到具有给定源点和组地址的数据报时，它才构造这个 Dijkstra 最短路径树。

12.5.2 多播距离向量路由选择

在本节，我们要简单地讨论多播距离向量路由选择及其在因特网中的实现 DVMRP。

多播距离向量路由选择

单播距离向量路由选择相当简单，但要把它扩展为多播路由选择则很复杂。多播路由选择不允许一个路由器把自己的路由表发送给邻站。其思想就是要使用单播距离向量路由表中的信息从零开始创建一个路由表。

多播距离向量路由选择使用源点基准树，但路由器从来没有真正地构造过一个路由表。当路由器收到一个多播分组时，它就转发这个分组，好像查找了路由表一样。我们可以说，这个最短路径树是一边生成一边消失的。当它被使用后（在分组被转发后），这个路由表就完全消失了。

要完成这件事，多播距离向量算法要使用基于四种判决策略的处理过程。每种策略都建立在它的前一策略之上。我们将逐个说明它们，并讨论每一种策略是怎样改进了前一种的不足之处。

洪泛

首先想到的策略就是洪泛。一个路由器收到了一个分组，连看都不看这个分组的目的组地址就把它从所有的接口（除了接收这个分组的接口之外）转发出去。洪泛（flooding）完成了多播的首要任务，即每一个具有活跃成员的网络都能收到这个分组。但是，那些没有活跃成员的网络也同样收到了这个分组。这是一种广播而不是多播。还有一个问题：它会产生环路。离开这个路由器的一个分组可能会从另一个接口或同一个接口又折返回来，然后再转发出去。某些洪泛协议为了避免环路，先把这个分组的副本保留一小段时间，然后丢弃任何重复的副本。下一个策略，反向路径转发，改正了这个缺点。

洪泛对分组进行广播，但在系统中产生了环路。

反向路径转发

反向路径转发（reverse path forwarding, RPF）是修正后的洪泛策略。为了防止环路，仅有一个副本被转发，其他所有副本都被丢弃。在使用 RPF 时，路由器仅转发从源点到这个路由器走的是最短路径的副本。要找出这个副本，RPF 利用单播路由表。路由器收到一个分组并提取它的源地址（单播地址）。它咨询自己的单播路由表，就好像它想把这个分组发送给源地址一样。路由表会告诉路由器下一跳的地址。如果这个多播分组正好是从路由表定义的这个下一跳到达的，那么这个分组就已经走过了从源点到该路由器的最短路径，因为在单播距离向量路由选择协议中的最短路径是对称的。如果从 A 到 B 的路径是最短的，那么从 B 到 A 的路径也是最短的。如果分组是从最短路径来的，那么路由器就转发它，否则就丢弃它。

这个策略可防止环路，因为从源点到这个路由器之间总是只存在一条最短路径。如果分组离开了路由器后又折返回来了，那么它走过的就不是最短路径。为了使这个概念更加清楚，让我们看一下图 12.22。

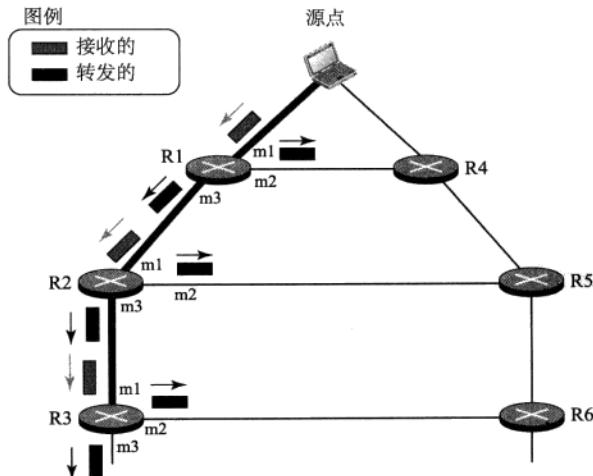


图 12.22 RPF

图中给出了某个域的一部分以及一个源点。路由器 R1、R2 和 R3 计算出的最短路径树如粗线所示。当 R1 在接口 m1 收到一个从源点发来的分组，它在咨询路由表后发现从 R1 到源点的最短路径是要经过接口 m1，于是它就转发这个分组。但是，如果分组的一个副本从接口 m2 到达，就要被丢弃，因为从 R1 到源点的最短路径上没有 m2。在 R2 和 R3 上也发生着同样的故事。你可能会想，如果有两个分组的副本从 R3 的接口 m1 到达，然后又经过 R6、R5 和 R2 后再次从接口 m1 到达 R3，会发生什么事情呢？这个接口对 R3 来说是正确的接口。这个分组的副本应当转发吗？答案是：这种情况永远不会发生，因为当分组从 R5 到 R2 时，会被 R2 丢弃而永远不会到达 R3。通向源点的上游路由器总是把没有经过最短路径的分组丢弃，这样就不会给下游的路由器造成混乱。

RPF 消除了洪泛过程中的环路。

反向路径广播

RPF 保证在不形成环路的条件下每个网络都能收到多播分组的一个副本。但是，RPF 不能保证每个网络只收到一个副本，实际上一个网络可能收到两个或更多的副本。原因在于 RPF 不是基于目的地址（组地址）的，其转发是基于源地址的。为了更加形象地了解这个问题，让我们来看图 12.23。

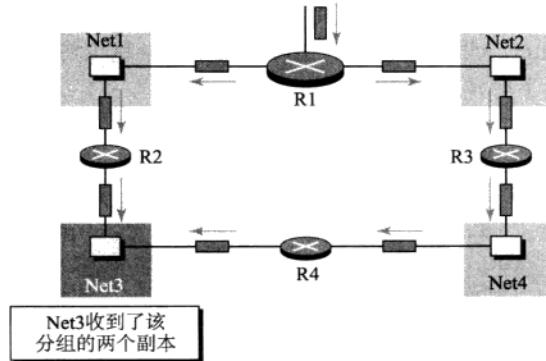


图 12.23 RPF 的问题

图中的 Net3 收到了分组的两个副本，虽然每个路由器从每个接口只发送了一个副本。出现重复是因为这里没有形成一个树，我们使用的不是树，而是图。Net3 有两个父结点：路由器 R2 和路由器 R4。

要消除这种重复，我们必须对每个网络只定义一个父路由器。我们必须做出这样的限制：一个网络只能从一个特定的父路由器那里接收来自特定源点的多播分组。

现在，这个策略已经很清楚了。对每一个源点，路由器只能从这样的接口向外发送分组，即对这个接口来说，该路由器是特定的父路由器。这个策略叫做反向路径广播 (reverse path broadcasting, RPB)。RPB 保证分组能够到达每一个网络，且每个网络只收到分组的一个副本。图 12.24 描绘了 RPF 和 RPB 之间的区别。

读者可能会问这个特定的父路由器是怎样确定的。特定的父路由器可以是与源点之间具有最短路径的路由器。因为路由器彼此之间要周期性地发送更新分组（在 RIP 中），它们能够很容易地确定相邻的哪一个路由器到源点的路径最短（这时要把源点解释为终点）。如

果有多个路由器合格，就选择 IP 地址最小的一个。

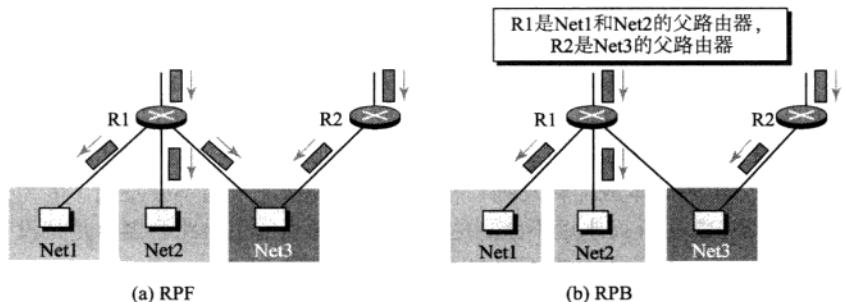


图 12.24 RPF 和 RPB 的对比

RPB 创建了从源点到达每一个终点的最短路径广播树。它保证每一个终点都收到且仅收到分组的一个副本。

反向路径多播 (RPM)

大家可能已经注意到了，RPB 对分组不是进行多播而是广播。这是低效率的。为了提高效率，多播分组必须只到达拥有特定组的活跃成员的网络。这就称为反向路径多播 (reverse path multicasting, RPM)。要把广播转变为多播，这个协议使用了两个过程，剪枝和嫁接。图 12.25 给出了剪枝和嫁接的概念。

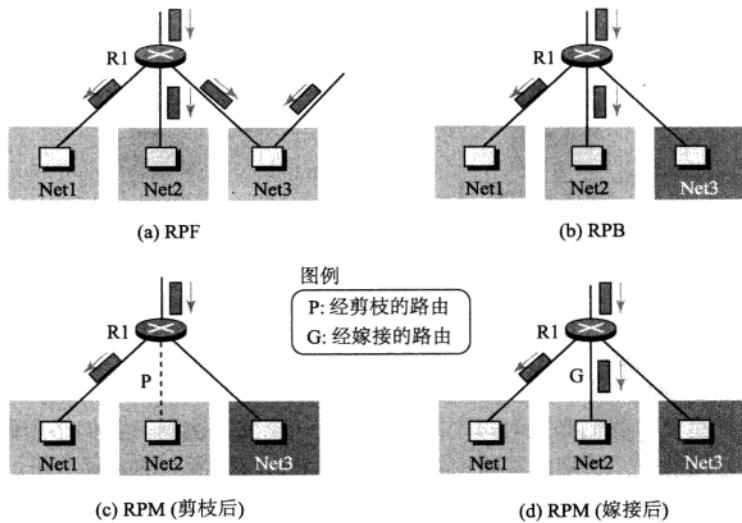


图 12.25 RPF、RPB 和 RPM

剪枝 每一个网络的特定的父路由器负责保留成员关系信息。这是通过前面介绍过的 IGMP 协议完成的。当连接到某个网络的路由器发现对某个多播分组没有兴趣时，这个过程就开始。这个路由器向上游路由器发送剪枝报文 (prune message)，使之剪枝相应的接口。这就是说，上游路由器停止通过该接口给这个组发送多播分组。如果这个路由器收到了来

自所有下游路由器的剪枝报文，那么就该轮到它向自己的上游路由器发送剪枝报文了。

嫁接 如果一个叶路由器（在树的底部的叶子）已经发送了一个剪枝报文，但突然通过 IGMP 发现它的某个网络又对接收该多播报文感兴趣了，那么它该怎么办？它可以发送嫁接报文（graft message）。嫁接报文强迫上游路由器恢复发送多播报文。

RPM 给 RPB 增加了剪枝和嫁接，以产生可支持动态成员关系变化的多播最短路径树。

12.5.3 DVMRP

距离向量多播路由选择协议（Distance Vector Multicast Routing Protocol, DVMRP）是多播距离向量路由选择的一个实现。它是一个基于 RIP 的源点基准路由选择协议。

12.5.4 CBT

核心基干树（Core-Based Tree, CBT） 协议是一个组共享协议，它使用一个核心作为树的根。自治系统划分成许多区，而每一个区选择一个核心（中心路由器或汇集点路由器）。

树的形成

当汇集点路由器选择好后，被选择的路由器的单播地址就通知给每一个路由器。每一个有感兴趣的组的路由器就发送一个单播加入报文（类似于嫁接报文）表示它愿意加入该组。这个报文途经发送方和汇集路由器之间的所有路由器。每一个中间路由器从这个报文中提取需要的信息，如发送站的单播地址以及这个分组到达的接口，然后再把这个分组转发给下一个路由器。当汇集路由器收到从这个组的每一个成员发来的所有的加入报文后，这个树就形成了。现在每一个路由器都知道它的上游路由器（通往树根的路由器）和下游路由器（通往树叶的路由器）。

如果某个路由器想退出这个组，它就给上游路由器发送退出报文。上游路由器就从树中删除到这个路由器的链路，并把这个报文转发给自己的上游路由器，依此类推。图 12.26 给出了一个组共享树和它的汇集路由器。

读者可能已经注意到 DVMRP 和 MOSPF 这两者与 CBT 之间有两个不同点。首先，前两者的树是从树根向上构造的，而 CBT 的树是从树叶向下构造的。其次，在 DVMRP 中，先构造树（广播），然后再剪枝，而在 CBT 中，开始时并没有树，而是使用加入（嫁接）的方法逐渐地构造出这个树。

发送多播分组

在形成了多播树后，任何一个源点（属于或不属于这个组的）都可以向这个组的所有成员发送多播分组。它只是简单地用汇集路由器的单播地址把分组发送给该汇集路由器，然后汇集路由器再把这个分组分发给所有的组成员。图 12.27 描绘了一个主机怎样把多播分组发送给所有的组成员。请注意，源主机可以是这个共享树以内的或以外的任何主机。在图中，我们给出的主机在共享树的外面。

选择汇集路由器

这个方法除了一点以外其他都很简单。我们要怎样选择汇集路由器使得这个处理过程

和多播效果都是最优化的？有几种方法已经实现了。但是，这个话题已经超出了本书的范围，因此我们将其交给更深入的一些书籍。

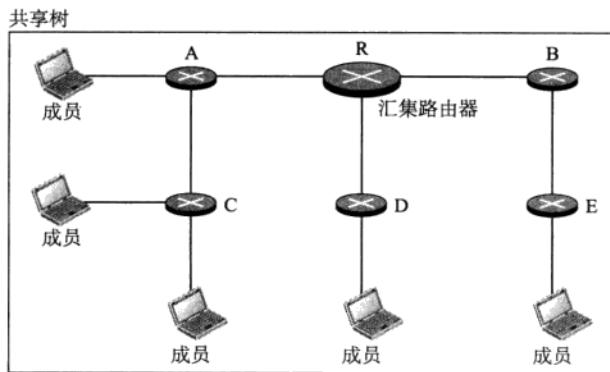


图 12.26 组共享树和它的汇集路由器

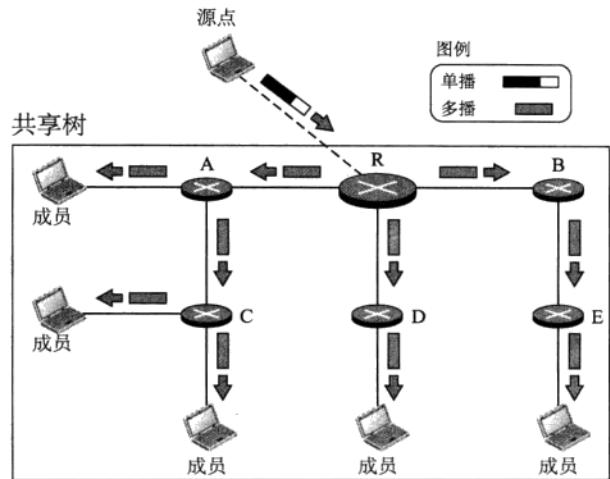


图 12.27 向汇集路由器发送多播分组

小结

归纳一下，核心基干树是一种组共享树，这个基于中心的协议为每个组使用一个树。树中有一个路由器称为核心。从源点向组成员发送分组的过程如下：

1. 源点可以是也可以不是树的一部分，它把多播分组封装在一个单播分组中，利用核心路由器的单播目的地址，把这个分组发送给核心路由器。这一部分的交付是用单播地址完成的，唯一的接收者是核心路由器。
2. 核心路由器把这个单播分组拆封，然后将其转发给所有“感兴趣”的接口。
3. 收到多播分组的每一个路由器接着再把它转发到所有“感兴趣”的接口。

在使用 CBT 时，源点把多播分组（封装成单播分组）发送给核心路由器。核心路由器把这分组拆封，再转发给所有感兴趣的接口。

12.5.5 PIM

协议无关多播（Protocol Independent Multicast, PIM）是给两个独立的多播路由选择协议：协议无关多播—密集方式（Dense Mode, PIM-DM）和协议无关多播—稀疏方式（Sparse Mode, PIM-SM）的统一名称。这两个协议都与单播协议有关，但它俩的相似之处仅止于此。我们将对它们分别进行讨论。

PIM-DM

PIM-DM 用在每个路由器都涉及多播的可能性很大的情况下（密集方式）。在这种环境中，使用对分组进行广播的协议是合适的，因为在此过程中几乎涉及到了所有的路由器。

PIM-DM 用在密集多播环境下，例如局域网的环境。

PIM-DM 是一种源点基准树路由选择协议，它使用 RPF 和剪枝/嫁接策略来进行多播。它的操作很像 DVMRP，但是，与 DVMRP 不同的是它并不依赖于某个特定的单播协议。它假定这个自治系统使用了某种单播协议，而每一个路由器的路由表都能够找出到达一个终点的最佳路径的转发接口。这个单播协议可以是距离向量协议（RIP），也可以是链路状态协议（OSPF）。

PIM-DM 使用 RPF 和剪枝/嫁接策略来处理多播。但是，它不依赖于底层的单播协议。

PIM-SM

PIM-SM 用在每个路由器都涉及多播的可能性很小的情况下（稀疏方式）。在这种环境中，使用对分组进行广播的协议是不合适的，类似 CBT 这种使用组共享树的协议会更合适些。

PIM-SM 用于稀疏多播环境，如在广域网中。

PIM-SM 是一种组共享树路由选择协议，它有一个汇集点（RP）作为树的源点。它的操作与 CBT 相似，不过更加简单些，因为它不需要确认加入报文。此外，为了防止 RP 出故障，它创建了一组备份的 RP。

PIM-SM 的特点之一就是在有必要时它可以从组共享树策略切换到源点基准树策略。如果在远离 RP 的地方出现活动密集区域，就会发生这种情况。用源点基准树策略代替组共享树策略在这样的区域中可以更有效地处理多播。

PIM-SM 与 CBT 相似，但使用了更加简单的过程。

12.6 MBONE

多媒体和实时通信增加了因特网的多播需求。但是，只有一小部分的因特网路由器是多播路由器。换言之，某个多播路由器在转发多播分组时可能会在近邻找不到其他的多播路由器。虽然可以在近几年内越来越多地推广多播路由器来解决这个问题，不过还有解决这个问题的另一种方法，那就是隧道技术。多播路由器可以被看成是位于单播路由器之上的一组路由器。这些多播路由器可能并没有直接连起来，但它们在逻辑上是连接在一起的。图 12.28 给出了这一概念。在这个图中，只有在有阴影圆圈中的路由器才有多播的能力。如果不使用隧道技术，那

么这些路由器就是孤立的。为了使多播能够顺利进行，我们利用隧道技术的概念用这些孤立的路由器构造出一个**多播主干网**（multicast backbone, MBONE）。

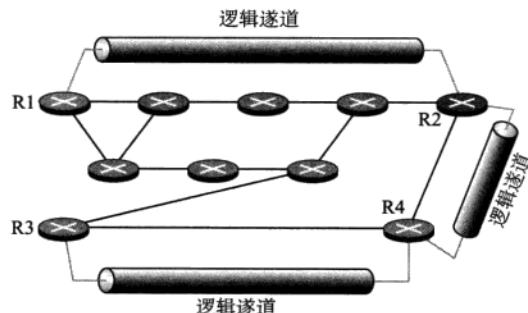


图 12.28 逻辑隧道技术

把多播分组封装在单播分组中就建立了逻辑隧道。多播分组变成了这个单播分组的有效载荷（数据）。这些中间的（非多播）路由器以单播路由器的身份转发分组，并把这个分组从一个多播路由器交付到另一个多播路由器。这样做就好像这些单播路由器并不存在，而这两个多播路由器是相邻的。图 12.29 给出了这个概念。到目前为止，支持 MBONE 和隧道技术的唯一协议就是 DVMRP。

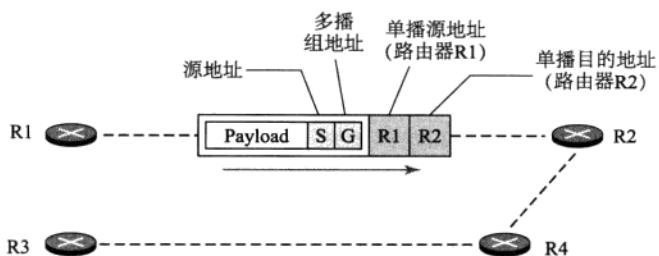


图 12.29 MBONE

12.7 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

12.7.1 参考书

有几本书全面覆盖了本章所讨论的内容。我们推荐[Com 06]、[Tan 03]和[Wit & Zit 01]。

12.7.2 RFC

有几个 RFC 涉及到了本章所讨论的内容，包括：RFC 1705、RFC 1585、RFC 2189、

RFC 2362 和 RFC 3776。

12.8 重要术语

核心基干树（CBT）协议	协议无关多播—密集方式（PIM-DM）
数据驱动	协议无关多播—稀疏方式（PIM-SM）
距离向量多播路由选择协议（DVMRP）	剪枝报文
洪泛	反向路径广播（RPB）
嫁接报文	反向路径转发（RPF）
组共享树	反向路径多播（RPM）
因特网组管理协议（IGMP）	最短路径树
多播主干网（MBONE）	源点基准树
多播开放最短通路优先（MOSPF）	隧道技术
协议无关多播（PIM）	

12.9 本章小结

- 多播就是向多个接收者同时发送相同的报文。多播有很多应用，包括分布式数据库、信息发布、电视会议和远程学习。
- 在无分类编址中，地址块 224.0.0.0/4 被用作多播地址。这个地址块有时也称为多播地址空间，并且根据不同的用途被划分为几个小地址块。
- 在收集本地的组成员状态信息时要涉及到因特网组管理协议（IGMP）。IGMP 最新的版本是 IGMPv3，它使用了两种类型的报文：查询和报告。
- 在使用源点基准树方式进行多播路由选择时，源点和组一起决定了这个树。RPF、RPB 和 RPM 是对源点基准树效率的不断提高。MOSPF/DVMRP 和 PIM-DM 是使用源点基准树方式进行多播的两种协议。
- 在使用组共享方式进行多播时，一个汇集路由器专门负责把多播报文分发到它们的终点。CBT 和 PIM-SM 是组共享树协议的例子。
- 为了在两个非邻接的多播路由器之间进行多播，我们构造了多播主干网（MBONE）以便使用隧道技术。

12.10 实践安排

12.10.1 习题

1. 请详细说明为什么我们对表 12.1 中的下列地址块不能使用 CIDR 记法。

- a. 在 224.0.2.0~224.0.255.255 之间的 AD-HOC 地址块。
 - b. 在 224.3.0.0~231.255.255.255 之间的第一个保留的地址块。
 - c. 在 234.0.0.0~238.255.255.255 之间的第二个保留的地址块。
2. 一个组织的 AS 号是 24101。试找出在 GLOP 地址块中这个组织可用的多播地址段。
3. 一个组的多播地址是 232.24.60.9。对于一个使用了 TCP/IP 的局域网来说，它的 48 位以太网地址是什么？
4. 将以下 IP 多播地址转换为以太网多播地址。其中有哪几个指向相同的以太网地址？
- a. 224.18.72.8
 - b. 235.18.72.8
 - c. 237.28.6.88
 - d. 224.88.12.8
5. 为什么没有必要让 IGMP 报文在它自己的网络以外传送？
6. 请回答以下问题：
- a. 在 IGMPv3 中一般的查询报文的长度是多少？
 - b. 在 IGMPv3 中特定组的查询报文的长度是多少？
 - c. 在 IGMPv3 中特定组和源的查询报文的长度是多少？
7. 在 IGMPv3 中，如果报告报文中含有三个记录，且每个记录包含了 5 个源地址（忽略辅助数据），此报告报文的长度是多少？
8. 有一台主机具有两个套接字：S1 和 S2，请给出它的套接字状态表（与图 12.11 类似）。其中 S1 是组 232.14.20.54 的成员，S2 是组 232.17.2.8 的成员。S1 希望只接收来自 17.8.5.2 的多播报文，而 S2 希望接收除了 130.2.4.6 以外的其他所有源发来的多播报文。
9. 试给出练习 8 中主机的接口状态。
10. 如果练习 9 中的套接字 S1 声称它还希望接收来自源 124.8.12.6 的报文，试给出其主机发送的组记录。
11. 如果练习 9 中的套接字 S2 声称它想退出组 232.17.2.8，试给出其主机发送的组记录。
12. 如果练习 9 中的套接字 S1 声称它想加入组 232.33.33.7，且愿意接收来自任何源的报文，试给出其主机发送的组记录。
13. 在图 12.14 中，如果路由器收到一个报告，其中的记录表明有一个主机希望加入组 232.77.67.60，并且愿意接收来自任何源点的报文，试给出该路由器接口 m1 的状态。
14. 请给出最大响应码分别为以下两种情况时的 MRT 值（以秒为单位）：
- a. 125
 - b. 220
15. 有一个 IGMP 报文的内容用十六进制表示如下：

11 03 EE FF E8 0E 15 08

请回答以下问题：

- a. 它是什么类型？
- b. 它的检验和是什么？
- c. 它的组标识是什么？

16. 有一个 IGMP 报文的内容用十六进制表示如下：

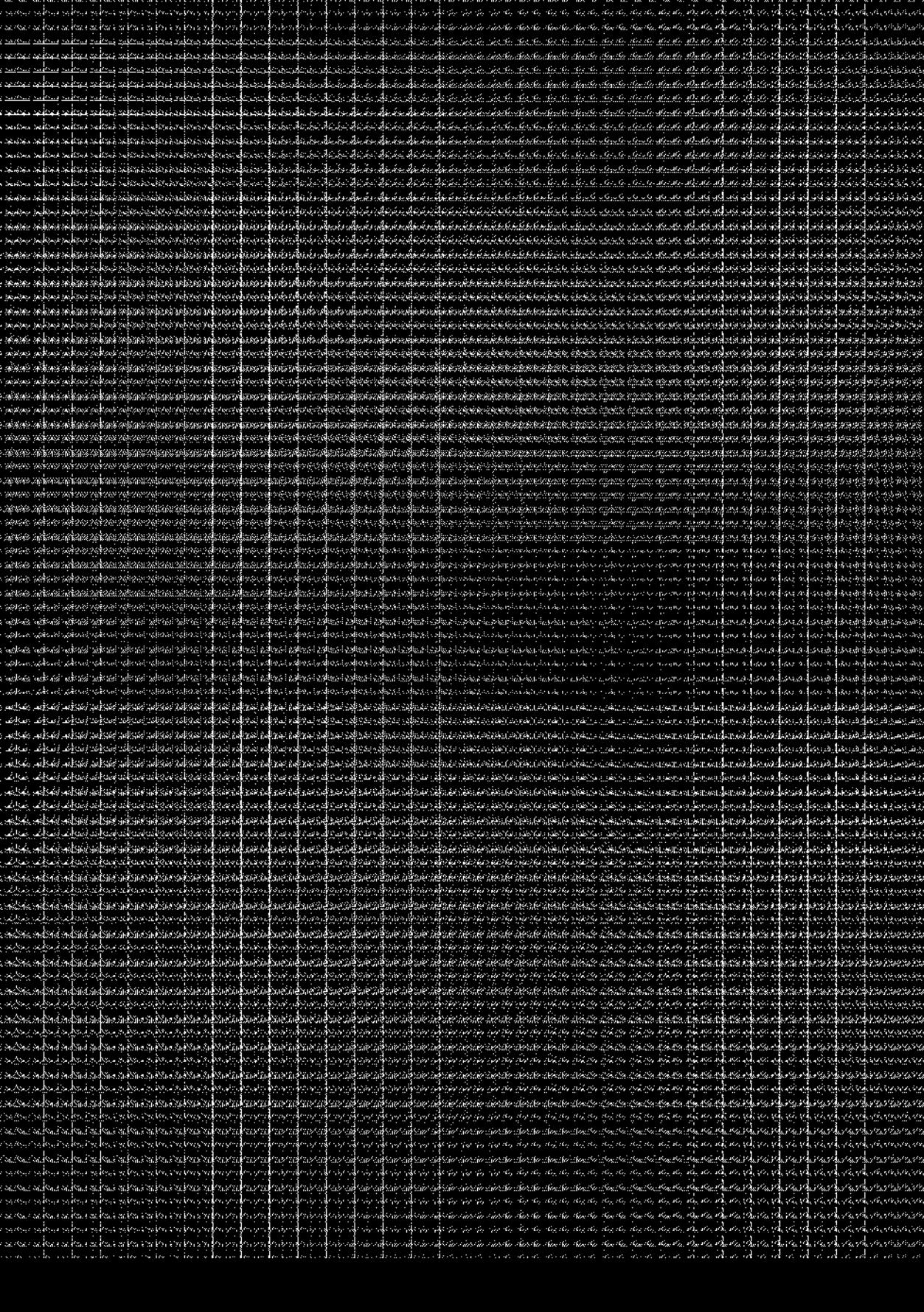
```
22 00 F9 C0 00 00 00 02
```

请回答以下问题：

- 它是什么类型？
 - 它的检验和是什么？
 - 记录数是多少？
17. 在图 12.18 中，试找出路由器 R2, R3 和 R4 的单播路由表，并给出最短路径树。
18. 在图 12.19 中，试给出路由器 R2, R3 和 R4 的多播路由表。
19. 一个使用了 DVMRP 的路由器从接口 2 收到一个源地址为 10.14.17.2 的分组。如果这个路由器转发该分组，那么在它的单播路由表中与这个地址相关的内容是什么？
20. 路由器 A 向路由器 B 发送了一个单播 RIP 更新分组告诉它 134.23.0.0/16 的距离有 7 跳之远。而路由器 B 向路由器 A 发送了一个更新分组则说 134.23.0.0/16 的距离有 4 跳之远。如果这两个路由器连接在同一个网络上，其中哪一个路由器是特定父路由器？
- RPF 有没有真正产生一个最短路径树？为什么？
 - RPB 有没有真正产生一个最短路径树？为什么？这时的树叶是什么？
 - RPM 有没有真正产生一个最短路径树？为什么？这时的树叶是什么？

12.10.2 研究活动

- 用 netstat 查找出你的服务器是否支持多播地址？
- 试找出 DVMRP 剪枝报文的格式。嫁接报文的格式又是什么？
- 对于 MOSPF，使一个网络和一个组相关联的组成员关系 LSA 分组的格式是什么？
- CBT 使用了九种类型的分组。试利用因特网找出每一种分组格式和目的。
- 试利用因特网找出 CBT 报文是怎样封装的。
- 试利用因特网找出有关我们讨论过的每一种多播路由选择协议的可扩展性的信息。用表的形式对它们进行比较。



第三部分

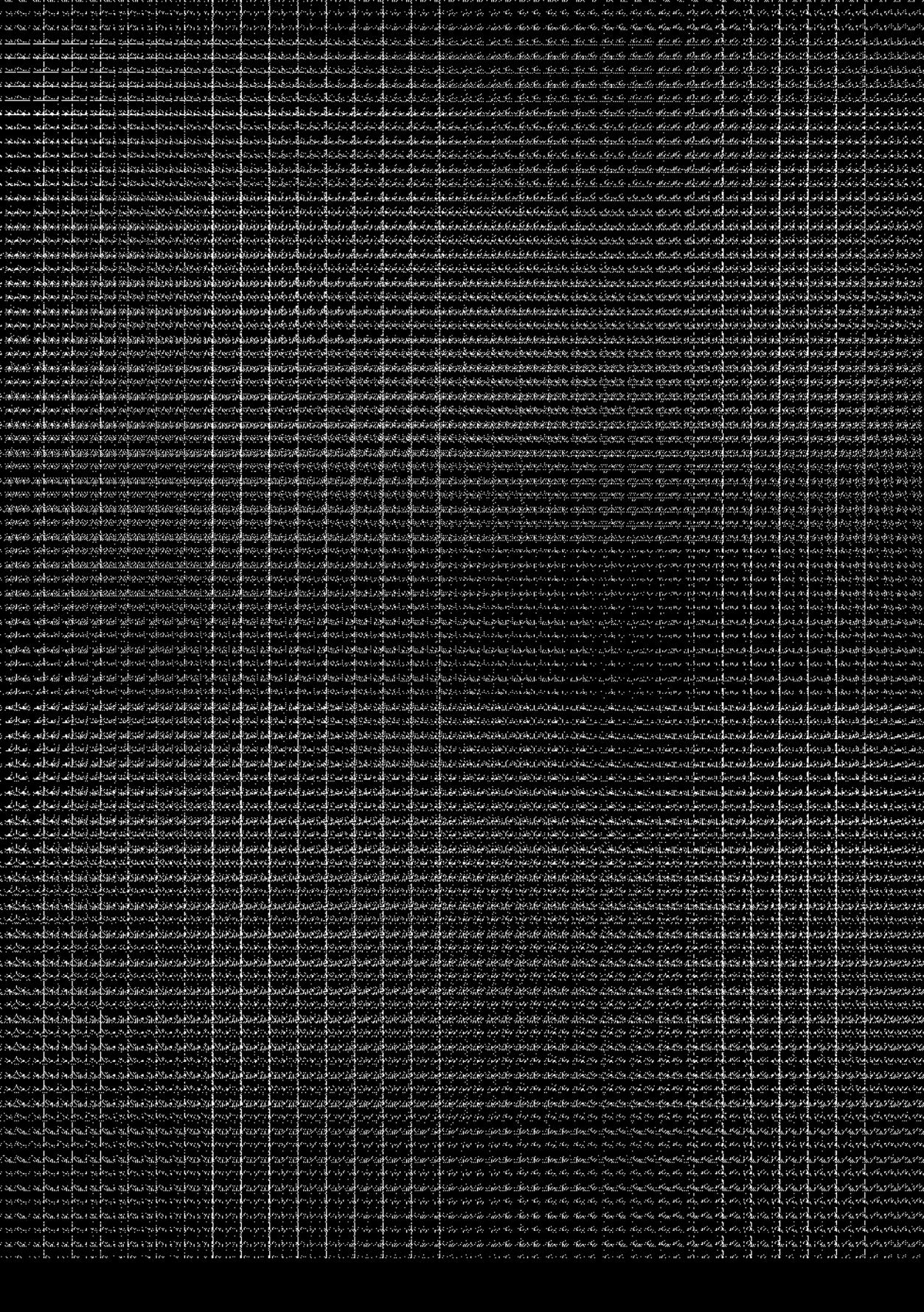
运 输 层

第 13 章 运输层简介

第 14 章 用户数据报协议（ UDP ）

第 15 章 传输控制协议（ TCP ）

第 16 章 流控制传输协议（ SCTP ）



第 13 章 运输层简介

本章讨论运输层协议能够提供的总体服务以及与这些服务相关的内容。本章还要描述针对不同状况而设计的一些通用运输层协议的行为。在后面几章的内容中，我们将会看到这些通用协议如何组合以建立诸如 UDP、TCP 以及 SCTP 这样的 TCP/IP 协议族运输层协议。

目标

本章有以下几个目标：

- 定义运输层的进程到进程的通信，并与网络层的主机到主机的通信进行比较。
- 讨论运输层的编址机制，讨论端口号，定义不同用途的端口号范围的划分。
- 解释运输层的分组化问题：报文的封装和解封。
- 讨论运输层提供的复用（多到一）和分用（一到多）服务。
- 讨论流量控制以及它在运输层是如何实现的。
- 讨论拥塞控制以及它在运输层是如何实现的。
- 讨论运输层的无连接服务和面向连接的服务，并通过有限状态机（FSM）来说明它们的实现。
- 讨论四个通用运输层协议的行为以及它们的应用：简单协议、停止等待协议、返回 N 协议和选择重传协议。
- 描述运输层利用捎带方式实现双向通信的思想。

13.1 运输层服务

正如我们在第 2 章中讨论的，**运输层位于网络层和应用层之间**。运输层负责向应用层提供服务，同时它接受来自网络层的服务。在本小节中，我们要讨论运输层能够提供的服务，而在下一小节，我们将讨论几个运输层协议背后的原理。

13.1.1 进程到进程的通信

运输层协议的首要任务是提供进程到进程的通信（process-to-process communication）。进程是指使用了运输层服务的一个应用层实体（正在运行的程序）。在我们讨论进程到进程的通信是怎样实现的之前，先要了解一下主机到主机的通信和进程

到进程的通信之间的区别。

网络层负责计算机级的通信（主机到主机的通信）。网络层协议只能把报文交付给目的计算机，但这还不算是完整的交付，报文必须要交付到正确的进程，而这正是运输层协议所要做的事。运输层协议负责把报文交付给合适的进程。图 13.1 所示为网络层和运输层的作用范围。

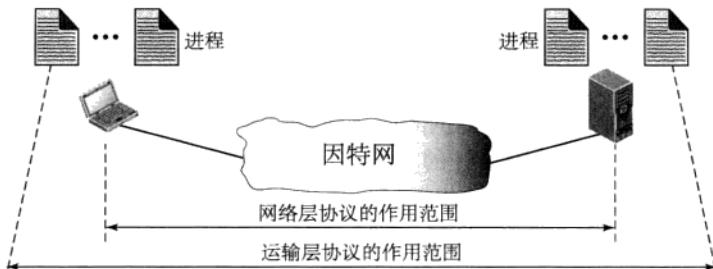


图 13.1 网络层与运输层的比较

13.1.2 编址：端口号

虽然要完成进程到进程的通信可以有很多方法，但最常用的方法是通过客户-服务器范式（client-server paradigm）（参见第 17 章）。位于本地主机上的进程称为客户，它通常需要远程主机上的一个称为服务器的进程所提供的服务。

这两个进程（客户和服务器）应当具有相同的名称。例如，若要获取远程主机上的日期和时间，我们需要在本地主机上运行一个叫做 Daytime 的客户进程，同时在远程主机上也运行着一个称为 Daytime 的服务器进程。

但是，今天的操作系统都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刻可运行多个服务器程序，正如本地计算机可在同一时刻运行一个或多个客户程序一样。对通信来说，我们必须定义：

- 本地主机
- 本地进程
- 远程主机
- 远程进程

本地主机和远程主机都是通过 IP 地址来定义的。要定义进程，我们需要第二种标识，称为端口号（port numbers）。在 TCP/IP 协议族中，端口号是 0~65535 之间的整数。

客户程序通过一个端口号来定义自己，这种端口号称为临时端口号（ephemeral port number）。“临时”这个词的含义是指生存时间较短，之所以用这个词是因为客户的生存时间通常都比较短。建议为临时端口号取一个大于 1023 的整数，这样可使一些客户/服务器程序能正常工作。

服务器进程也必须用一个端口号来定义自己。但是这个端口号不能随机选取。如果在服务器端的计算机上运行的服务器程序被指派了一个随机数作为其端口号，那么客户端的进程就是想要访问这个服务器并使用它的服务也无法掌握它的端口号。当然，一种可能的

解决办法是发送一个特别的分组，以请求得到某个特定服务器的端口号，但这就需要更多的额外开销。TCP/IP 决定让服务器使用全球通用端口号，它们称为熟知端口号（well-known port numbers）。对这个规则也有一些例外情况。例如，有的客户也被指派了熟知端口号。每个客户进程都知道相应的服务器进程的熟知端口号。例如，上面讨论过的 Daytime 客户进程可使用临时端口号 52000 来标识自己，但 Daytime 服务器进程则必须使用熟知（永久的）端口号 13。图 13.2 描绘了这一概念。

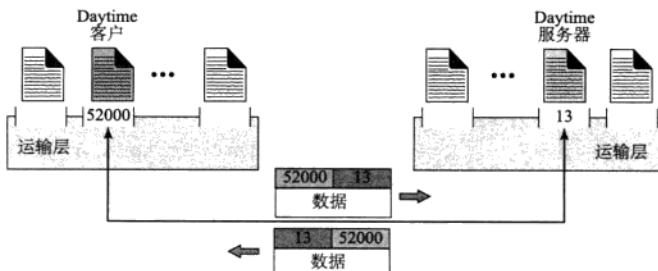


图 13.2 端口号

现在应当很清楚了，在选择数据最后的终点时，IP 地址和端口号起着不同的作用。
目的 IP 地址定义了全世界无数台主机之中的某一台主机，而当这台主机被选定后，端口号则定义了在这台特定主机上运行的多个进程之中的某一个（见图 13.3）。

ICANN 定义的范围段

ICANN 把端口号划分为三个范围段：熟知的、注册的和动态（或专用）的，如图 13.4 所示。

- 熟知端口 范围从 0~1023 的端口由 ICANN 指派和控制，这些都是熟知端口。
- 注册端口 范围从 1024~49151 的端口，ICANN 既不指派也不控制，但它们必须在 ICANN 处注册以防止重复。

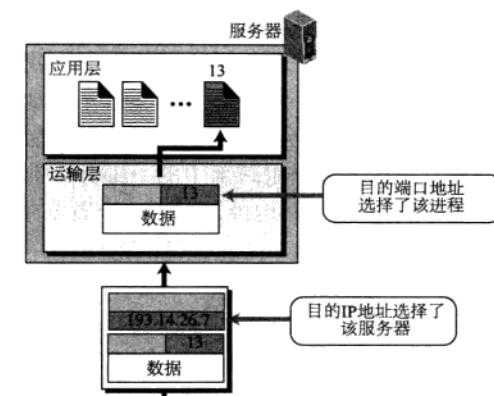


图 13.3 IP 地址与端口号之对比

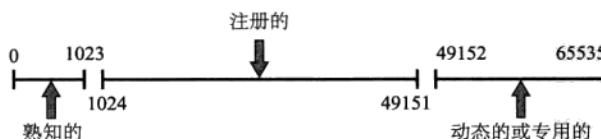


图 13.4 ICANN 定义的范围段

- 动态端口 范围从 49152~65535 的端口既不用指派也不用注册。它们可被用作临时的或专用的端口号。最初的建议是客户使用的临时端口号应当在这个范围之内选择。但是，大多数系统并没有遵循这个建议。

熟知端口号小于 1024。

例 13.1

在 UNIX 中，熟知端口被保存在/etc/services 文件中。这个文件中的每一行给出了一个服务器名及其熟知端口号。我们可以通过实用程序 grep 把我们想要的应用所对应的那一行提取出来。下面给出了 TFTP 的端口。请注意，TFTP 在 UDP 和 TCP 中都可以使用端口 69。SNMP（参见 24 章）使用了两个端口号（161 和 162），各有不同的用途。

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

$grep snmp /etc/services
snmp   161/tcp      #Simple Net Mgmt Proto
snmp   161/udp      #Simple Net Mgmt Proto
snmp   162/udp      #Traps for SNMP
```

套接字地址

TCP 协议族的运输层协议在建立连接时，需要在连接的两端同时使用 IP 地址和端口号。

IP 地址和端口号的组合就称为套接字地址 (socket address)。客户套接字地址唯一地定义了一个客户进程，正如服务器套接字地址唯一地定义了一个服务器进程一样（见图 13.5）。

要在因特网中使用运输层的服务，我们需要有一对套接字地址：客户套接字地址和服务器套接字地址。这四个信息是网络层分组首部和运输层分组首部中的一部分。前一个首部中包含了 IP 地址，后一个首部中则包含了端口号。

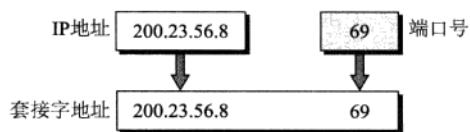


图 13.5 套接字地址

13.1.3 封装和解封

为了把报文从一个进程发送到另一个进程，运输层协议要对报文进行封装和解封（参见图 13.6）。

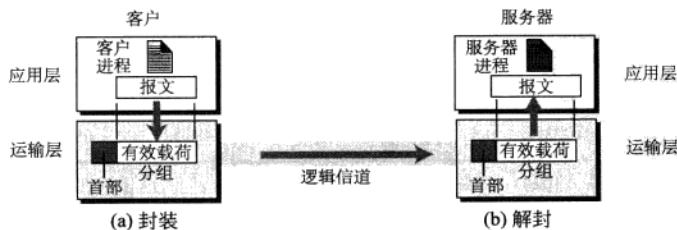


图 13.6 封装和解封

封装发生在发送方。当进程想要发送一个报文时，就把这个报文递交给运输层，随之一起递交的还包括一对套接字地址和其他一些信息，其具体内容取决于运输层协议。运输层接收数据并为之添加运输层首部。在因特网中，运输层的分组也称为用户数据报、数据

段或分组。在本章我们称之为分组。

解封发生在接收方。当报文到达目的运输层后，头部被拆除，然后运输层将报文交付给在应用层运行的进程。如果接收方需要对收到的报文进行响应，那么发送方的套接字地址也要一起递交给该进程。

13.1.4 复用和分用

当一个实体接受来自多个源的输入时，就称为复用（multiplexing）（多到一），而当一个实体将数据交付到多个源时，就称为分用（demultiplexing）（一到多）。源点的运输层执行的是复用，而终点的运输层执行的是分用（参见图 13.7）。

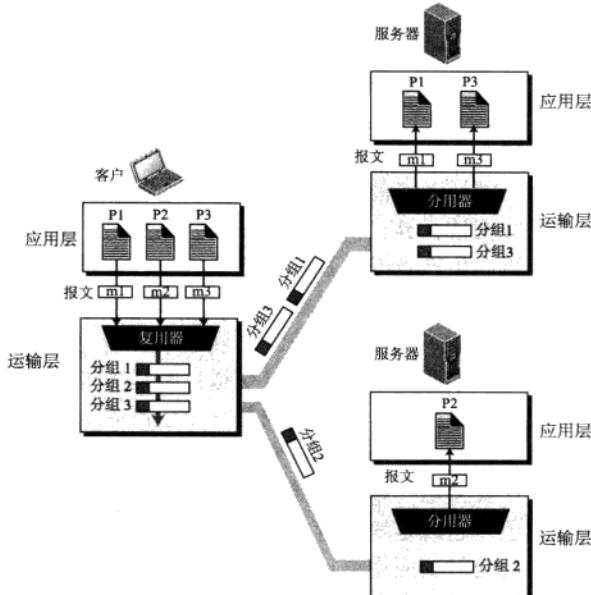


图 13.7 复用和分用

图 13.7 描绘了一个客户和两个服务器之间的通信。在客户端运行着三个进程：P1、P2 和 P3。客户进程 P1 和 P3 需要向某个服务器上运行的相应服务器进程发送请求，而客户进程 P2 则需要向另一台服务器上运行的相应服务器进程发送请求。客户端的运输层从这三个进程处接收到三个报文，并产生了三个分组，因此它起到复用器的作用。分组 1 和分组 3 使用相同的逻辑信道到达第一个服务器的运输层。当它们到达该服务器后，运输层就担负起分用器的作用，并将报文分别交付给两个不同的进程。在第二个服务器上的运输层接收到分组 2 并将它交付给相应的进程。

13.1.5 流量控制

当一个实体产生数据而另一个实体消耗这些数据时，数据的产生速度和消耗速度之间

应当达到某种平衡。如果数据产生的速度比消耗的速度快，那么消耗方就会因来不及处理而被迫丢弃一些数据。如果数据产生的速度比消耗的速度慢，消耗方就需要等待，从而使整个系统的效率降低。流量控制与前面所提到的第一个问题有关。我们需要防止的是发生在消耗端的数据丢失。

推送或拉取

从生产者到消费者的交付可以通过两种方式完成：推送或者拉取。如果发送方只要数据一产生就发送出去，不管消耗方之前是否请求过这些数据，这样的交付方式就称为推送。如果发送方只有在消耗方请求之后才发送这些数据，那么这样的交付方式就称为拉取。图 13.8 所示为这两种类型的交付。

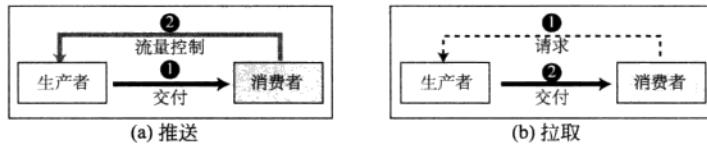


图 13.8 推送或拉取

当生产者推送数据时，消耗者可能会因数据太多而来不及处理，所以就需要在反方向上进行流量控制，以防止数据被丢弃。换言之，消耗者需要警告生产者停止发送数据，并在它再次准备好接收时，再向生产者发出通知。当消耗者拉取数据时，它是在自己准备好的情况下才会提出请求，因而此时不需要流量控制。

运输层的流量控制

在运输层的通信中，我们需要与四个实体打交道：发送方进程、发送方运输层、接收方运输层和接收方进程。应用层的发送进程仅仅是一个生产者。它产生报文块并将其推送到运输层。发送方的运输层要扮演两个角色：它既是消耗者也是生产者。它要消耗被生产者推送过来的报文，同时还要将这些报文封装成分组，并推送给接收方的运输层。接收方的运输层也要扮演两个角色：它既是消耗者，因为它要消耗来自发送方的报文，也是生产者，因为它要将报文解封后交付到应用层。不过，最后的交付通常是一种拉取交付，也就是说运输层会等待应用层的进程过来索取这些报文。

从图 13.9 中可以看出我们至少需要两种情况下的流量控制：从发送方的运输层到发送方的应用层以及从接收方的运输层到发送方的运输层。

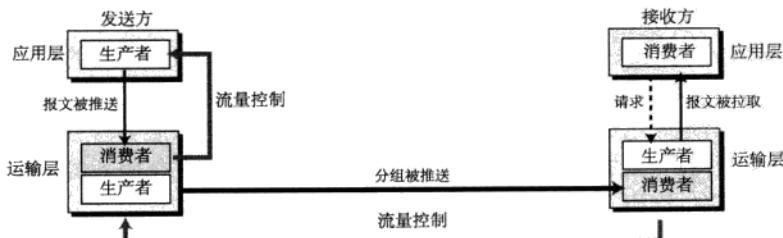


图 13.9 运输层的流量控制

缓存

虽然流量控制可以用多种方法实现，但最常见的一种解决方法是使用两个缓存。一个

位于发送方的运输层，而另一个位于接收方的运输层。此处的缓存是指可以在发送方和接收方用来保存分组的一组存储器。而对于流量控制的通信则可以通过从消耗者向生产者发送信号来实现。

当发送方的运输层缓存满溢时，它就通知应用层停止传递报文块，而当它又有了空位置时，就通知应用层可以再次传递报文块了。

当接收方的运输层缓存满溢时，它就通知发送方的运输层停止发送分组，而当它又有了空位置时，就通知发送方的运输层可以再次发送分组了。

例 13.2

前面的讨论要求消耗者和生产者在以下两种情况都要进行通信联系：当缓存满溢时和当缓存又有了空位置时。如果使用的缓存只有一个位置，那么通信就会变得很简单。假设这两个运输层都使用了只能存放一个分组的存储空间。当发送方运输层中的这个存储位置空闲时，发送方运输层就提醒应用层可以发送下一个报文块。当接收方运输层中的这个存储位置空闲时，它就向发送方的运输层发送一个确认，使之发送下一个分组。正如我们在后面将会看到的，像这种在发送方和接收方都只使用了一个存储位置的流量控制的效率是非常低的。

13.1.6 差错控制

在因特网中，因为负责从发送方运输层向接收方运输层运送分组的网络层（IP）是不可靠的，所以，如果应用程序要求可靠性，那么我们就必须让运输层变得可靠。可以通过在运输层中加入差错控制来实现其可靠性。运输层的差错控制负责：

1. 检测并丢弃损坏的分组。
2. 跟踪丢失和丢弃的分组并重传它们。
3. 识别重复的分组并丢弃它们。
4. 保存失序到达的分组，直至缺失的分组全部抵达。

差错控制不像流量控制，它只涉及到发送方的运输层和接收方的运输层，因为我们假设在应用层和运输层之间的报文块交换是无差错的。图 13.10 描绘了发送方和接收方的运输层之间的差错控制。与流量控制一样，在大多数时间都是由接收方的运输层来管理差错控制，它会通知发送方的运输层出现了什么问题。

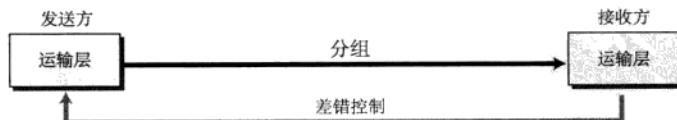


图 13.10 运输层的差错控制

序号

差错控制要求发送方的运输层知道哪个分组应当被重传，而接收方的运输层则要了解哪个分组是重复的，或者哪个分组失序到达。如果给分组编了号，那么就能够做到这一点。我们可以在运输层分组的首部中增加一个字段，以保存分组的序号（sequence

number)。当一个分组损坏或丢失时，接收方的运输层通过这个序号就能够以某种方式通知发送方的运输层重传该分组。同时，如果接收到的两个分组具有相同的序号，接收方的运输层也能够检测出这是两个重复的分组。通过观察序号的连续性还能识别出失序到达的分组。

分组被顺序地编号。不过，由于我们需要把这个序号包含在每个分组的首部中，因而不得不加以限制。如果分组的首部中有 m 位用于序号，那么这个序号的范围就是从 0 到 $2^m - 1$ 。例如，如果 m 是 4，那么序号就只能是在 0 到 15 之间。但是我们可以回过头来重新使用序号，因此在这种情况下的序号就是

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，其中 m 是序号字段的位数。

确认

我们可以用正信号或者负信号来实现差错控制。只要一个或一组分组安全且完好地到达了，那么接收方就可以为它们发送一个确认 (ACK)。另一方面，接收方可以简单地丢弃损坏的分组，发送方如果使用了计时器，就能够检测出这些丢失的分组。在发送分组时，发送方要启动一个计时器，如果该计时器到期了，而发送方还没有收到相应的 ACK，那么它就要重传这个分组。重复的分组可以被接收方悄无声息地丢弃掉，而失序到达的分组可以被丢弃（对发送方来说就好像是这个分组丢失了），也可以被保存起来，直到缺失的分组全部抵达为止。

13.1.7 流量控制和差错控制的组合

我们曾经讨论过，流量控制要求使用两个缓存，一个在发送方，另一个在接收方。我们也说过，差错控制要求双方使用序号和确认号。如果我们使用了两个带编号的缓存，一个在发送方，另一个在接收方，那么这两个需求就可以合二为一。

在发送方，当为分组的发送而做准备时，可以使用缓存中下一个空位的编号 x 作为该分组的序号。当该分组被发送时，它的一个副本就保存在该存储器中的 x 位置上，直至收到来自另一端的确认。当与已发送分组相关的确认到达时，存储器中该分组的副本就被清除，从而使该存储位置变为空闲。

在接收方，当序号 y 的分组到达时，就将其保存在存储器的 y 位置上，直到应用层准备好接收它，同时可以发送一个确认以声明分组 y 已经到达。

滑动窗口

因为序号是模 2^m 的，所以从 0 至 $2^m - 1$ 的序号可以表示为一个环（图 13.11）。

缓存被表示为一组小格子，称为滑动窗口 (sliding window)，在任何时候它们都占据了圆的一部分。在发送方，当一个分组被发送出去，相应的小片就被标记。当所有的小片都被标记后，就表示缓存已满，不允许从应用层接收更多的报文。当一个确认到达后，相应小片的标记就被取消。如果这个窗口前端连续多个小片都没有标记，那么这个窗口就向后滑动到相应的序号范围上，以允许窗口后端有更多空闲的小片。图 13.11 描绘了发送方

的滑动窗口。序号是模 16 ($m = 4$) 的，且窗口大小为 7。请注意，滑动窗口只不过是一种抽象，而实际上它是通过计算机变量来保存下一个将要发送的分组的编号以及刚刚发送出去的最后一个分组的编号来实现的。

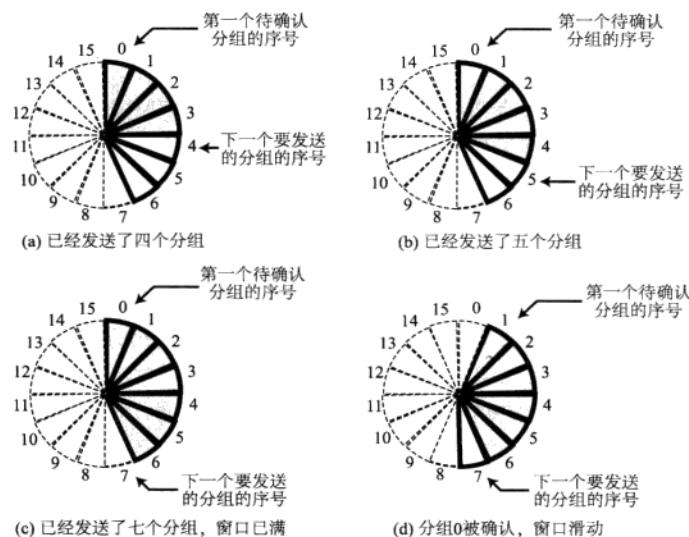


图 13.11 用圆形表示的滑动窗口

大多数协议使用条形来表示滑动窗口。其思想完全一样，只不过通常这样做可以节约纸张空间。图 13.12 所示为条形表示法。这两张图告诉我们的是一回事。如果我们捏住图 13.12 中各部分的两端分别向上弯曲，就可以得到如图 13.11 一模一样的图了。



图 13.12 用条形表示的滑动窗口

13.1.8 拥塞控制

因特网中存在的一个重要的问题是**拥塞** (congestion)。如果一个网络中的**负载** (load) (也就是发送到网络上的分组数量) 大于网络的容量 (也就是网络能够处理的分组数量)，这个网络就有可能发生拥塞。**拥塞控制** (congestion control) 指的是用来控制拥塞，以使负载保持低于容量的机制和技术。

我们可能会问为什么网络中会出现拥塞？只要是涉及到等待的系统都会发生拥塞。例如，任何因车流中发生了意外情况而导致的堵车都是高速公路上出现拥塞的原因，比如说在上下班高峰时间发生了车祸。

网络或者互联网中的拥塞是由于路由器或交换机中有一些队列，也就是在分组被处理之前或之后用来保存这些分组的缓存。分组被放入合适的外出队列并等待轮到自己被发送。这些队列空间是有限的，因此到达路由器的分组数量有可能会大于路由器能够保存的分组数量。

拥塞控制指的是能够在拥塞发生之前预防它，或者在拥塞发生之后消除它的技术和机制。

开环拥塞控制

在拥塞发生之前使用一些策略来预防拥塞称为开环拥塞控制（open-loop congestion control）。在此类机制中，拥塞控制可以由源点也可以由终点来处理。

重传策略 有时重传是不可避免的。如果发送方觉得一个发送出去的分组丢失或损坏了，那么这个分组就需要重传。一般来说，重传可能会加重网络的拥塞。但是，好的重传策略可以预防拥塞。这就需要将重传策略以及重传计时器设计得能够优化效率，同时还能够预防拥塞。

窗口策略 发送方窗口的类型也可能会影响到拥塞。我们将在本章稍后看到对于拥塞控制来说选择重传窗口就要比返回 N 窗口好。

确认策略 由接收方实施的确认策略也有可能会影响拥塞。如果接收方不是对每一个收到的分组都进行确认，那么也能够使发送方放慢发送速率，因而有助于预防拥塞。在这种情况下可以使用几种方式。接收方可以仅仅在自己有分组要发送时，或者在一个特殊的计时器到期时才发送一个确认。接收方也可以决定一次为 N 个分组发送确认。要知道这些确认也是网络负担的一部分。发送的确认越少，给网络带来的负担也越轻。

闭环拥塞控制

闭环拥塞控制（closed-loop congestion control）机制试图在拥塞发生后缓解拥塞的程度。有几个此类机制已经被用在不同的协议中。我们要描述的是在运输层中使用的机制。发送方的窗口大小可以是灵活的。决定发送方窗口大小的一个因素就是因特网的拥塞状况。发送方的运输层可以通过观察分组的丢失情况来监视因特网的拥塞状况，如果拥塞状况恶化，就用一种策略来减小窗口大小，反之亦然。我们将在第 15 章中看到 TCP 是如何使用这种策略来控制它的窗口大小的。

13.1.9 无连接的和面向连接的服务

像网络层协议一样，运输层协议也可以提供两种类型的服务：无连接的服务和面向连接的服务。不过这两种服务在本质上与网络层的不同。在网络层，无连接的服务意味着属于同一报文的多个数据报采取了不同的路径。而在运输层，我们并不关心分组采取什么物理路径（我们假设在两个运输层之间有一条逻辑连接），运输层的无连接服务表示分组和分组之间是互相独立的，而面向连接的服务则意味着它们相互之间有联系。让我们来详细说明这两种服务。

无连接的服务

在使用无连接的服务时，源进程（应用程序）需要将它的报文分割成大小可被运输层接受的数据块，并将这些数据块逐个地交付到运输层。运输层视这些数据块为独立的数据单元，块和块之间没有联系。当来自应用层的一个数据块到达后，运输层就把它封装到一个分组中，然后发送这个分组。为了说明分组之间的独立性，假设客户进程有三个数据块

要发送给服务器进程。这些数据块按顺序递交给无连接的运输协议。但是，因为在运输层这些分组之间是没有联系的，所以分组就有可能失序到达终点，并且同样失序地交付给服务器进程。在图 13.13 中，我们用时间线来说明这些分组的运动，但我们假设从进程到运输层的交付以及从运输层到进程的交付是即时的。

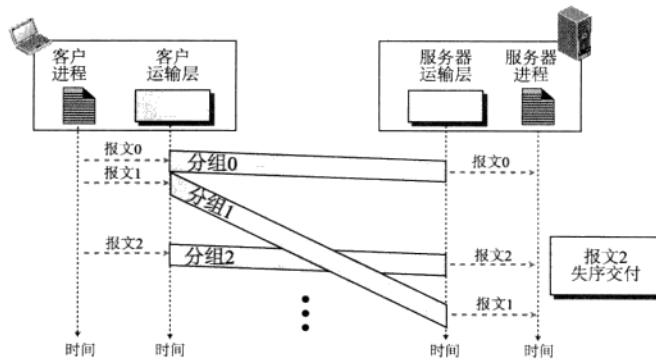


图 13.13 无连接服务

如图所示，在客户端，有三个报文块被按序（1、2 和 3）递交给客户运输层。因为第二个分组在传输时有额外的时延，所以在服务器端，这些报文并没有按序交付（1、3 和 2）。如果这三个数据块属于同一个报文，那么这个服务器收到的就有可能是一个奇怪的报文。

如果其中的一个报文丢失了，情况会更加糟糕。因为分组上没有编号，接收方运输层无法知道有一个分组已经丢失了。它只能向服务器进程交付两个数据块。

之所以会出现以上的问题，是因为这两个运输层彼此之间没有协调一致。接收方运输层既不知道什么时候第一个分组到达，也不知道什么时候所有的分组全部到齐。

我们可以说在无连接服务的情况下，没有哪种流量控制、差错控制或拥塞控制策略能够有效实施。

面向连接的服务

在使用面向连接的服务时，客户和服务器首先要在它们之间建立一条连接。数据的交换只能是在连接建立之后发生。在数据交换完成后，该连接需要被拆除（图 13.14）。正如我们在前面提到的，运输层面向连接的服务与网络层面向连接的服务有所不同。在网络层，面向连接的服务意味着两端的主机及其之间的路由器之间的协调一致。而在运输层，面向连接的服务仅涉及两个主机，也就是说服务是端对端的。这就意味着，不管是无连接的网络层协议，还是面向连接的网络层协议，我们都能在上面使用面向连接的运输层协议。图 13.14 描绘的是运输层的面向连接服务的连接建立、数据传送以及拆除连接这几个阶段。请注意，大多数协议将连接建立阶段中的第三个和第四个分组合并成一个分组，我们将会在第 15 章和第 16 章中了解到。

我们能够在面向连接的协议中实施流量控制、差错控制和拥塞控制。

有限状态机

不管是提供无连接的服务，还是面向连接的服务，一个运输层协议的行为可以很好地用一个有限状态机（finite state machine, FSM）来表示。图 13.15 就是用 FSM 表示的运输层协议。使用这个工具，各运输层（发送方或接收方）都可以被认为是一个具有有限状态

的机器。有限状态机总是处在某一种状态中，直到有一个事件发生。对每个事件的响应有两种：定义要执行的动作列表（可能为空），并决定下一个状态（也可能与当前状态相同）。在这些状态之中必须有一个是开始状态，也就是这个有限状态机一开机就进入的状态。在图中我们用圆角框表示状态，用黑体字（带颜色的文本）表示事件，用普通字体表示动作。开始状态有一个不来自其他状态的进入箭头。一条水平线用于分隔事件和动作，而在第 15 章中我们将用斜线来代替水平线。箭头方向表示向下一个状态的运动方向。

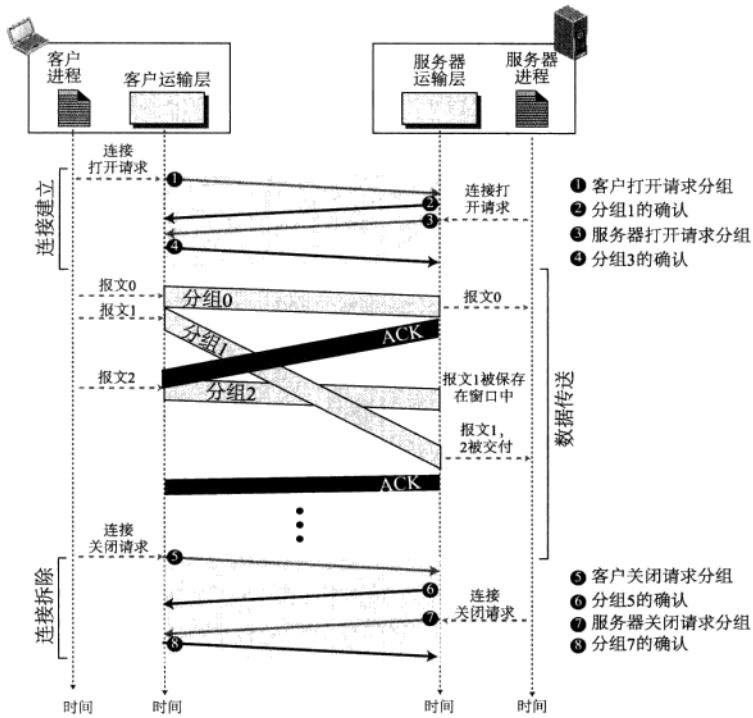


图 13.14 面向连接的服务

我们可以把无连接的运输层视为只有一个状态的 FSM。这个有限状态机的两端（客户和服务器）永远处于建立的状态，总是准备好发送和接收运输层的分组。

另一方面，在面向连接的运输层 FSM 中，到达建立的状态之前需要经过三个状态。而在关闭连接之前也需要经过三个状态。在没有连接时，有限状态机一直处于 Closed 状态。状态机保持此状态直至本地进程的打开连接请求到达，于是状态机就向远程运输层发送一个打开请求分组，并且进入 Open-wait-I 状态中。当收到来自另一端的确认时，本地 FSM 进入 Open-wait-II 状态。当状态机在此状态时，一条单向连接就已经建立好了，但是如果需要双向连接，那么状态机还要在这个状态继续等待，直至另一端也请求了连接。当收到这个请求后，状态机发送确认并进入 Established 状态。

当双方都处在建立的状态时就可以互相交换数据以及对数据的确认了。但是我们要记住，不管是在无连接的还是面向连接的运输层协议中，建立的状态本身就代表了一组数据

传送状态，我们将在下一节讨论运输层协议时讨论到。

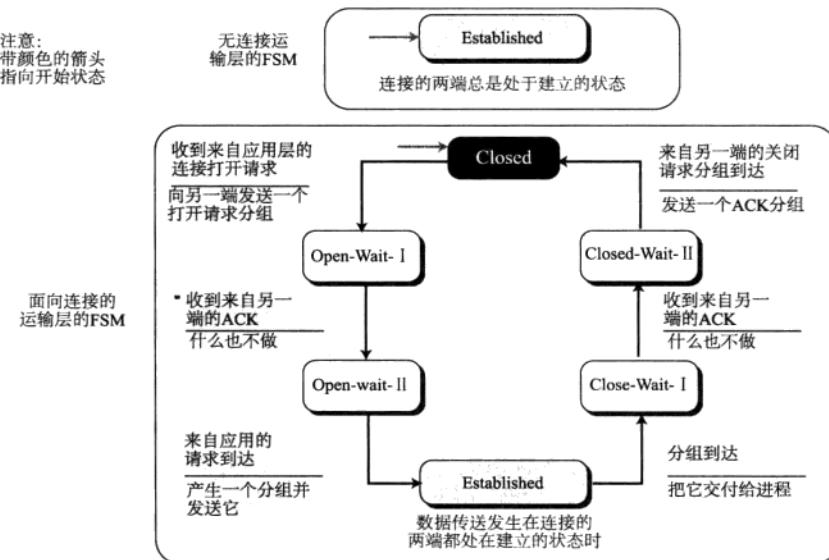


图 13.15 用 FSM 来表示无连接的服务和面向连接的服务

要拆除一个连接，应用层向本地运输层发送关闭请求报文。运输层就向另一端发送关闭请求分组，并进入 Close-wait-I 状态。收到来自另一端的确认后，机器进入 Close-wait-II 状态，并等待对方发来的关闭请求分组。当这个分组到达后，状态机发送一个确认并进入 Closed 状态。

面向连接的 FSM 有几种不同的变体，我们将在第 15 章和第 16 章中再讨论。在第 15 和 16 章中，我们还要讨论如何压缩或扩展这个 FSM，且状态的名称也会有所变化。

13.2 运输层协议

把我们前面所描述的一系列服务组合起来就能够创建一个运输层协议。为了更好地理这些协议的行为，我们先从最简单的协议开始谈起，再逐渐增加复杂度。TCP/IP 协议使用的运输层协议或者是对这些协议之一的改进，或者是它们的组合。这也是我们在本章要讨论这些协议的原因，就是为更好地理解接下来三章内容所介绍的比较复杂的协议铺平道路。为了简化讨论，对于所有这些协议，我们一开始都认为是数据分组只向一个方向传送的单向协议（即单工的）。在本章结尾，我们会简单地讨论如何将它们变成数据分组可以在两个方向同时传送的双向协议（即全双工）。

13.2.1 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即时处理它收到的任何分组。换言之，接收方永远不会因为涌入的分组太多而处理不过来。图 13.16 所示为这个协议的概要图。

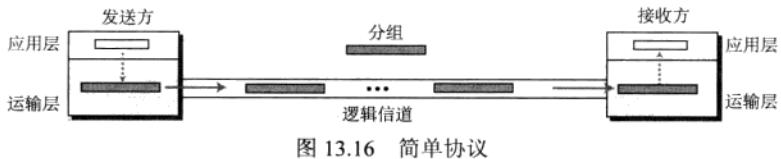


图 13.16 简单协议

发送方的运输层从应用层那里得到一个报文，用它产生一个分组，然后发送这个分组。接收方的运输层从网络层那里收到一个分组，从分组中提取出该报文，然后将该报文交付给应用层。发送方和接收方的运输层为它们各自的应用层提供传输服务。

FSM

在发送方的应用层有报文要发送之前，发送方不可能发送分组。在分组到达之前，接收方也不可能向应用层交付报文。我们可以用两个 FSM 来表示这些要求。每个 FSM 只有一个状态，即准备好状态。发送方状态机一直保持在准备好状态，直至应用层的进程有一个请求到来。当这个事件发生后，发送方状态机将报文封装成一个分组，并把这个分组发送给接收方状态机。接收方状态机也总是保持在准备好状态，直至来自发送方状态机的分组到达。当这个事件发生后，接收方状态机把收到的分组解封后提取出该报文，并把它交付给应用层的相应进程。图 13.17 所示为这个简单协议的 FSM。我们将在第 14 章中看到，UDP 协议就在这个协议的基础上做了一些改进。

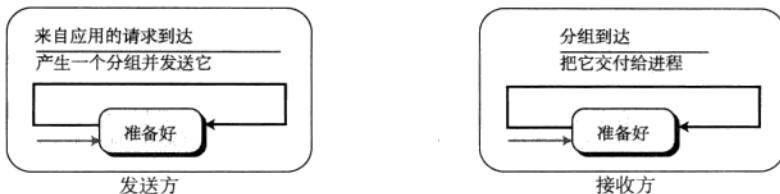


图 13.17 简单协议的 FSM

简单协议是一个无连接协议，既没有流量控制，也没有差错控制。

例 13.3

图 13.18 所示为使用这个协议进行通信的一个例子。它非常简单。发送方一个接一个地发送分组，甚至根本不用考虑接收方。

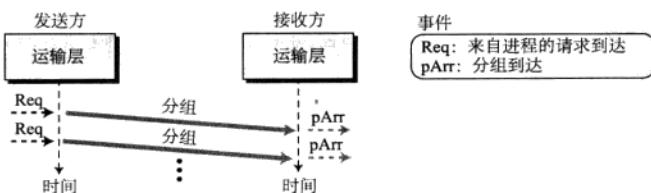


图 13.18 例 13.3 的流程图

13.2.2 停止等待协议

发送窗口和接收窗口都为 1

我们的第二个协议是面向连接的协议，称为停止等待协议（Stop-and-Wait protocol），它使用了流量控制和差错控制。发送方和接收方都使用了大小为 1 的滑动窗口。发送方一次发送一个分组，然后在发送下一个分组之前要先等待一个确认。为了检测损坏的分组，我们需要在每个数据分组中增加一个检验和，并在分组到达接收方后检查它。如果检验和不正确，就说明分组损坏了，这个分组被悄无声息地丢弃掉。接收方的沉默对发送方来说就是一个信号，说明分组不是损坏了就是丢失了。发送方在每发送一个分组时要启动一个计时器。如果在计时器到期之前确认到达了，计时器就停止计时，发送方继续发送下一个分组（如果它还有分组要发送的话）。如果该计时器超时，那么发送方就假定分组丢失或者损坏了，因而重传前一个分组。这就意味着发送方需要保存分组的副本直至它的确认到达为止。图 13.19 所示为停止等待协议的概要图。请注意，在任何时候信道中都只有一个分组和一个确认。

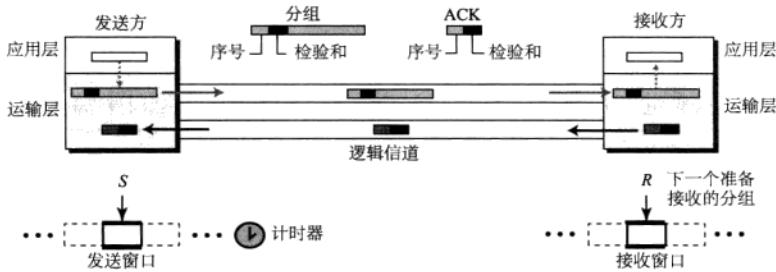


图 13.19 停止等待协议

停止等待协议是一种面向连接的协议，它提供了流量控制和差错控制。

在停止等待协议中，流量控制通过迫使发送方等待确认来实现，差错控制通过丢弃损坏的分组并让发送方在计时器超时后重传未确认的分组来实现。

序号

为了防止重复的分组，协议使用了序号和确认号。分组的首部要增加一个字段用来保存分组的序号。序号的范围是需要慎重考虑的。因为我们希望分组的长度最小化，所以我们寻求的是不会混淆通信的最小范围。让我们来推算一下所需的序号范围。假设我们使用 x 作为一个序号，那么此后只要用到 $x + 1$ 就够了，没有必要使用 $x + 2$ 。为了说明这个问题，我们假设发送方发送了一个序号为 x 的分组，可能会发生以下三种情况：

1. 分组安全完好地抵达接收方。接收方发送一个确认。这个确认到达发送方，使发送方继续发送下一个序号为 $x + 1$ 的分组。
2. 分组损坏或从未抵接收方。发送方在超时后重传这个分组（序号为 x ），接收方返回一个确认。
3. 分组安全完好地抵达接收方。接收方发送一个确认，但是这个确认损坏或丢失了。发送方在超时后重传这个分组（序号为 x ）。注意此时这个分组是重复的。接收方能够识别出这种重复，因为它希望接收的分组是 $x + 1$ ，但实际接收到的分组却是 x 。

我们可以看出序号 x 和 $x + 1$ 都是必要的，因为接收方需要用它们来区分情况 1 和情况 3。但是分组没有必要使用序号 $x + 2$ 。在情况 1 中，分组可以再次使用序号 x ，因为分组 x 和 $x + 1$ 都已经被确认了，发送方和接收方也没有什么弄不清楚的地方。在情况 2 和情况 3 中，新的分组是 $x + 1$ ，不是 $x + 2$ 。如果仅仅需要 x 和 $x + 1$ ，那么我们可以让 $x = 0$ 且 $x + 1 = 1$ 。也就是说该序号是 0, 1, 0, 1, 0...。这被称为模 2 运算（译注：通常记为 mod 2）。

在停止等待协议中，我们可以使用 1 位字段来为分组编号。这个序号基于模 2 运算。

确认号

因为序号必须既适合数据分组，又适合确认，所以我们有以下约定：确认号总是声明了接收方准备接收的下一个分组的序号。例如，如果分组 0 已经安全完好地抵达了，那么接收方就发送一个确认号为 1 的 ACK（表示下一个希望接收分组 1）。如果分组 1 安全完好地抵达了，那么接收方就发送一个确认号为 0 的 ACK（表示下一个希望接收分组 0）。

在停止等待协议中，确认号总是声明了模 2 运算的下一个希望接收的分组序号。

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空格。接收方也有一个控制变量，我们称之为 R (receiver)，它指向接收窗口中唯一的空格。

在停止等待协议中，所有的计算都是模 2 的。

FSM

图 13.20 描绘了停止等待协议的 FSM。因为这个协议是面向连接的，所以在交换数据分组前，连接的两端都应当处于建立的状态。我们在这里所描绘的状态实际上都是包含在建立的状态中的子状态。

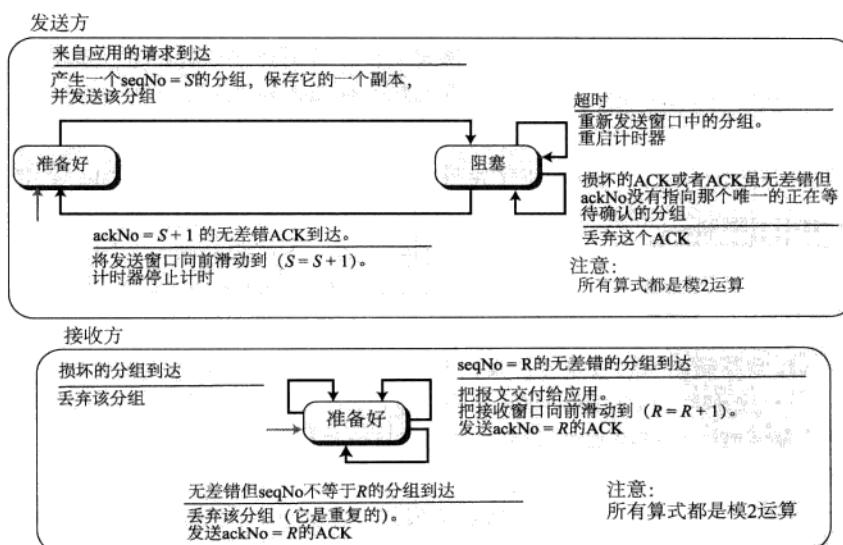


图 13.20 停止等待协议的 FSM

发送方 发送方在一开始时处于准备好状态，但是它可以在准备好和阻塞这两个状态之间变换。变量 S 的值初始化为 0。

□ 准备好状态 当发送方处于此状态时，它仅等待一个事件的发生。如果来自应用层的请求到达了，发送方就产生一个序号设为 S 的分组。这个分组的一个副本被保存起来，而这个分组被发送出去。发送方启动唯一的计时器，然后进入到阻塞状态。

□ 阻塞状态 当发送方处于此状态时，可能会发生的事件有三个：

1. 如果一个无差错的 ACK 到达，且它的确认号指向下一个要发送分组，也就是说 $\text{ackNo} = (S + 1) \bmod 2$ ，那么计时器停止计时，窗口滑动使 $S = (S + 1) \bmod 2$ 。最后发送方进入准备好状态。

2. 如果一个损坏的 ACK 到达，或者虽无差错但 $\text{ackNo} \neq (S + 1) \bmod 2$ 的 ACK 到达，这个 ACK 被丢弃。

3. 如果计时器超时，发送方重新传送那个唯一的待确认分组，并重启计时器。

接收方 接收方总是在准备好状态。变量 R 的初始值为 0。有三个事件可能会发生：

1. 如果一个 $\text{seqNo} = R$ 的无差错分组到达，那么这个分组中的报文被交付给应用层。然后窗口滑动到 $R = (R + 1) \bmod 2$ ，最后发送一个 $\text{ackNo} = R$ 的 ACK。

2. 如果一个 $\text{seqNo} \neq R$ 的无差错分组到达，这个分组被丢弃，但是要发送 $\text{ackNo} = R$ 的一个 ACK。

3. 如果一个损坏的分组到达，这个分组被丢弃。

例 13.4

图 13.21 所示为一个停止等待协议的例子。分组 0 被发送且确认。分组 1 丢失并在计时器超时后重传。重传的分组 1 被确认且计时器停止计时。分组 0 被发送且被确认，但是这个确认丢失了。发送方不知道丢失的是分组还是确认，因此在计时器超时后重传分组 0，重传的分组 0 被确认。

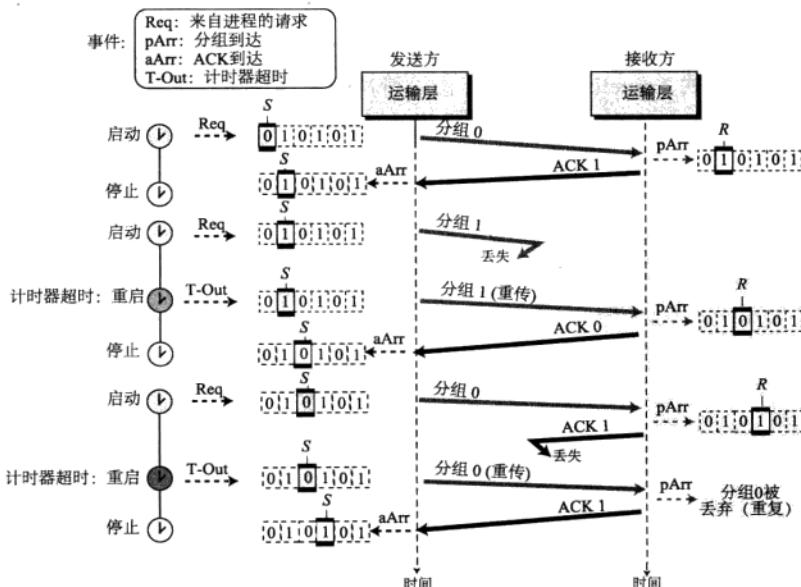


图 13.21 例 13.4 的流程图

效率

如果我们的信道又粗又长，那么停止等待协议是非常低效率的。所谓粗是指我们的信道有较大的带宽（高数据率），而所谓长是指往返时延较长。它们俩的乘积被称为带宽时延积（bandwidth-delay product）。我们可以把信道看成是一根管道，那么带宽时延积就是管道的容量，以比特为单位。管道一直放在那，如果我们不充分利用它，我们的效率就很低。带宽时延积是对发送方在等待接收方确认的同时能够通过系统传送的比特数的度量。

例 13.5

假设在一个停止等待系统中，线路带宽是 1 Mbps，且 1 比特的往返时间是 20 毫秒。它的带宽时延积是多少？如果这个系统的数据分组长度是 1000 比特，这条线路的利用率是百分之几？

解

它的带宽时延积是 $(1 \times 10^6) \times (20 \times 10^{-3}) = 20000$ 比特。这个系统在数据从发送方到达接收方，再由接收方返回确认这一段时间内总共可以发送 20000 比特的数据。但是这个系统仅发送了 1000 比特的数据。我们可以认为利用率仅为 $1000/20000$ 或 5%。正是由于这个原因，对于带宽很大或时延很长的线路来说，使用停止等待协议非常浪费线路容量。

例 13.6

假如在例 13.5 中我们有一个协议能够一次最多发送 15 个分组，然后再停下来等待它们的确认，此时线路的利用率的百分比是多少？

解

它的带宽时延积仍然是 20000 比特。系统能够在一个往返时间内发送 15 个分组或者说 15000 比特。这就意味着利用率可以达到 $15000/20000$ 或 75%。当然，如果其中有损坏的分组，这个利用率会因为分组的重传而大大降低。

流水线方式

在组网或其他领域内，一个任务通常会在它的前一个任务结束之前就已经开始了，我们称之为流水线方式（pipelining）。在停止等待协议中不使用流水线方式，因为发送方必须等待前一个分组到达它的终点并被确认之后才能发送下一个分组。但是，在我们要介绍的下面两个协议中都使用了流水线方式，因为发送方在接收到对前几个分组的反馈之前还可以再发送几个分组。因为相对于固定的带宽时延积，传送的比特数量变大了，所以说流水线方式提高了传输的效率。

13.2.3 返回 N 协议

为了提高传输的效率（填满管道），在发送方等待确认时应当有多个分组正在传送中。换言之，我们需要让多个分组处于等待确认的状态，以便在发送方等待确认的同时，信道也能保持忙碌状态。在这一小节中，我们先来讨论一个能够达到此目标的协议，而在下一小节我们再来讨论第二个此类协议。第一个协议称为返回 N（Go-Back-N, GBN），这个名字的由来稍后就会弄清楚。返回 N 协议的关键是我们在收到确认之前能够发送多个分组，但接收方只能缓存一个分组。发送方为发送出去的分组保存副本直至确认到达。图 13.22 所示为这个协议的概要图。请注意，在信道中同时可存在多个数据分组和确认。

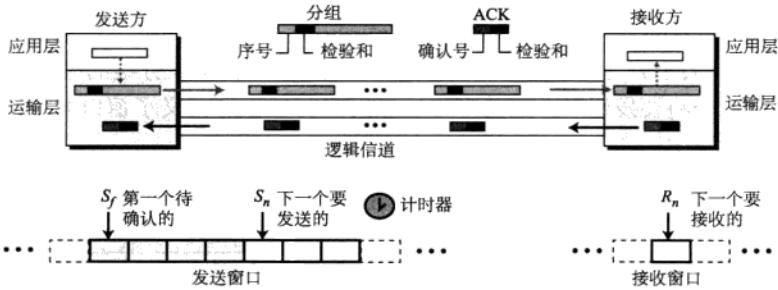


图 13.22 返回 N 协议

序号

我们在前面曾经说过序号必须是模 2^m 的，其中 m 是序号字段的长度，以比特为单位。

返回 N 协议中，序号必须是模 2^m 的，其中 m 是序号字段的长度，以比特为单位。

确认号

在这个协议中，确认号是一个累计值，定义了下一个希望接收的分组的序号。例如，如果确认号（ackNo）是 7，就表示一直到序号 6 之前的所有分组都已安全完好地到达了，并且接收方正在期盼的是序号为 7 的分组。

在返回 N 协议中，确认号是一个累积值，定义了希望下一个到达的分组的序号。

发送窗口

发送窗口是一个想象的方框，它覆盖了正在传送的或可以传送的数据分组的序号。在这个窗口中，有些序号定义的是已经发送出去的分组，另一些则定义了可以被发送的分组。

窗口的最大值是 $2^m - 1$ ，原因我们稍后再讨论。在本章，我们令这个窗口大小固定，并设置为最大值，但是我们将在后面的几章中看到，有一些协议可能使用可变的窗口大小。

图 13.23 描绘了一个返回 N 协议，其滑动窗口大小为 7 ($m = 3$)。

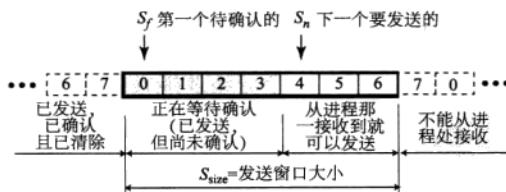


图 13.23 返回 N 协议的发送窗口

发送窗口在任何时刻都将可能的序号划分为四段。第一段在窗口的左边，它定义的序号属于已经被确认的分组。发送方不再为这些分组操心，也不保存这些分组的副本。第二段用阴影表示，在这个范围内的序号是属于已经发送出去但目前状态未知的分组。发送方还需要等待才能知道这些分组是被收到了还是丢失了。我们称它们是待确定分组。第三段是窗口中的白色区域，它定义了允许发送的分组的序号，只不过相应的数据还没有从应用层传过来。最后的第四段在窗口的右边，它定义的是还不能使用的序号，除非窗口滑动过去才行。

窗口本身就是一个抽象，在任何时刻都要用三个变量来定义它的大小和位置。我们称这些变量为 S_f （发送窗口，第一个待确认的分组）， S_n （发送窗口，下一个要发送的分组）， S_{size} （发送窗口，大小）。变量 S_f 定义了第一个（最老的）待确认分组的序号。变量 S_n 保存的是分配给下一个将要发送的分组的序号。最后，变量 S_{size} 定义了窗口大小，在我们的这个协议中是固定的。

发送窗口是一个抽象化的概念，它定义了一个想象的方框，最大窗口为 $2^m - 1$ ，且具有三个变量： S_f 、 S_n 和 S_{size} 。

图 13.24 描绘了当从另一端传来的一个确认到达时，发送窗口如何向右滑动一格或多格。在图中，一个确认号为 6 的确认到达。这就表示接收方等待接收序号为 6 的分组。

当 $ackNo$ 在 S_f 和 S_n 之间（采用模运算）的一个无差错的 ACK 到达后，发送窗口就向前滑动一格或多格。

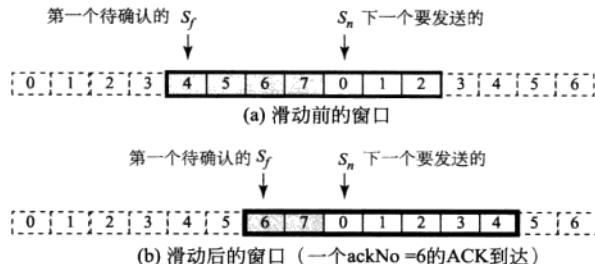


图 13.24 滑动发送窗口

接收窗口

接收窗口保证了正确的数据分组被接收，且正确的确认分组被发送。在返回 N 协议中，**接收窗口大小总是 1**。接收方总是在期待着某一个特定分组的到达。任何失序到达的分组都会被丢弃并重传。图 13.25 所示为接收窗口。请注意我们只需要一个变量 R_n （接收窗口，希望接收的下一个分组）来定义这个抽象窗口。窗口左边的序号属于已经接收且被确认的分组，窗口右边的序号定义的是不允许接收的分组。任何序号落在这两个区域中的分组都被丢弃。只有序号与 R_n 的值相匹配的那一个分组才能被接收和确认。接收窗口也要滑动，但一次只滑动一格。当一个正确的分组被接收后，窗口就滑动到 $R_n = (R_n + 1) \bmod 2^m$ 。

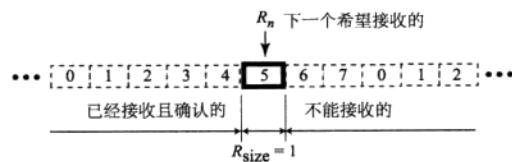


图 13.25 返回 N 的接收窗口

接收窗口是一个抽象概念，定义了一个大小为 1 且只有一个变量 R_n 的想象的方框。当一个正确的分组到达后，这个窗口就滑动，且一次只滑动一格。

计时器

虽然每个发送出去的分组都可以有一个计时器，但在我们的协议中总共只使用了一个计时器。原因是第一个待确认的分组总是会最先超时。当这个计时器超时后，我们就重传所有待确认的分组。

重传分组

当计时器超时后，发送方重传所有待确认的分组。例如，假设发送方已经发送了分组6 ($S_n = 7$)，但唯一的一个计时器超时了。如果 $S_f = 3$ ，那就是说分组3、4、5和6都尚未被确认，发送方返回到前面，重传分组3、4、5和6。这就是为什么这个协议叫做返回N的原因。一旦超时，发送方就返回到N位置并重传所有分组。

FSM

图13.26 描绘了GBN协议的FSM。

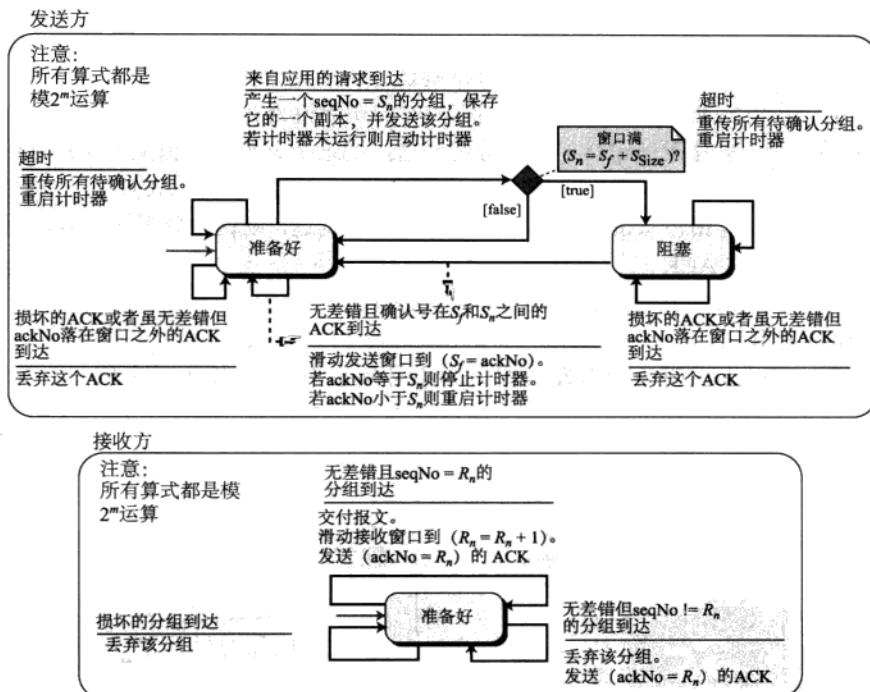


图13.26 返回N协议的FSM

发送方 发送方开始处于准备好状态，但是它有可能进入两种状态之一：准备好状态和阻塞状态。通常两个变量初始化为0 ($S_f = S_n = 0$)，但在后面的几章中我们将会看到在TCP/IP协议族中有些协议使用了不同的初始值。

□ **准备好状态** 当发送方位于准备好状态时有四个事件可能会发生。

1. 如果一个来自应用层的请求到达，发送方就产生一个序号置为 S_n 的分组。这个分组的一个副本被保存起来，而这个分组被发送出去。若唯一的那个计时器未运行则启动计时器。现在 S_n 的值递增到 $(S_n = S_n + 1) \bmod 2^m$ 。如果窗口满，即 $S_n = (S_f + S_{size}) \bmod 2^m$ ，则

发送方进入阻塞状态。

2. 如果一个无差错且 ackNo 与某一个待确认分组相关的 ACK 到达，发送方滑动窗口（置 $S_f = \text{ackNo}$ ），并且，如果所有待确认分组都被确认了 ($\text{ackNo} = S_n$) 则计时器停止计时。若还有待确认分组未被确认，则重启计时器。

3. 如果一个损坏的 ACK 或者虽无差错但 ackNo 与待确认分组无关的 ACK 到达，丢弃这个 ACK。

4. 如果计时器超时，发送方重传所有待确认分组并重启计时器。

□ 阻塞状态 在这种情况下可能会发生三个事件：

1. 如果一个无差错且 ackNo 与某一个待确认分组的 ACK 到达，发送方滑动窗口（置 $S_f = \text{ackNo}$ ），同时，若所有待确认分组都被确认了 ($\text{ackNo} = S_n$) 则计时器停止。若还有待确认分组未被确认，则重启计时器。然后发送方进入准备好状态。

2. 如果一个损坏的 ACK 到达或者虽无差错但 ackNo 与待确认分组无关的 ACK 到达，丢弃这个 ACK。

3. 如果计时器超时，发送方重传所有待确认分组并重启计时器。

接收方 接收方总是在准备好状态。变量 R_n 的初始值为 0。可能会发生三个事件：

1. 如果一个无差错且 $\text{seqNo} = R_n$ 的分组到达，则这个分组中的报文被交付给应用层。然后窗口滑动到 $R_n = (R_n + 1) \bmod 2^m$ 。最后发送一个 $\text{ackNo} = R_n$ 的 ACK。

2. 如果一个无差错但 seqNo 落在窗口之外的分组到达，则这个分组被丢弃，但是要发送一个 $\text{ackNo} = R_n$ 的 ACK。

3. 如果一个损坏的分组到达，这个分组被丢弃。

发送窗口大小

我们现在可以来说明为什么发送窗口大小必须小于 2^m 。作为一个例子，我们选择 $m = 2$ ，也就是说窗口大小可以是 $2^m - 1$ ，或者说 3。图 13.27 对一个大小为 3 的窗口和一个大小为 4 的窗口进行了比较。

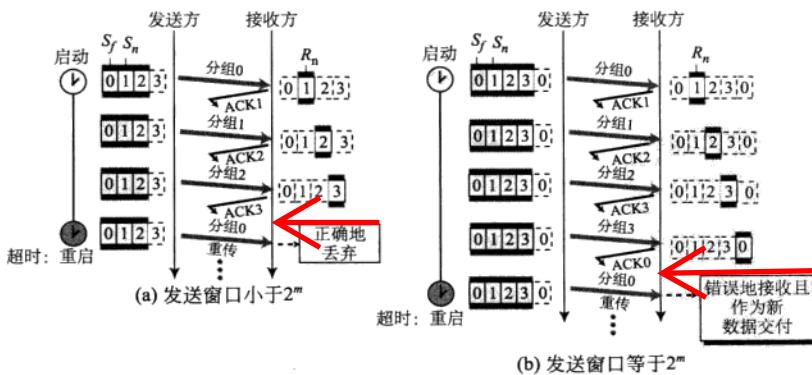


图 13.27 返回 N 协议的发送窗口大小

如果窗口大小是 3（小于 2^m ）且所有三个确认都丢失了，那么唯一的那个计时器超时后所有三个分组被重传。此时接收方期盼的是分组 3 而不是分组 0，因此这些重复的分组到达后会被正确地丢弃掉。另一方面，如果窗口大小是 4（等于 2^m ）且所有确认都丢失了，

发送方会发送重复的分组 0。但是此时接收方的窗口正在期盼着接收分组 0 (下一个循环)，所以它就接受分组 0，它认为这不是一个重复分组，而是下一循环的第一个分组。这就是一个错误。从中我们可以看出发送窗口大小必须小于 2^m 。

在返回 N 协议中，发送窗口大小必须小于 2^m ，接收窗口大小始终为 1。

例 13.7

图 13.28 所示为返回 N 协议的一个例子。这是前向信道可靠，但反向信道不可靠的一个例子。没有数据分组丢失，但有些 ACK 延迟了，且其中一个 ACK 丢失了。这个例子还说明了如果确认被延迟或丢失，累计确认应当如何提供帮助。

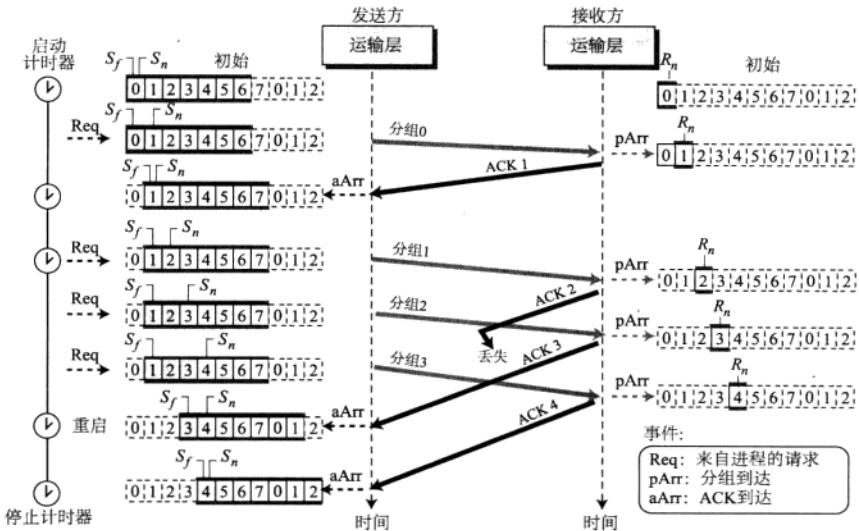


图 13.28 例 13.7 的流程图

在初始化之后有几个发送方事件。请求事件是由来自应用层的报文块触发的，到达事件是由来自网络层的 ACK 触发的，此处没有超时事件，因为在计时器超时之前所有待确认的分组都得到了确认。请注意，虽然 ACK2 丢失，但 ACK3 是一个累计确认，它既作为 ACK3，也作为 ACK2。在接收方一共发生了四个事件。

例 13.8

图 13.29 所示为在一个分组丢失后发生的情况。分组 0、1、2 和 3 被发送。但是分组 1 丢失了。接收方收到分组 2 和 3，但它们因失序到达而被丢弃（期盼的是分组 1）。当接收方收到分组 2 和 3 时，它会发送 ACK1 来表示自己期盼分组 1。但是这些 ACK 对于发送方没有起到什么作用，因为这个 ackNo 等于 S_f ，而不是大于 S_f ，因此发送方丢弃它们。当超时事件发生后，发送方重传分组 1、2 和 3，然后它们被确认。

返回 N 与停止等待的比较

读者可能发现在返回 N 协议与停止等待协议之间有相似之处。停止等待协议实际上就是一种返回 N 协议，不过它只有两个序号且发送窗口大小为 1。换言之就是 $m=1$ 且 $2^m - 1 = 1$ 。在返回 N 协议中，我们说计算都是模 2^m 的，在停止等待协议中计算都是模 2 的，

当 $m=1$ 时模 2 和模 2^m 是一样的。

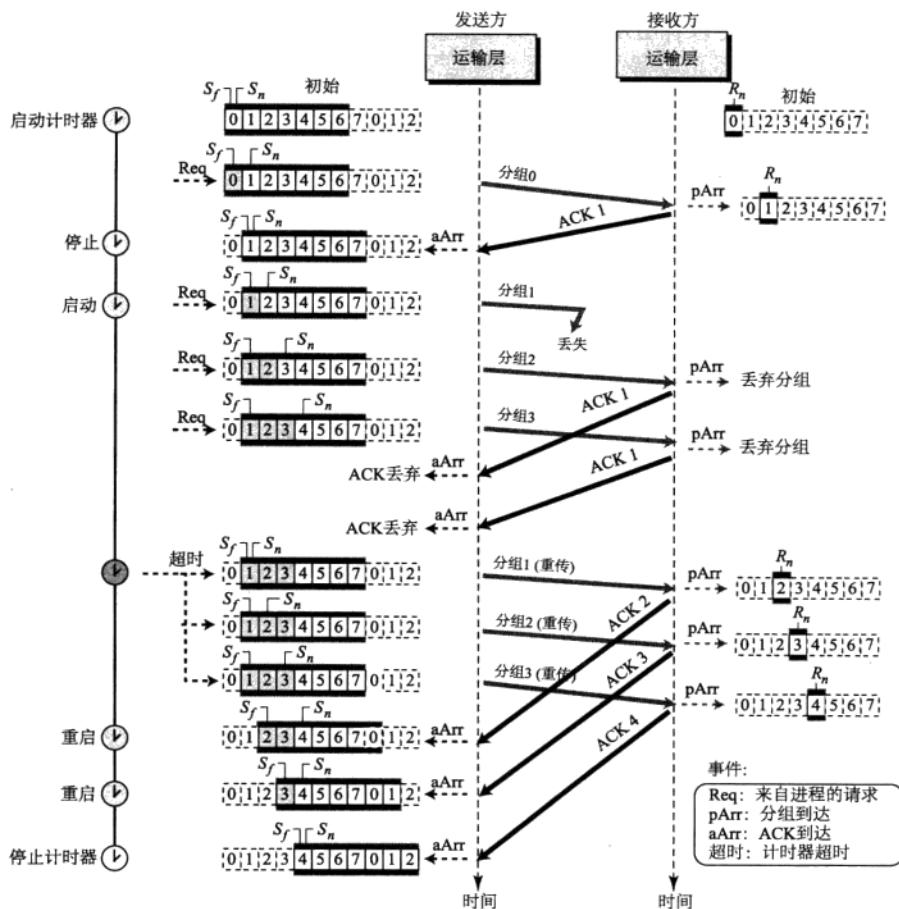


图 13.29 例 13.8 的流程图

13.2.4 选择重传协议

返回 N 协议简化了接收方的处理过程。接收方只需跟踪一个变量，并且不需要对失序到达的分组进行缓存，它们被简单地丢弃掉。但是如果底下的网络层协议丢失了很多分组，那么这个协议的效率就会很低。每当一个分组丢失或损坏，发送方就要重传所有待确认的分组，虽然其中有些分组实际上已安全完好地被接收了。如果因为网络的拥塞致使网络层丢失了很多分组，那么对所有这些待确认分组的重传会令拥塞状况加剧，并最终导致更多的分组丢失。它具有一种雪崩效应并有可能导致网络完全崩溃。

人们设计的另一种协议称为选择重传（Selective-Repeat, SR）协议，就像名字所暗示的那样，它只重传选择的分组，也就是真正丢失的分组。这个协议的概要图如图 13.30 所示。

窗口

选择重传协议也使用了两个窗口：一个发送窗口和一个接收窗口。但是这个协议中的两个窗口与返回 N 协议中的两个窗口有所不同。首先，它的发送窗口的最大值要小得多，是 2^{m-1} ，原因我们稍后再讨论。其次，它的接收窗口与发送窗口一样大。

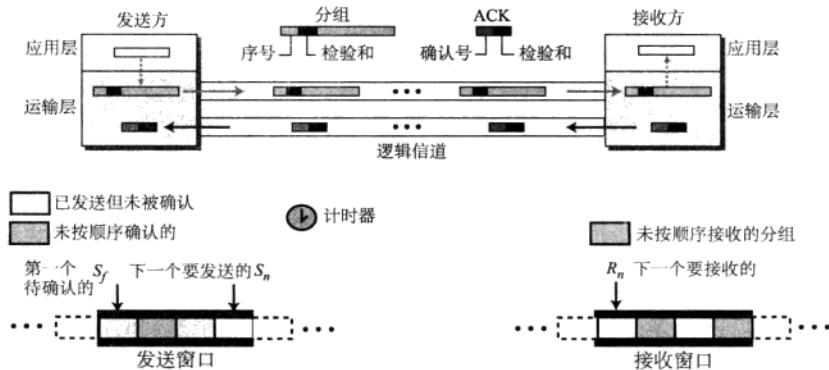


图 13.30 选择重传协议

发送窗口的最大值可以是 2^{m-1} 。例如，如果 $m = 4$ ，那么序号就是从 0 到 15，但窗口的最大值只能是 8（在返回 N 协议中，这个值是 15）。我们在图 13.31 中描绘了选择重传协议的发送窗口以强调它的大小。

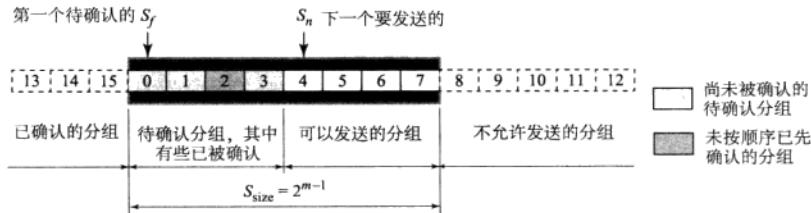


图 13.31 选择重传协议的发送窗口

选择重传的接收窗口与返回 N 协议的接收窗口完全不同。它的接收窗口与发送窗口一样大（最大为 2^{m-1} ）。选择重传协议允许与接收窗口大小一样多的分组失序到达并且保存这些分组直到连续的一组分组被交付给应用层。因为发送窗口大小与接收窗口的相同，所以发送出来的所有分组都可以失序到达，而且会被保存直到交付为止。但是我们需要强调一点，在一个可靠的协议中，接收方是永远不会把分组失序地交付给应用层的。图 13.32 描绘了选择重传协议中的接收窗口。窗口中有阴影的小格子定义的是已经失序到达的分组，它们在被交付给应用层之前，先要等待那些更早发送出来的分组的到达。

计时器

理论上，选择重传要为每个待确认的分组使用一个计时器。当某个计时器超时后，只

有相应的分组被重传。换言之，GBN 将所有待确认的分组当作一个整体对待，而 SR 则分别对待每一个分组。但是，大多数应用了 SR 的运输层协议仅使用了一个计时器。因此我们也只使用一个计时器。

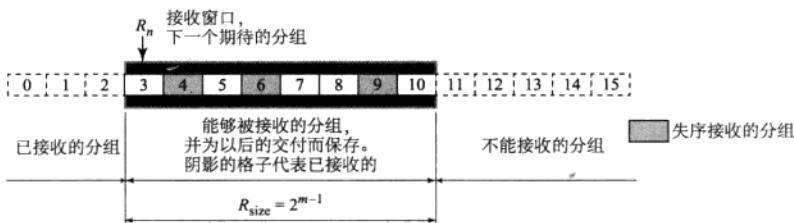


图 13.32 选择重传协议的接收窗口

确认

这两个协议之间还存在其他一些区别。在 GBN 中，`ackNo` 是累计的，它定义了下一个希望接收的分组的序号，同时也证实了在此之前的所有分组都已经安全完好地被接收了。在 SR 中，确认的语义与此不同。在 SR 中，`ackNo` 只定义了安全完好地收到的那一个分组的序号，并不反馈任何其他分组的信息。

在选择重传协议中，确认号定义了无差错地接收到的那一个分组的序号。

例 13.9

假设发送方发送了 6 个分组：分组 0、1、2、3、4 和 5。发送方接收到一个 `ackNo` 为 3 的 ACK。如果这个系统使用的是 GBN 或者是 SR，它的含义各是什么？

解

如果系统使用的是 GBN，它就表示分组 0、1 和 2 都已被接收且无损坏，接收方现在期盼的是分组 3。如果系统使用的是 SR，它就表示分组 3 已经被接收且无损坏，这个分组没有说有关其他分组的任何事。

FSM

图 13.33 描绘了选择重传协议的 FSM。它与 GBN 的类似，但有几处不同的地方。

发送方 发送方开始时处于准备好状态，但是之后它有可能进入两个状态之一：准备好状态和阻塞状态。下面列出了每种状态的事件及相应的动作。

□ **准备好状态** 在这种情况下可能会发生四个事件。

- 若一个来自应用的请求到达，发送方就产生一个序号置为 S_n 的分组。这个分组的一个副本被保存起来，并且这个分组被发送出去。若计时器未运行则发送方启动计时器。 S_n 的值递增， $S_n = (S_n + 1) \bmod 2^m$ 。若窗口满，即 $S_n = (S_f + S_{size}) \bmod 2^m$ ，发送方就进入到阻塞状态。

- 若一个无差错且 `ackNo` 与某个待确认分组相关的 ACK 到达，该分组被标记为已确认。若 $ackNo = S_f$ ，窗口向右滑动，直到 S_f 指向第一个未被确认的分组（所有连续被确认的分组现在都滑出窗口外边了）。若还有待确认分组未被确认，则重启计时器，否则计时器停止计时。

3. 若一个损坏的 ACK 到达或者虽无差错但 ackNo 与待确认分组无关的 ACK 到达，丢弃这个 ACK。

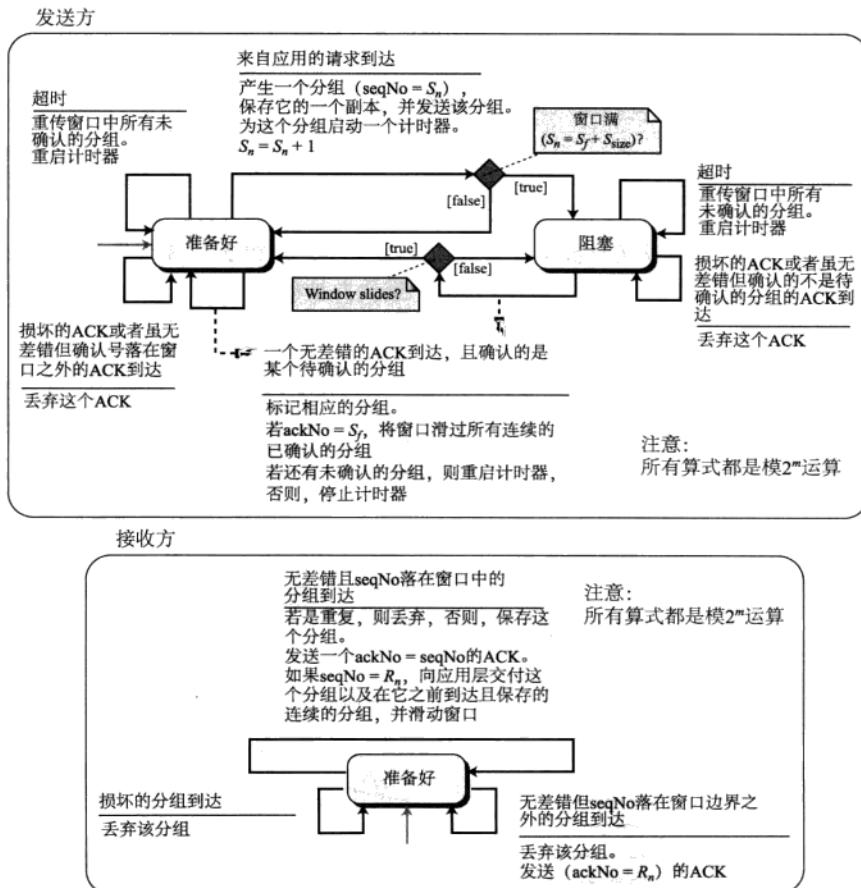


图 13.33 SR 协议的 FSM

4. 若计时器超时，发送方重传窗口中所有未被确认的分组并重启计时器。

阻塞状态 在这种情况下可能会发生三个事件：

1. 若一个无差错且 ackNo 与某一个待确认分组相关的 ACK 到达，该分组被标记为已确认。另外，若 $ackNo = S_f$ 则窗口向右滑动直到 S_f 指向第一个未被确认的分组（所有连续被确认的分组现在都滑出窗口外边了）。若窗口滑动了，发送方就进入准备就绪状态。
2. 若一个损坏的 ACK 到达或者虽无差错但 ackNo 与待确认分组无关的 ACK 到达，丢弃这个 ACK。

3. 若计时器超时，发送方重传窗口中所有未被确认的分组并重启计时器。

接收方 接收方总是在准备就绪状态。可能会发生三个事件：

1. 若一个无差错且 seqNo 落在窗口中的分组到达，则这个分组被保存，且发送一个

$\text{ackNo} = \text{seqNo}$ 的分组。另外，若 $\text{seqNo} = R_n$ ，则这个分组以及所有前面到达的连续的分组都被交付给应用层，且窗口滑动，使 R_n 指向第一个空格。

2. 若一个无差错但 seqNo 落在窗口之外的分组到达，则这个分组被丢弃，但是要返回一个 $\text{ackNo} = R_n$ 的 ACK 给发送方。这样做的目的是，如果与某个 $\text{seqNo} < R_n$ 的分组相关的 ACK 丢失了，发送方要能够滑动自己的窗口。

3. 如果一个损坏的分组到达，这个分组被丢弃。

例 13.10

这个例子类似于例 3.8（图 13.29），其中分组 1 丢失了。我们描绘了在这种情况下选择重传协议是如何动作的。图 13.34 描绘了此情况。

在发送方，分组 0 被发送且被确认。分组 1 丢失了。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）重传并被确认。发送窗口滑动。

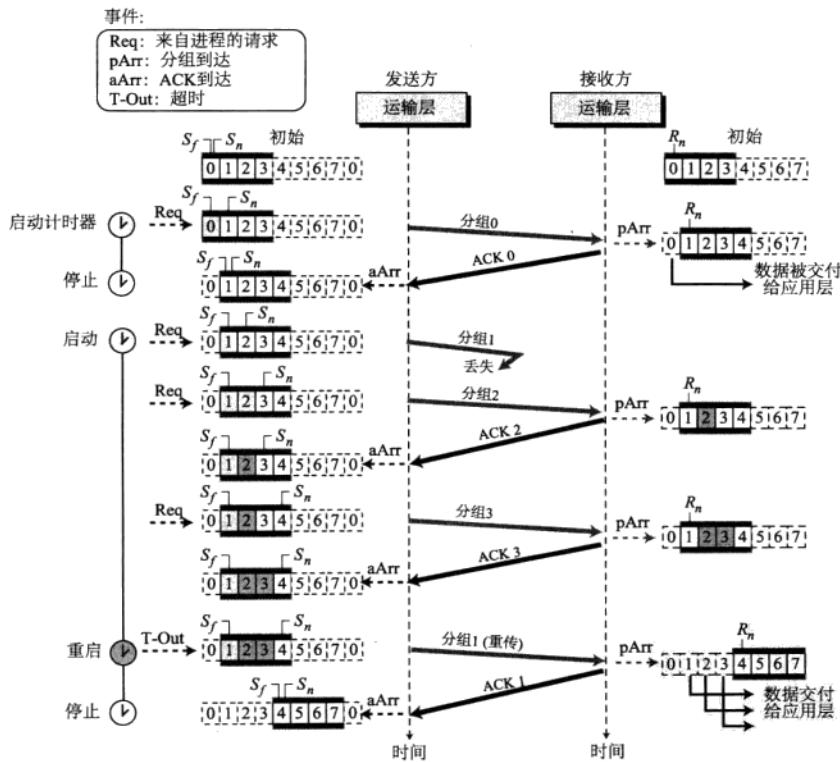


图 13.34 例 13.10 的流程图

在接收方，我们需要区别对待分组的接受和向应用层的交付。在第二个到达时刻，分组 2 到达并且被保存和标记（阴影格），但是它不能被交付，因为分组 1 还没有到。在下一个到达时刻，分组 3 到达并被保存和标记，但是仍然没有分组能被交付。只有在最后一个到达时刻，也就是当分组 1 的副本姗姗来迟后，分组 1、2 和 3 才能一起被交付给应用层。

向应用层交付分组需要两个条件：首先，已经到达的必须是一组连续的分组。其次，这一组连续的分组以窗口的前端为起始。在第一个到达时刻，只有一个分组且它就在窗口的前端。在最后一个到达时刻，一共有三个分组且第一个分组在窗口的前端。关键就是可靠的运输层允诺按顺序交付分组。

窗口大小

现在我们可以来说明为什么发送方和接收方的窗口的最大值只能是 2^m 的一半了。例如，我们选择 $m = 2$ ，也就是说窗口大小是 $2^m/2$ 或者 2。图 13.35 对大小为 2 的窗口和大小为 3 的窗口进行比较。

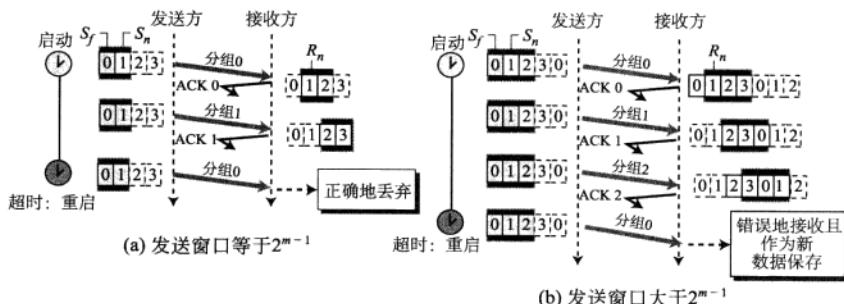


图 13.35 选择重传，窗口大小

如果窗口大小是 2 且所有确认都丢失了，当分组 0 的计时器超时后分组 0 被重传。此时接收方期盼的是分组 2 而不是分组 0，因此这个重复的分组会被正确地丢弃掉（序号 0 不在窗口中）。如果窗口大小是 3 且所有确认都丢失了，发送方发送重复的分组 0。但是这一次，接收方的窗口正在期盼着接收分组 0（0 属于窗口中），所以它就接受分组 0，它不认为这是一个重复的分组，而是认为这个分组是下一个循环的第一个分组。显然这是错误的。

在选择重传协议中，发送窗口和接收窗口的最大值只能是 2^m 的一半。

13.2.5 双向协议：捎带

在这一节我们讨论的四种协议全部都是单向的：数据分组仅在一个方向流动，而确认从另一个方向传过来。但是在现实生活中数据分组通常都是双向流动的：从客户到服务器，并且从服务器到客户。这就意味着确认也需要双向流动。一种称为捎带（piggybacking）的技术可用于提高双向协议的效率。当分组从 A 向 B 携带数据的同时也能够携带返回的确认，这些确认说明了从 B 发来的分组的到达情况。而当分组从 B 向 A 携带数据的同时也能够携带返回的确认，这些确认又说明了从 A 发来的分组的到达情况。

图 13.36 所示为利用捎带实现双向 GBN 协议的概要图。客户和服务器各使用两个独立的窗口：发送窗口和接收窗口。我们将这种情况（以及其他情况）的 FSM 留作练习。

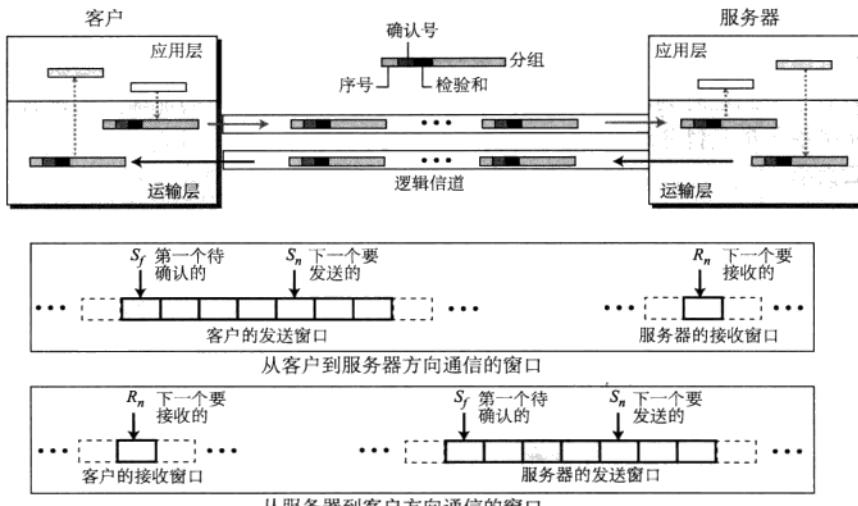


图 13.36 返回 N 协议中的捎带设计

13.3 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍：[Com 06]、[Pet & Dev 03]、[Kur & Ros 08]、[Gar & Vid 04]、[Far 04]、[Tan 03]和[Sta 04]，用方括号括起来的书目可以在本书末尾的参考书目清单中找到。另外，我们还要推荐一个提供了大量信息的论文：“The Transport Layer: Tutorial and Survey” *ACM Computing Surveys* vol. 31, no. 4, Dec. 1999, 作者：Sami Iren, Paul D. Amer 和 Phillip T. Conrad。

13.4 重要术语

确认号	开环拥塞控制
带宽时延积	捎带
客户-服务器范式	流水线方式
闭环拥塞控制	端口号
拥塞	进程到进程的通信
拥塞控制	选择重传协议
分用	序号
临时端口号	滑动窗口
有限状态机	套接字地址
返回 N 协议	停止等待协议
负载	熟知端口号
复用	

13.5 本章小结

- 运输层协议的主要任务是提供进程到进程的通信。要定义进程，我们就需要端口号。客户进程用一个临时端口号来定义自己。服务器用一个熟知端口号来定义自己。
- 为了把报文从一个进程发送到另一个进程，运输层协议要对报文进行封装和解封。封装发生在发送端，解封发生在接收端。
- 源点的运输层执行报文的复用，就是从多个进程那里收集报文并传输。终点的运输层执行分用，就是把报文交付给不同的进程。
- 流量控制用于平衡生产者和消耗者之间数据的交换。在运输层，当消耗者尚未准备好接收分组时，我们就用缓存来保存这些分组。运输层的可靠性通过加入差错控制来实现，包括检测损坏的分组，重传丢失的以及损坏的分组，丢弃重复的分组，并为失序到达的分组重新排序。要管理流量控制和差错控制，我们使用了序号来为分组编号，并使用确认号指向编号的分组。
- 运输层能够提供两种类型的拥塞控制：开环的和闭环的。在开环拥塞控制中，协议试图避免拥塞的发生。在闭环拥塞控制中，协议试图在拥塞发生之后检测并解除拥塞。
- 运输层协议能够提供两种类型的服务：无连接的和面向连接的。在无连接的服务中，发送方不用建立任何连接就可以向接收方发送分组。在面向连接的服务中，客户和服务器首先要建立它们之间的连接。只有在连接建立后才能交换数据。而当数据交换完毕后，连接需要被拆除。
- 本章我们讨论了几种通用的运输层协议。简单无连接协议既不提供流量控制，也不提供差错控制。面向连接的停止等待协议提供了流量控制和差错控制，但是效率很低。返回 N 协议是停止等待协议的一个版本，它利用了流水线方式。选择重传协议是对返回 N 协议的改进，它在处理分组丢失上更具优势。所有这些协议都能通过捎带技术实现双向传输。

13.6 实践安排

13.6.1 习题

1. 发送方使用 5 位的序号向同一个终点发送了一个分组序列。如果序号从 0 开始，那么第 100 个分组的序号是什么？
2. 使用 5 位的序号，对以下几种协议来说，发送窗口和接收窗口的最大值分别是多少？
 - a. 停止等待。
 - b. 返回 N 。
 - c. 选择重传。
3. 为一个想象的机器画出 FSM，它有三个状态：状态 A（开始状态）、状态 B 和状态 C。

C，并有四个事件：事件 1、2、3 和 4。以下是该机器的具体行为：

- 当在状态 A 时，可能发生两个事件：事件 1 和事件 2。如果事件 1 发生，机器执行动作 1 并进入状态 B。如果事件 2 发生，机器进入状态 C（无动作）。
- 当在状态 B 时，可能发生两个事件：事件 3 和事件 4。如果事件 3 发生，机器执行动作 2，但仍然保持在状态 B。如果事件 4 发生，机器就进入状态 C。
- 当在状态 C 时，机器永远保持在这个状态。

4. 请重新设计图 13.15 所示的 FSM，假设连接的建立只需要交换 3 个分组就可以完成（合并分组 2 和 3）。

5. 重画图 13.18，假设图中一共交换了 5 个分组（0、1、2、3、4），分组 2 丢失，且分组 3 在分组 4 之后到达。

6. 创建一个类似图 13.21 所示的场景，其中发送方发送了三个分组。第一个和第二个分组到达且被确认。第三个分组因延迟而被重传。接收方在对原始的第三个分组的确认发出之后又收到了这个重复的分组。

7. 创建一个类似图 13.21 所示的场景，其中发送方发送了两个分组。第一个分组到达并被确认，但是这个确认丢失了。发送方在计时器超时后重传该分组。第二个分组丢失并被重传。

8. 重画图 13.28，假设发送方发送了 5 个分组（0、1、2、3、4），分组 0、1 和 2 发送后被一个 ACK 一起确认，这个 ACK 在发送方发送完所有分组后到达。分组 3 被接收并通过一个 ACK 中确认。分组 4 丢失且重传。

9. 重画图 13.34，假设发送方发送了 5 个分组（0、1、2、3、4）。分组 0、1 和 2 按顺序逐个地接收并被确认。分组 3 延迟并在分组 4 之后被接收。

10. 请回答以下与停止等待协议的 FSM 相关的问题（图 13.20）：

- 发送方状态机在准备好状态且 $S = 0$ 。下一个要发送的分组的序号是什么？
- 发送方状态机在阻塞状态且 $S = 1$ 。如果计时器超时，下一个要发送的分组的序号是什么？
- 接收方状态机在准备好状态且 $R = 1$ 。一个序号为 1 的分组到达。对这个事件的响应是什么？
- 接收方状态机在准备好状态且 $R = 1$ 。一个序号为 0 的分组到达。对这个事件的响应是什么？

11. 请回答以下与返回 N 协议 ($m = 6$) 的 FSM 相关的问题（图 13.26）：

- 发送方状态机在准备好状态， $S_f = 0$ 且 $S_n = 15$ 。下一个要发送的分组的序号是什么？
- 发送方状态机在准备好状态， $S_f = 0$ 且 $S_n = 15$ 。发生一个超时事件。有多少个分组要重发？它们的序号是什么？
- 发送方状态机在准备好状态， $S_f = 0$ 且 $S_n = 15$ 。一个 $ackNo = 13$ 的 ACK 到达。 S_f 和 S_n 的下一个值分别是什么？
- 发送方状态机在阻塞状态， $S_f = 14$ 且 $S_n = 21$ 。这个窗口是多大？
- 发送方状态机在阻塞状态， $S_f = 14$ 且 $S_n = 21$ 。一个 $ackNo = 18$ 的 ACK 到达， S_f 和 S_n 的下一个值分别是什么？发送方状态机的状态是什么？

- f. 接收方状态机在准备好状态且 $R_n = 16$ 。一个序号为 16 的分组到达。 R_n 的下一个值是什么？对这个事件的响应是什么？
12. 请回答以下与选择重传协议 ($m = 7$ 位) 的 FSM 相关的问题 (图 13.33):
- 发送方状态机在准备好状态, $S_f = 0$ 且 $S_n = 15$ 。下一个要发送的分组的序号是什么？
 - 发送方状态机在准备好状态, $S_f = 0$ 且 $S_n = 15$ 。分组 10 的计时器超时。有多少个分组要重发？它们的序号是什么？
 - 发送方状态机在准备好状态, $S_f = 0$ 且 $S_n = 15$ 。一个 $ackNo = 13$ 的 ACK 到达。 S_f 和 S_n 的下一个值分别是什么？对这个事件的响应是什么？
 - 发送方状态机在阻塞状态, $S_f = 14$ 且 $S_n = 21$ 。这个窗口大小是多少？
 - 发送方状态机在阻塞状态, $S_f = 14$ 且 $S_n = 21$ 。一个 $ackNo = 14$ 的 ACK 到达，分组 15 和 16 已经确认。 S_f 和 S_n 的下一个值分别是什么？发送方状态机的状态是什么？
 - 接收方状态机在准备好状态且 $R_n = 16$ 。这个窗口大小为 8。一个序号为 16 的分组到达。 R_n 的下一个值是什么？对这个事件的响应是什么？
 - 接收方状态机在准备好状态且 $R_n = 16$ 。这个窗口大小为 8。一个序号为 18 的分组到达。 R_n 的下一个值是什么？对这个事件的响应是什么？

13.6.2 研究活动

- 重画图 13.16 中所示的简单协议双向图（使用捎带技术）。
- 重画图 13.19 中所示的停止等待协议双向图（使用捎带技术）。
- 重画图 13.30 中所示的选择重传协议双向图（使用捎带技术）。
- 画出使用了捎带技术的简单协议的双向 FSM 图。请注意双方都需要发送和接收。
- 画出使用了捎带技术的停止等待协议的双向 FSM 图。请注意双方都需要发送和接收。
- 画出使用了捎带技术的选择重传协议的双向 FSM 图。请注意双方都需要发送和接收。
- 试用伪码（或者用某种计算机语言）写出与简单协议相关的 FSM（图 13.17）的两个算法。
- 试用伪码（或者用某种计算机语言）写出与停止等待协议相关的 FSM（图 13.20）的两个算法。
- 试用伪码（或者用某种计算机语言）写出与返回 N 协议相关的 FSM（图 13.26）的两个算法。
- 试用伪码（或者用某种计算机语言）写出与选择重传协议相关的 FSM（图 13.33）的两个算法。

第 14 章 用户数据报协议 (UDP)

最初的 TCP/IP 协议族为运输层指明了两个协议： UDP 和 TCP。我们先讨论 UDP，这是比较简单的一个，然后第 15 章再讨论 TCP。一个新设计的运输层协议 SCTP 将在第 16 章讨论。

目标

本章有以下几个目标：

- 介绍 UDP 并展示它在 TCP/IP 协议族中与其他协议之间的关系。
- 说明被称为用户数据报的 UDP 分组的格式，并讨论首部中各字段的应用。
- 讨论 UDP 提供的服务，如进程到进程的交付，简单的差错控制，复用和分用以及排队。
- 说明如何计算可选的检验和，以及为什么 UDP 分组的发送方需要在计算检验和时给分组添加一个伪首部。
- 讨论某些应用程序如何从 UDP 的简单性中受益。
- 简单地讨论一个实现了 UDP 的软件包的结构，并给出控制块、输入和输出模块的描述。

14.1 引言

图 14.1 描绘了用户数据报协议 (User Datagram Protocol, UDP) 与 TCP/IP 协议族的其他协议以及其他层次之间的关系： UDP 位于应用层和 IP 层之间，它提供介于应用程序和网络功能之间的服务。

正如我们在第 13 章中讨论的，运输层协议通常具有几种责任。一种责任就是创建进程到进程的通信，UDP 使用端口号来完成这种通信。另一种责任就是在运输层提供控制机制。UDP 在一个非常低的水平上完成了这个功能。UDP 没有流量控制机制，在收到分组时也没有确认。但是，UDP 提供了某种程度的差错控制。如果 UDP 在收到的分组中检测出有差错，它就悄悄地丢弃这个分组。

UDP 是一种无连接、不可靠的运输协议 (connectionless, unreliable transport protocol)。它除了在 IP 服务的基础上增加了进程到进程的通信，使之不再是主机到主机的通信之外，就再没什么了。

如果 UDP 的功能这样弱，为什么有些进程还愿意使用它？缺点有时也会变成优点。



另外 UDP 的检验和是检验整个报文，而 IP 只检验首部。

UDP 是一个非常简单的协议，使用的额外开销也是最小的。若某进程想发送一个很短的报文并且不关心其可靠性，它就可以使用 UDP。使用 UDP 发送短报文时，在发送方和接收方之间的交互要比使用 TCP 时少得多。

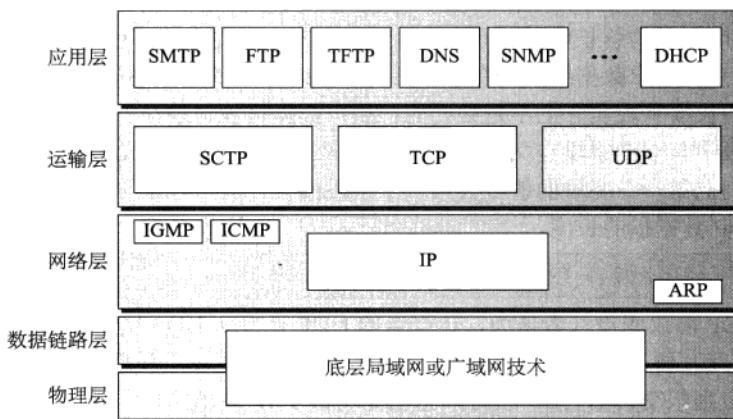
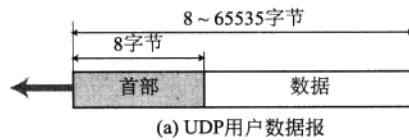


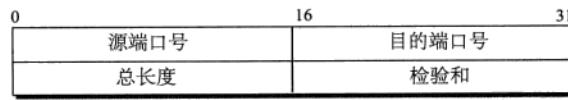
图 14.1 UDP 在 TCP/IP 协议族中的位置

14.2 用户数据报

UDP 分组叫做 **用户数据报** (user datagrams)，它有八个字节的固定首部。图 14.2 给出了用户数据报的格式。这些字段如下：



(a) UDP 用户数据报



(b) 首部格式

图 14.2 用户数据报的格式

- **源端口号** 这是在源主机上运行的进程所使用的端口号。它有 16 位长，这就表示端口号可以从 0~65535。若源主机是客户（当客户进程发送请求时），则在大多数情况下这个端口号是该进程请求的，并由运行在源主机上的 UDP 软件选择出来的一个临时端口号。若源主机是服务器（当服务器进程发送响应时），则在大多数情况下这个端口号是一个熟知端口号。
- **目的端口号** 这是运行在目的主机上的进程所使用的端口号。它也是 16 位长。若目的主机是服务器（当客户进程发送请求时），则在大多数情况下这个端口号是熟

知端口号。若目的主机是客户端（当服务器进程发送响应时），则在大多数情况下这个端口号是一个临时端口号。此时，服务器会把它收到的请求分组中的临时端口号复制过来。

- **总长度** 这是一个 16 位的字段，它定义了用户数据报的总长度，首部加上数据。
 16 位可定义的总长度是从 0~65535 字节。但是，实际的总长度必须比这个数值小，因为 UDP 用户数据报要放在总长度为 65535 字节的 IP 数据报之中。UDP 用户数据报中的长度字段实际上不是必要的。用户数据报被封装在一个 IP 数据报中。在 IP 数据报中有一个字段定义了总长度，另外还有一个字段定义了首部的长度。因此如果我们把第一个字段的值减去第二个字段的值，就可以推导出封装在 IP 数据报中的 UDP 数据报的长度。

$$\text{UDP 长度} = \text{IP 长度} - \text{IP 首部长度}$$

但是，UDP 协议的设计者认为，让终点的 UDP 直接从 UDP 用户数据报中提供的信息来计算数据长度，要比请求 IP 软件来提供这一信息效率更高。我们应当记得，当 IP 软件把 UDP 用户数据报交付给 UDP 层时，它已经剥去了 IP 首部。

- **检验和** 这个字段用来检测整个用户数据报（首部加上数据）出现的差错。在下一节将讨论检验和。

例 14.1

下面是以十六进制格式存储的一个 UDP 首部：

CB84000D001C001C

- 源端口号是什么？
- 目的端口号是什么？
- 这个用户数据报的总长度是多少？
- 数据长度是多少？
- 这个分组是从客户到服务器方向的，还是从服务器到客户方向的？
- 客户进程是什么？

解

- 源端口号是最前面的四位十六进制数字 (CB84_{16})，代表着源端口号为 52100。
- 目的端口号是第二个四位十六进制数字 ($000D_{16}$)，代表着目的端口号为 13。
- 第三个四位十六进制数字 ($001C_{16}$) 定义了整个 UDP 分组的长度为 28 字节。
- 数据的长度是整个分组的长度减去首部的长度，也就是 $28 - 8 = 20$ 字节。
- 因为目的端口号是 13（熟知端口），所以这个分组是从客户到服务器的。
- 客户进程是 Daytime（参见表 14.1）。

14.3 UDP 服务

我们在第 13 章中已经讨论过运输层协议提供的服务。在这一小节，我们将讨论 UDP 提供了这些通用服务中的哪些部分。

14.3.1 进程到进程的通信

UDP 使用套接字 (IP 地址和端口号的组合) 提供了第 13 章讨论的进程到进程的通信。几个被 UDP 使用的端口号如表 14.1 所示。

表 14.1 UDP 使用的熟知端口

端口	协议	说 明
7	Echo	把收到的数据报回送到发送方
9	Discard	丢弃收到的任何数据报
11	Users	活跃的用户
13	Daytime	返回日期和时间
17	Quote	返回日期的引用 (译注: 可参阅 RFC 865)
19	Chargen	返回字符串
53	Domain	域名服务 (DNS)
67	Bootps	下载引导程序信息的服务器端口
68	Bootpc	下载引导程序信息的客户端口
69	TFTP	简单文件传送协议
111	RPC	远程过程调用
123	NTP	网络时间协议
161	SNMP	简单网络管理协议
162	SNMP	简单网络管理协议 (陷阱)

14.3.2 无连接服务

正如前面讨论的, UDP 提供无连接服务。这就意味着由 UDP 发送的每一个用户数据报都是独立的数据报。不同的用户数据报之间没有任何关系, 哪怕它们是来自相同的源进程并且去往相同的目的程序。用户数据报没有编号。此外, 也没有像 TCP 那样的连接建立和连接终止过程。这就表示每一个用户数据报可以走不同的路径。

无连接导致的一个结果就是: 使用 UDP 的进程不能发送数据流到 UDP, 也不能期望 UDP 把这个数据流分割成为许多个相互关联的用户数据报。相反, 进程的每一个请求都必须足够小, 使其能够装进一个用户数据报中。只有那些发送短报文的进程才应当使用 UDP, 也就是报文的长度要小于 65507 字节 (65535 减去 8 个字节的 UDP 首部, 再减去 20 个字节的 IP 首部)。

14.3.3 流量控制

UDP 是一个非常简单的协议。它没有流量控制, 因此也没有窗口机制。接收方可能会因为入口的报文太多而溢出。缺少流量控制就意味着在必要时应当由使用 UDP 的进程来提

供这种服务。

14.3.4 差错控制

除检验和之外，UDP 没有其他差错控制机制。这就表示发送方并不知道报文是丢失了还是重复交付了。当接收方通过检验和检测出差错时，就悄悄地将这个用户数据报丢掉。缺少差错控制就意味着在必要时应当由使用 UDP 的进程来提供这种服务。

检验和

我们在第 7 章已经讨论过检验和的概念及其计算方法。UDP 检验和的计算与 IP 检验和的计算不同。这里的检验和涉及了三个部分：一个伪首部、UDP 首部以及从应用层来的数据。

伪首部（pseudoheader）是封装用户数据报的那个 IP 分组的首部的一部分，其中有些字段要填入 0（参见图 14.3）。



图 14.3 用于检验和计算的伪首部

若检验和不包括伪首部，用户数据报也可能是安全的和正确的。但是，若 IP 首部受到损伤，则它也有可能被交付到错误的主机。

增加一个协议字段是为了确保这个分组属于 UDP，而不是属于 TCP。我们在后面将会看到，若一进程既可使用 UDP 又可使用 TCP，则目的端口号可能是一样的。UDP 的协议字段值是 17。若在传输过程中这个值改变了，在接收端计算检验和时就可检测出来，UDP 就会丢弃这个分组。这样就不会交付给错误的协议了。

请注意这个伪首部中的字段与 IP 首部最后的 12 个字节的相似之处。

例 14.2

图 14.4 所示为一个非常小的用户数据报的检验和的计算过程，这个用户数据报只有 7 个字节的数据。因为数据的字节数是奇数，所以为了计算检验和就需要加入填充。当用户数据报被交付给 IP 后，这个伪首部和填充都会被拆除（参见附录 F）。

检验和的可选使用

UDP 分组的发送方可以选择不计算检验和。若不计算检验和，则在发送之前把检验和字段全部填入 0。如果发送方决定计算检验和，但巧的是得到的结果也是全 0，在这种情况下，在该分组发送之前，检验和变成全 1。换言之，发送方要对这个和做两次取反运算。

请注意，因为在正常情况下计算出的检验和的值永远不会是全 1，因此就不会导致混淆（参见下例）。

153.18.8.105		
171.2.14.10		
全0	17	15
1087	13	
15	全0	
T	E	S
I	N	G
填充		

10011001 00010010	→ 153.18
00001000 01101001	→ 8.105
10101011 00000010	→ 171.2
00001110 00001010	→ 14.10
00000000 00010001	→ 0 和 17
00000000 00001111	→ 15
00000100 00111111	→ 1087
00000000 00001101	→ 13
00000000 00001111	→ 15
00000000 00000000	→ 0 (检验和)
01010100 01000101	→ T 和 E
01010011 01010100	→ S 和 T
01001001 01001110	→ I 和 N
01000111 00000000	→ G 和 0 (填充)
10010110 11101011	→ 和
01101001 00010100	→ 检验和

图 14.4 一个简单的 UDP 用户数据报检验和的计算

例 14.3

在以下几种假设的情况下，检验和在发送时的值分别是多少？

- a. 发送方决定不包括检验和。
- b. 发送方决定包括检验和，但是这个和的值全部为 1。
- c. 发送方决定包括检验和，但是这个和的值全部为 0。

解

- a. 发送时检验和字段的值是全 0，以表示这个检验和不是计算出来的。
- b. 当发送方对这个和取反时，结果就变成了全 0。于是发送方在发送之前再次对这个结果取反。这个检验和的发送值就是全 1。第二次取反操作之所以需要，是为了避免与 a 的情况混淆。
- c. 这种情况永远不可能发生，因为它暗示着用于计算检验和的每一项的值全部都是 0，这是不可能的，在伪首部中就有一些字段非零（参见附录 D）。

14.3.5 拥塞控制

因为 UDP 是无连接协议，所以它不提供拥塞控制。UDP 假设发送的分组不大，并且分散发送，所以不可能造成网络的拥塞。但如今，随着 UDP 被用于音频和视频的实时传输，这种假设有可能不再成立。

14.3.6 封装和解封

要把报文从一个进程发送到另一个进程，UDP 协议就要对报文进行封装和解封（参见图 14.5）。

封装

当进程有报文要通过 UDP 发送时，它就把这个报文连同一对套接字地址以及数据的长

度传递给 UDP。UDP 收到数据后添加一个 UDP 首部。然后，UDP 把这个用户数据报连同套接字地址一起传递给 IP。IP 再加上自己的首部，在协议字段使用值 17，指出该数据是从 UDP 协议来的。然后再把这个 IP 数据报传递给数据链路层。数据链路层收到 IP 数据报后，加上自己的首部（可能还有尾部），再传递给物理层。物理层把比特编码成电信号或光信号，并将其发送到远程的机器。

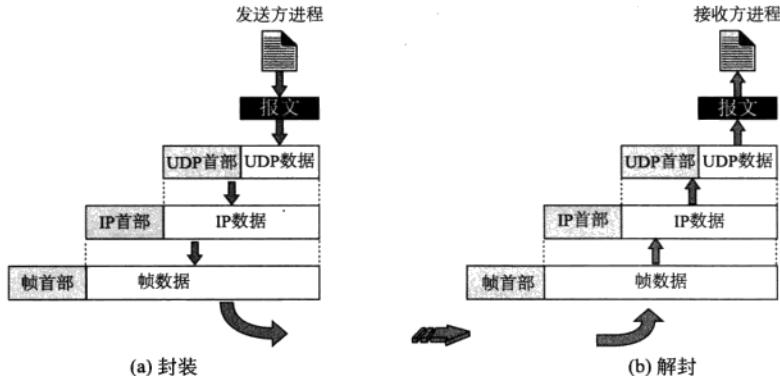


图 14.5 封装和解封

解封

当这个报文到达目的主机后，物理层首先将信号解码变成比特，传递给数据链路层。数据链路层利用首部（与尾部一起）检查数据。若无差错，就剥去首部和尾部，并把数据报传递给 IP。IP 软件进行它自己的检查。若无差错，就剥去首部，把用户数据报连同发送方和接收方的 IP 地址一起传递给 UDP。UDP 使用检验和对整个用户数据报进行检查。若无差错，则剥去首部，把应用数据连同发送方的套接字地址一起传递给接收进程。之所以要把发送方的套接字地址也传递给进程是因为有可能需要对收到的报文进行响应。

14.3.7 排队

我们已经谈过端口，但没有讨论端口是如何实现的。在 UDP 中，队列与端口相关联（见图 14.6）。

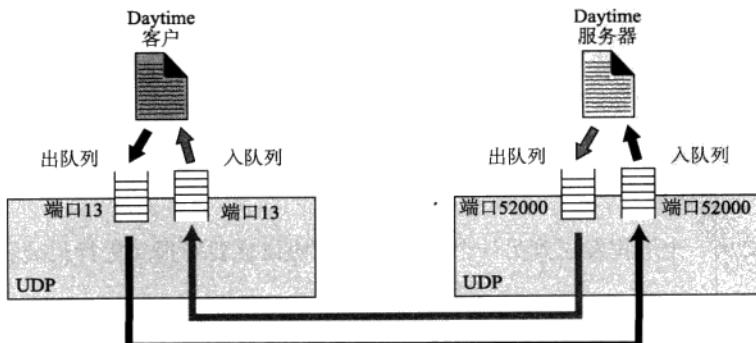


图 14.6 UDP 中的队列

在客户端，当一个进程启动时，它从操作系统那里请求得到一个端口号。有些实现为每个进程创建一个入队列和一个出队列与之关联。另外一些实现则只创建与每一个进程相关联的入队列。

请注意，哪怕一个进程希望和多个进程通信，它也只能得到一个端口号，并且最终只有一个出的和一个入的队列（queue）。在大多数情况下，由客户打开的队列由临时端口号标志。只要进程在运行，这些队列就能起作用。当进程终止时，队列被销毁。

客户进程使用在请求中指明的源端口号就可以把报文送到出队列。UDP 逐个地把报文取出，先添加 UDP 首部，再交付给 IP。出队列有可能出现溢出。若发生溢出，操作系统就要求客户进程在继续发送报文之前先等待。

当一个报文到达客户端时，UDP 首先要检查一下，看看这个用户数据报中的目的端口号字段指明的端口号所对应的入队列是否已创建。若已经有了这样的队列，UDP 就把收到的用户数据报放在该队列的末尾。若没有这样的队列，UDP 就丢弃这个用户数据报，并请求 ICMP 协议向服务器端发送端口不可达报文。所有发送给某个特定客户程序的入报文，不管是来自相同的或者是不同的服务器，都被放入同一个队列中。入队列有可能会溢出。若发生溢出，UDP 就丢弃这个用户数据报，并请求向服务器发送一个端口不可达报文。

在服务器端，创建队列的机制是不同的。在最简单的形式下，服务器进程在它开始运行时就请求为它的熟知端口创建入队列和出队列。只要服务器进程在运行，这些队列就一直是打开的。

当报文到达服务器进程时，UDP 首先要检查一下，这个用户数据报中的目的端口号字段指明的端口号所对应的入队列是否已创建。若已经有这样的队列，UDP 就把收到的用户数据报放在这个队列的末尾。若没有这样的队列，UDP 就丢弃这个用户数据报，并请求 ICMP 协议向客户端发送端口不可达报文。所有发送给某个特定服务器程序的入报文，不管是来自相同的或者是不同的客户，都被放入同一个队列。入队列有可能会溢出。若发生溢出，UDP 就丢弃这个用户数据报，并请求向客户发送端口不可达报文。

当服务器想要回答客户时，它就用在请求中指明的源端口号把报文送到出队列。UDP 逐个地把报文取出，先添加 UDP 首部，再交付给 IP。出队列有可能出现溢出。若发生溢出，操作系统就要求服务器进程在继续发送报文之前要等待。

14.3.8 复用和分用

在运行 TCP/IP 协议族的主机上只有一个 UDP，但可能会有多个进程希望使用 UDP 服务。要处理这种情况，UDP 可以进行复用和分用（参见图 14.7）。

复用

在发送端，可能有多个进程需要发送用户数据报。但是，只有一个 UDP。这是多对一的关系，因而需要复用。UDP 接受来自不同的进程的报文，这些进程通过指派给它们的端口号来区分。在添加了首部之后，UDP 就把用户数据报送往 IP。

分用

在接收端也只有一个 UDP。但我们可能有多个进程都接收用户数据报。这是一对多的关系，因而需要分用。UDP 接受来自 IP 的用户数据报。经过差错检查并剥除首部后，UDP

根据端口号把每一个报文交付到适当的进程。

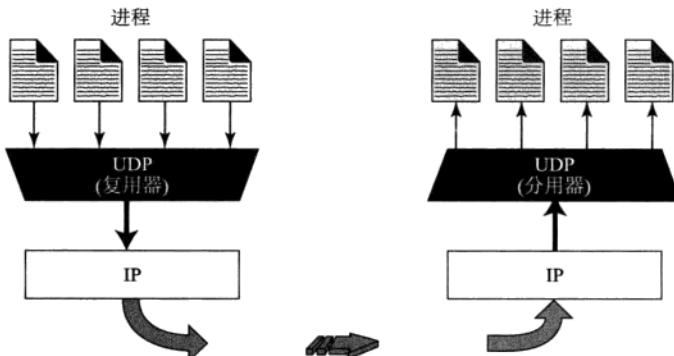


图 14.7 复用和分用

14.3.9 UDP 与简单协议的比较

我们可以将 UDP 与第 13 章中讨论的无连接简单协议进行比较。它们唯一的区别就在于 UDP 提供了可选的检验和，以便于在接收端检测损坏的分组。如果分组加上了检验和，那么接收方 UDP 就能够检测分组，并丢弃损坏的分组。但是对此没有任何反馈信息给发送方。

UDP 是我们在第 13 章讨论的无连接简单协议的一个例子，只有一个例外的地方，就是它还在分组上增加了一个可选的检验和用于差错检测。

14.4 UDP 的应用

虽然 UDP 几乎不能满足我们在第 13 章中所讨论的一个可靠的运输层协议的任何标准，但是对某些应用来说，它却是受欢迎的。原因在于某些服务可能会带来一些副作用，这些副作用或者不可接受，或者不受欢迎。应用程序的设计者有时需要让步以得到最优化。例如，在我们的日常生活中，我们都知道邮局递交包裹时，当天到达要比三天到达贵得多。虽然对于邮寄一个包裹来说，越快且花钱越少越好，但这两个方面是矛盾的。我们需要选择一个最优方案。

在这一小节，我们首先要讨论在设计应用程序时可能需要考虑的一些 UDP 的特点，然后再展示一些典型的应用。

14.4.1 UDP 的特点

我们简单地讨论 UDP 的一些特点以及它们的优缺点。

无连接服务

正如我们在前面提到的，UDP 是一个无连接协议。一个 UDP 分组与同一个应用程序

发送的其他分组之间都是互相独立的。这个特点可以被认为是一个优点，也可以被认为是一个缺点，完全取决于应用的需求。例如，如果客户应用需要向服务器发送一个简短的请求，并希望收到一个简短的响应，那么它就是一个优点。如果每个请求和响应都能够放进一个用户数据报中，那么无连接的服务也许比较合适。在这种情况下，建立连接和关闭连接的额外开销所占的比重不可忽视。在使用面向连接的服务时，仅为完成以上任务，在客户与服务器之间至少需要交换 9 个分组，而在无连接的服务时则只需要交换两个分组。无连接服务的时延较短，而面向连接的服务则产生了更多的时延。如果对于某个应用，时延是重点要考虑的，那么无连接的服务更加适合。

例 14.4

像 DNS（参见第 19 章）这样的客户-服务器应用使用了 UDP 的服务，原因是客户需要向服务器发送一个短小的请求，并希望接收来自服务器的快速响应。这个请求和响应都能被装进一个用户数据报中。因为在每个方向上只交换一个报文，所以有没有连接就不成问题，客户或服务器都不担心报文是否按顺序交付。

例 14.5

像 SMTP（参见第 23 章，用在电子邮件中）这样的客户-服务器应用就不能使用 UDP 的服务，因为用户可能会发送一个较长的电子邮件报文，其中还可能包括了多媒体（图片、音频或视频）。如果这个应用使用 UDP，而一个用户数据报又装不下这个报文，那么这个报文必须被应用程序分割成若干个不同的用户数据报。此时无连接的服务就有可能出现问题。用户数据报可能会失序到达并交付给接收方的应用。接收方的应用可能无法为这些片断重新排序。这就表示无连接的服务对于发送长报文的应用程序来说是有缺陷的。在 SMTP 中，一方在发送报文后并不期望立刻收到响应（有时连响应都需要），也就是说使用面向连接的服务所不可避免的额外时延对 SMTP 来说并不重要。

缺少差错控制

UDP 不提供差错控制，它提供的是不可靠服务。绝大多数应用程序都期望运输层协议能提供可靠的服务。虽然可靠的服务很受欢迎，但它也有一些副作用是某些应用所不能接受的。当运输层提供可靠服务时，如果报文的一部分丢失或损坏了，它就必须被重传。这就是说接收方的运输层不能即时将该部分交付给应用层，因此这个报文的各个部分在交付时就会存在不一致的时延。有些应用本来就不会去注意这些时延是否一致，但对有些应用来说，它们很关键。

例 14.6

假设我们正在从因特网上下载一个很大的文本文件。我们肯定要使用提供可靠服务的运输层协议。当我们打开这个文件时不希望这个文件有一部分丢失或损坏。而在交付文件的各个部分时产生的不同的时延并不会引起高度重视，因为我们在查看文件之前会等待形成完整的文件。在这种情况下，UDP 就不是合适的运输层协议。

例 14.7

假设我们正在计算机上观看一个实时的流视频。这种程序被认为是一个很长的文件，它被分割成很多文件块并实时播送。这些文件块一个接一个地发送出去。如果运输层认为应当重传损坏或丢失的帧，那么整个传输的同步性就会丢失。观众会突然看到一个空屏，然后需要等到第二次的传输到达。这是无法忍受的。但是，如果每一小块屏幕内容都用一

个用户数据报来发送，那么接收方 UDP 就能简单地忽略损坏或丢失的分组，并将其他分组交付给应用程序。部分屏幕可能会有短暂的空白，这对大多数观众来说甚至都不会注意到。但是，视频是不能不按顺序观看的，所以运行在 UDP 上的流音频、流视频和声音应用程序必须进行重新排序或丢弃失序到达的帧。

缺少拥塞控制

UDP 不提供拥塞控制。但是 UDP 不会给故障频出的网络带来额外的通信量。TCP 可能会多次重传一个分组，这也是产生拥塞或加重拥塞状况的一个原因。因此，在这种情况下，当拥塞成为一个大问题时，缺少差错控制的 UDP 也可以被认为是一个优点。

14.4.2 典型应用

下面列出的是使 UDP 的服务要比使用 TCP 的服务受益更大的典型应用：

- UDP 适用于只要求简单的请求–响应通信的进程，它们很少会考虑流量控制和差错控制。对于需要传送大量数据的进程，如 FTP（见第 21 章），通常不使用 UDP。
- UDP 适用于具有内部流量控制和差错控制机制的进程。例如，简单文件传送协议（TFTP）（见第 21 章）的进程就包括流量控制和差错控制。它能够很容易地使用 UDP。
- 对多播来说，UDP 是个合适的运输协议。多播能力已经嵌入在 UDP 软件中，但没有嵌入在 TCP 软件中。
- UDP 适用于管理进程，如 SNMP（见第 24 章）。
- UDP 适用于某些路由选择更新协议，如路由信息协议（RIP）（见第 11 章）。
- UDP 通常适用于实时应用，它们不能容忍在接收报文的各个片断之间存在变化的时延。

14.5 UDP 软件包

我们通过 UDP 软件包的简化版本来说明 UDP 怎样处理 UDP 分组的发送和接收。

我们可以认为 UDP 软件包涉及到五个构件：一个控制块表、若干个输入队列、一个控制块模块、一个输入模块和一个输出模块。图 14.8 给出了这五个构件以及它们之间的交互。

14.5.1 控制块表

在我们的软件包中，UDP 用控制块表来记录打开的端口。表中的每一项至少有四个字段：状态（可以是 FREE 或 IN-USE）、进程 ID、端口号以及相应的队列号。

14.5.2 输入队列

我们的 UDP 软件包使用了一组输入队列，一个队列对应于一个进程。在这个设计中，我们没有使用输出队列。

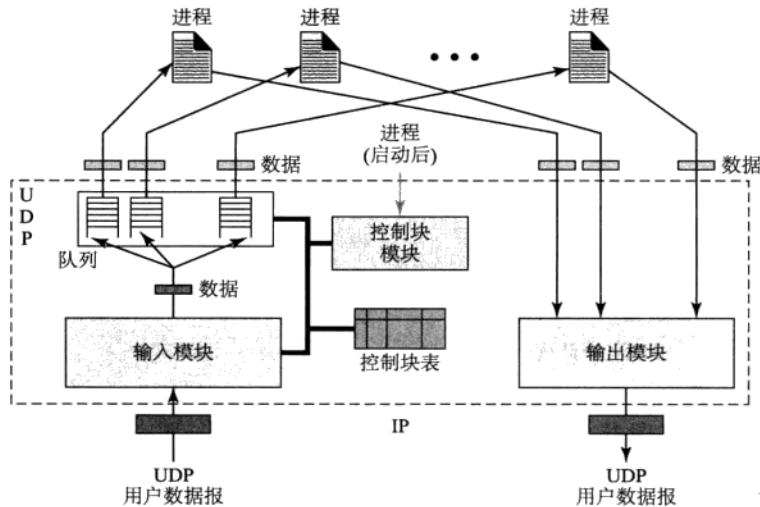


图 14.8 UDP 的设计

14.5.3 控制块模块

控制块模块（表 14.2）负责管理控制块表。当一个进程启动时，它就向操作系统请求得到一个端口号。操作系统为服务器指派一个熟知端口号，为客户指派一个临时端口号。进程把进程 ID 和端口号传递给控制块模块，以便在表中为这个进程创建一个表项。这个模块不创建队列。队列号的字段值是零。请注意，我们这里没有考虑当表已满时应采取什么策略。

表 14.2 控制块模块

