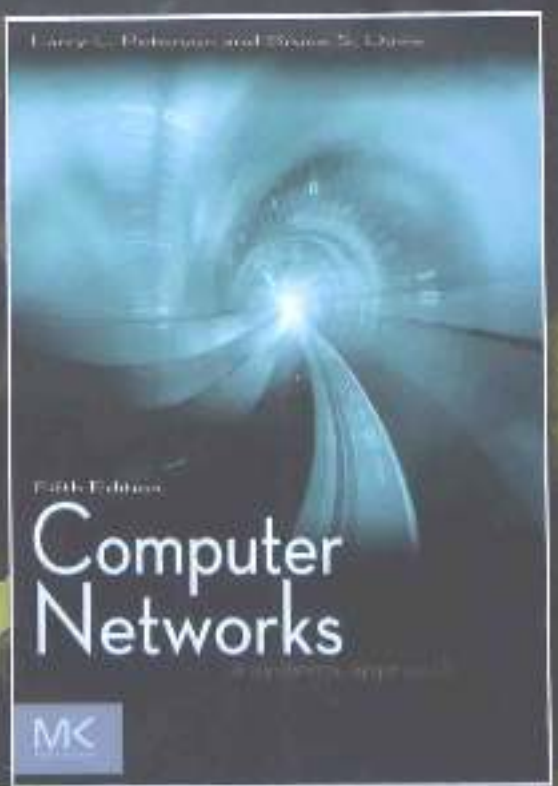
计 算 机 科 学 丛 书

原书第5版

计算机网络
系统方法

[美] 拉里 L. 彼得森（Larry L. Peterson） 普林斯顿大学
布鲁斯 S. 戴维（Bruce S. Davie） VMware公司 著
王勇 张龙飞 李明 薛静锋 等译

Computer Networks
A Systems Approach Fifth Edition



机械工业出版社
China Machine Press

计算机网络 系统方法 原书第5版

Computer Networks A Systems Approach Fifth Edition

本书以便于教学的方式阐述计算机网络领域的核心内容，或非常高兴地说，它比之前的所有版本都更棒！

—— David Clark 麻省理工学院

Peterson和Davie详尽且清晰地描述了各层网络协议，通过本书的参考资源诸局学生更深入且更全面地理解正在改变世界的网络技术。这本书的每个版本都与时俱进，日臻完善。

—— Jean Walrand 加州大学伯克利分校

这本杰出的教材通过精心的章节组织和"全景式"系统方法来展现计算机网络世界的奥秘，值得阅读！

—— Youshik Choi 伊利诺斯理工学院

本书是计算机网络方面的经典教科书，凝聚了两位顶尖网络专家几十年的理论研究、实践经验和大量第一手资料，自出版以来已经成为网络课程的主要教材之一，被哈佛大学、斯坦福大学、卡内基·梅隆大学、康奈尔大学、普林斯顿大学等众多名校采用。

本书采用"系统方法"来探讨计算机网络，把网络看作一个自相互关联的构造模块组成的系统，通过实际应用中的网络和协议设计实例，特别是因特网实例，讲解计算机网络的基本概念、协议和关键技术，为学生和专业人士理解现行的网络技术以及即将出现的新技术奠定了良好的理论基础。无论站在什么视角，无论是应用开发者、网络管理员还是网络设备或协议设计者，你都会对如何构建现代网络及其应用有"全景式"的理解。

本书特色

- 内容全面更新，包含更多对于网络技术人员和学生而言至关重要的主题，如P2P、无线技术、网络安全和视频技术。
- 更加关注应用层问题，应用层上的创新性研究和设计是当前关注的中心。
- 为了方便教学，本书配有丰富的教辅资源，包括教学课件、教材图片、实验手册和网络仿真软件OPNET，以及习题解答等。



计算机网络：系统方法（英文版·第5版）
书号：978-7-111-37720-7
定价：139.00元



计算机网络实验教程
书号：978-7-111-41585-5
定价：45.00元

本书译自原版*Computer Networks: A Systems Approach, Fifth Edition*
并由Elsevier授权出版

ELSEVIER 爱思唯尔

上架指导：计算机网络

ISBN 978-7-111-49907-7

9 787111 499077 >

定价：99.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上书店：www.china-pub.com
数字服务：www.hzmedia.com.cn

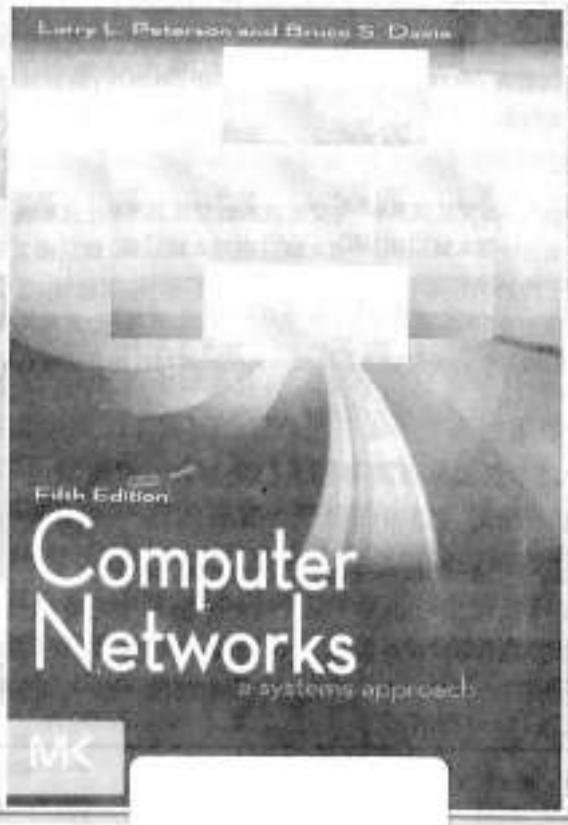
计 算 机 科 学 丛

原书第5版

计算机网络
系统方法

[美] 拉里 L. 彼得森（Larry L. Peterson） 普林斯顿大学
布鲁斯 S. 戴维（Bruce S. Davie） VMware公司 著
王勇 张龙飞 李明 薛静锋 等译

Computer Networks
A Systems Approach Fifth Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机网络：系统方法（原书第5版）/（美）彼得森（Peterson, L. L.），（美）戴维（Davie, B. S.）著；王勇等译．—北京：机械工业出版社，2015.4
（计算机科学丛书）
书名原文：Computer Networks: A Systems Approach, Fifth Edition

ISBN 978-7-111-49907-7

I. 计… II. ①彼… ②戴… ③王… III. 计算机网络 IV. TP393

中国版本图书馆CIP数据核字（2015）第072520号

本书版权登记号：图字：01-2012-0220

Computer Networks: A Systems Approach, Fifth Edition
Larry L. Peterson and Bruce S. Davie
ISBN: 978-0-12-385059-1
Copyright © 2012 by Elsevier Inc. All rights reserved.
Authorized Simplified Chinese translation edition published by the Proprietor.
Copyright © 2015 by Elsevier (Singapore) Pte Ltd. All rights reserved.
Printed in China by China Machine Press under special arrangement with Elsevier (Singapore) Pte Ltd. This edition is authorized for sale in China only, excluding Hong Kong SAR, Macau SAR and Taiwan. Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.

本书简体中文版由Elsevier(Singapore) Pte Ltd.授权机械工业出版社在中国大陆境内独家出版和发行。本版仅限在中国境内（不包括香港特别行政区、澳门特别行政区及台湾地区）出版及标价销售。未经许可之出口，视为违反著作权法，将受法律之制裁。

本书封底贴有Elsevier防伪标签，无标签者不得销售。

本书采用“系统方法”，将网络看作由相互关联的模块构成的交互式系统，通过丰富的因特网实例解析网络工作原理和应用设计方法。每章都以启发式问题开篇，章末辅以相关资源和习题，以完整的端到端系统为重点，不囿于传统分层模型。本书涵盖网络连接、交换和分组、TCP/IP协议、网络安全等基础理论和协议，第5版更新了无线技术、Web服务、域间路由和边界网关协议、多媒体应用协议等内容。本书可作为计算机相关专业研究生或高年级本科生的教材，也可供专业技术人员阅读参考。

出版发行：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：曲　熠　　责任校对：董纪丽
印　　刷：北京诚信伟业印刷有限公司　　版　　次：2015年6月第1版第1次印刷
开　　本：185mm×260mm　1/16　　印　　张：29.75
书　　号：ISBN 978-7-111-49907-7　　定　　价：99.00元

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

版权所有，侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光／邹晓东

文艺复兴以来，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垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出，独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅奠定了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

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

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

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专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重要帮助。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: A Systems Approach

由 Larry L. Peterson 和 Bruce S. Davie 两位顶尖网络专家撰写的《计算机网络：系统方法》已经成为计算机网络课程的主流教材，被哈佛大学、斯坦福大学、卡内基-梅隆大学、康奈尔大学、普林斯顿大学等国外众多名校采用。本书英文版已经出版了 5 个版本，为了方便国内高校和广大读者使用这本经典教材，我们翻译了最新版——第 5 版。

这一版对第 4 版做了重大的改进和提升：扩充和更新了无线技术（包括 802.11（Wi-Fi）、3G 和 4G 蜂窝无线技术）、拥塞控制机制（针对具有高带宽延迟积的网络和无线网络）、Web 服务（包括 SOAP 和 REST 体系结构）、域间路由和边界网关协议、多媒体应用协议（包括 VoIP 和视频流）等方面的材料；删减了异步传输模式（ATM）和令牌环等过时的内容；将交换、路由、可靠传输协议等材料分成“初级”和“高级”两部分，以适应不同层次读者的需要；更新了“相关主题”和“它们现在在哪”的内容，以反映现实世界中协议的最新发展状况。

作者强调网络现有工作方式的成因，采用“系统方法”将网络看作由相互关联的构造模块组成的系统（反对严格的分层），并引入丰富的因特网实例说明实际网络的设计。书中给出的程序代码不是基于某个特定的操作系统，而是经过重新改编以适应通用环境，说明网络软件是如何实现的，借此帮助读者了解网络的基础构件是如何结合在一起的。

本书是目前最新版的关于计算机网络的优秀教材，为学生和专业人士理解现行的网络技术以及即将出现的新技术奠定了良好的理论基础，是确保成功的课堂教学和高效的网络运行的重要资源。

本书由北京理工大学软件学院的王勇副教授主持翻译，参加翻译工作的有北京理工大学的王勇（第 1 和 8 章、前言、习题选答、术语）、薛静锋（第 2 章）、张龙飞（第 3、4、7 章）、陈琳（第 5、6 章）和公安部第三研究所的李明（第 9 章），另外，胡晶晶对本书的翻译也提供了很大帮助。

由于译者水平有限，书中可能有错误或不甚完善之处，读者如果有意见或建议，欢迎通过E-mail 与译者联系：wybit@sina.com。

译者

2015 年 3 月

为了与不断发展的计算机网络领域保持同步，我们对这本经典教材再一次进行了修订。虽然因特网和相关协议在当前所有网络互连中占主导地位，但我们看到因特网的支撑技术还在持续发展，例如，“二层”交换提供了丰富的功能和强大的网络管理工具。本书前一版将交换和路由分在两章中讲解，但是基于分层的描述并不总是表达教材要点的最好方法，因为所谓的交换和路由实际上起到类似且互补的作用。新版以一种综合方法来看待这些主题，这使得它们在功能上的相似点和不同点更加明显。路由的高级主题则移到下一章，可以根据课程的重点和层次略过该章。

我从来不喜欢用纯分层的方法讲授网络，正如第1版序言中所说（我们在本版中重印该序言只是为了好玩）。联网中的一些关键问题，包括安全和性能，不能通过将它们分配在一层中来解决——不存在一个“性能”层。这类主题不但关键，而且跨越多个层次，本书的组织结构仍然涵盖主题以及层次。组织结构反映了本书作为教材的丰富经验，同时也反映出一种方法倾向，即同时呈现基础知识和当前实践。

一些垂死的技术现在消失或萎缩了，包括令牌环（我曾经的最爱，但显然是它该离开的时候了）和ATM。新版意识到我们应该更多地关注应用设计，而不仅仅是分组转发。无线和移动技术也得到更多关注。

作者再次努力工作，以便写出能够以一种方便教学的方式阐述该领域重点内容的修订版。我很高兴地看到这一版比以前的更好。

David D. Clark

2010年11月

第1版序言

Computer Networks: A Systems Approach

面条型代码（spaghetti code）一词普遍被认为是一种侮辱。所有优秀的计算机科学家都推崇模块化，这是因为它能带来许多好处，最大的好处是解决问题而不必同时了解问题的所有环节。因此，本书在表达观点和编写代码时，模块化扮演着重要的角色。如果一本书的材料以模块化的方式有效地组织起来，那么读者就会很乐意从头读到尾。

在网络协议领域，采用国际标准形式，即ISO的七层网络协议参考模型，给出了“恰当的”模块化，这或许是独一无二的。该模型反映了一种分层的模块化方法，经常作为讨论协议的起点，以分析所讨论的设计是符合还是偏离这种模型。

看上去根据这种分层模型来组织一本网络书籍是显而易见的。但事实上这样做是有风险的，因为OSI模型在组织网络的核心概念时并不成功。一些基本需求，如可靠性、流量控制或安全，能够在多数（即使不是所有）OSI分层中解决。这种实际情况导致对参考模型理解的巨大混乱，有时甚至产生怀疑。实际上，如果严格按照层次模型来组织一本书，那么它就具有某些面条型代码的特点了。

本书是根据什么来组织的呢？Peterson和Davie遵循传统的分层模型，但他们并不否认该模型实际上无助于理解网络中的重大问题。相反，作者采用一种与层次无关的方法来组织基本概念的讨论。因此，在阅读本书后，读者将会理解流量控制、拥塞控制、可靠性增强、数据的表示以及同步等问题。同时，读者将了解在传统分层模型的某一层中如何解决这些问题。

这是一本与时俱进的书。本书着眼于当前使用的重要协议，尤其是有关因特网的协议。Peterson和Davie长期从事因特网相关工作，具有丰富的经验。因此，他们的书不仅从理论上而且从实践上揭示了协议设计的问题。本书也介绍许多最新的协议，读者可以从中获取最新的观点。更重要的是，对基本问题的讨论是源自问题的本质，而不受分层参考模型或当前协议细节的限制。在这一点上，本书既体现了时效性，又不受时间的限制。本书的独特之处在于它将有关的实际问题、实例和基本概念的解释有机地结合在一起。

David D. Clark

麻省理工学院

当本书第1版在1996年出版时，在因特网上购物还是很新奇的事情，如果一家公司用它的域名做广告，人们认为这是很超前的。那时，家庭接入因特网的主要方式是通过拨号调制解调器。今天，因特网商务已成为无法改变的现实，".com"股票已经历了一个完整的兴衰循环，无线网络无处不在，诸如智能手机和平板电脑等具有上网功能的新设备正以令人目眩的速度上市。关于因特网，似乎唯一可以预见的就是它会不断变化。

尽管有这么大的变化，但我们在第1版中提出的问题对于今天来说仍然是有效的：使因特网得以运行的基本概念和技术是什么？答案是，TCP/IP体系结构的大部分功能在今天仍然适用，这一点正像30多年前它的发明者所预见的那样。这并不是说因特网的体系结构枯燥乏味，而是完全相反。对于一个历经30多年，不但幸存下来，而且促进因特网这样快速地增长和变化的体系结构，将理解其设计原理作为学习网络的起点是很合适的。正如前几版一样，第5版将因特网体系结构中的"为什么"作为基础。

读者对象

我们的目的是把本书作为综合网络课程的教材，供研究生或高年级本科生使用。我们相信，本书对核心概念的专注不但对正在进行再培训以便完成网络相关工作的专业人员有吸引力，而且可以帮助网络从业人员理解他们每天接触的网络协议背后的"为什么"，并且着眼于网络的全貌。

根据我们的经验，第一次学习网络的学生和专业人员通常会把网络协议理解成一种从高层传到低层的命令，认为学习尽可能多的TLA（Three-Letter Acronyms，三个字母的缩写词）就可以了。事实上，协议是应用工程设计原理开发出来的复杂系统的构件。不仅如此，协议总是根据现实世界的经验不断地进行精炼、扩展和替换。基于这一点，本书的目标并不是单纯地介绍当前使用的协议，而是解释良好网络设计的基本原理。我们认为掌握这些基本原理是应对当前瞬息万变的网络领域的最好方法。

我们还意识到人们用很多不同的方法来看待网络。与我们写第1版时相比，大多数阅读本书的人都是具有丰富经验的网络用户。一些人希望成为网络产品或协议的设计者，其他人可能对管理网络感兴趣，而越来越多的读者正在或将要成为互联网设备的应用开发者。未来产品和协议的设计者历来是我们关注的重点，现在依然如此，但在本版中，我们也试图关注网络管理员和应用开发者。

第5版中的变化

尽管我们关注网络的基本原则，但我们还是要使用正在运行的因特网中的例子来阐述这些原则。因此，我们补充了相当多的新资料来跟踪近期网络技术的重要发展。我们同时对已有资料做了删除、重新组织和改写，以反映过去10年发生的变化。

也许自从编写第1版以来我们所觉察到的最重要的变化，就是现在几乎每一位读者都

对诸如万维网和电子邮件这样的网络化应用有了一定的了解。因此，我们从第 1 章开始就加大了对应用的侧重。我们把应用当作学习网络技术的动机，并得出一组需求。有用的网络只有满足这些需求才能在全球范围内支持当前和未来的各种应用。然而，我们保留了先前版本解决问题的方法，即从主机的互联问题开始，逐层向上讨论，最后对应用层的问题进行详细的考察。我们认为从各种应用及其需求起步对于在本书所覆盖的各主题间建立联系是很重要的。同时，我们感到对于诸如应用层协议和传输层协议等高层协议的问题，只有在明白主机互连和分组交换这样的基本问题之后才能很好地理解。也就是说，我们使得以一种更加自顶向下的方式组织材料成为可能，详情如下所述。

正如前几版，我们增大了对重要新主题的覆盖面，并使其他主题紧跟潮流。新版中主要的新主题或有实质性改动的主题包括：

- 更新了无线技术的内容，特别是 802.11（Wi-Fi）的各种相关内容以及包括第三代移动通信标准（3G）和正在逐步普及的 4G 标准在内的蜂窝无线技术。
- 更新了拥塞控制机制的内容，特别是具有高延迟带宽积的网络和无线网络。
- 更新了 Web 服务的材料，包括 SOAP 和 REST（Representational State Transfer，表述性状态转移）体系结构。
- 扩展和更新了域间路由和边界网关协议（BGP）的内容。
- 扩展了多媒体应用协议的内容，如基于 IP 的语音（VoIP）和视频流。

我们还减少了目前已不太相关的主题。在本版中，被转移到“历史性”类别中的内容包括异步传输模式（ATM）和令牌环。

新版最大的变化之一是将材料分成“初级”和“高级”两部分。我们希望不了解网络技术和协议的人更容易接受本书，同时也不丢弃高层次读者需要的高级内容。这种变化产生的最明显的影响是现在第 3 章包含交换、路由和网络互连的基础内容，而第 4 章包含更高级的路由主题，如 BGP、IPv6 和多播。与此类似，传输协议基础放在第 5 章，而更高级的材料（如 TCP 拥塞控制算法）出现在第 6 章。我们相信，这种变化会使得不熟悉该领域的读者掌握重要的基本概念，并且不会因为复杂主题而不堪重负。

就像上一版一样，我们加入了一些“它们现在在哪”的讨论。这些简短的讨论在本版中经过更新，重点关注现实世界中协议的成败。有时，它们描述了一个已经被大多数人忽视但实际上取得了名不见经传的成功的协议；有时，它们跟踪某个经过很长时间却失败了的协议的命运。这些内容的目的是通过展示技术在竞争的网络世界中的表现来把材料关联起来。

系统方法

对于像计算机网络这样动态的和不断变化的领域来说，一本教材能提供的最重要的东西是洞察能力，以便分清什么是重要的，什么是不重要的，什么是长久的，什么是肤浅的。根据我们在网络新技术研究、为本科生与研究生讲授网络最新趋势以及把先进的网络产品投放市场等方面超过 25 年的经验，我们提炼了自己的观点——系统方法，它是本书的精髓。系统方法有以下含义：

- 基本原则。与其将现成的网络产品作为准则，不如从基本原则开始了解当今网络技术的发展过程。这能让我们解释网络为什么像现在这样设计。根据我们的经验，

一旦理解了基本概念，那么理解遇到的任何新协议都将变得相对容易。

- 非分层主义者。虽然材料是围绕传统网络层次松散地组织起来的，从底层开始沿协议栈向上展开，但是我们并不采用严格分层的方法。许多主题涉及上下多个层，如拥塞控制和安全，所以我们不用传统的分层模型讨论它们。与此类似，路由器和交换机有很多共同点（并且经常结合起来组成单个产品），因此我们在同一章中讨论它们。简言之，我们相信分层是很好的辅助，但不必受它的限制，采用端到端的观点常常更有用。
- 现实世界的例子。与其抽象地解释协议如何工作，不如使用当今最重要的协议具体说明网络是如何工作的。许多协议都是源自 TCP/IP 因特网的，这就允许我们在讨论中借鉴实际经验。
- 软件。虽然底层网络是由可以从计算机销售商那里购买的硬件以及可以从电话公司租用的通信服务构建而成的，但是，只有软件才能使网络提供新服务，并迅速适应环境变化。这就是我们为什么强调网络软件如何实现，而不是只停留在描述所涉及的抽象算法上。我们还会给出实际运行的协议栈的代码段，以说明如何实现某些协议和算法。
- 以端到端为重点。网络是由许多组件构成的，虽然在解决具体问题时有必要忽略一些不感兴趣的元素，但是理解如何将这些组件组合成具有一定功能的网络是非常重要的。所以我们花大量的篇幅解释网络完整的端到端行为，而不只是单个组件，以便理解一个完整的网络是如何运行的，包括从应用到硬件的所有方面。
- 性能。系统方法包含通过实验研究性能问题，然后使用收集到的数据对各种设计选项进行定量分析并指导优化实现。这种对实验分析的强调贯穿全书。
- 设计原则。网络很像其他计算机系统，如操作系统、处理器体系结构、分布式和并行系统等。它们都很大，也很复杂。为了处理这种复杂性，系统设计者常常提出一组设计原则。在书中介绍这些设计原则时，我们会予以强调，并用计算机网络中的例子进行解释说明。

教学特色

第 5 版保留了以前版本中关键的教学特色，我们鼓励读者利用这些特色：

- 问题。在每一章的开始，我们通过一个问题指明网络设计中必须要解决的后续问题。这段描述将引出并推动本章要探讨的问题。
- 相关主题。在本书中，“相关主题”内容描述了正在讨论的主题或介绍相关高级主题，这些多为关于网络的逸闻趣事。
- 它们现在在哪？这部分内容跟踪协议在实际部署情况下的成败。
- 结论。这些段落归纳了在讨论中得出的重要结论，如广泛适用的系统设计原则。
- 实际协议。虽然本书的重点是核心概念而不是现有的协议规程，但实际协议常被用于说明大部分的重要观点。因此本书可以作为很多协议的参考源。为了帮助你找到这些协议的描述，相关章节标题中用括号括起来的内容指明在本节中描述的协议。例如，5.2 节描述可靠的端到端协议的原理，它提供对 TCP 的详细描述，

以作为这类协议的典型实例。

- *接下来会发生什么？*我们以研究团体、业界或整个社会正在探讨的一个重要问题作为每一章主体部分的结尾。我们发现讨论这些前瞻性问题有助于将网络主题变得更加相互关联且令人兴奋。
- *扩展阅读。*在每一章结尾有精选的扩展阅读列表。每个列表一般包含有关刚刚讨论的主题的重要论文。我们强烈推荐高级读者（如研究生）研究扩展阅读中的论文，以便补充各章所讲的材料。

本书结构和教学方法

本书按如下方式进行组织：

- 第 1 章介绍全书使用的核心概念。从各种应用出发，讨论网络体系结构的组成，介绍协议的实现问题，并定义通常能够促进网络设计的定量性能度量。
- 第 2 章综述用户与因特网等大型网络相连的多种方法，介绍链路的概念，描述所有链路层协议必须解决的许多问题，包括编码、组帧和差错检测。当前最重要的链路技术以太网和无线也在本章中讲述。
- 第 3 章从虚电路和数据报模型开始介绍交换和路由的基本概念，包括网桥和局域网交换，接下来是对网络互连的介绍，包括网际协议（Internet Protocol，IP）和路由协议。本章的结尾讨论了一系列基于硬件和软件构造路由器和交换机的方法。
- 第 4 章涵盖高级网络互连主题。包括多区域路由协议、域间路由和 BGP、IPv6、多协议标记交换（MultiProtocol Label Switching，MPLS）和多播。
- 第 5 章转移到传输层，详细描述因特网中用于创建客户/服务器应用的传输控制协议（Transmission Control Protocol，TCP）和远程过程调用（Remote Procedure Call，RPC），同时介绍支持多媒体应用的实时传输协议（Real-time Transport Protocol，RTP）。
- 第 6 章讨论拥塞控制和资源分配。本章的问题贯穿了链路层（第 2 章）、网络层（第 3、4 章）和传输层（第 5 章）。特别注意，本章还描述拥塞控制如何在 TCP 上工作，并且介绍在 IP 中用于提供服务质量的机制。
- 第 7 章考虑通过网络发送的数据。这涉及表示格式和数据压缩问题。XML 也在本章讲述，压缩部分包括对 MPEG 视频压缩和 MP3 音频压缩工作原理的解释。
- 第 8 章讨论网络安全，首先简要介绍密码学工具和密钥分发问题，然后讨论几种使用公钥和私钥的认证技术。本章的重点是使用诸如良好隐私（Pretty Good Privacy，PGP）、安全外壳（Secure Shell，SSH）和 IP 安全体系结构（IPSEC）构建安全系统。防火墙也在本章讨论。
- 第 9 章描述典型的网络应用实例和它们使用的协议，包括像电子邮件和万维网这样的传统应用、像 IP 电话和视频流这样的多媒体应用，以及像对等文件共享和内容分发网络这样的覆盖网络。本章还描述基础设施服务——域名系统（Domain Name System，DNS）和网络管理。用于开发新应用协议的 Web 服务体系也在本章介绍。

对于本科生课程，很可能需要追加课时来帮助学生理解第 1 章中的介绍性材料，而放弃第 4 章以及第 6～8 章中涵盖的高级主题。然后在第 9 章回到网络应用的一般主题。本科生课程可以合理地跳过更高级的部分（如 5.3 节、9.3.1 节、9.3.2 节和 9.2.2 节）。

相比之下，研究生课程的教师应该能够只用 1 次或 2 次课讲完第 1 章——让学生自己更认真地学习这些材料，从而可以空出更多的课时更深入地讲授第 4 章及后续章节。

对于自学本书的读者，我们相信所选的主题涵盖了计算机网络的核心内容，因此建议从前向后顺序阅读。另外，我们提供了详细的参考文献目录，帮助读者找到与感兴趣的特定领域相关的补充资料。我们还提供了习题选答。

本书用独特的方法讨论拥塞控制，即把有关拥塞控制和资源分配的所有主题集中到第 6 章。这样做是因为拥塞控制问题不能在任何一层单独解决，我们希望读者能够同时考虑各种设计选择（这与我们的观点是一致的，即严格的分层常常模糊了重要的设计权衡）。然而，用更传统的方法来讲解拥塞控制也是可能的，即在学习第 3 章时参考 6.2 节的内容，以及在学习第 5 章时参考 6.3 节的内容。

自顶向下的学习路线

因为现在学习网络课程的大部分学生都熟悉网络应用，因此一些课程从应用开始讲解。虽然本书的确在第 1 章提纲挈领地介绍了应用，但直到第 9 章才详细讨论应用层的问题。考虑到有些教授或读者希望使用更加自顶向下的方法，我们建议采用如下可能的方式使用本书的材料。

- 第 1 章。本章描述应用及其需求，为教材的其他内容做好铺垫。
- 第 9 章。传统应用（9.1 节）和多媒体应用（9.2 节）部分以读者熟悉的应用作为例子来引入网络协议的概念。9.3.1 节（DNS）也应该包括在内。
- 接下来应该讲授 7.2 节，解释多媒体应用产生的数据是如何编码和压缩的。
- 第 5 章。此时可以讲解传输协议基础，解释应用层协议产生的数据如何进行可靠的跨网络传输。
- 第 3 章。可以将交换、网络互连和路由理解为提供传输协议运行所依赖的基础设施。
- 第 2 章。最后，可以讲解数据到底是如何在像以太网链路和无线链路这样的物理介质上进行编码和传输的。

显然，我们在上述安排中略去了很多章节。对于更高级的课程或更深入的自学，可以在后面追加像资源分配（第 6 章）和安全（第 8 章）等主题以及第 4 章中的高级主题。安全几乎可以单独讲解，但所有高级主题只有在第 3 章和第 5 章中分别讲解了 IP 和 TCP 之后才有意义。

注意，访问 booksite.elsevier.com/9780123850591/lec.php，除了按照本书的顺序制作的一套课件以外，还包括按照上述自顶向下顺序制作的一套课件。

习题

每一版中都对习题进行了重大改进。在第 2 版中，我们增加了习题的数量，并且根据

课堂测验大大提高了习题的质量。在第 3 版中，我们对习题做了两方面的重大修改，包括：

- 对那些我们认为很有挑战性或需要本书以外知识（如概率知识）的习题，加上★标记以表明它们的难度更高。
- 我们在每一章都增加了一些额外的典型习题并在书后提供答案。这些习题用√标记，目的是为完成其他习题提供一些帮助。

