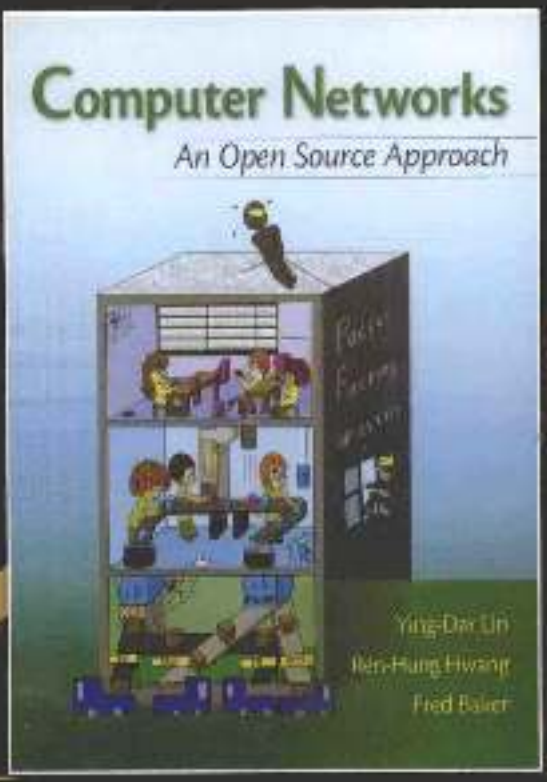
计 算 机 科 学 丛 书

计算机网络

——一种开源的设计实现方法

（中国台湾）Ying-Dar Lin Ren-Hung Hwang （美）Fred Baker 著
陈向阳 吴云桥 徐莹 译
李琼 审校

Computer Networks
An Open Source Approach



机械工业出版社
China Machine Press

计算机网络 一种开源的设计实现方法

Computer Networks An Open Source Approach

"本书对如何实现协议栈进行了详细的介绍。"

—— Mahaswera Sarkar，圣地亚哥州立大学

"本书以RFC和开源代码为基础，详细阐述了网络工程师的实践指导。"

—— Wen Chen，思科公司

"为了获得了教师以来设计与实现网络协议的资源，本书的模式设计的实例非常适合，使之成为一本理想的教科书。"

—— Mario Gerla，加州大学洛杉矶分校

本书讲述了为什么设计一个有效的协议比了解一个协议如何工作更重要，在解释协议行为的同时还介绍了它的核心概念和基本原理。为了进一步帮助读者理解设计和实现之间存在的差距，书中讨论了在何处以及如何基于Linux系统实现协议的设计。本书详细、全面地介绍了包括物理（8B/10B、OFDM、CRC32、CSMA/CD和crypto）、驱动程序（以太网和PPP）、内核（最长前缀匹配、攻击和、NAT、TCP拥塞控制、套接字、容器、调度器、防火墙和VPN），以及后台程序（RIP/OSPF/BGP、DNS、FTP、SMTP/POP3/IMAP4、HTTP、SNMP、SIP、流媒体和P2P）实现的56个开源实例。

本书特点

- 讲解为什么、解释以及如何设计和实现协议。
- 56个开源实例详细描述了核心协议和机制。
- 4个附录介绍因特网、开源社区、Linux内核、开发工具和调试工具。
- 包含69个有关历史演变（33）、行动原则（26）和性能问题（10）的专栏条。
- 每章后面都有常见问题解答和"常见陷阱"。
- 课堂所用PPT以及习题答案可以通过课程网站www.mhhe.com/lin获得。

作者简介

Ying-Dar Lin 中国台湾国立交通大学计算机科学教授。他于1993年从美国加州大学洛杉矶分校（UCLA）获得计算机科学博士学位。他目前担任《IEEE Communications Surveys and Tutorials》、《IEEE Communications Letters》和《Computer Communications,and Computer Networks》的编委。

Ren-Hung Hwang 中国台湾国立中正大学计算机科学系特聘研究教授。他于1993年在马萨诸塞大学阿默斯特分校获得计算机科学博士学位。他曾发表过150余篇有关计算机网络的国际会议论文。

Fred Baker 曾先后任职于CDC、Vitalink和ACC公司，一直活跃于网络和通信行业中。他目前是思科系统公司的会员。他曾担任IETF主席，主持许多IETF工作组。目前他是IETF与IPv6运行工作组的主席之一，并且是Internet Engineering Task Force Administrative Oversight Committee的会员之一。

Mc Graw Hill Education

www.mheducation.com

华章图书 www.hzbook.com

网上购书：www.china-pub.com

数字阅读：www.hzmedia.com.cn

ISBN 978-7-111-42604-2

定价：79.00元

计算机科学丛

计算机网络

—种开源的设计实现方法

[中国台湾] Ying-Dar Lin Ren-Hung Hwang [美] Fred Baker 著
陈向阳 吴云新 徐莹 译
李晖 审校

Computer Networks
An Open Source Approach



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机网络：一种开源方法的实现方法（中国台湾）林霖运，（中国台湾）黄仁勋，（美）贝克（Baker，F.）著；陈向阳等译．—北京：机械工业出版社，2013.6

（计算机科学丛书）

《名原文：Computer Networks：An Open Source Approach

ISBN 978-7-111-42604-2

Ⅰ．计… Ⅱ．①林… ②黄… ③贝… ④陈… Ⅲ．计算机网络 Ⅳ．TP393

中国版本图书馆 CIP 数据核字（2013）第 108358 号

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市嘉达律师事务所

本书版权登记号：图字：01-2011-5061

Ying-Dar Lin, Ren-Hung Hwang, Fred Baker. Computer Networks: An Open Source Approach（ISBN:978-0-07-337634-0）.

Copyright © 2012 by McGraw-Hill Education.

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

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

Copyright © 2014 by McGraw-Hill Education (Asia) and China Machine Press.

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

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

版权© 2014 由麦格劳-希尔（亚洲）教育出版公司与机械工业出版社所有。

本书封面贴有 McGraw-Hill Education 公司防伪标签，无标签者不得销售。

本书是目前国内外出版的计算机网络类教材中第一本以开放源代码实现形式介绍网络及应用问题的教科书。本书自底向上介绍网络的各层协议，每部分内容不仅介绍网络的基本概念、原理，而且还介绍网络的实现原理并给出实现的开源代码。通过 56 个开源网络的实现，详细讲解协议及设计的实现方法。学生通过实际动手和本书的全面推导，可以对网络有更加深刻的理解，动手能力将得到实质性的提高，可以有效解决学生学习计算机网络知识后不知所以然的问题。

本书可作为计算机科学或电子工程内高年级本科生或一年级研究生的计算机网络教材，还可以作为计算机网络教师的教学参考书，也可供数据通信行业的专业工程师使用。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：盛思源

北京诚信伟业印刷有限公司印刷

2014 年 2 月第 1 版第 1 次印刷

185mm×260mm · 30.75 印张

标准书号：ISBN 978-7-111-42604-2

定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991 88361066 报名热线：(010) 88379604

读者热线：(010) 88376294 88379649 68993259 读者信箱：kcsj@hzbook.com

文之复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了卓越的成果；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在工业化的进程中，美国的产业界与教育界紧密地结合，计算机学科中的许多泰斗人物同时身处科研和教学的最前线，由此产生的经典科学著作，不仅奠划了研究的范畴，还揭示了学术传承之道，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设一流大学的必由之路。

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

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

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

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

华章教育
华章科技图书出版中心

译者序

Computer Networks: An Open Source Approach

Ying-Dar Lin、Ren-Hung Hwang 和 Fred Baker 等人的《计算机网络：一种开源的设计实现方法》一书全面覆盖了计算机网络所涉及的内容，包括与技术相关的以及技术之外的内容，并探讨了计算机网络的最新进展和研究成果。既是一本不可多得的教科书，同时又是一本全面翔实的技术手册。它是目前国内外出版的计算机网络教材中第一本以开放源代码实现方式讨论网络层及其应用和相关实现问题的教科书。本书通过提供 56 个开放源代码的形式，缩短了理论知识与实际动手能力之间的差距，学生通过实际动手并在本书的全面指导下进行学习就可以对网络有更加深刻的理解，动手能力也将得到实质性的提高。目前有关计算机网络任几本国内外经典教材各有千秋，本书则是其中将理论知识与实际代码结合最紧密的，因此可以有效解决目前网络课程教学中所面临的学生虽然学习过计算机网络相应课程但还是不知其所以然的问题。所以，本书自然是广大计算机网络教师手边不可多得的教学参考书。其次对于立志网络教学方法改革的教师可以根据教材，那么就可以结合基本理论的开放源代码的实际项目实现，提高学生阅读代码的能力以及网络编程的能力和参与项目的能力。总之，高等院校计算机网络相关专业的学生、研究生、科研人员以及有一定基础的非专业人士都能够从本书中获益良多。虽然本书涉及的内容比较专业，但语言通俗易懂，内容由浅入深，结构层次清晰，并且使用了大量的图形和开放源代码帮助读者理解书中的内容。我们相信本书会给读者带来全新的感受与体验。

本书共分 8 章，分别介绍计算机网络基础知识、物理层、链路层、网络层、传输层、应用层、服务质量和网络安全的主体。4 个附录内容则是为了使对不熟悉 Linux 的读者阅读本书而提供的必要知识。

我们接受机械工业出版社的委托翻译本书，一方面是为了丰富国内有关计算机网络方面的教材，同时也可以在翻译过程中学习先进知识从而促进相应的科研工作。本书由武汉工程大学计算机科学与工程学院陈向阳、吴云娟以及湖北省档案局徐雪亚负责翻译，刘军、徐清、王剑、费波、闵开武、周洁、张宪军、段照、林好、罗雅丹、张敏等对本书的翻译提供了一定的帮助，在此特别致以衷心的感谢！本书的翻译得到了武汉工程大学研究生教材建设项目的资助，还得到了湖北省教育科学“十二五”规划 2013 年度立项课题“地方性高校新兴交叉学科建设的研究——以网络科学为例（项目编号 2013B060）”的资助。

在翻译时，因为本书涉及广泛的领域，具体包括应用数学、计算机科学、计算机应用、密码学、网络工程等领域，疏漏在所难免。虽然我们花了大量的时间和精力来翻译本书，但仍可能存在错误和不到之处。如果在使用本书时发现我们翻译中的疏漏和错误或者有更好的建议，敬请联系译者（xiangyangchenxun@yahoo.com.cn 和 xychenxun@aliyun.com）。我们期望与您一起努力，共同做好计算机网络的教学和研究。

译者

2013 年 12 月

网络课程的发展趋势

计算机网络中的技术已经经历了很多代的演变。许多技术失败了或逐渐消失了，而有些技术流行起来，还有些技术目前正在孕育即将出市。由 TCP/IP 驱动的因特网技术目前成为主流。因此，计算机网络课程内容组织的一个明显趋势就是围绕 TCP/IP 来进行，再加上一些更底层的链路技术和许多上层应用，同时也删去了一些过时技术的详细介绍，或许要解释它们为什么会过时。

计算机网络教科书也已经经历了多次演变，从传统的并且有时枯燥的协议描述到应用驱动的自顶向下的实现方法和系统化的实现方法。趋势之一就是除了解释协议行为如何工作外，更多地解释为什么会这样，以便读者可以更好地理解各种协议的设计。当然，演变还在继续。

设计与实现之间的差距

另外一个不那么明显的趋势就是为协议的描述添加了实用特色。其他教科书的读者可能不知道协议设计在哪里以及如何实现。最终只会让读者在进入研究生院做研究时，往往只会采用仿真的方法对性能进行评估，而不是使用具有真实基准测试的实际实现。那么当他们毕业后工作时就又要从头开始学习实现环境、技巧和相关问题。很显然采用这些教科书培训过的学生在理论知识和技能之间必然存在很大差距。这种差距可以很容易用从开源社区获得的真在使用的代码来弥补。

一种开源的实现方法

目前几乎所有正在使用的协议已经在 Linux 操作系统和许多开源软件包中得到实现。Linux 和开源社区正在不断地成长壮大。他们的应用在网络世界中占据着主导。但是，他们所提供的资源还不能被计算机科学，更确切地说是计算机网络中的教科书所阐述。我们预测一种教科书的发展趋势，对于多门课程可以补充支持开放源代码资源以便缩小专业领域知识和动手技能之间的差距。这些课程包括操作系统（利用 Linux 内核实现作为进程管理、内存管理、文件系统管理、I/O 管理等的例子），计算机组成原理（利用 www.opencores.org 中的 verilog 代码作为处理器、内存单元、I/O 设备控制器等的例子），算法（利用 GNU 库作为经典算法的例子）以及计算机网络（利用开放源代码作为协议实现的例子）。本书将是证明这种发展趋势的一个最早例子。

我们的开放源代码方法通过在协议行为描述中穿插介绍从开放源代码软件包中提取的样例实现来弥补上述差距。这些例子清楚地进行了编号，例如开放源代码实现 3.4。本书中还包括了可以下载完整可用例子的源代码站点。这样学生可以很容易地在因特网上访问这些内容。例如，在解释了路由表查询中的最长前缀匹配概念之后，我们接着演示了路由表是如何组织的（根据前缀长度排序的散列表数组），以及在 Linux 内核中如何实现匹配（如首次匹配，匹配过程是从最长前缀的散列表开始的）。这样就能够讲解路由表查询的设计及其实现，并提供扎实的动手练习项目。例如，描述路由表查询的瓶颈或者修改散列表的实现。我们认为这种穿插介绍方法要比单独采用另外一门课或另外一本教科书

更好。这样就会更加有利于普通学生学习，因为可以将设计与实现结合起来，并且大多数学生并不需要修二门课程。若还需要采用其他教科书，指导教师、助教和学生就必须额外努力去弥补忽略的或者在大多数情况下根本就没有触及的类型。

本书中的协议描述穿插了有代表性的 56 个开放源代码实现，涉及范围包括从编译码器（coder）的 Verilog 或 VHDL 码，循环冗余、CRC32、CSMA/CD 和密码技术，到与配置驱动器的 C 代码，PPP 后台守护程序和驱动程序，最长前缀匹配，IP/TCP/UDP 校验和，NAT，RIP/OSPF/BGP 路由后台守护程序，TCP 慢启动和拥塞避免，套接字（socket），流行的软件包支持 DNS、FTP、SMTP、POP3、SNMP、HTTP、SIP、流、P2P，到 QoS 队列，如流量整形器和调度器，以及安全功能，如防火墙、VPN 和入侵检测。这种系统认知可以通过每个源代码实现最后和每章最后的动手练习得到加强，读者需要运行、改变、延长、编译给定或修改特定为被代码设计的程序。通过程序或守护程序，学生若具备了这种系统认知和操作技能，加上他们的协议专业领域知识，就能在学术上进行扎实的研究工作并在业界进行扎实的开发任务。

理解协议为什么被设计成某种方式要比了解协议如何工作更加重要

编写本书的理念是理解为什么协议被设计成某种方式要比了解协议如何工作更加重要。在解释有关机制或协议如何工作之前先用图示说明大多数关键概念和基本原理。它们包括无状态、控制平面和数据平面，路由和交换，冲突和扩展域，桥接的可扩展性，无关和有关路由，地址翻译和配置，转发与路由，网口流量控制、RTT 估计，知名或已知动态端口、透气服务器（又称为循环服务器）和开发服务器、ASCII 应用协议消息，可变长度与固定字段协议消息，透明代理，以及许多其他内容。

曲笔与理解一样重要，它们值得特殊处理以便明确标示区分出来。每一章都是先从一组问题开始，然后再提出基本的问题。我们已经添加了行动原则、历史演变和性能问题等工具条。我们以无编号的常见陷阱（读者社区常见的误解）、进一步阅读、常见问题解答（读者预习和复习的关问题），以及一组动手练习和书面练习的小节作为结束。

读者预备的技能

无论指导教师还是学生熟悉 Linux 系统与否都不应该成为采用本书的决定因素。与 Linux 相关的动手练习技巧在附录 B、C、D 中进行了全面介绍。这三个附录会让读者具有足够的动手实作技巧，包括 Linux 内核定注（宿在源代码跟踪的指导）、开发工具（vim、gcc、make、gdb、ddd、kgdb、nsnpe、cvs/svn、qprof/kernprof、busybox、buildroot）、网络工具（host、arp、ifconfig、ping、traceroute、tcpdump、wireshark、netstat、ttcp、webbench、ns、nist-net、nessus）。附录 A 中还有一节向读者介绍开放源代码资源。在第 1 章中还包括有关“数据包的生命历程”一节，它生动地理解了本书的脉络。

在教学中，对课堂采用开放源代码实现遇到的障碍也进行了考虑。本书不是采用代码列表和解释，而是在需要时将它结构化组织到描述、框图、数据结构、算法实现和练习中。这样学生和指导教师都易于使用。

教学特点和补充材料

教科书一般都具有很多特点以便帮助读者，具有许多课堂支持材料以便帮助指导教师。本书具有很多特点和课堂支持材料，总结如下：

1）56 个明确编号的关键协议和机制的开放源代码实现。

2）4 个附录分别介绍互联网和开源社区的名人录（重要人物）、Linux 内核概述、开发工具和网络实用工具。

3）逻辑推理协议为什么、怎么以及如何设计和实现。

4）在每章的开头以一般问题作为激动，回答以大的问题作为结尾。

5）“数据包的生命历程”从服务器和路由器的角度演示本书的路标并演示如何跟踪代码中的数据包流。

6）在每章最后剖解的“常见陷阱”，标示出常见的误解。

7）除了书面练习之外，还有基于 Linux 的动手练习。

8）给出 69 个有关历史演变、原理、行动原则和性能问题的级块。

9）每章最后的常见问题解答有助于读者抓住主要问题回答并使于在读完每一章后进行复习。

10）课堂支持材料，包括 PowerPoint 演讲幻灯片，习题解答，并且在 PowerPoint 中采用的所有教材图片都可以从课本的网站（www.mhhe.com/lin）中找到。

读者和课程路标指南

本书可作为计算机科学或电子工程的高年级本科生或一年级研究生的计算机网络教材，也可供数据通信行业的专业工程师使用。对于本科生的课程，我们建议指导老师仅讲解第 1－6 章。作为研究生的课程，应该讲解所有的章节。对于既要为本科生又要为研究生授课的指导老师，两种其他可能的区别在于研究生课程会有更重的动手练习和额外的阅读作业布置。不管是本科生还是研究生课程，指导教师都可以让学生在前几周学习附录以便于熟悉 Linux 及其开发和应用工具。这种熟悉程度既可以通过动手练习测试，也可以通过动手练习作业来检验。在整个课程中，既可以布置书面练习，也可以布置动手练习以便加强知识和技巧的掌握。

本书各章的内容组织如下：

- 第 1 章为网络的需求和原理提供了背景知识，然后给出互联网解决方案来满足由基本原理提出的需求。剖示了互联网的设计理念，如无状态的、无连接的，以及端对端参数。在整个过程中，我们介绍了关键概念，包括连接性、可扩展性、资源共享、数据和控制平面、分组与电路交换、延迟、吞吐量、带宽、负载、丢包、抖动、标准和互操作性、路由和分组。接下来，我们将以 Linux 作为互联网解决方案的实现，以便于演示互联网体系结构及其协议在哪里以及如何在芯片、驱动程序、内核和后台守护程序中实现。本章最后以本书路标和“数据包的生命历程”的有趣描述作为结束。
- 第 2 章对物理层进行了简洁的论述。本章首先介绍有关模拟和数字信号、有线和无线媒体、编码、调制和多路复用的概念，然后介绍有关编码、调制和多路复用的技术和标准。两个开放源代码实现分别演示了使用 8B/10B 编码和使用 OFDM 的 WLAN PHY 的以太网 PHY 的硬件实现。

- 第3章介绍了三种主流链路：PPP、以太网（Ethernet）和WLAN。此外，还描述了蓝牙（Bluetooth）和WiMAX。然后介绍了通过第2层桥接的局域网互联。最后，详细描述了网络接口卡发送和接收分组的适配器驱动程序，给出了包括CRC32和以太网MAC的硬件实现等10个开放源代码实现。
- 第4章讨论了IP层的数据平面和控制平面。数据平面讨论包括IP转发处理、路由表查找、校验和、分段、NAT和有争议的IPv6，而控制平面讨论包括地址管理、错误报告、单播路由和组播路由。讲述了路由协议和算法。12个开放源代码实现实例中阐述了这些设计是如何实现的。
- 第5章上升到传输层，讲述有关端到端或主机到主机之间的问题。详细讲解了UDP和TCP，特别是有关TCP的设计理念、行为和版本。然后介绍了用于实时多媒体流量的RTP。最后通过专门的一节内容演示了套接字的设计与实现，这里分组在内核空间和用户空间之间复制。给出了10个开放源代码实现。
- 第6章既介绍传统的应用，包括DNS、Mail、FTP、Web和SNMP，也介绍新的应用，包括VoIP、流媒体和P2P应用。对实现这8种应用的开放源代码软件包进行了讨论。
- 第7章介绍了有关QoS的高级话题，讲述了各种流量控制模块，如策略器、整形器、调度器、丢弃器以及许可控制。尽管IntServ和DiffServ标准框架没有得到广泛的部署，但这些流量控制模块中的许多已经嵌入我们日常使用的产品中。因此它们值得专门利用完整的一章来叙述。给出了6个开放源代码实现。
- 第8章深入学习网络安全问题，范围包括访问安全（由TCP/IP防火墙和应用防火墙来保护）、数据安全（由VPN来保护）、以及系统安全（由入侵检测和反病毒软件来保护）。既介绍了算法（表示法、加密、认证、深度分组检查），也介绍了标准（3DES、MD5、IPsec）。最后还增加了8个开放源代码实现。

致谢

本书的草稿经历了多次演变和修订。在整个过程中，很多人直接或间接地提供了贡献。首先，许多中国台湾交通大学、中国台湾中正大学以及思科公司的试验室成员和同事为本书贡献了很多想法，列了私代码解释。这里要特别感谢Po-Ching Lin、Shih-Chiang Wenfue Tsao、Yi-Neng Lin、Huan-Yun Wei、Bow-Jyc Chang、Shun-Lee Stanley Chang、Yuan-Cheng Lai、Joi-Tsun Jason Hung、Shuo-Yu Jason Cheng、Chia-Yu Ku、Hsiao-Feng Francis Lu和Frank Liu。没有他们的帮助，我们就不可能将许多有趣的和原创性的思想编入本书。我们还要感谢中国台湾科学委员会（National Science Council，NSC）、工业技术研究院（Industrial Technology Research Institute，ITRI）、D-Link公司、Realtek半导体公司、ZyXEL公司、思科公司以及英特尔公司在过去几年中对我们网络研究工作的支持。

其次，我们感谢以下审阅了全部或者部分手稿的人：Emmanuel Agu，伍斯特理工学院（Worcester Polytechnic University）；Tricha Anjali，伊利诺伊理工大学（Illinois Institute of Technology）；Ladislau Boloni，中佛罗里达大学（University of Central Florida）；Charles Colbourn，亚利桑那州立大学（Arizona State University）；XiaoJiang Du，天普大学（Temple University）；Jiang Guo，加州州立大学洛杉矶分校（California State University，Los Angeles）；Robert Kerbs，加州州立理工大学波莫那分校（California State

Polytechnic University，Pomona）；Fang Liu，德克萨斯大学泛美分校（The University of Texas Pan American）；Oge Marques，佛罗里达泽大西洋大学（Florida Atlantic University）；Mitchell Neilsen，堪萨斯州立大学（Kansas State University）；Mahasweta Sarkar，圣地亚哥州立大学（San Diego State University）；Edwin Sloan，西尔斯波洛社区学院（希尔）（Hillsborough Community College）；Ioannis Vlachos，北卡罗来纳州立大学（North Carolina State University）；Bin Wang，莱特州立大学（Wright State University）；Daniel Zappala，杨百翰大学（Brigham Young University）。还要感谢南堆应用科技大学的王志强，他从语法上润色了我们的手稿。

最后，我们要感谢麦格劳－希尔公司（McGraw-Hill）的员工，在他们的指导下，我们顺利地过了编辑和出版阶段。特别感谢全球发行商 Raghu Srinivasan，我们的策划编辑 Lorraine Rusczek，出版项目经理 Jane Mohr 以及项目经理 Deepti Narwat，正是在他们的大力协助下，我们得以应服务和挑战。

作者简介

Computer Networks: An Open Source Approach

Ying – Dar Lin（林盈达） 中国台湾交通大学（NCTU）计算机科学教授。生于1993年从美国加利福尼亚大学洛杉矶分校（UCLA）获得计算机科学博士学位。曾于2007—2008年以休假的形式担任于思科系统公司担当研究学者。自从2002年起，他任网络基准实验室（Network Benchmarking Lab，NBL，www.nbl.org.tw）实验室主任兼NBL的创立者，NBL现已实施通信软件及软硬网络产品。他还于2002年与他人合作共同建立了利维网络有限公司（L7 Networks Inc.），该公司已经被D-Link公司收购。他的研究兴趣在于网络协议和算法的设计、分析、实现和标杆分析法，服务质量，网络安全，深度分组处理，P2P网络互联，以及嵌入式硬件/软件结构设计。他在"多跳蜂窝"的研究工作被引用过500多次。他主要担任《IEEE Communications Magazine》（IEEE通信杂志）、《IEEE Communications Surveys and Tutorials》（IEEE通信调查和教程）、《IEEE Communications Letters》（IEEE通信快报）、《Computer Communications》（计算机通信）和《Computer Networks》（计算机网络）的编委。

Ren – Hung Hwang（黄仁宏） 中国台湾中正大学计算机科学系特聘研究教授，并任副校长与中心主任。他于1993年在马萨诸塞大学阿默斯特分校获得计算机科学博士学位。他曾发表过150余篇有关计算机网络的国际会议论文。他的研究兴趣包括无线计算、P2P网络、下一代无线网络以及网络教育。他担任2009年度在中国台湾高雄市举办的第十届普适系统、算法和网络研讨会的程序委员会主席。他目前任《Journal of Information Science and Engineering》（信息科学与工程杂志）的编委。他曾荣获中国台湾中正大学2002年度杰出教学奖，并获得中国台湾教育部门1998—2001年通信网络课件杰出设计奖。他目前还任TWNIC IP委员会以及中国台湾"中华工程教育学会"（IECE）认证规则与程序委员会的委员。

Fred Baker（弗雷德·贝克） 自20世纪70年代末期，一直活跃于网络和通信行业中，曾先后任职于CDC、Vitalink和ACC。他目前在思科系统公司担任会员。他于1996—2001年曾担任IETF主席。他已经主持了很多IETF工作组，包括Bridge MIB、DS1/DS3 MIB、ISDN MIB、PPP扩展、IEPREP和IPv6运行。并且从1996—2002年服务于互联网架构委员会。他曾经与人合著或编辑过大约40个RFC并发其中RFC也被用过很多次。具体领域包括网络管理、OSPF和RIPv2路由、服务质量（实用集成服务或差分服务模型）、合法拦截、可联网计算机优先级服务等。此外，他还于2002—2008年担任互联网协会理事会（Board of Trustees of the Internet Society）的会员，并于2002—2006年期间担任主席。他还在联邦通信委员会技术咨询委员会（Technical Advisory Council of the Federal Communications Commission）的董事会担任一职。他当前在IETF中IPv6运行工作组的联合主席，并且还在互联网工程任务组行政监督委员会（Internet Engineering Task Force Administrative Oversight Committee）的会员。

推荐阅读



TCP/IP详解 卷1：协议（英文版·第2版）

作者：Kevin R. Fall 等 ISBN：978-7-111-35228-7 定价：129.00元



计算机网络：系统方法（英文版·第5版）

作者：Larry L. Peterson 等 ISBN：978-7-111-37720-7 定价：139.00元



计算机网络教程：自顶向下方法（英文版）

作者：Behrouz A. Forouzan 著 ISBN：978-7-111-38705-5 定价：79.00元



计算机网络（英文版·第5版）

作者：Andrew S. Tanenbaum 等 ISBN：978-7-111-38975-2 定价：69.00元

推荐阅读



云计算与分布式系统：从并行处理到物联网

作者：（美）Kai Hwang 等 ISBN：978-7-111-41065-2 定价：85.00元



嵌入式系统导论：CPS方法

作者：（美）Edward Ashford Lee 等 ISBN：978-7-111-50201-6 定价：66.00元



大规模并行处理器程序设计（英文版·第2版）

作者：（美）David B. Kirk 等 ISBN：978-7-111-41629-5 定价：79.00元



多处理器编程的艺术（修订版）

作者：（美）Maurice Herlihy 等 ISBN：978-7-111-54568-0 定价：59.00元

出版者的话
译者序
前言
作者简介

第1章 基础知识 …… 1
1.1 计算机网络互联的需求 …… 1
1.1.1 连通性：节点、链路、路径 …… 1
1.1.2 可扩展性：节点的数量 …… 4
1.1.3 资源共享 …… 4
1.2 基本原理 …… 6
1.2.1 性能测量 …… 6
1.2.2 控制平面上的操作 …… 9
1.2.3 数据平面上的操作 …… 10
1.2.4 互联作用 …… 12
1.3 互联网体系结构 …… 13
1.3.1 连通性解决方案 …… 13
1.3.2 可扩展性解决方案 …… 15
1.3.3 资源共享的解决方案 …… 17
1.3.4 控制平面和数据平面操作 …… 18
1.4 开放源代码实现 …… 19
1.4.1 开放与利用 …… 19
1.4.2 Linux 系统中的软件体系结构 …… 20
1.4.3 Linux 内核 …… 22
1.4.4 客户端和守护进程服务器 …… 22
1.4.5 接口驱动程序 …… 23
1.4.6 设备控制器 …… 23
1.5 本书路标：数据包的生命历程 …… 24
1.5.1 数据包数据结构：sk_buff …… 24
1.5.2 在 Web 服务器中数据包的生命历程 …… 25
1.5.3 数据包在网关中的生命历程 …… 26
1.6 总结 …… 29
演变的沙砾 …… 29
常见陷阱 …… 29
进一步阅读 …… 30
常见问题解答 …… 32
练习 …… 33

第2章 物理层 …… 36
2.1 一般性问题 …… 36
2.1.1 数据和信号：模拟的和数字的 …… 37
2.1.2 发送和接收流 …… 39
2.1.3 传输：线路编码和数字调制 …… 40
2.1.4 传输损失 …… 41
2.2 介质 …… 42
2.2.1 有线介质 …… 42
2.2.2 无线介质 …… 45
2.3 信息编码和其带传输 …… 46
2.3.1 信源编码和信道编码 …… 46
2.3.2 线路编码 …… 46
2.4 数字调制和多路复用 …… 54
2.4.1 通带调制 …… 54
2.4.2 多路复用 …… 61
2.5 高级主题 …… 62
2.5.1 扩频 …… 63
2.5.2 单载波与多载波 …… 69
2.5.3 多输入，多输出 …… 70
2.6 总结 …… 77
常见陷阱 …… 77
进一步阅读 …… 78
常见问题解答 …… 80

练习 …… 81
第3章 链路层 …… 83
3.1 一般问题 …… 84
3.1.1 成帧 …… 84
3.1.2 寻址 …… 85
3.1.3 差错控制和可靠性 …… 86
3.1.4 流量控制 …… 90
3.1.5 介质访问控制 …… 90
3.1.6 桥接 …… 91
3.1.7 链路层的数据包流 …… 91
3.2 点到点协议 …… 93
3.2.1 高级数据链路控制 …… 93
3.2.2 点到点协议 …… 95
3.2.3 因特网协议控制协议 …… 96
3.2.4 以太网上的 PPP（PPPoE） …… 98
3.3 以太网（IEEE 802.3） …… 98
3.3.1 以太网的演变：盒网 …… 98
3.3.2 以太网 MAC …… 100
3.3.3 以太网的精选主题 …… 109
3.4 无线链路 …… 112
3.4.1 IEEE 802.11 无线局域网 …… 112
3.4.2 蓝牙技术 …… 119
3.4.3 WiMAX 技术 …… 121
3.5 桥接 …… 124
3.5.1 自学习 …… 125
3.5.2 生成树协议 …… 128
3.5.3 虚拟局域网 …… 130
3.6 网络接口的设备驱动程序 …… 133
3.6.1 设备驱动程序的概念 …… 133
3.6.2 在 Linux 设备驱动程序中如何与硬件通信 …… 134
3.7 总结 …… 140
常见陷阱 …… 140
进一步阅读 …… 142
常见问题解答 …… 141
练习 …… 145
第4章 互联网协议层 …… 147
4.1 一般问题 …… 147
4.1.1 连通性问题 …… 147
4.1.2 可扩展性问题 …… 148
4.1.3 资源共享问题 …… 149
4.1.4 IP 层协议和分组流概述 …… 149
4.2 数据平面协议：互联网协议 …… 152
4.2.1 互联网协议版本 4 …… 152
4.2.2 网络地址翻译 …… 162
4.3 互联网协议版本 6 …… 169
4.3.1 IPv6 头部格式 …… 170
4.3.2 IPv6 扩展头部 …… 171
4.3.3 IPv6 中的分段 …… 171
4.3.4 IPv6 地址的表示法 …… 172
4.3.5 IPv6 地址空间分配 …… 172
4.3.6 自动配置 …… 174
4.3.7 从 IPv4 到 IPv6 的迁移 …… 174
4.4 控制平面协议：地址管理 …… 175
4.4.1 地址解析协议 …… 175
4.4.2 动态主机配置 …… 177
4.5 控制平面协议：错误报告 …… 181
4.6 控制平面协议：路由 …… 184
4.6.1 路由原理 …… 184
4.6.2 域内路由 …… 192
4.6.3 域间路由 …… 200
4.7 组播路由 …… 204
4.7.1 将复杂性迁移到路由器 …… 204
4.7.2 组成员管理 …… 205
4.7.3 组播路由协议 …… 206
4.7.4 域间组播 …… 212
4.8 总结 …… 214
常见陷阱 …… 214
进一步阅读 …… 215
常见问题解答 …… 218
练习 …… 220
第5章 传输层 …… 224
5.1 一般问题 …… 224

5.1.1 节点到节点与端到端 …… 225
5.1.2 差错控制和可靠性 …… 226
5.1.3 流量控制：流量分析和拥塞分析 …… 226
5.1.4 标准编程接口 …… 227
5.1.5 传输层分组流 …… 227
5.2 不可靠的无连接传输：UDP … 229
5.2.1 头部格式 …… 229
5.2.2 差错控制：各个分段的校验和 …… 230
5.2.3 承载单播/组播实时流量 … 231
5.3 可靠的面向连接的传输：TCP … 231
5.3.1 连接管理 …… 231
5.3.2 数据传输的可靠性 …… 234
5.3.3 TCP 流量控制 …… 236
5.3.4 TCP 拥塞控制 …… 239
5.3.5 TCP 头部格式 …… 245
5.3.6 TCP 定时器管理 …… 246
5.3.7 TCP 性能问题及增强 …… 249
5.4 套接字编程接口 …… 258
5.4.1 套接字 …… 258
5.4.2 通过 UDP 和 TCP 绑定应用程序 …… 258
5.4.3 绕过 UDP 和 TCP 传输 …… 263
5.5 用于实时流量的传输协议 …… 266
5.5.1 实时需求 …… 266
5.5.2 标准数据平面协议：RTP … 268
5.5.3 标准控制平面协议：RTCP …… 268
5.6 总结 …… 269
常见陷阱 …… 270
进一步阅读 …… 270
常见问题解答 …… 272
练习 …… 273
第 6 章 应用层 …… 276
6.1 一般问题 …… 277
6.1.1 端口如何工作 …… 278
6.1.2 服务器如何启动 …… 278
6.1.3 服务器分类 …… 278
6.1.4 应用层协议的特点 …… 282
6.2 域名系统 …… 282
6.2.1 简介 …… 282
6.2.2 域名空间 …… 283
6.2.3 资源记录 …… 284
6.2.4 名字解析 …… 286
6.3 电子邮件 …… 291
6.3.1 简介 …… 291
6.3.2 互联网邮件标准 …… 292
6.3.3 互联网邮件协议 …… 296
6.4 万维网 …… 303
6.4.1 简介 …… 303
6.4.2 Web 命名和寻址 …… 304
6.4.3 HTML 和 XML …… 306
6.4.4 HTTP …… 306
6.4.5 Web 缓存和代理 …… 309
6.5 文件传输协议 …… 314
6.5.1 简介 …… 314
6.5.2 双连接操作模型：带外信令 …… 315
6.5.3 FTP 协议消息 …… 316
6.6 简单网络管理协议 …… 320
6.6.1 简介 …… 320
6.6.2 体系结构框架 …… 320
6.6.3 管理信息库 …… 321
6.6.4 SNMP 中的基本操作 …… 324
6.7 VoIP …… 328
6.7.1 简介 …… 328
6.7.2 H.323 …… 329
6.7.3 会话初始化协议 …… 331
6.8 流媒体 …… 336
6.8.1 简介 …… 336
6.8.2 压缩算法 …… 337
6.8.3 流媒体协议 …… 337
6.8.4 服务质量和同步机制 …… 339

6.9 对等应用程序 …… 342
6.9.1 简介 …… 342
6.9.2 P2P 的体系结构 …… 344
6.9.3 P2P 应用的生活问题 …… 348
6.9.4 案例研究：BitTorrent …… 349
6.10 总结 …… 354
常见陷阱 …… 354
进一步阅读 …… 355
常见问题解答 …… 357
练习 …… 359
第 7 章 互联网服务质量 …… 361
7.1 一般问题 …… 362
7.1.1 信令协议 …… 362
7.1.2 QoS 路由 …… 362
7.1.3 许可控制 …… 363
7.1.4 分组分类 …… 363
7.1.5 监管 …… 363
7.1.6 调度 …… 363
7.2 QoS 体系结构 …… 365
7.2.1 集成服务 …… 365
7.2.2 区分服务 …… 367
7.3 QoS 组件的算法 …… 372
7.3.1 许可控制 …… 372
7.3.2 漏桶 …… 374
7.3.3 令牌桶 …… 375
7.3.4 分组调度 …… 378
7.3.5 分组丢弃 …… 383
7.4 总结 …… 386
常见陷阱 …… 386
进一步阅读 …… 386
常见问题解答 …… 388
练习 …… 388
第 8 章 网络安全 …… 390
8.1 一般问题 …… 390
8.1.1 数据安全 …… 390
8.1.2 协同安全 …… 391
8.1.3 系统安全 …… 392
8.2 数据安全 …… 392
8.2.1 密码学原理 …… 392
8.2.2 数字签名和消息认证 …… 398
8.2.3 链路层选项 …… 401
8.2.4 IP 安全 …… 401
8.2.5 传输层安全 …… 404
8.2.6 VPN 技术的比较 …… 406
8.3 访问安全 …… 407
8.3.1 简介 …… 407
8.3.2 网络层/传输层防火墙 …… 407
8.3.3 应用层防火墙 …… 410
8.4 系统安全 …… 412
8.4.1 信息收集 …… 412
8.4.2 漏洞利用 …… 412
8.4.3 恶意代码 …… 415
8.4.4 典型的防御 …… 417
8.5 总结 …… 424
常见陷阱 …… 425
进一步阅读 …… 426
常见问题解答 …… 428
练习 …… 428
附录 A 名人录 …… 430
附录 B Linux 内核概述 …… 440
附录 C 开发工具 …… 450
附录 D 网络实用工具 …… 466

第1章
Computer Networks: An Open Source Approach

基础知识

计算机网络或数据通信是有关计算机系统或设备之间通信的一组规则。它有其自身的需求和基本原理。自从1969年ARPANET（美国国防部高级研究计划局网络，后改名为因特网）建立了第一个节点后，有能转发分组交换技术就成为因特网体系结构，这是一个满足需求和数据通信基本原理的解决方案。这种解决方案于1983年与TCP/IP协议套件集合，此后就开始了演变过程。

互联网，或TCP/IP协议套件，其实只是恰好占据统治地位的众多可能解决方案之一。当然还存在其他的解决方案，也能符合需求并能满足数据通信的基本原理。例如，以是开发于20世纪70年代的X.25和开放系统互连（OSI），却最终被TCP/IP所替代。异步转移模式（ATM），曾经在20世纪90年代风靡一时，但因为难以与TCP/IP协议兼容而最终消失匿迹。多协议标签交换（MPLS）得以幸存下来，因为它从一开始就是作为TCP/IP协议的补充而设计的。

同样，在各种计算机系统或设备上也有许多互联网解决方案的实现。其中，开源实现与互联网架构一样共享同一种开放的和自底向上的精神，为公众提供实际可获得的软件源代码。在这种自底向上的方法中，志愿者能够以自己的设计或实现，同时寻求开发人员社区的支持和共识，这与受权威驱动的自顶向下的方法刚好相反。因为是开源和免费提供，所以这些实现能够作为各种网络机制如何在特定情况下工作的可靠的运行例子。

在本章中，我们让读者熟悉本书通篇用到的计算机网络基础知识。1.1节通过用连通性、可扩展性和资源共享对计算机网络进行定义以便标识进行数据通信的关键需求，还介绍了分组交换的概念。1.2节阐述了数据通信的基本原理。首先定义带宽、提供负载、非吐量、延迟、延迟变化和丢失等性能测量。然后，解释了用于处理控制分组和数据分组的协议和算法的设计问题。因为互联网是计算机网络的一种可能解决方案，所以1.3节介绍了连通性、可扩展性和资源共享以及控制分组和数据分组处理的互联网版本的解决方案。1.4节讨论了开源实现是如何进一步在运行的系统尤其是在Linux操作系统中实现互联网解决方案。我们说明了各种协议和算法模块在计算机系统的内核、驱动程序、守护程序以及控制器中实现的原因和方式。我们在1.5节利用分组经过Web服务器和中间互联设备中各种模块的历程画出了本书的学习路标。本节还为理解以后章节中将描述的开源实现奠定基础。互联网解决方案的设计和开源实现的贡献者，连同其他昙花一现的网络技术，将放在附录A中作为本章补充材料加以复习。

通过本章学习后，读者应该能够解释：1）为什么互联网解决方案是以目前这样的方式设计的；2）这种开放的解决方案是如何在实际系统中实现的。

1.1 计算机网络互联的需求

对计算机网络的一套需求可以转化为在设计、实施、操作计算机网络时必须满足的一组目标。多年来，这组目标虽在逐渐变化，但其核心需求仍保持不变："通过各种共享媒体和设备连接越来越多的用户和应用程序，以便它们可以相互通信。"这句话表明数据通信的三个需求以及要解决的有关问题：1）连通性：谁以及如何连接；2）可扩展性：有多少连接；3）资源共享：如何利用连接。本节将介绍这些核心需求并讨论大多数计算机网络（不仅是互联网）满足这些需求的通用解决方案。

1.1.1 连通性：节点、链路、路径

计算机网络，从连通性方面，可视为"从一组节点和链路构造的连通图，这里任何一对节点可以通过由一系列联系起来的节点和链路组成的路径到达彼此"。我们人类用户之间需要连通性以便交换消息或加入谈话，应用程序之间需要连通性以便维护网络操作，或者在用户和应用程序之间需要连通性以便访问数据或服务。可以使用各种媒体和设备建立节点之间的连通性，设备可以是条线器、交换

机、路由器或开关，而媒体可以是有线的或无线的。

节点：主机或中间设备

计算机网络中的节点既可以是一台主机也可以是一台中间互联设备。前者是一台系统用户可应用程序的终端计算机；而后者充当一个中间节点，提供多个链路接口连接主机或其他中间节点。像集线器、交换机、路由器和网关等设备就是常见的中间节点。与基于计算机的主机不同，中间节点设备可能会配备专门设计的CPU甚至硬件以提高处理速度或降低硬件和处理成本。随着链路或流速度的增加，或者处理需要更快的CPU已需要特殊的硬件，如专用集成电路（ASIC），从CPU上卸载负荷。

链路：点对点或广播

计算机网络中的链路如果正好连接了两个节点（在每个端点一个），就称为点对点，如果它连接两个以上节点，就称为广播。关键的区别在于，连接到一个广播链路上的节点需要通过竞争获得发送权。如果点对点链路是全双工链路，那么两边其中的节点同时传输就可以随意发送；如果是半双工链路，就会轮流发送；如果是单工链路，则利用两条链路传输，每个方向一条链路。即全双工链路支持双向同时传输，半双工链路允许同时在两个方向或在某时刻仅在一个方向上的双向通信，而单工链路仅支持单向通信。

链路的物理性质既可以是有线的也可以是无线的，既可以是点对点也可以是广播式的。一般情况下，有线或无线的局域网（LAN）中的链路是广播式的，而广域网（WAN）中的链路是点对点的。这因为广播链路中使用的多路访问方法通常在短距离内更有效，这一点我们将在第3章中看到。然而，也会存在例外。例如，为广域网设计的基于卫星的ALOHA系统使用广播式的链路。以太网，最初为局域网设计的广播链路，已经演变成无论在局域网还是在广域网中都使用的点对点链路。

有线或无线

对于有线链路，常见的媒体包括双绞线、同轴电缆和光纤。双绞线将两根铜线绞在一起以获得对噪声的更好免疫力，它们广泛用于普通老式电话系统（POTS）和局域网（如以太网）的接入链路。5类（Cat-5）双绞线比家庭使用的POTS线具有更高的规格，可以10Mbps传输几千米远或以1Gbps传输100多米。同轴电缆利用更厚的绝缘护套将更粗的铜线与较细的嵌入铜线分开，适合远距离传输，如适用于跨越40公里地区的100个6MHz电视频道的有线电视分发。通过电缆调制解调器，某些信道中的每一个，可以数字化为30Mbps的数据、语音或者视频服务。光纤具有大容量，而且它可以将信号传输到更远。远距干线用于骨干网络（Gbps～Tbps），有时用于本地局域网络（100Mbps～10Gbps）。

对于无线链路，有无线电（10^7～10^9Hz）、微波（10^9～10^{11}Hz）、红外（10^{11}～10^{14}Hz），以及其他（紫外线、X射线、伽马射线）不断提高的传输频率数量级。低频率（低于几个GHz）的无线链路通常是广播式，这是全向的，而超过几十GHz的高频率无线链路可能点对点的，方向性会更强。由于无线数据通信仍处于蓬勃发展的阶段，所以流行的系统包括无线局域网（在100米半径范围内，54Mbps～600Mbps的数据传输速率），通用分组无线电业务（GPRS）（在几公里内为128kbps）、3G（第三代移动通信，在几公里内达到384kbps到数Mbps），蓝牙（在10米范围以内，数Mbps），所有这些技术都工作在800MHz～3GHz的微波频谱内。

历史演变：链路标准

目前有很多链路标准用于数据通信，我们可将链路分成以下几类：本地、最后一公里和租用线路。表1-1列出了这些链路标准的名称和数据率。本地链路非常在局域网中使用，其中5类（Cat-5）以太网和2.4GHz无线局域网是两种主要技术。前者速度更快，并在5类双绞线上运行专用的传输标准，但后者设置简单，并支持更高的移动性。

所谓的最后一公里或第一公里链路连接从家庭或移动用户到互联网服务提供商（ISP）的"第一公里"。其中，非对称数字用户线（ADSL）、有线电视（CATV）和光纤到楼（FTTB）是最流行的有线链路技术，3G和全球微波互联接入（WiMAX）是目前最流行的无线技术。POTS和综合服务数字网（ISDN）是过时的技术。

表 1-1 流行的有线和无线链路技术

	有线	无线
本地	5 类双绞线以太网（10Mbps～1Gbps）	2.4GHz 频带 WLAN（2～54Mbps～600Mbps）
最后一公里	POTS（28.8～56kbps） ISDN（64～128kbps） ADSL（16kbps～55.2Mbps） CATV（50Mbps） FTTB（10Mbps～）	GPRS（128kbps） 3G（384kbps 到数 Mbps） WiMAX（40Mbps）
租用线路	T1（1.544Mbps） T3（44.736Mbps） OC-1（51.840Mbps） OC-3（155.250Mbps） OC-12（622.080Mbps） OC-24（1.244160Gbps） OC-48（2.488320Gbps） OC-192（9.953280Gbps） OC-768（39.813120Gbps）	

对于有线技术，FTTB 比其他技术更快，但也更贵。ADSL 利用传统的电话线，并随着到 ISP 距离的增加，传输速率会降低。CATV 租用电视网轴电缆，它对距离的限制较少，但需要与电视节目信号共享带宽。如果你需要一条不通过公共共享网络的站点到站点的连接，既可以从运营商处租用一条专用线路。在此处，从运营商处租用的线路服务包括：基于铜线的数字信号 1（DS1，T1）和 DS3（T3）。各种光 STS-x（同步传输信号，OC-x［光载波］）链路。后一个选项虽然价格昂贵，但正变得越来越流行，因为它能够满足日益增加的带宽需求。

路径：路由式还是交换式

任何时候要将两个远程节点连接起来就必须首先找到一条路径，即一系列联系起来的中间链路和节点。路径既可以是路由式的也可以是交换式的。当节点 A 要向节点 B 发送消息时，如果它们可以通过非预先设定的和独立选择的路径转发，也许经过不同的路径，那么就是对消息进行了路由。使用路由时，将消息的目的地地址与“路由”表进行匹配以便找到到达目的地的输出链路。这种匹配过程通常需要多次表查找操作，每次查找都需要进行一次内存访问和一次地址比较。另一方面，交换式路径需要中间节点建立路径并且在发送消息之前在“交换”表中记录路径的状态信息。然后将要发送的消息附加一个索引号，指向存储在“交换”表中的某些特定状态信息。交换消息会容易地经过一次内存访问就能索引到表中。因此，交换比路由快很多，但是要以建立连接的额外开销为代价。

我们可以将路由式路径看成是中间链路和节点的无状态的或无连接的连接，而将交换式链路看成是有状态的或面向连接的连接。ATM 的所有连接都是交换式的，也就是说，在开始传输数据之前，需要在源和目的地之间建立一条路径并记住路径上的所有中间节点。与此相反，因特网是无状态的、无连接的，将在 1.3 节中讨论这种无连接的设计理念。

历史演变：ATM 的衰落

ATM 曾经被认为是数据通信的骨干交换技术。与互联网架构不同，ATM 采用 POTS 的有状态的交换概念：其交换保持面向连接的状态信息以便决定如何交换连接。由于 ATM 是在 20 世纪 90 年代初出现的，所以它就必须找到一种与在当时占据主导的网络技术——与联网架构并存的方式。然而，将面向连接的交换与无连接的路由技术杂成，会产生大量的开销。将这两者集成既可以采用将 ATM 区域与互联网区域连接起来的方式，也可以使用 ATM 承载互联网分组的分层混合模式。这两者都需要找到现有的 ATM 连接或建立新的 ATM 连接，但在发送少数分组后就要拆除新建立的 ATM 连接。而且，分层混合方法都暴露出了互联网架构的无状态性质。这就注定了 ATM 要退出市场，只不过是时间早晚的问题。

1.1.2 可扩展性：节点的数量

能够连接10个节点与能够连接数百万个节点是完全不同的。由于能够在小型工作不一定主能够在更大的群组中工作，所以我们需要一种可扩展的方法来实现连接性。因此，从可扩展性方面来讲，计算机网络必须能够提供"一种可扩展到大量节点的平台，以便能够让每个节点知道如何到达任何其他节点。"

节点的层次化

将大量节点连接起来的一个直接方法就是将它们组织成很多组，其中每个组由少数节点组成。如果组的数量非常庞大，那么可以进一步将这些组聚合成大量的超组。如果有必要，还可以进一步聚合成"超超组"。这种逐层聚类方法可以创建一种可管理的树状层次化结构，每个组（或超组、"超级超组"等）仅与少数其他组连接。如果不应用这种条件，那么大量节点的互联网络可能看起来像一个混乱的网络。图1-1说明了40多亿个节点可以组织连接成一种简单的三层层次化结构，在底部和中间有256个分支，在顶层有65 536个分支。正如我们将在1.3节中会看到，互联网采用了类似的聚类方法，分别将组和超组称为子网和域。

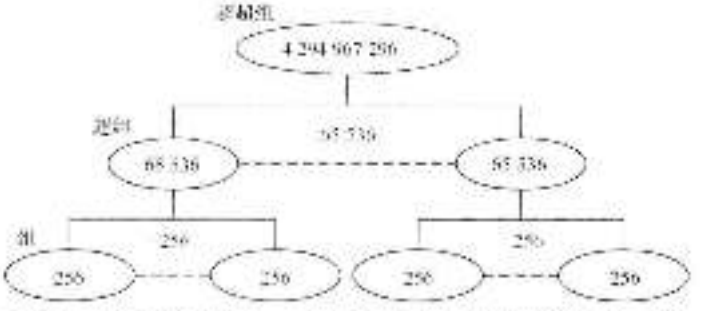


图1-1 节点的层次结构：三层层次化结构中的数亿个节点的分组

局域网、城域网、广域网

很自然地会利用位于小范围地理区域（比如，几公里范围）内的节点形成底层组。将小的底层组连接起来的网络称为局域网络（LAN）。对于一个大小为256的组，将需要至少256条（对于环形网络）、最多32 640条点对点的链路（全互联网络）来建立连接。由于在如此小范围内管理如此多链路将是很麻烦的，所以主要采用广播链路。通过将所有256个节点连接到一条广播链路（使用总线、环形或星形拓扑结构）上，就很容易实现在管理它们的连通性。一条广播链路的应用可以扩展到地理范围更大的网络，如城域网（MAN），以便连接远程的节点或者其全局域网。城域网通常有一个环形拓扑，从而构建一条能够实现链路故障容错的双总线。

然而，这样的广播环形安排对容许在网络上能够支持的节点或局域网数量都出了限制。点对点链路本身适用于无限制的广域连接。广域网（WAN）通常使用网状拓扑，这是由于地理上分散的网络端点位置的随机性所导致的。树形拓扑结构用在广域网中是低效的，因为在树形网络中，所有的流量必须上传到根部，并在某些分支处下传到达目的地节点。如果两个叶子节点之间的流量巨大，那么树形网络就额外需要一条点对点链路与接连接，这样就会在拓扑中造成循环从而将树状变成网状。

在图1-1中，在默认情况下，一个底层组是利用集线器或交换机连接少于256台主机的局域网来实现。一个中层组可能通过路由器将少于256个局域网互连成树状或网状结构的校园网或企业网。在顶层，可能是通过成千上万的层组以点对点链路连接而成的网状广域网。

1.1.3 资源共享

建立过可扩展的连接后，接下来我们讨论如何与网络用户共享这条连接，也就是共享链路和节点的容量。而且，从资源共享的角度，我们可以将计算机网络定义成"一种共享平台，使用节点和链路的容量在节点间传送通信流量。"这正是数据通信和传统语音通信之间最大的不同。

分组交换与电路交换

在 POTS 中，在开始语音对话之前，必须首先找到主叫方和被叫方之间的电路并进行交换。在整个会话过程中，需要为会话方之间维持 64kbps 电路，即使双方一直保持沉默也是如此。这种专用的资源分配称为电路交换技术，它能提供稳定的资源供应，从而可以在连续的数据流（如视频或音频信号）中维持高质量。然而，电路交换不适用于交互或文件传输应用，因为它们随时需要输入数据，但在大部分时间会闲置。显然，为这种突发流量分配一个专用电路是非常低效的。

一种更加宽松和高效的资源共享实践是让所有的流量竞争通行权。然而，这样就不可避免由于突发数据流量造成拥塞。那么，我们如何处理这种流量拥塞？可以通过排队来解决！在节点设置缓冲区空间，就能够化解由于临时数据突发所造成的拥塞，但如果拥塞持续了很长一段时间，最终会由于缓冲区溢出而产生丢弃。这种寻常转变的资源共享称为分组交换或数据报交换，将数据流量中的消息分成了分组或数据报。存储在路径上的每一个中间节点的缓冲区队列中，并沿着路径向目的地转发。

POTS 运用电路交换，而互联网和 ATM 运用分组交换。正如 1.1.1 节中所述，ATM 的路径是“交换式”的，而互联网的路径是“路由式”的。因此可能使读者感到困惑，互联网在分组“交换式”网络中具有“路由式”路径。不幸的是，这个社区不是按照名字来区分这些网络技术的。为了精确起见，互联网使用分组路由，而 ATM 和 POTS 分别使用分组交换和电路交换。从某种意义上说，ATM 模仿带有连接独立的电路交换以便获得更好的通信质量。

打包

为了发送消息，就必须给消息附加一些头部信息以便形成一个分组，以便让网络知道如何处理它。消息本身称为分组的有效载荷。头部信息通常包含源地址和目的地址以及用以控制分组发送处理的许多字段。但是分组和有效载荷可以有多大？这取决于底层所使用的链路技术。我们将在 2.4 节中学习到，可条链接对分组长度都有限制，这就可能导致发送节点将消息分成更小的分段，并在每一分段上附加头部以便能够通过链路传输，如图 1-2 所示。分组头部告诉中间节点如何发送以及目的节点如何重组分组。利用头部，每个分组在穿越网络时既可以完全独立地处理也可以半独立地处理。

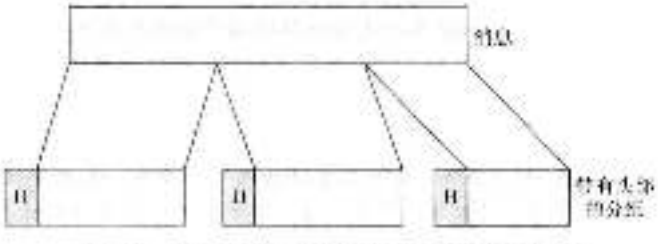


图 1-2 打包：将消息分段为添加了头部的分组

协议定义并标准化头部字段。根据定义，协议是一组用于经过通信信道发送信息所需要的数据表示、信令、错误检测的标准规则。这些标准规则定义了协议消息的头部字段以及接收端对接收到的协议消息应该做出如何反应。正如我们将在 1.3 节中学习到的，一条消息分段可能用多层头部对象，每层头部描述一组协议参数，并添加到上层头部之前。

排队

正如前面所述，网络节点分配缓冲区队列以便吸收由于突发数据流量所造成的拥塞。因此，当分组到达一个节点时，它就加入到带有其他已经到达分组的缓冲区队列中，等待被节点中的处理器处理。一旦分组移动到队列的前面，它就会被处理器处理，具体根据头部字段决定如何处理分组。如果节点处理器决定将分组转发给另一个数据传输端口，那么分组就会加入到另一个缓冲区队列中等待被端口的发送机发送。当分组在一条链路上传输时，它需要一些时间从链路的一端传输到另一端。这既可以用点对点也可以用广播方式进行。如果分组要通过具有 10 个节点的路径（即具有 10 条链路），那么上述过程将重复 10 次。

图 1-3 说明了在一个节点和该节点的输出链路上的排队过程，可将它们建模为带有一个队列和一台服务器的排队系统。在节点中的服务器通常就是一个处理器或一组 ASIC，其服务时间取决于节

点模块（例如，CPU、内存、ASIC）的时钟速率。另一方面，一条链路上的服务时间实际上要取决于：1）传输时间，这取决于收发机（发射机和接收机）能够多快地发送和接收数据以及分组有多大；2）传播时间，这取决于发送信号需要传播多长时间。节点上的前一阶段仅有一台服务器来处理分组，这个阶段分组发送所需要的时间可通过使用更快的收发器来减少。然而，在链路的后一阶段具有很多并行服务器（等于链路上允许的超常未完成分组的最大数），无论采用什么技术该阶段所消耗的时间都不能减少。信号以大约 2×10^{8} m/s 的速度在链路上传播。总之，节点的处理时间和传输时间，包括它们的排队时间，会随着新技术的出现可以进一步减少，但是传播时间将保持不变，因为它的值将受光速的限制。

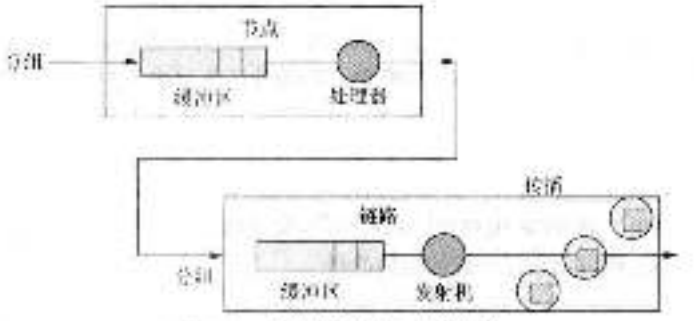


图1-3 在节点和链路上的排队

行动原则：数据通信与电信

这里还要再次强调一下数据通信（datacom），也就是数据通信或计算机网络，与电信（telecom）之间的区别，以便结束对计算机网络需求的讨论。对于连通性、可扩展性和资源共享，上述两者之间在扩展性方面没有太大区别，主要区别在于它们使用的连接类型以及它们进行资源共享的方式。传统的电信在通信双方之间仅建立一种类型的连接，支持单一的应用程序（电话）。另一方面，数据通信上则有着广泛的应用，要求不同类型的连接。连通性可以设置在两个客户端之间（如电话）、客户端和服务器进程之间（如文件下载或读媒体）、两个服务器进程之间（如邮件中继或内容更新），甚至也可以在一组个人或进程之间。每种应用可能有一个独特的流量特征，既可以是突发性的也可以是连续性的，与均匀的、通常连续的、具有很高使用效率的电路交换技术的电信流量不同，数据通信流量要求分组交换，资源共享。然而，与无缝冲区的电路交换相比（其中唯一主要关注的是呼叫阻塞或排队概率），分组交换技术引入了更加复杂的性能问题。正如我们将在1.2节中所讨论的，数据通信需要控制缓冲区溢出或丢失、吞吐量、延迟和延迟变化。

1.2 基本原理

作为数据通信的基本技术，分组交换给出了数据通信应遵循的原理。我们可以将这些原理分成三类：性能，控制着分组交换的服务质量；操作，详细描述分组处理需要的机制类型；互操作性，定义需要将哪些内容制定成标准协议和算法，哪些内容不行。

1.2.1 性能测量

在本节中，我们介绍基本的背景知识，以便使读者可以理解分组交换的规则。当分析整个系统的行为或某个特定的协议实体时，背景知识非常重要。不预先知道或事后才知道系统或协议在一般或极端的运行场景下的性能测量就去设计，实现是不可取的。系统性能测量结果既可以来自实际系统实现之前的数学分析或系统模拟分析，也可以在系统实现之后来自实验台的实验。

用户考虑一个系统如何实现可要取决于三件事：1）系统的硬件容量；2）对该系统的提供负载或输入需量；3）内置到系统中处理提供负载的内部机制或算法。具有高容量但机制设计不当的系统在处理重提供负载时不能很好地扩展，即使它能很好地处理轻提供负载。但是，具有良好设计的小容量系

统，不应放置在具有重荷量的位置。硬件容量通常称为带宽，它是网络领域中的一个常用术语，无论是节点、链路、路径，甚至是作为一个整体的网络都会用到。一个系统的提供负载是可变的，可以是轻负载、正常运行负载、极重负荷（即线速压力负载）。如果系统将维持稳定的操作，同时允许设计的内部机制提供用以获得更高的性能，那么带宽和提供负载之间就应该很好匹配。对于分组交换，吞吐量（输出流量与输入流量的提供负载相比较）似乎就是我们最关心的性能测量，虽然其他测量，如延迟、延迟变化（通常称为抖动）丢失也很重要。

带宽、提供负载和吞吐量

术语“带宽”来自电路领域的研究，最初是指用于传输数据的频带的宽度。然而，在计算机网络中这术语通常用来描述系统在某一时间内可以处理的最大数据量，系统可能是节点、链路、路径或网络。例如，一个 ASIC 能够以 100Mbps 加密，收发器能够以 10Mbps 传输，一条由 5 个 100Mbps 节点和 5 条 10Mbps 链路组成的端到端路径能够以高达 10Mbps 的速度处理。前提是着这条路径没有其他流量干扰。

人们可能会将链路的带宽想象成 1 秒内信号传播的距离中所传送或包含的位数。由于介质中的光速固定为大约 2×10^8m/s，所以更高的带宽意味着在 2×10^8m 内包含更多的位。对于一条 9600km 长的洲际链路（传播延迟为 9600km/（2×10^8m）=48ms），带宽为 10Gbps，在链路中包含的最大位数为 9600km/（2×10^8m）×10Gbps=480Mb。同样，在一条链路上的发送位的“宽度”会根据链路带宽而改变。如图 1-4 所示，在 10Mbps 链路上的位宽度在时间上为 1/（10×10^6）=0.1μs，或在长度上为 0.1μs×2×10^8m/s=20m。这种情况下，一位的信号波实际在光路上占用 20m。

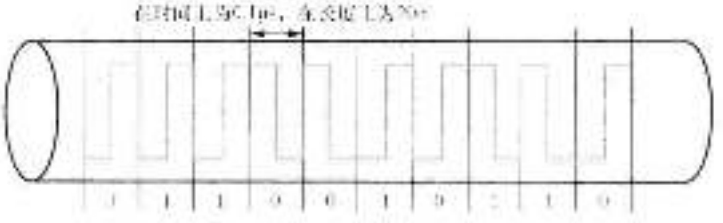


图 1-4 10Mbps 链路在时间和长度的位宽度，这里发送数据采用了曼彻斯特编码

提供负载或输入流量可以针对带宽进行归一化并用来指示系统利用率或者有多忙。对于一条 10Mbps 链路，5Mbps 的提供负载意味着归一化负载为 0.5，这说明链路平均 50%忙。归一化负载可能超过 1，尽管这将使系统处于不稳定的状态。吞吐量或输出流量可等于也可不等于提供负载，如图 1-5 所示。在理想情况下，在达到提供负载带宽（见曲线 A）之前它们应该是相同的（见曲线 A）。此外，吞吐量收敛于带宽。但在实际中，即使在提供负载（在广播链路）达到带宽之前，由于缓冲区溢出（在节点或链路）或冲突，吞吐量也可能低于提供负载（见曲线 B）。在具有冲突失控的链路上，随着提供负载不断增加，吞吐量可能会下降到零，如图 1-5 中曲线 C 所示。通过精心设计，我们可以防止这种情况发生，使吞吐量收敛于一个低于带宽的值。

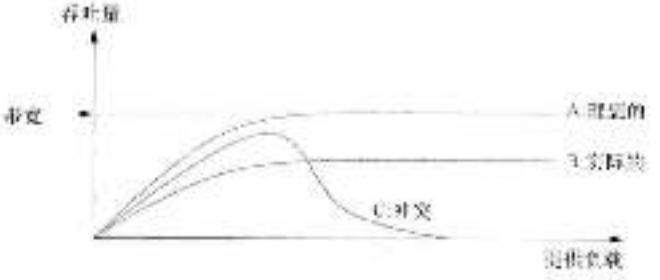


图 1-5 带宽、提供负载和吞吐量

延迟：节点、链路、路径

除了吞吐量外，延迟是另一个我们所关心的重要测量。最早由 Agner Krarup Erlang 于 1909 年在

1917年提出的排队理论能够告诉我们，如果分组到达间隔时间和分组服务时间都服从指数分布并且前者大于后者，再加上无限的缓冲区大小，那么平均延迟则与带宽和提供负载之差呈反比，即

$$T = 1/(\mu - \lambda)$$

其中μ是带宽，λ为提供负载，T是平均延迟。虽然在实际中指数分布并不能反映实际的网络流量，但这个方程为我们提供了一种带宽和提供负载、延迟之间的基本关系。从方程中可以看出，如果带宽和提供负载加倍，延迟将减半，这意味着更大的系统通常有更低的延迟。换句话说，从延迟的观点来看资源不应该分割成小块。此外，如果系统分割成相同的两个小系统来处理均分的负载，那么整个系统的延迟将会增加一倍。

行动原则：Little 结论

对于一个节点来讲，一个有趣的问题是如果我们能够测量其提供负载和延迟，那么在一个节点中到底能够包含多少个分组？由 John Little 在1961年开发的定理对此做出了回答：如果吞吐量等于提供负载，即无分组丢失，那么平均占用率（节点中的平均分组数）等于平均吞吐量乘以平均延迟。也就是说，

$$N = \lambda \times T$$

其中λ是平均提供负载，T是平均延迟，N是平均占用率。Little 的结论非常的强大，因为它不需对这些变量的分布做出假定。这一结论的一个很有用的应用是用来估计配置于节点中的缓冲区大小。假设我们可以测量节点的最大无丢失的吞吐量和在这样的吞吐量下的延迟。通过它们的乘积得到的占用率近似等于节点内所需要的最小缓冲区大小。在图1-6中，显示了没有丢失发生时对占用率的估计。

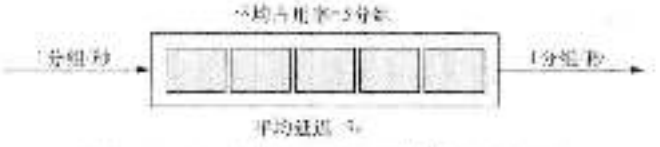


图1-6 Little 的结论：在盒子中有多少个分组

一个数据包的延迟（latency）实际上是排队时间（queuing）和服务时间的总和。后者对提供负载相对不敏感，但前者对提供负载非常敏感。一个节点的服务时间通常是处理数据包所花费的 CPU 时间（processing）。另一方面，在一条链路上的服务时间由传输时间（transmission）和传播时间（propagation）组成。也就是说，在一个节点上，

$$\text{latency} = \text{queuing} + \text{processing}$$

但在一条链路上，

$$\text{latency} = \text{queuing} + \text{transmission} + \text{propagation}$$

与将 Little 结论用于节点上相似，链路的带宽延迟乘积（BDP）能够说明在管道中有多少位正在传输。图1-7中对比了包含在长而粗的管道（链接）中的二进制位数与包含在短而细的管道中的位数。这里用L表示的延迟就是传播时间，而不是传送或排队时间，它由链路的长度来决定。BDP 是设计流量控制机制中的一个重要因素。具有大 BDP 的链路或路径应该采用预防控制机制而不是反应控制机制，因为后者会对拥塞做出太迟的反应。

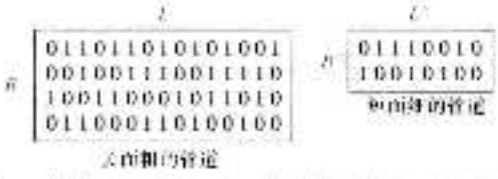


图1-7 带宽延迟乘积：长而粗的管道与短而细的管道

抖动或延迟变化

在数据通信中的某些应用中（例如，分组语音），不仅需要较小的而且还需要一致的延迟。其他

一些应用（如视频和音频流），可能会容忍非常高的延迟，甚至在某种程度上可以吸收延迟变化或抖动。因为流媒体服务器向客户端发送单向连续的流量，人们感觉到的播放质量就已经足够，只要在客户端的播放缓冲区不为空或溢出就行。这种客户端使用一种播放缓冲区，通过将所有分组的播放时间延迟到某一对齐的时间线上来吸收抖动。例如，如果抖动是2s，那么客户端会自动地将所有数据分组的播出时间延迟为播出时间戳增加2s。因此，就必须有一个能够将分组排队2s的缓冲区。尽管增大了延迟，但抖动也被吸收或减少了。对于分组语音，就根本不能采用这种抖动消除法，因为需要在两个节点之间进行互动。在这种情况下，就不能牺牲太多延迟来消除抖动。然而，对于非连续性流量，抖动就不再是一个重要的测量指标。

丢失

最后但并非是虽不重要的性能测量是分组丢失概率。有两个主要原因导致分组丢失：拥塞和错误。数据通信系统容易出现拥塞。当在链路或节点上发生拥塞时，分组就会在缓冲区中排队，以便吸收拥塞。但是，如果拥塞持续存在，那么缓冲区就开始溢出。假设某个节点有三条相同带宽的链路，当同时有从链路1和链路2进入的线速流量发向链路3时，节点将至少有50%的分组丢失。对于这种速度不匹配，缓冲区就起不到作用，就必须使用某种控制机制来替代。缓冲仅用于短期拥塞的解决。

在链路或节点上发生的错误，也会导致分组丢失。尽管目前许多有线链路传输具有好的传输质量，非常低的误码率，但是由于干扰和信号衰减，大多数无线链路仍然具有高的误码率。一位错误或多位错误会使整个分组没有用途而被丢弃。传输并非是错误的唯一来源，任意节点上的内存错误也占了相当大的比例，尤其是当内存模块使用了很多年后更是如此。当数据分组在节点缓冲区中排队时，缓冲区内存就可能发生位错误，导致读取的字节与写入的字节不一致。

1.2.2 控制平面上的操作

控制平面与数据平面

分组交换网络的操作涉及两种分组的处理：控制和数据。控制分组携带的消息用来指导节点如何转发数据，而数据分组则包括用户或应用程序实际上要转发的消息。一组用于处理控制分组的操作称为控制平面，而一组用于处理数据分组的操作称为数据平面。尽管还有一些用于管理目的的其他操作，因此而称为管理平面，但为了简单这里将它们合并到控制平面中。控制平面和数据平面之间的主要区别在于前者通常发生在具有更长的时间尺度（如几百毫秒（ms）至几十秒（s））的背景下，而后者则发生在前台具有较短的时间尺度（如介于微秒（μs）至纳秒（ns））并且更加实时。控制平面的每个操作往往需要更复杂的计算，以便决策如何路由流量以及如何分配资源，以便优化资源共享和使用。另一方面，数据平面需要在运行过程中处理和转发数据分组，以便优化吞吐量、延迟、丢失。本节说明控制平面需要什么样的机制，1.2.2节介绍数据平面所需要的机制。这里也给出了它们的设计考虑。

控制平面在数据通信中的任务是为数据平面提供良好的指令以便承载数据分组。如图1-8中所示，为了实现这一任务，中间设备的控制平面就需要计算出哪里路由分组（到哪条链路或端口），通常需要控制分组的交换和复杂路由的计算。此外，控制平面可能还需要处理其他问题，如错误报告、系统配置和管理以及资源的分配。无论这一任务能否完成好，一般都不直接影响性能测量，这一点与数据平面不同。相反，控制平面更多关注资源是否已经有效、公平、优化地使用。接下来让我们学习到底能够在控制平面中放入什么样的机制。



图1-8 中间节点上控制平面和数据平面上的某些操作

路由

大多数文献中不区分路由与转发。这里，我们将路由定义为找到向何处发送分组，而将转发定义为发送分组。因此路由是计算路线并将它们存储在转发分组时需要查找的表中。路由通常是在后台周期性地完成的，以便于维护和更新转发表。（请注意，在许多文献中将转发表称为路由表。在本书中我们交替使用两个术语来表示同样的含义。）当分组到达时再计算路由就来不及了，需要将分组立即转发。只给查表的时间，但却没有给出运行路由计算算法的时间。

作为路径计算的路由并不像人们乍看起来所认为的那么简单。在人们开始设计一种路由选择算法之前有很多问题需要回答。

> 路由应当在每台中间路由器逐跳地决定还是在源主机上计算（也就是源路由）？
>
> 路由选择决策的粒度是：按目的地，按源目的地、按流量，还是更规则地按分组进行呢？
>
> 在给定的粒度下，我们是选择单路径的路由，还是多路径路由呢？
>
> 路由计算是根据全局的还是局部的网络信息呢？
>
> 如何分发全局或局部信息？在所有路由器之间广播还是仅在相邻路由器间交换？
>
> 按照定义何谓最佳路径？它是最短的、最宽的，还是最健壮的路径？
>
> 路由器应当仅支持一对一转发还是一对多转发，也就是说，单播还是组播？

所有这些问题都必须首先考虑清楚。我们强调由互联网所做的设计选择，但是对于其他网络体系结构可能有一套不同的选择。我们不打算专门在此详述这些选择到底是如何在互联网上工作的。在此我们仅列举出路由协议和算法的设计问题，而将详细细节留到第4章中介绍。

流量和带宽分配

可以从一个更加面向性能的角度来考虑路由。如果流量和带宽资源可以被测量和操纵，我们就能够分配一定的流量容量，并将它分配给带有一定带宽的路径上。分配流量还有另外一个类似于路由的标签，即所谓的流量工程。带宽分配和流量工程两者通常都带有特定的优化目标，例如最小化端到端平均延迟和最佳的负载平衡，假定一组令人满意的系统限制。因为这种优化问题需要不能实时完成的非常复杂的计算，还因为仅有少数系统才能够在运行时调整带宽分配，所以流量和带宽分配通常在全球平面或者在网络规划阶段离线地完成。

1.2.3 数据平面上的操作

与在控制平面上可能仅适用于几百毫秒到几十秒时间内控制分组的操作不同，数据平面上的操作适用于所有的分组，并限制在数微秒或更短的时间内进行处理。数据平面的主要工作似乎就是转发分组，因为到达某个端口或路上的分组能够转发到另一个端口。事实上，转发可能仅是数据平面提供的服务之一。其他的服务还包括分组过滤，加密或甚至内容过滤。所有这些服务需要通过检查分组的多个字段在对分组进行分类，检查主要集中在头部，但可能还包括有效载荷，与控制平面维护的规则或由管理员预先设定的规则进行对比。一旦匹配，匹配的规则规定分组应该得到什么样的服务，以及如何运用这些服务。

转发本身并不能保证网络的健康运行。除了转发和其他已经提到的增值服务外，关键控制和流量控制是数据平面上的另外两个基本按分组的操作。前者确保分组完好无位错误地传输，而后者则是为了避免拥塞并保持良好的吞吐量性能。如果没有这两个基本操作，单独的转发会使网络很容易产生拥塞和错误的混乱。下面我们仔细学习图1-8中列出的这些操作。

转发

分组转发需要检查分组中的一个或多个头部字段，具体取决于如何确定在控制平面上的路由。可能仅取出目的地地址字段后就查找转发表，也可能需要取出更多的字段后再查找。路由决策直接决定如何完成转发，包括需要检查哪个头部字段，在转发表中匹配哪些条目等。由此看来，似乎如何进行游戏（转发）已经由在别处的另一个游戏（路由）所决定了，但其实这里的玩家（参与者）仍有很大的自由空间。对于分组转发，可能最重要的问题就是需要以多快的速度来转发分组。假设某个路由器

节点有 4 条链路，每条容量为 10Gbps，而且分组很短，固定为 64 字节。在路由器上每秒聚集的最大分组量将达到 $4 \times 10G/(64 \times 8) = 78\,125\,000$，这就意味着如果需要以线速转发，路由器将需要每秒转发 78 125 000 个分组（大约每 12.8ns 就转发一个分组）。这无疑对转发机制的设计构成了挑战。

如何实现转发表的数据结构，以及在这个数据结构上如何实现查找和更新等操作就摆在设计者面前。这些设计决定了节点是否能够以线速转发。在某些情况下，可能需要使用特定的 ASIC 以便从 CPU 上卸载转发任务，从而能够达到每秒数百万分组的转发速度。尽管速度最关键，但也与大小有关。存储转发表的数据结构可能会受到限制。对于每个表项大小为 2～3 字节的 80 000 个转发表项，可以试着将它们存储到不超过数百 KB 的树或数列表中，也可以存储到数百 MB 的平面索引表中。在转发表的实现上，总是需要在时间复杂度和空间复杂度之间进行折中。

分类

如前所述，许多服务需要对分组进行分类操作，即一种从分组头部中检查一个或多个字段与一套规则进行匹配。规则由两部分组成：条件和行动，指定在字段的什么条件下应该对匹配的数据分组采取何种行动。由于每种服务都有其自己的一组字段与其自己的一套规则进行匹配，所以对于一种特定的服务就需要一种分类器及其相关规则，或一种分类数据库。对于转发服务来说，转发表就是它的分类数据库。

一个与高速多核处理转发分组的问题相类似的就是需要多快速度分类分组。这里的速度取决于：字段的数量（一个整数个）和规则的数量（从数个到几万个）。这两个数值都直接影响分类器在吞吐量的扩展性。因此，目标是设计一种多字段的、可以良好扩展的、有一定数量的字段和规则的分类算法。如果当这两个数量较小时就具有高吞吐量，但当其中一个数字比较大而使吞吐量下降很多时，设计就具有较小的可扩展性。与转发相类似，设计者可能会采用 ASIC 硬件设计以取得高吞吐量的分组分类。

深度分组检验

转发与分类都检查分组头部字段。更有甚，恶意代码常常隐藏在分组的有效载荷中。例如，入侵和病毒分析会深藏在应用程序的头部和有效载荷中。有关这些内容的知识通常抽象成一种签名数据库，用来匹配进入分组的有效载荷。这种匹配的过程称为深度分组检查（DPI），因为它深入查看有效载荷。由于签名通常是用简单的字符串或正则表达式来表示，所以字符串匹配就成为深度分组检验的关键操作。

而且，能够多快地执行字符串匹配也是主要的考虑。与 1 维的转发和 2 维的分类相比，这是一种以签名数量、签名长度、签名字符串的字母集的大小为参数的 3 维问题。在这个大问题空间中设计一种向上、向下很好扩展的算法将更具有挑战性。毕竟，它也是一个为了获得高吞吐量需要 ASIC 硬件解决方案的难解的设计问题。

差错控制

正如 1.2.1 节中的讨论，在分组中可能发生位错误。在分组传输期间或分组存储在内存中时，就可能发生错误。需要回答两个基本问题：1）检测或纠正？2）在链或端到端？第一个问题是有关错误分组的接收器是如何检测和处理错误的。有两种方法：接收器可以通过额外的冗余位检测错误，并通知发送者重发；或者如果有额外的冗余位能够指示错误的准确位置，就可以直接用检测并纠正错误。后一种方法将需要更多的冗余位，从而产生更高的额外开销。是否进行错误纠正取决于承载的流量类型。对于实时流量，通知发送方重传不是一个很好的方法。如果应用程序能够容忍很小比例的丢失，那么就可以直接丢弃错误分组而不采取进一步的行动；否则就应该进行纠错。

第二个问题就是有关错误可能出现的位置：链路或节点？如果位错误仅出现在链路上，那么就可以在每条链路的接收器设置差错控制进行检测或者也可以纠错。这样路径将是无差错的，因为所有的链路都可以恢复错误。但是，如果存储在节点中的分组出现内存错误，那么位错误就会继续传播而不被检测到，因为每条链路的发送端和接收端只能发现链路上的传输错误。换句话说，逐段逐跳（或逐链路）的差错控制是不够的，还需要端到端的差错控制。或许有人会问：为什么不去掉逐跳差错控制而仅保留端到端的差错控制呢？从差错控制的角度来看，是可以这样做的。但问题在于恢复错误所需要的时间：去掉逐跳差错控制将会延长差错恢复处理所需要的时间。如果链路位错误率太高，这样做甚至还会增加恢复错误的难度，因为端到端错误恢复的成功概率是沿着路径上每条链路的错误恢复成

功概率的乘积。这实际上就是将在1.3节中详述的端到端参数。

流量控制

在数据平面上的另一个按分组操作就是控制分组流的输入过程。输送分组过快可能会让中间路由器或目标节点溢出，导致加剧拥塞的许多分组的重传。输送分组太慢则可能会使缓冲区下溢，导致较低的带宽资源利用率。流量控制是任何为了避免或解决拥塞机制的统称，但拥塞本身可能会相当复杂。它可能是端到端（在一条路径上的源和目的地之间）、逐跳（链路上的发送端和接收端之间），或热点（瓶颈节点或链路）的现象。流量控制是一种通信流量控制，它通过控制发送端与接收端之间的同步，防止速度快的发送端压垮较慢的接收端。发送端和接收端之间可以通过一条链路或一条路径连接起来，因此流量控制既可以是逐跳的也可以是端到端的。

作为另外一种流量控制，拥塞控制处理由一组流量源所造成的、更复杂的瓶颈拥塞。一个瓶颈，无论是节点还是链路，能让许多分组流量通过，每一个分组流都会对拥塞做出贡献。让源放慢甚至停止发送是一种显而易见的解决方案。不过，还有些细节有待解决。谁应该放慢以及放慢多少呢？处理策略是什么？我们可以让全部或部分源降低相同或不同数量的传输速率，但这应该由基本策略来决定。公平似乎是一个良好的策略，但如何定义公平以及如何以一种有效的方式执行公平策略，不同的网络体系结构会做出不同的设计选择。

服务质量

利用流量控制和拥塞控制来维持称心如意的的操作，网络就能很好地工作。但是存在显式地指定流量参数，如速度和突发长度以及其期望的性能测量（如延迟和丢失）等更加严格的需求，这就是显式的服务质量（QoS）。几十年来，它都是分组交换的一种！（大挑战！可能在入口点或网络的核心放置各种信息控制模块（如策略器、整形器、调度器），来调节流量以满足QoS的目标。虽然已经提出了多种解决方案架构，但它们都还没有大规模地部署到运营网络中。第7章中详述它们的发展。然而，许多信息控制模块已经嵌入到各种设备中作为部分QoS解决方案。

1.2.4 互操作性

标准与实现相关

让各种设备相互通信有两种可能的方式。一种就是只从同一家供应商购买所有的设备。另一种就是在设备之间定义标准协议，这样只要供应商遵循这些协议，我们就可以交互操作从不同供应商购买的设备。这种互操作性是必需的，尤其是我们不希望从一家供应商那里购买了首批设备后就被绑定到这家特定的供应商。另一方面，主宰市场的厂商可能想把一些专有协议，即厂商自己的而不是标准化组织的定义放入到他们的设备里，以此来约束他们的客户。但是，如果他们这样做得不够谨慎，那么他们的市场份额因此可能会悄无声息地下滑。

那么，哪些内容应该定义成标准，哪些内容不必呢？由互操作性充当标准。对于分组处理过程，某些部分需要标准化，而其余的部分可能要留给厂商来决定。需要标准化的部分就是影响来自不同供应商设备互操作性的部分。协议消息的格式，当然需要标准化。然而，许多不影响其他设备互操作性的内部机制（例如，表的数据结构及其查找和更新算法），是与实现相关的（特定厂商的），通常就是这些厂商的具体设计导致最终性能上的差异。本节指出在哪里可以实现标准和与实现相关的设计。

标准协议和算法

在默认情况下，协议应该标准化，尽管也确实存在一些专有协议。这种专有协议如果占领了市场，就有可能成为事实上的标准。除了体系结构外，在定义协议规范时，还需要定义两个接口：对等接口和服务接口。对等接口格式化支持该协议的系统之间交换的协议消息，而服务接口定义同一节点机器上其他模块的函数调用以便访问由该模块所提供的服务。一个协议可能有多种消息类型，每种都有它自己的头部格式。一个头部包含多个固定或可变长度的字段。当然，每个头部字段的句法（格式）、语义（解释）都需要标准化。发送方为了协议握手将信息编码为协议消息的头部，如果有数据，就将它添加为该协议消息的有效载荷。

控制协议在协议消息头部放置用于控制平面操作的控制数据。另一方面，数据协议将各种数据，

不管是用户数据还是控制数据都放置在其协议消息的有效载荷中，它们的头部信息仅告诉设备应该如何转发分组。

除了协议消息的句法和语义外，在控制平面和数据平面上的某些算法也应该是标准化的。例如，如果要规让所有参与的路由器对最短路径达成一致看法，那么所有参与的路由器对控制平面上的路由算法就必须达成一致。如果两台相邻的路由器，我们不妨称它们为 A 和 B，使用不同的路由算法计算到达目的地 X 的最短路径，那么就有可能将 A 指向 B 作为到达 X 的最短路径的下一跳；对于 B，反之亦然。这将导致发送到 X 的分组在 A、B 之间循环。在数据层的错误检测或校正算法也是同样的例子。如果发送方和接收方对数据编码和解码分别应用不同的算法，那么就无法正常工作。

与实现相关的设计

与协议规范不同，在协议实现中存在很大的灵活性。并非算法在控制平面和数据平面上的每一部分都需要标准化。例如，实现一种路由算法，如 Dijkstra 算法，需要一种数据结构来存储网络的拓扑结构，并用路由算法在该数据结构上查找到达目的地的所有最短路径，但是实现并不需要标准化。人们可以设计出比在教科书中更有效的方法进行计算。另一个例子是分组转发中的表查找算法。设计一种数据结构来存储大量的表项并设计出相应的查找和更新算法，使它们能够在速度和规模方面打败目前的最佳设计，始终都是一个有趣的挑战。

分层协议

实际上，互操作性问题不仅存在于两种系统之间，还存在于两种协议之间。只有一个协议不足以驱动系统。事实上，是协议栈驱动整个系统。协议栈由分层的一组协议组成，其中每层包含部分数据通信机制，并为上层提供服务。将一个复杂的系统拆散成模块化的实体（即就是这里的分层协议）是一种自然的进化，这样就隐藏了低层细节，并还要为它们的上层提供服务。

由于两个系统需要使用相同的协议才能进行通信，所以在不同层次上的协议也需要服务接口（如 send 和 recv）在同一系统中交换数据。当这两个协议之间使用一个公共接口时，系统就具有更多的灵活性，在需要对替代协议栈中的任何协议。例如，两个远程终端主机 X 和 Y 只有一个 A-B-C 的协议栈，其中 A 是上层，C 是特定链路的协议，那么 X 应该能够用 D 取代 C，也就是说用于更可靠链路的协议，同时仍保持 A、B 不变，与 Y 中相应的 A 和 B 进行互操作。然而，因为 X 运行 C 和 Y 运行 D 是在两个单独的链路上，所以还应该有一台中间设备，比如 Z，在 X 和 Y 之间将它们桥接起来。

1.3 互联网体系结构

假定分组交换的约束原则，互联网就有其解决方案，以实现 1.1 节所述的数据通信的三个要求，即连通性、可扩展性、资源共享。为互联网体系结构选用的所有解决方案都有其哲学上的理由。然而，还存在其他数据通信体系结构，如普经风靡一时的异步传输模式（ATM）和新兴的多协议标签交换（MPLS）。与互联网体系结构相比，它们具有共同点和一些特性。当然，它们只有一套哲学思想来证明它们的设计选择。一个特定的解决方案是否流行往往取决于：1）谁最先出现；2）谁能最好地满足三个要求。虽然互联网的首次出现可以追溯到 1969 年，并且已经刚满足达到了要求，尽管一直以来都存在使之经历根本性变化的压力。

本节提示了在互联网体系结构中所采用的关键解决方案。除了无状态路由外，为了解决连通性，端到端观点作为定义流在哪里放置机制，或在网络内和网络外应该做什么的关键哲学理念。在这一观点指导下定义了协议层，然后出现了子网环域的概念，以便支持所需要的可扩展性。作为分组交换中最棘手的问题，通过公共的尽最大努力的无管商服务、互联网协议（IP），再加上两个端到端的服务：传输控制协议（TCP）和用户数据报协议（UDP），就解决了资源共享问题。TCP 提供端到端的拥塞控制机制以共享带宽和一种可靠的无差失服务；UDP 则提供一种简单无控制的和不可靠的服务。

1.3.1 连通性解决方案

两个不相邻的端点通过带有节点和链路的路径来连接。为了决定如何建立和保持规则端连通性的互联网，人们必须做出三项决策：1）路由式或交换式的连接；2）端到端或逐跳机制，以维护连接的

正确性（可靠的和有序的分组分发）；3）在建立和维护这种连通性时如何组织任务。对于互联网，决定采用路由实现这种连通性，在端到端保护它的正确性，并将任务组织成四个协议层。

路由：无状态的和无连接的

如1.1.1节中所讨论的，尽管交换要比路由快，但它需要交换设备记忆所有的状态信息，也就是在虚电路表中从（输入端口，输入虚电路号）到（输出端口，输出虚电路号）的映射。与电信的连续语音通信不同，数据通信通常是突发性的。保持长时间但又非突发性的连接状态信息，从内存利用率来讲是低效的，因为这些状态信息长时间保存在内存中但偶尔才使用。同样，考虑到初始时间足以为预警的连接建立状态信息，仅为了传输几个分组就花费很大的开销也是低效率的。简单地说，对于数据通信，从空间和时间开销权衡，交换要比路由效率更低。

然而，路由并不是在各个方面都占优势。正如1.1.1节中介绍的那样，互联网上的路由将取出分组中完整的目的地地址与转发表（有时也称为路由表）进行对比，这就需要匹配过程能够遍历一个很大的数据结构，因此需要花费多次内存访问和匹配指令。另一方面，交换则提取分组中的虚电路号以便索引到虚电路表中，因此仅需要一次内存访问就够了。

许多网络体系结构，包括ATM、X.25、帧中继和MPLS，都采用交换式。它们都可以看做是来自电信行业的数据通信解决方案，都是从POTS（这当然是一个交换系统）演变而来的。在图1-9中，把所有这些体系结构上置于有状态的频谱上，这里状态不仅意味着存储在节点中的表项，还包括为流或链路预留的带宽。具有上述两种状态的POTS是纯粹的电路交换，而其余的都是分组交换。在分组交换中，互联网和MPLS分别为路由和“软状态”交换，而其余的都是“硬状态”交换。ATM比X.25和帧中继更有状态，因为它为每条连接提供带宽分配。

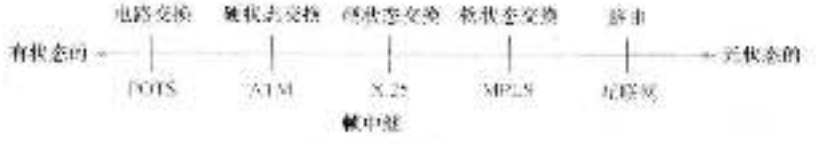


图1-9 状态的频谱

行动原则：不断受到挑战的无状态

有人可能会说，互联网架构最独特的选择是无状态的路由选择。最开始的决策已经将它引向了无状态、无连接的网络，所有分组都独立地发送而没有事先在中间路由器建立一条路径。无状态意味着路由器不保留任何跟踪经过分组流的状态信息。有了无状态路由的简洁性（以及本节其他关键的设计），互联网可以很好地扩展，并为数据通信中的所有应用程序提供灵活的连通性和经济的资源共享。

实际上，围绕互联网是否应该保持纯粹无状态引发了不少争议。事实上，许多新的需求，尤其是那些对服务质量（QoS）和组播的需求，已经制定了许多建议将有状态元素加入到互联网架构中，这一点我们将分别在第4章和第7章中学习到。并非只有QoS和组播能将对架构进行改进。作为另一个迫切需求，由于链路带宽的迅速增加，线速转发就需要分组交换而非路由。MPLS的目的在于通过交换更多的分组而进行更少的路由来加快互联网的速度。如前所述，交换比路由速度快，就是因为前者仅需要简单的索引虚电路表，而后者在查找表期间需要更为复杂的匹配。与硬状态交换的ATM不同，MPLS是软状态交换，这意味着如果一个分组流交换过期或不存在，它可以回过头来使用无状态的路由。是否能够将MPLS大规模地部署到原来的互联网架构上，目前仍处在研究之中，但新的要求（如保证性能的QoS、用于组通信或发布的组播、用于更快体系结构的线速转发）在得到满足之前还会有更新的要求不断地出现。

端到端的观点

为了从源到目的地提供可靠的、有序的分组发送，差错控制和流量控制应该建立在逐跳或端到端的基础上，也就是说，在所有链路上或者仅在终端主机上。逐跳观点就是，如果在所有链路上的传输是可靠的和有序的，那么就能保证端到端传输的可靠性和有序性。然而，这种观点只有当节点没有错误时才会成立。因为路径由节点和链路组成，保证链路操作的正确性并不包括节点操作的正确性，所

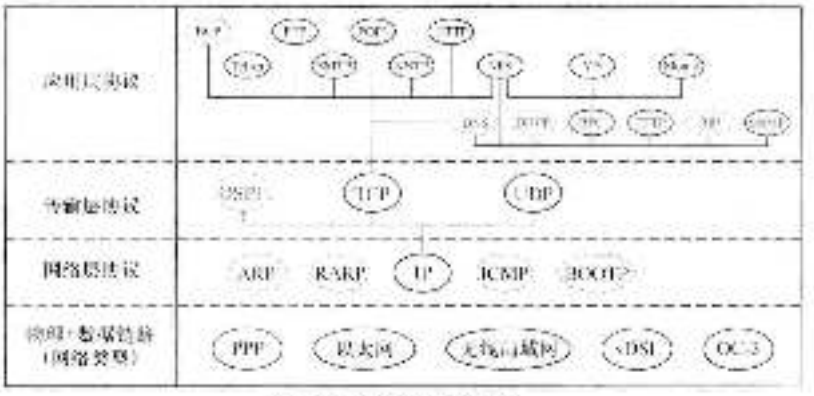
以不能保证链路径的端到端交付。在终端主机上仍然需要差错和流量控制机制来防止节点的错误。端到端的观点就是说，不要将它放在较低的层次，除非它能在比彻底完成，这样才能得以实施。虽然有些逐跳差错和流量控制仍然可以在链路上实现，但它们仅用于此线优化以便及早检测和恢复错误。端到端机制仍然是保证连接性正确的主要选择。

端到端观点也可以将复杂性推向网络的边缘并保持网络核心简单以便很好地扩展。对应用感知服务的处理应该仅在终端主机上完成，是在网络的外部而不是内部，从而仅在网络内部留下单一的运营商服务。我们将在资源共享的解决方案中认识到这一点。

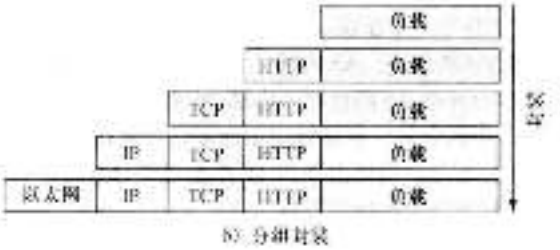
四层协议栈

在设计复杂数据通信系统时的抽象会导致协议层次的划分，下层对上层隐藏细节。但是，到底需要多少层？在每层中放入什么内容？四层互联网体系结构有时也简称为 TCP/IP 协议，这是根据两个代表两层的重要协议而得名。底层是链路层，其中可能由各种链路的许多协议组成。链路层协议依赖于硬件，通过将硬件（适配器卡）和软件（适配器驱动程序）结合起来实现。基于链路层，IP 层包括一个协议（IP），通过无状态路由提供主机到主机的连接（端到端的连接与数据链路层中的逐跳连接）。第三层是传输层，它包含两个协议（TCP 和 UDP）。TCP 和 UDP 提供最上面应用层所需要的进程到进程的连接。传输层将底层网络的拓扑结构细节隐藏到为应用层通信进程抽象的虚拟链路或信道之后。应用层对每一个客户端、服务器或对等应用都有一个协议。

图 1-10a 显示了互联网协议栈及常用的协议。标有虚线圆的协议属于控制平面协议，而其余的属于数据平面协议。注意，TCP、UDP 和 IP 用作核心协议支持很多应用协议，同时包括许多可能存在的链路。我们将在后面的章节中详细介绍图 1-10a 中的重要协议。在这种四层协议栈中的一个例子就是 HTTP、TCP、IP、以太网。数据有效载荷依次封装 HTTP 头部、TCP 头部、IP 头部，然后在传输时封装以太网头部。当接收后则反过来处理，如图 1-10b 所示。



a) 互联网协议栈：常用协议



b) 分组封装

图 1-10

1.3.2 可扩展性解决方案

如何将大量节点集聚起来决定了系统具有多大的扩展性。寻址这些节点就成为关键问题。图 1-1 说明了一种以三层结构组织 40 亿个节点的方法。但我们如何寻址和组织这些节点呢？将互联网扩展到

40亿台主机作为一个设计目标，就必须回答三个基本设计问题：1）层次化结构有多少层；2）每个层次有多少个实体；3）如何管理这个层次化结构。如果节点分组没有一层，组的大小为256，那么组的个数就是16 777 216，这对于互连的路由器来讲就太大而不能处理。这些路由器必须知道如此大量的组。如图1-1所示，如果添加另一个层次并且超组的大小也是256，那么超组内的组数量和超组数量将分别是256和65 536。对于网络运营商（一个组织或一个ISP）来讲，256个可管理的，而对于核心路由器来讲65 536是可以接受的大小。因此，互联网采用三层结构，子网作为最底层，自治系统（AS）作为中间层，而将许多AS组在顶层。

子网

互联网利用子网来表示物理网络中具有连续地址块的节点。一个物理网络由链路（点对点或广播）和与它相连的节点组成。广播链路上的子网形成了一个局域网，这就是一个广播域。主机是说，发送局域网的主机的分组可以被这个局域网上的任何主机或路由器转发，并被一跳以内的目的地主机自动接收。然而，在子网或局域网之间传输的分组需要由路由器逐跳转发。在点对点链路上的子网通常形成两台路由器之间的一条广域网链路。图1-11说明了由子网掩码和前缀定义的子网，这将在第4章中讨论。

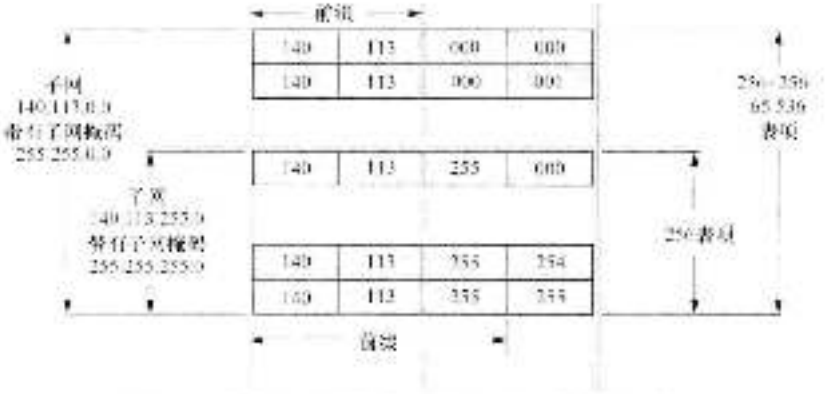


图1-11 子网、子网掩码、前缀：分段的网络地址块

点对点链路上的子网大小固定为两个节点。在广播链路上的子网大小通常取决于性能和管理策略。然而，一个子网中包含太多的主机，将导致严重的争用。与此同时，管理策略运营支撑点在其管理域中所有子网的大小保持固定。通常将子网的大小设置为256。

自治系统

将互联网上的节点分成组，以便形成很多由路由器互连的子网。目前，互联网规模已经有超过5亿台主机、数百万台路由器。如果子网的平均大小为50，那么子网的数量将有100万，这就意味着路由器将记住和维护太多的子网条目。

很显然，在子网之上还需要有另一个层次。一个自治系统（AS，有时又称为域）由一个组织管理的子网和将它们连接起来的路由器所组成。AS内的一台路由器知道所有在内部AS路由器和AS内的子网，以及负责AS之间路由的多台外部AS路由器。内部AS路由器将分组转发到同一AS中的主机上。如果分组的目的地是另一个AS中的主机，事情就会更加复杂。它首先通过多台内部AS路由器将分组转发到本地AS的一台外部AS路由器，然后由外部AS路由器将分组转发到目的地AS，最后再由目的地AS的内部AS路由器将分组转发到目的地主机。

有了子网和AS，内部AS或外部AS的分组转发都可以以一种可扩展的方式进行，而不会给内部AS和外部AS路由器带来太大的负担。如果AS内的子网平均数为50，AS数量将为20 000，这是一个外部AS路由器能够负担处理得起的数量。AS不仅解决了可扩展性问题，而且还为运营商保留了对网络的管理权。AS内部的路由和其他操作可以分开，并且对外部世界是不可见的。

图1-12说明在中国台湾交通大学（NCTU）的AS，在同一AS中，给每个系分配了多个子网。整个互联网有成千上万个类似于这样的域。

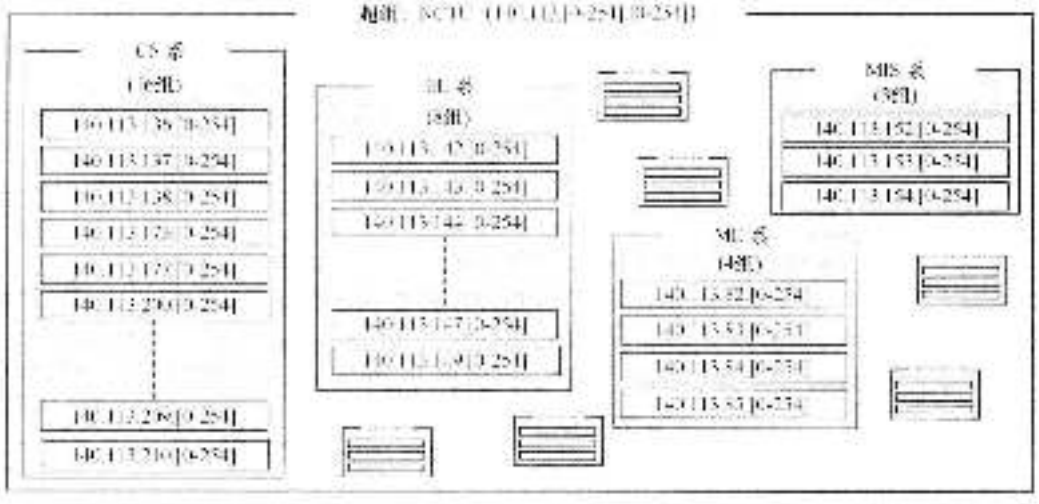


图 1-12 一个学术域，AS 或机构：NCTU

1.3.3 资源共享的解决方案

与主要用于电话的电信相比，数据通信有和类型多的应用，那么需要的是决定互联网体系结构是否应该有多种类型的运输层。有两种主要的方案。

多种应用共享单一的网络

由于分组交换导致的拥塞给互联网带来了一个更加艰巨的挑战。应该如何控制流量？拥塞控制和流量控制，以避免产生网络和接收方的资源又溢出。从端到端的观点中可以得出，互联网主要在端系统而不是在中间路由器上执行流控制。

总之，互联网架构在决定资源共享方式时主要考虑了三个问题：1）是否应该区分不同应用的流量处理；2）资源共享策略是什么；3）流控制应该放在哪里来执行策略。互联网在网络内部提供基础的尽最大努力服务。同时使用端到端的拥塞和流量控制在资源共享中实现公平的策略。

基础的尽最大努力服务：IP

可以将应用至少分成三种类型：交互式、文件传输和实时的。交互式应用产生少量的流量，但需要及时地做出响应。另一方面，文件传输应用产生大量的流量，可以容忍较高的延迟。实时应用则既有连续的流量也有低延迟的需求。如果要为了支持每种应用都要有一种连接类型，那么互联网内部的路由器就要是类型感知的，以便对不同的分组进行处理。相反，互联网提供了单一类型的连接服务，即尽最大努力 IP 服务。在共享有限的资源时，所有的 IP 分组都一样处理。

作为一种互联网核心的协议和服务，IP 具有分组交换数据报的形式。它非常简单，因为除了进行转发之外它不具有额外服务，只有无错检测的简单校验和。它没有内置的流量控制。从吞吐量、延迟、抖动和丢失上来讲，它是不可靠的。也就是说，它不能保证能以多快的速率发送分组，分组何时会到达目的地，甚至还不知道分组是否能够可以到达目的地。它也不能保证一系列分组的按序发送。分组到达目的地的顺序与分组离开源的顺序可能不相同。然而，如果校验和无效，它就丢弃分组，并且为具有错误头复制将恢复错误错到端的协议来完成。如果应用程序需要错误恢复或流量控制，这些增值服务就必须在下层的端到端的协议来完成。

端到端的流量控制和错误恢复：TCP

TCP 是“有礼貌”的端到端协议，用来调整从源流出的分组中的突发性，以便使所有流都可以公平地共享资源。通过要求所有源“有礼貌地”对拥塞做出响应，就会减少路上拥塞以及需要从错误中恢复的机会。TCP 也是运行端到端协议的一种可靠的端到端协议。从丢失方面来讲，它是可靠的，即由于错误或拥塞而丢失的分组可以按 TCP 协议恢复。然而，从其他性能测量（如吞吐量、延迟、抖动）来讲，它仍然是不可靠的。为了保证这些性能测量，分组交换就需要额外的、通常有状态的在网络内部实施的机制。虽然确实存在这样的解决方案，但它们没有被大规模地部署。TCP 的丢失保证

对于大多数数据通信应用来讲是足够的。

还有许多不需要无失失保证的应用。例如，分组语音或视频流应用就可以容忍很小百分比的丢失，同时仍能维持播放质量。事实上，因错误恢复而采用的端到端重传所导致的延长时间对于这种实时应用来讲是不可接受的。其他一些应用，如网络管理，可能有自己内置到它们客户端和服务器的差错控制，从而并不依赖于底层端到端传输服务中的差错控制。对于这些应用，通常使用 UDP。UDP 是另一种端到端协议，虽然它相当靠谱，仅有一个简单的校验和用于差错检测，但是却没有差错恢复或流量控制。在图 1.10a 中，我们可以分别看到 TCP 和 UDP 的应用。

为了避免拥塞并且能够公平地共享带宽，在 TCP 中嵌入了一个有趣的理念：来自每个流的突发位的数目应该大致相同。也就是说，所有活动的 TCP 流为互联网贡献的流量应该是相同的。突发位数实际上就是带宽延迟乘积（BDP）。对于相同的 BDP，如果一个 TCP 流以更高的延迟传输更远的路径，其带宽或传输速率就要更小一些。TCP 流没有显式的传输速率。相反，它们使用窗口大小来控制 BDP（突发位数）。考虑一条传输许多 TCP 数据流的链路，这些数据流的跳数或端到端延迟可能是不同的。为了取得相同的 BDP，其传输速率是不同的。即使在带宽充裕、没有拥塞的情况下，一条跨国 TCP 数据流肯定比本地 TCP 流的传输速率低。

除了公平策略外，TCP 需要调整其基于窗口的控制以反映当前的网络和接收方的情况。首先，速率应受到接收方容量的约束。其次，当网络开始拥塞时减慢速度，并当拥塞消弱时就要增加速率。但是，TCP 应该多快地降低或者提高其速率或窗口大小？较差增加和成倍递减（AIMD）显然是一种不错的选择，它会缓慢占用带宽，但拥塞时则响应迅速。许多性能问题和注意事项还需要进一步澄清，这些将在第 5 章中介绍。

1.3.4 控制平面和数据平面操作

有了解决连通性、可扩展性和资源共享的决策，为了让互联网能够如我期那样运行，仍然有许多具体工作要考虑。它们包括控制平面的路由和错误报告、数据平面的转发、差错控制和流量控制。

控制平面操作

在 1.2.2 节中，我们提出了路由协议和算法设计中所涉及的问题。所做的选择可以概括如下：在后台预先计算、逐跳、按每个目的地前缀（子网或 AS）粒度、内部 AS 路由的局部或全局网络状态信息、外部 AS 路由的局部网络状态信息，以及大多数情况下的单条最短路径。这些选择都有原因。当网络拓扑结构是动态的时，就需要使用按需源路由；否则，在每台路由器上预先计算逐跳路由最合适。对于可扩展层次结构的子网和 AS，内部 AS 和外部 AS 路由的粒度分别为按子网和按 AS。

正如在 1.3.2 节曾经讨论过的，在一个子网数目很少的（几十到几百个）AS 中，很容易收集到局部或全局的网络状态信息。然而，全局的 AS 数可能有成千上万，因此很难收集到最新的全局网络状态信息。全局网络状态信息包含整个网络的拓扑结构，并由来自所有路由器的处理状态广播所构造。另一方面，局部网络状态信息包含到达目的地或子网或 AS 的下一跳和距离，是通过和邻路由器之间交换距离向量构造的。最后，为了简洁，选择一条最短路径，而不是多条路径。到达一个给定目标子网或 AS 的多重路径将有更好的资源利用率和负载平衡，但这会使路由和转发的设计复杂化。到达某个目的地在转发表中有多个表项，在控制平面中维护表项并在数据平面选择要经过哪条表项就不再是谈不运道的工作了。路由信息协议（RIP）依赖于局部网络状态信息，而开放最短路径优先（OSPF）依赖于全局网络状态信息，它们是两种常用的内部路由协议。而边界网关路由协议（BGP）依赖于局部网络状态信息，在外部 AS 路由中处于主导地位。

在控制平面上还有一些其他的工作要做。还需要实现组播路由、错误报告和主机配置。组播路由是更加复杂的单播路由。尽管存在许多解决方案，但我们将在第 4 章中讨论。当路由器或目的地处理分组发生错误时，向源主机报告。错误报告也可以用来探测网络。互联网控制消息协议（ICMP）是用于报告错误的协议。主机配置协议，动态主机配置协议（DHCP）是一种自动配置任务的努力以实现即插即用。虽然完全自动配置整个网络在目前来讲仍然不可能，但是 DHCP 能让管理员不用手动地配置所有主机的 IP 地址和其他参数。然而，路由器配置还需要依靠手动配置来完成。

数据平面操作

转发分组实际上是一种表查找的过程，从分组中提取目的地 IP 地址，然后与表项中的 IP 前缀进行对比。对于内部 AS 和外部 AS 转发，表项的粒度分别是按子网或按 AS 级的。子网或 AS 的 IP 前缀可能是从 2～32 位的任何长度。匹配前缀的表项中包含转发分组的下一跳信息。但是，如果一个地址块被分配到两个子网或 AS，那么它就可能有多个匹配的前缀。例如，如果地址块 140.113 被分成两部分，如 140.113.23 和其余部分，并分配给两个 AS，外部 AS 转发表将分别包含前缀为 140.113 和 140.113.23 的两个表项。当接收到目的地为 140.113.23.52 的分组时，它将与两个表项匹配。默认时，遵循最长前缀匹配。

按照 1.3.1 节中讨论过的有关端到端的观点，互联网中的差错控制放到了端到端的 TCP 和 UDP 中。TCP 和 UDP 中的校验和可以检查整个数据分组中的错误，尽管它只能检测出单个位错误。如果检测到一个错误，UDP 接收方就丢弃并忽略这个分组，但是 TCP 接收方则会告知 TCP 发送方，请求重传。在 IP 中的校验和仅保护分组的头部，以避免协议处理中的错误，但它并不保护分组中的有效载荷。在节点上，如果检测到一个错误，那么节点就会丢弃这个分组，并向源发送一个 ICMP 分组。源如何处理，就要依赖具体的实现了。为了效率，许多底层的链路也会将差错控制放在链路层上实现，但这种差错控制独立于在 TCP、UDP 和 IP 已经实现的功能。

拥塞控制的目的是避免和解决拥塞，以及公平地共享带宽资源。TCP 提供了一个相当令人满意的解决方案，如 1.3.3 节中所述。另一方面，UDP 像一个狂野的赛车手那样随心所欲地发送分组。虽然目前从整体流量容量来讲 TCP 流量仍然占主流，但流媒体和 VoIP 应用将来可能有一天会使 UDP 流量超过 TCP。当混合了 UDP 流量后，TCP 流量则会受到损害。这急切需要新的研究通过类似于 TCP 那样的端到端拥塞和流量控制来限制 UDP。总之，UDP 流应该是 TCP 友好的，这样它对共存的 TCP 流的冲击影响与 TCP 流对另一个共存的 TCP 流的冲击影响一样。

行动原则：互联网体系结构特点

这是非常适合再次强调互联网特点的地方。为了解决通过性和资源共享的问题，互联网将端到端观点运用到了极致，在将复杂性推到边缘设备的同时维持无状态的核心设备。核心设备运行不可靠的无状态路由，而边缘通过差错控制和拥塞控制分别用来保证正确性和健壮性。一个带有子网和域的简单的三层结构就足以将互联网扩展到高达数十亿个节点。但还需要有额外的机制来保证能够遵守上述特点。但是 OSI、ATM、通过 IntServ/DiffServ 的 QoS 和 IP 组播等都是无法替代甚至要与互联网共存的反例。它们都需要一个有状态的核心，以保持经过连接的入口表项。转发更多的分组而路由较少分组的 MPLS，也面临着同样的困难。尽管它的灵活性，软状态切换允许 MPLS 更好地遵守无状态路由，可以轻松地部署在一个小规模的 ISP 上，但是在互联网上广泛采用 MPLS 依然存在挑战。

1.4 开放源代码实现

互联网体系结构为满足数据通信的要求和原则，提出了一套综合集成解决方案。这套解决方案是一个开放的标准。互联网体系结构的开源实现将同样的开放精神里向前推进了一步。本节讨论互联网体系结构开源实现的原因以及如何实现。首先，我们将开放式和封闭式的实现进行对比。然后我们说明在 Linux 系统中的软件体系结构，既包括主机上的包包括路由器上的。这种体系结构分成为几部分：内核、驱动程序、守护程序和控制器，并进一步对每一部分进行了简要的回顾。

我们将更多实现综述和调组有用的工具留在三个附录中介绍。附录 B 检查了 Linux 内核的源代码树，并总结了其网络互联代码。常用开发和应用工具分别放在了在附录 C、附录 D 中介绍。进行本书的动手练习之前，我们希望读者浏览一下这些附录。此外，开源的非技术方面，还包括发展历史、授权模式、资源等，这些将在附录 A 中的 A.2 节学习。

1.4.1 开放与封闭

提供商：系统、IC、硬件和软件

在描述互联网体系结构实现方式之前，我们应该确定系统中的主要组成部分和所涉及的提供商。

不管主机还是路由器，系统都要由软件、硬件和集成电路（IC）元件组成。在一台主机上，互联网体系结构大多数以软件和少部分的IC实现。协议栈中的TCP、UDP和IP是在操作系统中实现的，而应用协议和链路协议分别在应用程序和接口卡的IC上实现。如果CPU不能提供所需要的线速处理，那么部分协议实现可能从软件迁移到IC上，路由器上的实现与此类似。

系统提供商可能会自己开发和集成所有上述三种类型的组件，也可能会将它们中的一些外包给软件、硬件或IC元件提供商。例如，一个路由器的系统提供商可能利用来自一个或多个IC提供商的板载芯设计，实现并制造硬件，同时从软件提供商获得许可证并修改软件。

从专有、第三方到开放源代码

有三种方式将互联网体系结构实现到一个不管是主机还是路由器的系统中。它们分别是1）专有封闭式；2）第三方封闭式；3）开源。一个大的系统提供商可以负担得起数百名工程师的大型团队去设计和实施专用封闭的软件和集成电路。结果是封闭系统完全由提供商拥有其知识产权。对于小的系统提供商，维持这样一个庞大的团队就过于昂贵。因此，小的系统提供商宁愿采用软件或IC提供商提供的第三方解决方案，软件或IC提供商将自己的实现转给系统提供商，但会向他们收取许可费和每个拷贝的版税（软件）或购买费用（对于IC）。

软件和集成电路的开源实现提供了实现系统的第三种方法。无需自己维持一个庞大的团队或绑定到特定的元件提供商，系统提供商可以利用现有的丰富软件资源，而系统或IC供应商也可以利用不断增加的IC资源。他们反过来又会为开源社区做出回报。

开放性：接口或实现

当我们提到开放性时，指户要开放什么内容很重要。开放的是接口还是实现？如果是开源，那么我们就是指开放的实现。互联网体系结构是一种开放的接口，而Linux则是这种开放接口的一种开放实现。事实上，协议成为互联网体系结构的标准之一，就是要有既稳定又开放提供的可选有代码。这里，开放的接口和开放的实现往往是携手并进的。另一方面，IBM公司的结构化网络架构（SNA）是一种封闭的接口并具有一个封闭的实现，而微软的Windows是开放互联网体系结构的一种封闭实现。SNA已经消失了，但Windows仍巍然屹立。对于来自不同供应商的系统之间的互换性，必须有开放的接口，但不必要开放实现。然而，开放的实现具有许多优点。一个流行的开放源码包具有世界各地的贡献者，从而导致能够快速地打补丁来修复缺陷或增强功能，而且往往也会有更好的代码质量。

1.4.2 Linux系统中的软件体系结构

当一种体系结构转换成一个真正的系统时，确定在何处执行什么功能很重要。必须进行几个关键的决策：必须在何处实现控制平面和数据平面操作？什么功能必须在硬件、集成电路、软件中实现？如果在软件中实现，那么它应该在软件体系结构中的哪一部分？为了能够在基于Linux系统上进行这些决策，人们应该首先理解它的软件体系结构。

进程模型

像任何其他类UNIX或现代操作系统一样，Linux系统具有用户空间和内核空间程序。内核空间程序为用户空间程序提供服务。进程是一个可以在CPU上调度运行的用户空间或内核空间程序的化身。内核空间进程驻留在内核内存空间以管理系统操作，为用户空间进程提供服务，尽管它们并不直接提供服务。用户空间进程驻留在用户内存空间，可以在前台以应用程序客户端运行或在后台以应用服务器运行。在内核空间中，有些程序称为设备驱动程序，用来执行一些外围设备的I/O操作。驱动程序依赖于硬件而且必须能够识别外围硬件并对它加以控制。

当一个用户空间进程需要来自内核空间程序的某个特定服务（例如，发送或接收分组）时，它就发出一个系统调用，以便在内核空间中生成一个软件中断。进程然后切换到内核空间执行内核空间程序完成所要求的服务。一旦完成，进程便返回到用户空间运行它的用户空间程序。注意服务由内核空间程序（不是上述管理系统的内核空间进程）提供，这是由用户空间进程在它们切换到内核空间时执行的。系统调用作为用户空间和内核空间之间的应用程序接口（API）。套接字（socket）是专用于网络互联目的的系统调用的一个子集。在1.4.4节中将对套接字做更多的介绍。

在何处实现什么功能

假定上述进程模型，可以遵守多个原则来决定在何处实现什么功能。由于内核空间程序为用户空间程序提供基本的服务，所以最关于应用程序应该作为内核空间程序来执行，而将应用程序客户端和服务器放在用户空间程序中实现。在内核空间中，系统与硬件的处理应该以设备驱动程序的形式来实现，而其余的则驻留在核心操作系统上实现。遵循这些指导方针，在 Linux 系统中的何处实现什么功能就显而易见。所有的应用协议在用户空间的客户端和服务器中实现，而 TCP、UDP 和 IP 在 Linux 内核中实现。各种依赖于硬件的链路层在驱动程序和硬件中实现。根据已经在硬件（既可以是简单的数据电路也可以是 ASIC）上实现的功能，用于链路上的驱动程序可以是链路层协议处理器也可以是一个纯粹的“分组读取器和写入器”。对于定时必须保证正确链路协议操作很重要的链路上，链路层协议应该由 ASIC 来完成而无需 CPU 的参与。否则，用于链路的硬件可以是一个简单的收发器，而将协议的处理留给链路的驱动程序来完成。

在 IP 中实现转发时，差错控制大多放在 TCP 中，当然某些也可能在 IP 和 UDP 中实现，而流量控制在 TCP 中实现，但所有这些都要在 Linux 内核中实现。还有一个问题是：我们应该将互联网的控制平面操作放在何处？它们应该在 RIP、OSPF 和 BGP 中，包括路由，在 ICMP 中包括差错报告，在 DHCP 中包括主机配置。既然 ICMP 简单并独立于应用，它就应该和 IP 协议一起放入内核中。尽管独立于应用程序，但 RIP、OSPF、BGP 和 DHCP 非常复杂（尤其是前三个，需要运行复杂的路由计算算法），但仅用于控制分组的处理。因此，将它们放入用户空间程序，作为守护进程持续在后台运行。可以看到，所有的单播和组播路由协议都在守护程序中实现。不把它们放入内核的另一个原因是因为它们的数据太多了。但是，它们作为守护程序实现会产生另一个问题：路由守护进程需要更新位于内核中的 IP 转发程序所管理的转发表。决策就是为路由守护程序在内核中通过用户空间与内核空间之间的一套字 API 写入数据结构。

路由器和主机内部

通过以下两个例子向读者展示在网络节点中可以实现什么样的常用操作以及它们所处的位置。图 1-13 展示了路由器的常见操作。路由协议（RIP、OSPF、BGP 等）执行守护程序（routed、gated 或 zebra 用于动态的路由协议），从而更新用于“快速驱动程序”所查找的内核中的路由表（又称为转发表）。协议驱动程序包括 IP、ICMP、TCP 和 UDP，并调用适配器驱动程序发送和接收分组。另一个守护进程，inetd（超级网络守护进程），调用各种用于网络相关服务的程序。如图 1-13 中带有箭头的线条所示，控制平面的分组在协议驱动程序中由 ICMP 或者更往上由 RIP、OSPF、BGP 等守护进程处理。然而，在数据平面的分组将在协议驱动程序中的 IP 层被转发。

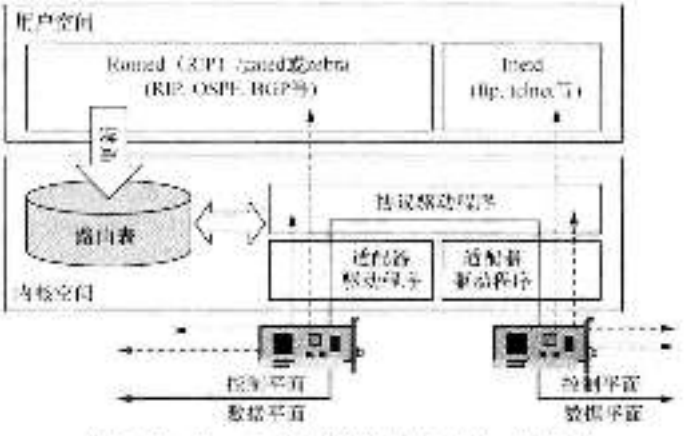


图 1-13 Linux 系统中的软件体系结构：路由器

同样，图 1-14 中显示了一台服务器主机提供的运行。各种应用协议（例如，Web、电子邮件）服务器是在守护程序（如 Apache、qmail、rot smtp 等）中实现的。主机和路由器之间的明显区别在于，在主机中没有进行分组转发，因此它只需要一个链路接口或适配器卡。对于该主机，大多数分组是来自于守护程序服务器的数据平面分组。唯一的控制平面协议可能是用于差错报告的 ICMP。

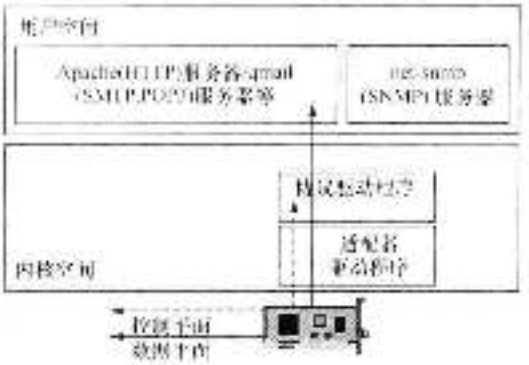


图 1-14 Linux 系统中的软件体系结构：服务器主机

1.4.3 Linux 内核

已经将协议实体定位在了守护进程、Linux 内核、驱动程序和 IC 后，让我们来看看这些组件的内部情况。这里我们没有打算很详细地介绍它们，相反，我们只是浏览一下每个组件的主要特点。

图 1-15 显示了 Linux 内核中的关键部件。就像任何其 UNIX 操作系统一样，它主要由五部分组成：进程管理、内存管理、文件系统、设备驱动、网络互联。这里我们不打算详细说明每个组件的用途。

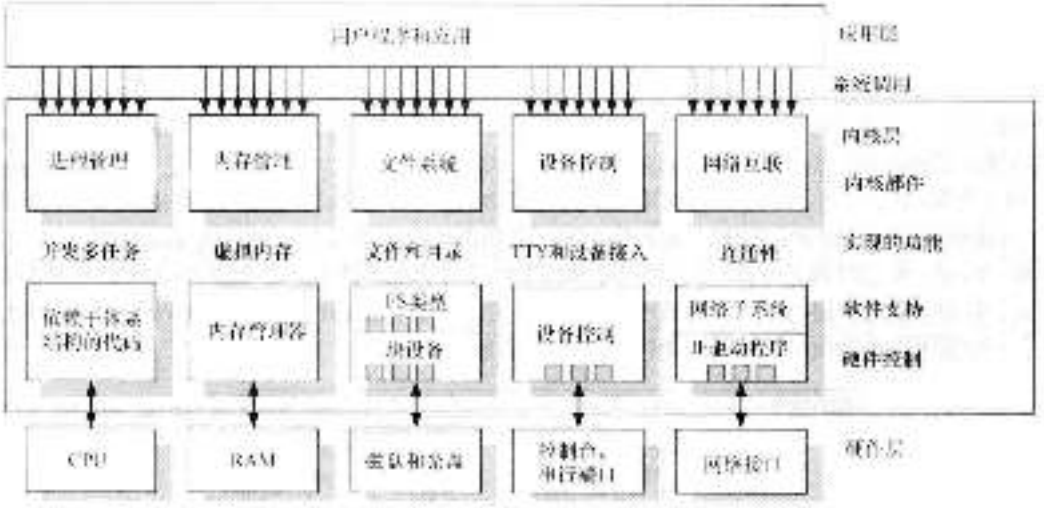


图 1-15 内核组件

每个组件有两个层次：独立于硬件的和与硬件相关的。与硬件相关的部分实际上就是用于驱动特定台式适配器卡的驱动程序，或依赖于 CPU 体系结构的代码和用于各种 CPU 体系结构的虚拟内存管理器。在这些组件中，网络互联是我们关注的焦点。附录 B 介绍了 Linux 内核源代码树，尤其是网络互联部分。

1.4.4 客户端和守护进程服务器

在内核之上，用户空间运行着我们的用户空间程序，虽然它们偶尔也会调用系统调用并切换到内核以便接收服务。对于网络互联服务，会接字 API 为用户空间运行与其他远程用户空间进程（通过 TCP 或 UDP 套接字）进行通信提供了一套系统调用，生成它自己的 IP 分组（通过原始套接字），直接监听接口卡（通过数据链路提供商的接口套接字），或者在同一台机器与内核通信（通过路由套接字）。这些套接字在图 1-16 中说明。对于每一个在特定套接字 API 的系统调用，Linux 内核通过一套内核空间函数来实现这个系统调用。

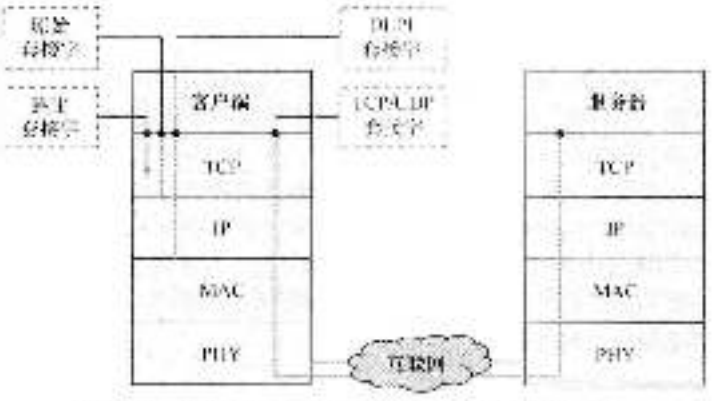


图 1-16 客户端与守护程序服务器：四种套接字 API

这些套接字被几乎不同的应用程序中，例如，Apache 服务器与许多其他的服务器一起使用 TCP 套接字。zebra 路由守护程序使用路由套接字更新在内核中的转发表，分别使用 UDP 套接字、原始套接字，以及 TCP 套接字分别发送和接收 RIP、OSPF 和 BGP 协议的消息。在图 1-10a 中的协议栈显示了它们选择的套接字 API。RIP、OSPF 和 BGP 分别运行在 UDP、IP 和 TCP 之上。

1.4.5 接口驱动程序

设备驱动程序是一组由内核调用的动态链接函数。关键是要知道，驱动程序常作为由硬件中断所触发的。当设备完成了一次 I/O 操作或检测到需要处理的事件时，他产生一个硬件中断。这种中断必须由了解此设备的驱动程序处理。但所有的中断必须首先由内核来处理。内核如何知道需要选择哪个驱动程序来处理这个硬件中断？设备的驱动程序应该将本身作为一个中断服务例程注册到内核中以便处理特定的硬件中断。然而，部分驱动程序不在中断服务例程中。这一部分被内核调用但不由中断处理。当然就不在中断服务例程中。图 1-17 显示了一个网络接口卡的驱动程序。分组接收和已经的分组发送都被接口卡注册为中断服务程序。由于从接口卡上产生了硬件中断，所以它们就被内核调用。部分发射机没有注册到中断服务例程中，因为只有当内核有分组要传送下个们才被调用。

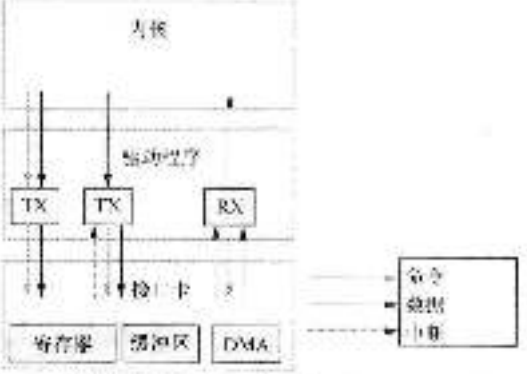


图 1-17 中断驱动的接口驱动程序：read (in) 和 write (out)

除了发送和接收分组外，驱动程序也可能对协议栈提供设备某些处理。虽然部分链路层协议可以在接口卡上的 ASIC 中实现，但仍有一些协议处理是在驱动程序中实现的，这些将在第 3 章中学习。

1.4.6 设备控制器

如 1.4.5 节中所述，位于内核上的驱动程序处理由设备所产生的中断。此外，驱动程序也需要在初始化阶段或者当内核想要更改某些配置时配置设备。那么驱动程序如何与设备通信呢？事实上，在

设备内部有一个设备控制器，通常它是由一个集成电路芯片负责与驱动程序进行通信。控制器提供了一组寄存器供驱动程序进行读和写操作。通过读或写这些寄存器，驱动程序可以发出命令，或者从设备中读取状态。此外，根据CPU体系结构的类型，有两种不同的方法来访问这些寄存器。有些CPU提供了一组特殊的I/O命令，例如，in和out用于驱动程序与设备进行通信，同时某些CPU预留了一系列的内存地址来供驱动程序发布I/O命令，如内存访问，即内存映射I/O。

因此设备控制器才是设备的核心。它不断地监视设备并对外部环境或驱动程序的事件立即做出响应。例如，当网络适配器中的控制器检测到驱动程序向命令寄存器写入了一条发送命令后，就会运行MAC协议传输分组。当出现冲突时，它可能会设置尝试重发。与此同时，它监视网线以便检测到传入的分组，并将它们接收到适配器内存中，根据MAC头部检查其正确性，然后触发一个中断以便请求相应的驱动程序将分组移动到主机内存中。

1.5 本书路标：数据包的生命历程

前面我们已经介绍了互联网体系结构形成的原因以及如何实现开源的，但是还没有进一步的详细介绍。在后续章节中，我们将详细介绍为什么以及如何在协议栈的每一层中实现，并解决互联网上的两个紧迫问题：QoS和安全性。在介绍这些章节之前，最好先看一看分组如何在终端或中间设备里存储和处理，这是非常具有启发性和娱乐性的。本节还介绍理解本书涵盖的开源实现所需要的背景知识。

1.5.1 数据包数据结构：sk_buff

对于1.3节中提到的分组封装，需要多个网络层（或模块）的合作以便将数据封装在分组里或从分组中取出数据。为了避免频繁地在这些模块之间复制数据，利用一个通用的数据结构来存储和描述分组，这样每个模块仅通过一个内存指针就可以传送或访问分组。在Linux中，将像这样的数据结构命名为sk_buff，在文件skbuff.h中定义。

利用sk_buff结构存放分组及其相关信息，例如，长度、类型或任何随着网络模块之间分组而交换的数据。如图1-18所示，该结构包括许多指针变量，这些指针的大部分指向实际上已经存储了分组的另一个固定大小的内存空间。带有前缀“+”的字段名代表基于字段头部的偏移量。变量next和prev将连接前面和下一个sk_buff结构，以便使节点中的分组维持在一个双向链表里。变量dev和sk分别指示分组来自或将要传输到的网络设备和套接字。在分组中存储了变量transport_header、network_header和mac_header，分别包含4、3、2层的头部位置的相对于由head变量指针所指位置的偏移量。

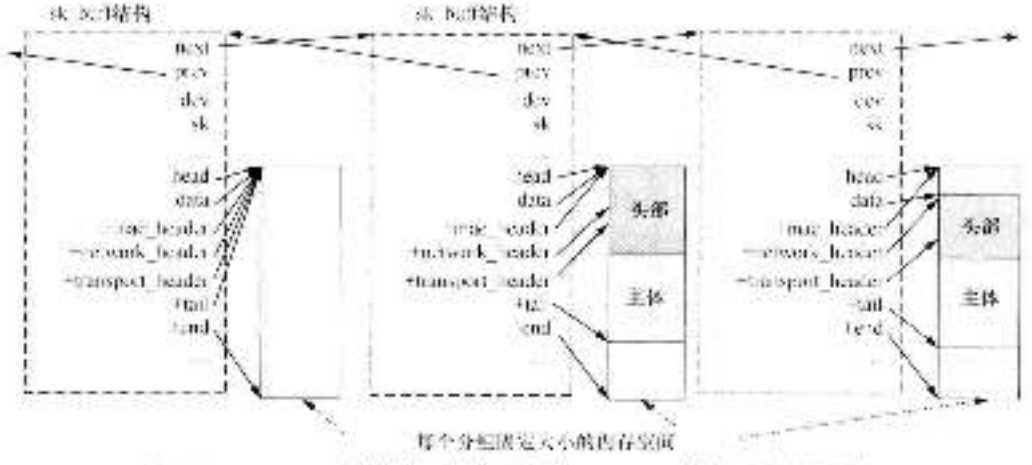


图1-18 sk_buff结构的双箭表以及在sk_buff中的一些重要字段

除了数据结构外，还为网络模块提供了一组例程，用来分配或释放sk_buff并修改sk_buff中的数据。当从网络设备上接收分组时，就调用例程alloc_skb()来为分组分配一个缓冲区。如图1-18所

系，最左边的 sk_buff 最初因为没有分组存储在已分配的空间中，所以所有指向分组空间的指针具有与变量头部相同的值。当一个进入分组到达分配的空间时，这可能看起来像图 1-18 中间部分的 sk_buff，将调用例程 skb_put () 将尾指针 tail 指向末端并且三个头部指针指向其相应的位置。下一步，当协议模块切换到除其头部并将分组传送到上层协议时，都会调用例程 skb_pull () 向下移动指针 data。在上层协议中的分组看起来像图 1-18 中最右边的 sk_buff。最后，处理完一个数据包后，就会调用例程 kfree_skb () 派出 sk_buff 所占用的内存空间。

在接下来的两节中，我们在一台 Web 服务器和一台网关（或路由器）中将数据包的生命历程更分成几个阶段并将这些阶段与我们的后续章节相关联，用数本书的路标。

1.5.2 在 Web 服务器中数据包的生命历程

图 1-19 画出了 Web 服务器中发生的四种数据包值。一般来说，当一个可联网客户端希望从一台 Web 服务器上获取一张网页时，客户端便发送出一个分组指示目的地 Web 服务器和所请求的网页。接下来，分组经过一系列的路由器转发最后到达 Web 服务器。分组被服务器的网络接口卡（NIC，简称网卡）收到后，在服务器中的行程就像路径 A 所绘制的那样。首先，NIC 将信号解码为数据，这将在第 2 章中介绍。网卡 NIC 然后根据驱动程序将分组转移到驱动程序从 sk_buff 组中分配的内存空间中。通过阅读第 3 章的内容，读者可以进一步学习在网卡和适配器驱动程序中运行的协议及机制。

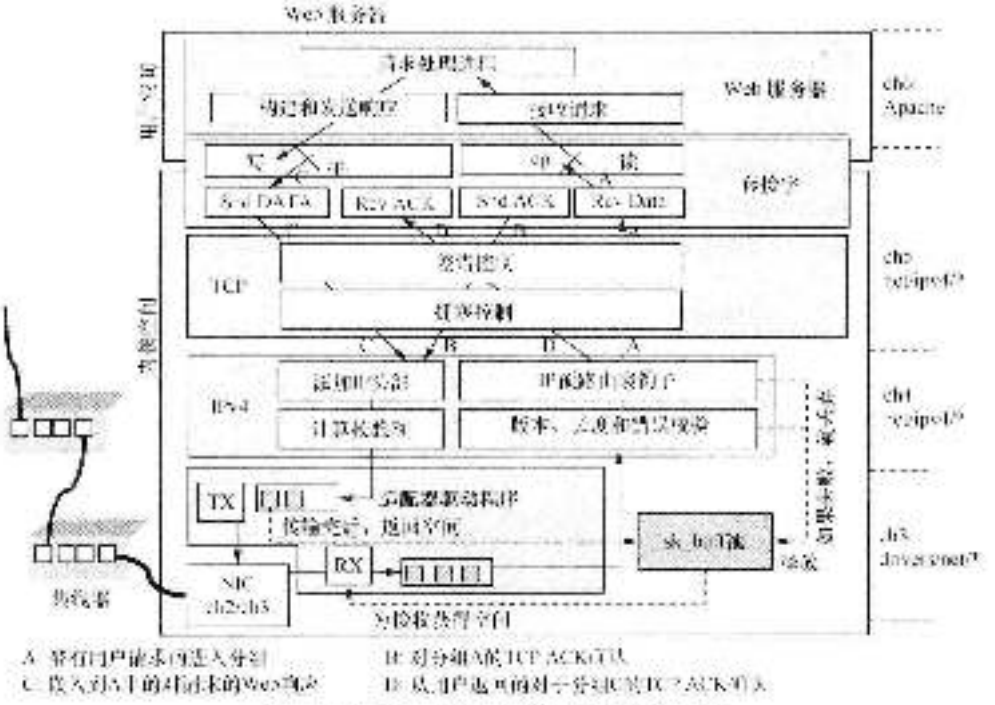


图 1-19 在 Web 服务器中数据包的生命历程

一旦将数据包存储到 sk_buff 中，适配器驱动程序调用并将一个指向分组的 sk_buff 指针传递给 IP 模块的接收函数。然后接收函数检查数据包的有效性，并将数据包挂接到 IP 预路由表上以便检查安全性。该表是 netfilter（网络过滤器）使用的重要结构之一，嵌入在 Linux 内核中的防火墙模块。IP 模块中的结构和操作将在第 4 章中详细介绍，而将来全操作留到第 8 章中讲述。接下来，数据包被 netfilter 推入到 TCP 模块中，第 5 章将介绍如何从在 sk_buff 中的分组中提取出用户数据，进行差错控制，并把它传递给应用程序——这里就是 Web 服务器。由于 Web 服务器是用户空间程序，所以数据包中的有效载载（也就是数据）就必须从内核内存复制到用户内存中。同时，根据收到数据包的头部，TCP 模块构建 ACK 分组，然后沿路径 B 传输。ACK 通过 TCP 模块、IP 模块、适配器驱动程序，

网卡和网络路径传输，最终到达客户端。因此，客户端保证将对所需要的网页请求成功地传送到 Web 服务器上。

与此同时，由将在第6章介绍的 Web 服务器处理在其套接字数据结构中的请求，这是从 TCP 模块中复制而来，生成响应并通过套接字接口发送出去。响应通过 TCP 和 IP 模块传输，如路径 C 所示。当它离开 IP 模块通过互联网传输时，使用协议头部封装，并且有可能拆分成多个数据包。最后，分配给数据包的空间将释放回到 sk_buff 池中。稍后当互联网客户端接收到响应时，它的 TCP 模块会发回一个 TCP ACK 给 Web 服务器的 TCP 模块，TCP ACK 通过路径 D 确认响应已成功交付到互联网的客户端上。

1.5.3 数据包在网关中的生命历程

由于路由器或网关的目标就是在互联网或互联网与企业内联网（Intranet）之间转发或过滤分组，所以它至少有两个网络适配器，如图1-21所示。需要注意的是，企业内联网是一种专用网络，它能安全地与员工共享组织的任何资源。此外，路由和过滤模块需要分别确定哪个适配器转发数据包以数据包是否应该被丢弃以便确保企业内联网的安全。基本的操作，例如 sk_buff 处理、差错控制及模块之间的交互，与前述服务器中的保持一致。路由器或网关，除了具有一些路由和安全功能的守护程序外，通常没有 TCP 或上层的模块，但它会在内核中有转发、防火墙、QoS 等功能，如图1-21所示。

性能问题：服务器中从套接字到驱动程序

图1-20中说明了具有 Intel 82566DM-2 以太网适配器和 2.0GHz CPU 的 PC 服务器的分组处理时间。在 Linux 内核中的最接口的设置函数是 rdtscll（）（或在 x86 架构上的汇编指令 RDTSC），所它来读取 TSC（时间戳计数器，以 CPU 确定或周期为单位）来计算每层消耗的 CPU 时间。对于一个 2.0GHz 的 CPU，主频周期等于 0.5ns。重复测试得到每个协议层的 CPU 平均消耗时间，其中不计算被抢占平均消耗 CPU 时间之很多的测试结果，以便排除上下文切换和中断处理的影响。除非特别注明外，本书中的所有性能问题都会采用这种方法。人们可以使用 do_gettimeofday（）和 printk（），或者直接使用附录 C 中介绍的时间测量分析工具 oprof/kernprof，但它们只精确到微秒级。

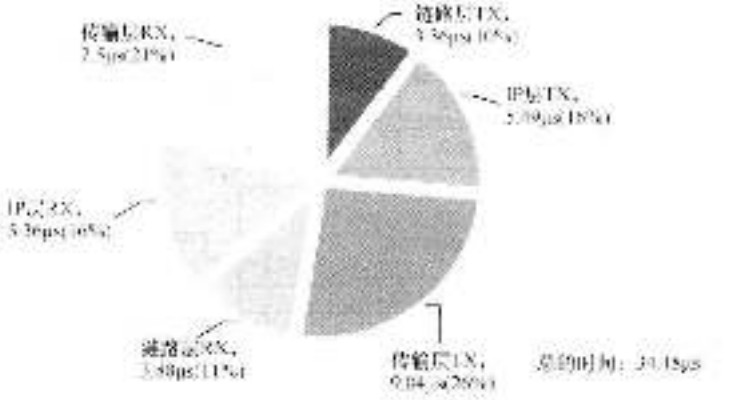


图1-20 服务器内从套接字到驱动程序消耗的CPU时间

所消耗的 CPU 时间可以分解成两部分。第一部分，RX，通过一个分组被链路层的设备驱动程序接收，在 IP 层和传输层的处理，并被复制给用户空间所引起的时间。第二部分，TX，描述的是内核空间中每个协议层处理来自用户空间服务器程序外出的分组需要的时间。总的时间为 34.18μs，这包括服务器内部的往返时间但不包括服务器内部请求和响应处理的时间。在这两个部分中，传输层都占很大比例的时间。显然，它消耗了大量的时间在用户和内核空间之间复制数据。这里，就链路层在 RX 和 TX 这两部分消耗的时间都是最少的。然而，我们必须知道，链路层所花费的时间在很大程度上取决于设备驱动程序和底层硬件的性能。在1.6节中我们将看到，在某些情况下，它消耗的时间将和图1-21中IP层消耗的一样多。

一旦收到来自企业内联网的分组，如图1-21右侧所示，网关就可以先验证其正确性，然后检查预

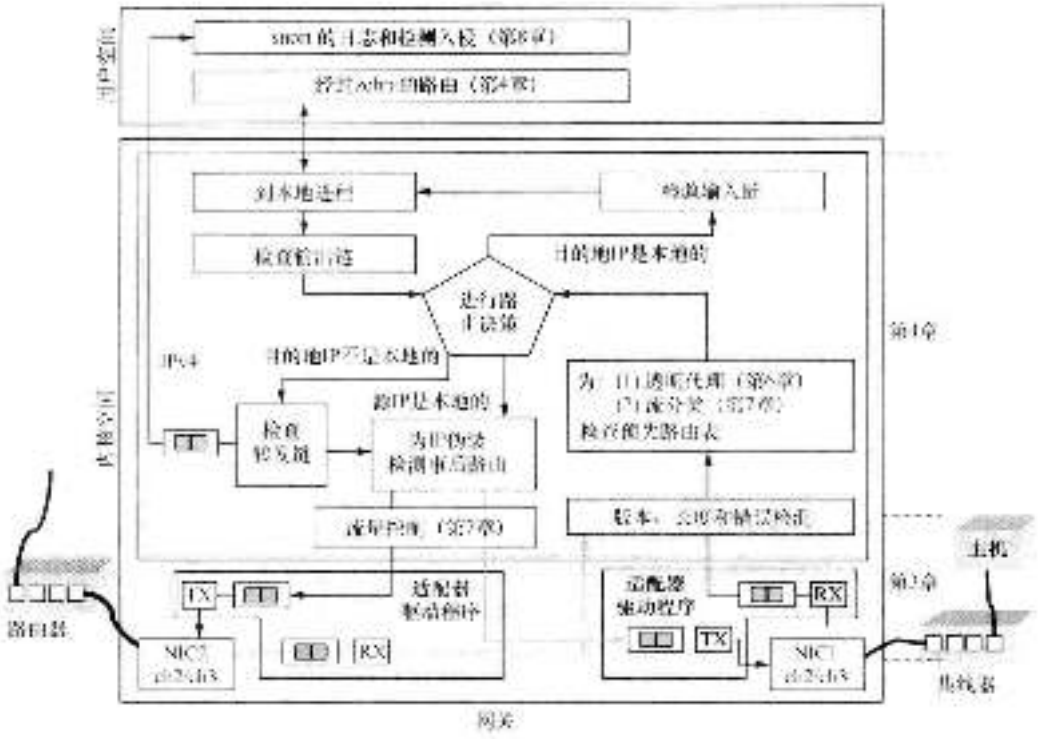


图 1-21　分组在网关中的历程

先路由表来决定是否将分组转发到互联网上。例如，如果在网关中启用透明代理功能，然后将发送一个 URL 请求分组，而不是直接发送到实际的网站，那么可能会重定向到本地的 Web 代理以寻求缓存过的页面，代理将在第 6 章中介绍。然后，通过检查转发链，即转发表或路由表看看是否包括远程目的地 IP 地址，做出转发或路由决策，这种处理过程将在第 4 章中叙述。由于安全方面的考虑和 IP 地址紧缺，网关可能在企业内联网中提供让所有主机共享一个公共 IP 地址的网络地址翻译（NAT）功能。所谓 NAT 功能，就是当传出的分组通过二次路由模块时，其源地址可能被替换，这通常称为 IP 伪装，这也将在第 4 章中介绍。最后，分组可能会被附加在一个预先路由模块中的标签以便区分在输出链路上具有带宽预留的分组服务类别和转发优先级，这由将在第 7 章中介绍的拥塞控制模块所管理。

另一方面，对于如图 1-21 中左边所示的来自互联网的分组，由于它会被检查以查看它是否包含由互联网主机传来的恶意代码软件，所以可以将分组从正常的转发链复制到有日志分析和检测的入侵检测模块。SNORT 是一个软件模块，它将在第 8 章中与其他几个提供安全功能的模块一起介绍。如果将分组提交给一个本地进程，即第 4 章中所介绍的路由守护进程，那么它就要通过输入链，然后再到达守护进程。此外，守护进程可能通过输出链发送分组。

性能问题：路由器内部从输入端口到输出端口

与服务器中的情况不同，分组通常并不需要通过路由器或网关中的传输层。如图 1-21 所示，当一个分组到达网络适配器时，首先会产生一个中断。在链路层的设备驱动程序触发 DMA 传输，将分组从适配器缓冲区转移到内核内存。然后将分组传递到 IP 层，它检查路由表并将分组转发到相应的外出适配器上。再次，外出适配器的设备驱动程序利用 DMA 传输将分组从内核内存复制到适配器缓冲区中，然后要求适配器对它进行传输。若穿个过程中，没有涉及传输层及以上层的内容。然而，有些控制平面的分组，可能会上升到传输层和应用层。图 1-22 显示了在路由器上处理分组所花费的 CPU 时间。这里的 DMA 时间就是一个例外。它实际上是走去的时间，而不是消耗的 CPU 时间。其他所有时间都是消耗的 CPU 时间。基于 PC 的路由器有一个 Intel PRO/100 以太网适配器和一个 1.1GHz 的 CPU。

由于使用了速度更低的 CPU，这里 IP 层接收时间比图 1-20 所示结果要高。此外，与图 1-20 相比，

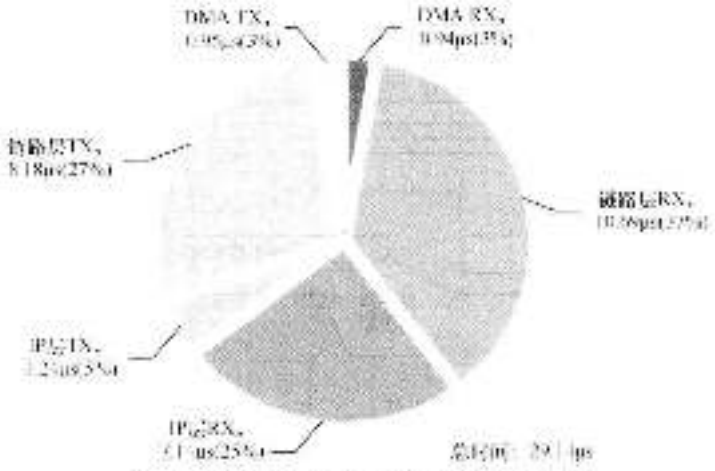


图1-22 路由器内从输入到输出的CPU时间

在链路层RX和TX的时间增加都比较显著，这是因为Intel PRO/100以太网适配器和100MB适配器的性能都低于Intel 82566DM-2以太网适配器和千兆适配器。路由器和服务器之间另一个明显的差异就是，通过IP层（即IP层TX）传送分组的时间。虽然这两种情况在IP层TX中经历相同的路径，但sk_buff中所携带的信息是不同的。在一台路由器中，sk_buff包含准备发送的信息，但需要更改的源MAC地址除外。然而，在一台服务器内，IP层需要首先将整个以太网头部添加到sk_buff中，然后才把它发送到链路层，这会导致服务器内IP层TX的处理时间要高于路由器的处理时间。最后，虽然使用了速度较低的硬件，但全部分组的处理时间，即29.14μs，仍然低于图1.20所示的由高端硬件服务器处理所需要的时间。

行动原则：互联网上分组的生命历程

检查Web服务器、路由器或网关中分组的生命历程的现状有趣。接下来，让我们叙述分组在客户端产生后，沿着多个路由器的路由，最后到达Web服务器的整个过程。第6章描述客户端程序，首先调用“socket”函数让内核准备一套套接字数据结构，然后调用“connect”函数，请求内核TCP模块通过第5章所详细描述的三次握手来与Web服务器中的TCP模块建立连接。通过在两个对应的TCP模块之间发送三个分组（SYN、SYN-ACK、ACK）。也就是说，在发送HTTP请求之前，已经交换过三个分组。它们遵从与在客户端、路由器或网关及服务器的HTTP请求相类似的步骤，但它们在TCP模块终止而不会上升到客户端和服务器程序。

在客户端和服务器之间建立TCP连接后，客户端程序在用户内存空间中产生HTTP请求，并调用“write”函数将请求发送到内核。然后被中断的内核从用户空间将HTTP请求复制到其套接字数据结构中，包括sk_buff，以便存储HTTP请求消息。在客户端程序中的“write”函数在该点返回。然后内核TCP模块负责其余工作，使用TCP头部封装HTTP请求，在将它传递到IP模块后封装IP头部，然后传递给适配器驱动程序，最后传递给网卡进行链路层封装。这个分组会穿越一系列的路由器或网关，在每一台设备内要经历1.5.3节中描述过的过程。即在每一台路由器或网关上，数据包在网卡上的接收会触发将信号解码成数据（详见第2章），然后中断适配器驱动程序（详见第3章）将其复制到sk_buff并把它传递到IP模块以便通过正常的转发链进行转发（详见第4章）。然后再由适配器驱动程序进行处理，完毕后将它传递到另一块网卡进行编码和传输（详见第2章）。

封装过的HTTP请求经过多台路由器转发后，最终到达服务器。它经历在1.5.2节中描述的过程。通过NIC后，被适配器驱动程序复制到sk_buff中，接过IP模块的检查，再由客户端上的TCP模块确认，然后被套接字接口复制到用户内存中，分组最终到达服务器程序。分组存储在服务器程序的用户内存中，在服务器解析HTTP请求信息并准备响应时其寿命也终止了。然后服务器程序重复相同的过程将HTTP响应发送回到客户端程序。响应也触发了从客户端TCP模块发送给服务器的

TCP 确认。如果 HTTP 会话结束，通常会发送四个数据包（FIN、ACK、FIN 和 ACK）以便终止 TCP 连接。至少有 3（TCP 连接）+1（HTTP 请求）+1（ACK 请求）+1（如果想到是以客户到一个分组的 HTTP 响应）+1（响应的 ACK）+4（TCP 连接拆除）=完成一次 HTTP 会话所需交换的 11 个数据包。

1.6 总结

我们从构建计算机网络必须满足的三个需求或目标，即连通性、可扩展性、资源共享作为入手点。然后，阐述了会限制我们探索解决方案空间的性能，操作和可操作性原则或约束。接下来给出了互联网解决方案和基于 Linux 的开源实现。最后，我们通过说明一个在 Web 服务器和路由器上的分组的生命历程来说明本书的整体布局。在本章中，我们介绍了许多将在这本书中使用的概念和术语。其中，交换、路由、无状态的、软状态、尽力服务、数据平面和控制平面是需要读者重点理解的。

在互联网演变发展中所做的唯一一个最大的设计决策是端到端的。它将差错和流量控制的复杂性抽给了终端主机，同时保持核心网络的简洁性。让核心如此简单，以便它能运行无状态的路由，而不是有状态的交换并仅提供尽最大努力的不可靠的 IP 服务。在主机上的端到端传输层运行带有差错和流量控制的可靠的面向连接的 TCP，或者运行没有太多控制的不可靠的无连接的 UDP。正是优秀的 TCP 运行流量和拥塞控制，以保持互联网的健康和资源共享社区的公平性。另一个大的决策是将互联网构建成带有域和相邻 IP 地址块子网的三个层次。它进行将路由问题分为域内和域间的问题从而解决了可扩展性问题。问题是前者的大小通常小于 256，而后者的大小是 65 536。两者的大小是可控的，但需要不同的方案进行扩展。

演变的沙漏

目前，互联网在网络层上有着唯一的 IP 技术而在传输层上则有多种技术，但是它是建立在许多类型的链路上的，同时提供庞大的应用服务。这个沙漏形的拓发线仍然处在不断地创新发展之中。中间层虽然仍然相当稳定，但面临着从 IPv4 过渡到 IPv6 以及限制不“礼貌的”UDP 流量所带来的压力。我们将分别在第 4 章和第 5 章中介绍。同时，它的无状态也一直遭到不断的挑战，就像我们前面已经讲述的那样。较低层已经收敛成唯一一种主流技术或占领市场的几种技术，尽管最后一公里的无线市场仍然是一个悬而未决的战场。我们将在第 2 章和第 3 章中学习到更多内容。在最上层，传统的客户端/服务器应用持续缓慢演变，但是新的对等（P2P）应用发展迅速，参见第 6 章中的介绍。

在 20 世纪 90 年代后期和 21 世纪初，人们希望重新设计互联网提供服务质量（QoS）以便难保延迟，吞吐量或失失率等。但所有的提案都要求在核心网络中加入某些有状态性，而这与原有的无状态性冲突，从而导致了最终的失败。目前，许多 QoS 技术仅适用于链路层，而不适用于端到端的层，对此第 7 章将有更多的描述。除了无线和 P2P 外，安全性可能是最亟待解决的紧迫问题。从早期关注控制“谁可以访问什么内容”和保护“在公共互联网上的私有数据”，目前注意力已经转移到保护系统不受到入侵、病毒和造成邮件破坏。第 8 章将对它们进行全面的介绍。

常见陷阱

传输延迟与传播延迟

这两者明显不同，但令人惊讶的是，如果我们不加以比较，有些读者可能在初次接触时就不能区分两者。传输延迟表示发格把一个分组完全推入到网络链路所需要的总时间。延迟取决于分组的长度（分组大小）和链路带宽。例如，对于一个长为 250 字节的分组，即 2000 位，在一台具有 1Gbps 链路的主机内的传输时间就是 2000（位）/10^9（位/秒）=2μs。

传播延迟表示一个分组通过链路所需要的总时间。它取决于信号传输的速度和距离。由于分组是通过电子传输的，其速度仅是光速的一个分数，并且仅受传输介质的影响。例如，对于通过全长为 1000km 的铜制海底光缆传输的分组，其传播延迟是 1000km/（2×10^8m/sec）=5ms。

吞吐量与利用率

前述同样的问题也发生在这两个术语上。吞吐量是用来描述在单位时间（通常是1s）内通过设备传输或处理的数据，通常以位或字节为单位。例如，我们测量到在1m内通过外出链路的数据量为75×10^{6}字节，那么我们可以计算出平均吞吐量为75×10^{6}（字节）/60（秒）$=1.25\times10^{6}$Bps。也就是说，平均每秒有1.25×10^{6}字节数据通过这条链路传递。吞吐量可以通过系统容量标准化，使其转变成0和1之间的值。

另一方面，利用率意味着链路上使用带宽的百分比或者设备繁忙时间的百分比。按照上面同样的例子并假设链路带宽是100×10^{6}bps，则链路利用率是1.25×10^{6}Bps/100×10^{6}bps = 10%。

第2、3、4、7层交换机

我们经常听到第2～7层交换机，但为什么需要这么多种交换机呢？交换机的基本工作原理是依靠分组上的标签来选择发送端口。这种原理可用来构建按照不同协议来获取标签的不同层的交换机。例如，第2层交换机可以通过观察来自某个端口传入的分组的源MAC地址来学习并记忆适配器在哪里，然后再将分组交换到具有目的地MAC地址的端口上。这样，MAC地址可用做第2层交换机的标签。

同样，IP地址、流id、URL可以分别用做第3层、第4层和第7层交换机的标签。第3层的IP交换机，实际上就是MPLS技术，将标签简化成一个数字并要求上游交换机用这个标签来标记将来到达的分组以便能够快速索引到标记表。这样一台IP交换机运行速度比传统的IP路由器更快。第4层交换机使用五元组流id（源IP地址，目的地IP地址，源端口号，目的地端口号，协议id）作为标签，并将属于同一流的分组交换到同一输出端口。这种持续的交换对于用户将全部事务交易切换到同一台服务器的电子商务应用来说非常重要。第7层网络交换机更进一步使用应用程序的头部信息，如URL或Web网页cookie作为持续交换的标签。这样就可以让电子商务交易在许多连接或流中持续更长的时间。有趣的是，没有第5层和第6层交换机，这是因为人们由于7层OSI模型的缘故喜欢称应用层为第7层而不是第5层。

基带与宽带

有些读者会混淆，具有较宽的带宽就是宽带，具有较小的带宽就是基带。事实上，这两个词汇几乎没有传达任何带宽数量的含意。在基带传输中，数据的数字信号直接通过链路传播。它是原始的方形传输信号。很容易发送或接收这样的信号。然而，一条链路可次只能传输一个这样的信号。这种方形的信号很容易衰减，所以不能维持一段很远的距离。因此基带主要用于局域网中。

在宽带传输中，数据的数字信号与模拟载波信号混合调整或特殊的频率。这样一来，不仅可以使产生的信号传播很远的距离而且可以在接收器恢复数字信号，链路也可以通过把每个数字信号与不同频率的模拟载波混合起来用来传输多个并行的数字信号。然而，这就需要一个更复杂的收发器。因此宽带主要用于广域网。

调制解调器与编解码器

有些读者可能会认为可以把编解码器单向用做调制解调器或者反之亦然，但事实并非如此。调制解调器是一种将数字数据转换成模拟信号或反之亦然的传输设备。前者称为调制，而后者则是解调。其目的是加强远距离传输时抵抗噪声的能力。最普遍的应用例子就是家用PC机通过ADSL调制解调器或电缆调制解调器接入互联网。

编解码器是一种将模拟数据转换成数字信号或反之亦然的设备。其目的是充分利用数字信号的纠错能力。最常见的例子是，当你在使用手机打电话时，你的模拟语音首先在手机上被数字化，然后再调制成模拟信号便于远距离传输到基站或更远的地方。数字信号可以在每个传输跳很容易地被恢复，因此，在接收端解调后，转换成原始的模拟语音。

进一步阅读

其他参考书

在scholar.google.com上搜索能够找到的有关计算机网络的大本重要教科书。下面列出这些教科书，并根据它们被引用的次数排序。

- A. S. Tanenbaum, *Computer Networks*, 4th edition, Prentice Hall, 2002.
- D. Bertsekas and R. Gallager, *Data Networks*, 2nd edition, Prentice Hall, 1992.
- W. Stallings, *Data and Computer Communications*, 8th edition, Prentice Hall, 2006.
- J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 3rd edition, Addison-Wesley, 2005.
- L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*, 4th edition, Elsevier, 2007.
- D. E. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture*, 4th edition, Prentice Hall, 2000.

Tanenbaum 的书有点面面俱到，在一本像讲故事一样的传统书籍。它主多地讲述如何而非为什么。Bertsekas 和 Gallager 的书仅着眼于性能建模与分析，可用于开设第二门课程。Stallings 的书是按百科全书式的平铺结构组织的，更加注重较低层。Kurose 和 Ross 的书采用自顶向下的顺序介绍分层协议，更加注重上层协议的处理。Peterson 和 Davie 的书讨论主多系统实现的问题，但大多都没有可运行的例子。Comer 的书则仅专注于 TCP/IP 协议栈，而将代码例子放在了第二卷。

互联网体系结构

下面阅读资料中的前三本讨论了推动互联网体系结构设计的一般理念。如果读者有兴趣追溯这些设计，它们将会是很好的参考书。有关以太网的文章可以作为以太网起源的经典参考资料。尽管以太网不是互联网体系结构的一部分，但我们仍然把它包括在内，因为它是支持互联网体系结构主要的有线基础设施。接下来的是有关建立互联网体系结构基础的三个关键 RFC 文档。下一个 RFC 文件是有关为保证服务质量而重新设计互联网长达十年的努力。最后两部重要著作是为了维护互联网健康的有关拥塞控制方面的深入研究。互联网工程任务组（IETF）网站包含有所有定义互联网的 RFC 文件以及许多其他资源。

- J. Saltzer, D. Reed, and D. Clark, "End-to-End Arguments in System Design," ACM Transactions on Computer Systems, Vol 2, No. 4, pp. 277-288, Nov. 1984.
- D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *ACM SIGCOMM*, pp. 106-114, Aug. 1988.
- K. Hafner and M. Lyon, *Where Wizards Stay up Late: The Origins of the Internet*, *Simon & Schuster*, 1996.
- R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, Issue 7, pp. 395-404, July 1976.
- J. Postel, "Internet Protocol," RFC 791, Sept. 1981.
- J. Postel, "Transmission Control Protocol," RFC 793, Sept. 1981.
- M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.
- R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: An Overview," RFC 1633, June 1994.
- V. Jacobson and M. J. Karels, "Congestion Avoidance and Control," *ACM Computer Communication Review: Proceedings of the SIGCOMM*, Aug. 1988.
- S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, Vol. 7, Issue 4, Aug. 1999.
- Internet Engineering Task Force, www.ietf.org.

开源的发展

下面文献中的前两篇分别是第一个开源项目和第一篇有关开源的文章。第三篇是第一篇开源的文章扩展成的书。接下来的两篇文献是开源发展的概述，首先是有关技术方面的，其次是有关如何组织项目的努力。FreshMeat.net 是一个中心枢纽（hub）可用来从三大的开源软件包库中下载，而 SourceForge.net 则

容纳了许多开源项目，即使硬件也可能是开源的。OpenCores.org 是开源硬件组织中心（hub）。

- R. Stallman, The GNU project, http://www.gnu.org.
- E. S. Raymond, "The Cathedral and the Bazaar," May 1997, http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.
- E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, Jan. 2001.
- M. W. Wu and Y. D. Lin, "Open Source Software Development: an Overview," *IEEE Computer*, June 2001.
- K. R. Lakhani and E. Von Hippel, "How Open Source Software Works: 'Free' User-to-User Assistance," *Research Policy*, Vol. 32, Issue 6, pp. 923-943, June 2003.
- Freshmeat, freshmeat.net.
- SourceForge, sourceforge.net.
- OpenCores, opencores.org.

性能建模与分析

下面列出资料中的前两篇是 Agner Krarup Erlang 在 1909 年和 1917 年以丹麦文书写的有关排队论的第一部作品，其第二篇是 1961 年出版的称为 Little 结论的经典论文。Kleinrock 在 1975 年和 1976 年出版发表的书籍是第一部运用排队理论建立计算机和通信系统模型的经典作品。Leon-Garcia 的书是第一部建立在排队论建模基础上生成随机过程的教程。最后一篇是有关性能分析的额外、新增作品。

- A. K. Erlang, "The Theory of Probabilities and Telephone Conversations," *Nyt Tidsskrift for Matematik B*, Vol. 20, 1909.
- A. K. Erlang, "Solutions of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges," *Elektroteknikeren*, Vol. 13, 1917.
- J. D. C. Little, "A Proof of the Queueing Formula L = λW," *Operations Research*, Vol. 9, pp. 383-387, 1961.
- L. Kleinrock, *Queueing Systems, Volume 1: Theory*, John Wiley and Sons, 1975.
- L. Kleinrock, *Queueing Systems, Volume 2: Applications*, John Wiley and Sons, 1976.
- A. Leon-Garcia, *Probability, Statistics, and Random Processes for Electrical Engineering*, 3rd edition, Prentice Hall, 2008.
- R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley and Sons, 1991.
- T. G. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation*, 3rd edition, Springer-Verlag, 2000.
- L. Lipsky, *Queuing Theory: A Linear Algebraic Approach*, 2nd edition, Springer, 2008.

常见问题解答

1. 互联网如何才能成数以亿计的主机？（详细说明用于组织主机的结构和层次，并计算每层的实体数量。）

   答：将互联网分成三层结构，其中 256 台主机可以组合成一个子网，而 256 个子网可以组合成一个域，即有 65 536 个子网就总共有 40 亿台主机。

2. 路由与交换对比：哪个是有状态或无状态的，哪个是面向连接或无连接的，哪个是匹配或索引的？（将这些功能与路由器交换联系起来。）

   答：路由：无状态的，无连接的，匹配式的。交换：有状态的，面向连接的，索引式的。

3. 哪些因素可能增加或减少互联网中的延迟？（有哪些因素可能分别会增加或减少排队、传输、处理和传播延迟？）

答：封装：流量负载，网络带宽或 CPU 容量。传输：网络带宽。处理：CPU 功能。传播：链路/路径的长度。

4. 我们从 Little 结论和带宽延迟乘积可以知道什么？（提示：前者是一个节点，而后者是一个链路或路径。）

答：Little 结论：在一个节点中，分组的平均数量是平均分组到达率和平均延迟/延迟的乘积，即黑匣子的平均数等于平均速率乘以平均延迟。带宽延迟乘积：在传输过程中一条链路/路径上的突发位的最大数量。

5. 端到端观点是如何解释网络互联的？

答：如果问题不能在下层（或路由器上）得到完全解决，那么就在上层（或终端主机上）解决。这将复杂性从核心路由器推到了终端主机上。

6. 根据端到端的观点，我们应该把差错控制放在互联网上的哪一层？但是我们为什么还要把它放到许多层，包括链路层、IP 层和传输层上实现呢？

答：在端到端的传输层上，因为链路和节点的错误都可以检测并纠正，即链路层只能处理链路的错误但不能处理节点的错误。但为了更有效率，也将差错控制放在链路层和 IP 层以便及早处理错误。

7. 何种类型的机制应该分别放入控制平面和数据平面？（注明它们的分组类型、目的、处理时间的粒度和例子操作）

答：控制平面：控制分组，保持数据平面的正常操作，通常在数秒内完成，如路由。

数据平面：数据分组，正确地传输分组，通常在数微秒内转发。

8. 在路由器中，哪些是标准的组件及哪些是依赖于实现的组件？（指定组件的类型和例子。）

答：标准：影响路由器之间互操作的协议消息格式和算法，路由协议，如 RIP。

依赖于实现：不影响互操作性的内部数据结构和算法，路由表及其查找算法。

9. Linux 发布版本中都包括什么内容？（指出你将在发布版本中找到的文件类型以及它们是如何组织的。）

答：文件的类型：文档、配置文件、日志文件、二进制对象文件、图像文件、内核和应用程序包的源程序。组织形式：放入目录中。

10. 分别在什么情况下将网络设备机制实现到 ASIC、驱动程序、内核和守护程序中？（指出它们的指导原则和具体例子。）

答：ASIC：通常用于 PHY/MAC，有时用于 IP/TCP/UDP 以及上层的加速器，Ethernet/WLAN PHY/MAC 和 crypto 加密的加速器。驱动程序：通常用于 MAC 和 IP 之间的接口，有时也用于某些链路层，以太网/无线局域网的驱动程序和 PPP 驱动程序。内核：通常为 IP/TCP/UDP 层、NAT 和 TCP/IP 防火墙。守护程序：应用客户端，服务器或对等；Web 客户端，服务器和代理。

练习

动手练习

1. 访问 freshmeat.net、sourceforge.net、opencores.org，然后对它们具有的内容进行概括总结和比较。
2. 安装最新的 Linux 发行版本，并总结：1）它的安装过程；2）Linux 发行版内部包含的内容。
3. 首先的范围是 B，然后再查找源文件所在的 src/、/usr/src 目录或其他目录下的程序，具体根据正在使用的 Linux 发行版本而定。对目录中的内容进行总结和分类。
4. 按照附录 C 的指导使用 gdb 和 kgdb 调试应用程序和 Linux 内核。另外，分别使用 gprof 和 kprof 为应用程序和 Linux 内核编写配置轮廓。试写一篇报告有关你是如何做的以及在调试和配置轮廓编写中所遇到的问题。
5. 尝试附录 D 中描述的 hont、ARP、ping、tracerouts、tcpcump 等工具探索并概括总结网络环境。
6. 跟踪 Linux 内核代码，找到：

a. 在图 1-19 中，哪个函数调用 alloc_skb () 分别为请求和响应分配 sk_buff。
b. 在图 1-19 中，哪个函数调用 ktree_skb () 分别为请求和响应释放 sk_buff。
c. 在图 1-21 中，哪个函数调用 alloc_skb () 分配 sk_buff。
d. 在图 1-21 中，哪个函数调用 ktree_skb () 释放 sk_buff。
e. 你是如何动态地或静态地跟踪上述操作的。

7. 找到一个具有"标准"（STD）状态的 RFC 文件。
a. 回该并总结协议是如何在 RFC 中描述的。
b. 在 Linux 课例中或 Linux 发布中搜索某个源代码实现，描述该协议是如何在找到的代码中实现的。
c. 如果你准备从头开发一个开源实现，你将如何从 RFC 实现自己的开发目标？

书面练习

1. 假设一条具有 40Gbps 带宽的 5000 英里的拥塞链路。假设传播速度为 2×10^8 米/秒。
a. 位在时间和长度上的宽度分别是多少？
b. 链路最多可以包含多少位？
c. 一个 1500 字节的分组需要多长的传输时间？
d. 通过这条链路的传播时间是多少？

2. 一个分组位在互联网主沿着具有 10 条链路和节点的路径传播。每条链路长 100km，拥有 45Mbps 容量，传播速度为 2×10^8 米/秒。假设没有流量控制，路径上也没有其他流量，源以线速度发送分组。
a. 在每条链路上将包含多少位？
b. 如果通过每个节点的平均延迟为 5 毫秒，则在每个节点平均包含多少位？
c. 在路径上平均包含多少位？

3. 假设有一条 1Gbps 的链路，具有指数级增长的分组到达间隔时间和服务时间。我们想运用排队论和 Little 结论来计算平均等待时间和平均占用时间。
a. 如果平均到达速率是 500Mbps，则平均延迟、排队时间、占用时间各是多少？
b. 如果链路带宽和平均到达速率分别增加幅度为 10Gbps 和 5Gbps，则平均延迟、排队时间和占用时间分别是多少？

4. 如果 30% 分组的大小为 64 字节，50% 分组的大小 1500 字节，其余的大小均匀分布在 64 – 1500 字节。则在一台拥有 12 条 10Gbps 链路的路由器上每秒汇聚分组的最大数量（PPS）是多少？

5. 假设每分钟有 300 万个新的电话呼叫到达全球范围的电话交换系统，平均每次通话时间 5 分钟，并且每个呼叫用户和被叫用户之间平均有 6 跳（即 6 条链路和 6 个节点）。那么为了支持全球交换连接平均需要占用多少内存表项？

6. 在一个拥有 4 294 967 296 个节点的集群中，如果仍然要保持如图 1-1 中所示的三个层次，但仍希望有相同数量的组成员，组和组上的超组、超组，以及"超超组"层，那么近似的数量是多少？

7. 如果由于 IP 地址短缺，我们将图 1-1 中的组和超组一分为二，每组最多有 128 个成员，每个超组最多有 128 个组，则最多允许多少这样的超组？

8. 比较数据通信和电信（语音）通信的需求和原则之间的差异。对最重要的三个差异进行命名并加以解释。

9. 为什么互联网被设计成路由式而不是交换式网络？如果将它设计成一个交换式的网络，则需要改变哪些层次和机制？

10. 比较路由分组和交换分组的开销。为什么路由时间复杂度高于交换，而交换的空间复杂度比路由分组高？

11. 如果要在路由器上支持新的路由协议，需要将哪些内容定义为标准，哪些内容应该留作与实现相关的设计？

12. 内容网络互联要求互联网本身成为更加具有应用感知的，即要求知道谁访问什么数据并与谁交流通信。但这会破坏原有的端到端观点。需要采取什么样的改变来支持内容网络互联？

13. ATM 和 MPLS 没有无状态的核心网络。它们保持了什么状态？它们保持这些状态的方式上有什么主要区别？
14. ATM 是一种用于数据通信的可选技术。为什么当它与 IP 互换作承载传输 IP 分组时具有很高的开销？
15. MPLS 示 IP 交换的一种标准，旨在交换多数但路由少数的 IP 分组。将其实施的障碍是什么？如何减少这一障碍的影响？
16. 当支持某个协议时，我们可能将协议实体放入内核或守护程序中。这里则需要考虑什么问题？主要是说，分别在什么时候将协议实体放入内核和守护程序？
17. 在图 1-13 中，我们为什么将路由的任务（业务）放入用户空间而同时将路由表查找放入内核中呢？为什么不能将两者都放入用户空间或内核空间中呢？
18. 当读者为网络适配器编写驱动程序时，应该将哪些部分写入到中断服务例程中？哪部分不应该写入？
19. 当读者实现数据链路协议时，分别应该将哪些部分实现到硬件和驱动程序中？
20. 我们需要了解硬件如何与驱动程序一同工作。
    a. 网络适配器的驱动程序与网络适配器控制器之间的接口是什么？
    b. 驱动程序如何要求控制器发送分组，控制器如何报告它已经完成工作？
    c. 当分组到达网络适配器时，控制器如何将它报告给驱动程序？
21. linux、apache、sendmail、GNU C library、bind、freeS/wan 和 encrl 是流行的开源软件包。在互联网上搜索它们的许可证模式，并概述这些许可证模式之间的差异。
22. 当用户在浏览器中输入一个 URL 时，会在几秒钟内得到相应的网页。分别简要描述在主机、中间路由器以及相关服务器上会发生什么。书写答案之前为了使你的答案更为准确，请参阅 1.5.2 节，但不一定要求运行在 Linux 系统上。

第2章

Computer Networking: An Open Source Approach

物理层

物理层（PHY）是计算机网络中OSI模型或TCP/IP模型中的最低层，它是唯一与传输介质相互作用的层。传输介质是一种可以将称为信号的能量波从发送端向接收端传播的物质实体。而且，也可以将自由空间认为是一种电磁波的传输介质。传输介质只能传输信号，而不是数据，但源于（或来自）链路层的信息属于数字数据。因此，物理层必须将数字数据转换成适当的信号波形。在现代数字通信中，这种转换是一种分成两步的过程。首先对数字数据进行信息编码以便于进行数据压缩和保护，然后将编码过的数据调制或进于在通信介质中传输的信号。应当说明的是，在模拟通信中只使用后一种调制过程。

为了能够实现高速传输，物理层需要根据介质的属性来决定编码和调制技术。一般来讲有线介质更可靠，因此物理层可以完全看限于提高其吞吐量和利用率。相反，无线介质是不可靠的并暴露在公众之下，因此物理层需要能应付噪声干扰并防止数据被损坏。除了需要提高吞吐量和利用率外，还需要有能够处理充满了噪声、干扰，甚至多径衰落介质的技术。

在一种介质上可以有多个信道共存。发射器和接收器之间的信道既可以是物理的也可以是逻辑的。在有线网络中，物理信道是一条通过电缆的传输路径；而在无线网络中，物理信道就是电磁波频谱的频率波段。逻辑信道是传输介质经不同的划分方法划分成的子信道，如时分、频分、码分和空分。因此，需要利用另一种称为多路复用的技术以便更好地利用介质。

本章将介绍基本的转换技术。在2.1节中，我们首先要介绍模拟数据/信号和数字数据/信号之间的差异。接下来说明发送和接收流、通过编码和调制进行数据/信号转换、通过多路复用以便更好地利用，此处还介绍了损害信号的因素。2.2节将传输介质分成两类：有线的和无线的。在2.3节中介绍了各种线路编码（或称为数字基带调制）技术，以便达到更好地发送方和接收方时钟同步。介绍了不归零（NRZ）、曼彻斯特、交替反转（AMI）、多级传输（MLT-3）、1B/5B等经典技术。还介绍了8B/10B编码的开源实现。

2.4节介绍了各种数字调制技术，包括幅移键控（ASK）、频移键控（FSK）、相移键控（PSK）、正交调幅（QAM）。调制是将数字位流通过模拟带通信道传播，模拟载波信号由数字位流调制。换句话说，就是把编码过的数据转换成带通信号，即一段真正（或复杂）的连续时间波形，以便于数字传输。由此产生的信号是一个以载波频率为中心的包含在有限带宽范围内的真正的连续时间波形。接下来我们介绍基本的多路复用技术，包括时分多路复用（TDM）、频分多路复用（FDM）和波分多路复用（WDM）技术。

在高级的主题首到2.5节讨论，包括扩频、码分多址（CDMA）和正交频分多址（OFDM）、多输入和多输出（MIMO）。扩频的目的包括抵抗、多路访问和隐私保护。这些都通过将源数据位传播到具有较高码片速率和较低能耗的芯片（码片）序列中得以实现。直序扩频（DSSS）、跳频扩频（FHSS）和CDMA就是三种例子。OFDM是一种利用多个载波的数字通信技术。MIMO通信是一个在发射器和接收器都使用了多个天线的新的传播媒介。MIMO通过引入空间复用和空间分集可以提高通信的可靠性和吞吐量。最后，我们讨论了采用OFDM的IEEE 802.11g发射机的开源实现。

2.1 一般性问题

物理层通过传输介质发送信号和接收信号。为了生成一个可以通过具有信道高吞吐量和利用率的特殊介质进行发送和接收的信号，必须解决几个方面问题。首先，从链路层来的数据需要转换成数字信号或模拟信号以便进行数字传输。我们首先把模拟数据/信号和数字数据/信号区分开来。其次，发送和接收流量在物理层经历多次转换。这两种流量需要加以说明。再次，需要编码和调制。为了进一步提高信道的利用率，我们需要多路复用和多路访问等技术，以便使多个用户能够访问相同的信道，

这是我们要解决的第四个问题。最后，为了对信道损害做尽可能，特别是在无线介质中，就必须实现一些补偿措施。

2.1.1 数据和信号：模拟的或数字的

数据和信号既可以是模拟的也可以是数字的。在计算机中，数据通常是数字的，而模拟数据（如语音和视频）通常需要转换成数字值以便于存储和通信。这是因为以模拟信号形式表示的模拟数据很容易受到噪声的影响。数字数据和信号可以通过再生中继器再生并通过纠错码保护免受侵害，所以它们对噪声有更强的抵抗力。因此，模拟数据通常转换为位流形式的数字数据。然后将它们转换为用于传输的信号。因此，在计算机网络中使用数字数据表示模拟信号源，如图像、声音、音频和视频。

在计算机网络中，位流或消息，通过传输介质从一台计算机经网络连接到另一台计算机。传输介质沿着一条物理路径传递信号能量，它们既可以在电缆中传输电信号，在光纤中传输光信号或存在自由空间中传递电磁信号。通常，模拟信号比数字信号传播得更远更持久。物理层起着将数字数据转换成数字信号或模拟信号以适用于特定传输介质的作用。在此我们说明了数据和信号、模拟和数字之间的差异。

模拟数据和信号

模拟信号是一种包含由模拟源生成的模拟信息的连续时间信号，如声音或图像。它常常具有连续的值。模拟通信的一个例子就是发声—听觉通信系统。可以对模拟信号进行采样并量化成数字数据用于存储和通信。

数字数据和信号

在计算机中数字数据取离散值，如0和1。它们可以转换为数字信号，能够直接传输一段短距离。另外，也可以调制成波（即周期性的模拟信号），使调制过的信号能够传输较远的距离。大多数教科书将调制信号当做数字信号，因为它们将数字调制方式当做一种数字传输或数据传输形式，即使调制是一种数模转换形式时也是如此。通过在离散时间采样并量化成离散值，模拟信号可以转换为数字信号。换句话说，经采样的模拟信号变成了离散时间信号，它可以进一步量化成数字信号。如果一个波形具有两个电平来表示二进制状态“0”和“1”，它就是一种二进制的数字信号，代表一个位流。这里，我们更正式地定义几个术语。

采样是一种在离散时间从连续时间信号（或者在图像处理中的连续空间）中提取样本的过程。每个采样值在采样周期内保持恒定不变。例如，一个连续时间信号 $x(t)$，其中 t 是定义在连续时间实线上的一个变量，可采样成离散时间信号，在采样时刻的样本值可以由数字序列或一个离散时间函数 $x[n]$ 表示，其中 n 是一个整数集的值用来表示离散时间的离散变量。采样信号是一种具有连续值的离散时间信号。

量化是将一段值范围映射到一个离散有限数字或值的集合的过程。这样的映射过程通常是通过使用模数转换器（ADC）进行的。一个量化的信号可以在连续的时间上，但具有离散值。当然量化会引入量化误差或量化噪声。

重建是从采样的离散时间信号恢复为原来的连续时间信号的插值过程。为了从一个取样序列中完美地重建原始信号，采样率只需要等于或大于原始信号中最高频率的两倍。这种充分条件是奈奎斯特—香农采样定理的一个结论。

行动原则：奈奎斯特定理与香农定理

一条通信信道是介于发送端和接收端之间的一条连接，其中信息是通过电缆、光纤或无线电频率频谱作为传输介质的一条路径传输的。信道可以是无噪声的也可以是有噪声的。如果认为信道是无噪声的，其最大数据传输速率可由奈奎斯特定理来决定；如果是有噪声的，最大数据速率则由香农定理来确定。

准确地重构信号所需的采样率是多少？在无噪声的信道上传输信息时，最大数据传输速率有多大？这些问题，由哈里·奈奎斯特（Harry Nyquist）于1924年提出，它们后来由奈奎斯特采样定理及推导

理由给出最大数据传输速率的界限。作为香农研究的结果，奈奎斯特准则仅仅适用于无噪声的信道。不过，奈奎斯特准则的一个结果，系统必须至少以信号带宽的两倍进行采样。例如，如果电话线的信道信号具有的最高频率为f_{max}，采样率f_s必须至少为$2\times f_{max}$。香农通过定理表明，具有带宽为B(Hz)的无噪声信道的最大数据率为$2\times B\log_2 L$，L是表示符号的信号级别数目。例如，如果应用一条3kHz的无噪声电话线和1位信号编码（两个状态），那么语音通过电话线传送时可能达到的最大的数据速率为多少？根据奈奎斯特定理，最大数据速率为$2\times 3k\times\log_2 2$kbps，或6kbps。

实际上，信道不可能是无噪声的，而是具有许多不想要的噪声，如热噪声、互调噪声、串扰噪声、脉冲噪声。这就需要有一个能够计算在噪声气氛下最大数据速率的新定理。1948年，克劳德·埃尔伍德·香农在噪声信道下计算最大数据速率提出了“一种通信数学理论”和“噪声环境下的通信”。香农定理指出，如果信号信噪比（SNR）为S/N，带宽为B(Hz)的有噪声信道上传输，那么最大数据速率是$B\times\log_2(1+S/N)$。香农定理也称为香农极限。此极限与编码方法无关，但它与信噪比有关。而且，考虑3kHz有噪声的电话线，如果信噪比为30分贝（dB）时，最大数据速率为多少？根据香农极限，最大数据速率是$3k\times\log_2(1+1000)$kbps或29.9kbps。

周期性和非周期性信号

如前所述，一个信号既可以是模拟的也可以是数字的。如果它是连续时间和连续值，那么它就是一个模拟信号。如果它是离散时间和离散值，那么它就是一种数字信号。除了这种区分外，信号也可以分为周期性的或非周期性的。周期性信号是一种经过一定时间重复本身的，而非周期性信号则不会重复。模拟和数字信号既可以是周期性的也可以是非周期性的。例如，一个人的语音声音信号是非周期性的模拟信号，数字时钟信号是一种周期性的数字信号。除了信号的时域描述外，还有一种基于傅里叶理论的频域描述方法。如果信号是由可能的无限个正弦频率构成的线谱，那么它就是周期性的。线谱是一种能够集中在特定波长的频谱。另一方面，如果信号具有可能无限的支持的连续频谱，那么它就是非周期性的。此外，如果信号具有有限的支持，那么就说它是频带有限的。就说它刚好是f_1~f_2的频率。图2-1显示了模拟信号的频谱。图2-1a使用离散频率100kHz和400kHz来表示具有不同振幅的两个周期性模拟信号。在图2-1b中，显示了非周期性有限带宽的模拟信号。

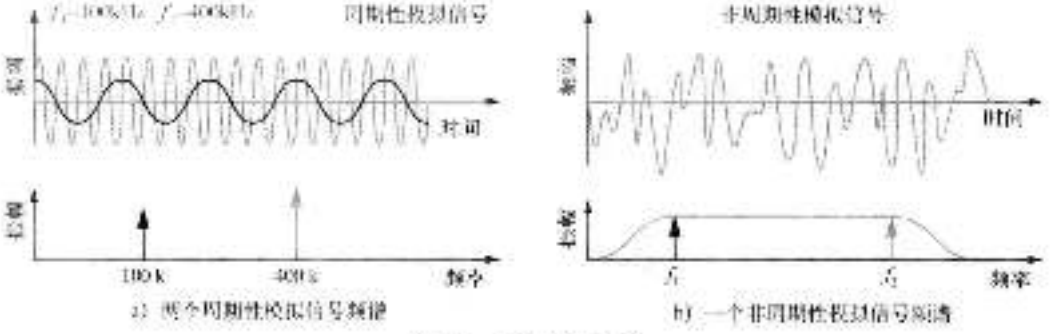


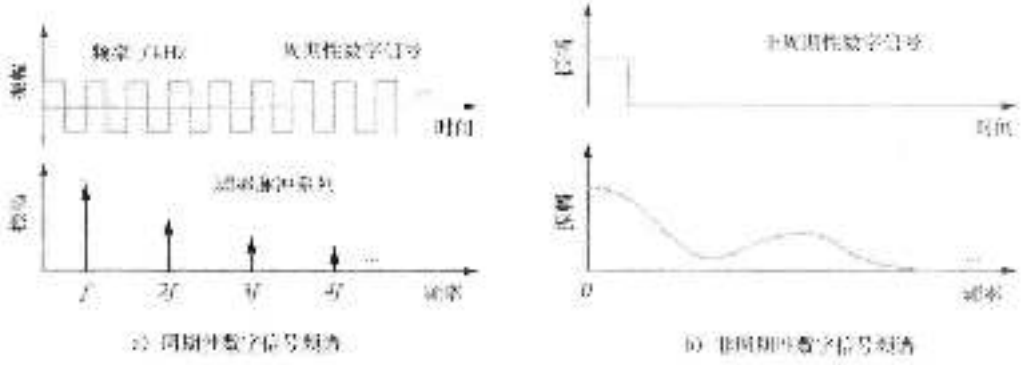
图2-1 模拟信号频谱

图2-2描述了数字信号的频谱。根据傅里叶理论，周期性的数字信号具有一种通过正弦（sine）频谱乘以由离散频率脉冲序列组成的周期性线谱而获得的线谱。非周期性的数字信号具有连续频谱，是通过正弦频谱乘以从零到无限范围的周期性连续频谱而获得的。傅里叶理论还告诉大家，数字信号可以由加权的不同频率、振幅和相位的正弦曲线组合。正弦和余弦，信号表示。将图2-1和图2-2结合起来，我们就可以得出如下结论：

- 如果信号是周期性的，那么它的频谱是离散的；如果信号是非周期性的，那么它的频谱是连续的。
- 如果信号是模拟的，那么它的频谱是非周期性。如果信号是数字的，那么它乘以正弦函数后，它的频谱是周期性的。

在数字通信中，经常用到周期性模拟信号或非周期性数字信号，因为周期性模拟信号需要较少的

带宽，非周期性数字信号可以表示数字数据的各种值，如图2-2a和图2-2b所示。在本章的其余部分，若没有明确的说明，数字信号意味着表示数据流的非周期性数字信号，时钟信号是指周期性数字信号，视频是指周期性模拟信号，而词组信号表示非周期性模拟信号。



a) 周期性数字信号频谱　　b) 非周期性数字信号频谱

图2-2　数字信号频谱

2.1.2　发送和接收流

在解释了模拟和数字信号的性能并区分了周期性和非周期性信号的各个方面后，我们接着在图2-3中说明通过物理层简化的发送和接收流。来自一个信息源的消息符号首先被信源编码压缩，然后通过信道编码将它编码为信道符号。符号表示一定长度的二进制元组。消息符号是来自信源的一序列数据流。信道符号表示已经被信源编码和信道编码处理过的数据流，并且可能和从其他源来的符号复用。然后合并后的信道符号通过线路编码（或数字基带调制）处理成基带波形。接下来基带信号可以直接通过有线网络（如电缆）传送给接收器，或者它也可以进一步与载波一起通过数字调制并经过无线网络传输。调制信号是一种带通波形，一种来自数字调制的带通信号，用于数字传输。（如果调制信号传输数字数据而不是模拟数据，许多教科书认为它是一种数字信号，而不是模拟信号。）最后，在数字通信系统中的发射机将带通波形（仍然是一个基带信号）转换成发送的信号，即RF（射频）信号。发射的信号，与干扰和噪声一起通过信道发送出去。

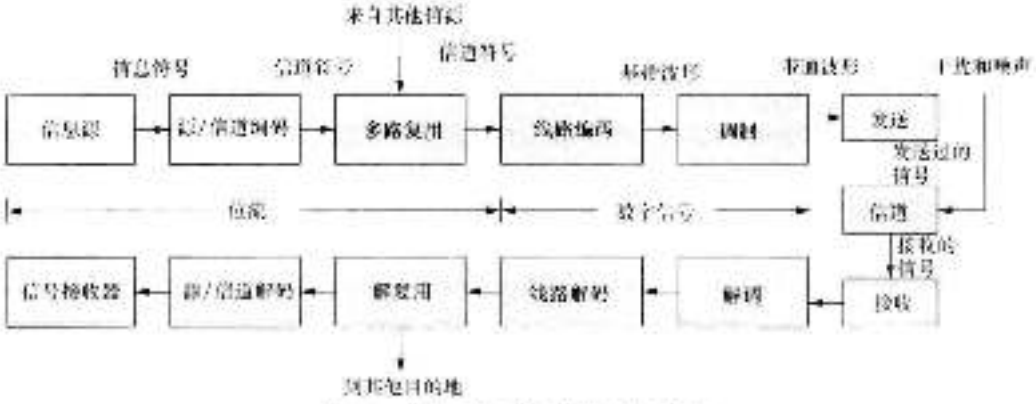


图2-3　数字通信系统的发送和接收流

多路复用将资源分成多个信道，以便共享总容量大于单独一个数据流需求的传输设施来提高信道的利用率。它结合了其他数据的数字流或通带信号的数字信号。因此多路复用可以发生在不同的地方。多路复用可以按照频率、时间、编码或空间通过频分多路复用（FDM）、时分多路复用（TDM）、码分多路复用（CDM）、空分多路复用（SDM）创建逻辑信道。多路复用方案的不同在于它们如何将物理信道划分成多个信道或逻辑信道。FDM技术是一种模拟技术。而TDM和CDM是数字技术。因此，TDM或CDM的位置可以放在图2-3中所示的多路复用/解复用模块。因此，TDM和CDM的位置可以

于多路复用/解复用的模块，如图2-3所示，FDM是通过合并其他信号并共享信道后经过通带调制产生而成。通信系统可以通过TDM建立多个信道，这些信道之一可以由一组用户使用特定的多路访问方案（如载波侦听多路访问（CSMA））来访问。需要注意的是多路复用方案是在物理层上提供的，而多路访问技术是在能源层决定的。

基带或宽带

图2-3所示的基带波形是可以在一条基带信道上直接传输而不需要转换成模拟信号的数字信号。这称为基带传输，它绕过了通带调制。如果信道是宽带信道，数字信号就需要使用与简单的线路编码调制不同的调制方式。宽带是指在比数字信号频率高很多的频带上的数据传输，这样就可以使多个数据流能够同时发送，多个信号也可以共享相同的介质。

如前所述，非周期性数字信号具有一种由正弦函数来以一个周期性连续频谱而得到的频谱。频谱的振幅下降，在高频率处接近于零。因此，在高频率的频谱可以忽略不计。在基带或宽带传输的消息取决于传输介质和信道的属性：

- 如果一个物理信道是低通宽带信道，数字信号就可以通过信道直接传输。这是由于高频率的损失使接收到的信号只有轻微的失真，可以在接收端恢复。这种基带传输处理非周期性数字信号如图2-2b所示，其高频成分具有较低的振幅并且可以忽略不计。
- 如果一个物理信道有一个不从零开始的有限带宽，那么信道就是一个带通信道。通过带通信道传输的消息需要一个承载传输消息的载波和一个在信道上传输的带通波形调制信号（称为通带信号）。通带信号的频率以载波频率为中心。这就是宽带传输。宽带传输承载数据穿越带通信道，其中数字基带信号必须经过调制转换成一种通带信号。在数字传输中，通带信号被认为是数字信号，但其波形是非周期性的频谱占用有限带宽的模拟信号形式，如图2-1b所示。

2.1.3 传输：线路编码和数字调制

在通信世界中，物理层利用各种编码调制技术将数据转换成信号，以便于消息在物理信道上传输，并且信号也能够通过传输介质。在计算机网络中，重点强调线路编码和数字调制技术。前者将位流数字信号转换为基带信道，而后者为带通信道将数字基带信号转换成带通信号。无论是线路编码还是数字调制，两者都是为了满足数字传输或数据传输的同一目的，但它们需要进行不同的转换。

同步、基线徘徊（漂移）和直流分量

线路编码，又称为数字基带调制，采用离散时间离散值信号，即方波或数字信号，只能通过振幅和定时来描述发送的0和1。然而，在数据流中，一长串不改变信号值的相同位值的序列可能会导致接收器时钟同步的丢失和漂移离开基线。

自同步可以用来校准接收器的时钟，以同步发射器和接收器之间的位间隔。基线用于为数字数据确定接收到的信号值。基线徘徊或漂移，使解码器更难确定接收到的信号数字值。同时，不归零（NRZ）等编码技术还可能引进直流（DC）分量。这就使得数字信号在0赫兹具有非零的频率分量，即直流分量或直流偏差。

将这种编码应用到位值相同的一个序列，不仅会产生同步问题，而且还会产生无相位变化的恒定电压的数字信号。与直流平衡波形的信号（不含直流分量）相比，具有直流分量的信号会消耗更多的功率。而且，某些类型的信道不能传输直流电压或电流。在这样的信道上传输信号，就需要没有直流分量的线路编码方案。

总之，线路编码的主要目的是防止基线徘徊，消除直流分量，激活自同步，提供错误检测和纠正，并提高信号对噪声和干扰的免疫能力。

振幅、频率、相位和编码

数字调制使用由振幅、频率、相位或编码描述的连续时间或离散时间的连续值信号，或模拟信号，表示来自一个信息源的位流。可将数字位流转换成一些带通信号。该带通信号可通过某一载波频率为中心并具有有限带宽的带通信道用于远距离传输。例如，通过无线信道传输消息需要线路编码和数字调制处理，这样消息可以由载波传输，同时其调制信号经过带通信道通过自由空间传播。利用振幅、

频率、相位、编码及它们的组合，就可以开发出多种数字调制技术。复杂调制技术的通用目标是在低带宽和充满噪声的信道上以较高的速率传输。

此外，线路编码或数字调制可被优化以适应任何给定介质的特点。例如，在无线通信中，链路自适应或自适应编码和调制（ACM）是与编码和调制的方法以及针对信道条件的通信协议的参数相匹配。

2.1.4 传输损失

传输介质并不是完美的。接收到的信号并不是与发送的信息完全一样。有多种因素可能损害介质传输的可靠性，如衰减、衰落、失真、干扰或噪声。这里对传输损失及其补偿措施进行讨论。

衰减：衰减是无线电波或电信号的通量强度逐渐损失的过程。衰减影响波和信号的传播。当信号通过介质传播时，因为传输介质具有电阻，它就会损失一些能量。例如，电磁波被水粒子吸收或在无线通信中发散，电磁辐射强度就衰减了。因此，在发射器和接收器两端都需要低噪声放大器来放大信号，以便经过某种处理后能够检测并恢复原来的消息。放大是一种对付衰减的手段。

衰落：在无线通信中，通过某种介质传输调制波形时会经历衰落。衰落是衰减随时间变化的偏差，因为它会随时间、地理位置或无线电频率而变。有两种类型的衰落：由多径传播引起的多径衰落；障碍物遮蔽而产生的遮蔽衰落。一个经历衰落的信道称为衰落信道。

失真：接收信号的形状可能与最初的信号不完全一样。这种失真通常发生在复合信号中。传播后，复合信号的形状就失真了，因为复合信号是由具有不同频率的信号组成，各种频率会经历不同的传播延迟。这样会产生不同的相移，从而产生扭曲了的信号形状。数字信号通常由多个周期性模拟信号构成的复合模拟信号表示。因此，数字信号在传输后往往会失真，不可能传输相同。为了弥补这一损失，人们会使用适合远距离传输的模拟信号波形。

干扰：干扰与噪声有着显著不同。它是指破坏通过信道传输信号的任何东西。它通常会给有用信号增加不需要的信号。著名的干扰明了包括同频干扰（CCI），又称为串扰；符号间干扰（ISI）；载波间干扰（ICI）。

噪声：噪声是模拟信号的随机波动。所有电子电路上都会有电子噪声。热噪声，或奈奎斯特（Nyquist）噪声，是由电荷载体的热扰动产生的。它常常是白噪声，即功率谱密度几乎与整个频谱一致。其他各种噪声包括感应噪声、脉冲噪声和量化噪声。噪声影响接收器恢复传输数据的能力。感应噪声来自家用电器等。脉冲噪声来自电源线或闪电，而量化噪声由量化误差导致的。信噪比（SNR）定义为平均信号功率与平均噪声功率之比，这是限制理论位速率的一种测量。为了弥补噪声对传输数据的影响，我们既可以提高信号的功率也可以降低传输位速率。另一种方法是使用对噪声具有更高价格性的调制技术。

因为在传播过程中信号强度的衰落，物理层一般将位流或数字波形转换成调制的通带信号，并通过物理信道发送信号。这些转换技术（编码和调制）将减轻通信系统上的上述损失。在接收端，对信号检测、解调、解码并恢复出原始数据。换句话说，数字通信系统需要能通过噪声信道传输信息，过滤噪声并从传播衰落中恢复信号的能力。

历史演变：软件定义的无线电

在传统的无线电系统中，信号一般都是由硬件（如ASIC芯片）处理而不是由软件处理。因为通用处理器的硬件技术已经发展到一个新的水平使得实时地处理信号成为可能，于是就出现了软件定义的无线电（SDR）或软件无线电。软件无线电的概念最先由J. Mitola于1991年提出。以此传统系统更低的费用使它适应于多种无线标准，从而极大地提高了无线电系统的灵活性。

在传统的和SDR的通信系统中，信号处理流程是相同的。区别在于信号是在哪里数字化然后再被软件处理。图2-4说明了一个经过一系列无线电器最可以实现各种无线标准的无线节点。与图2-3相比，图2-4扩展到包括了分别用于操作RF波形和IF波形的中介频率（Intermediate Frequency，IF）处理单元和无线电频率（Radio Frequency，RF）信道接入。RF，范围为3kHz～300GHz，是一组震荡载

或集合；而 IF，范围为 10～100MHz，是由 RF 和本地振荡器（LO）频率混合所产生的一个较低的、更容易处理的频率。

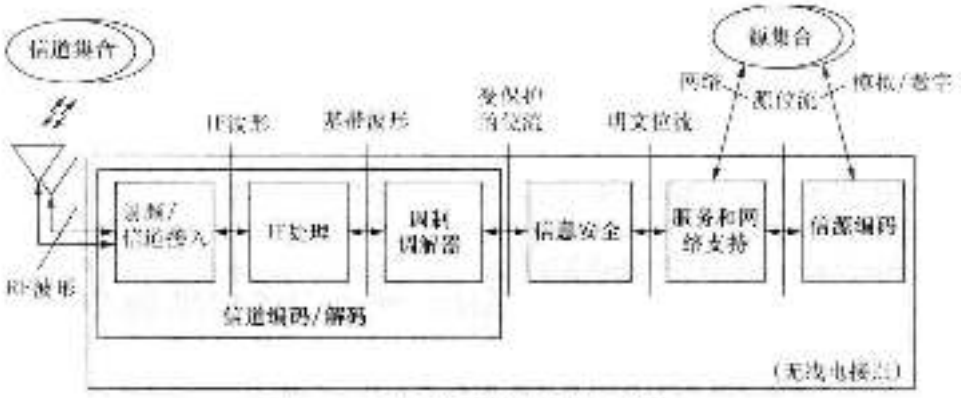


图 2-4 无线通信系统信号流的功能模型

在无线通信系统中，数字信号在发射机首先调制成带通波形（分配给带频率范围内），然后向上转换成 IF 和 RF 波形以便于在无线信道上传输。在接收端，接收到的 RF 波形首先被信频/信道接入模块处理，然后转换成 IF 波形，再向下转换成基带波形并进一步解调解码为位流。在 SDR 中，信号数字化可能发生在 RF、IF 或基带波形中，分别称为 RF 数字化、IF 数字化或基带数字化。RF 数字化是 SDR 在软件中完全处理整个无线电范围的一个理想位置。然而，因为高速宽带 ADC 和通用处理器计算能力的硬件限制，软件无线电很难实现 RF 数字化。此外，基带数字化不视为软件无线电系统的一部分，因为这样做没有任何好处，这一点与传统通信系统的数字化相同。因此，IF 数字化成为 SDR 数字化的最佳选择。

目前已经为软件无线电系统开发了多个公共软件无线电项目，如 SpeakEasy、联合战术无线电系统（Joint Tactical Radio System，JTRS）和 GNU Radio。GNU Radio 项目开始于 2001 年，由 Eric Blossom 开发，致力于构建最小硬件需求的无线电系统。GNU Radio 是一个开放源代码的开发工具包，提供了 C++ 信号处理模块库并与 Python 脚本语言可以用于构建软件无线电。GRC（GNU Radio Companion），一种 GUI 工具，允许用户以类似于 LabVIEW 或 Simulink 的方式连接信号处理模块，同时构建无线电系统。GRC 可以使得 GNU Radio 的研究，更加方便，并大大降低了学习曲线。通用软件无线电外设（Universal Software Radio Peripheral，USRP），由 Matt Ettus 开发，是目前 GNU Radio 最流行的硬件平台。

2.2 介质

传输介质被物理层用于在发射端和接收端之间传输信号。对于无线介质来说，是自由空间；对于有线媒体介质来说，包括金属和光纤。由于我们采取的编码和调制技术可能部分地依赖于传输介质的类型，所以我们首先考察这些传输介质的特点。另一个影响选择何种技术的因素是介质的运行成本，这很大程度上取决于距离和周围环境噪声情况。

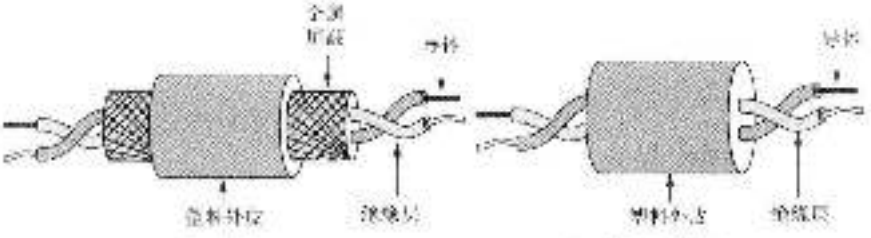
2.2.1 有线介质

常见的金属和光纤电缆有线介质包括双绞线、同轴电缆和光纤。通过这些介质传输的信号无论是电或光，都是有方向性的并受物理介质属性的限制。

双绞线

双绞线由缠绕在一起的两根导体组成，以防止来自外部的电磁干扰和线对间的串扰。双绞线既可以是屏蔽的也可以是非屏蔽的。屏蔽电缆称为屏蔽双绞线（STP），非屏蔽电缆对称为非屏蔽双绞线（UTP）。STP 和 UTP 的结构如图 2-5a、b 中所示。STP 有一层额外的金属屏蔽提供对电磁干扰的额外保护。但 UTP 双绞线由于其成本较低而很常见。由于技术的进步，UTP 足以满足实际使用。根据最大允

许的信号频率对双绞线进行分类。表2-1总结了在ANSI EIA/TIA标准568（美国国家标准学会、电子工业协会、电信行业协会）中常见的规格。生高的类意味着对铜导体每英寸有更多的缠绕重数从而能够维持更高的信号频率，因而具有更高的位速率。长度限制要根据目标的位速率而定。电缆越短，支持的位速率就越高。



a) 屏蔽双绞线，STP　　b) 非屏蔽双绞线，UTP

图2-5　双绞线电缆

表2-1　常用双绞线电缆规范

规　范	描　述	规　范	描　述
1/2类	主干传统的电话线，没有在TIA/EIA中规定	5(e)类	规定传输特性高达100MHz
3类	规定传输特性高达16MHz	5(e)类	规定传输特性高达250MHz（Cat-6）和500MHz（Cat-6a）
4类	规定传输特性高达20MHz	7类	规定传输特性高达600MHz

为了以更高位速率传输，人们既可以使用支持更高频率的电缆也可以设计更复杂的编码调制方案，以便在同一段时间内编码更多的位。虽然可以设计出复杂的编解码器或调制解调器在低频率信号传输数据，但因电路成本过高而使这种设计不实用。随着近年来电缆成本的降低，通过更好的电缆传输而不是依靠复杂的编码或调制方案会更经济。例如，虽然的确存在3/4类双绞线传输超过100Mbps的以太网技术，但很少在实践中应用。几乎现有所有的100Mbps以太网接口都运行在5类电缆的100Base-T。

同轴电缆

同轴电缆由内导体（铜芯）、绝缘层、编织外层导体，再加上一绝缘层和一个塑料外套组成，如图2-6所示。同轴电缆在许多应用中广泛常见，如有线电视网络和使用电缆调制解调器的宽带互联网接入。它曾经是用于以太网的流行媒介质，但它已被双绞线和光纤所替代。

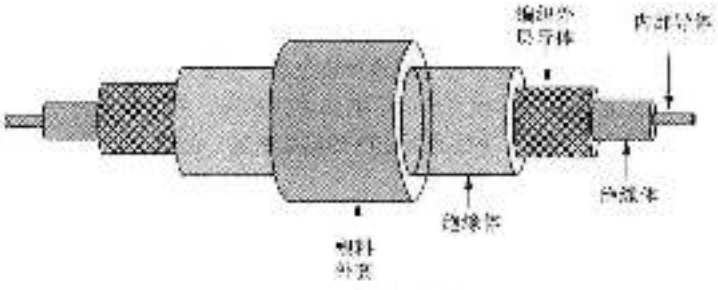


图2-6　同轴电缆

不同类型的同轴电缆具有不同的内部和外部参数，相应地也会影响阻抗等传输特性。最通用的类型是RG-6，其中芯径为0.0403英寸，可以工作在大约3GHz。

光纤

光能从一种透明的介质传入到另一种介质，但光的传播方向会发生变化。这种现象称为光的折射。方向改变多少要根据介质的折射率。光在真空中的速度与在介质中的速度之比来确定。这种折射现象的关系，即斯涅耳（Snell）定律，是由Willebrord Snell推导出来的。斯涅耳定律规定 $n_1\sin\theta_1 = n_2\sin\theta_2$，如

图 2-7 所示。当光从一个更高折射率的介质到另一个较低折射率的介质时，光可以折射 90°，即折射角。这时入射角为临界角，θ_c，如图 2-7 所示。如果光在这两种介质中的入射角大于 θ_c，它就不会进入第二种介质，将反射回第一种介质中。这称为全内反射（或全反射）。光纤应用就是基于全反射的原理。

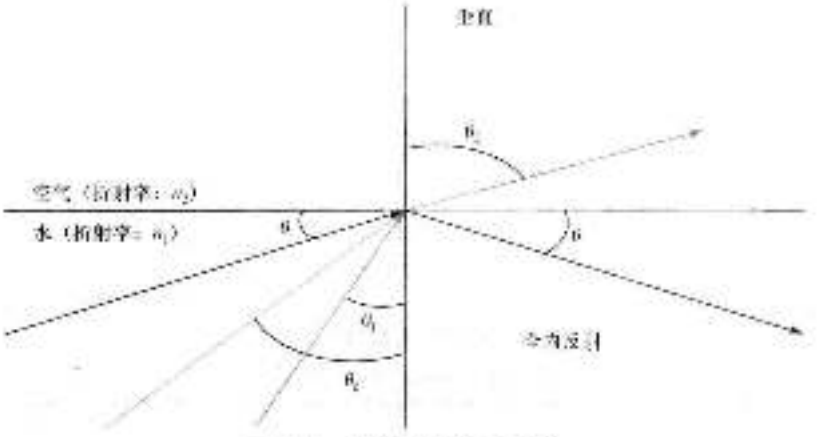


图 2-7 光线折射和全内反射

光纤以光的形式沿光缆内芯传播信号。由于全内反射，光线就可以保持在芯中。光源可以是发光二极管（LED）也可以是激光。光纤的结构如图 2-8 所示，在细的玻璃或塑料芯上覆盖了一层不同密度的包层玻璃，然后外面封套。包层介质具有低的折射率，芯的介质则具有较高的折射率。

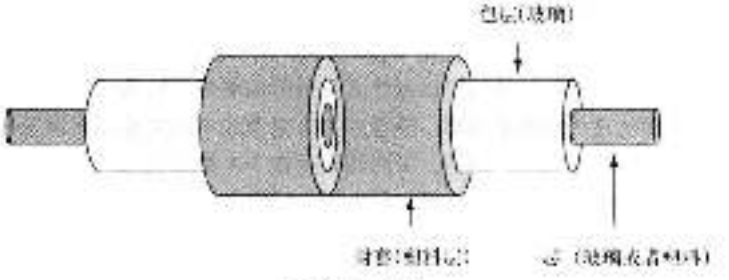


图 2-8 光纤

经过光纤的不同光传导形式称为模式。如果一根光纤以多于一种的模式以特定波长承载光，它就称为多模光纤。如果光纤非常细，只能允许传输一种模式通过芯，就称为单模光纤。图 2-9 显示了两种主要的光纤，分别是多模光纤和单模光纤。多模光纤有一个较粗的芯（通常大于 50 微米），光是通过反射而不是在一条直线上传播。尽管有更便宜的发射器和接收器，但由于光信号传播速度的多样性，多模光纤会引入较高的噪声水平。色散限制了多模光纤的带宽和通信距离。单模光纤有一个更细的芯（通常小于 10 微米），它强制光信号沿直线前进。单模光纤能够传输得更远，速度也会更快，但制造成本会更高。

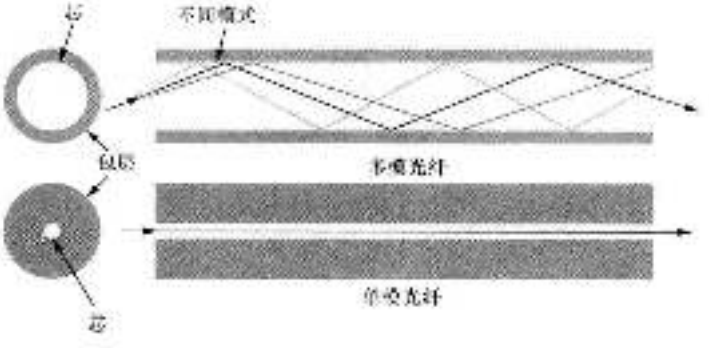


图 2-9 单模光纤和多模光纤

光纤与实导线相比具有很多优势，因为它们对外部电磁干扰具有极的衰减并且不容易受到干扰。它们比铜电缆更加难以窃听。因此，它们经常用在高速和远距离传输中。由于目前部署成本较高，它们大多用于骨干网而非用于私人。

2.2.2 无线介质

无线介质是允许电磁波传播的自由空间，而不需要使用任何物理电缆。电磁波在自由空间中是以广播或传输的，能够被波覆盖范围内的任何接收天线接收。

传播方法

有三种传播电磁波的方法，分别是：地面传播、天空传播和视距传播。地面传播由大气低层传输的低频率波或信号使用。地面传播的应用有无线电导航系统或无线电信标。更高频率的波会到达电离层再通过天空传播返射回地球。同屏（AM）无线电、调频（FM）无线电、蜂窝电话、无线局域网、甚高频（VHF）TV、超高频（UHF）电视和民用波段属于此种应用。在视距传播中，高频波从源直接传输到目的地。卫星通信是采视距传播方法的应用。视距传播的意思是发送者和接收者在一条直线上可以相互看到。但真正的长高频率的波才是如此。属于此类的许多信号除了直线传播外及射外，还以折射和分射传播。折射是当波以一定角度进入另一种介质时，在速度上发生变化，因此方向也会发生改变。衍射是指能够围绕障碍物弯曲并能通过小缺口传播出去的波。

传输波：无线电、微波、红外线

用于传输的电磁波分为三类：无线电、微波和红外线。无线电范围是大约3kHz～1GHz。频率范围包括VLF（甚低频，3～30kHz）、LF（低频，30～300kHz）、MF（中频，300kHz～3MHz）、HF（高频，3～30MHz）、VHF（甚高频，30～300MHz）、UHF（特高频，300MHz～3GHz）。无线电波通常使用全向天线，能以任意方向经过地面和天空接收、发送信号。使用全向天线的缺点是信号很容易受到附近其他使用相同频率的用户的干扰。优点在于信号可以由同一个无线发送，但可以被很多接收机接收。它适合于组播或广播。此外，经过天空传播的无线电波可以传输很远的距离。这就是选择无线电波进行广播的原因。其应用为调频广播和调幅广播、电视广播和寻呼。

微波的典型范围是1GHz～300GHz，涵盖部分UHF（300MHz～3GHz）、SHF（超高频，3～30GHz）、EHF（30～300GHz）。然而，大多数应用通常在1～40GHz范围内。例如，全球定位系统（GPS）在大约1.2～1.6GHz的范围内传输信号，IEEE802.11使用2.4GHz和5GHz，WiMAX工作在2～11GHz。如果发射和接收天线能够对准进行视距传播，那么更高频率的微波或使用定向天线来发送和接收信号。这种类型的定向天线是喇叭形，采用喇叭的弯曲形状可以平行地发出微波束。定向接收天线是抱物面蝶形天线，为了收集这些信号需要在一个共同点收集广泛的平行光束。收集的信号然后自通过导线传送到接收设备。

与无线电波相类似，微波传输需要使用信管部门分配的频谱中的可用频带。幸运的是，ISM（工业、科学和医疗）频段的使用是无须申请许可证的。使用ISM频段的一个常见例子是微波炉，工作在2.4GHz频带。无线电话、无线局域网，以及许多短距离无线设备也工作在ISM频段，但无须取得频段使用许可证。由于多种无线设备通常在同一时间共享ISM频段，所以在这些设备之间避免干扰是必要的。扩频，将传播信号功率扩展到一个更广泛的频谱上，是WLAN中用来避免干扰的技术之一。因为一个信号扩展到更宽的频谱上不再受窄带干扰的影响，从而接收器有更好的机会来准确地恢复传播信号。扩频将在3.5节中介绍。

红外线波的范围是300GHz～400THz，用于短距离传输。由于高频率的特性，红外线波不能穿透墙壁，因此它们可以在一个房间里使用，而不会受到其他房间中设备的干扰。有些设备，如无线键盘、鼠标、笔记本电脑和打印机，可以使用红外线波通过视距传播来传送数据。

移动性

无线通信与有线通信相比最明显的优势是移动性。与使用电缆传输不同，无线连接使用无线频谱。大多数无线系统使用微波频谱，尤其是从800MHz～2GHz，以便维持全方向和高位速率之间的平衡。更高的频谱可以提供更高的位速率，但那就会更加有方向性并失去可移动性。

2.3 信息编码和基带传输

在计算机网络和信息处理中，代码（code）是一种将信息从一种形式或表示转换成另一种的方案，编码（coding）是一种将信息源转换成符号的处理，而解码（decoding）则与编码处理。在2.1节中的传输流和接收流中，计算机网络中的信息源在传输或进一步调制之前分由信源编码、信道编码和线路编码处理的。信源编码和信道编码属于信息和编码理论领域，但为线路编码则属于数字基带调制领域。

信源编码用来压缩并减少信息的存储空间，从而提高信道的数据传输效率，尤其是在用于存储或传输图像、音频、视频和语音时更是如此。信源编码通常是在应用层上进行的。信道编码通常会在原始数据上增加额外的位，使数据对由信道引入的损害有更高的容错性。信源编码既可以在线路层以上以在物理层上实现。线路编码不仅将数字数据转换成数字信号，而且还处理基线漂移、同步丢失和直流分量等问题，参见2.1节中的讨论。本节介绍源编码和信道编码，并描述各种线路编码的方案。

2.3.1 信源编码和信道编码

信源编码

信源编码的目的是为了高效的信息源描述，以便可以减少所需要的存储或带宽资源。它已成为通信中的一个基本子系统，并且它使用来自数字信号处理（DSP）和集成电路（IC）的技术。有多种压缩算法和标准用于图像、音频、视频、语音等领域的信源编码。信源压缩的一些应用如下：

图像压缩：无压缩时，图像就太大以至于不能存储并在信道上传输。联合图像专家组（JPEG）和运动图像专家组（MPEG）是两种流行的图像压缩方案。

音频压缩：流行的音频压缩技术包括光盘（CD）、数字多功能光盘（DVD）、数字音频广播（DAB）和运动图像专家组音频层3（MP3）。

语音压缩：语音压缩通常应用于电话，尤其是蜂窝电话。例如，G.72x和G.711标准。

信道编码

信道编码被用来保护数字数据，在转发或检索数据时它能够通过可能会导致错误的、无噪声的传输介质或一种不完美的存储介质。在通信系统中的发射器，根据预定的算法调节会给消息添加冗余位。接收器可以检测并纠正由噪声、衰减或干扰所造成的错误。任何信道编码的性能都受限于香农（Shannon）信道编码定理，其中规定只要传输速率低于某一数量，又称为信道容量，那么在有噪声的信道上就可能无差错地传输数据。更正式地说，任何无限小 $\varepsilon > 0$ 和任何小于信道容量的数据传输速率，总会存在一种编码和解码方案，能够确保足够长代码的错误概率小于 ε。相反，香农信道编码定理也指出，以高于信道容量的速度传输将具有大于0的错误概率。

对于错误纠正系统，接收器通常采用两种方案以便纠正错误。一种是自动重发请求（ARQ），另一种是前向错误纠错（FEC）。与ARQ不同，FEC可以用来纠正错误，而不要求发送器重新发送原始数据。分支组（交错）是数字通信中用于对付突发错误的另一种方案，但它会增加延迟。它置换数据流中的编码位，使得只有在的连续编码字在传输过程中才会受到突发错误的影响。

错误纠正码可以分为分组码和卷积码。卷积码是对任意长度的信息逐位进行处理，而分组码是对信息中固定大小的分组逐块进行操作。分组码常见的例子包括海明码和Reed-Solomon码。Turbo码，是1993年开发的一种非常强大的纠错技术，是从预定义变换误差的卷积码中推导出来的。

海明码是1950年发现的并且沿用至今，例如用于存储设备中信错误校正等。Reed-Solomon码有着广泛的应用，例如CD、DVD、蓝光光盘、数字用户线路（DSL）、全球微波互联接入（WiMAX）、数字视频广播（DVB）、高级电视系统委员会（ATSC）、磁盘冗余阵列（RAID）系统，以及使用的Reed-Solomon码的应用。卷积码通常应用到数字无线电、移动通信和卫星通信等应用中。Turbo码可接近信道容量或香农极限。Turbo码广泛应用于3G移动通信标准、长期演进（LTE）项目，以及IEEE 802.16 WiMAX标准中。

2.3.2 线路编码

线路编码是一种将原始应用到二进制符号的处理过程，并产生一种脉冲编码调制（PCM）波

形。PCM 波形称为线路编码。脉冲调制采用一非规则的脉冲序列来表示对应的一系列采样信息的数据。有四种基本的脉冲调制形式：脉冲振幅调制（PAM）、脉码调制（PCM）、脉冲宽度调制（PWM）或脉宽调制（PDM），脉冲位置调制（PPM）。与 PAM、PWM 和 PPM 不同，PCM 使用可个不同振幅的序列来表示一个量化的采样或相应的信息，因此 PCM 就成为现代数字通信中最受关注的脉冲调制。这是因为从一个码状态序列检测并决定数据的值比准确地测量振幅，特别时间或脉冲分别在 PAM、PWM 和 PPM 接收机中的位置要简单得多。这里所描述的所有线路编码方案都属于 PCM。

自同步

在计算机网络中存储的数据是数字形式的位序列。这些序列需要转换为数字信号才能在物理信道上传输。正如章节 2.1 节中所述，线路编码将数字数据转换成可以通过基带信道传输的数字信号。如果通信在带通或宽带信道上进行的，就会使用不同的方案将数据转换成信号通带。图 2-10 说明了线路编码方案，数字数据是从发射器发送到接收器。

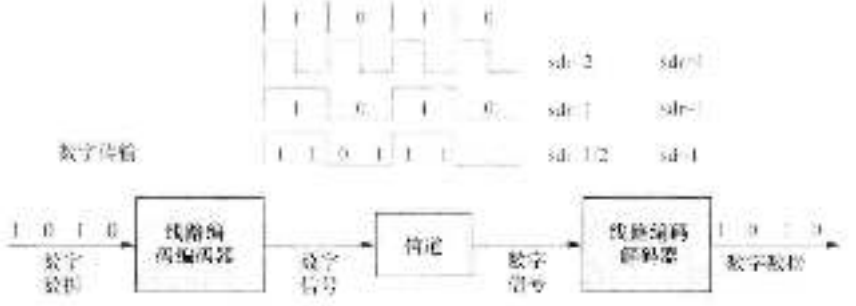


图 2-10 线路编码和信号数据率

在接收端，线路解码器的位间隔必须与对应发射机的线路编码器的位间隔完全匹配。任何位间隔的微小变化或偏移都可能导致信号的误解。为了使计算机能够将接收到的信号正确解码为与发射机发射的位序列相同，接收机时钟与发射机时钟保持同步非常重要。如果某线路编码方案在一个数字信号中嵌入了位间隔信息，接收到的信号就可以帮助接收机利用发射机的时钟来同步其时钟，并且其线路解码器可以从数字信号中检测到准确的数字数据。这是一种自同步技术。有些线路编码方案提供了自同步，而有些则不提供。