```

1   UDP_Control_Block_Module (process ID, Port Number)
2   {
3       查找控制块表中的 FREE 表项
4       If(未找到)
5           使用事先定义的策略删除一个表项
6           创建状态为 IN-USE 的新表项
7       输入进程 ID 和端口号
8       返回
9   }    //模块结束

```

14.5.4 输入模块

输入模块（表 14.3）从 IP 接收用户数据报。它搜索控制块表，查找具有和这个用户数据报同样端口号的表项。若找到这样的表项，模块就利用该表项中的信息把数据放入队列。若未找到这样的表项，它就产生一个 ICMP 报文。

表 14.3 输入模块

```

1      UDP_INPUT_Module (user_datagram)
2      {
3          在控制块表中查找相应的表项
4          If(找到)
5          {
6              检查队列字段，看是否已经分配了一个队列
7              If(队列未分配)
8                  分配一个队列
9              Else
10                 把数据放入相应的队列中
11             } //end if
12         Else
13         {
14             请求 ICMP 模块发送“端口不可达”报文
15             丢弃这个用户数据报
16         } //end if
17
18         返回
19     } //模块结束

```

14.5.5 输出模块

输出模块（表 14.4）负责创建和发送用户数据报。

表 14.4 输出模块

```

1      UDP_OUTPUT_Module (Data)
2      {
3          创建用户数据报
4          发送用户数据报
5          返回
6      }

```

14.5.6 举例

本节将给出几个例子说明我们的软件包是如何对输入和输出进行响应的。在我们的例子开始时的控制块表如表 14.5 所示。

表 14.5 在例子开始时的控制块表

状态	进程 ID	端口号	队列号
IN-USE	2345	52010	34
IN-USE	3422	52011	
FREE			
IN-USE	4652	52012	38
FREE			

例 14.8

第一件事就是到达了目的端口号为 52012 的用户数据报。输入模块查找这个端口号后发现了相应表项。给这个端口指派的队列号是 38，也就是说这个端口以前已经使用过了。输入模块把数据发送到队列 38。控制块表没有变化。

例 14.9

几秒钟后，有一个进程启动了。它向操作系统请求端口号，并得到了 52014 的端口号。现在这个进程把它的 ID (4978) 和这个端口号一起递交给控制块模块以便在表中创建一个表项。这个模块获取第一个 FREE 表项并将它接收到的信息插入其中。此刻，模块还没有分配队列，因为还没有以此为终点的用户数据报到达（见表 14.6）。

表 14.6 例 14.9 的控制块表

状态	进程 ID	端口号	队列号
IN-USE	2345	52010	34
IN-USE	3422	52011	
IN-USE	4978	52014	
IN-USE	4652	52012	38
FREE			

例 14.10

现在有一个端口号为 52011 的用户数据报到达。输入模块检查这张表，发现该终点尚未分配队列，因为这是以此为终点的用户数据报的第一次到达。这个模块就创建一个队列，并分配给它一个编号 (43)，见表 14.7。

表 14.7 例 14.10 的控制块表

状态	进程 ID	端口号	队列号
IN-USE	2345	52010	34
IN-USE	3422	52011	43
IN-USE	4978	52014	
IN-USE	4652	52012	38
FREE			

例 14.11

几秒钟后，一个端口号为 52222 的用户数据报到达。输入模块检查这张表，找不到对应于这个终点的表项。这个用户数据报被丢弃，并请求 ICMP 向源点发送“端口不可达”报文。

14.6 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC 文档。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

14.6.1 参考书

有几本书讨论了 UDP。我们特别推荐[Com 06]和[Ste 94]。

14.6.2 RFC

与 UDP 相关的主要 RFC 是 RFC 768。

14.7 重要术语

无连接、不可靠的运输协议

用户数据报

伪首部

用户数据报协议（UDP）

队列

14.8 本章小结

- UDP 是运输层协议，它建立进程到进程的通信。UDP（基本上）是一个不可靠的无连接协议，它只需要很少的额外开销，并且能够很快地交付。UDP 分组称为用户数据报。
- UDP 在差错控制方面仅有的尝试就是检验和。在计算检验和时包括一个伪首部可以检查出源 IP 地址和目的 IP 地址的差错。UDP 没有流量控制机制。
- 用户数据报封装在 IP 数据报的数据字段中。入队列和出队列存放着要进入 UDP 的和来自于 UDP 的报文。
- UDP 利用复用技术处理来自一台主机上的多个进程的用户数据报的发送。UDP 利用分用技术处理要交给一台主机上的多个不同进程的入口用户数据报。
- UDP 软件包由五个构件组成：一个控制块表、一个控制块模块、若干个输入队列、

一个输入模块和一个输出模块。输入队列存放入口用户数据报。控制块模块负责维护控制块表中的表项。输入模块创建输入队列。输出模块把用户数据报发送出去。

14.9 实践安排

14.9.1 习题

1. 在可靠性不是最重要的情况下, UDP 可能是一个很好的运输协议。试给出这种特定情况的一些具体的例子。
2. UDP 和 IP 是否不可靠程度相同? 为什么是或为什么不是?
3. 试给出携带了从 TFTP 客户发送到 TFTP 服务器的报文的 UDP 用户数据报首部中的各项。检验和字段填入 0。试选择一个适当的临时端口号和一个正确的熟知端口号。数据的长度为 40 字节。使用图 14.2 的格式来表示这个 UDP 分组。
4. 在 IP 地址为 122.45.12.7 的主机上有一个 SNMP 客户, 它向 IP 地址为 200.112.45.90 的主机上的 SNMP 服务器发送报文。在它们的通信中使用的一对套接字是什么?
5. 在 IP 地址为 130.45.12.7 的主机上的 TFTP 服务器, 向 IP 地址为 14.90.90.33 的主机上的 TFTP 客户发送报文。在它们的通信中使用的一对套接字是什么?
6. 回答以下问题:
 - a. UDP 数据报的最小长度是多少?
 - b. UDP 数据报的最大长度是多少?
 - c. 能够封装在一个 UDP 数据报中的进程数据的最小长度是多少?
 - d. 能够封装在一个 UDP 数据报中的进程数据的最大长度是多少?
7. 一个客户进程使用 UDP 向服务器发送数据。数据长度为 16 字节。试计算 UDP 级的传输效率 (有用字节数与总字节数之比)。
8. 重做习题 7, 计算 IP 级的传输效率。假定 IP 首部无选项。
9. 重做习题 7, 计算数据链路级的传输效率。假定 IP 首部无选项, 在数据链路层使用以太网。
10. 下面是十六进制格式的 UDP 首部。
0045DF000058FE20
a. 源端口号是什么?
b. 目的端口号是什么?
c. 用户数据报的总长度是多少?
d. 数据的长度是多少?
e. 该分组是从客户发送到服务器还是相反方向?
f. 客户进程是什么?

第 15 章 传输控制协议（TCP）

正如我们在第 14 章中讨论的，有几个协议已被指定为 TCP/IP 协议族的运输层协议。我们在这一章要讨论 TCP。TCP 是这个协议族的核心，它提供了大量的服务，因此这是一个复杂的协议。TCP 在过去二三十年中又经历了数个版本的发展。这就意味着本章会很长。

目标

本章有以下几个目标：

- 介绍 TCP 这个能够提供可靠的流交付服务的协议。
- 定义 TCP 的特点并与 UDP 的特点进行比较。
- 定义 TCP 报文段的格式及其字段。
- 说明 TCP 如何提供面向连接的服务，并描述在连接建立和连接终止阶段交换的报文段。
- 讨论 TCP 的状态转换图并研究一些实现情况。
- 介绍在 TCP 中用于流量控制和差错控制的窗口。
- 讨论 TCP 如何让接收窗口来控制发送窗口大小以实现流量控制。
- 讨论 TCP 在数据传输阶段使用的差错控制及其 FSM。
- 讨论 TCP 如何利用多种不同的策略来控制网络中的拥塞。
- 列举并解释 TCP 中各个计时器的作用。
- 讨论 TCP 的选项，并说明 TCP 是如何通过使用 SACK 选项来提供选择确认的功能。
- 给出一个 TCP 软件包的总体结构以及一个简化的伪码实现。

15.1 TCP 服务

图 15.1 描绘了 TCP 与 TCP/IP 协议族中其他协议的关系。TCP 位于应用层和网络层之间，它提供介于应用程序和网络功能之间的服务。

在我们详细研究 TCP 之前，我们要先解释 TCP 为应用层的进程所提供的服务。

15.1.1 进程到进程的通信

与 UDP 一样，TCP 也使用端口号（见第 13 章）提供进程到进程的通信。表 15.1 给出

了 TCP 使用的一些熟知端口号。

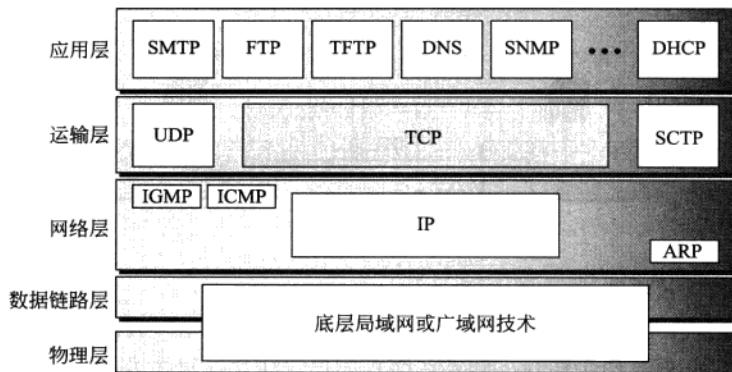


图 15.1 TCP/IP 协议族

表 15.1 TCP 使用的熟知端口号

端口	协议	说明
7	Echo	把收到的数据报回送到发送方
9	Discard	丢弃收到的任何数据报
11	Users	活跃的用户
13	Daytime	返回日期和时间
17	Quote	返回日期的引用（译注：可参阅 RFC 865）
19	Chargen	返回字符串
20 和 21	FTP	文件传送协议（数据和控制）
23	TELNET	终端网络
25	SMTP	简单邮件传送协议
53	DNS	域名服务器
67	BOOTP	引导程序协议
79	Finger	Finger
80	HTTP	超文本传送协议

15.1.2 流交付服务

和 UDP 不同，TCP 是一种面向流的协议。在 UDP 中，进程把已预先定义好边界的报文发送给 UDP 以便进行交付。UDP 对每个报文都添加上自己的首部，然后再把它们传递给 IP 来传输。从进程发来的报文称为用户数据报，并最终成为 IP 数据报。不论是 UDP 还是 IP，都不认为这些数据报之间存在任何的关联。

与此不同的是，TCP 则允许发送进程以字节流的形式来传递数据，并且也允许接收进程把数据作为字节流来接收。TCP 创造了一种环境，它使得两个进程好像被一个假想的“管

道”所连接，而这个管道经过因特网传输着两个进程之间的数据。这个假想的环境如图 15.2 所描绘。发送进程产生（写入）字节流，而接收进程消耗（读取）字节流。

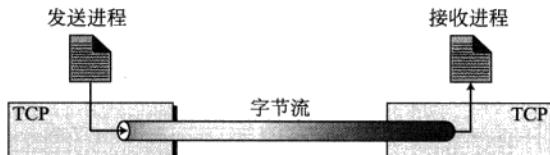


图 15.2 流的交付

发送缓存和接收缓存

因为发送进程和接收进程可能以不同的速度写入数据和读取数据，因此 TCP 需要用缓存来存储数据。此处有两个缓存，即发送缓存和接收缓存，每个方向各一个。在后面我们将会看到，这些缓存还被 TCP 用来进行流量控制和差错控制。缓存的一种实现方法是使用由 1 字节位置组成的环形阵列，如图 15.3 所示。为简单起见，我们描绘的这两个缓存都是 20 字节的，实际上这些缓存通常都是几百或几千字节，取决于具体的实现。我们还假定这两个缓存大小一样，但这并不是一定的。

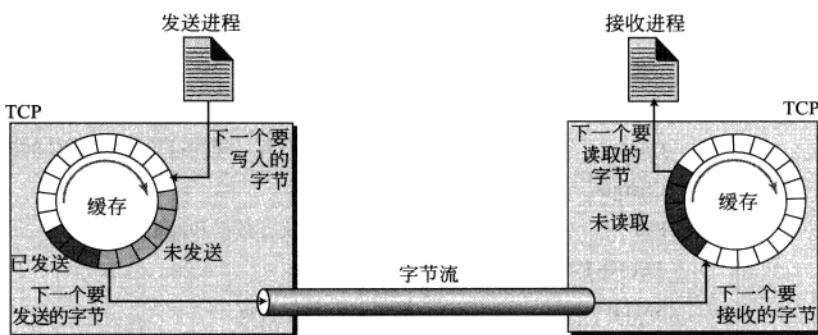


图 15.3 发送和接收缓存

图中描绘的是单向的数据移动过程。在发送方，该缓存有三种类型的槽。白色区域包含的是空槽，允许发送进程（生产者）填入数据。深灰色区域保存的是已经发送出去但还没有得到确认的字节。发送 TCP 在缓存中保存这些字节，直至收到相应的确认。灰色区域表示发送 TCP 将要发送的字节。但是，正如我们将在本章稍后将会看到的，TCP 可能只被允许发送灰色区域中的一部分数据。有可能是因为接收进程的速度较慢，也有可能是因为网络中发生了拥塞。还应当注意到，当深灰色槽中的字节被确认后，这些槽可以被回收并被发送进程再次利用。这也是为什么我们要用环形来表示缓存的原因。

接收方的缓存操作较为简单。环形缓存被划分为两个区域（白色和灰色）。白色区域包含的空槽将被从网络中接收到的字节填入。灰色区域包含的是已经接收到的字节，这些字节将被接收进程读取。当一个字节被接收进程读取后，相应的槽就可以被回收，并加入到空槽池中。

报文段

虽然可以用缓存来处理生产进程和消耗进程在速度上的差异，但我们在发送数据之前

还需要一个步骤。IP 层作为 TCP 的服务提供者，它必须以分组为单位发送数据，而不是按字节流来发送。在运输层，TCP 把若干字节组成一个分组，称为报文段。TCP 给每个报文段添加一个首部（用于控制），然后再把这个报文段交付给 IP 层传输。这些报文段被封装成 IP 数据报后发送出去。这整个过程对接收进程来说都是透明的。稍后我们将会看到，这些报文段在接收时有可能会失序、丢失，或受到损伤和重传，所有这些都是由 TCP 来处理的，而接收进程并不知道 TCP 的这些活动。图 15.4 描绘了如何用缓存中的字节构成报文段。

请注意，这些报文段并不一定长度相同。在图中，为简单起见，我们描绘了一个报文段有 3 字节，而另一个报文段有 5 字节。实际上，报文段可以是几百字节长，甚至是几千字节长。

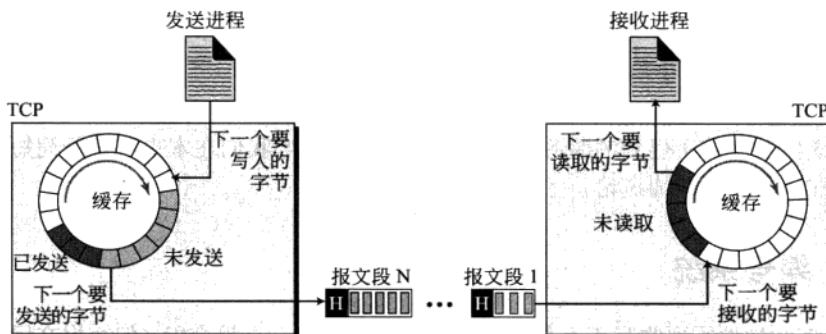


图 15.4 TCP 的报文段

15.1.3 全双工通信

TCP 提供全双工服务，即数据可在同一时间双向流动。TCP 的两个端点分别有自己的发送缓存和接收缓存，报文段可以在两个方向运动。

15.1.4 复用和分用

与 UDP一样，TCP 在发送端执行复用并在接收端执行分用。但是，因为 TCP 是一个面向连接的协议，所以每一对进程都需要建立一条连接。当我们在第 17 章讨论客户-服务器范式时，这一点会更加清楚。

15.1.5 面向连接的服务

和 UDP 不同，TCP 是面向连接的协议。如第 13 章中所描述的，当站点 A 的一个进程想和站点 B 的另一个进程交换数据时，需要经过以下三个阶段：

1. 这两个 TCP 在它们之间建立一条虚连接。
2. 数据在两个方向上交换。
3. 连接被终止。

请注意，这是一条虚连接而不是一条物理连接。 TCP 报文段被封装成 IP 数据报后，有可能在发送时会失序，或丢失，或损坏并被重传。每一个 IP 数据报可以走不同的路径到达终点。这里没有物理连接。TCP 创建了面向流的环境，它负责把这些字节按顺序交付到终点。

15.1.6 可靠的服务

TCP 是一个可靠的运输协议。它使用确认机制来检查数据是否安全完好地到达。我们将在差错控制这一节进一步讨论这个特点。

15.2 TCP 的特点

为了提供前一节所提到的服务，TCP 有一些特点需要我们在本小节进行简短的归纳，这些特点在以后还要详细讨论。

15.2.1 编号系统

虽然 TCP 软件需要掌握正在传送的或已接收到的每一个报文段，但在报文段首部中并没有存放报文段编号值的字段。实际上，在这个首部中有两个叫做序号和确认号的字段。
这两个字段所指的都是字节的编号而不是报文段的编号。

字节号

TCP 把在一个连接中要发送的所有数据字节（八位组）都编上号。两个方向的编号是相互独立的。当 TCP 接收来自进程的数据字节时，就把它存储在发送缓存中，并为它们进行编号。编号不一定要从 0 开始。TCP 选择 $0 \sim (2^{32}-1)$ 之间的一个随机数作为第一个字节的编号。例如，若这个数恰巧是 1057，而要发送的数据总共有 6000 字节，那么这些字节的编号就是从 1057~7056。我们将会看到，在流量控制和差错控制中都要用到字节编号。

每条连接上传送的数据字节都被 TCP 编了号。编号从一个随机产生的数开始。

序号

当字节都被编上号以后，TCP 就给每一个要发送的报文段指派一个序号。每个报文段的序号就是这个报文段中第一个数据字节的序号。

例 15.1

假设一条 TCP 连接要传送一个 5000 字节的文件。第一个字节的编号是 10001。如果该数据用 5 个报文段来发送，且每个报文段携带 1000 字节的数据，那么每个报文段的序号分别是什么？

解

下面给出每一个报文段的序号：

报文段 1	→	序号: 10001 (范围: 10001~11000)
报文段 2	→	序号: 11001 (范围: 11001~12000)
报文段 3	→	序号: 12001 (范围: 12001~13000)
报文段 4	→	序号: 13001 (范围: 13001~14000)
报文段 5	→	序号: 14001 (范围: 14001~15000)

报文段的序号字段值定义的是这个报文段包含的第一个数据字节所分配的编号。

当一个报文段同时携带了数据和控制信息（捎带）时，它使用一个序号。但如果报文段不携带用户数据，从逻辑上讲，它就不定义序号。序号字段总是存在的，但它的值是无意义的。不过某些仅携带控制信息的报文段还是需要有一个序号，以便于接收方的确认。这些报文用于连接建立、连接终止和连接异常终止。这每一个报文段都要消耗一个序号，就好像它携带了一个字节的数据，但实际上是没有数据的。我们将会在讨论连接时再来详细讨论这个问题。

确认号

正如我们前面讨论过的，TCP 的通信是全双工的。当一条连接建立后，双方能同时发送和接收数据。通常双方从不同的起始号开始对字节编号。每一个方向上的序号表示的是该方向的报文段所携带的第一个数据字节的编号。双方还使用了确认号对各自收到的字节表示确认，不过这个确认号定义的是它期望接收的下一个字节的编号。另外，确认号是累积的，也就是说，它把收到的最后一个（安全完好）字节的编号加上 1 所得到的值宣布为确认号。在这里术语累积的意思是指，如果某一方使用 5643 作为确认号，那就表示它已经收到了从开始一直到编号为 5642 的所有字节。请注意，并不是说这一方已经收到了 5642 个字节，因为第一个字节的编号不一定是从 0 开始。

报文段中确认字段的值定义了某一方期望接收的下一个字节的编号。

确认号是累积的。 → 类似 GBN...

15.2.2 流量控制

和 UDP 不同，TCP 提供了流量控制。发送 TCP 要对能够接受多少从发送进程传来的数据进行控制，接收 TCP 则要对发送 TCP 能够发送多少数据进行控制（参见第 13 章）。这样做是为了防止接收方因数据过多而来不及处理。编号系统使得 TCP 能够使用面向字节的流量控制，我们将在本章后面的内容中再讨论。

15.2.3 差错控制

为了提供可靠的服务，TCP 需要实现差错控制机制。虽然差错控制把报文段看作是差错检测的数据单元（报文段丢失或受损伤），但正如我们以后会看到的，TCP 的差错控制是面向字节的。

15.2.4 拥塞控制

和 UDP 不同, TCP 要考虑到网络的拥塞状况。发送方允许发送的数据量不仅要受接收方的控制(流量控制),而且还要由网络的拥塞状况(如果有的话)来决定。

15.3 报文段

在更详细地讨论 TCP 之前,让我们先介绍 TCP 分组的构成。TCP 的分组称为报文段(segment)。

15.3.1 格式

报文段的格式如图 15.5 所示。这个报文段包括了 20~60 字节的首部,紧随其后的是从应用程序传来的数据。首部在没有选项时是 20 字节,如果有选项则最多可达 60 字节。在本小节,我们将讨论首部中的一些字段。随着本章内容的不断深入,这些字段的含义及目的会变得越来越清楚。

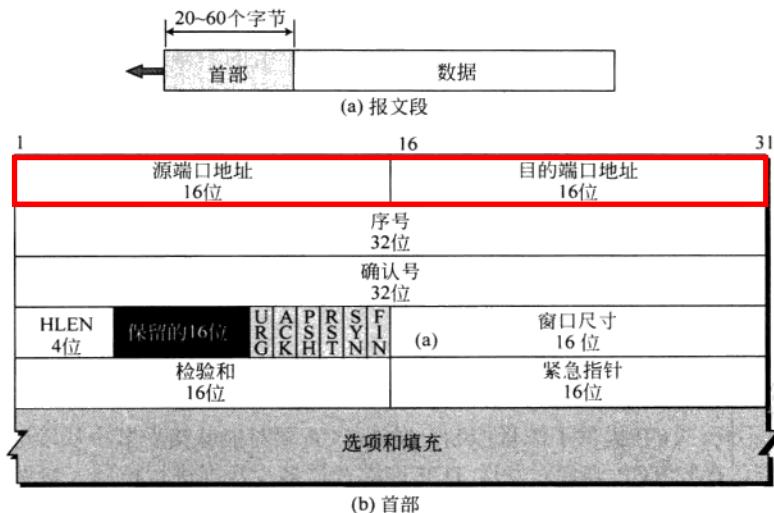


图 15.5 TCP 报文段的格式

- **源端口地址** 这是一个 16 位的字段,定义了发送这个报文段的主机中的应用程序的端口号。这和第 14 章讨论的 UDP 首部中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,定义了接收这个报文段的主机中的应用程序的端口号。这和第 14 章讨论的 UDP 首部中的目的端口地址的作用一样。
- **序号** 这是一个 32 位的字段,定义了指派给本报文段第一个数据字节的编号。如我们在前面所讲的,TCP 是流运输协议。为了保证连接性,要发送的每一个字节都

要编上号。序号可以告诉终点，报文段中的第一个字节是这个序列中的哪一个字节。在连接建立时（稍后讨论），双方使用各自的随机数产生器产生一个初始序号（initial sequence number, ISN）。通常，两个方向上的 ISN 是不同的。

- **确认号** 这个 32 位字段定义了报文段的接收方期望从对方接收的字节编号。如果报文段的接收方成功地接收了对方发来的编号为 x 的字节，那么它就返回 $x + 1$ 作为确认号。确认可以和数据捎带在一起发送。
- **首部长度** 这个 4 位字段指出 TCP 首部一共有多少个 4 字节字。首部长度可以在 20~60 字节之间。因此，这个字段的值可以在 5 ($5 \times 4 = 20$) 到 15 ($15 \times 4 = 60$) 之间。
- **保留** 这是一个 6 位字段，保留为今后使用。
- **控制** 这个字段定义了 6 种不同的控制位或标志，如图 15.6 所示。在同一时间可设置一位或多为标志。这些位用在 TCP 的流量控制、连接建立和终止、连接异常终止以及数据传送方式等方面。图中给出了每一位的简要说明。我们将在本章稍后讨论 TCP 的工作细节时再进一步介绍它们。

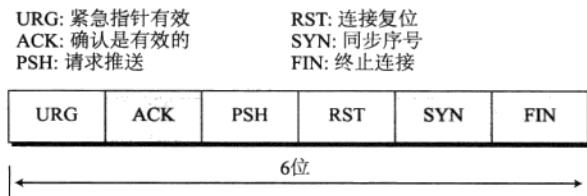


图 15.6 控制字段

- **窗口大小** 这个字段定义的是发送 TCP 的窗口大小，以字节为单位。请注意这个字段是 16 位，也就是说窗口的最大长度是 65535 字节。这个值通常被称为接收窗口（rwnd），并由接收方来决定。在这种情况下，发送方必须服从接收方的指示。
- **检验和** 这个 16 位字段包含的是检验和。TCP 检验和的计算与第 14 章讨论的 UDP 检验和的计算过程一样。但是，UDP 是否使用检验和是可选的，而 TCP 使用检验和则是强制性的。基于同样的原因，在计算检验和时报文段要附加相同的伪首部。但对于 TCP 的伪首部，协议字段的值是 6，见图 15.7。

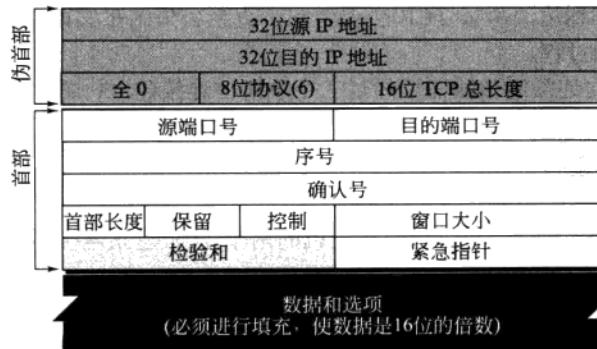


图 15.7 伪首部加到 TCP 数据报上

TCP 检验和的使用是强制性的。

- **紧急指针** 只有当紧急标志置位时，这个 16 位的字段才有效，此时报文段中包含了紧急数据。紧急指针定义了一个数值，把这个数值加到序号上就得出报文段数据部分中最后一个紧急字节的编号。这在本章后面的内容中将会讨论到。
- **选项** 在 TCP 首部中可以有多达 40 字节的可选信息。我们将在本章稍后讨论目前使用的 TCP 首部中的各个选项。

15.3.2 封装

TCP 报文段封装了从应用层接收到的数据，它本身被封装在 IP 数据报中，然后再封装成数据链路层中的帧，如图 15.8 所示。

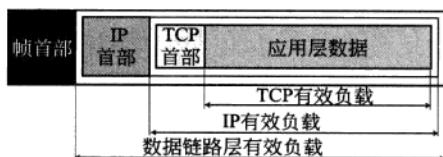


图 15.8 封装

15.4 TCP 连接

TCP 是面向连接的协议。正如我们在第 13 章中讨论的，面向连接的运输层协议在源点和终点之间建立了一条虚路径。同属于一个报文的所有报文段都沿着这条虚路径发送。为整个报文使用一条虚路径能够更容易地实施确认过程以及对损伤或丢失报文的重传。你可能会感到奇怪，既然 IP 是无连接协议，为什么使用 IP 服务的 TCP 却是面向连接的。关键在于 TCP 的连接是虚拟的，而不是物理的。TCP 工作在更高的层次上。TCP 使用 IP 的服务把一个个报文段交付给接收方，但是连接本身是由 TCP 控制的。如果一个报文段丢失或受到损伤，那么这个报文段就被重传。与 TCP 不同，IP 并不知道 TCP 的重传行为。如果一个报文段没有按序到达，那么 TCP 会保留它，直至丢失的报文段到达为止，但 IP 并不知道这个重新排序的过程。

在 TCP 中，面向连接的传输需要经过三个阶段：连接建立、数据传输和连接终止。

15.4.1 连接建立

TCP 以全双工方式传送数据。当两台主机的两个 TCP 建立连接后，它们应当能够同时向对方发送报文段。也就是说，在任何数据传送之前，双方都必须对通信进行初始化，并得到对方的认可。

三向握手

在 TCP 中使用的连接建立过程称为三向握手（three-way handshaking）。在我们的例子

中，一个称为客户的应用程序希望使用 TCP 作为运输层协议来和另一个称为服务器的应用程序建立连接。

这个过程从服务器开始。服务器程序告诉它的 TCP 自己已准备好接受连接。这个请求称为被动打开请求。虽然服务器的 TCP 已准备好接受来自世界上任何一个机器的连接，但是它自己并不能完成这个连接。

客户程序发出的请求称为主动打开。打算与某个开放的服务器进行连接的客户告诉它的 TCP，自己需要连接到某个特定的服务器上。TCP 现在可以开始进行如图 15.9 所示的三向握手过程。

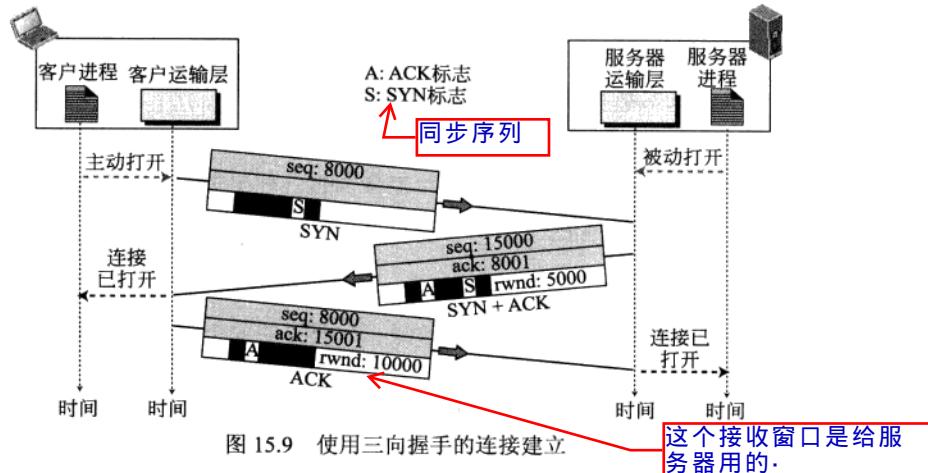


图 15.9 使用三向握手的连接建立

为了表示这个过程，我们使用了时间线。每个报文段的首部字段值都是完整的，并且可能还有一些可选字段也有相应的数值，不过，为了方便我们理解每个阶段，只画出了其中很少几个字段。图中显示了序号、确认号、控制标志（只有那些置 1 的）以及有关的窗口大小。这个阶段的三个步骤如下所示。

1. 客户发送第一个报文段（SYN 报文段），在这个报文段中只有 SYN 标志置为 1。这个报文段的作用是同步序号。在我们的例子中，客户选择了一个随机数作为第一个序号，并把这个序号发送给服务器。这个序号称为初始序号（ISN）。请注意，这个报文段中不包括确认号，也没有定义窗口大小。只有当一个报文段中包含了确认时，定义窗口大小才有意义。这个报文段还可以包含一些选项，我们将在本章的后面来讨论。请注意，SYN 报文段是一个控制报文段，它不携带任何数据。但是，它消耗了一个序号。当数据传送开始时，序号就应当加 1。我们可以说，SYN 报文段不包含真正的数据，但是我们可以想象它包含了一个虚字节。

SYN 报文段不携带任何数据，但是它要消耗一个序号。

2. 服务器发送第二个报文段，即 SYN + ACK 报文段，其中的两个标志（SYN 和 ACK）置为 1。这个报文段有两个目的。首先，它是另一个方向上通信的 SYN 报文段。服务器使用这个报文段来同步它的初始序号，以便从服务器向客户发送字节。其次，服务器还通过

ACK 标志来确认已收到来自客户端的 SYN 报文段，同时给出期望从客户端收到的下一个序号。因为这个报文段包含了确认，所以它还需要定义接收窗口大小，即 rwnd（由客户端使用），我们将在流量控制一节中讨论这个问题。

SYN+ACK 报文段不携带数据，但要消耗一个序号。

3. 客户发送第三个报文段。这仅仅是一个 ACK 报文段。它使用 ACK 标志和确认号字段来确认收到了第二个报文段。请注意，这个报文段的序号和 SYN 报文段使用的序号一样，也就是说，这个 ACK 报文段不消耗任何序号。客户还必须定义服务器的窗口大小。在某些实现中，连接阶段的第三个报文段可以携带客户的一个数据块。在这种情况下，第三个报文段必须有一个新的序号来表示数据中的第一个字节的编号。通常，第三个报文段不携带数据，因而不消耗序号。

ACK 报文段如果不携带数据就不消耗序号。

同时打开

当两个进程都发出主动打开请求时，可能会发生一种罕见的情况。此时，两个 TCP 都向对方发送 SYN+ACK 报文段，然后在它们之间建立一条连接。我们将在下一节讨论转换图时说明这种情况。

SYN 洪泛攻击

TCP 中使用的连接建立过程很容易碰到一个严重的问题，称为 **SYN 洪泛攻击** (**SYN flooding attack**)。当一个或多个恶意的攻击者向某台服务器发送大量的 SYN 报文段，并通过伪造报文段中的源 IP 地址来假装每一个报文段来自不同的客户时，这个问题就发生了。服务器认为这些客户发来了主动打开请求，于是就分配必要的资源，如创建传送控制块 (TCB) 表 (在本章的后面要讨论它)，并设置一些计时器。然后，TCP 的服务器向这些假冒的客户发送 SYN+ACK 报文段，而这些报文段都丢失了。但是，在服务器等待握手的第三步的这段时间里，大量的资源被占而没有利用。如果在很短的时间内，SYN 报文段的数量很大，服务器最终会因资源耗尽而不能接受来自合法客户的连接请求。这种 SYN 洪泛攻击属于一组称为 **拒绝服务攻击** (**denial of service attack**) 的安全攻击，即攻击者用大量的服务请求垄断了一个系统，使这个系统因超载而拒绝为合法的请求提供服务。

TCP 的某些实现采取了一些策略来减轻 SYN 攻击的影响。有些实现强制限定在指定时间内的连接请求次数。有些则把来自不希望的源地址的数据报过滤掉。最近的一种策略是通过使用 **Cookie**，做到推迟资源的分配，直至服务器能够证实连接请求来自合法的 IP 地址。我们在下一章要讨论的一个新的运输层协议，SCTP，就是采用了这种策略。

15.4.2 数据传送

在连接建立之后，双向的数据传送 (data transfer) 就可以开始。客户和服务器可以在两个方向上传送数据和确认。我们将在本章的稍后部分讨论确认的规则，目前只要知道数据可以双向传送，并且在同一个报文段中也可以携带确认就足够了。确认是随数据捎带过来的。图 15.10 给出了一个例子。

在这个例子中，当连接建立后，客户通过两个报文段发送了 2000 字节的数据。然后，服务器用一个报文段发送了 2000 字节的数据。前三个报文段既带有数据又带有确认，但最后一个报文段只有确认而没有数据，因为已经没有更多的数据要发送了。请注意序号和确认号的值。客户发送的数据报文段具有置 1 的 PSH (推送) 标志，因此服务器 TCP 知道要在收到这些数据后尽可能快地把它们交付给服务器进程。我们在后面还要详细讨论这个标志的使用。另一方面，从服务器发送来的报文段则没有把推送标志置 1。大多数 TCP 的实现把这个标志的置 1 还是不置 1 作为一个选项。

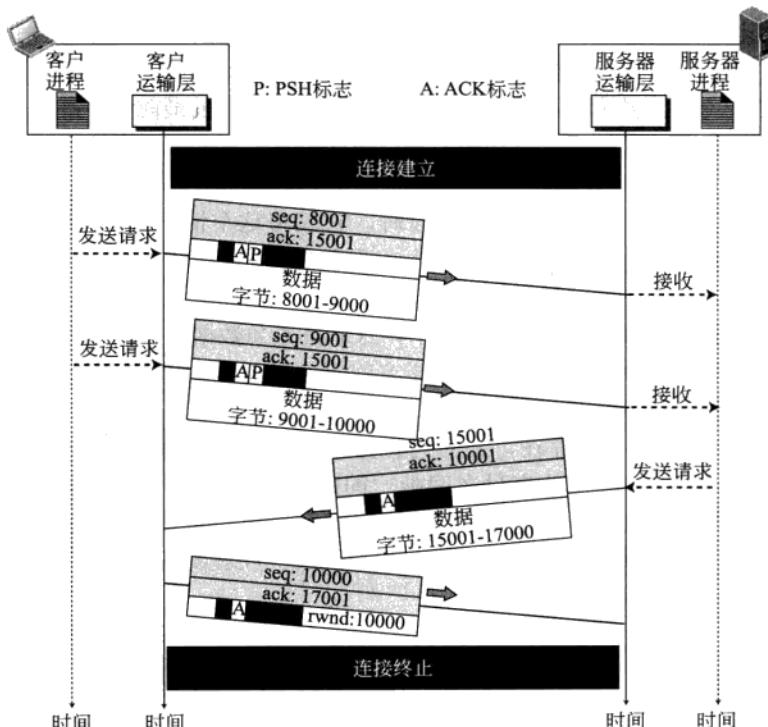


图 15.10 数据传送

推送数据

我们知道，发送 TCP 利用缓存来存储从发送应用程序传来的数据流。发送 TCP 可以选择报文段的长度。接收 TCP 在数据到达时也要将这些数据进行缓存，并在接收应用程序就绪时或者接收 TCP 认为方便时，再把这些数据交付给应用程序。这种灵活性提高了 TCP 的效率。

但是，在有些场合，应用程序不想要这种灵活性。例如，某个应用程序正在与对方的另一个应用程序进行交互式通信。其中一方的应用程序希望把它输入的字符发送给对方的应用程序，并希望立即收到响应。数据的延迟传输和延迟交付对此类应用程序来说是不可接受的。

TCP 可以处理这种情况。发送方的应用程序可以请求推送操作。这就是说，发送 TCP

不必等待窗口被填满，它必须马上创建一个报文段并立刻发送。发送 TCP 还必须置这个报文段的推送位（PSH）为 1，以便告诉接收 TCP，这个报文段所包括的数据必须尽快地交付给接收应用程序，不要等待更多数据的到来。

虽然推送操作可以由应用程序提出请求，但目前大多数的 TCP 实现都忽略了此类请求。TCP 可以选择使用或不使用这个操作。

紧急数据

TCP 是面向流的协议。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字节流中占有一个位置。但是，在某些情况下，应用程序需要发送紧急字节，也就是说有些字节需要被另一端的应用程序以特殊方式对待。解决这个问题的办法就是发送一个 URG（紧急）位置 1 的报文段。发送应用程序告诉发送 TCP 这块数据是紧急的。发送 TCP 创建一个报文段，并把紧急数据插入报文段的最前面。报文段的其余部分可以包含来自缓存的正常数据。首部中的紧急指针字段定义了紧急数据结束的地方（紧急数据的最后一个字节号）。

当接收 TCP 收到 URG 置 1 的报文段时，它就将这个情况通知给接收应用程序。如何做到这一点则取决于操作系统。在此之后会采取什么动作就完全由接收程序来决定了。

有一个很重要的地方还需要再强调一下，TCP 的紧急数据既不是优先服务，也不是加速数据服务。相反，TCP 紧急模式只是发送方的应用程序对某一部分字节流做了标记，要求接收方的应用程序特殊对待。

因此，指出紧急数据的存在，并对它在数据流中的位置进行标记，就是紧急数据的交付与其他正常 TCP 数据的交付之间唯一的区别。除此之外，紧急数据的处理与其他 TCP 字节流的处理完全一样。接收方的应用程序必须完全按数据被提交时的顺序来读取每一个字节，不论是否使用了紧急模式。标准的 TCP 实现永远不会失序地交付任何数据。

15.4.3 连接终止

参与数据交换的双方中的任何一方（客户或服务器）都可以关闭连接，虽然一般来说这是由客户启动的。目前，大多数 TCP 实现允许在连接终止时有两种选择：三向握手和具有半关闭选项的四向握手。

三向握手

今天的大多数实现都允许在连接终止时使用三向握手，如图 15.11 所示。

1. 在正常的情况下，客户 TCP 在收到客户进程发来的关闭命令后，就发送第一个报文段，这是一个把 FIN 位置 1 的 FIN 报文段。请注意，FIN 报文段可以包含客户发送的最后一块数据，或者也可以仅仅是一个如图所示的控制报文段。如果它只是一个控制报文段，那么它仅消耗一个序号。

如果 FIN 报文段不携带数据，它只消耗一个序号。

2. 服务器 TCP 在收到这个 FIN 报文段后，把这种情况通知它的进程，并发送第二个

报文段。第二个报文段是 FIN + ACK 报文段，以证实它收到了来自客户端的 FIN 报文段，同时也宣布另一个方向正在关闭连接。这个报文段还可以包含来自服务器的最后一块数据。如果它不携带数据，则只消耗一个序号。

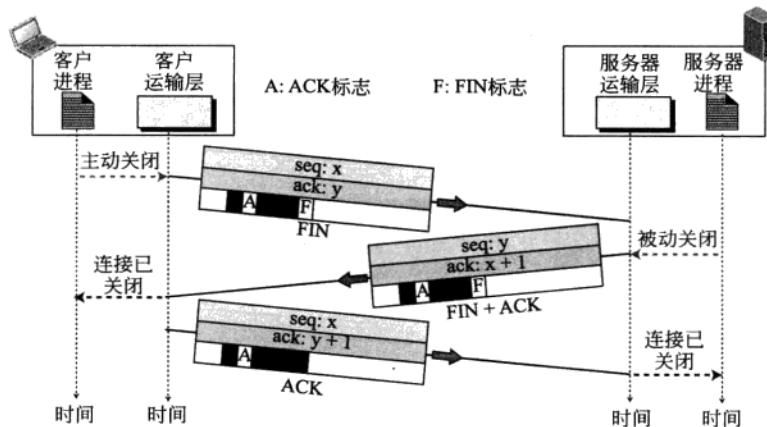


图 15.11 使用三向握手的连接终止

如果 FIN + ACK 报文段不携带数据，则它只消耗一个序号。

3. 客户 TCP 发送最后一个报文段，这是一个 ACK 报文段，以证实从 TCP 服务器收到了 FIN 报文段。这个报文段包括了一个确认号，它等于从服务器收到的 FIN 报文段的序号再加上 1。这个报文段不能携带数据，并且也不消耗序号。

半关闭

在 TCP 中，连接的一方可以停止发送数据，但仍然可以接收数据，这就称为半关闭 (half-close)。服务器或客户都可以发出半关闭请求。如果服务器在开始对数据进行处理之前需要所有数据都到齐，就会出现这种情况。排序就是一个典型的例子。当客户向服务器发送要求排序的数据时，服务器需要收到所有的数据之后才能开始排序。也就是说，客户在发送完所有的数据后，可以把从客户到服务器方向的连接关闭。但是，从服务器到客户方向的连接还必须是打开的，以便返回排序后的数据。服务器在接收完数据后还需要一些时间来进行排序处理，所以它的输出方向的连接必须是保持打开的。

图 15.12 是半关闭的一个例子。从客户到服务器的数据传送结束了。客户发送一个 FIN 报文段，此连接被半关闭了。服务器通过发送 ACK 报文段来表示接受这个半关闭。但是，服务器仍然可以发送数据。当服务器已经把所有处理过的数据都发送完毕后，就发送一个 FIN 报文段，并且被客户发来的 ACK 予以确认。

在连接被半关闭之后，数据可以从服务器传送到客户，并且确认可以从客户传送到服务器，但客户不能再给服务器发送任何数据了。请注意我们使用的序号。第二个报文段 (ACK) 不消耗序号。虽然客户已收到了序号 $y - 1$ ，并期望接收序号 y ，但服务器发送 ACK 报文段的序号仍然是 $y - 1$ 。在连接最后关闭时，最后一个 ACK 报文段的序号仍然是 x ，因为该方向的数据传送不消耗序号。

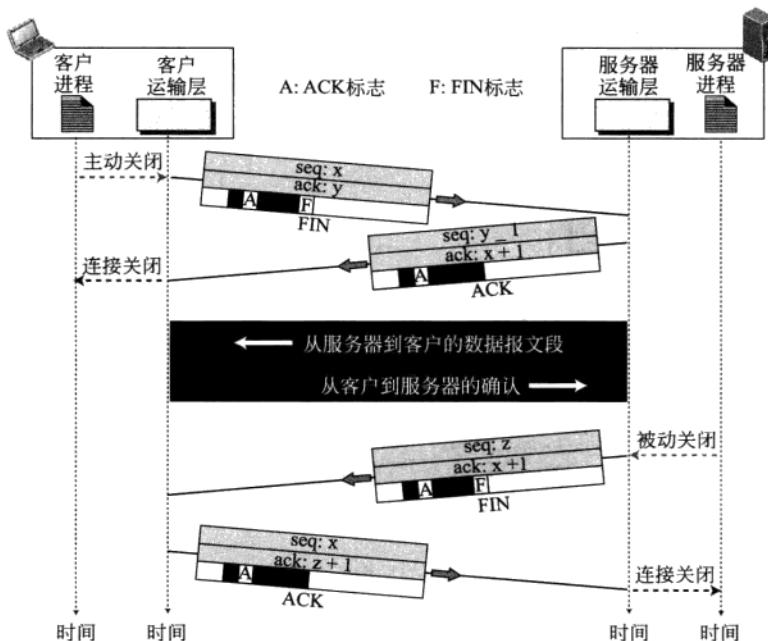


图 15.12 半关闭

15.4.4 连接复位

某一端的 TCP 可能会拒绝一个连接请求，也可能异常终止一条在用的连接，或者可能要终止一条空闲的连接。所有这些都是通过 RST（复位）标志来完成。

拒绝连接请求

假定某一端的 TCP 向一个并不存在的端口请求连接，另一端的 TCP 就可以发送 RST 位置 1 的报文段来拒绝这个请求。我们将在下节用一个例子来说明这种情况。

异常终止连接

由于出现了异常情况，某一端的 TCP 可能希望放弃一条在用的连接。它可以发送一个 RST 报文段来关闭这条连接。我们将在下节用一个例子来说明这种情况。

终止空闲的连接

某一端的 TCP 可能发现另一端的 TCP 已经空闲了很长的时间。它就可以发送 RST 报文段来终止这个连接。该过程与异常终止连接是一样的。

15.5 状态转换图

为了清楚地掌握在连接建立、连接终止以及数据传送时所发生的所有不同事件，TCP 以如图 15.13 所示的有限状态机的形式来定义。

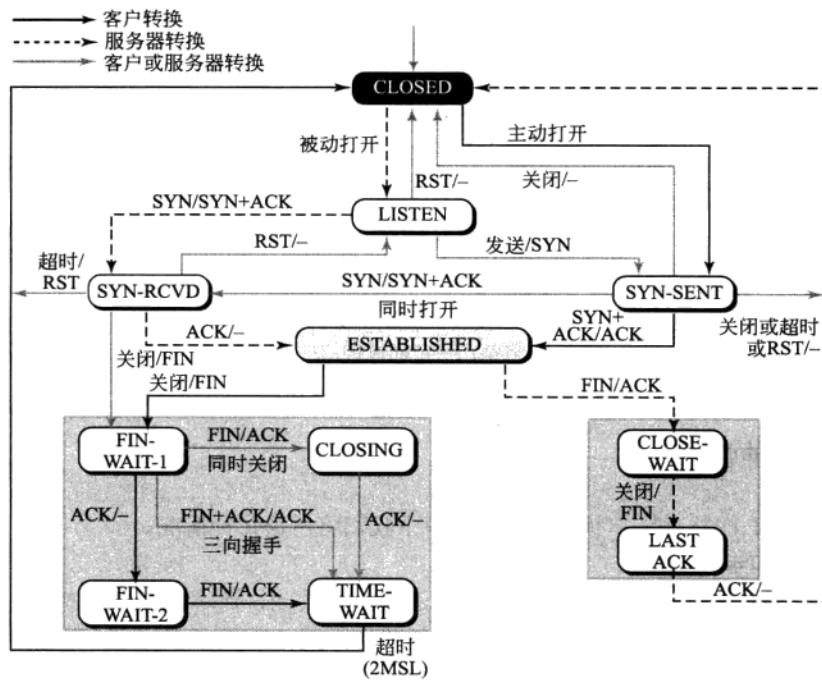


图 15.13 状态转换图

该图将 TCP 客户和服务器使用的两个 FSM 合并成了一张图。圆角框代表的是状态。从一个状态到另一个状态的转换用有向连线表示。每一条连线都注上用斜线隔开的两个字符串。第一个字符串是输入，即 TCP 接收到了什么。第二个字符串是输出，即 TCP 发送出去了什么。图中的黑色虚线代表的是通常由服务器经历的转换，而黑色实线代表的是通常由客户经历的转换。不过，在某些情况下，服务器的转换可以经过黑色实线，而客户的转换可以经过黑色虚线。带灰底的线条所示为特殊情况。请注意，标记为 ESTABLISHED 的圆角框实际上代表了两组状态，一组是客户的，另一组是服务器的，它们用于流量和差错控制（本章稍后将要讨论）。我们通过基于图 15.13 的几个情况来说明每种情况下的部分图示。

在状态图中，标记为 ESTABLISHED 的状态实际上是两组不同的状态，它们是客户和服务器进行数据传送时的状态。

表 15.2 给出了 TCP 状态的列表。

表 15.2 TCP 的各种状态

状态	说明
CLOSED	没有连接
LISTEN	收到了被动打开; 等待 SYN
SYN-SENT	已发送 SYN; 等待 ACK
SYN-RCV'D	已发送 SYN + ACK; 等待 ACK

续表

状态	说明
ESTABLISHED	连接建立; 数据传送在进行
FIN-WAIT-1	第一个 FIN 已发送; 等待 ACK
FIN-WAIT-2	对第一个 FIN 的 ACK 已收到; 等待第二个 FIN
CLOSE-WAIT	收到第一个 FIN, 已发送 ACK; 等待应用程序关闭
TIME-WAIT	收到第二个 FIN, 已发送 ACK; 等待 2MSL 超时
LAST-ACK	已发送第二个 FIN; 等待 ACK
CLOSING	双方都已决定同时关闭

15.5.1 几种情况

要了解 TCP 的状态机和状态图, 在本节中我们将讨论几种情况。

连接建立和半关闭终止

我们要描绘的是服务器进程发起被动打开和被动关闭, 而客户发起主动打开和主动关闭的情况。半关闭终止使我们能够展示更多的状态。图 15.14 所示为客户和服务器的两个状态转换图。

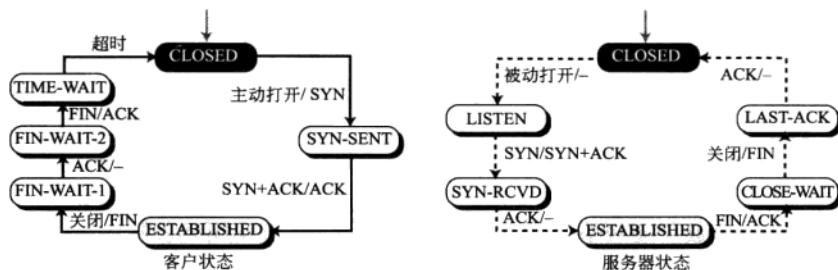


图 15.14 连接建立和半关闭终止的状态转换图

图 15.15 通过时间线图来展示同样的过程。

客户状态 客户进程向它的 TCP 发出一个命令, 以请求连接到某个特定的套接字地址, 这就称为主动打开。于是 TCP 发送一个 SYN 报文段, 并进入到 **SYN-SENT** 状态。在收到 SYN + ACK 报文段之后, TCP 发送 ACK 报文段, 并进入 **ESTABLISHED** 状态。此后数据开始传送和确认, 一般都是双向的。当客户进程没有更多的数据要传送时, 就发出称为主动关闭的命令。于是客户 TCP 发送 FIN 报文段, 并进入 **FIN-WAIT-1** 状态。当它收到对刚才发送的 FIN 报文段的 ACK 后, 就进入到 **FIN-WAIT-2** 状态, 并继续停留在这个状态, 直至收到一个来自服务器的 FIN 报文段为止。在收到这个 FIN 报文段后, 客户就发送 ACK 报文段, 并进入 **TIME-WAIT** 状态, 同时设置一个计时器, 它的超时时间是最大报文段寿命 (Maximum Segment Lifetime, MSL) 的两倍。MSL 是一个报文段被丢弃之前在因特网中能够生存的最大时间。我们可以回想一下, TCP 报文段是封装在生存时间 (TTL)

受限的 IP 数据报中。当 IP 数据报被丢弃时，封装在其中的 TCP 报文段也就丢失了。MSL 的常用数值是 30~60 秒。这里有两个理由使得我们需要有 TIME-WAIT 状态和 2MSL 计时器。

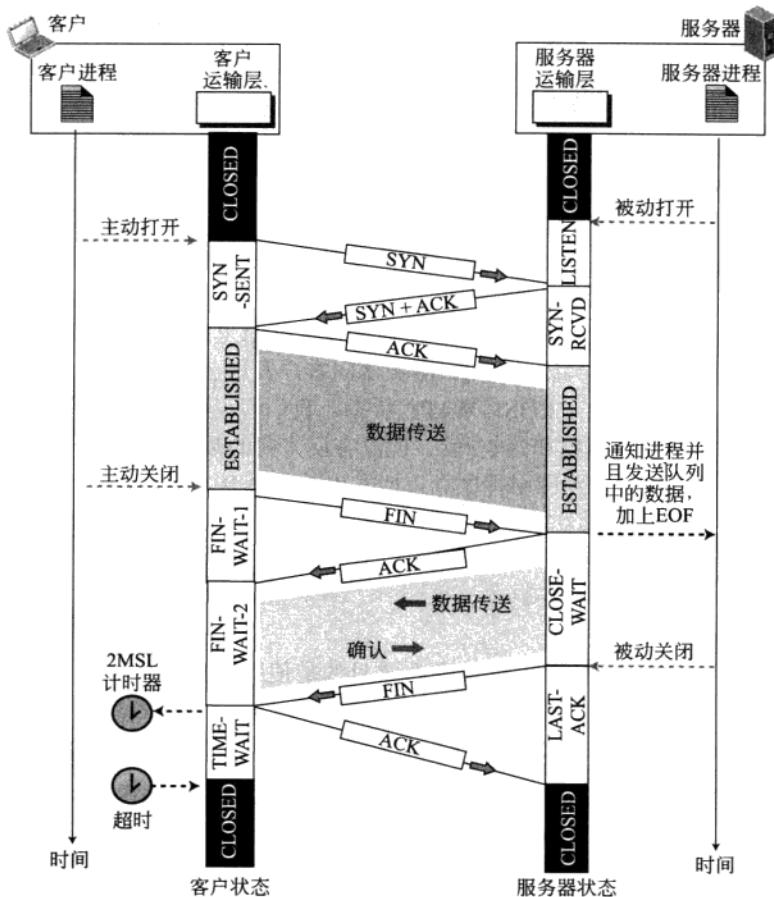


图 15.15 连接建立和半关闭终止的时间线图

1. 如果最后一个 ACK 报文段丢失了，那么服务器 TCP（它为最后的 FIN 设置了计时器）以为是它的 FIN 丢失了，因而重传它。如果客户已进入 CLOSED 状态，并在 2MSL 计时器超时之前就关闭了这条连接，那么客户就永远收不到这个重传的 FIN 报文段，因而服务器也就永远收不到最后的 ACK。服务器无法关闭这条连接。2MSL 计时器可以使客户等待足够长的时间，使得在 ACK 丢失（一个 MSL）的情况下，可以等到下一个 FIN 的到来（另一个 MSL）。如果在 TIME-WAIT 状态中有一个新的 FIN 到达了，客户就发送一个新的 ACK，并重新启动这个 2MSL 计时器。
设置这个的原因就是保证服务器也能正常被关掉!

2. 某个连接中的重复报文段可能会出现在下一个连接中。假定客户和服务器已经关闭了连接。经过短暂的时间后，它们又打开了一个新的连接，使用的是相同的套接字地址（同

样的源地址和目的地址，同样的源端口号和目的端口号）。这样的新连接称为旧连接的化身。如果这两个连接之间的时间间隔很短，那么前一个连接中的重复报文段有可能会到达新连接中，同时被解释为属于这个新连接的报文段。为了避免这个问题，TCP 规定这种化身必须经过 2MSL 时间之后才能出现。但是在某些实现中，如果这个化身的初始序号大于前一个连接使用的最后一个序号，就会忽略这个规则。

服务器状态 在我们的假设的情况下，服务器进程发出一个打开命令。这必须在客户发出打开命令之前发生。服务器 TCP 进入 **LISTEN** 状态，并继续被动地处在这个状态中，直至收到 SYN 报文段为止。当服务器 TCP 收到 SYN 报文段后，就发送 SYN + ACK 报文段，并进入 **SYN + RCVD** 状态，等待客户发送 ACK 报文段。在收到 ACK 报文段后，就进入到 **ESTABLISHED** 状态，此时可进行数据传送。

请注意，虽然双方（客户或服务器）都可以发起关闭过程，但是从大多数情况的角度考虑，我们假设由客户发起的关闭。

TCP 可以一直保持在这个状态，直至收到来自客户 TCP 的 FIN 报文段，这表示没有数据要发送了，因而连接可以关闭。此时，服务器向客户发送 ACK 报文段，并发送队列中尚未发送完的数据，然后进入 **CLOSE-WAIT** 状态。我们假设的情况是半关闭终止，服务器 TCP 可以继续向客户发送数据和接收确认，但没有反方向的数据能够传送过来。服务器 TCP 可以一直处在这个状态，直到应用程序真正地发出关闭命令。这时服务器 TCP 向客户发送 FIN 报文段表示它也要关闭这个连接了，并进入到 **LAST-ACK** 状态。服务器 TCP 一直处在这个状态，直至收到最后的 ACK 报文段，然后进入 **CLOSED** 状态。从第一个 FIN 报文段开始的终止阶段称为四向握手。

一种常见的情况

如前所述，在连接建立和终止阶段更为常见的是使用三向握手。图 15.16 所示为这种情况下客户和服务器的状态转换图。

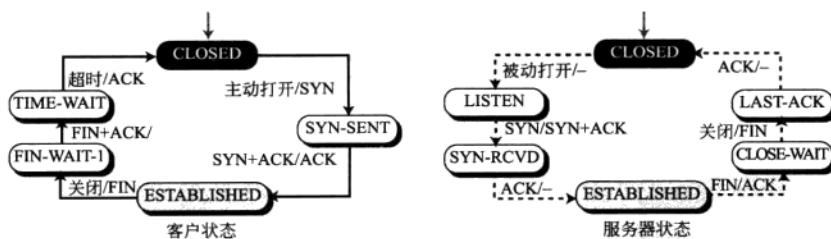


图 15.16 一种常见情况的转换图

图 15.17 用时间线描绘了相同的情况。其中的连接建立阶段与前一种情况中的相同，我们仅展示连接终止阶段。

如图所示，在数据传送阶段完成后，客户进程发出关闭命令。客户 TCP 发送 FIN 报文段，并进入 **FIN-WAIT-1** 状态。服务器 TCP 在收到这个 FIN 报文段后，继续向客户发送队列剩余的所有数据，并在最后附上 EOF 标记，表示这个连接要关闭了。服务器 TCP 进入 **CLOSE-WAIT** 状态，但它会推迟对收到的由客户发送来的 FIN 报文段的确认，直至它收到从自己的进程发过来的被动关闭命令。在收到被动关闭命令后，服务器 TCP 就

向客户发送 FIN + ACK 报文段，并进入 **LAST-ACK** 状态，等待最后的 ACK。从中可以看出，客户取消了 **FIN-WAIT-2** 状态而直接进入 **TIME-WAIT** 状态。剩下的部分和四向握手相同。

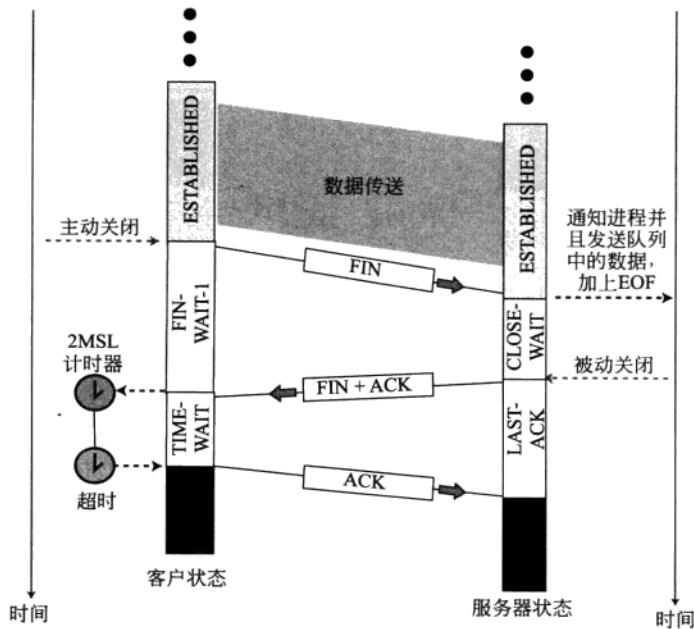


图 15.17 一种常见情况的时间线图

同时打开

同时打开 (simultaneous open) 就是双方的应用程序都发出主动打开命令。这是一种非常罕见的情况。此时没有客户，也没有服务器，通信的双方是对等的，彼此知道双方的本地端口号。TCP 允许出现这种情况，但它很少会出现，因为双方都需要向对方发送 SYN 报文段，这两个报文段要在同时传送。也就是说，两个应用程序必须几乎同时地发出主动打开命令。图 15.18 给出了这种情况下连接建立阶段。双方的 TCP 要经过 **SYN-SENT** 和 **SYN-RCVD** 状态之后才能进入 **ESTABLISHED** 状态。更仔细地观察可看出这两个进程在同一时刻既充当了客户，又充当了服务器。两个 SYN + ACK 报文段确认了两个 SYN 报文段后，连接就打开了。请注意，这个连接建立包括四向握手。数据传送阶段和连接终止阶段和前面的例子一样，这里不再画出。我们将这种情况的状态转换图留作练习。

同时关闭

另外一种并不常见，但也有可能出现的情况就是如图 15.19 所示的同时关闭 (simultaneous close)。

在这种情况下，两端都发出主动关闭。双方的 TCP 都进入 **FIN-WAIT-1** 状态，并发送 FIN 报文段，这两个报文段同时在传送。在收到 FIN 报文段后，每一端都进入 **CLOSING** 状态，并发送 ACK 报文段。**CLOSING** 状态取代了常见情景中的 **FIN-WAIT-2** 和 **CLOSE-WAIT**。在收到 ACK 后，双方都进入 **TIME-WAIT** 状态。请注意，这个时间段对

双方来说都是需要的，因为双方都发送了 ACK，而它们都有可能丢失。在图中我们省略了连接建立和数据传送阶段。我们将这种情景的状态转换图留作习题。

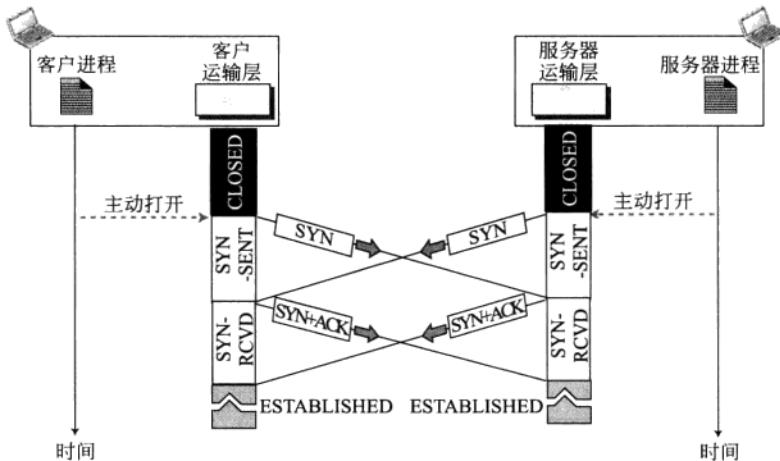


图 15.18 同时打开

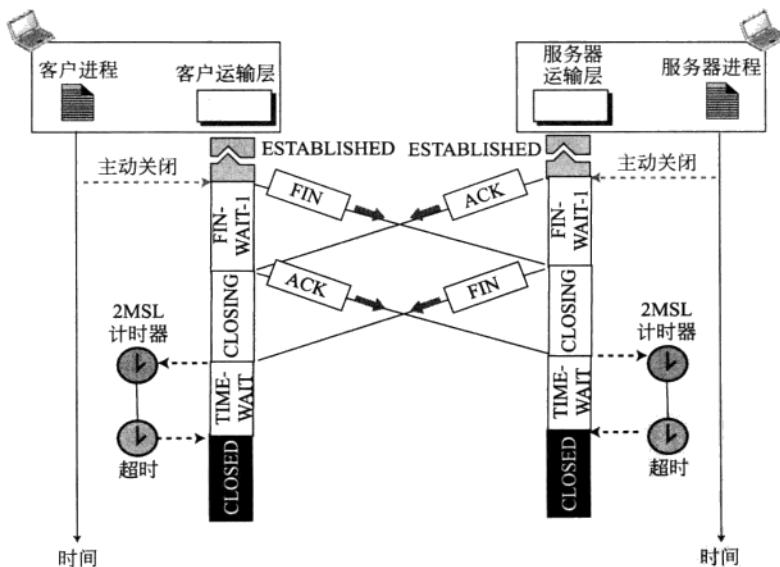


图 15.19 同时关闭

拒绝连接

另一种常见的原因是发生在服务器 TCP 拒绝连接时，可能因为 SYN 报文段中目的端口所定义的服务器当时并没有处于 LISTEN 状态。服务器 TCP 在收到这样的 SYN 报文段后会发送一个 RST + ACK 报文段，它确认了 SYN 报文段，但与此同时重置（拒绝）这条连接。服务器 TCP 进入 LISTEN 状态等待另一个连接。客户在收到这个 RST + ACK 报文

段后进入 **CLOSED** 状态。图 15.20 给出了这种情况。

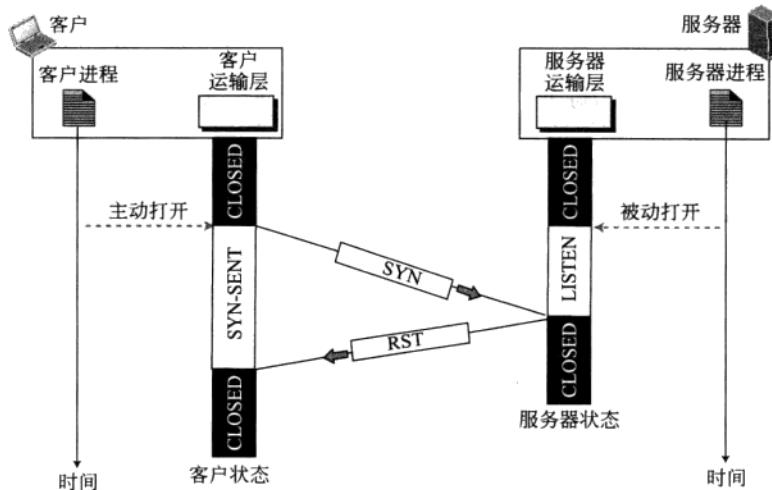


图 15.20 拒绝连接

异常终止连接

图 15.21 所示为客户进程发出异常终止的情景。请注意，此功能属性并没有在图 15.13 所示的通用转换图中展示出来，因为运营商可以选择实现它或是不实现它。

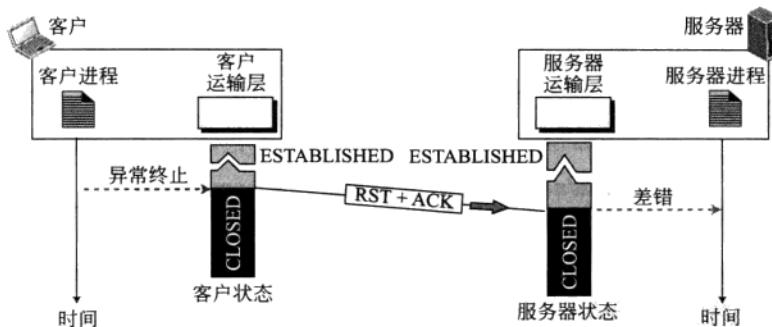


图 15.21 异常终止连接

除了关闭连接，进程还可以异常终止连接。出现这种情况是因为进程出了故障（可能是死锁在无限循环中），或者是不想发送队列中的数据了（由于数据中存在某些不一致）。另外，TCP 也有可能想要异常终止一条连接。如果 TCP 收到了属于上一个连接的报文段（化身）就有可能发生这种情况。在所有这些情况下，TCP 可以通过发送 RST 报文段使连接异常终止。图中的客户 TCP 发送 RST + ACK 报文段，并丢弃队列中的所有数据。服务器 TCP 也把队列中的所有数据都丢弃，并通过差错报文来通知服务器进程。双方的 TCP 都立即进入 **CLOSED** 状态。请注意，对 RST 报文段不用响应 ACK 报文段。

15.6 TCP 中的窗口

在讨论 TCP 中的数据传送以及诸如流量、差错和拥塞控制之前，我们先要描述一下在 TCP 中使用的窗口。TCP 为每个方向的数据传送各使用两个窗口（发送窗口和接收窗口），也就是说，对于双向通信总共有四个窗口。但是，为了简化讨论，我们要做一个在现实中很少见的假设，就是认为通信仅为单向的（以从客户到服务器为例）。双向通信可以被看成是具有捎带功能的两个单向通信的结合。

15.6.1 发送窗口

图 15.22 所示为发送窗口的一个例子。我们使用的这个窗口大小为 100 字节（通常应该有几千个字节），但是稍后我们会看到，发送窗口大小是由接收方（流量控制）以及底层网络的拥塞程度（拥塞控制）来决定的。图中描绘了一个发送窗口是如何打开、关闭和收缩的。

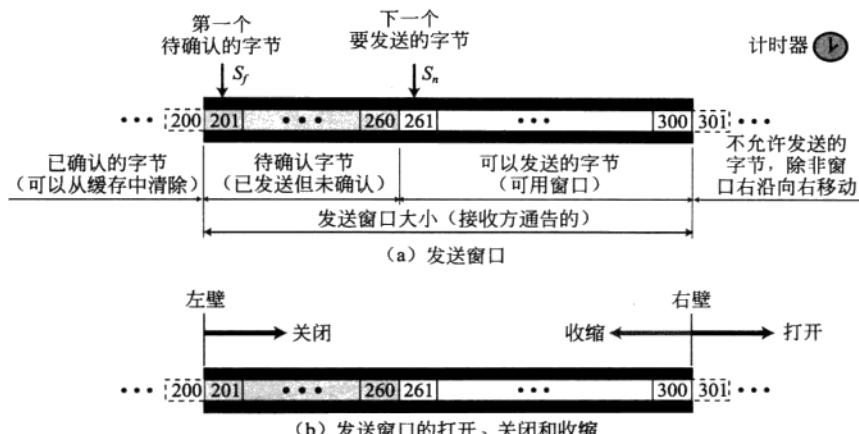


图 15.22 TCP 中的发送窗口

TCP 的发送窗口类似于选择重传协议的发送窗口（参见第 13 章），但是有几点区别：

1. 第一个区别是窗口所关联的实体的本质不同。SR 的窗口为分组编号，而 TCP 中的窗口要为字节编号。虽然 TCP 的传输实际上也是一个报文段一个报文段的，但用于控制窗口的变量以字节为单位。
2. 第二个区别是在某些实现中 TCP 可以保存从进程那里接收到的数据，并稍后再发出去。不过，我们假设发送 TCP 能够从它的进程那里一接收到数据就发送该数据的报文段。
3. 另一个区别是计时器的数量。理论上，选择重传协议可以为每个发送的分组使用数个计时器，但是 TCP 协议只使用一个计时器。我们在后面会解释差错控制中该计时器的使用。

15.6.2 接收窗口

图 15.23 所示为接收窗口的一个例子。我们使用的这个窗口大小为 100 字节（通常应该有几千个字节）。图中同样也描绘了该窗口是如何打开和关闭的，在实际中，接收窗口永远不会收缩。

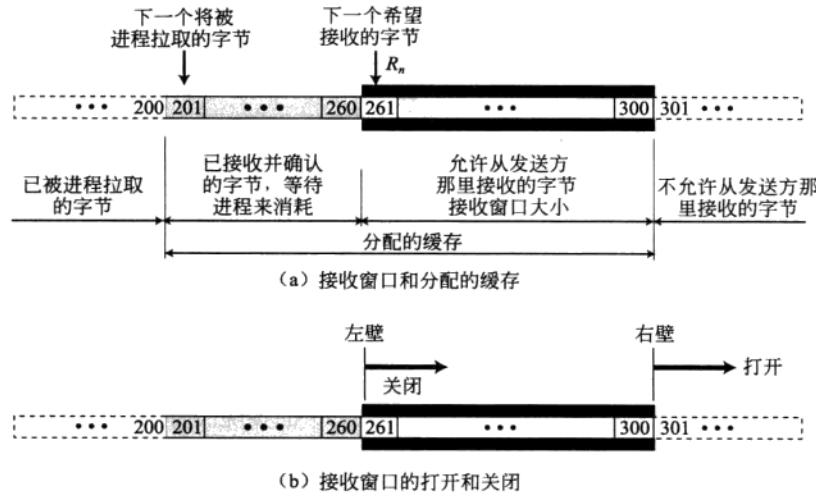


图 15.23 TCP 中的接收窗口

TCP 中使用的接收窗口与第 13 章讨论的 SR 中的接收窗口有两点区别：

- 第一个区别是 TCP 允许接收进程按照自己的节奏来拉取数据。**也就是说，分配给接收方的缓存中可能有一部分被已经接收和确认但正在等待接收进程来拉取的数据所占用。因此，接收窗口大小总是小于或等于缓存大小，如上图所示。接收窗口大小决定了接收窗口在溢出之前能够从发送方那里接收的字节数。换言之，通常被称为 *rwnd* 的接收窗口大小可以用以下公式计算：

$$\text{rwnd} = \text{缓存大小} - \text{正在等待被拉取的字节数}$$

- 第二个区别是 TCP 协议中使用的确认的方式不同。**回想在 SR 中，确认是选择性的，它指向已被接收的未损坏的分组，而 TCP 的主流确认机制是累积确认，它宣布的是下一个希望接收的字节（从这方面来看，TCP 有点像第 13 章讨论的 GBN）。不过，在 TCP 的新版本中，既使用了累积确认，也使用了选择确认，我们将在稍后介绍可选项的小节中讨论。

15.7 流量控制

正如在第 13 章所讨论的，流量控制平衡了生产者产生数据的速度和消费者消耗数据的速度。TCP 把流量控制从差错控制中独立出来。在本小节，我们将讨论流量控制，暂且不管差错控制。我们暂且假设在发送 TCP 和接收 TCP 之间的逻辑信道是不出错的。

图 15.24 所示为发送方和接收方之间的单向数据传送。双向数据传送过程可以从单向传送中推断出来，就像在第 13 章所讨论的那样。

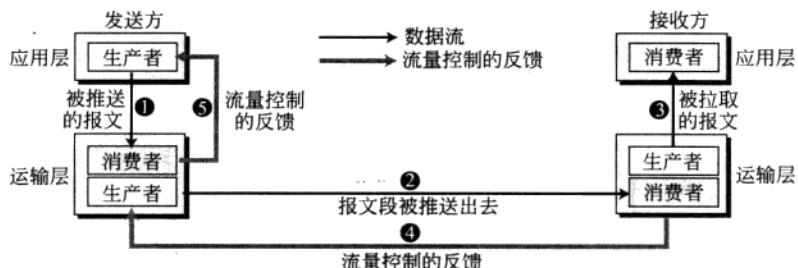


图 15.24 TCP 中的数据流和流量控制反馈

图中所描绘的数据走向是从发送进程向下到达发送 TCP，再从发送 TCP 到接收 TCP，然后从接收 TCP 向上到达接收进程（路径 1、2 和 3）。不过，流量控制的反馈走向是从接收 TCP 到发送 TCP，以及从发送 TCP 向上到发送进程的（路径 4 和 5）。TCP 的绝大多数实现都不提供从接收进程到接收 TCP 的流量控制反馈，而是让接收进程在准备就绪时去接收 TCP 那里拉取数据。换言之，接收 TCP 控制了发送 TCP，而发送 TCP 控制了发送进程。

从发送 TCP 到发送进程的流量控制反馈（路径 5）的实现很简单，一旦发送 TCP 的窗口满了，就拒绝接收数据。也就是说，我们对流量控制的讨论主要集中在从接收 TCP 到发送 TCP 的反馈（路径 4）。

15.7.1 打开和关闭窗口

为了实现流量控制，TCP 强制发送方和接收方不断调整它们的窗口大小，即使双方的缓存大小在连接建立时就被固定了下来。当有更多的字节从发送方传来时，接收窗口关闭（其左壁向右移动），而当更多的字节被进程拉取走时，接收窗口打开（其右壁向右移动）。我们假设接收窗口是不会收缩的（其右壁不会向左移动）。

发送窗口的打开、关闭和收缩是被接收方所控制的。当一个新的确认到达，且在这个确认允许时，发送窗口会关闭（其左壁向右移动）。当由接收方通告的接收窗口大小（rwnd）允许时，发送窗口会打开（其右壁向右移动）。发送窗口偶尔根据情况会收缩。我们假设这种情况不会发生。

一种假设的情况

我们要描绘的是在连接建立阶段如何设置发送窗口和接收窗口，并且在数据传送时，它们的状态又会发生怎样的改变。图 15.25 描绘了一个单向数据传送（从客户到服务器）的简单例子。就目前而言，让我们先忽略差错控制，假设没有报文段损坏、丢失、重复或失序到达。请注意，因为是单向数据传送，我们在这里只显示了两个窗口。

在客户和服务器之间一共交换了 8 个报文段。

1. 第一个报文段从客户到服务器（SYN 报文段）以请求连接。客户在这个报文段中宣布了它的初始序号为 100。当这个报文段到达服务器时，服务器分配一个大小为 800 字

节（假设）的缓存，并设置它的窗口覆盖整个缓存（ $rwnd = 800$ ）。请注意，下一个字节的编号为 101。

2. 第二个报文段是从服务器到客户。这是一个 ACK + SYN 报文段。这个报文段使用了 $ackNo = 101$ 来表示它希望接收编号从 101 开始的字节。同时它也宣布了客户可以将窗口大小设置为 800 字节。

3. 第三个报文段是从客户到服务器的 ACK 报文段。

4. 当客户根据服务器的指示设置自己的窗口大小（800）之后，进程又推送了 200 字节的数据。TCP 客户用 101~300 给这些字节编号，然后创建一个报文段并将其发送至服务器。这个报文段显示的起始字节编号为 101，并且携带了 200 字节的数据。然后通过调整客户的窗口来表示有 200 字节的数据已发送，且正在等待确认。当这个报文段到达服务器后，这些字节被保存下来，接收窗口关闭以表示下一个希望接收的是字节 301。被保存的字节占据了缓存中的 200 个字节。

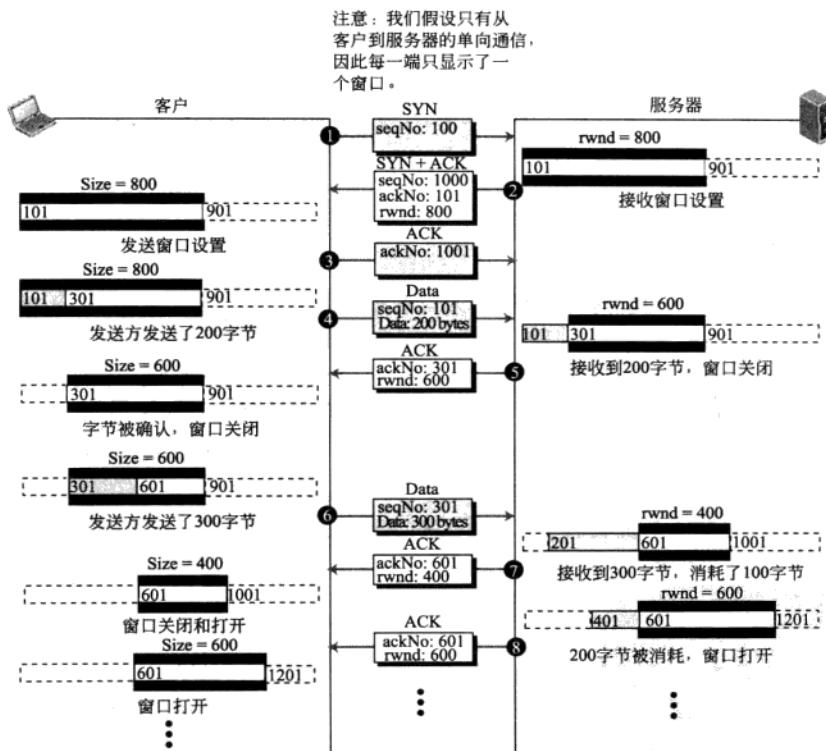


图 15.25 流量控制的一个例子

5. 第五个报文段是从服务器到客户的反馈。服务器确认了编号 300 及其之前的所有字节（希望接收字节 301）。这个报文段同时还携带了一个减小之后的接收窗口大小（600）。客户在收到这个报文段后，将已被确认的数据从它的窗口中清除出去，并关闭窗口，以表示下一个要发送的是字节 301。但是这个窗口大小要减小至 600。虽然分配的缓存能够放得

下 800 个字节，但是窗口不能打开（将其右壁向右移动），因为接收方没有允许它这样做。

6. 当客户进程又推送来另外 300 个字节后，客户 TCP 发送报文段 6。这个报文段的序号是 301，并且包含了 300 个字节。当这个报文段到达服务器后，服务器将它们保存起来，但是服务器必须减小它的窗口大小。在服务器进程拉取了 100 个字节的数据之后，窗口从左边关闭 300 个字节的量，同时又向右边打开了 100 个字节的量。结果，窗口大小只减小了 200 字节。现在，接收窗口大小是 400 字节。

7. 在第七个报文段中，服务器对收到的字节进行确认，并宣布自己的窗口大小是 400。当这个报文段到达客户后，客户没有其他选择，只能再一次减小它的窗口。客户 TCP 将窗口大小设置为服务器通告的 $rwnd = 400$ 。它的发送窗口从左边关闭 300 个字节，并在右边打开 100 个字节。

8. 报文段 8 是服务器进程又一次拉取了 200 字节之后从服务器发出的。服务器的窗口大小增加了，现在，新的 $rwnd$ 的值是 600。这个报文段告诉客户说，服务器仍然在期盼着字节 601，但是服务器的窗口大小已经扩展到 600 了。有一点我们需要说明，这个报文段的发送与否取决于具体实现的策略。有些实现可能不允许在这种情况下通告 $rwnd$ 的值，服务器必须在接收到一些数据之后才被允许做这件事。当这个报文段到达客户后，客户将发送窗口再打开 200 字节，并且没有做任何关闭动作。结果就是客户的窗口大小增加到 600 字节。

15.7.2 窗口的收缩

正如我们前面所提到的，接收窗口不允许收缩。但是，如果接收方指定的 $rwnd$ 值会导致发送窗口的缩小，那么发送窗口就收缩。某些实现是不允许发送窗口收缩的。它限制了发送窗口的右壁不允许向左移动。换言之，为了防止发送窗口的收缩，接收方在最近的和新的确认以及最近的和新的 $rwnd$ 值之间始终要满足下述关系：

$$\text{新的 ackNo} + \text{新的 rwnd} \geq \text{最近的 ackNo} + \text{最近的 rwnd}$$

这个不等式的左侧代表了窗口右壁的新位置，右侧则代表了窗口右壁原来的位置（相对于序号空间）。这个关系说明右壁永远不会向左移动。接收方必须按照这个不等式来检查它的通告是否合法。不过，请注意这个不等式仅当 $S_f < S_n$ 时才有效，我们应当谨记所有的计算都是模 2^{32} 的。

例 15.2

图 15.26 说明了为什么有些实现强制要求遵守以上不等式的原因。图中的 (a) 部分显示了最近收到的一个确认及其 $rwnd$ 的值。图中的 (b) 部分则显示了发送方发送了字节 206~214 之后的状态。字节 206~209 已经被确认并被清除出缓存。但是，新的通告指定的 $rwnd$ 值是 4，这样的话 $210+4 < 206+12$ 。当发送窗口收缩时就产生了一个问题：已经发送出去的字节 214 落在了窗口之外。前面所讨论的那个关系式将迫使接收方让窗口的右壁维持在 (a) 部分中所显示的位置上，因为接收方并不知道从字节 210 到 217 中有哪些已经发送出来了。要防止这种状况发生的一个办法是让接收方推迟发送反馈，直至它的窗口有足够的缓存空间可用。换言之，接收方应当等待，直至它的进程消耗了更多的字节，以满足上面给出的不等式。

窗口关闭

我们已讲过，使发送窗口的右壁向左移动以收缩发送窗口是非常不希望出现的。但是，有一个例外：接收端可以用发送 rwnd 为 0 的报文段来暂时关闭窗口。这种情况的发生是由于接收方因某种原因在一段时间内不愿意从发送方接收任何数据。此时，发送方实际上并非真正地把窗口大小收缩了，而只是暂停发送数据，直至收到一个新的通告为止。我们在后面将会看到，即使是在接收方的命令下关闭了窗口，发送方仍然可以发送具有一个字节数据的报文段，称为探测，用于防止死锁（见 TCP 的计时器的讨论）。



图 15.26 例 15.2

15.7.3 糊涂窗口综合征

在滑动窗口的操作中可能出现一个严重的问题，这就是发送应用程序产生数据的速度很慢，或者接收应用程序消耗数据的速度很慢，或者两者都有。不管是上述情况中的哪一种，都会使得发送数据的报文段很小，这就会降低运行的效率。例如，假若 TCP 发送的报文段只包含 1 个字节的数据，那么意味着我们为传送 1 字节的数据而发送了 41 字节的数据报（20 字节的 TCP 首部和 20 字节的 IP 首部）。此时的额外开销达到了 41/1，这就表示我们使用网络容量的效率非常低。再算上数据链路层和物理层的额外开销后，这种低效率的程度就更加严重了。这个问题称为糊涂窗口综合征 (silly window syndrome)。我们先来分别描述一下连接两端的问题各是怎样产生的，然后再给出解决问题的建议。

发送方产生的征状

如果发送 TCP 正在一个产生数据速度很慢的应用程序服务，例如一次产生 1 个字节，那么就有可能产生糊涂窗口综合征。这个应用程序一次把 1 个字节的数据写入发送 TCP 的缓存。如果发送 TCP 没有任何特殊的指令，它就会产生只包括 1 个字节数据的报文段。其结果就是有很多个 41 字节的报文段在互联网中传来传去。

解决这个问题的方法是防止发送 TCP 一个字节一个字节地发送数据。必须要强迫发送 TCP 等待，并把数据收集成较大的数据块后再发送。发送 TCP 需要等待多长时间呢？如果

它等待得过长，就会延迟了整个处理过程。如果它等待的时间不够长，最后很可能还是发送一个个小报文段。Nagle 找到了一个很精巧的解决方法。

Nagle 算法 Nagle 算法非常简单：

1. 发送 TCP 把它从发送应用程序收到的第一块数据发送出去，哪怕只有 1 个字节。
2. 在发送了第一个报文段后，发送 TCP 就在输出缓存中累积数据并等待，直至收到接收 TCP 发来的确认，或者已积累了足够的数据可以装成最大长度的报文段。这时，发送 TCP 就可以发送这个报文段了。
3. 对剩下的传输，不断重复步骤 2。如果收到了对报文段 2 的确认，或者已积累到足够多的数据可以装成最大长度的报文段，报文段 3 就必须发送出去。

Nagle 算法的巧妙之处在于它的简单，并且实际上它权衡了应用程序产生数据的速率和网络运输数据的速率。若应用程序比网络更快，则报文段就较大（最大长度报文段）。若应用程序比网络慢，则报文段就较小（小于最大长度报文段）。

接收方产生的征状

如果接收 TCP 为一个消耗数据很慢的应用程序服务，例如，一次消耗 1 个字节，它就有可能产生糊涂窗口综合征。假定发送应用程序产生 1 K 字节块的数据，但接收应用程序每次只消耗 1 字节的数据。再假定接收 TCP 的输入缓存为 4 K 字节。发送方先发送了 4 K 字节的数据。接收方将其存储在缓存中。现在缓存满了。接收方通告的窗口值为零，这表示发送方必须停止发送数据。接收应用程序从接收 TCP 的输入缓存中读取第一个字节的数据。现在输入缓存中有了 1 字节的空间。接收 TCP 宣布其窗口值为 1 字节，于是正在渴望发送数据的发送 TCP 会把这个宣布当作好消息，并发送一个只包含了 1 字节数据的报文段。这样的过程将持续下去。1 个字节的数据被消耗掉，然后发送携带 1 个字节数据的报文段。我们又一次遇到了效率问题和糊涂窗口综合征。

对于这种糊涂窗口综合征，即应用程序消耗数据的速度比数据到达的速度慢，有两种建议的解决方法。

Clark 解决方法 Clark 解决方法是只要有数据到达就发送确认，但在缓存中有足够大的空间放入最大长度的报文段之前，或者至少有一半的缓存空间为空之前，一直都宣布窗口大小为零。

推迟确认 第二种解决方法是推迟发送确认。这就表示当报文段到达时并不立即发送确认。接收方在对收到的报文段进行确认之前一直等待，直至输入缓存有足够的空间为止。推迟发送确认防止了发送 TCP 滑动它的窗口。发送 TCP 在发送了数据之后就停下来，因而防止了这种征状的出现。

推迟确认还有另外一个优点：它减少了通信量。接收方不需要对每一个报文段进行确认。但它也有一个缺点，就是确认的推迟有可能迫使发送方重传未被确认的报文段。

TCP 对这个优点和缺点进行了平衡。目前它的定义是推迟确认不能超过 500 ms。

15.8 差错控制

TCP 是可靠的运输层协议。这就表示应用程序把数据流交付给 TCP 后，要依靠 TCP

把整个数据流交付给另一端的应用程序，并且是按序的、没有差错、也没有任何一部分丢失或重复。

TCP 使用差错控制来提供可靠性。差错控制包括以下的一些机制：检测和重传受到损伤的报文段、重传丢失的报文段、保存失序到达的报文段直至缺失的报文段到齐，以及检测和丢弃重复的报文段。TCP 通过使用三个简单的工具来完成其差错控制：检验和、确认以及超时。

15.8.1 检验和

每个报文段都包括了一个检验和字段，用来检查报文段是否受到损伤。如果某个报文段因检验和无效而被检查出受到损伤，就由终点 TCP 将其丢弃，并被认为是丢失了。TCP 规定每个报文段都必须使用 16 位的检验和。在前面我们讨论过如何计算检验和。

15.8.2 确认

TCP 采用确认来证实收到了报文段。控制报文段不携带数据，但要消耗一个序号，它也需要被确认，而 ACK 报文段永远不需要被确认。

ACK 报文段不消耗序号，也不需要被确认。

确认类型

在以前，TCP 只使用一种类型的确认：累积确认。目前有一些 TCP 实现还采用了选择确认。

累积确认 (ACK) TCP 最初的设计是对收到的报文段进行累积确认。接收方通告它期望接收的下一个字节的序号，并忽略所有失序到达并被保存的报文段。有时这被称为肯定累积确认或 ACK。“肯定”这个词表示对于那些丢弃的、丢失的或重复的报文段都不提供反馈。在 TCP 首部中的 32 位 ACK 字段用于累积确认，而它的值仅在 ACK 标志为 1 时才有效。

选择确认 (SACK) 现在越来越多的实现又增加了另一种类型的确认，称为选择确认 (selective acknowledgment) 或 SACK。SACK 并没有取代 ACK，而是向发送方报告了更多的信息。SACK 要报告失序的数据块以及重复的报文段块（即收到不只一次）。但是，在 TCP 首部中并没有提供增加这类信息的地方，因此 SACK 是作为 TCP 首部末尾的选项来实现的。在我们讨论 TCP 的选项时将讨论这个新的特性。

产生确认

接收方在什么时候产生确认？在 TCP 的发展历程曾经定义过好几种规则，并用在多个实现中。这里我们给出几个最常用的规则。这些规则的顺序并不代表其重要性。

1. 当 A 端向 B 端发送数据报文段时，必须包含（捎带）一个确认，它给出 A 端期望接收的下一个序号。这条规则减少了所需的报文段的数量，因而也减少了通信量。

2. 当接收方没有数据要发送，但是收到了按序到达（序号是所期望的）的报文段，同时前一个报文段也已经确认过了，那么接收方就推迟发送确认报文段，直到另一个报文段

到达，或经过了一段时间（通常是 500 ms）。换言之，如果仅仅有一个按序到达的报文段还没有被确认，接收方需要推迟发送确认报文段。这条规则也减少了 ACK 报文段的通信量。

3. 当具有所期望的序号的报文段到达，而且前一个按序到达的报文段还没有被确认，那么接收方就要立即发送 ACK 报文段。换言之，在任何时候，不能有两个以上按序到达的报文段未被确认。这就避免了不必要的重传，而不必要的重传可能会引起网络的拥塞。

4. 当序号比期望的序号还大的失序报文段到达时，接收方立即发送 ACK 报文段，并宣布下一个期望的报文段序号。这将导致对丢失报文段的快重传（稍后讨论）。

5. 当一个丢失的报文段到达时，接收方要发送 ACK 报文段，并宣布下一个所期望的序号。这是告诉接收方，被通报为丢失的报文段已经收到了。

6. 如果到达一个重复的报文段，接收方丢弃该报文段，但是应当立即发送确认，指出下一个期望的报文段。这就解决了 ACK 报文段本身丢失所带来的一些问题。

15.8.3 重传

差错控制机制的核心就是报文段的重传。在一个报文段发送时，它会被保存到一个队列中，直至被确认为止。当重传计时器超时，或者发送方收到该队列中第一个报文段的三个重复的 ACK 时，该报文段被重传。

RTO 之后的重传

发送方 TCP 为每一条连接设置一个重传超时（retransmission time-out, RTO）计时器。当计时器时间到（即超时），TCP 发送队列中最前面的报文段（即序号最小的报文段），并重启计时器。请注意，此处再次强调我们假设 $S_f < S_n$ 。这个版本的 TCP 有时被称为 Tahoe。我们以后将会看到，在 TCP 中 RTO 的值是动态的，它根据报文段的往返时间（RTT）更新。RTT 是一个报文段到达终点并收到它的确认所需的时间。

三个重复的 ACK 报文段之后的重传

如果 RTO 值不是很大，前面描述的重传报文段的规则就足够了。但是，如果允许发送方更快地重传，而不用等待计时器超时，就能更有利于网络吞吐量，因此现在的大多数实现都遵守收到三个重复 ACK 则立即重传丢失的报文段的规则。这一特性称为快重传（fast retransmission），而使用了这个特性的 TCP 版本称为 Reno。在这个版本中，如果针对某个报文段有三个重复的确认（即原始的 ACK 再加上三个完全一样的复本）到达，那么下一个报文段将被立即重传，而不用等待计时器超时。

15.8.4 失序的报文段

现在的 TCP 实现都不会丢弃失序到达的报文段，而是把这些报文段暂时保存下来，并把它们标志为失序报文段，直至缺失的报文段到齐。但是请注意，TCP 从来不会把失序的报文段交付给进程。TCP 保证数据必须按序交付到进程。

数据可能不按序到达，接收方 TCP 把它们暂时保存下来，但是 TCP 保证没有失序的报文段会被交付给进程。

15.8.5 TCP 数据传送的 FSM

TCP 的数据传送接近于选择重传协议(在第 13 章中讨论过), 同时又有那么一点像 GBN 的地方。因为 TCP 接受失序到达的报文段, 因此它的行为可以被认为像 SR 协议一样, 但是因为最初它的确认机制是累积的, 这又使得它看起来像 GBN。不过, 如果 TCP 的实现中使用了 SACK, 那么它就更接近于 SR。

最适合 TCP 的模型是选择重传协议。

发送端 FSM

让我们用一个简化的 FSM 来描述 TCP 协议的发送端。这个 FSM 类似于我们为 SR 协议所讨论的, 只是有一些专门针对 TCP 的修改。我们假设通信是单向的, 并且报文段使用 ACK 来确认。就目前而言, 我们还忽略了选择确认和拥塞控制的问题。图 15.27 描绘了 TCP 发送端的简化 FSM。请注意, 这个 FSM 只是最基本的, 它不包括诸如糊涂窗口综合征(Nagle 算法) 和窗口关闭(稍后讨论) 等问题。因为它定义的是单向通信, 所以也忽略了所有涉及到双向通信的内容。

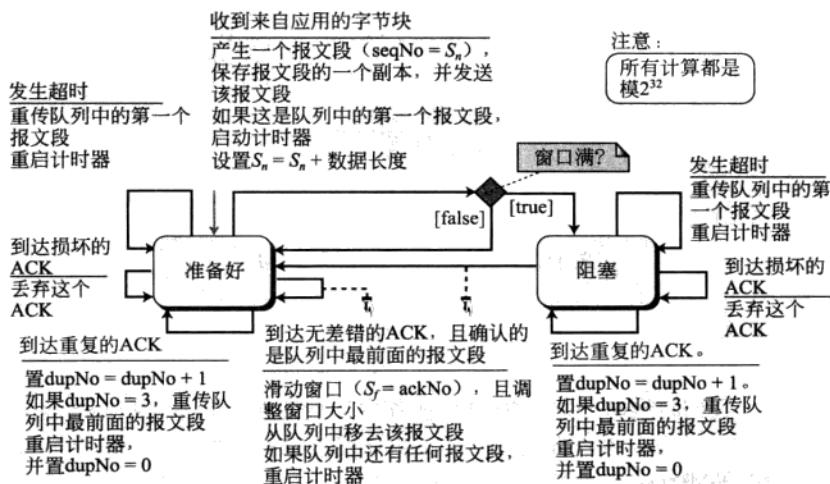


图 15.27 TCP 发送端的简化 FSM

上面这个 FSM 和我们在第 13 章中讨论 SR 协议时的 FSM 有一些区别。其中的一个区别是快重传(三个重复的 ACK 规则)。另一个区别就是窗口大小要根据 rwnd 的值来调整(目前先忽略拥塞控制的问题)。

接收端 FSM

让我们用一个简化的 FSM 来描述 TCP 协议的接收端。这个 FSM 类似于我们所讨论 SR 协议的状态机, 只是有一些专门针对 TCP 的修改。我们假设通信是单向的, 并且报文段使用 ACK 来确认。在此处我们还忽略了选择确认和拥塞控制的问题。图 15.28 描绘了 TCP 接收端的简化 FSM。请注意, 我们忽略了诸如糊涂窗口综合征(Clark 算法) 和窗口关闭

等问题。

同样，这个 FSM 和我们在第 13 章中讨论 SR 协议时的 FSM 也有一些区别。其中一个区别是单向通信中的 ACK 推迟技术。另一个区别是重复 ACK 的发送，以允许发送方实施快重传策略。

我们还要强调的是双向通信时接收方的 FSM 就不像 SR 协议那么简单了。我们需要考虑到其他一些策略，譬如接收方有数据要返回时，它就要发送一个即时的 ACK。

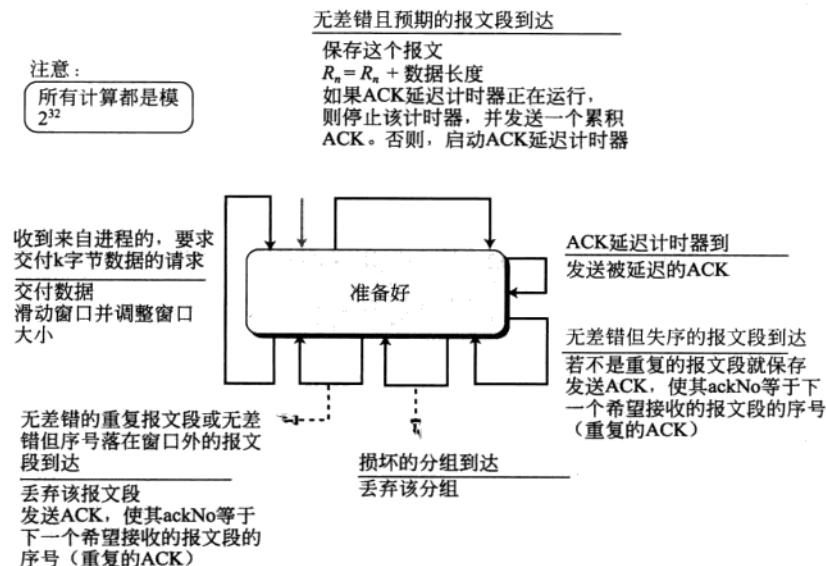


图 15.28 TCP 接收端的简化 FSM

15.8.6 几种情况

在这一小节，我们将例举一些在 TCP 的操作中出现的涉及到差错控制问题的情况。在这些情况中，我们用长方形表示报文段。如果报文段携带有数据，就给出字节编号的范围和确认字段值。如果报文段仅携带确认，就在一个小长方形中给出确认号。

正常运行

第一种情况是两个系统之间的双向数据传送，如图 15.29 所示。客户 TCP 发送了一个报文段，服务器 TCP 发送了三个报文段。图中还指出了每一个确认所遵循的规则。对于客户的第一个报文段和服务器的所有三个报文段，遵循前面给出的规则 1。因为报文段中有要发送的数据，所以显示出下一个期望的字节号。当客户收到服务器的第一个报文段时，它没有更多的数据要发送，因此只发送一个 ACK 报文段。但是，根据规则 2，这个确认需要推迟 500 ms，以便观察是否还有更多的报文段到达。当 ACK 推迟计时器到期时，就触发一个确认。这是因为客户不知道后面是否还有报文段要到达，而它也不能永远推迟确认。当下一个报文段到达时，设置另一个 ACK 推迟计时器。不过，在这个计时器超时之前，第

三个报文段到达了。根据规则 3，第三个报文段的到达触发了另一个确认。我们没有显示 RTO 计时器，因为没有报文段丢失或延迟。我们就假定 RTO 计时器一切正常。

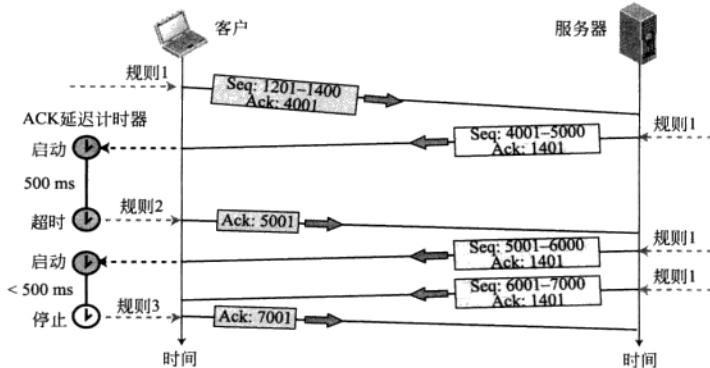


图 15.29 正常操作

报文段丢失

在这种情况下，我们要给出当一个报文段丢失或受到损伤时会发生什么。接收方对待丢失的报文段和受到损伤的报文段是一样的。丢失的报文段就是在网络中的某处被丢弃了，而受损伤的报文段是接收方自己丢弃的，这两种情况都被认为是丢失。图 15.30 描绘了一个报文段丢失的情况（也许是因网络拥塞而被网络中的某个路由器丢弃了）。

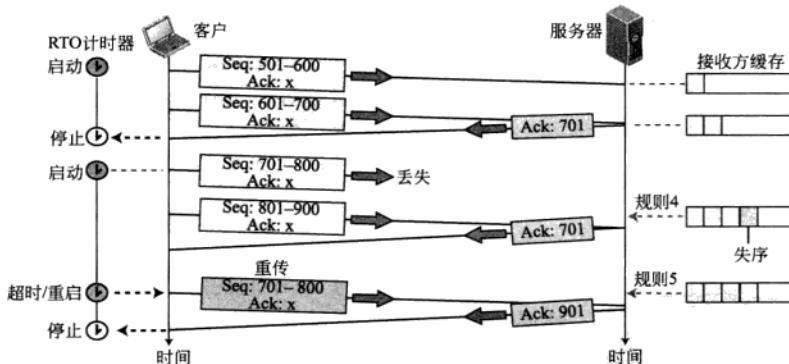


图 15.30 报文段丢失

我们这里假定数据传送是单向的；一端发送，另一端接收。在这种情况下，发送方发送了报文段 1 和 2，然后立即被一个 ACK 确认（规则 3）。但是报文段 3 丢失了，接收方收到报文段 4，顺序被打乱。接收方把这个报文段中的数据在缓存中存储下来，并且留出空隙，表明这里的数据是不连续的。接收方立即向发送方发送确认，并指出期望的下一个字节（规则 4）。请注意，接收方将字节 801~900 保存起来，但在间隙被填入之前绝不会把这些字节交付给应用程序。

接收方 TCP 只把按序的数据交付给应用进程。

发送方 TCP 在整个连接期间只保持一个 RTO 计时器。当第三个报文段超时后，发方 TCP 就重传报文段 3，这次它到达了接收方，并被正常地确认（规则 5）。

快重传

在这种情况下，我们想描绘一下快重传。这种情况与第二种情况基本一样，不同的只是 RTO 具有更大的数值（见图 15.31）。

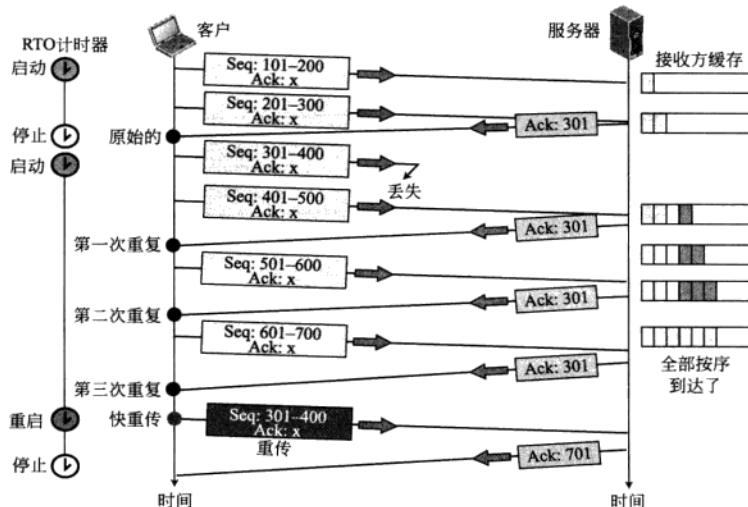


图 15.31 快重传

当接收方收到第四个、第五个和第六个报文段时，都触发了确认（规则 4）。发送方收到四个具有相同值的确认（其中三个是重复的）。虽然计时器还没有到期，但是根据快重传的规则，要求立即重传报文段 3，也就是所有这些重复的确认所期望的报文段。在重传报文段 3 之后，计时器重启。

延迟的报文段

第四种情况突出了延迟的报文段。TCP 使用 IP 的服务，而 IP 是无连接的协议。每个封装有 TCP 报文段的 IP 数据报可能走不同的路由，以不同的时延到达最后的终点。因此，TCP 报文段就有可能被延迟。延迟的报文段有时可能会超时。如果延迟的报文段在它的重传报文段到达之后才到达，会被认为是重复的报文段而被丢弃。

重复的报文段

发送方 TCP 有可能产生重复的报文段，例如，当一个延迟的报文段被接收方误认为丢失时，发方会这样做。终点 TCP 对这种重复的报文段的处理是一个很简单的过程。终点 TCP 期望的字节流是连续的。当一个报文段到达时，若其序号等于已经接收并保存的字节序号，则把这个报文段丢弃。此时需要发送一个 ACK，它的 ackNo 指向所期盼的报文段。

自动纠正丢失的 ACK

这种情况要说明的是一个丢失的确认中的信息会包含在下一个确认中，这也是使用累积确认最重要的优点。图 15.32 描绘了数据的接收方发送的一个确认丢失了。在 TCP 确认机制中，一个丢失的确认甚至可能不会被源点 TCP 注意到。TCP 使用累积的确认系统。我

们说，下一个确认自动地纠正了丢失确认带来的影响。

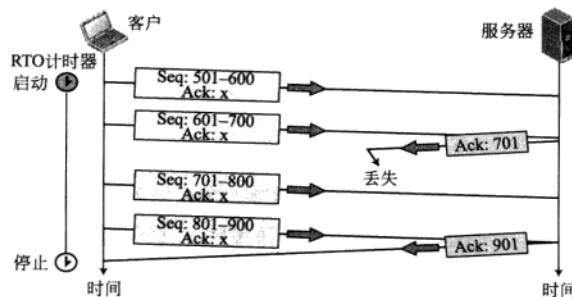


图 15.32 丢失的确认

丢失的确认被重传的报文段纠正

图 15.33 描绘了确认丢失的情况。

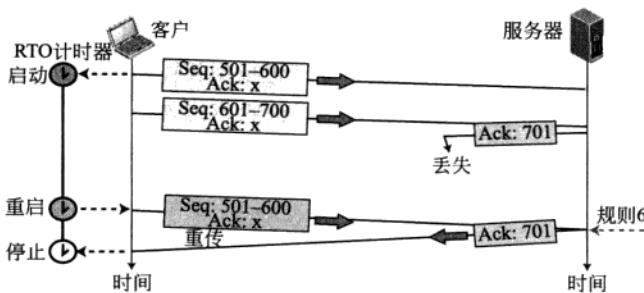


图 15.33 丢失的确认被重传的报文段纠正

如果下一个确认推迟了很长一段时间，或者没有下一个确认（丢失的确认就是最后一个），那么 RTO 计时器就会触发纠正过程，其结果是一个重复的报文段。当接收方收到这个重复的报文段时，就丢弃它，但是要立即重传上次的 ACK 来通知发送方这个报文段（或几个报文段）已经收到了。

请注意，图中虽然有两个报文段没有被确认，但只重传了一个报文段。当发送方收到了重传的 ACK 时，就知道这两个报文段都已经安全完好地传送到对方了，因为确认是累积的。

因确认丢失而产生死锁

也可能有这样一种情况，确认的丢失可能会引起系统的死锁。当接收方发送了确认，同时把 rwnd 置为 0，也就是请求发送方暂时关闭其窗口时，就会出现这种情况。过了一段时间之后，接收方打算取消这一限制，但是如果它没有数据要发送，就会发送 ACK 报文段，并且用 rwnd 等于一个非零的数值来取消这个限制。如果这个确认丢失了，那么就会产生问题。发送方一直在等待确认来宣布非零的 rwnd。而接收方则认为发送方已经收到了这个确认，因而在等待数据。这种情况叫做死锁 (deadlock)，也就是双方都在等待对方的响应，而什么也不会发生。此时重传计时器并没有设置。要避免死锁，就要设计一个持续计时器，

我们将在本章的后面讨论。

如果处理不当，丢失的确认可能会产生死锁。

15.9 拥塞控制

我们在第 13 章中已经简单地讨论过拥塞控制。TCP 中的拥塞控制同时基于开环和闭环两种机制。TCP 使用了一个拥塞窗口和一个拥塞策略来避免拥塞，并在拥塞发生后检测和缓解拥塞。

15.9.1 拥塞窗口

前面我们说明了流量控制，并讨论了接收方由于收到大量的数据而过载时的缓解方法。我们曾提到，发送方的窗口大小是由接收方的可用缓存空间（rwnd）来决定的。换言之，我们假定了只有接收方能够指示发送方应当使用多大的发送窗口。此时，我们完全忽略了另一个实体，即网络。如果网络不能像发送方产生数据那样快地把数据交付给接收方，那么它就必须通知发送方放慢速度。换言之，除了接收方之外，网络是决定发送方窗口大小的第二个实体。

发送方有两种信息：接收方通告的窗口大小和拥塞窗口大小。窗口的真正大小取两者之中较小的一个。

真正的窗口大小 = minimum (rwnd, cwnd)

让我们先简单地说明一下怎样决定拥塞窗口(cwnd)的大小。

15.9.2 拥塞策略

TCP 处理拥塞的一般策略是基于三个阶段：慢开始、拥塞避免和拥塞检测。在慢开始阶段，发送方从非常慢的传输速率开始，但很快就把速率增大到一个门限值。当到达门限后，数据率的增长开始放慢。最后，只要一检测到拥塞，发送方就又回到慢开始或者是拥塞避免阶段，具体取决于拥塞是如何检测到的。

慢开始：指数增大

慢开始 (slow start) 算法是基于这样的想法，即拥塞窗口大小 (cwnd) 从 1 个最大报文段的长度 (MSS) 开始。MSS 的数值是在连接建立时以同样名称的选项来确定的。每当有 1 个报文段被确认，拥塞窗口就增大 1 个 MSS。正如这个名称所暗示的，慢开始算法开始很慢，但按指数规律增大。为了说明这一概念，让我们看图 15.34。我们假定 rwnd 远大于 cwnd，因此发送方的窗口大小总是等于 cwnd。为了保持简单性，我们忽略了推迟 ACK 策略，并假设每个报文段都被单独确认。

发送方从 $cwnd = 1$ MSS 开始。这表示发送方只能发送一个报文段。在第一个 ACK 到达后，拥塞窗口大小增加 1，也就是说现在 cwnd 是 2。此时可以发送两个报文段。当

相应的两个 ACK 到达后，对于每一个 ACK，窗口大小增加 1 MSS，这就表示 cwnd 现在是 4。此时可以发送 4 个报文段。当四个 ACK 到达后，窗口大小又增加了 4，也就是 cwnd 等于 8。

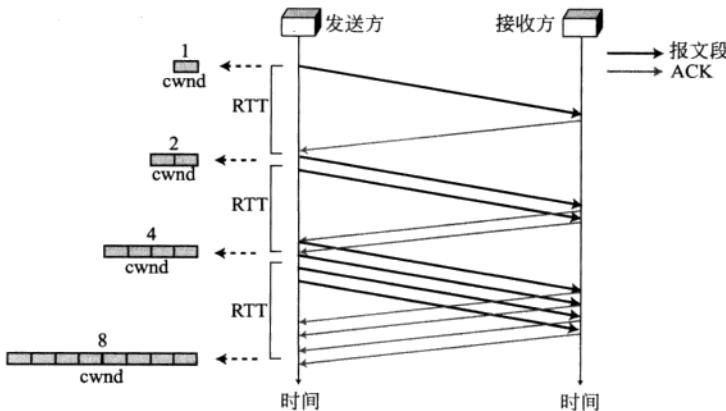


图 15.34 慢开始，指数增大

如果我们从往返时间 RTT 的角度来看 cwnd 的大小，就会发现它的增长速率是按指数规律的，如下所示：

cwnd 由收到的 ACK 决定的啦？

开始	\rightarrow	$cwnd = 1$
经过 1 RTT 后	\rightarrow	$cwnd = 1 \times 2 = 2 \rightarrow 2^1$
经过 2 RTT 后	\rightarrow	$cwnd = 2 \times 2 = 4 \rightarrow 2^2$
经过 3 RTT 后	\rightarrow	$cwnd = 4 \times 2 = 8 \rightarrow 2^3$

但是我们需要说明，在推迟确认的策略下，慢开始策略的增长速度要更慢一些。我们知道，对于每个 ACK，cwnd 只能增加一个 MSS。因此，如果三个报文段被累积确认，那么 cwnd 的值仅增加 1 MSS，而不是 3 MSS。这个增长仍然是指数的，但是它不再是 2 的乘方。如果每两个报文段使用一个确认，那么它大约是 1.5 的乘方。

慢开始不能无限制地继续下去。一定要有一个门限来停止这个阶段。发送方密切关注一个称为 ssthresh (慢开始门限) 的变量。当以字节计的窗口大小到达这个门限时，慢开始阶段就结束了，并进入下一个阶段。← 尼玛就是没说门限怎么定是吧？？？

在慢开始算法中，拥塞窗口大小按指数规律增长，直至达到门限为止。

拥塞避免：加法增大

如果我们从慢开始算法开始，拥塞窗口大小就按指数规律增长。为了在拥塞发生之前避免它，就必须使这种指数增长变慢。TCP 定义了另一个算法，叫做拥塞避免 (congestion avoidance)，使 cwnd 的值按照加法规律增长，而不是按照指数规律。图 15.35 描绘了这一思想。

当拥塞窗口大小达到慢开始的门限时（此时假设 $cwnd = i$ ），慢开始阶段就停止，而加法增加阶段就开始了。在这种算法中，每当一个“窗口”中的报文段都被确认后，拥塞窗口大小才增加 1。一个“窗口”就是指在一个 RTT 期间传输的报文段的数量。换言之，

这个增长是基于 RTT 的，而不是基于到达的 ACK 的数量。为了说明这个思想，我们把这个算法应用到和慢开始一样的情况中。此时，发送方收到了对一个完整窗口大小的报文段的确认后，窗口的大小增加 1 个报文段。如果我们从往返时间 RTT 的角度来看 cwnd 的大小，就会发现速率是按照加法规律增长的，如下所示：

开始	→	$cwnd = i$
经过 1 RTT 后	→	$cwnd = i + 1$
经过 2 RTT 后	→	$cwnd = i + 2$
经过 3 RTT 后	→	$cwnd = i + 3$

在拥塞避免算法中，拥塞窗口的大小按照加法规律增长，直至检测到拥塞为止。

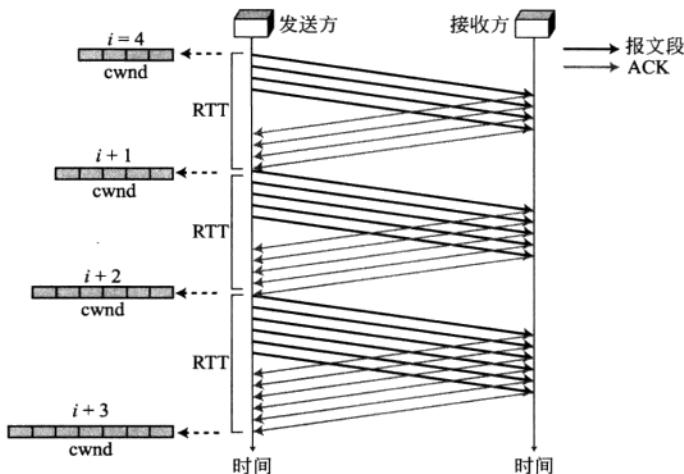


图 15.35 拥塞避免，加法增大

拥塞检测：乘法减小

若拥塞发生了，拥塞窗口的大小就必须减小。让发送方能够猜测到拥塞已发生的唯一现象就是它需要重传一个报文段。这也是 TCP 做出的一个重要的假定，之所以需要重传是为了恢复一个遗失的分组，而这个分组假设是因为某个路由器有太多的输入分组而不得不丢弃，所以才被丢弃掉的（即丢失），也就是说，路由器或者网络已变得超载或拥塞了。不过，重传可以发生在以下两种情况之一：当 RTO 计时器超时，或者是当收到了三个重复的 ACK 时。不管是哪一种情况，门限值都要下降到一半（乘法减小）。大多数的 TCP 实现有下面这两种反应：

1. 如果是计时器超时，那么出现拥塞的可能性就很大。某个报文段可能在网络中的某处被丢弃了，而后续的报文段也没有任何消息。在这种情况下，TCP 有下面较强烈的反应：
 - a. 把门限值设置为当前窗口大小的一半。
 - b. 把 cwnd 重新设置为一个报文段。
 - c. 再次从慢开始阶段开始。
2. 如果是收到了三个 ACK，那么出现拥塞的可能性就较小。某个报文段可能被丢弃

了，但之后的几个报文段已安全到了，因为它收到了三个重复的 ACK。这就称为快重传和快恢复。在这种情况下，TCP 有下面这样较弱的反应：

- 把门限值设置为当前窗口值的一半。
- 把 cwnd 设置为门限值（某些实现会在门限值上再增加 3 个报文段）。
- 启动拥塞避免阶段。

小结

在图 15.36 中，我们归纳了 TCP 的拥塞策略以及这三个阶段之间的关系。我们在图 15.37 中给出了一个例子。假设最大窗口大小的初始值是 32 个报文段。门限的初始值设置为 16 个报文段（最大窗口值的一半）。在慢开始阶段，窗口大小从 1 开始逐渐按指数规律增大，直至窗口大小达到门限值。在达到门限值后，拥塞避免（加法增大）过程使窗口大小按线性规律增长，直至出现了超时现象或者达到了最大窗口大小。在图中，在窗口大小达到 20 时发生了超时现象。此时，乘法减小过程开始生效，并把门限值减小到窗口大小的一半。因为在超时发生时，窗口大小是 20，所以新的门限现在是 10。图中还标注了“这是在说明门限值的设置方法吗？”

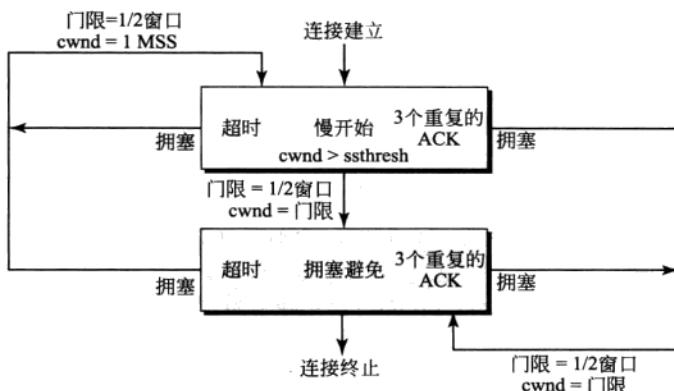


图 15.36 TCP 拥塞策略的小结

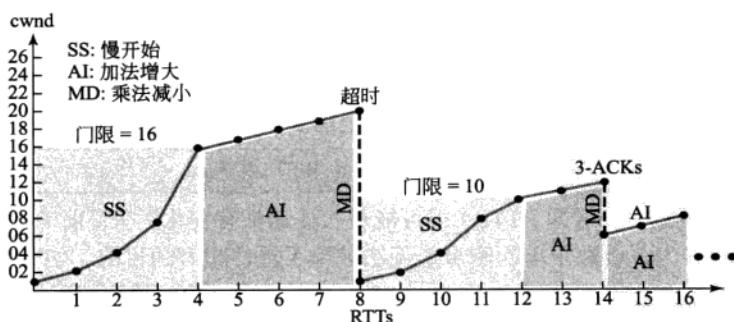


图 15.37 拥塞举例

TCP 重新进入慢开始，并从窗口值为 1 开始，到达新的门限时变为加法增大。当窗口值为 12 时，三个重复的 ACK 事件发生了。现在又转为乘法减小过程。门限值和窗口大小设置为 6，这一次 TCP 进入加法增大阶段。TCP 一直停留在这个阶段，直至发生另一个超

时事件或 3 个重复的 ACK 事件。

15.10 TCP 的计时器

为了能够顺利地进行 TCP 的操作，大多数的 TCP 实现至少要使用 4 个计时器，如图 15.38 所示。



图 15.38 TCP 的计时器

15.10.1 重传计时器

为了重传丢失的报文段，TCP 应用了一个重传计时器（在整个连接期间）来处理重传超时（RTO），也就是对报文段的确认的等待时间。我们可以为重传计时器定义以下规则：

1. 当 TCP 发送了位于发送队列最前端的报文段后，就启动这个计时器。
2. 当这个计时器超时后，TCP 重传位于发送队列最前端的报文段，并重启这个计时器。
3. 当一个（或多个）报文段被累积确认后，这个（或这些）报文段被消除出队列。
4. 如果队列为空，TCP 停止这个计时器；否则，TCP 重启计时器。

往返时间 (RTT)

要计算重传的超时期限 (RTO)，我们首先需要计算往返时间 (round trip time, RTT)。但是，计算 TCP 中的 RTT 是个复杂的过程，我们将通过一些例子逐步了解它。

测量 RTT 我们需要找出从发送出去一个报文段到收到对它的确认需要多少时间。这就是测量 RTT。我们应当记住，报文段和它的确认之间并非一对一的关系，好几个报文段有可能会被一起确认。一个报文段的测量 RTT 是指这个报文段到达终点并被确认所需要的时间，虽然这个确认可能还包含对其他报文段的确认。请注意，在 TCP 中，任何时刻只能有一个正在进行的 RTT 测量。也就是说，如果 RTT 的测量开始了，那么在这次的 RTT 测量结束之前，不能再开始其他测量。我们使用记法 RTT_M 表示测量 RTT。

在 TCP 中，任何时刻只能有一个正在进行的 RTT 测量。

平滑 RTT 测量 RTT (即 RTT_M) 很可能每次往返都有变化。在目前的因特网中，RTT 测量值的起伏非常大，以致于单次测量值无法被用于重传超时的目的。绝大多数实现使用的是一种平滑的 RTT，记为 RTT_S ，它是对 RTT_M 和前一个 RTT_S 的加权平均，如下所示：

最初	→	没有数值
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 的取值与实现有关，但通常为 1/8。换言之，新的 RTT_S 是由 7/8 的旧 RTT_S 和 1/8 的当前 RTT_M 相加而成。

RTT 的偏差 大多数的实现不仅使用了 RTTs，还要计算 RTT 的偏差，称为 RTTD。它是根据 RTTs 和 RTTM 并使用下面的公式计算得到的：

最初	→	没有数值
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 $

β 的取值与实现有关，但通常置为 1/4。

重传超时 (RTO)

RTO 的数值基于平滑的往返时间和它的偏差值。绝大多数实现使用下面的公式计算 RTO:

原始	→	初始值
在任一次测量后	→	$RTO = RTT_S + 4 \times RTT_D$

换言之，就是用当前的 RTT_s 值，加上当前的 RTT_D 值（通常是个较小的数）的四倍。

例 15.3

让我们给出一个假想的例子。图 15.39 所示为一个连接的一部分。图中画出了连接建立阶段和部分的数据传送阶段。

1. 当 SYN 报文段发送后, 没有 RTT_M 、 RTT_S 或 RTT_D 的值。RTO 值设为 6.00 秒。下面绘出此时这些变量的值为:

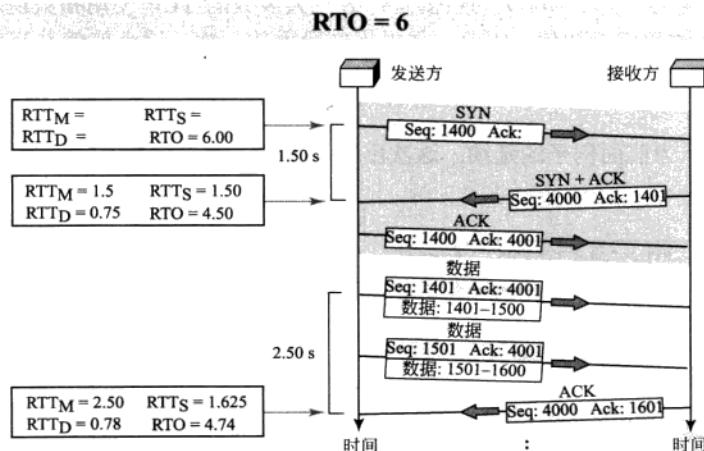


图 15.39 例 15.3

2. 当 SYN + ACK 报文段到达时，测量出 RTT_M 等于 1.5 秒。下面给出这些变量的值：

$$RTT_M = 1.5$$

RTT_s = 1.5

$$RTT_D = 1.5 / 2 = 0.75$$

$$RTO = 1.5 + 4 \times 0.75 = 4.5$$

3. 当第一个数据报文段被发送后，新的 RTT 测量就开始了。请注意，在发送方发送 ACK 报文段时不能开始一次 RTT 的测量过程，因为 ACK 报文段不消耗序号，也没有超时。在发送第二个数据报文段时也没有测量 RTT，因为已经有一个 RTT 测量正在进行之中。最

后一个 ACK 报文段的到达用来计算 RTT_M 的下一个数值。虽然最后一个 ACK 报文段确认了两个数据报文段（累积的），但它的到达完成的是对第一个数据报文段的 RTT_M 的计算。现在这些变量的值如下所示：

$$RTT_M = 2.5$$

$$RTT_S = 7/8 \times 1.5 + (1/8) \times 2.5 = 1.625$$

$$RTT_D = 3/4 \times (0.75) + (1/4) \times |1.625 - 2.5| = 0.78$$

$$RTO = 1.625 + 4 \times 0.78 = 4.74$$

Karn 算法

假定有一个报文段在重传计时器超时前未被确认，因而重传。当发送 TCP 收到对这个报文段的确认时，它无法知道这个确认是对原来报文段的确认，还是对重传报文段的确认，因为新的 RTT 值要根据报文段发送的时间来计算。但是，如果原始的报文段丢失了而确认是对重传的报文段的确认，则当前的 RTT 的计算就必须从重传的报文段发送的时间算起。这种模糊性被 Karn 解决了。Karn 的算法很简单。在计算新的 RTT 时，不要考虑重传报文段的往返时间。除非你发送了一个报文段并且在没有被重传的条件下收到了确认，否则不要更新 RTT 的值。

TCP 在计算新的 RTO 时不考虑重传报文段的 RTT。

指数退避

如果发生了重传，那么 RTO 的数值是什么？大多数的 TCP 使用指数退避的策略。每一次重传，RTO 的数值就加倍。因此，如果报文段重传一次，这个值就是两倍的 RTO。如果它被重传了两次，这个值就变成了四倍的 RTO，依此类推。

例 15.4

图 15.40 所示为前面例子的延续。这次出现了重传，因而用到 Karn 算法。

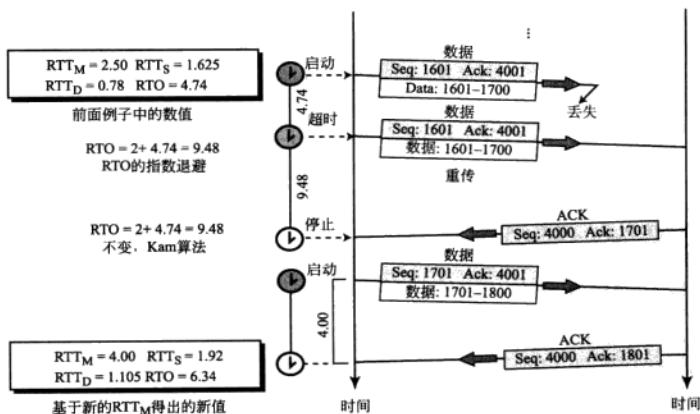


图 15.40 例 15.4

图中的第一个报文段发送出去，但丢失了。经过 4.74 秒后，RTO 计时器到期。这个报文段被重传，而计时器被设置为 9.48 秒，是原来的 RTO 值的两倍。这次在计时器超时之前收到了 ACK。我们继续等待，直至我们发送一个新的报文段，并收到了对它的确认，然后才能重新计算 RTO (Karn 算法)。

15.10.2 持续计时器

为了处理零窗口值的通告，TCP 需要另一个计时器。如果接收 TCP 宣布窗口值为零，那么发送 TCP 就会停止发送报文段，直至接收 TCP 发送来一个宣布窗口大小非零的确认。但是这个 ACK 报文段有可能会丢失。应当记住，在 TCP 中的确认报文段既不需要确认，也不需要重传。若确认报文段丢失了，接收 TCP 仍然认为这个确认已经完成了任务，并等待着发送 TCP 发送下面的报文段。对于仅含有一个确认的报文段是没有重传计时器的。发送 TCP 由于没有收到确认，就等待对方发送一个确认来通知窗口的大小。这两个 TCP 都在永远地等待着对方（死锁）。

为了纠正这个死锁问题，TCP 为每个连接使用一个持续计时器 (persistence timer)。当发送 TCP 接收到窗口值为零的确认时，就启动一个持续计时器。当持续计时器超时后，发送 TCP 就发送一个特殊的报文段，称为探测报文段。这个报文段只有 1 个字节的新数据。它有序号，但它的序号永远不需要确认，甚至在计算剩余数据的序号时，这个序号也被忽略。探测报文段促使接收 TCP 重传一个确认。

持续计时器的时间长度被设置为重传时间的值。但是，若没有收到从接收方发来的响应，则需发送另一个探测报文段，并将其持续计时器的值加倍，且该计时器复位。发送方继续发送探测报文段，不断地将持续计时器的值加倍和复位，直到这个值达到一个门限（通常是 60 秒）为止。在这以后，发送方每隔 60 秒就发送一个探测报文段，直到窗口重新打开。

15.10.3 保活计时器

在某些实现中要使用保活计时器 (keepalive timer) 来防止两个 TCP 之间的连接长时间空闲。假定一个客户打开了到服务器的一条连接，传送过一些数据，然后就变得静默了。也许是因为这个客户出了什么故障。在这种情况下，这个连接将永远处于打开状态。

为了解决这个问题，绝大多数的实现都是给服务器设置了一个保活计时器。每当服务器收到客户的信息，就把该计时器复位。超时通常设置为两小时。若服务器过了两小时还没有收到客户的任何信息，它就发送一个探测报文段。若连续发送了 10 个探测报文段（每隔 75 秒一个）还没有收到响应，它就假定客户出了故障，并终止这个连接。

15.10.4 TIME-WAIT 计时器

TIME-WAIT (2MSL) 计时器是在连接终止期间使用的。我们已在第 15.5 节（状态转换图）讨论过设置这个计时器的原因了。

15.11 选项

TCP 首部可以有多达 40 字节的可选信息。选项用于把附加信息传递给终点，或用来填

充对齐其他选项。我们将定义两大类选项：1字节选项和多字节选项。第一类选项包括两种选项：选项列表结束和无操作。在大多数实现中，第二类选项包括了五种选项：最大报文段长度、窗口扩大因子、时间戳、允许 SACK 和 SACK（见图 15.41）。

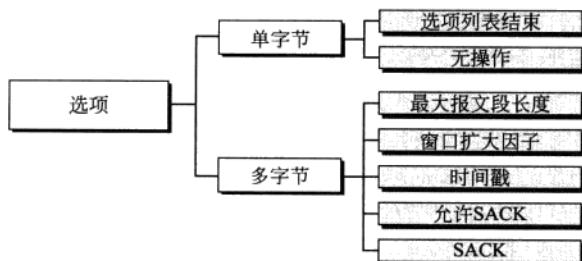
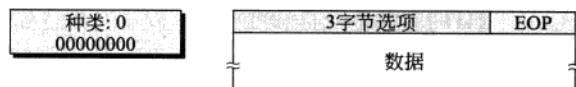


图 15.41 选项

选项结束 (EOP)

选项结束 (end-of-option, EOP) 选项是 1 字节选项，用来在选项区的结尾处进行填充。它只能用作最后一个选项。这个选项只允许出现一次。在这个选项之后，接收方就要检查有效载荷数据了。图 15.42 给出了一个例子。在首部后面有一个 3 字节的选项，在这个选项后面紧跟着的是数据。因此要用一个 EOP 选项插入进来使数据对齐下一个字的开头。



(a) 选项列表结束

(b) 用于填充

图 15.42 选项结束

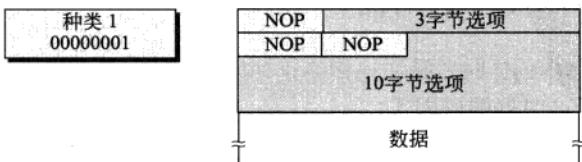
EOP 只能使用一次。

EOP 选项向终点传达了两种信息：

1. 首部中没有更多的选项了。
2. 从应用程序传递来的数据开始于下一个 32 位字开始的地方。

无操作 (NOP)

无操作 (no-operation, NOP) 选项也是 1 字节选项，用作选项之间的填充。但是，它通常用在另一个选项之前，帮助选项能够在 4 字节一格中对齐。例如，在图 15.43 中，它被用来对齐一个 3 字节的选项（如窗口扩缩因子）和一个 10 字节的选项（如时间戳）。



(a) 无操作选项

(b) 用于对齐一个选项的开始

图 15.43 无操作选项

NOP 可多次使用。

最大报文段长度 (MSS)

最大报文段长度选项 (maximum-segment-size option) 定义了能够被终点接收的 TCP 报文段的最大数据单元。虽然它的名字是这样，但它定义的是数据的最大长度，而不是报文段的最大长度。因为这个字段是 16 位长，所以这个值只能在 0 到 65535 字节之间。图 15.44 给出了这个选项的格式。

种类: 2 00000010	长度: 4 00000100	最大报文段长度
1字节	1字节	2字节

图 15.44 最大报文段长度选项

MSS 是在连接建立阶段确定的。每一方都要定义它在连接期间接收的报文段的 MSS。若有一方没有定义 MSS 的大小，则使用默认值，即 536 字节。

MSS 的值是在连接建立阶段确定的，在连接期间保持不变。

窗口扩大因子

首部中的窗口大小字段定义了滑动窗口大小。这个字段的长度是 16 位，表示这个窗口的范围可以从 0~65535 字节。虽然看起来这是一个非常大的窗口，但它还是有可能不够用，特别是当数据在一个长粗管道（即距离很远且带宽很大的信道）中传送时。

为了加大这个窗口大小，就要使用窗口扩大因子 (window scale factor)。新的窗口大小可以这样求出，即先计算 2 的窗口扩大因子次方，再把得出的结果乘以首部中的窗口值。

新的窗口值 = 首部中定义的窗口值 × 2^{窗口扩大因子}

图 15.45 给出了窗口扩大因子选项的格式。

种类: 3 00000011	长度: 3 00000011	扩大因子
1字节	1字节	1字节

图 15.45 窗口扩大因子选项

扩大因子有时称为移位计数，因为要把一个数乘以 2 的几次方，只需把这个数按位操作向左移几位就行。换言之，窗口大小的实际值可以这样确定：把分组中所显示的窗口值向左移动窗口扩大因子这样多的位数即可。

例如，假定窗口扩大因子的值为 3。一个端点收到的确认所给出窗口大小是 32768，那么它可用的窗口大小就是 32768×2^3 或 262144 字节。把 32768 向左移动 3 位也可以得到同样的数值。

虽然扩大因子最高可达到 255，但 TCP/IP 所容许的最大值是 14，这就表示最大窗口值可以是 $2^{16} \times 2^{14} = 2^{30}$ ，它小于序号的最大值。请注意，窗口的大小不能超过序号的最大值。

窗口扩大因子只能在连接建立阶段确定。在数据传送阶段，窗口大小（在首部中指明）可以改变，但它必须乘以相同的扩大因子。

窗口扩大因子只能在连接建立阶段确定，在连接期间它的值不能改变。

请注意，连接的一端可以把窗口扩大因子设置为 0，意思是说虽然它支持这个选项，

但在目前这个连接中，它不想使用这个选项。

时间戳

这是一个 10 字节的选项，其格式如图 15.46 所示。请注意，主动打开的那一端在连接请求报文段（SYN 报文段）中宣布了一个时间戳。如果它来自对方的下一个报文段（SYN + ACK）中也收到一个时间戳，那么就表示允许使用时间戳，否则就不再使用它。时间戳选项（timestamp option）有两个应用：测量往返时间和防止序号绕回。

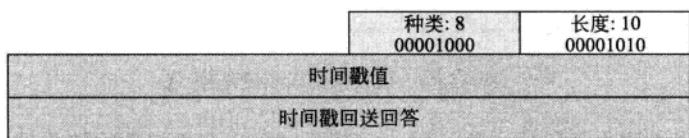


图 15.46 时间戳选项

测量 RTT 时间戳可用来测量往返时间（RTT）。当 TCP 准备好发送一个报文段时，就读取系统时钟值，并把这个 32 位数值插入到时间戳值字段中。接收方在发送对这个报文段的确认或包括了这个报文段的字节的累积确认时，就把收到的时间戳复制到时间戳回送回答字段中。发送方在收到该确认时，就用时钟所示的当前时间减去时间戳回送回答的数值，从而找出 RTT。

请注意，在这里不需要对发送方和接收方的时钟进行同步，因为所有的计算都是基于发送方的时钟。还应注意的是，发送方不需要记住或存储发送出去的报文段的时间，因为报文段本身就携带了这个时间。

接收方需要保持两个变量。第一个是 lastack，这是最近一次发送的确认号。第二个是 tsrecent，这是还没有回送的最近的一个时间戳。当接收方收到一个所包含的字节号与 lastack 匹配的报文段时，就把其中的时间戳字段的值保存到 tsrecent 变量中。在接收方发送对该报文段的确认时，再把 tsrecent 变量的值插入到回送回答字段中。

时间戳选项的一个应用就是测量往返时间（RTT）。

例 15.5

图 15.47 给出了连接的一端计算往返时间的例子。如果要计算另一端的 RTT，那么只需反过来即可。

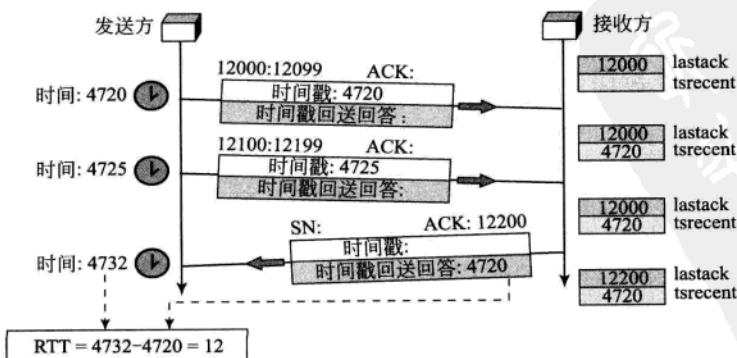


图 15.47 例 15.5

发送方只是简单地把时钟值（例如，从午夜到现在所经过的秒数）插入到第一个和第二个报文段的时间戳字段中。在收到确认时（第三个报文段），发送方检查其时钟值，然后从当前时钟值减去回送回答字段中的数值。在这种情况下得出 RTT 为 12 秒。

接收方的功能要更加复杂些，它要保留上次发送出去的确认号（12000）。当第一个报文段到达时，其中包含的字节序号是 12000~12099。第一个字节的序号与 lastack 的值一样。接收方就把该报文段的时间戳值（4720）复制到 tsrecent 变量中。lastack 的数值仍然是 12000（没有新的确认发送出去）。当第二个报文段到达时，因为这个报文段中的字节号没有和 lastack 数值一致的，因此就忽略它的时间戳字段的值。当接收方决定要发送一个累积确认（确认号为 12200）时，就把 lastack 的数值改为 12200，同时把 tsrecent 的数值插入到回送回答字段中。此后 tsrecent 的数值保持不变，直至收到携带字节 12200 的新报文段（下一个报文段）。

请注意，如本例所给出的，计算出来的 RTT 是从发送第一个报文段开始到收到第三个报文段为止的时间。这是 RTT 的真正含义：从一个分组发送出去至收到对它的确认的时间。第三个报文段携带了对第一个和第二个报文段的确认。

PAWS 时间戳选项还有另一个应用，就是防止序号绕回（Protection Against Wrapped Sequence numbers，PAWS）。在 TCP 中定义的序号长度是 32 位。虽然这是一个大数目，但在一个高速连接中，序号还是有可能发生绕回。这表示如果在某个时刻有个序号是 n ，那么有可能在这同一条连接的生存期中，序号会再次出现 n 。现在，若第一个报文段被复制了，而它在第二轮序号中到达，那么属于过去的报文段就有可能错误地被认为是属于新一轮序号的报文段。

解决这个问题的一个方法就是增加序号的长度，但这会涉及到窗口的增大以及报文段格式的改变等等。最简单的解决方法就是在报文段的标志中加入时间戳。换言之，可以用时间戳和序号的组合来标志一个报文段，其实也就是增加了标识的长度。两个报文段 400:12001 和 700:12001 明确地属于不同的化身（incarnation）。第一个是在时间 400 发送的，第二个是在时间 700 发送的。

时间戳选项可用于 PAWS。

允许 SACK 和 SACK 选项

正如我们在前面讨论的，TCP 报文段中的确认字段被设计为累积确认，也就是说，它报告了收到的最后一个连续的字节，但没有报告已经失序到达的那些字节。同样，它对重复的报文段也保持沉默。这些都可能对 TCP 的性能产生负面的影响。如果某些分组丢失或被丢弃，发送方必须等待计时器超时，并重传所有未被确认的报文段。接收方就有可能收到一些重复的分组。为了改进性能，人们提出了选择确认（SACK）。选择确认使发送方能够更好地知道哪些报文段真正丢失了，而哪些报文段已经失序到达了。新的建议甚至还包括了一个重复报文段的清单。如此一来，发送方就可以仅仅发送那些真正丢失的报文段。重复报文段的清单能够帮助发送方找出哪些报文段是由于较短的超时而被重传了。

这个建议定义了两个新的选项：允许 SACK 和 SACK，如图 15.48 所示。

两个字节长度的允许 SACK 选项（SACK-permitted option）只用在连接建立阶段。发

送 SYN 报文段的主机增加这个选项以说明它能够支持 SACK 选项。如果另一端在它的 SYN + ACK 报文段中也包含了这个选项，那么双方在数据传送阶段就能使用 SACK 选项。请注意，允许 SACK 选项是不能在数据传送阶段使用的。

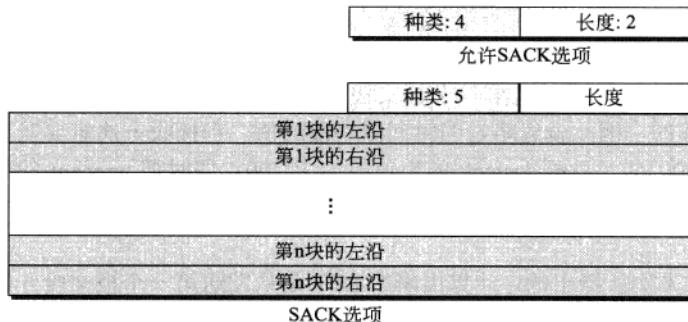


图 15.48 SACK

可变长度的 SACK 选项 (SACK option) 只能在双方都已同意 (即双方已在连接建立阶段交换了允许 SACK 选项) 的前提下，用于数据传送阶段。这个选项包含了一张失序到达的数据块的列表。每个块占用两个 32 位数，它们分别定义了块的开始和结束。我们将在一些例子中说明这个选项的用途，就目前而言，只要记住 TCP 所允许的选项大小仅有 40 字节，也就是说，SACK 选项定义的块不能超过 4 个。因为 5 个块的信息要占据 $(5 \times 2) \times 4 + 2$ 或 42 字节，它超出了报文段的选项区的长度。如果 SACK 选项和其他选项一起使用，那么块的数目还要减少。

SACK 选项的第一个块可用来报告重复收到的报文段。这仅在具体的实现里允许这个特点时才能使用。

例 15.6

让我们来看一下 SACK 选项是怎样把失序到达的数据块列出的。在图 15.49 中，连接的某端收到了五个数据报文段。(译注：本例题与后面两个例题在原理上是正确的，但与 RFC 2018 不一致。详细信息请参阅 RFC 文档。)

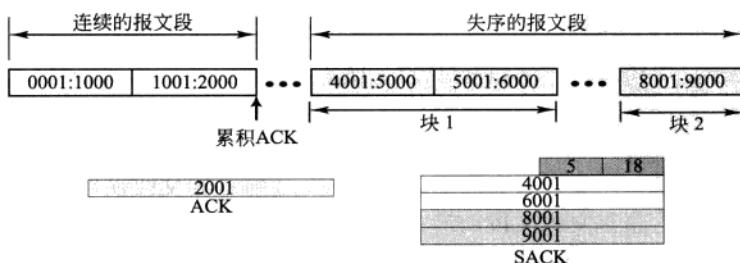


图 15.49 例 15.6

第一个和第二个报文段的顺序是连续的，可以对这两个报文段发送一个累积确认。但是报文段 3、4 和 5 都是失序的，在第二个和第三个报文段之间，以及在第四个和第五个报

文段之间都有间隙。通过 ACK 和 SACK 的组合，可以很容易地让发送方明白这种情况。ACK 的值是 2001，这表示发送方不需要再担心字节 1~2000。SACK 有两个块。第一个块宣布字节 4001~6000 已失序到达了。第二个块说明字节 8001~9000 也已失序到达。这就表明字节 2001~4000 和字节 6001~8000 丢失或被丢弃了。发送方可以只重传这些字节。

例 15.7

图 15.50 给出了用 ACK 和 SACK 的组合可以检测出重复报文段的例子。在这个例子中，我们有一些失序到达的报文段（在一个块中）和一个重复的报文段。为了同时指出失序的和重复的数据，在这种情况下，SACK 利用第一个块指出重复的报文段，并利用其他的块指出失序到达的数据。请注意，只有第一个块可被用于重复的数据。人们自然要问，发送方在收到这些 ACK 和 SACK 的值时是怎样能够知道第一个块是用来指出重复的数据（把本例和前一个例子相比）。答案就在于第一个块中的字节是在 ACK 字段中已经确认过的，因此，这个块指出的必定是重复的数据。

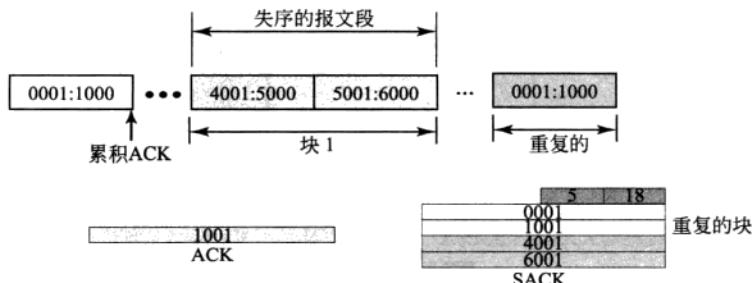


图 15.50 例 15.7

例 15.8

图 15.51 所示的例子指出，如果在失序到达的这部分报文段中有一个还是重复的，那么会发生什么？在这个例子中，有一个报文段（4001:5000）是重复的。

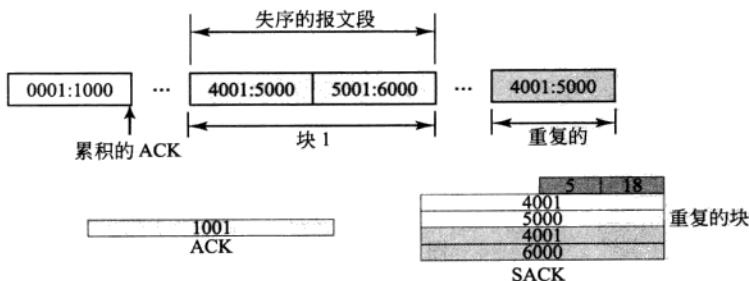


图 15.51 例 15.8

SACK 选项先宣布重复的数据，然后才是失序的块。但这时，重复的块还没有被 ACK 确认，不过由于它是失序块中的一部分（4001:5000 是 4001:6000 的一部分），发送方还是能够知道第一个块指出的是重复数据。

15.12 TCP 软件包

TCP 是一个非常复杂的协议。它是提供流服务的、面向连接的、并且有一张很复杂的状态转换图的协议。它使用了流量控制和差错控制。这个协议如此复杂以至于它的实际代码可达到数万行之多。

在本节，我们给出一个简化的 TCP 软件包的核心结构。我们的目的只是想说明我们能够像状态转换图所示的那样来模拟 TCP 的核心部分。

这个软件包涉及了几张称为传输控制块的表，一组计时器和三个软件模块：主模块、输入处理模块和输出处理模块。图 15.52 描绘了这五个构件及其交互关系。

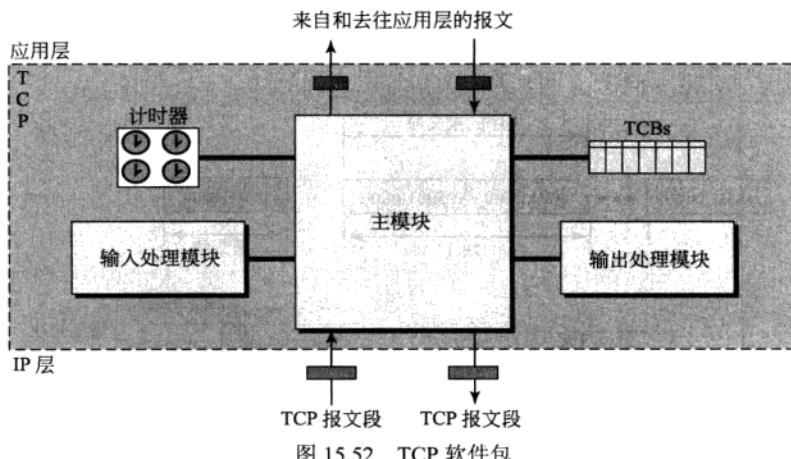


图 15.52 TCP 软件包

15.12.1 传输控制块 (TCB)

TCP 是面向连接的运输协议。一个连接可以打开很长一段时间。为了控制连接，TCP 要使用一种存储结构来保持每一条连接的有关信息，这就称为传输控制块 (TCB)。因为在任何时候都可能有好几个连接，所以 TCP 就以表的形式保存 TCB 的数组。这个表通常就称为 TCB（见图 15.53）。



图 15.53 TCB

在每个 TCB 都包括了许多字段。我们在这里仅给出一些最常用的：

- **状态** 这个字段按照状态转换图定义连接状态。
- **进程** 这个字段定义了本机上使用该连接的进程（作为客户或服务器）。
- **本地 IP 地址** 这个字段定义了连接使用的本机 IP 地址。

- **本地端口号** 这个字段定义了连接使用的本地端口号。
- **远程 IP 地址** 这个字段定义了连接的远程机器的 IP 地址。
- **远程端口号** 这个字段定义了连接使用的远程端口号。
- **接口** 这个字段定义的是本地接口。
- **本地窗口** 这个字段可以包括多个子字段，用来保持本地 TCP 的窗口信息。
- **远程窗口** 这个字段可以包括多个子字段，用来保持远程 TCP 的窗口信息。
- **发送序号** 这个字段保持发送序号。
- **接收序号** 这个字段保持接收序号。
- **发送 ACK 号** 这个字段保持已发送的 ACK 号。
- **往返时间** 多个字段，可用来保持关于 RTT 的信息。
- **超时值** 多个字段，可用来保持不同的超时值，如重传超时、持续超时、保活超时，等等。
- **缓存大小** 这个字段定义本地 TCP 的缓存大小。
- **缓存指针** 这个字段是指向缓存的指针，收到的数据都存放在缓存中，直到它们被应用程序读出。

15.12.2 计时器

我们在前面已经讨论过，TCP 需要几个计时器来监视其操作。

15.12.3 主模块

TCP 报文段的到达、超时事件的发生或来自应用程序的报文都可以调用主模块（表 15.3）。这是一个非常复杂的模块，因为它要采取的动作取决于 TCP 的当前状态。有好几种方法可以实现状态转换图，包括为每一个状态使用一个进程，或使用一张表（二维数组），等等。为了使我们的讨论简单，我们使用一些分支情况来处理状态。我们一共有 11 种状态，因此要使用 11 种不同的情况。每一种状态按照状态转换图中定义的来实现。对 ESTABLISHED 状态需要进一步的解释。当 TCP 处于这个状态时，若有数据或确认报文段到达，另一个模块（即输入处理模块）被调用进行处理。此外，当 TCP 处于这个状态时，若从应用程序发出了一个“发送数据”的报文，则另一个模块（即输出处理模块）被调用进行处理。

表 15.3 主模块

1	TCP_Main_Module (segment)
2	{
3	查找 TCB 表
4	If(相应的 TCB 未找到)
5	创建 TCB，它的状态为 CLOSED
6	找出 TCB 表中相应表项的状态

```

7     Switch (状态)
8     {
9         case CLOSED 状态:
10        if (收到“被动打开”报文)
11            进入 LISTEN 状态
12        if (收到“主动打开”报文)
13        {
14            发送 SYN 报文段
15            进入 SYN-SENT 状态
16        }
17        if (收到任何报文段)
18            发送 RST 报文段
19        if (收到其他任何报文)
20            发出差错报文
21        break
22
23     case LISTEN 状态:
24     if (收到“发送数据”报文)
25     {
26         发送 SYN 报文段
27         进入 SYN-SENT 状态
28     }
29     if (收到任何 SYN 报文段)
30     {
31         发送 SYN + ACK 报文段
32         进入 SYN-RCVD 状态
33     }
34     if (收到任何其他报文段或者报文)
35         发出差错报文
36     break
37
38     case SYN-SENT 状态:
39     if (超时)
40         进入 CLOSED 状态
41     if (收到 SYN 报文段)
42     {
43         发送 SYN + ACK 报文段

```

```
44          进入 SYN-RCVD 状态
45      }
46      if (收到 SYN + ACK 报文段)
47      {
48          发送 ACK 报文段
49          进入 ESTABLISHED 状态
50      }
51      if (收到任何其他报文段或者报文)
52          发出差错报文
53      break
54
55 case SYN-RCVD 状态:
56     if (收到 ACK 报文段)
57         进入 ESTABLISHED 状态
58     if (超时)
59     {
60         发送 RTS 报文段
61         进入 CLOSED 状态
62     }
63     if (收到“关闭”报文)
64     {
65         发送 FIN 报文段
66         进入 FIN-WAIT-1 状态
67     }
68     if (收到 RTS 报文段)
69         进入 LISTEN 状态
70     if (收到任何其他报文段或者报文)
71         发出差错报文
72     break
73
74 case ESTABLISHED 状态:
75     if (收到 FIN 报文段)
76     {
77         发送 FIN 报文段
78         进入 CLOSED-WAIT 状态
79     }
80     if (收到“关闭”报文)
```

```
81      {
82          发送 FIN 报文段
83          进入 FIN-WAIT-1 状态
84      }
85      if (收到 RTS 或 SYN 报文段)
86          发出差错报文
87      if (收到数据或 ACK 报文段)
88          调用输入模块
89      if (收到“发送”报文)
90          调用输出模块
91      break
92
93  case FIN-WAIT-1 状态:
94      if (收到 FIN 报文段)
95      {
96          发送 ACK 报文段
97          进入 CLOSING 状态
98      }
99      if (收到 FIN + ACK 报文段)
100     {
101         发送 ACK 报文段
102         进入 FIN-WAIT 状态
103     }
104     if (收到 ACK 报文段)
105         进入 FIN-WAIT-2 状态
106     if (收到任何其他报文段或者报文)
107         发出差错报文
108     break
109
110    case FIN-WAIT-2 状态:
111    if (收到 FIN 报文段)
112    {
113        发送 ACK 报文段
114        进入 TIME-WAIT 状态
115    }
116    break
117
```