在这一版中增加了新习题以反映更新的内容，现有习题可分为如下几类：

- 分析性习题，要求学生做简单的代数计算，展示他们对基本关系的理解。
- 设计习题，要求学生针对各种情况设计并评估协议。
- 实践习题，要求学生写少量代码以测试一个想法或使用现成的网络工具进行实验。
- 文献研究习题，让学生更深入地了解某个特定主题。

此外，正如下面所详细描述的那样，基于套接字的编程习题以及仿真实验可以在线获得。

补充材料和在线资源

为了方便教学，我们准备了教学手册，其中包括一些习题的解答。可以向出版社索取该手册。[⊖]

其他辅助材料，包括课件、教材图片、基于套接字的编程习题、测验题以及编程习题可以在 Morgan Kaufmann 的网站上找到，网址为 booksite.elsevier.com/9780123850591/index.php。

最后，与第 4 版一样，本书配有一套实验。这些实验由马萨诸塞大学达特茅斯分校的 Emad Aboelela 教授开发，通过仿真实验来探讨本书中协议的行为、可扩展性和性能。讨论实验相关材料的章节均带有标记。仿真使用 OPNET 仿真工具集。[⊜]

致谢

如果没有众多朋友的帮助，本书是不可能问世的。我们非常感谢所有为改进本书做出贡献的人。然而，在致谢之前应该说明的是，我们已经尽力改正审阅人指出的错误并且尽量准确地描述同事们向我们解释的协议和机制。如果还有什么错误，那是我们自己的责任。如果你发现任何错误，请发电子邮件给我们的出版商 Morgan Kaufmann，地址是 netbugsPD5e@mkp.com，我们将在本书再次印刷时进行改正。

首先，我们衷心感谢审阅过全部或部分书稿的人。除了审阅过以前版本的人，我们还想感谢 Peter Dordal、Stefano Basagni、Yonshik Choi、Wenbing Zhao、Sarvesh Kulkar-

---

⊖ 关于本书教辅资源，使用教材的老师需通过爱思唯尔的教材网站（www.textbooks.elsevier.com）注册并通过审批后才能获取相关资源。具体方法如下：在 www.textbooks.elsevier.com 教材网站查找到该书后，点击“instructor manual”便可申请查看该教师手册。有任何问题，请致电 010-85208853。——编辑注

⊜ 关于本书配套的《计算机网络实验教程》（中文版），欢迎访问华章网站（www.hzbook.com/books/6937.html）或扫描右侧二维码了解详细信息。全部实验在各章节的对应位置添加了注释。——编辑注



ni、James Menth 和 John Doyle（以及一个匿名审阅人）对全书的审阅。还要感谢 Dina Katabi 和 Hari Balakrishnan 对各章节的审阅。我们也要感谢所有提供反馈意见和信息来帮助我们写作该版的人。

普林斯顿网络系统组的几位成员向本书贡献了他们的想法、示例、修订、数据和代码。我们特别感谢 Andy Bavier、Tammo Spalink、Mike Wawrzoniak、Stephen Soltesz 和 KyoungSoo Park。还要感谢 Shankar M. Banik 为本书制作了两套清晰易懂的课件。

最后，我们衷心地感谢丛书编辑 David Clark，以及 Morgan Kaufmann 出版公司在本书编写期间帮助过我们的所有人。还要特别感谢我们原来的责任编辑 Jennifer Young、本版的编辑 Rick Adams、开发编辑 Nate McFadden、助理编辑 David Bevans 以及制作编辑 Paul Gottehrer。同时要感谢 MKP 的出版人 Laura Colantoni，她的领导激励着我们专注于本修订版。

本书赞誉

Computer Networks, A Systems Approach

我知道并使用这本书已经多年，无论是作为计算机网络教材还是网络技术人员的参考书，本书都非常有价值。第5版保持了之前版本的核心价值，并且通过最新技术和需求对网络协议进行了非常清晰的解释。除了描述过去和当前网络的细节，本书还成功激发了读者的好奇心，并有望促进对未来网络的新研究。

——Stefano Basagni，美国东北大学

本书非常适合认真学习计算机网络的学生，当然，它也适合休闲阅读。作者热情高涨，对网络中的有趣主题有着全面而及时的把握。本书不仅解释各种协议如何工作，而且说明它们为什么这样工作，以及某些协议为什么如此重要和有趣。本书还涉及历史背景介绍，从正文部分到“它们现在在哪”再到每章“扩展阅读”中介绍的文献，这些内容为读者提供了“知其所以然”的视角。总之，本书对网络的讲解清晰且富有文化内涵。

——Peter Dordal，芝加哥洛约拉大学

我在一门面向高年级本科生和一年级硕士生的通信网络基础课程中使用本书已经五年多了，其各个版本一直保持着自己的优势，即不仅描述协议“如何工作”，还描述“为什么这样工作”，并重点解释“为什么不这样工作”，这是一本培养工程直觉的书，在如今这个技术快速变化的时代，培养学生在设计或选择下一代系统时做出明智决定的能力是至关重要的。

——Roch Guerin，宾夕法尼亚大学

这是一本优秀的计算机网络教材，内容清晰、全面、包含大量实例。作者将网络分解成简单的、可管理的概念，同时不影响技术的严谨性。本书寻求网络体系结构设计的基本原则和构建在其上的应用程序之间的平衡，对于高级网络课程的本科生、研究生和教师来说价值连城。

——Arvind Krishnamurthy，华盛顿大学

本书是深入理解计算机网络的最好教材之一。第5版涵盖了该领域的最新进展。第1章系统介绍网络的基本组件，包括硬件和软件；最后一章则以应用为主题将所有概念综合起来。部分可选的高级主题单独成章。每章最后包含一组不同难度的习题，帮助学生掌握相关知识。

——Karkal Prabhu，德雷克塞尔大学

作者对各层互联网协议给出了详细而明确的描述。其中有很多参考资源，可帮助学生充分理解正在改变世界的网络技术。这本书的每个版本都在不断进步。

——Jean Walrand，加州大学伯克利分校

出版者的话
译者序
序言
第1版序言
前言
本书特色

第1章 基础 …… 1
问题：建造一个网络 …… 1
1.1 应用 …… 2
1.2 需求 …… 4
1.2.1 视角 …… 4
1.2.2 可扩展的连通性 …… 4
1.2.3 高性价比的资源共享 …… 7
1.2.4 支持通用服务 …… 10
1.2.5 可管理性 …… 13
1.3 网络体系结构 …… 14
1.3.1 分层和协议 …… 14
1.3.2 因特网体系结构 …… 19
1.4 实现网络软件 …… 20
1.4.1 应用程序编程接口（套接字） …… 21
1.4.2 应用实例 …… 23
1.5 性能 …… 25
1.5.1 带宽和时延 …… 25
1.5.2 延迟带宽积 …… 28
1.5.3 高速网络 …… 29
1.5.4 应用程序性能需求 …… 31
1.6 小结 …… 32
接下来会发生什么：云计算 …… 32
扩展阅读 …… 33
习题 …… 35

第2章 开始连接 …… 39
问题：连接到网络 …… 39
2.1 连接概览 …… 39
2.2 编码（NRZ、NRZI、曼彻斯特、4B/5B） …… 43
2.3 组帧 …… 45
2.3.1 面向字节的协议（BISYNC、PPP、DDCMP） …… 45
2.3.2 面向比特的协议（HDLC） … 47
2.3.3 基于时钟的组帧（SONET） … 48
2.4 差错检测 …… 50
2.4.1 二维奇偶校验 …… 51
2.4.2 因特网校验和算法 …… 51
2.4.3 循环冗余校验 …… 53
2.5 可靠传输 …… 55
2.5.1 停止-等待 …… 56
2.5.2 滑动窗口 …… 58
2.5.3 并发逻辑信道 …… 64
2.6 以太网和多路访问网络（802.3） …… 65
2.6.1 物理特性 …… 65
2.6.2 访问协议 …… 66
2.6.3 以太网使用经验 …… 69
2.7 无线 …… 69
2.7.1 802.11/Wi-Fi …… 73
2.7.2 蓝牙（802.15.1） …… 77
2.7.3 蜂窝电话技术 …… 78
2.8 小结 …… 80
接下来会发生什么："物联网" …… 81
扩展阅读 …… 82
习题 …… 83

第3章 网络互联 …… 89
问题：并不是所有网络都是直接相连的 …… 89
3.1 交换和桥接 …… 89
3.1.1 数据报 …… 91
3.1.2 虚电路交换 …… 92
3.1.3 源路由 …… 99
3.1.4 网桥和局域网交换机 …… 101
3.2 互联网基础（IP） …… 107
3.2.1 什么是互联网？ …… 107

3.2.2 服务模型 …… 109
3.2.3 全局地址 …… 113
3.2.4 IP 数据报转发 …… 114
3.2.5 子网划分和无类地址 …… 117
3.2.6 地址转换（ARP） …… 121
3.2.7 主机配置（DHCP） …… 123
3.2.8 差错报告（ICMP） …… 125
3.2.9 虚拟网络和隧道 …… 125
3.3 路由 …… 127
3.3.1 用图表示网络 …… 128
3.3.2 距离向量（RIP） …… 129
3.3.3 链路状态（OSPF） …… 134
3.3.4 度量 …… 140
3.4 实现和性能 …… 142
3.4.1 交换基础 …… 142
3.4.2 端口 …… 144
3.4.3 交换结构 …… 146
3.4.4 路由器的实现 …… 148
3.5 小结 …… 150
接下来会发生什么：未来因特网 …… 150
扩展阅读 …… 151
习题 …… 152

第 4 章 高级网络互联 …… 161
问题：扩展到数十亿节点 …… 161
4.1 全球互联网 …… 161
4.1.1 路由区 …… 162
4.1.2 域间路由（BGP） …… 164
4.1.3 IP 版本 6（IPv6） …… 170
4.2 多播 …… 178
4.2.1 多播地址 …… 179
4.2.2 多播路由（DVMRP、PIM、MSDP） …… 180
4.3 多协议标签交换（MPLS） …… 188
4.3.1 基于目的地的转发 …… 188
4.3.2 显式路由 …… 192
4.3.3 虚拟专用网和隧道 …… 194
4.4 移动设备之间的路由 …… 197
4.4.1 移动网络的挑战 …… 197
4.4.2 路由到移动主机（移动 IP） … 198
4.5 小结 …… 202
接下来会发生什么：部署 IPv6 …… 202
扩展阅读 …… 203
习题 …… 204

第 5 章 端到端协议 …… 208
问题：进程间如何通信 …… 208
5.1 简单多路分解（UDP） …… 209
5.2 可靠字节流（TCP） …… 210
5.2.1 端到端问题 …… 211
5.2.2 报文段格式 …… 212
5.2.3 连接建立与终止 …… 214
5.2.4 滑动窗口再讨论 …… 217
5.2.5 触发传输 …… 221
5.2.6 自适应重传 …… 223
5.2.7 记录边界 …… 225
5.2.8 TCP 扩展 …… 226
5.2.9 性能 …… 227
5.2.10 其他设计选择 …… 228
5.3 远程过程调用 …… 230
5.3.1 RPC 基础 …… 230
5.3.2 RPC 实现（SunRPC、DCE） … 232
5.4 实时应用程序传输（RTP） …… 239
5.4.1 需求 …… 240
5.4.2 RTP 设计 …… 241
5.4.3 控制协议 …… 243
5.5 小结 …… 245
接下来会发生什么：传输协议多样性 … 246
扩展阅读 …… 246
习题 …… 247

第 6 章 拥塞控制与资源分配 …… 254
问题：分配资源 …… 254
6.1 资源分配问题 …… 254
6.1.1 网络模型 …… 255
6.1.2 分类方法 …… 257
6.1.3 评价标准 …… 259
6.2 排队规则 …… 260
6.2.1 FIFO …… 261
6.2.2 公平排队 …… 262
6.3 TCP 拥塞控制 …… 264
6.3.1 加性增/乘性减 …… 265
6.3.2 慢启动 …… 267
6.3.3 快速重传和快速恢复 …… 270

6.4 拥塞避免机制 …… 272
6.4.1 DECbit …… 272
6.4.2 随机早期检测（RED） … 273
6.4.3 基于源的拥塞避免 …… 277
6.5 服务质量 …… 281
6.5.1 应用需求 …… 281
6.5.2 综合服务（RSVP） …… 284
6.5.3 区分服务（EF、AF） …… 291
6.5.4 基于等式的拥塞控制 …… 295
6.6 小结 …… 296
接下来会发生什么：网络重构 …… 297
扩展阅读 …… 297
习题 …… 299

第7章 端到端数据 …… 305
问题：我们用数据做什么？ …… 305
7.1 表示格式化 …… 306
7.1.1 分类方法 …… 307
7.1.2 例子（XDR、ASN.1、NDR） …… 309
7.1.3 标记语言（XML） …… 312
7.2 多媒体数据 …… 315
7.2.1 无损压缩技术 …… 316
7.2.2 图像表示和压缩（GIF、JPEG） …… 318
7.2.3 视频压缩（MPEG） …… 322
7.2.4 在网上传输 MPEG …… 325
7.2.5 音频压缩（MP3） …… 328
7.3 小结 …… 329
接下来会发生什么：无处不在的视频 … 329
扩展阅读 …… 330
习题 …… 330

第8章 网络安全 …… 334
问题：安全攻击 …… 334
8.1 密码基础 …… 335
8.1.1 密码原理 …… 335
8.1.2 对称密钥密码 …… 337
8.1.3 公钥密码 …… 338
8.1.4 认证码 …… 339
8.2 密钥预分发 …… 342
8.2.1 公钥预分发 …… 342
8.2.2 对称密钥预分发 …… 345
8.3 认证协议 …… 346
8.3.1 原始性和时效性技术 …… 346
8.3.2 公钥认证协议 …… 347
8.3.3 对称密钥认证协议 …… 348
8.3.4 Diffie-Hellman 密钥协商 … 351
8.4 系统实例 …… 352
8.4.1 良好隐私（PGP） …… 352
8.4.2 安全外壳（SSH） …… 354
8.4.3 传输层安全（TLS、SSL、HTTPS） …… 356
8.4.4 IP 安全（IPsec） …… 358
8.4.5 无线安全（802.11i） …… 360
8.5 防火墙 …… 362
8.6 小结 …… 365
接下来会发生什么：面对安全 …… 366
扩展阅读 …… 366
习题 …… 367

第9章 应用 …… 370
问题：应用需要自己的协议 …… 370
9.1 传统应用 …… 370
9.1.1 电子邮件（SMTP、MIME、IMAP） …… 371
9.1.2 万维网（HTTP） …… 376
9.1.3 Web 服务 …… 382
9.2 多媒体应用 …… 387
9.2.1 会话控制和呼叫控制（SDP、SIP、H.323） …… 388
9.2.2 多媒体应用的资源分配 … 393
9.3 基础设施服务 …… 397
9.3.1 名字服务（DNS） …… 397
9.3.2 网络管理（SNMP） …… 403
9.4 覆盖网络 …… 405
9.4.1 路由覆盖 …… 406
9.4.2 对等网 …… 411
9.4.3 内容分发网络 …… 418
9.5 小结 …… 422
接下来会发生什么：新的网络体系结构 …… 422
扩展阅读 …… 423
习题 …… 424

习题选答 …… 428
术语 …… 437
参考文献 …… 449

基　础

我必须创造一个体系，否则我将沦为别人体系的附庸；我不要推理和比较，我的工作是创造。

——威廉·布莱克

问题：建造一个网络

假设我们要建造一个计算机网络，它有发展到全球性规模的潜力，并且能够支持各种各样的应用，如远程会议、视频点播、电子商务、分布式计算和数字化图书馆等。那么要采用什么现有技术作为基础构件，以及使用何种软件体系结构才能把这些构件集成为一个有效的通信服务？本书最主要的目标就是回答这个问题，首先描述可用的构件，然后说明如何使用它们从头开始建造一个网络。

在理解如何设计计算机网络之前，我们首先应该在“什么是计算机网络”这一问题上达成共识。曾经有一段时期，网络（network）一词是指用于将哑终端连接到大型机的串行线集合。其他重要的网络包括语音电话网络和用于传播视频信号的有线电视网络。这些网络的主要共同点是专门处理某种特定类型的数据（按键、音频或视频），并且通常连接到专用设备（终端、手持接收器和电视机）。

计算机网络与其他类型的网络有什么区别？也许计算机网络最重要的特征就是通用性。计算机网络主要由通用可编程硬件来构建，并且不需要为诸如打电话或传输电视信号那样的特定应用做任何优化。相反，计算机网络能够传输多种不同类型的数据，并且支持各种各样不断增加的应用。今天的计算机网络正越来越多地接管以前由单一用途网络执行的功能。本章考察计算机网络的一些典型应用，然后讨论网络设计者为支持这些应用必须要了解的内容。

一旦我们清楚这些需求，接下来该怎么做呢？幸运的是，我们不是要建造第一个网络。其他人已经先于我们完成了这项任务，其中最著名的是因特网研究人员。我们将利用在因特网构建中得到的丰富经验来指导我们的设计。这些经验体现在网络体系结构（network architecture）中，网络体系结构指明可用的软硬件构件，并且说明如何将它们组织起来构成一个完整的网络系统。

除了理解如何建造网络，理解如何使用或管理网络、如何开发网络应用也很重要。现在，大部分家庭、办公室或车里都有计算机网络，所以使用网络不再是少数专家的事情。同时，随着智能手机等可编程联网设备的激增，将会有更多的人从事网络应用开发。因此，我们需要从多个视角来分析网络：网络建造者、网络运营者、应用开发者。

为了理解如何建造、运营网络和进行网络编程，本章将阐述四项内容。首先，揭示不同应用和不同人群对网络的需求；第二，引入网络体系结构的概念，它是本书后续部分的基础；第三，介绍实现计算机网络的几个关键因素；最后，介绍衡量计算机网络性能的关键度量标准。

1.1 应用

许多人是通过各种应用来了解因特网的，例如，万维网、电子邮件、在线社交网络、音频流和视频流、即时通信和文件共享。换言之，我们是作为网络用户（user）来访问因特网的。因特网用户是访问因特网的最大群体，但还有其他几个重要的群体。有一群人创造（create）应用，这个群体近几年扩展明显，原因是强大的编程平台和智能手机等新设备为快速开发应用并将其投放更大的市场创造了新机会。还有一群人运营（operate）或管理（manage）网络，这多半是幕后工作，但是非常关键且通常很复杂。随着家庭网络的普及，越来越多的人正在变成某种意义上的网络运营者。最后，还有那些设计（design）和构建（build）组成因特网的设备和协议的人。最后这个群体是网络教科书（如本书）的传统目标读者，本书将继续将这个群体作为主要关注对象。然而，在本书中，我们也考虑应用开发者和网络运营者的视角。考虑这些视角能够使我们更好地理解网络需要满足的不同需求。应用开发者如果能够理解底层技术的工作原理及其与应用的交互方式，他们就能开发出更好的应用。所以，在理解如何建造网络之前，先来仔细看看当今网络所支持的应用类型。

应用分类

万维网是一种将因特网从大多由科学家和工程师所使用的有些晦涩的工具跃升为当今主流现象的因特网应用。万维网已经变成了一个如此强大的平台，以至于很多人都将它与因特网混淆了（就像“Interwebs”），而且，将万维网看作一个单一应用有点夸张。

万维网采用基本形式提供了一个直观且简单的界面。用户浏览有很多文本和图形对象的网页时，单击他们想进一步了解的对象，就会弹出相应的新网页。大多数人都明白在这个应用的背后，是网页上的每个可选对象都绑定了一个指向下一个页面的标识符或要浏览的对象。这个标识符称为统一资源定位符（Uniform Resource Locator，URL），它提供了一种标识所有通过浏览器能浏览到的对象的方法。举个例子：http://www.cs.princeton.edu/~llp/index.html。这是本书一名作者的信息页面的URL：字符串http表明要下载页面应使用超文本传输协议（Hyper Text Transfer Protocol，HTTP），www.cs.princeton.edu是提供该网页的计算机的名字，而~llp/index.html唯一标识Larry在该网站的主页。

然而，大多数用户并不清楚，在单击这样一个URL后，在因特网上可能需要交换多达十几条消息，如果是有大量内嵌对象的复杂网页，则需要交换更多的消息。这些消息中，6条消息用于将服务器名（www.cs.princeton.edu）翻译成它所对应的网际协议（Internet Protocol，IP）地址（128.112.136.35），3条消息用于建立从浏览器到服务器之间的传输控制协议（Transmission Control Protocol，TCP）连接，4条消息用于浏览器发送HTTP的“GET”请求，并使服务器回送被请求的页面（以及双方确认收到消息），还有4条消息用于拆除TCP连接。当然，这不包含因特网节点全天交换的数以百万计的消息，这些消息只是让节点相互知道彼此的存在，并准备提供网页服务，把机器名翻译成网络地址，并将各种消息向它们的最终目的地转发。

另一类普及的因特网应用是音频流和视频流的传播，视频点播和因特网收音机这类服务就使用了这种技术。虽然我们经常从网站上发起流会话，但传播音频和视频与获取由文本及图片组成的网页存在一些重要的区别。例如，用户一般不想下载完整个视频文件（这个过程需要几分钟到几小时）才开始观看。以流的方式传输音频和视频意味着以一种更有

时效性的方式从发送方向接收方传输消息，接收方可以在收到消息后立即播放视频或音频。

注意，流式应用与传统的网页文本或静态图片传输的区别是，人们以连续的方式欣赏音频流和视频流，跳音或视频停滞等间断是不能接受的。相比之下，文本网页可以以位或块为单位来传输和阅读。这种区别会影响网络对不同类型应用的支持方式。

一种略有不同的应用类型是实时音频和视频。这些应用比流应用有更严格的时间约束。当使用 IP 电话应用（如 Skype）或视频会议应用时，参与者之间的交互必须是实时的。一端的用户做一个动作，这个动作必须尽快显示在另一端。当一个人试图打断另一个人时，被打断的人必须尽快[⊖]听到并决定是否允许被打断，或者继续说话而不理会打断。在这种环境下，太长的延迟会导致系统无法使用。与视频点播相比，如果从用户打开视频到第一幅图像被显示出来用了几秒钟，那么这个服务仍可接受。此外，交互式应用通常需要有双向流动的音频或视频，而流式应用则大多只向一个方向发送音频或视频。

因特网上运行的视频会议工具在 20 世纪 90 年代早期就开始出现了，但直到过去几年里才得到广泛的普及，因为网速更快，更强大的计算机也更加普及。图 1-1 显示了一个视频会议系统的例子。就像下载网页不只是满足视觉需要一样，视频应用也是这样。例如，使视频内容适合在低带宽网络中传输，或者确保视频和音频保持同步并及时到达来得到较好的用户体验，这些都是网络和协议设计者不得不考虑的问题。我们将在本书的后面考察这些问题，以及很多与多媒体应用相关的其他问题。

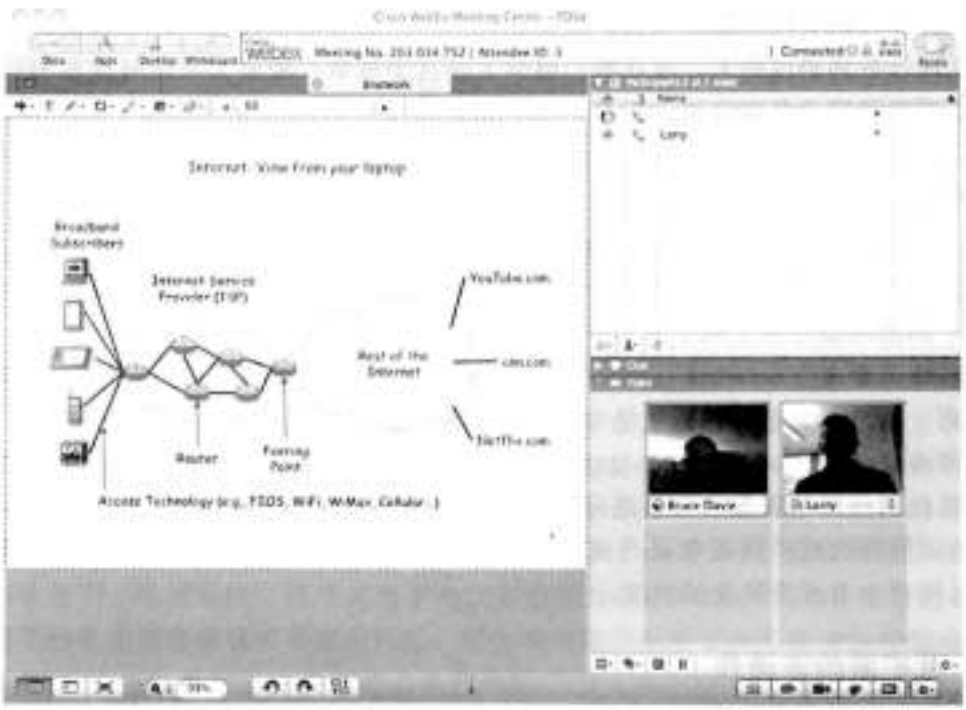


图 1-1　一个包含视频会议的多媒体应用

尽管这只是两个例子，但从网上下载页面和参加视频会议足以说明因特网应用的多样

⊖ 不完全是“越快越好”，人类因素研究表明，300ms 是打电话时人类能够忍受的往返时延的合理上限，超过这个上限，人们就要抱怨了，100ms 的时延看起来更好。

性，也暗示了因特网设计的复杂性。我们将在后面描述一个更完整的应用分类，来指导我们讨论关键的设计决策，从而建造、运营和使用能够支持大量不同应用的网络。第9章作为全书的结束部分，将重新讨论这两种特定的应用以及其他几种在当今因特网上流行的应用。现在，快速浏览几个典型应用就足以让我们开始考察为了建造支持多种应用的网络所必须解决的问题。[⊖]

1.2 需求

我们已经制定了一个宏伟的目标：了解如何从最底层开始建造一个计算机网络。要实现这个目标，我们将从基本原理开始，然后提出在建造实际网络时经常遇到的各种问题。在每一步中，我们会利用现有协议说明各种可行的设计方案，但是并不将这些人工设计的协议视作不变的真理。相反，我们还要不断地提出（并回答）网络为什么（why）要如此设计的问题。在力求理解现今网络的工作方式的同时，重要的是认识基本概念，因为随着技术的发展和新应用的不断出现，网络的变化日新月异。依照我们的经验，一旦人们理解了基本思想，那么对于遇到的任何新协议，消化和吸收起来都将变得相对容易。

1.2.1 视角

如前所述，学习网络的人有不同的视角。当我们写本书第1版时，大部分人根本不能访问因特网，那些能访问因特网的人都是在工作时、在大学里或在家里通过拨号调制解调器来访问，流行的应用也屈指可数。因此，就像当时的大部分教材一样，我们的书重点关注设计联网设备和协议的人。现在我们继续关注这个视角，希望大家在阅读本书后明白如何设计未来的联网设备和协议。但是，我们也想覆盖另外两个越来越重要的群体的视角：开发网络应用的人以及管理或运营网络的人。让我们想一下这三个群体会如何列举他们对网络的需求：

- *应用程序员*（application programmer）会列出应用所需的服务，例如，要保证应用程序发出的每条信息能在一定的时间内准确无误地传递，或随着用户的移动在不同的网络连接之间平稳切换的能力。
- *网络运营者*（network operator）会列出系统易于管理的特性，例如，故障易于隔离、新设备能够加入到网络中并正确配置、易于根据网络使用情况进行计费。
- *网络设计者*（network designer）会列出性价比高的设计所具备的属性，例如，有效地利用网络资源和公平地分配给不同的用户。性能问题可能也很重要。

本节试图将这些不同视角提升成关于驱动网络设计的主要关注点的一个高层次的概述，同时阐明本书所要解决的挑战。

1.2.2 可扩展的连通性

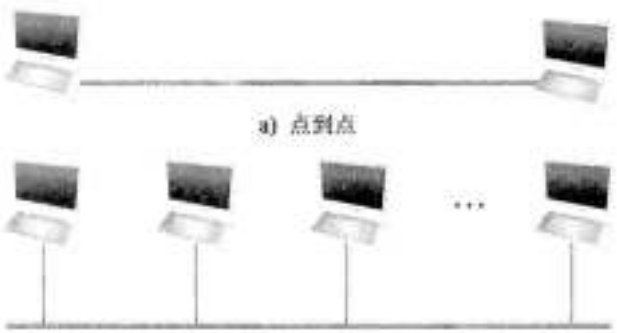
很明显，网络必须首先提供若干个计算机之间的连通性。有时候，只需要建立一个由选定的几台计算机连成的有限网络。但事实上，鉴于隐私性和安全性的考虑，许多专用的（企业的）网络都在限制接入网络的计算机方面有明确的目标。相反，其他一些网络（因特网是最明显的例子）则具有连入世界上所有计算机的潜力。如果一个系统设计支持任意

⊖ 参见"实验基础"。

规模的扩展，则称为可扩展的（scale）。本书以因特网为模型来解决可扩展性方面的挑战。

链路、节点和云形图

为了更全面地理解连通性需求，我们需要更仔细地考察计算机是如何连接到网络的。网络连通性体现在多个层次上。在最底层，网络可以由两台或更多台计算机通过某种物理介质（如同轴电缆或光纤）直接相连。我们称这样的物理介质为链路（link），并称被连接的计算机为节点（node）。（有时候，节点是指更为特殊的硬件而不是指计算机，这种区别我们通常不予区分。）如图 1-2 所示，有时一条物理链路仅存在于一对节点之间（这样的链路称为点到点（point-to-point）链路），而在其他情况下，多个节点可以共享一条物理链路（这样的链路称为多路访问（multiple-access）链路）。无线链路，如蜂窝网络和 Wi-Fi 网络提供的链路，成为越来越重要的多路访问链路。通常情况下，多路访问链路的大小是有限的，包括它们所能覆盖的地理范围和所能连接的节点数目。



a) 点到点

b) 多路访问

图 1-2 直接链路

如果计算机网络被局限于所有节点都通过一个公共的物理介质彼此直接相连的情况，那么，要么网络所能连接的计算机数目非常有限，要么从每个节点引出的线路数目很快会变得难以控制且代价高昂。幸运的是两个节点之间的连通性并不一定意味着它们之间有直接的物理连接，通过一系列合作节点可以实现间接的连通性。下面两个例子说明了一组计算机是如何间接连通的。