信号数据率

在图 2-10 中，信号数据率（SDR）（类似于术语信噪比）是信号元素的数量与数据元素的数量之比。数据速率是在一秒钟内发送的数据元素的数量，又称为位速率（单位为 bps），而信号速率则是在一秒钟内发送的信号元素的数量，又称为波特率、脉冲速率、调制速率。信号速率和数据速率之间的关系可以表示为 $S = c \times N \times \text{sdr}$，其中 S 是信号速率，c 为环境因子，N 为数据速率。分别在最坏的情况下、最好的情况下或平均情况下指定环境因子 c。平均情况下，c 的值为 1/2。信号速率越小，信道需要的带宽就越少。因此，可以从上述讨论得出，如果 sdr > 1，信号就可能包含自同步信息，所需要的信道带宽就会增加。

在 2.1 节中，我们提到非周期性的数字信号具有无限连续的频谱范围。然而，大多数的高频谱振幅很小可以忽略不计。因此，一个有效的有限带宽可以用于数字信号，而不是无限范围的带宽。带宽常常定义为以赫兹表示频率范围的传输信道。因此，我们假设在以赫兹（频率）表示的带宽与波特率（信号率）成正比，而以位每秒（bps）表示的带宽与位速率（数据速率）成正比。

线路编码方案

这里简要地描述在线路编码使用的术语。在二进制波形中，“1”称为“标记”或“HI”，而“0”称为“空”或“LO”。在单极性信号中，“1”代表一个有限的 V 伏电压，“0”表示零伏电压。在极性信号中，“1”表示具有有限的 V 伏电压，“0”表示具有 −V 伏电压。最后，在双极性信号中，“1”代表有限的 V 或 −V 伏电压，“0”代表零伏电压。线路编码方案可以分为几类，如表 2-2 所示。除了上述三类之外，还有多类和多维变转换（multitransition）类别。因为单极信号是直流不平衡的而且为了传输需要比极性信号更多的功率，目前，一般情况下已经不再使用。线路编码的波形如图 2-11 中所

示。每种编码方案所需要的带宽如图2-12所示。下面，我们使用图2-11和图2-12对这些方案进行了详细的描述。还给出了两种高级的编码方案，行程长度受限（RLL）和分组编码。

表2-2 线路编码分类

线路编码分类	线路编码	线路编码分类	线路编码
单极性	NRZ	多电平	2B1Q，8B6T
极性	NRZ，RZ，曼彻斯特，差分曼彻斯特	多维度编码	MLT3
双极性	AMI，伪3		

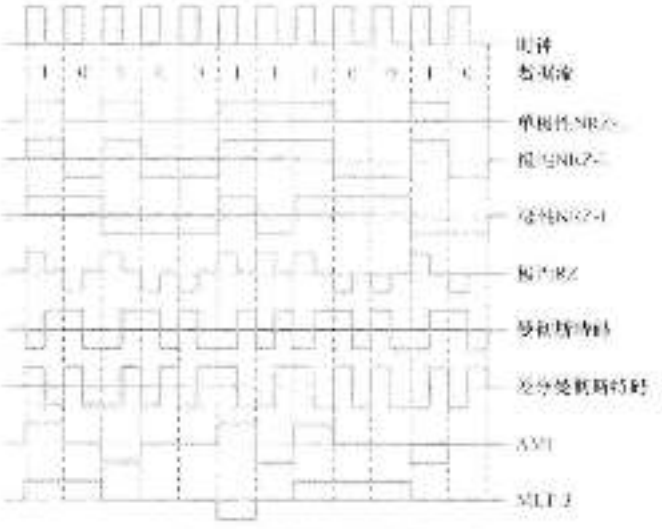


图2-11 线路编码方案的波形

没有自同步的单极性不归零（NRZ）

使用这种方案，位1定义为一个正的电压，位0定义为0电压。由于信号在位的中间不返回到零，所以该方案称为不归零。单极性NRZ所需要的功率是极性NRZ的两倍。

没有自同步的极性不归零（NRZ）

这种编码方案将正电平定义为1，将负电平定义为0。极性NRZ还有几个变种，包括极性不归零电平（极性NRZ-L）、极性不归零反相（极性NRZI）、极性不归零空（极性NRZS）。

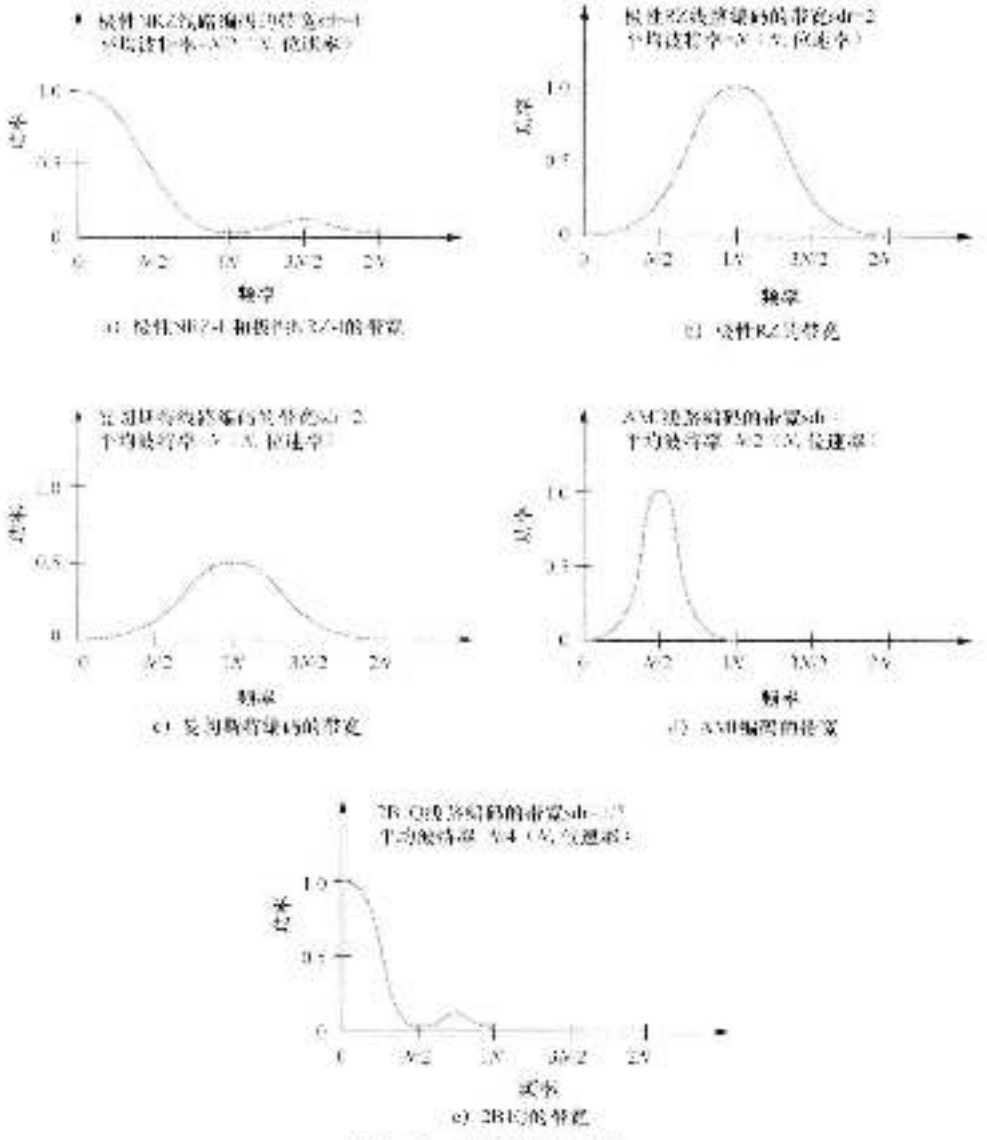
极性不归零电平（NRZ-L）：该方案将1定义成一个正电平，0定义成负电平。如果一长串位（无论是位1还是位0）没有发生变化，那么位间隔信息就可能会丢失。这种方案要求额外支持在发射器和接收器之间提供时钟同步。

极性不归零空（NRZ-S）："1"表示信号电平没有改变，"0"表示信号电平跳变。高级数据链路控制（HDLC）和通用串行总线（USB）使用此种方案，但会在一长串1中填充0。由于填充0可以激活跳变，所以就可以避免长的"没有变化"，从而实现同步。

极性不归零反转（NRZ-I）：与NRZ-S相反。这里的位"1"意味着一次跳变，位"0"是指没有跳变。假定一位，跳变发生在时钟的开始。类似地，一长串0而没有跳变就会破坏同步属性。在极性NRZ-I编码之前，可以将上节中讨论过的分组编码应用到该方案中，以减少失去同步的机会。也可将在后面进一步讨论的RLL与NRZ-I结合起来使用。

在极性NRZ-I中的基线漂移和同步问题比极性NRZ-S和极性NRZ-I中的严重两倍，因为在极性NRZ-I中不论是位"1"还是位"0"都可能会产生一长串没有变化的连续位，所以会造成偏离平均信号功率并失去同步，而在极性NRZ-S和极性NRZ-I中，只有一种类型的位，是"1"或"0"，将会产生一长串没有变化的序列。所有极性NRZ机制既没有自同步也没有自动静止状态，它们都是在信号电平零，因此就需要有一种额外的同步机制来防止位漂移。例如，磁盘驱动器使用极性NRZ-I的RLL编

码、USB 使用极性 NRZ-S 的传输光。极性 NRZ 方案很简单，便宜。1000BASE-X 以太网光纤使用极性 NRZ，就是因为它所对应的分组编码 8B/10B 能为以太网中的高速传输提供足够的同步。



a) 极性NRZ-I和极性NRZ-S的带宽　b) 极性RZ的带宽　c) 双相码的带宽　d) AMI编码的带宽　e) 2B1Q的带宽

图 2-12　线路编码的带宽

极性 NRZ 的 sdr 为 1，因此平均信号速率（波特率）$S = c \times N \times \text{sdr} = 1/2 \times N \times 1 = N/2$。如果带宽与波特率成正比，那么极性 NRZ 的带宽可表示为图 2-12a 所示。因为高功率密度大约是在频率 0。大多数能量分布在频率范围从 0～$N/2$，所以这意味着高效分量携带了很大的功率，功率不是均匀地分布在信号频率 $N/2$ 的两边。极性 NRZ 要比其他带有 DC 分量接近于零的方案消耗更多的功率。

带有自同步的极性归零（RZ）

二进制信号可以通过极性归零（极性 RZ）进行编码，如图 2-11 所示。代表每“1”或每“0”的脉冲总是在当前位的中间点返回到一个用零来标示的中立或静止状态。这种编码与极性 NRZ 相比的优点在于它时钟同步，但以使用双倍带宽为代价。极性 RZ 的带宽如图 2-12b 所示。这里，极性 RZ 编码的平均波特率为 N，这和位速率相同，并且 sdr 等于 2。功率密度均匀地分布在波特率 N 的两边，这里 DC 分量承载非常小的几乎接近于零的功率。然而，使用三电平也会增加编码和解码装置的复杂性。因此，曼彻斯特和差分曼彻斯特方案比极性 RZ 具有较好的性能，极性 RZ 目前已经不再使用了。

极性曼彻斯特编码和带有自同步的差分曼彻斯特编码

曼彻斯特编码利用由低到高跳变代表“1”，由高跳变到低表示“0”，每次跳变都发生在位“1”或“0”周期的中间。这个方案将极性RZ与极性NRZ-L结合起来。它通过在每个数据位引入一次信号跳变来保证自时钟。这样会再次加倍信号频率，因此曼彻斯特编码需要的带宽是NRZ需要带宽的两倍。因此，更高传输速率（如100Mbps）的以太网不采用曼彻斯特编码。然而，在较低速版本的802.3（以太网）和IEEE 802.4（令牌总线），如10BASE-T，采用曼彻斯特编码就是看中它的自同步优势。

差分曼彻斯特编码是曼彻斯特编码的一个变种，但性能要超过后者。在差分曼彻斯特编码中，“1”要求信号的前半部分与前一信号相同，“0”则要求与前面的相反，跳变总是发生在信号的中间。这种方案导致“1”需要一次跳变，“0”则需要两次跳变。这是极性RZ与极性NRZ-L的组合。因为检测信号的跳变比比较信号振幅是否超过固定阈值要可靠得多。差分曼彻斯特编码比曼彻斯特编码具有更好的错误控制性能。IEEE 802.5（令牌环LAN）采用差分曼彻斯特编码。

曼彻斯特和差分曼彻斯特两者都没有基线漂移与直流分量问题，但是与极性NRZ相比，它们必须加倍信号速率。它们的sdr（2）和平均信号速率N与极性RZ相同。带宽如图2-12c所示。

双极性交替标记反转（AMI）和无自同步的伪三元

在AMI编码中将一个“0”或“空”编码成零伏，“1”或“标记”编码成交替的正的或负的电压，如图2-11中所示。伪三元（伪三进制）是AMI的变种，其中“1”代表零伏，“0”被编码成一个正电压或负电压。通过交替同一位值的电压，直流分量得以平衡。如果数据在AMI中包含一长串的1或在伪三元中包含一长串的“0”，那么这种方案可能会失去同步。为了加以弥补，AMI编码器会在连续7个零后增加一个“1”作为第8位。通过这种位填充，与极性NRZ-S中使用的方法相类似，总的线路编码比源代码长平均不到1%。此编码被T载波用于远距离通信。这个方案的两个优点分别是零DC分量和能够更好地进行错误检测。它的带宽如图2-12d所示。sdr和信号速率与极性NRZ的相同。与极性NRZ不同的是，即使有一个长序列位“1”或位“0”，AMI也不存在直流分量问题，其功率密度集中在信号速率N/2，而不是零。

为了避免添加额外的位，AMI的一个变种称为改进AMI使用扰码，由T载波和E载波使用。它不在原始数据中增加位数。这里我们学习了两种扰码方案：双极8零替代（B8ZS）和高密度双极3零（HDB3）。B8ZS编码使用000VB0VB替换8个连续的0，其中V表示这是一个违反AMI编码规则的非零违反位，B是另外一个按照AMI编码规则的非零位。HDB3编码既可以使用000V也可以使用B00V取代4个连续的0，这要取决于最后更换后的非零位数。如果是奇数，它就使用000V；如果是偶数，就使用B00V。使用这一规则的目的就是在每次替代后保持偶数个非零位。

多电平编码：m进制，n电平（$mBnL$）

多电平编码方案的目的是通过在信号中使用多个电平代表数字数据来降低信号速率或者信道带宽。标记$mBnL$用来表示编码的方案。字母“B”意味着二进制数据；字母L为信号中电平的数量；字母m为二进制数据模式长度，字母n是信号模式的长度。如果$L=2$，就是用B（二进制），而不是L。同样，如果$L=3$，就是用T（三进制）；如果$L=4$，就使用Q（四进制）。因此，我们可能会看到一些如2B1Q、4B3T和8B6T等的多电平编码方案。

根据标记$mBnL$，我们有2^m种二进制数据模式，L^n种信号模式。如果$2^m=L^n$，则所有的信号模式用来表示数据模式。如果$2^m<L^n$，就会存在比数据模式更多的信号模式。这些额外的信号模式可以用于预防基线漂移并提供同步和错误检测。如果$2^m>L^n$，则信号模式不足以表示数据模式，从而无法完全编码全部的二进制数据。这里我们讨论三种典型的方案。

2-二进制，1-四进制（2B1Q）：2位数据被映射到一个有4个电平的信号元素上，如表2-3所示。因此sdr=1/2。平均波特率计算如下：$c\times N\times sdr=\frac{1}{2}\times N\times 1/2=N/4$，也就是说，等于位速率的1/4。2B1Q的带宽如图2-12e所示。与NRZ相比，2B1Q只需要NRZ所用带宽的一半。换句话说，在相同位速率下，2B1Q能够携带两倍NRZ的数据率。但是，使用2B1Q的设备比使用NRZ的更加复杂。因为2B1Q使用4个电平来表示4种数据模式。为了区分4个电平，设备需要使用更加复杂的电路。这种编码模式没有冗余的信号，因为$2^m=2^2=L^n=4^1$。综合业务数字网（ISDN）的物理层就使用这种编码方案。

表 2-3 2B1Q 编码映射表

DIBit（2 位）	00	01	10	11
上次的一个信号电平为正，下一个信号电平	+1	+3	−1	−3
上次的一个信号电平为负，下一个信号电平	−1	−3	+1	+3

4B3T 和 8B6T：线路编码 4B3T 用于 ISDN 的基本速率接口（BRI），这是用 3 个脉冲表示 4 位。8B6T 被 100BASE－4T 电缆使用。因为 8B 意味着数据模式，而 6T 意味着信号模式，许多冗余信号模式可以用于直流平衡、同步和错误检测。因为 sdr 为 6/8，平均波特率变成 3N/8，即 $c \times N \times sdr = \frac{1}{2} \times N \times 6/8 = 3N/8$。

无自同步的多电平传输三级电平（MLT-3）

修性 NRZ-I 和差分曼彻斯特编码可看作是二电平传输编码，根据连续位值的变化来编码二进制数据。MLT-3 使用三电平编码二进制数据。为了编码位“1”，它使用三个电平，+1、0、−1，四个状态，从电平 +1、0、−1、0 到 +1 依次形成一个循环。电平 +1 表示一个正的物理电平，电平 −1 表示负的电平。为了编码位“0”，电平会像前一位那样保持不变。因为 MLT-3 使用四个状态完成一个完整周期，或四个数据元素转换为一个信号元素（信号模式），sdr 近似于 1/4。根据 $S = c \times N \times sdr$，在最差的情况下，$c = 1$，波特率变成为 $S = c \times N \times sdr = 1 \times N \times 1/4 = N/4$，波特率仅为数据率的 1/4。这个特点使 MLT-3 适合在铜电缆上以更低的频率传输。100BASE-TX 采用 MLT-3，因为铜电缆仅能支持 31.25MHz 的波特率，但数据速率是 125Mbps。

行程长度受限编码

RLL 限制了重复位的长度以避免没有跳变的长的连续位流。行程是不变值的位数。如果将极性 NRZ-I 添加到 RLL 以便编码源数据，那么这里“1”代表了一个跳变和“0”代表没有跳变，行程就是 0 的个数。RLL 使用两个参数，其中 d 表示最小 0 位行程，k 表示最大 0 位的行程。因此，标记 RLL 就是（d，k）RLL。RLL 的最简单的形式是（0，1）RLL。用于某些硬盘的 RLL 行业标准为（2，7）RLL 和（1，7）RLL。它们的编码表如表 2-4 所示。表 2-4a 的（1，7）RLL 将两位数据映射成三位。除了四位（x，0，0，y）序列转化为（NOTx，xANDy，NOTy，0，0，0）外，一对位（x，y）要基于规则（NOTx，xANDy，NOTy）进行转换。

表 2-4 RLL 编码的例子

a）（0，1）RLL		b）（2，7）RLL		c）（1，7）RLL	
数据	（0，1）RLL	数据	（2，7）RLL	数据	（1，7）RLL
0	10	11	1000	00 00	101 000
1	11	10	0100	00 01	100 000
		000	000100	10 00	001 000
		010	100100	10 01	010 000
		011	001000	00	101
		0011	0001000	01	100
		0010	00100100	10	001
				11	010

分组编码

分组编码（块编码）是一种错误检测/校正技术，它将一种输入序列映射为另一种更长的序列，从而具有更好的误码性能。使用信道编码对误码性能的改善程度可用编码增益来评估，即描述未编码与已编码数据误码性能的 SNR 比值。由分组编码引入的冗余位可以用于同步和错误检测，因此可以简化后续的线路编码。通常分组编码是在线路编码之前进行的。分组编码用做错误检测编码时，可以在接收端检测传输错误并支持错误的确认。分组编码可以表示为 mB/nB，这里一个 m 位流编码成 n 位码字。

分组编码一般有三个步骤：分段、编码和连接合并。例如，一个位流分段为 m 位分段，这里编码成 n 位码字。最后这些 n 位码字被合并连接起来形成一个新的位流。

分组码通常是由硬决策算法进行解码，目前已经广泛地应用于许多通信系统中。这里要学习两种分组编码：4个二进制/5个二进制（4B/5B）和8个二进制/10个二进制（8B/10B）。

4B/5B 分组编码将每一个4位分组转换成5位。4B/5B 编码将一组4位映射成一组5位，如表2-5所示，这里5位码字最多只有一个引导零，最多两个后补0。如果任何5位码字与任何其他5位码字合并，产生的二进制元组将最多只有三个连续的0。经过4B/5B编码器后不会出现一长串的0。此外，来自有效数据字的5位字模式可以智能地选择以便平衡信号中的数字“1”和“0”，从而保证在线路编码中有足够数量的跳变。因为数据空间从16个4位字扩展到了32个5位码字，所以提供16个额外的码字用于其他用途，比如控制字，可以表示帧的开始和结束。有些保留字用于错误检测。因为没有有效的数据字可以转化为这些保留字，所以如果在接收方出现了一个保留字，那么就检测到传输错误。

表2-5 4B/5B 编码表

名称	4B	5B	描述	名称	4B	5B	描述
0	0000	11110	十六进制数据0	C	1100	11010	十六进制数据C
1	0001	01001	十六进制数据1	D	1101	11011	十六进制数据D
2	0010	10100	十六进制数据2	E	1110	11100	十六进制数据E
3	0011	10101	十六进制数据3	F	1111	11101	十六进制数据F
4	0100	01010	十六进制数据4	Q	n/a	00000	静止（信号丢失）
5	0101	01011	十六进制数据5	I	n/a	11111	空闲
6	0110	01110	十六进制数据6	J	n/a	11000	起始#1
7	0111	01111	十六进制数据7	K	n/a	10001	起始#2
8	1000	10010	十六进制数据8	T	n/a	01101	结束
9	1001	10011	十六进制数据9	R	n/a	00111	重置
A	1010	10110	十六进制数据A	S	n/a	11001	复置
B	1011	10111	十六进制数据B	H	n/a	00100	停止

4B/5B 编码通常与极性 NRZ-1 编码一起使用，如图2-13中所示体系结构。额外的位“1”为了同步产生一次额外的跳变。预定义的编码表将4位转换成5位，每位至少跳变两次。在应用了4B/5B分组编码后，输出的位速率增加了25%。4位码字以100Mbps速率的发送，现在需要125Mbps来发送新的5位码字。4B/5B的方法避免了NRZ-1中的同步问题，但它仍然有直流分量问题没有解决。基本频率只有数字数据率的1/4，从而至少需要四位产生一个完整的周期。输出信号可以很容易地通过CAT-5电缆传输。

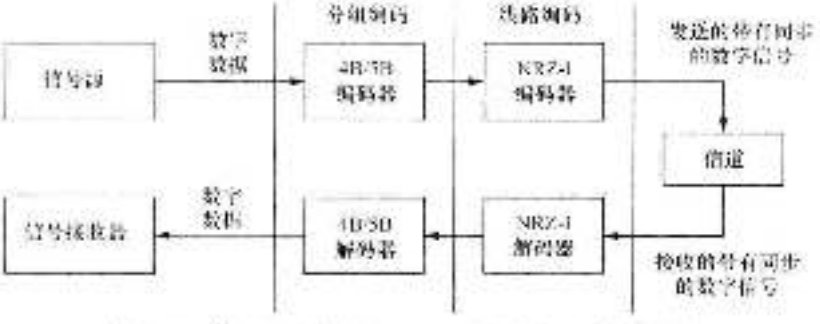


图2-13 将4B/5B编码与NRZ-1编码结合后的体系结构

更复杂的分组编码方法，如8B/10B和64B/66B一般应用于高速传输。这些复杂的编码技术可以平衡线路上传输的“0”和“1”的数量，计算哪里有较多的“1”或“0”并在运行中根据哪种位传输得更加频繁来选择适当的编码。既然10位码字具有最多一个额外的1或0的不平衡，计数仅包含1

代表当运行差异（RD）。每个传输的码字也有 RD，或为 RD+表示 1 比 0 多的情况，RD-表示相反。此外，更高时代的空间也允许在物理层上有较高程度的错误检测能力。

在结束本节之前，我们应当简要介绍过的 8B/10B 和 64B/66B 码是最简单的检测/纠错编码，主要用于通信信道更加可靠、噪声较少的有线物理通信中。具有很高噪声信道，如无线通信中常常就需要更强大、更长长度的编码。在这种应用中，编码长度可能高达数千或数万。此外，与用于解码 8B/10B 的硬决策解码相比，在概率域中经常使用更加复杂的软判定算法对很长的编码进行解码。

开源实现 2.1：8B/10B 编码器

综述

8B/10B 已经广泛地被多种高速数据通信标准（包括 PCI Express、IEEE 1394b、串行 ATA、DVI/HDMI 和千兆以太网等）所采纳，用于线路编码。它将 8 比特字映射为具有受限不一致的 10 比特字，这样就限制了两种重要的特性。一种是 DC 平衡方法，即允许特定的数据流提供几乎相等的 0 和 1，从而使电荷在线外介质上积聚起来。另一种方法是最大行程，即最大数量的连续 0 或 1，这就说明要求有充足的用于定时同步。一个开源的实现可以从 OpenCores 网站 http://opencores.org 上得到。它给出了 8B/10B 编码器和解码器在 VHDL 编码中的实现。这里 8B/10B 编码器是由 5B/6B 编码器和一个 3B/4B 编码器实现的。

框图

图 2-14 中描述了 OPENCORE 8B/10B 编码器的体系结构。它接收由 H、G、F、E、D、C、B、A 组成的 8 位并行原始（未编码的）数据字节。A 是最不重要的位。还有一个输入位，K，用来指示应该将字节输入编码成 12 个无效控制字符之一。

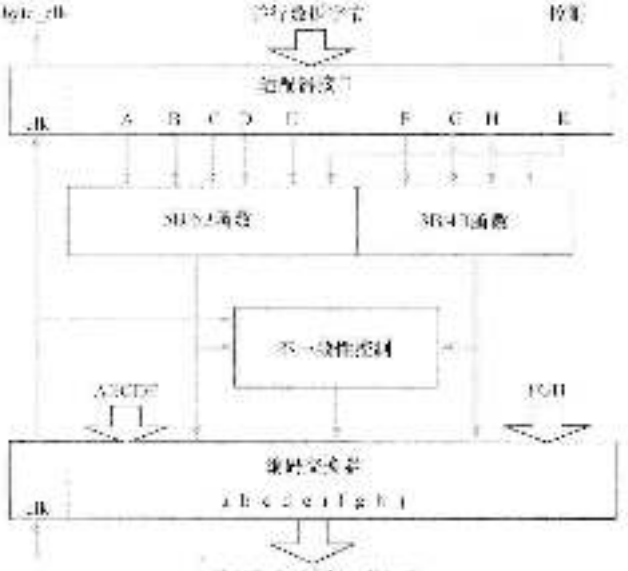


图 2-14 8B/10B 编码器的框图

实现将一个 8 位并行数据的输入块转为带有两个编码器的 10 位字符。一个是 5B/6B 编码器，它将 5 个输入位（A、B、C、D、E）映射为一个 6 位组（a、b、c、d、e 和 i）。另一个是 3B/4B 编码器，它将剩下的 3 位（F、G 和 H）映射为一个 4 位组（f、g、h 和 j）。

为了减少输入和点数，两个模块都多个输入位（元）组成类。例如，我们有 4 位（A、B、C、D），和 5 位码字可以划分为 4 类（D4+、D4-、D22、D40）。不一致性（差距）控制产生控制信号给编码类选择器，指示该块当前的正负的不一致性（差距）编码。该编码类被用于分类结果产生有个时钟的业编码块的位。

数据结构

8B/10B 编码器的数据结构主要是 8 输入位和 10 输出位，同所有的输入和输出同步到 clk；1）K。

H、G和F是在clk的下降沿内部锁定的；2）j、h、g和f是在clk的下降沿更新的；3）E、D、C、B和A是在clk的上升沿内部锁定的；4）i、e、d、c、b是在clk的上升沿更新的。

算法实现

在OPENCORE 8B/10B项目中，8B/10B编码器的VHDL（硬件描述语言）实现是在8b10_enc.vhd，而enc_8b10b_TB.vhd是编码器的测试平台（testbench）文件。图2-15中显示了5B/6B函数模块的代码分段，这是一个由多个NOT、AND和OR门组合而成的组合逻辑。其他模块也是从这些简单的逻辑门构造而来的，整个8B/10B编码器的实现不需要任何复杂的算术运算，例如，加法和乘法运算。因为整个8B/10B编码器代码太繁琐了，建议读者参见8b10_enc.vhd学习有关计算每一位的细节。

```
   L40 <= A1 and B1 and C1 and D1 ;                    -- 1,1,1,1
   -- Four 0's
   L04 <= not A1 and not B1 and not C1 and not D1 ;
   -- 0,0,0,0
   -- One 1 and three 0's

L13 <= (not A1 and not B1 and not C1 and D1)    -- 0,0,0,1
    or (not A1 and not B1 and C1 and not D1)       0,0,1,0
    or (not A1 and B1 and not C1 and not D1)    -- 0,1,0,0
    or (A1 and not B1 and not C1 and not D1)    -- 1,0,0,0
-- Three 1's and one 0

L31 <= (A1 and B1 and C1 and not D1)            -- 1,1,1,0
    or (A1 and B1 and not C1 and D1)            -- 1,1,0,1
    or (A1 and not B1 and C1 and D1)            -- 1,0,1,1
    or (not A1 and B1 and C1 and D1) ;          -- 0,1,1,1
-- Two 1's and two 0's

L22 <= (not A1 and not B1 and C1 and D1)        -- 0,0,1,1
    or (not A1 and B1 and C1 and not D1)        -- 0,1,1,0
    or (A1 and B1 and not C1 and not D1)        -- 1,1,0,0
    or (A1 and not B1 and not C1 and D1)        -- 1,0,0,1
    or (not A1 and B1 and not C1 and D1)        -- 0,1,0,1
    or (A1 and not B1 and C1 and not D1) ;      -- 1,0,1,0
```

图2-15 5B/6B函数的代码段

练习

如图2-14所示，在8b10_enc.vhd中找到与3B/4B编码交换相关的代码段，并指出控制输出定时的这行代码，即clk信号的下降沿或上升沿。

2.4 数字调制和多路复用

在电信网络和计算机网络中，需要利用数字调制将数字位流转换成带通波形以便通过模拟带通信道传输。带通波形，即一种通带信号，是一种从数字位流的振幅、相位、频率调制过的正弦模拟载波。这个处理过程称为数字通带调制，或简称为数字调制，与数字基带调制或线路编码相对。调制信号或原始数字信号两者都可能会进一步多路复用到一个物理信道上以便更好地利用信道。我们首先介绍基本的数字调制方案，包括调移键控（ASK）、相移键控（PSK）、频移键控（FSK）、混合正交振幅调制（QAM）。在此基础上，我们提出两种基本的多路复用方案：时分多路复用（TDM）和频分多路复用（FDM）。我们将在2.5节中讨论码分多路复用技术（CDM）和其他几种高级技术。

2.4.1 通带调制

通带调制是一个分为四步的处理过程。它根据所使用的调制方案，例如ASK、PSK、FSK或QAM等，首先将数字信号转换成基带复合值信号。然后这些基带波形乘以一个具有更高载波频率的正弦载波信号。删除虚拟成分，结果实值通带信号就准备好传输了。前者通常称为数字调制，而后者是由混

特定应的。这里重点强调了数字调制，如图 2-16 所示。与在 2.3 节中为基带传输介绍的线路编码不同，信号速率 $S = N \times 1/r$。这里不考虑情况因子，r 值是模拟信号可以携带的数据元素数，N 为数据速率，S 是调制之前的数字信号速率。

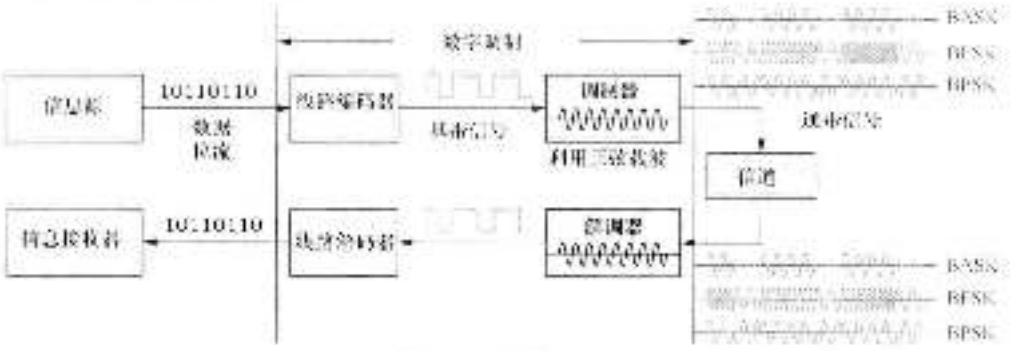


图 2-16 数字调制

在数字通信中，承带数字信号通常由更高频率的正弦载波承载以便能在更高频信道上传输。何谓正弦载波，以及载波是如何承载消息的？在无线通信中，发送端必须产生一个高频信号，称为载波，用来承载数据信号。接收端调谐到载波频率以便接收来自发送端的“载波承载的”数据信号。载波的各个方面或在振幅、频率和相位方面的更改都可以用来表示数字数据。使用数字数据技术修改载波的一个或多个方面称为调制或移键。它们分为调移键空（ASK）、频移键控（FSK）、相移键控（PSK）。存在一种混合振幅又包括相位的混合调制技术，称为正交振幅调制（QAM）。QAM 比 ASK、FSK 和 PSK 更有效率，因为它利用更多的方面。此外，载波各个方面的变更（如相变），也用于差分 PSK（DPSK）中。

星座图

星座图是用于定义从数字数据模式到信号星座点映射的一种工具。图中的星座点用于定义信号的振幅与信号元素的相位。这个图应用于所有数字调制中。图 2-17 是一种使用了两个载波的 4-PSK 的星座图，其中一个载波沿着实轴，另一个同相轴，另一个沿着虚轴是正交轴。在图 2-17 中，可以使用 4 个星座点定义 4 个不同的元素以便映射到 2 位的 4 种数据模式。图 2-18 说明了数字调制中的四种基本调制——ASK、FSK、PSK、DPSK。接下来，我们将逐一介绍其中每一种以及 QAM。

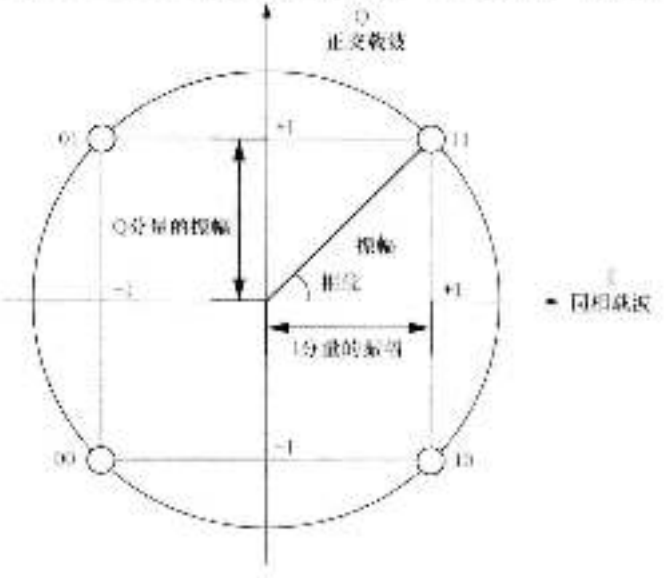


图 2-17 星座图：2 位的星座点：4 点

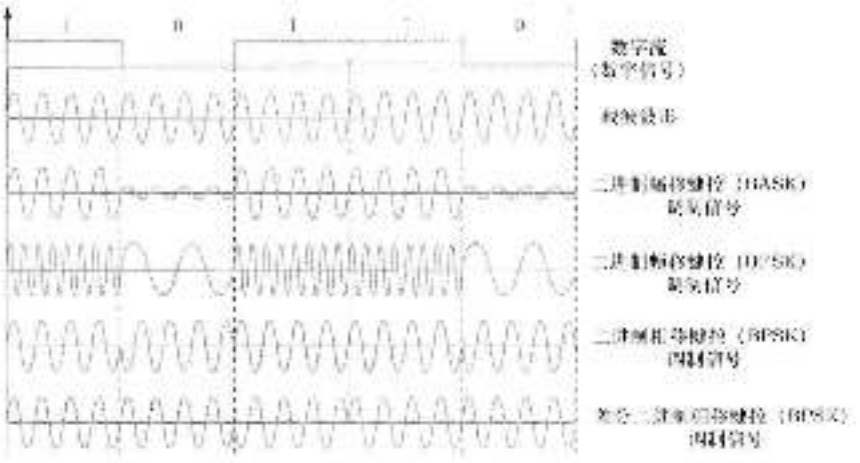
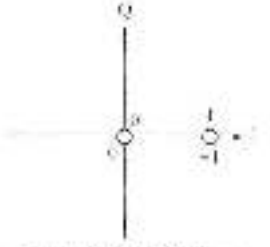


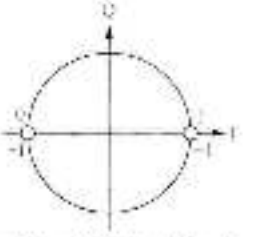
图 2-18 四种基本调制的波形

幅移键控

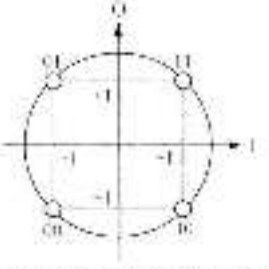
幅移键控技术使用不同振幅的载波来表示数字数据。通常在 ASK 中使用两个振幅：一个用于表示“1”，而另一个用于表示“0”，在调制期间载波的频率和相位都不改变。具有两个振幅 ASK 称为二进制 ASK（BASK），或开关键控（OOK）。它的星座图如图 2-19a 所示。只使用了一种载波即同相载波。



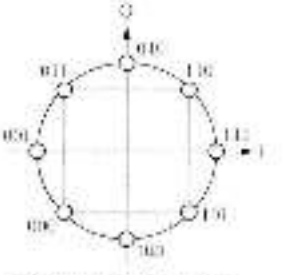
a) ASK（OOK）的星座：A_1



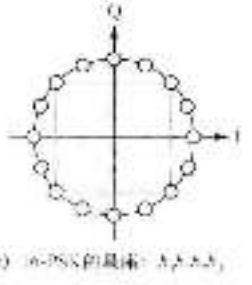
b) 4-PSK（QPSK）的星座：A_2



c) 4-PSK（QPSK）的星座：A_3A_4



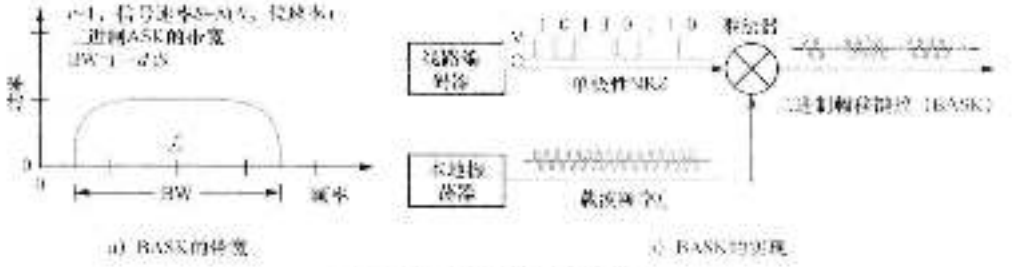
d) 8-PSK的星座：$A_5A_6A_7$



e) 16-PSK的星座：A_8~A_{23}

图 2-19 ASK 和 PSK 的星座图

零伏电压指示位“0”，而正电压表示位“1”。它的调制波形如图2-18所示。这里单极性NRZ线路编码器用来编码数字数据并且产生数字信号调制载波。根据BASK，$r=1$ 和 $S=N\times 1/r=N$。信号速率 S 等于数据速率 N。如果信号带宽与信号速率成正比，就可以获得带宽 $BW=(1+d)S$，其中 d 是0～1的因子，具体取决于调制和过滤处理过程。虽然载波是正弦信号，但ASK调制信号是一种非周期模拟信号。根据图2-1b，带宽是一种围绕载波频率的有限范围的频率，如图2-20a所示。实现ASK的机制如图2-20b所示。为了简化实现，利用乘法器将基带波形（一种单极性NRZ的输出）乘以本地振荡器的载波以获得一种调制信号。这种乘积称为混频，这是调带调制的第一步。



a) BASK的带宽　　b) BASK的实现

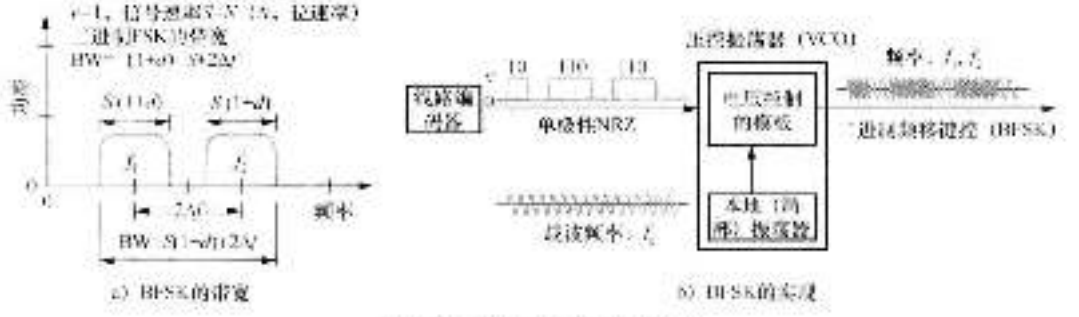
图2-20　BASK的带宽和实现

频移键控

频移键控技术使用载波频率来表示数字数据。换句话说，将载波频率更改为表示数字信号的值。最简化的频移键控方案采用“1”作为标记频率，“0”作为空频率。图2-18显示了二进制频移键控（BFSK）的波形，以及与其他移键控技术的对比。图2-21a显示了BFSK的频谱。在两种显著不同的频率 f_1 和 f_2 分别用来表示“0”和“1”。

在BFSK中，信元率的数量与信号元素的数量之比等于1，即 $r=1$，信号速率 $S=N\times 1/r=N$。如果认为BFSK技术是具有不同频率的两种BASK方案的结合，那么每个频率的带宽就是 $S(1+d)$。两个中心频率之差为 $2\Delta f$。频率之差必须大于以频率 f_1 为中心一半的带宽与频率 f_2 为中心一半的带宽之和，即 $S(1+d)$。因为 d 是0～1的一个系数，在最坏的情况下 $d=1$，那么就有 $2\Delta f\geq 2S$，即 $\Delta f\geq S$。这样就能保证两个信号的频谱不重叠，因此信号在频域内互不干扰。BFSK调制信号的总带宽为 $BW=S(1-d)+2\Delta f$，如图2-21a所示。

BFSK的一个简化实现方案如图2-21b所示，其中电压控制振荡器（VCO）用来改变载波的频率。FSK机制的输入是一种极性为正的NRZ信号。频移键控及其变种，最小频移键控（MSK）和高斯频移键控（AFSK），应用于GSM蜂窝电话标准和呼叫ID以便传送信息。



a) BFSK的带宽　　b) BFSK的实现

图2-21　BFSK的带宽和实现

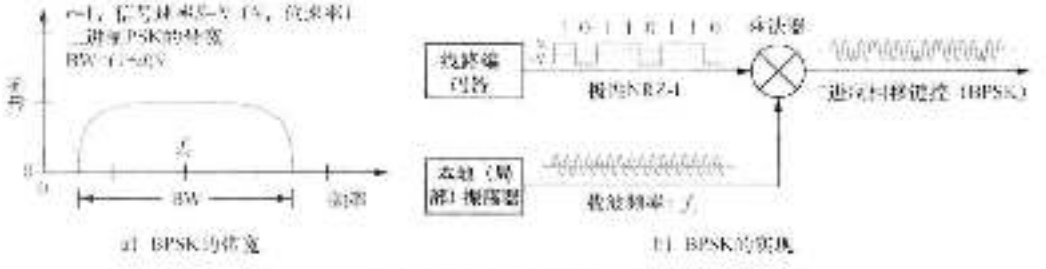
相移键控

相移键控技术通过调制载波的相位移组同数量的位编码或一个符号。换句话说，载波的相位用来表示数字数据。在键控中，载波的振幅和频率保持不变。接收器通过将有限数目的相位映射成有限数目的位模式，可以从接收到的信号取出数字信号。

用于m-PSK，如2-PSK、4-PSK、8-PSK、16-PSK的星座图，如图2-19所示，图中星座点均匀地围

绕着圆周放置。在这些 PSK 星座图中，只有相位的不同。根据图 2-19 所示，当其余的 m-PSK 使用两个载波，既使用同相载波也使用正交载波，我们会发现 BPSK 只使用一个载波，即同相（in-phase）载波。这样的布置有助于 PSK 取得最大的相位隔离并避免干扰。星座点的数目是 2 的幂，因为数字数据通常按二进制位发送。

二进制相移键控（BPSK）：BPSK 是仅使用一个载波（同相（in-phase）载波）的最简单的 PSK。如图 2-19b 所示的星座，两个不同的相位表示二进制数据；相位 0°为位“1”而相位 180°为位“0”。应用极性 NRZ 线路编码器以便于 BPSK 的实现，如图 2-22b 所示。极性 NRZ 信号的正电压并不改变载波的相位，而极性 NRZ 数字信号的负电压会将载波的相位转换 180°，即反相。BPSK 技术对噪声的免疫力高于 BASK，因为信号的振幅在遇到噪声时比信号的相位更容易降级。此外，BPSK 仅使用一个频率，而 BFSK 使用两个频率。因此 BPSK 性能上超越 BFSK。BPSK 的带宽与 BASK 相同，但小于 BFSK，如图 2-22a 所示。



a) BPSK的带宽　　b) BPSK的实现

图 2-22　BPSK 的带宽和实现

正交相移键控（QPSK）：QPSK 是使用两个载波的调制，一个同相载波和一个正交调制载波，以便携带两个序列数字数据。图 2-23 阐明了一个 QPSK 的简化实现。它可以比喻成将两种截然不同的，相位差为 90°的 BPSK 调制。在图 2-23 中，一个 11000110 位流首先分成两个均匀的子流。其中每一个都被一个极性 NRZ-L 线路编码器处理以便生成一个调制信号。其中一个将同相载波调制成一个 I 载波（同相信号）；另一个将正交载波调制成一个 Q 信号（正交信号）。将 I 信号和 Q 信号合并产生一个 QPSK 信号。每个信号元素可能有四个相位，45°、135°、−45°、−135°之一。因此一个二进制位流 11000110 转化成一个信号。I 信号、Q 信号和 QPSK 的波形如图 2-24 所示。实轴上的振幅将一个余弦波载波调制为 I 信号，而将虚轴上的振幅将一个正弦载波调制为 Q 信号。接收器收到的信号通过匹配滤波器、取样器、决策装置（设备）、多路复用器的处理深度为原始数据。QPSK 将两个数据元素（两位）编码成一个信号元素。这使得该项技术的数据处理速率为 BPSK 的两倍。在接收到的 QPSK 信号上会出现相位延迟，因此接收端时钟必须与发射器的时钟同步。此外，这种所谓的多普勒偏移（Doppler shift）会导致相对频率的偏置。由于信道导致的相位延迟，频率偏移必须在接收器由精确调整的正弦函数所补偿。电缆系统标准，数据有线电视服务接口规范（DOCSIS），指定 QPSK 或 16-QAM 用于上行调制。

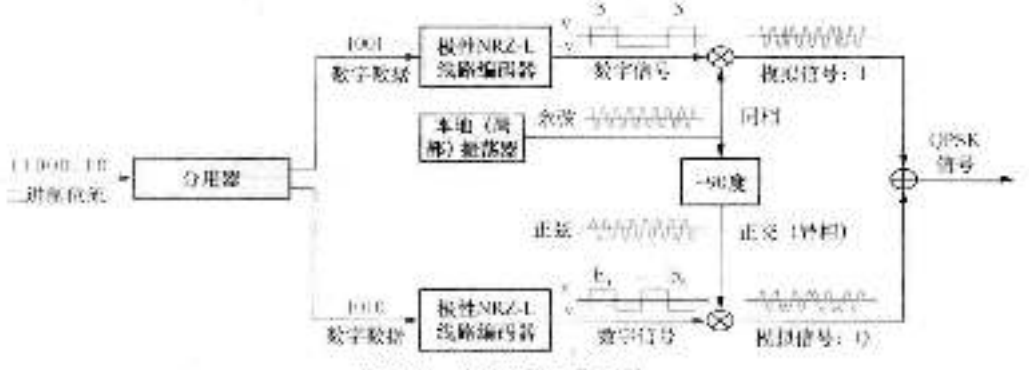


图 2-23　QPSK 的简化实现

差分相移键控（DPSK）：DPSK 是 PSK 的一个变种。这里，将位模式映射为信号相位的变化。这种方案极大地简化了调制、解调设备的复杂性。差分二进制相移键控（DBPSK）和 DQPSK 的波形如图 2-25所示。

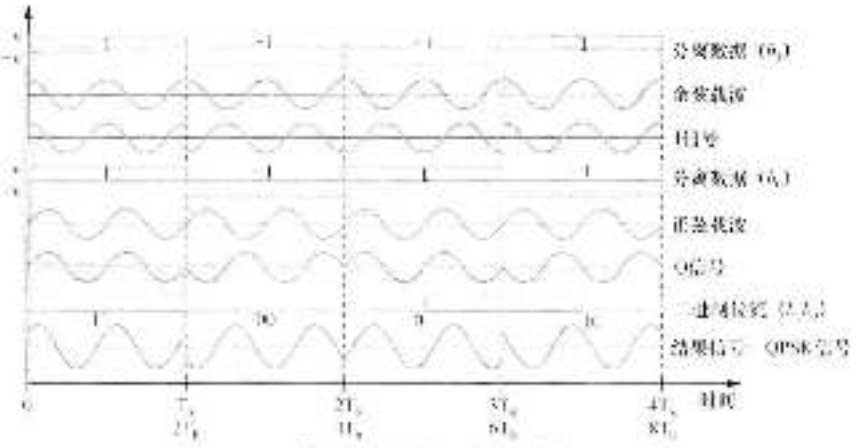


图 2-24 I、Q、QPSK 波形

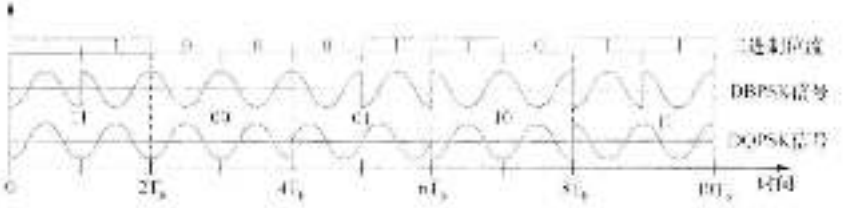


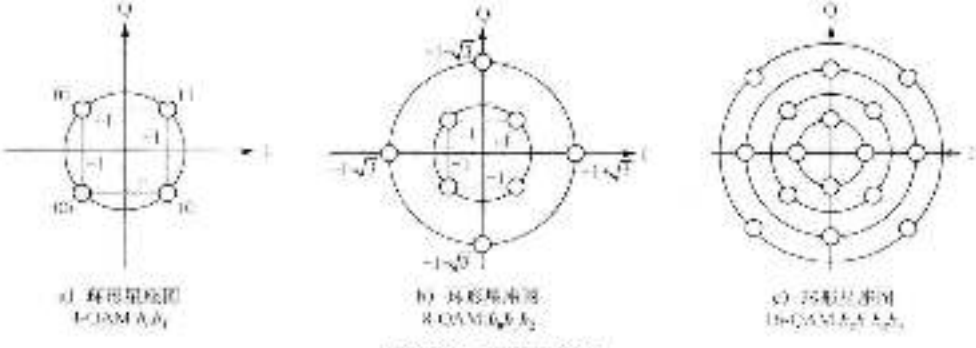
图 2-25 DBPSK 和 DQPSK 信号

在 DBPSK 调制中，如果信号的相位发生了变化，那么表示下来的信号就代表位 1；否则为 0。在 DQPSK 调制中，接下来的 2 位是根据信号相位的变化来决定的。如果没有相位变化，那么后两位就是 00。如果信号相位变化 $\pi/4$，后面两位就是 01。如果信号相位的变化为 $-\pi/4$，那么后面两位就是 10。如果信号的相位变化为 π，则后面两位就是 11。由于 DPSK 信号解调器不需要参考信号，所以调制解调器的简化设计就必然会导致更高的错误概率。然而，可以通过稍微提高信噪比来克服这一缺点。因此，DPSK 广泛地应用于 Wi-Fi 无线通信标准中。

正交振幅调制

正交振幅调制（QAM）通过改变载波的振幅和相位来形成不同信号元素的波形。QAM 使用不同幅度等级、同相载波、正交载波。因此它是 ASK 和 PSK 的一种组合。由于使用多种上述参数来表示信号中的多个位，所以使用 QAM 比使用 ASK 和 PSK 更容易实现更高的传输速率。举例来说，两个不同的振幅和两个不同相位，可以用来代表四种组合的 2 位模式。一个组合可以代表一个符号。因此，一种 2^N 个组合的符号可以一次携带 N 位数据。QAM 需要至少两个振幅和两个相位。

像 QPSK 一样，QAM 使用相位差为 90° 的两个正弦载波表示。QAM 采用两种类型的星座图：圆形（环行的）和矩形。图 2-26 显示了多个图形的星座图。4-QAM 的星座图与 QPSK 的相同。图 2-27 显示



a) 环形星座图
4-QAM A_1A_2

b) 环形星座图
8-QAM A_1A_2

c) 环形星座图
16-QAM $A_1A_2A_3$

图 2-26 环形星座图

向是矩形星座图，如4-QAM、8-QAM、16-QAM。在图2-28中，一个64-QAM矩形星座图代表64种不同的振幅和相位的组合。这种调制每个符号可以传输6位。然而，增加组合数量会使编码和解码处理更加复杂，当将这么多组合打包到一个符号中时就更难以区分两种组合了。由于调制信号容易出错，所以使用这种调制的传输就需要额外的错误检测技术。

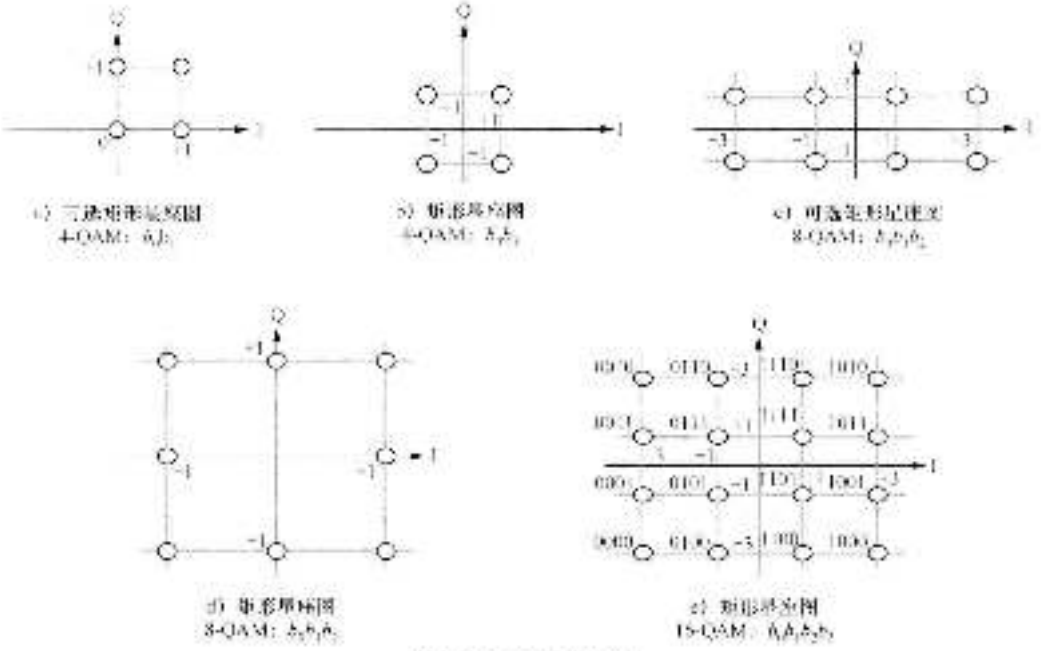


图2-27 矩形星座图

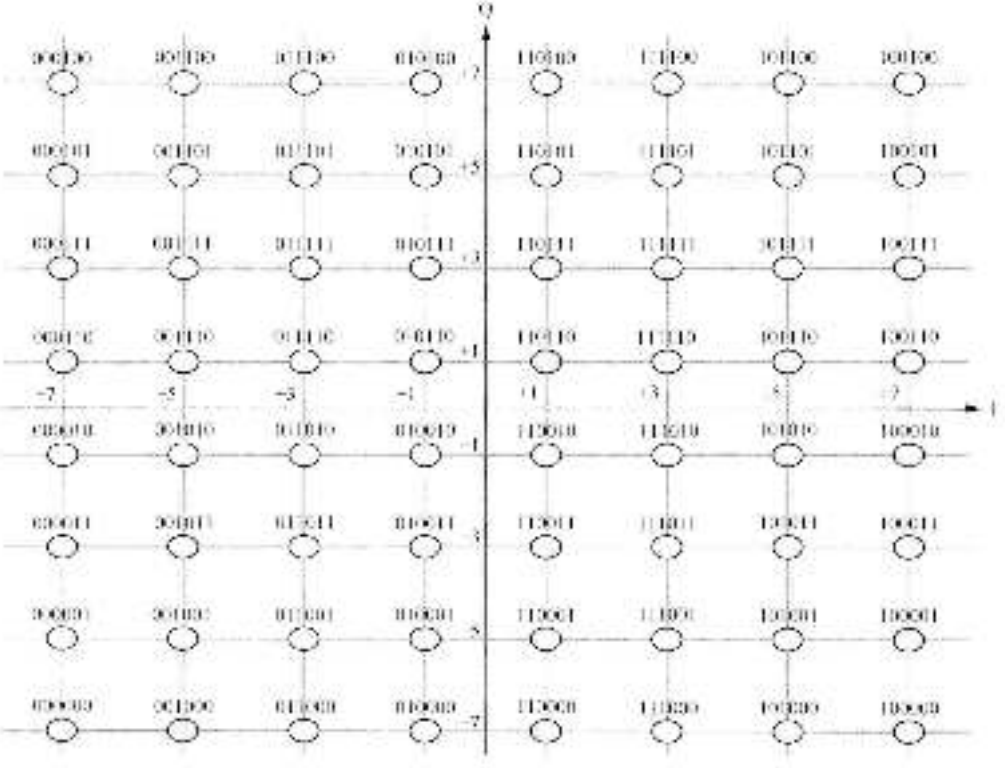


图2-28 矩形星座图64-QAM：$b_0b_1b_2b_3b_4b_5$

在 QAM 发射机上，每一个数据流分成两个子流。每个子流分由 ASK 调制器处理的。I 信道的输出乘以余弦函数，而 Q 信道的输出乘以正弦函数。最终的 QAM 信号是通过 I 信号和 Q 信号相加产生的。QAM 接收机应向处理以便获得原来的数据。DOCSIS 下行调制类用 64-QAM 或 256-QAM。上行调制可采用 QPSK 或 16-QAM。此外，新的 DOCSIS 2.0 和 3.0 的上行调制也使用 32-QAM、64-QAM 和 128-QAM。

2.4.2 多路复用

传输介质中的物理信道比每个数据流所需要的带宽。为了有效地利用信道容量，就要应用多种信道接入方案。利用信道接入方式，多个发送器就可以共享同一传输介质。信道接入方式主要有三种类型：电路模式、分组模式和全双工模式。多路复用是物理层使用的电路模式方法之一。链路层上的信道接入方式是指在媒体访问控制（MAC）子层中的多路访问协议的分组模式方法。全双工方法用来将上行和下行信道分离开。这里不介绍分组模式和全双工的方法。

带有多路复用器（MUX）和解复用器（DEMUX）的多路复用系统，如图 2-29 所示。图 2-29 显示了来自多个数据源的数据流复用以及通过共享物理信道进行传输。多路复用技术包括 TDM、FDM、波分复用（WDM）技术，码分多址（CDMA）和空间多路复用（SM）。表 2-6 显示信道接入及其相应的多路复用方法。这里我们仅介绍基本方法，而将 CDMA 留到 2.5.1 节中介绍。

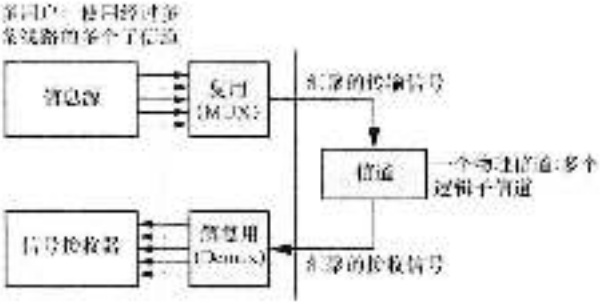


图 2-29 多个用户经过多个子信道所使用的物理信道

表 2-6 信道接入方案和多路复用的块层

多路复用	信道接入方案	应 用
FDM（频分多路复用）	FDMA（频分多路接入）	1G 蜂窝（移动）电话
WDM（波分多路复用）	WDMA（波分多路接入）	光纤
TDM（时分多路复用）	TDMA（时分多路接入）	GSM 电话
SS（扩频）	CDMA（码分多路接入）	3G 移动电话
DSSS（直序扩频）	DS-CDMA（直序 CDMA）	802.11b/g/n
FHSS（跳频扩频）	FH-CDMA（跳频 CDMA）	蓝牙
SM（空间复用）	SDMA（空分多路接入）	802.11n、LTE、WiMAX
STC（时空编码）	STMA（时空多路接入）	802.11n、LTE、WiMAX

时分多路复用

时分多路复用（TDM）是一种将来自低速率信道的多个数字信号结合成一条接入陈交者共享的高速率信道的技术。TDM 的简化方案如图 2-30 所示。来自不同源的数据流以时隙的方式交织在一起。

TDM 将一个时间分为多个固定长度的周期性时段。每个时段是某个子信道或整个信道的一部分。每个子信道用来传输一个数据流。交织时隙需要在多路复用器上同步。它可以通过在每个传输帧的开头加入一个或多个同步位来实现。与可以动态地为子信道分配时隙而不为空的输入线路分配时隙的统计 TDM 相对比，这就是所谓的同步时分复用（TDM）。如果输入的数据传输速率不同，就可以使用多

种技术，如多层次多路复用、多时隙分配、脉冲填充（或位插入、位填充）。电话行业中使用T线路以实现数字信号服务。T线路能分成从T1～T4的不同数据传输速率的服务。

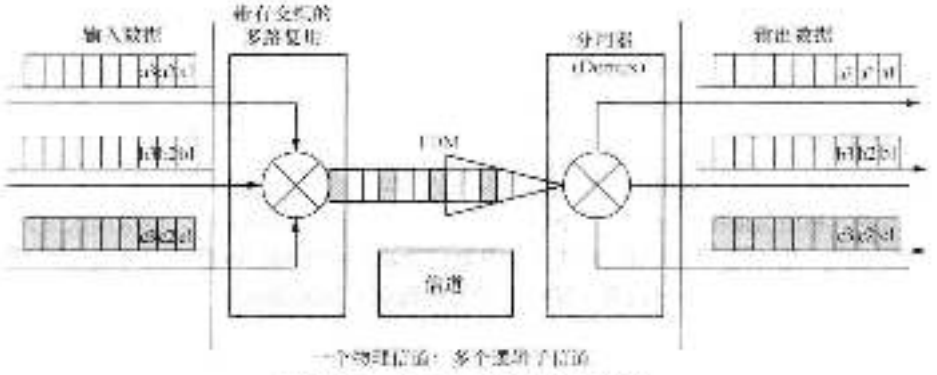


图2-30 时分多路复用的处理过程

TDM可扩展到支分多址访问（TDMA）方案中。通过在物理层实际上做工作的TDM，TDMA策略在链路层上得以实现。GSM电话系统就是其应用之一。

频分多路复用

FDM将频域分为几个非重叠的频率范围，每个是一个子载波使用的子信道。图2-31显示了FDM的处理过程。在发射器上，复用处理过程将来自数据流的硬有波形结合起来，其中一个子信道使用一个子载波，最终导致在一个物理信道上传输的复合信号。在接收端，使用多个带通滤波器，从收到的复合信号的子信道上提取消息。FDM只适用于模拟信号。模拟信号通过调制可以转换成数字信号，然后就可以应用FDM技术了。AM和FM信号的无线电广播是使用FDM技术的典型应用。例如，将从530～1700kHz的带宽分配给AM无线电收音机。这是一个物理信道介质的带宽，由多个无线电台共享。

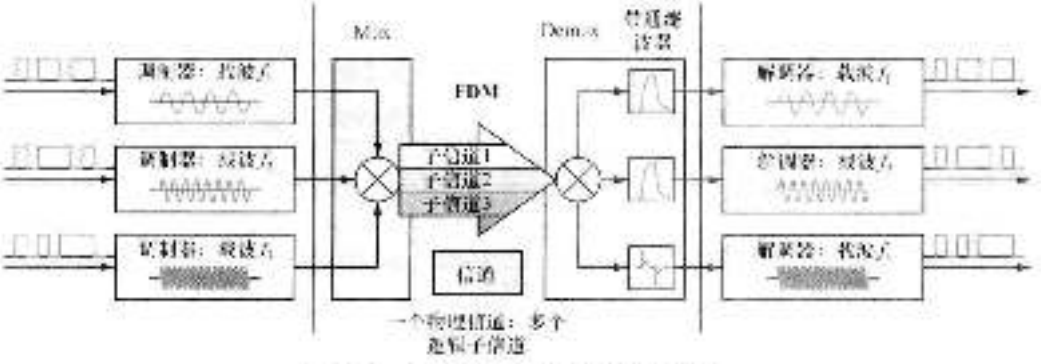


图2-31 频分复用（FDM）的处理过程

频分多路接入（FDMA）是由FDM扩展而来的访问方法。正交频分多路接入（OFDMA）是一种基于正交频分复用（OFDM）的FDMA变种。单载波FDMA（SC-FDMA）是另一种基于FDMA单载波频域均衡（SC-FDE技术）的变种。波分多址（WDMA）也是一种基于波分复用（WDM）FDMA的变种。WDM实际上相当于频分复用，但其常常用在光纤通信中，其中通常用术语波长来描述光信号调制的载波。WDM使用不同波长的激光传输不同的信号，每个波长指定为光纤中的一个子载波。由于光纤的数据传输速率比双绞线高得多，所以WDM波分复用通常用于聚合来自多个用户的数据。同步光纤网络（SONET）就是一种使用WDM的应用。

2.5 高级主题

本节描述数字调制的几个高级主题。具有较少电气工程背景的读者在第一次阅读本书时可以跳过本节。更多的教程和全面的理解可以在数据或数字课本中找到。对于需要可靠和安全的传输如军事和

无线应用中，常常要考虑使用扩频，因为信号经过扩展后在频域就会更像噪声，但同会更难以检测到干扰。直接序列扩频（DSSS）和跳频扩频（FHSS）是两个典型的方案。作为一种高级的复用或多路接入方案，码分多路接入（CDMA）为多个源实施扩频概念，通过正交或统计不相关的编码表示数据并将数据扩展到整个信道上。

与单载波调制相比，多载波系统在多个单独的载波信号上进行调制以提高带宽利用率并解决多径衰落。有了快速傅里叶变换（FFT）的实现，多载波调制技术（如正交频分复用（OFDM））目前已广泛应用在许多通信系统上。最近，在发射端有多个发射天线并且在接收端有多个接收天线的多输入多输出（MIMO）系统变得非常流行，因为它们在吞吐量和可靠性等性能得到了很大程度的提升。

表 2-7 显示了使用扩频、CCK 和 OFDM 技术的现有 IEEE 802.11 标准对比。

表 2-7　在 IEEE 802.11 无线局域网标准中使用的调制技术

	802.11a	802.11b	802.11g	802.11n
带宽	580MHz	83.5MHz	83.5MHz	83.5MHz/580MHz
运行频率	5GHz	2.4GHz	2.4GHz	2.4GHz/5GHz
不重叠信道数目	24	3	3	3/24
空间流的数目	1	1	1	1、2、3 或 4
每个信道的数据率	6～54Mbps	1～11Mbps	1～54Mbps	1～600Mbps
扩频方案	OFDM	DSSS、CCK	DSSS、CCK、OFDM	DSSS、CCK、OFDM
子载波调制方案	BPSK、QPSK、16QAM、64QAM	n/a	BPSK、QPSK、16QAM、64QAM	BPSK、QPSK、16QAM、64QAM

2.5.1　扩频

数据流的频谱可以分布到重宽的频带上。扩展可以提供额外的冗余。以减少无线通信因窃听、干扰和噪声所产生的脆弱性（编码）。扩频（SS）中的数据流是由特定的码序列（PN）序列所承载的。这是当数据流调制 PN 序列时所完成的。PN 序列是由重复出现的、由码片序列表示的 PN 编码所组成。将 PN 序列和输入数据流结合，就形成了数据流的扩频序列。码片本身就是一位。与数据位相比，码片只是 PN 码的发生器产生的位序列。因此，一个码片一般就是 -1 或 -1 幅值的矩形脉冲。所产生的扩展信号的能量分布到比数据流信号更宽的带宽上。像在纠错码中的冗余一样，冗余可以增强接收机上受损信号数据的恢复能力。

伪噪声（PN）编码和序列

一个 PN 序列，又称为伪随机数值（PRN）序列，并非一个真正的随机序列，但是能以一种确定的模式产生。序列是重复的，PN 编码在序列中重复出现。与数据流中的位是原子元素相类似，码片是 PN 码的原子元素。码片速度是每秒处理码片的数量。在图 2-32 中，PN 码是一个 11 个码片的 11 位巴克码（Barker code）。它不断地重复出现，就形成了 PN 序列。扩展序列是通过调节 PN 序列与数据流，利用 XOR 运算而生成的。在图 2-32 中的码片速率是数据传输速率的 11 倍。扩展序列的码片速率与 PN 序列的相同，但比数据流的速度高很多。这就解释了为什么发射一个扩展序列比数据流需要更大带宽的原因。

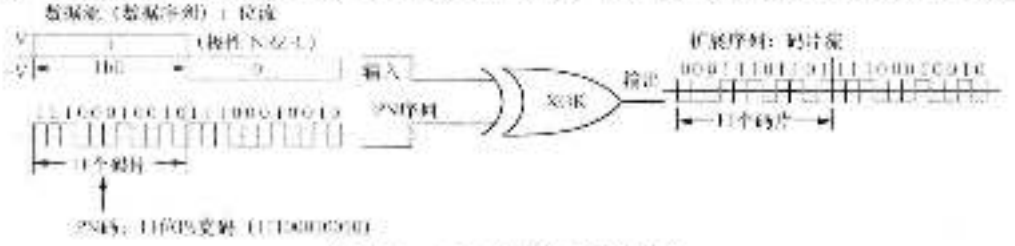


图 2-32　由 PN 序列扩展的数据流

图2-33显示了来自扩展序列的传输信号的扩展能量的广阔带宽。对扩展处理的测量——处理增益（PG），是由码片速率（C）与数据速率（R）之比定义的。码片速率总是高于位速率。此外，PN编码速率决定了传输扩展波形的带宽。处理增益是用来衡量扩频抵抗窄带干扰的性能优势。它可以看为解扩频后，接收器上的信号功率与干扰功率之比。假设数据速率是恒定的，那么码片速率越快，PG就越高。这意味着扩频占用了更大的带宽。如果PG足够大，扩展波形能够以小于噪声功率的功率通过噪声信道，而且数据流仍然可以在接收器上恢复出来。那么我们如何计算处理增益？例如，如果将一个11位的巴克码用于PN码，处理增益就可按如下计算：$10\log_{10}\frac{C}{R}\,\mathrm{dB}=10\log_{10}\frac{11}{1}\,\mathrm{dB}=10.414\mathrm{dB}$（码片/位）。这里，dB代表分贝。这是一个对数单位表示的物理量级，如码片速率，相对一个指定的参考水平，如数据传输速率。

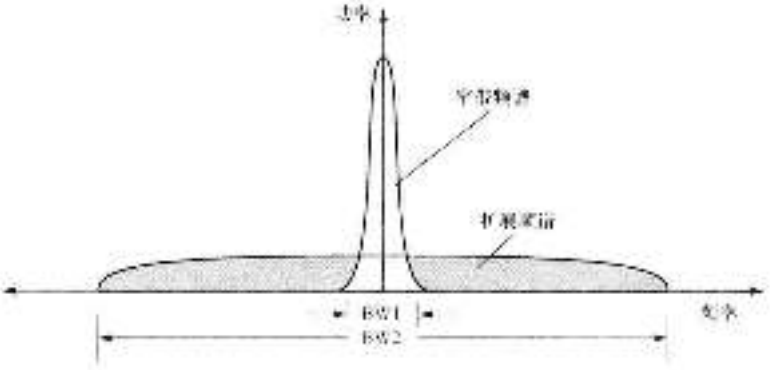


图2-33 扩展频谱和窄带频谱之间的对比

当扩展数据流时，PN序列起着关键的作用。其类型和长度决定着扩频系统的能力。一个精心选择的PN序列，有助于匹配的滤波器有效地拒绝大于一个码片延迟时间的多路径信号。我们稍后在CDMA中将看到类似的做法，其中PN码也用于接收器解扩接收到的信号。

巴克码、Willard码和补码键控

IEEE 802.11标准使用11位巴克码以码片速率11码片/数据符号作为一个PN码。巴克码具有良好的相关性。Willard代码是通过计算机模拟和优化获得的，可以提供比巴克码更好的性能。巴克码和Willard代码清单如表2-8所示。长的PN序列可以使用巴克码或Willard代码循环构造。DSSS技术使用11位和13位巴克码。IEEE 802.11b标准循环地使用11位巴克码以便以1Mbps或2Mbps扩展数据流。

表2-8 巴克码和Willard码

码长度（N）	巴克码	Willard码	码长度（N）	巴克码	Willard码
2	10或11	n/a	7	1110010	1110100
3	110	110	11	11100010010	11101101000
4	1101或1110	1100	13	1111100110101	1111100101000
5	11101	11010			

IEEE 802.11标准的高速扩展，采用补码键控（CCK）的调制方案，在2.4GHz频段以5.5Mbps或11Mbps编码数据。与巴克码不同，CCK序列可以完全消除旁瓣（side lobes）。在频谱中，除了需要主瓣（major lobe）外，旁瓣可以是任何其他瓣（lobe）。此外，CCK码字具有特殊的数学属性能够有效地抑制噪声和多径干扰，这里省略不讲。

扩频系统

图2-34显示了一个扩频系统，扩展信号传输通过带有窄带/宽带干扰和多路径的噪声信道。在扩频系统中，一个具有位速率为R_s的输入数据流d_s被码片速度为R_c的PN序列pn_s扩展，那么就获得了一个扩展码片流rx_s。基带带宽为R_s的输入数据流被扩展到更广泛的R_c上。带宽扩展因子SF或处理增益G_s是通过R_c/R_s得到的。接下来进行通带调制，rx_s变成了传输的rx。在接收器，扩频信号rx被天线接收，然后解调。接下来解调信号rx_s由一个使用自相关和互相关的PN序列pn_s解扩展。两个相关

序列即分别来自所需数据信号和多径信号的 PN 序列的自相关将接近 1，而两个互不相关的序列即分别来自所需数据信号和干扰信号的 PN 序列的互相关性将接近 0。输出数据流 d_i 是经过解扩后获得的。如果 pn_i 的 PN 码等于 pn_i 的 PN 码，那么 PN 序列 pn_i 与 pn_i 同步。输入的数据流 d_i 以输出数据流的 d_i 恢复，因为 pn_i 和 pn_i 的自相关性是明显的，即接近 1。

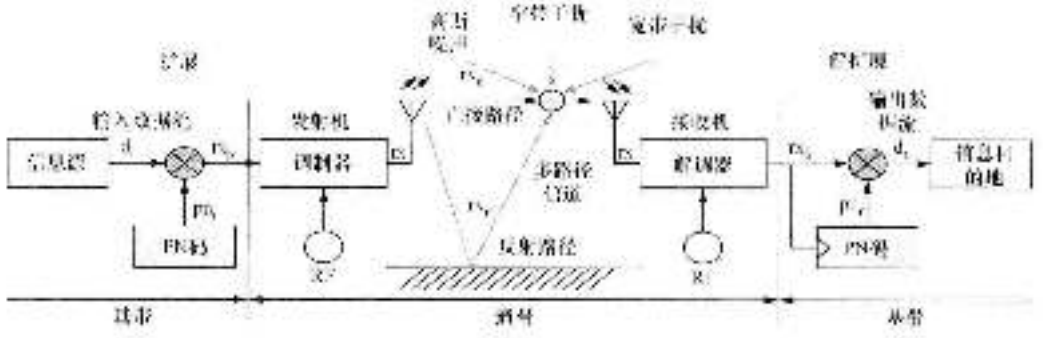


图 2-34 通过噪声信道的扩频系统

另一方面，如果 pn_i 和 pn_i 互相关性很小，无法解认，并像噪声似的，即接近 0，那么输入的数据流就不能在接收端恢复出来。如果不知道发射器上所用的 PN 码，那么接收器就会将扩展信号当做白噪声似的信号来处理。相应地，在 PN 码没有透露给第三方的情况下，通信隐私就仅被通信双方所掌握。

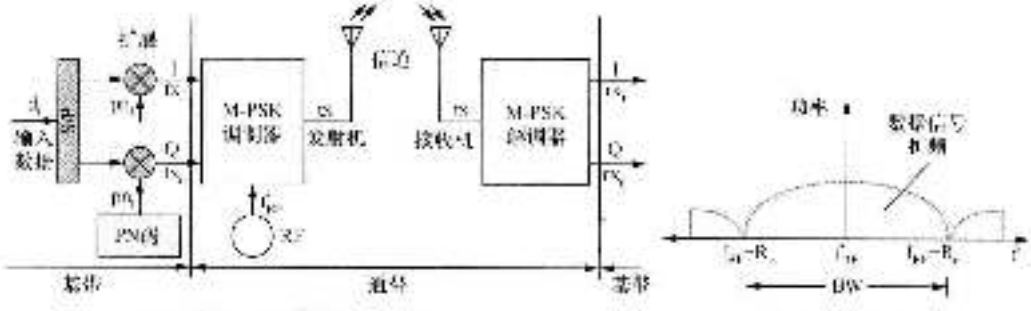
与多路复用技术一样，扩频可以将多个具有不同 PN 序列的多个数据源结合起来传输，但它需要具有重大的带宽用于传输。这样私密性和抗干扰能力也得到提高。扩展信号使之类似于噪声，使得信号能混合到背景干扰波形中并且经信道传输而不被检测或物所到。它是专门为无线通信设计的，因为传输介质暴露在公众下（公开曝光）并且传输信号很容易被截获。

在无线通信中，存在来自大气反射或折射，或者从地面、建筑物或其他物体反射的多种传播路径。这些多径信号，即图 2-34 中的 rx_i，可能会使经过图 2-34 中的直接路径接收到的信号 rx_i 产生波动。来自每条路径的信号都有自己的衰减和延迟时间。接收器必须能将从直接路径来的信号与其他路径来的信号以及干扰和噪声分开。如果多径信号延迟大于一个码片时间，那么它们就与所需信号无关，即自相关远离 1，互相关接近于 0。换句话说，从间接路径来的 PN 序列就不再与从直接路径来的 PN 序列同步。因此在扩频系统中的多径衰减不会造成重大的影响，并且可以有效地过滤掉。

直接序列扩频

从图 2-34 中可以看出，一个扩频系统通常后跟一个带通调制器，如 BPSK、M 进制 PSK（MPSK，M 大于 2）和 QAM。图 2-35a 显示了场景，其中直接序列扩展（DSSS）系统后跟一个 M 进制 PSK 调制器。由于 MPSK 信号调制器具有同相的同时又是正交的分量，所以系统需要两次扩展处理。

输入的数据被分成两个数据子流，每个都由一个 PN 序列扩展，一个是同相分量，一个是正交分量。DSSS 使用一个 PN 码或它的补位来代替数据流中的每一位。传输信号频谱是由图 2-35b 中扩展流的码片速率 R_c 决定，而不是由数据流的位速率 R_c 决定。



a) 使用MPSK调制的DSSS系统　　b) 扩展序列频谱

图 2-35 DSSS 和扩展序列频谱

干扰和噪声对 DSSS 的影响

假设一个 DSSS 系统受到干扰信号 i 的影响。在接收端，接收信号是一种包括干扰信号和噪声的复合信号。复合信号由一个 PN 序列解扩展以便能从发射器恢复出数据流。接下来解释扩频技术如何减轻干扰和噪声的影响。

- 如果 i 是一个窄带干扰，这意味着 i 是从另一个数据流中来的信号。解扩展后，来自窄带干扰产生的结果就变成一个高平的频谱并且比所需数据流的频谱功率密度比低很多的扩展序列。可以使用低通滤波器将它过滤掉。因此，扩展可以排除窄带干扰，但传统的窄带技术却不能排除干扰。
- 如果 i 是一个来自另一个用户的扩展序列但使用不同 PN 序列的宽带干扰。解扩后，宽带干扰的序列就再次是扁平的，因为宽带干扰使用了不同的 PN 序列，那么这种互相关的点乘积与使用相同 PN 编码的宽带信号相比会显得相当小并有点像噪声。扁平干扰可以通过低通滤波器很容易地过滤掉。这表明，扩频可以消除宽带干扰。
- 如果 i 是噪声，那么噪声的结果序列在码片速度上仍然像噪声一样地扩展序列，并且具有低功率密度。扁平的高斯噪声频谱也可以通过低通滤波器过滤。扩频系统的信号对噪声更有免疫性，特别是当信号经过一个有噪声的信道时，其效果就会更加显著。

IEEE 802.11 通常允许使用来自 2.412～2.462GHz 的 11 个信道（每个信道 5MHz 宽）。信道 1 的中心在 2.412GHz。IEEE 802.11 使用以 1Mbps 和 2Mbps 的速度进行 DSSS 调制，而 IEEE 802.11b 以 5.5Mbps 和 11Mbps 速度使用 CCK 调制。IEEE 802.11g 支持扩展速率 PHY（ERP）。ERP-DSSS、ERP-CCK 和 ERP-OFDM 的调制用于向后兼容。在 WLAN 中的 DSSS 物理层包括两个子层：物理层汇聚协议（PLCP）和与物理介质相关（PMD）子层。PLCP 主要是用于成帧。图 2-36 显示了 PMD 子层，扩展器就位于这个子层。

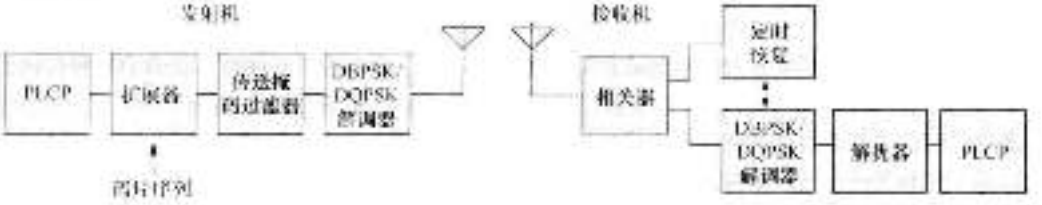


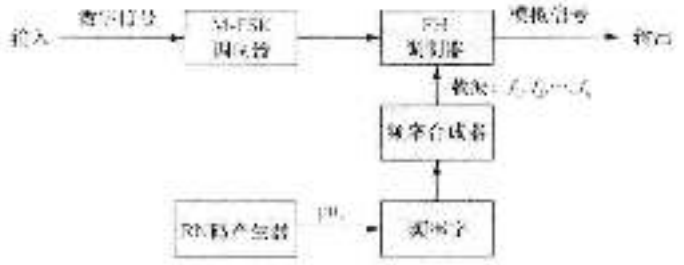
图 2-36 DSSS 收发机

跳频扩频

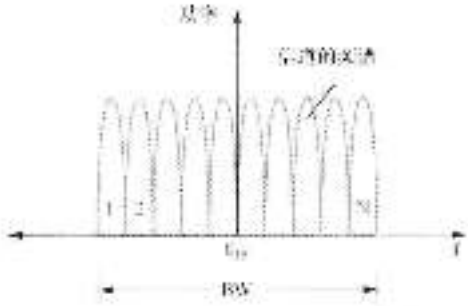
跳频扩频（FHSS）将带宽分为 N 个子信道，以便于传输信号能在这些子信道之间跳跃。传输信号会在每个子信道中停留一段时间，称为停留时间。在图 2-37a 中，PN 编码发生器产生一个映射或频率高的 PN 序列 ρ_i，用来表示表中的一个频率跳跃序列。这些频率同被依次传送到频率合成器以便产生具有不同频率的 N 载波，如图 2-37b 所示。发射机在这些 N 载波之间按顺序跳跃。在图 2-37 中，FHSS 系统将 M-PSK 调制器与 FH 调制器结合起来分别用于调制和跳频。FHSS 发射器和接收器使用相同的跳频模式。在每一跳中，传输信号的带宽与 M-PSK 的输出信号相同。驻留在 FHSS 子信道中的信号就是窄带信号。

FHSS 为源信号利用一个具有不同频率的载波池。每次使用其中一个载波，因此消息可以被不同的载波所传输。如果他中有 n 个载波，那么所需要的带宽就是由单载波所使用带宽的 n 倍再加上一些防护频带。与通过 PN 序列扩展源编码的 DSSS 不同，FHSS 从由 PN 编码导出的映射表中选择一个频率。只要每一跳使用不同的频率，所需的带宽就可以在多个用户之间共享。不同频率之间共享带宽的概念类似于频分复用（FDM）技术。PN 编码发生器为频率合成器重复产生位模式。这些模式可用来选择载波，以便传输在跳频期间输入的消息。可能有多个用户选择同一个子信道传输，这样一来就会产生干扰。当一个符号在几跳中重复传输时，只要在大多数跳中没有受到干扰，那么接收器仍然可以恢复信号。

如果跳周期时间很短，那么对于一个不了解 PN 码的窃听者来说，拦截信号就会很困难。对于不了解 PN 码的入侵者按照用户使用的序列通过跳到不同频率来阻塞流量同样是很困难的。FHSS 应用于蓝牙和早期的 IEEE 802.11 中。然而，当用于在很短时间间隔内进行高速传输的快速跳频时，发射器和接收器之间的同步会变得很困难。因此，它不用于 IEEE 802.11a/b/g/n。



2) FHSS带有一个PN码产生器以选择最终跳频率



3) 一个由N个子信道组成的FHSS信道频谱

图2-37 FHSS系统和子信道频谱

码分多路接入（CDMA）

码分复用（CDM）允许从多个独立源到达的信号同时在同一频段上通过一个信道传输。CDM是一种扩频技术，是码分多路接入（CDMA）所使用的多路接入技术。因此，CDMA是一种扩频多路接入（SSMA）技术。与TDMA和FDMA不同，CDMA没有按时间或频率将一个物理信道分成多个子信道。在CDMA系统中，每个用户在同一时间占据物理信道的整个带宽，同时使用单独的正交码或PN码。事实上，CDMA通过一组正交码或PN码来复用不同的用户。CDMA的几个变种都有其自己的复用方法，如基于DSSS的直接序列CDMA（DS-CDMA）和基于FHSS的跳频码分多路接入（FH-CDMA）。

CDMA也可以划分为同步CDMA和异步CDMA。同步CDMA系统采用正交码，而异步CDMA使用PN码。正交码是严格的零成对内积向量，而PN码是统计量（如果密切相关，则成对自相关接近1；如果非密切相关，则成对互相关接近0）。这两种码在接收器上均使用扩频增益以便确定所需要的信号而不是其他不想要的信号。如果所需要的用户信号与其他用户的信号不相关，那么在同步CDMA中的内积是零，在异步CDMA中的互相关内积趋近于零。与通用扩频中多径干扰的处理方案相类似，如果悉移大于一个码片时间，那么所需要的信号将具有零内积或者与经同一正交编码或PN编码调制过的信号具有低的自相关性。而且，此属性有助于消除多径干扰。

同步CDMA

在同步CDMA中，正交编码被映射到相互正交具有内积为零的一组向量。正交编码将分配给用户用以扩展用户数据频谱。编码可以从如图2-38所示的正交可变扩展因子（Orthogonal Variable Spreading Factor，OVSF）代码树中得到。这些代码是成对的正交。

OVSF码树基于Hadamard矩阵，即一种具有+1或-1项、其行相互正交的方阵。将每种编码或码片编码分配到一个单独的用户来表示数据流中的一位。例如，如果一个正交码为（1，-1，1，-1），那么编码矢量可以表示为v = 1，-1，1，-1。如果v表示0位而$-v$表示1位，那么“10110”的数据流可以表示成为（$-v$，v，$-v$，$-v$，v）。数据流扩展为（-（1，-1，1，-1），（1，-1，1，-1），-（1，-1，1，-1），-（1-1，1，-1），（1，-1，1，-1））。详最终成为（-1，1，-1，1，-1，1，-1，1，-1，1，-1，1，-1，1，-1，1，-1，1，-1，1，-1）。处理是通过XOR（异或）运算符实现的，如图2-39所示。数据信

具有脉冲持续时间 T_c，而正交编码信号具有 T_s 持续时间。换句话说，数据信号的带宽是 $1/T_c$，正交码带宽是 $1/T_s$。扩展因子或处理增益，是正交信号与数据信号的带宽比率，T_s/T_c。这限制了用户总数的上限。

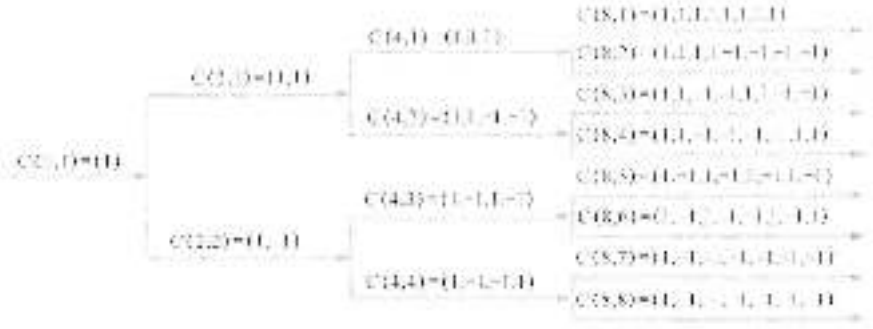


图 2-38 同步 CDMA 的 OVSF 代码树

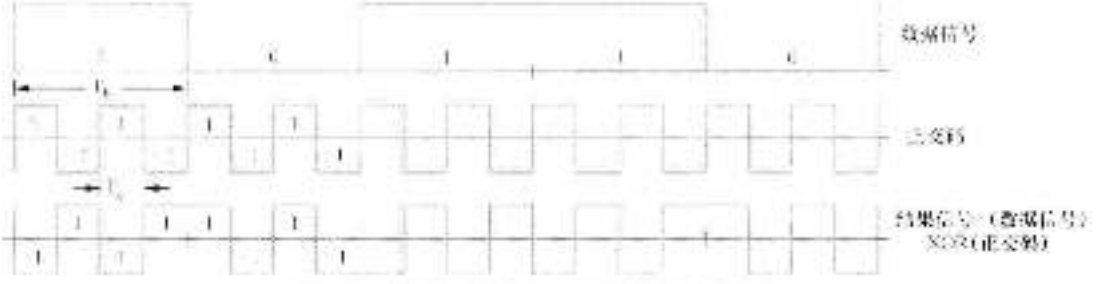


图 2-39 使用了信道的正交编码来扩展数据信号

异步 CDMA

异步 CDMA 采用 PN 码。与适用扩频一样，PN 码是具有随机性的二进制序列在接收端以确定性的方式再生出来。PN 码用于异步 CDMA 扩展和解扩展用户的信号，而正交码则用于同步 CDMA。它们在统计概率上几乎无关。与同步 CDMA 不同，其他用户的信号的确以噪声似的信号出现，并会降低干扰所需要的信号。也就是说，具有不同 PN 码的信号成为具有特定 PN 码所需信号的宽带噪声。即使接收到的信号与需要的信号具有相同指定的 PN 编码，但如果它在某一时间偏差内接收到，它也是以所需信号的噪声形式出现。

尽管同步 CDMA、TDMA 和 FDMA 能够完全摒弃其他信号，分别由于代码正交性、时隙、频率信道，但异步 CDMA 只能部分地摒弃不想要的信号。如果不需要的信号比所需信号强很多，那么所需信号将会受到严重的影响。因此，需要有一种功率控制方案来管理每个站点发射的功率。异步 CDMA 虽然有这种缺点，但也具有以下优点。

1）异步 CDMA 比 TDMA 和 FDMA 更有效率地使用频谱。TDMA 中的每个时隙需要有一个保护时间来同步所有用户的传输定时。在 FDMA 中的每个信道都需要有一个防护频段，以防止来自相邻信道的干扰。防护时间和防护频带两者都会浪费频谱的使用。

2）异步 CDMA 能够为用户灵活地分配 PN 代码，不对用户数量做严格的限制，与同步 CDMA、TDMA 和 FDMA 只能将其资源分配给固定数量的并发用户，具体取决于固定数量的正交码、时隙、频带。这是由于 PN 码很低的但非零互相关的性质以及自相关和互相关换作所决定的。在电话和数据通信的高流量突发中，异步 CDMA 能够更加有效地将 PN 码分配给更多的用户。然而，在异步 CDMA 中的用户数量仍然受到误码率的限制，因为信号干扰比（SIR）与用户数量成反比。

3）与使用正交码的同步 CDMA 一样，基于 PN 序列的抗干扰能力的异步 CDMA，都提高水平的隐私性。使用码的使用赋予扩频信号具有噪声一样的属性。如果没有具体指定 PN 序列，那么非同步 CDMA 接收器就无法获得消息。

CDMA 的优点

这里我们总结使用扩频 CDMA 的优点。CDMA 技术可以有效地减少多径衰落和窄带干扰，因为 CDMA 信号是占用了很广范围带宽的扩频信号，仅有很少一部分信号受到窄带干扰和多径衰落。冲突的部分可以通过过滤而去掉，而丢失的数据也可以通过使用纠错技术加以恢复。CDMA 可以有效地避免多径干扰，因为来自多径的延迟信号与所需信号变得几乎不相关，即使它们两者都具有相同的 PN 码也是如此。

CDMA 能够使用同一频率，因为信道可被各种正交码或 PN 码分隔开，而 FDMA 和 TDMA 却不能。蜂窝系统中，相邻蜂窝频率可用的能力使得 CDMA 能够使用软切换技术。软切换是一种蜂窝电话在一次呼叫期间能够同时与多个蜂窝连接的特点。移动手机中存有相邻蜂窝的功率测量列表，以便决定是否要进行软切换。软切换使得移动站点保持一种更好的信号强度和质量。

2.5.2 单载波与多载波

多载波调制（MCM）将数据流分割成多个数据子流，每一个数据子流为窄带子信道调制一个对应的载波。调制过的信号可以经过频分多路复用（FDM）进一步复用，这称为多载波传输。由 MCM 产生的复合信号是一个宽带信号，这个信号可以更好地避免多径衰落和码间干扰。相反，如果子信道被码分多路复用（CDM）所复用，我们就称之为多码传输。这里仅讨论了用于多载波的正交频分复用（OFDM）。

正交频分复用

OFDM 技术的主要特点是子载波的正交性，这样能使数据同时通过由在紧密的频率空间而没有相互干扰的正交子载波构成的子信道。OFDM 将复用技术、调制技术和多载波技术结合起来构成一个通信系统。OFDM 就是由逆向快速傅里叶变换和快速傅里叶变换（IFFT/FFT）实现的。不同于传统的 FDM 技术，每个数据流仅占用带有一个特定载波的一个子信道，OFDM 而是将数据流分开以便能够同时使用多个子信道。使用多载波的好处是：如果一个子信道出现故障，数据流仍然可以在接收方恢复，因为只有部分数据受到（如突发错误的）损坏。

OFDM 系统的结构如图 2-40 所示。IFFT 执行多载波调制器功能以产生 OFDM 复合信号。将一个循环的缀添加到 OFDM 信号前作为守护间隔，以避免符号间干扰（ISI），这个前缀会在接收方删除。一个符号是一种持续一段固定时间的信道状态，它可以被一位或多位编码。因此，一序列符号或符号之间的转换就可以用来表示数据。在接收端的解调器是通过 FFT 实现的。

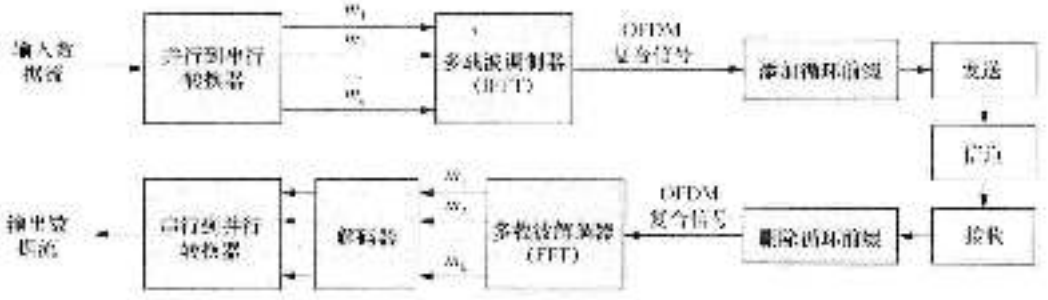


图 2-40 多载波 OFDM 系统

带有逆向快速傅里叶变换和快速傅里叶变换的正交频分复用系统

多载波调制器通常由一个 IFFT 处理实现，如图 2-41 所示。为了产生 OFDM 信号，IFFT 通过将单个数据子流调制过的正交载波信号结合起来。IFFT 有一个对应的 FFT。无论时域信号还是频域信号，都可被 FFT 或 IFFT 处理。如果信号被一对 IFFT 和 FFT 处理，输出结果就与源信号相同。这就是 OFDM 机制如何通过一对 IFFT 和 FFT 实现的。在图 2-41 中，IFFT 将频域信号转换成时域信号，而 FFT 则与此相反。这里，IFFT 的时域输入位被认为是在频域中的频率振幅，IFFT 的输出复合信号是类似于时域的信号。IFFT 和 FFT 都是数学概念，两者都是线性过程并且完全可逆。

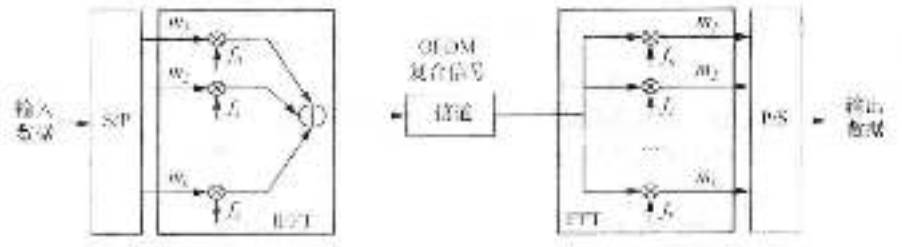


图2-41 IFFT和FFT的功能框图

正交性

频域信号的正交性，如图2-42中所示，将此采样格点的两个信号是相互正交的。每个频率分配一个子载波或子信道，并且可应用一种（例如，QAM或QPSK）经典调制方案。

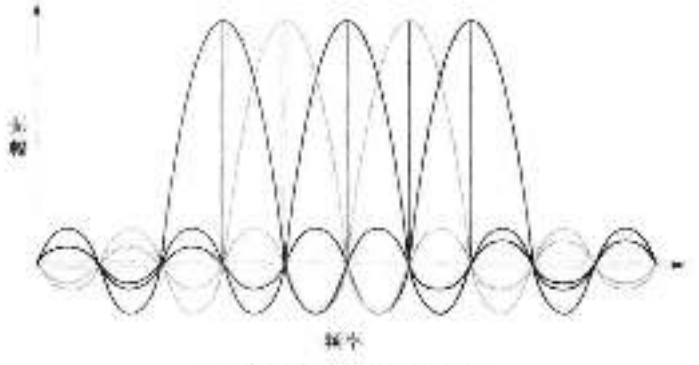


图2-42 OFDM的正交图

使用循环前缀作为符号之间的守护间隔，简化了传输信号的直接卷积和对循环卷积的多径信道响应，这就相当于采取FFT操作后的一种直接乘法，因而消除了ISI。然而，OFDM要求发射器和接收器之间精确的频率同步，因为任何频率漂移都会破坏子载波的正交性，并导致载波间干扰（ICI）或子载波间的串扰。

多径衰落

在无线通信中，多径传播是一种被传输的信号通过不同的路径以不同的时刻到达接收者天线的现象。由于传送器和接收器周围存在的反射物，传输信号会产生反射，而使信号从多条路径到达接收器。多径信号可能会导致不同程度的有益或有害的干扰，相移漂移、延迟和衰减。强烈有害的干扰就是指信号的严重（深度）衰落，使信噪比突然降低，并导致双方无法通信。多径信号可以视为传输信号的直接卷积和多径信道的响应。但是通过接收端的信道平衡器，通过在频域调制信号，以及通过使用循环前缀，这种效应可以被去掉或减轻，直接卷积被简化成循环卷积，并且依次在接收端采取过FFT操作后它就变成频域中的直接乘积。因此，OFDM完全去掉了在接收端需要的复杂平衡器，并简化了接收器的设计。以防为一些现针对特定子载波的深度衰落，接收信号仍旧可以通过带有错误纠正码的编码技巧进行恢复。

OFDM的应用

OFDM应用于ADSL和VDSL宽存接入、电力线通信（PLC）、DVB-C2、IEEE 802.11a/g/n无线局域网、数字音频系统（如DAB和DAB+）、地面数字电视系统和IEEE 802.16e的WiMAX。OFDM设计用于一个待发送的位流以一系列OFDM符号序列通过一个通信信道，但是也可以用于通过时分、频分或码分多路复用。正交频分多接入（OFDMA）为不同用户分配不同的子信道以实现FDMA。

2.5.3 多输入、多输出

一个多输入和多输出（MIMO）系统在发送端和接收端基本上都包括天线阵列和自适应信号处理

单元。该系统将多种分集方案用于数据通信。分集方案（例如，时间分集、频率分集、空间分集、多用户分集）使信号具有抵抗信号衰落的能力。时间分集要求信号在不同的时刻传输，而频率分集要求信号通过多个频率信道传输。空间分集允许信号从多个发射天线发送和被多个接收天线所接收。多用户分集被随机的用户调度，根据信道信息选择最佳用户来实现。MIMO 利用这些分集方案，提高系统的可靠性。

MIMO 既可用于有线也用于无线系统。例如，千兆 DSL（数字用户线）是一种有线的应用。这里，我们将重点放在使用天线阵列的 MIMO 无线传输系统。天线阵列通过使用多个发射和接收天线提供空间分集以提高质量和可靠性，如无线链路的误码率（BER）。与天线阵列之间的望路提供信号传播通过的多条路径。到达接收方具有不同传播延迟和衰落的多径信号，然后创建空分的信道。MIMO 将在传统无线系统中多径传播的缺点转变成了优点，特别是对于那些没有在视距内传输的系统来讲更是如此。当 MIMO 利用多径传播时，用户的数据传输率也就随之增加。

在空分复用（SDM），也称为空间复用（SM）中，多个位流通过不同的天线并行传输。空分多址（SDMA）是一种信道接入方法，它经过空间多路复用和分集创建并行的空间信道。SDMA 使用智能天线技术（这是从 MIMO 演变而来）和移动站点空间位置信息，以便在基站实现辐射模式，基站上的发送和接收都适应每个用户以便获得最高的增益。相反，在传统的蜂窝系统中，基站没有移动站点位置的信息，因此信号会在各个方向发送。这可能很会发射功率，并且与使用相同频率或蜂窝相邻产生干扰。

MIMO 系统的分类

MIMO 可以根据信道的使用情况或用户数进行分类。根据已知的信道知识，MIMO 可以分为三组：预编码、空间复用（SM）和分集编码。预编码方法需要信道状态信息（CSI），但分集编码方法则不需要。空间复用既可以使用也可以不使用信道知识。

信道感知的预编码利用有关信道状态的反馈信息来安排发射机上的波束形成或空间处理。波束形成是一种信号处理技术或空间过滤器，它将从小型单元向天线来的一组无线电信号组合起来，模拟一个更大的有向天线。这种模拟的定向天线用以确定传输信号的方向。这种预编码方法可以增加信号增益和减少多径衰落。SM 技术需要了解天线配置，以便将一个高速率的信号置分成多个低速率的子流，然后再使用相同的频率信道在不同的天线上传输。分集编码方法不需要有关信道的信息。信号在发射机通过空间编码进行编码，以便利用多天线链路中的独立衰落。使用分集编码技术的 MIMO 系统就没有波束形成或阵列增益。

如果 MIMO 系统根据用户数来分类，MIMO 的类型就可以分成单用户 MIMO（SU-MIMO）和多用户 MIMO（MU-MIMO）。SU-MIMO 是一种点对点通信，主要关心链路的吞吐量和可靠性，利用空间码和流复用传输。多天线扩大了信号处理和检测自由度。因此，SU-MIMO 提高了物理层的性能。

然而，MU-MIMO 系统做到系统的吞吐量。MIMO 既应用在物理层也应用到链路层。在链路层，多址接入协议在空间维度中极大地提高了 MIMO 天线阵列的性能优势，比如，更高的每个用户速率或信道的可靠性。MIMO 需要多用户信息设计用户的调度以便提高系统的吞吐量。因此，MU-MIMO 将物理层的编码和调制与链路层的资源分配和用户调度结合起来。一种最佳的用户调度取决于预编码的选择和信道状态反馈技术。使用 MU-MIMO 技术的无线通信会引起跨层设计问题。

MU-MIMO 系统

我们已经简要地介绍了使用预编码技术的 MU-MIMO 系统的体系结构，图 2-43 显示了无线广播信道构建天线阵列。在图 2-43 中，带有多个发射天线的基站（BS）向移动站点（MS）发送消息。每个移动站点配备了一个 M_r 天线的阵列，具有一个接收实体来处理多个并行子流。接收实体首先采用最小均方误差（MMSE）过滤和连续干扰消除（SIC）处理每个子流。MMSE-SIC 模块具有两个方面功能：干扰归零和干扰消除，既可以删除也可以减少来自那些已经检测发现的子流的干扰。然后这些子流再通过空间解复用合并起来。最后，在接收器获得输出数据流。

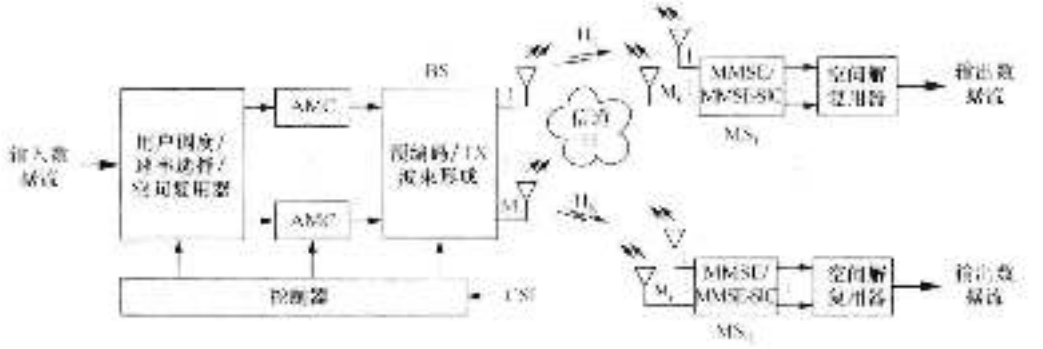


图2-43 多用户MIMO系统

因为这种体系结构是一种需要利用信道信息进行多用户调度的ML-MIMO系统，所以BS使用控制器来收集从接收器反馈的信道状态信息。此信息包括信道方向信息（CDI）和信道质量信息（CQI）。CDI确定了波束形成的方向，而CQI调整每个波束传播的速率。控制器在基站上使用信息执行时空处理，如多用户调度、能量和调制自适应（如自适应编码和调制（AMC））、波形等自适应）和波束形成。控制器控制AMC选择编码、调制类型和协议参数。为了得到更高的吞吐率，AMC的选择需要根据无线电链路状况，如路径损耗、干扰和接收器的灵敏度，以提高接收天线的使用效率。简言之，基站结合波束形成的CSI反馈信号信息，在发送消息的同时以获得最佳的传输模式。AMC函数和预编码的目的旨在最大限度地提高传输的吞吐量和最大限度地减少错误率。

ML-MIMO系统利用用户的分集（多样性）进行用户调度。有效的用户调度在时间和空间域都提供了很多优势，如空间波束形成。上行链路反馈信号为高级接收器，它可以与改进的SIC接收机结合，例如，所有发射天线可以根据SIC或最小均方误差（MMSE）分配给最佳用户。

总之，MIMO系统并行地通过多个天线发送多个数据流，以提高可靠性和频谱效率。而空分组编码（STBC）可帮助实现全面的发射分集。波束形成能够通过灵活干扰和线性地结合波束来提高链路的可靠性。发送和接收分集可以减少衰落的波动以获得分集增益。空间复用通过不同的发射天线在同一时间发送不同的数据信号来利用复用增益。

MIMO的应用

GSM增强数据率演进（EDGE）和高速下行链路分组接入（HSDPA）会使用速率适配算法的MIMO系统，以便根据无线电信道的质量来管理编码和调制的方案。标准3GPP的WCDMA/HSDPA使用带有用户调度的MU-MIMO。通过在每个40MHz信道上有四个空间流来增加多输入多输出（MIMO），IEEE 802.11n-2009将网络的吞吐量提高到最高600Mbps。此外，IEEE 802.11n在链路层帧进行帧聚合。

开源实现2.2：带有正交频分复用的IEEE 802.11a发射机

概述

802.11a是一种用于无线通信的IEEE标准。该标准采用OFDM调制方案，它也广泛地用于许多其他无线通信系统中，包括WiMAX和LTE。可以从OpenCores网站https://opencores.org下载一个开放源码的范例，其中介绍了以Bluespec系统Verilog语言实现（BSV）的802.11a发射机。我们首先在此给出OFDM发射机中的模块和处理流程概述，然后学习卷积编码器的操作。

框图

图2-44说明了OpenCores 802.11a发射机的体系结构，它主要由控制器、扰码器、卷积编码器、交织器、映射器、逆快速傅里叶变换（IFFT）和循环扩展器组成。具体描述如下：

- 控制器：控制系统接收来自MAC层的数据分组作为物理层（PHY有效载荷），并为每个数据分组创建头部字段。
- 扰码器：扰码器将每个数据分组与一个伪随机位模式进行异或运算。

- 卷积编码器：卷积编码器为它所接收的每一位生成两位的输出。
- 交织器：交织器重新排序在每个数据分组中的位。它以分组大小为48、96或192位的块来操作OFDM符号，具体取决于所使用的速率。
- 映射器：映射器也运行在OFDM符号级。它将交织位的数据翻译为64复杂数，这是针对不同频率使“看到”的调制值。
- IFFT：IFFT将复杂的调制值映射到每个子载波上并执行一个64点逆向快速傅里叶变换以使得它们的峰到时域。
- 循环扩展器：循环扩展器通过在完整消息主体上添加头部和尾部来扩展IFFT的符号。

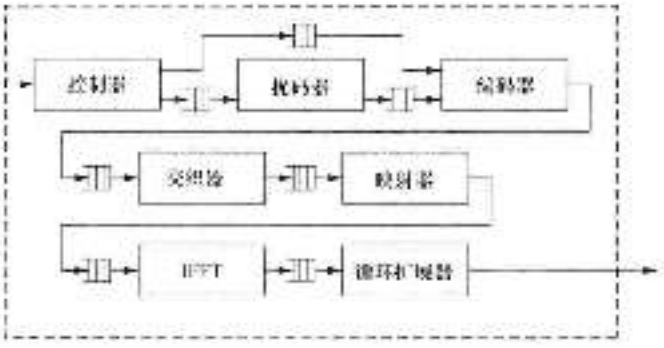


图2-44 802.11a发射机的框图

OpenCores 802.11a发射机的设计仅实现802.11a规范中最低的三个数据传输速率（16，12，24 MB/s）。在这些速率下，“打孔纸带机”不对数据进行操作，所以我们从这里的讨论中省略掉。

数据结构和算法的实现

最上部的模块，称为Transmitter.bsv，处理传输流量。流量从Controller.bsv开始，首先创建一个数据头部（在PHY数据分组格式中，信号字段长度为24位），然后从MAC层得到数据流（在PHY分组格式中的数据字段）。因此，该控制器具有两个FIFO输出：一个是由24位元素组成的toC（控制元素），另一个是包含多个24位元素的toS，这取决于MAC层数据长度（数据元素）。然后toS数据元素输入到scrambler.bsv中并与位伪随机模式进行异或运算。同时，将toC控制元素传送到Convolutional.bsv中并以1/2编码速率进行编码。

在下一轮循环中，扰码数据元素开始编码，仍以1/2编码速率，因为支持的数据传输速率只有6、12和24MB/s。convolutional.bsv将24位的输入元素编码成一个48位编码的FIFO元素（1/2编码率）。interleaver.bsv从FIFO队列中获取编码位并对OFDM符号进行操作，以分组大小为48、96或192位进行，具体根据使用的速率决定。它在一个数据分组内重新排序位。

假设每个分组每次只操作一个块，那么这意味着最快速度可以预期每四次循环仅输出一次，其中一个192位分组块的大小需要四个输入编码位流。mapper.bsv将交错（交织）位（48位）直接翻译成64复杂数据（频率“看到”）。IFFT.bsv执行64点的逆向快速傅里叶变换，这样就将复杂频率数据翻译转换成时域数据（IFFT处理位的符号带有64个复杂数据）。OpenCores的802.11a发射机提供多种IFFT的实现，并提出了一种基于四点蝴蝶块的组合设计。最后，由CycleExtender.bsv创建一个完整的发送消息。该信息结构的后16位复杂数据是由输入至快速傅里叶变换的结果一位接着一位组成。

为了避免冗长叙述，我们仅详细解释图2-45中关键卷积编码器编程实现的BSV代码分段。图2-46显示了一个卷积编码器的电路，它可以简洁地描述成每周期带有移位寄存器和多个或门的位。History表示是由移位寄存器中的输入位和所有延迟寄存器（Tb）位组成的位流。对于每个for循环迭代，两

个输出编码位从多有输入位与延迟寄存器位进行 OR 操作而产生出来。编码的位以位流形式存储在 rev_output_data。每个延迟寄存器 newHistVal 的值将保存为相同数据分组的接下来的 24 位输入位流。对于一个新的数据分组，每个延迟寄存器的值被复位至零。

```
Bit#(n6) history; // Bit#(n6) means bit vector with
length (n+6)
if(input_rate == RNone) // for new entry of the same
packet at next cycle
     history = {input_data, histVal};
else
     history = {input_data, 6'b0}; // for an new packet

Bit#(nn) rev_output_data = 0;
Bit#(1) shared = 0;
Bit#(6) newHistVal = histVal;

for(Integer i = 0; i < valueOf(n); i = i + 1)
begin // encoding
     shared = input_data[i]^history[i + 4]^history[i +
     3]^history[i + 0];
     rev_output_data[(2*i)-0] = shared^history[i +
     1];//output_data A
     rev_output_data[(2*i)-1] = shared^history[i +
     5];//output_data B
        // save the delay register status for next new
        entry
        // only last update will be saved
     newHistVal = {input_data[i], newHistVal[5:1]};
end
     // enqueue encoded bit stream
RateData#(nn) retval = RateData{
     rate: input_rate,
     data: reverseBits(rev_output_data)};

outputQ.enq(retval);

// setup for next cycle
histVal <= newHistVal;
```

图 2-45 Convolutional.bsv 中的代码分段

练习

当您使用图 2-46 中所示的卷积编码器对这些位进行编码时，计算输出位和状态。在表 2-9 中总结每次迭代后状态和输出位是如何改变的。

表 2-9 卷积编码器的输出位和状态

迭代	1	2	3	4	5	6	7	8	9	10
输入位	0	1	1	0	1	1	0	0	0	0
移位 Regs [543210]	000000									
输出 [A, B]										

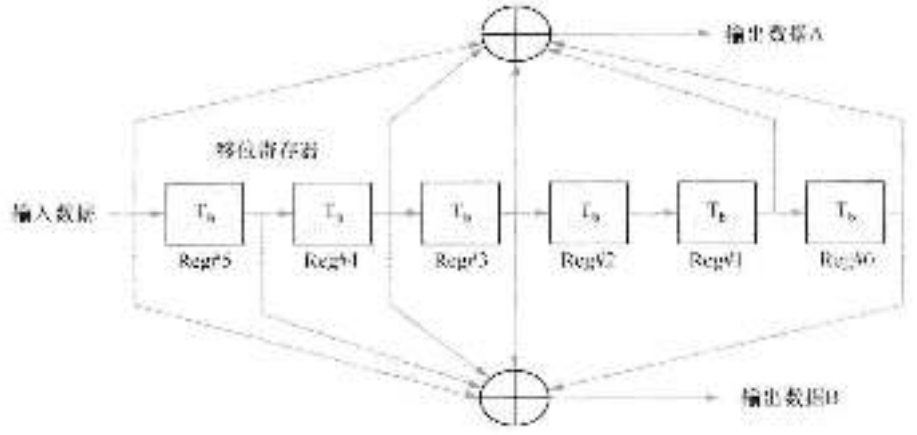


图 2-46 定义在 802.11a 中的卷积编码器电路

历史演变：蜂窝标准

蜂窝标准经历了从 1G、2G、3G 到 4G 的演变。其物理层属性如表 2-10 所示。在 1G 中，数据信号是以模拟形式发送的。例如，AMPS 和 TACS 标准。在 2G 中，数据信号是以数字形式传输的，GSM 标准就是其中非常流行的一种。3G 标准为多媒体和扩频传输提供高速 IP 数据网络，包括 CDMA2000 和 LTE（长期演进）。现在，4G 标准必须支持全 IP 交换网络、移动宽带接入、多载波传输（OFDM）和 MIMO，或者所谓的无线阵列或智能天线的所有功能。LTE-Advanced 和 WiMAX-m（IEEE802.16m）标准是 4G 的两个候选。此外，有些人认为，支持多种协议的融合解决方案也可以考虑作为 4G。因此，软件无线电和认知无线电也可以考虑作为 4G 技术。OFDM 技术而不是 CDMA 被选用到了 4G 中，就是因为它简单的调制和复用。它可以实现在 4G 标准指定的千兆位速度要求。Turbo 码用于 4G，以尽量减少接收端所需的 SNR。

表 2-10 蜂窝标准的物理层属性

蜂窝标准	AMPS	GSM 850/900/1800/1900	UMTS（WCDMA，3GPP FDD/TDD）	LTE
代	1G	2G	3G	3G
无线电信号	模拟	数字	数字	数字
调制	FSK	GMSK/8PSK（仅限 EDGE）	BPSK/QPSK/8PSK/16QAM	QPSK/16QAM/64QAM
多址接入	FDMA	TDMA/FDMA	CDMA/TDMA	DL：OFDMA；UL：SC-FDMA
双工（上行/下行）	n/a	FDD	FDD/TDD	FDD + TDD（非成对 FDD）
信道带宽	30kHz	200kHz	5MHz	1.25/2.5/5/10/15/20MHz
信道数	333/666/832 信道	124/124/374/299（8 个国际信道）	取决于服务	> 200 用户每蜂窝（对于 5MHz 频谱）
蜂窝数据率	信号速率 = 10kbps	14.4kbps 53.6kbps（GPRS） 384kbps（EDGE）	64kbps（静态）/384kbps（步行）；2Mbps（室内）；10Mbps（HSDPA）	DL：100 Mbps；UL：50Mbps（关于 20 MHz 频谱）

历史演变：LTE-Advanced 与 IEEE 802.16m 对比

LTE 标准是 4G 的前期技术，但它并不完全符合 IMT-Advanced 的要求。因此，LTE-advanced 标准，是一种 LTE 的进化，必须到达或超过 IMT Advanced 的要求。LTE-advanced 向后与 LTE 兼容，它具有多个技术特点，如协调多点传输和接收，支持更宽的带宽，空分复用（SDM）以及中继功能。中继功能增加了高数据速率的覆盖，组的移动性，临时网络部署，并提供新领域的覆盖。LTE-advanced 也支持在 20MHz 频段的频谱聚集以获取可使用的连带宽。这可能会导致每个方向，无论下行还是上行，达到总共 100MHz（5 块）的传输。支持高级服务的 LTE advanced 增强给位在高移动性的数据速率为 100Mbps，在低移动性时为 1Gbps。与 WiMAX 不同，LTE advanced 在上行链路（UL）上选用 SC-FDMA，两者在下行链路（DL）上都使用 OFDMA。因此，LTE Advanced 技术比 WiMAX 技术更加节能。

WiMAX 是一种由 IEEE 802.16 开发的标准，它的演进标准 WiMAX-m 是 LTE-advanced 的一种替代。WiMAX-m 是对 IEEE 802.16e 标准在 PAR P802.16m 下的一项修正。WiMAX-m 和 LTE-advanced 包含了多项“突破”技术——即 OFDM，MIMO 和智能天线。这些技术使全 IP 网络成为可能。LTE-Advanced 和 WiMAX-m 两者都支持全 IP 分组交换的网络，移动宽带接入和多载波特性。

移动 WiMAX（IEEE 802.16e）、WiMAX-m（IEEE 802.16m）、3GPP-LTE 和 LTE-advanced 标准的物理层特性在表 2-11 中列出。WiMAX-m 的下行峰值数据率预计将超过 350Mbps，而 LTE advanced 将达到 1Gbps。WiMAX 和 LTE-Advanced 蜂窝都期望具有相同的峰窝大小。对于，宏化峰窝大小从 1～5 公里，微峰窝大小为 30 公里时，性能也还算不错。如果峰窝大小高达 100 公里，系统仍然应该具有可接受的性能。WiMAX-m 的移动性与 LTE-advanced 非常相似，支持大约 350～500 公里/小时。对于 WiMAX-m，下行链路的频谱效率大于 17.5bps/Hz，而对于上行链路，超过 10 bps/Hz。LTE-advanced

表 2-11 标准 WiMAX（IEEE802.16e）、WiMAX-m（IEEE802.16m）、3GPP-LTE 和 LTE-advanced 的物理层属性

特征	移动 WiMAX（3G）（IEEE 802.16e）	WiMAX-m（4G）（IEEE 802.16m）	3GPP-LTE（pre-4G）（E-UTRAN）	LTE-adv（4G）
多路方式	上线 MAN-OFDMA	上线 MAN-OFDMA	DL：OFDMA UL：SC-FDMA	DL：OFDMA UL：SC-FDMA
峰值数据速率（TX×RX）	DL：64Mbps（2×2） UL：28Mbps（2×2 协作 MIMO）（10MHz）	DL：>350Mbps（4×4） UL：>200Mbps（2×4）（20MHz）	DL：100Mbps UL：50Mbps	DL：1Gbps UL：500Mbps
信道带宽	1.25/5/10/20MHz	5/10/20MHz 以及 20 倍（可扩展带宽）	1.25～20MHz	聚合带宽（例，四个 20MHz）
覆盖（典型半径，蜂窝大小）	2～7km	5km（默认）5～30km（在适应半径组织） 30～100km（非优可实现）	1～5km（典型）需达 30km	5km（默认）30km（合理性能），高达 100km（可接受性能）
移动性	高达 60～120km/h	120～350km/h，甚至 500km/h	高达 250km/h	350km/h，甚至达 500km/h
频谱效率（bps/Hz）（TX×RX）	DL：6.4（峰值） UL：2.8（峰值）	DL：>17.5（峰值） UL：>10（峰值）	5 bps/Hz	DL：30（8×8） UL：15（4×4）
MIMO（TX×RX）（天线技术）	DL：2×2 UL：1×N（协作 MU）	DL：2×2 或 2×4 或 4×2 或 4×4 UL：1×2 或 1×4 或 2×2 或 2×4	2×2	DL：2×2 或 4×2 或 4×4 或 8×8 UL：1×2 或 2×4
多址技术	IEEE 802.16e-05	IEEE 802.16e	GSM/GPRS/EDGPRS/UMTS/HSPA	GSM/GPRS/EDGPRS/UMTS/HSPA/LTE

在频谱效率上有更高的要求，对于下行链路为 30bps/Hz，对于上行链路为 15bps/Hz。WiMAX-m 和 LTE-advanced 两者都采用 MIMO 技术，以提高空间利用率。过去的 WiMAX 是 IEEE 802.16e，而 LTE-advanced 的前身是 GSM、GPRS、EGPRS、UMTS、HSPA 和 LTE。

2.6 总结

在本章中，我们学习了物理层的属性和该层所使用的技术，主要是编码和调制方案。已经对流行的线路编码方案，包括 NRZ、RZ、曼彻斯特、AMI、mBNL、MLT 和 BLL，以及对分组编码方案（如 4B/5B 和 8B/10B 等）做了介绍，其中自同步起着重要的作用。我们已经学习了基本的调制方案，包括 ASK、PSK 和 FSK，混合 QAM，以及主高级的技术，包括扩频技术（DSSS、FHSS、CDMA），多载波 OFDM 和 MIMO。在给定的带宽和信噪比下，发送更多位的挑战一直都是例新的驱动力。我们还介绍了基本的多路复用方案，如 TDM、FDM 和 WDM。总之，使用哪种方案取决于传输介质的属性、信道条件、目标位速率。对于有线链路，优先考虑 QAM、WDM 和 OFDM。对于脆弱的无线链路，高级系统目前优先选择 OFDM、MIMO 和智能天线。

为了简单起见，物理层对从链路层来的帧不加区分。因此，来自链路层的帧被转换成原始位流，并发送到物理层以进一步处理。原始位流由线路编码操作并调制成信号，因此信号可以通过具有特定传输介质的物理信道传输。在接收端，信号经历一个逆向的过程，并转换成位流用于在链路层通过某种机制即成帧进行定界。成帧将在第 3 章中讨论。

如果信道容量超过需求，那么多个用户就可以共享一个物理信道。复用技术，如 FDM、WDM、TDM、SS、DSSS、FHSS、OFDM、SM 或 STC 可用于物理层，使多个用户能够访问共享物理信道。相应地，为了访问共享信道，链路层必须提供一个仲裁机制以便优化使用并对信道进行访问。在链路层实现的信道接入机制包括 FDMA、WDMA、TDMA、CDMA、DS-CDMA、FH-CDMA、OFDMA、SDMA 和 STMA。

通过信道传输的信号会易受失真、干扰、噪声和其他信号的影响，尤其通过无线通信信道时更是如此。因为在传输期间很可能会出现错误，接收器必须能够检测到这些错误。为了解决这种问题，链路层可能丢弃、纠正或要求重发损坏的帧。因此，在链路层使用差错控制功能，如校验和和循环冗余校验（CRC）。为了访问信道，链路层必须检查物理信道的可用性以确定它是否是闲置/空闲或忙碌。这是分组模式的信道接入方式。例如，CSMA/CD（带有冲突检测的载波监听多路接入）适用于有线信道，而 CSMA/CA（带有冲突避免的载波侦听多路接入）适用于无线信道。这些将在第 3 章中介绍。

常见陷阱

数据速率、位速率和符号率

数据速率，也称为位速率，定义为单位时间内发送或处理的位数目。数据速率的单位是位/秒或 bps。总的位速率、原始位速率、线路速率或数据信号速率就是每秒经过通信链路传输的位数，包括了数据和协议开销。在数字通信中，一个符号可以代表一位或几位的数据。符号率或位速率，是在数字调制信号或线路编码下每秒更改状态的符号数量。符号速率的单位是符号/秒，或波特。在基带信道中，最大波特率称为奈奎斯特速率，也就是信道带宽的一半。

在计算中和信号处理中的带宽

在计算中的带宽指示数据速率，也称为网络带宽，单位是 bps。信号处理中的带宽可以指基带带宽或通带带宽，具体取决于上下文。基带带宽是基带信号上截止频率。通带带宽是指通带信号的上限和下限截止频率之差。信号处理中的带宽通常是用赫兹（Hz）来测量的。

窄带、宽带、广带、超宽带

窄带：在无线通信中，窄带意味着信道足够窄，在该信道上的频率响应是平坦的，即频率响应值是相似的。频率响应是对信道上的输入信号响应产生的系统输出频谱的测量。在一音频信道中，窄带表示声音占据了很窄范围的频率。

宽带（wideband）：在通信中，宽带用来描述频谱中范围广泛的频率，它与窄带相反。当信道具有

高数据速率时，它就需要使用宽带的带宽。

广带（Broadband）：在电信中，广带（俗称为"宽带"）是由处理比较宽广频率范围的可分为信道的一种信令方法。在数据通信中，它意味着多片数据同时发送以增加传输的有效速率。

超宽带：这是一种用于非常低功耗的短距离并使用了大部分无线电频谱的高带宽通信的无线电技术。

进一步阅读

PHY

流行的计算机网络教科书很少用专门的一章来介绍物理层，并且即使介绍了它们也没有完全涵盖所有主题。对更多细节感兴趣的读者就需要钻研数据通信方面的教科书。Proakis 的书对数字通信进行了全面的介绍。它介绍了研究生课程中的通信理论。Sklar 的书是另一本很好的涵盖多种类型的数字通信，同时将理论和应用结合起来的教科书。正如书名所暗示的，Forouzan 和 Fegan 试图平衡物理层和链路层以及上层网络互联通信的处理。它可以为电气工程专业的学生提供比其他教科书更多的计算机科学特色。同样，在整章中，我们试图给计算机科学专业的学生更多电气工程特色和一些开源特点。由 Charan Langton 管理的 Web 站点 ComplextoReal.com 上提供了有关模拟和数字通信各种主题的在线教程集合。Harry Nyquist 的文章"Certain Topics in Telegraph Transmission Theory"以及 Claud Elwood Shannon 的"A Mathematical Theory of Communication"和"Communication in the Presence of Noise"是现代数字通信的基础。

- J. G. Proakis, *Digital Communications*, McGraw-Hill, 2007.
- B. Forouzan and S. Fegan, *Data Communications and Networking*, McGraw-Hill, 2003.
- C. Langton, "Intuitive Guide to Principles of Communi-cations," http://www.complextoreal.com/tutorial.htm.
- B. Sklar, *Digital Communications*, 2nd edition, Prentice-Hall, 2001.
- H. Nyquist, "Certain Factors Affecting Telegraph Speed," *Bell System Technical Journal*, 1924, and "Certain Topics in Telegraph Transmission Theory," *Transactions of the American Institute of Electrical Engineers*, Vol. 47, pp. 617 – 644, 1928.
- H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *Proceedings of the IEEE*, Vol. 90, No. 2, pp. 280 – 305, 2002. (Reprinted from *Transactions of the AIEE* February, pp. 617 – 644, 1928.)
- C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, pp. 379 – 423, pp. 623 – 656, July & October 1948.
- C. E. Shannon, "Communication in the Presence of Noise," *Proceedings of the IEEE*, Vol. 86, No. 2, 1998. (Reprinted from *Proceedings of the IRE*, Vol. 37, No. 1, pp. 10 – 21, 1949.)

扩频

Lamarr 和 Antheil 共同发明了早期形式的扩频通信技术。1941 年 6 月，他们提交了"secret communication system"思想专利申请，获得美国专利 2292387。这就是以跳频扩频形式诞生的扩频。原意对扩频理论做进一步研究的读者可以参阅 Torrieri 的书。Nayerhaan 的报告介绍了扩频通信的基本概念和应用。

- H. Lamarr and G. Antheil, "Secret Communication System," U. S. Patent 2, 292, 387, Aug. 1942.
- D. Torrieri, *Principles of Spread Spectrum Communication Systems*, Springer, 2004.
- J. D. Nayerhaan, "Spread Spectrum Applications," Oct. 1999, http://sss-mag.com/sstopics.html.

OFDM 正交频分多路复用

沿着扩频，OFDM 已经演变了足够长的时间，其成果终于凝结为几本书。以下书籍介绍了 OFDM

系统的设计问题。Li 和 Stuber 的书对 OFDM 的理论和实践进行了全面的讨论。Chuch 和 Tsai 的书，在介绍 OFDM 接收器设计之前，对数字通信背景进行了简洁而全面的介绍。对于物理 IC 实现的硬件设计问题也进行了介绍。Hanzo 和 Munster 的书深入地介绍了 OFDM，MIMO-OFDM 和 MC-CDMA。

- T. Chuch and P. Tsai, *OFDM Baseband Receiver Design for Wireless Communications*, Wiley, 2007.
- L. Hanzo, M. Münster, B. J. Choi, and T. Keller, *OFDM and MC-CDMA for Broadband Multi-User Communications, WLANs and Broadcasting*, Wiley-IEEE Press, 2003.
- G. Li and G. Stuber, *Orthogonal Frequency Division Multiplexing for Wireless Communications*, Springer, 2006.

MIMO

MIMO 技术当然是一个很新的主题。Oestges 的书为 MIMO 信道的时空处理提出了深入的见解。他将传播、信道建模、信号处理以及时空编码关联起来。Kim 的论文就是关于一个用于 WCDMA/HSDPA 的使用用户调度、空间波束形成和反馈信号来控制系统的多用户 MIMO 系统。Gesbert 的论文讨论了多用户 MIMO 以及关于 MIMO 系统的其他理论。

- C. Oestges and B. Clerckx, *MIMO Wireless Communications: From Real-World Propagation to Space-Time Code Design*, Computers—Academic Press, 2007.
- D. Gesbert and J. Akhtar, "Breaking the Barriers of Shannon Capacity: An Overview of MIMO Wireless Systems," *Telenor's Journal: Telektronikk*, pp. 53 – 64, 2002.
- D. Gesbert, M. Kountouris, R. Heath, C. Chae, and T. Salzer, "From Single User to Multiuser Communications: Shifting the MIMO Paradigm," *IEEE Signal Processing Magazine*, Vol. 24, No. 5, pp. 36 – 46, 2007.
- D. Gesbert, M. Shafi, D. Shiu, P. Smith, A. Naguib, et al., "From Theory to Practice: An Overview of MIMO Space-Time Coded Wireless Systems," *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 3, pp. 281 – 302, Apr. 2003.
- S. Kim, H. Kim, C. Park, and K. Lee, "On the Performance of Multi-user MIMO Systems in WCDMA/HSDPA: Beamforming, Feedback and User Diversity", *IEICE Transactions on Communications*, Vol. E89-B, No. 8, pp. 2161-2169, 2006.

开发环境

在计算机网络中，消息从一个节点经过链路发送到另一个节点，其中信号是在物理层处理的。事实上，某些信号既可以在硬件也可以在软件上处理。Mitola 论文中提出了软件定义无线电，通过在通用处理器上相关软件中的无线电系统用于信号处理的步骤，如调制和解调。GNU 无线电项目为这条路线提供了开放源码的解决方案。

虽然信号处理部分可以在软件中实现，但通信系统仍然需要硬件平台来传输信号。在硬件平台上的组件可以包括 AD/DA 转换器、功率放大器（PA）、混频器、振荡器、锁相环（PLL）和微控制器或微处理器。这些组件既可以是模拟的也可以是数字集成电路的。因此，需要模拟电路设计和数字电路设计工具以便开发用于通信系统的硬件平台。例如，MATLAB 和 Simulink 可用于系统的分析、设计和模拟。Verilog（系统的 Verilog）和 VHDL 有助于设计模拟仿真数字集成电路（IC）。已经开发出从 MATLAB/Simulink 模型到 HDL 模型的自动转换工具，以便加快数字集成电路系统设计。SPICE（强调模拟集成电路的模拟项目）和 Agilent ADS（先进设计系统）是模拟仿真集成电路设计和射频 IC 设计的工具。参考资料如下：

- J. Mitola, "Software Radio Architecture: A Mathematical Perspective," *IEEE Journal on Selected Areas in Communications* Vol. 17, No. 4, pp. 514 – 538, Apr. 1999.
- GNU Radio Project: http://gnuradio.org/redmine/wiki/gnuradio.
- The MathWorks: A Software Provider for Technical Computing and Model-Based Design, http://www.mathworks.com/.
- VASG: Maintaining and Extending the VHDL Standard (IEEE 1076), http://www.eda.org/vasg/.

- IEEE P1800: Standard for System Verilog: Unified Hardware Design, Specification and Verification Language, http://www.eda.org/sv-ieee1800/.
- SPICE: A General-Purpose Open Source Analog Electronic Circuit Simulator, http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.
- Agilent Technologies Advanced Design System (ADS) 2009: A High-frequency/High-speed Platform for Co-design of Integrated Circuits (IC), Packages, Modules and Boards, http://www.home.agilent.com/.

常见问题解答

1. 什么是位速率和波特率？

答：位速率（或数据速率）：单位时间传输的位数。

波特率（或符号率）：单位时间内传输的符号数。

2. 采样定理，奈奎斯特定理和香农定理之间的区别是什么？

答：采样定理：计算在多大采样率下，信号可以唯一地重构。

奈奎斯特定理：计算无噪声下的最大数据速率。

香农定理：计算噪声信道下的最大数据传输率。

3. 在数字通信中，经常使用什么样的信号，为什么呢？

答：在数字通信中，通常使用周期性模拟信号和非周期性数字信号，因为前者需求较少的带宽而后者用来表示数字数据。

4. 与模拟信号相比，数字信号有哪些优点？

答：数字信号：对噪声具有更大的免疫性，并且当信号传输通过传输介质时更容易恢复信号。

模拟信号：易于被噪声和干扰破坏，更难以完全恢复。

5. 为什么在物理层需要线路编码？

答：线路编码可以防止基线漂移和引入直流分量，并可以启用自同步，提供错误检测和纠正，增加信号对噪声和干扰的免疫力。

6. 哪些因素可能会影响物理层的传输能力，尤其是在无线信道中？

答：衰减、衰落、失真畸变、干扰和噪声。

7. 什么是星座图？

答：它是一种用在模拟信号和其相应的数字数据模式之间映射的工具。

8. 什么是数字通信中的基本调制？

答：ASK、FSK 和 PSK 是数字通信中的三种最基本的调制。

ASK 调制：用不同的载波振幅表示数字数据。

FSK 调制：用不同的载波频率表示数字数据。

PSK 调制：用载波的相位，而不是相位的变化，表示数字数据。

QAM 调制：将 ASK 和 PSK 结合起来改变载波振幅和相位，以形成不同信号元素的波形。

9. 在数字通信中，为什么通过高频信道传输的信号必须进行调制？

答：如果是带数字信号（带有较低频率）想要通过高频信道，它就必须要由正弦载波承载。换句话来说，信号必须调制到高频载波上，以便数据信号能够传输通过信道。

10. 为什么要多路复用？

答：当信道的带宽超过数据流所需要的带宽时，信道就可以由多个用户共享，以便提高信道的利用率。

11. 扩展有哪些优点？

答：扩展后噪声似的信号就很难检测和干扰。额外的冗余可用来减少对无线通信的窃听、干扰、噪声等脆弱性。

12. OFDM 技术的主要特点是什么？为什么 4G 选用 OFDM 而不是 CDMA？

答：OFDM 的主要特征：子载波的正交性，允许数据在子信道上同时传输。

OFDM 技术的优点：

1）许多设备用，可能和多载波组合起来。

2）比 CDMA 更高的速率（一系列技术）。

13. 比较传统的无线电系统与软件无线电系统。

答：软件无线电尽可能多地在软件中实现，用于信号处理的系统电功能，而不是用于传统无线电系统中的专用电路。此外，在软件无线电系统中的DSP芯片可调动实现由软件驱动执行，而不是仅仅设备实现。

软件无线电的优点：

1）不断的升级性，更加灵活性，特别是具有可重配置的硬件以便支持具有更高级别的信号处理。

2）降低了切换到其他标准的成本和复杂性，加入市场的时间。

练习

动手练习

1. 在 www.opencores.org 中查找并总结与网络相关的模块，并加入到表中，注意区别。比如它们的协议层，用什么语言和关键实现算法或控制。

2. 在 www.opencores.org 找出 PHY（物理）层模块，对于每个模块，试述它是为何实现的，以及如何。算法及控制上的哪一部分被执行及考虑，哪部分没被执行。把你自己对于这个所研究的算法或协议总结起来。

3. GNU Radio 是一套软件无线电系统软件包。在安装了 Linux 操作系统的机器上构建 GNU 无线电系统。

a. 从 http://gnuradio.org/redmine/wiki/gnuradio/Download 下载最新稳定版本的 GNU Radio。

b. 参照从 http://gnuradio.org/ ... /Ubuntu 或 redmine/wiki/gnuradio/BuildGuide 下载的说明，按照其中的指示，建立一个 GNU 无线电系统。

c. 为 GNU Radio 安装依赖项，参照在 GNU Radio 第 3 章上的讨论。

d. 许多软件无线电的例子位于文件夹/usr/share/gnuradio/examples 内。运行示例 ... /gnuradio/examples/audio/dial_tone.py，这个示例实际一个用任何编程语言（如 C++、Java 或 Python）实写的“Hello World”例子。尝试运行生成的例子（提示：GNU 的 Radio 软件包已经集在 Fedora 库中，很容易利用 yum 或 rpm 工具来实现这个软件包）。

4. 在机器上安装从 http://www.joshknows.com/grcRC 下载的 GRC 工具（GNU Radio 手册）。GRC 有助于 GNU Radio 的研究学习。现在利用 GRC 的工具来设计如下的系统：

a. 可以过滤信道的一个系统。

b. QAM 调制器/解调器系统。

（提示：你可以参考由 Naveen Mamidala 所开的“GNU Radio Toolbox”。）

书面练习

1. 为什么数字信号通常表示为非周期的数字信号？为什么其模拟信号表示为非周期性的模拟信号？

2. 比较表示下列信号需要的频率数量和带宽大小：

a. 周期性模拟

b. 非周期性模拟

c. 周期性数字

d. 非周期性数字

3. 采样和量化之间的区别是什么？

4. 码元和二进之间的区别是什么？

5. 什么是 sdr（信号数值化）和 SNR（信噪比）？它们如何门下评估？

6. 分别在传线传输、反射、衍射和折射中比较点和信号和脉冲信号的功能。

7. 在单极性 NRZ-L、双性 NRZ-L、NRZ-I 和 BZ、短脚脚等，差分曼彻斯特码，AMI 和 MLT-3 中，哪种方案分别与同步、其线编码、自动检查元关？

8. 使用单极性 NRZ-L、双立 NRZ-L、NRZ-I 和 BZ 分别绘制以下数据流的波形。计算 sdr 此和平均波特率

a. 1010101010101
b. 111111000000
c. 111000111000
d. 000000000000
e. 111111111111

9. 使用曼彻斯特和差分曼彻斯特方案绘制以下数据流的波形。计算 sdr 值和半周波特率。
   a. 1010101010101
   b. 111111000000
   c. 111000111000
   d. 000000000000
   e. 111111111111

10. 使用 MLT-3 方案绘制以下数据流的波形。计算 sdr 值和半周波特率。
    a. 1010101010101
    b. 111111000000
    c. 111000111000
    d. 000000000000
    e. 111111111111

11. 给定数据流的位速率为 1Mbps，2Mbps 或 54Mbps，计算使用 8ASK、BPSK、QPSK、16-PSK、4-QAM 的 BFSK 16-QAM 和 64-QAM 调制的波特率。

12. 给定波特率为 8kBd 和 64kBd，计算 BFSK、BASK、BPSK、QPSK、16-PSK、4-QAM、16-QAM 和 64-QAM 调制的位速率。

13. 给定位速率 56kbps 或 256kbps 的数据流，如果将 11 位或 13 位巴克码用做 PN 码来扩展数据流，那么码片速度和处理增益为多少？

14. 同步 CDMA 和异步 CDMA 之间主要的区别是什么？

15. 比较 CDMA 中使用的 PN 码和正交码。为什么使用 PN 码比正交码能够支持更多的用户？

16. 我们如何才能知道在异步 CDMA 中使用的两个 PN 码是相关的还是不相关的？

17. 我们如何知道在同步 CDMA 中使用的两个码是否正交？

18. 解释为什么扩频可以减轻环境噪声并能去掉来自邻近用户的窄带或宽带噪声。为什么它能够提供更好的隐私保护？

19. 在 FHSS 中，两个发射站有可能在同一时间跳到相同的子信道上，即产生碰撞吗？证明你的答案。

20. 在 OFDM 中，用以实现多载波机制的主要硬件组件是什么？一个数据流如何利用多个载波并通过 OFDM 信道？

21. 在 OFDM 中，两个信号相互正交的标准是什么？

22. 具有或不具有信道状态信息的情况下，MIMO 系统的优点和缺点是什么？

23. 单用户 MIMO 和多用户 MIMO 之间的主要区别是什么？

链路层

通过物理链路从一个节点到另外一个节点有效地传输数据不只是将位编码或调制为信号这么简单。为了成功地进行数据传输，首先必须解决几个问题。举例来说，相邻链路对之间的电磁噪声可能会外加在信号传输信号并导致错误，因此为了可靠地传输数据，链路层需要合适的错误检测机制。发射器也可能以比接收器能够处理的速度更快的速率进行传输，如果这种情况发生，接收器就必须通知数据包源起源在位置让发射机放慢速度。如果多个节点共享一个局域网，就需要有一种补充机制来决定何时可以传输。除了所有这些情况外，我们需要互联局域网，即我们需要桥接不同的局域网以扩展数据包的转发已不再局限在单个局域网内。虽然这些问题需要由物理链路以上的一些功能来解决，但在OSI体系结构中的链路层为上层功能管理者物理链路，从而使它们能够免除控制物理层链路的繁重工作。链路层极大地减轻了上层协议的设计，并使之几乎能够独立于物理传输特性。

在本章中，我们介绍：1）链路层提供的功能或服务；2）流行的实际链路层协议；3）一组选择的链路层技术的开源软件和硬件实现集合。3.1节讨论了在设计链路层功能时考虑的一般问题，包括成帧、寻址、转译控制、流量控制、访问控制，以及与其他层的接口。我们说明了网络适配器的接口和数据包流，以及Linux中IP上层的套接口，作为1.5节中数据包的一个示例。

现实世界中各种链路技术总结在表3-1中，在本章几乎不可能对所有的技术都进行描述。这里我们仅专注于少数主流链路技术，我们将详细介绍：1）点到点协议，或写作PPP，及其开源实现将在3.2节中介绍；2）有线、静态路由协议，以太网，以其Verilog硬件实现将在3.3节中介绍；3）无线广播链路协议，无线局域网，将在3.4节中介绍，还包括对蓝牙和WiMAX的总结。我们选择这些例子，是由于其普及性。PPP是流行的“最后一公里”的发号服务或路由器，在点到点链路上承载各种网络协议。以太网在有线局域网技术中占据主导地位，同时也在取代为主要技术的无线局域网和广域网。与通常使用有线链路连接到网络的桌面计算机相比，用户移动设备（如笔记本计算机和手机等）更需要无线连接

表3-1 链路协议

	个域网/局域网（PAN/LAN）	城域网/广域网（MAN/WAN）
过时的或遗留的	令牌总线（802.4） 令牌环（802.5） HIPPI 记行通讯 同步（802.9） 需求优先（802.12） 记行分布式数据接口（FDDI） 异步传输模式（ATM） HIPERLAN	DQDB（802.6） HDLC X.25 帧中继 SMDS ISDN U-ISDN
主流的或兴旺活跃的	以太网（802.3） 无线局域网（802.11） 蓝牙（802.15） 龙门通讯 HomeRF HomePlug	以太网（802.3） 点到点协议（PPP） HDCSDS xDSL SONET 蜂窝（3G，EDL，WiMAX（802.16）） 弹性分组环（802.17） 异步转移模式

技术，即无线局域网、蓝牙和WiMAX。既然多个局域网能够通过桥接互连起来，我们将在3.5节中介绍这项技术及其两个关键组件：自学习和生成树的开源实现。最后，在3.6节中说明Linux设备驱动程序的一般概念，然后深入学习以太网驱动程序实现细节。

3.1 一般问题

位于物理链路和网络层之间的是链路层，它能够提供对物理通信的控制并提供对上面网络层的服务。该层执行以下主要功能。

成帧：将物理链路上传输的数据打包成以帧为单位。一个帧包含两个主要部分：在头部中的控制信息以及在有效载荷中的数据。控制信息，如目的地址、使用的上层协议、错误检测码等，对于帧处理非常关键。从上层来的数据部分与控制信息一起再采到帧中。由于帧在物理层上是以原始位流的形式传输，所以链路层服务应该将帧转换为位流来传输，而在接收时将位流分成帧。大部分文献中交替使用两个术语数据包和帧，但我们将链路层中的数据单元称为帧。

寻址：当我们给朋友写一封信时需要给出地址，当我们给朋友打电话时也需要一个电话号码。出于同样原因，在链路层上也需要寻址。一个链路层地址，经常以一定长度的数字形式给出，用以指定主机的身份。当主机A想要向主机B发送一个帧时，它就会包含自己的地址和主机B的地址分别作为帧的控制信息中的源地址和目的地地址。

差错控制和可靠性：通过物理介质传输的帧很容易出错，因此接收器就必须通过某种机制检测出这些错误。当检测到错误时，接收器就可以直接丢弃帧，或者它也可能会得知出现了错误并要求发射机重发帧。对于以太网的数据链路技术，误码率极低，所以为了效率起见会将重传机制留给高层协议（如TCP）来完成。对于802.11的无线连接技术，发射机会等待接收机的确认一段时间，如果超时仍没有收到确认，发射机将重传最后一帧，以便确保能够及时重传。

流量控制：发射机可能以超过接收机能接收的速度发送。在这种情况下，接收机将丢弃过多的帧而使发射机重传丢弃的帧，但这样做会浪费链路容量。流量控制提供了一种机制，使得接收机能让发送机放慢发送速度，以避免接收机因为发射机发送的数据过多而缓载。

介质访问控制：当有多台主机要通过共享介质传输数据时，就必须有一个仲裁机制来决定谁先发送。一个良好的仲裁机制必须能让主机公平地访问共享介质，同时保持共享介质高度可用以防止多台主机积压数据，即数据排队等待传输。

3.1.1 成帧

由于数据在物理层是以原始位流的形式传输，所以当接收位流时链路层就必须能够确定每一帧的开始和结束。另一方面，它也必须将帧变成原始位流以便于物理传输。此功能称为成帧。

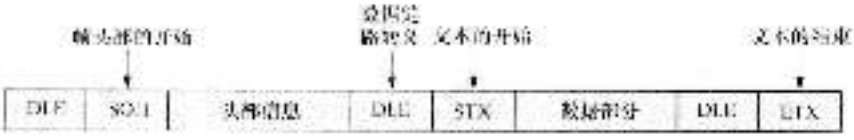
帧定界

有多种方法可以用来定界帧。特殊的位模式或前哨（sentinel）字符可以用来标记帧边界，例如将在稍后介绍的HDLC帧。某些以太网系统采用特殊的曼彻斯特编码标记帧的边界，而另一些则通过是否存在信号来标识边界[1]。快速以太网（即100Mbps）既生以前都是利用前者，因为它可以检测物理链路状态。后者则不能检测，因为它不能告诉物理链路是否断开或者是否没有帧传输（在这两种情况下，在链路上都无信号）。它曾经用在10Mbps以太网上，但在新的以太网技术中已经不再使用。

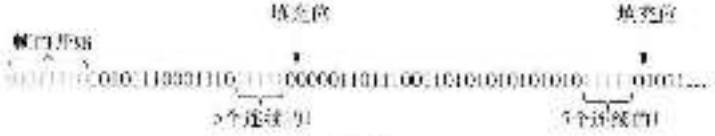
一个帧可以是面向位的，也可以是面向字节的，具体要根据其基本单元。面向位的成帧协议可以指定一种特殊的位模式，比如HDLC中的01111110，用以标记帧的开始和结束。面面向字节的成帧协议可以指定特殊字符，比如SOH（头部开始）和STX（文本开始），以标记帧头部和数据的开始。当正常的数据字符或位显示成与特殊的模式相同时，由于可能存在歧义，就需要使用一种称为字节或位填充的技术解决歧义，如图3-1所示。在面向字节的帧中，一种特殊的转义字符，又称为数据链路转义（DLE），前面的一个特殊字符指示下一个字符是普通正常数据。因为DLE本身也是一个特殊的字

[1] 以太网使用的术语"流"指的是帧的物理封装。严格地说，特殊的编码或存在特殊的信号是为流而不是成定界的。但是，这里我们不介绍细节问题。

符，所以两个连续的 DLE 代表一个普通正常的 DLE 字符。在 HDLC 中，每 5 个连续的 1 序列后插入一个二进制 0，因此模式 01111110 从不会在普通正常数据中出现。发射器和接收器两者都遵循相同的规则来解决歧义。



a) 字节填充



b) 位填充

图 3-1

以太网采用了不同的成帧方式。例如，100BASE-X 使用特殊的编码来标识边界。在第 2 章所述的 4B/5B 编码中，在 32（2^5）种可能的编码中只有 16 种来自实际的数据，而其余的用作控制代码。这些控制代码可以唯一地让接收器所识别。这样就用于将一个帧从一个位流中分隔（定界）出来。另一种以太网系统 10BASE-T，帧边界的识别就是根据是否存在某个信号来确定的。

帧格式

帧头部包含控制信息，帧数据包含链路层或网络层的数据。后者还包含控制信息和来自更高层的数据。在帧头部中的典型控制信息包括以下字段：

地址：这通常既表示源地址也表示目的地址。如果帧头部中的目的地址与它自己的地址匹配，接收器就知道帧是发给自己的。接收器还可以通过利用到达帧的源地址填写外出帧的目的地址来对到达帧的源信息回应。

长度：可以指示整个帧的长度，或者数据的长度。

类型：网络层标识的类型在该字段进行编码。链路层协议可以读取代码，以确定它是哪个网络层模块，如互联网协议（IP），被调用来进一步处理数据字段。

错误检测代码：这是一个数学函数的值，以帧的内容作为输入参数。发射机计算函数并将结果值放入帧中。收到帧后，接收机以同样的方式计算函数，比较结果是否与帧中嵌入的值相匹配。如果不匹配，就意味着内容在传输过程中已经改变了。

3.1.2 寻址

在通信中，地址是一种能将某台主机与其他主机区分开来的标识符。虽然名字更容易记住，但在低层协议中数值地址是一种更紧凑的表示。使用名字作为主机名的概念在第 6 章中介绍（详见数据系统）。

全球或本地地址

地址既可以是全球唯一的也可以是本地唯一的。全球唯一的地址是全世界独一无二的，而本地唯一的地址仅限于在本地站点是唯一的。一般情况下，一个本地唯一的地址占用较少的位，但需要管理员的更多工作，以确保本地的唯一性。由于地址中少量位数的开销是极不是道的，目前首选全球唯一的地址，因此管理员可以随意将主机添加到网络而不需要担心有关本地地址的冲突问题。

地址长度

地址应该有多长？长的地址需要发送更多的位，也很难记住或引用，但短的地址可能不足以确保

全球的唯一性。对于一台本地唯一的地址，8 位或 16 位应该是够了，但为了支持全球唯一地址就需要更长的地址。IEEE 802 中一个非常流行的地址格式是 48 位。我们留给读者一个练习，确定这个长度是否足够全球使用。

IEEE 802 MAC 地址

IEEE 802 标准为适配地址格式选择了很好的例子，因为它们广泛地被许多链路协议所采用，包括以太网、光纤分布式数据接口（FDDI）和无线局域网。尽管 IEEE 802 规定既可以使用 2 字节也可以使用 6 字节的地址，但大多数实现采用 6 字节（48 位）的地址格式。为了确保其在全球的唯一性，地址分成两部分：组织唯一标识符（OUI）和适配自己分配的部分，每部分各占 3 个字节。IEEE 管理前一部分，因此公司组织可以从 IEEE 申请一个 OUI，接下来由负责保持自己 OUI 的组织分配部分的唯一性。从理论上讲，按照 IEEE 802 规范，有 2^{24}（大约 10^{7}）个地址可以分配，而这个数目足以确保全球唯一性。一个 IEEE 802 地址常常用十六进制表示，每两个数字利用一次冒号分开，例如 00:32:4F:CC:30:58。图 3-2 说明了 IEEE 802 地址的格式。



图 3-2 IEEE 802 地址的格式

在传输顺序中的第一位保留用于指示地址是单播还是组播[○]。单播地址是一台主机，而组播地址是一组主机地址。组播的一个特殊例子就是广播，其地址中的所有位都是 1。广播是被发向所有主机，只要它在链路层上可达。请注意，地址中每个字节的位传输顺序与它们在内存中存储的顺序不同。在以太网中，传输顺序是每个字节中最低有效位（LSB）先发送，称为低字节序（little-endian）。例如，给定一个字节 $b_7b_6\cdots b_0$，以太网首先传输 b_0，然后传输 b_1，b_2 等。在其他协议中，如 FDDI 和令牌环中，传输顺序是每个字节中最高位（MSB）先发送，称为高字节序（big-endian）。

3.1.3 差错控制和可靠性

帧在传输过程中容易出现错误，接收链路层的设备能够及时发现这些错误。如 3.1.1 节中所述，错误检测码是如何产生的函数，由发射机计算后来填充到帧的一个字段中。接收器将使用相同的算法对所接收到的帧内容重新计算错误检测代码，并对比这两个代码值是否匹配。如果不匹配，在传输过程中必然发生了错误。下面我们介绍常用的两种错误检测函数：校验和与循环冗余校验（CRC）。

错误检测码

校验和计算就是将帧内容分成 m 位的多个块（分组），并取这些块的 m 位的和。计算本身很简单，可以很容易地在软件中实现。在开发实现 3.1 中，我们将介绍实现校验和计算的代码。

另一个流行的技术是循环冗余校验，这比校验和复杂。但容易在硬件中实现。假设帧内容有 m 位。发射机可以生成一个 k 位序列作为链接序列（FCS），以便 $m+k$ 位的帧可以被预定的位模式（又称为生成多项式）除。接收机以同样的方式除以接收到的帧，看余数是否为零。如果余数为非 0，那么在传输过程中就有错误。下面的例子演示了一个生成 FCS 的简单 CRC 程序。

帧内容 = 10101001110（11 位）

○ 有关 OUI 如何分配的信息，参见 http://standards.ieee.org/regauth/oui/oui.txt。

○ 第二位可以用来表示地址是本地唯一地址还是本地唯一地址。但是，这种用法很罕见，所以这里我们忽略不讲。

生成多项式 B = 101011（6 位）

FCS =（5 位）

步骤如下进行：

第 1 步 将 F 向左移 5 位（利用二进制的模 2 运算进行 2⁵ 乘 F 的运算，这相当于在 F 后面添加 5 个 0）并添加 5 个 0，得到 11010001110000000。

第 2 步 将第 1 步得到的结果模式除以 B，过程如下：（计算全部由模 2 算术运算）



第 3 步 将在此计算中得到的余数添加到原来帧的内容中，得到 11010001110**10001**。然后传送帧。接收机将接收到的帧内容除以相同的生成多项式，以验证帧。在接收机上的验证留给读者做练习。

上面的描述是经过简化的。因为实际 CRC 计算之后的处理在数学上很复杂。已经得到证明，CRC 可以检测到很多类型的错误，包括：

1）单位错误。

2）双位错误。

3）位数长度小于 FCS 的突发错误。

CRC 计算可以很容易地在硬件中应用异或门和移位寄存器实现。假设我们以形式 $a_n a_{n-1} a_{n-2} \cdots a_0$ 来表示生成多项式，这里 a_n 和 a_0 必须为 1。我们绘制出常用来实现 CRC 计算的电路体系结构，如图 3-3 所示。帧的内容逐位移入到该电路中，在移位寄存器中的最后位模式是 FCS，即 $C_{n-1} C_{n-2} \cdots C_1 C_0$。$C_{n-1}$ $C_{n-2} \cdots C_1 C_0$ 的初始值并不重要，因为一旦开始计算就将它们移出。对于非常高速的链路，则采用并行 CRC 计算电路，以满足高速的需求。

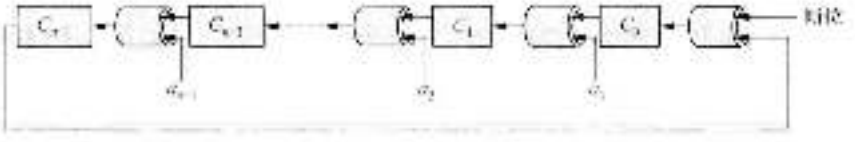


图 3-3 CRC 电路图

数据可靠性

但当接收到一个错误的帧时接收机将如何回应？接收机可以用以下方式响应：

1）默默地丢弃不正确的传入帧。

2）当传入帧是正确的时，回复对正确的确认。

3）当传入帧是不正确的时，回复对错误的确认。

发射机会重新发送接收错误的帧，或者只是忽略错误。在后一种情况下，高层协议，如 TCP，将负责重传。

另一项需要做出的决策是，是否在链路层执行数据确认。因为以太网的位错误率相当低，所以它不实用重认机制，如果要对每个传输帧都进行重认，就有点矫枉过正。因此，将确认机制留给高层协议（如 TCP）处理。对于无线链路，误码率比以太网高得多（即不可靠），因此最好确认每个传输的帧。但是，使用确认机制，却要以降低吞吐量为代价，因为发送者在下次发送前必须等待确认。在链路层设计中，仍需要在高吞吐量和及时检测错误之间做出权衡。

行动原则：CRC还是校验和

在更高层协议（如TCP、UDP和IP）中使用校验和，而CRC只用于以太网和无线局域网中。这种区别的背后有两个原因。首先，CRC很容易在硬件中实现，而不是在软件中。因为更高层协议几乎总是在软件中实现，为它们使用校验和也是一个自然选择。其次，CRC在数学上已经得到证明，对于多种物理传输错误是健壮的。由于CRC已经过选择了大部分传输错误，所以使用校验和重核不寻常的错误（例如，发生在网络设备内的错误）应该已经足够了。

行动原则：纠错码

像CRC和校验和的错误检测码只能检测传输错误。一旦发现错误，接收器无法做任何事情，只能放弃帧。另一种替代方法就是使用纠错码的前向纠错（FEC）。通过FEC，发送方在消息上添加更多的冗余位。纠错与错误检测之间的主要区别在于，前者可能准确出出现错误的位并予以纠正。帧中错误位被"纠正"后，就能够被接收而不需要重传。这里我们不会深入研究详细的数学知识，而是指出一般原则：要纠正更多的位就需要使用更多的冗余位。因此便会出现一个问题：为了纠错，添加更多的冗余位值得吗？

答案取决于误码率、可能的传输方向、数据的重要性等。在常见的数据链路协议（如以太网）中，误码率相当低。在以太网中，每传输10^{12}位数据才会出现1位错误。在这种情况下，使用纠错码显然是碰到过了，即使在无线局域网中，错误检测也是足够的。自数据在互联网上传输时，大多数错误是由于互联网拥塞而丢数据包，因此这里使用纠错码仍然帮助不大。

纠错码的常见应用是在太空通信、数据存储和卫星广播。在太空通信中，重传的整用很高，所以就值得使用纠错码。由于卫星广播是单向的，没有确认或重传；所以就需要纠错。在数据存储中，如果出现错误，错误检测就帮助不大，因为数据存储是唯一的数据源计且错误无论如何都不能恢复。在这种情况下，纠错码至少可以在一定程度上恢复位错误。在卫星广播中，因为接收机没有办法通知错误码的源，所以就首选纠错。

开源实现3.1：校验和

概述

校验和计算是一种用于互联网协议（如IP、UDP和TCP）中的常见错误检测码。其效率对于良好的路由性能至关重要，因为每个数据包需要在其网络层头部和传输层头部中计算校验和。例如，在TCP头部中的校验和字段既包括TCP分段头部和有效载荷的内容，也包括伪头部的额外信息，如源和目的地IP地址。如果在TCP协议栈中的校验和计算得不到很好的实施，那么在数据包转发过程中它将消耗相当数量的CPU周期。

框图

图3-4是一个说明如何实施校验和的框图。在开始时，变量sum和checksum初始化为0，并保持等批16位字的输入字节的覆盖范围更新一个数据包。经过最后的批量计算后，sum的值交叉折叠（见下面的讨论）以便导出checksum的值。下面详述校验和计算的Linux实现。



图3-4 校验和计算框图

数据结构

校验和计算的数据结构很不足道。它包含一个累积了整个导段和负载的16位字sum变量，以及一个用来计算还剩下多少16位字的count变量。注意sum变量是一个32位字以便捕获累积的溢出。计算过最后的16位字后，sum变量就折叠成一个16位字，checksum的值是折叠值的1的补码。

算法实现

对于将要包括在校验和计算中的连续字节，相邻的字节首先配对用成16位字，然后计算出这些配对的1的补码和。如果留下一个字节没有配对，就将它直接加到校验和中。最后，结果的1的补码填充到校验和字段。接收机遵循相同的步骤针对相同字节计算校验和字段。如果结果是全1，那么检查成功。为了提高效率Linux的校验和实现通常采用汇编语言编写，我们这里给出在RFC 1071中的C代码，以便有更好的可读性。第4章中的开源实现4.3中解释了在Linux内核中的汇编语言版本的IP校验和计算。

```
/* Compute Internet Checksum for "count" bytes
*        beginning at location "addr".
*/
register long sum = 0;
while( count > 1 ) {
        sum += * (unsigned short) addr++;
        count -= 2;
}
/* Add left-over byte, if any */
if( count > 0 )
        sum += * (unsigned char *) addr;
/* Fold 32-bit sum to 16 bits */
while (sum>>16)
        sum = (sum & 0xffff) + (sum >> 16);
checksum = ~sum;
```

练习

1. 当IP数据分组通过一台路由器时，IP数据分组中的TTL字段会减1，那么经过减法后的校验值也必须相应变改。请寻找一种有效的算法重新计算新的校验和的值（提示：请参阅RFC 1071和1141文档）。

2. 解释为什么IP校验和中没有将有效服务包括在其计算中。

开源实现3.2：硬件CRC－32

概述

CRC－32是常用于很多MAC协议（包括以太网和802.11无线局域网）中的一种错误检测码。为了高速计算，CRC－32通常是作为在网络接口卡中芯片功能的一部分在硬件中实现的。作为分批的4位数据输入或输出到物理链路上，它们是按照顺序处理的，以获得32位CRC值。计算结果既可以用来验证帧的正确性也可以添加到要发送的帧上。

框图

图3-5是表示CRC如何在硬件中实现的框图。最初是将一个32个1分配给crc变量。当每批4位被送入时，这些位会将当前的crc变量更新为crc next，即为crc或作用于下一批4位数据。更新处理涉及带有许多参数的计算，所以我们这里省略了具体细节。当所有数据都处理后，存储在crc变量中的值就是改后的结果。

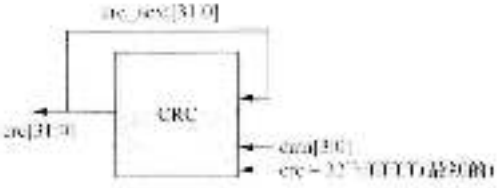


图3.5 CRC－32计算的框图

数据结构

CRC－32计算的数据结构主要是读取每一批4位数据后保持最新状态的32位crc变量。CRC－32计算的最终结果是读取完改后一组数据之后的状态。

算法实现

CRC－32的开源实现可以在OpenCores Web网站（http://www.opencores.org）中的以太网MAC

项目中找到。参见项目CVS库中的Verilog实现eth_crc.v。在这个实现中，每批4位，数据依次送入CRC模块中。最初，所有CRC值初始化为1。当前CRC值的每一位是通过将从输入的4位中选择的位与上一轮CRC值生行并或运算获得的。由于计算的复杂性，我们建议读者参考eth_crc.v，学习每一个位计算中的相关行节。数据位计算完成后，也同时导出了CRC值。接收机遵循同样的过程计算CRC值并检查到达域的正确性。

练习

1. crc_crc.v中的算法能够很容易在软件中实现吗？请解释你的答案。
2. 为什么我们在链路层中使用CRC-32而不是校验和计算？

3.1.4 流量控制

流量控制用以解决快速发射机和慢速接收机的问题。它提供了一种方法，允许一台被数据分组淹没的接收机告诉发送机放慢传输速率。最简单的一种流量控制方法是停止—等待，其中发射机发送一个帧，等待接收机的确认，然后才会再传送下一个帧。然而，这种方法会使链路的利用率很低。接下来将介绍更好的实现方法。

滑动窗口协议

更有效的流量控制可以通过滑动窗口协议来实现，其中发射机在没有得到确认消息的情况下能够传送一定数量的帧。当有确认信息从接收机传回后，发射机就向前移动窗口以便传输更多的帧。为了能够跟踪返回确认所对应的发送出去的帧，每一帧都附加上了一个序列号。为了防止一个号同时被多个帧使用，序列号范围就必须足够大；否则，我们无法知道这个序列号代表的是老帧还是一个新帧。

图3-6说明了滑动窗口的例子。假设发射机的窗口大小为9，就是说发射机在无确认的情况下最多可传输9个帧，从1号帧一直到9号帧。假设发射机已经传输了4个帧（如图3-6a所示）并且收到前三个帧已成功收到的确认。窗口向前滑动3个帧表示目前8个帧（5号帧到12号帧）在无确认的情况下可传输（如图3-6b所示）。窗口最初包括1号帧到9号帧，现在包括4号帧到12号帧，感觉就好像窗口沿着帧的顺序方向滑动。滑动窗口流量控制在TCP（传输控制协议）中是一项非常重要的技术，这是一种优秀而且非常实用的采用滑动窗口的例子。我们将在第4章介绍滑动窗口在TCP中的应用。

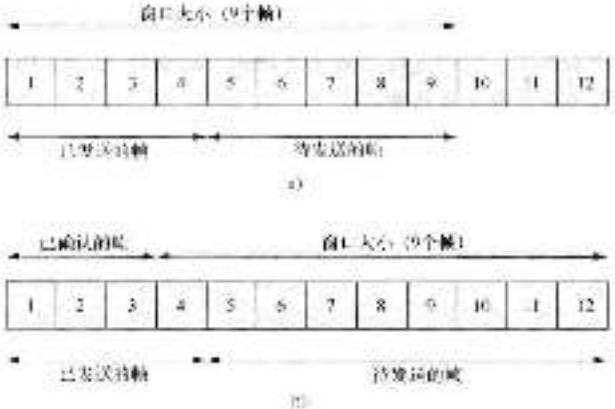


图3-6 通过滑动窗口传送帧

其他方法

实现流量控制还有很多方法。比如，在以太网中使用的机制包括音乐机制和PAUSE（暂停）帧。然而，为了理解这些方法就需要首先知道这些协议是如何工作的。我们将这些流量控制技术留在3.3.2节中讲解。

3.1.5 介质访问控制

当多个节点共享同一个物理介质时就需要用到介质访问控制（MAC）。为了能够公平，高效地共

享，它使用一种每一个节点都应该遵守的仲裁机制。我们把该技术分为两类。

基于争用的方法

通过这种方法，多个节点竞争共享介质的使用。ALOHA 就是这种方法中的一个经典例子，其中节点会随意传输数据。如果两个或更多个节点同时传输数据，那么就会发生冲突，它们传输的帧将会出现错误，从而降低吞吐量性能。改进的方法是将时隙 ALOHA，其中一个节点仅允许在它自己的时隙内开始时传输。更进一步的改进方法包括载波侦听和冲突检测。载波侦听就是节点侦听是否有信号正在共享介质上（在一种称为载波的信号上）传输。发射机将一直耐心等待直到共享介质空闲为止。通过一旦检测到冲突就停止传输的方法，冲突检测能够缩短减少错误的位置。

无争用方法

当不能及时检测到冲突时，以争用为基础的方法就会变得无效。在传输终止时，作为一个完整的帧可能已经出现错误。两种常用的无争用方法分别为时间片轮转法和基于预留的方法。在前一种方法中，令牌将在节点中一个接一个地传递，从而公平地共享介质，并且只有获得令牌的节点才有权传输自己的帧。经典例子包括：令牌环和光纤分布式数据接口（FDDI），尽管它们拥有不同的结构，但是机制相似。基于预留的方法是在真正传输帧之前尽量在共享介质上预留一个信道。一个众所周知的例子是 IEEE 802.11 无线局域网中的 RTS/CTS（请求发送/允许发送）机制。我们将在 3.4 节中详细介绍它。由于这个过程本身会带来额外的开销，所以使用预留方法时就要进行折中。如果一个帧的丢失无关紧要，如一个很短的帧，那么在这种情况下基于争用的方法更好一些。如果在一条点对点链路上仅有两个节点，而且是全双工链路时，那么访问控制可能就是不必要的。我们将在 3.2 节中进一步讨论全双工的运行方式。

3.1.6 桥接

将单独的局域网连接成互连的网络就可以扩大网络的通信范围。在链路层上的连接设备称为 MAC 网桥，简称网桥，它能将局域网互联起来，使它们的节点就像在同一个局域网上一样。网桥知道是否应该转发一个到达的数据帧以及转发到那个接口上。为了支持即插即用和便于管理，网桥能够自动学习接口属于哪一个目的地主机。

当桥接网络的拓扑变得越来越大时，网络管理员会不经意地在拓扑上形成一个环路。IEEE 802.1D，或者 IEEE MAC 网桥标准，制定了生成树协议（STP）用来去除桥接网络中的环路。还有一些其他的重要内容，比如要缓分割局域网，将多条链路结合起来成为主干网络用于更高的传输速率，并指定帧的优先级。我们将在 3.5 节中详细介绍上述这些内容。

3.1.7 链路层的数据包流

数据链路层位于物理层和网络层之间。在数据包传输过程中，它从网络层接收一个分组然后再封装相应的链路信息（如帧的头部加上 MAC 地址），在帧尾部加上帧校验和，然后再将帧传输到物理层。类似地，从物理层接收到一个数据包后，链路层会提取头部信息，验证帧校验序列，然后根据头部中的协议信息将有效载荷传递给网络层。但是在这些层之间，实际的数据包流会是什么呢？继续 1.5 节中有关数据包的生命历程的学习。我们将继续阐述数据包流在开源实现 3.3 节中对帧的接收和发送。

开源实现 3.3：调用图中的链路层数据包流

概述

链路层的数据包流沿着两条路径传输。在接收路径中，从物理层接收到一个帧，然后传递给网络层。在发送路径中，从网络层接收到一个帧，然后传递给物理层。在物理层和链路层之间的接口部分位于硬件中。例如，以太接口，将在开源实现 3.5 中介绍。我们将介绍设备驱动中的代码重点放在帧发送和帧接收中的软件部分。

框图

图 3-7 中介绍了链路层之上和之下的接口以及整个数据包流。“算法实现”部分将详细地解释数据

包流如何通过图3.7中的函数。对于介质访问控制和物理层之间的接口，请参考开源实现3.5。

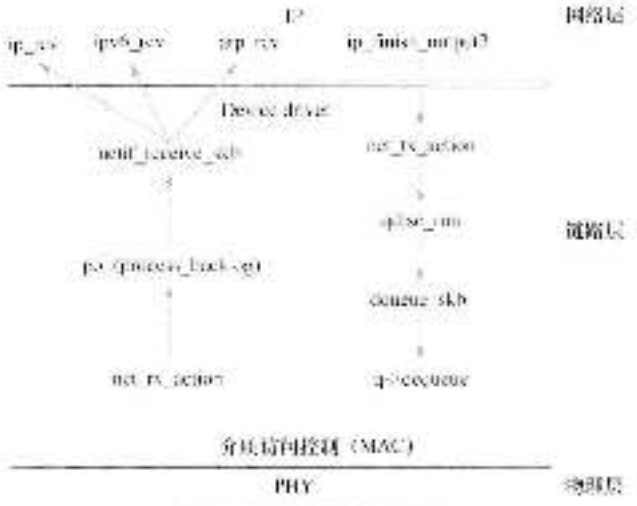


图3-7 链路层数据包流

数据结构

最关键的数据结构是sk_buff，它表示在Linux内核中的一个数据包。sk_buff中的某些字段用于记录目的，其他部分则用于存储数据包内容，包括头部和有效载荷。例如，在数据结构中的以下字段包括头部和有效载荷信息。

```
sk_buff_data_t          transport_header;
sk_buff_data_t          network_header;
sk_buff_data_t          mac_header;
unsigned char           *head,
                        *data;
```

算法实现

接收路径中的数据包流

当网络接口卡接收到一个帧时，就产生一个中断向CPU发出信号请求处理该帧。中断处理程序将分用dev_alloc_skb()函数分配sk_buff数据结构并将帧复制到数据结构中，然后处理程序将初始化sk_buff中的某些字段，尤其是用于上层协议的字段，并通知内核有帧的到达以便做进一步处理。

两种机制可以实现顺加处理：1）主的函数netif_rx()和2）自内核版本2.6以后处理进入的帧的新API net_rx_action()。前者只是一个中断驱动，后者为了高效使用中断和轮询的混合方式。例如，当内核正在处理一个帧时另一个新帧到达，内核可以继续处理前面的帧和入队列中的其他帧直到队列为空为止，而不会被新到达的帧所中断。根据一些基准测试结果，在高流量时使用新的API，CPU资源会降低，因此这里我们更强调使用新的API。

中断处理例程可包含一个或多个帧，具体根据驱动程序的设计而定。当内核被一个新到达的帧中断时，它调用net_rx_action()函数从软件中断NET_RX_SOFTIRQ中轮询接口列表。软件中断是一种后半部（bottomhalf）处理程序，它可以在后台运行以避免为了处理到达的帧而占用太多CPU时间。轮询以循环方式执行，带有一个能够被处理的最大帧数。net_rx_action()函数在每一个设备上调用poll()虚拟函数（一种在设备上轮流调用特定轮询函数的（调用）类函数）以便从入队列中出队。如果一个接口不能清除它的入队列，因为允许处理的帧数量或者net_rx_action()的可用执行时间达到极限，那么它将等待直到下一次轮询。在这个例子中，默认处理程序process_backlog()用于poll()函数。

poll()虚拟函数依次调用netif_receive_skb()来处理帧。当net_rx_action()被激活时，第三层（13）协议类型已经在sk_buff的协议字段中了，并且已由中断处理程序设置了。因此netif_

netif_receive_skb()知道第三层（L3）协议类型，并通过调用下列函数将帧复制到与第三层（L3）协议处理程序相关的协议套接字：

```
ret = pt_prev->func(skb, skb->dev, pt_prev, orig_dev);
```

这里函数指针 func 指向公用的第三层（L3）协议处理程序，例如 ip_rcv()、ipv6_rcv()和 arp_rcv()，分别处理 IPv4、IPv6 和 ARP（将在第 4 章中介绍）。到目前为止，帧接收处理已经完成，第三层（L3）协议处理程序接管数据包决定下一步做什么。

发送路径上的数据包流

在发送路径和接收路径上的数据包流是对称的。函数 net_tx_action()与 net_rx_action()相对应，当某个设备准备从软件中断 NET_TX_SOFTIRQ 发送一个帧时将调用该函数。与之前 NET_RX_SOFTIRQ 的 net_rx_action()一样，启动的处理程序 net_tx_action()可以管理延时的任务，如发送完一个帧后释放缓冲区空间。net_tx_action()执行两项任务：1）观察等待发送的帧的完成实被 dev_queue_xmit()函数发送出去；2）在发送结束后释放 sk_buff 结构的分配。外出队列中的帧请求一定的排队规则按照指定的时间发送。qdisc_run()函数选择下一个要发送的帧，并调用 dequeue_skb()从队列 q 中释放一个数据包。然后这个函数在队列 q 上调用相关的排队规则的 dequeue()虚函数。

练习

解释为什么在高流量负荷的情况下，使用新的 net_rx_action()函数会降低 CPU 负载。

3.2 点到点协议

本节将介绍点到点协议（PPP），这是一种在传统拨号上网或 ADSL 上网中广泛使用的协议。PPP 源于一种老的但是广泛使用的协议，高级数据链路控制（HDLC）。在它内部还运行着两个协议，链路控制协议（LCP）和网络控制协议（NCP）。随着以太网通过桥接设备，如 ADSL 调制解调器将家庭和组织机构连接到因特网服务提供商（ISP），就出现了对以太网点到点协议（PPPoE）的需求。图 3-8 表示了这些部分组成之间的关系。

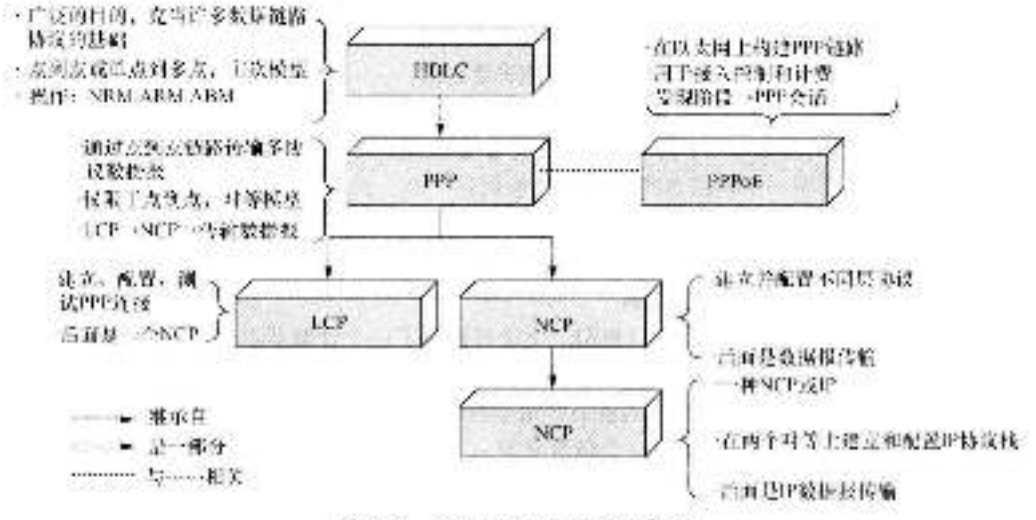


图 3-8 PPP 相关协议之间的关系

3.2.1 高级数据链路控制

HDLC 是从一个早期的协议，IBM 的同步数据链路控制协议（SDLC）演变而来的。它是一个 ISO 标准并且也是许多链路层协议的基础。例如，PPP 使用与 HDLC 相似的成帧。IEEE 802.2 逻辑链路控制（LLC）是对 HDLC 的修改。CCITT（国际电报电话咨询委员会）（1993 年被重命名为 ITU 国际电信联盟）修改了 HDLC，将它作为 X.25 标准的一部分，称为平衡链路访问规程（LAP-B）。在所有上述的变种中，

HDLC 支持点到点链路和单点到多点链路，半双工和全双工链路。下面我们来看看 HDLC 的操作。

HDLC 操作：介质访问控制

在 HDLC 中，节点既可以充当主站也可以充当次站。HDLC 支持以下三种传输模式，每一种提供一种控制节点访问介质的方法。

正常响应模式（NRM）：次站只能被动地传送数据以回应主站的轮询。轮询可由一个或多个帧组成。在单点到多点的情况下，次站通过主站通信。

异步响应模式（ARM）：次站可在没有主站轮询的情况下发起数据传输，但是主站仍然负责控制连接。

异步平衡模式（ABM）：通信中的每一方都可充当主站和次站，所以两种站点具有平等的地位。这种类型的站点称为组合站。

NRM 常用于单点到多点的链路中，就像一台计算机和它的终端之间的连接一样。尽管 ARM 很少使用，然而它在点到点链路中具有优势，但 ABM 会更好一些。ABM 具有更少的开销就像主站轮询所具有的特性一样，双方中的每一方都可以控制链路。ABM 特别适于点到点链路。

数据链路功能：成帧、寻址和差错控制

我们可以直接通过检查帧的格式来研究 HDLC 的成帧、寻址和差错控制问题，然后讨论流量控制和介质访问控制。图 3-9 描述了 HDLC 帧格式。

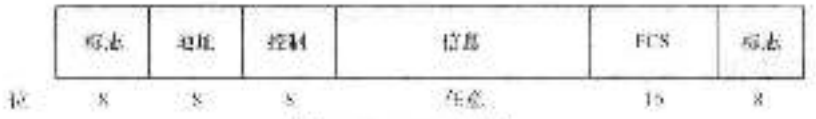


图 3-9 HDLC 帧格式

标志：标志值固定用 01111110 来定界帧的开头和结尾，如 3.1.1 节中所示，利用位填充来避免实际数据和标志值之间的歧义性。

地址：地址表示传输中包含的次站，尤其是在单点到多点的情况下。一个次站是在主站的控制下工作的，如同上面提到的在 HDLC 中的操作一样。

控制：这一字段指示帧的类型，且指示了其他控制信息，如帧的序列号。HDLC 具有三种类型的帧：信息、监管和无序帧。我们将在后面详细地研究它们。

信息：信息字段可以是任意长度的位。它承载着要传送的数据有效载荷。

FCS：一种 16 位的 CRC-CCITT 码。HDLC 允许肯定的和否定的确认。在 HDLC 中的差错控制是复杂的。肯定的确认表明一个帧或者所有帧成功地到达一点，而否定的确认指出某个接收的帧或者某个指定的帧。这里我们不再做过多的讨论。感兴趣的读者可以自行阅读“进一步阅读”中的补充材料。

数据链路功能：流量控制和差错控制

HDLC 中的流量控制也使用滑动窗口机制。发送方保留了一个计数器以便记录下一个要发送帧的序列号。另一方面，接收方也保留了一个计数器用以记录下一个将要到达帧的期望序列号，并负责检查接收到的帧的序列号是否与期望得到的帧的序列号相匹配。如果序列号正确并且帧没有出错，那么接收方将其计数器加 1，然后通过发送一个包含有下一个期望序列号的信息肯定确认发送方。如果接收帧不是所期望的或者利用 FCS 字段检测到帧出错，那么将丢弃这个帧，并且将向发送方发送否定确认并请求重新转发。一旦收到指示帧需要重发的否定确认，发送方将重新发送。这个方法就是 HDLC 中的差错控制。

帧类型

这些功能是通过各种各样的帧来实现的。信息帧，又称为 I-帧，承载来自上层的数据，也承载某些控制信息。包括两个 3 位字段，分别记录它自己的序列号以及来自接收方的确认号。正如前面提到的那样，这些序列号用来实现流量控制和差错控制。轮询/终止（P/F）也包含在控制信息中以便指示轮询是来自一个主站还是来自次站的上次响应。监管帧，又称为 S-帧，仅携带控制信息。在

前面我们对HDLC帧格式的讨论中，无论肯定确认还是否定确认都支持差错控制。一旦出现错误，根据控制信息中的规定，发送方可以重发所有正常帧也可以仅重传出错的帧。接收方也可以向发送方发送一个S－帧，请求临时停止传输操作。无序号帧，称为U－帧，也用于控制的目的。但是它不携带任何序列号，故此得名。U－帧包括各种命令，用于模式设置、信息转发和恢复，这里我们不进行详细讲解。

3.2.2 点到点协议

PPP是一种由IETF定义的标准协议，它通过点到点链路承载多协议数据包。它广泛用于拨号网和租用线路访问互联网。为了承载多协议数据包，它包括三个主要组成部分：

1）一种用于封装来自网络层数据包的封装方式。

2）一种用于处理连接建立、配置和断开循环的链路控制协议。

3）一种用于配置不同网络层选项的网络控制协议（NCP）。我们首先查看PPP的操作方式，然后了解它的功能。

PPP操作

在服务订购场景下，在进入类似于HDLC的MAC操作之前，PPP首先要完成登录和配置后才能发送数据包。PPP操作遵循图3-10中所示的阶段图。PPP首先要发送LCP数据包来建立和测试连接。连接建立后，发起连接的对等在交换网络层数据分组之前可能要对它进行认证。然后PPP开始发送NCP数据包以便配置用于通信的一个或多个网络层协议。配置完成后，网络层数据分组在连接终止之前通过链路发送。

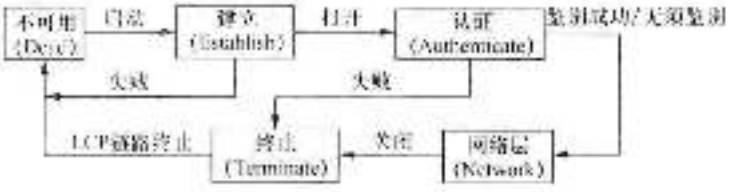


图3-10 PPP连接建立和释放的阶段图

我们将解释图3-10中的每一个主要状态迁移变化。

不可用到建立（Dead to Establish）：在对等方开始使用链路时，通过载波检测或网络管理者配置激活这个状态迁移阶段。

建立到认证（Establish to Authenticate）：LCP通过对等之间交换配置数据包来建立连接。所有不需协商的选项，直接配置为它们的默认值。只有独立于网络层的选项才需要进行协商，网络层协议配置的选项留给NCP来完成。

鉴别认证连接到网络（Authenticate to Network）：认证在PPP中是可选的，但是如果是链路建立阶段所需要的，那么将会切换到认证阶段。如果认证失败，则连接终止；否则，合适的NCP将开始协商每一个网络层协议。

网络连接终止（Network to Terminate）：很多情况下都会发生终止，包括载波丢失、认证失败、空闲连接过期、用户终止等。LCP负责交换终止数据包来关闭连接，稍后PPP会告诉网络层协议关闭。

有三种类型的LCP帧：配置、终止和维护。一对Configure-request（配置请求）和Configure-ack帧（配置确认帧）可以打开一条连接。选项（如最大可接收单元或者认证协议）在连接建立期间都可以协商的。表3-2总结了其他的功能。LCP帧是PPP帧的一种特殊情况。因此，在我们考虑LCP帧格式之前，我们首先介绍PPP帧的格式。

数据链路功能：成帧、寻址和差错控制

PPP帧被封装到一个类似于HDLC帧的格式中，如图3-11所示。标志值恰好与HDLC相同。它为成帧类供定界符。

表 3-2 LCP 的类型

类	类 型	功 能
配置	Configure-request	通过对选项的需要的更改来打开一条连接
	Configure-ack	确认配置请求
	Configure-nak	由于不可接收的选项而拒绝配置请求
	Configure-reject	由于不可识别的选项而拒绝配置请求
终止	Terminate-request	请求关闭连接
	Terminate-ack	确认终止请求
维护	Code-reject	来自对等的未知请求
	Protocol-reject	来自对等的不支持的协议
	Echo-request	回顾请求（用于调试）
	Echo-reply	由于对回顾请求的响应（仅用于调试）
	Discard-request	仅丢弃请求（用于调试）

	标志 01111110	地址 11111111	控制 00000011	协议	信息	FCS	标志 01111110
位	8	8	8	8或16	任意	16或32	8

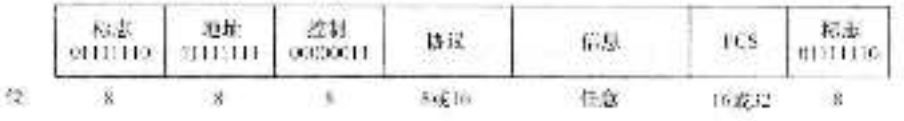


图 3-11 PPP 帧格式

PPP 帧和 HDLC 帧之间的区别总结如下：

1）地址值固定为 11111111，这是 HDLC 格式中的所有站点地址。因为在点到点链路中只有两个对等，所以就不需要指出单独站点的地址。

2）控制时具有固定的值 00000011，它对应于 HDLC 中的无序号帧格式。这意味着在默认时，PPP 无序号和确认。感兴趣的读者可参考 RFC1663 文档，其中定义了一个扩展部分以使 PPP 连接可靠。

3）添加的协议字段指示这个帧携带的是何种类型的网络层协议。IP 或 IPX。它的默认字段长度为 16 位，但是使用 LCP 协商可将它减少到 8 位。

4）信息字段的最大长度称为最大接收单元，默认为 1500 字节。MRU 的其他值是可以协商的。

5）默认 FCS 校验长度为 16 位，但是它可以通过 LCP 协商扩展至 32 位。如果帧内部检测到错误，则丢弃该帧。响的重传由上层协议负责。

数据链路功能：无流量控制和介质访问控制

由于 PPP 是全双工方式且在点到点链路中仅有两个站点，所以 PPP 不需要介质访问控制。另一方面，PPP 也不提供流量控制，而是将该功能留给上层协议去处理。

LCP 和 NCP 协商

LCP 帧是一种带有协议字段值为 0x8021 的 PPP 帧，0x 代表十六进制数。协商信息以如下 4 个主字段形式嵌入到信息字段中：编码（Code）表示 LCP 类型，标志符（Identifier）用来匹配请求和响应，长度（Length）指示 4 个字段的总长度，数据（Data）部分携带协商选项。

由于在因特网中 IP 是占据主导位置的网络协议，所以我们特别对 PPP 上的 IP 感兴趣。我们将在 3.2.3 节介绍用于 IP 的 NCP——因特网协议控制协议（IPCP）。

3.2.3 因特网协议控制协议

IPCP 是用来配置 PPP 上 IP 的 NCP 中的一个协议。PPP 首先通过 LCP 建立起一条连接，然后使用 NCP 来配置它所承载的网络层协议。配置后，数据数据包就可以通过链路传输。IPCP 使用一种类似于 LCP 的帧格式，并且它的帧也是将协议字段设置为 0x8021 的一种特殊 PPP 帧。IPCP 使用的交换机制也类似于 LCP。通过 IPCP，在两个对等上的 IP 模块都可设置成被激活、配置和禁止。

IPCP 提供配置选项：IP 地址、IP 压缩协议和 IP 地址。第一个选项已经废弃，被第三个选项所代

体。第二个选项指示使用 Van Jacobson 的 TCP/IP 头部压缩。第三个选项允许对等提供一个在本地终端上使用的 IP 地址。在 IPCP 协商之后，普通 IP 分组可以在 PPP 链路上通过将 IP 分组封装到协议字段值设置为 0x0021 的 PPP 帧中来转发。

开源实现 3.4：PPP 驱动程序

概述

PPP 在 Linux 中的实现由两部分组成：数据平面 PPP 驱动程序和控制平面 PPP 守护程序（PPPd）。PPP 驱动程序建立一个网络接口，并在串口、网络网络编码和 PPP 守护程序之间传递数据包。PPP 驱动程序负责在 3.2.2 节中描述的数据经路层功能。PPPd 与对等进行协商以便建立链路连接并建立 PPP 网络接口。PPPd 也支持认证，因此它可以控制哪个系统可以建立一条 PPP 连接并能够指定它们的 IP 地址。

框图

PPP 驱动程序由 PPP 通用层和 PPP 信道驱动程序组成，如图 3-12 所示。



pppd	处理控制平面数据包
内核	处理数据平面数据包
PPP通用层	处理PPP网络接口、/dev/ppp设备、VJ压缩、多链路
PPP信道驱动程序	处理封装和成帧

图 3-12 PPP 软件架构

数据结构

在 Linux 中有异步和同步 PPP 驱动程序（参见 drivers/net 目录下的 ppp_async.c 和 ppp_synctty.c）。两者的不同之处在于 PPP 信道驱动程序所连接的 tty 设备类型。当连接的 tty 驱动程序是一个同步 HDLC 卡时，如 FarSite 通信有限公司的 FarSyne T 系列串口卡，就使用同步 PPP 信道驱动程序。另一方面，当 tty 驱动程序是异步串行线路，如英飞凌（Infineon）科技公司制造的 PEB 20534 控制器时，就使用异步 PPP 信道驱动程序。

与两种驱动程序相关的输入/输出（I/O）函数指针在 tty_ldisc_ops 数据结构中定义，通过它关联 I/O 函数就可以正确地激活。例如，对于异步 PPP 的 read 字段指向 ppp_asynctty_read()；而对于同步 PPP 的 read 则指向 ppp_sync_read()。

我们下面介绍一般适用数据包流的发送和接收，因为它们可以更好地反映 PPP 驱动程序中的数据包流。我们不再深入介绍两种 PPP 驱动程序的详细内容。

算法实现

数据包传输

将要发送的数据包存储在 sk_buff 结构中。将给它传递给 ppp_start_xmit()，它给数据包添加 PPP 头部并将数据包存储在转发队列中，也就是 xq 中（参见 ppp_generic.c 中的 ppp_file 数据结构）。最后，ppp_start_xmit()调用 ppp_xmit_process()，将数据包从 xq 队列取出，然后调用 ppp_send_frame()来对数据包进行处理，比如头部压缩。这一步之后，ppp_send_frame()既可以调用异步 PPP 函数 ppp_async_send()，也可以调用同步 PPP 函数 ppp_sync_send()，通过参数的驱动程序发送数据包。

数据包接收

当同步或异步驱动程序接收到一个到达的数据包时，将这个数据包传递给 PPP 通用驱动程序的

ppp_input()函数，它会将到达的数据包加入接收队列中，也就是 inq 中。PPPd 将通过/dev/ppp 设备从队列中读取数据包。

练习

讨论为什么 PPP 功能在软件中实现，而以太网功能要在硬件中实现。

3.2.4 以太网上的 PPP（PPPoE）

对 PPPoE 的需求

随着以太网技术变得更加廉价占据市场主导地位，用户在家或者办公室建立以太局域网并不罕见。另一方面，宽带接入技术，比如 ADSL，已经成为家庭或者办公室访问因特网的一种普遍方法。在以太网局域网上的多个用户通过使用同宽带调制设备访问因特网，因此服务提供商想要找到一种类似于拨号上网的服务那样基于每个用户的计费接入控制。

PPP 成为在对等之间构建点到点关系的一种良好解决方案，但是以太网中包含多个站点。以太网上的点到点协议（PPPoE）就是用来协调同个对等体系的概念。它在以太网接口之上建立起一个虚拟接口，这样局域网上的每一个独立站点都可以与一个远程 PPPoE 服务建立一条 PPP 会话。这个服务器位于 ISP 中，又称为访问集中器（AC），通过公共的宽接设备相连。在局域网中的每个用户将 PPP 接口看成拨号服务一样。但是 PPP 帧被封装在以太网帧中。通过 PPPoE，用户计算机获得一个 IP 地址，ISP 就可以将静态地将 IP 地址和指定的用户名和密码相关联。

PPPoE 操作

PPPoE 的运行包括两个阶段：发现阶段和 PPP 会话阶段。在发现阶段，用户站点发现访问集中器的 MAC 地址并建立一个与访问集中器的 PPPoE 会话。一个独一无二的 PPPoE 标志符也被分配给了这个会话。一旦会话建立后，两个对等方进入 PPP 会话阶段就像 PPP 会话那样工作，即进行 LCP 协商。

发现阶段分为以下 4 个步骤进行：

1）接入因特网的站点广播一个初始化帧，以便请求远程集中器发回它们的硬件地址。

2）远程访问集中器将其 MAC 地址返回。

3）最初发起请求的站点选择一个访问集中器并发送一个会话请求帧给被选中的访问集中器。

4）访问集中器产生一个 PPPoE 会话标志符并返回一个含有会话标志符的确认帧。

PPP 会话阶段与普通 PPP 会话方式一样地运行，就像 3.2.2 节中看到的那样，除了在以太帧中仅承载 PPP 帧之外。当 LCP 终止一个 PPP 会话时，PPPoE 会话也会被拆除。一个新的 PPP 会话将从发现阶段就请求一个新的 PPPoE 会话。

一个普通 PPP 终止过程可以终止一个 PPPoE 会话。PPPoE 既允许发起站点也允许访问集中器发送一个显式的终止帧来关闭会话。一旦发送或接收到一个终止帧时，就不允许再传输帧了，即使普通的 PPP 终止帧也是如此（即不允许发送）。

3.3 以太网 （IEEE 802.3）

IEEE 802.3 最初是由 Bob Metcalfe 在 1973 年提出，以太网是众多局域网技术的竞争者之一，但最后成为了胜利者。在过去的 30 多年中，以太网被不断修改以适应新的需求，从而产生了庞大的 IEEE 802.3 标准族，并且未来还会继续演变。我们将向读者介绍以太网的演变过程和以太网的概念，并将讨论当前开发的热门话题。

3.3.1 以太网的演变：蓝图

作为标准的名称，“载波监听多路访问及冲突检测（CSMA/CD）访问方法和物理层规范”表明，以太网明显地区别于（如令牌总线或者令牌环等）其他局域网的是其介质的访问方法。以太网技术最初于 1973 年诞生于施乐（Xerox）实验室，后来由 DEC、Intel 和施乐公司于 1981 年标准化。就成了 DIX 以太网技术。尽管这个标准与施乐公司当初的设计不一样，但这个标准仍然保留了 CSMA/CD 的精髓。

到了1983年，IEEE 802.3工作组批准了经过微小改动后的DIX以太网标准。该标准成为后来著名的IEEE 802.3标准。此于施乐公司放弃了名为“以太网”的商标后，当人们提到以太网和IEEE 802.3两个术语时，它们之间已不存在差别。事实上，IEEE 802.3工作组自从制订了第一个以太网标准后，它就一直引领着以太网的发展。图3-13描绘了以太网发展的里程碑。在过去的30年里，它经历了多次重大的修改。这里我们只列出了主要的发展趋势。



图3-13 以太网发展过程中的里程碑

从低速到高速：从运行于3Mbps的原型提高到10Gbps，速度已经提高了3000倍。目前正在进行的工作（IEEE 802.3ba）旨在将数据率进一步提高到40～100Gbps。尽管令人吃惊，但这项技术仍然很贵，它被世界广泛接受。以太网技术已构建到几乎所有的台式计算机的主板和笔记本计算机的主板中。我们相信以太网将成为有线连接中的无处不在的技术。

从共享到专用介质：早期的以太网运行在同轴电缆构成的总线式结构上。多站点利用CSMA/CD MAC算法共享总线。因此在以太网总线上的冲突很常见。随着10BASE-T的开发，两台设备之间的独占专用介质成为主流。独占专用介质是广泛开发的全双工以太网所必需的。全双工允许两个站点通过独占专用介质同时发送，这样便有效地加倍了带宽。

从局域网到城域网再到广域网：以太网上著名的局域网技术。有两个因素使以太网技术向城域网和广域网市场发展。首先是成本。以太网因其简单性，而使其实现成本较低。如果城域网和广域网也采用以太网技术，那么它们之间互操作性的实现将花费更少的劳力和金钱。第二个因素是全双工的特点，全双工取消了对CSMA/CD的需要。从而放宽了对以太网使用距离上的限制——数据就可以尽可能远地传输到物理链路可达的任何地方。

丰富的媒介：术语“以太”曾被认为是传播的电磁波通过空间的介质。尽管以太网从未使用以太来传输数据，但是它的确运用各种介质来承载信息：同轴电缆、双绞线和光纤。“以太网就是多媒体！”——这是Rich Seifert在他的《Gigabit Ethernet》（1998）一书中描绘的美好情景。按照网速以及它们可以运行的介质，表3-3列出了所有802.3族的成员。

并不是所有的802.3成员在商业上都取得了成功。比如，100BASE-T2就是一个从来没有商业化的产品。相反，有些则是很成功的，几乎每个人都可以在连接到局域网的计算机后面找到10BASE-T或100BASE-TX网络接口卡（NIC）。目前大多数台式计算机的新主板都带有100BASE-TX或1000BASE-T以太网接口。括号中的数字表示IEEE批准规范的年代。

表 3-3 802.3 族

速度 \ 媒介	同轴电缆	双绞线	光纤
10Mbps（2）*		1BASE5（1987） 3BASE-TL（2003）	
10Mbps	10BASE5（1983） 10BASE2（1985） 10BROAD36（1985）	10BASE-T（1990） 10PASS-TS（2003）	10BASE-FL（1993） 10BASE-FP（1993） 10BASE-FB（1993）
100Mbps		100BASE-TX（1995） 100BASE-T4（1995） 100BASE-T2（1997）	100BASE-FX（1995） 100BASE-LX/BX10（2003）
1Gbps		1000BASE-CX（1998） 1000BASE-T（1999）	1000BASE-SX（1998） 1000BASE-LX（1998） 1000BASE-LX/BX10（2003） 1000BASE-PX10/20（2003）
10Gbps		10GBASE-T（2006）	10GBASE-R（2002） 10GBASE-W（2002） 10GBASE-X（2002）

以太网术语

以太网有着众多的物理规范，如表3-3所示。标注按［1/10/100/1000/10G］［BASE/BROAD/PASS］［-］phy 格式进行。第一项是速度，第二项说明信号是基带还是宽带。几乎所有的以太网信号都是基带信号，除了旧的10BROAD36和10PASS-TS之外。最初的第一项表示以100m为单位的最大长度，在第二项和第三项之间没有破折号。后来改成用来表示物理规范，如介质类型和信号编码，并在第二项和第三项之间使用破折号。

历史演变：以太网的竞争对手

历史上，有些局域网技术，如令牌环、令牌总线、FDDI、DQDB和ATM局域网仿真曾经与以太网竞争，但最终以太网在有线局域网系统中脱颖而出。以太网成功背后的根本原因在于以太网比其他技术更简单，简单意味着更低的成本。人们不愿意支付比必需更多的费用，从这一方面来讲以太网肯定会胜出。

以太网为什么会比其他技术更便宜？以太网没有其他技术可扮演的花哨功能，如优先机制、服务质量机制和中央（集中）控制。因此以太网并不需要处理令牌，也没有离开和加入环的复杂性。CSMA/CD相当简单，可以很容易地在硬件逻辑中实现（参见开源实现3.5）。全双工更简单。这种优点使以太网成为最终的赢家。

然而，目前以太网仍然遇到了许多竞争者。其中最强劲的对手就是无线局域网。无线网具有高度的移动性，这是以太网不具备的特点。只要需要移动性，无线局域网将获胜。但是，当移动性不是必要的时，正如台式计算机，以太网仍然是大多数人的选择，因为大多数主板内置了以太网接口。另一方面，以太网还试图将自己扩展用于第一公里和广域网技术中。鉴于目前拥有大量基于xDSL和SONE技术的安装，如果更换不可避免，我们认为以太网将在很长时间去逐步取代它们。由于同样的原因，尽管以太网是如此流行，但是如果现有的安装很便宜也很令人满意，那么更换可能永远不会发生。

3.3.2 以太网 MAC

以太网成帧、寻址和差错控制

802.3 MAC子层是独立于介质的以太网部分。与逻辑链路控制（LLC）子层一起在IEEE 802.2中指定，它们一起构成OSI模型中的数据链路层。与MAC子层相关的功能包括数据封装和介质访问控

道，以及用于 LLC 子层的功能目的是成为以太网、令牌环、无线局域网等的公共接口。在像网桥这样的功能中 Linux 主要实现了后一部分，因为前置帧是由 LLC 络式指定的（参见 3.6 节）。图 3-14 为我们给出了无标记[1]的以太网帧。通过帧格式，我们首先介绍了以太网成帧、寻址、差错控制，而将接入控制和流量控制问题留到以后再讨论。

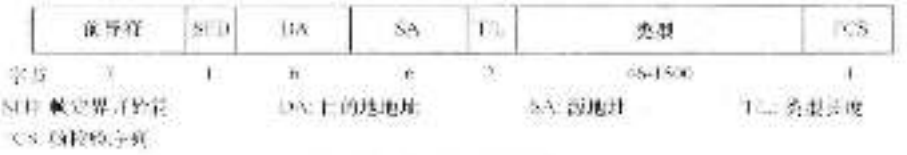


图 3-14 以太网帧格式

前导符：该字段同步在接收端的接收信号定时。它的值固定长 56 位，以 1010…1010 的顺序传输[2]。精确地，触边要是以特殊的物理编码发送信号来标识，具体取决于 PHY 是如何指定的。例如，100BASE-X 以太网中前导符中的第一个字节（1010/1010/使用 4B/5B 编码转换成值为 11000/10001/的四个特殊代码组 J/K/。4B/5B 编码将正常的 1010（以传输顺序）转换成 01011 以免产生歧义。同样，100BASE-X 会添加四个值为/01101/10001 的特殊代码组/T/R 来标记帧的结束。

SFD：此字段指示在传输顺序中具有值为 10101011 的帧的开始。从历史上看，DIX 以太网标准指定一个 8 字节的前导符。正好与 802.3 帧中的前导字段的定义相同，只是命名上有所区别而已。

DA：该字段包含 48 位以 3.1.2 节中介绍过的格式表示的目的地 MAC 地址。

SA：该字段包含 48 位源 MAC 地址。

类型/长度：由于历史原因，这个字段包含两个含义。DIX 标准指定该字段为一个有效载荷承载类型的代码，正如 IP，而 IEEE 802.3 标准指定该字段是载荷字段[3]的长度，而协议类型留给 LLC 子层。以后的 802.3 标准（1997 年）又进行了类型字段，导致出现了目前对该字段所具有的双重诠释。区分它们的方法很简单：因为数据字段永远不会大于 1500 字节，因数据字段的值小于或等于 1500 字节的就是长度字段，而数据字段的值大于或等于 1536（0X600）字节的则是类型字段。虽然目的不同，但这两种诠释的共存正是由于上述两很容易区分。介于两者之间的值没有定义。大多数帧将该字段用作类型字段，就是因为占主导地位的网络层协议，比较是 IP，使用它作为类型字段。

数据：这一字段携带着 46～1500 字节大小的数据。

FCS：这一字段携带着 32 位 CRC 码作为帧校验序列。如果接收端发现一个不正确的帧，则默默丢弃该帧。由是发送端对帧是否被丢弃则一无所知。帧重传的责任留给上层协议，如 TCP 协议来处理。这种方法非常有效，因为为了启动下一次发送，发送端不需要等待确认。这样，错误不会一个大问题，因为误码率在以太网物理层是非常低的。

帧的大小是可变的。我们除常除去前两个字段，以太网帧的最小长度就为 64（=6+6+2+46+4）字节，最大长度为 1518（=6+6+2+1500+4）字节。

介质访问控制：发送和接收流

我们现在来看看如何在以太网 MAC 内部发送和接收的，你将会非常详细地看到 CSMA/CD 是如何工作的。图 3-15 显示了 MAC 子层在帧发送和接收过程中所起的作用。CSMA/CD 以一种机名字含义一样的方式工作。当在帧传输时，CSMA/CD 首先侦听电缆。如果检测到载波信号，即电缆忙，它就继续监听电缆直到它变为空闲位置；否则，它等待一个很小的间隔后就开始发送。如果在传输过程中检测到冲突，CSMA/CD 使向电缆发送加塞信号，中止传输，再次发送之前等待一个随机后退时间。图 3-16给出了传输流程图后的正确流程步骤。注意，在全双工环境下，载波监听和冲突检测实际上是不存在的。

---

① 一个以太网帧可以携带一个 VLAN 标记，当我们在 3.5 节中介绍 VLAN 时，我们会了解这样的格式。

② 以太网传输是按位数据来进行的，这在物理与层中被实现。

③ 有关数字段认为代表帧的大小是一个广泛存在的误解，这种观点是不正确的。线缆上的标志是特殊的物理编码或者信号。以太网 MAC 可以很容易地计算出它已经接收了一帧中的多少字节。

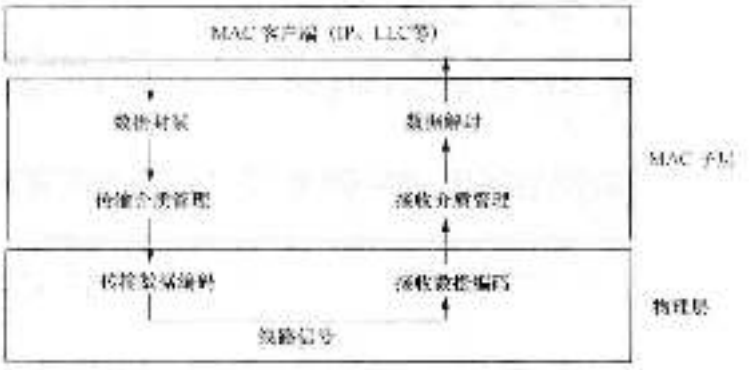


图 3-15 在 MAC 子层，帧的传输和接收

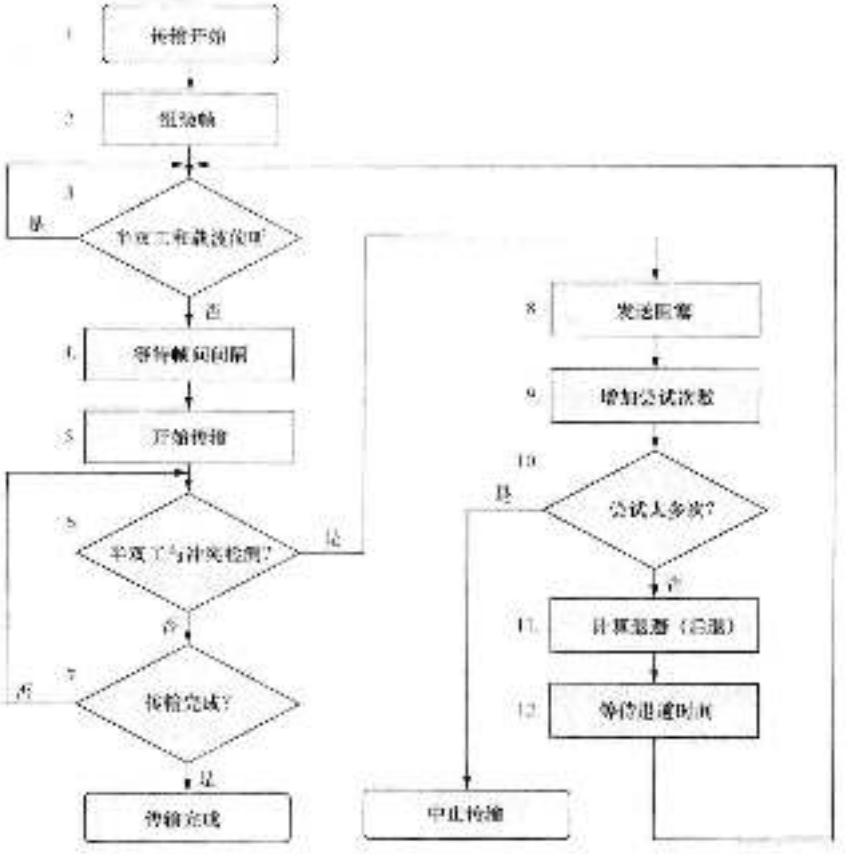


图 3-16 CSMA/CD 帧传输流

1）MAC 客户端（IP、LLC 等）请求传输帧。

2）MAC 子层对来自客户端来的数据前附加 MAC 信息（前导符、SFD、DA、SA、类型和 FCS）。

3）在半双工模式下，CSMA/CD 方法侦听载波以便确定传输信道是否忙。如果忙，则将传输推迟到该信道空闲为止。

4）等待时间称为帧间间隔（IFG）。所有以太网类型的时间长度都是 96 位的倍数。位时间是一位传输所需要经过的时间，即位速率的倒数。IFG 留给接收方处理的时间，例如用于识别帧的中断和指针激发。

5）开始传送帧。

6）在半双工模式下，如果在帧传输过程中发生冲突，则发送方应继续监控。监控方法取决于速度

的介质类型。在某个同轴电缆上的多个数据传输会导致绝对电压高于正常水平。对于双绞线，在发送帧的同时在接收线对上侦听接收信号就可以断定是否出现了冲突。

7）在传输过程中没有检测到冲突的情况下，帧就会继续传输，直到传输完成为止。如果在半双工模式下检测到冲突，就进行步骤8）～12）。

8）发送方发送一个32位长的阻塞信号，以确保冲突足够长的时间而让所有站点都知道。阻塞信号模式是未指定的。常见的实现方式是持续大于32位的数据，或者使用产生前导符的电路传输交替的1和0组合。

9）中止当前的传输，并试图按计划安排再次传输。

10）尝试重发的最大次数为16，如果仍然无法传输，则放弃该帧。

11）在尝试重新发送时，回退时间间隔是以时隙为单位随机地从$0-2^k-1$中选择一个数，其中$k=\min(n, 10)$，n是尝试的次数。取值范围呈指数增长，所以称为截断二进制回退算法。对于10/100Mbps以太网，时隙的持续时间是521位时间，对于1Gbps以太网，则为4096位时间。我们在3.3.3节中讨论千兆以太网时将说明选择时隙持续时间的原因。

12）等待一段回退的时间间隔，然后尝试重新发送。

当对帧长度（检查帧是否太长或太短）、目的MAC地址、FCS以及传递到MAC客户端之前的十进制边界检查完成后，帧的接收会容易得多。图3-17说明了接收流。过程如下所示。

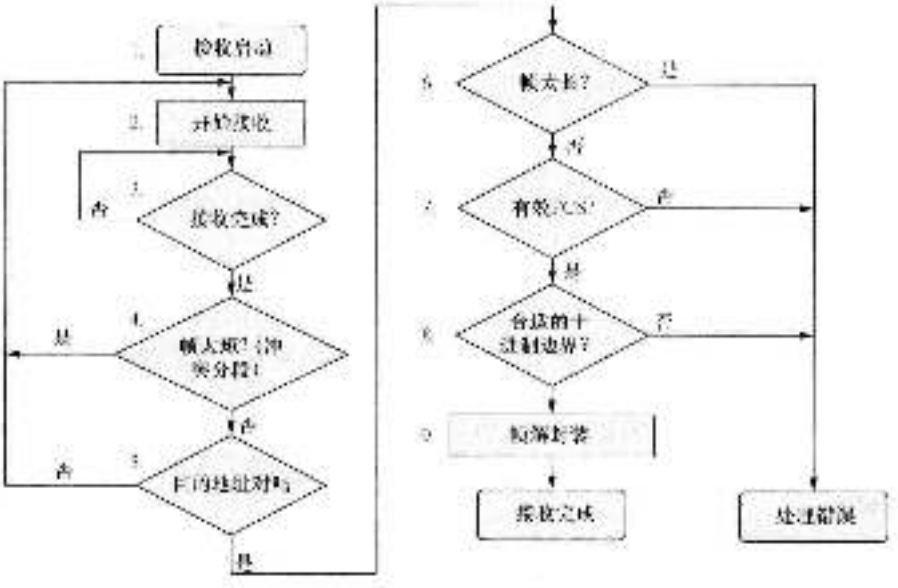


图3-17 CSMA/CD的顺接收流

1）接收方的物理层检测到有帧到达。

2）接收方将收到的信号进行解码并将数据传递到MAC子层，除了前导符和SFD外。

3）只要接收信号持续，接收进程就会继续。当信号中止时，到达的帧就被断成一个八进制字节的达界。

4）如果该帧太短（小于512位），它就会被视为冲突分段并被丢弃。

5）如果目的地址不是接收方，帧也将被丢弃。

6）如果帧太长，它将被丢弃，并记录错误用于管理统计。

7）如果原有一个不正确的FCS，它也将被丢弃，并记录错误。

8）如果帧的大小不是8位的整数倍，它将被丢弃，并记录错误。

9）如果一切正常，帧解封装并将该字段上传到MAC客户端。

冲突会影响性能吗

承诺冲突听上去很可怕！但是冲突是正常CSMA/CD仲裁机制的一部分，而不是系统故障的结果。

冲突会导致错误的帧，但如果检测到冲突时就停止传输，情况以及并不会很糟糕。在进一步分析因冲突所造成的时间浪费之前，我们先回答一个关键问题：冲突会在哪里发生呢？我们利用图 3-18 中的发传输帧来回答这个问题。



原则：往返时间2τ小于传输一个最小帧的时间

图 3-18 具有传播延迟的冲突检测

假设 A 站发送一个 64 字节的最小帧，帧的第一位到达 B 站之前的传播时间为 τ。即使具有载波侦听，B 站也架不住在 τ 时间内传输帧，因此会造成冲突。我们回一步假设一种最坏的情况，B 站正好在 τ 时刻发送，这样就产生了冲突。冲突然后经历了另一个 τ 后传输到达 A 站。如果 A 站在往返传播时间 2τ 之前完成最小帧的传输，它就没有机会启用冲突检测并安排重发。这样帧就丢失了。为了使 CSMA/CD 能够正常运行，往返时间应小于传输最小帧所需要的时间。也就是说 CSMA/CD 机制将两站点之间的距离限制在冲突域中。这一限制使得千兆以太网的设计更加复杂。我们将在 3.3.2 节介绍千兆以太网时深入讨论这个问题。因为最小帧的大小为 64 字节。这也意味着冲突必须发生在距离限制的帧的前 64 字节传输期间。如果发送了大于 64 字节的帧，那么由于载波能够被其他站点提前侦听到，所以就可以根除冲突。

如果我们将 32 位阻塞也考虑进来，那么已经传输的第一个的实际位数加上阻塞位数必须不超过 511 位，正如在步骤 4 中所述的那接收端。因为 512 位（=64 字节）是正常帧的最小长度。其他情况下，接收方会认为这些位是正常的帧，而不是一个碰撞分段。因此，浪费的位时间的最大数量是 511 + 64（来自重导符）+ 96（来自 IFG）= 671。这只是一个大帧中的一小部分。此外，我们必须强调这是在最坏的情况下发生的。大多数冲突是在前导符阶段检测到的，因为两个发送站点之间的距离并不远。在这种情况下，浪费的位时间只有 64（来自前导符）+ 32（来自阻塞）+ 96（来自 IFG）= 192。

最大帧速率

一个发射机（接收机）在一秒内能发送（接收）多少个帧？这是一个有趣的问题，尤其是当你设计或分析数据包处理设备（如交换机）时，或需要确定设备可秒需要处理多少帧。帧的传输是从一个 7 字节的前导符和 1 字节的 SFD 开始的。为了使链路到达其最高传输速率（以每秒多少个帧为单位），所有传输的帧应保持最小，即 64 字节。不要忘了两个连续传输的帧之间还有 12 字节（96 位）的 IFG。一个帧传输总共占用（7 + 1 + 64 + 12）×8 = 672 位时间。在一个 100Mbps 的系统中，每秒可传输的最大帧数为 $100 \times 10^6 / 672 = 148\,800$。此值通常称 100Mbps 链路最大帧速率。如果一台交换机有 48 个接口，那么聚合的最大帧传输速率将是 148 800 × 48 = 7 140 400，即超过 700 万。

全双工 MAC

早期的以太网使用同轴电缆作为传输介质并将工作站连接成总线拓扑结构。由于双绞线易于管理，所以在大多数情况下它取代了同轴电缆。主流的方法是利用双绞线将每台工作站连接到集中设备（如集线器或交换机）构成一种星形拓扑结构。对于流行的 10BASE-T 和 100BASE-TX，双绞线中有一对专用于用户发送或接收。[1] 如此一来就能在接收线对上一边通过侦听接收到的信号来确定碰撞，一边在发送线对上发送。然而，这仍然是低效率的。既然介质专用于星形拓扑结构设置中的点到点通信，那

[1] 在 1000BASE-T 中，发送与接收双对同时产生，考虑到标称为 8P 电缆分离为个信号与成本，实际不必须的。

久为什么新的以太网技术还要使用冲突作为一种“仲裁”方法呢？

1997 年，IEEE 802.3X 任务组在以太网中添加了全双工操作，也就是说，发送和接收可以在同一时间进行。在全双工模式下不再存在载波侦听或冲突检测，因为已经不再需要它们了——在一条专用线路上不存在“多路访问”了。因此，CS、MA 和 CD 都不见了。有趣的是，这对以太网的设计是一个戏剧性的变化，因为最初的以太网就是以其 CSMA/CD 而闻名的。为了运行全双工以太网，应满足以下三个条件：

1）传输介质必须能够在两个无干扰地传输和接收。

2）传输介质必须仅专用于两个节点，形成点到点链路。

3）两个端点必须都配置为全双工模式。

IEEE 802.3 标准则通过排除了集线器运行全双工模式的可能性，因为集线器的带宽是共享的，而不是专用的。全双工传输的三个典型案例场景分别是站点 - 站点链路，站点 - 交换机和交换机 - 交换机链路。在任何情况下，这些链路都需要专用的点到点链路。

全双工以太网在效果上加倍了两站点之间的带宽。它还放宽了由于使用 CSMA/CD 所导致的对距离的限制。这对高速和广域传输来讲非常重要，这一点我们将在 3.3.3 节进行讨论。目前，几乎所有的以太网接口都支持全双工模式。通信双方都能进行自动协商，以确定是否双方都支持全双工。如果是这样，为了更高的效率双方都将运行在全双工模式下。

以太网流量控制

以太网中的流量控制取决于双工模式。半双工模式采用一种称为假载波（false carrier）的技术，有了它如果接收机不支持更多传入的帧时，它就可以在共享媒体上传输一个载波，比如一系列 1010…10，直到它能够接受更多的帧为止。发射机将侦听载波，并推迟其随后的传输。此外，如果冲突检测机无论何时当侦听到帧的传输时都可以强制一次冲突，导致发送方回退并重新安排它的传输。这种技术称为载波冲突。这两种技术统称为背压（back pressure）。

然而，在全双工模式下背压是无效的，因为不再使用 CSMA/CD。IEEE 802.3 在全双工模式下指定了一个暂停 PAUSE 帧用于流量控制。接收方采取地向发送方发送一个 PAUSE 帧，发送方接收到 PAUSE 帧后就立即停止发送。PAUSE 帧携带一个字段，pause time，告诉发送方应该停止发送多长时间。由于不容易提前估计暂停时间，所以在实践中 pause time 总是设置为最大值来停止传输，并当接收方可以接收更多的帧时将另一个 pause_time = 0 的 PAUSE 帧发送到发送方以便恢复发送方的传输。

在以太网中，流量控制是可选的。它可以由用户或通过自动协商被配置。IEEE 802.3 标准提供了一种介于 MAC 和 LLC 之间的可选子层，即 MAC 控制子层，它定义了 MAC 控制帧以提供 MAC 子层运行的实现操作。PAUSE 帧是一种 MAC 控制帧。

开源实现 3.5：CSMA/CD

概述

CSMA/CD 是以太网 MAC 中的一部分。大部分以太网 MAC 是在硬件中实现的。一个开源以太网的例子可以从 OpenCore（www.opencores.org）获得，其中提供了一个全面的 Verilog 代码。全面的意思是 Verilog 代码是完整的，可以通过一系列的 EDA 编译成电路实现。它实现 10Mbps 和 100Mbps 以太网的 IEEE 规范的第 2 层协议实现。

框图

图 3-19 说明了 OPENCORE Ethernet Core 的体系结构，它主要由主机接口、发送（TX）模块、接收（RX）模块、MAC 控制模块、被称为介质访问管理模块（MIM）所构成。具体描述如下：

1）TX 和 RX 模块负责所有的发送和接收功能。这些模块处理前导码的生成和消除。这两个模块为错误检测包含了 CRC 产生器。此外，TX 模块执行用于冲突处理的随机时间的产生以及 Carrier Sense 和 Collision 信号以实现 CSMA/CD 的主体内容。

2）MAC 控制模块提供全双工流量控制，它在通信站点之间传送 PAUSE 控制帧。因此，MAC 控制

模块支持控制帧的检测和产生，与 TX 和 RX MAC 的接口，暂停 PAUSE 定时器和时限定时器。

3）MII 管理模块实现了 IEEE 802.3 MII 标准，它能提供以太网物理层与 MAC 层之间的互联。通过 MII 接口，处理器可以强制以太网物理层运行在 10Mbps 或 100Mbps，并配置它执行全双工或半双工模式。MII 管理模块具有操作控制器、移位寄存器、输出控制模块和时钟产生器的子模块。

4）主机接口是一种 WISHBONE（WB）总线将以太网 MAC 与处理器和外部存储器连接起来。WB 是 OpenCore 项目的开放规范，至今为止只支持 DMA 传输进行数据传送。主机接口还具有状态模块和注册模块。状态模块记录写入到相关缓冲区中的状态。寄存器模块用于以太网 MAC 操作，它包括配置寄存器、DMA 操作以及发送状态和接收状态。