```

118     case CLOSING 状态:
119         if (收到 ACK 报文段)
120             进入 TIME-WAIT 状态
121         if (收到任何其他报文段或者报文)
122             发出差错报文
123         break
124
125     case TIME-WAIT 状态:
126         if (超时)
127             进入 CLOSED 状态
128         if (收到任何其他报文段或者报文)
129             发出差错报文
130         break
131
132     case CLOSED-WAIT 状态:
133         if (收到“关闭”报文)
134         {
135             发送 FIN 报文段
136             进入 LAST-ACK 状态
137         }
138         if (收到任何其他报文段或者报文)
139             发出差错报文
140         break
141
142     case LAST-ACK 状态:
143         if (收到 ACK 报文段)
144             进入 CLOSED 状态
145         if (收到任何其他报文段或者报文)
146             发出差错报文
147         break
148     } //模块结束

```

15.12.4 输入处理模块

在我们的设计中，当 TCP 处于 ESTABLISHED 状态时，对收到数据或确认后所需要的处理细节都是由输入处理模块来完成的。这个模块在需要时会发送 ACK，并负责宣布窗口大小，负责进行差错检查，等等。这个模块的许多细节在这本入门的教材中并不

是必要的。

15.12.5 输出处理模块

在我们的设计中，当 TCP 处于 ESTABLISHED 状态时，对于发送从应用程序处接收来的数据所需的一切细节都是由输出处理模块完成的。这个模块处理重传超时、持续超时，等等。实现这个模块的一种方法是使用一个小的转换图来处理不同的输出条件。同样，这个模块的许多细节在这本入门的教材中并不是必要的。

15.13 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

15.13.1 参考书

有几本书全面地讨论了 TCP，包括 [Pet & Dav 03]、[Com 06]、[Tan 03] 和 [Ste 94]。

15.13.2 RFC

有多份 RFC 讨论了 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 和 RFC 3782。

15.14 重要术语

Clark 解决方法

保活计时器

拥塞避免

最大报文段长度选项

Cookie

乘法减小

数据传送

Nagle 算法

死锁

无操作（NOP）选项

拒绝服务攻击

持续计时器

选项结束（EOP）选项

防止序号绕回（PAWS）

快重传

重传超时（RTO）

半关闭

往返时间（RTT）

初始序号

SACK 选项

Karn 算法

允许 SACK 选项

报文段	慢开始
选择确认 (SACK)	SYN 洪泛攻击
糊涂窗口综合征	三向握手
同时关闭	时间戳选项
同时打开	窗口扩大因子

15.15 本章小结

- 传输控制协议 (TCP) 是 TCP/IP 协议族中的一个运输层协议。TCP 提供进程到进程的、全双工的和面向连接的服务。两个设备之间使用 TCP 软件传送的数据单元称为报文段，它有 20~60 字节的首部，首部的后面是来自应用程序的数据。
- TCP 的连接通常包括三个阶段：连接建立、数据传送和连接终止。连接建立需要三向握手；连接终止需要三向或者四向握手。TCP 通常是被当作一个有限状态机 (FSM) 实现的。
- TCP 应用流量控制来避免接收方因数据过多而超载，它的流量控制是以滑动窗口机制实现的。TCP 的窗口大小由接收方通告的窗口值 ($rwnd$) 或拥塞窗口值 ($cwnd$) 来决定，取其中较小的一个值。窗口可以被接收方打开或关闭，但不能收缩。每一条连接中传送的数据字节都被 TCP 编了号。编号从一个随机产生的数开始。
- TCP 应用差错控制来提供可靠的服务。差错控制利用检验和、确认和超时来处理。受损伤的和丢失的报文段最终都会被重传，重复的报文段要丢弃。数据可能不按序到达，接收 TCP 会把它们暂时存储下来，但 TCP 保证交付给进程的报文段都是按序的。在比较新的实现中，如果重传计时器到期或者有三个重复的 ACK 报文段到达，就要进行重传。
- TCP 应用拥塞控制来避免和检测网络中的拥塞。拥塞控制使用了慢开始（指数增大）、拥塞避免（加法增大）和拥塞检测（乘法减小）等策略。在慢开始算法中，拥塞窗口大小先按指数规律增长，直到它到达门限。在拥塞避免算法中，拥塞窗口大小按加法规律增长，直到检测出拥塞。不同的实现对拥塞检测的反应不同。如果是通过超时检测到拥塞，就开始新的慢开始阶段；如果是通过三个 ACK 检测出拥塞，就开始新的拥塞避免阶段。
- TCP 在运行中使用了四个计时器（重传计时器、持续计时器、保活计时器和时间等待计时器）。在 TCP 中，任何时刻只能有一个 RTT 测量正在进行。在计算 RTO 时，TCP 不考虑重传报文段的 RTT。
- TCP 使用了若干个选项以提供更多的服务。最大报文段长度选项在连接建立时用于定义最大容许的 TCP 报文段长度。MSS 的值在连接建立阶段确定，而在连接期间不能改变。窗口扩大因子是用来扩大窗口大小的一个乘数。时间戳选项用于指出数据从发送方到接收方共花费了多少时间。时间戳选项的一个应用是计算往返时间 (RTT)。另一个应用是用于 PAWS。TCP 最新的实现还使用另外

两个选项：允许 SACK 选项和 SACK 选项。这两个选项允许接收方选择确认所收到的报文段。

15.16 实践安排

15.16.1 习题

1. 试比较 TCP 的首部和 UDP 的首部。列出在 TCP 首部中出现，但在 UDP 首部中没有的字段。给出它们出现的理由。
2. 一个携带 TCP 报文段的 IP 数据报的目的地址为 130.14.16.17。因为目的端口地址受到损伤，它到达了 130.14.16.19。收方 TCP 将怎样响应这个差错？
3. 有一个 ICMP 报文（在第 9 章中讨论的）报告了目的端口不可达的差错。TCP 如何检测出是目的端口出了差错？
4. UDP 是面向报文的协议。TCP 是面向连接的协议。如果一个应用程序需要保护它的报文的边界，那么应当使用 UDP 还是 TCP？
5. TCP 首部的最大长度是多少？TCP 首部的最小长度是多少？
6. 若 HLEN 的值是 0111，在报文段中包含了多少字节的选项？
7. 若 TCP 报文段携带从 FTP 客户到 FTP 服务器的报文，试给出其首部的各项。把检验和字段填入 0。选项适当的临时端口号和正确的熟知端口号。数据的长度是 40 字节。
8. 当 TCP 报文段控制字段的值分别如下所示时，你认为它说明了什么？
 - a. 000000
 - b. 000001
 - c. 010001
 - d. 000100
 - e. 000010
 - f. 010010
9. 下面是打印出的 TCP 首部，以十六进制表示。


```
(05320017 00000001 00000000 500207FF 00000000)16
```

 - a. 源端口号是什么？
 - b. 目的端口号是什么？
 - c. 序号是什么？
 - d. 确认号是什么？
 - e. 首部长度是什么？
 - f. 报文段的类型是什么？
 - g. 窗口大小是多少？
10. TCP 报文段的控制字段为 6 位。我们可以有 64 种不同的位组合。请列出其中有效的组合。

11. 为了给初始序号一个随机数，大多数系统在引导时使计数器从 1 开始，然后每半秒钟增加 64000。试问经过多少时间这个计数器就要绕回？
12. 在 TCP 连接中，客户端的初始序号是 2171。客户打开连接，只发送了一个携带有 1000 字节数据的报文段，然后关闭连接。试问下面从客户端发送的各个报文段的序号分别是多少？
- SYN 报文段。
 - 数据报文段。
 - FIN 报文段。
13. 在一个连接中， $cwnd$ 的值是 3000 而 $rwnd$ 的值是 5000。主机已经发送了 2000 字节，都还没有被确认。试问还可以发送多少个字节？
14. TCP 以 14534 为初始序号 (ISN) 打开了一个连接，而对方以 21732 为 ISN 打开该连接。
- 试给出在连接建立阶段的三个 TCP 报文段。
 - 试给出数据传输阶段的报文段的内容。假定发起者发送的报文段包含报文“Hello dear customer”，而另一方的回答报文段包含“Hi there seller”。
 - 试给出连接终止阶段的报文段的内容。
15. 某客户使用 TCP 向服务器发送数据。数据共包括 16 字节。试计算它在 TCP 级的传输效率 (有用字节与总字节数之比)。试计算它在 IP 级的传输效率。假定 IP 首部无选项。试计算它在数据链路层的传输效率。假定 IP 首部无选项目且在数据链路层使用以太网。
16. TCP 以每秒 1MB 的速率发送数据。若序号从 7000 开始，试问过多长时间后序号返回到了零？
17. 一个 TCP 连接使用的窗口大小为 10000 字节，且上一次的确认号是 22001。它收到了一个报文段，确认号是 24001，且它通告的窗口大小是 12000。试用图来说明在这之前与之后的窗口情况。
18. 一个窗口保存着字节 2001 到 5000。下一个要发送的字节是 3001。试用图来说明在发生了以下两个事件之后的窗口情况。
- 收到了一个 ACK 报文段，其确认号是 2500，且通告的窗口大小是 4000。
 - 发送了一个携带 1000 字节的报文段。
19. 某 TCP 连接正处于 ESTABLISHED 状态。以下的事件相继发生：
- 收到一个 FIN 报文段。
 - 应用程序发送“关闭”报文。
- 在每个事件之后，连接的状态是什么？在每个事件之后采取的动作是什么？
20. 某 TCP 连接正处于 ESTABLISHED 状态。以下的事件相继发生：
- 应用程序发送“关闭”报文。
 - 收到一个 ACK 报文段。
- 在每个事件之后，连接的状态是什么？在每一个事件之后采取的动作是什么？
21. 某主机没有数据要发送。它在如下所示的不同时间收到了各个报文段 (以午夜后的时:分:秒:毫秒来表示)。试给出该主机发送的确认。
- 在 0:0:0:000 收到报文段 1。

- b. 在 0:0:0:027 收到报文段 2。
 - c. 在 0:0:0:400 收到报文段 3。
 - d. 在 0:0:1:200 收到报文段 4。
 - e. 在 0:0:1:208 收到报文段 5。
22. 某主机发送了五个分组，收到了三个确认。时间表示为小时:分:秒。
- a. 在 0:0:00 发送报文段 1。
 - b. 在 0:0:05 发送报文段 2。
 - c. 在 0:0:07 收到对报文段 1 和 2 的 ACK。
 - d. 在 0:0:20 发送报文段 3。
 - e. 在 0:0:22 发送报文段 4。
 - f. 在 0:0:27 发送报文段 5。
 - g. 在 0:0:45 收到对报文段 1 和 2 的 ACK。
 - h. 在 0:0:65 收到对报文段 3 的 ACK。

试计算 RTT_M , RTT_S , RTT_D 和 RTO 的值，假定一开始的 RTO 是 6 秒。发送方有没有漏掉对某个报文段的重传？试找出哪一个报文段应当被重传，以及应当在什么时候重传？重新写出包括重传时间的事件序列。

23. 如果某主机已经按序收到了字节 2001~3000, 字节 4001~6000 失序到达了，而字节 3501~4000 是重复的。试给出应当发送的 SACK 选项的内容。
24. 给出下面各情况如图 15.37 所示的拥塞控制图。假定最大窗口大小是 64 个报文段。
- a. 在第四个 RTT 之后收到了三个重复的 ACK。
 - b. 在第六个 RTT 之后发生了超时。
25. 请画出同时关闭情况下的状态转换图 (FSM) (参见图 15.19)。
26. 请画出拒绝连接情况下的状态转换图 (FSM) (参见图 15.20)。
27. 请画出异常终止连接情况下的状态转换图 (FSM) (参见图 15.19)。
28. 某发送窗口的 $S_f = 401$ 且 $S_n = 701$ 。如果该窗口大小为 1000 字节，请画出这个发送窗口在该站点接收到 $ackNo = 601$ 且 $rwnd = 700$ 的确认之前和之后的状态。忽略拥塞控制。这种情况是否意味着窗口的收缩？
29. 请为以下情况画出一张类似于图 15.25 的时间线图 (忽略差错控制和拥塞控制)：
- a. 时间 1：客户发送了 $seqNo = 301$ 的 SYN 报文段。
 - b. 时间 2：服务器将其缓存大小设置为 2000 字节。
 - c. 时间 3：服务器对 SYN 报文段进行确认。
 - d. 时间 4：客户发送了 SYN + ACK 报文段，其中包含了 300 字节的数据。
 - e. 时间 5：客户发送了携带 400 字节数据的报文段。
 - f. 时间 6：服务器进程拉取了 400 字节的数据。
 - g. 时间 7：服务器发送 ACK。
 - h. 时间 8：客户发送了携带 300 字节数据的报文段。
 - i. 时间 9：服务器进程拉取了 300 字节的数据。
 - j. 时间 10：服务器发送 ACK。
30. 请为以下情况画出一张时间线图 (类似于图 15.29)：

- a. 客户发送携带字节 1401~1700 的报文段。该报文段顺利抵达服务器端。
 - b. 服务器发送携带字节 2001~2100 的报文段，同时对收到的来自客户的一个报文段进行确认。该报文段顺利抵达客户端。
 - c. 客户发送携带字节 1701~1900 的报文段，同时对收到的报文段进行确认。但是该报文段丢失了。
 - d. 客户发送携带字节 1901~2100 的报文段。但是该报文段丢失了。
 - e. 客户端发生超时。
 - f. 客户对超时做出响应，重传报文段。该报文段顺利抵达。
 - g. 服务器在 ACK 延迟计时器超时后发送确认。
 - h. 客户端再次发生超时。
 - i. 客户对超时做出响应，重传报文段。该报文段顺利抵达。
 - j. 服务器在 ACK 延迟计时器超时后发送确认。
31. 重画图 15.34 中的时间线图，以允许服务器延迟确认，并在每达到 cwnd 窗口量的数据后发送 ACK。

15.16.2 研究活动

- 32. 我们没有给出关于状态图和 TCP 状态的所有的规则。为了让它完整，我们应当给出当任何类型的报文段到达时，对于任何一个状态的下一个状态。TCP 应当知道，当它处在某个状态时，若有任一类型的报文段到达，它应当采取什么动作。请给出一些此类规则。
- 33. 什么是 TCP 中的“半打开”情况？
- 34. 什么是 TCP 中的“半双工关闭”情况？
- 35. UNIX 或 Linux 的 `tcpdump` 命令可用来打印一个网络接口的分组首部。试用 `tcpdump` 观察报文段的发送和接收。

第 16 章 流控制传输协议 (SCTP)

在这一章我们将要描述一个称为 SCTP 的新运输层协议。SCTP 是能够处理多媒体和流通信量(stream traffic)的通用运输层协议，如今这种类型的通信量在因特网上与日俱增。

目标

本章有以下几个目标：

- 作为一个新的运输层协议来介绍 SCTP。
- 讨论 SCTP 的服务，并将其与 TCP 进行比较。
- 列出并解释 SCTP 中使用的各种分组类型，并讨论每种分组的作用及其中的各个字段。
- 讨论 SCTP 关联并解释其不同的情况，如关联建立、数据传送、关联终止和关联异常终止。
- 对照比较 SCTP 和 TCP 的相应状态转换图。
- 解释 SCTP 中使用的流量控制机制，并讨论发送方和接收方的行为。
- 解释 SCTP 中使用的差错控制机制，并讨论发送方和接收方的行为。
- 解释 SCTP 中使用的拥塞控制机制，并将其与 TCP 中类似的机制进行比较。

16.1 引言

流控制传输协议 (Stream Control Transmission Protocol, SCTP) 是一个新的、可靠的、面向报文的运输层协议。图 16.1 给出了 SCTP 在 TCP/IP 协议族中与其他协议的关系。SCTP 协议位于应用层和网络层之间，它提供介于应用程序和网络功能之间的服务。

但是，SCTP 主要还是为最近才出现在因特网上的一些应用程序而设计的。这些新的应用，如 IUA（在 IP 上运行 ISDN）、M2UA 和 M3UA（电话信令）、H.248（媒体网关控制）、H.323（IP 电话）和 SIP（IP 电话）等，都需要一些比 TCP 所能提供的还要复杂的服务。SCTP 提供了这种增强的性能和可靠性。我们简单地来比较一下 UDP、TCP 和 SCTP：

- UDP 是面向报文 (message-oriented) 的协议。进程把报文交付给 UDP，再由 UDP 封装成用户数据报并发送到网络上。UDP 保留了报文的边界，且每一个报文和其他的报文之间没有关系。正如我们将在本书的后面部分所看到的，当我们在处理如 IP

电话或实时数据传输这样的应用时，这种特性正是我们所希望有的。但是，UDP 是不可靠的，发送方无法掌握那些发送出去的报文的命运。这些报文可能丢失、重复或不按顺序收到。UDP 还缺少其他一些功能，如拥塞控制和流量控制，这些功能是一个友善的运输层协议应当具有的。

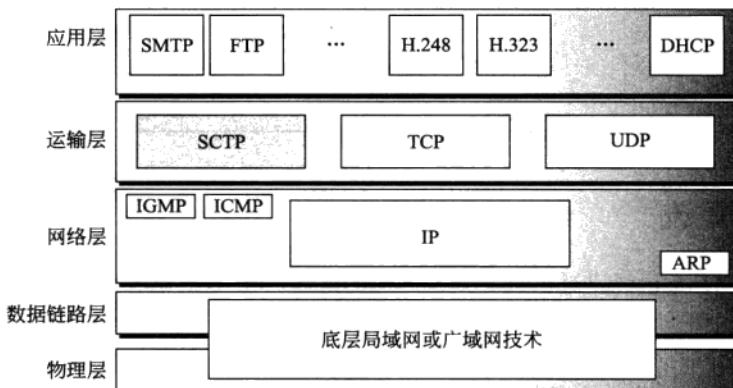


图 16.1 TCP/IP 协议族

- TCP 是面向字节 (byte-oriented) 的协议。TCP 接收来自进程的一个或多个报文，并把它们以字节流的形式存储起来，再以报文段为单位发送出去。这里不保留报文的边界。但是，TCP 是可靠的协议。重复的报文段能够被检测出来，丢失的报文段会重传，所有的字节要按序交付给目的进程。TCP 还有拥塞控制和流量控制机制。
- SCTP 结合了 UDP 和 TCP 的优点。SCTP 是可靠的面向报文的协议。它保留了报文的边界，与此同时也检测丢失的数据、重复的数据以及失序的数据。SCTP 还有拥塞控制和流量控制机制。稍后我们还会看到，SCTP 还有其他一些 UDP 和 TCP 都没有提供的新颖的特点。

SCTP 是面向报文的可靠的协议，它结合了 UDP 和 TCP 的优点。

16.2 SCTP 的服务

在我们讨论 SCTP 的工作原理之前，先让我们来说明一下 SCTP 向应用层进程所提供的服务。

16.2.1 进程到进程的通信

SCTP 使用了 TCP 空间的所有熟知端口。表 16.1 列出的是 SCTP 使用的一些额外的端口号。

表 16.1 某些 SCTP 应用

协议	端口号	描述
IUA	9990	在 IP 上运行 ISDN
M2UA	2904	SS7 电话信令
M3UA	2905	SS7 电话信令
H.248	2945	媒体网关控制
H.323	1718、1719、1720、11720	IP 电话
SIP	5060	IP 电话

16.2.2 多重流

我们在第 15 章已经讲过，TCP 是面向流的协议。在 TCP 客户和服务器之间的每一条连接都包含一个单独的流。这种处理方法的问题就在于这样的流中间任何一点出现的丢失都会使剩余部分数据的交付被阻塞。当我们在传送文本时，这是可以接受的，但是当我们传送像音频或视频这样的实时数据时就不行了。而 SCTP 允许在每一个连接中有多重流服务（multistream service），这在 SCTP 的术语中称为关联（association）。当某一个流被阻塞时，其他的流还能用来继续交付数据。这个概念和高速公路上的多个车道相似。每一个车道可用于不同类型的交通流量。例如，一个车道可用于正常的交通，而另一个车道可用于拼车专用车道（美国某些州为提高交通效率，专门开辟一条拼车专用车道，超过 1 个乘员的轿车和公交车等可在此专用车道上行驶——译者注）。如果正常车道交通堵塞，拼车车道上的车仍然能够到达它们的目的地。图 16.2 描绘了多重流交付的概念。

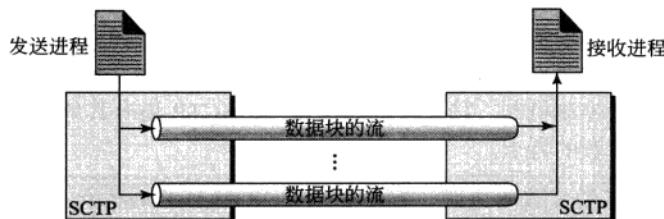


图 16.2 多重流的概念

在 SCTP 中的一个关联可以包含多重流。

16.2.3 多重归属

一个 TCP 只涉及到一个源 IP 地址和一个目的 IP 地址。这就表示，即使发送方或接收方是多重归属主机（即连接超过一个物理网络，并具有多个 IP 地址的主机），在连接期间每一端也只能使用一个 IP 地址。与此不同的是，SCTP 关联则支持多重归属服务（multihoming service）。一个关联两端的发送主机或接收主机可以定义多个 IP 地址。在这种容许故障的处理方法中，当一条路径出故障时，就可以使用其他的接口交付数据，而不会

使交付中断。当我们发送或接收像 IP 电话这样的实时有效载荷时，这种容许故障的特点是非常有用的。图 16.3 描绘了多重归属的概念。



图 16.3 多重归属的概念

在图 16.3 中，一个客户连接到两个本地网络，并具有两个 IP 地址。服务器也是连接到两个本地网络，并具有两个 IP 地址。客户和服务器可以使用四组不同的 IP 地址对构成一个关联。但是请注意，在目前的 SCTP 实现中，只有一对 IP 地址可用于正常的通信，只有当主选路径出了故障时，才可以使用另外的选择。换言之，目前的 SCTP 还不允许在不同的路径之间共享负载。

SCTP 关联允许每一端使用多个 IP 地址。

16.2.4 全双工通信

和 TCP 一样，SCTP 也提供全双工服务 (full-duplex service)，即数据在同一时间可以双向流动。因此每个 SCTP 都有发送缓存和接收缓存，并且分组可以双向发送。

16.2.5 面向连接的服务

和 TCP 一样，SCTP 也是面向连接的协议。但是，在 SCTP 中的连接称为关联 (association)。当站点 A 的进程想要与站点 B 的进程之间发送或接收数据时，就采取以下步骤：

1. 两个 SCTP 彼此之间建立关联。
2. 数据在两个方向上交换。
3. 这个关联被终止。

16.2.6 可靠的服务

和 TCP 一样，SCTP 也是一个可靠的运输协议。它使用确认机制来检查到达的数据的是否安全完好。我们将在后面差错控制一节中讨论这个问题。

16.3 SCTP 的特点

我们先讨论 SCTP 的一般特点，然后再把这些特点和 TCP 的特点进行比较。

16.3.1 传输序号 (TSN)

TCP 的数据单位是字节。在 TCP 中，数据的传送是通过对字节进行编号来控制的。与此不同的是，SCTP 中的数据单位是数据块，这些数据块与来自进程的报文可以是也可以不是一一对应的关系（因为有分片处理，稍后讨论）。在 SCTP 中，数据传送是由对数据块的编号来控制的。SCTP 使用 **传输序号** (transmission sequence number, TSN) 对数据块进行编号。换言之，TSN 在 SCTP 中的作用与 TCP 中的序号作用相似。TSN 的长度为 32 位，并取 $0 \sim (2^{32}-1)$ 之间的一个随机数初始化。每一个数据块必须在其首部携带相应的 TSN。

在 SCTP 中，数据块用 TSN 来编号。

16.3.2 流标识符 (SI)

TCP 的每一个连接中只有一个流，而 SCTP 的每一个关联中可能有多个流。SCTP 中的每个流都需要用一个流标识符 (Stream identifier, SI) 来进行标记。每一个数据块必须在其首部中携带这个 SI，这样，当数据块到达终点时，就可以把它放在相应流中的正确位置上。SI 是从 0 开始的一个 16 位数。

为了区分不同的流，SCTP 使用流标识符 SI。

16.3.3 流序号 (SSN)

当数据块到达目的 SCTP 时，它就被以合适的顺序交付到相应的流中。这就表示，除了 SI 外，SCTP 还应当为每个流中的每个数据块定义一个流序号 (stream sequence number, SSN)。

为了区分属于同一个流中的不同数据块，SCTP 使用了 SSN。

16.3.4 分组

在 TCP 中，用报文段来携带数据和控制信息，其中所携带的数据是一些字节的集合，而控制信息是在首部中用 6 个控制标志来定义的。SCTP 的设计则完全不同，数据由数据块携带，控制信息由控制块携带。若干个控制块和数据块被包装成一个分组。SCTP 中的分组的作用和 TCP 中的报文段的作用相似。图 16.4 对 TCP 的报文段和 SCTP 的分组进行了比较。

TCP 有报文段；SCTP 有分组。

我们将在下一节讨论 SCTP 的分组格式。目前就让我们简单地给出 SCTP 分组和 TCP 报文段之间的区别：

1. TCP 的控制信息是首部的一部分，而 SCTP 中的控制信息则在控制块中。控制块有多种类型，每一种用于不同的目的。

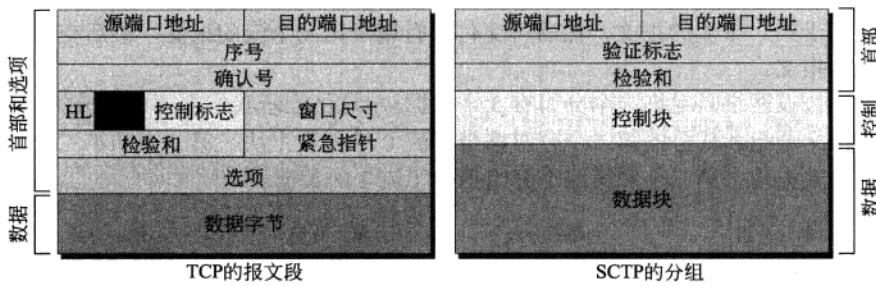


图 16.4 TCP 的报文段和 SCTP 的分组比较

2. TCP 报文段中的数据被当作一个实体，而一个 SCTP 分组可携带多个数据块，且每个数据块可以属于不同的流。

3. 选项部分可以作为 TCP 报文段的一部分，但在 SCTP 分组中则没有选项部分。SCTP 中的选项通过定义新的块类型来解决。

4. TCP 首部中必要的部分有 20 字节，而 SCTP 的通用首部只有 12 字节。SCTP 首部较短是因为：

a. SCTP 序号 (TSN) 属于各个数据块，因此被放置在数据块的首部中。

b. 确认号和窗口大小是每个控制块的一部分。

c. SCTP 不需要有首部长度字段（在 TCP 报文段中首部长度是 HL 字段），因为没有选项会使首部的长度成为可变的，SCTP 的首部长度是固定的（12 字节）。

d. 如我们将要看到的，在 SCTP 中不需要紧急指针。

5. TCP 的检验和是 16 位，SCTP 的检验和是 32 位。

6. SCTP 的验证标志是一个关联标识符，这在 TCP 中是没有的。在 TCP 中，IP 地址和端口地址组合起来定义了一条连接，而在 SCTP 中我们可以有多个使用不同 IP 地址的归属，必须用一个唯一的验证标志来定义每一个关联。

7. TCP 在首部中有一个序号用来定义数据部分的第一个字节的编号。SCTP 分组可以有若干个不同的数据块，TSN、IS 和 SSN 被用来定义每一个数据块。

8. 在 TCP 中，某些只携带控制信息的报文段（如 SYN 和 FIN）也必须消耗一个序号，而 SCTP 中的控制块永远不使用 TSN、IS 或 SSN 这样的编号，这三种标识符仅属于数据块，而不属于整个分组。

SCTP 的控制信息和数据信息处在不同的块中。

在 SCTP 中，我们有数据块、流和分组。一个关联可以发送多个分组，一个分组可以包含若干个块，而这些块可以属于不同的流。为了使这些名词的定义更加清楚，我们假定进程 A 需要向进程 B 用三个流发送 11 个报文。前 4 个报文在第一个流中，紧接着的 3 个报文在第二个流中，最后的 4 个报文在第三个流中。

虽然，如果一个报文很长的话，可以用多个数据块来运载，但我们在里假定每个报文都能装入一个数据块中。因此，我们在三个流中有 11 个数据块。

应用进程向 SCTP 交付了 11 个报文，每个报文都打上相应的流的标记。虽然进程可以先交付第一个流的一个报文，然后再交付第二个流的另一个报文，但此处我们假定进程先

交付所有属于第一个流的报文，然后再交付所有属于第二个流的报文，最后交付所有属于第三个流的报文。

我们还假设网络只允许一个分组有 3 个数据块，这就表示我们需要四个分组，如图 16.5 所示。第一个分组和部分的第二个分组携带了流 0 中的数据块，第二个和第三个分组携带了流 1 中的数据块；第三个和第四个分组携带了流 2 的数据块。

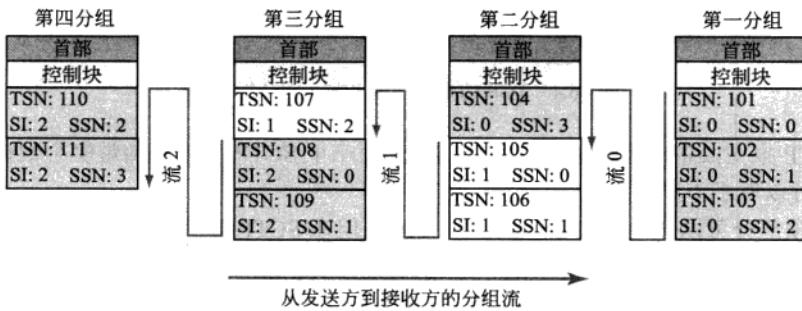


图 16.5 分组、数据块和流

请注意，每个数据块需要三个标识符：TSN、SI 和 SSN。TSN 是一个累积编号，用于流量控制和差错控制，这点将在后面讨论。SI 定义这个块属于哪一个流。SSN 定义在一个特定的流里的各个块的顺序。在我们的例子中，对每一个流，SSN 都是从 0 开始。

数据块用三个标识符来标志：TSN、SI 和 SSN。

TSN 是个标志了关联的累积编号； SI 定义各个流； SSN 定义一个流中的数据块。

16.3.5 确认号

TCP 的确认号是面向字节的，并且指向序号。SCTP 的确认号则是面向数据块的，它们指向 TSN。TCP 确认和 SCTP 确认的第二个区别在于控制信息。回想一下，确认号是属于 TCP 报文段头部中的一部分，TCP 在对只携带控制信息的报文段进行确认时，还是要使用序号和确认号（例如，SYN 报文段需要用 ACK 报文段来确认）。但是在 SCTP 中，控制信息由控制块携带，它不需要 TSN。这些控制块由另一个的相应类型的控制块来确认（有些不需要确认）。例如，INIT 控制块由 INIT-ACK 块来确认。这里不需要序号或确认号。

在 SCTP 中，确认号仅用来确认数据块；

如有必要，控制块用其他控制块来确认。

16.3.6 流量控制

如同 TCP 一样，SCTP 应用了流量控制以避免使接收方负荷过重。我们将在本章的后面讨论 SCTP 的流量控制。

16.3.7 差错控制

如同 TCP 一样, SCTP 应用了差错控制来提供可靠性。TSN 编号和确认号用于差错控制。我们将在本章的后面讨论差错控制。

16.3.8 拥塞控制

如同 TCP 一样, SCTP 应用了拥塞控制来决定有多少数据块可以注入到网络中。我们将在本章的后面讨论拥塞控制。

16.4 分组格式

在这一小节, 我们要介绍分组的格式以及不同类型块的格式。本节中给出的大部分内容在后面的介绍中可以变得更加清楚, 因此在第一次阅读时可以跳过本节, 或只当作参考使用。一个 SCTP 分组由一个强制性的通用首部和若干个块组成。这些块可分为两类: 控制块和数据块。控制块用于控制和维护关联, 数据块携带用户数据。在一个分组中, 控制块在数据块的前面。图 16.6 给出了 SCTP 分组的通用格式。

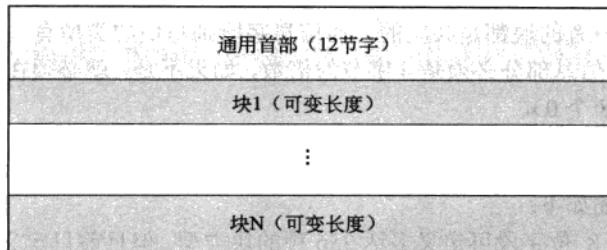


图 16.6 SCTP 的分组格式

在 SCTP 分组中, 控制块在数据块之前。

16.4.1 通用首部

通用首部 (分组首部) 定义了分组所属关联的两个端点, 以保证该分组属于某个特定的关联, 并维护分组内容 (包括首部本身) 的完整性。通用首部的格式如图 16.7 所示。

源端口地址 16位	目的端口地址 16位
验证标志 32位	
检验和 32位	

图 16.7 通用首部

在通用首部中共有四个字段：

- 源端口地址 这是一个 16 位字段，定义了发送分组的进程的端口号。
- 目的端口地址 这是一个 16 位字段，定义了接收分组的进程的端口号。
- 验证标志 这是一个使分组与关联相匹配的数值。有了它就可以避免前一个关联的分组被误认为是现在关联的分组。它被当作关联的标识符使用，在这个关联的每一个分组中都要重复出现。关联的两个方向使用不同的验证标志。
- 检验和 这个 32 位字段包含 CRC-32 检验和（见附录 D）。请注意，检验和的长度从 16 位（在 UDP、TCP 和 IP 中）增加到了 32 位，因而允许使用 CRC-32 检验和。

16.4.2 块（chunk）

控制信息或用户数据都放在块中。块的通用格式如图 16.8 所示。

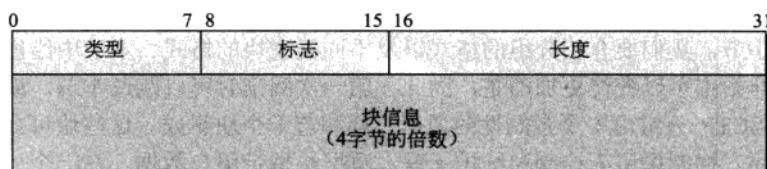


图 16.8 块的通用格式

前三个字段对所有的块都是共同的，而信息字段则与块的类型有关。这里需要记住的要点是，SCTP 要求信息部分必须是 4 字节的倍数，如果不是，就必须在这部分的最后加入一些填充字节（即 8 个 0）。

块需要结束在 32 位（4 字节）的边界上。

公共字段的描述如下：

- **类型** 这个 8 位字段可定义多达 256 种的块类型。但目前只定义了几种，其余的保留为今后使用。表 16.2 列举出了块的名称和描述。
- **标志** 这个 8 位字段定义了特定的块可能需要的特殊标志。根据块的类型不同，每一位都有不同的含义。

表 16.2 块

类型	块的名称	描述
0	DATA	用户数据
1	INIT	建立关联
2	INIT ACK	确认 INIT 块
3	SACK	选择确认
4	HEARTBEAT	探测对方存活
5	HEARTBEAT ACK	确认 HEARTBEAT 块
6	ABORT	异常中止关联

续表

类型	块的名称	描述
7	SHUTDOWN	终止关联
8	SHUTDOWN ACK	确认 SHUTDOWN 块
9	ERROR	报告差错但不关闭
10	COOKIE ECHO	在关联建立中的第三个分组
11	COOKIE ACK	确认 COOKIE ECHO 块
14	SHUTDOWN COMPLETE	在关联终止中的第三个分组
192	FORWARD TSN	调整累积的 TSN

□ 长度 因为信息部分的长度取决于块的类型，所以我们需要定义块的边界。这个 16 位字段定义了以字节计算的块的总长度，包括类型、标志和长度字段。如果一个块不携带任何信息，那么它的长度字段就是 4 (4 字节)。请注意，填充的长度（如果有的话）是不包含在长度字段的计算中。这有助于接收方找出一个块中携带了多少有用的字节。如果这个数值不是 4 的倍数，那么接收方也知道有多少填充。例如，当接收方发现一个块的长度为 17，而它知道下一个 4 的倍数是 20，因此共有 3 字节的填充必须丢弃。但是，如果接收方发现它的长度是 16，那么就没有填充。

长度字段值不包含填充的字节数。

DATA 块

DATA 块携带用户数据。一个分组可以包含零个或多个数据块。图 16.9 给出了 DATA 块的格式。

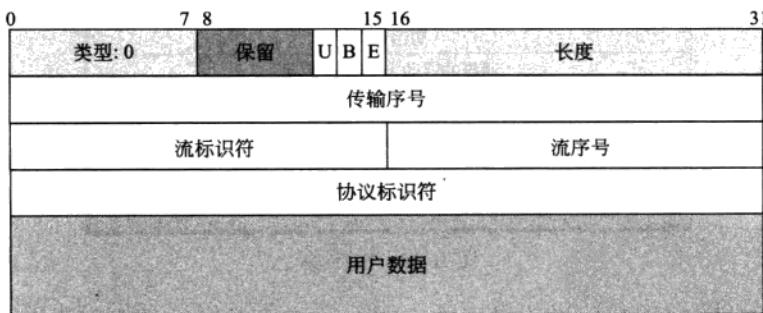


图 16.9 DATA 块

公共字段的描述同上。类型字段的值是 0，标志字段中有 5 位是保留的，只有 3 位有定义：U、B 和 E。当 U (不按序的) 字段置为 1 时，表示这是不按序的数据 (以后解释)。在这种情况下，流序号的值就被忽略。B 位 (开始) 和 E 位 (结束) 组合在一起定义了一个块在分片的报文中的位置。当 B = 1 和 E = 1 时，表示没有分片 (第一和最后)，整个报文只用了一个块。当 B = 1 和 E = 0 时，这就是第一个分片。当 B = 0 和 E = 1 时，这就是最后一个分片。当 B = 0 和 E = 0 时，这就是中间的分片 (既不是第一个，也不是最后一个)。

请注意，数据块的长度字段不包括填充，它的数值不能小于 17，因为一个 DATA 块必须携带至少一个字节的数据。

- **传输序号 (TSN)** 这个 32 位字段定义了传输序号。在一个方向上，序号在 INIT 块中被初始化，而在另一个相反的方向上，序号在 INIT ACK 中被初始化。
- **流标识符 (SI)** 这个 16 位字段定义了关联中的各个流。同一个方向上属于同一个流的所有块携带相同的流标识符。
- **流序号 (SSN)** 这个 16 位序号定义了特定方向上的特定流中的某一个块。
- **协议标识符** 这个由应用程序使用的 32 位字段定义了数据的类型。SCTP 层忽略这个字段。
- **用户数据** 这个字段携带的是真正的用户数据。SCTP 对用户数据字段有一些特殊规定。第一，块中携带的数据必须属于同一个报文，但是一个报文可以分布在多个数据块中。第二，这个字段不能是空的，它必须至少有一个字节的用户数据。第三，如果数据不能在 32 位的边界上结束，就必须进行填充。这些填充的字节不包括在长度字段值的计算中。

DATA 块携带的数据不能属于超过一个报文，但是一个报文可以分割成几个块。DATA 块的数据字段必须携带至少一个字节的数据，这就表示它的长度字段值不能小于 17。

INIT 块

INIT 块（开始块）是从某个端点发送出来的第一个块，用以建立关联。携带这个块的分组不能再携带其他任何控制块或数据块。这个分组验证标志的值是 0，也就是说还没有定义任何标志。图 16.10 给出了它的格式。

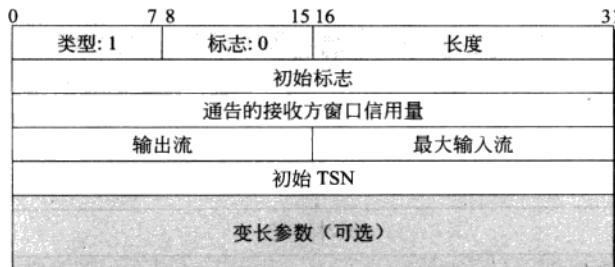


图 16.10 INIT 块

和前面的一样，它也有三个公共字段（类型、标志和长度）。类型字段值是 1，标志字段值是 0（没有标志），而长度字段的最小值是 20（如果有可选参数就会更大些）。其他几个字段解释如下：

- **初始标志** 这个 32 位字段定义了在相反方向上传送的分组的验证标志。正如我们曾经提到的，任何分组在通用首部中都有一个验证标志，这个标志对同一个关联在同一个方向上传送的所有分组来说，都是相同的。这个标志的值是在关联建立时确定的。发起关联的端点在初始标志字段中定义了一个标志值，这个值被用在另一个方向上传送的分组中作为验证标志。例如，若端点 A 发起与端点 B 的关联，A 定义了一个初始标志值，假设为 x ，这个值就被用做所有从 B 到 A 的分组的验证标志。

初始标志是一个从 1 到 $2^{32} - 1$ 之间的随机数。数值 0 表示没有关联，只能在 INIT 块的通用首部中出现。

- **通告的接收方窗口信用量** 这个 32 位字段用于流量控制，它定义了这个 INIT 块的发送方所能够允许的初始数据量（以字节计）。这个值就是 rwnd 值，接收方通过这个值就能知道要发送多少数据。请注意，在 SCTP 中，序号是块序号。
- **输出流** 这个 16 位字段定义了关联的发起者在输出方向建议的流数。另一端可以减少这个数值。
- **最大输入流** 这个 16 位字段定义了关联的发起者在输入方向能够支持的最大的流数。请注意，这是个最大值，另一端不能再将其增大。
- **初始 TSN** 这个 32 位字段对输出方向的传输序号（TSN）进行初始化。请注意，关联中的每个数据块都必须有一个 TSN。这个字段值也是一个小于 2^{32} 的随机数。
- **变长参数** 这些可选参数可以被附加到 INIT 块上，它们定义了发送端的 IP 地址、这个端点可支持的 IP 地址数（多归属）、状态 Cookie 的保存、地址的类型以及对显式拥塞通知（ECN）的支持。

携带 INIT 块的分组不能再携带其他块。

INIT ACK 块

INIT ACK 块（开始确认块）是在关联建立期间发送的第二个块。携带这个块的分组不能再携带其他任何控制块或数据块。这个分组的验证标志的值（在通用首部中）就是在收到的 INIT 块中定义的初始标志的值。图 16.11 给出了它的格式。

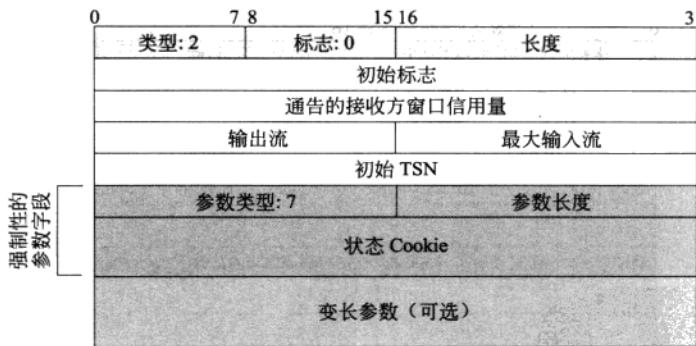


图 16.11 INIT ACK 块

请注意，这个块的主要字段部分和 INIT 块的定义一样。但是，这个块还有一个强制性的参数字段。参数类型 7 定义了这个块的发送者发送的状态 Cookie。在本章的后面我们将讨论 Cookie 的使用。这个块还可以有一些可选的参数。请注意，它的初始标志字段给出的是将来要从反方向传过来的分组的验证标志的值。

携带 INIT ACK 块的分组不能再携带其他的块。

COOKIE ECHO 块

COOKIE ECHO 块是在关联建立期间发送的第三个块。它由收到了 INIT ACK 块的端

点发送（通常就是发送 INIT 块的那一端）。携带这个块的分组可以携带用户数据。图 16.12 给出了它的格式。

请注意，这是一个非常简单的类型为 10 的块。在信息部分，它把上一次收到的 INIT ACK 中的状态 Cookie 回送。INIT ACK 的接收方无法打开这个 Cookie。

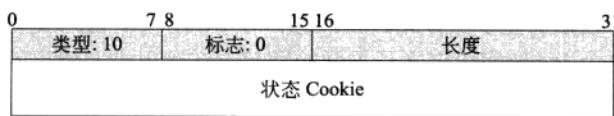


图 16.12 COOKIE ECHO 块格式

COOKIE ACK 块

COOKIE ACK 块是在关联建立期间发送的第四个和最后一个块。它由收到 COOKIE ECHO 块的端点发送。携带这个块的分组也可以携带用户数据。图 16.13 是它的格式。

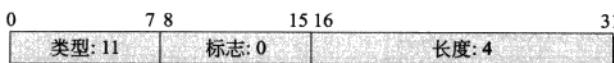


图 16.13 COOKIE ACK 块

请注意，这是一个类型为 11 的非常简单的块。这个块的长度正好是 4 字节。

SACK 块

SACK 块（选择确认块）对收到的数据分组进行确认。图 16.14 给出了 SACK 块的格式。

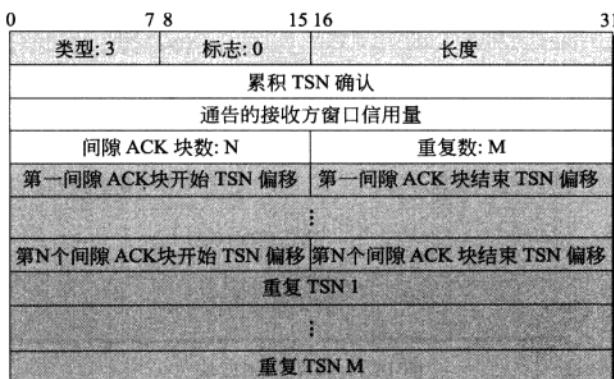


图 16.14 SACK 块

公共字段和前面讨论的一样。类型字段的值为 3。标志位全都置为 0。

- 累积 TSN 确认 这个 32 位字段定义了最后一个按序收到的数据块。
- 通告的接收方窗口信用量 这个 32 位字段是接收方窗口大小的更新值。
- 间隙 ACK 块的数目 这个 16 位字段定义的是在累积 TSN 之后的已收到的数据块间隙的数量。请注意，术语“间隙”在这里的表达不准确。这里的间隙指的是已收到块的序号，而不是丢失的块。

- **重复数** 这个 16 位字段定义了在累积 TSN 之后出现的重复块的数量。
- **间隙 ACK 块开始偏移量** 对每一个间隙块，这个 16 位字段给出相对于累积 TSN 的开始 TSN。
- **间隙 ACK 块结束偏移** 对每一个间隙块，这个 16 位字段给出相对于累积 TSN 的结束 TSN。
- **重复 TSN** 对每一个重复块，这个 32 位字段给出这个块的 TSN。

HEARTBEAT 和 HEARTBEAT ACK 块

HEARTBEAT 块和 HEARTBEAT ACK 块除了类型字段值不一样之外，其他都一样。前者是类型 4，后者是类型 5。图 16.15 给出了这两个块的格式。这两个块用于定期探测关联的状态。某一端发送 HEARTBEAT 块，如果对方处于活跃状态，就响应一个 HEARTBEAT ACK。在这个格式中包括了三个公共字段以及提供发送方特定信息的强制性的参数字段。在 HEARTBEAT 块中，这个信息包括本地时间和发送方的地址。而在 HEARTBEAT ACK 块中，这些信息都将无变化地被复制。

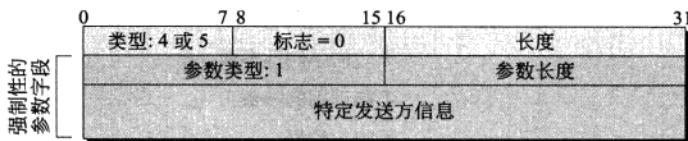


图 16.15 HEARTBEAT 和 HEARTBEAT ACK 块

SHUTDOWN、SHUTDOWN ACK 和 SHUTDOWN COMPLETE 块

这三个块是类似的，用于关闭一个关联。**SHUTDOWN** 块类型是 7，长度是 8 个字节，其中第二个 4 字节定义了累积 TSN。**SHUTDOWN ACK** 块类型是 8，长度是 4 字节。**SHUTDOWN COMPLETE** 块类型是 14，也是 4 字节长，它有一个位标志，即 T 标志。T 标志表示发送方没有 TCB 表（见第 15 章）。图 16.16 给出了它们的格式。



图 16.16 SHUTDOWN、SHUTDOWN ACK 和 SHUTDOWN COMPLETE 块

ERROR 块

当某一端发现收到的分组中有差错，就要发送 ERROR 块。请注意，发送 ERROR 块并不表示要异常终止这个关联（那需要 ABORT 块）。图 16.17 给出了 ERROR 块的格式。

表 16.3 中列出的是对差错的定义。

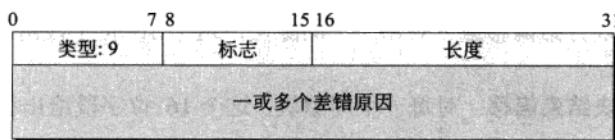


图 16.17 ERROR 块

表 16.3 差错

代码	描述
1	无效的流标识符
2	丢失强制性参数
3	状态 Cookie 差错
4	资源用完
5	不能解析的地址
6	不能识别的块类型
7	无效的强制性参数
8	不能识别的参数
9	无用户数据
10	正当关闭时收到 Cookie

ABORT 块

当某一端发现了致命的差错时，就应当发送 ABORT 块使这个关联异常终止。其中的差错类型和 ERROR 块的一样（见表 16.3）。图 16.18 给出了 ABORT 块的格式。

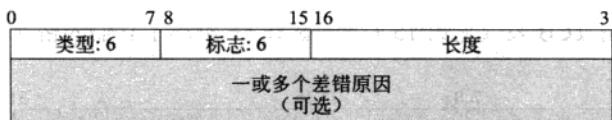


图 16.18 ABORT 块

FORWARD TSN 块

这是最近加入到标准（RFC 3758）中的块，用来通知接收方调整其累积 TSN。它提供部分可靠服务。

16.5 SCTP 关联

像 TCP 一样，SCTP 是面向连接的协议。但是，SCTP 中的连接称为关联，用来强调其多归属（multihoming）特性。

SCTP 中的连接称为关联。

16.5.1 关联建立

在 SCTP 中，关联建立（association establishment）需要四向握手。在这个过程中，一个进程（通常是客户）打算使用 SCTP 作为运输层协议和另一个进程（通常是服务器）建立关联。与 TCP 类似，SCTP 服务器需要准备好接受任何关联（被动打开）。但是，关联建立是由客户发起的（主动打开）。图 16.19 给出了 SCTP 的关联建立。

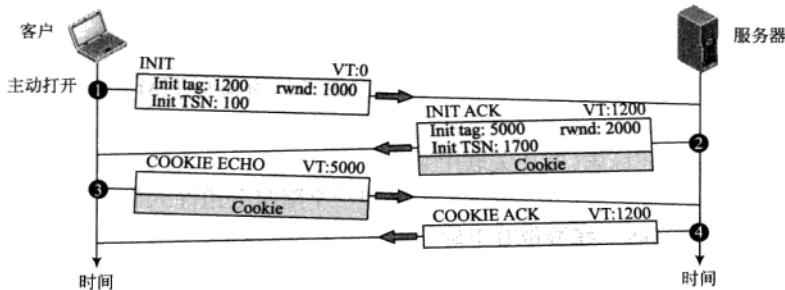


图 16.19 四向握手

正常情况下有如下的步骤：

1. 客户发送第一个分组，它包含一个 INIT 块。这个分组的验证标志（VT）（定义在通用首部中）是 0，因为在这个方向（从客户到服务器）还没有定义验证标志。INIT 块中包含了一个初始标志，用于给另一个方向（服务器到客户）传送的分组使用。这个块还定义了这个方向的 TSN，并通告 rwnd 的值。rwnd 值通常是在一个 SACK 块中通告的，但此处通告 rwnd 是因为 SCTP 允许在第三个和第四个分组中包含 DATA 块，所以服务器必须知道客户可用的缓存大小。请注意，第一个分组中不能发送其他的块。

2. 服务器发送第二个分组，它包含一个 INIT ACK 块。验证标志是 INIT 块中的初始标志字段的值。在这个块中给出了另一个方向使用的验证标志的初始值，同时也定义了从服务器到客户的数据流的初始 TSN，并设置了服务器的 rwnd。这里定义的 rwnd 值是为了允许客户在第三个分组中发送 DATA 块。INIT ACK 块还发送了一个 Cookie，它定义了此刻服务器的状态。稍后我们将会讨论 Cookie 的使用。

3. 客户发送第三个分组，它包含了一个 COOKIE ECHO 块。这是一个非常简单的块，它无变化地回送服务器发送的 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，它包含一个 COOKIE ACK 块，用来对收到的 COOKIE ECHO 块进行确认。SCTP 允许在这个分组中包含数据块。

携带 INIT 或 INIT ACK 块的分组不允许再有其他的块。

携带 COOKIE ECHO 或 COOKIE ACK 块的分组可以携带数据块。

交换分组的数量

在 TCP 的连接建立阶段交换的分组数量是三个，而在 SCTP 关联建立时一共交换了四个分组。看起来好像 SCTP 的效率没有 TCP 的高，但我们要考虑到，SCTP 允许在第三个

分组和第四个分组中交换数据。另外，如我们在后面将会看到的，它还在防止 SYN 拒绝服务的攻击方面提供了更好的安全性。数据在交换了两个分组之后就可以传送了。

验证标志

当我们比较 TCP 和 SCTP 时，我们发现 SCTP 中的验证标志在 TCP 中并不存在。在 TCP 中，一个连接是用 IP 地址和端口号的组合来标志的，而它们则是每个报文段中的一个部分。这样做会产生两个问题：

1. 一个盲目的攻击者（不是截获者）可以像我们在 SYN 洪泛攻击中所讨论的那样，使用随机选取的源端口地址和目的端口地址向一个 TCP 服务器发送报文段。

2. 前一个连接中的迟延的报文段可能在新连接中出现，此报文段使用了相同的源端口地址和目的端口地址（化身）。这也是 TCP 在终止连接时需要有 TIME-WAIT 计时器的一个原因。

SCTP 通过使用验证标志（即在关联中某个方向上传送的所有分组都要携带的一个公共值）解决了这两个问题。盲目的攻击者无法把一个随机分组注入到某一个关联中，因为这个注入的分组不太可能正好携带着正确的标志（概率只有 2^{32} 分之一）。旧的关联中的分组不会出现在化身中，因为即使源端口地址和目的端口地址都相同，其验证标志肯定会不一样。两个验证标志（每个方向一个）标记了一个关联。

Cookie

我们在第 15 章讨论过 SYN 洪泛攻击。在 TCP 中，恶意的攻击者用伪造的 IP 地址，使用大量假的 SYN 报文段，向一个 TCP 服务器进行洪泛攻击。这个服务器每收到一个 SYN 报文段，就要建立一个 TCB 并分配其他的资源，以等待下一个报文段的到来。但不久以后服务器就会因资源耗尽而崩溃。

SCTP 的设计者使用了一种策略来防止这种类型的攻击。这个策略就是延迟分配资源，直至收到第三个分组，因为此时发送方的 IP 地址已被验证。收到的第一个分组中的信息必须用某种办法存储下来，直至第三个分组到达。但是，如果服务器要保存这个信息，它就必须分配资源（存储器），这真是件矛盾的事。解决的方法是把信息压缩一下，再把它发回给客户，这就称为“产生了一个 Cookie”。这个 Cookie 随第二个分组发送到第一个分组中的那个地址。可能出现两种情况：

1. 如果第一个分组的发送者是个攻击者，那么服务器就永远不会收到第三个分组，Cookie 就此丢失，而没有任何资源被分配。服务器付出的代价仅仅是“烘烤”出了这个 Cookie。

2. 如果第一个分组的发送者是需要建立连接的真正客户，它就会收到带有 Cookie 的第二个分组。客户发送一个分组（这一组中的第三个），附上原封不动的 Cookie。服务器收到第三个分组，知道这个分组来自一个真正的客户，因为它之前发送出去的 Cookie 就在这个分组中。服务器现在就可以分配资源了。

只要没有哪个实体能够“吃掉”发送方“烘烤”的 Cookie，上面的策略就可以正常工作。要保证这一点，服务器用它自己的秘密密钥从这个信息中创建一个摘要（见第 29 章）。信息和摘要一起被制作成 Cookie，并在第二个分组中发送给客户。当这个 Cookie 在第三个分组中被返回时，服务器要根据信息计算出摘要。若这个计算的摘要与发送的摘要匹配，就表明 Cookie 没有被任何其他实体更改过。

16.5.2 数据传送

关联的作用就是要在两个端点之间传送数据。在关联建立之后，双向的数据传送就可以开始了。客户和服务器都可以传送数据。和 TCP 一样，SCTP 也支持捎带技术。

但是 TCP 的数据传送和 SCTP 的数据传送之间有一个重要的区别。TCP 把从一个进程那里接收到的报文当作是一个字节流，不会去辨认这些字节之间是否存在任何边界。发送进程可能会设置一些边界以利于对等进程的使用，但 TCP 把这些标记当作是正文的一部分。换言之，TCP 收取每一个报文，然后把它追加到自己的缓存中。一个报文段可以携带来自两个不同报文的部分内容。TCP 使用的唯一的定位系统就是字节号。

与此不同的是，SCTP 能够识别和维护边界。来自进程的每一个报文被当作是一个独立的单元，并被插入到一个 DATA 块中，除非它被分片了（在后面讨论）。从这个意义上讲，SCTP 有点类似于 UDP，但是它还具有一个很大的优点：数据块相互之间是有联系的。

来自进程的报文被接收后，通过在这个报文上添加一个 DATA 块首部后，就形成一个 DATA 块（如果被分片的话就是几个 DATA 块）。由一个报文或报文的一个分片所形成的每一个 DATA 块都具有一个 TSN。我们需要记住，只有 DATA 块需要使用 TSN，也只有 DATA 块会被 SACK 块确认。

在 SCTP 中，只有 DATA 块才消耗 TSN；

DATA 块是唯一能够被确认的块。

让我们通过图 16.20 描绘一个简单的情况。图中，客户发送了四个 DATA 块，并收到了两个来自服务器的 DATA 块。稍后我们将会更加详细地讨论在 SCTP 中使用的流量控制和差错控制。就目前而言，我们先假定此情况下一切工作正常。客户使用了验证标志 85，服务器使用的验证标志是 700。双方发送的分组描述如下：

1. 客户发送第一个分组，携带了两个 DATA 块，其 TSN 分别为 7105 和 7106。
2. 客户发送第二个分组，携带了两个 DATA 块，其 TSN 分别为 7107 和 7108。
3. 第三个分组来自服务器。它包含了一个 SACK 块，用于确认收到来自客户的 DATA 块。SCTP 对最后一个按序收到的 TSN 进行确认，而不是指下一个期望接收的。第三个分组中还包含了来自服务器的第一个 DATA 块，其 TSN 为 121。
4. 过了一会，服务器又发送了另一个分组，携带最后一个 DATA 块，其 TSN 为 122，但在这个分组中并没有包含任何 SACK 块，因为来自客户的上一个 DATA 块已经确认过了。
5. 最后，客户发送包含一个 SACK 块的分组，确认收到了从服务器发送过来的两个 DATA 块。

SCTP 中的确认给出了累积 TSN，即最后一个按序收到的数据块中的 TSN。

多归属数据传送

我们讨论过 SCTP 的多归属能力，这是 SCTP 有别于 UDP 和 TCP 的一个特点。多归属允许关联的两端可定义多个 IP 地址用于通信。但是，这些地址中只有一个地址被定义为主地址（primary address），而其他的都是备选地址。主地址在关联建立时被定义。一个有趣的地方是：一端的主地址是由另一端来决定的。换言之，源点定义了终点的主地址。

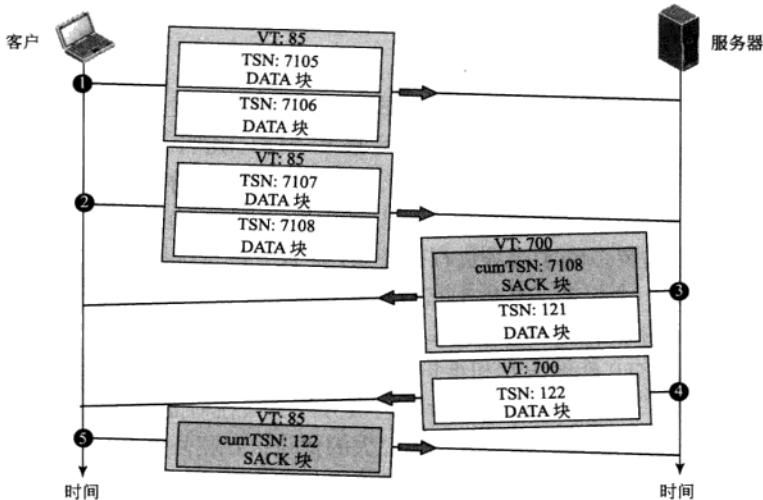


图 16.20 简单的数据传送

在默认的情况下，数据传送使用终点的主地址。如果主地址不可用，那么就改用一个备选地址。但是进程随时可以替换掉主地址，明确地请求把一个报文发送到某个特定的备选地址。进程还可以显式地更换当前关联的主地址。

于是这就出现了一个逻辑问题：SACK 要发送给谁。SCTP 指出 SACK 要发送给产生相应的 SCTP 分组的那个地址。

多重流交付

SCTP 的一个有意思的特点就是数据传送和数据交付是区别对待的。SCTP 使用 TSN 编号处理数据的传送，也就是数据块在源点和终点之间的移动。而数据块的交付则是由 SI 和 SSN 控制的。SCTP 支持多重流，也就是说发送进程可以定义多条不同的流，而一个报文可以属于这些流之一。每个流都被指派了一个流标识符（SI），它唯一地定义某一个流。但是，SCTP 在每个流中都可以支持两种类型的数据交付：**按序的（默认的）**和**不按序的**。对于按序的数据交付，流中的数据块通过流序号（SSN）来定义它们在该流中的顺序。当这些块到达终点时，SCTP 负责根据定义在块中的 SSN 交付报文。这种交付可能会被延迟，因为某些块可能没有按序到达。对于不按序的数据交付，流中的数据块把 U 标志置为 1，而它们的 SSN 字段值则被忽略。这些数据块不消耗 SSN。当一个不按序的数据块到达终点 SCTP 时，它就把这个块中所携带的报文交付给应用程序，而不用等待其他的报文。在大多数情况下，应用程序使用按序交付服务，但偶尔也有某些应用程序需要发送紧急数据，这些紧急数据不需要按序交付（回忆 TCP 的紧急数据和紧急指针）。在这些情况下，应用程序可以将交付定义成不按序的。

分片

在数据传送中存在的另一个问题就是分片（fragmentation）。虽然 SCTP 分享了 IP 中的这个术语，但 IP 的分片和 SCTP 的分片是属于不同层次的：前者在网络层，而后者在运输层。

SCTP 从一个报文产生 DATA 块时，如果报文长度（当被封装在 IP 数据报中时）不超

过路径的 MTU，那么从进程到进程都一直会保留该报文的边界。携带一个报文的 IP 数据报的长度计算可以通过在该报文的长度（以字节为单位）上增加 4 个额外开销：数据块首部、必须的 SACK 块、SCTP 通用首部和 IP 首部。如果总长度超过 MTU，那么报文就必须进行分片。

分片在源点 SCTP 处进行，采用以下的步骤：

1. 报文划分为更小的分片，以满足长度的需求。

2. 每一个分片必须附加具有不同 TSN 的 DATA 块的首部。这些 TSN 必须按序排列。

3. 所有的头部携带相同的流标识符 (SI)、相同的流序号 (SSN)、相同的有效载荷协议标识符以及相同的 U 标志。

4. 指派的 B 和 E 标志的组合如下：

- a. 第一个分片：1 0

- b. 中间的分片：0 0

- c. 最后的分片：0 1

分片在终点重装。如果一个 DATA 块到达时它的 B/E 位等于 11，就表示它没有被分片。接收方知道怎样把所有具有相同 SI 和 SSN 的块重装起来。分片的数目由第一个分片和最后一个分片的 TSN 号决定。

16.5.3 关联终止

和 TCP 一样，SCTP 中参与数据交换的任何一方（客户或服务器）都可以关闭这个连接。但是与 TCP 不同的是，SCTP 不允许“半关闭”一个关联。如果某一端关闭了这个关联，那么另一端必须停止发送新的数据。如果收到终止请求的一端在队列中还有未发送的数据，那么先把它发送出去，然后再关闭关联。关联终止使用了三个分组，如图 16.21 所示。请注意，虽然在图中给出的是由客户发起的终止，服务器同样也可以发起终止。

关联终止可以有多种情况。我们将在以后再讨论其中的一些情况。

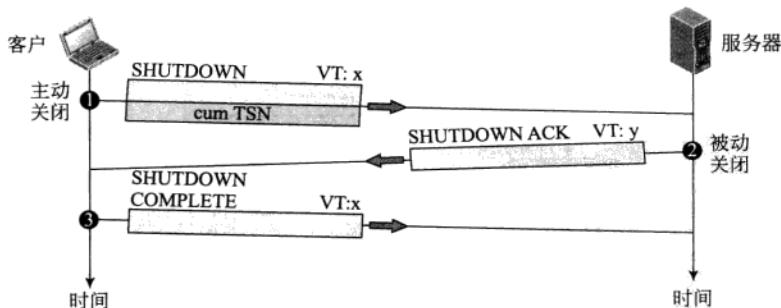


图 16.21 关联终止

16.5.4 关联异常终止

前一节所讨论的关联终止有时称为“从容终止”(graceful termination)。在 SCTP 中的关联也可以异常终止。异常终止可以由任一端的进程发出请求，也可由 SCTP 发出请求。

如果一个进程觉得自身有些问题(如从另一端收到错误的数据,进入了无限循环,等等),就可以使这个关联异常终止。服务器也可以异常终止这个关联,如果它收到了具有错误参数的 INIT 块,所请求的资源不可用(在收到了 Cookie 后),或操作系统需要关闭,等等。

SCTP 的异常终止过程非常简单。任一端都可以发送 ABORT 块来异常终止这个关联,如图 16.22 所示。不需要其他更多的块。

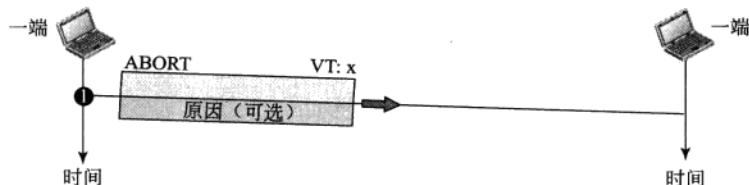


图 16.22 关联异常终止

16.6 状态转换图

为了跟踪在关联建立、关联终止和数据传送时所发生的各种不同事件,像 TCP一样,SCTP 软件也是以有限状态机的方式实现的。图 16.23 给出了客户端和服务端的状态转换图。

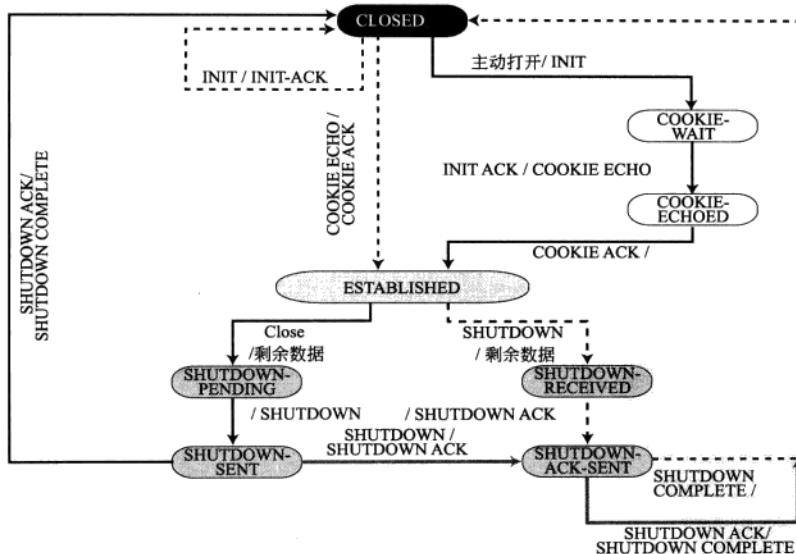


图 16.23 状态转换图

图中的虚线表示服务器的正常转换,实线表示客户的正常转换,粗线表示非正常的情况(非正常的情况下有“非正常转换”的字样)。在某些特殊情况下,服务器也可以按照实线转换,客户也可以按照虚线转换。表 16.4 给出了 SCTP 的一些状态。

表 16.4 SCTP 的状态

状态	描述
CLOSED	没有连接
COOKIE-WAIT	等待 Cookie
COOKIE-ECHOED	等待 Cookie 确认
ESTABLISHED	连接已经建立；数据正在传送
SHUTDOWN-PENDING	收到关闭命令后正在发送数据
SHUTDOWN-SENT	等待 SHUTDOWN 确认
SHUTDOWN-RECEIVED	收到 SHUTDOWN 后正在发送数据
SHUTDOWN-ACK-SENT	等待终止的完成

16.6.1 几种情况

为了了解 SCTP 状态机和转换图，我们在本节描述了一些情况。

一般情况

图 16.24 给出了一种典型的情况。这是一种正常的工作状态，打开命令和关闭命令都来自客户。这里我们给出的状态和以前描述相应的 TCP 情况时一样。图中具体描绘了关联建立（前 4 个分组）和关联终止（最后 3 个分组），以及在此期间客户和服务器所经历的状态。请注意，在关联建立期间，服务器保持在 **CLOSED** 状态，而客户进程则经过了两个状态 (**COOKIE-WAIT** 和 **COOKIE-ECHOED**)。

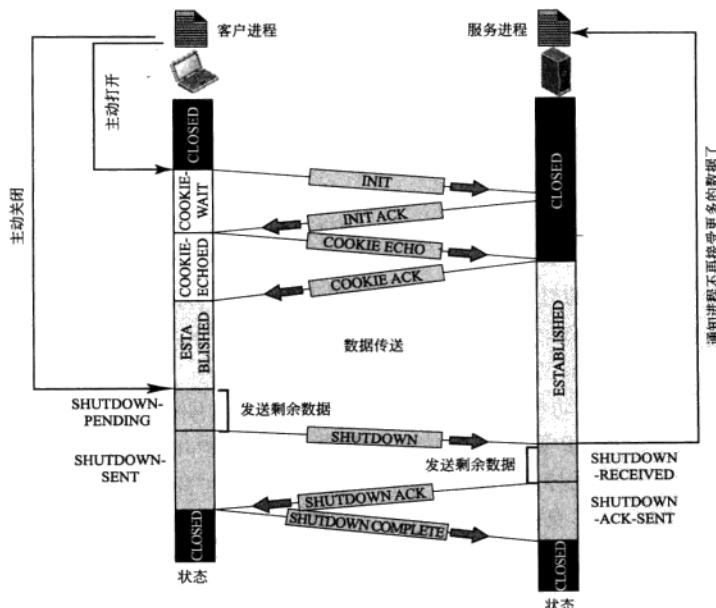


图 16.24 一般情况中的状态

当客户 SCTP 收到来自进程的主动关闭命令后，就进入 **SHUTDOWN-PENDING** 状态。在所有剩余数据都发送出去之前，客户一直处在这个状态，然后它发送 SHUTDOWN 块，并进入 **SHUTDOWN-SENT** 状态。服务器在收到 SHUTDOWN 块后，就通知其进程不再接受新的数据了。服务器进入 **SHUTDOWN-RECEIVED** 状态。在这个状态中，服务器把所有剩余数据发送给客户，然后发送 SHUTDOWN-ACK 块，接着就进入 **SHUTDOWN-ACK-SENT** 状态。在收到最后一个块后，客户发送 SHUTDOWN-COMPLETE 块，关联关闭。服务器在收到最后一个块后，也关闭了这个关联。

同时打开

在关联建立时，由于 Cookie 的存在而带来了一些复杂性。当某一端收到 INIT 块并发送 INIT-ACK 块时，它仍然处于 **CLOSED** 状态，它不会记得收到了什么和发送了什么。因此，如果某一端的进程发出了主动打开命令，而在此关联真正建立之前，另一端的进程也发出了主动打开命令，那么就会出现问题。图 16.25 给出了这种情况。

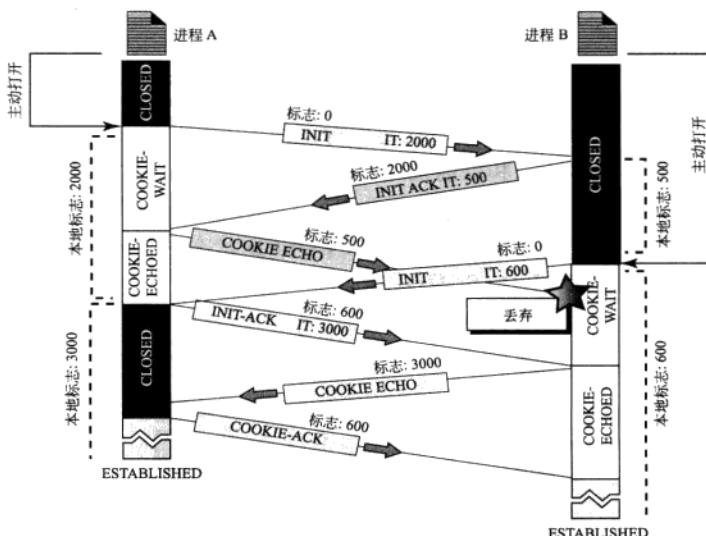


图 16.25 同时打开

站点 A 的进程发出主动打开命令，因而发送了 INIT 块。站点 B 收到这个 INIT 块，并发送了 INIT-ACK 块。而在站点 B 收到来自站点 A 的 COOKIE-ECHO 块之前，站点 B 的进程也发出了主动打开命令。站点 B 的 SCTP 不记得已经有一个关联在启动中，于是它发送一个 INIT 块给站点 A 启动一个新的关联。这两个关联发生碰撞。

为解决这个问题，SCTP 要求每个站点在发送 INIT 或 INIT-ACK 时，还要发送一个初始标志。这个标志称为本地标志，被保存在一个变量中。在任何时候，这个变量中只保存一个本地标志。每当一个具有验证标志的分组到达时，如果它的验证标志与本地标志不匹配，那么这个分组就要被丢弃。SCTP 还可以在关联建立的过程中开始一个新的关联，此时一个新的 INIT 块到达。

在我们假设的情况下给出了这些标志的使用。开始时，站点 A 发送了一个初始标志为

2000 的 INIT 块，现在它的本地标志变量的值是 2000。站点 B 发送了一个初始标志为 500 的 INIT-ACK 块，此时站点 B 的本地标志变量的值是 500。当站点 B 又发送了一个 INIT 块时，初始标志和本地标志都变为 600。站点 B 会丢弃收到的 COOKIE-ECHO，因为该分组中的标志是 500，它与本地标志 600 不匹配。未完成的第一个关联就此异常终止。但是，以站点 B 作为发起者的新的关联将继续下去。

同时关闭

两个端点可以同时关闭关联。这种情况发生在：当客户的 SHUTDOWN 块到达服务器时，服务器也已经发出了一个 SHUTDOWN 块。此时，客户和服务器双方各进入不同的状态来关闭这个关联，如图 16.26 所示。

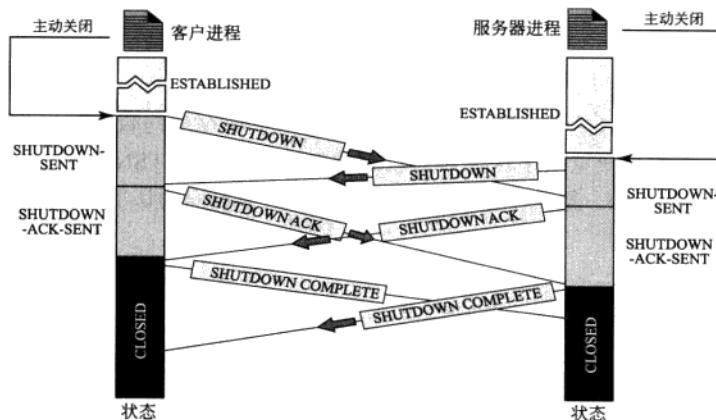


图 16.26 同时关闭

图中描绘的是客户和服务器都发出了主动关闭。我们假定两个 SCTP 都没有剩余数据要发送，所以 **SHUTDOWN PENDING** 状态就跳过去了。两个 SCTP 在收到 SHUTDOWN 块后，都进入 **SHUTDOWN-SENT** 状态，然后双方都发送 SHUTDOWN-ACK 块，并进入 **SHUTDOWN-ACK-SENT** 状态。双方保持在这个状态，直至它们收到从另一方发来的 SHUTDOWN ACK 块。此后它们发送 SHUTDOWN COMPLETE 块，并进入关闭状态。请注意，当收到最后一个块时，双方的 SCTP 都已经处在 **CLOSED** 状态。如果我们观察图 16.23 的状态转换图就可以看出，客户和服务器在进入 **ESTABLISHED** 状态后都沿着相同的路径变化，即客户所采取路径。但是，在进入 **SHUTDOWN-SENT** 状态后，双方向右转，而不是向左。当双方都到达 **SHUTDOWN-ACK-SENT** 状态时，它们就向下变化，而不是一直向前。

16.6.2 其他情况

还有许多情况，但在本书中没有地方，也没有时间容许我们继续讨论下去。我们还应当有关于计时器的信息，以及收到意外的块的处理过程。我们可能还需要 SCTP 的附加信息，包括相应的 RFC 文档。我们把这些情况留作习题或研究活动。

16.7 流量控制

SCTP 中的流量控制和 TCP 的相似。在 TCP 中，我们只需要面对一种数据单位，即字节。而在 SCTP 中，我们需要面对两种数据单位：字节和块。`rwnd` 和 `cwnd` 的值是用字节表示的，而 `TSN` 和确认则用块来表示。为了说明这个概念，我们做出一些不切实际的假定。我们假定网络中从不发生拥塞，网络也是无差错的。换言之，我们假定 `cwnd` 为无穷大，同时没有分组丢失、迟延或不按序到达。我们还假定数据传送是单向的。在后面的小节中，我们将会去掉这些不实际的假定。目前的 SCTP 实现中，流量控制仍然使用面向字节的窗口。但是在表示缓存时是以块为单位，这样可以使这个概念更容易理解。

16.7.1 接收方

接收方有一个缓存（队列）和三个变量。队列用于保存那些收到的、但还没有被进程读取的数据块。第一个变量用于保存最后收到的 `TSN`，即 `cumTSN`。第二个变量用于保存可用缓存的大小 `winSize`。第三个变量用于保存最后一个累积确认 `lastACK`。图 16.27 给出了接收方的队列和变量。

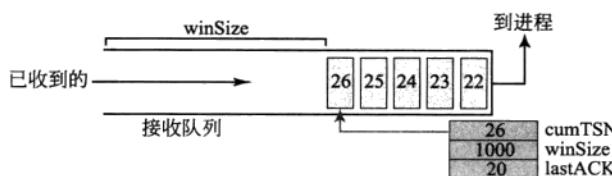


图 16.27 接收方的流量控制

- 当站点收到一个数据块时，就把它保存到缓存（队列）的末尾，并从 `winSize` 中减去这个块的长度。这个块的 `TSN` 号存放在变量 `cumTSN` 中。
- 当进程读取一个块时，就从队列的前端取出一个块，并把这个块的长度加到 `winSize` 中（重复利用）。
- 当接收方决定发送 SACK 时，它检查 `lastACK` 值，如果这个值小于 `cumTSN`，就发送一个 SACK，其累积 `TSN` 号等于 `cumTSN`。在这个 SACK 中还包括了 `winSize` 的值作为通告的窗口大小。然后更新 `lastACK` 的值以保存 `cumTSN` 的值。

16.7.2 发送方

发送方有一个缓存（队列）和三个变量：`curTSN`、`rwnd` 和 `inTransit`，如图 16.28 所示。我们假定每一个块有 100 字节长。

在缓存中保存的是由进程产生的块，这些块或者已经发送出去了，或者是准备将要发送的。第一个变量是 `curTSN`，指向下一个要发送的块。在队列中所有 `TSN` 小于这个数值的块都是已经发送出去，但还没有被确认的，即它们是待确认的。第二个变量是 `rwnd`，用

于保存由接收方通告的最近一个数值（以字节计）。第三个变量是 *inTransit*，保存正在传送中的字节数，即已经发送出去但还没有被确认的字节数。下面是发送方采用的过程：

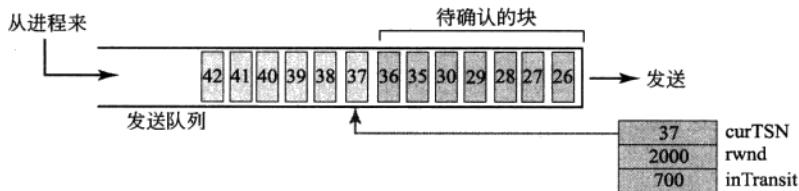


图 16.28 发送方的流量控制

1. 由 *curTSN* 指向的块可以被发送，只要它的数据长度小于或等于 (*rwnd* - *inTransit*) 的值。发送完这个块以后，*curTSN* 值就加 1，以指向下一个要发送的块。*inTransit* 值要加上已经发送的块的长度。

2. 当收到一个 SACK 时，队列中的 TSN 小于或等于 SACK 的累积 TSN 的块就被清除，发送方不用再为它们操心了。*inTransit* 的值减去被丢弃的块的总长度。*rwnd* 的值要更新到 SACK 中通告的窗口大小。

16.7.3 一种情况

让我们描述如图 16.29 所示的一种简单的情况。在开始时，发送方的 *rwnd* 值和接收方的 *winSize* 值都是 2000（在关联建立时通告的）。最初，在发送方的队列中有四个报文。发送方发送了一个数据块，把字节数（1000）加到 *inTransit* 变量中。过了一会，发送方检查 *rwnd* 和 *inTransit* 之差（是 1000 字节），所以又发送了另一个数据块。现在这两个变量之差为 0，因此不能再发送任何数据块了。又过了一会，一个 SACK 到达了，它确认了块 1 和块 2。这两个块就从队列中被清除。*inTransit* 值现在为 0。但是由于这个 SACK 通告的接收窗口值是 0，这就使得发送方把 *rwnd* 更新为 0。现在发送方被阻塞，它不能再发送任何数据块。

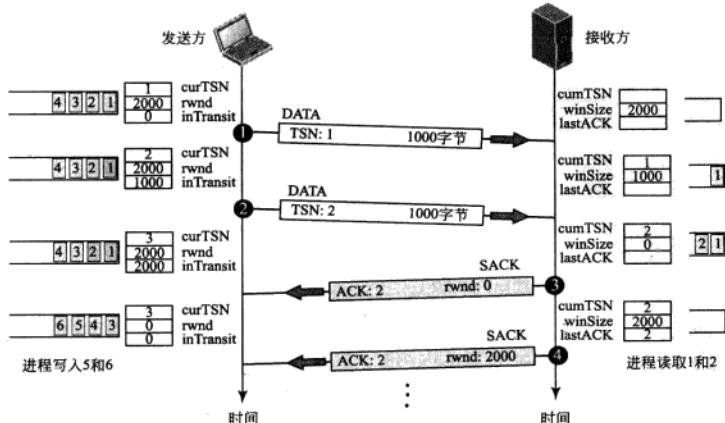


图 16.29 流量控制情况

在接收方，一开始时队列是空的。当收到第一个数据块后，队列中就有一个报文，cumTSN 的数值是 1。winSize 的值减少到 1000，因为第一个报文占据了 1000 字节。当收到第二个数据块后，winSize 值是 0，而 cumTSN 的数值是 2。现在，如我们在后面将会了解到的，接收方必须发送一个累积 TSN 值为 2 的 SACK。在发送了这第一个 SACK 后，进程读取了两个报文，也就是说队列中又有空间了，接收方再次用一个 SACK 通告这种情况，允许发送方继续发送数据块。剩下的事件没有在图中表示出来。

16.8 差错控制

和 TCP一样，SCTP是一个可靠的运输层协议。它使用 SACK 块向发送方报告接收方缓存的状态。在 SCTP的各种实现中，接收方和发送方使用了不同的实体和计时器的集合。而我们使用一个非常简单的设计把这个概念传递给读者。

16.8.1 接收方

在我们的设计中，接收方把所有到达的块（包括不按序的）都存储在它的队列中。但是，它会为丢失的块留下空间。它丢弃重复的报文，但会记录它们，以便向发送方报告。图 16.30 给出了一种典型的针对接收方的设计，以及在某个特定时刻接收方队列的状态。

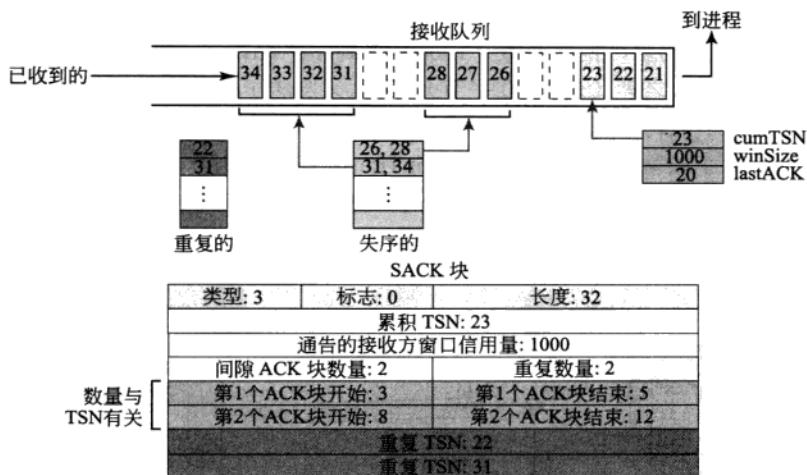


图 16.30 接收方的差错控制

发送出去的最后一个确认是对数据块 20 的确认，可用的窗口大小是 1000 字节，块 21~23 已经按序收到。第一个失序到达的块包含数据块 26~28。第二个失序到达的块包含数据块 31~34。一个变量保留 cumTSN 值。一个变量数组记录了每一个失序到达块的开始和结束。另外一个变量数组记录了收到的重复块。请注意，在队列中没有必要存储重复块，它们都被丢弃。这个图中还显示出了将要发送的 SACK 块，它把接收方状态报告给发

送方。其中失序块的 TSN 号是相对于累积 TSN 的值（即偏移）。

16.8.2 发送方

在发送方，我们的设计需要用到两个缓存（队列）：一个发送队列，一个重传队列。我们仍然使用三个变量：rwnd、inTransit 和 curTSN，如前一节所述。图 16.31 给出了一种典型设计。

发送队列中保存着块 23~40。块 23~36 已经发送出去了，但还没有被确认，它们是待确认的块。curTSN 指向下一个要发送的块（37）。我们假定每一个块是 100 字节，这表示共有 1400 字节的数据（块 23~36）在传送过程中。此时的发送方有一个重传队列。当一个分组被发送出去时，就会为这个分组启动一个相应的重传计时器（即对应于这个分组中的所有数据块）。有的实现为整个关联使用一个计时器，但为了简单起见还是继续为每一个分组设置一个计时器的传统。当对应于某个分组的重传计时器超时，或声明某个分组丢失的三个重复的 SACK 到达时（第 15 章讨论了快重传），这个分组中的数据块就被转移到重传队列中进行重传。这些数据块被认为丢失了，而不是待确认的。重传队列中的数据块有优先权。换言之，发送方在下一次发送数据块时，被发送的数据块就是重传队列中的块 21。

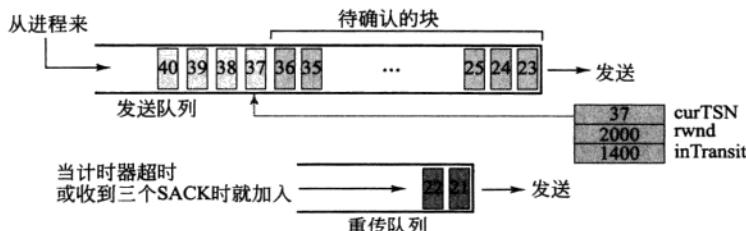


图 16.31 发送方的差错控制

要了解发送方的状态是怎样改变的，假定在图 16.30 中的 SACK 到达了图 16.31 中所示的发送方。图 16.32 给出了发送方的新状态。

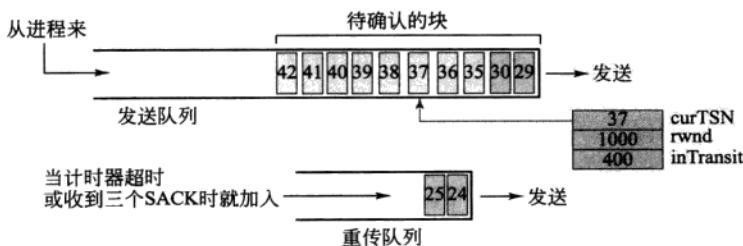


图 16.32 接收到 SACK 后发送方的新状态

1. 所有的块，如果其 TSN 等于或小于 SACK 中的 cumTSN，就从发送队列或重传队列中清除。它们不再是待确认的或被标记为重传的。块 21 和 22 从重传队列中清除。块 23

从发送队列中清除。

2. 我们的设计还要从发送队列中清除所有在间隙块中声明的块，不过，某些比较保守的实现会保留这些块，直至包含它们的一个 cumTSN 到达。这种防护措施在某些不常见的情况下是需要的，可能接收方还会在这些失序块中发现一些问题。我们忽略这种罕见的情况。因此，块 6~28 以及块 31~34 就从发送队列中清除。

3. 重复块的清单没有任何影响。

4. 如同在 SACK 中通告的，rwnd 值改变为 1000。

5. 我们还假定携带数据块 24 和 25 的分组的重传计时器超时了，它的两个数据块被转移到重传队列中，并按照在第 15 章中讨论的指数退避规则设置新的重传计时器。

6. 变量 inTransit 值变为 400，因为现在只有 4 个块正在传输。在重传队列中的块没有计入，因为它们假定丢失了，而不是正在传输。

16.8.3 发送数据块

任一端都可以发送数据分组，只要在发送队列中有 TSN 大于或等于 curTSN 的数据块，或者在重传队列中有数据块。重传队列具有优先权。但是，这个分组中的一个或多个数据块的总长度一定不能超过 (rwnd - inTransit)，并且帧的总长度一定不能超过 MTU 值（如我们在前面的小节中所讨论的）。如果假定，在前面描述的情况下，我们的分组只能携带 3 个块（由于 MTU 的限制），那么重传队列中的块 24 和 25，和发送队列中的下一个要发送的块 37 是可以发送的。请注意，在发送队列中的待确认的块不能发送，因为假设它们已经在传输中。还要注意，任何从重传队列中发送的块，同样也要设置重传计时器。新的计时器影响到块 24, 25 和 37。这里我们要指出，某些实现不允许把重传队列和发送队列中的块混合起来发送。在这种情况下，只有块 24 和 25 可以在这个分组中被发送。

重传

与 TCP 一样，为了控制丢失的或丢弃的块，SCTP 采用了两种策略：使用重传计时器以及收到对相同丢失块的 4 个 SACK。

重传计时器 SCTP 利用重传计时器来处理重传时间，即等待分组被确认的时间。在 SCTP 中计算 RTO 和 RTT 的过程和我们曾讨论过的 TCP 的一样。SCTP 使用测量的 RTT (RTT_M)、平滑的 RTT (RTT_S) 和 RTT 偏差 (RTT_D) 来计算 RTO。SCTP 还使用 Karn 算法来避免确认的二义性。请注意，如果一个主机使用多个 IP 地址（多归属），那么必须对每条路径分别计算 RTO。

四个丢失分组的报告 每当发送方连续收到四个重复的 SACK，且它们的间隙信息都指出一个或多个特定的块丢失了时，发送方就需要把这些块视为丢失的，并立即把它们转移到重传队列中。这个行为类似于 TCP 中的“快重传”（在第 15 章中讨论）。

16.8.4 生成 SACK 块

在差错控制中的另一个问题就是 SACK 块的生成。SCTP 生成 SACK 块的规则和 TCP 使用 ACK 标志作为确认是相似的。我们把这些规则归纳如下。

1. 当一端向另一端发送 DATA 块时, 它必须包含一个 SACK 块, 用来通告已收到的未被确认的 DATA 块。
2. 当一端收到包含数据的分组时, 如果它没有数据要发送, 就必须在规定的时间内(通常是 500 ms) 对这个收到的这个分组进行确认。
3. 当一端每收到两个连续的分组时, 必须至少发送一个 SACK。这个规则高于第二个规则。
4. 当一个具有失序 DATA 块的分组到达时, 接收方必须立即发送 SACK 块, 把这个情况报告给发送方。
5. 当一端收到一个具有重复 DATA 块的分组, 且其中没有新的 DATA 块时, 这个重复的数据块必须立即用 SACK 块报告。

16.9 拥塞控制

与 TCP 一样, 运输层协议 SCTP 所传送的分组可能会在网络中遭遇拥塞。SCTP 的设计者采用了在第 15 章讨论 TCP 时描述的相同策略。SCTP 有慢开始(指数增大)、拥塞避免(加法增大)和拥塞检测(乘法减小)几个阶段。像 TCP 那样, SCTP 也使用快重传和快恢复。

16.9.1 拥塞控制和多归属

SCTP 中的拥塞控制更加复杂一些, 因为一台主机可以有多个 IP 地址。在这种情况下, 网络中的数据所通过的路径可以超过一条。这些路径中的每一条都可能遭遇不同程度的拥塞。这就表示站点需要为每个 IP 地址设置不同的 cwnd 值。

16.9.2 显式拥塞通知

如同为其他广域网定义的那样, 显式拥塞通知(ECN)是使接收方能够明确地通知发送方有关在网络中受到了任何拥塞的一个过程。如果接收方发现有很多延迟的或丢失的分组, 这就是可能发生了拥塞的一种指示。SCTP 可以在 INIT 和 INIT ACK 中使用一个 ECN 选项, 使双方协商 ECN 的使用。如果双方都同意, 接收方就可以通知发送方发生了拥塞, 办法是在每个分组中发送一个 ECNE(显式拥塞通知回送)块, 直至它收到一个 CWR(拥塞窗口减小)块, 表明发送方已经减小了它的 cwnd 值。我们没有讨论过这两个块, 因为它们还没有成为标准的一部分, 同时也由于显式拥塞通知的讨论已经超过了本书的范围。

16.10 深入阅读

要更细致地了解本章所讨论的内容, 我们推荐以下书籍和 RFC。用方括号括起来的书

目可以在本书末尾的参考书目清单中找到。

16.10.1 参考书

我们极力推荐[Ste & Xie 01]，这本书是专门介绍 SCTP 协议的。

16.10.2 RFC

有多份 RFC 讨论了 SCTP 协议，包括 RFC 4820、RFC 4895、RFC 4960、RFC 5043、RFC 5061 和 RFC 5062。

16.11 重要术语

ABORT 块	初始标志
关联	面向报文
关联建立	多归属服务
面向字节	多重流服务
CLOSED 状态	按序交付
Cookie	主地址
COOKIE ACK 块	SACK 块
COOKIE ECHO 块	SHUTDOWN ACK 块
COOKIE-ECHOED 状态	SHUTDOWN 块
COOKIE-WAIT 状态	SHUTDOWN COMPLETE 块
DATA 块	SHUTDOWN-ACK-SENT 状态
ERROR 块	SHUTDOWN-PENDING 状态
ESTABLISHED 状态	SHUTDOWN-RECEIVED 状态
分片	SHUTDOWN-SENT 状态
HEARTBEAT ACK 块	流标识符 (SI)
HEARTBEAT 块	流序号 (SSN)
INIT ACK 块	传输序号 (TSN)
INIT 块	不按序交付
初始 TSN	验证标志

16.12 本章小结

- SCTP 是面向报文的可靠的协议，它结合了 UDP 和 TCP 的优点。SCTP 还提供了 UDP 或 TCP 没有提供的更多服务，例如多重流服务和多归属服务。SCTP 是面向连

接的协议。SCTP 的连接称为关联。SCTP 提供流量控制、差错控制和拥塞控制。

- SCTP 使用术语分组来定义运输单元。在 SCTP 中，控制信息和数据信息分别在相互独立的块中携带。
- 为了区分不同的流，SCTP 使用流标识符 (SI)。为了区分属于同一个流中的不同数据块，SCTP 使用流序号 (SSN)。一个数据块由三个标识符来标志：TSN、SI 和 SSN。
- SCTP 的确认号仅仅用来确认数据块。控制块不使用确认，如有必要则使用另一个控制块。
- SCTP 在状态转换图中有许多状态。SCTP 所定义的状态包括：CLOSED、COOKIE-WAIT、COOKIE-ECHOED、ESTABLISHED、SHUTDOWN-PENDING、SHUTDOWN-SENT、SHUTDOWN-RECEIVED 和 SHUTDOWN-ACK-SENT。
- 一个 DATA 块不能携带属于多个报文的数据，但一个报文可以被划分成几个块（分片）。
- SCTP 关联的建立通常需要四个分组（四向握手）。SCTP 关联的终止通常需要三个分组（三向握手）。SCTP 关联使用 Cookie 防止盲目洪泛攻击，使用验证标志避免插入攻击。
- SCTP 的确认 SACK 报告了累积的 TSN（即按序收到的最后一个数据块的 TSN）以及对收到的分组的选择 TSN。

16.13 实践安排

16.13.1 习题

1. 一分组携带两个 DATA 块，每个块包含 22 字节的用户数据。每个 DATA 块的长度是多少？这个分组的总长度是多少？
2. 一个 SACK 块报告了收到 3 个失序的数据块和 5 个重复的数据块。这个块的总长度是多少（以字节为单位）？
3. 一分组携带一个 COOKIE ECHO 报文和一个 DATA 块。若 Cookie 长度为 200 字节，用户数据长度为 20 字节，试问这个分组长度是多少？
4. 一分组携带一个 COOKIE ACK 报文和一个 DATA 块。若用户数据长度为 20 字节，试问这个分组长度是多少？
5. 五个已到达的 DATA 块携带了如下的信息：

TSN:27	SI:2	SSN:14	BE:00
TSN:33	SI:2	SSN:15	BE:11
TSN:26	SI:2	SSN:14	BE:00
TSN:24	SI:2	SSN:14	BE:00
TSN:21	SI:2	SSN:14	BE:10

- a. 哪个数据块是仅有 1 个分片？
- b. 哪个数据块是第一个分片？

c. 哪个数据块是最后一个分片?

d. 有多少中间的分片丢失了?

6. SACK 中的累积 TSN 的值是 23。而这个 SACK 的前一个累积 TSN 的值是 29。这里有什么问题?

7. SCTP 关联处于 **ESTABLISHED** 状态。它收到了一个 SHUTDOWN 块。若这个主机没有任何待确认的或正在进行中的数据, 试问它需要做些什么?

8. SCTP 关联处于 **COOKIE-WAIT** 状态。它收到了一个 INIT 块。试问它需要做些什么?

9. 下面是一个用十六进制表示的 DATA 块:

00000015 00000005 0003000A 00000000 48656C6C 6F000000

a. 这是按序的还是不按序的块?

b. 这是第一个、最后一个、还是中间的或仅有的分片?

c. 这个块携带了多少字节的填充?

d. 它的 TSN 是什么?

e. 它的 SI 是什么?

f. 它的 SSN 是什么?

g. 这是什么报文?

10. 下面是用十六进制表示的 SCTP 通用首部:

04320017 00000001 00000000

a. 源端口号是什么?

b. 目的端口号是什么?

c. 验证标志是什么?

d. 检验和的值是多少?

11. 接收方的状态如下:

a. 接收队列中的块是 1~8, 11~14 和 16~20。

b. 队列中还有 1800 字节的空间。

c. lastACK 值是 4。

d. 没有收到重复的块。

e. cumTSN 的值是 5。

试给出接收队列的内容和几个变量的值。

12. 试给出习题 11 中的接收方发送的 SACK 报文的内容。

13. 发送方的状态如下:

a. 发送队列中的块是 18~23。

b. curTSN 的值是 20。

c. 窗口大小的值是 2000 字节。

d. inTransit 的值是 200。

如果每个数据块包含 100 字节的数据, 试问现在还可以发送多少个 DATA 块? 下一个将被发送的数据块是哪个?

14. 打开关联的 SCTP 客户使用的初始标志是 806, 初始 TSN 是 14534, 窗口大小是

20000。服务器响应时的初始标志是 2000，初始 TSN 是 670，窗口大小是 14000。试给出在关联建立期间交换的所有 4 个分组的内容。忽略 Cookie 值。

15. 若在前面的习题中，客户发送 7600 个数据块，而服务器发送 570 个数据块，试给出在关联终止期间交换的所有 3 个分组的内容。

16.13.2 研究活动

16. 我们已经定义了与转换图相关的几种情况。试给出包括了下面列举的情况（但不仅限于下述情况）的一些情况：

- a. 在关联建立期间，4 个分组中的任何一个丢失、延迟或重复。
- b. 在关联终止期间，3 个分组中的任何一个丢失、延迟或重复。

17. 如果 SACK 块延迟或丢失了，会发生什么？

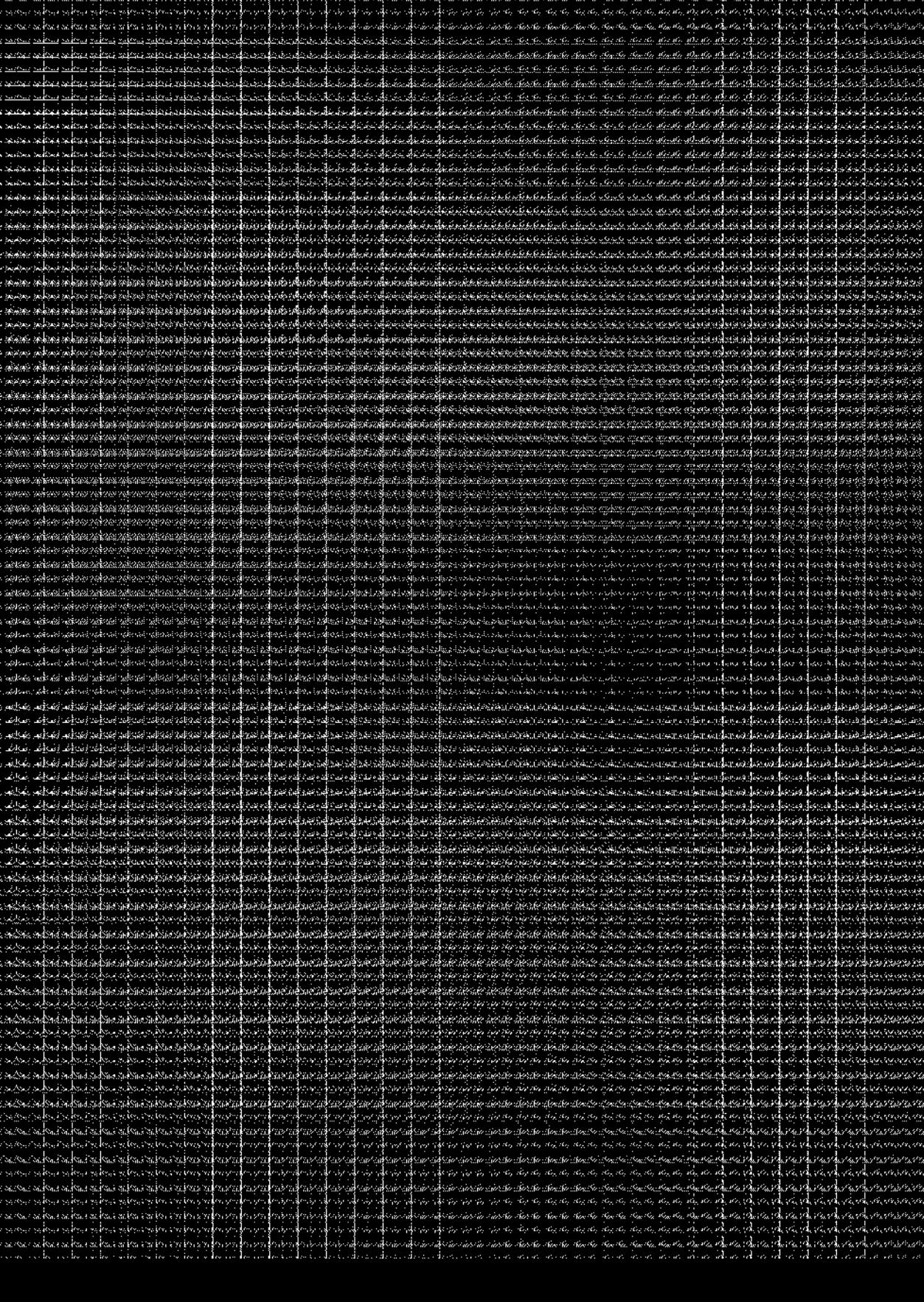
18. 试找出服务器验证 Cookie 需要采取的步骤。

19. 我们在 TCP 中讨论过两个计时器：持续计时器和保活计时器。试找出在 SCTP 中这两个计时器的功能。

20. 查找有关 SCTP 中的 ECN 的更多内容。找出这两个块的格式。

21. 试找出在某些 SCTP 控制块中使用的参数的更多一些的内容。

22. 某些应用程序（如 FTP）在使用 TCP 时需要多个连接。试问：SCTP 的多重流服务可以怎样帮助这些应用程序仅建立一个关联就能支持多个流？



第四部分

应用层

第 17 章 应用层简介

第 18 章 主机配置：DHCP

第 19 章 域名系统（DNS）

第 20 章 远程登录：TELNET 与 SSH

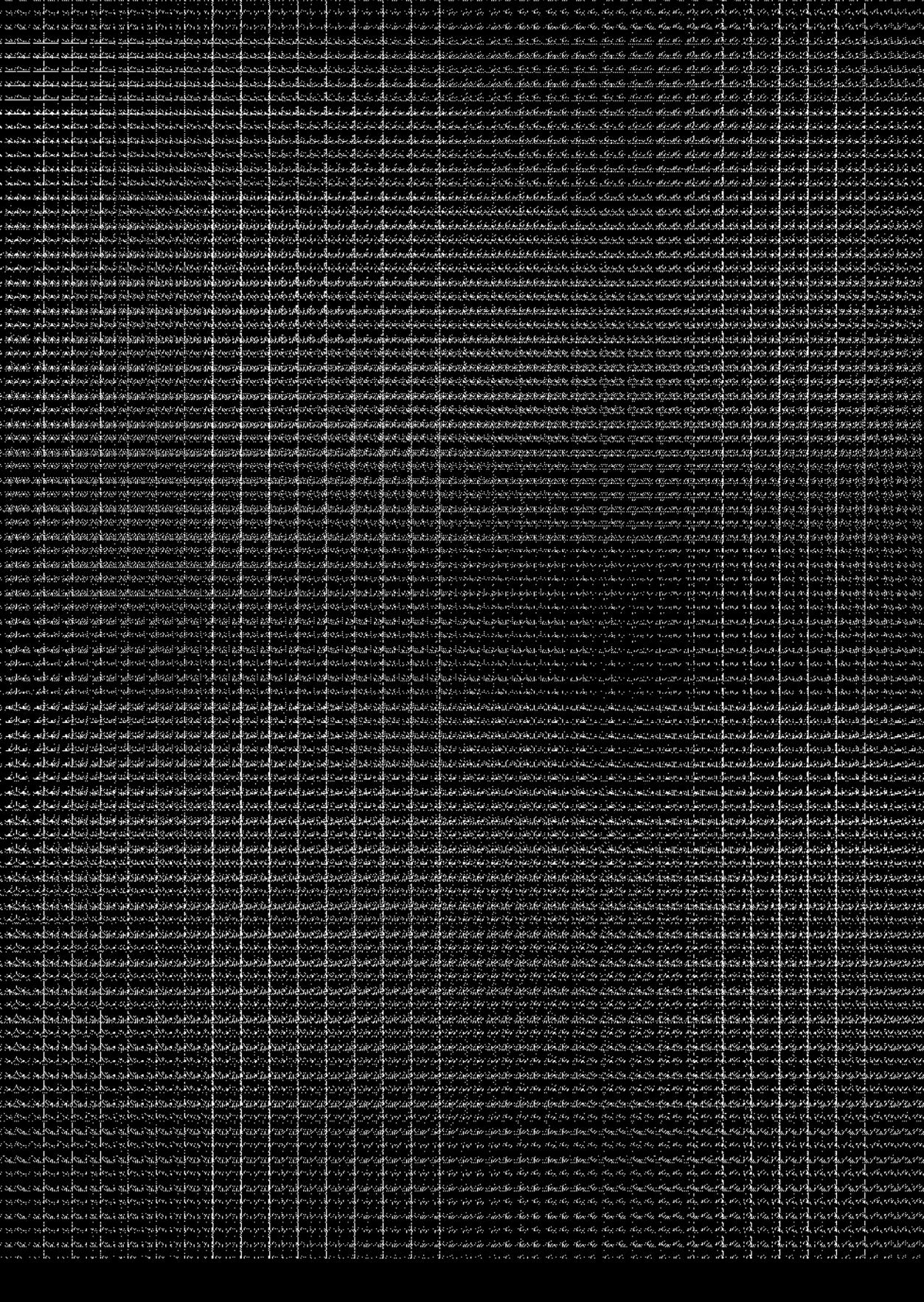
第 21 章 文件传送：FTP 和 TFTP

第 22 章 万维网和 HTTP

第 23 章 电子邮件：SMTP、POP、IMAP 和 MIME

第 24 章 网络管理（SNMP）

第 25 章 多媒体



第 17 章 应用层简介

本章是对应用层的总体概述。在接下来 8 章的内容中，我们将介绍一些因特网中常用的客户-服务器应用程序。在这一章，我们要呈现的是客户-服务器程序设计的概貌，并给出它们实现的一些简单的代码。网络应用程序领域内容极其丰富且复杂，因此不可能用一章的篇幅来涵盖。我们要做的只能是鸟瞰这一领域的全景，使后面 8 章的内容更加容易理解。

目标

本章有以下几个目标：

- 介绍客户-服务器范式。
- 介绍套接字接口并列举其中的一些常用函数。
- 讨论使用了 UDP 提供的无连接服务的客户-服务器通信。
- 讨论使用了 TCP 提供的面向连接服务的客户-服务器通信。
- 给出使用 UDP 服务的客户程序的一个例子。
- 给出使用 UDP 服务的服务器程序的一个例子。
- 给出使用 TCP 服务的客户程序的一个例子。
- 给出使用 TCP 服务的服务器程序的一个例子。
- 简单地讨论 P2P 范式及其应用。

17.1 客户-服务器范式

网络或者说互联网的作用就是向用户提供服务。本地站点的一个用户希望能够接收远程站点的一台计算机所提供的服务。要想实现此目标的一种方法就是运行两个程序。本地计算机上运行一个程序，它向远程计算机请求服务，远程计算机上也运行一个程序，它向请求者提供服务。也就是说，通过互联网相连的两台计算机上必须各运行一个程序，其中一个程序提供服务，另一个程序请求服务。

乍看之下，允许一个运行在本地，一个运行在远程的两个应用程序之间互相通信貌似很简单。但是当我们实现这种方式时就会引出很多问题。这些问题包括：

1. 这两个应用程序应当既能请求服务又能提供服务，还是只能二者选一？一种答案是让一个称为客户的运行在本地主机上的应用程序向另一个称为服务器的运行在远程主机上的应用程序请求服务。换言之，请求服务的责任与提供服务的责任互相独立。一个应用程

序不是请求者（客户），就是提供者（服务器）。也就是说，应用程序必须以客户和服务器的形式成对出现，且两者同名。

2. 服务器应当只向某个特定的客户提供服务，还是说服务器应当向任何请求了它所能提供的服务类型的客户提供服务？最普遍的答案就是让一个服务器向任何需要该类型服务的客户提供服务，而不是为某个特定的客户提供服务。换言之，服务器和客户之间的关系是一对多的。

3. 一台计算机是否只能运行一个程序（客户或者服务器）？答案是连接到因特网上的任何计算机都应当能够运行任意多的客户程序，只要安装了相应的软件。服务器程序需要运行在能够连续不断地运行的计算机上，关于这一点我们将在稍后详细了解。

4. 一个应用程序应当在何时运行？是所有时间，还是仅当它们需要服务时？通常，请求服务的客户程序应当仅在需要时才运行。而提供服务的服务器程序则应当在所有时间都运行，因为它无法知道什么时候它的服务会被请求。

5. 应当用一个通用的应用程序来提供用户希望的所有类型的服务，还是应当为每种类型的服务使用一个应用程序？在 TCP/IP 中，那些使用频率高且用户数量大的服务都有特定的客户-服务器程序。例如，我们有独立的客户-服务器应用程序来允许用户访问文件、发送电子邮件等等。而对于比较个性化一些的服务，我们应当有一个通用的应用程序以允许用户能够接入到远程计算机所提供的服务。例如，我们应当有一个客户服务器应用程序能够允许用户登录到一台远程计算机上，然后再使用由该计算机提供的服务。

17.1.1 服务器

服务器（server）就是运行在远程主机上向客户提供服务的程序。当它启动时，它敞开大门迎接来自客户的请求，但是它从来不会主动启动一个服务，只有当它被请求时才会如此做。

服务器程序是一个无限程序。当它启动后就一直运行，除非出现故障。它一直在等待着来自客户的请求到达。当一个请求到达时，它就响应这个请求，可能是顺序的，也可能是并发的，这一点我们稍后再详细解释。

17.1.2 客户

客户（client）是运行在本地主机上的一个程序，它请求得到服务器的服务。客户程序是有限的，也就是说它是由用户启动（或由另一个应用程序启动），并且在服务完成后终止。通常，客户利用远程主机的 IP 地址以及在该主机上运行的某个特定服务器程序的熟知端口地址来打开通信信道。在通信信道被打开后，客户就可以发送请求并接收响应。虽然这种请求-响应可能会往返重复多次，但整个过程是有限的，最终一切都会结束。

17.1.3 并发

客户和服务器都可以运行在并发模式下。

客户的并发

客户可以在一台主机上顺序地或者并发地运行。客户顺序地 (iteratively) 运行表示它们一个接一个地运行。一个客户程序必须启动，运行，并在终止之后，主机才能启动另一个客户。不过，今天的绝大多数计算机都允许并发 (concurrent) 的客户。也就是说，两个或多个客户可同时运行。

服务器的并发

顺序服务器一次只能处理一个请求。它接收一个请求，处理该请求，并向请求者发送响应，在此之后，它才能接着处理另一个请求。与此相反的是，并发服务器能够同时处理多个请求，因而需要在多个请求之间分享它的时间。

服务器或者使用无连接的运输层协议 UDP，或者使用面向连接的运输层协议 TCP/SCTP。因此，服务器的操作取决于两项因素：运输层协议和服务方式。理论上讲，我们可以有四种类型的服务器：无连接顺序型、无连接并发型、面向连接顺序型和面向连接并发型（参见图 17.1）。

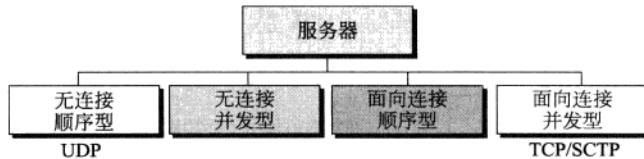


图 17.1 服务器类型

无连接顺序服务器

使用 UDP 的服务器通常都是顺序的，正如我们曾经提到的，它是指服务器一次只处理一个请求。服务器从接收到的 UDP 数据报中获得一个请求，处理该请求，并把响应递交给 UDP 以便返回给客户。在此期间服务器对其他数据报不予理睬，这些数据报被保存在一个队列中等待服务。它们可能全部来自同一个客户，也可能分别来自不同的客户。无论哪种情况，它们按照到达的顺序一个接一个地被处理。

为了完成这项任务，服务器只需要使用一个端口，即熟知端口。所有到达该端口的数据报排队等待着被服务，如图 17.2 所示。

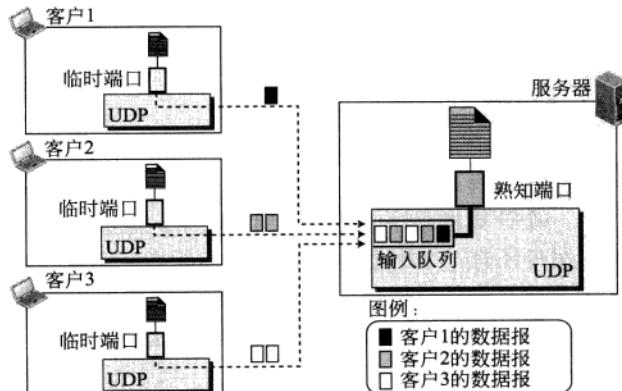


图 17.2 无连接顺序服务器

面向连接的并发服务器

使用 TCP（或 SCTP）的服务器通常是并发的。这就表示该服务器能够一次为多个客户服务。因为通信是面向连接的，也就是说请求是一个字节流，它可能在多个报文段中到达，并且返回的响应也可能要占用多个报文段。服务器与每个客户之间都建立了一条连接，并且这条连接一直保持打开，直至整个字节流处理完成且连接终止。

这种类型的服务器不能只用一个端口，因为有可能多个连接同时打开，而每条连接都需要一个端口。服务器需要多个端口，但能够被它使用的只有一个熟知端口。解决的办法是使用一个熟知端口和多个临时端口。服务器从熟知端口处接受连接请求。客户可以在发起连接时访问这个熟知端口。而在连接建立后，服务器就为该连接指派一个临时端口，以释放它自己的熟知端口。此后，数据的传送发生在两个临时端口之间，一个在客户端，另一个在服务器端。此时熟知端口就空闲出来可以为另一个客户建立连接了。要为若干个客户提供服务，服务器需要创建子进程，也就是原始进程（父进程）的复本。

服务器还必须为每条连接建立一个队列。来自客户的报文段被保存在相应的队列中，并由服务器并发地提供服务。图 17.3 描绘了此配置。

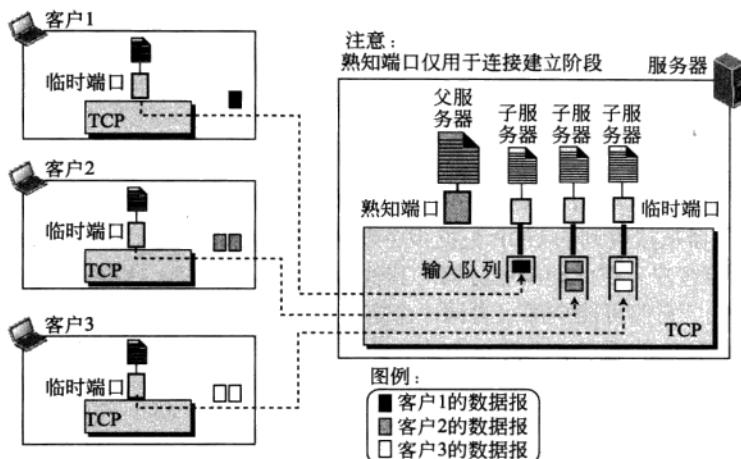


图 17.3 面向连接的并发服务器

17.1.4 套接字接口

客户进程如何与服务器进程通信呢？一个计算机程序就是一组预先定义的指令，它们告诉计算机要做什么。一个计算机程序中有一个用于数学运算的指令集合，还有一个用于处理字符的指令集合，另外还有一个用于输入/输出访问的指令集合。如果要让一个程序与运行在另一台主机上的程序之间能够通信，我们就需要一个新的指令集合来告诉运输层打开连接，向另一端发送数据，接收来自另一端的数据，以及关闭连接。此种类型的指令集合通常被称为接口（interface）。

接口就是为了两个实体之间的交互而设计的指令集合。

人们已经为计算机通信设计了若干接口。其中有三个接口是通用的：套接字接口（socket interface）、运输层接口（transport layer interface）以及 STREAM。虽然网络编程人员需要对这三个接口都很熟悉，但是在本章我们只简单地讨论套接字接口，以便展示应用层网络通信的总体思想。

套接字接口起源于 20 世纪 80 年代的 Berkeley 大学，当时是作为 UNIX 环境中的一部分。为了更好地理解套接字接口的概念，我们需要考虑到底层操作系统（如 UNIX 或 Windows）与 TCP/IP 协议族之间的关系。这是一个到目前为止都被我们忽略的问题。图 17.4 从概念上描绘了操作系统和 TCP/IP 协议族之间的关系。

作为指令集合的套接字接口位于操作系统和应用程序之间。应用程序如果想要接入由 TCP/IP 协议族提供的服务，就必须使用在套接字接口中定义的指令。

例 17.1

大多数编程语言都有一个文件接口，即一组允许程序员打开文件，从文件中读取，写入到文件，并对该文件执行一些其他操作，最后关闭该文件的指令集合。当程序需要打开某个文件时，就要使用由操作系统掌握的该文件的文件名。当该文件被打开后，操作系统会返回对该文件的一个引用（整数或指针），它被用于其他指令中，如读操作或写操作。

套接字

套接字（socket）是模拟了我们在生活中常见的硬件插口的软件抽象。要想使用通信信道，应用程序（客户或服务器）需要请求操作系统创建一个套接字。然后，应用程序就能够插入该套接字以发送和接收数据。为了使数据通信得以顺利进行，就需要一对套接字，通信的两端各一个。图 17.5 用我们日常生活中使用的插口和插头（例如，电话机的）来模拟这个抽象的概念。在因特网中，套接字就是一个软件数据结构，稍后我们将进一步讨论。



图 17.4 操作系统和 TCP/IP 协议族之间的关系



图 17.5 套接字的概念

数据结构

定义了套接字的数据结构的格式取决于进程所使用的底层语言。在本章剩余的部分中，我们都假定进程是用 C 语言来编写的。在 C 语言中，套接字被定义为五字段的结构（结构

或记录), 如图 17.6 所示。

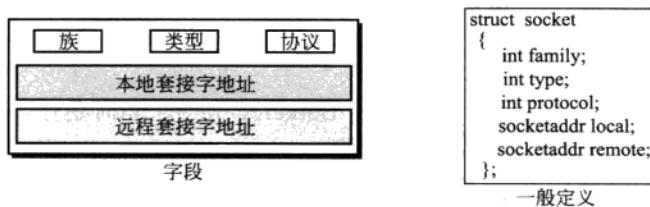


图 17.6 套接字的数据结构

请注意, 程序员不应重新定义这个结构, 它是已经定义好的。程序员只需要使用包含了这个定义的头文件即可(稍后再讨论)。让我们简单地定义一下这个结构中使用的各个字段:

- **族** 这个字段定义了一个协议组: IPv4、IPv6、UNIX 主域协议等等。在 TCP/IP 中我们使用的族类型的定义是: 用常量 IF_INET 来表示 IPv4 协议, 用常量 IF_INET6 来表示 IPv6 协议。
- **类型** 这个字段定义了四种类型的套接字: SOCK_STREAM (用于 TCP), SOCK_DGRAM (用于 UDP), SOCK_SEQPACKET (用于 SCTP) 和 SOCK_RAW (用于直接使用 IP 服务的应用)。这些类型如图 17.7 所示。
- **协议** 这个字段定义了接口使用的协议。对于 TCP/IP 协议族, 它被设置为 0。
- **本地套接字地址** 这个字段定义了本地套接字地址。正如我们在第 13 章中所讨论的, 套接字地址就是 IP 地址和端口地址的组合。
- **远程套接字地址** 这个字段定义了远程的套接字地址。

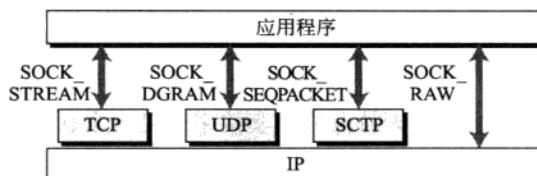


图 17.7 套接字类型

套接字地址的结构

在我们能够使用套接字之前, 还需要了解套接字地址的结构, 它是 IP 地址和端口地址的组合。虽然网络编程已经定义了很多种套接字地址, 但我们在里只定义用于 IPv4 的套接字地址。在这个版本中, 套接字地址是一个复杂的数据结构(C 语言中的结构类型), 如图 17.8 所示。

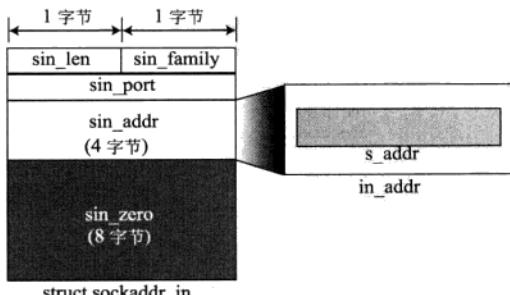


图 17.8 IPv4 的套接字地址

请注意，结构 `sockaddr_in` 有五个字段，其中的 `sin_addr` 字段本身又是一个 `in_addr` 类型的结构，这个结构只有一个字段 `s_addr`。我们先来定义结构 `in_addr` 如下：

```
struct in_addr
{
    in_addr_t s_addr; //一个 32 位的 IPv4 地址
};
```

现在我们来定义结构 `sockaddr_in`：

```
struct sockaddr_in
{
    uint8_t sin_len; //结构的长度 (16 字节)
    sa_family_t sin_family; //设置为 AF_INET
    in_port_t sin_port; //一个 16 位的端口号
    struct in_addr sin_addr; //一个 32 位的 IPv4 地址
    char sin_zero[8]; //未使用
};
```

函数

进程与操作系统之间的交互是通过一组预先定义的函数来完成的。在这一小节，我们要介绍这些函数，而在下面的小节中，我们将展示这些函数如何组合起来以建立进程。

□ socket 函数

操作系统定义的套接字结构如图 17.6 所示。但是除非收到进程的命令，操作系统是不会创建套接字的，因此进程需要调用 `socket` 函数来创建一个套接字。这个函数的原型如下所示：

```
int socket (int family, int type, int protocol);
```

调用这个函数会创建一个套接字，但是在新创建的套接字结构中只填写了三个字段（族，类型和协议）。如果调用成功，这个函数返回唯一的套接字描述符 `sockfd`（非负整数），它可以被用在其他调用中来指向该套接字。如果调用不成功，操作系统返回-1。

□ bind 函数

`socket` 函数只填写了该套接字的部分字段。要将本地计算机和本地端口绑定到该套接字，就需要调用 `bind` 函数。`bind` 函数填写本地套接字地址的值（本地 IP 地址和本地端口号）。如果绑定失败，则返回-1。它的原型如下所示：

```
int bind (int sockfd, const struct sockaddr* localAddress, socklen_t* addrLen);
```

在这个原型中，`sockfd` 是调用 `socket` 函数时返回的套接字描述符的值，`localAddress` 是指向一个套接字地址的指针，它需要已经被定义好（通过系统或程序员），`addrLen` 是套接字地址的长度。在后面我们将会了解何时需要使用 `bind` 函数，何时不需要使用。

□ connect 函数

`connect` 函数用于向套接字结构中添加远程套接字地址。如果连接失败，它返回-1。它的原型如下所示：

```
int connect (int sockfd, const struct sockaddr* remoteAddress,
socklen_t* addrLen);
```

这个原型中的参数与前面的相同，只是第二个和第三个参数定义的不是本地套接字地址，而是远程套接字地址。

□ listen 函数

`listen` 函数只能被 TCP 服务器调用。在 TCP 创建并绑定好套接字后，它必须通知操作系统说套接字已经准备好，可以接收客户的请求了。通过调用 `listen` 函数就能完成这个任务。参数 `backlog` 是连接请求的最大数量。如果调用失败，函数返回-1。它的原型如下所示：

```
int listen (int sockfd, int backlog);
```

□ accept 函数

`accept` 函数是服务器进程用来通知 TCP 它已经准备好接收来自客户的请求。如果调用失败，函数返回-1。它的原型如下所示：

```
int accept (int sockfd, const struct sockaddr* clientAddr, socklen_t*
addrLen);
```

后两个参数分别是指向地址和长度的指针。`accept` 函数是一个阻塞函数，也就是说当它被调用后，它会阻塞自己（指进程停在此处不再往下执行——译注），直至客户的连接建立起来。然后 `accept` 函数才能获得客户的套接字地址和地址长度，并将它们传递给服务器进程，以便用于访问客户。有以下几点值得注意：

- `accept` 函数的调用会使服务器检查缓存中是否还有任何正在等待的客户连接请求。如果没有，`accept` 函数让进程进入睡眠状态。当该队列中至少有一个请求时，进程被唤醒。
- 在成功调用 `accept` 之后，新的套接字被创建，在客户套接字与服务器的这个新的套接字之间的通信建立。
- `accept` 函数接收到的地址被填入新套接字的远程套接字地址字段。
- 收到的客户地址用指针返回给进程。如果程序员不需要这个地址，可以用 `NULL` 来替换。
- 返回的地址长度被传递给该函数，同样以指针的形式返回。如果不需这个地址长度，可以用 `NULL` 来替换。

□ fork 函数

`fork` 函数被进程用来复制自己。调用 `fork` 函数的进程称为父进程，创建出来的进程（也就是那个复本）称为子进程。它的原型如下所示：

```
pid_t fork (fork);
```

很有意思的一件事是 `fork` 进程调用一次，却要返回两次。在父进程中，返回值是一个正整数（调用它的父进程的进程 ID）。在子进程中，返回值是 0。如果出现错误，`fork` 函数返回-1。在调用 `fork` 函数之后，两个进程并发运行，CPU 交替地给每个进程分配运行时间。

□ send 和 recv 函数

进程调用 `send` 函数以便向运行在远程主机上的另一个进程发送数据。进程调用 `recv`

函数以便接收从运行在远程主机上的另一个进程发过来的数据。它们都假定在这两台主机之间已经有一条打开的连接，因此，它们只能用于 TCP（或 SCTP）。这两个函数返回的是发送或接收的字节数。

```
int send (int sockfd, const void* sendbuf, int nbytes, int flags);
int recv (int sockfd, void* recvbuf, int nbytes, int flags);
```

这里的 sockfd 是套接字描述符；sendbuf 是指向一个缓存的指针，在这个缓存中保存的是将要发送的数据；recvbuf 也是指向一个缓存的指针，在这个缓存中保存的是已接收的数据；nbytes 是将要发送或已接收的数据的大小。如果调用成功，这两个函数返回实际发送或接收的字节数，如果出现错误，则返回-1。

□ sendto 和 recvfrom 函数

进程调用 sendto 函数以便使用 UDP 服务向远程进程发送数据。进程调用 recvfrom 函数以便使用 UDP 服务接收来自远程进程的数据。因为 UDP 是无连接协议，所有其中的一个参数要定义远程套接字地址（目的地址或源地址）。

```
int sendto (int sockfd, const void* sendbuf, int bufLen, int flags
           struct sockaddr* destinationAddress, socklen_t* addrLen);
int recvfrom (int sockfd, void* recvbuf, int bufLen, int flags
              struct sockaddr* sourceAddress, socklen_t* addrLen);
```

这里的 sockfd 是套接字描述符；buf 是指向缓存的指针，在这个缓存中保存的是将要发送的数据或接收到的数据；bufLen 是缓存的大小。虽然 flag 的值可以非零，但在本章我们例举的简单程序中它们都被置为 0。如果调用成功，这两个函数返回发送或接收的字节数，如果出现错误，则返回-1。

□ close 函数

进程调用 close 函数来关闭一个套接字。

```
int close (int sockfd);
```

在调用此函数之后，该 sockfd 就无效了。这个函数返回一个整数。如果成功，则返回 0；如果出现错误，则返回-1。

□ 字节排序函数

计算机中的信息是以主机字节序来保存的，它可能是小数在前（little-endian），也就是最低位字节被保存在开始的地址中，也可能大数在前（big-endian），也就是最高位字节被保存在开始的地址中。而在网络编程中，数据和其他各种信息都是网络字节序的，它是大数在前的。因为当我们编写程序时，并不能确定像 IP 地址和端口号这样的信息在计算机中是如何保存的，所以我们需要把它们转换为网络字节序。人们为此而专门设计了两个函数： htons（host to network short）把一个短值（16 位）转换为网络字节序； htonl（host to network long）把一个长值（32 位）转换为网络字节序。另外还有两个完全相反的函数： ntohs 和 ntohl。这些函数的原型如下所示：

```
uint16_t htons (uint16_t shortValue);

uint32_t htonl (uint32_t longValue);
```

```

    uint16_t ntohs (uint16_t shortValue);

    uint32_t ntohs (uint32_t longValue);

```

□ 内存管理函数

最后我们需要一些函数来管理保存在内存中的值。在这里要介绍三个常用的内存函数，不过在本章它们并不是都用得到。

```

void* memset (void* destination, int chr, size_t len);

void* memcpy (void* destination, const void* source, size_t nbytes);

int memcmp (const void* ptr1, const void* ptr2, size_t nbytes);

```

第一个函数 `memset`（内存设置）用于在 `destination` 指针（起始地址）指定的内存中设置（保存）特定数量（`len` 的值）的字节。第二个函数 `memcpy`（内存拷贝）用于从内存中的某处（`source`）复制特定数量（`nbytes` 的值）的字节到内存的另一处（`destination`）。第三个函数 `memcmp`（内存比较）用于比较以 `ptr1` 和 `ptr2` 开始的两组（`nbytes`）字节。若这两组字节相同，则结果为 0；若第一组小于第二组，则结果小于 0；若第一组大于第二组，则结果大于 0。这个比较基于 C 语言中的字符串字节的比较。

□ 地址转换函数

通常，我们喜欢使用点分十进制格式的 32 位 IP 地址。但是，当我们想把这个地址保存到套接字中时，就需要将其转换为一个数值。有两个函数可分别用于将地址从点分十进制表示转换为一个数值，或从一个数值转换为点分十进制表示：`inet_pton`（从记法到数值）和 `inet_ntop`（从数值到记法）。我们要使用的 `family` 的值是常量 `AF_INET`。这两个函数的原型如下所示：

```

int inet_pton (int family, const char* stringAddr, void* numericAddr);

char* inet_ntop (int family, const void* numericAddr, char* stringAddr,
int len);

```

头文件

要使用前面所描述的函数，就需要一组头文件。我们在一个独立的文件中定义头文件，这个文件被命名为“`headerFiles.h`”。我们将在后面的程序中包括这个文件，以避免每次都要包括长长的头文件列表。不是在所有的程序中所有这些头文件都需要，但是最好还是把它们都包括了，以免有遗漏。

```

// "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>
```

17.1.5 使用 UDP 的通信

图 17.9 所示为此类通信的一个简化了的流程图。

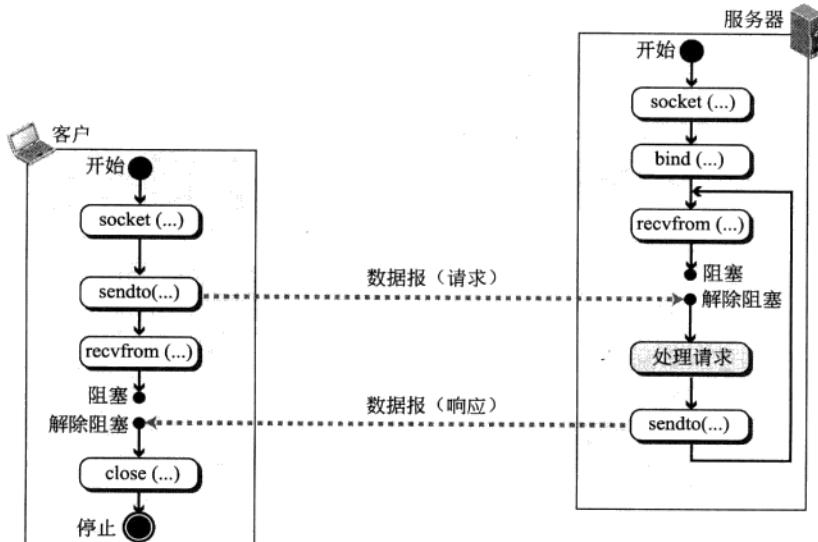


图 17.9 使用 UDP 的无连接的顺序通信

服务器进程

服务器进程先启动。服务器进程调用 `socket` 函数以创建套接字。然后它再调用 `bind` 函数将自己的熟知端口号和运行这个服务器进程的计算机的 IP 地址绑定到该套接字。然后服务器调用 `recvfrom` 函数，它会进入阻塞状态，直至有一个数据报到达。当数据报到达时，`recvfrom` 函数解除阻塞，并从接收到的数据报中提取客户套接字地址及其地址长度，然后把这两个信息返回给进程。进程保存这两个信息，接着调用一个过程（函数）来处理该请求。当结果准备好后，服务器进程调用 `sendto` 函数，并利用前面保存的信息将结果发送到发出此请求的客户。服务器使用了无限循环，以响应来自同一个客户或不同客户的的所有请求。

客户进程

客户进程更简单一些。客户调用 `socket` 函数以创建套接字。然后它调用 `sendto` 函数，并向其传递服务器的套接字地址和缓存位置。UDP 能够从这个缓存中获取到数据并生成数据报。然后，客户调用 `recvfrom` 函数，这个函数进入阻塞状态，直至服务器返回的响应到达。当响应到达时，UDP 把数据交付给客户进程，也就是 `recvfrom` 函数解除阻塞，然后把

收到的数据递交给客户进程。请注意，我们假设客户报文小得足以装进一个数据报中。如果情况不是这样，那么就要反复调用 sendto 和 recvfrom 这两个函数。但是，服务器并不知道这是一次多个数据报的通信，它只是独立地对待每一个请求。

例 17.2

作为例子，让我们来看看应当如何设计并编写以下两个程序：echo 服务器和 echo 客户。该客户向服务器发送了一行文本，服务器将相同的文本行返回给客户。虽然这个客户/服务器程序看起来好像没有什么用处，但实际上它还是有一些应用的。例如，当某台计算机想测试网络上的另一台计算机是否仍然在工作时就可以使用它。为了更好地理解程序中的代码，我们先用图 17.10 来描绘这两个程序所使用的变量的总体情况。

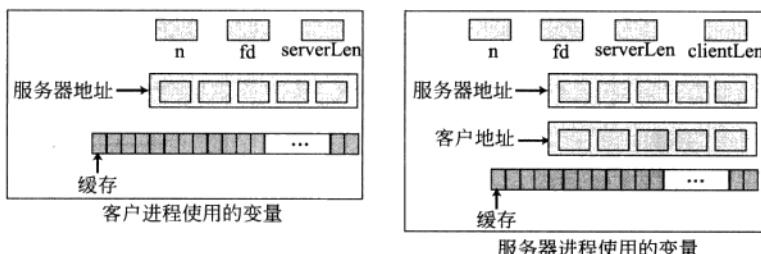


图 17.10 在使用了 UDP 服务的 echo 服务器和 echo 客户中所用到的变量

表 17.1 所示为 echo 服务器的程序。为了简化整个程序，我们忽略了很多细节和差错处理。在练习中，我们会要求读者提供更多的细节。

表 17.1 使用 UDP 服务的 echo 服务器程序

```

01 //UDP echo 服务器程序
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     //变量的声明和定义
07     int sd;                      //套接字描述符
08     int nr;                      //接收的字节数
09     char buffer [256];           //数据缓存
10     struct sockaddr_in serverAddr; //服务器地址
11     struct sockaddr_in clientAddr; //客户地址
12     int clAddrLen;              //客户地址长度
13
14     //创建套接字
15     sd = socket (PF_INET, SOCK_DGRAM, 0);
16
17     //将本地地址和端口绑定到套接字
18     memset (&serverAddr, 0, sizeof(serverAddr));
19     serverAddr.sin_family = AF_INET;

```

```

18 serverAddr.sin_addr.s_addr = htonl (INADDR_ANY); //默认地址
19 serverAddr.sin_port = htons (7) //我们假设端口为 7
20 bind (sd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
21 //接收并回送数据报
22 for (;;) //一直运行
23 {
24     nr = recvfrom (sd, buffer, 256, 0, (struct sockaddr*)&clientAddr, &clAddrLen);
25     sendto (sd, buffer, nr, 0, (struct sockaddr*)&clientAddr, sizeof(clientAddr));
26 }
27 } //echo 服务器程序结束

```

第 14 行创建了套接字。第 16~19 行创建了本地套接字地址，远程套接字地址在后面的第 24 行创建，稍后再解释。第 20 行将本地套接字地址绑定到该套接字。第 22~26 行接收和发送数据报，它们可能来自多个客户。当服务器接收到一个来自客户的数据报时，该客户的套接字地址及其长度被返回给服务器进程。这两个信息用于第 25 行向相应的客户返回回送报文。请注意，在这个程序中我们忽略了很多错误检查代码，它们被留作练习。

表 17.2 所示为 echo 进程的客户程序。我们假设该客户只发送了一个需要服务器回送的报文。如果我们要发送多个报文，数据发送代码段就应当使用一个循环语句来重复执行。

表 17.2 使用 UDP 服务的 echo 客户程序

```

01 //UDP echo 客户程序
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     //变量的声明和定义
07     int sd; //套接字描述符
08     int ns; //发送的字节数
09     int nr; //接收的字节数
10     char buffer [256]; //数据缓存
11     struct sockaddr_in serverAddr; //套接字地址
12     //创建套接字
13     sd = socket (PF_INET, SOCK_DGRAM, 0);
14     //建立服务器套接字地址
15     memset (&serverAddr, 0, sizeof(serverAddr));
16     serverAddr.sin_family = AF_INET;
17     inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
18     serverAddr.sin_port = htons (7)

```

```

19 //发送和接收数据报
20 fgets(buffer, 256, stdin);
21 ns = sendto (sd, buffer, strlen(buffer), 0,
22             (struct sockaddr)&serverAddr, sizeof(serveraddr))
23 recvfrom (sd, buffer, strlen (buffer), 0, NULL, NULL);
24 buffer [nr] = 0;
25 printf ("Received from server:");
26 fputs (buffer, stdout);
27 //关闭和退出
28 close (sd);
29 exit (0);
30 } //echo 客户程序结束

```

第 13 行创建了一个套接字。第 15~18 行所示为我们如何建立服务器套接字地址，在这里不需要建立客户套接字地址。第 20~26 行从键盘读取一个字符串，然后将其发送到服务器，最后接收服务器返回的字符串。第 24 行在接收到的字符串后增加了一个空字符，以便它在第 26 行中显示。第 28 行关闭了这个套接字。在这个程序中我们忽略了很多错误检查代码，它们被留作练习。

服务器和客户程序都还需要增加一些错误检查代码，才能使之完整。

17.1.6 使用 TCP 的通信

现在我们来讨论使用 TCP 服务的面向连接的并发通信（SCTP 的情况类似）。图 17.11 描绘了这种类型的通信的一般流程图。

服务器进程

服务器进程首先启动。它调用 `socket` 函数以创建套接字，这个套接字被我们称为监听套接字。监听套接字仅用于连接建立阶段。然后，服务器进程调用 `bind` 函数将运行该服务器的计算机的套接字地址绑定到连接。服务器进程接着调用 `accept` 函数。这个函数是一个阻塞函数，当被调用时，它就进入阻塞状态，直至 TCP 接收到来自客户的连接请求（SYN 报文段）。然后 `accept` 函数解除阻塞，并创建新的套接字，我们称为连接套接字，其中包含了发送该 SYN 报文段的客户的套接字地址。在 `accept` 函数解除阻塞后，服务器进程就知道有客户需要它的服务。为了提供并发性，服务器进程（父进程）调用 `fork` 函数。这个函数创建一个新的进程（子进程），它与父进程一模一样。在调用了 `fork` 函数之后，就有两个进程在同时运行，但是它们各做各的事。现在每个进程都有两个套接字：监听套接字和连接套接字。父进程把服务客户任务完全转交给子进程，并再次调用 `accept` 函数以等待另一个客户来请求连接。现在，子进程已准备好为客户服务。它首先关闭监听套接字，然后调用 `recv` 函数接收来自客户的数据。像 `recvfrom` 函数一样，`recv` 函数也是一个阻塞函数，在报文段到来之前它一直处于阻塞状态。子进程使用了一个循环并重复调用 `recv` 函数，直至它接收到由客户发来的所有报文段。接着，子进程把完整的数据传递给一个函数（我们

称它为 handleRequest)，由它来处理请求并返回结果。然后，通过一次调用 send 函数，把这个结果发送给客户。在这里我们需要强调几点。首先，我们使用的流程图是有可能出现的一种最简化的情况。服务器还可能会使用多个函数来接收和发送数据，以便为特定的应用程序选择一个合适的函数。其次，我们假设向客户发送的数据尺寸足够小，只需要调用一次 send 函数就能发送完，如果不是这样，我们就需要用一个循环来重复地调用 send 函数。第三，虽然服务器进程可以只调用一次 send 函数来发送数据，但是 TCP 在发送数据时可能需要多个报文段。因此，客户进程可能不只接收到一个报文段，这一点我们将在介绍客户进程时再解释。

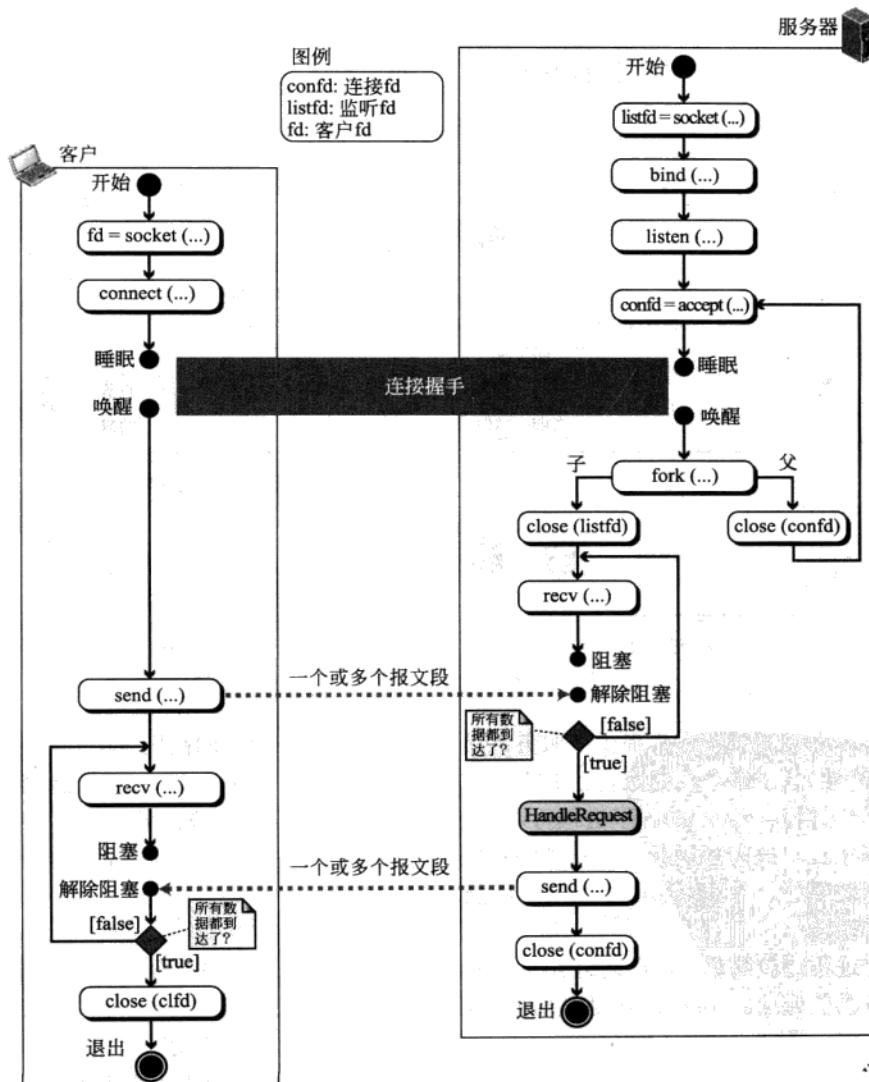


图 17.11 面向连接的并发通信流程图

图 17.12 所示为从套接字的角度看父进程和子进程的状态。图中的 a 部分描绘了 accept 函数返回前的状态。父进程使用监听套接字等待来自客户的请求。当 accept 函数阻塞并返回后 (b 部分)，父进程就有两个套接字：监听套接字和连接套接字。客户被连接到连接套接字。在调用 fork 函数后 (c 部分)，我们有了两个进程，每个进程都有两个套接字。这两个进程都要与客户连接。父进程需要关闭它的连接套接字，使之不再与客户相连，以便能够监听来自其他客户的请求 (d 部分)。在子进程开始向连接客户提供服务之前，它需要先关闭自己的监听套接字，以便后面的请求不会影响到它 (e 部分)。最后，当子进程完成对连接客户的的服务后，它需要关闭自己的连接套接字，使自己断开与提供服务的客户之间的联系 (f 部分)。

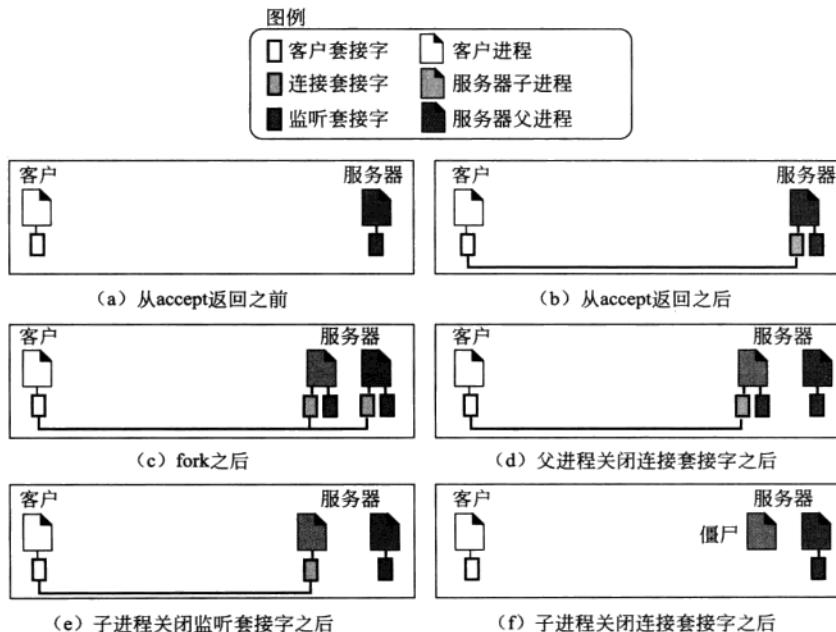


图 17.12 从套接字的角度看父进程和子进程的状态

子进程在向相应的客户提供了服务之后必须被销毁，虽然我们在程序中并没有这么做（为了简单性）。子进程在完成自己的职责后就进入安眠状态，这在 UNIX 系统环境中通常被称为僵尸进程（zombie）。子进程可以在它不再被需要时立即销毁。另外还有一种方式，系统可以每过一段时间运行一个特殊的程序，以销毁系统中所有的 zombie。这些 zombie 占用了系统空间，并且会影响到系统的性能。

客户进程

客户进程比较简单。客户调用 `socket` 函数来创建 socket。然后它调用 `connect` 函数以请求与服务器的连接。`connect` 函数是阻塞函数，它在两个 TCP 之间的连接建立之前一直处于阻塞状态。当 `connect` 函数返回后，客户调用 `send` 函数向服务器发送数据。我们仅调用了一次 `send` 函数，这是假设一次调用就能够发送所有的数据。根据应用程序的不同，我们可能会需要重复地调用这个函数（循环）。然后，客户调用 `recv` 函数，开始时这个函数也

处于阻塞状态，直至有报文段到达且 TCP 将这些数据交付给了进程。请注意，虽然这些数据是服务器进程通过一次调用 `send` 函数发送过来的，但是服务器端的 TCP 可能使用了多个报文段来传送这些数据。也就是说，我们可能需要重复调用 `recv` 函数才能接收到所有的数据。这个循环是由 `recv` 函数的返回值来控制的。

例 17.3

我们希望编写两个程序来说明使用 TCP 服务的 echo 客户和 echo 服务器是什么样的。图 17.13 描绘了我们在这两个程序中使用的变量。因为数据可能在不同的块中到达，所以需要两个指向缓存的指针。第一个指针是固定的，总是指向缓存的起始位置。第二个指针是移动的，以便到达的字节能够紧跟在前一个数据区的后面。

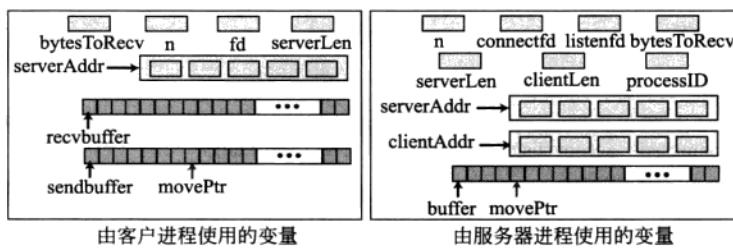


图 17.13 例 17.3 中使用的变量

表 17.3 所示为使用 TCP 服务的 echo 服务器程序。我们精简掉了很多细节，将它们留给有关网络编程的书籍。我们只是想给出此程序总体面貌。

表 17.3 使用 TCP 服务的 echo 服务器程序

```

01 //TCP echo 服务器程序
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     //变量的声明和定义
07     int listensd;           //监听套接字描述符
08     int connectsd;          //连接套接字描述符
09     int n;                  //每一次接收的字节数
10     int bytesToRecv;        //接收的总字节数
11     int processID;          //子进程的 ID
12     char buffer [256];       //数据缓存
13     char* movePtr;          //指向缓存的指针
14     struct sockaddr_in serverAddr; //服务器地址
15     struct sockaddr_in clientAddr; //客户地址
16     int clAddrLen;           //客户地址长度

```

```

17 //创建监听套接字
18 listenfd = socket (PF_INET, SOCK_STREAM, 0);
19 //为监听套接字绑定本地地址和端口
20 memset (&serverAddr, 0, sizeof (serverAddr));
21 serverAddr.sin_family = AF_INET;
22 serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
23 serverAddr.sin_port = htons (7);           //我们假设端口是 7
24 bind (listenfd, &serverAddr, sizeof (serverAddr));
25 //监听连接请求
26 listen (listenfd, 5);
27 //处理连接
28 for ( ; )                                //永远运行
29 {
30     connectfd = accept (listenfd, &clientAddr, &clAddrLen);
31     processID = fork ();
32     if (processID == 0)                      //子进程
33     {
34         close (listenfd);
35         bytesToRecv = 256;
36         movePtr = buffer;
37         while ( (n = recv (connectfd, movePtr, bytesToRecv, 0)) > 0)
38         {
39             movePtr = movePtr + n;
40             bytesToRecv = movePtr - n;
41         }          //while 结束
42         send (connectfd, buffer, 256, 0);
43         exit (0);
44     }      //if 结束
45     close (connectfd);                     //回到父进程
46 }
47 } //for 循环结束
    //结束 echo 服务器程序

```

这段程序以图 17.11 所示的流程图为蓝本。每当 `recv` 函数解除阻塞时，它就获得一些数据，并将这些数据保存在缓存的尾端。于是 `movePtr` 移动到指向下一个数据块将被保存的位置上（行 39）。将要读入的字节数也从初始值（256）不断减少，以防止缓存的溢出（行 40）。在所有数据都被接收后，服务器调用 `send` 函数把这些数据全部发送给客户。正如我们在前面所提到的，在这里只调用了一次 `send` 函数，但是 TCP 可能要用若干个报文段来发送这些数据。然后，子进程调用 `close` 函数以销毁该连接套接字。

表 17.4 所示为使用 TCP 服务的 echo 客户程序。它使用了与表 17.2 中所描述的一样的策略来创建服务器套接字地址。然后这个程序通过键盘获取需要被回送的字符串，把这个字符串保存到 sendBuffer 中，并发送它。服务器返回的结果可能在几个报文段中。这个程序使用了一个循环以重复调用 recv 函数，直至所有数据都到达。与往常一样，我们忽略了差错检查的代码，目的是为了保存程序的简洁性。如果要实际地使用这段代码来发送数据到服务器，那些代码还是需要添加的。

表 17.4 使用 TCP 服务的 echo 客户程序

```
01 //TCP echo 客户程序
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     //变量的声明和定义
07     int sd;                                //套接字描述符
08     int n;                                 //已接收的字节数
09     int bytesToRecv;                      //接收的总字节数
10     char sendBuffer [256];                 //发送缓存
11     char recvBuffer [256];                 //接收缓存
12     char* movePtr;                        //指向缓存的指针
13     struct sockaddr_in serverAddr;        //服务器地址
14
15     //创建套接字
16     sd = socket (PF_INET, SOCK_STREAM, 0);
17     //创建服务器套接字地址
18     memset (&serverAddr, 0, sizeof (serverAddr));
19     serverAddr.sin_family = AF_INET;
20     inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
21     serverAddr.sin_port = htons (7);          //我们假设端口是 7
22     //连接
23     connect (sd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
24     //发送和接收数据
25     fgets (sendBuffer, 256, stdin);
26     send (fd, sendBuffer, strlen (sendbuffer), 0);
27     bytesToRecv = strlen (sendbuffer);
28     movePtr = recvBuffer;
29     while ( (n = recv (sd, movePtr, bytesToRecv, 0) ) > 0)
30     {
```

```

31     movePtr = movePtr + n;
32     bytesToRecv = bytesToRecv - n
33 } //while 循环结束
34 recvBuffer[bytesToRecv] = 0;
35 printf ("Received from server.");
36 fputs (recvBuffer, stdout);
37 //关闭并退出
38 close (sd);
39 exit (0);
40 } //结束 echo 客户程序

```

17.1.7 预先定义的客户-服务器应用

因特网有一组已定义的使用了客户-服务器范式的应用程序。它们是一些成熟的程序，能够直接安装和使用。其中，有一些应用是为了提供某种特定的服务而设计的（如 FTP），还有一些是为了允许用户登录到服务器上以执行所需的任务而设计的（如 TELNET），另外还有一些是为了辅助别的应用程序而设计的（如 DNS）。我们将在第 18~24 章中详细讨论这些应用程序。

在附录 F 中，我们给出了表 17.1~17.4 的 Java 版本的简单程序。

17.2 P2P 范式

虽然目前因特网上正在使用的绝大多数应用程序都使用了客户-服务器范式，但是使用称为 P2P 范式 (peer-to-peer paradigm) 的思想最近越来越多地受到人们的关注。在这种范式中，两个对等的计算机（便携机、台式机或大型主机）可以彼此之间互相通信以交换服务。这种范式对某些领域是很有吸引力的，譬如文件传送，当客户想传送一个很大的文件（如音频或视频文件）时，如果使用客户-服务器范式将会给服务器带来很重的负担。另外如果两个对等主机需要不经过服务器来互相交换一些文件或信息，也会对使用 P2P 范式感兴趣。但是，我们必须说明的是，P2P 范式并没有完全抛弃客户-服务器范式。实际上，它所做的只是把服务器的负担让希望加入这一过程的若干个用户共同分担而已。例如，如果有多个客户要下载同一个大文件，服务器可以让每个客户先下载一部分文件，然后客户彼此之间共享，而不是让每个客户都建立一条连接并各自下载这个文件。不过，在下载部分文件或共享已下载的文件的过程中，某一台计算机扮演的是客户的角色，而其他的计算机则扮演服务器的角色。换言之，对某个特定的应用来说，一台计算机在这一时刻可能是一个客户，而在另一时刻又会成为一个服务器。此类应用目前在商业应用上还是受控制的，并没有正式加入到因特网中。我们把对这些应用的介绍与说明留给针对各种具体应用的书籍。

17.3 深入阅读

有几本书全面覆盖了网络编程的内容。特别地，我们推荐[Ste et al. 04]、[Com 96]、[Rob & Rob 96]和[Don & Cal 01]。

17.4 重要术语

客户-服务器范式	套接字接口
接口	流（STREAM）
P2P 范式	运输层接口（TLI）
套接字	

17.5 本章小结

- 在因特网中的绝大多数应用程序都被设计为使用客户-服务器范式，其中有一个应用程序称为服务器，它提供服务，而另外一个应用程序称为客户，它接收服务。服务器程序是一个无限的程序。当它启动后就一直运行下去，除非出现故障。它等待来自客户的请求。客户程序是有限的，也就是说它由用户启动，并在服务完成后终止。客户和服务器都可以运行于并发模式。
- 客户可以在一台主机上顺序地或者并发地运行。顺序服务器一次只处理一个请求。相反地，并发服务器能够同时处理多个请求。
- 客户-服务器范式基于一组预先定义的称为接口的函数。在本章，我们讨论了一些最常用的接口，称为套接字接口。套接字接口就是一个指令集合，它位于操作系统和应用程序之间。套接字是模拟了我们在生活中常见的硬件插口的软件抽象。要想使用通信信道，应用程序（客户或服务器）必须请求操作系统创建一个套接字。
- 进程和操作系统之间的交互是通过一系列预先定义的函数完成的。在本章，我们介绍了这些函数中的一个子集，具体地说，有 `socket`、`bind`、`connect`、`listen`、`accept`、`fork`、`send`、`recv`、`sendto`、`recvfrom` 和 `close`。我们还介绍了一些字节顺序函数和几个内存管理函数。
- 一个服务器程序可以被设计为使用 UDP 服务的无连接的顺序程序，也可以是使用 TCP 或 SCTP 服务的面向连接的并发程序。
- 虽然目前在因特网中，客户-服务器范式是最常见的，但是另外一种称为 P2P 的范式也有不少商业应用。

17.6 实践安排

17.6.1 习题

1. 使用小数在前，给出一个 32 位的数字是如何在四个内存位置（字节 x , $x + 1$, $x + 2$ 和 $x + 3$ ）中保存的。
2. 使用大数在前，给出一个 32 位的数字是如何在四个内存位置（字节 x , $x + 1$, $x + 2$ 和 $x + 3$ ）中保存的。
3. 编写一段小程序来看看你的计算机使用的是小数在前还是大数在前。
4. 在本章中我们使用了几种数据类型。编写一段小程序使用并测试它们。
5. 编写一段小程序来测试一下我们在本章中使用到的内存管理函数。
6. 在 UNIX 环境下，有几个函数可用于读取像 IP 地址和端口地址这样的主机信息，例如，`gethostbyname`, `gethostbyaddress`, `getservebyname` 和 `getservebyport`。试着查找一些关于这些函数的信息，并在一段小程序中使用它们。
7. 在表 17.1 中，请添加检查调用 `socket` 函数是否出现了错误的代码段。如果出现错误，程序需要退出。提示：使用 `perror` 和 `exit` 函数。`perror` 函数的原型如下所示：