图 1-3 显示了一组节点，每个节点都连到一条或多条点到点链路上。那些连着至少两条链路的节点运行软件，用于将从一条链路收到的数据转发到另一条链路上。如果将这些节点按系统化的方法进行组织，就形成一个交换网（switched network）。交换网有很多种类型，电路交换（circuit switched）和分组交换（packet switched）是其中最为常见的两种。前者主要用于电话系统，而后者用于绝大多数的计算机网络，本书将侧重后者。（但是，电路交换在光网络领域将要东山再起，因为对网络容量需求的不断增加使得电路交换变得更重要。）分组交换网最主要的特点是网络中的节点彼此间发送离散的数据块。可以将这些数据块看作应用数据，如一个文件、一封电子邮件或一幅图像。我们将每个数据块称为一个分组（packet）或一条消息（message），目前这两个术语可以互换使用，但它们也不是完全相同，我们将在 1.2.3 节讨论原因。

分组交换网一般使用一种叫作存储转发（store-and-forward）的策略。顾名思义，存储转发网络中的每个节点先通过某条链路接收一个完整的分组，将这个分组存储在内存中，然后将整个分组转发给下一个节点。与之不同的是，电路交换网首先通过一系列链路

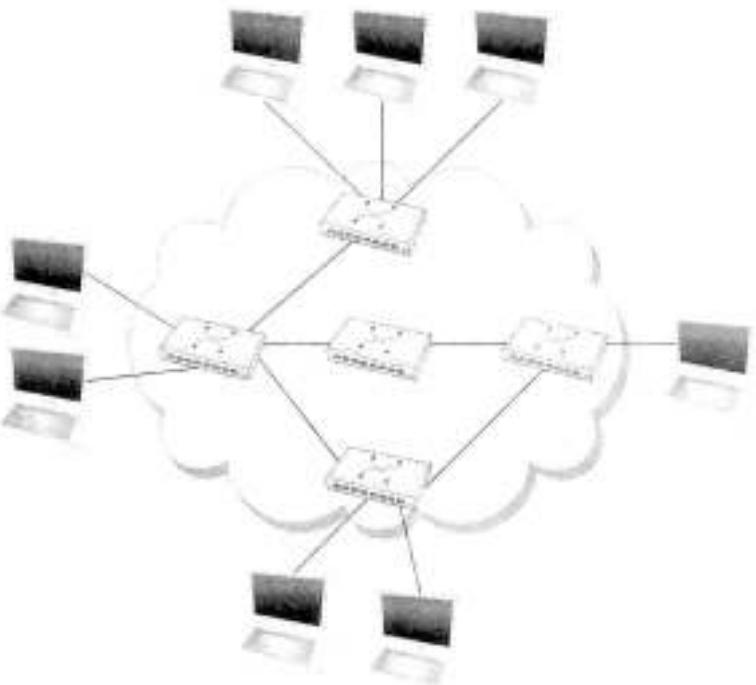


图 1-3 交换网

建立一条专用电路，然后允许源节点通过这条链路将比特流发送到目标节点。在计算机网络中使用分组交换而不使用电路交换的主要原因是分组交换效率较高。这个问题将在下一节讨论。

图 1-3 中的云形图将实现（implement）网络的内部节点（内部节点通常称为交换机（switch），其基本功能是存储和转发分组）与使用（use）网络的外部节点（外部节点通常称为主机（host），其任务是支持用户并运行应用程序）区分开来。还要注意，图 1-3 中的云形图是计算机网络中最重要的图标之一。一般来讲，我们可以利用云形图表示任何类型的网络，即无论网络是一条点到点链路、一条多路访问链路或是一个交换网。因此，无论何时在图上看到云形图，都可以把它看作本书讨论的任意一种网络技术的表示。[⊖]

计算机间接连通的另一种方法如图 1-4 所示。在这种情况下，一些独立的网络（云）互联形成一个互联网（internetwork，简称 internet）。我们依照因特网的习惯，将通称的互联网用“i”开头的 internet 表示，而将当前运行的 TCP/IP 因特网用“I”开头的 Internet 表示。连接到两个或多个网络的节点通常称为路由器（router）或网关（gateway），它与交换机所起的作用大致相同，即把消息从一个网络转发到另一个网络。注意，互联网本身可被看作是另一种类型的网络，也就是说，一个互联网可由多个互联网互联而成。这样，我们可以通过将云互联成更大的云来递归地构建任意大的网络。可以认为将区别很大的网络互联起来的思想是因特网的根本性创新，早期因特网架构师所做的优秀的设计决策使因特网成功扩展到全球规模，并拥有几十亿个节点，这些我们将在后面进行讨论。

⊖ 有趣的是，这种使用云的方式比云计算这个术语早至少几十年，但两种用法之间是有关联的，我们将在后面讨论。



图 1-4 网络的互联

仅仅将主机彼此直接或间接相连并不意味着已经成功地提供了主机到主机的连通性，连通性的最终要求是每个节点必须能够指明它想与网络上哪些其他节点进行通信。通过给每个节点指定一个地址（address）可以做到这一点。地址是标识节点的字节串，也就是说，网络可以使用节点地址来区分该节点与连接到网络的其他节点。当一个源节点通过网络传输一条消息给一个特定的目的节点时，它必须指明目的节点的地址。如果发送和接收节点不是直接相连的，那么网络上的交换机和路由器将使用这个地址来确定如何将消息转发到目标节点。根据目的节点地址来确定如何将消息转发到目的节点的过程称为路由（routing）。

上述关于寻址和路由的简介假设源节点要发送消息给单个目的节点（单播（unicast））。尽管这种情况最常见，但源节点也可能向网络上的所有节点广播（broadcast）一条消息，或者源节点要发一条消息给其他节点的某个子集，而并非所有节点，这种情况称为多播（multicast）。因此，除了特定节点的地址以外，网络的另一个需求是支持多播地址和广播地址。

结论 通过以上讨论，我们可以将网络递归地定义为通过一条物理链路相连的两个或多个节点，或者说通过一个节点连接的两个或多个网络。换言之，网络可以由网络的嵌套来构成，在最底层，网络由某种物理介质实现。实现计算机网络连通性的一个关键是为网络中的每个可达节点定义一个地址（包括支持广播和多播），并且能够利用这个地址将消息发往正确的目的节点。

1.2.3 高性价比的资源共享

如上所述，本书侧重介绍分组交换网。本节讲解计算机网络的关键需求——效率，这是我们选择分组交换策略的直接原因。

给定一个节点的集合，节点之间通过一个嵌套的网络间接相连，任何一对主机都可能通过一系列链路和节点互相发送消息。当然，我们不仅仅需要支持一对主机之间的通信，

而是希望提供在任何一对主机间交换消息的能力。那么，问题就是如何使所有希望通信的主机都能共享网络，特别是如果它们想同时使用网络时如何共享。假如这个问题还不太难解决，那么当多个主机需要同时使用同一条链路（link）时，应该如何实现共享？

为了弄清楚主机如何共享一个网络，我们需要引入一个基本概念——多路复用（multiplexing），意思是系统资源在多个用户之间共享。直观上可将多路复用与一个分时计算机系统类比，一个物理CPU被多个任务共享（多路复用），每个任务都认为它有自己的专用处理器。同样，由多个用户发送的数据可在构成网络的多条物理链路上被多路复用。

为了理解多路复用如何工作，考察图1-5所示的简单网络，网络左边的3台主机（发送方S1～S3）通过共享一个只包含一条物理链路的网络向网络右边的3台主机（接收方R1～R3）发送数据。（为简单起见，假设主机S1与主机R1通信，依此类推。）在这种情况下，对应于3对主机的3个数据流通过交换机1多路复用1条物理链路，然后再由交换机2多路分解（demultiplex）为独立的数据流。注意，我们有意对“数据流”的确切含义含糊其辞。为了便于讨论，假设左边的每台主机有大量的数据要发送到右边对应的主机。

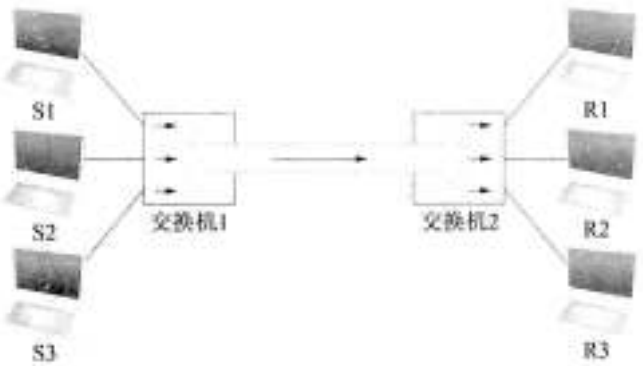


图1-5 在一条物理链路上复用多个逻辑流

将多个数据流多路复用到一条物理链路上有几种不同的方法。一种常用的方法是同步时分多路复用（synchronous time-division multiplexing，STDM）。STDM的思想是将时间划分为等长的时间片，以轮转方式使每个数据流都有机会将数据发送到物理链路上。换言之，在时间片1中，传输从S1发到R1的数据；在时间片2中，传输从S2发到R2的数据；时间片3传输从S3发到R3的数据，同时，从S1到R1的数据流再次准备开始传输，这个过程不断重复进行。另一种方法是频分多路复用（frequency-division multiplexing，FDM）。FDM的思想是将每个流以不同的频率发向物理链路，这一点与不同电视台以不同频率在电波上或在同轴电缆电视链路中传输电视信号很相像。

尽管原理很容易理解，但STDM和FDM都有两个方面的局限性。第一，如果有一个流（主机对）没有数据要发送，它占用的物理链路（即它的时间片或频率）就会空闲，即使其他流此时有数据要发送也无法使用。例如，上例中的S3必须要排在S1和S2后面发送数据，即使S1和S2没有数据要发送。对于计算机通信来说，一条链路空闲的时间可能会很长，例如，与获取页面的时间相比，阅读Web页面的时间较长，而在阅读页面时，链路是空闲的。第二，STDM和FDM所容纳的数据流的最大数目是事先固定和已知的。在STDM的情况下，修改时间片的大小、添加额外的时间片或在FDM情况下加入新频率，都是不实际的。

克服上述不足的多路复用形式，也是本书使用最多的多路复用形式是统计多路复用（statistical multiplexing）。尽管这个名字本身令人费解，但统计多路复用实际上非常简单，它包括两个关键的思想。首先，像 STDM 一样，它是按照时间来共享物理链路的，一个流的数据先被传输到物理链路上，然后再传输另一个流的数据，依此类推。然而，不同于 STDM 的是，每个流的数据是根据需要传输的，而不是在一个预先规定的时间片进行传输。这样，如果只有一个流有数据要发送，那么它不必等到它的时间片到来就可以发送数据，这样，也就不会出现分配给其他流的时间片白白浪费的情况。正是这种避免空闲时间的方法使分组交换具有较高的效率。

然而，截至目前的定义，统计多路复用没有这样的机制用以保证所有流最终都有机会在物理链路上传输数据。也就是说，一旦一个流开始发送数据，我们需要采用某种方法来限制它的发送，使得其他流能够有机会传输数据。考虑到这种需求，统计多路复用定义每个流在给定的时间内允许传输的数据块大小的上界。这个限定大小的数据块通常称为一个分组（packet），用以与应用程序可能传输的任意大小的消息（message）作区分。因为分组交换网限制分组的最大尺寸，所以主机可能无法用一个分组发送一条完整的消息。源节点可能需要将消息分划为几个分组，接收者再把这些分组重新组装成原始消息。

换言之，每个流通过物理链路发送一系列分组，以分组为单位决定下一次发送分组的流。注意，如果只有一个流有数据要发送，它可以连续地发送一个分组序列。然而，如果有多个流要发送数据，那么，它们的分组将会在链路上交替发送。图 1-6 描述了一个交换机将来自多个源节点的分组多路复用到一条共享链路上。

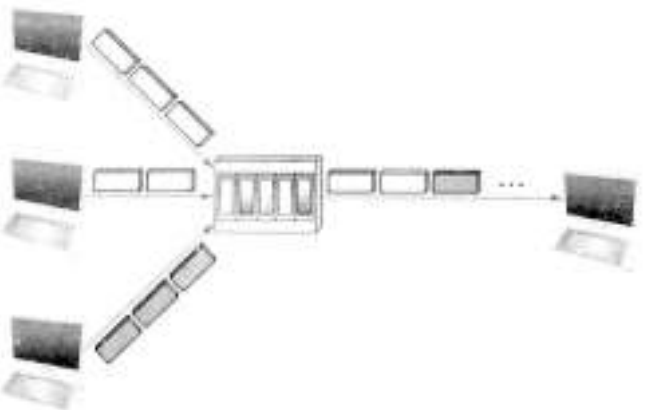


图 1-6 交换机将来自多个源节点的分组多路复用到一条共享链路上

确定在一个共享链路上发送哪一个分组有多种不同的方法。如图 1-5 所示，在由通过链路互联的交换机构成的网络中，可以由将分组发送到共享链路上的交换机来决定发送分组的顺序。（正如我们下面将要看到的，并不是所有的分组交换网都包含交换机，还可以用其他机制来决定下一步将哪个数据流的分组送到链路上。）在分组交换网中，每个交换机都能以分组为单位独立地决定发送顺序。网络设计者面对的问题之一是如何以一种公平的方式来做决定。例如，一个交换机可以设计成以先进先出（FIFO）的方式发送分组。另一种设计是交换机以循环方式转发来自通过该交换机发送数据的每一个不同的数据流的分组。这样做可以确保特定的数据流能分享链路的一定带宽，或者确保它们的分组在交换机中的延迟不会超过一段确定的时间。能够为特定数据流分配带宽的计算机网络有时称为

支持服务质量（Quality of Service，QoS），我们将在第6章讨论这个主题。

另外，值得注意的是，在图1-6中，由于交换机必须把三个进入的分组流多路复用到一条输出链路上，所以交换机接收分组的速度很有可能比共享链路传输分组的速度快。此时，交换机不得不将分组缓存在内存中。如果在较长的一段时间内，交换机接收分组的速度比它发送分组的速度快，那么交换机将最终耗尽其缓存空间，一些分组就不得不被丢弃。当交换机处于这种状态时，称为拥塞（congested）。

结论 最重要的是，统计多路复用定义了一种有效的方法让多个用户（例如主机到主机的数据流）以细粒度的方式共享网络资源（链路和节点）。它以分组粒度将网络链路分配给不同的数据流，每个交换机能够以分组为单位来规划如何使用与之相连的物理链路。为不同的数据流公平分配链路容量以及处理拥塞是统计多路复用面临的主要挑战。

相关主题

SAN、LAN、MAN 和 WAN

描述网络的一种方式是根据其大小。两个著名的例子是局域网（LAN）和广域网（WAN）。前者的范围通常用在1公里以内，而后者则可以是世界范围的。其他网络被归类为城域网（MAN），通常跨越几十公里。这种划分非常有趣，原因是网络的大小通常预示着它能使用的底层技术，其中一个关键因素是数据从网络的一端传播到另一端所花的时间。我们将在后面几章中更详细地讨论这个问题。

根据有趣的历史记录记载，广域网（wide area network）这个术语并没有用在第一个WAN上，因为当时没有其他类型的网络需要与之区分。当时计算机非常少并且也较昂贵，人们从没想到过将局部范围内的所有计算机连在一起，因为在那个范围内只有一台计算机。只有当计算机开始增多，需要有LAN时，“WAN”这个术语才被引入，用来描述将地理位置较远的计算机互联起来的更大的网络。

另一种我们需要了解的网络是SAN（现在称为存储区域网（storage area network），但以前也被称为系统区域网（system area network））。SAN通常限制在一个房间里，并且把一个大的计算系统的各种组件连接起来。例如，光纤信道是一种常见的SAN技术，用来将高性能计算系统连接到存储服务器和数据仓库上。尽管本书没有详细地描述这类网络，但了解它们还是很有意义的，因为它们通常在性能方面引领潮流，而且将这类网络连入LAN或WAN的情况也越来越多。

1.2.4 支持通用服务

前一节概述了在一组主机间提供高性价比的连通性所面临的挑战，但是，将计算机网络仅视作是在一系列计算机之间传输分组未免过于简单。更准确地说，应该将计算机网络视为能够为分布在这些计算机上的一系列应用进程提供彼此通信的手段。换言之，计算机网络的下一个需求是让连接到网络中的主机上运行的应用程序能够以一种有意义的方式进行通信。从应用开发者的角度来看，网络应该使其生活更美好。

当两个应用程序需要彼此通信时，并不仅仅是主机间的消息传输，而要涉及很多复杂的过程。一种选择是应用程序设计者把所有复杂的功能都集成在每个应用程序中。然而，

由于很多应用需要通用的服务，所以更明智的做法是一次性地实现这些通用服务，然后应用设计者再用这些服务构建所需的应用。如何确定正确的通用服务集是网络设计者面临的一个挑战，其目标是对应用隐藏网络的复杂性但也不过分限制应用设计者。

直观地讲，我们将网络视作为应用层进程提供逻辑信道（channel），使得它们能够基于逻辑信道相互通信，每个信道提供应用程序所需的服务集。换言之，正如我们用云抽象地表示计算机之间的连通性一样，现在，我们将信道视作一个进程与另一个进程的连接通道。图 1-7 显示了一对应用层进程在逻辑信道上通信的情况，逻辑信道是在连接主机的网络（云）上实现的。我们可以把信道看成连接两个应用程序的一条管道，发送应用程序把数据放在管道的一端，并希望数据能被网络传输给位于管道另一端的应用程序。

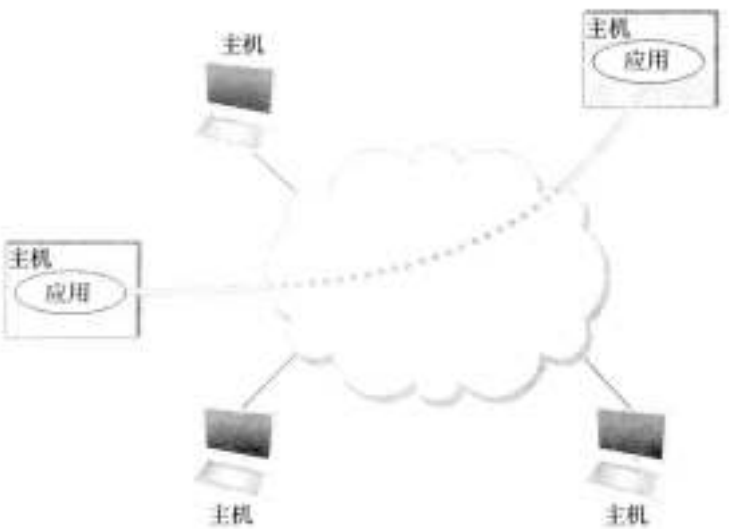


图 1-7 在抽象信道上进行通信的进程

确定信道应该给应用程序提供什么功能是当前面临的挑战。例如，应用是否需要保证在信道上发送的消息被可靠地传输，或者说是否允许丢失某些消息？应用要求消息必须按它们发送时的顺序到达接收进程，还是接收者并不关心消息到达的顺序？网络需要保证没有第三方在信道上窃听，还是并不在意隐私问题？总之，网络提供各种不同类型的信道，每个应用选择它所需要的类型。下面将说明定义有用信道的设计思路。

1. 确定通用通信模式

设计抽象信道首先需要理解一些典型应用的通信需求，然后提取出它们共同的通信需求，最后具体化为网络中符合这些需求的功能。

最早被所有网络支持的应用之一是文件访问程序，如文件传输协议（File Transfer Protocol，FTP）或网络文件系统（Network File System，NFS）。尽管很多细节不同，例如，是通过网络传输整个文件还是在给定的时间只读/写文件的一块，但远程文件访问的通信组件的特点都是有一对进程，一个进程请求读/写文件，另一个进程响应这个请求。请求访问文件的进程称为客户端（client），支持文件访问的进程称为服务器（server）。

读文件时，客户端给服务器发一个小的请求消息，而服务器用一个包含文件数据的大消息作为响应。写操作按照相反的方式进行，客户端给服务器发一个包含将被写入的数据的大消息，而服务器用一个确认数据已经写到磁盘上的小消息作为响应。

数字化图书馆是一个比文件传输更复杂的应用，但是它需要的通信服务是类似的。例如，国际计算机协会（Association for Computing Machinery，ACM）运营一个大型的计算机科学文献数字图书馆，网址是 http://portal.acm.org/dl.cfm。这个图书馆提供各种各样的搜索和浏览功能来帮助用户找到他们想要的文章，但最终其做得最多的工作还是响应用户的文件访问请求，如期刊文章的电子版，这与FTP服务器很类似。

以文件访问、数字化图书馆和前面介绍的两个视频应用（视频会议和视频点播）为典型例子，我们决定提供以下两种信道：请求/应答（request/reply）信道和消息流（message stream）信道。请求/应答信道可用于文件传输和数字化图书馆应用。它保证由一方发送的每条消息都被另一方接收，并且每条消息只传输一份拷贝。请求/应答信道还可以保证在其上传输的数据的隐私和完整性，未经授权不能读或修改客户与服务器进程之间交换的数据。

只要设置了参数来支持单向和双向传输以及不同的时延特性，消息流信道就可用于视频点播和视频会议应用。消息流信道可能不需要保证所有的消息都被发送，因为即使一些视频帧没有被接收到，视频应用仍可很好地运行。然而，它需要保证所传输的消息必须按发送的顺序到达，以避免播放乱序的帧。与请求/应答信道一样，消息流信道需要保证视频数据的隐私和完整性。最后，消息流信道需要支持多播，这样多方才能够同时参与远程会议或观看视频。

对网络设计者来说，通常会采用最少的抽象信道类型来为最多的应用提供服务，但抽象信道类型太少也会有风险。简单地说，如果你有一个锤子，便会觉得所有的东西都像钉子。举个例子，如果只有消息流信道和请求/应答信道，那么，在下一个应用出现时，你还是想使用这两种信道，即使任何一个信道都不能完全提供该应用所需的语义。因此，只要程序员不断创造出新应用，网络设计者就可能创造出新的信道类型，并为现有信道增加选项。

另外，值得注意的是，不管一个特定信道究竟提供何种（what）功能，都存在一个在何处（where）实现这种功能的问题。在很多情况下，最容易的方法是把底层网络中主机到主机的连通性看作是提供了一个比特管道（bit pipe），而高层通信语义由终端主机提供。这种方法的好处是可以使网络中的交换机尽可能地简单（仅转发分组），但要求终端主机承担更多的任务，支持具有丰富语义的进程到进程信道。另一种方法是把附加功能都放到交换机上，从而允许终端主机成为“哑”设备（如手机）。我们将会看到关于如何在分组交换机和终端主机（设备）间分配各种网络服务的问题是一个在网络设计中一再出现的问题。

2. 可靠性

正如上述例子所表明的，可靠的消息传输是网络提供的最重要的功能之一。然而，如果不先明确网络失败的原因，就很难决定如何提供可靠性。首先应该认识到，计算机网络并不是存在于一个完美的世界中。计算机系统崩溃后重启、光纤断裂、电磁干扰使正在传输的数据位出错、交换机缓存溢出，这些物理问题已经足以令人担忧，更何况管理硬件的软件可能存在漏洞，有时所转发的分组还会丢失。因此，网络的一个重要需求就是从某些故障中恢复，使得应用程序不必处理这些故障，甚至不知道这些故障的存在。

网络设计者必须考虑的故障通常有三类。首先，当通过物理链路传输一个分组时，数据中可能会出现比特错（bit error），也就是说，1变成0或0变成1。有时单个比特出错，

更多的时候会发生突发差错（burst error）——几个连续的比特损坏。比特差错通常是由外界原因引起的，如闪电、电力波动或微波炉干扰数据的传输。好在比特差错很少发生，在典型的铜电缆中传输时，平均每 $10^6 \sim 10^7$ 比特中出现一个比特差错，在典型的光纤中传输时，平均每 $10^{12} \sim 10^{13}$ 比特中出现一个比特差错。我们将看到，有些技术能够以很高的概率检测出这些比特差错。一旦检测到比特差错，就有可能改正这些错误。如果我们知道哪个比特或哪些比特出错的话，简单地取反即可；但是如果损坏非常严重，就必须丢弃整个分组。在这种情况下，则希望发送者重传分组。

第二类错误发生在分组级而不是比特级，也就是说，网络丢失了整个分组。发生这类错误的原因之一是分组含有不可纠正的比特差错，因此不得不丢掉。然而，可能性更大的原因是，一个必须处理该分组的节点（例如将分组从一条链路转发到另一条链路的交换机）由于过载而没有空间存储分组，因而被迫将其丢掉。这是我们在 1.2.3 节中提到的拥塞问题。另一种不太普遍的原因是处理分组的节点上运行的软件出现错误。例如，它可能将一个分组转发到一条错误的链路，使得分组找不到最终目的地。我们会看到，处理丢失分组的主要困难是识别该分组究竟是确实丢失了，还是仅仅延迟到达目的地。

第三类错误发生在节点和链路级，也就是说，物理链路被切断或所连接的计算机崩溃了。这类错误可能由软件崩溃、电源故障或人工操作失误所引起，由于网络设备的错误配置所引起的失败也是很普遍的，虽然这样的错误最终能被更正，但仍会在一段较长的时间内对网络产生很大的影响。然而，它们也不至于使网络完全不能使用。例如，在分组交换网中，有时可能路由到故障节点或链路。处理第三类错误的困难在于判断一台计算机是由于出了故障还是速度慢，或者判断一条链路是由于断了还是性能很差而导致大量的比特差错。

结论 经过以上讨论可知，定义有用的信道需要理解应用的需求并明确底层技术的局限性。我们所面临的挑战是如何缩小应用需求与底层技术所能提供服务之间的差距。这种差距有时称为语义鸿沟（semantic gap）。

1.2.5 可管理性

最后一个需求似乎经常被忽略或者到最后才被提出[⊖]，这就是网络的可管理性。管理网络包括当需要扩大网络来承载更多流量或支持更多用户时对网络做适当改变，以及当出现问题或性能不如预期时排除网络故障。

这个需求在某种程度上与上面所讨论的可扩展性有关，因为因特网已经扩展到能够支持几十亿用户和至少几亿台主机，确保整个网络正确运行且正确配置新添加的设备越来越成问题。配置网络中的路由器通常是受过训练的专家的任务，而配置几千台路由器并能找出如此大规模的网络为什么不能按照预期运行的原因则绝不是任何个人能承担的任务。另外，为了使得网络的运营具有可扩展性和更高的性价比，网络运营商一般需要很多管理任务能够自动执行，或者至少可以由技术水平相对不太高的人来完成。

自从我们写完本书第 1 版，网络发展的一个重要现象是家庭网络已经非常普及。这意味着网络管理不再是专家的任务，而是需要由经过少量培训或没有培训过的用户来完成。这有时也被描述为需要网络设备“即插即用”，然而事实证明这个目标相当模糊。稍后，我们将讨论部分满足该需求的一些方法，但是目前需要注意的是改善网络的可管理性依旧

⊖ 就像我们在本节中所做的那样。

是当前重要的研究领域。

1.3 网络体系结构

也许你尚未注意到，前一节已建立了一个相当丰富的网络设计需求的集合，即一个计算机网络必须提供大量计算机之间通用的、高性价比的、公平的和健壮的连通性。但似乎这还不够，因为网络不是一成不变的，它必须适应所依赖的底层技术的变化以及应用程序对网络需求的变化。另外，网络必须能够由不同技术水平的人来管理。设计一个满足这些需求的网络并非易事。

为了处理这个复杂的问题，网络设计者已经制定了通用的蓝图，通常称为网络体系结构（network architecture），用以指导网络的设计和实现。本节通过引入所有网络体系结构所共有的核心思想，来更详细地定义什么是网络体系结构。我们还将介绍两个被广泛参考的体系结构，即OSI（或7层）体系结构和因特网体系结构。

1.3.1 分层和协议

抽象——隐藏定义良好的接口背后的细节——是系统设计者用于处理复杂性的基本工具。抽象的思想是定义一个能捕获系统某些特征的模型，并将这个模型封装为一个对象，为系统的其他组件提供一个可操作的接口，同时向对象使用者隐藏对象的实现细节。困难在于如何确定抽象，使得它们既能够提供适用于大多数情况的服务，同时能够在底层系统中高效实现。这正是前一节中我们引入信道的概念时想要做的：我们想为应用提供一个抽象，对应用开发者隐藏网络的复杂性。

分层是抽象的自然结果，特别是在网络系统中。分层的总体思想是从底层硬件提供的服务开始，然后增加一系列的层，每一层都提供更高级（更抽象）的服务。高层提供的服务用低层提供的服务来实现。例如，总结前一节给出的有关需求的讨论，可以将一个网络简单设想为夹在应用程序和底层硬件之间的两层抽象，如图1-8所示。在这种情况下，硬件上面的第一层可提供主机到主机的连接，对两台主机之间任意复杂的网络拓扑进行抽象。上面一层基于主机到主机的通信服务，对进程到进程的信道提供支持，对网络偶尔丢失消息这样的事实进行抽象。



图1-8 分层网络系统示例

分层具有两个优点。第一，它将建造网络这个问题分解为多个可处理的部分。不是把想要的所有功能都集中在一个软件中，而是可以实现多个层，每一层解决一部分问题。第二，它提供一种更为模块化的设计。如果想增加一些新服务，只需要修改某一层的功能，同时可以继续使用其他各层提供的功能。