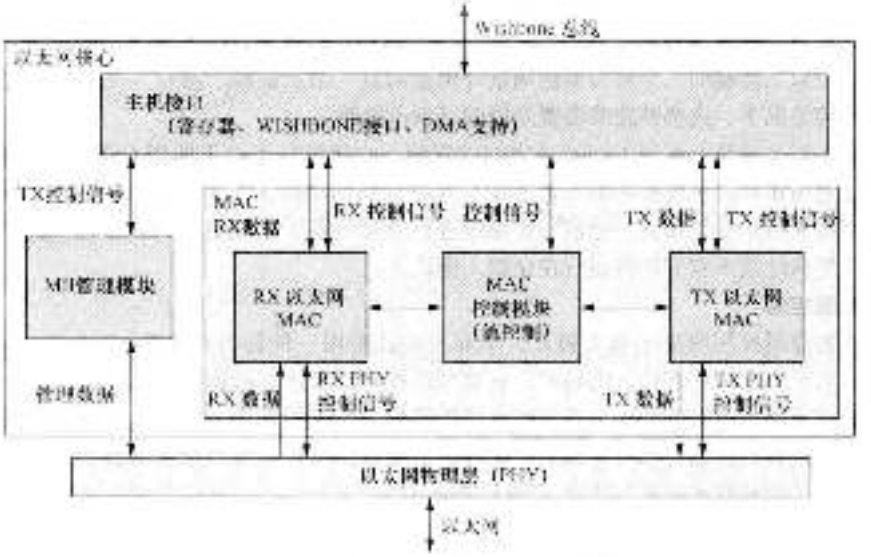


图 3-19 以太网 MAC 核心的体系结构

数据结构和算法实现

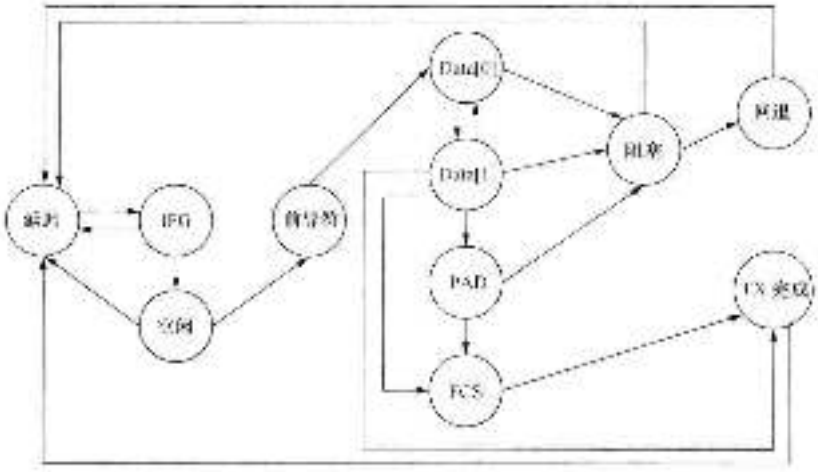
状态机：TX 和 RX

在 TX 和 RX 模块中，TX 和 RX 状态机分别控制它们各自的行为。图 3-20 中给出了两种状态机。这里我们仅描述 TX 状态机的行为，因为 RX 状态机的工作原理与其类似。TX 状态机从 Defer（延迟）状态开始，等待直到无载波（而 CarrierSense 信号为假），然后进入 IFG 状态。进入帧间间隔后（IFG），TX 状态机进入 Idle（空闲）状态，等待来自 WB 接口的传输请求。如果仍然没有载波，状态机就进入 Preamble（前导符）状态并启动传输；否则，它回到 Defer 状态，并等待再次出现无载波。在 Preamble 状态中，发送前导符 0x5555555 和起始定界符 0xd，并且 TX 状态机到达 Data [0] 和 Data [1] 状态以发送半位元组 nibbles，即以 4 位为半位的数据字节。单位元组传输是从最低有效字节（LSB）开始直到帧结束为止，每次发送一帧，TX 状态机告诉 Wishbone 接口提供下一个将要传输的数据字节。

- 如果在传输过程中发生了冲突，TX 状态机就进入 Jam（阻塞）状态以发送一个阻塞信号，等待 Backoff（回退）状态中的一段回退时间，然后又回到 Defer 状态重新尝试传输。
- 当仅剩下一字节需要发送（在传输过程中没有发生冲突）时，

  1）如果总的帧长度大于或等于最小帧长度，那么 TX 状态机就进入 FCS 状态以便计算数据的 32 位 CRC 值。如果启用 CRC 就将该值附加到帧结尾，然后就进入 TxDone（TX 完成）状态；否则，TX 状态机直接进入 TxDone 状态。

  2）如果帧的长度比最小帧长度短且启用了填充，那么 TX 状态机进入 PAD 状态，数据部分填充零直到满足最小帧长度条件为止（1）。其余的状态与 1）中所述相同。但是，当禁用填充时会跳过 PAD 状态。



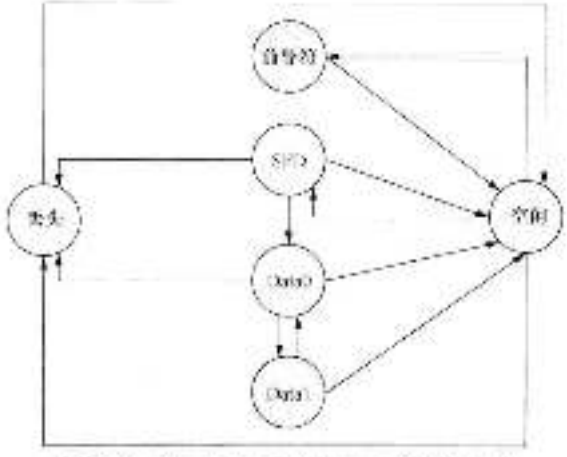


图 3-20 TX（上面）和 RX（下面）状态机

编程 CSMA/CD 信号和半位元组传输

图 3-21 是一个编程实现关键 CSMA/CD 信号和半位元组传输的 Verilog 代码段。输出信号是各种输入信号的一种算术组合，每个时钟周期会更新一次。所有输出信号也并行更新，这是与顺序执行的软件代码之间的重要区别。符号 ~、&、|、^ 和 ~ 分别表示运算“非”、“与”、“或”、“并或”和“赋值”。条件表达式“exp1? exp2: exp3”与 C 语言具有完全相同的含义（即，如果 exp1 的结果为真，取 exp2 值；否则，取 exp3 值）。

工作在半双工模式下的站点会观察 PHY 物理媒体上的活动。除了由于传输帧的联波外，也能观察到来自多个站点同时传输产生的冲突（由 Collision 变量指示）。如果发生冲突，所有站点停止传输，设置 StartJam（进入 Jam 状态）并在 Backoff 状态随机从间隔一个随机时间（设置 StartBackOff）。如果载波处在 Jam 状态或 Backoff 状态，状态机就回到 Defer 状态。

在“半位元组传输”下的代码，根据 TX 机器所在的状态，选择将要传输的半位元组（4 位）。在低字节（little-endian）传输期间 TX 状态机在 Data [0] 和 Data [1] 状态之间切换，因此 Mixed，发送数据半位元组。发替地装载在 TXDATA [3:0] 和 TXDATA [7:4] 中。在 FCS 状态中，CRC 值是逐个半位元组（nibble）装载的，就像使用 crc 移位寄存器实现的 CRC 计算一样。在 Jam 状态下，加载任意的十六进制值 1001（即 4'h9）作为阻塞信号，尽管阻塞信号的内容在 802.3 标准中没有指定。在 Preamble 状态下，前导符 0x5555555 和起始帧分隔符（定界符）0xd 依次装载。

```
CSMA/CD Signals
assign StartDefer = StateIPG & ~Rule1 & CarrierSense & NibCnt[5:0]
== IPGR1 & NibCnt[6:0] != IPGR2
  StateIdle & CarrierSense
  | StateBkn & NibCnt[q?] & (NoBackof | RandomEq0 | ~ColWindow |
MaxRetry)
  StateBackOff & (TxUnderRun | RandomEqByteCnt)
  StartTxDone | TooBig;
assign StartDefer = StateIdle & ~TxStartFrm & CarrierSense
                  | StateBackOff & (TxUnderRun | RandomEqByteCnt);
assign StartData[0] = ~Collision & StateData[0] & ~TxUnderRun &
                      ~MaxFrame;
assign StartJam = (Collision | UnderRun) & (StatePreamble
  & NibCntEq15)
                  (StateData[7:0]) | StateFCS | StateJam);
assign StartBackoff = StateJam & ~RandomEq0 & (Col?Window &
~RetryMax & NibCntEq7 & ~NoBckof;

Nibble transmission
always @ (StatePreamble or StateData or StateData or StateFCS or
StateJam or StateSFD or TxData or Crc or NibCnt or NibCntEq15)
begin
if(StateData[0]) MTxD_d[3:0] = TxData[3:0];   // Lower nibble
else if(StateData[1]) MTxD_d[3:0] = TxData[7:4]; // Higher nibble
else if(StateFCS) MTxD_d[3:0]={~Crc[28],~Crc[29],~Crc[30],
~Crc[31]}; // Crc
else if(StateJam)     MTxD_d[3:0] = 4'h9;     // Jam pattern
else if(StatePreamble)
if(NibCnt[q?5]      MTxD_d[3:0] = 4'hd;   // SFD
else      MTxD_d[3:0] = 4'h5;           // Preamble
    else     MTxD_d[3:0] = 4'h0;
end
```

图3-21 CSMA/CD信号和半位元组（nibble）传输

由于TX模块一旦检测到冲突后就开始回退等待，并等待一段从协调机指导导出的持续时间，如图3-22所示。应用“二进制指数”算法生成一个在规定区域内的随机回退时间。数组x中的x[i]元素是一个取值为0或1的随机值，数组x可以看成二进制的10位随机值的表示（总共10位，作为一系列0~$2^{i}-1$的随机数，这里i=min(n, 10)，而n是重试次数）。根据图3-22中的元素编码，

```
assign Random[0] = x[0];
assign Random[1] = (RetryCnt > 1) ? x[1] : 1'b0;
assign Random[2] = (RetryCnt > 2) ? x[2] : 1'b0;
assign Random[3] = (RetryCnt > 3) ? x[3] : 1'b0;
assign Random[4] = (RetryCnt > 4) ? x[4] : 1'b0;
assign Random[5] = (RetryCnt > 5) ? x[5] : 1'b0;
assign Random[6] = (RetryCnt > 6) ? x[6] : 1'b0;
assign Random[7] = (RetryCnt > 7) ? x[7] : 1'b0;
assign Random[8] = (RetryCnt > 8) ? x[8] : 1'b0;
assign Random[9] = (RetryCnt > 9) ? x[9] : 1'b0;
always @ (posedge MTxClk or posedge Reset)
begin
  if(Reset)
    RandomLatched <= 10'h000;
  else
    begin
      if(StateCam & StateCam_q)
        RandomLatched <= Random;
    end
end
assign RandomEq0 = RandomLatched == 10'h0;
```

图3-22 回退随机产生器

当 RetryCnt 大于 1 时，如果 x [1] -1 从 Random [1] -1；否则，Random [1] -0。读写语义，当 RetryCnt 为 1 时，直到此值中更多高位很可能被设置为 1，这就意味着随机数的范围值以重试次数指数地增长。得出随机值之后，会要将它递减于冲突而被延迟（从 State Jam 和 State Jam Q 变量进行判断）。它设置 RandomCatched 变量，如果随机值刚好为 0（即后退时间为 0），则设置 RandomEnd 变量并且此过程也不会启动（表图 3-21 中的最后一个赋值语句 Start Backoff 为假）。

练习

1. 如果以太网 MAC 总是双工模式运行（且没有冲突），那么设计中哪些特征是应该禁止？

2. 由于全双工模式比半双工模式具有更加简单的设计，读者的观点是，后者先出，那么为什么还要在以太网 MAC 中实现半双工模式？

历史演变：电力线路网络互联：HomePlug

以太网是局域网中占据主导的技术，但是它需要为有线连接从一个节点到另外一个节点布置网络电缆。虽然无线局域网可以完全去掉电缆，但无线信号容易受到各种干扰并且不稳定。介于两者技术之间的一种不经济的但有用的解决方案就是 HomePlug。它能够利用电力线路传输数据。以太网适配可以连接到电力线路适配器上，然后插入到标准插座上。其他设备也可以同样的方式实现连接，数据流能通过电力线路从计算机传输到计算机。从功能的角度来看，因此在两个电源插座之间就不需要额外的线路连接。

HomePlug 依赖于 OFDM 调制才能通过电力线路。HomePlug 1.0 以半双工模式允许速度高达 14MB/s。专用解决方案在 CuDo 模式允许高达 85Mbps 的速率。后来的标准规范将速度提升到 189Mbps。该解决方案是家庭和办公室线路布置的一种很有前途的互联方式。

3.3.3 以太网的精选主题

千兆以太网

为千兆以太网创建规范的任务原本分为两个工作组：802.3z 和 802.3ab。后来的以太网工作组，在第一公里（EFM）也规定了一个新的运行在千兆速率的 PHY。为简单起见，我们将后一部分的讨论留到 EFM 中。表 3-4 仅列出了在 802.3z 和 802.3ab 中的规范。

表 3-4 千兆以太网的物理规范

工作组	规范名字	描述
IEEE 802.3z (1998)	1000BASE-CX	使用 8B/10B 编码的 25m 150Ω 屏蔽双绞线（STP）
	1000BASE-SX	使用 8B/10B 编码的短波激光多模光纤
	1000BASE-LX	使用 8B/10B 编码的长波激光多模或单模光纤
IEEE 802.3ab (1999)	1000BASE-T	使用 8B/10B 的 100m 四对 5 类（或更好）非屏蔽双绞线（UTP）

千兆以太网设计的一个非点在于 CSMA/CD 的距离限制，这对于 10Mbps 和 100Mbps 以太网不存在问题。在 100Mbps 以太网中，铜线连接距离的大约是 200m，对普通的电缆已经足够了。该距离甚至要比 10Mbps 以太网的更长。但是，千兆以太网传输比 100Mbps 以太网快 10 倍，而距离限制则缩减为 1/10。对于许多网络部署，大约 20m 的限制是不能接受的，千兆以太网的目标之一就是在帧格式不变（即最小帧大小）的情况下就需能够工作良好。

IEEE 802.3 标准在就之后追加一系列扩展位以确保传输时间超过往返时间。这些位可以是物理层的任何非数据符号。这种技术称为载波扩展，在不改变最小帧大小的情况下扩展帧的长度。然而，最终的开销很差，尽管技术的初衷是好的。相反，全双工以太网并不需要 CSMA/CD，使得这一解决方案成为不必要的。全双工以太网的实施比半双工以太网简单得多。吞吐量更高，而且距离上的限制也不成问题。如果它是不必要的，那么为什么我们还要费心实施半双工千兆以太网？千兆以太网交换机可支持全双工。它们利用先进的 ASIC 技术来实现交换功能结果比以往任何时候都更便宜。对于千兆

以太网首部署，主要考虑性能而不是成本。半双工千兆以太网已经证明在市场上是失败的，因为目前市场上仅存在全双工千兆以太网产品。

万兆以太网

就像摩尔定律那样，每18个月就处理器的处理速度增加一倍，以太网的速度在初期也以指数级增长。万兆以太网标准由2002年成立的IEEE 802.3ae工作组负责。后来于2006年扩展到运行于10GBASE-T的双绞线上。万兆以太网具有以下特点：

全双工：IEEE 802.3开发人员吸取了千兆以太网发展的教训。在万兆以太网中只有全双工模式，根本不考虑半双工模式。

与以往标准兼容：帧格式和MAC操作保持不变，使得与现有产品的互操作性也比较容易实现。

走向广域网市场：既然千兆以太网已经走向城域网市场，万兆以太网还将进一步进入广域网市场。一方面，在新标准中最长的传输距离是40km；另一方面，定义了广域可PHY与同步光纤网络（SONET）基础设施中的OC-192（OC：光载波）的接口，其运行速度非常接近万兆位。在IEEE 802.3ae标准中，除了具有LAN PHY外，还带有一个可选的WAN PHY。两个PHY具有相同的传输介质，到此具有相同的传输距离。不同的是，WAN PHY在物理编码子层（PCS）中有一个广域网接口子层（WIS）。WIS是一个成帧器，能将以太网帧映射成一个SONET有效载荷，从而简化了以太网连接到OC-192设备的任务。

表3-5列出了IEEE 802.ae中的物理规范。在编号名中的字符“W”表示它们可以直接连接到OC-192接口的WAN PHY。其他则仅用于局域网。除了10GBASE-LX4外，其他物理规范都使用复杂的64B/66B分组编码。10GBASE-LX4使用8B/10B分组编码，并依靠4个波分复用（WDM）信道来实现10Gbps的传输速率。除了IEEE 802.3ae中首批万兆位规范外，以后的规范（如10GBASE-CX4、10GBASE-T）甚至还允许在铜线上以10Gbps等速。自2008年以来，10Gbps以太网无源光网络（EPON）扩展也在开发之中。

表3-5 IEEE 802.3ae中的物理规范

编号名字	波长	传输距离（m）	编号名字	波长	传输距离（m）
10GBASE-LX4	1310nm	300	10GBASE-SW	850nm	300
10GBASE-SR	850nm	300	10GBASE-LW	1310nm	10 000
10GBASE-LR	1310nm	10 000	10GBASE-EW	1550nm	40 000
10GBASE-ER	1550nm	10 000			

历史演变：骨干网络：SONET/SDH和MPLS

SONET和SDH是基于光纤的多路复用协议。SONET代表同步光网络，而SDH代表同步数字系列。前者用于美国和加拿大，而后者用于世界其他地区。SONET最初采用OC-x来指示，线路速率是大约为51.8*xMbps。因此，OC-3线路速率大约为155Mbps，而OC-12大约为622Mbps，以此类推。高速SONET/SDH，如大约10Gbps的OC-192，通常部署在骨干网上。

由于SONET/SDH的大型基础设施，很难很快地用以太网进行取代。这就是为什么万兆以太网需要支持所谓的广域网PHY的原因。因为它可以直接连接到OC-192接口上。因此，让万兆以太网与现有SONET/SDH基础设施共存是可行的。

为了在如此高速的网络转发数据包，多协议标签交换（MPLS）允许路由器利用标签来标记数据包，核心路由器仅检查标签就可以转发数据包。这种机制比普通路由器使用的最长IP前缀匹配快很多。这一点我们将在第4章中介绍。

第一公里以太网

我们看到以太网在有线局域网中占据主导地位。长期待它有朝一日能接管广域网。但局域网和广域网之间如何接口呢？即使局域网和广域网上有丰富的带宽，但你可能仍然在家里通过ADSL、电缆调制解调器等访问互联网。局域网和广域网之间的用户接入网络的网段，业界称为“第一公里”或“最后

一公里”，有可能成为端到端连接的瓶颈。由于在局域网第一公里和广域网中使用不同技术导致的协议转换，从而导致了不可忽视的开销。随着越来越多的用户接入网络，这种潜在的市场已经引起以太网开发者的高度关注。

IEEE 802.3ah 以太网第一公里（EFM）工作组努力为这种市场需求定义了一个标准。如果以太网在有线网络中无处不在，那么就不需要进行协议转换，这也将减少整体开销。总之，该标准预计将为潜在的、广阔的第一公里市场提供廉价和更快的技术。以太网有望成为无处不在的，标准的目标包括以下内容。

新的拓扑结构：用户接入网络的需求包括点到点光纤、单点到多点光纤和点到点铜导线。标准应符合以下要求。

新的 PHY：表 3-6 总结了在 IEEE 802.3ah 标准中的 PHY，包括如下规范。

表 3-6　IEEE 802.3ah 中的物理规范

编号名字	描　述
100BASE-LX10	在一对光纤上以 100Mbps 传输 10km 以上
100BASE-BX10	在一根光纤上以 100Mbps 传输 10km 以上
1000BASE-LX10	在一对光纤上以 1000Mbps 传输 10km 以上
1000BASE-BX10	在一根光纤上以 1000Mbps 传输 10km 以上
1000BASE-PX10	在无光源光纤网络上以 1000Mbps 传输 10km 以上
1000BASE-PX20	在无光源光纤网络上以 1000Mbps 传输 20km 以上
2BASE-TL	至少以 2Mbps 经过 SHDSL 传输 2700m 以上
10PASS-TS	至少 10Mbps 经过 VDSL 传输 750m 以上

点到点光纤：物理层是从一点到另外一点的单模光纤。它们包括 100BASE-LX10、100BASE-BX10、1000BASE-LX10、1000BASE-BX10，其中 LX 表示一对光纤而 BX 表示单根光纤。这里，10 指的是传输距离为 10km，比 IEEE 802.3z 千兆以太网的最大距离 5km 还长。

单点到多点光纤：在这种拓扑结构中，单点服务用于多个选择。其中之一是无源光分路器，因此这样的拓扑结构也称为无源光纤网络（PON）。PHY 包括 1000BASE-PX10 和 1000BASE-PX20。前者可以传输 10km，而后者可以传输 20km 以上。另一个努力是将以太网 PON 推至高达 10Gbps 的传输率也正在 IEEE 802.3av 标准中制定。Zheng 与 Mouftah 于 2005 年对以太网 PON 介质访问控制进行了总结。

点对点铜导线：物理层使用非负载、语音级别电缆。物理层包括 2BASE-TL 和 10PASS-TS。前者至少以 2Mbps 经过 SHDSL 传输高达 2700m，在后者至少以 10Mbps 经过 VDSL 传输高达 750m。如果不能提供光纤时，它们就是更经济的解决方案。

远端操作、管理和维护（OAM）：可靠性对用户接入网络很关键。为了方便 OAM，标准为远程故障指示、远端回环和链路监测定义了新的方法。

历史演变：第一公里网络互联：xDSL 和电缆调制解调器

各种数字用户线（DSL）技术能够在老的电话线上传输数据。由于电话线无处不在，所以 DSL 技术当然也就很受欢迎。在 xDSL 中的字母“x”指的是在 DSL 技术中的类型，包括 ADSL 表示非对称 DSL、vDSL 表示超高速 DSL、SHDSL 表示对称高速 DSL 等。由于 xDSL 的普及，即使 EFM 中的点到点铜线规也在其物理层中利用 SHDSL 和 VDSL 技术，而在链路层中保留以太网侧。

在 DSL 技术的类型中，ADSL 是最流行的一种。ADSL 为上行和下行提供不同速度。下行速度高达 24Mbps，上行速度最高可达 3.5Mbps，具体取决于 ADSL 调制解调器与本地电话局的距离。由于光纤入户（FTTH）的成本很高，所以 xDSL 对于光纤到楼栋（FTTB）应用也是很受欢迎的。光纤可以到达离家最近的街道配电柜，从这里开始就可以部署 vDSL。由于铜线的应用距离短，所以速度可以非常

度，最新的xDSL2标准可高达100Mbps。

与通过电话线传输数据的xDSL相比，电缆调制解调器是基于通过电缆服务接口规范（DOCSIS）传输数据，这是有关有线电视系统数据传输的标准规定。上行和下行传输吞吐量大约是30～40Mbps。虽然由于其传输介质是有线电视电缆，所以电缆调制解调器有更大的总体带宽和更远的传输距离，但带宽是在有线电视订户之间共享的。相比之下，xDSL用户在放后一公里接入网络具有专用的带宽。目前两者是仍然处于竞争之中的技术。

3.4 无线链路

无线链路非常具有吸引力，因为用户可以不受导线的阻离限制，而这可能是由于导线部署不方便或很显贵所造成的。然而，无线链路与有线链路的功能特点不同，对协议设计有着特殊的需求。我们将这些特点列出如下。

不可靠：信号在空气中传播，没有任何保护，使得传输容易受到干扰、路径损耗或多径交叉的损害。外界干扰是来自附近的无线信号源。橡皮炉和蓝牙设备对于无线链路就是可能的噪声源，因为它们都工作在免许可证的ISM（工业、科学和医疗）频段。路径损耗是信号在空气中传播时经历的衰减。衰减比有线中的更严重，因为信号散布到空中，而不是集中在有线链路上。多径失真来自信号的远离部分，因为它们从障碍物上反射从而通过不同的路径到达接收器。

更具移动性：因为没有线路限制站点的移动性，所以无线网络的网络拓扑结构可能会动态地变化。请注意移动和无线是不同的概念，虽然它们常常相提并论。无线并非是移动所必需的。例如，一个移动站点可以携带到某个位置，然后再插入有线网络上。移动性也不是无线所必需的。例如，两个高层建筑物可以利用固定无线中继设备进行通信，因为它们之间的布线太贵了。这个例子是网络部署中很常见的。

更少的电能可用性：移动站点通常由电池供电，它们有时可能会进入睡眠模式以节省电能。如果接收机处在睡眠模式下，发射器将缓冲数据直到接收器被唤醒接收数据为止。

更少的安全性：传输范围内的所有站点都可以很容易地窃听在空中传播的数据。可选的加密和认证机制可以保证数据的安全不受来自外部的威胁。

在本节中，我们选择IEEE 802.11无线局域网、蓝牙和WiMAX作为例子来介绍无线链路。我们选择这三者，因为IEEE 802.11毫无疑问在无线局域网中占据着主导地位，蓝牙主宰着无线个域网，WiMAX将成为最流行的城域网。因为它们具有的优势和重要性，所以它们可以代表无线链路技术。

3.4.1 IEEE 802.11 无线局域网

无线局域网（WLAN）的演变

IEEE 802.11工作组成立于1990年，为无线局域网制定了MAC和PHY规范。由于开发过程花费了太长时间，以至于标准的第1版直到1997年才出现。起初，有3种PHY，即红外线、直接序列扩频技术（DSSS）和跳频扩频（FHSS），规定以1Mbps和2Mbps的速度传输。扩频技术的目的是使信号在受到干扰时生命健壮。1999年提出了两种修正案，分别是802.11a和802.11b。IEEE 802.11将DSSS系统扩展到更高的数据传输速率，分别为5.5Mbps和11Mbps。IEEE 802.11a指定了新的工作在5GHz频段的正交的分多路复用（OFDM），而不是以前标准所用的2.4GHz频段。数据传输速率显著增加到54Mbps。然而，这两个标准互不兼容。运行在11Mbps的IEEE 802.11b产品已经在市场上流行起来。使用OFDM的802.11g标准也运行在54Mbps，并通过使用其调制技术向后兼容802.11b。IEEE 802.11n标准，使用MIMO-OFDM，能够工作在高达300Mbps的速率。带有多个发射器与接收机的OFDM参见第2章中所述。

除了无线局域网的速度不断增加外，IEEE 802.11还增强了其他功能。IEEE 802.11e为对时间有严格要求的应用定义了一套服务质量（QoS）功能。IEEE 802.11i为安全指定了增强机制，因为在最初的802.11标准中的有线等效保密（WEP）被证明是不安全的。正在开发中的一些标准也有趣

IEEE 802.11x 定义了在 ad hoc（自组织）模式的设备如何创建一个网格网络，IEEE 802.11k 和 IEEE 802.11r 用于无线漫游。前者提供信息以找到最合适的接入点，而后者允许运动中设备的连接和快速切换。

构建模块

802.11 无线局域网的基本构建模块是一种基本服务集（BSS）。BSS 是由使用了符合 IEEE 802.11 标准的 MAC 和 PHY 的站点组成。一个单独的 BSS 称为一个独立的 BSS（IBSS），或者常称为 ad hoc（自组织）网络，因为它经常是在没有优先规划下形成的。最小的 BSS 仅包含两个站点。多 BSS 可以通过分布式系统（DS）连接。IEEE 802.11 标准不强制实现 DS 应该实现的内容，但以太网就是一种常用的 DS。一个 DS 和一个 BSS 通过接入点（AP）连接起来，这个扩展的网络结构称为一种基础设施。图 3-23 显示了无线局域网中的构建模块。图 3-24 描绘了在 IEEE 802.11 中的层次结构。IEEE 802.11 PHY 由红外线、DSSS、FHSS 和 OFDM 组成，如第 2 章所述。在它们之上是 MAC 子层。在本节中，我们将重点介绍 IEEE 802.11。有关 PHY 的问题，我们鼓励有兴趣的读者参考“进一步阅读”中的内容。

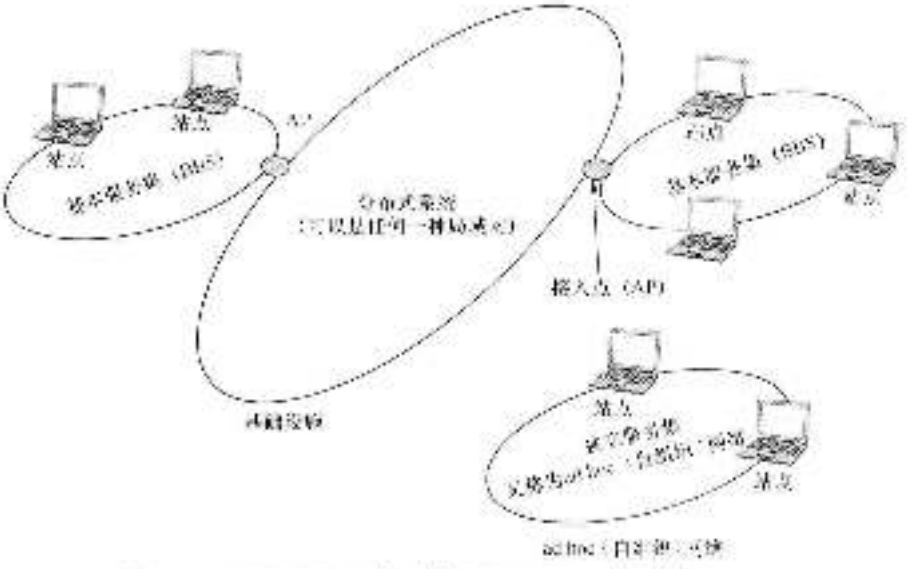


图 3-23 无线局域网中的 IEEE 802.11 构建模块

802.2 LLC				数据链路层
802.11 MAC				
FHSS	DSSS	IR	OFDM	物理层

FHSS：跳频扩频
DSSS：直接序列扩频
OFDM：正交频分多路复用
IR：红外线

图 3-24 IEEE 802.11 中的层次结构

CSMA/CA

IEEE 802.11 的 MAC 利用两个主要功能分配带宽：分布式协调功能（DCF）和点协调功能（PCF）。DCF 在 802.11 中是强制执行的。PCF 仅在基础设施网络中实现。两种协调功能可以同时在同一个 BSS 中运行。

DCF 思想又称为载波段所多路访问/冲突避免（CSMA/CA）。与以太网 MAC 的最明显区别在于冲

突避免。对于 CSMA/CD 来讲，站点必须在发送前进行监听。如果有一个站点正在传输，那么其他站点将被推迟直到信道空为止。一旦信道为空，站点将等待很短时间，称为被闭空间（IFS），它与以太网中的被闭间隙（IFG）相同。在上次传输的时间内，很有可能存在多个站点同时等待发送数据的情况。如果在 IFS 后它们都获准传输数据，那么就很可能导致冲突。为了避免可能的冲突，站点在再次传输前必须等待一个以时隙为单位的随机回退时间。回退时间是 0～CW 内随机选择的。CW 表示此争竞口，取值范围为 CWmin～CWmax。CWmin、CWmax 以及时隙都取决于 PHY 的特性。最初，将 CW 设置为 CWmin。如果信道空闲一个 IFS，那么回退时间就减少一个时隙；否则，回退时间固定直到信道空闲为止。当回退时间最终为零时，站点就开始传输。当帧被成功接收时，接收机便发送回一个确认给发送者。确认是必需的，以便确认帧是否在接收方发生了冲突。行动原则："为什么在 WLAN 中不使用 CSMA/CD？"对此有重要的说明。图 3-25 总结了 CSMA/CA 过程。除了确认外，接收过程类似于 CSMA/CD。

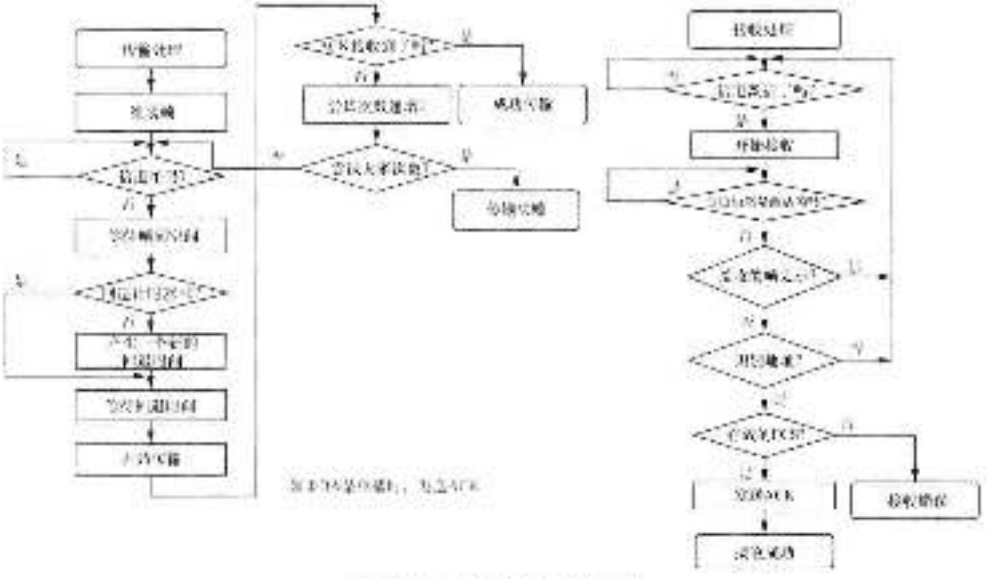


图 3-25 CSMA/CA 流程图

行动原则：为什么在 WLAN 中不使用 CSMA/CD

IEEE 802.11 MAC 和 IEEE 802.3 MAC 之间的明显区别在于，在无线局域网中的冲突检测是难以实现的。全双工无线电频率（RF）实现成本很高，潜在的隐藏站点会使冲突检测失败。后者又称为隐藏终端问题，如图 3-26 所示。站点 A 和站点 C 都无法侦测到对方的存在，因为它们位于彼此的传输范围之外。如果它们同时向站点 B 传输数据，冲突将在站点 B 上发生但不能被站点 A 和站点 C 检测到。与

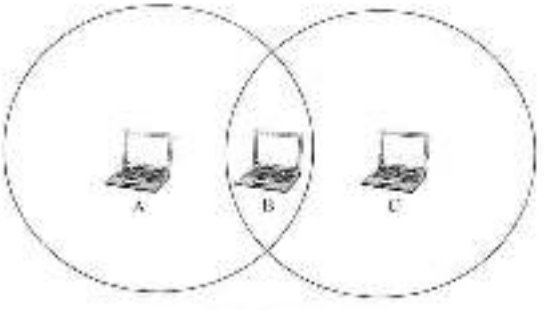


图 3-26 隐藏站点问题

以太网中如果检测到冲突便立即停止发送冲突检测不同，发送方没有办法知道传输中的帧是否已经被

环，才到传输完成后重新方收还有报到确认为止。否则，如果传输失败，冲突的成本是很大的。另一方面，如果成功接收到了 RTS 是正确的，那么接收方应该肯确认回答。但这大可就没有这种确认的必要。

RTS/CTS：首先进行清理

一个减少冲突成本的可选改进方法，是一种基于的 RTS/CTS 机制，如图 3-27 所示。在传输发之前，发送方（站点 A）通知目标接收方（站点 B）另一个小请求发送（RTS）。RTS 很容易遭遇冲突，因其传输会很小。接收方用一个小请求发送（CTS）帧作为响应，这也会通知其传输范围内的所有站点（包括站点 A 和站点 D）。这两个帧都携带了持续时间字段。RTS 中的持续时间字段是命令发送方（A 站点）后面的站点（如 C 站点）应该等待的时间，当接收器发送 CTS 后回给发送器时，在接收方（站点 B）传输范围之内的其他站点（如站点 E）将在 CTS 指定的持续时间内控制发送自己本需要物理上的载波侦听，因此 CTS 之后的帧在接收方（站点 B）就不会再有冲突了。因此，这种机制也称为虚拟载波侦听。注意只关心接收方产生的冲突，它不关心发送方的冲突。此外，RTS/CTS 机制只减少了单播帧。在组播和广播传输中，多个接收方的 CTS 将导致冲突。同样，在这两种情况下，也不会发送已经等待确认的确认帧。

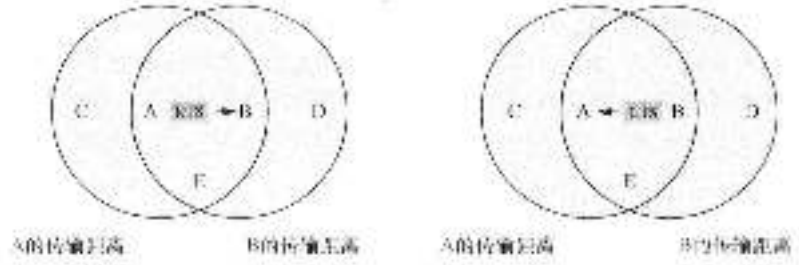


图 3-27 RTS/CTS 机制

PCF 和 DCF 交替

驻留在 AP 中的点协调器（PC）在每个 BSS 内实施 PCF。PC 定期发送信标帧，宣布无竞争阶段（CFP）。BSS 内的每个站点都利用信标帧上的 CFP 期间设置向量。PC 有权决定谁可以传输，并且仅被 PC 轮询过后站点才能传输。标准中未指定轮询顺序，具体由供应商决定。

DCF 和 PCF 可以在图 3-28 所示的场景中共存。在这个中，CFP 交织一步，由 CP（竞争阶段）是第一步。

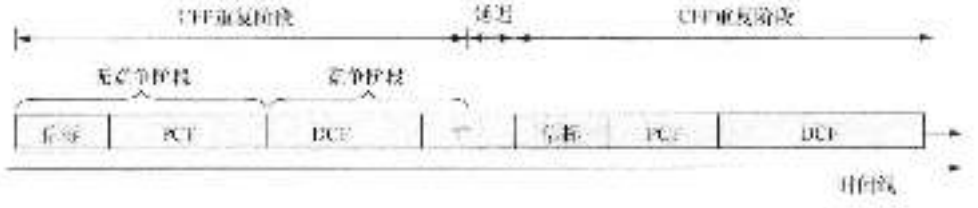


图 3-28 CDF 和 PCF 共存

1）DCF 可以实施一个 CFP，并且 BSS 进入一个称为竞争阶段（CP）的时期。

2）之后，PC 利用一个称为 CFP 重复间隔的字段发送信标帧，但是基在 CP 结束时信道恰好忙，就延迟一个 CFP 重复期间。

图 3-29 给出了使用的 IEEE 802.11 MAC 帧格式。某些帧类型可能只包含这些字段中的一个子集。四个地址字段可以包含源地址、目的地址、发送方地址（在无线桥接中，从接入点到一个无线站点）、接收方地址（到无线到另外一个接口的接入点）。三个地址是可选的，用于与接入点桥接。我们将帧分成三种类型。

1）控制帧：RTS、CTS、ACK 等。

2）数据帧：正常数据。

3）管理帧：信标等。

通用帧格式

	帧控制	持续时间/ID	地址1	地址2	地址3	序列控制	地址4	帧主体	FCS
字节	2	2	6	6	6	2	6	0-2312	4

图3-29 通用IEEE 802.11帧格式

要完全掌握这些类型，需要深刻理解每种IEEE 802.11的操作。除了四种地址外，帧控制字段指定帧的类型和与帧相关的一些信息。持续时间/ID字段指定介质或站点所属的BSS标识管理帧的忙碌间阶段。顺序控制字段指定帧的序列号以避免重复帧。由于使用的格式是复杂的，并且要取决于帧的类型，所以想要了解详情的读者可以参考IEEE 802.11标准。

开源实现3.6：用NS-2模拟IEEE 802.11 MAC

概述

与CSMA/CD不同，直到目前为止也没有CSMA/CA的开源硬件实现。因此，我们引进一种流行的开源模拟器NS-2来模拟一个802.11 MAC。NS-2是一种用于网络研究的离散事件模拟器，并对模拟通过有线和无线网络的TCP、路由、多播协议提供大力支持。在基于事件的模拟器中，网络中的所有活动都以带有时间戳的事件统计地产生，其中由事件调度器调度发生。许多研究人员利用NS-2评估他们在早期设计阶段的协议。最近NS-2已经广泛地用于模拟802.11网络的行为。

框图

图3-30给出了NS-2中的802.11 MAC和PHY的体系结构，它由几个网络模块组成。为了简单，可以将它们分为以下三个主要层：

1）层2有三个子层。首先是链路层对象，这是传统局域网中的逻辑链路控制（LLC）。链路层对象与将在第4章中介绍的地址解析协议（ARP）一起工作。其次是接口队列，为动态源路由协议（DSR）等的路由协议消息分配优先级。最后是802.11 MAC层，它处理所有RTS/CTS/DATA/ACK单播帧和所有数据（DATA）的广播帧。CSMA/CA就在这一层实现。

2）层1是802.11 PHY，一个可以基于直接序列扩频设置参数的网络接口。该部参数包括天线类型、能源模型、无线电传播模型。

3）层0是信道层。模拟用于无线通信的物理空气介质。信道层从无线节点向他则范围内的其他节点发送帧，并将帧复制到层1。

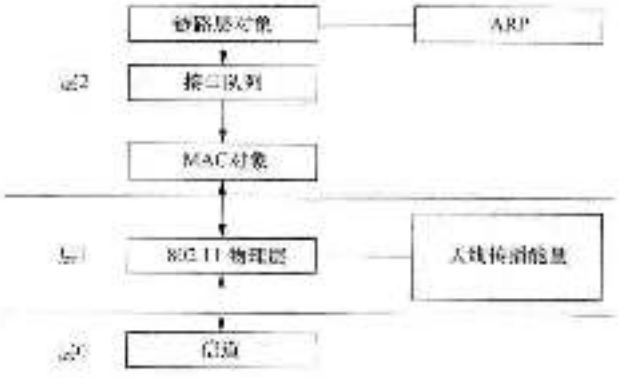


图3-30 NS-2 802.11 MAC和PHY的体系结构

数据结构

在本设计中最重要的数据结构是一组定时器，包括发送定时器、回退定时器、接收定时器和定时定时器等，如图3-31所示。以下部分将通过描述这些定时器与函数调用之间的交互作用，详细阐述

802.11 MAC 和 PHY 的操作。



图 3.31 802.11 MAC 在 NS-2 源代码

算法实现

802.11 MAC 在 NS-2 中的实现

802.11 MAC 是 MAC 层的一个子类，与之有关的源代码为 mac-802_11.cc、mac-802_11.h、mac-timer.cc 和 mac-timer.h。为了更好地理解 NS-2 MAC 源代码，在图 3.31 中列出了多个入口点，并描述各个相关函数的调用序列。由于 NS-2 是基于事件的，所以当指定的事件被触发时，除了 recv() 函数外，send()、timer()、deferHandler()、recv_timer() 和 backoffHandler() 也作为入口点。至于 802.11 MAC 的接收和传输，recv() 函数既处理从自动机层的数也处理来自上层的数。另一个 send() 函数是传输流的一个入口点，但它由 recv() 函数调用用于处理外出包。

下面是各入口点的详细解释。

- send_timer() 用于处理来自其他移动节点的握手数据包，并指定时器过期时被调用。根据发送帧的下四类型，对定时器的处理有着不同的解释。例如，如果最后发送的帧是 RTS，过期就意味着 CTS 没有收到，这可能是因为 RTS 发生了冲突，也可能因为接收节点处理繁忙。MAC 通过把参数 ssrc 增加 1 来记录传输 RTS 作为尝试。如果最后一次发送数据帧，过期就意味着没有收到 ACK，MAC 层会调用 helper 函数 sendDATA() 来处理这种情况。定时器发生处理完后，就需要各层的支持，将控制返回给 tx_resume() 函数。当最后一帧为 CTS 或 ACK 时，send_timer 将会直接调用 tx_resume()，无需进一步支持。经过 tx_resume() 处理后，如果帧序号，则对定时器就会以退避后的竞争窗口启动。
- recv() 处理无论是从物理层还是从上层传入的帧。当有帧要发送时，recv() 调用 send()。此外，send() 调用 sendDATA() 和 sendRTS() 为数据帧和 RTS 帧构建 MAC 头部。如果 recv() 没有接收帧，那么进入的数据就将通过 recv_timer() 开启动接收定时器。
- backoffHandler() 类似回退定时器过期时被调用的一个事件服务例程。当休眠节点，回退定时器当于暂停传输。调用 backoffHandler() 后，函数 check_pktRTS() 检查是否有 RTS 要等待发送。如果没有未处理的 RTS 帧，那么当定时器过期时将传输 RTS 或者数据帧，具体取决于是否启用了 RTS/CTS 机制。
- recv_timer() 是接收定时器处理例程，它将检查接收帧的类型和子类型。当接收定时器过期时，调用接收定时器处理函数。定时器过期意味着一个正在被接收并且准备进一步的处理 MAC recv_timer() 的决策取决于接收到的帧类型。如果它是 MAC_Type_Management，那么直接会被丢弃。如果接收到了 RTS、CTS、ACK 或 DATA 帧，那么将分别调用 recvRTS()、recvCTS()、recvACK() 或 recvDATA()。模拟设备处理后，流将控制类给 rx_resume()。
- deferHandler() 也是一个事件服务例程，并且当发送定时器到期时被调用。Defer 推迟定时

然等于延迟时间加上回返时间，这保证无线节点在传输前等待足够的时间，以减少冲突的机会。调用例程后，检查函数调用 check_pktRTS()、check_pktTx()和 check_pktCTRL()准备一次新的传输。如果其中任何一个 check 函数返回一个零值，那么 check 函数就成功地传送了一个帧，因为 Defer 推迟处理例程就结束了。对于 RTS 和控制帧，传输过程也可能启动接收定时器和发送定时器以使从另一个移动节点接收确认帧。

CSMA/CA 操作

CSMA/CA 操作是在 send()函数中进行的。图 3-32 显示了它的代码，这里 mhBackoff_.busy() == 0 意味着回退定时器不忙，is_idle() == 1 表示无线信道空闲，mhDefer_.busy() == 0 表示延迟定时器不忙。如果无线信道空闲并且回退和延迟定时器都不忙，那么 send()函数将进行一次延迟操作；否则经过等待而不重新设置定时器。如果它进行一次延迟操作，发送帧就需要延迟一个 DIFS 时间加上一个随机时间，即 phymib_.getDIFS() + rTime。随机时间是从（Random::random()% cw_) × (phymib_.getSlotTime()）计算出来的，间隔为 0 - cw_ 值，其中 cw_为当前的竞争窗口。如果回退定时器不忙，但无线信道并不空闲，这就意味着检测到 PHY 介质中，节点通过调用 mhBackoff_.start(cw_, is_idle())启动回退定时器。

```
void send(Packet *p, Handler *h) {
…
if(mhBackoff_.busy() == 0) {
    if(is_idle()) {
    if (mhDefer_.busy() == 0) {
        rTime = (Random::random() % cw_)*
        (phymib_.getSlotTime());
        mhDefer_.start(phymib_.getDIFS() + rTime);
    }
} else {
        mhBackoff_.start(cw_, is_idle());
    }
  }
}
```

图 3-32 在 send()函数中的 CSMA/CA 操作

Tcl 脚本模拟

NS-2 模拟可以通过定义模拟场景的 Tcl 脚本文件开始。Tcl 脚本包含网络拓扑定义、无线节点配置、节点坐标、运动场景和数据包跟踪。

图 3-33 描绘了一个由两个移动节点（节点 0 和节点 1）组成的自组织网络的一个简单场景。移动节点的移动区域在 500m×500m 范围内。为了 FTP 服务，还建立了一条 TCP 连接。表 3-7 在 wireless.tcl 脚本文件中描述了详细的场景，定义了如图 3-33 所示的例子。

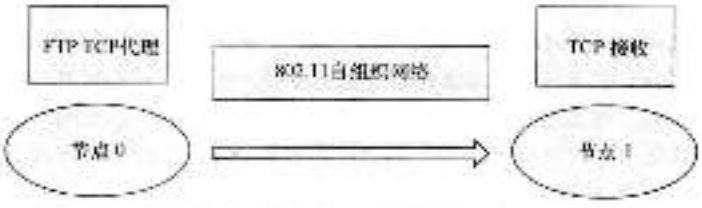


图 3-33 两个移动节点使用 TCP 和 FTP 的一个 NS-2 例子

练习

1. 为什么 send()函数从 recv()中调用？
2. 为什么发送帧要随机等待一段时间？

表 3-7 图 3-33 的 NS-2 Tcl 脚本

描 述	wireless.tcl 的主要代码	
定义选项：信道类型、无线电传播模型等	set val(chan) Channel/WirelessChannel ; set val(prop) Propagation/TwoRayGround ; set val(netif) Phy/WirelessPhy ; …	#信道类型 #无线电传播模型 #网络接口类型
创建一种模拟、跟踪和拓扑	set ns_ [new Simulator] set tracefd [open simple.tr w] … set topo [new Topography] $topo load_flatgrid 500 500	#创建一个模拟对象 #定义一个追踪文件，记录所有的帧 #创建一个拓扑 #设置 500m×500m 的拓扑范围
建立信道并配置 MAC 节点	create-god $val(nn) set chan_1 [new $val(chan)] $ns_node-config -adhocRouting $val(rp) \\ …	#创建 God #配置节点 #为 node-(Type $val(ll) \\ 设置参数
为 802.11 PHY 设置参数	Phy/WirelessPhy set Pt_0.031622777 Phy/WirelessPhy set bandwidth_11Mb…	
紧上箱以移动	for {set i 0 [$i < $val(nn) [incr i]] set node_($i) [$ns_node_ $node_($i) random-motion 0]	
建立并初始化两个无线节点坐标（X、Y、Z）	$node_(0) set X_10.0 … $ns_initial_node_pos $node_(0) 10 $ns_initial_node_pos $node_(1) 10	#节点 0 坐标设置在（10.0，20.0，0.0）
在两个节点之间建立 TCP 和 FTP 流	set tcp [new Agent/TCP/Sack1] … $ftp attach-agent $tcp	#创建一个 TCP 连接
开始模拟	$ns_at 1.0 "$ftp start" … $ns_run	#在 1.0 s，开始传输

3.4.2 蓝牙技术

除了计算机后面大量的电缆连接到计算机外围设备之外，还需要更多的电缆连接到不同类型的通信和网络设备。这些电缆是如此地繁琐累赘，为了方便，最好能够摆脱它们。蓝牙，根据 10 世纪丹麦国王的名字命名，恰好是支持短距离（通常在 10m）无线电链路以取代用电缆连接电子设备的技术。1998 年，五大公司分别是，爱立信、诺基亚、IBM、东芝和英特尔合作开发了蓝牙技术。为了确保蓝牙的普及，开发目标是将许多功能集成到单芯片以降低成本。稍后成立了由许多公司组成的蓝牙射 81 兴趣小组（Bluetooth SIG），以促进和定义新的标准。

蓝牙设备工作在 2.4GHz ISM 频段，与大多数 IEEE 802.11 设备使用的跳频方式相同。频带范围是 2.400～2.4835GHz，其中 791MHz 频率用于跳频以避免来自其他信号的干扰。之下和之上的信道分别是 2MHz 和 2.5MHz 的防护频带。细心的读者可能会立即注意到，当 IEEE 802.11 与蓝牙设备近距离操作时可能出现干扰问题。IEEE 802.11 和蓝牙设备的共存问题是一个大问题，对此我们将在本节中做更多的讨论。蓝牙因其短距离而被划入无线个域网（无线 PAN）。

在微微网和扩散网中的主设备和从设备

图 3-34 显示了基本的蓝牙拓扑结构。与 802.11 中的 BSS 一样，多台设备共享同一个信道形成一

个微微网（piconet）。与IBSS不同的是，在微微网中的所有站点都是同等的，一个微微网只能由一个主设备和多个从设备组成。主设备有权控制微微网中信道的接入，即决定和谁通信。从设备既可以是活动的也可以是休眠的，一个主设备同时控制最多7个从设备。休眠的从设备停止通信，但它们仍然与主设备保持同步，并根据主设备的需要可以变成活动的。如果一个主设备希望与多于7个的从设备通信，它将通知一个或多个活动的从设备进入休眠模式，然后激活目标休眠从设备变成活动的。为使更多的设备同时通信，多个微微网可以彼此重叠形成一个较大的扩散网（scatternet）。图3-34生动地说明了两个微微网利用两桥节点形成扩散网，网桥可以是两个微微网中的从设备或者一个微微网中的主设备。网桥节点以时分模式参与到两个微微网，以便它有时属于一个微微网，有时属于另一个微微网。

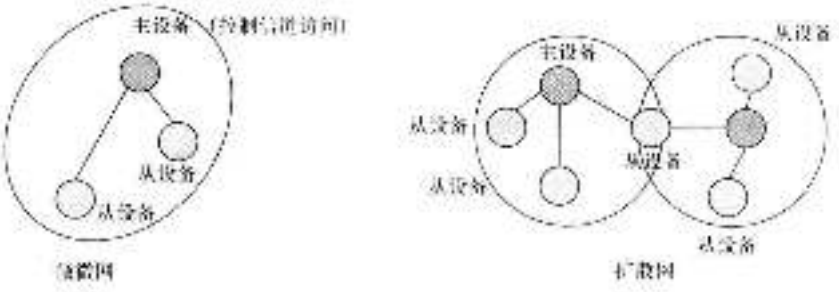


图3-34 蓝牙拓扑：微微网和扩散网

查询和寻呼过程

蓝牙设备为了通信必须彼此感知。查询过程是为了让附近的设备发现对方，随后通过寻呼过程建立一条连接。最初，在默认情况下所有蓝牙设备都处在侦听模式。有意通信的设备将尝试在其覆盖范围内广播查询。广播设备附近的其他设备如果愿意与它通信，就可能用自己的信息（如地址）对查询做出响应。一旦收到这些响应，查询者就会知道周围存在的设备，并成为微微网的主设备，而其他设备则成为微微网的从设备。查询结束后，主设备发送一个单播寻呼信息到目的设备。目的设备然后对一个确认进行响应，于是主设备和目的设备之间就建立了连接。然后，从设备可以运行相同的寻呼过程，接管微微网中的主设备。这个过程的细节，如图3-35所示。值得注意的是，多个对查询的响应会导致冲突。因此，接收设备应推迟一个随机回退时间后再做出响应。

跳频时隙

将微微网信道分成许多时隙，每个时隙都对应不同的跳频频率。一个时隙的持续时间是625微秒（μs），这是跳频率1600跳/秒的倒数。主/从设备对以相同的跳频序列非分复用在79个1MHz信道中的时隙，跳频序列是从双方都知道的一个伪随机序列中推导而来。其他的从设备与通信过程无关。当数据传输速率为1Mbps时，每个时隙在理想情况下可以传输625位数据。但是，由于时隙内的某一时间段预留用于跳频和稳定目的，所以每个时隙实际上最多可携带366位数据信息。通常，每个时隙可以承载一个蓝牙帧，它带有72位接入码的字段、54位头部信息，以及可变长度的有效载荷。显然，在理想情况下，可以承载625位的时隙内仅承载366－72－54＝240位（30字节），可见其效率非常低。为了提高效率，蓝牙帧在相同的频率下允许占据连续5个时隙，所以在5个时隙内仅浪费了625－366＝259位用于跳频控制。

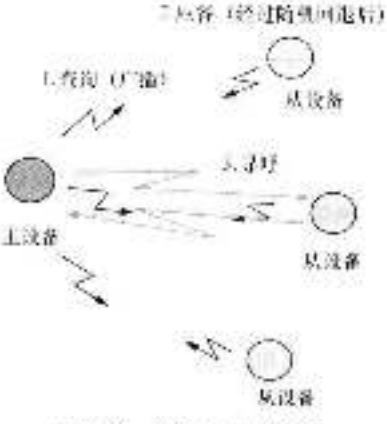


图3-35 查询和寻呼过程

交错穿插的预留时隙和分配时隙

蓝牙连接有两种使用时隙进行通信的选择。第一种是同步面向连接的链路（SCO链路），它定期保留时隙用于有时间限制的信息，如语音数据。例如，一个电话级语音的采样率为8kHz，每次采样

产生一字节；换句话说，每0.125ms产生一字节。因为在每个时隙内一帧可以携带30字节，每3.75ms（0.125ms×30）速保留一个时隙传输语音数据。每个时隙长度为625微秒，这意味着每6个（3.75ms/625ms）时隙中就要有秩留一个时隙。第二个不同步无连接的链路（ACL链路），其中时隙是按需分配的而不是保留的。主设备负责来自一个或多个从设备请求的时隙分配，以避免冲突并控制链路的服务质量（QoS）。当主站轮询时，从站允许向主站发送一个ACL帧。与无线局域网中的PCF和DCF类似，SCO和ACL时隙是交错穿插的，但是主要区别在于ACL运行了一种无冲突的轮询和时隙分配。图3-36显示SCO链路和ACL链路的时隙。SCO链路中的帧是相当有规律的，而ACL链路中的帧则是按需的。

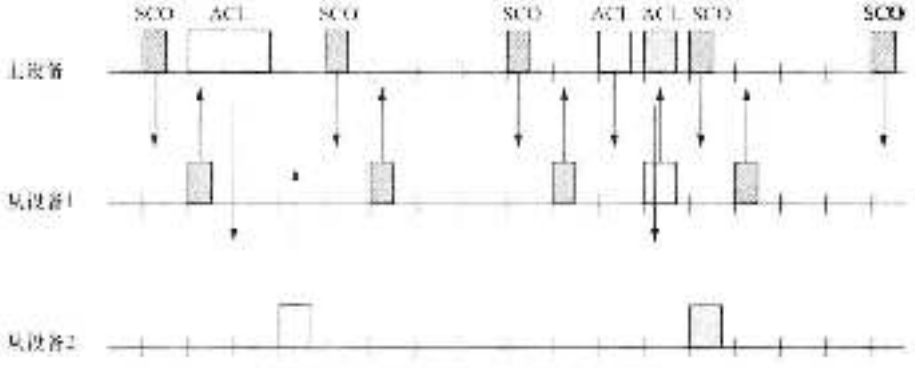


图3-36 SCO链路和ACL链路的时隙

图3-37描绘了蓝牙技术规范中的协议栈。每个软件模块的功能在图3-37中右侧描述。粗黑线以上的模块在软件中实现，其他在硬件中实现。基带和射频模块上的链路管理器协议负责蓝牙单元之间的链路建立。该协议还可以处理数据包大小和加密密钥的协商，并执行实际的加密和解密。



图3-37 蓝牙协议栈，其中基带和链路管理协议起到MAC子层的作用

逻辑链路控制和适配层（L2CAP）模块支持多路复用、分段和为高层协议重组数据包。它也支持QoS的通信。服务发现协议可以发现其他蓝牙设备上提供的服务。RFCOMM协议提供了使用蓝牙技术以替代经过电缆的串行通信。它可以通过L2CAP模拟RS-232串行端口。主机控制接口（HCI）控制为主机提供软件接口以便控制蓝牙硬件。

3.4.3 WiMAX技术

在IEEE 802.16中指定的WiMAX技术可以支持高达数公里的远距离无线通信。与IEEE 802.11中的无线局域网和在IEEE 802.15无线个域网相比，WiMAX也称为无线城域网，得名于远距离的通信范围。WiMAX设备部署既可以是固定的，也可以是移动的。IEEE 802.16-2004指定的技术用于固定连

接。IEEE 802.16-2004 的主要应用是当有线连接（如 ADSL 或电缆调制解调器）非常昂贵时使用的“第一公里”的宽带接入。IEEE 802.16e-2005 标准规定了用于移动的连接，其应用是经过移动设备的互联网接入。

具有带宽分配和调度的 MAC

WiMAX 在许多方面不同于 802.11 无线局域网。首先，它们的应用目标不同。IEEE 802.11 主要是为在家中或办公室等使用的短距离连接开发的，但 WiMAX 则是为了数公里远的宽带连接而开发的。其次，它们使用不同的介质访问控制机制。IEEE 802.11 是基于竞争的，这意味着无线设备必须竞争可用的带宽。因此，它是不适合于 VoIP 等时间敏感的应用，除非有 802.11e 标准能够提供 QoS 服务。相反，WiMAX 采用调度算法在设备之间分配带宽。在 WiMAX 中，基站为某个设备分配一个时隙，这样其他设备就不能使用该时隙。因此，基站就可以服务于大量的用户站点并为时间敏感的应用控制时隙分配。事实上，它的 MAC 类似于电缆调制解调器标准 DOCSIS，因为两者都有上行/下行的结构有利于集中式的带宽分配和调度。更多详细信息，可以从以下网站获得一个 WiMAX 网络的 NS-2 模拟模块：http://www.lrc.ic.unicamp.br/wimax_ns2。

历史演变：蓝牙与 IEEE 802.11 的比较

蓝牙和 IEEE 802.11 用于不同的目的。IEEE 802.11 的目的是无线局域网标准，而蓝牙用于无线个域网（无线 PAN，或 WPAN）。表 3-8 总结了 IEEE 802.11 和蓝牙之间的比较。IEEE 802.15 WPAN 工作组和蓝牙 SIG 合作以促进蓝牙标准。IEEE 802.15 工作组 2 侧重于解决由于可能的干扰而引起的共存问题，因此预计这两种标准可以实现共存。

表 3-8 蓝牙与 802.11 的比较

	IEEE 802.11	蓝 牙
频率	2.4GHz 的（802.11，802.11） 5GHz（802.11a）	2.4GHz
数据速率	1，2Mbps（802.11） 5.5，11Mbps（802.11） 54 Mbps（802.11）	1～3Mbps（建议将来可达 53～480Mbps）
范围	约 100m	1～100m，根据功率等级
功耗	较高（1W，通常 30～100mW）	较低（1mW、100mW，通常约 1mW）
PHY 调制	红外线，OFDM，FHSS，DSSS	（自适应）FHSS
MAC	DCF，PCF	时隙分配
价格	较高	较低
主要应用	无线局域网	短距离连接

从 OFDM 到 OFDMA

与使用免许可证 ISM 频带的 802.11 不同，WiMAX 在物理层中使用更宽的 2～11GHz 和 10～66GHz 需要许可证的频谱。最初版本的 WiMAX 工作在 10～66 GHz。工作在如此高频率使它具有提供更高可用带宽的优势，但信号也很容易受到障碍物的影响。因此，WiMAX 需要以较高的成本部署大量基站来规避障碍物。更高版本的 WiMAX 支持 2～11GHz 的频率，其中有些频带需要许可证而另一些是免许可证的。由于较低的频率，部署也变得更容易而且更便宜。为了让 WiMAX 设备能够避免与运行在相同频率范围内的其他技术设备产生干扰，该标准提供动态频率选择方案。此外，WiMAX 支持网状模式使用户站点可以从另一个站点获得数据。网状模式可以简化 WiMAX 的部署，因为可以在基站与另一个用户之间通信出现障碍的地方将用户站点部署为中继站。WiMAX 在其物理层支持 OFDM 和一个新的称为 OFDMA（正交频分多址）的方案，它为了多址访问将子载波分配给多个用户。使用 OFDMA，多个用户就可以同时在不同的子载波上访问信道；对于 WLAN，情况就不同，它使用 CSMA/CA 进行介质

访问。

OFDMA 在时间域中提供的资源是以有多种形式管理的，而在频域中是以子载波和进一步划分的子信道来管理的。与在频域中等间分区一的子信道相比，子载波是载波更细粒度的单位。最小的频率时间资源单位是一个包含 48 个数据子载波和特殊的个导频用于下载波三个符号用于上传（在强制 PUSC（部分使用子信道）模式下）的时隙。802.16 PHY 支持无分双工（TDD）、频分双工（FDD）和半双工 FDD 模式——虽然在概念上独立于 OFDMA，但它们都可以与 OFDMA 一起工作。TDD 是 WiMAX 的首选，因为它仅需要一个信道支持上联并能调整不平衡的下行链路/上行链路负载。相反，FDD 需要的两个信道分别是 DL 和 UL。收发器的设计在 TDD 中比在 FDD 中更容易。

注意，WiMAX 在 IEEE 802.16e-2005 标准中还支持移动操作。该标准支持切换和漫游达 75mile/h 的速度。此标准工作在较低的频率（2.3～2.5GHz），能让移动设备在周围移动，即使在设备和基站之间存在障碍也是如此。移动设备需要 OFDMA 以便更细致地利用子信道并减少干扰。移动应用 WiMAX 与流行的 3G 及其下一代 3GPP 竞争，但许多因素导致该域比卖日前还不清楚。尽管到目前为止 3G 已经遍布世界各地，但 WiMAX 具有更高的数据速率，即最达 75Mbps，其基站可以覆盖半径为 30 公里的区域。目前大多数笔记本计算机既没有配备 WiMAX 也没有配置 3G 上网，所以这将是第一个让 WiMAX 流行的潜在市场。

IEEE 802.16e 标准支持软、硬切换。硬切换会使用户某一时间仅停留在一个基站，这意味着，一个新的连接建立之前必须拆除旧的连接。硬切换简单，对于数据应用，它已经足够了。软切换，即一个新的连接可以在旧的连接拆除之前建立，这样匹交换机产生的间还会更短。相比，软切换更适合于实时上有严格要求的应用。

与基于短距离通信的 802.11 不同，WiMAX 主要是基于城域网，因此必须控制所有到/来自设备的数据，以避免出现可步问题。下一节将描述 WiMAX 在 TDD 模式下的帧结构，描述上行链路上（共连接填满子帧）的 5 个服务类，以及基站 MAC 数据包流的详细情况。

TDD 子帧

图 3-38 显示在 TDD 下的帧结构，其中包括 1）用于控制消息的 UL-MAP 和 DL-MAP；2）上行链路和下行链路数据子帧。带宽分配算法确定用于下行链路和上行链路的调度时隙，并指示在 UL-MAP 和 DL-MAP 消息中的调度安排。所有 UL-MAP/DL-MAP 和数据子帧由很多 OFDMA 时隙组成，其中一个时隙就是在上行链路由 3 个 OFDMA 符号组成的一个子信道，在下行链路由 2 个 OFDMA 符号组成的一个子信道。这种模式称为 PUSC（子信道部分使用），是 802.16 中的强制模式。

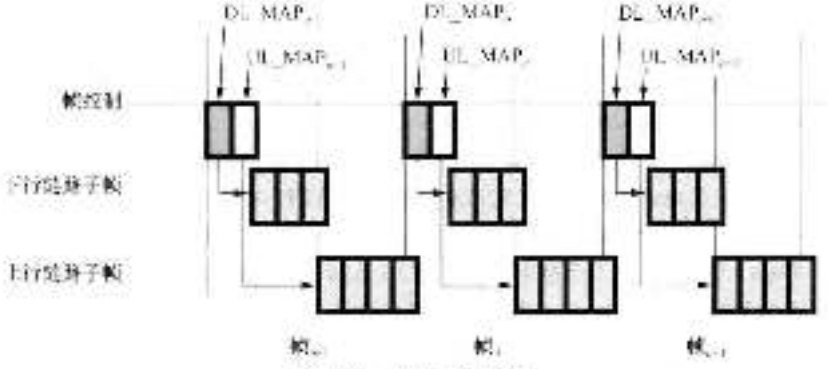


图 3-38 TDD 子帧结构

上行链路调度类

802.16e-2005 标准目前支持 5 种上行调度类，即非请求允许服务（UGS）、实时轮询服务（rtPS）、非实时轮询服务（nrtPS）、尽力而为（BE），以及最近才提出的扩展实时轮询服务（ertPS）。表 3-9 总结了这些服务类的特点，这与 DOCSIS 中的非常相似。每个服务类定义不同的数据处理机制进行服务

区分。UGS 具有最高优先级，在每个间隔为带宽保证预留了固定数量的突发。ertPS、rtPS 和 BE 依靠定期轮询从基站获得传输机会，而 nrtPS 像 UGS 一样，预留了固定数量的时隙并在竞争期间到知 BS 有关可能的更改。如果 nrtPS 和 BE 没有从轮询中获得足够的带宽，它们两者都将根据预先设定的优先级竞争传输机会。nrtPS 服务总是优先于 BE。

表 3-9 服务等级和相应的 QoS 参数

特征		UGS	ertPS	rtPS	nrtPS	BE
请求尺寸		固定的	固定但是可变的	可变的	可变的	可变的
申请间隔		N	N	Y	Y	N
竞争		N	Y	N	Y	Y
参数	最小速率	N	Y	Y	Y	N
	最大延迟	Y	Y	Y	Y	Y
	抖动	Y	Y	Y	N	N
	优先级	N	Y	Y	Y	Y
应用		不静音抑制的 VoIP	静音，带有静音抑制的 VoIP	视频，带有静音抑制的 VoIP	FTP，Web 浏览	电子邮件，基于消息的服务

MAC 层中的详细数据包流

BS MAC 上行链路和下行链路中完整的数据包流说明如下。对于下行链路处理的流，在网络层中的 **IP** 和 **ATM** 数据包都通过封装/解封装 **MAC** 头部转换成发往/到 **MAC** 会聚子层（**CS**）。根据地址和端口，将数据包分类为一个服务流的对应连接标识符，进一步决定 **QoS** 参数。然后进行分段和打包形成一个基本的 **MAC** 协议数据单元（**PDU**）。其大小要适应信道容量，随后将结果 **PDU** 分配到队列中。一旦分配开始，带宽管理单元安排数据突发传输以构建完整帧。**MAC** 生成器然后将安排（即分配结果）写入 **MAP** 消息中并当在时间槽中发送/接收到突发开始通知 **PHY** 接口。在它们发送到 **PHY** 之前，对 **PDU** 进行加密、头部校验和以及帧的 **CRC** 计算。上行链路处理流与下行链路相似，除了基站主会接收单独的或捆绑带宽请求外。在上述操作中，利用最带宽管理及带宽分配算法是至关重要的，需要精心设计以提高系统的性能。

历史演变：3G、LTE 和 WiMAX 的比较

IEEE 802.16e-2005 标准，也称为移动 WiMAX，旨在支持移动应用。正如文中提到的，WiMAX 具有高数据传输速率（75Mbps）和远传输距离（30 英里），而 3G 只有约 3Mbps 的数据传输速率。但是，3G 能够让过去使用移动电话的人成为自己的用户。

WiMAX 能否将成为移动应用的流行解决方案呢？它已经得到了多家厂商的认可。例如，英特尔公司已经将移动 WiMAX 功能集成到了基于一代芯片组的笔记本计算机的 Wi-Fi 芯片中。3G 技术也在演变——下一代 LTE（长期演进）由第三代合作伙伴计划（www.3gpp.org）开发，上行链路可以达到 300 Mbps 而下行链路可以达到 100 Mbps。假定 3G 技术已经有大规模的 3G 基础设施的它就可以迅速部署。IEEE 还在 IEEE 802.16m 标准中采用了 WiMAX 2.0，进一步提高数据传输速率，为移动用户提供 100Mbps，为固定应用提供 1Gbps。竞争是激烈的。与此同时，由于实施延迟和互操作性认证方面的原因，移动 WiMAX 部署仍然不广泛。这取决入市场的时间是关键因素，决定着移动 WiMAX 是否将在市场中取得成功。

3.5 桥接

网络管理员通常将单独的局域网连接成互联的网络以便延伸局域网或改善其管理工作。工作在链路层的互联设备称为 MAC 网桥或网桥。它通常称为第 2 层交换机，以太网交换机，或者简称交换机，稍后我们将详解其原因。网桥互联局域网就如同它存在同一个局域网内一样。IEEE 802.1D 标准规定其如何操作。下面我们将详细介绍。

几乎所有的网络都是透明网桥，因为互联局域网上的所有站点察觉不到它们的存在。发射站点只需将目的 MAC 地址封装到帧中，发送帧。如网桥自动转发该帧。另一类网桥是源路由网桥。在这种网桥中，站点应该能够发现路由，并将转发信息封装到帧中以指示网桥如何转发。由于以太网主导着局域网市场，所以目前市场上很少看到这种类型的网桥，因此这里我们仅介绍透明网桥。

网桥具有局域网连接的端口。每个端口工作在混杂模式，这意味着它能接收与它连接的局域网上的每一帧，而不管目的地址是什么。如果需要将帧转发到另一个端口上，网桥将完成相应的工作。

3.5.1 自学习

网桥之妙就在于网桥知道是否该转发到达的帧以及应该向哪个端口转发。图 3.39 说明了网桥的操作。网桥维护一张地址表，又称为转发表，表中存储 MAC 地址到端口号之间的映射。最初，地址表是空的，网桥不知道任何站点的位置。假设具有 MAC 地址为 00－32－12－12－6D－AA 的站点 1 向与 MAC 地址为 200－1C－1207－12－DD－3E 的站点 2 发送一个帧。由于站点 1 连接到网桥的端口 3 上，所以网桥将在端口 3 上接收帧。通过检查帧的源地址字段，网桥学习到 MAC 地址 00－32－12－12－6D－AA 位于端口 3 所连接的网段上，然后将学习到的知识保存在地址表中。但是，它仍然不知道目的地址为 00－1C－1207－12－DD－3E 所在的地方。为了确保目的地可以接收到帧，它就将帧广播到每一个端口上，除了发送帧的源端口外。假设稍后站点 2 向某一地方发送了一个帧。网桥将得知其地址来自端口 2 并立即将这一事实保存到地址表中。网桥发送到站点 2 的帧都将只转发到端口 2，而不需要广播。这种过程称为自学习。

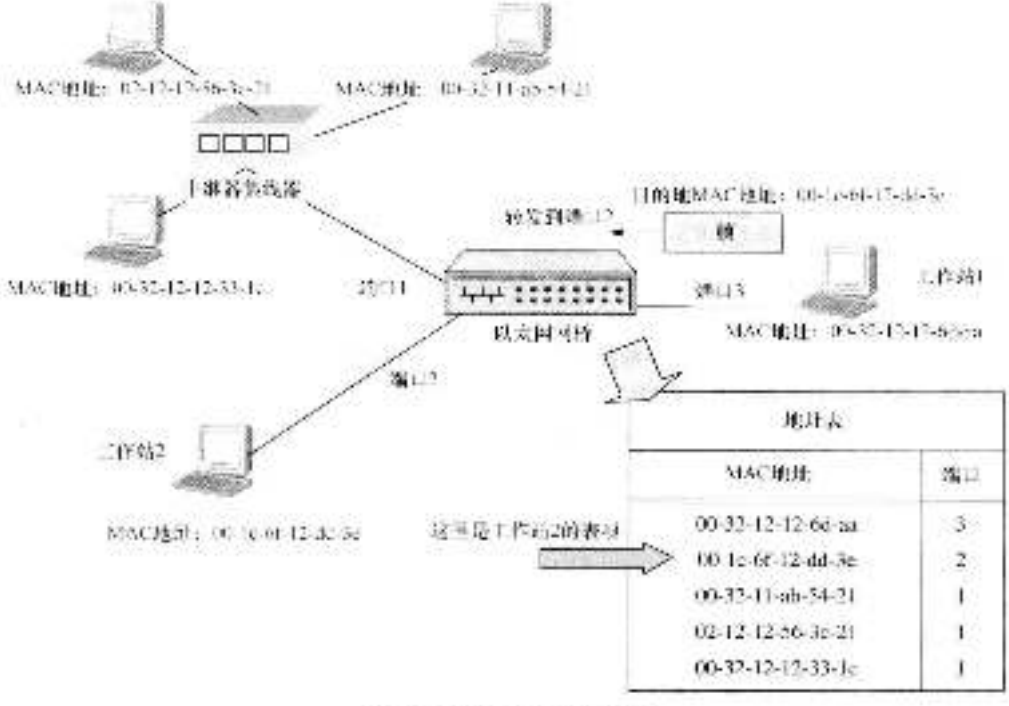


图 3.39 网桥操作：自学习

自学习极大地节省所有其他网络分段的带宽并减少冲突的概率。当然，如果站点 2 仍然保持沉默，网桥就无法知道它到底在那里，将广播发送到站点 2 的每一个帧，但这种情况很少发生。典型情况是，站点 2 在接收到一个发送给它的帧后做出响应，网桥就可以从响应中得知站点 2 在哪里。

有时某个站点会重新定位或删除，使得它在地址表中的表项老化过期。可以应用一种老化机制来解决这个问题。如果在特定的时间内一直未监听到下个站点，那么其表项将过期。随后发往该站的帧将被洪泛，直到再次学习到它的存在为止。

如果目的地址是组播或广播地址，那么网桥将向转发到所有端口上，除了来自的端口之外。但是洪泛会造成很大的浪费。为了减少不必要的洪泛成本，IEEE 802.1D 标准制定了 GMRP（组播注册协议）。GMRP 是通用属性注册协议（GARP）的一个子集。当启用该协议后，网桥就可以注册组播地址的目的接收者的需求。注册信息将在网桥之间传播以确定所有的接收者。如果在给定的路径上没有发现注册信息，那么就进行组播裁剪以切断这条路径。通过这一机制，桥接就只被转发到存在预期接收者的路径上。

注意，在图 3.39 中有一个称为中继器集线器的设备，或者简称为集线器。该设备是第 1 层设备，意味着它只是恢复信号的振幅和定时，将信号传播到除了源端口之外的所有端口上，但并不了解帧。毕竟，帧在物理层只不过是一系列物理层的编码位。

历史演变：直通与存储转发

四到一千兆的地址（DA）字段是除了前导符和 SFD 字段之外的帧中的第一个字段。查找桥表中查找 DA，网桥就能确定向哪里转发帧。网桥在完全接收帧之前就可以开始向目的端口转发帧。这种操作称为直通转发。另一方面，如果网桥只有在完全接收帧后才转发，其操作就称为存储转发。这两种方式之间的区别有其历史原因。1991 年以前，交换机无论是在 IEEE 标准中还是在市场上都称为网桥。早期的网桥以存储转发方式工作。1991 年，Kalpana（卡尔帕纳）公司销售名为“交换机”的第一台直通转发网桥以区分于其他产品的存储转发网桥，并宣布由于采用直通操作而降低了延迟。虽然在存储转发和直通转发之间如引起了很大的争论。表 3-10 总结了这两种机制。

表 3-10 存储转发和直通转发的对比

	存储转发	直通转发
发送时间	完全接收之后再发送一个帧	在完全接受一个帧之前可以发送帧[1]
延迟	较大，被认为延迟	只能具有较小一致的延迟
广播/生播	用于广播或生播帧没有问题	一般不可用于广播或生播帧
错误检测	可以及时检测 FCS	检测 FCS 可能太迟了
流行性	在市场上很常见	在市场上不太常见

[1] 如果 LAN 输出端口总输出队列短暂地处于占用，那么即使是在一直的直通转发也会变为存储转发。

网桥与交换机对比

按照 Kalpana 的命名规范，无论网桥操作是存储转发还是直通转发，它们都是以“交换机”的名字进行销售。IEEE 标准仍然采用“网桥”这个名字，并明确地表明这两个术语是同义词。大多数交换机目前仅提供存储转发，因为直通转发没有明显的优点，如表 3-10 所示。术语“交换机”也常用于根据来自上层的信息做出转发决策的设备上。这就是目前我们会看到 L3 交换机、L4 层交换机和 L7 层交换机的原因。

开源实现 3.7：自学习网桥

综述

交换机维护一个转发数据库，用以确定该如何转发给哪个端口。数据库的学习过程是自动的以便最小化管理工作。这就是我们称它为自学习的原因。自学习的主要思想很简单：如果传入帧的源 MAC 地址 A 关联端口 n，这意味着具有 MAC 地址的主机可以通过端口 n 到达，发送给 A 的帧将由交换机转发给端口 n。我们将介绍在 Linux 内核中实现自学习机制的源代码，Linux 主机也可以充当交换机（或网桥）。

框架

图 3-40 演示了自学习过程，其中转发数据库是用散列表来实现的。如果存在散列冲突，相同桶的表项会存储在一张链表上。

当带有源 MAC 地址 A 的帧进入交换机时，交换机计算 A 的散列值以确定表项在转发数据库中的位

置，并假设在该桶中找到 A（也许是历史表）。如果 A 已经在数据库中，那么原来的表项将被刷新，这意味着 A 对应的端口需要更新。最后，A 和帧的源端口将记录在转发表中。

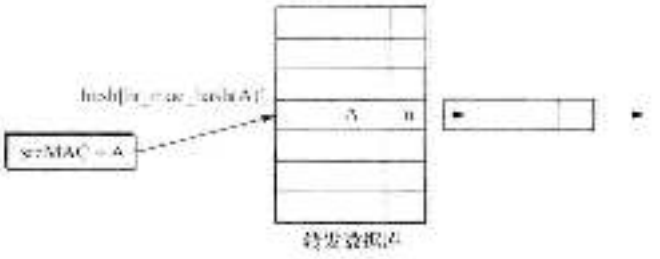


图 3-40 转发数据库的自学习过程

数据结构

最重要的数据结构就是转发数据库，它定义在 net_bridge 结构中（参见 br_private.h）。结构中的散列字段就是散列表，其定义如下：

```
struct hlist_head hash[BR_HASH_SIZE];
```

在桶表中的表项包含 MAC 地址与端口之间的关联，其定义如下。这里 mac 就是 MAC 地址，dst 是相应的交换机端口。因为一台主机可能连接到不同的端口，如果 ageing_timer 过期，该表项就应该删除在转发表中。

```
struct net_bridge_fdb_entry
{
        struct hlist_node       hlist;
struct net_bridge_port          *dst;
struct rcu_head                 rcu;
atomic_t                        use_count;
unsigned long                   ageing_timer;
mac_addr                        addr;
unsigned char                   is_local;
unsigned char                   is_static;
};
```

算法的实现

Linux 在 net/bridge/br_fdb.c 中实现查找表，其中 fdb 标识转发数据库。查找过程将源取 MAC 地址来标识数据库中的一个表项，并计算散列函数 br_mac_hash() 以确定正确的散列表桶。下面的代码段 br_fdb.c 说明如何查找表。

```
struct net_bridge_fdb_entry *__br_fdb_get(struct
net_bridge *br, const unsigned char *addr)
{
        struct hlist_node *h;
        struct net_bridge_fdb_entry *fdb;
              hlist_for_each_entry_rcu(fdb,h,
              &br->hash[br_mac_hash(addr)],hlist) {
              if (!compare_ether_addr(fdb->addr.addr,
                                                addr)) {
                    if (unlikely(has_expired(br, fdb)))
                          break;
                    return fdb;
              }
        }
        return NULL;
}
```

宏 hlist_for_each_entry_rcu() 搜索由 &br->hash [br_mac_hash(addr)] 指针指向的桶表，以便找到在 net_bridge_fdb_entry 中的正确表项，其中还包含要转发的端口。这里 rcu（Read Copy Update）（读取－复制－更新）是在内核 2.5 版本的开发过程中添加到 Linux 内核中的同步

机制，同时提供线程之间的互斥。用老化机制查找以避免搜索。如果一个表项已经过期，搜索就被忽略。如果网络拓扑发生变化，这种机制就更新数据库。

当接收到一个帧时，就将一个新表项插入转发数据库中。这就是网络运行中的自学习机制。本代码段在 br_fdb.c 中，如下所示。

```
static int fdb_insert(struct net_bridge *br, struct
net_bridge_port *source, const unsigned char *addr)
{
        struct hlist_head *head = &br->hash[br_mac_
        hash(addr)]; struct net_bridge_fdb_entry *fdb;
        if (!is_valid_ether_addr(addr))
                return -EINVAL;
        fdb = fdb_find(head, addr);
        if (fdb) {
                if (fdb->is_local)
                        return 0;
                fdb_delete(fdb);
        }
        if (!fdb_create(head, source, addr, 1))
                return -ENOMEM;
        return 0;
}
```

插入是以在转发数据库中查找到达的 MAC 地址开始的。如果找到一个表项，就将它替换成新的表项；否则，就将新的表项插入数据库中。

练习

1. 跟踪源代码，并了解老化定时器是如何工作的。
2. 在 Linux 内核源代码的 fdb 数列表中查找，看能找到多少表项。

3.5.2 生成树协议

随着一个桥接网络拓扑变得更庞大和更复杂，网络管理员可能无意中会在拓扑中创建了一个回路。这种情况是不需要的，因为帧可能循环流动并且也可能使地址表变得不稳定。例如，考虑以下由两个端口交换机形成一个回路并通过某个站点将帧广播到回路上后所导致的严重问题。每台交换机将接收到的帧向广播到其他交换机上，使它围绕回路无限地循环下去。

为了解决回路问题，IEEE 802.1D 制定了一个生成树协议（STP）以消除在桥接网络上出现的回路。因其实现简单，所以几乎所有的交换机都支持该协议。图 3-41 是一个形成生成树的简单例子，步骤如下。

1）最初，给每台交换机和端口分配一个由可管理的优先级值和交换地址（或端口标识符的端口号）组成的标识符。为了简单，这里我们使用 1～6 的标识符。

2）每条链路指定了一个与链接速度成反比的开销。这里我们假定所有链路开销为 1。

3）具有最小标识符的交换机充当根。在交换机之间通过交换配置信息被来选举根。

4）将每个局域网连接到当前有效拓扑中的某台交换机的一个端口上。局域网传输来自根的帧所经过的端口称为指定端口（DP），而该端口所在的交换机称为指定网桥。交换机从根交换机接收帧所经过的端口称为根端口（RP）。

5）配置信息以桥协议数据单元（BPDU 网桥协议数据单元）的形式周期性地从根向下传播，其中目的地址是为交换机保留的组播地址 01－80－c2－00－00－00。BPDU 被包含根标识符、传输交换机标识符、传输端口标识符和从根开始的路径开销等信息。

6）每台交换机可以根据接收到的 BPDU 携带的信息进行自我配置。配置规则：

- 如果通过与 BPDU 中通告的路径开销相比较，交换机发现自己可以提供一个更低路径开销，那么它将试图通过传输具有更低路径开销的 BPDU 指定一个网桥。
- 在出现歧义的情况下，如多种选择具有相同的开销，就选择具有最低标识符的交换机或端口作

为指定网桥（或端口）。

- 如果交换机发现自己有一个比目前的根更低的标识符，它就试图通过传送带有其标识符作为根标识符的 BPDU 成为新的根。
- 注意交换机并不转发任何进入的 BPDU 信息，但会创建新的 BPDU 以携代它的新状态给其他的交换机设备。

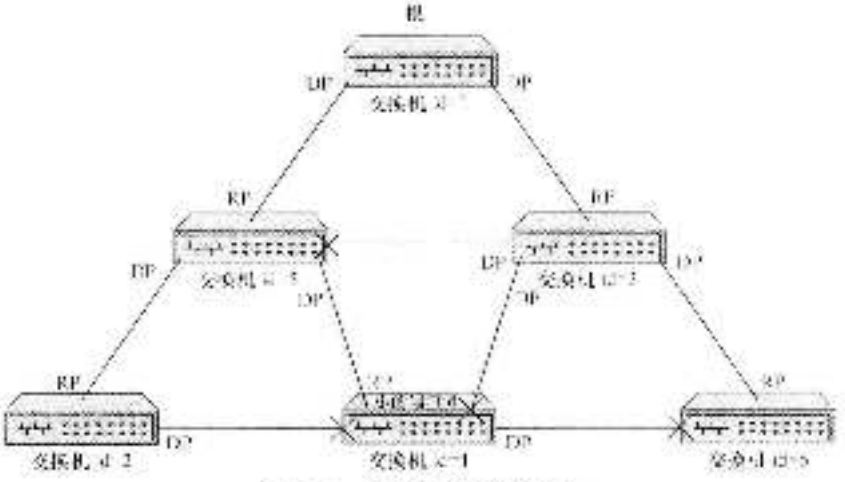


图 3-41　带有回路的桥接网络

7）除了 DP 和 RP 外，所有其他端口都被阻塞，不允许被阻塞的端口转发或接收数据帧，但阻塞端口仍继续监听 BPDU 看它能否再次激活。图 3-41 还给出了产生的生成树。读者最好能够跟踪学习该过程。总之，协议Ⅱ有效，它能根据可能的拓扑变化动态地更新生成树。

开源实现 3.8：生成树

概述

利用进入（ingress）BPDU 的信息更新生成树配置，如未将开始。当网桥接收到一个 BPDU 时，它首先通过解析或建立一个包含 BPDU 信息的结构，然后根据 BPDU 信息更新网桥配置。之后，选择新根并确定指定的端口，然后根据新的配置更新端口状态。

框图

图 3-42 说明了 BPDU 帧处理的调用流程。流程基本上按照上面介绍的顺序进行，我们下面将详细描述每一个函数调用。

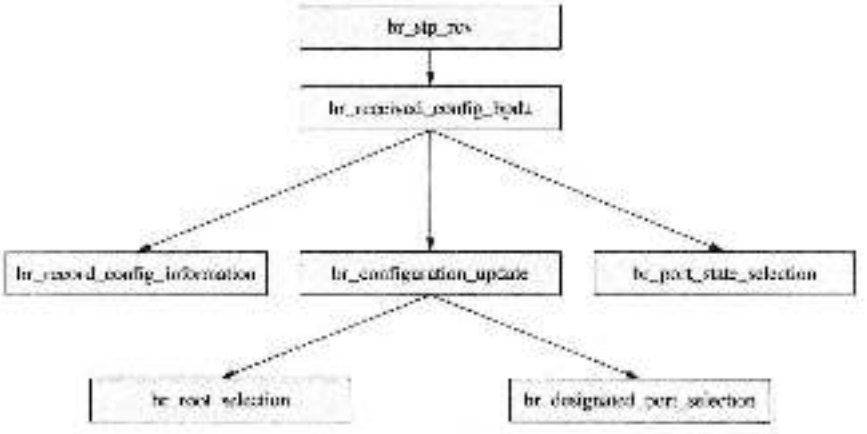


图 3-42　处理 BPDU 帧的调用流程

数据结构

br_config_bpdu 是最重要的结构（在 net/bridge/br_private_stp.h 中定义），经过分析成后从 BPDU 域中导出 BPDU 信息。在结构中包含以下字段，并且这些字段可以直接从 BPDU 域中的协议字段中被剔出。

```
struct br_config_bpdu
{
        unsigned        topology_change:1;
        unsigned        topology_change_ack:1;
        bridge_id       root;
        int             root_path_cost;
        bridge_id       bridge_id;
        port_id         port_id;
        int             message_age;
        int             max_age;
        int             hello_time;
        int             forward_delay;
};
```

用接收到的 BPDU 帧更新 net_bridge 结构（定义在 net/bridge/br_private.h）中的全局网桥配置。这种结构不仅用于生成树协议，也可用于其他协议。它还包含了网桥需要的全部数据结构，即接收数据库。因此，在本节暂不讨论。

算法实现

在 br_stp_bpdu.c（在 net/bridge 目录下）中的 br_stp_rcv() 函数处理有关生成树配置的更新。函数解析 BPDU 并构建 BPDU 信息的 br_config_bpdu 结构。然后将结构和端口信息传递给 br_stp.c 中的函数 br_received_config_bpdu()。这个函数首先调用 br_record_config_information() 在端口注册 BPDU 信息，然后调用 br_configuration_update() 更新网桥配置。代码段如下：

```
void br_received_config_bpdu(struct net_bridge_port
*p, struct br_config_bpdu *bpdu)
{
        // Skip some statements here
        if (br_supersedes_port_info(p, bpdu)) {
                br_record_config_information(p, bpdu);
                br_configuration_update(br);
        br_port_state_selection(br);
        // Skip some statements here
}
```

配置更新后，br_port_state_selection() 中的端口状态根据端口分配的角色也进行更新。例如，可以阻塞端口以避免回路。注意，可以从多处调用 br_configuration_update()。比如，系统管理员可以执行命令禁用端口或更改路径开销。这种情况也将触发网桥配置的更新。

函数 br_configuration_update() 直接调用 br_root_selection() 和 br_designated_port_selection() 函数以便分别选择一个新的根并确定指派的端口。如果根或指派的端口更改了，那么路径开销也要更新。

练习

1. 简要描述 BPDU 派是如何沿着生成树的拓扑传播的。
2. 研究 br_root_selection() 函数学习如何选择一个新根。

3.5.3 虚拟局域网

一旦将一台设备连接到一个局域网上，它就属于该局域网。即局域网的部署完全由物理连接来确定。在某些应用中，我们需要在物理部署之上构建逻辑连接。例如，我们想要交换机的一些端口属于一个局域网，而另一些端口则属于另一个局域网。此外，我们可能将多台交换机的端口分配给同一个局域网而将所有其他端口分配给另一个局域网。一般来讲，我们在网络部署中需要这种灵

活性。

虚拟局域网（VLAN）可以为局域网提供逻辑配置。管理员可以方便利用管理工具工作而不改变底层网络结构的物理连接性。此外，通过 VLAN 隔离，可以将交换机的端口分配给不同的 VLAN，每一个行为使用物理上彼此的交换机一样。这样做，我们可以提高网络安全性并节省带宽，因为流量，特别是广播和广播流量，限制在它所属的一个特定定义的虚拟局域网内。比如，广播帧或带有未知单播目的地址的帧会只发往没有 VLAN 划分的交换机的所有端口上，这个就不仅在无关的端口上浪费带宽，而且也会泄露用户信息。通过将端口划分到多个 VLAN 中，帧被限制在所要发向的由端口组成的虚拟局域网内。

图 3-43 显示了一个实际例子以说明 VLAN 的用途。考虑两个子网：140.113.88.0 和 140.113.241.0。每个子网包含多个站点。如果我们想要将这两个子网与一台路由器连接起来，可以按照图 3-43 中描述的方式部署网络。

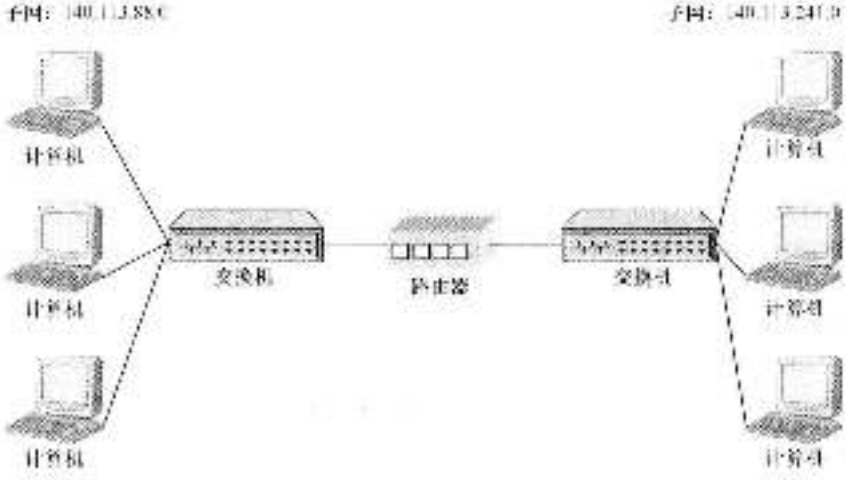


图 3-43 没有 VLAN 的两台交换机的部署

如果我们将交换机配置成仅带有两个 VLAN，那么只需要一台交换机就够了。将路由器连接到一个属于两个 VLAN 的端口上，并配置两个地址，每个对一个子网。在这种情况下，路由器将充当单臂路由器，如图 3-44 所示。目前，许多交换机，即第 3 层交换机，有能力充当按照第三层信息转发帧的普通路由器。利用虚拟局域网，管理员可以将端口任意分成多个子子网，这对于网络管理来讲非常方便。

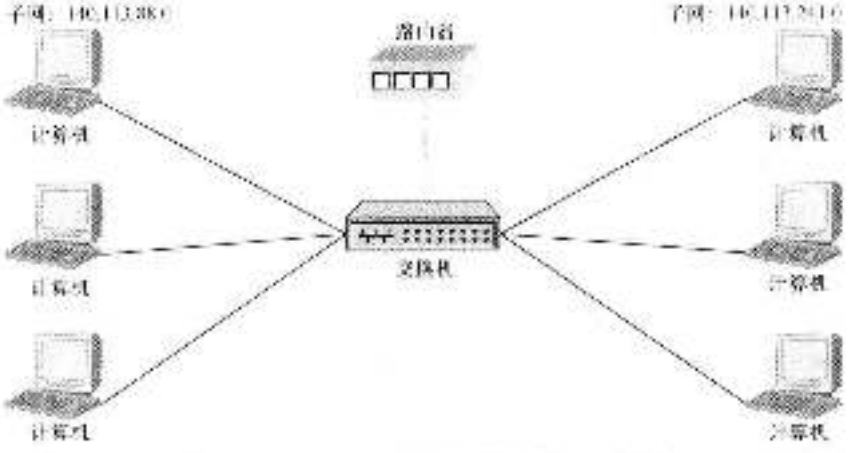


图 3-44 用 VLAN 和单臂路由器部署一台交换机

行动原则：虚拟局域网与子网

虚拟局域网是一个第2层的概念，它允许网络管理员在第2层配置连接性而不需要重新进行物理布线。例如，交换机的端口1和端口2可以配置成属于同一个虚拟局域网，而端口3和端口4属于另一个VLAN。虽然它们都是在同一台交换机上，但连接性可以从逻辑上进行分割。在同一个虚拟局域网内的主机在没有更高层设备，特别是路由器时，也可以相互通信，虚拟局域网限制能可以到达的范围（仅限于一个虚拟局域网内）。

子网是一个第3层的概念。在子网内的主机可以直接相互发送分组（也包括广播分组），而不需要路由器的帮助。这两个未能在该制广播域的上下文中类似。那么，它们有什么区别？

子网通过将主机设置成具有相同前缀的IP地址进行配置，这里由子网掩码决定前缀长度。相比之下，虚拟局域网是在一种第2层设备——交换机上配置的。前者是逻辑的，而后者是一种物理分隔。因此，在同一个VLAN内可以配置多个子网（例如，连接到同一台交换机而不分别VLAN）。但在逻辑上这些子网是分隔的。尽管逻辑分隔，但一个第2层的广播帧（带有全部为1的目的MAC地址）仍然可以达到整个虚拟局域网。这种情况下，最好在交换机上配置多个VLAN，在物理上分割广播域。

IEEE 802.1Q标准规定支持VLAN操作的一组协议和算法。这个标准从配置、配置信息的发布和中继等方面描述了VLAN体系结构准则。第一个是不言自明的。第二个是有关在支持VLAN的交换机之间允许发布VLAN成员信息的方法。第三个是处理如何分类和转发到达的帧，以及通过添加、修改或删除标签修改帧的过程。我们接下来讨论标签的概念。

IEEE 802.1Q标准没有说明结点应该如何与VLAN关联起来。虚拟局域网成员可以基于端口、MAC地址、IP子网、协议和应用。每个帧可以与携带VLAN标识符的标签关联起来，以便交换机能够快速确定其VLAN关联而不需要进行复杂的分类。但是标签会对帧的格式稍做修改。标签帧的格式在图3-45中描述[①]。一个虚拟局域网标识符有12位。允许VLAN的最大数量为4094（即$2^{12}-2$），假定其中一个标识符保留未使用，另一个是用来表示一个优先级的标签（参见下面）。

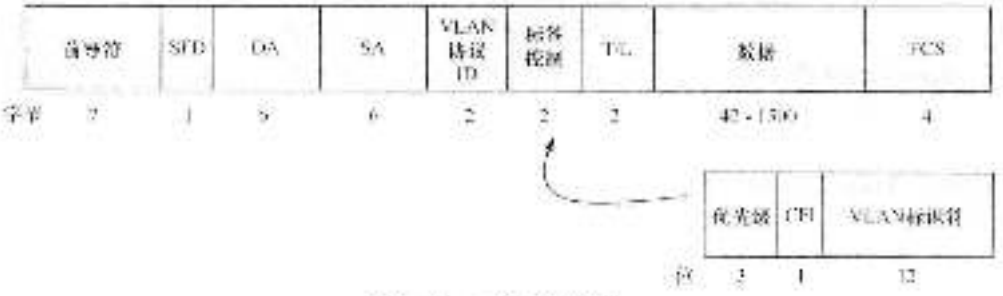


图3-45 标签帧的格式

优先权

如果在一个局域网中负载很高，那么用户会察觉到很长的延迟。有些语音或视频应用是时间敏感的，延迟很高时其质量将会恶化。传统上，局域网技术利用预留空间来解决问题，即能够提供比需要的更多的带宽。这种技术是可行的，因为在有线局域网中能够提供很便宜的高带宽。但是在短期的拥塞中，流量可能会暂时超过可用的带宽，因此可以为关键应用的帧分配更高的优先级以便保证它们能够得到更好的服务。

以太网本身并不具有优先级机制。至于802.1p，是后来集成到了IEEE 802.1D中的，它可将一个可选的优先级值分配给一个以太网帧。此值也在标签帧中携带，如图3-45所示。一个标签帧添加了4字节。它们分别是2字节类型字段，指示一个虚拟局域网协议类型（值=0x8100）；2字节标签控制信

① 注意VLAN并不局限于以太网。VLAN标准也可以应用到其他局域网标准中，比如令牌环。然而，由于以太网是最流行的，所以这里我们仅讨论以太网帧。

息字段。后者又可以进一步划分为3个优先字段：优先级、规范格式指示器（CFI）和虚拟局域网标识符。一个标签帧广不一定要携带 VLAN 信息。标签可以仅包含帧的优先级。VLAN 标识符有助于交换机识别帧属于哪个 VLAN。CFI 看起来很神秘。这是一个1位的字段用来指示 MAC 数据中可能携带的 MAC 扩展是否是数据规范格式。这里我们不详细讲解规范格式。有兴趣的读者可参照 IEEE 802.1Q 文档中的条款 9.3.2。

因为在优先字段中有3位，所以优先级机制允许8个优先级类。表3-11列出了标准中建议的优先值与流量类型的映射。一台交换机可以根据标签值对到达的流量进行分类并安排适当的队列服务来满足用户的需求。

表 3-11 建议的优先级值和流量类型的映射

优先级	流量类型	优先级	流量类型	优先级	流量类型	优先级	流量类型
1	背景	0（默认）	最大努力	4	受控负载流	6	<10ms 延迟和抖动
2	空闲	3	尽全力转发	5	<100ms 延迟和抖动	7	网络控制

链路聚合

最后，我们介绍链路聚合。多条链路可以聚合起来就像是一个大容量的管道。例如，如果需要更大的链路容量时，用户可以将2台千兆链路聚合成一根2千兆的管路。因为在网络部署中链路聚合已经带来了灵活性，所以你不需要购买10千兆以太网产品。

链路聚合原本是 Cisco 的一种技术，称为 EtherChannel，通常称为端口汇聚（trunking），后来于2000年标准化为 802.3ad，并不局限于交换机之间的链路：交换机和站点之间以及站点与站点之间的链路也可以聚合起来。链路聚合的原理很简单：发射机在聚合链接上分发帧，接收器从聚合链路收集这些帧。不过，某些难题使设计复杂起来。例如，考虑在多个短帧之后跟一个长帧的情况就是如此。如果将长帧分配到一条链路上而将短帧分配到另一条链路上，接收器可能会收到这些乱序的帧。虽然上层协议可以处理无序的帧，但是效率会很低。流中帧的排序必须在链路层中维持不变。为了负载均衡或者由于链路故障，可能将一个数据流从一条链路转移到另一条链路上传输。为满足这些要求，就设计了链路聚合控制协议（LACP）。建议对要了解详情的读者，参照 IEEE 802.3 标准中的条款43。

3.6 网络接口的设备驱动程序

3.6.1 设备驱动程序的概念

操作系统的主要功能之一是控制输入/输出设备。操作系统中的输入/输出部分可以从结构上分成四层，如图3-46所示。中断处理程序也是驱动程序的一部分。



图3-46 输入/输出软件的结构

所有设备相关的代码都嵌入设备驱动程序中。设备驱动程序向设备寄存器发出命令并监督它们是否能够正确运行。因此，网络设备驱动程序仅是操作系统的一部分，它知道网络适配器有多少个寄存

器以及作何用途。

设备驱动程序的工作就是接收从它之上的独立于设备的软件发来的抽象请求，并通过发出命令给设备寄存器来处理这些请求。发出命令后，或者设备驱动程序阻塞自己直到中断到来才解除阻塞，或者操作立即完成，此时不需要驱动程序阻塞。

3.6.2 在 Linux 设备驱动程序中如何与硬件通信

在设备驱动程序可以与设备通信之前，它必须初始化环境。初始化包括为了能够与设备寄存器通信而进行的系统 I/O 动作，以及为了正确安装中断处理程序而进行的核对 IRQ。我们还将讨论为了传输大批量数据而进行的直接内存访问。

探测 I/O 端口

硬件设备通常有多个寄存器，作为了读和写将它们映射到连续地址的区域。读和写这些地址（实际上就是寄存器）就可以控制设备。并非所有的输入/输出端口都绑定到设备寄存器。用户可以将内容转储（dump）到/proc/ioports 中以便查看地址与设备之间的映射。

设备程序员在 I/O 端口可以为设备指定一个区域。请求必须首先检查区域是否已经分配给其他设备。检查检查分配区域必须在一种原子操作中运行，否则，其他设备可能获得检查后的区域并产生错误。获取一个输入/输出端口的区域后，设备驱动程序可以通过读或写端口以 8 位、16 位、32 位为单位（具体取决于寄存器的宽度）探测设备寄存器。这些操作是利用前面即将介绍的特殊函数来进行的。操作之后，如果区域不再被使用，驱动程序就可以将区域返还给系统。

中断处理

除了不断探测设备寄存器外，在探测期间驱动程序可以使用中断放弃 CPU 将它转交给其他任务的处理。中断是一种从硬件产生的异步事件以引起 CPU 的注意。设备驱动程序可以将一段代码（即处理程序）注册到一个中断，以便如果发生中断就让处理程序执行。系统上的中断都进行了编号，编号到设备之间的映射可以从文件/proc/interrupts 中看到。

中断行的注册类似于 I/O 端口的获取。驱动程序可以请求一个中断行，使用它，完成工作后再释放它。问题在于具体哪个中断行将被设备使用。虽然用户可以手动指定一个中断行，但这种做法需要额外的工作才能找出其中可使用的中断行。一个更好的解决办法就是运行自动检测。例如，PCI 标准要求设备声明将在寄存器中使用的中断行，以便驱动程序可以通过从 I/O 端口检索号码学习设备的中断行。并非每个设备都支持这种自动检测。另外一种方法就是如果不提供支持就请求设备产生一个中断，并观察哪一行活跃可用。

中断处理的一个问题是如何在中断处理程序中执行长任务。在响应设备中断中经常有许多工作要做，但中断处理程序需要尽快完成而不会阻塞其他中断过长的时间。Linux 通过将中断处理程序分裂成两部分来解决该问题。上半部是对中断做出响应的例程，也是注册中断行的处理程序。下半部处理当时的部分，上半部在安全时间高度它的执行。说明执行时间需求不是那么关键。因此，在上半部处理程序完成后，就可以释放 CPU 去处理其他任务。Linux 内核具有两种机制实现下半部处理：BH（也称为下半部）和 tasklets（小任务）。前者是在老版本中实现的。新的 Linux 内核自 2.4 以后的版本实现 tasklets，因此在介绍下半部处理时我们主要介绍后者。

直接内存访问

直接内存访问（DMA）是在没有中央 CPU 的参与下，在数据传输大批量数据进、出内存的一种硬件机制。这种机制能够大大地增加设备的吞吐量并减轻处理器的负担。

DMA 数据的传输可以以两种方式来触发：1）软件从系统调用（如 read）来请求数据；2）硬件异步地写数据。当程序员式地从系统调用请求数据时使用前者，而当数据采集设备可以异步地将获取的数据写入内存时使用后者，即使在没有运程序要它时也是如此。

前一种方式的步骤概述如下：

1）当进程需要读取数据时，驱动程序分配一个 DMA 缓冲区。为了 DMA 缓冲区从硬件中读取数据，进程进入睡眠。

2）硬件将数据写入 DMA 缓冲区中，并在写结束后发出一个中断。

3）中断处理程序获取数据并唤醒进程。现在，这个进程就有了数据。

后一种方式的步骤概述如下：

1）硬件发出一个中断宣布数据到达。

2）中断处理程序分配 DMA 缓冲区并通知硬件开始传输。

3）硬件从设备向缓冲区写入数据，当它完成时发出另一个中断。

4）处理程序发送新的数据并唤醒有关进程处理数据。

我们将仔细观察在以下开源实现中的相关函数。

开源实现 3.9：探测 I/O 端口、中断处理和 DMA

概述

Linux 设备驱动程序与硬件之间交互是通过输入/输出端口探测、中断处理和 DMA。将输入/输出端口映射到硬件设备上的寄存器，这样设备驱动程序能够访问输入/输出端口读成写寄存器。例如，驱动程序可以将命令写入寄存器，也可以读取设备的状态。通常，当驱动程序将任务分配给设备执行时，它可能不断地轮询状态寄存器以便了解任务是否完成。但如果任务不能立即完成，这样做可能会浪费 CPU 周期。驱动程序可以利用中断机制来通知 CPU，之后调用相关中断处理程序处理中断。因此，处理器不需要仔细等待。如果有大量的数据需要传输，DMA 可以代表 CPU 处理传输。与这些机制相关的函数调用会在下面介绍。

函数调用

输入/输出端口

自 Linux 内核版本 2.4 以后，输入/输出端口已输入到通用资源管理。我们可以使用设备驱动程序中的下列函数获得设备的输入/输出端口：

```
struct resource *request_region (unsigned long start,
unsigned long n, char* name);
void release_region (unsigned long start , unsigned
long len);
```

我们使用 request_region()探测 I/O 端口，这里 start 是 I/O 端口区域的起始地址，n 是将要获得的输入/输出端口号，name 名字是设备名称。如果返回一个非零的值，表示请求成功。然后当完成时，驱动程序应该调用 release_region()释放该端口。

获得输入/输出端口的区域后，设备驱动程序可以访问端口以便控制设备上的寄存器，这可能是通过命令也可能是状态注册器。大多数硬件区分 8 位、16 位和 32 位端口，所以 C 程序必须调用不同的函数访问不同大小的端口。Linux 内核定义了以下函数访问 I/O 端口：

```
unsigned inb (unsigned port);
void outb (unsigned char byte, unsigned port);
```

inb()读字节（8 位）端口，而 outb()写字节端口。

```
unsigned inw (unsigned port);
void outw (unsigned char byte, unsigned port);
```

inw()读 16 位端口，而 outw()写 16 位端口。

```
unsigned inl (unsigned port);
void outl (unsigned char byte, unsigned port);
```

inl()读 32 位端口，而 outl()写 32 位端口。

除了一次性进和出操作外，Linux 支持以下字符串操作。如果 CPU 没有指令用于字符串的输入/输出，这实际上可能是由单个 CPU 指令或紧凑的环境执行的。

```
void insb (unsigned port, void *addr, unsigned long
count);
void outsb (unsigned port, void *addr, unsigned long
count);
```

insb()从字节端口读 count 字节，并将读出字节存到从地址 addr 开始的内存中。outsb()将位于存储器地址 addr 的 count 计数字节写入字节端口。

```
void insw (unsigned port, void *addr, unsigned long
count);
void outsw (unsigned port, void *addr, unsigned long
count);
```

它们的操作类似前两个，除了端口为 16 位端口外。

```
void insl (unsigned port, void *addr, unsigned long
count);
void outsl (unsigned port, void *addr, unsigned long
count);
```

它们的操作类似前两个，除了端口为 32 位端口外。

中断处理

与其等输入/输出端口的方法一样，驱动程序使用以下函数请求（安装）和释放（卸载）一个中断处理程序到一个中断号上。

```
#include <linux/sched.h>
int request_irq(unsigned int irq, irqreturn_t
(*handler) (int, void *, struct pt_regs *), unsigned
long flags, const char *dev_name ,void *dev_id);
void free_irq (unsigned int irq, void *dev_id);
```

在前者，irq是被请求的中断，handler是相关的中断处理程序。其他参数是：flags是中断的属性，dev_name是设备名称，dev_id是指向设备驱动程序的指针。free_irq()中参数的含义与request_irq()中的一样。

当一个中断发生时，在 Linux 内核中给中断处理程序中断号进入栈中，并调用 do_irq()函数（do_irq()函数调用它的参数是 irq 和 regs）。do_irq()将查找与中断相关的中断处理程序，如果它存在就通过 handle_irq_event()来被调用它。否则，函数被返回，CPU 以恢复原来被中断的工作。中断处理程序是可抢先的，因此其他中断不会被禁止太久。中断处理程序可以很快获取 CPU 并在其余工作被调度前完成它的下半部。

新版本的 Linux 为下半部函数使用 tasklet。例如，如果你编写一个下半部例程的函数 funct()，那么第一步就是通过宏 DECLARE_TASKLET(tsak, funct, 0)来声明 tasklet，这里 task 就是 tasklet 的名字。tasklet 被 tasklet_schedule(&task)调度后，tasklet 例程或 task 将在系统空闲的时候执行一会儿。

以下宏最适于使用 tasklets 很有用：

```
DECLARE_TASKLET(name, function, data);
```

这声明 tasklet。这里 name 是 tasklet 名字，function 就是将要执行的实际 tasklet 函数，data 就是将参传给 tasklet 函数中的参数。

```
tasklet_schedule(struct tasklet_struct *t);
```

函数调度将查看系统空闲的时候执行的 tasklet，这里 t 指的 tasklet 结构。

直接内存访问

DMA 缓冲区分配有点儿复杂，这是由 CPU 缓存的一致性问题所导致的。如果缓冲区中的内容改变，那么 CPU 就应该使它到 DMA 缓冲区中的缓存映射无效。因此，驱动程序应该小心保证 CPU 和这 DMA 传输。为了减轻程序员解决在这个问题的工作量，Linux 为分配缓冲区提供了一些函数。这里，我们为缓冲区分配介绍了一种常见的方法。

驱动程序分配了缓冲区后（使用 kmalloc()），它便使用以下函数指示缓冲区映射到该设备上。

```
dma_addr_t dma_map_single(struct device *dev,
void *buffer, size_t size, enum dma_data_direction
direction);
```

其中 dev 参数的该设备，buffer 是缓冲区的起始地址，size 是缓冲区的大小，而 direction 是数据移动的方向（分为：来自设备，到设备，或双向）。传输之后，使用以下函数解除映射

```
dma_addr_t dma_map_single(struct device *dev,
void *buffer, size_t size, enum dma_data_direction
direction);
```

使 I/O 缓冲区和中断一样，DMA 信道必须在使用之前进行注册。用于注册和释放的函数是：

```
int request_dma(unsigned int channel, const char
*name);
void free_dma(unsigned int channel);
```

该 channel 参数是一个 0～MAX_DMA_CHANNELS 之间的数（在 PC 上通常为 8），具体由内核配置定义。name 参数表示设备。

注册后，为了能够正确执行，驱动程序应配置 DMA 控制器。以下函数可以执行配置：

```
void set_dma_mode(unsigned int channel, char mode);
```

第一个参数是 DMA 信道，mode 参数可以是 DMA_MODE_READ，以使从设备读取，也可以是 DMA_MODE_WRITE，用于写入设备。而 DMA_ MODE_ CASCADE 用于连接 2 个 DMA 控制器。

```
void set_dma_addr(unsigned int channel, unsigned int
addr);
```

第一个参数是 DMA 信道，addr 参数是 DMA 缓冲区的地址。

```
void set_dma_count(unsigned int channel, unsigned int
count);
```

第一个参数是 DMA 信道，count 参数是将要传输的字节数。

练习

1. 通过研究 tasklet_schedule() 函数调用，解释如何调度 tasklet。

2. 列举一个软中断用于中断的实例。

一个典型的 Linux 系统为它的各种可能使用硬件配备了很多设备驱动程序。在这些驱动程序中，用于网络设备的与计算机网络显著相关的。Linux 内核支持多种网络接口驱动程序（参见 drivers/net 目录）。我们选择 NE2000 以太网接口的驱动程序介绍网络接口驱动程序的设计。

开源实现 3.10：Linux 中的网络设备驱动程序

概述

本节利用一个实际例子来说明设备驱动程序如何实现与网络接口之间的交互。交互主要包括设备初始化、传输处理和接收处理。在设备初始化中，驱动程序分配空间，并初始化重要的网络接口数据结构，如 IRQ 号和 MAC 地址。在传输和接收处理中，设备驱动程序利用中断来通知处理的完成。

框图

设备驱动程序中最重要的就是帧的传输和接收。我们在图 3-47 和图 3-48 中说明这个过程。

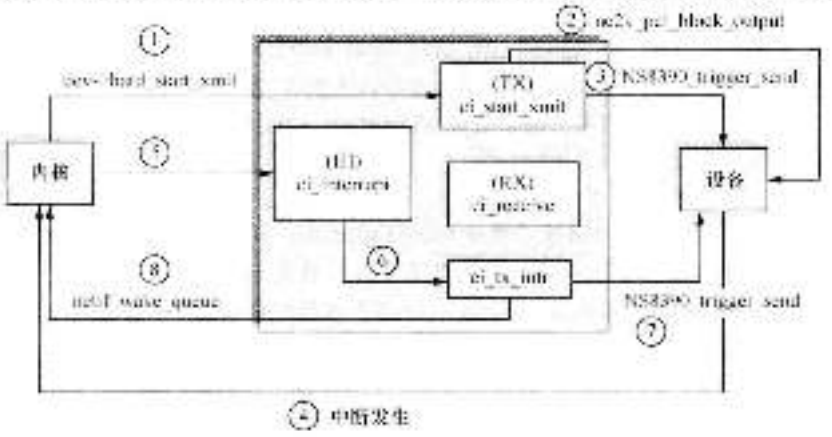


图 3-47 帧传输过程中执行的函数序列

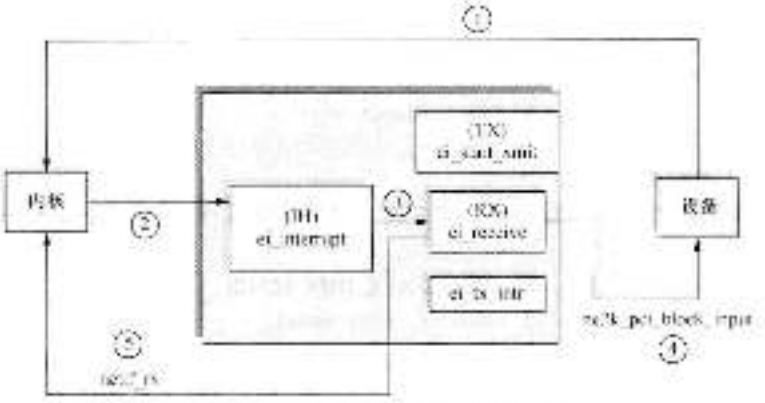


图3-48 帧接收期间执行函数的序列

数据结构

net_device数据结构是与网络设备的信息相关联。当初始化一个网络接口时，分配该接口的结构空间并进行注册。这种结构非常大，包含与配置、统计、设备状态、到底管理等相关的字段。下面列出配置中与初始化相关的几个字段。

char name [IFNAMSIZ]：设备名称，例如eth0。

unsigned int irq：设备使用的中断号。

unsigned short type：说明设备接口的号，如以太网。

unsigned char dev_addr [MAX_ADDR_LEN]：设备的链路层地址。

unsigned char addr_len：链路层地址的长度，以太网中为6字节。

int promiscuity：是否进行在混杂模式。

算法实现

设备初始化

Linux内核使用net_device数据结构表示一个网络设备，其中包括与该设备属性相关的字段。在网络接口可以使用之前，它的net_device结构必须初始化，并且设备必须进行注册。利用net/core/dev.c中的alloc_netdev()函数进行初始化。如果初始化成功，就返回一个指向新分配结构的指针。将三个参数传递给alloc_netdev：结构的大小、设备名称、启动例程。alloc_netdev()函数是通用的，可从各种设备类型的初始化函数调用。例如，net/ethernet/eth.c中的alloc_etherdev()调用带有设备名称为"eth%d"参数的函数alloc_netdev()。因此内核可以给该设备类型分配第一个未分配的编号。用dev_alloc_name()函数来填写名称。这就是为什么我们会在用户空间看到像"eth0"等名称的原因。初始化设置net_device数据结构中的IRQ、I/O内存、I/O端口、MAC地址、队列规程等字段。

使用alloc_netdev()分配和初始化net_device结构后，netdev_boot_setup_check()函数检查网络设备可选的启动配置参数，例如IRQ等。经过该过程后，该设备利用register_netdevice()函数在设备驱动程序中进行注册。同样，调用函数unregister_netdevice()从内核中卸载设备驱动程序，还应该释放设备占用的资源（如IRQ）。

传输过程

图3-47显示了从NE2000以太网接口发出的传输过程。当内核有一个帧要发送时，它首先调用适用的hard_start_xmit()函数，然后调用设备上的特定ei_start_xmit()函数。函数ei_start_xmit()调用ne2k_pci_block_output()将帧移动到网络接口上。当帧发送出去后，NE2000接口便发送中断通知内核，内核将调用相应的中断处理程序ei_interrupt()。ei_interrupt()函数将首先确定中断所属的类型。当它发现中断表明的是发送帧时，它就调用ei_tx_intr()函数，接下来又调用NS8390_trigger_send()在接口上传输下一个帧（如果有），然后调用netif_wake_queue()让内核继续下一个任务。

接收过程

图3-48显示了前面例子中的接收过程。当网络接口接收到这个帧时，它利用中断通知内核。内核然后调用相应的处理程序 ni_interrupt()。函数 ni_interrupt() 确定中断所属的类型，因为中断代表帧接收，所以调用 ci_receive() 函数。函数 ci_receive() 将调用 ne2k_pci_block_input() 将该从网络接口移动到系统内存，并将帧填入到 sk_buff 结构中。函数 netif_rx() 将帧传给上层，然后内核执行下一个任务。

练习

1. 解释网络设备中如何将帧移动到 sk_buff 结构中（见 ne2k_pci_block_input()）。
2. 找出设备注册的数据结构。

性能问题：驱动程序中断和 DMA 处理

表3-12显示了通过具有2.33GHz CPU的PC上Realtek 8169以太网适配器处理ICMP帧时所花费的中断处理时间和DMA逝去时间。DMA逝去时间不消耗CPU时间，因为数据传输由DMA去完成。结果显示，中断处理程序的处理时间并不随着的大小而改变。其原因在于，中断处理程序的主要任务，如通过向设备寄存器发出命令与底层硬件之间的交互，是独立于帧的。另一方面，DMA时间取决于传输帧的大小。另一种看法是，中断处理程序的RX（接收）时间要比TX（发送）时间高，而DMA的RX（接收）时间通过高于TX（发送）时间。RX中断处理程序为了发送需要分配和映射DMA缓冲区，因此它就比TX中断处理程序花费更多一点的时间。我们测得的RX（接收）DMA时间包括DMA传输时间以及额外的DMA控制器的硬件处理时间，但TX DMA时间只包含DMA发送时间，这导致RX DMA时间比TX DMA的时间高很多。

表3-12 中断和DMA的数据包处理时间 单位：微秒（μs）

	中断处理程序		DMA	
ICMP数据包的有效载荷大小	TX	RX	TX	RX
1	2.43	2.43	7.92	9.27
100	2.24	2.71	9.44	12.49
1000	2.37	2.51	18.58	83.95

最后，值得注意的是，中断处理时间取决于CPU的速度，DMA上坐去时间主要取决于底层的适配器。正如我们在1.5.3节中曾介绍过的，Intel PRO/100以太网适配器和1.1GHz CPU的DMA时间大约是1μs，在链路层上处理64字节数据包需要的时间大约是8μs（TX）和11μs（RX），与这里的值不同。表3-12中对应于100字节的数据包的一行显示中断时间较低，而DMA时间较高。虽然值发生了变化，但这里得到的观测值是独立于硬件的。

历史演变：驱动程序的标准接口

在早期的x86-DOS年代，操作系统没有提供任何网络模块，因此驱动器直接与应用程序绑定并且必须自己处理所有的网络功能。1986年FTP Software开发了PC/TCP产品，这是用于DOS的一个TCP/IP库，并且定义了数据包驱动程序接口，它控制PC/TCP和设备驱动程序之间的编程接口。有了公共接口的帮助，驱动程序开发人员在为新硬件开发驱动程序时就不需要修改太多。商业操作系统标准化了这些接口，例如，由Novell公司和苹果公司制定的开放数据链路接口（ODI）以及微软和3Com制定的网络驱动程序接口规范（NDIS）。Linux直到内核版本2.4才规定了其接口的名字。它使用中断驱动的方法来处理收到的帧。自内核版本2.5以后，设计了一种新的接口，称为新应用编程接口（NAPI），旨在支持高速网络，但它在内核版本2.6中实现驱动程序时仍是一个可选功能。NAPI设计的理念是过于频繁的中断会降低系统的性能。NAPI交替地使用中断处理程序，以保持短延迟，并且它采用循环轮询方式一次就能处理多个帧，而不是每次都要触发处理程序。

设备驱动程序还必须支持另一个接口：硬件规范。这是一个通常称为数据表单的规范，它记录驱

动程序和硬件之间接口的大体。它提供了详细的编程信息，包括 I/O 寄存器的功能和宽度以及 DMA 控制器的性能。设备开发者按照规范初始化硬件，读取状态，请求 DMA 传输，发送和接收帧。Novell 公司的 NE2000 网络卡特性非常成功，以至于它的设备驱动程序成为一个事实上的标准规范。很多厂商声称他们的网络芯片组是与 NE2000 兼容的，以简化驱动程序的开发。为了与 NE2000 兼容，I/O 寄存器和 DMA 控制器的功能完全模仿 NE2000 数据表。由于其功能有限，NE2000 已不再流行。用于控制器编程的硬件接口器的数据表早已经成为驱动程序开发者的标准实践。

3.7 总结

我们从介绍链路层的关键概念（包括成帧、寻址、差错控制、流量控制和介质访问控制等）的介绍入手。这些更高层次的概念，为两个或两个以上的节点互相通信提供了物理信号之上的传输机制。然后，我们就可以利用这些概念学习流行的有线和无线链接的链路技术。在有线和无线技术中，我们特别关注以太网和 IEEE 802.11 无线局域网，因为它们在各自的领域中已经成为占据主导的技术。一般来讲，以太网更快、更可靠，但 802.11 无线局域网具有移动性，而且部署起来更容易。我们还介绍了多个局域网互联的桥接技术。桥接的主要问题包括广播转发、避免转发回路的生成树协议、方便局域网配置的虚拟局域网等。介绍过所有这些技术之后，我们解释了网络接口设备驱动程序的实现。从这些实现中，你应该知道网络接口到底是如何工作的。

虽然多年来以太网和 IEEE 802.11 无线局域网的速度都大大提高了，但这主要是由于物理层信号处理技术的进步所导致的。链路部分，如成帧，为了保持向后兼容性几乎保持不变。然而，链路技术也有其自己的进步，例如更好的可靠性、更好的介质访问控制（如全双工操作），以及更好的安全性。有些机制，如链路聚合，也有助于节点之间的总吞吐量的提高。正在发生的渐变包括更高的速度、链路层的 QoS 和冗余机制。速度永远是追求的目标。目前，40Gbps 和 100Gbps 以太网正在兴起，802.11ac 标准的原始数据率提高到 600Mbps。在无线技术（如 WiMAX）中提供链路层的 QoS，以及节能技术也一直都是移动设备的主要问题。

链路层协议主要处理既可以通过有线也可以通过无线链路直接获取的两个节点之间的连通性。然而，互联网中任意两个节点之间的连通性会更困难，因为互联包含数十亿台主机的巨大互联网中数据包需要能够从一个节点通过多条链路到其他节点。首先，必须有一个可扩展的寻址机制来解决在互联网上这么多台的主机，使源和目的主机之间的节点不需要在整个地址空间中保持到达每一个可能的目的地的路由。其次，路由必须定义良好，以反映最新的从源到目的地的连接状态。例如，如果在路由中的某条链路断开，就必须通过某种方式知道到这个问题，并从源到目的地选择一条新的路由。这些都是第 4 章要解决的问题。因为互联网协议（IP）是网络层的主要协议，本章应该包括互联网协议是如何解决有关可扩展寻址、分组转发和可扩展路由信息交换等问题。

常见陷阱

以太网性能 （半双工和全双工模式利用率）

研究人员曾经对在极其沉重的负荷下以太网的最大信道利用率及其感兴趣，尽管事实上，如此重负荷的情况是不可能发生的。计算机仿真、数学分析和现实世界的测量都是获得有价值结论的可行办法。与 ALOHA 和时隙 ALOHA 等简单机制不同，很难从数学上分析一套完整的 CSMA/CD 机制。当实验性以太网在施乐实验室发明时，Bob Metcalfe 和 David Boggs 在 1976 年发表了一篇论文，报告使用他们建立的简化模型时以太网最多可以达到约 37% 的信道利用率。不幸的是，这个值多年来一直被引用。今天以太网技术已经完全不同于 DIX 标准之初的实验模型。除了将 CSMA/CD 的精髓保留下来之外，已经采用了不同的 PCS、不同的重导符、不同的地址格式、不同的 PHY 等。此外，在同一种冲突域中都能容 256 台站点，这在现实世界中是不可能的。

后来 David Boggs 等人在 1988 年发表了论文，试图澄清这个缺陷。他们对带有 24 个站点的 10 Mbps 以太网系统经过不断地洪泛帧，进行了实际的测试。在压力测试下，结果表明最大峰时利用率会超过 95%，

最小帧可为90%左右。[1]这表明以太网性能上比较令人满意的。

随着交换机变得越来越流行，多段网络分成了许多单独的冲突域。同一冲突域中许多站点的情况会进一步缓和。自全双工操作问世后，CSMA/CD所强制的任何限制就不存在了，因此链路两端都可以以最大速度传输。能够提供最大顺速率和数据容量的交换机和为或速或非且其交换机。

另一个可能值得关注的问题就是以太网帧中的数据字段不够"长"。不同于其他技术，如令牌环，在4Mbps时有4528字节，在16或100Mbps时有18 173字节的数据字段，1518字节的最大未标记帧中的数据字段只有1500字节。这让人不禁会联想到，以太网中非数据的额外开销包括头部信息、尾部和IFG所占的百分比会大于其他技术。

以太网帧没有那么长有一定的历史原因。以太网发明于30多年前，当时存储器的价格昂贵，用于帧的缓冲区存储器相当有限。设计一个不太长的帧或者数据字段都具有一定意义。对于大的数据传输，如FTP流量，趋向于传输长帧，数据字段可以占据高达1500/（1518+8+12）=97.5%的信道带宽。额外开销相当低！且善增加帧的最大长度对于降低额外开销不会有很大帮助。

冲突域、广播域和VLAN

对于初学以太网的学生来说常常会混淆前两个术语。冲突域就是在多个站点同时传输时导致冲突的网络覆盖范围。例如，中继器集线器和与之连接的站点就形成一个冲突域。相反，交换机明确地利用端口来分隔冲突域。换句话说，来自连接到交换机端口上的共享局域网的传输不会导致与来自从同一个局域网但通过另一个端口的其他传输产生冲突。

然而，当一个帧以广播地址作为目的地址时，交换机仍然将它转发到除了到来端口外的所有其他端口上。广播流量可以到达的广播范围称为广播域，因此我们可能出于安全或者节省局域网内带宽的原因而限制广播流量的范围。

VLAN方法也可以按此分隔广播域，但它是与物理连接不同的一种逻辑分离。换言之，不需要更改物理连接。它通过设备配置进行隔离，就像对它进行了物理更改一样。需要使用如路由器等设备，提供高层的连接，以连接两个或多个独立的VLAN。

5-4-3规则和多段网络

据说以太网遵循5-4-3规则。这听起来很容易记住，但规则不像听起来那么简单。规则实际上是一个验证了多段10Mbps以太网络正确性的保守性规则。但这并不是每个以太网部署都应该遵循的定律。

正如我们前面提到的，为了正确地操作，在冲突域中的往返传播时间不应该太长。但是不同的传输介质和中继器集线器的数量会产生不同的延迟。作为网络管理员的快速指南，IEEE 802.3标准提出了两种传输系统模型。传输系统模型1是一套符合上述要求的配置。换句话说，如果遵循这些配置，网络就能正常运行。有时，你可能需要将自己的网络配置成不同于传输系统模型1的配置。你必须亲自己估算自己的网络是否达到要求。传输系统模型2提出了一套计算方法来为你提供帮助。例如，它会告诉你某类介质网段的延迟值。

第13条，"多段10Mbps基带网络的系统考虑"，为传输系统模型1列举了以下规则："多一条传输路径是由4台中继器和5个网段组成的，最多可以混合3段而其余必须是链路分段。"

这就是众所周知的5-4-3规则。一个混合网段是一个具有两个以上的物理接口的介质。一条链路是两个物理接口之间的一条支持全双工的介质。人们通常将链路分段看成是一个没有PC的网段，但这并不是精确的描述。该规则意味着，如果以这种方式配置网络，它就可以工作。随着越来越多的网段以全双工模式操作，此规则已经过时。

[1] Boggs论文中计算了利用头部、尾部以及IFG的开销。因此，尽管在他的论文中考虑了这样的开销，但是如果没有发生以外冲突，则为100%的利用率。

高字节顺序和低字节顺序

熟悉网络编程的人，也可能会听说高字节顺序（big-endian）和低字节顺序（little-endian）。他们知道网络字节顺序。例如，互联网协议（IP）的字节顺序使用高字节顺序。然而，我们曾经提到，以太网使用低字节顺序传送数据。这是否存在矛盾？

考虑一个4字节的字。每个字节分别用$b_1b_2b_3b_4$以逻辑表示。这里将它存储在内存中有两种选择：

1）将b_1存放在最低字节地址，将b_2中存放在第二低的字节地址，以此类推。

2）将b_1存放在最高字节地址，将b_2中存放在第二高的字节地址，以此类推。

前者称为高字节顺序，而后者称为低字节顺序。顺序随着主机上的CPU和操作系统的不同而改变。当在网络上传输某些多字节数据（如端数）时，会导致结果不一致。需要强制执行网络字节顺序，以保持一致性。最流行的网络层协议，互联网协议（IP），采用高字节顺序。无论主机字节顺序是什么，在传输之前都将数据转换为网络字节顺序。接收后再转换为主机字节顺序。以防出现上述不一致的情况。

这就是互联网协议要做的工作。链路协议逐字节地接收来自上层协议的数据。上层协议的字节顺序与链路协议无关。链路协议主要关心传输的位顺序，而不是字节顺序。

以太网使用低位顺序。在传输中它最先传输最低位，最后传输最高位。相反，令牌环或FDDI最先传输最高位，最后传输最低位。它们称为使用低位顺序。它们不应该与字节顺序相混淆。

进一步阅读

点到点协议

PPP、PPPoE和IPCP分别定义在RFC1661、RFC2516和RFC 1332中。Sun指南中介绍了在UNIX上的实际PPP操作。

- W. Simpson, "The Point-to-Point Protocol (PPP)," RFC 1661, July 1994.
- L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, and R. Wheeler, "A Method for Transmitting PPP over Ethernet," RFC 2516, Feb. 1999.
- G. McGregor, "The PPP Internet Protocol Control Protocol (IPCP)," RFC 1332, May 1992.
- Sun, *Using and Managing PPP*, O'Reilly, 1999.

以太网

Seifert是IEEE 802.1和802.3标准的合著者之一。他的《Gigabit Ethernet》一书准确地描述了技术并具有市场洞察力。如果你希望深入了解千兆以太网的技术细节而又不被措辞枯燥的标准所困扰，那么就可以阅读该书。他还有一本全面讨论交换机的书。你会发现在他的书中详细介绍了STP、VLAN、链路聚合和其他概念。Spurgeon是一位经验丰富的网络设计师，他的书从管理的角度介绍了以太网。

- Rich Seifert, *Gigabit Ethernet*, Addison Wesley, 1998.
- Rich Seifert, *The Switch Book*, Wiley, 2000.
- Charles E. Spurgeon, *Ethernet: The Definitive Guide*, O'Reilly, 2000.

下面给出了标准文件的列表。所有的IEEE 802标准已经在http://standards.ieee.org/getieee802/中免费自由下载。业界联盟，一个致力于促进下一代10千兆以太网的技术联盟，已经发布了白皮书。

- ISO/IEC Standard 8802-3, "Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," 2000.
- 10 Gigabit Ethernet Alliance, "10 Gigabit Ethernet Technology Overview White paper," http://www.10gea.org, Sept. 2001.

以下是MAC网桥标准和VLAN网桥标准，也在上述网站中提供。

- ISO/IEC Standard 15802-3, "Media Access Control (MAC) Bridges," 1998 Edition.

- IEEE 802.1Q, "Virtual Bridged Local Area Networks," 1998 Edition.

以下是几篇有关以太网研究引用率很高的论文，重两个是早期的以太网性能分析。

- R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, Issue 7, July 1976.
- D. R. Boggs, J. C. Mogul, and C. A. Kent, "Measured Capacity of an Ethernet: Myths and Reality," *ACM SIGCOMM Computer Communication Review*, Vol. 18, Issue 4, Aug. 1988.
- W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *IEEE/ACM Trans. Networking*, Vol. 5, Issue 1, pp. 71-86, Feb. 1997.
- L. Kramer, B. Mukherjee, S. Dixit, Y. Ye, and R. Hirth, "Supporting Differentiated Classes of Service in Ethernet Passive Optical Networks," *Journal of Optical Networking*, Vol. 1, Issue 9, pp. 280-298, Aug. 2002.
- J. Zheng and H. T. Mouftah, "Media Access Control for Ethernet Passive Optical Networks: An Overview," *IEEE Communications Magazine*, Vol. 43, No. 2, pp. 145-150, Feb. 2005.

无线协议

这里，我们列出了无线局域网的规范，也可从上述网站中下载。这里还有几本关于 802.11 的好书，以及下面引用率很高的有关 IEEE 802.11 无线局域网的 QoS 增强和网络性能的论文。

- ANSI/IEEE Standard 802.11, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification," 1999 Edition.
- M. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd Edition, O'Reilly, 2005.
- Q. Ni, L. Romdhani, and T. Turletti, "A Survey of QoS Enhancements for IEEE 802.11 Wireless LAN," *Journal of Wireless Communications and Mobile Computing*, Vol. 4, Issue 5, pp. 547-577, Aug. 2004.
- Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan, "Characterizing User Behavior and Network Performance in a Public Wireless LAN," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, Issue 1, June 2002.
- D. Pilosof, R. Ramjee, D. Raz, Y. Shavitt, and P. Sinha, *Understanding TCP Fairness over Wireless LAN*, INFOCOM, 2003.

以下是标准文档、一个很好的教程，以及一篇引用率很高的有关蓝牙的论文。接下来是有关 WiMAX 引用率很高的论文和书籍。

- Bluetooth Specification Documents, http://www.bluetooth.com/English/Technology/Building/Pages/Specification.aspx.
- P. Bhagwat, "Bluetooth: Technology for Short-Range Wireless Apps," *IEEE Internet Computing*, Vol. 5, Issue 3, pp. 96-103, May/June 2001.
- A. Capone, M. Gerla, and R. Kapoor, "Efficient Polling Schemes for Bluetooth Piconets," IEEE International Conference on Communications, June 2001.
- Z. Abichar, Y. Peng, and J. M. Chang, "WiMAX: The Emergence of Wireless Broadband," *IT Professional*, Vol. 8, Issue 4, July 2006.
- Louti Nuaymi, *WiMAX: Technology for Broadband Wireless Access*, Wiley, 2007.

设备驱动程序

这是一本教你如何编写 Linux 设备驱动程序的优秀书籍。

- J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd Edition, O'Reilly, 2005.

常见问题解答

1. 在以太网之上的 IP 中，字节顺序和位顺序分别是什么？

   答：字节顺序：高字节顺序，即首先传送高字节。位顺序：低位顺序，即首先发送最低位。

2. 为什么 FCS 放在尾部？而 IP 校验和要放在头部？

   答：FCS：由软件计算，在运行时添加和检查。

   IP 校验和：通常是由软件计算、存储和处理。

3. 为什么大的带宽延迟乘积（BDP）不利于 CSMA/CD？

   答：与长的链路相比，大的 BDP 意味着小帧。当一个小帧经过一条长链路传播时，其他站点保持闲置时，这意味着低的链路效率。

4. 在半双工千兆以太网中的问题是什么？

   答：传输一个最小帧的时间可能要小于往返传播时间。那么冲突检测就不能及时终止一次冲突的传输，也就是说，在发送站点侦测到冲突之前就已经结束传输。

5. 在千兆以太网传输时，最小帧的长度是多少（以米为单位）？

   答：64 × 8/109 × 2 × 108 = 25.6 米。

6. 为什么 CSMA/CD 不能用于无线局域网？

   答：如果由于终端隐藏而不被发送者看见，那么在接收器上的冲突就不会被发送者检测到。因此，CD 在这里就不起作用。此外，发送者不能一边传输一边侦听。

7. RTS/CTS 机制为 CSMA/CA 的无线局域网解决什么问题呢？

   答：当数据帧被接收者接收时，通过让接收者周围的终端保持沉默（收到 CTS 之后）的方式，它解决了隐藏终端问题。

8. 冲突域、广播域和 VLAN 之间的区别是什么？（描述它们的定义，它们包括的范围，以及它们是否可以重叠。）

   答：冲突域：在这一域内两个站点不能同时成功地传输；同样是用于集线器中的广播域，但在交换机中减少为一个端口。

   广播域：广播帧将被这个域内的所有站点接收；它也是集线器中的冲突域，但是交换机中的一组端口。

   VLAN：将一台交换机或一组交换机人为地分割成广播域。

9. 将第 2 层桥接与第 3 层路由进行对比（比较它们的转发机制、管理和可扩展性）。

   答：桥接：通过洪泛或自学习表，即插即用，限制为数千台设备。

   路由：通过全局或局部信息表，需要配置，可扩展。

10. 在一个大的园区网中能够实现第 2 层桥接吗？为什么？

    答：在园区网的每一台桥接交换机需要学习和记忆园区网上的所有主机，这就需要一张大的表。与此同时，当还未学习所有主机时就会发生频繁的洪泛。

11. 为什么我们说，网桥对主机是透明的，而路由器则不是？

    答：在桥接中，无论目的地是否在同一个局域网，主机都会照样发送帧。在路由中，如果目的地不在同一子网，主机会显式地将分组发送到默认的路由上。因此，主机知道路由器，但不知道网桥。

12. 在透明网桥中，为什么需要一个生成树？

    答：为了消除拓扑结构中的回路，回路会使网桥在不能做出正确判断时导致帧的循环。

13. 我们如何在 IC 中设计 MAC？（描述一般的设计流程和编程中所使用的变量。）

    答：设计流程：带有输入/输出信号的框图→每个框/模块的状态机→Verilog 或 VDHL 并行硬件编程→集成和模拟电路→版图（layout）和下线送交制造（tape-out）。

    变量：程序输出变量/信号作为并行函数的输入变量/信号和局部变量/信号。

14. 驱动程序是如何发送和接收帧？（利用硬件与中断处理描述外出和进入数据包的处理。）

答：外出数据包处理：利用远程 DMA 将帧移动到接口卡，将命令写入寄存器，并且读取状态寄存器的中断处理程序，并发送后续的帧。

进入数据包处理：并且读取状态寄存器一断的中断处理程序，并利用远程 DMA 将到帧移动到主存中。

15. 当网络适配器驱动程序探测到它时它想要找什么？为了什么？哪个中断可能与效率较低的网络适配器的驱动程序？

答：1）IRQ 号：将中断处理程序绑定到一个硬件号。

2）I/O 端口号：将硬件寄存器映射到 I/O 端口区域用来读状态和写命令。

3）由于队列大，传输完成或异常传输的硬件中断。

练习

动手练习

1. 阅读下面的四个文件，了解 IEEE 标准是如何产生的，写一个有关标准化过程的总结。

   a. 10 千兆以太网摘要："10 千兆以太网技术概述"，白皮书，2001 年 9 月，http://www.10gea.org。

   b. http://www.ieee802.org/3/efm/public/sep01/gigella_1_0901.pdf。

2. 年可以从 http://standards.ieee.org/getieee802/下载 IEEE 802 标准。写下以下项目的主旨与范围：802.1w，802.3ac，802.15，802.16 和 802.17。

3. 找到你的 PC 机上的网络接口卡的 MAC 地址。或者 http://standards.ieee.org/regauth/oui/oui.txt，与已注册的 OUI 进行比较。

4. 使用嗅探器或类似的软件，看看你收到的以太网帧"类型"字段中有多少种"协议类型"。如果有，它们分别属于什么传输/应用层协议？

5. 找出你网络接口卡是运行在半双工还是运行在全双工模式。

6. 描述以下每段的源代码：

   a. HDLC　b. PPPoE　c. 无线局域网　d. 蓝牙

   重新在每段实现中的每个主函数的目的，并绘制带有函数名的流程图来表示执行流程。

7. 经过阅读编译并选择一种将要被实现的驱动程序后，我们该如何编译驱动程序，安装驱动程序，并运行这些模块？请编写一个小模块来验证你到答案。还要清楚什么样的命令能够使安装和拆除模块。如何验证一个已成功安装你的模块？（提示：阅读 insmod（8），rmmod（8），lsmod（8））

8. 数据包的生命历等：测试一个数据包在驱动程序、DMA 和 CSMA/CD 适配器上分别花费多少时间。（你可以使用 <asm/msr.h> 中定义过的 "rdtscll" 得到过去的 CPU 时钟周期。）

书面练习

1. 我们知道 32 位的 IPv4 地址可能不够长。那么 48 位 MAC 地址足够长吗？写一简短的讨论来证明你的答案。

2. 阅读 RFC 1071 和 RFC 1624，看看如何计算 IP 校验和。然后动手利用 7 面字的分别并行练习：

   0x3617　0x6720　0x2148　0x8912　0x2345　0x7863　0x8076

   如果上面的第一个字改成 0x3638 呢？RFC 在 http://www.ietf.org/rfc.html 上查找。

3. 计算 CRC 码，假定消息为 11010111011，（产生多项式）模式为 10011。验证代码是正确的。

4. 为什么目的地址字段通常位于帧的头部，而 FCS 字段位于帧的尾部？

5. 如果我们将大最小以太网帧，将会有哪些优点和缺点？

6. 假设你的数据有效载荷加了 40 字节的 IP 和 TCP 头部。以及每帧都加以太网 8 字节记帧，那么 100Mbps 以太网可以每秒钟携带多少位的数据有效载荷？

7. 在交换机转发前，是否应该重新计算出达帧的 FCS？

8. 在以太网帧中有一个可选的优先级标记，但不经常使用。为什么会这样呢？

9. 为什么以太网不能实现一个反压码（如流动窗口协议）流量控制机制？

10. 如果网络接口卡在一个共享网络中以全双工模式运行，会发生什么？
11. 交换机的每个端口都应该有自己的 MAC 地址吗？加以讨论。
12. 假设在交换机地址表中的每个表项需要记录 MAC 地址、8 位端口号、2 位老化信息。如果该表可以记录 4096 项，那么最低的内存大小是多少？
13. 假设 5 个连续的 1 之后用 0 进行位填充。假设位置中的 0 和 1 的概率是相等的，并且是随机发生的，那么位填充方案的传输开销是什么？（提示：列出一个递归公式 $f(n)$ 以便首先找到一个 n 位字串期望的开销位数。）
14. 编写一个模拟程序以验证第 13 题的数字答案是正确的。
15. 在 1000BASE-X 中，一个 64 字节的帧在传送之前首先使用 8B/10B 分组编码。假设传播速度为 2×10^8，那么以“米”计的项“长度”有多少？（假设电缆长为 500m）。
16. 两个站点第一次冲突后，重试 5 次解决冲突的概率有多少？（假设在冲突域中只有两个站点。）
17. 一台具有 16 个快速以太网（100Mbps）端口的交换机可以处理的最大帧数是多少？如果每个端口都运行在全双工模式下？
18. 一个 CPU 以 800 MIPS 执行指令。每次可以复制 64 位数据，对于每 64 位字复制需要 6 条指令。如果传入的帧需要复制两次，那么系统一行最多可处理多少位速率？（假设所有指令运行在全 800MIPS 速率。）
19. 一个 1500 字节的帧沿路径经过 5 台交换机。每条链路具有带宽为 100Mbps，长度为 100m，传播速度为 2×10^8 m/s。假设在每台交换机上的排队和处理延迟为 5m，那么对于该数据包（帧）的近似端到端延迟是多少？
20. 如果误码率是 10^{-9}，那么 100 个 1000 字节的帧的平均误差概率是多少？

互联网协议层

互联网协议（IP）层，又称为OSI模型的第3层或网络层。它提供了一种主机到主机的传输服务。这是互联网协议栈中最重要的一层，并且比链路层复杂得多。因为它在任意两台主机之间提供连通性，而两台主机可能相距数千里之遥。IP层的最大挑战是如何在两台主机之间有效地提供可扩展的连通性。具体而言，它面临着连通性、可扩展性和高效的资源共享问题。首先，关键问题是如何将布全球网络中任意位置的两台主机连接起来。其次，将分布在世界各地的数十亿台主机连接起来，需要可扩展的寻址、路由和分组转发机制。最后，中间设备（如路由器）所具有的处理能力和带宽等有限资源必须有效地共享，从而为终端用户提供满意的服务。

为了提供主机到主机的传输服务还需要控制平面机制和数据平面机制。控制平面处理控制协议，确定分组应该如何处理。例如，作为IP层最重要功能之一的路由，主要就是为了能在任意两台主机之间找到一条路径，并在路由器专门设计的数据结构（称为路由表或转发表）中存储路由信息。另一方面，数据平面解决如何处理分组。例如，IP层的另一个重要功能——转发，是根据路由表将分组从路由器进入网络接口传送到外出网络接口上。还需要其他机制支持连通性功能，如地址配置、地址翻译和错误报告。本章将介绍互联网中使用的控制平面和数据平面的所有主要机制。这些机制提供主机到主机的连接服务。

本章的组织结构：互联网协议层的设计问题，将在4.1节中讨论。数据平面和控制平面的机制及其开源实现，将在后面的章节中描述。对于数据平面机制，我们介绍互联网协议版本4（IPv4），并说明它是如何以可扩展的和有效的方式提供主机到主机的服务。在4.2节结束时，我们将说明网络地址翻译（NAT）的机制，它被认为是解决IPv4地址短缺问题的一个暂时的解决方案。在4.3节中，将介绍互联网协议版本6（IPv6）以解决IPv4中遇到的几个问题。

接下来的4节将讨论控制平面机制。在4.4节中，我们将学习地址等价机制，包括地址解析协议（ARP）和动态主机IP配置协议（DHCP）。在4.5节中介绍互联网错误处理协议。互联网控制报文协议（ICMP）。IP层最重要的控制机制是路由，即发现两台主机之间的路径。这些互联网路由协议将在4.6节中描述并说明如何以可扩展的方式进行路由。最后，在4.7节中，我们复习组播路由协议，即从点到点路由到多点到多点路由的一种方案。

4.1 一般问题

TCP/IP参考模型中的网络层或IP层的目的是从发送主机向接收主机传输分组。不同于链路层提供的服务（通信是在两台相邻主机之间实现的），网络层提供的服务允许任意两台主机之间的通信，无论它们之间有多远。这种连通性的要求引入了一个一般性的问题，即如何通过链路层技术将网络连接起来，如何在全球范围确定一台主机，以及如何在两台主机之间找到一条路径并能沿路径转发分组。对于这些问题的解决方案必须具有很强的可扩展性，以便能够容纳数十亿台主机之间的连接。最后，还需要解决如何有效地共享有限的资源（如带宽）。

4.1.1 连通性问题

网际互联

为了能够从一台主机向另一台主机传输分组，连通性当然是基本的要求。为了实现这种主机级的连通性，需要解决许多问题。首先，主机如何连接起来？主机可能通过不同的链路层技术（如以太网或无线局域网）连接到网络上。正如我们在第3章中所讲的，对这些链路层技术的基本限制就是距离，也就是说，一个局域网的覆盖范围不能超过一定的距离。还有一个限制就是可以共享局域网带宽的节点数。因此，它需要大量的局域网和成千上万的设备，将分散在世界各地的主机组织起来。一组连接起

来的网络称为互联网络，或简称为互联网。目前广泛使用的全球互联网络被称为“因特网”。将网络互连成互联网的互联设备通常称为路由器。任意两台主机之间的通信可以通过路由器将局域网连接成一个全球互联网络。图4-1显示了具有路由器和包含各种局域网的例子。

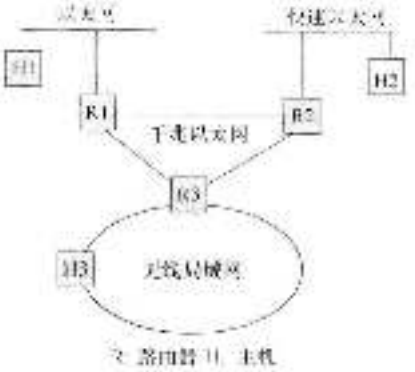


图4-1 互联网的例子

寻址

在网络层，连通性的第二个问题是如何在全球互联网中确定某台主机，这实际上是寻址问题。与链路层的寻址不同，在网络层的主机地址需要网络所在地的全球标识符。换句话说，一个主机地址需要确定主机所属的网络和主机本身。这种地址称为层次化地址。为一台主机分配一个网络层地址也会出现一个新的问题：一台主机除了它的链路地址外，还将为每个网络接口卡分配一个网络地址（或更多地址）。因此，在这两个层次之间的地址映射就会成为一个新的问题。与寻址有关的问题就是如何将网络层地址分配给主机。在现实中，它既可以自动地也可以手动地完成。如果它是自动地完成的，地址就可以静态地或动态地分配。在大多数情况下，一台主机也是自动和动态地配置其地址。因此就需要一个动态主机配置协议。

路由和转发

假定可以确定一台主机，接下来的问题就是如何找到从一台主机向另一台主机传输分组的一条路径。路径由组织路由器的序列组成。查找路径并沿路径传输分组的问题分别称为路由和转发。在控制平面上运行的路由协议负责在两台主机（或两个网络）之间查找路径。构建路由表是为了记录路由的结果。当一个分组到达某台路由器时，按照为该分组的目的地址的路由表表项，它会被转发到路由路径上的下一跳。这里，我们将路由和转发区别得很清晰：路由是通过路由协议来实现的，它需要交换路由消息并计算最短的路径；而转发是通过主机或路由器查找路由表并查找转发分组最合适的网络接口来运行。

4.1.2 可扩展性问题

当我们考虑连接到互联网上的主机和网络数量时，可扩展性对网际互联很重要。可扩展性对路由和转发特别重要——因为在数十亿台主机中有效地找到一条到达一台主机的路径非常具有挑战性。我们将在本章中看到如何使用网络的层次结构来解决可扩展性问题。在互联网上，将节点分组成子网络，通常就是子网（subnets）。每个子网代表一个逻辑广播域，因此子网内的所有主机可以直接相互发送分组而不需要路由器的帮助。多个子网组成域。域内路由和域间路由是由不同的路由协议分别来完成的，路由表中的表项可能代表一个子网或一个域。

行动准则：桥接和路由

桥接和路由有很多的相似之处。比如，两者都可以用于连接两个或多个局域网，两者都能查表转发分组。然而，在其他方面它们有着很大的区别。首先，网桥是各种网络设备的组件，无论是两端口还是多端口的。

分层：网络是一种链路层设备，而路由器是一种网络层设备。网桥根据链路层的帧头部信息（如目的MAC地址）转发帧，而路由器根据网络层头部信息（如IP的地址）转发分组。

表：网桥一般是通过透明地自学习来构建转发表，而路由器是通过运行路由协议显式地构建路由表。当多台网桥连接时，网桥还需要运行一种生成树协议以便避免出现环路。

冲突域与广播域：网桥用来隔离冲突域，而路由器用于分隔广播域。冲突域指的是在一个网段中，主机共享相同的传输介质。如果有两台以上的分组同时发送时就可能发生冲突。一个n端口网桥通过在n个冲突域上分隔一个冲突域可以将一个冲突域分成n个。然而，所有连接冲突域的处在同一广播域中，除非创建了虚拟局域网。广播域是指在网络内的所有节点可以经过桥路层的广播互相通信。从层

联网协议的角度来看，一个广播域对应于一个 IP 子网。一个 n 网口路由器能够将一个广播域分成 n 个广播域。当在骨干网桥上创建虚拟局域网内，广播域的概念就会得到非常重要了。在同一个虚拟局域网内的所有主机，不管其中包含多少台网桥，都处在同一广播域内并且能够接收到所有数据链路层的广播。另一方面，在两个不同虚拟局域网中的两台主机只有通过路由器才能通信，即使它们连接到同一网桥也是如此。

可扩展性：由于广播的发展，桥接与路由相比可扩展性不高。正如首面所述，由一台或多台网桥连接的主机仍处在同一个广播域内。它们应该能够接收到广播。因此，如果将数百万台主机桥接起来，就很容易把广播消息发送到所有主机。同时，当某个 MAC 地址连接有域学习到转发表中时，将会使用洪泛的方法转发帧，这在大型互联网络中效率会非常低。

关于路由，有多个问题需要解决，如第 1 章所述。现在应该清楚，考虑到可扩展性需求，为互联网路由所选择的解决方案应该是迷路的路由，最短路径路由是按照每个 E 的第一网络粒度来完成的。如何计算路径以及如何收集路由信息也取决于可扩展性。对于域内路由，可扩展性不成问题，简化往往会更加重要。因此，域内路由的目标之一就是有效地共享资源，这是通过找到每一对源到目的地之间的最短路径实现的。路由信息既可以仅通过相邻路由器之间信息交换也可以向同一域内的所有路由器洪泛路由信息来实现。因此，域内路由决策（求最短路径）既可以根据邻分路由信息也可根据全局路由信息。另一方面，对于域间路由，可扩展性比最优性更重要。域间路由需要考虑的另一个问题是，由不同的域管理员做出的管理策略，他们可能希望禁止某些流量通过某些域。因此，基于策略的路由比有效的资源共享更重要。对于可扩展性和基于策略的路由，域间路由通常仅交换相邻路由器的汇总信息并且根据后部路由信息进行路由决策。我们将在 4.6 节更详细地讨论这里所提出的路由问题。

4.1.3 资源共享问题

无状态的和不可靠的

最后，让我们解决资源共享问题。在互联网上，资源可以自由共享而没有任何网络层的控制。互联网协议为上层提供一种无连接服务模型。在无连接服务模型之下，分组需要在它们的头部携带足够的信息以便使中间路由器能够正确地路由并将分组转发到目的地。因此，在发送分组之前就不需要建立机制。这是共享网络资源最简单的方法。无连接服务模型也意味着尽最大努力服务，虽然它不必如此。当转发分组时，路由器只是根据路由表尽最大努力地将分组转发到目的地。如果出错了，如分组丢失、未能到达目的地，或发送后失去顺序，那么网络并不做任何事情去解决问题。网络只是尽最大努力发送分组。这也意味着由网络层提供的服务是不可靠的。

因为网络层提供的服务是不可靠的，所以就需要一种强调根本机制去通知源和源主机的上一层。发出的错误报告包括如何传送错误信息、如何确定错误的类型、如何让源知道哪些分组造成错误、怎样在源处理错误信息，以及是否应该限制错误消息所使用的带宽。

资源共享的最后问题是安全。安全问题有多个方面。第一个问题就是处理谁有权访问网络资源。数据安全涉及加密分组保护数据不被窃听。最后，还有系统安全问题，它保护主机不被入侵或病毒攻击。我们将这部分的讨论放到第 8 章进行，虽然有些人会认为访问控制和数据安全可以在网络层解决。

4.1.4 IP 层协议和分组流概述

图 4-2 给出了本章所讨论的协议路标。当主机接通电源时，可以使用 DHCP 协议来查其 IP 地址、子网掩码、默认路由器等。主机正确配置之后，复从 TCP 或 UDP 等上层发送一个分组然后自由 IP 层处理，以确定如何转发分组。不管分组是否能够直接发送到位于同一子网的接收方还是到转发分组的路由器，都要使用 ARP 协议将接收方的 IP 地址解决成它的链路层地址（MAC）。如果在 IP 处理中有一个错误，或利用 ICMP 协议向产生最初 IP 分组的源主机发送出错信息。如果将分组发送到路由器，可常是默认路由器，路由器根据分组的目的地址和路由表中的路由信息将分组转发。路由表是由运行

在路由器上的路由协议来维护的。当分组到达接收方式，分组就被接收并由网络协议处理，如果没有错误，它就被发送到相应的上层协议。如果因为隐私或安全原因而使用了私有地址，那么就使用网络地址翻译协议（NAT）翻译IP地址和IP分组的传输层标识符（TCP/UDP端口号）以便实现全球互联网的连接。

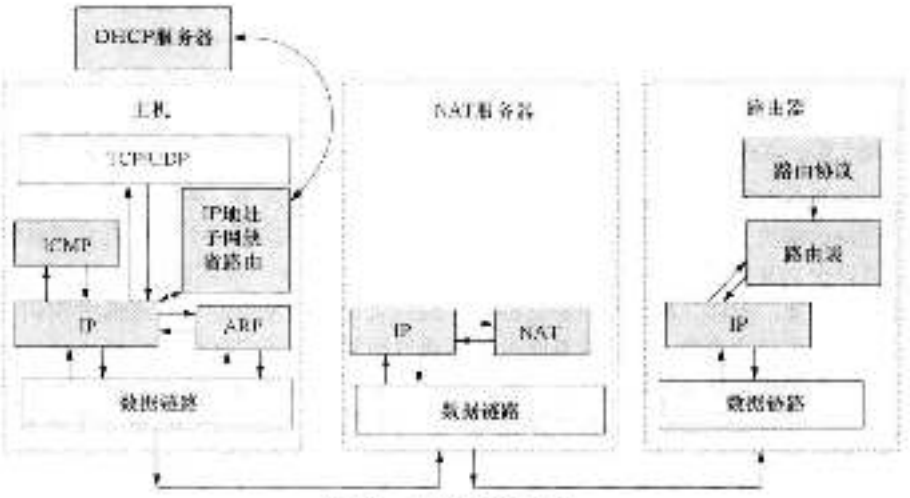


图4-2 本章讨论的协议

开源实现4.1：调用图中的IP层分组流

概述

IP层位于互联网协议栈中的链路层之上和传输层之下。利用分层方法，就应该在相邻层之间提供接口。因此，IP层的接口分别包括与链路层的和传输层的。正如在第3章中所述，我们通过分组的接收路径和分组的发送传输路径研究这些接口。在接收路径中，要从链路层接收分组，然后传递给传输层协议，包括TCP、UDP和原始IP套接字接口。在发送传输路径上，从一种传输层协议接收分组然后再传递给链路层。

接收路径

网络接口卡收到一帧后将触发一个中断，中断处理程序调用net_rx_action()处理传入的帧。如第3章所述，调用网络层协议处理程序的实际函数是natif_receive_skb()。然后调用backlog_dev_poll()进行注册以便于处理以下的接收操作。如图4-3所示，当注册到sk_buff中的协议类型是IP协议时，将调用ip_rcv()作为协议处理程序。分组然后经过多个IP层协议函数，这些将在本章稍后讨论。如果分组是发向本地主机的，那么就调用ip_local_deliver()，然后调用ip_local_deliver_finish()将分组发送到传输层协议处理程序。处理程序既可以是raw_v4_input()、udp_rcv()，也可以是tcp_v4_rcv()，具体取决于上层协议是IP套接字接口、UDP协议，还是TCP协议。

传输路径

传输路径也在图4-3中显示。上层协议将分组推到IP层后就进入它的队列。调用ip_append_data()、ip_append_page()或ip_queue_xmit()将分组发送到IP层，具体取决于所使用的传输层协议。为了避免太多的小分组发送，首两个函数首先在一个临时缓存上存储数据，然后针对临时缓冲区调用ip_push_pending_frames()，实际上也就是将数据打包成合适大小的分组。所有这些函数都将调用dst_output()，随后调用虚拟函数skb->dst->output()注册到sk_buff以便如果网络层协议为IP就调用网络层处理程序ip_output()。如果不需要分段，那么ip_finish_output2()将通过net_tx_action()将分组发送到链路层，如第3章所述。

练习

沿着接收路径和传输路径跟踪源代码以便观察在这两条路径上的函数调用初节。

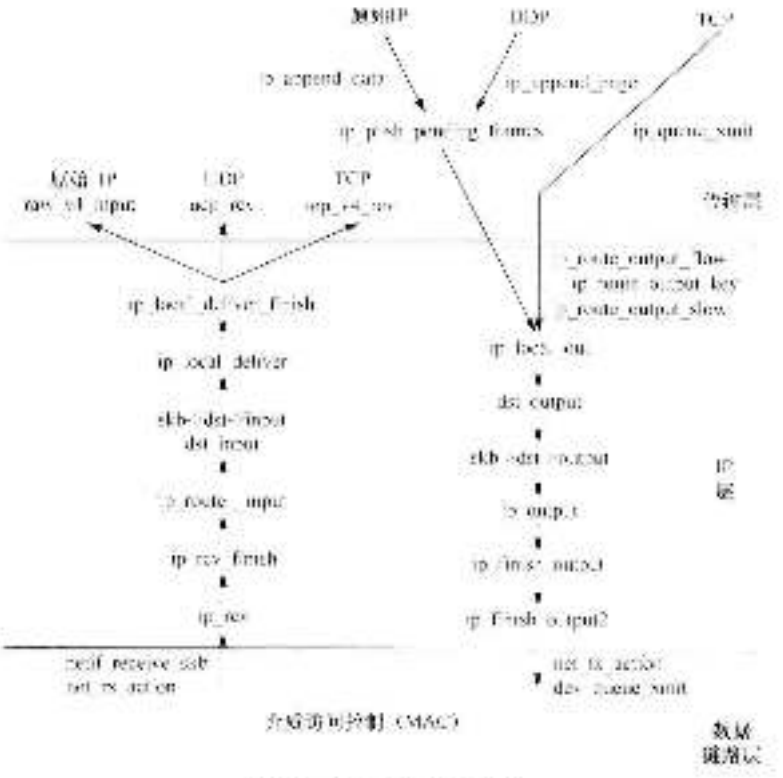


图 4-3 调用图中的分组流

性能问题：IP 层内的延迟

图 4-4 显示传输 64 字节的 ICMP 分组时，重要的 IP 层函数的延迟分解。总的延迟约 4.42μs，指数函数 ip_finish_output2() 占总处理时间的 50% 以上。正如在开源实现 4.1 中所述，ip_finish_output2() 将分组发送到数路层。在调用 net_tx_action() 前，它需要将以太网头部添加到分组的前面。这种添加任务调用内存复制并因此要比其他函数消耗更多的时间。

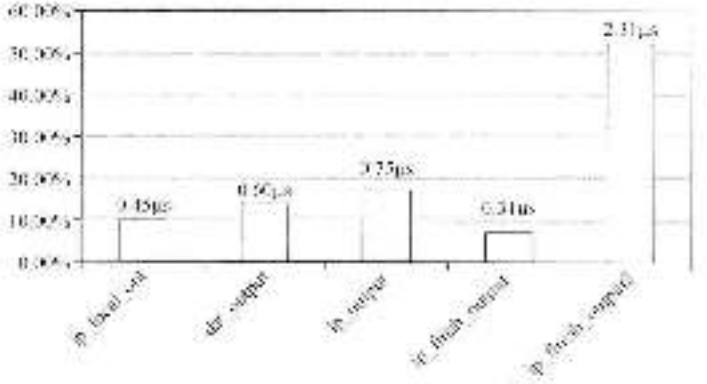


图 4-4 在 IP 层上传输 ICMP 分组产生的延迟

图 4-5 描述了 IP 层中分组接收函数的延迟。时间消耗多的前 4 名函数为 ip_route_input()（26%）、ip_local_deliver_finish()（24%）、ip_rev()（17%）和 ip_rev_finish()（16%）。ip_route_input() 在查询路由表时消耗时间。ip_local_deliver_finish() 分析 IP 头部，通过散列表查找以接收到分组的正确传输层协议处理程序，然后再把它传送给处理程序。ip_rev() 验证 IP 分组中的头部校验和字段。最后，ip_rev_finish() 更新路由表的统计值。

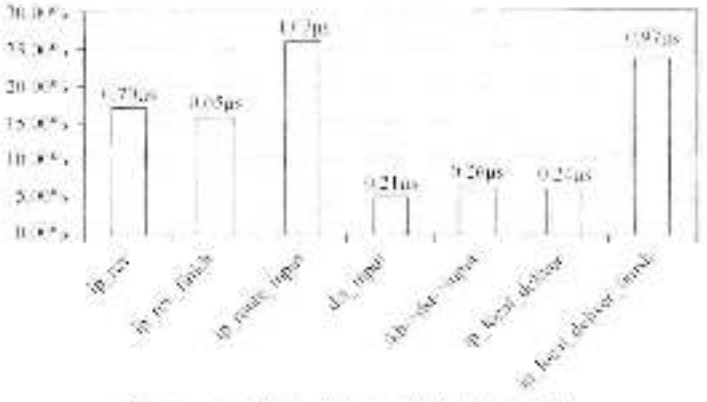


图4-5 IP层中接收ICMP分组产生的延迟

4.2 数据平面协议：互联网协议

在本节中，我们首先学习当前版本的互联网协议——IPv4。在IPv4中，因为安全和地址耗尽的原因而采用一种特殊类型的地址，称为私有IP地址。在4.2.2节，我们学习网络地址翻译（NAT）协议，它让使用私有地址的主机能够访问互联网。

4.2.1 互联网协议版本4

互联网协议，常称为IP协议，是互联网中使用的一种关键机制以便提供主机间的传输服务。IP协议有2个版本：协议版本4（IPv4），目前互联网使用的；协议版本6（IPv6），用于下一代互联网。IPv4协议定义在RFC 791中，而IPv6定义在RFC2460中。我们首先介绍IP寻址模型，并利用该模型解释互联网是如何提供连通性的。

IPv4寻址

在构建主机到主机之间连通性时，首先需要有一个全球性的和唯一的标识主机的寻址方案。主机经过一个接口（如以太网接口卡）连接到一个网络上。某些主机和路由器可以配备多个网络接口。对于每个网络接口，需要使用一个地址来标识接口以发送和接收分组。为了能够在数十亿主机之间找到一个网络接口，我们需要一种层次化的结构来全局地组织和定位IP地址。IP地址的层次化结构与邮政地址非常相似。我们家庭的邮政地址是由编号、街道、城市和国家组成。所以地局能够很容易地确定我们的邮件需要发送到哪里。同样，IP寻址方案具有层次化的结构，以便中间路由器很容易地识别分组应该发送到的网络。

每个IP地址有32位（4字节）长，并由句部分组成：网络地址和主机标识符（ID）。通常，将一个地址写成点分十进制标记法。例如，在图4-6中，IP地址的前8位是10001100，这相当于十进制的140。IP地址的4个十进制数是由点分开的。

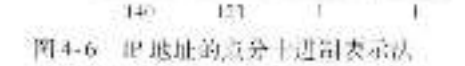


图4-6 IP地址的点分十进制表示法

IP使用一种有类的寻址方案。一共定义了5类地址，如图4-7所示。所有地址类都有一个网络地址和一个主机标识符，但它们在两个部分的长度会有所不同。A类地址具有8位网络地址和24位主机标识符。对于IPv4，互联网可以容纳2^7个A类网络地址，每个A类网络容纳高达$2^{24}-2$台主机（2个特殊地址保留，参见下边的讨论）。同样，互联网可以容纳高达2^{14}个B类网络和2^{21}个C类网络。一个B类网络和一个C类网络分别最多可以容纳$2^{16}-2$台主机和2^8-2台主机。D类地址为组播地址，允许多点到多点的传输。我们将在4.7节讨论IP组播。第五类，从地址前缀11110开始，预留用于将来使用。



图 4-7 有类 IPv4 地址格式

给定有类地址的起始位，如图 4-7 所示，每一类地址的范围也是固定的。A 类地址的取值范围从 0.0.0.0～127.255.255.255（注意，0.0.0.0/8 保留给本地，而 127.0.0.0/8 保留用于回环测试）。从 128.0.0.0～191.255.255.255 以及从 192.0.0.0～223.255.255.255 分别用于 B 类和 C 类地址。D 类地址的范围为从 224.0.0.0～239.0.0.0。最后，从 240.0.0.0～255.255.255.255 往往被丢弃使用。（注意，255.255.255.255 是子网中的一个广播地址。）

每类中都有一些地址是被用于特殊用途。如果主机标识符为 0，就用来代表一个子网。例如，140.123.101.0 是一个 B 类子网地址。另一方面，如果所有的主机标识符都是 1，它就用于在该子网中的广播。最后，当源主机还不知道它自己的地址时，IP 地址 255.255.255.255 用于在 IP 子网中广播分组。例如，当主机需要获取 DHCP 服务器获取其 IP 地址时就是如此。我们将在 4.4 节中讨论 DHCP 协议。

IP 子网划分

一个 IP 地址的网络地址是被用来唯一地标识一个物理网络。然而，一个物理网络通常采用局域网技术构建，如第 3 章所述。对于一个 A 类或 B 类网络，大量的主机标识符通常大于任何局域网技术所能够支持的主机数。因此，在 A 类或 B 类网络中期待只有一个物理网络或局域网是不切实际的。因此，一个拥有 A 类或 B 类甚至 C 类网络地址的组织往往把自己的网络分成多个子网。在逻辑上，在同一子网内的两台主机能够应用链路层技术直接相互发送分组，而不必通过路由器传递。为了维护 IP 地址的层次化结构，所有在同一子网内的主机在其 IP 地址中必须具有相同的前缀（最左边的位）。因此，部分主机标识符是用来表示 A 类、B 类、C 类网络中的子网地址，如图 4-8 所示。用来表示子网地址的位数取决于该组织管理员需要的子网数和子网内的主机数。例如，具有 8 位子网地址和 8 位主机标识符的 B 类地址将有 2^8 个子网，每个子网具有 2^8-2 台主机。

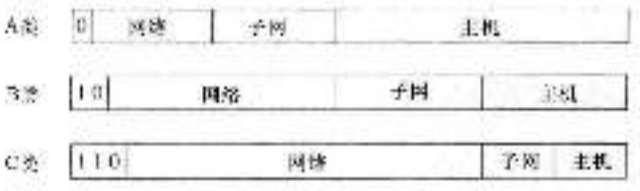


图 4-8 IP 子网寻址

为了确定两台主机是否在同一个子网内，将子网掩码标记应用于子网划分。子网掩码指示一个 IP 地址最左边的地址位长度，该用作子网地址。继续前面的 B 类地址子网划分的例子，子网地址就是 32 位 IP 地址的最左边 24 位。利用两种标记法来表示子网掩码。首先，我们可以使用一个 32 位的字串，其中子网地址部分和主机部分相对应地都是由 1 和 0 来表示子网掩码，如我们例子中的 255.255.255.0。我们也可以将一个 IP 地址表示为 140.123.101.0/24，这里/24 表示子网掩码是 24 位长。

因此通常，一个网络由多个子网组成，在同一个子网中的主机具有相同的子网掩码和子网地址。例如，在图 4-9 中，有 5 台主机连接到 3 个子网上。分别为 140.123.1.0、140.123.2.0、140.123.3.0、

主机 H1 和 H2 连接到同一个子网，因此具有相同的子网地址，即 140.123.1.0。子网之间利用路由器（R1～R3）连接起来形成一个互联网络。连接到一个子网的路由器网络接口也具有与同一个子网内主机相同的子网掩码和子网地址。值得注意的是，每台路由器通常配备了多个网络接口卡。其中有些将路由器与子网内的主机连接起来；然而，另一些用于连接其他路由器以便形成一个面向交换或分布式的骨干网络，如图 4-9 所示的 140.123.250.0 子网。

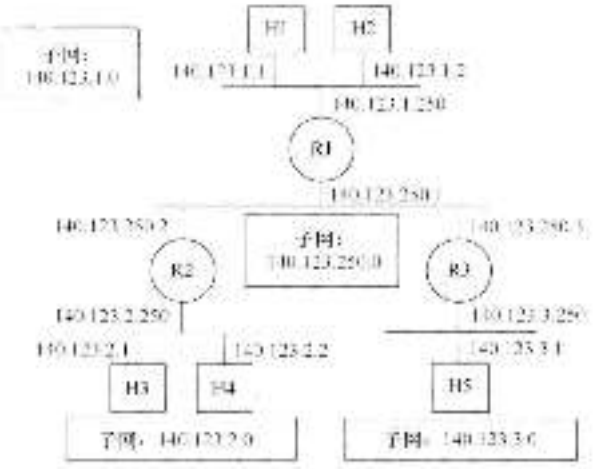


图 4-9 IP 子网划分的例子

CIDR 地址

有关 IP 地址存在一些问题。首先，由于网络地址的固定长度，为中等规模组织（如有多达 2000 台主机的一个组织）分配 IP 地址时就很困难。一个 C 类网络地址对于这样的组织太小，因为它最多只能支持 254 台主机；而 B 类网络地址太大，因为它将有超过 63 000 个以上的地址未能使用。一种可能的解决办法就是为组织分配多个 C 类网络地址，但是使用该解决方案，路由转发的可扩展性存在问题。使用该类 IP 网络时，每个 C 类网络地址在骨干路由器的路由表中占有一项。但是，对于具有多个 C 类网络地址的组织，与这些 C 类地址相关的所有路由表表项都应该指向到达组织的同一个路由路径上。这会导致骨干路由器上的路由表很大的问题，当存在很多 C 类网络地址时，路由表中的许多表项会携带相同的路由信息。

因此，为了解决该问题无类域间路由（CIDR）策略应运而生。利用 CIDR，IP 地址的网络部分可以具有任意长度。一个中等规模的组织将被分配一个 IP 地址块，该块常是连续的 C 类网络地址。例如，一个带有 2000 台主机的组织可以分配一块从 194.24.0.0 ～ 194.24.7.255 的地址。子网掩码为 255.255.248.0 或 194.24.0.0/21。即前 21 位用来指定组织的网络地址。骨干路由器仅需要一个路由表项记录到组织的网络路由，如图 4-10 所示。组织内的 IP 子网划分可以如上所述来完成。

目的地址	下一跳
194.24.0.0	19.1.1.250
194.24.1.0	19.1.1.250
194.24.2.0	19.1.1.250
194.24.3.0	19.1.1.250
194.24.4.0	19.1.1.250
194.24.5.0	19.1.1.250
194.24.6.0	19.1.1.250
194.24.7.0	19.1.1.250

→

目的地址	前缀	下一跳
……	……	……
194.24.0.0	/21	19.1.1.250
……	……	……

图 4-10 使用与不使用 CIDR 的路由表

分组转发

回顾4.1节所述，转发就是一种从上层或网络接口接收分组并且通过适当的网络接口将分组发送出去的过程。主机和路由器都需要转发分组。对于主机，来自上层的分组需要在它的一个网络接口上发送出去。对于路由器，从网络接口上接收的分组需要转发到其他网络接口上。IP转发过程的核心思想是，如果要转发分组的目的地址与转发节点位于同一子网，则将分组直接发送到目的地址；否则，转发节点需要查找路由表以便找到转发分组相应的下一跳路由器，然后直接将分组发送到下一跳路由器。路由表表项由（目的地址/子网掩码，下一跳）对组成，但它也可能包含额外的信息。具体细节依赖于运行的路由协议类型而定。目的地址通常表示成网络地址的形式，例如194.24.0.0/21。下一跳段可以是一个路由器IP地址但也可以是一个网络接口，这样路由器必须在同一子网作为网络接口，以便它可以直接通信。通常，有一个表项记录一个默认路由器作为目的地址0.0.0.0/0。如果分组的目的地址不匹配路由表中的任何项目，那么就将它转发到默认路由器。

我们可以从写个方面描述分组转发算法。首先，对于一台主机，我们考虑从上层（如TCP）传来一个分组需要发送到目的地址的例子。特别是，我们考虑最常见的情况，即主机只有一个网络接口卡和一台默认路由器的情况。在这种情况下，IP转发算法类似如下：

```
If the packet is to be delivered to the same host
    Deliver the packet to an upper-layer protocol
Else If (NetworkAddress of the destination == My subnet
address)
    Transmit the packet directly ... the destination
Else
    Look up the routing table
    Deliver the packet to the default router
End If
```

现在，让我们考虑具有转发能力的路由器或主机从一个网络接口接收分组的情况。在这种情况下，分组可能会被转发到适当网络接口的目的地址，或者如果目的地址是主机本身，就将分组传给本机的上层协议。转发算法如下所示。

```
If the packet is to be delivered to the upper layer
    Deliver the packet to an upper-layer protocol
Else Look up the routing table
    If the packet is to be delivered to a directly
connected subnet
      Deliver the packet directly to the destination
    Else
      Deliver the packet to a next hop router
    End If
End If
```

在前面两种算法中的三个运算值得进一步讨论。第一，转发主机如何获取目的网络地址以及节点本身是否与目的地址直接连接？使用下列运算可以轻松实现这个布尔判断：

```
If(HostIP^DestinationIP) & SubnetMask)==0
```

这里^是按位异或运算，&是按位与运算。

第二，在子网内将分组转发到目的地需要利用目的MAC地址形成一个第2层的帧。这涉及地址解析操作，我们将在4.4节中讲述。最后，查找路由表的步骤描述如下。

路由表查找

正如我们已经看到的，路由表查找是IP转发算法关键的操作。由于有了CIDR寻址，所以查找路由表已成了目前已知的最长前缀匹配问题。也就是说，查找与分组目的地址在长前缀相匹配的路由表项进行转发。考虑到在两个组织的情况。组织A拥有IP地址194.24.0.0～194.24.6.255。因为发送到该范围中任何地址的分组应该路由到同一网络接口，所以在路由表中它仅需要一个路由表项将分组路由到组织A。作为路由汇总的结果，组织A的路由表项的网络地址为194.24.0.0/21。组织B仅拥有一个C类网络地址，范围为194.24.7.0～194.24.7.255。因此，组织B的路由表项记录网络地址为194.24.7.0/24。现在，假设我们查找目的地址为194.24.7.10的路由表项。显然，目的地址匹配两个路由表项，即(194.24.7.10 194.24.0.0) & 8255.255.248.0) —— 0 和(194.24.7.10 194.24.7.0)& 255.255.255.0) —— 0。我们知道194.24.7.10属于组织B，因此应选择具有更长网络地址的路由表项，

194.24.7.0/24。详细研究这两种情况，我们发现194.24.7.10能够匹配194.24.7.0/24的前24位，但只匹配194.24.0.0/21的前21位。现在应该清楚为什么要采用最长前缀匹配。

最近，已经提出了最长前缀匹配的快速算法。在文献中，带有缓存、散列及基于硬件实现（并行算法，基于CAM或基于DRAM）的转发表就是一些知名的解决方案。在Linux中，查找算法主要是基于两级散列。传统的BSD实现使用trie数据结构。trie也称为前缀树，是一种有序树形数据结构。由于IP地址是一个位串，用于最长前缀匹配的trie是一个二进制trie，如图4-11所示。一台路由器首先构建一个字典，它由所有的路由前缀组成。然后就可以通过从字典中逐个地添加前缀到trie结构中来构建trie。在图4-11中带有×的节点，如果它对应于字典中的一个前缀，就在trie上携带着下一跳信息。在匹配目的地址最长前缀的搜索中，在trie上的每条边代表着一个二进制位串，指示搜索直到它不能继续向前为止。搜索终止的节点存储着下一跳信息作为最长前缀匹配的结果。例如，使用图4-10中的trie树搜索地址00001111的最长前缀匹配，我们从根开始，沿左树移动两次，最终在节点00*结束。由于地址的第三位和第四位是00，这与节点00*的任何子节点不相匹配。因此，最长前缀匹配就是前缀00*。

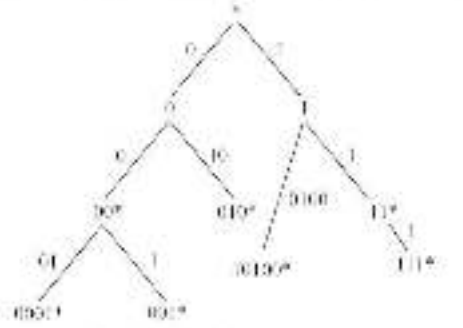


图4-11 具有前缀{00*，010*，11*，0001*，001*，10100*，111*}的trie的例子

开源实现4.2：IPv4分组转发

概述

现在让我们查看在Linux3.6内核中分组转发是如何完成的。利用最长前缀匹配算法选择的路由表表项来转发分组。所选择的表项还包含分组转发的下一跳信息。分组转发的第一步就是查找路由表找到最长前缀匹配的表项。查找路由表非常费时，尤其是进行最长前缀匹配时更是如此。因此，已经提出的好的数据结构可以加快路由表的搜索。例如，使用trie，或基于前缀长度的二分法检索（binary search），又称为折半检索。另一方面，因为同一目的地可能被频繁地访问，所以在路由缓存中存储有次访问过的搜索结果，然后再搜索路由缓存因为接下来的访问就可以节省很多路由表查找的时间。因此，在Linux3.6实现中，路由缓存用来加速目的地址的查找过程。在路由缓存的帮助下，路由表的全面搜索只有在缓存中没有找到的情况下才会发生。

框图

Linux2.6的IPv4分组转发过程的调用如图4-12所示。对于来自上层的分组，如果不知道路由路径，那么决定输出设备（接口）的主函数是_ip_route_output_key()（在src/net/ipv4/route.c中）。ip_route_output_key()使用散列函数rt_hash()试图在路由缓存中找到路由路径（输出设备），这最终会调用在include/linux/jhash.h中的Bob Jenkins的散列函数jhash()（参见http://burtleburtle.net/bob/hash/）。如果路由路径不在路由缓存中，那么就调用ip_route_output_slow()，通过调用fib_lookup()在路由表中查找目的地。

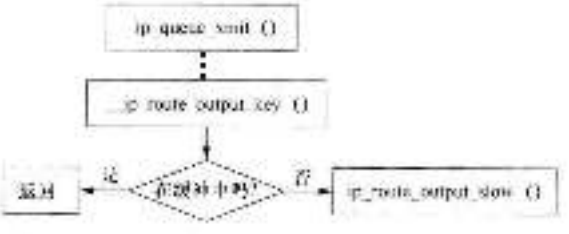


图4-12 IP转发实现：_ip_route_output_key

算法实现

一旦从网络接口收到一个分组，首先将分组复制到内核的sk_buff中。通常skb->dst为

NULL，即该分组不存在或被缓存路径，并且 ip_rcv_finish()调用 ip_route_input()以确定如何转发分组。像前面的情况一样，ip_route_input()首先试图在路由缓存中查找路由路径。如果没有找到，就调用 ip_route_input_slow()，接下来调用 fib_lookup()查找路由表。

数据结构

路由缓存利用 rt_hash_table 数据结构来维护，这是一个 rt_hash_bucket 数组。每个 rt_hash_table 表项指向一个 rtable 的列表，如图4-13所示。rt_hash()对从分组中提取的三个参数（源地址、目的地址和服务类型）进行散列处理，当通过 rt_hash()获得散列表项时，对 rtable 的列表进行线性搜索以便找到入口点。

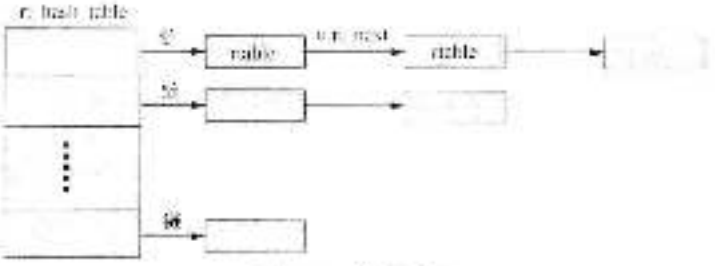


图4-13 路由缓存

如果在路由缓存中找不到目的地址，就搜索转发信息数据库（FIB）。FIB 数据结构相当复杂，如图4-14所示。Linux2.6 内核允许有多个 IP 路由表，每个都有一个独立的 fib_table 数据结构来描述。该数据结构的最后字段 tb_data，指向一个 fn_hash 数据结构，它是由一个散列表 fn_zones 和一个散列链表 fn_zone_list 组成。fn_zones 是33个 fn_zone 的数组，这里 fn_zones[z]指向一个前缀长度为 z 的散列表，0<=z<=32。然后 fn_zones 所有非空的项目由 fn_zone_list 链接起来，头部是具有最长前缀的表项。fib_lookup()调用起来的 tb_lookup()函数搜索路由表。默认 tb_lookup()函数为 fn_hash_lookup()（在 src/net/ipv4/fib_hash.c 中），通过遍历 fn_zone_list 查找每个散列表的前缀长度。这个顺序搜索在找到一个匹配时就会结束。通过搜索 fn_zone_list 的头部，就能保证最长前缀匹配，即首次匹配就是最长的匹配。

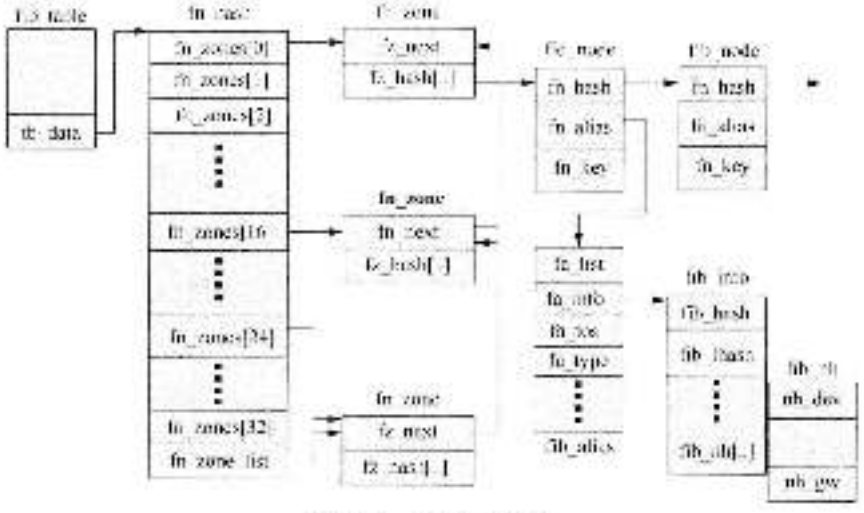


图4-14 FIB 数据结构

在图4-14的中间，每个 fn_zones 表项指向一个 fn_zone 数据结构。fn_zone 包含一个指向 fn_zone_list 的指针和一个散列表 fz_hash，这是一个指向 fib_node 的指针数组。一个 fib_node 则应于唯一一个子网。散列键 fn_key 是子网的前缀，例如，如果子网是 200.1.1.0/24，那么 fn_key 就是 200.1.1。散列函数 fn_hash()定义为一个 src/net/ipv4/fib_hash.c 中的内联函数。若每个

fib_node 中的 fa_alias 表项指向一个 fib_alias 结构，该结构包含子网的某些基本信息，如 fa_tos、fa_type、fa_scope 和一个指向 fib_info 数据结构的指针。最后，fib_info 包含详细的路由表项信息，包括输出设备和下一跳路由器。

对于任何非常前缀长度的发以散列表大小（fz_hash 表项数量）是 16。如果存储在散列表中的节点数量超过表大小的两倍，那么在第一次出现时表的大小就增加到 256，在第二次出现时则增加到 1024，在这之后无论何时条件满足时表的大小都会加倍。

练习

1. 利用一个例子来跟踪查找 ip_route_output_key()，然后写出如何搜寻路由缓存。
2. 追踪 fib_lookup()探讨如何查找 FIB。

性能问题：路由缓存和表中的查找时间

对于分组流中第一个到达的分组，路由机制很可能会导致两次路由查找操作，一次发生在路由缓存中，这次可能导致一次查找失败（未命中），然后在 FIB 路由表的另一次查找中产生了一次命中。对于分组流中随后到来的分组，路由机制能够在路由缓存中找到第一个分组到达的查找结果，这就意味着人们只在缓存中查找。令人们感兴趣的是，我们能够多快地在这两种数据结构中执行路由查找。我们需要测量用于执行 ip_route_output_key()和 ip_route_output_slow()的时间。在轻载 Linux 路由器上处理 64 字节长度的分组，运行 ip_route_output_key()和 ip_route_output_slow()后，测量分别得到 25μs 和 0.6μs，这说明两种相差倍数为 42。尽管两者都是散列表，但 FIB 表是一个散列表数组，它需要从具有最长前缀的表中连续地搜索。

分组格式

接下来，我们学习 IP 分组格式。一个 IP 分组由一个头部字段，紧随其后的是数据字段，它的长度必须是 4 字节字的整数倍。IP 头部的格式如图 4-15 所示。每个字段的含义描述如下：



图 4-15 IPv4 分组格式

版本号：版本号指定 IP 协议的版本。IP 协议当前版本是 4，下一代 IP 版本为 6。

头部长度：IPv4 头部具有可变的长度。该字段以 4 字节为单位指定头部的长度。没有选项字段，典型的头部长度是 5 个字，也就是说，20 字节。

- 服务类型（TOS）：TOS 指定 IP 分组需要的服务。最理想情况下，路由器会根据分组的 TOS 处理分组。然而，不是所有的路由器都具有这种处理能力。根据 RFC 791 和 1349（见图 4-16），TOS 中的前 3 位用来定义分组的优先级，随后的 4 位定义处理该分组时的性能度量。性能度量是延迟、吞吐量、可靠性和成本。最近，RFC 2474 将前 6 位定义为区分服务（DS）字段，由它携带分组的 DS 代码点。



图 4-16 TOS 定义

- 分组长度：这个字段指定总的 IP 分组的长度，包括头部和数据，以字节为单位。因为它有 16 位，所以 IP 分组的最大长度为 65 536 字节。这称为最大传输单元（MTU）。
- 标识号：该标识唯一地标识一个 IP 分组。它又称为序列号，在 IP 分段中特别有用。我们稍后将详细讨论 IP 分段。
- 标志：标志字段中的最低两位用于分段控制。第一个控制位称为不分段位。如果设置了该位，IP 分组就不能分段。最后一位称为更多分段位。如果该位设置为 1，就表示当前分组位于一个较大分组的中间。
- 分段偏移：如果当前分组是一个分段，那么该字段指示该分段在原分组中的位置。偏移的测量是以 8 字节为单位，因为该字段在其前面 3 位给标志字段后就只剩下 13 位了。
- 生存时间（TTL）：TTL 指定允许分组通过路由器的最大个数。它在新版本 IP 协议中称为跳限。每个路由器在转发分组给下一个路由器之前将 TTL 减 1。如果 TTL 为 0，那么就丢弃分组并产生一个错误信息，即一个 ICMP 消息，并将该消息发送到源。
- 上层协议：该字段指示将分组传给的上层协议。例如，设值为 1、6、17 分别表示上层协议为 ICMP、TCP、UDP。RFC 1700 定义用于该字段的可能编号。
- 头部校验和：校验和用于检测接收到的 IP 分组中的错误。这与 CRC 不同，这种 16 位校验和的计算和填充是按每个 IP 头部当为一个序列的 16 位字来处理的，使用 1 的补运算计算这些字段的总和，然后对结果进行补运算。我们在第 3 章中已经描述了一个类似的处理程序。显然这种 16 位校验和的保护不如 CRC-16 那样强。但是它可以更快地计算并且很容易地在软件中完成。在目的地，如果 IP 报头的所有 16 位字总和不产生一个 0，那么就检测到一个错误，通常会丢弃一个出错的分组。
- 源和目的 IP 地址：这两个字段指定源和目的 IP 地址。如上所述，目的地址是将分组转发到最终目的地的关键。
- 选项：并非在每个分组中都要求有选项字段。它具有可变的长度，具体取决于选项类型。简单、选项字段用于测试或调试。因此，它涉及路由器的操作。例如，源路由选项一种常用的用来指定路由路径的选项，即从源到最终的目的地的一系列路由器。选项字段很少使用，因此没有将它包含到 IP 报头位固定部分中。
- 数据：数据字段包含来自上层的协议数据单元（PDU），它将发送到目的地。

分组分段与重组

不仅 IP 协议具有自己的 MTU 限制，每个链路层协议也经常更严格地限制每次可以转发的最大帧。例如，以太网将 MTU 限制为 1518 字节，其中包括 18 字节的协议开销和 1500 字节的有效载荷（上层数据）。换句话说，当在以太网接口上发送 IP 分组时，IP 分组的最大长度为 1500 字节。然而，从上层来的分组可能大于以太网协议的极限 1500 字节，因为我们知道 IP 协议的 MTU 为 65 536 字节。通过分段可以将一个大的 IP 分组分成两个或两个以上较小的 IP 分组，而且必须足够小以使能够通过链路层，如图 4-17 所示。这些小的 IP 分组称为 IP 分段。由于 MTU 要根据底层所采用的链路层协议而变

化，所以分段既可以在源节点进行也可以在中间路由器上进行。重组是利用IP分段重建原IP分组的过程。在IP协议中，重组只在最终的目的地完成以避免延长路由器上的缓冲。

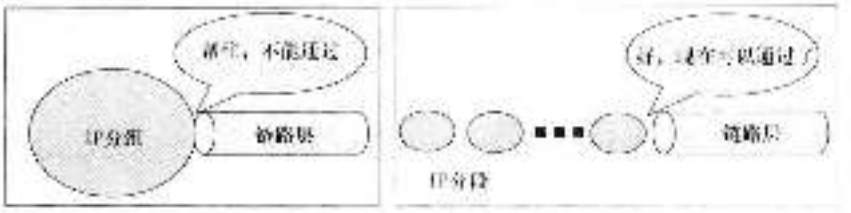


图4-17 IP分段

开源实现4.3：汇编语言实现的IPv4校验和

概述

IP头部校验和的计算通过将整个IP头部作为一个16位字序列进行处理，利用1的补运算将这些字加起来，然后再对结果进行补运算。

算法实现

IP头部校验和使用ip_fast_csum（在src/include/csum_i386/checksum.h中）函数计算。既然需要为每个IP分组计算校验和，那么就需要有一种快速算法。Linux内核通过使用依赖于机器的汇编语言编写该函数来优化校验和的计算。对于80x86机器，ip_fast_csum()函数是用32位而不是16位字进行相加运算的。用C语言编写的代码如下所示。

```
for (num=0;length>0;length--)
    sum += *buf();
In ip_fast_csum(), the code is translated to:
"1: add 16(%1), %0 ;\n" /* the sum is put in %0; summation
is in 32-bit */
"lea 4(%1), %1 ;\n" /* advance the buf pointer by 4 (in
bytes) */
"decl %2 ;\n" /* decrease the length by 2 (in 16 bits) */
"jne 1b ;\n" /* continue the loop until length==0 */
```

然后将结果复制到另一个寄存器中。这两个寄存器通过移位将16位放到最低位，然后再加起来。取结果的补运算得出校验和。

练习

编写一个程序计算IP校验和并验证程序的正确性，通过与Wireshark软件捕获的真实IP分组进行比较。

如何重新组装IP分组会影响分段步骤的设计，因此让我们首先考虑IP的重组处理过程。为了重组一个IP分组，我们需要收集同一分组的所有分段。因此，我们需要有一个标识符将这些分段与其他分组的分段区分开来，并且我们需要知道是否收集到了所有的分段。为了做到这一点，IP分段处理为所有来自同一分组的分段在头部的标识符字段（或序列号字段）设置了相同的数。它在标志字段使用更多分段（more fragments）标志，标明该分段是否为最后的分段。假设给出一个IP分组的所有分段，那么重组器需要确定每个分段在原分组中的位置。这是由IP报头中的分段偏移字段来确定的。因此，每个分段实际上就是一个携带头部分段信息和原分组中部分数据的普通IP分组。与普通IP分组一样，一个IP分段还可以在中间路由器上进一步分段。目的地使用头部中的标识符、标志和分段偏移字段来重组原分组。

图4-18显示了一个将3200字节的分组分成三个分段以通过以太网接口的例子。（以太网的MTU为1518字节，带有18字节的头部和尾部）。注意分段偏移是以8字节为单位的，因为它仅使用13位而不是16位来记录分段在原来的IP分组中的偏移位置。因此，每个分段的分组长度，除了最后一个分段外，必须是8字节的整数倍。如图4-18所示，除了20字节的IP报头，可以放入分段的最大字节数是1500－20，即1480。每个分段的头部与原分组相同，除了下面两个字段外：标志和分段偏移。标志的更多分段位必须设置为1，除了最后一个分段外。目的地可以通过标识符字段来区分是否属于同一分组，通过更多分段位来识别是否为最后一个分段，使用分段偏移将分段重组到原来的适当位置。



a）源分组　　b）分段

图4.18　IP分段的一个例子

开源实现4.4：IPv4分段

概述

当发送一个大小大于链路层MTU的分组时，就必须进行分段。因此，在传输一个IP分组之前就必须进行大小检查。同一个IP分组的所有分段都具有相同的标识符。此外，需要为所有分段设置更多分段标志，最后一个分段除外。还需要正确地设置偏移字段，字段偏移是以8字节为单位的，并且除了最后一个分段外的所有分段都应该为8字节的整数倍。为了能够将多个分段重组IP分组，重组函数需要根据来自这些分段头部中的标识符、更多分段标志和偏移字段信息来处理。此外，重组分段的实现应精心设计，以避免缓冲区溢出攻击。

数据结构

IP头部的数据结构是iphdr，定义在src/include/linux/ip.h中。

```
struct iphdr {
    #if defined(LITTLE_ENDIAN_BITFIELD)
        u8 ihl:4;
          version:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
        u8 version:4;
          ihl:4;
#else
#error "Please fix <asm/byteorder.h>"
#endif
        u8 tos;
        __be16 tot_len;
        __be16 id;
        __be16 frag_off;
        u8 ttl;
        __u8 protocol;
        __sum16 check;
        __be32 saddr;
        __be32 daddr;
        /*The options start here. */
};
```

算法实现

接下来，我们将重点研究分段和重组函数。当将IP分组发送到网络接口时，就可以分段。上层协议调用ip_queue_xmit()将上层数据通过IP层发送。在ip_queue_xmit()中确定路由之后，将调用ip_queue_xmit2()检查分组长度是否大于下一条链路的MTU。如果大于，就调用ip_fragment()

执行分段。在 ip_fragment 中的一个 while 循环负责将原分组进行分段。除了最后一个分段外，将分段大小设置为小于 MTU 的 8 字节的最大倍数。每一个分段在正确地设置了头部和数据后按顺序发送到网络接口。（这些函数位于 src/net/ipv4/ip_output.c。）

图 4-19 显示了重组步骤的调用图（大多数函数位于 src/net/ipv4/ip_fragment.c）。当从数据链路层接收到一个 IP 分组时，就调用 ip_rcv() 函数来处理该分组。它调用 ip_route_input() 确定是转发分组还是把它传送到上层。在后一种情况下，调用 ip_local_deliver()，如果头部中的更多分段位或分段偏移不为零，则调用 ip_defrag()。IP 分段在一个称为 ipq_hash 的数列表中维护，这是一个 ipq 结构的数组。调用数列函数 ipqhashfn() 将 IP 分段散列到 ipq_hash 散列表中，基于四个字段：标识符、源 IP 地址、目的 IP 地址和上层协议的标识符（id）。ip_defrag() 函数首先调用 ip_find()，然后调用 ipqhashfn() 查找存储同一分组的分段的 ipq 结构队列。如果没有找到这样的队列，就调用 ipq_frag_create() 创建一个队列，它然后再调用 ipq_frag_intern() 将队列放入散列表中。ip_defrag() 函数然后调用 ip_frag_queue() 将分段放入到队列中。如果接收到了所有分段，就调用 ip_frag_reasm() 重组分组。

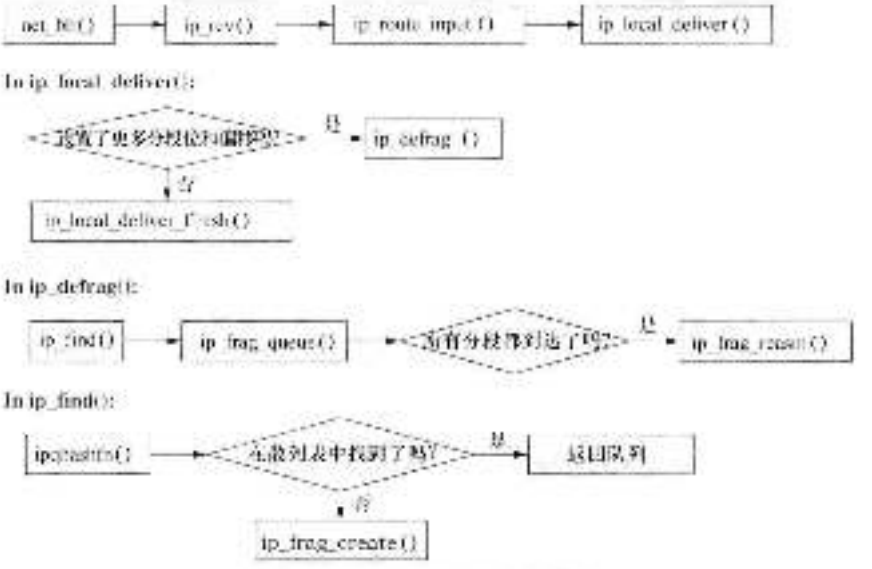


图 4-19 Linux 中的 IP 分段和重组

练习

使用 Wireshark 捕获一些 IP 分段并观察头部中的标识符、更多分段标志和偏移字段。

4.2.2 网络地址翻译

出于隐私和安全的原因，有些 IP 地址保留仅用于私人专用，企业内部的通信。这些地址，称为私有 IP 地址，定义在 RFC 1918 中。三块 IP 地址空间保留用于私有互联网，分别是：

10.0.0.0 — 10.255.255.255（10.0.0.0/8）

172.16.0.0 — 172.31.255.255（172.16.0.0/12）

192.168.0.0 — 192.168.255.255（192.168.0.0/16）

第一块就是一个 A 类网络号，第二块是一组 16 个连续的 B 类网络号，第三块是一组 256 个连续的 C 类网络号。

除了隐私和安全的考虑外，还有其他一些使用私有地址的原因。例如，当外部网络拓扑更改时（如更改 ISP）避免更改 IP 地址。最近，一个常见的原因就是由于 IP 地址耗尽问题。尽管我们可以通过采用下一代互联网协议自然地解决问题，但 IP 地址和网络地址翻译（NAT）可作为短期的应急解决方案。

基本 NAT 和 NAPT

NAT 是一种用于将一组 IP 地址映射为另一组 IP 地址的方法。更通常地，NAT 以一种透明的方式为终端用户提供公共互联网与私有互联网之间的连接。存在有两种 NAT 变种：基本 NAT 和网络地址端口翻译（NAPT）。为了让使用私有 IP 地址的主机能够访问公共互联网，基本 NAT 为私有网络中的每台主机分配一个全球唯一的公共 IP 地址，无论是动态的还是静态的。来自私有网络分组的源地址被其源主机分配的全球 IP 地址所代替。这也适用于进入分组的目的地地址是发向属于私有互联网的内部主机。

基本 NAT 要求访问公共互联网的每一台内部主机都要有一个公共 IP 地址。然而，对于小公司（小办公室，家庭办公室（SOHO）），许多内部计算机需要共享少量的 IP 地址。因此，另一种替换的方法，NAPT，将翻译机层到包括 IP 地址和传输层的标识符。通过 NAPT，两台共享同一个全局 IP 地址的内部网络主机将过传输层的标识符（如 TCP/UDP 端口号或 ICMP 消息标识符）进行区分。图 4-20 显示了最基本的 NAT 和 NAPT 翻译。为 IP 地址和传输层标识符的翻译，创建和维护一张 NAT 翻译表。对于基本 NAT，翻译表中的每一个表项都包含一对地址：（私有地址，全球或公共 IP 地址）。例如，在图 4-20 的 NAT 表中，私有地址 10.2.2.2 映射为 140.123.101.30。因此，所有源 IP 地址为 10.2.2.2 的分组将被截获并且它们的源 IP 地址由 NAT 服务器转换为 140.123.101.30。另一方面，NAPT 表中的每一表项包含 IP 地址和传输层标识符：（私有地址，私有传输层标识符，全球 IP 地址，全球传输层标识符）。例如，在图 4-20 的 NAPT 表中，所有带有 10.2.2.3 作为源 IP 地址和端口号为 1175 的分组将被截获，它们的源 IP 地址和端口号将由 NAT 的服务器相应地转换为 140.123.101.31 和 6175。



图 4-20 基本 NAT 和 NAPT 的例子

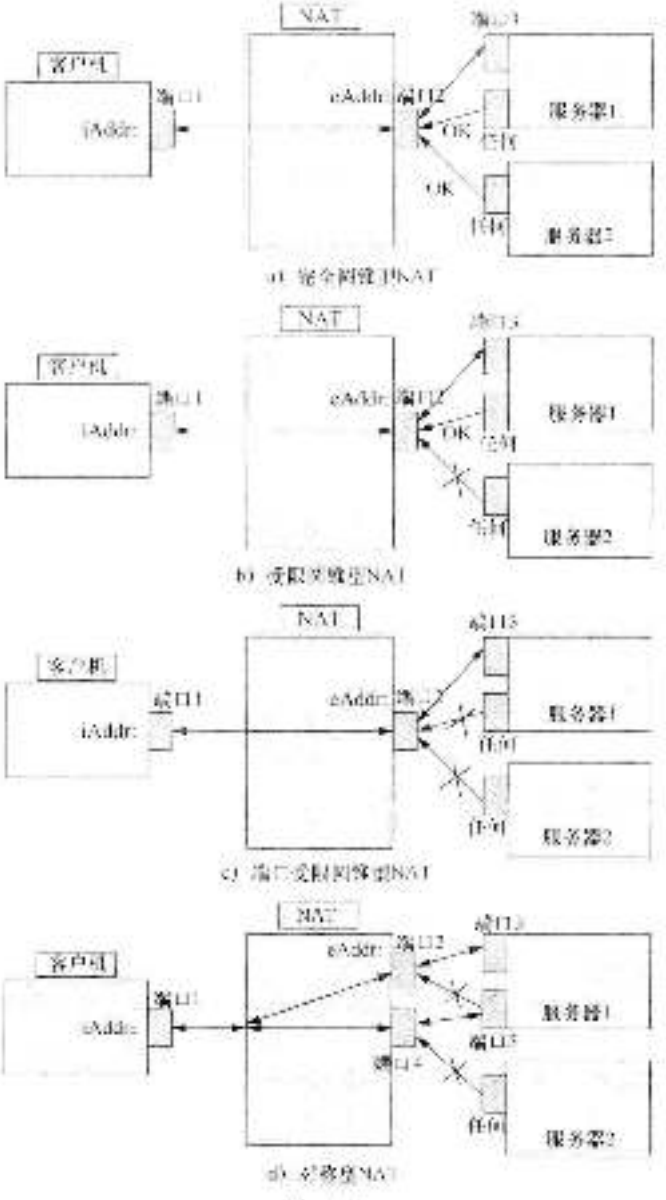
静态或动态映射

NAT 翻译表既可静态也可以动态地配置和更新。一个有足够的全球 IP 地址并且因为隐私和安全原因而使用 NAT 的组织，可以手动设置全局 IP 地址和私有 IP 地址之间一对一的映射关系。在这种情况下，每一个内部主机拥有一个唯一的对用户透明的全球 IP 地址，不仅能让内部网络主机访问公共互联网，而且也能以相反的方向访问。然而，在大多数情况下，NAT 表是按需更新的。NAT 维护一个全球 IP 地址池。当一个外部分组到达时，NAT 查表得到分组的源地址。如果找到一个表项，NAT 将私有地址翻译成相应的全球 IP 地址。（在 NAPT 中，也翻译传输层标识符。）否则，就从 IP 地址池中选择一个未分配的表项并将它分配给拥有源地址的内部主机（同样，在 NAPT 的情况下，选择一个新的传输层标识符）。每个表项关联一个定时器，以便释放不使用的表项。

尽管在大多数情况下，NAT 是用来单向访问公共互联网，但当有进入分组到达时创造一个新的 NAT 映射还是可能的。例如，当 NAT 接收到一个内部主机的域名查找，但是在 NAT 表中还没有对应的表项时，就可以创建一个新的表项和新分配的 IP 地址来用于回答域名的查找。在更为复杂的情况下，称为两次 NAT，也还可能的，这里通信的两台终端主机都是专用网络的内部主机（参见 RFC 2663）。

行动原则：不同类型的 NAT

具体取决于外部主机是如何通过预测公共地址和端口来发送分组的。NAT的实现可以分为四种类型：完全圆锥型（full cone，即一对一）、受限圆锥型（restricted cone）、端口受限圆锥型（port restricted cone）、对称型（symmetric）。其中，完全圆锥型是市场上最常见的实现，而对称型提供最好的安全性。完全因为在某种意义上，它是最难以遍历的。这些实现的操作细节将在图4-21中描述，在此仅做简要的描述。



a) 完全圆锥型NAT

b) 受限圆锥型NAT

c) 端口受限圆锥型NAT

d) 对称型NAT

图 4-21

完全圆锥型：一旦一个内部地址（iAddr：iport）已经映射到一个外部地址（eAddr：eport）时，所有来自（iAddr：iport）的分组将通过（eAddr：eport）发送，任何外部主机可以通过（eAddr：eport）向（iAddr：iport）发送分组。也就是说，NAT服务器将不检验进入分组的源IP地址和端口号。

受限圆锥型：和上面一样，除了仅有已经从（iAddr：iport）接收分组的外部主机能够通过（eAddr：eport）向（iAddr：iport）发送分组外。也就是说，服务器将记住外出分组的目的IP地址，并将进

入分组的源 IP 地址与记住的目的 IP 地址进行对比。

端口受限圆锥型：和上面一样，除了外部主机必须使用相同的已从（iAddr：iport）接收分组的端口通过（eAddr：eport）向（iAddr：iport）发送分组外。也就是说，NAT 服务器将检查进入分组的源 IP 地址和端口号。

对称型：对进入分组使用与端口受限圆锥型相同的操作。然而，对于每一个外部源 IP 地址和端口（iAddr：iport）→（eAddr：eport）的映射应该是唯一的。也就是说，如果外出分组有不同的目的 IP 地址或端口号，那么从相同的（iAddr：iport）来的外出分组将被映射到不同的（eAddr：eport）。

当两台主机通信时，如果发起者，经常是一台 NAT 之后的客户机，而响应者是一台不在 NAT 之后的服务器，那么将调用上述地址解析过程之一。另一个可选择的方法需要基本 NAT 或在 NAT 服务器上的端口重定向配置。如果两者都在 NAT 服务器之后怎么办？在 RFC 3489 中提出 UDP 简单穿越 NAT（Simple Traversal of UDP through NATs，STUN）以便在两台都在 NAT 服务器之后的主机提供 UDP 通信。基本思想是通过发送请求到 STUN 服务器来穿越 NAT 服务器。后来，UDP 和 TCP 简单穿越 NAT（STUNT）扩展了 STUN 以便包括 TCP 的功能。

端口重定向和透明代理

除了提供互联网接入外，NAT 还可以用于更安全或更有效的应用中，如端口重定向和透明代理。例如，网络管理员可能想将所有的 WWW 请求重定向到一个特定的 IP 地址和专用端口号。管理员就可以在域名服务器（DNS）数据库中创造一条新的记录，如一条将 www.cs.ecu.edu.tw 映射到 140.123.101.38 的记录。那么，在 NAT 表中创建了一条表项将映射重定向到想要的私有地址和端口号——例如，将 140.123.101.38：80 映射为 10.2.2.2：8080。这里“：80”和“：8080”代表端口号（这些将在第 5 章中正式介绍）。因此，在公共互联网络上的主机仅知道 www 服务器是 www.cs.ecu.edu.tw，其 IP 地址为 140.123.101.38。它的私有地址没有暴露，实际服务器就可以更加安全地对付入侵攻击。此外，也更容易将 WWW 服务器更换成具有不同私有 IP 地址的另一台机器。上述过程称为端口重定向。使用 NAT 的另一个例子称为透明代理，是将所有外出 WWW 请求重定向到一个透明代理，这样代理缓存就可以有助于加速请求处理，或者代理服务器可以检查请求或响应。例如，在 NAT 表中可以创造一个表项将 WWW 服务（140.123.101.38：80）映射到一个内部 WWW 代理（10.1.1.1：3128）。向外的 WWW 请求首先被 NAT 翻译并重定向到内部 WWW 代理服务器。在有缓冲代理的情况下，内部代理就可能直接从其本地缓存中准备响应，或者将请求转发到真正的服务器上。

行动原则：NAT 中的复杂 ALG

因为由 NAT 和 NAPT 所做的翻译更改了 IP 头部和传输层头部中的地址，所以这些头部中的校验和在翻译过后需要重新计算。此外，IP 地址和传输标识符的翻译可能会影响某些应用的功能。特别是，任何在其协议消息中编码源或目的 IP 地址/端口的应用将受到影响。因此，NAT 通信和应用级网关（ALG）一起使用。让我们考虑需要为 ICMP 和 FTP 对 NAT 进行修改。

ICMP 是一个用于 TCP/UDP/IP 错误报告的协议。我们将在 4.5 节中详细说明 ICMP。一个 ICMP 的错误消息，如目的地不可达错误，将错误分组嵌入在 ICMP 分组的有效载荷中。所以，不仅 ICMP 分组的地址，而且原来错误分组的源或目的地址都需要 NAT 进行翻译。然而，这些地址的任何更改都需要重新计算 ICMP 头部的校验和以及嵌入的 IP 头部的校验和。对于 NAPT 翻译，嵌入的 IP 头部的 TCP/UDP 端口也需要进行修改。在 ICMP 回送请求/应答消息中，使用查询标识符来标识回送（echo）消息，这种查询标识符相当于传输层标识符，因此也需要翻译。因此，如果查询标识符被修改，那么 ICMP 头部校验和也需要重新计算。

文件传输协议（FTP）是一种很受欢迎的互联网应用，将在第 6 章中介绍。在 NAT 翻译的情况下，FTP 也需要一个 ALG 方能保持正常工作。问题来自 FTP PORT 命令和 PASV 响应，因为这两个命令包含一个用 ASCII 编码的 IP 地址/TCP 端口对。因此，FTP ALG 需要确保正确地翻译 IP 地址、PORT 和 PASV 命令中的端口号。因为翻译后，用 ASCII 编码的 IP 地址的长度和端口号长度可能改变，比如，

从 10.1.1.1；3128 中的 13 个 8 位组到 140.123.101.38；21 中的 17 个 8 位组，这会使问题变得更加复杂。因此，分组长度也可能会改变，这也许会造成未来会影响后续 TCP 分组的序列号。为了使这些变动 FTP 应用透明，FTP ALG 需要一张特殊的表来列出 TCP 序列和确认号，列出需要对连接上的所有后续分组进行。

开源实现 4.5：NAT

概述

在 Linux 内核版本 2.2 之前，NAT 实现称为 IP 伪装。从 Linux 内核版本 2.4 开始，NAT 实现就基于了 iptables，对分组过滤功能的实现。NAT 的实现可以分为两种类型：源 NAT，用于外出的分组；目的 NAT，用于处理从互联网关上层来的进入分组。源 NAT 更改源 IP 地址和传输层标识符，而目的地 NAT 改变目的地址和传输层标识符。在分组过滤后和分组传送到输出接口前，完成源 NAT。源 NAT 在 iptables 中的钩子名字，在 Linux 中称为 NF_INET_POST_ROUTING。在分组过滤应用到来自网络接口卡或上层协议的分组之前，完成目的地 NAT。对于前者，钩子称为 NF_INET_PRE_ROUTING，而对于后者，它称为 NF_INET_LOCAL_OUT。

数据结构

建立源 NAT 和目的 NAT 钩子的 IP 表的数据结构为（见/net/ipv4/netfilter/nt_nat_rule.c）：

```
static struct xt_target ipt_snat_reg __read_mostly = {
    .name       = "SNAT",
    .target     = ipt_snat_target,
    .targetsize = sizeof(struct nf_nat_multi_range_
    compat),
    .table      = "nat",
    .hooks      = 1 << NF_INET_POST_ROUTING,
    .checkentry = ipt_snat_checkentry,
    .family     = AF_INET,
};
static struct xt_target ipt_dnat_reg __read_mostly =
{
    .name       = "DNAT",
    .target     = ipt_dnat_target,
    .targetsize = sizeof(struct nf_nat_multi_range_
    compat),
    .table      = "nat",
    .hooks      = (1 << NF_INET_PRE_ROUTING) |
    (1 << NF_INET_LOCAL_OUT),
    .checkentry = ipt_dnat_checkentry,
    .family     = AF_INET,
};
```

NAT 钩子函数的数据结构，如 nf_nat_in、nf_nat_out、nf_nat_local_in、nf_nat_fn，表们将被后跟踪（参见/net/ipv4/netfilter/nf_nat_standalone.c）：

```
static struct nf_hook_ops nf_nat_ops[] __read_mostly
= {
    /* Before packet filtering, change destination */
{
        .hook       = nf_nat_in,
        .owner      = THIS_MODULE,
        .pf         = PF_INET,
        .hooknum    = NF_INET_PRE_ROUTING,
        .priority   = NF_IP_PRI_NAT_DST,
    },
/* After packet filtering, change source */
{
        .hook       = nf_nat_out,
        .owner      = THIS_MODULE,
        .pf         = PF_INET,
        .hooknum    = NF_INET_POST_ROUTING,
        .priority   = NF_IP_PRI_NAT_SRC,
    },
```

```
/* Before packet filtering, change destination */
{
        .hook           = nf_nat_local_in,
        .owner          = THIS_MODULE,
        .pf             = PF_INET,
        .hooknum        = NF_INET_LOCAL_OUT,
        .priority       = NF_IP_PRI_NAT_DST,
    },
/* After packet filtering, change source */
{
        .hook           = nf_nat_fn,
        .owner          = THIS_MODULE,
        .pf             = PF_INET,
        .hooknum        = NF_INET_LOCAL_IN,
        .priority       = NF_IP_PRI_NAT_SRC,
    },
};
```

最后，跟踪连接的数据结构为：

```
struct nf_conn {
...
    struct nf_conntrack_tuple_hash tuplehash[IP_CT_DIR_
    MAX];
...
    struct nf_conn *master;
    /* Storage reserved for other modules: */
    union nf_conntrack_proto proto;
    /* Extensions */
    struct nf_ct_ext *ext;
...
};
struct nf_conn_nat
{
    struct hlist_node bysource;
    struct nf_nat_seq seq[IP_CT_DIR_MAX];
    struct nf_conn *ct;
    union nf_conntrack_nat_help help;
#if defined(CONFIG_IP_NF_TARGET_MASQUERADE) || \
    defined(CONFIG_IP_NF_TARGET_MASQUERADE_MODULE)
    int masq_index;
#endif
};
```

图 4-22 表示了这些数据结构之间的关系。

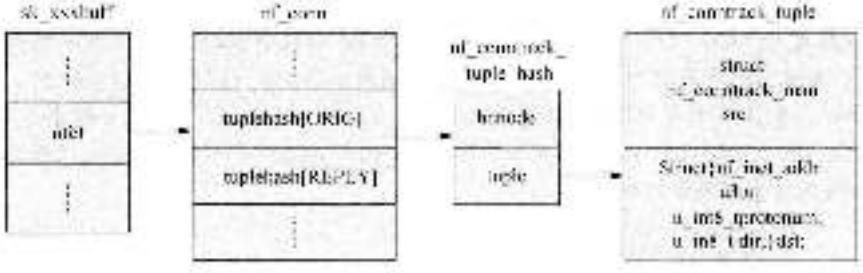


图 4-22 NAT实现的数据结构

算法实现

NAT模块通过调用 nf_nat_standalone_init()进行初始化，它调用 xt_nat_rule_init()注册 iptables 和 nf_register_hooks()以建立 NAT 钩子函数。初始化后，iptables 和钩子函数将会如图 4-23 所示进行设置。

如图 4-23 所示，为 NF_INET_PRE_ROUTING、NF_INET_LOCAL_OUT 和 NF_INET_ROUTING 的钩子

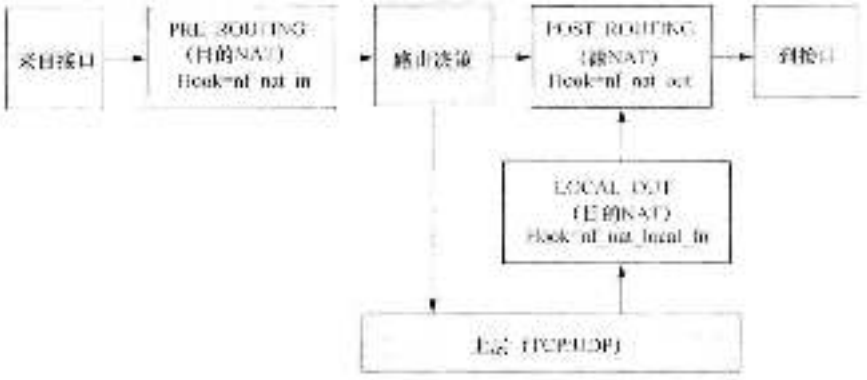


图 4-23 NAT 分组流

执行 NAT 的函数分别是 nf_nat_in()、nf_nat_local_in()和 nf_nat_out()。所有这 3 个函数最终都会调用 nf_nat_fn()实现 NAT 操作。

图 4-24 描绘了 nf_nat_fn()的调用图。nf_nat_fn()函数从 sk_buff 获得连接跟踪信息(nfct和 nfctinfo)。如果 nfctinfo 为 IP_CT_NEW 并且 NAT 没有初始化,那么将调用 alloc_null_binding()以为 LOCAL_IN 设置链。即还没有设置 NAT 规则;否则,调用 nf_nat_rule_find()。这两个函数都调用 nf_nat_setup_info()进行分组的网络地址翻译。在 nf_nat_setup_info()中,调用 get_unique_tuple()以获取得一种元组形式的结果。如果这是一个源 NAT (SANT),它就调用 find_appropriate_src()搜索 ipv4_nat_bysource 散列表。如果失败,则调用 find_best_ips_proto()为这次翻译获得一个新的元组。

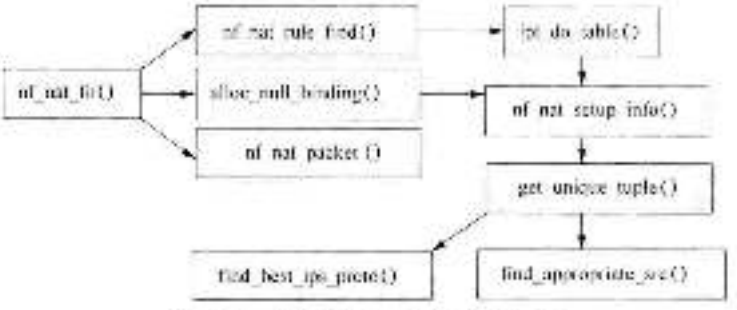


图 4-24 NAT 的 Linux 实现的调用图

如上所述,在完成了 IP 级别转换后,调用特定的 NAT 函数。ALG 函数又称为辅助函数。例如,FTP ALG 的辅助函数为 nf_nat_ftp()。图 4.25 中显示了 FTP ALG 在 linux 内核 2.6 中的调用图。遍历 mangle 数组,如果分组中含有 PORT 或 PASV 命令就调用 mangle_rfc959_packet();如果分组中含有 EPRT 命令(在 IPv6 中为 PORT)就调用 mangle_eprt_packet();如果分组中含有 EPSV 命令就调用 mangle_epsv_packet()。上述所有都调用 nf_nat_mangle_tcp_packet()来处理 TCP 所需要的更改,如序列号和校验和的重计算。

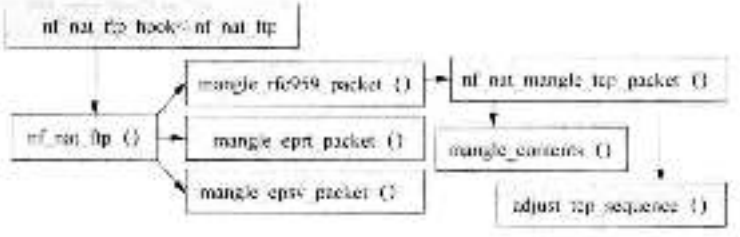


图 4-25 FTP ALG 的执行图

让我们以 ICMP 作为 NAPT 的一个例子。函数 icmp_manip_pkt() 用于更改 ICMP 消息的校验和及查询标识符。icmp_unique_tuple() 在指定范围内线性地搜索，就会在用户指定的范围内找到一个唯一的查询标识符。ICMP 和 IP 的校验和由函数 inet_proto_csum_replace4() 来重新计算，为了实际的校验和调整，它们依次调用 csum_partial()。为了更快地执行，函数 csum_partial() 是用汇编语言来实现的。