```
void perror (const char* string);
```
8. 在表 17.1 中，请添加检查调用 `bind` 函数是否出现错误的代码段。如果出现错误，程序需要退出。提示：使用 `perror` 和 `exit` 函数。
9. 对表 17.2 中的程序进行修改，使之允许客户发送多个数据报。
10. 对图 17.11 中的流程图进行修改，以描绘一个无连接的顺序服务器。

第18章 主机配置：DHCP

本章要讨论我们的第一个客户/服务器应用程序：动态主机配置协议（DHCP）。我们之所以首先讨论这个应用程序，是因为在一台主机启动后，DHCP 是第一个运行的客户/服务器应用程序。换言之，当一台主机启动后，如果它认为自己应当连接到因特网上，但又不知道自己的 IP 地址时，DHCP 就以引导程序的身份发挥作用。

目标

本章有以下几个目标：

- 给出我们需要主机配置的理由。
- 介绍过去曾用于主机配置的两个协议的历史背景。
- 定义 DHCP，它是目前使用的动态主机配置协议。
- 讨论当客户和服务器位于同一个网络或不同网络时的 DHCP 操作。
- 说明 DHCP 如何利用两个 UDP 端口来实现配置。
- 讨论客户向 DHCP 服务器租用一个 IP 地址所要经历的几个状态。

18.1 引言

每一个使用 TCP/IP 协议族的计算机都需要知道自己的 IP 地址。如果这个计算机使用无分类编址或者是一个子网中的成员，那么它还需要知道自己的子网掩码。今天的大多数计算机还需要另外两种信息：一个使之能够与其他网络进行通信的默认路由器的地址，以及一个名字服务器的地址，我们将在下一章看到，名字服务器使我们能够使用名字而不是直接使用地址访问网络。换言之，通常以下四种信息是必要的：

- 计算机的 IP 地址
- 计算机的子网掩码
- 一个路由器的 IP 地址
- 一个名字服务器的 IP 地址

这四种信息可以存储在配置文件中，并在引导过程时访问这个文件。但是，对于无盘工作站或者是有硬盘的计算机在第一次引导时，应当如何处理呢？

在无盘计算机的情况下，操作系统和一些网络软件可以存储在只读存储器（ROM）中。但是，制造商并不知道上述四种信息，因此它们不能被存储在 ROM 中。这些信息与机器的配置相关，并且定义了该机器所连接到的网络。

18.1.1 曾经使用过的协议

在 DHCP 成为正式的主机配置协议之前，还有过一些其他协议，我们在这里简单地介绍一下。

RARP

在因特网时代的初期，人们曾经设计了一个称为逆地址解析协议（Reverse Address Resolution Protocol，RARP）来向被引导的主机提供 IP 地址。实际上，RARP 是我们在第 8 章中所讨论的 ARP 的一个版本。ARP 将一个 IP 地址映射为一个物理地址，而 RARP 则将一个物理地址映射为一个 IP 地址。但是现在 RARP 已经被淘汰了，原因有两个：首先，RARP 利用了数据链路层的广播服务，这也就表示每个网络上都必须存在一台 RARP 服务器。第二，RARP 只能提供计算机的 IP 地址，但如今的计算机需要所有上述四种信息。

BOOTP

引导程序协议（Bootstrap Protocol，BOOTP）是 DHCP 的先驱。它是一个客户/服务器协议，被设计用来克服 RARP 协议存在的两个缺陷。首先，既然它是一个客户/服务器程序，那么 BOOTP 服务器就可以位于因特网的任何地方。其次，它可以提供所有上述四种信息，包括 IP 地址。为了能够提供上述四种信息，BOOTP 打破了 RARP 协议的束缚。但是 BOOTP 是一个静态配置协议。当客户请求自己的 IP 地址时，BOOTP 服务器就咨询一张表，将客户的物理地址映射为相应的 IP 地址。这就暗示着客户的物理地址和 IP 地址之间的绑定是已经存在的。这个绑定关系是事先设定好的。

在某些场合，我们需要的是一个动态配置协议。例如，当一台主机从一个物理网络移动到另一个物理网络时，它的物理地址就改变了。再举一个例子，有时候主机需要在某一段时间内使用一个临时的 IP 地址。BOOTP 无法处理这些状况，因为物理地址和 IP 地址之间的绑定是静态的，是固定存放在一张表中的，除非管理员更改这张表。稍后我们会看到，DHCP 的设计就是为了解决这些不足之处。

DHCP

动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）是一种客户/服务器协议，设计这个协议是为了将上述四种信息提供给无盘计算机或第一次启动的计算机。DHCP 是 BOOTP 的继承者，并且能够兼容 BOOTP。虽然 BOOTP 被认为已经过时，但可能还是有一些系统的主机配置仍然使用的是 BOOTP。在本章中，有一些讨论与 DHCP 的动态特点无关，它们同样适用于 BOOTP。

18.2 DHCP 操作

DHCP 客户和 DHCP 服务器可以在同一个网络上，也可以位于不同的网络。让我们分别讨论这两种情况。

18.2.1 同一个网络

虽然这种情况不是很常见，不过管理员可以把客户和服务器放在同一个网络中，如图 18.1 所示。

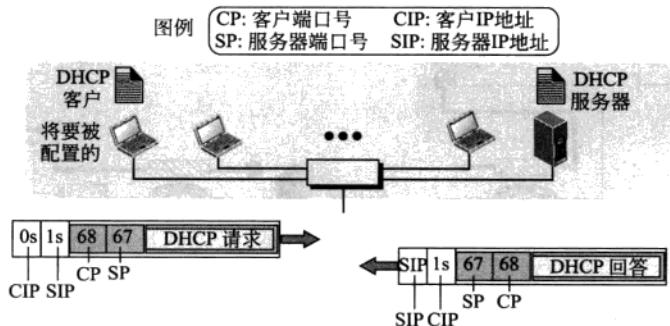


图 18.1 客户和服务器在同一个网络上

这种情况的操作如下：

1. DHCP 服务器在 UDP 端口 67 发出被动打开命令，等待客户请求。
2. 被引导的客户在 UDP 端口 68（以后要解释这个端口号）发出主动打开命令。这个报文被封装成 UDP 用户数据报，其目的端口号是 67，源端口号是 68。这个 UDP 用户数据报再封装成 IP 数据报。读者可能会问：客户既不知道它自己的 IP 地址（源地址），也不知道服务器的 IP 地址（目的地址），它怎么能够发送 IP 数据报呢？客户使用的是全 0 的源地址和全 1 的目的地址。
3. 服务器或者用广播报文，或者用单播报文来响应这个客户，它使用了 UDP 源端口 67 和目的端口 68。这个响应可以是单播的，因为服务器知道客户的 IP 地址，同时也知道客户的物理地址，也就是说它不需要使用 ARP 的服务进行从逻辑地址到物理地址的映射。但是某些系统不允许旁路掉 ARP，结果就要使用广播地址。

18.2.2 不同的网络

像其他应用层的进程一样，客户可以在某个网络上，而服务器可以在相隔好几个网络之外的另一个网络上。图 18.2 所示为这种情况。

但是，这就带来了一个必须要解决的问题。DHCP 请求是广播发送的，因为客户不知道服务器的 IP 地址。而广播的 IP 数据报不能通过任何路由器。路由器收到这样的分组就丢弃它。回想一下，全 1 的 IP 地址是一个受限广播地址。

要解决这个问题，就需要一个中介物。某台主机（或是一台能够配置为在应用层工作的路由器）可用来充当中继。在这种情况下，该主机就称为中继代理（relay agent）。中继代理知道 DHCP 服务器的单播地址，并在端口 67 监听广播报文。当它收到这种类型的分

组后，就把它封装成一个单播数据报，并且把此请求发送给 DHCP 服务器。携带了单播目的地址的分组可以被任何一个路由器转发，最终到达 DHCP 服务器。DHCP 服务器知道这个报文来自中继代理，因为在请求报文中有一个字段定义了中继代理的 IP 地址。中继代理在收到回答后，再把它发送给 DHCP 客户。

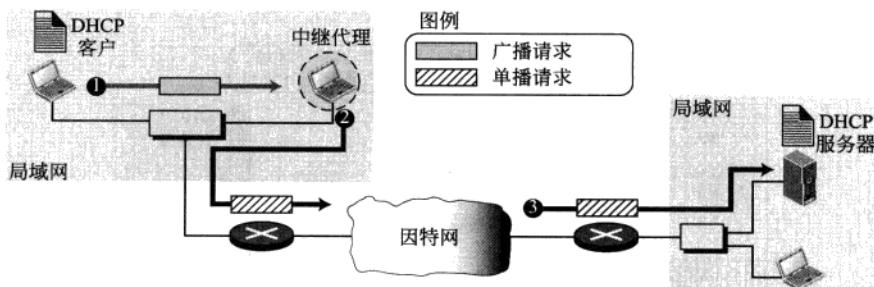


图 18.2 客户和服务器在不同的网络上

18.2.3 UDP 端口

图 18.3 所示为客户端和 DHCP 服务器的交互。服务器使用熟知端口 67，这是对 UDP 端口使用的正常情况。客户使用熟知端口 68，这是一种 UDP 端口使用的非正常情况。客户选择了熟知端口，而不是临时端口的原因，是为了防止在服务器使用广播进行回答时可能会出现的一个问题。为了更好地理解这个问题，我们分析使用临时端口的情况。假定网络上的主机 A 正在临时端口 2017（随机地选择）使用 DHCP 客户。在同一个网络上的主机 B 正在临时端口 2017（碰巧相同了）使用 DAYTIME 客户。

现在 DHCP 服务器广播发送回答报文，其目的端口号是 2017，而广播 IP 地址是 FFFFFFFFFF_{16} 。每个主机都要打开携带这个目的 IP 地址的分组。主机 A 发现在临时端口 2017 上有从应用程序来的报文，于是正确的报文交付给 DHCP 客户。但不正确的报文也交付给 DAYTIME 客户。这种混乱的原因是由基于套接字地址（见第 17 章）的分组分用造成的。套接字是 IP 地址和端口号的组合。在这种情况下，这两个客户都是一样的套接字。

使用熟知端口（小于 1024）能够防止两个相同目的端口号的使用。主机 B 不能选择 68 作为它的临时端口号，因为临时端口号必须大于 1023。

好奇的读者可能会问：若主机 B 上也运行 DHCP 客户，那么将会发生什么？在这种情况下，两个套接字地址是一样的，两个客户都会收到同样的报文。此时就需要第三个标识来区分不同的客户。DHCP 使用了另外一个编号，称为事务标识（transaction ID），

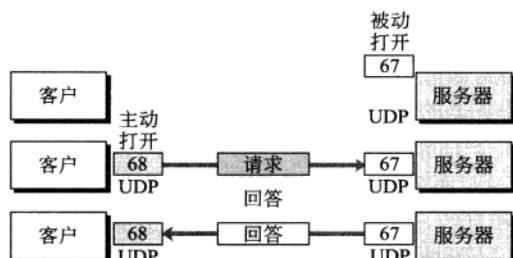


图 18.3 使用 UDP 端口

它由涉及 DHCP 的每一条连接随机选择。两个主机在同一时间选择了相同的事务标识的可能性相当渺小。

18.2.4 使用 TFTP

服务器并没有把客户为了引导计算机而可能需要的所有信息都发送过去，而是在回答报文中定义了文件的路径名，客户可从这里找到完整的引导信息。于是，客户可以使用封装成 UDP 用户数据报的 TFTP 报文（见第 21 章），来获取所需的其余信息。

18.2.5 差错控制

如果请求报文丢失了或受到损伤，该怎么办？如果响应报文受到损伤，又该怎么办？在使用 DHCP 时需要有差错控制。DHCP 使用了 UDP，而 UDP 不提供差错控制。因此，DHCP 必须提供差错控制。差错控制是通过以下两种策略完成的：

1. DHCP 要求 UDP 使用检验和。请记住，在 UDP 中，检验和的使用是可选的。

2. DHCP 客户使用计时器和重传策略来解决对发出的请求没有收到回答的问题。但是，为了防止因多台主机都需要对请求进行重传而出现的通信量堵塞（例如，在电源出故障之后），DHCP 强制规定客户在设置计时器时要使用随机数。

18.2.6 分组格式

图 18.4 给出了 DHCP 分组的格式。

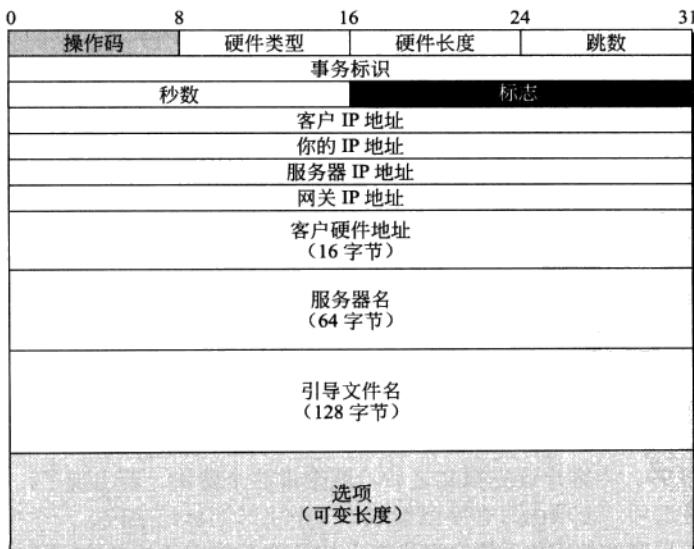


图 18.4 DHCP 分组格式

各字段简单介绍如下：

- 操作码** 这个 8 位字段定义了 DHCP 分组的类型：请求（1）或回答（2）。
- 硬件类型** 这个 8 位字段定义了物理网络的类型。每一种网络类型都指派有一个整数值。例如，对于以太网这个值是 1。
- 硬件长度** 这个 8 位字段定义了以字节为单位的物理地址的长度。例如，对以太网这个值是 6。
- 跳数** 这个 8 位字段定义分组可经历的最大跳数。
- 事务标识** 这个 4 字节字段携带了一个整数。事务标识由客户设置，用来对回答和请求进行匹配。服务器要在回答中返回同样的值。
- 秒数** 这个 16 位字段指出从客户开始引导算起一共经历的秒数。
- 标志** 这个 16 位字段只使用了它的最左边一位，其余位都应当置 0。最左边的一位指明服务器强制使用广播回答（而不是单播）。如果用单播发送回答给客户，则 IP 分组的目的 IP 地址就是指派给客户的地址。因为客户不知道自己的 IP 地址，它可能会丢弃这个分组。但若 IP 数据报是广播发送的，则每一个主机都要接收并处理这个广播报文。图 18.5 描绘了这个标志格式。
- 客户 IP 地址** 这个 4 字节字段包含客户 IP 地址。若客户没有这个信息，则这个字段的值是 0。
- 你的 IP 地址** 这个 4 字节字段包含客户 IP 地址。这是服务器（在回答报文中）在客户的请求下填入的。
- 服务器 IP 地址** 这个 4 字节字段包含服务器 IP 地址。它由服务器在回答报文中填入。
- 网关 IP 地址** 这个 4 字节字段包含路由器的 IP 地址。这是服务器在回答报文中填入的。
- 客户硬件地址** 这是客户的物理地址。虽然服务器可以从客户发送的帧中读取这个地址，但若由客户在请求报文中显式地提供这个地址，则更加有效。
- 服务器名** 这个 64 字节字段由服务器在回答分组中可选地填入。它包含了以空字符串为结尾的字符串，这个字符串由服务器的域名构成。如果服务器不想在这个字段中填入数据，则必须全部填 0。
- 引导文件名** 这个 128 字节的字段由服务器在回答分组中可选地填入。它包含了以空字符串为结尾的字符串，这个字符串由引导文件的全路径名构成。客户可以使用这个路径来读取其他引导信息。如果服务器不想在这个字段中填入数据，则必须全部填 0。
- 选项** 这个 64 字节字段有双重作用。它可以携带附加信息（如网络掩码或默认路由器地址），也可以携带某些厂商特定的信息。这个字段只用在回答报文中。服务器使用了一个称为魔块（magic cookie）的数字，这个数字的 IP 地址格式的值为 99.130.83.99。当客户读完报文之后，就搜索这个魔块。若出现了，则接下来的 60 字节就是选项。选项由三个字段组成：一个字节的标记字段、一个字节的长度字段以及可变长度的值字段。长度字段定义的是这个值字段的长度而不是整个选项的长度，见图 18.6。

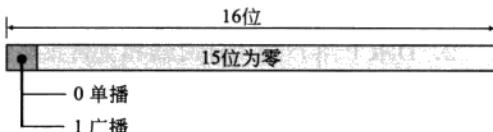


图 18.5 标志格式

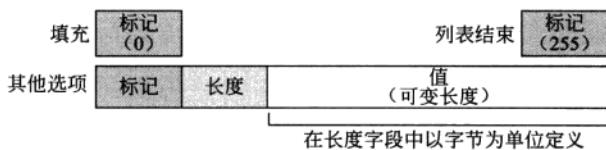


图 18.6 选项格式

选项的列表如表 18.1 所示。

表 18.1 DHCP 的选项

标记	长度	值	说明
0			填充
1	4	子网掩码	子网掩码
2	4	当天的时间	时间偏移
3	可变	IP 地址	默认路由器
4	可变	IP 地址	时间服务器
5	可变	IP 地址	IEN 16 服务器
6	可变	IP 地址	DNS 服务器
7	可变	IP 地址	Log 服务器
8	可变	IP 地址	Quote 服务器
9	可变	IP 地址	打印服务器
10	可变	IP 地址	Impress
11	可变	IP 地址	RLP 服务器
12	可变	DNS 名	主机名
13	2	整数	引导文件大小
53	1	稍后讨论	用于动态配置
128~254	可变	特定信息	厂商相关
255			列表结束

包含 IP 地址的字段长度是 4 字节的倍数。填充选项只有 1 个字节长，做对齐之用。列表结束选项也是只有 1 个字节长，它指出选项字段的结束。厂商可在回答报文中使用选项标记 128~254 来提供额外的信息。

18.3 配置

人们设计 DHCP 是为了提供静态的和动态的地址分配。

18.3.1 静态地址分配

对于此功能，DHCP 有一个专门的数据库，可以静态地把物理地址绑定到 IP 地址。在这种方式工作时，DHCP 与已过时的协议 BOOTP（前面已介绍过）向后兼容。

18.3.2 动态地址分配

DHCP 还有第二个数据库，包括一个可用的 IP 地址池。第二个数据库使 DHCP 成为动态的。当 DHCP 客户请求临时的 IP 地址时，DHCP 服务器就从可用（即未使用的）IP 地址池中取出一个 IP 地址进行指派，这个 IP 地址的使用时间长短可协商。

当 DHCP 客户向 DHCP 服务器发送请求时，服务器首先检查它的静态数据库。若静态数据库中存在所请求物理地址的表项，则返回这个客户的永久 IP 地址。反之，若静态数据库中没有这个表项，服务器就从可用 IP 地址池中选择一个 IP 地址，并把这个地址指派给客户，然后再把相应的表项加入到动态数据库中。

如果主机要从一个网络移动到另一个网络，或者与一个网络时连时断（就像用户和服务提供者的关系），那么 DHCP 的这种动态特性就有了用武之地。DHCP 可以在有限时间内提供一个临时的 IP 地址。

从地址池指派的地址都是临时地址。DHCP 服务器向客户授予在某一段时间内对该地址的租用权。当租用时效过期，客户或者停止使用这个 IP 地址，或者续租。服务器有权力选择同意或不同意续租。若服务器不同意，客户就停止使用这个地址。

18.3.3 转换状态

为了提供动态的地址分配，DHCP 客户可以像状态机那样从一个状态转换到另一个状态，状态转换取决于收到的报文和发送的报文。在这种情况下，报文的类型是由包含在 DHCP 分组中的标记为 53 的选项来定义的。换言之，设计者们决定不是在 BOOTP 协议的基础上增加一个新的字段，而是增加一个新的选项来实现这个目的。图 18.7 描绘了这个类型选项以及对它的值的解释，这些值分别对应各种 DHCP 分组类型。

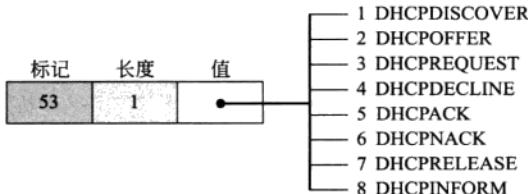


图 18.7 标记为 53 的选项

图 18.8 所示为它的一些主要状态的状态转换图。在 RFC 和一些 DHCP 的实现中可能提供了更多的状态，我们把对这些状态的研究留作练习。

INIT 状态

当 DHCP 客户首次启动时，它处于 INIT 状态（初始化状态）。客户使用端口 67 广播 DHCPDISCOVER 报文（一个带有 DHCPDISCOVER 选项的请求报文）。

SELECTING 状态

在发送了 DHCPDISCOVER 报文后，客户就进入 SELECTING（选择）状态。能够提供这种类型服务的服务器要用 DHCPOFFER 报文进行响应。在此类报文中，服务器提供了一个 IP 地址。它们还要提供租用时间长度，其默认值是 1 小时。发送 DHCPOFFER 报文的服务器，把提供的 IP 地址锁定，使这个地址不会再提供给任何其他的客户。客户选择所提供的地址中的一个，并向所选择的服务器发送 DHCPREQUEST 报文。然后就进入

REQUESTING 状态。如果客户没有收到 DHCPOFFER 报文，它还要再尝试四次，每一次间隔 2 秒。如果对这些 DHCPDISCOVER 都没有收到回答，客户就睡眠 5 分钟后再试。

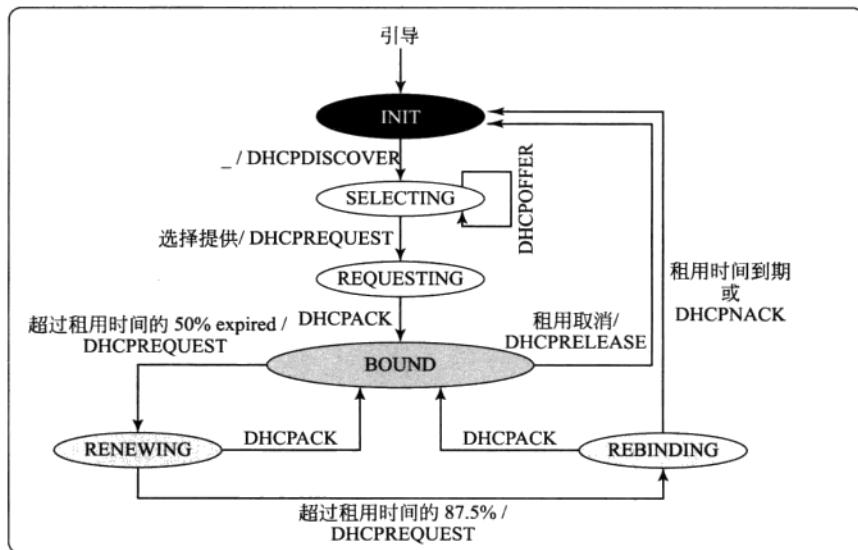


图 18.8 DHCP 客户转换图

REQUESTING 状态

客户保持在这个 REQUESTING（请求）状态，直至它收到来自服务器的 DHCPACK 报文为止，这个报文创建了客户物理地址和它的 IP 地址之间的绑定。客户收到 DHCPACK 报文后进入 BOUND 状态。

BOUND 状态

在这种状态下，客户可以使用该 IP 地址，直到租用时间到期。当到达租用时间的 50% 时，客户就再发送一个 DHCPREQUEST 报文以请求更新。于是，客户进入 RENEWING 状态。在 BOUND（绑定）状态时，客户也可以取消租用，并进入到初始化状态。

RENEWING 状态

客户保持在 RENEWING（更新）状态，直至下面两个事件之一发生。客户可以收到更新租用协定的 DHCPACK 报文。在这种情况下，客户把计时器复位，然后回到 BOUND 状态。或者，如果没有收到 DHCPACK，同时到达租用时间的 87.5% 时，那么客户就进入 REBINDING 状态。

REBINDING 状态

客户保持在 REBINDING（重新绑定）状态，直至下面三个事件之一发生。若客户收到一个 DHCPNACK 报文或者租用时间到期，则回到初始化状态，并尝试得到另一个 IP 地址。若客户收到 DHCPACK 报文，它就进入到绑定状态，并把计时器复位。

18.3.4 其他

在这一小节，我们要讨论几个与 DHCP 状态有关的话题。

提前释放

在某一段时间内被指派了地址的 DHCP 客户，可以在租用时间到期之前释放该地址。客户可以通过发送一个 DHCPRELEASE 报文告诉服务器它不再需要这个地址了。这样做有助于服务器向其他正在等待地址的客户分配该地址。

计时器

以上的讨论要求客户使用三个计时器：更新计时器、重新绑定计时器和超时计时器。如果服务器在分配地址时没有指明这几个计时器的超时时间值，那么客户就需要使用默认值。每个计时器的默认值如下：

更新计时器： → 租用时间的 50%

重新绑定计时器： → 租用时间的 87.5%

超时计时器： → 租用时间的 100%

18.3.5 交换报文

图 18.9 给出了与上面的状态转换图相关的报文交换过程。

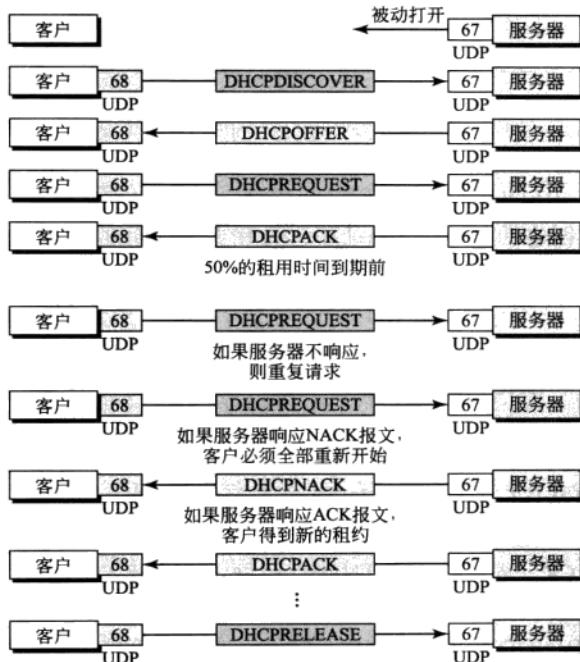


图 18.9 交换报文

18.4 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书

目可以在本书末尾的参考书目清单中找到。

18.4.1 参考书和 RFC

有几本参考书和 RFC 对 DHCP 做了简单但是全面的阐述，包括[Com 06]、RFC 3396 和 RFC 3342。

18.5 重要术语

引导程序协议（BOOTP）

魔块（magic cookie）

动态主机配置协议（DHCP）

中继代理

租用

18.6 本章小结

- 每个连接到 TCP/IP 互联网的计算机都必须知道自己的 IP 地址、一个路由器的 IP 地址、一个名字服务器的 IP 地址以及自己的子网掩码。动态主机配置协议（DHCP）是客户–服务器应用程序，用来把重要的网络信息传递给无盘计算机或首次引导时的计算机。
- 客户请求和服务器回答都使用相同的 DHCP 分组格式。DHCP 服务器被动地等待客户的请求。服务器的回答可以采用广播或单播。DHCP 的请求或回答都被封装在 UDP 用户数据报里。
- 当 DHCP 客户与服务器分别位于不同网络上时，需要使用中继代理，它把来自客户的本地 DHCP 请求发送给远程服务器。
- 当 DHCP 作为静态配置协议时，它利用表格把 IP 地址映射为物理地址。而当 DHCP 作为动态配置协议时，它把 IP 地址租给请求的客户使用。
- DHCP 客户被设计为使用了六个主要状态和三个计时器的状态机，它使得主机能够在一段指定的时间内租用 IP 地址。

18.7 实践安排

18.7.1 习题

1. DHCP 分组的最小长度是多少？最大长度是多少？
2. DHCP 分组封装在 UDP 分组中，再封装成 IP 分组，然后又封装成以太网的帧。试计算在不使用任何选项时 DHCP 分组的效率。此处的效率是指 DHCP 分组中的字节数与它

在数据链路层所传输的总字节数之比。

3. 试给出具有填充选项的 DHCP 分组的例子。
4. 试给出具有列表结束选项的 DHCP 分组的例子。
5. 在 DHCP 分组中的秒数字段中可存储的最大秒数是多少？
6. a. 试给出物理地址为 00:11:21:15:EA:21 的客户所发送的 DHCP 请求分组的所有字段的内容。
 - b. 把 a 题中的分组封装在 UDP 用户数据报中，请填写所有的字段。
 - c. 把 b 题中的分组封装在 IP 数据报中，请填写所有的字段。
 - d. 给出对 a 题中的请求进行响应的 DHCP 回答分组的所有字段的内容。
 - e. 把 d 题中的分组封装在 UDP 用户数据报中，请填写所有的字段。
 - f. 把 e 题中的分组封装在 IP 数据报中，请填写所有的字段。
7. 为什么新增加的主机必须知道自己的子网掩码？
8. 为什么新增加的主机必须知道一个路由器的 IP 地址？
9. 为什么新增加的主机必须知道一个名字服务器的 IP 地址？
10. 你认为为什么 DHCP 需要利用 TFTP 获取附加信息？为什么不能用 DHCP 来读取所有的信息？
11. 有一个 C 类以太网上的无盘客户使用了 DHCP。DHCP 服务器在一个 B 类以太网上。试画出该网络图，包括客户、服务器和中继代理，并为它们标出适当的 IP 地址。填写出一个 DHCP 请求分组和一个回答分组。

18.7.2 研究活动

12. 试给出 DHCPDISCOVER 报文的格式和内容。
13. 试给出 DHCPOFFER 报文的格式和内容。
14. 试给出 DHCPREQUEST 报文的格式和内容。
15. 试给出 DHCPDECLINE 报文的格式和内容。
16. 试给出 DHCPACK 报文的格式和内容。
17. 试给出 DHCPNACK 报文的格式和内容。
18. 试给出 DHCPRELEASE 报文的格式和内容。
19. 试找出用于设置对丢失的 DHCP 分组进行重传控制的计时器的随机数的取值范围。
20. 试找出客户在重传请求时是否有次数限制。
21. 试针对 DHCP 的安全性进行研究。

第 19 章 域名系统 (DNS)

本章我们要讨论第二个应用程序：域名系统 (DNS)。DNS 是一个用来辅助其他应用程序的客户/服务器应用程序。DNS 的作用就是把应用层的主机名映射为网络层的 IP 地址。

目标

本章有以下几个目标：

- 阐述 DNS 的作用。
- 定义域和域名空间的概念。
- 阐述名字空间的分布并定义区的概念。
- 讨论在因特网中 DNS 的使用，并描述三种域：类属域、国家域和反向域。
- 讨论名字-地址的解析，并说明两种解析方法：递归解析和迭代解析。
- 描述 DNS 报文的格式并说明它们如何被压缩。
- 讨论 DDNS 和 DNSSEC。

19.1 DNS 的必要性

为了标志一个实体，TCP/IP 协议使用的是 IP 地址，它能够唯一地标志连接到因特网的一台主机。但是，人们更愿意使用的却是名字，而不是地址。因此，我们需要有一种系统能够把名字映射为地址，或者把地址映射为名字。

在因特网的规模还很小的时候，映射是通过主机文件 (host file) 来实现的。在这个主机文件中仅包含名字和地址这两列。每台主机都可以把主机文件存储在自己的磁盘上，并根据一个总主机文件进行定期更新。当程序或用户需要把名字映射为地址时，主机就可以查找主机文件并找出相应的映射。

但是时至今日，已经不可能用一个单独的主机文件把所有的地址和名字关联起来。主机文件会因为太大而无法存储在每台主机上。另外，每当映射关系发生变化时，也不可能对全世界所有的主机文件都进行更新。

一种解决方法似乎可以把一个完整的主机文件存储在某一台计算机中，然后让每个需要映射的计算机都来访问这个集中管理的信息。但是我们知道，这样做会在因特网上产生非常大的通信量。

另一种解决方法就是目前所采用的，它把这个巨大的信息量划分为若干个较小的组成

部分，并把每一部分存储在不同的计算机上。使用这种方法时，需要映射的计算机可以寻找持有所需信息的最近的计算机。这种方法正是域名系统（Domain Name System, DNS）所使用的。在本章我们先讨论 DNS 的概念及其隐含的思想，然后再描述 DNS 协议本身。

图 19.1 描绘了 TCP/IP 是如何通过 DNS 客户和 DNS 服务器把名字映射为地址的。反之亦然。

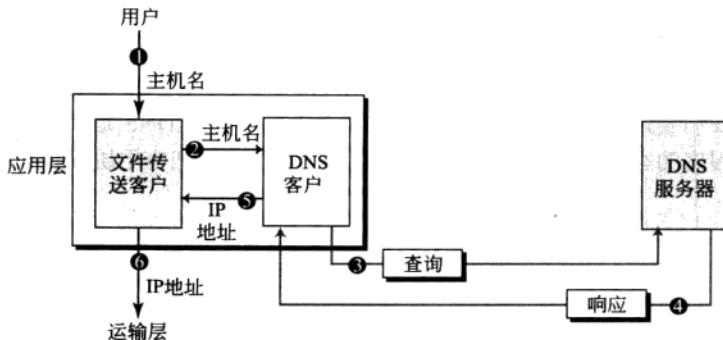


图 19.1 DNS 的作用

在图 19.1 中，某个用户希望通过文件传送客户程序来访问运行在一台远程主机上的文件传送服务器程序。用户只知道该文件传送服务器的名字，如 `forouzan.com`。但是，TCP/IP 协议族需要文件传送服务器的 IP 地址来建立连接。通过以下六个步骤可以将主机名映射为 IP 地址。

1. 用户把主机名传递给文件传送客户进程。
2. 文件传送客户进程把这个主机名传递给 DNS 客户进程。
3. 根据第 18 章，我们知道每台计算机在引导后都可获知一个 DNS 服务器的地址。

DNS 客户利用已知的 DNS 服务器的 IP 地址，向该 DNS 服务器发送报文，这个报文是指定了文件传送服务器名的查询报文。

4. DNS 服务器用所需的文件传送服务器的 IP 地址来进行响应。
5. DNS 客户把这个 IP 地址交付给文件传送客户进程。
6. 现在，文件传送客户进程就可以利用收到的 IP 地址来访问文件传送服务器了。

请注意，接入因特网的目的是为了在文件传送客户和服务器之间建立连接，但是在此之前，首先要在 DNS 客户和 DNS 服务器之间建立另一条连接。换言之，我们需要两条连接：第一条连接是为了把名字映射为 IP 地址。第二条连接才是为了传送文件（假设）。

19.2 名字空间

为了无二义性，指派给机器的名字必须从名字空间中小心地选择，而这个名字空间对名字和 IP 地址的绑定有着完全的控制权。换言之，因为地址是唯一的，因此名字也必须是唯一的。把每个地址映射为独一无二的名字的名字空间（name space）可以通过两种方法来组织，平面的和层次的。

19.2.1 平面名字空间

在平面名字空间 (flat name space) 中，每一个名字被指派给一个地址。在这个空间中的名字是无结构的字符序列。这些名字有没有共同部分都可以，即使有，也不具有任何意义。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发生重复，因而不能用在像因特网这样的大型系统中。

19.2.2 层次名字空间

在层次名字空间 (hierarchical name space) 中，每个名字都由几部分组成。第一部分可以定义组织的性质，第二部分可以定义组织的名字，第三部分可以定义组织中的各个部门，等等。在这种情况下，指派和控制名字空间的管理机构就可以分散化。中央管理机构可以仅指派定义了组织性质和组织名称的前两部分，而剩余部分则可交给该组织自己去负责管理。该组织可以给名字加上后缀（或前缀）来定义它的主机或其他一些资源。该组织的管理者不必担心自己为主机选择的后缀会被另一个组织也采用了，因为即使地址中有一部分是相同的，整个的地址还是不同的。例如，假定有两个学院和一个公司都把它们的计算机取名为 challenger。中央管理机构给第一个学院指派的名字是 fhda.edu，给第二个学院的名字是 berkeley.edu，给该公司的名字是 smart.com。当这些组织分别在已给定的名字的基础上，再添加上计算机名 challenger 后，最终得到的是三个完全可以区分的名字：challenger.fhda.edu、challenger.berkeley.edu 和 challenger.smart.com。这些名字都是唯一的，不需要由中央管理机构来指派。中央管理机构仅控制名字的一部分而不是全部。

19.2.3 域名空间

为了得到层次化的名字空间，人们设计了域名空间 (domain name space)。在这种设计中，名字被定义在根置于顶部的倒置树的结构中。这个树最多只能有 128 个级：从第 0 级（根）到第 127 级（见图 19.2）。

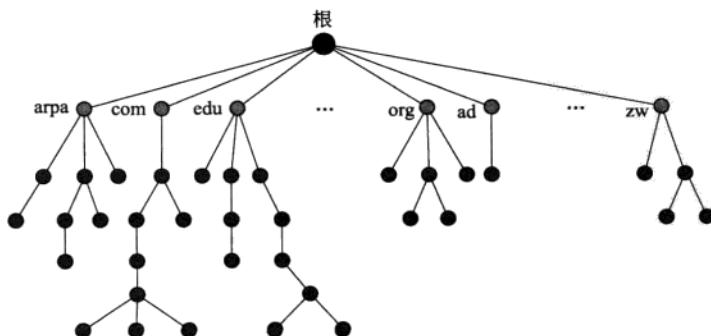


图 19.2 域名空间

标号

树上的每一个结点都有一个标号 (label)，即最多有 63 个字符的字符串。根标号是空字符串 (即没有字符串)。DNS 要求结点的所有子结点 (从同一个结点分支出来的结点) 都具有不同的标号，这就保证了域名的唯一性。

域名

树上的每一个结点都有一个域名。一个完全的域名 (domain name) 是用点 (.) 分隔开的标号序列。域名总是从结点向上读到根，最后一个标号是根标号 (空)。这就表示一个完全的域名总是以空标号结束，也就是说它的最后一个字符是一个点，因为空字符串表示什么也没有。图 19.3 给出了某些域名示例。

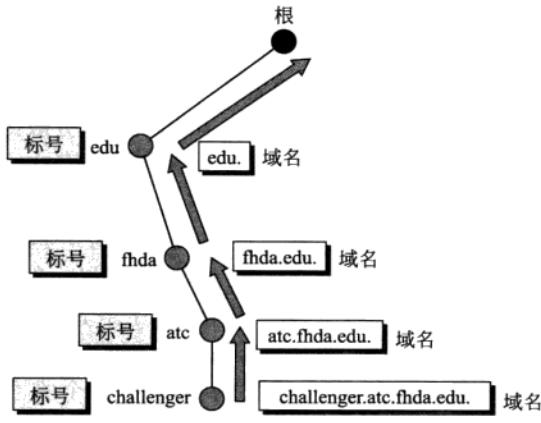


图 19.3 域名和标号

完整域名 (FQDN) 若标号以空字符串结束，就称为完整域名 (Fully Qualified Domain Name, FQDN) (译者注：FQDN 的意思是“完全合格的域名”)。FQDN 是包含主机全名的域名。它包括从最具体的到最一般的所有标号，并唯一地定义了该主机的名字。例如，安装在 De Anza 学院的 ATC (先进技术) 中心的一台名为 challenger 的计算机的域名就是一个完整域名。DNS 服务器只能将一个 FQDN 映射为一个地址。请注意：这种域名必须以空标号结束，但是因为空标号就是什么也没有，所以这种标号以一个点 (.) 来结束。

challenger.atc.fhda.edu.

不完整域名 (PQDN) 若标号不是以空字符串结束，则称为不完整域名 (Partially qualified domain name, PQDN) (译者注：PQDN 的意思是“部分合格的域名”)。PQDN 从一个结点开始，但并没有到达根。它使用的情况是当这个要被解析的名字和客户属于同一个站点。此时，解析程序可以补充缺少的部分，称为后缀，来创建 FQDN。例如，如果 fhda.edu 站点上的用户想获取计算机 challenger 的 IP 地址，用户就可以定义一个不完整的名字：

challenger

DNS 客户程序在把这个地址传递给 DNS 服务器之前，会添加上后缀 atc.fhda.edu。

DNS 客户程序通常保存了一份有关后缀的列表。下面的是 De Anza 学院的后缀列表。空后缀表示什么也没有。当用户定义了 FQDN 时就加上这个后缀。

Atc.fhda.edu fhda.edu 空

图19.4给出了一些FQDN和PQDN。

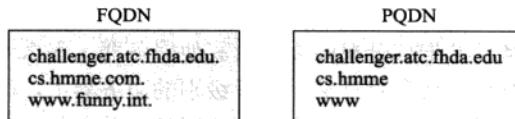


图19.4 FQDN和PQDN

19.2.4 域

域(domain)是域名空间中的子树。域的名字就是这个子树顶部结点的名字。图19.5给出了一些域。请注意，域本身又可划分为若干个域(有时也称它们为子域)。

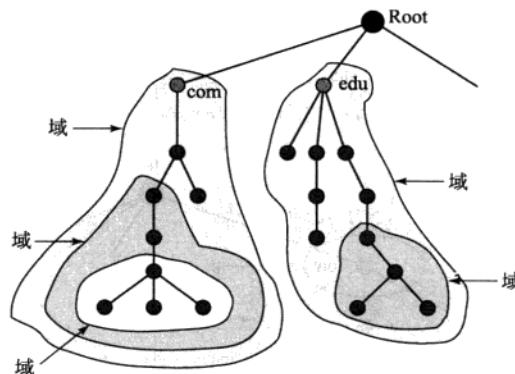


图19.5 域

19.2.5 域名空间的分布

域名空间所包含的这些信息必须被保存起来。但是，若只用一台计算机来存储这样大的信息量是非常低效和不可靠的。说它低效，因为要回答从全世界发来的请求将会造成非常大的负荷。说它不可靠，因为任何一个故障都会导致所有数据无法使用。

名字服务器的层次结构

解决这些问题的方法是把信息分布到多台计算机上，这些计算机称为**DNS服务器(DNS servers)**。一种做法是把整个空间划分为许多基于第一级的域。换言之，我们让根保持不动，然后创建与第一级结点数目一样多的域(子树)。由于用这种方法创建出来的域还是很大，因此DNS允许这些域再进一步划分为更小的域(子域)。每一个服务器或者对一个大的域负责，或者对一个小的域负责(授权的**authoritative**)。换言之，我们有分级的服务器，这与我们有分级的名字是相似的(见图19.6)。

区

因为完整的域名体系结构不能存储在一个单独的服务器上，所以它们分散地存储在若干个服务器上。一个服务器负责的范围，或者说有管理权限的范围，就称为区(zone)。我们可以把区定义为整个树中的一个相互邻接的部分。若服务器对某个域负责，而且这个域

没有再被划分为一些更小的域，那么“域”和“区”指的是同一件事。服务器中有一个数据库，称为区文件，它保留了这个域里面的每个结点的所有信息。但是，如果服务器把自己的域又划分为一些子域，并把部分权限委托给其他服务器，那么“域”和“区”就有了区别。在子域中的各个结点的信息就存储在较低级别的服务器上，而原来的服务器则保存着到这些较低级别的服务器的某种引用。当然，原服务器并不是完全没有责任了，它仍然有一个区，只是把详细的信息保留在较低级别的一些服务器上（见图 19.7）。

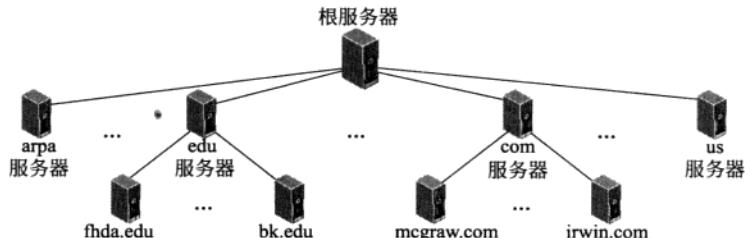


图 19.6 分级的名字服务器

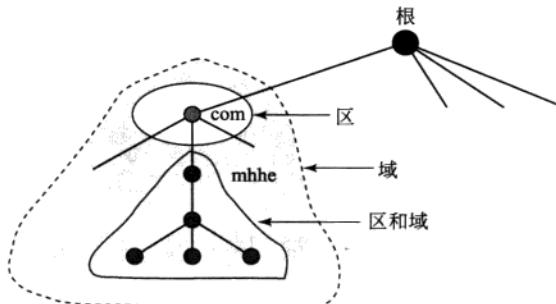


图 19.7 区和域

服务器也可以仅把它的域的一部分进行划分并委托责任，而自己仍保留这个域的另一部分。在这种情况下，它的区的构成包括了没有被委托而具有详细信息的那部分域，再加上对被委托的那些部分的引用。

根服务器

根服务器 (root server) 是这样的服务器，它的区包括了整个树。根服务器通常不存储关于这些域的任何信息，而是把它的权限委托给其他一些服务器，并保存到这些服务器的引用。根服务器有多个，每个根服务器都覆盖了整个域名空间。这些根服务器分布在世界各地。

主服务器和次服务器

DNS 定义了两种类型的服务器：主服务器和次服务器。**主服务器** (primary server) 存储了它所管辖的区的文件。它负责创建、维护和更新这个区文件。它把这个区文件存储在本地磁盘上。

次服务器 (secondary server) 上的关于区的全部信息都是从另一个服务器（主服务器或次服务器）传送过来的，它把这个区文件存储在自己的本地磁盘中。次服务器既不创建也不更新区文件。若需要更新，就必须由主服务器来进行，主服务器再把已更新的版本传送给次服务器。

主服务器和次服务器对它们所服务的区都具有管理权限。其思想并不是说要给次服务器以较低的权限级别，而是想要创建数据的冗余度，从而当某个服务器出现故障时，另一个服务器可以继续对客户提供服务。请注意，一个服务器可以是某个区的主服务器，同时又是另一个区的次服务器。正因如此，当我们提到某服务器是主服务器或次服务器时，我们必须注意所指的是哪一个区。

主服务器从磁盘文件装入所有的信息，次服务器则从主服务器装入所有的信息。当次服务器从主服务器下载信息时，就称为区传送（zone transfer）。

19.3 因特网中的 DNS

DNS 是能够在不同平台上使用的协议。在因特网中，域名空间（树）被划分为三个不同的部分：类属域、国家域和反向域（见图 19.8）。

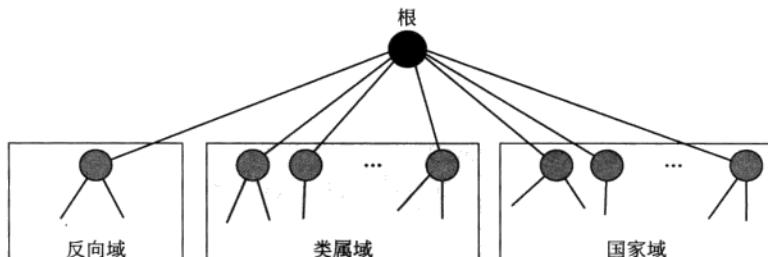


图 19.8 因特网中使用的 DNS

19.3.1 类属域

类属域（generic domains）按照主机的类属行为来定义注册主机。树中的每一个结点定义一个域，它是到域名空间数据库的索引（见图 19.9）。

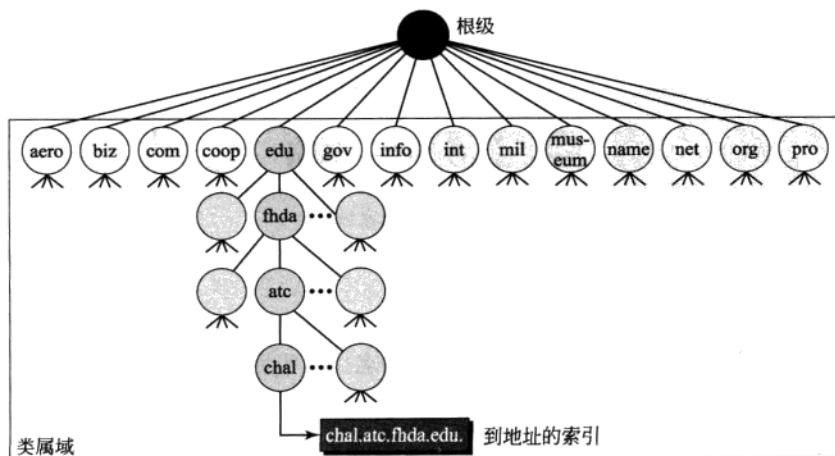


图 19.9 类属域

通过观察这个树，我们看到在类属域的第一级允许有 14 个可能的标号。这些标号描述如表 19.1 中所列举的机构类型。

表 19.1 类属域的标号

标号	说明	标号	说明
aero	航空和航天公司	int	国际机构
biz	企业或商行（与“com”相似）	mil	军事机构
com	商业机构	museum	博物馆和其他非盈利组织
coop	协作的企业组织	name	个人姓名（个人的）
edu	教育机构	net	网络支持中心
gov	政府机构	org	非盈利机构
info	信息服务提供者	pro	专业个体组织

19.3.2 国家域

国家域 (country domain) 这部分使用两字符的国家或地区的缩写（例如，用 us 代表美国）。第二级标号是由国家指定的，它们可以是组织的，或者还可以更具体些。例如，美国使用州的缩写作为子划分（例如 ca.us.）。

图 19.10 给出了国家域的部分。地址 anza.cup.ca.us 可翻译为在美国加州的 Cupertino 的 De Anza 学院。

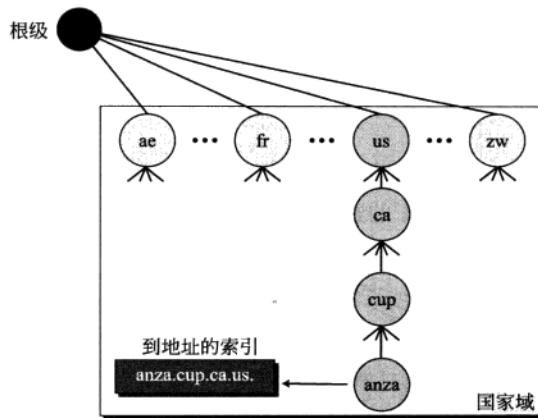


图 19.10 国家域

19.3.3 反向域

反向域 (inverse domain) 用于把地址映射为名字。当服务器收到来自客户的请求要完成某项任务时，有可能需要把地址映射为名称。例如，虽然这个服务器有一个文件，其中包含了授权客户的列表，但它只列出了客户的 IP 地址（从收到的 IP 分组中提取出来的）。为了确定这个客户是否在授权列表中，服务器可以请它的解析程序向 DNS 服务器发送查

询，以请求把地址映射为名字。

这种类型的查询称为反向查询或指针 (PTR) 查询。为了处理指针查询，在域名空间中增加了一个反向域，并且它的第一级结点称为 arpa (由于历史的原因)。第二级也是单个结点，称为 in-addr (表示反向地址)。域的其余部分定义 IP 地址。

处理反向域的服务器也是分级的。也就是说地址的 netid 部分要比 subnetid 部分的等级高，而 subnetid 部分要比 hostid 部分的等级高。按这种方式，为整个地区服务的服务器要比只为每个子网服务的服务器的等级高。与类属域和国家域不同，这种配置使得整个域看起来是反过来的。这种域标号的读法是从底向顶的，按照这个规则，IP 地址 132.34.45.121 (netid 为 132.34 的 B 类地址) 应被读为 121.45.34.132.in-addr.arpa。图 19.11 所示为反向域的配置。

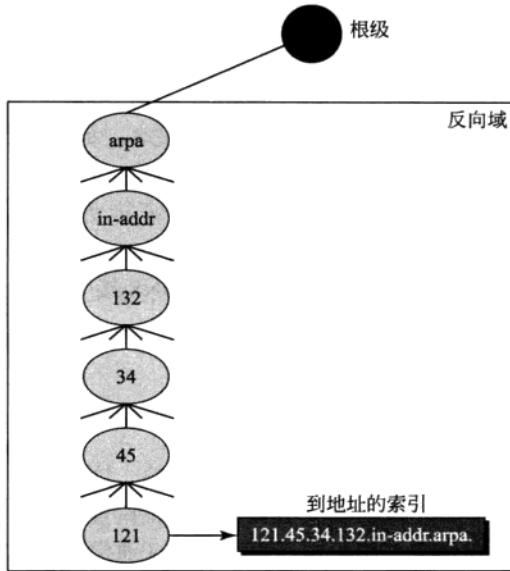


图 19.11 反向域

19.4 解析

把名字映射为地址，或者把地址映射为名字，都称为名字地址解析 (name-address resolution)。

19.4.1 解析程序

DNS 被设计为客户-服务器应用程序。需要把地址映射为名字或把名字映射为地址的主机要调用称为解析程序 (resolver) 的 DNS 客户程序。解析程序访问靠得最近的 DNS 服务器并发出映射请求。若 DNS 服务器有这个信息，就满足解析程序的要求，否则，或者让解析程序再去找其他的服务器，或者直接请其他服务器提供这个信息。

当解析程序收到映射后，就解释这个响应，看它是真正的解析还是差错，最后把结果交给请求映射的进程。

19.4.2 名字到地址的映射

在大多数的时间，解析程序是把域名交给服务器，请它给出相应的地址。在这种情况下，服务器检查类属域或国家域并查找映射。

若域名是来自类属域部分，解析程序就会收到如“chal.atc.fhda.edu”这样的域名。解析程序把这个查询发送到本地 DNS 服务器进行解析。若本地服务器不能解析这个查询，它或者让解析程序再找其他服务器，或者直接询问其他服务器。

若域名是来自国家域部分，解析程序就会收到如“ch.fhda.cu.ca.us”这样的域名。其过程是一样的。

19.4.3 地址到名字的映射

客户也可以向服务器发送 IP 地址并要求映射其域名。如前面指出的，这称为 PTR 查询。要回答这类查询，DNS 使用反向域。但是，在请求中 IP 地址必须反过来，同时还要附上两个标号 in-addr 和 arpa，以创建可以为反向域部分所接受的域。例如，若解析程序收到的 IP 地址是 132.34.45.121，解析程序首先把地址反过来，并在发送前加上两个标号。发出的域名是“121.45.34.132.in-addr.arpa”，它由本地 DNS 接受和解析。

19.4.4 递归解析

图 19.12 描绘了递归解析过程。

客户（解析程序）可以向名字服务器请求递归回答。这就表示解析程序期望服务器提供最终的解答。若服务器是这个域名的权限服务器，就检查它的数据库并响应。若服务器不是它的权限服务器，就将请求发送给另一个服务器（通常是父服务器）并等待响应。若父服务器是权限服务器，则响应，否则，就将查询再发送给另一个服务器。当查询最终被解析后，响应就沿原路返回，直至最后到达发出请求的客户。

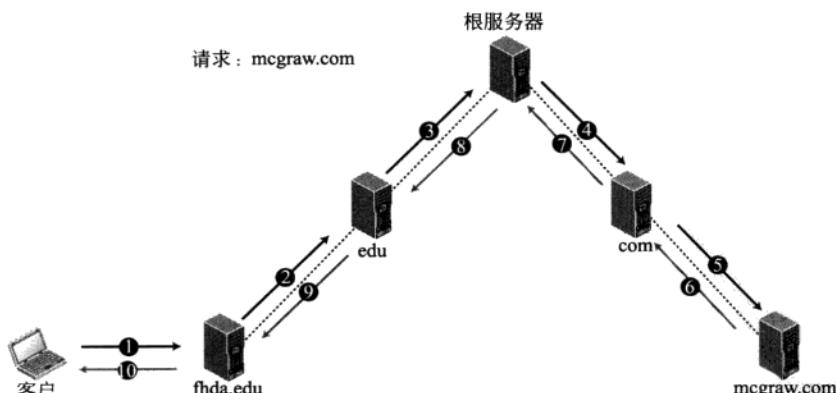


图 19.12 递归解析

19.4.5 迭代解析

若客户没有要求递归回答，则映射可以按迭代方式进行。若服务器是这个域名的权限服务器，它就发送解答。若不是，就返回（到客户）它认为可以解析这个查询的服务器的 IP 地址。客户再次向第二个服务器发送查询。若新找到的服务器能够解决这个问题，就用 IP 地址回答，否则，就向客户返回一个新的服务器的 IP 地址。现在客户必须向第三个服务器再次查询。这个过程称为迭代的，因为客户向多个服务器重复同样的查询。在图 19.13 中，客户在从 mcgraw.com 服务器获得解答之前，一共查询了五个服务器。

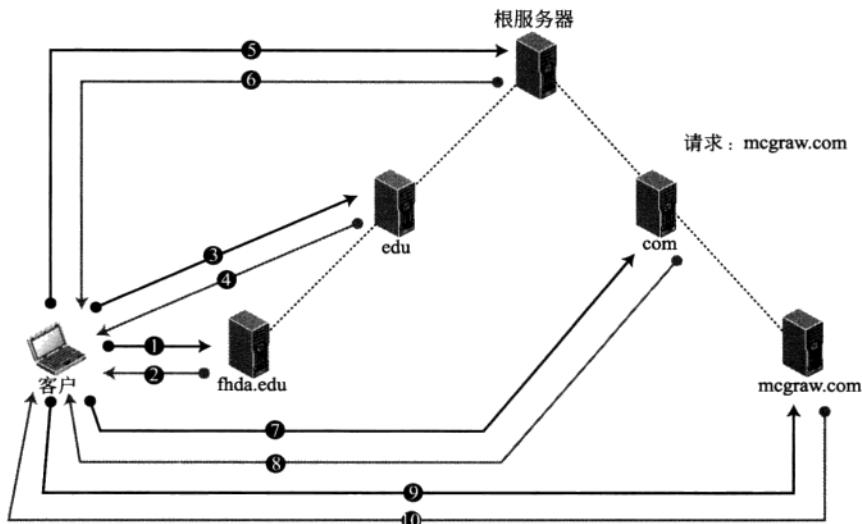


图 19.13 迭代解析

19.4.6 高速缓存

每当服务器收到一个查询，如果被查询的名字不在它的域中，服务器就要在它的数据库中搜索另一个服务器的 IP 地址。减少这种查找时间可以提高效率。DNS 使用一种称为 **高速缓存 (caching)** 的机制来处理这个问题。当服务器向另一个服务器请求映射并收到响应后，在把这个信息返回给客户之前，先要存储在自己的高速缓存中。若同一个客户或另一个客户请求同样的映射时，它就检查高速缓存并解析。不过，为了告知客户这个响应是来自高速缓存，而不是来自一个授权的信息源，这个服务器要把响应标记为未授权的。

高速缓存加速了解析过程，但也可能会带来问题。若服务器把映射放入高速缓存已有很长的时间，则可能会把过时的映射发送给了客户。要解决这个问题，有两种技术可以使用。第一，权限服务器总是把称为生存时间 (TTL) 的信息添加在映射上。生存时间定义了接收信息的服务器可以把信息放入高速缓存的时间长度 (以秒为单位)。经过这段时间后，这个映射就变成为无效的，因而任何查询都必须再发送给权限服务器。第二，DNS 要求每

一个服务器对缓存中的每一条映射都要保持一个 TTL 计数器。高速缓存必须定期地搜索，并清除那些 TTL 到期的映射。

19.5 DNS 报文

DNS 有两种类型的报文：查询和响应。两种类型的报文格式相同。查询报文包括一个首部和若干个问题记录。响应报文包括一个首部和若干个问题记录、回答记录、授权记录以及附加记录（见图 19.14）。

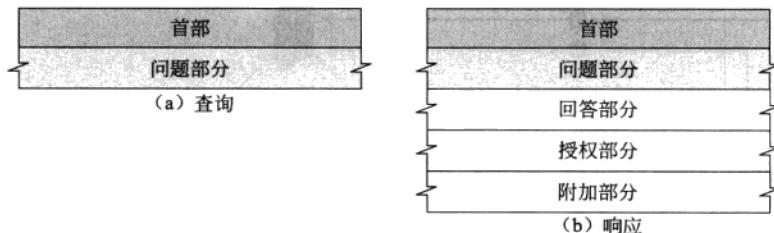


图 19.14 查询报文和响应报文

19.5.1 首部

查询报文和响应报文具有相同的首部格式，对于查询报文则把某些字段置为 0。首部是 12 字节，其格式如图 19.15 所示。

标识	标志
问题记录数	回答记录数 (在查询报文中是全0)
授权记录数 (在查询报文中是全0)	附加记录数 (在查询报文中是全0)

图 19.15 首部格式

首部中的各字段如下：

- **标识** 客户使用这个 16 位字段使响应与查询匹配。客户在每次发送查询时使用不同的标识号。服务器在相应的响应中重复这个标识号。
- **标志** 这个 16 位字段包括如图 19.16 所示的一些子字段。



图 19.16 标志字段

下面简单地说明一下各标志子字段。

- a. **QR** (查询/响应)。这是定义报文类型的 1 位子字段。若它为 0，就是查询报文。若为 1，就是响应报文。

b. **OpCode**。这个 4 位子字段定义了查询或响应的类型（若为 0 则是标准的，若为 1 则是反向的，若为 2 则是服务器状态请求）。

c. **AA**（授权回答）。这个 1 位子字段，当它置位时（值为 1），表示名字服务器是权限服务器。它只用在响应报文中。

d. **TC**（截断的）。这个 1 位子字段，当它置位时（值为 1），表示响应已超过 512 字节并已截断为 512 字节。当 DNS 使用 UDP 服务时会使用这个标志（见第 19.8 节的封装部分）。

e. **RD**（要求递归）。这个 1 位子字段，当它置位时（值为 1），表示客户希望得到递归回答。它在查询报文中置位，在响应报文中重复置位。

f. **RA**（递归可用）。这个 1 位子字段，当它在响应中置位时，表示可得到递归响应。它只能在响应报文中置位。

g. **保留**。这个 3 位子字段目前置为 000。

h. **rCode**。这个 4 位子字段表示在响应中的差错状态。当然，只有权限服务器才能做出这个判断。表 19.2 给出了这个字段的一些可能的值。

表 19.2 rCode 的值

值	意 义	值	意 义
0	无差错	4	查询类型不支持
1	格式差错	5	在管理上被禁止
2	问题在域名服务器上	6~15	保留
3	域参照问题		

- **问题记录数** 这个 16 位字段包含了报文的问题部分中的查询记录数。
- **回答记录数** 这个 16 位字段包含了响应报文的回答部分中的回答记录数。在查询报文中它的值是 0。
- **授权记录数** 这个 16 位字段包含了响应报文的授权部分中的授权记录数。在查询报文中它的值是 0。
- **附加记录数** 这个 16 位字段包含了响应报文的附加部分中的附加记录数。在查询报文中它的值是 0。

问题部分

这一部分包括了一个或多个问题记录。它在查询报文和响应报文中都要出现。我们将在下一节讨论这个问题部分。

回答部分

这一部分包括了一个或多个资源记录。它只在响应报文中出现，内容是从服务器到客户（解析程序）的回答。我们将在下一节讨论这些资源记录。

授权部分

这一部分包括了一个或多个资源记录。它只在响应报文中出现。这部分为该查询给出一个或多个权限服务器的相关信息（域名）。

附加信息部分

这一部分包括了一个或多个资源记录。它只在响应报文中出现。这部分提供了辅助解析程序的附加信息。例如，服务器可以在授权部分向解析程序提供权限服务器的域名，而

在附加信息部分提供同一个权限服务器的 IP 地址。

19.6 记录的类型

如我们在前一节所介绍的，DNS 使用两种类型的记录。问题记录用于查询报文和响应报文的问题部分。资源记录用于响应报文的回答、授权和附加信息部分。

19.6.1 问题记录

问题记录（question record）用于客户获取服务器上的信息。问题记录中包含了域名。图 19.17 给出了问题记录的格式。下面列出了问题记录的各字段。

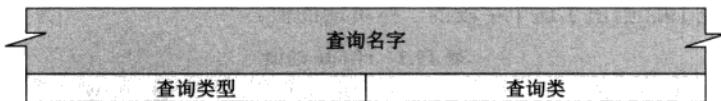


图 19.17 问题记录的格式

□ **查询名字** 这是包含了域名的可变长度字段（见图 19.18）。其中计数字段指出每一节中的字符数。

计数	计数	计数	计数	计数	计数
5	a	d	m	i	n

图 19.18 查询名字的格式

□ **查询类型** 这是定义查询类型的 16 位字段。表 19.3 给出了一些常用的类型。最后两种只能用在查询中。

表 19.3 类型

类型	助记符	说 明
1	A	地址。32 位的 IPv4 地址。它用来把域名转换为地址
2	NS	名字服务器。它标志了区的权限服务器
5	CNAME	规范名称。它定义主机的正式名字的别名
6	SOA	授权开始。它标记一个区的开始
11	WKS	熟知服务。它定义主机提供的网络服务
12	PTR	指针。它用来把 IP 地址转换为域名
13	HINFO	主机信息。它给出主机使用的硬件和操作系统
15	MX	邮件交换。它把邮件转发到一个邮件服务器
28	AAAA	地址。IPv6 地址（见第 26 章）
252	AXFR	请求传送完整区文件
255	ANY	请求所有记录

- **查询类别** 这是一个 16 位字段，定义了使用 DNS 的特定协议。表 19.4 给出了当前值。在本书中，我们只对类 1（因特网）感兴趣。

表 19.4 类别

类型	助记符	说 明
1	AN	因特网
2	CSNET	CSNET 网络（陈旧的）
3	CS	COAS 网络
4	HS	由 MIT 开发的 Hesoid 服务器

19.6.2 资源记录

每一个域名（树上的每一个结点）都与一个称为资源记录（resource record）的记录相关联。服务器数据库由许多资源记录组成。同时，资源记录也是服务器向客户所返回的信息。图 19.19 给出了资源记录的格式。

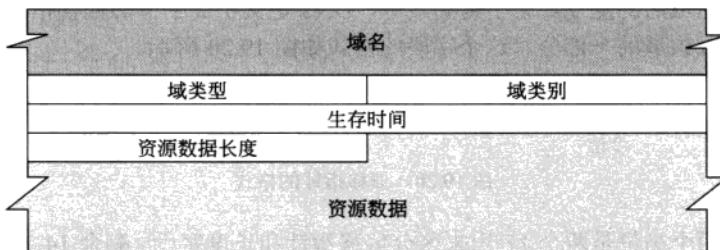


图 19.19 资源记录的格式

- **域名** 这是包含了域名的可变长度字段。它是问题记录中的域名的副本。由于 DNS 要求在名字重复出现的地方使用压缩，所以这个字段是问题记录中的域名的偏移量指针。参见第 19.7 节对压缩的介绍。
- **域类型** 这个字段与问题记录的查询类型字段相同，只不过最后两个类型是不允许的。更多的信息可参考表 19.3。
- **域类别** 这个字段与问题记录的查询类别字段相同（见表 19.4）。
- **生存时间** 这个 32 位字段定义了此回答的有效期，以秒为单位。在这段有效期内，接收方可以将此回答保存在高速缓存中。值为 0 就表示该资源记录只能用于单个的事务，而不能存放在高速缓存中。
- **资源数据长度** 这是定义资源数据长度的 16 位字段。
- **资源数据** 这个可变长度字段包含了对查询的回答（在回答部分），或者权限服务器的域名（在授权部分），或者一些附加信息（在附加信息部分）。这个字段的格式和内容取决于类型字段的值。它可以是下述之一：
 - 数值** 这个数以八位组为单位。例如，IPv4 地址是 4 个八位组的整数倍，而 IPv6

地址是 16 个八位组的整数倍。

- b. 域名 域名用标号序列来表示。每一个标号前面有 1 字节的长度字段，它定义标号中的字符数。因为每个域名都以空标号结束，因此每个域名的最后一个字节都是值为 0 的长度字段。为了区分长度字段和偏移指针（我们将在后面讨论），长度字段的两个高位永远是零（00）。这不会产生问题，因为标号的长度不能超过 63，即 6 位的最大值（111111）。
- c. 偏移指针 域名可以用偏移指针来代替。偏移指针是 2 字节的字段，它的两个最高位置为 1（11）。
- d. 字符串 字符串是用 1 字节的长度字段紧跟着一串该长度的字符。长度字段并不像域名长度字段那样受限。字符串可以多达 255 个字符（包括长度字段）。

19.7 压缩

当出现域名重复时，DNS 要求用偏移指针来代替域名。例如，资源记录中的域名通常是问题记录中的域名的重复。为了提高效率，DNS 定义了 2 字节的偏移指针，它指向前一次出现的该域或该域的一部分。这个字段的格式如图 19.20 所示。

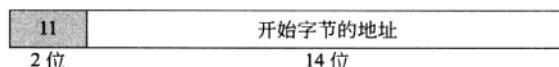


图 19.20 偏移指针的格式

最前面的两个高位是两个 1，用来区分偏移指针和长度字段。剩余 14 位代表了一个数值，指出在该报文中相应字段的偏移字节数。报文中的字节是从报文开始处计数，而第一个字节是字节 0。例如，若偏移指针指向报文的字节 12（第 13 个字节），那么这个值则是 1100000000001100。其中最左边的两位定义了这个字段是一个偏移指针字段，剩余的几位则定义了十进制数 12。我们将要在下面的例子用到这个偏移指针。

例 19.1

解析程序向本地服务器发送了查询报文，希望找出主机“chal.fhda.edu”的 IP 地址。我们分别来讨论查询报文和响应报文。图 19.21 给出了解析程序发送的查询报文。

0x1333		0x0100	
1		0	
0		0	
4	'c'	'h'	'a'
'T'	4	'f'	'h'
'd'	'a'	3	'e'
'd'	'u'	0	在下一行继续
1		1	

图 19.21 例 19.1：查询报文

前两个字节是标识 $(1333)_{16}$, 它被用做序号, 以便响应与查询相关联。由于解析程序可以向同一个服务器发送多个查询, 因此这个标识有助于对失序到达的响应进行排序。接下来的一个字节包含的是标志, 值为十六进制的 $0x0100$ 。它用二进制表示是 00000001 00000000, 但把它划分为如下所示的几个字段就会更有意义:

QR	OpCode	AA	TC	RD	RA	保留	rCode
0	0000	0	0	1	0	000	0000

QR 位定义了该报文是查询报文。OpCode 是 0000, 表示这是标准查询。RD 位是置位的。(参考前面的图 19.16 关于标志字段的描述)。这个报文只包含了一个问题记录。域名是 **4chal4fhda3edu0**。下面的两个字节定义了查询类型是 IP 地址, 最后两个字节定义了查询类别是因特网。

图 19.22 给出了服务器的响应。



0x1333	0x8180
1	1
0	0
4	'c'
'T'	'h'
'd'	'a'
'd'	'e'
1	'u'
0xC0	在下一行继续
1	1
0x0C	在下一行继续
1	12 000 在下一行继续
	4 153
18	8 105

图 19.22 例 19.1: 响应报文

响应与查询相似, 除了标志不同, 并且回答记录数为 1。此时标志的值是十六进制的 $0x8180$ 。用二进制表示, 它是 10000001 10000000, 不过我们还是要把它划分为以下几个字段:

QR	OpCode	AA	TC	RD	RA	保留	rCode
1	0000	0	0	1	1	000	0000

QR 位定义了此报文是响应报文。OpCode 是 0000, 表示这是标准响应。RA 位和 RD 位是置位的。这个报文只包含一个问题记录和一个回答记录。问题记录是从查询报文中复制的。回答记录中有一个值 $0xC00C$ (分成为两行), 它指向了问题记录, 而不是对这个域名再重复一遍。下一个字段定义域类型 (地址)。再下一个字段定义了域类别 (因特网)。值为 12 000 的字段是 TTL 字段 (12 000 秒)。再下面的字段是资源数据长度字段, 而资源数据则是一个 IP 地址 (153.18.8.105)。

例 19.2

FTP 服务器收到从 IP 地址为 153.2.7.9 的 FTP 客户发来的分组。FTP 服务器想验证这个 FTP 客户是否为授权客户。FTP 服务器可以咨询一个包含了授权客户列表的文件。但是, 这个文件中只包含域名。FTP 服务器只知道请求客户的 IP 地址, 也就是收到的 IP 数据报的源 IP 地址。FTP 服务器要求解析程序 (DNS 客户) 发送反向查询给 DNS 服务器, 以询

问这个 FTP 客户的域名。我们分别讨论查询报文和响应报文。图 19.23 给出了解析程序发送给服务器的查询报文。

0x1200		0x0900	
1		0	
0		0	
1	'9'	1	'7'
1	'2'	3	'T'
'5'	'3'	7	'T'
'n'	'.'	'a'	'd'
'd'	'r'	4	'a'
'r'	'p'	'a'	0
12		1	

图 19.23 例 19.2: 反向查询报文

前两个字节是标识 (0x1200)。标志字段的值是十六进制的 0x0900。用二进制表示，它是 00001001 00000000，我们把它划分为以下几个字段：

QR	OpCode	AA	TC	RD	RA	保留	rCode
0	0001	0	0	1	0	000	0000

OpCode 是 0001，表示反向查询。这个报文只包含一个问题记录。其中的域名是 19171231537in-addr4arpa。接下来的两个字节定义了查询类型是 PTR，最后两个字节定义了查询类别是因特网。

图 19.24 给出了响应报文。标志字段的值是十六进制的 0x8D80。用二进制表示，它是 10001101 10000000，而我们再把它划分为以下几个字段：

QR	OpCode	AA	TC	RD	RA	保留	rCode
1	0001	1	0	1	1	000	0000

0x1200		0x8D80	
1		1	
0		0	
1	'9'	1	'7'
1	'2'	3	'T'
'5'	'3'	7	'T'
'n'	'.'	'a'	'd'
'd'	'r'	4	'a'
'r'	'p'	'a'	0
12		1	
0xC00C		12	
1		在下一行继续	
24000		10	
4	'm'	'h'	'h'
'e'	3	'c'	'o'
'm'	0		

图 19.24 例 19.2: 反向响应报文

这个报文包含一个问题记录和一个回答记录。问题记录是从查询报文中复制的。在回答记录中有一个值 0xC00C，它指向了问题记录，而不是对此域名的重复。下一个字段定义域的类型 (PTR)。接下来的一个字段定义域类别 (因特网)，在其之后的字段定义了 TTL (24 000 秒)。再后面的字段是资源数据长度字段 (10)。最后一个字段是域名 4mhhe3com0，它表示 “mhhe.com”。

例 19.3

在 UNIX 和 Windows 中，实用程序 nslookup 可用来读取地址/名字的映射。下面给出的是在给定一个域名时我们怎样获取其地址。

```
$ nslookup fhda.edu
Name: fhda.edu
Address: 153.18.8.1
```

当给定如下地址时，实用程序 nslookup 也可以用来获取其域名。

```
$ nslookup 153.18.8.1
1.8.153.in-addr.arpa名字 = tiptoe.fhda.edu.
```

19.8 封装

DNS 可以使用 UDP，也可以使用 TCP。在这两种情况下，服务器使用的熟知端口都是端口 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 封装都会受到 512 字节的分组长度限制。若响应报文的长度超过 512 字节，就要使用 TCP 连接。此时，会出现以下两种情况之一：

- 若解析程序事先知道响应报文的长度超过 512 字节，就应当使用 TCP 连接。例如，如果次名字服务器（作为客户）请求主服务器的区传送，它就必须使用 TCP 连接，因为要传送的信息的长度一般都会超过 512 字节。
- 若解析程序不知道响应报文的长度，它可以使用 UDP 端口。但是，若响应报文的长度超过了 512 字节，服务器就要截断响应报文，并把 TC 位置 1（参见 19.16）。此时，解析程序打开 TCP 连接，并重复这个请求，以便从服务器得到完整的响应。

19.9 注册机构

新的域是怎样加入到 DNS 的呢？这要通过一个注册机构 (registrar)，即由 ICANN 认可的一个商业实体。注册机构首先验证这个请求的域名的唯一性，然后把它录入到 DNS 数据库。这是要收费的。目前存在许多注册机构，它们的名称和地址可以从以下地址获得：

<http://www.intenic.net>

如果某个组织想要注册，它需要提交自己的服务器的名字和 IP 地址。例如，一个名为

wonderful 的新商业组织有一台名为 ws 的服务器，并且该服务器的 IP 地址是 200.200.200.5，它需要向某一个注册机构提交以下信息：

域名：ws.wonderful.com

IP 地址：200.200.200.5

19.10 DDNS

在最初设计 DNS 时，没有人预料到地址会有这样多的变化。在 DNS 中，只要有一个变化，例如增加了一台主机、移走了一台主机或者改变了一个 IP 地址，那么这个变化就必定会使 DNS 主文件发生变化。这些变化会涉及到许多人工更新。今天的因特网规模已经不允许这样的人工操作了。

DNS 主文件必须动态地更新。因此，人们设计了**动态域名系统 (Dynamic Domain Name System, DDNS)** 以适应这个需要。在 DDNS 中，当名字和地址的绑定确定后，通常 DHCP (见第 18 章) 就会向主 DNS 发送这个信息。主服务器更新这个区，然后主动地或被动地通知次服务器。在主动通知时，主服务器向次服务器发送关于区的变化的报文，而在被动通知时，次服务器定期地检查是否有任何变化。不管是哪一种情况，在次服务器被通知有变化后，就请求关于整个区的信息 (区传送)。

为了提供安全性并防止 DNS 记录被未授权地修改，DDNS 可以使用一种鉴别机制。

19.11 DNS 的安全性

DNS 是因特网基础设施中最重要的系统之一，它向因特网用户提供了关键性的服务。像 Web 访问和电子邮件这样的应用程序在很大程度上要依赖于 DNS 的正常工作。DNS 可能会受到多种方式的攻击，包括：

1. 攻击者可能会读出一个 DNS 服务器的响应，以找出用户最常访问的站点的性质或名称。此类信息可被用来发现该用户的特征。为了防止此类攻击，DNS 报文需要加密（参见第 29 章）。
2. 攻击者可能会截获一个 DNS 服务器的响应，然后修改这个响应或创造一个全新的假冒的响应，从而将用户引导到攻击者希望该用户去访问的站点或主域。此类攻击可以通过使用报文原始鉴别和报文完整性措施来防止（参见第 29 章）。
3. 攻击者可能会用洪泛的方法使 DNS 服务器超载，甚或最终使之崩溃。此类攻击可以通过提供反拒绝服务攻击（denial-of-service attack）来防止。

为了保护 DNS，IETF 已经设计出一种称为 **DNS 安全性 (DNSSEC)** 的技术，它利用称为数字签名的安全服务来提供报文的原始性鉴别和维护报文的完整性（参见第 29 章）。但是 DNSSEC 没有提供对 DNS 报文的加密保护。在 DNSSEC 规范中也没有反拒绝服务攻击的具体保护措施。不过，高速缓存系统使得上层服务器能够或多或少地抵抗此类攻击。

19.12 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

19.12.1 参考书

有几本书籍和 RFC 简要且全面地覆盖了 DNS，包括[Tan 03]、[Ste 94]和[Com 06]。

19.12.2 RFC

DNS 经过了多次变化，通过一些 RFC 可以看出 DNS 的历次更新，包括 RFC 1034、RFC 1035、RFC 1996、RFC 2535、RFC 3008、RFC 3658、RFC 3755、RFC 3757 和 RFC 3845。

19.13 重要术语

高速缓存	迭代解析
压缩	标号
国家域	名字空间
DNS 服务器	不完整域名 (PQDN)
域	主服务器
域名	问题记录
域名空间	注册机构
域名系统 (DNS)	解析程序
动态域名系统 (DDNS)	资源记录
平面名字空间	根服务器
完整域名 (FQDN)	次服务器
类属域	子域
层次名字空间	区
反向域	

19.14 本章小结

□ 域名系统 (DNS) 是一个客户–服务器应用程序，它用唯一的、用户友好的名字来

标志因特网上的每一个主机。DNS 把名字空间组织成分级的结构以使涉及到命名的各种责任分散化。

- DNS 可以被描绘成倒过来的分级的树结构，它的根在最上面，最多可以有 128 级。树上的每一个结点都有一个域名。域被定义为域名空间中的任何一个子树。
- 名字空间的信息分布在不同的 DNS 服务器上。每一个服务器对它的区有管辖权。根服务器的区就是整个的 DNS 树。主服务器要创建、维护和更新它所管辖的区的信息。次服务器从主服务器上获取自己的信息。
- 域名空间划分为三大部分：类属域、国家域和反向域。类属域一共有 14 个，每一个域指明一个组织类型。每一个国家域指明一个国家。反向域是根据已知的 IP 地址来查找相应的域名，这称为地址到名字的解析。
- 运行 DNS 服务器程序的计算机称为名字服务器，它们以分级的方式组织。DNS 客户称为解析程序，它把名字映射为地址或把地址映射为名字。在递归解析过程中，客户把它的请求发送给服务器，服务器会返回最终的响应。在迭代解析过程中，客户在得到回答之前，可能要把它的请求发送给多个服务器。高速缓存是一种把查询得到的回答存放在存储器中（存放有限的时间），以便为今后的请求使用的方式。
- 完整域名（FQDN）是标号从主机开始一直向回走，通过每一个结点直至到达根结点的一种域名。不完整域名（PQDN）是没有完全包括从主机到根结点之间所有级的一种域名。
- DNS 报文有两种类型：查询报文和响应报文。DNS 记录也有两种类型：问题记录和资源记录。DNS 为报文中重复出现的域名信息使用了偏移指针。动态 DNS（DDNS）自动地更新 DNS 主文件。DNS 对小于 512 字节的报文使用 UDP 服务，否则，就使用 TCP。
- 为了保护 DNS，IETF 已经设计出一种称为 DNS 安全性（DNSSEC）的技术，它利用称为数字签名的安全服务来提供报文的原始鉴别和维护报文的完整性。

19.15 实践安排

19.15.1 习题

1. 试确定下面的域名中哪一个是 FQDN，哪一个是 PQDN？

- a. xxx
- b. xxx.yyy.
- c. xxx.yyy.net
- d. zzz.yyy.xx.edu.

2. 试确定下面的域名中哪一个是 FQDN，哪一个是 PQDN？

- a. mil.

- b. edu.
 - c. xxx.yyy.net
 - d. zzz.yyy.xxx.edu
3. 试给出查询报文在请求地址并需要递归回答时，各标志字段的值（用十六进制表示）。
4. 试给出携带反向响应的非授权报文的各标志字段的值（用十六进制表示）。解析程序要求递归响应，但递归回答不可用。
5. 试分析标志 0x8F80。
6. 试分析标志 0x0503。它合法吗？
7. 问题记录的长度是固定的吗？
8. 资源记录的长度是固定的吗？
9. 包含域名 fhda.edu 的问题记录的长度是多少？
10. 包含 IP 地址的问题记录的长度是多少？
11. 包含域名 fhda.edu 的资源记录的长度是多少？
12. 包含 IP 地址的资源记录的长度是多少？
13. 请求 challenger.atc.fhda.edu 的 IP 地址的查询报文长度是多少？
14. 请求 185.34.23.12 的域名的查询报文长度是多少？
15. 对习题 13 中的查询报文进行响应的响应报文长度是多少？
16. 对习题 14 中的查询报文进行响应的响应报文长度是多少？
17. 重做例 19.1，使用具有回答记录和授权记录的响应报文，这个授权记录定义了“fhda.edu”为权限服务器。
18. 重做习题 17，增加一个附加记录，定义权限服务器的地址为 153.18.9.0。
19. DNS 客户查找 xxx.yyy.com 的 IP 地址。试给出查询报文每一个字段的值。
20. 试给出对习题 19 的 DNS 服务器的响应报文。假定 IP 地址为 201.34.23.12。
21. DNS 客户查找对应于 xxx.yyy.com 和 aaa.bbb.edu 的 IP 地址。试给出查询报文。
22. 试给出 DNS 服务器对习题 21 中查询的响应报文。假定 IP 地址为 14.23.45.12 和 131.34.67.89。
23. 试给出习题 22 的响应报文，假定 DNS 服务器能够解析第一个查询，但不能解析第二个查询。
24. 某个 DNS 客户要查找 IP 地址为 132.1.17.8 的计算机的名字。试给出查询报文。
25. 试给出 DNS 服务器对习题 24 的查询所发送的响应报文。
26. 试把习题 24 中的查询报文封装成 UDP 数据报。
27. 试把习题 25 中的响应报文封装成 UDP 数据报。

19.15.2 研究活动

28. 试比较 DNS 的结构和 UNIX 的目录结构。
29. 在 UNIX 的目录结构中什么和 DNS 结构中的“点”(dot)作用相同？
30. DNS 的域名从一个结点开始，一直上升到树根。试问 UNIX 的路径名也是这样吗？

31. 我们能否说：DNS 的 FQDN 和 UNIX 的绝对路径名一样，而 PQDN 和 UNIX 的相对路径名一样？
32. 试找出怎样使用 Windows 中的实用程序 nslookup。
33. 试找出实用程序 nslookup 的所有选项。
34. 试对你所熟悉的一些域名试一试实用程序 nslookup。
35. 试使用实用程序 nslookup 找出某些商用万维网服务器的地址。

第20章 远程登录：TELNET与SSH

因特网及其 TCP/IP 协议族的主要任务就是向用户提供服务。虽然存在一些特殊的客户/服务器程序，我们将在后面的几章中对它们进行讨论，但是不可能对每一个需求都编写具体的客户-服务器程序。一种更好的解决方法是使用通用的客户-服务器程序，让用户能够接入到远程计算机上的任何应用程序。换言之就是允许用户登录到远程计算机上。登录后，用户可以使用远程计算机所提供的服务，并把结果返回到本地计算机上。在本章，我们要讨论两个通用的客户-服务器应用程序：TELNET 与 SSH。

目标

本章有以下几个目标：

- 介绍 TELNET 协议并说明它是如何利用网络虚拟终端的概念来实现本地的和远程的登录。
- 讨论 TELNET 中使用的选项和子选项，以及它们是如何协商的。
- 定义带外信令并说明它是如何在 TELNET 中实现的。
- 定义 TELNET 中不同的操作方式以及讨论它们的效率。
- 介绍 SSH，它可以代替 TELNET。
- 说明 SSH 中的各种构件如何被组合起来，以便在不安全的 TCP 连接上提供一条安全连接。
- 讨论 SSH 中的端口转发技术，以及如何利用这种技术向其他应用程序提供安全性。

20.1 TELNET

TELNET 是终端网络（TERminaL NETwork）的缩写。根据 ISO 的建议，它被作为虚拟终端服务的标准 TCP/IP 协议。TELNET 能够建立一条到远程系统的连接，使得本地终端就好像连接在远程系统上一样。

TELNET 是通用的客户-服务器应用程序。

20.1.1 概念

TELNET 与我们下面要讨论的几个概念有关。

20.1.2 分时的环境

在设计 TELNET 的那个时期,大多数操作系统(如 UNIX)都工作在分时(time-sharing)环境下。在分时环境下,一台大型计算机可支持许多用户。用户与计算机之间通过终端来彼此交互,这种终端通常是由键盘、监视器和鼠标组成。即使是一台微型计算机也能够用终端仿真程序来模拟一个终端。

在分时环境中,所有的处理过程都必须由中央计算机完成。当用户在键盘上键入一个字符时,这个字符通常会被发送给计算机,并在监视器上回显。分时技术创造了这样一种环境,使用户有专用计算机的错觉。用户可以运行程序,使用系统资源,从一个程序切换到另一个程序,诸如此类。

登录

在分时环境中,用户是系统的一部分,并具有使用系统资源的某些权利。每一个授权用户都有一个标识,可能还有一个口令。用户标识定义了该用户是系统的一部分。为了使用系统资源,用户需要通过用户标识或登录名来登录到系统中。系统还包括了口令检查以防止非授权用户使用资源。

本地登录 用户登录到本地分时系统就称为本地登录(local login)。当用户在终端上或在运行了终端仿真程序的工作站上进行键盘输入时,他的击键被终端驱动程序接受。

终端驱动程序把字符传递给操作系统,再由操作系统对这些字符的组合进行解释,并调用所需的应用程序或实用程序(参见图 20.1)。

但是,这种机制并不像看起来那样简单,因为操作系统可能会给一些特殊字符指派特殊的意义。例如,在 UNIX 中某些字符的组合具有特殊的意义,像字符 Ctrl 和字符“z”组合在一起就表示挂起,字符 Ctrl 和字符“c”组合在一起表示异常终止,等等。然而这些特殊情况在本地登录时不会产生任何问题,因为终端仿真程序和终端驱动程序都知道每一个字符或字符组合的准确含义,但是在远程登录时,它们就可能会产生一些问题。应当用哪一个进程来解释这些特殊字符?是客户还是服务器?我们将在本章稍后来说明这个问题。

远程登录 当用户想使用远程机器上的应用程序或实用程序时,他或她需要进行远程登录(remote login)。此时就需要用到 TELNET 客户程序和服务器程序。用户把自己的击键发送给终端驱动程序,本地操作系统从终端驱动程序那里接收这些字符,但并不解释它们。这些字符被送交给 TELNET 客户,再由 TELNET 客户把这些字符转换成称为网络虚拟终端(NVT)字符的通用字符集,然后把它们交给本地 TCP/IP 协议栈(参见图 20.2)。

这些命令或文本以 NVT 的形式经过因特网的传输到达远程机器的 TCP/IP 协议栈。在这里,字符被交付给操作系统,然后递交给 TELNET 服务器,再由 TELNET 服务器把这些字符转换为远程计算机可理解的相应字符。但是,这些字符不能直接交给操作系统,因为

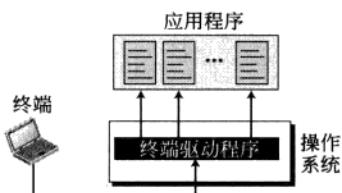


图 20.1 本地登录

远程的操作系统没有被设计成能够从 TELNET 服务器接收字符，它的设计使得它必须从终端驱动程序接收字符。解决这个问题的方法是增加一个称为伪终端驱动程序的软件，它将这些字符伪装成好像是从一个终端发来的。然后操作系统就可以把这些字符传递给适当的应用程序。

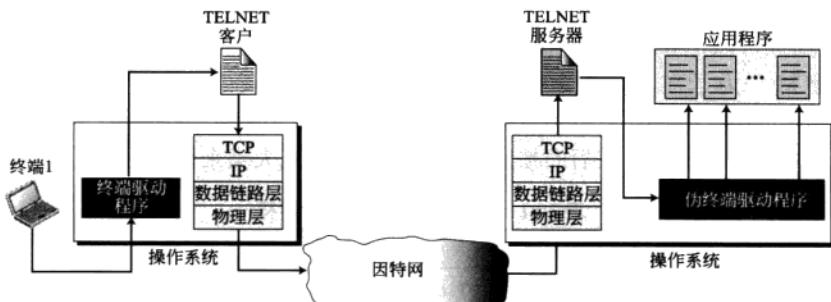


图 20.2 远程登录

20.1.3 网络虚拟终端 (NVT)

接入到远程计算机的过程很复杂，这是因为每一个计算机及其操作系统都会接受一些特殊的字符组合作为记号。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。如果我们想接入到世界上的任何一台远程计算机，那么我们必须首先知道将要连接的计算机的类型是什么，我们还必须安装那台计算机所使用的特定的终端仿真程序。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，TELNET 客户把来自本地终端的字符（数据或命令）转换成 NVT 形式，然后交付给网络。另一方面，TELNET 服务器则把 NVT 形式的数据和命令转换成远程计算机可接受的形式。在图 20.3 中描绘了这一概念。

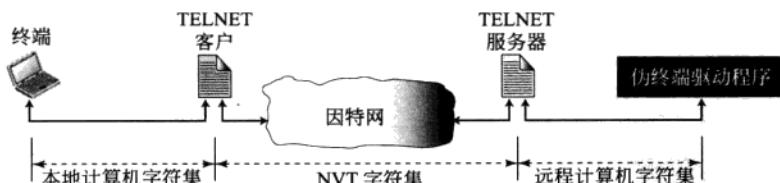


图 20.3 NVT 的概念

NVT 字符集

NVT 使用了两个字符集，一个用于数据，另一个用于远程控制。这两种字符都是 8 位字节字符（参见图 20.4）。



图 20.4 数据字符和控制字符的格式

数据字符 对于数据，NVT 通常使用称为 NVT ASCII 的字符集。这个 8 位字符集中的 7 个最低位和 US ASCII 相同，但最高位是 0（见图 20.4）。虽然也可以发送一个 8 位的 ASCII（最高位可以是 0 或 1），但这必须先在客户和服务器之间通过选项协商取得一致。

控制字符 为了在计算机之间（从客户到服务器或反之）发送控制字符（control characters），NVT 也使用了 8 位字符集，其最高位置为 1（见图 20.4）。表 20.1 列举了一些控制字符及其含义。我们将在后面按照它们的功能分成几大类来说明这些控制字符。

表 20.1 某些 NVT 远程控制字符

字符	十进制	二进制	意义
EOF	236	11101100	文件结束 (End of file)
EOR	239	11101111	记录结束 (End of record)
SE	240	11110000	子选项结束 (Suboption end)
NOP	241	11110001	无操作 (No operation)
DM	242	11110010	数据标记 (Data mark)
BRK	243	11110011	断开 (Break)
IP	244	11110100	中断进程 (Interrupt process)
AO	245	11110101	异常终止输出 (Abort output)
AYT	246	11110110	对方是否还在运行? (Are you there)
EC	247	11110111	擦除字符 (Erase character)
EL	248	11111000	擦除行 (Erase line)
GA	249	11111001	前进 (Go ahead)
SB	250	11111010	子选项开始 (Suboption begin)
WILL	251	11111011	同意使用选项
WONT	252	11111100	拒绝使用选项
DO	253	11111101	认可选项请求
DONT	254	11111110	拒绝选项请求
IAC	255	11111111	解释 (下一个字符) 为控制 (Interpret as control)

20.1.4 嵌入

TELNET 仅使用一条 TCP 连接。服务器使用熟知端口 23，客户使用临时端口。数据和控制字符的发送使用同一条连接。TELNET 是通过把控制字符嵌入到数据流中来做到这一点的。不过为了把数据和控制字符区分开，在每一个控制字符序列的前面要加上一个特殊

的控制字符，称为解释为控制（IAC）。例如，假设用户希望使用远程服务器上的某个程序来显示文件（file1），她键入了：

```
cat file1
```

其中 cat 是 UNIX 命令，用于在屏幕上显示该文件的内容。但是，这个文件的名字输入错了（键入了 filea 而不是 file1）。用户使用回退键进行改正：

```
cat filea<backspace>1
```

在 TELNET 的默认实现中，用户不能在本地进行编辑，编辑工作是在远程服务器上完成的。回退字符被转换为两个控制字符（IAC EC），它嵌入到数据中，并被发送到远程服务器。图 20.5 所示为发送给服务器的内容。

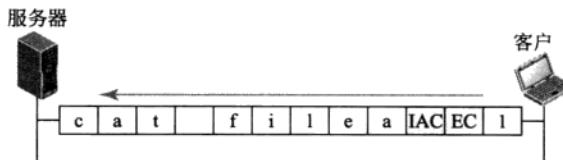


图 20.5 嵌入的例子

20.1.5 选项

TELNET 让客户与服务器在使用服务之前或期间可以互相协商选项。选项是为具有较复杂终端的用户提供额外的功能。使用较简单的终端的用户可以使用默认功能。前面讨论过的某些控制字符可用来定义选项。表 20.2 给出了一些常用的选项。

表 20.2 选项

代 码	选 项	意 义
0	Binary	使用 8 位二进制传输
1	Echo	将在一端收到的数据回显到另一端
3	Suppress go-ahead	抑制数据后面的前进信号
5	Status	请求 TELNET 的状态
6	Timing mark	定义定时标记
24	Terminal type	设置终端类型
32	Terminal speed	设置终端速率
34	Line mode	改变到行方式

下面是对这些选项的描述：

- **Binary**（二进制） 这个选项允许接收方将收到的每一个 8 位字符解释为二进制数据，但 IAC 除外。当收到 IAC 时，它的下一个或几个字符就被解释为命令。但是，若收到两个连续的 IAC 字符，则要丢弃第一个，而把第二个解释为数据。
- **Echo**（回显） 这个选项允许服务器回显收到的来自客户的数据。这就表示客户向服务器发送的每一个字符都要回显到客户终端的屏幕上。在这种情况下，

用户终端通常在字符被键入时并不显示，而要等到这些字符从服务器发送回来后才显示。

- SUPPRESS GO-AHEAD**（抑制前进） 这个选项抑制了前进（GA）字符（参见有关操作方式的小节）。
- Status**（状态） 这个选项使用户或运行在客户机器上的进程能够获取服务器端被允许使用的选项的状态。
- Timing mark**（定时标记） 这个选项允许一方发出定时标记，指出在此之前收到的所有数据都已被处理。
- Terminal type**（终端类型） 这个选项允许客户发送它的终端类型。
- Terminal speed**（终端速率） 这个选项允许客户发送它的终端速率。
- Line mode**（行方式） 这个选项允许客户切换到行方式。稍后我们将讨论这个行方式。

选项协商

要使用在前面提到的任何选项，首先需要在客户与服务器之间进行选项协商（option negotiation）。为此要用到四种控制字符，如表 20.3 所示。

表 20.3 用户选项协商的 NVT 字符集

字 符	代码	意义 1	意义 2	意义 3
WILL	251	以提供的方式允许使用	接受允许使用的请求	
WONT	252	拒绝允许使用的请求	以提供的方式禁止使用	接受禁止使用的请求
DO	253	同意允许使用	以请求的方式允许使用	
DONT	254	不同意允许使用	同意禁止使用	以请求的方式禁止使用

允许使用选项

某些选项仅能由服务器允许使用，另一些仅能由客户允许使用，还有一些则可由服务器或客户允许使用。可以通过提供或请求来允许使用选项。

提供以允许使用 任何一方都可以通过提供的方式来允许使用某个选项，只要它有此权限。对方可以同意或不同意这个提供。提供方发送 WILL 命令，表示“我可以使用这个选项吗？”另一方可发送 DO 命令，表示“同意”，或者发送 DONT 命令，表示“不同意”。参见图 20.6。

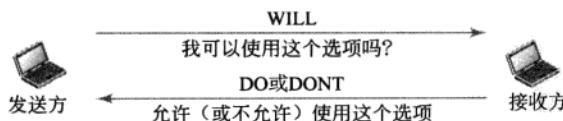


图 20.6 提供以允许使用选项

请求以允许使用 任何一方都可以通过请求的方式让另一方允许使用某个选项。另一方可以接受或拒绝这个请求。请求方发送 DO 命令，表示“请允许使用这个选项”。另一方可发送 WILL 命令，表示“同意”，或 WONT 命令，表示“不同意”。参见图 20.7。

禁止使用选项

已经被允许的选项可以被某一方禁止掉。可以通过提供或请求来禁止使用选项。

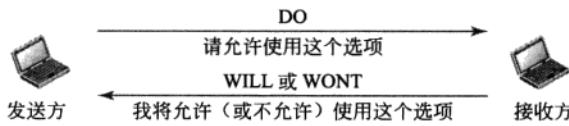


图 20.7 请求以允许使用选项

提供以禁止使用 任何一方都可以通过提供的方式来禁止使用选项。另一方必须同意，它不能不同意。提供方发送 WONT 命令，表示“我不再使用这个选项了”。回答必须是 DONT 命令，表示“不再使用这个选项”。图 20.8 所示为提供以禁止使用选项。

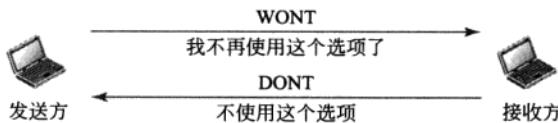


图 20.8 提供以禁止使用选项

请求以禁止使用 任何一方都可以通过请求的方式让另一方禁止使用某个选项。另一方必须接受这个请求，它不能拒绝。请求方发送 DONT 命令，表示“请不要再使用这个选项了”。回答必须是 WONT 命令，表示“我不再使用它”。图 20.9 所示为请求以禁止使用选项。

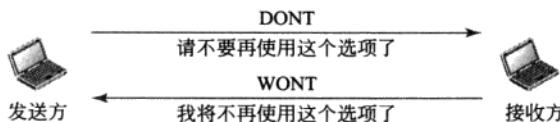


图 20.9 请求以禁止使用选项

例 20.1

图 20.10 所示为选项协商的例子。在这个例子中，客户希望服务器把发送给服务器的每一个字符进行回显。换言之，当用户在键盘终端上键入一个字符时，它必须传送给服务器，并在被处理之前回显到用户的屏幕上。服务器必须允许使用这个回显选项，因为是服务器把这些字符回送到用户终端的。因此，客户应当用 DO 命令来请求服务器允许使用这个选项。这个请求由三个字符组成：IAC，DO 和 ECHO。服务器接受该请求，并允许使用这个选项。它向客户发送三字符表示同意：IAC，WILL 和 ECHO。



图 20.10 例 20.1：回显选项

20.1.6 对称性

TELNET 的一个很有趣的特点就是对称的选项协商，即客户和服务器具有相同的机会。这就表示在连接开始时，假设双方使用的是默认的 TELNET 实现，没有允许使用任何选项。若有一方希望允许使用某个选项，它可以提供或请求。另一方有权同意对方的提供，或者，如果它没有能力或不愿意使用这个选项也可以拒绝对方的请求。这就使得 TELNET 能够进行扩充。客户或服务器可以安装较复杂的 TELNET 版本，以具有更多的选项。当它与某一方连接时，可以提供或请求这些新的选项。若另一方也支持这些选项，则这些选项就被允许使用，否则禁止使用。

20.1.7 子选项协商

某些选项需要附加的信息。例如，为了定义终端的类型或速率，双方的协商就要包括一个字符串或一个数值来定义该类型或速率。不论是哪一种情况，表 20.4 所示的两个子选项字符都是进行子选项协商所必需的。

表 20.4 子选项协商的 NVT 字符集

字符	十进制	二进制	意义
SE	240	11110000	子选项结束
SB	250	11111010	子选项开始

例如，让客户设置终端的类型，如图 20.11 所示。

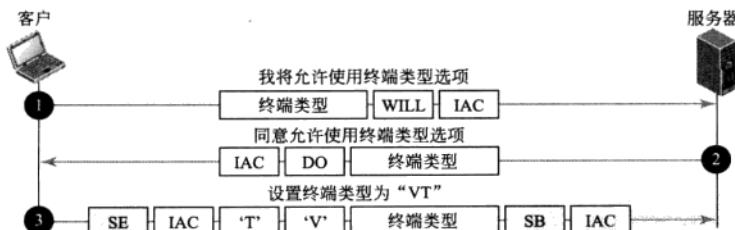


图 20.11 子选项协商的例子

20.1.8 对服务器进行控制

某些控制字符可用于对远程服务器进行控制。当应用程序运行在本地计算机上时，可以使用一些特殊字符来中断（异常终止）程序（例如，Ctrl+c），或擦除键入的最后一个字符（例如，删除键或回退键），诸如此类。但是，当程序运行在远程计算机上时，这些控制字符必须被发送到远程机器上。用户仍然键入同样的字符序列，但这些字符序列要被转换为特殊的字符后，再发送到服务器。表 20.5 中列出了一些字符，它们可以被发送到服务器，以控制在服务器上运行的应用程序。

表 20.5 用来控制在远程服务器上运行的应用程序的字符

字符	十进制	二进制	意义
IP	244	11110100	中断进程
AO	245	11110101	异常终止输出
AYT	246	11110110	对方是否在运行
EC	247	11110111	擦除最后一个字符
EL	248	11111000	擦除行

让我们更详细地讨论一下这些字符：

- **IP (Interrupt Process)** 当程序在本地运行时，用户可以中断（异常终止）这个程序，比如说，若这个程序进入了无限循环，用户可以键入 Ctrl+c 的组合，操作系统就调用使这个程序异常终止的函数。但是，如果这个程序运行在远程机器上，就应当由远程机器的操作系统来调用相应的函数。TELNET 定义了这个 IP（中断进程）控制字符，当远程机器在读到这个字符后就将其解释成相应的调用中断函数的命令。
- **AO (Abort Output)** 它与 IP 基本相同，但是它允许进程在不产生输出的前提下继续运行。如果进程除了产生输出之外还有另外的作用时，这种控制就有了用武之地。用户不希望有输出，但希望其他工作照常进行。例如，大多数 UNIX 命令会产生输出并且有一个出口（exit）状态。用户可能希望为将来的使用而保留出口状态，但对输出的数据并无兴趣。
- **AYT (Are You There?)** 这个控制字符用来判断远程机器是否仍在运行中，特别是当很长一段时间没有收到来自服务器任何消息后。当服务器收到这个字符时，通常要发送一个可闻信号或可视信号以证实它还在运行着。
- **EC (Erase Character)** 当用户从键盘向本地机器发送数据时，可以用删除字符或回退字符将最后键入的字符擦除掉。在远程计算机上做同样的事则需要使用 TELNET 定义的 EC 控制字符。
- **EL (Erase Line)** 这个字符被用来在远程主机中擦除当前行。

例如，图 20.12 描绘了如何中断运行在服务器端的已失控的应用程序。用户键入 Ctrl+c，但是 TELNET 客户向服务器发送的是 IAC 和 IP 的组合。

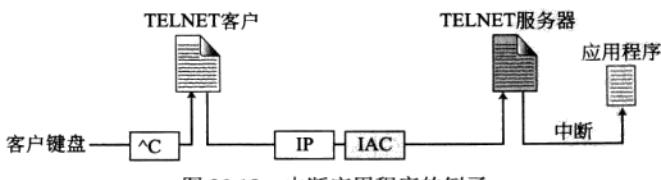


图 20.12 中断应用程序的例子

20.1.9 带外信令

为了使控制字符在某些特殊情况下仍然有效，TELNET 使用了带外信令（out-of-band

signaling)。在使用带外信令时，控制字符之前要加上 IAC，然后被发送到远程进程。

设想这样一种情况，服务器端运行的应用程序进入了无限循环，因而不再接受任何输入数据。用户希望中断这个应用程序，但该程序已不再从缓存中读取数据。服务器端的 TCP 发现缓存已满，因而发送报文段，指明客户窗口大小必须置为 0。换言之，服务器端的 TCP 宣布不能再接受更多的通信量了。为了解决这一问题，应当从客户端发送紧急 TCP 报文段给服务器。这个紧急报文段可以超越正常的流量控制机制。虽然 TCP 已不再接受正常的报文段了，但它必须接受紧急报文段。

当 TELNET 进程（客户或服务器）打算向另一个进程（客户或服务器）发送带外字符序列时，它就在数据流中嵌入这个序列，并插入称为 DM（数据标记）的特殊字符。但是为了使对方了解，它要创建紧急位置 1 的 TCP 报文段，并且紧急指针指向 DM 字符。当接收进程收到这个报文段时，它会读出数据，并丢弃在控制字符（例如，IAC 和 IP）前面的任何数据。当读到 DM 字符时，剩下的数据就按正常数据处理。换言之，DM 字符被用做同步字符，它将接收进程从紧急方式切换到正常方式，并使两端重新同步（见图 20.13）。

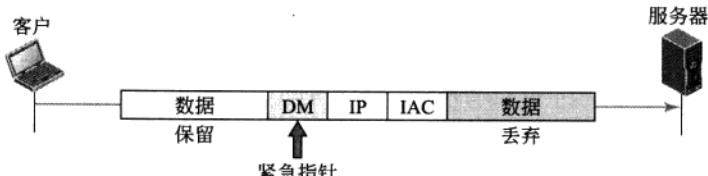


图 20.13 带外信令

使用这种方法时，控制字符（IP）以带外方式发送给操作系统，再由操作系统调用适当的函数来中断正在运行的应用程序。

20.1.10 转义字符

用户键入的字符在正常情况下是发送给服务器的。但是，有时用户希望这些字符由客户而不是服务器来解释。在这种情况下，用户要使用转义字符，通常是 Ctrl+] (记为^])。图 20.14 比较了在远程站点中断应用程序和在本地站点使用转义字符中断客户进程的过程。在该转义字符后会显示 TELNET 提示符。

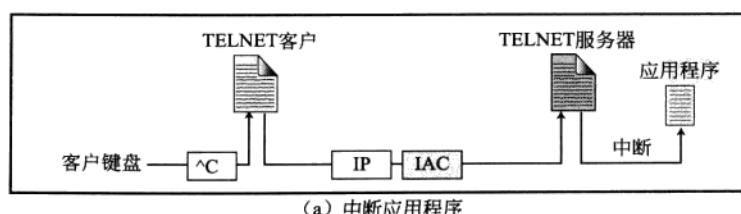
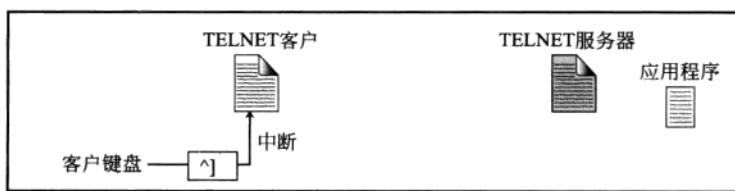


图 20.14 两种不同的中断



(b) 中断客户

图 20.14 (续)

20.1.11 操作方式

大多数 TELNET 实现工作在三种方式之一：默认方式、字符方式和行方式。

默认方式

如果没有通过选项协商调用其他方式，则使用默认方式（default mode）。在使用默认方式时，回显是由客户完成。用户键入一个字符，客户就把这个字符回显到屏幕（或打印机），但是在一整行都完成之前并不发送这个字符。在把一行发送给服务器后，客户在接受来自用户的新的一行之前，要等待服务器的 GA（前进）命令。这种操作是半双工的。TCP 连接本身是全双工时，这种半双工的操作是低效率的，因此这种方式已经过时了。

字符方式

在使用字符方式（character mode）时，每一个键入的字符都会由客户发送给服务器。服务器通常把这个字符回送并显示在客户的屏幕上。如果传输花费的时间较长（例如在卫星连接中），这种方式的字符回显可能被延迟。它还会产生网络的开销（通信量），因为对每一个数据字符必须发送三个 TCP 报文段：

1. 用户键入一个字符，它被发送给服务器。
2. 服务器确认收到的字符，并把该字符返回回显（用一个报文段）。
3. 客户确认收到回显的字符。

例 20.2

在这个例子中，我们使用默认方式来说明其概念及其低效率，虽然今天它已经基本过时了。其中，客户和服务器协商终端类型和终端速度，然后服务器检查用户的登录名和口令（参见图 20.15）。

行方式

人们已经建议了一种新的方式以弥补默认方式和字符方式的不足。这种方式称为行方式（line mode）。在使用这种方式时，行编辑（回显、字符擦除、行擦除，等等）是由客户来完成。然后客户把一行字符发送给服务器。虽然行方式看起来像默认方式，但实际上不是。默认方式工作在半双工方式，而行方式是全双工的，客户一行接着一行地发送，不需要服务器的 GA（前进）字符参与其中。

例 20.3

在这个例子中，我们要说明客户怎样切换到字符方式。这需要客户请求服务器允许使用两个选项：SUPPRESS GO AHEAD 和 ECHO（见图 20.16）。

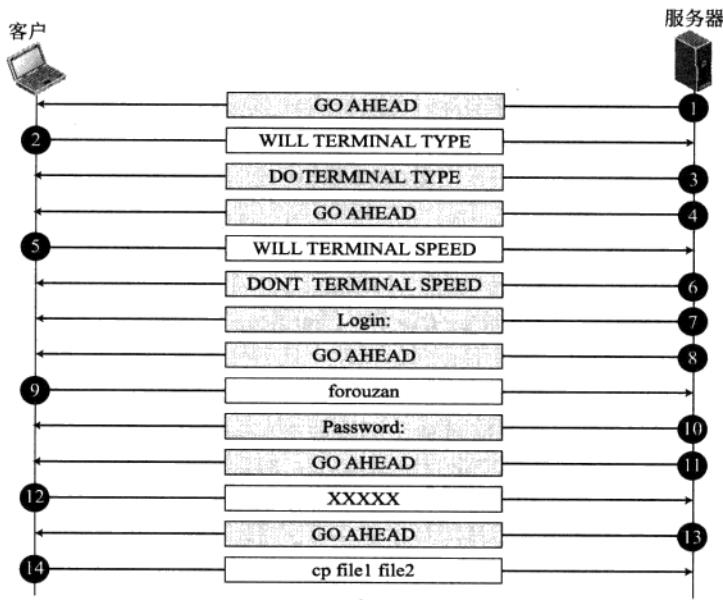


图 20.15 例 20.2

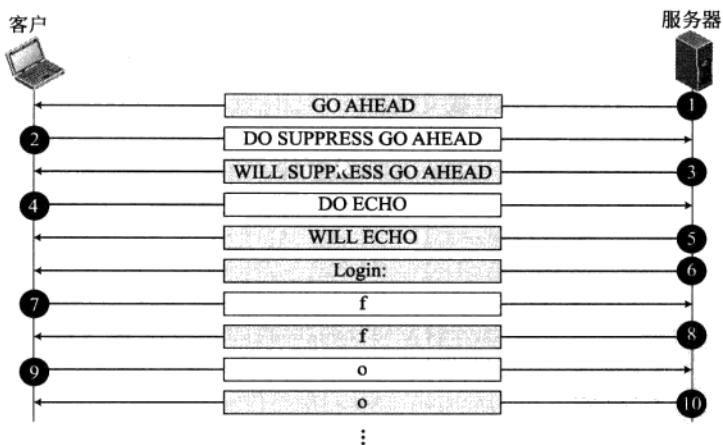


图 20.16 例 20.3

20.1.12 用户接口

普通用户并不需要使用上面定义的那些 TELNET 命令。通常，操作系统（例如，UNIX）会定义一个具有用户友好命令的接口。表 20.6 给出了这样一组命令的例子。请注意：这种接口负责把用户友好的命令转换为前面所定义的各种协议命令。

表 20.6 接口命令的例子

命令	意义	命令	意义
open	连接到远程计算机	set	设置操作参数
close	关闭这个连接	status	显示状态信息
display	显示操作参数	send	发送特殊字符
mode	改变到行方式或字符方式	quit	退出 TELNET

20.1.13 安全问题

TELNET 存在安全问题。虽然 TELNET 要求使用登录名和口令（当交换文本时），但一般来说这是不够的。连接在广播局域网上的微机可以使用嗅探器（snooper）软件很容易地截获登录名和相应的口令（哪怕它是加密的）。在第 29 章我们将进一步学习关于鉴别和安全的内容。

20.2 SSH

另外一种比较流行的登录应用程序是安全外壳（Secure Shell，SSH）。与 TELNET 一样，SSH 利用 TCP 作为底层传输协议，不过 SSH 更加安全，并且提供了比 TELNET 更多的服务。

20.2.1 版本

SSH 有两个版本：SSH-1 和 SSH-2，这两者完全不兼容。第一个版本 SSH-1 因存在安全漏洞，目前已经作废不用了。在这一节，我们只讨论 SSH-2。

20.2.2 组成

SSH 是建议的应用层协议，它由四个部分组成，如图 20.17 所示。

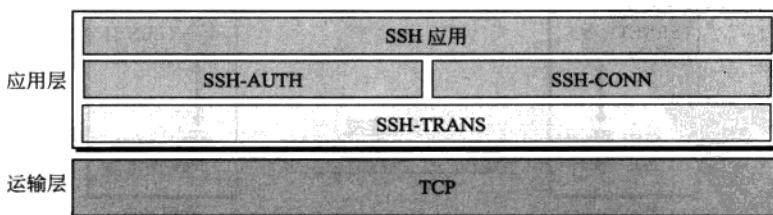


图 20.17 SSH 的组成

SSH 运输层协议 (SSH-TRANS)

因为 TCP 不是安全的运输层协议，所以 SSH 首先使用在 TCP 之上能够构建安全信道的协议。这个新的层是一个独立的协议，称为 SSH-TRANS。当这个协议的软件实现被调用后，它的客户程序和服务器程序先利用 TCP 协议建立一条不安全的准连接。然后它们相互交换几个安全参数，并在 TCP 之上建立一条安全的信道。我们将会在第 29 章详细讨论网络安全性，但是在这里我们简单地列出这个协议提供的一些服务：

1. 保密或加密报文的交换。
2. 数据的完整性，也就是说它保证在客户和服务器之间交换的报文不会被入侵者修改。
3. 服务器的鉴别，也就是说客户现在可以放心，这个服务器不会是假冒的。
4. 报文的压缩，可提高系统的效率并以此加大攻击的难度。

SSH 鉴别协议 (SSH-AUTH)

当客户和服务器之间的安全信道已建立，且客户对服务器鉴别过之后，SSH 就要调用另一个软件，它能够让服务器来鉴别客户。

SSH 连接协议 (SSH-CONN)

当安全的信道已建立，且服务器和客户相互鉴别过之后，SSH 就可以调用实现了第三个协议的软件，这第三个协议就是 SSH-CONN。SSH-CONN 协议提供的众多服务之一就是进行复用。SSH-CONN 利用前两个协议建立的安全信道，让客户在该信道上创建多个逻辑信道。

SSH 应用

在连接阶段完成后，SSH 允许几个应用程序一起使用该连接。每个应用程序如前所述地建立一条逻辑信道，然后就可以享用安全信道带来的好处。换言之，远程登录只是利用了 SSH-CONN 协议的众多服务之一，其他一些应用，比如说文件传送应用，也可以使用其中的一条逻辑信道来完成自己的任务。在下一章，我们将说明 SSH 如何被应用于安全的文件传送。

20.2.3 端口转发

在 SSH 协议所提供的众多具有吸引力的服务之中，有一项服务是端口转发（port forwarding）。我们可以利用 SSH 中空闲的安全信道来接入那些不提供安全服务的应用程序。像 TELNET 和 SMTP（参见第 23 章）这样的应用也可以通过这个端口转发机制来使用 SSH 的服务。SSH 的端口转发机制会在属于其他协议的报文所经过的路途上创建一条隧道。正因如此，有时这个机制被称为 SSH 隧道技术（tunneling）。图 20.18 所示为端口转发的概念。

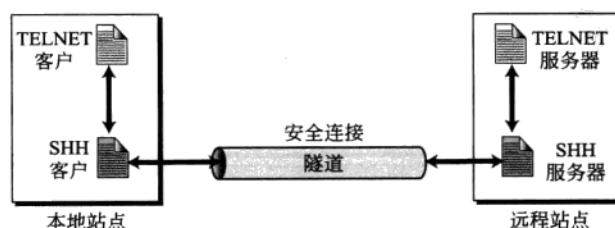


图 20.18 端口转发

TELNET 客户和 TELNET 服务器之间的连接虽然直接但缺少安全性，通过端口转发我们可以改变这种状况。TELNET 客户可以利用本地站点的 SSH 客户，以便与运行在远程站点上的 TELNET 服务器之间建立一条安全连接。任何从 TELNET 客户到 TELNET 服务器的请求都要经过由 SSH 客户和服务器提供的隧道进行传输。任何从 TELNET 服务器到 TELNET 客户的响应也要经过由 SSH 客户和服务器提供的隧道进行传输。我们将在第 30 章讨论隧道技术。

20.2.4 SSH 分组格式

图 20.19 所示为 SSH 协议使用的分组的格式。



图 20.19 SSH 分组格式

下面是对每个字节的简要介绍：

- **长度** 这个 4 字节的字段定义了分组的长度，包括类型字段、数据字段和 CRC 字段，但是不包括填充和长度字段本身。
- **填充** 在这个分组中增加 1~8 个字节的填充是为了增加其安全性，使攻击更加困难。
- **类型** 这个 1 字节的字段定义了 SSH 协议所使用的这个分组的类型。
- **数据** 这个字段长度可变。数据的长度可以用长度字段的值减去 5 个字节来推算得到。
- **CRC** 这个循环冗余检验字段用于差错检测（参见附录 D）。

20.3 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

20.3.1 参考书

有几本书籍和 RFC 简要且全面地覆盖了 TELNET 和 SSH，包括[Com 06]、[Mir 07]和[Bar et al. 05]。

20.3.2 RFC

通过一些 RFC 可以看出 TELNET 的历次更新，包括 RFC 854、RFC 855、RFC 856、

RFC 1041、RFC 1091、RFC 1372 和 RFC 1572。有关 SSH 的更多信息可以从 RFC 4250、RFC 4251、RFC 4252、RFC 4253、RFC 4254 和 RFC 4344 中找到。

20.4 重要术语

字符方式	端口转发
控制字符	远程登录
默认方式	安全外壳（SSH）
行方式	子选项协商
本地登录	终端网络（TELNET）
网络虚拟终端（NVT）	分时
选项协商	隧道
带外信令	

20.5 本章小结

- TELNET 是客户-服务器应用程序，它使用户能够登录到远程机器上，让用户能够接入远程系统。当用户通过 TELNET 进程接入到远程系统时，就相当于分时的环境。终端驱动程序可以正确地解释在本地的终端或终端仿真程序上的键盘输入。但是远程终端驱动程序可能并不能正确解释本地终端的键盘输入。
- TELNET 使用网络虚拟终端（NVT）系统对本地系统上的字符进行编码。在服务器所在的远程机器上，NVT 再把这些字符解码为该机器可接受的形式。NVT 有一组用于数据的字符集和一组用于远程控制的字符集。
- 选项用于增强 TELNET 进程功能的一些特性。TELNET 在使用服务之前或期间都可以用协商的方法，在客户和服务器之间设置传送条件。某些选项仅能被服务器允许使用，另一些仅能被客户允许使用，还有一些则可以被服务器或客户允许使用。一个选项是允许使用还是禁止使用都要通过提供或请求的方式完成。需要附加信息的选项可使用子选项字符。
- TELNET 的实现可以工作在默认方式、字符方式和行方式。在默认方式中，客户向服务器一次发送一行，并在可以接受来自用户的新行之前，需要等待前进（GA）字符。在字符方式中，客户向服务器一次发送一个字符。在行方式中，客户向服务器一次发送一行，一行接一行地发送，而不需要 GA 字符介入。
- 另一个较流行的登录应用程序是安全外壳（SSH），它比 TELNET 更安全，并且提供了更多的服务。SSH 有两个版本，我们只讨论了 SSH-2。
- SSH 由四个组件构成：SSH 应用，SSH-CONN，SSH-AUTH 和 SSH-TRANS。把以上四个组件结合起来就可以提供一个安全的远程登录，并被用于代替 TELNET。
- 在 SSH 协议所提供的众多具有吸引力的服务之中有一项是端口转发。我们可以利用 SSH 中空闲的安全信道来接入那些不提供安全服务的应用程序。

20.6 实践安排

20.6.1 习题

1. 试给出 TELNET 客户在对 11110011 00111100 11111111 做二进制传输时所发送的比特序列。
2. 若 TELNET 使用字符方式，要在 UNIX 中把名为 file1 的文件复制成名为 file2 的文件 (`cp file1 file2`)，在客户和服务器之间要往返多少个字符？
3. 要完成习题 1 的任务，在 TCP 级至少需要发送多少个比特？
4. 要完成习题 1 的任务，在数据链路层这一级（使用以太网）至少需要发送多少个比特？
5. 在习题 4 中有用的比特数与总比特数之比是多少？
6. 试给出从默认方式切换到字符方式时，在 TELNET 客户和服务器之间交换的字符序列。
7. 试给出从字符方式切换到默认方式时，在 TELNET 客户和服务器之间交换的字符序列。
8. 试给出从默认方式切换到行方式时，在 TELNET 客户和服务器之间交换的字符序列。
9. 试给出从字符方式切换到行方式时，在 TELNET 客户和服务器之间交换的字符序列。
10. 试给出从行方式切换到字符方式时，在 TELNET 客户和服务器之间交换的字符序列。
11. 试给出从行方式切换到默认方式时，在 TELNET 客户和服务器之间交换的字符序列。
12. 试解释 TELNET 客户或服务器收到的下列字符序列（以十六进制表示）代表了什么？
 - a. FF FB 01
 - b. FF FE 01
 - c. FF F4
 - d. FF F9
13. 如果你知道有一个站点允许你使用 SSH 服务器，请用你自己的计算机上的 SSH 客户与该站点建立一条连接。

20.6.2 研究活动

14. 试找出为 TELNET 建议的扩展选项。

15. SSH 提供了两种类型的端口转发：本地的和远程的。请查阅相关资料并找出它们之间的区别。
16. 另一个登录协议称为 Rlogin。试找出一些有关 Rlogin 的信息，并将其与 TELNET 和 SSH 相比较。
17. 还有一个登录协议称为远程桌面协议（Remote Desktop Protocol, RDP）。试找出一些有关 RDP 的信息，并将其与 TELNET 和 SSH 相比较。
18. 虚拟网络计算系统（Virtual Network Computing, VNC）是提供了远程桌面能力的程序。试找出一些有关 VNC 的信息，并将其与 TELNET 和 SSH 相比较。

第 21 章 文件传送：FTP 和 TFTP

从 一台计算机向另一台计算机传送文件是在连网或互联网环境中最常见的任务。事实上，今天在因特网上最大量的数据交换就来源于文件传送。在本章我们将讨论涉及到文件传送的两种协议：文件传送协议（FTP）和简单文件传送协议（TFTP）。

目标

本章有以下几个目标：

- 讨论 FTP 及此协议使用的两种连接：控制连接和数据连接。
- 讨论客户与服务器建立通信的六类命令。
- 解释由 FTP 传送的三种文件类型。
- 演示一些 FTP 界面所使用的容易掌握的命令。
- 讨论匿名 FTP 及其应用。
- 讨论如何使用安全管道传送文件。
- 讨论 TFTP 这个没有 FTP 那么复杂和高级的简单文件传送协议。
- 讨论五种 TFTP 报文及其应用。
- 讨论与 TFTP 流量及差错控制机制相关的“巫士徒弟的错误”问题。
- 演示如何通过共同使用 TFTP 和 DHCP 来下载文件并初始化设备。

21.1 文件传送协议（FTP）

文件传送协议（FTP）是 TCP/IP 提供的标准机制，用来从一个主机把文件复制到另一个主机。虽然从一个系统向另一个系统传送文件看起来很简单且直截了当，但还有一些问题必须先解决。例如，两个系统可能使用不同的文件名约定，两个系统可能用不同方法表示文本和数据，两个系统可能有不同的目录结构。所有这些问题已经由 FTP 以一种简单巧妙的方法解决了。

FTP 和其他客户-服务器应用程序的不同就是它在主机之间使用两条连接。一条连接用于数据传送，而另一条则用于传送控制信息（命令和响应）。把命令与数据的传送分开使得 FTP 的效率更高。控制连接使用非常简单的通信规则。我们需要传送的只是一次一行命令或一行响应。另一方面，数据传送需要复杂得多的规则，因为要传送的数据类型种类较多。

FTP 使用两个熟知 TCP 端口：端口 21 用做控制连接，而端口 20 用于数据连接。

FTP 使用 TCP 的服务。它需要两条连接。熟知端口 21 用于控制连接，而熟知端口 20 用于数据连接。

图 21.1 给出了 FTP 的基本模型。客户有三个构件：用户接口、客户控制进程和客户数据传送进程。服务器有两个构件：服务器控制进程和服务器数据传送进程。控制连接是在控制进程之间进行的。数据连接是在数据传送进程之间进行的。

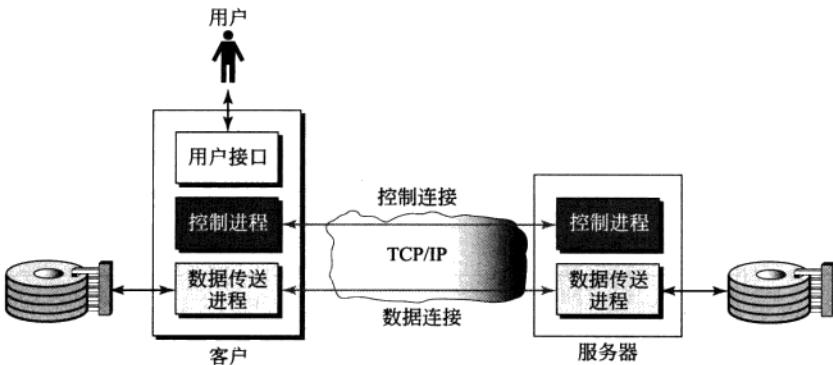


图 21.1 FTP

在 FTP 会话的整个交互过程中，控制连接始终处于连接状态。数据连接则在每一次文件传送时，先打开然后关闭。每当涉及到传送文件的命令被使用时，数据连接就被打开，而当文件传送完毕时连接就关闭。换言之，当用户开始一个 FTP 会话时，控制连接就打开。在控制连接处于打开状态时，若传送多个文件，则数据连接可以打开和关闭多次。

21.1.1 连接

两个 FTP 连接（控制和数据）使用不同的策略和不同的端口号。

控制连接

创建控制连接的方法与前面描述过的其他应用程序一样。这里共有两个步骤：

1. 服务器在熟知端口 21 发出被动打开命令，等待客户。
2. 客户使用临时端口发出主动打开命令。

在整个过程中这个连接一直是打开的。IP 协议使用的服务类型是最小延时，因为这是用户（人）和服务器的交互式连接。用户键入命令并期望收到的响应延时不能太大。图 21.2 给出了服务器和客户之间的初始连接。

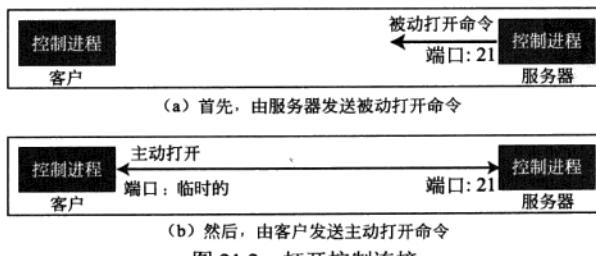


图 21.2 打开控制连接

数据连接

数据连接 使用服务器端的熟知端口 20。但创建数据连接的方法与我们已见过的不同。下面说明了 FTP 怎样创建一条数据连接：

1. 客户（而不是服务器）使用一个临时端口发出被动打开。这必须由客户来做，因为是客户发出传送文件的命令。
 2. 客户使用 PORT 命令把这个端口号发送给服务器（我们稍候将讨论这个命令）。
 3. 服务器收到这个端口号，并使用熟知端口 20 和收到的临时端口号发出主动打开。
- 图 21.3 给出了创建初始数据连接的几个步骤。以后我们将会看到，若使用 PASV 命令，则这几个步骤就改变了。

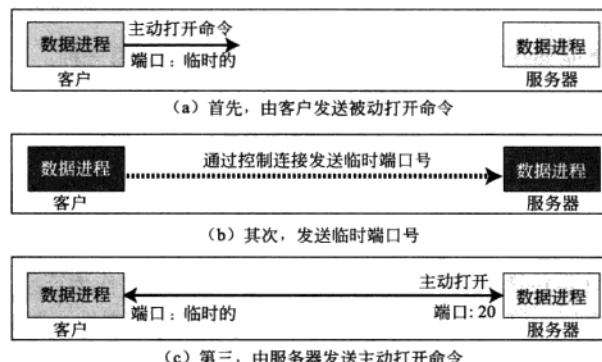


图 21.3 创建数据连接

21.1.2 通信

在不同计算机上运行的 FTP 客户和 FTP 服务器必须能够彼此进行通信。这两台计算机可以使用不同的操作系统、不同的字符集、不同的文件结构以及不同的文件格式。FTP 必须使这种异构性得到兼容。

FTP 使用了两种解决问题的方法，一种用于控制连接，另一种用于数据连接。我们将分别讨论每一种方法。

在控制连接上的通信

FTP 使用与 TELNET 或 SMTP 同样的方法在控制连接上通信。它使用 NVT ASCII 字符集（见图 21.4）。通信是通过命令和响应来完成的。这种简单的方法对控制连接是合适的，因为我们一次发送一条命令（响应）。每一条命令或响应都是一个短行，因此我们不必担心它的文件格式或文件结构。每一行的结束处是两字符（回车和换行）的行结束记号。



图 21.4 使用控制连接

在数据连接上的通信

数据连接的目的和实现与控制连接的不同。我们通过数据连接来传送数据。客户必须定义要传送的文件类型、数据结构以及传输方式。在数据连接上传送数据之前，我们通过控制连接准备传输。异构性问题可以由定义 3 个通信属性来解决：文件类型、数据结构以及传输方式（图 21.5）。

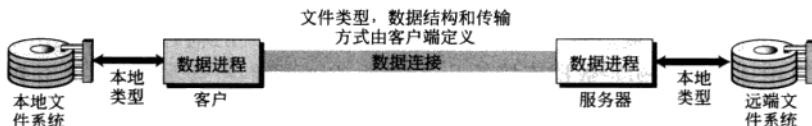


图 21.5 使用数据连接

文件类型 FTP 能够在数据连接上傳送下列文件类型中的一种：

- ❑ **ASCII 文件** 这是传送文本文件的默认格式。每一个字符使用 NVT ASCII 进行编码。发送端把文件从它自己的表示转换成 NVT ASCII 字符，而接收端从 NVT ASCII 字符转换成它自己的表示。
 - ❑ **EBCDIC 文件** 若连接的一端或两端使用 EBCDIC 编码，则可使用 EBCDIC 编码传送文件。
 - ❑ **图像文件** 这是传送二进制文件的默认格式。这种文件作为连续的比特流传送而没有任何解释或编码。这在大多数情况下是用来传送二进制文件，如已编译的程序。
若文件是用 ASCII 或 EBCDIC 编码的，则必须增加另一个属性来定义文件的可打印性。
 - a. **非打印** 这是传送文本文件的默认格式。这个文件不包含为打印用的垂直规约。这就表明，这样的文件若没有进一步的处理是不能打印的，因为没有字符能够被解释为使打印头在垂直方向移动。这种格式用来将文件存储和以后处理。
 - b. **TELNET** 在这种格式中，文件包含 NVT ASCII 垂直字符，如 CR (回车)，LF (换行)、NL (新行) 和 VT (垂直制表符)。这个文件在传送后可以打印。

数据结构 FTP 可以使用下面任何一种对数据结构的解释，在数据连接上传送文件：

- **文件结构**（默认） 这种文件没有结构。它是连续的字节流。
 - **记录结构** 这种文件划分为一些记录。这只能用于文本文件。
 - **页面结构** 这种文件划分为一些页面，每一个页面有页面号和页面首部。页面可以随机地或顺序地进行存取。

传输方式 FTP 可以使用下面 3 种传输方式之一在数据连接上传送文件：

- **流方式** 这是默认方式。数据作为连续的字节流从 FTP 交付给 TCP。TCP 负责把数据划分为适当大小的报文段。若数据是简单的字节流（文件结构），就不需要文件结束符。在这种情况下的文件结束就是由发送端来关闭数据连接。若数据划分为记录（记录结构），则每一个记录有 1 字节的记录结束（EOR）字符，而在文件的结束处有文件结束（EOF）字符。
 - **块方式** 数据可以按块从 FTP 交付给 TCP。在这种情况下，每一个块的前面有 3 字节首部。第一个字节称为块描述符，后两个字节定义块的大小，以字节为单位。
 - **压缩方式** 若文件很大，数据可进行压缩。通常使用的压缩方法是游程长度编码。

在这种方法中，数据单元的连续出现数可以用一个“出现(occurrence)”和“重复数”来替换。在文本文件中，这通常是空格。在二进制文件中，空字符常常被压缩。

21.1.3 命令处理

FTP 使用控制连接在客户进程和服务器进程之间建立通信。在通信时，从客户向服务器发送命令，而响应从服务器发回到客户（见图 21.6）。

命令

由 FTP 客户控制进程发送的命令形式

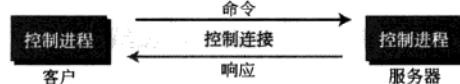


图 21.6 命令的处理

是 ASCII 大写字符，后面的变量可以有，也可以没有。我们可将命令粗略地划分为 6 组：接入命令、文件管理命令、数据格式化命令、端口定义命令、文件传送命令以及杂项命令。

□ 接入命令 这些命令使用户能够接入到远程系统。表 21.1 列举了这个组的常用命令。

表 21.1 接入命令

命 令	变 量	说 明
USER	用户标识符	用户信息
PASS	用户口令	口令
ACCT	应付费的账务	账务信息
REIN		重新初始化
QUIT		从系统注销
ABOR		前面的命令异常终止

□ 文件管理命令 这些命令使用户接入到远程计算机的文件系统。这些命令允许用户使用目录结构、创建新的目录、删除文件，等等。表 21.2 给出了这个组的常用命令。

表 21.2 文件管理命令

命 令	变 量	说 明
CWD	目录名	改变到另一个目录
CDUP		改变到父目录
DELE	文件名	删除文件
LIST	目录名	列出子目录或文件
NLIST	目录名	列出子目录名或无其他属性的文件
MKD	目录名	创建新目录
PWD		显示当前目录名
RMD	目录名	删除目录
RNFR	文件名（旧文件名）	标志要重新命名的文件
RNTO	文件名（新文件名）	重新命名文件
SMNT	文件系统名	安装文件系统

□ 数据格式化命令 这些命令让用户定义数据结构、文件类型以及传输方式。所定义的格式可由文件传送命令来使用。表 21.3 给出了这个组的常用命令。

表 21.3 数据格式化命令

命 令	变 量	说 明
TYPE	A (ASCII), E (EBCDIC), I (图像), N (非打印), 或 T (TELNET)	定义文件类型
STRU	F (文件), R (记录), 或 P (页面)	定义数据的组织
MODE	S (流), B (块), 或 C (压缩)	定义传输方式

□ 端口定义命令 这些命令定义在客户端的数据连接的端口号。有两种方法可以做到这点。第一种方法是使用 PORT 命令，客户可选择一个临时端口号，并使用被动打开把它发送给服务器。服务器就使用这个端口号创建主动打开。第二种方法是使用 PASV 命令，客户仅要求服务器先选择一个端口号。服务器在那个端口进行被动打开，并在响应中发送端口号（见表 21.7 中的编号为 227 的响应）。客户使用这个端口号发出主动打开。表 21.4 给出了这个组的常用命令。

表 21.4 端口定义命令

命 令	变 量	说 明
PORT	6 个数字的标识符	客户选择端口
PASV		服务器选择端口

□ 文件传送命令 这些命令实际上是让用户传送文件。表 21.5 给出了这个组的常用命令。

表 21.5 文件传送命令

命 令	变 量	说 明
RETR	文件名	读取文件；文件从服务器传送到客户
STOR	文件名	存储文件；文件从客户传送到服务器
APPE	文件名	与 STOR 相似，但是若文件存在就必须把数据添加在文件尾部
STOU	文件名	与 STOR 相同，但是文件名在目录中必须唯一
ALLO	文件名	在服务器为文件分配存储空间
REST	文件名	在指明的数据点给文件标记确定位置
STAT	文件名	返回文件的状态

□ 杂项命令 这些命令把信息交付给客户端的 FTP 用户。表 21.6 给出了这个组的常用命令。

表 21.6 杂项命令

命 令	变 量	说 明
HELP		询问关于服务器的信息
NOOP		检查服务器是否工作
SITE	命令	指明特定场所的命令
SYST		询问服务器使用的操作系统

响应

每一个 FTP 命令产生至少一个响应。响应有两个部分：3 位数字的数以及跟随在后面的文本。数字部分定义代码；文本部分定义所需的参数或额外的解释。我们把这 3 个数字记为 xyz。下面描述每一个数字的意义。

第一个数字 第一个数字定义命令的状态。在这个位置上可以使用 5 个数字之一：

- 1yz**（正面初步回答） 动作已经开始。服务器在接受另一个命令之前将发送另一个回答。
- 2yz**（正面完成回答） 动作已经完成。服务器将接受另一个命令。
- 3yz**（正面中间回答） 命令已经接受，但需要进一步的信息。
- 4yz**（过渡负面完成回答） 动作没有发生，但差错是暂时的。同样的命令后面可以发送。
- 5yz**（永久负面完成回答） 命令没有接受，不能够再试。

第二个数字 第二个数字定义命令的状态。在这个位置上可以使用 6 个数字中的一个：

- x0z**（语法）。
- x1z**（信息）。
- x2z**（连接）。
- x3z**（鉴别和账号）。
- x4z**（未指明）。
- x5z**（文件系统）。

第三个数字 第三个数字提供附加信息。表 21.7 给出了可能的响应的简短列表。

表 21.7 响应

代码	说明
正面初步回答	
120	服务不久即将就绪
125	数据连接打开；数据传送不久即将开始
150	文件状态是 OK；数据连接不久即将打开
正面完成回答	
200	命令 OK
211	系统状态或求助回答
212	目录状态
213	文件状态
214	求助报文
215	命名系统类型（操作系统）
220	服务就绪
221	服务关闭
225	数据连接打开
226	关闭数据连接

续表

代码	说明
正面完成回答	
227	进入被动方式；服务器发送它的 IP 地址和端口号
230	用户登录 OK
250	请求文件动作 OK
正面中间回答	
331	用户名 OK；需要口令
332	需要登录账号
350	文件动作在进行中；需要更多的信息
过渡负面完成回答	
425	不能打开数据连接
426	连接关闭；不能识别的命令
450	未采取文件动作；文件不可用
451	动作异常终止；本地差错
452	动作异常终止；存储器不足
永久负面完成回答	
500	语法差错；不能识别的命令
501	参数或变量的语法差错
502	命令未实现
503	不良命令序列
504	命令参数未实现
530	用户未登录
532	存储文件需要账号
550	动作未完成；文件不可用
552	请求的动作异常终止；超过分配的存储器空间
553	未采取请求动作；文件名不允许

21.1.4 文件传送

在控制连接上发送的命令的控制下，文件传送在数据连接上进行。但是，我们应当记住，FTP 的文件传送表示三件事中的一个（见图 21.7）。

- 从服务器把文件复制到客户（下载），这称为读取文件。这是在 RETR 命令的监督下完成的。
- 从客户把文件复制到服务器（上传），这称为存储文件。这是在

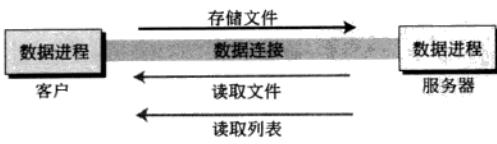


图 21.7 文件传送

STOR 命令的监督下完成的。

- 从服务器向客户发送目录列表或文件名。这是在 LIST 命令的监督下完成的。应注意的是，FTP 把目录或文件名列表当作文件。它在数据连接上发送。

例 21.1

图 21.8 给出使用 FTP 读取目录中的项目列表示例。

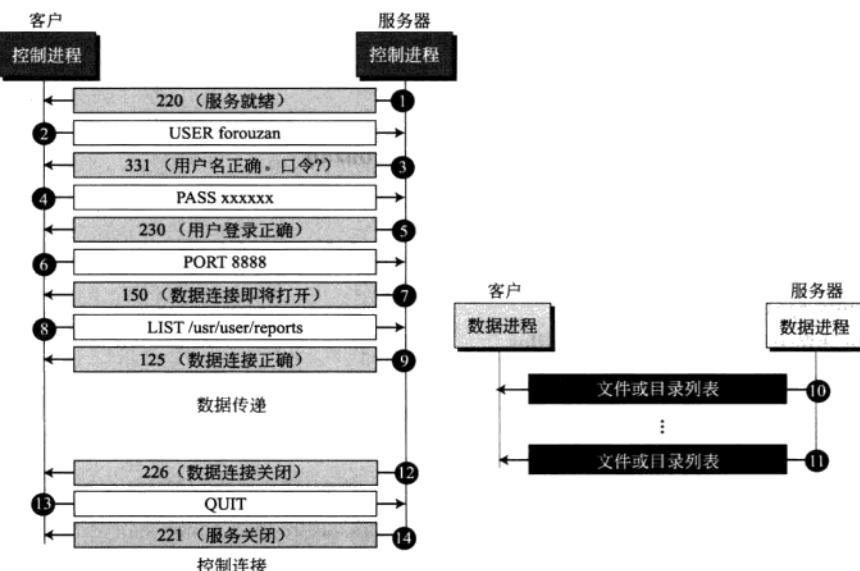


图 21.8 例 21.1

1. 在创建了到端口 21 的控制连接后，FTP 服务器在控制连接上发送 220（服务就绪）响应。
2. 客户发送 USER 命令。
3. 服务器响应 331（用户名正确，需要口令）。
4. 客户发送 PASS 命令。
5. 服务器响应 230（用户登录正确）。
6. 客户在临时端口发出被动打开消息，以便进行数据连接，同时发送 PORT 命令（在控制连接上），把这个端口号发送给服务器。
7. 服务器这时并没有打开这个连接，而是准备在端口 20（服务器端）和从客户收到的临时端口之间，发出主动打开数据连接。它发送响应 150（数据连接即将打开）。
8. 客户发送 LIST 报文。
9. 现在服务器响应 125，打开数据连接。
10. 然后服务器在数据连接上发送文件或目录列表（作为一个文件）。当整个列表（文件）发送出后，服务器在控制连接上响应 226（关闭数据连接）。
11. 客户现在有两个选择。它可以使用 QUIT 命令请求关闭这个控制，或者它可以发送另一个命令开始另一个活动（甚至打开另一个数据连接）。在我们的例子中，客户发送

QUIT 命令。

12. 收到 QUIT 命令后，服务器响应 221（服务关闭），然后关闭控制连接。

例 21.2

下面给出的是一个真实的 FTP 会话，用于与例 21.1 做对比。正常字体是从服务器控制连接来的响应；粗体字是客户发送的命令。灰色背景的黑字表示数据传送。

\$ ftp voyager.deanza.flida.edu

Connected to voyager.deanza.flida.edu

220 (vsFTPD 1.2.1)

530 Please login with USER and PASS.

Name (voyager.deanza.flida.edu:forouzan): forouzan

331 Please specify the password.

Password:

230 Login successful.

Remote system type is UNIX.

Using binary mode to transfer files.

ftp> ls reports

227 Entering Passive Mode (153,18,17,238,169)

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.

例 21.3

图 21.9 给出的示例说明图像（二进制）文件是如何存储的。

1. 在创建了到端口 21 的控制连接后，FTP 服务器在控制连接上发送 220（服务就绪）响应。

2. 客户发送 USER 命令。

3. 服务器响应 331（用户名正确，需要口令）。

4. 客户发送 PASS 命令。

5. 服务器响应 230（用户登录正确）。

6. 客户在临时端口发出为数据连接的被动打开，并发送 PORT 命令（在控制连接上），把这个端口号发送给服务器。

7. 服务器这时并没有打开这个连接，而是准备发出在端口 20（服务器端）和从客户收到的临时端口之间主动打开数据连接。它发送响应 150（数据连接即将打开）。

8. 客户发送 TYPE 命令。

9. 服务器应答响应 200（命令正确）。

10. 客户发送 STRU 命令。

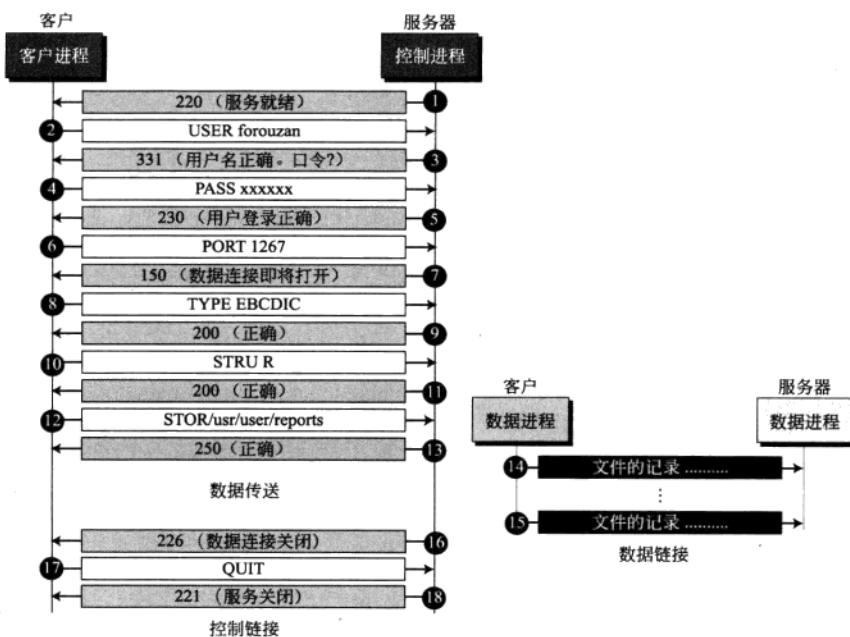


图 21.9 例 21.3

11. 服务器应答响应 200 (命令正确)。
12. 客户发送 STOR 命令。
13. 服务器打开数据连接并发送报文 250。
14. 客户在数据连接上发送文件。当整个文件发完后，数据连接就关闭。关闭数据连接表示文件结束。
15. 服务器在控制连接上发送响应 226。
16. 客户发送 QUIT 命令，或使用其他命令打开另一个数据连接以传送另一个文件。在我们的例子中，发送的是 QUIT 命令。
17. 服务器响应 221 (服务关闭) 并关闭控制连接。

21.1.5 匿名 FTP

要使用 FTP，用户就需要在远程服务器上有账号（用户名）和口令。某些网站有许多可供公众使用的文件。要访问这些文件，用户不需要有账号或口令。因此可以使用 *anonymous*（匿名）作为用户名，使用 *guest* 作为口令。

用户接入到这样的系统是受到限制的。某些网站只允许用户使用命令的一个子集。例如，大多数网站允许用户复制某些文件，但不允许用户任意查找目录。

例 21.4

下面是使用匿名 FTP 的一个例子。我们假定某些公用数据可在 internic.net 上得到。