然而，将系统看作层次的线性序列是一种过分简化。通常，在系统的任意一层上都提供多种抽象，每种抽象都建立在同样的低层抽象上，却分别向高层提供不同的服务。为了说明这一点，我们考察1.2.4节讨论过的两种信道：一种提供请求/应答服务，另一种支持消息流服务。这两种信道在多层网络系统的某个特定层上是可选的，如图1-9所示。



图1-9 特定层上具有可选抽象的分层系统

在有关分层讨论的基础之上，我们能够更准确地讨论网络的体系结构。对初学者而言，构成网络系统分层的抽象对象称为协议（protocol）。就是说，一个协议提供一种通信服务，供高层对象（如一个应用进程或更高层的协议）交换消息。例如，我们可以设想一个支持请求/应答协议和消息流协议的网络，分别对应于上面讨论过的请求/应答信道和消息流信道。

每个协议定义两种不同的接口。首先，它为同一计算机上想使用其通信服务的其他对象定义一个服务接口（service interface）。这个服务接口定义了本地对象可以在该协议上执行的操作。例如，一个请求/应答协议可支持应用的发送和接收消息的操作。HTTP 协议的实现能够支持从远程服务器获取超文本页面的操作。当用户在当前页面上单击链接时，像 Web 浏览器这样的应用将调用该操作来获取新页面。

其次，协议为另一台机器上的对等实体定义一个对等接口（peer interface）。第二种接口定义了对等实体之间为实现通信服务而交换的消息的格式和含义。这将决定一台机器上的请求/应答协议以什么方式与另一台机器上的对等实体进行通信。例如，在 HTTP 协议中，协议规范详细定义了“GET”命令的格式，包括该命令可以使用哪些参数，以及当接收到此命令时 Web 服务器该如何响应（有关此协议的详情参见 9.1.2 节）。

总之，协议定义了一个本地输出的通信服务（服务接口）以及一组规则，这些规则用于管理协议及其对等实体为实现该服务而交换的消息（对等接口）。这种情况如图 1-10 所示。

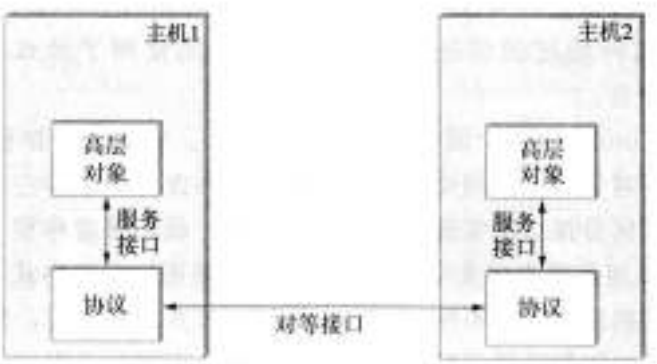


图 1-10 服务接口和对等接口

除了硬件层上的对等实体之间通过一条链路直接进行通信，对等实体间的通信都是间接的。每个协议和它的对等实体的通信过程是将消息传给更低层的协议，再由更低层协议将消息发给它的对等实体。另外，在任一层都可能有潜在的提供不同通信服务的多种协议，每一种协议提供不同的通信服务。因此我们用协议图（protocol graph）来表示构成网络系统的协议族，图中的节点对应于协议，边表示依赖关系。例如，图 1-11 描述了我们刚才讨论过的假想分层系统的协议图，协议 RRP（请求/应答协议）和 MSP（消息流协议）实现两种不同类型的进程到进程信道，并且都依赖于提供主机到主机连通服务的主机到主机协议（HHP）。

在这个例子中，假设主机 1 上的文件访问程序要使用协议 RRP 提供的通信服务给主机 2 上的对等实体发送一条消息。在这种情况下，文件应用请求 RRP 代表它发送消息。为了与对等实体进行通信，RRP 调用 HHP 服务，HHP 将消息送往另一台机器的对等实

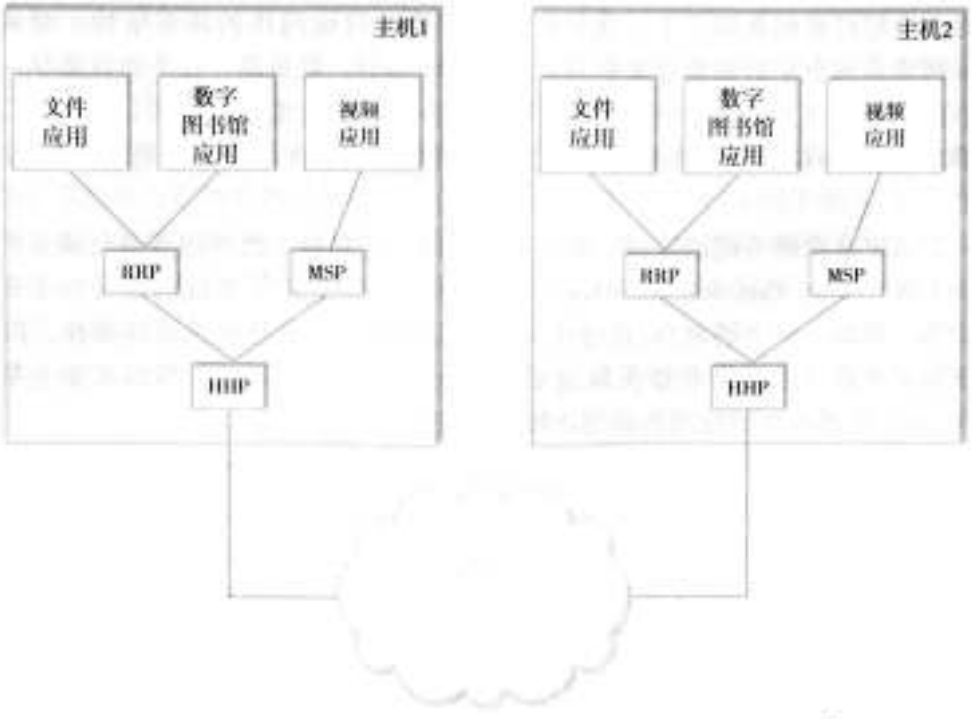


图 1-11 协议图示例

体。一旦消息到达主机 2 上的 HHP 实例，HHP 就将消息向上传给 RRP，RRP 再将消息传给文件应用。在这种特定的情况下，我们称该应用使用了协议栈（protocol stack）RRP/HHP 提供的服务。

注意，协议（protocol）这个词有两种不同的用法。它有时是指抽象接口，即由服务接口定义的操作以及对等实体之间交换的消息的格式和含义；有时它又表示实际实现这两种接口的模块。为了区分接口和实现这些接口的模块，我们通常称前者为协议规范（protocol specification）。规范通常用文字、伪码、状态转换图、分组格式和其他抽象符号的组合来表示。一个给定的协议可由不同的程序员按不同的方法来实现，只要符合协议规范即可。实现的困难在于如何保证使用同样协议规范的两个不同实现能成功地交换消息。能够准确地实现一个协议规范的两个或多个协议模块称为彼此可互操作（interoperate）。

我们可以设想出许多满足一组应用通信需求的不同的协议和协议图。幸运的是，有些标准化组织会为具体的协议图制定政策，比如国际标准化组织（ISO）和因特网工程任务组（IETF）。我们规定一个协议图的格式和内容的规则的集合称为网络体系结构（network architecture）。尽管超出本书范围，但 ISO 和 IETF 等标准化组织已经在它们各自的体系内建立了严格定义的用于引入、验证和最终批准协议的规程。我们将在后面描述由 ISO 和 IETF 定义的体系结构，但是首先需要解释有关协议分层机制的另外两个要素。

1. 封装

对于图 1-11，考虑当一个应用程序通过将消息传输给 RRP 协议来向它的对等实体发送消息时会发生什么情况。从 RRP 的角度来看，应用程序给出的消息是一个无解释的字节串。RRP 并不关心这些字节表示一个整型数组、一封电子邮件、一幅数字化图像或是其他什么，它只是负责将这些字节发给它的对等实体。然而，RRP 必须将控制信息传输

给它的对等实体，指示对等实体如何处理收到的消息。为做到这一点，RRP 将一个首部（header）附加到消息上。一般说来，首部是一个小的数据结构，从几个字节到几十个字节，用于对等实体彼此间的通信。顾名思义，首部通常加到消息的前面。然而在有些情况下，对等实体之间的控制信息在消息的末尾发送，这时称为尾部（trailer）。RRP 附加首部的确切格式是由其协议规范定义的。消息的其余部分称为消息体（body）或有效载荷（payload），即要替应用程序传输的数据。我们称应用程序的数据被封装（encapsulate）在一个由 RRP 协议创建的新消息中。

封装过程在协议图的每一层都被重复，例如，HHP 封装 RRP 的消息，加上一个自己的首部。如果我们假设现在 HHP 通过网络发送消息给它的对等实体，那么当消息到达目的主机时，它将以相反的顺序被处理：HHP 首先解释消息前面的首部（即根据首部的内容采取相应的行动），并且将消息体向上传给 RRP（不包括 HHP 的首部），RRP 根据其对等方所添加的首部的指示采取行动，并将消息体（不包括 RRP 的首部）向上传给应用程序。由 RRP 向上传递给主机 2 上的应用程序的消息正是应用程序向下传递给主机 1 的 RRP 的消息，应用程序看不见任何附加到消息上以实现更低层通信服务的首部。整个过程如图 1-12 所示。注意，在这个例子中，网络中的节点（如交换机和路由器）可能会检查消息前面的 HHP 首部。

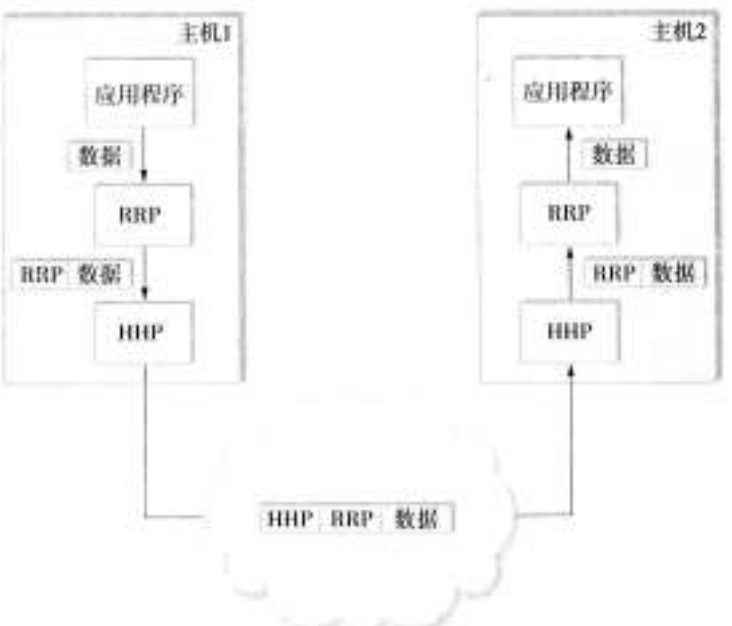


图 1-12 高层消息被封装在低层消息内

注意，当我们说一个低层协议不解释某个高层协议发送过来的消息时，是指它不知道如何从消息所含的数据中得出含义。然而，在有些情况下，低层协议会对接收到的数据做某种简单的转换，如压缩或加密。此时，协议转换整个消息体，包括原始的应用数据和由高层协议加在数据上的所有首部。

2. 多路复用和多路分解

回顾 1.2.3 节分组交换的基本思想：在一条物理链路上多路复用多个数据流。同样的思想也适用于协议图中的上层和下层，而不只用于交换节点。例如，在图 1-11 中，可将

RRP 看作实现了一条逻辑通信信道，来自两个不同应用的消息在源主机上多路复用到这条信道，然后在目标主机上多路分解到相应的应用。

实际上，这仅仅意味着 RRP 附加到消息上的首部包含一个标识符，用于记录消息属于哪个应用程序。我们将这个标识符称为 RRP 的多路分解键（demultiplexing key，简写为 demux key）。在源主机上，RRP 在首部中包含相应的多路分解键。当消息传到目标主机上的 RRP 时，它剥下首部，检查多路分解键，然后将消息多路分解到正确的应用程序中。

RRP 并不是唯一支持多路复用的协议，几乎每个协议都能实现这种机制。例如，HHP 有自己的多路分解键，用于确定哪些消息上传给 RRP，哪些消息上传给 MSP。但是，对于多路分解键的构成，在协议（甚至那些在同一个网络体系中的协议）之间并没有统一的规定。一些协议使用 8 位的字段（表示只能支持 256 个高层协议），而另一些协议使用 16 位或 32 位的字段。而且，一些协议在首部中只有一个多路分解字段，而另一些协议可能有一对多路分解字段。在前一种情况下，通信双方使用相同的多路分解键；而在后一种情况下，通信双方使用不同的键来识别消息将被传输给哪个高层协议（或应用程序）。

3. 七层模型

ISO 是最早正式定义计算机互联的通用方法的组织之一。它们的体系结构称为开放系统互联（Open Systems Interconnection，OSI）体系结构，如图 1-13 所示，将网络按功能划分为七层，由一个或多个协议实现分配给某个特定层的功能。这意味着图 1-13 并不是一个协议图，而是一个协议图的参考模型（reference model），通常称为七层模型。

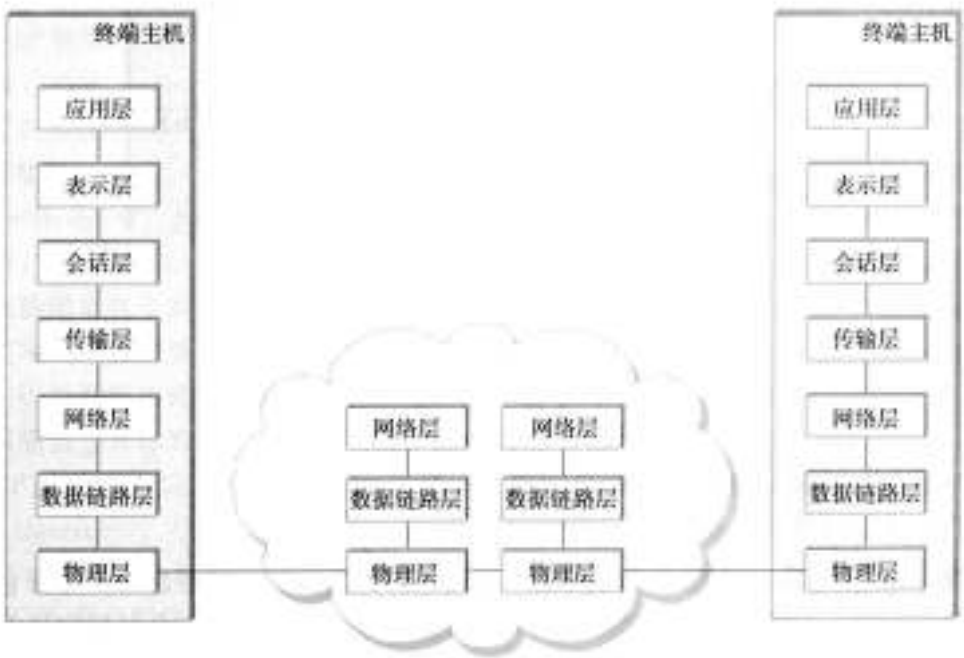


图 1-13 OSI 七层模型

从底层开始依次向上，物理（physical）层处理通信链路上原始比特的传输。然后，数据链路（data link）层收集比特流形成一个更大的集合体，称为帧（frame）。典型情况

下，由网络适配器和运行在节点操作系统上的设备驱动程序一起来实现数据链路层。这意味着实际传输给主机的是帧，而不是原始比特。网络（network）层处理一个分组交换网内节点的路由。在这一层，节点间交换的数据单元通常称为分组（packet）而不是帧，尽管它们基本上是一回事。较低的三层在所有网络节点中都要实现，包括网络内部的交换机和网络外部连接的主机。传输（transport）层实现我们提到过的进程到进程信道，在此，交换的数据单元通常称为消息（message）而不是分组或帧。传输层和更高层通常只在终端主机上运行，而不在中间交换机或路由器上运行。

关于上面三层的定义不太统一，部分原因是这三层并不总是出现，下面将会看到这一点。先看最顶层（第七层），这是应用（application）层。应用层协议包括超文本传输协议（HTTP）等，HTTP 是万维网的基础，它使 Web 浏览器能够向 Web 服务器请求页面。接下来，表示（presentation）层关注对等实体间交换的数据的格式，例如，整数是 16 位、32 位还是 64 位长，最先传输还是最后传输最高有效位，或者如何格式化一个视频流。最后，会话（session）层提供一个命名空间，用来将一个应用的各部分不同的传输流联系在一起。例如，它可以管理一个视频会议应用中结合在一起的一个音频流和一个视频流。

1.3.2 因特网体系结构

因特网体系结构有时也称为 TCP/IP 体系结构，因为 TCP 和 IP 是它的两个主要协议，如图 1-14 所示。图 1-15 给出另外一种表示方法。因特网体系结构是从早期的分组交换网 ARPANET 发展而来的。因特网和 ARPANET 都是由美国高级研究规划署（ARPA）资助创建的，ARPA 是美国国防部的研发基金机构之一。在 OSI 体系结构出现之前，因特网和 ARPANET 就已经存在了，并且在创建它们时所积累的经验对 OSI 参考模型产生了重大影响。

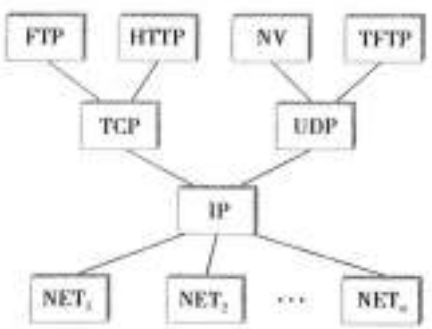


图 1-14 因特网协议图



图 1-15 因特网体系结构的另一个视角，子网层曾称为网络层，现在通常简称为第二层

尽管七层 OSI 模型理论上能够应用于因特网，但是实际上通常采用四层模型。在最低层是多种网络协议，表示为 NET_1、NET_2 等。实际应用中，这些协议由硬件（如网络适配器）和软件（如网络设备驱动程序）共同实现。例如，以太网或无线协议（如 802.11 Wi-Fi 标准）就会出现在这一层。（这些协议实际上还可以包含几个子层，但是因特网体系结构并没有对此做任何假设。）第二层只有一个网际协议（Internet Protocol，IP），这个协议支持多种网络技术互联为一个逻辑网络。第三层包括两个主要的协议：传输控制协议（Transmission Control Protocol，TCP）和用户数据报协议（User Datagram Protocol，

UDP)。TCP 和 UDP 为应用程序提供可选的逻辑信道：TCP 提供可靠的字节流信道，UDP 提供不可靠的数据报传输信道（数据报（datagram）可认为是消息的同义词）。在因特网的语言中，TCP 和 UDP 有时也称作端到端（end-to-end）协议，但称其为传输协议同样正确。

在传输层上运行了大量应用协议，如 HTTP、FTP、Telnet（远程登录）和简单邮件传输协议（Simple Mail Transfer Protocol，SMTP），使得常用的应用可以互操作。为了理解应用层协议和应用之间的区别，我们考虑所有正在使用或曾经使用过的不同 WWW 浏览器（如 Firefox、Safari、Netscape、Mosaic、Internet Explorer）。Web 服务器有大量类似的不同实现方式，可以使用这些应用程序中的任一个访问 Web 上的某个站点，原因是它们都符合同样的应用层协议：HTTP。有时用同一个词表示应用和该应用使用的应用层协议容易引起混淆，例如，FTP 通常用作实现 FTP 协议的一个应用的名字。

现在工作在网络领域的大部分人既熟悉因特网体系结构也熟悉七层 OSI 体系结构，并且就两个体系结构之间层次的对应关系达成了共识。因特网的应用层被看作是第七层，其传输层是第四层，IP 层（网络互联层或网络层）是第三层，IP 层下面的链路或子网层是第二层。

因特网体系结构有三个特点值得注意。第一，如图 1-15 所示，因特网体系结构没有严格地划分层。应用可以自由地跨过已定义的传输层，而直接使用 IP 层或低层网络。事实上，程序员可以自由定义新的信道抽象或在任何已有协议上运行的应用程序。

第二，如果仔细观察图 1-14 中的协议图，你会看到一个沙漏形状——顶部宽、中间窄、底部宽。这种形状实际上反映了该体系结构的中心哲学。就是说，IP 层作为体系结构的焦点，为各种网络中的交换分组定义一种通用方法。IP 层之上可以有多个传输协议，每个协议为应用程序提供一种不同的信道抽象。这样，从主机到主机传输消息的问题就从提供有用的进程到进程通信服务的问题中完全分离出来。IP 层之下，这个体系结构允许很多不同的网络技术，从以太网到无线再到单个的点到点链路。

因特网体系结构（或者更精确地称为 IETF 文化）的最后一个属性是：为了将一个新协议正式包含在网络体系结构中，必须有一个协议规范并至少有一个（最好两个）该规范的典型实现。被 IETF 采用的协议标准必须有能工作的协议实现。这种要求有助于保证体系结构的协议能够有效实现。也许因特网文化对于可工作软件的价值的最佳体现，就是 IETF 会议上常穿的 T 恤上印着的一句话：

> 我们拒绝国王、总统和选举。我们信奉的是大体上的一致意见和可执行的代码。
>
> ——Dave Clark

结论 在因特网体系结构的三个属性中，值得重申的是“沙漏”这一重要设计理念。沙漏的细腰部代表最小的、经过精心挑选的通用功能集，它允许高层应用和低层通信技术并存，共享各种功能并快速发展。细腰模型十分重要，它使得因特网能够快速适应用户的新要求和技术的变革。

1.4 实现网络软件

网络体系结构和协议规范是基础，但好的蓝图并不能完全解释因特网所取得的巨大成功：接入因特网的计算机数在将近 30 年的时间内呈指数增长（虽然现在很难获得确切的数字）。据估计，2009 年使用因特网的人数约 18 亿，这在世界人口中占有相当大的比例。

如何解释因特网的成功呢？当然有许多相关因素（包括一个好的体系结构），但因特网取得巨大成功的一个因素是它的许多功能是由通用计算机上运行的软件实现的。其意义在于只要进行“少量的编程”，就可以方便地增加一个新功能。于是，新的应用和服务一直在以令人难以置信的速度涌现，如电子商务、视频会议、IP 电话，等等。

另一个相关因素是商用计算机计算能力的显著增长。尽管计算机网络理论上能传输各类数据，如数字语音样本、数字化图像等，但如果计算机在发送和接收这些数据时的速度太慢，那么这些信息将失去价值，人们也会对它失去兴趣。实际上，当今所有计算机都能全速播放数字化语音，而且对某些应用（不是全部）而言，它能以可用的速率和分辨率播放视频。因此，当今的网络越来越多地用于承载多媒体，只有计算机硬件的速度变得更快，这种支持才会不断增强。

在本书第 1 版出版以后，写网络应用已经成为非常主流的活动，而不再是少数专家的工作。许多因素促成了这种现象，包括更好的工具使得编程工作对于非专业人员来说变得更简单，以及智能手机应用等新市场的出现。

值得注意的是，了解如何实现网络软件是理解计算机网络的重要部分，虽然可能不会要求你实现一个像 IP 这样的低层协议，但很有可能会找到实现一个应用层协议的理由——难以捉摸的“杀手级应用”会带来无法想象的名誉和利益。作为第一步，本节介绍在因特网上实现网络应用的相关问题，这些程序通常既是一个应用（用于与用户交互），同时也是一个协议（通过网络与对等方通信）。第 9 章作为本书的总结，将回到网络应用（应用层协议）这个主题，研究几个常见的例子。

1.4.1 应用程序编程接口（套接字）

实现网络应用时，要从网络提供的接口开始。由于大多数网络协议都是由软件实现的（特别是协议栈中的高层协议），而且几乎所有计算机系统都将网络协议的实现作为操作系统的一部分，所以当我们说“网络提供的”接口时，一般指的是操作系统为它的网络子系统提供的接口。这个接口通常称为网络应用程序编程接口（Application Programming Interface，API）。

虽然每个操作系统都可以自由地定义自己的网络 API（而且大多数已经定义），但随着时间的推移，有些 API 获得了广泛的支持，也就是说，它们被移植到除原始系统以外的操作系统中。套接字接口（socket interface）就是这样，它最初是由加州大学伯克利分校的 Unix 小组开发的，而现在几乎所有流行的操作系统都支持它，并且很多特定语言的接口都是以它为基础的，如 Java 套接字库。业界支持单一 API 的好处是应用程序很容易从一个操作系统移植到另一个操作系统，开发者很容易编写用于多个操作系统的应用。

在描述套接字接口之前，区分两件事很重要。每个协议提供一系列服务（service），API 则提供特定操作系统中调用这些服务所用的语法（syntax）。然后，接口的实现负责把 API 定义的具体操作集和对象集映射到协议所定义的抽象服务集上。如果接口定义得很好，那么就可能用这个接口的语法来调用许多不同协议的服务。尽管还很不完善，但这种通用性显然是套接字接口的目标。

毫无疑问，套接字接口的主要概念是套接字（socket）。理解套接字的好方法是把它看作本地应用进程接入网络的地方。套接字接口定义的操作包括创建套接字、将套接字连接到网络，通过套接字发送/接收消息以及关闭套接字。为了简化讨论，我们在此只说明

TCP 中如何使用套接字。

第一步是创建套接字，该步骤通过如下操作实现：

```
int socket (int domain, int type, int protocol)
```

这个操作带有三个参数，原因是套接字接口被设计成通用的，足以支持任意的底层协议族。domain 参数指定了将要使用的协议族（family），PF_INET 表示因特网协议族，PF_UNIX 表示 Unix 管道功能，PF_PACKET 表示直接访问网络接口（即绕过 TCP/IP 协议栈）。type 参数表明通信的语义，SOCK_STREAM 表示字节流，SOCK_DGRAM 是另一种选择，表示面向消息的服务，如 UDP 提供的服务。protocol 参数指明将要用到的特定协议。在我们的例子中，该参数为 UNSPEC，因为 PF_INET 和 SOCK_STREAM 结合起来表示 TCP。最后，socket 的返回值是新创建套接字的句柄（handle），即以后引用该套接字时使用的标识符，它将作为对该套接字的后续操作中的一个参数。

下一步工作取决于计算机是客户端还是服务器。在服务器主机上，应用进程执行被动（passive）打开——服务器表明它已准备好接受连接，但是它并不实际建立连接。为此，服务器调用以下三个操作：

```
int bind(int socket,struct sockaddr * address,int addr_len)
int listen(int socket,int backlog)
int accept(int socket,struct sockaddr * address,int * addr_len)
```

顾名思义，bind 操作将新创建的 socket 与指定的 address 绑定，这个地址是本地（local）参与者（服务器）的网络地址。注意，在使用因特网协议时，address 是一个包含服务器的 IP 地址和 TCP 端口号的数据结构。（正如我们将在第 5 章看到的，端口用于间接地标识进程，它们是一种多路分解键（demux key），其定义见 1.3.1 节。）端口号通常是一些特定于现有服务的众所周知的数字，例如，Web 服务器通常在端口 80 上接受连接。

然后，用 listen 操作定义在指定的套接字上可以有多少个待处理的连接。最后，accept 操作完成被动打开。这是一个阻塞操作，在远程参与者没有建立起连接前，它不会返回，一旦连接成功，它将返回一个与刚建立的连接相对应的新（new）套接字，并且 address 参数将包含远程（remote）参与者的地址。注意，当 accept 返回时，作为参数给定的原始套接字依然存在，并依然对应于这个被动打开，在以后调用 accept 时仍作为参数。

在客户端，应用进程执行主动（active）打开，也就是说，通过调用如下操作表明希望进行通信：

```
int connect(int socket,struct sockaddr * address,int addr_len)
```

该操作直至成功建立 TCP 连接后才返回，此时应用程序可以自由发送数据。在这种情形下，address 中包含远程参与者的地址。实际上，客户端通常只指明远程参与者的地址，让系统自动填写本地信息。鉴于服务器通常在众所周知的端口上监听消息，客户端一般并不关心它自己用哪个端口，操作系统简单地选取一个未用端口即可。

一旦连接建立，应用进程就调用以下两个操作来发送和接收数据：

```
int send(int socket, char * message,int msg_len,int flags)
int recv(int socket,char * buffer,int buf_len,int flags)
```

第一个操作在指定的 socket 上发送 message，而第二个操作从指定的 socket 上接收消息，并放入指定的 buffer。这两个操作都使用一组 flags 来控制操作的特定细节。

1.4.2 应用实例

现在，我们来看一个简单的客户端/服务器程序的实现，该程序利用套接字接口在TCP连接上发送消息。这个程序还用到了其他的Unix网络功能，后文将逐一介绍。这个应用允许一台机器上的用户输入文本并把它发送给另一台机器上的用户，它是Unix中talk程序的一个简化版本，类似于即时通信应用的核心程序。

1. 客户端

我们先从客户端开始，它将远程机器的名字作为参数，调用Unix程序gethostbyname把这个名字转化为远程主机的IP地址，然后构造套接字接口所需的地址数据结构(sin)。注意，这个数据结构表明我们将用该套接字与因特网（AF_INET）连接。在这个例子中，我们用TCP端口号5432作为已知的服务器端口号，它恰好不是分配给其他因特网服务的端口号。建立连接的最后一步是调用socket和connect。一旦connect操作返回，连接就建立了，客户端程序将进入主循环，从标准输入读文本并通过套接字发送出去。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char * argv[])
{
  FILE *fp;
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

  if (argc==2) {
    host = argv[1];
  }
  else {
    fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);
  }

  /* translate host name into peer's IP address */
  hp = gethostbyname(host);
  if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }
```

```
  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);