练习

跟踪 adjust_tcp_sequence() 并解释当分组出于地址翻译而改变时，如何调整 TCP 分组的序列号。

性能问题：NAT 实现及其他的 CPU 时间

尽管在 Linux 内核中 NAT 实现也进行数列运算，但它的执行时间比用于分组转发的查找函数的时间更长。导致这种情况出现有两个原因。在图 4-23 中，一个分组首先经过目的 NAT 然后经过源 NAT，每次调用都要查询数列表。另外一个原因是，除了 IP 地址和端口号翻译外，还需调用 ALG 辅助函数。图 4-26 中画出了携带有效载荷为 64 字节的 ICMP 分组在处理速度为 2.33GHz 的 CPU 分别对于转发（通过缓存和 FIB，分别标记成“路由缓存”和“路由 FIB”）、NAT、防火墙和 VPN（加密、认证分别标记为“3DES”和“HMAC-MD5”）处理的延迟。后两者将在第 8 章中介绍。尽管 NAT 消耗与防火墙同样数量的 CPU 时间，但大于转发所消耗的时间，但 NAT 延迟比验证和加密消耗的时间要少得多。显然对调用硬件加速的需要程度为：加密、认证、NAT、防火墙，然后是转发。当吞吐量小于 100Mbps 时，只有加密和认证必须需要硬件解决方案。在 Linux 内核下，软件实现对于 3DES 和 HAMC-MD5 分别取得的吞吐量为 75Mbps 和 85Mbps，但是在数十比吞吐量时，它们都需要借助于硬件加速器来完成。

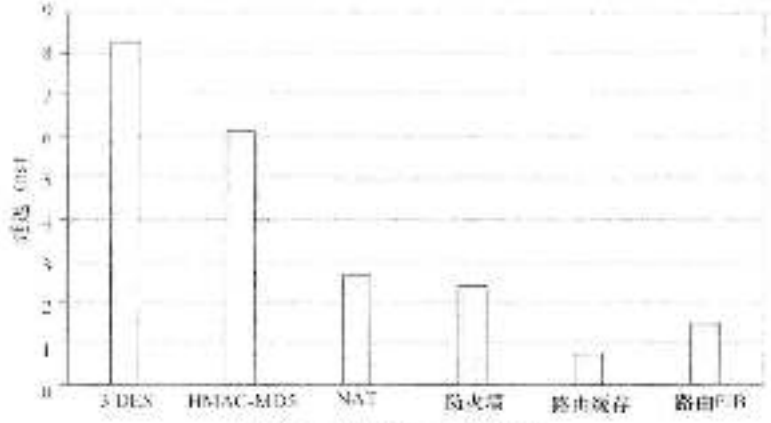


图 4-26 重要网络函数的延迟

4.3 互联网协议版本 6

当前 IP 协议的版本遇到了一些问题，其中最受关注的就是 32 位 IP 地址空间的不足。考虑到当前互联网发展趋势和 IP 地址无效率的利用情况，人们曾预测到 2011 年 IANA 未分配的 IPv4 地址将耗尽。早在 1991 年，IETF 呼吁为新的 IP 版本提建议，又称为下一代 IP（IPng）。后来收到多个建议，其中高中的建议称为简单互联网协议加强版（SIPP）。最初的地址长度建议为 64 位，后来被 IETF IPng 理事会加倍为 128 位。由于版本号 5 已经被试验性协议使用了，于是给新的 IP 协议分配的官方版本号为 6，即大家所熟知的 IPv6。对于 IPv4 地址耗尽问题的长久解决方案就是迁移到 IPv6 地址。

IPv6 考虑了某些新的特征。首先，将地址空间扩展到了 128 位。其次，为了加速在路由器上的分组转发，采用固定长度的头部格式。通过在头部包含一个流标签还考虑了对服务质量的支持。一种新的地址类型称为动播，建议用于向一组主机内任何主机发送分组（通常用于向同一个子网内的任意一

台可达路由器发送）。IPv6 也支持自动配置、与 DHCP 的功能相类似。最后，IPv6 使用扩展头部来支持分段、安全、增强型路由以及其他的功能特性。

历史演变：NAT 与 IPv6

NAT 与 IPv6 都试图解决 IPv4 地址空间短缺问题。显然，直到 2010 年，由于 NAT 与多数互联网兼容而被迫为解决方案。互联网的历史告诉我们，演变比革命更容易接受。从 IPv4 到 IPv6 的转换如同一场革命，需要对所有终端设备和互联网设备（如路由器）上的软件进行更新。另一方面，NAT 的使用就如同演变，它只需要在一些缺乏 IPv4 公共地址的子网部署 NAT 服务器。但是，随着未分配 IPv4 地址的逐步接近耗尽，似乎只有采用 IPv6 才能解决问题。如今，这仍然是一个难受争论的话题。

4.3.1 IPv6 头部格式

IPv6 的头部格式如图 4-27 所示。正如它最初的名字所指示的，IPv6 的设计原则简单。多个在大多数 IPv4 中没有使用的头部字段从 IPv6 中删除了，以便加速路由器上的分组处理和转发。这样做的结果就是，IPv6 头部具有固定长度，为 40 字节，没有选项字段。额外的功能使用扩展头部来实现，这些将在以后讨论。



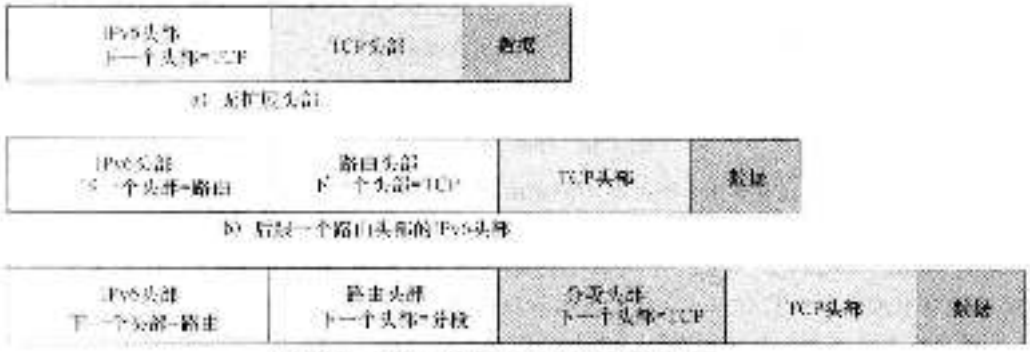
图 4-27 IPv6 头部格式

- 版本号：与 IPv4 一样，头部从一个版本字段开始，IPv6 的版本号为 6。
- 通信业务类别：通信业务类别字段指示分组需要的服务，与 IPv4 中的 TOS 字段相似。这个字段用于区分不同分组的服务类别。例如，前 6 位用于区分服务（DiffServ）框架中的 DC 代码点 [RFC 2472]。
- 流标签：流标签字段用于识别同一个流的分组以便提供不同的服务质量。例如，同一语音流的分组，当然要防止传输延迟和抖动，可当做一个流。但是并没有说明准确定义一个流的方式。因此，一个流可以是一个 TCP 连接或者一个源－目的地对，但在实际操作中，同一流的分组通常具有相同的源地址、目的地址、源端口号、目的端口号和相同的传输层协议。明显，根据这样的定义，同一个 TCP 连接中的分组将形成一个流。
- 有效载荷长度：16 位有效载荷长度字段表示除了 40 位头部长度外整个分组的字节长度。因此，最大有效载荷长度为 65 535 字节。
- 下一个头部：下一个头部字段表示上层协议或者下一个扩展头部。它用来取代 IPv4 中的协议字段和选项字段。如果这里没有特别的选项，则下一个头部标识运行在 IPv6 之上的上层协议，例如 TCP 或 UDP。如果需要特殊的选项（如分段、安全和增强路由），则 IPv6 头部后跟一个或多个扩展头部，其类型由下一个头部字段标识。
- 跳跃限制：跳跃限制字段在 IPv4 中称为生存时间（TTL），这里只是更改了名字，与 IPv4 中的 TTL 相同的方式使用。
- 源和目的地址：最后，头部由源和目的地址结束，每个地址占 128 位。在 IPv6 中有三种类型地址：单播、选播和组播，接下来将详细介绍。细心的读者也许发现了，在 IPv4 中存在的某些头部在 IPv6 中删除了。首先，不再有校验和。从头部中移除校验和有两个非常好的理由：首先，高层协议（如 TCP）中已经提供了可靠性保障，没有必要在中间路由器上重新计算校验和。其

次，不再有分段标志和偏移位，因为不允许在中间路由器上进行分段。这样其次减轻了路由器上的处理负担。分段和其他选项（如源路由）现在可以由扩展头部，一种比 IPv4 更高效灵活的机制来处理。由于在 IPv6 的头部中没有选项字段，所以 IPv6 的头部长度是固定的，并且固定长度的头部也能提高路由器的处理速度。

4.3.2 IPv6 扩展头部

IPv6 利用扩展头部来支持分段和其他选项。在 IPv6 头部中的下一个头部字段指示紧跟 IPv6 的扩展头部。每个扩展头部还有一个下一个头部字段，用来指示后面的扩展头部或高层协议头部。图 4-28 给出了使用 3 个扩展头部的例子。图 4-28a 是最常见的例子，IPv6 头部后跟着 TCP 头部。在该例子中，IPv6 头部的下一个头部字段的值为 6，它表示 TCP 的协议标识符。如果需要增强路由，则路由头部可以如图 4-28b 所示那样地使用。在这种情况下，IPv6 头部的下一个头部字段的值为 43，这里 43 指示紧跟 IPv6 头部的路由头部，路由扩展头部中的下一个头部字段包含值 6。同样，如果需要路由选项和分段，则扩展头部的序列号如图 4-28c 所示。路由头部的下一个头部字段的值为 44，表示下一个头部为分段头部。



a) 无扩展头部

b) 后跟一个路由头部的 IPv6 头部

c) 后跟一个路由头部和一个分段头部的 IPv6 头部

图 4-28 IPv6 扩展头部

在 RFC 2460 中推荐了多条处理扩展头部的规则。首先，扩展头部的顺序要按表 4-1 进行。如 RFC 2460 中所述，尽管 IPv6 节点必须接受并处理以任意顺序的扩展头部，但是我们强烈建议 IPv6 分组的源按照推荐的顺序进行。特别是，逐跳选项头部被沿着路径上的所有中间路由器处理时，它必须严格地紧接着 IPv6 头部之后出现。其次，扩展头部必须严格地按照它们在分组中出现的顺序处理，因为每个扩展头部的内容或语意决定着是否要处理下一个头部。再次，中间路由器（即非目的节点）除了增加逐跳转发的额外头部外，不能处理扩展头部。最后，每种扩展头部最多仅出现一次，除了目的地址选项头部最多可以出现两次外（一次在路由头部之前，一次在上层头部值之前）。

表 4-1 IPv6 扩展头部的顺序

基本 IPv6 头部
基本 IPv6 头部
逐跳选项头部（0）
目的地选项头部（60）
路由头部（43）
分段头部（44）
认证头部（51）
封装安全有效载荷头部（50）
目的地选项头部（60）
移动头部（135）
无下一个头部（59）
上层头部

4.3.3 IPv6 中的分段

IPv6 中的分段与 IPv4 中有些不同。首先，为了简化在路由器上的分组处理，在路由器上不允许分段。但就是说，分段仅在源端进行。其次，分段信息，例如“更多分段”位和分段偏移，并由一个称为分段头部的扩展头部而不是 IPv6 头部来携带。图 4-29 显示了分段头部格式，下一个头部字段指示下一个头部的类型。分段偏移和更多分段位（图中的 M 位）与 IPv4 中的使用方式相同。图 4-30 显示了将一个大的分组分成三个分段的过程。在前两个分段中，更多分段位设置为 1。分段偏移仍然用 8 个 8 位来表示。

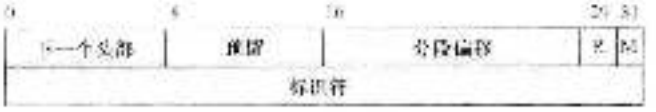
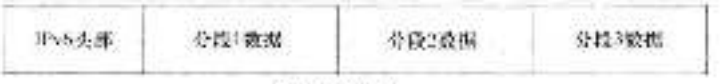


图4-29 分段头部



a) 原来的分组



b) 分段

图4-30 IPv6分段的例子

然而，还有一个问题没有解决：既然在中间路由器上不允许分段，那么源端如何知道整条链路上的MTU并相应地进行分段呢？这里有两种方法解决这个问题。首先，在IPv6的网络中，每条链路需要有1280字节或更大的MTU。因此，一个源总是可以假设每条路由路径的MTU为1280字节并将分组分段成1280字节或更短。其次，一个源可以运行路径MTU发现协议（RFC 1981）来发现路径上的MTU。强烈建议IPv6主机运行路径MTU发现协议以充分利用大于1280字节的路径优势。

4.3.4 IPv6地址的表示法

由于IPv6地址较长，曾经用于IPv4地址的点分十进制表示方法并不适用于表示IPv6地址。因此，采用冒号十六进制方法来表示IPv6的地址，格式为X:X:X:X:X:X:X:X，其中X为16位IPv6地址的十六进制表示。下面给出一个冒号十六进制表示法的例子：

3FFD:3600:0000:0000:0302:B3FF:FE3C:C0DB

冒号十六进制表示法仍然非常长，而且在绝大多数情况下含有很多连续的零。建议将零进行压缩，用一对冒号代替连续的零。例如，前面的例子就可以简写成：

3FFD:3600::0302:B3FF:FE3C:C0DB

4.3.5 IPv6地址空间分配

与IPv4不同的是，IPv6地址没有类。在IPv6中，一个前缀用来表示IPv6地址的不同用法。对IPv6中前缀用法的最新定义在RFC 4291:IPv6地址体系结构中。表4-2显示了最新IPv6前缀分配以及分配给给定前缀的IPv6地址空间，这是一种前缀占用空间与整个IPv6地址空间之比。正如我们从表4-2中所看到的，大部分地址当前还没有分配，只有15%的IPv6地址空间分配了。

IPv6的地址有三种：单播、组播和选播。有些值得注意的地址包括，与IPv4兼容的地址（前缀为00000000）、全球单播地址和链路本地单播地址。组播地址是以11111111前缀开始的。最后，选播地址具有一个子网首缀后跟全是0，这与IPv4子网地址的格式相似。一组节点（路由器）可以分享一个选播地址。发给选播地址的分组应该发送到组中的一个成员，一般是最近的一个。

以前缀00000000开始的地址保留用于与IPv4兼容。有两种方法将IPv4地址编码为IPv6地址。一台运行IPv6软件的计算机可以分配一个以96个0开始后跟IPv4地址的IPv6地址，称为IPv4兼容的IPv6地址。例如，140.123.101.160的IPv4兼容IPv6地址为0000:0000:0000:0000:0000:0000:8C7B:65A0，

表 4-2 IPv6 地址的前缀分配

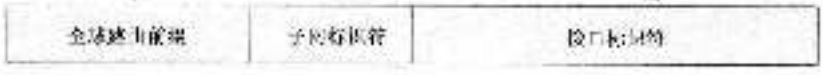
前缀	地址类型	部分	前缀	地址类型	部分
0000::/8	预留	1/256	A000::/3	未分配	1/8
0100::/8	未分配	1/256	C000::/3	未分配	1/8
0200::/7	未分配	1/128	E000::/4	未分配	1/16
0400::/6	未分配	1/64	F000::/5	未分配	1/32
0800::/5	未分配	1/32	F800::/6	未分配	1/64
1000::/4	未分配	1/16	FC00::/7	唯一本地单播	1/128
2000::/3	全球单播地址	1/8	FE00::/9	未分配	1/512
4000::/3	未分配	1/8	FE80::/10	链路本地单播地址	1/1024
6000::/3	未分配	1/8	FEC0::/10	未分配	1/1024
8000::/3	未分配	1/8	FF00::/8	组播地址	1/256

也可以写成::87CB:65A0。一个不能理解 IPv6 的常规 IPv4 的计算机将会分配一个以 80 个 0 开始后跟 16 个 1 再后跟 32 位 IPv4 地址的 IPv6 地址，称为 IPv4 映射的 IPv6 地址。例如，140.123.101.160 的 IPv6 不兼容地址或者 IPv4 映射的 IPv6 地址为::FFFF:8C7B:65A0。

两类特殊的地址也是以前缀 00000000 开始。全 0 的地址称为单播自指定地址，这个地址用于主机启动引导过程中。本地回路地址为::1，用于本地回不测试。

IPv6 允许在一个接口上配置多个 IPv6 地址。因此一个接口可能同时有多个全球单播地址和本地链路地址。本地链路地址并非全球唯一的，因此经常用在单条链路上用于自动配置地址和邻居发现。本地链路地址的前缀为 11111010，后跟 56 个 0 和 64 位的接口标识符。接口标识符可以从硬件地址（如 EUI-64 格式）编码得到。

IPv6 全球单播地址的一般格式如图 4-31 所示。为了支持路由和地址聚合，全球路由前缀一般都是层次化的结构。此外，所有全球单播地址除了以二进制 000 开始的以外都具有 64 位接口标识符的字段。正如表 4-2 所示，目前可分配的全球单播地址具有前缀 2000::/3。直到 2009 年 11 月，IANA 已经给 RIR（IHPE NCC、APNIC、ARIN、LACNIC 和 AfrNIC）分配了 36 个前缀。大部分最新的 IPv6 单播地址可以在 http://www.iana.org/assignments/ipv6-unicast-address-assignments 上找到。

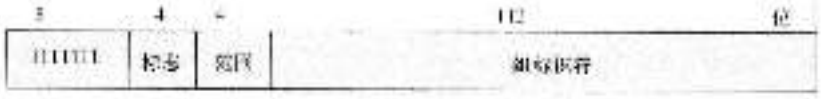


n 位	m 位	128-n-m 位
全球路由前缀	子网标识符	接口标识符

图 4-31 IPv6 全局单播地址格式

IPv6 组播地址以前缀 11111111 开始，如图 4-32 所示。与依靠 TTL 值来控制组播范围的 IPv4 不同，IPv6 组播地址包含一个范围字段来指示组播范围，支持 5 种组播范围：本地节点、本地链路、本地站点、本地组织，以及全球。它还有一个带有 T 位的标志来表示这个组播地址是临时地址（T=1）还是众所周知的地址（提供永久组播服务）。

8	4	4	112 位
11111111	标志	范围	组标识符



标志：0RPT
T=0：众所周知的组播地址
T=1：暂时的组播地址
P：基于前缀或不基于前缀的地址分配（RFC 3306）
R：嵌入式组播点地址或不嵌入（RFC 3956）

范围：组播组的范围
0000：预留
0001：本地节点范围
0010：本地链路范围
0101：本地站点范围
1000：本地组织范围
1110：全球范围

图 4-32 IPv6 组播地址的格式

有些组播地址保留用于特殊目的，例如，FF02:0:0:0:0:0:0:2 用于到达在同一个物理网络上的所有路由器。表 4-3 列出了一些预留的组播地址。

表 4-3 预留的 IPv6 组播地址

范 围	保留地址	目 的
本地节点	FF01:0:0:0:0:0:0:1	所有节点地址
	FF01:0:0:0:0:0:0:2	所有路由器地址
本地链路	FF02:0:0:0:0:0:0:1	所有节点地址
	FF02:0:0:0:0:0:0:2	所有路由器地址
	FF02:0:0:0:0:1:FFxx:xxxx	请求节点地址
本地站点	FF05:0:0:0:0:0:0:2	所有路由器地址
	FF05:0:0:0:0:0:0:3	所有 DHCP 服务器地址

4.3.6 自动配置

IPv6 有一个非常特别的功能就是支持自动配置。DHCP 在每个网络中需要一个 DHCP 服务器或者一个中继代理。与 DHCP 不同的是，IPv6 支持无状态服务器的自动配置。一台主机首先产生一个唯一的本地链路地址，包含了本地链路接口低位的 64 位接口标识符，可以从其唯一的硬件地址，编码如上所述。然后主机利用这个地址发送一个路由器请求信息（一个 ICMP 报文），一个路由器收到这个请求信息，就用一个包含子网前缀信息的路由器通告信息进行应答。然后主机就利用这个前缀产生自己的全球地址。

4.3.7 从 IPv4 到 IPv6 的迁移

我们在何时以及用何种方法将现有的互联网迁移到 IPv6 网络呢？问题在于作为新版本的 IP 意味着需要新版本的网络软件，而且也不可能在某一天让互联网中所有的主机都将自己的软件版本同时更新为支持 IPv6 的。在这种情况下，互联网如何操作才能让 IPv6 兼容的主机与 IPv4 兼容的主机共存呢？在 RFC 1933 中建议使用两种方式：双栈或隧道技术，另外还有一种方式，协议翻译，已建议用来解决 IPv6 地址迁移问题。

采用双栈的方式就是让一台主机（或路由器）同时运行 IPv6 和 IPv4 两种协议。考虑一个既包括 IPv4 兼容主机又包括 IPv6 兼容主机的子网。一台 IPv6 主机将同时运行 IPv6 和 IPv4，这样就能用 IPv4 分组与 IPv4 兼容的主机进行通信，采用 IPv6 分组与 IPv6 兼容的主机通信。另一个例子就是，让一个有纯 IPv6 网络的子网路由器同时运行 IPv4 和 IPv6 协议。来自子网的 IPv6 分组离开子网的时候就会被路由器转化为 IPv4 分组。另外一方面，路由器接收到的 IPv4 分组在转发之前转化成 IPv6 分组。注意在转化过程中或许会丢失一些信息，因为这两种协议的头部并不完全兼容。

另一种方法就是 IP 隧道技术，它是指将一种 IP 分组封装到另一种 IP 分组的有效载荷字段中的处理方法。既可以在发送端和接收端之间，也可以在路由器之间构建隧道。在第一种情况下，发送端和接收端都支持 IPv6，但它们之间的路由器不支持 IPv6。发送端就可以将 IPv6 分组其装到一个 IPv4 分组中并使用目的地作为接收端的地址。这个 IPv4 分组然后以普通 IPv4 分组在 IPv4 网络中转发，并最终到达接收端。知道接收端 IPv4 和 IPv6 地址的发送端可以使用 IPv4 兼容的 IPv6 地址。一条隧道已经可以在两台路由器之间建立。考虑到 IPv4 骨干网络连接的两个纯 IPv6 网络。来自子网的 IPv6 分组封装到一个 IPv4 分组中并使用目的地作为接收端的路由器。当这个 IPv4 分组到达接收端的路由器时，路由器识别这是一个封装过的分组，然后解封装，提取并转发嵌入的 IPv6 分组给接收端。对于隧道有很多建议，已经提出的建议包括配置隧道和自动隧道。在 RFC 4380 中提出的蓬通代理，有助于用户配置双向隧道。正如在 RFC 3056 中所述，一种特殊的地址前缀在助于经过 IPv4 云将 IPv6 域连接起来，它称为 6 到 4。对于 6 到 4 问题将 IPv4 NAT 之后的主机连接到 IPv6 主机的解决方案称为 Teredo，定义在 RFC

4380 中。另外一种自动隧道机制的目的是经过 IPv4 网络将 IPv6 主机和路由器连接起来，称为站点内自动隧道地址协议（Intra-site Automatic Tunneling Addressing Protocol，ISATAP），定义在 RFC 5214 中。

另外一种有效的方式就是协议翻译转换器，就是当纯 IPv4 主机与纯 IPv6 主机进行通信时，将一种协议翻译为另外一种协议。协议翻译转换需要在 IPv4 和 IPv6 网络之间有一台网关，或者在协议栈中有一个中间件将 IPv4 协议和地址翻译成 IPv6，反之亦然。这种解决方案包括 SIIT（RFC 2765）、NAT-PT（RFC 2766、4966）、BIS（RFC 3338）。这种翻译机制也需要 DNS 扩展以便支持 IPv6，在 RFC 3596 中定义。

4.4 控制平面协议：地址管理

在本节中，我们主要讨论两种 IP 地址管理机制。在 4.4.1 节中，我们将研究用来翻译互联网协议层（第 3 层）和链路层（第 2 层）地址的地址解析协议（ARP）。在 4.4.2 节中，我们将讨论用来动态和自动配置 IP 地址的动态主机分配协议（DHCP）。

4.4.1 地址解析协议

当一台主机想要将一个分组传送到目的地时，主机首先确定目的地是否和自己位于同一子网内。如果是，则分组直接通过链路层传送目的地；否则，分组也要通过链路层发送到路由器然后再进行转发。问题在于 IP 地址用于 IP 层，而硬件（MAC）地址（如 48 位以太网地址）用于链路层，主机是如何利用分组头部中的目的 IP 地址获得目的地或路由器的物理地址呢？因此，我们需要一种地址解析协议将 IP 地址翻译为物理地址。

通常，地址解析可以用两种方法实现：使用服务器或不使用服务器。如果有一台地址解析服务器，那么所有的主机都可以向服务器发送注册消息，以使服务器能够知道所有主机的 IP 地址与 MAC 地址之间的映射。当一台主机想要向同一子网内的主机（或路由器）发送分组时，它可以查询服务器。为了避免在每一台主机上手动配置地址解析服务器参数，主机可以广播注册消息。这种方法的缺点是，在每个 IP 子网内都需要一台地址解析服务器。互联网采用的地址解析协议（ARP），使用另一种无服务器的方法。当一台主机需要查询目的物理地址时，它就广播一个 ARP 请求消息。目的地收到请求后将回应一个 ARP 应答消息。由于 ARP 请求包含发送端的 IP 和 MAC 地址，所以目的地可以使用单播发送 ARP 应答。如果每次发送 IP 分组前都运行 ARP，那么就会太低效了。因此，每台主机维护一张（IP 地址，MAC 地址）缓存表，这样如果可以在缓存中找到映射就没有必要再运行 ARP 了。另一方面，ARP 采用软状态方法，它允许主机动态地改变它的 IP 地址或 MAC 地址（例如，更改网络接口卡）。也就是说，缓存表中的每一个表项都关联了一个定时器，定时器超时的表项会被自动地删除掉。由于 ARP 请求消息是一种广播消息，所以所有主机都可以接收到它并看到发送端的 IP 地址和 MAC 地址。作为“好的”一方面，发送端的缓存表项可以通过这种广播消息来更新。

在某些特殊的情况下，也许需要一种从 MAC 地址到 IP 地址之间的一种逆向映射，称为逆向 ARP 协议。例如，知道自己 MAC 地址的一个无盘工作站可能需要从服务器获得它的 IP 地址，在它能够使用 IP 地址访问网络文件系统（NFS）或者查询用于启动的操作系统镜像文件之前。接下来，我们将看到 ARP 也支持逆向 ARP 请求和应答操作。

ARP 分组格式

ARP 协议是一种用于网络层与数据链路层之间协议翻译的常用协议。ARP 分组的格式如图 4-33 所示。地址类型和地址长度字段允许 ARP 协议用于多种网络和链路层的协议。硬件地址类型和协议地址类型指示分别用于链路层和网络层的协议。最常见的硬件地址类型是以太网，值设置为 1；IP 协议类型值为 0x0800。地址类型字段后跟两个长度字段：硬件地址长度和协议地址长度。以太网和 IP 分别使用值 6 和 4。操作代码表示 ARP 消息的操作。共有四种操作代码：请求(1)、应答(2)、RARP 请求(3) 和 RARP 应答(4)。接下来的两个字段是发送者的链路层地址和 IP 地址。最后两个字段是接收者的链路层地址和 IP 地址。在一个 ARP 请求消息中，发送者将以 0 来填充目标硬件地址字段，因为它并不知道接收者的物理地址。



图 4.33 ARP 分组格式

由于 ARP 和 IP（以及其他的网络层协议）两者都在链路层帧的有效载荷中携带，所以在链路层头部中需要用于不同网络层协议分组复用或解复用的控制信息。例如，以太网具有一个 2 字节的类型字段指示上层协议。用于 IP 和 ARP 的协议标识符是不同的，分别为 0x0800 和 0x0806。在以太网中，广播一个 ARP 请求消息可以通过将目的地址置为 0xFFFFFFFFFFFF 来实现。

开源实现 4.6：ARP

概述

ARP 协议的实现需要一张 ARP 缓存表和发送、接收 ARP 分组的函数。大部分 ARP 的源代码可以在 src/net/ipv4/arp.c 中找到。

数据结构

最重要的数据结构是 arp_tbl，它保存 ARP 使用的最重要的参数。将 arp_tbl 定义为 struct neigh_table，由一个 hash_buckets 表项组成，存储邻居信息的 ARP 缓存。下面显示了用于 neigh_table 的数据结构：

```
struct neigh_table
{
    struct neighbour        *next;
    int                     family;
    int                     entry_size;
    int                     key_len;
    __u32   (*hash)(const void *pkey, const struct net_
dev *dev *);
    int                     (*constructor)(struct
neighbour *);
    int                     (*pconstructor)(struct
pneigh_entry *);
    void                    (*pdestructor)(struct
pneigh_entry *);
    void                      (*proxy_redo)(struct
sk_buff *skb);
    char                    *id;
    struct neigh_parms      parms;
    int                     gc_interval;
    int                     gc_thresh1;
    int                     gc_thresh2;
    int                     gc_thresh3;
    unsigned long             last_flush;
    struct timer_list         gc_timer;
    struct timer_list         proxy_timer;
    struct sk_buff_head       proxy_queue;
    atomic_t                entries;
    rwlock_t                lock;
    unsigned long             last_rand;
    struct kmem_cache           *kmem_cachep;
    struct neigh_statistics *stats;
    struct neighbour      **hash_buckets;
    unsigned int          hash_mask;
```

```
    __u32                   hash_rnd;
    unsigned int            hash_chain_gc;
    struct pneigh_entry     **phash_buckets;
};
```

框图

发送和接收 ARP 分组分别由函数 arp_send()和 arp_rcv()实现。arp_send()和 arp_rcv()的主要调用如图 4-34 所示。arp_send()调用 arp_create 创建一个 arp 分组，再由 arp_xmit()调用 dev_queue_xmit 发送 ARP 分组。当接收到一个 ARP 分组时，系统将 arp_process()挂在地址解分组，在 arp_process()中，调用 neigh_lookup()使用源 IP 从注册为邻列键值查找 *nsh_bucket。

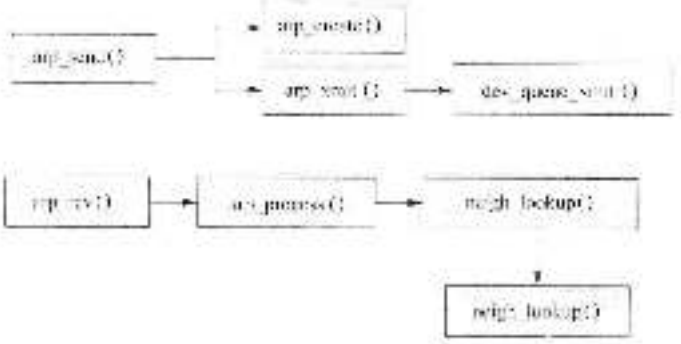


图 4-34 arp_send()和 arp_rcv()的调用图

算法实现

arp_process()的任务是发送应答：如果没有对主机的请求或者有到相当数量的发送人的主机请求时；它也可以处理来自另外一个人（主机请发送请求）的一个应答。对于后一种情况，更新 ARP 表中对应于应答消息的源表项。为了实现这种更新，arp_process()首先调用 neigh_lookup()找到 ARP 表中的对应表项，然后它调用 neigh_update()更新这个表项的标志。

练习

函数 __neigh_lookup()是一种实现散列搜索的常见函数。

1. 用一个免费文本搜索或交叉索引工具来找出哪个函数调用 __neigh_lookup()。
2. 重写 __neigh_lookup()并解释如何从一个散列桶中查找一个表项。

4.4.2 动态主机配置

从 4.2 节中，我们可以观察到每台主机需要主动地配置 IP 地址、子网掩码和默认路由器。（我们还需要在主机上配置与域名服务器有关的参数，这些内容将在第 6 章中讨论。）对于一个机构某个的用户，这样的配置过程没有任何意义，它往往主要成为网络管理人员的负担，因为配置网络层参数每天都会发生错误，所以不值得大惊小怪。不幸的是，与以太网卡的 MAC 地址不同，这些参数不可能在生产制造阶段配置，因为 IP 地址层次结构的缘故。显然，需要有一种自动配置的方法。IETF 建议了一种动态主机配置协议（DHCP）来解决自动化配置问题。

通常，DHCP 遵循客户机/服务器模型。一台充当客户机的主机，将其请求发送给 DHCP 服务器，而服务器用配置信息应答主机。可扩展性仍然是这种客户机/服务器模型设计的主要问题。首先，主机如何到达 DHCP 服务器？一种笨拙的方法就是在每个 IP 子网中部署一个 DHCP 服务器，并让每个 DHCP 客户机将其请求广播到它所在的子网中。但是，这样导致需要太多的服务器。为了解决这个问题，在没有 DHCP 服务器位于子网中可以使用中继代理。中继代理将 DHCP 请求消息转发到 DHCP 服务器，然后再将来自服务器的应答返回给主机。

可用多种方法为主机分配一个 IP 地址。静态配置方法将一个特定 IP 地址映射到一个特定的主机。例如，通过其 MAC 地址标识每一台主机。这样做的好处在于可以更好地管理网络，因为每台主机有唯一的 IP 地址。DHCP 有助于自动配置每台主机的 IP 地址，但当存在网络安全问题时，为了追踪哪台主

机拥有特定的 IP 地址，手动配置就会非常容易。但是，当子网内的主机数大于子网拥有的合法 IP 地址数时，就需要另一种方法——动态配置方法，以便动态地为客户主机分配 IP 地址。在这种方法中，DHCP 服务器先从能够按需分配给主机的 IP 地址池。当主机请求 IP 地址时，DHCP 服务器从地址池中选择一个尚未分配的 IP 地址并将它分配给主机。这种方法的一种更为复杂的使用是每台主机可以将其喜爱的 IP 地址，一般就是上次为它分配的 IP 地址，发送给服务器，服务器在该地址当前可用的情况下会将它分配给主机。为了防止 IP 地址被一台不活跃的主机占用，服务器仅将 IP 地址“租给”一台主机一段有限的时间。在租用时间达到之前，主机需要再次请求 IP 地址，当然，当前指定的 IP 地址将是首选地址。

DHCP 操作

详细的 DHCP 步骤如图 4-35 所示。当一台主机首次启动时，它将广播一条 DHCPDISCOVER 消息，消息装在一个 UDP 分组中并使用端口 67。所有接收到消息的 DHCP 服务器将在 UDP 端口 68 发回一个 DHCPOFFER 报文。如果有多种给予服务（offer）选择，客户机将从中选择一种给予服务（offer）。并向获得给予服务（offer）的服务器发送一条 DHCPREQUEST 消息。如果一切正常，服务器就回应一条 DHCPACK 消息。此时，客户机就配置了 IP 地址和其他由服务器提供的信息。在租赁更新定时器过期之前（典型设置成租赁过期时间的一半），客户机需要再次发送一条 DHCPREQUEST 消息。如果在租用重绑定时间之前没有收到 DHCPACK 消息，客户机就再次发送一个 DHCPREQUEST 消息。当接收到 DHCPNACK 消息或者当租赁定时器过期时，客户机就放弃当前它的 IP 地址。

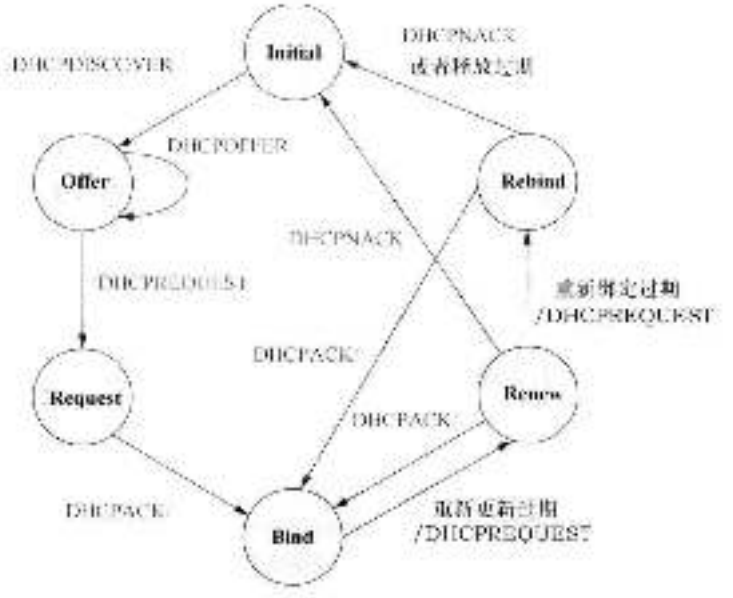


图 4-35 DHCP 的状态图

DHCP 分组格式（RFC 2131）如图 4-36 所示，这是从 BOOTP 导出的。（BOOTP 最初用于激活自动启动配置无盘工作站）。硬件类型用于表示链路层协议，硬件长度是链路层地址的字节长度。跳字段被客户机设置为零，每当经过一个中继代理时就递增 1。如果客户机想要使用广播地址而非其硬件的单播地址接收应答，就需要设置标志中的 B 位。有些字段用于 BOOTP 但未被 DHCP 使用。选项字段用来携带一些额外的信息，如子网掩码。可以将多个选项打包到一个消息中。选择字段是以 4 字节的魔法（magic cookie）0x63825363 开始的，后面跟了一系列选项列表。

每一种选项的格式，如图 4-37 所示，由一个 3 位组头部和跟在后面的数据字节组成。3 位组头部包括 1 位组的代码、1 位组的长度、1 位组的类型字段。为了传输不同类型的 DHCP 消息，代码位设置为 53；类型字段值如表 4-4 所示，指示发送哪种消息。例如，一个 DHCPDISCOVER 消息编码为代码 = 53、长度 = 1、类型 = 1。



图 4-36 DHCP 分组格式

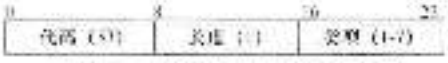


图 4-37 DHCP 头部中的选项字段

表 4-4 DHCP 消息类型

类型	DHCP 消息	类型	DHCP 消息	类型	DHCP 消息	类型	DHCP 消息
1	DHCPDISCOVER	3	DHCPREQUEST	5	DHCPACK	7	DHCPRELEASE
2	DHCPOFFER	4	DHCPDECLINE	6	DHCPNACK		

对于每种类型的 DHCP 消息，额外的选项装入到代码长度类型格式中并被添加到消息的最后。例如，DHCPDISCOVER 消息可以使用代码 50 来指定需要的 IP 地址。在这种情况下，选项中的代码 = 50，长度 = 4，类型 = 需要的 IP 地址。下面是一些常用的选项（RFC 2132）：

代码：0 填充选项
代码：1 子网掩码
代码：3 路由器
代码：6 域名服务器
代码：12 主机名称
代码：15 域名
代码：17 主机启动路径
代码：26 接口最大传输单元（MTU）
代码：40NIS 域名
代码：50 请求的 IP 地址（DHCPDISCOVER）
代码：51IP 地址租用时间
代码：53 消息类型
代码：54 服务器标识符
代码：55 参数请求列表
代码：56 错误消息
代码：57 最大的 DHCP 消息大小
代码：58 更新（T1）时间值
代码：59 重新绑定（T2）时间值
代码：60 供应商类标识符
代码：61 客户机标识符
代码：255 结束选项

图 4-38 显示了一个 DHCPOFFER 消息选项字段



0	8	16	24	31
Op = 0x02	H.T = 0x01	H.Len = 0x06	Hops = 0x00	
事务标志符 = 0x3981691221				
秒数 = 0		0	标志 = 0x0000	
客户机IP地址 = 0x0000				
你的IP地址 = 192.168.1.2				
服务器IP地址 = 0x0000				
中继器IP地址 = 0x0000				
客户机硬件地址 = 00:00:39:1c:86:2a				
服务器主机名字/启动文件名字 = 192字节个0				

选项:
魔法 = 0x63825363
消息DHCP选项
Code: 53; Length: 1; Message Type: 2(Offer)
Server Identifier DHCP Option
Code: 54; Length: 4; Address: 192.168.1.1
IP Address Lease Time DHCP Option
Code: 51; Length: 4; Value: 4294967295
Subnet Mask DHCP Option
Code: 1; Length: 4; Address: 255.255.255.0
Routers DHCP Option
Code: 3; Length: 4; Address: 192.168.101.3
Domain Name Servers DHCP Option
Code: 6; Length: 4; Address: 192.168.1.100
DHCP Option End
Code: 255;

图 4-38 一个 DHCPOFFER 的例子

开源实现 4.7：DHCP

概述

DHCP 作为 BOOTP 协议变种的实现，信息是在以魔饼（magic cookie）0x63825363 开始的选项字段中携带的。检注过这种魔饼（magic cookie）之后，DHCP 消息根据在 RFC 2132 中定义的选项代码进行处理。

数据结构

用于 BOOTP/DHCP 协议的数据结构是 src/net/inv4/ioconfig.c 中的 struct bootp_pkt。

```
struct bootp_pkt {                    /* BOOTP packet format */
struct iphdr iph;                     /* IP header */
struct udphdr udph;                    /* UDP header */
u8 op;                               /* Request, Reply */
u8 htype;                            /* HW address type */
u8 hlen;                             /* HW address length */
u8 hops;                             /* Used only by gateways */
__be32 xid;                           /* Transaction ID */
__be16 secs;                          /* Seconds since we
started */
__be16 flags;                         /* Just what it says */
__be32 client_ip;                      /* Client's IP address
if known */
__be32 your_ip;                       /* Assigned IP address */
__be32 server_ip;                    /* (Next, e.g. NFS)
server's IP address */
__be32 relay_ip;                          /* IP address of BOOTP
relay */
u8 hw_addr[16];                       /* Client's HW address */
u8 serv_name[64];                     /* Server host name */
u8 boot_file[128];                    /* Name of boot file */
u8 exten[312];                        /* DHCP options/BOOTP
vendor extensions */
};
```

算法实现

如果定义了自动配置，那么将调用ip_auto_config()，并且将使用定义的协议(RARP、BOOTP、DHCP)

配置主机的 IP 地址和其他参数。如图4-39所示，如果 DHCP 服务器的 IP 地址已知，从 ip_auto_config() 调用 ic_bootp_send_if() 将 DHCPREQUEST 消息发送到 DHCP 服务器，告别然广播 DHCPDISCOVER 消息。特别地，这些 DHCP 消息的选项，如请求子网掩码和默认网关，是由 ic_dhcp_init_options() 函数设置的。DHCP 客户端在使用请求的 IP 地址前，需要等待一个 DHCPACK，参见 ic_dynamic()。

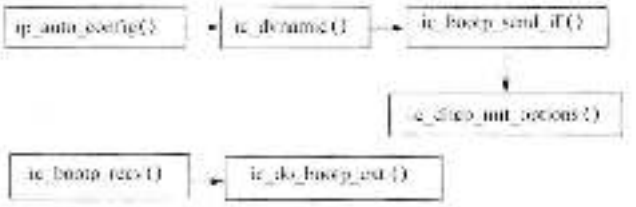


图4-39 DHCP 开源实现的调用图

接收到的 DHCP 信息由 ic_bootp_recv() 函数来处理。只有 DHCPOFFER 和 DHCPACK 消息是在当前实现中处理的。额外的配置信息由 ic_do_bootp_ext() 来处理，目前仅处理代码 1（子网掩码）、3（默认网关）、6（DNS 服务器）、12（主机名）、15（域名）、17（根路径）、26（接口 MTU）、42（NIS 域名）。注意，额外的配置信息总是 DHCP 消息的最后一部分以 0xFF 结束（参见图4-38 中的例子）。

练习

1. 跟踪 ic_bootp_recv() 并解释 DHCP 消息的选项字段是如何处理的。

2. RFC 2132 之后定义了很多新的 DHCP 选项。以 RFC 5417 为例，阅读 RFC 具体看看已经定义了什么选项。

4.5 控制平面协议：错误报告

在互联网上偶尔也会产生错误。例如，一个分组由于 TTL 等于零或者由于不可达的目的地而不能进一步转发。回想你经常在浏览器中看到的错误信息，显示服务器可能死机了。互联网可能以不同方式处理错误。例如，它可能仅忽略错误并悄无声息地丢掉分组。然而，为了调试、管理和跟踪网络状况，将错误报告发送给源节点或中间路由器是一个更好的解决办法。互联网控制报文协议（ICMP）主要用于向源节点报告通过路由器或者主机发现的错误。它也可以用于信息报告。

4.5.1 ICMP 协议

ICMP 可用于报告 TCP/IP 协议的错误和主机/路由器的状态。在大多数情况下，ICMP 作为 IP 的一部分实现。尽管这是一个 IP 层的控制协议，但 ICMP 消息由 IP 分组携带，也就是说，ICMP 位于 IP 之上，如图4-40 所示。因此，ICMP 就等一个 IP 的上层协议。一个 ICMP 消息是在 IP 分组的有效载荷中携带的，IP 报头部的上层协议标识符设置为 1，用于复用和解复用的目的。一个 ICMP 消息由两部分组成：头部和数据。头部有一个类型和代码字段，如图4-41 所示。一个 ICMP 消息的有效载荷可能包含用于信息报告的控制数据，或者为了错误报告它也可能包含头部和错误 IP 分组的部分有效载荷（在 RFC 792 中，数据报中用于触发错误的前 8 个字节用于报告；在 RFC 1122 中，将发送超过 8 个字节；在 RFC 1812 中，路由器应在有效载荷中尽可能多地报告原始数据报，只要 ICMP 数据报长度不超过 576 字节）。为不同类型的 ICMP 消息定义了不同的句法格式。

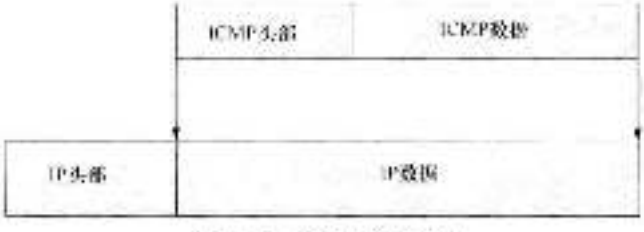


图4-40 IP 之上的 ICMP

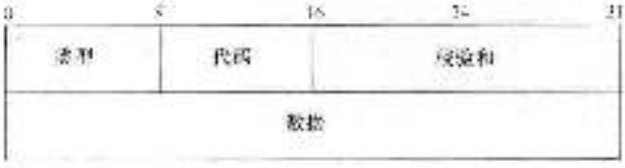


图4-41 ICMP分组格式

IPv4常用的ICMP消息类型和代码列表如表4-5所示。其中4个是信息消息，分别是echo应答和请求、路由器通告和发现，其余为错误消息。为了检测某道一个目的地是否还存在，它可以向目的地发送一个echo（回显）请求消息。目的地收到echo请求后，就用一个echo应答消息作为响应。这两种消息的有效载荷都包含一个16位标识符和一个16位序列号，这样数就可以将应答与对应的请求进行匹配。如图4-42所示，著名的调试工具ping就是使用ICMP echo请求和echo应答消息实现的。

表4-5 IPv4中ICMP的类型和代码

类型	代码	描　述	类型	代码	描　述	类型	代码	描　述
0	0	echo应答（ping）	3	5	源路由失败	8	0	echo请求（ping）
3	0	目的网络不可达	3	6	目的网络未知	9	0	路由公告
3	1	目的主机不可达	3	7	目的主机未知	10	0	路由器发现
3	2	目的协议不可达	4	0	源抑制（拥塞控制）	11	0	TTL为零
3	3	目的端口不可达	5	0	重定向（目的网络）	12	0	坏的IP头部
3	4	需要分段但设置DF	5	1	重定向（主机）			



图4-42 ICMP echo请求和应答消息格式

在ICMP错误消息中，目的地不可达（类型3）用来报告各种不可达（如网络、主机或端口）的原因。然而，类型3消息中的代码4用于报告在中间路由器（由于MTU）需要分段的错误，但在IP头部中设置了不分段位。类型4和5的消息很少在实际中使用。源抑制消息（类型4）用于当分组（由于拥塞）造成缓冲区溢出时让路由器发送一个错误消息给源。一旦收到源抑制消息，源就应该降低它的传输速率。对于一个具有超过两台路由器的IP子网，使用重定向消息（类型5）通知主机有关到达目的地的一条更好的替换路由。通常，更好的路由就是在同一子网内向另外一台路由器发送分组。类型12消息用于报告IP头部中的错误，如无效的IP头部或错误的选择字段。

当IP分组在一台路由器递减后的TTL为零时，发送超时消息（类型11）给源主机。这种类型的消息特别有趣，因为traceroute程序用它来追踪主机到目的地的路由。traceroute程序发送一系列的ICMP消息给目的地：首先它向目标主机发送一个TTL=1的ICMP echo请求消息。当到达目的地路径上的第一台路由器收到该消息时，它返回一个超时ICMP错误消息，因为TTL递减后为0。traceroute程序在接收到超时消息后，向目的地发送另一个TTL=2的echo请求消息。此时，消息会通过第一台路由器但将被第二台路由器丢弃，然后第二台路由器将向源发送另一个超时消息。Traceroute程序用TTL递增的值继续发送ICMP echo请求消息，直到它接收到从目的地来的应答为止。当traceroute程序接收到一个超时消息时，它就学习到路径上的一台路由器。（注意，实际上大多数traceroute程序对于给定的TTL值会发送3个echo请求消息，并记录来自每台路由器的响应时间。）

对于下一代互联网协议，定义了一套新的 ICMP 类型与代码，如表 4-6 所示。ICMPv6 报分组格式与 ICMPv4 相同，但 ICMPv6 类型字段值以一种更容易识别的方式定义，这样错误消息类型小于 127，而信息类型则大于 127 但小于 256。

表 4-6 ICMPv6 类型和代码

类型	代码	描述	类型	代码	描述	类型	代码	描述
1	0	没有到达目的地的路由	4	0	错误的报头字段	132	0	组播监听者完成
1	1	管理性地禁止与目的地的联系	4	1	无法识别的下一个头部类型	133	0	路由器请求
1	3	地址不可达	4	2	遇到无法识别的 IPv6 选项	134	0	路由器通告
1	4	端口不可达	128	0	echo 请求	135	0	邻居请求
2	0	分组太大	129	0	echo 应答	136	0	邻居通告
3	0	在传输中超过跳限制	130	0	组播监听者查询	137	0	重定向
3	1	通过分段重组时间	131	0	组播监听者报告			

开源实现 4.8：ICMP

概述

当分组不能转发或者分组收到某些 ICMP 报告请求（如一个 ECHO 请求）时，就发送一个 ICMP 消息。对于某一对情况，在分组转发过程中发送一个 ip_forward() 或 ip_route_input_slow() 的 ICMP 消息。对于另一种情况，从处理层接收一个 ICMP 消息，并且调用 icmp_rcv() 来处理请求。

数据结构

为了通过不同的处理程序处理不同类型的 ICMP 消息，使用 icmp_pointers[] 表来存储 ICMP 处理程序（参见 src/net/ipv4/icmp.c）。例如，icmp_unreach() 用于类型 3、4、11 和 12；icmp_redirect() 用于类型 5；icmp_echo() 用于类型 8；icmp_timestamp() 用于类型 13；icmp_address() 用于类型 17；icmp_address_reply() 用于类型 18；而 icmp_discard() 用于其他类型。icmp_pointers[] 表建立如下：

```
static const struct icmp_control icmp_pointers[NR_
ICMP_TYPES + 1] = {
    [ICMP_ECHOREPLY] = {
        .handler = ping_rcv,
        .error = 1,
    },
    [ICMP_DEST_UNREACH] = {
        .handler = icmp_unreach,
    },
    [ICMP_SOURCE_QUENCH] = {
        .handler = icmp_unreach,
        .error = 1,
    },
    [ICMP_REDIRECT] = {
        .handler = icmp_redirect,
    },
    [ICMP_ECHO] = {
        .handler = icmp_echo,
    },
    [ICMP_TIME_EXCEEDED] = {
        .handler = icmp_unreach,
        .error = 1,
    },
    [ICMP_PARAMETERPROB] = {
        .handler = icmp_unreach,
        .error = 1,
    },
    [ICMP_TIMESTAMP] = {
        .handler = icmp_timestamp,
    },
    [ICMP_TIMESTAMPREPLY] = {
        .handler = icmp_discard,
    },
    [ICMP_INFO_REQUEST] = {
        .handler = icmp_discard,
    },
    [ICMP_INFO_REPLY] = {
        .handler = icmp_discard,
    },
    [ICMP_ADDRESS] = {
        .handler = icmp_address,
    },
    [ICMP_ADDRESSREPLY] = {
        .handler = icmp_address_reply,
    },
};
```

算法实现

图 4-43 展示了发送和接收 ICMP 消息的调用链。当转发 IP 分组时调用 ip_forward() 来处理分组。如果分组有错误，ip_forward() 将调用 icmp_send() 发送一个 ICMP 消息给源主机。在 ip_forward() 中检验分组的一系列步骤如下：首先，如果分组的 TTL 小于或等于 1，将发送一个 ICMP 超时消息。其次，如果要求严格源路由并且从路由表中获得的下一跳不是分组指定的路由器，那么就发

是一个ICMP目的地不可达消息。再次，如果需要路由重定向，就调用 ip_rt_send_redirect() 重定向分组，它然后调用 icmp_send()发送一个ICMP重新定向消息。最后，如果分组长度大于接口的MTU，并且设置了不分段位，那么就发送ICMP目的地不可达消息（代码=4，"ICMP_FRAG_NEEDED"）。

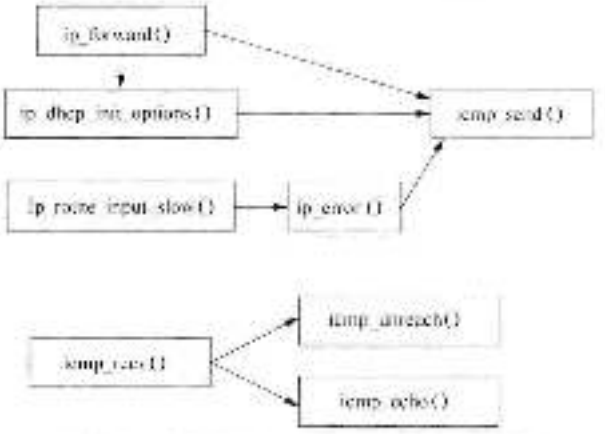


图4-43 发送和接收一个ICMP消息的调用图

当接收到不能与存储在缓存中的所有路由匹配的IP分组时，调用 ip_route_input_slow 函数来处理。如果路由表查询结果返回 RTN_UNREACHABLE，那么就调用 ip_error()，它将调用 icmp_send()向源发送一个ICMP目的地不可到达消息。

最后，让我们学习如何处理到达的ICMP分组。当接收到一个ICMP消息时，网络接口卡的下半部中断处理程序将调用 icmp_rcv()函数，然后根据ICMP消息的类型字段再调用合适的ICMP类型处理程序。大多数ICMP类型是由 icmp_unreach()函数处理的。除了检查接收到的ICMP消息外，icmp_unreach()函数将错误分组传递到适当的上层协议。条件是如果协议的错误处理程序已经定义。收到的echo请求由 icmp_echo()函数处理。如此如果echo应答选项没有禁止，那么将一个echo应答返回给源节点。

最后，ICMPv6函数以相似的方式实现（参见 arch/net/ipv6/icmp.c）。ICMP消息是由 icmpv6_send()发送的，同时调用 icmpv6_rcv()接收ICMP消息。echo应答消息由 icmpv6_echo_reply()回答，其他错误信息（如分组太大、目标不可达、超时和参数问题）由 icmpv6_notify()来处理。如果已经定义了错误处理程序，那么就会将错误分组传递给上层协议。邻居发现是IPv6的一个新的功能特点，由5种消息类型组成：路由器请求、路由器通告、邻居请求、邻居通告、路由重定位。一旦接收到这些类型的消息，就调用函数 ndisc_rcv()（参见 arch/net/ipv6/ndisc.c），同时该函数根据消息类型切换到不同的函数。例如，调用 ndisc_router_discovery()处理路由器通告。

练习

为traceroute程序编写源代码，假定你可以在内核中调用ICMP函数。

4.6 控制平面协议：路由

在数据平面中，我们已经学习了一台路由器如何通过查找其路由表来转发分组。假设可以正确地建立和维护路由表，转发过程就非常简单和直接。不过，所有这些都依赖于路由任务来计算路由和维护路由表。在本节中，我们将首先讨论路由基本原则，然后说明路由是如何在互联网上实现的。

4.6.1 路由原理

IP层的任务就是提供主机到主机之间的连通性。这种连通性允许从一台主机向另一台远程主机发送分组。为了实现这个任务，需要为每对源－目的地建立一条路由（一系列相邻的路由器），以便分组可以沿着路由转发。从源到目的地主机查找路由的任务称为路由。

路由机制需要的属性包括效率、稳定、简单、公平和可扩展的。由于互特网使用分组交换，所以

资源共享和分组的存储转发是由路由器实现的。因此，路由的主要目标就是实现有效的资源共享同时维持良好的性能——例如低时延和低的分组丢失，最优路由的目标应该能最大化资源利用，最小分组延迟和最小化数据分组丢失（注意这些目标可能是相互冲突的）。在互联网中，可扩展性是非常重要的。可扩展的路由包括一个路由表使用的可扩展的数据结构，一种可扩展的路由信息交换机制，一种可扩展的路由计算算法。另外，在路由内不要形成环路非常重要，因为分组循环可能会浪费大量的带宽并使网络变得不稳定。由于在互联网中存在大量的路由器，所以需要健壮的路由以防出现故障的链路或路由器，即防止单点故障影响整个网络。最后，还需要公平性，因为节点也应该同等对待。

有三大类路由：点到点，单点到多点和多点到多点。第一类称为单播路由，而另外两类称为组播路由。对于单播传输，分组是从一个源节点到一个目的地。对组播传输，可能会有一个或多个源主机，分组从这些源主机转发到多个目的地主机。显然，单播路由与组播路由截然不同。更常见的情况是，单播路由就是在源主机和目的地主机之间找到一条路径，而组播路由就是从一个或多个源到多个目的地之间找到多条路由，这通常形成一个树状结构，常称为组播树。在本节中，我们重点介绍单播路由，而组播路由留到下一节讲解。

全球或局部信息

单播路由协议在使用的路由信息类型、路由信息如何交换，如何确定路由等方面区别于其他的路由协议。一条路径可以根据网络的全球（完全）信息或本地（部分）信息计算。如果提供全球信息，路由计算就可以考虑网络中的全部路由器和网络连接状态。否则，路由计算仅考虑来自邻近路由器和链路的信息。在路由器之间需要交换路由信息，以便获得全球或本地网络的信息。通常，全球信息是通过可靠的广播机制获得的，而本地信息可以通过与相邻邻居交换信息获得。

每个确定路由可以通过几个方面来检查。首先，一条路由既可以动态地也可以静态地确定。静态路由表可以由网络管理员手动配置。但是，它们不能适应动态的网络故障。因此，应用路由协议动态地更新互联网中的路由表。其次，可以采用集中式或分布式算法确定路由。集中式的算法需要全局信息，它们既可以在一个中央站点也可以分布式地在每台路由器上运行。互联网上的某些路由协议采纳后一种方法，所谓的半集中式算法，以取得更好的健壮性。但是，某些互联网路由协议使用分布式算法分布式地确定路由。最后，路由可以在每台中间路由器逐跳地确定也可以在源主机上计算。如果路由在每跳（路由器）单独完成，那么既可以采用半集中式的算法也可以采用分布式的算法。互联网采用逐跳路由作为默认路由机制，同时也支持源路由作为可选项。

什么是最佳的路由？不同的应用程序可能有不同的标准。交互式应用（例如，远程登录可能想要一条最小延迟的路由，而多媒体应用程序可能想要一条带宽充足以及低延时和抖动的路由。传统上，一条链路附带了用来描述路由经过该链路所需要的成本。例如，一条链路的成本可以体现链路上的延迟或可用带宽。路由问题就可以建模为一个图论问题，其中节点是路由器，边是链路。将网络转换成一个图后，路由问题就等价于最小成本路径问题。在互联网中采用两种类型的路由算法，以便解决最小成本问题：链路状态路由算法和距离矢量路由算法。我们将详细地学习这两种算法。

逐跳路由的优化

你可能想知道如果路由是在每台路由器上单独完成，那么我们怎样才能确定分组会在最优路由上转发？原因是存在一个互联网逐跳路由的最优原则。也就是说，如果 k 是从源 s 到目的地 d 最优路由上的一个中间节点，那么在从 s 到 d 的最佳路由上的从 s 到 k 的路由也是从 s 到 k 的最优路由。因此，每台路由器可以简单地相信它的邻居，如果这位邻居位于到达远程目的地的最优路由的下一站，那么这位邻居将确实知道如何沿着最优路径将分组转发到目的地。根据最优原则，每台路由器可以以自己为根扩展到网络上的其他路由器来构建最短路径树。

行动原则：最佳路由

在实现中，用一张图来说明路由问题。一张网 $G=(N, E)$ 由一组节点 N 和一组边 E 组成。对应于 IP 路由问题，在图中的一个节点代表互联网上的一台路由器，在两个节点间的边代表两台相邻路由器之间的物理链路。图 4-44 显示了这种图模型的例子。

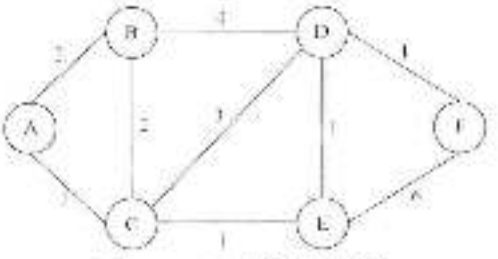


图4-44 路由计算的图模型

路由问题就是要找到一条从源节点到目的地节点之间的路径。显然，在每对源和目的地对之间有很多可选路径。最优路由就是选择每个源到目的地对之间的最佳路由。但是何谓最佳路由？如何定义路径质量？在如图4-45中，我们可以看到每条边都关联了一个成本。在图模型中，路径成本定义为沿着路径上所有边的成本的总和。假设给出了边的成本，最优路由问题就变成了查找最低成本路径问题。此外，如果图中的所有边具有相同的成本，那么最低的成本路径就变成了最短路径。在20世纪50年代，在图论理论文献中提出了一些著名的算法，如Kruskal算法和Dijkstra算法。大部分这些算法其实就是寻找从源节点到图中其他所有节点的最短路径，称为最短生成树或最小生成树。

```
For each v in V-{s} {
        If v is adjacent to s
                C(v)=c(s,v)
                p(v) = s
Else
                C(v)=∞
}
T = {s}
While (T≠V) {
        find w not in T s.t. C(w) is the minimum for all w in (V-T)
        T = T ∪ {w}
        For each v in V-T {
                C(v) = MIN(C(v), C(w)+c(w,v))
                If (C(w)+c(w,v) < C(v)) p(v) = w
        }
}
```

图4-45 Dijkstra算法

显然，如何定义边的成本决定着最小成本路径的质量和含义。在某些路由协议（如RIP）中，将所有边的成本设置为1，最小成本路径就变成了最短路径，即具有最少跳数的路径或穿越最少路由器数量的路径。这似乎是个合理的选择，因为通过路由器会引入额外的处理、传输和排队延迟。但是，每台路由器具有不同的处理能力，每条边也可能有不同的带宽以及来自其他源方的流量。因此，一些其他的路由协议（如OSPF）允许对边定义不同的成本，每一个对应某种服务质量度量（如延迟、带宽、可靠性或分组丢失）。可能支持多张路由表，每张表对应一种QoS类型。总之，虽然边的成本假定在图抽象模型中给出，但如何定义边的成本对于确定最优路径（即最小成本路径）的质量非常关键。

链路状态路由

链路状态路由需要全局信息计算最短成本路径。全局信息是指带有所有链路成本的网络拓扑，并且它是通过将它相邻外出链路的成本广播到网络中所有其他路由器获得的。因此，网络中的所有路由器都有一张网络拓扑和链路成本的一致视图。然后利用Dijkstra算法计算每台路由器的最小成本路径。因为所有路由器使用相同的最小成本路径算法和网络拓扑去构建它们的路由表，分组将以逐跳的方式在最小成本路径上转发（从互联网逐跳路由的最优原则）。

Dijkstra算法计算网络中某节点到所有其他节点的最小成本路径，形成一棵最小生成树。然后根据这个最小生成树构建路由表。Dijkstra算法的基本思想就是迭代地找出到达所有其他节点的最小成本路径。在每次迭代中，选择一条从源节点到一个目的地节点之间的最小成本路径。也就是说，经

过 k 次迭代后，到 k 个目的地节点的 k 条最小成本路径就找到了。因此，对一个有 N 个节点的网络，Dijkstra 算法将迭代 $N-1$ 次才会终止。图 4-45 显示了 Dijkstra 算法的伪编码。在伪编码中使用了以下符号：

- $l(x, v)$：从节点 x 到节点 v 的链路成本。如果节点 x 和节点 v 不是直接连接的，那么从节点 x 到节点 v 的链路成本设置为无穷大。
- $C(v)$：到当前节点的迭代，即从源节点 s 到节点 v 的最小成本路径。
- $p(v)$：沿着最短路径节点 v 的直接前驱（上游）节点。
- T：最小成本路径已知的节点集合。

最初，节点仅需要知道它的外出链路的链路成本。对一个邻接节点的成本设置为和它互连相连链路的成本。算法维护了一个在最小生成树上的节点集合 T。最初，T 仅包含源节点 s。在每次迭代中，它将从不在生成树上的节点中选出具有最小成本 w $C(w)$ 的节点。将节点 w 添加到树（即集合 T）上后，如果从源到 s 的成本因为经过了新添加的节点 w 而减少，那么就更新不在树上的节点 v 的成本。while 循环保证经过 $N-1$ 次迭代后终止，$p(v)$ 记录在最小生成树上 v 的父节点。路由表可以根据 $p(v)$ 来建立。

让我们通过一个实例来更深入地剖析 Dijkstra 算法。考虑图 4-46 中以节点 A 为源节点的网络。图 4-47 汇总了迭代的结果。起初，A 仅知道到 B 和 C 的成本分别是 4 和 1，到 D 和 E 的成本是无穷大，因为它们没有连接到节点 A。在每次迭代中，选择具有最小成本但又不包含在集合 T 中的一个节点（随机选择就会断开某个存在的连接）。因此，在第一次迭代中，选择节点 C 并添加到集合 T 中，这也意味着节点 A 到节点 C 的最小成本路径现在确定并且成本等于 1。有了该信息，所有其他节点现在就能够通过节点 C 连接到 A 了。（途经路由的最优原则）。例如，D 和 E 现在可以从 D（或 E）到 C 和从 C 到达 A 最小成本的总和通过 C 到达 A。我们还可以观察到，如果路径经过 C，那么到 B 的成本还可以进一步降低，即从 A 到 C，然后再从 C 到 B。在第一次迭代结束时，C 已经添加到集合 T 中。此外，B、D 和 E 已经更新了从 A 到达它们的最小成本。在第二次迭代中，E 具有最小成本，因此添加到集合 T 中。从 A 到 D 的最低成本也被更新，因为从 A 到 D 经过 E 的路径具有的成本比通过 C 的路径成本小。循环继续直到所有的节点都添加到集合 T 中，如图 4-47 所示。通过使用前驱节点的信息就可以构建从 A 到达所有其他节点的最小成本路径。不要忘记路由算法的任务是构建路由表。构建了从 A 到所有其他节点的最小成本路径后，A 最后的路由表如图 4-48 所示。例如，图 4-47 的结果显示从 A 到 D 的最小成本路径是 $A->C->E->D$，路径成本为 3。因此，到 4-48 中从 A 到 D 的下一跳是 C，成本为 3。

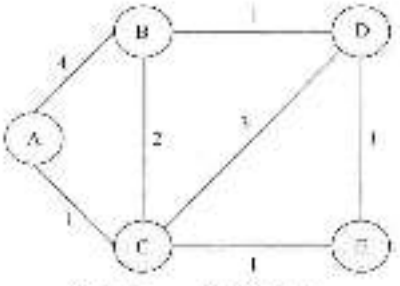


图 4-46 一个网络实例

迭代	T	$C(B), p(B)$	$C(C), p(C)$	$C(D), p(D)$	$C(E), p(E)$
0	A	$4, A$	$1, A$	∞	∞
1	AC	$3, C$		$4, C$	$3, C$
2	ACE	$3, C$		$3, E$	
3	$ACEB$			$3, E$	
4	$ACEBD$				

图 4-47 针对图 4-46 中的网络运行 Dijkstra 算法的结果

目的地	费用	下一跳
B	3	C
C	1	C
D	3	C
E	3	C

图4-48 图4-46网络中节点A的路由表

距离矢量路由

距离矢量算法是另一种用于互联网中的主要路由算法。全路状态算法是一种使用全局信息的半集中式算法，而距离矢量算法则是一种利用局部信息的异步、分布式算法。它仅使用从直接连接的邻居交换的信息。分布式的Bellman-Ford算法用于异步地计算最小成本路径。也就是说，与全路状态路由不同，它不要求所有的路由器交换全路状态信息并且在同一时间计算路由表。每台路由器在收到来自邻居的新路由信息时便会重新计算路由。在重新计算后，新的路由信息将会发送给它的邻居。

图4-49显示了距离矢量路由算法的伪代码。一开始，每台路由器只知道到它直连邻居的路径成本，正如在Dijkstra算法中一样。然后每台路由器异步地运行如图4-48所示的算法。当一台路由器有新的路由信息（如到目的地的新的最低成本）时，它就将路由信息发送到直接相连的邻居。当路由器收到来自邻居的路由信息后，如果有必要它就更新其路由表。路由信息可能包含对路由器来讲是新的到目的地的成本。在这种情况下，就创建一条新的路由表项，到目的地的成本可以将到邻居的成本加上从邻居到目的地的成本来得（后一成本来自路由信息）。如果到目的地的路由成本已经存在于路由表中，那么路由器将确定是否新的成本会导致出现一条新的最小成本路径。也就是说，如果到邻居的成本加上从邻居到目的地的成本之和低于路由表中记录的成本，那么就用新的成本更新路由表项。而邻居就成为到达目的地的新的下一跳。

```
While (1) {
    If node x received route update message from neighbor y {
        For each (Dest, Distance) pair in y's report {
            If (Dest is new) { /* Dest not in routing table */
                Add a new entry for destination Dest
                rt(Dest).Distance = Distance+c(x,y)
                rt(Dest).NextHop = y
            }
        else if (Distance+c(x,y) < rt(Dest).Distance)
            /* y reports a shorter distance to Dest */
            rt(Dest).Distance = Distance+c(x,y)
            rt(Dest).NextHop = y
        }
    }
    Send update messages to all neighbors if route changes

    Also send update messages to all neighbors periodically
}
```

图4-49 距离矢量路由算法

让我们再次以图4-46作为一个例子，并利用这个例子来说明节点A如何计算其基于距离矢量算法的路由表。由于距离矢量算法异步地运行，所以当每台路由器上的路由表异步地更改时就很难给出一张有关整个网络的活动图。因此，在我们的演示中，假设算法对节点运行在每台路由器上。也就是说，我们假设每台路由器与它的邻居同时交换新的路由信息。交换路由信息后，每台路由器同时计算新的路由表。最后上述过程，直至每台路由器的路由表收敛到一个稳定的状态（我们检查每台路由器的最终路由表是否与Dijkstra算法计算出来的相同）。

最初，节点A只知道到其邻居的成本，如图4-50所示。然后，节点A将有关路由表的矢量通告它的邻居。同样，节点B和C也向节点A发送它们的最新路由表信息。例如，节点B告诉节点A它到节点C和D的成本分别是2和1。基于这些信息，节点A为节点D创建一条新的成本为5（4+1）的路由表项。同样，

节点 C 也会告诉节点 A 它到节点 B、D 和 E 的成本分别是 2、3 和 1。有了这些信息，节点 A 更新其到 B 和 D 的成本为 3（1+2）、4（1+3）。既然节点 E 对节点 A 来讲是新的，那么节点 A 也为节点 E 创建一条成本为 2（1+1）的路由表项。此时，所有节点都更新了自己的路由表，它必须再次告知它们的邻居有关新的路由表信息（请注意，节点 C 同时知道其到节点 D 的最小成本，如图 4-51 所示）。当节点 A 收到来自节点 B 和 C 的新路由信息时，最后需要更新的就是到节点 D 的最小成本。节点 C 告诉节点 A 其新的成本是 2 而不是 3。因此，从节点 A 到节点 D 的新的最小成本为 3。如果不再有新的成本更新，那么每个节点得到的最终路由表如图 4-52 所示。读者应注意到，这与由 Dijkstra 算法计算出的结果完全相同。

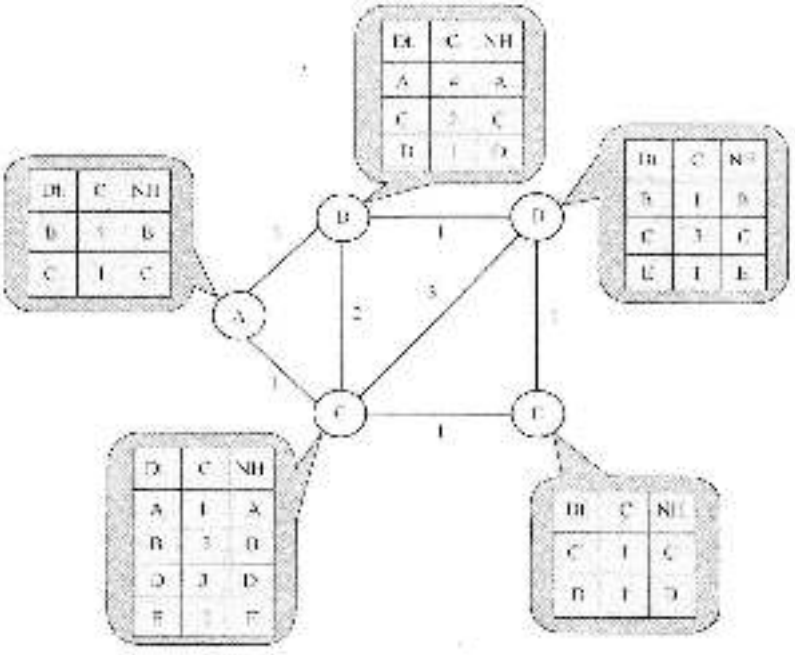


图 4-50 图 4-46 中节点的初始路由表

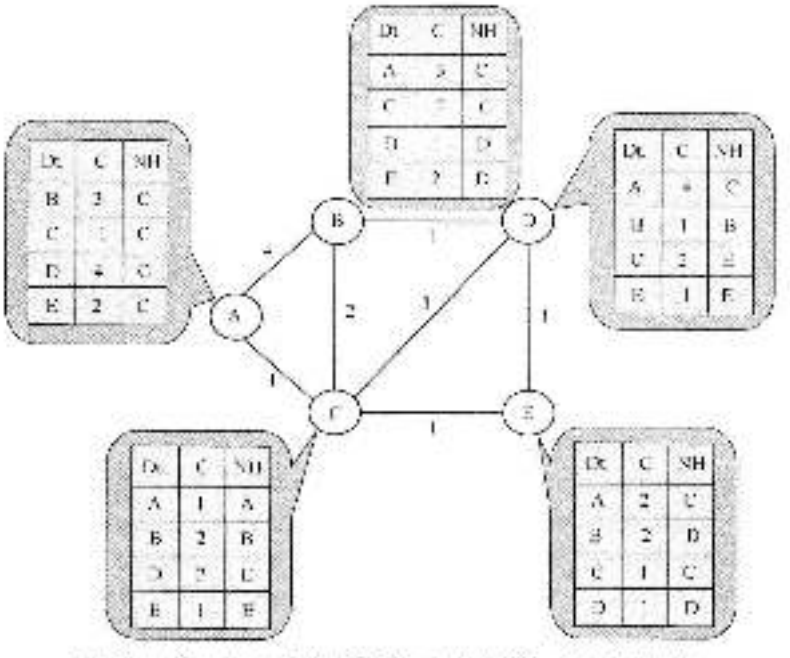
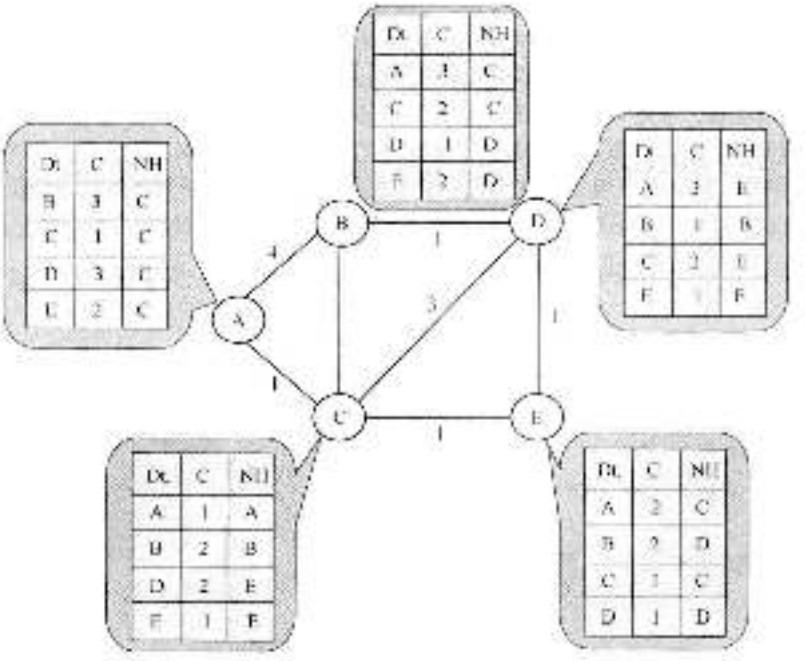


图 4-51 图 4-46 中节点中的路由表（直到第二个步骤为止）


A
B
C
D
E
4
1
3
1
1
1

也就是说，节点 C 认为应该经过 E 来转发目的地为 A 的分组，而节点 E 也认为这些分组应该转发到节点 C。然后分组将在节点 C 和 E 之间永远往返下去。问题是，节点 C 和节点 E 不会在很短的时间内学习到正确的路由。让我们继续讨论我们的例子看看所有节点何时才能学习到坏消息。当节点 C 更新其路由表后，就向邻居发送路由更新消息。节点 B、D 和 E 将其最小成本分别更新为 5、3、4。经过节点 E 将这个新的更新消息发送到 C 后，节点 C 将其到 A 的新成本更新为 5。此过程重复直到节点 B、C、D 和 E 都了解到，它们到 A 的最小成本路径都是通过 B 而不是 C。因为 A 和 B 之间的链路成本相当大，所以路由表收敛之前它将经过 25 次迭代路由更新。如果 A 和 B 之间没有链路，那么将重复步骤。直至到节点 A 的成本如此巨大，以至于节点 B、C、D 和 E 认识到到 A 的成本是无穷大的。因此，这种坏消息传播慢的原理又称为“计数到无穷大”的问题。

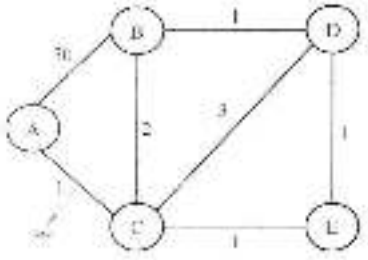


图 4-54 在距离矢量算法中坏消息缓慢地传输

在实践中已采用多个部分的解决方案以处理回路问题。从前面的例子中，我们可以观察到回路问题的发生，是因为节点 C 不知道从节点 E 到节点 A 的最小成本路径通过它本身。因此，最简单的解决方案称为水平分割，就是禁止节点 E 告诉节点 C 其到节点 A 的最小成本。在一般情况下，路由器不应该告诉其邻居网从它们身上学到的最小成本路由。例如，由于节点 E 从节点 C 学习到其到节点 A 的最小成本路径，那么节点 E 就不应该在发送给节点 C 的消息中包括它到节点 A 的最小成本路径。在一个更强大的，称为毒性逆转的方法中，节点 E 应该告诉节点 C 有关它到节点 A 的最小成本是无穷大的。不幸的是，这两种解决方案，仅解决了涉及两个节点的回路问题。对于更大的路由回路，就需要一种更加复杂的机制，例如，在路由更新消息中添加下一跳信息来解决。另一种被某些商业路由器采用的解决方案是使用抑制定时器。在这种方法中，路由器将保持其最小成本路径信息一段时间，时间长度等于路由更新之前抑制定时器的值。例如，继续前面的例子，当节点 E 从节点 C 接收到路由更新并且知道到节点 A 的最小成本变成无穷大时，节点 E 既不应该更新其路由表，也不应该向节点 C 发送新的路由更新直到抑制定时器过期为止。这将阻止节点 C 从所有其他节点接收 A 节点的最小成本路径信息，并给节点 A 和节点 C 一段时间，以便让所有其他节点知道节点 A 和节点 C 之间的链路故障。

层次化路由

在互联网上的路由器的数量非常巨大。因此，为了可扩展性，没有将路由器连成一个平面网络。否则，无论是链路状态算法，还是距离矢量算法都不能扩展为包括成百上千乃至数万台路由器的网络。如果将互联网上的所有路由器连接成一个平面网络，想象一下路由表项的尺寸有多大。还有另外一个我们喜欢将路由器分割成组的原因是：行政自主权。例如，有许多互联网服务提供商（ISP），每个 ISP 都有其自己的路由器和骨干网。自然，每个 ISP 都想要完全控制路由器和骨干网带宽，以便它可禁止来自从其他互联网服务供应商的流量通过其骨干网。因此，互联网路由器组织成两个层次。在低层，路由器分为行政域或自治系统（AS）。在同一个 AS 内的路由器处于相同的行政控制并且运行相同的路由协议，称为域内路由协议或内部网关协议（IGP）。从一个 AS 中选择的，称为边界路由器的设备，将具有连接到其他 AS 边界路由器的物理链路。边界路由器负责将分组转发到外部 AS。在这些边界路由器之间运行的路由协议，就是指域间路由协议或外部网关协议（EGP），可能与域内路由协议不同。

因此，互联网可看做是一组相互连接的自治系统。有三种类型的 AS：桩 AS、多穴 AS 和转送 AS。许多用户通过园区网或企业网访问互联网，这就是典型的桩 AS。由于桩 AS 仅有一个边界路由器并且仅连接到一个 ISP，所以没有转送流量通过桩 AS。多穴 AS 可能有多个边界路由器并且连接到多个 ISP

但是，多穴 AS 也不允许转送流量通过。大多数 ISP 需要允许转送流量并有许多边界路由器连接到其他 ISP。因此，它们称为转送 AS。

在 4.6.2 节和 4.6.3 节中，我们将分别学习域内路由和域间路由。图 4-55 显示了一个简单的由 A、B 和 C 三个域（AS）组成的网络。在每个域中，有多台域内路由器，例如，B 域内的域内路由器 B.1、B.2、B.3 和 B.4。这些路由器之间将运行内部网关协议以便建立并保持它们的路由表。A.3、B.1、B.4 和 C.1 是边界路由器，运行外部网关协议以便交换路由信息。A 和 C 域为单穴 AS，因为它们不允许转送流量，而 B 域是一个转送 AS。让我们来解释域内路由和域间路由每时用来从 A 域中的主机向 C 域中的目的地发送分组。首先，根据域内路由的结果，所有源自 A 域和发向 C 域的分组都需要发送到 A 的边界路由器 A.3 上。路由器 A.3 根据域间路由的结果将这些分组转发给 B.1。路由器 B.1 根据域间路由知道将这些分组转发给 B.4，但根据域内路由知道实际到 B.4 的路径；也就是说，在 B.1 和 B.4 之间路由是根据域内路由发现的）。最后，路由器 B.4 根据域间路由的结果将这些分组转发给 C 域的边界路由器 C.1。然后路由器 C.1 根据域内路由的结果将这些分组转发给适当的路由器。

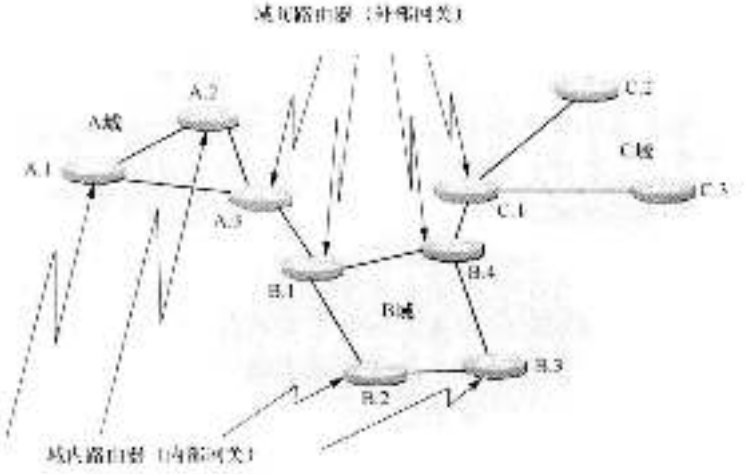


图 4-55 域间 AS 和域内 AS 的路由

最后，让我们来简要地讨论互联网路由的可扩展性问题。如果在图 4-55 中的所有路由器都按着一个平面网络，那么 10 台路由器中的每台路由器都需要知道在其他网络中的另外 9 台路由器的路由信息。然而，使用两个等级的层次化组织，每台路由器只需与两台或三台路由器通信。首先汇总一个域的路由信息并在边界路由器（或外部网关）之间交换，然后将汇总信息传播到所有的内部路由器。通过限制需要通信和交换路由信息的路由器数量来取得可扩展性。

4.6.2 域内路由

AS 是由多个管理网络通过路由器连接起来的。在这些网络之间连接起路由器的任务，前面曾经讲过，在一个 AS 中的路由器处在相同的行政控制之下。因此，AS 的网络管理员能够控制所有的路由器，并决定如何配置这些路由器，在这些路由器上运行什么路由协议，以及如何设置链路成本。假定同样的配置和路由协议，路由协议发现的最优路径即最小成本路径，能够反映管理员考虑的路由质量。例如，如果根据延迟设置链路成本，那么管理员将首选具有更短的延迟。通常，链路成本是按照 AS 内的资源共享效率设置的。

称为域内路由协议或内部网关协议（IGP）的路由协议，用于维持每台路由器的路由表，以便取得 AS 中的所有路由器之间的连通性。在实际中，常用的两种域内路由协议分别是路由信息协议（RIP）和开放最短路径优先（OSPF）。我们从以下几个方面学习这两个协议：使用什么样的路径选择算法、它如何运作，可扩展性和稳定性的考虑，分组格式和开源实现。

RIP

RIP 协议是使用最广泛的域内路由协议，它最初是为施乐 PARC 通用协议设计的，用于施乐网络

系统（XNS）架构。它的广泛使用是由于1982年将其（routed守护进程）包含到广受欢迎的伯克利软件分发（BSD）的UNIX版本中。第一个RIP版本（RIPv1的）定义在RFC 1058中，RIPv2的最新更新定义在RFC 2453中。

RIP是一个典型的距离矢量路由协议。这是一个非常简单的路由协议，用于小型网络。RIP使用的保守成本度量是跳数，即所有学路的成本都为1。此外，RIP通过路径的最大成本为15，即具有成本为16的路径表示不可达。因此，只适合直径小于15跳的小型网络。RIP协议使用两种类型的消息：请求和响应。响应消息也称为RIP通告。这些消息使用UDP端口52发送。因为距离矢量算法用于找到最小成本的路径，只要有一条通路的成本发生变化，相邻路由器就将发送RIP通告给它们的邻居。每个通告最多可以包含25条路由表项，即距离矢量。每个路由表项都包含一个目的网络地址、下一跳、以及到目的网络的距离。RIP支持多个地址簇。也就是说，目的网络地址使用簇字段和目标地址字段来指定。在RIP中，路由器定期地向邻居发送RIP通告，反认时间为30秒。此外，每个路由表项关联两个定时器。第一个是路由无效定时器，称为超时。如果超时定时器到期之前没有收到路由更新，那么这条路由表项便标记成无效（过时）表项。该定时器的默认值是180秒。一旦将一条表项标记为无效，便开始删除过程，发置第二个定时器，又称为垃圾收集定时器。为120秒，路由的成本也设置成16（无穷大）。当垃圾收集定时器过期时，路由便从路由表中删除。

RIP采用多种机制解决距离矢量路由的稳定性问题。首先，将路径成本限制为15，以便能够迅速地确定逆路故障。还采用针对回路问题的三个部分解决方案，即水平分裂、毒性逆转、稳定（抑制）定时器。如上所述，水平分割能够抑制反向路由上的更新。毒性逆转形式地向邻居发送更新，但是对于从邻居学习到的路由，毒性逆转将它们的路由度量在更新中设置为无穷大。稳定定时器避免太快地发送路由更新。

RIP分组格式

RIP的第二个版本比第一个版本具有更好的可扩展性。例如，RIPv2支持CIDR，允许任意长度前缀的路由聚合。RIPv2的分组格式如图4-56所示。每个分组填充了路由表项。每个路由表项包括地址（协议）簇、目的地址、子网掩码、下一跳和距离等信息。

0	8	16	24	31
命令	版本	必须为零		
网1的簇		网1的路由标签		
网1的地址				
网1的子网掩码				
网1的下一跳				
到网1的距离				
网2的簇		网2的路由标签		
网2的地址				
网2的子网掩码				
到网2的下一跳				
到网2的距离				
■■■■■				



图4-56 RIPv2的分组格式

RIP实例

让我们来看一个RIP路由表的例子。例4-57所示的路由表是一个大学院系的边界路由器（仅显示了部分路由表）。该路由器有多个端口，其中之一连接到AS边界网关140.123.1.250；其余的接口连接到本地IP子网上。VLAN将整个院系分割成多个VLAN。支持CIDR，因此目的网络地址关联了一个子网掩码的长度。大部分路由从RIP通告（标志R）中学习。直接连接的子网具有零成本并通过手动配置（标志C）。路由表中也显示了每个路由表项的更新定时器。

目的地	网关	距离/跳	更新定时器	标志	接口
35.0.0.0/8	140.123.1.250	120/1	00:00:28	R	Vlan1
127.0.0.0/8	自接连接			C	Vlan0
136.142.0.0/16	140.123.1.250	120/1	00:00:17	R	Vlan1
150.140.0.0/16	140.123.3.250	120/1	00:00:08	R	Vlan1
140.123.230.0/24	自接连接			C	Vlan230
140.123.240.0/24	140.123.1.250	120/4	00:00:22	R	Vlan1
140.123.241.0/24	140.123.1.250	120/3	00:00:22	R	Vlan1
140.123.242.0/24	140.123.1.250	120/1	00:00:22	R	Vlan1
192.168.100.0/24	140.123.1.250	120/1	00:01:04	R	Vlan1
0.0.0.0/0	140.123.1.259	120/3	00:00:08	R	Vlan1

图 4-57 来自 cs.nsu.edu.tw 的 RIP 路由表

开源实现 4.9：RIP

概述

路由协议的大多数开源实现，如 routed 和 gated，运行在用户空间，作为应用层的用户进程实现。路由协议可以通过 TCP 或 UDP 发送和接收消息（参见图 4-58）。自从 1996 年以来，Zebra 项目（http://www.zebra.org），一种在 GNU 通用公共许可证下自由发布的路由软件，已经成为在路由协议中开发实现的主要参与者之一。

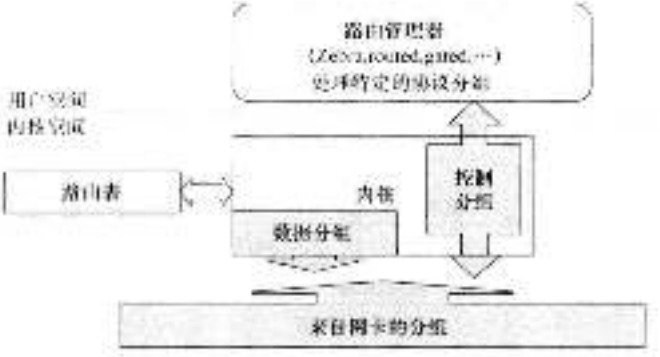


图 4-58 作为用户进程实现的路由协议

Zebra

Zebra 的目标是为提供所有全功能路由协议的可靠路由服务器。它支持多种常用的路由协议，如 RIPv1、RIPv2、OSPFv2、BGP-4（参见表 4-7）。Zebra 软件的模块化设计允许它支持多种路由协议，即 Zebra 为每个协议都有一个进程。模块化也使得 Zebra 灵活和可靠。每个路由协议可以独立进行升级，一个路由协议的软件不会影响整个系统。Zebra 另一个先进的功能是它采用多线程技术。Zebra 的这些特点使它成为一个顶级质量的路由引擎软件。Zebra 的当前版本是 2005 年发布的测试 0.95A。Zebra 支持的平台包括 Linux 操作系统、FreeBSD、NetBSD 和 OpenBSD。2003 年，从 GNU Zebra 分离出一个新的项目，称为 Quagga（http://www.quagga.net），它旨在建立一个以起 Zebra 更具包容性的社团。

表 4-7 Zebra 支持的 RFC

守护进程	RFC#	功 能	守护进程	RFC#	功 能
ripd	2453	管理 RIPv1、v2 协议	ospf6d	2740	管理 OSPFv3 协议
ripngd	2080	管理 RIPng 协议	bgpd	1771	管理 BGP-4 和 BGP-4+ 协议
ospfd	2328	管理 OSPFv2 协议			

框图

下面，我们将使用 Zebra 作为路由协议开源实现的例子，研究 RIP、OSPF 和 BGP 在 Zebra 中的实现。在我们查看每个路由协议在 Zebra 中的实现之前，先讨论 Zebra 的一般软件架构。图 4-59 显示了 Zebra 的架构，路由守护进程与 Zebra 守护进程通信，反过来又能通过各种不同的 API（如 netlink 和 rtnetlink）与内核进行通信。

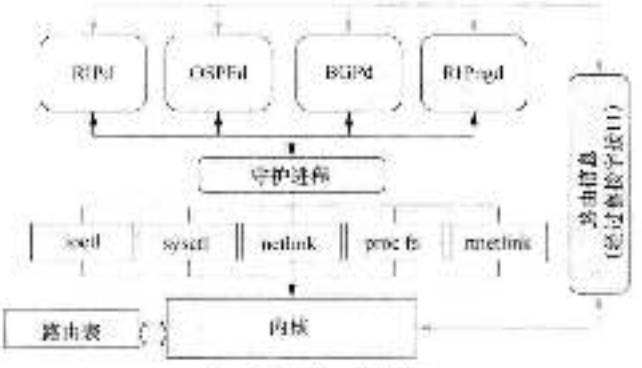


图 4-59 Zebra 的架构

路由守护进程和 Zebra 守护进程之间的相互作用按照客户机/服务器模型进行，如图 4-60 所示。在同一台机器上可以运行多个路由协议。在这种情况下，每个路由守护进程（进程）都有自己的路由表，但它们需要与 Zebra 守护进程通信以便改变内核的路由表。

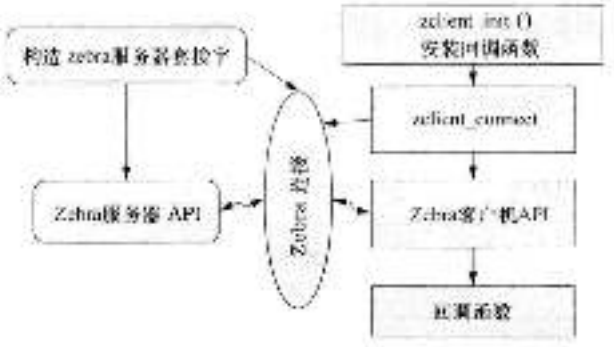


图 4-60 Zebra 的客户机/服务器模型

数据结构

Zebra 的全局路由表入口在数据结构 vrf_vector 中描述。vrf_vector 由一组动态路由表和一组静态路由配置构成，如下所示。

```
struct vrf {
    u_int32_t id; /* identifier (routing table
vector index). */
    char *name; /* Routing table name. */
    char *desc; /* Description. */
    u_char fib_id; /* FIB identifier. */
    struct route_table *table[AFI_MAX][SAFI_MAX];
      /* Routing table. */
    struct route_table *stable[AFI_MAX][SAFI_MAX];
      /* Static route configuration. */
};
```

每个 route_table 由路由表项树构成。每个路由表项由结构 route_node 来描述。在 route_node 结构中，两个重要变量是 prefix（struct prefix p;）和 info（void * info;），它们分别描述这个路由表项的实际前缀和路由信息。每个路由选择特定义自己的这些结构实例。例如，RIP 选择将变量 info 描述成 struct rip_info 的一个指针。

算法实现

路由进程通过一组函数（如 vrf_create()、vrf_enable()、vrf_lookup()、route_node_lookup()、route_node_get()、route_node_delete()）维护本路由表中的路由表和路由节点。例如，RIP 进程调用 route_node_get(rip->table,(struct prefix *)&p)获得前缀 p 的路由节点。

RIP 守护进程

概述

RIP 协议作为路由守护进程，称为 ripd。

数据结构

数据结构定义在 ripd/ripd.h 中，包括 RIP 分组格式的 rip packet 结构，rte 结构在 RIP 分组中用于路由表表项，结构 rip_info 用于 RIP 路由信息（由 route_node 指向其表述这一个节点在路由表中的详细信息）。在 RIP 分组中的 rte 包括四个重要的组成部分：网络前缀、子网掩码、下一跳、路由度量（距离），如下所示。

```
struct rte
{
  u_int16_t family;     /* Address family of this route. */
  u_int16_t tag;        /* Route tag which included in
RIP2 packet. */
  struct in_addr prefix;    /* Prefix of rip route. */
  struct in_addr mask;  /* Netmask of rip route. */
  struct in_addr nexthop;  /* Next hop of rip route. */
  u_int32_t metric;    /* Metric value of rip route. */
};
```

正如前面所述，RIP 的最大度量是16。这个最大值的定义，也可以在 ripd/ripd.h 中找到，如下所示。

```
#define RIP_METRIC_INFINITY       16
```

算法的实现

图 4-61 展示了 ripd 的调用图。

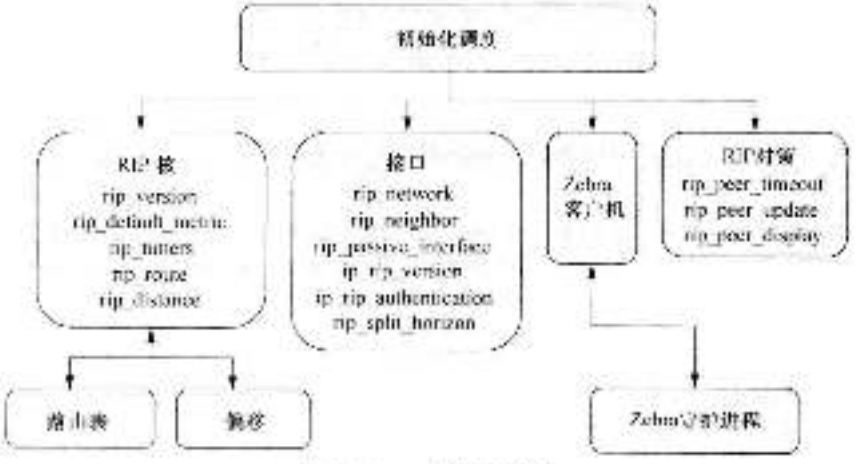


图4-61 ripd 的调用图

Bellman-Ford 算法由定义在 ripd/ripd.c 中的 rip_rte_process()函数实现。当接收到一个 RIP 分组时，此 RIP 分组中的 rte（路由表表项）作为参数调用 rip_rte_process()。依据 rte 的前缀，调用 route_node_get()以便从路由表中获取节点信息（route_node）。一旦通过“info”指针获得了 RIP 路由信息（rip_info），就执行 Bellman-Ford 算法。换句，如果这个节点没有 RIP 路由信息，那么前缀（Dest）一定是新的。通过调用 rip_info_new()创建一个新的 rip_info 结构，然

后，将 rtm 的下一跳和距离（度量）复制到新的表项中。最后，调用 zclip_zebra_ipv6_add() 将新的路由节点添加到路由表中。类似，如果 rtm 给有效（Dest）指定一个较的距离，那么 zlog_zebra_proncess() 中的代码对路由表中该前缀的路由节点执行路由更新。

练习

调用 zclient_node_get() 计算并显示前缀类型 rnode_node。

OSPF

开放最短路径优先（OSPF）是另一种常用的域内路由协议，是 RIP 协议之后出现主导地位的域内路由协议。OSPF 的第 2 版和基于 IPv6 的扩展版本分别定义在 RFC 328 和 RFC 5340 中。与 RIP 不同，OSPF 是一种链路状态路由协议。链路状态信息会洪泛到域内的所有路由器上。每台路由器利用 Dijkstra 算法计算与链路状态对时会将自己作为树根，然后根据该树构建路由表。

OSPF 具有很多独特的特点，这使得它先于 RIP 协议。首先，对于负载平衡，OSPF 支持相同成本的多路径路由。其次，为了支持 CIDR 路由，每条路由都由一个前缀长度来描述。再次，组播路由可以根据单播路由的结果。组播路由协议，组播 OSPF（MOSPF），使用与 OSPF 相同的拓扑数据库。另外，为了稳定性和安全性，一条路由信息附带了 8 字节的口令字段用于认证。最后，为了可扩展性，OSPF 在两层结构。因此可以将一个 OSPF 自治系统进一步划分为多个区域。一个区域就是一组相邻的网络和主机。一个区域的拓扑结构对外界是不可见的。因此自治系统内的路由分成两层：域内路由和域间路由。

层次化的 OSPF 网络

图 4-62 展示了一个 OSPF 网络的层次化结构。从图 4-62 中我们可以看到，路由器分为四种：两种边界路由器和两种内部路由器。一个区域由多台内部路由器和一台或多台区域边界路由器组成。内部路由器仅执行区域内路由并从区域边界路由器那里学习有关外部区域的路由信息。一台区域边界路由器同时参加区域间和区域内路由。它负责汇总 AS 内部和外部的其他区域的路由信息，并在整个区域内广播路由信息。AS 边界路由器参与区域内路由（在区域同层次）和区域间路由。它运行 OSPF 获得 AS 中的路由信息并运行一些外部路由协议，如 BGP，学习 AS 外的路由信息。外部路由信息不加修改地在整个 AS 间传播。每一个路由器层联系 AS 边界路由器与区域边界路由器的中介路由器。

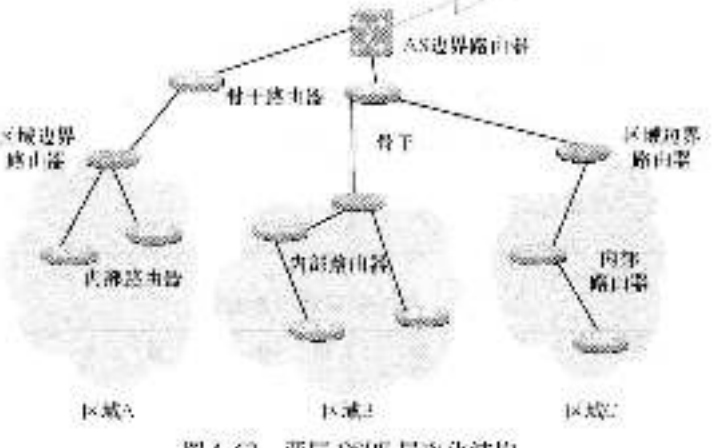


图 4-62 两层 OSPF 层次化结构

OSPF 例子

让我们以图 4-63 作为例子来解释两层的层次化路由是如何在 OSPF 中实现的[2]。图 4-63 中的 AS 由 5 台内部路由器（RT1、RT2、RT8、RT9 和 RT12）、4 台区域边界路由器（RT3、RT4、RT10、RT11）、1 台骨干路由器（RT6）和 2 台 AS 边界路由器（RT5、RT7）组成，配置到三个区域。区域 2 是一个特殊的称为白骨（stub）的区域。如果一个区域只有一个出口点，那么它就可以称为一个桩。

[2] 例子来自 RFC 2328。

将一个区域配置成桩的目的是为了避免将外部路由信息广播到桩区域。AS 由 11 个子网（N1～N11）组成并与 4 个外部网络（N12～N15）相连接。注意，在图 4-63 中的链路成本是有向的。也就是说，一条链路的两端点可能会给链路分配不同的成本。比如，RT3 和 RT6 之间的链路 RT3→RT6 和 RT6→RT3 的成本分别是 8 和 6。

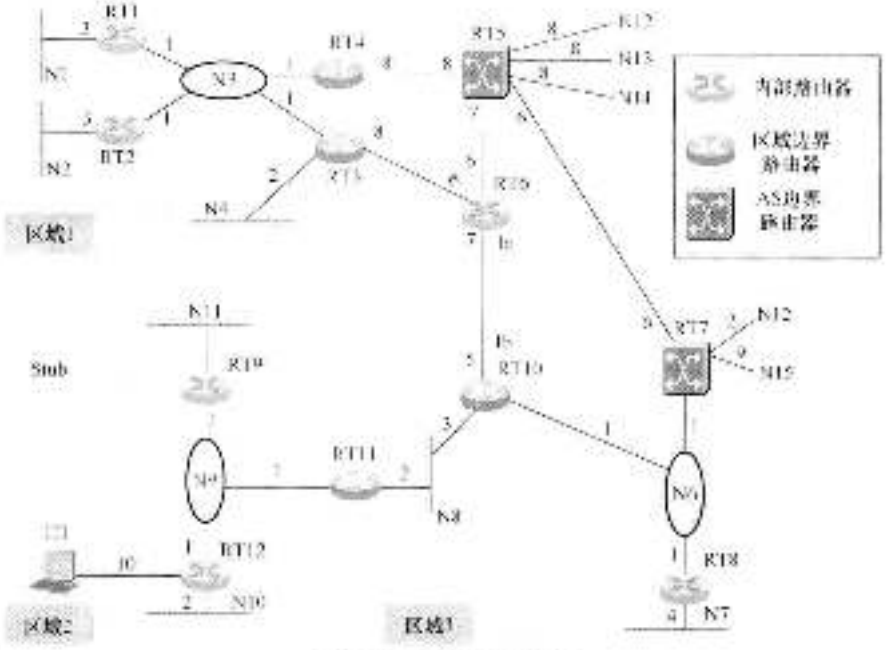


图 4-63 OSPF 网络例子

让我们首先看看区域 1 的内部路由。通过泛洪交换路由信息后，区域边界路由器 RT3 和 RT4 通过使用 Dijkstra 算法计算最短路径。然后将汇总后的路由信息通过区域间路由通告到 AS 骨干。表 4-8 显示了由 RT3 和 RT4 通告的路由。对于内部路由器 RT1 和 RT2，域内网络的路由表以相同的方式构建。表 4-9 显示了 RT1 的区域内部路由表。

表 4-8 通过 RT3 和 RT4 通告给骨干的路由

网 络	由 RT3 通告的成本	由 RT4 通告的成本	网 络	由 RT3 通告的成本	由 RT4 通告的成本
N1	4	4	N3	1	1
N2	4	4	N4	2	3

表 4-9 RT1 的区域内路由表

网 络	成 本	下 一 跳	网 络	成 本	下 一 跳
N1	3	直接	N3	1	直接
N2	4	RT2	N4	5	RT3

然后区域边界路由器在 AS 骨干上与其他路由器交换区域内路由汇总。每个区域边界路由器将会收到所有来自其他区域边界路由器的汇总。根据这些路由汇总，每个区域边界（路由器）还是根据 Dijkstra 算法形成一张到其区域外所有网络的距离图。然后区域边界路由器汇总，并且在每个区域泛洪整个 AS 路由。表 4-10 显示区域间路由经过 RT3 和 RT4 通告进入区域 1 中。注意，将区域 2 配置为一个单臂网络，因此将 N9、N10、N11 的路由信息压缩到一个入口表项。请看，一个网络如果仅有一个出口点就将其配置为一个桩区域。外部 AS 路由信息不能泛洪进入或通过整个桩区域。

表 4-10 RT3 和 RT4 将路由通告到区域 1

目的地	由 RT3 通告的成本	由 RT4 通告的成本	目的地	由 RT3 通告的成本	由 RT4 通告的成本
Lo、Ib	30	27	N9 - N11	29	26
N6	16	15	RT5	14	8
N7	30	19	RT7	20	14
N8	18	18			

除了区域间路由信息外，区域边界路由器 RT3 和 RT4 已接收到来自 AS 边界路由器 RT5 和 RT7 的 AS 外部路由信息。有两种类型的外部路由成本。类型 1 外部成本与区域内路由成本兼容，到外部网络的成本是内部成本和外部成本的总和。类型 2 外部成本比内部成本大一个数量级，因此到外部网络的成本由外部成本单独确定。当 RT3 或 RT4 得从 RT5 和 RT7 学习到的成本广播给区域 1 时，区域 1 的内部路由器成本（如 RT1）将设置通告的外部成本来为构建到外部网络的路由。最后，表 4-11 显示了带有区域间、区域内和外部路由的部分 RT4 路由表。

表 4-11 RT4 的路由表

目的地	路径类型	成本	下一跳	目的地	路径类型	成本	下一跳
N1	区域内	4	RT1	N8	区域间	25	RT5
N2	区域内	4	RT2	N9 - N11	区域间	36	RT5
N3	区域内	1	直接	N12	类型 1 外部	16	RT5
N4	区域内	3	RT3	N13	类型 1 外部	16	RT5
N6	区域间	15	RT5	N14	类型 1 外部	16	RT5
N7	区域间	19	RT5	N15	类型 1 外部	23	RT5

OSPF 分组格式

OSPF 消息有五种类型，所有类型都以如图 4-64 所示的相同头部。类型字段显示了消息的类型，表 4-12 显示了五种 OSPF 消息类型。类型字段后面是 IP 地址和源路由器的区域标识符。除了认证数据外，全部消息都被一个 16 位的校验和保护，这样就可以应用各种类型的认证机制。认证类型指示使用的机制。



图 4-64 OSPF 的报头格式

表 4-12 五种类型的 OSPF 信息

类 型	说 明	类 型	说 明	类 型	说 明
1	你好	3	链路状态请求	5	链路状态确认
2	数据库描述	4	链路状态更新		

除了 HeHo（问候）消息外，其他类型的 OSPF 消息用于请求、发送和回复链路状态信息。一个 OSPF 信息可能包含一个或多个链路状态通告（LSA）消息。每个 LSA 可以描述链路或路由器的成本信

息。这里主有五种类型的 LSA 消息，如表 4-13 所示，而且所有 LSA 类型共享相同的头部，如图 4-65 所示。每种类型的 LSA 由不同的路由器使用以描述不同的路由信息。比如，基于 AS 边界路由器的 AS 外部 LSA 用于描述到达在其他自治系统中的目的路由。

表 4-13 五种类型的 LSA

LS 类型	LS 名字	来 自	泛洪范围	说 明
1	路由器 LSA	具有活跃接口	区域	描述了路由器与区域接口的一组状态
2	网络 LSA	指定路由器	区域	包含了连接到网络的路由器列表
3	汇总 LSA（IP 网络）	区域边界路由器	相关区域	描述区域间网络的路由
4	汇总 LSA（ASBR）	区域边界路由器	相关区域	描述到 AS 边界路由器的路由
5	AS 外部 LSA	AS 边界路由器	AS	描述到其他 AS 的路由



图 4-65 LSA 头部格式

4.6.3 域间路由

域间路由的任务就是在互联网的自治系统之间实现连通性。与管域内路由是在同一管理控制下的 AS 中发生的，但由于大量的 AS 和 AS 间的复杂关系，域间路由很难实现。域间路由最明显的特征是，为达到比资源列表更重要。由于每个 AS 可能运行不同的路由协议并且根据不同的标准分配链路成本，在源和目的地对之间查找最小成本路径可能毫无意义。比如，在运行 RIP 的 AS 中成本 15 可能是大的成本，但在另一个运行 OSPF 的 AS 中它被认为是相当小的成本。因此，不同 AS 的链路成本可能不兼容，因此不可相加（正是由于同样的原因，OSPF 有两种类型的外部成本，类型 1 和类型 2）。另一方面，在域间路由中找到一条到达目的网络的无环路路径是必要的。AS 间复杂的关系使得这项查找任务非同寻常。例如，考虑一所大学，它拥有一个 AS 号，运行 BGP 连接到 AS 号分别为 X 和 Y 的两个网络服务供应商（ISP）。假设，这所大学从 AS 号为 X 的 ISP 购买了更多的带宽。而且，该大学当然不希望从 AS X 到 AS Y（或者从 AS Y 到 AS X）的转移流量通过它的域。因此，它可以建立一个策略："将所有流量路由到 AS X，除非它产生故障；否则，将流量路由到 AS Y"，并且"不承载从 AS X 到 AS Y 的流量（或从 AS Y 到 AS X 的流量）。这种路由称之为策略路由，这里策略允许路由域的管理员设置如何将分组路由到目的地的规则。这种策略可以指定首选的 AS 或不转运 AS。策略路由也解决安全和信任问题。比如，我们可能有一个策略声明表示发到 AS 的流量不能路由经过某些域，或者带有前缀 p 的分组发路由经过 AS X，如果前缀 p 是从 AS X 可达的。总之，在域间路由中，可扩展性和稳定性比最优更重要。

开源实现 4.10：OSPF

概述

OSPF 源代码中最有趣的部分是实现图 4-45 所示的 Dijkstra 算法实现。Dijkstra 算法是在 ospf_spf_calculate()（在 ospf_spf.c 中定义）中实现的。当特定的定时器到期（由 ospf_spf_calculate_schedule()预定的）时，由 ospf_spf_calculate_timer()调用以使计算到各个区域的最短路径。

数据结构

相关的数据结构包括定义在 ospfd/ospf_spf.h 和 lib/vertex.h 中的 vertex、route_table 和

route_node。跨越一个区域的最短路径所的路由交替 area -> spf 指向，并且树中的每个节点由 vertex 结构来描述：

```
    struct vertex
      {
        u_char flags;
        u_char type; /* router vertex or network vertex */
        struct in_addr id; /* network prefix */
        struct lsa_header *lsa;
        u_int32_t distance;
        list child; /* list of child nodes */
        list nexthop; /* next hop information for routing
  table */
      };
```

算法实现

当接收到各种类型的 LSA（网络 LSA、路由器 LSA、汇总 LSA）或者更改了监控链路。区域边界路由器的状态时，就调度运行 ospf_spf_calculate()。图 4-66 显示 Zebra 的 ospfd 调用图。

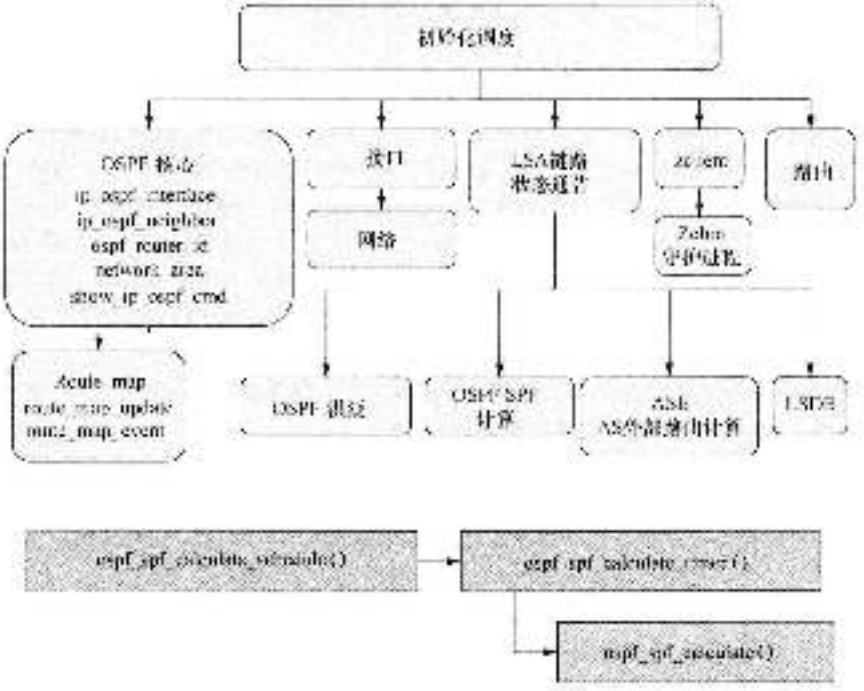


图 4-66 OSPF 的 Zebra 实现

图 4-45 中的 while 循环由 ospf_spf_calculate() 中的 for 循环实现。不包括在 T（即图 4-45 中的 V-T）中的节点（候选者）列表首先通过 ospf_spf_next() 来获得到。具有最小成本的节点是从候选者列表的头部中获得。调用 ospf_vertex_add_parent() 建立 F—路信息（即图 4-45 中的 c(v) = w），然后由 ospf_spf_register() 将节点添加到 SPF 树上。更新节点成本的操作（C(v) = MIN(C(v),C(w) + c(w,v)）也是在 ospf_spf_next() 中通过下面的语句执行：

```
w->distance = v->distance + nbha (1->m[0].metric);
```

练习

追踪 Zebra 的源代码，并解释如何维护每个区域的最短路径树。

性能问题：路由守护进程的计算开销

图 4-67 比较了 RIP 和 OSPF 中路由守护进程核心函数的执行时间，分别对应于 rip_rte_process() 和 ospf_spf_calculate()。RIP 即使是在具有 1500 台路由器的网络环境下也能够很好地扩展。然而，在一个具有 250 台路由器的网络中，OSPF 的执行时间超过 10 ms；而在一个

具有1500台路由器的网络中，这个时间会超过100 ms。路由算法的计算复杂性是执行时间中的关键因素。RIP采用了Bellman-Ford算法，这个算法的时间复杂性比OSPF采用的Dijkstra算法的时间复杂性小。

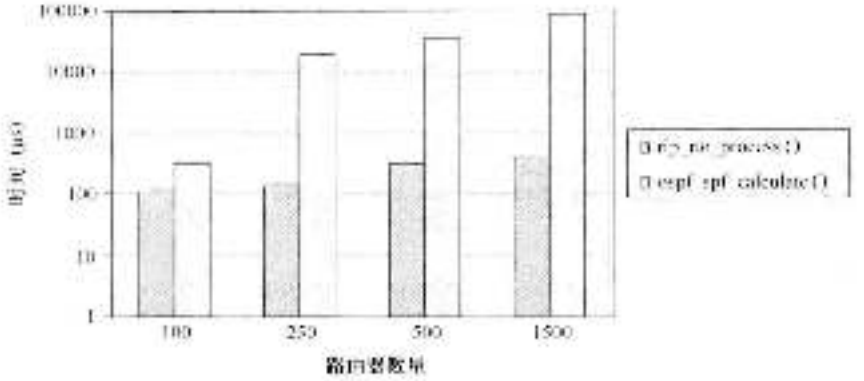


图4-67 RIP和OSPF的执行时间

BGP

边界网关协议版本4（BGP-4）是目前域间路由的事实标准。BGP-4最新RFC是RFC 4271。因为管辖许多企业和园区AS的人和ISP件，网上很可能有不止一台边界路由器与其他AS相连接，所以有两种类型的BGP：内部BGP（IBGP）和外部BGP（EBGP）。为了在同一个AS中的两台BGP路由器之间的通信，建立IBGP会话，而为了在不同的AS中的两台BGP路由器之间的通信，建立EBGP。IBGP的目的是，如果有很多路由器在同一个AS中运行BGP时，应确保路由器之间路由信息的一致。在AS中至少一台路由器选为AS的代表（代理），称为BGP发言者。BGP发言者使用EBGP会话与在其他AS中的BGP发言者交换路由信息。而且，由于稳定性和可靠性对域内路由很重要，所以BGP在端口179上运行TCP，可以使用认证进一步保证TCP连接的安全。对于同一个AS中的路由器，根据TCP和底层IBGP会话构建逻辑全连接的网络。（而且，在内部BGP路由器之间，将其中一个指定为BGP发言者，代表AS发言）。最后，BGP也支持CIDR。

路径矢量路由

在互联网内有大量的AS路由器使得距离矢量算法比链路状态算法更适用于BGP。然而，由于可达性和无回路操作比路由最优化更重要，BGP采用路径矢量算法——距离矢量算法的变体，在两个网络之间查找路由路径。路径矢量算法也仅与其邻居交换路由信息。但是为了防止回路，当交换一条路由表项时，就通告完整路径信息。由于每个AS都有一个唯一的AS号（一个16位的标识符），一个完整的路径保留路径已经经过的AS号的有序序列。如果当前的AS号在路径中找到，那么就检测到一个环路。而且，由于在不同AS中成本定义的不一致性，路由交换信息不包括成本信息。因此，路由路径的选择大多根据可管理的优先级和路径上的AS号。

BGP分组有四种类型，包括OPEN、KEEPALIVE、UPDATE和NOTIFICATION。在两台BGP路由器之间建立一条TCP连接后，就将一个OPEN信息发送到通信的对方。之后，就定期发送KEEPALIVE信息以便确保对方保持激活。路由信息通过UPDATE信息进行交换。与RIP协议不同，由于表的尺寸过于庞大，BGP不会定期更新全部的路由表。UPDATE信息包括发送者想要撤销（withdraw）的一组路由和到一组目的地网路的路径（path）信息。UPDATE消息的格式如图4-68所示。将路径属性应用到在目的网络中列出的所有目的地（称为网络层可达信息，NLRI）。路径属性携带的信息可能包括路径信息（来自IGP、EGP或不完整）的来源、到目的地路径上的AS列表、到目的地的下一跳、用于多个AS出口点（MULTI_EXIT_DISC，MED）的鉴别符号、用来表示AS内路由器优先的本地优先（LOCAL_PREF）、汇总路由、以及汇总路由的AS标识符。最后，当遇到错误时就向对方发送NOTIFICATION消息。

每个BGP路由器保留所有到达目的地的可行路径，但仅将“最好”路径通告给邻居。选择“最好”的路由取决于AS策略。然而，在一般情况下，优先选择较大的LOCAL_PREF、较短的路径、较低的

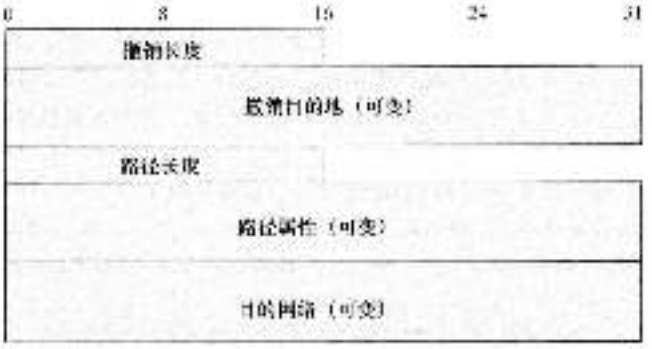


图 4-68 BGP UPDATE 消息的分组格式

起源编码号（lower origin code）（对于 EGP，优先选择 IGP）、较低的 MED、更靠近的 IGP 邻居，以及具有更低 IP 地址的 BGP 路由器。为目的地确定了“最佳”路由后[①]，然后 BGP 发言者（speaker）就将每个目的地的最高优先级经过 EBGP 通告给邻居 BGP 发言者。BGP 发言者也会将其学到的路由信息通过 IBGP 传播给 BGP 路由器（非 BGP 发言者）。

BGP 例子

最后，让我们来看看 BGP 路由表的一个例子。表 4-14 取自一所大学的边界路由器的部分 BGP 表（一台互联网骨干路由器的完整 BGP 表具有超过 30 万条路由项目。当前 BGP 表的大小参见 http://bgp.potaroo.net/）。大学的 AS 号为 17712。第一个路由表项指示 BGP 路由器已经接收到 UPDATE 消息，目的网络为 61.13.0.0/16，分别来自三个邻居 139.175.56.165、140.123.231.103、140.123.231.100。到 61.13.0.0/16 的最佳 AS 路径是通过 140.123.231.100（可能只是因为它是最短路径）。起源编码号指示路由器的邻居 140.123.231.100 通过 IGP 协议学习到 AS PATH。

表 4-14 一个 BGP 路由表的例子

网　络	下一跳	LOCAL_PREF	加权	最佳	路　径	起　源
61.13.0.0/16	139.175.56.165		0	N	4780，9739	IGP
	140.123.231.103		0	N	9918，4780，9739	IGP
	140.123.231.100	0	0	Y	9739	IGP
61.251.128.0/20	139.175.56.165		0	Y	478，9277，17577	IGP
	140.123.231.103		0	N	9918，4780，9277，17577	IGP
211.73.128.0/19	210.241.222.62		0	Y	9674	IGP
218.32.0.0/17	139.175.56.165		0	N	4780，9919	IGP
	140.123.231.103		0	N	9918.4780.9919	IGP
	140.123.231.106		0	Y	9919	IGP
218.32.128.0/17	139.175.56.165		0	N	4780，9919	IGP
	140.123.231.103		0	N	9918.4780.9919	IGP
	140.123.231.106		0	Y	9919	IGP

开源实现 4.11：BGP

概述

BGP 采用距离矢量路由，但为了避免循环，它的消息中包括路由路径信息。它强调策略路由而不是路由成本优化。因此，在其实现中，我们将看到它如何根据策略选择最低路径。

① 通常，它可能是一组目的网络。

数据结构

BGP路由表是一个bgp_table结构，其中包括BGP的节点（bgp_node结构）（参见bgpd/bgp_table.h）。每个bgp_node具有指向BGP路由信息的指针struct bgp_info，定义在bgpd/bg_route.h中。bgp_info由一个指向struct peer的指针组成，用来存储邻居路由器的信息。

算法实现

图4-69显示了用来处理BGP分组的bgpd调用图。当接收到BGP UPDATE分组时，就调用bgp_update()函数，使用路径属性attr作为其参数之一。bgp_update()然后调用bgp_process()处理有关路由信息的更新，依次调用bgp_info_cmp()，根据以下优先级规则比较两条路由的优先：

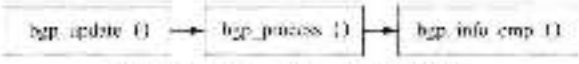


图4-69 Zebra中bgpd的调用图

0）Null（空路由）检查：优先选择非空路由。

1）加权检查：优先选择较大的加权。

2）本地优先检查：如果设置了本地优先，优先选择较大的本地优先。

3）本地路由检查：优先选择静态路由、重分配路由或汇总路由。

4）AS路径长度检查：优先选择较短的AS路径长度。

5）来源检查：按照下列顺序优先选择学习到的路由：IGP、EGP、不完整。

6）MED检查：优先选择较低的MED（MULTI_EXIT_DISC）。

7）对等类型检查：优先选择EBGP对等，然后选择IBGP对等。

8）IGP度量检查：优先选择更近的IGP。

9）成本社区检查：优先选择低成本。

10）最大的路径检查：没有执行。

11）如果两条路径是外部的，优先选择首先收到的路径（最老的一个）。

12）路由器标识符比较：优先选择更低的标识符。

13）集群长度比较：优先选择更低的长度。

14）邻居地址比较：优先选择较低的IP地址。

练习

请读者研究目前BGP路由表的前缀长度分布。首先，浏览http://thyme.apnic.net/current/，你会发现由APNIC路由器看到的BGP路由表的分析，结果很有趣。尤其是"每个前缀长度宣布的前缀数量"将让你知道骨干路由器路由项目数量和选择路由表项前缀长度的分布。

1. 在你访问URL的当天，骨干路由器有多少路由表项？

2. 绘制图形用对数显示前缀长度分布（长度从1～32），因为宣布的前缀号码从0到数万不等。

4.7 组播路由

到目前为止，我们了解了从单一源向单一目的地传送（从主机到主机分组）的完整互联网解决方案。然而，许多新兴的应用需要从一个或多个源将分组传送到一组目的地。例如，视频会议和流媒体、远程学习教育、WWW缓存更新、共享白板，并且网络游戏是流行的多方通信应用程序。将分组发送到多个接收者称为组播。组播会话包括一个或多个发送者，通常在同一个组播地址有多个发送或接收分组的接收者。

4.7.1 将复杂性迁移到路由器

可扩展性仍然是实现互联网组播服务的主要问题。在考虑可扩展性的同时，我们首先解决来自发送者、接收者、路由器等方面的问题。发送者可能面临以下问题：发送者如何将分组发送给一组接收者？发送者是否需要知道接收者在哪里以及他是谁？发送者是否能够控制组成员？多个发送者可以同时向一组接收者发送分组吗？保持发送者的工作尽可能简单就可以使将一个分组发送到组播组的任务

具有高度可扩展性，因此互联网组播提供的解决方案就是完从发送者去掉组播负担而将它留给互联网路由器。但是，这会将复杂性上移给核心网络。路由器，而远离边缘主机。它将使原来的核心网络从无状态变为有状态，这一点我们在后面将会看到，这会对基础设施有重要影响。因此，是否将组播留在 IP 层还是留给应用层仍然是一个有争议的问题。在本节结束时，我们会讨论这个问题。

如图 4.7 所示，D 类 IP 地址空间是为组播预留的。为组播组分配一个 D 类 IP 地址。要向组播组发送分组的发送者只需要将组的 D 类 IP 地址放到 IP 头部的目的地字段即可。发送者并不需要知道接收者在哪里以及分组如何发送给组成员。换句话说，发送者不负责维护组成员名单，并将接收者的 IP 地址放入 IP 头部中。可扩展性就是这样实现的，因为从发送者的角度，发送组播分组就像发送一个单播分组一样简单。多个发送者可以同时向一个组播组发送分组。缺点是发送者在互联网层不能控制组成员（但在应用程序却可以控制）。

从接收者的角度，人们会问以下问题：一个人如何才能加入组播组？人们如何才能知道互联网上正在进行的组播组？任何人都可以加入一个组吗？一个接收者可以动态地加入或离开一个组吗？接收者能够知道在该组中的其他接收者吗？另外，互联网解决方案就是使接收组播分组的任务就像接收单播分组一样简单。接收者向最近的路由器发送一条加入（join）消息表明它希望加入的组播组（一个 D 类 IP 地址）。然后接收者就可以像接收单播分组一样简单地接收组播分组。接收者可以随时加入和离开一个组播组。除了为组手动配置一组 D 类 IP 地址外，没有特别的机制。然而，有些用于在互联网上通告组播会话地址的协议和工具。此外，IP 层不提供了解组播组内所有接收者的机制。将该工作留给应用层协议来完成。

最后，路由器可能会问如何发送组播分组。路由器是否需要知道组播组中的所有发送者和接收者？因为组播发送者和接收者都是组播的负担，所以路由器就要担负起这项工作。组播路由器有两个任务：组成员管理和组播分组分发。首先，路由器需要知道在各直接连接的子网内的主机是否已加入一个组播组。用于管理组播组成员信息的协议称为互联网组管理协议（IGMP）。其次，路由器需要知道如何将组播分组发给所有成员。有人可能会想到建立许多一对一连接发送组播分组。然而，这肯定不是一种有效的方法，因为这样互联网将充满重复的分组。一种更有效的方法是建立一个组播树，根在每个发送者或者被整个组所共享。然后在组播树上发送组播分组，只在树枝上复制组播分组。建立组播树的任务由组播路由协议（如 DVMRP、MOSPF、PIM）来完成。

现在应该很清楚 IP 层组播的解决办法就是要使发送者和接收者尽量简单而将负担留给路由器。下面，我们将重点放在路由器的任务上。具体来讲，我们首先学习组成员管理协议，它运行在主机与 IP 子网指定的路由器之间。它允许指定的路由器知道是否至少有一台主机加入到一个特定的组播组中。其次，我们讨论组播路由协议。组播路由协议运行在支持组播的路由器之间，用于为每个组播组建立组播树。最后，由于大多数组播路由协议用于域内组播，所以我们应该为域间组播路由引入一些新的开发设计。

4.7.2 组成员管理

负责将组播分组发送到它直接连接的 IP 子网的路由器称为指定路由器。指定路由器需要维护子网中所有主机的组成员信息，以便它能够知道发往一个特定组播组的分组是否应该转发到子网。互联网使用的组成员管理协议称为互联网组管理协议（IGMP）。

互联网组管理协议

当前 IGMP 的版本为 IGMPv3，定义在 RFC 3376 中。IGMP 允许路由器在其直接连接的子网中查询主机，看它们中是否已加入到一个特定的组播组。它也允许一台主机对查询做出响应报告或者通知路由器主机将离开组播组。

基本上，有三种 IGMP 消息类型：查询、报告、离开。IGMP 分组格式如图 4.70 所示。查询消息有一个类型值为 0x11。查询消息可能是通用查询也可能是特定组查询。当它是一个通用查询消息时，组播组的地址就用 0 来填充。一个 IGMPv3 成员报告消息具有类型值为 0x22。为了能够向后兼容性，IGMPv1 成员报告、IGMPv2 成员报告和 IGMPv2 离开组消息分别使用类型 0x12、0x16 和 0x17。IGMP 消

息是在 IP 分组中携带协议标识符 2，并且发送到所有系统组播地址和所有路由器组播地址等的特定多播地址。

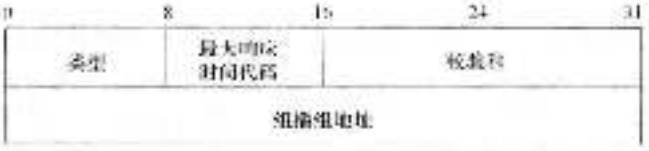


图 4-70 ICMP 分组格式

让我们简要地看看 IGMP 的运行。组播路由器扮演两个角色之一：查询者或非查询者。查询者负责维护成员信息。如果在一个 IP 子网中有多台路由器，那么具有最小 IP 地址的路由器就称为查询者，其他路由器为非查询者。路由器通过听取其他路由器发送的查询消息来确定自己的角色。查询者周期发送成员查询消息来征求成员信息。将这些查询消息发送到 224.0.0.1（ALL SYSTEMS 组播组）。

至少有一个成员或者没有成员

当主机接收到一个通用查询消息时，它会等待一个介于 0 与最大响应时间的一段短暂时间。这是在通用查询消息中给定的。然后，当定时器过期时，这台主机就发送一个 TTL = 1 的报告消息。然而，如果这台主机看见同一组播组的其他主机发送的报告信息。主机就停止定时器并删除这个报告信息。采用随机定时器是为了抑制这一步来自其他组成员的报告消息，因为路由器仅关心是否至少有一台主机加入组播组。当主机接收到一个特定组查询消息时，如果主机是查询消息指定的组播组成员，那么就采取相似的动作。

当路由器接收到一个报告消息时，它就将消息中报告的组添加到其数据库组播组列表中。它还将成员定时器设置为“组成员时间间隔”，并且在定时器过期之前如果没有收到报告就删除成员表项（查询消息是周期性地发送的，因此路由器期望在定时器过期前看见返回的报告）。除了响应查询消息外，当主机想要加入一个组播组时，它可以立即发送一个称为请求报告。

当一台主机离开一个组播组时，它应该给所有路由器组播地址（224.0.0.2）发送一个离开组消息。如果它是最后一台应答组查询消息的主机。当查询路由器接收到一个离开消息时，对于每一个“最后成员查询时间间隔”，它给连接子网上的相关组发送特定组查询“最后成员查询计数”次数。如果在“最后成员查询时间间隔”过期前还没有收到报告，那么这个路由器将假设相关的组没有本地成员，没有必要将那一组的组播转发给所连接的子网。通过这种假设，当一台主机离开这个组时，路由器就不需要计算有多少台主机加入相关组的成员。它只是简单地问：“还有主机在这个组中吗？”

通过对 IGMP 操作的概述，我们可以看出，没有对谁能够加入组播组或者谁能够向组播组发送分组进行控制。这里也没有一种用来了解组播组中接收者的 IP 层机制。IGMPv3 添加了对“源过滤”的支持。也就是说，接收者可能请求仅接收来自特定源地址的分组。接收者通过激活 `IPMulticastListen(socket,interface,multicast-address,filter-mode,source-list)` 函数加入到组播组中。其中 `filter-mode` 既可以是 `INCLUDE` 也可以是 `EXCLUDE`。如果这个 `filter-mode` 是 `INCLUDE`，那么接收者仅希望接收 `source-list` 中的发送者。另一方面，如果 `filter-mode` 是 `EXCLUDE`，那么就不能接收来自 `source-list` 中的发送者。

4.7.3 组播路由协议

组播的第二个组件是组播路由协议。它为组播分组分发构建组播树。那么组播树应该是什么样的呢？从发送者的角度，它应该是一个以发送者为根能够到达所有接收者的汇合树。但是，如果有多个发送者时，组播组将会发生什么？在互联网中，采用两种方法构建组播树。它们的区别在于所有的发送者是否使用单个树发送分组，或者是否每个发送者都有一个特定源的组播树用以发送分组。这两种方法的扩展性如何？第一种方法，如果奏效，更具有可扩展性，因为组播路由器只维护单组的状态信息。而第二种方法，基于源的方法，需要每个发送者的状态信息。然而，基于源的方法最终会了更短路径，因为分组计名找来途历。构建基于源的树的组播路由协议包括距离矢量组播路由协议（DVMRP），

OSPF 的组播扩展（MOSPF）和协议独立组播协议密集模式（PIM-DM）。另一方面，PIM 稀疏模式（PIM-SM）和基于核心树（CBT）构建组共享树。一个成员稀疏地分布到网络拓扑上的稀疏组内，可扩展性似乎是共享树方法最好的，这一点以后会更加清楚。

Steiner 树与最小成本路径树

在我们详细描述组播路由协议之前，让我们研究构建组播树涉及的两个问题。我们已讨论了最佳点到点路由。什么是最佳组播路由？在文献中，将组播问题建模为一个图论问题，其中每条链路分配一个成本。最佳组播路由包括找到一个最小成本的组播树，组播树成本是树上所有链路成本的总和。当然，组播树必须以源为根并包括所有接收者。最优组播树，或者具有最低总成本的树，称为 Steiner 树。不幸的是，查找 Steiner 树的问题已知是 NP 完全问题，即使所有的链路具有单位成本。因此，大多数以往的研究人员都将重点放在采用多项式时间并产生近似最优结果的启发算法设计上。而且，这些启发式算法经常能够保证他们的解决方案处于最优解成本两倍之内。然而，尽管启发式算法显示出好的性能，但暂时还没有尝试解决 Steiner 树问题的互联网组播路由协议。为什么呢？有三个明显的原因使这些启发式算法不切实际。首先，大多数算法是集中式的并需要全局信息——也就是说，需要网络中所有链路和网络节点的信息。但是，集中式的解决方案并不适合分布式互联网环境。其次，Steiner 树问题是针对静态成员的组播形成的，源节点和所有接收者是固定并且预先知道的。互联网当然并非如此。最后，对于在线计算，大多数启发式算法的计算复杂性是不可接受的。毕竟，最小化组播树成本并没有可扩展性重要。此外，没有明确定义的链路成本，我们如何解释组播树的成本以及它对最小化成本有多重要？

构建组播树的另一个问题是组播路由协议是否依赖于某些特定的单播路由协议。不是解决 Steiner 树问题，最新的互联网组播路由协议是根据最小成本路径算法来构建组播树。对于基于源的树，从源到每个目的地的路径是通过单播路由找到的最小成本路径。结合从源到每个接收者的成本最小的路径，就形成了一个以源作为根的最小成本路径树。对于组共享树，最小成本路径树是从一个中心节点（称为一个汇聚点或核心）到所有接收者形成的。此外，使用最小成本路径从源向中心节点发送分组。由于这两种类型的树都是根据最小成本路径构建的，所以单播路由的结果当然能够加以利用。然后问题就在于组播路由协议是否需要某种特定单播路由协议的合作，或者它是否独立于底层单播路由协议。对于当前的互联网解决方案，DVMRP 是前者的一个例子；而 PIM，顾名思义，是独立单播路由协议。下面，我们介绍其中最常用的组播路由协议：DVMRP 和 PIM。

行动原则：当 Steiner 树不同于最小成本路径树时

图 4-71 中显示了一个简单的例子，这里最小成本路径不是 Steiner 树。在这个例子中，*A* 是一个源节点，*C* 和 *D* 是 2 个接收器。从 *A* 到 *C* 的最小成本路径是从 *A* 到 *C* 成本为 3 的直接链路。从 *A* 到 *D* 的最小成本路径与此相同。因此，最小成本路径树是以 *A* 为根扩展到 *C* 和 *D* 的树，成本为 6。然而，最优解决方案 Steiner 树，是以 *A* 为根，首先连接到 *B*，然后扩展到 *C* 和 *D*。Steiner 树的成本为 5，低于最小成本路径树。

距离矢量组播路由协议

DVMRP 在 RFC 1075 中提出，是互联网中第一个也是最广泛使用的组播路由协议。DVMRP 将 RIP 作为其内置的单播路由协议。当互联网发起组播时，DVMRP 是运行在实验骨干网（称为 MBone）的组播路由协议。DVMRP 为每个组播发送者构造一个基于源的树。组播树通过两步构建。第一步，利用反向路径广播（RPB）向所有路由器广播组播分组。然后利用修剪信息将 RPB 树修剪成反向路径组播树（RPM）。

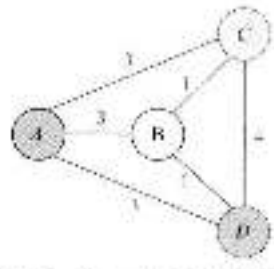


图 4-71 Steiner 树不同于最小成本树的例子

反向路径广播

传统上，在网状网络中的广播是通过洪泛实现的，即将广播分组转发到所有外出链路上，除了接收到分组的接口外。然而，由于洪泛使路由器多次接收到相同的分组。如何才能避免路由器将同一分组转发多次？RPB 是一个非常好的思

路，参见图4-72。当路由器接收到分组时，只有当分组到达从路由器回到发送者的最短（最小成本）路径上的链路上时才会洪泛（转发）分组。否则，就直接丢弃分组。广播分组仅保证被路由器洪泛一次。当所有路由器都洪泛一次后，洪泛过程就停止。路由器仍然可能接收到相同的分组一次以上，但不存在循环或无限洪泛的问题。

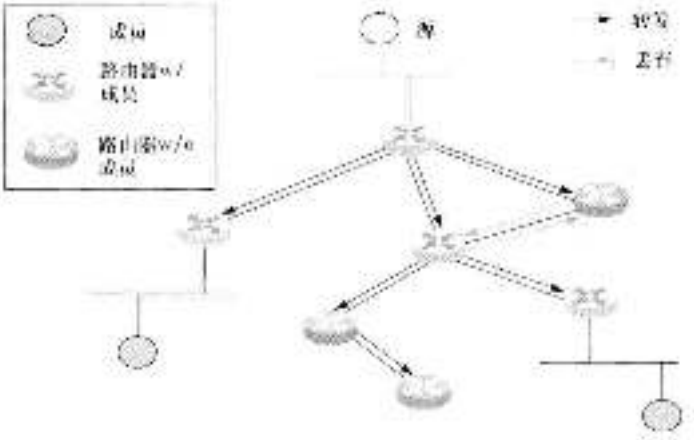


图4-72 反向路径广播

显然，RPB要求每台路由器构建自己的单源路由表。也就是说，DVMRP需要底层单播路由算法。RPB称为"反向路径"，因为尽管最短路径树应该以发送源为根并明确接收者，但每台路由器根据"反向最短路径"（即从路由器到发送源）的信息决定是否洪泛分组。因此，分组经过从发送者到发送者之间的最短路径到达目的地。为什么不只使用转发最短路径呢？距离矢量算法寻找从路由器到目的地的下一跳。因此，接收了广播分组的路由器不知道从发送者到达它自己的最短路径，但知道从它本身到发送者的最短路径。

反向路径组播

当激将组播分组广播到所有路由器（和子网）时，不想收到这个分组的许多路由器和子网不可避免地接收到了它。为了克服这个问题，不到任何接收者的一台路由器将在其上游路由器发送修剪，如图4-73所示（路由器通过IGMP知道其成员信息）。中间路由器为每个组播组维护一张依赖的下游路由器列表。当一台中间路由器接收到一个修剪消息时，它就进行检查以验证不存在的已经加入到多播组的下游路由器成员，以就此说，所有下游路由器都已经向它发送过修剪消息。如果是，它就发送另一个修剪消息到上游路由器。这样就不会将数据分组再发送到已经从树上修剪掉的路由器上。如图4-74所示，修剪过的RPB树就形成反向路径组播（RPM）树。

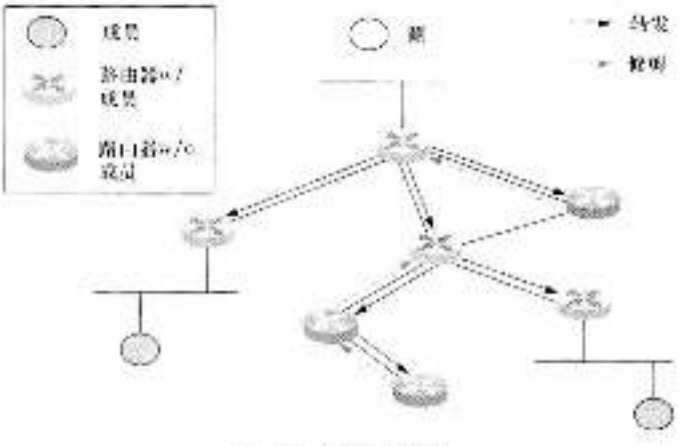


图4-73 修剪RPB树

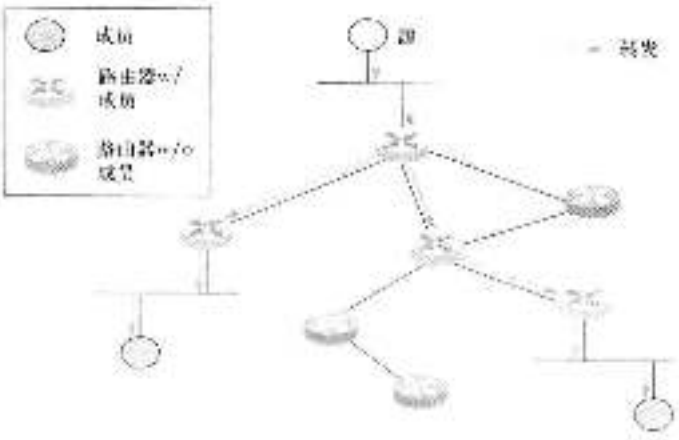


图 4-74 图 4-73 修剪过的 RPM 树

接下来的问题是，如果在修剪树上的一台主机想要加入多播组将会发生什么。有两种可能的解决方案。首先，修剪信息将包含修剪寿命，它指定一条修剪过的树枝保持被修剪掉多长时间。因此，修剪寿命过期后，修剪掉的树枝将重新添加到树中。换句话说，当修剪寿命过期时，多播分组将周期性地洪泛。另一方面，路由器也可以显式地发送一个修接消息给它的上游路由器强制将修剪掉的树枝重新添加到组播树中。

DVMRP 有几个缺点。例如，首先，几乎没有组播分组需要洪泛到所有路由器上。这使得它仅对具有密集成员的组能够很好地工作。其次，修剪消息的寿命特点也需要路由器定期刷新其修剪状态。最后，由于 DVMRP 构建基于源的树，所以每台路由器需要维护每个源、每个组的状态信息。对于具有两个发送者的组播组，中间路由器需要为该组维护两个状态，因为不同发送者的组播树也不同。也就是说，如何发送分组取决于发送者是谁。因此，DVMRP 需要在每台路由器上存储大量的状态信息。总之，尽管 DVMRP 不能很好地扩展，但因为 DVMRP 的简洁性，它仍然是最广泛使用的协议。

协议独立组播协议

正如我们已经了解到的，对于稀疏分布成员的组播组，DVRMP 不能很好地扩展。原因有两个：首先，面向源的树的构建，RPB 树修剪成 RPM 树的方式是不可扩展的；其次，构建一个基于源的组播树是不可扩展的，因为它需要太多的状态信息。随着路径变得更长以及组数和每组发送者数量的增长，状态开销也迅速增长。对于稀疏分布的组成员，带有面向接收者树的共享树构建将更具可扩展性，因此为互联网建议了一种新的组播路由协议，称为协议独立组播协议（或协议无关组播，PIM）。通过使用两种模式，PIM 显式地支持两种构造组播树的方法。PIM 密集模式（PIM-DM）以一种类似于 DVMRP的方式构造一种基于源的组播树，适用于密集分布成员的组播组。另一方面，PIM 稀疏模式（PIM-SM）为每个组播组仅构造一个组共享树，因此，适用于成员广泛分散的组。既然 PIM-DM 非常类似于 DVMRP，我们仅在本节讨论 PIM-SM。PIM-SM 的一个最新版本协议在 RFC 4601 中描述。我们还注意到，组播的可扩展性问题来自大量的组播组数，既不能由 DVMRP 也不能由 RIM 来解决。

PIM-SM 遵守如下的设计原则。如果一台路由器不导向任何接收者，那么它就不应该参与到组播会话的多播路由中。因此，在 PIM-SM 中，树是按照接收者驱动的方式构建的；也就是说，导向接收器所在子网的一台路由器需要显式地发送加入消息。共享树的中心节点称为一个汇聚点（RP）。每个组播组 RP 唯一地由散列函数确定，这一点我们将在稍后讨论。共享树因此称为基于 RP 的树（RPT）。路由器负责转发组播分组，发送加入消息给称为指定路由器（RP）的子网。路由表称为组播路由信息库（MRIB），DR 用它来确定发送任何加入/裁剪消息的下一跳邻居。MRIB 既可以直接取自单播路由表也可以由一个单独的路由协议导出。让我们看看以面向接收器的方式如何构建 RPT PIM-SM 的三个阶段。

第一阶段：RP 树

在第一阶段，如图 4-75 所示构建一个 RP 树。我们分别从两个方面描述构建过程：接收者和发送

者。当接收者想加入组播组时，它使用IGMP向DR发送一个加入消息。一旦接收到加入消息，DR就向RP发送一个通用组加入消息。通用组加入消息由（*，G）表示，它表示接收者想要从所有源接收组播分组。因为PIM加入消息沿着从DR到RP的最短路径发送给RP，所以它最终可能到达RP（例如，在图4-75中A的加入消息），或也可能到达一台已经在RPT中的路由器（例如，来自图4-75中B的消息）。在这两种情况下，在RPT上的路由器将知道DR想要加入组播组并沿着从RP到DR之间最短路径的反向转发组播分组。PIM-SM的一个特定功能，就是不会将响应中的无确认消息发回给DR。因此，DR需要定期发送加入消息以便维护RPT；否则，时间到期后，它将被修剪掉。

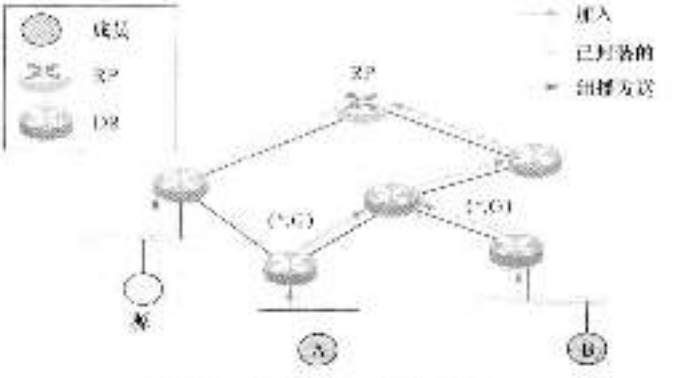


图4-75 PIM-SM第一阶段的操作

另一方面，想要发送组播分组的发送者可以将分组发送到组播组地址。一旦接收到组播分组，发送者的DR将它封装到PIM注册分组中，然后再将它转发到RP。当RP接收到PIM注册分组时，它就解封装并转发到RPT。你可能想要知道为什么发送者的DR需要封装组播分组。记住，任何主机都可以成为一个发送者。那么RP如何知道潜在的发送者在哪里？即使PR知道发送者在哪里，RP如何从发送者那里接收分组？在第一阶段，在发送者DR的帮助下，RP接收来自发送者的组播分组，因为发送者DR能够识别组播分组并且知道用于组播组的RP所在的位置。

第二阶段：注册停止

尽管封装机制允许RP从发送者的DR接收组播分组，但封装和解封装的操作太昂贵了。因此，在第二阶段（参见图4-76），RP更愿意直接从发送者接收组播分组而不经过封装。为了实现上述功能，RP发起一个PIM特定源加入消息给发送者。一个特定源加入消息由（S，G）来表示，这表明接收者想要仅从特定的源S接收组播分组。当特定源加入消息通过从RP到源的最短路径传播时，路径上的所有路由器都在它们的组播状态信息中记录加入消息。加入消息到达源的DR后，组播分组就开始沿着特定源树（SPT），（S，G）树，流到RP。因此，RP可能收到重复的分组，其中一个分组就是原始的组播格式而另一个则是封装过的。RP丢弃封装过的分组并向发送者的DR发送一个PIM注册停止。

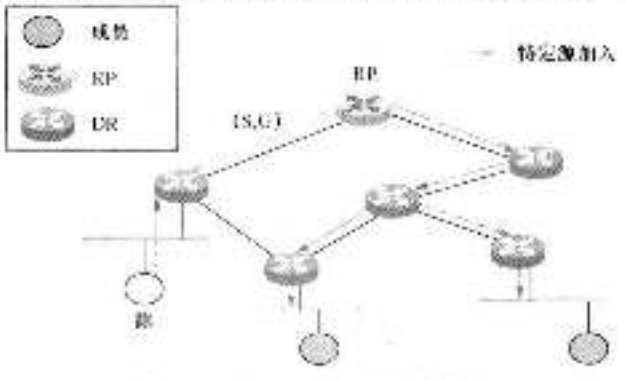


图4-76 PIM-SM第二阶段的操作

消息。之后，发送者的 DR 就不会再封装和转发组播分组到 RP 了，因此 RP 就可以直接接收来自发送者的组播分组。

该阶段的一个有趣场景是：若一路由器与 RPT 位于同一个特定源树时会发生什么？显然，该可能通过将从特定源树接收到的组播分组直接发送到 RPT 的下游路由器而走捷径。

第三阶段：最短路径树

共享树发送多播分组的一个缺点是，从发送者到 RP，然后再从 RP 到接收者之间的路径可能会很长。PIM-SM 的一个新特点是，允许接收者的 DR 选择性地启动从一个 RPT 切换到一个特定源树上。图 4-77 显示了从一个 RPT 切换到 SPT 的过程。接收者首先发起一个特定源的加入消息（S，G）到源 S。加入消息最可能到达源 S 可能仅经到 SPT 的某一路由器。然后 DR 开始从两棵树接收两个组播分组的副本。它将丢弃从 RPT 收到的副本。然后 DR 发送一个特定源修剪消息（S，G）到 RP。修剪消息既可以到达 RP 也可以收敛到 RPT 的某个路由器。DR 将不再接收来自 RPT 的分组。注意，修剪消息是一个特定源消息，因为 DR 仍然想经过 RPT 接收来自其他发送者的分组。

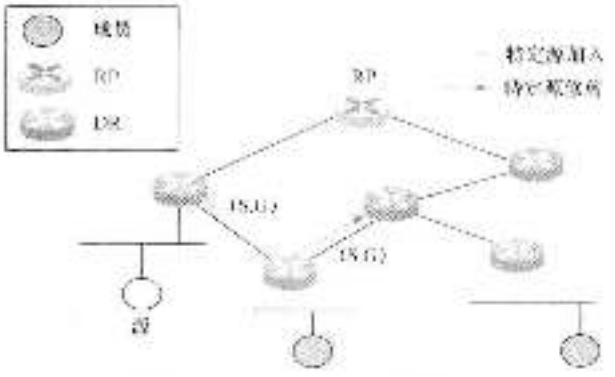
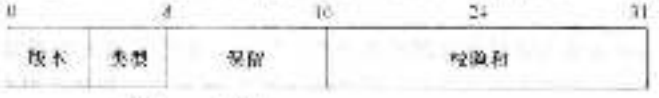


图 4-77 PIM-SM 第三阶段的操作

PIM-SM 也可以与 IGMPv3 的某些新功能配合，特别是特定源加入功能。如果接收者使用 IGMPv3 发送一个特定源加入，那么接收器的 DR 可能会直接进行一次普通的组加入（*，G）。相反，它应该发出一个特定源（S，G）加入消息。为特定源组播保留的组播地址范围为 232.0.0.0～232.255.255.255。此外，定义在 RFC 4607 的特定源组播（SSM）引入了一种新的一对多组播模型。它描述了一个源地址和一组地址运行组播是如何实现的，特别适合于传播式的应用，可以使用 PIM-SM 来实现。

PIM 分组格式

图 4-78 显示了 PIM 分组的头部。第一个字段描述其 PIM 版本，当前 PIM 版本是 2。第二个字段是类型字段，有 9 种 PIM 分组类型，如图 4-78 所示。第三个字段是保留给未来使用。最后一个字段是 PIM 分组的校验和，它是整个 PIM 分组的 1 的补码总和的 16 位 1 的补码。



类型	描述
0	Hello
1	Register
2	Register-Stop
3	Join/Prune
4	Bootstrap
5	Assert
6	Graft（用于 PIM-SM）
7	Graft（用于 PIM-DM）
8	Candidate-RP-Advertisement

图 4-78 PIM 分组格式

4.7.4 域间组播

与一组共享一个 RP 的思想使得 PIM-SM 与域的自治性质相左，因此难以应用于域间组播的目的。例如，如果一个发送者和同一域内的一群接收者形成一个组播组，但组的 RP 位于另一个域中，那么所有的分组在它们能够被那些接收器接收到之前它们就需要先到其他域中的 RP。因此，通常不使用 PIM-SM 穿越域。每个组在每个域内都将有一个 RP。

如果 PIM-SM 是用在单个域中，那么每个 RP 就知道所有的源和在其管理下的所有组的接收者。然而，它没有机制知道它域外的源。组播源发现协议（MSDP）建议用于 RP 以便学习远程域中的组播源。具体来讲，每个域中的 RP 会与远程域中 RP 建立一种 MSDP 对等关系。当 RP 获悉它自己域内的一个新的组播源时，它就使用源活跃（Source Active，SA）消息通知它的 MSDP 对等。具体地讲，RP 将从源接收到的第一个数据分组对装到一条源活跃消息中，然后将 SA 发送到所有对等，如图 4-79 所示。如果接收 RP 有一个 SA 组的（S，G）表项，RP 就向源 RP 发送一条（S，G）加入消息以便分组可以转发到 RP。RP 解开装数据并将它沿着共享树转发，如果在其域中有接收者，可以通过发送一条特定源（S，G）加入消息从源建立一条更短的路径。每个 RP 也定期发送 SA，包括在其域中的所有源，到其对等。RFC 3446 提出选播 RP 协议以便为 MSDP 应用提供 PIM-SM 域中的容错和负载共享。

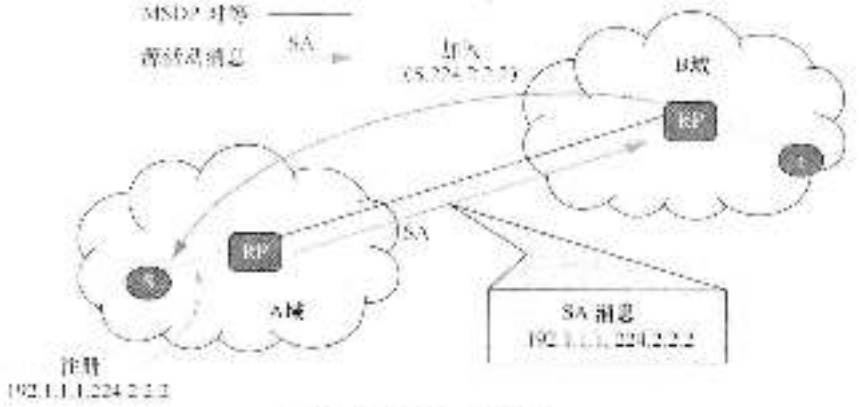


图 4-79 MSDP 的操作流

另一方面，多协议扩展到边界网关协议（MBGP）定义在 RFC 2858 中，它也允许路由器之间交换路由信息。因此，如果采用 MBGP 提供 MRIB，PIM-SM 的 DR 也将会有域间路由。

行动原则：IP 组播或应用组播

在目前的互联网中，由于多方面的考虑 IP 组播仍然不能广泛部署。支持 IP 组播的路由器必须维护所有的活跃（活动）组播会话的状态，因此随着这些会话数的增多就可能成为系统瓶颈，导致差的可扩展性。此外，传输层功能支持组播仍然广受争议。例如，不存在满足所有组播应用可靠性和拥塞控制要求的最优解。此外，很少有互联网服务提供商（ISP）能够支持组播是因为缺乏诊断的付费结算机制，使得组播难以广泛地部署。

有些研究人员已经提出应用层组播（ALM）的概念以便解决这些问题。ALM 的基本思想是，组播服务是由应用层提供而非网络层。用户空间部署配置使 ALM 与当前的 IP 网络兼容。也就是说，路由器和互联网服务供应商不需要更改或特殊的支持。此外，ALM 在定制特定应用方面允许更灵活的控制，使得传输层功能易于部署。ALM 会话的参与者构成参与者之间单播连接的覆盖。参与者既可以是专用机也可以是终端主机。一个基于基础设施的 ALM 方法是指一个由专用机构成的覆盖方法，而基于对等的 ALM 方法的覆盖是由终端主机形成的。最近，ALM 已经成为对等网络模型的一种特殊应用，这一点我们将在第 6 章中进一步说明。

开源实现 4.12：mrouted

概述

我们将要学习的组播路由的开源实现是 mrouted，它实现 DVMRP 协议。

数据结构

在 mrouted 中，组播路由表存储为路由表项的一个双链表，由结构“rtentry”（在 mrouted/route.h 中）表示。对于每个子网，如果有组播功能，那么就有一条路由表项。子网中的活跃组播组列表，简称组表，由 rt_groups 指针指向，如图 4-80 所示。组表由 2 个双链表组表项组成，它们又由结构 gtable（定义在 mrouted/prune.h）表示。第一个链表是同一个源的活跃组通过组地址排序后的链表，在路由表项中由指针 gt_next 和 gt_prev 所指。第二个链表（由 gt_gprev、gt_gnext 指标起来）是一个所有源和组中活跃组的列表并且由 kernel_table 指向。

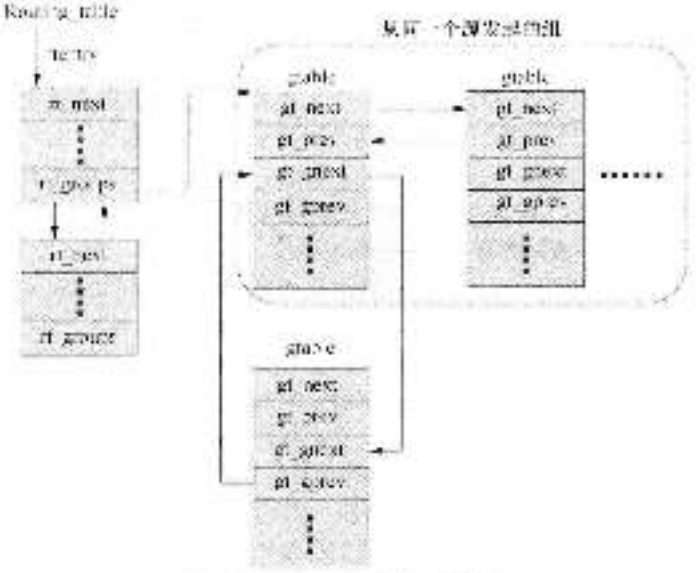


图 4-80 mrouted 的数据结构

算法实现

图 4-81 显示了与 mrouted 中组播路由相关的步数调用图。当接收到一个 IGMP 分组时，就调用 accept_igmp() 函数来处理分组。根据该分组的类型和代码，相应地调用不同的步数。如果类型与 IGMP 协议相关，例如，成员查询或报告（版本 1 或 2），那么就分别调用 accept_membership_query() 或 accept_group_report()。另一方面，如果分组的类型是 IGMP DVMRP，那么就检查该分组的代码以确定对应的操作。例如，如果代码是 dvmrp_report，那么就调用 accept_report()。在 accept_report() 中，处理分组中报告的路由并调用 update_route() 更新路由。如果代码是 DVMRP PRUNE，那么就调用 accept_prune()。在 accept_prune() 中，如果所有的子路由器已经表示对组没有兴趣，那么就调用 send_prune() 向上游路由器发送修剪消息。

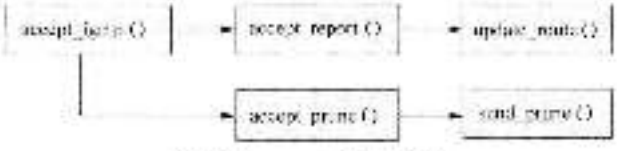


图 4-81 mrouted 开源实现

练习

在 mrouted 的源代码中继续以下三个函数：accept_report()、update_route() 和 accept_prune()，并画出流程图。比较所绘画的流程图与本节中介绍的 DVMRP 协议。

4.8 总结

在本章中，我们学习了网络协议栈中的互联网协议（IP）层或网络层，它是实现全球网络连接的最重要的一层。我们已经讨论了关于互联网的控制平面和数据平面的多种机制以便提供主机到主机之间的连接服务。在我们已经讨论过的这些机制中，路由和转发是该层中最重要的。路由是一种控制平面机制，它决定分组从源路由器到目的路由器的路由或路径。转发是一种数据平面操作。在路由器中，根据控制平面计算出的路由表将一个从进入网络接口到达的分组转发到外出网络接口。

假定一个路由器每秒需要处理数百万个分组，可扩展性对于这两种机制就非常重要。关于路由，我们已经知道互联网采用二层路由结构，也就是所谓的域内路由和域间路由。在较低层，将路由器分组为自治系统（AS）。在一个AS中的路由器处于同一行政控制并运行相同的域内路由协议，如RIP或OSPF。选择的路由器，称为边界路由器，相互连接起来并使用域间路由协议（如边界网关协议BGP）负责在AS之间转发分组。我们也学习了两种基本的路由算法，即距离矢量路由和链路状态路由。目前的互联网路由协议就是根据这两种基本的路由协议之一设计的。距离矢量路由算法采用分布式的方法，仅与邻居交换路由信息；而链路状态路由算法是一个集中式的方法，将所有路由信息泛洪到同一域内的所有路由器，因此每台路由器都建立一个所有路由器的全局拓扑数据库。对于转发，我们已经学习了路由表的数据结构以及这种数据结构的查找及更新算法。它们对可扩展性非常关键。在当前互联网骨干上的路由表有300 000条以上的表项，使转发更具挑战性。在某些情况下，可能需要特定的ASIC从CPU卸载路径表查找以便实现每秒百万分组的转发速度。

在本章中讨论了两种互联网协议（IP），分别是IPv4和IPv6。我们预测在未来的几年中，IPv6将占主导地位。为了应对地址耗尽问题，我们也介绍了网络地址翻译（NAT）协议和私有IP地址。除了IP协议外，我们还研究了多个控制平面协议，如ARP、DHCP和ICMP。

在本章中，我们还描述了一种通信类型，即单播、组播和广播。此外，我们已经看到了新的通信类型，IPv6支持的任播。单播，即点到点通信，一直是我们讨论的重点。IPv4和IPv6当然也支持广播和组播。IP子网定义为具有一个IP地址（称为子网地址）的广播域，它可以通过IP地址和子网掩码的AND（与）运算获得。将一个子网地址作为分组的目的地址时，分组就会传送到子网内的所有主机，这常对应于由多台两层设备组成的局域网。我们已经学习了多个依赖于广播服务的协议，如ARP和DHCP。在本章的最后一节，我们还学习了多个组播路由协议和成员管理协议。

完成了主机到主机连通性的研究后，接下来我们将学习处理到进程连通性，互联网协议栈的下一个上层。我们将学习来自同一台主机上不同进程的分组如何多路复用一起通过IP协议发送。我们还将学习如何在由IP协议提供的尽最大努力服务上构建可靠的通信。最后，我们将看到如何通过套接字编程接口编写网络应用程序。

常见陷阱

MAC地址、IP地址和域名

每个网络接口至少有一个MAC地址、一个IP地址和一个域名。它们被协议栈的不同层用于寻址。MAC地址总是与网卡在一起，它被链路层协议使用，它是一种在每个网卡的生产过程中分配、"固化"的唯一地址。因此，它是一个硬件地址。通常，MAC地址不具有层次化的结构，仅用于广播环境中的互通。IP地址被网络层协议使用，如本章所述。与MAC地址不同，IP地址具有层次化的结构，可用于路由。它可以手动或自动配置，因此，它是一个软件地址。域名是一串人类可读的字符。虽然在大多数情况下，域名是一个由英文字母组成的字符串，但目前它可以是任何语言。域名的目的是使人们容易记住主机的地址，尤其是像万维网这样的应用。域名表示成URL格式。当发送分组时，每层协议都需要地址翻译以便取得正确的地址。因此，使用域名系统（DNS）将域名翻译成IP地址，并利用地址解析协议（ARP）将IP地址翻译成MAC地址。DNS和ARP都支持缓存翻译。

转发和路由

再次强调理解转发和路由之间的差异非常重要。转发是一种数据平面功能，而路由是一种控制平面功能。转发的任务就是在一台路由器内将分组从一个进入网络接口转发到一个外出网络接口，而路由则是在任意两台主机之间查找一条路由路径。

有类 IP 和 CIDR

有类 IP 寻址是指在互联网协议中 IP 地址的最初设计。对于有类 IP 地址，每类 IP 地址的网络前缀长度是固定的，并且地址能很容易从首几位加以区别。此外，一个网络前缀可容纳的最大主机数量也是固定的。然而，这种设计造成 IP 地址分配的不灵活性并增加了路由表中 C 类地址的表项。无类域间路由（CIDR）允许可变网络前缀长度。CIDR 在聚合多个连续 C 类地址中是最有效的。目前，大多数的路由器支持 CIDR。

动态主机配置协议和 IPv6 自动配置

在 IPv4 中，动态主机配置协议（DHCP）用来自动配置主机地址。然而，在 IPv6 中，自动配置是通过 ICMPv6 协议使用路由器通告和路由器请求消息来支持的。它们是不同的吗？在一个全 IPv6 的网络中还需要 DHCP 吗？这两个问题的答案是肯定的。动态主机配置协议是基于 BOOTP 的。因此，在分组头部中的许多字段都未使用，尽管使用这些字段承载我们需要的信息。在 IPv6 中，自动配置过程是一种新的设计，而不是基于 DHCP 或 BOOTP。然而，出于安全或网络管理的考虑，网络管理员可以选择实用 DHCP 服务器控制 IP 地址的分配。

组播树和 Steiner 树

Steiner 树，以 Jakob Steiner 命名，是以源节点为根并以最小成本扩展到一组目的节点。它不同于最小生成树，因为目的节点集合不一定包括网中的所有节点（顶点）。因此，一棵 Steiner 树可以视为组播路由的一个最优解。不过，在我们研究的所有组播路由协议中，还没有一个试图建立一棵 Steiner 树。相反，它们中的大部分构建反向最短路径树，或可以以源为根也可以以汇聚点（RP）为根。原因在于找到一个 Steiner 树是一个 NP 完全问题，并且大多数启发式算法都需要全局信息。因此，反向最短路径树成为互联网构建组播树更加切实可行的解决方案。

进一步阅读

IPv4

从互联网协议发展的历史观点来看，下列文件很老但曾经是重要的先驱奠基性工作。在本章和第 1 章中已经学习了其主要思想。

- V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications*, Vol. 22, pp. 637–648, May 1974.
- J. B. Postel, "Internetwork Protocol Approaches," *IEEE Transactions on Communications*, Vol. 28, pp. 604–611, Apr. 1980.
- J. Saltzer, D. Reed, and D. Clark, "End-to-End Arguments in System Design," ACM Transactions on Computer Systems (*TOCS*), Vol. 2, No. 4, pp. 195–206, 1984.
- D. Clark, "The Design Philosophy of the Internet Protocols," *Proceedings of ACM SIGCOMM*, Sept. 1988.

与 IPv4、ICMP 和 NAT 相关的 RFC 分别是：

- J. Postel, "Internet Protocol," RFC 0791, Sept. 1981. (Also STD 0005.)
- K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS

Field) in the IPv4 and IPv6 Headers," RFC 2472, Dec. 1998.

- J. Postel, "Internet Control Message Protocol," RFC 792, Sept. 1981. (Also STD 0005)
- P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Jan. 2001.
- J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, Oct. 2008.

快速表查找

数据平面分组处理的一个有趣话题，就是分组转发和分组分类的快速表查询。前者是在一个字段（目的IP地址）的最长前缀匹配，而后者则是多字段匹配，如5元组（源/目的地址、源/目的端口号、协议号）。下面列出的第一篇论文是有关分组转发的软件算法，它只需要一张很小的表。而接下来的两个是硬件解决方案。最后两篇论文则介绍分组分类的软件解决方案。

- M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *ACM SIGCOMM'97*, pp. 3–14, Oct. 1997.
- M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed Routing Lookups," *ACM SIGCOMM'97*, pp. 25–36, Oct. 1997.
- P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," *IEEE INFOCOM*, Apr. 1998.
- P. Gupta and N. Mc Keown, "Packet Classification on Multiple Fields," *ACM SIGCOMM*, Sept. 1999.
- V. Srinivasan, G. Varghese, and S. Suri, "Packet Classification Using Tuple Space Search," *ACM SIGCOMM*, Sept. 1999.

IPv6

Bradner和Mankin的RFC是下一代IP的发起者。他们还出版了一本有关IPng的书。当前版本IPv6、ICMPv6和DNS可以分别在RFC 2460、RFC 4443和RFC 3596中找到。

- S. Bradner and A. Mankin, "The Recommendation for the Next Generation IP Protocol," RFC 1752, Jan. 1995.
- S. Bradner and A. Mankin, *IPng: Internet Protocol Next Generation*, Addison-Wesley, 1996.
- S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, Dec. 1998.
- A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Mar. 2006.
- S. Thomson, C. Huitema, V. Ksinant, and M. Souissi, "DNS Extensions to Support IP Version 6," RFC 3596, Oct. 2003.

主机和路由器中的基本IPv4向IPv6的过渡迁移机制在RFC 4213中描述。此外，迁移机制的问题方面可以在RFC 4038中找到。对于一种迁移方法，即双栈、隧道和协议转换，已经提出了许多建议。例如，在RFC 3053中提出了隧道代理以帮助用户配置双向隧道。6to4和其补救措施Teredo，分别在RFC 3056和RFC 4380中描述。一个新的有IPv4基础设施上快速部署IPv6的机制（6rd），即在6到4之上构建是在RFC 5569中建议的。ISATAP定义在RFC 5214中。协议转换解决方案如SIIT和NAT-PT分别定义在RFC 2765和RFC 4966中。最后，Geoff Huston写了多篇关于IPv4地址耗尽问题和过渡到IPv6方法的有趣文章。

- E. Nordmark and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers," RFC 4213, Oct. 2005.
- M-K. Shin, Ed., Y-G. Hong, J. Hagino, P. Savola, and E. M. Castro, "Application Aspects of

IPv6 Transition," RFC 4038, Mar. 2005.
- A. Durand, P. Fasano, I. Guardini, and D. Lento, "IPv6 Tunnel Broker," RFC 3053, Jan. 2001.
- B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds," RFC 3056, Feb. 2001.
- C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, Feb. 2006.
- E. Exist and R. Despres, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)," RFC 5569, Jan. 2010.
- T. Templin, T. Gleeson, and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)," RFC 5214, Mar. 2008.
- E. Nordmark, "Stateless IP/ICMP Translation Algorithm (SIIT)," RFC 2765, Feb. 2000.
- C. Aoun and E. Davies, "Reasons to Move the Network Address Translator-Protocol Translator (NAT-PT) to Historic Status," RFC 4966, July 2007.
- Geoff Huston, "IPv4 Address Report," retrieved April 24, 2010, from http://www.potaroo.net/tools/ipv4/index.html.
- Geoff Huston, "Is the Transition to IPv6 a 'Market Failure?'," *The ISP Column*, Apr. 2010, retrieved April 24, 2010, from http://ccr-report.org/isp-col/2009-09/v6trans.html.

路由

RIP、OSPF 和 BGP 的最新 RFC 是：

- C. Malkin, "RIP Version 2," RFC 2453, Nov. 1998.
- J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998. (Also STD0054.)
- R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340, July 2008.
- Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006.

在文献中以经济由于竞争元化表示成一种网络资的问题，其中流量被为网络中的弧和上的账之间流。Bertsekas 和 Gallagher 的教科书针对这种处理给了一个很好的表程。

- D. Bertsekas and R. Gallagher, *Data Networks*, 2nd edition, Prentice Hall, Englewood Cliffs, NJ, 1991.

有关互联网路由，OSPF 和 BGP 更加详细的研究，参从下三书中可能会很有用。

- C. Huitema, *Routing in the Internet*, 2nd edition, Prentice Hall, 1999.
- S. Halabi and D. McPherson, *Internet Routing Architectures*, 2nd edition, Cisco Press, 2000.
- J. T. Moy, *OSPF: Anatomy of an Internet Routing Protocol*, Addison-Wesley Professional, 1998.
- I. V. Beijnum, *BGP*, O'Reilly Media, 2002.

或同路由的安全性已经证证重和受方式受到人们的重视。2005 年 12 月 11 月出版的《IEEE Network Magazine》有关域间路由的专刊上，可以发现许多有趣的结果。近来，BGP 容错也备受关注，尤其是基于多路径路由的解决方案。Xu 等人和 Wang 等人的论文就是很好的例子。

- M. Caesar and J. Rexford, "BGP Routing Policies in ISP Networks," *IEEE Network*, Vol. 19, Issue 6, Nov/Dec 2005.
- R. Musunuri and J. A. Cobb, "An Overview of Solutions to Avoid Persistent BGP Divergence," *IEEE Network*, Vol. 19, Issue 6, Nov/Dec 2005.
- A. D. Jaggard and V. Ramachandran, "Toward the Design of Robust Interdomain Routing Protocols," *IEEE Network*, Vol. 19, Issue 6, Nov/Dec 2005.
- W. Xu and J. Rexford, "MIRO: Multi-Path Interdomain Routing," *ACM SIGCOMM*, Sept. 2006.
- F. Wang and L. Gao, "Path Diversity Aware Interdomain Routing," *IEEE IEEE INFOCOM*, Apr. 2009.

组播

虽然在互联网上的部署不是很成功，但已经为域内和域间路由协议提出了许多协议。在 Ramalho 的专题论文中对组播进行了相当全面的研究。对组播的早期工作，Deering 和 Cheriton 的论文是必读的。关于 IPv6 组播与 IPv4 组播的比较，读者可以参阅 Metz 和 Tatipamula 的论文。

- M. Ramalho, "Intra-and Inter-Domain Multicast Routing Protocols: A Survey and Taxonomy," *IEEE Communications Surveys and Tutorials*, Vol. 3, No. 1, 1st quarter, 2000.
- S. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Transactions on Computer Systems*, Vol. 8, pp. 85-110, May 1990.
- C. Metz, and M. Tatipamula, "A Look at Native IPv6 Multicast," *IEEE Internet Computing*, Vol. 8, pp. 48-53, July/Aug 2004.

组播成员管理和路由的最新 RFC 是：

- B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," RFC 3376, Oct. 2002.
- D. Waitzman, C. Partridge, and S. E. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, Nov. 1998.
- B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification (Revised)," RFC 4601, Aug. 2006.
- N. Bhaskar, A. Gall, J. Lingard, and S. Venaas, "Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM)," RFC 5059, Jan. 2008.
- D. Kim, D. Meyer, H. Kilmer, and D. Farinacci, "Anycast Rendevous Point (RP) Mechanism Using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP)," RFC 3446, Jan. 2003.

常见问题解答

1. 对于网络接口，我们为什么既需要 MAC 地址又需要 IP 地址呢？为什么不只使用一种地址呢？
   答：如果只使用 IP 地址，就没有协路层寻址，没有转接，没有广播能力。
   如果仅使用 MAC 地址，就没有分层的互联网体系结构，没有子网掩码，没有路由。
2. 为什么 MAC 地址是平面的，但 IP 地址是层次化的？
   答：MAC 地址：全球唯一制造，但没有与位置关联起来，因此是平面的。
   IP 地址：全球唯一配置，包含与位置关联的信息，因此是层次化的。
3. 为什么在路由器和主机内部使用子网掩码？
   答：路由器：在匹配的前缀中选择最长的一个。
   主机：确定目的 IP 地址是否在用户的子网内。
4. 路由与转发有什么不同？（比较它们的工作类型和所使用的算法。）
   答：转发：数据平面，通过表查找匹配最长的前缀。
   路由：控制平面，由 Dijkstra 或 Bellman-Ford 算法计算最短的路径。
5. 在路由器的转发表中，为什么会有多个匹配的 IP 前缀？（解释什么网络配置可能导致它的发生。）
   答：如果分配了前缀（比如 140.113/16）给组织已经创建了层次分支机构（如 140.113.0/18 和 140.113.192/18），那么将有多个前缀，在本例的所有路由器中有 3 个。如果将分组发到 140.113.221.86，它既与 140.113/16 匹配又与 140.113.192/18 匹配，后者是最长匹配。
6. 在 Linux 内核中是怎样构造前缀表的？为什么匹配的前缀保证是最长的呢？
   答：转发表组织成相同前缀长度的链列表数组。根据前缀长度对数组进行排序。从具有最长前缀的非空数列表开始，从而保证第一个匹配是最长匹配。
7. 在 Linux 内核中转发表是如何组织的？

答：它包括一个转发缓存和一个 FIB（转发信息库），其中前者是一个散列表，用于存储最近查找过的表项，而后者是按照前缀长度的散列表数组（转发缓存中没有命中后可以继续查找）。

8. 在目的主机的 IP 重组中需要什么头部字段？

答：标识符、更多位和分段偏移。

9. 为了让 FTP 通过 NAT 需要修改哪些分组？

答：非 ALG 修改：外出（进入）分组的源（目的）IP 地址和源端口号、IP 头部校验和、TCP 校验和。

ALG 修改：在 FTP 消息中的 IP 地址和端口号、TCP 序列号和 TCP 确认号。

10. 让 ICMP 通过 NAT 需要修改哪些分组？

答：非 ALG 修改：外出（进入）分组的源（目的）IP 地址和源端口号、IP 头部校验和。

ALG 修改：ICMP 消息中的 ICMP 校验和、IP 地址。

11. 在 Linux 内核中 NAT 表如何实现？

答：散列表。

12. IPv4 中的哪些头部字段在 IPv6 中被取消了，又有哪些新的字段添加到 IPv6 头部中？为什么？

答：删除的：头部校验和、分段（标识符、更多位、不分段位、分段偏移）、协议及选项。

添加的：流标签和下一个头部。

13. IPv4 和 IPv6 如何才能共存？

答：双协议栈：在路由器和主机中有 IPv4 和 IPv6 协议栈。

隧道：在 IPv6 孤岛之间采用 v6 - v4 - v6 隧道，而在 IPv4 孤岛之间采用 v4 - v6 - v4 隧道。

14. 主机如何通过 ARP 将 IP 地址翻译成 MAC 地址？

答：以指定的 IP 地址在本地子网上广播一个 ARP 请求，并从具有指定 IP 地址的主机上获得一个单播 ARP 响应。

15. 主机如何通过 DHCP 或 ARP 获取其 IP 地址？

答：DHCP：广播 DHCPDISCOVER 以便找到一台 DHCP 服务器，然后得到配置。

ARP：广播一个含有自己 MAC 地址的 RARP 请求，并从 RARP 服务器获得单播 RARP 响应。

16. ping 和 tracepath 是如何实现的？

答：Ping：ICMP echo 请求和应答。

Tracepath：以 TTL = 1，2 等，重复发送 UDP 或 ICMP echo 请求，直到从目标主机接收到一个 ICMP 端口不可达（对于 UDP）或 ICMP echo 应答（对于 ICMP echo 请求）为止。

17. 在 RIP 中为何会发生计数无穷问题？

答：路由器感到链路故障更新并与邻居路由器交换距离矢量。如果路由器接收并接受来自邻居的距离矢量而不检查路径是否经过它自身，那么直到可用路径信息传播到这里路由器可能才会终止相互递增地更新距离矢量。在此期间，在两台对等路由器之间可能产生分组循环。

18. RIP 与 OSPF 有什么不同？（比较它们的网络状态信息和路由计算。）

答：RIP：与邻居交换距离矢量，距离矢量更新使用 Bellman-Ford 算法。

OSPF 协议：向所有路由器广播链路状态，利用 Dijkstra 算法根据全部拓扑计算路由表。

19. 距离矢量路由与链路状态路由有什么不同？（比较它们路由消息的复杂性、计算的复杂性、收敛速度和可扩展性。）

答：路由信息的复杂度：DV > LS。

计算的复杂度 LS > DV。

收敛速度：LS > DV。

可扩展性：DV > LS。

20. RIP 与 BGP 有什么不同？（总结它们的异同。）

答：相同之处：交换邻居信息，Bellman-Ford 算法。

不同之处：距离矢量与路径矢量（对于无环路路由），UDP 上的 RIP 与 TCP 上的 BGP，最短路径路由与最短路径非策略路由，单路径与多路径。

21. 为什么 RIP、OSPF 和 BGP 分别运行在 UDP、IP、TCP 上？

答：RIP：一种无连接的UDP套接字，运行在UDP端口52，它可以接收请求并向所有的邻居路由器发送响应（通告）。

OSPF：一种原始IP套接字，用于向域中所有的路由器广播链路状态。

BGP：面向连接的TCP套接字，用于与远程对等路由器之间的可靠传输。

22. 你能估计出AS内和AS间路由器中的路由表项的数量吗？（估计范围或数量级。）

答：AS内：几十到数百，取决于域有多大。

AS间：遍布世界各地，成千上万，取决于前缀数。

23. 在zebra中，路由协议如何与其他路由器交换信息，如何更新内核中的路由表？

答：路由消息交换：通过各种套接字（IP、UDP、TCP）与其他路由器交换信息。

路由表更新：通过inet、sysctl、netlink、rtnetlink等访问内核。

24. 路由器如何通过IGMP知道在其子网中的主机是否已经加入到组播组中？

答：路由器在其子网广播通用查询或特定组查询（发送到224.0.0.1所有系统组播组）以便请求成员信息。如果在其知机家时器过期之前子网上没有一个响应，那么主机通过响应/广播TTL=1的IGMP报告来加入。该路由器知道在特定组播组中是否存在一台主机，但不知道是谁以及有多少。

25. 路由器是否能够准确地知道已经加入到一个组播组中的主机？

答：不知道。它只知道子网中是否有一台主机在特定组播组中。

26. 将基于源与基于核心的组播列进行对比（比较它们的状态数量和可扩展性）。

答：状态数：基于源 > 基于核心。

可扩展性：基于核心 > 基于源。

27. 为基于源和基于核心的组播路由器在路由器中分别存放多少状态？（考虑组播组和源的数量。）可能会保持什么样的状态信息？

答：基于源：每个组×每个源，即（组，源）对。

基于核心：每个组的状态信息：子网的成员状态、修剪状态或加入状态。

28. 组播分组在以DVMRP中的反向路径多播中真的会沿着最短路径流动吗？

答：不一定。最短路径是从下游路由器到源路由器。而反向路径不一定是从源路由器到下游路由器中最短的。

29. 为了DVMRP中的反向路径组播，路由器中应该保存哪些状态信息？

答：每组、源修剪状态。

30. 如何确定PIM-SM中组播组的RP？

答：域中的所有组播路由器使用相同的散列函数对D类组播IP地址。转换为散列值以便从候选路由器列表中选出组播路由器。

练习

动手练习

1. 使用Wireshark或类似的软件，观察一个大的IP分组的分段。
2. 使用Wireshark或类似的软件，持续捕捉分组数秒。从捕获的分组数据中寻找ARP和IP分组。比较这两个分组的MAC头部之间的区别。你能找出ARP和IP的协议标识符吗？ARP报文的目的地址是广播地址还是单播地址？ARP分组是一个请求还是应答分组？检查这个ARP分组的有效载荷。
3. 使用Wireshark或类似的软件捕获IP分组，分析这个分组的头部和有效载荷。是否能够识别出传输层协议和应用层协议？
4. 使用Wireshark或类似的软件探究如何使用ICMP消息实现ping。用捕获的分组来验证你的答案。请注意，ping命令在不同操作系统上可以不同（提示：使用Wireshark进行捕获，然后使用命令行发出一条ping命令）。
5. 使用Wireshark或类似的软件探究如何使用ICMP消息实现traceroute。
6. 使用cisco-route或traceroute探究你所在域的基础结构以及到国外的路由（提示：traceroute会提供一张

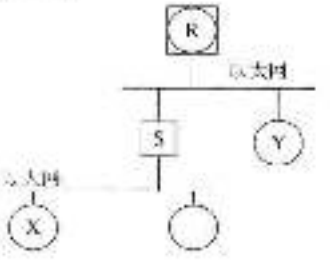
路由器列表（尝试通过其子网地址和往返延迟识别不同类型的路由器）。

7. 使用基于 Linux 的 PC 构建 NAT 服务器（提示：Linux 通过 IP TABLES 实现 NAT）。
8. 使用基于 Linux 的 PC 构建 DHCP 服务器。
9. 编写程序来执行 ping 命令（提示：使用原始套接字接口发送 ICMP 分组，请参阅第 5 章套接字接口）。
10. 在 Linux 的源代码中跟踪 ip route input()和 ip route output key()，分别描述 IP 分组如何转发到上层和下一跳（提示：这两个函数都可以在 net/ipv4/route.c 中找到）。

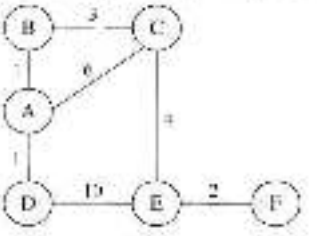
书面练习

1. 当两台主机使用相同的 IP 地址而忽略了彼此的存在时，会发生什么问题？
2. 试比较电话系统与互联网的寻址层次化结构（提示：电话系统使用地理寻址）。
3. 为什么在 IP 中需要分段？为了分段和重组，IP 头部需要哪些字段？
4. IPv4 头部的标识符（序列号）字段的作用是什么？打包（wrap around）是一个问题吗？举一个说明打包（wrap around）问题的例子。
5. IP 协议如何区分 IP 分组的上层协议？例如，它如何知道分组是否是 ICMP、TCP 或 UDP 分组？
6. 以太网驱动程序如何确定一帧是否是一个 ARP 分组？
7. 考虑 IP 分组穿越路由器：当 IP 分组穿越路由器时，路由器必须修改 IP 头部中的某个字段？IP 头部中哪个字段可以由路由器修改？为重新计算校验和字段设计一种有效的算法（提示：考虑这些字段如何更改）。
8. 考虑给公司分配一个 IP 前缀 163.168.80.0/22。该公司拥有三个分支机构，分别有 440、70、25 台计算机。为每个分支机构分配带有两个广域网接口的路由器以便提供网络互联，这样三台路由器是完全连接的。如果要求你为这三个分公司和路由器接口地址的子网地址进行规划，你将如何做？（提示：两台路由器之间的链路也需要一个子网。）
9. 如果主机的 IP 地址为 168.168.168.168，子网掩码为 255.255.255.240，那么其子网地址是什么？这个子网的广播地址是什么？在该子网中有多少台块的 IP 地址？IP 地址是一个 B 类地址。假设它属于一家公司。如果所有子网的子网掩码为 255.255.255.240，那么该家公司可以创建多少个子网？
10. 考虑主机 X 的 IP 地址为 163.168.2.81，子网掩码为 255.255.255.248。现在，假设 X 向以下 IP 地址（主机）分别发送分组：163.168.2.76、163.168.2.86、163.168.168.168、140.123.101.1。对于每个 IP，路由有何不同？如何发送不同的 ARP 分组以便找到 MAC 地址？（对于每一个 IP 地址，路由和 ARP 分组的发送，可以是相同的也可以是不同的。解释你的答案。）
11. 当一个 IP 分组分段时，丢失一个分段将导致整个分组被丢弃。考虑一个包含 4800 字节数据（来自上层）的 IP 分组，将它发送到直接连接的目的地。考虑具有不同 MTU 的两种链路层类型。类型 A 技术使用 5 字节的头部并有 53 字节（你可能认为它是 ATM 技术）的 MTU。类型 B 技术使用 18 字节的头部而 MTU 是 1518 字节（比如以太网）。假设类型 A 的帧丢失率是 0.001，类型 B 的帧丢失率为 0.01。比较这两种类型的链路层技术的分组丢失率。
12. 通过快速以太网连接发送 1MB 的 MP3 文件最少需要多少个 IP 分段？（提示：忽略 IP 层以上的头部。一个最大的 IP 分段包含一个 20 字节的头部和一个 1480 字节的有效载荷。）
13. 重组分段时，接收器如何知道两个分段属于同一个 IP 分组？它如何知道每个分段的大小是正确的？
14. 在你看来，服务质量如何在 IPv6 中得到更好的支持？
15. 为什么 IPv6 扩展头的顺序很重要，而不能更改？
16. 描述在 RFC 1981 中定义路径 MTU 的发现过程。
17. 比较 IPv4 和 IPv6 头部格式之间的差异，找到不同，并解释为什么要进行这些更改。
18. 比较 ICMPv4 和 ICMPv6 之间的差异。在 IPv6 中还需要 DHCP、ARP、IGMP 吗？
19. 在 IPv4 头部中，有一个协议标识符字段。这个字段的作用是什么？在 IPv6 中是否也有相应的字段？
20. 给出一个 6000 字节的 IP 分组，假设分组是通过以太网传输的。分别解释在 IPv4 和 IPv6 中它是如何分段的（你应该清楚地说明，会产生多少分段，每一项的大小，以及在每个 IP 头部中如何相应地进行设置）。
21. 讨论在虚电路子网（如 IP over ATM）上建立无连接服务的难点。
22. ARP 缓存时间到期值是如何影响其性能的？

23. 在一个子网内广播 ARP 请求以获得同一子网内主机的 MAC 地址。使用 ARP 请求获得子网外的一台远程主机的 MAC 地址是否有意义？
24. 如果入侵者使用一台 DHCP 欺骗设备在真正 DHCP 服务器做出应答之前发送对 DHCP 请求的应答，会发生什么后果？
25. 一台攻击设备不断地从一台真正的 DHCP 服务器通过不断变化的 MAC 地址来请求 IP 地址，是否可能？（提示：这是所谓的 DHCP 饥饿问题。）
26. BOOTP 和 DHCP 之间的区别是什么？为什么 DHCP 是基于 BOOTP 设计的？
27. 设 A 是一台具有私有 IP 的主机，通过 NAT 服务器连接到互联网上。主机 A 所在子网以外的主机能够远程登录（telnet）到 A 吗？
28. 为什么 NAT 会成为 P2P 应用的问题？我们需要为对称 NAT（symmetric NA）和非对称 NAT（cone NAT）提供不同的解决方案吗？
29. 考虑下面的局域网，具有一台以太网交换机 S，一台域内路由器 R 和两台主机 X 和 Y。假设交换机 S 已经通电。
    a. 当 X 向 Y 发送 IP 分组时，描述在 X、Y 和 S 上执行的路由和地址解析步骤。
    b. 描述当 Y 向 X 应答 IP 分组时，在 X、Y 和 S 执行的路由和地址解析步骤。
    c. 当 X 向域外的一台主机发送 IP 分组时，描述在 X、S 和 R 上执行的路由和地址解析步骤（提示：不要忘记解释 X 如何知道路由器 R。）

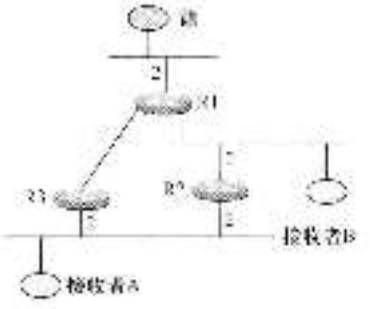


30. 考虑以下的网络拓扑结构。说明节点 A 分别使用链接状态路由、距离矢量路由，如何构造其路由表。



31. 继续第 30 题。现在假设链路 A—B 发生故障。LS 和 DV 路由将如何应对这种变化？
32. 比较 LS 和 DV 路由的消息复杂性和收敛速度。
33. 假设对所有链路成本，已知有一个正的下界。设计一种新的链路状态算法，在每次迭代时它可以将多个节点添加到集合 N 中。
34. 距离矢量路由算法既被域内路由（如 RIP）也被域间路由（如 BGP）所采用，但是以不同的节点和附加功能来实现的。当它们都使用距离矢量算法时，试比较域内路由和域间路由之间的差异。
35. 路由循环是 RIP 中的一个问题。为什么在 BGP 中就没有这个问题？
36. 链路状态路由和距离矢量路由之间的主要区别是什么？距离矢量算法的稳定性问题是什么，对这些问题的可能解决方案是什么？
37. 如果路由的目标是找到最大可用带宽路径（称为最宽的路径），那么我们应该如何定义链路成本？在计算路径成本时需要改变什么？（不只是将链路成本相加到路径成本中！）
38. 何谓最长前缀匹配？为什么路由器要使用最长前缀匹配？这仍将是 IPv6 中的问题吗？（为什么是或者不是，证明你的答案。）

39. 为多媒体应用提供 QoS，曾经需要研究 QoS 路由（但没有成功）。考虑一个需要固定位速率传输的视频流应用。我们如何为这种应用执行 QoS 路由？解释如何定义链路成本函数，如何从链路成本计算路径成本。路由决策的统计、应用协议和 QoS 之间的相互作用。
40. 考虑在两个 mrouters 之间的隧道技术。描述如何将组播分组封装到单播分组中。在隧道另一端的 mrouter 如何知道它是一个封装的分组？
41. 由于 IP 组播地址分配没有集中控制，如果它们随机选择地址，那么两组用户选择相同组播地址的概率是多少？
42. 考虑 IGMP 协议的操作，当一个路由器（查询者）向其子网内的一台路由器发送特定组查询消息时，如果有许多用户的组播组，那么如何抓住 ACK（报告消息）爆炸问题？
43. 在 IGMPv3 中，我们如何执行定义限定组播分组？
44. DVMRP 能够最小化到每个目的地所使用的网络带宽或端到端延迟吗？节点会接收相同分组的多个副本吗？如果是，建议一个使所有节点仅收到一份的新协议副本。
45. PIM 包括两种模式：密集模式和稀疏模式。这两种模式之间的差异是什么？为何定义这两种模式？
46. 在 PIM-SM 中，路由器如何知道在哪里可以为组播组新加入的成员找到 RP？
47. 当主机向组播组发送分组时，由指定路由器在 DVMRP 和 PIM-SM 下处理分组有何不同？
48. 具有最小成本的组播树称为 Steiner 树。为什么在 IETF 的 RFC 建议中没有任何协议尝试构建一棵 Steiner 组播树？
49. 一般情况下，我们认为一个基于源的树的成本应小于基于共享的树的成本。你同意还是不同意？为什么？构建一个反例证明基于源的树的成本实际上大于基于共享的树的成本。
50. 说明在以下网络拓扑下，DVMRP 构建的组播树。



第5章

Computer Networks: An Open Source Approach

传输层

传输层又称为端到端协议层，就像是整个互联网协议簇的接口一样，为应用程序提供端到端的服务。第3章的重点放在链路层，它在直接链接起来的节点之间提供节点到节点的单跳通信信道。在第3章出现的“多快地发送数据？”和“数据能够正确地到达连接到同一有线或无线链路上的接收者吗？”这样的问题并得到解答。在IP层提供穿越互联网的主机到主机的多跳通信信道。在第4章上会产生与IP层相类似的问题并得到解答。接下来，因为在一台主机上可能运行多个应用进程，所以传输层将在互联网上不同主机的应用进程之间提供或程到进程的通信信道。传输层提供的服务包括：1）寻址；2）多路控制；3）可靠性；4）速率控制。寻址服务确定一个数据包属于哪个应用进程；如果接收到的数据是有效的，就要通过差错控制的检测；可靠性服务保证传输的数据到达其目的地；为了流量控制和网络拥塞控制，速率控制则节发送方应多快地将数据传送接收方。

广泛的应用程序会有各种不同的需求，传输层应该提供什么服务是一个大问题。随着时间的推移传输协议已经演变成了两个主要协议：复杂的传输控制协议（TCP）和原始的用户数据报协议（UDP）。尽管TCP和UDP利用相同的寻址方案和类似的差错控制方法，但它们在可靠性和速率控制上的设计存在很大的区别：TCP实现了上述所有的服务，但UDP则完全忽略了可靠性和速率控制服务。由于其复杂的服务，TCP需要首先在通信主机之间建立一条端到端逻辑连接（也就是说，它是面向连接的），并在终端主机保持必要的每个连接或每个流的状态信息（即，它是有状态的）。这种面向连接的和有状态的设计目的就是为了实现每个流的可靠性和特定的进程到进程信道的速度控制。另一方面，UDP是无状态的和无连接的，不需要建立一条连接行使其寻址和差错控制方案。

为了能够支持运行实时传输的应用，TCP或UDP所提供的服务由于缺乏通信主机之间的定时和同步信息是有局限性和不足的。因此，大多数实时应用经常会在原始的UDP之上集成一个额外的协议层，以提高服务质量。为了实现该目的，有一对标准协议分别是：实时传输协议（RTP）/实时控制协议（RTCP）。这对协议提供音频和视频流之间的同步、数据压缩和解压缩的信息，以及路径质量的统计（数据包丢失率，端到端的延迟及其变化）等服务。

由于传输层与应用层直接耦合，所以互联网套接字常简称为套接字，充当程序员访问互联网协议族底层服务的一个重要应用编程接口（API）。然而，TCP和UDP套接字接口不只是由应用层访问。应用程序可以绕过传输层直接使用IP层或链路层提供的服务。稍后我们将讨论Linux程序员如何通过各种套接字接口访问传输层以下的IP层甚至链路层的服务。

本章的组织结构如下：5.1节说明传输层的端到端问题，并将它们与链路层的问题进行比较。5.2节和5.3节描述互联网如何解决传输层的问题。5.2节说明原始的传输层协议UDP，它提供了基本的进程到进程的通信信道和差错控制。5.3节重点介绍广泛使用的传输层协议TCP，它使应用不仅具有进程到进程的通信信道和差错控制，而且具有可靠性和速率控制。到目前为止，讨论过的互联网协议簇服务包括哪些在第3、4、5章中的内容，应用程序员可以通过各种套接字接口直接访问。5.4节介绍了实现套接字接口的Linux方法。然而，由于实时应用中的额外软件层，通常嵌入协议RTP/RTCP作为应用程序中的库函数。5.5节介绍了应用层如何采用RTP/RTCP协议。

在本章结束时，读者应该能够回答：1）为什么将互联网协议簇的传输层设计成今天这样的方式；2）Linux如何实现传输层协议。

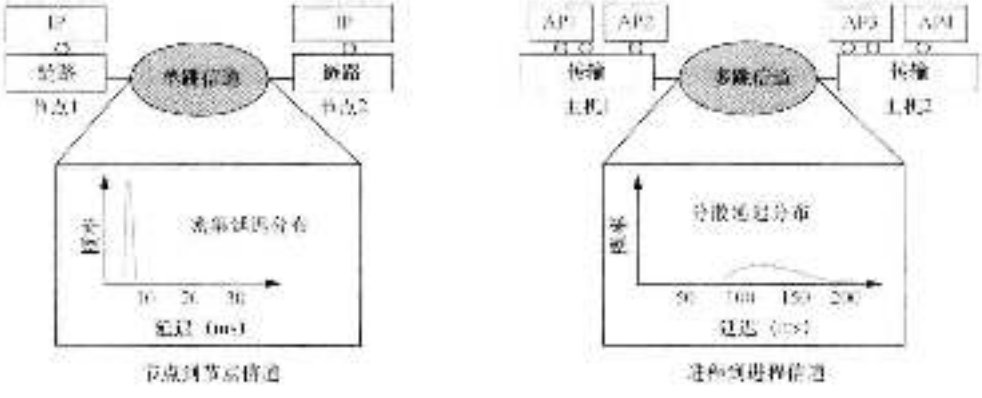
5.1 一般问题

运输层或端到端协议，顾名思义，定义通信信道的终点之间传输数据的协议。让我们首先定义贯穿本章所使用的一些术语：在操作系统之上运行的应用程序就是一个进程，在传输层的数据传输单元简称为分段，在进程到进程的信道中流动的通信量为流。传输层中最明显的服务是向应用进程提供

程到进程的通信信道。由于在一台主机上可能同时运行了多个进程，所以在进程到进程信道的帮助下，在互联网上的任何一台主机上运行的任何进程都能够与此通信。进程到进程信道的有关问题与第3章中节点到节点信道的有关问题相类似。一般情况下，传输层通过给报到进程通信加上以每个分段为基础的多路复用和以每个分段为基础的可靠性因素来解决这些问题。它通过对每个新创建的进程分配一个端口来解决了要求共享需求。

5.1.1 节点到节点与端到端

通过进程到进程信道的通信，第3章中已经讨的经典问题会再次出现，但曾经的解决方案却不适用于这里。如图5-1所示，单跳节点到节点与多跳进程到进程信道之间的主要区别在于处理分配。延迟是从一台主机传输到另一台主机通过信道的时间延迟。在第3章中，可靠性和速率控制问题很容易解决，因为直接相连的主机之间的延迟分布非常集中地位于一定值的范围内，具体取决于所选择的链路层技术。相反，进程到进程信道的延迟很大，可能有很大的变化。所以在传输层中的可靠性和速率控制算法应该容纳大的延迟和很大的延迟变化（通常称为抖动）。



○ 服务接入点

图5-1 单跳和多跳信道之间的区别

表5-1介绍了单跳信道上的链路层协议与多跳信道上的传输协议之间的详细比较。传输协议在IP层之上提供服务，而链路协议在物理层之上提供服务。因为在链路上可能连接了多个节点，所以链接层通过定义节点地址（MAC地址）来确定点对点链路上的节点到节点通信信道。同样，在每端的主机上可能运行多个进程，因此传输层定义端口号来说明主机上的进程。

表5-1 链路层协议与传输层协议之间的比较

		链路层协议	传输层协议
基本任务		物理链路	IP层
服务	寻址	局域网的节点到节点信道（由MAC地址表示）	主机之间的进程到进程信道（由端口号表示）
	无链接的	提供	按分段
	可靠性	无比较	提供
	速率控制	按链路	按端
信道延迟		很集中	扩散分布

寻址

在传输层上的寻址相当简单，我们只需要使用一个唯一的标识号标记运行在本地主机上的每个进程。因此，进程地址的长度与链路层或网络层地址相比应该非常短。本地主机操作系统可以轻松地分

配进程地址。正如我们将要在本章中学习到的，互联网解决方案使用一种16位端口号作为进程地址。一个特定应用程序的端口号既可以是一个众所周知的被个球所有主机使用的端口号，也可以是由本地主机动态分配的任何可用的端口号。

5.1.2 差错控制和可靠性

差错控制和可靠性对端到端通信非常重要，因为互联网偶尔会丢弃、重新排序或者重复发送分组。差错控制主要对一个传输数据单元内的位错误进行检测或恢复，不管它是一个帧还是一个分段，可靠性都能够进一步提供重发机制来恢复可能丢弃的或没有正确接收的数据单元。表5-1说明链路层协议采用基于帧的差错控制方式，而传输层协议采用基于分段的差错控制。在链路层协议和传输协议中使用的错误检测通常分别是循环冗余校验（CRC）和校验和。正如3.1.3节所述，CRC在多位错误检测中更强大并且更容易在硬件中实现，而传输层协议中的校验和只能充当一种在节点处理数据时发生的节点错误的复查。

为了可靠传输，端到端协议提供每个流的可靠性控制，但是大多数链路层协议，如以太网和PPP，或者在其机制中加入重传功能。它们将重传的任务留给它们的上层协议来完成。然而，有些链路协议，如WLAN，工作在可能发生严重帧损失的环境下，所以这些链路协议都内置了可靠性机制，以提高由于上层协议频繁地重传导致的效率低下。例如，来自传输层的巨大的流出分段在IP层分为10个分组后，然后会在WLAN形成10个帧，WLAN可以可靠地传输10个帧而不用采用所有分段的端到端重传。因此，与没有内置可靠性机制的无线局域网情况相比，整个帧具有更低的重传概率。

延迟分布对于可靠地传输也很重要，因为它影响多个定时器的设计。如图5-1所示，链路信道的延迟分布集中于某一个值的周围，所以将链接信道的重传定时器设置为经过一个固定时间段的超时，如10毫秒是合当的。但是，由于端到端信道的扩展延迟分布，所以将这种技术应用到传输层会出现很多问题。例如，在图5-1中，如果我们为端到端信道设置超时值为150毫秒，那么某些分段会错误地重发，网络会因此而包含许多重复的分段，但是如果我们设置超时值为200毫秒，丢弃的分段不会重发，直到这个长等待定时器到期为止，最终导致差的性能。所有这些权衡都会影响链路和端到端信道的设计选择。

5.1.3 速率控制：流量控制和拥塞控制

速率控制包括流量控制和拥塞控制，在传输层协议中起着比链路层协议更重要的作用。因为传输层协议所运行的广域网环境比链路层协议运行的局域网环境复杂得多。流量控制仅运行在源和目的地之间，而拥塞控制运行在源和目的地网络之间。也就是说，网络中的拥塞能够通过拥塞控制而得到缓解，但不能通过流量控制得到缓解。在链路层协议中没有拥塞控制，因为发射机离接收器只有一跳。

拥塞控制既可以由发送者也可以由网络来完成。基于网络的拥塞控制在中间路由器上采用各种排队规则和调度算法，以避免网络拥塞。基于发送者的拥塞控制依赖于每个发送者的自我控制，以避免太快发送太多的数据到网络中。然而，基于网络的拥塞控制，超出了本章的范围，将在第7章中讲解。

在文献中，流量控制或拥塞控制机制都可分为成基于窗口的和基于速率的。基于窗口的控制是通过控制能够同时传输的未经确认的分组的数量来调节发送速率。一个未经确认的分组代表已发送的分组，但还没有返回对它的确认。另一方面，当基于速率控制的发送者收到明确通知它应该多快地发送时，它直接调整其发送速率。

实时需求

既然实时应用需要额外的信息构建帮放，那么除了上述外还应该提供其他额外的支持。它们可能包括音频和视频流之间的同步、数据压缩和解压缩信息，以及路径质量统计值（分组丢失率、端到端延迟及其变化）。为了支持这些额外的需求，所有需要的额外信息，如时间戳、编解码器类型、分组丢失率，都必须在协议消息头部中携带。由于TCP和UDP在其头部中没有这些字段，所以就需要其他传输协议来满足实时的需求。

5.1.4 标准编程接口

网络应用程序通常通过套接字编程接口访问网络层服务。大多数应用程序运行在 TCP 或 UDP 之上，其实现者可以对套接字发送缓冲区和接收缓冲区的大小进行控制，也可以控制诸如最大报文段长度（MSS）等参数。应用程序开发者可以控制传输层协议（TCP 或 UDP），也可以控制传输层协议的一些参数。

套接字接口最初是提供给运行在 UNIX 操作系统上的进程间通信（IPC）使用的。IPC 有三个组成部分：提供 IPC 的系统调用接口、独立于通信协议的通信语义的实现以及特定的通信协议。套接字接口的最高层是一个系统调用接口，应用程序可以通过这个接口创建套接字、连接套接字、发送数据和接收数据。套接字接口的中间层实现了独立于通信协议的通信语义。最低层实现了特定的通信协议，例如 TCP 和 UDP。

套接字接口的实现方式有很多种，例如，可以将套接字接口实现为一个库，应用程序通过调用库中的函数来访问传输层服务。这些函数通过向操作系统内核发送适当的命令来实现应用程序的请求。

套接字接口的实现方式也可以是将套接字接口直接嵌入操作系统内核中。这种实现方式的优点是效率高，缺点是灵活性差，因为每次修改套接字接口都需要重新编译内核。

套接字接口的实现方式还可以是将套接字接口实现为一个独立的进程，应用程序通过与这个进程通信来访问传输层服务。这种实现方式的优点是灵活性好，缺点是效率低，因为每次通信都需要进行进程间通信。

套接字接口的实现方式还可以是将套接字接口实现为一个虚拟设备，应用程序通过读写这个设备来访问传输层服务。这种实现方式的优点是灵活性好，缺点是效率低，因为每次读写都需要进行系统调用。

套接字接口的实现方式还可以是将套接字接口实现为一个网络协议栈，应用程序通过与这个协议栈通信来访问传输层服务。这种实现方式的优点是灵活性好，缺点是效率低，因为每次通信都需要进行进程间通信

户空间

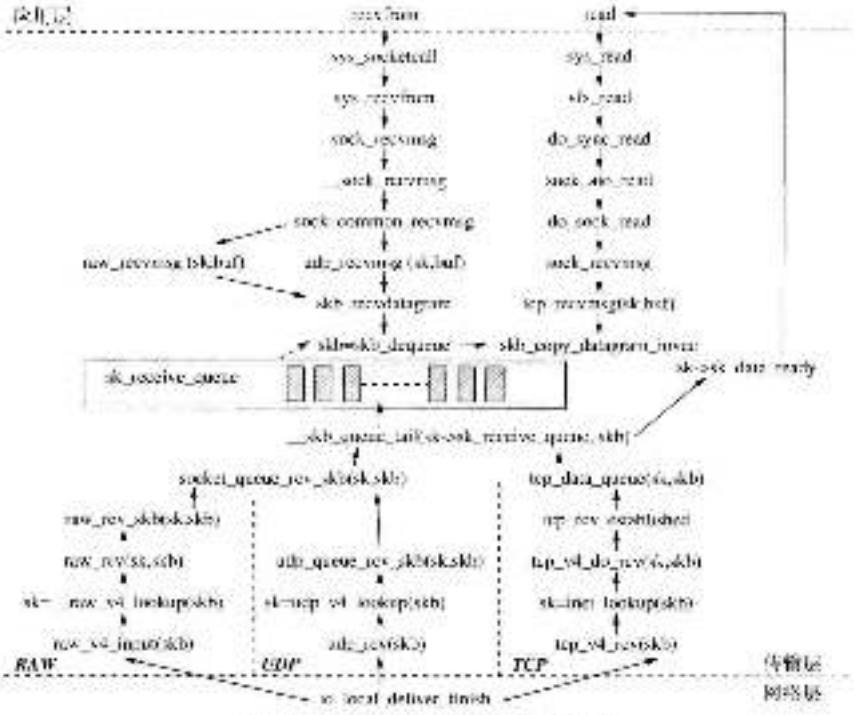


图 5.2 传输层中进入分组的调用图

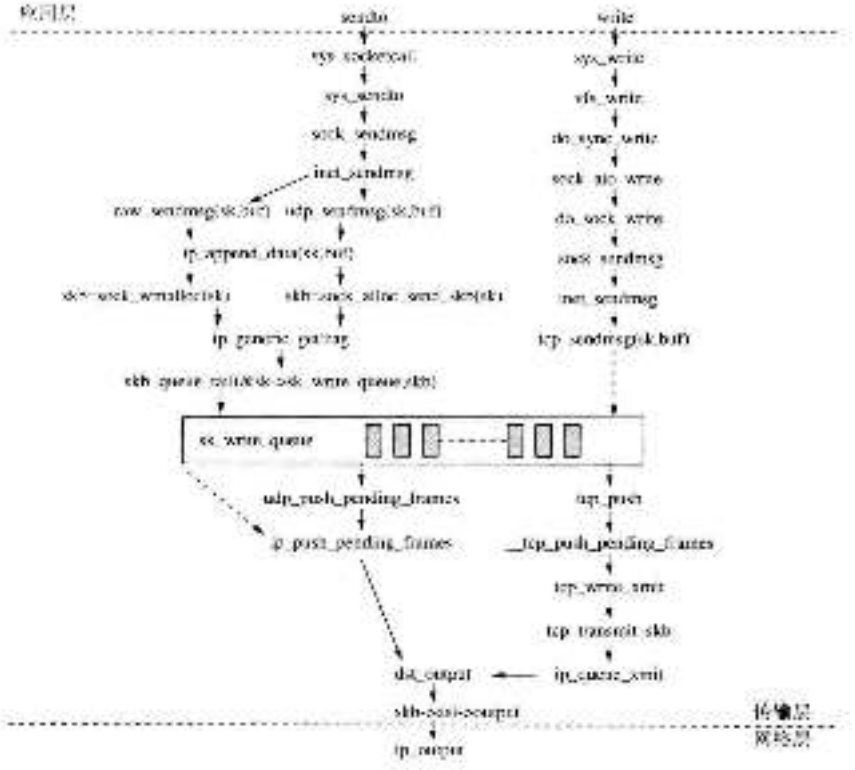


图 5-3 传输层中外出分组的调用图

内存中复制到 skb 缓存中。最后，将 skb 插入到 sock 结构的 sk_write_queue 中。另一方面，ip

push_pending_frame()重复地从队列中删除数据，然后将它们传发到IP层。同样，对于TCP套接字，使用tcp_sendmsg()和skb_add_data()分别从队列中消除队尾skb并将数据复制到内核内存中。如果写入的数据量大于无部skb提供的空间，那么由sk_stream_alloc_page()在内核空间内存中分配一个新的skb缓存。最后，调用ip_queue_xmit()通过ip_output()函数将数据从sk_write_queue转发到IP层。

练习

1. 利用如图5.3所示的调用图，我可以追踪udp_sendmsg()和tcp_sendmsg()，了解这些函数是如何实现的？

2. 解释在tcp_sendmsg()中两个大的"while"循环目的是什么。为什么这样的循环结构未出现在udp_sendmsg()中？

5.2 不可靠的无连接传输：UDP

用户数据报协议（UDP）是一个不可靠的无连接的传输协议，不提供可靠性和进程控制。这是一个无状态的协议，一个分段的发送或接收独立于其他任何一个分段。尽管UDP提供差错控制，但它并可选的。由于其简单而不重传的设计，许多可靠性不是那么重要的实时应用会采用RTP通过UDP传输实时或流式数据。近年来，对等应用使用UDP协议向对等发送大量的查询，然后再使用TCP与选定对象交换数据。UDP提供最简单的传输服务：1）进程到进程通信信道；2）每个分段的差错控制。

5.2.1 头部格式

UDP头部仅有两个功能：导址和错误检测。它包括四个字段：源和目的端口号、UDP长度和UDP校验和，如图5-4所示。为了让互联网中位于不同主机上的两个应用进程提供一个通信信道，每个进程应该与其本地主机上的一个唯一局部端口号绑定。虽然运行主机独立地处理端口与进程的绑定，但它经常使用的服务器进程（如WWW）与众所周知的周知端口号很有用。通过众所周知端口就可以访问它们的服务。但是，客户端进程的端口号是随机选择绑定的，不必是众所周知的。



图5-4 UDP数据报格式

源/目的端口号，在IP头部中与源和目的IP地址、协议标识符（指示TCP或UDP）结合起来，形成一个5元组的套接字对，总长为32×16×2+2+8=104位。由于IP地址是全局唯一的，而端口号是本地唯一的，因此5元组也唯一地标识一个进程到进程通信信道的流。换句话说，同一个置中的分组具有相同的5元组值。在IPv6分组中，IP头部中的"流标识符"字段是专为流量识别设计的。注意，套接字对是全双工的，这意味着可以同时在两个方向上通过套接字连接传输数据。在图5-1中，从应用进程API流出的分组从其源端口道向绑定到应用进程AP3的目的端口。主机1上应用进程API封装在同一个5元组字段的任何数据可以准确地传输到主机2上应用进程AP3。

UDP允许不同主机上的应用程序直接相互发送数据分段而不需要首先建立一条连接。UDP端口从本地应用进程接收分段，将它们打包为不大于64K字节的称为数据报的单元中，并填充16位源和目的端口号和数据报中其他UDP头部字段。每个数据报都是以独立的IP分组逐跳地转发到目的地，如第4章所述。

当包含UDP数据的IP分组到达目的地时，它们就被导向到与接收应用进程绑定的UDP端口。

5.2.2 差错控制：每个分段的校验和

除了端口号外，为检查每个数据报的完整性，UDP头部还提供一个16位的校验和字段，如图5.4所示。由于UDP数据报的校验和计算是可选的，所以通过将校验和字段设置为0就可以禁止校验和检查。发送者生成校验和值并将它放入校验和字段，最终由接收者进行验证。为了确保每一个接收到的数据报和发送者发送的数据报一样，接收者对接收到的数据报重新计算校验和并验证结果是否与存储在UDP校验和字段中的值相匹配。UDP接收者将丢弃校验和字段与计算结果不匹配的数据报。这种机制保证了每个分段数据的完整性，但不能保证每个分段数据的可靠性。

UDP校验和字段存储在头部和负载中所有16位字段和的1的补码。它的计算与第4章中讨论过的IP校验和计算相类似。如果UDP数据报包含一个奇数字节的校验和，那么最后一个字节的后面位填充0以便形成一个用于校验和计算的16位字。注意，填充并非作为数据报的一部分传输，因为在接收者的校验和验证过程中相同的填充过程。校验和也包括一个96位的伪首部，由IP头部的4个字段组成：源IP地址、目的IP地址、协议和长度。包括伪首部的校验和使接收者能够检测到带有不正确发送、协议或长度信息的数据报。关于校验计算结果为0的情况，按将0xFFFF放入到该字段中。开源实现5.2详细介绍了校验和的计算过程。

UDP校验和，虽然是可选的，但值得推荐，因为某些链路协议不进行差错控制。当在IPv6上实现UDP时，UDP上的校验和就是强制性的，因为IPv6完全不提供校验和。仅对于某些实时应用才会省略UDP校验和，因为这里应用进程之间的延误和抖动比差错控制更重要。

开源实现5.2：UDP和TCP校验和

概述

在Linux 2.6中的校验和计算流程图与IP校验和都可以通过源源源源 tcp_ipv4.c 中的 tcp_v4_send_check()函数学习。UDP校验和的流程图与TCP中的完全一样。

数据结构

skb结构中称为csum的字段用来存储一个sk_buff承载的应用数据的校验和。csum的定义可以在 include/linux/skbuff.h 中找到。当有一个分段将发送时，在skb->csum中的值会和分组头部一起传输到校验和函数以使计算分组的最终校验和。

算法实现

图5.5列出了在 tcp_v4_send_check()中的部分代码。应用数据首先计算校验和进入skb->csum，然后通过函数调用csum_partial()，skb->csum将被计算校验和利用指针th索引的传输层头部。计算结果将被执行校验和，通过共享了csum_tcpudp_nofold()的tcp_v4_check()与IP头部中的源和目的IP地址进行校验和计算。最后的结果存储在TCP/UDP校验和字段中。另一方面，IP校验和是与IP头部独立计算的，可以通过在 net/ipv4/ip_input.c 中搜索词语"iph->check"找到。

```
th->check = tcp_v4_check(len, inet->saddr, inet->daddr,
    csum_partial((char *)th,
    th->doff << 2,
    skb->csum));
```

图5.5 TCP/IP校验和程序代码部分代码

框图

校验和计算的流程图按照前面所述绘制，如图5-6所示。我们从图5.5中可以总结出以下几点发现：1）伪首部校验和是从应用数据的校验和计算出来的；2）IP校验和并不包括IP有效载荷。在图5-6中，D表示指向应用数据的指针，len表示应用数据的长度，T表示指向传输层头部（TCP或UDP）的指针，len表示传输层头部的长度，len表示分段（包括分段的头部）的长度，iph表示指向IP头部的指针，SA表示源IP地址，DA表示目的IP地址。

练习

如果查看csum在sk_buff中的定义，你会发现它的4字节内存空间是与另外两个变量：csum_start和csum_offset共享的。解释这两个变量的用法，为什么这两个变量要与csum共享内存空间。



图 5-6 在 Linux 2.6 中的 TCP/IP 头部校验和计算

5.2.3 承载单播/组播实时流量

由于其简洁性，UDP 是一种适合于单播或组播实时流量的载体。这是因为实时流量具有以下特点：1）它不需要每个流（per-flow）的可靠性（重发丢弃的实时分组可能没有意义，因为分组不可能按时到达）；2）它的传送率（带宽）主要取决于所选的编解码器，不大可能进行流量控制。这两种特性简化了用于实时流量的传输层以便仅提供寻址服务。

然而，除了基本的进程到进程通信服务外，实时应用也需要额外的服务，其中包括音频和视频图像流之间的同步、数据压缩和解压缩的信息以及路径质量统计。这些服务大多数是由 RTP 提供，因此，在 5.5 节我们将探讨建立在 UDP 上的 RTP 设计。

5.3 可靠的面向连接的传输：TCP

目前，大多数的网络应用使用传输控制协议（TCP）进行通信，因为它提供可靠、按顺序的数据发送。而且，TCP 自动调整它的发送速率以适应网络拥塞或者接收机接收能力的变化。

TCP 首先提供：1）进程到进程通信信道的寻址；2）每个分段的差错控制；3）每个流的可靠性；4）每个流的流量控制与拥塞控制。在 TCP 中的信道寻址和每个分段的差错控制与 UDP 协议中的一样，因为后者的两个目标是在每个流的基础上。所以我们在 5.3.1 节首先讨论 TCP 流如何建立和解除，然后在 5.3.2 节中说明 TCP 的可靠性控制。TCP 流量控制和拥塞控制的介绍分别在 5.3.4 节和 5.3.3 节中给出。然后在 5.3.5 节中阐述 TCP 头部格式。TCP 定时器管理问题将在 5.3.6 节讨论。最后，在 5.3.7 节中讨论 TCP 的性能问题及其功能增强。

5.3.1 连接管理

连接管理涉及端到端连接的建立和终止过程。与 UDP 一样，一个 TCP 连接唯一地由 5 元组标识：源/目的 IP 位址，源/目的端口号和协议标识符。建立和终止一条 TCP 连接类似于日常生活中通过电话与他人交谈。为了通过电话与他人交谈，我们拿起电话，然后再选择拨打的电话受话人的电话号码（IP 地址）和分机号（端口号）。然后，我们拨打电话受话人（发出连接请求），等待回应（连接建立），并开始说话（传输数据）。最后，我们说再见，挂掉电话（断开连接）。

通过互联网建立一条连接并不像听起来那么容易，因为事实上互联网偶尔会丢弃、存储和重复分组。在互联网上，分组以“存储转发”方式发送到目的地，也就是说，中间路由器首先存储接收到的分组，然后把它们转发到目的地或者下一跳。在互联网上存储分组导致了延迟和重复，可能会将发送者或接收者弄混，如果分组能够在网络中永远存在，那么为了解决歧义性就特别复杂。因此 TCP 决定将分组的最大生存时间限制为 120s。在此选择下，TCP 使用 Tomlinson 1975 年建议的三次握手协议，解决由延迟的重复分组造成的歧义性状况。

连接建立/终止：三次握手协议

在连接启动时，客户机和服务器进程都通机选择它们的初始序列号（ISN）以减少由于延迟重复分组引入的歧义性影响效应。当客户机进程想要与服务器进程建立连接时，如图 5-7a 所示，它发送一个 SYN 分段指定：1）客户机想连接的服务器端口；2）从客户机发送的数据分段的 SYN。服务器进程

用一个带有（ACK + SYN）的分段响应 SYN 分段：1）确认请求；2）宣告从服务器进程发送的数据分段的 ISN。最后，客户机进程必须确认来自服务器进程的 SYN 以便确认连接的建立。注意，每个方向的 ISN、序列号和确认号必须遵守图 5-7a 中的语义描述。这种协议称为三次握手协议。