```
% ftp internic.net
```

```

Connected to internic.net
220 Server ready
Name: anonymous
331 Guest login OK, send "guest" as password
Password: guest
ftp > pwd
257 '/' is current directory
ftp > ls
200 OK
150 Opening ASCII mode
bin
...
...
...
...

```

ftp > close

221 Goodbye

ftp > quit

FTP 的安全性

最初设计 FTP 协议时，安全性并不是一个很严重的问题。尽管 FTP 需要密码，但是密码是以（未加密的）明文发送的。这意味着潜在的攻击者可以截获并使用这些密码。数据传送连接也使用明文来传送数据，这也是不安全的。要使其安全，可以在 FTP 应用层和 TCP 层之间加入安全套接层（见第 30 章）。这样的 FTP 被称为 SSL-FTP。

安全文件传送协议（sftp）程序

另一种使用安全管道传送文件的方式是使用另一个被称为 sftp（安全文件传送协议）的独立协议。它实际上是 UNIX 中的一个被称为 sftp 的程序，这个程序是 SSH 协议的一部分（见第 20 章）。当 SSH 在 SSH 客户和 SSH 服务器之间建立了一个安全连接之后，能够使用这种链接（多路复用）的应用程序之一就是 sftp。换言之，sftp 是 SSH 应用组件的一部分。Sftp 程序是交互式程序，它类似于 FTP，通过使用一组接口命令在 SSH 客户和 SSH 服务器之间传送文件。

21.2 简单文件传送协议（TFTP）

有时我们只需要复制一个文件而不需要 FTP 协议的全部功能。例如，当无盘工作站或路由器在被引导时，我们需要下载引导和配置文件。这里我们并不需要 FTP 全部的复杂功能，只需要一个能够迅速复制这些文件的协议。

简单文件传送协议（TFTP）就是为传送这些文件而设计的。它很简单，以致其软件包能够放入无盘工作站的只读存储器中。它可用于引导时。它能够放入 ROM 是因为它只需要基本的 IP 和 UDP。但是，TFTP 没有安全措施。TFTP 可以为客户读或写文件。读表示

从服务器端把文件复制到客户端。写表示从客户端把文件复制到服务器端。

TFTP 在熟知端口 69 上使用 UDP 服务。

21.2.1 报文

TFTP 共有 5 种类型的报文，RRQ，WRQ，DATA，ACK 和 ERROR，如图 21.10 所示。



图 21.10 报文的种类

RRQ

读请求（RRQ）报文由客户使用，用来建立一条从服务器读数据的连接。它的格式如图 21.11 所示。

OpCode = 1	文件名	全为 0	方式	全为 0
2 字节	可变	1 字节	可变	1 字节

图 21.11 RRQ 报文的格式

RRQ 报文的各字段如下：

- **OpCode** 这第一个字段是 2 字节的操作码。对 RRQ 报文这个值是 1。
- **文件名** 下一个字段是定义文件名的可变长度的字符串（用 ASCII 编码）。因为文件名是可变长度的，因此在结束时要有全 0 的 1 字节字段作为记号。
- **方式** 下一个字段是另一个定义传送方式的可变长度的字符串。方式字段用另一个全 0 的 1 字节字段作为结束。方式可以是两种字符串中的一种：“netascii”（对于 ASCII 文件）或“octet”（对于二进制文件）。文件名和方式字段可以使用大写或小写或大小写的组合。

WRQ

写请求（WRQ）报文由客户使用，用来建立一条把数据写到服务器的连接。它的格式与 RRQ 相同，除了 OpCode 是 2（见图 21.12）。

OpCode = 2	文件名	全为 0	方式	全为 0
2 字节	变量	1 字节	变量	1 字节

图 21.12 WRQ 报文的格式

DATA

数据（DATA）报文由客户或服务器使用，用来传送数据块。它的格式如图 21.13 所示。DATA 报文的各字段如下：

OpCode=3	块数	数据
2 字节	2 字节	0~512 字节

图 21.13 DATA 报文的格式

- **OpCode** 第一个字段是 2 字节的操作码。对 DATA 报文这个值是 3。
- **块号** 这个 2 字节字段包含块号。数据的发送端（客户或服务器）使用这个字段安排序号。所有的块都用数字顺序编号，从 1 开始。我们将立刻会看到，编号对确认是必要的。
- **数据** 在所有的 DATA 报文中，这个块必须准确地等于 512 字节，但最后一块必须在 0~511 字节之间。一个非 512 字节的块用做一个信号，表示发送端已经发完了所有的数据。换言之，它用做文件结束的指示符。若文件中的数据碰巧是 512 字节的整数倍，那么发送端必须再发送一个具有 0 字节的额外的块以表示传输的结束。数据可用 NVT ASCII (netascii) 或二进制八位组(octet) 来传送。

ACK

确认 (ACK) 报文由客户或服务器使用，用来确认收到数据块。这个报文只有 4 字节长。它的格式如图 21.14 所示。

ACK 报文的各字段如下：

- **OpCode** 第一个字段是 2 字节的操作码。对 ACK 报文这个值是 4。
- **块号** 下一个字段是 2 字节字段，它包含收到的块号。

ACK 报文可以是 WRQ 报文的响应。服务器发送这个报文指示它已经准备好接收来自客户的数据。在这种情况下块号字段的值是 0。ACK 报文的例子在后面一节给出。

ERROR

ERROR 报文由客户或服务器在一条连接不能建立或在数据传输中出现问题时使用。它可以作为对 RRQ 或 WRQ 的负面响应。它还可在真正的数据传送阶段中如果下一个块不能发送时使用。差错报文不能用于对受损伤或重复的报文声明。这些问题由本章后面要讨论的差错控制机制来解决。ERROR 报文的格式如图 21.15 所示。

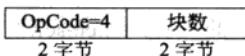


图 21.14 ACK 报文的格式

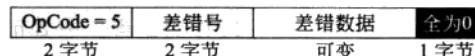


图 21.15 ERROR 报文的格式

ERROR 报文的各字段如下：

- **OpCode** 第一个字段是 2 字节的操作码。对 ERROR 报文来说这个值是 5。
- **差错号** 这个 2 字节字段定义差错的类型。表 21.8 给出了差错号及其相应的意义。

表 21.8 差错号及其意义

差错号	意 义
0	未定义
1	文件未找到
2	存取被破坏
3	磁盘满或磁盘上的份额超过
4	非法操作
5	未知的端口
6	文件已存在
7	无此用户

- **差错数据** 这个可变字节字段包含原文的差错数据，并以全 0 的 1 字节字段结束。

21.2.2 连接

TFTP 使用 UDP 服务。因为在 UDP 中不提供连接建立和终止，UDP 在传送每一个数据块时，把它封装在独立的用户数据报中。但是，在 TFTP 中，我们并不是希望仅传送一个数据块；我们也不希望把文件作为许多独立的数据块传送。我们希望传送的数据块能够连接在一起，因为它们属于同一个文件。TFTP 使用 RRQ、WRQ、ACK 和 ERROR 报文来建立连接。它使用小于 512 字节（0~511）的数据块来终止连接。

连接建立

用于读文件的连接建立与用于写文件的连接建立不同（见图 21.16）。

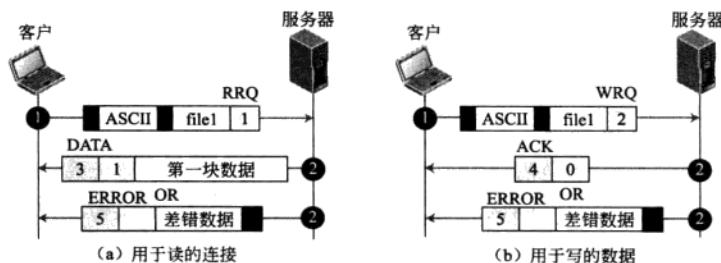


图 21.16 连接建立

- **读** 要建立读的连接，TFTP 客户发送 RRQ 报文。文件名和传输方式都定义在这个报文中。若服务器能传送这个文件，它就正面响应 DATA 报文，并包含第一个数据块。若有问题，如打开有困难或受限制不准许打开，则服务器发送 ERROR 报文作为负面响应。
- **写** 要建立写的连接，TFTP 客户发送 WRQ 报文。文件名和传输方式都定义在这个报文中。若服务器能接受该文件的副本，则发送 ACK 报文作为正面响应，使用的块号为 0。若有问题，则服务器发送 ERROR 报文作为负面响应。

连接终止

在整个文件传送完后，必须终止连接。如以前所述的，TFTP 并没有使用特殊报文作为终止。终止就是用发送最后的数据块（即必须小于 512 字节）来完成的。

21.2.3 数据传送

数据传送阶段是在连接建立和连接终止之间发生的。TFTP 使用 UDP 服务，它是不可靠的。

文件划分为若干个数据块，除最后一块外，每一块是准确的 512 字节。最后一块必须

在 0~511 字节之间。TFTP 可传送 512 字节或二进制格式。

UDP 没有任何流量控制和差错控制机制。TFTP 必须创建流量控制和差错控制机制，以便传送由连续数据块构成的文件。

流量控制

TFTP 使用 DATA 报文发送数据块，并等待 ACK 报文。若在超时之前发送端就收到了确认，它就发送下一个块。这样，实现流量控制的方法是给数据块编号和在发送下一个数据块之前等待 ACK。

读取文件 当客户打算读取（读）文件时，它就发送 RRQ 报文。服务器响应 DATA 报文，发送第块号为 1 的数据块（若无问题）。

存储文件 当客户打算存储（写）文件时，它就发送 WRQ 报文。服务器响应块号为 0 的 ACK 报文（若无问题）。在收到这个确认后，客户使用块号 1 发送第一个数据块。

差错控制

TFTP 的差错控制机制与其他协议的不同。它是对称的，即发送端和接收端都使用超时。发送端为数据报文使用超时；接收端为确认报文使用超时。若丢失了数据报文，在超时到期时发送端就进行重传。若丢失了确认报文，在超时到期时接收端就进行重传。这样就保证了平滑的操作。

差错控制在 4 种情况下是需要的：报文受损伤、报文丢失、确认丢失和报文重复。

报文受损伤 没有负面的确认。若数据块受到损伤，接收端就检测出来并丢弃这个数据块。发送端等待确认，但在超时期间内不会收到确认。这个数据块将再发送一次。请注意，在 TFTP 的 DATA 报文中没有检验和字段。接收端可用于检测数据受到损伤的唯一方法，是通过 UDP 用户数据报的检验和字段。

报文丢失 若数据块丢失了，它就永远不能到达接收端，而确认也不会发出。在超时之后发送端重新发送这个数据块。

确认丢失 若确认丢失了，则可能发生两种情况。若接收端的计时器比发送端的计时器先到期，则接收端重传确认；否则，发送端重传这个数据。

报文重复 接收端通过块号可检测出数据块的丢失。若数据块重复了，接收端就简单地将其丢弃。

巫士徒弟的错误

虽然在 TFTP 中流量控制和差错控制机制是对称的，但仍会出现一种问题，称为巫士徒弟的错误（sorcerer's apprentice bug），这个名称是来自一个动画人物，这个人不经心地对一个拖把施加魔法，结果这个拖把连续地复制它自己。若分组的 ACK 报文没有丢失而是延迟了，则会发生这种情况。这时，每一个成功的数据块要发送两次，而每一个成功的确认也要接收两次。

图 21.17 说明了这一问题。对第五个数据块的确认延迟了。在超时到期后，发送端重传第五个数据块，它将再次被接收端确认。发送端接收了两个对第五个数据块的确认，这又引起它发送第六个数据块两次。接收端收到第六个数据块两次，它也发送两个确认，而这又引起第七个数据块发送端两次。以下继续下去。

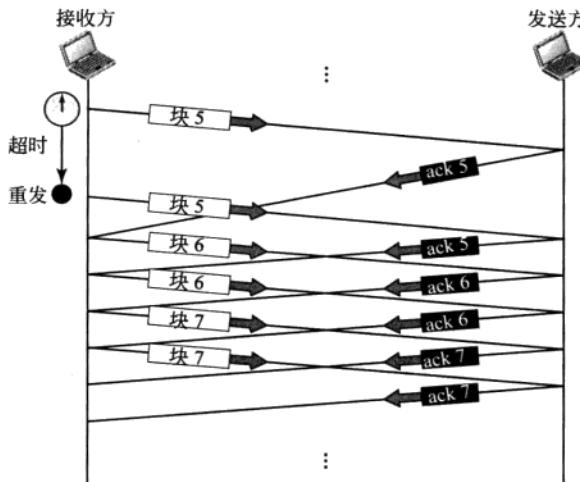


图 21.17 巫士徒弟的错误

21.2.4 UDP 端口

当进程使用 UDP 服务时，服务器进程就在熟知端口上发出被动打开命令，等待客户进程在临时端口上发出主动打开命令。在连接建立后，客户和服务器就使用这两个端口进行通信。

这种情况在 TFTP 中是不同的。在 TFTP 客户和 TFTP 服务器之间的通信可能持续很长的时间（几秒钟或几分钟）。若 TFTP 服务器使用熟知端口 69 和客户进行长时间的通信，那么在这段时间内就没有其他的客户能够使用这样的服务。解决这问题的方法是使用熟知端口进行初始连接，但对剩下的通信则使用临时端口（见图 21.18）。

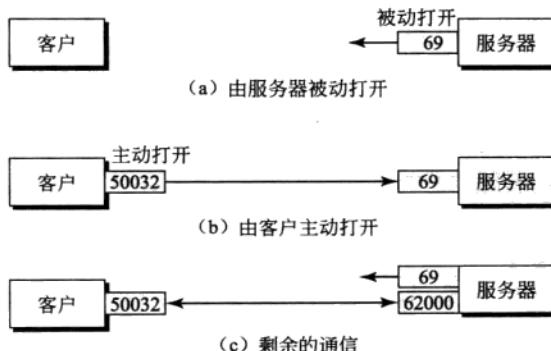


图 21.18 TFTP 使用的 UDP 端口号

步骤如下：

1. 服务器使用熟知端口 69 被动打开连接。
2. 客户主动打开连接，它使用临时端口作为源端口而熟知端口 69 作为目的端口。它

使用 RRQ 报文或 WRQ 报文做到这点。

3. 服务器主动打开连接，它使用新的临时端口作为源端口，而使用收到的来自客户的临时端口作为目的端口。它使用这些端口发送 DATA 或 ACK 或 ERROR 报文。服务器腾出熟知端口（69）使其他客户能够使用。当客户收到来自服务器的第一个报文时，它就使用自己的临时端口和服务器发送的临时端口进行今后的通信。

21.2.5 TFTP 举例

在图 21.19 中给出了 TFTP 的例子。客户打算读取名叫 file1 的 2000 字节的文件的内容。客户发送 RRQ 报文。服务器发送第一个报文，携带第一个 512 字节，它原封不动地被接收和确认了。这两个报文是连接建立的报文。携带第二个 512 字节的数据块丢失了。超时后，服务器重传这个数据块，它被收到了。携带第三个 512 字节的数据块原封不动地收到了，但确认丢失了。超时后，接收端重传确认。携带剩下 464 字节的最后一个数据块收到了，但受到损伤，因此客户简单地把它丢弃。超时后，服务器重传这个数据块。这个报文被认为是连接终止，因为这个数据块携带小于 512 字节的数据。

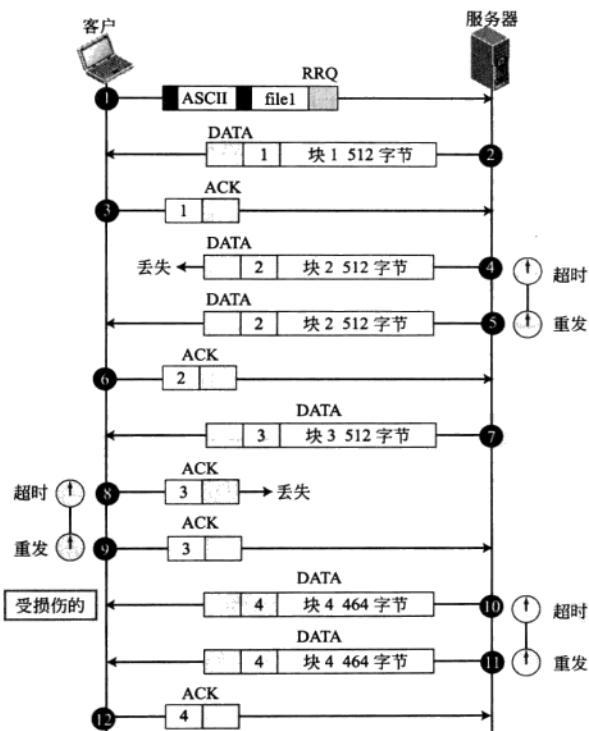


图 21.19 TFTP 举例

21.2.6 TFTP 选项

对 TFTP 协议已经提出了一种扩充方案，该方案允许在 RRQ 和 WRQ 报文后面附加一些选项。选项主要是用于协商数据块的大小和可能协商的初始序号。在没有选项时，数据块的大小除最后一块外都是 512 字节。协商能够定义具有任意字节数的数据块大小，只要这个报文能够封装成一个 UDP 用户数据报。

已经提出了一种称为选项确认（OACK）的新的报文类型，它可以使另一方接受或拒绝这些选项。

21.2.7 安全性

关于 TFTP 我们必须记住的一个重要问题就是它不提供安全性：没有用户标识或口令。但在今天却必须要很小心地防止黑客对文件的存取。要做到这点的一种方法是仅限于将非关键文件用于 TFTP 的使用。要达到在最小安全性的一种方法是在靠近 TFTP 服务器处的路由器上实现安全机制，这个路由器只允许某些主机接入 TFTP 服务器。

21.2.8 应用

当安全问题不太大时，TFTP 用于基本的文件传送是非常有用的。它可以用于初始化一些设备，如网桥或路由器。它的主要应用是和 DHCP 协议结合在一起使用的。TFTP 只需要很少量的存储器，同时仅使用 UDP 和 IP 的服务。它很容易地配置在 ROM（或 PROM）中。当工作站加电后，TFTP 就连接到一个服务器，就可以从这个服务器下载配置文件。图 21.20 给出了这一概念。加了电的工作站使用 DHCP 客户从 DHCP 服务器获得配置文件名。工作站再把文件名传递给 TFTP 客户，以便从 TFTP 服务器处得到配置文件的内容。

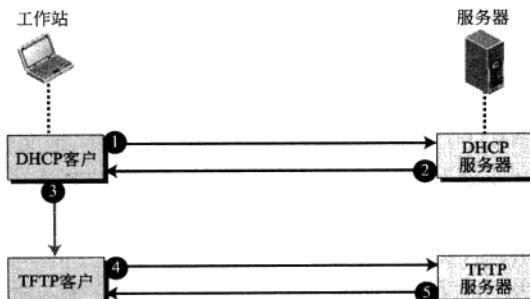


图 21.20 使用 DHCP 的 TFTP

21.3 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。被方括号括起来的书

目可以在本书末尾的参考书目清单中找到。

21.3.1 参考书

有几本书和 RFC 深入浅出地覆盖了有关 FTP 和 TFTP 的内容，包括[Com 06]、[Mir 07]以及[Ste 94]。

21.3.2 RFC

有几份 RFC 给出了有关 FTP 的更新内容，它们包括 RFC 959、RFC 2577 和 RFC 2585。有关 TFTP 的更多信息可以参考 RFC 906、RFC 1350、RFC 2347、RFC 2348 和 RFC 2349。

21.4 重要术语

匿名 FTP	流量控制
ASCII 文件	图像文件
块方式	读
压缩方式	记录结构
连接建立	巫士徒弟的错误
控制连接	安全文件传送协议（SFTP）
数据连接	流方式
EBCDIC 文件	简单文件传送协议（TFTP）
文件结构	写
文件传送协议（FTP）	

21.5 本章小结

- 文件传送协议（FTP）是客户-服务器应用程序，用来把文件从一个主机复制到另一个主机。FTP 需要两条连接用于数据传送：一条控制连接和一条数据连接。FTP 在两个不相同的系统之间的通信使用 NVT ASCII。在真正传送文件之前，客户要通过控制连接定义文件类型、数据结构和传输方式。
- 客户发送的用来与服务器建立通信的命令共有 6 类：接入命令，文件管理命令，数据格式化命令，端口定义命令，文件传送命令，杂项命令。文件传送的类型共有 3 种：从服务器把文件复制到客户，从客户把文件复制到服务器，从服务器向客户发送目录或文件名列表。
- 使用 FTP 传送文件并不安全。一种解决方法是在 FTP 应用层和 TCP 层之间加入安全套接字层（SSL）。另一种方法是使用一个完全独立的文件传送应用程序 sftp。sftp 是 SSH 协议的一种应用。

- 简单文件传送协议（TFTP）是简单的文件传送协议，它没有 FTP 那样的复杂和完善。客户使用 TFTP 的服务来读取文件的副本，或把文件的副本发送到服务器。TFTP 报文的类型共有 5 种：RRQ，WRQ，DATA，ACK 和 ERROR。TFTP 可以和 DHCP 一起使用，以便下载配置文件对设备进行初始化。
- 在 TFTP 中，有 4 种情况下需要使用差错控制：报文受损伤、报文丢失、确认丢失和报文重复。巫士徒弟的错误是由 TFTP 的流量控制和差错控制机制引起的确认报文和数据报文的重复。

21.6 实践安排

21.6.1 习题

1. 当 FTP 正在传送数据时突然其控制连接出现严重故障，你认为会出现什么问题？
2. 试解释为什么客户在控制连接上发出主动打开命令而在数据连接上发出被动打开命令？
3. 为什么对匿名 FTP 会有些限制？一个不讲道德的用户可能会怎样做？
4. 试解释为什么 FTP 没有规定报文格式？
5. 试给出携带 FTP 命令的 TCP 报文段。
6. 试给出携带一种 FTP 响应的 TCP 报文段。
7. 试给出携带 FTP 数据的 TCP 报文段。
8. 试解释若例 21.2 中的文件已经存在时将会发生什么？
9. 试使用 PASV 而不是 PORT 命令重做例 21.1。
10. 试使用 STOU 而不是 STOR 命令来存储具有唯一名称的文件，重做例 21.2。若已经有一个同名文件存在，会发生什么？
11. 试使用 RETR 而不是 STOR 命令来读取一个文件，重做例 21.2。
12. 试给出使用 HELP 命令的例子。
13. 试给出使用 NOOP 命令的例子。
14. 试给出使用 SYST 命令的例子。
15. 一用户打算在目录 /usr/usrs/letters 下构造名叫 Jan 的目录。主机称为“mcGraw.com”。使用例 21.1 和例 21.2 作为参考，试给出所有的命令和响应。
16. 一用户打算把他的当前目录移动到其父目录下。主机称为“mcGraw.com”。使用例 21.1 和例 21.3 作为参考。试给出所有的命令和响应。
17. 一用户打算把在目录 /usr/usrs/report 下的名叫 file1 的文件移到目录 /usr/usrs/letters 下。主机称为“mcGraw.com”。使用例 21.1 和例 21.2 作为参考，试给出所有的命令和响应。
18. 一用户打算读取目录 /usr/usrs/report 下的名叫 file1 的 EBCDIC 文件。主机称为“mcGraw.com”。该文件很大，因此用户在传送之前要使用压缩。使用例 21.1 和例 21.2 作为参考，试给出所有的命令和响应。

19. 在 TFTP 中我们为什么需要 RRQ 或 WRQ 报文，而在 FTP 中不是这样？
20. 试说明如何把 RRQ 报文封装在 UDP 用户数据报中。假设文件名是“Report”而方式是 ASCII。UDP 数据报的长度是多少？
21. 试说明如何把 WRQ 报文封装在 UDP 用户数据报中。假设文件名是“Report”而方式是 ASCII。UDP 数据报的长度是多少？
22. 试说明如何把携带块号为 7 的 TFTP 数据报文封装在 UDP 用户数据报中。数据报的总长度是多少？
23. 主机 A 使用 TFTP 从主机 B 读取 2150 字节的数据。
 - a. 试给出所有的 TFTP 命令，包括需要进行连接建立和连接终止的命令。假定无差错。
 - b. 试给出两个主机之间交换的所有用户数据报。
24. 重做习题 23，但假定第二个数据块出现差错。同时给出两个主机之间交换的所有用户数据报。

21.6.2 研究活动

25. 试找出路由器怎样为 TFTP 使用安全机制的。
26. 试使用 UNIX 或 Windows 找出 FTP 使用的所有命令。
27. 试使用 UNIX 或 Windows 找出 TFTP 使用的所有命令。
28. 试找出已经提出的 OACK 报文格式。
29. 试找出建议附加在 RRQ 和 WRQ 报文后面的选项类型。

第 22 章 万维网和 HTTP

万维网（World Wide Web, WWW）是遍及全世界且相互链接起来的信息储藏所。WWW 具有独特的灵活性、可移植性和用户友好的组合特性，它和因特网提供的其他服务都不同。WWW 项目最初是由 CERN（欧洲粒子物理实验室）发起的，是为了创建能够处理分布式资源的、供科学用的研究系统。在本章中，我们先讨论有关万维网的问题，然后再讨论 HTTP 协议，它用来从万维网读取信息。

目标

本章有以下几个目标：

- 讨论 WWW 的体系结构并描述超文本和超媒体的概念。
- 描述 Web 客户，Web 服务器及其组成部分。
- 定义 URL 作为标识 Web 服务器的工具。
- 介绍三种不同的 Web 文件：静态文档，动态文档和活动文档。
- 讨论 HTTP 及其事务。
- 定义并列出请求报文中的字段。
- 定义并列出响应报文中的字段。
- 定义 HTTP 中的持续连接和非持续连接。
- 介绍 Cookie 及其在 HTTP 中的应用。
- 讨论 Web 缓存，它的应用以及更新缓存的方法。

22.1 体系结构

今天的 WWW 是分布式的客户-服务器服务，其中的客户用浏览器就能够得到服务器提供的服务。但是，这种服务的提供者分布在许多称为网站(site)的地方。每一个网站保存有一个或多个文档，称为 Web 页面。每一个 Web 页面可以包含到同一个或其他网站的其他 Web 页面的链接。换言之，每一个 Web 页面可以是简单页面或是复合页面。简单 Web 页面不包含到其他 Web 页面的链接。复合 Web 页面包含一个或多个到其他 Web 页面的链接。每一个 Web 页面即是一个包含名称及地址的文件。

例 22.1

假使我们要读取一个含有名人传记的 Web 页面，传记中还包含一些图片，而这些图片内嵌在页面中。由于这些图片不是储存为单独的文件，因此整个文档就是一个简单 Web 页

面。如图 22.1 所示，我们可以使用一个单一请求/响应事务来读取这个页面。

例 22.2

现在假设我们要读取一个科学文档，其中包含一个到其他文本文件的引用，还包含一个到大型图片的引用。这种情况如图 22.2 所示。

主文档和图片分别储存在同一网站的两个不同文件中（文件 A 和文件 B）。引用的文本文件储存在另一个网站（文件 C）。由于我们要处理三个不同的文件，因此要看到整个文档需要有三个事务。第一个（请求/响应）事务可以取得主文档（文件 A）的一个拷贝，其中包含到第二个和第三个文件的引用（指针）。当这个主文档的拷贝被读取及浏览时，用户可以点击指向图片的引用。这会触发第二个事务来读取图片（文件 B）的拷贝。如果用户想更进一步查看引用的文本文件内容，她可以点击这个文本文件的引用（指针）来引发第三个事务并由此取得文件 C 的拷贝。注意尽管文件 A 和 B 都储存在网站 1，它们是两个相互独立的文件，各自有不同的名称和地址。读取它们需要两个事务。

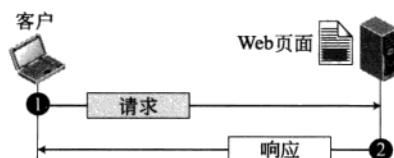


图 22.1 例 22.1

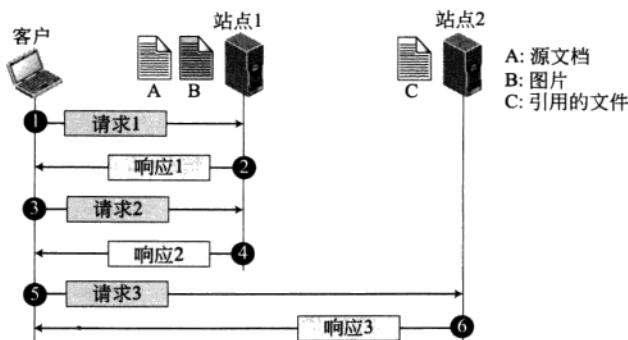


图 22.2 例 22.2

例 22.3

我们需要记住的非常重要的一点是例 22.2 中的文件 A、B 和 C 是相互独立的 Web 网页，各自有着独立的名称和地址。尽管文件 A 包含指向文件 B 和 C 的引用，这并不是说这三个文件不能被单独读取。第二个用户可以用一个事务来读取文件 B。而第三个用户可以用一个事务来读取文件 C。

22.1.1 超文本和超媒体

前面三个例子演示了超文本（Hypertext）和超媒体（Hypermedia）的概念。超文本是指创建的文档包含有指向其他文档的引用。在超文本文档中，部分文本可以被定义为到其他文本的链接。当在浏览器中浏览超文本时，可以通过点击这个链接来取得其他的文档。当文档包含到其他文本文档的链接或图片、视频、音频时，我们称之为超媒体。

22.1.2 Web 客户（浏览器）

许多厂商提供商用浏览器，可以解释和显示 Web 页面，而几乎所有的浏览器都采用同样的体系结构。每一个浏览器通常由 3 个部分组成：控制程序、客户协议及解释程序（见图 22.3）。

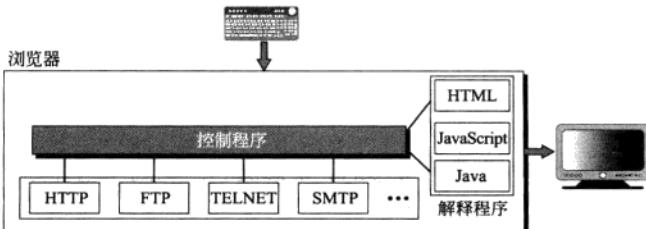


图 22.3 浏览器

控制程序从键盘或鼠标接收输入，使用客户程序访问要浏览的文档。在文档找到后，控制程序就使用某个解释程序在屏幕上显示文档。客户协议可以是前面提到过的某个协议，如 FTP 或 TELNET 或 HTTP（后面还要介绍）。解释程序可以是 HTML、Java 或 JavaScript，这取决于文档的类型。在本章的后面我们将讨论这些基于文档类型的解释程序。常见的商用浏览器包括 Internet Explorer、Netscape Navigator 和 Firefox。

22.1.3 Web 服务器

Web 页面存储在服务器上。每当有客户请求到达时，对应的文档就发送给客户。为了提高效率，服务器通常在其高速缓存的存储器中存储被请求过的文档；对存储器进行访问要比磁盘快得多。通过多线程或多进程可使服务器的效率更高。在这种情况下，服务器在同一时间可回答多个请求。常见的服务器有 Apache 和微软的 Internet Information Server。

22.1.4 统一资源定位符（URL）

客户要访问 Web 页面就需要文件名称和地址。为了方便地访问在世界范围的文档，HTTP 使用定位符。统一资源定位符（URL）是在因特网上指明任何种类的信息的标准。URL 定义了 4 样东西：协议、主机、端口和路径（见图 22.4）。



图 22.4 URL

协议是客户-服务器程序，用来读取文档。许多不同的协议可用来读取文档，其中有 Gopher、FTP、News 和 TELNET。现在最常用的是 HTTP。

主机是信息所存放的地点的域名。Web 页面通常存放在计算机上，而这个计算机通

常使用以字符“WWW”开始的域名别名。但这不是强制性的，因为主机可以使用任何域名。

URL 可以有选择地包含服务器的端口号。如果包含了端口，那么端口就插入在主机和路径之间，和主机用冒号分隔开。

路径是信息存放的路径名。请注意，路径本身可以包含斜线，在 UNIX 操作系统中斜线把目录与子目录和文件分隔开。换言之，路径定义文档在目录系统中存放的完整文件名。

22.2 Web 文档

WWW 里的文档可分为 3 大类：静态的、动态的和活动的。这种分类基于文档内容被确定的时间。

22.2.1 静态文档

静态文档是固定内容的文档，它在服务器中创建，并存储在服务器中。客户只能得到文档的一个副本。换言之，文件的内容是在文件被创建而不是在使用时确定的。当然，在服务器中的内容是可以改变的，但用户不能改变它。当客户访问文档时，文档的一个副本就被发送出去。用户可以使用浏览器显示这个文档（见图 22.5）。

静态文档可用众多语言中的一种来准备。这些语言包括超文本置标语言（HTML）、可扩展置标语言（XML）、可扩展样式语言（XSL）和扩展超文本置标语言（XHTML）。我们会在附录 E 中讨论这些语言。

HTML、XML、XSL 和 XHTML 在附录 E 中讨论。

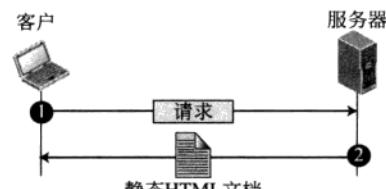


图 22.5 静态文档

22.2.2 动态文档

动态文档是在浏览器请求该文档时才由 Web 服务器创建。当请求到达时，Web 服务器就运行创建动态文档的应用程序或脚本。服务器返回这个程序或脚本的输出，把它作为对请求该文档的浏览器的响应。因为对每一个请求都创建出新的文档，因此每一个请求得到的动态文档的内容就会不同。一个非常简单的例子就是从服务器得到时间和日期的动态文档。时间和日期是一种动态信息，它们时刻在变化。客户可以请求服务器在 UNIX 下运行叫做 date 的程序，并把程序的结果发送给客户。

通用网关接口（CGI）

通用网关接口（CGI）是一种创建和处理动态文档的技术。CGI 是一组标准，它定义动态文档应如何写，输入数据应如何加到程序上，以及输出结果应如何使用。

CGI 并不是一种新的语言；相反，它允许程序员使用许多语言中的任何一种，如 C、

C++、Bourne Shell、Korn Shell、C Shell、Tcl 或 Perl。CGI 唯一定义的是程序员必须遵循的一组规则和术语。

CGI 中的“通用”指出这个标准所定义的一组规则对任何语言或平台都与通用的。术语“网关”在这里表示 CGI 可用来访问其他资源，如数据库、图形软件包，等等。术语“接口”在这里表示一些预定义的术语、变量、调用等等都可以在任何 CGI 程序中使用。CGI 程序的一种最简单形式是用支持 CGI 的一种语言编写的代码。任何程序员，只要他能够把一系列思想编码成为一个程序，同时知道上述语言中的一种的语法，就能够编写简单的 CGI 程序。图 22.6 说明了使用 CGI 技术创建动态程序的步骤。

输入 在传统的编程中，当程序执行时，参数可以传递给程序。参数传递可以使程序员编写能够使用在不同情况下的类属程序(generic program)。例如，类属复制程序可以编写成把任何文件复制到另一个文件。用户使用这个程序把名为 *x* 的文件复制为另一个名为 *y* 的文件时，需要传递 *x* 和 *y* 作为参数。

从浏览器输入到服务器是使用表单(form)来发送的。若表单中的信息很少(如一个字)，则它可附加在 URL 的后面的问号之后。例如，下面的 URL 携带表单信息(23，一个值)：

<http://www.deanza/cgi-bin/prog.pl?23>

当服务器收到这个 URL 时，它就使用在问号前面的 URL 部分来访问要运行的程序，并将问号后面的部分(23)解释为客户发送的输入。服务器把这个字符串存储在一个变量中。当 CGI 程序执行时，它就能够访问这个值。

若从浏览器来的输入太长而无法放入查询字符串中，浏览器就可以请求服务器发送表单。浏览器就将输入数据填入该表单，并把它发送给服务器。在表单中的信息可用作给 CGI 程序的输入。

输出 CGI 整个的思想就是在服务器端执行 CGI 程序，并把输出发送给客户(浏览器)。输出通常是普通正文或 HTML 结构的正文；但是，输出也可以是其他形式的东西。它可以是图形、二进制数据、状态码、给浏览器用来将结果放入高速缓存的指令，或给服务器用来发送现有文档(而不是真正的输出)的指令。

要告诉客户所发送的文档类型，CGI 程序必须创建首部。事实上，CGI 程序的输出总是包括两个部分：首部和主体。首部和主体之间用空行隔开。这就表示，任何 CGI 程序首先要创建首部，然后是空行，然后是主体。虽然这个首部和空行在浏览器的屏幕上并不显示，浏览器要使用这个首部来解释主体。

动态文档的脚本技术

CGI 技术的问题是低效率，其原因是当要创建的文档的一部分是固定的，并不随着请求变化。例如，假定我们需要检索一个特定牌子的轿车的备份零件表、它们的供货情况以及价格。虽然供货情况和价格会随时间变化，但零件的名字、描述和照片则是固定不变的。如果我们使用 CGI，那么对每一次的请求这个程序必须生成整个的文档。解决这个问题的

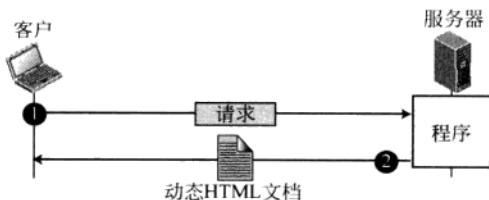


图 22.6 使用 CGI 的动态文档

方法就是使用 HTML 来生成包含固定部分的文件，同时嵌入一个脚本源代码，它可以由服务器来运行以便提供变化的供货情况和价格。图 22.7 表示这种概念。

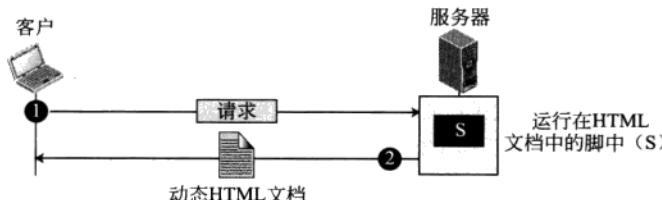


图 22.7 使用服务器端脚本的动态文档

在使用脚本生成动态文档方面已经有了一些技术。其中最常用的是超文本预处理器 (PHP)，它使用 Perl 语言；JSP (Java Server Pages)，它使用 Java 语言进行编排；ASP (Active Server Pages)，这是一个微软的产品，它使用 Visual Basic 语言进行编排；以及 ColdFusion，它在 HTML 文档中嵌入了 SQL 数据库查询。

动态文档有时称为服务器端动态文档。

22.2.3 活动文档

对于许多应用，我们需要程序能够在客户端运行。这就称为**活动文档**。例如，设想我们要运行在屏幕上产生动画图形或与用户进行交互的程序。毫无疑问，这个程序应当在客户端运行，即在动画和交互发生的地方运行。当浏览器请求活动文档时，服务器就发送这个文档的一个副本或脚本。然后这个文档就在客户（浏览器）端运行。

Java 小应用程序

产生活动文档的一种方法是使用**Java 小应用程序**。**Java** 是高级编程语言、运行时间环境以及类库的组合，它允许编程者编写活动文档，然后由浏览器运行它。它也可以是一个不使用浏览器的单独程序。

小应用程序是在服务器端用 Java 编写的程序。它编译后就可以运行。这个文档的格式是字节码（二进制）。客户进程（浏览器）创建这个小应用程序的一个实例，然后就运行它。Java 小应用程序可以在浏览器端以两种形式运行。第一种方法是浏览器在 URL 中直接请求 Java 小应用程序，并以二进制形式接收这个小应用程序。第二种方法是浏览器可以读取并运行已嵌入小应用程序地址（作为一个标记）的 HTML 文件。图 22.8 表示在第一种方法中 Java 小应用程序的使用；第二种方法是相似的，但需要两个事务（transaction）。

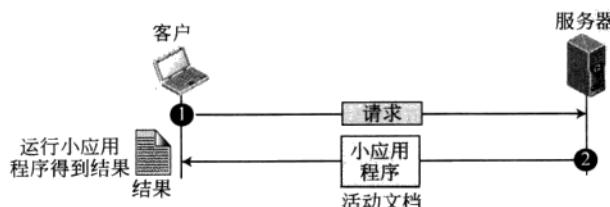


图 22.8 使用 Java 小应用程序的活动文档

JavaScript

在动态文档中的脚本概念也可以用到活动文档中。如果在文档中的活动部分不大，那么它就可以用脚本语言来编写；然后它就可以在客户端在同一时间被解释和运行。脚本是源代码（文本）而不是二进制形式。在这种情况下的脚本技术通常是 JavaScript。JavaScript 与 Java 相似，是为此目的开发的一个非常高级的脚本语言。图 22.9 表示了 JavaScript 怎样用来创建活动文档。

活动文档有时称为客户端动态文档。

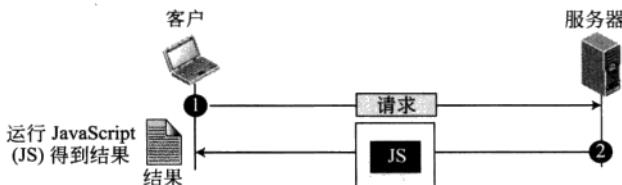


图 22.9 使用客户端脚本的活动文档

22.3 HTTP

超文本传送协议（HTTP）是主要用在万维网上存取数据的协议。HTTP 的功能像是 FTP（见第 21 章）和 SMTP（见第 23 章）的组合。它与 FTP 相似是因为它能够传送文件并使用 TCP 连接。但是，它比 FTP 简单得多，因为它只使用一条 TCP 连接。没有独立的控制连接；在客户和服务器之间只有数据传送。

HTTP 与 SMTP 相似是因为在客户和服务器之间传送的数据看起来很像 SMTP 报文。此外，报文的格式受类似于 MIME 的首部控制。但是，HTTP 与 SMTP 不同的地方是，HTTP 报文并非为了给人看的：这些报文用于 HTTP 服务器和 HTTP 客户（浏览器）读取和解释。SMTP 报文采用存储转发方式，但 HTTP 报文是立即交付。从客户到服务器的命令嵌入在请求报文中。所请求的文件内容或其他信息则嵌入在响应报文中。HTTP 在熟知端口 80 上使用 TCP 的服务。

HTTP 在熟知端口 80 上使用 TCP 的服务。

22.3.1 HTTP 事务

图 22.10 说明了客户和服务器之间的 HTTP 事务。虽然 HTTP 使用 TCP 的服务，但 HTTP 本身是无状态的协议。也就是说服务器不会保存有关客户的信息。客户发送请求报文来初始化这个事务。服务器发送响应进行回答。

请求报文

请求报文的格式在图 22.11 中给出。请求报文包括一个请求行、一个首部，有时还有一个主体。

请求行 在请求报文中的第一行是**请求行**；请求行共有 3 个字段，如图 22.11 所示，

它们以字符定界符分隔。这 3 个字段分别称为方法、URL 和版本。它们以空格符为分隔。在该行最后，以回车和换行这两个字符为结束。方法字段定义了请求类型。在 HTTP 版本 1.1 中定义了几种方法，如表 22.1 所示。

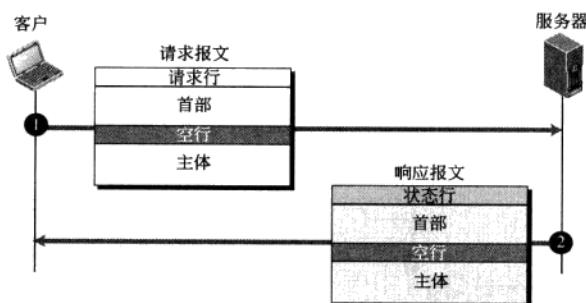


图 22.10 HTTP 事务

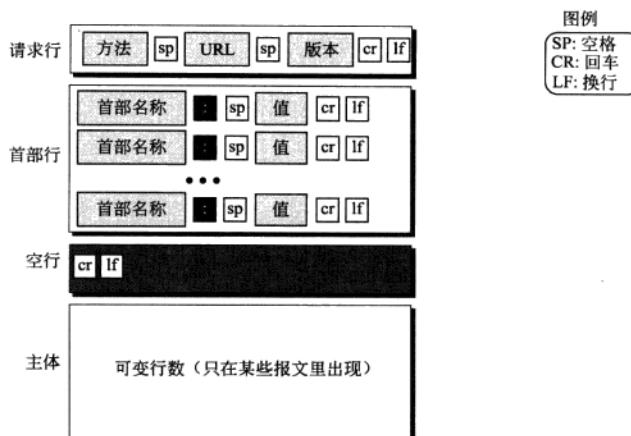


图 22.11 请求报文格式

表 22.1 方法

方法	动作
GET	请求服务器的文档
HEAD	请求关于文档的信息，但不是这个文档本身
POST	从客户向服务器发送一些信息
PUT	从服务器向客户发送文档
TRACE	把到达的请求回送
CONNECT	保留
DELETE	删除 Web 网页
OPTIONS	询问关于可用的选项

第二个字段，URL，已经在本章前面部分讨论过。它定义了相关 Web 网页的名称和地址。第三个字段，版本，给出了协议的版本。HTTP 的最新版本是 1.1。

请求报文的首部行

在请求行之后，可以有 0 到多个请求首部（request header）行。每个首部行从客户向服务器发送附加信息。例如，客户可以向服务器请求文档要以某种特殊格式发送。首部可以有一个或多个首部行。每一个首部行由一个首部名、一个冒号、一个空格和一个首部值组成（见图 22.11）。我们将在本章的后面在一些例子中给出一些首部。表 22.2 给出了请求报文中一些常用的首部名。首部值字段定义了与首部名相关联的值。首部值的完整列表可以在相应的 RFC 中找到。

表 22.2 请求首部名

首部	描述
User-agent	标志客户程序
Accept	给出客户能够接受的媒体格式
Accept-charset	给出客户能够处理的字符集
Accept-encoding	给出客户能够处理的编码方案
Accept-language	给出客户能够接受的语言
Authorization	给出客户具有何种准许
Host	给出客户的主机和端口号
Date	给出当前日期
Upgrade	指明优先使用的通信协议
Cookie	把 Cookie 回送给服务器
If-Modified-Since	只有在指明日期以后更新的文档才发送

请求报文实体

实体可以出现在请求报文中。请求报文中的实体部分通常是一些需要发送的备注信息。

响应报文

响应报文的格式如图 22.12 所示。响应报文包含一个状态行、一些首部行、一个空行。有时也包含一个实体。

状态行

响应报文中的第一行被称为状态行（Status Line）。该行共有 3 个字段，以空格符为分隔，回车换行符为结束。第一个字段定义 HTTP 协议的版本（目前是 1.1 版）。状态码字段定义了请求的状态。它由 3 个数字组成。100 系列的代码指示提供信息，200 系列的代码则指示成功的请求。300 系列的代码是把客户重新定向到另一个 URL，400 系列的代码指示在客户端的差错。最后，500 系列的代码指示在服务器端的差错。表 22.3 列出了最常用的一些状态码和状态短语。

响应报文的首部行

在状态行之后，可以有 0 到多个响应首部行。每个首部行从服务器向客户发送附加信息。例如，服务器可以发送有关文档的额外信息。

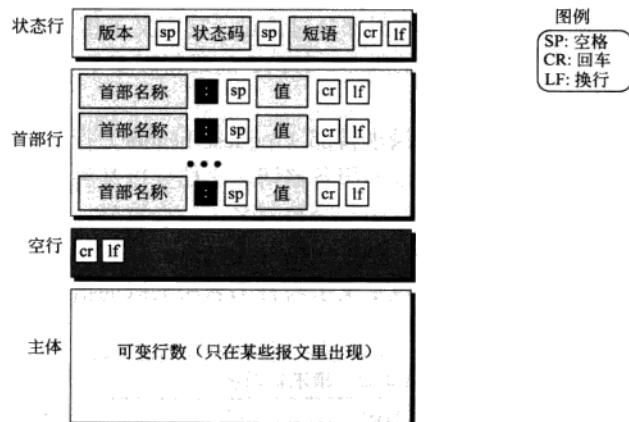


图 22.12 响应报文格式

表 22.3 状态码和状态短语

状态码	状态短语	说明
提供信息的		
100	Continue	请求的开始部分已经收到，客户可以继续它的请求
101	Switching	服务器同意切换协议
成功		
200	OK	请求成功
201	Created	新的 URL 被创建
202	Accepted	请求被接受，但还没有马上起作用
204	No content	主体中没有内容
重新定向		
301	Moved permanently	服务器已不再使用所请求的 URL
302	Moved temporarily	所请求的 URL 已暂时地删除了
304	Not modified	文档还没有被修改
客户差错		
400	Bad request	在请求中有语法差错
401	Unauthorized	请求缺少适当的授权
403	Forbidden	服务被拒绝
404	Not found	文档未找到
405	Method not allowed	这个方法不被这个 URL 所支持
406	Not acceptable	所请求的格式不可接受
服务器差错		
500	Internal server error	在服务器端有差错，如崩溃
501	Not implemented	所请求的动作不能完成
503	Service unavailable	服务暂时不可用

每个首部行可以由一个部首名、一个冒号、一个空格和一个首部值组成。我们将在本章的后面的一些例子中给出一些首部。表 22.4 列出了响应报文中一些常见的首部名。

表 22.4 响应首部名

首部	描述
Date	给出当前日期
Upgrade	指明优先使用的通信协议
Server	给出服务器的相关信息
Set-Cookie	服务器请求客户保存 Cookie
Content-Encoding	指明编码方案
Content-Language	指明语言
Content-Length	给出文档长度
Content-Type	指明媒体类型
Location	请求客户将请求发送到另一站点
Accept-Ranges	服务器将接受请求的字节范围
Last-modified	给出上次改变的日期和时间

主体 主体包含从服务器向客户发送的文档。如果响应不是错误报文，则主体将出现在响应报文中。

例 22.4

这个例子是读取文档（见图 22.13）。

我们使用 GET 方法来读取路径为 /usr/bin/image1 的图像。请求行给出方法 (GET)、URL 以及 HTTP 的版本 (1.1)。首部有两行，给出了客户可以接受 GIF 和 JPEG 格式的图像。请求报文没有主体。响应报文包含状态行和 4 行首部。这些首部行定义日期、服务器、MIME 版本和文档长度。文档的主体在首部之后。

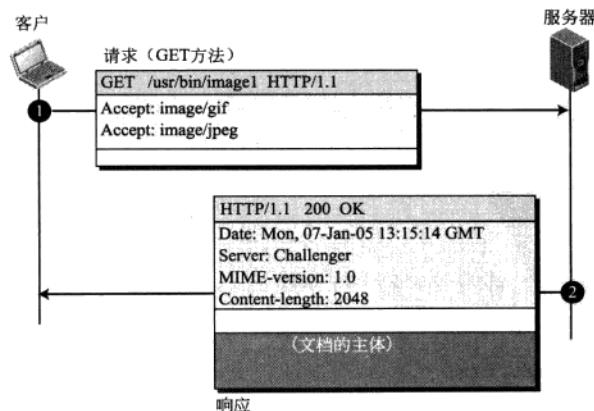


图 22.13 例 22.4

例 22.5

这个例子是客户打算向服务器发送数据。我们使用 POST 方法。请求行给出方法 (POST)、URL 以及 HTTP 的版本 (1.1)。首部共有 4 行。请求报文在主体中包含了输入信息。响应报文包含状态行和 4 行首部。被创建的文档是 CGI 文档，它附在响应报文的主体中 (见图 22.14)。

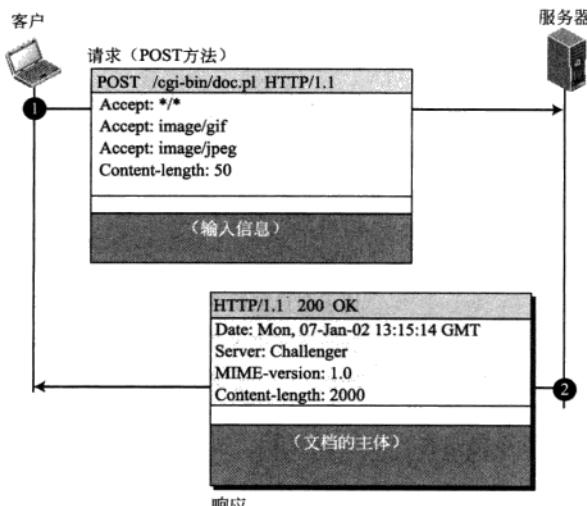


图 22.14 例 22.5

例 22.6

HTTP 使用 ASCII 字符。客户可以使用 TELNET 在端口 80 登录直接与服务器相连。前 3 行表明连接是成功的。我们接着键入 3 行。第一行是请求行 (GET 方法)，第二行是首部 (定义主机)，第三行是空行，结束请求。服务器的响应有 7 行，从状态行开始。最后的空行结束服务器的响应。在空行之后是接收到的文件，共有 14230 行 (这里未给出)。最后一行是客户的输出。

```
$ telnet www.mhhe.com 80
Trying 198.45.24.104
connected to www.mhhe.com (198.45.24.104)
Escape character is '^'
GET /engcs/compsci/forouzan HTTP/1.1
From: forouzanbehrouz@fhda.edu

HTTP/1.1 200 OK
Date: Thu, 28 Oct 2004 16:27:46 GMT
Server: Apache/1.3.9 (UNIX) ApacheJServ/1.1.2 PHP/4.1.2 PHP/3.0.18
MIME-version:1.0
Content-Type: text/html
Last-modified: Friday, 15-Oct-04 02:11:31 GMT
Content-length: 14230

Connection closed by foreign host
```

22.3.2 有条件请求

客户可以在请求中增加条件。在这种情况下，服务器将在条件满足时发送被请求的 Web 网页，否则服务器将通知客户。客户要求最常见的一种条件是 Web 网页被修改的时间和日期。客户可以通过发送 `If-Modified-Since` 首部行来通知服务器只发送在指明时间点之后修改过的页面。

例 22.7

这个例子显示了客户如何在请求中加入修改时间和日期的条件。

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

响应中的状态行表明文件在给定的时间点之后没有被修改过。因此响应报文中的主体部分为空。

```
HTTP/1.1 304 Not Modified
Date: Sat, Sept 06 08 16:22:46 GMT
Server: commonServer.com
```

(Empty Body)

22.3.3 持续连接

在版本 1.1 以前的 HTTP 指明的是非持续连接，而在版本 1.1 中，持续连接是默认的连接。

非持续连接

在非持续连接中，对每一个请求/响应都要建立一次 TCP 连接。下面列出这种策略的步骤：

1. 客户打开 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，直到它遇到文件结束标记；然后它关闭连接。

使用这种策略时，对于在不同文件中的 N 个不同图片(都位于同一服务器上)，连接必须打开和关闭 $N + 1$ 次。非持续连接策略给服务器造成了很大的开销，因为服务器需要 $N + 1$ 个不同的缓存，而每次打开连接时都要使用慢开始过程。

例 22.8

图 22.15 给出了非持续连接的一个例子。客户需要读取一个包含两个图片链接的文件。文本文件和图片位于同一服务器上。

持续连接

HTTP 版本 1.1 指明持续连接是默认的策略。在使用持续连接时，服务器在发送响应后，让连接继续打开以服务更多的请求。服务器可以在客户请求时或超时的时限到时关闭这个连接。发送端通常在每一个响应中都发送数据的长度。但是，在某些情况下，发送端并不知道数据的长度。当文档是动态文档或活动文档时就属于这些情况。在这些情况下，服务器把长度未知这件事通知客户，并在发送数据后关闭这个连接，因此客户就知道数据结束的地方已经到了。

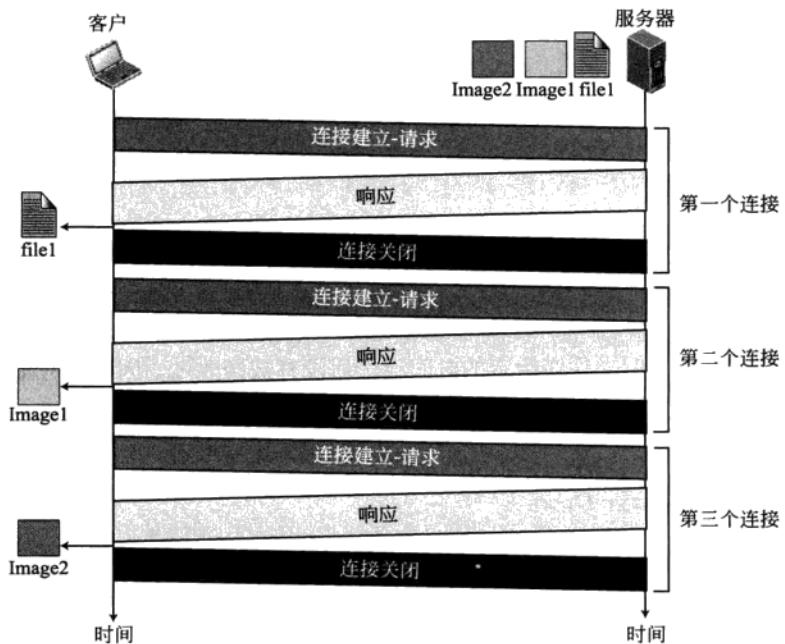


图 22.15 例 22.8

HTTP 版本 1.1 指明默认采用持续连接。

例 22.9

图 22.16 演示了与例 22.8 相同的情况，不过这次使用的是持续连接。

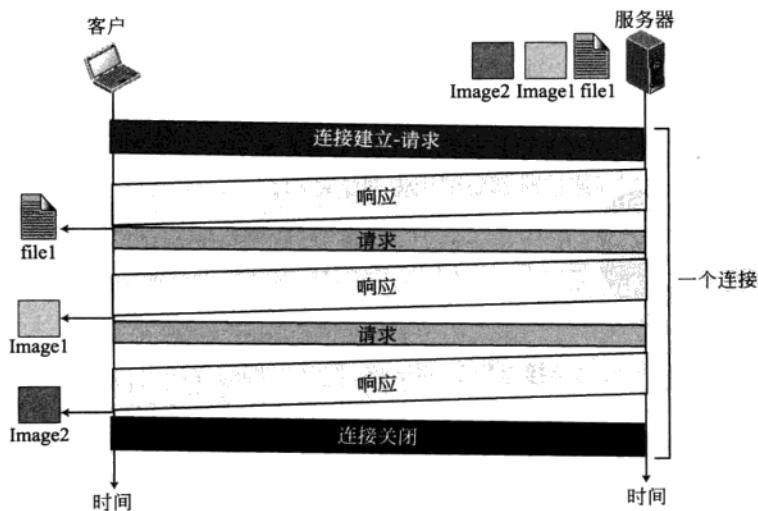


图 22.16 例 22.9

22.3.4 Cookie

万维网最初设计为无状态实体。客户发送请求，服务器回答。这样的关系就结束了。原来的万维网设计是为了公开地读取可用的文档，且正好符合了这种目的。今天的万维网有其他的功能，下面是其中的一些：

(1) 网站用做电子商店，允许用户浏览这个商店，选择所需的项目，把它们放入电子购物车，最后使用信用卡付款。

(2) 某些网站只允许已经注册的客户访问。

(3) 某些网站用做入口：用户可选择他们愿意看的 Web 网页。

(4) 某些网站仅仅是广告。

Cookie 机制就是基于以上目的而设计的。我们在第 15 章已经讨论了 Cookie 在传输层的使用。这里我们将讨论它在 Web 网页中的使用。

创建和存储 Cookie

创建和存储 Cookie 取决于具体实现，但各种实现的原理是相同的。

1. 当服务器收到来自客户的请求后，它就把有关客户的信息存储在一个文件或字符串中。这个信息可以包含客户的域名、Cookie 的内容（服务器已经收集到的客户信息，如名字、注册号等等）、时间戳，以及其他与实现有关的信息。

2. 服务器向客户发送的响应包含这个 Cookie。

3. 当客户收到响应时，浏览器把这个 Cookie 存储在 Cookie 目录中，它是按域名服务器的名字来分类的。

Cookie 的使用

当一个客户向服务器发送请求时，浏览器就查找在 Cookie 目录中是否有那个服务器发送的 Cookie。如果找到了，就把这个 Cookie 包含在请求中。当服务器收到这个请求时，它就知道这是一个老客户，而不是一个新客户。请注意，这个 Cookie 的内容从来没有被浏览器读过，也没有暴露给用户。Cookie 是服务器制作的，也是被服务器吃掉的。现在我们看一下，Cookie 是怎样在前面讲过的 4 种情况下使用的：

(1) 电子商店（电子商务）可以让客户购货者（client shopper）使用 Cookie。当客户选择了一个项目并把它放入购物车时，包含该项目信息（如数量和单价）的 Cookie 就发送到浏览器。如果客户又选择了第二个项目，Cookie 就会被新选择的信息更新，诸如此类。当客户结束购物并要结账时，最后的 Cookie 就被读取，总的费用就可以计算出来。

(2) 限制已注册的客户才能访问的网站，在客户首次注册时，要向客户发送一个 Cookie。对以后的任何重复的访问，只有那些发送适当的 Cookie 的客户才允许访问。

(3) 万维网的入口也以类似方式使用 Cookie。当客户选择了所喜爱的页面时，一个 Cookie 就被创建和发送。如果这个网站再次被访问，这个 Cookie 就被发送到这个服务器，说明这个客户想寻找什么。

(4) 广告公司也可以使用 Cookie。广告公司可以在经常有用户访问的主网站上放置横幅广告。广告公司可以仅仅提供链接到这个横幅广告的地址 URL，而不需要给出这个横幅广告本身。当用户访问这个主网站时，点击一个广告公司的图标，就把请求发送到这个广

告公司。这个广告公司就发送一个横幅广告（例如是一个 GIF 文件），但它还包含具有用户 ID 的 Cookie。以后对这个横幅广告的任何使用都会在数据库中增加这个用户浏览万维网的行为概要。广告公司搜集了用户的兴趣，并且可以把这些信息卖给其他公司。Cookie 的使用引起许多争议。值得庆幸的是，有些新的规定将会出台以便保护用户的秘密。

例 22.10

图 22.17 给出了一个电子商店如何从使用 Cookie 中受益的场景。假设一个购物者想要从一家叫 BestToys 的电子商店购买一个玩具。这名购物者的浏览器（客户）首先向 BestToys 的服务器发送一个请求。服务器则会为这位客户创建一个空的购物车（一个列表）并赋予其一个 ID（例如，12343）。接着服务器会发送一个响应报文，其中包含所有在售玩具的图片。每张图片下面会有一个链接，点击这个链接就会选中购买这个玩具。这个响应报文同时也包含 Set-Cookie 首部行，它的首部值为 12343。客户端会显示这些图片并将这个 Cookie 的值保存在名为 BestToys 的文件中。购物者看不见这个 Cookie。当这名购物者通过点击选中了一个玩具，客户端会发送一个请求，并把 ID 12343 包括在 Cookie 首部行中。即使服务器很忙并忘记了这位购物者，当接收来自客户的请求时会在检查首部时发现一个值为 12343 的 Cookie。这样服务器会知道这位顾客不是新来的，于是它搜索 ID 是 12343 的购物车。购物车（列表）会被打开并被加入选中的玩具。接下来服务器会发送另一个响应给购物者

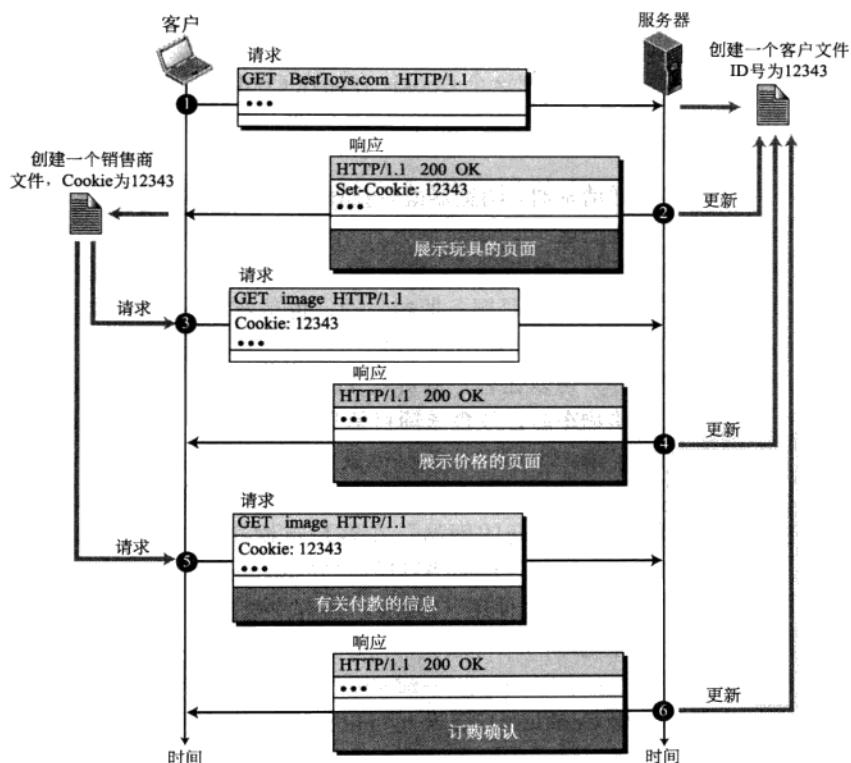


图 22.17 例 22.10

通知她合计价格并要求这名顾客付款。购物者会提供有关她信用卡的信息同时发送一个新的包含 Cookie 值为 ID 12343 的请求。当这个请求抵达服务器端，服务器再次看见 ID 12343。这时服务器会接受这个订单及付款，同时在响应中发送确认信息。其他有关客户的信息，例如信用卡号码，姓名及地址也会被储存在服务器中。如果这位购物者将来某个时候再次来到这个电子商店，客户端会再次发送 Cookie，而商店通过读取已存储的文件就可以知道有关这个顾客的所有信息。

22.3.5 Web 缓存：代理服务器

HTTP 支持代理服务器（proxy server）。代理服务器是个计算机，它保留对最近请求的响应的副本。在有代理服务器的情况下，HTTP 客户把请求发送给代理服务器。代理服务器检查它的高速缓存。如果在高速缓存中没有这个响应，代理服务器就把请求发送给相应的服务器。到达的响应被发送给代理服务器并存储起来，以便在以后为其他客户的请求使用。

代理服务器减少了原始服务器的负荷，减小了通信量，也减小了延时。但是，要使用代理服务器，客户必须配置成访问代理服务器而不是目标服务器。

请注意代理服务器同时扮演着客户和服务器的角色。当它从客户接收到一个请求，并发现它已经有对应的响应，这时它以服务器的角色发送响应给客户。当它从客户接收到一个请求，并发现它没有对应的响应，它先以客户的角色发送请求到目标服务器。当收到响应之后再以服务器的角色发送响应给客户。

代理服务器的位置

代理服务器通常在客户端。这说明我们可以有如下所示的代理服务器分级结构：

1. 客户计算机也可以被用做小容量的代理服务器，用来存储客户经常请求的响应。
2. 在公司中，可以在计算机局域网中安装代理服务器以减轻进出局域网的负荷。
3. 有众多顾客的因特网服务供应者（ISP）可以安装代理服务器以减轻进出其网络的负荷。

缓存更新

一个很重要的问题是响应在代理服务器中应该被存放多长时间才该将其删除或替换。针对这一问题有多种策略可选择。一种方案是保存一个不经常更新信息的网站列表。例如，新闻通讯社可能只在每天早上更新它的新闻页面。这就是说代理服务器可以在每天早上很早的时候取得这些新闻并将其保存至第二天。另一种推荐方案是增加一些首部来显示信息上次更新的时间。代理服务器就可以利用首部中的这些信息来估算这些信息多长时间之内会有效。有关 Web 缓存还有很多的推荐方式，不过在这里我们把它们留给其他具体讨论这个话题的书籍。

22.3.6 HTTP 的安全

HTTP 本身并不提供安全。然而，如我们将在第 30 章所示，HTTP 可以运行在安全套接层(SSL)之上。这种情况之下的 HTTP 被称为 HTTPS。HTTPS 可以提供保密性，客户和

服务器鉴别以及数据完整性。

22.4 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

22.4.1 参考书

有几本书深入浅出地覆盖了有关 HTTP 的内容，包括[Com 06]、[Mir 07]、[Ste 94]和[Tan 03]。

22.4.2 RFC

有几份 RFC 给出了有关 HTTP 的更新内容，包括 RFC 2068 和 RFC2109。

22.5 重要术语

活动文档	JavaScript
Active Server Pages (ASP)	Java Server Pages (JSP)
小应用程序	非持续连接
浏览器	路径
ColdFusion	持续连接
公共网关接口（CGI）	代理服务器
动态文档	请求首部
可扩展置标语言（XML）	请求行
可扩展样式语言（XSL）	请求类型
主机	响应首部
超媒体	静态文档
超文本	状态码
超文本置标语言（HTML）	状态行
超文本预处理器（PHP）	统一资源定位符（URL）
超文本传送协议（HTTP）	Web 网页
Java	万维网（WWW）
Java 小程序	

22.6 本章小结

- 万维网（WWW）是遍及全世界且相互链接起来的信息储藏所。超文本和超媒体文档通过指针相互链接。
- 万维网的体系结构是由客户和服务器组成的。客户或浏览器解释和显示 Web 文档。浏览器包括控制程序、客户程序和解释程序。服务器储存 Web 网页。
- Web 文档可以分为静态的、动态的或活动的。静态文档是内容固定且存储在服务器上的文档。动态 Web 文档仅在浏览器请求时才在服务器创建。活动文档是被客户读取并在客户端运行的程序副本。
- 超文本传送协议（HTTP）是访问在万维网（WWW）上的数据所使用的主要协议。HTTP 使用 TCP 连接来传送文件。HTTP 事务由请求和响应报文组成。
- HTTP 有两种模式：持续连接和非持续连接。非持续连接对每一个事务都要建立一个新的 TCP 连接。持续连接只使用一个连接。持续连接是新版本 HTTP 的默认连接方式。
- HTTP 可使用 Cookie 来保存事务的状态。服务器发送 Cookie，Cookie 储存在客户端以供服务器以后读取。
- Web 缓存使用代理服务器来提高 HTTP 的效率。代理服务器安装在客户站点。
- HTTPS 是 HTTP 的安全版本。它使用安全套接字（SSL）协议来提供保密性，客户和服务器鉴别以及数据完整性。

22.7 实践安排

22.7.1 习题

1. 假设有一个服务器的域名是 `www.common.com`。
 - a. 试给出读取文档 `/usr/users/doc/doc1` 的请求。请使用至少两个通用首部、两个请求首部和一个实体首部。
 - b. 对应问题 a，试给出一个成功请求的响应。
 - c. 对应问题 a，试给出一个文件已永久移动到 `/usr/deads/doc1` 的响应。
 - d. 对应问题 a，试给出请求中有语法错误的响应。
 - e. 对应问题 a，试给出客户没有被授权访问文档的响应。
2. 假设有一个服务器的域名是 `www.uncommon.com`。
 - a. 试给出询问在`/bin/users/file` 下的文档的信息的请求。使用至少两个通用首部和一个请求首部。
 - b. 试给出对问题 a 成功请求的响应。
3. 假设有一个服务器的域名是 `www.public.edu`。
 - a. 试给出把位于`/bin/usr/bin/file1` 的文件复制到`/bin/file1` 的请求。

b. 试给出对问题 a 的响应。

4. 假设有一个服务器的域名是 www.bigBusiness.com。

a. 试给出删除位于 /bin/file1 的文件的请求。

b. 试给出对问题 a 的响应。

5. 假设有一个服务器的域名是 www.EveryOne.com。

a. 试给出在 /bin/letter 存储文件的请求。客户必须标明它能够接受的文档类型。

b. 试给出对问题 a 的响应。响应必须给出文档的生成时间，以及当内容可能改变时的日期和时间。

6. 在图 22.5 中，试找出客户与服务器间交换的 TCP 报文段的实际数量并参照该图画出所有报文段。

7. 在图 22.6 中，试找出客户与服务器间交换的 TCP 报文段的实际数量并参照该图画出所有报文段。

8. 参照图 22.17，对于服务器仅允许注册用户访问的情况，试画图表明 Cookie 在该场合中的应用。

9. 参照图 22.17，试画图表明 Cookie 在服务器作为网站入口情况下的应用。

10. 参照图 22.17，对于服务器在网络广告中使用 Cookie 的情况，试画图表明 Cookie 在该场合中的应用。

11. 试画图表明将代理服务器作为客户计算机的一部分使用时的情况：

a. 当响应已储存在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

b. 当响应不在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

12. 试画图表明代理服务器安装在客户所在局域网中的情况：

a. 当响应已储存在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

b. 当响应不在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

13. 试画图表明代理服务器安装在因特网供应商 (ISP) 网络中的情况：

a. 当响应已储存在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

b. 当响应不在代理服务器中时，试指出客户与代理服务器及目标服务器之间的事务。

22.7.2 研究活动

14. 试找出为什么 IP 地址不能代替 Cookie?

15. 试找出有关网络镜像 (Web mirroring) 的信息，它的缓存是在服务器端完成而非客户端。

第 23 章 电子邮件：SMTP、POP、 IMAP 和 MIME

电子邮件（E-mail）是最流行的互联网服务之一。互联网的设计者可能从来不曾想象到这个应用程序如此普及。它的体系结构包括几个构件，我们将在本章中讨论。

在互联网时代的初始阶段，用电子邮件发送的报文都很短，并且只包含文本；这些电子邮件让人们快速交换备忘录信息。今天的电子邮件要复杂得多。它允许报文包含文本、声音和视频。它还允许把一个报文发送给一个或多个收件人。

在本章中，我们首先讨论电子邮件的一般体系结构，它包含三个主要构件：用户代理、报文传送代理和报文读取代理。我们下面就描述实现这些构件的协议。

目标

本章有如下几个目标：

- 通过四种情况解释电子邮件的体系结构。
- 解释用户代理（UA），它提供的服务，以及用户代理的两种类型。
- 解释电子邮件的收发机制。
- 介绍报文传送代理（MTA）的角色以及简单邮件传送协议（SMTP），它是处理 MTA 的正式协议。
- 解释电子邮件传送的几个阶段。
- 讨论两种报文读取代理（MAA）：POP 和 IMAP。
- 讨论作为一组软件功能的 MIME，它负责 ASCII 数据和非 ASCII 数据的相互转换。
- 讨论基于万维网的电子邮件的概念。
- 解释电子邮件系统的安全性。

23.1 体系结构

为了说明电子邮件的体系结构，我们给出四种情况。我们从最简单的情况开始，然后逐渐增加复杂性。第四种情况在交换邮件时最为常见。

23.1.1 第一种情况

第一种情况是电子邮件的发送方和接收方都是在同一个邮件服务器上的用户（或应用程序）；它们直接连接在一个共享的邮件服务器上。管理员为每一个用户创建一个邮箱，用来存储收到的报文。一个邮箱是本地硬盘的一部分，是具有准许限制的特殊文件。只有邮箱的拥有者才能够对它进行存取。当 Alice 需要向 Bob 发送报文时，她就运行用户代理（UA）程序来准备这个报文，然后把它存储在 Bob 的邮箱中。这个报文有发送方和接收方的邮箱地址（文件名）。Bob 可以在他方便时通过用户代理读取其邮箱中的内容。图 23.1 表示了这种概念。

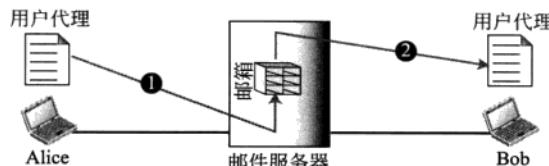


图 23.1 第一种情况

这种情况和在一个办公室中的雇员之间交换传统的便条是相似的。有一个邮件室，里面有每一个雇员的邮箱，上面有雇员的名字。当 Alice 需要给 Bob 发送便条时，她就写了便条，插入到 Bob 的邮箱中。Bob 检查他的邮箱，发现了 Alice 的便条，就阅读这个便条。

当电子邮件的发送方和接收方处在同一个邮件服务器上时，我们只需要两个用户代理。

23.1.2 第二种情况

第二种情况是电子邮件的发送方和接收方是在两个不同的邮件服务器上的用户（或应用程序）。报文需要通过互联网来发送。这里我们需要用户代理（UA）和报文传送代理（MTA），如图 23.2 所示。

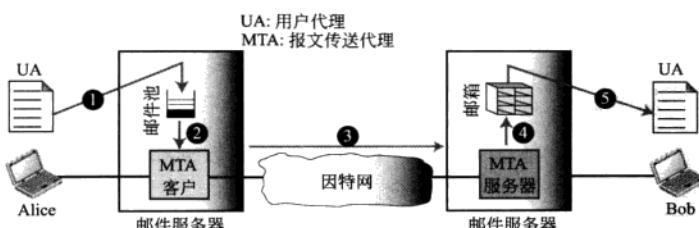


图 23.2 第二种情况

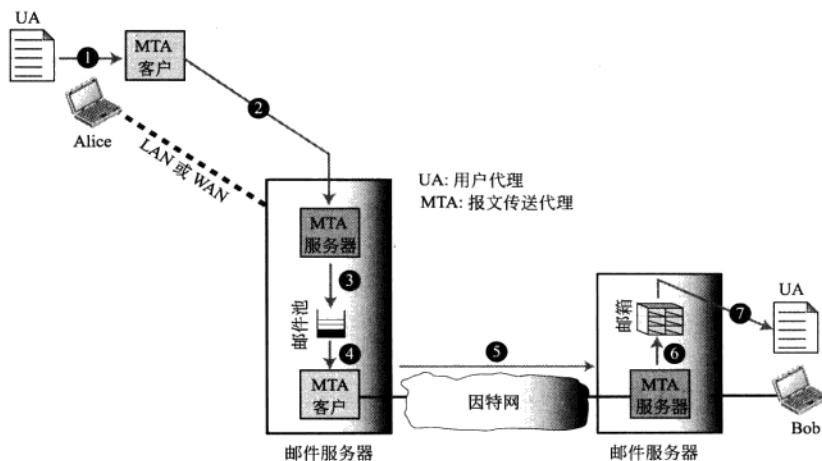
Alice 需要通过用户代理程序把她的报文发送给在她这边的邮件服务器。在她这边的邮件服务器用一个队列（邮件池）存储等待发送的邮件。Bob 也需要一个用户代理程序来读取存储在他这边的系统邮箱中的报文。但是这个报文需要从 Alice 这边通过因特网发送到 Bob 这边。因此这里需要两个报文传送代理：一个客户和一个服务器。像大多数互联网上的客户-服务器程序一样，服务器需要一直不停地运行，因为它不知道在什么时候会有一个

客户请求连接。另一方面，当队列中有报文要发送时，系统就会触发客户。

当电子邮件的发送方和接收方在两个不同系统上时，我们就需要两个 UA 和一对 MTA（客户和服务器）。

23.1.3 第三种情况

图 23.3 显示了第三种情况。在这种情况下，Bob 直接连接到他的邮件服务器（这点和第二种情况一样）。但是 Alice 则和她的系统是分开的。Alice 或者是通过一个点对点广域网（如拨号上网的调制解调器、DSL 或电缆调制解调器）¹连到邮件服务器，或者是连接到某组织的局域网，该局域网使用一个邮件服务器来处理电子邮件；所有的用户需要把他们的邮件发送到这个邮件服务器。



Alice 仍然需要用户代理来准备她的报文。她需要通过局域网或广域网发送报文。这就要通过一对报文传送代理（客户和服务器）。只要 Alice 有报文要发送，她就调用她的用户代理，接着用户代理再调用 MTA 客户。MTA 客户与系统的 MTA 服务器（它一直都在工作）建立连接。在 Alice 这边的系统把所有收到的报文进行排队。然后再使用 MTA 客户把这个报文发送到 Bob 那边的系统，该系统收到报文后就存储在 Bob 的邮箱中。Bob 在他方便时通过他的用户代理读取这个报文。请注意，我们需要两对 MTA 客户和服务器程序。

若发送方是通过局域网或广域网连接到邮件服务器上，那么就需要两个 UA 和两对 MTA（客户和服务器）。

23.1.4 第四种情况

第四种情况是最常见的。Bob 也是通过广域网或局域网连接到他的邮件服务器。在报文到达 Bob 的邮件服务器后，Bob 需要读取它。现在我们需要另一组客户-服务器代理，即

报文读取代理 (MAA)。Bob 使用 MAA 客户读取他的报文。MAA 客户向 MAA 服务器 (它一直在运行) 发送请求, 即请求传送这个报文。图 23.4 给出了这种情况。

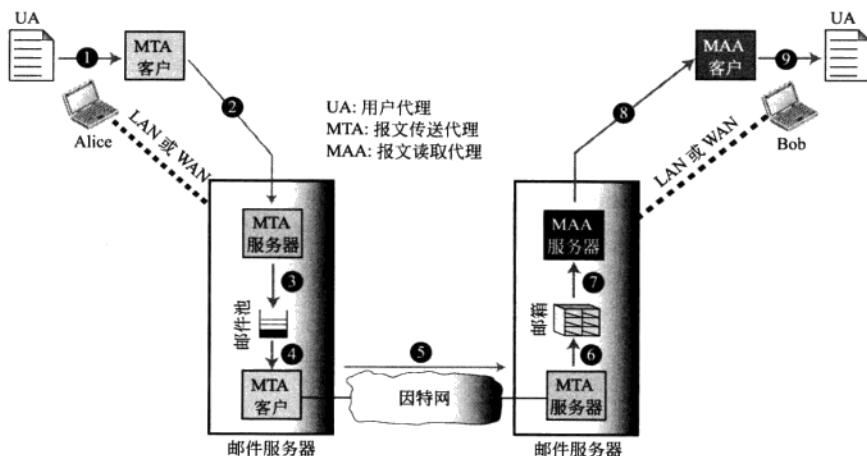


图 23.4 第四种情况

这里需要强调两个要点。第一, Bob 不能跳过邮件服务器直接与 MTA 服务器相连。如果要直接使用 MTA 服务器, Bob 就必须一直运行 MTA 服务器程序, 因为他不知道在什么时候会有报文到达。这就表示, 如果是通过局域网连接到他的系统, Bob 必须一直让他的计算机在运行。如果他是通过广域网连接, 那么他就必须一直保持这个连接。这两种情况当前都是不可行的。

第二, 请注意, Bob 需要另一对客户-服务器程序: 报文读取程序。这是因为 MTA 客户-服务器程序是一个推送(push)程序: 客户把报文推送给服务器。但现在 Bob 需要拉取(pull)程序。客户需要把报文从服务器拉取过来。图 23.5 说明了这种区别。



图 23.5 推送与拉取的区别

当发送方和接收方都是通过局域网或广域网连接到邮件服务器时, 我们就需要两个 UA、两对 MTA (客户和服务器) 和一对 MAA (客户和服务器)。这是现今最常见的情况。

23.2 用户代理

电子邮件系统的第一个构件就是用户代理 (UA)。它向用户提供服务, 使发送和接收报文的过程更加容易。

23.2.1 用户代理提供的服务

用户代理是一个软件包（程序），它撰写、阅读、回复和转发报文。它同时还处理用户计算机上的本地邮箱。

23.2.2 用户代理类型

共有两种类型的用户代理：命令驱动和基于 GUI 的。命令驱动用户代理属于早期的电子邮件。它们仍然作为基本的用户代理，存在于服务器中。命令驱动的用户代理通常从键盘接受一字符命令来完成它的任务。例如，用户可以在命令行提示符处键入字符 r 来回复报文的发件人，或键入 R 来回复发件人和所有的收件人。

命令驱动用户代理的一些例子包括 mail、pine 和 elm。

现代的用户代理都是基于 GUI 的。它们包含图形用户界面（GUI）构件，允许用户使用键盘和鼠标与软件进行交互。它们有图形构件，如图标、选项栏和视窗等，使服务易于获取。

基于 GUI 的用户代理的一些例子包括 Eudora、Outlook 以及 Netscape 等。

23.2.3 发送邮件

要发送邮件，用户通过 UA 创建邮件，它看起来很像邮政邮件。邮件有信封和报文（见图 23.6）。

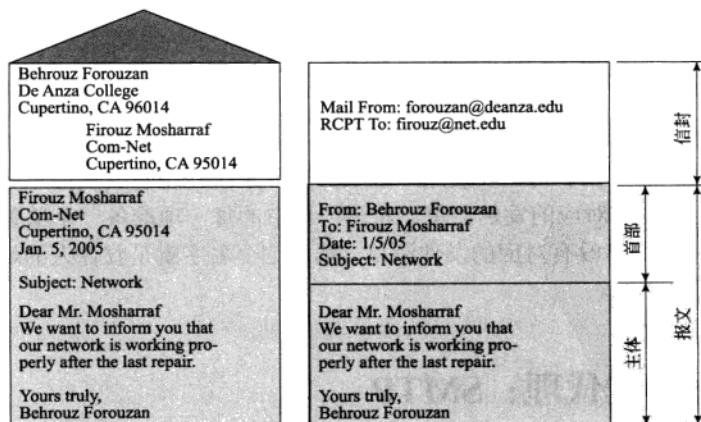


图 23.6 电子邮件的格式

信封

信封通常包含发件人地址、收件人地址以及其他信息。

报文

报文包含首部和主体。报文的首部定义发件人、收件人、报文的主题，以及一些其他信息。报文的主体包含收件人将要读取的真正信息。

23.2.4 接收邮件

用户代理由用户（或计时器）触发。若用户有邮件，UA 就用通知消息告诉用户。若用户准备读取邮件，则可显示清单，其中每一行包含在邮箱中特定报文的信息概要。这个概要通常包括发件人的邮件地址、主题，以及发送或接收这个邮件的时间。用户可以选择这些报文中的任何一个，并把它的内容显示在屏幕上。

23.2.5 地址

为了交付邮件，邮件处理系统必须使用唯一的编址系统。互联网的邮件地址包括两个部分：本地部分和域名，并且用符号@分隔开（见图 23.7）。



本地部分

本地部分定义了一个特殊文件的名字，称为用户邮箱。在用户邮箱中存储了所有收到的用户邮件，以便用户代理进行读取。

域名

地址的第二部分是域名。一个组织通常选择一个或多个主机来接收和发送电子邮件，这些主机通常称为邮件服务器或邮件交换器 (mail exchanger)。指派给每一个邮件交换器的域名或者来自 DNS 数据库，或者是一个逻辑名字（例如该组织的名字）。

23.2.6 发件清单或分组清单

电子邮件允许用一个名称——别名 (alias) ——来表示几个不同的电子邮件地址；这称为发件清单(mailing list)。每当要发送一个报文时，系统就在别名数据库中检查收件人的名字；如果这个别名有对应的发件清单，就为清单中的每一项准备一个单独的报文，交给 MTA 发送。如果该别名没有对应的发件清单，那么这个名字就是收件人的地址，就把该报文交付给邮件传送实体。

23.3 报文传送代理：SMTP

真正的邮件传送是通过报文传送代理 (MTA)。要发送邮件，一个系统必须有客户 MTA，而要接收邮件，一个系统就必须有 MTA 服务器。定义互联网中的 MTA 客户和 MTA 服务器的正式协议称为简单邮件传送协议 (SMTP)。如我们以前讲过的，在大多数的情况下（第四种情况），我们需要两对 MTA 客户-服务器。图 23.8 给出了这种情况下

SMTP 协议的范围。

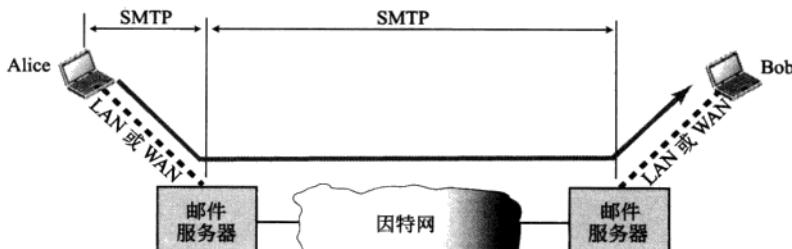


图 23.8 SMTP 的范围

SMTP 使用了两次，即在发送方和发送方的邮件服务器之间，以及在两个邮件服务器之间使用。我们很快将要看到，在邮件服务器和接收方之间还需要另一个协议。

SMTP 只是定义了必须来回发送的一些命令和响应。每一个网络可以自由地选择一个软件包来实现它。我们将在本节的剩下部分讨论 SMTP 的邮件传送机制。

23.3.1 命令和响应

SMTP 使用一些命令和响应在 MTA 客户和 MTA 服务器之间传送报文（见图 23.9）。

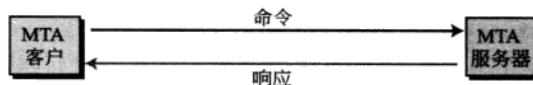


图 23.9 命令和响应

每一个命令或响应都以二字符（回车和换行）的行结束标记终止。

命令

命令是从客户发送到服务器。命令的格式如下所示。

关键字：变量

命令包括关键词，后面跟着零个或多个变量。SMTP 定义了 14 个命令。这些命令在表 23.1 中列出，更详细的描述见下文。

表 23.1 命令

关键词	变量
HELO	发送方的主机名
MAIL FROM	发件人
RCPT TO	预期的收件人
DATA	邮件的主体
QUIT	
RSET	

续表

关键词	变量
VRFY	收件人名字
NOOP	
TURN	
EXPN	邮件发送清单
HELP	命令名
SEND FROM	预期的收件人
SMOL FROM	预期的收件人
SMAL FROM	预期的收件人

- **HELO** 这个命令由客户用来标志自己而使用的。其变量是客户主机的域名。格式是：

```
HELO: challenger.atc.fhda.edu
```

- **MAIL FROM** 这个命令由客户用来标志发件人。它的变量是发件人的电子邮件地址（本地部分加上域名）。格式是：

```
MAIL FROM: forouzan@challenger.atc.fhda.edu
```

- **RCPT TO** 这个命令由客户用来标志预期的收件人。它的变量是收件人的电子邮件地址。若有多个收件人，则命令要重复使用。格式是：

```
RCPT TO: betsy@mcgraw-hill.com
```

- **DATA** 这个命令用来发送真正的报文。在 DATA 命令后面所有的行都被当作是邮件报文。报文的终止是只包含一个点的行。格式是：

```
DATA
```

```
This is the message
to be sent to the McGraw-Hill
Company
```

- **QUIT** 这个命令结束报文。格式是：

```
QUIT
```

- **RSET** 这个命令使当前的邮件事务异常终止。所存储的关于发件人和收件人的信息都被删除。连接将被复位。格式是：

```
RSET
```

- **VRFY** 这个命令用来验证收件人的地址，它作为变量发送出去。发送方可以请接收方证实名字是否标志有效的收件人。它的格式是：

```
VRFY: betsy@mcgraw-hill.com
```

- **NOOP** 这个命令由客户使用，用来检查收件人的状态。它需要收件人的回答。它的格式如下：

```
NOOP
```

- **TURN** 这个命令让发件人和收件人交换位置，即发件人变成收件人，收件人变成

发件人。但是，今天大多数 SMTP 的实现并不支持这个功能。格式是：

TURN

- EXPN** 这个命令要求接收邮件的主机把作为变量的发送清单进行扩展，并返回组成清单的收件人的邮箱地址。格式是：

EXPN: x y z

- HELP** 这个命令要求收件人发送作为变量的命令的有关信息。格式是：

HELP: mail

- SEND FROM** 这个命令指明这个邮件是要交付到收件人的终端，而不是邮箱。若收件人没有登录，邮件就被返回。变量就是发件人地址。格式是：

SEND FROM: forouzan@fhda.atc.edu

- SMOL FROM** 这个命令指明邮件是要交付到收件人的终端或邮箱。这就表示若收件人已登录了，邮件就只交付到终端。若收件人未登录，邮件就交付到邮箱。变量是发件人地址。格式是：

SMOL FROM: forouzan@fhda.atc.edu

- SMAL FROM** 这个命令指明邮件是要交付到收件人的终端和邮箱。这就表示若收件人已登录，邮件就交付给终端和邮箱。若收件人未登录，邮件就只交付给邮箱。变量是发件人地址。格式是：

SMAL FROM: forouzan@fhda.atc.edu

响应

响应是从服务器发送到客户。响应是三位十进制数字，后面可以跟着附加的文本信息。这个概念和我们在第 22 章中所讨论的 HTTP 响应的情况很相似。表 23.2 列出了一些常用响应。

表 23.2 响应

代码	说明
正面完成回答	
211	系统状态或求助回答
214	求助报文
220	服务就绪
221	服务关闭传输信道
250	请求命令完成
251	用户不是本地的；报文将被转发
正面中间回答	
354	开始邮件输入
过渡负面完成回答	
421	服务不可用
450	邮箱不可用
451	命令异常终止：本地差错
452	命令异常终止：存储器不足

续表

代码	说明
永久负面完成回答	
500	语法差错；不能识别的命令
501	语法的参数或变量差错
502	命令未实现
503	命令序列不正确
504	命令暂时未实现
550	命令未执行；邮箱不可用
551	用户非本地的
552	所请求的动作异常终止；存储位置超过
553	所请求的动作未发生；邮箱名不允许用
554	事务失败

23.3.2 邮件传送阶段

邮件报文的传送共有 3 个阶段：连接建立、报文传送和连接终止。

连接建立

当客户与熟知端口 25 建立了 TCP 连接后，SMTP 服务器就开始它的连接阶段。这个阶段包括以下 3 个步骤，如图 23.10 所示。

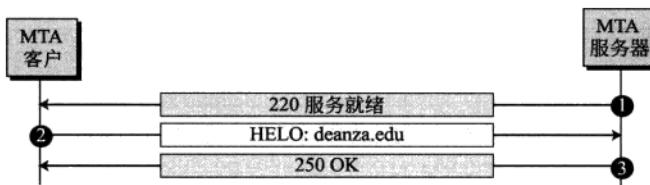


图 23.10 连接建立

1. 服务器发送代码 220（服务就绪）告诉客户它已准备好接收邮件。若服务器未就绪，它就发送代码 421（服务不可用）。

2. 客户发送 HELO 报文，并使用它的域名地址标志自己。这个步骤是必要的，用来把客户的域名通知服务器。请记住，在 TCP 的连接建立阶段，发送方和接收方是通过它们的 IP 地址来知道对方的。

3. 服务器响应代码 250（请求命令完成）或根据不同情况给出其他一些代码。

报文传送

在 SMTP 客户与服务器之间建立连接后，发件人就可以与一个或多个收件人交换单个的报文了。这个阶段包括 8 个步骤。若收件人超过一个，则步骤 3 和 4 将重复进行（见图 23.11）。

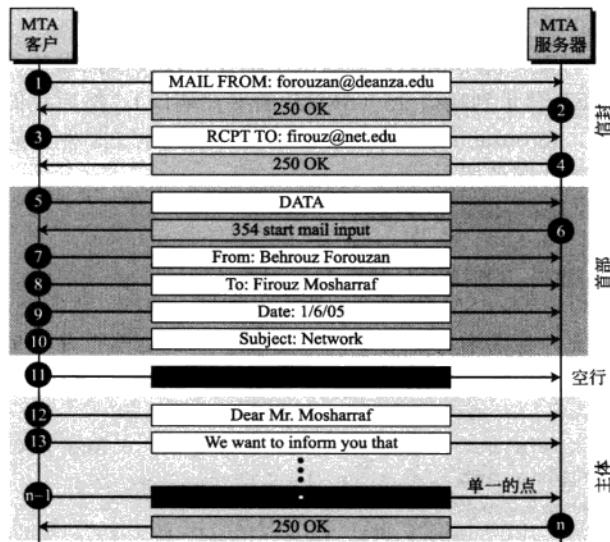


图 23.11 报文的传送

1. 客户发送 MAIL FROM 报文介绍报文的发送者。它包括发件人的邮件地址（邮箱和域名）。这个步骤是必要的，它可以把返回邮件地址交给服务器，以便返回差错或报告报文时使用。
2. 服务器响应代码 250 或其他适当的代码。
3. 客户发送 RCPT TO（收件人）报文，包括收件人的邮件地址。
4. 服务器响应代码 250 或其他适当的代码。
5. 客户发送 DATA 报文对报文的传送进行初始化。
6. 服务器响应代码 354（开始邮件输入）或其他适当的报文。
7. 客户用连续的行发送报文的内容。每一行以二字符的行结束标记（回车和换行）终止。这个报文以仅有一个点的行结束。
8. 服务器响应代码 250（OK）或其他适当的代码。

连接终止

在报文传送成功后，客户就终止连接。这个阶段包括两个步骤（见图 23.12）。

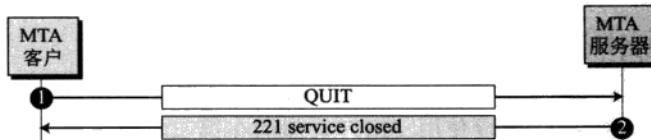


图 23.12 连接终止

1. 客户发送 QUIT 命令。
 2. 服务器响应代码 221 或其他适当的代码。
- 在连接终止阶段后，TCP 连接必须关闭。

例 23.1

让我们看一下如何直接使用 SMTP 发送电子邮件，以及模拟本节所讨论过的命令和响应。我们使用 TELNET 登录到端口 25（SMTP 的熟知端口）。然后我们直接使用命令发送电子邮件。在本例中，`forouzanb@adelphia.net` 给自己发送电子邮件。前几行表示 TELNET 尝试连接到 `adelphia` 邮件服务器。

```
$ telnet mail.adelphia.net 25
Trying 68.168.78.100...
connected to mail.adelphia.net (68.168.78.100).
```

在连接后，我们可以输入 SMTP 命令，然后收到如下响应。我们把命令用黑体字表示。请注意，为了清楚起见，我们增加了用“=”表示的注释。这些行并不是电子邮件过程的一部分。

```
===== 连接建立 =====
220 mta13.adelphia.net SMTP server ready Fri, 6 Aug 2004
HELO mail.adelphia.net
250 mta13.adelphia.net
===== 信封 =====
MAIL FROM: forouzanb@adelphia.net
250 Sender <forouzanb@adelphia.net> Ok
RCPT TO: forouzanb@adelphia.net
250 Recipient <forouzanb@adelphia.net> Ok
===== 首部和主体 =====
DATA
354 Ok Send data ending with <CRLF>.<CRLF>
From: Forouzan
To: Forouzan

This is a test message
to show SMTP in action.
.
===== 连接关闭 =====
250 Message received: adelphia.net@mail.adelphia.net
QUIT
221 mta13.adelphia.net SMTP server closing connection
Connection closed by foreign host.
```

23.4 报文读取代理：POP 和 IMAP

邮件交付的第一和第二阶段使用 SMTP。然而在第三阶段中并不使用 SMTP，因为 SMTP 是一个推送协议；它把报文从客户推送至服务器。换言之，数据块（报文）的方向是从客户到服务器。但另一方面，在第三阶段需要一个拉取协议；客户必须把报文从服务器拉取到客户。数据块的方向是从服务器到客户。第三阶段使用报文读取代理。

目前共有两种报文读取协议可以使用：邮局协议版本 3（POP3）和互联网邮件读取协议版本 4（IMAP4）。图 23.13 给出了最常见的情况（第四种情况）下这两种协议的位置。

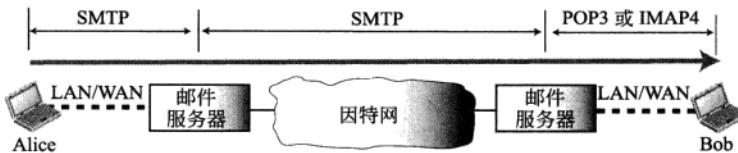


图 23.13 POP3 和 IMAP4

23.4.1 POP3

邮局协议版本 3 (POP3) 很简单，但它的功能有限。客户 POP3 软件安装在收件人的计算机上；服务器 POP3 软件安装在邮件服务器上。

当用户需要从邮件服务器的邮箱中下载电子邮件时，客户就开始读取邮件。客户（用户代理）在 TCP 端口 110 打开到服务器的连接。然后它发送用户名和口令，访问邮箱。用户可以列出清单，并逐个读取邮件报文。图 23.14 给出了使用 POP3 进行下载的例子。

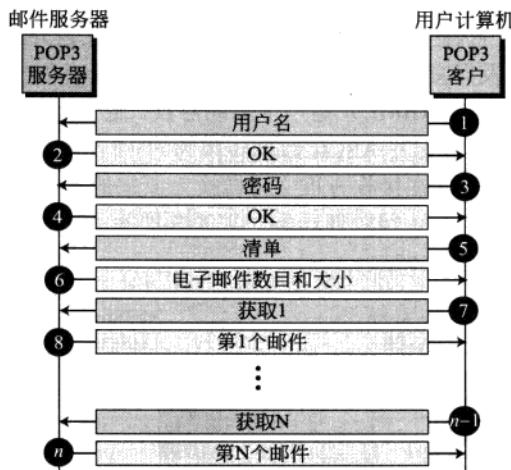


图 23.14 POP3

POP3 有两种方式：删除方式和保存方式。删除方式就在每一次读取邮件后就把邮箱中的这个邮件删除。保存方式就是在读取邮件后仍然在邮箱中保存这个邮件。删除方式通常用在用户使用固定计算机工作的情况下，而用户在读取或回答邮件后可以保存或整理所收到的邮件。保存方式通常用在用户离开她的主要计算机（例如使用膝上型计算机时）来读取她的邮件。邮件读取后还保存在系统中，供日后读取和整理。

23.4.2 IMAP4

另一种邮件读取协议是互联网邮件读取协议版本 4 (IMAP4)。IMAP4 和 POP3 相似，但具有更多的特点，IMAP4 的功能更强，同时也更复杂。

POP3 在以下几个方面有不足之处。它不允许用户在服务器上整理她的邮件；用户在服

务器上不能有不同的文件夹。(当然, 用户可以在她自己的计算机上创建各种文件夹)此外, POP3 不允许用户在下载邮件之前部分地检查邮件的内容。

IMAP4 提供了以下一些更多的功能:

- 用户在下载邮件之前可以检查邮件的首部。
- 用户在下载邮件之前可以用特定的字符串搜索电子邮件的内容。
- 用户可以部分地下载电子邮件。如果在电子邮件中包含了高带宽需求的多媒体信息而带宽又受到限制, 那么 IMAP4 就特别有用。
- 用户可以在邮件服务器上创建、删除邮箱, 或对邮箱更名。
- 为了存放电子邮件, 用户可以在文件夹中创建分层次的邮箱。

23.5 MIME

电子邮件的结构很简单。但是它的简单是有代价的。它只能发送使用 NVT 7 位 ASCII 码格式的报文。换言之, 它有些限制。例如, 它不能使用英文之外的语言(如法文、德文、希伯来文、俄文、中文以及日文)。此外, 它不能用来发送二进制文件或发送视频和音频数据。

多用途因特网邮件扩充 (MIME) 是一个辅助协议, 它允许非 ASCII 数据能够通过电子邮件传送。MIME 在发送方把非 ASCII 数据转换为 NVT ASCII 数据, 并把它交付给客户 MTA, 通过因特网发送出去。在接收方报文再转换为原来的数据。

我们可以把 MIME 想象为一组软件功能, 它能够把非 ASCII 数据转换为 ASCII 数据, 以及进行相反的转换(见图 23.15)。

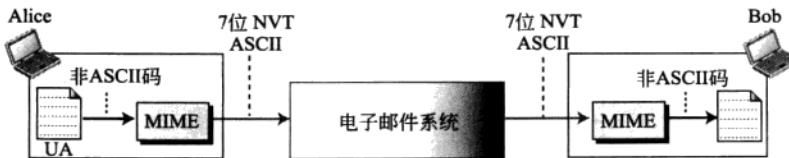


图 23.15 MIME

23.5.1 MIME 首部

MIME 定义了 5 种首部, 用来加在原始的电子邮件首部区域以定义转换参数:

1. MIME 版本 (MIME-Version)
2. 内容-类型 (Content-Type)
3. 内容-传送-编码 (Content-Transfer-Encoding)
4. 内容-标识 (Content-Id)
5. 内容-描述 (Content-Description)

图 23.16 给出了 MIME 首部。我们将详细地讨论每一个首部。

MIME-Version

这个首部定义 MIME 使用的版本, 当前的版本是 1.1。

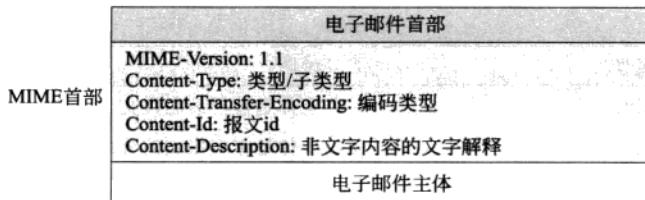


图 23.16 MIME 的首部

Content-Type

这个首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斜线分隔开。根据子类型的不同，首部还可包含其他一些参数。MIME 允许 7 种不同的数据类型。这些都列举在表 23.3 中。

- 正文** 原始报文是 7 位的 ASCII 格式，不需要用 MIME 来转换。现在有两种子类型：纯文字和 HTML。
- 多部分** 主体包含多个独立的部分。多部分首部需要定义每一个部分的边界。为此目的使用了一个参数。这个参数是一个标记串，放在每部分的前面；它占据单独一行，前面有两个连字符。主体结束的地方也使用边界标记，前面仍然有两个连字符，后面以两个连字符结束。该类型定义了 4 种子类型：混合、并行、摘要和交替。在混合子类型中，必须提供给收件人的那些部分与报文中的顺序完全一样。每部分有不同的类型，并在边界被定义。并行子类型与混合子类型相似，除了各部分的顺序是不重要的。摘要子类型也和混合子类型相似，除了默认的类型/子类型是如后文所定义的 message/RFC822。在交替子类型中，同样的报文使用不同的格式重复着。

下面是使用混合子类型的多部分报文的例子：

```
Content-Type: multipart/mixed; boundary=xxxx
```

```
--xxxx
Content-Type: text/plain;
-----
--xxxx
Content-Type: image/gif;
-----
--xxxx--
```

表 23.3 MIME 中的数据类型和子类型

类型	子类型	说明
正文	Plain（纯文字）	无格式的正文
	HTML	HTML 格式（见第 22 章）
多部分	Mixed（混合）	主体包含不同数据类型的有序部分
	Parallel（并行）	同上，但无序
	Digest（摘要）	与“混合”相似，但默认的是 message/RFC822
	Alternative（可供选择的）	几个部分是同一个报文的不同版本

续表

类型	子类型	说明
报文	RFC822	主体是被封装的报文
	Partial (部分)	主体是更大报文的分片
	External-Body (外部主体)	主体是到另一个报文的索引
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
音频	Basic (基本)	8 kHz 的单声道语音编码
	PostScript	Adobe PostScript
应用	Octet-stream (八位组流)	一般的二进制数据 (8 位字节)

- **报文** 在报文类型中，主体就是完整的邮件报文、邮件的一部分，或到邮件报文的指针。现在使用的有 3 种子类型：RFC822、部分或外部主体。RFC822 子类型用于主体被封装在另一个报文中（包括首部和主体）的情况。部分子类型用于原始报文被分片成不同的邮件报文，而这个邮件报文是这些分片之一的情况。这些分片必须在终点由 MIME 进行重装。还有三个参数必须加上：id、编号和总数。id 对报文进行标志，它在所有的分片中都出现。“编号”定义分片的顺序。“总数”定义组成原始报文的分片数。下面是由 3 个分片组成的报文例子：

```
Content-Type: message/partial;
id="forouzan@challenger.atc.fhda.edu";
number=1;
total=3;
```

.....
.....

子类型外部主体指出主体不包含真正的报文，而仅有对原始报文的索引（指针）。子类型后面的参数定义如何找到原始报文。下面是例子：

```
Content-Type: message/external-body;
name="report.txt";
site="fhda.edu";
access-type="ftp";
```

.....
.....

- **图像** 原始报文是一幅静止图像，指出没有动画。当前使用的两种子类型是使用图像压缩的联合照相专家组格式 (JPEG) 和图形交换格式 (GIF)。
- **视频** 原始报文是时变图像 (动画)。唯一的子类型是电视图像专家组 (MPEG)。若动画包括声音，则必须使用音频内容类型分开发送。
- **音频** 原始报文是声音。唯一的子类型是使用 8 kHz 标准音频数据的基本子类型。
- **应用** 原始报文是一种前面没有定义的数据类型。现在使用的只有两种子类型：

PostScript 和八位组流。PostScript 用于数据为 Adobe PostScript 格式的情况。八位组流用于数据必须解释为 8 位字节序列（二进制文件）的情况。

Cotent-Transfer-Encoding

这个首部定义把报文编码为便于传输的 0 和 1 的方法：

Content-Transfer-Encoding: <type>

表 23.4 举了 5 种类型的编码。

表 23.4 Content-Transfer-Encoding

类型	说明
7bit	NVT ASCII 字符和短行
8bit	非 ASCII 字符和短行
二进制	ASCII 字符和长度不限的行
Base64	6 位数据块被编码成 8 位 ASCII 字符
引用可打印	非 ASCII 字符被编码成等号后面跟随一个 ASCII 码

- **7bit** 这是 7 位 NVT ASCII 编码。虽然不需要特殊的信息，但行的长度不能超过 1000 字符。
- **8bit** 这是 8 位编码。非 ASCII 字符可以发送，但行的长度仍不能超过 1000 字符。这里 MIME 不进行任何编码；下面的 SMTP 协议必须能够传送 8 位非 ASCII 字符。因此，不推荐这种类型。应尽量使用 Base64 和引用可打印类型。
- **二进制** 这是 8 位编码。非 ASCII 字符可以发送，且行的长度可以超过 1000 字符。这里 MIME 并不进行任何编码；下面的 SMTP 协议必须能够传送二进制数据。因此，不推荐这种类型。应尽量使用 Base64 和引用可打印类型。
- **Base64** 当要发送的数据是由字节组成且最高位不一定是 0 时，这种类型是一种解决问题的方法。Base64 把这种类型的数据转换为可打印字符，然后就可以作为 ASCII 字符或底层邮件传送机制支持的任何类型的字符集发送出去。Base64 把二进制数据（由比特流组成）划分为 24 比特的块。每一块再分为 4 个部分，每部分由 6 比特组成（见图 23.17）。

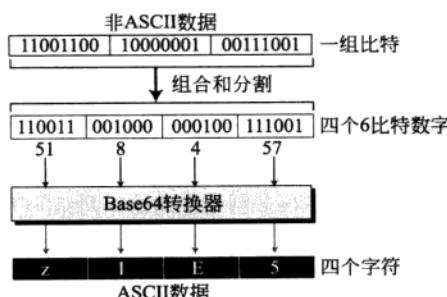


图 23.17 Base64

每一个 6 比特部分按照表 23.5 被解释为一个字符。

表 23.5 Base64 编码表

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

□ 引用可打印(Quoted-printable) Base64 是冗余编码方案；它把 24 比特变为 4 个字符，而最终发出 32 比特。开销为 25%。若数据由大部分的 ASCII 字符和一小部分非 ASCII 字符组成，那么我们就可以使用引用可打印编码。若字符是 ASCII，则就按原样发送。若字符是非 ASCII，则用 3 个字符发送出。第一个字符是等号(=)。后两个字符是用十六进制表示的字节。图 23.18 给出了例子。

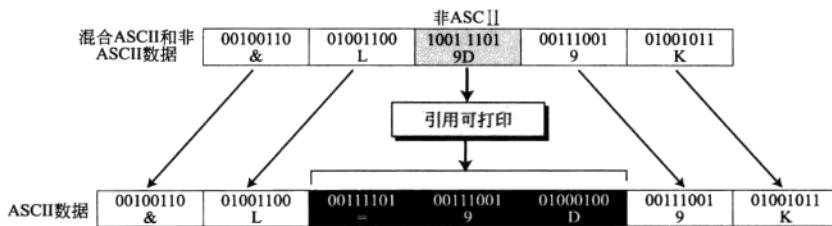


图 23.18 引用可打印

Content-Id

这个首部在多报文环境中唯一地标志整个的报文。

Content-Description

这个首部定义主体是否为图像、音频或视频。

23.6 基于万维网的邮件

电子邮件是一种很普通的应用，现在很多网站也向访问网站的所有人提供这种服务。有三个流行的网站称为 Hotmail、Yahoo 和 Google。这种概念是很简单的。让我们看以下两个案例。

23.6.1 案例一

这个案例中，Alice 是发件人，她使用传统的邮件服务器。Bob 是收件人，他在万维网

服务器上有一个账户。邮件通过 SMTP 从 Alice 的浏览器传送到邮件服务器。从发送邮件服务器到接收邮件服务器传送报文仍然使用 SMTP。然而从接收服务器（万维网服务器）到 Bob 的浏览器之间传送报文却使用 HTTP。换言之，在大多数情况下使用 HTTP 而非 POP3 或 IMAP4。当 Bob 需要读取他的电子邮件时，他发送一个 HTTP 报文请求到这个网站（例如 Hotmail）。这个网站则发送一份表单让 Bob 填入信息，这包括登录名称和密码。如果登录名称和密码都匹配的话，电子邮件清单会通过 HTML 格式从万维网服务器传送到 Bob 的浏览器上。这样，Bob 就可以浏览他收到的电子邮件。通过更多的 HTTP 事务，他可以逐一得到他的电子邮件。图 23.19 演示了这个案例。

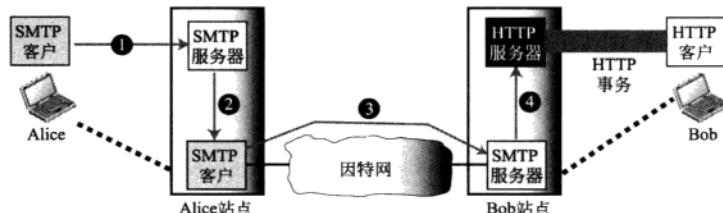


图 23.19 基于万维网的邮件，案例一

23.6.2 案例二

在第二个案例中，Alice 和 Bob 都使用万维网服务器，但并非一定是同一台服务器。Alice 使用 HTTP 事务来发送报文到服务器。Alice 发送一个 HTTP 请求报文到她的万维网服务器，这个报文使用 Bob 的邮箱名称和地址作为统一资源定位符 (URL)。Alice 这边的服务器把报文传给 SMTP 客户，然后使用 SMTP 协议将它发送到 Bob 这边的服务器。Bob 使用 HTTP 事务接收报文，但是报文从 Alice 这边的服务器到 Bob 这边的服务器仍然使用 SMTP 协议。图 23.20 演示了这种情况。

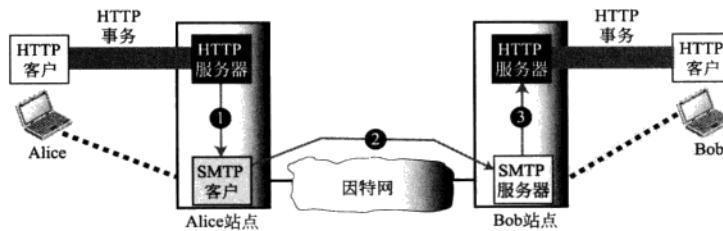


图 23.20 基于万维网的邮件，案例二

23.7 电子邮件的安全性

本章所讨论的协议本身并没有提供任何安全保障。然而，电子邮件的交换可以使用两个专为电子邮件系统而设计的应用层安全机制来实现安全性。其中的两个协议，相当好的保密 (PGP) 和安全 MIME (S/MIME) 将在我们讨论过基本网络安全之后的第 30 章中讨论。

23.8 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。被方括号括起来的书目可以在本书末尾的参考书目清单中找到。

23.8.1 参考书

有几本书深入浅出地覆盖了有关电子邮件的内容，它们包括[Com 06]、[Ste 94]、[Tan 03] 和[Kur & Ros 08]。

23.8.2 RFC

有几份 RFC 给出了有关 SMTP 的更新内容，包括 RFC 2821 和 RFC2822。POP3 在 RFC1939 中有解释。有几份 RFC 涉及了 MIME，包括 RFC2046、RFC2047、RFC2048 和 RFC2049。

23.9 重要术语

别名	本地部分
主体	报文读取代理（MAA）
连接建立	报文传送代理（MTA）
连接终止	多用途互联网邮件扩充（MIME）
域名	邮局协议版本 3（POP3）
信封	简单邮件传送协议（SMTP）
首部	用户代理（UA）
互联网邮件读取协议版本 4（IMAP4）	

23.10 本章小结

- 电子邮件是互联网上最常见的一种应用。电子邮件体系结构包括很多部分，例如用户代理（UA）、报文传送代理（MTA）和报文读取代理（MAA）。
- UA 准备报文、创建信封，并把报文放入信封。邮件地址包括两部分：本地部分（用户邮箱）和域名，其形式是 localpart@domainname。别名允许使用发件清单。
- MTA 在互联网上、局域网上或广域网上传送邮件。实现 MTA 的协议称为简单邮件传送协议（SMTP）。SMTP 使用命令和响应在 MTA 客户和 MTA 服务器之间传送报文。传送邮件报文的步骤是：连接建立、报文传送和连接终止。

- 实现 MAA 使用了两种协议：邮局协议版本 3 (POP3) 和互联网邮件读取协议版本 4 (IMAP4)。它们都是用来从邮件服务器拉取报文的协议。
- 多用途互联网邮件扩充 (MIME) 允许传送多媒体报文。MIME 将多媒体字符转换成可以通过电子邮件系统传送的 ASCII 字符。
- 基于万维网的电子邮件变得越来越普及，这些系统通过网站为用户提供免费电子邮件服务。在基于万维网的电子邮件系统中，部分数据传送使用 SMTP 完成，部分则通过 HTTP 协议完成。
- 安全电子邮件可以通过两种技术实现：相当好的保密(PGP)和安全 MIME (SMIME)。

23.11 实践安排

23.11.1 习题

1. 发件人发送一个无格式的文本。试给出 MIME 首部。
2. 发件人发送一个 JPEG 报文。试给出 MIME 首部。
3. 1000 字节的非 ASCII 报文使用 Base64 进行编码。试问编码后的报文有多少字节？有多少字节是冗余的？冗余字节与报文总字节之比是多少？
4. 1000 字节的报文使用引用可打印进行编码。报文包括 90% 的 ASCII 和 10% 的非 ASCII 字符。试问编码后的报文有多少字节？有多少字节是冗余的？冗余字节与报文总字节之比是多少？
5. 试比较习题 3 和 4 的结果。若报文是 ASCII 和非 ASCII 的混合，则效率能改进多少？
6. 试将以下报文编码为 Base64 格式：
01010111 00001111 11110000 10101111 01110001 01010100
7. 试将以下报文编码为引用可打印格式：
01010111 00001111 11110000 10101111 01110001 01010100
8. 试将以下报文编码为 Base64 格式：
01010111 00001111 11110000 10101111 01110001
9. 试将以下报文编码为引用可打印格式：
01010111 00001111 11110000 10101111 01110001
10. HELO 和 MAIL FROM 命令都是必需的吗？为什么或为什么不是？
11. 在图 23.11 中，信封中的 MAIL FROM 和首部中的 FROM 有何不同？
12. 若已经建立了 TCP 连接，为什么要传送邮件还要建立 TCP 连接？
13. 试给出从 aaa@xxx.com 到 bbb@yyy.com 的连接建立阶段。
14. 试给出从 aaa@xxx.com 到 bbb@yyy.com 的报文传送阶段。报文是“Good morning my friend”。
15. 试给出从 aaa@xxx.com 到 bbb@yyy.com 的连接终止阶段。
16. 用户 aaa@xxx.com 把一个报文发送给 bbb@yyy.com，该报文又被转发给

ccc@zzz.com。试给出所有的 SMTP 命令和响应。

17. 用户 aaa@xxx.com 把一个报文发送给 bbb@yyy.com。后者回复了。试给出所有的 SMTP 命令和响应。

18. 在 SMTP 中，若我们在两个用户之间只发送一行报文，试问要交换的命令和响应共有多少行？

23.11.2 研究活动

19. 一个 SMTP 新的版本，称为 ESMTP，现在已在使用。试找出它们之间的区别。

20. 试找出有关 smileys 用来表示用户感情的信息。

第 24 章 网络管理（SNMP）

简 单网络管理协议（Simple Network Management Protocol, SNMP）是用 TCP/IP 协议族对互联网上的设备进行管理的框架。它提供了一组基本的操作，用来监控和维护互联网。

目标

本章有以下几个目标：

- 讨论 SNMP，它是用 TCP/IP 协议族对互联网上的设备进行管理的框架。
- 定义了管理器和代理，管理器是运行 SNMP 客户程序的主机，代理是运行 SNMP 服务器程序的路由器或主机。
- 讨论 SNMP 所使用的 SMI 和 MIB。
- 说明 SMI 如何为对象命名，如何定义数据类型以及对数据编码。
- 说明如何利用 ASN.1 来定义数据类型。
- 说明 SMI 如何利用 BER 对数据进行编码。
- 通过三种方式来说明 SNMP 的功能。
- 讨论 SNMP 报文格式。
- 说明 SNMP 如何使用两个不同的 UDP 端口。
- 说明 SNMPv3 如何在其之前版本的基础上增强了安全性。

24.1 概念

SNMP 使用了管理器和代理的概念。也就是说，管理器（通常是主机）控制和监视着一组代理，它们通常是路由器或服务器（见图 24.1）。

SNMP 是应用级的协议，它以少量的管理器站对一组代理进行控制。这个协议被设计在应用级，因而能够监视安装在不同的物理网络上由不同厂家制造的设备。换言之，SNMP 使得管理任务与被管设备的物理特性和底层组网技术都没有关系。它可以用在由不同厂家制造的路由器连接起来的不同局域网和广域网的异构互联网中。

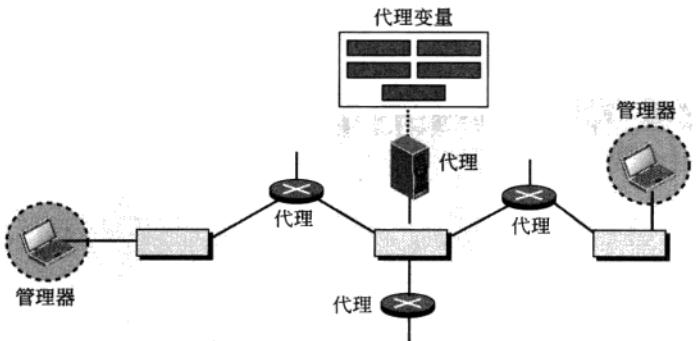


图 24.1 SNMP 的概念

24.1.1 管理器和代理

管理站称为管理器（manager），是运行了 SNMP 客户程序的主机。被管理的站称为代理（agent），是运行了 SNMP 服务器程序的路由器（或主机）。管理是通过管理器和代理之间的简单交互来实现的。

代理把性能信息保存在数据库中。管理器可以使用这个数据库中的值。例如，路由器可以把已收到的和已转发的分组数量保存在相应的变量中。管理器可以读取和比较这两个变量，以便发现路由器是否拥塞。

管理器还可以让路由器执行某些特定动作。例如，路由器定期地检查重新引导计数器的值，以便知道何时应当重新引导自己。比如说，当计数器的值为 0 时就应当重新引导自己。管理器可以利用这一点，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，迫使这个计数器的值为 0。

代理也可以参与到管理过程中。在代理上运行的服务器程序可以检查环境，若发现有异常现象就可以向管理器发送告警报文（称为陷阱）。

总之，使用 SNMP 的管理基于以下三个基本思想：

1. 管理器通过请求获取信息来检查代理，这些信息能够反映代理的行为。
2. 管理器可以重新设置代理数据库中的某些值，以便强迫代理完成某个任务。
3. 代理参与管理过程的方法是向管理器发出对异常情况的告警。

24.2 管理构件

为了完成管理任务，SNMP 还要用到另外两个协议：管理信息结构（Structure of Management Information，SMI）和管理信息库（Management Information Base，MIB）。换言之，对因特网的管理要通过三个协议的共同协作：SNMP、SMI 和 MIB，如图 24.2 所示。

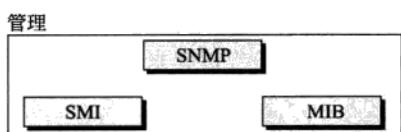


图 24.2 在因特网上网络管理的构件

让我们详细描述这些协议之间的相互关系。

24.2.1 SNMP 的作用

SNMP 在网络管理中起着非常特殊的作用。它定义了从管理器发送到代理以及从代理发送到管理器的分组格式。它还要解释结果并产生统计（常常需要依靠其他管理软件的帮助）。所交换的分组包含对象（变量）名和它们的状态（值）。SNMP 负责读取和改变这些值。

SNMP 定义了管理器和代理之间交换的分组的格式。它读取和改变在 SNMP 分组中的对象的状态（变量的值）。

24.2.2 SMI 的作用

要使用 SNMP，就需要有一些规则。我们需要有命名对象的规则。这一点特别重要，因为 SNMP 的对象形成了一种层次结构（一个对象可以有一个父对象和几个子对象）。名字的一部分可以是从父对象继承来的。我们还需要定义对象类型的规则。SNMP 可以处理什么类型的对象？SNMP 可以处理简单类型还是结构化类型？可以有多少种简单类型？这些类型的大小是什么？这些类型的范围是什么？此外，每一种类型是如何编码的？

SMI 定义了一些通用规则，用于命名对象，定义对象类型（包括范围和长度），以及说明如何对这些对象和值进行编码。

我们需要这些通用规则，因为我们并不知道发送、接收或存储这些值的计算机的体系结构。发送站可能是一个高性能计算机，它用 8 字节数据存储一个整数，而接收站则可能是一个小型计算机，它使用 4 字节数据存储一个整数。

SMI 是定义这些规则的协议。但是我们必须知道，SMI 只是定义了这些规则，它并没有定义在一个实体中管理了多少个对象，或哪个对象使用的是哪一种类型。SMI 是通用规则的集合，这些规则用来命名对象并列出它们的类型清单。对象和类型的关联并不是 SMI 要做的事。

24.2.3 MIB 的作用

希望此刻大家应该已经明白，我们还需要另一个协议。对于每一个被管理的实体，这个协议要定义它的对象数量，并按照 SMI 定义的规则给这些对象命名，还要让每一个命名的对象与某种类型关联起来，这个协议就是 MIB。MIB 创建了为每个实体而定义的一组对象，类似于数据库（就像大多数数据库的元数据那样，是一些没有值的名称和类型）。

MIB 为被管理的实体创建了一组命名的对象、它们的类型以及它们彼此之间的关系。

24.2.4 类比

在详细讨论这几个协议之前，让我们先来打一个比方。这三个网络管理构件与我们为

解决某个问题而用某种计算机语言编写程序是类似的。图 24.3 给出了此种类比。

语法：SMI

在我们编写程序之前，必须有预先定义的语言语法（例如 C 或 Java）。这个语言还要定义变量的结构（简单的、结构化的、指针，等等），规定这些变量如何命名。例如，一个变量名的长度必须是 1 到 n 个字符，并且必须以一个字母开头，后面接着的是字母或数字字符。这个语言还要定义使用的数据类型（整数、浮点、字符，等等）。在编程中这些规则是由这个语言的语法所定义的。在网络管理中，这些规则由 SMI 来定义。

对象的声明和定义：MIB

大多数计算机的语言都要求在每个具体的程序中要对变量进行定义和声明。定义和声明使用预定义的类型来创建对象并为其分配内存地址。例如，若某个程序有两个变量（一个是命名为 counter 的整数，另一个是命名为 grades 的 char 类型的数组），那么在程序的一开始就必须先对这两个变量进行声明：