  /* active open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
  {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }
  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
  }
}
```

2. 服务器

服务器也很简单。首先构造地址数据结构，填上自己的端口号（SERVER_PORT）。但它并不指明IP地址，从而使应用程序可以接受来自本地主机的任何一个IP地址的连接。下一步，服务器执行被动打开连接的预备步骤：创建套接字、将它绑定到本地地址，设置允许连接的最大数目。最后，主循环等待来自远程主机的连接。当发生一个连接时，它就接收并输出从连接上送达的字符。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT  5432
#define MAX_PENDING  5
#define MAX_LINE     256

int
main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;

  /* build address data structure */
```

```
bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = htons(SERVER_PORT);

/* setup passive open */
if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
  perror("simplex-talk: socket");
  exit(1);
}
if ((bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
  perror("simplex-talk: bind");
  exit(1);
}
listen(s, MAX_PENDING);

/* wait for connection, then receive and print text */
while(1) {
  if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
    perror("simplex-talk: accept");
    exit(1);
  }
  while (len = recv(new_s, buf, sizeof(buf), 0))
    fputs(buf, stdout);
  close(new_s);
 }
}
```

1.5 性能

目前，我们的注意力主要集中在网络功能方面。与任何一个计算机系统一样，计算机网络也应具有良好的性能，因为分布式网络计算的效率通常直接依赖于网络传输数据的效率。有一句古老的编程格言，“先使它正确，再使它快速”，虽然在许多情况下都有效，但是在网络中，通常需要“为性能而设计”。因此，了解各种影响网络性能的因素是很重要的。

1.5.1 带宽和时延

网络性能有两种基本度量方法：带宽（bandwidth）（也称为吞吐量（throughput））和时延（latency）（也称为延迟（delay））。网络的带宽是在一段特定的时间内网络所能传输的比特数。例如，一个网络带宽为 10Mbps，就意味着每秒能传输 1 千万个比特。有时候，从传输每个比特所花时间长短的角度来分析带宽也是很有用的。例如，在一个 10Mbps 的网络上，传输每个比特用 0.1μs。

虽然我们可以将网络的带宽作为一个整体来讨论，但有时需要更准确些，例如重点考虑一条物理链路的带宽或者一条进程到进程的逻辑信道的带宽。在物理层，带宽不断提高，无法预测它的上限。直观地说，如果将 1s 看作可测量的一段距离，同时把带宽看作在这段距离中可以容纳的比特数，那么就可将每个比特看作具有一定宽度的脉冲。例如，在一条 1Mbps 的链路上，每个比特 1μs 宽，而在一条 2Mbps 的链路上，每个比特 0.5μs 宽，如图 1-16 所示。发送和接收技术越复杂，每个比特就会变得越窄，这样，带宽就越

大。对于进程到进程的逻辑信道，带宽也受其他因素的影响，包括信道实现软件必须处理并可能转换数据的每个比特的次数。

a) 以1 Mbps传输的比特数（每比特1μs宽）

b) 以2 Mbps传输的比特数（每比特0.5μs宽）

图 1-16 以特定带宽传输的比特可以看作具有一定的宽度

相关主题

带宽和吞吐量

带宽（bandwidth）和吞吐量（throughput）是网络中最容易混淆的两个词。虽然我们可以试图给每个词下一个精确的定义，但重要的是知道别人怎么使用这两个词，并且知道它们如何互换。首先，带宽的字面定义是频带宽度的度量。例如音频级电话线路，支持300～3300Hz范围内的频带，那么就说它的带宽为3300Hz－300Hz＝3000Hz。如果你看到带宽一词在某个情况下以Hz为单位使用，那么它可能是指能容纳信号的范围。

当我们讨论通信链路的带宽时，一般是指链路上每秒所能传输的比特数，有时也称作数据率（data rate）。我们可以说一个以太网的带宽是10Mbps。但是，我们也可以做一个有用的区分，即区分链路上的可用带宽与链路中每秒实际传输的比特数。我们倾向于用吞吐量一词来表示一个系统的测量性能（measured performance）。这样，由于受到实现中各种低效率因素的影响，由一段带宽为10Mbps的链路连接的一对节点可能只达到2Mbps的吞吐量。这就意味着，一台主机上的应用程序能够以2Mbps的速度向另一台主机发送数据。

最后，我们经常谈论应用的带宽需求（requirement），它是应用得以执行而需要在网络上每秒传输的比特数。对某些应用来说，它可能是"所有我能获得的（带宽）"，对另一些应用来说，它可能是某个固定的数值（最好不超过可用链路带宽），而对其他应用来说，它可能是一个随时间变化的数值。本节还将提供有关这个主题的更多信息。

第二个度量性能的尺度是时延，它对应于将一条消息从网络的一端传到另一端所需花费的时间。（像解释带宽时一样，我们也可以将讨论重点放在单条链路的时延或端到端信道的时延上。）时延是严格用时间来测量的。例如，一个横贯北美大陆的网络可能有24ms的时延，即一条消息从北美的一端传到另一端将花费24ms。在很多情况下，更重要的是知道一条消息从网络的一端传到另一端并返回所花费的时间，而不只是单程的时延。我们称它为网络的往返时间（round-trip time，RTT）。

我们通常认为时延有三个组成部分。第一个是光速传播延迟，发生这种延迟的原因是没有什么（包括电线上的一个比特）能比光的传播速度更快。如果知道两点间的距离，就可以计算出光速的时延，然而必须注意光在不同介质中以不同的速度传播：它在真空中以3.0×10^8m/s的速度传播，而在电缆中的传播速度是2.3×10^8m/s，在光纤中的传播速度

是 2.0×10^8m/s。第二个是发送一个数据单元花费的时间，它是网络带宽和运载数据的分组的大小的函数。第三个是网络内部的排队延迟，因为分组交换机在将分组转发出去之前通常需要将它存储一段时间，如 1.2.3 节讨论的那样。因此，我们定义总时延为：

$$\text{Latency}=\text{Propagation}+\text{Transmit}+\text{Queue}$$

$$\text{Propagation}=\text{Distance}/\text{SpeedOfLight}$$

$$\text{Transmit}=\text{Size}/\text{Bandwidth}$$

其中，Distance 是数据需要穿越的线路长度，SpeedOfLight 是光在线路中的有效速度，Size 是分组的大小，Bandwidth 是传输分组的带宽。注意，假设消息只包括 1 比特且我们讨论单条链路（而不是整个网络）的情况，那么 Transmit 和 Queue 的定义就无关紧要了，时延只与传播延迟有关。

带宽和时延结合起来定义了一个给定链路或信道的性能特征，然而它们的相对重要性是依赖于应用的。对有些应用来说，时延比带宽重要。例如，客户发送 1 字节消息给服务器并接收返回的 1 字节消息是受时延限制的。假设在准备应答的过程中没有大量的计算，那么应用程序在一个横贯大陆的 100ms RTT 的信道上与在一个穿过房间的 1ms RTT 信道上的执行有很大不同。尽管信道是 1Mbps 还是 100Mbps 相对来说并不重要，但是前者意味着传输 1 字节的时间（Transmit）是 8μs，而后者的传输时间是 0.08μs。

对比之下，考虑一个数字化图书馆程序要获取一幅 25MB 的图像，带宽越宽，向用户返回图像的速度也就越快。这里，信道的带宽决定性能。为了说明这种情况，假设信道带宽为 10Mbps，那么它将花 20s（$(25\times10^6\times8\text{bit})\div(10\times10^6\text{Mbps})=20\text{s}$）完成图像传输，而图像是在 1ms 还是 100ms 的信道上传输相对来说并不重要，20.001s 与 20.1s 的响应时间之间的差别是可以忽略的。

图 1-17 显示了在不同的情况下时延或带宽如何决定性能。在 RTT 为 1 ～100ms 的范

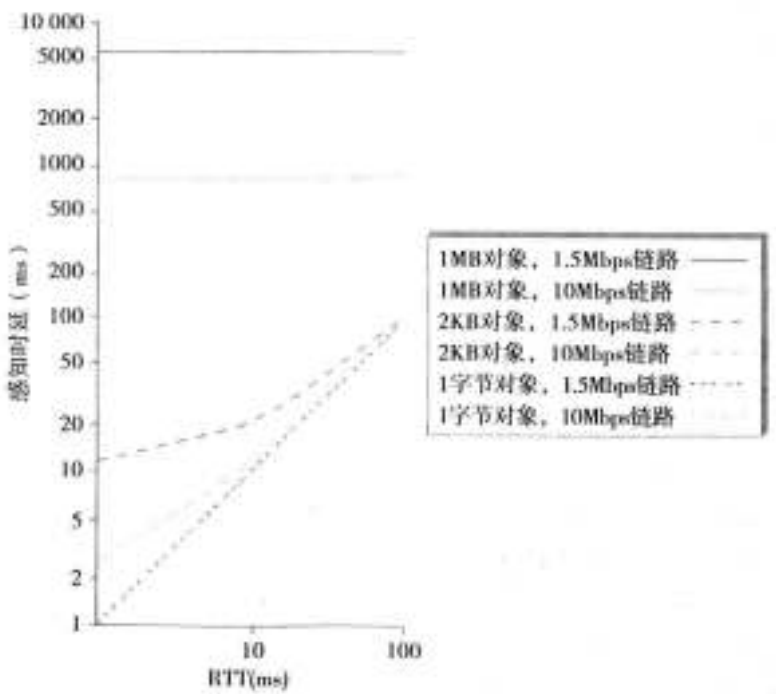


图 1-17 在不同速率的链路上传输不同大小对象的时延（响应时间）与往返时间的对比

围内，链路速度是 1.5Mbps 或 10Mbps 的网络中转移不同大小的对象（1B、2KB、1MB）所花的时间。我们用对数比例来说明相对性能。对于 1 字节的对象（如一次击键），时延基本等于 RTT，因此无法区别 1.5Mbps 和 10Mbps 的网络。对于 2KB 的对象（如一封电子邮件），链路速度对 1ms RTT 网络的影响较大，而对 100ms RTT 网络的影响可忽略。而对于 1MB 的对象（如一幅数字图像），RTT 没有任何影响，即无论 RTT 是多少，都是链路速度决定性能。

注意，本书始终按惯例使用术语时延（latency）和延迟（delay），即用它们来描述完成某一功能的时间，如传输一条消息或转移一个对象所花的时间。当我们特指一个信号从链路的一端传播到另一端所用的时间时，我们使用术语传播延迟（propagation delay）。此外，我们要在讨论的上下文中分清是指单程时延还是往返时间。

另外，计算机正变得如此之快，以至于当我们将其连到网络上时，有时候按指令数英里（instructions per mile）来考虑（哪怕是象征性的）也是有用的。考虑一个每秒执行 10 亿条指令的计算机向一条 RTT 为 100ms 的信道发出一条消息，情况将会怎样。（为了使计算更容易，假设消息穿越 5 000 英里的距离。）如果计算机在等待应答信息的 100ms 内保持空闲，那么它就少执行 1 亿条指令，或者说每英里少执行 20 000 条指令。网络上的这种浪费是很值得我们反思的。

1.5.2 延迟带宽积

讨论这两种度量的乘积也是很有用的，通常称为延迟带宽积（delay×bandwidth product）。直观地说，如果我们将一对进程之间的信道看成一条中空的管道（见图 1-18），时延相当于管道的长度，带宽相当于管道的直径，那么延迟和带宽的乘积就是管道的容积，即在任意给定的时间内正在通过管道传输的最大比特数。换一种说法，如果时延（用时间度量）相当于管道的长度，那么给定每个比特的宽度（也用时间度量），就可以计算出管道中能容纳多少比特。例如，一条横贯大陆的信道，单向时延为 50ms，带宽为 45Mbps，则能够容纳



图 1-18 网络像一个管道

$$50\times10^{-3}\ \text{s}\times45\times10^{6}\ \text{bit/s}=2.25\times10^{6}\ \text{bit}$$

或近似为 280KB 数据。换言之，这个信道（管道）所容纳的字节数相当于 20 世纪 80 年代初一台个人计算机的内存所能容纳的字节数。

构造高性能网络时知道延迟带宽积是很重要的，因为它相当于第一个比特到达接收者之前，发送者必须发送的比特数。如果发送者希望接收者给出比特已开始到达的信号，而且这个信号返回到发送者需要经过另一个信道时延（即我们对信道的往返时延比单程时延更感兴趣），那么发送者在接收到接收者发出的信号之前能够发完多达 RTT×带宽的数据。在管道中的比特称为“在飞行中”，这意味着如果接收者告诉发送者停止发送，那么在发送者准备响应之前，接收者可能已经收到大小为 RTT×带宽的数据了。在上面的例子中，约有 5.5×10^{6} 比特（671KB）数据。另一方面，如果发送者没有填满管道——即发送数量等于 RTT×带宽的数据后才停下来等信号——那么发送者就不能充分利用网络。

注意，大多数情况下我们对 RTT 感兴趣。在这种情况下，我们简单地称之为延迟带宽积，并不具体地说明这个“延迟”表示 RTT（即单程延迟乘以 2）。通常，“延迟带宽

积”中的“延迟”是否表示单程时延由上下文明确地指出。表 1-1 提供了一些典型网络链接的延迟带宽积的实例。

表 1-1 延迟带宽积实例

链路类型	带宽（典型值）	距离（典型值）	往返延迟	RTT×带宽
拨号	56kbps	10km	87μs	5bit
无线局域网	54Mbps	50m	0.33μs	18bit
卫星	45Mbps	35 000km	230ms	10Mb
跨国光纤	10Gbps	4 000km	40ms	400Mb

相关主题

1M 有多大？

当我们用到关于网络的常见单位（MB、Mbps、KB 和 kbps）时，有几点需要澄清。首先要仔细区分比特（bit）和字节（byte）。在本书中，我们总是用 b 表示比特，用 B 表示字节。其次，务必要正确使用兆（M）和千（K）。例如，M 等于 2^{20} 或 10^6，类似地，k 等于 2^{10} 或 10^3。糟糕的是，网络中通常两种定义都用，原因如下。

网络带宽通常用 Mbps 指定大小，它是由控制比特传输速度的时钟速度来决定的，10MHz 的时钟用于以 10Mbps 发送比特。因为 MHz 中的 M 等于 10^6 Hz，Mbps 通常也定义为每秒 10^6 比特（类似地，kbps 是每秒 10^3 比特）。另一方面，当讨论要发送的消息时，通常以 KB 给出它的大小。因为消息存储在计算机内存中，内存通常以 2 的幂次方度量，KB 中的 K 通常表示 2^{10}（类似地，MB 通常为 2^{20}）。当两者放到一起时，时常说在 10Mbps 信道上发送 32KB 消息，这可以解释为以 10×10^6 比特每秒的速度发送 $32\times2^{10}\times8$ 比特。本书中我们采用这种解释，除非另有明确说明。

好在大多数情况下，我们对这种快速且不复杂的计算感到满意，完全可以假想 10^6 就等于 2^{20}（使得 M 的两种定义之间更容易转换），这种近似只造成 5%的误差。我们甚至可以在某些情况下假设 1 字节有 10 比特，这会造成 20%的误差，但对于粗级的估算已经足够好了。

为了帮助你进行粗略计算，100ms 用来表示横穿一个国家的往返时间是一个比较合理的数字，至少在讨论美国时是这样，1ms 则是横穿一个局域网的 RTT 的合理近似值。在前一种情况下，我们把由光速决定的在光纤上传输的往返时间从 48ms 增加到 100ms，因为，正如我们已经说过的，还会有其他造成延迟的原因，如网络内部交换机的处理时间。我们还能确信两点之间的光纤不是一条直线。

1.5.3 高速网络

当今网络的可用带宽正以惊人的速度增长，我们可以乐观地认为它将永远不断地增长下去。这将促使网络设计者开始思考在极限情况下会发生什么事情，或者从另一方面讲，如果带宽可以达到无限，会对网络设计产生什么影响。

尽管高速网络使应用可获得的带宽发生了巨大的变化，但在我们考虑它对网络的未来所产生的方方面面的影响时，要注意不会（not）随着带宽的增加而变化的方面：光速。引用《Star Trek》（《星际旅行》）中 Scotty 的话来说，“你不能改变物理定律”。换言之，“高速”并不意味着时延会和带宽以同样的比率改善，一条贯穿大陆的 1Gbps 链路的 RTT

和一条 1Mbps 链路的 RTT 是一样的，都是 100ms。

虽然时延是固定的，但持续提高带宽非常重要，为了理解这一点，让我们来比较一下当 RTT 均为 100ms 时，在 1Mbps 的网络和在 1Gbps 的网络上传输一个 1MB 的文件分别需要什么。在 1Mbps 的网络中，用 80 个 RTT 来传输文件，每个 RTT 传输文件的 1.25%。对比之下，同样一个 1MB 的文件在 1Gbps 的链路上，甚至不足以填满一个 RTT 的值，它的延迟带宽积为 12.5MB。

图 1-19 说明了两个网络的不同。事实上，需要传输的 1MB 的文件，在 1Mbps 的网络上像一个数据流，而在 1Gbps 的网络上则仅像一个分组而已。为了使问题更清楚，可以认为 1MB 的文件对于 1Gbps 的网络而言，就像 1KB 的分组对于 1Mbps 的网络一样。

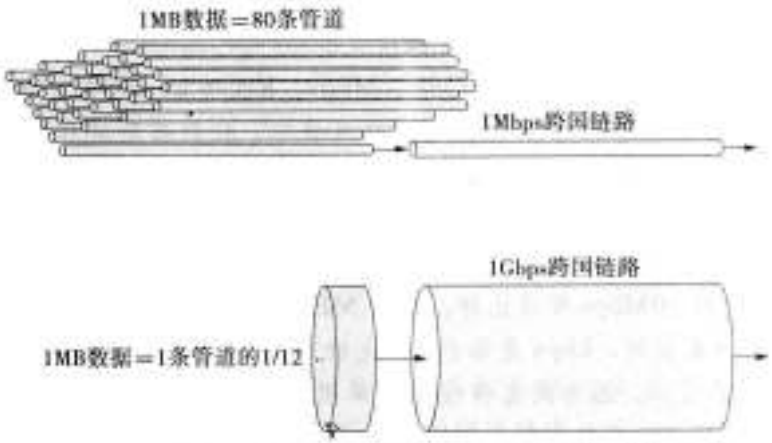


图 1-19 带宽和时延之间的关系。一个 1MB 的文件将占用 1Mbps 链路 80 次，但仅占用 1Gbps 链路 1/12 次

结论 考虑这种情况的另一种方法是，在高速网络上每个 RTT 内能够传输更多的数据，传输速度如此高，以至于单个 RTT 都变成了很大的时间量。因此，虽然不必考虑用 101 个 RTT 与用 100 个 RTT 的文件传输之间的区别（相对差别仅为 1%），但 1 个 RTT 与 2 个 RTT 之间的区别却很大——增加了 100%。换句话说，我们在网络设计时考虑的主要因素应是时延而不是吞吐量。

或许理解吞吐量和时延的关系的最好方法还是从基础开始。在网络上可获得的有效端到端吞吐量由下面的简单关系式给出：

$$\text{Throughput} = \text{TransferSize} / \text{TransferTime}$$

其中 TransferTime 不仅包括本节前面讲到的单程 Latency，而且还包括请求或建立传输的附加时间。通常，我们用下式表达它们的关系：

$$\text{TransferTime} = \text{RTT} + 1/\text{Bandwidth} \times \text{TransferSize}$$

本式中，我们用 RTT 表示在网上发一条请求消息并返回数据的时间。例如，考虑这样的情况：在 1Gbps 的网络上，用户要获取一个 1MB 的文件，其往返时间为 100ms。TransferTime 包括 1MB 的传输时间（1/1Gbps×1MB=8ms）和 100ms 的 RTT，总的传输时间为 108ms。这意味着有效吞吐量不是 1Gbps，而是

$$1\text{MB}/108\text{ms} = 74.1\text{Mbps}$$

显然，传输更大量的数据有助于提高有效吞吐量，而在极限条件下，传输数据量无限大

时，有效吞吐量将接近网络带宽。另一方面，有时不得不承受多于一个 RTT 的传输时间（例如重传丢失的分组），这会降低任何有限数据量传输的有效吞吐量，尤其是小数据量传输的有效吞吐量。

1.5.4 应用程序性能需求

本节的讨论以网络运行的效率为中心展开，也就是说，我们讨论一条给定的链路或信道能支持什么。在未加说明的情况下，假设应用程序有简单的需要——它们所需的带宽为网络所能提供的带宽。这对于前面提到的数字化图书馆程序获取一幅 25MB 图像的例子来说，显然是成立的，可用带宽越宽，程序将图像回送给用户的速度就越快。

然而，一些应用能指明它们需要的带宽上限，视频应用程序是一个典型例子。假设你想播放一个视频图像流，而它的大小是标准电视图像的 1/4，也就是说，它的分辨率是 352 像素×240 像素。如果每一个像素是由 24 比特的信息表示的，就像 24 位色彩一样，那么每一帧的大小将是

$$(352\times240\times24)/8=247.5\text{KB}$$

如果一个视频应用程序需要每秒发送 30 帧，那么它可能要求 75Mbps 的吞吐量。这种应用对网络提供更多带宽的能力没有兴趣，因为在一个给定时间周期内它只有那么多数据要发送。

不幸的是，实际情况并不像这个例子所描述的那么简单。因为在一个视频流中，任何两个相邻帧之间的差别通常非常小，可以通过只传输相邻帧之间的差别而将视频压缩。每一帧也可以被压缩，因为人眼无法察觉图像中的所有细节。这种压缩视频的流动不是匀速的，而是根据动作的数量、图片的细节以及所使用的压缩算法随着时间变化。因此，可以说平均带宽需求是多少，但瞬时速率则可能大一些或小一些。

关键的问题是计算平均值时使用的时间间隔。假设该视频应用实例能被压缩到平均只需 2Mbps。如果它在 1s 内发送 1Mb，而在下一秒内发送 3Mb，那么这 2s 间隔内它发送的平均速率是 2Mbps。然而，这对设计在任一秒内不超过 2Mb 的信道而言没有多少帮助。显然，只知道一个应用所需的平均带宽有时是不够的。

然而，一般我们可以为这种应用所发送的最大突发量设定一个上界。这种突发量可由某段时间内保持的峰值速率来描述，或者可以被描述为在转为平均速率或一个更低的速率之前能以峰值速率发送的字节数。如果这个峰值速率比可用信道容量高，那么超出的数据不得不被放入某处的缓冲区，以备以后发送。知道突发量的大小可以使网络设计者分配足够的缓冲区来容纳它。我们将在第 6 章更准确地描述突发流量这个主题。

正如应用对带宽的需求可能不是“能得到的全部带宽”，应用对延迟的要求也可能不是“尽可能少的延迟”那么简单。说到延迟，有时网络的单程时延是 100ms 还是 500ms 并不如分组间时延的变化那么重要。时延的这种变化称为抖动（jitter）。

考虑源端每 33ms 发送一个分组的情况，这正是视频应用中每秒传输 30 帧的情况。如果帧恰好每隔 33ms 到达目的地，那么我们可以推断，网络中每个分组经过的延迟恰好相同。然而，如果分组到达目的地的间隔——有时称为分组间距（inter-packet gap）——是变化的，那么分组的序列所经历的延迟一定也是变化的，称为网络在分组流之间有抖动，如图 1-20 所示。这种变化通常不是由单条物理链路引起的，而是在多跳分组交换网中由分组经历的不同排队延迟引起的。排队延迟相当于本节前面定义的延迟中的 Queue 部分，它是随时间变化的。



图 1-20 由网络引起的抖动

为了理解与抖动相关的内容，假设网络中传输的分组含有视频帧，为了在屏幕上显示这些帧，接收器需要每隔 33ms 接收一个新帧。如果一个帧提前到达，那么接收器会将它保存到需要显示的时候。然而，如果一个帧迟到了，那么接收者将没有所需的帧来及时刷新屏幕，视频质量就会下降，画面可能不平滑。注意，我们不必消除抖动，只要知道它的最坏程度即可。因为如果接收器知道一个分组所经历的延迟的上界和下界，就可以将视频的开始（即显示第一帧）推迟足够长的时间，以保证将来需要时总有帧显示。接收者通过将这个帧存放在缓冲区中实现帧延迟以有效地平滑抖动。第 9 章我们会再讨论抖动问题。

1.6 小结

计算机网络，特别是因特网，在过去的 30 年中经历了突飞猛进的发展，现在已经能够提供从管理商业到提供娱乐到建立社交网络的广泛服务。能够取得这样的发展很大程度上归功于计算机网络的通用性，特别是能够通过编写运行在不太昂贵的高性能计算机上的软件来为网络添加新的功能。因此，本书最主要的目标就是按照这样一种方法来描述计算机网络：当你读完本书时将会感觉到，如果有一支程序员队伍由你指挥，你就能够从头建立一个具备完整功能的计算机网络。本章为实现这个目标奠定了基础。

实现这个目标的第一步是仔细甄别我们从网络中真正希望得到什么。例如，网络首先必须在一组计算机之间提供高性价比的可扩展的连接。这是通过节点和链路的嵌套互联来完成的，并通过使用统计多路复用来共享硬件基础设施。这样就得到一个分组交换网，然后我们在它上面定义一系列进程到进程的通信服务。

第二步是定义一个分层的体系结构，作为我们的设计蓝图。这个体系结构的核心对象是网络协议。协议既向较高层协议提供通信服务，同时又定义与另一些机器上运行的对等实体之间交换信息的格式和含义。我们简单地探讨了两种使用最广泛的体系结构：OSI 体系结构和因特网体系结构。无论是组织结构还是实例来源，本书更贴近因特网体系结构。

第三步是实现网络协议和应用程序，它通常由软件来完成。不论协议还是应用，都需要一个接口，它们通过这个接口调用网络子系统中其他协议的服务。在应用程序和网络子系统之间最常用的接口是套接字接口，但它与网络子系统内部常用的接口有一些微小差别。

最后，网络作为一个整体必须提供高性能，其中，我们最关心的两个性能指标是带宽和吞吐量。正如我们将在后面几章中会看到的那样，正是这两个指标的乘积——延迟带宽积——在协议设计中起着决定性作用。

接下来会发生什么：云计算

毫无疑问，计算机网络正成为许多人日常生活的一部分。从 40 多年前像 APPANET（通过长途电话线连接大型计算机）这样的实验性系统开始，到如今计算机网络已经变成

了我们生活中普遍存在的一部分。网络已经变成了大型产业，哪里有大型产业，哪里就有众多的使用者。在这种情况下诞生的计算行业已经越来越多地参与到计算和通信的集成中：电话和有线电视运营商也认识到不仅要发展语音和电视业务，还要开拓传输各类数据的市场；另外，也许最重要的是大量企业家创建了基于因特网的新应用和新服务，如 IP 语音（Voice Over IP，VOIP）、在线游戏、虚拟世界、搜索服务、内容托管、电子商务等。值得注意的是，当今在“云计算”中最响亮的名字之一——Amazon.com，它赢得当前显赫地位的方法是，先采用因特网技术出售图书等商品，然后将他们的计算架构作为网络上的服务提供给其他人使用。

多年前，网络的一个合理目标也许是为每一个家庭提供网络接入，但至少是在发达国家，这个过程已经很久远了。无处不在的网络现在允许用户从任何地方访问，包括在飞机和火车上，以及在不断增加的各类设备上。因特网在固定主机时期经历了巨大演变，然后是个人电脑时期，现在互联起来的设备包括移动电话以及传感器（传感器也有可能是移动的）等更小的设备。因此，似乎很清楚的一点是，因特网不得不继续扩展以支持比现在多几个数量级的设备，而且很多设备是移动的，它们可能通过质量变化极大的无线链路断断续续地连接到网络。同时，这些设备将被连接到大型数据中心，其中包含几万个处理器和许多 PB 字节的存储空间，数据中心将存储和分析所产生的数据，并希望实现有助于我们管理日常生活的更强大的应用。我们持有的设备通常只是接入“云”（云是用于存储和处理文档、照片、数据和社交网络等内容的不固定的机器集合）的一种手段，而且希望能够从任何地方接入。

有关网络未来的预言有一个趋势，就是这些言论过几年就会显得很傻（好多高调的预言都未能成真，如因特网将濒临崩溃）。我们能够确信的是依然存在大量技术挑战——连通性、可管理性、可扩展性、可用性、性能、可靠性、安全、公平性、成本效益等。当前的技术水平与很多人认为的全球无处不在的异构网络之间已经很接近了。换言之，网络领域非常活跃，有很多有趣的问题需要解决，这些问题以及解决问题的工具正是本书的重点。

扩展阅读

计算机网络并不是第一个进入社会日常生活的面向通信的技术。例如，20 世纪初电话的出现，以及之后 50 年代时电视的普及。当我们考虑网络的未来，即它将分布得多么广和我们将如何使用它时，研究它的发展历史是很有益的。我们的第一篇参考文献是研究网络的一个好的起点（整篇文章描述了电信发展过程中前 100 年的历史）。

第二篇和第三篇文章分别是有关 OSI 和因特网体系结构的开创性文章。最后两篇文章不是针对网络的，但却提前阐明了本书中“系统方法”的观点。Saltzer 等人的文章提供并描述了系统设计中使用最广泛的规则之一，即端到端观点（end-to-end argument），这篇文章直到今天依然保持着较高的引用率。Mashey 的文章描述精简指令集（Reduced Instruction Set Computer，RISC）体系结构隐含的思想，正如我们即将揭示的，将功能置于一个复杂系统的合适位置正是系统设计所要解决的问题。

- Pierce，J. Telephony—A personal view. *IEEE Communications* 22（5）：116-120，May 1984.
- Clark，D. The design philosophy of the DARPA Internet protocols. *Proceedings of the SIGCOMM'88 Symposium*，pages 106-114，August 1988.

- Saltzer, J., D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems* 2 (4): 277-288, November 1984.
- Mashey, J. RISC, MIPS, and the motion of complexity. *UniForum 1986 Conference Proceedings*, pages 116-124, February 1986.