TCP 连接终止需要四个阶段而不是三个分段，如图 5-7b 所示，它是一种对每个方向的两次握手，由一个 FIN 分段紧跟着一个 FIN 分段的 ACK。TCP 连接是一个全双工数据。从客户机到服务器或者从服务器到客户机都是彼此独立的。由于用一个 FIN 分段关闭一个方向，不会影响另外一个方向，所以为一个方向的关闭连续需使用一个 FIN 分段。注意，通过 3 次握手可能关闭一个连接。以最快速，客户机发送一个 FIN，服务器回送一个 FIN + ACK（只是将两个分段合并为一个），最终客户机回送一个 ACK。

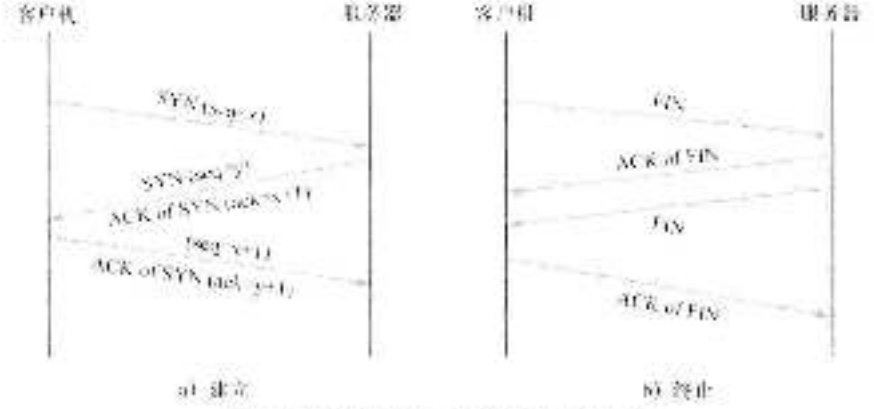


图 5-7 TCP 连接建立和终止的握手协议

发送第一个 SYN 初始化 TCP 连接的一方执行主动打开，而监听端口接收到达的连接请求的对等方执行被动打开。同样，发送第一个 FIN 终止 TCP 连接的一方执行主动关闭，而其对等方执行被动关闭。它们之间的详细区别可由接下来描述的 TCP 状态迁移图来说明。

TCP 状态转换

TCP 连接在其生存时间中通过一系列的状态变化。这里 TCP 连接有 11 种可能的状态，分别是：LISTEN、SYN-SENT、SYN-RECEIVED、ESTABLISHED、FIN-WAIT-1、FIN-WAIT-2、CLOSE-WAIT、CLOSING、LAST-ACK、TIME-WAIT 和虚构的状态 CLOSED。CLOSED 是虚构的，因为它表示 TCP 连接终止的状态。TCP 状态的含义是：

- LISTEN：等待来自任何远程 TCP 客户机的连接请求。
- SYN-SENT：已经发送一个连接请求后，等待一个匹配的连接请求。
- SYN-RECEIVED：已经接收和发送一次连接请求后，等待连接请求的确认。
- ESTABLISHED：一个打开的连接，数据可以在两个方向发送。连接的数据传输阶段的正常状态。
- FIN-WAIT-1：等待来自远程 TCP 的连接终止请求，或者以前发送的连接终止请求的确认。
- FIN-WAIT-2：等待来自远程 TCP 的连接终止请求。
- CLOSE-WAIT：等待来自本地用户的连接终止请求。
- CLOSING：等待对来自远程 TCP 连接终止请求的确认。
- LAST-ACK：等待对上次发送到远程 TCP 的连接终止请求的确认。
- TIME-WAIT：在转换到 CLOSED 状态之前等待足够的时间以确保远程 TCP 收到它的最后 ACK。

上面 RFC 793 中的定义，TCP 的工作是通过运行一个如图 5-8 所示的状态机。客户机和服务器进程的行为都遵循这种状态转换图。图 5-8 中粗体箭头和虚线箭头分别对应客户机和服务器进程的正常状态转换。在图 5-8 中整个状态转换可以分为三个阶段：连接建立、数据传输和连接终止。当客户机和服务器都迁移到 ESTABLISHED 状态时，TCP 连接进入数据传输阶段。在数据传输阶段，客户机可以向服务器发送一个服务请求，一旦请求得到许可，双方可以通过 TCP 连接相互发送数据。在数据服务中，服务器进程最常见的是充当 TCP 发送者，将请求的数据文件发送到客户机。

正常连接建立和连接终止的状态转换如图5-9所示，带有表示由客户机和服务器输入的标签。因为双方可能同时向对方发送SYN来建立TCP连接，虽然这种可能性很小，但在图5-8中也要考虑这种"同时打开"的状态转换。图5-10a显示了同时打开的状态转换。同样，在TCP中它允许双方关闭，这称为"同时关闭"。这种情况下的状态转换如图5-10b所示。

另一方面，在某些异常的情况下，包括丢失SYN、丢失SYN/ACK、在连接建立期间丢失ACK，分别在图5-11a、b、c中显示。丢失的分段会触发客户机连接超时，然后再返回到关闭状态。如图5-11a、b所示。但是，在服务器上的连接超时如图5-11b、c所示，会导致返回到CLOSED状态，并且将发送一个RST分段重置客户机的状态。

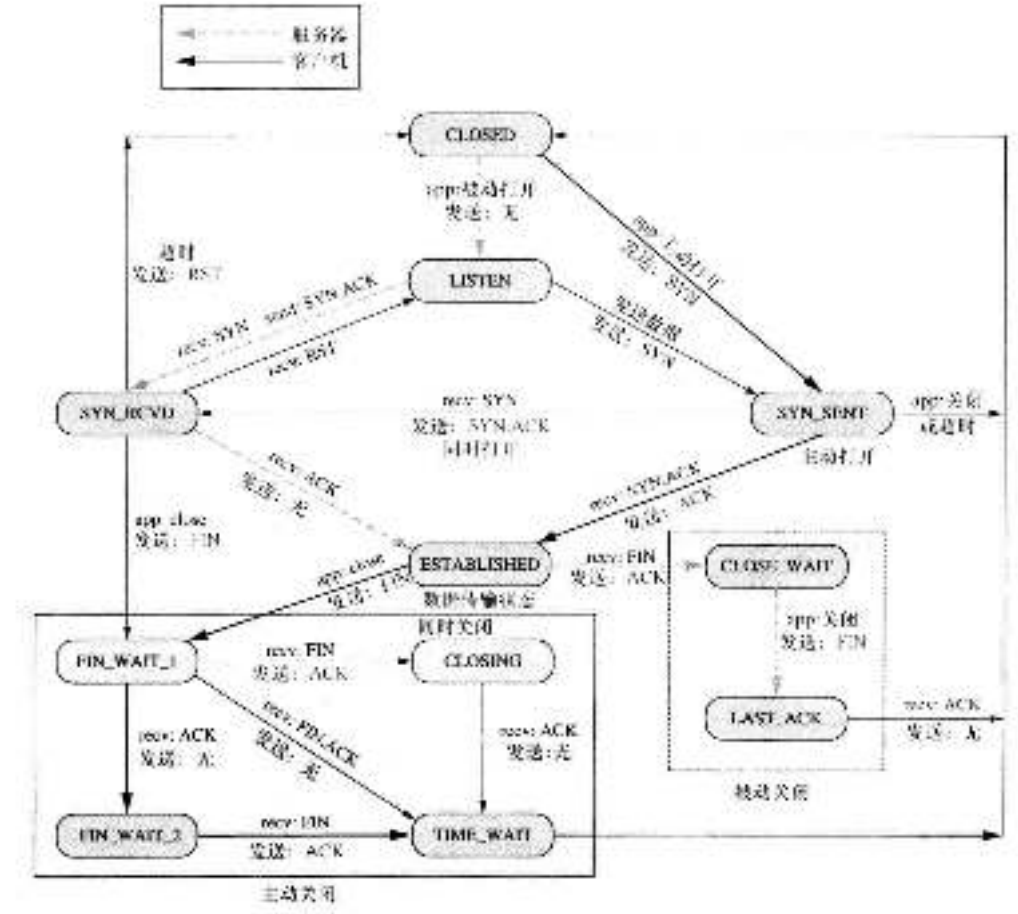


图5-8 TCP状态转换图

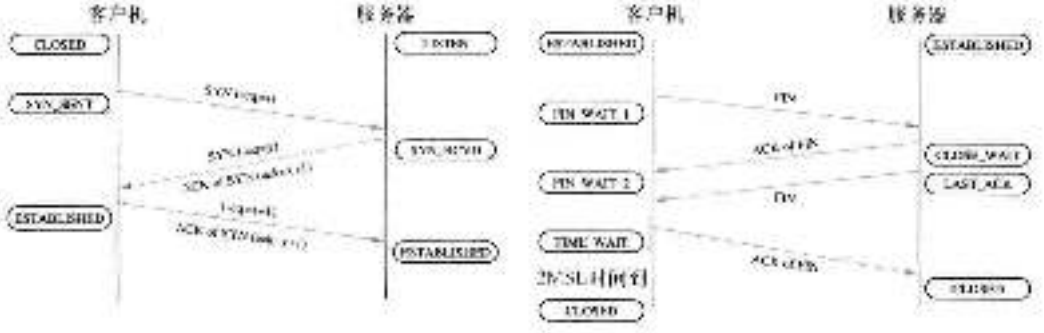
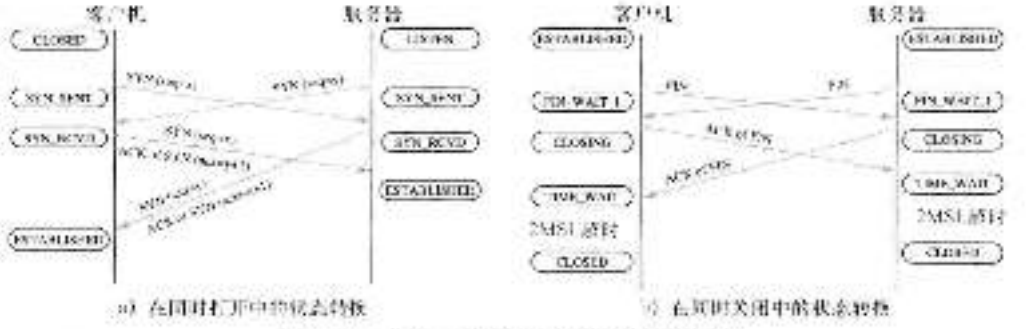
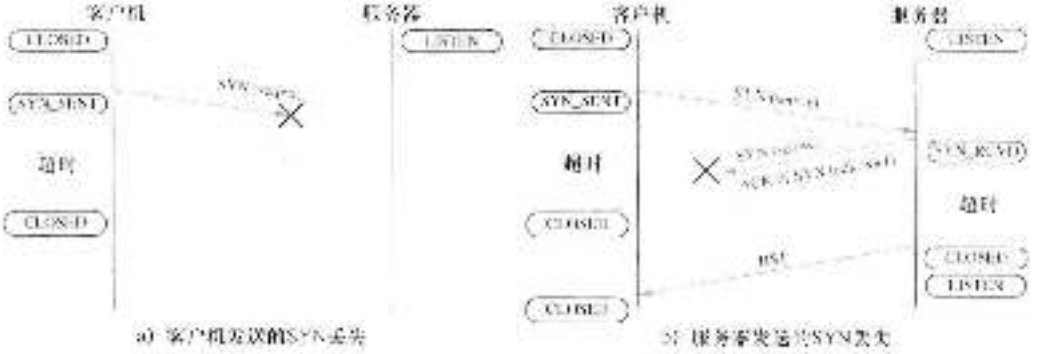


图5-9 在连接建立和终止中的状态转换

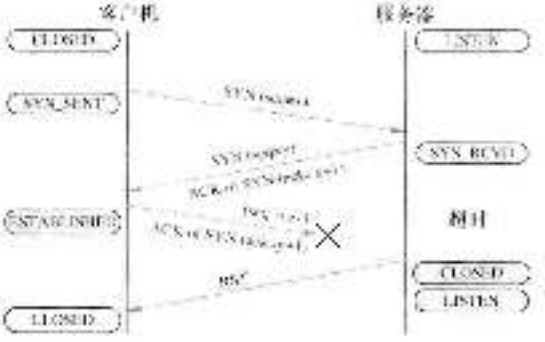


a) 两端同时打开的状态转换　　b) 两端同时关闭中的状态转换

图 5-10　同时打开和同时关闭时的状态转换



a) 客户机发送的SYN丢失　　b) 服务器发送的SYN丢失



c) 客户机发送的SYN ACK丢失

图 5-11　在连接建立中有分组丢失的状态转换

也存在一些异常的情况，例如，在连接维持期间的半打开连接。当 TCP 连接的一端的主机崩溃时，该 TCP 连接称为半打开。如果剩下的一端空闲，那么连接会无限地保持半打开状态。5.3.6 节中介绍的 keepalive（保活）定时器就能解决这个问题。

5.3.2　数据传输的可靠性

TCP 为每个分段的差错控制使用校验和，为每个流的可靠性控制使用确认序列号。这里描述它们的目标和解决方案的不同。

每个分段的差错控制：校验和

正如 5.2 节所述，TCP 校验和的计算与 UDP 的完全一样。它也包括一些 IP 头部字段以确保分组到达正确的目的地。与前 UDP 校验和是可选的，但 TCP 校验和是强制性的。与前两种协议都为数据完整性提供校验和字段，例如 3.1 节所述，与以太网中使用的 32 位循环冗余校验和相比，这里讲述的校验和是一个相对较弱的检查。

每个流的可靠性：序列号和确认

Per-segment 每个分段的校验和不足以保证可靠地和按序地发送全部分组化的数据流，分组化的数据流将通过进程到进程信道按顺序发送到目的地。由于分组的数据偶尔也会在互联网上丢失，所以就必须有一种机制来予传丢失的分组。而且，因为按顺序发送的分组由于互联网路由的无状态性质可能会不按顺序接收，所以必须提供另外一种机制予重排列无序的分组。这两种机制分别依赖于确认（ACK）和序列号以便提供每个流的可靠性。

从理论上，给每字节数据分配一个序列号。然后，一个分段的序列号表示它的第一个数据字节的序列号，存储在 TCP 头部的 32 位序列号字段。TCP 发送者进行编号并跟踪已经发送的数据字节并等待它们的确认。一旦收到一个数据分段，TCP 接收者就回应一个 ACK 分段，该分段携带的确认号显示：1）期望接收的下一个数据分段的序列号；2）所有在指定 ACK 号码以前的数据字节已经成功接收。例如，通过应答一个 ACK = x，TCP 接收者可以确认一个成功接收的分段，这里 x 表示："所有在 x 之前的数据字节已经收到。接下来期望接收的分段序列号为 x，请发送给我。"

有两种可能的 ACK 类型：选择性 ACK 和累积 ACK。选择性 ACK 指示，接收机已经收到了一个分段其序列号等于指定的 ACK 号码。累计 ACK 指示在指定的 ACK 之前的所有八位字节数据已经收到。因为不对必链路组管理，这样拥塞就可能发生在从与客户机（接收器端）到服务器（发送方端）的逆向路径上，ACK 就有可能让数据更经常丢失。因此，TCP 使用累积 ACK 利用后续的 ACK 对丢失的 ACK 进行补偿。

异常情况：数据丢失，ACK 丢失，延迟和失去顺序

图 5-12 描述了可能发生在 TCP 传输期间的四个异常例子。在数据丢失的情况下，在重传超时后发送者就会察觉到这种丢失，然后就传输丢失的分段，如图 5-12a 所示。另一方面，大的传播延迟可能会导致过早超时，导致不必要的重传。如图 5-12b 所示，接收者将重传分组仅作为重复数据而丢弃。在

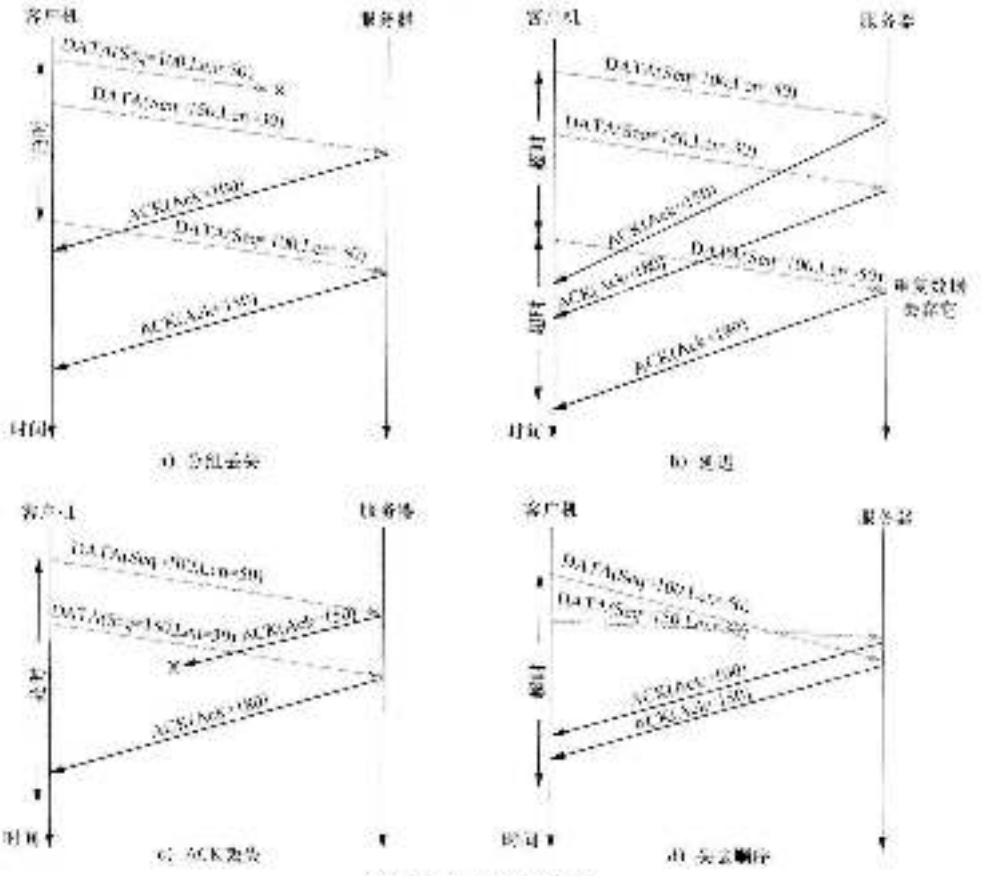


图 5-12 TCP 的可靠性

这种情况下，可靠性虽然仍能得到保证，但如果这种情况经常发生就会严重浪费带宽。因此，如何估计一个适当的重传超时是非常重要的，这个估计在5.3.6节进行解释。图5-12c说明了在TCP中使用累积ACK的优点。该里ACK丢失不会造成任何不必要的数据重传，因为随后的ACK会重复丢失ACK中的确认信息，即ACK=180重复丢失的ACK=150中的信息。当接收到的数据分段丢失乱序时，使用累积ACK也会导致出现一种非常有趣的情况。接收者一旦收到下一个数据分段就回复多足ACK，如图5-12d所示，就好像接收者丢失了数据分段一样。从图5-12中，我们可以理解使用累积ACK和重传超时进行确认，TCP可以实现可靠的传输。

5.3.3 TCP流量控制

互联网上的延迟分布是扩散的，TCP发送者需要足够智能和自适应才能最大化性能，同时礼貌地对待其接收者缓冲空间和其他发送者共享的网络资源。TCP采用基于窗口的流量控制与拥塞控制机制来确定在各种条件下应该多快地发送。通过流量控制，TCP发送方就可以知道有多少带宽可以消耗而不会造成接收者缓冲区的溢出。同样，通过拥塞控制，TCP发送者就能避免增加全球共享网络资源的负担。本节描述TCP流量控制，而将TCP拥塞控制留到5.3.4节讲解。

滑动窗口流量控制

基于窗口的流量控制利用滑动窗口机制，目的在为了提高数据传输吞吐量。发送者保持一个窗口序列号，称为发送窗口，由一个初始序列号和结束的序列号来描述。只有序列号在该发送窗口内的数据分段才可以发送。数据分段发送但没有确认过的放在一个重传缓冲区中。当带有最始序列号的数据分段被确认时，该发送窗口将滑动到待发送的数据。

图5-13显示了发送者滑动窗口的伪代码。图5-14也显示了一个滑动窗口的例子。为了清晰起见，我们假定所有的数据分段都具有相同的大小。在图5-14中，为了按序发送分段的字节流，窗口只需要从左向右滑动。为了控制传输中未应答分段的最大量，窗口必须也缩短。正如我们稍后将要看到的，当数据分段流向目的地时，对应的ACK分段反向流到发送者以触发窗口的滑动。当窗口尚未覆盖传送的分段时，这些分段就被发送到网络管道上。在最初的情况下，如图5-14a所示，滑动窗口的起始是从分段4～分段8，即这些分段已经发送过。发送者接收到ACK（Ack=5），表示接收者已经成功收到分段4，滑动窗口中的第一个分段。因此，发送者将滑动窗口1次后滑动一个分段，如图5-14b所示。同样，图5-14c和d说明当发送者分别接收到ACK（Ack=6）和ACK（Ack=7）时窗口的滑动情况。在正常情况下，当发送者接收到一个按顺序的ACK时，它就将窗口滑动一个分段。

```
SWS：发送窗口尺寸。
n：当前序列号，即下一个将要传输的数据分组。
LAR：接收上次的确认。
If 如果发送者有数据要发送
        在最后确认LAR之前，最多传输SWS个分组，
        即只要n<LAR+SWS，就可以发送n个分组
endif
If 一个ACK到达
        如果它的ack num>LAR，则将LAR设置成ack num
endif
```

图5-13 发送者的滑动窗口伪代码

现在我们观察分组不按序到达接收者的另外情况，如图5-15所示。在这种情况下，接收者首先接收到DATA 5、DATA 6，然后才收到DATA 4。由于TCP采用累计确认，所以发送者一旦收到DATA 5就接收来自接收者的第一个重复ACK（Ack=4），见图5-15b。此时窗口不能滑动。一旦当发送者收到DATA 6后接收来自接收者发送的第二个重复ACK（Ack=4），窗口仍然不能滑动，见图5-15c。当发送者收到接收者在接到延迟的DATA 4后发送的ACK（Ack=7）后，就将窗口滑动三个数据分段。

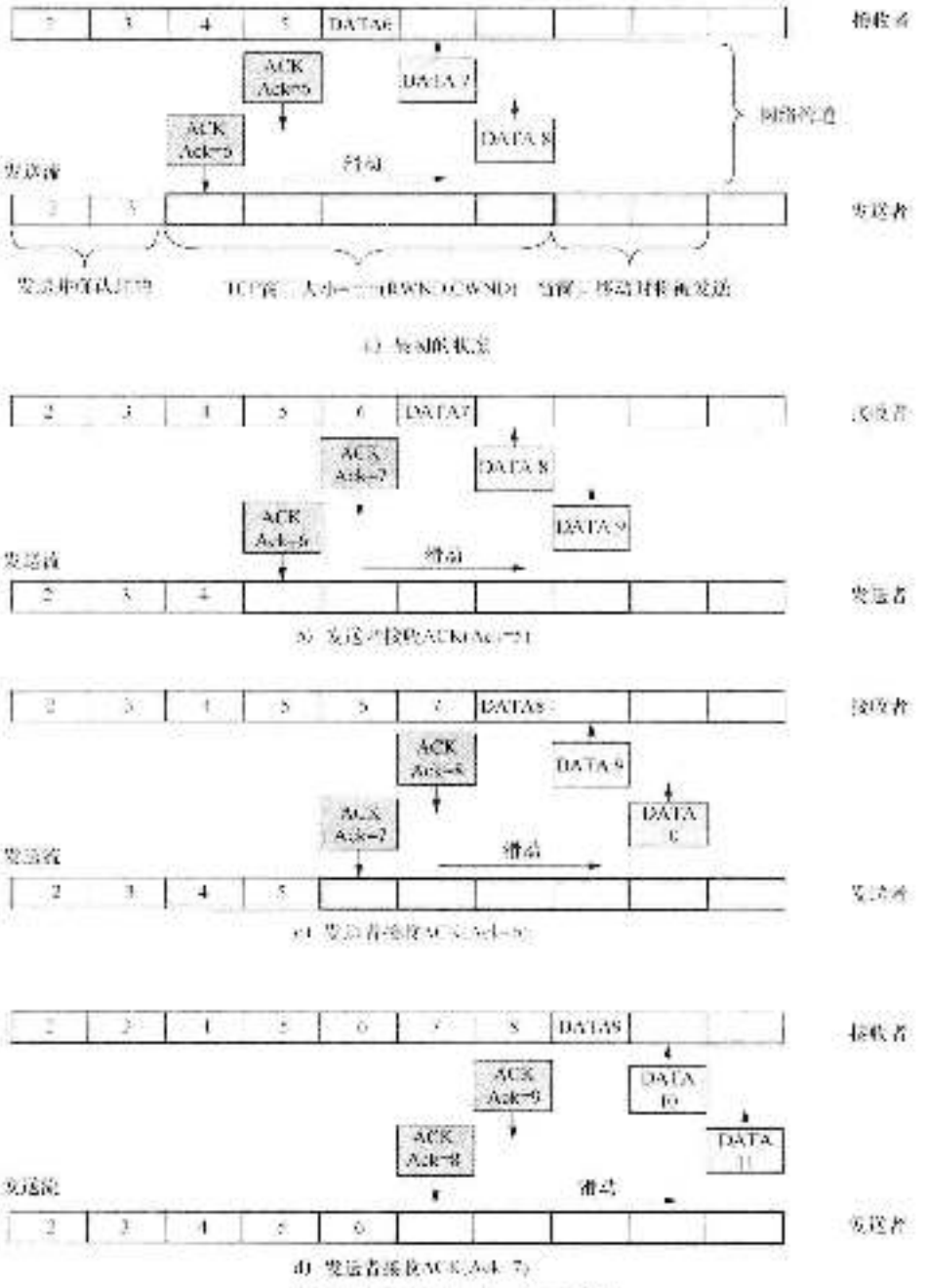
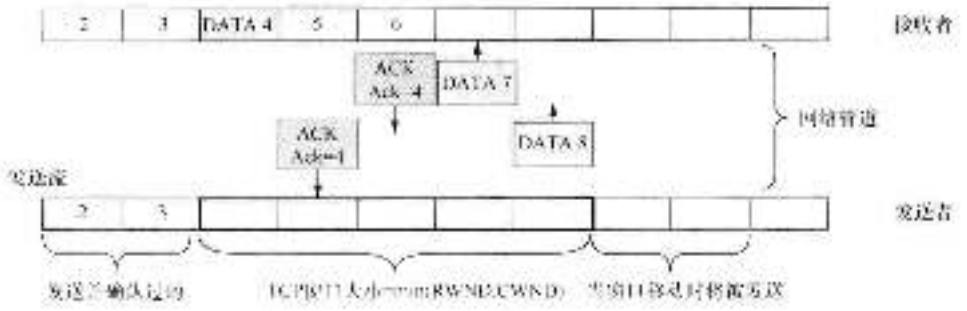


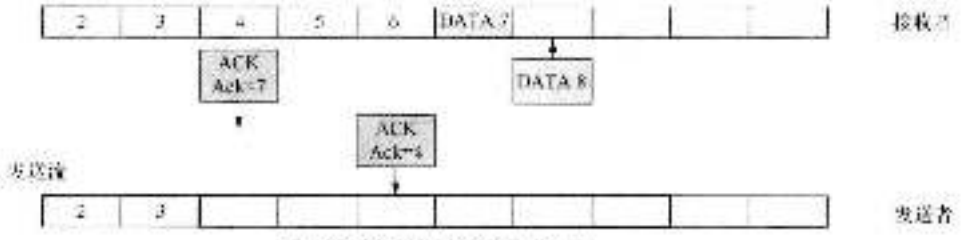
图5-14 TCP滑动窗口的可视化

窗口大小的扩大和缩小

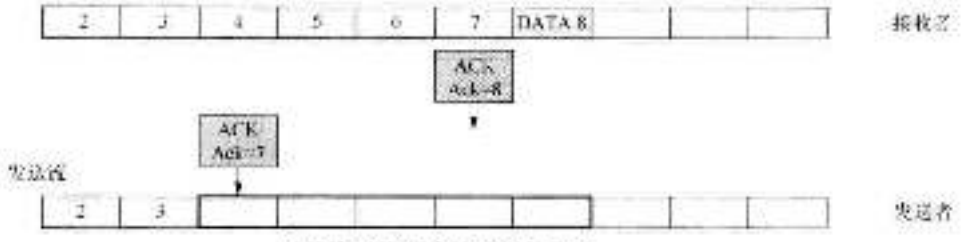
滑动窗口坐标控制中的另一个重要问题是窗口大小。窗口大小取决于两个窗口值中的最小值：接收窗口（RWND）和拥塞窗口（CWND），如图5-16所示。TCP发送者试图同时考虑接收者的能力（RWND）和网络容量（CWND），将其发送速率限制为min（RWND，CWND）。RWND由接收者通告，而CWND由发送者计算，这将在5.3.4节中研究。注意窗口的大小实际上是以字节为单位而不是以分段为单位计算的。TCP接收者在TCP报头的16位窗口大小中通告缓冲区(1)中的字节数。只有当已经确认了分段即当设置了ACK控制位时，才使用通告。另一方面，TCP发送者还给出了允许在网络中传输的字节数，以最大分段大小（MSS）为单位。



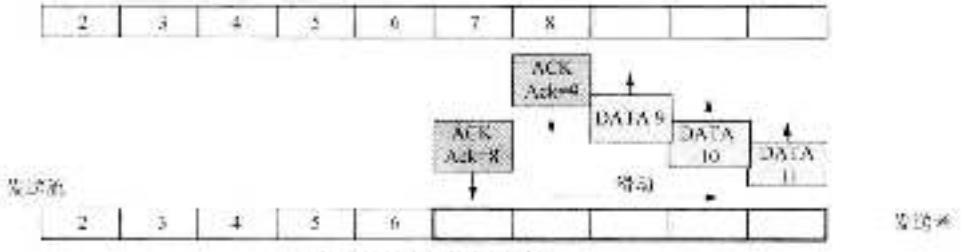
a) 初始的状态



b) 发送方收到数据5的确认 (Ack=5)



c) 发送者发送数据6且确认 (Ack=7)



d) 发送者接收确认 (Ack=7)

图 5-15 当数字分组不按顺序发生，TCP 滑动窗口的例子

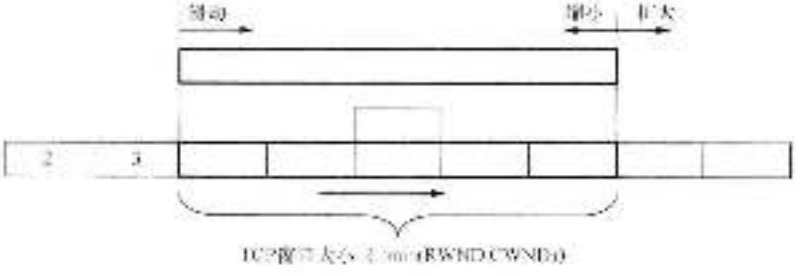


图 5-16 窗口的大小和滑动

开源实现 5.3：TCP 滑动窗口流量控制

概述

Linux 2.6 内核在 tcp_output.c 中实现 tcp_write_xmit() 函数，将分组写入到网络上。函数检查是通过调用 tcp_snd_test() 函数来检查是否有要发送的内容，这里内核会根据滑动窗口的概念进行多次测试。

算法实现

第一，在 tcp_snd_test() 中调用三个检查函数：tcp_cwnd_test()、tcp_snd_wnd_test() 和 tcp_nagle_test()。在 tcp_cwnd_test() 中，通过评估条件 tcp_packets_in_flight() < tp->snd_cwnd，内核判断是否有未经确认的分段号（包括正常和重传的分段）超出了当前网络容量（cwnd）。第二，在 tcp_snd_wnd_test() 中，内核决定是否最后发出的分段已经超过接收器缓冲区的限制，通过调用函数 after(TCP_SKB(skb)->end_seq, tp->snd_una + tp->snd_wnd)。after(x,y) 函数是一个布尔函数对应于"x>y"。如果最后发送的分段(end_seq)已经超出了选择未确认的入位位组(snd_una)加上窗口大小(snd_wnd)，那么发送者就应当停止发送。第三，在 tcp_nagle_test() 中，内核通过 tcp_nagle_check() 执行将在 5.3.7 节讨论的 Nagle 测试。只有分段通过了这些检查，内核才能调用 tcp_transmit_skb() 函数发送窗口内更多的分段。从这实现中，我们可以观察到的另一个有趣的现象就是 Linux 2.6 内核使用最细粒度发送窗口大小内的分段。也就是说，它在通过了所有首面的检查后只发送一个分段，有下一个分段会发送并还要重复所有的测试。在发送分段的过程中，如果有任何窗口扩大或缩小，内核就可以在即时改变网络上允许的分段数量。但是，这样做会引起大量的开销，因为它每次仅发送一个分段。

练习

在 tcp_snd_test() 中，在上述三个检查函数之前需要调用另外一个函数 tcp_init_tso_segs()。解释该函数的作用是什么？

5.3.4 TCP 拥塞控制

TCP 发送者还通过检测数据分段丢失事件来推断网络是否拥塞。一个丢失事件后，发送者会相应地减慢其传输速率并保持在将会触发丢失事件的速率以下。这个过程称为拥塞控制，旨在取得资源有效利用率的同时避免网络拥塞。通常，TCP 拥塞控制的思想就是为每个 TCP 头部确定从发送者到接收者路由路径上的可用带宽，所以它知道发送多少分段是安全的。

从基本的 TCP、Tahoe、Reno 到 NewReno、SACK/FACK 和 Vegas

TCP 协议的发展经过二十年了，已经提出了许多版本的 TCP 以提高传输性能。第 1 版于 1981 年由 RFC 793 标准化，其中定义了 TCP 的基本结构：即基于窗口的流量控制和粗粒度的重传定时器。注意，RFC 793 并没有定义拥塞控制机制，因为当时所使用的电传网络设备具有句链路的流量控制，而且互联网流量也比现在少得多。TCP 拥塞控制是由 Van Jacobson 在 20 世纪 80 年代末引入到互联网中的，大约是在 TCP/IP 协议族运行了 8 年后。此时，互联网已经开始经历拥塞崩溃，主机也开始以超出接收方通告窗口允许的速度向互联网发送分组，那么拥塞就会在某些路由器上发生，导致分组的丢失、主机超时和重传丢失的分组，造成更加严重的拥塞。因此，1988 年在 BSD 4.2 上发布的第 3 版，TCP Tahoe，增加了由 Van Jacobson 提出的拥塞避免和快速重传方案。第 3 版，TCP Reno 通过添加快速恢复而扩展了拥塞控制。RFC 2001 对 TCP Reno 进行了标准化，RFC 2581 对它进行了总结。到了 2000 年，TCP Reno 已经成为最流行的版本，但在最近的一份报告中，TCP NewReno 目前已经更加流行。

在 TCP Reno 中存在一些缺点，最引人注目的是多分组丢失（MPL）问题，由于多个分段在短时间间隔内丢失，所以 Reno 常常会导致超时和低的利用率。NewReno、SACK（选择性确认，定义在 RFC 1072 中），以及 Vegas（由 L. Brakmo 和 L. Peterson 于 1995 年提出）以三种不同的方法来解决这个问题。TCP FACK（前向确认）版本，进一步增强了 TCP SACK 版本。我们首先学习 TCP 拥塞控制的基本

版本，即 TCP Tahoe 和 TCP Reno。通过 NewReno、SACK、FACK、Vegas 的进一步增强在 5.3.7 节中介绍。

TCP Tahoe 拥塞控制

Tahoe 使用拥塞窗口（cwnd）控制在一个往返时间（RTT）内发送的数据量，使用最大窗口（mwnd）约束 cwnd 的最大值。Tahoe 以 snd_nxt、snd_una 的形式统计未经确认的数据量 awnd，这里 snd_nxt 和 snd_una 分别为未发送数据的序列号和未确认数据的序列号。当 awnd 小于 cwnd 时，发送方继续发送新的分组；否则，发送方停止发送。Tahoe 的控制方案可分为四种状态，其转换图如图 5-17 所示并且解释如下：

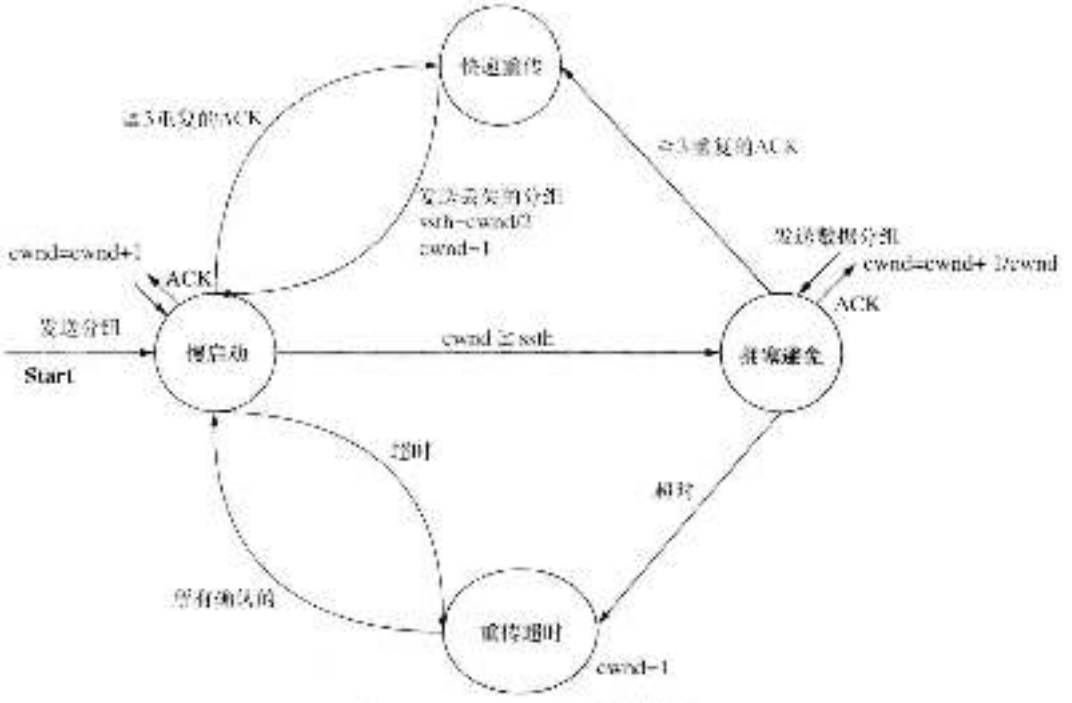


图 5-17 TCP Tahoe 拥塞控制算法

慢启动：慢启动首在短短的几个 RTT 时间内迅速探测到可用的带宽。当一个连接启动或发生重传超时后，通过将 cwnd 的初始值设置成一个分组开始慢启动状态，即 MSS。发送方通过每次只要收到一个 ACK 时就添加一个分组到 cwnd 中而呈指数式增加 cwnd。经过每个 RTT 后，如果所有的 ACK 正确及时地收到，就加倍 cwnd（1，2，4，8 等），如图 5-18 所示。因此，慢启动一点也不慢。TCP 发送方一直处在慢启动状态，直到它的 cwnd 达到慢启动的阈值 ssthresh（或图 5-17 中的 ssth）为止之后，它就进入拥塞避免状态。注意，当发起一次连接时，将 ssthresh 设置为 sathresh 的最大值（具体取决于存储 sathresh 的数据类型），所以不限制慢启动带宽探测。如果收到 3 个重复的 ACK，TCP 发送方就进入快速发送状态并且将 cwnd 重置为 1。如果重传超时前没有收到 ACK，那么就将 cwnd 重置为 1，TCP 发送方进入重传超时状态。

拥塞避免：拥塞避免的目的在于及早地探测可用带宽，但迅速地响应拥塞事件。它遵循加法增加乘法减少（AIMD）的原则。由于慢启动状态中的窗口大小呈指数增加，所以以比不断增加的速度发送分组就会很快导致网络拥塞。为了避免这种情况，当 cwnd 超过 ssthresh 时就启动拥塞避免状态。在这种状态下，一旦收到一个 ACK，cwnd 就增加 1/cwnd 个分组，从而使得窗口的大小呈线性地增长。因此，cwnd 通常每经过一个 RTT 时间会递增 1（每收到一个 ACK 就增加 1/cwnd），但如果收到三个重复的 ACK，就重置为 1，以便触发迅速传输状态。同样，重传超时将发 cwnd 的重置并切换到重传超时状态。图 5-19 描绘了加法增加行为。

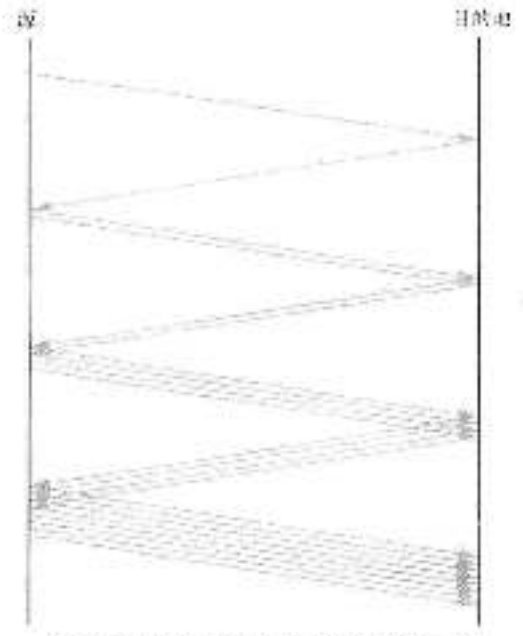


图 5-18 慢启动过程中传输分组的演示图

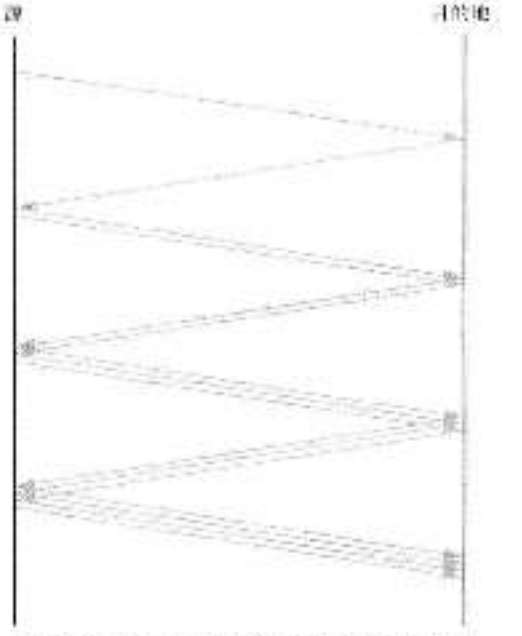


图 5-19 在拥塞避免期间传输分组的可视化

快速重传：快速重传的目的是立即传送丢失的分组而无需等待重传定时器超时。正如 5.3.2 节中所述，重复的 ACK 是由于丢失的分组（见图 5-12a），或者由于重复的分组（见图 5-12b），或者由于在接收方收到无序的分组（见图 5-12c）所造成的。在丢失数据分组的情况下，发送方应该重传。由于发送方不知道什么原因导致出现重复的 ACK，所以快速重传使用启发式算法：如果一行中接收到 3 个或更多的重复 ACK（3 个重复的 ACK（TDA））。那么 TCP 发送方就假定已经发生了分组丢失。发送方然后重传假定丢失的分组，而无需等待一个超时重传定时器超时。发送方传输了丢失的分组后，它根据 AIMD 就将它的 ssthresh 设置成等于 cwnd 值的一半，并再次启动将 cwnd 重置为 1 的慢启动状态。

重传超时：重传超时提供最后的和最慢的重传丢失分组的方法。发送方维护一个重传定时器，它用来检查确认的超时。它可以超过发送窗口的左边缘。如果发生超时，如快速重传状态中的处理，发送方将 ssthresh 减少为 cwnd/2，将 cwnd 重置为 1 并从慢启动状态重新启动。超时值很大程度上取决于 RTT 和 RTT 偏差。RTT 测量的波动越大，保持的超时值就越大，这样不会重传已经到达的分段。测量到的 RTT 越稳定，就可以设置更接近 RTT 的超时值。以便快速重传丢失的数据分段。TCP 实施由 Van Jacobson 在 1988 年建议的一种高度动态的算法，根据连续测量的 RTT 不断地调整超时间隔。这部分内容将在 5.3.6 节中讨论。

历史演变：TCP 版本的统计值

TCP NewReno 已逐步成为互联网中 TCP 的主要版本。国际计算机科学研究所（ICSI）的一份研究报告中指出所有成功探测的 35 242 台 Web 服务器中，使用 TCP NewReno 的服务器百分比从 2001 年的 35% 增加至 2004 年的 76%。支持 TCP SACK 服务器比例也从 2001 年的 40% 增加到 2004 年的 68%。此外，TCP NewReno 和 SACK 在多个流行的操作系统，包括 Linux、Windows XP 和 Solaris 中得到支持。与 NewReno 和 SACK 这类越多的应用相比，TCP Reno 和 Tahoe 的比例却分别下降到 5% 和 2%。TCP NewReno和 SACK 历史成功的原因之一，就是它们提供更高的吞吐量，为用户提供理想的易用，而不会恶化网络状况，这是网络管理员首要关注的问题。

开源实现 5.4：TCP 慢启动和拥塞避免

概述

Linux 2.6 内核 tcp_cong.c 中的慢启动和拥塞避免是通过 tcp_slow_start()、tcp_reno_cong_

avoid()和tcp_cong_avoid_ai()三个函数实现的。

数据结构

在上述三个函数中，tp是指向套接字结构tcp_sock的指针，其定义可以在linux/include/linux/tcp.h中找到。在tcp_sock中包含snd_cwnd,snd_ssthresh用于存储拥塞窗口和慢启动门限，snd_cwnd_cnt用于记录在接收到ACK分组就递增1/cwnd个分组的拥塞避免实现，snd_cwnd_clamp用于限制拥塞窗口（非标准）。

算法实现

Linux 2.6内核tcp_cong.c中的慢启动和拥塞避免在图5-20中进行了总结。注意，在拥塞避免中，将收到ACK后递增1/cwnd个分组简化为一次接收到cwnd个接收的所有ACK就增加一个全尺寸的分段（MSS字节）。如图5-20的行5～11所示。

```
1:  if (tp->snd_cwnd <= tp->snd_ssthresh) {              /* Slow start*/
2:      if (tp->snd_cwnd < tp->snd_cwnd_clamp)
3:          tp->snd_cwnd++;
4:  } else {
5:      if (tp->snd_cwnd_cnt >= tp->snd_cwnd) { /* Congestion
Avoidance*/
6:      if (tp->snd_cwnd < tp->snd_cwnd_clamp)
7:          tp->snd_cwnd++;
8:      tp->snd_cwnd_cnt=0;
9:      } else {
10:         tp->snd_cwnd_cnt++;
11:     }
12: }
```

图5-20 Linux 2.6中TCP慢启动和拥塞避免

练习

当前在tcp_cong.c中的实现提供了一种灵活的架构，允许用其他方法替代Reno慢启动和拥塞避免。

1. 解释如何实现上述功能。

2. 从内核源代码中找到一个通过这种体系结构更改Reno算法的例子。

TCP Reno拥塞控制

TCP Reno通过在分组丢失后将快速恢复状态引入到随后的恢复阶段扩展了Tahoe拥塞控制方案。Reno控制方案如图5-21所示。快速恢复将注意力集中在网络管道中保留足够未经确认的分组以使保留TCP的自同步行为。网络管道概念和TCP的自同步行为将在5.3.7节中详细介绍。当执行快速重传时，将ssthresh设置成cwnd的一半，然后将cwnd设置成ssthresh+3是由于三个重复的ACK。每一个收到的重复ACK表示另一个分组已退出网络管道，因此对于触发快速重传的三个重复的ACK，更正确的思路是将cwnd减3而不是加3，其中cwnd是在网络管道中未经确认的分组的数量。然而，在Reno中，cwnd的计算是snd_nxt－snd_una，在此状态下是固定不变的。因此Reno递增cwnd，而不是减少cwnd，以达到同样的目的。当收到重传分组的ACK时，将cwnd设置为ssthresh，发送方重新进入拥塞避免状态。换句话说，经过快速恢复后，cwnd重置为等于cwnd值的一半。

我们用一个例子来说明Tahoe和Reno之间的区别，分别在图5-22和图5-23中显示。在这两张图中，收到分组30的ACK后，发送方发送分组31～38。假设cwnd等于8个分组并且分组31在传输过程中丢失了。由于分组32、33、34、35、36、37和38已经接收到，所以接收方发送7个重复的ACK。Tahoe发送方在接收到第三个重复的ACK时察觉到分组31丢失了，然后立即将cwnd设置为一个分组，重传丢失的分组，返回到慢启动状态。在收到4个更多的重复ACK后，发送方维持cwnd值为1和ssthresh为8（39－31）。在接收到分组38的ACK后，发送方就可以发送新的分组39。

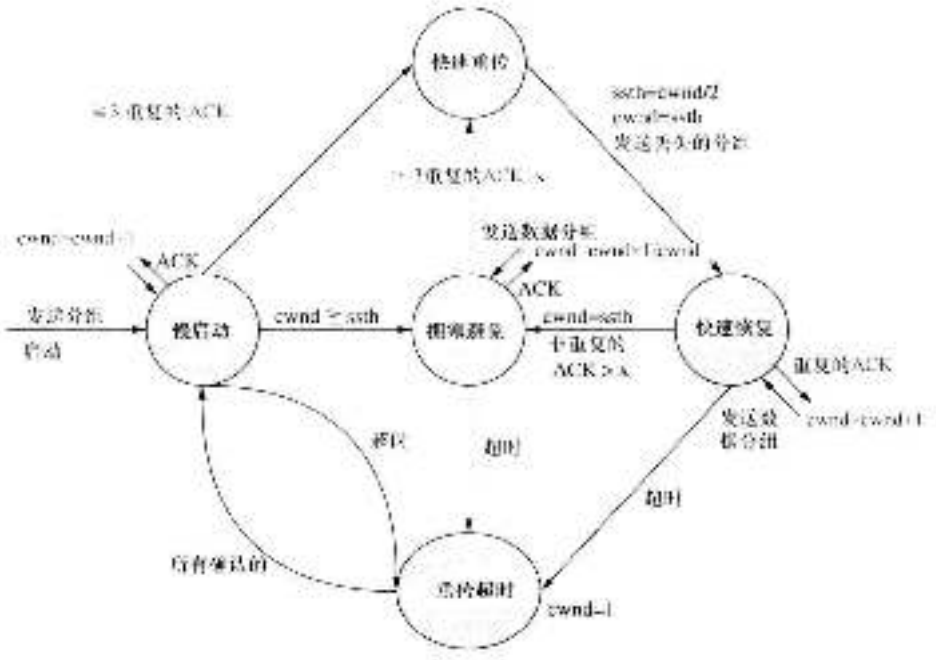


图 5-21 TCP Reno 拥塞控制算法

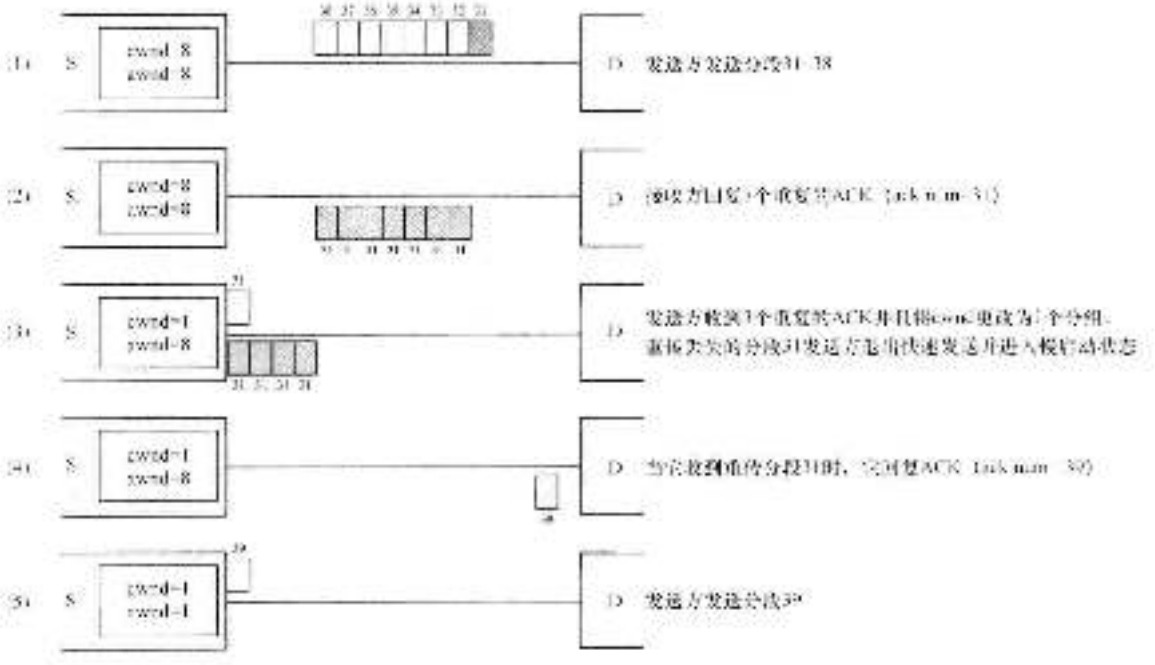


图 5-22 TCP Tahoe 拥塞控制的例子

另一方面，当 Reno 发送方检测到分组 31 丢失后，它立即将 cwnd 设置为 [8/2] + 3 分组，重传丢失的分组，并进入快速恢复状态。发送方收到 4 个更多的重复 ACK 后，发送方继续将 cwnd 加 4 并传送新的分组 39、40 和 41。在收到分组 38 的 ACK 后，发送方退出快速恢复，进入拥塞避免，并将 cwnd 设置为 4 个分组，这是老的 cwnd 值的一半。因为现在 swnd 等于 3（42 - 39），所以发送方可以发送新的分组 42。

比较图 5-22 和图 5-23 中的步骤（4），Tahoe 不能发送任何新的分组了，但 Reno 却可以。因此即

明显，在丢失分组后TCP Reno利用快速恢复得到更高效的传输。

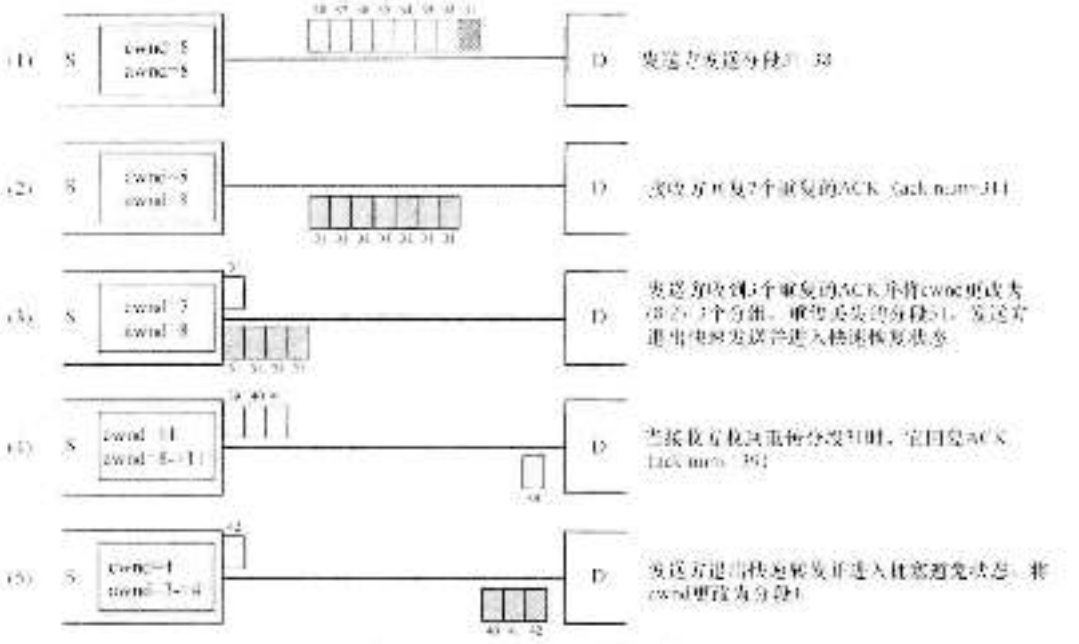


图 5-23 TCP Reno拥塞控制的例子

虽然Reno一直是最流行的TCP版本，但它仍存在多分组丢失的问题，降低了其性能。我们将在5.3.7节更进一步研究这个问题及其解决方案。

行动原则：TCP拥塞控制行为

Linux 2.6中已经实现了多种TCP版本，包括NewReno、SACK和FACK，这些将在5.3.7节中研究。然而，它们在一个分组丢失情况下的基本行为与Reno非常相似。图5-24显示了Linux 2.6 TCP拥塞控制的一个例子实现。它是通过处理发送窗口大小的内核日志和嗅探分组丢失产生的。

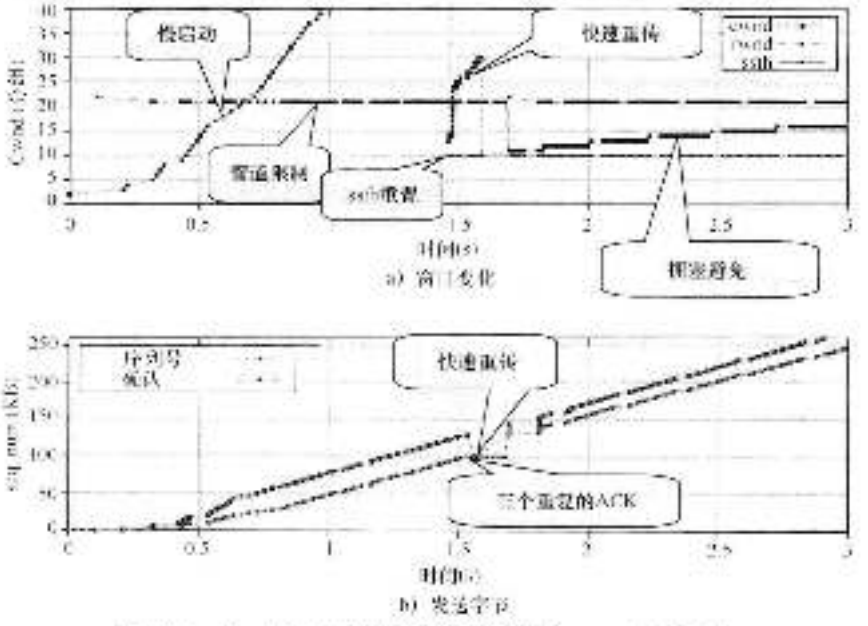


图 5-24 Linux 2.6中的慢启动和拥塞避免：cwnd与序列号

在图 5-24a 中，在拥塞发生前的 1.45s，cwnd 快速增长超出了在慢启动状态中图的边界。但是，注意 rwnd 几乎一直保持在 21 个分组，因此发送速率限制在 21 分组 KTT 个子/0.75s 和 1.45s 之间，如图 5-24b 所示。这是因为实际的发送窗口大小由 cwnd 和 rwnd 之间的最小值确定。因此，从 0.75～1.45s 的 cwnd 增长比从 0～0.75s 的增长要缓和很多，因为进入 ACK 的速率是固定的，在 0.75～1.45s，从 0.75～1.45s，全双工网络管道不断填充到 21 个分组，如果网络的前向路径和反向路径是对称的，那么其中大约一半的分组是 ACK。

当拥塞发生在 1.5s 时，三个重复的 ACK 就激发快速重传，重传丢失的分段。TCP 源由此就进入快速恢复状态，将 ssthresh 重置为 cwnd/2 = 10，而将 cwnd 重置为 ssthresh + 3。在快速恢复期间，当 TCP 发送方接收到一个重复的 ACK 时就将 cwnd 增加一个 MSS，以便保持足够的分段在传输中。当丢失的分段被恢复时，快速恢复状态就结束于 1.7s。在这一刻，将 cwnd 重置为 ssthresh（以前设置为 10）并更改为拥塞避免状态。在此之后，当收到滑动窗口的所有 ACK 时，cwnd 就会递增一个 MSS。

5.3.5 TCP 头部格式

在本节中，我们将学习到此为止还未见到的图 5-25 中 TCP 头部的其他字段。如 5.3.2 节中所指出的，一个 TCP 分段包含一个 16 位源端口号、一个 16 位目的端口号、一个 32 位序列号和一个 32 位的确认号。这些字段在 TCP 分段头部中携带以便在网络上传输。当 SYN 位未设置时，序列号就对应于该分段中的第一个数据八位位组。如果设置了 SYN 位，序列号就是初始序列号（ISN）并且第一个数据字节编号为 ISN + 1。如果设置了 ACK 控制位，那么确认字段包含 ACK 分段的发送方期望接收的下一个序列号的值。随后的确认号是一个 4 位的头部长度字段，它指示 TCP 头部中 32 位字的数量，其中包括 TCP 选项。从技术的角度看，它也意味着应用数据的开始。在图 5-25 中的 16 位窗口大小，仅当分段是一个带有 ACK 控制位设置的确认时才会使用。它指定愿意接收的分段字节数，这是从确认字段中发送者即 TCP 接收者所指定的字节开始的。窗口的大小取决于套接字缓冲区的大小和接收方的接收速度。套接字缓冲区的大小可以使用套接字 API setsockopt() 编程。



图 5-25 TCP 报头格式

头部长度字段后跟 6 位控制位。第一位是 URG 位，设置为 1 表示使用 16 位紧急指针字段。紧急指针是一个序列号偏移，指示最后一个紧急数据字节。这种机制有利于 TCP 连接的带内信令实现。例如，用户可以使用 Ctrl + C 来触发一个紧急信号以便取消对等终端上进行的操作。接下来的是 ACK 位，指定确认号字段是有效的。如果没有设置 ACK 位，那么就忽略确认号字段。接下来是 PSH 位，其任务是通知发置 PSH 分组的接收方，立即将缓冲区中的所有数据发送到接收应用程序而不用等待有足够的应用数据填充缓冲区后再发送。接下来的位是 RST，用来重置一条连接。任何接收到 RST 设置分组的主机应立即关闭与该分组相关的套接字对。下一位是 SYN 位，用于初始化一条连接，如 5.3.1 节中所示。最后一位是 FIN，如 5.3.1 节中所示，用来指示发送方不再有更多数据发送，双方都可以关闭连接了。

TCP报头，以及将要讨论的选项，必须是一个32位字的整数倍。将可变的填充位添加到TCP报头，以确保TCP报头的结束和TCP有效载荷的开始都在一个32位边界上。填充是由零位组成的。

TCP选项

选项占据TCP头部最后的空间。一个选项是多个字节，可能从任何字节边界上开始。目前定义的选项包括：选项列表结束、无操作、最大分段尺寸、窗口缩放系数和时间戳。注意，所有选项都包含在校验和计算中。图5-26描述了TCP选项的格式。选项列表结束和无操作是只有一个字节的选项字段，其余的选项每个都包含3元组字段：一个字节的选项类型，一个字节的选项长度和选项数据。选项长度计算选项类型和选项长度的两个字节以及选项数据的字节。注意，选项列表可能会比数据偏移字段的短，因为强制选项列表结束选项的头部内容必须是零填充位。



图5-26 TCP选项

选项列表结束指示所有选项的结束，而不是每个选项的结束。根据数据偏移字段，选项列表结束仅在它没有刚好与TCP头部结束相一致时才使用。在选项之间可以使用无操作，例如，为了对齐字边界上随后选项的开始。不能保证发送方一定使用此选项，所以即使它们不从字边界上开始，接收方也必须为处理该选项做准备。

如果给出最大分段大小（MSS）选项，那么它就以在发送该分段的TCP端利用最大接收分段大小进行通信。该字段必须只在初始连接请求中发送（在带有SYN控制位设置的分段中）。如果不使用此选项，那么允许任意大小的分段。

如果发送的大小超过2^{16}字节，那么32位的序列号可用完。通常情况下，这不成问题，因为序列号可以循环使用。然而，在高速网络中序列号可能非常迅速地循环，所以很不使用的序列号可能会造成混乱。因此，就需要防止循环回绕的序号（PAWS）以避免产生副作用。使用TCP窗口缩放系数选项，TCP接收方可以通告一个非常大的窗口大小，通过与发送方协商一个移位计数来解释窗口大小的缩放。以这种方式，发送方就可以以非常高的速率发送。为了执行PAWS，利用TCP时间戳选项为每一个发送的分段附加一个时间戳。接收方将时间戳值复制到它相应的ACK中，以便于使用循环回绕序列号的分段可以被识别出来而不会与RTT估计器相混淆。

使用额外的TCP SACK选项以便提高在TCP拥塞控制快速恢复阶段的性能。选项包含两个字段，指示连续收到分段序列号的开始和结束。TCP SACK将在5.3.7节中详细研究。

5.3.6 TCP定时器管理

每个TCP连接保持一组定时器以驱动其状态机，如图5-8所示，即使是没有到达的分组触发状态转换也是如此。表5-2总结了这些定时器的功能。在本小节中，我们将详细研究两种强制的定时器，

重传定时器和坚持定时器，以及一个可选的定时器，保活定时器。由于对性能的考虑，这些定时器在不同操作系统上以不同的方式实现。

表 5-2 所有定时器的功能

名 字	功 能
连接定时器	为了建立一条新的 TCP 连接，发送一个 SYN 分段。如果在连接超时时间内没有收到对 SYN 分段的响应，那么发送方终止
重发定时器	如果数据没有得到确认并且此定时器到期，那么 TCP 就重传数据
延迟 ACK 定时器	接收方必须等到延迟 ACK 超时才发送 ACK。如果在此期间有数据要发送，那么它就用捎带方法发送 ACK
坚持定时器	死锁问题是通过发送方在坚持定时器超时后定期发送探测解决的
保活定时器	如果连接闲置了数小时，保活定时器超时，那么 TCP 就发送探测。如果没有收到响应，TCP 就认为另一端已经崩溃
FIN_WAIT_2 定时器	此定时器避免将一条连接永远留在 FIN_WAIT_2 状态，如果另一端已经崩溃
TIME_WAIT 定时器	该定时器用于 TIME_WAIT 状态以免进入 CLOSED 状态

[1] TCP 重传定时器

TCP 重传定时器的作用已在 5.3.2 节和 5.3.4 节中介绍过，本节研究 RTT 估计器的内部设计。为了测量 RTT，发送方使用 TCP 选项在每个数据分段放置一个时间戳，接收方在 ACK 分段中返回这些时间戳。然后发送方使用减法就可以测量每一个 ACK 的准确 RTT。RTT 估计器采用指数加权移动平均（EWMA）方法，它由 Van Jacobson 于 1988 年提出，它取新测量的 RTT1/8 加上老的平滑的 RTT 值的 7/8，形成新的 RTT 估计。8 为 2 的指数值，因此此操作可以通过一个 3 位移位指令轻松地完成。"移动平均"表示这种计算是基于平均的递归形式。同样，新的平均偏差是从 1/4 的新测量和 3/4 以前的平均偏差计算而来的。4 仅仅使用一个 2 位的移位指令就可实现。重传超时（RTO）是用测量平均 RTT 和平均 RTT 偏差的线性函数计算的，用公式表示为 RTO = RTT + 4 × 偏差（RTT）。在一条具有高延迟偏差的路径上，RTO 会显著地增加。

RTT 的动态估计遇到的一个问题是，当分段时间过剩并只要再次发送时，应该如何做。当一个确认到达时，不清楚该确认是指第一次传输还是相后的一次传输。一个错误的猜测可能会严重地损害 RTT 的估计。1987 年 Phil Karn 发现了这个问题，并建议对有任何重传的分段不更新 RTT。相反，RTO 在每次重传超时都会增加一倍，直到分段在第一次时间获得通过为止。此修改称为 Karn 算法。

开源实现 5.5：TCP 重传定时器

概述

在文献中，用于后退消息的时钟默认值是 500 毫秒，即发送方每 500 毫秒检查一次超时。由于在超时前没有分组重传，所以 TCP 连接可能需要很长时间才能从这种情况中恢复，TCP 性能会严重退化，尤其在重传超时（RTO）值远远小于 500 毫秒时，在当前的互联网下这是很可能出现的。目前在 Linux 2.6 中保持一个初粒度的定时器，以避免这种退化。

算法的实现

当有一个来自 IP 层的进入 ACK 时，将它传送到 tcp_input.c 中的 tcp_ack()函数。那里，它通过 tcp_ack_update_window()函数更新发送窗口，看看是否可以由函数 tcp_clean_rtx_queue()从重传队列中取出什么，是否由 tcp_cong_avoid()相应地调整 cwnd。tcp_clean_rtx_queue()函数更新多个变量，并调用 tcp_ack_update_rtt()更新 RTT 测量。如果使用了时间戳选项，那么函数总是调用 tcp_rtt_estimator()来计算平滑的 RTT，如图 5.27 所示。它通过 tcp_set_rto()函数使用平滑的 RTT 来更新 RTO 值。如果没有提供时间戳选项，那么当到达的 ACK 确认一个重传的分段时就不会执行更新（根据 Karn 算法）。

```
m -= (tp->srtt >> 3);      /* m is now error in rtt est */
tp->srtt += m;       /* rtt = 7/8 rtt + 1/8 new */
if (m < 0) {
    m = -m; /* m is now abs(error) */
    m -= (tp->mdev >> 3); /* similar update on mdev */
if (m > 0)
        m >>= 3;
} else {
    m -= (tp->mdev >> 2); /* similar update on mdev */
}
```

图 5.27 Linux 2.6 中的 RTT 估计器

tcp_rtt_estimator()的内容如图 5-27 所示，按照 Van Jacobson 1988 年的建议（1990 年他做了进一步的改进）开始估计平滑的 RTT 值。注意，srtt 和 mdev 分别是因特定的 RTT 版本和平均偏差，以便能尽快地计算出结果。按照 RFC 1122 定义，RTO 初始化为 3 秒，而且在连接中，它可以取 20～120 毫米的值。这些值定义在 net/tcp.h 文件中。

在图 5-27 中，m 代表 RTT 当前的测量值，tp 是指向 tcp_sock 数据结构的指针，可以在开源实现 5.4 中看到，mdev 代表平均偏差，srtt 代表平滑 RTT 的估计值。操作成右移 3 位和除以 8 相同，同理，右移 2 位和除以 4 相同。

练习

图 5-27 显示了如何根据 m 和它们以前的值来更新 srtt 和 mdev。你知道在哪里以及如何设置 srtt 和 mdev 的初始值？

【2】TCP 坚持定时器

TCP 坚持定时器只是为了预防以下死锁：接收方发送一个接收窗口大小为 0 的确认，告诉发送方等待。稍后，接收方更新并通告它的窗口大小，但是带有更新的分组丢失了。现在发送方和接收方都在等待对方做出行动，这就是死锁。因此，当坚持定时器超时后，发送方就向接收方发送一个探测信号，这个探测的回复就带回了窗口大小。如果窗口大小仍然是 0，那么坚持定时器就再次设置并重复循环；如果不为 0，那么就可以发送数据了。

【3】TCP 保活定时器（非标准）

通过 TCP/IP 检测崩溃的系统很困难。如果应用程序不传送任何信息，TCP 就不需要在连接上传输任何信息，TCP/IP 使用的许多介质（以太网）并不提供可靠的方法来判断是否有一个特定的主机打开着。如果一台服务器没有从客户机收到什么，那么很可能它什么都没发送，在服务器和客户机之间的网络可能发生了故障，服务器和客户机的网络接口可能断开了，或者客户机可能已经崩溃了。网络故障经常是暂时的（例如，当一台路由器发生故障后，通常要花几分钟来建立新路径才能稳定下来），那么不应该以放弃 TCP 连接而告终。

保活是 API 套接字的一个特征，它周期性地在空闲的链路上发送空分组，如果远程系统仍然正常，就会得到一个确认，如果它重启时由 RST 重置，如果它关闭时就超时。直到连接空闲数小时后，上述这些才会被发送。这样做的目的不是立即检测出冲突，而是让不需要的资源不永远地占用。

如果需要更快地检测远程主机，就应该在应用层协议中实现它。目前应用（如 FTP 和 Telnet）的大部分守护远程可以检测用户是否空闲。如果空闲，守护进程就关闭连接。

开源实现 5.6：TCP 坚持定时器和保活定时器

概述

在 Linux 2.6 内核中，坚持定时器也叫做探测定时器。它由 tcp_timer.o 中的 tcp_probe_timer()函数维护，而保活定时器由 tcp_timer.o 中的 tcp_keepalive_timer()函数维护。

数据结构

为了准时调用两个函数，它们应该被挂到一个时间链表上。例如，tcp_keepalive_timer()

通过 inet_csk_init_xmit_timers()接收到 sk->sk_timer 上。sk_timer 是 timer_list 结构，它的定义可以在 include/linux/timer.h 文件中找到。这个结构包含一个指示哪个函数将在时间到了的时候被调用的函数指针。此外，使用变量 data 保存将要传递给函数的参数。这里的 data 保存了一个指向对应套接字的指针以便让 tcp_keepalive_timer()知道检查哪个套接字。

算法实现

tcp_probe_timer()调用 tcp_send_probe0()发送一个探测分组。在函数名中的“0”意味着大小为 0 的窗口被接收方更新了。如果探测器定时器超时，那么发送方将发送一个零窗口探测分段，它包含激发接收方应答新窗口更新的新的序列号。

保活定时器的默认调用周期是 75 秒。当它被激活后，它检查每一个建立的连接，看看是否为空闲并对它们发起一次新的探测。对每一个已建立的连接，探测数默认限制为 5。所以，如果另一端的主机崩溃但没有重启，那么探测发送方就利用 tcp_keepopen_proc()函数将 TCP 状态清零；如果另一端崩溃而且在 5 次探测之内就重启了，那么当它接收到一个探测分组时它就回复一个 RST。然后探测的发送方就把 TCP 状态清零。

练习

阅读 net/ipv4/tcp_timer.c 弄清楚 tcp_probe_timer()在哪里以及是如何钩接的。它是直接或钩住 timc_list 结构就像 tcp_keepalive_timer()一样吗？

5.3.7 TCP 性能问题及增强

基于 TCP 应用的传送方式可以分为：1）交互式连接；2）批量数据传输。交互式应用，例如 telnet 和 WWW，处理由连续的请求和响应对组成的事务。相反，某些应用具有批量数据传输，例如使用 FTP 和 P2P 的下载/上传文件。这两种数据传送方式都有它们自己的性能问题，如表 5-3 所示，如果还使用前面提及的 TCP 版本。本节将介绍这些问题并给出解决方案。

表 5-3 TCP 问题和解决方案

传输类型	问题	解决方案
交互式连接	糊涂窗口综合症	Nagle，Clark
批量数据传输	ACK 大型	Zhang
	Reno 的 MPL[1]问题	NewReno，SACK，FACK

[1]MPL 表示多个分组丢失

（1）交互式 TCP 的性能问题：糊涂窗口综合症

对于交互式事务，在 TCP 中基于窗口流量控制的性能经历一种著名的状态：糊涂窗口综合症（SWS）。当这种症状发生时，小分组通过连接进行交换，而不是整个数据分段交换，这就意味着同样的数据量要以更多的小分组形式来发送。由于每个分组具有固定大小的头部开销，所以以小分组方式的传送意味着带宽的浪费。这种情况尽管在局域网中无关紧要，但在广域网中情况却非常严重。

SWS 状态可以由任何一端引起。发送方可以发送一个小的分组，而不用等待发送应用程序发送的更多数据以便以完整尺寸的分组来发送。例如，telnet 远程登录：因为在远程登录中每次按键都会产生一个分组和一个 ACK，通过 WAN 很长的往返时间，远程登录会浪费全球共享的 WAN 带宽。读者也可能会说，交互式应用的分组应该立即发送不管它们有多小。然而，这种分组大约 0.01～0.1s 的延迟限制不会影响可察觉到的交互性。

接收方也可能会导致 SWS 状态。接收方没有等待以便能从缓冲区中移动更多数据到接收应用程序，因此就可以通告一个小于完整分组大小的接收窗口，最终导致 SWS 状态。我们来看图 5-28 中的一个例子。假设 MMS＝320 并且服务器的初始接收窗口也同样设置为 320。再假设客户机总有数据要发送，因此服务器非常忙，每收到 4 字节的数据它只能从缓冲区中移走 1 字节。这个例子的工作如下：

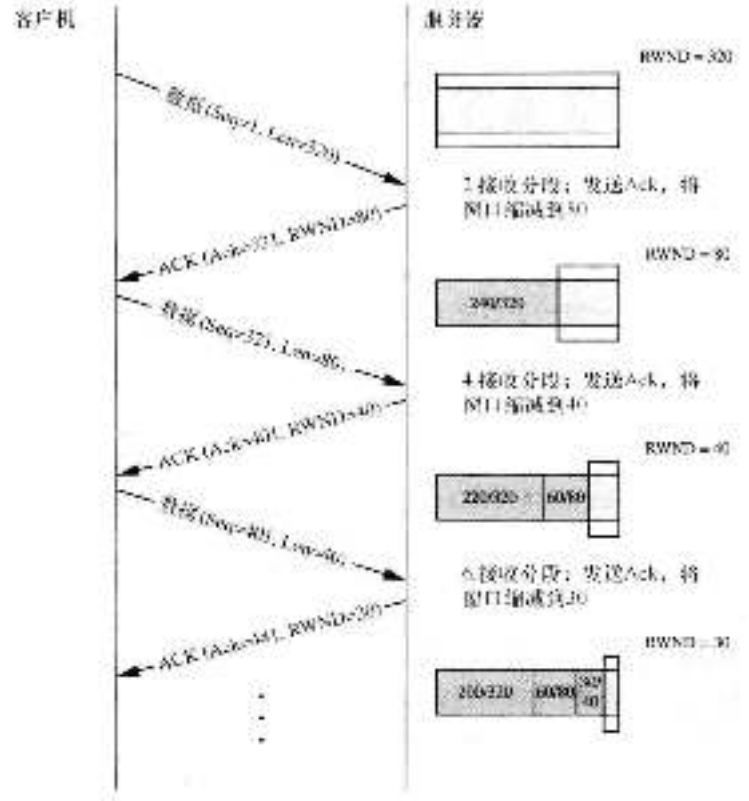


图 5-28 由接收方造成的糊涂窗口综合症

1）客户机窗口大小为 320，所以它可以立即给服务器发送 320 字节的分段。

2）当服务器接收到这个分段后，它就确认这个报文。由于只移走了 80 字节，所以服务器将窗口大小从 320 缩小到 80，并且在 ACK 中声明 RWND 为 80。

3）客户机接收到这个 ACK 后，知道窗口大小减小到 80，因此它发送 80 字节的分段。

4）当 80 字节的分段到达后，缓冲区中就包含 220 字节（第一个分段共有 240 字节，并且假设在传播延迟中另有 20 字节被移走）。然后服务器立即处理 80 字节中的 1/4，即 20 字节，所以将 60 字节加入到缓冲区中。然后服务器发送一个带有 RWND = 40 的 ACK。

5）客户机接收到这个 ACK 后，知道窗口大小减小到 40，因此它发送一个 40 字节的分段。

6）服务器在传播延迟期间移走 20 字节，留下 260 字节在缓冲区中。服务器从客户机那儿接收到 40 字节的分段，移走 1/4 即 10 字节，所以 30 字节又加入到缓冲区中，缓冲区中有 290 字节，因此服务器应把窗口大小从 320 字节减小到 30 字节。

糊涂窗口综合症的解决方案

为了阻止发送方引起 SWS，1984 年 John Nagle 提出了一种简单但优秀的算法，称为 Nagle 算法。这个算法在带宽饱和时减少发送分组的数量：设想未经确认的数据时就不发送小的新的分组，相反，TCP 将小的分段收集到一起当 ACK 到达时生一起发送。收集受到 RTT 往返时间的限制，因此不会受到互动的影响。Nagle 算法的优秀归功于它的自同步行为：如果 ACK 回来得很快，带宽就可能很大以致于分组发送得也很快；如果 ACK 回来花了很长的 RTT 时间，这意味着分布带路径，Nagle 算法就会通过发送完整大小的分段来减少小分段的数量。Nagle 算法的伪代码如图 5-29 所示。

另一方面，为了防止接收方进入 SWS 状态，使用 David D. Clark 在 1982 年提出的解决方案：通告声明会被延迟直到接收方缓冲区有一半是空的或能够装下整个分段，因此它为发送方保证了一个大窗口通告。当然延迟也是受限制的。

```
if there is new data to send
  if window size >= MSS and available data >= MSS
    send complete MSS segment
  else
    if there is outstanding data and queued data live time <threshold
     enqueue data in the buffer until an ACK is received
    else
      send data immediately
    endif
  endif
endif
```

图 5-29 Nagle 的算法

[2] 批量数据传输的性能问题

基于窗口流量控制的批量数据传输的性能可以用带宽延迟乘积（BDP）和管道大小很好地理解。在图 5-30 中，我们可以看到一条由前向传输数据信道和反向 ACK 信道组成的全双工端到端 TCP 网络管道。可以将一条网络管道功能想象成一根水管，它的宽和长分别对应于带宽和 RTT。利用这种比喻，管道尺寸就对应于可以在水管中传输的水。如果全双工信道总是满的，就可以容易地导出这样连接的性能：

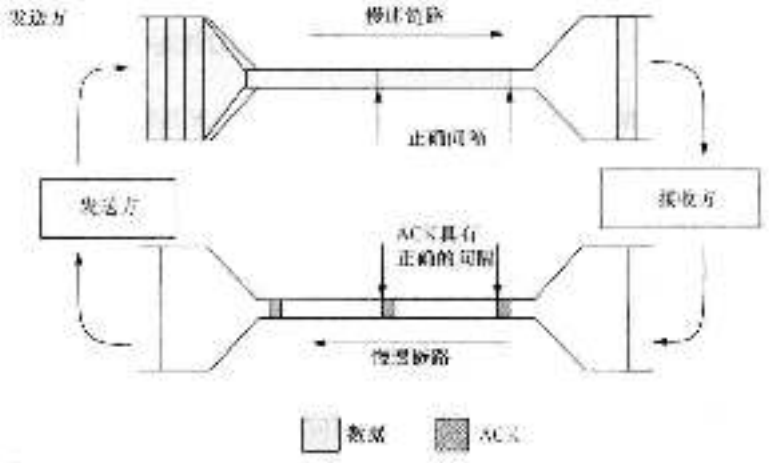


图 5-30 端到端全双工网络管道的可视化

$$\text{Throughput（吞吐量）} = \text{pipe size（管道尺寸）}/\text{RTT} \tag{5-1}$$

直观地说，式（5-1）意味着在 RTT 时间内管道中可以传输的数据量。当然吞吐量等于管道的带宽。然而，管道并不是总是满的。当 TCP 连接启动并且遇到分组丢失时，TCP 发送方将调整其窗口以适应网络拥塞。在 TCP 填满管道之前，它的性能如下：

$$\text{吞吐量} = \frac{\text{最大字节数}}{\text{RTT}} = \min(\text{CWND}, \text{RWND})/\text{RTT} \tag{5-2}$$

式（5-1）和式（5-2）指出：如果 TCP 连接的 RTT 是固定的，那么连接吞吐量就会受到管道容量（管道尺寸）、接收方缓冲区（RWND）、网络状态（CWND）的约束。也就是说，式（5-1）是连接吞吐量的上限。

因为更好的性能说明更有效地利用网络管道，填充管道的过程严重影响其性能。图 5-31 演示了使用 TCP 填充网络管道的步骤。图 5-31（1）～（6）演示了第一个分组从左边传送到右边和从接收方向发送方传输的 ACK。在接收到 ACK 后，发送方把拥塞窗口提高到 2，如 5-31（7）所示。这个过程继续就像图 5-31 接下来显示子图那样。在拥塞窗口到达 6（如图 5-31（35）所示）后，网络管道就满了。

注意利用 TCP 的批量数据传输的吞吐量可以建模成多个参数（如 RTT 和分组丢失率的函数）。这个领域的发展是以能够精确地预测 TCP 源的吞吐率作为目标。主要挑战在于我们如何解释以前取样的分组丢失事件以便于预测 TCP 连接的未来性能。分组丢失之间的间隙既可以是独立的也可以是相关的。

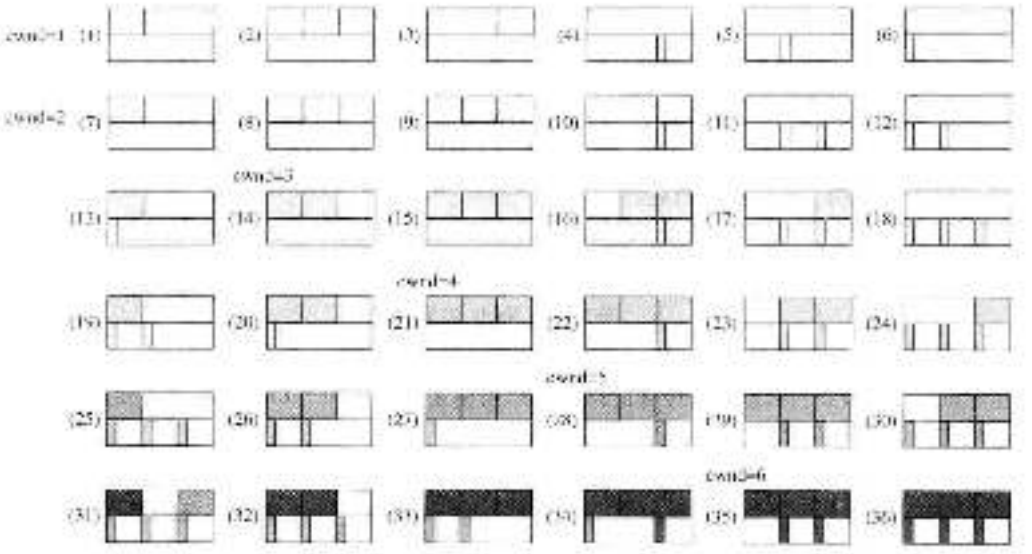


图 5-31 使用 TCP 填充管道的步骤

在 Paithye 的工作中，使用了一个易于理解的模型，它不仅考虑通过快速重传算法恢复丢失的分组而且还通过超时重传恢复的分组。

下面我们将学习批量数据传输中遇到的两种主要性能问题：ACK 压缩问题和 TCP Reno 的多分组丢失问题，并讨论了有关建议或解决方案。

ACK 压缩问题

在图 5-32 中，全双工管道仅包括来自左边发送方的数据流，所以 ACK 之间的间隔定义为能够触发发送方发送新数据的固定时钟速率。然而，当也有来自右边产生的分组流时，如图 5-32 所示并与图 5-30 进行比较，因为 ACK 在反向管道中可以与数据流混合在同一个队列中，所以连续的 ACK 就可能不正确地分隔。因为一个大分组的传送时间远远大于 64 字节的 ACK，所以 ACK 能够周期性地压缩成串，并且造成发送方突发数据流量，导致中间路由器队列长度有很大的波动。在数据分组中捎带 ACK 可以减轻 ACK 压缩问题。但是，因为线到端信道实质上是由逐跳系统热噪而成，所以中间互联网路由器中的穿叉流量也会引起这种现象。

目前还没有明确的方法来解决 ACK 压缩问题。1991 年 Zhang、Shenker、Clark 建议利用 TCP 发送方调整发送数据分组的速率而不仅依赖 ACK 时钟来减轻这种现象。ACK 时钟已经证明是无效率的，如图 5-32 所示。

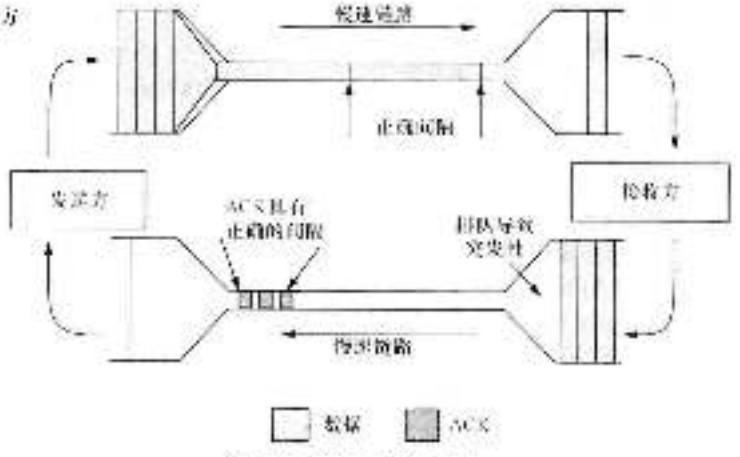


图 5-32 ACK 压缩现象

TCP Reno 的多分组丢失问题

在 Reno 中，当一个窗口发生多分组丢失时，接收方总是利用重复 ACK 作为响应，因为发送方假设每个 RTT 最多只有一个分组丢失。因此，在这种情况下，发送方必须花费多个 RTT 来解决所有这些丢失。同时，重传超时更加频繁，因为尽管有许多未被确认的分组等待重传，但只有少数的分组（由于快速恢复触发的 cwnd 的减少而限制）才能发送。让我们来观察图 5-33 中描述的例子。接收了分组 30 的确认号后，发送方就发送分组 31 – 38 的分组。再次为了清晰起见，ACK 分组中的确认号是接收到的分组的序列号而不是接收方期望接收的下一个分组的序列号。

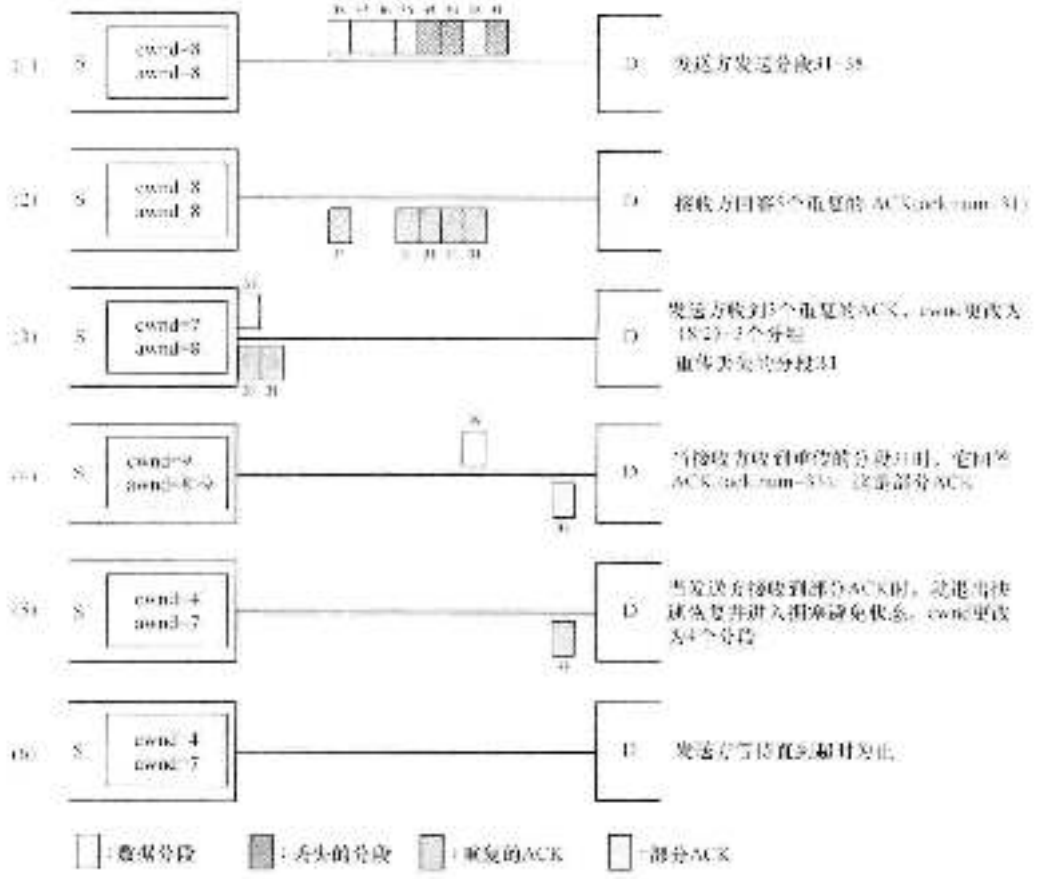


图 5-33 Reno 的多分组丢失问题

假设拥塞窗口 cwnd 等于 8 个分组，而且分组 31、33 和 34 在传送过程中丢失了。因为接收到了分组 32、35、36、37、38，所以接收方将为对丢失的分组 31 发送 5 个重复的 ACK。当发送方接收到第 3 个重复的 ACK（ack num = 31）时，它认识到分组 31 丢失了，然后它立即将 cwnd 设为 [8/2] +3 = 7 个分组并重传丢失的分组。在收到另外两个重复的 ACK 后，发送方继续将 cwnd 增加 2 并发送一个新的分组 39。在收到 ACK（ack num = 33）后，发送方从快恢复过渡到拥塞避免，并将 cwnd 设置为 4 个分组。然后，发送方接收一个重复的 ACK（ack num = 35）。当 cwnd = 4，awnd = 7（40 – 35）时，发送方停止发送任何导致超时重传的数据。

注意，当在一个数据窗口中丢失多个分段时，Reno 并不总是超时。当 cwnd 非常大的情况下发生多个丢失事件时，任何部分 ACK 可能不仅使 Reno 退出快速恢复，而且还会由于另外一批三个重复的 ACK 而触发另一个快速重传。到目前为止这些正是我们要求，尽管它会减准丢失恢复。但是，如果太多的分组在一个 RTT 内丢失，那么将会引起 cwnd 在下一个 RTT 内减半很多次以至于太少的未经确认的报文在管道中而不能触发下一次快速重传，Reno 将超时，这将进一步延长丢失恢复。

为了对经多个分组丢失问题，NewReno 和 SACK（选择性确认，定义在 RFC 1072 中）版本通过两

种不同的途径去解决这个问题。前者，一旦接收到部分确认，发送方就会继续使用快速恢复算法而不是回到拥塞避免算法。另一方面，SACK 修改接收方的行为以便告诉发送方已经接收到一组不连续的数据并进行了排队，在重复确认中还附带了额外的 SACK 选项。利用 SACK 选项中的信息，发送方就可以正确快速地重传丢失的分组。Mathis 和 Mahdavi 又提出了转发确认（FACK）来改进 SACK 中的快速恢复方案。与被完善快速重传和快速恢复算法的 NewReno/SACK/FACK 相比，1995 年提出的 TCP vegas，使用能够度 RTT 来辅助检测分组丢失和拥塞，因此减少了 Reno 超时的可能性。

历史调查：NewReno、SACK、FACK 和 Vegas 算法中的多分组丢失恢复

这里我们更加细致深入地分析 Reno 的 MPL 问题如何在 NewReno，SACK，FACK 和 Vegas 中得到改善，使用如图 5-33 所示的同一个例子。

TCP Reno 问题的解决方案 1：TCP NewReno

NewReno 的标准化文件是 RFC 2582，它修改了 Reno 快速恢复阶段以便解决 MPL 问题。仅当发送方检测到第一个丢失分组之前传送到确认最后发送分组的 ACK 时，它才会脱离原始的快速恢复。在 NewReno 中，这种选择被定义为“快速恢复的终止点”，同时任何在此时间之前的非重复 ACK 都被当做部分 ACK 确认。

Reno 将部分 ACK 确认当做是丢失分组的一次成功重传，因此发送方返回到拥塞避免以便发送新的分组。相反，NewReno 认为这是一种进一步发生分组丢失的信号，因此发送方立即重传丢失的分组。当部分 ACK 到达后，发送方通过成去新数据的确认总量再增加一个用于重传数据的分组来调整 cwnd。发送方仍然在快速恢复中直到快速恢复的终止点为止。因此当数据窗口中丢失多个分组时，NewReno 就能够以每次一个 RTT 的发生检测到传。

对于图 5-33 所示的例子，当在第 4 步中重传的分组 31 被接到时，就发送部分 ACK（ack num = 33）。图 5-34 阐述了 NewReno 的改进。当发送方接收到部分 ACK（ack num = 33）后，它立即重传丢失的分组 33 并且将 cwnd 设置为（9 - 2 + 1）= 8，其中 2 是新确认的数据数量（分组 31 和 32），1 代表已经退出管道的重传分组。类似地，当发送方收到部分 ACK（ack num = 34）时，它就立即重传丢失的分组 34。直到分组 40 的 ACK 被接收到时，发送方成功地退出快速恢复而不会发生任何超时。

TCP Reno 问题的解决方案 2：TCP SACK

尽管 NewReno 能够解决多个分组丢失问题，但发送方仅会得知在一个 RTT 中一个新丢失的分组。然而，在 RFC 1072 中提出的 SACK 选项解决了这个问题。发送方通过结合 SACK 选项来传送重复的 ACK 许方对丢失分组的响应。RFC 2018 首次定义了 SACK 选项。为确定描述了发送方和接收方的行为。

SACK 选项用于报告接收方成功接收不连续的数据块。通过每个块内的第一个和最后一个分组的两个序列号。由于 TCP 选项长度的限制，在一个重复的 ACK 中有 SACK 选项的最大数。第一个 SACK 选项必须报告最后接到的数据块，其中包含触发这个 ACK 的分组。

SACK 直接调整 cwnd 而不是 ssmc。因此一旦进入快速恢复，cwnd 就被减半并在这一阶段固定下来。无论发送方发送一个新分组还是重传一个老的分组，awnd 都增 1。然而，当发送方接收到一个带有 SACK 选项，指示新数据已被到重复 ACK 时，awnd 都减 1。SACK 发送方也以特殊的方式处理部分 ACK。也就是说，发送方将 awnd 减 2 而不是减 1，因为一个部分 ACK 代表两个离开网络管道的分组：一个原始分组（假设已经丢失）和一个重传的分组。

图 5-35 说明了 SACK 算法的一个例子。每个重复 ACK 都包含成功接收到的数据块信息。当发送方接收了 3 个重复 ACK 后，它就知道分组 31，33 和 34 丢失。因此，如果条件允许，发送方就会立即重传丢失的分组。

TCP Reno 问题的解决方案 3：TCP FACK

FACK 是作为一种 SACK 的精确而提出的。在 FACK 中，发送方利用 SACK 选项来确定已收到的最前面的分组，最前面的分组意味着已经收到的最大序号的分组。为了提高精确度，FACK 将 awnd 估计为：awnd = snd.nxt - snd.fack + retran_data，其中 snd.fack 是在 SACK 选项中报告的最前面的

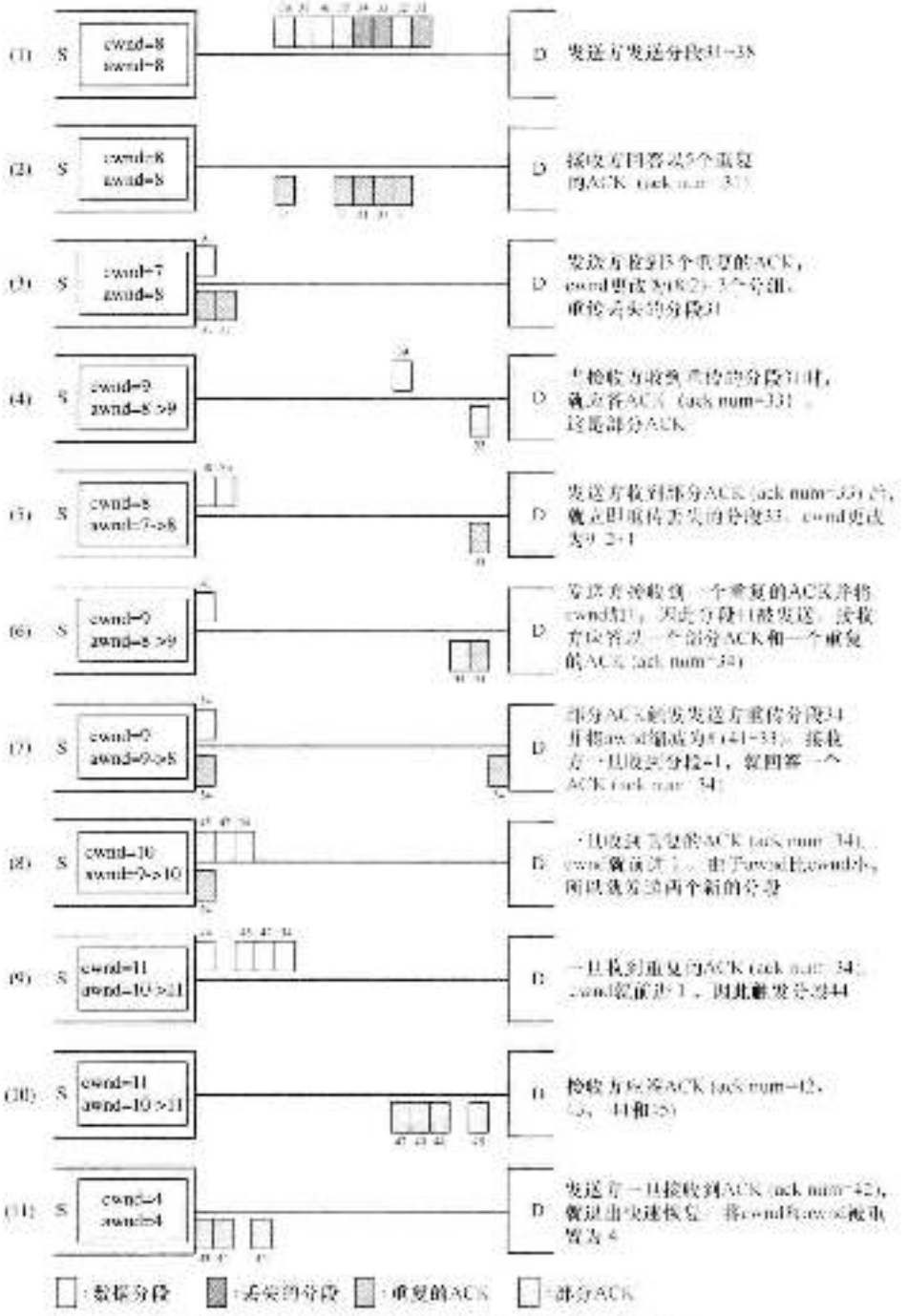


图5-34 对TCP Reno问题的解决方案1：NewReno

分组如1，retran data是标记前面的部分ACK后重传的分组数。由于发送方需要很长时间等待3个重复的ACK，所以FACK会更早进入快速恢复。也就是说，当(snd.fack - snd.una) > 3时，发送方此不用等待有3个重复的ACK，可以进入快速恢复。

图5-36演示了FACK的发送。发送方在接收到第二个重复的ACK时就发起重传，因为(snd.fack - snd.una)，即(36 - 31) = 5 > 3。丢失的分组在FACK中比在SACK中更快重传，因为前者计算awnd更加准确。因此，在图5-36中，虽然未经确认的分组数量初定为4，

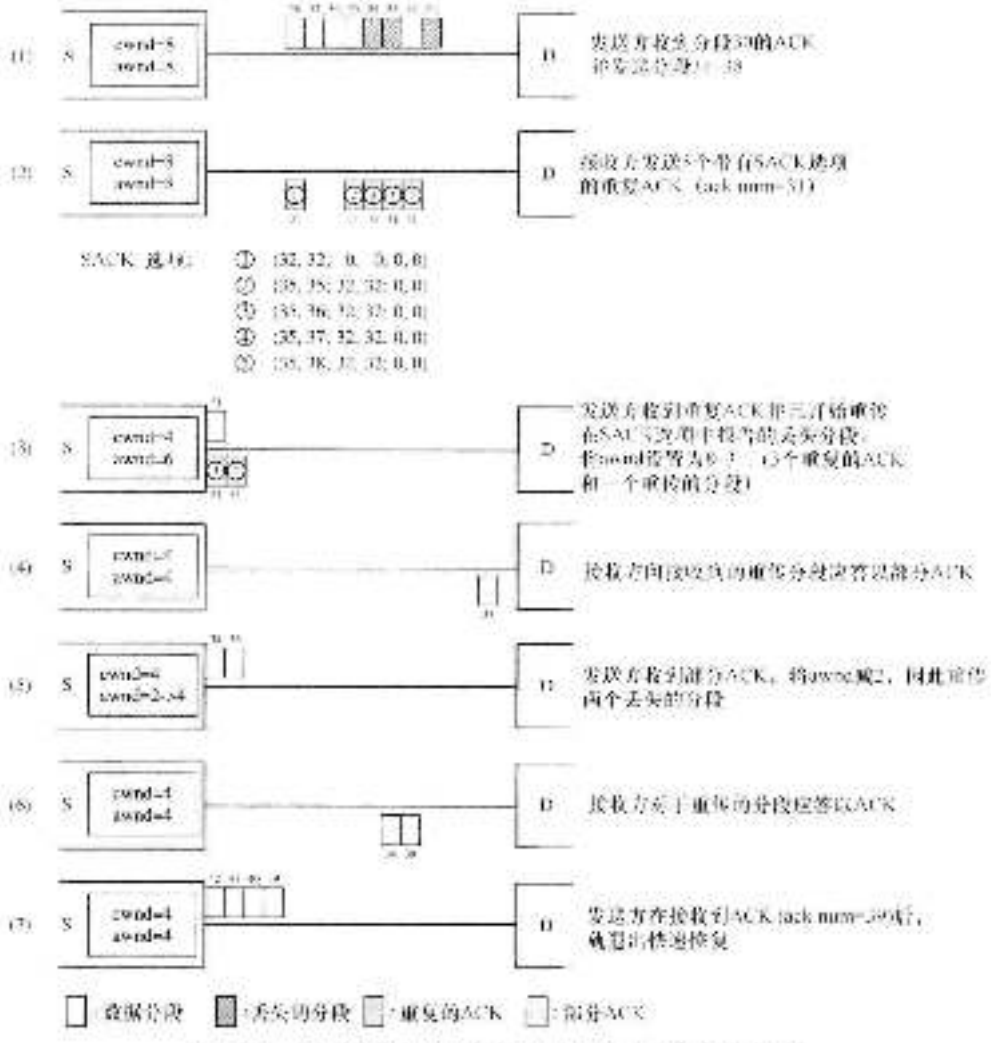


图5.35 对TCP Reno问题的解决方案2：TCP SACK选项

TCP Reno问题的解决方案4：TCP Vegas

Vegas首先修改了Reno触发快速重传的时机。一旦接收到重复的ACK，通过检测当前时间和相关分组发送时间之差加上最短RTT是否比超时值大，Vegas决定是否触发快速重传。如果大，Vegas就不等待更多的重复ACK到达，然立即触发快速重传。这种改进可以避免一种情形：发送方永远接收不到3个重复的ACK，因此就必须依赖于粗粒度的重传超时。

经过重传后，发送方过过检查未确认分组的粗粒度超时来判断是否有多个分组丢失。如果发生了超时，发送方就立即重传分组而不需要等待任何重复ACK的到达。

事实上，TCP Vegas也利用粗粒度RTT来改进拥塞控制机制。与对分组丢失做出响应然后再降低发送速率来缓解拥塞的Reno相比，Vegas尝试预测拥塞，然后及早减少发送速率以便避免拥塞和分组丢失。为了预测拥塞，Vegas在连接期间跟踪最短的RTT，然后把它保存在一个名为BaseRTT的变量中。然后，用awnd除以BaseRTT，Vegas了解到期望的发送速率，标记成Expected（期望值），连接可以使用此值而不会拥成分组在路径上的排队等待。下一步，Vegas将预期发送率（Expected）与标记为Actual的当前实际发送率进行比较，然后相应地调整awnd。假设Diff = Expected − Actual并给出两个阈值 $a<b$，单位为KB/s。当Diff < a时Vegas中的awnd在每个RTT中会增1即（awnd + 1）/RTT；当Diff > b时，每个RTT将Vegas中的awnd减1；如果 $a<$ Diff $<b$，则awnd保持不变。

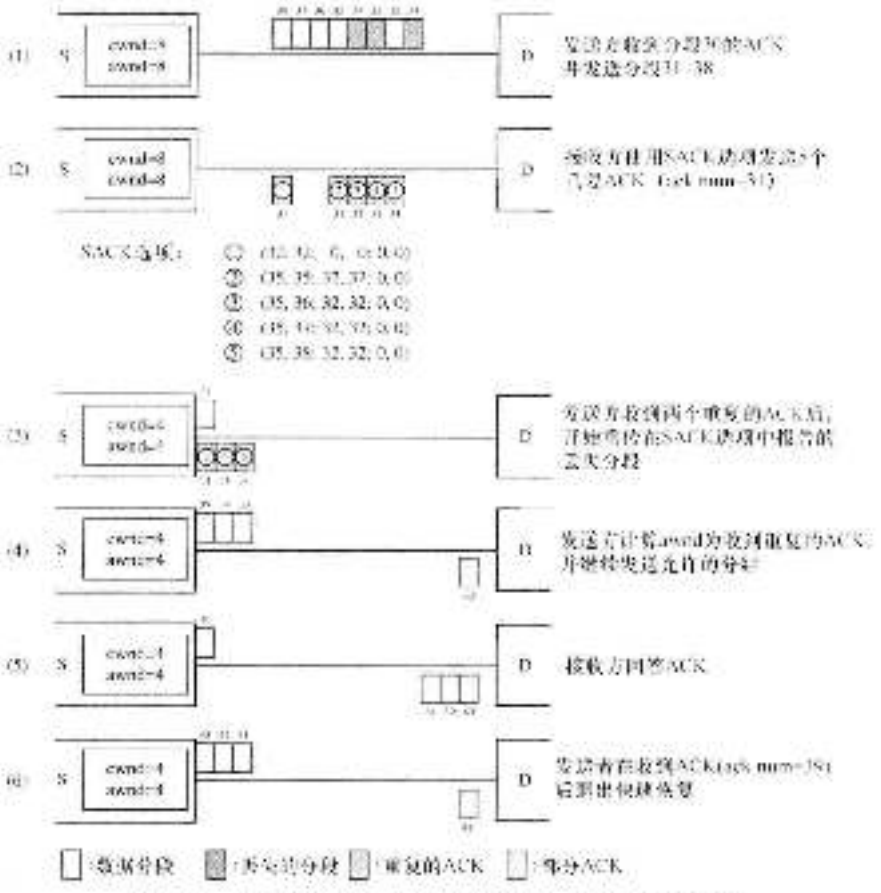


图 5-36 对 TCP Reno 问题的解决方法 3：TCP FACK 修改

调整发送率将 Diff 控制在 α 和 β 之间，表示 Vegas 连接平均占用的网络缓冲区至少为 α 字节/秒才能最佳利用带宽，并且最多为 β 字节/秒以避免网络过载。按照 Vegas 算法作者的建议，α 和 β 分别设为 1 倍和 3 倍的 MSS/BaseRTT。

行动原则：TCP 用于具有巨大带宽延迟乘积的网络

随着网络技术的进步，链路速率不断提高，会产生一条带有大带宽延迟乘积的网络路径。这就是路径带宽和它的 RTT 的乘积。这种网络的一个例子就是卫星连接，在这种情况下，RTT 非常大，链路带宽也可能非常高。

在这种网络中使用标准的 TCP 往往会非常差，因为它不可能充分利用可获得的带宽。如果发送方能够发送足以带宽延迟乘积的足够大的未经确认的数据，协议只能获得最佳吞吐量。如果发送的数据量不够多，那么这条路径就不能保持它，协议只能在路径的峰值效率之下工作。然而，在一个有着大带宽延迟乘积的网络中，这种无效率状态很可能会出现。为应对 TCP 现实拥塞方案，如 BIC、CUBIC、FastTCP 和 HighSpeedTCP 都尝试解决这个问题。它们在提高传送速率方面有很大进步，当遇到分组丢失时会后退很多快又重新开始加速传输。

在 BIC 中使用的最重要的部件就是二进制搜索递增。当分组丢失事件发生时，BIC 就减小它的窗口。窗口大小在减小前，设置为最大；在减小后，窗口大小设为最小。然后，通过快速探测到“目标”BIC 执行二进制搜索，目标就是在最大和最小窗口之间的中间点。按照分组是否会发生丢失，将最大值或最小窗口设置成无限期的。当目前窗口大小与目标窗口大小之差很大时，二进制搜索递增允许 BIC 更快地进行。当两个窗口大小的差距缩小时，为了 TCP 公平，它就切换成线性增长。

CUBIC 使用一种更简单的函数，一种立方函数，其形状与 BIC 窗口的很相似，以实现同样的目

标。这种函数在接受目标窗口时把二进制搜索结构维护起来。快速 TCP 可以减少队列延迟而不是减少丢失概率，判断网络中是否出现拥塞。通过测量这个因素，就把 TCP 更快地增大拥塞窗口。当 HS-TCP 创建一个窗口调节时，它也急剧地增大拥塞窗口，因此它可以更好地适应可获得带宽的变化。它使用一张表来判断什么因素决定增大拥塞窗口。

5.4 套接字编程接口

网络应用程序使用由底层协议提供的服务来执行特殊目的的网络互联任务。例如，telnet 和 ftp 等应用程序使用由传输层提供的服务；ping、traceroute 和 arp 直接使用由 IP 层提供的服务；分组捕获应用程序直接运行在链路层协议之上，它能够配置或捕获整个分组，包括链路协议头部。在本节中，我们将学习 Linux 如何为编程上述的应用程序而实现套接字接口。

5.4.1 套接字

套接字是一个对通信信道端点的抽象。因名思义，“端到端”协议层控制着信道两端之间的数据通信。端点由网络应用程序使用适当类型的套接字 API 创建的。然后网络应用程序就可以在该套接字上执行一系列操作。在公接字上可以执行的操作包括多种操作：例如，将端口号与套接字关联起来，在公接字上发起或接受连接，或者释放套接字）、承载传输操作：例如，通过套接字将数据写入到某些对等应用程序，或者通过套接字从某些对等应用程序上读取数据），以及状态操作（如查找与套接字关联的 IP 地址）。可以在公接字上执行的全部操作集合构成了套接字 API。

为了打开一个公接字，应用程序应该首先调用 socket() 函数以便初始化一个端到端的信道。标准公接字调用，sk = socket(domain,type,protocol)，需要 3 个参数。第一个参数指定域或地址族。常用的族有绑定在本地主机上用于通信的 AF_UNIX，用于基于 IPv4 协议通信的 AF_INET。第二个参数指定套接字类型。当处理 AF_INET 族时常用的套接字类型值，包括 SOCK_STREAM（通常与 TCP 一起使用）和 SOCK_DGRAM（与 UDP 一起使用）。套接字类型影响分组在传递到应用程序前内核如何处理它。最后一个参数指定处理经过套接字分组的协议。socket 函数返回一个文件描述符，通过它就可以将套接字应用到套接字上。

套接字参数的值依赖于所使用的底层协议。在接下来的两个小节中，我们探讨了 3 种类型的套接字 API。它们分别对应于传输层、IP 层和链路层的访问，正如在它们的开源实现中所见。

5.4.2 通过 UDP 和 TCP 绑定应用程序

网络应用程序广泛使用的服务是由 UDP 和 TCP 传输协议所提供的。由 socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP) 函数返回套接字文件描述符，并初始化为一个 UDP 套接字，其中 AF_INET 表示可联网地址族，SOCK_DGRAM 代表数据报服务，IPPROTO_UDP 表示 UDP 协议。在描述符上可以执行一系列操作，如图 5-37 所示的函数。

在图 5-37 中，在连接建立之前，UDP 服务器以及客户机创建一个套接字并使用 bind() 系统调用为套接字分配一个 IP 地址和端口号。注意的 bind() 是可选的，通常不在客户机上用它。当没有调用 bind() 时，内核就为客户机选择默认的 IP 地址和端口号。然后，当 UDP 服务器绑定到一个端口之后，它就准备好接收来自 UDP 客户机的请求。UDP 客户机通过循环使用 sendto() 和 recvfrom() 函数做一些有用的工作，直到完成它的任务为止。UDP 服务器继续接受请求，处理请求，并使用 sendto() 和 recvfrom() 返回结果。通常情况下，UDP 客户机不需要调用 bind()，因为它并不需要使用众所周知的端口。当客户机调用 sendto() 时，内核就动态地为它分配一个未使用的端口。

同样，从 socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) 返回的一个套接字文件描述符初始化为一个 TCP 套接字，其中 AF_INET 指示互联网地址族，SOCK_STREAM 代表可靠的字节流服务，IPPROTO_TCP 意味着 TCP 协议。在描述符上执行的函数如图 5-38 所示。这里默认情况下，不在客户机上调用 bind()。

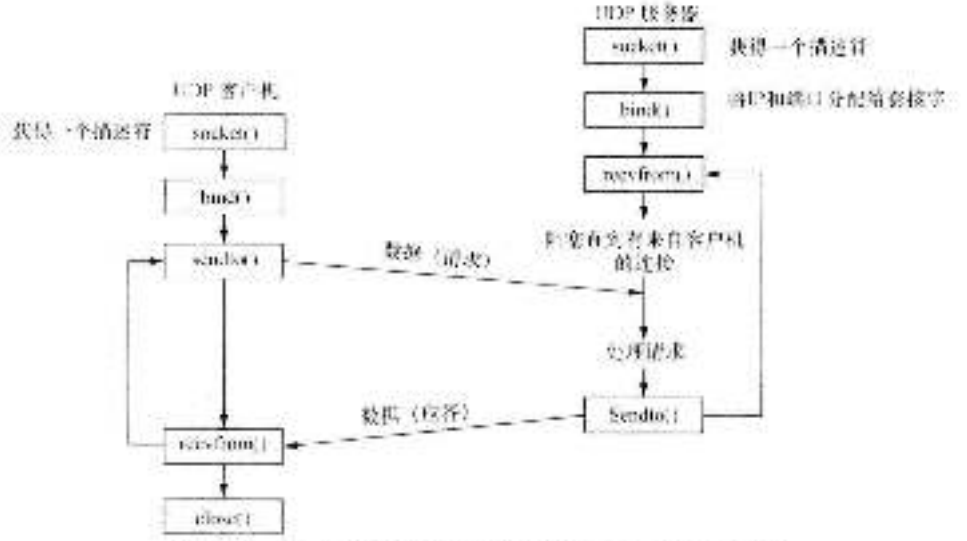


图3.37 简单UDP客户机/服务器应用程序的套接字函数

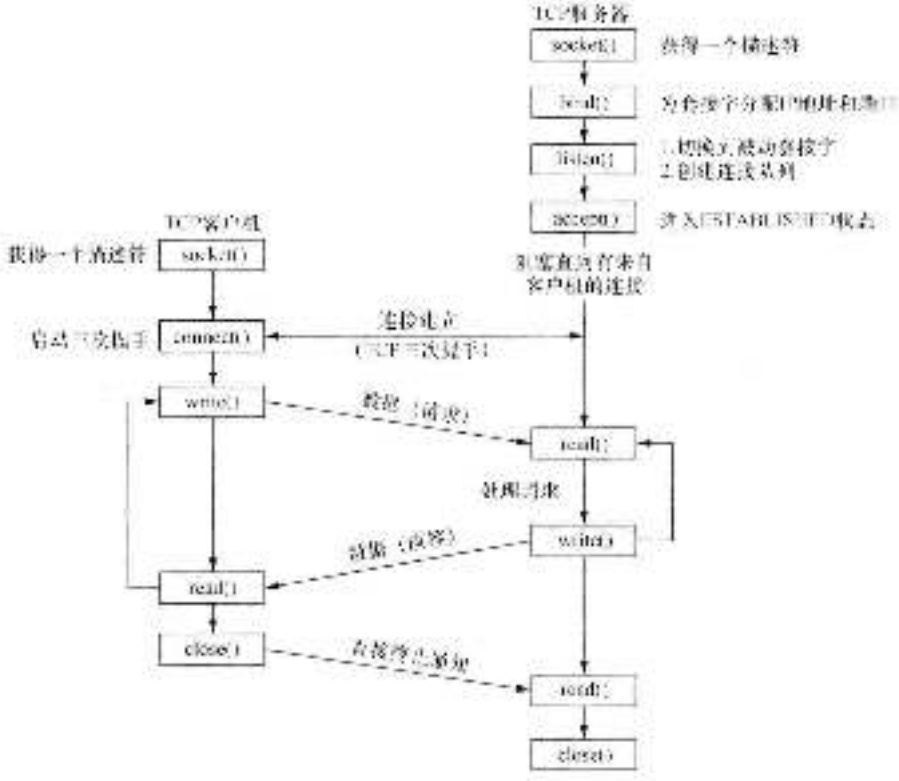


图3-38 简单的TCP客户机/服务器程序的套接字函数

简单的TCP客户机/服务器程序的流程图有点儿复杂，因为它是面向连接的TCP，它包含了连接建立，数据传输和连接终止等环节。除了bind()外，服务器调用listen()为套接字创建连接队列并等待来自客户机的连接请求。listen()系统调用表达了服务器愿意开始接受进入连接请求的意愿。每个监听套接字包含两个队列：1）部分建立请求队列；2）完全建立请求队列。在三次握手期间，请求首先驻留在部分建立队列中。一次握手完成连接建立后，请求将移到完全建立请求队列中。

在大多数操作系统中的部分建立请求队列中有一个最大的队列长度，例如，5，即使用户指定的

的比它更大也是 5。因此，部分建立请求队列可能成为拒绝服务（DoS）攻击的目标。如果一个黑客不断地发送 SYN 请求而不完成三次握手，请求队列将饱和，不能接受新的来自合法客户机的连接请求。

系统调用 listen() 通常后跟着 accept() 系统调用，它们的工作是从完全建立的请求队列中将第一个请求出队，初始化一个新的套接字对，并返回为客户机创建的新套接字文件描述符。也就是说，由 BSD 所提供的 accept() 系统调用会导致自动地创建一个新套接字，这与 TLI 套接字存在着很大的不同，因为应用程序必须显式地为新的连接创建一个新的套接字。请注意，原来的监听套接字仍然监听众所周知端口上新的连接请求。当然，新的套接字对包含客户机的 IP 地址和端口号。然后服务器程序决定是否接受客户机的连接请求。

TCP 客户机使用 connect() 调用三次握手过程建立连接。在此之后，客户机和服务器可以执行它们之间的字节流传输。

行动原则：SYN 洪泛和 cookies

使用三次握手协议可能会导致 SYN 洪泛攻击。此时攻击者会向受害者的系统发送许多连续的 SYN 请求。它相当于服务器在收到 SYN 时就分配资源，但从来不会收到 ACK。当这些半打开的连接耗尽了服务器上的所有资源时，就不能再建立新的合法连接，从而导致拒绝服务（DoS）。主要有两种方法实现 SYN 洪泛攻击：故意不发送最后的 ACK，或者在 SYN 中使用欺骗的源 IP 地址，导致服务器将 SYN ACK 发送给错误的 IP 地址，从而永远不会收到 ACK。

可以利用 SYN cookies 来防范 SYN 洪泛攻击。SYN cookies 定义为“由 TCP 服务器特别选择的初始 TCP 序列号”。当使用 SYN cookies 的服务器的 SYN 队列（存储到达的 SYN）满时，不必放弃连接。相反，它发送回一个具有特殊设计的初始序列号的 SYN，也就是一个 SYN cookie 的 SYN + ACK。当服务器从客户机接收到随后的 ACK 时，首先检查该序列号，然后重建的 SYN 队列表项，就像 SYN 存储在它的 SYN 队列中一样，在这个序列号中使用编码的信息。也就是说，当发出 SYN cookies 时，服务器不会依赖于 SYN 队列来追踪三次握手。相反，它依赖于编码的 SYN cookies。因此，即使它的 SYN 队列已满，服务器仍然能够接受完成三次握手的真正连接。正如我们将要在第 6 章中学习到的，类似的 cookie 思想也可用于无状态的超文本传输协议（HTTP）以便跟踪长期的会话状态。

开源实现 5.7：套接字向内、向外的读/写

概述

图 5-39 显示了每个协议的 Linux 2.6 内核部分的相对位置。一般套接字 API 及其后续的函数调用位于 net 目录中。与 IPv6 的情况一样，IPv4 的源代码单独放在 ipv4 目录中。BSD 套接字只不过是一个到其底层协议（如 IPX 和 INET）的接口。如果套接字地址族指定为 AF_INET，那么目前广泛使用的 IPv4 协议就对应于 INET 套接字。占主导地位的路路器技术，以太网，将其头部构建在 net/ethernet/eth.c 内。之后，以太网帧就被位于 drivers/net/目录中的以太网驱动程序从主存储器移动到网络接口卡上。在此目录中的驱动程序依赖于硬件，因为许多厂商具有不同内部设计的以太网卡产品。类似的结构，也适用于 WLAN 以及其他链路。

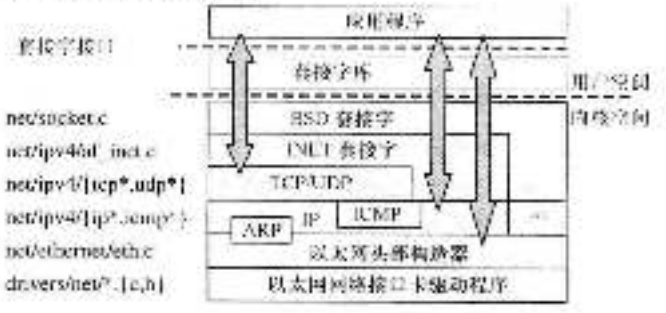


图 5-39 Linux 2.6 中的协议栈和编程接口

算法实现

在 Linux 中，简单的 TCP 客户机/服务器程序所使用的套接字 API 的内部情况，如图 5-40 所示。将用户空间程序调用的编程 API 转换为 sys_socketcall()内核调用，然后再调用它们对应的 sys_*()调用。sys_socket()（在 net/socket.c 内）调用 sock_create()分配套接字，然后调用 inet_create()根据给定的参数初始化 sock 结构。其他 sys_*()函数调用它们对应的 inet_*()函数，因为 sock 结构初始化为互联网地址族（AF_INET）。由于在图 5-40 中，read()和 write()不是特定套接字的 API 而是通常由文件 I/O 操作所使用的，它们的调用流程会沿着文件系统中它们的 inode 操作进行，以便找到某个给定的文件描述符实际上与某个 sock 结构相关。接下来将它们编译为相应的 do_sock_read()和 do_sock_write()函数，这是套接字感知的。

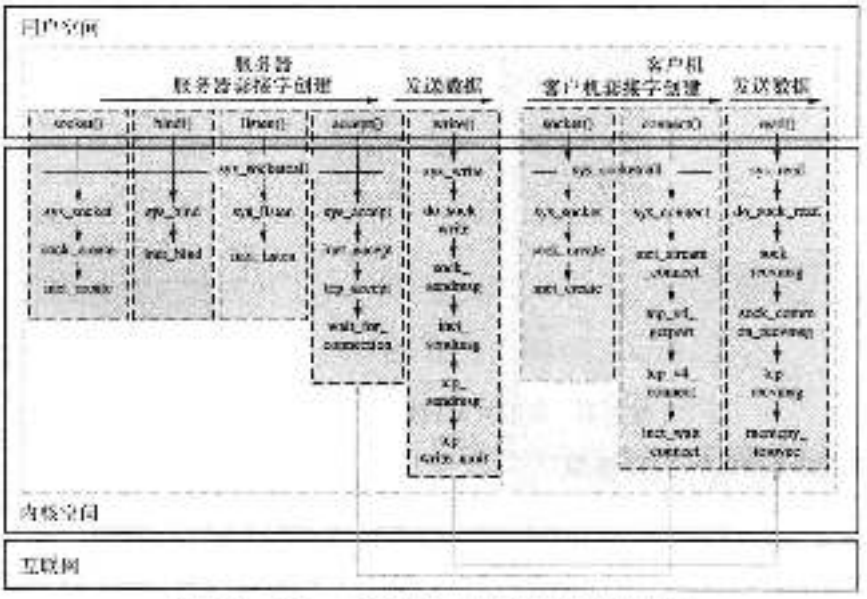


图 5-40 Linux 中的读/写：内核空间与用户空间

在大多数 UNIX 系统中，将 read()/write()函数集成到虚拟文件系统（VFS）中。VFS 是内核中的软件层，提供文件系统到用户空间的接口。它还提供了一个在内核中允许不同文件系统实现共有的抽象。

数据结构

在 Linux 2.6 中，由 TCP 连接函数（如图 5-40 所示）使用的内核数据结构如图 5-41 所示。发送方初始化套接字并获取文件描述符（假设在打开的文件表 FD[1]中），当用户空间程序在该描述符上操作时，它就沿着箭头键接以便指向文件结构，其中它包含了指向 inode 结构的一个目录表项 f_dentry。inode 结构中可以初始化为 Linux 支持的各种文件系统类型，也包括套接字结构类型之一。套接字结构包含一个 sock 结构，它保持与网络相关的信息以及从传输层到链路层的数据结构。当套接字初始化为一个字节流的、可靠的、面向连接的 TCP 套接字时，然后传输层协议信息 tp_pinfo 再初始化为 tcp_opt 结构，其中存储了许多与 TCP 相关的变量和数据结构，如拥塞窗口 snd_cwnd。sock 结构的 proto 指针链接到包含协议操作原语的 proto 结构。proto 结构中的每个成员都是一个函数指针。对于 TCP，将函数指针初始化为指向包含在 tcp_func 结构中的函数列表。任何想在 Linux 上编写自己传输协议的人应该遵循由 proto 结构定义的接口。

练习

如图 5-41 所示，在结构 sock 中的结构 proto 提供了链接到套接字的必要操作的函数指针列表，例如 connect、sendmsg 和 recvmsg。通过将不同函数集合链接到列表，套接字就可以通过不同的协议发送或接收数据。找到并阅读其他协议（如 UDP）的函数集合。

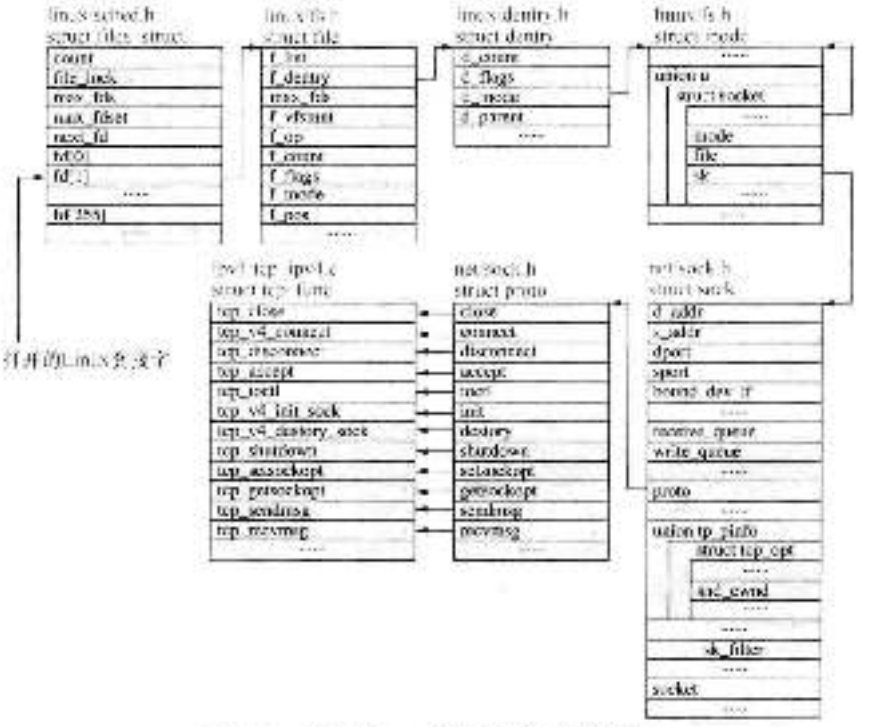


图 5-41 套接字 API 使用的内核数据结构

性能问题：套接字上的中断和内存复制

在套接字上接收分段实际上调用两个外理流程，如图 5-42 中的调用图所示。第一个流量从系统调用 read()开始，然后在 tcp_recvmsg()（对于 TCP 的情况下）等待，该需要由 sk_data_ready()来触发，以在返回用户空间时结束。因此，花费在流流上的时间会导致用户察觉到的延迟。第二个流是当分组到达时 tcp_v4_rcv()（对于 TCP 的情况下）被 IP 层调用开始的，并以调用 sk_data_ready()隐发第一个流的恢复作为结束。图 5-42 中显示了传输层接收 TCP 分段所花费的时间。tcp_recvmsg()负责将数据从内核结构复制用户缓冲区中，因此消耗了大部分时间（2.6 微秒）。系统调用 read()在用户模式和内核模式之间切换的花费时间。此外，也会在系统表现上花费时间。因此，read()花费大量的时间（2.4 微秒）。最后，在第二个流上，花费在 tcp_data_queue()和 tcp_v4_rcv()上的时间分别是排队和验证分段的有效性。

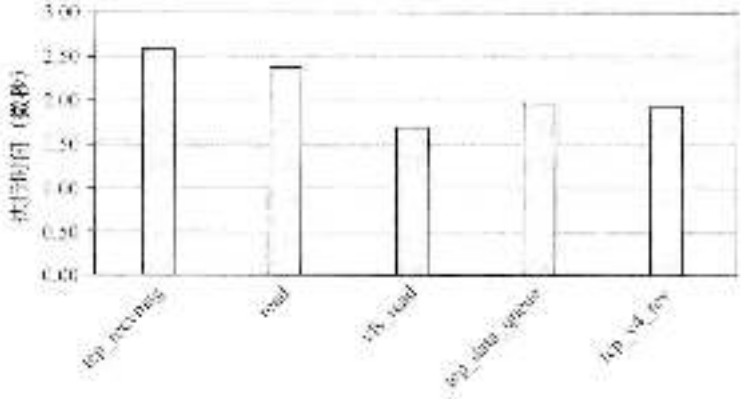


图 5-42 在 TCP 层接收 TCP 分段产生的延迟

图 5-43 显示了在传输 TCP 分段时所花费的时间。两个最终时的函数在功能上与接收情况相似。它

们分别是 tcp_sendmsg()，从用户缓冲区向内核缓冲复制数据；系统调用 write()，用户模式和内核模式之间的切换。在给定了 TCP 分组的传输和接收时间后，我们可以得出结论，TCP 层的延迟发生在两个地方：用户缓冲区和内核缓冲之间的内存复制，用户模式和内核模式之间的切换。

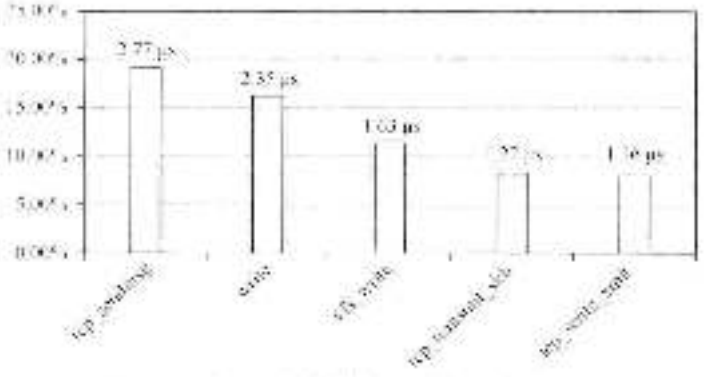


图 5-43 在 TCP 层中传输 TCP 分段产生的延迟

5.4.3 绕过 UDP 和 TCP 传输

有时，应用程序不希望使用传输层所提供的服务，ping 和 traceroute 等工具直接发送分组而不打开 UDP 或 TCP 套接字。它们仅使用 IP 层提供的服务。有些应用程序甚至绕过 IP 服务，并直接加载数据链路层通信。例如，分组嗅探实用程序，如 tcpdump 和 Wireshark，直接在线路上抽取原始分组。这样的应用程序需要打开完全不同于 UDP 或 TCP 的套接字。本节的目的在于探索在 Linux 下可以实现这些目标的编程方法。接下来我们将学习上述任务的 3 个开源实现。

开源实现 5.8：绕过传输层

概述

自 Linux 2.0 出现以后，就引入了一个新的协议族，称为 Linux 分组套接字（AF_PACKET）。允许应用程序直接发送和接收分组时直接访问卡的驱动程序来处理，而不是通常的 TCP/IP 或 UDP/IP 协议栈来处理。任何通过套接字发送的分组可直接传输到以太网接口，同时接口接收到的分组将直接传送给应用程序。

算法实现

AF_PACKET 族支持两个略有不同的套接字类型，分别是 SOCK_DGRAM 和 SOCK_RAW。前者请求和和删除以太网头部的重担留给了内核，而后者则让应用程序完全控制以太网头部。它们的实现在 net/packet/af_packet.c 中。通过检查结构变量 packet_ops，你可以找到对应的多路模传函数，如 packet_bind()、packet_sendmsg()和 packet_recvmsg()。

在 packet_recvmsg()中的代码很容易理解。首先，调用 skb_recv_datagram()经过 skb 的缓冲区获得分组。然后，分组数据由 skb_copy_datagram_iovec()复制到用户空间，稍后再传送给用户空间程序。最后，skb 由 skb_free_datagram()释放。

与 packet_recvmsg()相比，packet_sendmsg()有一个更复杂的程序。它首先检查链路层源地址是否已经由上层空间程序分配了。如果没有分配，它将根据存储在输出设备数据结构上的信息设置地址。然后由 sock_alloc_send_skb()分配 skb 缓冲区，用户空间数据将由 memcpy_fromiovec()复制到 skb 缓冲区中。如果套接字以 SOCK_DGRAM 类型打开，就调用 dev_hard_header()处理以太网头部。最后，分组将由 dev_queue_xmit()发送出去，skb 缓冲区将由 kfree_skb()释放。

用法示例

为了打开 AF_PACKET 族的套接字，在 socket()调用中的协议字段必须与/usr/include/

linux/if_ether.h中定义的以太网协议号相匹配。协议标识符表示可以在以太网帧中传输的注册过的协议。除非处理非常特殊的协议，通常使用ETH_P_IP，它将包括所有IP族协议（TCP、UDP、ICMP、原始IP等）。不过，如果你想要请求所有的分组，则使用ETH_P_ALL代替ETH_P_IP。如在下面例子中所示。

```
#include "stdio.h"
#include "unistd.h"
#include "sys/socket.h"
#include "sys/types.h"
#include "sys/ioctl.h"
#include "net/if.h"
#include "sys/time.h"
#include "stdlib.h"
#include "netinet/in.h"
#include "linux/if_ether.h"

int main()
{
    int i;
    int sd;
    char buf[2048];
    if((sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P
_ALL))) < 0)
    {
        printf("fail to open socket\n");
        return(1);
    }
    while(1)
    {
      n = recvfrom(sd, buf, sizeof(buf),0,0,0);
      if(n>0)
        printf("recv %d bytes\n", n);
    }
    close(sd);
}
```

由于AF_PACKET族的套接字具有许多特殊的安全问题，例如，你可以使用欺骗的MAC地址发送以太网帧，所以它们仅能被具有权限root权限的用户使用。

练习

修改和编译前面的例子将MAC头部字段转储到一个文件中，并为每个收到的分组标识传输协议。记住，你需要有机器的root权限才能运行。

分组捕获：混杂模式与非混杂模式

可以直接访问传输介质的任何人都能够捕获有线或无线网络上的分组。能够完成这种事情的应用程序称为分组嗅探器，通常用于测试网络应用程序以检查分组是否以正确的头部和有效载荷发送出去。当它们在网卡上被接收到时，AF_PACKET族允许应用程序检索数据分组，但它仍然不允许应用程序读取不是发给它主机的分组。正如我们前面学习过的，这是因为网卡将丢弃所有不包含自己MAC地址的分组——称为非混杂操作模式，其中每个网卡仅关心自己的事情并且只读取发送给它的帧。此规则有三个例外：

1）如果帧的目的MAC地址是特殊的广播地址（FF:FF:FF:FF:FF:FF），那么它可以被任何网卡接收。

2）如果帧的目的MAC地址是组播地址，那么它将被所有启用组播接收功能的网卡接收。

3）设置成混杂模式的网卡将会接收它能侦测到的所有帧。

开源实现5.9：设置混杂模式

概述

显然，上述最后三个例外是我们最感兴趣的。为了将网卡设置成混杂模式，我们只需要先将网卡上相关参数予以特定的ioctl()调用。由于这是一个可能危及安全的操作，只允许具有root权限的用户调用。如果"sock"命令一个已经打开的套接字，那么利用以下指令就能实现：

```
strncpy(ethreq.ifr_name,"eth0",IFNAMSIZ);
ioctl(sock, SIOCGIFFLAGS, &ethreq);
ethreq.ifr_flags |= IFF_PROMISC;
ioctl(sock, SIOCSIFFLAGS, &ethreq);
```

ethreq 是一种定义在/usr/include/net/if.h 中的 ifreq 结构。首先 ioctl 请求以获取网卡标志的当前值，然后标志与 IFF_PROMISC 进行或运算，来为混杂模式对写时到具有第二个 IOCTL 的网卡上。执行 ifconfig 命令很容易地进行检查，并观察输出的第三行。

算法实现

在调用系统调用 ioctl()后，将网卡设置成混杂模式会发生什么？当应用程序调用 ioctl()时，内核将调用 dev_ioctl()处理所有网络类型的 I/O 控制请求。然后，根据传入的系统将调用不同的函数来处理对应的任务。例如，当给定 SIOCSIFFLAGS 时，将调用 dev_ifsioc 设置对应于 sock 的接口标志。下一步，将调用 dev_set_promiscuity()通过网络设备驱动程序提供的回调函数 ndo_change_rx_flags()和 ndo_set_rx_mode()来改变设备的标志。当启用组播或混杂模式时，前一个函数允许设备接收方类改位置，而后一个函数则通知设备接收方有关地址列表过滤的更改。

练习

阅读网络设备驱动程序，看看 ndo_change_rx_flags()和 ndo_set_rx_mode()是如何实现的。如果你找不到实现它们的代码，那么驱动程序中启用混杂模式的地关代码在哪里？

内核中分组的捕获和过滤

作为应用程序和在用户空间中运行的进程，分组嗅探器进程在分组到达时不可能立即由内核调度。因此内核应该将它缓存在内核套接字缓冲器中直到分组嗅探器进程被调度为止。此外，用户可以指定分组过滤器仅嗅探对其感兴趣的分组。当分组在用户空间过滤时分组捕获的性能可能会降级，因为会有大量无关的分组将跨越内核用户空间边界的传输。如果在一个忙的网络上嗅探，这种嗅探器就不能在分组溢出套接字缓冲区之前及时捕获分组。把分组过滤器迁移到内核中将有效地提高性能。

开源实现 5.10：Linux 套接字过滤器

概述

tcpdump 程序通过命令行参数获取来自用户的过滤请求，以满足感兴趣的分组集合。然后 tcpdump 调用 libpcap（使用分组捕获库）访问合适的内核级分组过滤器。在 BSD 系统中，Berkeley 分组过滤器（BPF）在内核中执行分组过滤。Linux 一直没有自己的内核分组过滤，直到在 Linux 2.0.36 中出现 Linux 套接字过滤器方法。BPF 和 LSF 之间差大同小异，除了一些细微的差别，如用户访问该服务的权限。

框图

图 5-44 中给出了一个用于分析捕获和过滤的分层模型。到达的分组从正常的协议栈复制到 BPF，然后根据相应的应用程序安装的 BPF 指令在内核中过滤分组。由于只有通过 BPF 的分组才会被导向到用户空间程序，所以用户空间和内核空间之间数据交换的额外开销可以显著地降低。

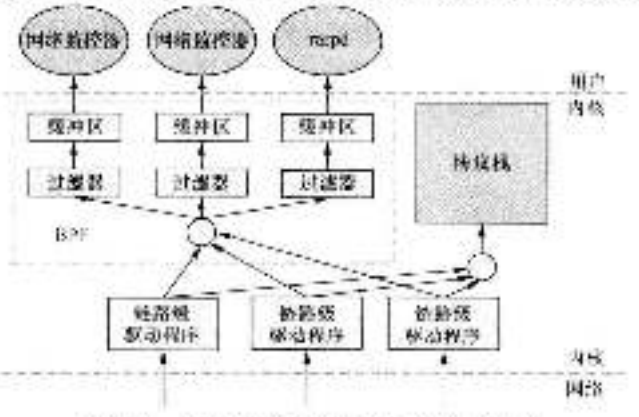


图 5-44 实现有效的分组过滤：分层的模型

为了对套接字应用 Linux 套接字过滤器，可以使用 socket.c 中实现的 setsockopt 日函数将 BPF 指令传递给内核，并为 SO_ATTACH_FILTER 设置参数 optname。该函数将 BPF 指令分配给图 5-41 所示的 sock > sk_filter。BPF 分组过滤引擎是用一种特定的伪码语言编写的，该语言是由 Steve McCanne 和 Van Jacobson 开始导的。BPF 实际上看起来更像一种真正的汇编语言，带有一些寄存器和几个指令来加载和存储值并执行算术运算和有条件的分支。

过滤器代码在连接的套接字上检查每个分组。过滤器处理结果是一个整数，表示套接字应该将多少字节的分组传递给应用层。这是切功于进一步的优点，因为通常所感兴趣的分组消费和过滤目的只不过是一个短短几字节的分组，如果不复制多余的字节，那么就能够节省处理时间。

练习

如果你阅读 tcpdump 的主页，你将会发现 tcpdump 能够生成人们易读的 BPF 代码或 C 程序分段，根据给定的过滤条件：如 tcpdump-d host 192.168.1.1。首先计算生成的 BPF 代码。然后，写一段程序打开原始套接字（参见开源实现 5.8），打开混杂模式（见开源实现 5.9），使用 setsockopt 将 BPF 代码注入到 BPF 中，然后观察是否确实从套接字仅收到匹配给定过滤器的分组。

5.5 用于实时流量的传输协议

目前类到的传输协议都不是为适应实时流量要求而设计的。那么就必须有一些其他的精密机制在互联网上传输实时流量。本节首先强调实时流量所强制施加的要求。

到目前为止，TCP 能满足非实时数据流量所规定的所有要求，包括差错控制、可靠性、流量控制和拥塞控制。然而，实时流量既不能由 TCP 满足也不能由 UDP 来满足，因此开发了多个其他的传输协议。其中最流行的是 RTP 和它的伙伴协议 RTCP。由于这些传输协议还没有成熟到广泛部署的程度，所以它们没有在内核中实现而只是常作为一个应用程序调用的函数库实现。许多实时应用，如 Skype 和网络收音机，可以调用这些库函数，然后在 UDP 上传输数据。由于解决的需求实际上是传输层问题，因此我们将它们放在本章讨论而不是放在第 6 章。

5.5.1 实时需求

实时流量经常将多种流（如视频、音频和文本等）放在一个会话中传输。会话是传输这些数据流的一组连接。因此，第一个新的需求就是需要在一个会话中同步多个数据流。同步还需要在发送方和接收方之间传输，播放流时实现。两种同步要求在发送方和接收方之间传递定时消息。此外，实时流量对穿越不同网络的移动性所导致的中断更为敏感。因此，在移动性下支持服务连续性成为第二个新的需求。

实时流量是连续的，因此需要一个稳定的或平滑的、无带后传播的可用速率。但它仍然需要平滑的拥塞控制以便保持互联网的健康并对自我调节的 TCP 流量友好。这使得平滑性和 TCP 友好性成为第三个需求。有些实时流量自适应性很好，它甚至能够改变媒体编码率，即当可用速率波动时编码 1 秒钟内容需要的平均位数。这当然是为发送方提供路径质量报告而收集数据的第四个需求。

不幸的是，还没有一个流行的传输协议能够满足所有这四个实时性需求。正如我们接下来将看到的，每个协议或多或少地满足了一些需求。RTP 和 RTCP 满足第一和第四个需求，似乎是最流行的实时传输协议。

多流和多穴

另一种传输协议，流控制传输协议（SCTP），是由 R. Stewart 和 C. Metz 在 RFC 3286 中引入的并在 RFC 4960 中定义。像 TCP 一样，它为数据传输提供了一种可靠的信道，并使用相同的拥塞控制算法。然而，由于术语“流”出现在 SCTP 中，所以 SCTP 提供两个非常有利于流应用的额外属性，也就是支持多穴和多流。

支持多流意味着多个数据流，如音频和视频，可以并发地通过一个会话传输。也就是说，SCTP 能够支持独立有序地接收每个数据流，避免可能发生在 TCP 上的行头（Head-Of-Line，HOL）阻塞。在 TCP 中，控制消息或某些重要的消息常常被阻塞，因为大量分组挂在发送或接收缓冲区的前面。

支持多穴意味着甚至当一个移动用户从一个网络移动到另一个网络时，他将不会丢失它所接收到流的任何一部分。为了支持多穴属性，一个 SCTP 会话可由多条连接开发组成，通过不同的网络适配器。例如，以太网和无线局域网。此外，对于每条连接还有一个心跳消息，以确保其连接性。因此，当一条连接出现故障时，SCTP 就可以立即通过其他连接传输流量。

SCTP 协议还可以修改 TCP 连接的建立和关闭步骤。例如，建议连接建立采用四次握手机制，以便克服 TCP 的安全问题。

平滑速率控制和 TCP 友好性

虽然 TCP 流量在互联网上仍然占主导地位，但研究表明在大多数 TCP 版本中使用的拥塞控制机制可能会导致过大的传输速率振荡而无法满足有延时动要求的实时流量。由于 TCP 可靠不适合实时应用，所以开发人员常常减少其中的拥塞控制，甚至避免使用拥塞控制。这样的做法导致互联网导向担心，因为互联网的带宽是共享的，没有控制机制来决定一个主在互联网上应该使用多少带宽，过去这一直靠由 TCP 进行自我控制。

1998 年，在 RFC 2309 中建议了一种 TCP 友好的概念。这个观念的主要思想是在面临相同的网络条件如（相同的分组丢失率和往返时间）时，流应该在传输状态对拥塞状态做出响应，并且使用不会比处于稳定状态 TCP 流更多的带宽。这样的概念要求任何互联网流使用拥塞控制机制并且不会比其他 TCP 连接使用更多的带宽。不幸的是，在这个问题上并没有明确哪种拥塞控制是最佳的。在这种情况下，E. Kohler 等人在 RFC 4340 中提出了一个新的协议，称为数据报拥塞控制传输协议（DCCP）。DCCP 允许对于拥塞控制方案有自主选择权。该协议目前仅包含两个方案，类似于 TCP 和 TCP 友好速率控制（TFRC）。TFRC 最早在 2000 年提出并在 RFC 3448 更新中定义为一种协议，它详细阐述了应该在两个终端设备之间交换什么信息来满足 TCP 友好性。

行动原则：流采用 TCP 还是 UDP?

为什么 TCP 不适用于流传输？首先，重传机制紧密地嵌入 TCP 中，这对流传输而言并非必需的甚至会增加流数据的延迟和抖动。其次，特别的速率成为可能不利于流数据传输。也就是说，尽管对于流数据，对可用带宽的估计是选择编码速率所必需的，但流数据不适用于震荡的传输速率，尤其不适用于分组丢失的激烈响应。因为它原本设计用于避免潜在的连续分组丢失。流应用程序可以接受也可以放弃处理分组丢失问题。由于在 TCP 中的某些机制并不适用于流数据，所以人们就自然转向通过 UDP 传输流数据。遗憾的是，UDP 太简单，不提供可估算当前可用速率的机制。此外，由于安全性考虑，UDP 分组有时也会被中间网络设备丢掉。

尽管 TCP 和 UDP 都不适用于流数据，但它们仍不失为当今互联网仅有的两个成熟的传输协议。实际流数据的确还是由这两个协议传输。UDP 用于传输纯的音频数据流，比如音频和 VoIP。这些流数据可以简单地用恒定速率发送而无需太多拥塞控制，因为它们所需要的带宽通常低于可用的带宽。另一方面，TCP 用于互联网不断总是满足所需带宽的流传输——例如，视频和音频的混合。然后，为了减轻 TCP 的震荡速率（带宽检测机制的负效应），接收方使用一个大的缓冲区，这会加大延迟。尽管延迟对于单向应用是可以忍受的，例如，在 YouTube 上面看视频剪辑，但对于像视频会议这样的交互式应用程序却不是可以忍受的。这就是为什么开发人员宁愿要开发 UDP 上所述的平滑速率控制机制的原因。

回放重构和端径质量报告

由于互联网是共享数据报网络，所以在互联网上传输的数据会有不可预测的延迟和抖动。但是，实时应用（如在 IP 上的语音（VoIP）和视频会议）需要适当的定时信息来重构在接收方的回放。在接收方的重建要求编解码类型选择正确的解码器以便解压缩有效载荷，时间戳重构原来的定时以便以正确的速率播放数据，序列号用于按顺序正确地放置到达的数据分组并用于分组丢失检测。另一方面，实时应用程序的发送方也需要来自接收方的路径质量反馈以便对网络拥塞做出响应。此外，在组播环境下，成员信息也需要管理。这些控制平面机制也应该构建到标准协议中。

总之，实时应用程序的数据平面需要处理编解码器、序列号及时间戳；控制平面的重点放在端到端延迟/抖动/丢失的反馈报告和成员管理上。为了满足这些需求，已提出了在接下来两节中将要介绍

的 RTP 和 RTCP。注意，RTP 和 RTCP 经常由应用程序自身来实现而不是由操作系统实现。因此应用程序具有对每个 RTP 分组的完全控制，如 RTP 头部选项字段定义。

5.5.2 标准数据平面协议：RTP

在 RFC 1889 中描述了一种数据平面协议：实时传输协议（RTP）。它是用来传输语音/视频流量类同等起网络低协议。RTP 没有在发现与制定或口，因为它与部够与端口标识的不同应用程序一同工作。因此它可以在一个 UDP 端口上运行，以 5004 为作为默认端口。RTP 与辅助协议 RTCP 一起用来获取有关数据传输质量和参与正在进行会话信息的反馈。

RTP 是如何工作的

RTP 消息由当部和有效载荷组成。图 5-45 显示了 RTP 首部格式的内容。实时流量是在 RTP 分组中的有效载荷里携带的。注意，RTP 本身并没有处理资源管理和预留，并且也不能保证实时服务的服务质量。RTP 假设这些属性（如果提供的话）由底层的网络提供。虽然互联网偶尔会丢失乱序或延迟分组，或者延迟一个可变的时间量，为了处理这些问题，RTP 头部包含时间戳信息和序列号，这样就可以允许接收方重建由源产生的定时。利用这两个字段，RTP 就能够保证分组是按序的、确定是否有分组丢失、并同步通信流。序列号对于每个 RTP 分组发送都会递增 1，时间戳反映了取样的 RTP 数据分组中的第一个字节。取样瞬间必须是从单调线性增加时间的时钟上获取的以便允许同步和抖动计算。也就是说，当视频分解为多个 RTP 分组时，它们都有相同的时间戳，这就是为什么时间戳不足以为分组进行重新排序的原因。



图 5-45 RTP 头部格式

包含在 RTP 头部的一个字段是 32 位同步源标识符（SSRC），它能够区分在同一个 RTP 会话中的同步源。由于多个音频/视频流可以使用同一个会话，所以在接收应用程序上为了同步目的用 SSRC 字段标识信息的发送方。原则上选择数以使得在两个同步源不在同一个 RTP 会话中使用同一个号。例如，分支机构 B 的会使用 VoIP 网关来建立两者之间的会话。然而，在每一边都安装了很多电话，因此 RTP 会话可能同时包含了很多呼叫连接。这些呼叫连接可以通过 SSRC 字段进行多路复用。

编解码器封装

为了在接收方重建实时流量，接收方必须知道如何翻译它所接收的分组。有效载荷类型标识符指明了有效载荷的格式和编码/压缩方案。有效载荷类型包括 PCM、MPEG1/MPEG2 音频和视频、JPEG 视频和 H.261 视频流。在任何给定的传送时间内，RTP 发送方只能发送一种类型的有效载荷，尽管这种有效载荷类型在传输期间可以发生改变，比如，满够网络拥塞。

5.5.3 标准控制平面协议：RTCP

RTCP 是与 RTP 一起使用的控制协议。它在 RFC 1889 和 RFC 1890 中标准化。在 RTP 会话中，参与者周期性地发送 RTCP 分组来传送有关数据发送数量的反馈和有关成员的信息。RFC 1889 定义了 5 种 RTCP 数据包类型：

1）**RR**：接收方报告。接收方报告是由 RTP 会话中不活跃的发送方发送的。其中包括了有关数据发送的接收质量反馈，包括接收的最大分组序列号、分组丢失数、到达间隔抖动（inter-arrival-jitter）和用来计算发送方和接收方之间往返时延的时间戳。这些信息对于自适应编码很有用。比如，随着时间的流逝，如果 RTP 会话质量恶化，发送方也许会切换到更低位速率的编码，这样用户会感觉实时传输更平滑些。另一方面，网络管理员通过监督 RTCP 分组来评价网络的性能。

2）**SR**：发送方报告。发送方报告由活跃的发送方产生。除了在 RR 中的接收质量反馈外，SR 还包括一个发送方信息部分，提供媒体间同步、累计分组计数和发送的字节数。

3）**SDES**：用来描述源的源描述项。在 RTP 数据分组中，源是由随机产生的 32 位标识符来标识的。这些标识符对于用户有些不便。RTCP SDES 包括会话参与者的独一的标识符。它们可以包括用户姓名、电子邮箱地址或者其他信息。

4）**BYE**：指示参与者的结束。

5）**APP**：特定应用的函数。APP 专门用于为新应用程序或者新特性的测试。

由于参与者随时可以加入或者退出会话，所以知道谁参与会话以及这些参与者的发送质量如何很重要。因此，不活跃参与者应该周期性地发送 RR 分组并在退出时发送 BYE。另一方面，那些活跃的发送方应该发送 SR 分组，这不仅提供与 RR 数据相同的功能而且还能保证每个参与者都懂得如何再次重放接收到的媒体数据。最后，为了帮助这些参与者得到更多关于其他参与者的信息，参与者还应该周期性地发送含有标识符号的 SDES 分组来介绍它们的联系信息。

历史演变：RTP 实现资源

RTP 是一个不提供预实现系统调用的开放协议。实现整密地与应用程序结合起来。应用程序开发者必须亲自在应用层添加完整的功能。然而，共享和应用代码而不是从头开始编码讨论更加有益。在 RFC 1889 规范中包含了无数可以直接被应用程序使用的代码段。这里我们提供一些可用的源代码实现。对在源代码中的许多模块略做修改就可以使用。下面是一张可用资源的列表：

- RFC 1889 中的独立样例代码。
- **vat**（http://www-nrg.ee.lbl.gov/vat/）。
- **tptools**（ftp://ftp.cs.columbia.edu/pub/schulzrinne/rtptools/）。
- **NeVoT**（http://www.cs.columbia.edu/~hgs/rtp/nevot.html）。
- **RTP Library**（http://www.iasi.rm.cnr.it/iasi/netlab/gettingSoftware.html）E. A. Mastromartino 提供了将 RTP 功能集成到 C++ 互联网应用程序的便利方法。

5.6 总结

在本章中，我们首先学习了提供通过互联网的进程到进程信道的传输层的 3 个主要特点：1）端口寻址；2）可靠的分组传输；3）流速率控制。然后我们学习了不可靠的无连接传输协议 UDP 和广泛使用的传输协议 TCP。与仅在 IP 层上添加一个端口寻址功能的 UDP 相比，TCP 就像具有多种经过良好验证过的技术的完整解决方案，具体包括：1）用于连接建立和释放的三次握手协议；2）确认和重传机制，确保接收方准确无误地接收从位于数千米之外的发送方发送的数据；3）提高吞吐量、降低分组丢失率的滑动窗口流量控制和演变的拥塞控制算法。我们举例说明了各种 TCP 版本，在可能分组丢失重传的情况下，比较了它们的性能。最后我们学习了有关多流、多穴、平滑速率控制、TCP 友好性、回放重构和路径质量报告等的传输层协议的需求和问题。

除了协议外，本章还介绍了实现套接字的 Linux 方法并描述了它们的函数调用。套接字接口是内核空间网络协议和用户空间应用程序之间的分界线。因此利用套接字接口，应用程序开发人员就可以将精力集中在想要通过互联网发送或者接收什么内容，而不用处理网络协议中复杂的匹层协议和内核问题，因此大大降低了开发人员的负担。在第 6 章中，我们将看到有趣和日常使用的应用程序，包括电子邮件、文件传输、万维网、即时文件/语音通信、在线音频/视频流和对等应用程序。它们是在 UDP、TCP 或者同时在它们两者之上进行的。

常见陷阱

窗口大小：分组计数模式与字节计数模式

不同的实现会对 TCP 标准有不同的解释。读者也会对分组计数模式和字节计数模式的窗口大小感到迷惑。尽管接收方报告的接收窗口是以字节为单位的，但是前面有关 cwnd 的示例是以分组为单位的，因此为了从 min(cwnd, rcvd) 选择窗口大小就要通过乘以 MSS 转换为字节。有些操作系统直接使用字节计数模式的 cwnd，因此算法应该调整如下：

```
if (cwnd < ssthresh){
    cwnd = cwnd + MSS;
else {
    cwnd = cwnd + MSS*MSS/cwnd
}
```

也就是说，在慢启动阶段，不是采用分组计数模式的 cwnd 递增 1，而是当收到一个 ACK 确认时，我们就在字节计数模式增加 1 个 MSS。在拥塞避免阶段，不是在分组计数模式下将 cwnd 增加 1/cwnd，而是在接到一个 ACK 确认时，增加 MSS/cwnd。

RSVP、RTP、RTCP 和 RTSP

本章讨论了在互联网上将 RTP 协议和 RTCP 协议用于实时流量。然而，它们与相关协议（如 RSVP 和 RSTP）之间的不同，有必要加以澄清：

- RSVP 是一个信令协议，用以通知沿路径的网络元素为实时应用预留足够的资源，如带宽、计算能力、排队空间等。它并不能传送数据。RSVP 将在第 6 章中学习。
- RTP 是一个用于实时数据的传输协议。它提供时间戳、序列号和其他处理实时数据传输中的定时问题。它依赖于 RSVP（如果支持）进行资源预留以便提供服务质量。
- RTCP 是一种与 RTP 一起使用的控制协议，具有服务质量和成员管理。
- RTSP 是一种控制协议，能够发起和引导来自多媒体服务器的多媒体数据流的发送。它是“互联网 VCR 远程控制协议”。它用于提供远程控制。实际数据传送靠 RTP 一类单独地完成。

进一步阅读

TCP 标准

TCP 的头部和状态图最早由 Postel 在 RFC 793 中定义，但是因为拥塞控制在互联网早期并不是问题，所以 TCP 的拥塞控制技术后来才由 Jacobson 提出和修正。TCP 拥塞控制的观点在 Zhang、Shenker 和 Clark 等人的研究工作中给出，而对主机系统实现的需求和对 TCP 的其他更正在 RFC 1122 中给出。Stevens 和 Paxson 标准化了 TCP 一些拥塞控制的四种关键行为。SACK 和 ACK 分别定义在 RFC 2018 以及由 Mathis、Mahdavi 发表的论文中。Nagle 算法和 Clark 解决糊涂窗口综合症的方法，分别在 Nagle 的 SIGCOMM’84 论文和 RFC 813 中描述。

- J. Postel, “Transmission Control Protocol,” RFC 793, Sept. 1981.
- V. Jacobson, “Congestion Avoidance and Control,” *ACM SIGCOMM* pp. 273–288, Stanford, CA, Aug. 1988.
- V. Jacobson, “Modified TCP Congestion Avoidance Algorithm,” mailing list, end2end-interest, 30 Apr. 1990.
- L. Zhang, S. Shenker, and D. D. Clark, “Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic,” *ACM SIGCOMM*, Sept. 1991.
- R. Braden, “Requirements for Internet Hosts—Communication Layers,” STD3, RFC 1122, Oct. 1989.
- W. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,”

RFC 2001, Jan. 1997.

- V. Paxson, "TCP Congestion Control," RFC 2581, Apr. 1999.
- M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.
- M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," *ACM SIGCOMM* pp. 281 – 291, Stanford, CA, Aug. 1996.
- J. Nagle, "Congestion Control in IP/TCP Internetworks," *ACM SIGCOMM*, pp. 11-17, Oct. 1984.
- D. D. Clark, "Window and Acknowledgment Strategy in TCP," RFC 813, July 1982.

有关 TCP 版本

前两篇论文比较了不同版本的 TCP，第一篇论文介绍了 TCP Vegas，而最后两篇论文研究并提供了在高带宽延迟乘积网络中拥塞控制应用的解决方案。

- K. Fall and S. Floyd, "Simulation-Based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, Vol. 26, No. 3, pp. 5 – 21, Jul. 1996.
- J. Padhye and S. Floyd, "On Inferring TCP Behavior," in *Proceedings of ACM SIGCOMM*, pp. 287 – 298, San Diego, CA, Aug. 2001.
- L. Brakmo and L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, pp. 1465 – 1480, Oct. 1995.
- D. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, Vol. 14, No. 6, pp. 1246 – 1259, Dec. 2006.
- D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM*, pp. 89 – 102, Aug. 2002.

TCP 吞吐量建模

两个广泛引用的 TCP 吞吐量公式是在以下两篇论文中提出的。给出分组丢失率、RTT 和 RTO，这些公式将给出 TCP 连接的平均吞吐量。

- J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *ACM SIGCOMM*, Vancouver, British Columbia, Sept. 1998.
- E. Altman, K. Avrachenkov, and C. Barakat, "A Stochastic Model of TCP/IP with Stationary Random Losses," *IEEE/ACM Transactions on Networking*, Vol. 13, No. 2, pp. 356 – 369, April 2005.

柏克利分组过滤器

以下是 BDS 分组过滤器的起源。

- S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-Level Packet Capture," *Proceedings of the Winter 1993 USENIX Conference*, pp. 259 – 269, Jan. 1993.

用于实时流量的传输协议

前两篇参考文献给出了用于流通信量的协议，而后两篇参考文献是控制流经互联网流通信量吞吐量的经典 TCP 友好拥塞控制算法。

- R. Stewart and C. Metz, "SCTP: New Transport Protocol for TCP/IP," *IEEE Internet Computing*, Vol. 5, No. 6, pp. 64-69, Nov/Dec 2001.
- S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *ACM SIGCOMM*, Aug. 2000.
- Y. Yang and S. Lam, "General AIMD Congestion Control," *Proceedings of the IEEE ICNP 2000*, pp. 187-98, Nov. 2000.

- E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion Control Without Reliability," *ACM SIGCOMM Computer Communication Review*, Vol. 36, No. 4, Sept. 2006.

NS2 模拟器

NS2 是互联网研究社区广泛使用的网络模拟器。

- K. Fall and S. Floyd, ns - Network Simulator, http://www.isi.edu/nsnam/ns/.
- M. Greis, "Tutorial for the Network Simulator ns," http://www.isi.edu/nsnam/ns/tutorial/index.html.

常见问题解答

1. 对比第 2 层信道与第 4 层信道（比较它们的信道长度、差错和延迟分布）。

   答：信道长度：链路与路径。

   信道差错：链路与链路和节点。

   信道延迟分布：密集与稀疏。

2. TCP 与 UDP 对比（比较它们的连接管理、差错控制和流量控制）。

   答：连接管理：TCP 有，UDP 没有。

   差错控制：UDP：可选的校验和，无 ACK，无重传机制。

   TCP：校验和、ACK、序列号和重传机制。

   流量控制：UDP：无流量控制。

   TCP：动态窗口大小用来控制传输中的未经确认的字节，具体根据网络状况和接收方缓冲区占用情况而定。

3. 为什么大多数实时流量通过 UDP 传输？

   答：大多数实时传输能够忍受某些丢失但并不需要延时重传。它的位速率取决于发送方的编解码器而不受流量控制机制的影响。

4. 在 TCP 中需要什么机制以便支持差错控制？

   答：校验和、ACK、序列号和重传机制。

5. 为什么 TCP 需要三次握手而不是两次？

   答：两边都需要通知和确认来自每一边的起始序列号。第一个 ACK 已经结合了第二个通知。因此我们就拥有三个分段：第一个通知、第一个 ACK（第二个通知）和第二个 ACK。

6. 什么时候重传丢失的 TCP 分段？

   答：三个重复 ACK（快速）或者 RTO（重传超时）（卷）。

7. 在确定 TCP 窗口大小时应该考虑哪些因素？

   答：最小值（拥塞窗口大小、接收窗口大小），其中拥塞窗口大小运行 AIMD 算法（加法增加乘法减少）并且接收方窗口大小就是通告的接收方可用的缓冲区空间。前者是关于网络状况的，而后者是关于接收方状况的。

8. 在卷启动和拥塞避免算法中，窗口是如何增长的？

   答：慢启动：是以 1、2、4、8 等的指数形式增加的。

   拥塞避免：从 32 到 33、34、35 等的线性增加。

9. 为什么要将快速重传和快速恢复添加到 TCP 中？新的 Reno 做了哪些重大更改？

   答：快速重传：收到了三个重复的 ACK 时，就重传，比 RTO 更早。

   快速恢复：在丢失恢复期间维持自同步时钟行为（通过具有足够大的窗口大小发送新的分段）。

   新的 Reno：在检测到三重 ACK 被确认之前扩展快速恢复阶段（具有足够大的窗口尺寸）直到所有分段发送完为止，这样就加速了多分组丢失的恢复。

10. 在 Linux 中，套接字是如何实现的？（简单描述套接字函数的处理流和套接字的数据结构。）

    答：处理流：在客户机或服务器的套接字函数调用是产生软件中断的系统调用，它强制系统进入内核模式并执行它用过的内核函数来处理中断。这些内核函数在内核空间缓冲区，即 ak_buff 和用户空间

之间移动数据。

数据结构：在文件系统中存储 Tracer 结构。

11. Linux 的内核空间和用户空间程序之间通过什么向应用来过滤和抽取分组？

答：内核空间：Linux socket filter。

用户空间库：Libpcap。

用户空间工具：tcpdump，wireshark 等。

12. 在 RTP 和 RTCP 上可以实现哪些不通过 UDP 的额外支持？

答：RTP：端到端时钟、用于延迟敏感应用的序列号、以及回收机制。

RTCP：端到端、诊断、告知发送方发送速率以便调节编码器的发送率。

练习

动手练习

1. NS-2 是用于 TCP 研究最受欢迎的模拟器。NS-2 包含一个称为 NAM 的软件包，它能够以任意时间刻度可视化地回放仿真。在互联网上可以找到关于 NS-2 的介绍。使用 NAM 观察一个从源到目的地的 TCP 运行，在一台中间路由器上带有或者没有缓冲区。

- 第一步：查找 NS-2 对你计算机对用户目标平台的版本。
- 第二步：按照安装提示来安装所有软件包。
- 第三步：构建一个包含三个端系统节点的方案。一个 p 点用于 Reno TCP 数据源，一个用于中间网关，一个用于目的地。将它们连接起来的链路是全双工 1Mbps。
- 第四步：配置一台具有大缓冲区的网关，并在目的地运行一个 TCP 数据源。
- 第五步：配置一台具有小缓冲区的网关，并在目的地运行一个 TCP 数据源。

对一个首选平台测试中 Reno TCP 源输入所在 Reno TCP 状态，并将输出写入跟踪文件。跟踪文件应该包含 TCP 数据源所处的状态。这些图形必须是标准的，例如，为了表示慢启动行为，你可以画出下列图形显示：1）一个 ACK 正在返回；2）ACK 触发两个新的数据分组。认真打标这些图形以便实现该练习的结果不多于一张 A4 纸。仅显示在屏幕转储中的必要信息。仅处理图形以便图中不显示窗口的装饰（窗口边缘，NAM 按钮）。

2. 当控制需要平滑快速波动的值时，你常常需要使用指数加权移动平均（Exponential Weighted Moving Average，EWMA）。典型的应用程序平滑处理测量的往返时间，或者计算在随机早期检测（RED）队列中的平均队列长度。在这个练习中，希望你能运行并观察 EWMA 程序的结果。改变网络流量参数来观察 EWMA 值是如何变化的。

3. 重新生成到 5-24。

- 第一步：为实验打补丁；日志记录打印拥塞的 CWND/SeqNum。
- 第二步：重新编译；
- 第三步：安装新内核并启动。

4. 当你想要产生任意一种分组类型时，Linux Packet Socket（分组套接字）就非常有用。查找并更改一个示例程序以便产生一个分组并使用这一程序发送分组。

5. 在 FREE-BSD 5.X 稳定版中按照直传式时钟管理。它是如何管理定时器的？以一张表或表形式将它与 Linux 2.6 做对比。提示：你可以通过阅读和跟踪 FreeBSD 的 sys/netinet/tcp_timer.c 中的 tcp_timer_rexmt() 函数和 Linux 的 net/ipv4/tcp_timer.c 中的 tcp_retransmit_timer() 函数的调用路径来开始本练习。

6. Linux 如何将 NewReno、SACK 和 FACK 集成在一起？找出在 5.3.3 节中提到的变量上的主要不同之处并且说明 Linux 是怎样解决这些冲突？

7. 在 Skype、MSN 或者其他通信软件中使用什么传输协议？请使用 wireshark 观察它们的流量并找出答案。

8. 在 MS Media Player（微软媒体播放器）或 RealMedia 中使用的传输协议是什么？请使用 wireshark 观察它们的流量并找出答案。
9. 利用套接字接口编写一个客户机/服务器程序。一旦用户键入回车键后，客户机将发送一个问给服务器，服务器用包含任何意义的词作为应答。但是，当服务器收到词 bye 后，就会断开服务。同样，一旦有人输入"给我你的视频"，服务器将立即以 500 字节消息大小发送 50MB 信息数据。
10. 编写一个客户机/服务器程序或者修改第 9 题中的程序，每幅 0.1 秒计算和记录数据传输率，对于 50MB 数据以 500 字节消息大小传输。并用 xgraph 或 gnuplot 来显示结果。
11. 继续进行第 9 题中的工作。修改客户机程序，利用一个没有会接字过滤器的会接字来过滤所有包含"the_packet_is_infected"的分组。然后比较 50MB 数据传输由会接字提供的平均传输速率和由客户机仅在用户层丢弃消息的平均传输速率。提示：开源实现 5.10 提供了有关如何在会接字内嵌入会接字过滤器的信息。
12. 修改第 9 题中所编写的程序以便创建一个基于 SCTP 的套接字，验证语音表话可以持续而不会因为传输大的文件而产生阻塞，即演示来自 SCTP 中的多流。提示：你可以在搜索引擎中输入关键字"SCTP multi-streaming demo code"在互联网上找到演示代码。

书面练习

1. 比较数据链路层、IP 层和端到端层之间差错控制的作用。数据链路层技术选择以太网作为讨论的主题。用一张带有关键词的表格来比较其目的，包含的字段、算法、字段长度和其他相同/不同的属性。为什么在整个分组的生命周期中包含如此多的差错控制？列出你的理由。
2. 比较数据链路层、IP 层、端到端层和实时传输层之间寻址的作用。对于链路层技术，使用以太网进行讨论。在实时传输协议中，选择 RTP 来讨论。用一张填写关键字的表比较目的，唯一性、分配/层次化和其他属性。
3. 比较数据链路层和端到端层之间流量控制的作用。数据链路层技术选择交换以太网。用一张填写关键字的表比较目的、流量控制算法、拥塞控制算法、重传定时器/算法和其他一些重要属性。还要进一步解释不重要的表项。
4. 一个移动的 TCP 接收方正在从 TCP 发送方中获取数据。当接收方逐渐来越远（距离发送方）然后再逐渐接近距离时，RTT 和 RTO 是如何变化的？假定移动速率非常快，因此传播延迟在 1 秒内从 100ms 变为 300ms。
5. 一条运行 TCP 的连接在一条具有 500ms 传播延迟的路径上传输，没有被中间网关所阻塞。当没有使用窗口大小缩放选项时，最大吞吐量是多少？当使用窗口大小缩放选项时，最大吞吐量是多少？
6. 假定 TCP 连接的吞吐量与其往返时间 RTT 成反比，共享同一队列的不同 RTT 的连接享有不同的带宽值。如果传播时延分别为 10ms、100ms、150ms，同时共享队列的服务速率为 200kbps，那么三个连接中最终带宽共享比例分别是多少？假设队列长度无穷大并且没有缓冲区溢出（无分组丢失），TCP 发送方的最大窗口为 20 个分组，每个分组包含 1500 字节。
7. 如果把第 6 题中的共享队列的服务速率改为 300kbps，那么答案将是多少？
8. 如果由 TCP 发送方保持的平滑 RTT 是当前的 30ms，但接下来测量的 RTT 分别为 26、32 和 24ms，那么新的 RTT 估计是多少？
9. TCP 提供一种可靠的字节流，但是需要由应用开发者来对客户机和服务器端之间的数据成"帧"。如果 TCP 分段通过 IP 分组来携带，那么它的最大有效载荷为 65 495 字节。为什么还要选择这样一个奇怪的数字呢？为什么绝大多数 TCP 发送方发送小于 1460 字节的分组呢？比如，尽管客户机经过 write() 能够发送 3000 字节的分组，但是服务器却只能读取 1460 字节的分组。
10. 在绝大多数的 UNIX 系统中，拥有 root 权限执行直接访问互联网层或链路层的程序很重要。但是有些类似于 ping 和 traceroute 的常用工具却仅使用普通用户表导就可以访问互联网层。在这个悖论的背后又有怎样的含义？你如何使自己的程序像上述工具那样直接访问互联网层？简要提出两个解决方案。
11. 使用一张表来比较并解释可以被 Linux 2.6 支持的所有套接字字段、类型和协议。

12. 除了集成了时间戳外，RTP 还集成了序列号字段。RTP 可以设计成去掉序列号字段并使用时间戳来重排不按序接收的分组吗？（回答是/不是，为什么？）
13. 假设读者正在准备设计一个利用 RTP 运行在 TCP 之上而不是 UDP 之上通过互联网传输的实时流应用程序。如图 5-21 所示的每种 TCP 拥塞控制状态中，发送方和接收方会遇到什么情形？以表格的形式比较你所期望的情况与在 UDP 之上设计的情况。
14. 从图 5-21 中，延迟分布在单跳和多跳环境中的同样问题却要求不同的解决方案。如果重传信道是 1 跳、2 跳和 10 跳时，那么延迟分布如何变化？画出三个类似于图 5-21 所示的关联延迟分布图，以便更好地演示增加跳数（例如 1 跳、2 跳和 10 跳）的最佳步骤。
15. 当对分段添加一个每个分段的校验和时，在分段传递给其下层。即 IP 层之前 TCP/UDP 都会包含一些在 IP 头中的字段。TCP 和 UDP 如何知道 IP 头部的值？
16. 本书从某种程度上详细介绍了不同版本的 TCP。找出三个其他的 TCP 版本。逐项填写详细记录并用 3 行以内的文字重点说明每个 TCP 版本做出的贡献。
17. 如图 5-41 所示，在 Linux 2.6 中的许多部分并非特定于 TCP/IP，例如，读/写函数和套接字结构。以 C 程序员的观点，分析 Linux 2.6 如何组织函数和数据结构以便于很容易地初始化为不同的协议。为了达到目的，简要说明基本 C 语言的编程机制。
18. 正如 5.5 节所述，许多协议和算法都是为了处理在互联网上携带流数据的挑战而提出的，请找出支持流媒体数据在互联网上传输的开源解决方案。然后，观察这些解决方案，观察是否能够处理以及如何处理 5.5 节中所提到的问题。这些解决方案实现了这里介绍的协议和算法吗？
19. 比较用于 NewReno 和 SACK 中的丢失驱动的拥塞控制，TCP Vegas 是一种 RTT 驱动的拥塞控制（其实是一个全新的思路）。但是，TCP Vegas 流行用于互联网吗？TCP Vegas 的流量会与处理网络瓶颈的丢失驱动控制的流量产生竞争吗？
20. 除了 TCP Vegas 外，还有 RTT 驱动的拥塞控制吗？或者你能够找到一个同时考虑分组丢失和 RTT 以便避免拥塞并控制速率的协议吗？它们部署在互联网上健壮和安全吗？
21. 正如 5.4.1 节中所述，当你想要为进程间或主机间打开一个套接字时，你需要分别分配 AF_UNIX 和 AF_INET 这样的域参数。在套接字层下，如何为不同域参数的套接字实现不同数据流和函数调用？对于域参数还有其他广泛使用的选项吗？在何种条件下你才需要为参数添加一个新的选项？
22. 除了 AF_UNIX 和 AF_INET 外，还有其他广泛用于域参数的选项吗？它们的功能是什么？

第6章

Computer Networks An Open Source Approach

应用层

有了底层的TCP/UDP协议栈，我们可以在互联网上提供哪些有用的、有趣的应用服务？从20世纪70年代初开始，开发出了一些互联网应用型子使得用户可以在互联网上传输信息。1971年，RFC 172发布了文件传输协议（FTP），该协议允许用户列出远程服务器上的文件并允许在本机和远程服务器之间来回传输文件。1972年，开发出了第一个电子邮件（e-mail）软件（SNDMSG和READMAIL），此协议后来在1983年标准化为RFC 821中的简单邮件传输协议（SMTP）。SMTP允许在计算机之间传输电子邮件，并且它逐渐成为最受欢迎的网络应用程序。同年，还发布了第一个远程登录规范RFC 318。远程登录允许用户登录到远程服务器机器，就像他们坐在这些计算机前一样。1979年创建了USENET，它是UNIX公司和用户的组织。USENET用户形成了运行在公告牌似系统上的数千个新闻组，用户可以在其上读取和发送信息。新闻服务器之间信息传送于1986年标准化成为RFC 977中的网络新闻传输协议（NNTP）。

20世纪80年代，一种新型互联网服务开始崭露头角。与以往只允许授权用户访问的系统不同，许多新型互联网服务几乎向所有具有适当客户机软件的人群开放。Archie是第一批允许用户在选择的匿名FTP站点数据库中搜索文件的互联网搜索引擎。Gopher服务器提供一种菜单式接口搜索互联网寻找标题或文件摘要中的关键词。Gopher协议于1993年标准化为RFC 1625。广域信息服务器（WAIS）于1994年定义在RFC 1625中，它能够控制多个Gopher搜索互联网并且将搜索结果按相关性进行排序。万维网（WWW）是由欧洲粒子物理研究所（CERN）于1989创立的。随后于1996年在RFC 1945中定义为超文本传输协议（HTTP）。它允许以集成了文本、图表、声音、视频和动画的超文本标记语言（HTML）格式访问文档。

随着新的互联网应用的不断出现，出现了一个有趣的问题：创造新应用的驱动力是什么？从上述互联网服务的演变中，很容易得出结论。那就是人—机器和人—人之间的通信已然成为推动新的互联网应用开发的驱动力。一般来讲，人—机器通信只是用来访问互联网上的数据和计算资源。例如，远程登录提供一种使用远程机器上的资源的一种方法；FTP位于数据共享；GOPHER、WAIS、WWW能够搜索并获取文档等。1987年RFC 1035文档描述的域名系统（DNS）则通过将主机IP地址映射为一种易于人们理解的主机名来解决主机地址和命名的问题。1990在RFC 1157文档规定的简单网络管理协议（SNMP），可被管理员用于执行远程网络管理和监控。另一方面，人—人之间的通信用于消息交换。举一些简单的例子，电子邮件在用户之间提供了异步交换消息的方法；1999年RFC 2453文档中描述会话启动协议（SIP）为网络电话（VoIP）是一个人—人通信的同步方式。VoIP让人们可以将互联网作为电话呼叫的传输介质来使用。同时，机器—机器通信，如对等（P2P）应用程序正在崭露头角。P2P被看做是信息和数据交换的未来，它允许用户不必通过集中式服务器的对等来交换文件。BitTorrent（BT）就是一个P2P应用的实例。尽管IETF并没有针对P2P的标准化协议，但JXTA（Juxtapose）、Sun Microsystems在2001年提出了一种P2P协议规范，旨在提高P2P应用程序之间的互操作性。

在开始设计一种新的互联网协议之前，人们首先应该解决一些一般性和特定应用的问题。大体要求从如何设计用于客户端请求和服务器响应的信息协议（这些消息是否以预定长度二进制字符串作为底层协议还是以长度可变的ASCII来表示）到客户端如何定位服务器或服务器如何才能使自己对客户机是可以访问的。客户机/服务器会话应该运行在TCP或者UDP上，服务器是否应该并发地或缓冲地服务于客户机。但是，特定应用取决于功能与应用的特点。6.1节将讨论这些一般性的问题。特定应用问题留到有关特定应用的章节中进行讨论。

首先，6.2节介绍层次化名字服务的DNS。我们介绍了DNS的关键思想，如域层次化、名字服务器和名字解析，以及经典的开放源代码软件包，伯克利互联网域名（Berkeley Internet Name Domain,

BIND)。接下来，在6.3节中讨论电子邮件。我们首先介绍消息格式和3个基于邮件协议，利用Perl作为开放源代码实现的例子进行讲解。6.4节将介绍WWW，它覆盖了网络命名和寻址、还讨论了Web数据格式、HTTP及其代理机制，并且利用著名的Apache作为开放源代码的实例。在6.5节中将学习FTP和它的文件传送服务、运行概要和开放源代码实例wu.ftpd。网络管理将在6.6节中介绍。我们将学习SNMP，包括其体系结构[RFC3411]、用于信息管理的数据结构，并以net-snmp作为其开放源代码实现。然后，分别在6.7节和6.8节中讨论两个互联网多媒体应用：VoIP和流媒体，实现的开放源代码例子为Asterisk和Darwin。最后在6.9节中讨论文件应用并用BitTorrent作为例子实现。

历史摘要：移动应用

与桌面应用相比，移动应用主要特点有高移动性的手持设备，如智能手机和移动电话。这些设备一般是配置了多种程序的计算能力和互联网连接能力并具有足够存储和无线电源的表现力强大小的计算设备。智能手机拥有较大的屏幕、多种键盘接口、图形加速芯片、加速计和基于定位的技术，移动应用程序考虑到用户保持非常有序、寻找附近的资源以及在工作场所之外工作，并保持数据与他们的在线账户或个人计算机同步。许多种移动设备平台同样也开发了很多应用程序。表6-1列出了6个移动应用程序市场（Application Marketplace）提供商。Application Marketplace是一种允许用户浏览和下载从消息到游戏、从娱乐到教育的一切流行应用程序的流行服务。

表6-1 6个移动Application Marketplace提供商

名 字	提 供 商	提供的应用	操作系统	开发环境
Android Market	Google	15 000	Android	Android SDK
App Catalog	Palm	250	webOS	Mojo SDK
App Store	Apple	100 000	iPhone OS	iPhone SDK
App World	RIM	2000	BlackBerry OS	BlackBerry SDK
Ovi Store	Nokia	2500	Symbian	Symbian SDK
Marketplace for Mobile	Microsoft	376	Windows Mobile	Windows Mobile SDK

消费类和应用的迅速发展导致处在决定之中，一种新的移动应用在走进人们的视野。让我们来看看4种流行的iPhone应用。利用Evernote，用户可以通过任何一个浏览器跨越多种平台同步保存批注文本、图片和视频文件。作为一个基于位置GPS/3G的服务，AroundMe列出一些附近的服务，如用户周围的饭店、餐馆等；Associated Press Mobile News Network是基于位置的个性化新闻服务，它将本地和专项的新闻发给用户；Wikipanion就是一种Wikipedia浏览应用程序，当用户输入关键字时它就会自动搜索匹配的维基页。

6.1 一般问题

由于多种应用需要在互联网上共存，所以客户机/服务器如何识别彼此是首要的技术问题。我们在此回顾第5章中介绍过的端口概念来解决这一问题。在互联网上提供服务之前，服务器应该首先启动守护进程（daemon）。守护进程就是指运行在后台指导服务的软件程序。因此第二个问题就是服务器如何启动。启动守护进程的方式既可以是集成的也可以是可装的，也就是说，一个守护进程既可以独立运行也可以在一个超级守护进程的控制下启动。互联网应用程序可以分为交互式的、文件传送式的，每种程序对于延迟、抖动、丢失要求都有不同的要求。有些应用程序输出的实时流最有严格的低延迟要求，但可以容忍一定量的数据丢失。另一些应用程序则会产生巨大或者长的文件传输流量，前者要求低延迟后者则可以容忍较长的延迟。很显，两者都优先选用无丢失的可靠传输。这些要求需要由服务器控制，因此互联网服务器分类也成为第二个问题。最后，尽管所有应用协议报文都是固定长度的二进制字节格式，但是相同的格式基本适用于应用层协议，因为它们的请求和应答消息是不同的并且通常包含可变长的参数。因此，这就是第四个一般性问题。

6.1.1 端口如何工作

正如5.1节所述，互联网上的每台服务器都是通过TCP或者UDP端口来提供服务的。端口用于命名承载长期或短期会话的逻辑连接的端点。根据互联网号码分配机构（IANA）分配的端口号，端口号分为以下三类：

1）众所周知端口号范围从0～1023。

2）注册端口号范围从1024～49 151。

3）动态或者私用端口号范围从49 152～65 535。

众所周知端口号仅用于系统（或者根）进程或者具有特权用户执行的程序。注册端口号只能由普通用户进程使用。动态端口可供任何人使用。

为了演示端口如何工作，我们举一个如图6-1所示的例子。服务器机器运行了4个守护进程以提供不同的服务。每个后台程序会在它自己的唯一端口上监听inbound（向内）客户机请求的到达以及客户机请求内容。在图6-1中，以FTP守护进程为例，只监听端口21，等待客户端请求的到达。当FTP发起一个outbound（向外）连接时，内核就为它分配一个大于1023的未使用端口作为其源端口。在我们的例子中为2880。连接中的FTP客户机请求它自己的IP地址和源端口以及服务器机器的IP地址和服务端口，即21。然后将连接请求发往服务器。一旦收到客户端连接请求，FTP守护进程立即自我复制，称为派生（forking）一个子进程。然后子进程就建立与FTP客户机的连接并处理随后来自该客户机的请求，而父进程则返回监听来自其他客户机的请求。

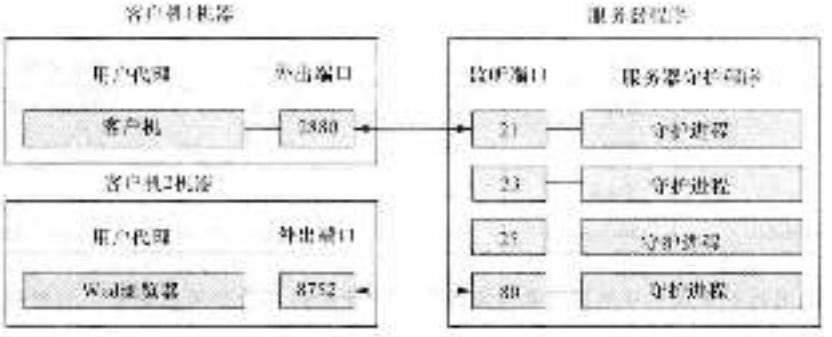


图6-1 端口工作的一个例子

6.1.2 服务器如何启动

在大多数UNIX/Linux平台上，服务器进程既可以独立运行也可以在一个称为(x)inetd的超级守护进程控制下运行。当运行时，为了应用配置文件中列出的服务，(x)inetd通过将套接字绑定到端口来监听所有的服务端口。当发现一个客户机的请求到达它的一个端口时，(x)inetd就检查并确定与端口对应的服务，调用相应的服务器程序来为客户机服务，并且继续监听端口。

利用(x)inetd来启动服务器程序有许多优点。首先，当服务器程序的配置文件更改后，由于每当客户机请求到达时(x)inetd都会重新启动程序读取配置文件，所以更改会立即生效。另一方面，独立服务器要求在配置更改生效之前显式地重新启动。其次，当一台服务器崩溃时，(x)inetd客户生一个新的服务器进程，而崩溃的独立运行的服务器却保持原样。因此其服务已不能提供。尽管(x)inetd有上述优点，它也有两个导致性能降级的缺陷。一个缺点必须为每个到达的客户机请求派生并执行一个服务器程序。另一点就是服务器程序必须为每个客户机构建它的可执行映像并读取配置文件。一般来讲，对于重负载的服务器非常使用独立方案。

6.1.3 服务器分类

互联网服务器可以从两方面进行分类。一是服务器如何处理请求，是并发还是迭代？另一种分类

方法根据底层的传输协议。面向连接的服务器使用 TCP 实现，而无连接的服务器使用 UDP 来实现。这几方面的结合就产生了 4 种类型的互联网服务器。

并发服务器与迭代服务器

大多数互联网服务都是基于客户机/服务器模型的，其目的在于激活用户进程共享网络资源。也就是说，多个客户机请求可以并发地到达服务器。服务器有两种方案对这种设计原则做出响应：并发服务器同时控制多个客户端；而迭代服务器会一个接着一个地处理客户机。

一台并发服务器进程在同一时间并发地处理多个客户机。当服务器接收到一个客户机请求时，它就创建一份自我复制，无论是子进程还是一个线程。为了简单起见，这里我们假定创建了一个子进程。每个子进程都是服务器程序的一个实例，它继承了套接字部分（在 5.4.2 节中描述过）和其他从父进程继承的变量。子进程服务客户机并解放父进程，这样它就可以接收新的客户机请求。由于父进程仅处理新到达的客户机请求，因此它不会被来自客户机的重负载所阻塞。同样，由于每台客户机由一个子进程处理并且所有的进程按计划由处理器运行，所以就可以实现子进程间（并且因此在现有的客户机之间）的并发执行。

与并发服务器相反，迭代服务器每次仅处理一个客户机请求。当多个客户机请求到达服务器时，不是派生子进程，而是服务器对客户机请求排队并按顺序处理。如果当前有太多的客户机或者有些客户机请求了很长的服务时间，迭代处理会造成阻塞，这将会延长对客户机的响应时间。因此迭代服务器仅适用于少量的和服务时间的请求。

面向连接服务器与无连接服务器

另一种划分服务器的方法是根据它们是面向连接的还是无连接的。面向连接的服务器利用 TCP 作为其底层的传输协议；而无连接的服务器采用 UDP。下面我们讨论这两者之间的不同之处。

首先，对每一个要发送的分组，面向连接的服务器都有 20 字节的 TCP 头部开销，而无连接服务器仅有 8 字节的 UDP 头部开销。其次，面向连接的服务器在传递数据之前必须与客户机之间创建一条连接，发完数据后就终止连接。而无连接服务器则无需创建连接就可传输数据。因为一台无连接服务器并不维持任何连接状态，所以它可以动态地支持更多的时间的客户机。最后，当一条或者多条客户机和服务器之间的链路拥塞时，面向连接的服务器将利用 TCP 的拥塞控制来抑制客户机，如 5.3.4 节中所述。另一方面，无连接的客户机和服务器具有不能调节的数据发送速率，它仅受到应用程序自身或接入链路的带宽控制。因此，由于持续拥塞导致的过多信息丢失就可能发生在无连接的互联网服务上，并且在需要时它将耗费客户机或者服务器更多的精力来重传丢失的数据。

表 6-2 中列出了流行的互联网应用程序和与之对应的应用程序协议和传输协议。该表中显示了电子邮件、远程终端访问、文件传送和 Web 应用程序使用 TCP 作为底层的传输协议传送应用程序级数据，因为这些应用程序的正常操作运行或决于可靠的数据传输。另一方面，像 DNS 的名字解析应用程序（如 DNS）运行在 UDP 上而不是 TCP，以避免连接建立延迟。网络管理应用程序利用 UDP 在互联网上传输网络管理消息，因为它们是基于事务的协议并且即使在一个糟糕的网络环境里也必须运行。UDP 比起 TCP 更适用于 RIP，因为 RIP 路由表在相邻路由器之间周期性更新，所以一些丢失的更新也可以被最新的更新所恢复。但是，在 BGP 中采用 TCP，使得 BGP 路由器能够与远程对等 BGP 路由器维持一种保活机制。互联网电话和音频/视频流应用程序一般都运行在 UDP 上。这些应用程序可以忍受一个小部分的分组丢失，因而对于它们的运行来说，可靠的信息传输并不是那么重要。此外，大多数组播应用运行在 UDP 上，因为 TCP 不能与组播一起工作。有关 P2P 应用的有趣点在于它们通过 UDP 来向对等发送大量的搜索查询，然后再使用 TCP 向选择的对等传输数据。

服务器的四种类型

迄今为止，已经介绍过的服务器可以分为以下四种类型：

1）迭代无连接服务器。

2）迭代面向连接服务器。

3）并发无连接服务器。

4）并发面向连接服务器。

迭代无连接服务器是最常见的服务器并且很容易实现。图 6-2 中显示了迭代无连接客户机和服务

表 6-2 应用层协议与底层协议

应　　用	应用层协议	底层协议
电子邮件	SMTP、POP3、IMAP4	TCP
远程终端访问	Telnet	TCP
文件传输	FTP	TCP
Web	HTTP	TCP
Web 缓存	ICP	典型的 UDP
名字解析	DNS	典型的 UDP
网络文件系统	NFS	典型的 UDP
网络管理	SNMP	典型的 UDP
路由协议	RIP、BGP、OSPF	UDP（RIP）、TCP（BGP）、IP（OSPF）
网络电话	SIP、RTP、RTCP 或厂商专用（如 Skype）	典型的 UDP
流媒体	RTSP 或专用（如 RealNetworks）	典型的 UDP，有时 TCP
P2P	专用（如 BitTorrent、eDonkey）	UDP 用于请求而 TCP 用于随机传输

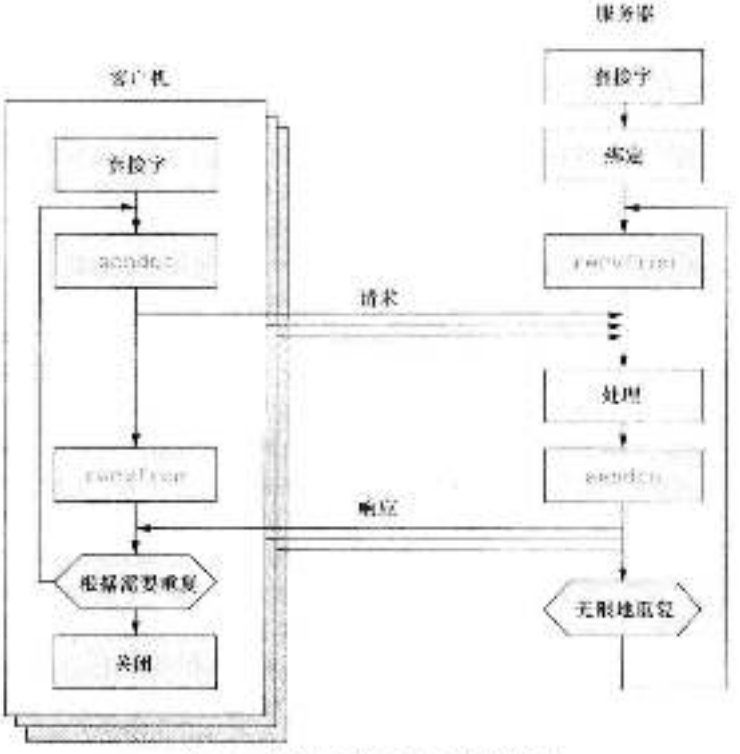


图 6-2 迭代无连接客户机和服务器

器之间的对应工作流。首先，服务器为提供的服务创建一个绑定到众所周知端口的套接字。此后，服务器就重复调用 recvfrom() 函数从套接字中读取客户机请求，这里客户机请求队列中的请求一个地接着一个地得到服务。服务器通过调用 sendto() 发送响应。由于这种体系结构的简洁性，所以迭代无连接服务器非常适用于短的和不变化的服务。

迭代面向连接服务器是一个每次处理一个客户机连接的单进程。与前面一样，服务器首先创建一个绑定到某个端口的套接字。然后服务器将套接字置于被动模式监听第一个连接（请求）的到达。一旦收到一个客户机连接（请求），它将重复地接收来自客户机的请求并做出响应。当客户机结束时，

服务器关闭连接并返回到等待接收（wait to accept）状态直到下一个连接（请求）到达。在服务时间内如果有新到达的连接请求则会排队。对于一个短时间的连接，这代面向连接模式能很好地工作，但是服务器如果运行在迭代无连接服务器模式就会有更小的开销。对于一个长时间的连接，这种模式会造成较差的并发性和延迟。因此，很少使用。

并发无连接服务器适用于拥有高请求量但仍需要快速周转时间的服务。DNS 和网络文件系统（NFS）就是两个例子。并发无连接服务器首先创建一个绑定到端口的套接字，但是却让套接字是无连接的，即不能与任何客户机通信。然后服务器开始从客户机接收请求并通过派生子进程（或线程）处理它们，一旦完成就立即退出。

并发面向连接服务器的处理流如图 6-2 所示，它广受欢迎。基本上，它们与并发无连接服务器的工作原理相似，但在连接的建立上却有差异，它包含一种额外的 TCP 三次握手。服务器程序使用两种套接字：父进程使用的监听套接字和子进程使用的已连接或正在接收套接字。服务器创建一个与端口绑定的套接字后将它置于监听模式，服务器进程就会阻塞 accept() 函数直到一个客户机连接请求到达为止。为已连接套接字一旦从 accept() 返回一个新的文件描述符，它将派生一个继承了两个套接字的子进程。子进程关闭监听套接字并通过已连接套接字与客户机通信。父进程随后关闭已连接套接字，返回到 accept() 的阻塞状态。

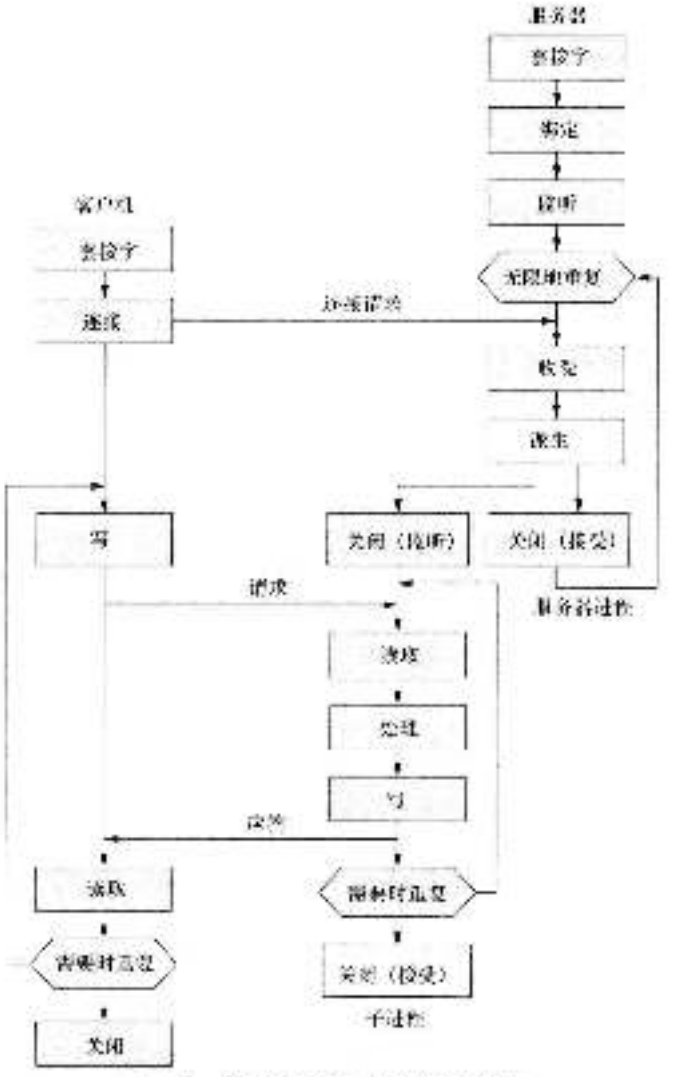


图 6-2 并发面向连接客户机和服务器

历史演变：云计算

传统上，一个组织倾向于拥有自己的网络基础设施、服务器平台、应用软件以及运行它自己的计算环境。另外一种过程，称为云计算，是将计算环境外包给其公共基础设施既可以是分布式，也可以是集中式统一管理的集中式服务提供商。国家标准和技术协会（NIST）将云计算定义为一种用于激活按需访问可配置计算资源（可以快速供给并以最小管理努力释放）共享池的模型。从具有简单瘦客户机胖服务器概念的网络计算演变而来，云计算进一步将胖服务器外包给服务器提供商。使用下面三种服务供给模型——软件即服务（SaaS）、平台即服务（PaaS）、基础设施即服务（IaaS）。

Google Apps（http://google.com/a/）和 Apps.Gov（http://apps.gov）是两家早期的云服务供应商。这些仅会有极少数像谷歌、亚马逊公司所运营的公共 B2C（商家对客户）云，但是会有很多由商业公司（如 IBM）所运营的公共 B2B（商家对商家）云，甚至有更多由厂商和制造商提供的更专用的云。这里“公共”或者“专用”意味着云是否向公众提供服务。

6.1.4 应用层协议的特点

互联网服务器和客户机运行在终端主机上并且通过应用层协议进行通信。应用层协议指定从客户机到服务器消息交换的句法和语义。句法定义了消息的格式，而语义指定客户机和服务器应该如何解释消息并向对等做出响应。与底层的传输协议和互联网协议相比，应用层协议具有很多不同的特点。

可变的消息格式与长度

与信息总有固定格式或者长度的第二层－第四层协议不同，应用层协议无论是请求还是响应都可以是可变的格式和长度。这是取决于各种选项、参数或者在命令或应答中传输内容的大小。例如，当发送一个 HTTP 请求时，客户机可以将一些字段添加到请求中以便指示客户机使用的浏览器和语言。同样，一个 HTTP 响应会根据不同种类的内容更改其格式和长度。

多种数据类型

应用层协议有多种数据类型，这就意味着命令和应答既能以文本且能以非文本形式传输。比如，telnet（远程登录）客户机和服务器发送以一个特殊字节（11111111）开头的二进制格式的发送命令，而 SMTP 客户机和服务器使用美国 7 位 ASCII 码进行通信。FTP 服务器以 ASCII 或者二进制格式来传递数据。网络服务器以文本 Web 网页和二进制图像进行回复。

有状态的

许多用户层协议都是有状态的。也就是说，服务器保留有关与客户机会话的信息。比如，FTP 服务器会记住客户机的当前工作目录和当前传输类型（ASCII 或二进制）。SMTP 服务器在等待 DATA 命令传输消息内容时会记住电子邮件发送者和接收者的信息。然而，为了效率和可扩展性，将 HTTP 设计为一个无状态协议，尽管在原始协议基础上添加的某些状态会被客户机和服务器保留。另一个无状态的例子就是 SMTP，它也要处理效率和可扩展性问题。DNS 既可以是无状态的也可以是有状态的，具体取决于它如何运行，这一点我们将在下面看到。

6.2 域名系统

域名系统（DNS）是一种层次化的、分布式数据库系统，用于为各种互联网应用程序映射主机名和域名。它是用来提供实用的、可扩展的名字到地址（有时是地址到名字）之间的翻译服务。在本节中，我们将讲到 DNS 的三个方面：1）DNS 工作的名字空间的结构；2）定义名字到地址映射的资源记录（RR）结构；3）名字服务器和解析器之间的操作模型。最后，我们将介绍开放源代码的实现 BIND，以期为读者提供一个 DNS 的实际视图。

6.2.1 简介

连接到网络上的计算机主机需要一种彼此识别的方法。除了二进制的 IP 地址外，每台主机可能会注册了一个 ASCII 字符串作为其主机名。就像在邮政系统中的邮编一样，它包含一系列的字符来说明国家、城市、街道信息。ASCII 字符串唯一地标识主机的位置并且它比 IP 地址更便于记忆。

回顾 ARPANET 时代，文件 HOST.TXT 被一个称为网络信息中心（NIC）的权威组织使用，以便存储 ASCII 名字和区域内与所有主机对应的 IP 地址。ARPANET 的主机管理员周期性地从 NIC 获取最新的 HOST.TXT，并将它们的更改发送给 NIC。无论什么时候一台主机要和另一台主机建立连接，它都必须使用 HOST.TXT 将目的地主机名翻译为对应的 IP 地址。这种方法仅适用于小的网络。然而，当有一个大型网络一起时，就会出现可扩展性和管理问题。

RFC 882 和 RFC 883，后来由于 RFC 1034 和 RFC 1035 的出现而被废除，提出了可扩展的互联网主机寻址和邮件转发支持以及用于实现域命名设施的协议和服务器的层次化分布式 DNS 的概念和规范。DNS 中的主机有一个唯一的域名。域的每一台 DNS 名字服务器维护它自己的名字地址映射数据库，这样互联网上的其他系统（客户端）可以通过 DNS 协议消息查询名字服务器。但是我们如何将域名空间划分成域呢？6.2.2 节将告诉我们答案。

6.2.2 域名空间

顶级域

为了在一个大型网络中管理和扩展域名翻译，将域名空间分为多个顶级域，如表 6-3 所示。每个域代表特别的服务或者含义，并且进一步向下分成各个子域，以此类推分成子子域，形成一个层次化域树。在某个域树中的一台主机继承了反向到根路径上相继域的目的。例如，主机 www.w3.org 的目的是服务于 WWW 社区的官方站点，这就像顶级域 org 一样是一个大型利的组织。

表 6-3 顶级域

域	概 述
com	商业组织例如英特尔 Intel（Intel.com）
org	非盈利组织例如万维网联盟（w3.org）
gov	美国政府机构如国家科学基金会（nsf.gov）
edu	教育机构例如 UCLA（ucla.edu）
net	网络互联网机构如联网地址分配机构，主它维护 DNS 根服务器（gtx.servers.net）
int	由政府间国际条约建立的机构。例如，国际电信联盟（itu.int）
mil	专为美国军事机构创建。例如，国防部网络信息中心（nic.mil）
双字母国家代码	双字母国家代码顶级域（ccTLDs）是根据 ISO 3166-1 双字母国家或地区代码而来。例如 cn（China，中国）和 uk（United Kingdom，英国）
arpa	大多数人被使用。in-addr.arpa 域例外，它用于维护用于逆向 DNS 查询的数据库
最新	如 biz（business，用于商业），biz（用于个人）和 info（与 .com 类似）

利用域名空间的层次化划分，在域树中的主机（或域）地址可以很容易地标识出来，因此允许我们推测出 DNS 的结构。在图 6-4 中，为我们提供了一个 cs.nctu.edu.tw 域是如何被识别的例子。注意：域名不分大小写，这意味着大写“CS”等价于小写“cs”。

除了已经标准化的顶级域外，一个域全权管理它的连续域。域管理员可以将它任意划分成子域并且把它们分配给其他组织，既可以按照顶级域的形式（例如，tw 下的 com）也可以创建一个新的域名（例如，uk 下的 co）。这个过程称为域委派，它会极大地减少上级域的管理负担。

区和名字服务器

在继续本节内容学习之前最好清楚地区分域（domain）和区（zone）是非常重要的。一个域通常是区的一个超级集合。以图 6-4 为例来说明，tw 域包含 4 个区：org、com、edu 和 tw 本身（你可以尝试一下 http://www.tw）。为了特别说明，tw 区包括 *.tw 形式的域名，除了 *.org.tw、*.com.tw 和 *.edu.tw。.tw 区的主机通常用于管理代理的子域。

通常在端口 53 上运行的名字服务器是包含用区数据文件构建的数据库和一个解析程序以便回答 DNS 查询的主机。在名字服务器中，区是基本的管理单元，它的信息存储在一个区数据文件中。一旦

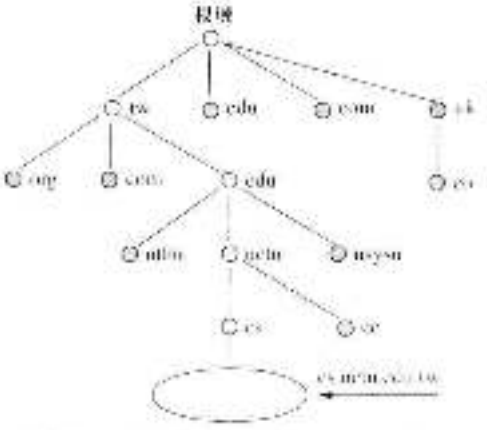


图6-4 在名字空间中定位 cs.nctu.edu.tw

接收到DNS查询，即给出域名并请求对应的IP地址，名字服务器就查询它的数据库寻找答案。如果找到相匹配的答案就将其回复给客户端；否则，就在其他命名服务器上做进一步查找。一台名字服务器可以负责多个区，这就意味着数据库包括多个区并且有多个区数据文件。如图6-5所示。在图6-5中，方形和椭圆分别代表名字服务器和区，位于中间的名字服务器包括区A和区B。

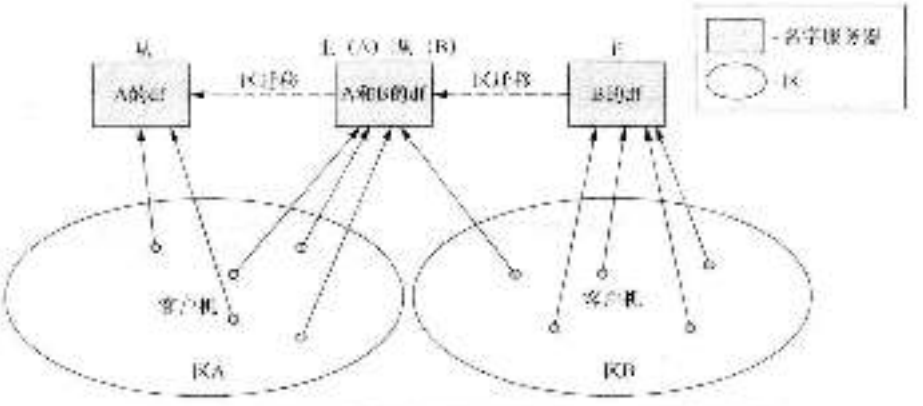


图6-5 主、从名字服务器之间的关系（cf：zone区数据文件）

为了可用性和负载平衡性，一个区也许会被多台名字服务器监控，以避免可能在单台服务器上的高请求率引起的中断。为同一个区服务的服务器可以划分为一台主机和多台从机。一台主名字服务器具有在一个区中删除或者添加主机的权力。而从名字服务器只能通过一种处理（即区传输）从它们的主名字服务器获取区信息。在图6-5中详细说明了区A的主名字服务器是区B的从名字服务器。

6.2.3 资源记录

一个区数据文件由描述区的DNS设置的多个资源记录（RR）组成。一个RR一般包含一个五元组：所有者（索引RR的域名），TTL（生存时间，解析器可以缓存该条RR的一段有限时间），类（通常采用IN，代表互联网系统），RDATA（一个可变长度的八字节字符串可以描述资源）。通常有六种类型的RR用于描述域名的不同方面：

起始授权机构（SOA）：SOA标志着一个DNS区数据文件的开始，并确定了该区的授权机构。也就是说，当我们想知道一个区的授权机构名字服务器时，我们发出对SOA的查询。一个例子如下：

进一步解释“cs.nctu.edu.tw域有一个授权机构名字服务器 cserv.cs.nctu.edu.tw”将更具可读性。这个翻译适用于其余的RR。域名 help.cs.nctu.edu.tw 通常授权机构服务器指定负责该区的管理员的地址，如RFC 1035中定义。然而，help.cs.nctu.edu.tw 将由DNS应用程序自动地转译成 help@

cs.nctu.edu.tw。从名字服务器使用序号来触发对区数据文件的更新。仅当从名字服务器的副本具有比主名字服务器的副本更小值序号时才会发生。为此，序号通常设置为修改日期。

```
cs.nctu.edu.tw. 86400 IN SOA csserv.cs.nctu.edu.tw.
  help.cs.nctu.edu.tw.

 (
    2009112101 ; Serial number
    86400      ; Refresh after 1 day (86400 seconds)
    3600       ; If no response from master, retry after
                 1 hour
    1728000    ; If still no update, the data expires
                 after 20 days
    86400      ; TTL of RRs in caches in other name servers
 )
```

地址（A）：这是最重要和最常用的 RR，用于将域名与 IP 地址进行匹配，以转发查询请求。由于多穴主机有多个网络接口卡，所以就有多个 IP 地址，针对同一个域名允许有多个 ARR。多穴主机的一个例子如下：

```
linux.cs.nctu.edu.tw    86400  IN  A  140.113.168.27
                        86400  IN  A  140.113.209.27
```

这意味着对 linux.cs.nctu.edu.tw 的查询将返回两个 IP 地址。

规范名（CNAME）：CNAME 使用一个指向 IP 地址规范域名的域名创建一个别名。这对于从一个 IP 地址运行多个服务特别有用。在下面的例子中，cache.cs.nctu.edu.tw 最初在 IP 地址 140.113.166.122 上用于 Web 缓存服务并且因此它是这个 IP 地址的规范名。然而，同时，它也充当 Web 服务器，因此就为 www.cs.nctu.edu.tw 创建了一个别名：

```
www.cs.nctu.edu.tw.    86400 IN CNAME cache.cs.nctu.edu.tw.
cache.cs.nctu.edu.tw. 86400 IN A      140.113.166.122
```

这样，当查询别名时，名字服务器首先查找相应的规范名，然后查找规范名的 IP 地址，最后返回这两个结果。

指针（PTR）：与 A RR 相反，PTR RR 从推定的 IP 地址指向域名。所谓的逆向查询，查询 IP 地址的域名，就采用此方案。例如，RR 提供

```
10.23.113.140.in-addr.arpa. 86400 IN PTR laser0.cs.nctu.edu.tw.
```

从 IP 地址 140.113.23.10 到域名 laser0.cs.nctu.edu.tw 的逆向映射。这里将 IP 地址表示为用于逆向 DNS 查找在 in-addr.arpa 域的一个子域。注意，PTR 将 IP 地址存储为逆序字节顺序，因为域名由左到右安排不那么明确了。换句话说，IP 地址 140.113.23.10 以域名 10.23.113.140.in-addr.arpa 存储指向它的规范名。

名字服务器（NS）：一个 NS RR 标志着一个 DNS 区数据文件的开始，并提供该区名字服务器的域名。它通常出现在 SOA RR 的后面以便为 6.2.4 节中描述的请求转发提供额外的名字服务器。例如，在 NCTU 的名字服务器 ndns.nctu.edu.tw 中的 CS 名字服务器的一个 NS 表项可以是：

```
cs.nctu.edu.tw. 86400 IN NS csserv.cs.nctu.edu.tw.
```

这使域名服务器得知在 cs.nctu.edu.cn 域中的主机查询转发给授权主机 csserv.cs.nctu.edu.tw。注意，在 SOA RR 中定义的名字服务器总有一个 NS RR，并且对应的一个 A 类型 RR 必须在区数据文件中针对一个 in-zone 名字服务器附带有 NS RR。

邮件交换（MX）：MX RR 为域名公布邮件服务器的名字。这是用来实现邮件转发的。例如，一个邮件发送者试图将电子邮件发至 help@cs.nctu.edu.tw，可能会请求名字服务器有关 cs.nctu.edu.tw 的邮寄信息。利用以下 MX RR，

```
cs.nctu.edu.tw  86400  IN  MX  0 mail.cs.nctu.edu.tw.
cs.nctu.edu.tw  86400  IN  MX  10 mail2.cs.nctu.edu.tw.
```

发件者知道将电子邮件转发给邮件交换器 mail.cs.nctu.edu.tw。当存在多个邮件交换器可就要从中选择一个。在邮件交换器字段之前的数字代表其优先值。在这个例子中，邮件发送者将选择第一个，它有一个更好（更低）的邮件转发的优先值。第二个一般不会被选中除非第一个停机。

6.2.4 名字解析

DNS的另一个重要组成部分是解析器程序。它通常由程序库组成，应用程序（如Web浏览器）使用它将URL的ASCII字符串转换为有效的IP地址。解析器给驻留的UDP支持下TCP端口53的域名服务器生成DNS查询，并解释来自服务器的应答。域名解析包括查询解析器和查询过的本地名字服务器，有时还包括其他区的名字服务器。

多次迭代查询

在最好的情况下，如果服务器在其数据库中找到答案则本地名字服务器将马上回答来自解析器的查询。否则，服务器将进行多次迭代查询，而不是简单地将未回答的查询退还给解析器。如图6.6所示，当本地名字服务器收到一个对“www.dli.gov.uk”的查询后，它就开始多次迭代查询。到过的域中的域名服务器而不仅仅是上一级域名服务器，因为它们可能不知道答案或者向谁转发。如果没有找到答案，根服务器将转发给域名层次结构中离目的主机最近的名字服务器。也可能有多个适于转发的候选者，但以轮转方式选择其中一个。然后本地服务器重复查询转发的名字服务器，如“uk”名字服务器，它可能再次应答以再一次转发，如“gov.uk”名字服务器等，直到查询到达目的地主机所在域的名字服务器。在我们的例子中，查询终止于“dli.gov.uk”名字服务器。该名字服务器将主机IP地址提供给本地名字服务器，然后它将答案转发给解析器，完成名字解析过程。

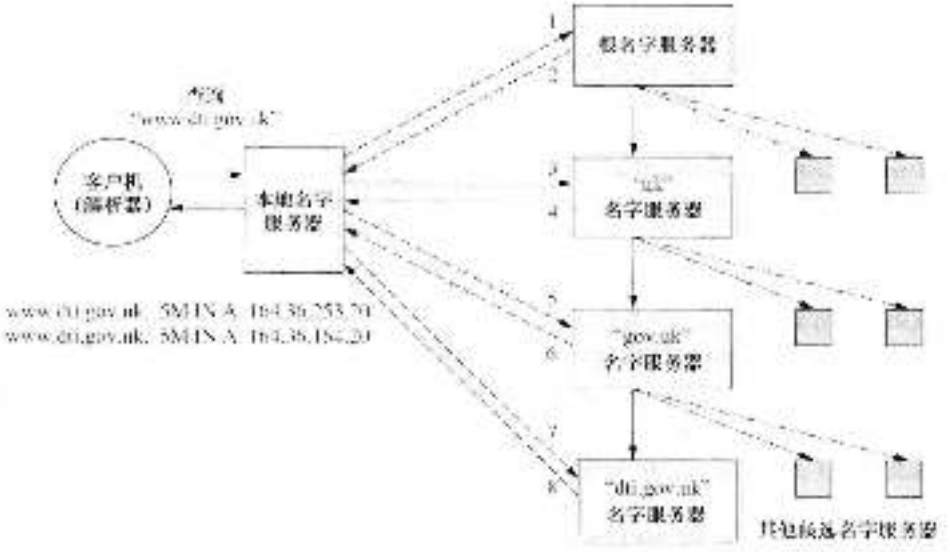


图6.6 对www.dli.gov.uk的多次迭代解析

值得注意的是，这里的解析器给出一种递归查询，而能递归的本地名字服务器保持状态以便通过多次迭代查询解析远程查询。如果本地名字服务器不能递归，解析器就需要发送迭代查询到其他已知的名字服务器。幸运的是，大多数本地名字服务器都能够递归。

让我们以逆向DNS查找作为另外一个例子来说明。逆向查询对应于IP地址164.36.164.20的域名是通过多次迭代查询来解析的一系列步骤。我们查找RR：

```
20.164.36.164.in-addr.arpa. 86400 IN PTR www.dli.gov.uk.
```

当本地名字服务器不能在它的数据库中找到对应的RR时，它就请求.arpa域的权威根名字服务器。根名字服务器，虽然它也可能没有所需的RR，但它可能会提供一些RR的转发信息，例如：

```
164.in-addr.arpa.      86400  IN  NS   ARROWROOT.ARIN.NET.
ARROWROOT.ARIN.NET.    86400  IN  A    198.133.199.110
```

其中声明164.*.*.*域名在IP地址为198.133.199.110的名字服务器ARROWROOT.ARIN.NET管理下。它可以根据本地RR再次将查询转发给主好的名字服务器：

```
36.164.in-addr.arpa.    86400   IN   NS   NS2.JA.NET.
NS2.JA.NET.             86400   IN   A    193.63.105.17
```

最后，所需的 RR 在名字服务器 NS2.JA.NET 上找到。

正如我们从这两个例子中可以看到，除了本地名字服务器外，所有名字服务器，当它们没有所需要的 RR 时，它们只提供转发，而不是以本地名字服务器的名义发出查询。后一种方法对于非本地名字服务器保持处理中所有查询的状态是不可扩展的。因此，只有能够选择的本地名字服务器是有状态的，而其他非本地名字服务器都运行在无状态模式下。

历史演变：遍布世界的根 DNS 服务器

在层次化域名空间中，根 DNS 服务器回答对 DNS 根区的查询。根区是顶级域名（TLD），是互联网的最高等级域名，包括通用顶级域（gTLD）（如 .com）和国家代码顶级域（ccTLD）如 .cn。到了 2010 年年初，在根区中有 20 个通用顶级域名（TLD）和 248 个 ccTLD。根 DNS 服务器对于互联网非常重要，因为它们处在将域名变换成为在全球 IP 地址的一环。

共有 13 个根域名服务器（参见 http://www.root-servers.org/）实现根 DNS 区。13 个根域名服务器中的每一个用从 A 到 M 的字母标识符标识。虽然仅使用了 13 个字母的标识符，但每个标识符的运营商可以使用完全的物理服务器机器以使满足高性能 DNS 查找和高可靠性。表 6-4 显示了这 DNS 服务器，包括二列中显示代理服务器的 IP 地址。在名字解析过程中，本地名称域名服务器经常的查询首先被发到 13 个根之配置的根服务域名服务器，以配成这种的方式。根理服务器然后将本地域名服务器重定向到其完全服务器之一，然后对下一级权威域名服务器的地址应答。在表 6-4 中，全球和本地网站之间的区别在于负载均衡考虑的是全球的还是局部的。

表 6-4 根 DNS 服务器

字母	IP 地址	运 营 商	地 理 位 置	站 点
A	IPv4: 198.41.0.4 IPv6: 2001:503:BA3E::2:30	VeriSign（威瑞公）公司	美国，加利福尼亚州，洛杉矶； 美国，弗吉尼亚州，阿什本； 美国，加利福尼亚州，帕洛阿尔托； 美国，弗吉尼亚州，维也纳	全球：4
B	IPv4: 192.228.79.201 IPv6: 2001:478:65::53	ISI（USC）南加州大学信息科学研究中心	美国，加利福尼亚州，马里纳德尔雷	全球：1
C	IPv4: 192.33.4.12	Cogent Communications 科进通信公司	美国，弗吉尼亚州，赫恩登； 美国，加利福尼亚州，洛杉矶； 美国，伊利诺伊州，芝加哥； 美国，弗吉尼亚； 由卡文，马德兰	全球：8
D	IPv4: 128.8.10.90	马里兰大学	美国，马里兰州，帕克分区	全球：1
E	IPv4: 192.203.230.10	NASA 艾姆斯研究中心	美国，加利福尼亚州，莫菲特菲	全球：1
F	IPv4: 192.5.5.241 IPv6: 2001:500:2f::f	互联网系统协会，公司	全球：美国加利福尼亚州帕洛阿尔托； 美国加利福尼亚州旧金山； 本地：47 处一备用	全球：5 本地：47
G	IPv4: 192.112.36.4	美国国防部网络信息中心	美国俄亥俄州哥伦布； 美国得克萨斯州圣安东尼奥； 美国夏威夷州檀香山； 日本，绳生； 德国斯图加特－巴登符； 西太平，菲律宾	全球：6

（续）

字母	IP地址	运营者	地理位置	站点
H	IPv4：128.63.2.53 IPv6：2001:500:1::803f:235	美国陆军研究实验室	美国马里兰州阿伯丁试验场	全球：1
I	IPv4：192.36.148.17	瑞典斯德哥尔摩	31 世界范围	本地：34
J	IPv4：192.58.128.30 IPv6：2001:503:c27::2:30	VeriSign（威瑞信）公司	全球：55 世界范围； 本地：美国华盛顿特区； 美国加利福尼亚州； 美国，加利福尼亚州，帕洛阿尔托； 美国，加利福尼亚州，洛杉矶； 日内，北京； 台北市，大都会； 埃及，开罗	全球：55 本地：9
K	IPv4：193.0.14.129 IPv6：2001:7fd::1	RIPE NCC 欧洲 IP 网络资源协调中心	全球：美国，伦敦； 荷兰，阿姆斯特丹； 德国，法兰克福；日本，东京； 美国，佛罗里达州，迈阿密； 印度，孟买； 本地：12 加各范围	全球：6 本地：12
L	IPv4：199.7.83.42 IPv6：2001:500:3::42	ICANN 互联网名称与数字地址分配机构	美国，加利福尼亚州，洛杉矶； 美国，佛罗里达州，迈阿密； 法国共和国，布拉格	全球：3
M	IPv4：202.12.27.33 IPv6：2001:dc3::35	WIDE Project（分布式计算扩充环境项目）	全球：日本，东京（3个站点）； 法国，巴黎； 美国，加利福尼亚州，旧金山； 本地：韩国，首尔	全球：3 本地：1