```
int counter;
char grades [40];
```

MIB 在网络管理中就做这样的事情。MIB 给每个对象命名，并定义对象的类型。因为这些类型是由 SMI 定义的，所以 SNMP 能够知道它们的范围和大小。

程序编码：SNMP

在程序中的声明语句之后，就需要写出一些语句，以便在变量中保存一些的值，并在需要时改变它们。SNMP 在网络管理中完成同样的任务。按照 SMI 定义的规则，SNMP 存储、改变和解释这些已经由 MIB 声明的对象的值。

24.2.5 概览

在详细讨论每一个构件之前，让我们用一个简单的情景来说明每个构件起到怎样的作用。这只是一个概览，而在本章的后面将会进一步展开。一个管理器站（SNMP 客户）希望向一个代理站（SNMP 服务器）发送报文，以便了解这个代理收到的 UDP 用户数据报的数量。图 24.4 概要地给出了所涉及到的几个步骤。

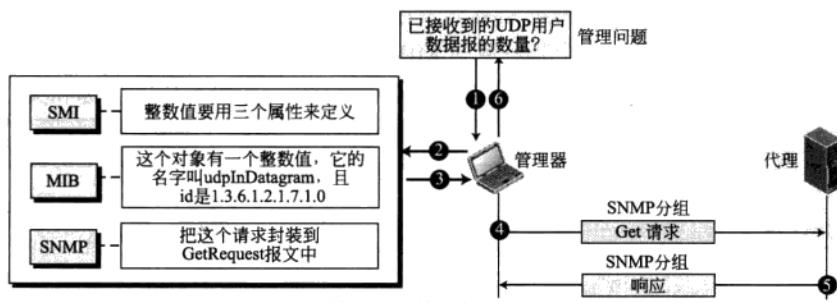


图 24.4 管理概览图

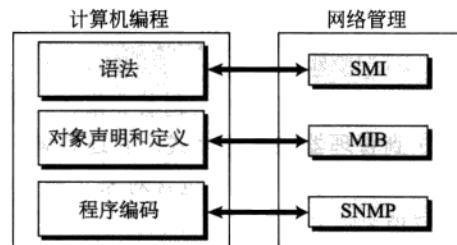


图 24.3 比较计算机编程和网络管理

MIB 负责找出存储了收到的 UDP 用户数据报数量的对象。SMI 在其他嵌入协议的帮助下，负责对这个对象的名字进行编码。SNMP 负责创建一个称为 GetRequest 的报文，并把已编码的报文进行封装。当然，与这个简单的概览图相比，实际情况要复杂得多，但我们首先需要更加详细地了解每一个协议。

24.3 SMI

管理信息结构版本 2 (SMIV2) 是网络管理中的一个构件。它的功能是：

1. 为对象命名。
2. 定义可在对象中存储的数据的类型。
3. 给出如何对网络上传输的数据进行编码的方法。

SMI 是 SNMP 的一个工作指南。它强调了处理对象所需的三个属性：名字、数据类型和编码方法。

24.3.1 名字

SMI 要求每个被管理的对象（如某个路由器、路由器中的某个变量、某个值，等等）都具有唯一的名字。为了全局性地给对象命名，SMI 使用了对象标识符 (object identifier)，它是基于树结构的分层次的标识符（见图 24.5）。

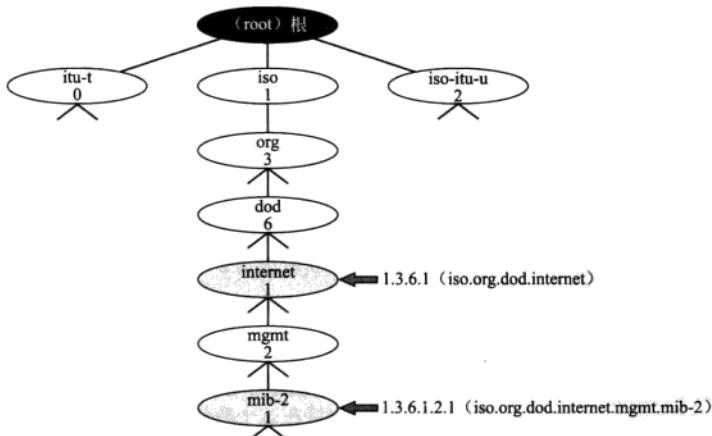


图 24.5 对象标识符

这个树结构从一个不命名的根开始。每个对象都可以用点分隔开的整数序列来定义。这个树结构也可以使用点分隔开的文本名字序列来定义对象。在 SNMP 中使用的是整数-点的表示方法。名字-点的记法是给人类使用的。例如，下面给出了同一个对象的两种不同记法：

iso.org.dod.internet.mgmt.mib-2



1.3.6.1.2.1

SNMP 使用的所有对象都位于 mib-2 对象的下面，因此它们的标识符永远从 1.3.6.1.2.1 开始。

所有被 SNMP 管理的对象都要被赋予一个对象标识符。

这个对象标识符永远以 1.3.6.1.2.1 为开始。

24.3.2 类型

对象的第二个属性是它所存储的数据的类型。为了定义数据类型，SMI 使用了**抽象语法记法 1** (Abstract Syntax Notation 1, ASN.1) 的基本定义，同时也增加了几个新的定义。换言之，SMI 既是 ASN.1 的一个子集，也是 ASN.1 的一个超集。

SMI 使用了两大类数据类型：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 types) 是原子数据类型。它们中有一些直接取自 ASN.1，另一些是 SMI 新增的。表 24.1 给出了其中最重要的一些类型。前五个取自 ASN.1，后七个是 SMI 定义的。

表 24.1 数据类型

类型	大小	说明
INTEGER	4 字节	在 $-2^{31} \sim 2^{31}-1$ 之间的整数值
Interger32	4 字节	和 INTEGER 相同
Unsigned32	4 字节	在 $0 \sim 2^{32}-1$ 之间的无符号值
OCTET STRING	可变	不超过 65535 字节长的字节串
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	可从 0 增加到 2^{32} 的整数，当它到达最大值时就返回到 0
Counter64	8 字节	64 位的计数器
Gauge32	4 字节	与 Counter32 相同，但当它到达最大值时不返回，而是保持在这个数值直到复位
TimeTicks	4 字节	记录时间的计数值，以 1/100 秒为单位
BITS		比特串
Opaque	可变	不解释的串

结构化类型

把简单的和结构化的数据类型组合起来，我们就可以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 types): sequence 和 sequence of。

- **Sequence** sequence 数据类型是简单数据类型的组合，但不必都是相同的类型。它与类似 C 的编程语言中使用的 struct 或 record 的概念相似。
- **Sequence of** 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。它与类似 C 的编程语言中使用的 array 的概念相似。

图 24.6 给出了数据类型的概念性图示。

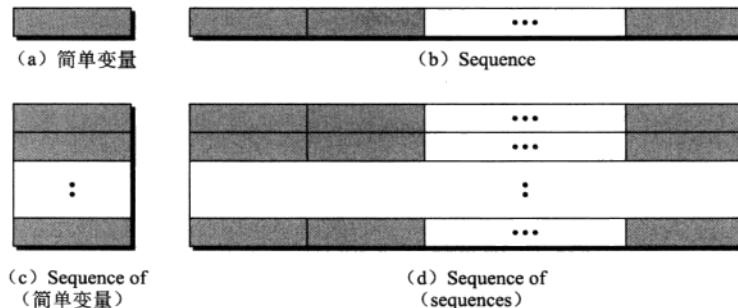


图 24.6 概念性的数据类型

24.3.3 编码方法

SMI 利用另外的标准，即基本编码规则（Basic Encoding Rules, BER），对即将在网络上传输的数据进行编码。BER 指明了每一块数据都要被编码成为三元组格式：标记、长度和值，如图 24.7 所示。

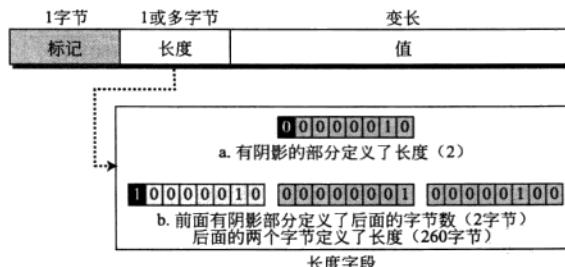


图 24.7 编码格式

标记是 1 字节字段，它定义了数据的类型。表 24.2 给出了我们在本章用到的数据类型，以及用二进制和十六进制表示的相应标记。长度字段为 1 个字节或多个字节。若它是 1 字节，则最高位必定为 0，其余的 7 位定义了数据的长度。若它大于 1 字节，则第一字节的最高位必定为 1。第一个字节的其余 7 位则定义长度所需的字节数。值字段按照 BER 中定义的规则对数据的值进行编码。

表 24.2 数据类型的代码

数据类型	标记（二进制）	标记（十六进制）
INTEGER	00000010	02
OCTET STRING	00000100	04
OBJECT IDENTIFIER	00000110	06
NULL	00000101	05
Sequence, sequence of	00110000	30
IPAddress	01000000	40
Counter	01000001	41
Gauge	01000010	42
TimeTicks	01000011	43
Opaque	01000100	44

例 24.1

图 24.8 所示为如何定义 INTEGER 14。请注意我们同时使用了二进制和十六进制来表示这个标记。长度字段的大小来自表 24.1。

02	04	00	00	00	0E
00000010	00000100	00000000	00000000	00000000	00001110
标记 (INTEGER)	长度 (4 字节)	值 (14)			

图 24.8 例 24.1: INTEGER 14

例 24.2

图 24.9 所示为如何定义 OCTET STRING “HI”。

04	02	48	49
00000100	00000010	01001000	01001001
标记 (OCTET STRING)	长度 (2 字节)	值 (H)	值 (I)

图 24.9 例 24.2: OCTET STRING “HI”

例 24.3

图 24.10 所示为如何定义 ObjectIdentifier 1.3.6.1 (iso.org.dod.internet)。

06	04	01	03	06	01
00000110	00000100	00000001	00000011	00000110	00000001
标记 (Objectld)	长度 (4 字节)	值 (1)	值 (3)	值 (6)	值 (1)

|————— 1.3.6.1 (iso.org.dod.internet) —————|

图 24.10 例 24.3: ObjectIdentifier 1.3.6.1

例 24.4

图 24.11 所示为如何定义 IPAddress 131.21.14.8。

40	04	83	15	0E	08
01000000	00000100	10000011	00010101	00001110	00001000

标记 (IPAddress) 长度 (4 字节) 值 (131) 值 (21) 值 (14) 值 (8)

131.21.14.8

图 24.11 例 24.4: IPAddress 131.21.14.8

24.4 MIB

管理信息库版本 2(MIB2)是网络管理中的第二个构件。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。MIB2 中的对象被划分为 10 个不同的组: system(系统)、interface(接口)、address translation(地址转换)、ip、icmp、tcp、udp、egp、transmission(传输)和 snmp。这些组都位于对象标识符树中 mib-2 对象的下面(见图 24.12)。每个组定义了一些变量和/或表。

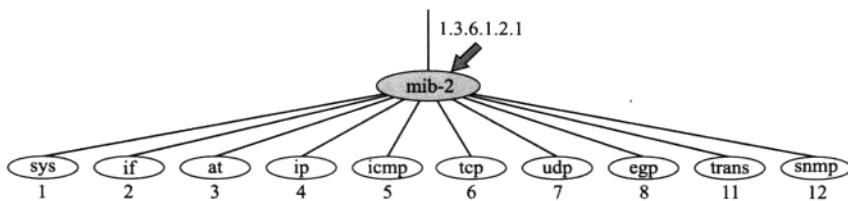


图 24.12 mib-2

下面是一些对象的简单描述。

- **sys** 这个对象(系统)定义有关结点(系统)的通用信息,如名字、位置和生存时间。
- **if** 这个对象(接口)定义与结点的所有接口有关的信息,如接口号、物理地址和 IP 地址。
- **at** 这个对象(地址转换)定义了有关 ARP 表的信息。
- **ip** 这个对象定义了与 IP 有关的信息,如路由表和 IP 地址。
- **icmp** 这个对象定义与 ICMP 有关的信息,如已发送和已收到的分组数,以及产生的差错总数。
- **tcp** 这个对象定义与 TCP 有关的通用信息,如连接表、超时值、端口号,以及已发送和已收到的分组数。
- **udp** 这个对象定义与 UDP 有关的通用信息,如端口号、已发送和已收到的分组数。
- **snmp** 这个对象定义与 SNMP 本身有关的通用信息。

24.4.1 访问 MIB 变量

为了说明如何访问各种变量,我们以 udp 组作为例子。在 udp 组有 4 个简单变量和一个记录序列(表)。图 24.13 给出了这些变量和表。

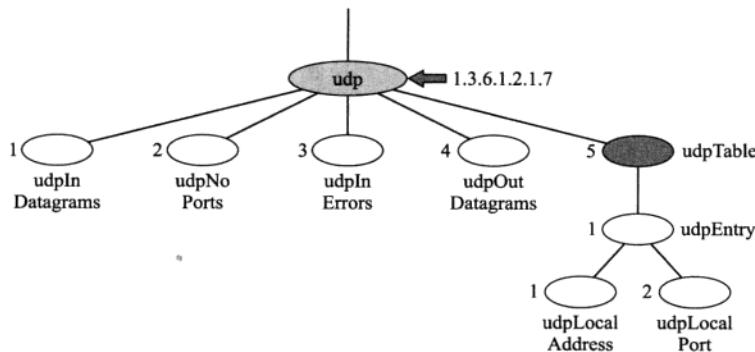


图 24.13 udp 组

我们将说明如何访问每一个实体。

简单变量

要访问任何简单变量，我们使用组的 id (1.3.6.1.2.1.7) 后面跟着该变量的 id。下面给出了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）。要给出每个变量的实例或内容，我们必须增加实例的后缀。简单变量的实例后缀就是零。换言之，要给出以上变量的实例，我们使用以下的方法：

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

为了标识表，我们首先要使用表的 id。如图 24.14 所示，udp 组只有一个表 (id 是 5)。因此，要访问这个表，我们使用以下方法：

udpTable	→	1.3.6.1.2.1.7.5
-----------------	---	-----------------

但是，这个表不是在树结构的树叶级。我们不能访问这个表，而是要定义这个表 (id 是 1) 中的项目 (sequence)，如下所示：

udpEntry	→	1.3.6.1.2.1.7.5.1
-----------------	---	-------------------

这个项目也不是树叶，我们还是不能访问它。我们需要定义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

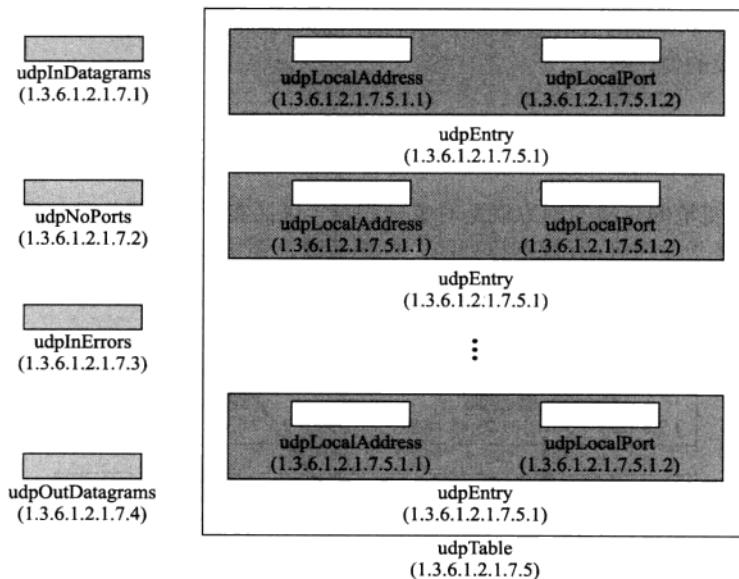


图 24.14 udp 的变量和表

这两个变量是在树叶上。虽然我们能够访问它们的实例，但我们需要定义是哪一个实例。在任何时候，这个表中的每一个“本地地址/本地端口对”可以有不同的值。要访问表中某个特定的实例（行），我们应当给上述 id 再加上索引。在 MIB 中，数组的索引不是整数（而在大多数编程语言中数组的索引都用整数表示）。这些索引基于项目中的一个或多个字段的值。在我们的例子中，`udpTable` 的索引基于本地地址和本地端口号。例如，图 24.15 给出了一个具有 4 行的表，以及每一个字段的值。每一行的索引是这两个值的组合。

181.23.45.14	23
1.3.6.1.2.1.7.5.1.1.181.23.45.14.23	1.3.6.1.2.1.7.5.1.2.181.23.45.14.23
192.13.5.10	161
1.3.6.1.2.1.7.5.1.1.192.13.5.10.161	1.3.6.1.2.1.7.5.1.2.192.13.5.10.161
227.2.45.18	180
1.3.6.1.2.1.7.5.1.1.227.2.45.18.180	1.3.6.1.2.1.7.5.1.2.227.2.45.18.180
230.20.5.24	212
1.3.6.1.2.1.7.5.1.1.230.20.5.24.212	1.3.6.1.2.1.7.5.1.2.230.20.5.24.212

图 24.15 udpTable 的索引

为了访问本地地址实例的第一行，我们使用标识符加上实例的索引：

`udpLocalAddress.181.23.45.14.23` → `1.3.6.1.2.1.7.5.1.1.181.23.45.14.23`

请注意，并非所有的表的索引都是这样的。某些表的索引是使用一个字段的值，有些是使用两个字段的值，等等。

24.4.2 字典式排序

关于 MIB 变量的有趣的地方就是对象标识符（包括实例标识符）是按照字典的顺序排列的。表的顺序是按照列–行规则排列的，也就是说，要按逐列的顺序走。在每一个列中，必须是从顶向底走，如图 24.16 所示。

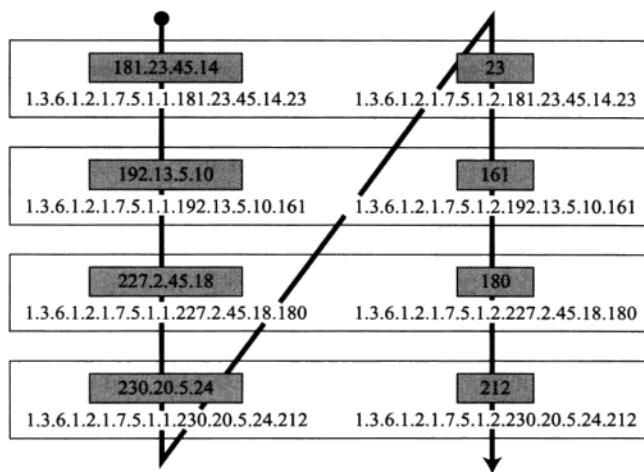


图 24.16 字典式排序

字典式排序 (lexicographic ordering) 使管理器在定义了第一个变量后，可以一个接一个地访问一组变量，我们将在下一节讨论 GetNextRequest 命令时看到这点。

24.5 SNMP

SNMP 在因特网的网络管理中使用了 SMI 和 MIB。SNMP 是应用程序，它允许：

1. 管理器读取代理定义的对象值。
2. 管理器把值存储在代理定义的对象中。
3. 代理把关于异常情况的告警报文发送给管理器。

24.5.1 PDU

SNMPv3 定义了八种类型的数据单元（或称为 PDU）：GetRequest、GetNextRequest、GetBulkRequest、SetRequest、Response、Trap、InformRequest 和 Report（见图 24.17）。

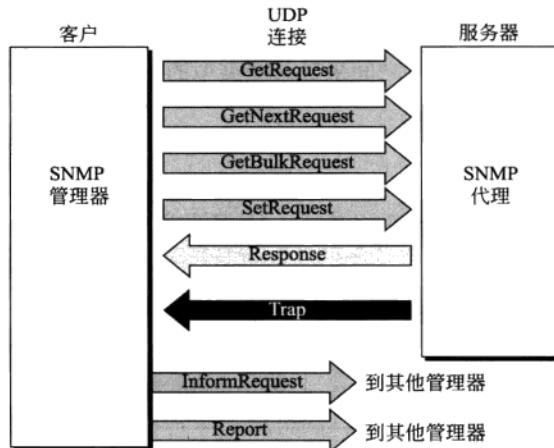


图 24.17 SNMP PDU

GetRequest (读取请求)

GetRequest PDU 是由管理器（客户）发送给代理（服务器），用来读取一个变量或一组变量的值。

GetNextRequest (读取下一个请求)

GetNextRequest PDU 是由管理器发送给代理，用来读取变量的值。所要读取的值是 PDU 中定义的 ObjectId 后面的对象的值。它主要是用来读取一个表中的项目的值。若管理器不知道这个项目的索引，它就不能读取这个值。但是，它可以使用 GetNextRequest 以及定义表的 ObjectId 实现这一点。因为第一个项目的 ObjectId 紧接在表的 ObjectId 的后面，因此用 GetNextRequest 返回的第一个项目的值就是所要结果。然后管理器可以使用这个 ObjectId 来得到下一个项目的值，依此类推。

GetBulkRequest (读取块请求)

GetBulkRequest PDU 由管理器发送给代理，用来读取大量的数据。它可用来代替多个 GetRequest PDU 和 GetNextRequest PDU。

SetRequest (设置请求)

SetRequest PDU 是由管理器发送给代理，用来设置（存储）变量值。

Response (响应)

Response PDU 是由代理发送给管理器以便响应 GetRequest 和 GetNextRequest。它包含管理器所请求的一个或多个变量的值。

Trap (陷阱)

Trap 有时也称为 SNMPv2 Trap，以便和 SNMPv1 Trap 区分开。Trap PDU 由代理发送给管理器，用来报告事件。例如，若代理重新被引导了，它就通知管理器，并报告重新引导的时间。

InformRequest (通知请求)

InformRequest PDU 由一个管理器发送给另一个远程管理器，以便从一些由远程管理器

控制下的代理那里得到一些变量的值。远程管理器用 Response PDU 响应。

Report (报告)

Report PDU 被设计用于管理器之间报告某些类型的差错。它现在还未使用。

24.5.2 格式

八种 SNMP PDU 的格式如图 24.18 所示。GetBulkRequest PDU 与其他种 PDU 的两处不同之处已标注在图中。

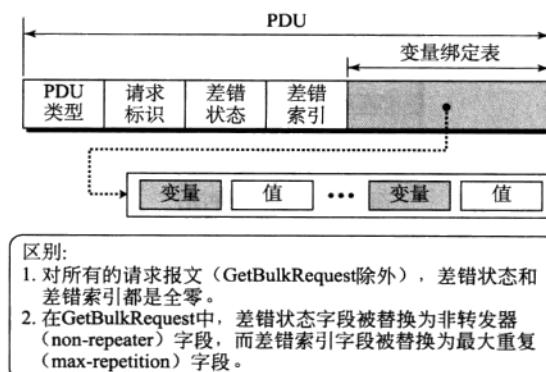


图 24.18 SNMP PDU 的格式

这些字段列举如下:

PDU 类型 这个字段定义了 PDU 的类型 (见表 24.3)。

表 24.3 PDU 类型

类型	标记 (二进制)	标记 (十六进制)
GetRequest	10100000	A0
GetNextRequest	10100001	A1
Response	10100010	A2
SetRequest	10100011	A3
GetBulkRequest	10100101	A5
InformRequest	10100110	A6
Trap(SNMPv2)	10100111	A7
Report	10101000	A8

- 请求标识 (Request ID)** 这个字段是一个序号，由管理器在请求 PDU 中使用，而代理在响应 PDU 中重复。它用来使响应和请求相匹配。
- 差错状态 (Error status)** 这是只用于响应 PDU 中的一个整数，用来给出代理报告的差错类型。在请求报文中它的值是 0。表 24.4 列举了可能出现差错类型。
- 非转发器 (Non-repeaters)** 这个字段只用在 GetBulkRequest 中，用来替换差错状态字段，本来请求 PDU 中的差错状态字段是空的。

- **差错索引 (Error index)** 差错索引是一个偏移值，它告诉管理器是哪个变量引起这个差错。

表 24.4 差错的类型

状态	名称	意义
0	noError	无差错
1	tooBig	响应太大无法放入一个报文
2	noSuchName	变量不存在
3	badValue	要存储的值无效
4	readOnly	不能修改这个值
5	genErr	其他差错

- **最大重复 (Max-repetition)** 这个字段也只用在 GetBulkRequest 中，用来替换差错索引字段，本来在请求 PDU 中差错索引字段是空的。
- **变量绑定表 (VarBindList)** 这是管理器希望读取或设置的一组具有相应值的变量。在 GetRequest 和 GetNextRequest 中它的值是空。在 Trap PDU 中，它给出了与各特定 PDU 有关的变量和值。

24.5.3 报文

SNMP 并不是只发送 PDU，而是把 PDU 嵌入在报文中。SNMPv3 的报文是 sequence 类型，由四部分组成：Version（版本）、GlobalData（全局数据）、SecurityParameters（安全参数）和 ScopePDU（它包含已编码的 PDU），如图 24.19 所示。第一和第三部分是简单数据类型。第二和第四部分是 sequence 类型。

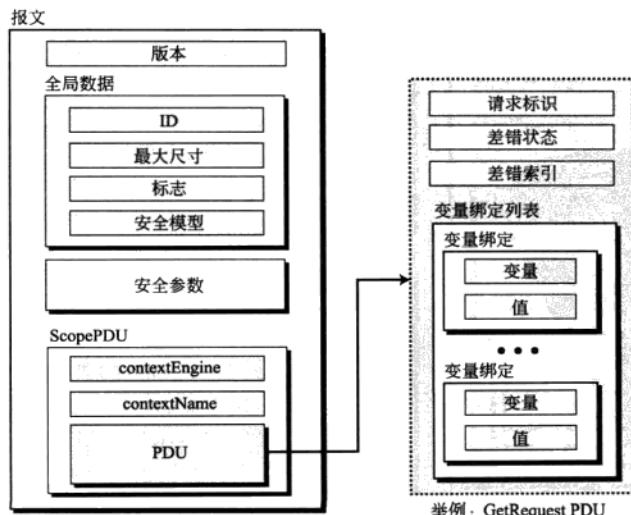


图 24.19 SNMP 报文

版本

版本字段的数据类型是 INTEGER，它定义了版本。当前版本为 3。

全局数据

全局数据字段是 sequence 类型，由四个简单数据类型的元素组成：ID（标识符）、Max-size（最大尺寸）、Flags（标志）和 SecurityModel（安全模型）。

安全参数

这一部分是 sequence 类型，在版本 3 中，根据安全实现的类型，它可以非常复杂。

ScopePDU

最后一个部分包含了两个简单数据类型和真正的 PDU。我们只给出过一个 GetRequest PDU 的例子。请注意 VarBindList（变量绑定列表）是 sequence 类型，由一个或多个被称为 VarBind（变量绑定）的 sequence 组成。每一个 VarBindList 由两个简单数据类型的元素构成：变量和值。

例 24.5

在本例中，管理器站（SNMP 客户）使用 GetRequest 报文读取路由器收到的 UDP 数据报数量（图 24.20）。它只有一个 VarBind 实体。与这个信息对应的 MIB 变量是 udpInDatagrams，其对象标识符是 1.3.6.1.2.1.7.1.0。管理器希望读取一个值（而不是存储），因此这个值定义一个空实体。发送的字节用十六进制表示。

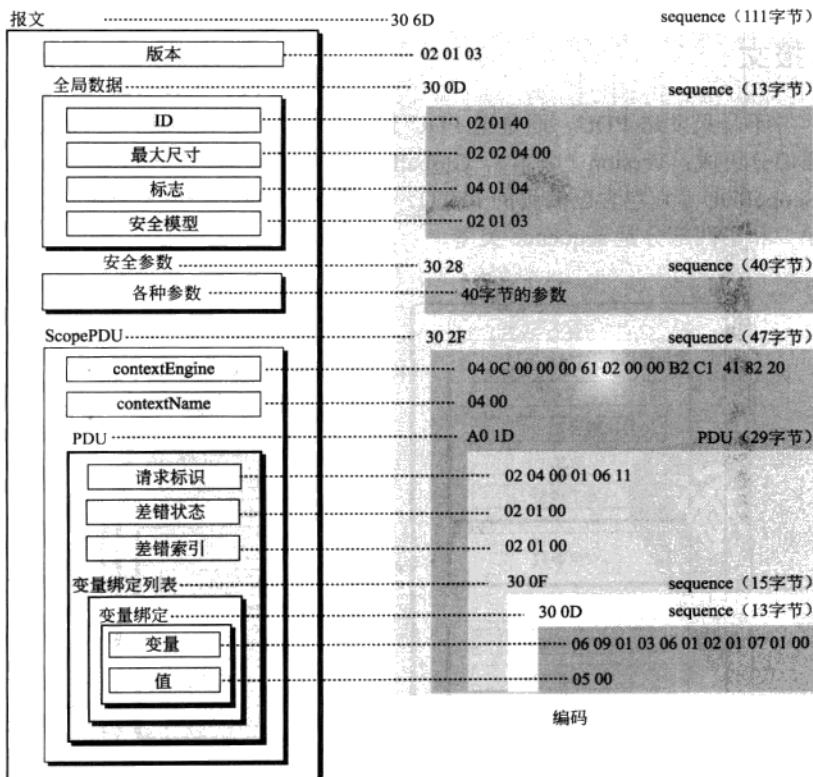


图 24.20 例 24.5

此处的 VarBindList 只有一个 VarBind，其中变量的类型为 06，长度为 09，值的类型为 05，长度为 00。整个 VarBind 是长度为 0D (13) 的 sequence，因而 VarBindList 是一个长度为 0F (15) 的 sequence。GetRequest PDU 的长度是 1D (29)。这个 PDU 内嵌在 ScopePDU 中，它是一个长度为 47 字节的 sequence。安全参数为 40 个字节，但这里没有给出细节。全局数据本身是 13 字节长的 sequence。因此这个报文一共内嵌了三个 sequence 和一个整数 (版本)，加起来长度为 111 字节。整个报文的长度为 113 字节。

请注意，我们在这里特别给出了字节数，以便说明一个 sequence 可以只包含简单数据类型，也可以包含 sequence 以及简单数据类型。需要注意的是 PDU 类似于 sequence，但是它的标记为 A0 (十六进制)。

图 24.21 给出了实际发出的报文。为了便于阅读，我们以每行 16 个字节的形式给出这段报文，但实际上报文是以每行 4 个字节的形式发出。用横线表示的字节是与安全参数有关的内容。

报文																			
30	6D	02	01	03	30	0D	02	01	04	02	02	04	00	04	01				
04	02	01	03	30	28	--	--	--	--	--	--	--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	30	2F	
04	0C	00	00	00	61	02	00	00	B2	C1	41	82	20	04	00				
A0	1D	02	04	00	01	06	11	02	01	00	02	01	00	30	0F				
30	0D	06	09	01	03	06	01	02	01	07	01	00	05	00					

图 24.21 例 24.5 中实际发送的报文

24.6 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(管理器)使用。

代理(服务器)在端口 161 上发出被动打开。然后它就等待来自管理器(客户)的连接。管理器(客户)使用临时端口发出主动打开。客户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。服务器向客户发送响应报文，使用熟知端口 161 作为源端口而临时端口作为目的端口。

管理器(客户)在端口 162 发出被动打开。然后它就等待从代理(服务器)来的连接。只要代理(服务器)有 Trap 报文要发送，就使用临时端口发出主动打开。这个连接是单向的，从服务器到客户(参见图 24.22)。

SNMP 中的客户-服务器机制与其他协议不同。这里的客户和服务器都使用了熟知端口。此外，客户和服务器都必须无限制地运行下去。其原因在于请求报文是由管理器(客户)发出的，但 Trap 报文则是由代理(服务器)发出的。

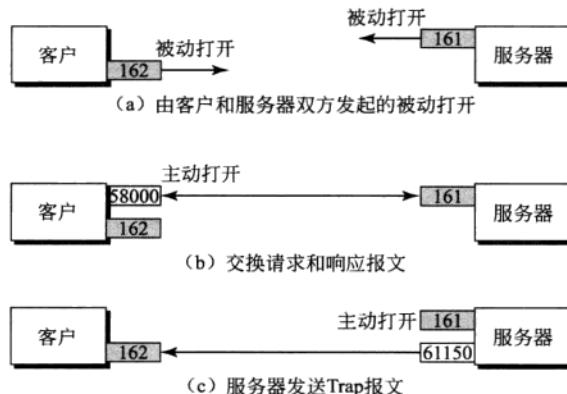


图 24.22 SNMP 的端口号

24.7 安全

SNMPv3 在之前的版本基础上新增了两个特性：安全性和远程管理。SNMPv3 允许管理器在访问代理时选择一级或多级安全设置。管理器可以设置不同的安全配置，以便实现报文鉴别、保密以及完整性。

SNMPv3 还允许管理器远程改变安全配置，这表示管理员不一定必须亲临设备所在地。

24.8 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

24.8.1 参考书

有几本书详尽地覆盖了有关 SNMP 的内容，它们包括[Com 06]、[Ste 94]、[Tan 03] 和 [Kur & Ros 08]。[Mau & Sch 01]是一本专注于讨论 SNMP 的书籍，我们推荐通过它深入学习该协议提供的不同特性。

24.8.2 RFC

有几份 RFC 给出了有关 SNMP 的更新内容，包括 RFC 3410、RFC 3412、RFC 3415 和 RFC 3418。有关 MIB 的更多信息可以在 RFC 2578、RFC 2579 和 RFC 2580 中找到。

24.9 重要术语

抽象语法记法 1 (ASN.1)	对象标识符
代理	简单数据类型
基本编码规则 (BER)	简单网络管理协议 (SNMP)
字典式排序	管理信息结构 (SMI)
管理信息库 (MIB)	结构化数据类型
管理器	陷阱 (Trap)

24.10 本章小结

- 简单网络管理协议(SNMP)是用 TCP/IP 协议族对互联网中的设备进行管理的框架。管理器(通常是一个主机)控制和监视一组代理(通常是路由器)。管理器是运行 SNMP 客户程序的主机。代理是运行 SNMP 服务器程序的路由器或主机。SNMP 使管理任务与被管设备的物理特性和下层的组网技术无关。SNMP 需要使用另外两个协议提供的服务: 管理信息结构 (SMI) 和管理信息库 (MIB)。
- SMI 为对象命名, 定义可存储在对象中的数据类型, 并对数据进行编码。SMI 的对象按照分层次的树结构来命名。SMI 的数据类型按照抽象语法记法 1 (ASN.1) 来定义。SMI 使用基本编码规则 (BER) 对数据进行编码。
- MIB 是能够被 SNMP 管理的对象组的集合。MIB 使用字典式排序管理它的变量。
- SNMP 的功能有三种实现方法: 管理器可以读取代理定义的对象值。管理器可以把一个值存储在代理定义的对象中。代理可以向管理器发送告警报文。
- SNMP 定义了八种类型的报文: GetRequest、GetNextRequest、SetRequest、GetBulkRequest、Trap、InformRequest、Response 和 Report。SNMP 在两个熟知端口 161 和 162 上使用 UDP 的服务。SNMPv3 与之前的版本相比, 具有增强的安全特性。
- 第三版的 SNMP 在之前版本上又增加了两个新特性: 不同级别的安全和远程管理。

24.11 实践安排

24.11.1 习题

1. 试给出 INTEGER 1456 的编码。
2. 试给出 OCTET STRING “Hello World”的编码。
3. 试给出任意长度为 1000 的 OCTET STRING 的编码。
4. 试说明下面的记录 (sequence) 是怎样编码的。

INTEGER 2345	OCTET STRING “COMPUTER”	IP Address 185.32.1.5
-----------------	----------------------------	--------------------------

5. 试说明下面的记录 (sequence) 是怎样编码的。

Time Tick 12000	OCTET STRING 14564	Object Id 1.3.6.1.2.1.7
--------------------	-----------------------	----------------------------

6. 试说明下面的数组 (sequence of) 是怎样编码的。其中每个成员都是整数。

2345	1236	122	1236
------	------	-----	------

7. 试使用本章所述的基本编码规则 (BER) 说明下面的记录数组 (sequence of sequence) 是怎样编码的。请注意，每列的标题给出了数据的类型。

INTEGER	OCTET STRING	Counter
2345	“COMPUTER”	345
1123	“DISK”	1430
3456	“MONITOR”	2313

8. 试进行下面解码：

- a. 02 04 01 02 14 32
- b. 30 06 02 01 11 02 01 14
- c. 30 09 04 03 41 43 42 02 02 14 14
- d. 30 0A 40 04 23 51 62 71 02 02 14 12

24.11.2 研究活动

9. 试找出有关抽象语法记法 1 (ASN.1) 的更多信息。

第 25 章 多 媒 体

近年来科学技术的发展已经改变了我们对音频和视频的使用。在过去，我们用无线电收音机收听音频广播节目，用电视机观看视频广播节目。我们还使用电话网络与另一方进行交互式的通信。但是时代已经改变了。人们希望利用因特网不仅进行文字和图像通信，还要得到音频和视频服务。在这一章，我们关注的重点是利用因特网提供音频和视频服务方面的应用。

目标

本章有以下几个目标：

- 说明音频和视频文件如何通过因特网下载以便将来使用或者向客户广播。因特网也可以用做直播音频和视频的交互。在通过因特网发送之前，音频和视频需要进行数字化处理。
- 讨论音频和视频文件如何压缩以便在因特网上传输。
- 讨论一种称为抖动的现象，这种现象出现在使用分组交换网络传送实时数据时。
- 介绍在多媒体应用中使用的实时运输协议（RTP）和实时运输控制协议（RTCP）。
- 讨论 IP 上的话音，它是一种实时交互式音频/视频的应用。
- 介绍会话发起协议（SIP），它是负责建立、管理和终止多媒體会话的应用层协议。
- 介绍服务质量（QoS）以及如何利用调度技术和流量整形技术对其改进。
- 讨论综合服务和区分服务以及它们是如何实现的。
- 介绍资源预留协议（RSVP），它是一种信令协议，可以帮助 IP 创建流并进行资源预留。

25.1 引言

我们可以把音频和视频服务划分为三大类：流式存储音频/视频（streaming stored audio/video）、流式直播音频/视频（streaming live audio/video）以及交互式音频/视频(interactive audio video)，如图 25.1 所示。“流式”（streaming）表示的是用户在下载文件开始后即可听（或看）。

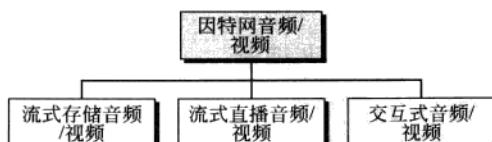


图 25.1 因特网音频/视频

第一大类，流式存储音频/视频，指文件被压缩后存储在服务器上。客户通过因特网下载这些文件。有时它也称为按需音频/视频（on-demand audio/video）。存储音频文件的例子包括歌曲、交响乐、有声读物以及著名演讲。存储视频文件的例子包括电影、电视节目以及音乐视频剪辑。

流式存储音频/视频指的是可按需请求到的压缩的音频/视频文件。

第二大类，流式直播音频/视频，指用户可以通过因特网收听的广播音频和视频。这种应用类型的一个很好的例子就是因特网电台。有的无线电台仅在因特网上广播它们的节目，也有些在因特网和空中同时广播它们的节目。因特网电视现在还不普及，但许多人相信在今后会有一些电视台将在因特网上广播它们的节目。

流式直播音频/视频指的是通过因特网广播的电台和电视节目。

第三大类，交互式音频/视频，指人们使用因特网与其他人进行交互式通信。这种应用的一个很好的例子就是因特网电话和电视会议。

交互式音频/视频指的是通过因特网进行交互式的音频/视频应用。

我们将在本章讨论这三种应用，但是首先我们需要讨论有关音频/视频的其他问题：对音频和视频的数字化处理，以及对音频和视频的压缩处理。

25.2 数字化音频和视频

在音频或视频信号能够通过因特网发送之前，必须先要将其数字化。我们分别讨论音频和视频两种情况。

25.2.1 数字化音频

当声音进入话筒时，电的模拟信号就产生了，它代表声音的振幅，是个时间的函数。这个信号称为模拟音频信号。像音频这样的模拟信号可以被数字化变成数字信号。根据 Nyquist 定理，如果信号的最高频率是 f ，我们就需要有每秒 $2f$ 次的采样。对音频信号的数字化还有其他一些方法，但在原理上都是一样的。

话音的采样速率是每秒 8000 个采样，每个采样为 8 位，结果得到 64 kbps 的数字信号。音乐的采样速率是每秒 44100 个采样，每个采样为 16 位，结果得到 705.6 kbps 的单声道数字信号和 1.411 Mbps 的立体声数字信号。

25.2.2 数字化视频

视频由帧的序列组成。如果这些帧在屏幕上显示得足够快，那么我们就得到运动的印象。原因在于我们的眼睛无法把非常快地闪过的这些帧区分成一个个单独的帧。每秒的帧数并没有一个标准，在北美通常是每秒 25 帧。但是为了避免一种称为闪烁的状态，每个帧

还需要被刷新。电视产业对每一帧刷新两次。这就表示每秒需要发送 50 帧，或者，如果发送方有存储器，可以每秒发送 25 帧，每一帧再从存储器刷新一次。

每个帧被划分为许多小方格，称为图像元素或像素 (pixel)。对于黑白电视，每个像素用 8 位表示，代表了 256 种不同灰度之一。对于彩色电视，每个像素是 24 位，每种原色（红、绿和蓝）需要一个 8 位。

对某个特定的分辨率，我们就可以计算出 1 秒钟需要多少位。在最低分辨率的情况下，一个彩色帧由 1024×768 个像素组成。这表示我们需要：

$$2 \times 25 \times 1024 \times 768 \times 24 = 944 \text{ Mbps}$$

这个数据率需要非常高速的技术，如 SONET。为了利用低速技术发送视频，我们需要对视频信号进行压缩。

在因特网上发送的视频需要压缩。

25.3 音频和视频压缩

在因特网上发送的音频和视频需要压缩 (compression)。在这一小节，我们首先讨论音频压缩，然后再讨论视频压缩。

25.3.1 音频压缩

音频压缩可被用于语音或音乐。对于语音，我们需要压缩 64 kHz 的数字化信号，而对于音乐，我们需要压缩 1.411 MHz 的信号。音频压缩有两类技术：预测编码和感知编码。

预测编码

使用预测编码 (predictive encoding)，就是把采样之差而不是把所有采样的值进行编码。这种类型的压缩通常是用于语音。为此已经定义了多种标准，如 GSM (13 kbps)、G.729 (8 kbps) 和 G.723.3 (6.4 或 5.3 kbps)。对这些技术的详细讨论超出了本书的范围。

感知编码：MP3

最常用的能产生 CD 质量的音频压缩技术是基于感知编码 (perceptual encoding) 的技术。如我们在以前所指出的，这种类型的音频至少需要 1.411 Mbps，这使得它如果不压缩就无法在因特网上传输。MP3 (MPEG 音频层 3) 是 MPEG 标准 (在视频压缩部分讨论) 的一部分，使用的就是这个技术。

感知编码的基础是音质科学 (science of psychoacoustics)，它研究人们如何感知声音。这种思想是基于我们听觉系统中的一些缺陷：某些声音可能掩蔽其他的声音。在频率和时间上都会发生掩蔽效应。对于频率掩蔽 (frequency masking)，一个频率范围中的强音可以部分地或全部地掩蔽在另一个频率范围中的弱音。例如，当房间里有声音很强的金属乐队演出时，我们就听不见舞伴说的话。对于时间掩蔽 (temporal masking)，一个很强的声音会在这个声音停止后的一个短时间内使我们的耳朵失去感觉。

MP3 利用了这两种现象 (频率掩蔽和时间掩蔽) 来压缩音频信号。这个技术对频谱进

行分析，并把它划分为若干组。对于完全被掩蔽的频率范围就不分配位。对部分被隐蔽的频率范围只分配少量的位。而大量的位被分配给未被掩蔽的频率范围。

MP3 产生三种数据率：96 kbps、128 kbps 和 160 kbps。这个数据率取决于原始模拟音频信号的频率范围。

25.3.2 视频压缩

我们在前面已经指出了视频信号是由多个帧组成的。每一帧就是一个图像。我们压缩视频信号时首先要压缩图像。在市场上有两个主流标准：**JPEG**（联合图像专家组）用来压缩图像。**MPEG**（活动图像专家组）用来压缩视频。我们将简单地讨论 JPEG，然后再讨论 MPEG。

图像压缩：JPEG

正如我们在前面讨论过的，如果图像不是彩色的（只有灰度），那么每一个像素可以用 8 位整数表示（256 个等级）。如果图像是彩色的，那么每一个像素可以用 24 位表示（ 3×8 位），其中每 8 位表示红、绿或蓝（RGB）之一。为了简化讨论，我们重点关注灰度的图像。

在 JPEG 中，一个灰度图像被划分为许多 8×8 像素的块（见图 25.2）。

把图像划分为许多块是为了减少运算量，因为稍后会看到，对每个图像来说，数学运算量是这些单元数目的平方。

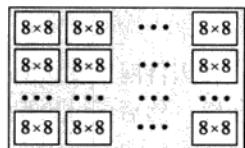


图 25.2 JPEG 灰度

JPEG 的整体思想是把图像转换成能够反映图像的冗余度的数字的线性（向量）集合。然后使用一种文本压缩方法就可以去除冗余度（即没有变化）。这个过程的一种简化的机制如图 25.3 所示。

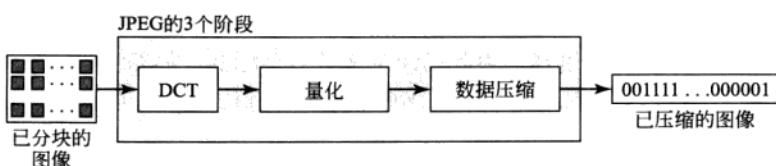


图 25.3 JPEG 过程

离散余弦变换 这一步骤是把每个 64 像素的块进行离散余弦变换（Discrete Cosine Transform, DCT）。这个变换改变了原始的 64 个数值，使之在保持了像素之间相对关系的同时，能够反映出冗余度。我们在这里不给出具体的公式，但是会给出三种情况下的转换结果。

情况 1 在这种情况下，我们有一块均匀的灰色，每个像素的值都是 20。当我们进行转换时，得到的第一个元素是非零值（左上角），而其他所有的像素的值则都是 0。 $T(0, 0)$ 就是所有 $P(x, y)$ 值的平均值（乘以一个常数），称为 dc 值（直流值，是电气工程使用的术语）。 $T(m, n)$ 中其余的值称为 ac 值，表示了像素值的变化。但由于这里并没有变化，因此其余的值都是 0（见图 25.4）。

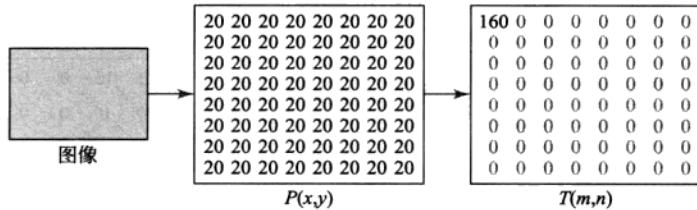


图 25.4 情况 1：均匀灰度

情况 2 在第二种情况下，我们有一块具有两个不同的均匀灰度的部分。这些像素的值有一个突变（从 20~50）。当我们进行转换后，就得到一个 dc 值和一些非零的 ac 值。但是，围绕 dc 值只有很少的几个非零值。大多数的值还是 0（见图 25.5）。

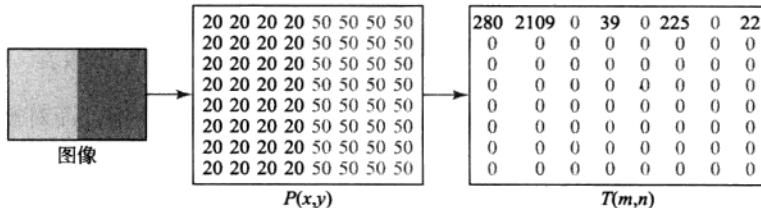


图 25.5 情况 2：两个部分

情况 3 在第三种情况下，我们有一块逐渐变化的块。也就是说相邻像素之间没有快速变化的值。当我们进行转换后，就得到一个 dc 值，还有一些非零的值（见图 25.6）。

综上所述，我们得出如下的结果：

- 把表 P 进行转换后生成表 T 。
- dc 值是像素的平均值（乘以一个常数）。
- ac 值表示了变化。
- 相邻像素之间如果没有变化就得到 0。

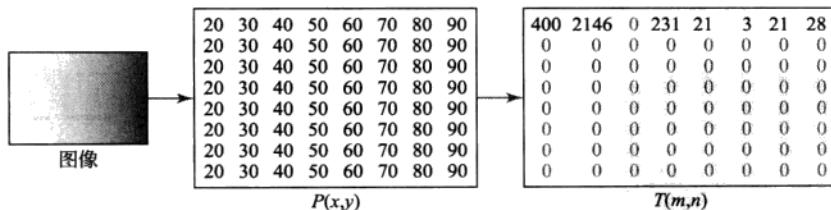


图 25.6 情况 3：渐变灰度

量化 在表 T 产生后，这些值还要经过量化，以便减少需要编码的位数。在量化 (quantization) 之前，我们先把每个值的小数部分丢掉，只保留整数部分。在这里，我们把这些值除以一个常数，然后丢弃小数部分。这就更进一步地减少了所需的位数。在大多数的实现中，用一个量化表 (8×8) 来定义如何量化每一个值。除数取决于 T 表中的值的位置。这样做是为了针对各种特定的应用来优化位数和 0 的数目。请注意，在整

个处理过程中，只有这个量化阶段是个不可逆的过程。在此处我们会损失一些信息，且不可恢复。其实，JPEG之所以称为有损压缩，就是因为有了这个量化阶段。

压缩 经过量化之后，表中的值被读出，冗余的0被丢弃。但是，为了把0聚合在一起，这个表并不是逐行或逐列地读，而是像“之”字形那样沿对角线方向读。原因就是如果图像是均匀地变化的，那么T表的右下角应该全部是0。图25.7描绘了这个过程。

视频压缩：MPEG

活动图像专家组（MPEG）的方法是用来压缩视频的。从原理上看，活动图像是一组快速流动的帧，而每个帧就是一个图像。换言之，一个帧是像素的空间组合，而视频是一个接一个地发送的帧的时间组合。因此，压缩视频就表示要对每个帧进行空间压缩，而对帧的集合要进行时间压缩。

空间压缩 每个帧的空间压缩（spatial compression）是使用JPEG（或修改过的）完成的。每个帧都是一个能够独立进行压缩的图像。

时间压缩 在时间压缩（temporal compression）时，冗余的帧被删除。当我们看电视时，每秒接收50个帧。但是，大多数连续的帧几乎都是相同的。例如，当某人在说话时，后一个帧的绝大部分都和前一帧一样，除了围绕嘴唇的那部分，只有它们在每个帧中是不同的。

为了在时间上压缩数据，MPEG方法首先把所有的帧划分为3大类：I帧、P帧和B帧。图25.8给出了帧序列的样本。

图25.9描绘了I帧、P帧和B帧是怎样从7个帧的序列中构成的。

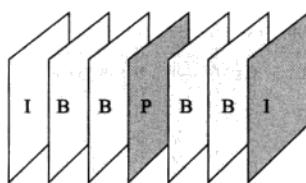


图25.8 MPEG帧

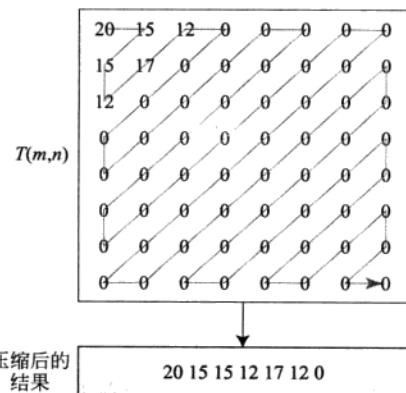


图25.7 读取表

压缩后的结果

20 15 15 12 17 12 0

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

- **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 帧包含的只是与前一个帧相比较的变化。但是，这种变化不能覆盖很大的一个区域。例如，对于快速移动的对象，新的变化可能无法被记录在一个 P 帧中。P 帧只能从前一个 I 帧或 P 帧中构造。P 帧携带的信息要比其他类型的帧少得多，而经过压缩后还要少一些。
 - **B 帧** 双向帧 (bidirectional frame, B 帧) 与前一个和后一个 I 帧或 P 帧有关。换言之，每一个 B 帧与过去和将来都有关。请注意，B 永远不会与另一个 B 帧有关。
- MPEG 经过了两个版本的发展。MPEG1 是设计为 CD-ROM 用的，数据率为 1.5 Mb/s，MPEG2 是设计为高质量 DVD 用的，数据率为 3~6 Mb/s。

25.4 流式存储音频/视频

我们已经讨论了音频/视频的数字化和压缩处理，现在就把注意力转向特定的应用。首先是流式存储音频/视频。从万维网服务器下载这类文件与下载其他类型的文件有所不同。要理解这个概念，让我们讨论四种方法，它们各有不同的复杂性。

25.4.1 第一种方法：使用万维网服务器

压缩的音频/视频文件可以像文本文件那样下载。客户（浏览器）可以使用 HTTP 的服务，发送 GET 报文来下载文件。万维网服务器向浏览器发送压缩的文件，然后浏览器就可以用一个辅助的应用程序来播放这个文件，通常这个应用程序称为媒体播放器 (media player)。图 25.10 描绘了这种方法。

这种方法非常简单，并没有涉及到流式。但是它有一个缺点。一个音频/视频文件即使是在压缩后通常也是很大的。一个音频文件通常有几十兆字节，而一个视频文件可能有几百兆字节。使用这种方法，文件需要先被完全下载后才能够播放。以目前的数据率来看，用户在播放这个文件之前需要等几秒钟或几十秒钟。

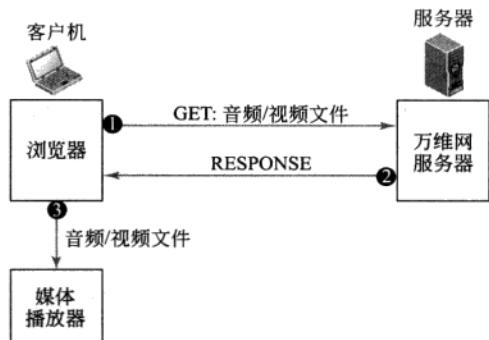


图 25.10 使用万维网服务器

25.4.2 第二种方法：使用具有元文件的万维网服务器

另一种方法是用媒体播放器直接连接到万维网服务器来下载音频/视频文件。万维网服务器存储了两个文件：真正的音频/视频文件和元文件 (metafile)，元文件保存有关于音频/视频文件的信息。图 25.11 描绘了这种方法的步骤。

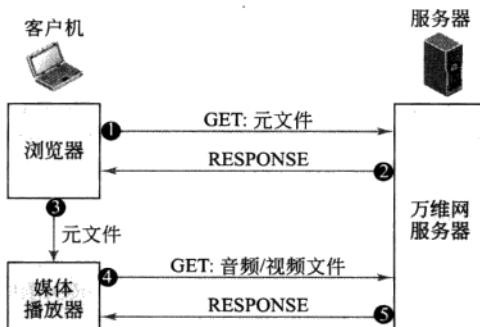


图 25.11 使用具有元文件的万维网服务器

1. HTTP 客户使用 GET 报文接入到万维网服务器。
2. 收到的响应中有关于元文件的信息。
3. 把元文件传递给媒体播放器。
4. 媒体播放器使用元文件中的 URL 访问这个音频/视频文件。
5. 万维网服务器给出响应。

25.4.3 第三种方法：使用媒体服务器

第三种方法的问题是浏览器和媒体播放器都要使用 HTTP 的服务。但 HTTP 是设计在 TCP 上运行的。它适合于读取元文件而不适合于读取音频/视频文件。原因是 TCP 要重传丢失的或受损伤的报文段，而这与流式的原理背道而驰。我们需要避开 TCP 及其差错控制。我们应当使用 UDP 而不是 TCP。但是，访问万维网服务器的 HTTP 和万维网服务器本身都是为使用 TCP 而设计的，因此我们需要另外的服务器，即媒体服务器（media server）。图 25.12 描绘了这个概念。

1. HTTP 客户使用 GET 报文接入到万维网服务器。

2. 收到的响应中有关于元文件的信息。

3. 把元文件传递给媒体播放器。

4. 媒体播放器使用元文件中的 URL 接入到媒体服务器，下载这个文件。这个下载可以利用任何使用 UDP 的协议。

5. 媒体服务器给出响应。

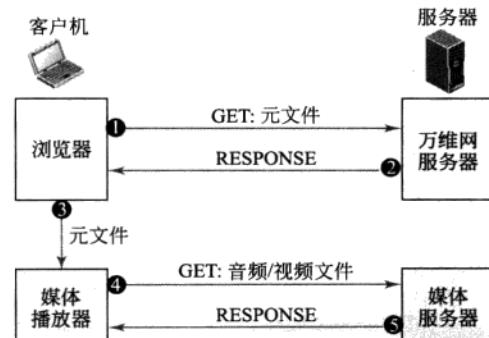


图 25.12 使用媒体服务器

25.4.4 第四种方法：使用媒体服务器和 RTSP

实时流式协议（Real-Time Streaming Protocol, RTSP）是为了给流式过程增加更多的功

能而设计的一种控制协议。使用 RTSP，我们可以控制音频/视频的播放。RTSP 是个带外控制协议，它和 FTP 的第二条连接相似。图 25.13 描绘了媒体服务器和 RTSP。

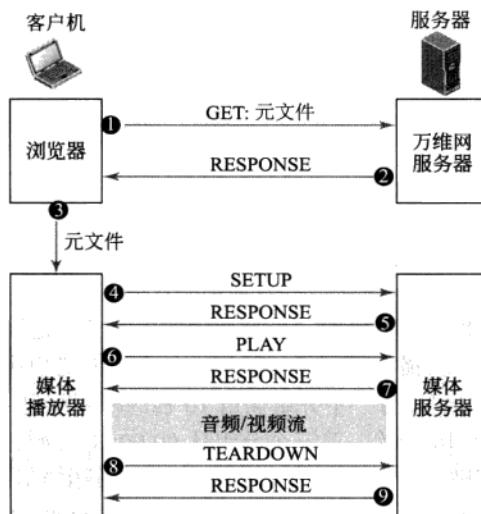


图 25.13 使用媒体服务器和 RTSP

1. HTTP 客户使用 GET 报文接入到万维网服务器。
2. 收到的响应中有关于元文件的信息。
3. 把元文件传递给媒体播放器。
4. 媒体播放器发送 SETUP 报文与媒体服务器建立连接。
5. 媒体服务器给出响应。
6. 媒体播放器发送 PLAY 报文开始播放（下载）。
7. 音频/视频文件被下载，使用的是在 UDP 上运行的其他协议。
8. 使用 TEARDOWN 报文断开连接。
9. 媒体服务器给出响应。

媒体播放器还可以发送其他类型的报文。例如，PAUSE 报文可暂时停止下载，PLAY 报文可继续下载。

25.5 流式直播音频/视频

流式直播音频/视频与无线电台和电视台的音频和视频广播是相似的。但不是从空中广播，而是通过因特网广播。流式直播音频/视频与流式存储音频/视频在许多方面是相似的。它们都对延迟敏感，都不能接受重传。但是有一点不同。流式存储音频/视频的通信是单播的和按需的。流式直播音频/视频的通信是多播和现场直播的。直播的流式更适合于 IP 多播服务和使用如 UDP 和 RTP 这样的协议（在后面讨论）。但是，目前流式直播仍然使用了 TCP 和多个单播而不是多播。在这个领域还有很多的研究正在进行。

25.6 实时交互式音频/视频

在实时交互式音频/视频中，人们相互之间实时地进行通信。因特网电话或 IP 上的话音通信就是这种应用类型的例子。电视会议是另一个例子，它使人们可以进行视觉上的和声音上的通信。

25.6.1 特性

在讨论用于这类应用的协议之前，我们先要讨论实时音频/视频通信的一些特点。

时间关系

分组交换网上的实时数据要求保留一个会话中的各个分组之间的时间关系。例如，我们假定实时视频服务器产生直播视频图像，并把它们在线发送出去。视频信号被数字化并形成分组。这里只有三个分组，每一个分组保留 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。我们还要设想，每一个分组要花 1 秒钟（为了简单，这里夸大了）到达终点（相同的延时）。接收方在 00:00:01 时重放第一个分组，第二个分组在 00:00:11，第三个分组在 00:00:21。虽然在发送方摄像人看到的和远程观看者在计算机屏幕上看到的两者之间有 1 秒钟的时差，但动作发生过程是实时的。各分组之间的时间关系被保留下。1 秒的延时并不重要。图 25.14 给出了这个概念。

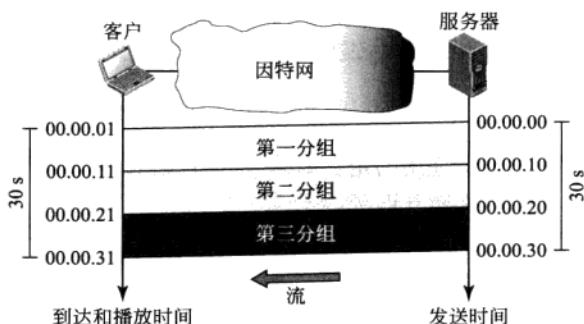


图 25.14 时间关系

但是如果分组以不同的延时到达，那么又会发生什么呢？例如，第一个分组到达时间是 00:00:01 (1 s 延时)，第二个分组到达时间是 00:00:15 (5 s 延时)，第三个分组到达时间是 00:00:27 (7 s 延时)。如果接收方在 00:00:01 开始播放第一个分组，那么它将在 00:00:11 结束。但是，第二个分组还没有到达，它要在 4 s 后到达。在远程观看这个视频节目时就会看到在第一个和第二个分组之间，以及在第二个和第三个分组之间存在着间隙。这个现象称为抖动 (jitter)。图 25.15 说明了这种情况。

由于分组之间的不同延时，在实时数据中产生了抖动。

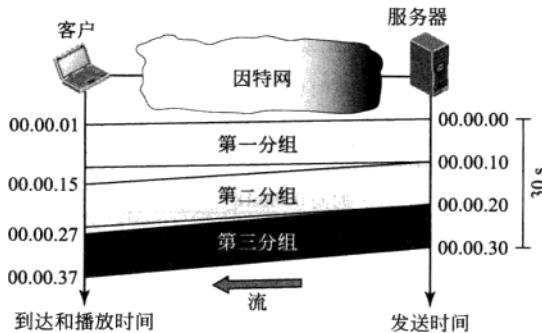


图 25.15 抖动

时间戳

解决抖动的一种方法是使用时间戳 (timestamp)。如果每个分组都有一个时间戳，并且这个时间戳给出了它相对于第一个（或前一个）分组的产生时间，那么接收方就可以在这个时间加到它开始重放的时间中去。换言之，接收方知道每个分组应当在何时播放。设想前面的例子中的第一个分组有一个时间戳 0，第二个分组的时间戳是 10，第三个分组的时间戳是 20。如果接收方开始播放第一个分组的时间是 00:00:08，那么第二个分组应在 00:00:18 播放，而第三个分组应在 00:00:28 播放。这些分组之间没有间隙。图 25.16 给出了这种情况。

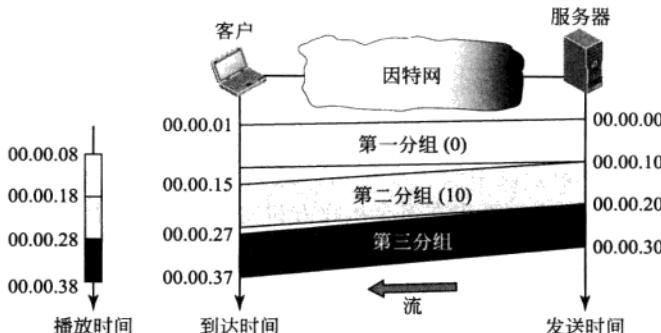


图 25.16 时间戳

重放缓存

为了能够把到达时间和重放时间分开，我们需要用缓存来存储数据，直到它们被重放为止。这个缓存称为重放缓存 (playback buffer)。当会话开始时（第一个分组的第一个位到达），接收方要推迟播放这个数据，直至到达一个门限值。在前面的例子中，第一个分组的第一个位的到达时间是 00:00:01，如果门限值是 7 秒，那么重放时间是 00:00:08。门限以数据的时间单位来测量。一直到数据的时间单位等于门限值时重放才能开始。

数据可能以变化速率存储在缓存中，但数据被提取出和被重放则是以固定速率进行。请注意，缓存中的数据量可能会缩小或扩大，但只要延迟小于重放门限值之内的数据所需的时间，就不会产生抖动。图 25.17 给出了在我们的例子中，不同时间的缓存状况。

为了防止抖动，我们可以给分组打上时间戳，并把到达时间和播放时间区分开来。

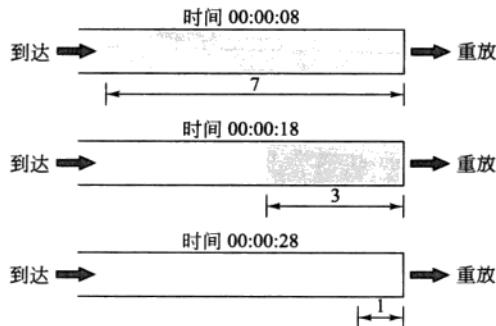


图 25.17 重放缓存

对于实时通信量，需要有重放缓存。

排序

对于实时通信，除了时间关系信息和时间戳以外，还需要有另外一种能力。对每个分组，我们都需要有一个序号。仅靠时间戳还不能够通知接收方是否某个分组丢失了。例如，假定时间戳分别是 0、10 和 20。如果第二个分组丢失了，接收方只收到时间戳为 0 和 20 的两个分组。接收方会假定时间戳为 20 的分组是第二个分组，是在第一个分组之后 20 秒产生的分组。接收方无法知道第二个分组实际上已经丢失了。要处理这种情况，需要用序号对分组进行排序。

对于实时通信量，需要给每个分组加上序号。

多播

多媒体在音频和视频会议中起主要作用。通信量可能会很大，且数据要使用多播 (multicasting) 方式发布。会议要求在多个接收方和发送方之间进行双向通信。

实时通信量需要多播的支持。

转换

有时，实时通信量还需要转换 (translation)。转换器是一个可以把高带宽的视频信号转换为低质量的窄带宽信号的计算机。例如，某个信号源产生了 5 Mbps 的高质量视频信号，要将其发送给带宽小于 1 Mbps 的接收者，就需要用到转换。为了接收这样的信号，转换器需要先将信号解码，然后再按照需要带宽小的较低质量进行编码。

转换意味着改变有效载荷的编码，使它具有较低质量，这样就可以和接收网络的带宽相匹配。

混合

如果在同一时间可以发送数据的源点超过一个（例如，在开视频或音频会议时），那么通信量就是由多个流组成。为了把通信量合并到一个流，从不同源点来的数据可以进行混合。混合器（mixer）从数学上把来自不同源点的信号相加，以产生一个单信号。

混合的意思就是把几个通信量的流合并成一个流。

来自运输层协议的支持

前面提到的这些过程都可以在应用层实现。然而，这些过程在各种实时应用中都是大同小异的，因此更好的办法是通过运输层协议来实现。让我们看一下，哪一个现有的运输层协议适合于这种类型的通信。

TCP 不适合于交互式通信。它不提供时间戳，也不支持多播。但是，它提供了排序（序号）。TCP 特别不适合于实时通信的一个特点就是它的差错控制机制。对于交互式的通信量，我们不能允许重传一个丢失的或受损伤的分组。如果一个分组丢失了或受到损伤，那么把它忽略就行了。重传使得时间戳和重放的概念遭到破坏。今天，在音频和视频信号中有很多的冗余（哪怕使用了压缩），因此我们只要忽略丢失的分组就行了。远程的听众或观众甚至可能不会注意到这一点。

TCP 因其复杂性而不适合于交互式多媒体通信量，尤其我们不能容许分组的重传。

UDP 更加适合于交互式的多媒体通信量。UDP 支持多播，并且没有重传策略。但是，UDP 没有提供时间戳、排序和混合功能。一个新的运输层协议，即实时运输协议（RTP），可以提供这些缺少了的功能。

UDP 比 TCP 更加适合于交互式通信量。但是，我们需要的是另一个运输层协议 RTP 所提供的服务，它弥补了 UDP 的一些不足。

25.7 RTP

实时运输协议（Real-time Transport Protocol, RTP）是设计用来处理因特网上的实时通信量的协议。RTP 没有交付机制（多播、端口号，等等），它必须和 UDP 一起使用。RTP 的位置在 UDP 和应用程序之间。RTP 的主要贡献是：时间戳功能、排序功能以及混合功能。图 25.18 表示 RTP 在协议族中的位置。

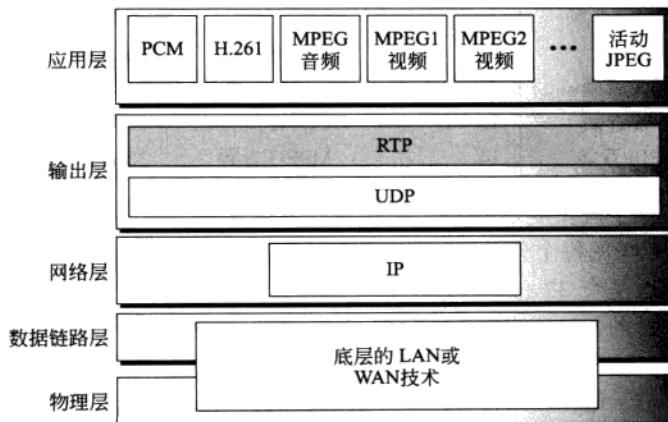


图 25.18 RTP

25.7.1 RTP 分组格式

图 25.19 描绘了 RTP 分组首部的格式。这个格式非常简单，并且其通用性足以涵盖所有的实时应用。需要更多信息的应用可以在它有效载荷的前端加上这些信息。各字段的描述如下：

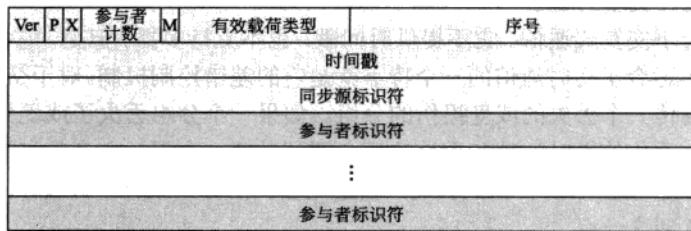


图 25.19 RTP 分组首部的格式

- Ver (版本)** 这个 2 位字段定义了版本号。当前的版本是 2。
- P (填充)** 这个 1 位字段在置 1 时表示在分组结束处有填充。在这种情况下，填充的最后一个字节的值定义填充长度。如果分组是加密的，那么填充是规范的。如果 P 字段值是 0 就表示没有填充。
- X (额外)** 这是个 1 位字段，若置为 1，就表示在基本首部和数据之间还有额外的扩展首部。若这个字段是 0，就表示没有额外的扩展首部。
- 参与者计数** 这个 4 位字段指出参与者的数目。应当注意，我们最多有 15 个参与者，因为 4 位字段只能表示 0 到 15 之间的数。
- M (标记)** 这个 1 位字段是个标记。例如，应用程序用它指出数据的结束。
- 有效载荷类型** 这个 7 位字段指出有效载荷的类型。到目前为止已经定义出几种有效载荷类型。在表 25.1 中我们列出了一些常见的应用。这些类型的讨论已超过本书的范围。

表 25.1 有效载荷的类型

类型	应用	类型	应用	类型	应用
0	PCM μ 音频	7	LPC 音频	15	G728 音频
1	1016	8	PCMA 音频	26	活动 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	DV14 音频	14	MPEG 音频	33	MPEG2 视频

- 序号** 这个字段长度是 16 位。它用来给 RTP 分组编号。第一个分组的编号是随机选择的，后续的每个分组序号加 1。接收方使用这个序号来检测分组的丢失或失序。
- 时间戳** 这是个 32 位字段，用来指出分组之间的时间关系。第一个分组的时间戳是随机选择的。对后续的每个分组，时间戳的值是前一个时间戳加上第一个字节产生（采样）的时间。时钟滴答值与应用有关。例如，音频应用通常产生 160 字节的数据段，对于这种应用，时钟滴答值就是 160，因此对于它的每个 RTP 分组，时间戳就增加 160。

- **同步源标识符** 如果只有一个源点，这 32 位字段就定义这个源点。但如果有好几个源点，那么混合器是同步源点，而其他源点是参与者。源点标识符的值是由源点选择的一个随机数。在发生冲突时（两个源点从同一个序号开始），这个协议提供了解决的策略。
- **参与者标识符** 这些 32 位标识符中的每一个都定义了一个源点（最多 15 个）。当一次会话中有超过一个源点时，混合器是同步源点，而其余的源点是参与者。

25.7.2 UDP 端口

虽然 RTP 本身是一个运输层协议，但 RTP 分组并不直接封装在 IP 数据报中。实际上，RTP 被看成是应用程序，它的分组被封装在 UDP 用户数据报中。但是，与其他应用程序不同，没有熟知端口指派给 RTP。端口号是按需选取的，唯一的限制是：端口号必须是偶数。下一个数（奇数）由实时运输控制协议（RTCP）使用，RTCP 是和 RTP 一同成对使用的协议。

RTP 使用临时的偶数 UDP 端口。

25.8 RTCP

RTP 只允许一种类型的报文，即从源点到终点携带数据的报文。在许多情况下，会话还需要使用其他的报文。这些报文的作用是控制流和数据的质量，并允许接收者把反馈发送给源点或几个源点。实时运输控制协议（Real-time Transport Control Protocol, RTCP）就是为了这个目的而设计的。RTCP 有五种类型的报文，如图 25.20 所示。在每一个方框右边的数字定义报文的类型。



图 25.20 RTCP 的报文类型

25.8.1 发送方报告

在会议中，发送方报告（sender report）从活动的发送方周期性地发出，它报告在这段间隔中发送的所有 RTP 分组的传输和接收统计。发送方报告包括了一个绝对时间戳，这是从 1970 年 1 月 1 日午夜起经过的秒数。绝对时间戳使接收方可以和不同的 RTP 报文同步。在同时发送音频和视频时（音频和视频使用分开的相对时间戳）时，绝对时间戳就特别有用。

25.8.2 接收方报告

接收方报告（receiver report）是为不发送 RTP 分组的被动参加者使用的。这个报告通知发送方和其他接收方有关服务质量的状况。

25.8.3 源点描述报文

源点周期性地发送源点描述报文（source description message），以便给出有关自己的附加信息。这个信息可以是姓名、电子邮件地址、电话号码以及拥有者地址或源点控制者地址。

25.8.4 再见报文

源点通过发送再见报文（bye message）来关闭一个流。再见报文允许这个源点宣布它将要退出这个会议。虽然其他源点能够检测到某个源点的缺席，但这个报文是直接宣布的。它在混合器中非常有用。

25.8.5 特定应用报文

特定应用报文（application-specific message）是为想要使用新的应用（没有在标准中定义的）的应用而设计的。它允许定义新的报文类型。

25.8.6 UDP 端口

RTCP 像 RTP 那样，不使用熟知 UDP 端口，而是使用临时端口。它所选择的 UDP 端口必须紧接着 RTP 选择的 UDP 端口号，因而使得这个端口号必定是奇数。

RTCP 使用奇数 UDP 端口号，这个端口号紧接着 RTP 所选择的端口号。

25.9 IP 话音

让我们重点讨论一种实时交互式音频/视频应用：**IP 话音**（voice over IP）或因特网电话。它的思想是使用因特网作为具有一些附加能力的电话网络。这个应用不是在电路交换网上实现通话，而是允许双方在分组交换网上进行通信。已经设计了两个协议来处理这种类型的通信：SIP 和 H.323。我们将简单地讨论这两个协议。

25.9.1 SIP

会话发起协议（Session Initiation Protocol, SIP）是 IETF 设计的。它是一个应用层协

议，用来建立、管理和终止一个多媒体会话（呼叫）。它可以用产生两方、多方或多播的会话。SIP 被设计为不依赖下面的运输层；它可以在 UDP、TCP 或 SCTP 上运行。

报文

SIP 是像 HTTP 一样的基于文本的协议。SIP 和 HTTP 一样使用了报文。SIP 共定义了六种报文，如图 25.21 所示。



图 25.21 SIP 报文

每一种报文都有首部和主体。首部由若干行组成，描述了报文的结构、呼叫方的能力、媒体类型等等。我们给出每一种报文的简单描述。然后给出它们在一个简单的会话中的应用。

呼叫方用 INVITE 报文发起会话。在被叫方回答这个呼叫后，呼叫方发送 ACK 报文进行确认。BYE 报文用来终止会话。OPTIONS 报文向一个机器查询其能力。CANCEL 报文取消一个已经开始的初始化进程。当打不通被叫方时，REGISTER 报文用来进行连接。

地址

在常规的电话通信中，用一个电话号码标识发送者，另一个电话号码标识接收者。SIP 则十分灵活。在使用 SIP 时，电子邮件地址、IP 地址、电话号码或其他类型的地址，都可用来标识发送者和接收者。但是，这个地址必须使用 SIP 格式（也称为模式）。图 25.22 给出一些常用的格式。

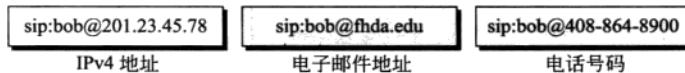


图 25.22 SIP 的格式

简单的会话

使用了 SIP 的简单会话包括三个模块：建立、通信和终止。图 25.23 描绘了使用 SIP 的简单会话。

建立会话 SIP 建立会话需要三向握手。主叫方在开始通信时使用 UDP、TCP 或 SCTP 发送 INVITE 报文。如果被叫方愿意开始会话，就发送一个回答报文。为了确认回答代码已经收到，主叫方再发送一个 ACK 报文。

通信 会话建立后，主叫方和被叫方就可以用两个临时端口进行通信。

终止 任何一方都可以通过发送 BYE 报文终止会话。

跟踪被叫方

如果被叫方没有坐在自己的终端旁边会发生什么呢？她可能远离自己的系统或者在另一个终端上。如果她使用了 DHCP，那么甚至没有一个固定的 IP 地址。SIP 有一种机制（与 DNS 相似），可以找出被叫方所在的那个终端的 IP 地址。为了实现这样的跟踪，SIP 使用注册的概念。SIP 定义了一些服务器作为注册机构。在任何时刻，用户至少要向一个注册机构服务器（registrar server）进行注册，于是这个服务器就知道了被叫方的 IP 地址。

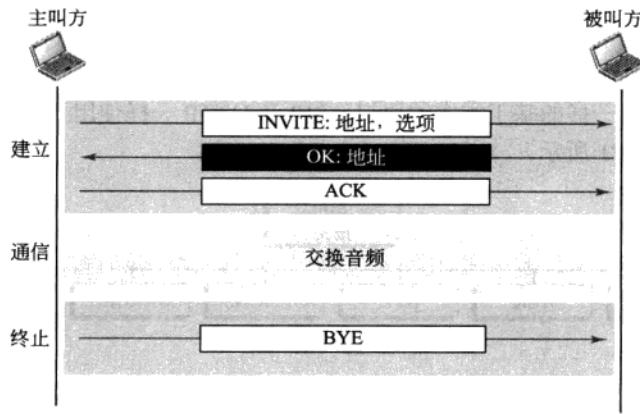


图 25.23 SIP 的简单会话

当主叫方需要和该被叫方通信时，主叫方在其 INVITE 报文中可以利用电子邮件地址代替 IP 地址。这个报文先到达一个代理服务器上。代理服务器向某个注册机构服务器（被叫方就是在这里注册的）发送一个查找报文（这个报文不属于 SIP）。当代理服务器从注册机构服务器收到回答报文时，它就取出主叫方的 INVITE 报文，插入刚才发现的被叫方的 IP 地址。然后这个报文被发送给被叫方。图 25.24 描绘了这个过程。

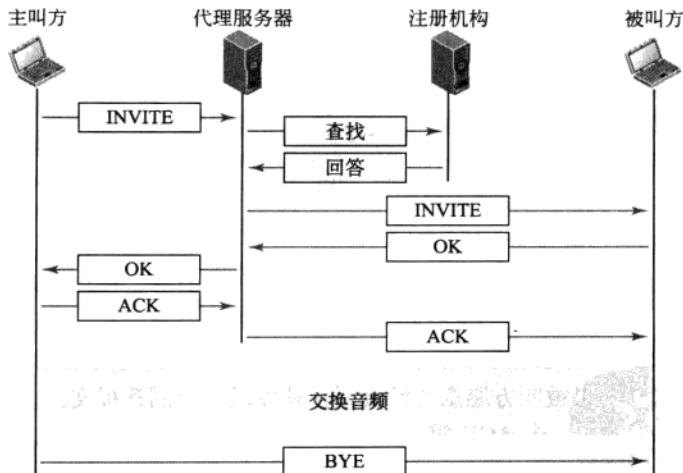


图 25.24 跟踪被叫方

25.9.2 H.323

H.323 是 ITU 设计的标准，它使得在公用电话网上的电话有可能和连接在因特网上的计算机（称为 H.323 终端）通话。图 25.25 给出了 H.323 的一般体系结构。

网关（gateway）把因特网和电话网连接起来。一般来说，网关是一个具有五层协议的

设备，它把一个报文从一种协议栈转换到另一种。这里的网关要做的正是这件事。它把电话网中的报文转换为因特网的报文。如我们在 SIP 协议中讨论过的，在局域网上的网关（gatekeeper）服务器起到注册机构服务器的作用。

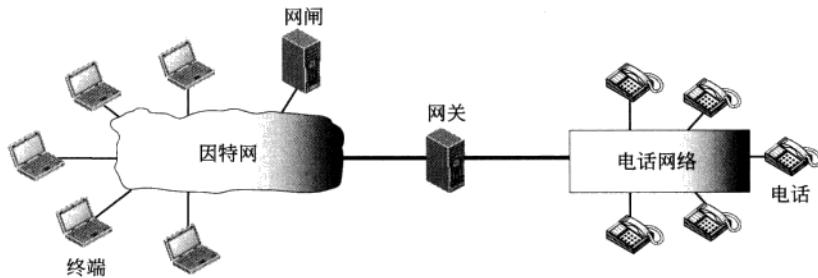


图 25.25 H.323 体系结构

协议

H.323 使用了多个协议来建立和维持语音（或视频）的通信。图 25.26 给出了这些协议。

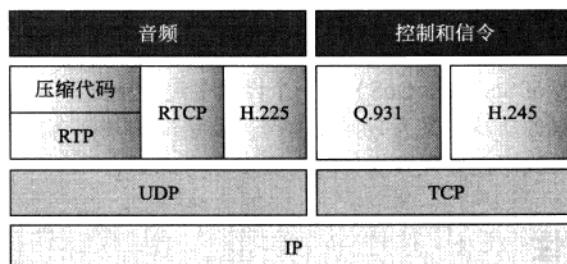


图 25.26 H.323 的协议

H.323 使用 G.71 或 G.723.1 进行压缩。它使用一个称为 H.245 的协议，以允许双方协商压缩方法。协议 Q.931 用来建立和终止连接。另一个协议称为 H.225 或 RAS（Registration/Administration/Status），被用来进行网关注册。

操作

让我们用一个简单的例子来说明使用 H.323 的电话通信操作过程。图 25.27 描绘了使用一个终端和一个电话进行通信的步骤。

1. 终端向网闸发送广播报文。网闸以其 IP 地址响应。
2. 终端和网闸使用 H.225 进行通信，协商带宽。
3. 终端、网闸、网关和电话使用 Q.931 进行通信，建立连接。
4. 终端、网闸、网关和电话使用 H.245 进行通信，协商压缩方法。
5. 终端、网关和电话使用 RTP，并在 RTCP 的管理下交换音频。
6. 终端、网闸、网关和电话使用 Q.931 终止此次通信。

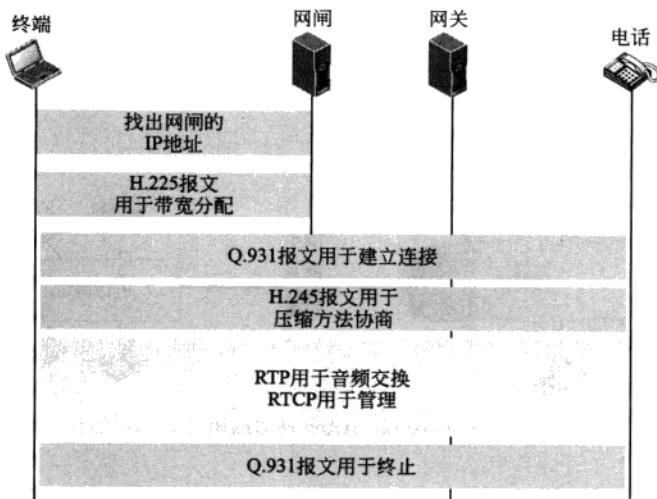


图 25.27 H.323 举例

25.10 服务质量

服务质量 (Quality of Service, QoS) 是一个讨论多过于定义的与网际互连相关的内容。我们可以非正式地把服务质量定义为数据流所努力要达成的某样东西。尽管 QoS 可应用于文本数据和多媒体，但我们在处理多媒体时会更多地考虑到它。

25.10.1 流的特性

传统上，人们给流赋予了四种特性：可靠性、延迟、抖动和带宽，如图 25.28 所示。



图 25.28 流的特征

可靠性

可靠性 (reliability) 是流所需要的一种特性。缺乏可靠性意味着分组或确认的丢失，随之带来的结果就是重发。不过不同的应用程序对可靠性的要求也不同。例如，与电话或者语音会议相比较而言，电子邮件、文件传送和因特网接入对传输的可靠性的要求更高。

延迟

从源到终点的延迟 (delay) 是流的另一种特性。类似地，不同的应用程序能够容忍不同程度的延迟。从这个角度来说，电话、音频会议、视频会议和远程登录要求最小的延迟，

而对文件传送或者电子邮件来说，延迟则不那么重要。

抖动

抖动 (jitter) 是指属于同一个流的分组延迟的变化量。例如，如果 4 个分组在时间点 0、1、2、3 出发，并在时间点 20、21、22、23 抵达，它们都有相同的延迟（20 个时间单位）。另一方面，如果上述四个分组分别在时间点 21、23、21、28 抵达，那么它们就有不同的延时：21、22、19 和 24。

对于音频和视频应用程序而言，上述第一种情况是完全可以接受的，而第二种情况却不可以。对这些应用程序来说，只要所有分组的延迟都相同，分组抵达时的延迟长短就无关紧要。对这种应用来说，第二种情况是不能接受的。

抖动被定义为分组延迟的变化量。高抖动意味着延迟之间的差异很大，低抖动则说明延迟之间的变化很小。

带宽

不同的应用程序需要不同的带宽。在视频会议中我们需要每秒发送数百万比特来刷新彩色的屏幕，而一封电子邮件中的全部比特数之和可能都不到一百万。

25.10.2 流的分类

根据流的特性，我们可以把流划分为不同的组，每个组具有相似级别的特性。这种分类并非正式的和通用的。一些协议，如 ATM，已经定义了这些分类。

25.10.3 提高 QoS 的技术

在前一小节，我们试着从流的特性的角度定义了 QoS。在本小节，我们将讨论能够被用来提高服务质量的一些技术。我们简要地讨论四种常用的方法：调度、流量整形、许可控制、资源预留。

调度

来自不同流的分组抵达交换机或路由器后等待处理。好的调度技术能以公平适当的方式处理不同的流。有一些调度技术是为了提高服务质量而设计的。我们在这里讨论其中三种：先进先出排队、优先级排队和加权公平排队。

先进先出排队 使用先进先出 (First-In First-Out, FIFO) 排队时，分组先在一个缓冲池（队列）中等待，直到结点（交换机或路由器）准备好处理它们。如果平均抵达速率高于平均处理速率，队列将被填满而新的分组将被丢弃。FIFO 队列和人们在公交车站等公共汽车的情况很相像。图 25.29 给出了一个 FIFO 队列的概念图。

优先级排队 使用优先级排队 (priority queuing) 时，分组首先被赋予一个优先级。每个优先级有其自己的队列。最高优先级中的分组首先被处理。最低优先级中的分组最后处理。请注意，系统会一直处理某个队列，直至该队列为空。图 25.30 给出了有两个优先级的优先级排队（为简单起见）。

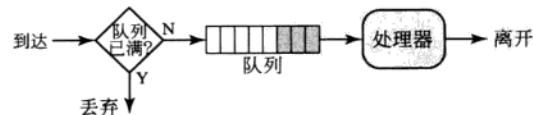


图 25.29 FIFO 队列

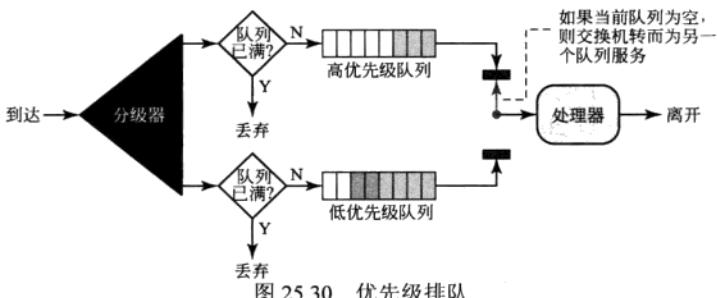


图 25.30 优先级排队

优先级队列能提供比 FIFO 队列更好的 QoS，因为有较高优先级的流量，如多媒体，能够以较小的延迟抵达目的地。不过它也有一个潜在的缺点。如果在高优先级队列中有连续不断的流，低优先级队列中的分组可能永远没有机会被处理，这种情况称之为饿死。

加权公平排队 一种更好的调度技术是加权公平排队 (weighted fair queuing)。使用这种技术时，分组依然要被赋予不同的级别，并被分配到不同的队列。不过，这些队列基于自己的优先级设置一个权重。高优先级意味着权重也高。系统以循环的方式处理每个队列中的分组，处理的分组数量是以每个队列的权重为基础的。例如，如果队列的权重分别为 3、2、1，那么在第一个队列中将处理三个分组，第二个队列中将处理两个分组，而第三个队列中将只处理一个分组。如果系统不实施优先级分类，那么所有的权重都相同。通过这种方式，我们就能得到具有优先级的公平排队。图 25.31 描绘了使用三个优先级分类的加权公平队列。

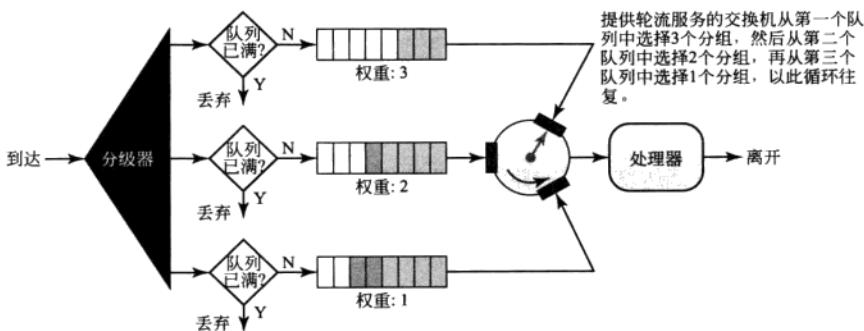


图 25.31 加权公平排队

流量整形

流量整形 (traffic shaping) 是一种控制发送到网络中的流量大小和速率的机制。两种技术可以对流量整形：漏桶算法和权标桶算法。

漏桶算法 如果水桶的底部有一个小洞，只要桶中有水，水便会以恒定的速率从洞中漏出。水从洞中漏出的速率并不取决于向桶中注水的速率，除非桶中没有水。输入的速率可以是变化的，但是输出的速率却保持恒定。类似地，在互联网络中，有一种被称为漏桶算法 (leaky bucket) 的技术可以使突发通信量变得平滑。突发的数据块被存储在桶中并以

均匀的速率送出。图 25.32 描绘了漏桶算法及其效果。

在图中，我们假设网络保证主机能够拥有 3 Mbps 的带宽。通过使用漏桶算法对输入通信量进行整形就可以实现这个保证。在图 25.32 中，主机以 12 Mbps 的速率发送突发数据，并持续 2 秒钟，所以总共有 24 Mb 的数据。接着这个主机静默了 5 秒，接着再以 2 Mbps 的速率发送数据，并持续 3 秒，总共是 6 Mb 数据。从整个过程看，主机在 10 秒的时间里发出了 30 Mb 的数据。漏桶算法可以平滑通信量，使得在同样的 10 秒之内以 3 Mbps 的速率把这些数据发送出去。如果不使用漏桶算法，刚开始的突发通信量可能会消耗超过预留给主机的带宽，从而影响到整个网络。我们还可以看到漏桶也能用于预防网络拥塞。打个比方，想象一下高峰时期的高速公路（突发流量）。如果上班的人群可以错开他们的工作时间，高速公路上的堵塞现象也就可以避免。

图 25.33 给出了一个简单的漏桶算法的实现。我们使用一个 FIFO 队列来暂时保存分组。如果这些通信量是由固定尺寸的分组构成（例如，ATM 网络中的信元），那么处理时可以在每个时钟周期从队列中取走固定数量的分组。如果这些通信量是由可变长度的分组构成的，那么固定的输出率必须基于字节或者比特的数量。



图 25.32 漏桶算法

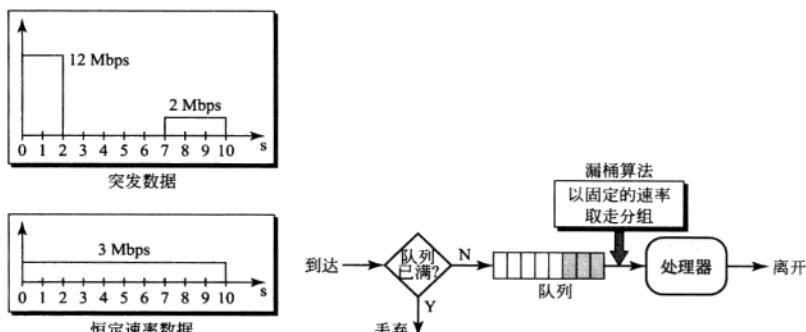


图 25.33 漏桶算法的实现

下面我们给出用于可变长度分组的一种算法：

1. 在时钟周期开始时将一个计数器初始化为 n 。
2. 如果 n 大于分组的长度，就发送该分组，并在计数器上减去该分组的长度。重复本步骤直到 n 小于分组尺寸。
3. 重置计数器并回到步骤 1。

漏桶算法 通过平均数据率的方法将突发的通信量整形为固定速率的通信量。如果漏桶已满，则有可能丢弃分组。

权标桶算法 漏桶算法不够灵活。它没有为空闲的主机加分。例如，如果某主机在一段时间内没有发送数据，它的漏桶会变空。如果这个主机突然又有了突发数据，漏桶算法只允许它以平均速率发送数据。主机的空闲时间并没有被考虑在算法之内。与此相反，权标桶算法（token bucket）则允许空闲的主机以权标的形式累积积分以供将来使用。在每个时钟周期，系统向权标桶发放 n 个权标。系统在每发送一个单元（或字节）时去掉一个权

标。例如，如果 n 等于 100 且主机空闲了 100 个时钟周期，那么桶中就会有 10000 个权标。现在主机可以在一个时钟周期发送 10000 个单元，这样会消耗掉所有权标。这个主机也可以用 1000 个时钟周期，以每个时钟周期发送 10 个单元的速度消耗掉所有的权标。换言之，只要桶中还有权标主机就可以发送突发数据。图 25.34 描绘了这个概念。

权标桶算法通过一个计数器就可以轻松实现。权标数量在开始时为 0。每次权标增加，计数器也增加 1。每当一个单位的数据被发送出去时，计数器就减 1。当计数器变为 0 时，主机就不能发送数据。

权标桶使突发通信量能够以可调节的最大速率发送。

结合权标桶和漏桶 这两种技术可以结合起来使用，以便在调节通信量的同时能够为空闲主机加分。我们可以在应用权标桶算法之后再应用漏桶算法，而漏桶的速率需要比权标桶中权标减少的速率高。

25.10.4 资源预留

一个数据流需要很多资源，例如缓存，带宽，CPU 时间等等。如果这些资源可以在事先预留，那么服务质量就可以提高。我们将在 25.11 小节中讨论一种称为综合服务的 QoS 模型，这个模型在很大程度上要依赖资源预留来提高服务质量。

25.10.5 许可控制

许可控制指的是路由器或交换机使用的一种机制，它根据提前给定的参数（被称为流规范）来接受或拒绝一个流。在路由器接受一个流并对其进行处理之前，先要检查流规范以判断根据自己的性能（用带宽，缓存大小，CPU 速度等来表示）及其自己对之前其他流所作出的保证，是否能够处理这个新的流。

25.11 综合服务

根据 25.10 小节所讨论的内容，人们已设计了两种模型来提供因特网的服务质量：综合服务和区分服务。这两个模型都强调了网络层（IP）的服务质量的使用，虽然它们也能用于其他层（如数据链路层）。我们在本小节讨论综合服务，并在 25.12 小节讨论区分服务。

我们在第 7 章已经知道，IP 最初被设计为“尽最大努力”的交付。这意味着每个用户都会收到相同级别的服务。这种类型的交付不能保证对诸如实时音频和视频这样的应用程

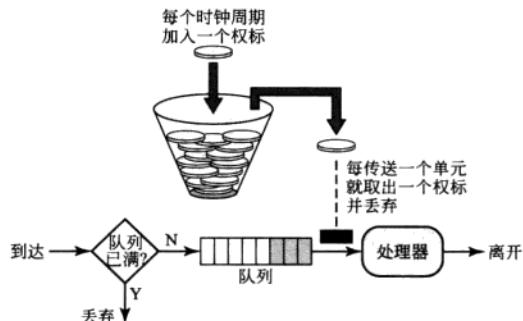


图 25.34 权标桶算法

序提供最低程度的服务，例如带宽。如果此类应用程序意外地得到一些额外的带宽，它很可能会影响其他的应用程序，从而导致网络堵塞。

综合服务（Integrated Services）是一种基于流的 QoS 模型，有时也称为 **IntServ**，它意味着用户需要创建从源点到终点的流（类似虚电路），并将它的资源需求通知所有的路由器。

综合服务是一种为 IP 而设计的基于流的 QoS 模型。

25.11.1 信令

读者可能还记得 IP 是一个无连接的，使用数据报的，分组交换式的协议。我们怎样才能在无连接的协议上实现一个基于流的模型呢？答案是一个运行在 IP 上的信令协议，它提供了信令机制来实现资源预留。这个协议就称为资源预留协议（Resource Reservation Protocol，RSVP）。我们接下来很快会讨论到这个协议。

25.11.2 流规范

当一个源在进行资源预留时，它需要定义流规范。流规范包括两个部分：Rspec（资源规范）和 Tspec（通信量规范）。Rspec 定义了需要为流预留的资源（缓存、带宽等等）。Tspec 定义了流的通信量特性。

25.11.3 许可

在路由器接收到来自应用程序的流规范之后，它要决定允许还是拒绝服务。这个决定基于路由器之前提供的保证以及当前可用的资源。

25.11.4 服务类别

综合服务定义了两个类别的服务：保证服务和控制负载服务。

保证服务类别

这种类型的服务是为需要保证最小的端到端延迟的实时通信量而设计的。端到端延迟是所有的路由器的延迟，加上媒体传播延迟以及连接建立机制所需时间的总和。其中只有第一项，路由器的时延是可以被路由器保证的。这类服务保证分组在特定的交付时间内抵达，并且只要流通信量在 Tspec 范围内就不会被丢弃。我们可以说保证的服务是可量化的服务，其中端到端的延迟量及其数据率必须由应用程序指明。

控制负载服务类别

这种类型的服务是为那些可以接受一些延迟，但对超负荷的网络和分组丢失都十分敏感的应用程序而设计的。这种应用程序的典型例子是文件传输、电子邮件和因特网接入。控制负载服务是不可量化类型的服务，其中应用程序请求的是低损失或无损失的可能性。

25.11.5 RSVP

在综合服务模型中，应用程序需要资源预留。我们在讨论综合服务模型时知道，资源预留是针对流的。这意味着如果我们想在 IP 层使用综合服务，就需要在最初被设计为使用数据报的分组交换网的 IP 之上创建一个流，也就是类似虚电路的网络。在数据通信量启动之前，虚电路网络首先需要一个信令系统来建立虚电路。资源预留协议（RSVP）就是这样的信令协议，它被用来帮助 IP 创建流并在之后进行资源预留。在讨论资源预留协议之前，我们需要指出它是从综合服务模型中分离出来的一个独立的协议。今后它可能会被用在其他模型中。

多播树

RSVP 与我们之前看到的其他信令系统的不同之处就在于它是为多播设计的。不过 RSVP 也可以在单播中使用，因为单播就是多播的一种特殊情况（当多播群中只有一个成员时）。这样设计的原因是为了使这个协议能够为所有的通信量类型，包括经常使用要多播的多媒体通信，提供资源预留。

基于接收方的预留

在 RSVP 中，是由接收方而非发送方来进行资源预留的。这种策略与其他的一些多播协议相同。例如，在多播路由协议中，是由接收方而非发送方决定加入还是离开多播群。

RSVP 报文

RSVP 有多种类型的报文。不过，根据我们的需要，这里只讨论其中的两种：**Path**（路径报文）和 **Resv**（预留报文）。

Path 报文 回想一下，在 RSVP 中是由流的接收方来进行资源预留的，但是接收方在预留完成之前并不知道分组传递经过的路径，而预留需要用到路径。为了解决这个问题，RSVP 使用 Path 报文。Path 报文从发送方出发，并抵达多播路径上的所有接收方。一路上，Path 报文保存那些对接收方来说是必要的信息。一个 Path 报文被发送到多播环境中，在路径分叉时，就会创建一个新的报文。图 25.35 所示为 Path 报文。

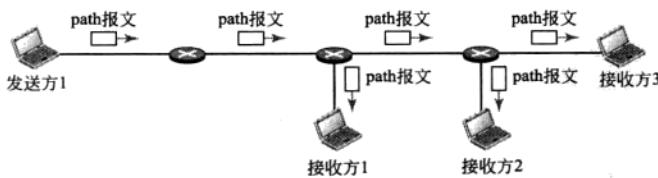


图 25.35 Path 报文

Resv 报文 在接收方收到 Path 报文之后，它会发送一个 Resv 报文。Resv 报文向着发送方传送（上行），并且在支持 RSVP 的路由器上进行资源预留。如路径上的某个路由器不能支持 RSVP，它会如我们之前讨论的那样，基于尽最大努力交付的方法转发分组。图 25.36 所示为 Resv 报文。

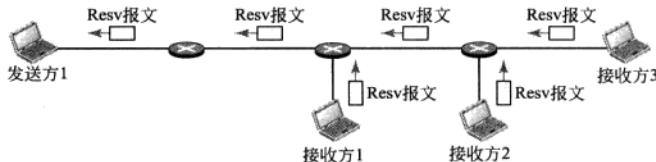


图 25.36 Resv 报文

预留合并

在 RSVP 中，资源并不是为一个流的每个接收方单独预留的，这种预留是合并的。在图 25.37 中，Rc3 请求一个 2 Mbps 的带宽，而 Rc2 请求一个 1 Mbps 的带宽。需要进行带宽预留的路由器 R3，就合并了这两个请求。此时预留的带宽是 2 Mbps，即两个请求中较大的一个，因为 2 Mbps 的输入预留可以同时满足这两个请求。对于路由器 R2 来说也是一样的道理。读者可能会问为什么 Rc2 和 Rc3 都属于同一个流，但是却请求不同的带宽？答案是：在多媒体环境中，不同的接收方可能处理不同等级的质量。例如，Rc2 也许只能接收 1 Mbps 的（低质量）视频，而 Rc3 却能够接收 2 Mbps 的（高质量）视频。

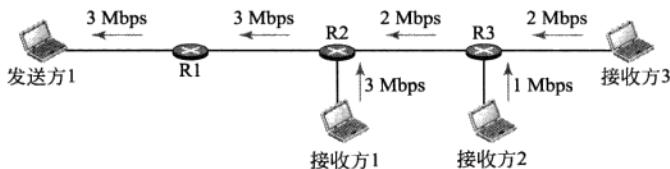


图 25.37 预留合并

预留风格

当存在多个流的时候，路由器需要预留能够容纳所有流的资源。RSVP 定义了三种预留风格，如图 25.38 所示。

通配符过滤器风格 在这种风格下，路由器为所有发送方创建一个预留。这个预留是基于最大请求的。当来自不同发送方的流不会同时发生时，我们使用这种风格。

固定过滤器风格 在这种风格下，路由器为每一个流创建一个不同的预留。这意味着，如果存在 n 个流，就要做 n 个不同的预留。当来自不同发送方的流很有可能同时出现时，我们使用这种风格。

共享显式风格 在这种风格下，路由器创建一个可以被一组流共享的预留。

软状态

一个流保存在每个结点中的预留信息（状态）需要被定期刷新。这个状态被称为软状态，用以区别在其他虚电路协议（如 ATM 或帧中继）中的硬状态，在这些协议中，有关流的信息会一直保留，直到被擦除。目前软状态的默认刷新间隔是 30 秒。



图 25.38 预留风格

25.11.6 综合服务存在的问题

至少有两个问题影响了综合服务在因特网中的全面实施：可扩展性和服务类型限制。

可扩展性

综合服务模型要求每个路由器为每个流都保留信息。随着因特网每天都在变大，所以这是个严重的问题。

服务类型限制

综合服务模型只提供了两种类型的服务：保证服务和控制负载服务。反对这个模型的观点认为应用程序有可能需要更多类型的服务。

25.12 区分服务

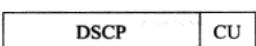
区分服务 (Differentiated Services, DS 或 Diffserv) 是由 IETF (因特网工程部) 提出的，目的是为了解决综合服务的缺陷。其中有两个根本性的改变：

1. 主要处理过程从网络核心部分移到了网络边缘部分。这样就解决了可扩展性问题。路由器不需要存储有关流的信息。应用程序或主机在每次发送分组时需要定义它们所需要的服务类型。
2. 基于每个流的服务改成了基于每个类的服务。路由器根据分组中定义的服务类别来转发分组，而不是根据流。这样就解决了服务类型限制的问题。我们可以根据应用程序的不同需要定义不同的类别。

区分服务是为 IP 而设计的基于类别的 QoS 模型。

DS 字段

在区分服务中，每个分组都包含了一个字段称为 DS 的字段。这个字段的值在网络的边界处由主机或者第一个被设置为边界路由器的路由器设定。因特网工程部提议用 DS 字段代替现存的 IPv4 中的 TOS (服务类型) 字段或 IPv6 中的



class 字段，如图 25.39 所示。

图 25.39 DS 字段

DS 字段包含两个子字段：DSCP 和 CU。区分服务代码

点 (Differentiated Services Code Point, DSCP) 是一个 6 位的子字段，定义了每跳行为 (per-hop behavior, PHB)。2 位的目前未使用 (Currently Unused, CU) 子字段目前没有使用。

能够支持 Diffserv 的结点 (路由器) 把 DSCP 的 6 位用做一张表的索引，这张表为正在处理的当前分组定义了分组处理的机制。

每跳行为

Diffserv 模型为每个接收分组的结点定义了一些每跳行为 (PHB)。目前为止已经定义了三种每跳行为：默认行为 (DE PHB)、加快转发行 (EF PHB) 和保证转发行 (AF PHB)。

默认行为 默认行为 (default PHB, DE PHB) 与尽最大努力交付相同，并与 TOS 兼容。

加快转发行 加快转发行 (expedited forwarding PHB, EF PHB) 提供如下服务：

- 低损失
- 低延迟
- 保证带宽

加快转发行为等同于在源和终点之间架设一条虚连接。

保证转发行为 保证转发行为 (assured forwarding PHB, AF PHB) 为分组的交付提供高度保证, 只要该类别的通信量不超过结点的通信量设定。网络用户需要注意有些分组还是可能被丢弃。

通信量调节器

为了实现 Diffserv, DS 结点需要使用像测量器、标记器、整形器和丢包器这样的通信量调节器, 如图 25.40 所示。

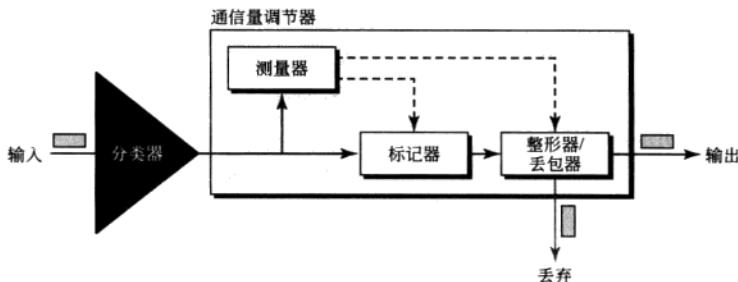


图 25.40 通信量调节器

测量器 测量器 (Meter) 检查输入的流是否与已协商的通信量设置匹配。测量器同时也把检查结果发送给其他组件。测量器可以使用多种工具来检查设置, 例如权标桶。

标记器 标记器 (Marker) 可以为使用尽最大努力交付 (DSCP: 000000) 的分组重新设定标记, 它也可以根据来自测量器的信息来降低一个分组的标记。降低分组标记 (降低流的类别) 发生在该流不能与协商的设置相匹配时。标记器不能提高一个分组的标记 (提升类别)。

整形器 整形器 (Shaper) 利用从测量器接收到的信息来对通信量进行重新整形, 如果该流与协商的设置不匹配的话。

丢包器 丢包器 (Dropper) 如同一个没有缓存的整形器, 当该流严重违反已协商的设置时, 就丢弃这些分组。

25.13 深入阅读

要更细致地了解本章所讨论的内容, 我们推荐以下书籍和 RFC。被方括号括起来的书目可以在本书末尾的参考书目清单中找到。

25.13.1 参考书

有几本书在某种程度上涵盖了有关多媒体的内容。[Com 06]讨论了基于 IP 的视频和音频。[Kur & Ros 08]和[Tan 03]中有关于多媒体的详尽讨论。[Gar & Vid 04]给出了有关音频和视频压缩的详尽讨论。

25.13.2 RFC

有几份 RFC 给出了本章所讨论的话题及其更新发展过程，它们包括 RFC 2198、RFC 2250、RFC 2326、RFC 2475、RFC 3246、RFC 3550 以及 RFC 3551。

25.14 重要术语

双向帧（B 帧）	像素
压缩	重放缓存
延迟	预测帧（P 帧）
区分服务（DS 或 Diffserv）	预测编码
离散余弦变化（DCT）	优先级排队
先进先出（FIFO）排队	服务质量（Qos）
频率掩蔽	量化
网闸	实时流式协议（RTSP）
网关	实时运输控制协议（RTCP）
H.323	实时运输协议（RTP）
综合服务（IntServ）	注册机构服务器
交互式音频/视频	可靠性
内编码帧（I 帧）	资源预留协议（RSVP）
抖动	Resv 报文
联合照相专家组（JPEG）	会话发起协议（SIP）
漏桶算法	空间压缩
媒体播放器	流式直播音频/视频
媒体服务器	流式存储音频/视频
元文件	时间压缩
混合器	时间掩蔽
活动图像专家组（MPEG）	时间戳
多播	权标桶
按需音频/视频	通信量整形
开环拥塞控制	转换
Path 报文	IP 话音
感知编码	加权公平排队
每跳行为（PHB）	

25.15 本章小结

- 音频/视频文件可以下载以后使用（用做流式存储音频/视频）或在因特网上广播给

客户（流式直播音频/视频）。因特网还可用做直播音频/视频交互。音频和视频在发送到因特网之前必须进行数字化。

- 音频文件可通过预测编码或感知编码进行压缩。联合图像专家组（JPEG）是一种压缩图像和图形的方法。活动图像专家组（MPEG）是一种压缩视频的方法。
- 我们可以使用万维网服务器、具有元文件的万维网服务器、媒体服务器和 RTSP 来下载流式音频/视频文件。
- 在分组交换网上的实时数据需要保留一个会话中的各分组之间的时间关系。在接收方，连续的分组之间的间隙产生了称为抖动现象。通过使用时间戳和正确选择重放时间可以控制抖动。
- 实时的多媒体通信量既需要 UDP，也需要实时运输协议（RTP）。RTP 处理打时间戳、编序号和混合。实时运输控制协议（RTCP）提供了流量控制、数据质量控制和到源点的反馈。
- IP 话音是一种实时交互式的音频/视频应用。会话发起协议（SIP）是应用层协议，用来建立、管理和终止多媒體会话。H.323 是一个 ITU 标准，它允许连接在公用电话网上的电话可以和连接在因特网上的计算机之间进行通话。
- 一个流可以以它的可靠性、延迟、抖动和带宽为特征。调度、通信量整形，资源预留和许可控制是提高服务质量（QoS）的技术。
- 综合服务是为 IP 设计的基于流的 QoS 模型。资源预留协议（RSVP）是一个信令协议，它可以帮助 IP 创建流并进行资源预留。区分服务是为 IP 设计的基于类别的 QoS 模型。

25.16 实践安排

25.16.1 习题

1. 在图 25.17 中，在以下的每一个时间，重放缓存中的数据量分别是多少？
 - a. 00:00:17
 - b. 00:00:20
 - c. 00:00:25
 - d. 00:00:30
2. 试把 RTP 与 TCP 进行比较和对比。它们所做的都是同样的事吗？
3. 我们能否说 UDP 加上 RTP 就和 TCP 一样？
4. 为什么 RTP 需要另一个协议（RTCP）的服务，而 TCP 就不需要？
5. 在图 25.12 中，万维网服务器和媒体服务器能否在不同的机器上运行？
6. 在本章我们讨论 SIP 用于音频。有没有缺点使它不宜用于视频？
7. 你认为 H.323 实际上是和 SIP 一样吗？它们的区别是什么？试对这两者进行一些比较。
8. H.323 能否用于视频？
9. 在使用漏桶控制液体流动时，假设输出速率为 5 加仑/分钟，如果有一个突发流

入为 100 加仑/分钟，并持续 12 秒，接着有 48 秒没有液体流入，此时桶中还剩多少加仑的液体？

10. 假设一个路由器的输出接口使用漏桶算法设计，每秒（时钟周期）可以发送 8000 字节。如果下列帧按顺序被接收，试给出在每秒钟内发出的帧。

- a. 帧 1, 2, 3, 4：每帧 4000 字节
- b. 帧 5, 6, 7：每帧 3200 字节
- c. 帧 8, 9：每帧 400 字节
- d. 帧 10, 11, 12：每帧 2000 字节

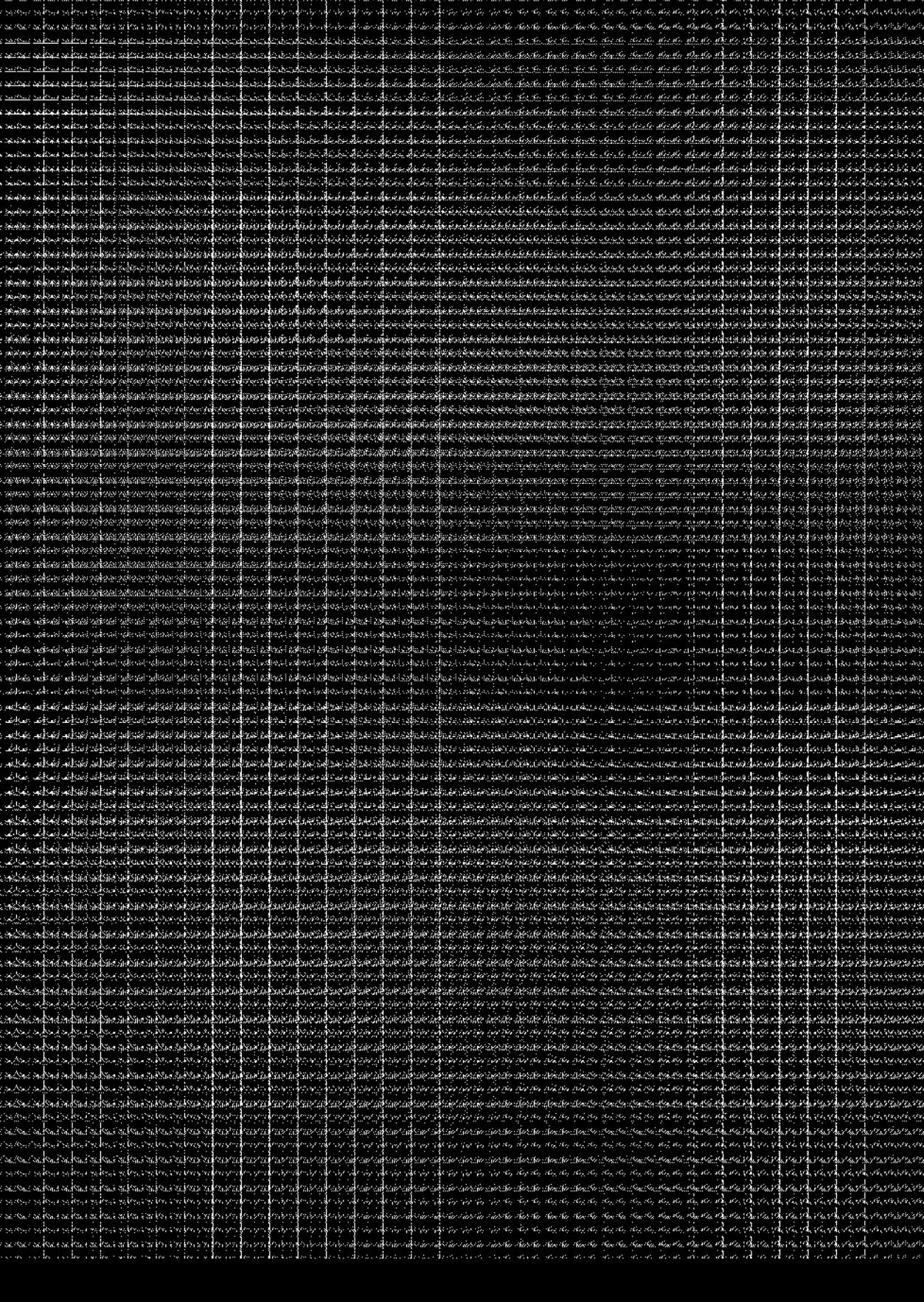
第五部分

下一 代

第 26 章 IPv6 编址

第 27 章 IPv6 协议

第 28 章 ICMPv6



第 26 章 IPv6 编址

在本书第五部分的第一章，我们要介绍的是 IPv6 中的编址。IPv4 协议的地址耗尽问题是开发 IPv6 协议的一个主要原因。正如我们将在本章看到的，IPv6 地址的结构与 IPv4 地址的结构有着一些本质上的不同。在 IPv6 中再也不用担心地址耗尽问题了。

目标

本章有以下几个目标：

- 介绍 IPv6 的编址机制以及这一版所使用的几种表示地址的不同记法。
- 解释 IPv6 中使用的三类编址类型：单播、任播和多播。
- 说明 IPv6 的地址空间，以及它是如何被划分为若干个地址块的。
- 讨论地址空间中的一些被保留的地址块以及它们的应用。
- 定义全球单播地址块以及它如何被应用于单播通信。
- 讨论在 IPv6 布署全球单播地址块时，如何应用了分为三级的分级编址技术。
- 讨论 IPv6 地址的自动配置和重新编号问题。

26.1 引言

IPv6 地址的长度是 128 位或者说 16 个字节（八位组），如图 26.1 所示。IPv6 地址的长度是 IPv4 地址长度的四倍。

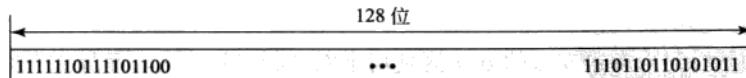


图 26.1 IPv6 地址

26.1.1 记法

在计算机中，地址通常以二进制保存，但是很显然，128 位的长度对人类来说不是那么容易掌握的。为了让人类能够处理这些地址，已经建议了多种用来表示 IPv6 地址的记法。

点分十进制记法

为了与 IPv4 地址兼容，我们会很自然地想到要使用点分十进制记法，就像表示 IPv4

地址那样(第 5 章)。虽然这种记法对于 4 字节的 IPv4 地址很方便,但是对于 16 字节的 IPv6 地址来说,这种记法显得有点太长,如下所示:

221.14.65.11.105.45.170.34.12.234.18.0.14.0.115.255

这种记法很少被使用,除了有些情况可能会使用其中的某一部分,稍后将会看到。

十六进制冒号记法

为了使地址的可读性更好,IPv6 地址协议指明了十六进制冒号记法 (colon hexadecimal notation)。在这种记法中,128 位被划分为八区,每个区的长度为两字节。在十六进制记法中,两个字节需要 4 个十六进制数字,因此,IPv6 地址由 32 个十六进制数字组成,每 4 个数字用一个冒号分隔开。图 26.2 所示为用十六进制冒号记法表示的一个 IPv6 地址。

FDEC : BA98 : 7654 : 3210 : ADBF : BBFF : 2922 : FFFF

图 26.2 十六进制冒号记法

这个 IP 地址即使用十六进制格式表示起来也很长,不过其中有许多数字都是零。在这种情况下,我们可以对这个地址进行简写。一个区(即两个冒号之间的 4 个数字)开头的几个零可以忽略。使用这种简写方式,0074 可以写为 74,000F 可以写为 F,而 0000 则写为 0。请注意,3210 不能被简写。

如果连续几个区都只包含了 0,那么这个十六进制冒号记法还可以更进一步简写,通常称为零压缩(zero compression)。我们可以把这些零全部去掉,取而代之的是一个双冒号。图 26.3 描绘了这个概念。



图 26.3 零压缩

请注意,这种类型的简写对一个地址仅能使用一次。若有两串零段,则只能将其中之一进行压缩。

混合表示法

有时候我们会看到一种混合表示的 IPv6 地址:十六进制冒号记法加上点分十进制记法。这种表示方法适用于将一个 IPv4 地址嵌入到一个 IPv6 地址中的过渡时期(作为最左边的 32 位)。我们可以对地址最左边的 6 个区使用十六进制冒号记法,而最靠右的两个区则以 4 个字节的点分十进制记法取代,如下所示:

FDEC:14AB:2311:BBFE:AAAA:BBBB:130.24.24.18

但是,通常只在这个 IPv6 地址最右边的区全部或大部分为零时才会这样写。例如,下面所示为一个合法的 IPv6 地址,其中的零压缩表示这个地址最左边的 96 位全部是零:

::130.24.24.18

CIDR 记法

稍后我们会看到,IPv6 使用了分级的编址技术。正是因为这个原因,IPv6 允许无

分类编址和 CIDR 记法。例如，图 26.4 所示为我们如何使用 CIDR 定义一个 60 位的前缀。在后面我们将会说明一个 IPv6 地址是如何划分前缀和后缀的。

FDEC :: BBFF : 0 : FFFF/60

图 26.4 CIDR 地址

例 26.1

给出以下 IPv6 地址的未经简写的十六进制冒号记法表示：

- 64 个 0 之后跟着 64 个 1 的地址。
- 128 个 0 组成的地址。
- 128 个 1 组成的地址。
- 128 个 1 和 0 交替组成的地址。

解

- 0000:0000:0000:0000:FFFF:FFFF:FFFF:FFFF
- 0000:0000:0000:0000:0000:0000:0000:0000
- FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
- AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA

例 26.2

下面所示为例 26.1 中的地址经过零压缩后（c 和 d 部分不能被简写）的形式：

- ::FFFF:FFFF:FFFF:FFFF
- ::
- FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
- AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA

例 26.3

给出以下地址的简写表示：

- 0000:0000:FFFF:0000:0000:0000:0000:0000
- 1234:2346:0000:0000:0000:0000:0000:1111
- 0000:0001:0000:0000:0000:1200:1000
- 0000:0000:0000:0000:0000:FFFF:24.123.12.6

解

- 0:0:FFFF::
- 1234:2346::1111
- 0:1::1200:1000
- ::FFFF:24.123.12.6

例 26.4

对以下地址进行解压缩，给出对应的未经简写的 IPv6 地址的完整表示：

- 1111::2222
- ::
- 0:1::
- AAAA:A:AA::1234

解

- 1111:0000:0000:0000:0000:0000:2222

- b. 0000:0000:0000:0000:0000:0000:0000:0000
- c. 0000:0001:0000:0000:0000:0000:0000:0000
- d. AAAA:000A:00AA:0000:0000:0000:0000:1234

26.1.2 地址空间

IPv6 的地址空间包含了 2^{128} 个地址，如下所示。这个地址空间是 IPv4 地址数量的 2^{96} 倍，肯定不会存在地址耗尽的问题了。

$$2^{128} = 340\,282\,366\,920\,938\,463\,374\,607\,431\,768\,211\,456$$

例 26.5

为了对这个地址数量有一个直观的印象，让我们假设地球上的总人口数量即将达到 2^{34} （超过 160 亿）。那么每个人仍然有 2^{94} 个地址可供使用。

例 26.6

如果我们每年向用户指派 2^{60} （几乎每秒 10 亿）个地址，那么需要花 2^{68} 年才能让地址耗尽。

例 26.7

如果在地球的整个表面上，不管是陆地还是海洋，建造一座高耸入云的大楼，使得每一平方米的地面上都能容纳 2^{68} 台计算机，还是有足够的地址可以把所有的计算机都连接到因特网上（地球的表面面积大约为 2^{60} 平方米）。

26.1.3 三种地址类型

在 IPv6 中，一个目的地址可以属于以下三种类型之一：单播的、任播的和多播的。

单播地址

单播地址定义了一个接口（计算机或路由器）。发送到单播地址的分组必须交付给这个指定的计算机。我们将在稍后会看到，IPv6 为单播通信设计了一个很大的地址块，其中的单播地址都可以指派给接口使用。

任播地址

任播地址定义了一组共享一个地址的计算机。发送到任播地址的分组会被交付给这个组的成员之一，也就是最容易到达的那个。例如，当多台服务器都能响应某个查询时，就可以使用任播通信。这个查询请求被发送到最容易到达的那台服务器。产生请求的硬件和软件只生成了一份请求的副本，而该副本也仅到达其中的一台服务器。IPv6 没有为任播设计专门的地址块，这些任播地址是从单播地址块中指派的。

多播地址

多播地址定义的也是一组计算机，但是任播和多播是有区别的。在多播通信中，多播组的每个成员都会接收到一个副本。稍后我们将会看到，IPv6 发送到多播地址的分组必须交付给该组中的每一个成员。IPv6 为多播通信设计了一个地址块，从这个地址块中可以向一个组的所有成员指派相同的地址。

26.1.4 广播和多播

有趣的是 IPv6 没有定义广播通信，哪怕是在有限范围内的广播，这一点不像 IPv4。在第 5 章中，我们讨论了在一个地址块中的某些地址能够被用于有限的广播。我们将会看到，IPv6 认为广播就是多播的一种特殊情况。

26.2 地址空间分配

像 IPv4 的地址空间一样，IPv6 的地址空间被划分为若干个大小不同的地址块，并为每种特定的任务分配一个地址块。这些地址块中的绝大部分尚未指派，被留作将来使用。为了更好地理解地址空间中每个地址块的分配及其所处的位置，我们首先将完整的地址空间划分为八个大小相等的区。这种划分并不表示地址块的分配，只是我们认为这样做可以更好地说明每个实际的地址块所处的位置（图 26.5）。



图 26.5 地址空间的分配

每一个区是整个地址空间的 1/8（即 2^{125} 个地址）。第一个区包含了六个长度不等的地址块，其中三个地址块是保留的，另外三个地址块是未指派的。第二个区被认为就是一个地址块，用于全球单播地址，我们在本章的后面将会进一步讨论。接下来的五个区都是未指派的地址。最后一个区划分为八个地址块，其中有些地址块属于尚未指派的地址，另一些则被保留作特殊用途。从图中可以看出有 5/8 的地址空间还没有被指派。只有 1/8 的地址空间用于用户之间的单播通信。

表 26.1 给出了每种类型的地址的前缀。第五列表示每一种地址类型相对于整个地址空间所占据的份额。最左边一列不是标准中的一部分，它只是显示了图 26.5 所描绘的地址区。

表 26.1 IPv6 地址的类型前缀

地址块前缀	CIDR	地址块分配	份 额
1 0000 0000	0000::/8	保留 (IPv4 兼容)	1/256
	0100::/8	保留	1/256
	0200::/7	保留	1/128

续表

地址块前缀	CIDR	地址块分配	份 额
0000 01	0400::/6	保留	1/64
0000 1	0800::/5	保留	1/32
0001	1000::/4	保留	1/16
2 001	2000::/3	全球单播	1/8
3 010	4000::/3	保留	1/8
4 011	6000::/3	保留	1/8
5 100	8000::/3	保留	1/8
6 101	A000::/3	保留	1/8
7 110	C000::/3	保留	1/8
8 1110	E000::/4	保留	1/16
1111 0	F000::/5	保留	1/32
1111 10	F800::/6	保留	1/64
1111 110	FC00::/7	唯一的本地单播	1/128
1111 1110 0	FE00::/8	保留	1/512
1111 1110 10	FE80::/10	本地链路地址	1/1024
1111 1110 11	FEC0::/10	保留	1/1024
1111 1111	FF00::/8	多播地址	1/256

例 26.8

图 26.5 仅显示出了用于全球单播通信的地址在地址空间中是哪一部分，那么在这个地址块中一共有多少个地址？

解

这个地址块仅占地址空间的 1/8。为了计算这些地址的数量，我们可以用完整的地址空间除以 8 或 2^3 。结果是 $(2^{128})/(2^3) = 2^{125}$ ，很大的一个地址块。

算法

为了说明根据表 26.1 中列出的那些前缀，能够为一个 IPv6 地址无二义性地找到其相应的地址块，我们构造了如图 26.6 所示的算法流程图。这个算法可用于编写一段程序，以找出一个给定的地址所属的地址块。这个算法最多只需检查 10 位就能找出该地址所属的地址块。请注意，为了保持该图的简洁性，图中没有显示保留的地址块（除了与 IPv4 兼容的地址）。

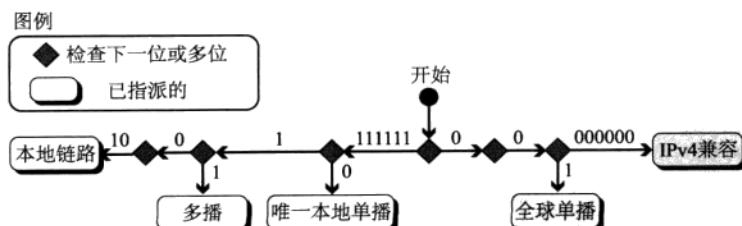


图 26.6 查找所分配的地址块的算法

26.2.1 指派的和保留的地址块

这一小节要讨论各个已指派的和被保留的地址块的特点和作用，让我们从表 26.1 的第一行开始。

IPv4 兼容地址

使用前缀（00000000）的地址是保留的，但是其中有部分用来定义了一些IPv4兼容地址。这个地址块占据整个地址空间的1/256，也就是说在这个地址块中一共有 2^{120} 个地址。用CIDR记法时，这个地址块可定义为0000::/8。这个地址块进一步被划分为若干个子块，下面将具体讨论。

未指明地址 未指明地址是只包含了一个地址的子地址块，它定义的这个地址的所有后缀部分也都是零。换言之，整个的地址都是由零组成。这个地址用于主机引导期间，当主机不知道自己的地址而需要发送查询以便找出自己的地址时。因为任何 IPv6 分组都需要一个源地址，所以该主机就使用这个地址来完成此任务。请注意，未指明地址不能用做目的地址。这个单地址子块的 CIDR 记法是 $::/128$ 。图 26.7 所示为未指明地址的格式。

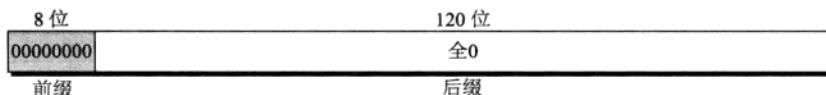


图 26.7 未指明地址

IPv6 中的未指明地址是 ::/128。

它永远也不应当被用做目的地址。

例 26.9

试比较 IPv4 中的未指明地址和 IPv6 中的未指明地址。

解

在这两种体系结构中，未指明地址都是一个全零的地址。在 IPv4 中，这个地址是属于 A 类地址的一部分。在 IPv6 中，这个地址是保留地址块中的一部分。

环回地址 这个子块同样也只包含了一个地址。我们在第 5 章中讨论过环回地址。这个地址是主机在不需要连接到网络的情况下用来测试它自己的。在这种情况下，由应用层产生一个报文，发送到运输层，再传递到网络层。但是，这个报文并没有传送到物理网络中，而是返回到运输层，再传递到应用层。在把计算机连接到网络上之前，可以用这个地址来测试上面几层的软件包的功能。图 26.8 描绘了环回地址，它由前缀 00000000 和后面跟着 119 个 0 和一个 1 组成。这个单地址子块的 CIDR 记法是::1/128。

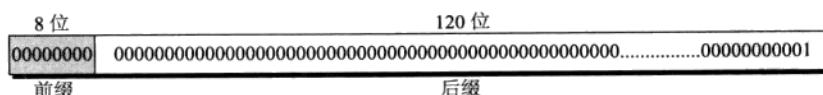


图 26.8 环回地址

例 26.10

试比较 IPv4 中的环回地址和 IPv6 中的环回地址。

解

它们之间有两点不同。在分类编址中分配给环回地址的是一个完整的地址块，而在 IPv6 中只有一个地址被作为环回地址分配。另外，分类编址中的环回地址块是 A 类地址块的一部分，而在 IPv6 中，它只是保留地址块中的一个地址。

嵌入的 IPv4 地址 我们将在第 27 章中看到，从 IPv4 到 IPv6 的过渡期间，主机可以在 IPv6 中嵌入它们的 IPv4 地址。为此而设计了两种格式：兼容的和映射的。**兼容地址** (*compatible address*) 是在 96 位 0 之后紧跟 32 位的 IPv4 地址。当使用 IPv6 的计算机要把报文发送给另一个使用 IPv6 的计算机，但是，分组又必须要通过仍然使用了 IPv4 网络的区域时，就要使用这种地址。发送方必须使用与 IPv4 兼容的地址，使得分组能够通过 IPv4 的网络区域。例如，IPv4 地址 2.13.17.14 (点分十进制格式表示) 被转换为 0::2.13.17.14 (混合格式表示)。在这个 IPv4 地址前面加上 96 个 0 产生了一个 128 位的 IPv6 地址(见图 26.9)。这个子地址块是能够包含 2^{32} 个地址的保留区，它的 CIDR 记法是 ::/96。我们将在第 27 章更详细地讨论此类地址。

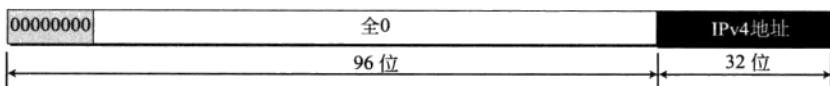


图 26.9 兼容地址

一个**映射地址** (*mapped address*) 由 80 位 0 之后紧跟着 16 位 1，再后面跟着的就是 32 位的 IPv4 地址。当某个计算机已经过渡到 IPv6 而打算把分组发送给一个仍然使用 IPv4 的计算机时，就要使用这种地址。这个分组所经过的大部分网络是 IPv6 的，但最后要交付到使用 IPv4 的主机。例如，IPv4 地址 2.13.17.14 (点分十进制格式) 被转换为 0::FFFF:2.13.17.14 (十六进制冒号格式)。IPv4 地址的前面加上 80 个 0 和 16 个 1，得到一个 128 位的 IPv6 地址 (见第 27.3 节有关过渡策略的介绍)。图 26.10 所示为一个映射地址。

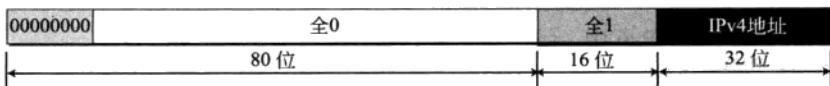


图 26.10 映射地址

关于映射地址和兼容地址有一个非常有意思的地方，它们的设计使得在计算检验和时，可以使用嵌入的地址，也可以使用完整的地址，因为额外的 0 或 1，只要其个数是 16 的倍数，对检验和的计算来说就没有任何影响。这一点对于使用了需要计算检验和的伪首部的 UDP 和 TCP 来说很重要，因为若分组的地址被某个路由器从 IPv6 地址变为 IPv4 地址后，检验和的计算不受影响。

全球单播地址块

这是因特网上主机与主机之间的单播通信所使用的主要地址块。我们将在后面全面而详细地讨论这个地址块，以说明它是如何在因特网中被用来提供分级编址的。

唯一的本地单播地址块

我们在第 5 章介绍 IPv4 协议时讨论过专用地址。我们说在 IPv4 地址空间中有一些地址块被保留做专用地址。IPv6 为专用地址分配了两个地址块：一个是站点级的，一个是链路级的。在这一小节我们先讨论前一个地址块，下一小节再讨论后一个地址块。

在唯一的本地单播地址块 (unique local unicast block) 中有一个子块可被站点以个人行为进行创建和使用。这类地址作为目的地址的分组是不会被路由器转发的。这类地址的块标识是 1111 110，紧接着的一位可以是 0 或者 1，它们定义了这个地址是如何被选择的（本地选择的，还是管理机构选择的）。接下来的 40 位是由站点选择的，它使用了一个长度为 40 位的随机生成数。也就是说这整个 48 位定义了一个看起来很像全球单播地址的子地址块。40 位的随机数使得地址重复的可能性极小，参见图 26.11。请注意，这些地址在格式上和全球单播地址（本章后面讨论）有着相似之处。

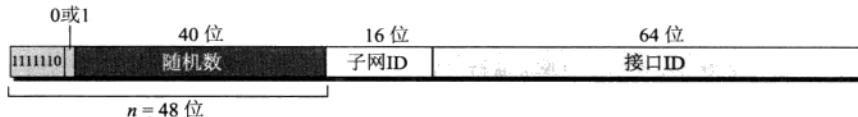


图 26.11 唯一的本地单播地址块

本地链路地址块

设计给专用地址的第二个地址块是本地链路地址块 (link local block)。在这个地址块中有一个子块可用做网络中的专用地址。这类地址的块标识是 1111111010。接下来的 54 位都被设置为 0。最后的 64 位是可变的，以定义每个计算机的接口（参见图 26.12）。请注意，这些地址在格式上与全球单播地址（本章后面讨论）有着相似之处。

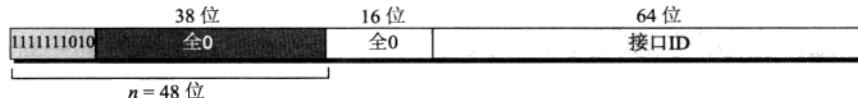


图 26.12 本地链路地址

多播地址块

我们在第 5 章介绍 IPv4 协议时讨论过多播地址。多播地址用于定义了一组主机而不是仅仅一个主机。在 IPv6 中为多播通信指派了一个很大的地址块。所有多播地址都使用前缀 11111111。第二个字段是定义组地址为永久的或暂时的一个标志。永久组地址由因特网管理机构定义，并可在任何时间进行访问。另一方面，暂时组地址只是临时使用。例如，加入到远程会议中的各个系统就可以使用一个暂时的组地址。第三个字段定义组地址的范围。现在已经定义了许多不同的范围，如图 26.13 所示。

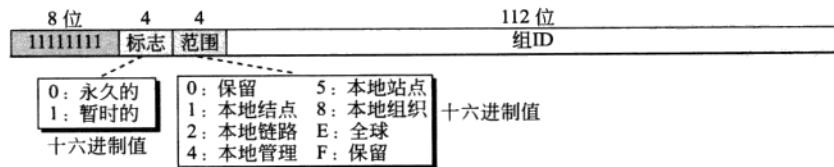


图 26.13 多播地址

26.3 全球单播地址

在 IPv6 地址空间中，用于因特网上两个主机之间的单播（一对一）通信的地址块称为全球单播地址块。这个地址块的 CIDR 记法是 2000::/3，也就是说，对这个地址块中的所有地址来说最左边的三位都相同（001）。这个地址块的大小是 2^{125} ，就因特网的发展速度来看，它在未来很多年以后都足够用了。

26.3.1 三级结构

这个地址块中的地址被划分为三个部分：全球路由选择前缀、子网标识和接口标识，如图 26.14 所示。

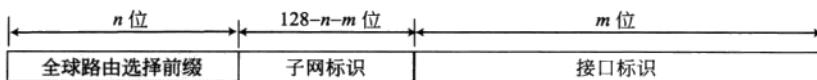


图 26.14 全球单播地址

这三个部分的推荐长度如表 26.2 所示。

表 26.2 单播地址中各部分的推荐长度

块的指派	长 度
全球路由选择前缀 (n)	48 位
子网标识 ($128 - n - m$)	16 位
接口标识 (m)	64 位

全球路由选择前缀

全球单播地址的最前面 48 位称为全球路由选择前缀。这 48 位用来为分组选择路由以通过因特网到达某个组织的站点，比如说拥有这个地址块的 ISP。因为在这个部分中的最前面三位是固定的（001），所以剩下的 45 位就可以定义高达 2^{45} 个站点（个体组织或 ISP）。全球因特网上的路由器都会根据 n 的值来转发分组以到达它的目的站点。

子网标识

接下来的 16 位定义了组织中的一个子网。这就意味着一个组织可以拥有高达 $2^{16} = 65536$ 个子网，这个数量显然是足够用了。

接口标识

最后 64 位定义了接口标识。这个接口标识类似于 IPv4 的主机标识，只不过用接口标识这个术语会更加确切一些，因为正如我们在第 5 章中所讨论的，实际上主机标识定义的也是一个接口，而非一个主机。如果主机从一个接口移动到另一个接口，它的 IP 地址也要随之改变。

在 IPv4 编址中，在主机标识（IP 层的）和物理或 MAC 地址（数据链路层的）之间是没有特殊关联的，因为通常物理地址的长度要远远超出主机标识的长度。例如，在使用以太网

技术时，物理地址的长度是 48 位，而主机标识的长度小于 32 位。但是 IPv6 编址就允许这种可能性的存在。只要长度小于 64 位的物理地址都可以被嵌入到接口标识中，作为该接口标识的一部分或者全部，从而消除了地址映射过程。对此可以考虑两种常见的物理编址机制：由 IEEE 定义的 64 位的扩展唯一标识（EUI-64）和由以太网定义的 48 位的物理地址。

EUI-64 的映射 为了映射 64 位的物理地址，该格式中的全球/本地控制位需要从 0 变为 1（从本地的变为全球的）以定义一个接口地址，如图 26.15 所示。

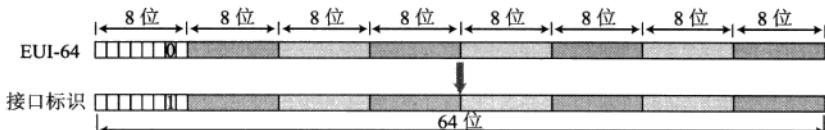


图 26.15 EUI-64 的映射

以太网 MAC 地址的映射 将一个 48 位的以太网地址映射为一个 64 位的接口地址要更复杂一些。我们需要把全球/本地控制位变为 1，还要另外插入 16 位。这个附加的 16 位被定义为 15 个 1 后面跟着 1 个 0，或 FFFE_{16} 。图 26.16 描绘了此映射。

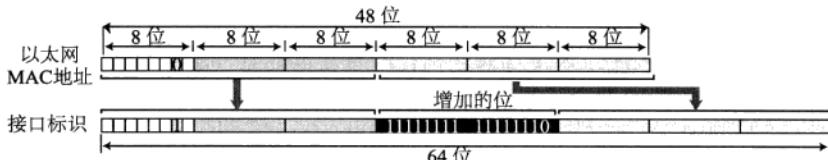


图 26.16 以太网 MAC 地址的映射

例 26.11

如果一个 EUI 物理地址是 $(\text{F5-A9-23-EF-07-14-7A-D2})_{16}$ ，试用我们为 EUI 地址定义的格式来给出这个接口标识。

解

我们只需要将第一个八位组的第七位从 0 变为 1，然后再转换成十六进制冒号记法。其结果是 $\text{F7A9:23EF:0714:7AD2}$ 。

例 26.12

如果一个以太网地址为 $(\text{F5-A9-23-14-7A-D2})_{16}$ ，试用我们为以太网地址定义的格式来给出这个接口标识。

解

我们只需要将第一个八位组的第七位从 0 变为 1，然后插入两个八位组 FFFE_{16} ，再将其转换成十六进制冒号记法。结果是 $\text{F7A9:23FF:FE14:7AD2}$ 。

例 26.13

一个组织经指派得到了地址块 $2000:1456:2474/48$ 。这个组织的第一个和第二个子网的地址块的 CIDR 记法是什么？

解

从理论上说，第一个和第二个子网应当分别使用子网标识为 0001_{16} 和 0002_{16} 的地址块。

也就是说这两个地址块是 2000:1456:2474:0000/64 和 2000:1456:2474:0001/64。

例 26.14

一个组织经指派得到了地址块 2000:1456:2474/48。如果在它的第三个子网中有一台计算机的以太网物理地址为(F5-A9-23-14-7A-D2)₁₆，那么这个接口的 IPv6 地址是什么？

解

这个接口的接口标识是 F7A9:23FF:FE14:7AD2（参见例 26.12）。如果我们在这个接口标识上再添加它的全球前缀和子网标识后，得到：

2000:1456:2474:0003:F7A9:23FF:FE14:7AD2/128

26.4 自动配置

IPv6 的一个很有意思的地方是主机的自动配置 (autoconfiguration)。如我们在讨论 IPv4 时提到的，主机和路由器最初是由网络管理员手工配置的。不过，动态主机配置协议 DHCP 可以为加入网络中的主机分配一个 IPv4 地址。在 IPv6 中，仍然可以使用 DHCP 来为主机分配一个 IPv6 地址，但是主机也可以自己进行配置。

当 IPv6 中的一个主机加入到网络上时，它可以按照以下过程对自己进行配置：

1. 主机首先为自己创建一个本地链路地址。具体做法是，先在 10 位的本地链路前缀 (1111 1110 10) 后添加 54 个 0，再加上 64 位的接口标识，所有主机都知道应该如何从它的接口卡上得到一个接口标识。最后的结果是 128 位的本地链路地址。
2. 主机测试这个本地链路地址是否唯一，是否没有被其他主机使用。因为 64 位的接口标识应当是唯一的，所以生成的这个本地链路地址是唯一的可能性很高。但是为了确保它的唯一性，主机要发送一个邻站询问报文（参见第 28 章）并等待邻站通告报文。如果该子网中有任何主机正在使用这个本地链路地址，那么这个过程就失败了，主机不能自动配置自己，它需要换别的方法，如 DHCP 协议。
3. 如果这个本地链路地址通过了唯一性测试，那么主机把这个地址作为它的本地链路地址保存起来（用于专用通信），但是它还需要一个全局单播地址。于是该主机向一个本地路由器发送路由器询问报文（参见第 28 章）。如果在这个网络上运行着一台路由器，那么主机就会收到一个路由器通告报文，其中包括了全局单播前缀和子网前缀，这正是主机需要用来添加到接口标识上的，从而产生该主机的全局单播地址。如果路由器不能帮助主机进行配置，那么该路由器就要在路由器通告报文中将这一情况如实相告（通过设置一个标志位），于是主机就需要用其他方法进行配置。

例 26.15

假设以太网地址为(F5-A9-23-11-9B-E2)₁₆的一台主机加入到网络中。如果该组织的全局单播前缀是 3A21:1216:2165 且子网标识为 A245:1232，那么它的全局单播地址应该是什么？

解

主机首先从接口卡上读出以太网地址，以生成自己的接口标识 F7A9:23FF:FE11:9BE2。然后这个主机创建自己的本地链路地址为

FE80::F7A9:23FF:FE11:9BE2

假定这个地址是唯一的，主机发送了一个路由器询问报文并收到路由器通告报文，路由器在报文中宣布全球单播前缀和子网标识的组合为 3A21:1216:2165:A245:1232。于是主机把自己的接口标识附加在这个前缀后面，就可以得到如下的全球单播地址并保存起来：

3A21:1216:2165:A245:1232:F7A9:23FF:FE11:9BE2

26.5 重新编号

为了允许站点更换自己的服务提供者，在 IPv6 编址中内建了对地址前缀（n）的重新编号（renumbering）。正如我们在前面所讨论的，每个站点都由它所连接的服务提供者指定一个前缀。如果站点要更换服务提供者，那么地址前缀也必须被更换。这个站点所连接的路由器可以通告一个新的前缀，同时也让站点在完全废除旧前缀之前，还可以短期内继续使用旧的前缀。换言之，在过渡期内一个站点有两个前缀。使用这种重新编号机制带来的最主要的问题是 DNS 的支持，DNS 需要传播与域名相关联的新地址。称为下一代 DNS 的一个新的 DNS 协议正在研究中，以提供对重新编号机制的支持。

26.6 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

26.6.1 参考书

有不少书籍全面地介绍了 IPv6。我们推荐[Com 06]和[Los 04]。

26.6.2 RFC

通过一些 RFC 可以看出 IPv6 地址的不断更新过程，包括 RFC 2375、RFC 2526、RFC 3513、RFC 3587、RFC 3789 和 RFC 4291。

26.7 重要术语

任播地址

兼容地址

自动配置

本地链路地址

十六进制冒号记法

本地链路地址块

映射的地址	唯一的本地单播地址块
重新编号	零压缩

26.8 本章小结

- IPv6 是网际协议的最新版本，它有 128 位的地址空间。IPv6 使用十六进制冒号记法，对其还可使用简写。有三种类型的地址：单播、任播和组播。变长的类型前缀字段定义了地址的类型或作用。
- 为了使地址的可读性更好，IPv6 地址协议指明了十六进制冒号记法。在这种记法中，128 位被划分为八个区，每个区的长度为 2 字节（四个十六进制数字）。为了简写地址，一个区的开头几个零可以忽略，还可以使用零压缩。IPv6 也允许 CIDR 记法。
- 在 IPv6 中，一个目的地址可以属于以下三种类型之一：单播的、任播的和多播的。单播地址定义了一个接口。任播地址定义了一组计算机，但是分组只有一个副本被发送到这个组中的某一台计算机上。多播地址也定义了一组计算机，这个组的每一个成员都会收到该分组的一个副本。
- IPv6 的地址空间被划分为若干个大小不同的地址块，并为每种特定的任务分配一个地址块。这些地址块中的绝大部分尚未被指派，留作将来使用。有些地址块用于保留的地址。最重要的地址块是前缀为 001 的块，它用于全球单播地址（类似于 IPv4 中的 A 类、B 类和 C 类地址）。
- 在 IPv6 编址中，有两个很有意思的特色是自动配置和重新编号。在 IPv6 中，除了使用 DHCP 外，主机还能自动地对自己进行配置。重新编号则允许一个站点把自己的连接更换到另一个提供者，并自动地接收一个新的前缀。

26.9 实践安排

26.9.1 习题

1. 试给出以下 IPv6 地址的未经简写的十六进制冒号记法：
 - a. 64 个 0 之后跟着 32 个两位的 01
 - b. 64 个 0 之后跟着 32 个两位的 10
 - c. 连续 64 个两位的 01
 - d. 连续 32 个四位的 0111
2. 试给出习题 1 中的地址经过零压缩后的写法。
3. 试给出以下地址的简写。
 - a. 0000:FFFF:FFFF:0000:0000:0000:0000:0000

- b. 1234:2346:3456:0000:0000:0000:FFFF
 - c. 0000:0001:0000:0000:0000:FFFF:1200:1000
 - d. 0000:0000:0000:0000:FFFF:FFFF:24.123.12.6
4. 对以下地址解压缩，给出对应的完整未经简写的 IPv6 地址：
- a. ::2222
 - b. 1111::
 - c. 0:1:2::
 - d. B:A:CC::1234:A
5. 给出以下地址的原始（未经简写）形式：
- a. 0::2
 - b. 0:23::0
 - c. 0:A::3
 - d. 123::12:23
6. 根据表 26.1，与以下每个地址相应的地址块或子块分别是什么？
- a. FE80::12
 - b. FEC0::24A2
 - c. FF02::0
 - d. 0::1
7. 根据表 26.1，与以下每个地址相应的地址块或子块分别是什么？
- a. 0::0
 - b. 0::FFFF:0:0
 - c. 582F:1234::2222
 - d. 4821::14:22
8. 如果一个接口的 EUI 物理地址是(F5-A9-23-AA-07-14-7A-23)₁₆，用我们为 IEEE EUI 地址定义的格式给出这个接口的标识。
9. 如果一个接口的以太网物理地址是(F5-A9-23-12-7A-B2)₁₆，用我们为以太网地址定义的格式给出这个接口的标识。
10. 某个组织被指派得到一个地址块为 2000:1234:1423/48。这个组织的第一个和第二个子网的地址块 CIDR 记法分别是什么？
11. 某个组织被指派得到一个地址块为 2000:1110:1287/48。如果它的第三个子网中有一台计算机的以太网物理地址为(F5-A9-23-14-7A-D2)₁₆，那么这个接口的 IPv6 地址是什么？
12. 使用 CIDR 记法，请给出 IPv4 地址 129.6.12.34 的 IPv6 兼容地址。
13. 使用 CIDR 记法，请给出 IPv4 地址 129.6.12.34 的 IPv6 映射地址。
14. 使用 CIDR 记法，请给出 IPv6 的环回地址。
15. 使用 CIDR 记法，请给出结点标识为 0::123/48 的本地链路地址。
16. 使用 CIDR 记法，请给出结点标识为 0::123/48 的本地站点地址。

第 27 章 IPv6 协议

在第 26 章我们讨论了 IPv6 的地址空间，并说明通过使用这种地址空间，我们在 IPv4 中遇到的地址耗尽问题已不足为虑。地址的改变使得 IP 数据报的格式也不得不发生改变，以便容纳 128 位的地址。我们将在本章看到，新的 IPv6 协议还对近几十年来 IPv4 在操作上遇到的一些问题一并作出响应。这一章专门讨论 IPv6 数据报的格式。在下一章，我们要讨论 ICMPv4 也因自身的一些缺点而被更新到 ICMPv6。

目标

本章有以下几个目标：

- 给出 IPv6 数据报的格式，它由一个基本首部和一个有效载荷组成。
- 讨论 IPv6 数据报基本首部中使用的各种字段，并将它们与 IPv4 数据报的字段相比较。
- 说明 IPv4 首部中的选项在 IPv6 中是如何通过扩展首部来实现的。
- 说明在 IPv6 中如何实现安全性。
- 讨论从 IPv4 过渡到 IPv6 所使用的三种策略：双协议栈、隧道技术和首部转换。

27.1 引言

在这一介绍性质的小节中，我们要讨论两个话题：采用新协议的缘由和采用进度延缓的原因。

27.1.1 改变的缘由

我们可以提出许多理由来说明为什么需要网际协议版本 6 (Internet Protocol version 6, IPv6) 这个新的协议。其中最主要的原因就是地址耗尽问题，这在前一章已经讨论过了。其他还有一些原因包括像某些不必要的处理过程而使得整个处理速度较慢，需要一些新的选项，需要对多媒体的支持，以及对安全性的渴求。

IPv6 协议采取了以下几个方面的改变，以应对上述问题：

- **更大的地址空间** IPv6 地址的长度为 128 位。与 32 位的 IPv4 地址相比，其地址空间的增加量是巨大的 (2^{96})。
- **更好的首部格式** IPv6 使用了新的首部格式，把它的选项和基本首部分开，并且

在需要时即可插入到基本首部与上层数据之间。这就简化和加快了路由选择的过程，因为大多数的选项并不需要被路由器检查。

- **新的选项** IPv6 有新的选项来实现更多的功能。
- **允许扩充** 当新的技术或应用有需要时，IPv6 在设计上允许协议进行扩充。
- **支持资源分配** 在 IPv6 中，服务类型字段取消了，但是增加了两个新字段，通信量类别和流标号，它们使得源点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这样的通信。
- **支持更多的安全性** IPv6 中的加密和鉴别选项提供了分组的保密性和完整性。

27.1.2 采用进度延缓的原因

IPv6 的采用进度已经变慢了。原因在于最初开发 IPv6 的动机是 IPv4 地址的紧缺，但由于有了三种短期解决的方法（无分类编址、使用 DHCP 进行动态地址分配以及 NAT），IPv4 地址紧缺问题已得到缓解。但是，因特网的快速发展和一些新服务的出现，如移动 IP、IP 电话、IP 移动电话等，都有可能要求 IPv6 完全代替 IPv4。人们曾经预测，到 2010 年分布于世界各地的所有主机都使用 IPv6，但是从本书付印之时来看，这件事已不可能。

27.2 分组格式

IPv6 的分组格式如图 27.1 所示。每一个分组由强制性的基本首部和紧随其后的有效载荷组成。有效载荷由两部分组成：可选的扩展首部和从上层传来的数据。基本首部共有 40 字节，而扩展首部和从上层来的数据包含了最高可达 65 535 字节的信息。

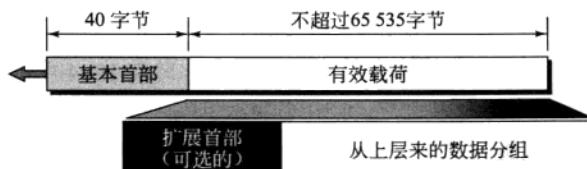


图 27.1 IPv6 数据报

27.2.1 基本首部

图 27.2 给出了具有八个字段的基本首部。

这些字段分别是：

- **版本** 这个 4 位字段定义了 IP 的版本号。对于 IPv6，其数值为 6。
- **通信量类别** 这个 8 位字段用于区分具有不同交付需求的各种有效载荷。它取代了 IPv4 的服务类型字段。

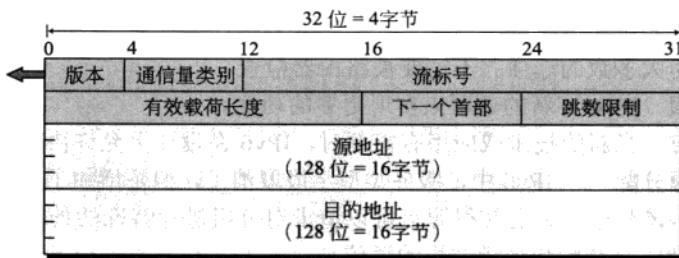


图 27.2 基本首部的格式

- **流标号** 流标号 (flow label) 是一个 20 位的字段，它的设计是为了对特定数据流提供特殊的处理。我们将在后面讨论这个字段。
- **有效载荷长度** 这个 2 字节的有效载荷长度字段定义了 IP 数据报除基本首部外的总长度。
- **下一个首部** 下一个首部 (next header) 是一个 8 位字段，它定义了在数据报中紧跟在基本首部后面的首部。下一个首部可能是 IP 自己要使用的一个可选的扩展首部，也可能是被封装的分组的首部，如 UDP 或 TCP。在每一个扩展首部中也包含了这个字段。表 27.1 给出了下一个首部的值。请注意，在版本 4 中这个字段称为协议字段。

表 27.1 下一个首部的代码

代 码	下一个首部	代 码	下一个首部
0	逐跳选项	44	分片
2	ICMP	50	加密的安全有效载荷
6	TCP	51	鉴别
17	UDP	59	空 (没有下一个首部)
43	源路由选择	60	终点选项

- **跳数限制** 这个 8 位的跳数限制 (hop limit) 字段和 IPv4 中的 TTL 字段的作用是一样的。
- **源地址** 源地址字段是 16 字节 (128 位) 的因特网地址，它标志了数据报的源点。
- **目的地址** 目的地址字段是 16 字节 (128 位) 的因特网地址，它通常标志数据报的最后终点。但是，若使用了源路由选择，这个字段包含的就是下一个路由器的地址。

27.2.2 流标号

IP 协议最初设计为一个无连接的协议。但是正如我们在第 4 章和第 6 章中讨论的，目前的趋势是将 IP 协议当作一个面向连接的协议使用。第 6 章讨论的 MPLS 技术使我们能够通过一个标号字段把 IPv4 分组封装在 MPLS 首部中。而在 IPv6 中，流标号字

段被直接加入到数据报的首部格式中，以允许我们将 IPv6 当作一个面向连接的协议来使用。

对路由器来说，一个流就是共享某些特性的一个分组序列，比如说经过相同的路径，使用相同的资源，或具有相同的安全类型等等。支持流标号处理的路由器拥有一张流标号表。这个表为每一个活动的流标号设置一个表项，每个表项定义了相应的流标号所需的服务。当路由器收到一个分组时，它就咨询自己的流标号表，找出该分组定义的流标号值所对应的表项，然后它就向该分组提供表项中提到的各种服务。但是请注意，流标号本身并没有给流标号表带来任何表项信息，这些信息是通过其他方法提供的，如逐跳选项或其他协议。

在最简单的形式下，流标号可用来加速路由器对分组的处理。当路由器收到一个分组时，它不用查找路由表并使用路由选择算法来确定下一跳的地址，而是可以很容易地在流标号表中找到下一跳的地址。

在更加复杂的形式下，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需要像高带宽、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这些资源还需要除 IPv6 之外的其他一些协议，如实时协议（RTP）和资源预留协议（RSVP），参见第 25 章。

为了有效地利用流标号，人们定义了以下三个规则：

1. 流标号由源主机指派给一个分组。这个标号是在 $1 \sim (2^{24}-1)$ 之间的一个随机数。当已存在的流仍处于活跃状态时，源点一定不能给新的流重复使用已用过的流标号。
2. 若主机不支持流标号，它应当把这个字段置为零。若路由器不支持流标号，它就简单地忽略它。
3. 属于同一个流的所有分组必须具有相同的源地址、相同的目的地址、相同的通信量类别和相同的选项。

27.2.3 IPv4 首部和 IPv6 首部的比较

以下列出了 IPv4 和 IPv6 首部之间的比较。

- IPv6 取消了首部长度字段，因为在这个版本中首部长度是固定的。
- IPv6 取消了服务类型字段。通信量类别和流标号字段合在一起取代了服务类型字段的功能。
- IPv6 取消了总长度字段，取而代之的是有效载荷长度字段。
- IPv6 的基本首部中取消了标识、标志和偏移字段。这些字段被包含在分片扩展首部中。
- IPv6 将 TTL 字段称为跳数限制字段。
- 协议字段被下一个首部字段代替。
- 首部检验和取消了，因为检验和由上层协议提供，因此在这一层不需要。
- IPv4 的选项字段在 IPv6 中以扩展首部来实现。

27.2.4 扩展首部

基本首部的长度是固定的 40 字节，但是，要使 IP 数据报具有更多的功能，在基本首部的后面还可以增加最多六个扩展首部 (extension headers)。这些扩展首部中有很多本来就是 IPv4 的选项。图 27.3 给出了扩展首部的格式。

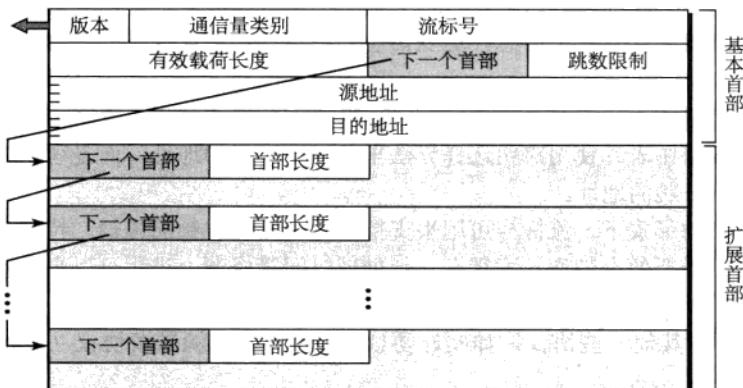


图 27.3 扩展首部的格式

已经定义的扩展首部有六种，它们分别是逐跳选项、源路由选择、分片、鉴别、加密的安全有效载荷和终点选项（见图 27.4）。



图 27.4 扩展首部的类型

逐跳选项

当源点需要把信息传递给数据报经过的所有路由器时，就需要使用逐跳选项 (hop-by-hop option)。例如，有可能需要将某些关于管理、排错或控制功能通知给一些路由器，或者，若数据报的长度超过了正常的 65 535 字节，那么所有路由器都必须知道这个信息。图 27.5 给出了逐跳选项首部的格式。其中第一个字段定义了出现在首部链中的下一个首部。首部长度字段定义了首部的字节数（包括下一个首部字段）。这个首部的其余部分则包含不同的选项。

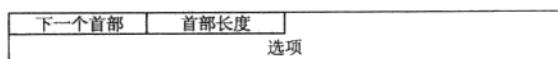


图 27.5 逐跳选项首部的格式

到目前为止只定义了三种逐跳选项：Pad1、PadN 和特大有效载荷。图 27.6 给出了这些选项的一般格式。

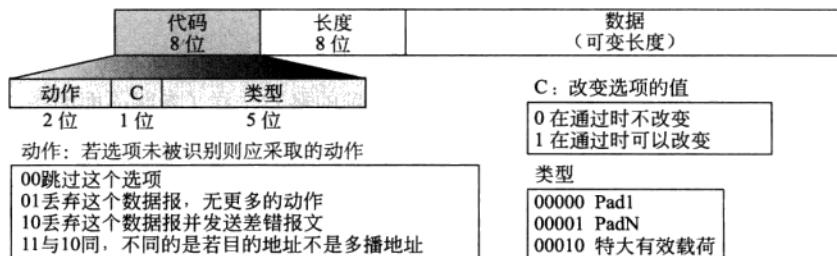


图 27.6 逐跳选项首部的选项格式

- **Pad1** 这个选项是 1 字节长，其作用是为了对齐。某些选项需要在 32 位字的特定位开始（见后面的特大有效载荷的描述）。如果选项正好差一个字节才能满足这个需求，则可加上 Pad1 来填补这个差额。Pad1 既不包含选项长度字段，也不包含选项数据字段。它仅包含了一个把所有位都置为 0 的选项代码字段（动作是 00，改变位是 0，类型是 00000）。Pad1 可在逐跳选项首部中的任何地方插入（见图 27.7）。

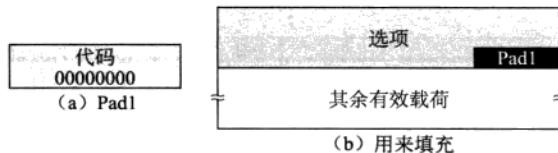


图 27.7 Pad1

- **PadN** PadN 在概念上与 Pad1 相似。不同的地方是当两个或更多字节需要对齐时就要使用 PadN。这个选项包括 1 字节的选项代码、1 字节的选项长度和可变数目的零填充字节。这个选项的代码值是 1（动作是 00，改变位是 0，类型是 00001）。选项长度包含的是填充字节的数目，见图 27.8。



图 27.8 PadN

- **特大有效载荷 (jumbo payload)** 我们已经讲过，IP 数据报的最大有效载荷长度是 65 535 字节。但是，如果由于某种原因需要使用更长的有效载荷，我们就可以使用特大有效载荷选项来定义这个更大的长度。特大有效载荷选项必须总是从扩展首部的最前端算起的 4 字节的倍数再加上两个字节的地方开始。也就是说特大有效载荷选项在 $(4n + 2)$ 字节处开始，这里的 n 是一个小整数，参见图 27.9。

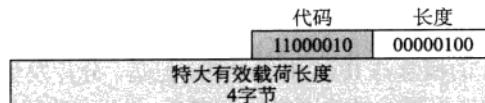


图 27.9 特大有效载荷

终点选项

终点选项 (destination option) 用于当源点需要将信息仅传递给终点时。中间的各路由器不允许读取这些信息。终点选项的格式与逐跳选项的格式一样 (参照前面的图 27.5)。到目前为止, 仅定义了 Pad1 和 PadN 选项。

源路由选择

源路由选择扩展首部把 IPv4 的严格源路由和不严格源路由的概念合并在一起。源路由选择扩展首部至少包含七个字段 (见图 27.10)。前两个字段 (下一个首部和首部长度) 与逐跳扩展首部的一样。类型字段定义了不严格的或严格的路由选择。剩余地址字段指出了要到达终点还需要的跳数。严格的/不严格的掩码字段确定路由选择的严格程度。若置为严格的, 则路由选择必须完全按照源点的指示进行。反之, 若掩码字段是不严格的, 则除了首部中规定的路由器外, 还可以访问其他的路由器。

下一个首部	首部长度	类型	剩余地址
保留		严格的/不严格的掩码	
		第一个地址	
		第二个地址	
		:	
		最后一个地址	

图 27.10 源路由选择

使用源路由选择时, 数据报的目的地址并没有遵循我们前面的定义 (即数据报的最后终点)。相反地, 它要从一个路由器到另一个路由器地发生变化。例如, 在图 27.11 中, 主机 A 希望以特定的路由把数据报发送给主机 B: 从 A 到 R1 到 R2 到 R3 到 B。请注意基本首部中的目的地址, 如果你认为它是固定不变的, 那就错了。事实上, 它在每一个路由器中都要发生变化。而且扩展首部中的地址也会随着从一个路由器到另一个路由器而发生变化。

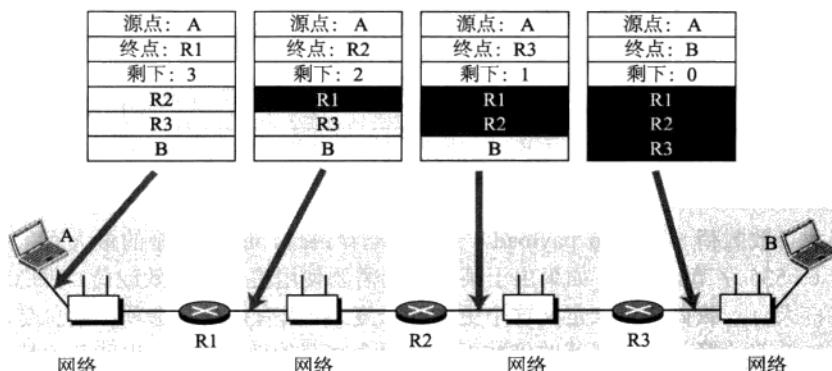


图 27.11 源路由选择的例子

分片

分片 (fragmentation) 的概念与 IPv4 中的一样。但是, 分片发生的地方却不同。在 IPv4 中, 若数据报长度超过数据报要经过的网络的 MTU, 则源点或路由器就要把它进行分片。在 IPv6 中, 只有最开始的源点才能进行分片。源点必须使用路径 MTU 发现技术 (Path MTU)

discovery technique) 来找出该路径上所有网络支持的最小的 MTU。然后源点再利用这个知识进行分片。

若源点不使用路径 MTU 发现技术，它就必须把数据报分片为 1280 字节或更短的长度。这是连接到因特网的每一个网络必须支持的 MTU 的最小值。图 27.12 给出了分片扩展首部的格式。

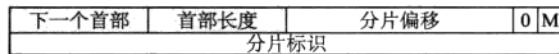


图 27.12 分片

鉴别

鉴别 (authentication) 扩展首部有双重作用：它证实报文的发送方并保证数据的完整性。我们需要前者，因为这样接收方就可以相信这个报文是来自真正的发送方，而不是来自冒名顶替者。我们需要后者，因为这样可以检查出数据在传输过程中有没有被黑客改动过。

鉴别扩展首部的格式如图 27.13 所示。安全参数索引字段定义了鉴别所使用的算法。鉴别数据字段包含由该算法产生的实际数据。我们将在第 29 章中讨论鉴别。

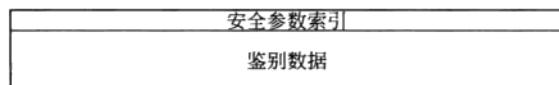


图 27.13 鉴别

鉴别可使用多种不同的算法。图 27.14 概述了计算鉴别数据字段的方法。发送方将 128 位的安全密钥，连同整个的 IP 数据报，再一次加上那个 128 位的安全密钥，一起传递给这个算法。在数据报中，那些在传输过程中会改变数值的字段（例如，跳数计数），则都被置为零。传递给算法的数据报包括了鉴别首部扩展本身，其中的鉴别数据字段则被置为零。这个算法产生一个鉴别数据，并在数据报传输之前把这个鉴别数据插入到扩展首部中。

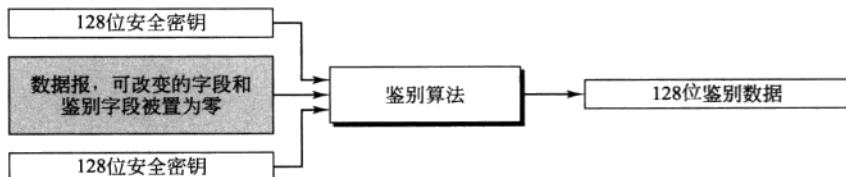


图 27.14 鉴别数据的计算

接收方以类似的手段进行处理。它将该安全密钥和接收到的数据报（同样地，把改变的字段都置为零），一起交给鉴别算法。若得到的结果与鉴别数据字段中的数据一致，则这个 IP 数据报是真实的，否则，就丢弃这个数据报。

加密的安全有效载荷

加密的安全有效载荷 (Encrypted Security Payload, ESP) 是提供了保密性并能防止窃听的一种扩展。图 27.15 给出了它的格式。安全参数索引字段是一个 32 位字，它定义所使用的加密/解密类型。其他字段包含的是被加密的数据以及算法所需的任何附加的参数。加密 (encryption) 可通过两种方法实现：运输方式和隧道方式，我们在第 30 章讨论 IPSec 时会详细介绍。

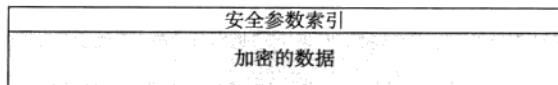


图 27.15 加密的安全有效载荷

27.2.5 IPv4 和 IPv6 的比较

下面简单扼要地对 IPv4 中的选项和 IPv6 中的选项（作为扩展首部）之间进行了比较。

- 在 IPv4 中的无操作和选项结束选项，在 IPv6 中被替换成 Pad1 和 PadN 选项。
- 在 IPv6 中没有实现记录路由选项，因为它未被使用。
- 在 IPv6 中没有实现时间戳选项，因为它未被使用。
- 源路由选项在 IPv6 中称为源路由扩展首部。
- 分片字段是 IPv4 基本首部中的一部分，但在 IPv6 中已经移到分片扩展首部。
- 在 IPv6 中的鉴别扩展首部是新的。
- 在 IPv6 中的加密安全有效载荷首部是新的。

27.3 从 IPv4 过渡到 IPv6

由于因特网上的系统数量非常之庞大，因此从 IPv4 过渡到 IPv6 不可能突然发生。要使因特网上的每一个系统从 IPv4 过渡到 IPv6，

需要花费相当长的时间。过渡必须是平稳的，以防止在 IPv4 和 IPv6 的系统之间出现任何问题。IETF 已经设计了三种策略来帮助从 IPv4 到 IPv6 的过渡（见图 27.16）。



图 27.16 三种过渡策略

27.3.1 双协议栈

根据 IETF 的推荐，所有主机在完全过渡到版本 6 之前都要使用双协议栈（dual stack）。换言之，一个站点必须同时运行 IPv4 和 IPv6，直到整个的因特网都使用 IPv6。图 27.17 给出了双协议栈的配置的概要图。

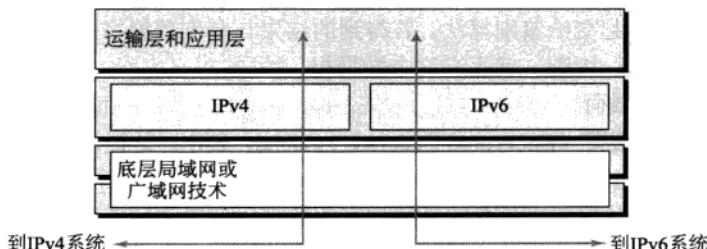


图 27.17 双协议栈

在向终点发送一个分组时，为了确定应使用哪一个版本，源主机要向 DNS 查询。若 DNS 返回 IPv4 地址，源主机就发送 IPv4 分组。若 DNS 返回 IPv6 地址，源主机就发送 IPv6 分组。

27.3.2 隧道技术

当两个使用了 IPv6 的计算机要互相通信，但分组又需要通过使用 IPv4 的区域时，就要使用隧道技术（tunneling）策略。要通过 IPv4 区域，分组必须具有 IPv4 地址，因此当进入这个区域时，IPv6 分组要封装成 IPv4 分组，而当分组离开这个区域时，再拆掉这个封装。这就好像是 IPv6 分组从隧道的一端进入，又从另一端出来。为了清楚地表示 IPv4 分组所携带的是 IPv6 分组，它的协议值被置为 41。隧道技术如图 27.18 所示。



图 27.18 隧道技术策略

27.3.3 首部转换

当因特网的大部分已经过渡到 IPv6，但某些系统仍使用 IPv4 时，就必须进行首部转换（header translation）。发送方希望使用 IPv6，但接收方不能识别 IPv6。这种情况不能使用隧道技术，因为这个分组必须是 IPv4 格式的才能被接收方识别。在这种情况下，首部格式必须通过首部转换彻底地改变过来。IPv6 分组的首部被转换为 IPv4 首部（见图 27.19）。

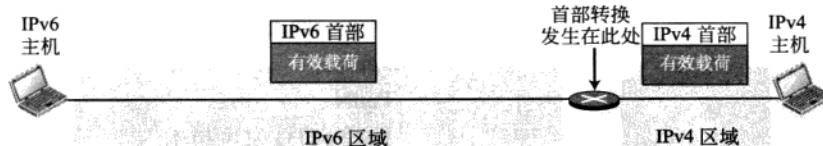


图 27.19 首部转换策略

首部转换用映射的地址把 IPv6 地址转换为 IPv4 地址。以下列出了把 IPv6 分组首部转换为 IPv4 分组首部所使用的一些规则。

- 用提取最右边 32 位的方法，把 IPv6 映射的地址转换为 IPv4 地址。
- IPv6 的通信量类别字段值被丢弃。
- IPv4 的服务类型字段值置为零。
- 计算 IPv4 的检验和，并插入到相应的字段中。

- 忽略 IPv6 的流标号。
- 兼容的扩展首部要转换成选项，并插入到 IPv4 的首部中。某些选项可能会被丢弃。
- 计算出 IPv4 首部的长度，把它插入到相应的字段中。
- 计算出 IPv4 分组的总长度，把它插入到相应的字段中。

27.4 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

27.4.1 参考书

有不少书籍全面地介绍了 IPv6。我们推荐[Com 06]、[Los 04]和[Tan 03]。

27.4.2 RFC

通过一些 RFC 可以看出 IPv6 的不断更新过程，包括 RFC 2460、RFC 2461 和 RFC 2462。

27.5 重要术语

关注的批量数据通信量	首部转换
鉴别	跳数限制
后台数据	逐跳选项
基本首部	网际协议，版本 6 (IPv6)
终点选项	特大有效载荷
双协议栈	下一个首部
加密的安全有效载荷 (ESP)	Pad1
加密	PadN
扩展首部	路径 MTU 发现技术
流标号	隧道技术
分片	不关注的数据通信量

27.6 本章小结

- IPv6 数据报由基本首部和有效载荷组成。40 字节的基本首部包括版本、通信量类别、流标号、有效载荷长度、下一个首部、跳数限制、源地址以及目的地址等字段。

通信量类别字段用来衡量数据报的重要性。流标号标志了分组序列的特殊处理的需求。

- 有效载荷包括可选的扩展首部以及从上层传来的数据。扩展首部为 IPv6 数据报增加了更多的功能。逐跳选项用来把信息传递给路径上的所有路由器。源路由选择扩展用在源点希望指明传输路径时。分片扩展用于有效载荷是报文的一个分片时。鉴别扩展证实了报文的发送方，并保护数据不受黑客篡改。加密的安全有效载荷扩展为发送方和接收方之间提供了保密性。终点扩展专门把信息从源点传递到终点。
- 从版本 4 过渡到版本 6 使用的三个策略分别是双协议栈、隧道技术和首部转换。

27.7 实践安排

27.7.1 习题

1. 某 IPv6 分组由基本首部和一个 TCP 报文段组成。数据长度为 320 字节。试描绘这个分组并给出每一个字段的值。
2. 某 IPv6 分组由基本首部和 TCP 报文段组成。数据长度为 128 000 字节（特大有效载荷）。试描绘这个分组并给出每一个字段的值。

27.7.2 研究活动

3. 试找出为什么在 IPv6 中有两种安全协议（AH 和 ESP）？

第 28 章 ICMPv6

本章是第五部分的最后一章。在第 26 章讨论了 IPv6 地址和第 27 章讨论了 IPv6 数据报之后，我们在本章要介绍的是 ICMPv6 协议。ICMPv6 是曾经讨论过的 IPv4 中的三个协议的合并：ICMP、IGMP 和 ARP。我们先要介绍 ICMPv6 的概念并将其报文划分为四个大类，然后再简单地讨论每一类中的各个报文。

目标

本章有以下几个目标：

- 介绍 ICMPv6，并将其与 ICMPv4 进行对比。
- 讨论 ICMPv6 中的差错报文，并将其与 ICMPv4 中的差错报文进行对比。
- 讨论 ICMPv6 中的信息报文，并将其与 ICMPv4 中的信息报文进行对比。
- 讨论作为 ND 和 IND 协议一部分的邻站发现报文。
- 讨论作为 MLDv2 协议一部分的组成员关系报文。

28.1 引言

在 TCP/IP 协议族的版本 6 中被修改的另一个协议是 ICMP。新的版本称为网际控制报文协议版本 6 (Internet Control Message Protocol version 6, ICMPv6)，它延续了版本 4 的策略和目标。但是 ICMPv6 比 ICMPv4 要复杂得多：在版本 4 中的一些独立的协议现在已成为 ICMPv6 中的一部分，并且还增加了一些新的报文，使得 ICMPv6 能发挥更大的作用。图 28.1 比较了网络层的版本 4 和版本 6。版本 4 中的 ICMP、ARP 和 IGMP 协议被合并成了一个协议，就是 ICMPv6。

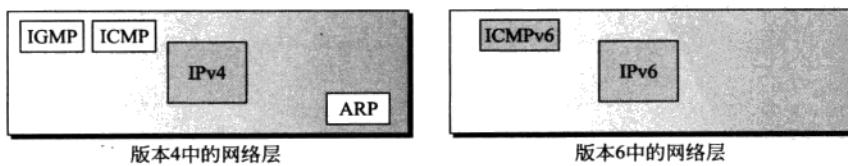


图 28.1 版本 4 和版本 6 网络层比较

与 ICMPv4 一样，ICMPv6 是面向报文的协议，它利用报文来报告差错、获取信息、探测邻站或管理多播通信。但是在 ICMPv6 中还增加了几个定义报文的功能及含义的其他协

议。在对 ICMPv6 报文进行归类时，不同的文献和 RFC 使用了不同的策略，它们把其中的一些报文定义为 ICMPv6 报文，而把另一些报文定义为 ND 报文或 MLD 报文。我们认为所有这些报文都是 ICMPv6 报文，但是根据这些报文的功能和作用的不同，我们把这些报文划分为不同的类别。在介绍每一类报文时，我们会提到并描述相应的新增加的协议，它们定义了报文的功能及含义。我们使用这种分类方法的原因在于所有这些报文都具有相同的格式，并且所有报文类型都由 ICMPv6 协议处理。我们认为像 ND 和 MLD 这样的协议都是运行在 ICMPv6 协议之下的。基于此种考虑，我们定义的 ICMPv6 报文分类如图 28.2 所示。

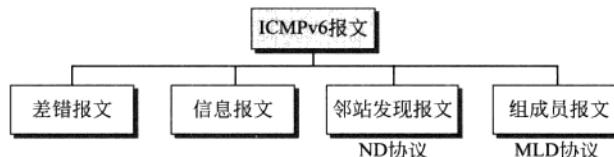


图 28.2 ICMPv6 报文的分类

从图中可以看出，其中有两个组的报文是在 ND 协议和 MLD 协议的控制下进行发送和接收的。

28.2 差错报文

如我们在讨论版本 4 时所看到的，ICMP 的主要责任之一就是对差错进行报告。有四种类型的差错可以处理：终点不可达、分组太大、超时和参数问题（参见图 28.3）。请注意，在版本 4 中用于拥塞控制的源抑制报文在这个版本中已被取消，因为 IPv6 认为拥塞控制应当由优先级和流标号字段来负责。改变路由报文也从差错报告类转移到邻站发现类，因而我们把它作为一种邻站发现报文来讨论。



图 28.3 差错报告报文

ICMPv6 生成差错分组，再封装成一个 IPv6 数据报，然后交付给出差错的数据报的源点。

28.2.1 终点不可达报文

终点不可达报文的概念和我们在 ICMPv4 中描述的完全一样。当路由器无法转发一个数据报，或者一个主机无法将数据报的内容交付给上层协议时，该路由器或主机就丢弃数据报并向源主机发送一个终点不可达差错报文。图 28.4 给出了终点不可达报文的格式。

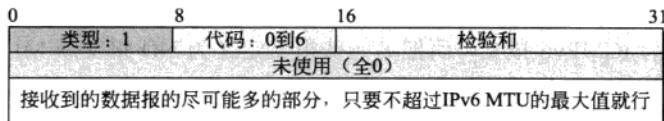


图 28.4 终点不可达报文

这种类型的代码字段指明丢弃这个数据报的原因，并精确说明传送失败的原因：

- 代码 0** 没有路径到达终点。
- 代码 1** 与终点的通信在管理上被禁止。
- 代码 2** 超出源地址的范围。
- 代码 3** 目的地址不可达。
- 代码 4** 端口不可达。
- 代码 5** 源地址失败（因过滤策略）。
- 代码 6** 拒绝转发到终点。

28.2.2 分组太大报文

这是新增加到版本 6 中的一个报文类型。因为 IPv6 不允许在路由器上进行分片，如果路由器收到的数据报大于它必须通过的网络的最大传输单元（MTU），那么就会发生两件事。首先，路由器要丢弃这个数据报，然后，要向源点发送一个 ICMP 差错分组——**分组太大报文**（packet-too-big message）。图 28.5 给出了这个分组的格式。请注意，它只有一种代码（0），而 MTU 字段告诉发送方该网络能接受的最大分组长度。

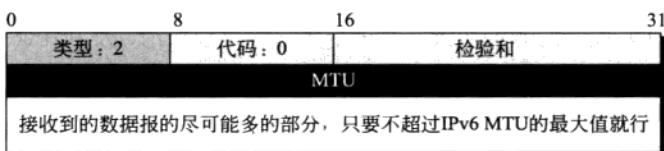


图 28.5 分组太大报文

28.2.3 超时报文

正如我们在第 9 章讨论的，超时差错报文的产生有两种情况：当寿命的值变成 0 时，以及当一个数据报的某些分片未能在限期内到达时。版本 6 的超时报文在格式上与版本 4 的相似。唯一的区别是类型值变为 3。图 28.6 给出了超时报文的格式。

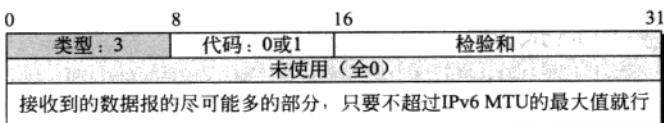


图 28.6 超时报文

和版本 4 一样，当数据报由于跳数限制字段的值为零而被路由器丢弃时，就使用代码 0。当因为在限期内还有其他的分片没有到达而将数据报的分片丢弃时，就使用代码 1。

28.2.4 参数问题报文

正如我们在第 9 章中讨论的，当一个数据报在因特网中传输时，其首部中的任何含义不清之处都会产生严重的问题。如果路由器或终点主机发现有任何字段含糊不清或者缺少必要的值，那么它就会丢弃该数据报，并向源点发送一个参数问题差错报文。这个报文在 ICMPv6 中与版本 4 的相应报文相似。不过，类型值已变为 4，并且偏移量指针字段的长度增加到 4 字节。它还有三个不同代码，而不是两个。图 28.7 所示为参数问题报文的格式。

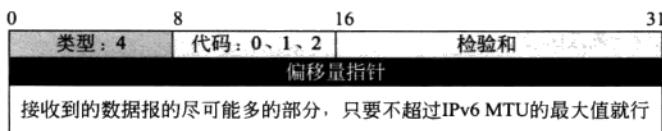


图 28.7 参数问题报文

代码字段指明丢弃数据报的原因和传送失败的原因：

- 代码 0** 有出错的首部字段。
- 代码 1** 不能识别的下一个首部类型。
- 代码 2** 不能识别的 IPv6 选项。

28.3 信息报文

有两种 ICMPv6 报文可被归类为信息报文：回送请求报文和回送回答报文。正如我们在第 9 章中讨论的，回送请求和回送回答报文被设计用于检测因特网上的两个设备之间是否能够互相通信。主机或路由器可以向另一个主机发送回送请求报文，而收到该请求的计算机或路由器则用回送回答报文进行响应。

28.3.1 回送请求报文

回送请求报文的格式和思路与版本 4 中的一样。唯一的区别就是类型值不同，如图 28.8 所示。

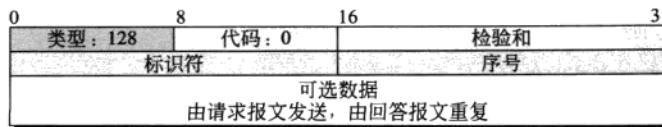


图 28.8 回送请求报文

28.3.2 回送回答报文

回送回答报文的格式和思路也与版本 4 中的一样。唯一的区别就是类型值不同，如图 28.9 所示。

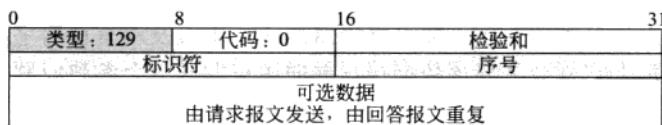


图 28.9 回送回答报文

28.4 邻站发现报文

ICMPv4 中的有些报文在 ICMPv6 中被重新定义，以处理有关邻站发现的内容。同时还增加了几个新的报文以提供更多的功能。其中最重要的是定义了两个新协议：邻站发现 (Neighbor-Discovery, ND) 协议和反向邻站发现 (Inverse-Neighbor-Discovery, IND) 协议，它们明确地定义了这几组报文的功能。这两个协议由位于相同链路（网络）上的结点（主机或路由器）使用，有以下三个主要目的：

1. 主机使用 ND 协议来发现能够为它们转发分组的邻居路由器。
2. 结点使用 ND 协议来发现邻站的链路层地址（相连在同一个网络上的结点）。
3. 结点使用 IND 协议来发现邻站的 IPv6 地址。

28.4.1 路由器询问报文

路由器询问报文背后的思想与版本 4 中的一样。主机使用路由器询问报文来发现网络中能够为该主机转发 IPv6 报文的路由器。到目前为止，为这种报文定义的唯一选项是在报文中包含主机的物理（数据链路层）地址，为路由器在响应时提供方便。这种报文的格式如图 28.10 所示，报文的类型是 133。

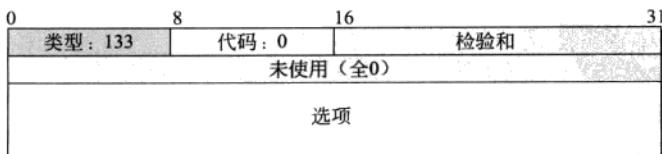


图 28.10 路由器询问报文

28.4.2 路由器通告报文

路由器通告报文是一个路由器为了响应路由器询问报文而发送的。图 28.11 所示为路

由器通告报文的格式。

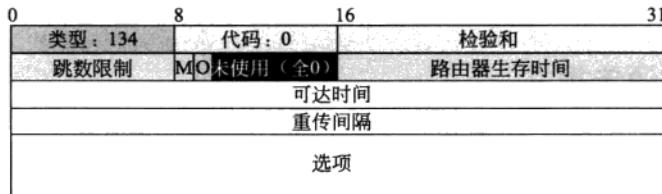


图 28.11 路由器通告报文

其中的字段说明如下：

- **跳数限制** 这个 8 位字段给出的跳数限制就是请求者应当在自己的 IPv6 数据报中使用的跳数限制的值。
- **M** 这个 1 位字段是“管理地址配置”字段。当 M 位被置 1 时，主机需要使用管理配置。
- **O** 这个 1 位字段是“其他地址配置”字段。当 O 位被置 1 时，主机需要使用适当的协议来配置。
- **路由器寿命** 这个 16 位字段定义了该路由器作为默认路由器的寿命(以秒为单位)。当这个字段的值为 0 时，表示该路由器不能作为默认路由器。
- **可达时间** 这个 32 位字段定义了该路由器可达的时间期限(以秒为单位)。
- **重传间隔** 这个 32 位字段定义了重传间隔(以秒为单位)。
- **选项** 一些允许的选项包括：此报文发送时所经过链路的链路层地址、该链路的 MTU 以及地址前缀信息。

28.4.3 邻站询问报文

图 28.12 所示为邻站询问报文 (neighbor-solicitation message) 的格式。如前所述，版本 4 的网络层包含一个独立的协议，称为 ARP。在版本 6 中，这个协议取消了，而它的任务并入到 ICMPv6 中。邻站询问报文与 ARP 请求报文的任务相同。当主机或路由器有一个报文需要发送给某个邻站时就要发送这个报文。报文的发送方知道接收方的 IP 地址，但是还需要接收方的数据链路地址。数据链路地址是 IP 数据报在封装成为帧时所必需的。它的

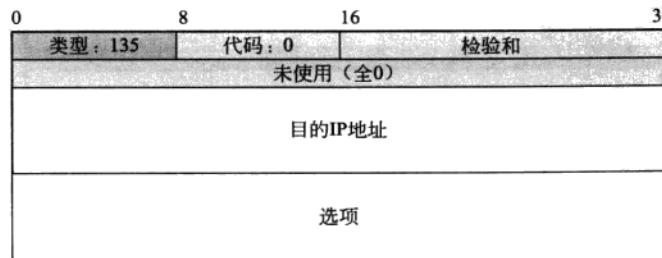


图 28.12 邻站询问报文

唯一选项宣布了发送方的数据链路地址，为接收方在响应时提供方便。接收方可以利用这个发送方的数据链路地址来进行单播响应。

28.4.4 邻站通告报文

邻站通告报文（neighbor-advertisement message）是在响应邻站询问报文时发送的。它相当于 IPv4 中的 ARP 回答报文。图 28.13 所示为这个报文的格式。

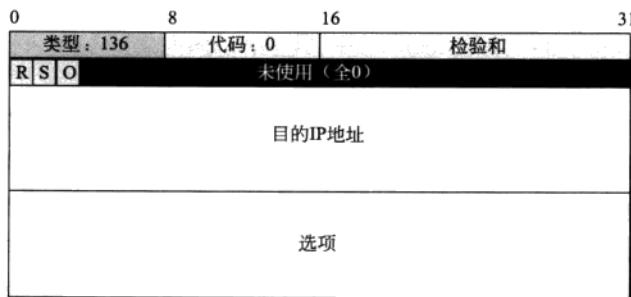


图 28.13 邻站通告报文

其中的字段说明如下：

- R** 这个 1 位字段是“路由器”标志。当它被置 1 时，就表示这个报文的发送方是路由器。
- S** 这个 1 位字段是“询问”标志。当它被置 1 时，就表示发送方是为了响应邻站询问报文而发送的这个通告。主机或路由器也可以在没有询问的情况下发送通告。
- O** 这个 1 位字段是“覆盖”标志。当它被置 1 时，就表示这个通告应当覆盖高速缓存中已存在的信息。
- 选项** 唯一允许的选项是通告方的数据链路地址。

28.4.5 改变路由报文

改变路由报文的作用和我们在版本 4 中讨论的一样。但是，为了适应版本 6 中的 IP 地址长度，这个分组的格式改变了。此外，还增加了一个选项，以便让主机知道目的路由器的物理地址（见图 28.14）。

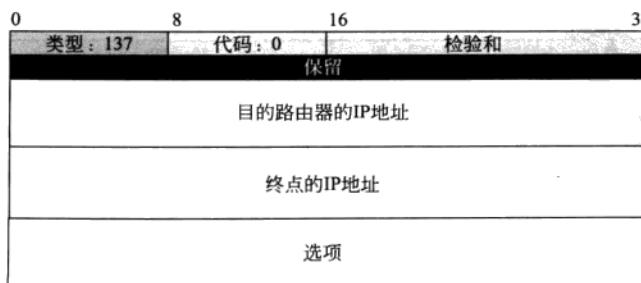


图 28.14 改变路由报文

其中允许的选项包括：发送方的数据链路地址以及改变路由的 IP 数据报首部的一部分内容，只要这个报文的总长度不超过 MTU 就行。

28.4.6 反向邻站询问报文

当结点知道邻站的链路层地址，但是却不知道该邻站的 IP 地址时，这个结点就发送反向邻站询问报文（inverse-neighbor-solicitation message）。这个报文封装在目的地址为全结点多播地址的 IPv6 数据报中。发送方必须在选项字段中发送以下两个信息：它的链路层地址和目的结点的链路层地址。发送方也可以包括自己的 IP 地址和链路的 MTU 值。图 28.15 所示为该报文的格式。

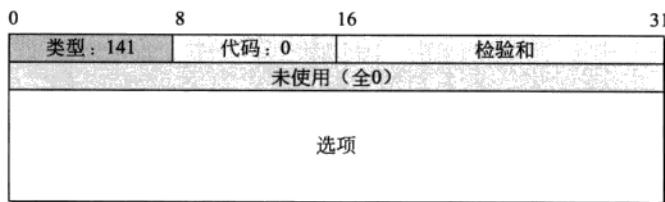


图 28.15 反向邻站询问报文

28.4.7 反向邻站通告报文

反向邻站通告报文（inverse-neighbor-advertisement message）是在响应反向邻站询问报文时发送。这个报文的发送方必须在选项字段中包括发送方的链路层地址和目的结点的链路层地址。图 28.16 所示为这个报文的格式。

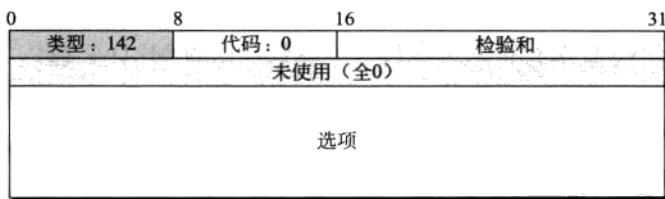


图 28.16 反向邻站通告报文

28.5 组成员关系报文

正如我们在第 12 章所讨论的一样，在 IPv4 中，管理多播交付的任务是由 IGMPv3 协议完成的。而在 IPv6 中，相应的责任被转交给了多播监听交付（Multicast Listener Delivery, MLD）协议。MLDv1 相当于 IGMPv2，而 MLDv2 则相当于 IGMPv3。这一小节的讨论用到的素材全部来自于 RFC 3810。它的思想与我们讨论的 IGMPv3 一样，只是报文的大小和格式经改变后能够适应 IPv6 的较大的多播地址长度。与 IGMPv3 一样，

MLDv2 有两种类型的报文：成员关系查询报文和成员关系报告报文。前一类报文还可以再划分为三个子类：通用的（general）、特定组的（group-specific）以及特定组与源点的（group-and-source specific）。

28.5.1 成员关系查询报文

成员关系查询报文是由路由器发送的，目的是为了找出网络中活跃的组成员。图 28.17 所示为此类报文的格式。

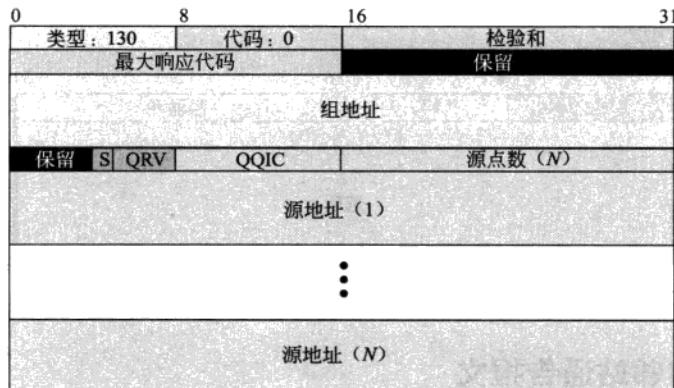


图 28.17 成员关系查询报文格式

其中的字段几乎都与 IGMPv3 中的字段相同，除了多播地址和源地址的长度从 32 位增加到 128 位。另一个长度变化比较明显的字段是最大响应代码字段，它的长度从 8 位增加到 16 位。稍后我们再讨论这个字段。还要注意的是这个报文的前 8 个字节与其他 ICMPv6 分组在格式上保持了一致性，因为 MLDv2 被认为是 ICMPv6 的一部分。

28.5.2 成员关系报告报文

图 28.18 所示为成员关系报告报文的格式。

请注意，MLDv2 的成员关系报告报文的格式与 IGMPv3 中的完全一样，除了字段的长度会因为地址长度的变化而改变之外。特别地，记录的类型与 IGMPv3 中定义的完全一致（类型 1~6）。

28.5.3 功能性

MLDv2 的工作方式与 IGMPv3 基本相同。不过，我们在这里还是要简单讨论一下它们之间的几个不同之处。

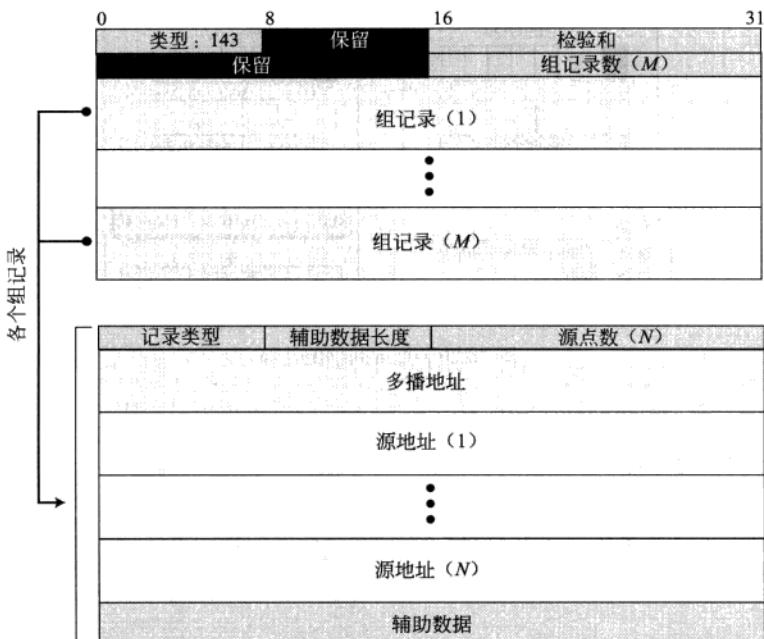


图 28.18 成员关系报告报文格式

最大响应时延的计算

如我们前面提到的，MLDv2 的最大响应代码字段的长度是 IGMPv3 中相应字段长度的两倍。基于这个原因，在这个协议中对最大响应时间的计算会略有不同，如图 28.19 所示。

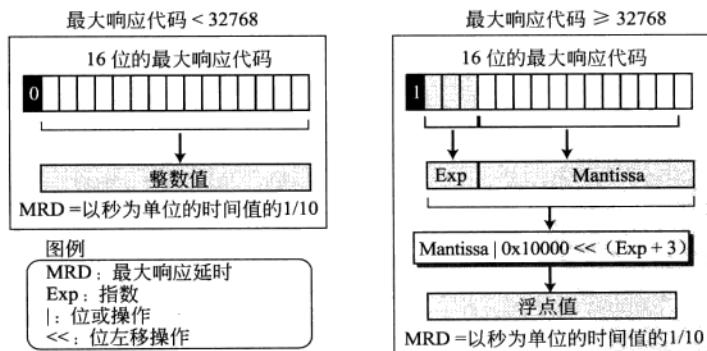


图 28.19 最大响应时间的计算

查询间隔的计算

查询间隔的计算采用与计算最大响应时延同样的过程，它的值是根据 QQIC 字段的值计算得到的，如图 28.20 所示。

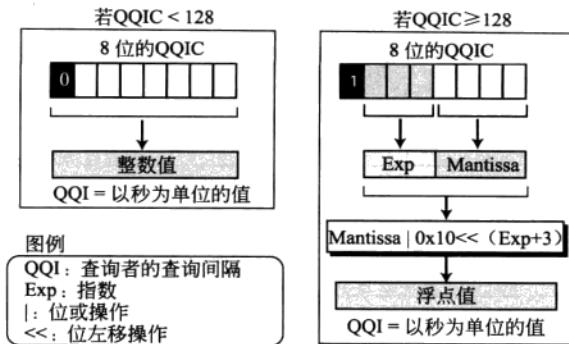


图 28.20 查询间隔的计算

28.6 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍和 RFC。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。

28.6.1 参考书

有不少书籍对 ICMPv6 有所介绍。我们推荐[Com 06]、[Los 04]和[Koz 05]。

28.6.2 RFC

通过一些 RFC 可以看出 ICMPv6 的不断更新过程，包括 RFC 2461、RFC 2894、RFC 3122、RFC 3810、RFC 4443 和 RFC 4620。

28.7 重要术语

因特网控制报文协议版本 6 (ICMPv6)
反向邻站发现 (IND) 协议
反向邻站通告报文
反向邻站询问报文
多播监听交付协议

邻站发现 (ND) 协议
邻站通告报文
邻站询问报文
分组太大报文

28.8 本章小结

□ ICMPv6 和 ICMPv4 一样，是一个面向报文的协议。它用于报告差错，获取信息，

探测邻站或管理多播通信。但是在 ICMPv6 中还增加了几个定义报文的功能及含义的其他协议。

- 我们把 ICMPv6 中的所有报文划分为四大类：差错报文、信息报文、邻站发现报文和组成员关系报文。
- 一共讨论了四种类型的差错报文：终点不可达、分组太大、超时和参数问题。
- 一共讨论了两种类型的信息报文：回送请求报文和回送回答报文。
- 我们讨论了七种类型的邻站发现报文。前五种报文（路由器询问、路由器通告、邻站询问、邻站通告和改变路由）都在 ND 协议的控制下。后两种报文（反向邻站询问和反向邻站通告）是在 IND 协议的控制之下。
- 我们讨论了两种组管理报文：成员关系查询和成员关系报告。它们都在 MLDv2 协议的控制下。

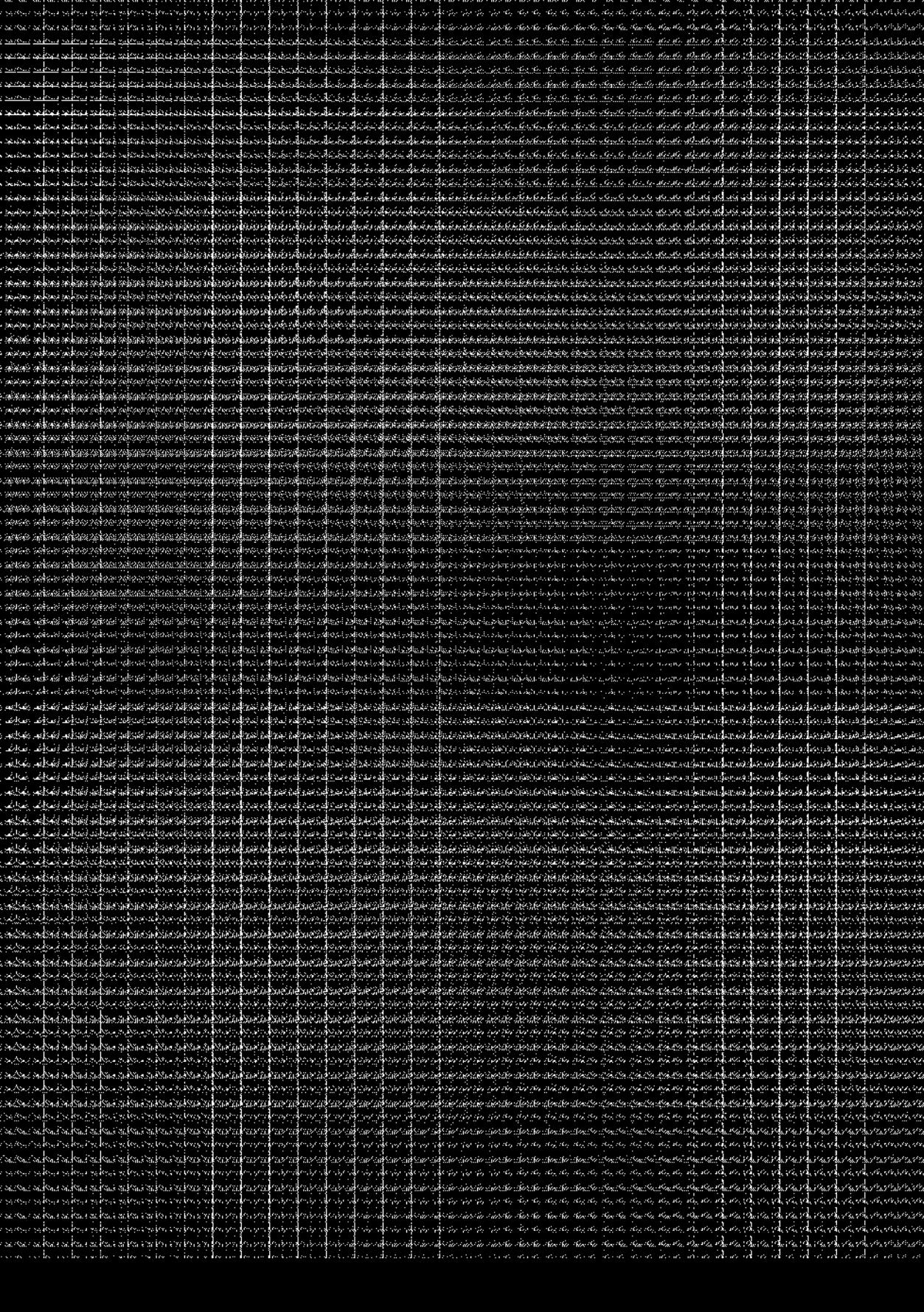
28.9 实践安排

28.9.1 习题

1. 哪些类型的 ICMP 报文包含了 IP 数据报的一部分？为什么要包含这部分？
2. 通过一张表对 ICMPv6 的差错报告报文和 ICMPv4 的差错报告报文进行比较。
3. 通过一张表来对 ICMPv6 的信息报文和 ICMPv4 的信息报文进行比较。
4. 通过一张表来对 ICMPv6 的邻站发现报文和 ICMPv4 中相应的报文进行比较。
5. 通过一张表来对 ICMPv6 的反向邻站发现报文和 ICMPv4 中相应的报文进行比较。
6. 通过一张表来对 ICMPv6 的组成员关系报文和 ICMPv4 中相应的报文进行比较。
7. 试计算以下各个最大响应代码值对应的最大响应时延，以秒为单位（参见图 28.19）。
 - a. 22000
 - b. 43000
8. 试计算以下各个 QQIC 值所对应的 QCI 的值（参见图 28.20）。
 - a. 78
 - b. 202

28.9.2 研究活动

9. 通过 RFC 3810 和 RFC 4604 更多地了解 MLDv2。
10. 通过 RFC 2461 更多地了解 ND 协议的作用。
11. 通过 RFC 3122 更多地了解 IND 协议的作用。
12. 通过 RFC 2894 更多地了解 IPv6 的路由器重新编号技术。

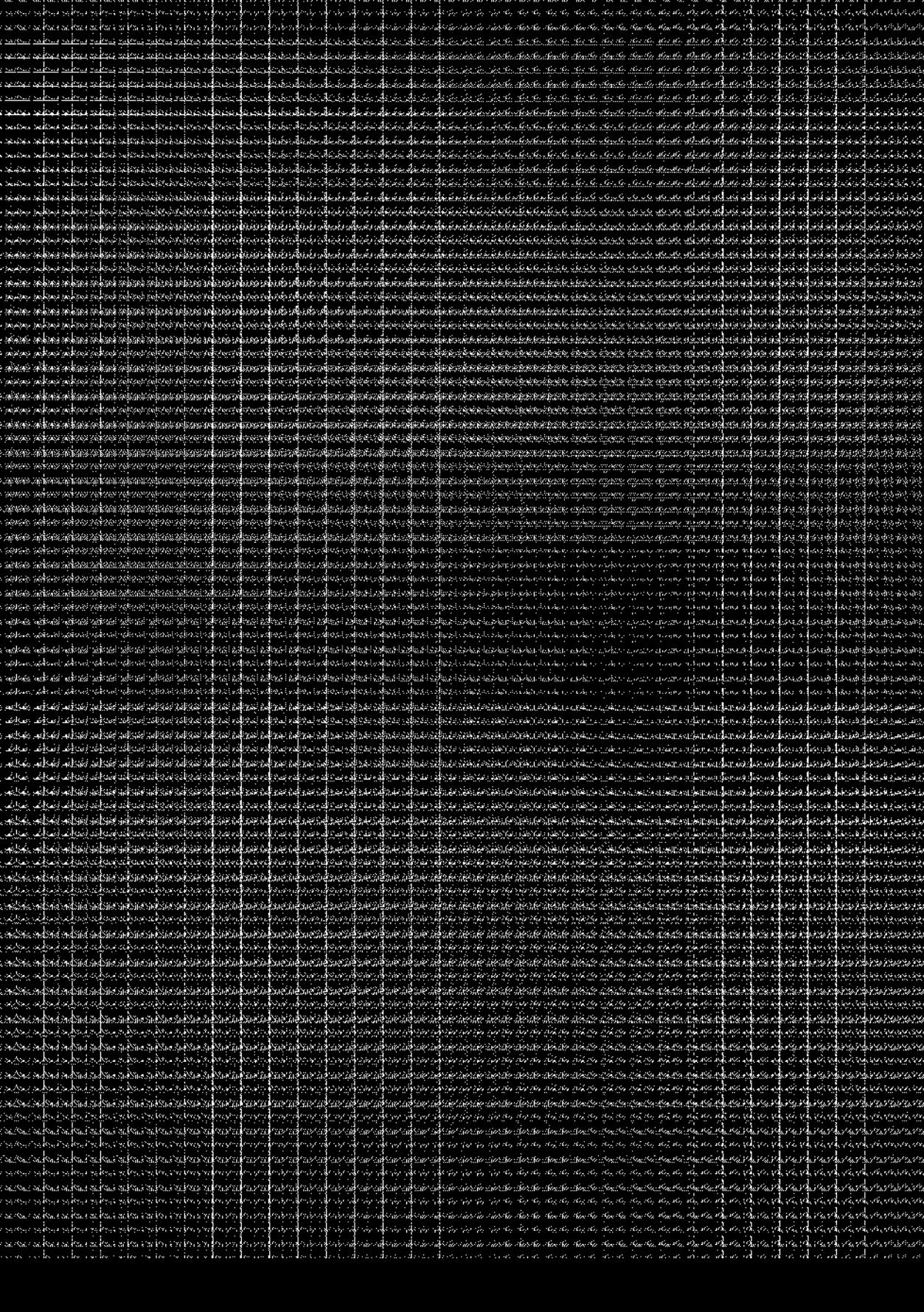


第六部分

安 全 性

第 29 章 加密术和网络安全

第 30 章 因特网安全



第 29 章 加密术和网络安全

加 密术和网络安全是一个非常广泛的话题，涉及到许多数学的特定领域，譬如数论。本章我们试图对这个话题做浅显易懂的介绍，目的是为下一章讨论因特网安全做好背景知识的准备。我们的目标是在不考虑其背后隐含的数学细节的前提下，简单地讨论与加密术和网络安全相关的一般内容。

目标

本章有以下几个目标：

- 介绍安全的目标，并讨论威胁安全目标的攻击类型。
- 介绍传统的加密方法如对称密钥加密方法 (symmetric-key cipher)，为理解现代对称密钥加密方法准备好背景知识。
- 介绍现代的块加密方法 (block cipher) 的组成元素，并举例说明这些元素在现代块加密方法中的使用。
- 讨论不对称密钥加密方法隐含的总体思想，并介绍一种最常见的不对称密钥加密方法。
- 讨论报文完整性，并说明如何通过使用加密散列函数来产生报文摘要。
- 介绍报文鉴别的思想，并说明通过报文摘要和密钥两者的结合是如何能够鉴别发送方的。
- 说明电子签名的思想是如何通过私钥-公钥对的使用而被应用到报文鉴别中的。
- 介绍实体鉴别的概念，并展示了一些简单的实体鉴别机制，它们或者使用一个密钥，或者使用一对私钥-公钥。
- 说明如何利用 KDC 或认证管理机构 (CA) 来分配和管理对称密钥加密中的密钥以及不对称密钥加密中的公钥。

29.1 引言

我们生活在信息时代。我们需要保存生活中各个方面信息。换言之，信息已成为一种财产，就像其他财产一样具有价值。作为一种财产，信息必须被保护起来以免受到攻击。要想保证安全，信息需要对未授权的访问来说是隐蔽的（保密性）；对未授权的修改来说是被保护的（完整性）；而对授权的实体来说，只要需要就能得到（有效性）。

在最近三四十年中，计算机网络为信息的应用带来了翻天覆地的变化。信息现在已经是分布式的。授权的用户可以通过计算机网络远距离地发送和读取信息。虽然上述三个要求（保密性、完整性和有效性）从来都不曾改变，但是如今它们的含义又有了新的外延。信息不仅在存储时要求保密，当它从一台计算机传输到另一台计算机时，也应当有某种方式来保护它的安全。

在这一小节中，我们首先讨论信息安全的三个主要目标，然后了解一下各种攻击是如何威胁这三个目标的，接着，讨论与这些安全目标相关的安全服务，最后，我们要定义两种用以实现安全目标和防卫攻击的技术。

29.1.1 安全的目标

让我们首先来讨论安全的三个目标：保密性、完整性和有效性。

保密性

保密性（*confidentiality*）可能算得上是信息安全的最基本的要求。我们需要保护自己的保密信息。而一个组织则需要防御那些威胁到它的信息保密性的恶意行为。保密性不仅要应用于信息的存储，还要应用于信息的传输。当我们发送一份即将保存到远程计算机上的信息时，或者当我们从远程计算机上读取一份信息时，也要为其在传输期间保守秘密。

完整性

信息需要不断地更新。在银行，当客户存款、取款时，她的账户收支平衡就要发生改变。完整性（*integrity*）的意思是信息的改变只能由授权的实体通过授权的机制来完成。完整性的破坏不一定都是恶意行为造成的：系统的一次偶然中断也可能对某些信息带来意料之外的改变，譬如一次电源故障。

有效性

信息安全的第三个要素是有效性（*availability*）。由某个组织产生及保存的信息应当对授权的实体是可用的。如果信息不可用，那么它就是没有用的信息。信息需要不断地更新，也就是说它必须让授权的实体能够访问得到。对一个组织来说，信息的不可用与缺少保密性或完整性一样糟糕。想象一下，如果客户无法访问自己的账户进行交易会发什么情况。

29.1.2 攻击

我们的三个安全目标（保密性、完整性、有效性）可能会受到安全攻击（*attack*）的威胁。虽然在文献中采用了不同的方法对这些攻击进行归类，但我们根据对应的安全目标将它们分为三组。图 29.1 所示为攻击的分类。

威胁保密性的攻击

一般来说，有两种类型的攻击会威胁到信息的保密性：窃取（*snooping*）和通信量分析（*traffic analysis*）。

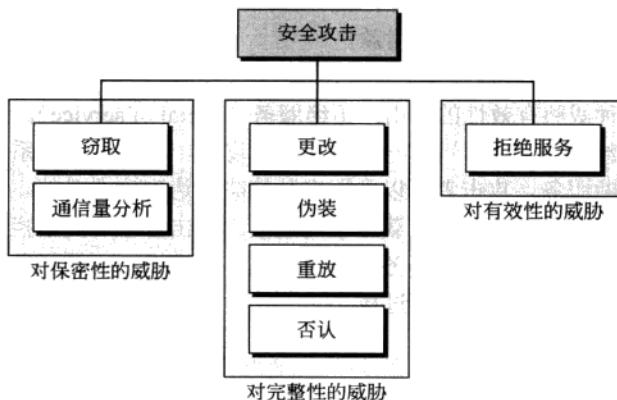


图 29.1 与安全目标关联的攻击分类

窃取 窃取指的是对数据未授权地访问或拦截。例如，通过因特网传送的某文件中可能包含了保密信息，未授权的实体可能会拦截此次传输，并利用该文件的内容使自己受益。要防止窃取，数据必须通过使用加密技术让拦截者无法辨识。

通信量分析 虽然对数据加密能够使拦截者无法辨识数据，但她还是能够通过监视在线通信量的方法来获得其他方面的一些信息。例如，她可以找出发送者或接收者的电子地址（如电子邮件地址），也能通过搜集请求-响应的信息来帮她猜出此次事务本身的某些特点。

威胁完整性的攻击

数据的完整性可能会受到多种类型的攻击：**更改** (modification)、**伪装** (masquerading)、**重放** (replaying) 和**否认** (repudiation)。

更改 在截获并读出信息之后，攻击者可以更改该信息，使之变得对自己有利。例如，用户向银行发送报文进行某种交易。攻击者截获了这个报文并改变交易的类型，使之对自己有利。请注意，有时攻击者只是简单地删除报文或者使报文延迟就能破坏系统或从中获益。

伪装 当攻击者假装自己是另外一个人时，就称为伪装或假冒。例如，攻击者可能窃取了某个银行客户的银行卡以及 PIN (个人身份号码)，并且假装自己就是那个客户。有时攻击者也可以把自己伪装成接收实体。例如，用户试图联系银行，但是另一个站点假装它就是那个银行，并从用户那里获取一些信息。

重放 重放是另一种形式的攻击。攻击者截获用户发送的报文，并在此后的某个时间试图重放这个报文。例如，某人向银行发送了请求，要求银行向攻击者支付一笔报酬，因为攻击者为他做了某件事。攻击者截获该报文，并再次发送这个报文，以便从银行重复领取这笔钱。

否认 这种类型的攻击与其他的都不同，因为它是由通信双方中的某一方（发送者或接收者）实施的。报文的发送者可能事后否认她曾经发送过某个报文，而报文的接收者也可能事后否认自己曾接收到某个报文。发送者否认的一个例子是，某个银行用户要求该银行向第三方发送一笔钱，但是事后她却否认自己曾经提出过这个请求。接收者否认的一个

例子是，某人从生产商那里购买了一个产品，并用电子付款方式结了账，但是生产商在事后否认已经收到付款，并要求她再次付款。

威胁有效性的攻击

我们只讨论一种威胁有效性的攻击：**拒绝服务**（denial of service）。

拒绝服务 拒绝服务（DoS）是一种很常见的攻击。它会使一个系统的服务变得越来越慢，甚至完全中断服务。攻击者可以利用多种策略来达到这个目的。她可能会向服务器发送大量的假请求，使得服务器因负载过重而崩溃。攻击者也可能会拦截并删除服务器向客户发回的响应，使得客户认为该服务器没有响应。攻击者还可能拦截来自客户的请求，致使客户多次发送请求，从而使系统过载。

29.1.3 服务

为了实现上述安全目标并防御攻击，ITU-T 定义了一些安全服务。其中的每一种服务都是为了防御一种或多种攻击以维护上述安全目标而设计的。

29.1.4 技术

安全目标的真正实现需要使用一些技术。目前有两种主流的技术，一种非常普遍（加密术），另一种有些特别（隐密术）。

加密术

有些安全服务可以利用加密术来实现。**加密术**（Cryptography）这个词源自希腊语，意思是“秘密的书写”。但是在今天，这个词汇表示为了使报文变得安全和免于受到攻击而对其进行转换的一门科学和艺术。虽然过去加密术指的仅仅是利用密钥对报文进行加密（encryption）和解密（decryption），但是在今天，对它的定义要涉及到三种不同的机制：对称密钥加密术、不对称密钥加密术以及散列技术。我们将在本章后面的内容中对这三种技术逐个地讨论。

隐密术

虽然本章和下一章的内容都是以加密术作为实现安全服务的基本技术，但是另外还有一种在过去曾被用于秘密通信的技术现在又重新引起了人们的关注：隐密术。**隐密术**（steganography）这个词起源于希腊语，意思是“掩盖的书写（covered writing）”，它与加密术不同，加密术的意思是“秘密的书写（secret writing）”。加密术意味着通过加密的手段来隐藏报文的内容，而隐密术则意味着通过其他什么东西的遮盖来隐藏报文本身。我们把对隐密术的讨论留给专门介绍这门技术的书籍。

29.2 传统加密方法

现在我们来考察一下首要的安全目标：保密性。保密性可以通过使用加密方法来实现。传统加密方法称为**对称密钥加密方法**（symmetric-key ciphers）或**密钥加密方法**（secret-key

ciphers)，因为加密和解密时使用了相同的密钥，并且这个密钥可用于双向通信。图 29.2 所示为对称密钥加密方法背后的基本思想。

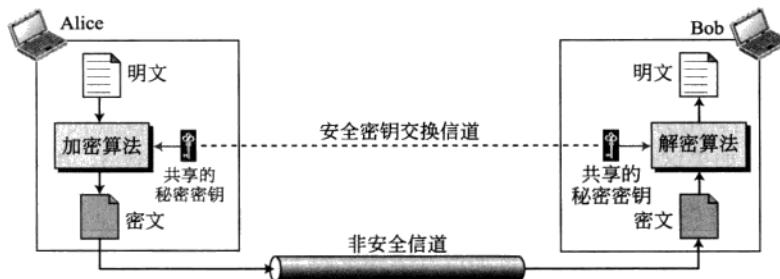


图 29.2 传统加密方法的基本思想

对称密钥加密方法也称为密钥加密方法。

从图 29.2 中可以看出，Alice（一个实体）通过一条非安全信道向 Bob（另一个实体）发送报文，并认为 Eve（一个对手）用简单的信道窃取方法将无法理解报文内容。

从 Alice 到 Bob 的原始报文称为明文（plaintext），通过信道发送的报文称为密文（ciphertext）。为了从明文产生密文，Alice 要使用加密算法（encryption algorithm）和共享密钥（shared secret key）。为了把密文还原成明文，Bob 要使用解密算法（decryption algorithm）和相同的密钥。我们把加密/解密算法统称为加密方法（cipher）。密钥（key）就是一组值（数字），由作为算法的加密方法对其进行操作。

请注意，在对称密钥保密中，加密和解密使用的是同一个密钥（当然这个密钥本身可能是一组数值）。另外，加密算法和解密算法互为逆运算。如果 P 是明文，C 是密文，K 是密钥，加密算法 $E_k(x)$ 从明文中产生密文，而解密算法 $D_k(x)$ 从密文中产生明文。我们假设 $E_k(x)$ 和 $D_k(x)$ 互为逆运算，也就是说如果对某个输入，我们先执行一种算法再执行另一种算法后，这两种算法的效果就互相抵消了。我们有：

$$\text{加密: } C = E_k(P)$$

$$\text{解密: } P = D_k(C)$$

其中， $D_k(E_k(x)) = E_k(D_k(x)) = x$ 。我们需要强调，最好是让加密和解密算法公开，而把共享密钥保密。这就表示 Alice 和 Bob 还需要另一条信道，一条安全的信道，以便互相交换密钥。Alice 和 Bob 可以见一次面并亲自交换密钥。这里的安全信道就是面对面的密钥交换过程。他们也可以通过一个信任的第三方把密钥交给双方。还可以使用另一种加密方法（不对称密钥加密方法）来产生一个临时的密钥，我们稍后再介绍。

29.2.1 密钥

加密可被视为把报文锁进一个盒子里，而解密则可被视为打开盒子的锁。在对称密钥保密中，上锁和开锁使用的是同一把钥匙，如图 29.3 所示。稍后我们将会看到不对称密钥保密则需要两把钥匙，一把用于上锁，另一把用于开锁。



图 29.3 对称密钥保密中用同一个密钥来上锁和开锁

29.2.2 替代加密方法

我们将传统的对称密钥加密方法划分为两个大类：替代加密方法和置换加密方法。替代加密方法（substitution cipher）用一个符号来替代另一个符号。如果明文中的符号是字母字符，那么我们就用另一个字符来替代该字符。例如，我们用字符 D 替代字符 A，用字符 Z 替代字符 T。如果这个符号是数字（0~9），那么我们可以用 7 替代 3，用 6 替代 2。

替代加密方法用一个符号来替代另一个符号。

替代加密方法又可分为单态字符加密方法和多态字符加密方法。

单态字符加密方法

使用单态字符加密方法（monoalphabetic cipher）时，明文中的一个字符（或符号）总是被转换成相同的另一个字符（或符号），而不考虑它在明文中所处的位置。例如，如果算法认为明文中的字母 A 应当变成字母 D，那么每一个字母 A 都会变成字母 D。换言之，明文中的字母与密文中的字母之间是一对一的关系。

最简单的单态字符加密方法是加法加密方法（additive cipher），也称为位移加密方法（shift cipher）。假设计明文由小写字母（a 到 z）组成，而密文由大写字母（A 到 Z）组成。为了在明文和密文上应用数学运算，我们为每个字母指派一个数值（小写或大写），如图 29.4 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
数值 →	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 29.4 用模 26 方法表示的明文和密文字符

在图 29.4 中，每个字符（小写的或大写的）被指派了一个模 26 的整数。Alice 和 Bob 之间的密钥也是一个模 26 的整数。加密算法就是用把明文字符的值加上密钥的值，解密算法就是把密文字符的值减去密钥的值。所有的运算都是模 26 的。

在加法加密方法中，明文、密文和密钥都是模 26 的整数。

历史上，加法加密方法也称为位移加密方法，因为它的加密算法可被解释为“向下移动 key 个字符”，而解密算法可被解释为“向上移动 key 个字符”。恺撒（Julius Caesar）就曾经使用过一种密钥为 3 的加法加密方法与他的军官们通信。因为这个原因，加法加密方法有时也被称为恺撒加密方法（Caesar cipher）。

例 29.1

请用 $key = 15$ 的加法加密方法对报文“hello”进行加密。

解

我们逐个字符地对这个明文应用加密算法：

明文：h → 07

加密： $(07 + 15) \bmod 26$

密文：22 → W

明文：e → 04

加密： $(04 + 15) \bmod 26$

密文：19 → T

明文：l → 11

加密： $(11 + 15) \bmod 26$

密文：00 → A

明文: $1 \rightarrow 11$ 加密: $(11 + 15) \bmod 26$ 密文: $00 \rightarrow A$ 明文: $0 \rightarrow 14$ 加密: $(14 + 15) \bmod 26$ 密文: $03 \rightarrow D$

结果得到的是“WTAAD”。请注意，这个加密方法是单态字符加密方法，因为两个相同的明文字符(1)加密后的字符(A)也相同。

例 29.2

请用 $\text{key} = 15$ 的加法加密方法对报文“WTAAD”进行解密。

解

我们逐个字符地对这个密文应用解密算法：

密文: $22 \rightarrow W$ 解密: $(22 - 15) \bmod 26$ 明文: $h \rightarrow 07$ 密文: $19 \rightarrow T$ 解密: $(19 - 15) \bmod 26$ 明文: $e \rightarrow 04$ 密文: $00 \rightarrow A$ 解密: $(00 - 15) \bmod 26$ 明文: $1 \rightarrow 11$ 密文: $00 \rightarrow A$ 解密: $(00 - 15) \bmod 26$ 明文: $1 \rightarrow 11$ 密文: $03 \rightarrow D$ 解密: $(03 - 15) \bmod 26$ 明文: $o \rightarrow 14$

结果得到的是“hello”。请注意，这里的运算都是模 26 的，也就是说我们需要在减出来的负值上再加 26（例如 -15 就变成 11 ）。

对于使用了密钥穷尽搜索方法的攻击（即强行攻击）来说，加法加密方法是非常脆弱的。加法加密方法的密钥范围很小，一共只有 26 个密钥，而且其中的密钥 0 又不起作用（密文与明文相同）。只剩下 25 个可能的密钥。Eve 很容易就可以对密文发起强行攻击。

因为加法加密方法的密钥域很小，导致它们非常容易被攻击。一种更好的解决方案是在明文字符和相应密文字符之间建立一个映射关系。Alice 和 Bob 可以同时使用一个列表，该列表说明了每个字符的映射关系。图 29.5 所示为此类映射的一个例子。

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 29.5 单态字符替代加密方法所使用的密钥的一个例子

例 29.3

我们使用图 29.5 中所示的密钥来加密以下报文：

this message is easy to encrypt but hard to find the key

它对应的密文是：

ICFVQRVVNEFVRNVSIYRGAHSLIOJCNHTIYBFGTICRXRS

多态字符加密方法

在使用多态字符替代 (polyalphabetic substitution) 时，同一个字符的每一次出现都可能有不同的替代。明文中的字符与密文中的字符是一对多的关系。字符“a”出现在明文前端被加密为“D”，但出现在中间某个地方则变为“N”。多态字符加密方法的优点是隐藏了底层语言中字母出现的频率。Eve 无法使用单字母频率统计值来破译这个加密方法。

为了创建一个多态字符加密方法 (polyalphabetic cipher)，我们需要让每个密文字符不仅依赖于相应的明文字符，而且与该明文字符在报文中的位置也有关。这就表示我们的密钥应当是一个子密钥流，其中的每个子密钥与被加密的明文字符的位置或多或少都有关系。

换言之，我们需要有一个密钥流 $k = (k_1, k_2, k_3, \dots)$ ，其中 k_i 用于对明文中的第 i 个字符进行加密，以产生密文中的第 i 个字符。

为了理解密钥与位置的相关性，让我们讨论一种简单的多态字符加密方法，称为自动密钥加密方法（autokey cipher）。使用这种加密方法时，密钥是一个子密钥流。第一个子密钥是 Alice 和 Bob 秘密协商后同意的一个预定值。第二个子密钥是第一个明文字符的值（0~25）。第三个子密钥的值是第二个明文字符的值。依此类推：

$$P = P_1 P_2 P_3 \dots$$

$$C = C_1 C_2 C_3 \dots$$

$$k = (k_1, P_1, P_2, \dots)$$

$$\text{加密: } C_i = (P_i + k_i) \bmod 26$$

$$\text{解密: } P_i = (C_i - k_i) \bmod 26$$

这种加密方法的名称“自动密钥”暗示了它的子密钥是在加密过程中，根据明文密码自动生成的。

29.2.3 置换加密方法

置换加密方法（transposition cipher）不是用一个符号来替代另一个符号，而是改变这些符号的位置。明文中第 1 个位置上的符号可能出现在密文的第 10 个位置上，而明文中第 8 个位置上的符号则可能出现在密文的第 1 个位置上。换言之，置换加密方法对符号进行重新排序（置换）。

置换加密方法对符号重新排序。

假设 Alice 希望秘密地向 Bob 发送报文“Enemy attacks tonight”。加密过程和解密过程如图 29.6 所示。

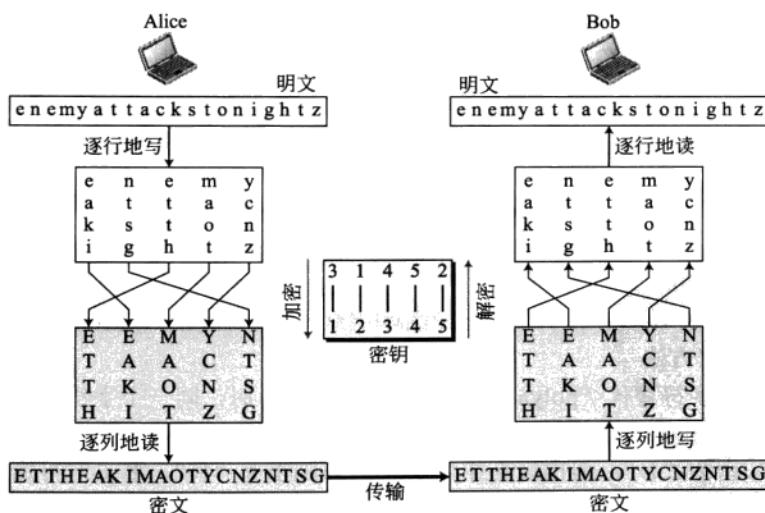


图 29.6 置换加密方法

第一张表是 Alice 把明文按逐行方式排列后形成的，然后她用密钥把表中列的次序打乱，再逐列地读出第二张表，就得到了相应的密文。Bob 逆向执行以上三个步骤。他把密

文逐列地写在第一张表中，对列进行重新排序，然后逐行地读出第二张表。请注意加密和解密过程使用的是相同的密钥，但是这两个算法在使用密钥时的方向正好相反。

29.2.4 流和块加密方法

文献把对称加密方法划分为两大类：流加密方法和块加密方法。

流加密方法

在使用流加密方法（stream cipher）时，加密和解密都是一次一个符号地（譬如一个字符或一个比特）进行。我们有一个明文流，一个密文流和一个密钥流，并且称这个明文流为 P，密文流为 C，密钥流为 K。

$$P = P_1 P_2 P_3 \dots \quad C = C_1 C_2 C_3 \dots \quad k = (k_1, k_2, k_3, \dots)$$

$$C_1 = E_{k1}(P_1) \quad C_2 = E_{k2}(P_2) \quad C_3 = E_{k3}(P_3)$$

块加密方法

在使用块加密方法（block cipher）时，长度为 m ($m > 1$) 的一组明文符号被同时加密，并产生一组等长的密文。基于这个定义，在块加密方法中，单个密钥被用来加密整个分组，不管这个密钥是不是由多个值组成的。在块加密方法中，一个密文块要依赖于整个明文块。

组合

实际上，每个明文块是单独加密的，但对于整个报文来看，就是用一个密钥流来一个数据块一个数据块地进行加密。换言之，就数据块个体来看，这个加密方法是一种块加密方法，但是就整个报文来看，因为每个块被视为一个单元，所以这个加密方法又是一种流加密方法。每个块使用了不同的密钥，这些密钥可能是在加密处理之前或中间产生的。在后面的章节中将会看到这样的例子。

29.3 现代加密方法

到目前为止，我们研究的传统对称密钥加密方法都是面向字符的加密方法（character-oriented cipher）。随着计算机的发明，我们需要面向比特的加密方法（bit-oriented cipher）。这是因为被加密的信息不仅仅是文本，还可能包含了数值、图像、音频和视频数据。如果把这些类型的数据都转换成比特流，然后对这个比特流进行加密，再发送加密后的比特流就很方便了。另外，当文本在比特一级进行处理时，一个字符会被转换成 8（或 16）位，也就是说符号的数量增加了 8（或 16）倍。被混合的符号越多，安全性也就越高。现代加密方法可以是块加密方法也可以是流加密方法。

29.3.1 现代块加密方法

对称密钥的现代块加密方法（modern block cipher）对 n 位明文块进行加密或者对 n 位密文块进行解密。加密算法或解密算法都要使用 k 位密钥。解密算法必须是加密算法的逆运算，且这两个运算必须使用相同的密钥，这样 Bob 就可以读懂 Alice 发来的报文了。

图 29.7 所示为现代块加密方法的加密和解密的基本思想。

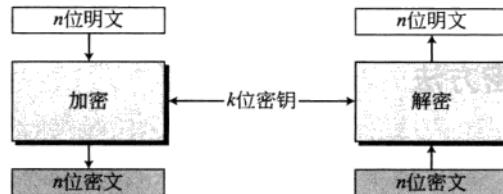


图 29.7 一种现代块加密方法

如果报文长度小于 n 位，那么必须添加填充使之成为 n 位的分组；如果报文长度大于 n 位，那么就应当把这个报文划分为若干个 n 位的分组，并为最后一个分组添加必要的填充。 n 的常见值包括 64、128、256 和 512。

现代块加密方法的组成

从整个块来看，现代块加密方法是一种替代加密方法。但是，现代块加密方法并没有被设计为一个独立的单元。为了提供防御攻击的能力，现代块加密方法由置换单元（有时称为 P 盒）、替代单元（有时称为 S 盒）、XOR 运算、位移要素、对换要素、分裂要素以及组合要素一起构成。图 29.8 描绘了现代块加密方法的组成。

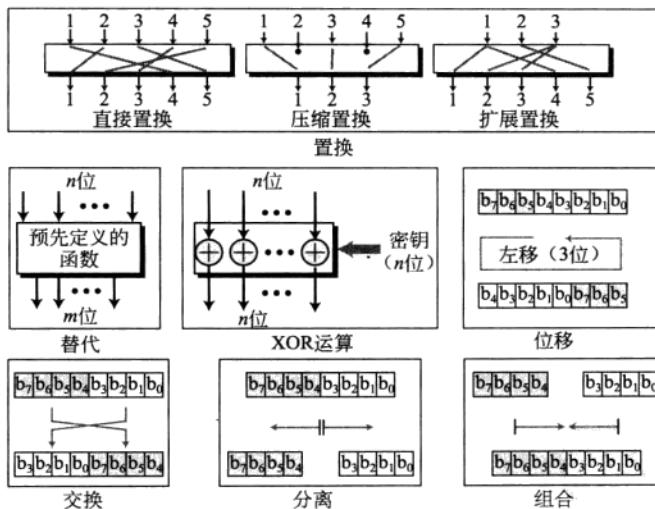


图 29.8 现代块加密方法的组成

图中 **P 盒** (permutation box) 类似于处理字符的传统置换加密方法，不过它所置换的是比特。在现代块加密方法中，我们可以看到三种类型的 P 盒：直接 P 盒、扩展 P 盒和压缩 P 盒。**S 盒** (substitution box) 可被认为是一个微缩版的替代加密方法，只不过它替代的是比特。与传统替代加密方法不同的是 S 盒可以有不同数量的输入和输出。在大多数块加密方法中，一个很重要的构件是 XOR 运算，也就是说若两个输入相同，则输出为 0，若两个输入不同，则输出 1。在现代块加密方法中，我们使用 n 位 XOR 运算，它把 n 位数据和

n 位密钥结合起来。一般来说，只有在这个 XOR 运算中才会用到密钥。

另外，在某些现代块加密方法中还可以看到循环位移运算（circular shift operation）。位移可以向左移，也可以向右移。循环左移操作把一个 n 位字的每一位向左移动 k 个位置。最左边的 k 位从左边消失而变为最右边的位。对换操作（swap operation）是一种特殊的循环位移操作，它被移动的位数正好是 $k = n/2$ 。

在一些块加密方法中还可能看到的其他两种操作是分裂和组合。分裂操作（split operation）把一个 n 位的字从正中间一分为二，从而产生两个等长的字。组合操作（combine operation）通常是把两个等长的字连接起来，从而产生一个 n 位的字。

29.3.2 数据加密标准（DES）

作为现代块加密方法的一个例子，让我们来讨论数据加密标准（Data Encryption Standard, DES）。图 29.9 所示为 DES 加密方法在加密端的组成要素。

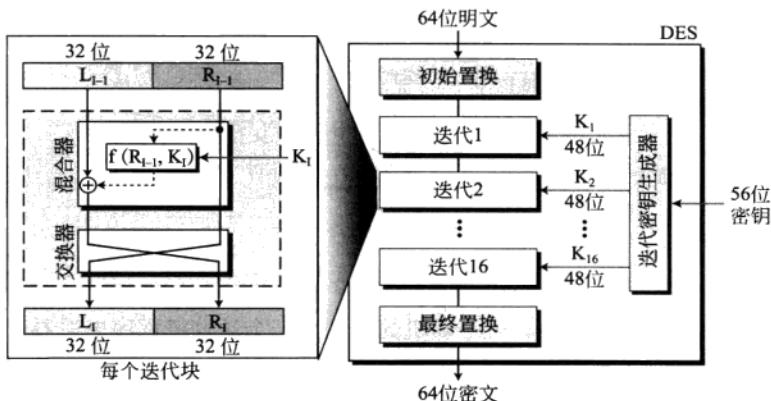


图 29.9 DES 的一般结构

在加密端，DES 用 64 位的明文产生 64 位的密文；在解密端，DES 用 64 位的密文产生 64 位的明文块。加密和解密过程中使用相同的 56 位密钥。

初始置换接收 64 位的输入，并根据预先设定的规则对它们进行置换。最终置换是初始置换的逆操作。这两个置换在效果上彼此抵消。换言之，如果从这个结构中拿走迭代块，那么密文和明文就是一模一样的。

迭代块

DES 使用了 16 个迭代块。DES 的每个迭代块都是一个互反的转换，如图 29.9 所示。每个迭代块接收上一个迭代块（或初始置换盒）输出的 L_{i-1} 和 R_{i-1} ，并生成 L_i 和 R_i ，以便交给下一个迭代块（或最终置换）使用。每个迭代块最多可以有两个密码要素（混合器和交换器）。每个要素都是互反的。交换器显然是互反的。它把文本的左半边与右半边交换。因 XOR 操作而使得混合器也是互反的。所有不是互反的要素都被集中在函数 $f(R_{i-1}, K_i)$ 中。

DES 函数

DES 的核心是 DES 函数。DES 函数对最右边 32 位输入应用 48 位的密钥，以产生 32

位的输出。这个函数由四部分组成：扩展 P 盒、XOR 构件（用来加入密钥）、一组 S 盒以及直接 P 盒，如图 29.10 所示。

因为 R_{t-1} 是 32 位的输入，而 K_t 是 48 位的密钥，所以我们首先要把 R_{t-1} 扩展到 48 位。此次扩展置换要遵守预先定义好的规则。

在经过扩展置换后，DES 对已扩展的右区和迭代密钥应用 XOR 运算，再由 S 盒进行真正的混合。DES 使用了 8 个 S 盒，每 6 位输入产生 4 位输出。DES 函数的最后操作是用 32 位的输入直接置换后得到 32 位的输出。

密钥生成器

迭代密钥生成器从 56 位的加密密钥中产生 16 个 48 位的迭代密钥。不过，加密密钥通常以 64 位密钥的形式给出，其中多出来的 8 位用作奇偶校验位，这 8 位在实际的密钥生成过程之前就被丢弃了，如图 29.11 所示。

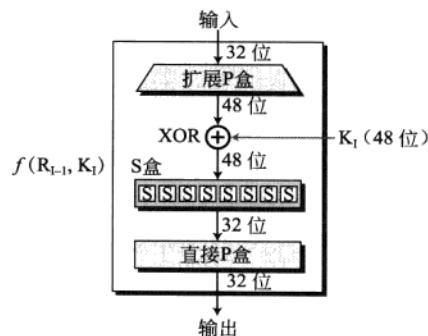


图 29.10 DES 函数

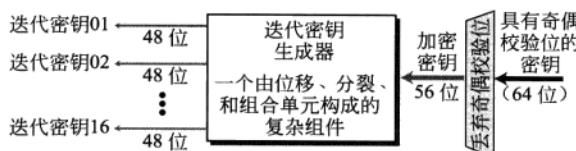


图 29.11 密钥的生成

例 29.4

我们选择一段随机的明文块、一个随机的密钥和一个计算机程序来看看生成的密文块会是什么（全部用十六进制表示）：

明文	密钥	密文
123456ABCD132536	AABB09182736CCDD	C0B7A8D05F3A829C

例 29.5

可以通过改变输入中的某一位的方法来检测 DES 的有效性，假设我们使用了两个只有一位不同，其余都相同的明文。在没有改变密钥的情况下，结果得到的两个密文是完全不同的。

明文	密钥	密文
0000000000000000	22234512987ABB23	4789FD476E82A5F1
明文	密钥	密文
0000000000000001	22234512987ABB23	0A4ED5C15A63FEA3

虽然两个明文块只有最右边的一位不同，但是密文块则有 29 位不一样。

29.3.3 现代流加密方法

除了现代块加密方法外，我们还可以使用现代流加密方法。现代流加密方法与现代块加密方法之间存在类似的区别。在现代流加密方法中，加密和解密过程都是一次处理 r 位。

我们有明文的位流 $P = p_n \dots p_2 p_1$, 密文的位流 $C = c_n \dots c_2 c_1$ 以及密钥位流 $K = k_n \dots k_2 k_1$, 其中 p_i 、 c_i 和 k_i 都是 r 位字。加密就是 $c_i = E(k_i, p_i)$, 而解密就是 $p_i = D(k_i, c_i)$ 。流加密方法比块加密方法的速度要快一些。流加密方法的硬件实现也要简单一些。当我们需要对二进制流进行加密并以固定的速率发送时, 使用流加密方法是一个更好的选择。流加密方法同时还对传输过程中的位损坏具有更强的免疫力。

一种最简单且最安全的同步流加密方法称为一次填充 (one-time pad), 它是由 Gilbert Vernam 发明并注册专利的。一次填充加密方法对每一次的加密都要随机选择一个密钥流。加密算法和解密算法各使用一个 XOR 操作。根据 XOR 操作的特点, 加密和解密算法是互为逆算法的。值得注意的是, 在这个加密方法中, XOR 操作是一次一位地进行的。同样还要注意, Alice 和 Bob 之间必须有一条安全信道, 这样 Alice 才能向 Bob 发送密钥流序列 (参见图 29.12)。

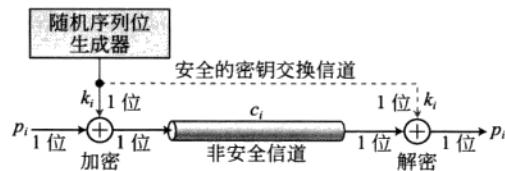


图 29.12 一次填充

一次填充是一种很理想的加密方法, 表现完美。对手根本没有办法猜测出密钥或明文和密文的统计数据。在明文和密文之间也不存在任何关系。换言之, 密文是真正随机比特流, 哪怕明文中反复出现相同的位组合样式。除非 Eve 尝试了所有可能的随机密钥流, 否则她不可能破解这个加密方法, 而对于长度为 n 位的明文来说, 这个随机密钥流可能有 2^n 之多。但是这种加密方法有一个问题。发送者和接收者怎样才能在每次需要通信时都能共享一次填充的密钥? 他们必须在某种程度上对随机密钥达成一致意见。因此这个完美而理想的加密方法是很难实现的。不过还是有一些安全程度稍微低一点的现实可行的版本。其中最常用的一种类似的方法称为反馈位移注册器 (feedback shift register, FSR), 不过我们把对这种有趣的加密方法的讨论留给专门介绍安全问题的书籍。

29.4 不对称密钥加密方法

在前面几小节里, 我们讨论的都是对称密钥加密方法。在本小节, 我们要开始讨论不对称密钥加密方法 (asymmetric-key ciphers)。对称密钥加密方法和不对称密钥加密方法将会长期共存并继续为大众服务。实际上, 我们认为它们彼此之间是互补的, 一方的优点正好弥补了另一方的缺点。

从概念上讲, 这两种系统之间的区别就在于它们用何种方式来保存密钥。在对称密钥加密术中, 这个密钥必须是双方共享的。而在不对称密钥加密术中, 密钥属于个人 (非共享的), 每个人都要生成并保存他或她自己的密钥。

在由 n 个人组成的群体中, 若使用对称密钥加密术, 则需要 $n(n - 1)/2$ 个共享密钥, 而在不对称密钥加密术中, 只需要 n 个私人密钥。对于人数达到百万的群体而言, 对称密钥加密术需要将近 5000 亿个共享密钥; 而不对称密钥加密术就只需要百万个私人密钥。

对称密钥加密术的基础是共享密钥；

不对称密钥加密术的基础是个人密钥。

除了保密之外，对于其他一些安全问题也需要使用不对称密钥加密术，包括鉴别和数据签名。只要某个应用程序是基于个人密钥的，我们就要使用不对称密钥加密术。

如果说对称密钥加密术基于对符号（字符或位）的替代和置换，那么不对称密钥加密术则基于应用数学公式计算数值。在对称密钥加密术中，明文和密文都被认为是符号的组合。加密和解密过程是将这些符号的次序打乱或用一个符号来替代另一个符号。而在不对称密钥加密术中，明文和密文都是数，加密和解密过程都是一些数学公式，用以对一些数值进行运算后得到另外一些数值。

在对称密钥加密术中，符号被置换或替代；

在不对称密钥加密术中，被处理的都是数值。

29.4.1 密钥

不对称密钥加密术使用了两个独立的密钥：一个私钥和一个公钥。如果把加密过程和解密过程视为用钥匙锁上挂锁和打开挂锁，那么被一把公钥锁住的挂锁就只能被一把对应的私钥打开。图 29.13 描绘的是如果 Alice 用 Bob 的公钥把挂锁锁住，那么只有 Bob 的私钥才能打开这把锁。

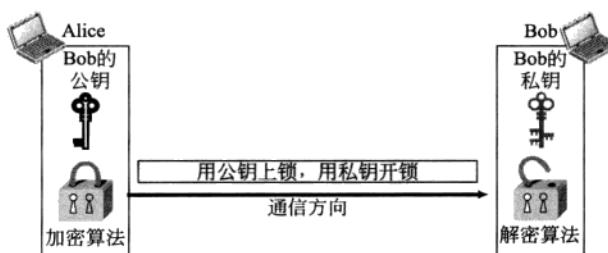


图 29.13 不对称密钥加密系统中的上锁和开锁

如上图所示，与对称密钥加密术不同的是，在不对称密钥加密术中有两个不同的密钥：一个私钥(private key)和一个公钥(public key)。虽然在一些书籍中使用了术语“密钥(secret key)”，而不是“私钥”，但是我们只在对称密钥加密术中使用术语“密钥”，而不对称密钥加密术中使用术语“私钥”和“公钥”。为了区分这三种密钥，我们甚至在图示中使用了不同的符号。换言之，我们希望说明“密钥”和“私钥”不能互换，它们是两种完全不同的密码。

不对称密钥加密方法有时也称为公钥加密方法。

29.4.2 总体思想

图 29.14 描绘了不对称密钥加密术在加密时的总体思想。

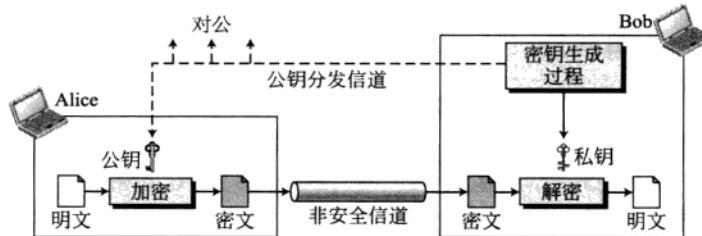


图 29.14 不对称密钥加密系统的总体思想

图 29.14 向我们展示了几个重要的事实。首先，它强调了这种加密系统的不对称性。也就是说提供安全的责任几乎全部由接收者（如图中的 Bob）承担。Bob 需要生成两个密钥：一个私钥和一个公钥。Bob 负责向整个团体分发他的公钥，这可以通过公钥分发信道来完成。虽然不要求这个信道提供安全性，但它必须提供鉴别和完整性。不应允许 Eve 向团体发布她自己的公钥并假冒那是 Bob 的公钥。

其次，不对称密钥加密术意味着 Bob 和 Alice 不能把一套密钥用于双向通信。团体中的每个实体都应当创建自己的私钥和公钥。图 29.14 描绘的是 Alice 如何使用 Bob 的公钥来向 Bob 发送加密的报文。如果 Bob 想回应，他就需要使用 Alice 的公钥。

第三，不对称密钥加密术意味着 Bob 只需要一个私钥就能接收来自团体内任何人传来的讯息，但是 Alice 则需要 n 个公钥才能与团体中的 n 个实体进行通信，一个公钥对应一个实体。换言之，Alice 需要有一大串公钥才行。

明文/密文

与对称密钥加密术不同，在不对称密钥加密术中，明文和密文被作为整数来对待。报文在加密前必须先进行编码形成整数（或整数的集合），而这个整数（或整数集合）在解密后必须经过解码才能还原成报文。不对称密钥加密术通常用来对量比较小的信息进行加密或解密，譬如说对称密钥加密术中的密钥。换言之，不对称密钥加密术通常并不用于报文本身的加密，而是为了达到辅助性的目标。但是这些辅助性的目标在今天的加密术中有着举足轻重的地位。

加密/解密

不对称密钥加密术中的加密和解密过程都是将数学公式应用于一些数值上，这些数值代表了明文和密文。密文可以被看成是 $C = f(K_{\text{public}}, P)$ ；明文可以被看成是 $P = g(K_{\text{private}}, C)$ 。加密函数 f 仅用于加密过程，解密函数 g 仅用于解密过程。

两者同样重要

有一个重要的事实很容易被人们误解：不对称密钥（公钥）加密术的诞生并不表示人们可以抛弃对称密钥加密术了。原因在于不对称密钥加密术使用数学公式进行加密和解密，这就比使用对称密钥加密术慢很多。为了加密长报文，对称密钥加密术仍然是不可取代的。另一方面，对称密钥加密术在速度上的优势也不能抹杀不对称密钥加密术的作用。对于报文鉴别、数字签名和密钥的交换来说，不对称密钥加密术也是必不可少的。总之，要想使用今天的所有安全服务，我们既需要对称密钥加密术，也需要不对称密钥加密术。二者互相取长补短。

29.4.3 RSA 加密系统

虽然不对称密钥加密系统不止一个，但最常见的公钥算法还是 **RSA 加密系统** (RSA cryptosystem)，这个名字源于发明者的姓名 (Rivest、Shamir 和 Adleman)。RSA 用了两个指数 e 和 d ，其中 e 是公开的， d 是保密的。假设 P 是明文， C 是密文。Alice 用 $C = P^e \pmod{n}$ 从明文 P 生成密文 C ；Bob 用 $P = C^d \pmod{n}$ 从 Alice 发来密文中读取明文。这个模 n 是一个非常大的值，是在密钥生成过程中创建的。

过程

图 29.15 描绘 RSA 使用的处理过程隐含的总体思想。

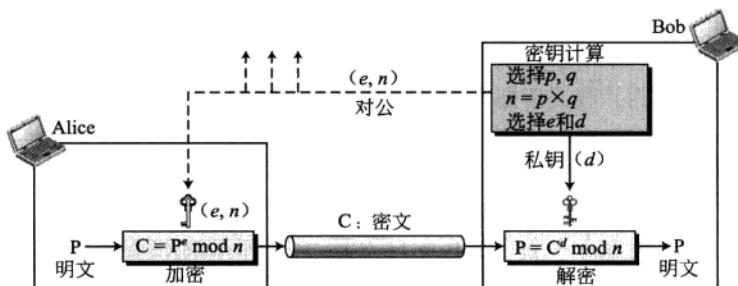


图 29.15 RSA 中的加密、解密和密钥生成

Bob 选择了两个较大的数 p 和 q ，然后计算 $n = p \times q$ 且 $\phi = (p - 1) \times (q - 1)$ 。然后，Bob 选择 e 和 d ，比如说让 $(e \times d) \bmod \phi = 1$ 。Bob 把 e 和 n 作为公钥向团体发布，同时把 d 作为密钥保存。任何人，包括 Alice，都可以用 $C = P^e$ 来对报文加密，并把密文发送给 Bob。只有 Bob 能够用 $P = C^d$ 来解密这个报文。如果 p 和 q 是非常大的数，像 Eve 这样的入侵者就无法解密这个报文（她不知道 d 是什么）。

例 29.6

我们所举的这个例子只是为了演示，假设 Bob 选择 7 和 11 作为 p 和 q ，并计算 $n = 7 \times 11 = 77$ 。另一个值 $\phi(n) = (7 - 1)(11 - 1) = 60$ 。如果他选择 e 为 13，那么 d 为 37。请注意， $(e \times d) \bmod 60 = 1$ 。现在假设 Alice 希望发送明文 5 给 Bob。她用公钥 13 作为指数来为 5 加密。这个系统并不安全，因为 p 和 q 都太小了。

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 29.7

这是一个更加符合实际的由计算机进行计算的例子。我们选择 512 位的 p 和 q ，计算出 n 和 $\phi(n)$ ，然后再选择 e 并计算出 d 。最后，我们给出加密和解密的结果。

整数 p 是一个长度为 159 个数字的数值。

$$p = \begin{array}{|c} \hline 961303453135835045741915812806154279093098455949962158225831508796 \\ 479404550564706384912571601803475031209866660649242019180878066742 \\ 1096063354219926661209 \\ \hline \end{array}$$

整数 q 是一个长度为 160 个数字的数值。

$$q = \begin{array}{|c} \hline 120601919572314469182767942044508960015559250546370339360617983217 \\ 314821484837646592153894532091752252732268301071206956046025138871 \\ 45524969000359660045617 \\ \hline \end{array}$$

模数 $n = p \times q$, 它有 309 个数字。

$$n = \begin{array}{|c} \hline 115935041739676149688925098646158875237714573754541447754855261376 \\ 147885408326350817276878815968325168468849300625485764111250162414 \\ 552339182927162507656772727460097082714127730434960500556347274566 \\ 628060099924037102991424472292215772798531727033839381334692684137 \\ 327622000966676671831831088373420823444370953 \\ \hline \end{array}$$

$\phi(n) = (p - 1)(q - 1)$, 它有 309 个数字。

$$\phi(n) = \begin{array}{|c} \hline 115935041739676149688925098646158875237714573754541447754855261376 \\ 147885408326350817276878815968325168468849300625485764111250162414 \\ 552339182927162507656751054233608492916752034482627988117554787657 \\ 013923444405716989581728196098226361075467211864612171359107358640 \\ 614008885170265377277264467341066243857664128 \\ \hline \end{array}$$

Bob 选择 $e = 35535$ (理想的数字是 65537), 然后他找出 d 的值。

$$e = \begin{array}{|c} \hline 35535 \\ \hline \end{array}$$

$$d = \begin{array}{|c} \hline 580083028600377639360936612896779175946690620896509621804228661113 \\ 805938528223587317062869100300217108590443384021707298690876006115 \\ 306202524959884448047568240966247081485817130463240644077704833134 \\ 010850947385295645071936774061197326557424237217617674620776371642 \\ 0760033708533328853214470885955136670294831 \\ \hline \end{array}$$

Alice 想要发送报文 “THIS IS A TEST”。这个报文通过 00-26 编码机制 (26 是空格键) 被转换为一个数值。

$$P = \begin{array}{|c} \hline 1907081826081826002619041819 \\ \hline \end{array}$$

Alice 计算的密文是 $C = P^e$, 也就是:

$$C = \begin{array}{|c} \hline 475309123646226827206365550610545180942371796070491716523239243054 \\ 4529606131993285666178434183591141511974112520056829797945717360361 \\ 012782188478927415660904800235071907152771859149751884658886321011 \\ \hline \end{array}$$

483541033616578984679683867637337657774656250792805211481418440481 4184430812773059004692874248559166462108656

Bob 可以用 $P = C^d$ 从密文中恢复明文，也就是：

P = 1907081826081826002619041819

经过解码后，恢复的明文就是“THIS IS A TEXT”。

29.4.4 应用

虽然 RSA 能够用来加密和解密实际的报文，但是如果报文比较长，它的速度会很慢。因此，RSA 适用于较短的报文。实际上，RSA 用在数字签名中，还有其他一些经常需要对短报文加密，而又不想使用对称密钥的加密系统。RSA 还用于鉴别，我们将在下面的章节中了解到。

29.5 报文完整性

目前为止，我们所讨论的加密系统都是为了提供秘密性或是保密性的，但没有涉及到完整性。但是，在有些环境下我们甚至不需要保密性，相反地，却必须要有完整性。例如，Alice 可能写了一份遗嘱，以便去世后分配她的财产。这份遗嘱不需要加密。在她去世后，任何人都可以查看这份遗嘱。但是，这份遗嘱的完整性却必须受到保护。Alice 不希望有人篡改遗嘱的内容。

29.5.1 报文和报文摘要

保护文档完整性的一种方法是通过使用手印（fingerprint）。如果 Alice 要确保自己的文档内容不被人改动，她可以在文档的开始和结尾处按上自己的手印。Eve 没有办法更改这份文档的内容或制造一份假文档，因为她无法伪造 Alice 的手印。为了确保文档没有被改动过，可以用档案中 Alice 的手印和文件中 Alice 的手印进行对比。如果它们不相同，那么这份文档就不是来自 Alice 的。文档和手印这对搭档就相当于电子世界中的报文和摘要。为了保护一个报文的完整性，这个报文要经过一个称为加密散列函数（cryptographic hash function）算法的处理。这个函数为报文产生一个压缩的印记，称为摘要（digest），它可以像手印一样使用。为了检查报文或文档的完整性，Bob 也要运行相同的加密散列函数，并比较新旧两个摘要。如果这两个摘要相同，那么 Bob 就可以肯定原始的报文没有被改动过。图 29.16 所示为这种思想。

这两对搭档（文档/手印和报文/报文摘要）很相似，只有一点区别。文档和手印是在物理上绑在一起的。报文和报文摘要可以不被绑在一起（或者说可以分别发送），但最重要的是这个报文摘要必须确保没有被他人改动过。

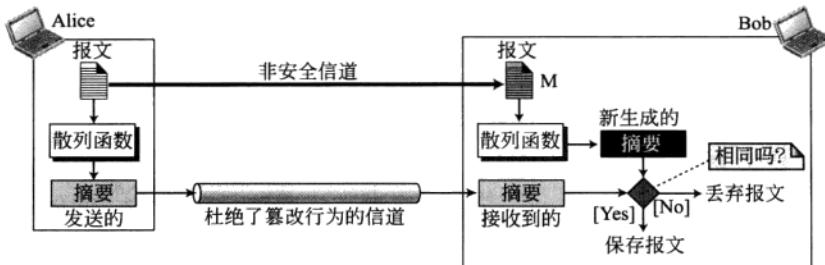


图 29.16 报文和摘要

报文摘要必须确保没有被他人改动过。

29.5.2 散列函数

加密散列函数以任意长度的报文作为输入，并产生固定长度的报文摘要。所有的加密散列函数都必须能够从变长的报文中产生出固定长度的摘要来。要创建这样的函数最好使用循环方法。我们不是使用一个输入变长报文的散列函数，而是创建一个输入长度固定的散列函数，然后在需要时反复多次地使用它。这个输入长度固定的函数称为压缩函数（compression function）。它压缩一个 n 位的字符串以产生一个 m 位的字符串，通常 n 都大于 m 。这种机制称为重复加密散列函数（iterated cryptographic hash function）。

有一些散列算法是由 Ron Rivest 设计的。它们被称为 **MD2**、**MD4** 和 **MD5**，其中 MD 代表“报文摘要（Message Digest）”。最新版本的 MD5 是 MD4 的加强版，因为 MD4 把报文划分为 512 位的块，并产生 128 位的摘要，结果人们发现长度为 128 位的报文摘要太小了，很容易受到攻击。

安全散列算法（Secure Hash Algorithm，SHA）是由美国国家标准和技术学会（NIST）开发的一个标准。SHA 也经历了多个版本。

29.6 报文鉴别

摘要可用于检查报文的完整性，也就是说报文没有被篡改过。为了确保报文的完整性以及实现数据来源的鉴别（也就是说 Alice 是这个报文的发起者，而不是其他人），我们需要在这个处理过程中再包括由 Alice 掌握的密钥（Eve 不知道的），因此我们需要创建报文鉴别码（message authentication code，MAC）。图 29.17 描绘了这个思想。

Alice 利用散列函数从密钥与报文的组合 $h(K + M)$ 中产生 MAC。她把这个报文和 MAC 一起通过非安全信道发送给 Bob。Bob 将报文与 MAC 剥离，然后再用该报文与密钥的组合生成新的 MAC。于是 Bob 就可以把新创建的 MAC 与接收到的 MAC 进行比较。如果这两个 MAC 一致，则说明这个报文来源正确，且没有被对手修改过。

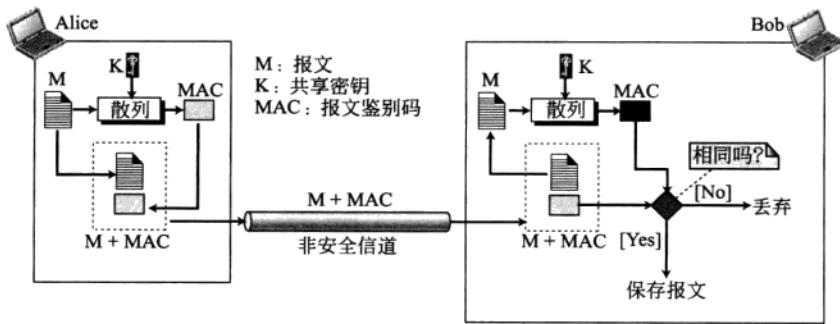


图 29.17 报文鉴别码

请注意，在这种情况下不需要使用两条信道。报文和 MAC 都可以在同一条非安全信道上发送。Eve 可以看到这个报文，但是她没有办法伪造一个新的报文来取代它，因为 Eve 不知道 Alice 和 Bob 之间的共享密钥。她无法创建出与 Alice 的一模一样的 MAC。

MAC 通过使用散列函数与密钥的组合，提供了报文的完整性保证和报文的鉴别。

29.6.1 HMAC

NIST 公布了嵌套的 MAC 的标准，一般称之为 **HMAC**（散列的 MAC）。HMAC 实现起来比简单 MAC 要复杂得多。

29.7 数字签名

另一种提供报文完整性和报文鉴别性（以及其他更多的安全服务，稍后将会了解到）的方法是数字签名。MAC 使用密钥来保护摘要，而数字签名则使用了一对私钥-公钥。

数字签名使用了一对私钥-公钥。

我们对签名这个概念都不陌生。一个人在文档上签名是为了表明这份文档是由她撰写或者是经她批准的。这个签名对接收者来说就是一个证据，说明该文档来自于正确的实体。当顾客在支票上签字后，银行就需要查实这个支票的确是由该顾客签发的，而不是别人伪造的。换言之，文档上的签名在被证实之后，就成为一个鉴别的记号，说明了这份文档来源属实。想象一幅有画家签名的作品。这幅作品上的签名，如果被证明属实，那么就表示这幅作品很有可能就是一幅真迹。

当 Alice 向 Bob 发送报文时，Bob 需要检查发送者的真实身份。他需要确保这个报文来自 Alice，而不是来自 Eve。Bob 可以要求 Alice 以电子方式签署该报文。换言之，电子签名可以证实报文发送者的真实身份就是 Alice。我们称这种签名为 **数字签名**（digital signature）。

29.7.1 比较

让我们来考察一下传统签名和数字签名之间的区别。

包含

传统签名包含在文档中，它是文档的一部分。在我们开支票时，签名就写在支票上，而不是在另一个独立的文档中。但是当我们对一个文档进行数字签名时，这个签名作为一个独立的文件发送。

验证方法

这两种类型的签名之间的第二个区别在于验证签名的方法不同。对于传统签名，当接收者收到一个文档时，她要用一个文件中的签名来和文档中的签名进行比较。如果这两个签名一样，就说明该文档的来源属实。因此接收者需要有一个文件，上面有同一个人的签名，这样才能进行比较。对于数字签名来说，接收者收到报文和签名。他不需要在其他任何地方保存同样的签名。接收者只需要对该报文和签名的组合应用某种验证技术，就能证实报文来源属实。

关系

对于传统签名，在签名和文档之间通常是一对多的关系。一个人使用相同的签名来签署多份文档。对于数字签名，签名和报文之间是一对一的关系。每个报文都有自己的签名。某一个报文的签名不能被用于其他报文。如果 Bob 一前一后连续收到两个来自 Alice 的报文，他不能用第一个报文的签名来验证第二个报文。每个报文都需要一个新签名。

可重复性

这两种类型的签名之间的另一个区别是称为可重复性 (duplicity) 的特点。使用传统签名时，签了名的文档的副本能够与原件区分开来。而在数字签名中却不存在此种区别，除非在文档上使用了时间元素（比如时间戳）。例如，假设 Alice 发送了一个文档，指示 Bob 向 Eve 付款。如果 Eve 截获了这个文档和签名，她就可以在此之后重放这个文档，以便再次要求 Bob 付款。

29.7.2 过程

图 29.18 所示为数字签名的过程。发送者使用 **签名算法** (signing algorithm) 来签署报文。该报文和签名被发送给接收者。接收者收到报文和签名，并对它们的组合应用相应的验证算法 (verifying algorithm)。如果结果是真，则该报文被接受，否则拒绝接受。

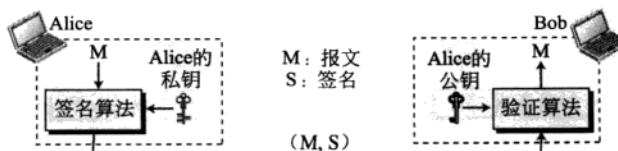


图 29.18 数字签名过程

传统的签名就好像是一个私人密钥，属于文档的签署人。签署人用它来签署文档，其他任何人都没有相同的签名。文件上的签名的副本就像是一个公开密钥，任何人都可以用它来验证文档的真伪，只要把它与原始的签名相比对即可。

在数字签名中，签署人利用她的私人密钥，通过签名算法来签署文档。另一方面，验证人利用签署人的公钥，通过验证算法来验证文档。

请注意，一旦文档被签署，任何人，包括 Bob，都可以对其进行验证，因为每个人都可以访问得到 Alice 的公钥。Alice 不能用她的公钥来签署文档，因为这样做的话任何人都可以伪造她的签名。

我们能用对称密钥加密中的密钥来签署和验证签名吗？答案是否定的，原因如下所述。首先，这个密钥只有两个实体知道（例如 Alice 和 Bob）。因此如果 Alice 想要签署另一份文档并将其发送给 Ted，她就需要另一个密钥。第二，我们将会看到，为一次会话而创建一个密钥将会涉及到鉴别，但是鉴别又要用到数字签名，这就形成一个恶性循环。第三，Bob 可以用他和 Alice 之间的密钥签署一份文档，然后将其发送给 Ted，并假装是 Alice 发送过来的。

数字签名需要的是公钥系统。

签署者用自己的私钥签署文档，验证者用签署者的公钥进行验证。

我们应该弄清楚在数字签名中和在为了保密而使用的加密系统中，私钥和公钥的使用方式是不同的。在后一种情况的处理过程中使用的是接收者的私钥和公钥，也就是说发送者用接收者的公钥加密，接收者用自己的私钥解密。而在数字签名中使用的是发送者的私钥和公钥，也就是说发送者用自己的私钥签名，接收者用发送者的公钥验证。

加密系统使用的是接收者的私钥和公钥；

数字签名使用的是发送者的私钥和公钥。

29.7.3 对摘要的签名

前面我们已经讲过，在处理长报文时不对称密钥加密系统的效率很低。在数字签名系统中，报文通常都很长，但我们又必须要使用不对称密钥机制。解决的办法是对该报文的摘要进行签名，这个摘要通常比报文要小得多。一个精心选择的报文摘要与该报文之间有一对一的关系。发送者可以对报文摘要进行签名，而接收者可以验证这个报文摘要。其效果是一样的。图 29.19 所示为在数字签名系统中对摘要进行签名的过程。

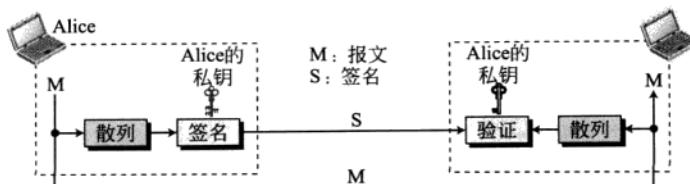


图 29.19 对摘要的签名

在 Alice 的站点上，先从报文中生成一个摘要。随后，用 Alice 的私钥对这个摘要进行签名。Alice 再把这个报文和签名发送给 Bob。

在 Bob 的站点上，先用相同的公开散列函数从接收到的报文中生成一个摘要。然后实施验证处理。如果验证通过，报文就被接受，否则被拒绝。

29.7.4 服务

在本章的一开始我们讨论了几个安全服务，包括报文保密、报文鉴别、报文完整性和不可否认性。数字签名可以直接提供后三种服务，而对于报文的保密，我们还是需要加密/解密技术。

报文鉴别

安全的数字签名机制就像安全的传统签名一样（不能轻易地被复制），能够提供报文的鉴别（也称为数据源鉴别）。Bob 可以证实该报文是由 Alice 发送来的，因为他使用了 Alice 的公钥进行验证。用 Alice 的公钥，由 Eve 的私钥签署的签名是不能通过验证的。

报文完整性

不管我们是否是对整个报文进行签名，报文的完整性都能得到保证，因为如果报文有改变，签名不可能不变。今天的数字签名机制在签名算法和验证算法中都使用了一个散列函数，以保证报文的完整性。

不可否认性

如果 Alice 签署了一份报文，然后又否认有这回事，Bob 在事后能不能证明 Alice 确实签署过？例如，如果 Alice 向银行（Bob）发送了一个报文，要求从她的账户上转出 10000 美元到 Ted 的账户上，Alice 事后能否否认她曾发送过这个报文吗？就目前为止我们提到的那些机制来说，Bob 可能会遇到麻烦。Bob 必须保存文件中的签名，并在事后用 Alice 的公钥来建立原始的报文，以证明文件中的报文和新生成的报文是完全一样的。但是这不太可行，因为 Alice 可能在段时间内更改过她的私钥或公钥，她也可能声明包含了签名的文件本身就是假冒的。

一种解决办法是引入可信的第三方，也就是说可以创建彼此都信任的第三方。在本章的后面，我们将会看到这个可信的第三方能解决很多与安全服务和密钥交换相关的问题。图 29.20 描绘了这个可信的第三方是如何防止 Alice 否认她曾经发送过该报文的。

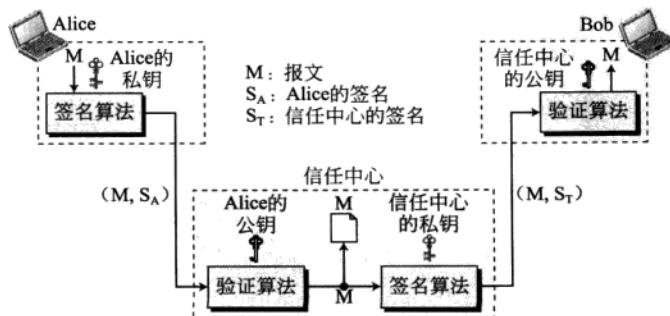


图 29.20 为了不可否认而使用一个信任的中心

Alice 用她的报文 (S_A) 产生签名，并把这个报文、她的身份、Bob 的身份以及该签名一起发送给中心。该中心在检查过 Alice 的公钥的合法性后，通过 Alice 的公钥验证报文的确来自 Alice。该中心然后将报文的副本以及发送者身份、接收者身份和时间戳一起保存在它的档案库中。该中心用自己的私钥生成报文的另一个签名 (S_T)，然后把报文以及新的签名、Alice 的身份、Bob 的身份一起发送给 Bob。Bob 用可信的第三方的公钥来验证这个报文。

如果将来有一天，Alice 否认她曾发送过这个报文，该中心就可以出示被保存的报文的副本。如果 Bob 出示的报文与中心保存的报文完全一致，那么 Alice 就在争议中输了。为了使这一切都具有保密性，在此机制上还需要增加一定程度的加密/解密过程，具体情况将在下一节讨论。

保密性

数字签名并不能提供保密的通信。如果需要保密性，那么报文和签名都必须使用对称密钥加密或公钥加密系统进行加密。

29.7.5 RSA 数字签名机制

在最近的几十年中已演化出了若干个数字签名机制 (digital signature scheme)，其中有一些已被实现。在这一小节中，我们将简单地介绍其中之一：RSA。在上一小节中，我们讨论了如何使用 RSA 加密方法系统来提供保密性。RSA 的思想也可用于对报文的签名和验证。此时它被称为 RSA 数字签名机制 (RSA digital signature scheme)。数字签名机制改变了私钥和公钥的作用。首先，这个机制使用的是发送者的私钥和公钥，而不是接收者的私钥和公钥。其次，发送者用自己的私钥来签署文档，接收者使用发送者的公钥来验证它。如果我们将这个机制与传统的签名进行比较，就会发现这个私钥所起的作用相当于发送者的签名，而发送者的公钥则相当于公开签名的副本。很显然，Alice 不能用 Bob 的公钥来签署文档，因为如果这样做，则其他任何人都可以这样做。签名和验证使用的是相同的函数，但是参数不同。验证方要比较报文与函数的输出是否相同 (在模运算下)。如果结果相同，则报文被接受。图 29.21 描绘了这个机制，其中签名和验证过程都是针对报文摘要的，而不是针对报文本身的，因为公钥加密方法用于长报文时效率很低，而摘要比报文本身要短得多。

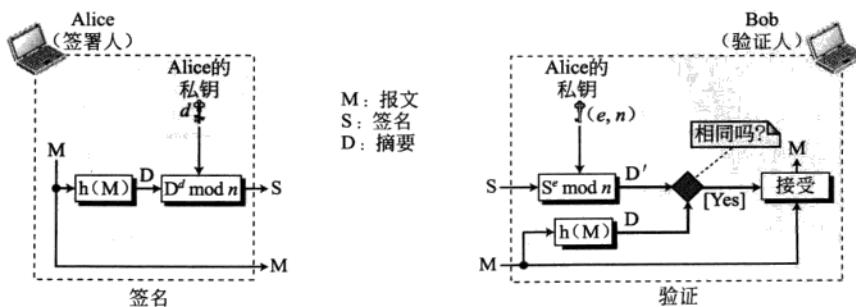


图 29.21 对报文摘要的 RSA 签名

作为签署人的 Alice 首先用约定好的散列函数从报文生成一个摘要 $D = h(M)$ 。然后她对这个摘要进行签名 $S = D^d \bmod n$ 。该报文和签名一起被发送给 Bob。作为验收人的 Bob 收到该报文以及签名。他首先用 Alice 的公开密钥读出报文的摘要 $D' = S^e \bmod n$ ，然后对接收到的报文应用同样的散列函数以获得 $D = h(M)$ 。现在 Bob 可以比较这两个摘要 D 和 D' 。如果它们相等（用模运算），则 Bob 可以接受这个报文。

29.7.6 数字签名标准

数字签名标准（Digital Signature Standard, DSS）是美国国家标准和技术学会（NIST）在 1994 年采纳的。DSS 是一个非常复杂且更安全的数字签名机制。

29.8 实体鉴别

实体鉴别是为了让一方证实另一方的身份而设计的技术。实体可以是人、进程、客户程序或服务器程序。需要被证实其身份的实体称为申请者（claimant），而试图证实申请者身份的实体称为验证者（verifier）。

29.8.1 实体鉴别和报文鉴别的比较

实体鉴别和报文鉴别（数据源鉴别）之间存在两个区别。

1. 报文鉴别（或者说数据源鉴别）不可能实时发生，而实体鉴别则有可能。对前者来说，Alice 发送报文给 Bob。当 Bob 鉴别这个报文时，Alice 有可能在，也有可能不在通信的现场。与此相反的是，当 Alice 请求实体鉴别时，除非 Alice 已经通过了 Bob 的鉴别，否则不可能发生实时的报文通信。Alice 需要在线等待并参与到整个过程中。只有在她被鉴别后，Alice 和 Bob 之间才能有报文上的往来。Alice 向 Bob 发送电子邮件后，需要的是数据源鉴别。而 Alice 从自动取款机上取现金时，需要的就是实体鉴别。

2. 报文鉴别只是对一个报文的鉴别。对于每个新报文，这个处理过程都要重复一遍。而实体鉴别则是对申请者在整个会话期间的身份鉴别。

29.8.2 验证类别

在实体鉴别时，申请者必须向验证者表明自己的身份。申请者可以通过以下三种证据来体现：一些记在脑子里的东西、一些拿在手里的东西或一些与生俱来的东西。

- **一些记在脑子里的东西** 这是一些只有申请者知道并能被验证者检查的秘密。例如口令、PIN、密钥或私钥。
- **一些拿在手里的东西** 这是一些能够证实申请者身份的物件。例如护照、驾驶证、身份证件、信用卡或智能卡。
- **一些与生俱来的东西** 这是一些申请者天生的特点。如传统签名、指纹、话音、面

部特点、视网膜以及笔迹等。

在这一小节，我们只讨论第一种类型的证据，即一些记在脑子里的东西，通常它被用于远程（在线）实体鉴别。其他两种类型的证据通常由申请者在现场亲自出示。

29.8.3 口令

最简单、最古老的实体鉴别方式就是使用口令（password），即申请者知道的某个东西。当用户需要访问一个系统以使用该系统的资源（即登录）时就要用到口令。每个用户有一个公开的用户标识和一个秘密的口令。不过，口令很容易受到攻击，它可能会被窃取、拦截或被猜出来等等。

29.8.4 查问-响应

在使用口令鉴别时，申请者通过出示自己知道的秘密（即口令）来证实自己的身份。但是，因为申请者必须发送这个秘密，所以它就很有可能被对手截获。使用查问-响应鉴别（challenge-response authentication）时，申请者不用发送该秘密就可以证实自己知道那个秘密。换言之，申请者不需要向验证者发送秘密，验证者也知道这个秘密或者可以找出这个秘密。

使用查问-响应鉴别时，申请者不需要向验证者发送秘密就能证明自己知道这个秘密。

查问是由验证者发送的一个随时变化的值，譬如一个随机数或是一个时间戳。申请者对这个数值应用一个数学公式，然后将结果（即响应）发送给验证者。从这个响应可以看出申请者是否知道这个秘密。

使用对称密钥加密方法

有一些查问-响应鉴别方式使用了对称密钥加密方法。此时申请者和验证者都知道的秘密就是共享密钥，而应用于查问的数学公式就是加密算法。虽然这种方式有多种实现方法，但我们在本书中只介绍最简单的一种，以便说明其最基本的思想。图 29.22 所示为这种方法。

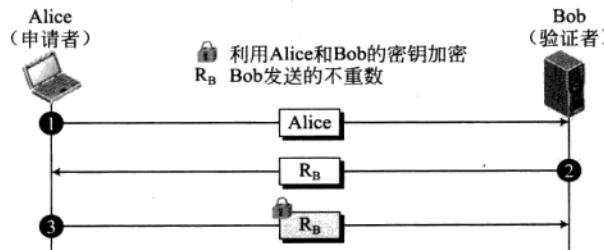


图 29.22 单向对称密钥鉴别

图中的第一个报文既不是查问也不是响应，它的作用是告诉验证者有一个申请者希望被查问。第二个报文就是查问。 R_B 是由验证者（Bob）随机选择的一个不重数（nonce，是 number once 的缩写），用于向申请者提问。申请者用只有自己和验证者才知道的共享密钥

对收到的不重数进行加密，并将结果发送给验证者。验证者对收到的报文进行解密。如果解密后得到的数值与验证者先前发出的不重数完全相同，那么 Alice 的身份就被证实。

请注意，在这个过程中申请者和验证者都需要保存用于加密的对称密钥，同时验证者还必须保存发送给申请者的不重数，直至收到响应为止。

使用不对称密钥加密方法

图 29.23 描绘了这种方法。

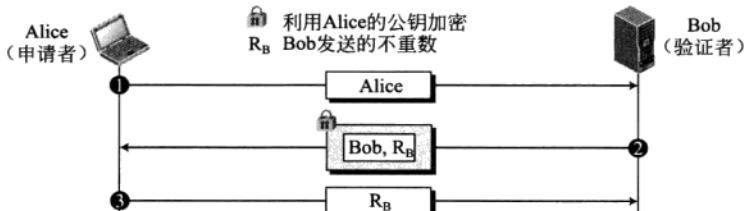


图 29.23 单向不对称密钥鉴别

除了用对称密钥加密方法之外，我们还可以用不对称密钥加密方法进行实体鉴别。此时那个秘密就是申请者的私钥。申请者必须表明她掌握着与公众开放的公钥相对应的私钥。也就是说，验证者必须用申请者的公钥对查问进行加密，而申请者则使用自己的私钥对该报文进行解密。对查问的响应就是该查问被解密后得到的结果。如果在第三个报文中收到的 R_B 与第二个报文中发送的 R_B 相同，Alice 的身份就被证实。

使用数字签名

实体鉴别也可以使用数字签名来实现。当数字签名用于实体鉴别时，申请者用自己的私钥进行签名。在图 29.24 所示的第一种方法中，Bob 用明文发送查问，而 Alice 用签名来进行响应。如果在第三个报文中收到的 R_B 与第二个报文中的相同，那么 Alice 的身份就被证实。

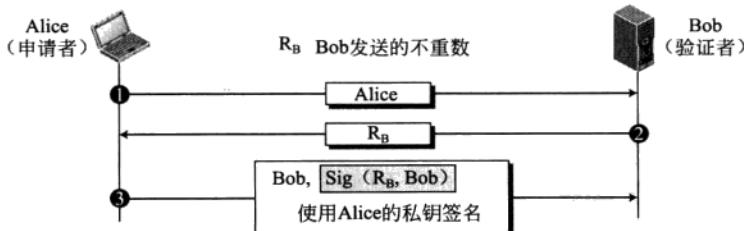


图 29.24 使用数字签名的单向鉴别

29.9 密钥管理

在前几节的内容中，我们讨论了对称密钥和不对称密钥加密术。但是，我们还没有讨论对称密钥加密术中的共享密钥和不对称密钥加密术中的公钥是怎样分发和管理的。这里我们就要讨论这两个重要问题。

29.9.1 对称密钥的分发

在对长报文进行加密时，对称密钥加密术要比不对称密钥加密术效率更高。但是对称密钥加密术需要双方之间有一个共享密钥。

如果 Alice 要与 N 个人交换秘密报文，那么她就需要 N 个不同的密钥。如果这 N 个人之间也要相互通信，那又会怎么样呢？如果我们要求 Alice 和 Bob 用两个密钥进行双向通信，那么一共就需要 $N(N - 1)$ 个密钥。如果我们允许双向通信只用一个密钥，那么一共就需要 $N(N - 1)/2$ 个密钥。也就是说，如果有百万个人要相互通信，那么每个人都要掌握大约一百万个不同的密钥，总共大约需要十亿个密钥。这通常被称为 N^2 问题，因为对于 N 个实体就大约需要 N^2 个密钥。

除了密钥的数量之外，密钥的分发也是一个问题。如果 Alice 和 Bob 需要互相通信，他们就需要交换一个密钥。如果 Alice 希望与一百万个人通信，她又如何与一百万个人交换一百万个密钥呢？利用因特网肯定不是安全的办法。很显然，我们需要一种更有效的方式来维护和分发密钥。

密钥分配中心：KDC

一种实际的解决方案是利用一个可信的第三方，称为密钥分配中心（key-distribution center，KDC）。为了减少密钥的数量，每个人要建立与 KDC 之间的共享密钥。也就是说在 KDC 和每个成员之间都有一个密钥。现在的问题是 Alice 怎样才能把秘密报文发送给 Bob。过程如下：

1. Alice 向 KDC 发送请求，表示自己和 Bob 之间需要会话（临时）密钥。
2. KDC 把 Alice 的请求通知到 Bob。
3. 如果 Bob 同意，就在他们俩之间创建一个会话密钥。

Alice 和 Bob 与 KDC 之间的密钥用于 Alice 和 Bob 向 KDC 鉴别自己的身份，以防止 Eve 假冒他们俩中的任何一个。

多 KDC 配置 当使用 KDC 的人数增加到一定程度后，系统会变得不可控制，并因此而产生瓶颈。为了解决这个问题，我们就需要多个 KDC。我们可以把整个世界划分为几个区。每个区都可以有一个或多个 KDC（为了防止故障而做的备份）。现在，如果 Alice 希望向位于另一个区的 Bob 发送一个秘密报文，那么 Alice 首先要与自己的 KDC 联系，然后由它与 Bob 所在区的 KDC 联系。通过这两个 KDC 就可以建立 Alice 和 Bob 之间的密钥。这些 KDC 可以分为本地 KDC、国家 KDC 和国际 KDC。当 Alice 需要与生活在另一个国家的 Bob 通信时，她向本地 KDC 发送自己的请求，本地 KDC 将请求转发给国家 KDC，国家 KDC 将请求转发给国际 KDC。然后这个请求又被一路转发到 Bob 所在地区的本地 KDC。图 29.25 描绘了分级的多 KDC 的配置。

会话密钥 KDC 为每个成员建立一个密钥。这个密钥仅被用于 KDC 和该成员之间，而不是用于两个成员之间的。如果 Alice 希望与 Bob 进行秘密通信，那么她就需要与 Bob 之间有一个密钥。利用他们与 KDC 之间的密钥，KDC 可以生成 Alice 与 Bob 之间的会话密钥（session key）。他们与 KDC 之间的密钥在会话密钥生成之前用于中心鉴别 Alice 和 Bob 的身份，以及他们彼此之间的鉴别。在通信结束后，这个会话密钥就作废了。

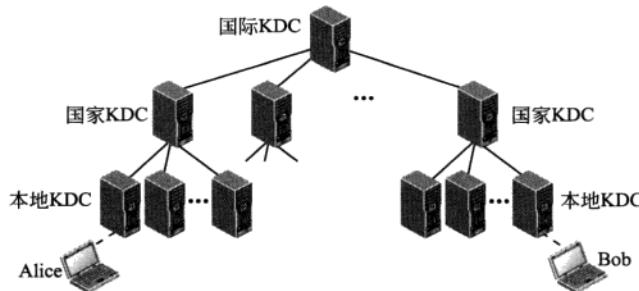


图 29.25 多 KDC

通信双方之间的对称会话密钥一次性有效。

为了建立会话密钥，人们已经提出了多种不同的方法，它们的基本思想我们在前几节中讨论过。我们在图 29.26 中描绘了一种最简单的方法。虽然这种方法很原始，但它有助于我们理解其他文献中的各种更加复杂的方法。

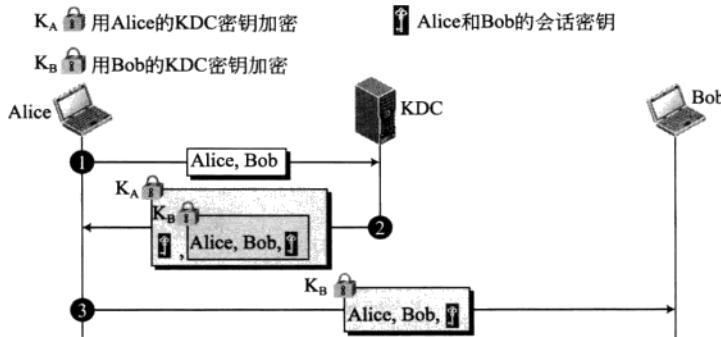


图 29.26 利用 KDC 建立会话密钥

1. Alice 向 KDC 发送明文报文，以期得到她与 Bob 之间的对称会话密钥。这个报文包含了她的注册身份（即图中的 Alice）和 Bob 的注册身份（即图中的 Bob）。这个报文不加密，是公开的。KDC 并不担心这个。

2. KDC 收到这个报文后就产生所谓的签条（ticket）。这个签条用 Bob 的密钥 (K_B) 加密。在这个签条中包含了 Alice 和 Bob 的身份，以及一个会话密钥 (K_{AB})。这个签条和该会话密钥的副本一起被发送给 Alice。Alice 收到这个报文后，将其解密，提取出会话密钥。她无法解密 Bob 的签条，因为那个签条是给 Bob 的，而不是给她的。请注意，这个报文包含了双重加密，签条是加密的，整个报文也是加密的。通过第二个报文，实际上 Alice 是向 KDC 证实自己的身份，因为只有 Alice 才能用自己与 KDC 之间的密钥打开整个报文。

3. Alice 把这个签条发送给 Bob。Bob 打开签条，并且知道了 Alice 需要用 K_{AB} 作为会话密钥给他发送报文。请注意，通过这个报文，Bob 也向 KDC 鉴别了自己，因为只有 Bob 才能打开这个签条。Bob 向 KDC 鉴别的同时也就是通过了 Alice 的鉴别，因为 Alice 信任

KDC。同理，Bob 也鉴别了 Alice，因为 Bob 信任 KDC，而 KDC 向 Bob 发送的签条中包括了 Alice 的身份。

29.9.2 对称密钥协商

Alice 和 Bob 即使不经过 KDC 也可以生成他们之间的会话密钥。这种会话密钥的产生方式称为对称密钥协商。虽然有多种方法可以达到这个目的，在这里我们只讨论一种方法，称为 Diffie-Hellman 方法，从中可以看到其他更复杂的（更加不易被攻击的）方法所使用的基本思想。

Diffie-Hellman 密钥协商

使用 **Diffie-Hellman 协议**，通信双方不需要 KDC 的帮助就能够生成对称会话密钥。在生成一个对称密钥之前，双方需要选择两个数 p 和 g 。从数论来看，这两个数具有一定的特点，但对此的讨论超出了本书范围。这两个数不需要被保密。它们可以通过因特网发送，是可公开的。图 29.27 描绘了这个过程。

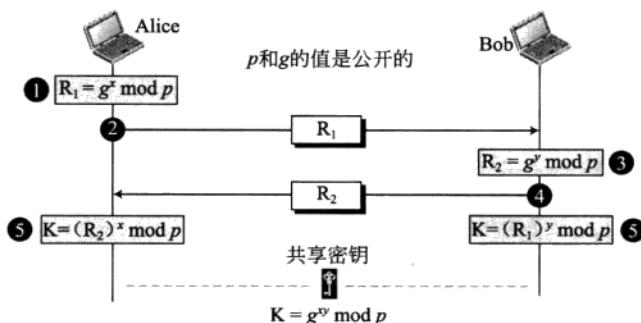


图 29.27 Diffie-Hellman 方法

步骤如下：

1. Alice 选择一个随机的大数 x ，让 $0 \leq x \leq p - 1$ ，并计算 $R_1 = g^x \bmod p$ 。
2. Alice 把 R_1 发送给 Bob。请注意，Alice 没有发送 x 的值，她只发送了 R_1 。
3. Bob 选择另一个随机的大数 y ，让 $0 \leq y \leq p - 1$ ，并计算 $R_2 = g^y \bmod p$ 。
4. Bob 把 R_2 发送给 Alice。同样要注意，Bob 没有发送 y 的值，他只发送了 R_2 。
5. Alice 计算 $K = (R_2)^x \bmod p$ 。Bob 也计算 $K = (R_1)^y \bmod p$ 。K 就是这次会话使用的对称密钥。

$$K = (g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$$

Bob 计算了 $K = (R_1)^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$ 。Alice 也计算了 $K = (R_2)^x \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$ 。双方都得到了同样的数值，而 Bob 不知道 x 的值，Alice 也不知道 y 的值。

在 Diffie-Hellman 方法中的对称（共享）密钥是 $K = g^{xy} \bmod p$ 。

例 29.8

让我们给出一个简单的例子，使这个过程更加清楚。我们的例子使用了一个较小的数，但应注意，在实际情况中这个数应当很大。假定 $g = 7$ 且 $p = 23$ 。步骤如下：

1. Alice 选择 $x = 3$ 并计算 $R_1 = 7^3 \bmod 23 = 21$ 。
2. Alice 把数 21 发送给 Bob。
3. Bob 选择 $y = 6$ 并计算 $R_2 = 7^6 \bmod 23 = 4$ 。
4. Bob 把数 4 发送给 Alice。
5. Alice 计算对称密钥 $K = 4^3 \bmod 23 = 18$ 。Bob 计算对称密钥 $K = 21^6 \bmod 23 = 18$ 。Alice 和 Bob 得出的 K 值是相同的； $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$ 。

29.9.3 公钥分配

在不对称密钥加密术中，人们无须掌握对称共享密钥。如果 Alice 想要给 Bob 发送报文，她只需要知道 Bob 的公钥，这个公钥是面向大众公开的，每一个人都可以使用。如果 Bob 要给 Alice 发送报文，他只需要知道 Alice 的公钥，而这个公钥是大家都知道的。在公钥加密术中，每一个人都隐藏了一个私钥，同时公开了一个公钥。

在公钥加密术中，人人都可以获得其他任何人的公钥；

公钥是面向大众的。

与密钥一样，公钥也要经过分配才能使用。让我们简单地讨论一下公钥的分配办法。

公告

一种比较天真的做法是公开地宣布公钥。Bob 可以把公钥放在自己的网站上，或者通过本地或全国性的报纸来公布自己的公钥。当 Alice 需要向 Bob 发送秘密报文时，她可以从 Bob 的网站或报纸上获得 Bob 的公钥，或者甚至可以发送报文向 Bob 直接索取。但是，这种方式是不安全的，很容易被假冒。例如，Eve 也可以做类似的公告。在 Bob 还没有来得及行动之前，损失已经造成了。Eve 可以愚弄 Alice，使 Alice 把报文发送给她，而不是给 Bob。Eve 也可以用相应的假私钥来签署一份文档，使得所有人都认为这个文档是由 Bob 签发的。Alice 直接向 Bob 索取公钥的方式也不牢靠。Eve 可以截获 Bob 的响应，并用自己假造的公钥来代替 Bob 的公钥。

认证管理机构

最常见的分配公钥的方式就是建立公钥证书（public-key certificate）。Bob 希望做两件事：他希望人们知道他的公钥，同时又希望没有人会把伪造的公钥当成是他的。Bob 可以去认证管理机构（certification authority, CA），这是美国的联邦或政府机构，它把一个公钥和一个实体绑定起来，并发出一个证书。图 29.28 描绘了这种概念。

CA 自己有一个大家都知的公钥，是不可能被伪造的。CA 检查 Bob 的身份（使用有照片的身份证和其他证明）。然后 CA 询问 Bob 的公钥，并写在这个证书上。为了避免这个证书本身被伪造，CA 用自己的私钥对这个证书进行签名。现在 Bob 就可以上传这个签了名的证书。任何需要 Bob 的公钥的人都可以下载得到这个被签名的证书，并且利用 CA 的公钥提取 Bob 的公钥。

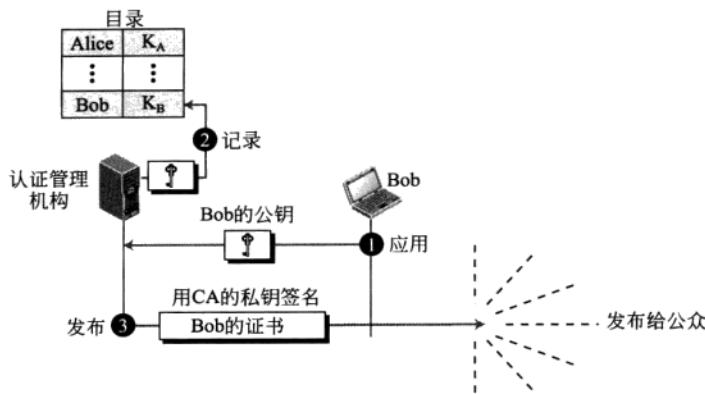


图 29.28 认证管理机构

X.509

虽然使用 CA 可以解决伪造公钥的问题，但它产生了一个副作用。每个证书可能有不同的格式。如果 Alice 想用一个程序自动下载属于不同人的证书和摘要，那么这个程序有可能无法完成任务。一种证书可能有一种格式的公钥，而另一种证书可能有另一种格式的公钥。在某证书中公钥可能在第一行，而在另一种格式中它则可能在第三行。任何需要通用的东西都必须有通用的格式。为了消除这种副作用，ITU 设计了 **X.509**，它是已经被因特网接受（有一些改动）的一个推荐协议。**X.509** 是用结构化的形式描述证书的一种方法。它使用了一种熟知的称为 ASN.1（抽象语法记法 1）的协议，这种协议对字段的定义方式是计算机程序员非常熟悉的。

29.10 深入阅读

有不少书籍对加密术和网络安全做了全面介绍。特别地，我们推荐[For 08]、[Sta 06]、[Bis 05]、[Mao 04]和[Sti 06]。

29.11 重要术语

加法加密方法
不对称密钥加密方法
攻击
自动密钥加密方法
有效性
面向位的加密方法
块加密方法
恺撒加密方法

认证管理机构（CA）
查问-响应鉴别
面向字符的加密方法
加密方法
密文
循环位移操作
组合操作
压缩函数

保密性	口令
加密术	P 盒
加密散列函数	明文
数据加密标准 (DES)	多态字符加密方法
解密	多态字符替代
解密算法	私钥
拒绝服务	公钥
Diffie-Hellman 协议	公钥证书
摘要	重放
数字签名	否认
数字签名机制	RSA 加密系统
数字签名标准 (DSS)	RSA 数字签名机制
加密	S 盒
加密算法	安全散列算法 (SHA)
实体鉴别	会话密钥
HMAC	共享密钥
完整性	位移加密方法
重复加密散列函数	签名算法
密钥	窃取
密钥分配中心 (KDC)	分裂操作
伪装	隐密术
MD2	流加密方法
MD4	替代加密方法
MD5	交换操作
报文鉴别码 (MAC)	对称密钥加密方法
现代块加密方法	签条
现代流加密方法	通信量分析
更改	置换加密方法
单态字符加密方法	验证算法
一次填充	X.509

29.12 本章小结

- 信息是具有价值的财产，需要安全保护。信息需要对未授权的访问来说是隐蔽的（保密性）；对未授权的修改来说是被保护的（完整性）；而对授权的实体来说，只需要就能得到（有效性）。我们的三个安全目标可能会受到安全攻击的威胁。人们已设计出两种用以保护信息不受攻击的技术：加密术和隐密术。
- 传统加密方法称为对称密钥加密方法，因为加密和解密时使用了相同的密钥，并且

这个密钥可用于双向通信。我们可以把传统的对称密钥加密方法划分为两大类：替代加密方法和转换加密方法。替代加密方法就是用另一个字符来替代一个字符。转换加密方法就是对字符重新排序。

- 现代加密方法是面向比特的加密方法。现代加密方法可以是块加密方法，也可以是流加密方法。现代块加密方法使用了多次由替代、转换、XOR 以及其他混合块位操作构成的组合。目前最常用的一种块加密方法是 DES。现代流加密方法一次一比特地对比特流进行加密和解密。
- 不对称密钥加密方法使用了两个独立的密钥：私钥和公钥。不对称密钥加密术意味着 Bob 和 Alice 不能把一套密钥用于双向通信。Bob 只需要一个私钥就能接收来自团体内任何人传来的讯息，但是 Alice 则需要 n 个公钥才能与团体中的 n 个实体进行通信。
- 保持文档完整性的一种办法就是通过使用报文摘要。报文通过一个称为加密散列函数的算法的处理。这个函数产生报文的一个压缩的印记，称为摘要，可以像使用手印一样使用它。
- 为了确保报文的完整性以及数据来源的鉴别，我们需要创建报文鉴别码（MAC），就是通过散列函数来加入 Alice 和 Bob 之间的密钥。另一种提供了报文完整性和报文鉴别的方法是使用数字签名。MAC 用密钥来保护摘要，而数字签名则使用了一对私钥-公钥来做同样的事。
- 实体鉴别是为了让一方证实另一方的身份而设计的一种技术。实体可以是人、进程、客户程序或服务器程序。需要被证实其身份的实体称为申请者，而试图证实申请者身份的实体称为验证者。
- 为了使用对称密钥和不对称密钥加密术，我们需要管理密钥。在对称密钥加密术中，我们可以使用 KDC 的服务来产生两个实体之间的会话密钥。在不对称密钥加密术中，我们可以使用认证管理机构（CA）的服务来发布经认证的公钥。

29.13 实践安排

29.13.1 习题

1. 定义以下各种情况的攻击类型：
 - a. 某学生闯入教授的办公室，并得到了下次考试试卷的副本。
 - b. 某学生用一张 10 美元的支票买了一本二手书。后来这个学生发现这张支票一共支取了 100 美元现金。
 - c. 某学生用假的电子邮箱地址每天向学校发送上百个电子邮件。
2. 一个小型私人俱乐部有 100 个会员，试回答以下问题：
 - a. 如果这个俱乐部的所有会员都要互相发送秘密报文，那么一共需要多少个密钥？
 - b. 如果每个人都信任俱乐部的主席，那么需要多少个密钥？如果一个会员想要发送报文给另一个会员，她先把报文发送给主席，再由主席把报文发送给另一个

会员。

- c. 如果主席决定需要通信的两个会员必须首先与他联系，然后由他创建一个临时密钥用于两个会员之间的通信，这个临时密钥在加密后被发送给双方，那么共需要多少个密钥呢？
3. Alice 要给朋友发送报文，但她的计算机上只能使用加法加密方法。她认为如果对报文加密两次，且每次使用不同的密钥，报文的安全程度就更好。她的想法对吗？请说明为什么？
4. 用 $key = 20$ 的加法加密方法对报文 “this is an exercise” 进行加密。忽略字与字之间的空格。并对报文解密以获得原始的明文。
5. a. 试给出字 $(10011011)_2$ 进行 3 位循环左移后的结果。
b. 若对 a 题结果得到的字进行 3 位循环右移后又会得到什么？
c. 试比较 b 题结果得到的字与 a 题中原始的字。
6. a. 交换字 $(10011011)_2$ 。
b. 对 a 题的结果再次交换。
c. 试比较 b 题的结果与 a 题的结果，以说明交换操作是自反的操作。
7. 试给出下列操作的结果：
a. $(01001101) \oplus (01001101)$
b. $(01001101) \oplus (10110010)$
8. 一个 4×3 的 S 盒的最左边一位决定了其他三位的旋转方向。如果最左边一位是 0，则其他三位向右旋转 1 位（循环右移）。如果最左边一位是 1，则其他三位向左旋转 1 位（循环左移）。如果输入是 1011，则输出是什么？如果输入是 0110，输出又是什么？
9. 在 RSA 中，假设 $n = 12091$, $e = 13$ 且 $d = 3653$ ，请用 00-26 编码机制对报文 “THIS IS TOUGH” 编码后加密，并对密文解密后得到原始报文。明文或密文使用 4 个数字为一个块。
10. 在 RSA 中，为什么 Bob 不能选择 1 作为公钥 e ？
11. 请解释为什么在创建 MAC 时不能使用私钥-公钥对？
12. 假设我们有一个非常简单的报文摘要。我们的这个假想的报文摘要就是 0 到 25 之间的一个数字。初始时将摘要设置为 0。加密散列函数就是在当前的摘要值上再加当前字符的值（0 到 25 之间）。这里的加法是模 26 的加法。如果报文是 “HELLO” 的话，摘要的值是什么？为什么这个摘要不安全？
13. 修改图 29.22，使得 Alice 和 Bob 能够互相鉴别。
14. 修改图 29.23，使得 Alice 和 Bob 能够互相鉴别。
15. 修改图 29.24，使得 Alice 和 Bob 能够互相鉴别。

29.13.2 研究活动

16. 利用文献找出历史上的 Vigenere 多态字符加密方法。
17. 有一种很有趣的传统加密方法称为 Hill 加密方法。利用文献找出有关这种加密方法的更多知识。

18. 我们经常会听说 Feistel 加密方法和 non-Feistel 加密方法。利用文献或因特网找出这两类加密方法的主要区别是什么？DES 又是属于哪一类的呢？
19. 因为 DES 中的加密方法密钥长度都不大（只有 56 位），所以人们今天使用了三重 DES 块加密方法。利用文献找出我们如何将 DES 改为只有两个密钥的三重 DES。在三重 DES 中，密钥的长度是多少？
20. 高级加密标准（Advanced Encryption Standard, AES）是一种新的现代块加密方法。利用文献学习有关这种加密方法的知识，并将它与 DES 进行比较。AES 是一种 Feistel 加密方法，还是一种 non-Feistel 加密方法？
21. 另外一种不对称密钥加密方法称为 ElGamal。利用文献找出有关这种加密方法的知识，并将其与 RSA 比较。
22. 利用文献找出 HMAC 的总体概况（与图 29.17 中的 MAC 类似）。
23. 一种普遍被人们看好的加密散列函数是 Whirlpool。利用文献找出与这个函数相关的一些知识。这个函数与 AES 块加密方法之间有着什么关系？
24. 利用文献找出更多关于 DSS 的知识。
25. 在 UNIX 中使用口令时有一种非常有趣的想法，就是 salt 口令。利用文献找出有关 salt 口令的情况。
26. 利用文献找出有关 Lamport 一次性口令的资料。
27. 实体鉴别目前最热门的是有关“零知识（zero knowledge）”的话题。利用文献研究有关这个话题的内容。它是如何在实体鉴别中使用的？
28. 一种常见的很有意思的 KDC 是称为 Kerberos 的协议。利用因特网对它进行研究，并将其与我们在本章讨论的简单 KDC 机制进行比较。
29. 我们在本章使用的密钥管理协议 Diffie-Hellman 并不是很安全。它会受到“中间人”的攻击。人们已经定义了另一种协议称为“站到站的协议”，它可以提高 Diffie-Hellman 的安全性。查找有关这个协议的资料，并将其与 Diffie-Hellman 比较。

第 30 章 因特网安全

我们在第 29 章讨论的各种技术经过组合后就可以提供因特网的网络层、运输层以及应用层的安全服务。对这些技术的讨论及应用非常复杂，要涉及多个协议和数十种不同的分组。虽然详细地研究这些协议和分组是一件十分有趣的事，但真要这么做的话，即使不需要上千页，也需要好几百页的内容。在本书的最后一章，我们对这些协议的介绍不过是惊鸿一瞥，以便为读者对此内容做更进一步的阅读打好基础。

目标

本章有以下几个目标：

- 介绍因特网网络层的安全思想以及 IPSec 协议，它通过两种方式来实现这种思想：运输方式和隧道方式。
- 讨论 IPSec 中的两个协议：AH 和 ESP，并解释它们各自提供的安全服务。
- 介绍安全关联以及它在 IPSec 中的应用。
- 介绍虚拟专用网（VPN），它是 IPSec 隧道方式的应用。
- 介绍因特网运输层的安全思想以及用以实现这种思想的 SSL 协议。
- 说明 SSL 如何生成客户和服务器所使用的六个加密密钥/参数。
- 讨论在 SSL 中使用的四个协议以及它们之间的相互关系。
- 介绍因特网应用层的安全思想以及实现这种思想的两个协议：PGP 和 S/MIME。
- 说明 PGP 和 S/MIME 如何提供保密和报文鉴别能力。
- 讨论防火墙以及如何用它们来保护站点不被入侵。

30.1 网络层安全

我们从网络层的安全来开始这一章的内容。虽然在后面的内容中我们会讨论运输层和应用层的安全，但是网络层的安全同样必不可少，原因有三个。首先，不是所有的客户/服务器程序都会在应用层受到保护。其次，正如我们在讨论运输层时所提到的，不是所有应用层的客户/服务器程序都使用了 TCP 服务，有些程序使用的是 UDP 的服务，而只有使用 TCP 服务的程序才能受到运输层安全的保护。第三，像路由选择协议这样的应用，它们直接使用了 IP 服务，因此需要 IP 层的安全。

IP 安全（IP Security, IPSec）是因特网工程部（IETF）设计的一组协议，用来为网络层的分组提供安全。IPSec 为 IP 层生成鉴别的和保密的分组提供帮助。

30.1.1 两种方式

IPSec 有两种工作方式：运输方式和隧道方式。

运输方式

使用运输方式（transport mode）时，IPSec 保护由运输层交付给网络层的东西。换言之，运输方式保护的是在网络层封装的有效载荷，如图 30.1 所示。

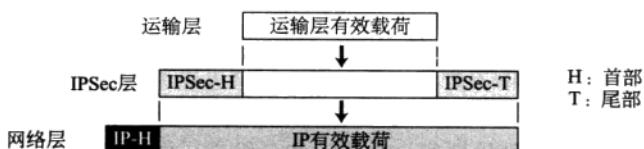


图 30.1 运输方式下的 IPSec

请注意，运输方式不保护 IP 首部。换言之，运输方式不是保护整个 IP 分组，它只保护运输层的分组（即 IP 层的有效载荷）。在这种方式下，IPSec 的首部（以及尾部）先被添加到从运输层传来的信息上，之后再添加 IP 首部。

工作在运输方式下的 IPSec 不保护 IP 首部；

它只保护从运输层传来的信息。

通常，当我们需要从主机到主机（端到端）的数据保护时就使用运输方式。发送主机用 IPSec 来鉴别和/或加密从运输层传来的有效载荷。接收主机用 IPSec 来检查鉴别和/或解密这个 IP 分组，然后将其交付给运输层。图 30.2 描绘了这一概念。

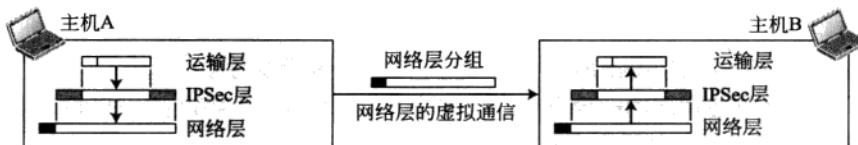


图 30.2 工作中的运输方式

隧道方式

使用隧道方式（tunnel mode）时，IPSec 保护整个 IP 分组。它先取一个 IP 分组（包括首部），在整个分组上应用 IPSec 安全方法，然后再增加新的 IP 首部，如图 30.3 所示。

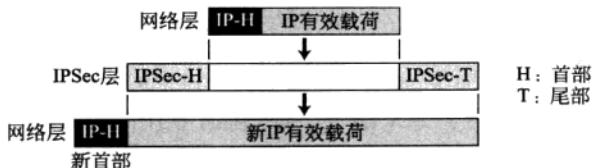


图 30.3 隧道方式下的 IPSec

稍后我们将会看到，这个新的 IP 首部与原始的 IP 首部相比有一些不同的信息。隧道方式通

常用在两个路由器之间、主机和路由器之间或者路由器和主机之间，如图 30.4 所示。发送方和接收方之间的整个原始分组都会受到保护防止入侵，就像整个分组通过一条假想的隧道一样。

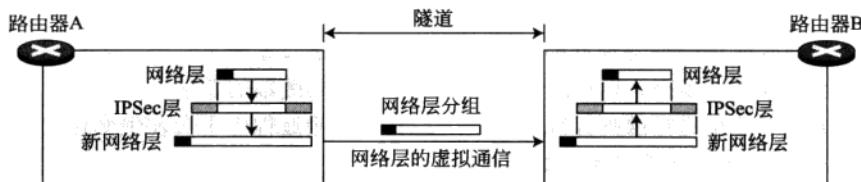


图 30.4 工作中的隧道方式

工作在隧道方式下的 IPSec 会保护原始 IP 首部。

比较

在运输方式下，IPSec 层位于运输层和网络层之间。在隧道模式下，整个数据流是从网络层到 IPSec 层，然后再返回网络层。图 30.5 对这两种方式进行了比较。



图 30.5 运输方式和隧道方式之比较

30.1.2 两个安全协议

IPSec 定义了两个协议——鉴别首部（AH）协议和封装安全有效载荷（ESP）协议——以提供 IP 级分组的鉴别和加密。

鉴别首部

鉴别首部（Authentication Header, AH）协议的设计是为了鉴别源主机的身份并确保 IP 分组所携带的有效载荷的完整性。这个协议使用散列函数和对称密钥产生报文摘要，再将报文摘要插入到鉴别首部中（参见第 29 章的 MAC）。然后，根据所采用的方式（运输方式或隧道方式），将 AH 插入到适当的位置。图 30.6 给出了运输方式中鉴别首部的各个字段和位置。

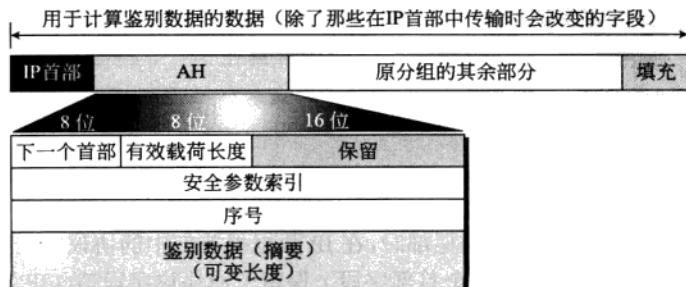


图 30.6 鉴别首部 (AH) 协议

当 IP 数据报携带了鉴别首部时，IP 首部协议字段中原来的数值被更改为 51。而鉴别首部中有一个字段（即下一个首部字段）保留了协议字段原来的数值（即这个 IP 数据报携带的有效载荷类型）。添加鉴别首部要经过以下几个步骤：

1. 在有效载荷上添加鉴别首部，其中的鉴别数据字段置为零。
2. 可能需要添加填充，以使总长度满足特定的散列函数的要求。
3. 根据整个分组进行散列计算。不过，只有在传输过程中不会发生变化的 IP 首部的字段才会被包含在报文摘要（鉴别数据）的计算中。
4. 在鉴别首部中插入鉴别数据。
5. 把协议字段值修改为 51 之后，加上这个 IP 首部。

下面是对每个字段的简要描述。

- **下一个首部** 这个 8 位的下一个首部字段定义了 IP 数据报携带的有效载荷类型（如 TCP、UDP、ICMP、OSPF 等等）。
- **有效载荷长度** 这个 8 位字段的名字有误导之嫌。它不是定义了有效载荷的长度，而是定义了鉴别首部的长度，以 4 字节的倍数计算，但不包含最前面的 8 个字节。
- **安全参数索引** 这个 32 位的安全参数索引（SPI）字段起着虚电路标识符的作用，并在一个称为安全关联（稍后讨论）的连接期间对所有的分组都保持不变。
- **序号** 这个 32 位的序号对数据报的序列提供排序信息。序号可防止重放。请注意，即使分组重传时序号也不重复。当序号到达 2^{32} 时也不回绕，而是必须建立新的连接。
- **鉴别数据** 最后，鉴别数据字段就是把散列函数应用到整个 IP 数据报所得到的结果，除了那些在传输过程中会变化的字段（例如，寿命）。

AH 协议提供报文鉴别和完整性，但不提供保密。

封装安全有效载荷

AH 协议不提供保密性，它只提供报文鉴别和完整性。IPSec 后来又定义了另一个类似的协议，称为封装安全有效载荷（Encapsulating Security Payload, ESP），它可以提供源的鉴别、完整性以及保密性。ESP 添加了首部和尾部。请注意，ESP 的鉴别数据被放在分组的尾部中，这是为了使计算更加方便。图 30.7 给出 ESP 首部和尾部的位置。

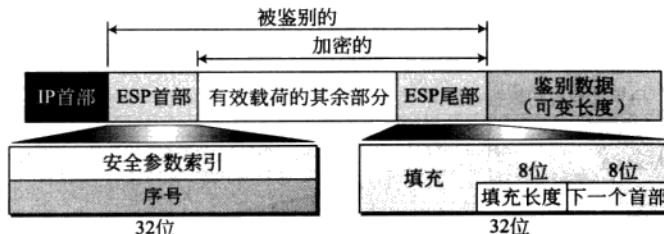


图 30.7 封装安全有效载荷（ESP）

当 IP 数据报携带了 ESP 首部和尾部时，在 IP 数据报首部中的协议字段值应更改为 50。在 ESP 尾部中有一个字段（即下一个首部字段）保留了原协议字段值（IP 数据报携带的有效载荷类型，如 TCP 或 UDP）。ESP 处理过程的步骤如下：

1. 给有效载荷添加 ESP 尾部。
2. 对有效载荷和尾部加密。
3. 添加 ESP 首部。
4. 使用 ESP 首部、有效载荷以及 ESP 尾部共同生成鉴别数据。
5. 鉴别数据被放入 ESP 尾部的末端。
6. 把协议字段值修改为 50 之后，加上这个 IP 首部。

ESP 首部和尾部的各个字段如下：

- 安全参数索引** 这个 32 位的安全参数索引字段和 AH 协议中的定义相似。
- 序号** 这个 32 位的序号字段和 AH 协议中的定义相似。
- 填充** 这是一个可变长度字段（0~255 字节），全部为 0，用做填充。
- 填充长度** 这个 8 位的填充长度字段定义了填充的字节数。它的值是 0~255。但是，填充很少会用到最大值。
- 下一个首部** 这个 8 位的下一个首部字段和 AH 协议定义的相似。它与封装前 IP 首部中的协议字段的功能一样。
- 鉴别数据** 最后，鉴别数据字段是把鉴别机制应用到数据报的各部分的结果。请注意 AH 和 ESP 的鉴别数据的区别。在 AH 中，部分 IP 首部也包含在计算鉴别数据之中，但在 ESP 中则没有。

ESP 提供报文鉴别、数据完整性和保密。

IPv4 和 IPv6

IPSec 支持 IPv4 和 IPv6。但在 IPv6 中，AH 和 ESP 都是扩展首部的一部分。

AH 与 ESP 之比较

ESP 协议是在 AH 协议已经使用后才开始设计的。ESP 能做所有 AH 能做的事，另外还增加了保密性。问题是：为什么我们还需要 AH？答案是：我们并不需要它。但是，AH 的实现已经包含在一些产品中。这就表示，AH 还会继续是因特网的一部分，直到这些产品被完全淘汰为止。

30.1.3 IPSec 提供的服务

AH 和 ESP 这两个协议能够在网络层为分组提供若干安全服务。表 30.1 所示为每种协议可提供的服务列表。

表 30.1 IPSec 服务

服 务	AH	ESP
接入控制	是	是
报文鉴别（报文完整性）	是	是
实体鉴别（数据源鉴别）	是	是
保密性	否	是
对重放攻击的保护	是	是

接入控制

通过使用安全关联数据库（SAD），IPSec 能够间接地提供接入控制，我们将在下一节讨论。当一个分组到达终点，但却没有找到为该分组已建立好的安全关联时，这个分组被丢弃。

报文完整性

AH 和 ESP 都能保护报文的完整性。发送方会产生数据摘要发送出去，由接收方进行检查。

实体鉴别

在 AH 和 ESP 中，安全关联以及由发送方发送的密钥散列摘要实现了对数据发送者的鉴别。

保密性

在 ESP 中，对报文的加密就提供了保密性。不过 AH 不提供保密性。如果需要保密，就应当使用 ESP，而不是 AH。

对重放攻击的保护

在使用这两个协议时，都可以通过序号和接收方的滑动窗口来防止重放攻击。在安全关联已建立的情况下，每个 IPSec 首部中都包含了一个唯一的序号。这个序号从 0 开始，不断递增，直至达到 $2^{32} - 1$ 。当序号达到最大值时，它被复位为 0，同时旧的安全关联（参见下一节）被删除，建立新的安全关联。为了防止处理重复分组，IPSec 强制接收方使用固定大小的窗口。窗口大小由接收方决定，默认值是 64。

30.1.4 安全关联

安全关联是 IPSec 一个非常重要的内容。IPSec 需要在两个主机之间建立一个逻辑关系，称为安全关联（Security Association，SA）。这一小节首先要讨论它的基本思想并说明如何在 IPSec 中使用它。

安全关联的思想

安全关联就是通信双方之间的一份协约，也就是在双方之间建立一条安全的信道。假设 Alice 需要与 Bob 单向通信。如果 Alice 和 Bob 只对保密这个安全问题感兴趣，那么就在他们之间建立一个共享密钥。我们可以说在 Alice 和 Bob 之间有两个安全关联（SA）：一个出 SA，一个入 SA。两个人都把密钥的值保存在变量中，并保存与对方之间的加密/解密算法。Alice 用算法和密钥把发送给 Bob 的报文加密，Bob 在需要时就用相应的算法和密钥对来自 Alice 的报文进行解密。图 30.8 描绘了简单的 SA。

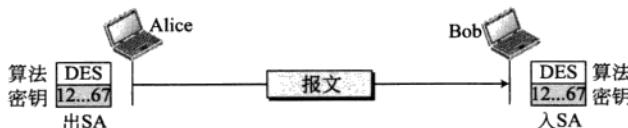


图 30.8 简单 SA

如果双方还需要报文的完整性及鉴别，那么其安全关联就会更加复杂一些。每个关联还需要其他一些数据，如报文完整性的算法、密钥以及其他一些参数。如果双方还需要为不同的协议（如 IPSec AH 或 IPSec ESP）使用特定的算法和特定的参数，事情就会变得复杂得多。

安全关联数据库（SAD）

安全关联可以非常复杂，特别是当 Alice 希望向很多人发送报文，而 Bob 又希望接收来自很多人的报文时，情况尤其严重。另外，每个站点既要有入 SA，也要有出 SA，以便允许双向通信。换言之，我们需要的是能够被集中保存在数据库中的一组 SA。这个数据库称为安全关联数据库（Security Association Database，SAD）。安全关联数据库可以被看成是一张二维表格，其中每一行定义了一个 SA。通常有两种 SAD，一个入，一个出。图 30.9 描绘了每个实体的出 SAD 或入 SAD 的概念。

索引	SN	OF	ARW	AH/ESP	LT	Mode	MTU
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							

安全关联数据库

图例

SPI：安全参数索引 SN：序列号
 DA：目的地址 OF：溢出标志
 AH/ESP：这两种协议分别对应的信息 ARW：反重放攻击窗口
 P：协议 LT：寿命
 Mode：IPSec方式标志 MTU：路径MTU

图 30.9 SAD

当主机需要发送一个必须携带 IPSec 首部的分组时，该主机需要在出 SAD 中查找相应的表项，以便获得信息对该分组实施安全保护。同样，当主机接收到一个携带有 IPSec 首部的分组时，它要在入 SAD 中查找相应的表项，以便获得信息来检查该分组的安全性。这个查找必须具体到能够让接收主机确定找到的信息就是用于处理该分组的正确信息。因此在入 SAD 中的每个表项都要用三个索引来选择：安全参数索引（一个 32 位的值，它定义了在终点上的 SA）、目的地址以及协议（AH 或 ESP）。

安全策略

IPSec 的另一个重要内容是安全策略（Security Policy，SP），它定义了分组发送时或到达时被应用于该分组的安全类型。在使用 SAD 之前，主机必须先判断为该分组预先定义的策略是什么。

安全策略数据库

使用 IPSec 协议的主机需要维护一个安全策略数据库（Security Policy Database，SPD）。同样，我们也需要一个入 SPD 和一个出 SPD。SPD 中的表项需要用六个索引来访问：源地址、目的地址、名称、协议、源端口和目的端口，如图 30.10 所示。

源地址和目的地址可以是单播、多播或任播地址。名称通常定义了一个 DNS 实体。协议不是 AH 就是 ESP。源端口和目的端口是在源主机和终点主机上运行的进程的端口地址。

出 SPD 当一个分组即将被发送出去时，就需要咨询出 SPD。图 30.11 描绘了发送方对分组的处理过程。

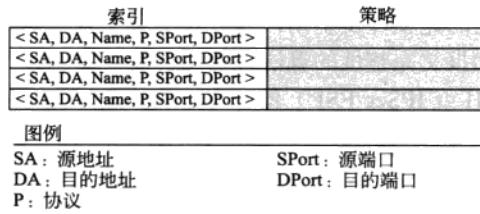


图 30.10 SPD

出 SPD 的输入是六个索引值，输出是以下三种情况之一：丢弃（分组不能发送）、旁路（旁路安全首部）以及应用（根据 SAD 来实现相应的安全保护，如果没有 SAD，则创建一个新的表项）。

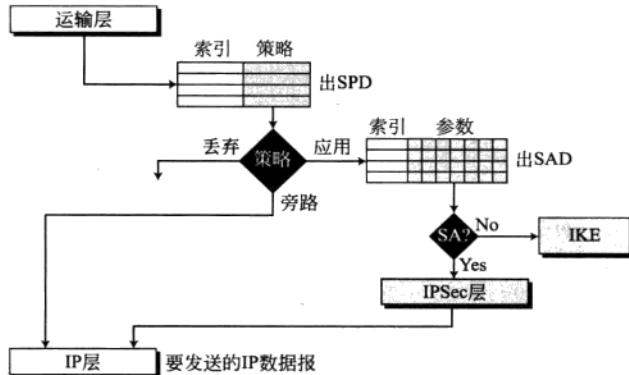


图 30.11 输出处理过程

入 SPD 当一个分组到达时，就需要咨询入 SPD。入 SPD 中的表项同样也要以六个索引来访问。图 30.12 描绘了接收方对分组的处理过程。入 SPD 的输入是六个索引值，输出是以下三种情况之一：丢弃（丢掉该分组）、旁路（旁路安全检查，直接将分组交付给运输层）以及应用（根据 SAD 实施相应的安全策略）。

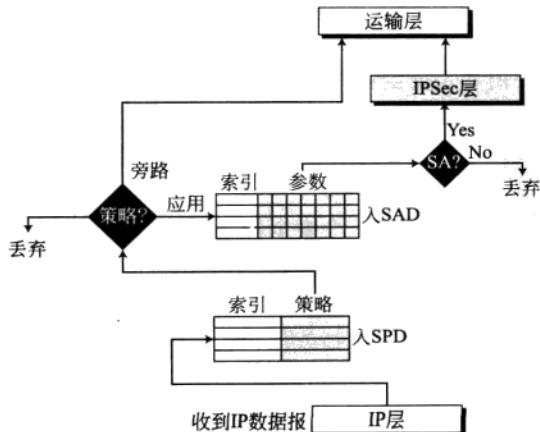


图 30.12 输入处理过程

30.1.5 因特网密钥交换 (IKE)

因特网密钥交换 (Internet Key Exchange, IKE) 是为了建立入安全关联和出安全关联而设计的一个协议。正如我们在前一小节中讨论的，当对等的双方需要发送 IP 分组时，它会去咨询安全策略数据库 (SPD)，看看对于此通信量是否存在 SA。如果没有相应的 SA，就要调用 IKE 来建立一个。

IKE 为 IPSec 建立 SA。

IKE 是一个复杂的协议，它以另外三个协议为基础：Oakley、SKEME 和 ISAKMP，如图 30.13 所示。

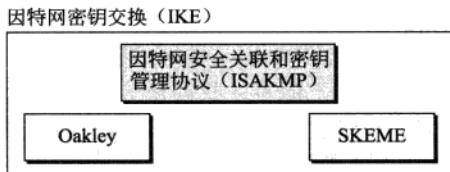


图 30.13 IKE 的组成

Oakley 协议是由 Hilarie Orman 研发的。它是一个密钥生成协议。**SKEME** 是由 Hugo Krawcyzk 设计的另一种用于密钥交换的协议。它利用公钥加密来实现密钥交换协议中的实体鉴别。

因特网安全关联和密钥管理协议 (Internet Security Association and Key Management Protocol, ISAKMP) 是由美国国家安全署 (NSA) 设计的一个协议，它实际上就是实现了 IKE 中定义的密钥交换。这个协议定义了若干个分组、协议及参数，这就使得 IKE 的交换能够以标准化、格式化的报文建立 SA。我们把对这三个协议的讨论留给专门介绍安全内容的书籍。

30.1.6 虚拟专用网 (VPN)

IPSec 的一个应用就是虚拟专用网。虚拟专用网 (virtual private network, VPN) 这种技术越来越受到大型组织的青睐，这些组织利用全球因特网来实现组织内部和组织之间的通信，同时又要求在它们的组织内部的通信是保密的。VPN 是专用却虚拟的网络。说它是专用的，因为它保证在组织内部通信的保密性；说它是虚拟的，因为它并没有使用真正的专用广域网，这个网络从物理上来说还是公共的，但从虚拟网络的角度看是专用的。图 30.14 描绘了虚拟专用网的概念。路由器 R1 和 R2 使用 VPN 技术来保证组织内部的保密性。VPN 技术通过隧道方式使用 IPSec 的 ESP 协议。它把一个保密的数据报（包括它的首部）封装在 ESP 分组中。发送站点的边界路由器利用自己的 IP 地址和目的站点路由器的地址生成新的数据报。公共网络（因特网）负责把分组从 R1 传送到 R2。无关人员无法解密分组内容或源地址和目的地址。解密发生在 R2 上，由 R2 找出分组的目的地址并交付分组。

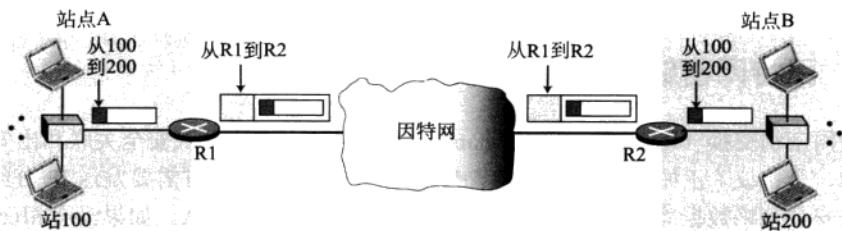


图 30.14 虚拟专用网络

30.2 运输层安全

目前提供了运输层安全的协议主要有两个：**安全套接层**（Secure Sockets Layer, SSL）协议和**运输层安全**（Transport Layer Security, TLS）协议。后一种协议实际上是前一种协议的 IETF 版本。我们在这一节只讨论 SSL, TLS 与此类似。

图 30.15 描绘了 SSL 和 TLS 在因特网模型中所处的位置。



图 30.15 SSL 和 TLS 在因特网模型中的位置

这两个协议的目标之一就是提供对服务器和客户的鉴别以及数据的保密和完整性。使用了 TCP 服务的应用层客户/服务器程序，像 HTTP（参见第 22 章），可以把它们的数据封装在 SSL 分组中。如果服务器和客户程序能够运行 SSL（或 TLS）程序，那么客户就允许使用 <https://...>这样的 URL，以取代 <http://...>，这使得 HTTP 报文能够被封装在 SSL（或 TLS）分组中。例如，网上购物者的信用卡号就能够安全地通过因特网传输。

30.2.1 SSL 的体系结构

SSL 的设计是为了给应用层生成的数据提供安全及压缩服务。一般来说 SSL 能够接收来自任何应用层协议的数据，但是通常这个协议就是 HTTP。从应用程序传来的数据经过压缩（可选）、签名和加密，然后再被传递给可靠的运输层协议，如 TCP。Netscape 公司于 1994 年研发出了 SSL。它的版本 2 和版本 3 于 1995 年发布。在这一小节，我们讨论 SSLv3。

服务

SSL 对应用层传来的数据提供多种服务。

- 分片** SSL 把数据划分为长度小于或等于 2^{14} 字节的数据分片。
- 压缩** 数据分片通过使用一种由客户和服务器协商好的无损压缩方式进行压缩。这个服务是可选的。
- 报文完整性** 为了保护数据的完整性，SSL 使用密钥散列函数来创建 MAC（参见第 29 章）。
- 保密** 为了提供保密性，原始的数据和 MAC 一起用对称密钥加密术加密。
- 组帧** 先在被加密的有效载荷上添加一个首部，然后再把这个有效载荷传递给可靠的运输层协议。

密钥交换算法

为了交换经过鉴别和保密的报文，客户和服务器程序各需要一组加密用的密钥/参数。但是，为了产生这些密钥/参数，双方之间必须先产生一个前主密（pre-master secret）。SSL 定义了多种密钥交换方式，用于建立这个前主密。

加密/解密算法

客户和服务器程序还需要协商同意使用一组加密和解密算法。

散列算法

SSL 使用散列算法来提供报文完整性（报文的鉴别）。为此已定义了几种散列算法。

加密方法族

把密钥交换、散列以及加密算法合在一起就为 SSL 会话定义了一个加密方法族（cipher suite）。

压缩算法

在 SSL 中，压缩是可选的，它没有定义具体的压缩算法。因此，系统可以根据自己的意愿使用任何压缩算法。

加密参数的生成

为了实现报文的完整性和保密性，SSL 需要六个加密用的密钥/参数：四个密钥和两个 IV（初始向量）。客户需要一个用于报文鉴别的密钥、一个用于加密的密钥，以及一个在计算时用做初始块的 IV。服务器程序也有相同的需求。SSL 要求两个方向的密钥各不相同。如果在一个方向上受到攻击，另一个方向上不会受到影响。这些参数的生成采用以下步骤：

1. 客户和服务器先交换两个随机数：一个由客户生成，另一个由服务器生成。
2. 客户和服务器使用预先定义的密钥交换算法来交换一个前主密（pre-master secret）。
3. 通过应用两个散列函数（SHA-1 和 MD5），从前主密中生成 48 字节的主密（master secret），如图 30.16 所示。

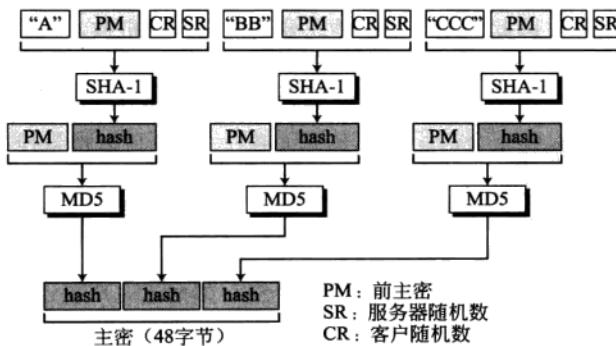


图 30.16 从前主密中计算获得主密

4. 通过应用同一组散列函数以及在主密的前面附加不同常量的办法，可以从这个主密中产生变长的密钥材料（key material），如图 30.17 所示。这个模块不断重复，直至产生适当长度的密钥材料为止。

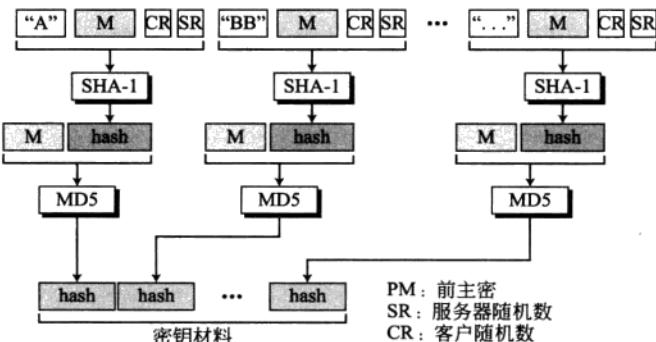


图 30.17 从主密中计算获得密钥材料

请注意，密钥材料块的长度取决于所选的加密方法族以及这个加密方法族需要的密钥长度。

5. 从这个密钥材料中提取六个不同的密钥/参数，如图 30.18 所示。

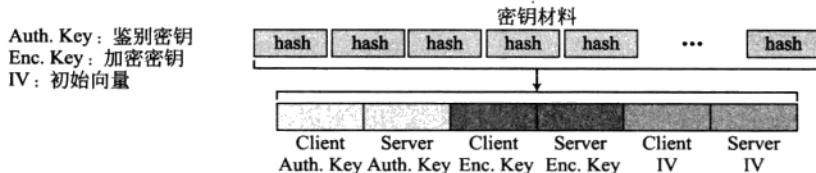


图 30.18 从密钥材料中提取加密用的密钥/参数

会话和连接

SSL 区分连接（connection）和会话（session）。会话是客户和服务器之间的关联。在会话建立后，双方就具有一些共同的信息，比如会话标识符、各自的鉴别证书（如有必要）、压缩方法（如有必要）、密码族以及用于产生报文鉴别密钥和加密密钥的一个主密。

对两个需要交换数据的实体来说，会话的建立是必需的，但并不是充分的，它们还需要在双方之间建立连接。因此，这两个实体还要交换两个随机数，并利用主密来产生一组密钥和参数，用以交换涉及到鉴别和保密的报文。

一个会话可以包含多个连接。双方之间的连接可以在同一会话内终止和重新建立。当连接终止后，双方也可以终止会话，但这不是强制的。会话可以被悬挂，并重新启动。

30.2.2 四个协议

我们已经讨论了 SSL 的思想，但还没有说明 SSL 如何来完成自己的任务。SSL 定义了两层共四个协议，如图 30.19 所示。

其中记录协议是一个载体，它运载来自其他三个协议的报文以及来自应用层的数据。从记录协议传来的报文就是运输层（通常为 TCP）的有效载荷。握手协议为记录协议提供安全参数。它要建立加密集并提供密钥和安全参数。它还要负责客户对服务器的鉴别以及服务器对客户的鉴别（如有必要）。改变加密规约协议用于通知各项加密用密钥/参数已准备好了。告警协议用于报告异常状态。我们将在这一小节简单地讨论这几个协议。

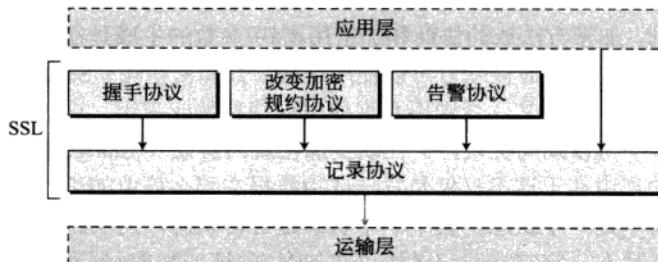


图 30.19 四个 SSL 协议

握手协议

握手协议（Handshake Protocol）通过报文来协商将要使用的加密方法族、为客户鉴别服务器以及为服务器鉴别客户（如有必要），并交换那些用于建立加密用密钥/参数的信息。握手过程分为四个阶段，如图 30.20 所示。

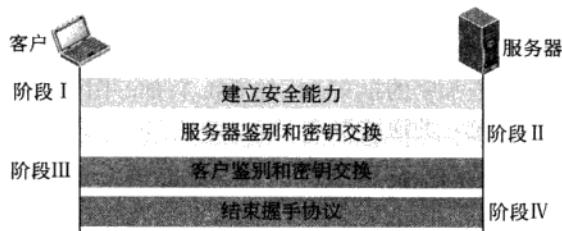


图 30.20 握手协议

阶段 I：安全能力的建立 在阶段 I，客户和服务器各自宣布它们的安全能力，并选择双方都方便使用的安全能力。在这个阶段要建立会话 ID 并选择加密方法族。双方要对特定的压缩方法达成一致意见。最后还要选择两个随机数（客户和服务器各选择一个），用于生成一个主密，如我们在前面所讨论的。

在第 I 阶段之后，客户和服务器都知道了要使用的 SSL 的版本，加密的算法，压缩的方法以及用于密钥生成的两个随机数。

阶段 II：服务器密钥的交换和鉴别 在阶段 II，如果需要的话服务器可以向客户鉴别自己的身份。服务器可以向客户发送自己的证书和公钥，同时也可请求客户的身份认证。

在第 II 阶段之后，客户鉴别了服务器的身份，并且如有必要的话，客户也可以获得服务器的公钥。

阶段 III：客户密钥的交换和鉴别 阶段 III 的设计是用来鉴别客户的。

在第 III 阶段之后，服务器鉴别了客户的身份，并且客户和服务器都掌握了一个前主密。

阶段 IV：完成和结束 在阶段 IV，客户和服务器互相发送报文以更改加密规约，并结束握手协议。

改变加密规约协议

我们已经看到，加密方法族的协商和加密用密钥/参数的生成是在握手协议期间逐步完成的。现在的问题是：什么时候双方可以使用这些参数和密钥呢？SSL 强制规定除非双方发送或接收了一个特殊的报文，否则不允许使用这些参数或密钥。这个报文就是改变加密规约报文，它在握手协议期间交换，但在改变加密规约协议（ChangeCipherSpec Protocol）中定义。这样做的理由在于这不仅仅是发送或接收报文那么简单的事，发送方和接收方需要有两个状态，而不是一个状态。其中一个状态是等待状态，用于跟踪密码参数和密钥，另一个状态是活跃状态，它掌握着记录协议在签名/验证或加密/解密报文时要使用的参数和密钥。另外，每个状态都要保存两组值：读（入）和写（出）。

告警协议

SSL 通过告警协议（Alert Protocol）来报告差错和异常状态。它只使用了一个报文，这个报文描述了问题及其严重程度（仅仅是警告的，还是不可挽救的）。

记录协议

记录协议（Record Protocol）运载来自上一层（握手协议、改变加密规约协议、告警协议或应用层）的报文。这个报文经过分片和压缩（可选）处理，再用协商后的散列算法来计算得到 MAC，并将其添加到压缩后的报文上。这个压缩后的分片与 MAC 一起经过协商后的加密算法的加密。最后在加密的报文上添加 SSL 首部。图 30.21 所示为发送方的处理过程。接收方的处理过程就是它的逆过程。

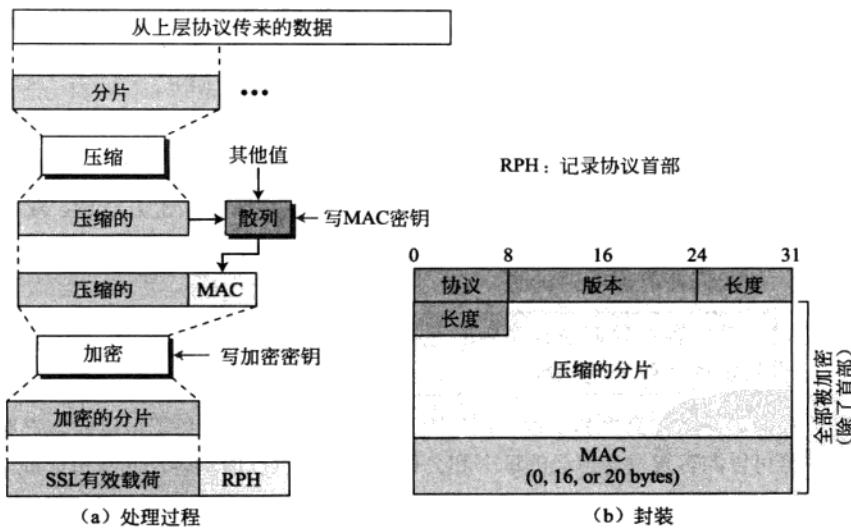


图 30.21 由记录协议完成的处理过程

30.3 应用层的安全

这一节要讨论为电子邮件提供安全服务的两个协议：相当好的保密（PGP）和安全/多用途因特网邮件扩充（S/MIME）。

30.3.1 电子邮件的安全

发送电子邮件是个即时的行为。这种行为在本质上与我们前两节所讨论的不同。使用 IPSec 或 SSL 时，我们假设双方在相互之间建立起一个会话并双向地交换数据。而在电子邮件中没有会话存在。Alice 和 Bob 不会为此而建立任何会话。Alice 向 Bob 发送了一个报文，而在此后的某个时间，Bob 阅读了该报文，他有可能会回复，也有可能不会回复。我们所讨论的是单向报文的安全问题，因为 Alice 向 Bob 发送邮件与 Bob 向 Alice 发送邮件完全没有关系。

加密算法

如果说电子邮件是即时的行为，那么发送方与接收方如何才能就用于电子邮件安全的加密算法达成一致意见呢？如果双方之间不存在会话，也就没有握手的过程来协商在加密/解密和散列过程中使用的算法，那么接收方如何知道发送方为实现各种目的而选择了哪种算法呢？

要解决这个问题，协议为每种操作定义了一组算法，以便用户在他/她的系统中使用。Alice 要把她所使用的算法的名称（或标识）包含在电子邮件中。例如，Alice 可以选择用 DES 进行加密/解密，并选择用 MD5 进行散列。当 Alice 向 Bob 发送报文时，她要在自己的报文中包含与 DES 和 MD5 相对应的标识。Bob 在接收到该报文后，首先要提取这些标识，然后就能知道在解密和散列时应当分别使用哪种算法了。

为了电子邮件的安全，报文的发送方需要在报文中包含所用算法的名称或标识。

加密的密钥

加密算法在使用加密密钥时也存在同样的问题。如果没有协商过程，双方如何在彼此之间建立密钥？目前的电子邮件安全协议要求使用对称密钥算法进行加密/解密，并且这个一次性的密钥要跟随报文一起发送。Alice 可以生成一个密钥并把它与报文一起发送给 Bob。为了保护密钥不被 Eve 截获，这个密钥需要用 Bob 的公钥进行加密。换言之，这个密钥本身也要被加密。

为了电子邮件的安全，加密/解密使用对称密钥算法实现，但是用于解密报文的密钥也需要用接收方的公钥加密，并与报文一起发送。

证书

在我们讨论任何具体的电子邮件安全协议之前，还有一个问题需要考虑。很显然，要实现电子邮件的安全就必须使用某些公钥算法。例如，我们需要对密钥加密或者对报文签名。为了对密钥进行加密，Alice 就需要 Bob 的公钥，同样为了验证被签名的报文，Bob 也需要 Alice 的公钥。因此，为了发送一小段鉴别的且保密的报文，就需要用到两个公钥。但是 Alice 如何才能确认 Bob 的公钥，Bob 又如何才能确认 Alice 的公钥呢？不同的电子邮件安全协议有不同的方法来验证密钥。

30.3.2 相当好的保密（PGP）

这一节我们要讨论的第一个协议称为相当好的保密（Pretty Good Privacy，PGP）。PGP

是由 Phil Zimmermann 发明的，它能够为电子邮件提供保密性、完整性和鉴别。PGP 可用于生成安全的电子邮件的报文。

几种情况

让我们先来讨论 PGP 的总体思想，从最简单的情况入手，并逐步深入地讨论更为复杂的情况。在这里，我们使用了术语“数据”来表示处理前的报文。

明文 最简单的情况就是用明文发送电子邮件报文，如图 30.22 所示。在该情况中不存在报文的完整性和保密性。



图 30.22 明文报文

报文完整性 接下来我们要稍加改进，可以让 Alice 对报文进行签名。Alice 产生一个报文摘要，并用自己的私钥对它签名。图 30.23 描绘了这种情况。

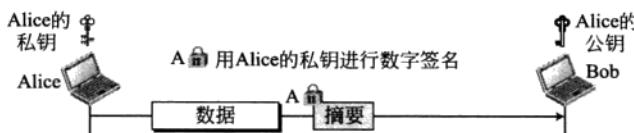


图 30.23 鉴别的报文

当 Bob 收到这个报文时，他要用 Alice 的公钥来验证报文。在这一情况中需要用到两个密钥。Alice 需要知道自己的私钥，而 Bob 需要知道 Alice 的公钥。

压缩 下一步的改进就是要对这个报文进行压缩以使分组更加紧凑。这种改进不会对安全带来什么好处，但是它能减少通信量。图 30.24 描绘了这一情况。

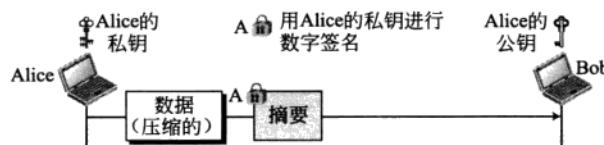


图 30.24 压缩的报文

使用一次性会话密钥的保密 图 30.25 描绘了这种情况。正如我们在前面所讨论的，电子邮件系统中的保密性可以通过使用一次性会话密钥的传统加密方法来实现。Alice 生成一个会话密钥，并用这个会话密钥加密报文和摘要，然后把这个密钥连同报文一起发送出去。不过，为了保护这个会话密钥，Alice 要用 Bob 的公钥对其进行加密。

当 Bob 收到这个分组时，他首先要摘取密钥，并用自己的私钥对这个密钥进行解密。然后 Bob 再用这个会话密钥对报文的其余部分解密。在对报文的其余部分成功解压缩后，Bob 还要生成这个报文的摘要，并检查它与 Alice 发送来的摘要是否一致。如果一致，那么这个报文通过鉴别。

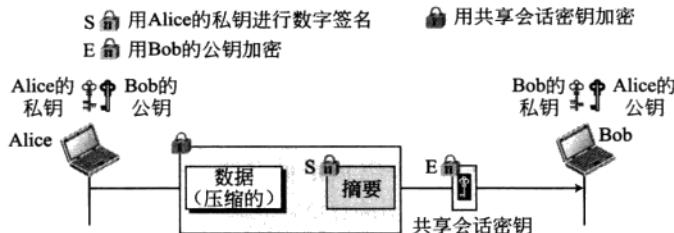


图 30.25 保密的报文

编码转换 PGP 提供的另外一个服务就是编码转换。大多数电子邮件系统只允许报文由 ASCII 字符组成。为了转换 ASCII 字符集中不存在的其他字符, PGP 使用 Radix-64 转换 (参见第 23 章)。

分段

PGP 允许报文在转换为 Radix-64 之后进行分段, 使每个传送单元的长度都是底层电子邮件协议能够支持统一的长度。

30.3.3 密钥环

在前面提到的所有情况下, 我们假设了 Alice 只需要向 Bob 一个人发送报文。实际情况当然不会是这样的, Alice 可能需要向很多人发送报文, 她需要的是一个密钥环 (key rings)。Alice 需要一个公钥环, 上面的公钥属于 Alice 要与之通信 (发送或接收报文) 的每一个人。另外, PGP 的设计者还指明了由私钥/公钥对组成的密钥环。一个原因是 Alice 可能希望时常更换自己的密钥对。另一个原因是 Alice 可能需要与几组人员 (朋友、同事, 等等) 联系。Alice 希望跟每一组人员使用不同的密钥对。因此, 每个用户都需要两个密钥环: 一个是私钥/公钥环, 另一个是别人的公钥环。图 30.26 描绘了三个人之间的通信, 每个人都有一个私钥/公钥环, 同时还有一个公钥环, 其中的公钥属于这个团体中的其他人。



图 30.26 PGP 中的密钥环

例如, Alice 有几对属于自己的私钥/公钥, 同时还有一些属于其他人的公钥。请注意, 每个人都可能有多个公钥。有以下两种情况可能发生:

1. Alice 需要向团体中的其他人发送报文。
 - a. 她用自己的私钥来签名摘要。
 - b. 她用接收方的公钥来加密新生成的会话密钥。
 - c. 她用生成的会话密钥来加密报文和签名的摘要。
2. Alice 收到来自团体中其他人的报文。
 - a. 她用自己的私钥来解密会话密钥。

- b. 她用会话密钥来解密报文和摘要。
- c. 她用对方的公钥来验证这个摘要。

PGP 算法

PGP 定义了一组不对称密钥算法和对称密钥算法、几个加密散列函数以及几种压缩方法。我们把对这些算法的细节留给专门介绍 PGP 的书籍。当 Alice 向 Bob 发送电子邮件时，对于每一种用途，她都要定义自己所使用的算法。

30.3.4 PGP 的证书

与我们在前面讨论的其他协议一样，PGP 也利用证书来鉴别公钥。但是其过程却完全不同，说明如下。

PGP 的证书

PGP 不需要 CA，密钥环中的任何一个人都可以为环中的其他任何人签署证书。Bob 可以为 Ted、John、Anne 等签署证书。在 PGP 中，信任是不分等级的，也没有树结构。缺少等级的结构可能会导致这样一个事实，Ted 可能有一个由 Bob 签发的证书，同时还有一条由丽兹签发的证书。如果 Alice 想追踪 Ted 的证书的线索，那么就会发现有两个路径：一条始发于 Bob，另一条始发于丽兹。有趣的是，Alice 可能完全信任 Bob，而部分地信任丽兹。从完全信任或部分信任的证书签发者到一个证书之间的信任线索上可能存在多条路径。在 PGP 中，证书的签发者通常称为介绍人（introducer）。

在 PGP 中，从完全信任或部分信任的证书签发者到任何证书都可能有多条路径。

信任与合法性

PGP 的整个操作的基础就是介绍人信任、证书信任以及公钥合法性。

介绍人的信任度 因为缺少集中式的管理机构，所以如果 PGP 密钥环上的每个用户都必须完全信任其他所有人，很显然这个密钥环就不能很大（即使在现实生活中我们也不能完全信任我们认识的每一个人）。为了解决这个问题，PGP 允许存在不同的信任度。信任度的级数完全取决于不同的实现，但是为了使问题简化，在这里我们为介绍人分配三级信任度：none（不）、partial（部分）和 full（完全）。介绍人的信任度指明了介绍人对环上的其他人来说所具有的信任程度。例如，Alice 可能完全信任 Bob，部分信任 Anne，而根本不信任 John。PGP 中不存在任何机制用于判断如何决定介绍人的信任程度，这件事完全由用户自己判断。

证书的信任度 当 Alice 从介绍人那儿收到证书时，她把这个证书保存在某人（被认证的实体）的名下。Alice 要为这个证书分配信任度。证书的信任度通常与发出这个证书的介绍人的信任度一致。假设 Alice 完全信任 Bob，部分信任 Anne 和 Janette，并且不信任 John，那么就会发生以下的情况：

1. Bob 发出了两个证书，一个为 Linda（连同公钥 K1）；另一个为 Lesley（连同公钥 K2）。Alice 把给 Linda 的公钥和证书保存在 Linda 名下，并为这个证书分配了 full 的信任度。同样，Alice 把给 Lesley 的证书和公钥保存在 Lesley 的名下，并为这个证书也分配了 full 的信任度。

2. Anne 为 John 发出了一个证书（连同公钥 K3）。Alice 把这个证书和公钥保存在 John 的名下，不过只给这个证书分配了 *partial* 的信任度。

3. Janette 发出了两个证书，一个为 John（连同公钥 K3）；另一个为雷（连同公钥 K4）。Alice 把 John 的公钥和证书保存在 John 的名下，把雷的公钥和证书保存在雷的名下，每个证书的信任度都是 *partial*。请注意，John 现在有两个证书，一个来自 Anne，一个来自 Janette，这两个证书的信任度都是 *partial*。

4. John 为丽兹发出了一个证书。Alice 可以丢弃这个证书，或者用信任度 *none* 来保存这个证书。

密钥的合法性 介绍人和证书的信任度的作用就是为了判断公钥的合法性。Alice 需要知道 Bob、John、丽兹、Anne 等这些人的公钥的合法程度。PGP 定义了一个非常明确的过程用于判断密钥的合法性。一个用户的密钥的合法程度就是该用户的加权信任度。例如，假设我们为证书的信任度分配如下的权重：

1. 对不信任的证书，权重为 0；
2. 对部分信任的证书，权重为 1/2；
3. 对完全信任的证书，权重为 1。

那么为了完全地信任某个实体，Alice 需要具有该实体的一个完全信任的证书或者两个部分信任的证书。例如，Alice 可以使用上一个事例中的 John 的公钥，因为 Anne 和 Janette 都为 John 发出了证书，每个证书的信任度都是 1/2。请注意，一个用户的公钥的合法性与该用户本人的信任度没有任何关系。虽然 Alice 可以用 John 的公钥向 John 发送报文，但 Alice 不能接受由 John 发出的任何证书，因为对于 Alice 来说，John 的信任度还是 *none*。

PGP 中的信任模型

正如 Zimmermann 提议的，我们能够以密钥环上的某个用户为中心，为该用户创建一个信任模型。这个模型看上去就像图 30.27 所描绘的一样。图中给出了在某一时刻 Alice 的信任模型。

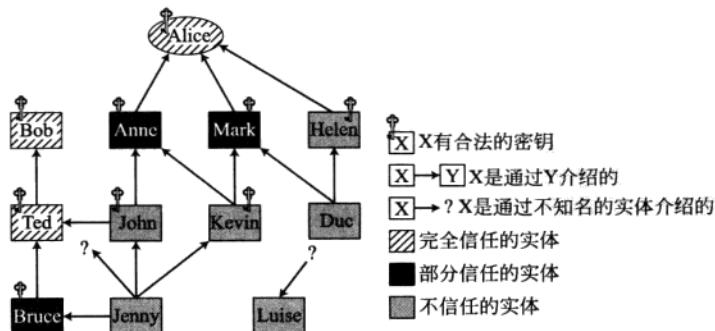


图 30.27 信任模型

让我们来详细地解说这幅图。图 30.27 显示出在 Alice 的密钥环上有三个实体是完全信任的（Alice 自己、Bob 和 Ted）。图中还显示出有三个实体是部分信任的（Anne、Mark 和 Bruce），有六个实体是不可信任的。一共有九个实体具有合法的密钥，Alice 可以向他们中

的任何一位发送加密的报文或验证由他们发来的签名（Alice 的密钥不会在这个模型中使用）。图中还有三个实体不具有任何合法的与 Alice 通信的密钥。

Bob、Anne 和 Mark 通过用电子邮件发送密钥并通过电话验证指纹的方法使得他们的密钥成为合法。另一方面，Helen 发送了一份来自 CA 的证书，因为她本人是不被 Alice 信任的，而用电话来验证的方法不可行。虽然 Ted 本人也是完全被信任的，他还是给了 Alice 一份由 Bob 签发的证书。John 向 Alice 发送了两份证书，一份由 Ted 签发；另一份由 Anne 签发。凯文也发送了两份证书给 Alice，一份由 Anne 签发，一份由 Mark 签发。这两份证书各给了凯文一半的信任度，因此凯文的密钥也是合法的。Duc 发送了两份证书给 Alice，一份由 Mark 签发；另一份由 Helen 签发。因为 Mark 只有一半的信任度，而 Helen 是不可信任的，所以 Duc 没有合法的密钥。Jenny 发送了四份证书，一份由只有一半信任度的实体签名，两份由不可信任的实体签名，还有一份由未知的实体签名。Jenny 还是没有获得足够的分数来使自己的密钥合法化。Luise 发送了一份由未知的实体签名的证书。请注意，Alice 在表中仍然保存了 Luise 的名字，这是为了在将来 Luise 的证书到达后使用。

信任关系网

PGP 最终能够为一群人编织出一张信任关系网（web of trust）。如果每个实体都向其他实体介绍更多的实体，那么每个实体的密钥环都会变得越来越大，而环上的实体也可以相互之间发送安全的电子邮件。

密钥的废弃

有时一个实体可能需要从密钥环上废弃他或她的公钥。如果密钥的主人认为自己的密钥受到了威胁（例如被偷了）或仅仅是因为用得时间太长而不安全了，那么这个密钥就需要被废弃。要废弃一个密钥，密钥的主人可以发送一个由本人签名的废弃证书。这个废弃证书必须用旧的密钥签名，并且必须要发送给环上所有使用了这个密钥的人。

PGP 分组

PGP 中的报文由一个或多个分组组成。在 PGP 不断改进的过程中，分组类型的格式和数量都有所改变。我们在这里不再讨论这些分组的格式。

PGP 的应用

PGP 已经被广泛地应用于个人电子邮件中，并且这种情况很可能会持续下去。

30.3.5 S/MIME

另外还有一种为电子邮件而设计的安全服务，就是安全/多用途因特网邮件扩充（Secure/Multipurpose Internet Mail Extension, S/MIME）。这个协议是我们在第 23 章讨论的多用途因特网邮件扩充（MIME）协议的增强版。

加密的报文语法（CMS）

为了说明如何能够在 MIME 内容类型中增加像保密性和完整性这样的安全服务，S/MIME 定义了加密的报文语法（Cryptographic Message Syntax, CMS）。这个语法逐个地定义了用于每一种内容类型的明确编码机制。下面所描述的是报文的类型以及从这些报文中产生出来的不同的子类型。对于更详细的介绍，请读者参考 RFC 3369 和 RFC 3370。

数据内容类型 这是一个任意的字符串，产生的对象称为 data（数据）。

签名的数据内容类型 此类型仅提供数据的完整性。它包含了任意类型以及零到多个签名值。这种编码的结果是称为 signedData（签名的数据）的对象。图 30.28 描绘了生成此类对象的过程。以下是这个过程的步骤：

1. 对于每个签名者，先用该签名者所选定的散列算法，从内容中产生报文摘要。
2. 每个报文摘要都要用相应签名者的私钥进行签名。
3. 然后把内容、签名值、证书以及算法收集起来生成 signedData 对象。

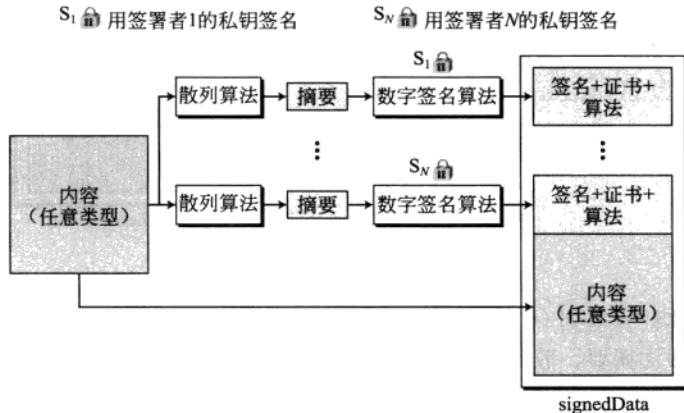


图 30.28 签名的数据内容类型

包装的数据内容类型 此类型用于提供报文的保密性。它包含了任意类型以及零到多个加密的密钥和证书。这种编码的结果是称为 envelopedData（包装的数据）的对象。图 30.29 描绘了产生此类型对象的过程。

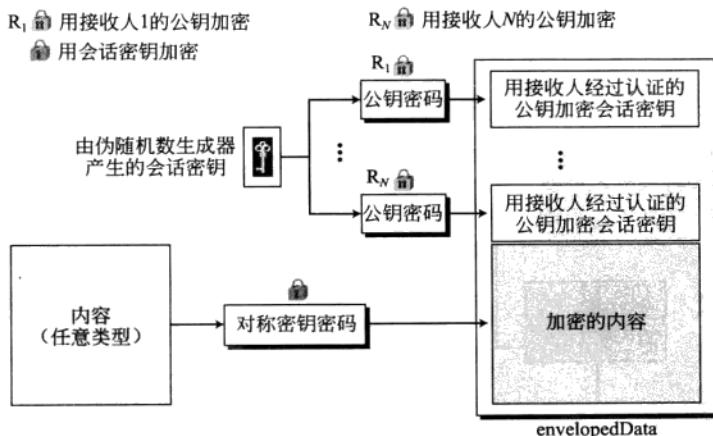


图 30.29 包装的数据内容类型

1. 生成伪随机会话密钥，以便用于对称密钥算法。

2. 对于每个接收人，用该接收人的公钥来加密这个会话密钥的副本。
3. 内容用定义的算法和生成的会话密钥加密。
4. 把加密的内容、加密的会话密钥、使用的算法和证书用 Radix-64 进行编码。

摘要的数据内容类型 此类型用于提供报文的完整性。结果被用做包装的数据内容类型中的内容。这种编码的结果是称为 `digestedData`（摘要的数据）的对象。图 30.30 描绘了产生此类型对象的过程。

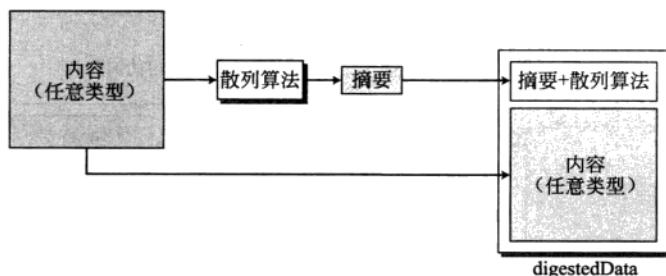


图 30.30 摘要的数据内容类型

1. 根据内容计算得到一个报文摘要。
2. 把这个报文摘要、散列算法以及内容合在一起生成 `digestedData` 对象。

加密的数据内容类型 此类型用于生成对任意内容类型的加密的版本。虽然它看起来很像包装的数据内容类型，但是加密的数据内容类型没有接收人。它用于保存加密的数据，而不是用于传输。它的处理过程很简单：用户应用任意密钥（通常是根据口令推导出来的）和任意算法对内容进行加密。这个加密的内容在保存时并不包括密钥和算法。此种对象称为 `encryptedData`（加密的数据）。

鉴别的数据内容类型 此类型用于提供数据的鉴别。这种对象称为 `authenticatedData`（鉴别的数据）。图 30.31 描绘了它的产生过程。

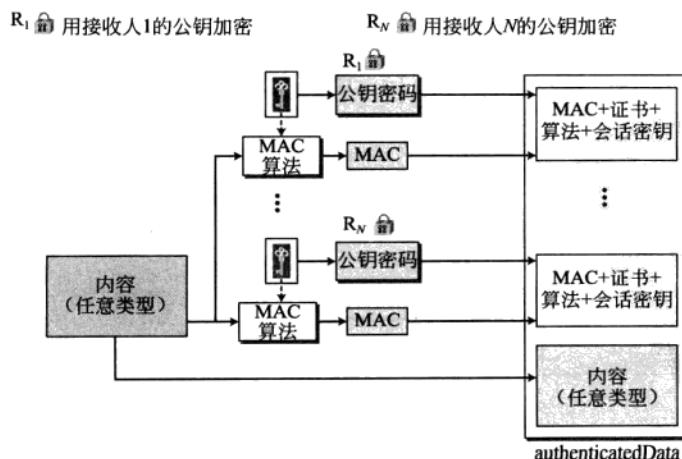


图 30.31 鉴别的数据内容类型

1. 使用伪随机数生成器，为每个接收人产生 MAC 密钥。
2. 用接收人的公钥加密这个 MAC 密钥。
3. 从内容中产生 MAC。
4. 把这个内容、MAC、算法以及其他信息合在一起形成 authenticatedData 对象。

密钥管理

S/MIME 中的密钥管理综合了 X.509 和 PGP 所使用的密钥管理机制。S/MIME 使用由认证中心签发的公钥证书。但是它又像 PGP 定义的那样，由用户负责维护信任关系网来验证签名。

密码的算法

S/MIME 定义了几个加密算法。我们把这些算法的细节留给专门讨论因特网安全问题的书籍。

例 30.1

以下给出的是一个包装的数据的例子，它用三重 DES 对一个小报文进行了加密。