还有几篇文章介绍了计算机网络：Stallings 给出有关这个题目的百科全书式的叙述，重点在 OSI 分层结构的低层 [Sta07]；Comer 给出对因特网体系结构的综述 [Com05]。

如果你要用更广阔的眼界来看计算机网络，有两本书是必须读的，其中一本描述过去，另一本展望未来。第一本是 Holzmann 和 Pehrson 的 *The Early History of Data Networks* [HP95]。令人惊讶的是，这本书中的很多思想是 18 世纪提出的。第二本书 *Realizing the Information Future: The Internet and Beyond*，由美国国家研究委员会的计算机科学和电信部 [NRC94] 编写。

在本书中，我们努力将系统方法用于计算机网络领域。我们推荐 Saltzer 和 Kaashoek 的 *general treatment of computer systems* [SK09]，这本书讲解适用于网络以及其他系统的很多重要原理。操作系统对网络的很多方面都是非常重要的，Tanenbaum [Tan07] 提供了对 OS 概念的介绍。

若要从头追溯因特网的发展历史，读者应当仔细阅读因特网的 RFC（Request for Comments）系列文档。文档内容从 TCP 规范到愚人节笑话，包罗万象，获取地址为 http://www.ietf.org/rfc.html。例如，TCP、UDP 和 IP 的协议规范分别是 RFC793、768 和 791。

为了更好地了解因特网的哲学和文化，推荐阅读两篇文章，这两篇文章的叙述也非常生动有趣。Padlipsky 给出对网络早期发展的描述，包括因特网和 OSI 体系结构的详细比较 [Pad85]。如果想知道此后在因特网工程任务组（Internet Engineering Task Force，IETF）中真实发生的事情，我们推荐 Boorsook 的文章 [Boo95]。

有大量的文章讨论有关协议实现的各个方面。有关协议实现最好从理解两个完整的协议实现环境开始：来自 System V Unix 的流机制 [Rit84] 和 *x*-kernel [HP91]。另外，[LMKQ89] 和 [SW95] 描述了使用广泛的 Berkeley Unix 中 TCP/IP 的实现。

更为普遍的是，有大量的工作致力于解决构造和优化协议实现的问题。Clark 是最早讨论模块化设计与协议性能之间关系的人之一 [Cla82]。后来的文章则介绍在构造协议代码时向上调用的使用 [Cla85] 及研究 TCP 的处理开销 [CJRS89]。最后，[WM87] 讲述如何通过选择适当的设计和实现方案来获得效率。

有几篇文章介绍用于提高协议性能的具体技术和机制。例如，[HMPT89] 讲述 *x*-kernel 中用到的一些机制，[MD93]、[VL87] 和 [DP93] 也描述了其他提高协议性能的技术。同时，[BG93] 和 [NYKT94] 讨论在并行处理器上运行的协议的性能，在这种环境中，加锁是关键的问题。

最后，我们用一组不断更新的参考文献作为每一章中“扩展阅读”一节的结束，也就是 WWW 上的一组 URL 地址，从中可以了解与该章所讨论主题有关的更多东西。由于参考文献是不断更新的，不可能无限期地保持不变。所以，我们把每章结尾的网址集中在这样一些站点上，这些站点提供软件、服务、报告工作组或标准化组织正在开展的活动。换言之，对于那些不易于用标准引文引用的材料，我们只给出其 URL。本章给出 3 个网址：

- http://mkp.com/computer-networks：有关本书的信息，包括附录、补遗等。
- http://www.ietf.org/：有关 IETF 和它的各个工作组的信息。

- http://dblp.uni-trier.de/db/index.html：可搜索有关网络研究论文的文献目录。

习题

1. 使用匿名 FTP 连接到 ftp.rfc-editor.org（in-notes 目录），获取 RFC 索引，同时获取 TCP、IP 和 UDP 的协议规范。
2. Unix 工具 whois 可用来查找一个组织对应的域名，或查找域名对应的网络地址。阅读 whois 的主页文档并试用它。对于初学者，试用 whois princeton.edu 和 whois Princeton，也可以通过 http://www.internic.net/whois.html 研究 whois 接口。
3. 计算在下列情况下传输一个 1 000KB 的文件所需的总时间，假设 RTT 为 50ms，分组长度为 1KB，在数据发送前的初始“握手”时间为 2×RTT。
   (a) 带宽为 1.5Mbps，数据分组可连续发送。
   (b) 带宽为 1.5Mbps，但每发送完一个分组后必须等一个 RTT 后再发送下一个分组。
   (c) 带宽是“无限的”，这意味着我们可以认为传输时间为 0，且每个 RTT 最多发送 20 个分组。
   (d) 带宽是无限的，在第一个 RTT 内能发送一个分组（2^1-1），在第二个 RTT 内能发送两个分组（2^2-1），在第三个 RTT 内能发送四个分组（2^3-1），依此类推。（我们会在第 6 章中给出指数增长的原因。）
4. 计算在下列情况下传输一个 1.5MB 的文件所需的总时间，假设 RTT 为 80ms，分组长度为 1KB，在数据发送前的初始“握手”时间为 2×RTT。
   (a) 带宽为 10Mbps，数据分组可连续发送。
   (b) 带宽为 10Mbps，但发送完每个分组后必须等一个 RTT 后再发送下一个分组。
   (c) 链路允许无限快地传输，但限制带宽使每个 RTT 最多能发送 20 个分组。
   (d) 和 (c) 一样，传输时间为 0，但在第一个 RTT 内能发送一个分组，在第二个 RTT 内能发送两个分组，在第三个 RTT 内能发送四个分组（2^3-1），依此类推。（我们会在第 6 章中给出指数增长的原因。）
5. 考虑一个长度为 4km 的点到点链路。对一个 100 字节的分组，带宽为多大时传播延迟（速度为 2×10^8 m/s）等于传输延迟？对于 512 字节的分组，情况如何？
6. 考虑一个长度为 50km 的点到点链路。对一个 100 字节的分组，带宽为多大时传播延迟（速度为 2×10^8 m/s）等于传输延迟？对于 512 字节的分组，情况如何？
7. 邮政地址的哪些特性可能被网络寻址方案借鉴使用？你希望找到哪些差别？电话号码的哪些特性可能被网络寻址方案借鉴使用？
8. 地址的一个特性是唯一性，如果两个节点有相同的地址便无法区分。网络地址还可能有哪些有用的特性？你能想象网络（或邮政、电话）地址可以不唯一的任何情况吗？
9. 给出一个适合使用多点播送地址的例子。
10. STDM 是语音电话网络有效的多路复用形式，而 FDM 是电视和广播有效的多路复用形式，它们在通信模式上有什么不同？通用的计算机网络中不使用这两种形式是由于性价比低的原因吗？
11. 在 1Gbps 的链路上 1 个比特有多“宽”？假设传播速度为 2.3×10^8 m/s，在铜线上 1 个比特有多长？
12. 在一个 yMbps 的链路上传输 xKB 的数据需要花费多长时间？用 x 与 y 比率的形式给出答案。
13. 假设在地球和新的月球定居地之间架设了一条 1Gbps 的点到点链路。从月球到地球的距离大约是 385 000km，而且数据在链路上以光速传播，即 3×10^8 m/s。
   (a) 计算链路的最小 RTT。
   (b) 使用 RTT 作为延迟，计算链路的延迟带宽积。
   (c) 在 (b) 中计算的延迟带宽积的意义是什么？
   (d) 在月球基地上的一部照相机拍摄了一张地球的照片，并以数字的形式存入磁盘。假设地球上的任

务控制中心希望下载最新的图像，大小是25MB。计算从发出请求到传输完毕耗费的最小时间。

14. 假设在地球和一个火星探测车之间架设了一条128kbps的点到点链路。从火星到地球的距离（当它们离得最近时）大约是55Gm，而且数据在链路上以光速传播，即3×10^8m/s。

(a) 计算链路的最小RTT。

(b) 计算链路的延迟带宽积。

(c) 探测车上的一部照相机拍摄周围的照片，并发送回地球。计算从拍完一幅图像到这幅图像到达地球上的任务控制中心所用的时间。假设每幅图像的大小为5MB。

15. 对于下面列出的在远程文件服务器上的操作，讨论它们是对延迟敏感还是对带宽敏感。

(a) 打开文件。

(b) 读出文件的内容。

(c) 列出目录中的内容。

(d) 显示文件的属性。

16. 计算下列情况的时延（从第一个比特发送到最后一个比特接收）：

(a) 100Mbps以太网，其路径上有一个存储转发式交换机，分组长度为12 000比特。假设每条链路的传播延迟为10μs，并且交换机在接收完分组后立即转发分组。

(b) 有三个交换机，其他同(a)。

(c) 同(a)，但是假设交换机实现“直通式”交换：可以在收到分组的前200比特后就开始转发该分组。

17. 计算下列情况的时延（从第一个比特发送到最后一个比特接收）：

(a) 1Gbps以太网，其路径上有一个存储转发交换机，分组长度为5 000比特。假设每条链路的传播延迟为10μs，并且交换机在接收完分组后立即开始转发该分组。

(b) 同(a)，但是有三个交换机。

(c) 同(b)，但是假设交换机实现“直通式”转发：可以在收到分组的前128比特后就开始转发该分组。

18. 计算下列情况的有效带宽。对于(a)和(b)，假设要发送的数据来源稳定；对于(c)，只计算12个小时的平均值。

(a) 10Mbps以太网通过三个存储转发交换机转发。同习题16(b)中的情况。交换机在一条链路上接收数据的同时可以在另一条链路上发送数据。

(b) 同(a)，但是发送方在每发12 000比特数据分组后必须等待一个50字节的确认分组。

(c) 100个DVD（每个4.7GB）整夜（12小时）传输。

19. 计算下列链路的延迟带宽积。使用单向延迟，按从第一个比特发送到第一个比特接收计算。

(a) 100Mbps以太网，延迟10μs。

(b) 100Mbps以太网，有一个存储转发交换机，同习题16(b)中的情况，分组长度为12 000比特。每条链路的传播延迟为10μs。

(c) 1.5Mbps T1链路，贯穿大陆的单向延迟为50ms。

(d) 通过一个地球同步轨道卫星的1.5Mbps T1链路，卫星高度为35 900km。唯一的延迟为从地球到卫星的往返光速传播延迟。

20. 如图1-21所示，主机A和B分别通过100Mbps链路连接到交换机S上。每条链路的传播延迟为20μs。S是一个存储转发式设备，它在收到一个分组35μs后再开始将其转发。计算从A到B发送10 000比特所需的总时间。

(a) 作为一个分组。

(b) 作为两个5 000比特的分组一个紧接另一个发送。



图1-21 习题20图

21. 假设某主机有一个1MB的文件要发送给另一台主机。文件用1s CPU时间压缩50%，或者用2s压缩60%。
(a) 计算当带宽为多少时，两种压缩选择的压缩时间+传输时间的值相等。
(b) 解释为何时延不影响你的答案。

22. 假设某一通信协议的每个分组用于首部和建立帧的信息开销为50字节。我们利用此协议来发送10^6字节的数据，但是，分组中一旦有一个字节被破坏，包含该字节的整个分组将丢失。给出分组长度分别为1 000字节、10 000字节和20 000字节时，信息开销与丢失字节的总数。分组长度为哪个值时是最优的？

23. 假设在一条由信源、信宿、7条点对点链路和5个交换机组成的网路上传输n字节的文件。假设每条链路的传播延迟为2ms，带宽为4Mbps，而且交换机支持电路交换和分组交换。你可以把文件分割成1KB的分组或在交换机之间建立起一个电路并把文件作为一个连续的比特流发送。假设每个分组有24字节的分组首部信息和1 000字节的有效载荷，而且每个交换机在完全收到一个分组后对分组进行存储转发的过程会引起1ms的延迟。分组可以被连续发送而不需要等待确认，建立电路需要发送1KB的消息，在路径上往返一次在每个交换机产生1ms的延迟。假设交换机不会给通过电路的数据带来延迟。也可以假设文件大小是1 000字节的整数倍。
(a) 文件大小为多少个字节时，电路交换在网络上发送的总字节数少于分组交换？
(b) 文件大小为多少个字节时，电路交换使整个文件到达目的地时产生的总延迟小于分组交换？
(c) 以上结果是与路径上交换机的数目有什么关系？与链路的带宽有什么关系？与分组首部大小和分组大小之比又有什么关系？
(d) 本题给出的网络模型能否准确反映电路交换和分组交换的优缺点？是否忽略了使这两种交换方式受到质疑的重要因素？如果有，这些因素是什么？

24. 考虑一个闭环网络（如令牌环），带宽为100Mbps，传播速度为2×10^8m/s。假设节点不产生延迟，环的周长为多少时恰好可容纳一个1500字节的分组？如果每100m一个节点，且每个节点的延迟为10比特，环的周长应为多少？

25. 根据带宽、延迟和抖动，比较语音传输和实时音乐传输对信道的需求。有哪些必须要改进的地方？改进幅度大约是多少？可否放宽对任一种信道的需求？

26. 下列情况下，假设不对数据进行压缩，尽管这在实际应用中几乎是不可能的。对于(a)～(c)，计算实时传输需要的带宽：
(a) 视频的分辨率为640像素×480像素，3B/像素，30帧/s。
(b) 视频的分辨率为160像素×120像素，1B/像素，5帧/s。
(c) CD—ROM音乐，假设CD播放75分钟，大小为650MB。
(d) 假设一个传真机以每英寸72像素的分辨率发送一幅8英寸×10英寸的黑白图像。在14.4 kbps的调制解调器上需要传输多长时间？

27. 下列情况下，和上题一样，假设不对数据进行压缩。计算实时传输需要的带宽：
(a) HDTV高清晰度视频，分辨率为1 920像素×1 080像素，24位/像素，30帧/秒。
(b) 8比特POTS（普通的电话服务）语音音频，采样频率为8KHz。
(c) 260比特GSM移动语音音频，采样频率为50Hz。
(d) 24比特HDCD高保真音频，采样频率为88.2KHz。

28. 根据平均带宽、峰值带宽、时延、抖动和丢失容限，讨论与下列应用相关的性能需求：
(a) 文件服务器。
(b) 打印服务器。
(c) 数字化图书馆。
(d) 远程气象设备定时监视。
(e) 语音。

(f) 候车室视频监视。

(g) 电视广播。

29. 假设共享介质 M 以循环方式向主机 A_1、A_2、…、A_N 提供传输一个分组的机会，没有分组要传的主机立即放弃 M。它与 STDM 有何不同？与 STDM 相比，这种方式对网络的利用率如何？

★ 30. 考虑在链路上传输文件的一个简单协议。经过一些初始协商后，A 向 B 发送长度为 1KB 的分组，然后 B 回答一个确认信息。A 在发送下一个数据分组前都要等待 ACK，这就是我们熟知的停止和等待(stop-and-wait)。丢到的分组被认为已丢失并重传。

(a) 在不考虑分组丢失和重复的情况，说明为何在分组首部中不需要包括任何“序号”数据。

(b) 假设链路偶尔会丢失分组，但实际到达的分组总是按发送的顺序到达。对于 A 和 B 而言，用 2 比特表示序号（即 N mod 4）是否足以检测并重发任何丢失的分组？用 1 比特序号是否足够？

(c) 现在假设链路可以无序地传递数据，而且有时一个分组会在它后面的分组已到达长达 1 分钟以后才被传输。这种情况下，对序列号的要求有哪些改变？

★ 31. 假设主机 A 和主机 B 由一条链路相连。主机 A 以一定的速率持续地传输一个高精度时钟中的当前时间，其速度快到可以消耗整个可用带宽。主机 B 读出这些时间值并把它们写成它自己的与主机 A 时钟同步的本地时钟的时间对。假设链路有如下情况，定性地给出主机 B 输出的例子。

(a) 高带宽，高时延，低抖动。

(b) 低带宽，高时延，高抖动。

(c) 高带宽，低时延，低抖动，偶尔丢失数据。

例如，一条无抖动链路，带宽高到足够每隔一个时钟脉冲输出一次，一个时钟周期可能产生像(0000，0001)、(0002，0003)、(0004，0005)这样的结果。

32. 获取并构建如书中所示的 simplex-talk 套接字程序实例。分别在独立的窗口中启动一个服务器和一个客户端。当第一个客户端运行时，再启动连接到同一个服务器上的 10 个其他客户端；这些其他客户端很有可能在后台被启动，它们的输入重定向来自一个文件。这 10 个客户端会发生什么情况？它们的 connect() 操作会失败、超时还是成功？其他的调用是否会被阻塞？现在让第一个客户端退出，会出现什么情形？将服务器的 MAX_PENDING 设置为 1，再试一次。

33. 修改 simplex-talk 套接字程序，使其客户端每次给服务器发送一行，然后服务器把这一行发送回客户端。客户端（和服务器）现在必须轮流调用 recv() 和 send()。

34. 修改 simplex-talk 套接字程序，使其使用 UDP 而不是 TCP 作为传输协议。你必须在客户端和服务器上同时将 SOCK_STREAM 修改为 SOCK_DGRAM。然后，在服务器端删除对 listen() 和 accept() 的调用，将结尾的两个嵌套循环用一个单循环代替。这个单循环用套接字 s 调用 recv()。最后，观察当两个 UDP 客户端同时连接到同一个 UDP 服务器时会发生什么情况，并和 TCP 的情况进行比较。

35. 考察可以为 TCP 连接设置哪些不同的选项和参数（在 Unix 下执行 man tcp）。试验用不同的参数设置，观察它们如何影响 TCP 的性能。

36. Unix 的工具 ping 可以用来找出到各种因特网主机的 RTT 值。阅读 ping 的主页，并使用它找出到新泽西州的 www.cs.princeton.edu 以及加利福尼亚州的 www.cicso.com 的 RTT。在一天的不同时间里测量 RTT 的值，并比较结果。对于这些差异将如何解释？

37. 可以用 Unix 中的工具 traceroute 或 Windows 中相应的 tracert，查看消息在路由选择时所经过的路由器序列。用它来查看从你的站点到某些其他站点的路径。其中的跳数和由 ping 得知的 RTT 时间有何关系？跳数和地理距离有何关系？

38. 用上题中的 traceroute 画出你所在组织的一些路由器图示（或证明没有使用路由器）。

开始连接

过度地考虑未来是错误的。命运像一串链条，一次只能处理其中的一个环节。

——温斯顿·丘吉尔

问题：连接到网络

在第1章中我们看到网络是由节点间相互连接的链路组成的，我们面临的一个基本问题就是如何把两个节点连接在一起。我们也介绍了“云”的概念，它是一个网络，但无需考虑其内部的复杂性。我们也需要考虑一台主机连接到云的问题。对于每个因特网服务提供商（ISP）来说，当它想把一个新客户连接到网络中时，需要考虑类似的问题：如何把多个节点连接到ISP的云中。

无论是想要构造一个只有两个节点和一条链路的简单网络，还是把第十亿台主机连接到现存的网络（如因特网）中，我们都需要考虑一些事情。首先需要连接物理介质。介质可以是铜线、光纤或者是可以在其中传输电磁信号（如无线电波）的无线介质（例如空气或自由空间）。它可能覆盖一个小的区域（如一栋办公大楼），或一个大的区域（如大洲）。然而，用一种合适的介质连接两个节点仅仅是第一步。在节点能成功地交换分组之前，必须先解决下面五个问题。

第一个问题是对传送到介质上的比特编码（encoding），使其能被接收主机理解。第二个问题是把在链路上传输的比特序列描述为完整的消息，以便传送到端节点，这称为组帧（framing）问题，而传给端主机的消息通常称为帧（frame），有时也叫分组（packet）。第三个问题，因为在传输过程中帧有时会出错，所以有必要检测这类差错并且采取适当的行动，这是差错检测（error detection）问题。尽管帧一次又一次地出错，但在这种情况下，还是要建立一条看起来可靠的链路，这是第四个问题。最后一个问题是在多台主机共享一条链路（尤其是无线链路）的情况下，必须协调对该条链路的访问，这是介质访问控制（media access control）问题。

虽然编码、组帧、差错检测、可靠传输和介质访问控制这五个问题可以在理论上讨论，但是它们是在不同网络技术中以不同方式解决的很实际的问题。本章在以下三个特定的网络技术下考虑这些问题：点到点链路、载波监听多路访问（Carrier Sense Multiple Access，CSMA）网络（以太网是其最著名的例子）和无线网络（802.11是应用最广泛的标准）。本章的目的是纵览实用的网络技术并探讨这五个基本问题，我们将会看到如何把各种不同的物理介质连接起来构造鲁棒性和可扩展性俱佳的网络。

2.1 连接概览

正如在第1章中所看到的，网络由两类硬件构件组成：节点（node）和链路（link）。本章关注如何构造有用的连接，尤其是在包含着数百万连接的大规模可靠网络中。

大规模网络的操作者处理的是通过冰箱大小的路由器所连接起来的数百或数千公里长的链路，网络的典型用户接入链路的方式大多是将一台计算机连接到全球因特网。这种连

接可能是在咖啡店中的无线连接（Wi-Fi），也可能是在办公楼或大学中的以太网连接。对于正在快速增长中的网络用户，可能是由电信公司或ISP提供的光纤连接，也有许多用户在使用铜缆连接。幸运的是，有许多不同的连接策略对于协议栈高层来说是可靠并有用的。本章将探讨这些策略。

图2-1显示了当前因特网中典型终端用户采用的各种连接方式。左边是各种各样的终端用户设备，包括移动电话、PDA、计算机等，它们通过各种方式连接到因特网服务提供商。其接入方式可能是上面提到的任何连接方式，也可能是其他连接方式。在该图中它们采用同样的连接方式，即通过一条直线连接到路由器。同时，图中有一些链路把ISP内部的路由器相互连接在一起。图中还有一条链路把ISP连接到“因特网其余部分”，它包含许多其他ISP及其连接着的主机。这些链路虽然画得并不漂亮，但网络体系结构（正如1.3节中所讨论）的作用之一是提供复杂链路的简单抽象。基本思想就是你的笔记本或智能手机不必考虑它们被接入了什么类型的链路，唯一要考虑的就是它接入了因特网。类似地，路由器不必考虑它通过什么类型的链路与其他路由器连接起来，只要知道它在链路上发送了一个分组后，分组能够按预期到达链路的另一端。

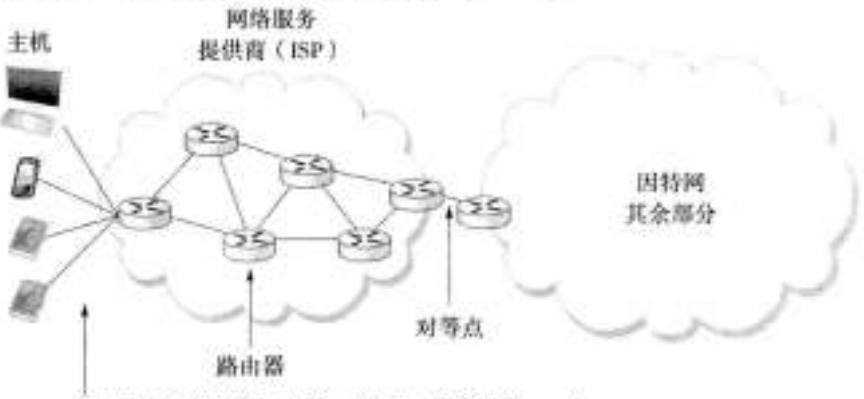


图2-1 因特网终端用户概览

如何使不同的链路对于终端用户和路由器来说是一样的呢？关键是我们必须处理存在于现实世界中的所有物理约束和链路缺陷。本章中我们会对这些问题进行概述。首要的问题是链路是能够传输信号（例如无线电波或其他类型的电磁波）的物理介质，但我们实际需要发送的是比特。本章后面会介绍如何对比特进行编码以便在物理介质中传输，接下来会介绍上面提到的其他问题。本章最后会介绍如何在各种链路上发送完整的分组，而不考虑其采用的物理介质。

相关主题

链路容量和香农定理

在信号处理和信息论的相关领域中，研究者们已做了许多工作，其中包括在经过一段距离后信号如何衰减，以及一个给定的信号能有效地运载多少数据等。这一领域中最著名的工作是一个称为香农-哈特雷定理（Shannon-Hartley theorem）[⊖]的公式。简单地说，香农定理以每秒比特（bps）的形式给出一条链路容量的上限，表示为链路信噪比的函数，

⊖ 有时简称香农定理，但实际上香农提出了相当多的定理。

用分贝（dB）度量。其中涉及的信道带宽用赫兹（Hz）度量。（正如先前提到的，带宽是通信的负载量，这里指的是通信中可用的频率范围。）

例如，我们使用香农定理来确定调制解调器的信息传输速率，当调制解调器以这个速率在一个音频电话线上传输二进制数据时，不会出现过高的差错率。标准音频电话线支持的典型频率范围为300～3 300Hz，信道带宽是3kHz。

香农定理通常由下面的公式给出：

$$C=B\log_2(1+S/N)$$

其中C是可达到的信道容量，单位为bps，B是用Hz表示的信道带宽（3 300Hz－300Hz=3 000Hz），S是信号的平均功率，N是噪声的平均功率。信噪比（S/N，或SNR）通常用dB表示，计算公式如下：

$$\mathrm{SNR}=10\times\log_{10}(S/N)$$

假设一个典型的信噪比为30dB，这表示$S/N=1\,000$。因此我们有

$$C=3\,000\times\log_2(1001)$$

近似等于30kbps。

在20世纪90年代，调制解调器是接入因特网的主要方式，其标准容量是56kbps（拨号的上限）。然而在实践中，调制解调器能够达到的速率要低一些，因为并不总是有足够高的信噪比使其能够达到56kbps。

香农定理可以应用于各种链路，包括无线链路、同轴电缆和光纤等。显然构建高容量的链路只有两种方法，提高带宽或提高信噪比，或者同时提高带宽和信噪比。有人希望通过设计编码方法来使传输速率达到信道的理论上限值，但这也无法保证高容量的链路。这种想法在今天的无线链路中特别明显，依托给定的无线频谱（信道带宽）和信号功率（SNR）以求获得更高的信息传输速率。

链路分类

本书的大部分读者已经遇到了一些不同类型的链路，这有助于理解现有的不同类型的链路及其属性。所有链路都依赖于通过介质或自由空间传播的电磁波。链路分类的方法之一就是依据其所采用的介质来划分，典型的有铜线，如数字用户线（DSL）和同轴电缆，还有光纤，如商业型光纤到户服务和因特网骨干网中的长距离链路，以及用户无线网络介质空气或自由空间。

链路的另一个重要属性是频率（frequency），以赫兹（Hz）作为测量单位，反映了电磁波的摆动情况。波的一对相邻最高点或最低点之间的距离称为波长（wavelength），单位为米（m）。由于所有电磁波均以光速传播（具体速度依赖于介质），所以该速度除以波的频率就等于它的波长。我们已经看到音频电话线的例子，它在300～3 300Hz的范围内传送连续的电磁信号，一个300Hz的波通过铜线传播的波长为

$$\text{Speed Of Light In Copper}\div\text{Frequency}=2/3\times3\times10^8\div300=667\times10^3\ \text{m}$$

通常，电磁波横跨一个很宽的频率范围，从无线电波到红外线到可见光到X射线和伽马射线。图2-2描绘了电磁波的频谱，并说明了何种介质用于传送哪一频段。

迄今为止，我们了解到一条链路就是用来传送电磁波信号的介质。这种链路为传输各种信息提供了基础，包括传输中我们感兴趣的数据类型，即二进制数据（1和0）。我们称

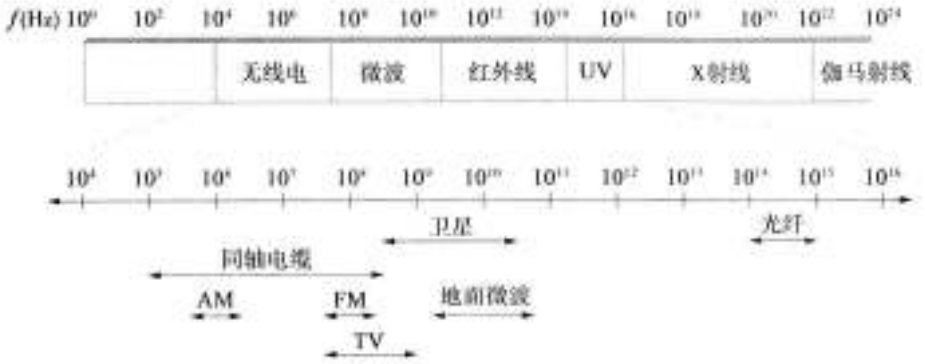


图 2-2 电磁波频谱

二进制数据被编码（encoded）到信号中。把二进制数据编码成电磁波信号是一个复杂的问题。为了使问题更加易于处理，我们可以把它分成两层来考虑。下层涉及调制（modulation），即通过改变信号的频率、振幅或相位来实现信息的传输。调制的一个简单例子是改变单一波长的振幅。在直观上这相当于开灯和关灯。由于在讨论链路作为计算机网络构件时调制问题是次要的，我们只假设有可能传输一对可区别的信号——把它们想象成“高”信号和“低”信号。所以我们只考虑上层，即只关心将二进制数据编码成这两种信号的问题。2.2 节将讨论这样的编码问题。

对链路进行分类的另一种方法是依据其使用方式，不同的经济考虑和部署方式都会影响链路类型，大部分消费者可能会通过无线网络（在咖啡店、机场、大学等区域）与因特网进行交互，也可能会通过因特网服务提供商提供的所谓“最后一英里”的链路，正如图 2-1 中所描述的一样。表 2-1 总结了这些链路类型。选取这些类型作为代表是因为它们对于数百万用户而言都是性价比较高的。例如 DSL 依托于现有的双绞线，而这些双绞线在普及的传统电话服务中已经搭建完成。对于建造一个完整的网络来说，这些技术大多无法胜任，例如在大型网络中通过长距离高速链路来连接若干个城市。