协议消息格式

在 DNS 协议中使用的消息包含以下 5 个部分，如图 6-2 所示。



图 6-2 DNS 消息的内部构造

头部部分：这包括有关查询的控制信息。该 ID 是标识消息的唯一数字，并用来匹配未确认查询的应答。第二行包含一些标志，表明类型（在 QR 中指定的查询或响应）、类型（在 OPCODE 中指定的正向或反向查询）、所需要的（RD）或可以提供的（RA）递归查询、消息的错误代码（RCODE）。其他字段给出了在头部之后部分的表项编号。

问题部分：这是用来提问查询中的问题。头部部分的 QDCOUNT 指示在本部分中的表项数（通常为 1）。

答案、权威和附加部分：它们中的每一个都包含很多 RR，所有者信息以 ASCII 格式存储，具有可变的长度，在头部部分中的 ANCOUNT，NSCOUNT 和 ARCOUNT 中指定 RR 的计数。前两个部分告诉问题的答案和指定的权威域名服务器，而附加部分提供与查询有关的有用信息，但并不是确切的答案。例如，如果在权威部分有一个如“ncku.edu.tw. 259200 IN NS ns.ncku.edu.tw.”的表项，那么很可能在附加部分有一个如“ns.ncku.edu.tw. 259200 IN A 140.113.250.135”的表项以便提供权威域名服务器的主机 IP。

开源实现 6.1：BIND

概述

互联网域名（BIND）由互联网软件协会（ISC）维护，是为 BSD 派生操作系统实现的一种域名服务器系统。BIND 由一个多线程（因操作系统而定）后台守护进程 named 和一个解析器库构成。解析器是系统库中提供接口的一套例程，通过它应用程序就能够访问域名服务。也包括一些高级的功能和安全组件也包含在 BIND 中。BIND 是目前为互联网提供 DNS 服务的最广泛软件。它可以运行在大多数类 UNIX 操作系统上，包括 FreeBSD 和 Linux。

我们可以将 BIND 视为为 DNS 的并发多线程实现。BIND 在端口 53 上既支持面向连接服务也支持无连接服务。但后者经常用于快速响应。对解析器的默认查询解析是递归的，但由多个迭代查询进行。请求进程，除了本地 DNS 服务器外，DNS 服务器保持无状态。

框图

典型情况下，named 守护进程作为根进行。出于安全考虑，通过 chroot() 系统调用（称为“最少特权”机制）named 也可以不作为根运行。通常情况下，它监听对 TCP 或 UDP 端口 53 的请求。按照惯例，由于性能考虑我们选择 UDP 传输普通消息，但是必须使用 TCP 进行区域转以避免可能丢失区域数据文件。

有了多线程的支持，此能够创建三个主管理器线程：任务管理器、定时器管理器和套接字管理器，如图 6-8 所示。在图 6-8 中，每个任务都关联一个来自定时器管理器的定时器。在所有任务中，有 4 个任务能够运行，套接字管理器的任务 1（Task1）发送一个 I/O 完成事件。

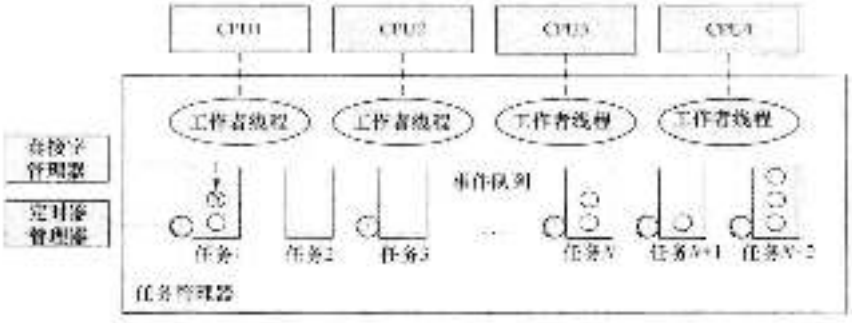


图 6-8 BIND 中的任务管理器、定时器管理器和套接字管理器之间的关系

由于 BIND 9 支持多处理器平台，所以每个 CPU 关联一个由任务管理器创建的工作者线程以便处理各种任务。一个任务有一系列在队列中排序的事件（例如，解析请求、定时器申请）。当任务的事件队列为非空时，这个任务就是可运行的。当任务管理器给工作者线程分配一个可运行的任务时，工作者线程通过处理任务事件队列中的事件来执行任务。

任务和定时器是基于各种目的的，如客户端请求超时、高速缓存失效。定时器管理器

用于建立和调整定时器。它们本身就是事件的来源。套接字管理器提供 TCP 和 UDP 套接字，它们也是事件的来源。当网络 I/O 完成时，就将套接字的一个完成事件发送给请求 I/O 服务的任务事件队列。

创建许多其他的子管理器以支持刚刚描述过的管理器——named，用于传输的层管理器，用于处理传入的请求并产生对应答复的客户端管理器。

数据结构

BIND 的数据库中存储着关于 view 数据结构的三信息，它具有一组区。它将用户分为具有访问 DNS 服务器不同权限的用户组。换句话说，一个用户只允许访问授权给他查看的 view。

利用 view 来划分用户的一个实际例子就是所谓的分离 DNS。企业或服务提供商通常包含两种主机：客户主机和服务器。由于服务器的 DNS 信息需要向外界发布，所以在一定程度上，企业必须允许外部用户查询访问其本地名服务器。然而，这样一来企业的网络拓扑可能因此暴露给外面世界，如果外部用户可以查询点其域中的其他主机。这可以通过分离 DNS 来解决，其中有两种类型，即外部和内部 DNS 服务器。前者为外部查询提供服务器信息，而后者服务于内部主机。虽然这个方案很好地解决了私人信息暴露的潜在风险，但使用额外的 DNS 服务器会造成额外的财政支出。幸运的是，在 view 结构的帮助下，通过将用户分成内部组和外部组，只需要一台服务器就能够支持这种分离 DNS。

图 6-9 展示了 named 使用的数据结构。如果在配置文件中没有显式的 view 声明语句，那么匹配任何用户的默认 view 就由所有区数据文件生成。当创建了两个以上的 view 时，它们就合并成一张链表。根据其源地址和 view 的访问控制列表，服务器利用 view 匹配进入的查询。之后，选择第一个匹配的。

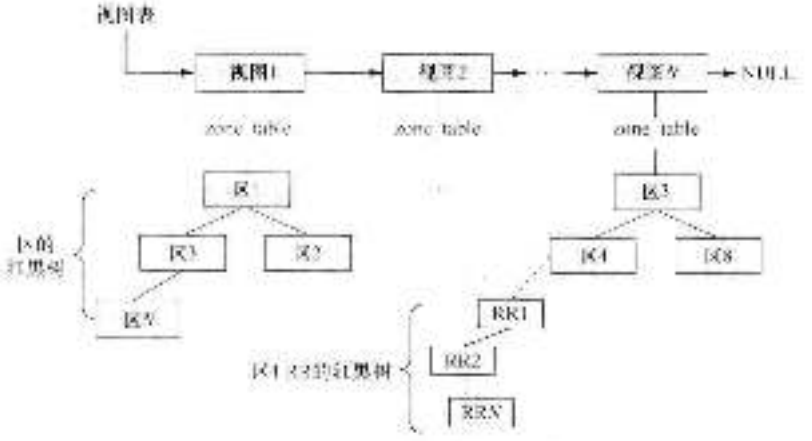


图 6-9 named 内部的数据结构

算法实现

在 view 中是组织成红黑树（RBT）的权威区。RBT 是一种更复杂的平衡树，但它也提供了一个相当良好的搜索时间 log（N），其中 N 表示树中的节点数量。在区数据文件中的 RR 也作为 RBT 实现以便利用区域内的代码和设备。在这个阶段，对请求的 RR 查询在区开在区立的 RBT 中经过进行匹配过程直到找到需要的 RR 为止。如果没有找到匹配 RR，服务器求助于外部域名服务器为客户端执行查询过程。

dig——一个小解析器程序

除了 BIND 套件中的 named 外，还有一个强大的解析工具，称为域信息搜索器（dig）。如图 6-10 所示，它执行 DNS 查找，与另一种流行的工具 ns-lookup 相比，它能够显示被查询域名服务器的大量信息。除了简单的查询外，用户甚至可以用带有“+trace”选项的 dig，沿域名路径跟踪被查询的域名服务器。

```
; <<>> DiG 9.2.0 <<>> www.nctu.edu.tw
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26027
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.nctu.edu.tw.        IN      A

;; ANSWER SECTION:
www.nctu.edu.tw.        259200  IN      A       140.113.250.9

;; AUTHORITY SECTION:
nctu.edu.tw.            259200  IN      NS      ns.nctu.edu.tw.
nctu.edu.tw.            259200  IN      NS      ns2.nctu.edu.tw.
nctu.edu.tw.            259200  IN      NS      ns3.nctu.com.tw.

;; ADDITIONAL SECTION:
ns.nctu.edu.tw.         259200  IN      A       140.113.250.132
ns2.nctu.edu.tw.        259200  IN      A       140.113.6.2
ns3.nctu.com.tw.        259200  IN      A       163.28.64.11
```

图 6-10 利用 dig 对 www.nctu.edu.tw 主机进行查询的例子

练习

1. 在按实现迭代解析 dns_server.c 文件时代码行。
2. 在一台主机上分别以非递归查询和递归查询查找 IP。
3. 在一台本地域名服务器上使用 dig 检索所有的 RR。

6.3 电子邮件

电子邮件、FTP 和 telnet 是 20 世纪 70 年代以来最早的 3 种互联网应用。然而，电子邮件（e-mail）比其他两个更普及。虽然手机消息正在迅速上升，但电子邮件仍然是日常生活中必不可少的互联网应用。本节首先介绍电子邮件发送系统的组件和处理流程。然后，我们描述基本的和高级的电子邮件报文格式。互联网报文格式支持多用途互联网邮件扩展（MIME）。接下来我们讨论用于发送和读取电子邮件的协议，简单邮件传输协议（SMTP）；从电子邮箱中检索电子邮件的协议，邮局协议（POP）和互联网消息访问协议（IMAP）。最后，我们将 qpopper 作为电子邮件开源实现的一个例子。

6.3.1 简介

如今的电子邮件服务可以追溯到早期 ARPANET 阶段当时在 1973 年（RFC 561）提出的电子邮件的编码标准。在 20 世纪 70 年代初，电子邮件的发送与如今可联网上的电子邮件发送非常相似。在 20 世纪 80 年代初代进一步简化奠定了当前电子邮件业务的基础。

电子邮件是一种通过计算机网络传递邮件从一个用户发送到另一个或多个用户的方法。实际上，一封简单的邮件大体写，收入到信箱中，暂存在邮箱，然后投递到收件人的邮箱中，当后收件人从其邮箱检索到。发送、接收和检索电子邮件的方式类似于传统邮件的传递过程。发信人利用计算机编写电子邮件，并按照规定发送到邮件服务器。在此之后，邮件服务器将邮件传递到在目的邮件服务器上的收件人邮箱中。最后收件人从他的邮箱中利用账户和密码信息检索消息。这样，电子邮件可以在几秒内传递给任何收件人，而不是像通过普通邮件那样需要数天之久。

互联网邮件寻址

就像信封上的姓名和地址一样，为了传输上的需要一种机制来表示电子邮件的发件人和收件人信息。通过每个人的电子邮件地址，可以到达每个电子邮件用户，格式定义如下

```
user@host.network
```

电子邮件地址由 3 部分组成。第一部分和第二部分分别标识收件人的用户名和邮件服务器。在第一部分和第二部分之间总有一个@符号分隔，意味着“该用户所在的邮件服务器”。第二部分告诉网

络成员，邮件服务器所在的位置。请注意，为了简单，第二部分经常被省略，因为接收的邮件服务器可以从 DNS 的邮件交换器（MX）资源记录中查询，如 6.2 节中所述。

互联网邮件系统的组件

接下来的问题是哪些组件构成了电子邮件系统。电子邮件系统由 4 个关键的逻辑元素组成：邮件用户代理（MUA）、邮件传输代理（MTA）、邮件投递代理（MDA）和邮件检索代理（MRA），如图 6-11 所示。下面首要介绍它们。

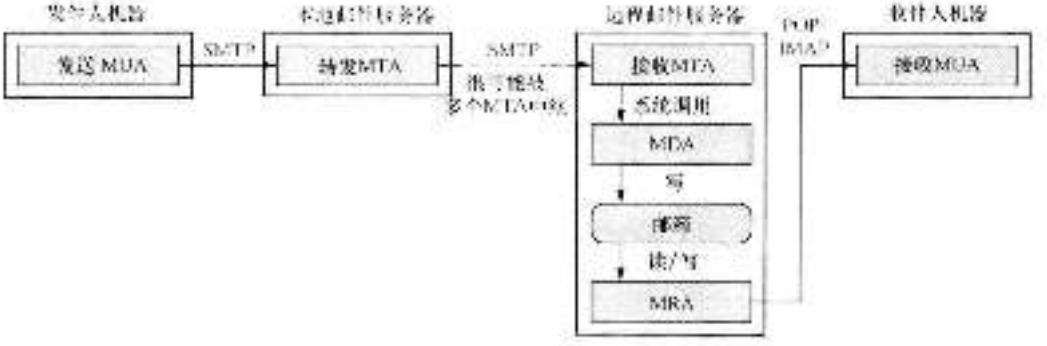


图 6-11 电子邮件系统的逻辑元素

邮件用户代理

邮件用户代理（MUA）是电子邮件客户端程序，通过它用户就可以发送和接收消息。MUA 经常采用一种编辑器软件为用户显示和编辑消息。除了阅读和编写消息外，MUA 还使用户能够添加电子邮件附件。常用的 MUA 应用程序包括在 elm、mutt、基于 UNIX 系统的 Pine，诸如 Windows 系列的 Outlook Express 和 Thunderbird。为用户发送和接收消息的自动化脚本或程序也可以认为是 MUA。

邮件传输代理和邮件投递代理

像 sendmail、qmail 和用于 UNIX 平台的 postfix 这样的 MTA 用于通过简单邮件传输协议（SMTP）从 MUA 接收消息并将它们直接传递给远程邮件服务器上的 MTA 或中间 MTA 以便于中继。然后远程邮件服务器上的 MDA 从接收的 MTA 获得消息并将它们写入收件人的邮箱以供以后检索。

邮件检索代理

MRA 是用来从服务器的邮箱中获取信息，然后通过邮局协议（POP）或互联网消息访问协议（IMAP）将它们传递到 MUA。从图 6-11 中，我们可以看到一条消息，它从发件人编写到最后阶段被收件人机器获取，穿越了电子邮件系统的多个组件。这些组件需要利用某些协议来传输消息。通常情况下，发件人 MUA 使用 SMTP 将邮件发送到 MTA。因此，MTA 也称为 SMTP 服务器。收件人可以通过 POP 或 IMAP 从邮件服务器获取它的消息。因此，存储消息并处理来自用户检索请求的邮件服务器称为 POP 服务器或 IMAP 服务器。我们将在 6.3.3 节详细讨论这些协议。

6.3.2 互联网邮件标准

电子邮件消息通常由两部分组成，一部分是特殊数据，另一部分是消息正文。特殊数据可以根据所需要的管理目的来划分。第一类是针对传输介质的信息，如发件人和收件人的地址。因此，这种类型的数据称为信封。它可以通过 MTA 作为消息单独传输给收件人。第二类是消息头部，包括主题和收件人的姓名。后跟一个空行和消息正文。

电子邮件消息的基本标准定义在 RFC 822 中，后来因出现了 RFC 2822 被废弃了，然后是 RFC 5322。RFC 822 规定了相当详细的邮件头部格式，邮件正文采用 ASCII 文本。然而，它可以应付各种不断增长的需求，如支持二进制字符、国际字符集和多媒体附件扩展。因此，在 RFC 1341 中提出了一种称为多用途互联网邮件扩展（MIME）的增强版本，后来因为出现了能够处理这些新需求的 RFC 2045～2049 而被废弃。在本节的其余部分中，我们将介绍这两种标准。

RFC 822：互联网消息格式

RFC 822 规定电子邮件消息的句法。在其定义中，一个消息由信封消息内容组成，消息内容包括消息头和正文。表 6-5 总结了 RFC 822 中定义的常用的消息头部字段。每个字段由字段名、一个冒号和一个值（对大多数字段来讲）组成。这些字段可以根据其目的分为不同的类型。

表 6-5 常用消息头部字段

类型	字段	描述
Originator	From:	发送消息的人
	Reply-To:	提供一种适用的机制以便指示响应发送到的邮箱
Receiver	To:	消息的主要收件人
	Cc:	抄送的第二个收件人
	Bcc:	邮件副本接收消息的收件人而没有其他人，包括 To: 和 Cc: 收件人，查看接收到邮件的其他人
Trace	Received:	将该字段的每一个副本添加到中继该消息的每一个传输服务
	Return-Path:	由将该消息发送给其收件人的最终传输系统来添加
Reference	Message-ID:	包含一个唯一的由原始系统上的邮件传输产生的标识符
	In-Reply-To:	该消息回答的前一封邮件
Other	Subject:	提供消息的概要或者说明消息的性质
Date	Date:	提供邮件发送的日期和时间
Extension	X-anything:	用于实现还没有进入 RFC 或将来或该也不会进入 RFC 的功能

Originator 字段指定邮件消息的发件人信息。From：字段表明谁撰写和发送消息。Reply-To：字段指定发件人想要收件人答复发向的地址。当发件人有多个电子邮件地址时，这是很有用的，但想要接收来自最常使用邮箱的响应。

Receiver 指示消息的收件人。一种常见的句法是在任何两个相邻的收件人地址之间用一个逗号分隔。To：字段给出消息的主要的收件人。Cc：字段给出将接收消息副本的收件人地址列表。事实上，在消息发送中的主、次收件人之间没有区别。通常，希望 To：字段中的收件人按照消息采取行动而 Cc：收件人只是接收副本以供参考。Bcc：字段包含一个额外的收件人地址列表，该地址列表将收到密件副本。不同的是，Bcc：收件人对接收消息的其他用户来说是隐藏的。

Trace 提供消息处理历史的审计线索并指出返回邮件发件人的路由。处理消息的每台机器将其机器名、消息 ID、接收消息的日期和时间、来自哪台机器、使用何种传输软件等插入到 Received to：字段。Return-Path：字段由最后发送该消息的传输系统添加。该字段将说明如何将响应路由回消息源。

Message-ID：字段包含原始系统上邮件传输系统生成的一个唯一标识符。它也说明了消息的版本。In-Reply-To：字段标识消息应答的前一封邮件。Subject：字段用几句话描述消息的内容。Date：字段提供消息发送的日期和时间。Extension 用于实现尚未在标准中定义的附加功能。所有用户定义的字段应该有以符串“X-”开始的名字。

图 6-12 是消息头部的例子，说明从 in@cs.netu.edu.tw 向 chhwang@exodus.cs.ccu.edu.tw 发送主题为“book”的消息。消息由 mail.cs.netu.edu.tw 处理，由 smailgate.cs.netu.edu.tw 进行病毒扫描，并最终传递到在 exodus.cs.ccu.edu.tw 上的邮件服务器。

多用途互联网邮件扩展

多用途互联网邮件扩展（MIME）是一种为了增强传统互联网消息格式的规范。MIME 使电子邮件具有

1）支持非 7 位 ASCII 字符集的文本头部和消息正文。

2）在单个消息内传输多个对象。

```
Return-Path: <ydlin@cs.nctu.edu.tw>
Delivered-To: rhhwang@exodus.cs.ccu.edu.tw
Received: from csmailgate.cs.nctu.edu.tw (csmailgate2.cs.nctu.edu.tw [140.113.235.117])
by exodus.cs.ccu.edu.tw (Postfix) with ESMTPS id 431B212B01D
for <rhhwang@exodus.cs.ccu.edu.tw>; Tue, 23 Jun 2009 00:25:52 +0000 (UTC)
Received: from mail.cs.nctu.edu.tw (csmail2 [140.113.235.72])
by csmailgate.cs.nctu.edu.tw (Postfix) with ESMTP id 119193F65F
for <rhhwang@exodus.cs.ccu.edu.tw>; Tue, 23 Jun 2009 00:22:57 +0800 (CST)
Received: from nctuelec065391 (f5bc76.RAS.NCTU.edu.tw [140.113.5.76])
by mail.cs.nctu.edu.tw (Postfix) with ESMTPSA id 0577762148
for <rhhwang@exodus.cs.ccu.edu.tw>; Tue, 23 Jun 2009 00:22:57 +0800 (CST)
Message-ID: <6CF49F76B3C6488AAB184E4A82FFDF66@nctuelec065391>
Reply-To: "Dr Ying-Dar Lin" <ydlin@cs.nctu.edu.tw>
From: "Dr Ying-Dar Lin" <ydlin@cs.nctu.edu.tw>
To: <rhhwang@exodus.cs.ccu.edu.tw>
Subject: book
Date: Tue, 23 Jun 2009 00:22:59 +0800
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="    =_NextPart_000_04F2_01C9F398.C3392310"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2900.5512
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.5579
X-UIDL: mcA"!Ak,"!-Xn!!pg"!
```

图6-12 消息头部的一个例子

3）二进制或特定应用的文件附件。

4）多媒体文件，如图像、音频和视频文件。

MIME定义了新的头部字段，如表6-6所示。虽然RFC 822曾经是互联网消息格式的唯一标准，但仍然存在以下情况，即当邮件处理代理需要知道消息是否是用新的标准编写的。因此，MIME-Version：字段用来声明所使用的互联网消息格式。Content-Type：字段用来描述消息正文中包含的数据，这样MUA可以挑选一种合适的机制将数据提供用户。它通过给出类型和子类型标识符并提供某些类型所需要的参数来指定正文或正文部分中数据的性质。在一般情况下，顶级媒体类型声明数据的一般类型，而子类型则为该数据类型指定一个特定的格式。Content-Type：字段的句法是：

```
Content-Type := type "/" subtype ";" parameter]...
```

表6-6 MIME头部字段

字　段	描　述
MIME-Version:	描述MIME消息格式的版本
Content-Type:	描述MIME内容类型和子类型
Content-Transfer-Encoding:	指示传输的编码方法
Content-ID:	允许一个消息正文指向另外一个消息正文
Content-Description:	对信息正文的可能描述

有7个预定义的内容类型，其基本特征总结在表6-7中。文本(text)类型主要用于发送文字形式的资料。多数(multipart)表示数据的正文由多个部分组成，每个部分有其自己的数据类型。消息类型表示一种封装过的消息。应用(application)类型表示不符合任何其他类型的数据，如不能由邮件应用程序处理，翻译的二进制数据或信息。图像(image)和音频(audio)类型分别表示图像和音频数据。视频(video)类型表示正文包含一个随时间变化的图像映像，可能带有颜色和协调的声音。

表 6-7 MIME 内容类型集合

类 型	子 类 型	重要参数
text	plain，html	Charset
multipart	mixed，alternative，parallel，digest	Boundary
message	RFC 822，partial，external body	Id，number，total，access-type，expiration，size，permission
application	octet-stream，postscript，rtf，pdf，msword	type，padding
image	jpg，gif，tiff，x xbitmap	无
audio	basic，wav	无
video	Mpeg	无

许多内容类型是以其自然的格式（如 8 位字符或二进制数据）表示的。这些类型的消息可通过各种网络发送。然而，有些传输协议不能传输这些数据。例如，SMTP 将邮件消息限制为 7 位 US-ASCII 数据，行不超过 1000 个字符，包括所有的后缀 CRLF 行分隔符。这样，MIME 编码具有非 ASCII 部分的消息。Content-Transfer-Encoding：字段告诉收件人消息正文的编码方式以及如何将其解码。该字段可能的值是：

- **可打印字符引用编码**：它的目的是插入那些与 US-ASCII 字符集中可打印字符相对应的字节组成的数据。这里行不超过 76 个字符。在第 75 个字符后，其余的被截断并用一个“=”标记作为转义字符。
- **Base64 编码**：这是对数据或者使用 MIME 邮件程序的人有意义的其他文本。base64 使用 US-ASCII 字符集的一个 65 个字符的子集来编码和解码字符串。与可打印字符引用编码一样，行不超过 76 个字符。
- **7 位**：这是默认值，意思是消息内容是纯 ASCII 文本。
- **8 位**：这是由 8 位字符构成的数据，带有以 CRLF 结束的短行。
- **二进制**：与 8 位编码一样，但没有 CRLF 行分界。
- **X 编码**：它代表任何非标准的内容传输编码（Content Transfer Encoding）。因此，任何附加值都必须有一个以“x-”开头的名字。

在构造一个高级用户代理时，可能希望允许一个消息正文指向另一个消息正文。消息正文可能根据 Content-ID：字段相应地加上标记，此字段的语法与 RFC 822 中的 Message-ID：字段相同。Content-ID 的值应该尽可能地保持唯一。Content-Description：字段用于为给定的消息正文放置一些描述性信息。例如，将一个“image”消息正文标记为“The front cover of the book”（书的封皮）将非常有用。这可以让收件人知道这张图片的意义。

图 6-13 显示了一个 MIME 消息的例子。这张图片使用 base64 编码进行编码。

```
From: 'Yi-Nong Lin' <ynlin@cs.nctu.edu.tw>
To: ynlin@cs.nctu.edu.tw
Subject: Cover
MIME-Version: 1.0
Content-Type: image/jpg;
      name=cover.jpg
Content-Transfer-Encoding: base64
Content-Description: The front cover of the book

<... base64 encoded jpg image of cover...>
```

图 6-13 一个 MIME 消息的例子

6.3.3 互联网邮件协议

电子邮件系统依靠邮件协议在客户端和服务器之间传输消息。这里我们将介绍 3 种常用的邮件协议——SMTP、POP3 和 IMAP4。正如 6.3.1 节中所述，SMTP 用于从邮件客户端向邮件服务器发送消息，即从 MUA 到 MTA，也在服务器之间传输消息，即 MTA 到 MTA。POP3 用于客户端检索来自服务器的消息，即从 MRA 到 MUA。IMAP4 类似于 POP3，但它提供一些额外的功能，如存储和处理邮件服务器上的邮件。我们接下来将介绍这些协议。

简单邮件传输协议

简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）首次定义在 RFC 821 中。在 RFC 2821 和 RFC 5321 出现后被废除，它是一个标准的主机到主机的邮件传输协议，默认使用 TCP 协议，端口号为 25。监听端口 25 并使用 SMTP 协议的守护进程称为 SMTP 服务器，即 MTA。SMTP 服务器处理从发件人或其他邮件服务器发送的邮件。它接收输入连接，然后将消息发给正确的收件人或者下一台 SMTP 服务器。如果一台 SMTP 服务器无法向某一个具体的地址发送消息，而且错误不是由于永久性拒绝，那么将消息放置到消息队列中以供稍后再次传输。发送尝试会持续下去直到发送成功或者 SMTP 放弃为止，放弃时间一般至少 4～5 天。如果 SMTP 服务器放弃发送，它就返回一个带有错误报告的不可达消息给发送方。

一个 SMTP 客户端（MIA 或者 MTA）与一个 SMTP 服务器（MTA）建立一条双向传输信道之后，客户端就可以产生并向服务器发送 SMTP 命令。服务器发送 SMTP 应答回复客户端作为响应。表 6-8 列出了一些重要的 SMTP 命令。HELLO 用于在会话的开始标识 SMTP 客户端到 SMTP 服务器。MAIL FROM：通知 SMTP 服务器邮件发送者是谁。它在指定每条消息的收件人之前或者在 RSET 之后使用。RCPT TO：向邮件服务器声明谁发送了消息。允许有多个收件人，但每个收件人都必须在 RCPT TO：中列出其自己的邮箱地址，该命令紧跟在 MAIL FROM：之后。DATA 指示邮件数据。任何在 DATA 后输入的内容都将作为邮件的正文对待，并且将发送给收件人。邮件数据将终止于字符序列“<CRLF>.<CRLF>”，这是一行包含“.”（句点）的新行后跟着另外一行新行。当输入句点时，消息将排队或立即发送。RSET 重置当前会话的状态，当前事务的 MAIL FROM：和 RCPT TO：都被清除。最后，利用 QUIT 关闭会话。

表 6-8 重要的 SMTP 命令

命　令	描　述	命　令	描　述
HELO	用发送者的域名问候接收者	DATA	指示邮件数据，一行中列出一个“.”断开
MAIL FROM：	指示发送者，但也可能会被伪造	RSET	重置会话
RCPT TO：	指示接收者	QUIT	关闭会话

当 SMTP 服务器接收来自客户端的命令时，服务器用 3 位数字的代码响应，指示命令是成功还是失败。表 6-9 总结了响应代码。200 或 2xx 响应意味着前一个命令执行成功。

表 6-9 SMTP 应答

响　应	描　述	响　应	描　述
2xx	接收并处理命令	4xx	关键系统或传输故障
3xx	查询课户端	5xx	SMTP 命令错误

学习了 SMTP 命令和应答的句法和语义后，让我们看看在图 6-14 的例子会话中的服务器和客户端之间的互动。yulin 给 yulin 发送了一封电子邮件。注意“B”来自接收服务器的响应，而“S”是发送客户端的输入。

邮局协议版本 3

邮局协议版本 3（Post Office Protocol version 3，POP3）首次定义在 RFC 1081 中，后来因为 RFC

```
R: 220 mail.cs.nctu.edu.tw Simple Mail Transfer Service Ready
S: HELO CS.NCTU.EDU.TW
R: 250 MAIL.CS.NCTU.EDU.TW Hello [140.113.235.72]
S: MAIL FROM:<cynlin@CS.NCTU.EDU.TW>
R: 250 OK
S: RCPT TO:<cyl@in@CS.NCTU.EDU.TW>
R: 250 2.1.5 <cyl@in@CS.NCTU.EDU.TW>
S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: ...mail content...
S: .
R: 250 2.6.0 <SX4MaY5AYg0000000010@CS.NCTU.EDU.TW> Queued mail for delivery
S: QUIT
R: 221 mail.cs.nctu.edu.tw Service closing transmission channel
```

图 6-14 一个 SMTP 会话

1225、RFC 1460、RFC 1725、RFC 1939 现已规被废除。它用于用户对邮件的访问。监听端口 110 并使用 POP3 的守护程序称为 POP3 服务器。POP3 服务器接收来自客户端的连接并检索它的发送的信息。当客户端和服务器之间建立一条 TCP 连接时，服务器向客户端发送一条欢迎信息，然后就能与客户端交换命令和应答。

一个 POP3 会话在其整个生命周期内要经历很多状态。这些状态包括 AUTHORIZATION（授权）、TRANSACTION（交易）和 UPDATE（更新）。一旦客户端连接到 POP3 服务器上并且接收来自服务器的一条欢迎信息，那么会话就进入 AUTHORIZATION 状态。然后，为了验证身份，客户端必须告诉 POP3 服务器他的用户名和密码。当客户端通过身份检查后，会话就进入到 TRANSACTION 状态。这下，客户端可以向服务器发送命令并请求服务器执行命令，例如，列出收件箱中的邮件。当客户端发出 QUIT 命令时，会话进入 UPDATE 状态。在这个状态中，POP3 服务器释放所有在 AUTHORIZATION 状态中分配给客户端的资源，发送“再见”给客户端，并且最终关闭和客户端之间的连接。

表 6-10 总结了一些必要的 POP3 命令。第二列指示命令所属的会话状态。USER 和 PASS 用于标志在 AUTHORIZATION 状态的客户端。STAT 用于从邮箱中获取邮件数量和邮件字节大小。LIST 用于获取一封或多封邮件的大小。如果在 LIST 后跟一个加外的名字作为参数，将报告这封邮件的信息。RETR 用于从邮箱查询获取邮件。DELE 标识邮件被删除，未来任何在 POP3 命令引用这种标记消息都产生一个错误。注意标注删除的邮件实际上并未被删除直到 POP3 会话输入 UPDATE 状态为止。NOOP 代表无操作，对此 POP3 服务器除了发送一个肯定应答外不做任何事情。RSET 将所有标注为删除的邮件重置为未标注。最后，QUIT 将 POP3 会话实施 UPDATE 状态，然后终止会话。

表 6-10 最少的 POP3 命令

命 令	描 述	会话状态
USER *name*	向服务器标识用户	AUTHORIZATION
PASS *password*	输入用户密码	AUTHORIZATION
STAT	获取邮箱中邮件数量和字节大小	TRANSACTION
LIST [*msg*]	获取一个或所有邮件的大小	TRANSACTION
RETR *msg*	从邮箱中获取一个邮件	TRANSACTION
DELE *msg*	将 *msg* 标记删除从邮箱中删除	TRANSACTION
NOOP	无操作	TRANSACTION
RSET	将所有标记为已删除的邮件重置为未删除状态	TRANSACTION
QUIT	终止会话	AUTHORIZATION、UPDATE

所有 POP3 应答都从一个状态行开始。状态行包括一个状态指示符和一个关键词，可能后面还跟了额外的信息。目前有两种状态指示符：肯定（“+OK”）和否定（“-ERR”）。额外的信息紧跟在命

令结果单行上的状态指示符之后。

图 6-15 显示了一个 POP3 会话。注意“S”是来自服务器的响应，而“C”是客户端的输入。在这个例子中，一个用户登录到 POP3 服务器。首先，他列出他邮箱中的所有邮件。然后，他检索一个邮件并终止会话。

```
S: +OK POP3 Server mail.es.retrasdate
C: USER ydlin
S: +OK send your password
C: PASS *******
S: +OK maildrop locked and ready
C: ejqwa
S: -ERR illegal command
C: STAT
S: +OK 1 296
C: LIST
S: +OK 1 messages (296 octets)
C: RETR 1
S: +OK 296 octets
... <server start to send the mail content> ...
C: QUIT
S: +OK ydlin POP3 server signing off (maildrop empty)
```

图 6-15 一个 POP3 会话

互联网消息访问协议 v4

互联网消息访问协议 v4（Internet Message Access Protocol Version 4，IMAP4）首次定义在 RFC 2730，因为 RFC 2060 和 RFC 3501 的出现而被废除，它用来代替 POP3 协议。它为了满足在任意地方使用 Web 浏览器访问服务器上的电子邮件而不用实际下载它们的需要而产生的。IMAP4 和 POP3 主要的区别在于，IMAP4 允许在邮件系统上查找和处理邮件，而 POP3 只允许用户在自己的机器上下载、存储和处理他们的邮件。监听端口 143 并运用 IMAP4 的守护进程称为 IMAP4 服务器。IMAP4 让用户在任何 PC 机上使用 IMAP4 邮件客户端读取。回复并在 IMAP4 服务器上层次化的文件夹上存储邮件，让客户端邮件与 IMAP4 服务器同步。

IMAP4 会话要经过三个阶段：客户端/服务器连接的建立，从服务器发来的初始化欢迎信息和客户端/服务器的交互。一次交互由客户端命令、服务器数据和服务器完成响应组成。IMAP4 服务器可能处于 4 种状态之一。大多数命令只有在特定状态下是有效的。4 个状态分别是：

1）Non-authenticated　当在 IMAP4 服务器和客户端之间建立连接时，服务器就进入 non-authenticated 状态。在大多数命令允许执行前，客户端必须提供认证凭证。

2）Authenticated　当开始一个预认证连接时，在提供可接受的认证凭据或者发生一个邮箱选择错误后，服务器就进入 authenticated 状态。在 authenticated 状态，在影响邮件消息的命令获得许可之前客户端必须选择一个邮箱进行访问。

3）Selected　当已经成功选择一个邮箱后，服务器就进入 selected 状态。在这个状态中，已经选择了要访问的邮箱。

4）Logout　当客户端请求退出服务器时，服务器就进入 logout 状态。此时，服务器将关闭连接。

表 6-11 列举了 IMAP4 命令。这里我们不解释每一条命令。总之，IMAP4 包括创建、删除和重命名邮箱的操作；检查新邮件；永久删除邮件；设置和清除标志；RFC 822 和 MIME 信息解析和搜索；以及选择性获取消息属性、文本和其中一部分。IMAP4 服务器上的邮件利用邮件序列号和唯一标识符来进行访问。

表 6-11　IMAP4 命令

会话状态	命　令
Any（任何）	CAPABILITY，NOOP，LOGOUT
Non-authenticated	AUTHENTICATE，LOGIN
Authenticated	SELECT，EXAMINE，CREATE，DELETE，RENAME，SUBSCRIBE，UNSUBSCRIBE，LIST，LSUB，STATUS，APPEND
Selected	CHECK，CLOSE，EXPLANDE，EXAMIN，FETCH，SCORE，COPY UID

每一个 IMAP4 命令都从带有一个名为“标签”的标识符（通常是一个简短的字母数字字符串，如 A001、A002 等）开始。每一个发送的命令必须使用一个唯一的标签。在两种情况下，客户端命令可以不完整地发送。这两者之中的任意一种情况下，客户端发送不带标签的命令中的第二部分，而服务器

会使用一个以符号“+”开始的行对此命令进行响应。客户端在发送另一个命令之前必须完成发送完整的命令。

在 IMAP4 中的状态表下以批量标签也可以不设置标签。一个有标签的状态响应指示一个带有匹配标签的客户端命令已经完成。不带标签的状态响应指示服务器正在发送其他不是命令完成的服务器状态。服务器状态响应可以有三种形式：状态响应、服务器数据和命令继续请求。客户端必须随时准备接收以下响应：

1）**状态响应** 状态响应既可以表示完成客户端命令的结果（OK、NO、BAD），也可以表示服务器问候和警报（PREAUTH 和 BYE）。

2）**服务器数据** 准备收发客户端必须记录某些服务器数据，就像该数据的描述一应该注意的一样。数据传递给响应所有后续命令和响应解释的关键信息。从服务器传输给客户端的数据和开下表示命令完成的状态响应称为无标签的响应。每一个无标签的响应是以“*”字符作为前缀的。

3）**命令继续请求** 这些响应表示服务器已准备接收来自客户端的后续命令。这个响应的其余部分就是一文本行。

图 6-16 显示了一个 IMAP4 会话的例子。首先利用他的用户名和密码登录到 IMAP4 服务器。认证后，下载作“收件箱”邮箱。用户获取一条消息，将消息标志成被删除，最后登出会话。

```
S: * OK Dovecot ready.
C: a001 login user passwd
S: a001 OK Logged in.
C: a002 select inbox
S: * FLAGS (Answered Flagged Deleted Seen Draft $unknown 3 unknown 4 unknown 5 NonJunk
$MDNSent Junk $Forwarded)
S: * OK [PERMANENTFLAGS (Answered Flagged Deleted Seen Draft $unknown 3
$unknown 4 unknown 5 NonJunk $MDNSent Junk $Forwarded \*)] Flags permitted.
S: * 583 EXISTS
S: * 0 RECENT
S: * OK [UNSEEN 539] First unseen.
S: * OK [UIDVALIDITY 1243566168] UIDs valid
S: * OK [UIDNEXT 5146] Predicted next UID
S: a002 OK [READ-WRITE] Select completed.
C: a003 fetch 2 full
S: * 2 FETCH (FLAGS (\Seen) INTERNALDATE "05-Apr-2009 17:20:01 +0800"
   RFC822.SIZE 2164 ENVELOPE ("Sat, 5 Apr 2009 17:50:01 +0800"

   "=?big5?B?Rm6lFBIOiC4ZZZP+Mcj5KTOssOsScV9ssbicKc=?="
   (("dhhuang" NIL "dhuang" "dhwang@exodus.cs.ccu.edu.tw")) (("dhhuang" NIL "dhhuang"
   "dhwang@exodus.cs.ccu.edu.tw")) (("dhhuang" NIL "dhhuang" "dhwang@exodus.cs.ccu.edu.tw"))
   BODY ("text" "html" ("charset" "big5") NIL NIL "base64" 7223 32))

S: a003 OK Fetch completed
C: a004 fetch 2 body[header]
S: * 2 FETCH (BODY[HEADER] {384}
S: From: "dhhuang" <dhwang@exodus.cs.ccu.edu.tw>
S: To: "ydlin" ydlin@cis.nctu.edu.tw>
S: Subject: =?big5?B?Rm6lFBIOiC4ZZZP+Mcj5KTOssOsScV9ssbicKc=?=
S: Date: Sat, 5 Apr 2009 17:50:01 +0800
S: MIME-Version: 1.0
S: Content-Type: text/html; charset="big5"; Content-Transfer-Encoding: base64
S: X-Priority: 3; X-MSMail-Priority: Normal; X-MimeOLE: Produced By Microsoft MimeOLE
S: a004 OK Fetch completed.
C: a005 store 2 +flags \deleted
S: * 2 FETCH (FLAGS (\Deleted \Seen))
S: a005 OK Store completed.
C: a006 logout
S: * BYE Logging out
S: a006 OK Logout completed.
S: Connection closed by foreign host.
```

图 6-16 一个 IMAP4 会话

历史演变：基于 Web 的邮件与桌面邮件

Webmail 是一种通过 Web 浏览器访问的电子邮件服务，而不是桌面电子邮件程序，如 Microsoft Outlook 或 Mozilla 的 Thunderbird。2008 年一项由 USA Today 进行的调查显示，使用率前 5 位的 Web 邮件供应商为 Microsoft Windows Live、Hotmail、Yahoo! Mail、Google Gmail 和 AOL Mail。这些供应商同时也向用户提供桌面电子邮件服务以取回电子邮件。与桌面电子邮件 Webmail 相比有两个优点：随时随地的可访问性和几乎可忽略不计的低维护成本。使用 Webmail，可以通过 IMAP4 命令在远程电子邮件服务器上维护和操作电子邮件。相反，桌面电子邮件服务则需要客户端使用 POP3 或者 IMAP4 命令从电子邮件服务器取回邮件，并且需要用户将邮件存储在本地用户计算机上。然而，桌面电子邮件有两个优点：拥有电子邮件完全的控制权；能够有效地访问存储在本地的电子邮件。有趣的是，由于需要完全的控制，所以工程师和科学家，更倾向于选择桌面电子邮件而不是 Webmail。

在 Webmail 服务中有两个接口：1）在客户端和首端邮件服务器之间使用 GET 和 POST 等 HTTP 命令的 Web 接口；2）在前端 Webmail 服务器和后端电子邮件服务器之间使用 POP3/IMAP4 命令的电子邮件接口。首端 Webmail 服务和后端电子邮件服务器就可以单独也可以集成到一起，分别如图 6-17a 和 6-17b 所示。在图 6-17b 中，将两个接口集成到了一台机器上。



图 6-17 Webmail 服务体系结构

开源实现 6.2：qmail

概述

qmail 是一个为类 UNIX 系统设计的安全、可靠、有效和简单的 MTA。其目的是为了替代互联网上最流行的 MTA——sendmail。至今为止，qmail 已经成为互联网上位居第二的最普及的 SMTP 服务器，并且是所有 SMTP 服务器中增长速度最快的。这里我们不介绍 sendmail 的原因是，它的程序和配置文件不容易理解。我们首先介绍 qmail 系统结构、控制文件和数据流。然后我们详细介绍 qmail 的队列结构。

总之，qmail 是一个面向连接、有状态的 SMTP（端口 25），POP3（端口 110）和 IMAP4（端口 143）协议的并发实现。它还支持 MIME 消息。

框图

电子邮件系统执行各种任务，如处理输入的消息、管理队列、发送消息给用户。从程序结构的角度，sendmail 是整体单一的（monolithic），意思是所有功能在一个庞大的、复杂的程序中实现。这会导致更多安全缺陷并且更难以维护程序。但是，qmail 是模块化的（modular），意思是整个 qmail 系统由多个模块组成。每个 qmail 模块都很小且简单，从而能有效地执行特定的任务。模块化设计使得每个程序尽可能以最小权限运行，因此可增强安全性。由于它的优秀设计，qmail 也易于安装和管理。qmail 的核心模块以及它们的功能如表 6-12 所示。图 6-18 显示了这些核心模块的框图。在这些框图中指示的数据流将会在算法实现中进行说明。

表 6-12 qmail 的核心模块

模块	描述	模块	描述
qmail-smtpd	通过 SMTP 接收邮件	qmail-lspawn	调度本地发送
qmail-inject	预处理并发送一封邮件	qmail-local	发送或转发邮件
qmail-queue	将邮件排队以便发送	qmail-rspawn	调度远程发送
qmail-send	从队列中发送邮件	qmail-remote	通过 SMTP 发送邮件
qmail-clean	清除队列目录	qmail-pop3d	通过 POP3 发布邮件

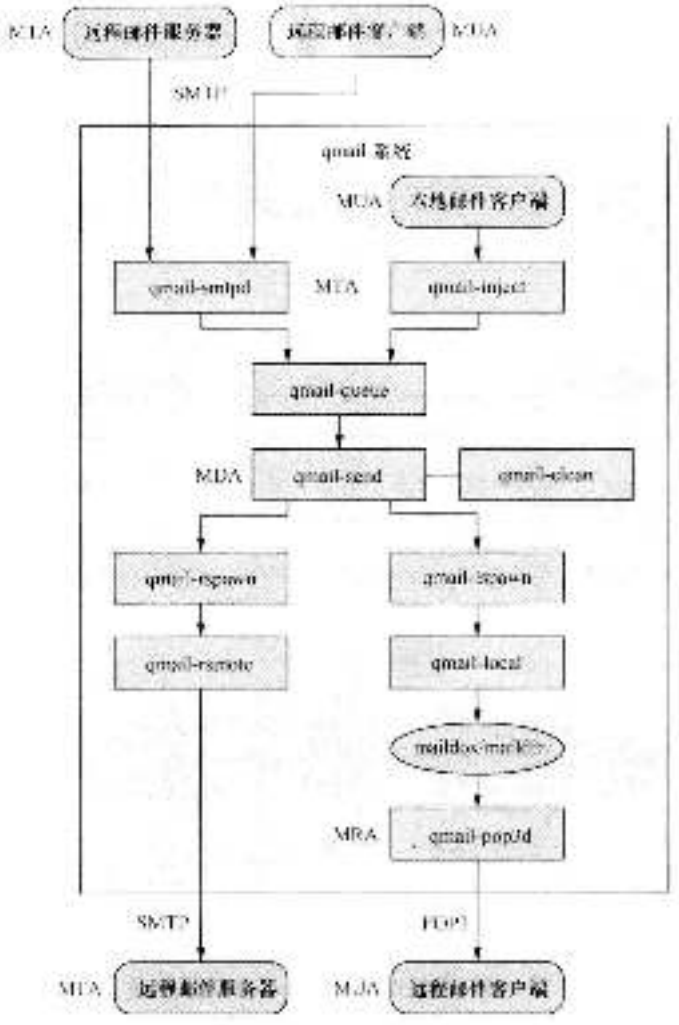


图 6-18 在 qmail 套件中的数据流

数据结构

qmail 使用许多配置文件来更改系统的行为。这些文件位于/var/qmail/control 目录下。在启动 qmail 系统之前，为了所需要的配置，我们需要修改部分文件。表 6-13 列出了一些控制文件。这里我们介绍 3 种最重要的文件。me 文件将本地主机的完全合格域名（Full Qualified Domain Name，FQDN）。rcpthosts 记录 qmail 将要接收邮件的所有主机。注意，所有本地域必须包括在该文件中。locals 包含本地主机，即发送到这些主机的邮件应该传递给本地用户。

表 6-13 qmail 的一些控制文件

控　制	默　认	使 用 者	描　述
me	系统的 FQDN	各　种	对于许多控制值是默认的
rcpthosts	（无）	qmail-smtpd	smtp 接收邮件的表
locals	me	qmail-send	qmail 本地发送的域
defaultdomain	me	qmail-inject	默认的域名字
plusdomain	me	qmail-inject	添加到任何以加号（+）结束的主机名后
virtualdomains	（无）	qmail-send	虚拟域和用户

qmail 队列结构

qmail 将接收的邮件暂时存储在一个中央队列目录中以供稍后发送。这个目录位于/var/qmail/queue，该目录中有许多子目录用来存储不同的信息和数据。表6-14 描述了这些子目录以及它们包含的内容。

表6-14 qmail 中的子目录及其内容

子目录	内容	子目录	内容
Bounce	永久发送错误	Mess	消息文件
Info	信封发送者地址	Pid	qmail-queue 使用以便获取一个一结点号
Intd	由 qmail-queue 构造的信封	Remote	远程信封接收者地址
Local	本地信封接收者地址	Todo	完整的信封
Lock	锁文件		

邮件从进入到离开 qmail 队列会经过许多子目录，如图6-19 所示。这包括3个阶段：1）邮件进入队列；2）预处理队列中的邮件；3）发送预处理的邮件。

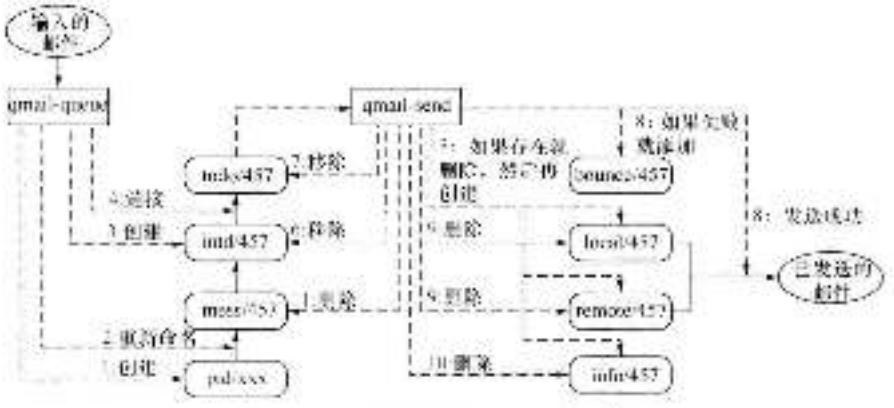


图6-19 邮件如何通过 qmail 队列传递

进入队列

对于输入的邮件，qmail-queue 首先在"pid"目录下创建一个唯一文件名的文件。文件系统给该文件分配一个唯一的"inode"号，如457。qmail-queue 用这个唯一的 inode 号标识邮件。qmail-queue 重命名新创建的文件，pid/命名成任意名，如 mess/457，然后把邮件写入 mess/457。然后，qmail-queue 创建另一个新文件 intd/457，并给它写上信封信息。接下来，qmail-queue 将 intd/457 链接到 todo/457。完成这一步后，邮件已成功地排队，并进行了预处理。

邮件预处理

邮件预处理的目的是让 qmail-send 来决定哪个接收方是本地的而哪个接收方是远程的。当 qmail-send 找到 todo/457 时，它首先删除 info/457、local/457 和 remote/457，如果它们存在。然后它读取 todo/457，创建 info/457，和可能的 local/457、remote/457。之后，它删除 intd/457 和 todo/457。此时邮件的预处理完成。现在 local/457 或 remote/457 包含接收方的地址。每一个地址或者标记为 NOT DONE，或者标记为 DONE。NOT DONE 和 DONE 的定义如下所述。

NOT DONE 如果邮件没有会试传递，那么它们遇到了临时错误。稍后 qmail-send 将尝试向这个地址传递邮件。

DONE 邮件成功传递或者最后一次传递尝试永久失败。不管是哪种情况，qmail-send 将不再尝试向这个地址传递邮件。

邮件传递

qmail-send 在其空闲时间将邮件发送到 NOT DONE 地址。如果接下来的邮件传递成功，那么它将

就把此标识为 DONE。如果遇到一个永久发送失败，那么 qmail-send 将首先在 bounce/457 上附加一条注释，如果需要，就创建 bounce/457，然后再将该地址标记为 DONE。注意，qmail-send 向 bounce/457 添加一条新的退信通知时到来 bounce/457。qmail-send 不断从待邮件发送给 todo/457 和 remote/457 中的地址。当所有地址都待定之后，qmail-send 删除 local/457 和 remote/457。然后 qmail-send 到除该邮件一首先，qmail-send 检查 bounce/457 是否存在。如果 bounce/457 存在，则 qmail-send 以上述方式对其进行处理。一旦 bounce/457 被到除，qmail-send 就删除 todo/457，最后再删除 mess/457。

算法的实现

设置 qmail 主进程运行后，它就准备对接收自发送者的邮件。一封邮件被 qmail 接收，都从并获得提交的接收者，可能经过许多个模块。图 6-38 显示了 qmail 整个邮件处理过程。首先，环境接收自发送者的邮件。这个程序可能直接通过 SMTP 发送邮件给 qmail-smtpd，也可能是本地生成邮件给 qmail-inject。qmail-queue 被 qmail-smtpd 或者 qmail-inject 调用将邮件放进一个队列目录中。然后邮件由 qmail-send 与 qmail-lspawn 或者 qmail-rspawn 合作发送，最后由 qmail-clean 清理。如果邮件发送给本地用户，那么 qmail-lspawn 调用 qmail-local 将邮件存储到接收者的投递邮件目录中。如果邮件的接收者不是本地用户，那么 qmail-rspawn 就调用 qmail-remote 将邮件发送到接收者的邮件服务器。本地系统上的接收者可以通过 qmail-pop3d 取回他们的邮件。值得注意的是，qmail-send、qmail-clean、qmail-lspawn 和 qmail-rspawn 是一直运行的后台守护进程，其他的仅在需要时才被调用。

练习

1. 找到实现 qmail-smtpd、qmail-remote 和 qmail-queue 的 .c 文件和代码行。
2. 在 qmail 结构的对象中找到 qmail 队列的准确结构定义。
3. 找出电子邮件是如何存储在邮箱和邮件目录中的。

6.4 万维网

由于简单自遗天，万维网（World Wide Web，WWW）为互联网的快速增长做出了贡献，并通过信息共享改变了两个世界。通过对匿名 FTP、Archie、Gopher 和 WAIS 等匿名信息共享服务的开发，万维网更进一步对寻址方式进行标准化并简化，将其成为统一资源定位符（Universal Resource Locator，URL），将多媒体内容格式转换为超文本标记语言（HyperText Markup Language，HTML）和未来的扩展标记语言（eXtensible Markup Language，XML），并将应用协议转化为超文本传输协议（HyperText Transfer Protocol，HTTP）。本节首先介绍使用 URL 的 Web 命名和寻址以及其他类似的方案。然后我们描述 HTML、XML 和 HTTP，还将探讨 Web 缓存和代理功能。最后，将 Apache 作为开源实现的例子，对其体系进行简要分析。

6.4.1 简介

万维网为统一地获取知识提供了一种网络空间，并且它允许来自不同地点的合作者分享他们的想法和一个有关项目的主方面。除非两个项目是软件开发的两个是独立的，否则来自双方的结果是不可能集成一个混合的工作。从 Tim Berners-lee 在欧洲核子研究组织（CERN）的一个项目开始，WWW 已经成为自 1989 年以来各种信息检索中最流行的媒介。

通过使用像微软 Internet Explorer（IE）这样的商用浏览器或者其他的新兴浏览器（如 Firefox，Chrome 和 Opera Web），用户浏览如图 6-20 所示的基本步骤就可以访问任何在线的 Web 网页。首先，将指定讨论的统一资源定位符（URL）中的服务器名字，通过 DNS 解析为 IP 地址。浏览器通过 TCP 三次握手连接到这个特殊 IP 地址上的 TCP 端口（一般是端口 80）的 Web 服务器。一旦 TCP 连接建立，浏览器就发出一个 HTTP 请求以便得到 Web 服务器上的资源。另一个请求的资源是一个 HTML 网页。Web 浏览器对二解析这个 Web 网页，对 Web 页面上的图片和文件还可能发出额外的请求。

1996 年 RFC 1945 对 HTTP1.0 进行了标准化，而 1997 年 RFC 2068 对 HTTP1.1 进行了标准化，1999 年



图6-20 Web客户端如何与Web服务器交互

由于RFC 2616的出现而被废除。1995年RFC 1866定义了HTML，2000年因RFC 2854的出现而被废除。1995年RFC 1801定义了统一资源标识符（Uniform Resource Identifier，URI），2005年由于RFC 3986的发布而被废除。

6.4.2 Web命名和寻址

Web是一个用超链的网页和文档形成的信息空间。Web上的信息单元称为资源。如何在这个空间中寻找和操作资源是一个重要问题。Web命名是一个对Web上的资源命名机制，而Web寻址则提供访问资源的途径。URI是一种标识Web资源的短字符串。URI通过多种命名方案来访问方法使资源可用。统一资源定位符（URL）是URI的一个子集。用于描述Web上可访问的资源地址。另一种URI是统一资源名称（Uniform Resource Name，URN）。URN是一个全球范围的名称，而不是指位置。图6-21显示了URI、URL和URN之间的关系。注意，URL用于定位或者寻找资源，而URN用于标识。

统一资源标识符

统一资源标识符是一个用于标识抽象或物理资源的紧凑字符串。任何资源，无论它是文本页面、图片、视频或音频剪辑，还是程序，都有一个用URI编码的名字。一个URI通常由3个部分组成：

1）用于访问资源的命名方案。

2）放置资源的机器名字。

3）资源本身的名字，以目录或者文件名的方式给出。

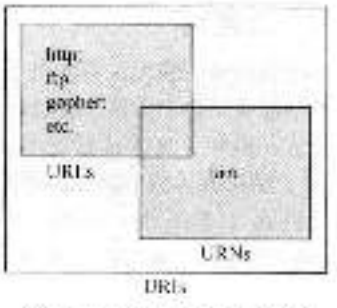


图6-21 URI、URL和URN之间的关系

URI的通用句法包括绝对和相对两种形式。绝对标识符指的是独立于当前上下文的资源，而相对标识符指的是它与当前上下文URI的差别来标识的资源。绝对URI的句法是：

```
<scheme>:<scheme-specific-part>#<fragment>
```

其中包括3部分：使用的方案名称（<scheme>），一个解释取决于该方案的字符串（<scheme-specific-part>），以及一个可用于传送额外参考信息的可选的分段标识符（<fragment>）。

URI的子集共享常用的句法，表示命名空间中的层次化关系，就得到"通用的URI"。

```
<scheme>://<authority><path>?<query>#<fragment>
```

其中，具体方案部分进一步细分成<authority>、<path>和<query>几个组件。许多URI方案包括一个用于命名权威的顶级层次元素，因此<authority>用来管理其他URI定义的命名空间。<path>组件包含特定权威的数据，用来标识指定的方案和权威的资源。<query>组件是由资源解释的信息。

有时，URI也可以采取相对URI的形式，其中方案和权威组件常常省去。其路径通常指向与当前上下文位于同一台机器上的资源。相对URI的句法为

```
<path>?<query>#<fragment>
```

图6-22显示了一些URL例子。第一个例子的URL解释如下：位于服务器speed.cs.nctu.edu.tw上的书籍信息可以利用HTTP协议通过路径~/ydlin/index.html访问。最后的例子是相对URL。假设我们有某本URL为http://www.cs.nctu.edu.tw/，则上例子中的URL将扩展成完整的URL：http://www.cs.nctu.edu.tw/icons/logo.gif。

```
http://speed.cs.nctu.edu.tw/~ydlin/index.html#books
http://www.google.com/search?q=linux
ftp://ftp.cs.nctu.edu.tw/Documents/RFC/rfc2400-2499/rfc2396.txt
mailto:ydlin@cs.nctu.edu.tw
news:comp.os.linux
telnet://bbs.cs.nctu.edu.tw/
../icons/logo.gif
```

图6-22 部分URL例子

统一资源定位符

统一资源定位符是互联网上资源定位的紧凑字符串表示形式。它是URI的一种形式。URL既可以将人也可以将软件导向各种信息，通过许多不同的互联网协议提供。URL的通用句法如下：

```
<service>://<user>:<password>@<host>:<port>/<url-path>
```

部分或全部<user>、<password>@、":"<password>、":"<port>和/<url-path>可能被忽略。在前面句法中，<service>指向提供资源的具体方案。这里讲述的方案在表6-15中列出。具体方案之后跟着以双斜线//开始的数据。<user>和<password>是可选的用户名和密码。如果存在，用户名和密码就用冒号":"隔开，后跟符号@。<host>指示域名或网络主机的IP地址。<port>和主机之间用冒号":"隔开。<port>是要连接的主机端口号。<url-path>指定如何寻址特定资源的详细情况。注意，在host（或port）和URL路径之间的单斜线/不是URL路径的一部分。

表6-15 URL中的具体方案

服务	说明	服务	说明
ftp	文件传输协议	nntp	使用NNTP访问USENET新闻
http	超文本传输协议	telnet	双向交互式会话
copaca	Copher协议	wais	广域信息服务器
mailto	电子邮箱地址	file	特定主机上文件名
news	USENET新闻	prospero	Prospero目录服务

图6-23显示了一些URL的例子。第一个URL指示Web站点www.cs.nctu.edu.tw上图像文件的位置，而第二个是通过SSL（安全套接字层）协议（由"https"服务方案指定）可访问的计算机学院Webmail站点。第三个URL表示FTP服务器ftp.cs.nctu.edu.tw可提供的一个文本文件。在这个例子中，用户用他的用户名"john"和密码"secret"登录到FTP服务器。第四个例子是指在新闻服务器news.cs.nctu.edu.tw上的网络新闻组cs.course.computer的编号为5238的新闻文章，最后一个URL显示了一个可以通过端口110利用telnet协议访问的交互式服务。

```
http://www.cs.nctu.edu.tw/chinese/ccg/titleMain.gif
https://mail.cs.nctu.edu.tw/
ftp://john:secret@ftp.cs.nctu.edu.tw/projects/book.txt
nntp://news.cs.nctu.edu.tw/cis.course.computer-networks/5238
telnet://mail.cs.nctu.edu.tw:110/
```

图6-23 一些URL例子

统一资源名称

在Web网页上，URL提供给定资源的位置。如果将资源移动到另一个位置，那么它的URL也会

变化。URN 通过为资源提供永久标识符试图来克服这个问题。URN 是一种独立于位置的名字，用于标识 Web 上的资源。URN 句法，包括 4 个部分，它们分别是：

```
<URN> ::= "urn:" <NID> ":" <NSS>
```

其中 <URN> 只是一个标识名称为 URN 的标签，"urn:" 是用来确定如何处理 URN 的名字空间标识符，<NID> 是一个名字空间标示符，它用来指定这个 URN 方案的权威，<NSS> 是特定名称的字符串，其句法和意义在 <NID> 的上下文中定义。换句话说，<NSS> 的含义由拥有特定 URN 名字空间的 <NID> 分配和决定。

图 6-24 给出了一些 URN 例子。"path"、"www-cs-nctu-edu-tw" 和 "isbn" 是名字空间标识符。第一个例子由命名权威或路径 "/home/ydlin/courses" 以及一个唯一的字符串 "index.html" 组成。第二个例子说明了一个在域 www-cs-nctu-edu-tw 上的学生。最后一个例子是一本书的 URN。URN 使用这本书的 ISBN 号作为名字。如果一个服务想要使用 URL 指向该书，它可能看起来像

```
urn:path/home/ydlin/courses/index.html
urn:www-cs-nctu-edu-tw:student
urn:isbn:0-201-56317-7
```

图 6-24 一些 URN 例子

```
http://www.isbn.com/0-201-56317-7
```

其中包含一个特定的协议和一个可能随着时间而更改的域名。URN 不包含这些可变内容，所以它比较稳定。但是，如果有一个能够将名字映射到对应资源的系统就会更有用。这种过程称为解析。类似于 DNS 将域名解析为 IP 地址的方式。虽然 URN 可以解析为任何网络资源或服务，但 RFC 1737 主要将 URN 解析为 URL。

6.4.3 HTML 和 XML

超文本标记语言（HTML）是 Web 网页设计的最主要的标记语言。从作为万维网联盟（W3C）指定的标准通用标记语言（SGML）演变而来，HTML 通过把文本表示为链接、标题、段落、列表等提供了一种基于文本的文档信息结构描述手段，为文本补充了交互式表格、嵌入式图像和其他对象。HTML 是以尖括号括起来的"标签"形式书写的。

然而，纯格式已认为不足以帮助读者理解信息。我们希望，人们可以在标记语言中定义自己的标签来描述数据，而不是简单地格式化它们。因此，扩展标记语言（XML）应运而生。XML 于 2007 年定义在 RFC 4826 文档中。它允许用户定义标记元素，并有助于信息系统共享结构化数据。与 HTML 仅支持有限的格式不同，XML 提供了一种称为可扩展格式语言（XSL）的标准格式规格。与只能接受具有少数层次的 HTML 相比，它能够允许任意层次的嵌套结构。它支持标准化解析的正式语法，并使解析更容易。除了类似 HTML 的简单链接外，XML 是可扩展的链接，其中目标包括多个相同或不同类型的对象资源。这使内容提供灵活，并可以在 XML 链接语言（XLink）和 XML 指针语言（XPointer）上实现。

6.4.4 HTTP

HTTP 消息包括客户端和服务器之间的请求和响应。请求消息包括：1）请求行，其中包括应用于资源的方法、资源标识符和使用的协议版本；2）头部，它定义了请求或提供的各种数据对象；3）空行，它用来将头部和消息正文分隔开；4）一个可选的消息正文。

表 6-16 列出了请求信息中使用的请求方法。它们中的许多值得进一步说明。CONNECT 用来动态地从一条连接切换到一条隧道，如 SSL 加密隧道，以确保通信安全。GET 是检索指定资源最广泛使用的方法。HEAD 是 GET 的伪码版本，经常用来测试超文本链接的有效性、可访问性、最近修改。在没有启动资源行动或资源检索的情况下，客户端可以使用 OPTIONS 请求为特定的 URL 提供通信选项信息。

POST 提交数据作为指定资源的新从属分支。PUT 请求数据准确地在指定资源存储。如果指定资源已经存在，那么数据应该视为位于原始服务器上的数据修改版本。虽然 POST 和 PUT 很相似，但还有一定的差异。PUT 是一种有限制的操作，只不过是把数据放在一个指定的 URL 上。但是，检查服务器

的资源，POST 允许服务器对数据做任何想要的操作，包括将它存储在指定的页面、替换页面、数据库或者直接丢弃。方法 TRACE 用来调用请求的远程应用层回送地址。TRACE 允许客户端查看在请求的接收端收到的内容，并使用这些数据作为调试或诊断信息。

表 6-16 HTTP 协议的请求方法

请求方法	描述	请求方法	描述
CONNECT	动态地将请求连接到任何协议，如 SSL 隧道	OPTIONS	请求有关可选项的信息并与指定 URL 相关联的状态
DELETE	如果可能，删除服务器上指定的资源	POST	将数据发送到指定资源的分支提交
GET	请求指定资源的表现	PUT	表示数据存储在指定的资源上
HEAD	要求作为 GET 的响应，但是没有实际内容	TRACE	沿着请求的远程应用层回送地址

接收并解释请求消息后，服务器就用 HTTP 响应消息作为响应。响应消息的第一行包含协议的版本，后跟数字状态代码和与它相关的文字短语。表 6-17 描述了响应状态代码。状态代码是 3 位整数代码，用来报告满足请求的尝试结果。2xx 状态代码表示请求已被成功地处理。

表 6-17 HTTP 协议的响应状态代码

响应状态代码	描述
1xx	通知性的——请求收到，继续处理
2xx	成功——行动已经成功地被接收、理解和接受
3xx	重定向——必须采取进一步的行动，以完成请求
4xx	客户端错误——请求包含语法错误或无法完成
5xx	服务器错误——服务器无法完成一个明显有效的请求

图 6-25 给出了一个 HTTP 会话的例子，其中客户端下载了一些图像文件，然后将多个文件上传到远程服务器上。

```
C: GET / HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /images/doclist/icon_5_spread.gif HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /images/doclist/icon_5_chrome_folder.gif HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /doclist/client/js/3852707503f-cocliet_modularized_webkit_app__zh_tw.js HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: POST /r HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /DocAction?action=upload&hl=zh_TW HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /doclist/client/js/2829347588-doclist_upload__zh_tw.js HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET /images/doclist/icon_5_folder.gif HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: POST /upload/resumableupload HTTP/1.1\r\n
S: HTTP/1.1 201 Created\r\n
C: POST
/upload/resumableupload/AEnB2Upo0ch4TITW3KbikSuyKdlpIEcH-mVAd2cv1dSFD1jSIQd1eNd1cZ
bVbOsKEcVO1VeR9MP_gkeuCIwU2ir003vUHUYvsQ0 HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: GET / HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
C: POST /r HTTP/1.1\r\n
S: HTTP/1.1 200 OK\r\n
```

图 6-25 一个 HTTP 会话的例子

从无状态到有状态

HTTP基本上是一个无状态协议，也就是说，在与客户端交易期间服务器不会保存任何状态。服务器获取客户端请求的页面，完成交易，因此每一个交易都是独立的。然而，如果能有客户端的帮助，那么HTTP服务器就可以以好像是有状态地工作。

有两种方法可以为需要有状态的应用实现有状态的HTTP交易。首先是使用会话的概念，其中在客户没有察觉的情况下将所有与潜在会话有关的参数保存在服务器上。然而，由于服务器的内存空间有限，所以这种方法缺乏可扩展性，导致会话状态很快过期。为了修复这个缺点，利用相对较小的cookies作为替代，其中将状态通过HTTP头部发送到客户端，然后以cookie的形式存储。客户端在随后的同一会话的HTTP请求中嵌入cookie。虽然可扩展性得到了极大的扩展，但这仍然需要客户端的合作，通常是通过用户手动设置来启用cookies，并且会给客户端带来安全隐患。

除交易级的有状态外，HTTP1.1聚特提供了额外的连接级的有状态。也就是说，只要有一条TCP连接就足以为客户端与服务器进行所有交易，这需要使用可配置的超时定时器。与普通的HTTP1.0的每个交易都建立一条连接相比，这显然节省了许多时间和内存空间。

行动原则：通过端口80或HTTP的非WWW流量

通常互联网应用与一个众所周知的服务器端口号相关联。例如，端口53用于DNS服务，端口20和21用于FTP服务，端口25、110和143分别用于SMTP、POP3和IMAP4服务，端口80用于HTTP服务。如今，网络传输更加复杂的流量，如使用动态分配的端口号的P2P流量。然而，在这些非知名端口上的流量经常由于各种原因而被企业防火墙阻塞。因此，许多这样的应用利用TCP端口80或HTTP消息来操作自己的流量以便能够通过防火墙。例如，Skype可以配置成运行在端口80上。Windows Live Messenger使用微软通知协议（MSNP）通过TCP端口1863发送消息，但作为可选项，它能够将MSNP消息封装在HTTP消息中。

通过端口80的传输与通过HTTP的传输是不同的。通过构建与端口80的连接级容易实现前者，而后者将原始流量封装在HTTP消息中。在任一情况下，我们的目标是通过端口80或HTTP消息旁路流量以躲避防火墙。因此，防火墙或网络管理员无法根据端口号或HTTP消息标识消息的类型，因为即使被识别为Web流量的流量也可能是其他类型。

历史演变：谷歌应用程序

在云计算时代，软件将作为服务组织客户端而不是出售和被拥有。虽然谷歌以提供互联网搜索服务而闻名于世，但它已经发布了多个基于Web的产品，包括Gmail、谷歌地图、谷歌日历、谷歌聊天、谷歌文档、谷歌网站、谷歌笔记本和谷歌Chrome，以及Picasaweb/Picasa。谷歌利用复制服务器、数据备份和云计算等技术在服务器之间分散工作负载以提高整体性能。最初，谷歌应用仅支持在线版本，其中所有的操作都转化成串行命令，通过互联网传输并由谷歌服务器来完成，但目前它们还支持用户本地操作的脱机版本，并且当连接到谷歌的服务器时就能传输最后的结果。

表6-18概括了不同谷歌应用的特点。谷歌文档类似于（Microsoft）Office，是一种基于Web的在线应用程序套件，支持更多在线协同工作。谷歌笔记本是一种基于Web的在线笔记本。谷歌Chrome是一种使用WebKit布局引擎和V8 JavaScript引擎的Web浏览器。谷歌地球是一种显示详细的卫星地图，甚至街道概览的地理信息系统。谷歌会话是一种基于Web的服务，旨在为个人和协作通信将电子邮件、即时通信和社交网络集成起来。参与者可以发送、回复和编辑名为会话的消息文件，添加参与者，通知变改，当有其他合作者输入时实时地做出应答。

表6-18 谷歌应用分类

种　类	应用程序的名字	评　论
办公程序	谷歌文档	·支持文档、表单和表示的文本编辑 ·协作编辑文档 ·支持在线使用

（续）

种　类	应用程序的名字	评　论
Web	谷歌网站、 谷歌笔记本	·Web 站点内容的协作编辑 ·支持在线使用
图片编辑	Picasaweb、 Picasa	·组织/编辑数字照片 ·支持在线/离线使用
即时通信/聊天	谷歌聊天	·使用 XMPP/Jingle 协议 ·支持在线使用
Web 浏览器	谷歌 Chrome	·Webkit 布局引擎 ·V8 Javascript 引擎 ·支持在线/离线使用
时间管理	谷歌日历	·Agenda 日程管理、安排、共享在线日历以及移动日历同步 ·支持在线/离线使用
地图	谷歌地图 谷歌地球	·在线影射服务 ·支持在线使用
通信/协作	谷歌会话	设计用于集成电子邮件、即时通信、Wiki 和社交网络服务
基于邮件	Gmail	·基于 Web 的接口 ·支持 POP3、IMAP4 和 SMTP ·支持在线/离线使用

6.4.5 Web 缓存和代理

Web 缓存是万维网上的一种加快文件下载的机制。就像计算机系统中的普通缓存概念一样，将以前用户检索过的远程内容副本保存在本地的缓存服务器上以便将来的访问，目的是提高带宽效率，而且最重要的是，上网体验更为快捷。这对于频繁访问的网页特别有帮助。

一旦收到请求，缓存服务器就检查是否有一个有效的副本可供使用。如果有（高速缓存命中），那么服务器将立刻将缓存的页面返回给客户端；否则客户端（浏览器）将收到消息页面未找到。客户端浏览器通过直接发送请求给 Web 服务器继续检索的查询，并且绕过缓存服务器。为了达到 Web 缓存的最大满意度，某些方面需要加以考虑。

缓存的对象　虽然近几年磁盘制造技术已经有了较大的进步，但大小限制仍然需要用户不要滥用磁盘配额。这同样适用于主要依靠磁盘存储的高速缓存机制。因此，为了确定缓存的目标，筛选过程是必要的，这往往意味着频繁地抓取静态网页，而不是基于 CGI/PHP/ASP 的动态内容。

内容置换　为了进一步处理可能的磁盘存储短缺，通常会采用删除以及阈值等置换技术。前者，在有限的存储中使用，简单地删除旧的网页以便能腾出空间，当然应用选择步骤可能要基于网页受欢迎的程度和新鲜度。在相对宽松的存储需求中，针对内容设置阈值，超过阈值就执行内容置换。

高速缓存一致性　除了找出并移除旧内容的普通置换外，给每个缓存项设置过期时间以防止它的内容过时。过期时间可以从最后一次请求文件或从上一次验证日期中计算出来，这认为是更合适的。但是对于后者的权衡是，后者会增加计算和通信开销，尤其是在高峰时间。

透明代理

缓存服务器也可以作为代理服务器，这有助于在出现缓存未命中时将查询转发到正确的目的地。转发目的地可能是另一台缓存服务器或相应的 Web 服务器。此功能在两个方面是有益的。第一，重新从客户端向 Web 服务器发送请求的开销去掉了。第二，也是最重要的，通过将所有的访问集中在代理服务器并对它们进行监控可以最大限度地控制网络。

通常情况下，Web 缓存功能要求客户端预先设置浏览器，以确保它首先检查缓存服务器。换句话

记，浏览器必须知道缓存服务器的地址。然而，复杂的手动配置通常会阻碍用户激活 Web 缓存。幸运的是，这些工作可以通过透明代理处理，其中包括一个称为端口重定向的网关级技术。以同时支持缓存和代理的流行的开源软件包 Squid 作为例子，网络的网关服务器收集所有发向端口 80 的 Web 访问并将其重定向到（例如，在 Linux 中使用 iptables）Squid 服务器上，它一般集成在相同的网关中。这样，服务器对于一般用户几乎是透明的，因此就不必要进行手动配置。图 6-26 中的场景（1）描述了集成在网关中的透明代理的概念。

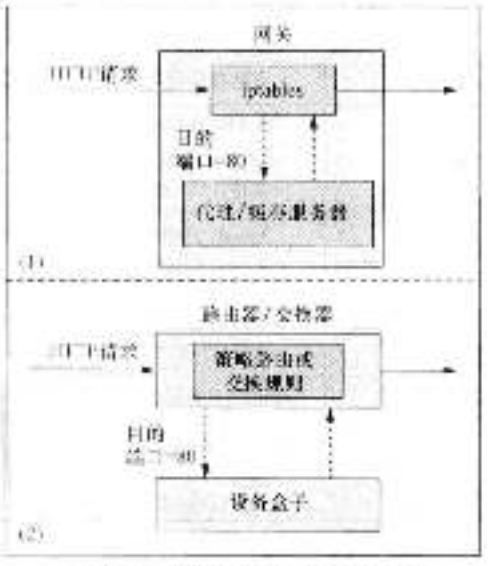


图 6-26 透明代理的两种配置场景

然而，并不是所有的系统管理员喜欢集成的代理/缓存服务器部署，这既可能是由于性能上的考虑也可能是由于在网络拓扑中没有网关的缘故。在这种情况下，通过策略路由，一个独立的服务器盒子可以和一个单独的路由器一同应用，或者通过基于目的端口号的交换规则与第 4 层交换一起应用，如图 6-26 中的场景（2）所示。

开源实现 6.3：Apache

概述

毋庸置疑，当谈到开源 Web 服务器时，Apache 能够作为当代的实出代表。由于具有支撑的功能，如带有数据库的动态页面（如 PHP + MYSQL 或者 mod_cool 模块）、SSL 支持、IPv6 支持、XML 支持、可扩展成多线程体系结构，2010 年 Apache 以 47% 的市场份额继续统治着 Web 服务器市场。

随着各种 Web 相关服务需求的不断增长，Apache Web 服务器也已经成为开源社区中最复杂的服务器之一。但是，由于它的模块化设计，还是有可能理解 Apache 程序的内部设计。一般来讲，Apache 是一种面向连接的无状态 HTTP 协议绑定到端口 80 的并发预派生（preforked）实现。通过将 cookies 嵌入到 HTTP 消息，Apache 还支持长期的状态。

框图

Apache 服务器程序的子类组件，本质上是层次化的，可以分为三个部分：1）服务器进程初始化；2）主服务器；3）取决于实现的工作者进程或工作者线程，如图 6-27 所示。我们将根据图 6-29 中的处理流程来描述它们。接下来，我们先复习在设计该软件时具有重要意义的“池”概念。

数据结构

与普通的池概念所为一组处理实过程一样，在 Apache 内存资源中它也是作为从来操作的，每个池都有源的空表示被处理的基本数据来管理。然而，在池中分配段时，有的要在合适的时间清理以防



图 6-27 Apache 的内部结构

止程序态讯释放内存。为了确保块的正确释放，发行许多内置的带有不同生存时间的池，如图 6-28 所示。池还包括子池的健表。而池 pglobal 在整个服务器的运行时间中存在，pconf、plog 和 ptrans 从仅在服务器更新启动时才会存在。类似的生存时间规则适用于 pchild（子过程工作者进程/线程 process/thread）、pconn（连接）和 preq（请求）。可以通过图 6-29 中 apr_pool_create() 创建池，这里将它指定为 newpool 的父池。将父池 pglobal 在服务器启动时自动创建以便于子池在需要时可以随时启动（即当新的连接建立时，新的请求到达时等）。



图 6-28 Apache 中池的层次

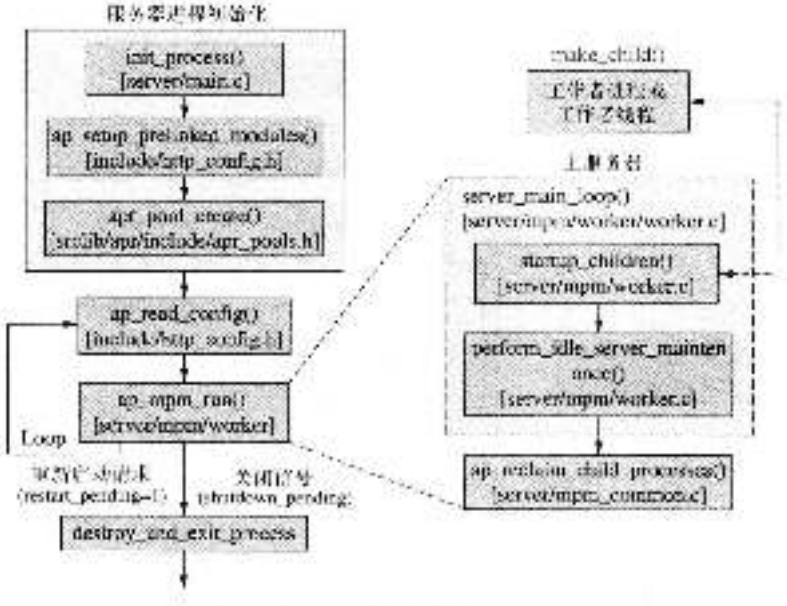


图 6-29 Apache Web 服务器的内部

算法的实现

现在，让我们讨论图 6-29 中 Apache 的处理流程。服务器的启动是通过 init_process() 完成的，初次使用时它创建一个进程池。然后 ap_setup_prelinked_modules() 初始化初始操作中所涉及的模块

一旦 apr_pool_create() 创建了上述提到的各种资源池后，服务器进程的初始化就完成了。接下来 ap_read_config 处理命令行传递的指令，并读取在相关子目录中阅读相关配置文件。

真实现多处理模块（MPM）中的一个步骤是注册就是调用 ap_mpm_run() 启动一个进程作为主服务器。支持两种类型的 MPM：预派生（prefork）和工作者（worker）。预派生MPM实现一种非线程的，预派生的 Web 服务器，其中请求由预先派生数量的进程以使响应式加服务到达的请求。它也是激活每个请求的最好 MPM，这样单个请求的问题不会影响任何其他的请求。但是，这种 MPM 缺乏可扩展性并且最好用在不能很好支持线程库的老操作系统中。现代操作系统，如 Linux 和 FreeBSD 都很好地配置了线程库，因此不存在这个问题。

为了弥补预派生 MPM 的缺点，工作者 MPM 实现一种混合的多线程、多进程的服务器。与预派生机制先类似，有多进程都是预派生的，但多个线程在每个进程中进一步被生成。与纯粹的基于进程的服务器相比，使用线程可以使更少的系统资源来服务大量的请求。然而，工作者 MPM 通过运行多个进程，每个进程带有很多线程，从而仍然保留基于进程的许多稳定性。现在，在下面的段落中我们将使用工作者 MPM，当然 server/mpm/prefork/ 目录下的类似程序也可以用于预派生 MPM。

在 ap_mpm_run() 内，通过重复 startup_children 中的 make_child() 函数调用 server_main_loop() 来产生预置数量的子服务器进程。根据选择的多处理策略，每个子服务器可能需要另一个初始化阶段以便访问正常操作所需要的资源，如为了连接到数据库。这可以通过调用 make_child() 中的 child_main() 来完成。如图 6-30 所示。调用 apr_run_child_init() 完启动环境设置，如子服务器的关键部分，然后通过 apr_thread_create() 创建预定数量的线程，随后调用 start_threads()。start_threads() 处理两种类型线程的创建：create_listener_thread() 函数创建监听线程以便监听新的连接请求；而 worker_thread() 创建工作者线程，它通过 process_socket() 和 ap_process_connection() 处理套接字。注意，不创建监听线程除非存在多个工作者线程。这种思想可以用一个简单的比喻来解释：一个餐厅在服务员/女服务员（监听线程）可以开始接受客户的订单之前，需要确定厨师（工作者线程）已经准备好。因此，当必要不可地在工作者线程访问资源，在需要时将还要补充线程池。

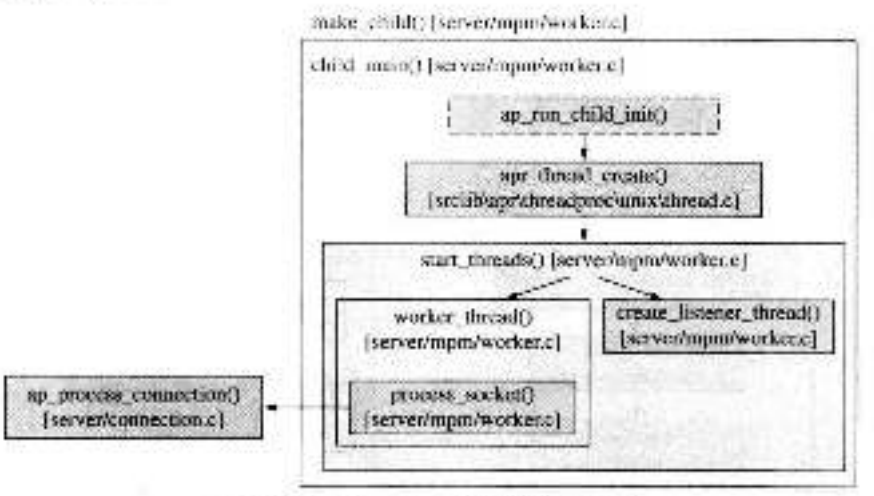


图 6-30 make_child() 中的内部情况

由于服务器忙于处理请求时，图 6-29 中所有 server_main_loop() 的主服务器在创建服务器后都进入 perform_idle_server_maintenance()，并找得死亡（SERVER_GRACEFUL 状态，这意味着正常关机）和已经死亡的（SERVER_DEAD 状态）子服务器。通过监控死亡和已死亡的服务器，Apache 就可以知道是否创建更多的服务器。最后，如果 ap_mpm_run() 前接了一个关机信号，那么主服务器就用 ap_reclaim_child_processes() 开始回收所有的子服务器。

练习

1）找到实现预派生的 .c 文件并浏览代码行。何时调用预派生？

2）找到实现 cookie 一致性的 .c 文件和代码行。

3）找到实现 HTTP 请求处理和响应准备的 .c 文件和代码行。

性能问题：Web 服务器的吞吐量和延迟

图 6-31 显示了 Apache 网络服务器上的 HTTP 请求处理的调用图。ap_run_create_connection() 为到达的请求分配并初始化数据结构，ap_read_request() 解析请求，然后 ap_process_request_internal() 检查授权。为了应答请求，ap_invoke_handler() 调用内容发生器冷备响应数据，check_pipeline_flush() 完成任何延迟的响应，ap_run_log_transaction() 将有关连接的数据记录在日志中。最后 ap_lingering_close() 关闭连接并清理数据结构。

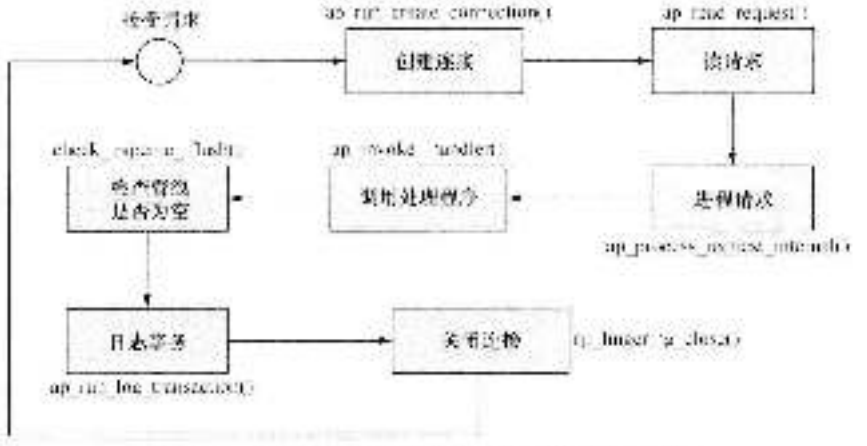


图 6-31　在 Apache Web 服务器中的 HTTP 请求处理

图 6-32 说明每个函数处理 HTTP 请求所花费的时间。最引人注目的现象是，花费在 ap_invoke_handler() 上的时间随着文件的大小增加而增加。在本实验中，将 HTTP 响应机器或静态的，即在磁盘上的网页响应，因此 ap_invoke_handler() 调用的内容发生器的任务是从磁盘上读取网页内容，然后将文件的内容传输给客户端。如果在传输之前需要将所有文件从磁盘读取到用户空间内存中，那么这将是一个非常复杂耗时的任务。

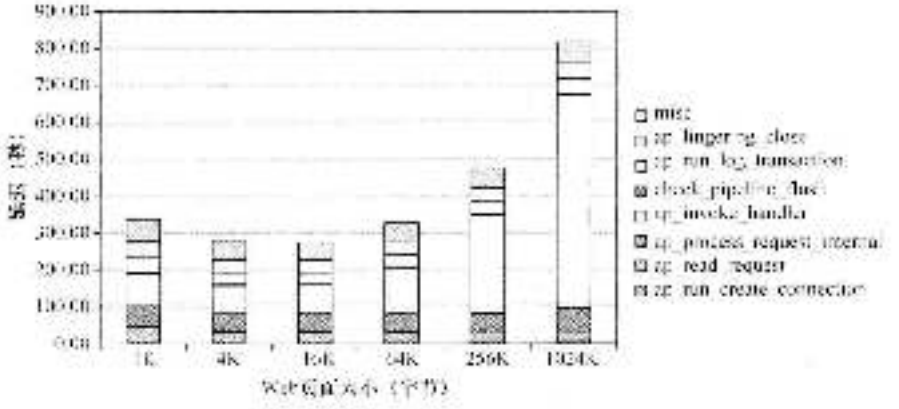


图 6-32　在 HTTP 请求处理中主函数的延迟

Linux 提供了 sendfile() 系统调用以便加快数据复制任务，ap_invoke_handler() 利用系统调用生成 HTTP 响应。sendfile() 的原型是 size_t sendfile (int out_fd, int in_fd, off_t * offset, size_t count)。通过这些 Linux 内核直接从一个文件描述符（如磁盘上的文件），复制到其他的文件描述符（如会接字），而无需用户空间和内核空间之间频繁的上下文切换。此功能称为零复制。每次调用它，sendfile() 都复制文件的一部分，其大小取决于文件系统的结构，因此在完成文件

复制之前需要多次调用sendfile()。表6-19列出调用sendfile()发送网页所需要的时间。当Web网页增加时，sendfile()所占消耗的ap_invoke_handler()的执行时间会增长。只有35%的ap_invoke_handler()的执行时间花费在sendfile()发送1 KB网页返回到客户端上，而当网页的大小是1024 KB时它就变为87%。

表6-19 sendfile()与ap_invoke_handler()的比例

文件大小	1 KB	4 KB	16 KB	64 KB	256 KB	1024 KB
调用sendfile()的次数	1	1	1	2	7	15
sendfile()的总执行时间（μs）	37	37	42	74	215	527
sendfile()与ap_invoke_handler()的时间比例	35%	38%	49%	53%	77%	87%

6.5 文件传输协议

作为最早的互联网应用之一，文件传输协议（FTP）并不是像看起来那么简单。事实上，它具有一种独特的执行带外控制命令的双连接操作模型，它将命令/应答和用户数据分别在单独的控制和数据连接上传输。大多数其他的应用程序利用内信令运行，控制和数据会经过相同的连接。大概与此唯一类似的就是P2P，它经常发送大量的UDP分段作为查询/响应并为实时数据传输建立TCP连接。本节说明这种精巧的双连接操作模型以及FTP服务器如何从主动模式更改为被动模式以便能够与防火墙或NAT的后面设备也能建立连接。还会介绍FTP协议信息。我们选择KD-ftp作为开源实现的例子。

6.5.1 简介

几十年前，人们编写程序并将它们保存在磁带或磁盘上。为了在远程机器上运行程序，所有的磁带和磁盘必须被装运并加载到那台机器上。这通常是不方便的和耗费时间的。为了解决这种在磁带和磁盘上传输文件的低效率，文件传输协议（FTP）允许用户高效、可靠地通过互联网从一台主机向另一台主机传输文件。FTP的另一个优点是数据复制，这能够实现较大规模的数据备份。1971年FTP在RFC 172中首次提出，后来由于出现了RFC 265，354，542，765而被废弃了，1985年在RFC 959中被标准化。2007年RFC 3659是对FTP扩展的最新更新。

像其他许多网络应用一样，FTP运行在客户端/服务器模型上并在TCP上运行。因此保证了可靠的点到点连接。FTP提供了两种类型访问：认证和匿名。前者为了用户认证需要账号/密码对；而后者通常是无限制的，虽然出于管理的考虑可能禁止了一些IP地址。匿名用户所要做的就是以“anonymous”（匿名）或“ftp”登录，并输入用户的电子邮件作为密码。在许多情况下不进行严格的检查。

例如，如果你想要通过FTP从为一所大学下载文件，那么你就需要先登录到本地计算机中。除非你使用匿名FTP，否则你还需要一个登录名和密码才能访问远程FTP服务器上你的账户运行文件下载。FTP会话包括5个主要步骤：

1）连接或登录到下载（或上传）目标文件所在的计算机。

2）调用FTP客户端程序。

3）连接到文件下载（或上传）的远程FTP服务器。

4）提供登录到远程服务器上的用户名和密码。

5）向FTP服务器发出一系列命令查看和传输目标文件。

FTP客户端应用程序可以运行在类UNIX或Windows系统上。FTP服务器网站一般支持基本的命令，如表6-20所示。

当然也可以使用Web浏览器发起FTP会话。例如，在匿名模式，如果你在浏览器中的URL字段键入

```
ftp://ftp.ca.acts.edu.tw
```

表 6-20 FTP 用户命令

命 令	意 义	命 令	描 述
OPEN	连接到远程主机	RENAME	命名远程主机上的文件
CAT	在远程主机上的文件	BYE	撤销远程主机上的文件
GET	取回远程主机上的文件	QUIT	终止 FTP 会话

如果该网址允许匿名登录，浏览器就会自动地注册以便允许用户登录到 FTP 网址。在认证模式，在 URL 字段中的登录格式为

ftp://username@ftp.cie.ncku.edu.tw

这就允许用户以“username”登录 ftp.cie.ncku.edu.tw，然后就会出现用于输入密码的输入窗口。

6.5.2 双连接操作模型：带外信令

FTP 客户端和服务器之间的通信采用两个单独的连接，服务器的控制进程监听 TCP 端口 21，服务器的数据进程监听 TCP 端口 20。匿名发送、控制连接处理命令、参数、应答、错误恢复等标记的交换，而数据连接专门用于文件的传输。前者在整个 FTP 会话期间都存在，而后者则根据需要创建和删除。与大多数其他应用程序将控制和数据消息混合在一起并运行同一连接传输（即带内信令方式）不同，这种双连接机制通常称为带外信令。如图 6-33 所示，FTP 会话过程详述如下。

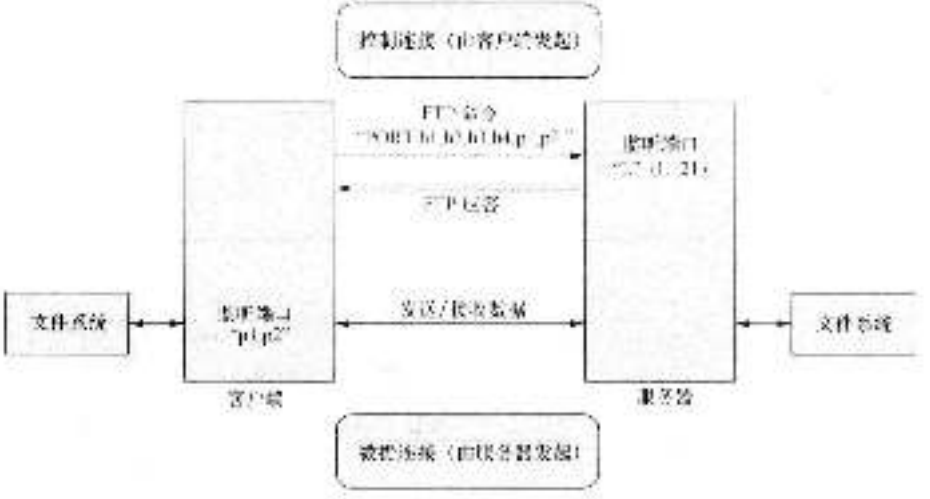


图 6-33 FTP 的基本操作模型

在控制连接建立并完成认证过程后，客户端向服务器发出一个 FTP 指令，PORT h1，h2，h3，h4，p1，p2，“你能不能给我建立一个到 IP 地址 h1.h2.h3.h4 上的端口号 p1p2 的数据连接？”并监听该地址的指定端口。需要注意的是，h1～h4 和 p1、p2 是十六进制。然后服务器用适当的状态代码回答，以确认客户端。下一步，客户端就可以发出命令运行监听、下载、添加或给服务器的文件系统上传文件。服务器将启动一个用于文件传输的数据连接。请注意，因为这些命令中的每一个都包括一个独立的数据连接，所以 PORT 命令应该总是在它们之前发布。所有操作都完成后，客户端通过控制连接向服务器发送“QUIT”以便终止 FTP 会话。

有时发出 FTP 命令的主机并不一定是客户端或服务器，也就是说，它可能只是客户端和服务器之间的代理，通过 FTP 命令安排它们之间的数据连接。例如，可以使用该模型，在文件服务器监听中央控制器指定端口的互备份系统中，等待数据传输的命令。

历史演变：为什么在 FTP 中使用带外信令

由于 FTP 是互联网历史上最古老的应用，telnet 仅仅提前两年的几天问世，所以当时 FTP 为什么采用带外信令的确切原因现在无从得知，但是对此有一个猜测，涉及当时有关的文件完整性的。

文件传输服务的最初设计是使用数据传输协议（DTP）作为数据平面协议，而 FTP 是仅负责控制连接。IP 创建后，DTP 被 TCP 取代。不是将控制和数据连接合并成一个 TCP 连接，而是 FTP 继续使用双连接机制最大限度地减少对已有实现的影响。

令人惊讶的是，这种带外信令连接高了 FTP 的性能。它避免了在单连接情况下区分控制和数据分段的额外工作。也就是说，通过专用的数据连接传输文件，避免了处理头部或控制信息的额外开销。另一个优点是，在数据连接上长时间文件传输期间，控制连接仍然可以用于目录查找或者经过另外一个数据连接发起了另一次文件传输。此外，两种连接模型，还能使用一个如上所述的中间控制主机。

主动模式与被动模式

在前面的模型中，控制连接是由客户端发起的，而数据连接是由服务器发起的。从服务器的角度，这种发起称为主动模式。然而，还有一种方案称为被动模式，其中两个连接都是由客户端发起的。

如图 6-34 所示，当处于主动模式的服务器收到 FTP 请求时，将连接到客户端。然而，如果客户端位于 NAT 或防火墙之后，那么来自服务器的数据连接就可能被阻塞。当检测到这种阻塞问题时，无论用户手动还是客户端应用程序自动进行，客户端都会通过发出 PASV 命令再次请求服务器的被动 FTP 模式，它请求服务器监听特定端口上的数据连接。如果请求得到服务器许可，那么服务器就通过发出带有 IP 地址和端口号的 PORT 确认客户端，其中端口号不是 20 而是目前正在监听的端口号。现在，双方都进入被动模式。然后客户端初始化服务器的数据连接，并开始文件传输。

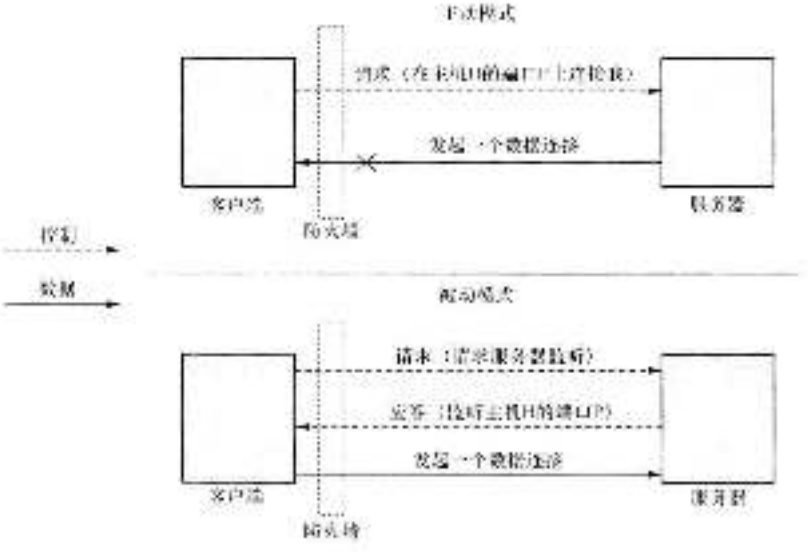


图 6-34 主动模式与被动模式

6.5.3 FTP 协议消息

表 6-21 列出了主要的 FTP 命令。注意，这里的命令与表 6-20 中为最终用户站点所支持的命令不同，这些是在 RFC 中定义的 FTP 协议消息。服务器将用户命令映射为执行实际操作的一个或多个 FTP 命令。例如，当我们输入用户命令

```
rename path_of_source_file path_of_dest_file,
```

服务器将它映射为以下两个操作：

```
ENTR path of source file     (ReNameFRom)
ENTC path of dest file       (ReNameTO)
```

来完成文件的重命名。

FTP 服务器总是发送一个答复，以便确认客户端先前发出命令的执行状态。应答代码有 5 个功能组，如表 6-22 所示，利用第二个数字指示句法错误、控制状态和数据连接等，利用第三个数字表示在第 2 个数字范围内状态的细致度等级。

表 6-21 主要的 FTP 命令

命　令	描　述	类　型
USER	发送用户名字	访问控制
PASS	发送口令	访问控制
PORT	发送客户端的 IP 和端口间就是查询的（服务器）	传输参数
PASV	通知服务器表明数据端口正不是显示数据产生	传输参数
RETR	请求服务器将请求文件的副本传送给客户机	文件服务
STOR	使服务器接收和应用数据并将它存储为文件	文件服务
RNFR	指定重命名单文件的路径	文件服务
RNTO	指定重命名目的地文件的路径	文件服务
ABOR	通知服务器终止以前的命令和相应的就是传输	文件服务

表 6-22 FTP 应答的 5 种类型

应　答	描　述	类　型
1yx	发起的请求行动，在进行就命令之前期待另一个应答	肯定的初步答复
2yx	请求的行动已经成功完成	肯定的完成答复
3yx	命令已经被接收，但是请求的行动 being held，等待来自另一命令的进一步消息	肯定的中间答复
4yx	命令没有被接收，请求的行动没有发生，行动可能被再次请求	暂时否定完成答复
5yx	除了错误状态是永久的外，与 4yx 相类似，因此行动不能被再次请求	永久否定完成应答

图 6-35 是一个 FTP 会话的例子。我们以用户“www”登录并检索一个名为“test”的文件。客户端要求服务器与它连接到 140.113.189.29 的两个不同端口，即 4135（十六进制为 1027）和 4145（十六进制为 1031），分别用于检索目录清单和文件“test”。

```
STATUS:> Connecting to www.cis.nctu.edu.tw (ip = 140.113.166.122)
STATUS:> Socket connected. Waiting for welcome message... 220
         www.cis.nctu.edu.tw FTP server [Version wu-2.6.0(1) Mon Feb 28 10:30:30 EST
         2000] ready.
COMMANDS:>          USER www
         331 Password required for www.
COMMANDS:>          PASS ********
         230 User www logged in.
COMMANDS:>          TYPE I
         200 Type set to I.
COMMANDS:>          REST 100
         350 Restarting at 100. Send STORE or RETRIEVE to initiate transfer.
COMMANDS:>          REST 0
         350 Restarting at 0. Send STORE or RETRIEVE to initiate transfer.
COMMANDS:>          pwd
         257 "/home/www" is current directory.
COMMANDS:>          TYPE A
         200 Type set to A.
COMMANDS:>          PORT 140,113,189,29,10,27 ← 告诉服务器连接的哪里
         200 PORT command successful.
COMMANDS:>          LIST                          ← 查询 [illegible]
         150 Opening ASCII mode data connection for /bin/ls. ← 文件或是目录
         [illegible]
.....list of files....
COMMANDS:>          TYPE I
         200 Type set to I.
COMMANDS:>          PORT 140,113,189,29,10,31
         200 PORT command successful.
COMMANDS:>          RETRtest                      ← 在文件 "test"
         150 Opening BINARY mode data connection for test (5112 bytes).
```

图 6-35 一个 FTP 会话的例子

带有检查点的重新启动传输

到目前为止，我们已经介绍了FTP会话的初始化、命令和应答。FTP还实现了一种重新启动机制，用于在两台一个断开的路径和死机或其程等错误时的恢复。主要还是使用“标记”（marker），其中包括已经传输文件的位置。

在文件传输期间，发送者在数据流中方便的地方插入一个标记。一旦接收者收到标记后，就将以前收到的所有数据都写入数据。在本地文件系统中的相应位置做标记，并将发送者和接收者同步最新标记的位置应答给用户，即控制主机。它可与数可以不与发送者在同一台机器上。当服务发生故障时，用户就可以发出带有以前标记信息的重启命令重新启动在上次传输检查点的发送者。

开源实现6.4：wu-ftpd

概述

wu-ftpd是最流行的FTP守护程序之一。最早由华盛顿大学开发，它目前由WU-FTPD开发组（http://www.wu-ftpd.org/）维护。

除了前面所述的基本文件传输功能外，wu-ftpd还提供了有用的工具，如虚拟FTP服务器和即时（在需要时创建）压缩。这些工具没有在RFC中定义，但确实方便管理工作，提高了文件传输的效率。总之，wu-ftpd是一个绑定到端口20和21的面向连接的、有状态的FTP协议的并发实现。

算法实现

在wu-ftpd工作中有两个主要阶段：服务初始化发起阶段和命令读取/执行阶段。如图6-36所示，wu-ftpd实现了一种典型的派生子进程服务于客户端的并发服务器模型。

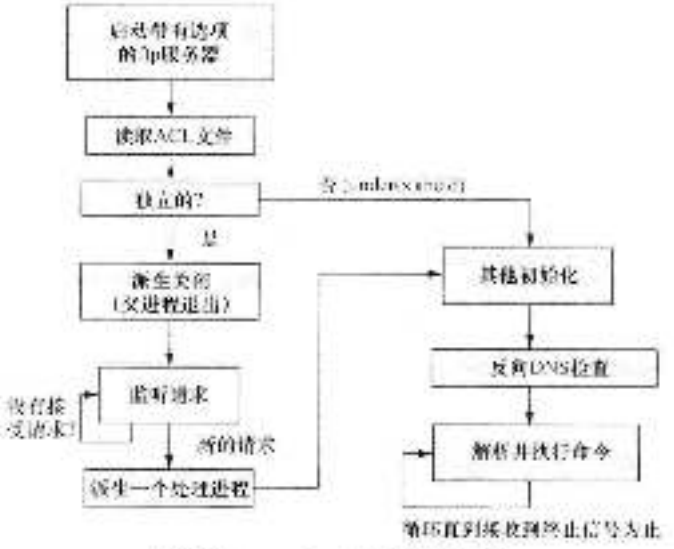


图6.36 wu-ftpd内部的执行流

在服务初始化阶段，我们首先执行“ftpd”命令，无论是从shell（命令行解释器）还是从（X）inetd都会带有一些选项来描述其行为以便启动服务器。例如，选项-l指定应用进程级别限制以避免系统受到的浪费；-t则设置超时口令。当服务器进程的所有者没有超级用户特权，它意味着用户只能够使用大于1024的端口号，而不是默认的20。然后服务器将ftpaccess文件中的访问控制列表读到内存中，通知服务器有关主访问功能的设置。

读取了主配置后，最初的服务器进程派生一个新的过程用于监听新请求的独立服务器，然后这出让新创建的服务器进程开始运行。一旦接受请求后，服务器派生一个处理进程处理FTP会话中的后续步骤。如果服务器不是作为一个独立的服务器运行，那么这就意味着服务器被（x）inetd调用。在服务初始化阶段的结果是其他初始化工作，用于反向DNS检查客户端，文件访问检查，以及虚拟主机分配以便将对不同目的地址名称的请求映射到相应的配置号。

在第二阶段，FTP命令解析和执行的主要任务是通过使用Yacc（Yet Another Compiler-Compiler，编译器代码生成器）完成的。Yacc用户指定FTP输入的结构，以及当结构被识别时要调用的代码段。Yacc利用FTP命令的结构输入，将这种规范转换成在编译时的处理输入的子进程。

虚拟FTP服务器

当在单台机器上服务多个域时，还需采用虚拟FTP服务器。它们允许管理员配置系统，这样一个用户连接到ftp.site1.com.tw而让另一个用户连接到ftp.site2.com.tw。每个用户得到自己的FTP广告和目录，即使它们在同一台机器的相同端口上。如图6-37所示，这可以通过使用名为“ftpaccess”的配置文件来实现。建立一台虚拟FTP服务器需要设置4个基本参数：服务器名（或IP），根目录、欢迎消息广告和传输日志记录。一旦收到一个请求，FTP守护进程就将请求中的目的站点名与ftpaccess中指定的规则进行匹配。匹配的请求被获取，然后就像普通FTP服务器一样进行处理。

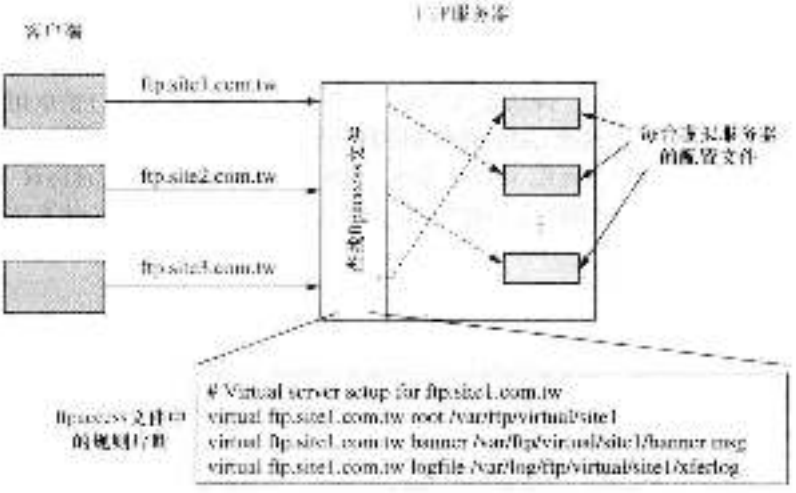


图6-37 虚拟FTP服务器的连接

即时压缩

因为FTP至少需要两个协议消息（PORT和RETR）才能下载文件，所以我们可以很容易地想象当下载大量小文件时它将如何影响网络——将会有大量的用于连接建立和拆除的消息。为了弥补这个缺点，wu-ftpd提供了另一种优秀的工具，称为“即时压缩”，即服务器在发送给用户之前压缩文件（目录）。图6-38是一个例子。

```
Uservnlin logged in.
Logged in to wwwpc.cis.nctu.edu.tw.
ncftp /home/ynlin > ls
Ltar.gz          Desktop/          ucd-snmp-4.2.1/
ncftp /home/ynlin > get ucd-snmp-4.2.Ltar.gz
ucd-snmp-4.2.ltar.gz                7393280 bytes 552.83 kB/s
ncftp /home/ynlin >lls -l
drwxr-xr-x 24 gis88559 gis88        3584 Oct 8 12:18 .
drwxr xr x 88 root gis88            2048 Sep 10 17:48 ..
-rw-r---- 1 gis88559 gis88          7393280 Oct 8 12:18 ucd-snmp-4.2.ltar.gz
```

图6-38 当有即时压缩的文件下载

就像我们在该例子中所见，当服务器上没有这样的“tar-ball”（压缩包）客户端会获取的文件“ucd-snmp-4.2.1.tar.gz”（压缩的tar归档文件）而只有一个目录ucd-snmp-4.2.1时，该程是服务器会提取文件名称的后缀，并根据名为“ftpconversions”的配置文件中指定的规则执行适当的行动。在这种情况下，被调用的动作就是对给定文件名执行“tar -zcf”命令。表6-23列出了wu-ftpd的一些重要配置文件。

表 6-23 wu-ftpd 的 4 个重要配置文件

文件名字	描述
ftpaccess	用于配置 ftp 守护程序的操作
ftpconversions	指定检索文件的后缀及其对应操作
ftphosts	用于拒绝/允许某些主机以某一账户登录
ftpservers	列出虚拟服务器和包含他们自己配置文件的对应目录

练习

1. FTP 会话的控制和数据连接的并发处理是如何以及在什么地方进行的？它们是由同一个进程还是由两个进程处理的？

2. 找到执行主动模式和被动模式的 x 文件和代码行。何时调用被动模式？

6.6 简单网络管理协议

在本章介绍的所有应用程序中，网络管理是唯一一个不是为普通用户设计的应用程序。事实上，它用于网络管理员进行远程网络管理。我们首先介绍网络管理的概念和框架。然后，我们介绍用于表示被管理设备状态的标准化管理信息库（MIB）和用于访问 MIB 的简单网络管理协议（SNMP）。通过一种称为 NET-SNMP 的开源实现能够让我们跟踪它的操作，以便更好地理解整个体系结构。

6.6.1 简介

自从互联网诞生以来，人们一直渴望能够监控和控制网络。为了实现这个目标，许多小工具已使用多年。例如，ping、traceroute 和 netstat（参见附录 D），其中前两个是基于 ICMP 的，而后者是通过系统调用的（如 ioctl）。即使它们能够满足由多台主机和网络设备组成的小规模的网络环境，但这些工具所提供的信息却不能满足大型网络的网络管理员的需求。他们要求的是一个更加通用的、系统的基础设施，以方便网络管理工作。

这就是简单网络管理协议（SNMP）的作用。该协议的思想是在所有被管理设备上安装代理程序，这样通过标准协议查询代理，管理器程序可以收集和更新设备的管理信息。管理信息是在标准化管理信息库（MIB）的管理对象中维护。这些就提供了多个优点。首先，标准化 MIB 和 SNMP 的使用能够支持多厂商管理器和设备之间的互操作性。其次，主要由于程序的移植造成的代理开发成本大大地减少了。同样，可以为网络管理员清晰地定义管理功能，管理器程序开发人员也是如此。因此用被管理设备的数量表示的体系结构更具可扩展性。

MIB 和其增强版本的 MIB-II 分别在 1988 年和 1990 年首次定义在 RFC 1066 和 RFC 1158 中。1989 年 SNMP 首次在 RFC 1098 中提出，称为 SNMPv1，它得到了积极的响应用于集成多类别的管理对象和多厂商产品之间的互操作性。1993 年提出了 SNMP 的第 2 个版本，称为 SNMPv2，在 RFC 1441 中提出以增强第 1 个版本的功能。最后，1998 年 SNMPv3 在 RFC 2261 中发布，解决了第 1 个版本中讨论过的某些安全附加功能。所有这 3 个版本的 SNMP 都具有相同的基本结构和组件。

网络管理的演变还要继续下去，并且要归功于由所有应用层协议所产生的最高百分比的 RFC。网络管理有许多其他的补充协议和 MIB 建议，如具有广泛流量测量的网络远程监控（RMON）MIB 于 1991 年在 RFC 1271 中定义，它的增强版本 RMON2 于 1997 年在 RFC 2021 中定义，最近提出的用于基于 IPv6 的 OSPFv3 的 MIB 于 2009 年在 RFC 5643 中定义。然而，它们超出了本书的范围，不在此讨论。

6.6.2 体系结构框架

SNMP 环境通常包含 5 个基本组件：管理工作站、代理、被管理对象、被管理设备和管理协议。这些组件之间的关系如图 6-39 所示。

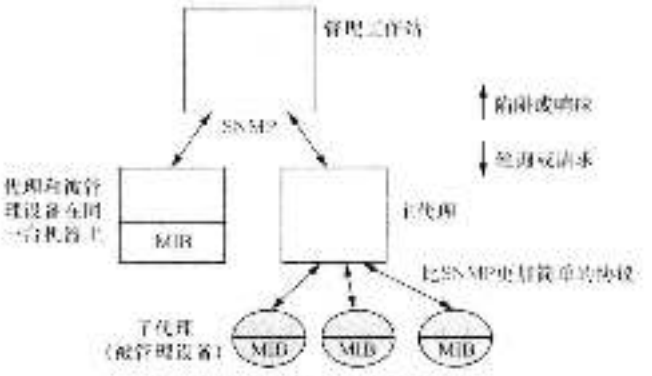


图 6-39 SNMP 的体系结构框架

管理工作站 又称为管理器，负责协调在其管辖范围内的所有代理。它定期检查每个代理的状态，并根据需要查询或设置被管理对象的值。

代理 作为运行在管理站点和被管理对象之间的被管理设备上的中间件，代理负责执行管理站点所要求的网络管理功能。

被管对象和 MIB 被管对象描述被管理设备某一方面的特点。例子包括系统上常运行时间、已收到的分组数量，以及在系统中活动 TCP 连接的数量。MIB 是形成虚拟信息存储的被管理对象的集合。

被管设备 它可能是一台路由器、交换机、主机或任何安装了代理和 MIB 的设备。

管理协议 它是用于管理工作站和代理之间传达信息的常见方法。

基于轮询和基于陷阱的检测

在运行 SNMP 的环境中有三种基本活动：获取、设置和陷阱。前两种用于管理站点获取/设置代理中对象的值；而最后一种由代理使用，用来通知某些事件的管理工作站。

由于 SNMP 是基于 UDP 的，所以这里没有持续的 TCP 连接，但有一些事务用于管理站点以便了解代理的健康状况。在代理的状态检测中，通常会看到两种方案：基于轮询的检测和基于陷阱的检测。在基于轮询检测中，管理站点周期性地向代理发送查询消息并接收代理状态的响应。尽管基于轮询的检测直观、简单，但当有大量的代理监控时，利用这种方案管理站点就会成为瓶颈。

基于陷阱的检测就是为了避免这种缺点。当在被管理对象上发生事件时在这种检测不是被动地询问，而是代理会主动地将获管理站点。事件驱动的陷阱能够减少我们在基于轮询的检测中看到的不必要的消息。在大多数情况下，当重启时，管理站点只检查代理以便有一张所有代理的基本图。

代理服务

除了普通管理工作站和代理之间的普通关系外，代理服务被认为是 SNMP 中另一个有用的操作方案。对于简单和廉价的设备，如调制解调器、集线器和网桥，只是为了与 SNMP 兼容而在它们上实现整个 TCP/IP 协议簇（包括 UDP）可能不切合实际。为了适应那些不支持 SNMP 的设备，提出一种代理服务的概念：一种是一个系统在另一个系统的前面响应协议请求的机制。前面的系统称为主代理，而后面的系统是子代理。如图 6-39 所示，主代理不用任何 MIB 实现，代表子代理处理从管理工作站来的 SNMP 请求。所有主代理的工作就是将 SNMP 请求转换为某些子代理可以理解的非 SNMP 消息。虽然子代理都应该是很简单的，但是一些协议（如可扩展代理（AgentX）和 SNMP 复用（SMUX））的开发，都是用于增强子代理。

6.6.3 管理信息库

MIB 可以看做一个树状的虚拟信息存储，尽管它并不用做数据库来存储信息。其实它只是一个列出被管理对象的规范，在 MIB 树中的每个对象唯一地由一个对象标识符（OID）来标识。例如，图 6-40 显示了可联网标准 MIB（MIB-II）的结构，IP 对象组由 OID 1.3.6.1.2.1.4 标识。有了相关的 OID，对象就有更好的可访问性。只有 MIB 树中的 leaf 对象是 OID 值可访问的。例如，System 组下的叶子节点

对象 sysUpTime。因此，访问 MIB 对象的一种典型情况可能会是这样的：

1）管理站点向它所查询的具有特定对象的 OID 的代理发送消息。

2）在接到请求后，代理首先检查对象是否存在，然后验证可访问性。如果行动失败，代理就用相应的错误消息响应管理站点；否则，它寻找本地系统以获得、寄存器或目录器中对象实例的对应值。

对象和对象实例

有人可能会对“对象”和“对象实例”的含义感到困惑。例如，人们认为他们想要得到管理信息的对象，而实际上，他们却得到对象实例。对象有两个属性：类型和实例。对象类型为我们提供语法和对象属性，而对象实例为一个给定到特定值的对象类型的特定实例。以对象 sysUpTime 为例加以说明。对象类型说明系统正常运行时间由用 TimeTicks 编码的，并且对于所有访问都是只读的。另一方面，对象实例告诉我们自从系统上次重启以来所逝去的时间。除了简单的对象外，有两种类型的复合对象：标量和表格。标量对象定义了一种单一的、组织化的对象实例，而表格对象定义了在一张表中成组的多个标量对象实例。为了区别表格对象下的标量对象的表述，普通的标量使用带有一个额外 0 的 OID 对象。

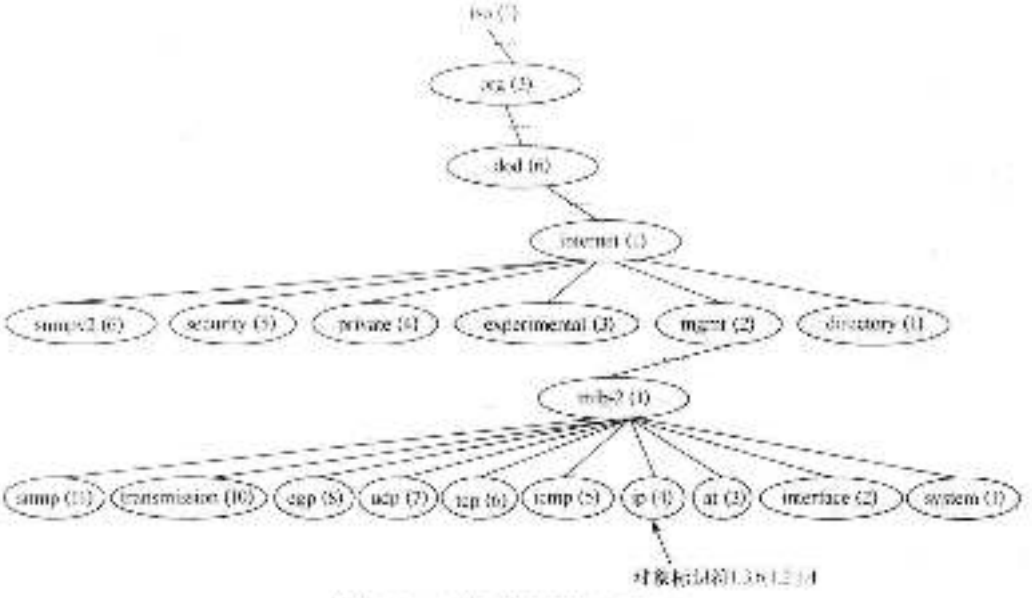


图 6-40 互联网标准 MIB：MIB-II

如今，几乎所有的 MIB 活动都发生在 ISO 分支的部分，并专用于对象标识符 1.3.6.1 的互联网社区。在 SNMP 中采用 MIB 也提供了可扩展性，这样人们就可以在实验的和私有专用的分支下建立自己的 MIB。前者是用于标识 IETF 工作组设计的对象，这些对象一旦成为标准，它们就被移到 mgmt (2) 子树下。在私有专用分支的正下方有一个称为企业的 MIB 子树，这是为网络设备供应商保留的。然而，为了保证互操作性，并避免来自不同供应商设备 OID 之间的冲突，建议一定要向互联网号码分配机构（IANA）注册 MIB 对象。

MIB-II 的主要贡献是对象组“mib-2”的定义。它更准确地描述基于 TCP/ IP 互联网的管理。我们在下面的列表中总结了 MIB-II 的每个对象组。

1）system 提供有关被管理系统的一般信息，例如，名称、正常运行时间和系统的位置。

2）interface 保存每个物理接口的配置信息和统计数据，例如，类型、物理地址和接口状态。

3）at 网络地址和物理地址之间的地址转换。然而，它在 RFC 中是不赞成使用的，并且只有网络地址可能与每个物理地址相关联。

4）ip 有关 IP 在本地系统上实现和运行的信息，例如，路由表和默认 TTL。

5）icmp　有关 ICMP 实现和操作的信息。例如，发送和接收的 ICMP 消息数量。

6）tcp　有关 TCP 实现和操作的信息。例如，系统中最大的活动连接数。

7）udp　有关 UDP 实现和操作的信息。例如，发送的数据报数量。

8）egp　有关 EGP 实现和操作的信息（外部网关协议）。

9）transmission　有关不同传输方案的相关信息和统计信息。

10）snmp　有关访问（获取、设置和陷阱）在 SNMP 的错误报告。

例子：在 MIB-II 中的 TCP 连接表

MIB-II 的 tcp 组下的 TCP 连接表如图 6-41 所示。它为我们提供了一个管理信息结构（SMI）（首次定义在 RFC 1442 中）如何用于实现 MIB 的很好例子。

在图 6-41 中的 TCP 连接表是一张二维表，其中每一行代表一条连接（TcpConnEntry）包含一条 TCP 连接以列的形式表示的 5 个属性：连接状态、本地/远程 IP 地址、本地/远程端口号。行中的每一列是一个标量元素，其属性字段定义在 SMI 中。通过使用两个抽象语法标记（ASN.1）："SEQUENCE OF" 和 "SEQUENCE" 来构造表。前者是将相同类型的一个或多个对象成组，在这种情况下是 TcpConnEntry，后者是可能有不同类型的标量元素成组，在这种情况下是 tcpConnState、tcpConnLocalAddress、tcpConnLocalPort、tcpConnRemAddress 和 tcpConnRemPort。此外，在一行内它包括 4 个元素，如图 6-41 中左边个别部分的 INDEX 语句所指示的，以标识一条连接。表 6-24 是一个 TCP 连接表的例子，它给我们一个明确的概念。

```
-- the TCP Connection table
--
-- The TCP connection table contains information about this
-- entity's existing TCP connections.
--
tcpConnTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "A table containing TCP connection-specific information."
    ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
    SYNTAX  TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
        "Information about a particular current TCP connection.  An
object of this type is transient, in that it ceases to exist when (or soon after)
the connection makes the transition to the CLOSED state."
    INDEX   { tcpConnLocalAddress,
              tcpConnLocalPort,
              tcpConnRemAddress,
              tcpConnRemPort }
    ::= { tcpConnTable 1 }

TcpConnEntry ::=
    SEQUENCE {
        tcpConnState INTEGER,
        tcpConnLocalAddress IpAddress,
        tcpConnLocalPort INTEGER (0..65535),
        tcpConnRemAddress IpAddress,
        tcpConnRemPort INTEGER (0..65535)
    }

tcpConnState OBJECT-TYPE
    SYNTAX  INTEGER {
        closed(1), listen(2), synSent(3), synReceived(4),
        established(5), finWait1(6), finWait2(7), closeWait(8),
        lastAck(9), closing(10), timeWait(11), deleteTCB(12)
    }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        "The state of this TCP connection."
    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local IP address for this TCP connection.  In the case of a
connection in the listen state which is willing to accept connections for any IP
interface associated with the node, the value 0.0.0.0 is used."
    ::= { tcpConnEntry 2 }

tcpConnLocalPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The local port number for this TCP connection."
    ::= { tcpConnEntry 3 }

tcpConnRemAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The remote IP address for this TCP connection."
    ::= { tcpConnEntry 4 }

tcpConnRemPort OBJECT-TYPE
    SYNTAX  INTEGER (0..65535)
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "The remote port number for this TCP connection."
    ::= { tcpConnEntry 5 }
```

图 6-41　在 MIB-II 规范中的 TCP 连接表

从表中我们可以看到，系统当前给出 4 条连接，每一条连接可以唯一地由本地/远程 IP 地址和本地/远程端口号（也称为 "套接字对"）（索引）来标识。请注意，表中的每个标量对象本身具有自己的 OID。这样可以修改其值。例如，在第 4 个条目中 "established state"（建立的状态）的 OID 试值为 "x.1.1.140.113.88.164.23.140.113.88.174.3082"。其后缀段将按照所述的连接来定。当实态变更时其值也相应地修改。

表 6-24 在 Tabular 视图中的 TCP 连接表

tcpConnTable (1.3.6.1.2.1.6.13) tcpConnEntry = (x.1)					
	tcpConnState (x.1.1)	tcpConnLocalAddress (x.1.2)	tcpConnLocalPort (x.1.3)	tcpConnRemAddress (x.1.4)	tcpConnRemPort (x.1.5)
x.1	最好	0.0.0.0	23	0.0.0.0	0
x.1	最好	0.0.0.0	161	0.0.0.0	0
x.1	关闭等待	127.0.0.1	161	127.0.0.1	1029
x.1	已建立	140.113.88.164	23	140.113.88.174	3082

索引

6.6.4 SNMP 中的基本操作

我们曾经提到过，在 SNMP 中有 3 种活动类型：获取、设置和陷阱。事实上，我们可以进一步指定如表 6-25 所示的操作，其中每个操作都封装在协议数据单元（PDU）中作为 SNMP 操作的基本单元。注意，表中的版本字段表示提出 PDU 时的 SNMP 版本。版本 1 的 PDU 仍然得到了广泛的应用，版本 2 中某些功能得到增强。

表 6-25 SNMP 中的基本操作

PDU	描　述	版　本
GetRequest	取回叶子对象的值	V1
GetNextRequest	获取字典顺序上接近指定对象的对象	V1
SetRequest	设置（更新）叶子对象的值	V1
GetResponse	对 GetRequest（值）或 SetRequest（ACK）做出响应	V1
Trap	由代理发出，一步通知管理工作站发生了某些事件	V1
GetBulkRequest	取回大的数据块，例如表中的多行	V2
InformRequest	允许 MS 发送陷阱信息给另一个 MS 并接收响应	V2

PDU：SNMP 操作中的基本数据单元；MS：管理工作站；变量绑定列表：PDU 中变量和对应值的列表

每个封装在 UDP 数据报中的 SNMP 消息都有三个主要部分：分组 SNMP 头部、操作头部和变量绑定列表。普通 SNMP 头部由 SNMP 版本、社区（用于访问控制的明文密码）和 PDU 类型组成。表 6-25 的第一列列出了可能的 PDU 类型。操作头部提供了关于操作的信息，包括请求-ID（分配用于匹配未经确认的请求）和错误状态。由一系列变量值对组成的变量绑定列表用于支持信息交换。常规的操作分别由 GetRequest 和 SetRequest 执行单个检索和设置对象。但是，同时访问多个对象还是可能的。管理站点将对象的 OID 放入变量绑定列表中的“变量”字段中，然后将 PDU 发送给一个代理。代理依次填入对应值字段并用 GetResponse PDU 回复管理站。

遍历一棵 MIB 树

GetNextRequest 用来获取按字典顺序接近指定 OID 的对象。尽管在很大程度上它与 GetRequest 相似，但是它有助于查找 MIB 树的结构。为了澄清这种 PDU 的理念，让我们再一次使用表 6-24，但是这次是以树的形式显示在图 6-42 中。

图中存在层次化的关系，因此在 OID 树中有可以使用深度优先搜索（DFS）遍历的字典顺序。考虑管理站点只使用 GetRequest PDU 的情况。因为在一棵 MIB 树中的 OID 并不是连续的，所以如果它没有一张完整的 OID 表，管理站点就无法知道 MIB 的结构。然而，一旦配备了 GetNextRequest，管理站点就可以完整地遍历树。

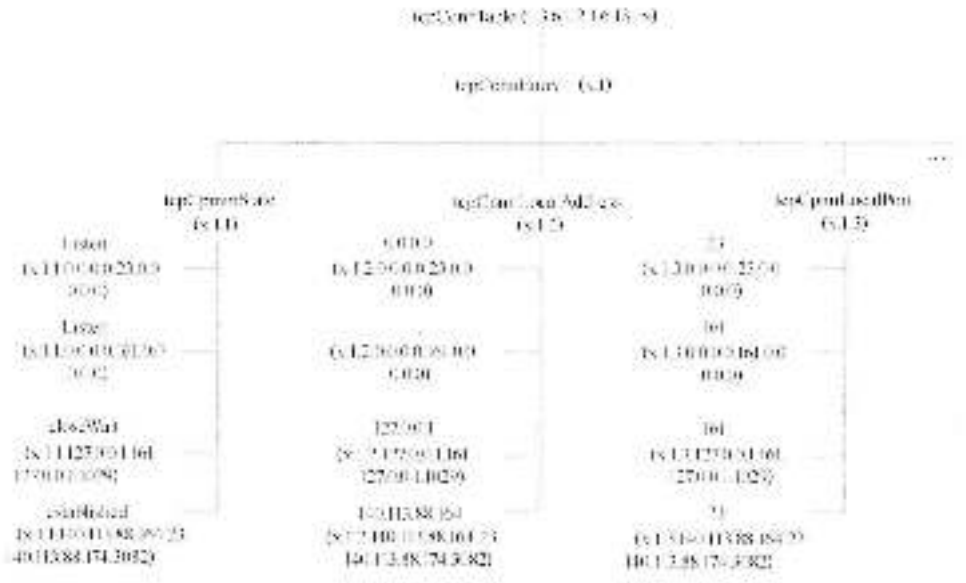


图 6-42 以字典型图表示的 TCP 连接表

MIB 对象的批量传输

为了效率，SNMPv2 采用了 GetBulkRequest PDU。与 GetNextRequest 相比，GetBulkRequest 支持更强的检索方案，范围检索多个对象，而不是多次迭代检索。管理站点具体在 PDU 中指定启始 OID 和检索范围。接收 PDU 的代理给管理站点发送回一个响应实例表中嵌入了请求变量值对的变量 GetResponse。对于图 6-42 中的例子，GetBulkRequest(2, 4) (system, interface tcpConnState, tcpConnLocalAddress, tcp ConnLocalPort) 将返回 4 个变量值对。

开源实现 6.5：Net-SNMP

概述

Net-SNMP 最初开发于卡内基梅隆大学（大约在 1995 年）和美国加州大学戴维斯分校（1995—2000 年），目前该软件包由 NET-SNMP 开发团队维护（自从 2000 年以后），参见网站 http://sourceforge.net/projects/net-snmp。它提供：1）具有 MIB 编译器的可扩展代理，通过它人们可以开发自己的 MIB；2）可以进一步开发的 SNMP 库；3）从 SNMP 代理获取或设置信息的工具；4）产生和处理 SNMP 陷阱的工具。它还支持 SNMPv1、v2、v3 和其他 SNMP 相关的协议。与本章中其他大多数开源实现不同，Net-snmp 是一种迭代实现，同时支持无状态 SNMP 协议的非面向连接（在 UDP 端口 161）和面向连接（在 TCP 端口 161）模型。

基本命令和例子

表 6-26 显示了 Net-SNMP 中和相应 PDU 使用的一些命令描述。基本上它们是不同版本的 PDU 实现。在图 6-43 中，我们使用 snmpget、snmpset 和 snmpwalk 命令进行说明。snmpwalk 使用 GetNextRequest PDU 检索子树下的所有对象。我们使用预配置的用户名如 "ynlin" 和密码如 "ynlinpasswd" 来检索对象实例 system.sysContact.0。安全级别设置为 "authNoPriv"（意味着只需要认证，不需要数据和密性，即数据加密），验证方法设置为 MD5。

表 6-26 在 Net-SNMP 中用于查询、设置和陷阱的一些命令

名　字	描述和例子	使用的 PDU
SNMPGET	发送 get 查询一个叶子节点的值	GetRequest
SNMPSET	设置（更新）叶子节点的值	SetRequest
SNMPBULKGET	一次获取多个对象，可能在不同的子树下	GetBulkRequest

（续）

名　字	描述和例子	使用的 PDU
SNMPWALK	改变 MIB 一网扩下的所有对象	GetNextRequest
SNMPTRAP	使用 TRAP 请求向网络管理器发送信息。多个对象标识符可以用作参数	Trap
SNMPSTATUS	用于从一个网络实体中查询几个重要统计量。如果有错误的话，还会报告错误	
SNMPETSTAT	显示从远程系统上查询的各种网络相关信息值，使用 SNMP 协议	

算法实现

图6-44 显示了 Net-SNMP 内部是如何运行的。通过执行带有某些选项（如 syslog 选项）和启动某些模块的 snmpd 来启动服务器。然后调用 init_agent() 读配置文件，建立需要的数据结构（如对象树），同时初始化其他的子代理（如 AgentX）。并且由 init_snmp() 进一步完成配置。它也用于解析 MIB 模块。然后主代理就启动了。它定义需要的会话，其结构如图6-45 所示，同时为相应的会话注册回调。例如 handle_master_agentx_packet() 函数就是为特定 AgentX 分组处理的名为 sess 的会话进行注册。最后，程序进入接收循环处理各种会话。

```
$ snmpget -v 3 -u ynlin -l authNoPriv -a MD5 -A ynlinsnmp localhost system.sysContact.0
system.sysContact.0 = ynlin@cis.nctu.edu.tw

$ snmpset -v 3 -u ynlin -l authNoPriv -a MD5 -A ynlinsnmp localhost system.sysContact.0
s gis88559 system.sysContact.0 = gis88559

$ snmpget -v 3 -u ynlin -l authNoPriv -a MD5 -A ynlinsnmp localhost system.sysContact.0
system.sysContact.0 = gis88559

$ /usr/local/bin/snmpbulkwalk -v 3 -u ynlin -l authNoPriv -a MD5 -A ynlinpasswd localhost
system system.sysDescr.0 = Linux ynlin2.cis.nctu.edu.tw 2.4.14 #5 SMP Thursday
November 22 23:6 system.sysObjectID.0 = OID: enterprises.ucdavis.ucdSnmpAgent.linux
system.sysUpTime.0 = Timeticks: (30411450) 3 days, 12:28:34.50 system.sysContact.0 =
gis88559 system.sysName.0 = ynlin2.cis.nctu.edu.tw system.sysLocation.0 = ynlin2
system.sysORLastChange.0 = Timeticks: (0) 0:00:00.00 system.sysORTable.sysOREntry.
sysORID.1 = OID: iIMIB
system.sysORTable.sysOREntry.sysORID.2 = OID: .iso.org.dod.internet.snmpV2.snmpB
system.sysORTable.sysOREntry.sysORID.3 = OID: tcpMIB system.sysORTable.sysOREntry.
sysORID.4 = OID: ip system.sysORTable.sysOREntry.sysORID.5 = OID: udpMIB
```

图 6-43　SNMPv3 中 snmpget、snmpset 和 snmpwalk 的例子

会话由通过 I/O 与其他守护进程复用的 select() 函数服务。但是，snmp_select_info() 函数与这项技术无关。相反，它执行管理工作针对：1）对于即将到来的 select() 活动会话；2）即将关闭的会话和记录在 fd_set 和 numfd 结构上的活动会话。snmp_read() 函数读取选择会话的请求。它会使用 snmp_parse() 函数检查 fd_set 中的分组是否属于 SNMP 分组，然后从请求中去掉不必要的部分，从而形成了 SNMPPDU。将由此产生的 PDU 传递给前面为会话注册的回调例程，一旦例程成功返回，就将所要的信息返回给请求者。

最后，notsnmp_check_outstanding_agent_requests() 检查看是否有在任何未经确认的差机请求。如果有者，它就使用访问控制模块（ACM）进行验证，一旦验证通过，就处理请求。

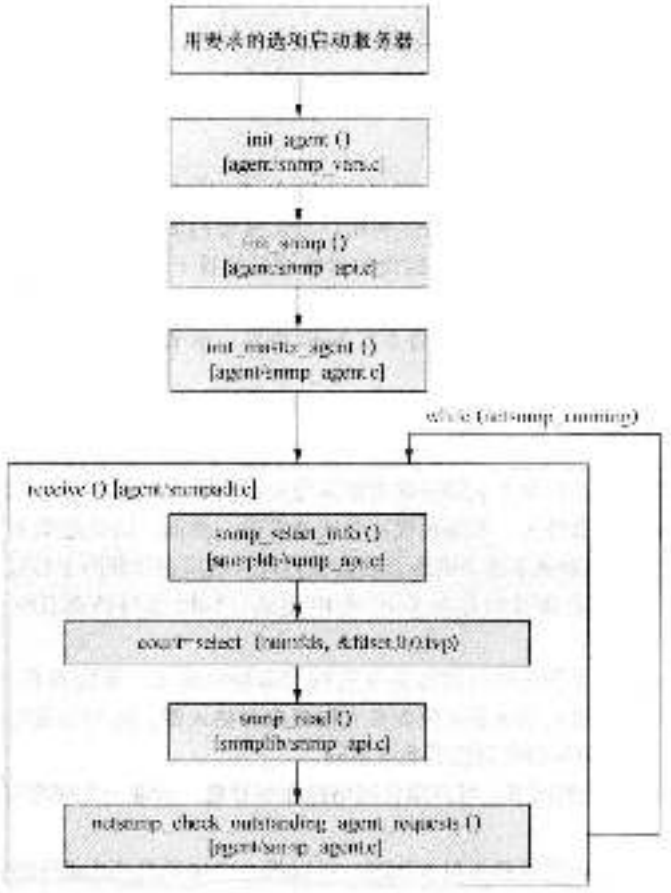


图 6-44 Net-SNMP 内部的处理流程

```
/** snmp version */
long          version;
/** Number of retries before timeout. */
int           retries;
struct snmp_session *subsession;
struct snmp_session *next;
/** UDP port number of peer. (NO LONGER USED - USE peername INSTEAD) */
u_short       remote_port;
/** My Domain name or dotted IP address, 0 for default */
char          *localname;
/** My UDP port number, 0 for default, picked randomly */
u_short       local_port;
/** Function to interpret incoming data */
netsnmp_callback callback;
/** Session id - AgentX only */
long          sessid;
* SNMPv1, SNMPv2c and SNMPv3 fields
```

图 6-45 一个会话的结构（部分）

当模块需要更多的时间来完成传入的请求时，它就将请求标记为 delegated（授权）的并返回，允许代理处理其他请求。例如，代理将任何必须由 AgentX 子代理处理的请求标记为 delegated（授权），以便在等待子代理回应时能够有空处理其他的请求。NET-SNMP 要求所有 pending delegated（等待的授权）请求在 set 请求处理之前处理完。如果仍有 pending（等待）请求，那么 set 和所有其他到达的请求就要排队等待，直到它们完成为止。

练习

1. 查找实现 rcc 操作的 .e 文件和代码行。
2. 查找 SNMP 会话的准确结构定义。

6.7 VoIP

在本章所描述的两个实时应用中，IP 电话（VoIP）被认为是硬实时的而流媒体是软实时的。前者拥有大约 250 毫秒的往返时间（RTT）约束作为用户延迟察觉的域值，但后者可以容纳数秒延迟播放的 RTT。21 世纪 00 年代初以来，由于大量铺设的过供给光纤骨干网，所以 VoIP 开始普及。随着它的不断发展，逐渐形成了两个标准：ITU-T 的 H.323 和 IETF 的 SIP。SIP 最终获得了胜利，但还没有在市场中占据主要地位，因为还有多个其他的专有 VoIP 协议。本节将介绍和比较 H.323 和 SIP，并以 Asterisk 为例说明 SIP 的开源实现。

6.7.1 简介

电话服务及其相关设备在世界上大部分地方被认为是理所当然。不论固定还是移动电话的可用性，以及对低成本、高品质的网络接入，都是现代社会所必需的。然而，语音通信不再被传统的公共交换电话网（PSTN）所主导。随着越来越多的语音通信被打包，然后在互联网上传输，通信范式迁移已经发生。使用互联网协议进行语音通信称为 VoIP 或 IP 电话，VoIP 变得特别有吸引力，因为具有以下优点：

廉价的费用 长途电话尤其是拥有国际分支机构和市场的公司，可能有真正意义上的费用节省。在互联网上统一费率的收费模式意味着你只需要支付固定的接入费，而与你发送多少数据或持续多长时间发送数据无关。这与 PSTN 的收费模式非常不同。

简单 一个集成语音/数据网络，可以简化网络操作和管理。管理一个网络应该比管理两个网络更有成本效益。

消耗更少的带宽 使用脉冲编码调制（PCM）可以把一个电话公司电路中的语音信道分成一种标准的 64 kbps。另一方面，在 IP 网络下，如果具备强大的编解码器，那么使用 G.723.1 可以把单个语音信道的带宽可以进一步降低到 6.3 kbps。

可扩展性 支持利用实时语音通信和数据处理的新类型服务。例如，新功能可以扩展到白板、呼叫中心、远程办公、远距学习等应用。

虽然 VoIP 有许多优点，但是需要解决服务质量（QoS）等问题，以便减少从 IP 网络继承来的丢失、延迟和抖动的影响。由于 2000 年前后对光纤网络基础设施的巨大投资，现在 VoIP 应用程序的运行在大部分地区还是令人满意的，这是 10 年前不可想象的。

本节将介绍两种 VoIP 协议，H.323 和 SIP，以及它们的扩展架构。由国际电信联盟（ITU-T）定义的 H.323 协议开发得较早，但已经被 IETF 的 SIP 取代。SIP 的简单性使它成为比 H.323 更令人满意的解决方案。1999 年 SIP 定义在 RFC 2543 中，后来因为 2002 年 RFC 3261 的出现而被废弃。

历史演变：专用 VoIP 服务—— Skype 和 MSN

在 VoIP 的历史中曾经开发过很多应用程序，既有公用也有专用的。但是其中只有部分流行起来，如 Skype（专用的）、MSN（专用的）和 Asterisk（开源的）。令人惊讶的是，上述三者中只有 Asterisk 遵循 SIP 协议，而其他两者开发了自己的协议，分别是加密的 Skype 协议和 MSNMS（MSN 信使服务）协议。虽然 MSN 在其 2005 年的版本中提供了 SIP 选项，但它却抛弃了互操作性。

因为它们已被广泛地讨论和分析过，所以相信它们采取了同样的方法，既一种类 SIP 协议。然而，由于商业上的考虑，通过使用不同于 RTP/RTCP 的传输协议和不同的编解码器维护专用社区群体。这种趋势非常类似于传统的电信市场，因此不同供应商的产品之间很少具有兼容性。2010 年年初，Skype 的用户人数为 4430 万（每天有 422 万的活跃用户），而 MSN 的相关信息没有披露。由于安装和操作的复杂性，Asterisk 的用户数可能比其他两个都要少得多。绝大多数采用它的用户都是公司，而不是终端用户。

6.7.2 H.323

H.323 协议曾经被广泛商业产品采用的与计算机兼容的 VoIP 协议。它首次发布于 1996 年，原来是针对局域网上的多媒体会议，但后来扩展为代表 VoIP。进一步的增强功能包括通过资源预留协议（RSVP）终端配置 QoS、URL 格式的地址、呼叫建立、带宽管理和安全特点等。

H.323 网络中的元素

H.323 环境称为区，通常包括 4 种元素：一个或多个终端、网关、多点控制单元（MCU）和一个管理关守。

1）终端　H.323 终端通常是客户端软件，用于初始化与另一个 H.323 终端、多点控制单元或网关的双向通信。

2）网关　网关充当 VoIP 区和其他类型网络（通常是 PSTN 网络）间中间人，为双向通信提供转换服务。

3）多点控制单元　MCU 就是操纵 3 个或多个终端或网关参与多点会议的 H.323 终点。它既可以是独立的也可以集成到终端、网关或关守中。

4）关守　关守为网络中的其他实体提供地址转换、接入控制、带宽控制等各种服务。像定位服务，因为注册的终端定位网关和通话管理等辅助服务也可以包括进来。然而，由于两个终端仍然可以在没有额外服务支持的情况下相互通信，所以它是可选的。

图 6-46 显示了在 H.323 区中 4 个元素之间的关系。普通的 VoIP 事务可以描述如下：在 H.323 网络中的每个实体都具有唯一的网络地址。当终端想要与另一个终端连接进行语音会话时，如果需要它首先向关守发出呼叫许可的请求。如果被允许，那么主叫方发送一个包含指目标地址和端口的连接请求给远程终端，如 ras://host@ domain:port。经过一些能力协商后，在两个终端之间就建立一个通信信道。MCU 和网关分别只应用于三次呼叫和互联网络呼叫中。

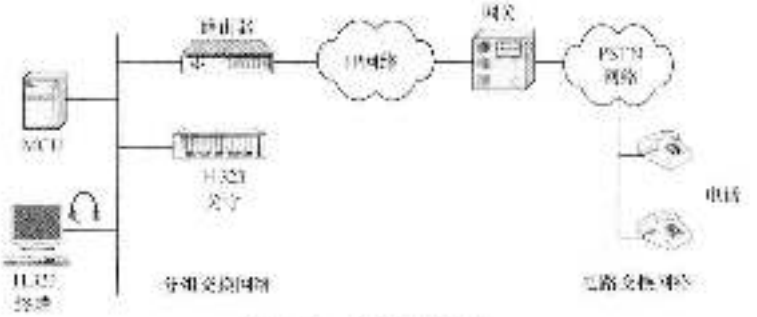


图 6-46　H.323 的环境

H.323 的协议栈

图 6-47 显示了 H.323 协议栈，它可以分成两个平面：控制平面和数据平面。控制平面负责协调 VoIP 会话的建立和拆除，而数据平面处理语音或多媒体数据的编码和传输。下面描述每个协议的功能。

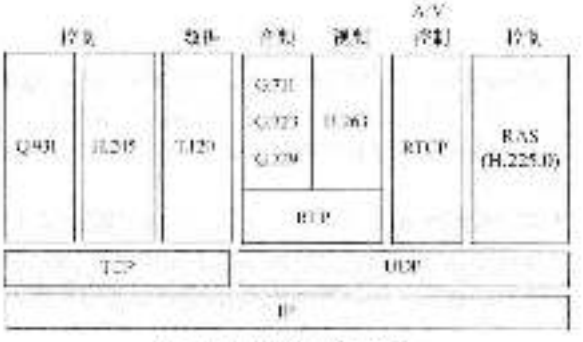


图 6-47　H.323 的协议栈

注册许可和状态（RAS）是一种在关守和终端之间的信令方法。它定义在 H.225.0 中，支持注册/注销、许可、带宽更改和到终端呼叫的解除。Q.931 是一个用于两个终端之间呼叫的建立和拆除的信令方法。因为它是为 PSTN 定义的 Q.931 协议的一个变种，所以 H.323 和 PSTN 的网络互连及计费被简化了。H.245 用于能力协商，即两个终端之间的音频（G.711、G.723、G.729）或视频（H.263）编解码器的类型，并确定终端之间的主从关系。主从区分是必要的，因为需要一个仲裁者（主）来描述逻辑信道的特点并为所有的 RTP / RTCP 会话确定组播地址。初始化完成后，可以通过 H.245 建立很多逻辑信道。T.120 是一套服务于多媒体会议的数据协议组成，如应用程序共享、电子白板和 VoIP 会话期间的文件传输。

如 5.5 节所述，实时传输协议（RTP）是一个简单的协议，旨在通过利用没有出现在已有的传输协议（如 UDP）中的序列号来传输和同步实时流量。实时控制协议（RTCP）由是向 IETF 作为 RTP 的相关协议定义的，按照会话中的所有参与者用带性能传输控制分组实现的。它主要负责向所有参与者提供有关数据传输的质量反馈，这有助于选择正确的编解码器。例如，通过给 UDP 使用不同的端口号等方式，底层传输协议必须提供 RTP 和 RTCP 分组的复用。RTP 和 RTCP 在 5.5 节中介绍。除了它们，应用在 VoIP 的 H.323 和 SIP 中外，我们将在 6.8 节中还会看到它们应用在流媒体的连续中。

H.323 呼叫的建立过程

H.323 呼叫建立过程有两种情况：一个涉及关守，一个没有关守。通常情况下，一个高质量、完全可控制的呼叫涉及本地的与远程关守的合作。这种模型称为“关守路由呼叫信令”。

图 6-48 描绘了一个关守的跨出 H.323 呼叫的一般建立过程。存在关守的情况下，所有的控制信息，包括呼叫请求、都将发送或路由到关守。呼叫请求由关守中实现的 RAS 处理。由于注册许可和其他服务（如地址转移、带宽分配）。然后本地关守给被叫方发出一条建立消息，如果被叫者想要处理这次会话，它就会请求自己的关守。如果允许，被叫方发出一个肯定的应答答复初发起的关守，然后来来所有的控制消息都将经过这两个关守路由。



图 6-48 H.323 呼叫的建立过程

在两边的关守允许一个呼叫请求后，呼叫方将继续 Q.931 建立过程。Q.931 协议实际上仅是用于“振铃”并为 H.245 控制信道获得一个动态分配的端口。H.245 信道的建立过程经过能力协商和主从确立后，打开逻辑信道，两个终端可以开始基于 RTP 并由 RTCP 监控的会话了。信道的关闭和呼叫的断开是以类似的方式完成的。

不难看出，在 H.323 中信息交换的额外开销是巨大的，尤其是在使用关守路由呼叫时。为了克服这个缺点，引入了所谓的“快速连接”步骤。在快速连接步骤中，H.245 信息是在 Q.931 消息中携带的，并没有使用单独的 H.245 控制信道。因此呼叫完成主会变得更快。通过发送 Q.931 Release Complete（释放完成）消息来实现呼叫释放。与 H.245 一样，Q.931 消息还起到关闭所有与呼叫相关的逻辑信道的作用。

6.7.3 会话初始化协议

会话初始化协议（SIP）是一种 VoIP 可靠的信令协议。它由 IETF 提出，由于它的简单性和与其他大多数也是由 IETF 定义的 IP 网络中的已有协议相兼容优势，它旨在取代 ITU-T 的 H.323。由于拥有很多其他补充协议所提供的会话描述和组播功能，所以它很容易处理建立、修改和指除多媒体会话。由于实时性的本质，所以它也依赖 RTP 作为其传输协议。

与基于文本的协议 HTTP 一样，SIP 也借用其消息类型、头部字段和客户端/服务器方案。然而，与 HTTP 通过 TCP 的特性不同，SIP 既可以使用 UDP 也可以使用 TCP。多个 SIP 事务可在一个 TCP 连接或者一个 UDP 数据流中携带。此外，用户的可移动性也由提供音前用户位置的代理和重定向来满足。

SIP 网络中的元素

由于 SIP 是基于客户端/服务器的，所以它必须至少有一个作为用户代理客户端（UAC）的呼叫方和一个作为用户代理服务器（UAS）的被叫方，再加上一些辅助服务器，如图 6-49 所示。

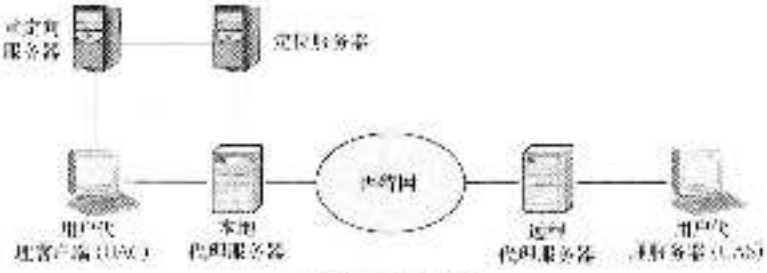


图 6-49 SIP 环境

1）代理服务器 与 HTTP 中的代理服务器一样，SIP 代理服务器充当客户端的角色，很可能在转换请求后将它转发给其他服务器。它可以用来存储计费或记账目的信息。

2）重定向服务器 重定向服务器通过告知其所请求的服务器地址来响应客户端的请求。它不会像代理服务器那样发起一个 SIP 请求，也不会像 UAS 那样接受呼叫。

3）定位服务器 定位服务器用来处理代理服务器或重定向服务器对被叫方可能位置的请求。它通常是一个使用非 SIP 协议或路由策略来定位用户的外部服务器。用户可以向服务器送回当前的位置。与其他 SIP 服务器协同定位是可能的。

UAC 直接向 UAS 或通过代理发出呼叫请求，又称为 INVITE 请求。在前一种情况下，如果 UAC 只知道 UAS 的 URL 但不知道 UAS 的位置，那么 INVITE 请求就被发送给重定向服务器，然后由重定向服务器向位置服务器询问有关 UAS 的位置信息，假设 UAS 的位置已经注册到服务器，那么就给 UAC 应答。如果使用代理，那么 UAC 只是将请求发送给代理而不关心 UAS 的位置。代理将联系定位服务器。通常情况下，代理服务器实现重定向能力。当 UAC 与代理服务器联系时，它只指定它需要的服务是重定向还是代理。

SIP 的协议栈

如图 6-50 所示，需要多个协议才能建立 SIP 操作的基础。由 RTP 实时传输，由 RTCP 监控，这些与 H.323 相同。我们将详细介绍 SIP 及其补充协议，会话公告协议（SAP）和会话描述协议（SDP）。

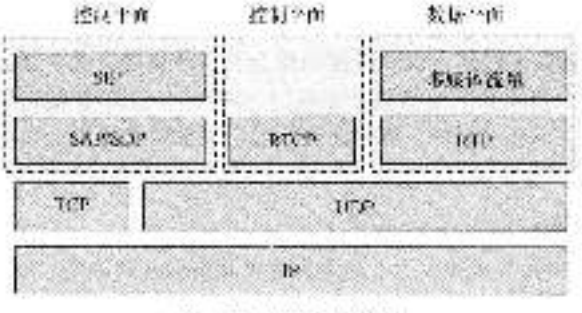


图 6-50 SIP 的协议栈

SIP客户端由“用户的主机”格式的SIP URL确定。请注意，这种类型的寻址看起来类似于电子邮件地址。用户部分可能是用户名或可能是电话号码，主机部分可能是域名、主机名或数字网络地址。例如，

```
sallee@sw.cnu.edu.tw and
+sbkk@sw.cu.edu.tw
```

SIP呼叫的建立过程

一旦知道被叫方的地址，主叫方就发出一系列命令或操作来发起一次呼叫。表6-27列出了SIP中的操作符或命令。第4个操作符用于呼叫建立和拆除。通常情况是，呼叫方给新VoIP会话的URL中指定的被叫方发送INVITE（它通常包含SDP中的会话描述），然后等待响应。如果知道目的IP地址，那么请求直接发送给被叫方；否则，以重定向模式或代理模式将它发送给拥有内置定位服务器的本地代理服务器。对于后一种情况，代理将根据来自位置服务器的地址信息转发INVITE消息，很可能经过其他代理，最后到达目的地。

表6-27 一些SIP命令

操作符	描述	操作符	描述
INVITE	邀请用户参与一次呼叫	OPTIONS	关于呼叫支持的功能
ACK	对最终响应做出确认	REGISTER	向位置服务器注册当前客户端的位置
BYE	终止两个端点之间的呼叫	INFO	用于中间会话的信息
CANCEL	中止对用户的搜索或对呼叫的请求		

现在被叫方的电话振铃。如果被叫方同意本地机器接受会话要求并愿意参加会话，那么就接通电话，他就利用正确的应答码（如200 OK）来应答呼叫方，如表6-28所示。然后呼叫方通过发送ACK消息确认被叫方的响应。因此完成握手并开始会话。但是，有时会存在被叫方繁忙，在很长一段时间内不能处理INVITE请求的情况。在这种情况下，呼叫方可能会放弃并发送一个CANCEL消息。

表6-28 SIP应答代码

应答码	描述	应答码	描述
1xx（通知性的）	尝试、振铃和排队	4xx（请求失败）	来自特定服务器的失败响应
2xx（成功）	请求成功	5xx（服务器失败）	当服务器本身有错误时，给出的失败响应
3xx（重定向）	给出有关接收方新位置的信息	6xx（全局故障）	忙、拒绝、请求不可接受

当通话将要结束时，参与者之一挂断电话并导致发送一个BYE消息。然后接收主机用200 OK来响应以使确认收到的消息，同时呼叫终止。

当SIP在会话中充当命令发生器时，基于文本协议的SDP也用来为会话双方描述会话的特性。会话由很多媒体流组成。因此，会话的描述涉及每个媒体流的很多参数规范，如传输协议和媒体类型（音频、视频或应用），以及会话本身，如协议版本、来源、会话名称和会话的起始/终止时间。

虽然SDP描述会话的特点，但是它没有提供在会话建立的开始时会话通知方式。因此，SAP用于在SIP会话期间向参会者通知多媒体会议和其他组播会话，并发送有关会话建立的信息。SAP广播员周期性地发送通告分组给众所周知的组播地址和端口（9875），这样潜在的会话参与者可以使用会话描述来启动所要求的工具加入到会话中。需要注意的是，包含组播传送会话描述符的分组的有效载荷必须以SDP格式书写，以利于参与者之间的互操作，因为在SAP通告中没有能力协商。

历史演变：H.323与SIP对比

虽然H.323协议早在1996年就定义了，但它却没有得到相应的市场。分析家提出H.323失败的各种原因，包括其复杂的信令、可扩展性问题、安全问题。这也是SIP得以开发的原因——为了具有轻量级和易于实施。除了其他优点外，SIP还是一个视觉标准，定义在IFC 2543中。后来因为得到IETF

此次和支持的 RTC 3261 的出现而被应用。尽管如此，H.323 仍然有其优点。以下是这两种协议之间存在的某些其他方面的差异。

1）编码格式　H.323 以二进制格式编码消息，而不是以 ASCII 文本格式，因此它更紧凑，更适于传输。但是，使用 ASCII 字符的开发人员更容易调试和调试。在 SIP 中还提供 ASCII 编码的方法。

2）报头类型　H.323 协议可以交换和协商信道类型、音视频、音频和视频信道。SIP UAC 只能发送一个信道，在其中具体 UAS 会发到该信道。如果不支持，UAS 用一个编码代码，如 488（该位置不可接受）或 606（不可接受），或者专代码，如 304（媒体类型不可用），响应 INVITE 请求。

3）控制会议　H.323 支持控制、音频和视频会议（利用 T.120）。它还确定了控制会议的规程，而 SIP 只支持控制和音频会议，却不能控制会议。

开源实现 6.6：Asterisk

概述

与其学习一个商业的点到点 VoIP 软件，我不如研究一下免费的 PBX（专用交换机）的系统——Asterisk，它使 PSTN 以及传输软件电话之间及音软件电话与在 PSTN 中的传统电话连接起来。如图 6-51 中所示，一台 Asterisk 服务器充当 PSTN 和 VoIP 网络之间的通信翻译（communicator）。VoIP 网络可以包括一台基于 PC 的安装了 VoIP 软件的电话或者一部支持 SIP 功能的电话。为与一台能够传输报信号的模拟 VoIP 设备进行模拟电话连接（ATA 适配器）准备时，甚至可以使用传统的电话。从技术上讲，Asterisk 是面向连接的、有状态的 SIP 协议和无连接的、无状态的 RTCP/ RTP 协议的开放式网关。它确定的时间并不是固定的。

Asterisk 提供了一种用来构建定制 VoIP 系统的框架，如图 6-52 所示，内置的灵活性来自模块的添加和删除，如编码、RTP 和用于建立基本呼叫服务的底层器。Asterisk 的核心功能是作为交换系统（也就是说，在办公室或建筑物内）呼叫的 PBX。然而，为了易于部署或的管理也可能还有额外的实用程序工具，如 HTTP 服务器、SNMP 代理和呼叫详细记录（CDR）引擎。



图 6-51　基于 Asterisk 的 VoIP 环境

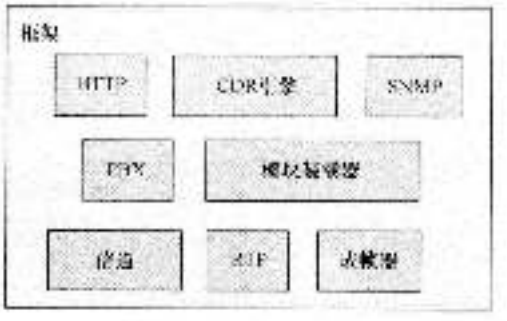


图 6-52　Asterisk 的框架

数据结构

PBX 将呼叫交换到其对应的目的地。然而，某目的地还可能有多个分机号码，因此需要为一个指定的号都交换。为了实现这种方案，在 Asterisk PBX 为都引进了名为上下文和扩展的两个概念，其中后者扩大了被呼叫组，而前者则进一步扩展了支持的组数。如图 6-53 所示，通过集成上下文多家公司或组织中的每个成员都可以有自己的扩展空间，同时共享一台 PBX。

通过进一步设计，这样每个扩展可以有多个步骤（按照标为优先级）以便能够组织一个条件开户方（自动化自动应答机）自己呼叫的拨号计划。将优先级与执行特定行动的应用程序相关联。例如，呼叫可以由以下组成：1）呼叫行动，通过“Dial”应用程序以优先级 1 连接到被叫方；2）应答行动，涉及应用程序“Answer”以优先级 2 播放初步录制的声音文件；3）挂断行动，应用程序“Hangup”以优先级 3 关闭信道。

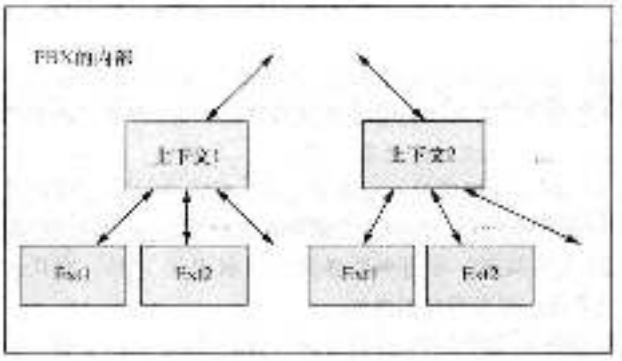


图6-53 多组的上下文及其扩展

算法实现

Asterisk 的内部执行可以分成4个步骤：1）管理接口的初始化；2）分析所需要参数（如优先级和应用程序）的呼叫发起；3）数据传输气通建立；4）派生一个服务线程，以便建立PBX结构并进行通话。

管理接口的初始化

详细的处理流程如图6-54所示。应用始于调用 init_manager() 加载配置并注册事件的回调函数，即所谓的“行动”。除了上面提到的那些“行动”外，还包括以下几个：1）ping 用于测试两个对点并保持连接状态；2）Originate 用于发起呼叫；3）Status 用于列出当前通道状态。初始化完成后，它就开始监听连接请求。请注意，在 Asterisk 中的“管理器”会话，通常是指一个 HTTP 会话管理接口为用户执行要求的动作。因此，可能有多个管理器会话通过非阻塞套接字创建相应的线程，并且对这些会话引触发的行动实用事件队列。

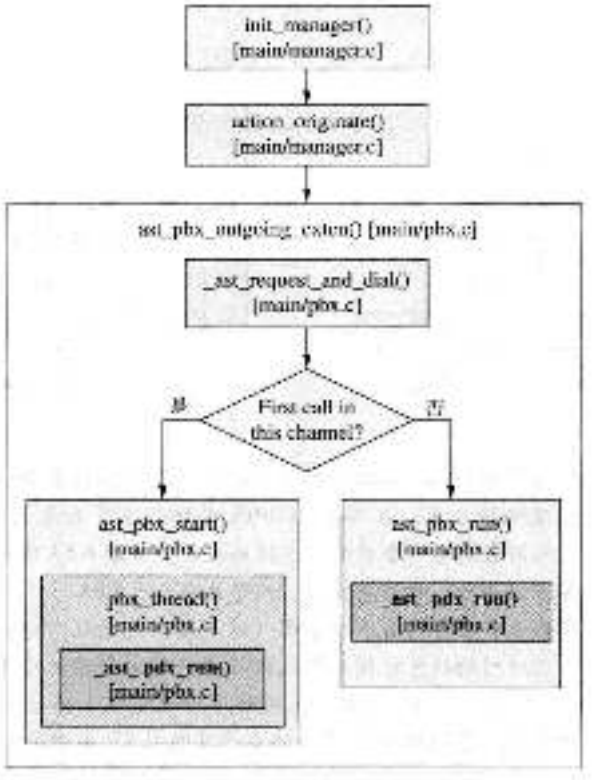


图6-54 Asterisk 内的执行流

呼叫发起

当管理器会话的用户进行呼叫时，就使用一个包含各种描述呼叫的参数，如呼叫者 ID、动作 ID、信道名称、扩展/上下文/优先级，用户客户和应用的调用 action_originate()。发起的动作实际上是放在呼叫方队列中，而不是立即执行，以避免在资源不足的情况下发生同时呼叫的事件。ast_pbx_outgoing_exten()包含了一系列执行呼叫的步骤处理程序，在用户客户的身份验证后就执行。

在 ast_pbx_outgoing_exten()调用_ast_request_and_dial()后，随后调用 ast_request()，然后调用 chan->tech->requester()，请求一个用于传输语音的信道。

信道建立

chan 是一个描述信道的 ast_channel 结构的实例。结构属性中最关键的部分与 tech 有关，结构 ast_channel_tech 的一个实例用来指定传输技术。由于 chan->tech->requester()已经定义为函数指针，所以这里或们的算实取的情况是 sip_request_call()注册或相应的回调函数以便请求一个基于 SIP 的信道。最终获得一个信道，并且信道标识符也将沿路返回给 ast_pbx_outgoing_exten 以管未来其他上层程序使用。

位于 channels/chan_sip.c 中的 sip_request_call()检查是否有指定的编解码器的支持，如果有，然调用 sip_alloc()建立一个 SIP 的专用数据记录 sip_pvt。sip_pvt 结构由数十个在会话注册期间描述 SIP 会话的专用和人对话的内容和呼叫占位的元素组成，例如，主叫方标识符、IP 地址、能力、SDP 会话标识符和 RTP 会话字描述符。

线程派生

ast_pbx_outgoing_exten()既可以调用 ast_pbx_start()也可调用 ast_pbx_run()系统调号，这取决于这是否是到相同目的地的第一次调用。如果它确实是第一次呼叫，就创建一个服务线程来执行 pbx_thread()，然后调用 ast_pbx_run()，并将呼叫计数递增 1。

在图 6-55 中，_ast_pbx_run()作为呼叫的主要服务处理，使用 ast_calloc()为描述建立专用 PBX 结构并使用 ast_cdr_alloc()记录调用活动的 CDR 结构。然后，它针对该和上下文/扩展的所有优先级进行循环，使用 ast_spawn_extension()执行指定的应用程序直到触发了挂断事件并由 ast_handup()处理（即以前述的挂断动作）。在 ast_spawn_extension()内部，如前所述，预注册的回调函数 sip_request_call()被调用以使构造描述 SIP 会话的 PVT 结构。之后在 sip_request_call()执行期间已通过执行 ast_rtp_new_with_bindaddr()（未在本图中显示）发起一个 RTP/RTCP 传输并且分配 PVT 结构。

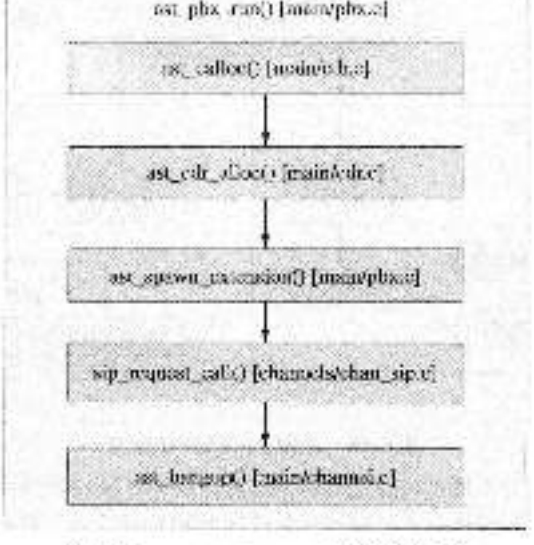


图 6-55 _ast_pbx_run()函数的内部

练习

1. 查找将 sip_request_call 注册为回调函数的 .c 文件以及其中具体的代码行。
2. 描述 sip_pvt 结构并解释该结构中的重要变量。
3. 查找为 SIP 会话建立 RTP/RTCP 传输的 .c 文件及其具体的代码行。

6.8 流媒体

2000 年前后光纤骨干网的巨大投资之前，作为一种软实时应用，流媒体与 20 世纪 90 年代末的 VoIP 相比，更快流行起来。它可以吸收，因此建立比 VoIP 高得多的延迟和抖动。在本节中，我们首先介绍流媒体客户端和服务器的体系结构和组件。然后，我们描述常见的压缩/解压缩技术。它们大大地降低了视频/音频位速率，有利于网络传输。接下来介绍并比较两种流媒体机制，实时流协议（RTSP）和 HTTP 流。介绍包括流中 QoS 控制和网步等高级问题。最后，作为开源实现的例子介绍达尔文流媒体服务器（DSS）。

6.8.1 简介

传统的多媒体娱乐大多数是在播放之前通过存储或下载到客户端 PC 上进行的。然而，这种边下载边播放的方式不能支持现场直播节目，并且例如对于录制好的节目在开始播放时需要大的延时以及在客户端上需要大容量存储。其目的是为了克服这些流媒体的缺点，用来即时向观众分发直播或录制好的媒体流。与边下载边播放的模式不同，电影的初始部分一旦到达客户端就可播放了。然后，传输和播放是并发或交替进行的。流媒体电影其实从未真正下载过，因为分组一旦播放完后就会丢弃。用这种方式，既节省了客户端启动延迟和存储的开销，又能够支持现场直播节目。

为了形成流媒体系统，需要很多功能。例如，需要一种压缩机制将来自数码相机的视频和音频数据转换成合适的格式。我们还需要专用的传输协议用于实时数据传输。必须提供 QoS 控制，以确保平滑的流媒体会话。客户端需要以硬件或软件的解压缩器或解码器和播放机制以适应延迟、抖动和损失。需要一些同步机制以协调视频和音频的播放。

流媒体的体系结构如图 6-56 所示。通常有两种类型的参与者：分发媒体内容的流媒体服务器和参加多媒体会话的很多客户端。一般的流媒体处理概括如下：来自录音设备的原始视频和音频数据需要进行压缩，即编码，并存储在存储设备中。当收到客户端的请求时，流媒体服务器检索通过传输协议发送的存储内容。发送内容之前，调用某些应用程序级的 QoS 控制模块，根据网络状态和 QoS 要求调整位流。

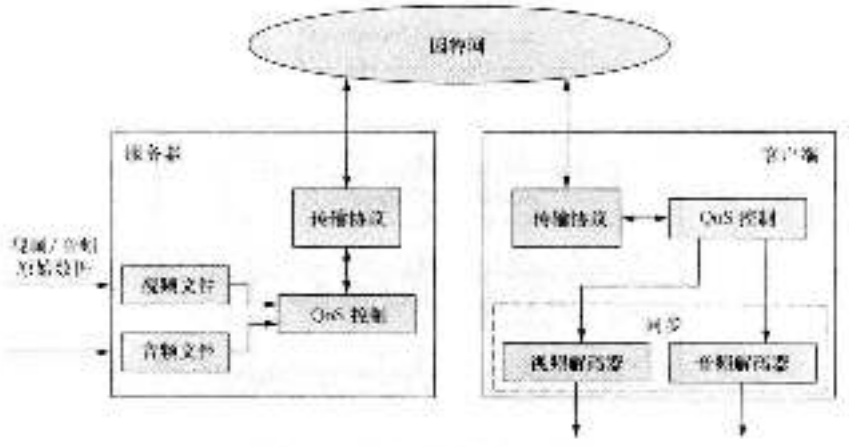


图 6-56 流体系的结构和组件

在客户端成功接收后，分组通过传输层然后由接收方的 QoS 模块处理，并最终在视频/音频解码器解码。在分组播放之前，执行媒体同步机制以便同步视频和音频表示。我们在接下来的三个小节中阐述这些组件。

6.8.2 压缩算法

事实上，原始视频/音频数据是巨大的。数十秒原始的，未压缩的 NTSC（国家电视系统委员会，一个电视标准）文件将占用高达 300 MB 的存储空间。这就是为什么非常需要压缩，尤其是要创建足够小的能在 Web 上播放的视频文件的原因。

压缩算法分析数据并删除或更改位，以便在尽可能减少文件大小和位速率的同时保留原始内容的完整性。通常要检查压缩算法的 3 个特点：时间和空间，有损或无损，对称或不对称。

空间和时间

空间压缩寻找静止帧中类似的模式或重复。例如，在一张包括蓝天的图片中，空间压缩会注意一个特定的区域，即天空，它包含了类似的像素。通过录制更短的位流来表示“指定的区域是浅蓝色”来摆脱描述成千上万的重复像素的负担。ITU-T 和 ISO 识别的几乎所有视频压缩方法/格式都采用离散余弦变换（DCT）实现空间冗余的减少。

另一方面，时间压缩则是查找帧序列的变化。例如，在一次谈话的视频剪辑中，因为在通常情况下，它只有话筒移动，时间压缩只会注意到环绕话筒的像素更改。它与第一帧比较，这是经过全面描述的并称为关键帧，接下来的称为增量帧，用以发现变化。关键帧之后，只在随后帧中保留更改信息。如果有一个场景变化那么它的大部分内容与前一帧不同，它就将新场景的第一帧称为下一个关键帧，并继续将后续帧与这一新的关键帧进行比较。因此，结果文件的大小对关键帧的数量相当敏感。运动图像专家组（MPEG）标准中最流行的一种视频编解码器——就是采用时间压缩。

注意，这两种技术并不相互排斥。例如，几乎所有的 QuickTime 电影都同时包含两种压缩技术。

无损或有损

压缩算法是无损还是有损取决于解压缩文件时是否能够恢复所有的原始数据。对于无损压缩，原先该文件中的每一个数据位经过文件解压缩后仍保持不变。所有的信息都完全恢复。这通常是文本或电子表格文件选择使用的技术，如果丢失文字或财务数字肯定会带来问题。对于多媒体数据，图形交换文件（GIF）是一种用于在 Web 上的提供无损压缩的图像格式。其他格式包括 PNG 和 TIFF。

另一方面，有损压缩通过永久地清除某些信息特别是冗余信息来缩减文件。当解压缩文件时，虽然用户可能没有注意到，但只有部分的原始信息保留下来。有损压缩通常用于视频和音频，大多数用户检测不到一定量的信息丢失。JPEG 图像文件通常用在 Web 上的照片和其他复杂的静止图像，它是有损压缩。使用 JPEG 压缩，编辑者能够决定引进多少损失并在文件大小和图像质量之间进行折中。视频压缩标准（如 MPEG-4 和 H.264）也采用比无损方案相对较大压缩比的有损压缩。

对称或不对称

对称和不对称之间的主要区别在于压缩和解压缩所需要的时间。对称方法压缩和解压缩的时间是相同的，而采用不对称方法时它们是不同的。更具体地说，不对称意味着它需要更多的时间来压缩多媒体数据，在某种意义上，由此产生的通用也会较高。因此，流媒体服务器一般进行不对称文件压缩（如 MPEG 和 AVI 视频）以减轻压缩负荷，并向客户端提供满意的质量。尽管如此，对于通过手机的实时视频会议，经常使用对称编解码，如 H.264。目前编码器硬件根本就没有强大到足以支持不对称方案。

6.8.3 流媒体协议

图 6.57 是流媒体协议栈。虽然还有其他的专用流媒体协议，但这里我们介绍公共领域中经常使用的两种流媒体机制：HTSP 和 HTTP。

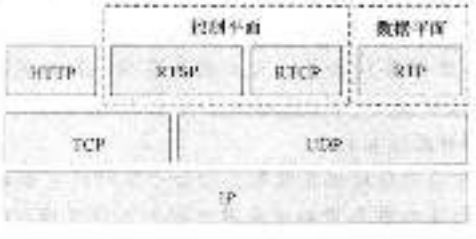


图 6.57 流媒体协议栈

实时流媒体协议

实时流媒体协议（RTSP）是一种客户端/服务器的多媒体会话控制协议，无论是大批观众（组播）还是媒体点播单个浏览器（单播）都能很好地工作。其主要功能之一，就是建立和控制流媒体服务器和客户端之间的视频和音频媒体。流控制方式是指可以通过使用电子邮件客户端获得定义在服务器端的播放说明文件。它声明编码、使用的语言、传输功能和其他参数，使客户端选择最合适的媒体组合。它还支持类似于VCR的控制操作，如停止、暂停/恢复、快进、快退。类似于SIP，RTSP协议也可以通话其他人参与现在的流媒体会话。总之，它具有以下属性：

1）HTTP的友好性和可扩展性　由于RTSP类似于HTTP有ASCII字符串的句法和消息格式，所以RTSP消息就可以由标准的HTTP分析器解析，而同时，可以很容易地添加更多的方法。URL和状态代码也可以重用。

2）独立的传输　通过客户端和服务器之间的传输协商，UDP和TCP都可以用来提供RTSP消息控制。然而，TCP并不适于传输多媒体播放，这依赖于基于时间的操作还是大规模的广播。通过TCP的HTTP流媒体将在后面学习。两种传输协议下的默认端口都是554。

3）能力协商　例如，如果不在服务器上实现搜寻，客户端就必须在用户接口中禁止移动滑动位置指示器。

RTSP方法

通过请求中的URL指示资源有多种实现方法。与只能由客户端发起请求的HTTP不同，支持RTSP的流媒体服务器可以与客户端通信以便通过ANNOUNCE更新播放说明文件，可以使用GET_PARAMETER作为“ping”检查客户端的健康。以下是在RTSP实现中必须支持的某些方法以便执行基本的RTSP会话。

1）OPTIONS　当客户端尝试一个非标准的请求时，可能发出OPTIONS请求。如果该请求得到服务器的允许，就返回一个200 OK响应。

2）SETUP　当在URL中检索数据流时，用来指定传输机制。

3）PLAY　PLAY方法通知服务器使用SETUP中指定的传输机制开始发送数据。请求头部的一些参数可以设置额外的功能。例如，将“缩放”设置为2，意思是加倍观看速率，即快进。

4）TEARDOWN　这是停止一个特定URL流媒体的发送。关闭相应的会话，直到发出另一个SETUP请求。

HTTP流媒体

除了RTSP协议外，还可以通过HTTP传输流视频和音频内容，这也称为伪流媒体。该窍是客户端需要缓冲当前媒体内容，这是通过TCP以可能高于播放器需要的带宽发送，并从缓冲区中播放。然而，由于TCP重传机制，它在发送后续分组之前会重发丢失的分组，所以它就更可能导致主要分组的丢弃、低性能、高时延抖动。因此它就不能提供UDP和RTSP那么多的内容。在5.5.1节中已经提到了这些问题。虽然这种方法并不稳健、有效，但它在没有RTSP支持时作为一种合理和方便的替代能够提供小规模的流媒体内容。

历史演变：Realplayer、媒体播放器、QuickTime和YouTube的流媒体

自从1995年RealNetworks引入流媒体后，多媒体播放器厂商开始外充自己产品线的流媒体功能。这样就产生了三大阵营：微软（媒体播放器）、苹果（QuickTime）和充驱RealNetworks（RealPlayer），除了这些外，其他播放器的市场规模都小得多。尽管如此，为了流形成一个完整的解决方案，播放器还需要与内容提供商（即一台服务器）合并。为此微软有Windows媒体服务器（专用），而苹果有QuickTime流媒体服务器（专用）和诗尔文流媒体服务器（开源），以及RealNetworks配备了Helix DNA服务器（同时支持专用和开源版本）。

虽然RTSP/RTCP/RTP基于公共传输体系结构，但由于不同的流媒体容器格式（例如，支持AVI、RM、WMV）和许可证限制，在这些服务器和播放器之间不支持互操作性。尽管如此，一般都支持标准格式，如MPEG。

另一个迅速崛起的流媒体技术是通过 Adobe 系统开发的 Flash 媒体服务器，使用专用的内容类型（FLV）和传输方法（实时消息协议，RTMP，通过 TCP），它已成为通过 HTTP 的点播视频流媒体的主要方法，在本节中称为伪流媒体。著名的视频分享门户网站 YouTube，就是采用这种技术。

6.8.4 服务质量和同步机制

用户直观感受问题在网络服务中一直都是很重要的。在流媒体中，有两个因素直接影响用户感知的质量：数据传输的服务质量（QoS）控制和视频、音频内容的同步。

服务质量控制机制

试想有一个具有差的服务质量控制的流媒体会话。在通常负载的网络下，质量一般能够得到满足。然而，当网络负载过重时，不断增加的分组丢失率会导致损坏或延迟帧，从而导致差的播放。此外，由于会话协调者不了解网络状况，甚至可能允许进入额外的流媒体数据从而恶化所有涉及的会话通量。

因此，服务质量控制的目标是在分组丢失的情况下最大限度地保证流媒体质量。流媒体中的服务质量控制通常采用速率控制的形式，试图通过使流速率与可用带宽相匹配来实现目标。这里，我们简要介绍两种速率控制方法。

1）基于源的速率控制　顾名思义，发送方通过对网络状况的反馈，或者根据一些模型公式来调整视频传输速率。反馈通常是从探测中获得的可用带宽。速率适配可以使分组丢失低于某一阈值。也可根据一些类似 TCP 的模型进行调整，使分组丢失可以像 TCP 一样得到缓解。

2）基于接收方的速率控制　根据接收方的速率控制，通过添加或删除与发送方的信道，接收方完成调节。由于视频可以分解成不同重要性的层，所以每一层都在相应的信道中传输，网络可以通过删去层，由此删除不太重要的信道。

还有另一种是基于前两种方法的混合版本。在这种版本中，通过添加或删除信道接收方调整接收速率，而发送方根据接收到的反馈调整发送速率。除了速率控制外，缓冲区管理机制（防止上溢或下溢以使实现流畅地播放）经常应用到接收方以使更好地容忍可能的网络变化。

同步机制

用户感知流媒体质量的第二个因素是视频和音频内容是否很好地同步。网络和操作系统可能造成媒体流的延迟，要求媒体同步以便保证在客户端多媒体播放的正确再现。

有三个层次的同步：流内同步、流间同步和对象间同步。下面简要地进行介绍。

1）流内同步　一个流由一序列需要严格排序的和很好的部分依赖于时间的数据单位组成。没有流内同步，流的播放就可能会被暂停、跳过，或者临时快进所干扰。

2）流间同步　由于多媒体会话主要由视频和音频流组成，所以不合适的流间同步会导致如嘴唇形状和扬声器声音之间的不匹配。

3）对象间同步　流媒体内容可以进一步抽象为对象级，从而分为两大类：依赖于时间的对象（用在上述两项方案中）和独立于时间的对象。独立于时间的一个很好的例子是出现在屏幕边缘的商业广告或图像，无论视频还是音频流。作为一种对象间同步的结果，商业广告可能会错误地显示，比如说，它不应该出现的一则新闻报道中。

开源实现 6.7：达尔文流媒体服务器

概述

达尔文流媒体服务器（DSS）是苹果公司 QuickTime 流媒体服务器（QTSS）的开源版本。DSS 允许用户通过互联网利用 HTTP 和 RTSP 发送流媒体。用户可以收听到现场或预先录制好的节目播放，或者他们也可以根据需要查看提供预先录制好的节目。DSS 提供了高度的可定制性，开发人员可以扩展和修改现有的模块以满足他们的需求。运行在各种操作系统上的 DSS 支持各种多媒体格式，包括 H.264/MPEG-4 AVC、MPEG-4 Part 2、3GP 和 MP3。此外，DSS 提供一种易于使用的基于 Web 的管理，身份认证，服务器端播放列表，中继支持和集成的广播者管理。

框图

DSS 可以分为两个部分：核心服务器和模块。图 6-58 显示了 DSS 框图。核心服务器就像是客户端和模块之间的接口以便提供任务调度，而模块被任务对象调用以便提供特定的功能。对象在后面的数据结构中定义。在这样的框架下，DSS 可以支持以接受客户端请求、查询分配、调度请求、暂停请求、流媒体程序，到与客户进行交互循环以便利用资源的异步操作。

为了解释两种对象（会接写事件和任务）之间的关系，我们说明客户端连接是如何处理的。当 DSS 接受来自客户端的连接时，将调度套接字事件并向 RTSP Listener Socket 任务对象发信号。如

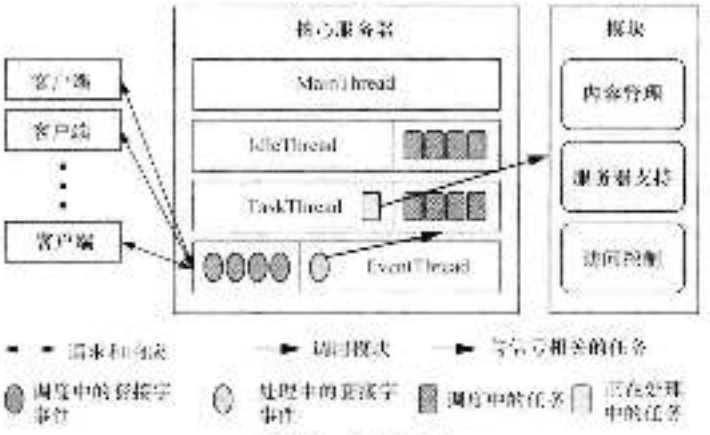


图 6-58 DSS 框图

果一切顺利，RTSP Listener Socket 任务对象将创建一个新的 RTSP Session 任务对象来处理这个 RTSP 会话。接下来，客户端可以发送一条 PLAY 命令请求媒体内容。处理此命令后，RTSPSession 任务对象可以创建一个新的 RTPSession 任务对象，然后将它以连续不断的流媒体内容返回给客户端。这两个任务对象将持续下去，直到客户端发送一个 TEARDOWN 命令来关闭这个 RTSP 会话为止。

模块可以静态编译也可以动态链接。有三种类型的模块：1）内容管理模块，管理与媒体源（如再储文件或直接）相关的 RTSP 请求和响应；2）服务器支持模块，它执行服务器数据收集和日志记录功能；3）访问控制模块，它提供身份认证和授权功能以及 URL 路径操作。当流服务器开始运行时，核心服务器加载和初始化这些模块。

数据结构

为了了解核心服务器是如何工作的，首先我们应该知道任务是什么。图 6-59 显示了 DSS 的重要对象类。类 Task 是可以调度的所有类的基类。一个任务就是一个对象实例，它直接或间接地从类 Task 中继承而来，因此可以在 TaskThread 的 fHeap 和 fTaskQueue 中调度。虽然 fTaskQueue 是一个 FIFO 队列，但在 fHeap 内的任务会根据其预期的唤醒时间被弹出。

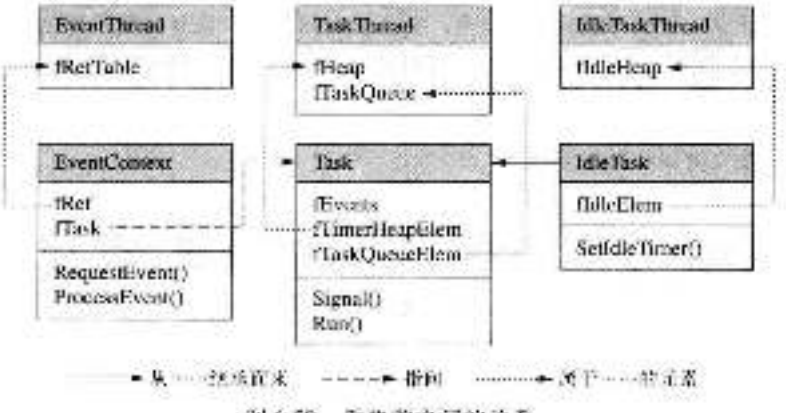


图 6-59 重要类之间的关系

利用在 fEvents 变量中特定事件的标记，Signal()用于调度任务对象进入到 Take Thread 的 fTaskQueue 中。当任务对象操作时，虚函数 Run()提供了一个被调用的通用接口。在一般情况下，Run()根据 fEvents 变量中标记的事件运行。

算法实现

任务处理

DSS 使用一组预先派生的线程支持这里提到的操作。这与 Apache 和 wu-ftpd 等在整个会话期间专用一个线程服务于客户端的其他服务器是不同的。DSS 的 Task 任务可以支持不同的线程之间的切换和调度。这种类似于操作系统设计的原因来自流媒体应用的长的会话生命周期。在这种情况下，少数几个线程就可以处理大量重叠的会话。

如图 6-60 所示，除了 MainThread 外，在核心服务器中还有三种类型的线程：1）EventThread（事件线程）；2）TaskThread（任务线程）；3）IdleTaskThread（空闲任务线程）。从 EventContext 类继承的对象将 fRef 注册到 EventThread 的 fRefTable 中。一旦一个客户端连接 DSS 或者发送一个命令给 DSS，EventThread 将获得会接写事件，然后从 fRefTable 找到关联的 EventContext，执行它的 ProcessEvent()以便发信号给由 fTask 指向的相关任务对象，对客户端做出响应。

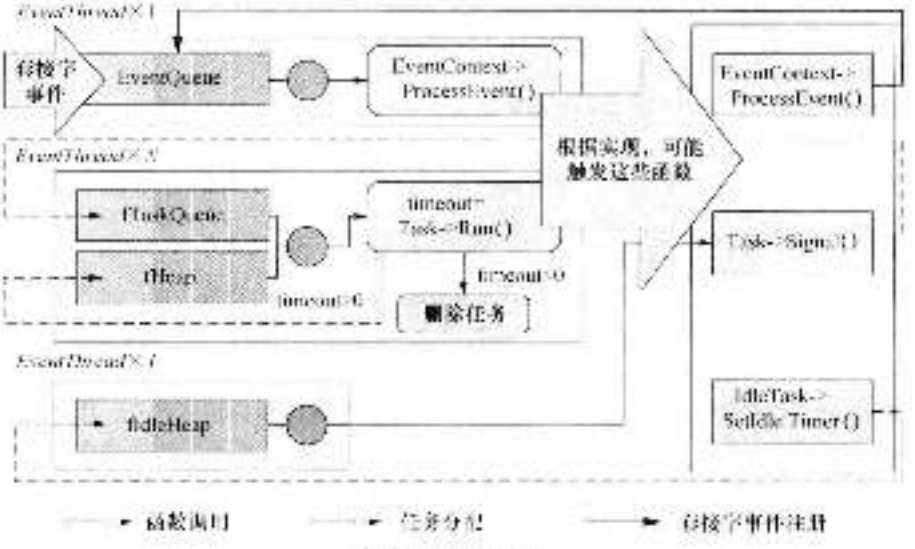


图 6-60 任务处理

当向任务对象发信号时，它会以循环的方式分配给 N 个 TaskThreads 之一并放入 fTaskQueue 中。TaskThread 将首先检查在 fHeap 中是否有睡眠时间已经到期的任务。如果没有，它将接着检查 fTaskQueue。一旦 TaskThread 得到一个任务，它就调用该任务的 Run()实现，处理在 fEvents 变量中标记过的事件。根据 Run()的返回值，任务将被删除或者放入 fHeap 稍后再进行处理。

一旦调用一个 IdleTask 对象的 SetIdleTimer，它也是一个任务对象，任务对象就被放入 fIdleHeap 等待睡眠时间到期。这类似于将任务对象放入 TaskThread 的 fHeap 中，但不同的是，从 fIdleHeap 弹出任务对象后，IdleTaskThread 什么也不做，而是发信号让任务对象再次获得调度。

根据 Run()和 ProcessEvent()的不同实现，可以调用如 RequestEvent()、Signal()和 SetIdleTimer()函数用于将被调度的任务。如何设计任务适合于像 DSS 这样的系统是程序员的另一个工作。

RTSP 会话处理

当 RTSPListenerSocket 对象接受一个连接时，它就创建一个 RTSPSession 对象，并使这个对象可调度。在 RTSPSession 类的 Run()实现的内部，有一个良好定义的状态机，它用来跟踪 RTSP 处理的进程状态。因为真正的状态过移图太过复杂难以在这里描述，所以我们将它简化成如图 6-61 所示

的图。从 Reading First Request 开始，如果 RTSP 会话的第一个请求是用于 HTTP 隧道，那么状态切换到 RTP Tunnel 以便处理 HTTP 隧道。

如果它是一个通用 RTSP 请求，那么状态将经过 Filtering Request 来解析请求。Routing Request 将请求路由到内容目录，Access Control 用于身份认证和授权，Processing Request 用于 RTP 会话建立和计费。所有上述这4个状态使用相关模块提供功能。在将响应返回给客户端并为处理该请求清理数据结构后，状态返回到 Reading Request 等待下一个 RTSP 请求。

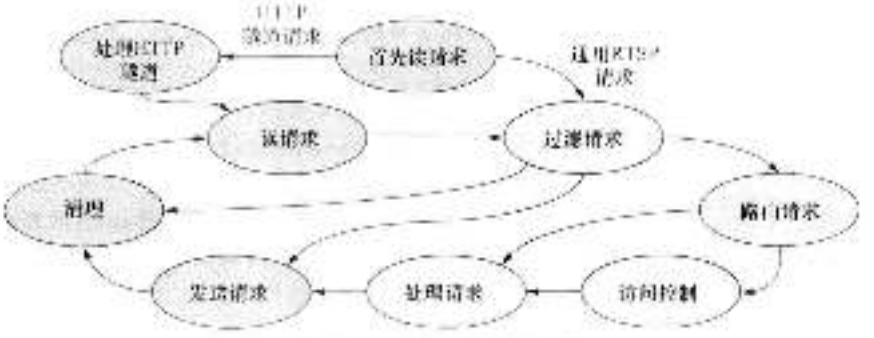


图6-61 RTSP 处理状态迁移图

练习

1. 找出在什么情况下，DSS 核心服务器会将 RTSPListenerSocket 对象放入 IdleTaskThread 的 IdleHeap 中等待。

2. 请参阅函数 Task::Signal()。解释将一个 Task 对象分配给 TaskThread 的步骤。

6.9 对等应用程序

在20世纪90年代，因为各种原因客户端/服务器模型被认为是互联网应用的可扩展解决方案。例如，用户的计算机在计算能力和存储方面是一个瘦终端。80-20 规则表明，大多数的网络流量专门用于检索最流行的网页。过去已经证明一个强大的服务器能够用于存储信息并以高效、稳定、可扩展的方式共享。然而，随着计算能力、网络带宽、个人计算机的硬盘存储器的快速增长，用户的计算机不再处于默默无闻的地位。此外，随着家庭宽带接入的盛行，更多的计算机充当总是待在互联网上的服务器。因此，近年来开发了更多基于对等（P2P）体系结构的互联网应用程序。这些应用程序不仅向互联网中引入一种新的通信模式，而且也是一种新的创造性思想和商业模式。值得注意的是，根据 CacheLogic 的报告，P2P 已经占据互联网流量的60%。

这里，我们从4个方面介绍 P2P 应用程序：1）P2P 操作的一般概述；2）多种 P2P 体系结构的评估；3）P2P 的性能问题；4）流行的 P2P 文件共享应用程序 Bit Torrent（BT）案例学习和它的开源实现。鉴于 P2P 的复杂性行为，与本章中其他章节相比建议读者对本节更为重视。

6.9.1 简介

与客户端/服务器模型不同，P2P 是一种分布式体系结构，其中的参与者既充当客户端又充当服务器。在 P2P 网络中的参与者通常是普通的用户计算机。基于某些 P2P 协议，它们可以在底层 IP 网络之上的应用系构建一个虚拟覆盖网络，如图6-62 所示。在覆盖网络中的网络节点是参与者，而一条重叠覆盖链路通常是一条在两个参与者之间的 TCP 连接。例如，在图6-62 中 P1 和 P2 之间的虚拟链路是一条通过底层 IP 网络中路由器 R1、R2、R3 的 TCP 连接。在 P2P 网络中的参与者称为对等，因为它们被认为扮演同等的角色，即作为资源的消费者又作为资源的生产者。对等能够共享它们自己的部分资源，如处理能力、数据文件、存储容量和网络连接容量，通过直接通信而不用经过中间节点。

通常，P2P 系统中的操作包括3个阶段：加入 P2P 覆盖网络、资源发现和资源检索。首先，一个对等通过某些加入步骤加入到 P2P 覆盖网络中。例如，一个对等发送一个加入网络的请求给它知道的一

个引导服务器，以便获得在该覆盖网络中已经存在的对等列表，当然也可以通过手动配置获得。加入P2P覆盖网络后，对等通常会试着在网络中搜索出其他对等共享的对象。如何在分布式网络中搜索对象是P2P应用的最大挑战。搜索算法可以基于集中式目录服务器、洪泛技术、或分布式散列表（DHT），这取决于底层的P2P体系结构。我们将介绍不同的P2P体系结构。如果搜索成功，对等将获得资源持有人的信息，如他们的IP地址。对资源共享对象简单，这像直接的TCP连接可以在搜索对等和资源持有人之间建立一样。然而，考虑到持有人的上传带宽、开发下载、持有人意外断线、一断对等后重新连接到互联网等如何恢复下载等因素，可能需要复杂的下载机制。

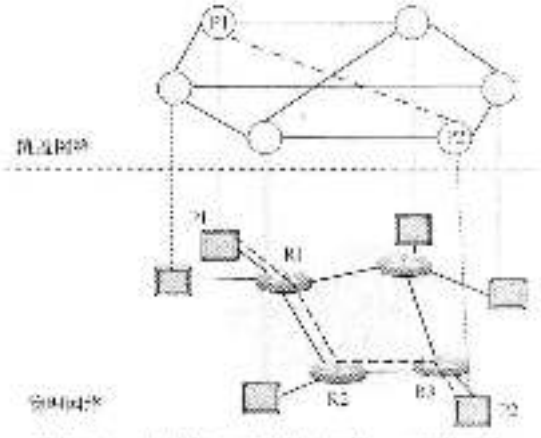


图6-62 在底层IP网络之上的P2P覆盖网络

历史演变：流行的P2P应用

除了文件共享外，P2P还有许多热门应用，诸如即时、音频、视频等P2P通信很流行。通过P2P构成很好的覆盖，用于合作和对资源的P2P计算也是如此。表6-29对流行的P2P应用进行了分类。一般来讲，P2P应用程序的整个操作过程可以分为P2P和非P2P部分。在P2P应用的初始阶段，参与者通常连接到某一预先配置好的服务器以获取更新的消息，这是传统的客户端/服务器关系。随后，参与者相互之间开始建立临时TCP连接以便交换信息，这就是P2P关系。

表6-29 对等应用的分类

类　别	应用名称	特　点
文件共享	Napster、Limewire、Gnutella、BitTorrent、eMule、Kazaa	·从其他地方搜索并下载共享文件 ·大的文件可以分成块 ·P2P通常为最大努力
IP电话	Skype	·在因特网上实时通话为常用用户 ·专业生Kazaa之上的P2P文件共享 ·用户专生消息和skype外出计费的服务者
流媒体	Freenet、Pioneer、Coolstreaming、PPLive、PPStream	·与聊天P2P文件共享之上的kazaa ·点播因特网电话 ·通过对等搜索并中继流
即时消息	MSN Messenger、Yahoo Messenger、AOL Instant Messenger、ICQ	·消息/音频/文件交换
协作计算	Microsoft GROOVE	·文档共享与合作 ·保持本地之间共享数据的更新 ·实时消息和聊天会话
网格计算	SETI@HOME	·科学计算 ·汇聚个体自愿的计算和搜索外星生命

历史演变：Web 2.0 社交网络：Facebook、Plurk 和 Twitter

与传统的 Web 1.0（其中内容和服务仅由服务器提供并且仅在服务器上进行信息才是可获的）相比，Web 2.0 支持客户端提供内容。服务器与对等点协作，以便包括协作百科、Web 服务、博客、微博和在线社区等。典型的 Web 2.0 应用程序，允许客户端通过电子邮件或即时消息与其他人进行交互，更新个人资料后通知其他人，或者修改网站的内容。Facebook、Plurk 和 Twitter 属于 Web 2.0 的社交网络服务，构建与朋友联系的虚拟社区，并连续创可信的推荐系统。此外，Plurk 和 Twitter 还提供微博服务，它允许发布的博客中包含的内容的大小，受限于所允许用一个简单句子，图片或者由 10 秒的短视频组成。表 6-30 总结了其特点。Facebook 流行，因为它丰富的功能和应用程序让它能够更轻松地与朋友交互。Plurk 和 Twitter 更为流行，因为它们能够实时地与朋友分享意见。

表 6-30 Facebook，Plurk 和 Twitter 的特点

应 用	服务类型	特 点
Facebook	社交网络	· 数以亿计的活动用户 · 超过 200 个能不用完整或专业知识 · 一种标记语言，Facebook 标记语言，新开发者用来定制应用 · 墙壁帖：一种为朋友精密消息的用户空间 · 戳：一种提醒提示（nudge）以便吸引其他人的注意 · 照片：一张相片 · 状态：通知朋友有关自己的行踪和行为 · 礼物：向朋友发送虚拟礼物 · 市场：搜索免费分类广告 · 活动：通知朋友有关重要事件 · 视频：共享家庭录制的视频 · 异步游戏：两用户可移动棋在走点并且下次移动可以在任何时候进行
Plurk	社交网络、微博	· 短消息（最多 140 个字符） · 以年月日次序列出短箱（称为 plurks） · 通过发消息来成短箱 · 朋友日历日志 · 基于文本的表情符号
Twitter	社交网络、微博	· 短消息（最多 140 个字符） · 把作者网页上的消息（称为在 Twitter 上发微博）发送给订阅者（又称粉丝） · 支持 SMS 发短

6.9.2 P2P 的体系结构

构造 P2P 覆盖网络的方式可以分为三类：集中式、分散和非结构式、这主与 P2P 应用的演变有关，集中式 P2P 是第一代，分散和非结构式 P2P 是第二代，分散和结构式 P2P 是第三代。P2P 覆盖网络的组织方式，被称为基础设施，影响其搜索操作和覆盖维护开销。

集中式

集中式方法利用一个中央目录服务器来定位 P2P 网络中的对象，如图 6-63 所示。中央目录服务器是一个被定的、始终运行的服务器，就像 WWW 或 FTP 服务器一样。对等可以通过先将自己注册到目录服务器来加入到 P2P 网络中。对等也会告知目录服务器共享的对象，如带有元数据的音乐文件列表。为了搜索对象，对等仅需要向中央目录服务器发送查询信息。搜索可以采取关键字搜索形式也可以采取元数据搜索，如歌名或歌手名字。因为所有的共享对象都已经注册到服务器列表中，所以搜索只需要在服务器上进行。如果搜索成功，就将匹配所有

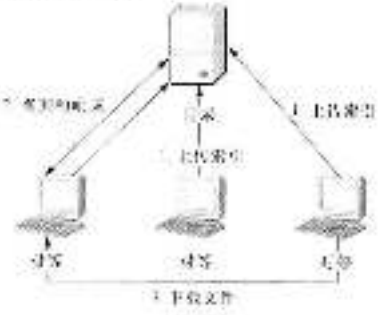


图 6-63 集中式 P2P

者列表组成的应答发送回给搜索者。查询者接着就会从列表中选择一个或更多对等直接下载对象。

这种方法被 Napster 采纳，它被认为是最近 P2P 开发的先驱。Napster 由 shawn Fanning 创建，这是一个通过集中式的目录服务器允许用户共享和交换音乐文件的程序。在第 1 个版本发布后就非常受欢迎。然而，它在 1999 年 12 月被美国录音工业协会（RIAA）以版权侵犯为由起诉。2000 年 7 月，法庭命令 Napster 关闭。后来 Napster 公司在 2002 年年末被 Bertelsman 收购。

集中式的方法非常简单，易于实现，并且可以支持各种搜索，如关键字、全文检索、元数据搜索。具有讽刺意味的是，它不是一个真正的 P2P 系统，因为它依赖于中央目录服务器。如果没有这个服务器，系统就无法正常工作。因此，它具有客户端/服务器模型的问题，如服务器会成为性能瓶颈，由于单点故障的不可靠性，不可扩展性和易于遭受 DoS 攻击。最重要的是，它对侵犯版权负有责任。

分散和非结构式

为了摆脱集中式目录服务器方法，分散和非结构式方法通过洪泛方法给同一个覆盖网络内的对等发送查询消息来搜索共享对象，如图 6-64 所示。为了减少由洪泛过程引起的开销，采用有限范围的洪泛方法，查询消息在经过一定跳数后不被转发。收到查询消息后，如果一个相邻的对等拥有和查询相匹配的资源，那么当前就将以查询命中消息回应前一个发送者，而不是图 6-64 中的原始查询者。如果查询不是重复的，并且没有达到其范围限制，那么就将查询消息转发给所有周边的对等；否则丢弃该消息。查询命中消息将沿着反向路径返回到请求者。接着请求者就可以直接从对象的所有者下载对象了。

为了加入到 P2P 网络，对等需要其种带外机制至少知道已经在覆盖网络上的一个对等。然后对等向已经在覆盖网络上的对等发送加入消息（或 ping 消息）。然后覆盖网上已存在的对等把自己的 ID 以及其相邻对等的清单发送给请求者。它也可能将请求消息转发给一个或所有相邻的对等。一旦收到加入应答消息，新加入的对等就知道该覆盖网络上更多的对等，并开始与选定的将成为其邻居的对等建立 TCP 连接。

这种方法的优点在于它是完全分散的，对对等的故障具有健壮性，并且也很难关闭。然而，因为洪泛方法会产生过度的查询流量，所以显然它不具备可扩展性。在有限范围洪泛的情况下，另一个关键问题出现了，即它可能偶尔无法找到实际上已经存在于系统中的共享对象。Gnutella 的第 1 个版本就是这样的例子。为了解决可扩展性问题，FastTrack，Kazaa 网络的专用标设，以及 Gnutella 后来的版本采用了分层的覆盖，如图 6-65 所示。

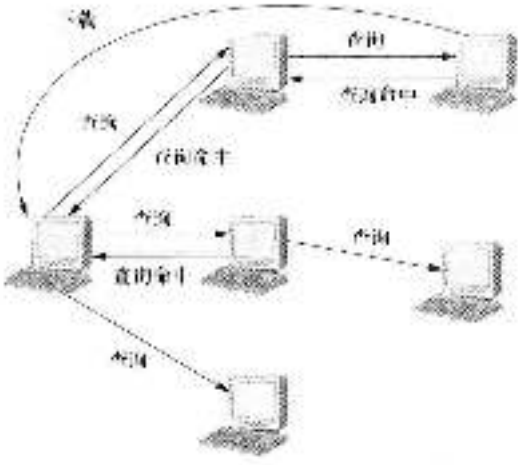


图 6-64 在分散和非结构式 P2P 系统中的有限范围内的查询洪泛

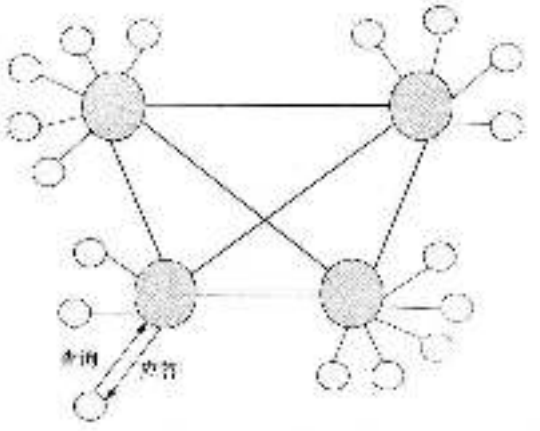


图 6-65 带有超级对等的层次化覆盖

层次化的重叠把对等分为普通对等和超级对等。当一个对等首先加入覆盖网络时，它充当普通对等并至少与一个超级对等连接。以后它可能被选为超级对等，如果它在该网络停留很长一段时间或者具有较高的上传带宽。超级对等作为本地目录数据库为低级对等存储共享对象的索引。为了搜索数据

对象。普通对等会给其相邻的超级对等发送查询消息。如果超级对等可以在本地目录中发现共享对象，就可能回复普通对等的查询请求；否则，它使用范围有限的洪泛查询广播到邻近的超级对等。因此，这种方法建立了两个层次的分层覆盖网络。底层网络采用中央目录服务器的方法，而上层网络采用分散非结构式的方法。

分散但结构式

Napster 和 Gnutella 都没有将他们的对等组织成一种结构化的覆盖网络。Napster 中的集中式目录是不可扩展的，而在 Gnutella 中传播查询的方式是相当随机的，因此也不是很有效。因此，一种更好的方法是将分布式目录服务和一个有效的查询路由方案相结合，从而导致分散和结构式 P2P 系统的开发，如 Chord，CAN 和 Pastry。

这种方法的主要思想是：对于分布式的目录服务，散列函数将对等和对象映射到相同的地址空间中，因此对象就可以以分布式方法确定分配给对等。为了有效地查询路由，根据对等在地址空间中的位置将对等组织成一种结构化的覆盖（网络）。散列函数将一组对等和对象均匀地散列到地址空间中，称为一致性散列。散列函数分布式地运行在每个对等中，因此这种方法也称为分布式散列表（DHT）。下面将概述 DHT 系统的操作并使用 Chord 作为例子。

如前所述，将所有的对等和对象散列到相同的地址空间中。为了避免冲突，地址空间应足够大，如 128 位。对等可能使用其 IP 地址或其他身份作为到散列函数的输入，并将获得的散列结果作为其节点 ID。同样，对等可能通过将对象的文件名或其他形式的 URI 输入到散列函数，从而获得一个对象的对象 ID。由于节点 ID 和对象 ID 共享相同的地址空间，所以主要思想就是让每个对等能为那些对象 ID 与其节点 ID 相同的对象存储目录服务。

基于这种思想，每个节点通过指定义的散列函数首先生成其节点 ID。然后对于每个保留的并且将被共享的对象，通过同一个或另一个散列函数生成对象 ID。对于每个对象，对等将注册信息发送给节点 ID 和对象 ID 相同的节点。如果对等要查询对象，它就使用散列函数生成对象 ID 并将查询信息发送给保留该对象 ID 的节点。我们假定有一个有效的路由机制传送查询消息。如果地址空间被对等和对象占满了，那么某些节点的节点 ID 就会与对象 ID 相同。但是我们期望对等和对象稀疏地占据地址空间，不存在节点 ID 与对象 ID 相同的对等。为了解决这个问题，将对象 ID 的注册信息路由到最接近对象 ID 的节点 ID 的对等。查询信息也以同样的方式处理。以这种方式，对等就能为对象所具有的 ID 靠近其节点 ID 的对象提供目录服务了。

问题是如何将消息路由到距离目的 ID（结构化覆盖网络中对象或对等的 ID）最近对等的节点 ID 上。关键是让每个对等维护一个专门设计的路由表，以便让每一个对等都能将到达的消息转发给更接近目的地的具有节点 ID 的邻居对等上。让我们用 Chord 作为例子来解释如何设置路由表来实现有效的路由。Chord 把它的地址空间视为一个一维的圆形空间，所以空间中的对等形成一个环形覆盖。

图 6-66 显示了一个包括 10 个节点的 Chord 覆盖在一个 6 位地址空间的例子。在 Chord 中的路由表称为一个查询表。对于一个 m 位的地址空间，ID 为 n 节点的查询表至多包含 m 项并且第 i 个表项指向紧跟在 ID 为 $n+2^{i-1}$ 对 2^m 取模的第一个节点 ID，其中 $1 \leqslant i \leqslant m$。让我们考虑图 6-66 中节点 N8 的查询表，其中 $m=6$。在这个例子中，节点 ID 范围从 N0～N63，但只有 10 个节点确实存在环上存在。

每个节点负责为大于前一个节点的 ID 但是小于或者等于它自己 ID 的对象提供目录服务。例如，节点 N15 将保留从 9～15 之间 ID 的对象信息。记住这一点，接下来我们分析 N8 表项的查询表。第一个表项，$i=1$，保留指向紧跟 N9 节点的下一跳信息。这个表项指向第一个节点，其 ID 大于或等于 9，这就是节点 N15。也就是说，如果有一个关于对象 ID 为 9 的请求消息，那么消息就传输给 N15，它实际上为这个对象提供目录服务。让我们使用最后的表项，$i=6$，作为另外一个例子。最后的表项将指向为 ID 为 $8+32=40$ 的对象提供目录服务的节点。表项指向 N42，负责 ID 在 31～42 之间的对象。

现在，我们考虑从 N8 将消息路由给对象 54 的情况，如图 6-67 所示。为了路由消息，节点从它的查询表中查找最后一个表项，其 ID 号小于对象 ID。因此，N8 查找最后一个表项（ID = 40 < 54），并将消息转发给 N42。从 42～54 的距离是 $12<2^3$，因此，N42 查找第四个表项（ID = 50 < 54）并将消息转发给 N51。最后，从 51～54 的距离是 $3<2^2$，因此 N51 查找第二个表项（ID = 53 < 54）并将消息转

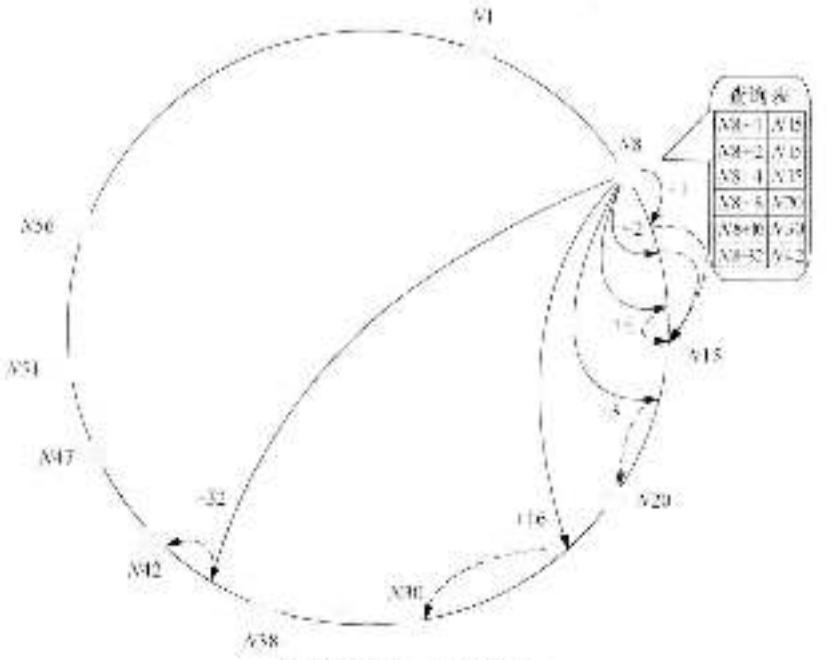


图 6-66 Chord 的查询表

发给 N56，由于 N56 负责对象 54 的目录服务，所以一旦收到查询，它就会回复查询消息。一个有趣的问题是转发消息需要多少跳。答案是，它是小于 $O(\log_2 n)$。一个直观的理由是，每个路由跳将到达目的 ID 的距离减少了 2 倍。例如，从 8～54 的距离是 46，二进制表示为 101110。因此，在 N8 选择 2^5 的表项，将该消息转发给一个 ID 大于或等于 40（8+32）的节点，它到节点 54 的距离小于 25（46/2）。换句话说，利用查询表，Chord 在每个路由的步骤就能够将搜索空间减少 2 倍。

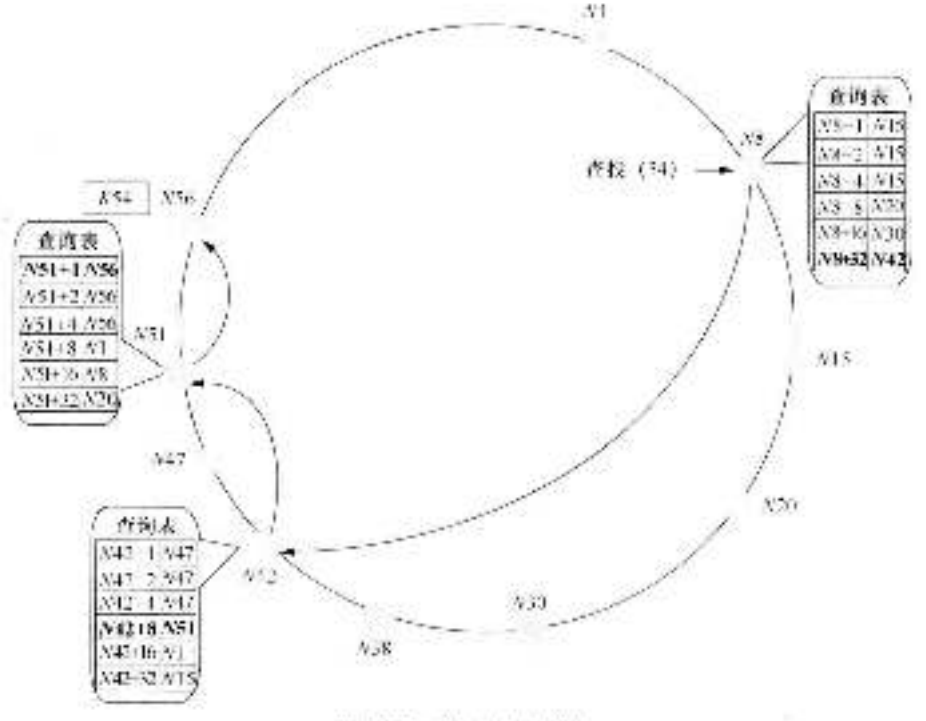


图 6-67 Chord 中的路由

已经提出了许多基于 DHT 的 P2P 系统的巧妙设计。然而，它们是建立在基于结构化覆盖的分布式目录服务和高效路由的相同思想上。虽然它们是分散的和高效的，但 DHT 的主要缺点是搜索限制为精确匹配。前面讲过通过散列处理对象的名字获得对象 ID。名字的微小差别将导致散列结果很大的差异。因此，在 DHT 中通过关键字、语义搜索、全文搜索进行搜索就很困难。DHT 的另一个缺点是覆盖构建和维护所需要的额外开销。

其他原因

因为不同的基础设施都有自己的优点和缺点，所以在文献中已经提出了多种混合模式和层次化的基础设施。

6.9.3 P2P 应用的性能问题

P2P 应用有一些性能问题也引起了研究人员的高度重视。下面讨论一些主要问题。

搭便车问题

P2P 系统的可扩展性依赖于来自对等的贡献。如果对等节点只消耗但贡献很少或根本没有共享资源，那么它就变成了系统的搭便车者。如果在系统中有许多搭便车者，那么该系统将退化为客户端/服务器模型，其中大部分的搭便车者充当客户端，而只有少数的非搭便车者充当提供绝大部分资源的服务器。如果 P2P 系统不采取某些机制来防止这种情况，就会出现严重的问题。2005 年 Hughes、Gaulson 和 Walkerdine 的研究结果表明，没有任何反搭便车机制时，85% 的对等在 Gnutella 中不共享文件。对搭便车的问题的一个共同的解决方案是要实施一些激励机制。例如，我们接下来会学习 BitTorrent 中的针锋相对机制，对于贡献更高上载速度的对等将给予更高的上传优先级。其他解决方案，文献中推荐了基于奖励的和基于信用的机制等。

瞬间拥挤

瞬间拥挤现象是指对某一特定对象需求的突然的、意外的增长，例如，新发布的 DVD 视频或 MP3 文件。与这一现象有关的问题包括如何应对突然大量出现的查询消息以及需要多长时间找到并在很短时间内下载的对象。虽然不同类型的 P2P 基础设施需要不同的解决方案，但一般情况下，在已经转发应答消息的对等上缓存对象的索引可以减少查询的流量和查询消息的延迟。另一方面，将对象尽可能多地复制到对等可以提高下载速度。例如，当一个对等节点已完全下载文件后，它将成为一颗种子，即一个资源提供者。

拓扑感知

基于 DHT 的 P2P 系统可以保证路由路径长度上的上限。然而，路径上的一条学路对应于底层物理网络中的一条传输层连接，如图 6-62 所示。这种虚拟学路既可以是跨洲的一条长的端到端的连接，也可以是一条短的局域网络的连接。换句话说，如果对等选择它们的覆盖邻居而不考虑底层的物理拓扑结构，那么最终产生的 P2P 覆盖网络可能产生与底层物理网络拓扑结构严重不匹配问题。因此，如何执行拓扑感知的覆盖构造和覆盖路由将大大影响 P2P 系统的性能。人们已经为 P2P 覆盖系统提出了许多种路由接近、邻居接近的改进，它们是基于 BTT 的测量、路由域或 ISP 的偏好或地理信息。

NAT 跨越

如果目的对等有一个公共 IP 地址时，一个对等就可以直接建立一条到另一对等的传输层连接。然而，许多宽带接入用户是通过 NAT 设备连接到互联网。如果两个对等都在 NAT 设备之后，没有其他对等或 STUN 服务器的帮助它们就不可能相互连接，如第 4 章中讨论的那样。因此，对 P2P 系统的基本需求就是为对等提供 NAT 跨越机制。在大多数情况下，NAT 跨越是通过具有公共 IP 地址的中继对等或超级对等来解决的。

搅动

搅动是指对等随意地动态加入或离开系统。直观地说，高搅动率严重影响了 P2P 系统的稳定性和可扩展性。例如，高搅动率可能会造成基于 DHT 系统的巨大覆盖维护开销和路由性能（包括路由正确性）的急剧恶化。为了对付搅动现象，P2P 系统应避免对等之间的刚性结构或关系，如 P2P 视频流中的树形结构，并且对等应该维护一张潜在邻居的列表，在需要时用于快速动态邻居的更换。

安全

在 P2P 系统中存在多个安全问题。P2P 程序带有后门（特洛伊木马）、虚假的内容，或不应该共享文件的泄露。在 P2P 系统中虚假内容或内容污染问题会降低内容的可用性并增加冗余流量。例如，恶意用户可能共享一首流行的 MP3 文件，但修改（污染）了部分内容。下载这个被污染文件的用户通常会尝试再从其他来源下载该文件。如果污染的内容（对象）遍布 P2P 系统，那么用户就可能失去加入到这个 P2P 系统中的兴趣，因为大多数下载的对象是无用的。内容污染问题的解决方案包括：利用消息摘要（如 MD5）、对等信誉系统和对象信誉系统。例如，在 BitTorrent 中，将共享文件的每一段 MD5 求要存储在元数据文件中，例如 .torrent 文件。在 FastTrack 中，UUHash 机制使用 MD5 散列选择的文件块防止文件污染。

版权侵犯

最后，应当指出的是通过 P2P 系统共享受到版权保护的对象会成为一个严重的问题，它限制了 P2P 系统的推广和使用。许多大学和组织禁止他们的用户运行 P2P 应用程序。此外，不仅是 P2P 用户应该为版权侵犯负责任，容纳 P2P 应用的公司，无论是没有自己的服务器就不可能让 P2P 系统存在的公司也要负有责任，就像 Napster 的情况一样。

6.9.4 案例研究：BitTorrent

BitTorrent（BT）最初由 Bram Cohen 在 2001 年设计，目前它已经成为一个非常流行的文件共享软件。2004 年，它贡献了大约 30% 的互联网流量。虽然目前有多个竞争对手，如 eDonkey 和 eMule，但它仍然是一个非常流行的文件共享软件。BT 是一个经过深思熟虑的、具有多个独特功能的协议：1）利用计算相对作为一种激励机制，处理搭便车问题；2）使用带外搜索避免版权侵犯问题；3）使用基于拍的群群技术实现负载平衡；4）使用散列检查，以防止虚假片的传播；5）一个对等成功下载一个文件后，它就变成了发布文件的种子。

在描述协议之前，我们首先介绍在 BT 中使用的术语。一个将共享的文件切成固定大小的片。每一片进一步划分成块，它对等请求内容的基本数据单位。每一片的完整性受到 SHA-1 散列代码的保护，因此受污染的片不会传播。如果对等已经成功地下载了文件，那么它就变成了种子。对于将被共享的每一个文件或一组文件，都有一个跟踪者（tracker）。跟踪者跟踪下载的对等和种子，并协调对等之间的文件分发。虽然自 2005 年以来在 BT 系统中每个对等都有可能充当跟踪者，但更常用集中式跟踪器。因此，我们将学习含有集中式跟踪器的 BT 协议。

操作概述

图 6-68 是有关 BT 操作的简要概述。为了共享文件，一个对等首先创建一个 ".torrent" 文件，它

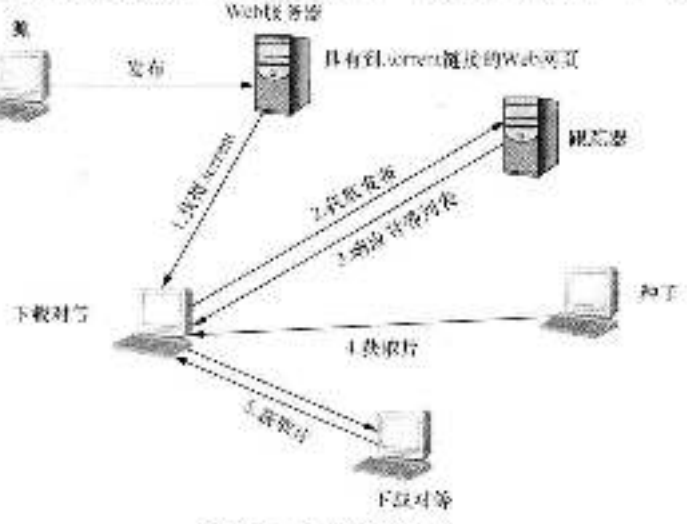


图 6-68 BT 的操作步骤

包括将被共享文件的元数据、文件名、文件长度、正在使用的片长、每一片的 SHA-1 散列代码、每一片的状态信息和跟踪器的 URL。通常将 torrent 文件发布到一些知名网站上。为了查找并下载某个文件，用户首先浏览 Web 网页以便首先找到该文件的 torrent。然后用 BT 客户端程序打开 torrent 文件。客户端连接到跟踪器，并得到当前正在下载文件的对等清单。在此之后，客户端便连接到这些对等，按片选择算法获得文件的各个片。

片的选择

对于前几片（通常约为 4），客户端只是随机选择一片下载，称为随机第一片选择。经过初始阶段后，采用最稀缺优先策略。最稀缺优先策略选择最稀缺的一片下载，因为最稀缺的片稍后可能会因为某些对等的离开而不再提供。它还能保证从种子下载各种片。最后，为了加快最后文件下载的完成，仅有少数片缺失的对等将进入一种残局（end-game）模式，并向所有对等发送请求以便得到缺失的片。

对等选择

对等可能会收到来自其他对等的对片的请求。BT 采用一种内置激励算法，称为针锋相对（tit-for-tat）算法，选择对等来上传它们感兴趣的片。针锋相对是博弈论中因使困境最普遍采用的策略。其基本思路是，如果对手合作；那么代理以合作；否则，如果对手挑衅，代理将采取报复行动。对等选择算法由三部件组成：阻塞/疏通（非阻塞），乐观疏通（非阻塞），防冷落。

针锋相对是阻塞/疏通算法所采用的。阻塞是指暂时拒绝上传给对等。在开始的时候，所有的对等都阻塞。然后对等疏通基于固定数字（通常为 4）的对等，其中一些（通常为 3）是基于针锋相对的，而另一些（通常为 1）是基于乐观疏通的。在那些对对等片感兴趣的对等中，针锋相对算法会从对等下载最多的对等中选择一个固定个数（通常为 3）的对等。具体来说，选择完全基于对每片的下载速度。针锋相对算法每隔 10 秒就停用，并且下载速度基于 20 秒的移动窗口速率计算。但是，当新的对等节点最初加入该系统时，它需要迈开它的第一步，但需要移动第一步去探索目前还没有合作过的对等。因此，乐观疏通思想就是从对对等的片感兴趣的对等中随机选择一个，而不管其下载速率。乐观疏通每 30 秒进行一次以便选择按环形排序的对等。最后，当一个对等被其所有的对等阻塞（受到冷落，例如，在 60 秒内时它没有收到任何数据）时，就运行防冷落算法。一个受到冷落的对等的前更经常运行乐观疏通来探索更多愿意合作的对等。因此，防冷落算法停止上传由针锋相对选择的对等，以便让乐观疏通可以更经常地运行。

开源实现 6.8：BitTorrent

概述

有多个用于文件共享网络的免费客户端软件程序，如 Gnutella 的 LimeWire、eDonkey 的 eMule、BitTorrent 的 uTorrent 和 Azureus。它们以不同的设计理念和基础设施分别解决不同的 P2P 系统的性能问题。例如，Gnutella 采用分散和非结构化拓扑，后来又采用超级对等的层次化结构。eMule（电驴）和 BT 采用 DHT 技术，以避免集中式跟踪器（服务器）。因此，Gnutella 以其分散化、无服务器拓扑而著名，对随机节点故障具有最极大的弹性。电驴以其分布式跟踪器而著名，是一个基于 DHT 解决方案的 Kademlia 协议。BT 以其将大文件分成片而著名，采取针锋相对算法以应付搭便车问题，利用文件的完整性检查以防止传播污染的片。由于 BT 有这么多独特的功能，所以它仍然是最流行的 P2P 文件共享软件程序之一。首面讲过，为处理性能问题，BT 采取如下的解决方案：

1）采用针锋相对算法以避免搭便车。BT 根据两个对等之间的下载速度来实现针锋相对算法。使用下载速率作为回报标准的优点若干，它可以根容易地利用每个对等上的本地信息实现。替代的解决方案，如基于信誉的方法和到所有对等的下载速度，都需要来自其他对等的信息，而这些信息是否正确仍然是一个问题。另一方面，BT 的方法对于新的到来者是不公平的。

2）带外搜索避免版权侵犯。BT 假定对等能够首先找到 torrent 文件而不用指定如何找到。这种方法是一个简单而有效的方式，以控制版权侵权责任。缺点是依赖第三方服务器发布 torrent 文件。

3）用于负载平衡的基于拉的蜂群。基于针锋相对算法，为了下载它们需要的片对等将片上传到其他对等。这种方法非常有效地强制对等首先找出它们已经下载的片。因此，随着更多的对等加入到该系

统中，对等可以加快下载过程。这种方法的一个潜在的问题是，对等一旦完成下载就会离开系统。这就是所谓的搭便车问题。原因，种子停在网上的时间越长，全部性能就越好。

4) 为点播模式保护每个片的完整性　BT 采用 SHA-1 保护每个片的完整性。虽然这种方法可以有效防止污染片的传播，但它需要对每一个收到片进行 SHA-1 计算。在 FastTrack（Kazaa）中，为点播模式只基于一个文件的部分缓，这样做会节省一些计算开销，但它也将让攻击者污染一个文件后不会被检到。

由于 BitTorrent 的协议规范是免费使用的，所以许多 BT 客户端程序是开源的。其中，uTorrent，Vuze 和 BitComet 是最流行的客户端程序。在本节中，我们将给 Vuze4.2.0.2，这是用 Java 语言实现的。

文件和数据结构

大多数 Vuze 的核心软件包都位于 \com\aelitis\azureus\core 目录下。在该目录下可以找到的软件包，如表 6-31 所示。

表 6-31　在 com.aelitis.azureus.core 中的软件包

软件包		软件包		软件包	
软件包	clientmessageservice	软件包	impl	软件包	peer
软件包	custwork	软件包	instancemanager	软件包	security
软件包	content	软件包	ipfs	软件包	speedmanager
软件包	crypto	软件包	messenger	软件包	stats
软件包	custom	软件包	networkmanager	软件包	subs
软件包	devices	软件包	nonblocking	软件包	torrent
软件包	dht	软件包	nat	软件包	update
软件包	dht.transport	软件包	networkmanager	软件包	util
软件包	download	软件包	vivaldi	软件包	versioncheck
软件包	dum	软件包	peer	软件包	vuzefile
软件包	helpers	软件包	peermanager		

对等选择和片选择的大多数代码都位于 \com\aelitis\azureus\core\peermanager 目录下。在该目录下，对等选择和片选择算法可以分别在\piecepicker 和\uncholker 目录下找到，也连续对等的状态信息代码可以在\peerdb 目录中找到。

另一个重要的目录是\org\gudy\azureus2\core3 目录。控制片和对等选择的主程序是 PEPeerControlImpl.java，它位于该目录的\peer\impl\control 下。对等和片对象类为 PEPeer 和 PEPiece，定义在\org\gudy\azureus2\core3\peer 下。

图 6-69 显示了 PEPeer，PEPiece 和 PEPeerManager 的类层次化结构。

算法实现

主程序

控制片和对等选择的主程序是 PEPeerControlImpl 类，它从两个类 PEPeerManager 和 PEPeerControl 中继承而来。ePeerControlImpl 的详细继承图显示在图 6.70 中。该类的构造器创建对象 oPiecePicker。函数也定义在该类中。schedule() 调用 checkRequests() 和 piecePicker.doOptimisticDisconnect()。如果对等不处于种子模式，那么 allocateRequests() 调用片请求。然后它调用 doUnchokes() 来处理对等连接和断开。在 doUnchokes() 中，调用 unchoker.calculateUnchokes() 确定哪个对等点通（unchoke）。

对等选择的实现

下载对等和种子对等的疏通算法在 \com\aelitis\azureus\core\peermanager\uncholker 目录下的 DownloadingUnchoker.java 和 SeedingUnchoker.java 中实现。我们将研究针对下载对等的疏通算法的代码实现。计算相对的主函数是在 calculateUnchokes() 中实现。有 4 个对等列表用在该函数中：chokes，

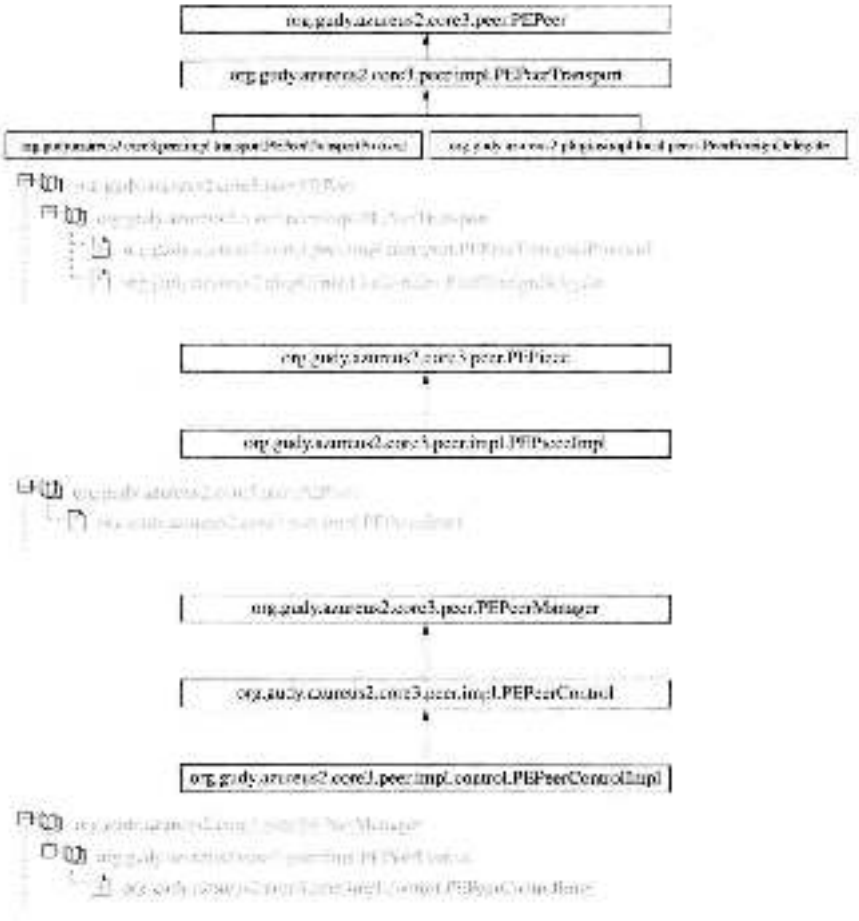


图6-69 PEPeer、PEPiece和PEPeerManager的类层次化结构

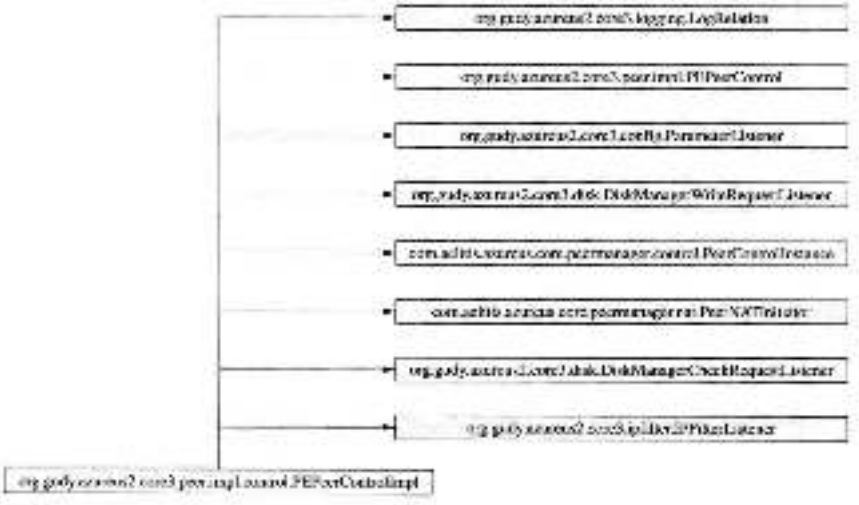


图6-70 PEPeerControlImpl的继承关系

unchokes、optimistic_unchokes和best peer。它们用来管理阻塞的、畅通的、乐观畅通的对等，以及根据下载速率有效畅通的最佳对等。calculateUnchokes()的实现如下：

```
    calculateUnchoke()
BEGIN
    get all the currently unchoked peers;
    IF the peer is not previously choked by me {
        IF the peer is unchokable {
            add it to the unchokes list;
            IF the peer is previously optimistic unchoked
                add it to the optimistic_unchokes list;
        }
        ELSE
            add the peer to the chokes list;
    }
    IF not forced to refresh the optimistic unchoke peers
      Move the peers in the optimistic_unchokes list to the
best_peers list until the number of peers exceeds max_optimistic;
      Add peers to the best_peers list if its download rate is higher than 256;
      Call UnchokerUtil.updateLargestValueFirstSort to sort
the best_peers list according to the download rate;
    IF we still have not enough peers in the best_peers list
        (less than max_to_unchoke) {
        fill the remaining slots with peers that we have downloaded
            from in the past (uploaded_ratio < 3);
    }
    IF we still have remaining slots
      Call UnchokerUtil.getNextOptimisticPeer to get more optimistic
          unchoke peers. (factor_recalculated is set to true);
    Call chokes.add() to update chokes
    Call unchokes.add() to update unchokes
  END
```

在这个函数中，对等根据当前的状态首先放入到 chokes、unchokes 或 optimistic unchokes 列表中。步骤就比较复杂了，直到优化疏通的数量超过了 max optimistic 阈值为止。然后，对片感兴趣的对等（peer.isInterestingOn()）并疏通的对等（UnchokerUtil.isUnchokable()），调用方法 peer.getStats().getSmoothDataReceiveRate() 以获得它们的下载速率。这些对等通过调用 UnchokerUtil.updateLargestValueFirstSort() 按照速率排序到 best_peers 列表中。如果在 best_peers 列表中的对等数量小于将被疏通的最大对等数 max_to_unchoke，那么将 uploaded_ratio 小于 3 的对等添加到 best_peers 列表中，这里 uploaded_ratio 是发送的总字节数与接收的总的字节数（加上 BLOCK_SIZE - 1）之比。如果 best_peers 的大小仍然小于 max_to_unchoke，那么就调用 UnchokerUtil.getNextOptimisticPeer() 以便找到更多用于乐观疏通的对等。当选择乐观对等时，UnchokerUtil.getNextOptimisticPeer() 函数或者考虑对等回报率，或者只随机地从 optimistic unchokes 列表中选择对等，具体根据 factor_recalculated 是否为真来决定。按各得分定义为总的发送字节数与总的接收数据字节数之差，低速优待分数。

片选择的实现

getRequestCandidate() 方法定义在 /com/bulitis/azureus/core/peermanager/piecepicker/impl 目录下的 PiecePickerImpl.java 中，是决定下载哪个片的核心方法。需要首先知道两个参数：优先权和数量。优先权定义查片的集合优先级，这只是给查片的范围在预定区间内的可用性等级。在这个方法中有三个阶段。第一，如果有一个 FORCED_PIECE 或一个预留的片，如果可能，它将被启动/恢复。第二，通过扫描所有活动的片找到最好的具有最高优先权的已经被请求的并且可能继续下载的片。片的可用性由 availability[] 表示。第三，如果没有要恢复的片，那么就找到一个具有最高优先级的最稀有片列表作为保证有开始下载一个新的片。如果所做的请求被接受，那么方法返回 int[] pieceNumber 和 blockNumber。

练习

1. 通过考虑对疏远对等且相应地更改 getNextOptimisticPeer() 函数中的随机选择代码研究为

练习。例如，你可能会研究考虑过度延迟确认的对等。

2. 讨论为什么在选择采取激进对等中考虑局部性很重要。需要注意的是，系统能够发现潜在的下载对等时会发生哪些变化的角色。

6.10 总结

与本书的其他章不同，本章主要目的仍然是获得互联网应用一般知识所必要的。本章先从所有互联网应用设计的一般问题开始。我们学习了众所周知端口的工作方式、服务器如何作为后台守护进程运行、并发和面向连接组合服务和决优无连接服务之间的区别，以及为什么应用层协议有可变长的ASCII消息和有状态/无状态。然后我们介绍了主要的应用层协议，从基本的DNS，到经典的SMTP、POP3、IMAP4、HTTP、FTP和SNMP，再到实时的SIP、RTP、RTCP、RTSP和各种P2P协议。对每个应用协议，我们描述了设计概念、协议消息和行为，需要时还给出了例子会话和一个流行的开源软件包。这里我们不打算总结每个应用的设计概念。相反，我们重新审查它们通用的特点：众所周知端口、变长ASCII码、无状态的和并发性。通过这些，我们就可以更好地体会这些特征并很可能能够发现有缺陷。

第一，根据端口号对应用进行分类已经不合适了。许多应用的运行通过端口80或者将它们的信息封装在HTTP消息中以便允许Web流量通过防火墙。此外，P2P应用经常动态地选择不是众所周知端口的端口号。因此，为了精确的分类就需要深度分组检测（deep packet inspection，DPI）应用的头部或者有效载荷。第二，与无连接的、进程固定长度协议头部不同，应用层协议具有可变长的ASCII格式。这使不能使用由单字段（目的地IP地址）分组转发和多字段（5元组）分类的表查询。相反，需要带有正则表达式解析或基于特征的字标者匹配的DPI用于分类或安全目的。这种固定到可变之间的共鸣在数据库系统领域中有着类似的例子。传统的关系数据库系统具有固定长度的表。而与此相对以XML等式的结构化的数据或由搜索引擎引擎的非结构化数据就是可变长度字符串。就像半结构化或无结构的数据库中所占的百分比不断增长一样，可变长度协议消息的处理在网络世界中也获得更多的关注。

第三和第四，无状态是协议的设计选择，而并发是协议服务器的一种实现决策。除了HTTP和SNMP为了效率和扩展性外，为了跟踪客户端的连接太多数应用协议选择有状态的设计。DNS介于两者之间。本地DNS服务器大多数是有状态的和递归的对DNS的查询完全负责，由于效率和可扩展性所在其他的DNS服务器是无状态的、迭代的。尽管本质上是无状态的，但HTTP服务器可以通过cookies机制将长的重新变成有状态的。对于并发，决定取决于服务会话或请求所需要的时间。如果服务时间很短，服务器就可以保持迭代。SNMP就属于这种，因此在net-snmp中的服务器就是以迭代服务器实现的。本章中的所有其他开源软件包具有并发服务器的实现，由于它们较长的服务时间。

从第2章到第6章，我们学习了所有协议层。有两个高级问题需要持续处理：服务质量（QoS）和网络安全。一旦我们实现了连通性，就希望连接足够快、足够安全。服务质量或性能问题一直是所有网络系统或组件设计的中心问题。在第7章中，利用两个完全的解决方案（IntServ和DiffServ），以及6个重要的构建模块形式化地处理服务质量。尽管这两种解决方法都没有成功，但某些构建模块技术已在我们日常互联网生活中普及起来。在第8章中，我们将安全问题分为访问安全、资源安全和系统安全，分别解决谁可以访问什么、在公共互联网上的专用数据和对入侵者来讲的系统漏洞。对这些问题的最新解决方案进行了介绍。

常见陷阱

服务器的并发替代方案

编写一个并发的、面向连接服务器的最简单方法就是按需派生一个子进程，该子进程服务于一个新的接收客户端连接。这种方法在开源实现6.4介绍的wu-ftp中得到实现。但还有许多其他考虑了开销、还没有可扩展性问题的替代方法来实现这种并发。派生一个进程是昂贵的，因为它涉及在进程表中创建一个新的表项、为进程体分配内存空间。从父进程体向子进程体复制。一种很开销的替代方法是线程，这里线程是使用与其父线程共享的进程体创建的，因此没有内存分配或复制。开源实现6.6

中的 Asterisk 就属于这一类。另一方面，在服务到达客户端时就生成新进程会引入延迟。带有空闲进程或线程的预先生成或按生成线程分发都完全会减少这种启动延迟。可以周期性地监控该池，将其大小保持在高阈值和低阈值之间。在开源实现 6.3 中介绍的 Apache 就运行此方案。

最后，一个更难的问题是，服务器处理成千上万的并发连接上的可扩展性问题。这经常发生在负载服务器上，这种服务器处在客户端和服务器之间。在一台服务器上维护成千上万的运行或线程上下文的代价是不可行的。有两种常见的解决方案：为有 I/O 多路复用的单运行或服务于更大量连接并在更小量运行或线程之间切换，以减少开销。或为单个连接的进程或线程。前者通过 select() 函数在单进程中进行 I/O 复用，监听每个连接套接字的数据并用高效的分拆处理这些套接字。Squid 就产生一种这样运行的开源代理。后者在整个连接生命周期内调度并连接进程和线程池之间的连接。开源实现 6.1、6.2、6.7 中介绍的 BIND、qmail 和 Darwin 运行此种决方案。

DNS 查询：递归或迭代

当我们发一个 DNS 查询时，我们希望有单个 DNS 服务器都是递归的和有状态的。事实上，大多数 DNS 服务器是递归的和有状态的。在 DNS 层次结构中的所有其他的 DNS 服务器是迭代的和无状态的。主要是缓存。当用户发来或者更高向从本地 DNS 服务器来的查询，如下将查询转发给其他的 DNS 服务器上。原因是对本负载尤其是非常高层次结构中高粘接方的服务器的可扩展性问题。另一方面，本地 DNS 服务器不会产生过载，因为它们是远离层次的根，并能够处理递归管理。虽然实际不太可能，但本地 DNS 服务器也有可能迭代地运行。那么智能器（DNS 客户端）的任务就是处理重试。

ALM 与 P2P

由于缺乏大规模部署的网络层 IP 组播，应用层组播（ALM）在 21 世纪初获得了大量关注。从名思义，ALM 通过 TCP 或 UDP 会话在支持参与节点之间的单应用。也就是说，ALM 在应用层实现组播服务，而不需要网络层组播协议。它可以被看做是一种特殊类型的对等（P2P）应用程序。当它在应用层构建组播树并且需要时的中间节点将分组从父节点中继到子节点。因此，这些节点的行为与数据的消费者又像数据的提供者，与 P2P 系统中的对等一样。如何构建组播树是 ALM 研究的重点。另一方面，P2P 系统需要或不需要组播支持的范围更广泛的问题。例如，加密的应用（如文件共享）完全不需要组播支持。即使在数据流应用，大多数最新开发的 P2P 系统采用数据驱动的覆盖网络，或马赛克覆盖的概念，而不是树形覆盖，主要为了鲁棒性。CoolStreaming 就是一个典型的实现例子。

进一步阅读

DNS

自从 1987 年 DNS 首次标出以来，已经标出一些有关 DNS 的 RFC。这里，我们列出了一些经典的 RFC 标准。Albitz 和 Liu 还出版了一本关于此内容很受欢迎书。为了他们学习，还增加了 BIND 项目主页。

- P. Mockapetri, "Domain Names—Concepts and Facilities," RFC 1034, Nov. 1987.
- P. Mockapetri, "Domain Names—Implementation and Specification," RFC1035 Nov. 1987.
- M. Crawford, "Binary Labels in the Domain Name System," RFC 2673, Aug. 1999.
- P. Albitz and C. Liu, DNS and BIND, 5 th edition, O' Reilly, 2006.
- BIND—a DNS server by Internet Systems Consortium, available at https://www.isc.org/products/BIND/

邮件

这里我们列出一些有关电子邮件的最新 RFC。很明显，电子邮件系统的设计从来没有停止过演变。

还为你感觉电子邮件系统的初步试验提供了 qmail 项目网站。

- J. Yao and W. Mao, "SMTP Extension for Internationalized E-mail Addresses," RFC 5336, Sept. 2008.
- J. Klensin, "Simple Mail Transfer Protocol," RFC 5321, October 2008.
- P. Resnick, "Internet Message Format," RFC 5322, October 2008.
- The qmail project, http://www.qmail.org/top.html.