```
Content-Type: application/pkcs7-mime; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
name="report.txt";
cb32ut67f4bhijHU21oi87eryb0287hmnk1sgFD0Y8bc659GhIGfH6543mhjkdsaH23YjBnm
Nybm1kzjhgfdyhGe23Kjk34XiuD678Es16se09jy76jHuytTMDcbnmlkjgfFdiuyu678543m
0n3hG34un12P2454Hoi87e2ryb0H2MjN6KuyrlsgFD0Y897fk923jljk1301XiuD6gh78EsU
yT23y
```

30.3.6 S/MIME 的应用

预计 S/MIME 将会成为业界为商业电子邮件提供安全的选择。

30.4 防火墙

前面我们讨论的所有安全措施都不能防止 Eve 向一个系统发送有害的报文。为了控制对一个系统的接入，我们需要使用防火墙。防火墙（firewall）是一种设备（通常是路由器或计算机），安装在一个组织内部网络和外部因特网之间。设计防火墙的目的是为了转发某些分组，而过滤（不转发）其他的分组。图 30.32 描绘了一个防火墙。

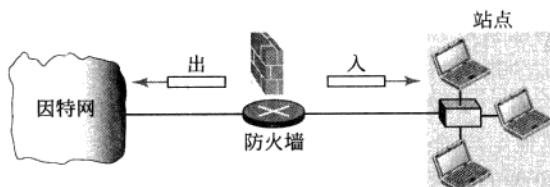


图 30.32 防火墙

例如，某个防火墙可以过滤掉所有发往特定主机或特定服务器（如 HTTP）的分组。防火墙可用来拒绝接入到一个组织内部的某个特定主机或特定服务。通常，防火墙可划分为分组过滤防火墙和基于代理的防火墙两大类。

30.4.1 分组过滤防火墙

防火墙可被用做分组过滤器。它能够基于分组的网络层或运输层首部中的信息来转发或阻拦该分组，这些信息包括源 IP 地址、目的 IP 地址、源端口、目的端口、协议类型（TCP 或 UDP）。分组过滤防火墙（packet-filter firewall）是一种路由器，它使用过滤表来决定哪些分组必须丢弃（不转发）。图 30.33 给出了这种防火墙的过滤表的例子。

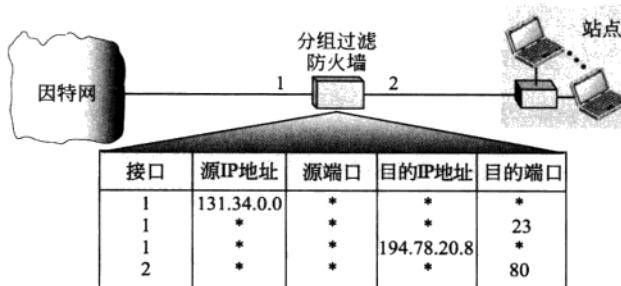


图 30.33 分组过滤防火墙

根据这个表，以下分组都要被过滤掉：

- 从网络 131.34.0.0 来的入分组都被阻拦（安全性预防）。应当注意，星号 “*” 表示“任何”。
- 发给任何内部 TELNET 服务器（端口 23）的入分组都被阻拦。
- 发送给内部主机 194.78.20.8 的入分组都被阻拦。这个组织希望这个主机仅作为内部使用。
- 发送给 HTTP 服务器（端口 80）的出分组都被阻拦。这个组织不愿意雇员们浏览因特网。

分组过滤防火墙在网络层或运输层起过滤作用。

30.4.2 代理防火墙

分组过滤防火墙的依据是分组的网络层和运输层首部中的信息（IP 和 TCP/UDP）。然而有时我们需要基于这个报文本身提供的信息（在应用层）来过滤报文。例如，假定某组织针对它的万维网网页希望实现以下策略：只有以前和这个公司建立了业务关系的那些因特网用户才能接入，其他用户的接入必须被阻止。在这种情况下不能使用分组过滤防火墙，因为路由器不能区分到达 TCP 端口 80（HTTP）的各种分组。此类检测必须在应用层（使用 URL）实现。

解决问题的一种方法是安装代理计算机（有时也称为应用网关（application gateway）），它位于顾客计算机和公司计算机之间。当用户的客户进程发送报文时，应用网关就运行一个服务器进程来接收这个请求。该服务器进程在应用层打开分组，并检查这个请求是否合法。如果是合法的，服务器进程就充当客户进程，把报文发送给公司真正的服务器。如果不合法，就丢弃这个报文，并发送差错报文给外部用户。使用这种方法，可以根据应用层的内容来过滤外部用户的请求。图 30.34 描绘了 HTTP 的应用网关的实现。

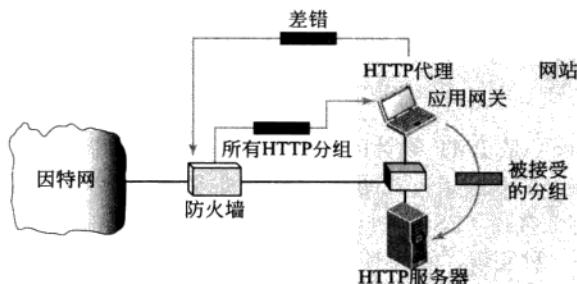


图 30.34 代理防火墙

代理防火墙在应用层进行过滤。

30.5 深入阅读

要更细致地了解本章所讨论的内容，我们推荐以下书籍。用方括号括起来的书目可以在本书末尾的参考书目清单中找到。特别地，我们推荐[For 08]、[Sta 06]、[Bis 05]、[Mao 04]、[Sti 06]、[Res 01]、[Tho 00]、[Dor & Har 03]和[Gar 01]。

30.6 重要术语

告警协议	因特网安全关联和密钥管理协议 (ISAKMP)
应用网关	IP 安全 (IPSec)
鉴别首部 (AH) 协议	密钥材料
改变加密规约协议	密钥环
加密方法族	主密
连接	Oakley
密码的报文语法 (CMS)	分组过滤防火墙
封装安全有效载荷 (ESP)	前主密
防火墙	相当好的保密 (PGP)
握手协议	记录协议
因特网密钥交换 (IKE)	安全套接层 (SSL) 协议

安全/多用途因特网邮件扩充 (S/MIME)	SKEME
安全关联 (SA)	运输层安全 (TLS) 协议
安全关联数据库 (SAD)	运输方式
安全策略 (SP)	隧道方式
安全策略数据库 (SPD)	虚拟专用网络 (VPN)
会话	信任关系网

30.7 本章小结

- IP 安全 (IPSec) 是 IETF 设计的一组协议，用来为网络层的分组提供安全。IPSec 的操作有运输方式和隧道方式两种。IPSec 定义了两个协议：鉴别首部 (AH) 协议和封装安全有效载荷 (ESP) 协议，它们为 IP 级的分组提供鉴别和加密。虚拟专用网是 IPSec 的一种实现，它为具有多个站点的组织提供了保密性。
- 运输层安全协议为使用了可靠运输层协议（如 TCP）的应用程序提供端到端的安全服务。目前主要有两个协议在运输层提供安全服务：安全套接层 (SSL) 和运输层安全 (TLS)。SSL (或 TLS) 为接收自应用层的数据提供诸如分片、压缩、报文完整性、保密性以及组帧的服务。
- 由 Phil Zimmermann 研发的相当好的保密 (PGP) 为电子邮件提供保密性、完整性和鉴别。另一种为电子邮件而设计的安全服务是安全/多用途因特网邮件扩充 (S/MIME)。这个协议是多用途因特网邮件扩充 (MIME) 协议的增强版。
- 防火墙是安装在组织内部网络和外部因特网之间的一个设备（通常是路由器或计算机）。它的设计是为了转发某些分组，而过滤掉其他的分组。防火墙通常可分为分组过滤防火墙和代理防火墙两大类。分组过滤防火墙依据分组的网络层和运输层首部中的信息来阻拦或转发该分组。代理防火墙基于应用层的信息来阻拦或转发分组。

30.8 实践安排

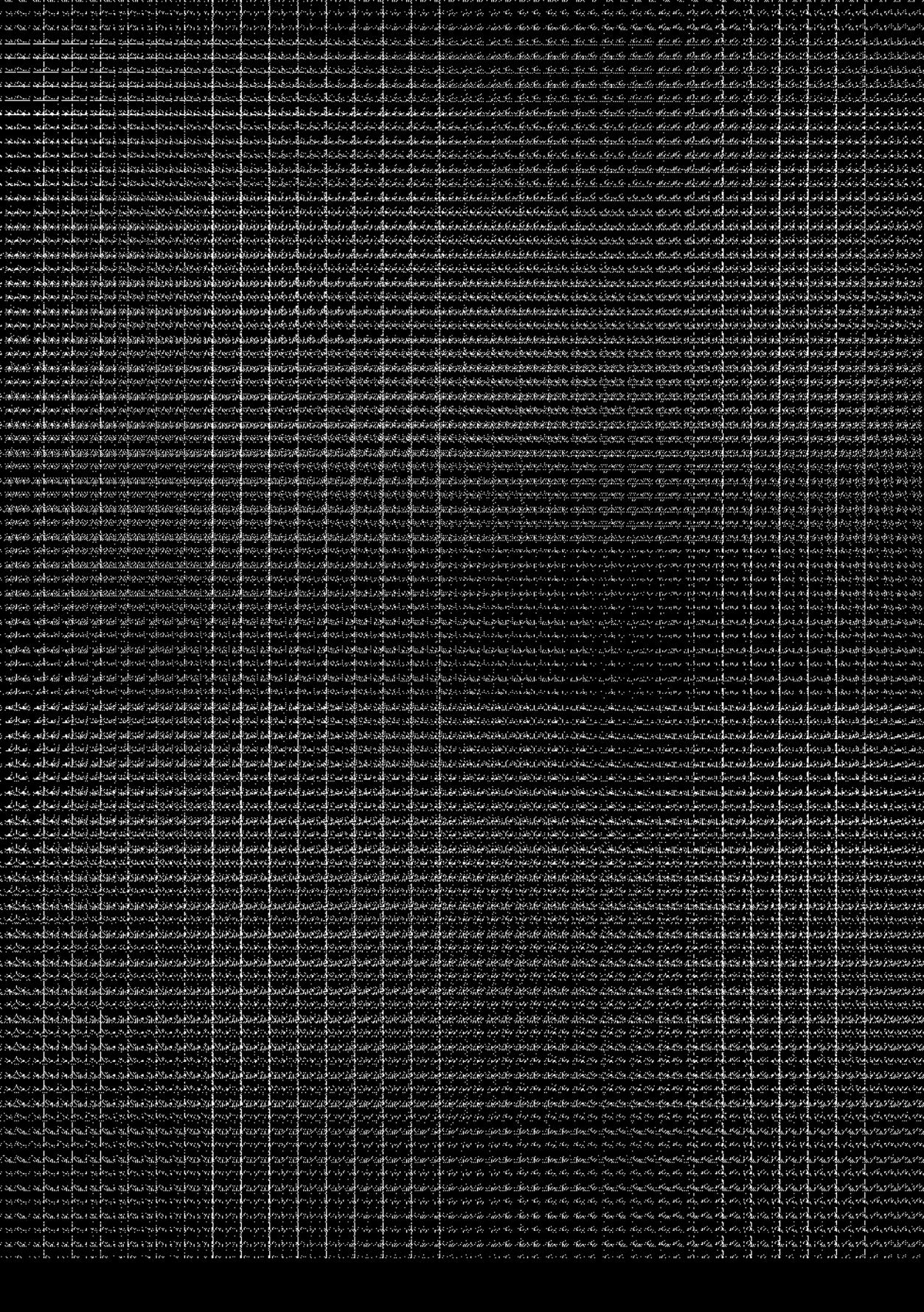
30.8.1 习题

1. 主机 A 和主机 B 以运输方式使用 IPSec。我们可以说这两台主机必须在它们之间建立一条虚拟面向连接的服务吗？为什么？
2. 当我们说到 IPSec 中的鉴别时，我们指的是报文鉴别还是实体鉴别？为什么？
3. 当我们说到 SSL 中的鉴别时，我们指的是报文鉴别还是实体鉴别？为什么？
4. 当我们说到 PGP (或 S/MIME) 中的鉴别时，我们指的是报文鉴别还是实体鉴别？为什么？
5. 既然 PGP 或 S/MIME 中的加密算法无法经过协商确定，那么电子邮件的接收人如何确定发送人使用了什么算法？

6. 我们能够为 UDP 使用 SSL 吗？为什么？
7. 为什么不需要 SSL 的安全关联？
8. 试对比 PGP 和 S/MIME。它们各有什么优缺点？
9. 假如 Alice 和 Bob 要连续不断地互相发送报文，我们是否能建立一条一次性的安全关联，并将其用于每个分组的交换？为什么？
10. SSL 中的握手过程是在 TCP 的三次握手过程之前还是之后发生的？它们能合并到一起吗？为什么？

30.8.2 研究活动

11. 我们只讨论了为应用层的 SMTP 所提供的安全服务。是否还有为其他的应用层协议提供的安全服务？
12. 利用文献和因特网更多地了解 IKE。
13. 利用文献和因特网更多地了解 SSL 中的握手协议。
14. 利用文献和因特网更多地了解 TLS。



第七部分

附录

附录 A Unicode

附录 B 进位制计数系统

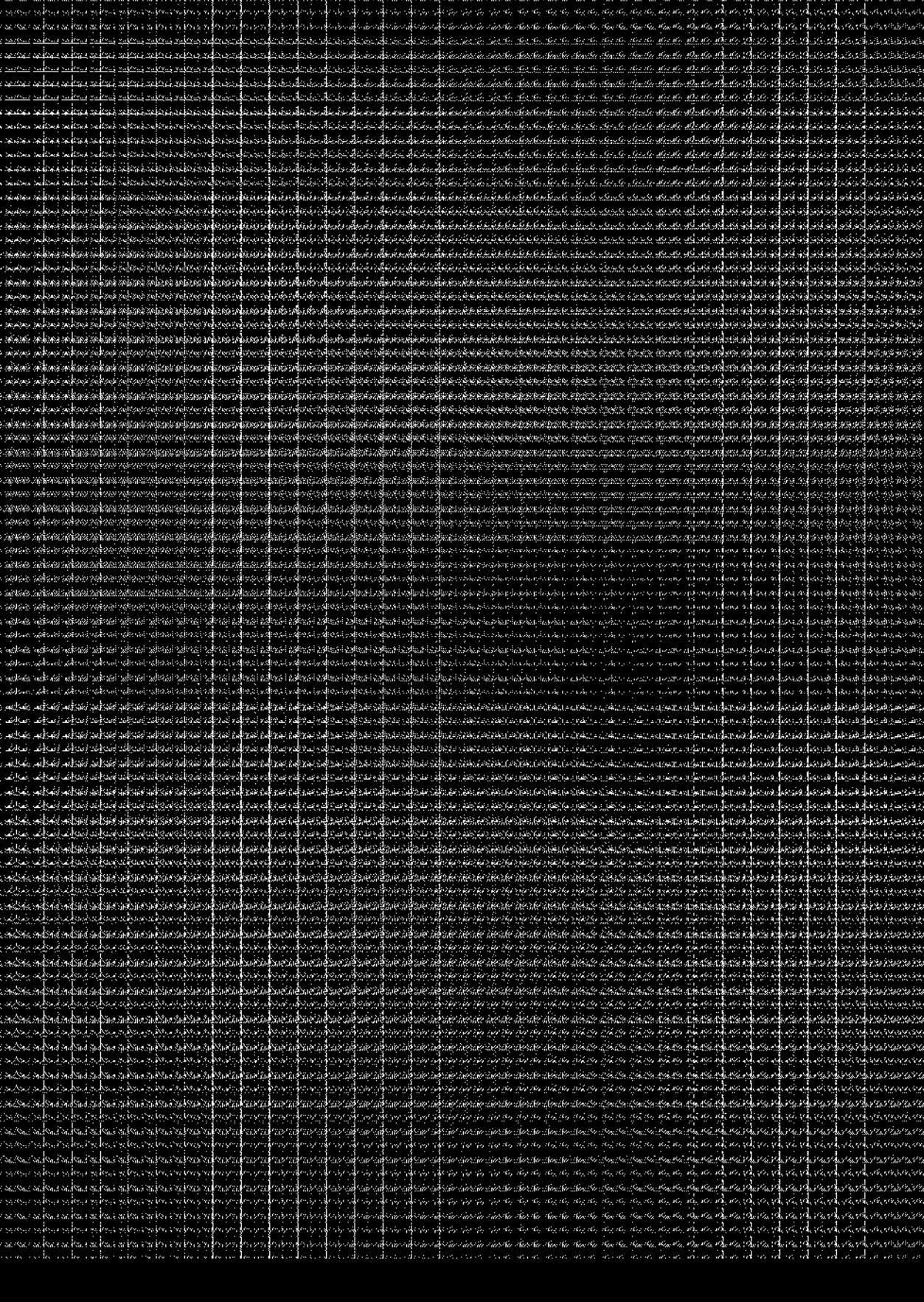
附录 C 差错检测码

附录 D 检验和

附录 E HTML、XHTML、XML 和 XSL

附录 F Java 中的客户-服务器编程

附录 G 其他信息



附录 A Unicode

计算机处理的是数字。它们在存储字符时要给每个字符分配一个数值。早期的编码系统称为 ASCII（美国信息交换标准码），一共有 128（0 到 127）个值，每个值用一个 7 位数保存。ASCII 可以满足小写字母、大写字母、数字、标点符号和一些控制字符的处理。人们曾尝试将 ASCII 字符扩充到 8 位。这种新的被称为扩充 ASCII 的编码一直没有成为国际性的标准。

为了克服 ASCII 和扩充 ASCII 先天上的不足，Unicode Consortium（多语言软件生产商群体）创建了一种能够提供广泛字符集的通用编码系统，称为 **Unicode**。

Unicode 最初设计为 2 字节的字符集。但是版本 3 的 Unicode 用的是 4 字节编码，并且与 ASCII 和扩充 ASCII 完全兼容。现在被称为 Basic Latin（基本拉丁文）的 ASCII 字符集就是前 25 位全部置为零的 Unicode 码。而现在称为 Latin-1（拉丁文 1）的扩充 ASCII 字符集就是前 24 位全部置零的 Unicode 码。图 A.1 描绘了这几个不同的系统是如何兼容的。

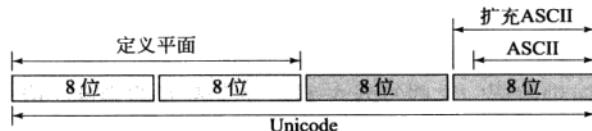


图 A.1 Unicode 的兼容性

Unicode 中的每个字符或符号由一个 32 位数来定义，因此这种编码可以定义高达 2^{32} （4 294 967 296）个字符或符号。它的记法使用了十六进制数字，格式如下：

U-XXXXXX

每个 X 都是一个十六进制数字。因此，它的数值从 U-00000000 开始一直到 U-FFFFFFFFFF 结束。

A.1 平面

Unicode 把有效的编码空间划分为许多平面。最高 16 位就是用来定义这些平面的，也就是说我们共有 65 536 个平面。每个平面最多可以定义 65 536 个字符或符号。图 A.2 所示为 Unicode 空间和平面的结构。

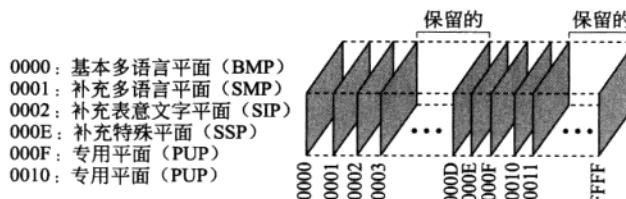


图 A.2 Unicode 平面

A.1.1 基本多语言平面 (BMP)

平面(0000)₁₆是基本多语言平面 (Basic Multilingual Plane, BMP)，它被设计来兼容早期的 16 位 Unicode。这个平面的最高 16 位全部为 0。通常，此类编码被表示为 U+XXXX，其中 XXXX 被心照不宣地定义为最低的 16 位。这个平面主要定义了在不同语言中使用的字符集，除此之外还有一些用于控制或其他特殊用途的字符。

A.1.2 其他平面

还有一些其他（非保留的）平面，我们简单讨论如下。

补充多语言平面 (SMP)

平面(0001)₁₆是补充多语言平面 (Supplementary Multilingual Plane, SMP)，它被设计来为没有包含在 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是补充表意文字平面 (Supplementary Ideographic Plane, SIP)，它被设计来为表意符号提供编码，这些符号主要表达了符号的意义（或含义），而不是声音（发音）。

补充特殊平面 (SSP)

平面(000E)₁₆是补充特殊平面 (Supplementary Special Plane, SSP)，用于特殊字符。

专用平面 (PUP)

平面(000F)₁₆和(0010)₁₆是专用平面，由专用系统使用。

A.2 ASCII

美国信息交换标准码 (American Standard Code for Information Interchange, ASCII) 是一种 7 位码，设计来为 128 个大多数是美国英语里使用的符号提供编码，今天 ASCII 码（或称为 Basic Latin）已成为 Unicode 中的一部分。它占了 Unicode 中的前 128 个码（从 00000000 到 0000007F）。表 A.1 包含了十六进制编码及其表现符号。表中的十六进制编码只定义了 Unicode 的两个最低位数字。要找出其实际的编码，还要在这个编码前面附加十六进制的 000000。

表 A.1 ASCII 码

十六进制	符号	十六进制	符号	十六进制	符号	十六进制	符号
00	NUL	20	SP	40	@	60	'
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	\$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(48	H	68	h
09	HT	29)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[7B	{
1C	FS	3C	<	5C	\	7C	
1D	GS	3D	=	5D]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?`	5F	_	7F	DEL

ASCII 的一些特点

ASCII 有一些很有趣的特点，我们简单介绍如下：

1. space 字符(20_{16})是一个可打印的字符。它打印出一个空格。

2. 大写字母从(41_{16})开始。小写字母从(61_{16})开始。比较时，大写字母在数值上比小写字母要小。也就是说，如果根据 ASCII 的值进行排序的话，大写字母会出现小写字母的前面。

3. 大写字母与小写字母在它们的 7 位编码中只有一位不同。例如，字符 A 是(1000001_2)，而字符 a 是(1100001_2)，不同之处在第 6 位，大写字母是 0，小写字母是 1。如果我们知道了其中一个编码，就很容易能找出另一个编码，可以通过加上或减去(20_{16})，或者我们也可以简单地将第 6 位翻转。

4. 小写字母并没有紧跟在大写字母后面，在这两者之间还有几个标点符号。

5. 数字(0~9)从(30_{16})开始。这就意味着如果你希望把数字字符的 ASCII 值转换为该字符面上的整数值，那就要用这个 ASCII 值减去($30_{16} = 48$)。

6. 从(00_{16})到($1F_{16}$)这 32 个最前端的字符以及最后一个字符($7F_{16}$)都是非打印字符。字符(00_{16})被用做定界符，以定义字符串的结束。字符($7F_{16}$)是删除字符，它被某些编程语言用来删除前一个字符。剩下的非打印字符称为控制字符，用于数据通信。表 A.2 给出了对这些字符的描述。

表 A.2 字符描述

符号	说明	符号	说明
SOH	首部开始	DC1	设备控制 1
STX	正文开始	DC2	设备控制 2
ETX	正文结束	DC3	设备控制 3
EOT	传输结束	DC4	设备控制 4
ENQ	查询	NAK	否认
ACK	确认	SYN	同步
BEL	响铃	ETB	传输块结束
BS	退格	CAN	取消
HT	水平制表符	EM	媒体结束
LF	换行	SUB	代替
VT	垂直制表符	ESC	转移字符
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	数据链路转义字符		

附录 B 进位制计数系统

进位制计数系统 (positional number system) 使用了一组符号。不过，每个符号所代表的值和它的字面值 (face value) 与进位值 (place value) 都有关系，所谓进位值是指与它在该数中所处的位置相关的值。换言之，我们有

$$\text{符号值} = \text{字面值} \times \text{进位值}$$

$$\text{数值} = \text{符号值之和}$$

在附录 B 中，我们只讨论整数，也就是不带小数部分的数。对于带有小数部分的实数的讨论与此类似。

B.1 不同的系统

首先我们要说明如何用四种不同的系统来表示整数：基 10、基 2、基 16 和基 256。

B.1.1 基 10：十进制

我们讨论的第一个进位制系统称为十进制系统 (decimal system)。十进制的英文名字 “decimal” 来自拉丁语词干 *deci* (意思是十)。十进制系统使用了 10 个符号 (0、1、2、3、4、5、6、7、8 和 9)，其字面值与符号本身一致。十进制系统中的进位值是 10 的乘方。图 B.1 描绘了整数 4782 中的进位值和符号值。

	10^3	10^2	10^1	10^0	进位值
符号	4	7	8	2	符号值
数值	4000	+ 700	+ 80	+ 2	数值
					4782

图 B.1 一个十进制数的例子

十进制系统使用 10 个符号，其中的进位值是 10 的乘方。

B.1.2 基 2：二进制

我们讨论的第二个进位制系统称为二进制系统 (binary system)。二进制的英文名字 “binary” 来自拉丁语词干 *bi* (意思是两个两个地)。二进制系统使用两个符号 (0 和 1)，其字面值与符号本身一致。二进制系统中的进位值是 2 的乘方。图 B.2 描绘了二进制数(1101)₂ 中的进位值和符号值。请注意，我们用下标 2 来表示这个数是二进制数。

2^3	2^2	2^1	2^0	进位值
1	1	0	1	符号
8	4	0	1	符号值

13 数值（用十进制表示）

图 B.2 一个二进制数的例子

二进制系统使用两个符号，其中的进位值是 2 的乘方。

B.1.3 基 16：十六进制

我们讨论的第三个进位制系统称为十六进制系统 (hexadecimal system)。十六进制的英文名字 “hexadecimal” 来自希腊语词干 hex (意思是 6) 和拉丁语词干 deci (意思是十)。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)，前十个符号的字面值与符号本身一致，从符号 A 到符号 F 的字面值分别为 10 到 15。十六进制系统中的进位值是 16 的乘方。图 B.3 描绘了十六进制数(A20E)₁₆ 中的进位值和符号值。请注意，我们用下标 16 来表示这个数是十六进制数。

16^3	16^2	16^1	16^0	进位值
A	2	0	E	符号
40960	+ 512	+ 0	+ 14	符号值（用十进制表示）

41486 数值（用十进制表示）

图 B.3 一个十六进制数的例子

十六进制系统使用 16 个符号，其中的进位值是 16 的乘方。

B.1.4 基 256：点分十进制记法

我们要讨论的第四个进位制系统是基 256 的，它称为点分十进制记法。这个系统用于表示 IPv4 地址。这个系统的进位值是 256 的乘方。不过，因为要使用 256 个符号几乎是不可能的事，所以这个系统的符号就是十进制数 0~255，其字面值与符号本身一致。为了把一个数与另一个数分隔开，这个系统采用了我们在第 5 章讨论过的点分法。图 B.4 描绘了地址 (14.18.111.252) 中的进位值和符号值。请注意，我们在 IPv4 地址中使用的符号从不超过四个。

256^3	256^2	256^1	256^0	进位值
14	18	111	252	符号
234881024	+ 1179648	+ 28416	+ 252	符号值

236089340 数值（用十进制表示）

图 B.4 一个点分十进制记法的例子

点分十进制记法使用十进制数字 (0~255) 作为符号，不过要在每两个符号之间插入一个点。

B.1.5 比较

表 B.1 给出了三种不同的系统如何表示十进制数 0~15。例如，十进制数 13 等于二进制数 1101，并且等于十六进制数 D。

表 B.1 三种系统的比较

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0	0	8	1000	8
1	1	1	9	1001	9
2	10	2	10	1010	A
3	11	3	11	1011	B
4	100	4	12	1100	C
5	101	5	13	1101	D
6	110	6	14	1110	E
7	111	7	15	1111	F

B.2 转换

我们需要掌握如何把一个系统的数转换为另一个系统中等价的数。

B.2.1 从任意数制到十进制的转换

图 B.2 至图 B.4 实际上已经告诉我们如何把一个任意数制中的数手工转换为十进制数。不过使用图 B.5 所示的算法会让转换更加简单。图中的算法利用了这样的事实，后一个进位值是前一个进位值乘以基（2, 16 或 256）。此算法是一个通用算法，它可以把任意给定基数的符号串转换为一个十进制数，其中针对不同的数制而不同的唯一之处是如何从符号串中提取下一个符号，并找出它的字面值。在基 2 的情况下事情非常简单，它的字面值可以通过把符号转变为数字来得到。在基 16 的情况下，我们需要考虑到符号 A 的字面值是 10，符号 B 的字面值是 11，依此类推。在基 256 的情况下，我们需要提取用点分隔的每个字符串，并把这个字符串转变为它的数值。我们把这些子算法的细节留为作业，因为它们通常都与使用的编程语言相关。

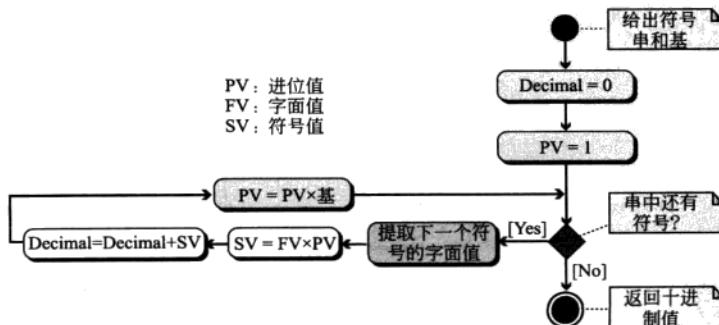


图 B.5 从任意数制到十进制转换的算法

下面给出了用手工方法对一些不是很大的数实现上述算法的几个例子。

例 B.1

给出等价于二进制数 $(11100111)_2$ 的十进制数。

解

我们的算法计算如下：

128		64		32		16		8		4		2		1	进位值
1		1		1		0		0		1		1		1	字面值
128		64		32		0		0		4		2		1	符号值
231		103		39		7		7		7		3		1	Decimal = 0

变量 Decimal 的初始值置为 0。随着循环结束，最后得到 Decimal 的值为 231。

例 B.2

给出等价于 IPv4 地址 12.14.67.24 的十进制数。

解

我们的算法计算如下：

16 772 216		65 536		256		1	进位值
12	•	14	•	67	•	24	字面值
201 266 592		917 504		17 152		24	符号值
202 255 272		988 680		71 176		24	Decimal = 0

变量 Decimal 的初始值置为 0。随着循环结束，最后得到 Decimal 的值为 202255272。

B.2.2 从十进制到任意数制的转换

如果我们不断地用十进制数除以基数，以得到每次的余数和商，就能够把十进制数值转换为任意数制数值。其中余数是下一个符号的字面值，而商则是用于下一次循环中的十进制数值。与它的逆算法一样，对应于不同的数制，我们也需要有不同的算法来把字面值转变为实际的符号，并把它插入到表示转换后的值的字符串中。我们把这些子算法的细节留为作业。图 B.6 所示为这个主算法。

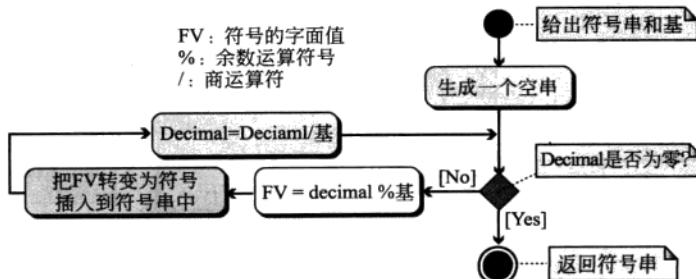


图 B.6 从十进制到任意数制的转换

下面给出了用手工方法来实现上述算法的几个例子。

例 B.3

把十进制数 25 转换为相应的二进制数。

解

我们不断地把这个十进制数除以 2 (二进制的基)，直至商为 0。在每次除运算中，我们把余数作为要插入到二进制串中的下一个符号。向下的箭头指向余数，向左的箭头指向商。当十进制数成为 0 后就停止，结果得到二进制串(11001)₂。

0	←	1	←	3	←	6	←	12	←	25	十进制
		↓		↓		↓		↓		↓	
		1		1		0		0		1	二进制

例 B.4

把十进制数 21432 转换为相应的十六进制数。

解

我们不断地把这个十进制数除以 16 (十六进制的基)，直至商为 0。在每次除运算中，我们把余数作为要插入到十六进制串中的下一个符号。结果得到十六进制串(53B8)₁₆。

0	←	5	←	83	←	1339	←	21432	十进制
		↓		↓		↓		↓	
		5		3		B		8	十六进制

例 B.5

把十进制数 73 234 122 转换为基 256 (IPv4 地址)。

解

我们不断地把这个十进制数除以 256 (基)，直至商为 0。在每次除运算中，我们把余数作为要插入到这个 IPv4 地址中的下一个符号。我们还要插入点分十进制记法所必需的点。结果得到的 IPv4 地址是 4.93.118.202。

0	←	4	←	1 117	←	286 070	←	73 234 122	十进制
		↓		↓		↓		↓	
		4	•	93	•	118	•	202	IPv4 地址

B.2.3 其他转换

从一个非十进制系统转换到另一个非十进制系统通常要简单一些。通过把每 4 位一组的二进制数字转换为一个十六进制数字，我们可以很容易地把二进制数转换为等价的十六进制数。我们也可以把一个十六进制数字转换为 4 位为一组的二进制数字。我们给出几个例子来说明这个过程。

例 B.6

把二进制数(1001111101)₂ 转换为相应的十六进制数。

解

我们从右端开始每 4 位划分为一组。然后，用每一组等价的十六进制数字去替换。请注意，最后一组还需要补充两个 0。

0010	0111	1101	二进制
↓	↓	↓	
2	7	D	十六进制

结果得到(27D)₁₆。

例 B.7

把十六进制数(3A2B)₁₆ 转换为相应的二进制数。

解

我们把每个十六进制数字转换为与它等价的 4 位二进制数字。

3	A	2	B	十六进制
↓	↓	↓	↓	
0011	1010	0010	1011	二进制

结果得到(0011 1010 0010 1011)₂。

例 B.8

把 IPv4 地址 112.23.78.201 转换为相应的二进制格式。

解

我们把每个符号用与其等价的 8 位二进制数字来代替。

112	•	23	•	78	•	201	IPv4 地址
↓		↓		↓		↓	
01110000	•	00010111	•	01001110	•	11001001	二进制

结果得到(01110000 00010111 01001110 11001001)₂。

附录 C 差错检测码

C.1 引言

网络必须能够以可接受的准确性来传送数据。对大多数应用来说，系统必须保证接收的数据与发送的数据完全一致。数据在传送过程中可能会受到损伤。有些应用要求有一种机制能够检测出差错，并最终能纠正差错。

C.1.1 差错的类型

差错可能只影响一个比特，也可能影响多个比特。术语**单比特差错**（single-bit error）的意思是在给定的数据单元中（例如一个字节、字符或分组），只有 1 个比特被改变了。术语**突发差错**（burst error）则表示在数据单元中有两个或更多的比特改变了。图 C.1 描绘了单比特差错或突发差错对数据单元的影响。

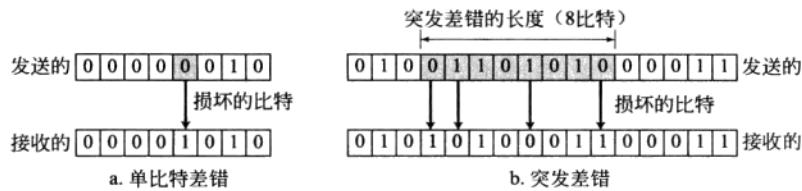


图 C.1 单比特差错和突发差错

突发差错比单比特差错更容易发生。噪声的持续时间通常大于 1 个比特的持续时间，也就是说当噪声影响数据时，通常会影响到一组比特。

C.1.2 冗余

差错检测或纠正的核心思想是冗余（redundancy）。为了能够检测或纠正差错，我们需要与数据一起发送一些额外的比特。这些冗余的比特由发送方添加，并由接收方消除。它们的存在使得接收方能够检测到受损的比特，并最终纠正差错。

C.1.3 检错与纠错的比较

差错纠正要比差错检测困难得多。在**差错检测**（error detection）时，我们只需要找找看是否有差错发生。答案很简单，是或者不是。我们甚至对差错的数量都不感兴趣。而在**差错纠正**（error correction）时，我们需要准确地知道受损比特的数量，更重要的是，还要知道这些比特

在报文中的位置。差错纠正主要有两种方法，前向纠错（forward error correction）就是接收方试图通过冗余的比特来猜测得到正确报文的过程。而通过重传（retransmission）进行差错纠正的这种技术则是由接收方检测差错的发生，并请求发送方重传报文。在这一节附录中，我们仅讨论差错检测，并且假设差错纠正都是通过重传的方法来实现的。

C.1.4 编码

冗余可以通过多种编码机制来实现。发送方用一个过程来添加一些冗余的比特，使得它们与实际的数据比特之间建立某种关系。接收方检查这两组比特之间的关系以检测或纠正差错。对任何编码机制来说，冗余比特的数量与数据比特的数量之比以及这个过程的强壮性都是至关重要的因素。图 C.2 描绘了编码的一般思想。

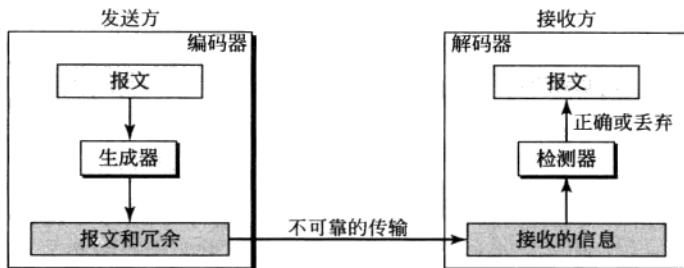


图 C.2 编码器和解码器的结构

我们可以把这些编码机制划分为两大类：块编码（block coding）和卷积编码（convolution coding）。在本附录中，我们重点讨论块编码，卷积编码要复杂得多，超出了本书的范围。

C.2 块编码

使用块编码时，我们把报文划分为称为数据字（datawords）的块，每个块有 k 位。我们给每个块添加 r 个冗余位，使得块的长度变成 $n = k + r$ 。结果得到的 n 位块就称为码字（codewords）。至于如何选择或计算这额外的 r 位，我们稍后再谈。就目前而言，更重要的是我们必须知道现在有一组数据字，它们的长度是 k ，还有一组码字，它们的长度是 n 。对于 k 位数据字，我们可以产生 2^k 种不同的组合，对于 n 位码字，我们可以产生 2^n 种不同的组合。因为 $n > k$ ，所以可能出现的码字在数量上要比可能出现的数据字要多一些。块编码过程是一对一的，相同的数据字总是被编码成相同的码字。也就是说我们还有 $2^n - 2^k$ 个码字没有用到。我们称这些码字为无效的。图 C.3 对此进行了描绘。

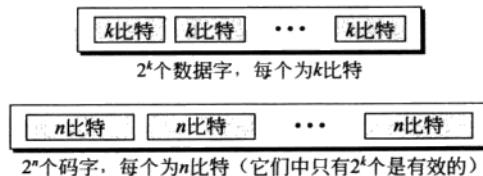


图 C.3 块编码中的数据字和码字

C.2.1 差错检测

如何利用块编码来检测差错呢？如果能够满足以下两个条件，接收方就能够检测出原始码字的改变。

1. 接收方有（或者能够找出）有效码字的列表。
2. 原始码字变成了无效码字。

发送方通过应用了某种编码规则和过程（稍后讨论）的生成器，从数据字中产生码字。发送到接收方的每个码字在传输期间都有可能发生改变。如果接收到的码字与某个有效码字相同，则这个字就被接收，并从中提取出相应的数据字以供使用。如果接收到的码字是无效的，它就被丢弃。但是，如果码字在传输期间受损，但接收到的字却仍然与一个有效的码字相符，那么这个差错就没有被检测出。

C.2.2 汉明距离

差错控制编码的一个最核心的概念就是称为汉明距离的思想。两个字（长度相同）之间的汉明距离（Hamming distance）就是对应位不相同的数量。我们把两个字 x 和 y 之间的汉明距离表示为 $d(x, y)$ 。如果我们对两个字进行 XOR 操作 (\oplus)，然后再计算其结果中 1 的数量，就能很容易地找出它们之间的汉明距离。

C.2.3 最小汉明距离

虽然差错检测码和差错纠正码在处理时的关键点是汉明距离，但是用来设计编码的度量却是最小汉明距离。在一组字中，最小汉明距离（minimum Hamming distance）就是指所有可能成对的字之间的最小的汉明距离。我们用 d_{\min} 来定义一个编码机制的最小汉明距离。为了计算这个值，我们需要找出所有字和字相互之间的汉明距离，然后选择出其中最小的。编码机制 C 被写为 $C(n, k)$ ，并附带有独立的 d_{\min} 表达式。

汉明距离和差错

在研究差错检测或纠正的标准之前，让我们先来讨论一下汉明距离与传输期间发生的差错的关系。如果一个码字在传输时受损，那么发送码字和接收码字之间的汉明距离就是被差错所影响的比特数。换言之，接收码字与发送码字之间的汉明距离就是传输期间受损的比特数。例如，如果发送的码字是 00000，而收到的却是 01101，那么有 3 个比特出了差错，这两个字之间的汉明距离是 $d(00000, 01101) = 3$ 。

用于差错检测的最小距离

如果希望最多能检测出 s 个差错，让我们看看这种编码的最小汉明距离是多少。若传输期间出现了 s 个差错，则发送码字和接收码字之间的汉明距离就是 s 。若我们的编码最多能检测出 s 个差错，那么所有有效码字之间的最小汉明距离必须为 $s + 1$ ，只有这样，刚才接收到的那个码字才不会匹配到任何一个有效的码字。换言之，如果所有有效码字之间的最小汉明距离是 $s + 1$ ，接收到的码字就不可能被错误地认为是另一个码字，

于是差错就可以被检测出。在这里我们还要阐明一个问题：虽然 $d_{\min} = s + 1$ 的编码在某些特殊情况下也能够检测到多于 s 个差错，但是它只能保证一定能检测出 s 个或更少的差错来。

我们可以从几何学的角度来看这个问题。让我们假设发送码字 x 是一个半径为 s 的圆的中心。所有由 $1 \sim s$ 个差错而导致接收不正确的码字就是在这个圆内的或圆周上的点。其他所有的有效码字必须在这个圆之外，如图 C.4 所示。从图中可以看出 d_{\min} 必须是大于 s 的整数，也就是说 $d_{\min} = s + 1$ 。

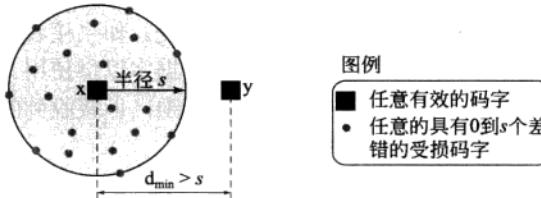


图 C.4 用几何概念来找出差错检测中的 d_{\min}

C.3 线性块码

目前在使用的块编码几乎全都属于称为线性块码 (linear block codes) 的子集。把非线性块码用于差错检测和纠正的情况并不多见，这是因为它们的结构致使理论分析和应用都很困难。因此我们着重讨论线性块码。

线性块码的正式定义需要用到抽象代数（特别是伽罗华域）的知识，它超出了本书的范围，因此我们给出一个非正式的定义。就我们的任务而言，线性块码是这样一种编码，对两个有效码字进行异或（模 2 加法）运算将产生另一个有效码字。

C.3.1 线性块码的最小距离

要找出线性块码的最小汉明距离非常简单。它的最小汉明距离就是具有最少个 1 的非零有效码字中 1 的个数。

简单奇偶检验码

人们最熟悉的差错检测码可能就是简单奇偶检验码 (simple parity-check code)。使用这种编码时， k 位数据字被转换为 n 位码字，其中 $n = k + 1$ 。这个额外的比特称为奇偶位，它的选择应当使该码字中出现的 1 的总数为偶数。虽然有一些应用指明了奇数个 1，但我们只讨论偶数的情况。这一类检测码的最小汉明距离是 $d_{\min} = 2$ ，也就是说这个编码是单比特差错检测码。图 C.5 描绘了编码器（在发送方）和解码器（在接收方）的一种可能的结构。

编码器使用的生成器以 4 位数据字 (a_0, a_1, a_2 和 a_3) 的副本为输入，产生一个奇偶位 r_0 。4 位的数据字再加上这个奇偶位 (parity bit) 就生成了 5 位的码字。奇偶位的加入要使码字中 1 的个数为偶数。通常的做法是把数据字的 4 个比特相加（模 2 加法），得到的结果就是奇偶位。换言之，

$$r_0 = (a_3 + a_2 + a_1 + a_0) \bmod 2$$

如果 1 的个数是偶数，那么得到的结果就是 0；如果 1 的个数是奇数，那么得到的结果就是 1。在这两种情况下，码字中 1 的总数都是偶数。

发送方发送的码字在传输期间可能会受损。接收方接收到一个 5 位的字。接收方的检验器要做的事与发送方的生成器一样，只有一点不同：此时是对 5 个比特的加法。得到的结果是 1 个比特，称为 syndrome。当接收到的码字中的 1 的个数为偶数时，syndrome 就是 0，否则它是 1。

$$s_0 = (b_3 + b_2 + b_1 + b_0 + q_0) \bmod 2$$

这个 syndrome 通过判决逻辑分析器。如果 syndrome 为 0，说明接收到的码字没有差错，这个接收的码字中的数据部分就作为数据字被接受。如果 syndrome 为 1，接收的码字中的数据部分被丢弃，没有产生数据字。

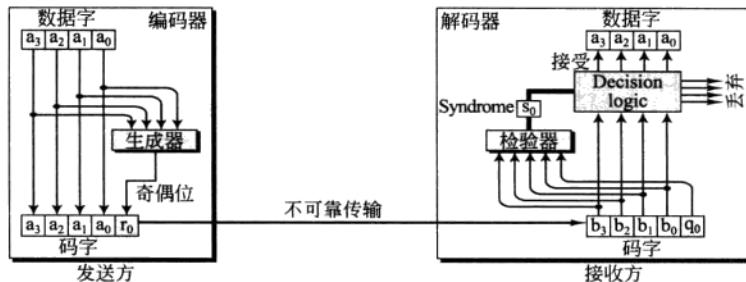


图 C.5 简单奇偶检验码的编码器和解码器

汉明码

现在让我们来讨论称为汉明码的这一类差错纠正码。这种编码最初被设计为 $d_{\min} = 3$ ，也就是说他们能够最多检测出两个差错。

首先让我们找出汉明码中 n 和 k 的关系。我们需要选择一个整数 $m \geq 3$ 。然后，根据 $n = 2^m - 1$ 和 $k = n - m$ ，从 m 中计算出 n 和 k 。检验位的个数 $r = m$ 。例如，如果 $m = 3$ ，那么 $n = 7$ 且 $k = 4$ 。这是一个 C(7, 4) 且 $d_{\min} = 3$ 的汉明码，它能够至少检测出两个差错。我们把码字的产生过程留给专门讨论差错纠正的书籍。

C.4 循环码

循环码是一种特殊的线性块码，它具有一种独特的性质。在循环码（cyclic code）中，如果把一个码字循环地移动（旋转），得到的结果是另一个码字。例如，如果 1011000 是一个码字，我们将其循环左移，那么 0110001 也是一个码字。

C.4.1 循环冗余检验

我们可以通过创建循环码来进行差错纠正。不过为此所需要的理论背景知识超出了本书的范围。在这一小节，我们简单地讨论称为循环冗余检验（cyclic redundancy check, CRC）的一类循

环码，它用在像局域网和广域网这样的网络中。图 C.6 描绘了这种编码器和解码器可能的设计。

在编码器中，数据字为 k 位（此处为 4），码字为 n 位（此处为 7）。通过向数据字的最右边添加 $n - k$ 个 0 来扩展数据字的长度，得到的 n 位结果被输入生成器。生成器使用长度为 $n - k + 1$ 位的除数，这个除数是预先定义且一致同意的。生成器把扩展后的数据字除以除数（模 2 除法），得到的商被丢弃，余数 ($r_2r_1r_0$) 被附加到数据字上，从而产生码字。

解码器收到可能受损的码字。 n 位码字的副本被输入到检验器中，这个检验器与生成器完全一样。由检验器生成的余数就是 $n - k$ 位的 syndrome，它被输入到判决逻辑分析器中。这个分析器的功能非常简单。如果 syndrome 全部为 0，则码字最左边的 k 位作为数据字被接受（解释为无差错），否则这个 k 位被丢弃（有差错）。

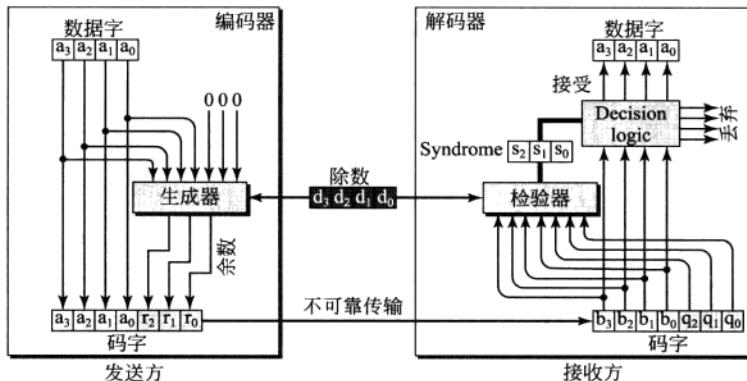


图 C.6 CRC 编码器和解码器

编码器

让我们更详细地了解一下编码器。编码器输入数据字，并用 $n - k$ 个 0 对其进行扩展。然后，它把扩展后的数据字除以除数，如图 C.7 所示。模 2 的二进制除法的过程与我们所熟悉的十进制数的除法过程一样。不过，正如我们在本章开始时提到的，此时的加法和减法都一样，我们都用 XOR 运算来实现。乘法可以用 AND 运算实现。

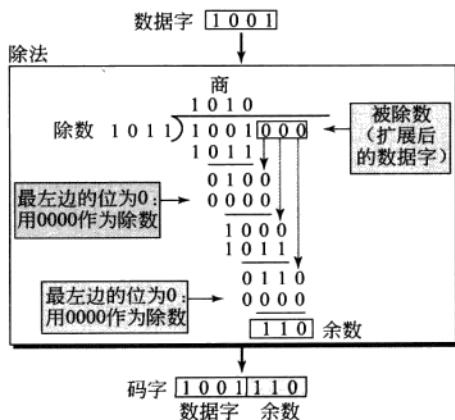


图 C.7 CRC 编码器中的除法

与十进制除法一样，这个过程是逐步完成的。在每一步的计算中，除数的副本与被除数中的 4 位进行 XOR 运算。XOR 运算的结果（余数）是一个 3 位数（在这种情况下），加上从被除数中挪下来的一位后，就变成了 4 位数，被用于下一步计算。此类除法中有一点很重要，需要我们记住。如果被除数（或在每一步中用做被除数的那部分）的最左边一位为 0，那么这一步的计算就不能使用正常的除数，我们要使用全 0 的除数。当被除数中再没有待处理的位剩下时，我们就得到了最后的结果。此时的 3 位余数就形成了检验位 (r_2 、 r_1 和 r_0)。它们被附加到数据字上产生码字。

解码器

码字在传输过程中可能被改变了。解码器的除法处理过程与编码器一样。此时除法的余数就是 **syndrome**。如果 syndrome 为全 0，就说明没有出现差错，数据字从接收到的码字中剥离出来并被接受。否则，所有码字都被丢弃。图 C.8 描绘了这两种情况。

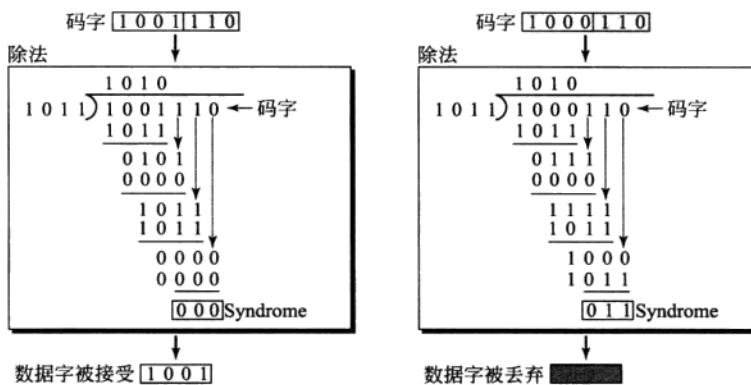


图 C.8 两种情况下 CRC 解码器中的除法

左手这张图描绘了没有差错发生时的 syndrome 的值，它是 000。右手这张图描绘了出现一个差错时的情况，此时的 syndrome 不是全 0（它是 011）。

除数

你可能会好奇这个除数 1011 是如何选择的。这里有一些策略，不过我们把对这个问题的讨论留给专门介绍差错检测的书籍。

C.4.2 循环码的优点

循环码在检测单比特、双比特和奇数比特差错以及突发差错时都有非常好的表现。它们很容易用硬件和软件来实现。它们的硬件实现在速度上可以做到非常快。

C.4.3 其他循环码

我们在这一小节讨论的循环码非常简单。通过简单的代数计算就能得到检验位和 syndrome。不过，目前还有一些更强壮的循环码被用于差错检测和纠正，如 **Reed-Solomon 码**。

附录 D 检验和

在附录 C 中，我们讨论了一些差错检测码。在 TCP/IP 协议族的上三层中，占主导地位的是一种特殊的差错检测方法，称为检验和。我们将在此附录中讨论这种方法。

D.1 传统的检验和

让我们先讨论一下因特网中使用的传统检验和，然后再看看不同于传统检验和的一些新的提议。

D.1.1 思想

传统检验和的思想非常简单，我们通过一个简单的例子来说明。

例 D.1

假设我们要发送到终点的数据是五个 4 位数的序列。除了发送这几个数之外，我们还要发送这些数之和。例如，假设这组数是 (7, 11, 12, 0, 6)，那么我们要发送 (7, 11, 12, 0, 6, 36)，其中的 36 是五个原始数之和。接收方把五个数相加，并将结果与这个和进行比较。如果两个数相同，则接收方认为没有差错，就接受这五个数并丢弃和。否则，说明有某处出现差错，数据不能被接受。

二进制反码加法

在上一个例子中存在一个很大的缺陷。每个数都能写成 4 位的字（它们都小于 15），但和却不能。一种解决办法是使用二进制反码 (one's complement) 运算。在这种运算中，我们可以用 n 位来表示从 0 到 $2^n - 1$ 之间的无符号整数。如果一个数多于 n 位，那么多余的最左边的位被加到最右边的位上（绕回）。

例 D.2

在上一个例子中，十进制数 36 用二进制表示就是 $(100100)_2$ 。要把它变成 4 位数，我们要把左边多余的位与最右边的 4 位相加，如下所示：

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

我们发送的是 (7, 11, 12, 0, 6, 6)，其中和是 6，而不是 36。接收方可以对前五个数用二进制反码相加。如果结果是 6，这个数就被接受，否则拒绝。

检验和

如果要让接收方的工作更加轻松一些，我们可以发送和的反码，称为检验和。在二进

制反码运算中，一个数的反码可以通过对所有比特求反（所有 1 变为 0，所有 0 变为 1）得到，它与用 $2^n - 1$ 减去这个数的效果是一样的。二进制反码运算中有两个 0：正 0 和负 0，它们互为反码。正 0 就是所有位全部为 0，负 0 就是所有位全部为 1（也就是 $2^n - 1$ ）。如果我们把一个数与它的反码相加，就会得到负 0（所有位全部为 1）。当接收方把 6 个数（包括检验和）全部相加后，应当得到负 0。接收方可以把这个结果再次求反，就能得到正 0。

例 D.3

让我们在例 D.2 中使用检验和的思想。发送方把五个数用二进制反码运算相加后得到的和等于 6。然后发送方对这个结果求反得到检验和等于 9，也就是 $15 - 6 = 9$ 。请注意 $6 = (0110)_2$ ，而 $9 = (1001)_2$ ，它们互为反码。发送方发送这五个数以及检验和 $(7, 11, 12, 0, 6, 9)$ 。如果传输过程中没有损伤，接收方接收到的是 $(7, 11, 12, 0, 6, 9)$ ，把它们用二进制反码运算相加就得到 15。发送方对 15 求反后得到 0。这就表明数据没有受到损伤。图 D.1 描绘了这个过程。

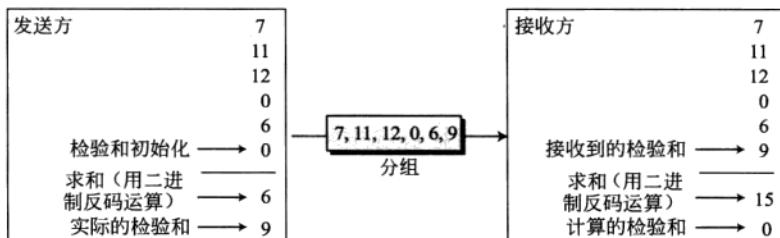


图 D.1 例 D.3

D.1.2 因特网的检验和

传统上，因特网使用了一个 16 位的检验和。发送方和接收方按照表 D.1 所描述的步骤执行。发送方需要五个步骤，而接收方只需要四步。

表 D.1 计算传统检验和的过程

发送方	接收方
1. 报文被划分为 16 位的字。	1. 报文被划分为 16 位的字。
2. 检验和的值初始化置为 0。	2. 用二进制反码加法把所有的字相加。
3. 用二进制反码加法把所有的字相加，包括检验和。	3. 对和求反，从而生成新的检验和。
4. 对和求反，从而形成检验和。	4. 如果检验和的值是 0，报文被接受，否则，报文被拒绝。
5. 检验和随数据一起发送。	

算法

我们可以用图 D.2 中的流程图来说明检验和的计算算法。根据这个算法，不论用哪种编程语言都能很容易地写出相应的程序。

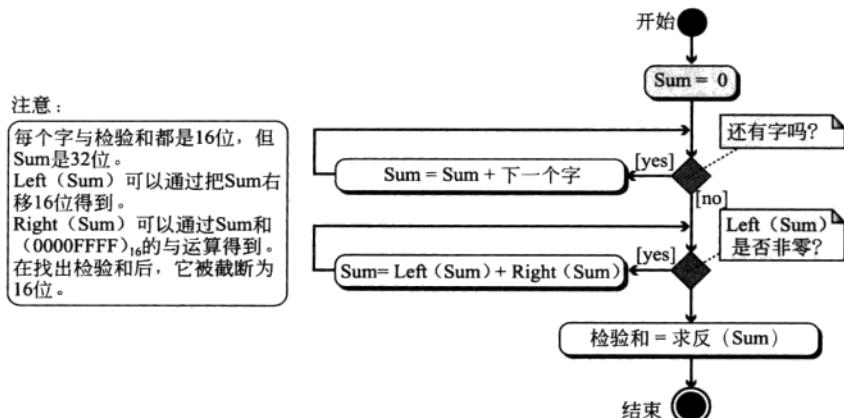


图 D.2 计算传统检验和的算法

性能

传统的检验和使用了较少的比特数来检测任意长度（有时达到上千比特）的报文中的差错。但是在差错检测能力上，它的强壮性不如 CRC。例如，如果一个字的值变大了，而另一个字的值变小了，且减少量恰好等于增加量，那么这两个差错就无法被检测出来，因为和与检验和仍然保持没变。另外，虽然有多个值都增加了，但是若和与检验和保持未变，差错还是不能被检测出。Fletcher 和 Adler 建议了一些加权的检验和，其中每个字都要乘以一个数（它的权），这个数与它在文本中的位置有关。这样就消除了我们提到的第一个问题。但是目前因特网的倾向是用 CRC 来取代检验和，特别是在设计新的协议时。

D.2 Fletcher 检验和

正如我们在前面提到的，传统的检验和计算中存在一个很重要的问题。如果两个 16 位的项在传输时被转置了，那么检验和就不能发现这个差错。原因是传统的检验和没有加权，也就是说它对所有的数据项一视同仁。换言之，数据项的顺序对计算并不重要。Fletcher 检验和的设计是要对每一个数据项按其位置加权。

Fletcher 提出了两种算法：8 位的和 16 位的。第一种是 8 位的 Fletcher，它对 8 位的数据项进行计算，并生成 16 位的检验和。第二种是 16 位的 Fletcher，它对 16 位的数据项进行计算，并生成 32 位的检验和。

八位 Fletcher

八位 Fletcher 是在数据八位组（字节）上进行计算，然后生成 16 位的检验和。计算以模 256（即 2^8 ）进行，这就表示中间的结果都要除以 256，并保留余数。这个算法使用了两个累加器：L 和 R。第一个累加器简单地把各数据项相加；第二个累加器在计算时加入权。八位 Fletcher 算法有多种变型，我们在图 D.3 中给出了其中一种简单的算法。

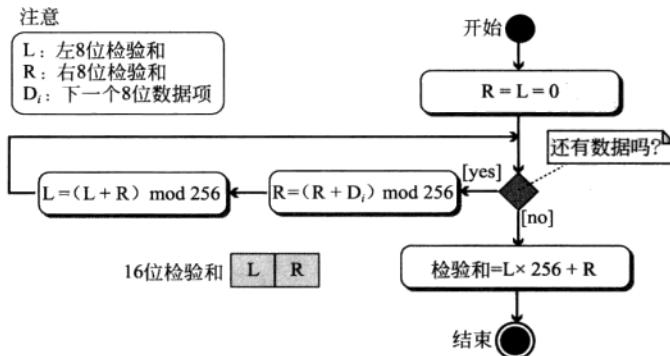


图 D.3 八位 Fletcher 检验和的计算算法

可以证明，累加器 L 是数据项的加权和。我们有：

$$R = D_1 + D_2 + \dots + D_n$$

$$L = nD_1 + (n-1)D_2 + \dots + D_n$$

例如，如果在传输过程中 D₁ 和 D₂ 对换了，那么接收方计算的 L 就与发送方计算出的不一样。

作为一个例子，让我们计算字符串“Forouzan”的八位 Fletcher 检验和。我们把每一个字符转换为相应的 ASCII 码值，并计算 R 和 L 的值如表 D.2 所示。

表 D.2 8 位 Fletcher 检验和的例子

字节	D _i	R = 0	L = 0
F	70	R = 0 + 70 = 70	L = 0 + 70 = 70
o	111	R = 70 + 111 = 181	L = 70 + 181 = 251
r	114	R = 181 + 114 = 39	L = 251 + 39 = 34
o	111	R = 39 + 111 = 150	L = 34 + 150 = 184
u	117	R = 150 + 117 = 11	L = 184 + 11 = 195
z	122	R = 11 + 122 = 133	L = 195 + 133 = 72
a	97	R = 133 + 97 = 230	L = 72 + 230 = 46
n	110	R = 230 + 110 = 84	L = 46 + 84 = 130

$$\text{检验和} = L \times 256 + R = 33364$$

这个 16 位的检验和是(8254)₁₆。请注意，检验和实际上是由 L = (82)₁₆ 和 R = (54)₁₆ 的拼接。换言之，当 R 和 L 计算出后，L 就成为最左边的字节，而 R 成为最右边的字节。

十六位 Fletcher

十六位 Fletcher 是对 16 位数据项进行计算，并生成 32 位的检验和。这里的计算是按模 65536 进行的。

D.3 Adler 检验和

Adler 检验和是 32 位的检验和。图 D.4 用流程图来描绘了它的一种简单算法。

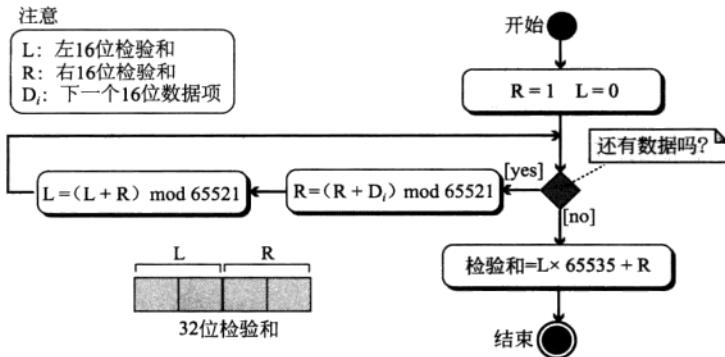


图 D.4 Adler 检验和的计算算法

它与 Fletcher 检验和相似，但有三点不同。第一，计算在单个字节上进行，而不是一次两个字节地进行。第二，模数是一个素数（65521）而不是 65536。第三，L 初始化为 1 而不是 0。现已证明了，在某些数据组合下，用素数做模数具有更好的检测能力。

让我们计算字符串“Forouzan”的 Adler 检验和。我们把每一个字符转换成相应的 ASCII 码值，并计算 R 和 L 的值如表 D.3 所示。此时的 32 位检验和是 $(0E8A0355)_{16}$ 。请注意，这个检验和实际上是 $L = (0E8A)_{16}$ 和 $R = (0355)_{16}$ 的拼接。

表 D.3 Adler 检验和的例子

字节	D_t	R = 1	L = 0
F	70	R = 1 + 70 = 71	L = 0 + 71 = 71
o	111	R = 71 + 111 = 182	L = 71 + 182 = 253
r	114	R = 182 + 114 = 296	L = 253 + 296 = 549
o	111	R = 296 + 111 = 407	L = 549 + 407 = 956
u	117	R = 407 + 117 = 524	L = 956 + 524 = 1480
z	122	R = 524 + 122 = 646	L = 1480 + 646 = 2126
a	97	R = 646 + 97 = 743	L = 2126 + 743 = 2869
n	110	R = 743 + 110 = 853	L = 2869 + 853 = 3722
检验和 = $3722 \times 65536 + 853 = 243925845$			

附录 E HTML、XHTML、 XML 和 XSL

E.1 HTML

超文本置标语言（Hypertext Markup Language, HTML）是一种用来创建网页的语言。术语“置标语言”源自图书出版业。一本书在录入完毕并打印成稿后，编辑要阅读原稿并在上面做一些标记。这些标记告诉排版人员如何对文本进行格式处理。例如，若编辑想让一行中的某一部分用粗体字印刷，他就在这一部分下面画一道波纹线。类似地，网页上的数据是由浏览器来进行格式处理和解释的。

E.1.1 标签

HTML 文档由文本和一些命令组成，这些命令定义了浏览器在显示文档内容时应当如何对这些文本进行格式处理和解释。HTML 中的命令称为 **标签**（tag）。当浏览器遇到标签时，它不显示标签，而是对跟在后面的文本进行格式处理或解释。因为一个文档中的不同部分需要不同的格式或解释，所以标签的设计是成对出现的：开始标签和结束标签。不过，有些标签可能没有显式的结束符号，因为在文本中隐含了结束。下面给出了一对标签的通用格式。