表 2-1 家庭接入的基本服务方式

服 务	带宽（典型值）
拨号	28～56kbps
ISDN	64～128kbps
DSL	128kbps～100Mbps
CATV（电缆电视）	1～40Mbps
FTTH（光纤到户）	50Mbps～1Gbps

过去 20 年中，在长距离链路中使用的同轴电缆目前几乎无一例外地被光纤所取代，这些链路使用了一种称为 SONET（同步光学网络）的技术，该技术可以满足电信传输的需求，2.3.3 节将会更详细地介绍 SONET。

最后，除了最后一英里和骨干网链路外，在一个建筑物或一所校园内通常采用的接入方式是局域网（LAN）。正如 2.6 节中所介绍的，以太网在这一领域占有统治地位，已经取代了先前的令牌环技术。以太网经久不衰，目前与基于 802.11 标准的无线技术并驾齐驱，2.7 节将会进一步介绍无线技术。

这里关于链路类型的概述虽然并不全面，却也揭示了现有链路类型的基本方法及链路类型的多样性。在下节中，我们将会看到网络协议利用这种多样性展现高层的一致性，而无需考虑低层的复杂性细节。

2.2 编码（NRZ、NRZI、曼彻斯特、4B/5B）

将节点和链路变成可用构件的第一步，是清楚它们如何连接，以使比特从一个节点传输到另一个节点。正如在前一节中提到的，信号是在物理链路上传播的。因此，我们的任务是将源节点准备发送的二进制数据编码为链路能够传送的信号，然后在接收节点将信号解码成相应的二进制数据。我们忽略调制的细节并假设处理两种离散信号，即高信号电平和低电平。实际上，这些信号可能对应着铜线链路上的两个不同电压或光纤链路上的两个不同能量级。

正如之前已提到的，本章讨论的大部分功能是由网络适配器（network adaptor）完成的，它是一个将节点连接到链路上的硬件。网络适配器包括一个信令构件，它在发送节点把比特编码为信号，而在接收节点将信号解码为比特。因此，如图 2-3 所示，信号在两个信令构件之间的链路上传输，而比特在两个网络适配器之间流动。



图 2-3 信号在信令构件之间传输，比特在适配器之间流动

回到将比特编码为信号的问题。显然，要做的就是将数值 1 映射为高电平，数值 0 映射为低电平。这是一种称为不归零（Non-Return to Zero，NRZ）的编码方案所采用的映射。例如，图 2-4 以图解方式描述了一个特定的比特序列（图的上部）及其对应的 NRZ 编码信号（图的下部）。



图 2-4 一个比特流的 NRZ 编码

NRZ 的问题是，几个连续的 1 表示在一段时间内信号在链路上保持为高电平，类似地，几个连续 0 表示信号在一段时间内保持为低电平。一长串 0 和 1 导致两个基本问题。第一个问题是，它会导致基线漂移（baseline wander）状态。尤其是接收方保持一个它所看到的信号平均值，然后用这个平均值区分高、低电平。当收到的信号远低于这个平均值时，接收方就断定看到了 0，同样，远高于这个平均值的信号被认为是 1。当然，问题是太多连续的 1 或 0 会使这个平均值发生改变，使得检测信号中很难出现明显的变化。

第二个问题是，由高到低和由低到高的频繁转换必须使用时钟恢复（clock recovery）。直观地讲，时钟恢复问题就是：编码和解码过程都由一个时钟来驱动，每个时钟周期发送方发送 1 比特，接收方恢复 1 比特。为了使接收方能恢复发送方发送的比特，发送方和接收方的时钟必须精确同步。如果接收方时钟比发送方时钟稍快或稍慢，那么，接收方就不能正确地解码信号。可以采用在另一条线上发送时钟给接收方的方法，但这种方案不太可行，因为这使布线费用增加一倍，所以接收方改由收到的信号得到时钟。这就是时钟恢复

过程。无论何时，只要信号有从1到0或从0到1的跳变，接收方就知道这是在时钟周期的边界上，它能够自己进行重新同步。然而，若长时间没有这样的跳变就会导致时钟漂移。所以，无论传送什么数据，时钟恢复都依赖于信号内有许多跳变。

有一种方法可以解决这个问题，称为*不归零反转*（Non-Return to Zero Inverted，NRZI），发送方将当前信号的跳变编码为1，将当前信号的保持编码为0。这样就解决了连续1的问题，但是显然未解决连续0的问题。NRZI如图2-5所示。还有一种方法称为*曼彻斯特编码*（Manchester encoding），这种颇具独创性的方法通过传输NRZ编码数据与时钟的异或值使时钟与信号结合在一起。（把本地时钟看作一个从低到高变化的内部信号，一对低/高变化的电平看作一个时钟周期。）图2-5也给出了曼彻斯特编码。注意，曼彻斯特编码将0作为由低到高的跳变，1作为由高到低的跳变。因为0和1都导致信号的跳变，所以接收方能有效地恢复时钟。（还有一种曼彻斯特编码的变种，称为差分曼彻斯特（differential Manchester）编码。其方法是若信号的前一半与前一比特信号的后一半信号相等则编码为1，若信号的前一半与前一比特信号的后一半信号相反则编码为0。）

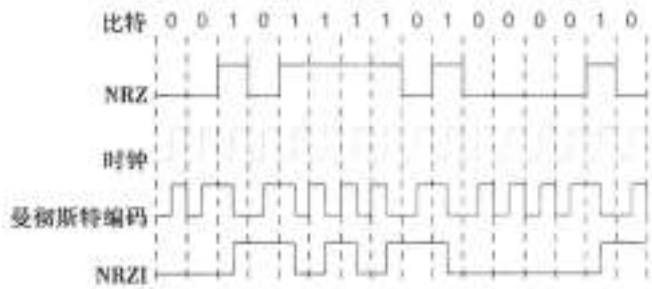


图2-5 不同的编码策略

曼彻斯特编码方案存在的问题是使链路上信号跳变的速率加倍，这意味着接收方有一半的时间在检测信号的每一个脉冲。信号变化的速率称为链路的*波特率*（baud rate）。在曼彻斯特编码中，比特率是波特率的一半，所以认为编码的效率仅为50%。记住，如果接收方保持比图2-5中的曼彻斯特编码要求的更快的波特率，那么在相同的时间段中，NRZ和NRZI能传输2倍的比特数。

我们考虑的最后一种编码方法称为4B/5B，它力求不扩大高信号或低信号的持续期而解决曼彻斯特编码的低效问题。4B/5B的思想是在比特流中插入额外的比特以打破一连串的0或1。准确地讲，就是用5个比特来编码4个比特的数据，之后再传给接收方，因此称为4B/5B。5比特代码是由以下方式选定的：每个代码最多有1个前导0，并且末端最多有两个0。因此，当连续传送时，在传输过程中任何一对5比特代码连续的0最多有3个。然后，再将得到的5比特代码使用NRZI编码传输，这种方式说明了为什么仅需关心多个连续0的处理，因为NRZI已解决了多个连续1的问题。注意，4B/5B编码的效率为80%。

表2-2给出了16个可能的4比特数据符号对应的5比特代码。注意，由于5比特足以编码32个不同的代码，因此我们仅用了16个，剩下的16个可用于其他目的。其中，11111

表2-2 4B/5B编码

4比特数据符号	5比特编码
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

可用于表示线路空闲，00000 表示线路不通，00100 表示停止。在剩下的 13 个码中，7 个是无效的（因为它们违反了 1 个前导、0 或两个末尾 0 的规则），另外 6 个代表各种控制符号。在本章后面将会看到，某些组帧协议会使用这些控制符号。

2.3 组帧

我们已经看到在一个点到点链路（从一个适配器到另一个适配器）上如何传输比特序列，现在考虑图 2-6 中的情形。回顾一下第 1 章，我们关注的是分组交换网，即在节点间交换的是数据块（在这一层称为帧（frame））而不是比特流。网络适配器使节点间能够交换帧。当节点 A 希望向节点 B 传送一帧时，它告诉自己的适配器从节点的内存中传送一帧，这导致一个比特序列传到链路上。然后，节点 B 的适配器收集链路上到达的比特序列，并在 B 的内存中存放相应的帧。准确识别什么样的比特集合构成一帧，即决定帧从哪里开始到哪里结束，是适配器面临的主要挑战。



图 2-6 适配器之间的比特流、主机之间的帧

解决组帧问题有几种方法。本节从设计的角度使用不同的协议来说明各种方法。注意，尽管我们是在点到点链路的环境中讨论组帧问题，但在多路访问网（如以太网和令牌环）中组帧也是一个基本问题。

2.3.1 面向字节的协议（BISYNC、PPP、DDCMP）

最早的组帧方法是把每一帧看成一个字节（字符）集，而不是一个比特集，这种方法源于终端与大型机的连接。面向字节（byte-oriented）方法的实例有：20 世纪 60 年代末 IBM 开发的二进制同步通信（Binary Synchronous Communication，BISYNC）协议，以及用于数字设备公司 DECNET 网络上的数字数据通信消息协议（Digital Data Communication Message Protocol，DDCMP）。近来广泛使用的点对点协议（PPP）则是这种方法的另一个实例。

1. 起止标记法

BISYNC 协议用起止标记法组帧，图 2-7 表示这个协议的帧格式。本书此后将用这样的图来表示帧或分组格式，所以此处稍加解释。我们将一个分组表示为一个带标记的字段序列，每个字段上方的数字表示字段所占的比特长度。注意，分组是从最左端的字段开始传输的。

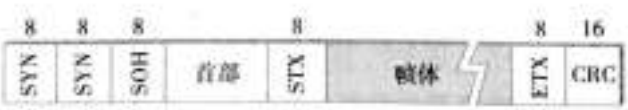


图 2-7 BISYNC 帧格式

BISYNC 使用称为起止字符（sentinel character）的特定字符表示帧的开始与结束。一帧的开始由发送一个特定的 SYN（同步）字符表示。其后帧的数据部分包含在两个特殊的起止字符之间：STX（正文开始符）和 ETX（正文结束符）。SOH（首部开始符）字段与 STX 字段的目的是一样的。自然，起止标记法存在的问题是 ETX 字符可能会出现在帧的数据部分。BISYNC 通过对 ETX 字符“转义”的方法解决这个问题，无论 ETX 出现在帧体中什么位置，都在其前加上一个 DLE（数据链路转义）字符，帧体中的 DLE 字符也采用同样的方法（在其前多加一个 DLE）处理。（C 程序员可能注意到，这类似于当引号出现在一个字符串中时用反斜线转义的处理方法。）因为要在帧的数据部分插入额外的字符，所以这种方法常称为字符填充法（character stuffing）。

帧格式还包括一个用于检验传输差错的字段，标记为 CRC（循环冗余校验）。2.4 节会给出各种差错检测算法。最后，帧中包括附加的 Header（首部）字段，用于链路层可靠传输算法，2.5 节会给出这些算法的例子。

最近 PPP 通常用于在各种点对点链路上传输 IP 分组，并且同 BISYNC 一样使用起止字符填充法。图 2-8 给出 PPP 的帧格式。特殊的正文起始字符 STX 在图 2-8 中表示为 Flag（标志）字段，其值为 01111110。Address（地址）和 Control（控制）字段通常取默认值，所以不用理会。Protocol（协议）字段用于多路分解：它标识高层协议，如 IP 或 IPX（一个由 Novell 公司开发的类似 IP 的协议）。帧的 Payload（有效载荷）长度是可以协商的，但它的默认值是 1 500 字节。Checksum（校验和）字段的长度是 2 字节（默认）或 4 字节。

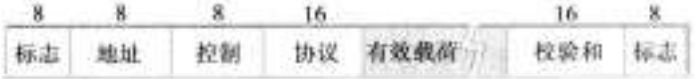


图 2-8 PPP 帧格式

PPP 帧格式中有几个字段长度是可以协商而不是固定的，这种协商由链路控制协议（Link Control Protocol，LCP）管理。PPP 和 LCP 协同工作：LCP 发送封装在 PPP 帧中的控制消息，该消息由 PPP 的 Protocol 字段中的一个 LCP 标识符表示，然后返回并根据包含在控制消息中的信息来改变 PPP 的帧格式。当两端的对等实体都检测到载波信号时（例如，当光接收器检测到所连接的光纤上有发来的信号时），在它们之间建立一条链路的过程也会用到 LCP。

2. 字节计数法

正如每一位计算机科学专业的学生所知道的，可以代替起止标记值检测文件结束的另一种方法是在文件开始处包含文件中的项目个数。组帧也是如此，包含在帧中的字节数可放在帧首部的一个字段中。DECNET 网络的 DDCMP 协议就是采用这种方法，如图 2-9 所示。在这个例子中，Count（计数）字段指明在帧体中包含多少字节。

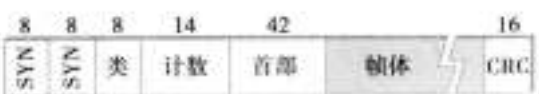


图 2-9 DDCMP 帧格式

这种方法的危险是，传输差错可能破坏计数字段，在这种情况下，将不能正确检测到帧的结束。（如果 ETX 字段被破坏，类似的问题也存在于起止标记法中。）假如发生此类

差错，接收方就会累计错误的 Count 字段所指示的字节数，然后使用差错检测字段确定帧出错了。我们有时将这种情况称为组帧差错（framing error）。然后接收方将等待，直到看到下一个 SYN 字符，便会开始收集组成下一帧的字节。因此，组帧差错可能引起多个后续帧的错误接收。

2.3.2 面向比特的协议（HDLC）

与面向字节的协议不同，面向比特的协议不关心字节的边界，它只把帧看成比特的集合。这些比特可能来自某个字符集，如 ASCII 码，它们可能是一幅图像中的像素值或一个可执行文件的指令和操作数。由 IBM 开发的同步数据链路控制（Synchronous Data Link Control，SDLC）协议就是一个面向比特的协议，后来由 OSI 将它标准化为高级数据链路控制（High-Level Data Link Control，HDLC）协议。在下面的讨论中，我们将 HDLC 作为一个例子，它的帧格式如图 2-10 所示。



图 2-10 HDLC 帧格式

HDLC 用特定的比特序列 01111110 表示帧的开始与结束。在链路空闲时也发送这个序列，以保证发送方和接收方的时钟同步。这样，双方的协议本质上都使用起止标记法。因为这个序列可能出现在帧体中的任何位置（事实上，比特序列 01111110 可以跨字节边界），所以面向比特的协议使用类似于 DLE 字符的方法，这种方法称为比特填充法（bit stuffing）。

在 HDLC 协议中，比特填充过程如下。在发送方，任意时刻从消息体中发出 5 个连续的 1 后（发送方试图发送特别序列 01111110 除外），发送方在发送下一比特之前插入一个 0。在接收方，如果 5 个连续的 1 到达了，接收方根据它看到的下一比特（即 5 个 1 后面的比特）做出决定。如果下一比特为 0，则一定是填充的，接收方就把它去掉；如果下一比特是 1，则有两种情况，即帧结束标记或是比特流中出现差错。通过查看下一（next）比特，接收方可以区别这两种情形：如果看到 0（即最后 8 比特为 01111110），那么它一定是帧结束标记；如果看到 1（即最后的 8 比特为 01111111），则一定是出错了，需要丢弃整个帧。在后一种情形中，接收方必须等到下一个 01111110 出现才能再一次开始接收数据，结果，接收方有可能连续两次接收帧失败。显然，仍存在组帧差错未被检测出来的情形，例如，可能由于差错而产生假的帧结束模式，但这种差错相对而言不大可能。我们将在 2.4 节讨论健壮的检错方式。

比特填充法和字符填充法都有一个有趣的特性：一个帧的长度由帧的有效载荷中传送的数据决定。事实上，如果在任何帧中携带的数据是任意的，所有的帧就不可能同样大。（为了使人信服，考虑如果一个帧体的最后字节是 ETX 字符将会出现什么情况。）确保所有帧为同样大小的组帧形式在下一小节讨论。

相关主题

层中包含什么？

第 1 章提到的 OSI 参考模型的一个重要贡献在于，它提供了对协议（特别是协议层）进行讨论的一些术语。这些术语引起了很多争议，例如，“你的协议把功能 X 放在 Y 层，而 OSI 参考模型说明这个功能应放在 Z 层——这是分层违例。”事实上，为执行给定的功能而给出正确的层是非常困难的，而且其理由通常比“OSI 模型怎么说”还难以说清。这

就是本书避免严格分层方法的部分理由。本书的做法是展示许多需要由协议执行的功能，并考察已经成功实现它们的一些方法。

尽管我们不采用分层方法，但有时我们需要方便的方式来讨论协议类，用协议在其上进行操作的层的名称是最好的选择。例如，本章主要关心链路层协议。(2.2节描述的比特编码是一个例外，它被认为是物理层功能。)链路层协议可通过它们是否运行在单一链路上来识别（本章讨论的网络类型就是单一链路）。对比之下，网络层协议运行在交换网上，其中包含许多由交换机或路由器互联起来的链路。第3章和第4章将讨论网络层协议的相关主题。

注意，当谈及协议类和把建网的问题划分为可管理的子任务时，协议层是有用的，因为它们提供了有助于讨论的方法。然而，这并不意味着过分的限制：事实是分层通例不会使是否值得分层的争论停止。换句话说，分层可以做一个好仆人，而不是一个可怜的主人。在第6章讨论拥塞控制时，又会出现某一功能最好放在哪一层的有趣争论。

2.3.3 基于时钟的组帧（SONET）

组帧的第3种方法以同步光纤网络（Synchronous Optical Network，SONET）标准为例。由于没有广为接受的通用术语，我们简单地将这种方法称为基于时钟的组帧(clock-based framing)。SONET最初是由贝尔通信研究室提出的，然后由美国国家标准协会（ANSI）开发用于在光纤上传输数字数据，此后被ITU-T采纳。但是谁进行标准化、标准化什么以及何时标准化并不是我们关心的问题。关于SONET应该记住的是，它是在光网络上远距离传输数据的主要标准。

在深入研究SONET之前需要了解的一件重要的事情是，SONET的全部规范比本书还厚。因此，下面的讨论只能涉及这一标准的关键点。SONET同样也解决组帧问题和编码问题。它还解决了一个对电话公司来说非常重要的问题：几条低速链路多路复用到一条高速链路。我们首先讨论组帧，然后讨论其他问题。

正如以前讨论的组帧方案，一个SONET帧包含一些特殊的信息，告诉接收方哪里是帧的开始、哪里是帧的结束。然而，这是仅有的相似之处。特别地，由于并不使用比特填充，所以帧的长度不依赖于传送的数据。SONET帧的问题是接收方如何知道每一帧从哪里开始和到哪里结束。我们针对速率为51.84Mbps的STS-1低速SONET链路来考虑这个问题。一个STS-1帧如图2-11所示，它有9行，每行90个字节，并且每行的前3个字节是系统管理信息，其余字节可用于链路上传送的数据。帧的前两个字节包含一个特定的比特模式，使得接收方能够确定帧从哪里开始。然而，由于不使用比特填充，所以这个模式可能会偶尔出现在帧的有效载荷部分。为了避免出现这种情况，接收方不断地寻找特定比特模式，希望它每810个字节出现一次，因为每帧的长度是$9\times90=810$字节。当特定的模式在应有的位置出现多次时，接收方可推断它处于同步状态，然后就能正确地解释帧。



图2-11 SONET STS-1帧

由于SONET的复杂性，我们没有描述所有其他系统管理信息字节的详细用法。它的复杂性可部分归结为，SO-

NET 是在电信公司的光网上运行的，而不仅仅是在一条单链路上运行。（回忆一下，我们掩盖了电信公司实现网络的事实，而仅考虑租用一条 SONET 链路，并使用这条链路建立自己的分组交换网。）额外的复杂性还来自于 SONET 可提供更丰富的服务，而不仅仅是数据传输。例如，为用于维护的语音信道预留一个容量为 64kbps 的 SONET 链路。

SONET 帧的系统管理信息字节可用 NRZ 编码，这是前一节描述的一种简单编码，其中 1 表示高 0 表示低。然而，为确保有足够的跳变使接收方获得发送方的时钟，有效载荷字节需混杂编码（scramble），通过计算被传输数据与一个已知的比特模式的或（异或）来完成。比特模式的长度是 127 比特，其中有许多从 1 到 0 的跳变，因此，与被传输数据进行异或运算后可得到具有足够多跳变的信号，使接收方能够恢复时钟。

SONET 以下面的方式支持多个低速链路的多路复用。一个给定的 SONET 链路以一组有限的可能速率运行，从 51.84Mbps（STS-1）到 2 488.32Mbps（STS-48）或更高。注意所有速率都是 STS-1 的整数倍。组帧的意义在于，单个 SONET 帧能包含多个较低速率信道的子帧。第二个相关的特征是每一帧长 125μs。这就是说，在 STS-1 速率下，一个 SONET 帧是 810 字节长，而在 STS-3 速率下，每个 SONET 帧是 2 430 字节长。注意这两个特征间的关系是：3×810=2 430，即三个 STS-1 帧正好放在一个 STS-3 帧中。

直观地讲，STS-N 帧可认为是由 N 个 STS-1 帧构成的，这些帧中的字节是交叉的，即传输第一帧的一个字节，然后传输第二帧的一个字节，等等。交叉每个 STS-N 帧中的字节的原因是保证每个 STS-1 帧的字节传送进度均匀，即字节在接收方以 51Mbps 的平滑速率出现，而不是在 125μs 间隔的某个 1/N 的时间内全部成串出现。

尽管 STS-N 信号可看成是多路复用的 N 个 STS-1 帧，但这些 STS-1 帧的有效载荷还可以串接在一起形成一个更大的 STS-N 的有效载荷，这样的链路表示为 STS-Nc（c 表示串接的（concatenated））。系统管理信息中的一个字段就用于这个目的。图 2-12 描述了三个 STS-1 帧串接成一个 STS-3c 帧的情形。将 SONET 链路设计成 STS-3c 而不是 STS-3 的意义在于，在前一种情形中，用户可将链路看成一个单一的 155.25Mbps 管道，而 STS-3 实际应看作共享一条光纤的三条 51.84Mbps 链路。

最后，前面描述的 SONET 过于简单，因为它假设每一帧的有效载荷完全包括在帧内（为什么有例外的情况？）。事实上，应该简单地把刚刚描述的 STS-1 帧看作帧的容器，而真正的有效载荷可能跨越（float）帧边界。这种情形如图 2-13 所示。在此可以看到，STS-1 有效载荷跨越两个 STS-1 帧，而且有效载荷向右移了几个字节绕了回来。帧的系统管理信息中的一个字段指向有效载荷的起点。这样做的价值是简化电信网中的时钟同步工作，而时钟同步是电信网要花很多时间来处理的事情。

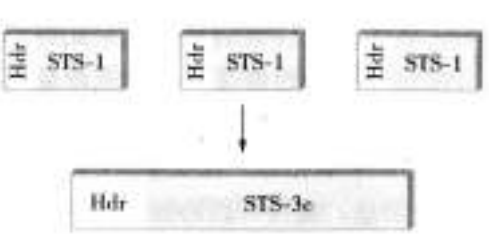


图 2-12 三个 STS-1 帧串接成一个 STS-3c 帧

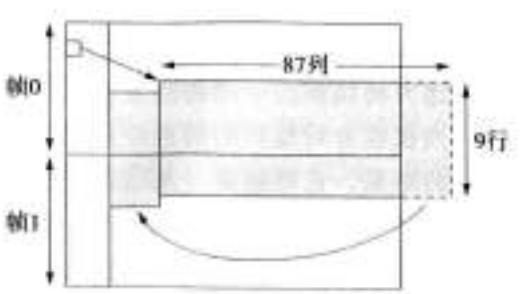


图 2-13 跨越帧边界的 SONET 帧

2.4 差错检测

如第1章中的讨论，帧中有时会发生比特错。例如，由于电干扰或热噪音，就会发生这样的差错。尽管差错很少，特别是在光链路上，但还是需要某种机制来检测这些差错，以便采取纠错措施。否则，终端用户会奇怪为什么刚刚成功编译的C程序现在突然会有一个语法错。发生这样的差错是因为这个程序是通过一个网络文件系统拷贝过来的。

用来处理计算机系统中比特错的技术已有很长的历史，至少可以追溯到20世纪40年代。早在使用打孔机以及当数据保存在磁盘或早期的磁心存储器时就已经开发出了汉明（Hamming）码和里德－所罗门（Reed-Solomon）码。本节介绍网络中最常用的一些差错检测技术。

检错只是问题的一部分，另一部分是一发现差错就立即纠错。当消息的接收方检测到差错时，可以采取两种基本方法。一种是通知发送方消息受到破坏，使发送方重发消息的副本。如果比特错很少，那么重传的副本很可能没有差错。另一种方法是采用几种差错检测算法，它们使接收方即使在消息出错后仍可以重新构造正确的消息。这些算法依赖于下面讨论的纠错码（error-correcting code）。

检测传输差错的一项最常用的技术叫作循环冗余校验（Cyclic Redundancy Check，CRC）。它几乎用在前几节讨论的所有链路层协议中，例如HDLC、DDCMP以及本章后面讲到的CSMA和无线协议。2.4.3节概述基本的CRC算法。在讨论该算法之前，我们考虑两种也被广泛使用的较为简单的差错检测方法：二维奇偶校验（two-dimensional parity）与校验和（checksum）。前者在BISYNC协议传输ASCII码字符时采用（当BISYNC用于传输EBCDIC[⊖]时用CRC），而后者在若干因特网协议中使用。

任何差错检测方案的基本思想都是在帧中加入冗余信息来确定是否存在差错。极端情况下，可以想象传输数据的两个完整副本。如果这两个副本在接收方是相同的，那么可能它们都是正确的；如果不同，那么其中之一或者两者都有错误，必须将它们丢弃。这是相当差的差错检测方案，原因有两点：第一，它为n比特消息发送n比特冗余信息；第二，有许多差错检测不到，如恰好在消息的第一和第二个副本的相同比特位置出错时，便检测不到。

幸运的是，我们有比这个简单方案更好的方法。一般说来，当为n比特消息仅发送k个冗余比特时，我们能够提供相当强的差错检测能力，其中$k \ll n$。例如，在以太网上，一个12 000比特（1 500字节）的数据帧仅需要一个32位的CRC码，通常表示为CRC-32。下面将会看到，CRC码能发现大多数的差错。

之所以说发送的额外比特是冗余信息，是因为它们不是向消息中加入新的信息，而是用某种明确定义的算法直接从原始消息中导出信息。发送方和接收方都确切知道这个算法，发送方将该算法应用到消息上以产生冗余比特，然后，它将该消息和冗余比特都传输出去。当接收方对收到的消息应用同一算法时，（在没有差错的情况下）应该产生与发送方相同的结果。它将结果与发送方发给它的结果进行比较，如果它们相等，就可能（以很高的或然率）做出结论，消息在传输过程中没有出错；如果不相等，就能够确定消息或冗余比特受到破坏，对此必须采取适当的措施，那就是丢弃消息，或在可能的情况下纠错。

⊖ 用于20世纪60年代的另一种字符编码模式。

注意这些额外比特的术语。一般说来，它们指的是差错检测码（error detecting code）。在特定的情况下，当产生编码的算法是以相加为基础时，可能称为校验和（check sum）。我们将会看到，因特网校验和的命名是很恰当的：它是使用求和算法的一种差错检测机制。不幸的是，校验和这个词常被不准确地用于表示任何形式的差错检测码，包括CRC。这可能引起混乱，因此，我们主张将校验和这个词仅用于真正使用求和运算的代码，而用差错检测码这个词表示本节描述的一般类型的代码。

2.4.1 二维奇偶校验

二维奇偶校验就是这个名字所指的含义。它基于“简单的”（一维）奇偶校验——通常把额外的1比特附加到7比特编码上，来平衡字节中1的个数。例如，奇校验根据字节中1的个数置位第8比特位，使8个比特中1的个数为奇数；而偶校验则置位第8比特位使8个比特中1的个数为偶数。二维奇偶校验对帧中每一字节的每一比特位置进行类似的计算。结果除了每一字节增加1个奇偶校验位外，整个帧产生了一个额外的奇偶校验字节。图2-14说明二维偶校验对一个含有6字节数据的帧的操作过程。需要注意的是，奇偶校验字节中的第3个比特是1，因为帧中6个字节的第3位中含有奇数个1。可以证明，二维奇偶校验可检测所有1、2、3比特错及大部分4比特错。在这种情况下，我们给42比特的消息加入了14比特的冗余信息，从而对于一般差错具有比上面描述的“重复码”更强的保护能力。

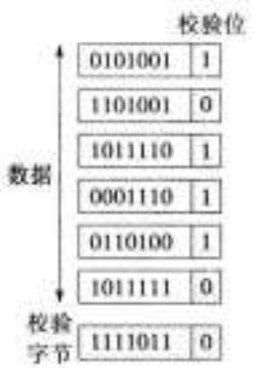


图2-14 二维奇偶校验

2.4.2 因特网校验和算法

差错检测的第二种方法以因特网校验和为例。虽然此方法不在链路层使用，但它仍提供了与CRC和奇偶校验同样的功能，因此在这里进行讨论。在4.1节、5.1节和5.2节将看到它的实例。

因特网校验和的思想非常简单——将传输的所有字加起来，然后传输这个相加的结果，此结果称为校验和。接收方对收到的数据执行同样的计算，然后把得到的结果与收到的校验和进行比较。如果传输的任何数据（包括校验和本身）出错，那么结果将不相同，接收方就知道发生了错误。

可以想象校验和基本思想的许多不同变种。因特网协议所使用的具体方案如下。把计算校验和的数据看作一个16位整数序列，采用16位反码算法（下面解释）将它们加在一起，然后对结果取反码，得到的16位数即为校验和。