WWW

以下是有关 WWW 的一些经典著作，其中包括一篇有关 Web 搜索的先驱性文章和 HTTP 1.1 的 RFC，1999 年对 HTTP 1.0 更新后它就得到了广泛应用。还可以阅读 Tim Berners-Lee 有关 WWW 未来架构的描述。

- S. Lawrence and C. L. Giles, "Searching the World Wide Web," *Science*, Apr. 1998.
- R. Fielding et al., "Hypertext Transfer Protocol — http/1.1," RFC 2616, June 1999.
- World Wide Web Consortium (W3C), "Architecture of the World Wide Web, Volume One," W3C Recommendation, Dec. 2004.
- The Apache project, http://www.apache.org/.

FTP

似乎 FTP 的发展仍然处在不断进行中，尽管速度相对缓慢。你可能会对最新的 FTP 扩展性感觉好奇，那就不妨学习 RFC 3659。

- J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," RFC 959, Oct. 1985.
- S. Bellovin, "Firewall-Friendly FTP," RFC 1579, Feb. 1994.
- M. Horowitz et al., "FTP Security Extensions," RFC 2228, Oct. 1997.
- P. Hethmon, "Extensions to FTP," RFC 3659, Mar. 2007.
- The wu-ftp project, available at http://www.wu-ftpd.org/.

SNMP

有关 SNMP 的 RFC 数量会令你大吃一惊。下面是一些重要 RFC，供大家参考。但我们建议你迷失在 SNMP 丛林中之前买一本书来看看。

- M. Rose and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets," RFC 1155, May 1990.
- J. Case et al., "A Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.
- J. Case et al., "Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)," RFC 1903, Jan. 1996.
- J. Case et al., "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)," RFC 1905, Jan. 1996.
- J. Case et al., "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)," RFC 1907, Jan. 1996.
- J. Case et al., "Introduction to Version 3 of the Internet-Standard Network Management Framework," RFC 2570, Apr. 1999.
- D. Harrington, "An Architecture for Describing SNMP Management Frameworks," RFC 2571, Apr. 1999.
- The Net-SNMP project, available at http://www.net-snmp.org/.
- Douglas Mauro and Kevin Schmidt, *Essential SNMP*, 2nd edition, O'Reilly 2005.

VoIP

以下是有关 VoIP 主要构件的 RFC。RTCP 是 RFC 3550 的一部分。使用 Asterisk 体验 VoIP 的世界。

- M. Handley et al., "Session Announcement Protocol," RFC 2974, Oct. 2000.

- J. Rosenberg et al., "SIP: Session Initiation Protocol," RFC 3261, June 2002.
- H. Schulzrinne et al., "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, July 2003.
- Asterisk, the Open-Source PBX and Telephony Platform, available at www.asterisk.org/.

流媒体

虽然流媒体应用的传输协议各异，但不管是普通常用的 RTP 还是如来自 RealNetworks 的专有的 RDT，实际的控制协议基本上都是采用 RTSP。试用 Darwin 和 Helix 软件包。此外，也不要错过利用最热门的 RTMP 协议的学习，像 Youtube 的 Flash 视频门户网站就是利用该协议的。

- H. Schulzrinne et al., "Real Time Streaming Protocol (RTSP)," RFC 2326, Apr. 1998.
- M. Kaufman, "QuickTime Toolkit Volume One: Basic Movie Playback and Media Types," Apple Computer, Inc., 2005.
- The Darwin Project, available at http://developer.apple.com/opensource/server/streaming/index.html.
- The Helix Project, available at http://en.wikipedia.org/wiki/Helix_(project).
- The RTMP protocol specification, available at http://www.adobe.com/devnet/rtmp/.

P2P

还没玩过的 P2P 客户端介绍了吗？下面的研究工作一定会让你深入理解 P2P。

- Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," in *Proceedings of ACM Supercomputing*, 2002.
- S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, Vol. 36, No. 4, pp. 335-371, Dec. 2004.
- Daniel Hughes, Geoff Coulson, and James Walkerdine, "Free Riding on Gnutella Revisited: The Bell Tolls?," *IEEE Distributed Systems*, Vol. 6, No. 6, June 2005.
- Javed I. Khan and Adam Wierzbicki (eds.), "Foundation of Peer-to-Peer Computing," Special Issue, *Computer Communications*, Volume 31, Issue 2, Feb. 2008.

常见问题解答

1. 为什么大多数互联网应用协议都总是 ASCII 和可变长度？

   答：ASCII：容易解码并且可灵活地扩展。

   可变长：更广泛的参数值和长度。

2. 为什么运行在 TCP 和 UDP 之上的服务器分别会有并发和迭代实现？

   答：并发：如果服务时间很长（通常较长）。

   迭代：如果服务时间很短（通常较短）。

   TCP：可靠的面向连接服务。

   UDP：不可靠的无连接服务。

   最常见的组合是并发 TCP（长时间和可靠的）和迭代 UDP（短时和不可靠的）。

3. DNS 服务器如何将域名解析为 IP 地址？

   答：本地域名服务器首先检查自己的缓存。如果结果没有命中，则就查询根域名服务器，将本机域名服务器重定向到二级域名服务器。二级域名服务器再重定向到三级域名服务器，以此类推，直至有一台域名服务器对该域名响应一个 RR 记录（A 记录）。这个过程是迭代的，这比递归更常见（所有相关的名字服务器在查询过程中都要保持状态）。

4. DNS 服务器如何将 IP 地址解析为域名？

   答：除了将 A 记录替代为 PTR 记录外，与问题 3 的答案相同。

5. 在前向 DNS 和反向 DNS 中分别使用什么资源记录？

   答：前向 DNS：A 记录。

反向 DNS：PTR 记录。

6. 如果你发送一封电子邮件让朋友阅读会涉及什么样的实体和协议？

答：SMTP：MUA（邮件用户代理）→本地邮件服务器的 MTA（邮件传输代理）。

SMTP：本地邮件服务器的 MDA（邮件投递代理）→远程邮件服务器 MDA 的 MTA→远程邮件服务器邮箱。

POP3 或 IMAP4：远程邮件服务器的 MRA（邮件检索代理）→MUA

7. POP3 与 IMAP4 有哪些不同？（比较命令的数量、灵活性和使用。）

答：命令的数量：IMAP4 > POP3

灵活性：IMAP4 > POP3

使用：Web 邮件（IMAP4）与下载（如 Outlook）（POP3）

8. 当你从服务器下载邮件时，交换了哪些 POP3 消息？

答：STAT、LIST、RETR、DELE、QUIT、+OK、-ERR 等。

9. 在下载、填写和上传 Web 表单时，交换了哪些 HTTP 消息？

答：GET、POST 或 PUT、HTTP/1.1 200 OK 等。

10. 在 HTTP 1.1 中，连接持久性意味着什么？

答：可以在一次 TCP 连接中发送多个 HTTP 请求。

11. 前向缓存与反向缓存有什么不同？（比较它们的位置和缓存内容。）

答：前向缓存：在内容消费者一端（客户端），来自许多站点的异构性。

反向缓存：在内容提供者一端（大的 Web 站点），来自该站点的同构性。

12. HTTP 代理如何拦截发向 HTTP 服务器的 HTTP 请求？

答：它与客户端进行 TCP 三次握手，接受 HTTP 请求，处理请求（如缓存、过滤、日志记录），如果 OK 就向客户端发送 HTTP 响应。如果需要，它建立一条与 HTTP 服务器的 TCP 连接，将 HTTP 请求转发给服务器，获得响应，处理响应（如过滤和记录），并将响应返回给客户端。

13. 如果 HTTP 缓存代理发生缓存未命中，那么它会做什么？对于一个特定的客户端，可以建立多少条 TCP 连接？

答：它会建立一条到服务器的 TCP 连接并将请求转发给服务器。然后将响应传回客户端。它有两条 TCP 连接：一条 TCP 连接是与客户端，另外一条 TCP 连接是与服务器。

14. 主动模式与被动模式 FTP 的区别？（从不同的角度描述模式以及如何建立数据连接。）

答：从服务器的角度来看。

主动模式：客户端通过控制连接，向服务器发出“端口 IP 地址 端口号”。服务器回复 200，然后连接到客户端以便建立数据连接。

被动模式：客户端通过控制连接，向服务器发出“PASV”通过到服务器的控制连接。服务器用它愿意监听的 IP 地址和端口号应答。然后，将客户端连接到指定的端口，建立数据连接。

15. FTP 中的控制和数据连接？（解释为什么需要两条连接。）

答：这种带外信令用来交换控制消息，甚至当正在进行长的数据传输时也是如此。

16. FTP 中上传和下载一个文件时，分别使用主动模式和被动模式，在控制连接上交换了哪些协议消息？

答：主动下载：PORT、200、RETR、200。

被动下载：PASV、200IP 地址端口号、RETR、200。

主动上传：PORT、200、STOR、200。

被动上传：PASV、200IP 地址端口号、STOR、200。

17. 为什么流媒体对互联网时延、抖动和丢失具有健壮性？

答：许多流媒体源采用一种可伸缩的分层编码方案，它们根据测量的网络状况，调整自己的编解码位速率。大多数流媒体接收器包有一个抖动缓冲区用于延迟音频/视频的播放时间以便吸收抖动并顺畅地播放。由于流量是单向而无交互的，所以增加的延迟用户还是可以忍受的。

练习

动手练习

1. 首先阅读 BIND9 的“dig”手册（实际最新版本），尤其是“+trace”（跟踪）和“+recursive”（递归）选项，并回答下列问题。
   a. 默认情况下，通过 dig 产生的查询是递归查询（因此本地域名服务器继续代表客户端进行查询）。为什么要使用 dig（或者其他应用程序中的解析器例程）？递归查询 www.ucla.edu，解释在应答中的所有 5 个部分中的每个 RR。
   b. 在使用 dig 对 www.ucla.edu 进行迭代查询中，描述咨询过的每台域名服务器。
2. 在 Linux PC 上使用 qmail 建立电子邮件系统。系统应该提供 SMTP、POP3 和 IMAP4 服务。逐步写下操作步骤。请参阅 http://www.qmail.org/中的文档。
3. 阅读 SMTP 和 POP3 命令。然后 telnet 你的 SMTP 服务器（端口 25）并给自己发送一条消息。在此之后，telnet 你的 POP3 服务器（端口 110）并检索该信息。记录在该会话中发生的一切。
4. 在 Linux PC 机上使用 Apache 构建一台 Web 服务器。修改配置文件以便建立两个虚拟主机。此外，编写一些 HTML 页面，并把它们放在 Apache 的文档根目录中。写下虚拟主机的设置并对显示 HTML 文件的浏览器屏幕拍照。
5. Telnet 你的 Web 服务器（端口 80）上，并使用 HTTP 1.0 获得一个文档。观察 HTTP 响应头部。记录在该会话中发生的一切。
6. 在 Linux PC 机上使用 Squid 构建缓存代理服务器，并且配置让你的 Web 浏览器使用它。浏览自己的 Web 站点并跟踪 Apache 和 Squid 的日志文件，观察是对哪个服务器服务请求。解释日志文件的内容。
7. 读取 HTTP 请求和响应头部的描述。使用 Sniffer 或类似的软件观察在练习 6 中生成的 HTTP 请求和响应。截屏并加以解释。
8. 安装并运行 wu-ftpd 或任何其他的 FTP 服务器。将它配置为支持两个虚拟的 FTP 服务器并实现非对压缩。写出每步操作和配置文件。
9. 安装并运行 Net-SNMP。在本地主机使用 snmpbulkget 检索 tcpConnTable。写下每步操作并记录下结果。
10. 研究局部性，通过更改 getNextOptimisticPeer() 函数中的随机选择代码，将往返时间延迟考虑进来。例如，你可能优先考虑具有更低往返延迟的对等。讨论在选择乐观疏通对等时为什么考虑局部性很重要。注意，乐观疏通起到查找潜在计算相对对等的最初赋值角色。

书面练习

1. 在本章中介绍的无联网应用中使用了哪个端口和启动模式（是 [x]inetd 还是单机）？将答案列成一张表。
2. 在处理并发请求时，交互的无连接服务器与并发的面向连接服务器的主要区别是什么？
3. 在图 6-4 中，netu 域中有多少个区？
4. 有多少个根域名服务器？请列举出来。
5. 在下列情况下可以使用什么样的 RR？使用例子解释其中的每一个。
   a. 在前向查询过程。
   b. 在反向查询过程。
   c. 解析域名 B，它是域名 A 的别名。
   d. 在邮件转发中。
6. 发送电子邮件时，我们可以将收件人的电子邮件地址放在 Cc：和 Bcc：字段。两个字段之间有什么区别？
7. Webmail 是基于 Web 浏览器的，并且包括对 POP3 和 IMAP4 的支持。说明基于 POP3 协议的 Webmail 和基于 IMAP4 的 Webmail 之间的差异。
8. 垃圾邮件是以相同电子邮件的多份副本供论互联网，企图迫使不愿意接受它的人们不能有选择地接收消息。提出一些与垃圾邮件做斗争的策略。

9. URL、URL 和 URN 之间的关系是什么？对于每种方案写出两个例子，并解释其含义。
10. HTML 和 XML 有什么相同和不同之处？
11. 什么是 HTTP 1.1 流水线和持久连接？它们都有哪些优点？
12. 描述 HTTP 和 HTML 如何将 HTTP 重定向到不同的目的地。
13. 缓存代理何时不再缓存一个对象？
14. 强缓存、敏性和弱缓存、敏性有哪些主要区别？那种方案适用于新闻站点？为什么？
15. 不经过手动修改就能让浏览器使用代理缓存，那么如何强制 HTTP 请求通过缓存代理？
16. 分别描述（包括使用的命令和参数）为 FTP 设立一个主动和被动连接的过程。假设已经在端口 21 上建立控制连接。
17. 解释图 6-34 中 FTP 会话例子的应答代码。
18. ASN.1、SMI 和 MIB 之间有何关系？
19. 管理站点如何有效地使用 GetNextRequest PDU 获得图 6-39 中 MIB 树中的对象？请说明这一点（提示：在可变绑定列表中的多个对象）。
20. 一个代理只有什么样的应用？SNMP 代理如何处理对它的引擎和应用的查询请求？
21. 比较通过 IP 和帧中继传输语音的利弊。请在性能和部署拓扑/成本上进行比较。
22. SIP、SDP 和 SAP 之间的关系是什么？
23. IL 323 和 SIP 之间的区别是什么？从它的组件和功能方面进行解释。
24. RTSP 和 HTTP 流媒体之间都有哪些优点和缺点？
25. QoS 控制在流媒体服务器和客户端是如何实现的？如果分组的延迟/抖动很高，那么客户端会如何做？
26. 在流媒体中，音频和视频信息是如何同步的？

互联网服务质量

互联网已经以健康的方式发展了数十年。它的成功性极大地依赖于TCP，它利用端到端的拥塞控制避免互联网上的超载。然而，运行在UDP上的实时应用，如流媒体、VoIP和部分P2P不是TCP友好的，如5.5节中的讨论。它们其中的某些还要求特定的服务质量（QoS）。对于这些应用来说，仅保证能传送有限的延迟、抖动等是不够的，需要动态的QoS。

遗憾的是，目前很难为互联网提供这样的路径，因为互联网是建立在目前竞争的网络体系结构上，其核心是无状态的，而将复杂的控制功能推给终端主机。因此，在终端主机上修改协议比在中间路由器和交换机上修改协议容易得多。大多数IP路由器只支持尽力而为的服务。路由器尽可能快地转发任何应用的分组而不关心终端主机是否或者何时才能够接收到。更具体地说，尽力而为服务意味着将到达路由器的所有分组插入队列中，直到队列溢出为止，而路由器以最大速率按顺序发送队列中分组。当网络负载很轻时，尽力而为服务对于大多数应用来讲是足够的。然而，负载取决于互联网带宽多快地增长与新的应用和用户多快地消耗互联网的带宽。

我们应该如何做才能将互联网变成能够支持QoS的网络呢？这里可以参阅异步传输模式（ATM）网络来找到答案，因为它能够支持QoS。受到ATM网络中QoS设计的启发，研究人员提出了类似的体系结构，称为集成服务（IntServ）。为两个终端主机之间提供一条具有带宽和延迟保证的路径。类似于ATM，IntServ中某个流的终端主机在发送分组之前需要与路由器协商以建立一条该路径并组着路径预留资源。此外，在预留路径上的所有路由器，除了需要知道分组应该向何处转发，而且还要知道何时去转发分组的正确时间以满足其QoS的要求。为了达到这一目标，路由器中单队列体系结构被多个或每个流的队列（这里分组不再以先入先出（FIFO）方式服务）所取代。当分组到达路由器时，将它分发到一个专门的队列并且要比其能更早到达的时间被分发到低优先级队列中的分组更早发送。

遗憾的是，IntServ为每个流的处理在全球互联网，或者甚至于单一服务提供商的区域网络这样的大型网络中是不可扩展的。因此，提出了名为区分服务（DiffServ）的另一种体系结构。不是为流按每个流的处理，DiffServ为基本目标是执行每个类的处理，以便流量有区分服务类的标头。主机使用网络之前首先与服务协商关于预定在汇聚流的服务类，而服务提供者为服务类分配已经设计级别的资源。然后，DiffServ网络的每一路将按照其服务类以本国的转发行为来处理分组。由于在DiffServ内，动态路径建立和按流资源预留不是必需的，所以它比IntServ更简单、更具可扩展性。同比，它就更加有可能在互联网上实施。

除了IntServ或DiffServ外，为了在控制平面和数据平面提供整体解决方案，还必须提供哪些基本组件？7.1节将以概括的方式回答这个问题，并提供了一个QoS框架。它还描述了将Linux系统流量控制（TC）模块作为QoS框架的参考设计。然后7.2节将深入研究前面两种QoS体系结构，按每个流的集成服务和按每个类的DiffServ，并对它们之间的区别进行比较。

虽然到目前为止还没有大规模实现QoS的IP网络存在，但大多数IP流量控制模块已经在操作系统中提供了。事实上，某些组件已经用于无处不在的路由器、网关或服务器上，尽管尚公部署整体的QoS解决方案。因此，很值得进一步研究这些组件。只便了解为它们已经开发了哪些排队算法？7.3节对这个问题进行了回答。与每个组件的算法讨论一起，还提出了在Linux下TC的开源实现以使演示TC中采用的算法，并研究它如何在路由器中实现。

历史演变：2000年左右QoS的动作

20世纪90年代，由于万维网（WWW）的引入造成了互联网爆炸。WWW不仅提升了互联网的用户数量，而且还改变了互联网的内容（从文本到多媒体，从静态到动态）。最初通过电话和电话线接

的内容目前组成在互联网上传播。这些趋势引起的变化用尽了互联网的带宽，并无法满足了实时服务的需求。

对于这个问题，由于当时的互联网带宽的高成本，研究人员研究了如何将流量划分成不同的类，然后为每个用户提供优质的服务，并向公众提供多方向为服务，假设所有流量均能在同一个网络上传输。与此同时，为什么研究人员就因为成本昂贵带宽。后来，由于光纤技术的突破，互联网的带宽变得廉价。本质，基础设施供给。当需要更多的带宽时，互联网服务提供商只需要多投资些设备为带宽的光纤链路。因此，21世纪初有关互联网上有关QoS研究论文的发表数量逐渐减少。

目前，来自互联网的QoS问题和需求消失了吗？没有，它的从移动到从有线转移到无线环境，由于无线带宽昂贵仍然是稀缺的；还从网络转移到服务器，因为网络带宽足够大时服务器就会成为瓶颈。事实上，很多最新的无线标准（如WiMAX）中找到QoS。然而，QoS问题的范围目前局限于链路和节点上，而不是全球互联网。

7.1 一般问题

为了能在IP网络中提供QoS，IP路由器需要配备许多额外的功能。首先，主机需要通过信令协议，要求沿着到其到目的地路径上的路由器预留资源。请求可能被路由通过一条路径，该路径上的路由器有更好的机会来提供请求的资源，这称为QoS路由，它与普通的不关心资源可用性的路由相对应。然后沿着该路径上的路由器执行许可控制，接受或拒绝请求。如果一台主机的预留请求被路径上的所有路由器接受，那么路由器就预留资源并准备服务于从主机发出的流量。路由器需要强制QoS供应，所有传入的分组首先划分为按每个流或按每个类的队列，管理队列必须它们所要的资源是合法请求的多，然后调度这些队列以便保证它们能够得到它们名下的共享带宽。这些数据平面操作分别称为分类、监管和调度。

以最通用的形式，QoS组架可能有6个组件，如图7-1所示。在本章中，我们将介绍它们的概念和功能。从讨论中可以看到在设计它们时的困难。对每个组件的算法设计讨论将留到7.3节中讨论。最后，我们给出在Linux下开源流量控制（TC）模块的概述。



图7-1 构建QoS感知网络元素的6个组件

7.1.1 信令协议

信令协议是用于与路由器协商资源预留的一种常见语言。这是支持QoS网络的第一个需求，因为QoS必须通过网络中的所有主机和路由器之间的合作来提供的。出于各种目的，提出了多种信令协议。其中，最有名的信令协议是资源预留协议（RSVP），它被应用程序用于在网络中预留资源。另一个例子是公共开放策略服务（COPS）协议，这是一种简单的查询－响应协议，用于部分QoS管理体系结构的策略管理系统中。在IETF中成立一个称作为下一步信令（NSIS）的工作组，以便研究更灵活的IP信令体系结构和协议（参见RFC 4080）。

7.1.2 QoS路由

如果传统由在沿路口与导航到的静态路径标志，那么QoS路由就可以看成是一个高级通路标

志系统，它不仅提供经过各种道路到达目的地的距离，而且提供车辆预计到达的时间。它根据各条道路的拥堵状况态实上述信息。在当前的 IP 网络中，路由器根据一些基本信息（如目的 IP 地址）选择最小跳数的路径，这就好像路标上的距离信息。但是，QoS 路由器还需要考虑带宽、延迟和满足 QoS 要求的路径上的丢失率。因为这个信息比起跳数更动态，所以在一个大型网络中它们就更难以收集和交换。

7.1.3 许可控制

QoS 路由，就像前边高级道路标志系统的例子一样，可以指导分组到最佳路径，但选择的道路以后还可能会拥塞。为了使分组避免产生拥塞，需要进一步控制在路径中许可的分组数量和类型。许可控制就负责这项工作。它是部署在网络或网关路由器中的项目，通过比较现有可用的资源量与所需要的资源量，决定该分组是否允许进入网络。这样的比较是困难的，因为资源量是随时间变化的。图 7-2 显示了一个例子。一个 3 Mbps 的带宽请求到达路由器 A。然后，路由器根据随时间变化的可用带宽决定是否接受请求。路由器 A 的决策难度在于如何正确估计带宽的使用，以确保有足够的带宽或功转发许可的流量，同时保持带宽的高度利用。

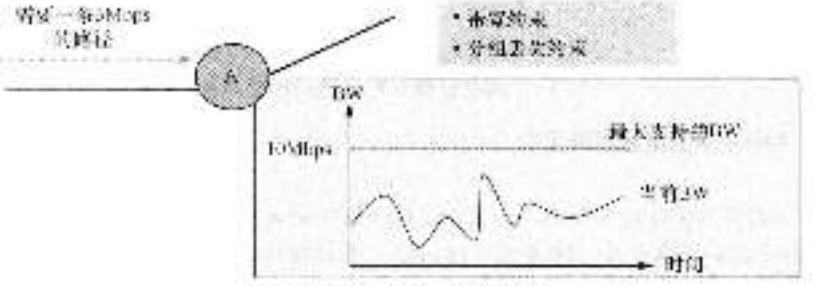


图 7-2 许可控制的操作例子

7.1.4 分组分类

一个合适的路径分别通过信令协议、QoS 路由和许可控制经过协商、选择、允许后，最后将分组发送到目的地。然而，为了强制执行 QoS，网络仍然需要一个组件来标识分组。例如，我们需要知道分组属于哪个流，以便提供相应的 QoS。因为有不同的规则对分组进行分类，所以可能会经过多次比较将某个分组分类为某一个特定的流或类。虽然分组分类工作十分繁重，但服从许多分类规则的快速分类是必要的。因此，如何快速地分类分组成为组件的主要问题。

在 IntServ 中，分类组件根据分组头部中 5 个字段的值来标识分组属于哪个流。在 DiffServ 中，它在网络边缘对多个字段执行范围匹配，在网络核心对单个分段执行简单的匹配将分组分成类而不是流。

7.1.5 监管

总会有某些车辆超过道路上的限速，这将给其他车辆的司机带来危险。类似的情况也会发生在网络上，因此我们需要一个的监管组件监控流量。如果流量流的到达速率超过其分配的速率，监管组件就需要标记、丢弃、或延迟某些分组。然而，在大多数情况下，监管阈值并非是一个准确的值，而且阈值的轻微变化是可以容忍的。因此，源流量通常由流量模型来描述，并且监管要根据流量模型来执行。最流行的监管机制称为令牌桶，它允许将监管流量限制为平均速率，但是也允许在突发时间段以最大速率发送。

7.1.6 调度

调度是支持 QoS 网络的重要组件。其总目标是根据预定义的规则或比率在不同的流或类之间执行

资源共享。已经提出了各种调度算法以实现特定的目的。有些方法简单，有些方法复杂和巧妙，这是以提供公平共享的准确保证。

如图 7-3 所示，调度器应该提供两个外部函数，入队（enqueue）和出队（dequeue），分别用于接收新到达的分组和决定要转发的下一个分组。那么应该有一个内部算法在“调度黑色管道”中调度分组，这可以分成：1）队列内的缓冲区管理；2）在多个队列之间的资源共享。在某个队列内的缓冲区管理又称为排队规则（queuing discipline），我们将在 TC 的开源实现中看到。

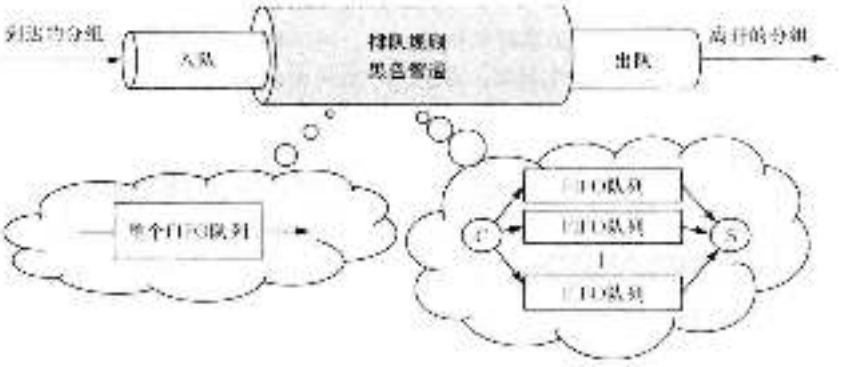


图 7-3 调度的概念和可能的体系结构

开源实现 7.1：Linux 中的流量控制元素

概述

Linux 内核提供各种流量控制函数。人们可以使用这些函数来构建 IntServ 路由器、DiffServ 路由器或者任何其他 QoS 感知的路由器。TC 和其他路由器函数之间的关系在图 7-4 中给出。这里用 TC 替代在原来 Linux 内核中 Output Queuing 的作用。它由以下 3 种类型的元素组成：

- 过滤器
- 排队规则（qdisc）
- 类

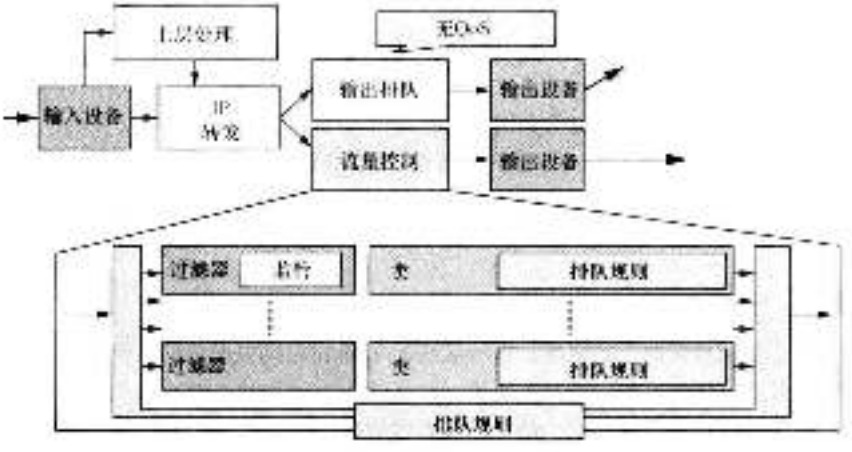


图 7-4 Linux 中 TC 元素的简单组合

框图

在图 7-4 中，过滤器负责根据某些特定的规则或分组头部中的字段来对分组进行分类。它们的源代码文件命名为 cls_prefix，并放在目录/usr/src/linux/ncbcd/中。例如，文件 cls_rsvp.c 实现 IntServ 路由器中需要的流标志。

排队规则支持两种基本函数，入队（enqueue）和出队（dequeue）。前一个函数决定要么丢弃分组要么将分组入队，而后一个函数确定排队分组等待的次序或者能够实现对排队分组的发送。最简单的排队规则是 FIFO。它将到达的分组排队到队列尾为止并将队列中的分组按照它们到达的次序发送出去。然而，其他排队规则会更加复杂，例如，在 sch_cbq.c 中实现的 CBQ，它将分组分成不同的类，每个类使用它自己的排队规则，如 FIFO 或 RED。排队规则和类的源代码也位于 /usr/src/linux/net/sched/ 中，但是它们的文件名是以 sch_ 开始的。基于分组封装分类的成员，ak_pcrr 用于映射分组，其结构已经在 1.5 节中讨论过。

算法实现

图 7-4 中给下半部分显示了一个上述机制元素的可能组合。可能有各种组合，可能由多个类和多个发送器组成的排队规则来将分组合到另一类中。TC 用户可以直接指令内核通过 Perl 脚本根据需要设计适合流量控制结构。

图 7-5 进一步演示了 TC 中更加简单的流程图，这里仅仅显示了一个 qdisc。当来自上层的分组到达 TC 时，调用 qdisc_enqueue() 将分组插入对应的队列。然后，时钟查发送分组时，定时器将触发 qdisc_wakeup() 分别调用 qdisc_dequeue() 和 hard_start_xmit() 选择和发送分组。一旦发送分组后，net_bh() 可能也会请求 qdisc_run_queues() 激活下一个分组的等待。

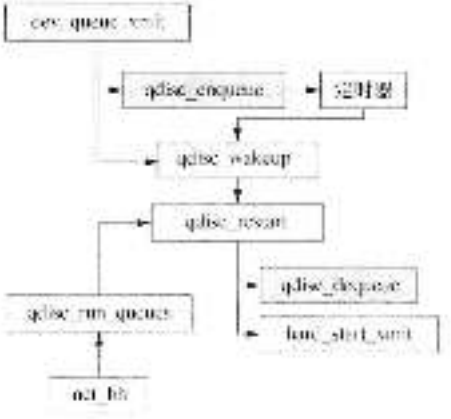


图 7-5 qdisc 元素的流程图

练习

你是否能够通过重新配置 Linux 内核来安装 TC 模块，并思考如何启用这些模块？在本章的开源实现中，我们将开发几个与课本相关的 TC 元素。因此，这是为练习做准备的最佳时间。你可以查本章"进一步阅读"中找到有用的参考文献。

7.2 QoS 体系结构

这里我们将介绍两种于 20 世纪 90 年代提出的 QoS 体系结构，并为互联网从尽力而为转变成支持 QoS 的网络提供指导。其总目标是一种完整的体系结构，能够满足由严格网络应用产生的任何 QoS 需求。它能够为这些应用提供一种可有带宽预留和保证最差情况下有延迟极限的虚拟路径。但是，这是一种昂贵的解决方案。因此提出了一种简单的和更加实用的体系结构：区分服务体系结构。顾名思义，DiffServ 的目的是为不同等级的用户群供不同的服务，而不是提供预留和延迟保证的服务。遗憾的是，即使 DiffServ 比 IntServ 更实用，但仍没有在全球互联网上得以广泛部署。然而，许多服务提供商的确提供优先级服务给某些应用（如 VoIP 应用），通过与其客户签订的服务等级协议（SLA）。

7.2.1 集成服务

本节将描述 IntServ 的一般操作过程。我们首先介绍应用能够进入 IntServ 网络的一种服务类型。然

后我们从应用程序的角度讨论预留请求，即信令协议。然后我们介绍 IntServ 路由器如何处理和满足预留请求。

服务类型

除了当前互联网为流提供的尽力而为服务外，在 IntServ 规范中还定义了两个额外的服务类型：保证服务和控制负载服务，如表 7-1 所示。

表 7-1　IntServ 中提供的服务类型

服务类型	有保证的	控制负载	尽力而为
提供 QoS	- 保证带宽 - 确保端到端延迟限制	根据应用程序的经验或网络	无
应用例子	VoIP 和视频会议	视频流	网络浏览
RFC	RFC 2212	RFC 2211	无

一旦某个应用订购了保证服务，它就可以将流量发到具有保证可用带宽和端到端最差情况下有延迟限制的路径上。任何交互式实时应用（如 VoIP 和视频会议）可以订购这种服务，因为任何额外的分组延迟都可以严重地影响用户的感受并且不被用户所感受。

在 IntServ 中的控制负载服务向订购流提供路径，这里分组传输很可能通过低利用率链路。也就是说，当订购服务时，流需要的质量在大多数时候得到满足。尽管控制负载服务没有有保证的服务好，但它更加便宜，适用于某些非实时的应用。例如，在线电影可以定购控制负载服务来保证大多数分组能够在预期的时间收到，特别是当由尽力而为服务提供的带宽远低于编解码器速度时也是如此。那么，在播放视频流之前，播放程序可以缓存某段时间的媒体数据来掩盖在流传输期间由短期拥塞造成的质量降级。由于控制负载服务的用户可以容忍短期的质量降级，所以网络资源就可以由更多的用户来共享，因此控制负载服务比保证服务更便宜。

资源预留请求的历程

一个在线电影播放器的应用程序决定定购服务类型以及它的流需要多少带宽和多少延迟之后，它就需要发送一个带有定购信息和源流量描述的 QoS 请求以便在 IntServ 域中预留资源。请求将被最近的 IntServ 路由器接收，如图 7-6 所示。路由器将根据其状态决定是否接受请求，如果它接受请求，那么它就会将请求转发到下一台路由器。路径上的所有路由器都接受请求后，资源预留处理就完成了，应用程序就可以以有保证的 QoS 开始接收分组了。RSVP 就是 IntServ 中用于这种预留通信的协议。

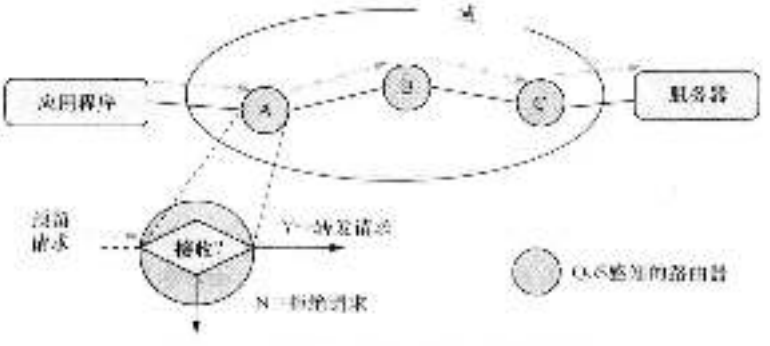


图 7-6　从应用程序的角度看 RSVP 处理

IntServ 路由器的请求处理

一旦接收到 RSVP 请求，IntServ 路由器就将它传递给信令处理组件——对应于图 7-1 所示的信令协议组件。根据它与许可控制组件协商的结果，路由器将更新信令分组并将它转发给下一个路由器。也就是说，信令处理组件仅充当一个转发器（transceiver）来转录由许可控制组件所做的决策。实际上，许可控制是负责输出链路资源管理的实际组件。许可控制的功能可分为两个部分。一部分用来采集在

维护当前使用的输出线路，另一部分则来决定余下的资源是否足以满足新请求的需要。

除了前面提到的两个组件外，在IntServ路由器控制平面中的另外一个组件是QoS路由。因为IntServ类型许可控制管理带宽分配，所以这里就不能叫QoS路由组件。但是，它可以用于寻找提供所需要的QoS保证的路径。因此之故，存在QoS路由，尽管是可选的，这在旨于提高流量被到达路径上的资源预留成功机会。

在IntServ路由器中的请求流

成为到达路径之后，就可以在它上面传输数据分组了。路径上的路由器应该保证应用程序的分组处理方式符合它们所订阅的服务。这种许诺由路由器数据平面中的三个基本组件执行：分类器、监管器和调度器。如图7-7所示。对于数据分析，路由器的入口在实际转发组件接受数据分组头部的5个字段（源IP地址、目的IP地址、源端口号、目的端口号或协议标识符）标识分组是否属于某个预留流。将那些属于许可的流的分组插入到对应的流队列中。本质上，在IntServ体系结构中，每个预留的流都有一个独立的队列。将不属于任何预留流的分组归类到尽力而为的FIFO队列。值得注意的是，需要为尽力而为流量预留部分资源以避免饥饿现象的发生。

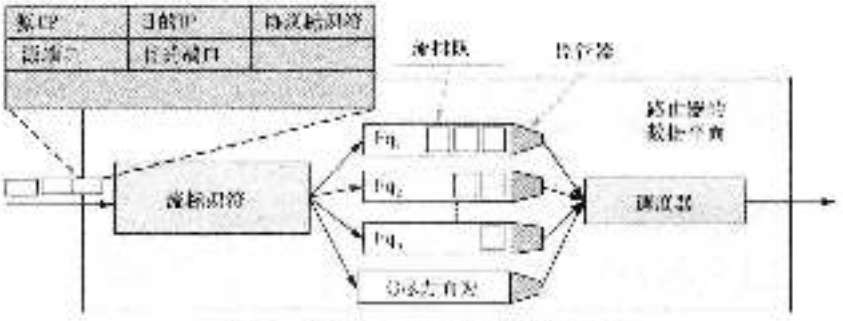


图7-7 在IntServ路由器中的数据平面

当分组进入到对应的流队列后，处理分组的下一个组件是监管器。它监督流的到达分组以便确定流量是否符合共资源预留请求中声称的行为。这叫“轮廓描述”。超外”的分组可能会被丢弃或延迟直到流量符合声称的行为。接下来，调度器在被监管的流队列中从排在前边的分组中选择出一个分组。将对应路由的分组发送到输出链路上。在大多数情况下，监督器不具有排队的分组，因为输出速率必须大于或等于接收速率。

调度器的角色对于IntServ来讲很重要，因为它是提供具有端到端延迟要求的关键。它的目的是减少最差情况下流的延迟并在所有预留流之间提供公平的处理。值得注意的是，公平不等于均等，更多解释将在7.3.4节中给出。

7.2.2 区分服务

尽管IntServ提供一种精确的QoS，但IntServ体系结构对ISP的部署来说过于复杂。此外，它的实现机制可能需要太多的计算资源，特别是对于需要处理很多流的核心路由器来说是如此。这两点复杂的设计将会造成网络资源被共同到一个对流的部署者。显然，互联网需要一种简单的、可扩展的可管理的解决方案。区分服务就是为了这种目标而设计的。图7-8显示了DiffServ的基本元素网。第一层显示了DiffServ涉及的功能，而接下来的各层则显了支持这些功能的特定协议或组件。

通用模型

DiffServ网络由一个或多个DiffServ域组成，而DiffServ域由多台路由器组成。在DiffServ域中有两种类型路由器，如图7-9所示。在域边界的路由器称为边界路由器，不管它是入口（ingress）还是出口（egress）路由器。而在内部的路由器称为核心路由器。入口路由器是域的入口。它在进一步转交给核心路由器之前分两步处理分组。第一步，按照预定义的监管标识和标记分组。分组的标记影响域中分组的转发处理。第二步，根据在开始服务之前客户与提供者之间协商的流量轮廓描述，监管和整形分组。第二步保证进入到域中的流量是在域的服务能力之内，因为域内核心不再进行控制了。核心路由器

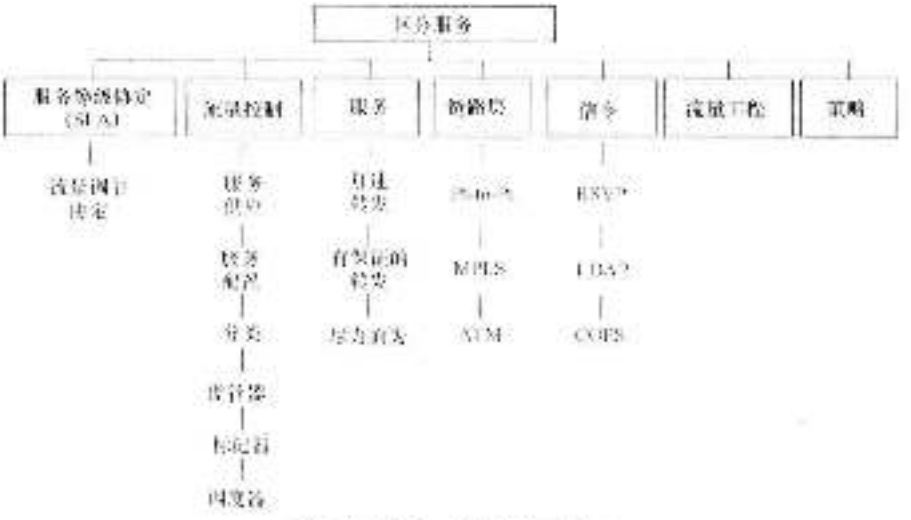


图 7-8 DiffServ 的基本元素树

的任务就是按照分组上的标记以特殊行为转发分组。在外出路由器上，可以按照当前域与下一个域之间协商的服务协议进行分组重标记和流量重整形。

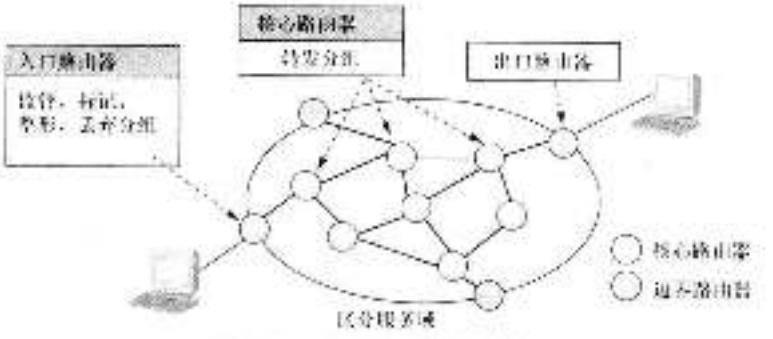


图 7-9 DiffServ 域的体系结构

DS 字段

边界 DiffServ 路由器标记进入 DiffServ 域的分组。分组上的标记告诉核心路由器如何处理分组。因为 DiffServ 直接构建在 IP 网络之上而没有引入额外的层，所以标记必须保存在 IP 头部中。因此 DiffServ 重新声明了 IPv4 头部中的 8 位服务类型（TOS）字段，用来指示转发行为。替代的字段称为 DiffServ（DS）字段，并且只有 6 位用做 DS 代码点（DSCP），用来编码转发行为（PHB），如图 7-10 所示。6 位 DSCP 字段可以表示 64 个不同的值。这些值分成如表 7-2 所示的 3 个部分。定义在第一部分中的代码点对应于主要的标准 PHB。另外两部分中的代码点分别保留用于试验和局部使用。

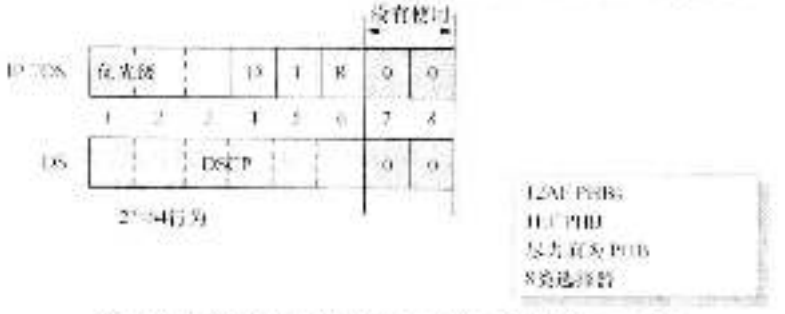


图 7-10 将 IPv4 头部中的 TOS 字段重新定义为 DS 字段

按每跳转发行为

这里我们介绍4组转发行为及其对应的在标准中定义的推荐代码点。前两组，默认PHB和类选择器PHB，提供有限的向后兼容，因为DS字段是从原来的IP TOS字段重新定义而来的。其他两个PHB组，保证转发（AF）（PHB）和加速转发（EF）PHB由IETF标准化，所以提供DiffServ。

表7-2 代码点的分配空间

序号	代码点空间	分配的场景
1	xxxxx0	标准动作
2	xxxx11	试验和局部使用
3	xxxx01	与上述类似但是受标准动作的影响

默认PHB组

对于IP网络中的大多数分组，TOS字段没有用，其值设置为0。为了让这些无DiffServ感知的分组通过DiffServ网络，DiffServ定义默认的DSCP值为000000，即等于大多数无DiffServ感知分组中TOS字段的值。对于这些分组，DiffServ将它们插入到尽力而为队列中并为它们预留最小的带宽。

类选择器PHB组

尽管在大多数情况下没有使用TOS字段，但某些厂商实际上使用TOS的前3个位来标识某些IP功能。为了让这些IP功能与DiffServ实现能够共存，将一个包含xxx000的DSCP字段映射到以不同优先级转发分组的一组PHB上。所有较大DSCP的分组比带有较小DSCP值的分组具有更高优先级转发。

AF PHB组

AF组的PHB保证转发流量源中的每个分组，如果这些到达的分组与其流的流量轮廓描述保持一致。轮廓描述称为流量条件协定（TCA）。然后，对于超过其TCA的分组，如果可能，AF PHB就会转发。

在AF PHB组中有4种转发类型，每类分配了一定量的带宽和缓冲区空间。对于每个类，流量分成三个等级的丢弃优先级。也就是说，在AF组中总共有12个单独的PHB。一旦一个类的缓冲区接近满，也就意味着到达的流量总量超过了类分配的带宽，具有高丢弃优先级的分组比具有低优先级的分组以更高的概率丢弃。

为了避免类内的拥塞，需要对到达类的流量总量加以控制。而且，由于一个分组的类在同一DiffServ域内不变，所以边界路由器需要许可、整形，甚至丢弃分组以保持DiffServ域不会过载。正如前面提到过的，DiffServ依赖于在边界路由器上的供给和监控来提供QoS。

事实上，为了检测是否发生拥塞是一个有趣的研究课题。有很多关于缓冲区管理的算法用于提前检测拥塞和降低负面效应，例如，1993年S. Floyd和V. Jacobson建议的随机早期检测（RED）。我们将要在7.3.5节学习这些缓冲区管理算法。

EF PHB组

EF PHB用来提供类似于传统点到点租用线路服务的功能，以低的丢失率、低的延迟和低的抖动转发分组。为了提供三种特性，DiffServ中的核心路由器必须能够在任何时候预留足够量的带宽，以便按照在源流量轮廓描述中指定的速率传输EF流量。

在核心路由器中，EF流量会抢占优先于其他流量类型以便获得对三个“低”特性的保证。在核心路由器中，实现这种保证的最简单方法是将不同类型的流量分类到单独的队列中，然后总是从EF流量队列中转发分组直到队列变空为止。但是，为了避免其他流量饥饿或者以突发的方式转发EF流量，需要严格约束将EF流量发送到网络的速度。通常，通过令牌桶实现的整形器将部署在边界路由器上以满足这种约束。那么，所有轮廓描述以外的不符合流量将使用默认PHB转发或者简单地在边界路由器上丢弃。

与AF PHB相比，EF PHB具有更高的服务质量和更低的突发容忍性。显然，EF PHB是恒定位速率和高质量需求流量的最佳选择。另一方面，AF PHB更适用于突发但能容忍分组丢失的流量。它们的相对特点在表7-3中列出。

在DiffServ域中分组的生命历程

在DiffServ域中分组的历程可以分成三个阶段：入口、内部和出口。第一个和第三个阶段是由边界路由器上处理的，而第二个阶段是由核心路由器处理的。这里通过描述在对应路由器上的操作来详述每个阶段。

表7-3 AF PHB和EF PHB及其相对特点

PHB组	AF（保证转发）					EF（快速转发）	尽力而为
特点	Olympic服务：4个队列；每个队列4个丢弃优先级类，每个具有3个丢弃优先级子类					额外分类/监管机制确保服务	无
在DS字段中建议的DSCP		AF1	AF2	AF3	AF4	101110	000000
	低	001010	010010	011010	100010		
	中	001100	010100	011100	100100		
	高	001110	010110	011110	100110		
流量控制	缺乏SLA监管、分类、标记、RIO/WRED调度					对齐SLA监管、分类、标记、优先级/WFQ调度	FIFO调度
不一致性的流量	重新标记为尽力而为					丢弃	转发

进入阶段

如图7-11所示，在进入阶段，每个分组将经过3个模块：流量分类、流量调整和流量转发。在第一个模块中，分类器根据预定义的策略标识到达的流量，并告诉组件应该采用哪种流量描述轮廓来管理流经的行为。然后将已分类的分组传递到第二个模块，进行流量调整。

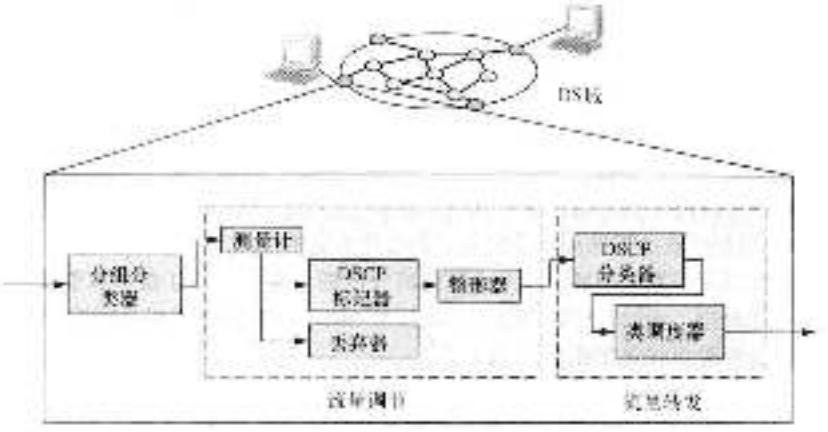


图7-11 在边界路由器中分组的进入阶段

在第二个模块中，按照流量描述轮廓中规定义，计量器测量流并将分组分类为描述轮廓之内还可以分类为描述轮廓之外。对于描述轮廓之内的分组，标记器将给予它们合适的代码点以使计它们能够通过域。对于描述轮廓之外的分组，它们可能丢弃也可能标记为对应于具有高丢弃概率转发行为的代码点。另外，它们也可以只是以描述轮廓之内的分组传给整形器。但是，与穿过整形器几乎没有任何延迟的描述轮廓之内的分组不同的是，描述轮廓之外的分组将被延迟直到它们与其流量描述轮廓一致为止。

在流量转发模块中，将已标记的分组带入对应类队列中。DSCP分类器的实现比在第一个模块中提到的分组分类器简单得多，因为只是查看在流量调整模块标记过的分组的DS字段，然后将分组到对应的类队列中。同时，类调度器以特定的转发速率从每类队列中转发分组，这是按照许可流量的容量配置的。

内部阶段

与具有多个处理模块的进入阶段不同，内部阶段只有一个模块，如图7-12所示。内部阶段的简单体系结构减少了核心路由器的处理成本并提高了转发速度。核心路由器仅负责根据分组的DSCP转发与处理行为，这与外出阶段的第3个模块的责任相类似。

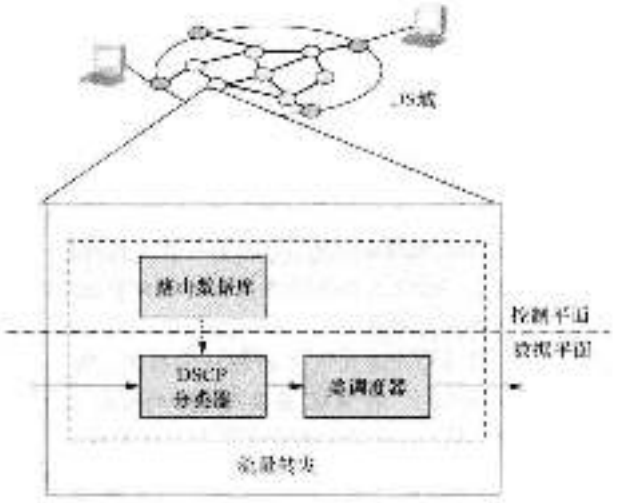


图 7-12 在核心路由器中分组的内部阶段

外出路由器

与负责 DS 域正常操作的进入路由器相比，外出路由器仅需要收集有关离开域分组的统计值。统计值可能包括实际的吞吐量和当分组通过 DS 域时用户察觉到的延迟。这些统计值在验证协商等级的 QoS 是否得到满足时非常有用。

与 IntServ 的比较

与 IntServ 相比，DiffServ 体系结构的实现更简单但是更加粗糙。首先，DiffServ 不能提供由 IntServ 提供的按需资源预留。在 DiffServ 下，用户需要与服务提供商签订合同，静态或动态地定义要注入到网络多少流量或者到底需要什么类型的 QoS。然后，根据合同，服务提供商就可以构建一个带有足够资源满足用户需求的网络。其次，DiffServ 中的到达流量被分成不同的组，又称为转发类。由于组的数量有限，来自多个用户的流量可能汇聚成一个类，一个队列，也就是说，用户可以察觉的传输质量可能受到同类其他用户的影响。最后，在 DiffServ 域的边界上处理 5 元组分组分类。也就是说，只有边缘路由器需要分类和标记分组。核心路由器以不同的行为转发分组就是根据分组头部中的标记进行的。

表 7-4 显示了 DiffServ 和 IntServ 体系结构之间的主要不同之处。DiffServ 的设计避免分类并调度大量分组到核心路由器中的困难，这其实是 IntServ 的主要问题。

表 7-4 DiffServ 和 IntServ 之间的不同

比较项目	DiffServ	IntServ
可管理的单位	类	流
路由器能力	边缘和核心	多合一
在标准中定义	转发行为	服务类型
需要有保证	供应	预留
工作区域	域	端到端

行为原则：为什么 DiffServ 和 IntServ 都失败了

尽管 DiffServ 比 IntServ 更具可扩展性，但它仍然没有得到广泛部署。有人会说因为光纤技术的进步导致带宽过供应和高速链路上的投资已经消除了利用复杂的 QoS 体系结构和机制支持互联网支持 QoS 的需求了。另外一个争论在于 IntServ 和 DiffServ 会将无状态的 IP 网络分别变成有状态的和半有状态的。对于 IntServ，所有的路由器将变成有状态的以便跟踪所有的预留状态。对于 DiffServ，至少所有的边缘路由器需要保留这些状态来分类和标记分组。

这些当然违反了互联网最初的设计理念，尽管互联网这种无状态属性是否应该保留仍然有争论

从某种意义上，多协议标签交换（MPLS）打破了本地域中的无状态性，通过允许有状态的包交换来替代无状态的包交换的路由。MPLS 成功地扩展到了更多的环境。但是将 MPLS 扩展到全球互联网上，将面临 IntServ 和 DiffServ 遇到的同样问题。然而，QoS 是扩展到多个域或服务提供商的一个很到位问题，但 MPLS 仅适在一个域内提高性能和平滑流取病线的解决方案。

行动原则：无线骨路中的 QoS

尽管 QoS 体系结构 DiffServ 和 IntServ 不能部署到互联网中，但是我们可以在底层的无线标准（如 IEEE 802.11e 和 802.16）中找到与 QoS 相关的规范。QoS 仍然在无线网络中获得关注，因为如今的无线网络中没有为用户提供充分的带宽。因此，当这种流量与其他背景流量混合在一起时，就需要 QoS 相关的机制和算法来保证实时流量的导输质量。

对于无线局域网中的用户，IEEE 802.11e 定义了两种访问模式：增强分布式信道接入（EDCA）和混合协调功能控制器信道接入（HCCA），在 MAC 层满足 QoS 的要求。EDCA 提供的高优先级流量比低优先级流量具有更多的机会发送。EDCA 中的高优流量比后高流量经历更短的等待时间以及更长的时间间隔发送，称为时机会（TXOP）。另一方面，与在 IEEE 802.11 中的访问模式 PCF 一样，HCCA 是一种基于轮询的模式，接入点主动地安排站点轮询的顺序和频率，从而确定每个站点接收的 QoS。主要区别在于，PCF 中的轮询站点在一次轮询时中仅发送一个分组，但 HCCA 中的站点在一轮轮询中的确定的 TXOP 可以传发送分组。因此，HCCA 对于每个分组发送比 PCF 具有更低的开销。

对于 WMAN 中的用户，IEEE 802.16 引入了四类服务：非请求的带宽分配（UGS）、实时轮询服务（rtPS）、非实时轮询服务（nrtPS）以及尽力而为（BE）。UGS 设计提供 T1/E1 仿线周期性地发送固定长度的分组，而 rtPS 和 nrtPS 分别为实时和非实时可变长度分组保证最小的带宽量。与基于 CSMA/CA 的 802.11e 相比，802.16 使用 TDMA 来管理无线媒体。在 802.16 中基站（BS）需要为每个定购者站点调度媒体，这是一种自然的集中式控制模式和一种用于 QoS 部署的方便体系结构。

7.3 QoS 组件的算法

给出了支持 QoS 的 IP 网络体系结构之后，我们就将重点放在构建支持 QoS 网络组件的技术上。这些体系结构具有更多的研究问题。有许多与这些 QoS 组件有关的算法。首先，我们描述用于许可控制和流标识的算法。然后，我们介绍整形和监管机制，如令牌桶及其变种。接下来，我们描述比其他组件更加复杂的调度算法。最后，我们讨论几种分组丢弃机制，这可以部署在核心路由器或者当前互联网路由器上以便缓解由于突发流量引起的拥塞问题。我们描述用 TC 的 5 种开源实现说明经典 QoS 相关算法的实现。

7.3.1 许可控制

应用程序向路由器发送 QoS 请求以便建立流后，许可控制组件需要决定是否接受新流通过路由器。新流的许可是根据输出链路的当前资源使用情况以及在请求中描述的需求来决定的。好的许可控制应该许可尽可能多的请求来耗尽路由器的资源，同时保证所有许可流的 QoS 请求都能得到满足。方法可以分成两类：基于统计的和基于测量值的。

基于统计值的控制

对于基于统计值的许可控制，流量源应该描述它的行为，如平均速率或峰值速率，并且路由器应该有按计算累计流量函数来估计资源总的使用情况同时决定是否接收流。但是，很难在带宽利用率和丢失概率之间折中来定义累计流量函数。

例如，我们可以通过两个参数：峰值速率和平均速率来描述流量源，假设它服从一种开－关（on-off）模型，这就意味着源既可以以峰值速率传输也可以空闲。那么，累积流量函数就可以是所有流的平均速率总和。如果添加了新请求发来的峰值速率之后，计算结果仍于最大约束，就接收请求；否则，就拒绝。这样的函数保证为流量分配带宽而不会有分组的丢失，但是会导致较低的带宽利用率。如果我们使用平均速率的总和作为累积流量函数，结果会如何呢？尽管链路将具有更高的利用率，但接收的

流将经常遭受拥塞和分组丢失。

对于这种折中，1991 年引入了术语等效容量，经常用于许可控制的文献中。等效容量表示一组复用流量通过一条（对遇到队列溢出时的概率具有限制的）链路所需要的最小的带宽。如此一来，当给定溢出概率和流的统计性质时，在需要的总的许可流的平均带宽小于等容量时人们就可以计算出等效容量并设计一种机制来允许新的流同时保证流遇到的分组丢失也将会低于给定的阈值。

基于测量的控制

由于很难有一个合适的累积流量函数，所以有些研究人员建议直接测量当前的带宽使用情况。为了获得有代表性的测量值并避免突发值，可以利用指数加权移动平均（EWMA）来计算新的使用估计值（$Estimation_N$）。也就是说，将新的测量与上次的估计值（$Estimation_E$）进行平均，

$$Estimation_N = (1-w) \times Estimation_E + w \times Measured_N$$

其中 w 为新测量值的加权比例。w 越大就会使历史值过期得更快，这就意味着算法更快并且资源以更高的概率被过度使用，但是流可能得不到它们需要的在其 QoS 需求中描述的处理。如果当前估计限好在某些最大约束之下，许可控制可能接受一个请求，但是很可能在下次估计会变成高值。那么，接收可能造成资源过载并影响所有接收流的处理，也包括新的流。

另外一种测量方法是时间窗口。估计值是从多个连续的测量间隔得出的，计算如下：

$$Estimation = f(C_1, C_2, C_3, \cdots, C_n)$$

其中，C_i 为采样间隔上测量的平均比率，f 为最大函数。同样，当 n 较小时，估计的带宽使用率通常更低，因此有更多的空间来接收新的流，并因此可以取得更高的利用率。另一方面，当 n 值较大时，算法在估计带宽使用率时就变得很保守，从而接收较少新的流。在大多数情况下，流能够得到更好的处理，但是是以牺牲资源利用率作为代价。

开源实现 7.2：流量估计器

概述

TC 提供了一个简单的模块用来估计当前传输速率，以字节和分组为单位。可以在文件 net/core/gen_estimator.c 中找到该模块。如前所述，测量传输速率有两种方法：EWMA 与时间窗口。由于 EWMA 比时间窗口方式花费更少的内存并且更容易实现，所以 Linux 使用 EWMA 方式进行通信量估算。

数据结构

用来保持速测量结果的数据结构称为 gen_estimator，如下所示：

```
struct gen_estimator
{
  struct list_head        list;
  struct gnet_stats_basic  *bstats;
  struct gnet_stats_rate_est *rate_est;
  spinlock_t              *stats_lock;
  int                     ewma_log;
  u64                     last_bytes;
  u64                     avbps;
  u32                     last_packets;
  u32                     avpps;
  struct rcu_head         e_rcu;
  struct rb_node          node;
}
```

由于速率估计器依分别以字节和分组为单位提供速率估计，所以可以在估计器中找到成对的变量名称，例如，last_bytes 和 last_packets，avbps 和 avpps，其用法将在后面解释。事实上，基于数据结构 rate_est 和 bstats 也是由两个变量组成的：一个用于以字节估计，另外一个以分组来估计。rate_est 用于保存速率估计结果，而 bstats 记录估计器到目前为止记录的计数的数据量。

算法实现

流量估计器包括 3 个主要函数。函数 gen_new_estimator()创建新的估计器，函数 gen_kill_estimator()用于删除闲置的估计器。一旦设定的时间到，Linux 内核调用函数 est_timer()，这里

时间间隔设置为（1 << interval），即 $2^{interval}$ 秒。在函数 est_timer() 中，发送率的计算和 EWMA 是通过图 7-13 中的代码实现的。

```
1: struct gen_estimator *e;
…
2: nbytes = e->bstat->bytes;
3: npackets = e->bstat->packets;

4: brate = (nbytes - e->last_bytes)<<(7 - idx);
5: e->last_bytes = nbytes;
6: e->avbps += ((s64)brate - e->avbps) >> e->ewma_log;
7: e->rate_est->bps = (e->avbps+0x8)>>5;

8: rate = (npackets - e->last_packets)<<(12 - idx);
9: e->last_packets = npackets;
10: e->avpps += ((long)rate - (long)e->avpps) >> e->ewma_log;
11: e->rate_est->pps = (e->avpps+0x1FF)>>10;
```

图 7-13：在 estimator.c 中函数 est_timer() 的一个代码分段

关于成对的变量名，可以在图 7-13 中找到成对的代码（第 4～7 行与第 8～11 行进行对比）。流每 $2^{(idx-2)}$ 就执行一次代码。每个流具有其所需的 idx，在函数 gen_new_estimator() 中设置。此外，为了避免浮点运算，在 avbps 和 avpps 中的值与它们的实际值相比分别扩大了 2^5 和 2^{10}。下一步，以字节为单位估计，如第 4 行中所写，估计器并充在 2^{idx} 秒内获得计数的数据量。通过从 e->bstat->bytes 中减去 e->last_bytes。

然后，为了获得在过去的 $2^{(idx-2)}$ 内的平均速率 avbps，与前面不同的是要除以 $2^{(idx-2)}$，即向右移动（idx-2）位。但是为了让 5 位二进制分数像 avbps 所做的那样，在第 4 行的运算就是左移（7-idx）位，即 >>(idx-2) << 5。然后，获得新的平均率后，第 6 行执行 EWMA 运算获得平滑的以字节为单位的估计值，最后将估计速率保存在 rate_est->bps 中。同样，我们可以通过运行从第 8～11 行就可以获得以分组为单位的平滑速率估计值。

练习

1. 解释第 6 行或第 10 行是如何执行 EWMA 运算的。在 EWMA 方程中，历史参数 w 的值为多少？

2. 阅读 gen_estimator.c，找出所有流量的 gen_estimator 是如何划分组的。你知道为什么参数 idx 是从 2 开始计数的吗？

7.3.2 流标识

在 IntServ 中，因为为每个流都保留了单独的资源，所以流标识或分类是必要的，以决定一个分组属于哪个流。此外，需要有一张带有每个流表项的表来存储流标识符和 QoS 参数。IntServ 中的流标识符由分组头部中的 5 个头部字段组成，这就是源 IP 地址和端口，目的 IP 地址和端口以及 7.2.1 节中提到过的协议标识符。标识符的长度是 32 + 16 + 32 + 16 + 8 = 104 位，占 13 字节。我们需要一种有效的数据结构来存储表和执行流量标识。

标识或分类是一种典型的数据搜索问题。很多数据结构都能够存储流表，但在时间和空间之间要做出权衡。简单的数据结构是二叉树，其空间需求小，但为了标识分组就需要多个内存访问操作。另一个极端是直接内存映射，但它不符合空间需求。为了平衡时间和空间的需求，使用散列结构就是一种常见和流行的方法。然而，如果我们进一步研究散列结构，我们就可以发现有很多不确定的原因会影响在散列表中的流量标识，例如，散列函数和冲突解决方法。

开源实现 7.3：流标识

概述

有许多建议的算法和数据结构实现流标识。它们共同的问题是如何在最短的时间内进行分组分类，同时还要使用最小的内存空间。当你继续深入时，你会发现在 Linux 的 TC 中使用二层散列结构。显

点，与直接内存映射和树结构相比，散列结构在时间和空间之间有着更好的权衡。可以在 net/sched/cls_rsvp.h 中找到相关的结构和代码。

数据结构

根据 IntServ 的定义，5 个字段标识一个流。二层散列结构如图 7-14 所示。第一层散列是输入的目的 IP 地址、端口号、协议标识等。其散列结果指示分组所属的 RSVP 会话列表。基于 RFC 中 RSVP 的定义，RSVP 会话代表了单播或是由目的 IP 地址、目的端口号、协议标识所组合标识的一流下去。通过使用第二层散列与源 IP 地址和源端口号作为散列键值，可以进一步确定分组属于哪个流。

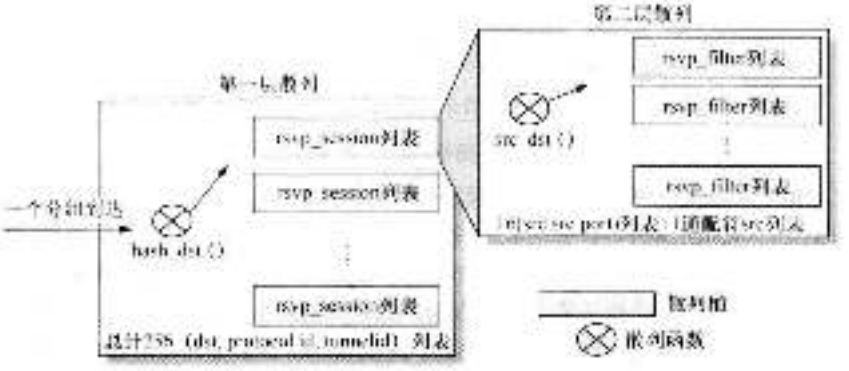


图 7-14 在 CLS_RSVP.H 中二层散列的结构

算法的实现

支持流标识的主要函数是 rsvp_classify()，其流程图如图 7-15 左侧部分所示。图 7-15 右半部分显示的是函数 rsvp_change() 的流程图，其中增加了一个新的流标识过滤器或者修改现有的流标识过滤器。这两个流程图都是容易遵循的，并且是不言自明的。

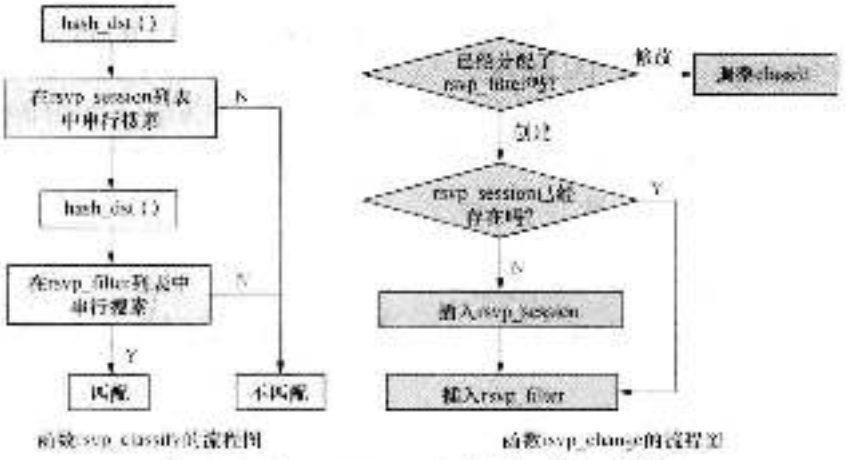


图 7-15 在 CLS_RSVP.C 中的两个函数的流程图

练习

1. 为什么要使得目的 IP 地址和端口号在源 IP 地址和端口号之前用于散列处理？
2. 通过阅读 net/sched/cls_rsvp.h 中的代码，是否能够找到基于标识的散列函数？

7.3.3 令牌桶

令牌桶机制可以监管流的到达率并确定队列在某个范围内。在 DiffServ 中，它能够置在边缘路由器上调节流的到达速率，使得速率符合与 ISP 合同中规定的速率。如图 7-16 所示，该机制由一个令牌桶和一个令牌流组成。

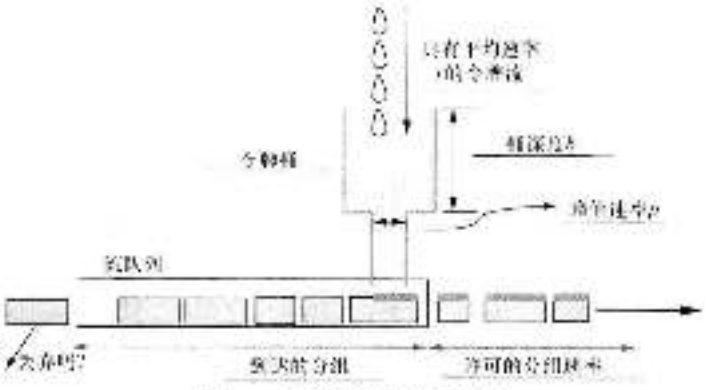


图 7-16 漏桶的操作体系结构

流以固定的速率 r 填充令牌桶，而桶可以累积令牌到最大量 b，这就是桶深。基本原理是，需要一定数量的令牌来允许分组的通过。让分组通过所需要的令牌数量等于分组的字节长度。随着分组被允许离开队列，令牌开始从桶中退出，直到桶空为止。然后，当桶空时，分组就被阻塞在队列中，而当队列满时，新到达的分组甚至会被丢弃。另一方面，如果在队列里没有分组或令牌消耗率小于 r，令牌就会在桶里累积，但累积不超过桶的容量 b。根据上述的原则，令牌桶允许的最大突发长度等于 $r \times t + b$，其中 t 是从分组到达突发开始所经过的时间。当分组突发到一个带有令牌桶完全装满的空队列时就会发生这种情况。另一方面，最大突发长度也可以表示为 $p \times t$，因为峰值速率受到 p 的限制。因此，我们得到最大突发长度的时间 $t = b/(p - r)$。

图 7-17 显示了在 24s 内令牌桶可能的运行情况。假设 $r = 1$ 单位/秒，$p = 2$ 单位/秒，$b = 15$ 单位，前 10s 没有分组到达。也就是说，如图 7-17a 所示，在令牌桶中累积 10 个令牌。然后，假设刚好在第 10s 到达了 3 个分组，其长度分别等于 10、9、5 个单位。下一步，如图 7-17b 所示，因为峰值速率 $p = 2$，所以第一个分组在 $t = 15$s 释放，即便有 10 个单位的令牌，在 $t = 10$s 时令牌桶中的令牌足以释放分组。释放第一个分组后，在第 10～15s 期间积累了 5 个单位的令牌。第 15～19s 期间加入了另外 4 个单位的令牌，第二个分组可以在 19s 时释放，如图 7-17c 所示。最后，因为在 $t = 19$s 时桶中没有令牌离开，所以最后一个分组需要另外等待 5s 获得足够的令牌后被释放，如图 7-17d 所示。

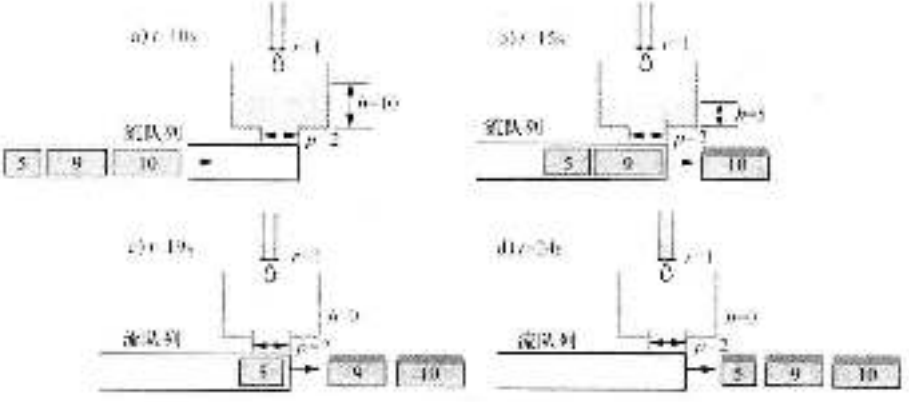


图 7-17 漏桶的操作例子

开源实现 7.4：令牌桶

概述

令牌桶机制的思想很简单，但直接实现可能会给内核增加沉重的负担。在简单的实现中，该机制应该有一个定期向桶中填充令牌的令牌生成器。此外，为了支持细粒度的速率调节，生成器不得不在

短时间内添加一些小令牌，而不是长时间的大令牌。显然，这样的实现会成为内核的噩梦，尤其是对于许多流量具有许多令牌桶时，每个生成器需要以高频率添加小令牌，以软调节高速率的流量。幸运的是，没有在 Linux 下以这种方式实现。当检查到有合格的分组离开队列时，才会进一步添加令牌。

数据结构

令牌桶机制广泛用于基带或整形网络流量。可以在 Linux 内核的 sch_police.c 或 sch_tbf.c 中找到其实现。下面我们就介绍 sch_tbf.c 中的代码。令牌桶实现使用的参数和变量定义为：

```
struct tbf_sched_data
{
/* Parameters */
  u32        limit; /* Maximal length of backlog: bytes */
  u32        buffer;/* Token bucket depth/size: MUST BE >= MTU/B */
  u32        mtu;
  u32        max_size;
  struct qdisc_rate_table *R_tab;
  struct qdisc_rate_table *P_tab;
/* Variables */
  long tokens;                  /* Current number of B tokens */
  long ptokens;                 /* Current number of P tokens */
  psched_time_t_ t_c;       /* Time checkpoint */
  struct timer_list_ wd_timer;      /* Watchdog timer */
}
```

在 tbf_sched_data 中的 R_tab 和 buffer 分别对应于图 7-17 所示令牌桶的 r 和 b，而 tokens（令牌）代表在桶中已经累积的令牌数量。值得注意的是，在结构 tbf_sched_data 中，令牌桶大小的单位是微秒。也就是说，给定传输速率 R_tab，buffer 表示允许传输分组的最大时间。此外，toker（令牌）表示允许传输分组的实际时间。

算法的实现

在令牌桶的最初设计中，当累积的令牌数大于队列头部的分组大小时，允许分组以峰值速率发送，这里峰值速率通常等于链路速率。然而，由于某些应用程序可能需要限制峰值速率为特定值，所以 TC 提供第二套令牌桶（P_tab,mtu,ptokens）来支持这个需求。这两个令牌桶机制用来确保通信量满足平均速率 =R，而最大突发周期 =buffer，峰值速率 =P，其中R 和P 分别在结构 R_tab 和 P_tab 中指示。

图 7-18 显示了 sch_tbf.c 中 enqueue() 的流程图。首先，函数检查分组的长度是否小于最大允许的队列长度。然后，如果队列还没满，就将分组插入队列中。与函数 enqueue() 相比，函数 dequeue() 更为复杂，因为它需要维护令牌桶的变量，如图 7-19 所示。

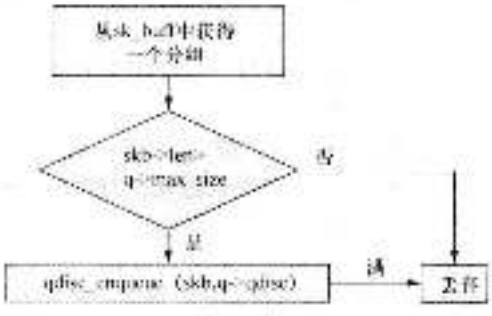


图 7-18 sch_tbf.c 中函数 enqueue() 的流程图

最后检查时间 t_c 之后，函数首先计算累积的令牌数量并将数量保存到 toks 中。当然，这种累积限制为缓冲区的最大值。然后，如果峰值速率已经给出，并且以峰值速率P 发送当前分组，它就估计允许传输分组的剩余时间并将估计值保存在 ptoks 中。同样，如果以平均速率R 发送分组，它也估计剩余时间，并将它保存回到 toks 中。接下来，如果这两种估计的剩余时间大于 0 时，则允许该分组发送出去，否则，将分组插回其相应队列的头部，等待下次传输机会。

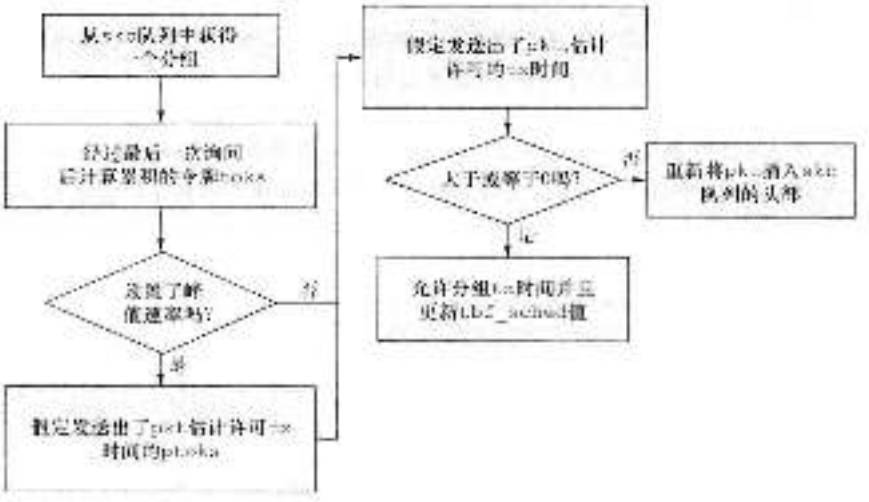


图 7-19 sch_tbf.c 中函数 dequeue() 的流程图

练习

如果感兴趣的话，可以找到另一个令牌桶的实现 act_police.c，解释令牌桶如何实现流量管制。

7.3.4 分组调度

有许多类型的调度算法以不同方式分配带宽。其中，公平排队类型是最著名并广为研究的分组调度。它确保所有流得到它们所需要的带宽，并保证在最坏情况下延迟得到接收到的分组。此外，如果流没有用完分配的带宽，带宽可以按比例和公平地在其他繁忙的流之间共享。我们可以进一步将这种类型的算法分为两大类：基于循环的和基于排序的。

基于循环的

这种类型的算法是启发式的。其中最著名的一个是加权轮询（WRR）调度器。在 WRR 中，每个活动流可以在一次循环中发送特定数量的分组。一个流发送的分组数量对应于其权值。例如，如果两个流的权值分别是 1 和 2，那么两个流在一次循环中发送的分组的数量可能分别是 100 和 200。WRR 很简单，但它只有在所有分组的长度和平均网络中才能执行得很好，因为 WRR 不考虑调度的分组的大小。一个具有小权值的流会得到更多发送的数据，如果它的所有分组大小比具有大权值的流中的分组大小的大。

一种改进版本，赤字循环（DRR），1996 年由 M. Shreedhar 和 G. Varghese 为具有不同大小分组的网络（如互联网）提出的。不再限制一次循环中允许发送的分组的数量，DDR 限制每个流在一次循环中发送的字节数。如图 7-20 所示，为每个流维护一个赤字计数器，以便跟踪在该循环中允许的数据量。无论何时轮到一个流时，计数器就会加上一个与流的权值成比例的固定数量的值。然后，分组就可以从流队列中发送出去，当计数器减去该组分组的大小直到计数器中的剩余量不够下一个分组的发送为止。然后，调度器将继续服务于下一个流。原则上，在流计数器中的剩余量将为其下一个流保留。然而，如果流在队列中没有等待发送的分组，那么它的计数器将重置为零，直到它的下一个分组到达。此外，流也会暂时从循环列表中删除。由于删除循环列表将变得越来越短，所以活动流可以更频繁地得到服务，并在一个周期内可以发送更多的分组来共享不活动流未使用的带宽。

图 7-21 中的 4 个部分显示了图 7-20 在接下来的 4 次循环中的变化。如图 7-21a 所示，第一次循环操作后，从每个流队列中不少释放了一个分组除了流 1 外，因为流 1 计数器中的值不够释放它的第一个分组（100 < 200）。另一方面，信用 100（300 - 200）留在流 3 的计数器中，这是为下一次循环保留的，因为第三个流队列现在空。接下来，在第二次循环中，4 个计数器分别增加 100、200、300 和 400，

其值分别变为200、200、400和400，现在就足够释放在流1、2和3中的第一个分组了。这样就得到图7-21b。请注意，将流4的计数器值400复位为0，因为第4个流队列是空的。然后，在图7-21c所示的第三次循环中，流4的计数器保持为0，因为空队列，前3个流的计数再次相加，但只有一个分组是从流3释放出来。当空流队列剩下的信用将获留用于第四次循环。最后，经过第四次循环的信用更新，流1和流2的计数器分别变为200和400，足够为每个流释放一个分组，如图7-21d所示。此外，由于在流2和流3中没有分组排队，所以，就像流4一样，流2和流3的剩余信用将丢弃。

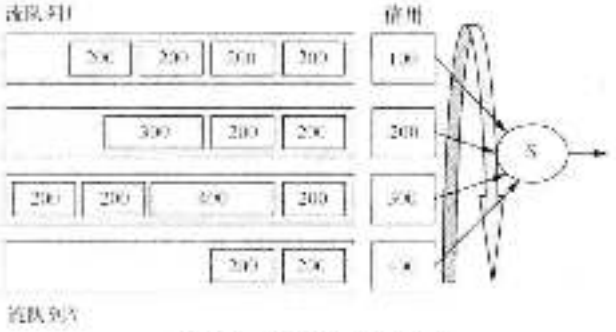


图7-20 DRR算法的说明

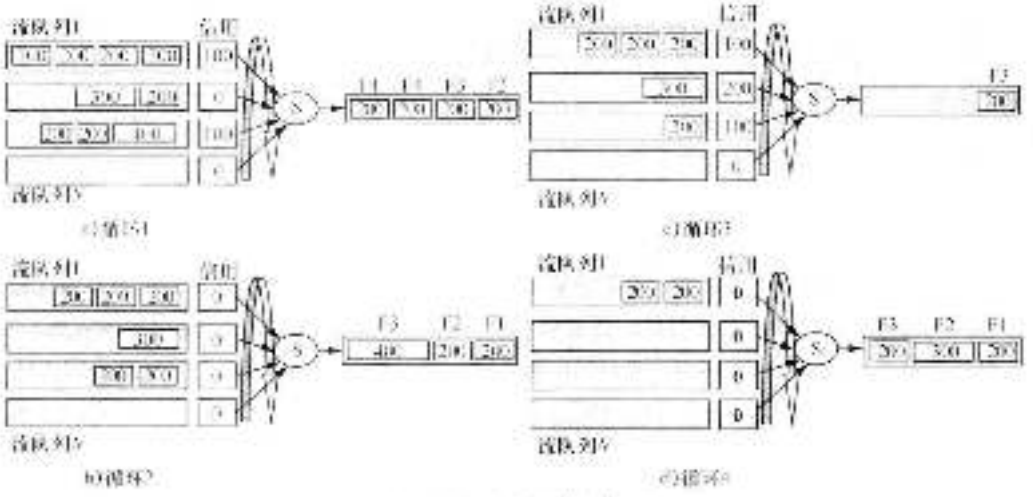


图7-21 DRR操作的例子

这种类型的调度器在其概念和实施上都很简单，但它只是确保每个流在很长的时间尺度上得到所需要的带宽。随着流的数量变大，一个流可能会等很长时间才有机会发送分组。如果流的分组以恒定位速率到达，那么长的等待时间可能会导致流有大的延迟抖动。这意味着某些分组可能会很快发送出去，其余可能会慢慢得到服务，具体根据它们到达队列的时间。

基于排序

基于排序的调度器的概念和基于循环的调度器是很不一样的。在描述它之前，我们首先介绍一个仅是用于流体模型网络体系结构的概念调度器。假设有3个流公平地共享3 Mbps链路。在流体模型体系结构中，调度器可以将链路分为3条速率链路。每个流都在其虚拟链路上可以连续地发送1 Mbps的分组而不会因其他流影响而造成延迟。此外，在某一个流没有分组发送时，剩余带宽可以按比例地被其他流量共享。换句话说，其他两个流中的每个都可以使用1.5 Mbps。这种理想的调度称为通用处理器共享（GPS），这是不可能实现的，因为输出链路每次只能传输一个分组。现实情况同于分组模型的情况。图7-22显示了流体模型和分组模式之间的差异。在流体模型中来自不同流的分组传输会同时进行，但分组模式则是交错进行的。

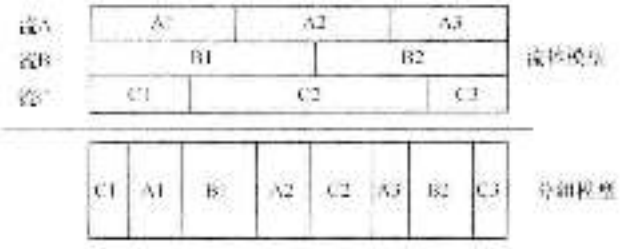


图7-22 在流体和分组模型中分组发送的顺序

虽然最优的调度器无法实现，但可以根据分组的大小和到达时间来计算流体模型中分组的发送顺序。然后，如果它可以按照流体模型最优调度器准确完成的相同顺序发送分组，分组模型调度器就被认为是理想的。这个想法很容易，但是按照流体模型中相同的传输顺序完成并不是很不足道的。有很多建议使用的算法，但是它们的设计可以归结为精确的带宽共享和实现的复杂性之间的平衡。下面我们讲述基于排序的调度器的一个版本，它由 A. K. Parekh 和 R. G. Gallager 于 1993 年提出的分组 GPS（PGPS），以上说明这些调度器的操作。

分组 GPS

PGPS 又称为加权公平队列（WFQ）。默认操作是每个分组在到达流队列时，会得到一个虚拟的发送时间戳（VFT），调度器选择所有流队列中具有最小 VTF 的分组发送出去。VFT 的计算与到达虚拟系统时间（VST）、分组大小、分组所属流的预留的带宽有关。由于分组的 VTF 确定其传输顺序，所以 VTF 的计算是确定分组模型调度器能够有效地模拟流体模型调度器的关键。

根据该算法，如果流是活动的，这就意味着在该流队列中还有分组等待，那么下一个到来分组的 VTF 将等于

$$F_i^j = F_i^{j-1} + \frac{L_i^j}{\phi_i}$$

其中 F_i^j 是流 i 的第 j 个分组的 VTF，L_i^j 是流 i 的第 j 个分组的长度，ϕ_i 为分配的带宽。理论上，如果每个流的第一个分组都同时到达，那么所有的流将永远相同。根据上式，就很容易得到与流体模型调度器中相同的传输完成顺序。不幸的是，这是很罕见的情况。在实际情况下，流会先闲置然后再忙。因此，有必要考虑如何为活动流的第一个分组设置 VTF。在一般情况下，第一个分组到达的 VTF 的计算方法如下：

$$F_i^1 = V(t) + \frac{L_i^1}{\phi_i}$$

这里其中 $V(t)$ 就是 VTF，这是一个在每个由分组事件到达或离开引起队列分裂的时间间隔区间的实际时间 t 的线性函数。假设将整个时间分为 n 个区间。T_i 和 S_i 分别表示第 i 个区间的开始实际时间和虚拟时间，其中 $i = 1 \cdots n$。然后，在第 i 个区间的 $V(t)$ 可以表示为：

$$V(t) = S_i + (t - T_i)K_i$$

$$S_i = S_{i-1} + (T_i - T_{i-1})K_{i-1}, S_0 = 0, T_0 = T_1, K_i = \left(\sum_{j \in A} \phi_j\right)^{-1}$$

A 是在第 i 个区间的活动流集合。

另外，根据图 7-20 所示情况，图 7-23 显示了当使用 WFQ 对分组的调度结果，4 个流的权值分别是 0.1、0.2、0.3 和 0.4，其比例如图 7-20 给出的带宽。假设所有分组在同一时间到达，这意味着它们的 $V(t) = 0$。除了显示分组的长度外，在图 7-23a 中每个分组用标识符进行标记。然后，图 7-23b 显示了每个分组的 VTF。根据上述公式进行计算。例如，在流队列 2 中的第一个分组的 VTF 可以通过将 $V(0)$ 添加到其长度 200 和流权重 0.2 的商得到，因此等于 1000。那么，它的下一个分组（标识符为 6 的分组）的 VFT 等于 $V(0) + (200/0.2) + (200/0.2)$，或者 2000。最后，得到所有分组的 VFT 后，就容易得到 WFQ 调度器的分组传输顺序，如图 7-23c 所示。

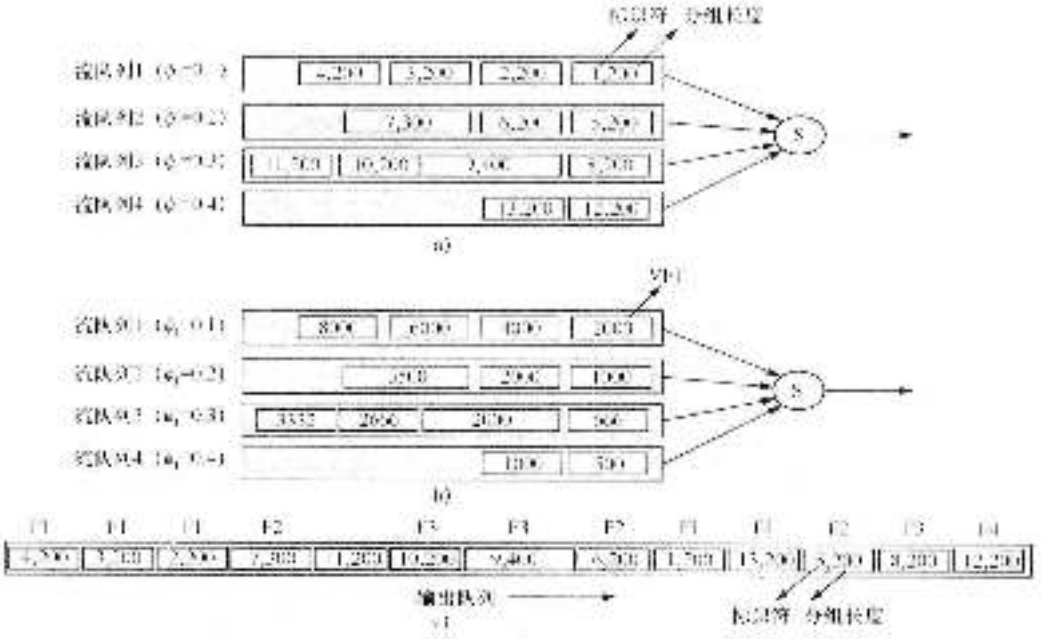


图 7.23 一个 WFQ 的集件到了

实际上，FQ 的定义是设计这样一种基于排序的调度器的关键问题。虽然它需要花费太多的精力来计算 FQ，但它可以获得 PGPS 中分组的传输完成次序恰好等于 GPS 中的传输完成次序，一个容易计算的 FQ 可能会导致一个新到活动流获得或小或大的 VFT，因此比其他活动流共享更多或更少的带宽，这不仅降低了带宽分配的公平性，而且也影响了调度器的最坏情况延迟保证。

开源实现 7.5：分组调度

概述

已经提出了许多算法来处理调度问题。事实上，即使我们只考虑公平排队调度，它也会是一个很长的清单。有这么多的算法，为何单独介绍，为何如此成分组。下面的解释是基于排序的 PGPS 类算法的实现，这是许多现有算法的基础，也是我们实现的一个。

数据结构

PGPS 算法是在 net/sched/sch_csz.c 中 csz_qdisc_ops 模块实现的。该模块为每个流分配一个 csz_flow 结构，以保存其信息。这里有两个变量，start 和 finish，保存在其流队列中分组的最小和最大完成时间戳。本质上，流队列的头部分组具有最小的完成时间戳，是队列具有最大的时间戳。除了结构 csz_flow 外，csz_qdisc_ops 模块维护了两个列表，分别为 s 和 f，用来实现 PGPS 调度器。在列表中的项目是指向结构 csz_flow 的指针。虽然这两个列表包含相同的活动流，但按表 s 根据结构 csz_flow 中的变量 snext 排序，其中每个流的 start 保存流队列中头部分组的最短完成时间戳。同样列表 f 帮助 csz_dequeue() 快速从成为下一个传输的流队列中取出下一个传输的分组，因为在列表 s 中的第一个流队列的头部分组一定具有最小的 VFT。另一方面，列表 f 根据变量 finish 进行排序，该变量每个流的 finish 保存流队列中尾部分组的 VFT。列表 f 也会在 PGPS 的虚拟系统时间计算中，将在后面与函数 csz_update() 一起介绍。

算法实现

接下来我们介绍 csz_qdisc_ops 模块中的 3 个主要函数，并说明它们的流程图。函数 csz_enqueue() 是模块的入口，其流程图如图 7.24 所示。当分组到达时，函数 csz_enqueue() 首先计算其 VFT。为了计算一个活动流的第一个分组的 VFT，分首先把系统时间（VST）更新为当前。因此，计算之前需要调用函数 csz_update()。为了使流变为活动，csz_enqueue() 需要把传输入列表 s 之处理它，给分组再次成为流头分组的机会。

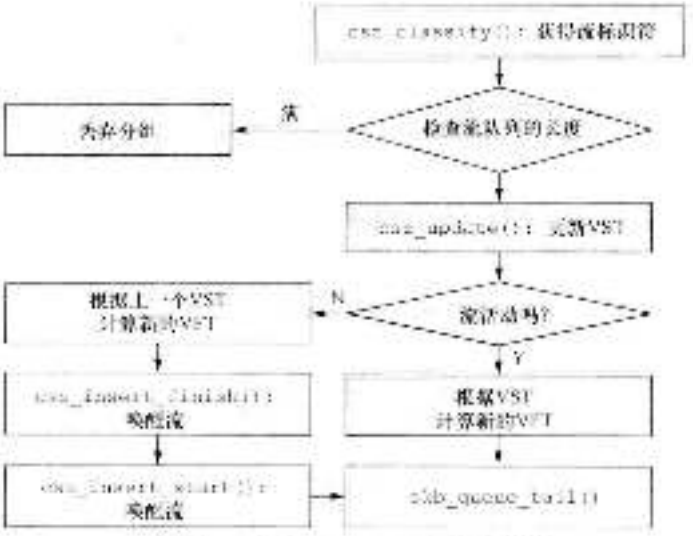


图7-24 函数 csz_enque() 的流程图

如图7-25所示，函数 csz_dequeue() 精确发送列表 s 中第一个表项所描述队列的头部分组。当一个流的分组发送出去时，csz_dequeue() 将调用 csz_insert_start() 重新再将流插入列表 s 以便保持其下一次循环的机会，如果流队列不为空。如果流队列为空，就将它从列表 s 中删除，以避免浪费系统资源。

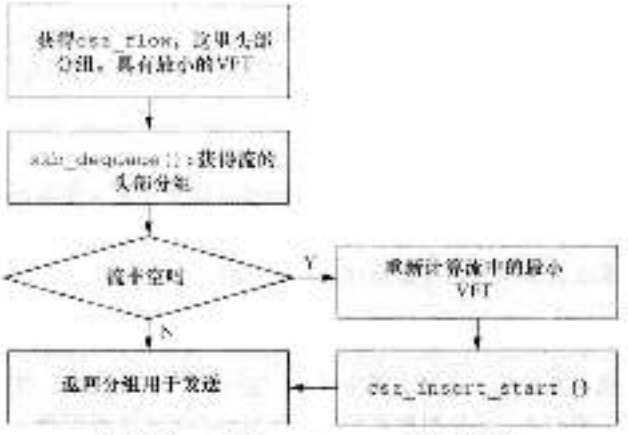


图7-25 函数 csz_deque() 的流程图

第三个函数，csz_update()，在 csz_qdisc_ops 模块中起着关键的作用，如图7-23所示。它负责计算VST。根据在PGPS中的描述，当分组到达和离开时，就进行一次VST计算。然而，通过对列表 f 的维护，只有当分组到达时 csz_qdisc_ops 才重新计算VST。这是由函数 csz_update() 维护的，如前面所述。首先，csz_update() 记录从上次调用时间到一个可变延迟所消去的时间。其次，它假设自从上次调用的所有流仍然是活动的，并计算出当前的VST。然后VST与列表 f 中标记为F的第一个表项变量 finish 进行比较。如果VST小于F，流一定是不活动的。csz_update() 将从列表 f 中删除它并在流变成不活动时计算VST，即不活动流发送流后一个分组所需要的时间。延迟也要更改为从流变成不活动开始所消去的时间。下一步，csz_update() 返回到图7-26中带有双边框的步骤，直到得到正确的VST，并从列表 f 中删除所有不活动的流。

练习

1. 与复杂的PGPS相比，DRR无论在概念还是实现上都容易得多。可以在 sch_drr.c 中找到其实现。请阅读代码，并解释这个简单而又实用的算法是如何实现的。

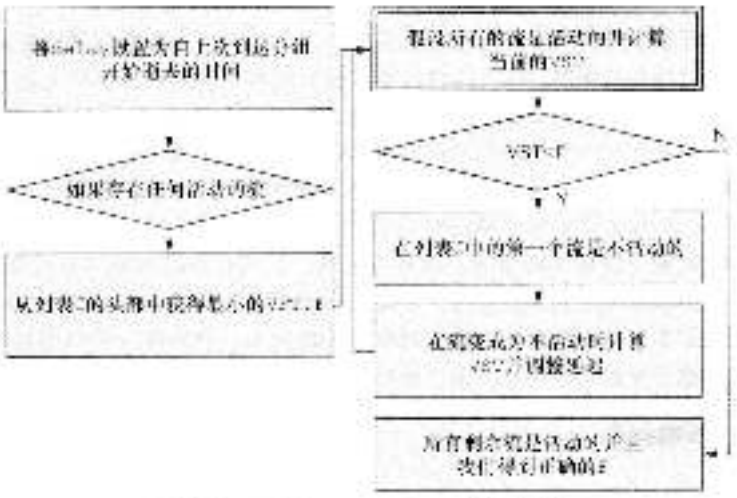


图 7-26 函数 [illegible] 的流程图

2. 在文件夹 sched 中有多种调度算法的实现。对于每种实现，你可以找出它与其他算法的不同之处吗？它们是否都属于分子和队列成？

7.3.5 分组丢弃

除了调度算法处理多个队列外，在服务质量体系结构中每个队列的分组丢弃机制是必要的，这里有多个流可能将分组转储到一个单类队列来共享为队列分配的带宽。这样一种丢弃机制会阻止行为不端的流将更多的数据分组输入到队列中而过度消耗其带宽，下面介绍两类数据分组丢弃机制。

尾部丢弃

尾部丢弃是最简单的分组丢弃策略，通常用于先进先出（FIFO）排队。当队列中没有多余空间时，该策略就丢弃新到达的分组，如图 7-27a 所示。到达的分组将被丢弃，直到有可用的队列空间为止。因为尾部丢弃是先进先出排队的默认策略，在先进先出队列中经常出现的问题也会在尾部丢弃中出现。例如，当一个突发源与其他具有平滑稳定速率的源共享一个先进先出队列时，这个突发源可能在短时间内占用所有可用的队列空间，从而造成从其他源新到达的分组丢弃。如果把单一队列分成多个队列，这个问题就可以避免。即每个流的源拥有长度有限的队列。然而，这意味着即使当路由器还有队列空间时，有些分组也可能会被丢弃。

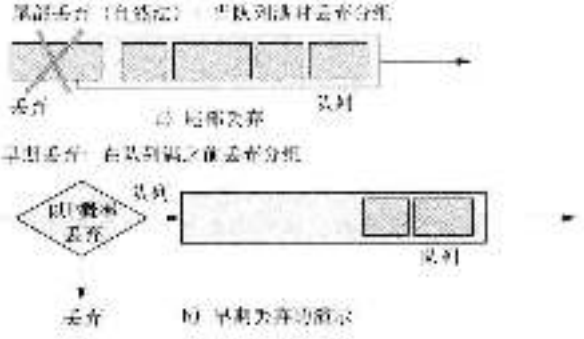


图 7-27

因此，在许多路由器中目前的实现是最长队列尾部丢弃（LQTD）。所有服务队列共享一个共同的内存池，并且当没有更多的空间给一个新分组排队时位于最长队列尾部的分组首先被丢弃。这种细化使得分组的到达率超过分配给它们的服务速率的服务类具有较高的丢弃概率，而在分配的速率之内工作的服务类会有低的丢弃概率，因为它们通常有更短的队列。

早期丢弃

虽然 LQTD 可以有效地阻止任何流过度占用队列空间，但它需要分组分类以便确定分组属于哪个流。如果有很多流经过这台路由器，那么这将是很沉重的负担。因此，如果优先选择所有流使用一个队列，那么阻止资源滥用的可能方式是在队列将满时以一定的概率丢弃新到达的分组，如图 7-27b 所示。这种思想称为早期丢弃。这种策略期望及早地向流的源报警，说明队列的大小是不足的，以便于这些源降低它们的传送速率防止发送过来的分组丢失。这种策略可以避免像尾部丢弃策略在很短的时间丢弃连续的包。那么连续丢弃常常产生降低 TCP 流的吞吐量。

设计这种策略的关键问题是判断排队空间是否将满。队列长度的阈值可能是最具自发式的方式。一旦队列长度大于阈值，新到达的分组将会以一定概率被丢弃。然而，由于队列长度的变化很大，一个意外原理的长队列并不总是意味着一个满队列事件即将到来。在同样分组到达情况下，早期丢弃的结果可能会导致比尾部丢弃策略更多总的丢弃分组数，尽管连续的丢弃可以被早期丢弃所避免。

开源实现 7.6：随机早期检测

概述

就像调度算法的情况一样，为了避免排队阻塞和 TCP 吞吐量降级，提出了许多排队管理（QM）算法来决定如何丢弃分组。其中，我们介绍一种早期丢弃类型的排队管理算法，随机早期检测（RED）。这里我们介绍 RED，因为它是最著名的早期丢弃排队管理算法。

数据结构

RED 算法的实现可以在 net/sched/sch_red.c 中找到。结构 red_sched_data 保存必需的参数以便运行算法，如 qave、qth_max 和 qth_min。RED 依赖于平均队列长度 qave 预测一个即将满的队列，避免不必要的分组丢弃。当 qave 小于最小阈值 qth_min 时，可以将所有到达的分组插入队列中。当 qave 大于 qth_min 时，以概率 Pb 丢弃到达的分组，Pb 的计算如下：

$$Pb = \max\_P \cdot \frac{(\min(qave, qth\_max) - qth\_min)}{(qth\_max - qth\_min)}$$

max_P 是当 qave>=qth_max 时丢弃分组的最大概率。但是其值可以建议为 0.1 或 0.2，而不是 1。将 max_P 设置为 1 时，会丢弃所有到达的分组，这是不必要的因为即使当 qave 比 qth_max 大时，仍有空间来排队分组。

算法实现

现在我们检查决定是否入队或丢弃（标记）一个分组的代码段。

```
1 if (++q->qcount) {
2 if ((q->qave + q->qth_min)>=q->Wlog)*q->qcount + q->qR)
3 goto enqueue;
4 q->qcount = 0;
5 q->qR = net_random()&q->Rmask;
6 sch->stats.overlimits++;
7 goto mark;
```

在代码段中，第 2 行判断到达的分组是否入队列。qR 是一个 0 → Rmask 的整型随机变量，这里 Rmask = 2^{Wlog}。在实现中，为了确保所有算法只使用移位操作，qave、qth_max 和 qth_min 是固定的浮点数，存储在 Wlog 中，即它们的实际值等于它们目前的值除以 2^{Wlog}。明显，第 2 行不同于上面所述的概率方程。为了了解第 2 行如何实现方程，我们首先忽略变量 qcount。下一步，因为在实现中 max_P 很被随机选择为

$$\frac{qth\_max - qth\_min}{2^{Wlog+Pb}}$$

我们可以为 Pb 重写方程为

$$Pb = \frac{(qave - qth\_min)}{2^{Wlog+Pb}}$$

然后，如果 $Pb < \frac{q->qR}{2^{Wlog}}$，这个分组就会入队。这里不等式的右边是根据第 5 行对 qR 定义的一个

0～1的随机变量。最后，在不等式两边乘以2^{Wlog}，选择列第2行的表达。这里我们通过采用解释qcount结构的。第2行中qcount的值是0，当qave第一次跨入到两个阈值之间并被计为1表明qave离开了这个范围为止。因此，它确保qave值落入这个范围后第一个分组到达后就可以插入队列。

在RED中，另一个关键设计是计算平均队列长度qave，这是实际队列长度的指数平均，即

$$qave = qave \times (1-w) + sch->stats.backlog \times w$$

sch->stats.backlog是当前队列长度，w是在计算新值中一个旧的qave的权值，并将它设置为$1/(2^{Wlog})$。然后，由于sch->stats.backlog是整数，通过把它的结果为Wlog中固定浮点数，就像qave一样，清晰的分程实现定为：sch->qave=q->qave-(q->qave>>q->Wlog)+sch->stats.backlog。

例子

接下来，图7-28给出了一个说明RED操作的例子。令qth_min=1，qth_max=4，max_P=0.1，还有w=1。因此w表明qave总是等于当前队列长度。然后，如图7-28a所示，由于队列中没有分组，所以qave=0并且将丢弃概率Pb设置为0。然而，如图7-28b所示，在分组1和分组2插入队列后，qave增加到2，导致将Pb调整为0.033。在这种情况下，有些分组可能被丢弃，例如分组3。最后，如图7-28c所示，随着更多的分组插入，不可避免的qave超过了qth_max的界值4，这样就会启用一个更高的Pb值0.1，也就是说，更多的分组可被丢弃，例如分组8和分组10。

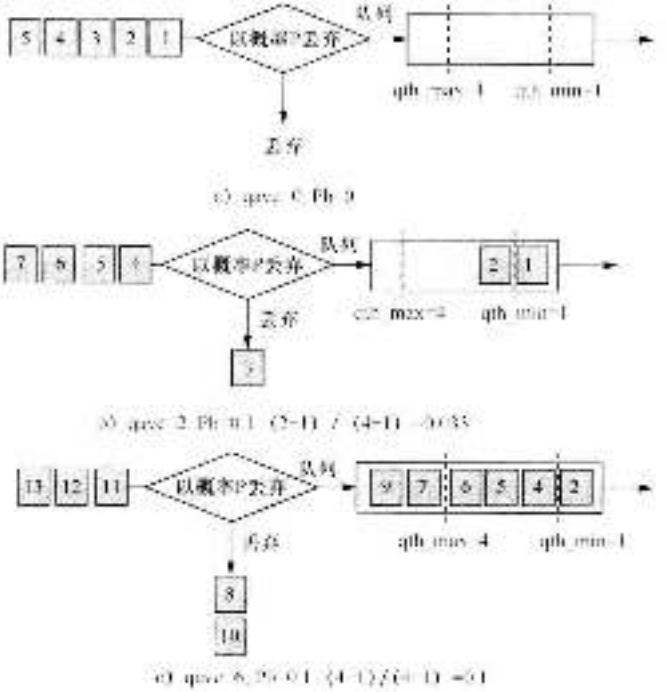


图7-28 RED的操作例子

练习

从/net/sched/中，你可以找到一个RED变种，称为可调RED早期检测（GRED），在sch_gred.c中实现。设法说明它是如何工作的以及与RED有何不同。

行动原则：目前日常使用的服务质量组件

虽然IntServ和DiffServ从未没有在全球范围部署以实现端支持QoS的互联网，但有些服务需要经过自治域在互联网的局部范围内。它们中没有一个是控制平面的组件，因为它们需要使用传统互联

网关协议的改进版本的 RSVP 协议。

这里让我们回顾一些实例。流标识或分类在防火墙中用于实现访问控制，但这是为了安全而不是为了服务质量的目的，这里是把 IP 字段或端口号会被禁止。令牌桶用于以太网交换机将经常被视为在给定于某些交换机端口的链路容量。分组调度常用于嵌入式操作系统上的带宽管理，不是完全消耗大量带宽的 P2P。在许多路由器上使用分组丢弃或队列管理是为了减轻拥塞和避免重载的分组丢弃。总之，尽管 IntServ 和 DiffServ 可能成为历史，但是它们的许多技术组件都流行起来。

7.4 总结

本章开始介绍了构建服务质量网络所涉及的 6 个关键组件，3 个控制平面组件，信令协议，QoS 路由和许可控制负责路由器之间资源的协商，确定一个保证服务质量的路径和控制网络的负载。两个数据平面组件，监管器和调度器，组成第三个数据平面组件——分类器，将分组分为不同的队列，从而控制转发时间和数列分组的顺序。

利用这些补充作为构建模块，IETF 提出了两种互联网服务质量体系结构：IntServ 和 DiffServ，以便构造一种能够支持服务质量的网络。集成服务是一种昂贵的解决方案，或许它可以提供带宽和延迟保证的虚拟私有网络。相反，区分服务是一种实用的体系结构，但只为不同层次的用户提供了区分服务。在介绍了支持服务质量的 IP 网络的体系结构之后，我们将详细说明构建一个支持服务质量网络组件的技术。这比体系结构有更多的研究问题。尽管集成服务和区分服务不能得以部署，但还是有许多 QoS 组件在我们的日常使用中得以运行，尽管是在一个相当有限的范围内。

常见陷阱

整形和调度

尽管这两种操作都能调节流的吞吐量，但它们的目的是不同的。整形的目标是改变或强制一个流的吞吐量的平均值或方差（偏差），以确保通过整形器后符合流的轮廓。操作仅针对一个流。然而，调度通常用于一组流争夺一条有限带宽的链路。调度负责为这些流分配带宽，根据预定义的策略，如均等或按比例共享。利用一组整形器替代调度器后果会如何？这真的是一个坏主意。第一，运行一组整形器比调度器花费更多的硬件资源。第二，由于在这些整形器之间没有通信，所以很难以实现的方式来使用的可变分配给更需要的流。

WRR 和 WFQ

因为 WRR 和 WFQ 是具有代表性的两种任务调度算法，所以再次强调它们之间的差异是很重要的。WRR 是一种为一组流调度分组的简单的方式。给每个流队列分配一个权值，调度器的作用仅以循环的方式服务于这些流，然后，当服务一个流时，允许发送分组的数量是其权值的倍数。虽然 WRR 的概念实现很简单，但当流添加时每个流可能会等待很长的时间才能得到服务，尤其是如果流中的分组错过了自己的发送机会时无法赶上，WFQ 的设计就是为了避免这个问题。它以动态顺序服务流队列，根据每个流的头分组中的时间戳排序。因此，在一个分组应该发送的时刻，WFQ 应该为它服务然后再服务于另一个流。由于在 WFQ 下无一遗漏，所以 WFQ 可以保证更短的最坏情况延迟给延迟和让 DRR 主导的公平性。

进一步阅读

QoS 架构和协议

这里引用的阅读资料中，首先，是一本专门介绍 QoS 的书。其次，是一份很好的有关 IP QoS 主题的专题论文。再次，介绍的是 IntServ 的总体思路和体系结构，而后两个分别说明它的保证服务和

控制负载服务。第6个是一篇有关RSVP很有用的辅导论文。第7个说明区分服务的体系结构，并且从第7个阅读资料中你可以学到IPv4头部中的ToS字段如何在区分服务中重新定义使用。下面的两个RFC描述了在DiffServ中转发的两种类型。最后两篇文献分别是与QoS在无线和Web服务的部署有关：

- Z. Wang, *Internet QoS: Architectures and Mechanisms for Quality of Service*, Morgan Kaufmann Publishers, 2001.
- X. Xiao L. M. Ni, "Internet QoS: A Big Picture," *IEEE Network*, Vol. 13, Issue 2, pp. 8 – 18, Mar. 1999.
- R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," RFC 1633, June 1994.
- S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," RFC 2212, Sept. 1997.
- J. Wroclawski, "Specification of the Controlled-Load Network Element Service," RFC 2211, Sept. 1997.
- L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource Reservation Protocol," *IEEE Network*, Vol. 7, Issue 5, Sept. 1993.
- S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.
- K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 3474, Dec. 1998.
- J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group," RFC 2597, June 1999.
- B. Davie et al., "An Expedited Forwarding PHB," RFC 3246, Mar. 2002.
- H. Zhu, M. Li, I. Chlamtac, and B. Prabhakaran, "A Survey of Quality of Service in IEEE 802.11 Networks," *IEEE Wireless Communications*, Vol. 11, No. 4, pp. 6-14, Aug. 2004.
- R. Pandey, J. Fritz Barnes, and R. Fritz Barnes, "Supporting Quality of Service in HTTP Servers," *Proceedings of ACM Symposium on Principles of Distributed Computing*, pp. 247-256, 1998.

QoS组件

下面是有关QoS组件的经典论文清单。前3个是许可控制，第4个是分组分类。接下来的5篇论文是有关调度的，最后两个是有关RED和AQM算法。

- R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 7, pp. 968 – 981, Sept. 1991.
- S. Jamin, P. B. Danzig, S. J. Shenker, and L. Zhang, "A Measurement-Based Admission Control Algorithm for Integrated Service Packet Networks," *IEEE Transactions on Networking*, Vol. 5, Issue 1, pp. 56 – 70, Feb. 1997.
- J. Qiu and E. W. Knightly, "Measurement-Based Admission Control with Aggregate Traffic," *IEEE/ACM Transactions on Networking*, Vol. 9, Issue 2, pp. 199 – 210, Apr. 2001.
- T. V. Lakshman and D. Stiliadis, "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching," *ACM SIGCOMM*, pp. 203 – 214, Oct. 1998.
- A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE/ACM Transactions on Networking*, Vol. 1, Issue 3, pp. 344 – 357, June 1993.
- M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *IEEE/ACM*

Transactions on Networking, Vol. 4, Issue 3, pp. 375 – 385, June 1996.

- D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Transactions on Networking*, Vol. 6, Issue 5, pp. 611 – 624, Oct. 1998.
- J. C. R. Bennett and H. Zhang, "WF2Q: Worst-Case Fair Weighted Fair Queueing," in *Proceedings of the IEEE INFOCOM*, Mar. 1996.
- J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in *Proceedings of the IEEE INFOCOM*, June 1994.
- S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, Issue 4, pp. 397 – 413, Aug. 1993.
- S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: Active Queue Management," *IEEE Network*, Vol. 5, Issue 3, pp. 48 – 53, May/June 2001.

Linux 流量控制模块

下面是有关如何在 Linux 环境下设置和配置流量控制模块的两个 Web 网页。

- Jason Boxman, "A Practical Guide to Linux Traffic Control," URL: http://blog.edseek.com/~jasonb/articles/tralfic_shaping/.
- Martin A. Brown, "Traffic Control HOWTO," URL: http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html.

常见问题解答

1. 在互联网中要供服务质量保证需要什么样的控制平面和数据平面机制？
   答：控制平面：信令协议、许可控制、QoS 路由。
   数据平面：分类、监管策略、调度。
2. WFQ 与 WRR 对比。（比较其复杂性和可扩展性。）
   答：复杂度：WFQ（O（log n））> WRR（O（1））。
   可扩展性：WRR > WFQ。
3. 为什么 RED 比 FIFO 更好，尤其是应用于实时流时更是如此？
   答：RED 避免突发性丢弃，即连续的分组丢失，也就是发生在 FIFO 上的情况。实时流往往有冗余和分层的编码，这里只要丢失是分散的，即不连续的，那么数据就是可以恢复的。
4. 将 Int Serv 与 Diff Serv 进行对比。（比较它们的 QoS 粒度，在边缘路由器上的复杂性，在核心路由器的复杂性、可扩展性。）
   答：QoS 粒度：IntServ（每个流）> DiffServ（每个类）。
   在边缘路由器上的复杂性：DiffServ ≥ IntServ。
   在核心路由器上的复杂度：IntServ > DiffServ。
   可扩展性：DiffServ > IntServ。
5. 部署 IntServ 的障碍是什么？（至少列出 2 种障碍。）
   答：可扩展性（每个流的 QoS）、有状态的路由器、应用程序的 QoS 信令。
6. 部署 DiffServ 的障碍是什么？（至少列出 2 种障碍。）
   答：有状态的边缘路由器、应用程序的 QoS 信令，或带宽代理。

练习

动手练习

1. RSVP 是一种为终端主机设计的信令协议，用来与路由器协商资源预留。从运行 TC（流量控制）模块的基于 Linux 的 PC 上发送 RSVP 请求，并使用 Wireshark 进行捕获。然后看看是否理解请求中每个字段值的含义。

2. 假设有一个路由器在三个链路分别为 A 和 B。设置路由器来测量从 A 到 B 流量的平均速率，以字节/秒为单位。测量应该按照每分钟为一次为基础。最好能够实时地将测量显示在一张列上，使得网络管理员可以通过网页浏览器远程地检查它。
3. 继续练习 2。在路由器上实现一个许可控制器，当流量从 B 到 A 传送的 TCP SYN 请求的 TCP 连接建立，然后当从 A 到 B 发送 B 到 A 的平均每秒最接近链路 B 的容量的 90%时，让路由器产生过滤这种请求。
4. 作为练习 3 的延续，不可只是建立错误，等待可能放在路由器上的 TCP 重传信息来通知请求者发送失败？
5. 在 Linux 系统上建立令牌桶以便将网络适配器从最大的出速率减为一个更小的值。然后通过测量 FTP 连接的吞吐量来观察减慢效应。
6. 假设你是一个互联网服务供应商，打算为你的商业客户在他们的接入链路上提供带宽管理服务。你的一个客户希望为研发组（R&D）预留 50% 的 T1 带宽。建立一个 QoS 感知的 ISP 一侧的边缘网关来满足这一 SLA。怎样完成你的设备需要识别进入的分组分成两类。第一类应该发往 R&D 主发送给 PC 的分组，而另一类则是其他分组。你可以在研发组 R&D 中的一台 PC 上观看在线流媒体，来证明完全不受到其他在 PC 上安装下载软件的影响。以此类证明该服务是可行的。
7. 首先安装用于外出链路非队列管理的 RED。然后，通过链路建立多个长寿命的 TCP 流并记录它们的总吞吐量与使用 FIFO 链路管理器等可量进行比较。此外，实际观察在开源实现 7.6 中实现的 qcow 与实际队列长度之间的区别。

书面练习

1. 第 7.1 节所述，支持 QoS 的路由器需要有 6 个基本组件。画出一图说明一个 IntServ 路由器的 6 个组件和它们之间的基本关系。当然，允许添加额外的组件。
2. 假定流量由一个带有参数 (r, p, B) 的令牌桶来典节。讨论令牌桶能造成的影响。例如，调整后流量的行为如何？或者，如果其参数中的一个参数、行为如何变化？
3. 在基于测量的许可控制中，介绍了两种使用的流量估计方法。一个是 EWMA，另外一个是时间窗口。进一步比较两者在估计上的差异。
4. 有一个 10 Gb/秒的链路并采用 WRR 进行调度。假设该链路由 N 个分组大小为 125 字节的流共享。假设按照为其中一半的流均匀地分配 8 × 10 Gb/秒的带宽，剩下的带宽为另一半流分配。然后，如果 N − 1 个流是活动的和无活动（backlogged），那么非活动流的第一个分组可能承受的最大延迟是多少？
5. 一般来说，WRR 适用于分组大小是固定长度的网络，而 DRR 是一种改进版本，可以处理各种长度的分组。事实上，由于它的实现简单，DRR 更受欢迎，但它不能保证小的最坏情况延迟。研究它们最坏情况延迟保证的能力，是否可能改进 DRR 来提供比 WRR 更小的最坏情况延迟？
6. 在最初的 RED 算法中需要周期性地跟踪队列长度和计算平均队列长度，这会给实现带来很大的负担。在 BSD 中，提供了更好的技术以便减小负担。你应该观察库文件 subr_prof.c 中的源代码并尝试画出流程图，描述实现是如何解决问题的。
7. 在一条接入链路中，下行链路流量的带宽管理是在 ISP 一侧的边缘路由器上实现的。然而，如果供应商本想提供这样的服务，那么在客户一侧的边缘网关上要实现服务是否可能或者有意义？证明你的答案。
8. 找到一个在其网络上成功地部署支持 QoS 路由器的营业案例并为它们的客户提供 QoS。
9. 排队管理算法 RED 的提出是为了缓解路由器中的拥塞。在许多研究报告中已经分析和改进 RED 算法。这些文件主要解决什么问题？研究群体为什么对此这么有兴趣？RED 广泛部署在互联网的路由器上了吗？
10. 列举一个企业所需要的典型流量类，然后用分组丢失、带宽、延迟和抖动确定每个类的 QoS。
11. 在过去的二十年里，提出了许多流量的调度算法，但它们中的大多数都是专为有线网络设计的。若将这些算法应用在无线网络中为用户提供 QoS 时，决策它们的需求是困难。
12. 描述使用 WFQ 来实现带宽管理的优点和缺点。
13. 图 7-17 中说明了一个令牌桶是如何操作的。假设为 $r = 1$ 单位/s，$p = 4$ 单位/s，$b = 20$ 单位并且在第 15-s 没有分组到达。然后，在 15-s 到来有 4 个分组到达，长度分别等于 2、2、10 和 4 个单位。采用上述假设，计算释放 4 个分组的时间。

第8章

Computer Networks: An Open Source Approach

网络安全

因为在互联网上发生许多安全故障，特别是近几年更是如此，所以网络安全已经成为一个非常重要的问题。计算机科学家并不是唯一对这些事件感兴趣的人，一般公众也会经常从新闻中了解到。当动称的互联网用户的个人资料受到攻击并被用于恶意目的时，他们就可能成为这些意外事件的受害者。

安全问题考虑起始于数据安全，然后再将重点放在访问安全上，而最近则是系统安全。第一个问题是有关在公共互联网上传输的私人数据保护问题，使数据不会被窃听或伪造。第二个问题是基于一个组织或互联网服务提供商的策略对内部和外部网络访问的问题。也就是说，它决定谁可以访问什么内容。最后，第三个问题是保护网络和系统，使它们不容易受到来自互联网的攻击。在本章中，我们讨论主要问题以及涉及解决它们所包含的设计问题。

在互联网上传输的数据是不安全的，因为它们很容易在途中被捕获、分析。数据包含的内容可能会泄露给第三方，在途中能悄被修改，或者被一个恶意源进行伪造。因此，数据安全的关键是如何保护数据防止被偷窥、修改或伪造。首先，数据应该加密或乱码密文，这样仅有接收方才能将其解密为原始数据。加密和解密机制必须是使第三方非常困难和耗费大量时间才能破解的，但是发送方和接收方要能足够有效地实现。其次，传输的数据必须得到认证以证明内容的完整性。

一个组织或互联网服务提供商可能要根据自己的策略适当控制其对互联网上网络资源和系统的访问。因此，访问安全性的关键问题是谁可以访问什么内容。确定“谁”可以访问的信息可能位于分组的网络层和传输层头部，如其地址、端口号和协议，也可能在分组的有效载荷中，如URL网址。管理可以在内部网络和外部网络之间部署多文墙或过滤设备，并且用一套规则来执行控制策略。这些设备检查进入或外出分组中的信息，这可能是在网络层、传输层或应用层，以便确定是否传递或丢弃它们。

最后，让我们看看系统的安全。攻击者出于各种目的可能会尝试寻找和利用系统的漏洞，如窃取关键信息、控制该系统发动另一次攻击、禁用重要的服务等。系统安全的关键问题是如何识别各种类型的入侵并保护系统防御它们的攻击。识别包括检查攻击特征或发现异常行为。检查特征可能漏掉未知的攻击（即漏报），但异常分析可能导致误报正常传输的行为异常，所以在误报和漏报之间必须做出权衡。因此，一定要设计精确的识别方法以便最大限度地减少误报和漏报。

8.2节、8.3节和8.4节分别介绍数据安全、访问安全和系统安全。在8.2节，我们提出几种加密算法说明它们如何设计用于保护数据，以及它们是如何在互联网协议和体系结构（如安全套接字层（SSL）和虚拟专用网络（VPN））中实现的。我们将3DES加密算法（VHDL硬件描述语言）、MD5认证算法（在Linux内核中）和虚拟专用网络（在Linux内核中）的开源实现穿插在书中介绍。

在8.3节中，我们介绍了实现了访问控制的防火墙和过滤设备的各种类型。Netfilter/iptables和防火墙工具包（FWTK）分别作为网络传输层和应用层防火墙的开源实现给出。在8.4节中，我们学习常见的攻击技术和恶意程序，然后详细说明如何区分它们，同时也考虑了设计问题和权衡。著名的ClamAV和Snort分别作为杀病毒和入侵检测的开源实现。还学习了另外一个用于反垃圾邮件的开源软件——SpamAssassin，由于它与ClamAV和Snort很相似，它们都非常依赖于特征的检查。

8.1 一般问题

如上所述，安全议题包括数据安全、访问安全和系统安全。数据安全的基础是密码学。我们首先介绍密码算法及它们在保护公共网络上私人数据传输中的应用。对于访问安全，防火墙系统是最受欢迎的设备。然后介绍了不同类型的防火墙和它们的工作原理。如今的互联网容易遭受各种网络攻击。因此，我们研究在关各种攻击的问题以及如何保护系统免受攻击。

8.1.1 数据安全

随着电子交易越来越普及，发送银行业务数据、密码、信用卡号码等敏感数据的安全问题已经变

得很严重。这类敏感数据可能被截获被用于记录、分析、重放或欺骗的目的。解决这种问题上有机械性。如果网络安全没有得到保证，那么很少人会为上述目的而使用网络。

在本章中，我们解释密码操作并列举了3个虚拟人物：Alice（代表发送方A），Bob（代表接收方B）和Trudy（代表入侵者T）。例如，当Alice把无保护的明文数据发送给Bob时，中间人Trudy可以轻松地阅读和收集他们之间的明文，并可以重放、修改或伪造这些数据。当假数据到达Bob时，他认为它们是从Alice传送来的。明文数据应在发送前进行加密。我们首先讨论加密和解密技术。

我们从传统密码学理论开始。最早提出来的是对称加密算法或单密钥加密系统。它使用一个公共密钥来加密和解密数据。由于必须交换公共密钥，所以以有效的发布方式发布密钥很关键。1976年，Diffie和Hellman提出了非对称加密方法。此方法利用不同的密钥加密和解密数据，故得名非对称加密。因此，在网络上发布密钥变得简单而安全。对应这两种加密系统开发了多种有代表性的系统。例如，数据加密标准（DES）和国际数据加密算法（IDEA）是基于对称加密算法的，而RSA（取自于三位发明者姓氏的首字母缩写）是基于非对称加密算法。

假设发送方Alice和接收方Bob位于不同的地点。因为他们无法面对面直接识别对方，所以他们必须相互认证以确定通信中彼此的身份。他们还需要确保收到的数据与原始数据是相同的，而没有在网络交易中修改、欺骗或恶意伪造。本章将详细描述数字认证和保证数据完整性的技术。

介绍加密算法后，我们将介绍链路层、网络层和传输层的网络协议如何在加密的基础上实现网络安全。除了链路层的隧道协议外，因特网安全协议（IPSec）工作在网络层。在网络层运行安全协议有几个优点。首先，不仅是运行在TCP之上的应用，还有其他应用也可以拥有IPSec提供的安全机制。其次，它不容易受到TCP之上的常见攻击。如伪造RST以断开连接。IPSec在IP网络层支持两种类型的安全协议：认证头部（AH）协议和封装安全载荷（ESP）协议。前者提供源节点的认证和数据的完整性。后者支持完整的身份认证、数据完整性和安全机制，因此它比AH更复杂。IPSec协议及其应用、虚拟专用网络（VPN）将在本章中详细描述。

安全套接字层（SSL）协议用做一种传输加密数据的安全机制。该协议广泛应用于安全的网页浏览和安全的邮件交付等应用中。虽然SSL对于数据通信是安全的，但是它仍然不能为电子商务中通过信用卡的在线支付提供一个完整的安全环境。接收方有机会在其他地方接到信用卡信息。因此，我们介绍安全电子交易（SET）标准，并解释其操作。

8.1.2 访问安全

在外部网络和内部网络之间的边界上实施访问控制对于网络安全很重要。访问控制一般是双向的。限制从外部到内部的访问可以保护内部网络中的主机不被非法访问，而限制从内部到外部的访问通常是基于策略考虑的。例如，一个组织可能不想让关工在工作时间内访问任何外部FTP站点，因此与发往常用FTP端口（即端口21）的流量直接被组织的过滤设备丢弃。

防火墙系统根据安全策略中的规则过滤进入和外出的分组来强制实施访问控制。它们既可以允许也可以拒绝与规则中分组信息相匹配的分组。因此，快速规则匹配对于防火墙的性能非常关键。如果匹配不够快，那么防火墙本身就会成为瓶颈。

防火墙系统通常有两种类型：基于分组过滤的防火墙和基于应用网关的防火墙。前者通过查找网络层（简称为L3）和传输层（简称为L4）头部中的字段过滤分组，因此它工作在网络层和传输层。因为这些字段通常是在固定位置的几个字节，所以过滤速度快。但是访问控制仅查看L3和L4的信息可能是不够的，因为它可能将服务隐藏在众所周知端口以外的端口。例如，如果FTP服务是在1234端口提供的，然后阻塞对端口21的访问就起不到作用。因此，如果要求精确过滤，那么查找应用特征有时是必要的，这是指基于网关的防火墙。此外，如果策略与分组内容相关，那么就必须检查应用信息。例如，两个允许和禁止请求的网址可能发向端口80。既不限制也不打开端口80也可以工作。解决办法是将请求与一套黑名单或白名单的URL进行匹配。

检查应用内容当然不会是免费的。首先，分组可能需要组装起来以便恢复应用内容用于检查。其次，防火墙可能需要保留连接状态来反映每条连接的状态。再次，扫描分组内容查找应用特征比匹配

分组头部固定字段更为复杂，因此性能问题就更加重要。8.3 节中介绍 2 个开源实现的例子：基于分组过滤的防火墙，Netfilter 和基于应用网关的防火墙，防火墙工具包（FWTK）。

8.1.3 系统安全

协议、软件和系统的设计缺陷导致出现了黑客可以利用的安全漏洞。黑客可能入侵一个脆弱的系统窃取秘密信息，使系统崩溃，获得系统管理员权限以及传播恶意程序。为了系统的安全，这些恶意的行为必须得到有效检测和控制。鉴于每天有大量的网络流量，网络管理员不可能手动检查流量日志文件以查看发生了什么。因此需要部署入侵检测（或防御）系统来分析流量来检测攻击。它可以产生指示攻击类型及相关信息（例如，攻击的来源）的警报，也可以在检测到攻击时就加以阻塞来防止攻击。检测可能是基于一套规则，该规则描述或签名已知攻击的攻击场景或特征，或者寻找流量中的异常。前者可能忽略未知攻击（所谓的漏洞），因为没有规则可以描述这种攻击。后者可以发现未知攻击，但也可能更容易产生漏洞，因为正常流量可能会表现得像异常流量。这两种技术之间要有一个权衡。此外，入侵检测通常是用于取证，因为它只监测流量却没有阻止攻击。如果系统太慢而来不及处理流量，那么它就会丢弃一些分组，这就可能失去了对少数可能攻击的警报，但是这不会损害正常的通信。另一方面，入侵防御系统部署在进、出流量的网关上，它能够阻止检测到的攻击，但是如果它的速度来不及处理流量，它就会成为系统性能的瓶颈，并且双向通信也可能会懵下来。

网上疯狂传播的恶意软件也称为恶意程序，它已成为互联网的一个主要威胁。这些程序可能对受感染的系统造成损害，如会使服务崩溃或机密信息被窃取。它们可能看起来与其他正常程序没有什么区别，甚至在用户没有察觉的情况下潜入。检测它们的一个常用方法是扫描这些恶意程序的特征代码，但是随着它们将自己以各种方式隐藏起来而使检测变得困难。例如，它们可以用各种密钥加密代码，仅在运行它们时才解密代码。因此，扫描可执行的特征匹配就变得无效。一种可能的解决方案是实际运行恶意软件并观察其行为。然而，恶意软件可能配备了多种反检测机制，如检测到存在分析程序并发现有人试图分析它时就假装表现良好。我们也将在 8.4 节中研究这些问题。

8.2 数据安全

8.1 节中讨论了密码学的演变及其应用。本节首先深入研究密码法的原理。对于对称密钥系统，我们介绍数据加密标准（DES）、三重 DES（3DES）和高级加密标准（AES）。对于非对称密钥加密系统，我们介绍 RSA（Rivest，Shamir 和 Adleman）。接下来，我们介绍用于认证的消息摘要算法 5（MD5）。最后，介绍如何将密码学应用到虚拟专用网络（VPN），包括链路层的点到点隧道协议（PPTP）和第 2 层隧道协议（L2TP）；网络层的 IP 安全（IPSec），以及传输层的安全套接字层（SSL）和安全电子交易（SET）。开源实现的例子是 3DES、MD5 和 IPSec。

8.2.1 密码学原理

图 8-1 显示数据加密和解密的过程。为了实现数据安全性，Alice 在传输之前加密数据。虽然 Trudy 截获了加密数据，但就不能得到原来的明文。因此，数据加密保护了原始文本的机密性。Bob 接收到加密的数据后，利用解密密钥就能够再解来自 Alice 的明文。



图 8-1 数据的加密和解密

对称密钥系统采用一个共同的密钥来加密和解密文本，而在非对称密钥系统中加密和解密的密钥是不同的。由于 Alice 和 Bob 共享一个密钥，所以密钥必须以安全的方式从一个人发送到另一个人，否则加密数据以会泄漏给未经授权的人。简单地加密密钥然后再发送会面临与数据加密相同的问题。密钥分发中心（KDC）可以为用户提供注册并分享他们的密钥服务。使用 KDC 的著名例子就是 Kerberos 协议，其中工作中的 2 个实体可以彼此标识。然而，KDC 本身必须是可信的，它可能成为一个单点故障。因此，利用非对称密钥系统作为解决方案。加密和解密密钥是配对的。Alice 使用配对中的一个密钥加密数据，而只有 Bob 拥有另一个密钥来解密。因此，就不需要将 Bob 的密钥分发给 Alice。虽然这个系统看起来很理想，但关键问题是加密和解密的计算很慢。因此在实践中，非对称密钥系统用于对称密钥系统中密钥的分发。两个对等拥有了对称密钥后，它们就可以使用密钥传送大量的数据。安全和效率从而得到了平衡。

对称密钥系统

对称密钥系统的一个著名例子是美国政府在 1977 年采取的确保数据安全的数据加密标准（DES）。DES 使用一个 56 位对称密钥用于加密和解密。国际数据加密算法（IDEA）也使用对称密钥系统。目前，56 位 DES 算法仍然很受欢迎，虽然可以使用一种更安全的系统，即 112 位的 DES 算法，但仅限于美国。

56 位 DES 通过一个 56 位密钥加密每个 64 位的数据块单元，并且得出单字母的结果；因此，如果 DES 使用相同的密钥加密相同的明文数据，就将获得相同的加密数据。DES 的操作包括置换密码和替代计算中重复 16 次迭代的密码。图 8-2 显示了 DES 操作原理，其描述如下：首先，将明文分割成 64 位的数据块。每个块 $T = t_1t_2\cdots t_{64}$，执行初始移位获得 T_0，T_0 是 $t_{58}t_{50}t_{42}\cdots t_{23}t_{15}t_7$，从而形成两个 32 位的块，$R_0$ 和 L_0，如下：

$$T_0 = L_0R_0$$

这里

$$L_0 = t_{58}t_{50}t_{42}\cdots t_{15}t_8$$
$$R_0 = t_{57}t_{49}t_{41}\cdots t_{15}t_7$$

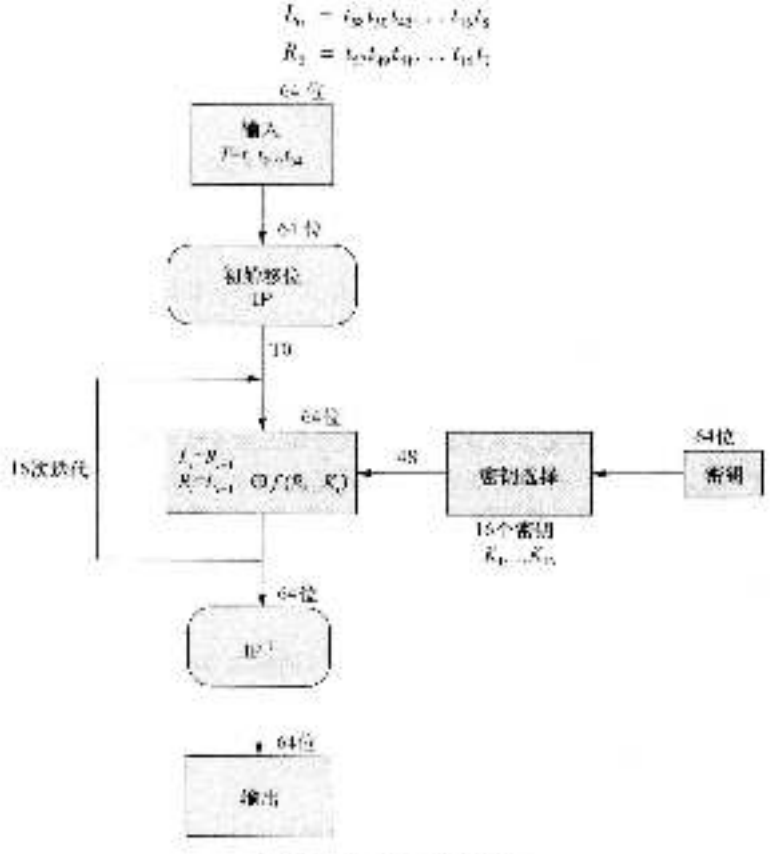


图 8-2 DES 的加密过程

L_1和R_1的数据块作为下一次迭代的输入

$$L_1 = R_1$$
$$R_1 = L_0 \oplus f(R_0, K_1),$$

其中，K_1是从 56 位密钥中导出的。

此后，结果就变成$T_1 = L_1R_1$。将 56 位密钥首先计算为 16 个 48 位密钥：K_1，K_2，…，K_{16}。图 8.3 显示了$f(R_0, K_1)$的处理过程，其中 32 位的R_0和 48 位的K_1是加密输入。首先，32 位R_0通过$E(R_0)$操作扩展获得一个 48 位的结果。其次，48 位$E(R_0)$和 48 位K_1都执行 XOR 操作来获得一个 48 位的结果，将它分割为 8 个 6 位输入，B_1，B_2，…，B_8用于下面的替代。

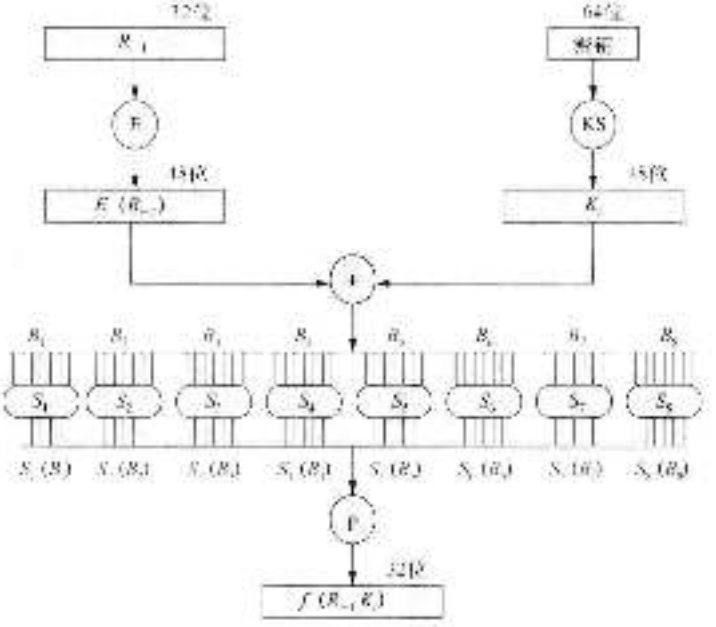


图 8.3 $f(R_{i-1}, K_i)$的计算过程

S_i替代后，就获得 8 个 4 位的块，$S_i(B_i)$。然后计算结果执行一种 32 位置换的操作得到$f(R_0, K_1)$，最后R_1也可以通过$L_0 \oplus f(R_0, K_1)$来获得。

重复 16 次同样的迭代，即$L_{i+1}R_{i+1} \leftarrow L_iR_i$，$i = 0, \ldots, 15$，获得$T_{16} = L_{16}R_{16}$。这个计算将进行逆初始换位获得 64 位加密的数据。将加密过程逆向，就可以从加密数据中解密出明文。

RSA 数据安全公司曾经提供 10 000 美元奖励给任何能够从经过 56 位 DES 算法加密的文本中解码出明文的人。后来一个团队在 4 个月内解密了加密数据。另一个人在 1999 年最新一能解密 DES 挑战中在 22 个小时内解密。因此，如果数据是非常重要的，利用 DES 加密是不安全的。对于普通的应用，可以认为它是足够安全的。多重 DES 算法比单 DES 系统更安全，因为攻击者可能需要在重复迭代中找到更多的密钥。例如，美国政府推荐三重 DES（3DES）和 128 位 DES 算法作为美国统一的加密和解密标准。假设K_1、K_2和K_3是 3 次迭代中的 3 个密钥。加密和解密的过程是

加密：$E_{k3}(D_{k2}(E_{k1}(P))) = C$

解密：$D_{k1}(E_{k2}(D_{k3}(C))) = P$

其中P是明文，C是密文，E是 DES 加密算法，D 是 DES 的解密算法。所有的过程都由 DES 运算组成，并能重用现有的 DES 函数块实现。开源实现 8.1 描述一个 3DES 实现。

开源实现 8.1：硬件 3DES

概述

这里我们主要介绍 3DES 的硬件实现，因为它是一种加速 3DES 计算的常见方法。3DES 计算涉及大量的位级操作，如替代和置换（参见对 DES 计算的介绍）。如果 3DES 计算采用软件实现，那么每个

操作将采取多条指令来完成，因为写的操作数通常是包括多字节的字。此外，硬件实现方法在数据块上进行地操作，因此在硬件上实现3DES计算会更加自然。

在OpenCore网站上的开源项目（www.opencores.org）致力于从VHDL语言来实现3DES。我们利用该项目作为例子来演示开源实现。它支持3个64位的DES密钥，并与NIST的FIPS46—3标准兼容。由于3DES可算3次DES，所以软硬件设计的每个阶段的组件被相到DES运算的功能模块上，如S盒用于替代，而P盒用于置换。被引入的类，第次使用完整的公号会比实际有面临到的操作。

框图

图8-4说明了主要功能模块。DES计算的3个模块是级联序列的，利用3个不同的密钥进行计算的。每个模块的数据输出是下一个模块的输入。先每个模块内部是密钥扩展、替代、置换等函数模块。它们之间的关系可以从图8-2和图8-3中的计算流程图表。



图8-4 在3DES模块中的函数模块

数据结构

在主函数模块接口中的信号在表8-1中列出。

注意，即使每个密钥的长度是64位，但8位用于校验并在加密/解密计算有关中。每个密钥实际上只有56位。

表8-1 在主函数模块接口中的信号

信　号	方向	描　述	信　号	方向	描　述
KEY1_IN [0:63]	IN	第一个64位密钥	LOADATA	IN	指示数据准备好
KEY2_IN [0:63]	IN	第二个64位密钥	RESET	IN	重置为初始状态
KEY3_IN [0:63]	IN	第三个64位密钥	CLOCK	IN	同步时钟输入
FUNCTION_SELECT	IN	加密或者解密	DATA_OUT [0:63]	OUT	64位加密/解密数据
LDKEY	IN	指示密钥准备好	OUT_READY	OUT	输出数据准备好

算法实现

一旦初始化，主函数块（des top.vhd）设置下一个状态WaitKeyState，并读取FUNCTION_SELECT来确定是执行加密或解密。然后模块等待密钥直到它们可用（LDKEY = '1'），进入WaitDataState，其中模块等待数据，直到它们可用（LOADATA = '1'），用于初始化的代码如下所示。

```
if reset = '1' then
    nextstate          <= WaitKeyState;
    ldcount_internal   <= '0';
    out_ready          <= '0';
    ld_internal        <= function_select;
    ld_internal_inv    <= not function_select;
else
    data_out           <= data_out_internal;
    out_ready          <= out_rdy_internal;
    case nextstate is
      when WaitKeyState =>
        if ldkey = '0' then
          nextstate    <= WaitKeyState;
        else
        -- read keys here, the codes not shown
        nextstate      <= WaitDataState;
        end if;
```

```
        when WaitDataState =>
          if ldata = '0' then
            nextstate          <= WaitDataState;
            ld_data_internal   <= '0';
          else
          // read data here; the codes not shown;
          end if;
    end if;
```

在设计中有3个DES模块（在模块中名为des_cipher_top），每个分别读取3个密钥。DES的操作是在e_expansion_function.vhd,p_box.vhd和s_box.vhd中实现。例如，替代操作进一步分为8个模块，从s1_box.vhd到s8_box.vhd，因为有8个这样的替代（如图8-3所示）。在s1_box.vhd中替代的代码段如下所示。

```
begin
sbox        <= "0110" when A = x"0" else
              "0000" when A = x"1" else
              "0100" when A = x"2" else
              "1111" when A = x"3" else
              "1101" when A = x"4" else
              "0111" when A = x"5" else
              "0001" when A = x"6" else
              ... // all combinations of 6-bit A's
end
```

为了简单，我们不深入理解每个模块，得知节留给读者自己学习。

练习

1. 如果它在软件中实现，指出设计中的哪些组件可能是饿效的。
2. 在代码中查找最初的56位密钥在16次迭代中的每一次是如何转化为48位密钥的。

尽管DES和3DES可以很容易地在硬件中实现，但由于其位级操作，所以它们在软件中的实现速度很慢。高级加密标准（AES）算法可以更好地在软件中实现。加密/解密过程中的替代和置换都是字节级的。操作更适合软件实现，因为交换两个字节或更换软件中一个字节的内容很容易。AES算法带有128位、192位和256位的密钥。除了AES的效率外，它比DES更安全，因为其庞大的密钥空间。即使只有128位的密钥也需要很大的努力才能破解它。因此，AES用于数据的安全是足够的。

非对称密钥系统

对称密钥系统的加密和解密使用相同的密钥，但密钥应该首先以一种安全的方法发布。非对称密钥系统（或公共密钥体系）利用一对密钥分别进行加密和解密数据。一个密钥是众所周知的，即公钥；另外一个密钥必须是私有的，即私钥。例如，在图8-5中，Alice使用Bob的公钥进行加密，Bob使用他的私钥解密加密过的数据。因为只有Bob有私钥，所以其他人就不能解密加密过的信息，即使公钥是众所周知的。此外，Alice也可以唯一地确定利用她的私钥加密数据，因为没有其他人具有她的私钥（参见8.2.2节）。

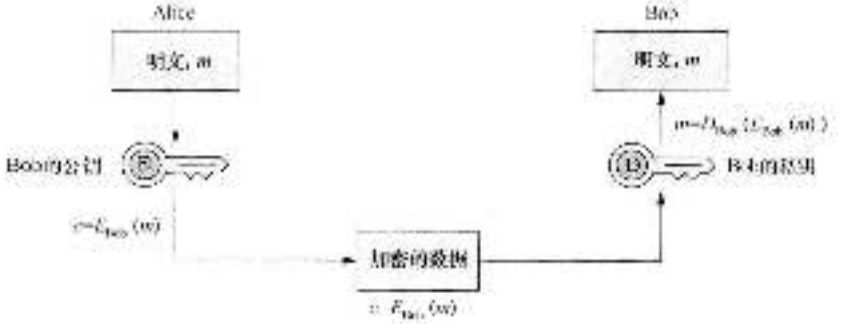


图8-5 非对称密钥加密

RSA是最著名的非对称密钥算法，它根据麻省理工学院3个教授Ronald Rivest、Adi Shamir和

Leonard Adleman 的名字命名的。虽然 RSA 解决了密钥的分配问题，但其计算复杂度较高，因此不适于加密/解密普通数据。因此，RSA 经常用于密钥的发布或身份验证，而利用对称密钥系统加密和解密大型数据。图 8-6 描述了 RSA 中选择公钥和私钥的过程：

1）选择两个非常大的参数，p 和 q，参数越大，就越难破解，计算时也将显著增加。RSA 实验室建议决定将 n 数要大于 10^{308}。

2）计算 $n=p\times q$ 和 $z=(p-1)\times(q-1)$，分别得到 n 和 z，即 $n=p\times q$ 和 $z=(p-1)\times(q-1)$。

3）选择一个 e 作为公共密钥，它要小于 n 并且与 z 互素。

4）计算出一个值 d 当为私钥，以使 $e\times d-1$ 能够被 z 所整除。

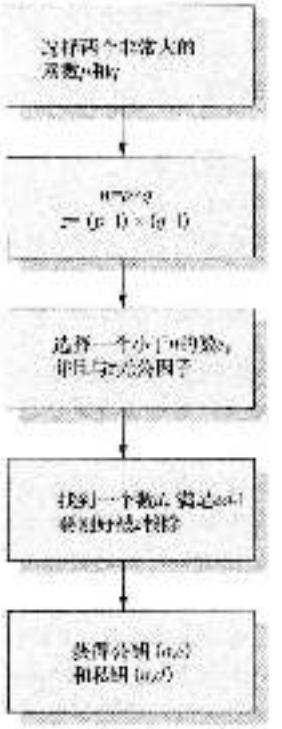


图 8-6 RSA 公钥和私钥的选择过程

因此，Bob 可以将公钥 (n, e) 分配给任何人，然后 Alice 可以使用此密钥来加密数据，而 Bob 可以使用自己的私钥 (n, d) 来解密数据。例如，Alice 想要向 Bob 发送明文 m，这里 $m<n$。Alice 首先计算 m^e，然后再除以 n 得到余数 c，这里 c 是密码或加密过的数据。一旦 Bob 收到加密过的数据 c 后，他计算 c^d 然后除以 n 得到余数 m，其中 m 是原始明文。下列方程总结了处理过程：

$c=m^e \bmod n$//使用公钥 (n, e) 将明文加密成密文数据 c；

$m=c^d \bmod n$//使用私钥 (n, d) 来解密加密过的数据，然后得到明文 m。

下一步，我们给出一个简单的例子来阐述处理过程。首先，Bob 选择 $p=11$ 和 $q=17$，然后计算 $p\times q$（$n=187$）和 $(p-1)\times(q-1)$（即 $z=160$）。其次，Bob 选择 23 作为 e，其中 e 是 z 的一个素数。最后，Bob 计算 $(e\times 1)/z$ 得到 $d=7$。因此，Bob 将公钥分给 Alice（$n=187$，$e=23$）。她使用公钥加密明文 m 得到加密数据 c。Bob 收到加密数据 c 后，能用自己的私钥（$n=187$，$d=7$）解密。

假设 Alice 将明文“clap”发送给 Bob。她首先将字符 A～Z 映射为数字 1～26，得到“c”=3，“l”=12，“a”=1，“p”=16。图 8-7a 显示了利用公钥（$n=187$，$e=23$）的加密过程，图 8-7b 显示了利用私钥（$n=187$，$d=7$）的解密过程。

明文	m	m^e	$c=m^e \bmod n$
c	3	94 143 178 827	181
l	12	5 624 576 424	177
a	1	1	1
p	16	4 363 [illegible] 77	166

a） Alice 使用公钥（$n=187$，$e=23$）加密明文“clap”的过程

加密的文本，c	c^d	$m=c^d \bmod n$	明文
181	6 [illegible] 2 612 [illegible] 45	3	‘c’
177	3 [illegible] 4 436 [illegible] 33	12	‘l’
1	1	1	‘a’
166	3 [illegible] 6 [illegible] 73 [illegible] 13	16	‘p’

b） Bob 使用私密钥（$n=187$，$d=7$）为解密过程

图 8-7

加密和解密过程有指数运算，导致较高的计算复杂度。软件实现 DES 的效率大约比 RSA 快 100 倍，在硬件中实现要快 10^3-10^4 倍。因此，大多数应用会将对称和非对称密钥系统结合起来使用。发送者 Alice 随机生成一个会话（对称）密钥将明文加密成密文 c。然后，她使用 Bob 的公钥（非对称）加密会话密钥，并将 c 发送给 Bob。当 Bob 接收到 c 后，他首先使用自己的私钥（非对称）来解密加密过的会话密钥，然后推导出未来的加密和解密用的会话密钥。因此，密钥分配过程是安全的并且产生有效率的数据传输。

在实践中，用户的公钥可以从认证中心（CA）得到，CA 是一个维护公钥的可信性和用户身份的组织。在这种机制下，用户首先必须注册他的公钥，而 CA 必须认真核实用户的身份；否则就很容易受到破坏。公钥和用户的身份信息将包含在由 CA 颁发的数字签名证书中。因此，其他用户就可以从数字证书验证用户的身份，并且确保公钥是真的来自那个用户。

行动原则：安全的无线信道

实质上，无线网络在广播帧。保护无线传输信道安全是无关重要的，否则任何人都可以很容易地嗅探到传输的流量。在无线局域网技术中，IEEE 802.11 配备了有线对等保密（WEP）标准，保护无线介质中的数据。在该标准中，数据流使用 RC4 算法加密，RC4 相当简单。在发射站，RC4 将数据流与密钥流并或生成密文流。在接收站，RC4 将加密流与完全相同的密钥流异或，以恢复原始数据。发射站和接收站可以同步密钥流，因为两者都使用相同的种子利用伪随机数发生器产生密钥流。

由于设计上的缺陷，WEP 不是很安全。2001 年 8 月，一篇由 Fluhrer 等人发表的题为“Weakness in the Key Scheduling Algorithm of RC4”的论文来攻击 WEP。简而言之，很可能从 WEP 密钥一部分的弱初始化向量（IV）中导出密钥流的内容。IV 值是以明文传输的，通过收集足够多的误，密钥流就可以从其中的弱 IV 值恢复出来。一个称为 AirSnort 的程序（airsnort.shmoo.com）公开提供这种攻击的实现。

为了解决这种严重的缺陷，Wi-Fi 联盟推出 Wi-Fi 保护接入（WPA）规范，它后来扩展成为 IEEE 802.11i，2004 年得到批准。在 802.11i 中，高级加密标准（AES）成为无线数据加密的方法，而 IEEE 802.1X 用做身份验证机制。这个新标准在许多新产品中提供，使得无线局域网成为更安全的环境。

8.2.2 数字签名和消息认证

除了数据加密外，Bob 还需要确保从 Alice 处发来的数据是真实的，因为入侵者 Trudy 可能会假装成 Alice 发送数据。在非对称密钥系统中，数字签名是最流行的身份认证方法。将数字签名应用到传输数据有 3 个优势：1）接收者能够确保数据确实是来自发送者的；2）发送者不能否认传输；3）没有人可以修改接收到的数据，从而得到了数据的完整性。

非对称密钥系统和散列函数可以用于实现数字签名。在图 8-8 和图 8-9 中，当 Alice 向 Bob 发送明文时，她就使用一个数字签名来认证自己。在图 8-8 中，Alice 首先计算明文，通过散列函数导出一个独特的散列值“12340782”，然后用她的私钥加密散列值，并发送加密文本“??!!??!!”，即 Alice 的数字签名与明文一起发送给 Bob。Bob 收到 Alice 的明文和数字签名后，利用 Alice 的公钥解密数字签名“??!!??!!”得到“12340782”的散列值，使用相同的散列函数计算明文以获取“12340782”的散列值。如果两个散列值是相等的，Bob 就可以肯定明文是由 Alice 发送的。换句话说，数字签名可以实现以下 3 个功能：

Alice 不能否认她已发送了此文件，因为她使用了自己的私钥加密了散列值。

Bob 不能修改接收到的文件，否则两个散列值将不同。

该文件没有被修改，因为它包含了相同的散列值“12340782”。

如果发送方使用密钥加密明文的散列值，同时接收方也使用相同的密钥解密加密过的散列值以便进行验证，加密的散列值称为消息认证代码（MAC）。这个概念类似于数字签名，但用来加密和解密

散列值的密钥是对称的。因此，接收方必须事先共享密钥。MAC 与消息认证码类似，但是可以提供数字签名的功能，因为任何知道密钥的人都可以生成其他消息的 MAC。

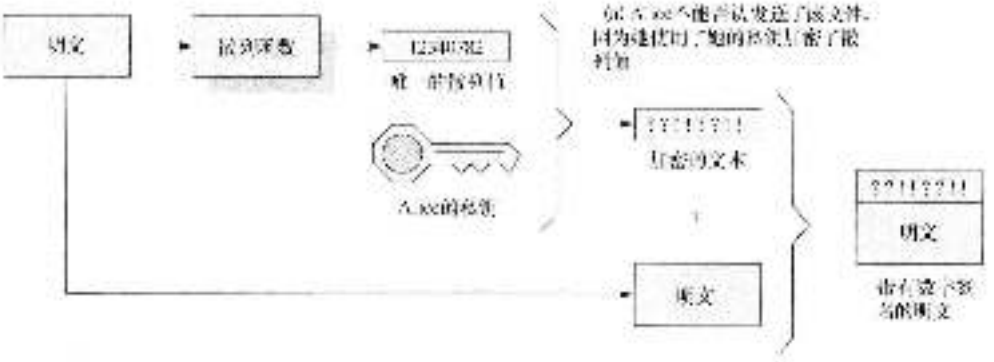


图 8-8 Alice 使用数字签名发送文件

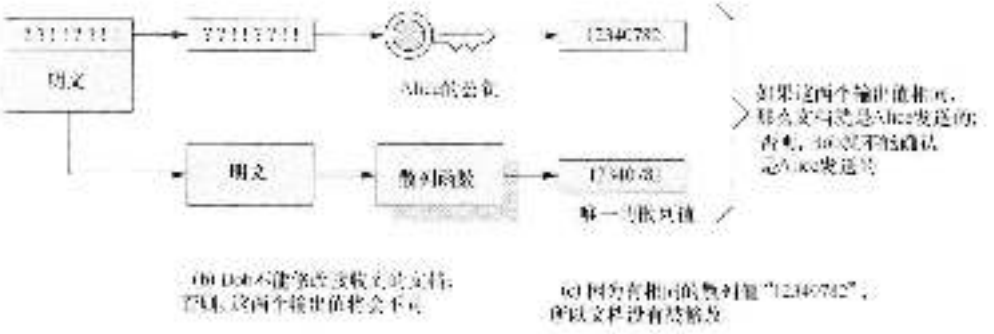


图 8-9 Bob 确定收到的带有数字签名的文件是否来自 Alice

我们已经提到，Alice 应该为相应的明文产生一个散列值。散列值称为消息摘要（MD），消息摘要的功能就是保证数据完整性。流行的散列函数包括 MD4、MD5 和安全散列算法（SHA）等。MD4 和 MD5 由 Ron Rivest 于 1992 年提出；MD5 是最普遍采用的算法，用来生成 128 位的消息摘要。美国政府使用 SHA－1，它可以产生 160 位的消息摘要，比 MD5 更强大。

开源实现 8.2：MD5

概述

MD5 常用于对算法安全应用中，对一给定数据是否被篡改进行检验。通过数据计算出其 MD5 值。MD5 的一个开源实现在 md5.c 中，位于广泛使用的 Linux 2.6.x 内核源码 crypto 目录下。每到算法可从任意消息开始读取 512 位后，MD5 算法不断从更新 128 位的状态（即 4 个 32 位字）。在最后一次迭代中，最后一批数据如果比 512 位短，就需要填充至 512 位。

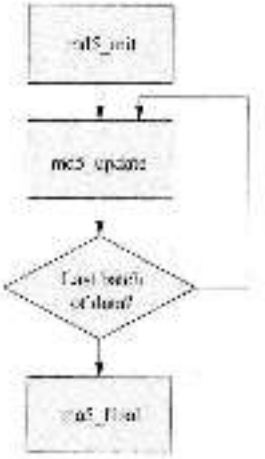


图 8-10 在 md5.c 和执行流中的主要函数模块

框图

在 md5.c 中的 3 个主函数分别为 md5_init()、md_updata()和 md_final()。首先，初始化 128 位的状态。其次，在每次迭代中不断更新状态。最后，从最后一批数据计算最终的 MD5 值。图 8-10 说明了 MD5 计算的执行流程。

数据结构

128 位的状态在 md5_init()函数中初始化，其中 md5_x 的构还包括辅助变量，如 block 数组和 byte_count 变量用于计算；

```
mctx->hash[0] = 0x67452301;
mctx->hash[1] = 0xefcdab89;
mctx->hash[2] = 0x98badcfe;
mctx->hash[3] = 0x10325476;
```

算法实现

初始化后，md5_update()函数按每批 64 字节（相当于 512 位）抓取消息源，并用下面的代码段进行 MD5 计算：

```
        const u32 avail = sizeof(mctx->block) - (mctx->byte_
count & 0x3f);
        mctx->byte_count += len;
        if (avail > len) {
                memcpy(char *(mctx->block +
(sizeof(mctx->block) - avail)), data, len);
                return 0;
        }
        memcpy(char *(mctx->block +
(sizeof(mctx->block) - avail)),data, avail);
        md5_transform_helper(mctx);
        data += avail;
        len -= avail;

        while (len >= sizeof(mctx->block)) {
                memcpy(mctx->block, data,
sizeof(mctx->block));
                md5_transform_helper(mctx);
                data += sizeof(mctx->block);
                len -= sizeof(mctx->block);
        }
        memcpy(mctx->block, data, len);
        return 0;
```

这里的 data 数组是将被转化为 MD5 值的源消息，消息的长度是 len。md5_update()函数尝试从 data 中读取数据，直到 block 数组（64 字节）填满为止。一旦数组被填满，就调用 md5_transform_helper()函数，它再调用 md5_transform()函数。

md5_transform()函数是 MD5 计算的主要组成部分（详情参见 RFC 1321）。该函数按顺序执行 4 个类似的 16 个操作的循环。例如，在第一次循环中的 16 个操作如下：

```
MD5STEP(F1, a, b, c, d, in[0] + 0xd76aa478, 7);
MD5STEP(F1, d, a, b, c, in[1] + 0xe8c7b756, 12);
MD5STEP(F1, c, d, a, b, in[2] + 0x242070db, 17);
MD5STEP(F1, b, c, d, a, in[3] + 0xc1bdceee, 22);
MD5STEP(F1, a, b, c, d, in[4] + 0xf57c0faf, 7);
MD5STEP(F1, d, a, b, c, in[5] + 0x4787c62a, 12);
MD5STEP(F1, c, d, a, b, in[6] + 0xa8304613, 17);
MD5STEP(F1, b, c, d, a, in[7] + 0xfd469501, 22);
MD5STEP(F1, a, b, c, d, in[8] + 0x698098d8, 7);
MD5STEP(F1, d, a, b, c, in[9] + 0x8b44f7af, 12);
MD5STEP(F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
MD5STEP(F1, b, c, d, a, in[11] + 0x895cd7be, 22);
MD5STEP(F1, a, b, c, d, in[12] + 0x6b901122, 7);
MD5STEP(F1, d, a, b, c, in[13] + 0xfd987193, 12);
MD5STEP(F1, c, d, a, b, in[14] + 0xa679438e, 17);
MD5STEP(F1, b, c, d, a, in[15] + 0x49b40821, 22);
```

这里 a、b、c 和 d 是来自 hash 数组的 32 位字的状态，MD5STEP 是一个如下定义的宏：

```
#define MD5STEP(f, w, x, y, z, in, s) \
(w += f(x, y, z) + in, w = (w<<s | w>>(32-s)) + x)
```

将应用于 4 个循环中的从 F1 到 F4 的非线性函数定义如下：

```
#define F1(x, y, z) (z ^ (x & (y ^ z)))
#define F2(x, y, z) F1(z, x, y)
#define F3(x, y, z) (x ^ y ^ z)
#define F4(x, y, z) (y ^ (x | z))
```

如果消息的长度不是 64 字节的倍数，那么沉填充放后一部分直到总长度是 64 字节的整数倍。简而言

之，填充是一位 1 后跟多个 0，最后原始消息结束或以位表示。md5_Final()函数进行填充并调用 md5_transform()函数对最后一个 block 数组计算最终的输出。md5 值最终存储在 md5_t 结构中的 hash 字段中。

练习

1. 先 CPU 中的数据可以表示成低字节序（little-endian）或高字节序（big-endian）。解释 md5.c 程序怎样处理在计算机表示上的差异。

2. 查看源码目录中与 sha1_generic.c 进行比较，找到 sha_transform()函数在哪里以及如何实现。md5_transform()和 sha_transform()实现的最大不同是什么？

8.2.3 链路层隧道

在链路层中一种流行的安全机制是隧道，它是基于分组到发的。隧道通过公共网络构建一条专用通信信道。如果用户希望访问企业网络，他只需拨打该公司的本地网络接入服务器（NAS）并使用 NAS 建立一条到公司网络的隧道。第 2 层隧道可以同时支持 IP、IPX、NETBEUI 和 AppleTalk。

第 2 层隧道的一个众所周知的例子是（点到点隧道协议 PPTP），定义在 RFC 2637 中，它用于 VPN。在隧道中 PPTP 将加密的 PPP 帧封装在 IP 数据报中。PPTP 隧道具有两种模式：1）客户端发起模式，客户端发起直接连接到 PPP 服务器的链接；2）ISP 发起模式，客户端首先建立与 ISP 接入服务器的 PPP 会话，然后与运行 PPTP 服务器建立一条隧道。通过呼叫标识符区分多个连接可以共享建立的隧道。

第 2 层隧道协议（L2TP）将思科公司的第 2 层转发（L2F）和微软的 PPTP 的优点结合起来。L2TP 具有多个优于它前身的优点。例如，PPTP 只支持一个隧道和一个用户，而 L2TP 可以支持多个并发的隧道。注意，即使 L2TP 通常在其隧道内承载 PPP 会话并且在其名字中有“第 2 层”，但 L2TP 分组是在 UDP 数据报中传输的。

L2TP 隧道的每一端在客户端充当 L2TP 访问集中器（LAC），在服务器充当 L2TP 网络服务器（LNS）。在 L2TP 有两种消息类型：控制和数据。建立和管理隧道的控制消息是通过underlying 可靠的传输服务发送的。数据消息通过不可靠的传输模式（如 UDP）来承载实际的数据。像在 PPTP 中一样，建立的隧道也可以通过呼叫 ID 的方法被多连接共享。L2TP 还可以与 8.2.4 节中介绍的 IPSec 一起实现。经过 L2TP 隧道的协商和建立后，L2TP 分组然后由 IPSec 封装以保证其机密性。

8.2.4 IP 安全

IETF 在网络层建立了一个开放的网络安全协议标准，即互联网协议安全（IPSec）。我们首先介绍 IPSec 的概念，然后描述其机制，通过定义 IP 认证头、IP 封装安全有效载荷和密钥管理实现数据完整性、认证和安全通信中的隐私。

许多企业服务是在互联网之上构建运行的，因此在互联网上的专用通信就是主要问题。在会话层和应用层已经提出了多个网络安全的标准。例如，SET 和 SSL 可以实现安全的 HTTP。由于互联网以互联网协议（IP）为基础，所以在 IP 层就需要有一种安全机制以综合上层的各种安全机制。因此，IETF 为 IPv4/v6 建立 IP 安全（IPSec）以实现以下目标：身份验证、完整性、机密性和访问控制。

IPSec 的第 1 个版本（RFC 1825 到 RFC 1829）是 1995 年提出的。它有两个主要模式：IP 认证头（AH）和 IP 封装安全有效载荷（ESP）。前者提供数据完整性和认证，而后者提供安全的数据传输。AH 和 ESP 头都包括两个可选的头部以便在 IPv6 中使用 IPSec。第 1 个版本的 IPSec 中没有对密钥交换和管理的描述。它只定义了分组格式。1998 年，标准的第 2 版（RFC 2401、RFC 2402、RFC 2406），它包括安全关联（SA）和密钥管理，互联网密钥管理（IKE）。因此，IPSec 得以完整实现。

安全关联

为了 IPSec 中的专用通信，设计了安全关联（SA）以便建立一个安全传输的单向连接，它也是 IPSec 中最重要的概念。它定义的多个重要参数：认证算法及其密钥、加密/解密算法和密钥、密钥的有效期等。唯一的 SA 可以由主机的 IP 地址、安全识别码（代表 AH 或 ESP）和 32 位的安全参数索引（SPI）定义。由于 SA 是单向的，所以它需要两个 SA 来构建一个双向的点到点连接。一个 SA 既可以使用 AH 也可以使用 ESP 作为安全协议。

认证头部

RFC 1828 建议 IPSec 使用 MD5 算法进行认证。发送者从传输的 IP 分组和使用 MD5 算法的密钥来计算单息，然后将消息添加到分组中。接收到该分组后，接收者执行相同的 MD5 计算并得到消息值。接收者将该值与分组中的值进行比较。如果两者相等，身份认证成功。由于将 MD5 计算应用到整个 IP 分组，所以这种方法不仅执行认证，而且还验证数据完整性。

为了认证，IPSec 定义了两种模式：端到端模式和端到中间设备模式。图 8-11 中显示了它们之间的主要区别。在第一种模式中，通信的双方都执行认证。当通信双方不信任网络设施的安全性但希望保证传输的安全性时，就应用这种模式。对于后者，认证在其中一方和另一方所在的局域网中的路由器或者防火墙上进行。因此，路由器或者防火墙将扮演一个“安全网关”的角色。换句话说，网关应该保证局域网的安全性。

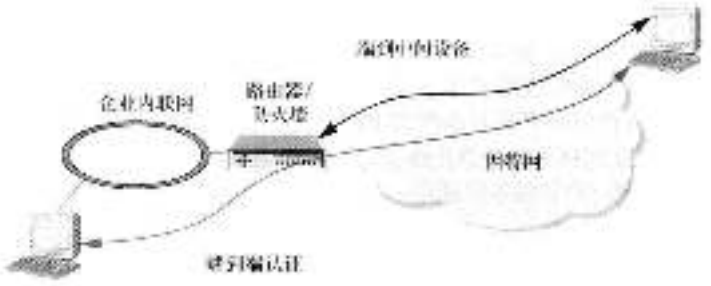


图 8-11 认证类型

图 8-12 显示了认证头部的格式。第一个字段，下一个头部，表示认证头部后的下一个有效载荷的协议类型。后面是一个长度为 8 位的字段。长度为 16 位的预留字段供应用于将来，目前设置为 0。SPI 字段表示一个独特的 SA。序列号表示分组的序列号，用于防止重放攻击。



图 8-12 认证头部

封装安全有效载荷

封装安全有效载荷（ESP）采用 DES 或者 3DES 提供安全的 IP 分组传输。ESP 不仅保证数据安全，也同样实现认证和数据完整性。ESP 可以在两种模式下运行：传输模式——在传输层加密数据块；隧道模式——加密整个 IP 分组。

图 8-13 和图 8-14 中分别显示了这两种模式。在传输模式中，ESP 头部位于数据块之前。这种模式与隧道模式相比，有一个较短的加密部分，所以它要求的带宽也较少。在隧道模式中，ESP 头部位于加密 IP 分组之前。这种模式产生了一个新的 IP 头部。它适用于被安全网关保护的环境。在传输过程中，发送者或者网关加密一个 IP 分组，之后这个加密的分组将发给接收者的网关。接收者的网关将解密这个 IP 分组并将原始的简单文本数据发送给接收者。



图 8-13 传输模式 ESP

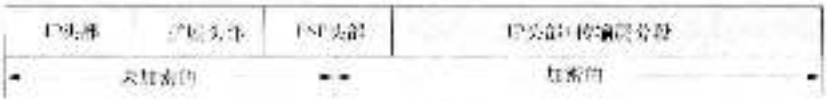


图 8-14 隧道模式 ESP

密钥管理

因为 AH 认证和 ESP 加密时需要加密和解密密钥，所以密钥管理在 IPSec 标准中非常重要。主要的密钥管理协议包含：IP 简单密钥管理协议（SKIP）和 ISAKMP/Oakley（因特网密钥交换，IKE）。SKIP，由 Sun Microsystems 提出，采用 Diffie Hellman 的密钥交换算法传输基于公钥系统的密钥。

ISAKMP 由两个主要步骤组成。第一步，ISAKMP 的实现通过协商建立一个安全和认证的信道，第一个 ISAKMP SA。第二步，它使用第一个 SA 构建 AH 或者 ESP SA。ISAKMP SA 和 IPSec SA 之间的主要区别是 ISAKMP SA 是双向的，而 IPSec SA 是单向的。

开源实现 8.3：IPSec 中的 AH 和 ESP

概述

IPSec 的实现既可以在开源的软件包又可以在 Linux 内核中获得。前者包括 Openswan（http://www.openswan.org）和 strongSwan（http://www.strongswan.org），后者是 Linux 内核 2.6 将 IPSec 集成到 Linux 内核中是一件从头开始的新产品。鉴于 Linux 内核在 Linux 用户中非常流行，这里我们介绍 Linux 内核中的 IPSec 实现。

IPSec 有两种主要的模式：AH 和 ESP。AH 为 IPv4 实现的源代码在 net/ipv4/ah4.c 中（或者 IPv6 的实现在 ipv6/ah6.c 中），而 IPv6 的 ESP 实现的源代码在 net/ipv4/ah4.c 中（或者 IPv6 的实现在 ipv6/ah6.c 中）。

框图

图 8-15 显示了实现的框图。ah4_init()和 esp4_init()两个初始指针注册到相关联的处理程序，两者中的头部也都初始化。当分组被接收或者被发送时，xfrm_input()和 xfrm_output()函数将根据使用的模式：AH 或 ESP，调用相应的函数。

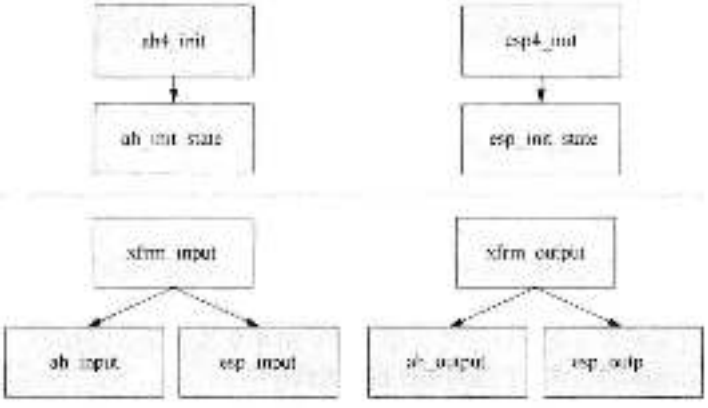


图 8-15 Linux 内核中在 AH 和 ESP 内实现的主函数

数据结构

在两种模式中，认证头和 ESP 头部都定义在 include/linux/ip.h 中，如下所示。

```
struct ip_auth_hdr {
        __u8 nexthdr;
        __u8 hdrlen;
        __be16 reserved;
        __be32 spi;
        __be32 seq_no;
        __u8 auth_data[0];
};
```

```
struct ip_esp_hdr {
        __be32 spi;
        __be32 seq_no;
        __u8 enc_data[0];
};
```

算法实现

在 AH 实现的开始，ah4_init()函数调用 inet_add_protocol(&ah4_protocol,IPPROTO_AH)注册 AH 处理程序的结构，即 ah4_protocol。这个结构的处理程序事实是被接收和处理 IPSec 分组的 xfrm4_rcv()函数。这意味着该协议的解析留到以后讨论。xfrm4_rcv()函数将最终调用函数来解码分组并且获得输入状态（参见 net/xfrm/xfrm_input.c 中的函数 xfrm_input()），依次调用 ah4_input()函数（挂接到 ah_type 结构，参见 ah4.c）。解析 AH 头部后，这个函数将执行以下代码来确认 ah->auth_data（即在 AH 头部中的认证数据）是否等于 ICV（完整性验证）。

```
u8 auth_data[MAX_AH_AUTH_LEN];
memcpy(auth_data, ah->auth_data, ahp->icv_trunc_len);
skb_push(skb, ihl);
err = ah->auth_digest(ahp, skb, ah->auth_data);
if (err)
        goto unlock;
if (memcmp(ahp->work_icv, auth_data, ahp->icv_trunc_len))
        err = -EBADMSG;
```

在代码中，原先的 ah->auth_data 首先复制到 auth_data 中，然后等 ah_output()函数一样，ah_mac_digest()函数计算 ICV（在 ahp 结构中）（见下面）。两个值将进行比较以便检查分组是否有效。如果有效，接收分组；否则，丢弃分组。

在输出一边，为了验证函数 ah_output()调用 ah_mac_digest()产生 ICV，使用以下代码将这些值复制到 AH 头部的 ah->auth_data 中：

```
err = ah_mac_digest(ahp, skb, ah->auth_data);
memcpy(ah->auth_data, ahp->work_icv, ahp->icv_trunc_len);
```

为了简单起见，这里我们仅介绍 AH 的实现。ESP 的实现留给读者进一步研究。

练习

1. 在 xfrm_input.c 中找出函数 xfrm_input 是如何确定与该分组以及调用 ah_input()函数或 esp_input()函数的。

2. 简要描述特殊的数列算法并源实现，如 MD5。其中要包含 md5_init、md5_update 和 md5_final 是如何在 ah_mac_digest 函数中实现的。

8.2.5 传输层安全

在网络安全中，在客户机和服务器主机之间提供安全和可靠交易的一个重要方法就是在传输层将密码学和认证机制相结合。一种好的传输层安全解决方案就是安全套接字层（Secure Socket Layer，SSL）。然而，电子商务需要为交易以及与信用卡系统的集成提供更多的保护，因此安全电子交易（Security Electronic Transaction，SET）就作为这目的而设计的。

安全套接字层

NetScape 建议利用 SSL 来支持 Web 客户机和服务器之间数据交换的加密和认证。之后在 RFC 2246 中将 SSL 标准化为传输层安全协议（TLS）。SSL/TLS 工作在传输层和应用层之间。在执行 SSL 前，客户机和服务器需要协商数据加密算法，如 DES 或 IDEA。协商之后，在数据传输中进行加密和解密。图 8-16 演示了 SSL 交易流程。

- 客户机发送“SSL Client Hello”来启动与服务器之间的加密机制。
- 服务器向客户机回复一个“SSL Server Hello”消息，然后将它的证书发送给客户机以便请求客户机的证书。
- 客户机将它的证书发送给服务器（客户机的证书应该为大部分没有让书客户机的可选项）。

- 在此之后，服务器和客户协商密钥交换。其中会话密钥使用服务器的公钥加密。最后，客户机和服务器两者共享会话密钥并且利用它执行安全的数据交换。

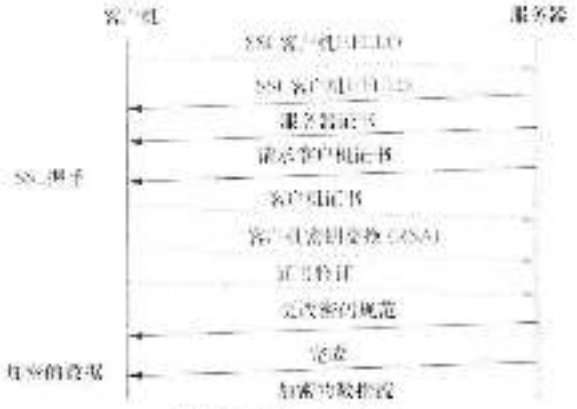


图8-16 SSL交易流程

SSL支持客户机和服务器之间的数据加密，但缺乏它自己合适的一种完整的安全支持机制，如信用卡的安全支持。假设Alice在Bob商店的某个货架商品。然后她使用信用卡付款。信用卡内的信息应该安全地传递给Bob。由于Bob具有密钥来加密和解密Alice的信用卡信息，所以Bob有可能滥用Alice的信用卡信息。SSL也缺乏一种证书机制来确认客户机的信用卡交易。一旦黑客拥有了某人的信用卡号码，他也可以滥用它。

历史演变：HTTP安全（HTTPS）和安全外壳（SSH）

HTTP Secure（HTTPS）使用SSL/TLS协议为HTTP提供数据加密和安全认证。因此，不相关者就不可能看到有效载荷的内容，即使分组可能被窃听也是如此。Web服务器一定要提供一个由认证中心（CA）签名的公钥证书。这个证书包含一个公钥和使用者的身份证明。CA就是一个颁发这种证书的机构，并且保证这个证书是可信的。当用户浏览Web网站提供的HTTPS接入时，浏览器将核实证书的真实性，并且证书被嵌入任何发出的警告提醒用户。因此用户不容易被欺骗有的Web到特殊点。然而，仍然由用户来定义HTTPS的安全性。如果用户完全忽视了警告并且继续浏览Web站点，那么安全性将无法得到保证。与HTTP相反，HTTPS默认运行在端口443上，通过以"https"开头的URL访问。

与HTTPS一样，SSH（Secure Shell）运行在端口22上，能够提供同样的安全性能。与Telnet相反，在SSH中传输的分组是加密的，因此它可以作为不可靠Telnet的一种很好的替代。同样，与HTTPS类似，SSH支持密钥交换和服务器认证。除了密码机制外，它还为用户认证提供一种公钥机制。例如，用户可以生成一个公钥并且将它存储在服务器上。用户保管自己的私钥，在用户登录到服务器时密钥对就可以得到验证。由于它的安全性，SSH在远程shell和安全文件传输中非常受欢迎。

安全电子交易

Visa、MasterCard、IBM、Microsoft和HP合作，于1996年2月提出安全电子交易（Secure Electronic Transactions，SET）协议。安全电子交易组织LLC（又称为SETCo）成立于1997年7月，致力于管理和促进SET协议。SET的特点有：

- SET仅提供支付中相关信息的加密，而SSL则加密客户机和服务器之间的信息。
- SET加密购买者、销售方以及销售方所在银行之间传输的高度敏感数据。所有都需要拥有数字证书。
- SET和SSL之间的主要区别在于，SET不会将信用卡号码给卖家，所以卖家不可能滥用这个号码。

图8-17说明了SET的操作过程，其中包含4个主要角色：买家Bob、电子商业卖家Alice、信用卡的开户银行和电子商店的开户银行。Bob的公钥（E_b）和私钥（D_b）、Alice的公钥（E_a）和私钥（D_a），4个方面的证书也包含在了SET的操作过程中。交易流程如下：

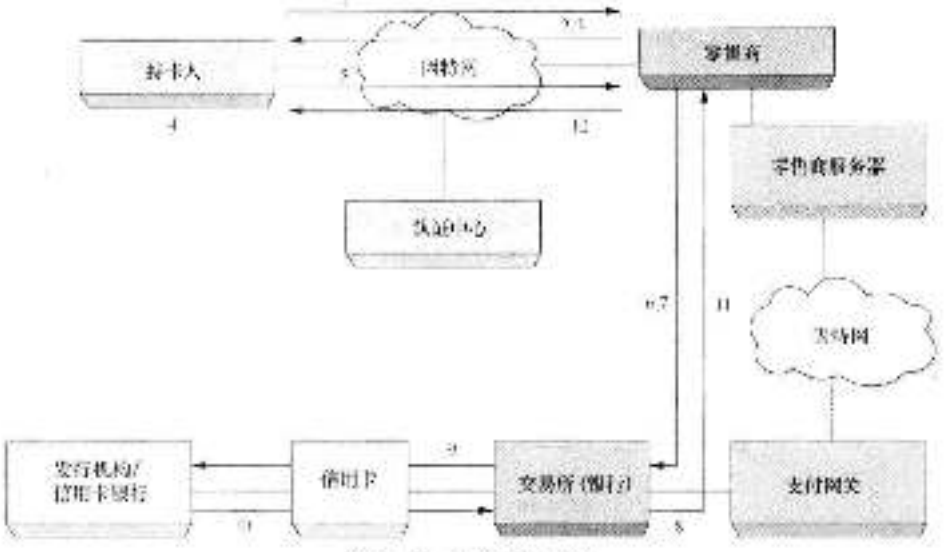


图 8-17 SET 操作流程

1）Bob 从 Alice 的电子商店中选择了一种产品，并且告知 Alice 他将使用信用卡支付。

2）Alice 将本次订单的交易 ID 返还给 Bob。

3）Alice 将她的证书信息、公钥以及她的开户银行的公钥发送给 Bob。

4）Bob 收到了第 3 步的消息。

5）Bob 通过网络定义并且具有订购信息（OI）和购买信息（PI）。他使用 Alice 的公钥加密 OI 并将它发送给 Alice。同时，Bob 使用 Alice 开户银行的公钥加密 PI 并将它发送至开户银行。

6）Alice 将“请求证书”消息和订购 ID 发送给 Bob 的信用卡的开户银行。

7）Alice 使用其银行的公钥来加密如下信息：Bob 加密的 PI、她自己的证书和“请求证书”消息，并将它们都发送到她的开户银行。

8）Alice 的开户银行解密这些加密过的信息并检查它们是否被修改过。

9）Alice 的开户银行使用信用卡原始交换机制来处理相关操作。

10）Bob 的开户银行向 Alice 的开户银行回复了证书结果。

11）如果 Alice 的开户银行收到了“确认成功”消息，就立即回复消息给 Alice。

如果上述一切顺利，Alice 就将这次订单的回复消息发送给 Bob 来确定交易已经完成。每对请求和应答都需要双方保护它不被第三方修改或者得到安全信息。此外，Alice 无法获得 Bob 的信用卡号。因此，SET 就能确认一条通过网络的安全交易环境。

8.2.6 VPN 技术的比较

下面比较前面三节中提出的 VPN 技术。根据 VPN 所基于的协议，VPN 技术能够分为 L2TP/PPTP（在链路层）、IPSec（在网络层）、SSL（在传输层）VPN。总的来说，VPN 操作所在的层次越高，其建立和配置也就越简单。然而，在更低层的操作可以支持更广泛的上层协议。

L2TP VPN 支持 PPP 的认证和隐私保护。L2TP 支持的认证协议包括：PAP、CHAP、MS-CHAP v1 和 v2 等，同时数据也可以使用 3DES 算法进行加密。它的最大优势是能够传输广泛的第 2 层协议：以太网、帧中继、PPP、ATM 等，以及非 IP 协议，如 IPX 或 Appletalk。然而，L2TP 不够安全，可能遭受多种攻击。简单地说，L2TP 在隧道层提供 LAC 和 LNS 之间的认证，但是这种认证不是以每个分组为基础。例如，在隧道建立之前攻击者有机会劫持 L2TP 隧道。而且，L2TP 不提供密钥管理服务。由于互联网协议占据主导地位，所以对于非 IP 协议支持的需求也在不断降低。L2TP 也要求在使用之前安装 L2TP 客户端软件。

在 IPSec 的隧道模式，整个 IP 分组都是加密的，并且分组使用一个新的 IP 头部进行封装。因此，

IPSec 可以用来创建 VPN。我们不重复介绍 IPSec 的细节，仅总结 IPSec VPN 的优点和缺点。像 L2TP VPN 一样，除了支持 TCP 外，IPSec 还支持所有的上层协议。而且，它还对 TCP 相关的攻击（如拒绝服务攻击和伪造 RST 来终止连接）具有免疫性。它的主要缺点是难以部署。首先，系统必须能够支持 IPSec 并且有相关的客户端软件。这个缺点在大规模的部署中具有消极的影响。其次，公司的防火墙必须要允许允许 IPSec 中的流量。因此使用 IPSec VPN 就要额外地对防火墙进行配置。

SSL VPN 因为其用户友好性，最近变得流行起来。用户可以简单地使用普通的浏览器来配置 VPN。SSL VPN 的步骤很简单。在开始的时候，用户通过浏览器和 VPN 网关联系起来。在用户登录之后，网关将向用户认证自己并且传送加密过的会话 cookie 给浏览器。然后两者开始通信并且交换会话密钥。因为如此多的设备可以浏览 Web，所以该特性非常具有优越性，这也是 SSL VPN 流行起来的原因之一。

8.3 访问安全

在私有和公共网络之间控制访问是网络安全的一个关键解决方案。在本节中，我们将介绍用于访问控制的网络设备：防火墙。防火墙可以根据网络层/传输层中的信息（如 IP 地址和端口号）或应用层信息（如 URL）过滤网络流量。在对防火墙系统有了一个简单的了解之后，我们将介绍每种类型的防火墙。Netfilter/iptables 和 PNIX 是两种开源防火墙实现的例子。

8.3.1 简介

防火墙是一种通过提供私有网络和公共网络之间的访问控制来保护企业网络的常用设备。如果允许分组访问列表，则分组就能够通过防火墙；否则，它们将被阻塞。阻塞操作也会被审计。因为私有网络中的主机隐藏在防火墙的后面，所以对它们的访问必须明确地得到防火墙的允许，未授权的外部用户不了解服务器或者主机在防火墙后面的私有网络中。

防火墙过滤是双向的。一个组织也可以根据某些访问策略来限制私有网络上的用户访问互联网。例如，雇员在上班工作时间不允许访问外部 FTP 站点。

因为无论哪个方向的分组都必须穿过防火墙，所以防火墙对大量分组进行分类的效率很重要，否则防火墙就有可能成为瓶颈。即使被检查的分组信息通常是第 3 层和第 4 层的信息，例如 IP 地址和端口号，但是只检查第 3 层和第 4 层信息是不够的，因为 1）给定的服务可能不在标准的众所周知端口上运行，使它很容易躲避控制策略；2）需要基于应用层信息实现精度的控制。例如：Web 过滤器可以检查 URL 请求是否指向一个被允许的站点。

因此，按照数据分组中被检查的字段，我们需要两种类型的防火墙：网络层/传输层防火墙和应用层防火墙。有关这两种类型防火墙的详细描述将在接下来的两节中给出。

8.3.2 网络层/传输层防火墙

网络层/传输层防火墙也称为分组过滤器，即它根据网络层的字段和管理员配置的规则来过滤分组。这些字段可以是协议标识符、源/目的 IP 地址、TCP 或者 UDP 的源/目的端口号等。这样的防火墙可以进一步分成屏蔽主机防火墙和屏蔽子网防火墙。图 8-18 显示了屏蔽主机防火墙的结构。在屏蔽主机防火墙中，无论是进入还是外出的流量都必须通过堡垒主机。换句话说，IP 过滤器由路由器可以按照指定允许访问和不允许访问 IP 地址的规则来允许或者阻止分组通过堡垒主机。

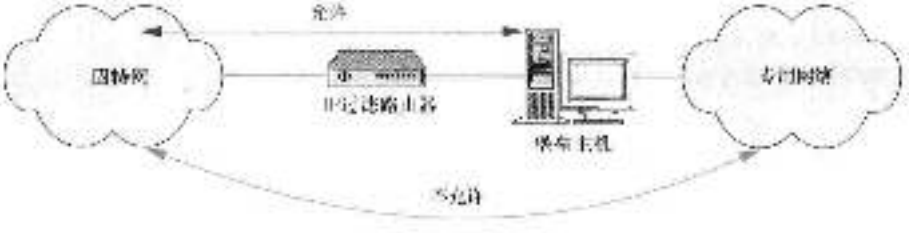


图 8-18 屏蔽主机防火墙

在这种结构中，堡垒主机在专用网络之外而且必须阻止所有的攻击。这种结构的优点是在 IP 过滤路由器上设置分组过滤很简单，因为同个方向上的分组都必须通过堡垒主机。它的缺点是，如果管理员在专用网络中允许特定的服务，那么整个专用网络就会暴露给公共网络。一旦分组能够经过这些服务访问专用网络，那么其安全性能会显著地降低。

图 8-19 显示了屏蔽子网防火墙的结构，它利用网络上的两台 IP 过滤路由器和非军事化区域（DMZ）。实际上，这种结构已通过一台带有多接口的路由器（分别连接到专用网络、公共网络和 DMZ）来实现。因为 IP 过滤路由器在专用网络周边构建，所以即使 IP 过滤路由器在因特网附近开启了某些不需要穿过堡垒主机的服务，专用网络中的主机也不会将它们自己暴露给公共网络。这样就可以避免屏蔽主机防火墙的缺点。设置 IP 过滤路由器与设置屏蔽主机防火墙类似。邻近公共网络的 IP 过滤路由器设置访问规则来确认到专用网络的目的地 IP 地址，同时从专用网络到公共网络的源 IP 地址是堡垒主机的地址。邻近专用网络的 IP 过滤路由器设置访问规则来确认来自专用网络的分组的目的 IP 地址，并且到专用网络的分组的源 IP 地址必须是堡垒主机。

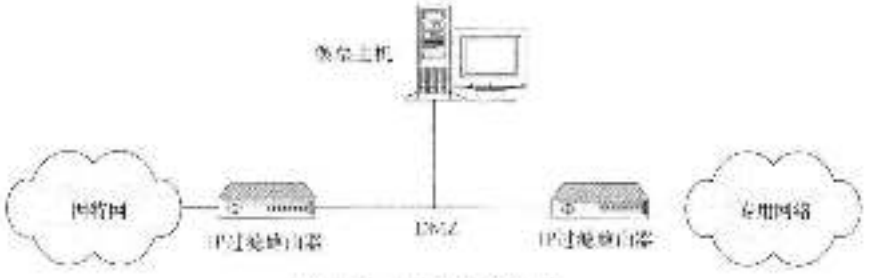


图 8-19 屏蔽子网防火墙

如果一个组织向互联网提供一些服务，例如电子邮件、Web 和 DNS，那么这些服务器一定不能在专用网络中。如果这些服务器遭到攻击，攻击者就能够运过已被攻破的服务器到达专用网络中所有的主机，就好象防火墙不存在一样。DMZ，一个介于外部防火墙和内部防火墙之间的区域，是放置服务器的合适位置。即使攻击者已经攻破了 DMZ 之内的服务器，他仍然不能够访问内部防火墙之后专用网络中的主机。需要注意的是，在 DMZ 中的服务器不允许主动地访问专用网络；否则攻击者在 DMZ 的服务器被攻破的情况下，就有机会到达专用网络。

开源实现 8.4：Netfilter 和 iptables

概述

Netfilter 是系统内核中的一组检查点，它们能够监视通过某个通信协议的分组。这些检查点称为钩子（hook）。通过这些钩子，在内核中分组路径上操作分组就成为可能。每一个钩子都有一个唯一的钩子号，因此 Netfilter 就可以检查当前的通信协议是否有一个已经注册的钩子用于经过 Netfilter 处理的分组。如果注册的钩子存在，就必须检查这些分组并且必须遵守已经定义的规则。通过以下 5 步来进行处理：

- NF_ACCEPT：接收分组。
- NF_DROP：不允许任何处理就丢弃分组。
- NF_STOLEN：Netfilter 处理分组，不再是内核的通信协议。
- NF_QUEUE：将分组保存到队列中。
- NF_REPEAT：调用该钩子再次处理分组。

框图

在 Netfilter 中，根据 5 个已注册的钩子，IP 头执行分组经过。

1）NF_INET_PRE_ROUTING

2）NF_INET_LOCAL_IN

3）NF_INET_FORWARD

4) NF_INET_POST_ROUTING

5) NF_INET_LOCAL_OUT

图8-20显示了5个钩子之间的关系。

NF_INET_PRE_ROUTING代表主机接收分组之前和开始处理路由函数之前的钩子。NF_INET_LOCAL_IN是用于寻找在处理完路由函数后那一个目的地址是主机的钩子。NF_INET_FORWARD是用于分组在经过路由函数处理后传输到另一个主机的钩子。NF_INET_POST_ROUTING是结束路由函数之后的钩子。NF_INET_LOCAL_OUT代表路由函数处理之前的钩子。

当钩子检查分组时，就必须应用来自iptables的分组过滤规则。它是用户空间程序用以指定接收、关系或者将分组插入队列。

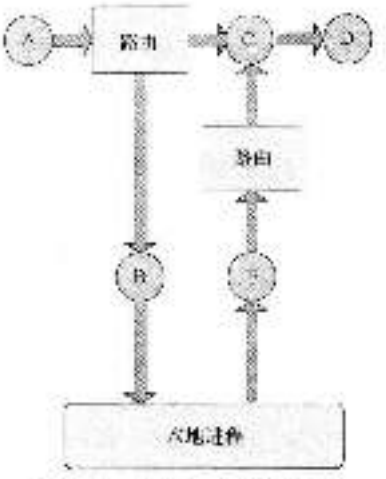


图8-20 Netfilter注册的钩子

数据结构

下面的3个数据结构代表了一个iptables规则。

1) struct ipt_entry包含以下这些字段。

- struct ipt_ip：需要匹配的IP头部。
- nf_cache：保序列表示IP分组头部中的哪一个字段必须要检查。
- target_offset：表示结构ipt_entry_target的初始化位置。
- next_offset：高ipt_entry_target结构所在的规则开头的偏移量。
- comefrom：内核用于跟踪分组传输的字段。
- struct ipt_counters：记录分组数量和本身匹配规则的分组数量。

2) struct ipt_entry_match：记录匹比较的分组内容。

3) struct ipt_entry_target：表示被定成后记录行为（即目标）。

算法实现

iptables的源代码位于net/ipv4/netfilter目录内（此时于内核源代码树）。在ip_tables.c中的ipt_do_table()函数通过使用表iptables程序中指定的规则处理分组的检查。钩子的原型如下：

```
int_do_table(struct sk_buff *skb, unsigned int hook,
local struct net_device *in, local struct net_device
*out, struct xt_table *table);
```

这里skb指向分组，hook是标识钩子的钩子号，in和out是网络设备的输入和输出，而table是规则表。在开始时，ipt_do_table()函数需要从下列语句中hook的索引中定位规则列表。

```
private = table->private;
table_base = (void *)private->entries[smp_processor_id()];
e = get_entry(table_base, private->hook_entry[hook]);
```

这里e是一个指向规则的ipt_entry结构体的指针。找到了从hook索引的规则列表之后，代码开始匹配IP分组头部信息，如源IP地址和目的IP地址，每一个进入的分组都必须匹配这些规则。如果与分组头部信息匹配，那么代码就继续与ipt_entry_match结构中所指定的内容进行匹配。下面的代码对应于匹配处理。

```
if (ip_packet_match(ip, indev, outdev, ...))
{
    struct ipt_entry_target *t;
    if (IPT_MATCH_ITERATE(e, no_match, skb, &mtpar) != 0)
        goto no_match;
```

这里ip_packet_match()函数匹配头部信息，IPT_MATCH_ITERATE宏匹配分组内容。当分组与规则匹配后，就调用下列语句获得目标（即动作）。

```
t = ipt_get_target(e);
```

否则，执行将会继续匹配下一条规则（来自no_match标签）。

练习

1. 指出最终调用哪一个函数在 IPT_MATCH_ITERATE 表中对分组内容进行匹配。
2. 从钩子中找出在何处调用 ipt_do_table()函数。

8.3.3 应用层防火墙

应用层防火墙通过在应用层对比一组应用签名（如 URL）来过滤分组。这种类型的防火墙比仅检查第 3 层和第 4 层信息的防火墙可以提供更细致的访问控制。例如，HTTP 请求的目的端口号一般都是 80。如果网络管理员想要阻塞对黑名单上 Web 站点的访问，即使所有的 HTTP 请求以 80 端口为目标，那么利用防火墙只检查端口号也是不够的。因此，有必要检查分组的内容。

由于应用内容一直都不可用直到 TCP 连接建立为止，所以常见的应用层防火墙是代理服务器。例如，访问 Web 资源的主机首先要和代理服务器建立连接，然后发送 URL 请求，它将被防火墙检查。如果这个 URL 是被允许的，那么这个代理将通过另一条连接转发该 URL 请求到 Web 服务器；否则，这个请求就绝对不会被转发。不仅是 HTTP，还包括其他协议，如 FTP、SMTP、POP3 等可以使用类似的方式进行过滤。开源实现 8.5 提供了一个防火墙工具箱 Toolkit（FWTK）的例子，这是一个支持过滤应用层协议的应用层防火墙。

开源实现 8.5：防火墙工具箱（FWTK）

概述

防火墙工具箱（FWTK）是一组构建应用层防火墙的代理程序集合。在集合中是针对主要应用层协议，如 SMTP、FTP 和 HTTP 等代理服务器设计的防火墙。原先的 FWTK 软件包开发在 10 年前就已经停止了，但是最近又在 SourceForge 上以名为 opentwtk 的软件包得以重新复活。当该软件包中的一个程序执行时，将依赖文件 netperm-table 用于分组过滤的设置和规则。这个文件在 FWTK 的应用中很常见，并且包含两种规则类型：通用代理规则和对应应用规则。在 cfg.c 中的 cfg_append()函数调用 read_config_line()函数来读取和解析 netperm-table 中的每一行。

在 netperm-table 中的每一个表项都有 3 个字段：应用名称、参数名称和参数内容。我们使用 squid-gw HTTP 应用代理配置作为示例。例如，下面两条规则阻止了所有访问，以给定主机作为主机名并且以简单的 HTTP URL 允许对 .edu 域的访问。

```
squid-gw: deny-destinations http:*//*.*.*/*
squid-gw: destinations http://www.*.edu/
```

框图

我们使用 squid-gw 代理作为示例来介绍应用层防火墙的执行流程。图 8-21 显示了在 squid-gw 代理的 main()函数中调用的主要模块。在执行流程中，配置首先由 config_global()读取。当 HTTP 请求到达时，处理它并将它与 http_process_request()中的规则进行匹配。如果访问需要允许，http_send_request()就发送请求，http_response()处理应答。