```
<tagName> ... </tagName>
```

<tagName> 是开始标签，</tagName> 是结束标签。在它们之间可以有一行或多行文本。tagName 可以是小写字母，也可以是大写字母，不过在本附录中我们都用小写字母。

除了名字之外，标签还可以有一组属性（attribute）及其相应的值（value）。通过属性和值就可以定义更多的信息，我们稍后再讨论。

```
<tagName attribute = value attribute = value ...> ... </tagName>
```

在这节附录中，我们只讨论有限的几个标签，目的是为了说明 HTML 的总体思想。要想了解完整的标签列表，可以参考专门介绍 HTML 的书籍。

文档标签

在 HTML 中，网页被称为文档。在 HTML 中，一个文档（document）的开始和结束都要用预先定义的标签来表示，它们把 HTML 文档与其他类型的文档区分开。这个开始标签

是<html>, 结束标签是</html>。在这两个标签之间的可以是文本，也可以是用于解释文本的其他标签。下面给出了使用这两个标签的文档格式。

```
<html>
  ...
</html>
```

虽然文档的内容不需要缩进，但为了清晰易读通常我们会这样做。

头和主体标签

HTML 文档由两部分组成：头部和主体。文档的头部通常只包含文档的标题，但也可能有一些其他信息资料。在这里我们只使用了文档标题。请注意，文档的标题不会被浏览器显示出来，它只具有信息价值。文档的主体中嵌入了 HTML 文档的文本、图片、链接以及其他信息。下面给出了包括头部和主体的一个 HTML 文档。

```
<html>
  <head>
    <title> TCP/IP Protocol Suite </title>
  <head>
  <body>
    ...
  </body>
</html>
```

我们只用了单行的文档标题，不过文档标题可以扩展为多行。还要注意到我们没有对文档标题做任何格式处理，因为浏览器并不会显示它。

段落和行标签

可能我们首先需要了解的是那些把文本划分为段落和组织成行的标签。段落标签是<p>和</p>，折行标签是
和</br>。使用一对中间没有任何文本的折行标签将会在文本中显示出一个空行。下面给出了使用这些标签的一个例子（在文档的主体中）。

```
<p>
  <br> This is the first line </br>
  <br> </br>
  <br> This is the third line </br>
</p>
```

字形标签

下一组标签使得我们能够以粗体字或斜体字来显示文本。粗体标签是和，斜体标签是<i>和</i>。下面给出使用这些标签的一个例子。

```
<b> This is the first line </b>
<i> This is the third line </i>
```

标题标签

在 HTML 中，我们最多可以定义六级标题，不过通常所需的标题级数并不多。标题标签是<h_n>和</h_n>，其中 n 定义了级数（1~6）。例如，下面给出了我们如何在文档中定义两级标题。

```
<br><h1> TCP/IP Protocol Suite </h1><br>
<br><h2> A Book by B. Forouzan </h2><br>
```

第一行用较大的字体来显示，第二行用较小的字体来显示。

列表标签

在 HTML 中，我们可以定义两种类型的列表：不排序列表和排序列表。不排序列表的标签是``和``。列表中的每一项还需要用到标签``和``。列表中的项通常前面会有项目符号，后面会由浏览器插入一个折行符。排序列表的标签是``和``。列表中的每一项还需要标签``和``。然后，这个列表中的项目会被编号，且浏览器会在每一项结束的地方插入一个折行符。下面给出了一个不排序列表的例子和一个排序列表的例子（分别在两个文档中）。

```
<ul>
  <li> CIS 011 </li>
  <li> CIS 015 </li>
  <li> CIS 051 </li>
</ul>                                <ol>
                                         <li> CIS 011 </li>
                                         <li> CIS 015 </li>
                                         <li> CIS 051 </li>
</ol>
```

图片标签

我们也可以在文档中包含图片（图形）。但是图片不能直接嵌入到文档中，图片标签只是包含了对保存图片的位置的引用。浏览器会负责到标签所定义的位置上获取图片，并在标签出现的地方显示它。图片标签的格式如下所示：

```
<img attribute = "value" attribute = "value" ...>
```

可用于图片标签的几个属性包括：

- **src** 根据 `src`（资源）属性定义的地址（URL）所指向的位置可以找到该图片。这个值需要使用双引号。
 - **align** `align` 属性定义了该图片应当如何与文档中的文本对齐。这个值可以是 `top`、`middle` 和 `bottom`。
 - **alt** `alt`（替代）属性所定义的文本，在图片万一不能显示的情况下用来替代该图片。
- 下面给出了图片标签的一个例子：

```
<img src = "mypicture.gif" alt = "Family Picture" align = "middle">
```

链接标签

HTML 依据的主要思想是提供超链接。HTML 文档中的链接使用户能够从世界上任何地方的一个文档导航到另一个可能远在千里之外的文档。链接标签是`<a>`和``。到另一个文档的链接由两部分组成：该文档的 URL 和锚（anchor）。锚所定义的文本（下划线或颜色表示）或图片会在文档中显示出来，是用户能看到的。当用户点击这个锚时，就会被导引到一个新的文档中。下面给出了链接标签的格式：

```
<a href = "value" > anchor </a>
```

下面给出了链接标签的一个例子：

To find the site of this book, please go to:

[McGraw-Hill](http://www.mhhe.com/engcs/compsci/forouzan/)

在冒号之后只有文本“McGraw-Hill”会被显示出来。当用户点击这个锚后，浏览器找到 McGraw-Hill 网站并显示其中的内容。

表单和输入标签

HTML 2.0 在前一个版本的 HTML 基础上增加了表单（form）的概念。在 HTML 1.0 中，用户可以下载 HTML 文档并浏览它。作为一种商业工具，为了让顾客能够一边浏览，一边随心所欲地购买商品，就引入了表单的概念。表单是一组文本框（按钮）的集合，用户能够填入并把信息发送给该网站。下面给出的例子中使用了表单和输入标签。

```
<form action = ... method = POST>
  <input name = "user" size = ...>
  <input name = "address" size = ...>
  ...
</form>
```

其他标签

还有很多其他标签，我们留给专门介绍 HTML 的书籍。

E.1.2 XHTML

扩展 HTML (XHTML) 是 HTML 的新版本（在版本 4.0 之后），它囊括了 XML 和 XSL（稍后定义）。总体来说，XHTML 类似于 HTML，但是对语言规则的使用更加严格。特别地，在 XHTML 中有一些变化，讨论如下：

- 所有标签和属性都必须是小写字母。
- 结束标签是必需的。如果 HTML 中没有相应表示结束的标签，在 XHTML 中需要在大于号之前插入斜线。例如，XHTML 中的图片标签是<image .../>。
- 属性必须用引号括起来（不管是字符串还是数值）。
- 标签嵌套必须恰当。
- 每个 XHTML 文档都必须有一个文档类型，就像 XML 和 XSL 中定义的一样（接下来讨论）。

E.2 XML 和 XSL

HTML 使用预先定义的标签对文档进行格式处理和解释。但是 HTML 无法提供像 C 语言那样的编程语言能够定义的数据结构和数据表示。用类似 C 语言的编程语言编写出来的程序能够完成以下两个独立的任务：

1. 我们可以在程序中定义数据结构，如数组或记录，并用适当的值对其进行初始化。例如，我们可以定义一个学生记录，其中的字段定义为姓名、学号、出生日期等等。
2. 我们可以用像 printf 这样的打印或格式化函数来对已定义记录的内容进行格式处理和打印。

耐人寻味的是这两个任务能够做到互不干扰。我们可以修改保存在记录中的内容，而不需要改变打印格式。我们也可以通过改变打印格式的方法，把相同的记录用不同的格式打印出来。这两个任务分别被赋予 XML 和 XSL。

扩展置标语言（ XML ）

XML 作为一种语言，它允许用户定义数据表示或数据结构，并为该结构中的每个部分（字段）分配相应的值。换言之， XML 是定制的 HTML，它使得用户能够定义自己的标签，如<name>、<id>等等。唯一的限制是用户必须遵守 XML 定义的规则。例如，下例显示了我们如何利用三个字段： name 、 id 和 birthday 来定义学生记录。

```
<?xml version="1.0"?>
<student>
    <name> George Brown </name>
    <id> 2345 </id>
    <birthday> 12-08-82 </birthday>
</student>
```

扩展风格语言（ XSL ）

在 XML 文档中被定义并被赋予初始值的数据还需要使用另外一种语言，一种风格语言，来定义这些数据应当如何呈现。要做到这一点的方法之一是使用 XSL 。 XSL 用格式化的语句，甚或是重复的语句来定义如何显示 XML 文档中定义的数据。换言之， XSL 不是真正的 HTML 文档，而是要应用到一个 XML 文档中的风格。我们把 XSL 格式的详细说明留给专门介绍网页设计的书籍。

附录 F Java 中的客户-服务器编程

在本附录中，我们要简单地了解一下在 Java 中使用套接字接口的客户-服务器编程。Java 提供的网络编程能够访问顺序服务器和并发服务器。在 Java 中为顺序服务器编写程序会比较直截了当，不过编写并发服务器程序就要复杂得多，因为它们需要用到 Java 的线程。在本附录中，我们只关心顺序编程，不管是为 UDP 的，还是 TCP 的。我们把并发服务器的编程留给专门介绍 java 网络编程的书籍。

在本附录的第一小节中，我们给出了两个简单的程序：使用 UDP 服务的 echo 客户程序和 echo 服务器程序。在第二小节中，我们还是给出了两个简单的程序：使用 TCP 服务的 echo 客户程序和 echo 服务器程序。

为了编写服务器程序，Java 使用了几个类。其中一些类是特别为使用 UDP 而设计的，另外一些类则只能与 TCP 一起使用。DatagramSocket 类及一些方法被用于为 UDP 创建套接字对象。ServerSocket 类及一些方法被用于为 TCP 创建套接字对象。UDP 还使用了 DatagramPacket 类，它帮助生成数据报分组。

我们还使用了若干个 input/output 类、stream 类和 buffered 类，这些都是 Java 程序员非常熟悉的。

F.1 UDP 程序

我们首先关注的是 UDP 的客户和服务器程序。表 F.1 给出了 UDP echo 服务器程序。它是表 17.1（第 17 章）中程序的 Java 版本。在第 11 行我们用 DatagramSocket 类创建了一个套接字。然后在第 15 行和 16 行用 DatagramPacket 类创建了一个数据报分组。第 17 行的阻塞 receive 方法阻塞了程序的运行，直到有来自客户的分组到达。然后它再次使用 DatagramPacket 类来创建一个新的分组（第 19~21 行）。最后，在第 22 行发送这个响应数据。

表 F.1 使用 UDP 服务的 echo 服务器程序

```
01 //UDP echo 服务器程序
02 import java.io.*;
03 import java.net.*;
04 public class UDPEchoServer
05 {
```

续表

```

06     public static void main (String args [ ]) throws Exception
07     {
08         byte [ ] recvBuf = new byte [256]; // 接收缓存
09         byte [ ] sendBuf = new byte [256]; // 发送缓存
10         // 创建服务器套接字
11         DatagramSocket socket = new DatagramSocket (7);
12         for ( ; ; ) // 永远循环
13         {
14             // 收到一个数据报
15             DatagramPacket receivePacket =
16                 new DatagramPacket (recvBuf, recvBuf.length);
17             socket.receive (receivePacket);
18             // 发送这个数据报
19             DatagramPacket sendPacket =
20                 new DatagramPacket (sendBuf, sendBuf.length,
21                     receivePacket.getAddress (), receivePacket.getPort ());
22             socket.send (sendPacket);
23         } // 结束 for 循环
24     } // 结束 main
25 } // 结束 class

```

表 F.2 给出了客户程序（第 17 章表 17.2 中的程序的 java 版本）。第 11 行创建一个套接字对象。第 13 行说明如何使用 InetAddress 类中的 getByName 方法来找出服务器的地址。在第 15、16 和 17 行中，我们从键盘读用户数据并创建一个发送缓存。第 19~20 行发送数据。程序在第 22 行和 23 行接收回送的数据。最后我们用第 26 行和 27 行显示回送的数据。请注意，我们假设服务器为 echo 程序使用了端口 7。

表 F.2 使用 UDP 服务的 echo 客户程序

```

01 //UDP echo 客户程序
02 import java.io.*;
03 import java.net.*;
04 public class UDPEchoClient
05 {
06     public static void main (String args[ ]) throws Exception
07     {
08         byte[ ] recvBuf = new byte[256]; // 接收缓存
09         byte[ ] sendBuf = new byte[256]; // 发送缓存
10         // 创建客户套接字

```

```

11   DatagramSocket socket = new DatagramSocket();
12   // 找出服务器地址
13   InetAddress serverAddr = InetAddress.getByName ("server name");
14   // 输入字符串
15   BufferedReader In = new BufferedReader(new InputStreamReader(System.in));
16   String sendString = In.readLine ();
17   sendBuf = sendString.getBytes();
18   // 发送数据报
19   DatagramPacket sendPacket =
20       new DatagramPacket (sendBuf, sendBuf.length, serverAddr, serverPort);
21   socket.send (sendPacket);
22   // 接收数据报
23   DatagramPacket recvPacket = new DatagramPacket (recvBuf, recvBuf.length);
24   socket.receive (recvPacket);
25   // 输出字符串
26   String recvString = new String (recvPacket.getData());
27   System.out.println ("Received from server:" + recvString);
28   // 关闭套接字
29   socket.close ();
30 } // 结束 main
31 } // 结束 class

```

F.2 TCP 程序

与前面一样，为了使讨论更加简单，我们假设的是一个顺序的 TCP 服务器（而不是并发服务器）。表 F.3 给出了第 17 章表 17.3 中的程序的顺序服务器的版本。在第 11 行，我们创建了一个 TCP 套接字。在第 15 行，我们用 ServerSocket 类中的一个方法来创建监听套接字。第 17 行和 18 行使用了 InputStream 和 OutputStream 类创建了两个流对象，用来接收和发送数据流。由客户发送来的字符串可能在不同的报文段中到达，因此我们使用了一个 while 循环来读取所有报文段中的数据，然后再把这些数据回送给客户。请注意，我们假设服务器为 echo 程序使用了端口 7。

表 F.3 使用 TCP 服务的 echo 服务器程序

```

01 //TCP echo 服务器程序
02 import java.io.*;
03 import java.net.*;
04 public class TCPEchoServer
05 {

```

续表

```

06  public static void main (String args [ ] ) throws Exception
07  {
08      byte [ ] buffer = new byte [256]; // 字节缓存
09      int br = 0; // 读入的字节数
10      // 创建服务器套接字
11      ServerSocket listenSocket = new ServerSocket (7);
12      for ( ; ; ) // 永远循环
13      {
14          // 创建连接套接字为客户服务
15          Socket connectSocket = listenSocket.accept();
16          // 创建输入和输出流以接收和发送数据
17          InputStream in = connectSocket.getInputStream();
18          OutputStream out = connectSocket.getOutputStream();
19          // 从流中读出和写入
20          while ((br = in.read(buffer)) > 0)
21          {
22              out.write (buffer, 0, br);
23          } // 结束 while 循环
24          connectSocket.close();
25      } // 结束 for 循环
26  } // 结束 main
27 } // 结束 class

```

表 F.4 给出了相应的客户程序。在第 13 行我们用 `Socket` 类创建了一个连接套接字。第 15~17 行从键盘读数据并把它们保存到发送缓存中。我们在第 19 行和 20 行发送这个数据，并在第 23~27 行接收返回的数据。第 29 行和第 30 行打印接收到的数据。

表 F.4 使用 TCP 服务的 echo 客户程序

```

01 //TCP echo 客户程序
02 import java.io.*;
03 import java.net.*;
04 public class TCPEchoClient
05 {
06     public static void main (String args[ ] ) throws Exception
07     {
08         byte[ ] recvBuf = new byte[256]; // 接收缓存
09         byte[ ] sendBuf = new byte[256]; // 发送缓存
10         int tbr = 0; // 接收的字节总数

```

```
11     int br; // 接收字节数
12     // 创建套接字
13     Socket socket = new Socket("server name", 7);
14     // 输入字符串
15     BufferedReader In = new BufferedReader(new InputStreamReader(System.in));
16     String sendString = In.readLine ();
17     sendBuf = sendString.getByte();
18     // 发送数据
19     OutputStream out = socket.getOutputStream ();
20     out.write (sendBuf);
21     // 接收数据
22     InputStream in = socket.getInputStream ();
23     while (tbr < sendBuf.length)
24     {
25         br = in.read (recvBuf, tbr, sendBuf.length - tbr);
26         tbr = tbr + br;
27     }
28     // 输出回送的字符串
29     String recvString = new String (recvBuf);
30     System.out.println ("Received from server:" + recvString);
31     // 关闭套接字
32     socket.close ();
33 } // 结束 main
34 } // 结束 class
```

G.1 端口号

表 G.1 列出了我们在本书中提到的所有端口号。

表 G.1 端口号与端口

端口号	UDP 或 TCP	协议
7	UDP/TCP	ECHO
13	UDP/TCP	DAYTIME
19	UDP/TCP	CHARACTER GENERATOR
20	TCP	FTP-DATA
21	TCP	FTP-CONTROL
22	TCP	SSH
23	TCP	TELNET
25	TCP	SMTP
37	UDP/TCP	TIME
67	UDP	DHCP-SERVER
68	UDP	DHCP-CLIENT
69	UDP	TFTP
70	TCP	GOPHER
79	TCP	FINGER
80	TCP	HTTP
110	TCP	POP-3
111	UDP/TCP	RPC
143	TCP	IMAP
161	UDP	SNMP
162	UDP	SNMP-TRAP
179	TCP	BGP
443	TCP	HTTPS
520	UDP	RIP

G.2 RFC

在表 G.2 中，我们列出了与本书中的内容直接相关的 RFC。更多的信息可查找网址 <http://www.rfc-editor.org>。

表 G.2 各协议的 RFC

协议	RFC
ARP	826、1029、1166 和 1981
BGP	1654、1771、1773、1997、2439、2918 和 3392
DHCP	3396 和 3342
DNS	1034、1035、1996、2535、3008、3658、3755、3757 和 3845
Forwarding	1812、1971 和 1980
FTP	959、2577 和 2585
HTTP	2068 和 2109
ICMP	792、950、956、957、1016、1122、1256、1305 和 1987
IPMPv6	2461、2894、3122、3810、4443 和 4620
IPv4 Addressing	917、927、930、932、940、950、1122 和 1519
IPv4	760、781、791、815、1025、1063、1071、1141、1190、1191、1624 和 2113
IPv6	1365、1550、1678、1680、1682、1683、1686、1688、1726、1752、1826、1883、1884、1886、1887、1955、2080、2373、2452、2463、2465、2466、2472、2492、2545 和 2590
IPv6 Addressing	2375、2526、3513、3587、3789 和 4291
IPv6	2460、2461 和 2462
MIB	2578、2579 和 2580
MIME	2046、2047、2048 和 2049
Mobile IP	1701、2003、2004、3024、3344 和 3775
MPLS	3031、3032、3036 和 3212
Multicast Routing	1584、1585、2117 和 2362
Multimedia	2198、2250、2326、2475、3246、3550 和 3551
OSPF	1583 和 2328
POP3	1939
RIP	1058 和 2453
SCTP	4820、4895、4960、5043、5061 和 5062
SMTP	2821 和 2822
SNMP	3410、3412、3415 和 3418
SSH	4250、4251、4252、4253、4254 和 4344
TCP	793、813、879、889、896、1122、1987、1988、1993、1975、2018、2581、3168 和 3782
TELNET	854、855、856、1041、1091、1372 和 1572
TFTP	906、1350、2347、2348 和 2349
UDP	768
WWW	1614、1630、1737 和 1738

G.3 联系地址

表 G.3 给出本书所提到过的各种机构的联系地址。

表 G.3 联系地址

ATM Forum	International Telecommunication Union
Presidio of San Francisco	Place des Nations CH-1211
P.O. Box 29920	Geneva 20 Switzerland
572B Ruger Street	intwww.itu.int/home
San Francisco, CA 94129-0920	
www.atmforum.com	
Federal Communications Commission	Internet Corporation for Assigned Names and Numbers (ICANN)
445 12th Street S.W.	4676 Admiralty Way, Suite 330
Washington, DC 20554	Marina del Rey, CA 90292-6601
www.fcc.gov	www.icann.org
Institute of Electrical and Electronics Engineers (IEEE)	Internet Engineering Task Force (IETF)
Operations Center	E-mail: ietf-infor@ietf.org
445 Hoes Lane	www.ietf.org
Piscataway, NJ 08854-1331	
www.ieee.org	
International Organization for Standardization (ISO)	Internet Society (ISOC)
1, rue de Varembe	1775 Weihi Avenue, Suite 102
Case postale 56	Reston, VA 20190-5108
CH-1211 Geneva 20 Switzerland	www.isoc.org
www.iso.org	

词 汇 表

1000Base-CX, 1000Base-LX, 1000base-SX, 1000Base-T IEEE 802.3 工作组制订的 1 Gbps 数据率以太网实现标准。

100Base-FX, 100Base-T4, 100Base-TX, 100Base-X IEEE 802.3 工作组制订的 100 Mbps 数据率快速以太网实现标准。

10Base2, 10Base-F, 10Base-E, 10Base-L IEEE 802.3 工作组制订的 10 Mbps 数据率细缆以太网实现标准。

10Base5 IEEE 802.3 工作组制订的 10 Mbps 数据率粗缆以太网实现标准。

10GBase-L IEEE 802.3 工作组制订的 10 Gbps 数据率标准。

Abstract Syntax Notation 1 (ASN.1, 抽象语法记法 1) 使用抽象语法定义协议数据单元 (PDU) 结构的一种形式化语言。

access control (接入控制) 一种安全服务，可防止对数据的非授权存取。

Acknowledgement (确认) 由接收者发送的一个分组以显示正确接收。

active attack (主动攻击) 一种可以改变数据或损害系统的攻击。

active close (主动关闭) 由客户端关闭 TCP 连接。

active document (活动文档) 在万维网中，在本地使用 Java 执行的文档。

active open (主动打开) 由客户端创建一条到服务器的连接。

additive cipher (加法加密方法) 最简单的单态字符加密方法，每一个字符通过将其本身的值与密钥相加的方式加密。

additive increase (加法增大) 慢启动使用的一种拥塞避免策略，窗口大小仅增加一个报文段而不是指数递增。

Address (地址) 识别分组的源点或终点的整数。

address mask (地址掩码) 一个 32 位数，最左边的一些 1 定义了网络 id。

Address Resolution Protocol (ARP, 地址解析协议) 在 TCP/IP 中，当因特网地址已知时，获得该结点物理地址的一种协议。

address space (地址空间) 一种协议使用的地址总数。

Advanced Encryption Standard (AES, 高级加密标准) 由 NIST 发布的一种 non-feistel 对称密钥块加密方法。

Advanced Networks and Services (ANS, 高级网络和服务公司) 1995 年以后因特网的拥有者和运行者。

Advanced Networks and Services Networks (ANSNET) 高速因特网主干网。

Advanced Research Project Agency (ARPA, 远景研究规划局) 出资建造 ARPANET

的美国政府机构。

Advanced Research Project Agency Network (ARPANET, 网络名) 由 ARPA 出资建造的分组交换网。

American National Standards Institute (ANSI, 美国国家标准化局) 美国国家性的标准化组织，负责制订美国国内标准。

American Standard Code for Information Interchange (ASCII, 美国信息交换标准码) 一个由 ANSI 开发的字符集，在数据通信里广泛使用。

anonymous FTP (匿名 FTP) 远端用户可以不需要账户或口令访问另一台机器的协议。

anycast address (任播地址) 一个地址，该地址允许分组转发给一组计算机里的任意一台。

applet (小程序) 通常用 Java 写成的计算机程序，用来创建活动 web 文档。

application adaptation layer (AAL, 应用适配层) ATM 协议中一个用于承载用户数据的层。

application layer (应用层) OSI 模型的第七层或 TCP/IP 协议族中的第五层。它提供对网络资源的访问。

application programming interface (API, 应用编程接口) 编程人员在编写客户/服务器程序时需要遵守的一组声明、定义和过程。

area (区域) 在自治系统中的一些网络、主机和路由器的集合。

Asymmetric Digital Subscriber Line (ADSL, 非对称数字用户线) 一种下行数据率高于上行数据率的通信技术。

Asymmetric-key cryptosystem (非对称密钥加密系统) 一种在加密和解密时使用不同密钥的加密系统：加密采用公钥，解密使用私钥。

Asymmetric-key encipherment (非对称密钥保密) 采用非对称密钥加密系统的保密。

Asynchronous Transfer Mode (ATM, 异步传递方式) 一种以高数据率和同等长度分组（信元）为特征的广域网协议；ATM 适合传递文本、音频和视频数据。

ATM adaptation layer (AAL, ATM 适配层) ATM 协议中用于承载用户数据的层。

ATMARP (ATM 地址解析协议) 一种在 ATM 网络上使用的 ARP 协议版本。

authentication (鉴别) 一种检验线路另一端参与者身份的安全服务。

Authentication Header Protocol (鉴别首部协议) 由 IPSec 定义的在网络层提供净荷完整性服务的协议。

Autokey cipher (自动密钥加密方法) 一种流加密方法，该方法中流的子密钥与前一个明文字符相同。第一个子密钥是仅通信双方拥有的秘密值。

Autonomous System (自治系统) 在单独的行政单位管辖下的一组网络和路由器。

Availability (可用性) 信息安全的要素之一，它要求某个组织产生和存储的数据应该能够提供给授权的实体。

Base64 一种在 MIME 里使用的代表非文本数据的编码。

Basic Encoding Rules (BER, 基本编码规则) 把要通过网络的数据进行编码的标准。

Basic Service Set (BSS, 基本服务集) IEEE 802.11 标准定义的无线局域网的构件。

Bellman-Ford algorithm (Bell-Ford 算法) 在距离向量路由选择算法中用来计算路由表的一种算法。

best-effort delivery (尽最大努力交付) IP 使用的不可靠传输机制, 它不保证报文的交付。

big-endian byte order (大数在前字节序) 最高位字节最先存储或传送的一种格式。

biometrics (生物度量) 用于识别一个人的生理学或行为特征的度量。

Bit (比特/位) 以 0 或 1 表示的二进制数(译者注: 在本书中, 通常情况下, 信息传输时翻译为比特, 信息存储处理时翻译为位)。

bit-oriented cipher (面向比特的加密方法) 一种明文、密文和密钥符号都是比特的加密方法。

Block (块) 一组比特作为一个单元处理。

block cipher (块加密方法) 一种加密方法, 在该方法中明文块使用同一个密钥加密。

bootstrap process (引导进程) 计算机的引导过程, 在此过程中需要其本身 IP 地址, 子网掩码, 缺省路由器地址和名字服务器地址。

Bootstrap Protocol (BOOTP, 引导程序协议) 从表(文件)提供配置信息的协议。

Border Gateway Protocol (BGP, 边界网关协议) 基于路径向量路由选择的自治系统间路由选择协议。

bridge (网桥) 工作在 OSI 模型的前两层的网络设备。具有过滤和转发功能。

broadcast address (广播地址) 允许报文向网络的所有结点传输的地址。

broadcast/unknown server (BUS 广播/未知服务器) 连接到 ATM 交换机的可以多播和广播分组的服务器。

browser (浏览器) 可显示万维网文档的应用程序。浏览器通常使用其他因特网服务以访问文档。

buffer (缓冲区) 预留来暂时存放信息的内存。

byte (字节) 8 位一组。

caching (高速缓存) 将信息存储在小而快速的用于保存正在处理数据元素的存储区的过程。

Caesar cipher (恺撒加密方法) 恺撒使用过的一种固定值密钥的加法加密方法。

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA, 具有冲突避免的载波监听多点接入) 一种无线局域网的接入方法, 该方法发现信道空闲时强迫站点发送预约报文以避免碰撞。

Carrier Sense Multiple Access with Collision Detection (CSMA/CD, 具有冲突检测的载波监听多点接入) 一种接入方法, 该方法在传输媒体可用时就发送数据, 出现碰撞时再重传。

cell (信元/蜂窝) 一种小的、固定长度的数据单元; 在蜂窝电话通信中指由一个蜂窝小区交换机服务的地理区域。

Certification Authority (CA, 认证中心) 联邦或州的机构, 负责公共密钥和对应实

体的绑定并颁发证书。

challenge-response authentication (查问-响应鉴别) 一种鉴别方法，使用该方法申请人可证明她知道一个秘密而不用发送该秘密。

character-oriented cipher (面向字符的加密方法) 一种明文、密文和密钥符号都是字符的加密方法。

checksum (检验和) 用于差错检验的字段。该字段通过比特流经反码算术运算后取反获得。

cipher (加密方法) 一种加密或解密算法。

ciphertext (密文) 已加密的数据。

claimant (申请者) 在实体鉴别中，其身份需要被证明的实体。

Clark's solution (Clark 的解决方法) 防止糊涂窗口综合征的一种解决方法。当数据到达时立即发送确认，但是声称的窗口大小为 0，直到接收方有足够的空间放下最大长度的报文段或缓冲区已半空。

classful addressing (分类编址) IPv4 的编址机制，它把 IP 地址空间划分为 5 类：A 类、B 类、C 类、D 类和 E 类。每一类占用了地址空间的一部分。

classless addressing (无分类编址) IP 地址空间不按类划分的一种编址机制。

Classless Inter-Domain Routing (CIDR, 无分类域间路由选择) 使用超网以减少路由表中表项数的一种技术。

client process (客户进程) 在本地运行的应用程序，它向运行在远端的应用程序请求服务。

client-server model (客户-服务器模型) 两个应用程序的交互模型，一端的程序（客户）请求另一端程序（服务器）提供服务。

client-server paradigm (客户-服务器范式) 一种两台计算机通过互联网连接的范式；每一台都必须运行程序，一个提供服务而另一个请求服务。

collision (冲突) 在为每次只能有一个站发送而设计的信道上，有两个站同时发送时就会出现的事件；数据将被损毁。

Colon hexadecimal notation (十六进制冒号记法) 在 IPv6 中使用的一种地址记法，包含 32 个 16 进制数，每 4 个数字由一个冒号分隔。

Common Gateway Interface (CGI, 公共网关接口) HTTP 服务器和可执行程序之间的通信标准之一。CGI 用于产生动态文档。

compression P-box (压缩 P 盒) 有 n 个输入和 m 个输出的 P 盒， $n > m$ 。

concurrent client (并发客户) 一个与其他客户程序同时运行的客户程序。

concurrent server (并发服务器) 可以同时处理多个请求的服务器，多个请求共享该服务器处理时间。

confidentiality (保密性) 定义过程在未鉴别实体面前隐藏信息的安全目标。

configuration file (配置文件) 包含有计算机引导时所需信息的文件。

congestion (拥塞) 网络或互联网中出现过多的通信量，因而引起服务质量的普遍下降。

Connectionless iterative server (无连接迭代服务器) 一次处理一个请求的无连接服

务器。

connectionless service (无连接服务) 没有连接建立或连接终止的数据传送服务。

Connection-oriented concurrent server (面向连接的并发服务器) 能够同时服务多个客户的面向连接服务器。

connection-oriented protocol (面向连接协议) 数据传输与连接建立与结束过程的协议。

connection-oriented service (面向连接服务) 包含连接建立和终止的数据传送服务。

Consultative Committee for International Telegraphy and Telephony (CCITT), 国际电报电话咨询委员会 国际标准化机构, 现在改名为 ITU-T。

contiguous mask (连续掩码) 一种掩码, 由一串 1 后面接着一串 0 组成。

control character (控制字符) 一种字符, 它传递的是关于传输的信息而不是真正的数据。

control traffic (控制通信量) 最高优先级的通信量, 如路由选择报文和管理报文。

Core-Based Tree (CBT, 核心基干树) 使用中心路由器作为树根的组共享多播协议。

cryptanalysis (密码分析) 破解码字的科学和艺术。

cryptographic hash function (加密散列函数) 一个从输入产生短得多的输出的函数。要实用的话, 该函数必须能够防止镜像攻击 (image attack)、反译攻击 (Preimage attack) 和碰撞攻击 (collision attack)。

Cryptographic Message Syntax (CMS 加密报文语法) S/MIME 里使用的语法, 定义了每种内容类型的编码策略。

cryptography (密码学) 把报文转换为安全的和不易受到攻击的一种科学和手段。

Computer Science Network (CSNET, 计算机科学网) 由国家科学基金资助的原用于高校间交流的网络。

Data Encryption Standard (DES, 数据加密标准) 由 NIST 标准化的一种采用多次 Feistel 加密方法的对称密钥块加密方法。

data link layer (数据链路层) OSI 模型的第二层。它负责结点到结点的交付。

datagram (数据报) 在分组交换中独立的数据单元。

datagram socket (数据报套接字) 设计给一个无连接协议如 UDP 使用的数据结构。

decapsulation (拆装) 从报文移去首部和尾部。

decryption (解密) 从已加密的密文恢复成原始的明文。

default mask (默认掩码) 没有划分子网的网络的掩码。

default routing (默认路由选择) 当路由表里无匹配路由时, 所采取的选路方法, 即把所有接收的分组分配给某路由器。

Defense Advanced Research Project Agency (DARPA, 国防部远景研究规划局) 政府机构, 曾在名为 ARPA 时资助了 ARPANET 和因特网。

Defense Data Network (DDN, 国防数据网) 因特网中的军用部分。

denial of service (拒绝服务) 一种仅有的针对可提供性的攻击形式, 它可使系统速度慢下来甚至中断系统。

destination address (目的地址) 数据单元接收者的地址。

dialog control (对话控制) 会话层用来控制对话的技术。

Diffie-Hellman Protocol (**Diffie-Hellman 协议**) 一个不使用 KDC 生成会话密钥的协议。

digest (摘要) 文档的压缩版本。

digital signature (数字签名) 一种安全机制, 发送者可以在报文上电子化签名, 接收者可以验证报文, 以证明该报文确实是发送者签署的。

Digital Signature Standard (DSS) NIST 采用的标准号为 FIPS 186 的数字签名标准。

Digital Subscriber Line (DSL, 数字用户线) 使用现有的通信网络获得高速交付数据、语音、视像和多媒体的一种技术。

Dijkstra's algorithm (**Dijkstra 算法**) 在链路状态路由选择中, 可找出到其他路由器最短路径的一种算法。

direct delivery (直接交付) 这种交付是: 分组最后的终点是和交付者连接在同一个网络上的主机。

Direct Sequence Spread Spectrum (DSSS, 直接序列扩频) 一种无线传输方法, 即发送端要发送的每一位都被称为码片 (chip code) 的比特序列所代替。

Distance Vector Multicast Routing Protocol (DVMRP 距离向量多播路由选择协议) 基于距离向量路由选择结合 IGMP 处理多播选路的协议。

distance vector routing (距离向量路由选择) 一种路由选择方法, 其特点是: 每一个路由器向它的各邻站发送一个表, 表中包含这个路由器所能够到达的网络以及到这些网络的距离。

distributed database (分布式数据库) 信息存储在许多地点。

DNS server (域名服务器) 保存有关于名字空间信息的计算机。

domain (域) 域名空间的子树。

Domain Name System (DNS, 域名系统) 以用户友好的方式将名字转换为 IP 地址的一种 TCP/IP 应用服务。

dotted-decimal notation (点分十进制记法) 便于 IPv4 地址阅读而设计的一种记法, 每个字节转换成其等效的十进制, 并且用点把它和相邻的字节分隔开。

dual stack (双协议栈) 在同一个站上的两种协议 (IPv4 和 IPv6)。

dynamic document (动态文档) 在服务器网点上运行 CGI 程序产生万维网文档。

Dynamic Domain Name System (DDNS 动态域名系统) 动态更新 DNS 主文件的一种方法。

Dynamic Host Configuration Protocol (DHCP, 动态主机配置协议) 对 BOOTP 的扩展, 它动态地指派配置信息。

dynamic mapping (动态映射) 协议用来解析地址的技术。

dynamic port (动态端口) 即临时端口, 它既不是受控的, 也不需要注册, 可以被任何进程使用。

dynamic routing (动态路由选择) 路由表项由协议自动更新的路由选择。

electronic mail (E-mail) (电子邮件) 用电子方式基于邮箱地址来发送报文而不是主

机到主机的直接交换的一种方法。

Electronics Industries Alliance (EIA) (电子工业联盟) 推进与电子学有关的产品的一个机构。它开发了诸如 EIA-232、EIA-449 和 EIA-530 这样的接口标准。

Encapsulating Security Payload (ESP) (封装安全有效载荷) IPSec 定义的一种协议, 它提供保密, 以及完整性和报文鉴别的组合。

encapsulation (封装) 这种技术把来自某个协议的数据单元置入另一个协议的数据单元的数据字段部分中。

encipherment (保密) 参见 encryption。

encryption (加密) 将报文转换为看不懂的形式, 只要不解密就无法看懂。

Entity authentication (实体鉴别) 这种技术设计来使一方能够证实另一方的身份。

ephemeral port (临时端口) 由客户使用的端口号。

error control (差错控制) 在数据传输中对差错的检测和处理。

Ethernet (以太网) 使用 CSMA/CD 接入方法的一种局域网。

Extended Binary Coded Decimal Interchange Code (EBCDIC) (扩充的二-十进制交换码) 由 IBM 开发和使用的一种 8 位字符代码。

Extended Service Set (ESS) (扩展服务集) 由 IEEE 802.11 标准定义的一种无线局域网服务, 它由两个或更多的具有 AP 的 BSS 组成。

extranet (外联网) 一种使用 TCP/IP 协议族的专用网, 它允许外部用户经过授权接入。

Federal Communications Commission (FCC) (联邦通信委员会) 管理无线电、电视和电信的 (美国) 政府机构。

Feistel cipher (Feistel 加密方法) 一种由互反的和非互反的构件共同组成的加密方法。Feistel 密码结合了一个单元 (在文中称为混合器) 中的所有非互反元件, 并在加密算法和解密算法中使用相同的单元。

File Transfer Protocol (FTP) (文件传送协议) 在 TCP/IP 中, 用来在两个地点之间传送文件的应用层协议。

finite state machine (有限状态机) 经历有限数目状态的机器。

firewall (防火墙) 安装在机构的内部网和因特网的其余部分之间的设备 (通常是路由器), 以提供安全保障。

flat namespace (平面地址空间) 将名字映射为地址的一种方法, 它没有层次结构。

flooding (洪泛) 用一个报文使网络饱和。

flow control (流量控制) 一种对帧 (分组或报文) 的流量速率进行控制的技术。

fork 一种 UNIX 函数, 用来创建与父进程的映像完全一致的子进程。

forum (论坛) 对一种特定的新技术进行测试、评价和标准化的机构。

four-way handshake (四向握手) 在客户和服务器之间, 为建立和终止连接而使用的包含 4 步的事件系列。

fragmentation (分片) 为了适应协议的 MTU 而把分组划分为一些更小的单元。

frame (帧) 代表了一个数据块的一组比特。

Frame Relay (帧中继) 为 OSI 模型的前两层而定义的一种分组交换规约。不包括

网络层。差错检测是在端到端的基础上进行的，而不是在每条链路上。

Frame Relay Forum (帧中继论坛) 由 Digital Equipment Corporation、Northern Telecom、Cisco 和 StrataCom 几大公司合建的一个群组，以促进帧中继的推广和实现。

Frequency Hopping Spread Spectrum (FHSS) (跳频扩频) 一种无线传输方法，发送方在一个载频上发送一段短时间，然后跳到另一个载频上发送同样长时间，然后再跳到下一个载频发送同样长时间，依此类推。

full-duplex Ethernet (全双工以太网) 一种以太网的实现，即每个站通过两条互相独立的路径连接到中央集线器。

Full Qualified Domain Name (FQDN) (完整域名) 一种域名，它包含从主机起的标号，一直向后通过每一级到根的顶部。

gateway (网关) 一种连接设备，用来连接两个独立的使用了不同通信协议的网络。

generic domain (通用域) 在域名系统中使用通用后缀的子域。

geographical routing (地理路由选择) 一种路由选择技术，它的整个地址空间根据地理上的大陆块来划分为地址块。

Gigabit Ethernet (吉比特以太网) 具有 1000Mbps 数据率的以太网。

global Internet (全球因特网) 因特网。

grafting (嫁接) 多播报文的再继续。

graph (图) 非等级形式的一种数据结构。

group-shared tree (组共享树) 多播路由选择的特点，在这个系统中的每一个组共享同样的最短路径树。

H.323 一种 ITU 标准，定义了 IP 电话使用的协议族。

half-duplex mode (半双工方式) 一种传输方式，它可以双向通信，但不能同时进行。

handshaking (握手) 建立或终止连接的过程。

hardware address (硬件地址) 数据链路层使用的地址，用来标志一个设备。

hash function (散列函数) 一种算法，它从可变长度的报文生成固定长度的摘要。

hashed message authentication (散列报文鉴别) 使用了报文摘要的鉴别。

hashed message authentication code(HMAC) (散列报文鉴别码) 由 NIST (FIPS 198) 发布的一个标准，用于嵌套的 MAC。

hashing (散列) 一种从可变长度的报文生成固定长度的摘要的密码技术。

header (首部) 加在数据分组开始处的控制信息。

hierarchical name space (层次名字空间) 由若干部分构成的名字空间，其中的每一个后继部分都会变得越来越细。

hierarchical routing (层次路由选择) 一种路由选择技术，即整个地址空间基于特定的准则划分为一些等级。

High bit rate Digital Subscriber Line (HDSL) (高比特率数字用户线) 类似于 T1 线的一种服务，它的操作范围最远可达到 3.6 km。

home address (归属地址) 移动主机在其归属网络上的永久地址。

home agent (归属代理) 通常是一台路由器，它连接在移动主机的归属网络上，能

够接收和发送分组（为移动主机）到外地代理。

home network (归属网络) 移动主机的永久归属网络。

homepage (主页) 在万维网上可用的超文本或超媒体的一个集合, 它是一个机构或个人的主要页面。

hop count (跳计数或跳数) 一条路由沿途的结点数。在路由选择算法中, 跳数是距离的一种度量。

hop limit (跳数限制) 数据报在被丢弃前所经过结点的数量。

host (主机) 在网络中的一个站或结点。

host file (主机文件) 一种文件, 在因特网的规模还不大时, 用来把主机名映射为主机地址。

hostid (主机标识符) IP 地址中用来标志主机的一部分。

Host-specific routing (特定主机路由选择) 一种路由选择技术, 其中在路由表中给定了一个主机的完整 IP 地址。

host-to-host protocol (主机到主机协议) 把分组从一个物理设备交付到另一个物理设备的协议。

hybrid network (混合网络) 一种网络, 它既具有专用内联网, 又能接入到全球因特网。

hypermedia (超媒体) 包含了文本、图片、图形和声音等的信息, 可通过点击链接到其他文档。

hypertext (超文本) 由文本组成的信息, 可通过点击链接到其他文档。

HyperText Markup Language (HTML) (超文本置标语言) 指明万维网文档的内容和格式的计算机语言。它允许包含一些附加代码来定义文本的字体、版式、嵌入的图形和超文本链接。

HyperText Transfer Protocol (HTTP) (超文本传送协议) 读取万维网文档的一种应用服务。

initial vector (IV, 初始向量) 在计算中用于初始化第一次循环的块。

Institute of Electrical and Electronics Engineers (IEEE) (电气和电子工程师学会) 由专业工程师组成的团体, 下设许多起草各种标准的专业学会。

integrity (完整性) 一种安全服务, 被设计用于保护数据不被修改、插入、删除和重放。

interactive traffic (交互式通信量) 有必要与用户交互的一种通信量。

interface (接口) 两个设备之间的界面。它也指机械的、电气的和功能特征上的连接。在网络编程中, 它是使得上层能够利用下层服务的一组过程。

International Standards Organization (ISO) (国际标准化组织) 针对各种各样的主题定义和开发标准的世界性组织。

International Telecommunications Union–Telecommunication Standardization Sector (ITU-T) (国际电信联盟-电信标准部) 原先名为 CCITT 的标准化组织。

internet (互联网) 网络的集合, 这些网络由一些网际互连设备 (如路由器或网关) 互连起来。

Internet (因特网) 使用 TCP/IP 协议族的全球互联网。

Internet address (因特网地址) 32 位或 128 位的网络层地址，用于唯一地定义连接在使用了 TCP/IP 协议族的互联网上的主机。

Internet Architecture Board (IAB) (因特网体系结构研究委员会) ISOC 的技术咨询机构，管理 TCP/IP 协议族的继续开发。

Internet Assigned Number Authority (IANA) (因特网赋号管理局) 由美国政府支持的一个团体，在 1998 年 10 月以前曾负责管理因特网的域名和地址。

Internet Control Message Protocol (ICMP) (网际控制报文协议) TCP/IP 协议族中用来处理差错和控制报文的一个协议。目前使用的两个版本是 ICMPv4 和 ICMPv6。

Internet Corporation for Assigned Names and Numbers (ICANN) (因特网名字与号码指派公司) 由国际董事会管理的私营非赢利公司，担任原来 IANA 的工作。

Internet draft (因特网草案) 正在进行加工处理的因特网文档，它没有正式的状态，并且只有 6 个月的生存期。

Internet Engineering Steering Group (IESG) (因特网工程指导小组) 管理 IETF 的活动的机构。

Internet Engineering Task Force (IETF) (因特网工程部) 设计和开发 TCP/IP 协议族和因特网的工作组。

Internet Group Management Protocol (IGMP) (网际组管理协议) TCP/IP 协议族中处理多播的协议。

Internet Key Exchange (IKE) (因特网密钥交换) 设计来创建 IPSec 中的安全关联的一种协议。

Internet Mail Access Protocol (IMAP) (网际邮件存取协议) 用来从电子邮件服务器上提取电子邮件报文的一种很复杂但功能强大的协议。

Internet Network Information Center (INTERNIC) (因特网网络信息中心) 负责收集和发布有关 TCP/IP 协议族的信息的机构。

Internet Protocol (IP) (网际协议) TCP/IP 协议族中的网络层协议，管理通过分组交换网的无连接传输。目前在用的两个版本是 IPv4 和 IPv6。

Internet Protocol, next generation (IPng) (下一代网际协议) 网际协议第六版的另一种称呼。

Internet Research Task Force (IRTF) (因特网研究部) 集中讨论关于因特网的长期研究课题的工作组论坛。

Internet Security Association and Key Management Protocol (ISAKMP) (因特网安全关联和密钥管理协议) 由美国国家安全署 (NSA) 设计的一个协议，实现了 IKE 中定义的密钥交换。

Internet Service Provider (ISP) (因特网服务提供者) 通常是指提供因特网服务的公司。

Internet Society (ISOC) (因特网协会) 宣传因特网的一个非赢利机构。

Internet standard (因特网标准) 已经彻底测试过的规约，是所有在因特网上工作的人需要使用的，同时也是他们必须遵守的。因特网标准是必须遵循的正式规则。

internetworking (网际互连) 使用如路由器或网关这样的网际互连设备把若干网络连接起来。

intranet (内联网) 使用 TCP/IP 协议族的专用网。

inverse cipher (逆向加密方法) 解密算法。

IP datagram (IP 数据报) 网际协议的数据单元。

IP Security (IPSec) (IP 安全) 由 IETF (因特网工程部) 设计的一组协议, 用来提供在因特网上传送分组的安全。

Java 用来创建动态万维网文档的编程语言。

jitter (抖动) 实时通信中的一种现象, 是由接收端相邻分组之间的间隙产生的。

Karn's algorithm (Karn 算法) 一种算法, 它在计算往返延时时不包括重传的报文段。

key (密钥) 一组值, 在加密和解密算法中要对其进行运算。

key ring (密钥环) 在 PGP 中使用的一组公钥或私钥。

Key Distribution Center (KDC) (密钥分配中心) 在双方之间建立共享密钥的被信任的第三方。

Link Control Protocol (LCP) (链路控制协议) 负责建立、维持、配置和终止链路的 PPP 协议。

link local address (本地链路地址) 一种 IPv6 地址, 当局域网想要使用因特网协议, 但又出于安全考虑而不愿连接到因特网时所使用的地址。

link state database (链路状态数据库) 链路状态路由选择中的数据库, 它由 LSP 信息构成, 并对所有路由器都是相同的。

little-endian byte order (小数在前字节序) 首先从最低字节开始保存或传输的一种格式。

local address (本地地址) 电子邮件地址中的一部分, 定义了一种特殊的文件 (称为用户邮箱) 的名称, 其中保存的是为该用户而接收的所有邮件, 等待用户代理人读取。

Local Area Network (LAN) (局域网) 把单个建筑物内或几个彼此靠近的建筑内的一些设备连接起来的网络。

local client program (本地客户程序) 在本地运行的程序, 它向远程服务器程序请求服务。

local host (本地主机) 用户物理上正在使用的计算机。

local login (本地登录) 使用终端直接和计算机连接。

local loop (本地环路) 把用户连接到电话交换局的线路。

logical address (逻辑地址) 在网络层定义的地址。

Logical IP Subnet (LIS) (逻辑 IP 子网) ATM 网络中的一组结点, 其中的连接是逻辑的而不是物理的。

logical tunnel (逻辑隧道) 把多播分组封装到单播分组中, 以便在非多播路由器上进行多播路由选择。

loopback address (环回地址) 主机用来测试内部软件的地址。

magic cookie (魔块) 在 DHCP 中, 这是在 IP 地址格式中出现的值为 99.130.83.99

的数，表示了选项的存在。

mail access protocol (邮件接入协议) 由远程用户代理使用的一种协议，以接入邮箱并获取邮件。

mail transfer agent (MTA) (邮件传送代理) SMTP 中的构件，以通过因特网传送邮件。

Management Information Base (MIB) (管理信息库) SNMP 使用的数据库，它保存了管理网络所必需的信息。

mapped address (映射地址) 一种 IPv6 地址，当已经迁移到 IPv6 的计算机要向仍然使用 IPv4 的计算机发送分组时，就要使用这种地址。

mask (掩码) 对于 IPv4，是 32 位的二进制数，用它和这个地址块中地址进行 AND 操作时，就得出这个地址块的第一个地址（这个网络的地址）。

masking (掩码运算) 从 IP 地址中提取物理网络地址的过程。

master secret (主密) 在 SSL 中，根据前主密而创建的 48 个字节的密钥。

maturity level (成熟等级) RFC 要经过的几个阶段。

Maximum Transfer Unit (MTU) (最大传送单元) 一个特定网络能够处理的最大数据单元长度。

message authentication (报文鉴别) 提供无连接通信中对发送方的身份鉴别。

message authentication code (MAC) (报文鉴别码) 包含了双方之间的密钥的 MDC。

message digest (报文摘要) 一个固定长度的字符串，是通过对报文应用散列函数后得到的。

Message Digest (MD) (报文摘要) 由 Ron Rivest 设计的一组若干个散列算法，称为 MD2、MD4 和 MD5。

metric (度量) 被指派的，通过一个网络所需的代价。

metropolitan area network (MAN) (城域网) 在地理面积上占据一个城市大小的网络。

Military Network (MILNET) (军用网) 为军方使用的网络，原先是 ARPANET 的一部分。

mobile host (移动主机) 可以从一个网络移动到另一个网络的主机。

modern block cipher (现代分组加密方法) 一种对称密钥密码，每 n 位的明文块通过相同的密钥被加密成 n 位的密文块。

modern stream cipher (现代流加密方法) 一种对称密钥密码，加密和解密都是一次 r 位地进行的，使用了密钥流。

monoalphabetic cipher (单态字符加密方法) 一种替代密码，明文中的符号在密文中总是被改变为相同的另一个符号，而不考虑它在文中所处的位置。

monoalphabetic substitution cipher (单态字符替代加密方法) 一种密码，它的密钥就是每个明文字符与相应的密文字符之间的映射关系。

multicast address (多播地址) 用于多播的地址。

multicast backbone (MBONE) (多播主干网) 一组通过使用隧道技术支持多播的互联网路由器的集合。

Multicast Open Short Path First (MOSPF) (多播开放最短路径优先) 一种多播协议, 它使用多播链路状态路由选择链路创建基于源点的最小代价树。

multicast router (多播路由器) 一种路由器, 它使用和每个路由器接口有关的忠实成员表来分发多播分组。

multicast routing (多播路由选择) 把多播分组传送到它的终点。

multicasting (多播) 可以把单个分组的若干个副本发送给选定的一组接收者的一种传输方法。

multihommed device (多归属设备) 连接到超过一个网络的设备。

multimedia traffic (多媒体通信量) 由数据、视频和音频组成的通信量。

multiple unicasting (多个单播) 从一个源点发送报文的多个副本, 每一个副本具有不同的单播地址。

multiplexing (复用) 为了在一条数据链路上传输, 把多个信源的信号进行合并的过程。

multiplicative decrease (乘法减小) 一种拥塞避免技术, 即门限设置为上次拥塞窗口值的一半, 然后拥塞窗口又从 1 开始。

Multipurpose Internet Mail Extension (MIME) (多用途因特网邮件扩充) SMTP 的一种补充, 允许通过 SMTP 发送非 ASCII 码的数据。

Multistation access unit (MAU) (多站接入单元) 在令牌环中, 容纳了多个独立的自动交换机的设备。

Nagle's algorithm (Nagle 算法) 一种算法, 试图在发送端阻止产生糊涂窗口综合征。无论是产生数据的速率, 还是网速都要加以考虑。

National Institute of Standard and Technology (NIST) (美国国家标准和技术局) 开发标准和技术的美国政府机构。

National Science Foundation (NSF) (美国国家科学基金会) 负责为因特网提供资金的(美国)政府机构。

National Science Foundation Network (NSFNET) (国家科学基金会网络) 由 NSF 提供资金的主干网。

National Security Agency (NSA) (美国国家安全局) 美国国家情报收集安全管理机构。

national service provider (NSP) (国家服务提供者) 由指定公司建立和维护的主要网。

netid (网络标识符) IP 地址中标志网络的那部分。

Network Access Point (NAP) (网络接入点) 把一些主干网连接起来一种复杂的交换站。

network address (网络地址) 用来向因特网上的其他地方标志该网络的地址; 它是地址块中的第一个地址。

Network Address Translation (NAT) (网络地址转换) 一种技术, 允许专用网使用一组专用地址进行内部通信, 同时使用一组全球因特网地址进行外部通信。

network byte order (网络字节序) 与大数在前字节序相同。

Network Control Protocol (NCP) (网络控制协议) 在 PPP 中的一组控制协议, 它允许把来自网络层的数据进行封装。

Network File System (NFS) (网络文件系统) 一种 TCP/IP 应用协议, 它允许用户存取和操纵远程文件系统, 就像这些文件系统在本地一样。NFS 使用远程过程调用协议。

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

Network Interface Card (NIC) (网络接口卡) 包含了可以使该站连接到网络的电路的一种电子设备, 可以在站点内部, 也可以在站点外部。

network layer (网络层) OSI 模型 (或 TCP/IP 协议族) 中的第三层, 负责将分组交付到最后的终点。

network virtual terminal (NVT) (网络虚拟终端) 允许远程登录的 TCP/IP 应用层协议。

network-specific routing (特定网络路由选择) 这种路由选择是网络上的所有主机共享路由表中的一个表项。

Network-to-Network Interface (NNI) (网络到网络接口) 在 ATM 中两个网络之间的接口。

next-hop address (下一跳地址) 分组接下来将交付给的第一个路由器的地址。

next-hop routing (下一跳路由选择) 一种路由选择方法, 即在路由表中只列出下一跳地址, 而不是这个分组必须停留的所有路由器地址的完整清单。

noise (噪声) 可以被传输媒体吸收的随机电信号, 其结果是使数据受到损伤或产生失真。

noncontiguous mask (不连续掩码) 这种掩码由一串比特组成, 但不是一串 1 后面紧跟着一串 0, 而是 0 和 1 的混合。

non-Feistel cipher (非 Feistel 加密方法) 只使用可互换的组件的加密方法。

nonpersistent connection (非持续连接) 一种连接, 即每一次请求/响应需要建立一条 TCP 连接。

nonrepudiation (不可否认) 一种安全状况, 即接收者必须能够证明所收到的报文是来自特定的发送者。

Oaklay 由 Hilarie Orman 开发的一种密钥交换协议, 是对 Diffie-Hellman 方法的改进。

one's complement (二进制反码) 二进制数字的一种表示方法, 通过把一个数的所有位都反过来就得出这个数的二进制反码。

one-time pad (一次填充) 由 Vernam 发明的一种密码, 其中密钥是随机的符号序列, 且与明文一样长。

one-time password (一次口令) 只能使用一次的口令。

Open Shortest Path First (OSPF) (开放最短路径优先) 基于链路状态路由选择的一种内部路由选择协议。

open systems interconnection (OSI) (开放系统互连) 由 ISO 定义的数据通信的七层模型。

out-of-band signaling (带外信令) 控制数据和用户数据在不同信道上传输的一种信令。

方法。

packet (分组) 数据单元的同义词, 经常在网络层使用。

Packet Internet Groper (PING) (分组因特网搜寻器) 一种应用程序, 它使用 ICMP 回送请求和回答来确定某个终点的可达性。

packet-filter firewall (分组过滤器防火墙) 一种防火墙, 它根据网络层和运输层首部的信息转发或阻拦分组。

page (页面) 用在万维网上的超文本或超媒体的单元。

Partially Qualified Domain Name (PQDN) (不完整域名) 一种域名, 它不包括从主机到根结点的所有级。

passive attack (被动攻击) 一种攻击类型, 攻击者的目标是获取信息。这种攻击不会修改数据或损坏系统。

passive open (被动打开) 服务器的一种状态, 当它一直等待着从客户来的请求时就处于这种状态。

password-based authentication (基于口令的鉴别) 最简单、最古老的实体鉴别方法, 通过一个口令来证实被验者的身份。

Path MTU Discovery technique (路径 MTU 发现技术) IPv6 用来发现被路径上任何一个网络都支持的最小 MTU 的一种方法。

path vector routing (路径向量路由选择) 一种路由选择方法, 是 BGP 的基础。在此方法中必须明确列出分组必须经过的自治系统。

P-box (P 盒) 现代分组密码中的一个构件, 用于置换位。

peer-to-peer process (对等进程) 发送的机器和接收的机器上都有的一个进程, 并在给定的层次上进行通信。

Peer-to-peer paradigm (P2P 范式) 两个对等的计算机能够相互通信以交换服务的一种范式。

persistent connection (持续连接) 一种连接, 即服务器在发送了一个响应后, 该连接继续处于打开状态, 以便接受更多的请求。

physical address (物理地址) 设备在数据链路层使用的地址 (MAC 地址)。

physical layer (物理层) OSI 模型的第一层, 负责传输媒体在机械和电气方面的规约。

physical topology (物理拓扑) 设备在网络中连接的方式。

piggybacking (捎带) 在数据帧中包括了确认。

plaintext (明文) 加密之前或解密之后的报文。

playback buffer (重放缓存) 存储数据的缓存, 直到这些数据准备好被播放为止。

point-to-point link (点对点链路) 两个设备之间的专用传输线路。

Point-to-Point Protocol (PPP) (点对点协议) 在串行线路上传送数据的协议。

poison reverse (毒性逆转) 分割范围的一种变体。在这种方法中, 路由器用接收到的信息来更新路由表, 然后再把这些信息通过所有接口发送出去。但是从某个接口接收的信息在通过同一个接口发送出去时, 要把相应表项的度量设置为 16。

policy routing (策略路由选择) 路径向量路由选择的一个特点, 即路由表的基础是

网络管理员设定的规则，而不是某种度量。

Polyalphabetic cipher (多态字符加密方法) 一种加密方法，此时同一个字符的每一次出现可能都有不同的替代。

port address (端口地址) 在 TCP/IP 协议中用来标志进程的一个整数（参见端口号）。

port number (端口号) 定义了某主机上正在运行的一个进程的整数。

Post Office Protocol (POP) (邮局协议) 一种流行而简单的 SMTP 邮件读取协议。

prefix (前缀) 对一个网络来说，这是地址范围的共同部分的另一种叫法（类似 netid）。

preimage resistance (前映像防御) 这是人们希望加密散列函数所具有的一种属性，此时若给出某个摘要，敌方要找到另一个具有相同摘要的报文将会是非常困难的事。

pre-master secret (前主密钥) 使用 SSL 时，在计算主密钥之前，客户和服务器之间交换的密钥。

presentation layer (表示层) OSI 模型的第六层，负责转换、加密、鉴别和数据压缩。

Pretty Good Privacy (PGP) (相当好的保密) 由 Phil Zimmermann 创建的一种协议，为电子邮件提供保密性、完整性和鉴别。

privacy (保密) 安全的一个方面，即报文只能让预期的接收者看懂。

private key (私钥) 在不对称密钥加密系统中，这个密钥用于解密。在数字签名中，这个密钥用于签名。

private network (专用网) 与因特网相隔离的网络。

process (进程) 正在运行的应用程序。

process-to-process communication (进程到进程通信) 两个正在运行的应用程序之间的通信。

promiscuous ARP (proxy ARP) (代理 ARP) 能够产生划分子网效果的技术。用一个设备为多个主机回答 ARP 请求。

protocol (协议) 通信的规则。

Protocol Independent Multicast (PIM) (协议无关多播) 具有两个成员（PIM-DM 和 PIM-SM）的多播协议族，这两个协议都依赖于单播协议。

Protocol Independent Multicast, Dense Mode (PIM-DM) (协议无关多播，密集方式) 一种基于源点的路由选择协议，它使用 RPF 和剪枝/嫁接策略来处理多播。

Protocol Independent Multicast, Sparse Mode (PIM-SM) (协议无关多播，稀疏方式) 一种组共享路由选择协议，它与 CBT 相似，并且有一个汇集点作为树的源点。

protocol suite (协议族) 为复杂的通信系统定义的协议栈或一族协议。

proxy firewall (代理防火墙) 一种能够基于报文本身提供的信息来过滤报文的系统（在应用层）。

proxy server (代理服务器) 保留对最近的请求响应的副本的计算机。

pruning (剪枝) 停止从一个接口发送多播报文。

pseudoheader (伪首部) 从 IP 首部得到一些信息，仅为了计算 UDP 和 TCP 分组的检验和。

public key (公钥) 在不对称密钥加密系统中，这个密钥用于加密。在数字签名中，这个密钥用于验证。

public key encryption (公钥加密方法) 一种加密方法，它的基础是不可逆的加密算法。这种方法要使用两类密钥：公钥面向大众公开，私钥只有接收方知道。

public-key infrastructure (PKI) (公钥基础结构) 根据 X.509 来创建和分发证书的一种模型。

pure ATM LAN (纯 ATM 局域网) 一种局域网，用 ATM 交换机来连接局域网中的站点，并且这些站点以同样的方式连接到以太网交换机。

pushing data (推送数据) 发送站点上的应用程序不需要等待窗口被填满的一种技术。它可以立即创建报文段并发送。

quality of service (QoS) (服务质量) 丢失、延迟和容量等指标综合的测量。

queue (队列) 等待的列表。

quoted-printable (引用可打印) 一种编码机制，用于大部分数据是 ASCII 字符，而只有小部分是非 ASCII 字符时。如果字符是 ASCII 的，就直接发送。如果字符是非 ASCII 的，就要发送三个字符：第一个字符是等号 (=)，后面两个字符是该字节的十六进制表示。

Rate adaptive Asymmetrical Digital Subscriber Line (RADSL) (速率自适应非对称数字用户线) 一种基于 DSL 的技术，它的特点是根据不同的通信类型可以有不同的数据率。

raw socket (原始套接字) 为某些既不使用流套接字，也不使用数据报套接字，而是直接使用 IP 服务的协议设计的结构。

Read-Only Memory (ROM) (只读存储器) 内容不能改变的永久性存储器。

real-time multimedia traffic (实时多媒体通信量) 包含了数据、音频和视频的通信量，并且它们的产生和使用是同时进行的。

real-time traffic (实时通信量) 一种产生和使用同时进行的通信量。

Real-time Transport Control Protocol (RTCP) (实时传输控制协议) 和 RTP 一起使用的伴侣协议，它的报文对数据流和数据质量进行控制，并允许接收者向源点（或多个源点）发送反馈信息。

Real-time Transport Protocol (RTP) (实时传输协议) 用于实时通信量的一种协议，它与 UDP 一起使用。

regional ISP (地区 ISP) 一种小型的 ISP，连接一个或多个 NSP。

registered port (登记端口) IANA 不指派和控制的端口号，范围从 1024~49151。

registration (登记) 远程主机和移动主机通信的一个阶段，在这个阶段，移动主机把自己的信息告诉外地代理。

remote host (远程主机) 当用户实际上坐在另一台计算机前时，希望接入的一台主机。

remote login (rlogin) (远程注册) 从连接在本地计算机的终端向远程计算机登录的过程。

rendezvous router (汇集路由器) 作为每一个多播组的核心或中心的路由器。它将成为树的根。

rendezvous-point tree (汇集点树) 一种组共享树方法，其中每一个组有一个树。

repeater (转发器) 一种设备，它用信号再生的方法扩展了信号的传送距离。

repudiation (否认) 一种对信息完整性的攻击，它是通信中的任一方（发送方或接收方）都可能发起的攻击。

Request for Comment (RFC) (请求评论) 涉及到因特网的某个问题的因特网正式文档。

requirement level (需求等级) RFC 的五个需求等级之一。

reserved address (保留地址) 因特网管理机构为专用网的使用而特别留出的一些 IP 地址，或者是具有保留前缀的 IPv6 地址。

resolver (解析程序) 当主机需要把地址映射为名字，或把名字映射为地址时，就要使用这个 DNS 客户程序。

retransmission timer (重传计时器) 控制等待确认某个报文段的时间的计时器。

Reverse Address Resolution Protocol (RARP) (逆地址解析协议) 使主机根据它的物理地址找出相应的 IP 地址的一种 TCP/IP 协议。

reverse path broadcasting (RPB) (反向路径广播) 路由器只转发经过最短路径从源点到达该路由器的分组的一种技术。

reverse path forwarding (RPF) (反向路径转发) 路由器只转发经过最短路径从源点到达该路由器的分组的一种技术。

reverse path multicasting (RPM) (反向路径多播) 为 RPB 增加了剪枝和嫁接的一种技术，以建立能够支持动态改变成员关系的多播最短路径树。

ring topology (环状拓扑) 所有设备连接成环的一种拓扑。环上的每个设备都要接收前面一个设备的数据，再生这些数据，并将它们转发给下一个设备。

rlogin 由 BSD UNIX 设计的远程登录应用。

round (一周) 在循环的分组加密办法中的每个循环部分。

Round-Trip Time (RTT) (往返时间) 数据报从源点传送到终点然后再回来总共需要的时间。

router (路由器) 工作在 OSI 模型前三层的网际互连设备。路由器连接了两个或更多个网络，并从这个网络到那个网络地转发分组。

routing (路由选择) 路由器执行的处理过程，即找出数据报的下一跳。

Routing Information Protocol (RIP) (路由信息协议) 基于距离向量路由选择算法的路由选择协议。

routing table (路由表) 含有路由器转发分组所需信息的表。这些信息可能包括网络地址、代价、下一跳的地址等等。

RSA cryptosystem (RSA 加密系统) 一种最常用的公钥加密算法，由 Rivest, Shamir 和 Adleman 提出。

RSA signature scheme (RSA 签名机制) 一种基于 RSA 加密系统的数字签名机制，但是它的私钥和公钥的作用与 RSA 加密系统不同。发送方用自己的私钥来签署文档，而接收方使用发送方的公钥来验证它。

salting (加盐) 一种用来改善基于口令的鉴别的方法，即用一个随机字符串（称为 salt）与该口令连接起来。

S-box (S 盒) 分组加密方法中的一个构件，它把输入中的比特用新的比特替代来产

生输出。

secret-key encryption (秘密密钥加密) 一种加密算法，其加密和解密使用相同的密钥，即发送方和接收方都有相同的密钥。

Secure Hash Algorithm (SHA) (安全散列算法) 由 NIST 开发的一系统散列函数标准，以 FIPS 180 发布，大部分基于 MD5。

Secure Key Exchange Mechanism (SKEME) (安全密钥交换机制) 由 Hugo Kraweyz设计的一个协议，用于基于公钥加密方法的实体鉴别中的密钥交换。

secure shell (SSH) (安全外壳) 提供了安全的客户-服务器程序。

secure socket layer (SSL) (安全套接层) 为应用层产生的数据提供安全以及压缩服务而设计的协议。

Secure/Multipurpose Internet Mail Extension (S/MIME) (安全/多用途因特网邮件扩展) 一种对 MIME 的增强，为电子邮件提供安全。

security (安全) 保护网络不受未经授权的接入、病毒以及事故等的损害。

Security Association (SA) (安全关联) 在 IPSec 中，两个主机之间的逻辑关系。

Security Association Database (SAD) (安全关联数据库) 一张二维表，其中的每一行定义了一个安全关联。

security attacks (安全攻击) 威胁系统的安全目标的攻击。

security goals (安全目标) 信息安全的三个目标是：保密性、完整性和有效性。

Security Policy (SP) (安全策略) 是 IPSec 中的一组事先定义的安全需求，应用于一个分组将要发送或到达时。

Security Policy Database (SPD) (安全策略数据库) 安全策略的数据库。

security services (安全服务) 与安全目标和攻击对应的五种服务是：数据保密性、数据完整性、鉴别、不可否认以及接入控制。

segment (报文段) TCP 层的分组。

segmentation (分段) 把报文划分为多个分组，通常在运输层完成。

semantics (语义) 每一部分数据的含义。

sequence number (序号) 用来表明一个帧或分组在报文中的位置的编号。

session (会话) 在 SSL 中客户和服务器之间的一个关联。在会话建立后，双方就掌握了一些共用的信息，如会话 ID，彼此的鉴别证书（如有必要），压缩方法（如需要），加密方法集，以及用于产生报文鉴别加密密钥的主密钥。

session key (会话密钥) 双方之间的一次性密钥。

session layer (会话层) OSI 模型的第五层，负责建立、管理和终止两个端用户之间的逻辑连接。

shared secret key (共享密钥) 用于对称密钥加密术中的密钥。

shift cipher (移位加密方法) 一种加法加密方法，其中的密钥定义了向字符集的后端移动几个字符。

shortest path (最短路径) 从源点到终点的最佳路径。

silly window syndrome (糊涂窗口综合征) 当接收方通告的是一个小窗口，而发送方也发送小报文段的一种情况。

Simple Mail Transfer Protocol (SMTP) (简单邮件传送协议) 定义因特网上电子邮件服务的 TCP/IP 协议。

Simple Network Management Protocol (SNMP) (简单网络管理协议) 指明因特网的管理过程的 TCP/IP 协议。

simplex mode (单工方式) 通信单向进行的一种传输方式。

site local address (本地场所地址) 一种 IPv6 地址, 使用在这样的情况: 一个场所有几个网络使用因特网协议, 但由于安全原因都没有连接在因特网上。

slash notation (斜线记法) 指出掩码中 1 的个数的一种简写方法。

sliding window protocol (滑动窗口协议) 允许在收到确认之前可以发送好几个数据单元的一种协议。

slow convergence (慢收敛) RIP 缺点的表现, 即当互联网中的某处发生变化时, 要传播到互联网的其他地方则很慢。

slow start (慢开始) 一种拥塞控制方法, 使拥塞窗口值在开始时按指数规律增长。

Snooping (窃取) 对保密信息的未授权的访问。是对信息安全中保密目标的一种攻击。

socket (套接字) 一个进程的端点。通信时需要两个套接字。

socket address (套接字地址) 保存了一个 IP 地址和一个端口号的结构。

socket interface (套接字接口) 基于 UNIX 的 API, 定义了一组系统调用 (过程) 集合, 是对 UNIX 中用于访问文件的系统调用的扩展。

something inherent (一些与生俱来的东西) 这是申请者天生的一些特点。如传统签名、指纹、话音、面部特点、视网膜以及笔迹等, 用于实体鉴别。

something known (一些记在脑子里的东西) 这是一些只有申请者知道并能被验证者检查的秘密。

something possessed (一些拿在手里的东西) 这是一些能够证实申请者身份的物件。例如护照、驾驶证、身份证件、信用卡或智能卡。

sorcerer's apprentice bug (巫士徒弟的错误) TFTP 的一个问题, 即分组并没有丢失, 只不过被延迟了, 于是每一个后继的分组就发送两次, 而每一个后继的确认也收到两次。

source quench (源点抑制) ICMP 用于流量控制的一种方法, 即在发生拥塞时, 通知源点减慢或停止发送数据报。

source-based tree (源点基准树) 多播协议使用的一种多播树, 即对源点和组的每一个组合就构成一个树。

source-to-destination delivery (源点到终点交付) 报文从初始的发送者到预期的接收者的传输。

spanning tree (生成树) 一种树, 源点充当它的根, 组成员充当它的叶, 即连接了所有结点的树。

split horizon (分割范围) 改进 RIP 稳定性的一种方法, 路由器有选择地挑选更新信息转发出去的接口。

split operation (分裂操作) 分组加密方法中的一种操作, 它把分组从中一分为二, 从而产生两个等长的分组。

spoofing (伪装) 参见 masquerading。

spread spectrum (扩频) 一种无线传输技术, 它需要的带宽是原始带宽的好几倍。

standard (标准) 所有人都同意的基准或模型。

star topology (星形拓扑) 所有的站都连接到中心设备 (集线器) 的一种拓扑。

State (状态) AES 中间阶段的一种数据单元, 由 16 字节的矩阵组成。

state transition diagram (状态转换图) 用来说明有限状态机的状态的图。

static document (静态文档) 万维网上在服务器中创建和存储的固定内容文档。

stationary host (固定主机) 一直连接在某个网络上的主机。

station-to-station protocol (站到站协议) 根据 Diffie-Hellman 协议产生会话密钥的一种方法, 它用公钥证书来防止中间有人攻击。

steganography (隐密术) 一种安全技术, 通过其他东西的遮盖来隐藏报文本身。

steiner tree (steiner 树) 一种找出多播树的方式, 其中的最优树就是所有链路代价总和为最小的树。

stop and wait (停止等待) 一种流量控制方式, 此时对每个数据单元都必须确认之后才能发送下一个数据单元。

stop and wait ARQ (停止等待 ARQ) 使用了停止等待流量控制的差错控制协议。

straight P-Boxes (直 P 盒) 有 n 个输入和 n 个输出的 P 盒。

stream cipher (流加密方法) 一种加密方法, 它的加密和解密都是一次一个符号地 (如一个字符或一个比特) 进行。

stream socket (流套接字) 一种结构, 被设计来与面向连接的协议 (如 TCP) 一起使用。

Structure of Management Information (SMI) (管理信息结构) 在 SNMP 中, 用于网络管理的一个构件。

stub link (残桩链路) 仅和一个路由器连接的网络。

subnet address (子网地址) 子网的网络地址。

subnet mask (子网掩码) 子网的掩码。

subnetting (划分子网) 把一个网络划分为几个较小的单元。

subnetwork (子网) 网络的一部分。

substitution cipher (替代加密方法) 用另一个符号来替代一个符号的加密方法。

suffix (后缀) 对于网络来说, 就是地址的变化部分 (和 host-id 相似)。在 DNS 中, 后缀是一个串, 被机构用来定义自己的主机或资源。

supernet (超网) 由两个或更多个较小的网络构成的网络。

supernet mask (超网掩码) 超网的掩码。

supernetting (构成超网) 把几个 C 类地址块合并, 创建更大的地址范围。

switch (交换机) 把多条通信线路连接在一起的设备。

switched Ethernet (交换以太网) 用交换机代替集线器的以太网, 它可以把帧直接传输到终点。

Switched Virtual Circuit (SVC) (交换虚电路) 一种虚电路传输方法, 即仅在交换持续时间内, 虚电路才被创建和存在。

Symmetric Digital Subscriber Line (SDSL) (对称数字用户线) 基于 DSL 的技术，和 HDSL 相似，但仅使用单个的双绞线电缆。

symmetric-key cryptosystem (对称密钥加密系统) 加密和解密都使用相同密钥的加密系统，有时也称为密钥加密系统。

Symmetric-key encipherment (对称密钥保密) 使用对称密钥加密系统的保密。

Synchronous Digital Hierarchy (SDH) (同步数字系列) 与 SONET 等效的 ITU-T 标准。

Synchronous Optical Network (SONET) (同步光纤网) ANSI 开发的使用光纤技术传送高速数据的标准。它可用来传送文本、音频和视频。

syntax (语法) 数据的结构或格式，也就是指数据呈现的顺序。

TCP/IP protocol suite (TCP/IP 协议族) 用在互联网中的一组层次化的协议。

Terminal Network (TELNET) 允许远程登录的通用客户-服务器程序。

three-way handshake (三向握手) 连接建立或终止的事件序列，包括请求，然后是对请求的确认，再后是对确认的证实。

ticket (签条) 一个加密的报文，它是要给实体 B 的，但却发送给实体 A，再由实体 A 交付给实体 B。

T-lines (T 线) 一种等级化的数字线路，被设计来运载数字形式的语音和其他信号。

topology (拓扑) 包含了各个设备的物理布局的网络结构。

traffic analysis (通信量分析) 对保密性的一种攻击类型，攻击者通过监视在线通信量来获取一些信息。

trailer (尾部) 附着在数据单元后面的控制信息。

transient link (穿越链路) 有多个路由器连接在上面的网络。

Transmission Control Protocol (TCP) (传输控制协议) TCP/IP 协议族中的运输层协议。

Transmission Control Protocol /Internetworking Protocol (TCP/IP) (传输控制协议/网际协议) 一个五层的协议族，定义了通过因特网的传输交换。

transport layer (运输层) OSI 模型的第四层，负责可靠的端到端的交付以及差错恢复。

Transport Layer Security (TLS) (运输层安全) 一种在运输层的安全协议，它提供在万维网上的安全。它是 IETF 版本的 SSL 协议。

transport mode (运输方式) 对 TCP 报文段或 UDP 用户数据报加密时，先加密，然后再封装成 IPv6 分组。

transpositional cipher (置换加密方法) 通过改变明文中符号的位置来产生密文的一种加密方法。

tree (树) 一种分层的数据结构，其中树上的每一个结点只有一个父结点，但可以有零个或多个子结点。

triple DES (3DES) (三重 DES) 一种加密方法，在加密时使用了三次 DES 加密方法的实例，在解密时同样要使用三次逆 DES 加密方法的实例。

Trivial File Transfer Protocol (TFTP) (简单文件传送协议) 一种不可靠的 TCP/IP

文件传送协议，它不需要客户和服务器之间的复杂交互。

tunnel mode (隧道方式) IPSec 的一种方式，可以保护整个 IP 分组。它先获取一个 IP 分组，包括其首部，然后对整个分组应用 IPSec 安全方式，再添加一个新首部。

tunneling (隧道技术) 在多播中，把多播分组封装成一个单播分组，然后再通过网络发送出去的一种处理过程。

unattended data traffic (不关注的数据通信量) 用户并不等待（关注）其数据的一种通信量。

unicast address (单播地址) 定义单个终点的地址。

unicasting (单播) 发送的分组只有一个终点。

Uniform Resource Locator (URL) (统一资源定位符) 标志万维网页面的字符串（地址）。

urgent data (紧急数据) 在 TCP/IP 中必须尽可能快地向应用程序交付的数据。

urgent pointer (紧急指针) 指向紧急数据和正常数据的边界的指针。

User Agent (UA) (用户代理) 一种 SMTP 构件，它准备报文，创建信封，并把报文放在信封中。

user datagram (用户数据报) UDP 协议中的分组的名称。

User Datagram Protocol (UDP) (用户数据报协议) 无连接的 TCP/IP 运输层协议。

user interface (用户界面) 用户和应用程序之间的界面。

variable-length subnetting (可变长度子网划分) 使用不同的掩码在网络上创建子网。

Very high bit rate Digital Subscriber Line (VDSL) (甚高速率数字用户线) 短距离的基于 DSL 的技术。

Virtual Circuit (虚电路) 在发送计算机和接收计算机之间形成的逻辑电路。

Virtual Private Network (VPN) (虚拟专用网) 在物理上公用的网络上创建虚拟专用网络的技术。

web of trust (信任关系网) 在 PGP 中，由一群人共享的那些密钥环。

well-known port (熟知端口) 通常是在服务器上标志进程的口号。

Whirlpool 一种基于改良的 AES 的加密系统。

Whirlpool hash function (Whirlpool 散列函数) 基于 Whirlpool 加密系统的循环加密散列函数。

wide area network (WAN) (广域网) 一种网络，它使用可跨越很大地理距离的技术。

word (字) 在 AES 中，一组 32 位被当作一个实体来对待，是 4 字节矩阵中的一行或一列。

working group (工作组) 集中研究特定因特网课题的 IETF 委员会。

World Wide Web (WWW) (万维网) 一种多媒体的因特网服务，允许用户通过链接在因特网上从一个文档转移到另一个文档，这些链接把所有文档都连在一起。

X.25 定义数据终端设备和分组交换网之间的接口的 ITU-T 协议。

X.509 由 ITU 制订的建议，并且为因特网所接受，它以结构化的方式来定义证书。

zone (区) 在 DNS 中，被一台服务器负责或管辖的东西。

参 考 文 献

- [Bar et al. 05] Barrett, Daniel, J., Silverman, Richard, E., and Byrnes, Robert, G. *SSH: The Secure Shell*, Sebastopol, CA: O'Reilly, 2005.
- [Bis 05] Bishop, Matt, *Introduction to Computer Security*. Reading, MA: Addison-Wesley, 2005.
- [Cer 89] Cerf, V. *A History of Arpanet, The Interoperability Report*.
- [Com 06] Comer, Douglas E. *Internetworking with TCP/IP*, vol. 1. Upper Saddle River, NJ: Prentice Hall, 2006.
- [Don & Cal 01] Danaher, Michael J., and Calvert, Kenneth, L. *TCP/IP Sockets, version C*. San Francisco, CA: Morgan Kaufmann, 2003.
- [Dor & Har 03] Doraswamy, H., and Harkins, D. *IPSec*. Upper Saddle River, NJ: Prentice Hall, 2003.
- [Far 04] Frankel, S. *Demystifying the IPSec Puzzle*. Norwood, MA: Artech House, 2001.
- [For 03] Forouzan, B. *Local Area Networks*. New York: McGraw-Hill, 2003.
- [For 07] Forouzan, Behrouz. *Introduction to Data Communication and Networking*. New York: McGraw-Hill, 2007.
- [For 08] Forouzan, Behrouz. *Cryptography and Network Security*. New York: McGraw-Hill, 2008.
- [Gar & Vid 04] Garcia, A., and Widjaja, I. *Communication Networks*. New York: McGraw-Hill, 2004.
- [Gar 01] Garret, P. *Making, Breaking Codes*. Upper Saddle River, NJ: Prentice Hall, 2001.
- [Jen et al. 86] Jennings, D. M., Lancaster, L. M., Fuchs, I. H., Farber, D. H., and Arison, W. R. "Computer Networking for Scientists and Engineers," *Science*, vol. 231.
- [Kle 04] Kleinrock, L. *The Birth of the Internet*.
- [Koz 05] Kozierock, Charles M. *The TCP/IP Guide*. San Francisco: No Starch Press, 2005.
- [Kur & Ros 08] Kurose, James F., and Ross, Keith W. *Computer Networking*, 4th ed. Reading, MA: Addison-Wesley, 2008.
- [Lei et al. 98] Leiner, B., Cerf, D., Clark, R., Kahn, L., Kleinrock, D., Lynch, J., Postel, L., Roberts, and Woolf, S., *A Brief History of the Internet*.

- [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.
- [Mau & Sch 01] Mauro, D., and Schmidt, K. *Essential SNMP*. Sebastopol, CA: O'Reilly, 2001.
- [Mir 07] Mir, Nader F. *Computer and Communication Network*. Upper Saddle River, NJ: Prentice Hall, 2007.
- [Moy 98] Moy, John. *OSPF*. Reading, MA: Addison-Wesley, 1998.
- [Per 00] Perlman, Radia. *Interconnections*, 2nd ed. Reading, MA: Addison-Wesley, 2000.
- [Pet & Dav 03] Peterson, Larry L., and Davie, Bruce S. *Computer Networks*, 3rd ed. San Francisco: Morgan Kaufmann, 2003.
- [Res 01] Rescorla, E. *SSL and TLS*. Reading, MA: Addison-Wesley, 2001.
- [Rob & Rob 96] Robbins, Kay A., and Robbins, Steven. *Practical UNIX Programming*, Upper Saddle River, NJ: Prentice Hall, 1996.
- [Sam et al. 99] Sami, Iren, Amer, Paul D., and Conrad Phillip T. "The Transport Layer: Tutorial and Survey," *ACM Computing Surveys*, vol. 31, no. 4, Dec. 1999.
- [Seg 98] Segaller, S. *A Brief History of the Internet*.
- [Sta 04] Stallings, William. *Data and Computer Communications*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 1997.
- [Sta 06] Stallings, William. *Data and Computer Communications*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 1997.
- [Ste & Xie 01] Stewart, Randall, R. and Xie, Qiaobing. *Stream Control Transmission Protocol (STCP)*. Reading, MA: Addison-Wesley, 1998.
- [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 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.
- [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.
- [Wit & Zit 01] Wittmann, R., and Zitterbart, M. *Multicast Communication*. San Francisco: Morgan Kaufmann, 2001.