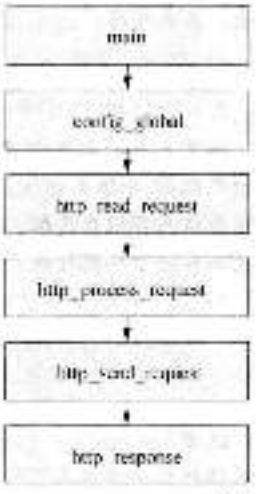


图 8-21 squid-gw 执行流程的框图

数据结构

在该分组中最重要的数据结构是 cfg 结构，它存储了配置和规则。它的定义如下给出。

```
typedef struct cfgrec {
        int     flags;          /* see below */
        int     lnr;            /* linear in config file */
        char    *op;            /* facility name */
        int     argc;           /* number of arguments */
        char    **argv;         /* vector */
        struct cfgrec   *next;
} cfg;
```

从 next 字段，可以发现结构是列表中的一个结点。在读取了 netperm-table 的每一行后，cfg_append()函数将配置中的每一行存储在一

个网址在列表中的节点中。flag 字段为 PERM_ALLOW、PERM_DENY 或者 0（如果没有匹配与许可、拒绝都无关）。cp 字段是对象，如前面所述的 connect 和 bref。相关的变量和编号将会保存在 argv 和 argc 中。

算法实现

执行规则的主函数在 squid-gw.c 中。当收到一个来自客户端的转发请求时，squid-gw 中是从将请求与每个节点的规则进行比较，从 http_process_request() 中调用函数 static int match_destination (CfgEnt *cf, const char *s, const struct url *u, const char *method) 中的规则匹配代码段。

```
if (ct->argc > 0 && strcmp (ct->argv[0], "GET") == 0)
                 || strcmp (ct->argv[0], "HEAD") == 0
                 || strcmp (ct->argv[0], "POST") == 0)
{
  if (strcmp (ct->argv[1], method) != 0)
    return 0; /* Exit the complete rule */
  ++i; /* Skip the current method */
}
```

在这一段代码中，当 nonparm-table 解析后将配置存储在 ct 结构中。如果请求方法，即 GET、HEAD 或 POST 是在规则中指定（ct->argv[0]），那么首先检查转发请求的方法。如果这两种方法不匹配，那么完整的规则立即被忽略。比较了请求方法后，执行下面的代码段。

```
  while (i < ct->argc && ct->argv[i][0] != '*')
    {
      if (strcmp (ct->argv[i], "*") == 0)
        cmp = 0;
      else
        {
          if (url_parse (&pattern, (octet*) ct->argv[i],
strlen (ct->argv[i]), CDF_WILDCARD | CDF_DOMAINONLY) == 0)
            url_error ("domain name", ct->argv[i]);
          cmp = url_compare (&url, &pattern, CDF_DOMAINONLY);
url_free (&pattern, (octet*) 0, 0, CDF_DOMAINONLY | CDF_WILDCARD);
        }
      if (cmp == 0)
      // A URL match is found, details skipped here.
    }
```

在规则中指定的 URL 形式与转发请求中的内容进行匹配。如果比较的 URL 是一个通配符，那么匹配确定（cmp = 0）；否则，函数对 URL 形与转发请求中的 URL 进行比较。继续进行比较，直到找到一个匹配的为止。如果没有找到匹配的，将被拒绝（没有在上面的代码段中显示）。URL 解析和 URL 比较的两个函数分别为 url_parse() 和 url_compare()。它们在 url.c 中实现。我们将 URL 解析和比较的详细情况留给读者完成。

练习

1. 解释 url_parse() 和 url_compare() 是如何在软件包中实现的。
2. 你认为匹配匹配方法有效吗？有哪些方式能够提高其效率？

行动原则：无线访问控制

无线访问控制的一种方法是只允许来自 MAC 地址在白名单上的无线接口的访问。然而，这种方法并非是很有效。因为源 MAC 地址可以很容易伪造，因此并不能使用一种更强的访问控制。

访问控制列表可以存储在路由器上或防火墙上。在较高层，Wi-Fi 保护访问（WPA）规范中的预共享密钥模式提供了不需要验证服务器的密码短语保护。在 IEEE 802.1X 中的扩展认证协议（EAP）模式定义了在允许网络访问前识别和验证用户的多个过程。在更高的层，部署虚拟专用网络是一种保护无线网络的不错选择，依赖强实认证、强加密保密的 VPN 技术，是一对在 WPA 之上的额外保护。

与检查网络层和传输层头部信息的防火墙相比，应用层防火墙主为复杂。在网络层或传输层头部字段的位置和长度上是固定的。在前面的防火墙中的分组过滤的复杂性处中等的，它检这些固定字段

与策略规则进行匹配。相比之下，应用层防火墙需要检查与上面的某应用程序相同的视图中的分组的内容，解析应用协议，恢复其语义，并搜索分组内容（在相关的应用字段中）中的一组签名。签名可能出现的位置不固定，签名的长度也可能会有很大差异。签名可以用复杂的形式（如正则表达式）来表示。这使得签名匹配更加复杂。

除了签名匹配的复杂性外，如果应用层防火墙像 FWTK 那样作为代理来实施，那么它可以利用操作系统中的协议栈来执行所有的工作，恢复分组的内容，和分组重组。防火墙对用户不是透明的。用户必须为使用代理配置应用程序。应用层防火墙也可以是透明的，默默地监听分组并重组分组以恢复其内容，但该程序应该处理 TCP 协议栈所处理的复杂性。这个开销不是微不足道的。所有这些问题，对应用层防火墙的工作效率形成很大的挑战。

8.4 系统安全

攻击一个系统的方法包括 3 个任务：收集信息、利用漏洞、然后用恶意代码攻击。收集信息是指用通过监控、扫描和社会工程方法手段获得关键的成员人信息。知道系统的信息后，攻击者试图找到并利用该系统上的漏洞。最后，攻击者通过渗透到系统中的恶意程序发起攻击或直接攻击系统。我们接下来介绍上述每一个任务中的技术。从攻击者的角度介绍了相关技术后，我们提供各种防御技术。我们以 ClamAV、Snort 和 SpamAssassin 作为开源实现的例子。

8.4.1 信息收集

攻击者通常扫描目标系统以便收集（如服务提供程序、打开的端口）信息，甚至在发起后续攻击之前能对这些信息加以利用。收集技术包括扫描和监控。两个典型的扫描分别是远程扫描和本地扫描。监控收集有关网络或计算机系统的信息，如密码。两种典型的监控分别是嗅探和观察。Jung 等人在 2004 年讨论了如何利用统计的方法检测端口扫描。

远程扫描

远程扫描意为扫描远程目标系统以收集信息，如主机名、打开的服务、服务提供程序，以及可能的某种漏洞。这种扫描软件的一个例子是 Nessus（www.nessus.org），它采用客户机/服务器框架，提供了一个易于操作的图形化用户界面。

本地扫描

本地扫描意为扫描入侵的目标系统以收集信息，如重要的系统文件、特殊权限程序和主机内可以利用的漏洞。其代表是 UNIX 的 COPS。TIGER 是另一个工作在 UNIX 的本地扫描程序。

嗅探

嗅探意味着通过局域网拦截分组访问信息。主机通常只接收发向它自己的分组，但是通过将它的网络接口配置成“混杂模式”，它可以窃听所有经过它的分组。用于这种类型监控的最著名的程序就是 Sniffer。

一种称为分布式网络嗅探器的嗅探程序能够隐藏在服务器和客户机上。攻击者可以入侵一台主机，并安装“客户机”程序监控所有的分组、分析用户标识符和密码，并将这些数据发送到“服务器”。

观察

这种类型的系统监控意味着监控内存、磁盘或其他存储数据，以获得主机内部的信息。例如，监控和记录键盘上键入的信息。根据所收集的信息，攻击者就可以以后入侵到其他主机。

观察者通常使用后门程序包。我们将后门程序描述成恶意代码也可以起到系统监控功能。

社会工程

社会工程试图利用人类的弱点而非通过系统或互联网的攻击。例如，攻击者发送一封电子邮件或打电话给用户，声称他是系统操作员并且询问用户的私人信息。社会工程也包括黑客站在用户背后窥探密码信息。

8.4.2 漏洞利用

漏洞是程序或软件中的设计错误，入侵者可以利用它获得重要的系统信息或管理员的特权或者使

系统瘫痪。在世界上有无数的程序，其中许多程序可能有错误。即使程序设计中没有错误，但是在使用时，用户仍然可能错误操作从而产生安全漏洞。

缓冲区溢出是最知名的设计错误。输入数据可能会溢出缓冲区空间，因为没有仔细检查缓冲区的容量。如果用户在100字节的数组中存储100字节，额外的一个字节就可能会覆盖其他变量并导致意外的执行结果。图8-22显示了一个例子，其中用户使用缓冲区溢出漏洞来执行他的程序。当调用函数hallod()时，操作系统将为该函数建立一个栈。在这个例子中，用户只需要在缓冲区中存储包含攻击代码的数据。如果程序不检查输入的大小，数据就可能任意长，因此可能会用一段攻击代码的起始地址覆盖栈中的返回地址。当函数结束后，控制将返回给调用者通过指向恢复的假返回地址，同时程序流将流到攻击代码。



图8-22 缓冲区溢出的例子

远程漏洞

黑客可能入侵远程系统以获得未经授权的数据、用户ID和密码，或者通过远程攻击获得系统管理员权限。由于目标是远程系统，所以这些漏洞通常发生在在线服务，例如，由sendmail提供的邮件服务，据报道已经被攻击过多次。其中大多数漏洞都属于缓冲区溢出。

另一个例子是wu-ftpd，其2.6版本具有缓冲区溢出问题。它发生在命令site exec的函数*printf中。黑客可能会使用一个格式化字符串覆盖返回地址，以得到缓冲区溢出。

一个名为Piranha的Web集群软件包在安装后就附带一个默认的用户标识符Piranha和密码q。如果系统操作员安装软件包后而不改变默认的账户，那么黑客就可以应用此用户标识符访问程序，导致远程攻击。表8-2列出了多个可以提供对操作员密码访问的远程攻击。

另一个远程漏洞的例子是基于协议的攻击。通过利用错误、差的设计或TCP/IP中含糊不清的定义，基于协议的攻击试图攻击一台远程主机。例如，IP欺骗就可以攻击基于地址的认证系统，黑客通过假扮成可以被系统接受的目的IP地址来入侵系统。

表8-2 获得管理员权限的远程漏洞列表

漏洞	应用	版本	原因
phf远程命令执行漏洞	Apache Group Apache	1.0.3	输入验证错误
Multiple vendor BIND（NXT溢出）漏洞	ISC BIND	8.2.1	缓冲区溢出
MS IIS FrontPage 98扩展缓冲区溢出漏洞	Microsoft IIS	4.0	缓冲区溢出
华盛顿大学imapd缓冲区溢出漏洞	华盛顿大学imapd	12.264	缓冲区溢出
ProFTPD远程缓冲区溢出	专业FTP proftpd	1.2pre5	缓冲区溢出
Berkeley Sendmail daemon模式漏洞	Eric Allman Sendmail	8.8.2	输入验证错误
RedHat Piranha虚拟服务器软件包默认账号和密码漏洞	RedHat Linux	6.2	配置错误
Wu-ftpd远程格式字符串栈重写（overwrite）漏洞	华盛顿大学wu-ftpd	2.6	输入验证错误

本地漏洞

在对本地漏洞的攻击中，黑客从某个系统普通用户的身份获得未经授权的数据或更高优先级的权限，如管理员的密码。此漏洞通常发生在一个特权程序设计中或者一个实现错误中。

例如，Xterm 就是 X Window 系统中的终端模拟器。在其早期版本中，它很容易遭受缓冲区溢出漏洞的攻击。如果系统用 SUID root 替换 Xterm，即 Xterm 以 root 的身份执行，那么攻击者就可能获得管理员权限。

密码破解

密码破解通过尝试所有可能的密码试图找出系统的密码。这些可能的密码可以直接从字典文件中派生出来，并从字典中字的无数组合和置换中得到。密码破解程序需要一个系统文件，其中存储了系统的账户和密码，如 UNIX 系统中的/etc/shadow。密码破解如下进行。

1）从字典文件中挑选可能的密码。密码可能是该文件中的一个或多个单词的组合或置换。

2）以与系统密码文件相同的加密方式来加密密码，如使用 SHA1。

3）加密过的密码与被破解的密码进行比较。如果两者是相同的，破解成功；否则，回到步骤 1），并尝试其他密码直到破解程序猜测正确。

密码文件通常得到很好的保护，只有管理员可以读取该文件。密码破解在实际中并非小于一件。常见的方法是利用系统的漏洞（例如，使用缓冲区溢出攻击），在管理员权限的上下文中执行程序来获取文件，或者发现管理员以一种不安全的方式存储密码文件的副本。如果不能获得密码文件，仍然可以通过尝试登录来猜测密码。然而，主机很可能会记录攻击者的尝试，并只允许少数几次密码错误。破解密码的效率取决于系统的速度和密码的复杂性。如果系统非常快并且密码很容易猜测（例如，密码是一个普通的英语单词），那么会花费较少的时间。

拒绝服务

拒绝服务（DoS）攻击会阻塞服务器上的服务，让其他人无法访问它们。其诀窍是耗尽有限的资源，造成该服务无法进行下去。例如，TCP SYN 洪泛攻击填满了目标主机的等待队列，而 ICMP echo 应答洪泛攻击则耗尽了主机的带宽。在 TCP SYN 洪泛攻击的情况下，由于 TCP 采用三次握手的方法建立连接，所以攻击者发出连续的带有假地址的 SYN 分组，但从来没有在握手的第三步提供 ACK 分组，从而导致等待队列满。当队列已满时，主机就不能接受更多的连接。在 ICMP echo 应答洪泛攻击中，黑客同时产生大量的 ICMP echo 请求给目标系统。由于目标系统将回应相同数量的答复给请求者，所以数量庞大的 ICMP 分组将完全耗尽网络带宽。Schuba 等人于 1997 年深入地分析了 DoS 是如何工作的。

大规模地发起 DoS 攻击称为分布式 DoS（DDoS）攻击。图 8-23 显示了 DDoS 攻击的一个例子。黑

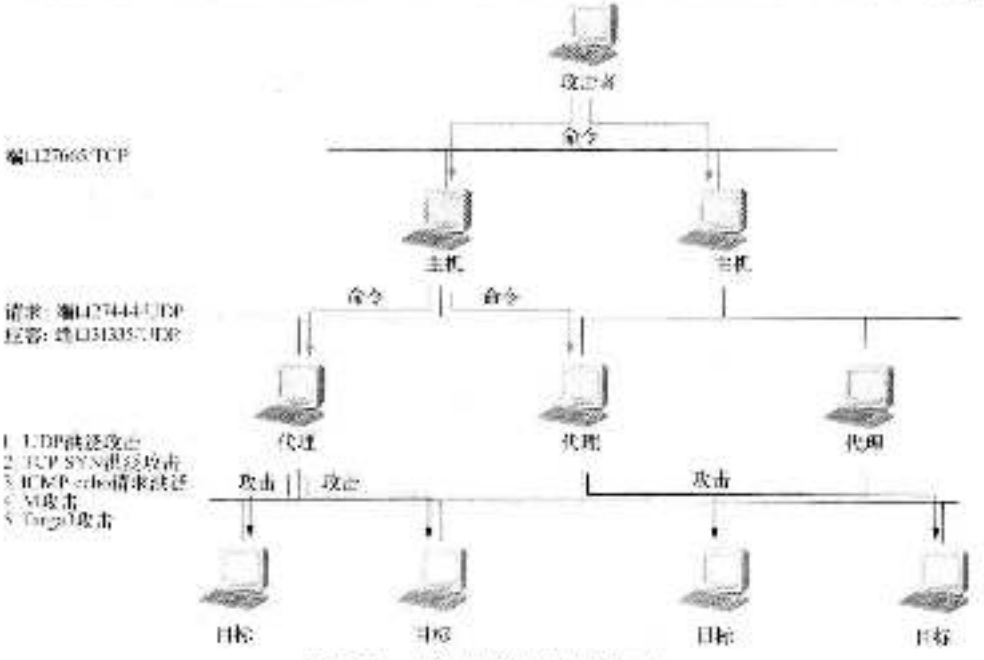


图 8-23 分布式拒绝服务（DDoS）

客控制了大量受害者主机上的代理。一旦黑客发送攻击命令给所有的代理，它们就可以同时产生大量的攻击。

Trinoo 是一个基于 UDP 洪泛攻击的客户机/服务器拒绝服务攻击程序。攻击者发送大量的 UDP 分组（其中可能包含欺骗地址，以避免被服务）给受害者的系统，这将导致流量拒塞甚至停止服务。Trinoo 程序包括多个主机和众多守护进程。攻击者首先连接到主机并发出带有多个重要参数（如目标的 IP 地址以及何时开始攻击）的攻击命令。接到攻击命令后，主机将连接到所有的守护程序上，然后开始攻击所有预定义的受害者系统。攻击过程如下：

1）攻击者使用端口 27665/TCP 连接到主机。

2）主使用端口 27444/UDP 连接到守护程序。

3）守护进程使用 31335/UDP 对主机响应。

4）守护进程使用 UDP 洪泛攻击开始攻击受害者系统。

8.4.3 恶意代码

利用恶意代码又称为恶意软件的攻击，包括黑客通过外部设备或网络攻击目标系统。这里介绍几种类型的恶意代码。

病毒

病毒的行为中包含自我复制和破坏。最初，病毒是指一种能够修改其他程序以便包括其自身副本的程序。早期，一般的感染路径是从磁盘复制一个文件。术语病毒对于普通公众来说，已经成为恶意软件的通用名称。

蠕虫

由于互联网的普及，它已经成为恶意软件传播的主要途径。蠕虫是一种在互联网上能够自我繁殖的程序。一次攻击会将一个蠕虫植入到目标系统，攻击目标，然后再将蠕虫传播到其他系统。它从一台受感染的主机开始相信其他易受攻击的主机，然后也将自身复制给它们。网络可能被网络上大量传播的蠕虫所堵塞。Staniford 等人在 2002 年研究了多种有效传播蠕虫的方法。

两个著名的例子分别是 Code Red（红色代码）和 Nimda（尼姆达）。它们使用分布式拒绝服务（DDoS）攻击 Microsoft IIS 系统。蠕虫传播占据了大量的网络带宽，使得服务器无法接受正常的请求。DDoS 攻击在世界各地迅速蔓延，导致网络中严重的流量拥塞。

特洛伊木马

特洛伊木马（Trojan）将自己伪装成一个无害的程序或文件。这个词汇来自古希腊传说，一支攻击部队通过隐藏在一个巨大的木制马中，然后被对手带进城门，从而渗透进入城中。由于该木马看起来无害，用户会被诱骗执行或者打开它。用户打开木马程序或文件后，特洛伊木马会做一些有害的事情，如安装另一个恶意软件或者使程序崩溃。

后门

成功地入侵后，为了方便下次再次进入受害者系统，黑客就可能植入一个隐藏的后门程序。例如，Back Orifice 2000（BO2K）就是一个在 Windows 环境下，为了完全控制系统通过 TCP 或 UDP 连接的后门程序。它还支持文件传输、监控和记录用户操作。此外，它可以利用额外的插件加强，如当主机连接到互联网时，代码会向攻击者发送一封电子邮件。

Bot

Bot（僵尸）是“robot”的简称，意味着受感染系统可以由“botmaster”（僵尸主）通过命令和控制（C&C）信道来进行控制。受感染的系统和 botmaster 称为“僵尸网络”。botmaster 接管这些系统后，它可以命令被感染系统发起一次分布式拒绝服务，从这些系统中窃取有价值的信息，发送巨大数量的垃圾邮件。Rajab 等人在 2006 年曾经深入研究过僵尸网络的行为。

基于 C&C 信道之间的差异，僵尸网络可以分为 IRC 僵尸网络、P2P 僵尸网络，或混合僵尸网络。IRC 僵尸网络中的 botmaster 使用 IRC 协议控制僵尸。由于 botmaster 以集中方式控制僵尸，所以它容易产生单点故障。因此，通过 P2P 网络（如 Overnet）传输命令的 P2P 僵尸网络越来越受欢迎，因为它们更加健壮。

开源实现 8.6：ClamAV

概述

ClamAV 是一种用于病毒扫描的开源软件包。由于恶意代码及其重新的快速扩展，ClamAV 声称它的 0.95.2 版本能够检测到超过 570 000 条恶意代码（病毒、蠕虫和木马等）。特征以多种形式存在，其中包括用于整个可执行文件的 MD5、用于某个 PE 部分（可执行文件的一部分）的 MD5，用定字符串的基本特征（在整个文件中扫描）、扩展特征（以一种简化形式的正则表达式包括多个部分，加上目标文件类型的规范、签名的偏移量等）、逻辑特征（带有逻辑运算符组合的多个特征）和基于扫描元数据的特征。详情请仔细阅读 ClamAV 的文件中的 sigpoutures.pdf。

框图

下面给出在两个主要执行流程中的框图。第 1 个流程是加载特征数据库，第 2 个是病毒扫描文件。图 8-24 给出了特征加载的框图。cl_load()函数先行了一些初步的检查，然后调用 cli_load()读取特征文件。根据文件的扩展名（即特征文件类型），cli_load()调用不同的函数用于加载和解析特征。以文件 ndb 为例。调用 cli_loadndb()函数，解析 ndb 文件中的每一行并添加到表示特征的数据结构中。

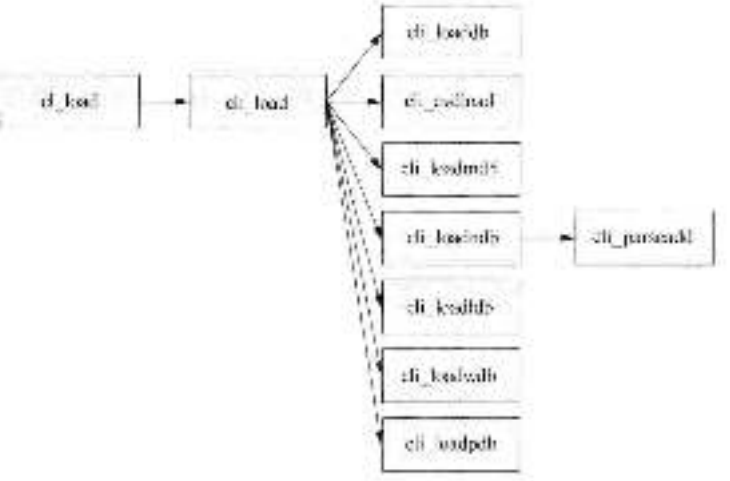


图 8-24 特征装载的框图

图 8-25 给出了用于特征匹配的框图。cli_scanfile()打开文件并调用 cli_magic_scandesc()扫描文件。cli_magic_scandesc()试图标识文件类型，并调用相应的例程来处理文件。例如，如果将文件压缩为 RAR 格式，那么 cli_magic_scandesc()调用 cli_scanrar()来解压缩文件。注意，该文件可能会在一个压缩文件中打包数个文件。因此，cli_scanrar()可能会再次递归调用。

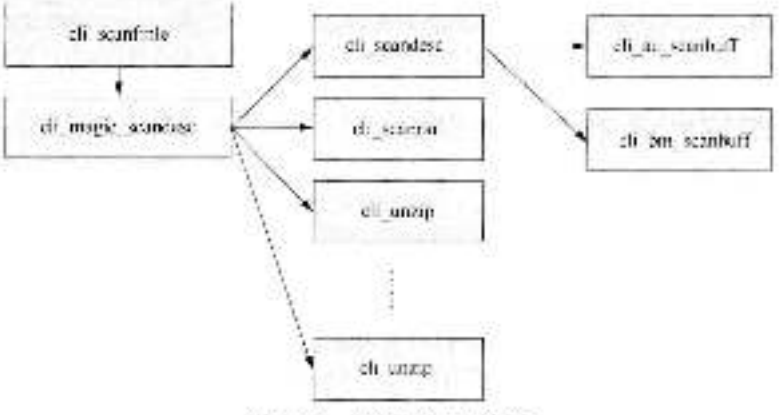


图 8-25 特征匹配的框图

cli_magic_scandesc()来处理这些文件。在最后一层递归（如果有）中，将调用cli_scandesc()逐批地将文件读入到缓冲区中。然后扫描例程，调用cli_ac_scanbuff()和cli_bm_scanbuff()扫描缓冲区中的病毒。首者用Aho-Corasick（AC）算法扫描缓冲区，后者使用Wu-Manber算法扫描同一缓冲区（在ClamAV中，不正确地称为Boyer-Moore［BM］算法）。

数据结构

在ClamAV中允许的目标文件类型在matcher.h中指定。规范的代码如下。

```
static const struct cli_mtarget cli_mtarget[CLI_MTARGETS]
= {
  { 0, "GENERIC", 0, 0 },
  { CL_TYPE_MSEXE, "PE", 1, 0 },
  { CL_TYPE_MSCLER, "CLER", 2, 1 },
  { CL_TYPE_HTML, "HTML", 3, 1 },
  { CL_TYPE_MAIL, "MAIL", 4, 1 },
  { CL_TYPE_GRAPHICS, "GRAPHICS", 5, 1 },
  { CL_TYPE_ELF, "ELF", 6, 1 },
  { CL_TYPE_TEXT_ASCII,"ASCII", 7, 1 },
  { CL_TYPE_PE_DISASM, "DISASM", 8, 1 }
};
```

正如上面已经提到的，ClamAV还要根据目标文件通过使用Aho-Corasick和Wu-Manber算法将特征存储在单独的数据结构中。这两种算法的特征存储在cli_matcher结构（参见matcher.h）中。如果一个特征的目标文件未指定（例如，一个基本特征），那么特征将以“通用”类型添加到数据结构中。为了访问这些数据结构，在cl_engine结构（参见others.h）中的root字段（指向cli_matcher结构的数组指针）是文件类型索引的数组，并指出每个单独的数据结构。因此，ClamAV只扫描文件中的“通用”类型以及与特定文件类型相关联的特征。例如，当扫描一个PE格式的文件（微软可执行文件）时，ClamAV就不加载与其他类型（如HTML）相关的特征。因为需要扫描更少的特征，所以这种方法可以减少误报并加快扫描过程。

算法实现

ClamAV的驱动引擎是libclamav库，其中包含处理归档文件、压缩文件和可执行文件打包器（从打包可执行代码到混淆代码分析和扫描的程序）以及特征匹配引擎的代码。当启动ClamAV时，在readdb.c中的函数cl_load()从数据库中装载特征并根据目标文件类型和匹配算法（Aho-Corasick和Wu-Manber）将特征存储在单独的数据结构中。两种算法的匹配器分别在matcher-ac.c和matcher-bm.c中。前者是负责多个部分的扩展特征，因为在Aho-Corasick算法中的自动化可以更好地表示这些特征。后者负责基本特征和MD5的特征，加上单个部分的扩展签名（即没有特殊的字符，如通配符），因为Wu-Manber算法可以很容易地处理固定（通常长的）的字符串。

scanners.c文件包含驱动病毒扫描的主函数。正如我们在上面提到的，这些函数是从cl_scanfile()开始调用的。经过多次函数调用后，cli_magic_scandesc()函数确定由cli_filetype2()函数调用的文档或压缩文件（或原始文件）的类型。对于每种文件类型，cli_magic_scandesc()调用特定的函数，比如，cli_scanrar()用于解码和扫描文件。不管是什么文件类型，经过解码或解压缩直到原始文件导出后，最终将调用函数cli_scandesc()启动病毒扫描。它调用特征匹配器（在matcher-ac.c和matcher-bm.c中），然后它扫描文件寻找对应目标类型的签名。

练习

1. 找出cli_filetype2()是如何被cli_magic_scandesc调用标识文件类型的。

2. 找出在当前ClamAV版本中的两种扫描算法中，与每个文件类型（或通用类型）关联的特征数量（提示：使用sigtool解压缩ClamAV的病毒库文件［*.cvd］并检查结果的扩展对压格式的文件［*.cdb］）。

8.4.4 典型的防御

描述过攻击方法后，本节将介绍几种防御方法。防御系统包括：数据加密、身份认证、访问控制、审计、监控和病毒扫描。表8.3列出了流行的软件包可分为4种类型，即防御、控制、检测和记录。

防御意味着使攻击者不能得逞，如数据加密。控制采用身份认证和访问控制以防止未经授权的用户访问未经授权的密码/标识符。检测意味着检测任何攻击，如监测和扫描。记录意味着记录消息跟踪攻击者，如审计。这些技术在接下来的各节中描述。

表 8-3 防护软件包

防御类型	软 件	URL
数据加密	PGP	http://www.pgpi.org/
	SSH	http://www.openssh.com/
访问控制	Firewall-1	http://www.checkpoint.com
	Ipchains	http://people.netfilter.org/~nsty/ipchains/
	TCP Wrappers	ftp://ftp.porcupine.org/pub/security/index.html
	Portmap	http://neil.pcown.name/portmap
	Xinetd	http://www.xinetd.org/
监控	Tripwire	http://www.tripwire.com
	RealSecure	http://www.iss.net
扫描	Pc-rollin	http://www.trend.com.tw

审计

审计在日志文件中记录与安全相关的事件，如记录登录失败次数或某些重要活动的次数。日志文件对于跟踪和分析谁或哪个系统被攻击很有用，因此管理员可以保护系统以避免再次出现同样的攻击。现有的操作系统具有审计功能，如 UNIX 的系统文件 wtmp。wtmp 文件记录所有用户的登录和注销状态。在微软的 Windows 系统中，事件查看器执行相同的审计功能。

监控

这种机制监控系统上的任何异常活动，如连续的登录失败。当它检测到攻击时，系统将：1）通过发送电子邮件、寻呼或报警来呼叫系统操作员；2）停止系统或相关服务以便减少可能出现的损失；3）尝试跟踪攻击者。系统可能将攻击特征作为线索以确定攻击类型。

有两种监控类型：基于网络的和基于主机的。前者监控网络主机中任何异常的互联网活动。它截获来自网络接口卡的分组，然后分析任何对主机的不寻常影响并做出适当响应。基于网络的监控，可以检测拒绝服务攻击，如 TCP SYN 洪泛。一旦发现 SYN 分组的来源是非法的，监控将发送一个 RST 分组给受到攻击的主机，并停止无限地继续等待不可能出现的反馈。基于主机的监控器可以监控主机上的任何异常行为，如用户日志和系统操作者（运营商）的活动和文件系统。如果检测到异常活动，监控器会做出适当的反应。一个例子是 Tripwire，它定期检查重要文件并将这些文件与数据库中的文件进行对比以便发现任何非法修改。

入侵检测和防御

入侵检测系统（IDS）根据已知特征或入侵异常流量检测入侵。前一种方法根据已知的入侵特征扫描分组，但可能会出现漏报。后一种方法寻找流量中的异常，通常以统计方法，所以即使未知的入侵也可以检测到。但如果正常互联网活动的行为表现异常，它可能会产生误报。需要在这两种方法之间进行权衡。不管是哪种方法，IDS 都会产生警报，当检测到入侵时日志记录触发警报的分组。

入侵检测系统的局限性在于它们在检测到入侵时只报警；但它们无法防止入侵。IDS 只是被动地监听导线因而无法阻止入侵内部网络。解决这个问题的一个方法是向连接的源或目的地发送 TCP RST 分组，以便让含有恶意流量的连接终止。然而，这种方法是不可靠的。首先，这种方法只对 TCP 连接有效。其次，RST 分组的序列号应该与接收者期望的序列号匹配（即接收者给发送者的确认号）。如果由于繁重的网络流量造成 RST 分组传输延迟很大，就会存在一种竞争状态。发送者可能向接收者发送更多的流量，造成 RST 的序列号不正确。因此 RST 分组可能被直接拒绝。IDS 可以内联地工作，通

对占据分组传输的线路，但可以与防火墙（例如在 Linux 上的 netfilter）一起工作。因此，如果 IDS 发现入侵主动地阻塞流量，这种类型的系统称为入侵防御系统（IPS）。

虽然 IPS 可以阻塞入侵，但它也有一些缺点。首先，如果警报是误报，那么将会阻塞无害的流量。IDS 就没有这样的问题。它只是产生误报的警报，管理员可以忽略警告消息。其次，如果 IPS 是内联的，但速度不足以赶上网络的传输速度，那么 IPS 将成为瓶颈。这对 IDS 来说不成问题，IDS 可以丢弃一些分组，并可能有漏报，但流量传输不会因 IDS 而放慢。

行动原则：IDS 中的瓶颈

场景是，在 2000 年之后，模式匹配是网络入侵检测系统，特别是 Snort 的瓶颈。自此以后，很多研究工作中大多集中在通过软件的加速上，使得 Snort 的字符串匹配增加到了数千兆位每秒。因此，性能问题似乎已经得到了很好的解决。

但是事情并不像看起来那么简单。第一，许多研究者假设特征可以从一个很大的字节流中标记出来。然而，分组必须在成为一个字节流被重新组装，并且攻击者可能会将分组分成小的 IP 片段或 TCP 分段，需要额外的工作来规范分组重组。第二，为了避免由于短特征的产生误报，特征通常与特定的上下文相关联，这意味着只有当上下文条件也同时满足时它们才有意义。下面是一个 Snort 规则的例子。

```
web-cl ert (msg:"ET
INDY3_GET any (msg:"WEB-CLIENT Possible Executable Binary
file transfer"; flow:to_client,established; content:"|50|
00 "; byte_jump:4,58,relative,little; content:"PE|00|
00 "; within:6; distance:-64; flowbits:set,exe.contloc;
(msg:"suricata"; metadata:service http; classtype:misc-
activity; sid:15306; rev:1;)
```

byte_jump、distance 和 within 等可选项代表给定的位置（实际上，其他协议字段），在这些位置的特征（在 content 中）是有效的。通常，协议解析在确定匹配中变得更加重要，因为越来越多的特征取决于协议解析分出的上下文。

第三，攻击者可能会通过各种类型的字符编码混淆不同类型分组的内容，使得在特征匹配之前需要标准化。第四，有些检测技术，如端口扫描检测，可能将从多个连接来的信息关联起来。因此特征匹配会涉及来自多个连接的字节流。

鉴于这些复杂性，我们想要整个入侵检测系统的性能像仅有特征匹配的每个单一核（按照 Amdahl 定律最终上升到很多千兆位每秒）一样高对于流成（5元组）时，系统的实际性能和具有挑战性的情况都需要研究，以使能够达到目标性能。

行动原则：无线入侵

与监控网络流量寻找入侵特征的常规入侵检测系统不同，无线入侵检测系统监控无线入侵的无线电频谱，其中涉及无线授权接入点的存在。检测是很重要的，因为如果一个组织的雇员使用了“流氓”接入点，那么整个内部网络可能会暴露给外部接入。无线防御系统可以自动防止这种威胁的发生。无线入侵检测系统的另一个功能是检测无线攻击，其中包括未经授权的关联、中间人攻击、MAC 地址欺骗、拒绝服务攻击等。

无线入侵/等御系统由三个主要组分组成：1）传感器；2）服务器；3）控制台。传感器可以在其整个覆盖区域的无线信道上监视帧。然后服务器分析这些帧用于入侵检测，而控制台被管理员用来配置系统和报告可能的入侵。管理员可以像在一个常规的入侵检测系统上一样查看报告。

开源实现 8.7：Snort

概述

Snort 是一种流行的开源检测工具，用于监视网络并检测可能的入侵。它可以使用 libpcap 库创建网络接口上的分组，也可以以 PCAP 格式读取分组以进行以后离线分析。在获得到手分析的分组后，Snort 将检查分组是否匹配检测规则，其中规则会考虑在分组头部中指示入侵定义的某一值和分组内容中的某一特征。如果找到匹配，Snort 将产生警报通知管理员可能发生了入侵。下面是一个检测规则的例子。

```
alert tcp any any -> 10.0.1.0/24 80 (content: "/cgi-bin/
pdf"; msg: "SMTP probe";
```

图 8-26 给出了 Snort 中的主要框图。SnortMain()是利用 ParseCmdLine()读取命令行参数的主函数。然后它调用 pcap_dispatch()，这是由 Snort 通过 libpcap 库捕获分组时 libpcap 中的一个函数。pcap_dispatch()函数注册一个回调函数 PcapProcessPacket()。当准备处理每个消费的分组时就调用它。PcapProcessPacket()将调用 Preprocess()，它调用每个预处理器（参见"算法实现"中的内容）按顺序赴接右列表中。每个预处理后，调用 Detect()函数解析协议去出并与分组的唯一端口相关联的规则进行匹配（参见下面的"规则检测"）。当发现入侵时，此生成一个警报给检出插件，它也可能会将警报记录在一个文件中，并将它转储给控制等。

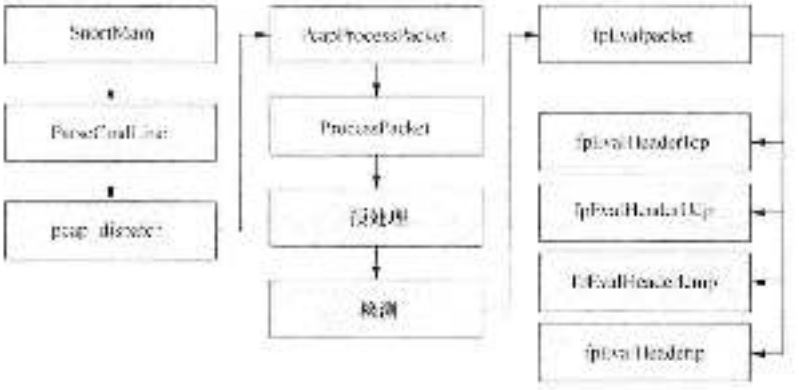


图 8-26 Snort 的框图

数据结构

除了包含全球信息的 PV 结构外，在 Snort 中最重要数据结构是存储 Snort 规则的规则树。规则树共有三个主结构：ListHead、RuleTreeNode、OptTreeNode。ListHead 结构定义如下。

```
typedef struct _listhead
{
    RuleTreeNode *IpList;
    RuleTreeNode *TcpList;
    RuleTreeNode *UdpList;
    RuleTreeNode *IcmpList;
    struct _OutputFuncNode *LogList;
    struct _OutputFuncNode *AlertList;
    struct _RuleListNode *ruleListNode;
} ListHead;
```

在数据结构中，与给定协议（IP、TCP、UDP 或 ICMP）关联的规则存储在单独的规则树中。在 RuleTreeNode 中的主要字段是源 IP 集合、目的 IP 集合、源端口、目的端口。由于 RuleTreeNode 结构中有许多字段，所以这里我们就不介绍了。在这些字段中，目的端口通常是作为唯一的端口对 OptTreeNode 中规则分类。例如，与 SMTP 关联的规则放在一起成为一组，端口 25 意味着在该组中的规则。因此，Snort 可以通过查看唯一的端口号来决定匹配哪一个规则组。在 RuleTreeNode 中有一个指向 OptTreeNode 节点列表的字段。每个节点存储每个规则中的规则选项（包括内容特征）。因此 Snort 可以遍历通过 OptTreeNode 节点的列表，并逐个地进行规则匹配。

由于涉及匹配规则的线性，所以 Snort 将在同一列表中的内容特征也类到一个集合中并一起匹配它们，称为集合感知（set-wise）匹配。如果找到了特征，接下来将会检查关联的 OptTreeNode 节点中余下的规则选项。如果在该节点中所有的规则选项都匹配，那么规则匹配并产生相应的警报。由于有了集合感知（set-wise）匹配，所以 Snort 并不逐一遍历节点列表，匹配的效率得到提高。

算法实现

预处理

分组可能是 IP 分片或 TCP 分段（特别是较小的）。它们可能会妨碍分组中特征的正确检测。例如，

一个包含特征“bad”的分组，可以分成分别包含“b”、“a”和“d”的单独分片。如果 Snort 逐个地检查，就找不到特征。因此，IP 分片或 TCP 分段在进一步检查之前首先应该重组以恢复原来的语义。

此外，HTTP 请求可能以多种方式编码，并使分析复杂化。在分析之前应该规范化这些请求。请注意，在前面例子中的分片/分段或不同的编码可能是攻击者故意而为之以逃避 Snort 的检测。该技术被称为规避，这种攻击应该能够在最新的 NIDS 中加以处理。而且，IP 地址和端口号的多种检查也可以审计以便确定是否发生了端口扫描。简而言之，在检测分组之前应分多个阶段进行预处理。

为了可扩展性，Snort 利用多个预处理的外来现预处理。这些插件可以将它们的函数挂接到一系列表上，在 Preprocess 函数（在 detect.c 中）中被调用。Snort 请遍历列表逐个地调用它们。下面的代码介绍了如何调用预处理器。

```
int PreprocessPacket * p)
{
  ...
        PreprocessFuncNode *idx = PreprocessList;
        /*
        ** Turn on all preprocessors
        */
        SetAlltPacket(p->preprocessor_bits);
        for (; idx  = NULL && !(p->packet_flags &
PKT_PASS_RULE); idx = idx->next)
        {
            if ((p->packet_bits & idx->proto_mask) ||
(idx->proto_mask == ANY_IP_PROTO_ALL))
                DebugPrint(DEBUG_DECODE, "Preprocessor: %s\n",
                idx->entryFunc(p, idx->preprocessor_bits,
                    idx->context);
            }
        }
}
```

分组解码

经过预处理后，分组解码器解码协议栈中每一层的分组头部。下面是在 fpEvalPacket() 函数中的示例代码（在 fpdetect.c 中）。

```
int fpEvalPacket(Packet *p)
{
  ...
  if( ip_proto = GET_IPH_PROTO(p),
  switch(ip_proto)
  {
     case IPPROTO_TCP:
        DEBUG_WRAP(DebugMessage(DEBUG_DETECT,
                   "Detecting on TCP\n"));

        if(p->tcph == NULL)
        {
            ip_proto = -1;
            break;
        }
        return fpEvalTcpHeader(p);

     case IPPROTO_UDP:
        DEBUG_WRAP(DebugMessage(DEBUG_DETECT,
                   "Detecting on UDP\n"));

        if(p->udph == NULL)
        {
            ip_proto = -1;
            break;
        }
        return fpEvalUdpHeader(p);
  ...
  }
}
```

从示例代码中，你可以看到对上层协议（TCP、UDP 或其他）的分组头部进行解析，然后利用相应的步骤进一步进行解码。

规则检测

检测引擎检查检测规则中的很多选项。为了可扩展性，它们也可以在插件中实现（在 detection-plugins 目录中）。在选项中，content 和 pcre 是最重要的，因为它们指定了在固定字符串和正则表达式中的恶意特征用于模式匹配（分别在 sp_pattern_match.c 和 sp_pcre.c 中执行）。然而，仅指定特征是不够的，因为它们可能只在某些情况下才能作用（例如，在应用内容的某一字段或位置）。如果没有将特征限制在一定的上下文内，那么你会收到许多误报。有些选项有助于精确指定上下文。例如，distance 选项指定 Snort 应该深入到分组中何种程度来寻找某个特征。byte_test 和 byte_jump 选项可以解析应用字段，并跳过某些不会出现特征的字段。除了指定上下文的选项外，某些选项可以指定在警报或规则标识符中出现的消息。有兴趣的读者可以参阅 Snort 手册。

日志和警报

日志和警报包括多种记录和报警模式。扩展性可以在插件中实现。因为它们与网络安全关系不大，这里我们就不再详细讲解了。

练习

1. 列出 Snort 中的 5 个预处理器并研究其中每一个的执行流程。
2. 找出在 Snort 中何种多字符串匹配算法用于特征匹配以及这种算法是在哪里实现的。

垃圾邮件过滤

像入侵检测一样，识别和过滤垃圾邮件也涉及扫描邮件消息（重组分组内容）查找垃圾邮件特征。与入侵检测不同，在垃圾邮件过滤规则中的特征匹配只意味着该邮件更可能是垃圾邮件，但正常邮件也可能会具有规则中所描述的特征。只根据规则来判断一个邮件消息会导致很高的误报率。通常还要根据从规则匹配中累积的足够多的证据来做出决定。通过研究一个流行垃圾邮件过滤软件包 SpamAssassin 的开源实现，我们来学习如何过滤垃圾邮件。

开源实现 8.8：SpamAssassin

概述

SpamAssassin 是一个开源软件包，它可以识别和过滤垃圾邮件——那些不请自来的电子邮件。它是利用 Perl 实现的，并可以与邮件服务器一起在用户接收到垃圾邮件之前进行过滤。过滤机制包括针对邮件标题和文本的分析、贝叶斯过滤和 DNS 的黑名单等方法。为了灵活性，这些分析步骤是以插件形式实现的。

框图

Mail::SpamAssassin（在 SpamAssassin.pm 中实现）是一种使用一组规则对邮件头部和正文进行测试以识别垃圾邮件的 Perl 对象。图 8-27 给出了分析邮件并确定它是否是垃圾邮件的主要执行流程。check_message_text 方法依次调用 parse 和 check 用于垃圾邮件的分析。前者将原始邮件内容（例如，从 MIME 结构中解码）解析为 Mail::SpamAssassin::Message 对象，以便后面对垃圾邮件做检查。

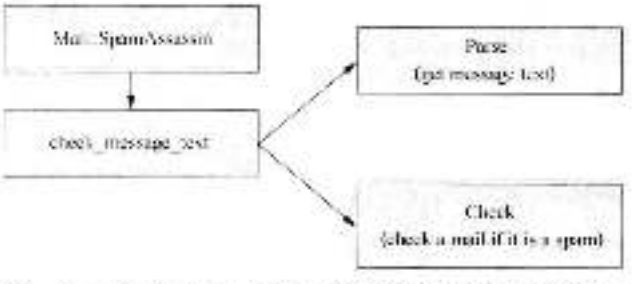


图 8-27 Mail::SpamAssassin 中垃圾邮件分析的主要函数

在 Mail::SpamAssassin::PerMsgStatus 中的后一种 check 方法将运行 SpamAssassin 规则检查邮件消息。该进程调用用于各种分析方法的很多插件，如 HeaderEval.pm 和 BodyEval.pm 用于解析邮件头部和正文。还有一些有趣的插件，如用于计算邮件中图像数量的、发现图像文件的 imageinfo.pm 插件等。检查票和表示垃圾邮件可能性的得分。换句话说，得分越高，邮件就越有可能是垃圾邮件。如果找到垃圾邮件并启用贝叶斯学习，SpamAssassin 就可能调用 learn 方法以便从垃圾邮件报告中进行学习。

数据结构

Mail::SpamAssassin 对象在 SpamAssassin 中有多个重要的属性。它们中的 conf 用于配置信息、plugins 用于插件处理程序和多个路径变量，如规则路径。正如我们已经提到的，在分析垃圾邮件期间会调用插件处理程序。在解析邮件后，邮件消息封装在 Mail::SpamAssassin::Message 对象中，在垃圾邮件分析期间，得分可储在 Mail::SpamAssassin::PerMsgStatus 对象中。

算法实现

SpamAssasin 通过很大的规则集合过滤电子邮件，规则文件位于 rules 目录下的 *.cf 文件中。请注意，SpamAssassin 以词法顺序读取 *.cf 以便于后面的文件可以覆盖前面的文件。规则描述了垃圾信息的可能特征，通过在 Perl 兼容正则表达式（PCRE）中指定特征。如果找到了特征，指示垃圾邮件的得分就会累积。例如，在用于分析邮件头部的规则文件 20_head_tests.cf 中的样品规则如下：

```
header FROM_BLANK_NAME    From =~ /[:]$ ^[^"]*[:]$/
describe FROM_BLANK_NAME  From: contains empty name
```

规则检查发送者名字的字符是否是空格或空。如果发现一个空白名，SpamAssassin 给分数增加 1.0（如果得分没有显式地指定），这有助于确定该邮件是否是垃圾邮件。每个规则文件包含像上述这样的规则用于垃圾邮件的分析。如果累计得分最后达到了 user_prefs 文件中指定的阈值（默认为 5.0），那么将正在分析的邮件视为垃圾邮件。SpamAssassin 也可以从贝叶斯学习中确定垃圾邮件（参见 sa-learn.raw 文件中的源代码）。这里我们不深入研究这一部分的内容。SpamAssassin 可以根据黑名单或白名单来调整得分。例如，假设发送者发送了一封邮件消息具有 20 得分，该邮件被认为是垃圾邮件。如果稍后又发送了另一封邮件，根据规则得分为 2.0，得分将自动调整。通过增加一个 delta 值 (mean - 2.0) × factor，这里 mean = (20 ± 2.0)/2 = 11。因此，除了手动配置黑名单或白名单外，还可以自动调整。

评估类些复杂规则的函数可以用插件来实现，例如 HeaderEval.pm 和 BodyEval.pm，在目录 lib/Mail/SpamAssassin/Plugin 下以便于扩展。例如，在 20_head_tests.cf 文件中的 check_illegal_chars 用于检查应该用 MIME 编码的 8 位和其他非法字符。评估不能简单地用 PCRE 表示，因此它作为一个插件函数来实现。

spamd.raw 程序是 SpamAssassin 的后台守护进程。它加载 SpamAssassin 的过滤器，然后侦听传入的请求来处理邮件消息。在默认的情况下它监听 783 端口，但端口号在命令行是可配置的。当接收到一个连接时，它产生一个子进程处理 SpamAssassin 中来自网络的邮件，并在关闭连接之前将处理过的消息转储返回给套接字。下面是从 spamd.raw 产生一个子进程的实现代码。

```
sub spawn {
  ...
  $pid = fork();
  die "spamd: fork: $!" unless defined $pid;
  if ($pid) {
    ## PARENT
    $children{$pid} = 1;
    info("spamd: server successfully spawned child
process: pid $pid");
    ...
else {
    ## CHILD
  ...
    $spamtest->call_plugins("spamd_child_init");
  ...
}
```

spamd 依赖于 SpamAssassin.pm，它是 SpamAssassin 的主要组件。它处理邮件的解析和检查（通过规则评估、学习、黑/白名单侦听等），并使用前面提到的机制评估邮件是否是垃圾邮件。如果邮件是垃圾邮件，SpamAssassin.pm 将调用 report as spam 函数返回一份报告。

练习

1. 为什么 SpamAssassin 是用 Perl 实现的，而不是用 C 或 C++ 实现的？
2. 讨论使用几种垃圾过滤器与基于规则的方法的优缺点。

性能问题：比较入侵检测、杀病毒、垃圾邮件过滤、内容过滤和 P2P 分类

许多网络安全软件包利用字符串匹配算法来匹配特征。图 8-28 画出了字符串匹配函数的平均执行时间，针对每种程序处理一个字节的分组数据（即应用头部和有效载荷）。值得注意的是，一个程序的总执行时间随着字符串匹配函数消耗时间的增加而增长。入侵检测系统，即 Snort，是最有效的程序，处理每字节花费的时间小于 10 ns；而杀病毒程序，即 ClamAV，消耗的时间是 IDS 的 1000 倍，即 10 微秒。其他程序（包括内容过滤，即 DansGuardian；垃圾邮件过滤，即 SpamAssassin；P2P 分类，即 L7 过滤器）处理一字节的花费大约为 100 ns。

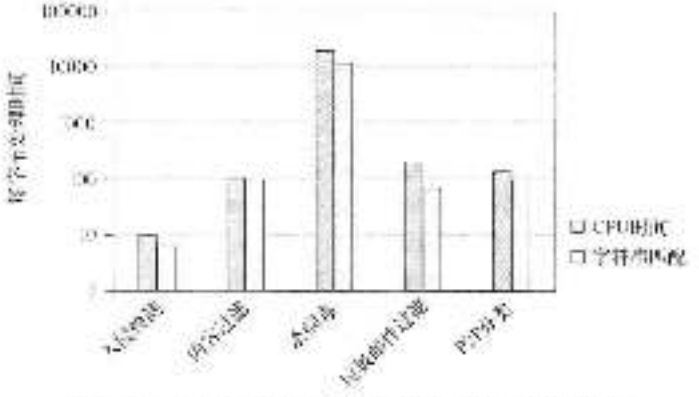


图 8-28 各种应用程序的执行时间和字符串匹配时间

这些程序的性能与字符串匹配所在位置以及如何工作高度相关。因此，表 8-4 中从花费在字符串匹配上的时间百分比以及字符串匹配应用的位置对 5 个软件包进行了比较。Snort 的整体性能是有效的，因为它采用字节跳转的字符串匹配，即仅检查分组中特定偏移量的内容以加速匹配任务。这是因为攻击大多嵌入在特定应用的头部而不是在有效载荷中。DansGuardian 通过使用它的黑名单数据库来授权 HTTP 分组，其中文档记录了数百个被拒绝的 URL、文件类型和关键字。DansGuardian 的数据库比杀病毒程序的更简单也更小，后者通常至少包含数万个特征。因此，尽管 DansGuardian 花费在字符串匹配上的时间百分比高于 ClamAV，但每次字符串匹配的时间比 ClamAV 少得多。最后，SpamAssassin 通过跳过电子邮件中的附件来降低字符串匹配的开销，L7 过滤通过仅检查连接的前几个分组减少字符串匹配的开销。

表 8-4 应用程序进行的字符串匹配

	Snort	DansGuardian	ClamAV	SpamAssassin	L7 filter
字符串匹配所占的百分比	62%	86%	57%	31%	70%
检测深度	字节跳转	Http 请求/响应	所有的附件内容	邮件标题/正文	前 10 个分组

8.5 总结

我们将网络安全问题分为数据安全、访问安全和系统安全。数据安全涉及通过密码学保护和验证在互联网上传输的数据。加密和解密的密钥可以是对称的，也可以是不对称的。前者计算速度更快，但问题在于如何可靠地分发密钥。后者不需要分发对称密钥，但对于大数据量计算时间会很长。安全

协议，如安全套接字层（SSL），在实际网络协议中实现这些加密机制。除了数据保护外，数字签名和消息认证也可以保证收到的数据是原来发送的数据。

访问安全控制内部和外部网络之间的访问，因此在两个网络之间的流量必须遵守网络管理员制定的规则。网络设备（如防火墙）可以执行访问策略。这些设备可以根据 IP 头部中的信息、TCP/UDP 端口号或应用程序头部和有效载荷来过滤网络流量。

系统安全的目的是保护系统不受外部攻击。攻击者（也许是一个自动攻击程序，如蠕虫）可能试图找到系统的漏洞，然后利用漏洞控制或禁用系统。防御系统应该在攻击者之前找到漏洞并为它们打补丁，找到可能的攻击，阻止攻击进入系统。这就是为什么要有漏洞扫描、入侵检测系统，病毒扫描和垃圾邮件过滤器。

攻击者和防御者之间的斗争是无止境的。由于我们的日常生活严重依赖于互联网，所以网络安全已成为重要的课题。如今，只用加密来保护数据是不够的。攻击者将尝试利用潜在的漏洞并以隐身的方法访问系统，所以防御者就需要 1）尽可能地消除漏洞；2）有效地检测攻击。前者不是一件小事情，因为软件越来越大，也越来越复杂，所以寻找可能存在的漏洞也越来越困难。用户应经常打补丁来消除漏洞。后者也是一个挑战，因为攻击者将发现每一种可能的方式来逃避检测。更糟糕的是，攻击者还可以利用加密来保护攻击，例如，加密恶意程序或者加密在互联网上传输的恶意内容，使有效的检测面临比以往更大的挑战。即使我们可以为入侵检测设计一种智能和复杂的方法，但随着互联网流量的增加，我们还应该关注如何加速入侵检测。因此，利用硬件加速或者多核处理器来加速入侵检测或病毒扫描也是一种趋势。

常见陷阱

何时使用： DES、 3DES 和 AES

给出了多种对称加密算法，如 DES、3DES 和 AES，对何时使用哪个很难做出选择。正如本章所讨论的，DES 密钥只有 56 位，使用蛮力就可以破解密钥。这就是为什么出现 3DES，但 3DES 在计算成本上增加了 3 倍。需要记住的是，这两种算法设计是硬件实现的，因为软件实现位操作（如替代和置换）很慢。随着 AES 在 2001 年前后的出现，它越来越受欢迎，可以在如名软件（如 SSH 客户端或 Skype 等）中发现它。AES 最终在数年中替代 3DES，因其在实现上的优势。

无状态和有状态的防火墙

防火墙可以是无状态的也可以是有状态的。无状态防火墙单独检查分组，而不知道连接的存在。在某些应用中这种无状态性质可能是不够的。例如，当 FTP 客户端以主动模式连接到服务器时，它将告诉服务器它监听的端口用于服务器重新连接。如果客户端一侧的防火墙不知道 FTP 命令通知的监听端口，那么由于防火墙的阻塞，服务器将无法重新连接。另一种情况是，通过在一组众所周知的端口中选择一个源端口，攻击者可能使流量像请求的应答一样。由于防火墙通常允许客户端从内部网络连接到外部网络，所以防火墙通常在响应中不阻塞流量。如果防火墙没有记录在连接中曾经有一个请求，那么它就没有办法区分响应是假的还是真的。那么，攻击者通过伪造响应就有机会穿透防火墙。在这些情况下，利用有状态的防火墙来保持有关连接的信息可以解决这个问题，尽管这会需要更复杂的设计。

恶意软件： 病毒、 蠕虫、 特洛伊木马、 后门程序、 僵尸及其他

Malware（恶意软件）是“malicious software”的缩写，它的目的是危害计算机系统。恶意软件有各种形式：病毒、蠕虫、特洛伊木马、后门、僵尸等。这些术语是根据其传播策略或恶意行为而得名的。然而，它们也被公众称为“病毒”。太多的术语看起来很混乱也容易误导，并且它们也很容易被误用。

请注意，计算机病毒最初定义为：“通过修改其他程序以便包括自身的一个演变的副本以此来感染其他程序的一种程序。”此定义不再能够形容恶意软件的多样性。尽管使用术语“病毒”来指代所有类型的恶意软件是不正确的（例如，开源病毒扫描器实际上扫描病毒、木马、后门等），但网络安全

专业人员知道这些术语之间的区别。

警告： 在沙盒中的恶意软件分析

特征匹配是一种用于恶意软件检测的常见技术。但是，许多恶意程序可以通过多态性代码很容易地逃避检测，这是使用相同语义的指令，或者通过打包（压缩或加密代码）。使特征匹配或代码的静态分析难以进行。

另一个常用的方法是在沙盒上运行一个可疑程序，这实质上是一台虚拟机。由于执行环境是虚拟的，所以任何对环境的损害可以很容易地恢复而不会伤害真正的系统，并且运行条件也容易控制。动态分析可以查看调用的系统调用、文件和注册表的变化，以及在运行程序中的网络活动。

尽管动态分析似乎可行，但恶意软件编写者还有一些技巧使分析不可靠。首先，恶意软件可能尝试检测虚拟机的存在。如果虚拟机存在，恶意软件会假装表现良好，并正常地退出。据我们所知，存在的虚拟机总可以以某种方式被检测到，甚至最新的虚拟化技术，如Intel VT（虚拟技术），也是如此。其次，恶意软件的恶意行为可能会被某些值或条件所触发。例如，恶意软件仅在一个特殊的日期或者与一个给定的文件共存时才会被激活。因此，单独运行就难以确定有关的恶意行为。虽然有些研究工作在执行过程中尝试分析可能的分支以找出隐藏的行为和触发值，但由于触发条件的多样性，它们的方法在一般情况下可能无法正常工作。因此，应仔细地解释恶意软件分析的输出，否则会有漏报。

进一步阅读

通用问题

以下是对计算机和网络安全进行一般介绍的教科书和杂志。对于想要在该领域深入学习的同学，这些参考书为迅速掌握基本概念提供了学习材料。

- W. Stallings, *Cryptography and Network Security*, 4th edition, Prentice Hall, 2005.
- C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*, Prentice Hall, 2002.
- W. Stallings, *Network Security Essentials: Applications and Standards*, 3rd edition, Prentice Hall, 2006.
- *IEEE Security & Privacy Magazine*.

加密和安全协议

下面选择的书籍和文件与加密和安全协议有关。加密技术本身是一个广泛的研究领域，不可能在短短的几节中介绍清楚。首先要对该领域有一个很好的了解。我们也为下面的安全协议提供一些参考资料。

- J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*, Chapman & Hall, 2007.
- R. Rivest, "The MD6 Message-Digest Algorithm," Apr. 1992, http://sunsite.auc.dk/RFC/rfc/rfc1321.html.
- MIT distribution site for PGP (Pretty Good Privacy), http://web.mit.edu/network/pgp.html.
- The OpenSSH Project, http://www.openssh.com.
- S. R. Fluhrer, I. Mantin, and A. Shamir, "Weakness in the Key Scheduling Algorithm of RC4," *Lecture Notes in Computer Science (LNCS)*, Vol. 2259, pp. 1-24, Aug. 2001.

网络安全设备和监控

常见的网络安全设备包括防火墙、虚拟专用网络、入侵检测系统、杀毒系统、内容过滤器及其他。前3本书介绍了防火墙、VPN设施和一个著名的入侵检测系统Snort。第4本书介绍了一些网络安全监

控工具和技术。最后两篇是广为引用的有关 Snort 的入侵检测以及检测第三扫描的论文。

- E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls*, 2nd edition, O' Reilly Media, 2000.
- R. Yuan, T. Strayer, and W. T. Strayer, *Virtual Private Networks: Technologies and Solutions*, Addison-Wesley, 2001.
- B. Caswell, J. Beale, and A. R. Baker, *Snort IDS and IPS Toolkit*, Syngress, 2007.
- R. Bejtlich, *The Tao of Network Security Monitoring: Beyond Intrusion Detection*, Addison-Wesley, 2004.
- V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *USENIX Security Symp.*, Jan. 1998.
- J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," *IEEE Symp. On Security and Privacy*, May 2004."

黑客技术

下列书籍介绍了黑客软件、系统和网络漏洞的技术。读者可以从这些书籍的材料中知道黑客会做什么。后三篇是有关网络上攻击技术和事件的广为引用的论文。

- J. Scambray, S. McClure, and G. Kurtz, *Hacking Exposed: Network Security Secrets & Solutions*, 6th, McGraw-Hill, 2009.
- J. Erickson, Hacking: *The Art of Exploitation*, 2nd edition, No Starch Press, 2008.
- S. Harris, A. Harper, C. Eagle, and J. Ness, *Gray Hat Hacking: The Ethical Hacker's Handbook*, 2nd edition, McGraw Hill, 2007.
- G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley, 2004.
- C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a Denial of Service Attack on TCP," *IEEE Symp. Security & Privacy*, May 1997.
- S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time," USENIX Security Symposium, Aug. 2002.
- M. Handley and V. Paxson, "Network Intrusion Detection: Evasion, Traffic Normalization and End-to-End Protocol Semantics," USENIX Security Symposium, Aug. 2001.
- M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon," ACM Internet Measurement Conference (IMC), Oct. 2006.

恶意软件检测和分析

第1本书介绍了为拒绝攻击和恶意软件所建立的蜜罐。接下来的两本书则为病毒研究和实现实践以及检测 rootkits 提供了很好的叙述。后三本是广泛引用的有关恶意软件分析的论文。

- N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, Addison-Wesley, 2007.
- P. Ször, *The Art of Computer Virus Research and Defense*, Addison-Wesley, 2005.
- B. Blunden, *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*, Jones & Bartlett, 2009.
- C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security & Privacy*, Vol.5, Issue 2, pp.32-39, Mar. 2007.
- G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation," USENIX Security Symposium, June 2007.
- N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost in the Browser: Analysis of Web-based Malware," USENIX Workshop on Hot Topics in Botnets (HotBots),

Apr. 2007."

常见问题解答

1. 将私钥算法与公钥算法进行对比（比较它们的计算复杂性、安全性和使用）。
   答：计算的复杂性：公钥算法 > 私钥算法。
   安全性：公钥算法 > 私钥算法。
   用途：公钥算法用于小数据，私钥算法用于大数据。
2. 我们如何才能将私钥算法与公钥算法结合起来？
   答：使用公钥算法（如 RSA）传输私钥算法（如 3DES）中使用的密钥。一旦完成，就可以使用私钥算法进行数据加密和解密。
3. 我们在何种情况下使用传输模式 IPSec 和隧道模式 IPSec？它们加密了分组中的哪些部分？
   答：传输模式：用在远程客户端和办公室之间，仅加密 TCP 或 UDP 分段。
   隧道模式：用在分支机构之间，加密整个 IP 分组。
4. 在隧道模式 IPSec 中，如果认证是在加密之前完成，那么在分组中将有什么样的头部顺序？（头部包括 AH、ESP、IP、TCP 或 UDP。）
   答：IP、ESP、IP、AH、TCP 或 UDP。
5. 分组过滤器与应用代理防火墙之间有什么不同？（比较它们的目的，以及它们在 Linux 系统中的何处实现。）
   答：分组过滤器防火墙：基于五元组的访问控制，在内核（iptables）中实现。应用代理防火墙：基于应用请求和响应的访问控制，在一个代理守护程序中（FWTK 或 squid）中实现。
6. 病毒与蠕虫有什么不同？（比较它们的特性并复制模型。）
   答：病毒：附加在文件上的程序，通过电子邮件附件或 Web 网页。
   蠕虫：一个独立的程序，通过安全漏洞进行攻击。
7. DoS（拒绝服务）与缓冲区溢出攻击有哪些不同？（比较它们的目的和操作。）
   答：DoS：耗尽或阻塞服务资源，发送大量的耗尽服务器的请求或者仅发送一个恶意请求来运行服务器进入阻塞或死锁模式。
   缓冲区溢出攻击：在受害者主机上放置一个后门程序，通过传递一个过大的参数来使受害者主机的栈程序计数器及其程序代码产生溢出。
8. 一个 IDS（如 Snort）在什么情况下将会有误报或漏报？
   答：误报：特征太短并且正常文本中碰巧包含特征。
   漏报：特征不够通用未能匹配文本中的入侵。

练习

动手练习

1. crypt 函数是根据 DES 算法及其变种来进行密码加密的。请编写一段程序找到利用 crypt 函数（其用法参见帮助页面）将密码加密为 "xyNcx0cazF87U"。读者可以使用蛮方法或任何其他聪明的方法（首选）进行猜测。密码并不是一个随机字符串，所以读者有机会来破解它。
2. 设置 iptables 来阻塞到某一个 IP 地址的外出连接，并检查阻塞是否有效。
3. 在使用 RSA 的公钥系统中，Bob 拥有私钥 $d = 5$、$n = 35$，得到给他的密文 $c = 10$。那么明文 M 是什么？
4. 使用 Nessus（http://www.nessus.org/nessus）找到在你的子网中主机上运行的服务，并指明服务是什么。是否有服务没有运行在众所周知的端口上，如 Web 服务没有运行在端口 80 上？
5. 运行 Snort 监听指定接口上的流量。流量报多或好。Snort 能够支持哪些警报？你认为它们是误报吗？你可以将流量捕获到一个文件中并让 Snort 离线读取文件。因此，你有机会手动分析产生警报

的分组或连接。

6. 跟踪 Snort 源代码的预处理器目录下的 portscan.c 和 spp_portscan.c 中的源代码。需要地描述 Snort 如何检测流量跟踪中的端口扫描。
7. 找出当前 Snort 规则集中有多少条规则。
8. 跟踪 matches-so.c 和 matches-len.c 中的源代码，并描述 ClamAV 如何扫描病毒特征。
9. 使用 UPX 打包器（http://upx.sourceforge.net）以 PE 格式打包一个 Windows 二进制可执行文件。在这之后，使用 PE 浏览器，如 Anywhere PE viewer（http://upx.sourceforge.net/），指出已经发生了哪些改变。

书面练习

1. 在 DES 系统中每次迭代的主要加密函数是什么？
2. 计算 32、56、128 和 168 位密钥的破解时间，如果单次解密时间分别为 1μs 和 10^{-6}μs。
3. 在一个使用 RSA 的公钥系统中，公钥为 $e=5$，$n=35$，Trudy 拦截到密文 $C=10$，那么明文 M 是什么？
4. 在软件中执行 DES（或者 3DES）算法有效吗？为什么要使用硬件加速实现？
5. 数字签名的需求是什么？
6. 网络防火墙和应用层防火墙之间的区别是什么？
7. DDoS 攻击的步骤是什么？2001 年 10 月 Nimda 蠕虫的攻击步骤是什么？
8. 在 IPSec 中通过使用认证头部（AH）和加密安全有效载荷（ESP），如何同时实现认证和隐私安全？
9. 一次攻击如何拥有大量主机以便发起一次分布式拒绝服务（DDoS）攻击？请讨论可能的方法。
10. 一个 NIDS 有能力“看到”网络上的分组，因此它就有机会扫描分组有效载荷中的病毒。但是在实际中却不是这样的。尝试这么做会有哪些困难？
11. NIDS 警报中产生误报的可能原因有哪些？
12. 请列出一个不能被 Snort 规则检测到的攻击例子，即使你尝试写一个新的规则也检测不到。为什么它不能被规则检测到？
13. 躲避 NIDS 分析的可能技术是什么？请列举出其中几个。
14. ClamAV 声称有一个非常大的特征集（大于 500 000）。在实际中即在互联网上真的有这么多病毒吗？需要这么多特征的可能理由是什么？

名人录

许多组织机构和个人曾经为数据通信做出过突出的贡献。然而，这里不可能——列举所有的人。由于本书的主题是互联网体系结构及其开源实现，所以我们将重点放在两个部分，即互联网工程任务组（IETF）和几个开源社区。前者定义互联网的体系结构，而后者以开放源代码的方式加以实现。其他标准机构和网络研究团体主在发挥着重要作用，因此共有成员了每个的互联网而自己却没准备头了的技术。它们在演变中被淘汰了。尽管在本附录中给出的材料都是非技术性的，但它们给出了一条回顾今天互联网所有一切的途径。了解进化及其背后的驱动力会让读者欣赏这些努力工作，并将鼓励读者以受身于正在进行的演变之中。

与许多其他标准组织不同，IETF 并没有一个明确的成员身份制度，并且它运以自愿为主的而不是自上而下的方式运作的。欢迎任何人参与其中，其中活跃的人员将会领导工作。你不必为参与的工作支付费用。此外，与定义这点，与广多组织中“无所定原则”相似。“我们拒绝国王、总统和投票。我们相信粗略地达成一致意见和运行代码”，互联网体系结构的一个重要贡献者 David Clark 曾经这样说。设计一个标准 RFC 文档的过程看起来很简单，但是一旦标准达成一致就要平少转换成一个（最好有两个）扎实的和公开提供的实现。我们可以说，互联网体系结构的标准化进程具有开源精神，可以作为验证建议的解决方案能够很好工作的一种方法。

虽然开源运动起于 1984 年，也就是于 1969 年建立第一个互联网节点的 15 年后，但它是与互联网携手工作的，因为它们能够相互利用。互联网提供了指导性的标准，以维护各种开源实现的互操作性，并且来自当地社区分布在世界各地研究努力的一个平台。开源运动推进了互联网的“边制定边实施”的标准化进程，这有助于吸引全世界的贡献者和用户。如果它们不是开源代码，就很难向世界各地的用户分发这些运行代码以便于被采用或者协调分布的工作来使代码得以固定和运行。

除了 IETF 外，还有一些研究机构和标准机构主在帮助设计协议或实现所设计的协议。南加州大学（USC）的信息科学研究所（ISI）设计并实施了多个关键的协议。伯克利的国际计算机科学研究所（ICSI）开发了一些重要的算法和工具。卡内基－梅隆大学的计算机应急反应小组（CERT）协同安全威胁管理。欧洲电信标准协会（ETSI）制定了移动通信标准。最流行的链路层协议标准，包括以太网和无线局域网，是由电气和电子工程师学会（IEEE）学会制订的。与此同时，许多研究人员为互联网上使用的体系结构、协议或算法做出了重要的贡献。所有这些贡献应该得到承认。

当我们回顾名人录时，我们不仅应该学习保留下来的技术也应该学习过时的技术。它们可能是在发展之前曾经盛极一时的“恐龙”，或者它们曾经吸引了巨大的投资但最终破裂的“泡沫”。它们失败的原因不能是技术上的主可能去直接上的。一项需要与大的开展进行互操作或者替换现有技术的革命技术，可能作为历史的一部分而终结。处于劣势但更简单的解决方案可能会抢垮更多高级的竞争对手。达成的共识是“无处不在的 IP，或一切都通过 IP 和 IP 超越一切”。同样，“以太网无处不在”（进入办公室以及家庭）已经成为另一个共识。IP 和以太网无非在所有方面都是优秀的，但它们已经占据了主导，并将很好地延续到未来。

在 A.1 节中，我们首先回顾 IETF 的历史，以及它是如何生成 RFC 的。统计数据显示已有超过 6000 份的 RFC 文档。然后我们将在 A.2 节中介绍几个生产实际运行代码的开源社区，超过 10 000 个软件包，其全 ASIC 硬件设计。它们分别从顶部（即应用程序）、中间（即内核和驱动程序）、下到底层硬件（即 ASIC 设计）开放系统。这些开源资源都是可以访问的。网络研究社区和其他标准团体将在 A.3 节中介绍。最后，在 A.4 节，我们回顾过去的已经过时的技术，并试图解释它们为什么会这样。

A.1 IETF：定义 RFC

我们打算在这里回答很多问题。互联网的标准化机构是如何演变的呢？是谁扮演着重要的角色？

IETF 如何定义 RFC？为什么会有这么多的 RFC？这些 RFC 在定义协议栈的各个层次中是如何发布的？这些问题的答案应该能够打开理解及参与 IETF 活动的大门。

A.1.1 IETF 的历史

在 20 世纪 70 年代末，认识到互联网的发展是伴随着对其感兴趣的研究团体规模的增长，因此急需一种协调机制。DARPA 的互联网项目经理 Vint Cerf 组建了多个协调机构。1983 年，这些机构之一变成了管理多个任务组的互联网活动委员会（IAB）。互联网工程任务组（IETF）当时只是许多 IAB 任务组之一。后来，更加实际的和工程方面的互联网显著增长。这种增长导致了在 1985 年参加 IETF 会议的人数爆炸性地增加，时任 IETF 主席的 Phil Gross 被正以工作组（WGS）的形式为 IETF 创建了机构。

这种增长还在继续。IETF 将工作组合并成为区域，并为每个区域指定区域董事，互联网工程指导组（IESG）由地区董事组成。IAB 认识到 IETF 日益增加的重要性，并重组了标准处理过程，明确认可 IESG 作为主要标准审查机构。IAB 也进行了重组，以便使其他的任务组（除 IETF 外）合并到互联网研究任务组（IRTF）中。1992 年 IAB 进行了重组，并更名为互联网体系结构委员会，在互联网协会的监督下运行。一个更加“对等”的关系在新的 IAB 和 IESG 之间定义，与 IETF 和 IESG 一起为标准批准承担更大的责任。

IETF WGs 成员之间的合作主要通过邮件列表，并且每三年举办一次会议。互联网用户可以免费参加 IETF，因此要做的只是通过订阅具体工作组的邮件列表就可以，通过它们可以与工作组的其他成员通信。定期会议旨在能够让积极的 WG 提出并讨论他们的工作成果。截至 2010 年 3 月，IETF 共举行了 76 次会议。每次会议为期 6－7 天，会议地点由主办机构选择。

历史演变：在 IETF 中的名人录

近 40 年来，总共已发布了超过 6000 份 RFC 文档。最著名的贡献者是 Jonathan Postel，他从 1969 年开始直到 1998 年去世，一直担任 RFC 的编辑。他参与了 200 多个 RFC，其中大部分是互联网的基础协议，如 IP 协议和 TCP 协议。仅次于 Jonathan Postel 的是 Keith McCloghrie，是参与 RFC 第二最多的人。Keith 有 94 个 RFC，其中大部分是有关 SNMP 和 MIB 的。表 A-1 列出了按照发布 RFC 数量排名的前 10 名贡献者及其主要贡献。

表 A-1 按数量排序的前 10 名 RFC 贡献者

名　字	RFC 的数量	主要贡献	名　字	RFC 的数量	主要贡献
Jonathan B. Postel	202	IP、TCP、UDP、ICMP、FTP	Henning Schulzrinne	62	SIP、RTP
Keith McCloghrie	94	SNMP、MIB、COPS	Bob Braden	59	FTP、RSVP
Marshall T. Rose	67	POP3、SNMP	Jonathan Rosenberg	52	SIP、STUN
Yakov Rekhter	62	BGP4、MPLS	Bernard Aboba	48	RADIUS、EAP

A.1.2 RFC 过程

IETF 将 WG 分为 8 个领域，每个领域都包含一两个区域董事。大多数的 RFC 在经过特定 WG 的内部工作后发布。图 A-1 显示了 RFC 过程。发布 RFC 需要通过多个阶段，每个阶段由 IESG 来审查。为了发布一个 RFC，首先发布互联网草案（ID），将 ID 放在将被审查的 IETF 互联网草案编号目录中。ID 发表一段时间后（至少两周），ID 的作者就可以向 RFC 编辑发送一封电子邮件，请求 ID 进入知识性的或实验性的 RFC，然后 RFC 编辑要求 IESG 审查该 ID。在它成为 RFC 之前，该草案的作者可以修改其内容。如果在 6 个月内 ID 没有修改或者成为 RFC，那么它将会从 IETF 的 ID 目录中删除，并通知作者。同时，如果 ID 通过审查并准备成为 RFC，那么其作者将有 48 小时检查文档错误，如不正确的拼写错误或错误引用的文件。一旦它成为 RFC，它的内容就不允许进行修改了。

如图 A-1 所示，每个 RFC 有一个状态，包括未知、标准（STD）、历史、目前最佳实践（BCP）和

一般（知识性的或实验性的）。未知状态分配给 IETF 早期发布的大多数 RFC，自 1989 年 10 月以来，一直没有再次出现过。标准 STD 状态表示一个互联网标准，BCP 状态表示完成某事的最佳方式。而一般状态表示 RFC 还没有准备好，或者可能还不打算成为标准。一个 RFC 必须经历三个阶段才成为一个 STD：建议 STD、草案 STD 和标准 STD。这些阶段称为成熟等级，以表示成熟。标准 STD 状态的 RFC 应该通过所有这些阶段。到不同阶段的步骤有不同的限制。例如，如果一个 RFC 是稳定的，解决了已知的设计问题，相信已得到很好理解，并且已经引起团体的足够兴趣，被认为是有价值的，那么它将被授予建议 STD 状态。为了获取草案 STD 状态，一个建议 STD RFC 必须至少有两个独立和互操作的实现并且在处理队列中当少 6 个月。为了从草案 STD 前进到 STD 状态，RFC 就必须有重大的实现和成功的操作经验，而且也必须至少在处理队列中 4 个月。一个规范要一个最新的文档所取代或者因为某种原因被认为是过时的，那么就被赋予历史状态。

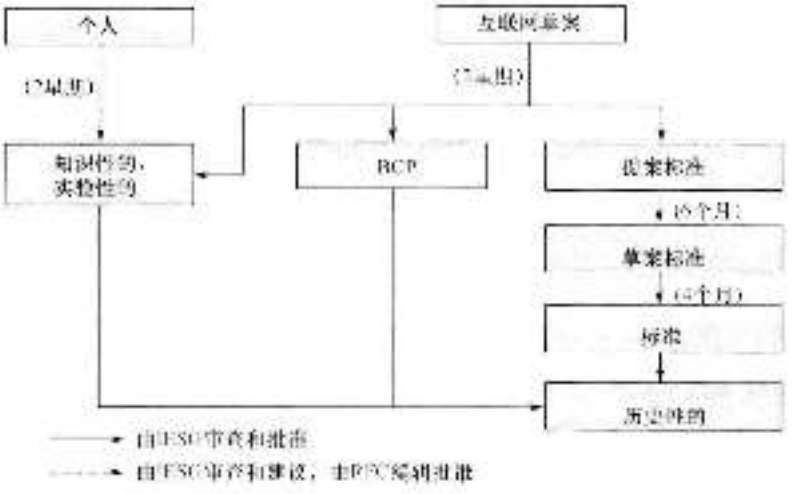


图 A-1 RFC 过程

BCP 过程与建议 STD 过程相似。将 RFC 提交给 IESG 进行审查，一旦得到 IESG 的批准，过程就结束。知识性的和实验性的过程与标准 STD 和 BCP 的过程不同。处于这些非标准状态的将要发布的文档既可以通过 IETF WG 提交给 IESG，也可以由个人直接提交给 RFC 编辑。在第一种情况下，IESG 仍然负责审查和批准文档，与在标准 STD 过程中一样。但是在第二种情况下，由 RFC 编辑做出最终的决策，IESG 仅审查和提供反馈。RFC 编辑首先将此文档作为一个互联网草案发布，等待两周来自社团的评价，从他的专业角度判断它是否适于知识性的或实验性的 RFC，然后做出接受或拒绝。IESG 审查文档并建议是否需要进行标准化。如果文档要建议进行标准化并且作者同意，它将进入标准 STD 过程；否则，它将作为知识性的或实验性的 RFC 发表。图 A-1 显示了用于标准 STD、BCP 和一般状态的 RFC 过程。

RFC 序号是根据批准顺序分配的。某些序号具有特殊的含义。例如，以 99 为结尾的 RFC 序列号表示对该序列的 99 份 RFC 做出简短介绍，而以 00 结尾的序列号表示 IAB 官方指定标准，它提供当前 RFC 标准的简短状态报告。对此感兴趣的读者可以进一步参阅 RFC 2026：互联网标准过程。

A.1.3 RFC 统计数据

截至 2010 年 11 月，RFC 序号已分配到了 6082。其中，有 205 个序列号未使用，所以只有 5877 份 RFC。要了解如何分配 RFC，我们编制了对这些 RFC 的统计数据。图 A-2 给出的统计表明前 3 名 RFC 状态分别为知识性的、提案标准的和未知的。这并不奇怪。RFC 1796 指出："并非所有的 RFC 都是标准"。发布一个知识性的 RFC 比通过 STD 过程更容易。要成为一个标准，需要得到广泛的证明，所以很多的 RFC 停留在提案标准的水平。最后，未知状态排名第三，因为 IETF 还没有发展到成熟水平并审查过程，直到 RFC 1310 出现。

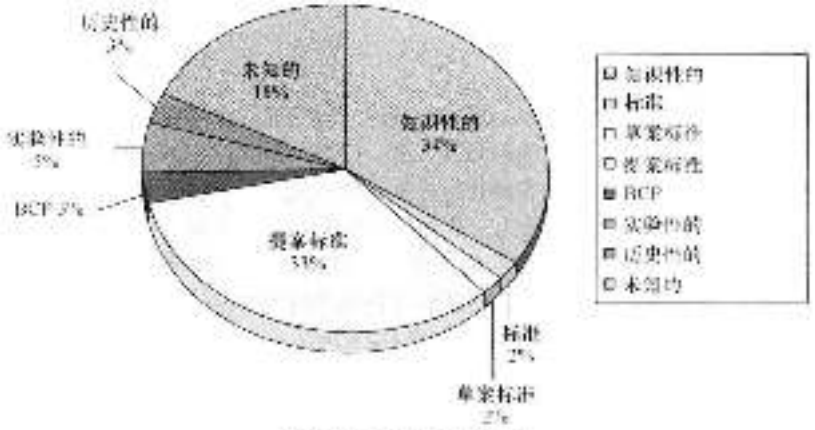


图 A-2 RFC 状态统计

表 A-2 统计与四层中知名协议相关联的 RFC 总数。统计信息包括数据链路层的点到点协议（RFC 1661）、网络层的互联网协议（RFC 791）、传输层的传输控制协议（RFC 793），以及应用层的远程登录协议规范（RFC 854）。事实上，6000 份 RFC 中只有大约 30% 经常用在互联网上，这就提出了一个问题：为什么会有那么多的 RFC？这里有几个原因。第一，一旦产生一个 RFC，没有人可以修改它。因此，许多 RFC 是此时的或者被新的 RFC 所更新。第二，一个单一的协议可能由多个 RFC 的定义。具有丰富功能的单一协议，不可能在其第 1 版就完成，因此当需求出现时就要单独添加新的功能或修改。第三，无数定义新兴技术的 RFC 可能没有部署，因为各种困难或更新的替代品的出现。

表 A-2 1561 份 RFC 定义的未知协议

层	协议	数量	层	协议	数量
数据链路层	ATM	46	传输层	TCP	111
	PPP	87		UCP	21
网络层	ARP/RARP	24	应用层	DNS	105
	BOOTP/DHCP	69		FTP/TFTP	51
	ICMP/ICMPv6	16		HTTP/HTML	37
	IP/IPv6	259		MIME	99
	Multicast	95		SMTP	41
	RIP/BGP/OSPF	154		SNMP/MIB	238
				TELNET	108

以远程登录 TELNET 协议作为例子。大约有 108 个 RFC 描述这个协议。在 108 个 RFC 中，60 定义远程登录 TELNET 的选项，只有 8 个 RFC 描述主协议，剩下的文档是协议相关的讨论、评价、加密方法或体验。这些选项定义为 Telnet 协议出现的新需求，它们使协议的功能更加完善。在 8 个 RFC 中，RFC 854 是定义 TELNET 的最新协议，而其他 7 个已经过时或已被更新。

A.2 开源社区

正如前面提到的，对于每个标准 RFC，实施是必要的并且应该对公众开放以证明其可用性。这样的指导方针推动了开源的发展。事实上，许多开源社区都致力于实施这些新的互联网标准。在介绍这些社区之前，我们打算先回答以下问题：整个开源运动是如何以及为何发起的呢？发放、使用、修改和分发一个开源软件包的运动规则是什么？到目前为止，在应用程序、内核以及 ASIC 设计中已经产生了什么样的运行代码？本综述将引领读者进入开源运动。

A.2.1 开始和开源规则

免费软件革命会

1984 年 Richard Stallman（RMS，www.stallman.org）成立了免费软件基金（www.fsf.org），它是为 GNU 项目（www.gnu.org）分集资金的免税慈善工作。GNU，是“GNU's Not Unix”的递归缩写，读作“new”的同音字。旨在开发与 UNIX 兼容的软件并推动软件免费。建议著作权（copyleft）和通用公共许可证（GPL）以保证这种免费。在不放上，著作权是具有 GPL 规定的著作权。RMS 本身不仅是“传教士”，而且还是最主要的开源软件的贡献者。他是 GNU C 语言编译器（GCC）、GNU 符号调试器（GDB）和 GNU Emacs 等的主要作者。所有这些软件包是 GNU/Linux 中的基本工具。在 Fedora 8.0 的发布中约有 55 个 GNU 软件包，总共有 1491 个软件包。

许可证模式

如何处理开源软件的知识产权是一个有趣的、有时也是有争议的问题。选择一种合适的许可模式向公众发布开源代码很重要。一般来讲，许可证模式有三种你应该要注意：1）它是免费软件吗？2）它有著作权吗？3）它与 GPL 兼容吗？免费软件意味着程序可以自由修改和重新发布。著作权通常意味着放弃知识产权和私人许可证。GPL 兼容软件包是指链接 GPL 软件是合法的。不过，有太多的许可证模型。我们只描述了三个主要的：GPL、LGPL 和 BSD。

通用公共许可证（GPL）是一个免费的软件许可证或著作权许可证。它具有自我延续和传递性，严格确保后续衍生作品遵循相同的许可模式，即 GPL 下发布。Linux 内核本身是 GPL。除了后续衍生作品外，与 Linux 静态链接的程序也应该是 GPL。然而，动态链接到 Linux 的程序不一定是 GPL。更宽松的 GPL（LGPL）一旦被认定是受 GPL，就允许与非免费的（专有的）模块链接。例如，因为有很多其他的 C 库，所以如果 GNU C 库也需要遵守 GPL，那么可以让专有软件的开发人员使用其他的替代库。因此，在某些情况下 LGPL 有助于免费软件以吸引更多的用户和程序员。GNU C 库就是这种 LGPL。另一个选项，伯克利软件分发（BSD）声明代码可以免费分发，并允许衍生作品冠以不同的术语，只要获得必要的许可。Apache 与 BSD 相关的操作系统，以及免费版本的 Sendmail 等都采用 BSD 许可证。总之，GPL 意味着它是公共财产，你不能私人拥有，而 BSD 意味着任何人都可以带走它。所有其他授权模式都介于这两个极端之间。

A.2.2 开源资源

Linux

人们很少提到 Linux 操作系统是 GNU/Linux 的。事实上，Linux 内核是要求每个 GNU 软件包执行所有的指数。1991 年，当时还是芬兰赫尔辛基大学研究生的 Linus Torvalds，编写了一个真正 UNIX 兼容的操作系统，并将它张贴在新闻组 comp.os.minix 上。1994 年以后，他把内核维护交给 Alan Cox，同时他仍然监控内核版本并决定内容的添加和删除，而让其他人处理用户空间问题（库、编译器，以及各种进入 Linux 发行版的工具和应用）。GNU/Linux 已被证明是一种成功的组合。1998 年另一个“传道士” Eric Raymond（www.tuxedo.org/~esr）将这种软件开发中的变革形容为“开源运动”。

软件包分类

开源软件包的数量已超过 1 万个。这个巨大的库可分为三大类：1）带有控制台或 GUI 接口的操作环境；2）提供各种服务的后台守护程序；3）程序员的编程工具包和库。我们深入挖掘这个巨大的库，总结统计数据显示在图 A-3 中。例如，共有 97 种 HTTP 守护程序，Apache 只是其中之一，恰好又是最流行的一种。

Linux 发行版

如果内核是构建的基础，那么每一个开源软件包就是它上面的一块砖，厂商的 Linux 发行版就是一个带有基础、各种砖和装饰家具的建筑物外观。这些厂商调试、整合并将开源软件集成到一起。接下来，我们介绍几个知名的 Linux 发行版。



图 A-3 开源软件包分类

Slackware（www.slackware.com）是一个有着悠久历史的发行版，广为发布，而且大多是非商业性的，十分稳定，易于使用。Debian（www.debian.org）形成并由近千名志愿者维护。许多高级用户已经发现 Debian 发行版的灵活性和试验度。红帽子 Linux（www.redhat.com）由标准普尔 500 公司之一的红帽公司发布，开始使用红帽子包装管理器（RPM）为 Linux 发行版打包，提供比原始 ".tar.gz" 更容易的安装、卸载、升级。RPM 使软件系统也透明，不那么麻烦。红帽公司将红帽 Linux 作为自己的商业产品，红帽企业版 Linux（RHEL），直到 2004 年以后，可能由于版权和专利问题终止了维护红帽子 Red Hat Linux。目前，在红帽子公司资助支持下，RHEL 是从一个由社区支持的发行版名为 Fedora 演变而来。CentOS（社区企业操作系统，www.centos.org）是另一个社区支持的发行版。它提供了一个免费的企业级计算平台，保持与 RHEL 约 100% 二进制兼容性。除 RHEL 和 Fedora 之间的关系一样，SuSE Linux（www.novell.com/linux/）和 openSUSE（www.opensuse.org）分别是由 Novell 公司赞助的企业产品和社区支持的发行版。SuSE 因其良好的文档和丰富的软件包资源而闻名。Mandriva Linux（http://www.mandriva.com/）的前身是 Mandrake Linux，开始只是结合红帽发行版与 KDE（K 桌面环境）等许多其他独特的功能丰富的工具。这种组合结果证明是如此受欢迎，以至于因此成立了 Mandriva 公

司。Ubuntu（http://www.ubuntu.com）取名源自 Bantu 的后“人道地对待他人”，它是一种基于 Debian 并使用 GNOME（GNU 网络对象模型环境）作为多用途图形桌面环境的发行版。因其易于安装和用户友好的界面而闻名。自 2006 年以来，据报道 Ubuntu 成为最流行的发行版。

A.2.3 开源网站

Freshmeat.net 和 SourceForge.net

Web 站点 Freshmeat.net 的建立为 Linux 用户提供了一个平台以便找到并下载开源许可证下发布的软件包。对于每个软件包，除了简要说明外，还包括主页 URL、发布焦点、最近更改和依赖的库。在 Freshmeat.net 中给出 3 个有趣的指标，分别是评级、活力和受欢迎度。用户投票表决机制提供了评级，而其他两个指标的计算根据项目时间、公告数量、上次公告日期、订阅数量、URL 网址点击量和记录次数。除了软件包外，Freshmeat.net 包括许多原创的文章介绍软件和编程。

Freshmeat.net 是由 Geeknet 公司支持和维护的。根据 Web 网站上所得到的统计数字，Freshmeat.net 已经引进了 4 万多个项目。统计还报告了前 10 个项目，按照受欢迎度和活力排序。例如，排在前 10 位中的两个著名项目是 GCC 和 MySQL。GCC 是前面提到的众所周知的 GNU 编译器，MySQL 是互联网上最流行的开源数据库之一。

与 Freshmeat.net 提供信息为用户查找、比较、下载软件包不同，SourceForge.net 为软件包开发者管理项目、发布、通信和编码提供免费的平台。它承载了 23 万个项目！在 SourceForge.net 上最活跃的项目是 Notepad++，而下载次数最多的是 eMule。前者是在 Windows 中使用的文本编辑器，而后者则是一个 P2P 文件共享程序。

OpenCores.org

不仅软件包可以是开源的，硬件设计也可以是开源的。Open Cores.org 社区收集对硬件开发感兴趣并像开源软件一样喜欢与他人分享其设计的人。唯一的区别在于代码是用 Verilog 和 VHDL 硬件描述语言编写的。社团及其门户网站由 Damjan Lampret 于 1999 年成立。截至 2009 年 12 月，在其网站收集了 701 项目，收到 50 万次点击/月。这些项目分成 15 个类，如算术内核、通信内核、加密内核和 DSP 内核。

与 Freshmeat.net 一样，OpenCores.org 也为每个项目维护了一些有趣的指标，如受欢迎度、下载、活动和评级。例如，受欢迎度前 6 名的为 OpenRISC 1000，以太网 10/100Mbps，ZPU，I2C 内核，VGA/LCD 控制器和 Plasma。

A.2.4 重大事件和人物

表 A.3 列出了开源运动中的重大事件。许多捐助者花费时间开发开源软件。这里，我们虽然只提及一些知名的人，但同时也应该感谢那些受公众关注较少的人员。

表 A-3 开源时间线

1969	固特网以 APPAnet 形式出现，UNIX
1979	伯克利软件套件（BSD）
1983	Allman 发布了 Sendmail
1984	Richard Stallman 发起了 GNU 项目
1986	伯克利固特网域名（BIND）
1987	Elaine Ashton 发布了 Perl
1991	Linus Thorvald 编写了 Linux
1994	Allan Cox 接受 Linux 内核维护，PHP 复由 Rasmus Lerdorf 维护
2/1995	Apache HTTP 服务器项目具有 8 个团队成员
3/1998	阿梁塔开源
8/1998	“我们开发会担心了！”——微软总裁 steve Ballmer

（续）

3/1999	苹果公司正式发布了 Darwin（Mac OS X 的内核）
3/2003	Apache Web 服务器软件超过了 1000 万（占据全部市场的 62.8%）
10/2003	推出 StarOffice 光盘
10/2003	英国电信宣布与一家律师在开源软件之下的交易
10/2004	为开源开发者赢得 500 项专利
1/2005	公开了 Solaris 操作系统
5/2007	微软声称 Linux 侵犯了其专利
11/2007	公布了一个称为 Android 的开源移动设备平台
9/2008	微软首次统计宣布认 40% 为 Web 服务器运行在 Windows 上，但 60% 的 Web 服务器运行在 Linux 之上
7/2009	谷歌引入开源操作系统，Google Chrome OS

A.3 研究与其他标准社区

除了 IETF 和计费社区外，还有几个重要的研究机构和标准组织对互联网的演变做出了许多贡献。这里我们介绍他们的情况。

ISI：美国南加州大学信息科学研究所

ISI 是一个针对先进计算机和通信技术的研究和开发中心，成立于 1972 年。ISI 现在有 8 个部门，可以容纳超过 300 名研究人员和工程师。其计算机网络分部是互联网的前身 ARPANET 的发祥地之一。该分部也参与了许多日常使用的互联网协议和软件包（如 TCP/IP、DNS、SMTP 和 Kerberos）的开发。

ICSI：伯克利国际计算机科学研究所

ICSI 是一个独立的非营利的从事计算机科学研究的机构，成立于 1988 年，包括 4 个主要的研究小组：互联网研究、理论计算机科学、人工智能和自然语言处理。互联网研究小组的科学家曾参与了许多知名的和广泛部署的网络算法和工具的研究，如 RED、TCP SACK、TFRC 和网络模拟器 NS2。该小组还对目前的互联网流量和安全状态提出了一系列的测量和意见，这对新网络协议和算法的设计和测试是非常有用的。

CERT：卡内基梅隆大学计算机应急响应小组

正当 Morris（莫里斯）蠕虫在 1988 年造成 10% 的互联网系统停止时，CERT 协调中心和军方在软件工程研究所成立了。该中心的主要工作包括软件保证、安全系统、组织安全、协调响应、教育/培训。CERT 也是万维网的诞生地。

ETSI：欧洲电信标准研究所

ETSI 创建于 1988 年，是一个欧洲电信产业标准化组织。其制定的标准包括固定、移动、无线电和互联网技术。ETSI 推出的最成功的标准是 GSM（全球移动通信系统）。

IEEE：电气和电子工程师协会

IEEE 是电气工程、计算机科学和电子产品最大的专业协会，拥有分布在 150 多个国家的超过 365 000 名会员（截至 2008 年）。该协会已经发行约 130 种期刊或杂志，每年举办 400 多会议，并出版了许多教科书。它也是远信领域国际标准的重要开发者。许多 PHY 和 MAC 协议就是在 IEEE 802 标准系列中规定的，包括 802.3（以太网）、802.11（无线局域网）和 802.16（WiMAX）。这些标准的内容在第 3 章中介绍。

ISO：国际标准化组织

ISO，作为世界上国际标准化组织最大的开发者和发布者，参与几乎所有领域——技术、商业、政府和社会。许多著名的电信系统都是通过国际标准化组织进行标准化的。例如，电话网络就是基于其公共电话交换网（PSTN）标准。ISO 也制定了许多数据网络标准，尽管并非所有这些标准在当前的互联网中使用，如 OSI 的 7 层网络体系结构。

个人贡献

最后，我们感谢某些个人，因为通过他们的努力建立了互联网的基本体系结构。J. C. R. Licklider 和 Lawrence Roberts 在 20 世纪 60 年代领导的 ARPA 项目创建了 ARPANET。Paul Baran、Donald Davies 和 Leonard Kleinrock 早在 1969 年同构建了初始的分组交换式 ARPANET 网常常被誉誉为“互联网之父”。Bob Kahn 和 Vint Cerf 在 20 世纪 70 年代初开发了 TCP 和 IP。Robert M. Metcalfe 和 David R. Boggs 于 1973 年共同发明了第一个以太网技术。Jon Postel 随后为 TCP/IP、DNS、SMTP、FTP 等编写了很多 RFC。David D. Clark 在 20 世纪 80 年代的互联网体系结构开发中担任首席协议体系结构师。Van Jacobson 在 20 世纪 80 年代末对 TCP 拥塞控制做出了贡献。Sally Floyd 在 20 世纪 90 年代开发了 RED 和 CBQ 并改进了 TCP。Tim Berners-Lee 于 1989 年发明了万维网，并导致 20 世纪 90 年代互联网的爆炸性增长。

A.4 历史

本节将介绍互联网上新新的或失败的技术及其失败的原因。图 A-4 显示了这些技术的时间轴。空心条意味着该技术研究了多年，但没有得到部署或为市场所接受，而实心条意味着它已经部署，但被后来的技术所替代或者没有能取代现有的技术。简要的历史如下所述。



图 A-4 某些短命及失败技术的时间线

体系结构标准：OSI

1980 年 Zimmerman 提出了 7 层的开放系统互联（OSI）体系结构，后来在 1994 年被国际标准化组织（ISO）采纳。它旨在取代 1981 年定义的新兴的 4 层 TCP/IP 协议栈。它有两个额外的层，表示层和会话层，认为它比 TCP/IP 结构更完整和更具结构化。但为什么 OSI 却失败了呢？主要有两个原因。第一，TCP/IP 协议已经普遍存在于大多数计算机上运行的 UNIX 操作系统中。也就是说，它有一个巨大的载体渗透到世界各地。第二，所声称的结构化完整性并不重要，只要所有种类的应用能够在 TCP/IP 上顺畅地运行。结果证明，在互联网上没有绝对的权威。即使是由国际标准组织批准和支持的技术也可能会失败。而且 OSI 只是许多失败例子之一。

综合服务：ISDN、B-ISDN 与 ATM

在同一个网络上承载语音和数据是电信行业的一项长期努力。最早可以追溯到 20 世纪 80 年代中期，当时该行业只有 POTS 用于语音服务，而推出面向连接的 X.25 大多为金融应用和企业网络连接提供有限的数据服务。X.25 是一项成功的服务，在 20 世纪 90 年代逐渐被轴中继所取代，然后在 21 世纪 00 年代通过数字用户线（DSL）的 IP 作为数据服务向互联网演变。但数据服务仍然与语音服务分开，直到在 20 世纪 80 年代末综合业务数字网（ISDN）作为第一次将这两个公共服务结合的尝试为止。ISDN 综合用户接口访问数据和语音服务，但仍然有两个单独的骨干网络，一个电路交换承载声音，另一个分组交换（但非面向连接的）承载数据。ISDN 是温和但短暂的，在 20 世纪 80 年代末到 20 世纪 90 年代中期许多国家的服务提供商取得了成功。

为了消除 ISDN 在固定用户接口、宽带宽和单独骨干网的限制，在 20 世纪 90 年代初期向 ITU 提出了宽带 ISDN（B-ISDN）以提供灵活的接口、宽带服务和统一的骨干网——这依赖于信元交换异步传输模式（ATM）技术，ATM 中的一个信元是一个固定大小、53 字节的分组，以方便通过硬件交换。与 OSI 的命运相似，ATM 是一种完整的、复杂的技术，但它需要与已经在公共数据网络中占主导地位的

TCP/IP 协议共存。这种共存是很高兴的，因为面向连接的信元交换 ATM 和无连接的分组交换 IP 的神奇性质，既可以通过互联网络（ATM-IP-ATM 或 IP-ATM-IP）又要通过混合堆叠（IP 通过 ATM）。经过投资数十亿美元和"大量"研究后，20 世纪 90 年代末最终放弃了这种努力。B-ISDN 从未实现商业部署。TCP/IP 协议赢得了第二次主要战争并继续从数据向语音及视频扩大其服务。

广域网服务：X.25 和帧中继

如果 TCP/IP 是数据通信阵营提出的数据服务解决方案，那么 X.25 就是电信阵营提出的第一个数据服务解决方案。X.25 有 3 层协议并且是面向连接的，但由于高的协议处理开销而使它的速度很慢。因此，它被重新设计压缩到两层协议栈的帧中继，当然仍然是面向连接的。过渡到帧中继是渐进的几乎不被人注意到。如今，仍有使用 X.25 或帧中继的金融系统，但大多数企业客户已经切换到 DSL 的 IP 了。

一个有趣的现象是，从电信阵营中推出的几乎所有数据服务都是面向连接的，并最后以失败或者被替换而告终。同样的事情可能发生在无线数据服务，包括 GSM/GPRS 和 3G 电路交换语音服务和分组交换，都是面向连接的数据服务。另一方面，由通信阵营推广的 WiMAX 和一些电信参与者不明确区分数据和语音，将自身定位为一个纯粹的第 2 层技术来承载 IP 及以上数据。如果历史会重演，那么最终的结果也应该是不言而喻的。

局域网技术：令牌环、令牌总线、FDDI、DQDB 和 ATM

类似于 IP，自 20 世纪 80 年代初以来以太网已经成为长期的赢家。它之所以会赢主要是因为它仍然保持简单，并经过了许多代的演变。20 世纪 80 年代它的第一个竞争对手是令牌环和令牌总线，由于循环令牌传递的性质，所以连接到环或总线上的站点具有有极大的传输延迟。但这种优势并没有为它们赢得市场份额，因为接口卡和集中器中硬件的复杂性较高。理论上无限的延迟并没有损害以太网，因为实际中的延迟是可以接受的。第二个竞争对手是 20 世纪 90 年代初的光纤分布式双接口（FDDI）和双头列双总线（DQDB），它们运行在 100Mbps，与当时 10 Mbps 以太网相比，能够提供服务质量即具有有限的延迟。类似于令牌环网，FDDI 增强了令牌传递协议，而 DQDB 运行一种复杂的机制——"在上行微时隙中请求下行数据时隙"。作为回应，以太网演变成 10/100 Mbps 的版本并再次以其硬件的简洁性赢得了市场垄断。

在 20 世纪 90 年代中期的第三个竞争对手是 ATM，它是源自 B-ISDN 巨大努力的一个分支。在 20 世纪 90 年代，ATM 旨在跨越不仅最后一公里的接口，而且还包括广域网（骨干网）和局域网。ATM 局域网提供了一个令人印象深刻的吉位/秒的容量，以及从局域网到广域网的全面综合。它的失败是因为 B-ISDN 保护伞被放弃了并且以太网演变为了 10/100/1000 Mbps 版本。第 3 章个血介绍了以太网的演变。

进一步阅读

IETF

几乎所有 IETF 相关的文档都是在线可访问的。没有特定的书籍是单独介绍 IETF 或 RFC 的。因此建议读者参考 IETF 在 www.ietf.org 的官方网站。

开源的发展

第一项是第一个开源项目。接下来的两个是有关开源的著名文章以及由之而来的书籍。第四个是有关开源发展的概述。

- R. Stallman, The GNU project, http://www.gnu.org
- E. S. Raymond, "The Cathedral and the Bazaar," May 1997, http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar
- E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly & Associates, Jan. 2001.
- M. W. Wu and Y. D. Lin, "Open Source Software Development: An Overview," *IEEE Computer*, June 2001.

附录 B

Computer Networks: An Open Source Approach

Linux 内核概述

在教授操作系统设计的时候，我们发现学生很难理解 Linux 内核源代码。学生可能不熟悉 Linux 用户和管理员的环境，他们也没有足够的经验来理解复杂的源代码。关于 Linux 下用户和管理员的环境，有大量且易于阅读的参考资料可供使用。对于开发者来说，有几个很好的参考资料，但它们对学习计算机网络课程的学生来说太厚了而不能快速阅读学习。学习跟踪复杂的源代码是另一个需要克服的障碍。因此，本附录提供了源代码，而 Linux 的开发工具和实用工具将分别在附录 C 和附录 D 中介绍。

本附录为跟踪 Linux 内核提供了一个简单的指南。同样的技术可用于跟踪 Linux 的应用程序。我们首先回顾了 Linux 源代码树的重点是网络子系统，这是本附录的重点。其次介绍几个有用的跟踪 Linux 源代码的工具。B.1 节中回顾内核源代码树，在默认名为/usr/src/Linux 的目录下，将 20 个目录分成 7 类，描述这些类包括的内容，并列出它们重要的例子模块。

B.2 节在表中总结了第 3 章、第 4 章和第 5 章中介绍过的开源实现。这将指导读者将重点放在网络互联上并缩小跟踪的程序函数。需要注意的是，第 6 章中的开源实现是在 Linux 的用户空间而非 Linux 内核中。总结表不包括第 7 章至第 8 章讨论的高级 QoS 和安全性的开源实现功能，虽然其中一些主是在 Linux 内核中实现的。

为了跟踪复杂的源代码，有效的工具是必不可少的。B.3 节介绍了几种流行的追踪工具并亲身经历一个样本例子的开源实现，第 4 章中介绍的 IP 重组。利用工具 LXR（Linux 的交叉引用）。读者可以采用相同的技法，打开本书中介绍的所有其他的开源实现。

发布和版本

Linux 是由 Linus Torvalds 在 1991 年编写的，当时他还是芬兰赫尔辛基大学的一名研究生。开发是在针对使用 80386 处理器的 PC 的 Minix 上完成的。但是，内核本身并没有系统软件，如 shell、编译器、或一文本编辑器等而不能工作。因此，Linux 0.99 是利用 GNU 通用公共许可证于 1992 年 12 月发布的。Torvalds 后来于 1994 年将内核的维护交给了 Alan Cox。

有许多 Linux 发行版，如红帽、SuSE、Debian、Fedora、CentOS 和 Ubuntu。但是，具体安装哪个发行版并不重要，因为它们共享同一个 Linux 内核。它们之间的区别在于它们的附加组件。可以选择带有图形化安装程序的发行版，便于服务器配置工具、更高安全性和良好的在线支持。

Linux 内核也有许多版本。每个版本都表示成 x.y.z，其中 x 是主版本号，y 是次要版本号，z 是发布号。在 2.6.8 版本之后，可能增加了第 4 个数字，用来表明一个较不足道的简单版本号。作为惯例，Linux 内核使用奇数的次要版本号码来表示开发版本，甚至次要版本号码表示稳定版本。截至 2009 年 6 月，最新版本是 v2.6.30。

B.1 内核源码树

Linux 2.6.30 的源代码由以下 20 个一级目录组成：Documentation、arch、block、crypto、drivers、firmware、fs、include、init、ipc、kernel、lib、mm、net、samples、scripts、security、sound、usr、virt。每个目录都包括用于特殊目的的文件。例如，在 Documentation/下的文件用来说明设计概念和 Linux 内核的实现细节。由于 Linux 高度演变的本性，目录的名字和位置可能会发生变化。例如，固件装载文件自版本 2.6.27 后就从 drivers/目录移取到了 firmware/目录。或者为新的框架创建了新的目录，例如，版本 2.5.5 建议的声音体系结构及其一级目录 sound/，或者由新功能驱动，虚拟化平台的支持导致 2.6.25 中的 virt/目录。

从高层看，我们在然能将这些目录分成 7 类，如表 B-1 中所示。这 7 类为创建、特定体系结构、内核核心、文件系统、网络、驱动程序和帮助程序。本节其余部分介绍每一类以便提供 Linux 内核的概述。

- 创建　在该类中的文件有助于完成和内核相关系统的制作。两个目录属于该类：scripts/和 usr/目录。scripts/目录包含命令行脚本和用于构建内核的 C 源代码。例如，当你在内核源代码的

表 B-1 Linux 内核源代码的总结

类 别	目 录	描 述
联通	usr/scripts/	帮助编译内核
特定体系结构	arch/x86/	特定体系结构的源代码和头文件
内核核心	init/kernel/ include/ lib/ block/ ipc/ mm/ security/ crypto/	内核中使用的核心函数和框架
文件系统	fs/	与系统相关文件的源代码
网络	net/	与网络相关的源代码
驱动程序	drivers/ sound/ firmware/	设备驱动程序
帮助程序	Documentation/ samples/	帮助你内核开发的文档和实例代码

顶级目录下键入 "make menuconfig" 时，它实际上执行在 scripts/kconfig/Makefile 中定义的步骤。然后在 usr/ 目录下的源代码可以用来构建一个 cpio 归档文件[1]的初始化 ramdisk，initrd。这可以在装载实际文件系统之前由内核在启动阶段执行[2]。

- **特定体系结构** 依赖于平台的代码放在 arch/ 目录下。为了减少移植工作，Linux 内核的设计将依赖的、特定体系结构的函数，如内存复制（memcpy）与一般例程分开。在早期版本中，特定体系结构的头文件，即 *.h 文件，位于子目录 include/asm-<arch>/ 下。自从 2.6.23 发布以来，逐步将这些文件转移到 arch/ 目录下。例如，x86 PC 体系结构的特定代码放在 arch/x86/ 子目录下。目前所有特定体系结构的头文件都放在 arch/ 目录下。

  Linux 还计划支持基于内核的、硬件辅助的虚拟化。相关代码位于 virt/ 目录下。目前，只有一个实用 Intel VT-x 扩展的模块可用。
- **内核核心** 这个类别包含提供内核核心函数的代码，它包括内核启动程序和管理例程。具体地说，init/main.c 调用许多初始化函数，启动 ramdisk，执行用户空间系统初始化程序，然后开始调度。初始化、调度、同步和进程管理函数的执行实际上是在 kernel/ 目录下。而内存管理例程是在 mm/ 目录下，进程间通信（IPC）的函数（某些协同存管理）实际上是在 ipc/ 目录下。

  该类别还包括内核空间共享函数，如字符串比较函数（strcmp），在 lib/ 目录下执行的。加密应用程序的程序接口（API）被组织到一级目录 crypto/ 下。它们的头文件（*.h）以及其他常见的头文件，如 TCP 头部，被放在内核模块共享，位于 include/ 目录下。

  最后，由 Linux 定义的通用框架，也属于这一类别。这些包括块设备接口，位于 block/ 目录下，以及安全框架及其实施，例如，安全增强 Linux（SELinux），位于 security/ 目录下。虽然声音体系结构称为高级 Linux 声音体系结构（ALSA），但是 Linux 中的多数体系结构，但我们喜欢把这样的文件放在驱动程序类别下。这是因为在 sound/ 目录下的大多数文件其实是设备驱动程序。
- **文件系统** Linux 在 fs/ 目录下支持几十个文件系统的实现。所有文件系统的核心，称为虚拟文件系统（VFS），是一个抽象层，提供文件系统与用户空间之间的接口。简而言之，新的文件系统遵守 VFS，将调用 register_filesystem() 和 unregister_filesystem() 分别用于向内核注册和自身脱离内核。

  在这些文件系统中，目前最常见的一种可能是第三种扩展文件系统（ext3）。ext3 和它的后继 ext4 的源代码分别位于 fs/ext3 和 fs/ext4 子目录下。此外，著名的网络文件系统（NFS）的源代码放在 fs/nfs 子目录下。
- **网络** 在 net/ 目录下的网络体系结构和协议栈实现可能是内核开发中最活跃的部分。例如，自 2.6.29 发布以来，Linux 已经支持了 WiMAX，其源代码放在 net/WIMAX/ 子目录中。B.2 节将阐述 net/ 目录。

---

[1] 在 Linux 2.6 之前，initrd 是从文件系统的镜像产生的，即将文件系统的布局基于节块转换。为了确保转为工作，设置和管理的权限，这对于开发人员可能不方便。Linux 2.6 中添加了一个新的 initramfs 格式，有效地允许用户空间的归档文件 cpio 创建 initrd。这样，在有内核开发的用户都可以进行内核编译而不用担心管理员权。

[2] 如果内核不能识别实际的文件系统，即它存储在主要的磁盘上，那么内核就需要 initrd 取出实际的文件系统。

- **驱动程序** 该类别在三个一级目录 drivers/、sound/和 firmware/内包括设备驱动程序的内核源代码。除了声卡的驱动程序外，其余的各种驱动器都放在 drivers/目录下。由于存在一个上述提到的统一的 sound（声音）体系结构，即 ALSA，所以声卡驱动程序移到了一级目录 sound/下。firmware/目录包含从设备驱动程序中提取的固件镜像文件。这些固件镜像的许可证在 firmware/WHENCE[①]中。
- **帮助程序** 大量的文档留在 Documentation/目录下的内核源代码中，可以帮助内核新手成为大师级高手。KOKTO 文件可能是你应该首先阅读的，它教你如何成为一个 Linux 内核开发者。在 kernel-docs.txt 文件中列出了数以百计的以说明从内核开发到驱动程序开发的在线文档。如果你打算参加内核开发，CodingStyle、SubmittingDrivers、SubmittingPatches 和 development-process/subdirectory 下的文件很值得一读。在这个目录下，你可以找到一个特定驱动程序或子系统的设计文档，例如，与文件系统相关的文件位于 filesystems/子目录下。最后，若要知道 Documentation/目录的全貌，DO-INDEX 将是你的首选。

伴随着 2.6.24 的发布，出现了一级目录 samples/。顾名思义，示例代码都放在这个目录下。例如，你可以通过 Kconfig 文件学到如何为“make menucontig”接口添加自定义的选项。目前，在该目录中只有几个例子，但我们相信在以后的版本中数量应该会增加。

最后，图 B-1 总结了内核源码树的 20 个一级目录，对应的 7 类别在上面给出的例子。

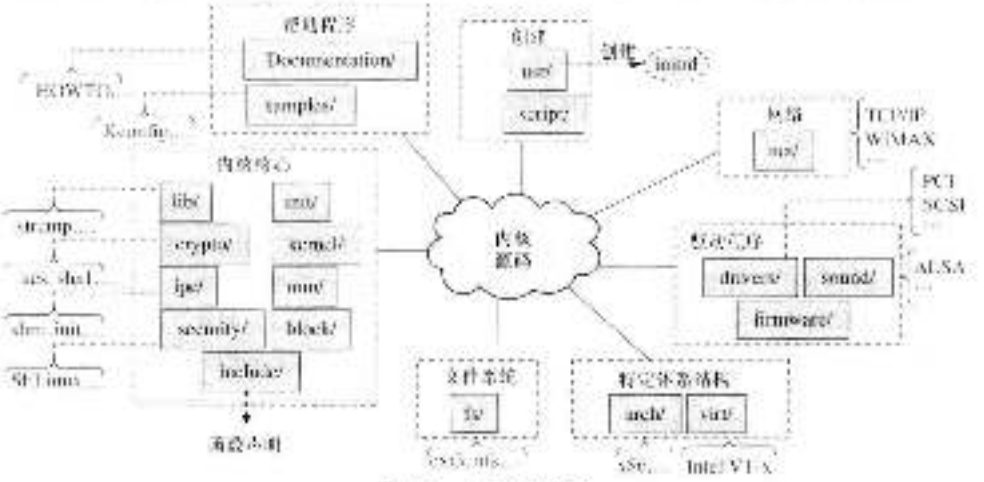


图 B-1 内核源码树

B.2 网络的源代码

在上述这些目录中，include/、net/和 drivers/与本书中介绍的协议最相关。include/目录中包含声明文件（*.h）。与内核和网络相关的声明分别是在 include/linux 和 include/net 目录下。例如，IP 头部，struct iphdr，是在 include/linux/ip.h 中声明的，而与 IP 相关的标志、常数和函数则是在 include/net/ip.h 中声明的。

另一方面，net/目录具有与网络相关的大部分代码。具体地说，常用的核心函数定义在 net/core 下的 .c 文件中，如 dev.c 和 skbuff.c。套接字接口的实现是在 net/socket.c 中完成的。TCP/IPv4 协议的代码位于 net/IPV4 下，如 ip_input.c、ip_output.c、tcp_conq.c、tcp_ipv4.c 和 tcp_output.c。IPv6 协议的代码放在 net/ipv6 下，如 ip6_input.c、ip6_output.c 和 ip6_tunnel.c。

最后，驱动程序、硬件设备和操作系统之间的接口，是在 drivers/目录下实现的。在这个目录下有很多子目录。以太网网络接口卡的驱动程序可以在 drivers/net 目录下找到，例如，3c501.c、3c501.h 作为 3Com 的 3c501 以太网驱动程序。在第 3 章中讨论过的 PPP 协议的代码也是放在这个目录

① 固件是一种机器代码或二进制配置，目的是用以优化硬件的功能。它可保存为镜像文件，即固件镜像文件，并且既可以与驱动程序一起编译，也可以在运行期间装载。而且是由厂商提供，因此未需要来自厂商的许可证。

表 B-2 与网络相关的目录和文件的总结

层次	主题	目录	文件	函数	描述
数据链路层	接收帧	net/core/	dev.	enet_rx_action()->netif_receive_skb()	一旦接收到 NET_RX_SOFTIRQ 中断，入核就调用 net_rx_action()，它依次调用 netif_receive_skb() 处理帧
数据链路层	发送帧	net/core/	dev.	enet_rx_action()->dev_queue_xmit()	一旦接收到 NET_TX_SOFTIRQ 中断，内核调用 net_rx_action()，它依次调用 dev_queue_xmit() 处理帧
数据链路层	Netcard 网卡驱动程序	drivers/net/	3c501.c, etc.	el_interrupt()、el_open()、el_close() 等	网络接口驱动程序，包括中断处理程序
数据链路层	外出数据	drivers/net/	ppp_generic.c	ppp_start_xmit()、ppp_send_frame()、start_xmit()	当内核调用 ppp_start_xmit() 时，PPP 守护程序调用 ppp_write
数据链路层	外出数据	drivers/net/	ppp_generic.c	ppp_start_xmit()、ppp_input()、ppp_receive_frame()、netif_rx()	ppp_send_receive() 取出 ppp->dev_queue，通过 netif_rx() 或 skb_queue_tail() 接收帧
数据链路层	受接	net/bridge/	br_fdb.c	__br_fdb_get()、fdb_insert()	自学习网桥，MAC 表查询
数据链路层	受接	net/bridge/	br_stp_bpdu.c	br_stp_rcv()、br_received_config_bpdu()、br_received_config_information()、br_configuration_update()	生成树协议
网络层	分组转发	net/ipv4/	route.c	ip_queue_xmit()、__ip_route_output_key()、ip_route_output_slow()、fib_lookup()、ip_rcv_finish()、ip_route_input()、ip_route_input_slow()	根据路由缓存转发分组，如果缓存没有命中，就要根据路由表进行转发
网络层	IPv4 校验和	include/asm/386/	checksum.h	ip_fast_csum()	使用依赖机器的汇编语言编写的代码加速校验和计算

（续）

层　次	主　题	目　录	文　件	函　数	描　述
网络层	IPv4 分段	net/ipv4/	ip_output.c、ip_input.c、ip_fragment.c	ip_fragment()、ip_local_deliver()、ip_defrag()、ip_find()、ipqhashfn()、inst_frag_find()、ipc_frag_create()	IP 分组分段和重组过程，使用散列标识一个分组的各段
网络层	NAT	net/ipv4/ net/fib/	nf_conntrack_core.c、nf_nat_standalone.c、nf_nat_ftp.crd_nat_proto_icmp.c ip_nat_helper.c	nf_conntrack_in()、resolve_normal_ct()、nf_conntrack_find_get()、nf_nat_in()、nf_nat_out()、nf_nat_local_fn()、nf_nat_fn()、nf_nat_ftp()、nf_nat_mangle_tcp_packet()、mangle_contents()、adjust_tcp_sequence()、icmp_manip_pkt()	分组过滤后和发送到输出接口前执行源 NAT。在分组过滤前对来自网卡或上层协议的分组执行源 NAT。NAT ALG（帮助函数）用于 FTP 和 ICMP
网络层	IPv6	net/ipv6/	ip6_fib.c	fib6_lookup()、fib6_lookup_1()、ipv6_prefix_equal()	查找 IPv6 路由表（FIB），它存储在一个二进制 radix 树中
网络层	ARP	net/ipv4/	arp.c	arp_send()、arp_rcv()、arp_process()	ARP 协议的实现，包括发送、接收以及处理 ARP 分组
网络层	DHCP	net/ipv4/	ipconfig.c	ic_bootp_send_if()、ic_dhcp_init_options()、ic_bootp_recv()、ic_do_bootp_ext()	DHCP/BOOTP/RARP 协议的实现，我们跟踪 DHCP 消息的发送和接收过程
网络层	ICMP	net/ipv4/	icmp.c	icmp_send()、icmp_unreach()、icmp_redirect()、icmp_echo()、icmp_timestamp、icmp_address()、icmp_address_reply()、icmp_discard()、icmp_rcv()	ICMPv4 的实现，不同类型的 ICMP 消息由相应的函数处理
网络层	ICMPv6	net/ipv6/	icmp.c ndisc.c	icmpv6_send()、icmpv6_rcv()、icmpv6_echo_reply()、icmpv6_notify()、ndisc_rcv()、ndisc_router_discovery()	ICMPv6 的实现，包括 5 种新的 ICMPv6 消息，即路由器请求、路由器广告、邻居请求、邻居广告和路由重定向消息

传输层	UDP 和 TCP 校验	net/ipv4/	tcp_ipv4.c	tcp_v4_send_check(), csum_partial(), csum_tcpudp_magic()	TCP/UDP 分段（包括伪头部）的校验和计算
传输层	TCP 滑动窗口流量控制	net/ipv4/	tcp_output.c	tcp_send_test(), tcp_packets_in_flight(), tcp_nagle_check()	在将 TCP 分段发送单元之前检查下面 3 个条件：1）未经确认的分段小于 cwnd；2）发送分段数加 1 后小于 rcwnd；3）执行 Nagle 测试
传输层	TCP 慢启动和拥塞避免	net/ipv4/	tcp_cong.c	tcp_slow_start(), tcp_cong_avoid(), tcp_cong_avoid_ai()	TCP 慢启动和拥塞避免
传输层	TCP 重传定时器	net/ipv4/	tcp_input.c	tcp_ack_update_rtt(), tcp_rtt_estimator(), tcp_set_rto()	测量 RTT，计算平滑的 RTT，并更新重传超时 RTO
传输层	TCP 坚持定时器和保活定时器	net/ipv4/	tcp_timer.c	tcp_probe_timer(), tcp_send_probe0(), tcp_keepalive(), tcp_keepopen_proc()	用于管理坚持定时器（探测定时器）和保活定时器的函数
传输层	TCP FACK 实现	net/ipv4/	tcp_output.c	tcp_adjust_fackets_out(), tcp_adjust_pcount(), tcp_init_mtuprobe_queue()	使用 FACK 算法计算传输中的分组
传输层	套接字读/写入	net/	socket.c	sys_socketcall(), sys_socket(), sock_create(), inet_create(), sock_read(), sock_write()	解析用户空间的套接字接口主向本内核空间中实现
传输层	套接字过滤器	net/	socket.c	SYSCALL_DEFINE5 (setsockopt, ...) sock_setsockopt()	包含利用过滤器（BPF）的实现

下，例如，pop_generic.c。

表 B-2 总结了第 3、4、5 章中跟踪过的目录、文件和开源实现函数。当跟踪它们时，你会首先发现用于特定源代码的文件，然后跟踪在此表中列出的主函数，以便了解程序执行的主要流程。B.3 节介绍有效跟踪源代码的工具。

B.3 源代码跟踪工具

有多种浏览 Linux 源代码的方法，以便搜索变量/函数的声明或变量/函数的用法（参考），最容易的方法是在 Web 网站上浏览源代码。例如，LXR（http://lxr.linux.no/），Linux 的交叉引用，提供了基于 Web 的 Linux 源代码索引、交叉引用和导航。LXR 的搜索功能允许你搜索一个变量或函数的声明和引用位置。它还提供全文检索。

另一个黑客常用的工具是 cscope。cscope 是一个交互式的、面向屏幕的工具，它允许用户定位 C、lex 或 yacc 源文件中的指定代码元素。它在源文件中使用符号交叉引用定位函数、函数调用、宏、变量和预处理器符号。作为一个例子，cscope 可以分两步来跟踪 Linux 源代码。首先，在源代码的目录下，可以通过使用“find -name** -[chly]*-print -scrt > cscope.files”得到这个目录和子目录下的一个称为 cscope.files 的文件的文件名列表。然后，可以通过“cscope -b -q -k”生成符号交叉引用数据库，默认为 cscope.out。现在，可以使用 cscope -d 命令搜索变量或函数。在 C.3.1 节中将更详细地介绍 cscope。

最后，有几个使用起来相当得心应手的源代码文档生成器工具。例如，Doxygen（http://www.stack.nl/~dimitri/doxygen/），是在 GNU 通用公共许可证下发布的免费软件程序，可以交叉引用文档和代码以便生成各种格式的文件，包括 HTML、Latex、RTF（MS-Word）、PostScript、超链接 PDF、压缩 HTML 和 UNIX 帮助手册网页。它也可以从没有归档的源文件中提取代码结构。有很多方法实现代码结构的可视化，包括依赖图、继承图和协作图。

例子：跟踪 IPv4 分段的重组

让我们利用第 4 章中的图 4-19 作为使用 LXRWeb 站点跟踪源代码的例子。图 4-19 中显示了 IP 分组分段重组过程的调用图。为了便于解释，我们将它重绘成图 B-2。

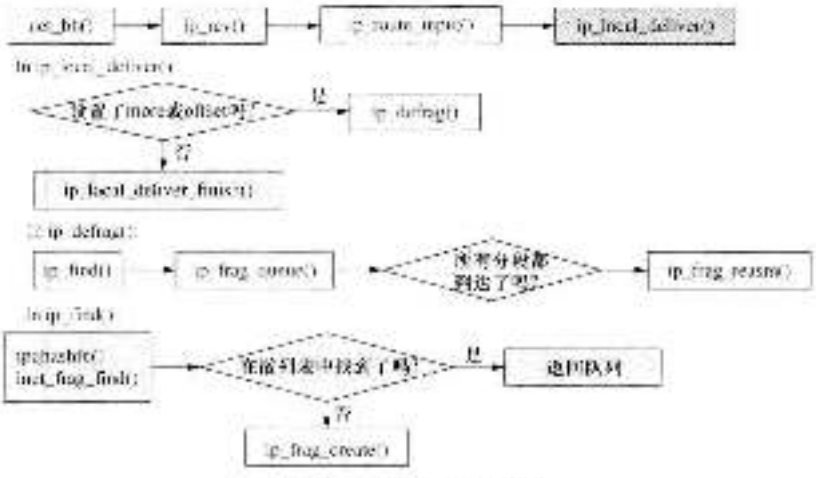


图 B-2 重组过程的调用图

接下来，让我们从定位 ip_local_deliver 函数开始。为了能够找到该函数，我们使用 LXR 的 Web 站点搜索工具栏并输入 ip_local_deliver 来搜索函数，如图 B-3 所示。

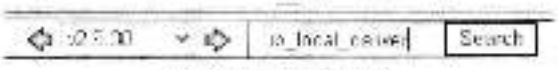


图 B-3 LXR 搜索栏

LXR 返回一个网页，指示了两种信息：函数在哪里实现的和函数在哪里声明的。在我们的例子中，如图 B-4 所示，ip_local_deliver()的代码是从 net/ipv4/ip_input.c 文件中的第 257 行开始的，而对 ip_local_deliver()的声明是在 include/net/ip.h 中的第 98 行。为了过渡 ip_local_deliver()的源代码，我们可以点击 net/ipv4/ip_input.c 的超链接。

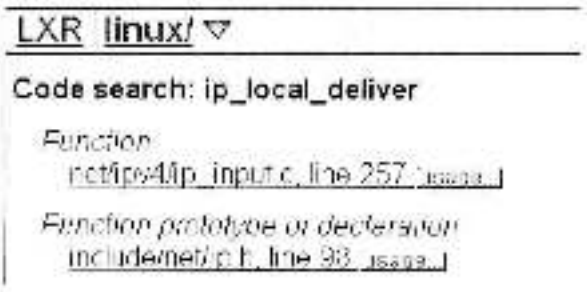


图 B-4 LXR 的搜索结果

除了定位代码和函数的声明外，我们也可以检查此函数的用法，但就是说，通过点击 [usage] 链接，知道函数在哪里被引用（调用）。例如，当点击声明的用法链接时，LXR 返回如图 B-5 所示的参考信息。从这个页面中我们可以看到，ip_local_deliver()被在两个文件 net/ipv4/ipmr.c 和 net/ipv4/route.c 中定义的函数引用了 6 次。

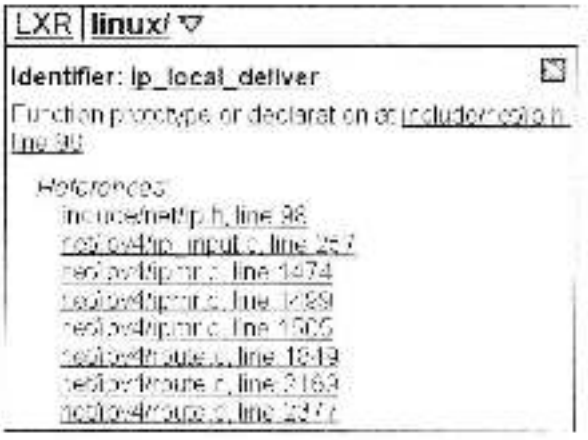


图 B-5 ip_local_deliver()的用法

如果我们点击 net/ipv4/ip_input.c，LXR 将返回在 net/ipv4/ip_input.c 中 ip_local_deliver()的代码实现，如图 B-6 所示。通过引用图 4-19 的调用图，可以清楚地了解是否有偏移值及是否设置了 more 位或 offset 位，ip_defrag()将被调用。因此，通过点击超链接（ip_defrag）让我们继续跟踪 ip_defrag()代码。

```
252 int ip_local_deliver(struct sk_buff *skb)
253 {
254         /*
255          *      Reassemble IP fragments.
256          */
257
258         if (ip_hdr(skb)->frag_off & hnone(IP_MF | IP_OFFSET)) {
259                 if (ip_defrag(skb, IP_DEFRAG_LOCAL_DELIVER))
260                         return 0;
261         }
262
263         return NF_HOOK(NF_INET, NF_INET_LOCAL_IN, skb, skb->dev, NULL,
264                        ip_local_deliver_finish);
265 }
```

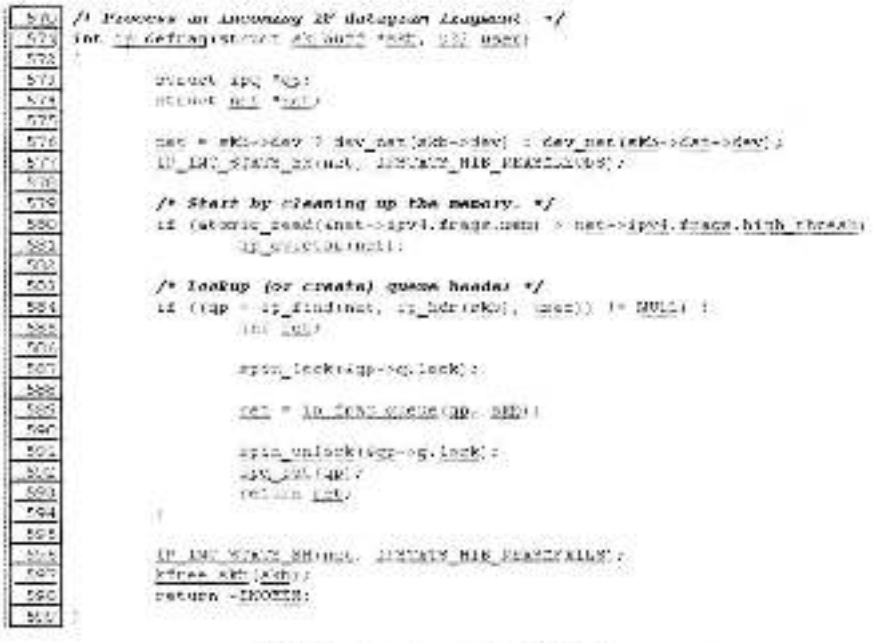
图 B-6 ip_local_deliver()的源代码

LXR将显示ip_defrag的搜索结果，如图B-7所示。该网页会告诉我们，它是在/net/ipv4/ip_fragment.c中实现的。让我们通过点击链接继续跟踪。



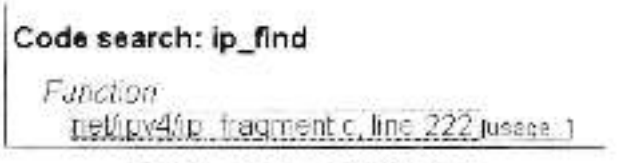
图B-7 ip_defrag()的搜索结果

图B-8显示了ip_defrag()的代码。再次参阅调用图，经过一些琐碎的工作后，ip_defrag()首先调用ip_find()查找或为该分组的分段创建队列头部。然后，它调用ip_frag_queue()处理分段。如果所有的分段都收到了，那么ip_frag_queue()将调用ip_frag_reasm()重组分组。



图B-8 ip_defrag()的源代码

通过点击ip_find的超链接，我们将能够找到函数所在位置并获得其源代码，分别如图B-9和图B-10所示。而且，参考调用图，ip_find首先调用ipqhashfn()获得散列值，并使用它通过调用inet_frag_find()找到队列头部。



图B-9 ip_find()的搜索结果

在inet_frag_find()中，如果没有找到队列的头部，它将通过调用inet_frag_create()创建一个。inet_frag_find()的源代码如图B-11所示。



图 B-10 ip_find()的源代码

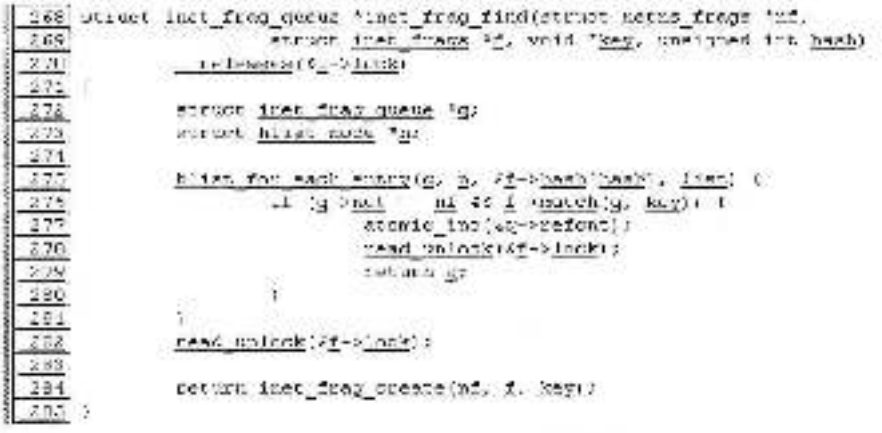


图 B-11 ip_frag_find()的源代码

进一步阅读

相关书籍

以下两本是由 O' Reilly 出版的入门书，也可以作为许多内核开发人员的参考书籍。第一本介绍 Linux 内核基础，如内存管理、进程管理、调度例程和文件系统。第二本书详细介绍 Linux 设备驱动程序的开发。

- M. Cesati and D. P. Bovet, *Understanding the Linux Kernel*, 3rd edition, O' Reilly Media, 2005.
- J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd edition, O'Reilly Media, 2005.

在线链接

这里，我们列出了 3 个与附录内容相关的 Web 站点。这些经典的网站在未来几年内可能一直存在下去和蓬勃发展。

1. Linux kernel archives, http://kernel.org/
2. Linux foundation, http://www.linuxfoundation.org/
3. Linux cross reference (LXR), http://lxr.linux.no/

附录 C

Computer Networks: An Open Source Approach

开发工具

在 B.3 节中，我们介绍了几种源代码跟踪工具和利用 LXR（Linux 交叉参考）的应用例子。然而，跟踪仅是否向理解程序的第一步，还需要其他步骤来完成开发过程。本附录为 Linux 开发人员介绍了一套完整的综合开发工具。Linux 开发人员可以在 Linux 主机上编写程序，但要在一台非 Linux 的目标机上运行它，反之亦然。也有可能主机和目标机器都是基于 Linux 的。这里，我们重点在 Linux 主机上，即开发环境，而不管目标机的平台是什么。我们将介绍在开发过程中从编程、调试和维护一直到轮廓分析和嵌入等各个阶段所使用的必要的和流行的工具。

C.1 节引导读者利用工具开始开发的旅程。良好的开始是使用一个强大的文本编辑器（如欢迎的视觉（vim）或 GNOME 编辑器（gedit））编写第一段代码，然后使用 GNU C 编译器（gcc）进行编译，并使用 make 工具进一步自动化某些重复性的编译步骤。

老的 80/20 规则仍然适用于编程，其中 80% 代码来自 20% 的工作，留下的 80% 工作去调试余下 20% 的程序。因此，就需要一些在 C.2 节中讨论过的调试工具，包括源代码级的调试器，GNU 调试器（gdb）；带有图形用户界面的数据显示调试器（ddd）和远程内核调试器，内核 GNU 调试器（kgdb）。因为软件组件之间的依赖性更为复杂以及贡献源变得更加分散，所以 C.3 节说明了开发人员如何使用 cscope 管理几十个源文件，并且合作开发人员应对版本控制系统达成一致，如全球信息跟踪器（Git），以避免开发混乱并且易于合作。

为了找到某个程序的瓶颈，C.4 节向开发人员介绍了如何利用分析工具，GNU Profiler（gprof）和 Kernel Profiler（kernprof）。C.5 节介绍了如何利用空间优化工具加快嵌入式系统的移植 busybox（轻量级的工具链），uClibc 和根文件系统的嵌入式镜像构建工具 buildroot。

对于每一个工具，我们介绍它的目的和功能，随后再用例子加以说明。最后，提供一些技巧来帮助你熟悉工具。本附录并非是一个完整的用户指南，但应该可以作为一个学习的入门。

C.1 编程

本节介绍了编程的重要工具，从用于程序编辑的 vim 和 gedit 到用于程序编译的 gcc 和 make。这里没有讨论编程语言的本质。

C.1.1 文本编辑器：vim 和gedit

不论选择什么编程语言，你都需要有一个编辑器，它是一个用于创建和修改文本文件的程序。它在程序员工作时起着至关重要的作用，因为笨拙的文字编辑器会浪费时间，而有效的工具则能轻松地解决问题并为程序员留下更多的思考时间。

什么是vim和gedit？

在许多文本编辑器如 pico、joe 和 emacs 中，Visual Improved（vim），vi 的一个改进版本，是目前最流行的文字编辑器之一。它在易用性和功能之间进行了平衡，比 emacs 使用起来更友好，比 pico 具有更丰富的功能。vim 有一个可扩展的句法字典，其中对于它所识别的文件（包括 C 代码和 HTML）会用不同的颜色突出加亮句法显示。高级用户使用 vim 来编译自己的代码，编写宏，浏览文件，甚至可以编写游戏，如 TicTacToe。

作为一个命令行编辑器，vim 被管理员广泛应用。然而，作为桌面应用程序工具，它使用起来有点复杂。内置的 GUI 编辑器，GNOME 编辑器（gedit）常用于 Linux 的桌面环境中。gedit 允许用户使用一个选项卡（tab bar）编辑多个文件，像 vim 一样突出加亮句法，拼写检查文本，以及打印文件。

如何使用vim和gedit

在开始使用 vim 之前，用户应该知道 vim 工作在两个阶段（模式）而不是像 pico 或其他普通的文本

编辑器那样只有一个阶段（或模式）。尝试：启动 vim（输入 vim 编辑一个新文件或者 vim filename 打开一个现有的文件），并编辑一分钟。如果你不输入任何特殊字符，那么在屏幕上你什么都看不到。当你按箭头键尝试移动时你会发现光标不动。更糟的是，你无法找到退出的方法。这些最初的障碍阻挠了不少新手的使用。然而，当你知道什么时候插入文本以及何时发出命令时，这些问题就迎刃而解了。

在正常模式（命令模式），字符都被视为命令。这意味着它们会触发特殊的动作，例如，分别按下 h、j、k 和 l 向左、上、下和右移动光标。然而，在插入模式，字符只是以文本插入。大多数命令都可以合并成一个更复杂的操作。

```
[#1] commands [#2] target;
```

括号内的任何内容都是可选的。#1 是一个可选的数字，例如，3，是指定命令要进行 3 次；command 是任何有效的 vim 操作，例如，y 是复制文本；#2 是另一个可选的数字，类似于#1，指定被 command 影响的目标数量（或范围）；而 t 是你将 command 应用的文本，例如，G 为文件的结束。虽然大多数的命令显示在主屏幕上，但有些冒号命令（以冒号开始的命令）显示在屏幕的最底部。处理这些冒号命令时，需要输入一个冒号，将光标移动到屏幕的最后一行，然后发出命令字符串，按下 <Enter>（回车键）来终止它。例如，:wq 将保存当前文件并退出。vim 文本编辑器的全部操作模式在图 C-1 中显示。重要的编辑命令在表 C-1 中列出。

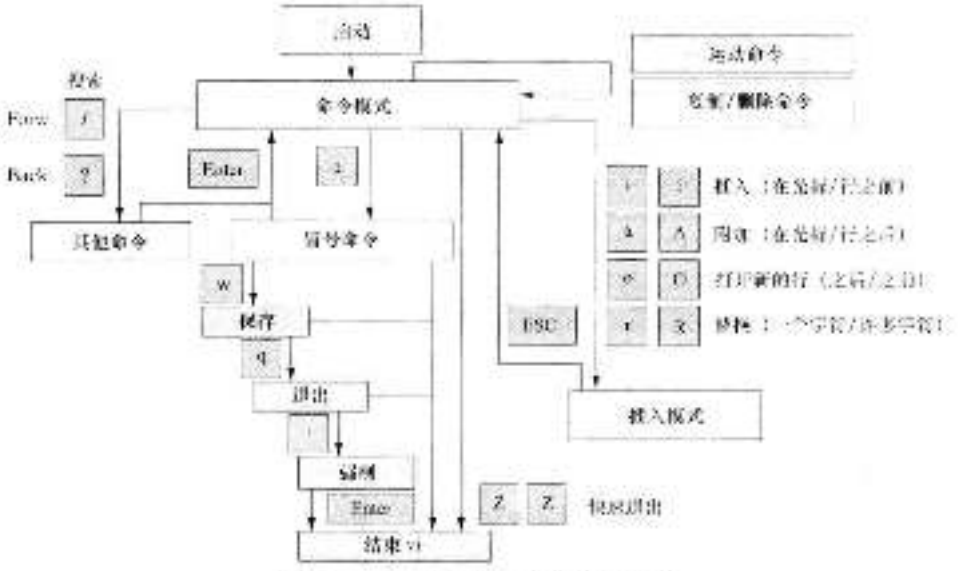


图 C-1 vim 文本编辑器的操作模式

表 C-1 光标移动和文本编辑的重要命令

命令模式		效果	命令模式		效果
运动	h,j,k,l	左、下、上、右	运动	fc,Fc	前进、后退到字符 c
	w,W	前进到下一个字，空白分开字		L,M,h	屏幕的上、中、下部
	e,E	前进到字结束，空白分开字	复制	Yy	复制当前行
	b,B	后退到字的开始，空白分开字		:y	复制当前行
	(,)	句子后退、前进		Y	复制直到行结束
	{,}	段落后退、前进	删除	Dd	删除当前行
	0,3	开始，行结束		:d	删除当前行
	1G,G	开始，文件结束		D	删除直到行结束
	nG 或者 :n	n 行			

gedit 的使用比 vim 简单得多。一个正在编辑的文件会显示在编辑区域中，这里你可以用鼠标定位并突出加亮显示文本。选项卡列出所有被编辑的文件。如果一个文件是修改过的就标记一个星号也不保存。选项卡提供了最简单的方式来创建，打开、保存和打印文件。gedit 的一张屏幕截图如图 C-2 所示。

技巧

- vim 工作在两种模式：插入模式和命令模式。如果你对插入模式命令感到困惑，你可以按 ESC 键回到命令模式。

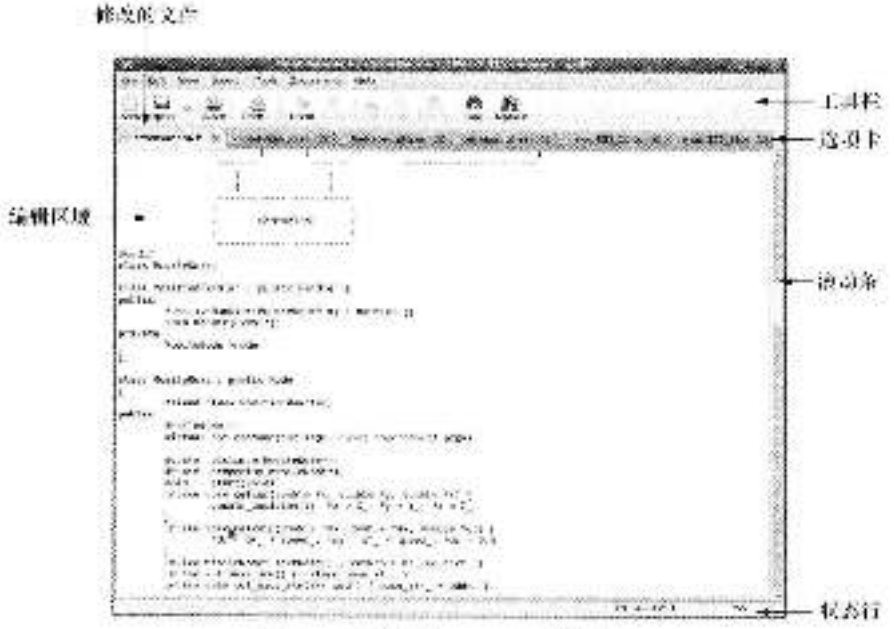


图 C-2 截取屏幕：gedit 的主窗口

C.1.2 编译器：gcc

手边有一个文本编辑器，人们就可以开始编写程序了。那么你还需要一个编译器，将高级语言编写的源代码转换为二进制目标代码。由于经常需要集成已经编译过的现有例程，第二阶段处理使用一种称为连接器的工具，通过它将编译过的代码与已有的例程连接起来以便创建最终的可执行应用程序。gcc 编译器的多阶段过程如图 C-3 所示。

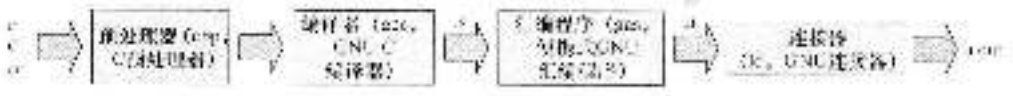


图 C-3 gcc 的工作流程图[1]

什么是gcc

默认支持 ANSI C[2] 的 GNU C 编译器（gcc）是一个存在于大多数 UNIX 系统上的非常著名的 C 语言编译器。它主要由 Richard Stallman（编写），他成立了一个慈善的、免费软件基金会（FSF），以筹集资金从事 GNU 项目工作。并在其他倡导者的努力下，gcc 已经集成了多个编译器（C/C++，Fortran，Java），现在称为 GCC 编译器集合。

[1] 预处理器的它们正常直接送入编译器中。

[2] ANSI C 比传统的 C 在类型上要强一些，它可能更容易帮助你在编码调试及早发现一些错误。

如何使用gcc

假设你正在编写一个程序，并已经决定分成两个源文件。其中一个主文件称为main.c，另一个为sub.c。为了编译程序，你可能只需要输入：

```
gcc main.c sub.c
```

将默认地创建一个名为a.out的可执行程序。如果你愿意，你可以指定可执行文件的名称，例如，prog，通过以下命令：

```
gcc -o prog main.c sub.c
```

正如你可以看到的，它很简单。但是，这种方法可能非常低效，特别是如果你每次只更改一个源文件，并且当你不断地重新编译它时更是如此。相反，你应该如下编译程序：

```
gcc -c main.c
gcc -c sub.c
gcc -o prog main.o sub.o
```

前两行创建目标文件main.o和sub.o，第三行将对象连接成一个可执行的文件。如果你只更改了sub.c，那么只需要输入最后两行，就可以重新编译程序。

对于上面的例子，这一切似乎有点笨拙，但如果你有10个源文件而不是两个，那么后一种方法会为你节省很多时间。实际上，整个编译过程可以自动完成，我们在下一节看到。

技巧

使用gcc时，有两种常见的错误。一种错误指示源代码中的句法错误，另一种错误就是当连接对象文件时出现类解析的符号。gcc如下显示句法错误：

```
* sourcefile: In function `function_name':
* sourcefile:linum: error: error messages
```

上面语句的意思是，sourcefile中行linum附近可能有错误。值得注意的是，错误并不总是靠近行linum。例如，当错误是由于括号丢失或者有不必要的括号造成时，报告的linum可能会离实际的错误点很远。

未解析的连接符号的格式为：

```
* objectfile: In function `function_caller':
* sourcefile: undefined reference to `function_callee'
```

该语句告诉开发人员函数function_callee在连接时，不能解析。未解析的函数由文件sourcefile中的函数function_caller使用。为了解决这个问题，可以检查是否有一个包含function_callee的必要的对象文件或库是未连接的。

C.1.3 自动编译：make

虽然一次成功、无错的编译过程肯定是一大快事，但在程序开发期间不断重复的编译处理过程将是程序员的一件苦差事。一个可执行程序，可从数十或数百个.c文件中构建，要求所有的.c文件使用gcc编译为.o文件，然后再连接在一起，很可能还要与额外的库例程连接。这个过程是繁琐并且可能发生错误的。这就是为什么要使用make的原因。

什么是make

make为一个能够提供比较高级的方法来指定所需要的源文件，建立一个派生对象和步骤以重自动化构建过程的程序。make减少了出错的可能性，并简化了程序员的工作。需要注意的是，make为许多常用的动作（如将.c文件转变为.o）提供的认知规则或捷径。

如何使用make

make处理一个称为Makefile的文件。Makefile中的基本句法是：

```
target: dependencies
<command list>
```

它告诉make通过执行命令列表从dependencies中产生target。依赖性需要在产生当前目标之前解决，这为通过分治的方式构建一些大的应用程序带来了某些机会。

例子

在图C-4中的例子中，Makefile的内容首先声命令cat列出来。这个Makefile说明prog依

赖于两个文件 main.o 和 sub.o，而且除了常见的头文件 incl.h 外，它们还依赖于对应的源文件（main.c 和 sub.c）。通过执行命令 gcc，依赖性，main.o 和 sub.o，编译后自动连接到目标 prog。

技巧

- 在编写 Makefile 中，每条命令语句的开头都要放置一个 TAB 字符[1]。



```
$ cat Makefile
#Any line beginning with a "#" sign is a comment and will be
#ignored by the "make" command. To generate the executable
#programs, simply type "make".

prog: main.o sub.o
        gcc -o prog main.o sub.o
main.o: incl.h main.c
        gcc -c main.c
sub.o: incl.h sub.c
        gcc -c sub.c
$ ls
incl.h main.c Makefile prog sub.c
$ make
gcc -c sub.c
gcc -c sub.c
gcc -o prog main.o sub.o
$ ls
incl.h main.c main.o Makefile prog sub.c sub.o
```

Makefile的内容

← 编译前

运行

← 编译后

图 C-4 make 的一个例子

C.2 调试

当编写程序时，除了是太微不足道，否则必须尽最大努力找出和更正程序中的错误。这种找出错误的过程称为调试。使用的工具是调试器。一般来说，调试器的目的是让你调查程序运行时内部发生了什么或者程序崩溃时程序正在做什么。这里，我们介绍三种调试器：常用的，gdb；图形化的版本，ddd 和远程内核调试器，kgdb。

C.2.1 调试器：gdb

在 Linux/FreeBSD 中使用的传统调试器为 gdb。GNU 项目调试器。它工作在不同的语言环境中，但主要是针对 C 和 C++ 的开发人员。虽然 gdb 是一个命令行界面，但它也有几种图形界面，如 ddd。此外，gdb 也可以运行在串行端路上用于远程调试，如 kgdb。

什么是gdb

gdb 可以做的事情主要有 4 种，再加上支持这些的其他东西，可以帮助你当场发现错误：

1）启动程序，指定任何可能影响其行为的任何事情。

2）在特定条件下让程序停止。

3）当程序已经停止运行时，检查发生了什么状况。

4）调整程序，以便你能够测试更正错误后的效果。

如何使用gdb

可以阅读官方 gdb 手册来学习有关 gdb 的所有内容。然而，几条命令就足以开始使用调试器。在将可执行程序装入 gdb 前，目标程序（如 prog）应该带-g 标志进行编译，例如，gcc -g -o prog prog.c。

然后，使用命令 gdb prog 来启动 gdb。接下来你应该看到一个 gdb 提示符。然后使用命令 list 浏览源代码，默认情况下显示当前函数的前 10 行源代码，接下来调用 list 显示接下来的 10 行等。

当试图定位一个错误时，进入 gdb 后就可以再一次运行程序并确保重新产生错误。然后，你可以

[1] 这是一个历史读留问题，但很有人愿意去改变它。

backtrace（回溯）看到一个栈跟踪，这通常就可以找出问题出在哪里。现在你可以再次使用 list 确定问题的位置，使用 next 逐步执行，慎重利用命令符 break 设置断点并利用命令 print 打印变量。你应该能够定位错误，并最终 quit gdb。顺便说一句，gdb 具有一组信息页面并且也有内置的帮助，这些可以通过 help 命令来访问。

例子

在图 C.5 的例子中演示了一个常见的由于内存分配造成的编程故障。当程序首次在 gdb 中运行时，它导致一个分段故障。我们检查当前栈中的帧，发现它可能是一个在函数 Hello 中的错误。因此在这里设置一个断点，并再次运行程序。当到达断点时，gdb 会暂停。我们浏览源代码，并 step 执行。我们检查变量 str 后，找到错误；指针没有有效的内存地址。

```
$ gdb prog
GNU gdb (GDB) Fedora (7.0.1-35.6-17)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/book/C.2.1/prog ; done.
(gdb) run
Starting program: /home/book/C.2.1/prog

Program received signal SIGSEGV, Segmentation fault
0x0029b545 in memcpy () from /lib/libc.so.6
Missing separate debuginfos, use: debuginfo-install glibc-2.11.1-1.i686
(gdb) backtrace
#0 0x0029b545 in memcpy () from /lib/libc.so.6
#1 0x0000000 in ??()
(gdb) break Hello
Breakpoint 1 at 0x804841a; file sub.c, line 8
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/book/C.2.1/prog

Breakpoint 1, Hello () at sub.c:8
8           char*str = NULL;
(gdb) list
3
4     #include "inclib"
5
6     void Hello()
7     {
8           char*str = NULL;
9           strcpy(str, "hello worldn");
10          printf(str);
11    }
(gdb) next
9           strcpy(str, "hello worldn");
(gdb) print str
$1 = 0x0
(gdb) quit
A debugging session is active.

      Inferior 2 [process 24886] will be killed.

Quit anyway? (y or n) y
```

← 段错误

← 检查当前栈中的帧

← 设置断点

浏览当前源代码

← 单步程序

← 检查一个变量

图 C.5 一个利用 gdb 调试的例子

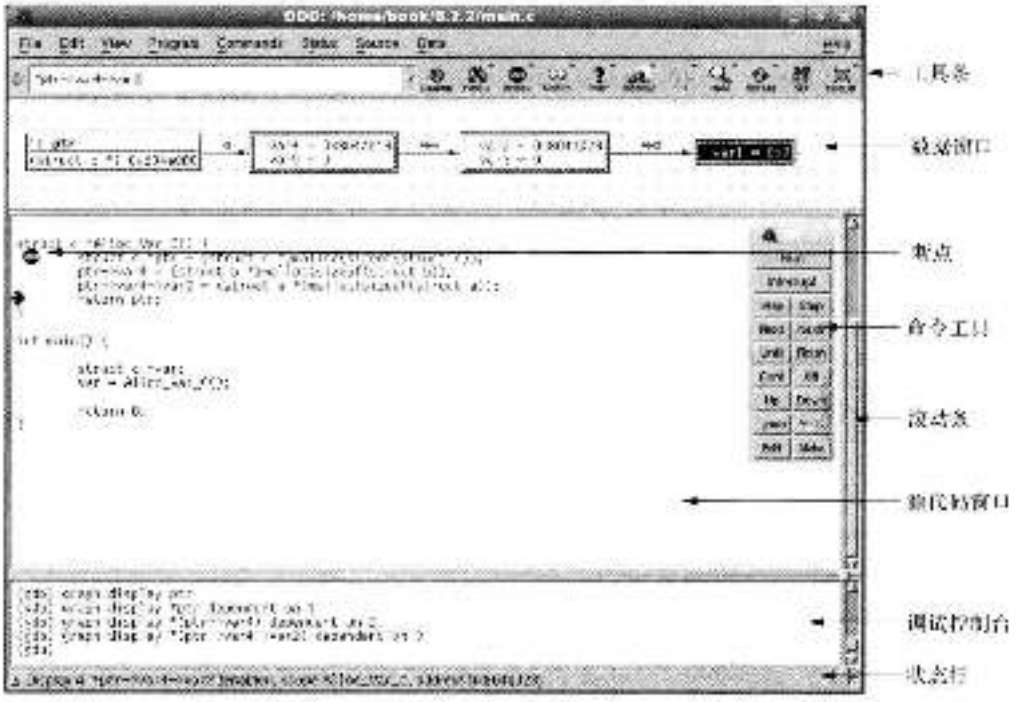
C.2.2 GUI 调试器：ddd

什么是ddd

由于gdb和许多其他工具是命令行调试器，使用起来就没有那么友好，所以数据显示调试器（Data Display Debugger，ddd）为所有这些调试器提供了一个方便的前台。除了具有gdb已有的功能外，ddd之所以很受欢迎，是因为它的交互式图形数据显示，它将数据结构都显示成图形。

如何使用ddd

为了使用ddd，还必须利用包括调试信息来编译代码。在UNIX中，这意味着必须在gcc编译命令中包括-g选项。如果你以前从来没有运行过ddd，你可能要在命令行提示符下输入"ddd -gdb"告诉ddd使用gdb调试器。你只需要这样做一次。然后，为了运行ddd你输入"ddd progg"，这里progg是程序的名字，就会弹出一个如图C-6所示的窗口。



图C-6 截图：ddd的主窗口

图C-6中一切的重心是源代码。当前执行的位置由一个箭头指示，断点显示为一个停止标记。你可以使用工具栏上的Lookup（查找）按钮或者使用"文件"（File）菜单下的Open Source按钮浏览代码。双击函数名，就可以找到它的定义。使用"命令"（Command）工具中的Undo和Redo按钮，你可以导航到前面和后面的位置——类似于Web浏览器。

在源代码窗口某条语句的左边空白区间中右击，可以设置和清除断点。为了单步调试程序或者继续执行，可以使用右边的弹动命令工具。命令行爱好者还可以在底部找到一个调试控制台。如果你还有其他的需要，尝试使用"帮助"（Help）菜单寻找详细说明。

将鼠标指针移动到一个活跃的变量上，会在一个小的弹出式屏幕中显示其值。更复杂的值快照可以在调试控制台中"输出"。为了永久地查看一个变量，使用Display按钮。这将创建一个永久性的数据窗口，显示变量名及其值。每次程序改变其状态时这些显示都会被更新。

为了访问一个变量值，必须将程序运行到变量实际上会活跃的状态下，也就是说，位于当前执行位置的范围内。通常情况下，在感兴趣的函数内设置一个断点，运行程序，然后显示函数的变量。

为了真正可视化数据结构（即数据以及相互间的关系），ddd允许你通过双击指针变量由现有的

显示器创建一个新的显示器。例如，如果你已经显示了一个类计变量列表，那么你可以取消对它的引用，并查看它所指向的值。每一个新的显示器会以支持列表和树的简单可视化的方式自动地展示。例如，如果一个元素已经拥有前驱，那么其后继将与这两个一致地显示。你可以通过直接拖放显示随意地手动移动元素。此外，ddd 界面可以左右滚动，设计结构，手动更改值，或者在程序运行时观察它们的改变。撤销/重做（Undo/Redo）功能甚至可以让你重新显示程序以前和以后的界面，因此你可以看到数据结构是如何演变的。

C.2.3 内核调试器：kgdb

什么是kgdb

kgdb 是一个用于 Linux 内核的源代码级的调试器，它提供了一种使用前面介绍过的调试器 gdb 来调试 Linux 内核的机制。kgdb 是一个内核补丁，一旦打过补丁后你就需要重新编译内核。它允许用户在开发主机上运行 gdb 以便连接到目标（通过串行 RS-232 线路）运行打过 kgdb 补丁的内核。然后内核开发人员就可以“进入”目标的内核中，设置断点，检查数据，以及其他人们所望的相关调试功能。事实上，这与人们使用 gdb 在用户空间程序上所做的非常类似。

由于 kgdb 是一个内核补丁，所以它在目标机器内核上增加了以下组件：

1）gdb 根——gdb 根是调试器的核心。它可以处理来自开发机器（或主机）上 gdb 的请求，控制目标机器上所有处理器的执行流。

2）修改故障处理程序——当发生意外错误时，内核将控制交给调试器。不包含 gdb 的内核在出现意外故障时产生错误（panics）。修改故障处理程序允许内核开发人员分析意外故障。

3）串行通信——该组件在内核中使用串行驱动程序，并在内核中提供到 gdb 桩的接口。它负责在串行线路上发送和接收数据。该组件也负责处理从 gdb 发送的控制断点请求。

如何使用kgdb

自 Linux 内核 2.6.26 以来，kgdb 已经集成到了主流的内核源代码树中。你所要做的就是在内核配置中激活 kgdb 选项，然后编译并安装打过 kgdb 补丁的内核。为了强制内核暂停引导过程，并等待来自 gdb 的连接，应该将参数“gdb”传递给内核。这可以通过在 LILO 命令行内核名字之后输入“gdb”来完成。将默认的串行设备和波特率设置为 ttyS1 和 38400。这些参数可以在命令行中通过使用“gdbttyS=”和“gdbbaud=”来更改。

当内核启动到需要等待来自 gdb 客户端的连接点时，需要在开发机器上完成 3 件事：设置目标机器可接受的波特率，设置它使用的串口，重新启动目标机器。这些可以通过以下 3 条命令在 gdb 中完成：

```
* set remotebaud <your baud rate>
* target remote <tbl local serial port name>
* continue
```

为了在目标机器上触发调试模式，既可以在目标机器上按下 Ctrl C，也可以从开发机器上使用 gdb 命令 interrupt（中断）。然后，可以使用所有 gdb 命令来跟踪或调试目标 Linux 机器。

提示

- 在目标机上设置成功后将在内核启动期间显示一条类似“等待远程调试…”的提示消息。kgdb 将等待来自开发机器的命令，以便指示它的下一步。通常情况下，将得到 gdb 命令 continue（继续）。如果没有任何输入命令，目标将冻结。

C.3 维护

当软件项目很小时，很容易记住大多数数据结构定义和函数实现的位置。然而，当项目变大时，就变得很难记住一切了。你需要一个工具来帮助你管理数百个变量和函数声明。同样的道理，好的项目也出自优秀的开发者团队。代码开发者必须使用版本控制系统来同步来自彼此的修改。本节介绍 cscope，它为单个开发者处理源代码的浏览任务，而 GIT 为代码开发者控制源代码。

C.3.1 源代码浏览器：cscope

什么是cscope

cscope是开放源代码的浏览工具，最初是由贝尔实验室开发的，2000年，它最开始能够搜索变量、宏、函数声明、预处理函数和函数调用者，以及替换文本，并不能调用外部文本编辑器来修改源代码。它如此强大，以至于AT&T公司用它来管理涉及500万行的C/C++代码项目。

如何使用cscope

使用cscope的第一步是从源文件（.c）和头文件（.h）建立交叉引用表。cscope默认这些文件的列表是以名为cscope.files的文件编写的。因此，可以发出以下的UNIX命令来准备cscope.files：

```
% find . -name "*.[chly]" -print | sort > cscope.files
```

可以利用下面的命令告诉cscope来构建交叉引用表：

```
% cscope -b -q [-k]
```

其中标志-b构建交叉引用表，标志-q激活可建反向索引表，标志-k是一个可选的标志，只有当项目是内核源代码的一部分时才使用。执行命令后，创建3个文件：

cscope.out：交叉索引表

cscope.in.out和cscope.po.out：反向索引表

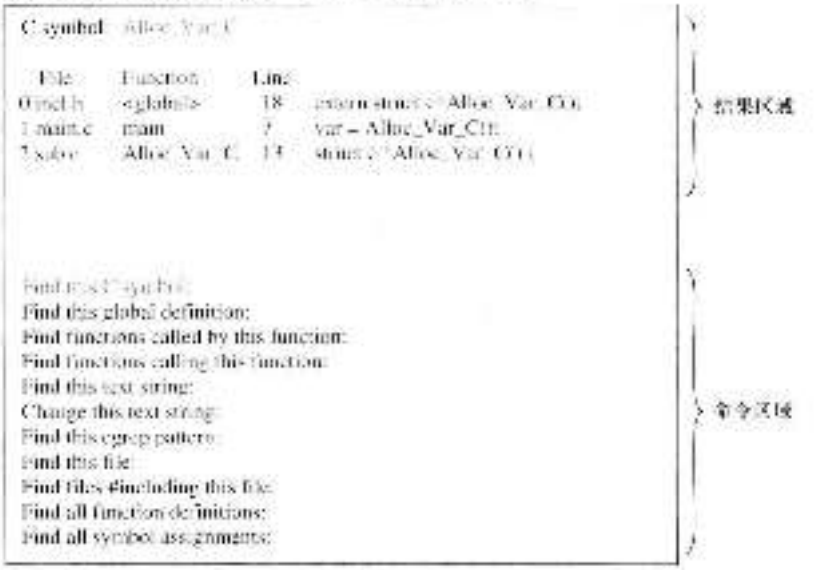
接下来运行cscope。尝试如下命令，将执行带有文本模式用户界面的cscope：

```
% cscope -d
```

这里标志-d指示使用现有的交叉引用表而不更新它们。此外，cscope可以与emacs或者vim结合，如通过将“cs add cscope.out”添加到.vimrc中实现与vim的集成。

例子

图C.7显示了一个cscope屏幕截图。在两个区域显示在cscope界面上。一个是cscope函数列出的命令区域。另一种是显示查询结果的结果区域。可以通过按下TAB键，在这两个区域之间切换。此外，通过在该区域使用UP和DOWN键，可以在命令区域切换命令或者在结果区域选择结果。尝试在命令区域中指定一个函数名，“查找这个C符号”（Find this C symbol），结果将显示在结果区域内。现在，可以切换到结果区域，选择一个项目，按下回车键（Enter）调用外部文本编辑器来修改它。最后，按下？键为你显示帮助手册，按下CTRL-D退出cscope。



图C.7 运行cscope的一个例子

提示

- 当你忘记了一个符号的全名时，cscope 命令 "find this egrep pattern"（查找本 egrep 的模式）可以帮助你找出你想要的搜索。
- 当你需要一次改变一个广泛地分布在许多文件中的符号名称时，使用 cscope 命令，"Change this text string"（更改这个文本字符串）。

C.3.2 版本控制：Git

什么是Git

Git[1]（全球信息跟踪器）是一个源代码控制系统，最初由 Linus Torvalds 所开发。Git 的两个著名特点是高性能和分布式体系结构。因为它的高效率，目前有数百个开源项目，包括 Linux 内核和 Google Android 操作系统，都在由 Git 来控制的。Git 提供分布式控制体系结构。每个开发人员通过使用 Git 可以决定何时以及如何分支一个项目。

如何使用Git

第一件事就是创建一个库。库是一个包含源控制所需的目录。既可以通过初始化一个容纳工作项目的目录，也可以从一个现有的 Git 控制的项目克隆来拥有一个库。可以通过以下命令来完成：

```
* git init project_directory
```

或者：

```
* git clone git_controlled_url
```

这里 git controlled url 是其他开发者主持的 Git 库的位置。git controlled url 的一些典型格式列于表 C-2 中。

表 C-2 Git 提供的重要 git_controlled_url

格　式	描　述	Git 检查的例子
local_path	在 local_path 上的 Git 库	Git 克隆/home/Bob/project
http://host/path	Git 由 Git 感知为 web 服务器控制[2]	Git 克隆 https://1.2.3.4/project.git
http://host/path	与上面相同，但已带有 SSL 加密	Git 克隆 git://1.2.3.4/project.git
ssh://user@host/remote_path	Git 库存储在 host/remote_path 中，可以使用 ssh 协议的安全签名访问它[3]	Git 克隆 ssh://Bob@1.2.3.4/home/Bob/project
git://host/remote_path	通过 Git 协议将 Git 库存储在远程主机[4]	Git 克隆 http://1.2.3.4/project.git

创建一个库后，就可以创建一个新文件或者修改库中现有的文件。提交修改之前，你可能需要检查上一个版本和当前工作项目之间的差异。可以通过以下命令来完成：

```
* git diff
```

为了提交你的修改，可以执行以下命令：

```
* git add ; git commit -m"your log messages"
```

通常，一个项目在开发过程中会留下几个分支。针对不同的需求来运行分支，有些分支实施实验性的思想，有些则就是不同客户的不同需求，另一些则是里程碑式的。在 Git 中创建一个新的分支是

---

[1] 实际上，Git 一词在英国俚语"蠢笨滑头，总是认为自己与正确的、好事者的。"Torvalds 被指控说："我是一个自私混蛋，我以自己的名字命名我所有项目。首先是 Linux，现在则是 Git。"

[2] 阅读在 Git 主页上的 Git 用户手册，了解如何将 Web 服务器与 Git 集成在一起。

[3] SSH（安全 Shell）是一种网络协议，在两台网络设备之间提供加密和认证的信息。它为管理员提供了与 Linux 发行版，如 Fedora 中，检查相关，可能需要对新配置防火墙设置，允许进入流量访问端口 22，并启动它的守护进程 sshd。

[4] 支持 Git 协议最简单的方式是通过命令来运行 Git 守护进程："Git daemon"。

通过以下命令实现的：

```
* git branch new_branch_name
```

可以通过以下命令列出现有的分支：

```
* git branch
```

请注意，目前活跃的分支都标有一个星号，默认分支的名称是主分支。分支之间的切换可以通过以下命令来完成：

```
* git checkout branch_name
```

将一个分支与当前活跃的分支合并，使用以下命令：

```
* git merge branch_name
```

如果项目是从现有的开源项目中克隆的，那么你也应该将你的更改方式给开源社区。Git 为代码开发者提供多种合并他们库的方法。最简单的方法是通过电子邮件发可源代码补丁。每个补丁都代表一个版本及其后续版本之间的差异。Git 使用下面的命令产生一系列的补丁，从克隆时开始到你提交的最新版本。

```
* git format-patch origin
```

事实上，补丁文件被格式化为电子邮件文件，因此它们可以通过电子邮件客户端直接发送给其他开发者。接收者使用命令导入补丁：

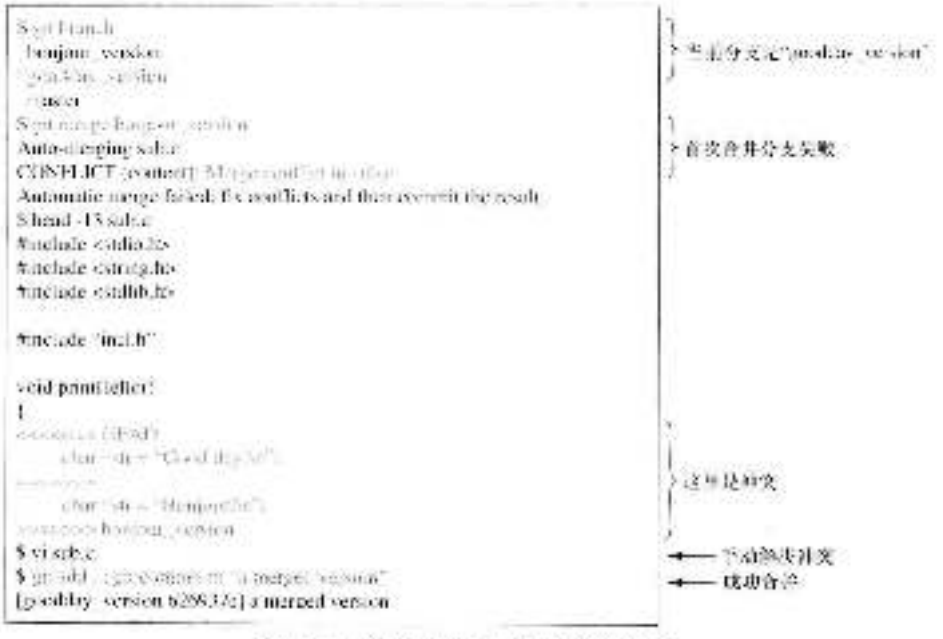
```
* git am *.patch
```

例子

虽然 Git 可以自动合并两个分支的源代码，但合并针对同一代码段上的不同修改所造成的冲突仍然不够聪明。不幸的是，这很常见，尤其是在热门的开发项目中。幸运的是，手动合并步骤足够简单能让每一个人学习掌握。

考虑下面的情况：两个分支以不同的方式修改相同的函数功能。比如，一个分支为 bonjour_version，指定字串“bonjour!\n”给变量 str。第二个分支为 goodday_version，修改同一变量，但是关联了字符串“Good day!\n”。当一个人要合并这两个分支时，就会产生冲突。

为了帮助解决这些冲突，Git 将发生冲突的代码段包括在 3 行：<<<<<<<，=======和>>>>>>>中。在小于符号和等于符号内的代码段属于当前分支，等于和大于符号内的源代码分段是其他分支的源。可以很容易地找到它们之间的区别，通过编辑源代码来解决冲突，然后成功地提交正确的版本。这个例子在图 C-8 中说明。



```
$ git branch
  bonjour_version
  goodday_version
  master
$ git merge bonjour_version
Auto-merging sub.c
CONFLICT (content): Merge conflict in sub.c
Automatic merge failed; fix conflicts and then commit the result.
$ head -13 sub.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "incl.h"

void printHelter()
{
<<<<<<< (HEAD)
      char *str = "Good day!\n";
=======
      char *str = "Bonjour!\n";
>>>>>>> bonjour_version
$ vi sub.c
$ git add -- gitcommit -- "a merged version"
[goodday_version 6298376] a merged version
```

当前分支是"goodday_version"

首次合并分支失败

这里是冲突

手动解决冲突

成功合并

图 C-8 需要手动合并 GIT 冲突的例子

提示

- CVS（并发版本系统）是另一种源代码控制系统，非常像一度流行的 RCS（修订控制系统）。与 RCS 不同，CVS 是集中式的。也就是说，它有一个中央数据库存储所有项目。每个开发人员都可以到他的私人目录中检查他自己树的副本。在 CVS 下，多个开发人员可以在同一时间工作在同一项目。
- SVN 子版本是 CVS 的后继者，因此它的句法不起来像原来的 CVS。与 CVS 相比，SVN 的主要优点是它支持事务。因此，当提交多个文件时，按照“全有或全无”的原则，SVN 保证所有的文件或者都成功地提交或者都没有改变。

C.4 分析

编码和调试后，程序员在开始执行任务了。如何才能知道程序是否有效地运行？如果没有分析，那么你可能需要一个秒表来评估它。分析（profiling）会记录一个正在运行程序的统计值。因此，开发人员通过分析报告可以测量其实现的性能。本节介绍了两个分析工具，一个用于用户空间应用程序，另一个用于内核程序。

C.4.1 分析器：gprof

什么是gprof

GNU 分析器 gprof 能够让你分析一个正在运行的程序。它将分析结果报告分为两个表，平面分析（flat profile）和调用图。平面分析报告每个函数的调用次数和每个函数消耗的时间。调用图进一步详细说明消耗的时间以及某个函数与其后代的关系。检查 gprof 的结果，使你能够发现程序中的瓶颈和低效的设计。

如何使用gprof

为了使用 gprof，程序必须带有一个特殊 gcc 标志 -pg 重新编译，以便 gcc 可以将监控和记录提交给程序。例如，为了启用名为 prog 程序的分析功能，可以使用命令

```
gcc -pg -o prog prog.c
```

然后程序像往常一样运行，以便收集分析结果。将结果存储在文件 gmon.out 中。程序终止后，可以通过执行下面命令来读取结果

```
gprof -b program_name
```

这里标志 -b 将告诉 gprof 不要显示详细的说明。

例子

在图 C.9 中显示了由 gprof 报告结果的例子。有 3 个函数 funcA、funcB 和 funcC 可以被主例程使用。在平面分析中报告，funcA 消耗了大部分时间，约为 3.38 秒，funcB 反复被调用了 101 次。在 101 次调用中，调用图进一步说明其中的 100 次调用来自 funcC。

提示

- 为了分析后台守护进程（daemon），必须关闭其后台守护进程的功能，因为只有当一个程序终止时才能提供分析结果。在大多数情况下，可以按如下步骤做。先找到函数调用 daemon，然后再将它注释掉。
- 在 Linux 中，有两个知名的分析器 LTTng（lttng.org）和 Oprofile（oprofile.sourceforge.net）。LTTng 需要使用由 LTTng 提供的分析 API 或者由测试人员编写的回调函数提交目标程序的源代码。Oprofile 能够在不修改源代码的情况下分析程序。与 gprof 使用的编译器协助技术不同，Oprofile 依靠于内核驱动程序。与 Oprofile 一起的驱动程序定期地收集统计信息。Oprofile 的优点是，它可以在没有源代码的情况下分析多个程序（它称为全系统分析器）；缺点是，它的系统开销太大并需要超级用户权限。

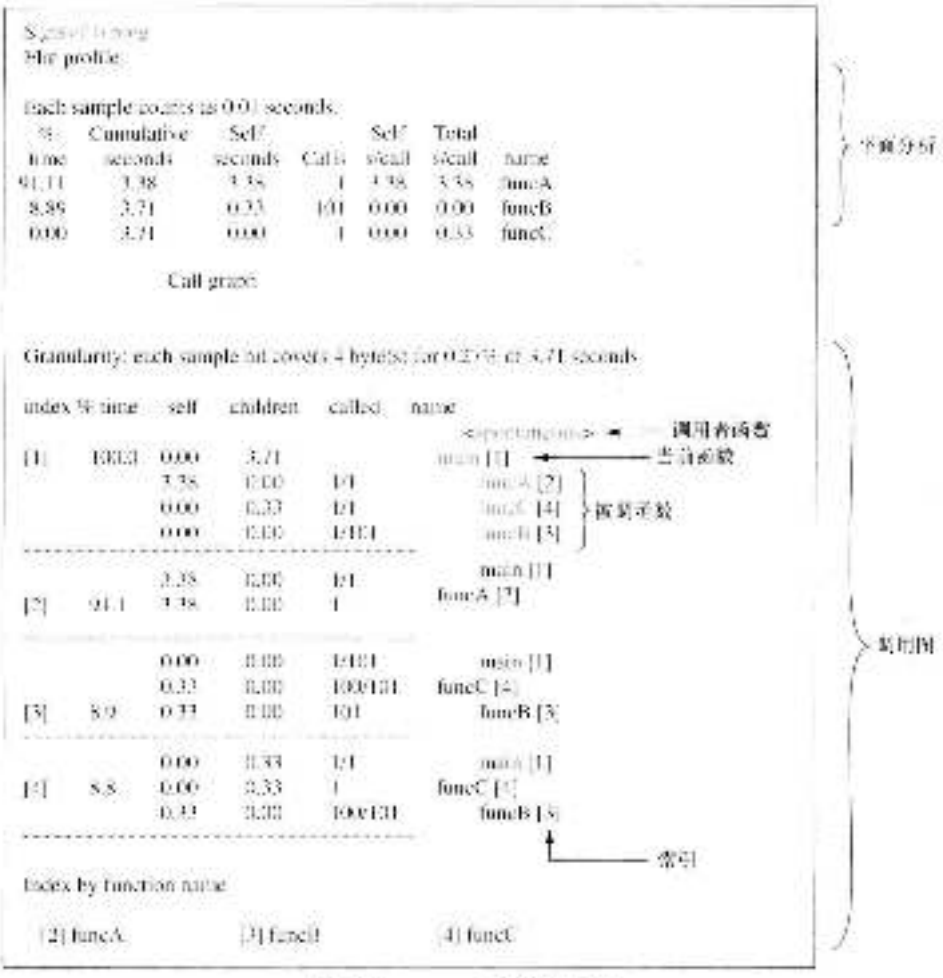


图 C-9 gprof 的屏幕截图

C.4.2 内核分析器：kernprof

什么是内核分析器

kernprof 是一套 Linux 内核补丁和工具，由硅谷图形公司（Silicon Graphics International，SGI）以开源的形式提供。利用 kernprof，系统分析人员就能看到每个内核函数消耗的时间，从而找到内核中的瓶颈，类似于 gprof 为用户空间应用程序所做的。

如何使用kernprof

为了使用 kernprof，必须为内核源代码打补丁。因此，你必须首先在 kernprof 的网页上下载与你的内核版本相匹配的补丁。为内核打过补丁后，你可以重新编译并安装打过补丁的 kernprof 内核。其次，必须手动创建一和字符设备，它用于提供用户空间控制程序（名为 kernprof）与打过补丁的内核之间的通信信道。这可以通过以下命令来完成：

```
* mknod /dev/profile c 190 0
```

其中，值 0 代表系统的第一个 CPU。同样，你可以为余下的每个 CPU 创建一个字符设备。

kernprof 提供多种分析模式。程序计数器（PC）抽样模式定期搜集执行函数的信息，其结果与 gprof 生成的平面分析类似。使用图模式将给一个调用图，这对于内核跟踪很有用。带有注释的调用图模式是上述两种方式的混合，因此结果与 gprof 的默认输出相匹配。与 gprof 在程序运行期间收集信息不同，你必须亲自通过发出的命令明确地启动与关闭 kernprof。例如，用注释的调用图模式启

动 kernprof，你可以指定命令

```
* kernprof -b -t acg
```

这里标志-t acg 表示注释的调用图模式。为了停止 kernprof 并产生一个可读结果的 gprof，即 gnome out。可以使用以下命令：

```
* kernprof -e
* kernprof -q
```

最后，通过发布以下命令，也可以使用 gprof 阅读结果。

```
* gprof file_of_vmlinux
```

提示

- 一个激活使用（enabled）kernprof 的内核会使 Linux 的运行变慢。根据 kernprof 的常见问题解答，它可能会将系统性能降低 15% 以上。因此你可以准备两个内核，其中一个是激活使用 kernprof，而另一个是正常的，并使用 boot loader 引导程序在它们之间进行切换。
- 除了前面提到的 LTTng 和 Oprofile 外，还有一种用于 Linux 内核分析的著名分析工具，内核函数跟踪器 Kernel Function Tracer，KFT（elinux.org/Kernel Function Trace）。在分析之前，与 kernprof 一样，LTTng 和 KFT 需要内核补丁，而 Oprofile 是以内核驱动程序和用户空间后台守护程序的方式提供。因此，它可以动态地加载并执行而不会污染内核源代码。

C.5 嵌入式

将你的工程移植到一个嵌入式系统比在桌面上开发难得多。首先最重要的设计目标是削减代码大小，因为嵌入式系统具有有限的资源。此外，你可能需要一个工具链，即交叉编译器和函数库，为具体的目标结构编译和链接程序，并且准备根文件系统。文件系统包含"/"根目录和启动所需要的所有文件和目录，如/bin、/etc 和/dev 目录。有开源工程来帮助你构建一个足够微小的嵌入式 Linux。本节将讨论如何使用空间优化的公共程序（所谓的工具），轻载工具链和嵌入式根文件系统加快移植工作。

C.5.1 微小的实用程序：busybox

什么是busybox

需要几十个基本实用程序来运行一个 Linux 应用程序。然而，这些实用程序中的许多具有共同的例程，如字符串复制，很少使用的函数，如按需帮助，以及不需要的文档，如操作手册。最好删除它们以降低程序的大小。busybox 将许多常见的和基本的实用程序集成到一个空间优化的程序中。

如何使用busybox

因为 busybox 是高度可配置的，所以编写一个定制的版本非常容易。首先，你可以使用命令

```
* make menuconfig
```

来选择你需要的实用程序并禁用一些不必要的实用程序。特别是在 menuconfig 屏幕上，你可以用 UP 和 DOWN 键来移动光标，Enter 键来选择子菜单，SPACE（空格键）选择/去掉一个选项，并选择 Exit 选项来保存和退出子菜单或配置。图 C-10 是 menuconfig 的截图。

其次，输入

```
* make
```

来编译 busybox，它将在当前的目录下产生一个可执行的、名为 busybox 的程序。在运行时通过首先检查其名字 busybox 充当不同的实用程序，所以你必须将每个实用程序名字的一个符号连接构建到 busybox 中。例如，如果实用程序 find 被 busybox 替代了，符号链接

```
find-busybox
```

就必须存在。因此，安装 busybox 涉及将程序复制到嵌入式系统中并准备符号链接。

提示

- 虽然配置和编译 busybox 都很容易，但最难的部分是选择嵌入式系统中真正必要的。一个捷径是观察现有的嵌入式 Linux 系统的根文件系统，例如，C.5.2 节介绍的由 buildroot 构建的根文件系统。

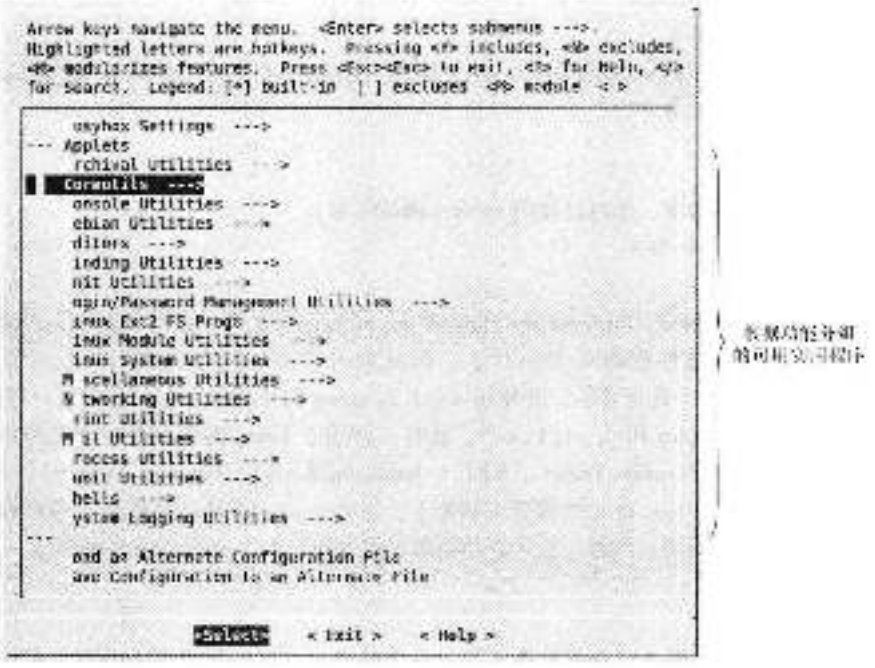


图 C.10 busybox的配置

C.5.2 嵌入式开发：uClibc和buildroot

什么是uClibc和什么是buildroot

GNU C库，glibc是用于Linux桌面或服务器系统最常见的C库。它兼容各种C标准和现代系统，当然其弊性会增加它的大小。它还提供了许多方法来优化运行速度，虽然它们中的某些需要更多的内存空间。uClibc为嵌入式系统进行了全新的设计。因此，一个与uClibc连接的程序可能比与glibc连接的程序小很多。

一个C库需要一个相应的工具链，也就是说，需要交叉编译器和系统软件工具来将程序与它连接。uClibc开发团队认为一次性备好有这些的最容易方式就是使用buildroot工程。buildroot是能够自动地从互联网下载所需要软件包以构建一个定制的根文件系统的一组Makefile。默认情况下，它编译并将程序与uClibc连接起来。此外，它利用C.5.1节介绍的busybox。因此，由buildroot构建的文件系统所需要的空间较小并适合嵌入式系统使用。

如何使用buildroot

像busybox一样，buildroot是高度可配置的，它们的编译步骤是相同的。因此，我们输入命令make menuconfig来配置设置，并输入make编译buildroot。由buildroot工程构建的文件系统镜像位于buildroot/output/images/rootfs.tar中。图C.11是它的menuconfig的屏幕截图。

提示

- 如果目标机器与开发平台具有相同的体系结构，那么就有一种验证构建的根文件系统功能性的方法[5]。第一步是要定位找到编译的根文件系统。buildroot编译接近结束时，一个变量，staging_dir，就是我们所要寻求的。假定目录名叫作directory_root，也就是说，staging_dir=directory_root。那么，键入chroot directory_root sh，从而将root更改为编译过的。然后你就可以执行任何程序就好像你已经安装在目标机上一样。在任何时候，你都可以使用命令exit，返回到你原来的shell。

[5] 例如，为了确定检查shell脚本是否可以加载某些程序，/bin/your_prog，一种方法是将一个root登录到目标机器上，然后使用root shell来验证其执行情况。在本提示中提供的类似方法在开发平台上测试它。

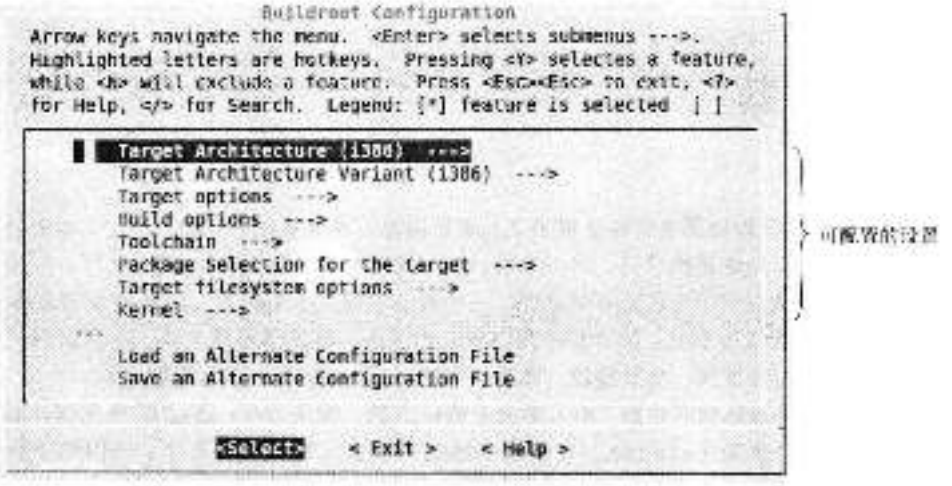


图 C-11 配置 buildroot

进一步阅读

相关书籍

下列书籍涵盖了本附录中提到的大部分主题。但没有一本关于分析（profiling）所编写的书籍值得推荐。

- R. Mecklenburg, *Managing Projects with GNU Make (Nutshell Handbooks)*, 3rd edition, O'Reilly Media, 2009.
- R. M. Stallman, R. Pesch, and S. Shebs, *Debugging with GDB: The GNU Source-Level Debugger*, 9th edition, Free Software Foundation, 2002.
- M. Bar and K. Fogel, *Open Source Development with CVS*, 3rd edition, Paraglyph, 2003.
- C. Pilato, B. Collins-Sussman, and B. Fitzpatrick, *Version Control with Subversion*, 2nd edition, O'Reilly Media, 2008.
- C. Hallinan, *Embedded Linux Primer: A Practical Real World Approach*, Prentice Hall, 2006.

在线链接

这里，我们总结了本附录介绍的所有开发工具的网站。这些经典网站很可能继续存在很多年。

1. VIM (Vi IMproved), http://www.vim.org/
2. gedit, http://projects.gnome.org/gedit/
3. GCC, http://gcc.gnu.org/
4. GNU Make, http://www.gnu.org/software/make/make.html
5. GDB, http://sourceware.redhat.com/gdb/
6. DDD, http://www.gnu.org/manual/ddd/
7. kGDB, http://kgdb.sourceforge.net/
8. cscope, http://cscope.sourceforge.net/
9. CVS, http://www.cvshome.org/
10. GNU gprof, http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html
11. Kernprof, http://oss.sgi.com/projects/kernprof/
12. BusyBox, http://www.busybox.net/
13. uClibc, http://www.uclibc.org/
14. Buildroot, http://buildroot.uclibc.org/

网络实用工具

Linux 系统用户和管理员都常常需要很多工具来帮助他们理解系统。例如，可能需要检查主机的 IP 地址或者查看网络接口的流量统计值。另一方面，除了在附录 C 中给出的开发工具外，开发人员也可能需要工具来观察网络以便于更容易调试进程。一个真正系统的开发前后，人们可能需要模拟（或模仿）系统设计并测试开发的系统。综合起来我们将上述这些工具称为网络工具。本附录将这些工具分为六类：名字寻址、边缘探测、流量监控、基准、模拟/仿真，最后还包括黑客工具。

D.1 节讨论了名字寻址如何帮助了解互联网上的名人的，使用 host 通过 DNS 查询并利用地址解析协议（arp）和接口配置（ifconfig）获取本地的（例如，LAN）名人录。当网络不能像预期的那样工作时，就应该利用 D.2 节中讨论的边缘探测或者 ping 可用的远程主机或者 traceroute 任何网络段。一旦完成了故障诊断，分组就应该开始流动了。D.3 节给出了用于流量监控的工具。为了使用 tcpdump 或 Wireshark 详细检查头部和有效载荷，可以转储分组。可以使用 netstat 搜集有用的网络统计值和信息。

因为性能是一个关键的问题，所以当可联网络的性能通过测试后才认为它是可以工作的。因此，D.4 节介绍标准管理工具，TestTCP（ttcp）对主机到主机的吞吐量进行分析。另一方面，开发一个系统而没有预先评估设计和需要昂贵的和危险的。在这种情况下，应该使用 D.5 节讨论的网络模拟器（ns）模拟或者使用 NIST Net 仿真。最后，D.6 节简要地描述了 Nessus 漏洞扫描（Exploit scanning）的黑客方法。这可能会有争议，但是放在这里作为第 8 章的补充。

D.1 名字寻址

通信的第一步通常是将对等的名字解析为 IP 地址或者将对等的 IP 地址解析为 MAC 地址。前者是第 6 章讨论的可联网的名人录，可以通过域名系统（DNS）来完成；而后者是在第 4 章中学习过的本地名人录，可以通过地址解析协议（ARP）来完成。本节将讨论名字寻址工具如何帮助了解互联网或局域网上的名人录。

D.1.1 互联网名人录：host

什么是host

host 是一个程序，能让用户查询 IP 地址对应的域名，反之亦然。它实现了 DNS 协议与本地 DNS 服务器的通信，依次查询具有映射的其他 DNS 服务器。

如何使用host

实用 host 很简单。查询域名的 IP 地址，直接执行下列命令

```
% host domain_name
```

同样，寻找 IP 地址的域名，可以如下操作

```
% host ip_address
```

例子

在如图 D-1 所示的例子中，我们想要查询 www.google.com 的 IP 地址。host 告诉我们，网络 www.google.com 有一个别名 www.l.google.com，并且该名字绑定了 6 个 IP 地址。

提示

- 默认情况下，host 向系统配置的本地 DNS 服务器发送查询。你也可以通过以下命令指定一个域名服务器，如 target_dns。

```
% host query_name target_dns
```

```
Server:         [illegible]
www.google.com is an alias for www.l.google.com.
www.l.google.com has address 74.125.153.103
www.l.google.com has address 74.125.153.104
www.l.google.com has address 74.125.153.105
www.l.google.com has address 74.125.153.106
www.l.google.com has address 74.125.153.147
www.l.google.com has address 74.125.153.99
```

图 D-1 使用 host 的一个例子

D.1.2 局域网的名人录：arp

什么是arp

上层应用程序间的通信是通过 IP 层，而在局域网内实际分组的发送是根据 MAC 地址。arp 是一个帮助用户查询 IP 地址的 MAC 地址，或者反之亦然的程序。管理员也可以使用 arp 管理整个系统的 ARP 表，比如添加一个静态的 ARP 表项。

arp 程序内部是完成 ARP 协议。基本上，arp 在局域网上广播一个 ARP 请求信息以便查询一个特定 IP 地址的局域网 MAC 地址。实用该 IP 地址的设备会发送一个单播 ARP 回应查询主机。结果可能动态缓存在查询主机的系统 ARP 表中以便加快将来的查询请求响应。

如何使用arp

使用 arp 很简单。为了查询 IP 地址的 MAC 地址，可以指定命令

```
* arp -a IP_address
```

使用下列命令在 ARP 表中添加一个表项

```
* arp -s IP_address MAC_address
```

删除一个表项，使用下列命令

```
* arp -d IP_address
```

最后，可以通过输入以下命令浏览系统级的 ARP 表

```
* arp
```

例子

图 D-2 演示了系统 ARP 表的浏览结果。在桌面上，我们可以看到有两个表项，88-router.cis.ncku.edu.tw 和 140.113.88.140，绑定网络接口 eth0。标志 C 显示这是系统上的一个缓存表项（不是一个静态表项）。

```
$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
88-router.cis.ncku.edu.t ether   00:19:06:e8:0e:4b   C                     eth0
140.113.88.140           ether   00:16:35:ac:f5:6c   C                     eth0
```

图 D-2 使用 arp 的例子

提示

- 当 LAN 内的一台主机改变它的网络适配器时，由于 ARP 缓存表的缘故你可能无法立即访问到它。为了解决这一问题，既可以等待缓存超时也可以使用 arp-d 删除上述缓存表项。

D.1.3 我是谁：ifconfig

什么是ifconfig

ifconfig（InterFace CONFIGuration）是一个程序，它允许用户查询 IP 地址、MAC 地址和网络接口统计值。管理员也可以用它来配置 IP 地址，并启用/禁用网络接口。

如何使用ifconfig

使用 ifconfig 非常简单。为了查询网络接口的设置，可以指定命令

```
* ifconfig [interface_name]
```

这里 interface_name 是一个可选的参数，用于指定网络接口。没有指定任何参数时，ifconfig 会显示当前活跃接口的设置。管理员可以使用

```
* ifconfig interface_name hw ether 12:34:56:78:90:ab
```

设置网络接口的 MAC 地址，并使用

```
* ifconfig interface_name down/up
```

禁用/启用接口。

例子

图 D-3 是使用 ifconfig 的例子。结果说明该系统具有一个接口，名为 eth0。IP 地址为 192.168.1.1，MAC 地址为 00:1D:92:F7:8A:F9。接口目前为活跃的（标注 UP 标志）并已经传输了 296 781 个分组（或大约为 105MB）。其他输出的详细含义可从联机手册上找到。

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1D:92:F7:8A:F9
          inet addr:192.168.88.253  Bcast:192.168.88.255  Mask:255.255.255.0
          inet6 addr: fe80::21d:92ff:fef7:8af9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1147154 errors:0 dropped:0 overruns:0 frame:0
          TX packets:796781 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:312e08565 (298.1 MiB)  TX bytes:1a0f0e54 (105.9 MiB)
          Memory:fe940000-fe960000
```

图 D-3 使用 ifconfig 的例子

提示

- 在 Microsoft Windows 平台上，有一个类似的命令行程序，ipconfig。

D.2 边缘探测

当网络不能像预期那样工作时，就可以运用边缘探测到工具检验主机的可用性或者找到网络的瓶颈。

D.2.1 探测是否存活：ping

什么是ping

ping 可以检查从主机到目标机器之间路径的可用性。它使用两条互联网控制信息协议（ICMP）定义的消息。第一条消息是 ICMP echo 请求，这是从主机发送到目标的消息。当接收到请求时，目标使用一条 ICMP echo 应答消息对主机做出回应。因此主机就能知道可用性并计算请求和应答之间的时间间隔。

如何ping

尝试以下命令

```
* ping target_machine
```

检查你的系统和目标之间主机的可用性。按 Ctrl-C 终止检查，得到一个总结报告。

例子

图 D-4 中的例子显示了 ping 的结果。通过阅读结果，我们可以知道到目标 192.168.1.2 的分组丢失率和响应时间，包括最小、平均和最大响应时间。

提示

- 默认地，ping 每秒发送一个请求。可以通过指定间隔选项 -i 调整，即

  ```
  ping -i 10 192.168.1.2
  ```

  表示每隔 10 秒发送一个 ICMP echo 请求。

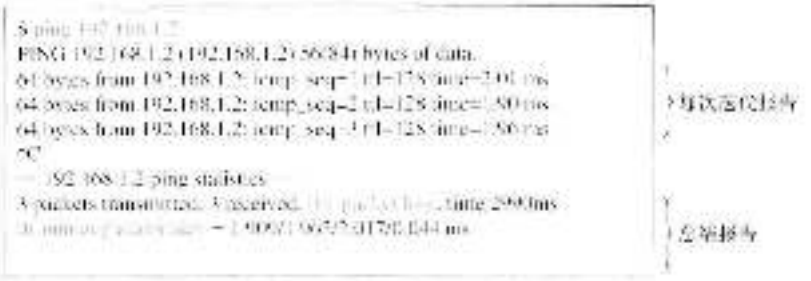


图 D-4 标准 ping 的例子

D.2.2 找路径：tracepath

什么是tracepath

使用 ping，可能发现到达目标的分组丢失率异常高或者耗费时间很长。为了找到你的主机和目标之间路径上的“瓶颈”，就要使用tracepath 工具。

tracepath 利用 IP 头部中的存活时间（TTL）字段一次发送一个 UDP/IP 探测信息。其中 TTL 设置为 1，这样最近的路由器将立即向源回应一个 ICMP 应时。据此可以计算到最近路由器的往返主间（RTT）。同样，tracepath 在查询信息中设置 TTL 以测量到外路由器的 RTT。有了这些 RTT，用户就可以找到从源到目标路径上的瓶颈。tracepath 自能发现路径上的最大传输单元（MTU）。

如何使用 tracepath

使用 tracepath 很容易。为了检查到达目标机器的路径，可以简单地使用命令

```
$ tracepath [target_machine]
```

可以添加一个标志 -pktlen，设置初始查询消息的大小。当遇到从中间路由器返回来的因消息太长的拒绝时，tracepath 会自动调整探测长度。一个可选的/port 参数可以附加在 target_machine之后，来指定在 UDP 查询信息中的目标端口号。在 tracepath 的某些版本中，默认为 44444。在另一些版本中则是随机选择的。不幸的是，有些路由器只回应目标端口范围从 33434 - 33534 之间的查询信息，这些设置在经典的 traceroute 工具中。因此，当你使用工具时，我们建议你明确地指定端口号（33434 是一个奇怪的神奇的号码）。

例子

图 D-5 显示对 www.google.com 使用 tracepath 的结果。在报告每个中间路由器（以及月份）之间的往返行程时间显示为一行结果。开始，tracepath 发出一个 2000 字节的查询信息。第一跳拒绝

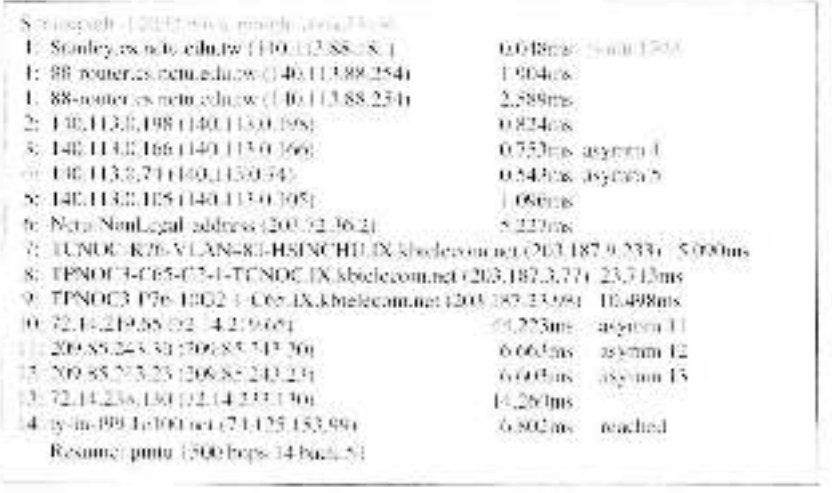


图 D-5 使用 tracepath 的例子

了它，并要求 traceparh 使用 1500 字节，即在第一行消息 "proto 1500"。1500 字节消息对所有其余测量可以接受的。最后，消息 "asymm 0" 代表由 traceparh 发现的一个可能不对称的路由路径。

提示

- traceroute 是所有基于 UNIX 世界中的另一种选择。但是，出于安全考虑，有些 Linux 发行版，如 ubuntu，并没有包括它。
- tracert 是一个在 Windows 平台上相类似的工具。不使用 UDP，tracert 而是使用 ICMP echo 请求作为它的查询消息。

D.3 流量监控

网络互连协议的实现有待于在实际网络上的验证。本节介绍流量监控工具。分组可以转储以便能够详细地检查它们的头部和有效载荷，并且可以收集一些有用的统计值和信息。

D.3.1 转储原始数据：tcpdump

什么是tcpdump

tcpdump 是最受欢迎的命令行嗅探器，能够让有特权的用户转储在一个网络接口上接收的流量。转储流量可以立即在控制台上输出或者保存在文件中用于稍后的分析。tcpdump 的强大来自使用了 libpcap 库，它为捕获流量提供了一个编程接口。Windows 平台的一个项目，WinDump 就是 tcpdump 的移植。

如何使用tcpdump

为了捕捉到所有信息，你可以只输入命令

```
$ tcp-dump
```

并按下 Ctrl-C 终止捕捉。使用 tcpdump 更常见的方式是设置过滤条件，以便只转储匹配条件的分组。tcpdump 可以利用很多过滤条件。这里我们只结合实例介绍重要的条件，它足能满足最常用的抓获要求以便于协议分析。其场景如下：你想要记录流的源或目的 IP 地址是 target_machine，TCP 端口号是 target_port。假定 target_machine 在网络接口 eth0。tcpdump 命令将为

```
$ tcpdump -i eth0 -X -s 0 host target_machine and port 80
```

这里参数-i eth0 告诉 tcpdump 追踪经过 eth0 的分组。参数-X -s 0 请求 tcpdump 输出完整分组包含头部和有效载荷，表达式 host target_machine 指示主机只捕捉源或目的地是 target_machine 的分组，同样，表达式 port 80 限制了端口号。

例子

图 D-6 显示追踪两个 ping 运行的结果。为了将抽获限制为只有 4 个分组，可以指定参数-c 4。

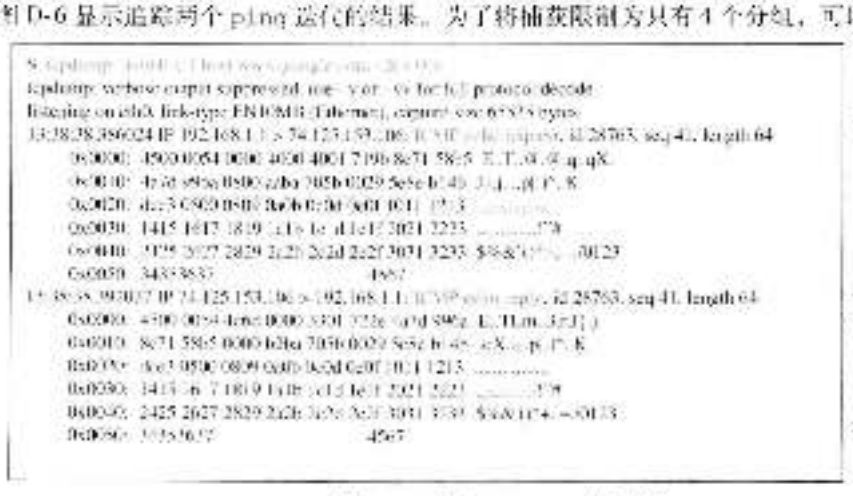


图 D-6 使用 tcpdump 的例子

其中有两列给出一个被捕获的分组：左边以十六进制字符给出，右边以 ASCII 字符显示。不可输出的字符用点代替。

D.3.2 GUI 嗅探器：Wireshark

什么是Wireshark

Wireshark 是另一个具有 GUI 功能的嗅探器。Ethereal 是它最初的名字，这个项目于 2006 年被更名为 Wireshark。

如何使用Wireshark

在 Wireshark 中开始抽获可以通过按下菜单栏里 Capture 抽获了菜单中的 Interfaces 按钮来完成，也可以在于菜单中按到 stop 按钮。

Wireshark 的强大来自它友好的用户界面。如图 D.7 所示，在 Wireshark 的主窗口有 4 个主要区域。第一个区域是过滤器，你可以设置过滤的点，既可以直接输入过滤规则，也可以使用表达式 Expression 按钮。第二个区域显示捕获的分组。当光标指向一个表项时，它的描述，如 MAC 地址，会显示在第三个区域内。最后，第四个区域将显示完整分组的内容。



图 D-7 Wireshark 的屏幕截图

D.3.3 收集网络统计值：netstat

什么是netstat

netstat 是一个命令行工具，能够显示连接状态、协议使用情况的统计值和路由表。

如何使用netstat

netstat 的第一个主要功能是显示连接状态。可以键入以下命令

```
* netstat -ad
```

其中标志-t 告诉 netstat 列出所有协议的状态，标志-n 说明以数值形式显示结果的地址，这要比显示域名快很多。

netstat 的第二个功能就是显示协议使用情况的统计值，可以通过以下命令来完成

```
* netstat -s
```

最后一项功能是通过执行以下命令给出路由表

```
* netstat -rn
```

例子

图 D-8 显示了连接状态。从图 D-8 中可以看到这台机器正在监听许多端口（带有状态 LISTEN），例如，80（Apache Web 服务），有一条来自 192.168.1.2 的连接。

提示

- 连接状态作为一种检测黑客的工具也非常有用。例如，拒绝服务（DoS）攻击的特征就是数千个非 LISTEN 连接。可以从 netstat 的结果逆向追踪攻击源。

```
$ netstat
Active Internet connections (servers and established)
Proto Recv Q Send Q Local Address          Foreign Address        State
tcp        0      0 0.0.0.0:22             3.3.3.3:*              LISTEN
tcp        0      0 0.0.0.0:80             3.3.3.3:*              LISTEN
tcp        0      0 [illegible]            [illegible]            [illegible]
```

图 D-8 netstat 的运行结果

D.4 标杆分析法

一个连接好的网络只有经过测试和测量才认为是可以工作的。本节介绍了主机到主机吞吐量分析的常用标杆分析工具。

主机到主机吞吐量：ttcp

什么是ttcp

测试 TCP，即 ttcp，是用于在两台主机之间测试 TCP 和 UDP 吞吐量的标杆程序。有些路由器集成了这种工具的一个版本，使你很容易评估网络性能。

如何使用ttcp

ttcp 有两种模式，分别为传输模式和接收模式，分别由参数-t 和-r 来指定。标杆处理通过在一台机器运行 ttcp 接收模式开始，然后在另外一台机器上运行 ttcp 传输模式。你可以使用不同的工作负载，在传输模式的 ttcp 上还需以文件的形式给出。工作负载将传输到接收模式的 ttcp。统计值将会在传输的同端显示。结果将包括吞吐量和 I/O 调用次数/秒。

例子

在图 D-9 所示的例子中，发送方读取文件 test_file 作为工作负载，并将它传输到在 192.168.1.1 的接收方。接收方并不保存收到的内容而是直接丢弃，即输出到/dev/null。结果显示发送方需要 12 500 次 I/O 调用以便传输 102 400 000 字节，接收方测量的吞吐量为 723 557.59KB/s。

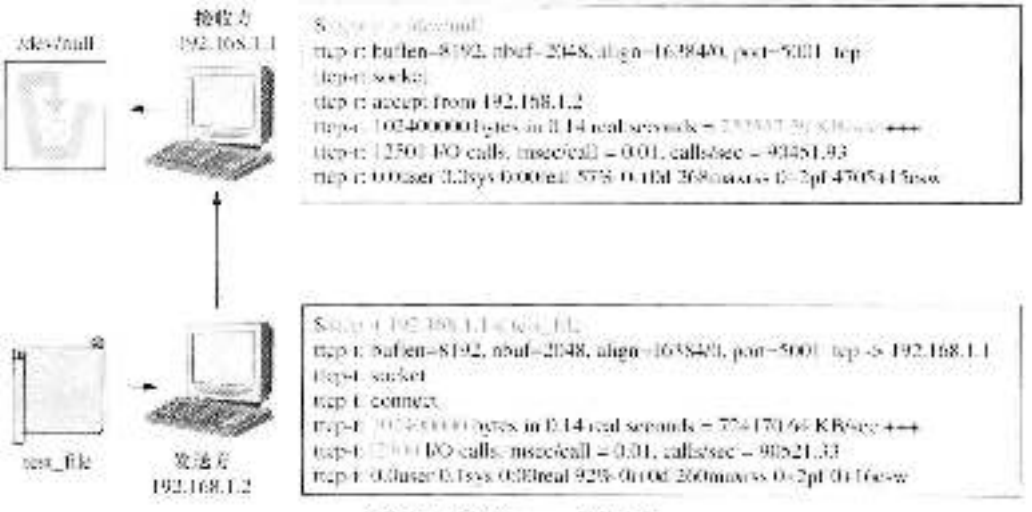


图 D-9 使用 ttcp 的例子

提示

- 为了在发送方一侧产生一个大的演示文件，可以如下使用命令 dd：
- dd if=/dev/zero of=demo_file size=<size_in_512_bytes>，这里 if=/dev/zero 告诉 dd 创建一个内容填充了零的文件，of=demo_file 指定输出文件名字，size_in_512_bytes 是一个指示输出文件大小的数字。

- 为了测量 UDP 吞吐量，可以在调用 iperf 时指定标志-u。

D.5 模拟和仿真

设计开发一个真实的网络会非常昂贵。在设计之前，可以完成很多便宜的性能评估。模拟或仿真工具可以用于评估全部网络或者网络组件的设计。

D.5.1 模拟网络：ns

什么是ns

始于 1989 年作为 REAL（实际和大型）网络模拟器的变种的 ns 是一种动作模拟平台，用以提供通常的参考和测试软件。以模拟不管是有线还是无线网络的状态的从链路层及以上各层的分组级的离散事件。它的几个强大功能包括场景生成，这可以创建一种定制的模拟环境，并在 nam（网络动画）帮助下实现可视化。值得注意的是，ns 是利用 C++（用于 ns 内核）和 OTcl（用于 ns 配置）两种语言实现的，以平衡运行时效率和场景编写的方便。该项目就是众所周知的 ns-2，因为版本 2 是其当前的稳定版本。

如何使用ns

编译 ns 很容易，因为项目有一个可以自动地进行配置和编译的脚本 install。使用以下命令构建 ns：

```
$ cd ns-allinone-<version>; ./install
```

其中 <version> 就是 ns 项目的版本号。截至 2009 年 6 月，最新的版本是 2.34。ns 模拟用 OTcl 脚本句法编写的网络场景。假设已经编写了一个场景脚本，名为 demo.tcl。你就可以执行如下命令进行模拟：

```
$ ns demo.tcl
```

场景包含网络类型、拓扑结构、节点、通信流量和定时事件。可以记录模拟过程，以便可以使用 ns 工具网络动画模拟器（nam）来可视化动画过程。

例子

利用 OTcl 脚本语言编写网络脚本有点儿复杂。幸运的是，ns 项目有很多例子脚本可以使用。你可以在目录中阅读它们：

```
$ cd ns-allinone-<version>/ns-<version>/tcl/ex
```

图 D-10 中演示了例子 simple.tcl 的模拟结果。在这个例子中，使用 nam 将 4 个节点连接起来，调度了两个通信流量，记录和可视化了模拟过程。

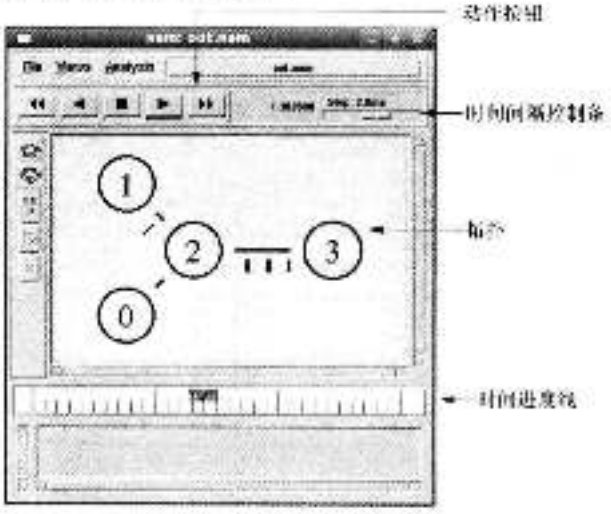


图 D-10 nam 的截取屏幕

D.5.2 仿真网络：NIST Net

什么是NIST Net

网络仿真器以较少的实验配置仿真各种网络类型。它检查了网络参数（如延迟、丢失、抖动等）的简单用户入口。利用NIST Net，用户就可以观察很多网络统计值，包括分组延迟、分组重新排序（由于延迟变化）、分组丢失、分组重复和带宽限制。图 D-11 说明了 NIST Net 的网络体系结构。直接连接到 NIST Net 的端点间信流量经历网络参数的冲击，就如同它们真的通过大的网络一样。



图 D-11 NIST Net 的网络体系结构

如何使用NIST Net

NIST Net 由用户空间工具和内核模块组成。内核模块仿真网络参数对数据流量的冲击。尽管是以内核模块给出而不需要内核补丁，但 NIST Net 的编译仍然要参考 Linux 内核源代码的设置，例如在/usr/src/linux/.config 中的内容。因此，在编译 NIST Net 之前，需要进行内核源代码的配置。这些可以在 Linux 内核源目录下通过输入如下命令来完成：

```
* make menuconfig
```

编译并安装 NIST Net 软件包之后，就可以使用以下命令来装载内核模块：

```
* load_nistnet
```

现在，用户可以使用命令：

```
* xnistnet
```

配置和监控 NIST Net。

例子

图 D-12 展示了 xnistnet 程序的屏幕截图。你可以添加一个源/目的地对并使用程序修改它的网络参数。地址对的通信流量将经历网络参数的冲击，如延迟、带宽和丢失率等。

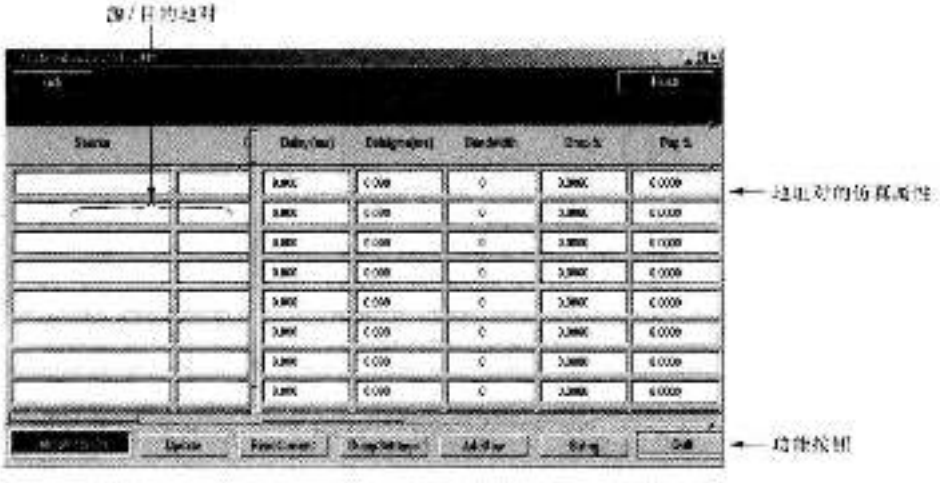


图 D-12 NIST Net 的屏幕截图

提示

- 为了中继数据流量，NIST Net 机器必须配置成路由激活模式。这可以通过在文件/proc/sys/net/ipv4/ip_forward 上将值设置成 1 来完成。换句话说，命令为：
  - echo 1 > /proc/sys/net/ipv4/ip_forward

D.6 黑客

本节给出使用搜索扫描工具的黑客方法。网络管理员可以使用这些工具来确定所管理网络的弱点和漏洞。

漏洞扫描：Nessus

什么是Nessus

Nessus 是一种用于 Linux 社区的最流行的扫描程序。如图 D-13 所示，Nessus 具有三层体系结构。客户机是一种允许管理员控制和管理 Nessus 守护进程的 GUI 程序。漏洞利用方法和黑客数据库构建到 Nessus 守护进程中。Nessus 守护进程还负责扫描目标网络，收集扫描结果，并向客户机报告。

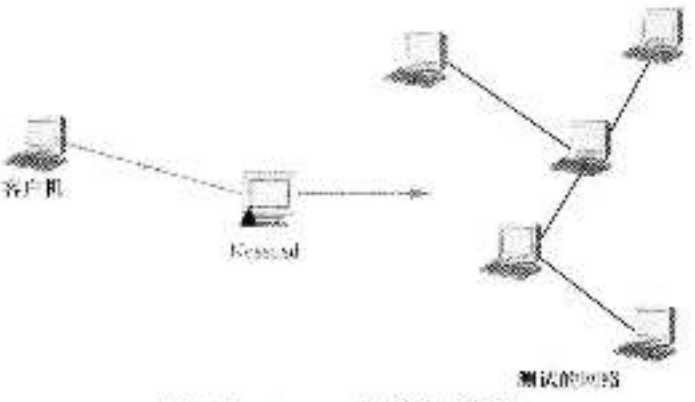


图 D-13 Nessus 的网络体系结构

如何使用Nessus

Nessus 由 4 个部分组成：nessus library、libnasl、nessus core，以及黑客数据库，在 Nessus 世界中称为 plugin。为了安装 Nessus，需要从 Nessus 的主页上下载这几部分，并依次进行编译和安装。接下来就是执行命令：

- nessus-mkcert

这将准备一张在 Nessus 客户机和守护进程之间通信中使用的证书。

成功安装后，就可以启动 Nessus 守护进程，即 nessusd，并通过运行以下命令添加第一个有效的 Nessus 管理员。

- nessus-adduser

然后你就可以在客户机器上运行 Nessus 客户机，即 nessus，并将客户机连接到守护进程上。

例子

为了检查目标机器的漏洞，Nessus 管理员首先使用 Nessus 客户机选择一些黑客工具。可以在选项卡上按下 plugin 按钮并在选项卡窗口上选择可用的黑客工具。选择窗口的屏幕截图如图 D-14 所示。接下来，在 Target 窗口上分配目标机器并按下 start the scan 按钮进行扫描。扫描之后，就会弹出一个报告窗口报告扫描结果。

提示

- 安装 Nessus 的动态连接库 libnasl 之后，将需要刷新系统范围内的库缓存。这就可以通过重启也可以通过使用如下命令来完成：
  - ldconfig /usr/local/lib

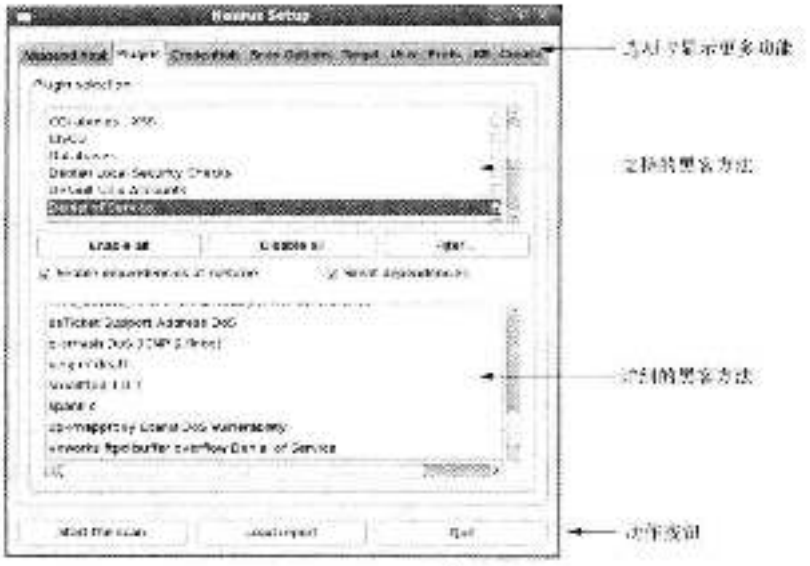


图 D-14 Nessus 2 的屏幕截图

- 注意，某些扫描方法会危害目标主机甚至还会导致它崩溃。
- 自从 Nessus 3 发布以来，Nessus 就不再是开源的了，而仅发布二进制可执行代码。对于非营利的组织仍然可以免费使用最新版本的 Nessus。目前有一个称为 OpenVAS 的分支项目是开源的并处在开发过程中。

进一步阅读

相关书籍

下列书籍覆盖了本附录中提到的大多数主题。第一本书介绍了如何使用 GNU 工具来管理网络。第二本书提供了 Linux TCP/IP 使用指导。第三本书，尽管是 1998 年出版的，但它仍然是学习网络编程的经典课本。

- Tobin Maginnis, *Sair Linux and GNU Certification, Level 1: Networking*, John Wiley & Sons, 2001.
- P. Eyler, *Networking Linux, a Practical Guide to TCP/IP*, New Riders, 2001.
- W. Richard Stevens, *UNIX Network Programming*, Prentice Hall, 1998.

在线链接

下面我们总结了在本附录中使用的所有网络工具的 Web 站点。当然这些经典的站点还可能持续存在几年。

1. arp and ifconfig, http://www.linuxfoundation.org/en/Net:Net-tools
2. nmst (a.k.a., bind9-host), https://www.isc.org/download
3. ping, http://directory.fsf.org/project/inetutils/
4. tracepath, http://www.skbuff.net/iputils
5. tcpdump, http://www.tcpdump.org/
6. Wireshark, http://www.wireshark.org/
7. ttcp, http://www.pcausa.com/Utilities/pcattcp.htm
8. WebBench 5.0, ftp://ftp.pcmag.com/benchmarks/webbench/
9. The Network Simulator-ns-2, http://www.isi.edu/nsnam/ns/
10. NIST Net, http://snad.ncsl.nist.gov/itg/nistnet/
11. Nessus, http://www.nessus.org/