在反码算法中，负整数$-x$用x的反码表示，也就是说，将x的每一位取反。当在反码算法中进行加法运算时，需要将来自于最高有效位的一个进位加到结果中。例如，考虑4比特整数的反码算法中-5与-3相加。$+5$是0101，所以-5是1010；$+3$是0011，所以-3是1100。如果我们将1010和1100相加而不考虑进位，则得到0110。在反码算法中，将最高位的进位再加到结果上，得到0111，正如我们预期的，它是-8的反码表示（对1000各位取反而获得）。

下面的例程给出因特网校验和算法的一个直接实现。参数count给出以16位为单位的buf的长度。例程假设buf的末尾已经用0填充成16位。

```
u_short
cksum(u_short *buf, int count)
{
    register u_long sum = 0;

    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            /* carry occurred,
               so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

这段代码确保计算使用反码算法而不是大部分机器上使用的补码算法。注意 while 循环之内的 if 语句，如果在 sum 的 16 位最高位有一个进位，那么就像在前面的例子中一样给 sum 加 1。

这个算法比重复编码要好，因为它使用很少的冗余比特，即对任意长度的消息仅用 16 位，但是它的差错检测能力却不太好。例如，有一对单比特错，一个比特错使一个字增加 1，而另一个比特错使另一个字减少 1，若两种差错数量相同便无法检测。虽然在检错能力上相对较弱（例如与 CRC 相比），但我们仍然使用这种算法，原因很简单：这个算法易于用软件实现。ARPANET 网的经验表明，这种形式的校验和就足够了。理由是校验和是端到端协议的最后一道防线，大部分差错将由链路层上更强的差错检测算法（如 CRC）检查出来。

相关主题

简单概率计算

当处理网络差错和其他（我们希望）不大可能发生的事件时，通常使用简单概率估计。这里有一个有用的近似值，如果两个独立事件具有小概率 p 和 q，那么两个事件之一发生的概率是 $p+q$，准确答案是 $1-(1-p)(1-q)=p+q-pq$。对于 $p=q=0.01$，估计值是 0.02，而准确值是 0.0199。

考虑这个结果的一个简单应用，假设一条链路上的每比特差错率是 $1/10^7$。假设需要估计在一个 10 000 比特分组中至少 1 比特出现差错的概率。利用上述方法对所有比特反复近似，我们感兴趣的是第 1 比特错的概率、第 2 比特错的概率或第 3 比特错的概率，依次类推。若比特错完全是独立的（其实不是），我们可以估计，一个 10 000（10^4）比特的分组中至少有一个差错的概率是 $10^4\times10^{-7}=10^{-3}$。由 $1-P$（无错）计算出的准确答案是 $1-(1-10^{-7})^{10\,000}=0.000\,999\,50$。

对于复杂一些的应用，我们计算一个分组中出现两个差错的概率，即逃脱 1 奇偶校验位校验和差错的概率。考虑在一个分组中的两个特定的比特，分别为比特 i 和比特 j，恰是这两个比特错的概率是 $10^{-7}\times10^{-7}$。在这个分组中，可能的比特对的总数是

$$\binom{10^4}{2}=10^4\times(10^4-1)/2\approx5\times10^7$$

所以，再次利用多偶然事件（其中包括任意可能出错的比特对）概率的反复相加近似求得至少两个比特错的概率是 $5\times10^{7}\times10^{-11}=5\times10^{-7}$。

2.4.3 循环冗余校验

至此，我们应该清楚设计差错检测算法的主要目标是用最少的冗余比特检测最多的错误。循环冗余校验使用一些功能很强的数学算法来达到这个目标。例如，一个 32 比特的 CRC 码对于上千字节长的消息中一般的比特错具有很强的检测能力。循环冗余校验的理论基础源于一个称为有限域（finite field）的数学分支。尽管听起来深奥，但它的基本思想很容易理解。

首先，考虑一个由 n 次多项式（即最高幂次项是 x^n 的多项式）表示的 $n+1$ 比特消息。用多项式表示消息时，消息的每一比特值为多项式中每一项的系数，并从最高有效位代表最高幂次项开始。例如，一个 8 比特消息 10011010 对应多项式

$$M(x)=1\times x^7+0\times x^6+0\times x^5+1\times x^4+1\times x^3+0\times x^2+1\times x^1+0\times x^0=x^7+x^4+x^3+x^1$$

这样，我们可以认为发送方和接收方在互相交换多项式。

为了计算 CRC，发送方和接收方必须商定一个除数（divisor）多项式 $C(x)$。$C(x)$ 是一个 k 次幂的多项式。例如，假设 $C(x)=x^3+x^2+1$，在这种情况下，$k=3$。在多数实际情况中，"$C(x)$ 从何而来"这个问题的答案是"到书中去找"。事实上，正如下面要讨论的，$C(x)$ 的选择对于能被可靠检测的差错类型有很大影响。有少数除数多项式对于各种情况都是很好的选择，并且准确的选择通常是协议设计的一部分。例如，以太网标准使用一个众所周知的 32 次幂的多项式。

当发送方想要传输一个 $n+1$ 比特长的消息 $M(x)$ 时，实际发送的是该 $n+1$ 比特的消息加上 k 比特。我们将这个包括冗余比特的完整消息称为 $P(x)$。我们要做的就是试图使表示 $P(x)$ 的多项式恰能被 $C(x)$ 整除，下面解释如何去做。如果在一条链路上传输 $P(x)$，并且在传输过程中没有发生差错，那么接收方应该能用 $C(x)$ 整除 $P(x)$，余数为 0。另一方面，如果在传输过程中 $P(x)$ 中出现了某个差错，那么收到的多项式总体上可能无法被 $C(x)$ 整除，因此接收方得到一个非零余数，说明发生了差错。

了解一些多项式运算的知识将有助于理解下面的内容，它与普通的整数运算略有不同。这里，我们处理一类特殊的多项式运算，其中系数仅为 1 或 0，并且用模 2 算术执行对系数的运算。称为"模 2 多项式算术"。由于这是一本关于网络的书而不是数学课本，所以让我们将重点集中在这类运算的主要特征上（需要接受的定理）。

- 对于任意多项式，如果 $B(x)$ 的幂次比 $C(x)$ 高，则 $B(x)$ 都能被 $C(x)$ 除。
- 如果 $B(x)$ 的幂次与 $C(x)$ 相同，则任何多项式 $B(x)$ 都能被除数多项式 $C(x)$ 除 1 次。
- $B(x)$ 除以 $C(x)$ 所得余数是由 $B(x)$ 减去 $C(x)$ 得到的。为了得到 $B(x)$ 减去 $C(x)$ 的结果，我们仅仅在每一对匹配的系数上执行或（异或）操作。

例如，多项式 x^3+1 能被 x^3+x^2+1 除（因为它们的幂次都是 3），余数是 $0\times x^3+1\times x^2+0\times x^1+0\times x^0=x^2$（由每一项系数的异或得到）。以消息的形式，我们可以说 1001 能被 1101 除，余数为 0100。可以看出，余数恰好是两个消息的比特的按位异或。

现在我们知道了多项式除法的基本规则，我们能够做长除法，这对于处理较长的消息

是必要的。下面看一个例子。

回忆一下，我们希望创建一个从原始消息 $M(x)$ 得到的用于传输的多项式，它比 $M(x)$ 长 k 个比特，并且能被 $C(x)$ 整除。可以按下面的步骤执行：

1) 用 x^k 乘 $M(x)$，就是在消息的末尾加上 k 个 0，这称为零扩展的消息 $T(x)$。

2) 用 $C(x)$ 除 $T(x)$，并求出余数。

3) 从 $T(x)$ 减去余数。

显然，这时所剩的是能被 $C(x)$ 整除的一个消息。注意，作为结果的消息由 $M(x)$ 后接从第 2 步得到的余数组成，因为在减去余数（不会长于 k 比特）时，仅仅是将它与第 1 步中加入的 k 个 0 进行异或。这部分内容用一个例子说明将会更加清楚。

考虑消息 $x^7+x^4+x^3+x^1$ 或 10011010。我们从将它乘以 x^3 开始，因为除数多项式的幂次是 3。这就得到 10011010000。用 $C(x)$ 除这个多项式，这时 $C(x)$ 对应于 1101。图 2-15 显示了多项式的长除运算。给定上面讨论的多项式算术规则，长除运算很像整数除法。因而，在例子的第 1 步，我们看到除数 1101 除消息的前 4 个比特（1001），因为它们是等幂的，得到余数 100（1101 异或 1001）。下一步是从消息多项式中拿下一位数字，直到得到与 $C(x)$ 等幂的另一个多项式，在本例中是 1001。再次计算余数（100），继续下去直到计算完成。注意，我们对出现在算式上方的长除法结果并不感兴趣，重要的是计算后得到的余数。

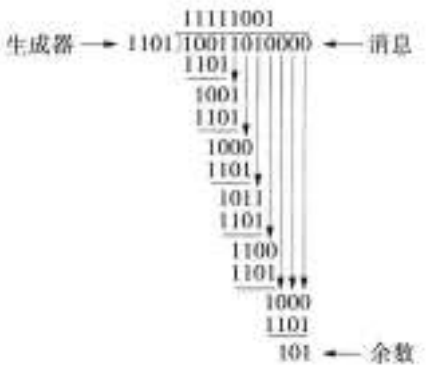


图 2-15 使用多项式长除法计算 CRC

从图 2-15 的底部可以看到，本例计算的余数是 101。由此可知，10011010000 减去 101 一定可以被 $C(x)$ 整除，并且这个差就是我们所传送的数据。多项式算术中的减法运算就是逻辑异或运算，因此实际发送的是 10011010101。如上面指出的那样，此时传送的正是原始消息再附带上余数（由长除法计算得出）。接收方用 $C(x)$ 除接收到的多项式，若结果为 0，则无错；若结果非 0，可能有必要丢弃出错的消息。使用某些编码有可能纠正小的差错（例如仅有 1 比特的差错）。能够纠正差错的编码称为纠错码（Error-Correcting Code，ECC）。

现在考虑多项式 $C(x)$ 从哪里来的问题。直观地讲，选择的多项式不大可能除尽有错的消息。如果被传输的消息是 $P(x)$，则可以将引入的差错看作是加入另一个多项式 $E(x)$，所以接收方看到的是 $P(x)+E(x)$。检测不到差错的唯一情形是收到的消息能被 $C(x)$ 整除，由于知道 $P(x)$ 能被 $C(x)$ 整除，所以只有当 $E(x)$ 也能被 $C(x)$ 整除时才会发生。选择 $C(x)$ 的技巧是使常见类型的差错多半不能被其整除。

一种常见的差错类型是单比特错，第 i 比特错时可表示为 $E(x)=x^i$。如果我们选 $C(x)$ 时它的第一项及最后一项不为 0，那么就有一个有两项的多项式，它不可能整除单项的 $E(x)$。因此，这样的 $C(x)$ 能够检测所有单比特错。一般说来，可以证明具有下述性质的 $C(x)$ 能检测出以下几类差错：

- 只要 x^k 和 x^0 项的系数不为 0，可检测所有单比特错。
- 只要 $C(x)$ 含有一个至少三项的因子，可检测所有双比特错。
- 只要 $C(x)$ 包含因子 $x+1$，可检测任意奇数个错。

- 任何组长度小于 k 比特的“成组”差错（即连续的错误比特序列，大部分大于 k 比特的成组差错也能检测到）。

链路层协议中广泛使用 6 种版本的 $C(x)$（如表 2-3 所示）。例如，以太网使用 CRC-32，而 HDLC 使用 CRC-CCITT，第 3 章描述的 ATM 使用 CRC-8、CRC-10 和 CRC-32。

表 2-3 常见的 CRC 多项式

CRC	$C(x)$
CRC-8	$x^8+x^2+x^1+1$
CRC-10	$x^{10}+x^9+x^5+x^4+x^2+1$
CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	$x^{16}+x^{12}+x^5+1$
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

最后，我们注意到，CRC 算法看起来复杂，但在硬件上使用一个 k 比特移位寄存器和若干异或门是比较容易实现的。移位寄存器中的位数等于生成多项式的幂次（k）。图 2-16 显示了用于前面例子的生成器多项式 x^3+x^2+1 的硬件。消息从左边移入，以最高有效位开始，并以附在消息后的 k 个 0 的比特串结束。正如长除法中的例子一样，当所有的比特都被移入并进行了相应的异或时，寄存器包含余数，即 CRC（右边是最高有效位）。异或门的位置按如下方式确定：如果移位寄存器中的各位从左到右标记为 0~$k-1$，那么，如果生成多项式中有 x^n 项，就在 n 位之前放置一个异或门。因而，对于生成多项式 $x^3+x^2+x^0$，在位置 0 和 2 之前有异或门。

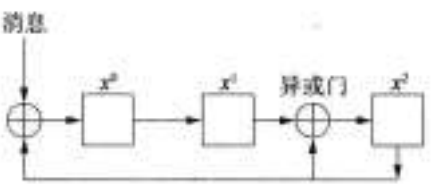


图 2-16 使用移位寄存器计算 CRC

相关主题

检错还是纠错？

我们提到过有可能使用既能检错又能纠错的编码。由于理解这种编码的细节需要的数学知识比理解 CRC 需要的数学知识更复杂，所以这里不加详述。然而，纠错相对于检错的优点却值得我们考虑。

乍看起来，纠错总是更好，因为检错时，我们被迫丢弃消息并请求传输另一份副本。这会用尽带宽并可能在等待重传时产生时延。然而，这对于纠错却更严重：它通常需要更多冗余比特来发送与检错码同样强（即能够处理同一差错范围）的纠错码。因而，虽然检错在差错发生时需要发送较多比特，但纠错却始终需要发送更多的比特。因此，当处于下列两种情况之一时，纠错才是最有用的：差错很有可能发生，例如在无线环境中就是这样；重传的代价太高，例如在卫星链路上重传一个分组时的时延。

网络中使用的纠错码有时称为前向纠错（Forward Error Correction，FEC），因为纠错是通过发送额外的信息“预先”进行的，而不是等待发生差错后再通过重传来处理。

2.5 可靠传输

正如前一节所述，当帧在传输中可能出错时，可以用像 CRC 这样的差错码来检测这

类差错。虽然有些差错码功能很强，还可以纠错，但实际上却由于开销太大以至于无法处理网络链路上发生的比特错和成组差错。即使在用纠错码时（例如在无线链路上），某些差错也可能过于严重而无法纠正。因此，某些差错帧必须丢弃。一个想要可靠传输帧的链路层协议必须能以某种方式恢复这些丢弃（丢失）的帧。

值得注意的是可靠性可以是链路层提供的功能，但是当前许多链路层技术忽略了该功能。幸运的是，在高层协议中经常提供可靠传输，包括传输层（在5.2节中介绍）和应用层（在第9章中介绍）。到底应该在哪一层提供可靠传输依赖于很多因素。本节介绍可靠传输的基本思想，该思想是跨层的，应该意识到该思想不仅只适用于链路层（详见上面的相关主题"层中包含什么?"）。

通常使用两种基本机制——确认（acknowledgement）和超时（timeout）的组合来完成上述工作。确认（简称ACK）是协议发给它的对等实体的一个小的控制帧，告知它已收到刚才的帧。所谓控制帧就是一个无任何数据的首部，但是协议也可以将ACK捎带（piggyback）在一个恰好要发向对方的数据帧上。原始帧的发送方收到确认，表明帧发送成功。如果发送方在一段相当长的时间后未收到确认，那么它重传（retransmit）原始帧。等待一段相当长的时间的动作称为超时。

使用确认和超时实现可靠传输的策略有时称为自动请求重发（Automatic Repeat Request，ARQ）。本节用通用的语言描述三种不同的ARQ算法，就是说，我们不给出一个特定协议首部字段的详细信息。

2.5.1 停止-等待

最简单的ARQ方案是停止-等待（stop-and-wait）算法，停止-等待的思想很简单：发送方传输一帧之后，在传输下一帧之前等待确认。如果在一段时间之后确认没有到达，则发送方超时，并重传原始帧。

图2-17说明此算法的四种不同情形，图中使用时间线，这是描述协议行为的一种常用方法。左侧表示发送方，右侧表示接收方，时间从上向下流动。图2-17a表示在计时器超时之前发送方收到ACK的情况，图2-17b和图2-17c分别表示原始帧和ACK丢失的情况，图2-17d表示超时过快发生的情况。回忆一下，"丢失"的意思是指帧在传输中出错时，接收方用差错码检测到这类差错，接着将帧丢弃。

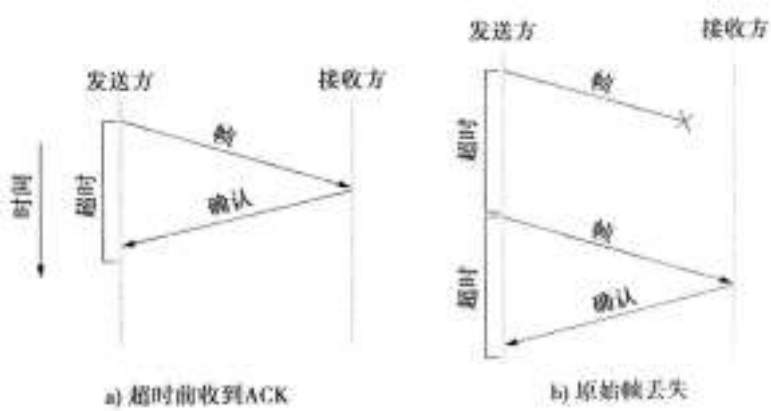


图2-17 停止-等待算法的四种不同情形的时间线

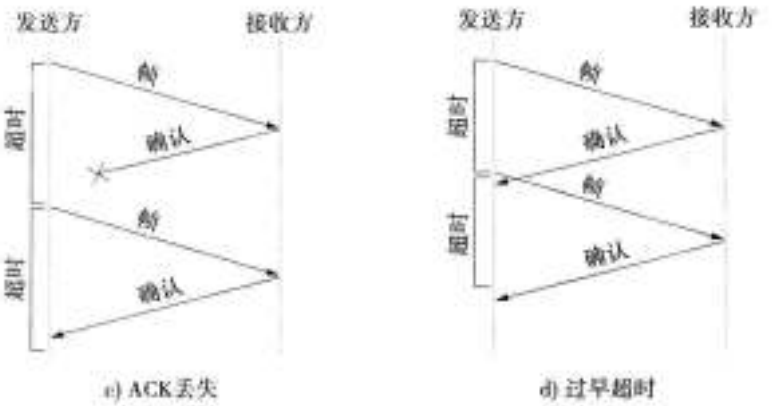


图 2-17 （续）

在停止-等待算法中有一个重要的细节。假设发送方发送一个帧，并且接收方确认它，但这个确认丢失或迟到了，如图 2-17e 和图 2-17d 所示。在这两种情况下，发送方超时并重传原始帧，但接收方认为那是下一帧，因为它正确地接收并确认了第一帧。这就引起重复传送帧的问题。为解决这个问题，停止－等待协议的首部通常包含 1 比特的序号，即序号可取 0 和 1，并且每一帧交替使用序号，如图 2-18 所示。因此，当发送方重传帧 0 时，接收方可确定它是帧 0 的第二个副本而不是帧 1 的第一个副本，因此可以忽略它（在第一个 ACK 丢失的情况下，接收方仍确认它）。

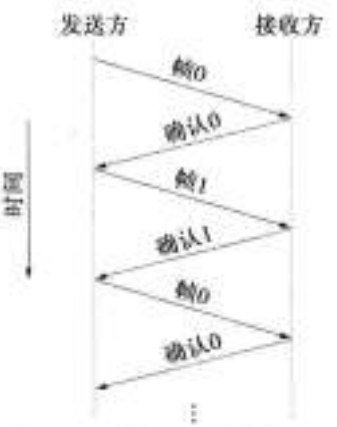


图 2-18 带有 1 比特序号的停止－等待时间线

停止－等待算法的主要缺点是，它允许发送方每次在链路上只有一个未确定的帧，这可能远远低于链路的容量。例如，考虑一条往返时间为 45ms 的 1.5Mbps 链路。这条链路的延迟与带宽的乘积为 67.5Kb（约 8KB）。由于发送方每个 RTT 仅能发送一帧，并假设一帧的大小为 1KB，这就隐含最大发送速率为

BitsPerFrame÷TimePerFrame＝1024×8÷0.045＝182kbps

或者大约是链路容量的 1/8。为充分利用链路，我们希望发送方在等待一个确认之前最多能够发送 8 帧。

相关主题

时间线和分组交换

图 2-17 和图 2-18 是在教学、解释和设计协议中经常使用的一种工具的两个实例：时间线或分组交换图。本书中有很多这样的图——参见更为复杂的实例图 9-9。由于这些图可以随着时间发展可视地捕获分布式系统的行为，有些行为可能是很难分析的，因而非常有用。在设计协议时，必须为一些意外做准备，比如系统崩溃、消息丢失或者是一些本期望很快发生的事件结果占用了很长时间。这种图表通常有助于理解在此情形下可能出现什么故障，进而帮助协议设计者为应对每一种不测做准备。

延迟带宽积的重要性在于，它表示可传输的数据总量。我们希望不等待第一个确认而能够发送这么多的数据。这里使用的原理通常称为保持管道满载（keeping the pipe full）。下面两节中的算法正是这一原理的体现。

2.5.2 滑动窗口

再次考虑链路的延迟带宽积为8KB和帧的大小为1KB的情况。我们想让发送方在第一帧的ACK到达的同一时刻准备发送第九帧。允许这样做的算法称为滑动窗口（sliding window），其时间线如图2-19所示。

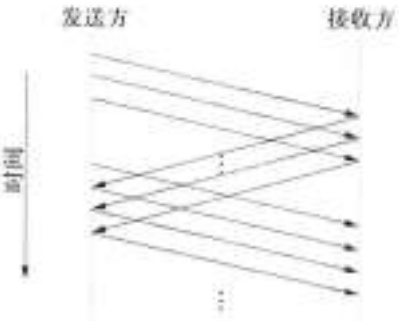


图2-19 滑动窗口算法的时间线

1. 滑动窗口算法

滑动窗口算法的工作过程如下。首先，发送方对每一帧赋予一个序号（sequence number），记作SeqNum。现在，忽略SeqNum是由有限大小的首部字段实现的事实，而假设它能无限增大。发送方维护三个变量：发送窗口大小（send window size），记作SWS，给出发送方能够发送但未确认的帧数的上界；LAR表示最近收到的确认帧（last acknowledgement received）的序号；LFS表示最近发送的帧（last frame sent）的序号。发送方还遵循如下的不等式：

$$LFS-LAR\leqslant SWS$$

这种情况如图2-20所示。

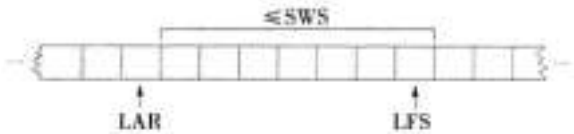


图2-20 发送方的滑动窗口

当确认到达时，发送方向右移动LAR，从而允许发送方发送另一帧。同时，发送方为所发的每个帧设置一个定时器，如果定时器在接收到ACK之前超时，则重传此帧。注意，发送方必须最多能缓存SWS个帧，因为在它们得到确认之前必须准备重传。

接收方维护下面三个变量：接收窗口大小（receive window size），记为RWS，给出接收方所能接收的无序帧数目的上界；LAF表示最大的可接收帧（largest acceptable frame）的序号；LFR表示最后收到的帧（last frame received）的序号。接收方也遵循如下不等式：

$$LAF-LFR\leqslant RWS$$

这种情况如图2-21所示。

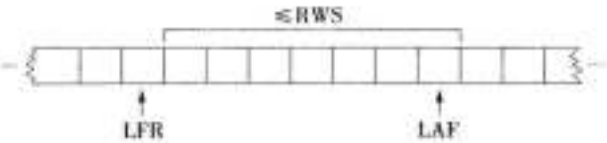


图2-21 接收方的滑动窗口

当一个具有序号SeqNum的帧到达时，接收方采取如下行动：如果SeqNum≤LFR或SeqNum>LAF，那么帧不在接收方窗口内，于是丢弃；如果LFR<SeqNum≤LAF，那么帧在接收方窗口内，于是接收。现在接收方需要决定是否发送ACK。令SeqNumToACK表示未被确认帧的最大序号，这样序号小于等于SeqNumToACK的帧都已收到。即使已经收到更高序号的分组，接收方仍确认SeqNumToACK的接收。这种确认是累积的。

然后，设 LFR=SeqNumToACK，并调整 LAF=LFR+RWS。

例如，假设 LFR=5（即上次接收方发送的 ACK 是为了确认序号 5 的），并且 RWS=4。这意味着 LAF=9。如果帧 7 和帧 8 到达，则把它们存入缓冲区，因为它们在接收方窗口内。然而并不需要发送 ACK，因为帧 6 还没有到达。帧 7 和帧 8 被称为是错序到达的。（从技术上讲，接收方可以在帧 7 和帧 8 到达时重发帧 5 的 ACK。）如果帧 6 当时到达了（或许它因第一次丢失后必须重传而晚到，或许它只是被延迟了），接收方确认帧 8，LFR 变为 8，LAF 置为 12。如果事实上帧 6 丢失了，则出现发送方超时，引发重传帧 6。

我们看到，当发生超时时，传输数据量减少，因为发送方在帧 6 确认之前不能向前移动窗口。这意味着当分组丢失时，此方案将不再保证管道满载。注意到分组已经丢失所经历的时间越长，这个问题越严重。

注意，在这个例子中，接收方可以在帧 7 一到达就为帧 6 发送一个否认应答（Negative Acknowledgment，NAK）。然而，由于发送方的超时机制足以发现这种情况，所以发送 NAK 反而为接收方增加了复杂性，因此不必这样做。同时，正如之前已提到的，当帧 7 和帧 8 到达时，为帧 5 发送一个额外的 ACK 是合理的。在某些情况下，发送方可以使用重复的 ACK 作为帧丢失的线索。这两种方法都考虑到尽早检测分组的丢失，有助于改进性能。

关于这个方案的另一个变种是使用选择确认（selective acknowledgments），就是说，接收方能够准确地确认那些已收到的帧，而不只是确认按顺序收到最高序号的帧。因此，在上例中，接收方能够确认帧 7、帧 8 的接收。如果给发送方更多的信息，就能使其较容易地保持管道满载，但增加了实现的复杂性。

发送窗口大小是根据一段给定时间内链路上有多少待传输的帧来选择的，对于给定的延迟带宽积，SWS 是容易计算的。另一方面，接收方可以将 RWS 设置为任何想要的值。通常的两种设置是：RWS=1，表示接收方不缓存任何错序到达的帧；RWS=SWS，表示接收方能够缓存发送方传输的任何帧。由于错序到达帧的数目不可能超过 SWS，所以设置 RWS > SWS 没有意义。

2. 有限序号和滑动窗口

现在讨论引入算法中做过的一个简化，即假设序号是可以无限增大的。当然，实际上是在大小有限的首部字段中说明帧的序号。例如，一个 3 比特字段意味着有 8 个可用序号（0～7）。因此序号必须可重用，或者说序号能回绕。这就带来了一个问题：要能够区别同一序号体现的不同帧，这意味着可用序号的数目必须大于所允许的待确认帧的数目。例如，停止-等待算法允许一次有一个待确认帧，并有两个不同的序号。

假设序号空间中的序号数比待确认的帧数大 1，即 SWS≤MaxSeqNum－1，其中 MaxSeqNum 是可用序号数。这就够了吗？答案取决于 RWS。如果 RWS=1，那么 MaxSeqNum≥SWS+1 就足够了。如果 RWS=SWS，那么有一个只比发送窗口尺寸大 1 的 MaxSeqNum 是不够的。为清楚这一点，考虑有 8 个序号 0～7 的情况，并且 SWS=RWS=7。假设发送方传输帧 0～6，并且接收方成功接收，但 ACK 丢失。接收方现在希望接收帧 7 和帧 0～5，但发送方超时并发送帧 0～6。不幸的是，接收方期待的是第二次的帧0～5，得到的却是第一次的帧 0～5。这正是我们想避免的情况。

事实是，当 RWS=SWS 时，发送窗口的大小不能大于可用序号数的一半，或更准确地说：

$$SWS < (MaxSeqNum+1)/2$$

直观上，这说明滑动窗口协议是在序号空间的两半之间交替，就像停止-等待协议的序号是在0和1之间交替一样。它们之间唯一的区别是，滑动窗口协议在序号空间的两半之间连续滑动而不是离散的变换。

注意，这条规则是特别针对RWS=SWS的。我们把确定适用于RWS和SWS的任意值的更一般的规则留作习题。还要注意，窗口的大小和序号空间之间的关系依赖于一个很明显以至于容易被忽略的假设，即帧在传输中不重新排序。因为在直接的点到点链路上，一个帧不可能赶上另一个帧。然而，我们将在第5章看到用在一个不同环境中的滑动窗口算法，并且需要设计另一条规则。

3. 滑动窗口的实现

前面的例程说明如何实现滑动窗口算法的发送和接收。例程取自一个正在使用的协议，即滑动窗口协议（Sliding Window Protocol，SWP）。只要不涉及协议图中的邻近协议，我们就用高层协议（High-Level Protocol，HLP）表示SWP上层的协议，用链路层协议（Link-Level Protocol，LLP）表示SWP下层的协议。

我们从定义一对数据结构开始。首先，帧首部非常简单：它包含一个序号（SeqNum）和一个确认号（AckNum）；还包含一个标志（Flags）字段，表明帧是一个ACK帧还是携带数据的帧。

```
typedef u_char SwpSeqno;

typedef struct {
    SwpSeqno   SeqNum;   /* sequence number of this frame */
    SwpSeqno   AckNum;   /* ack of received frame */
    u_char     Flags;    /* up to 8 bits worth of flags */
} SwpHdr;
```

其次，滑动窗口算法的状态有如下结构。对于协议的发送方，这个状态包括本节前面所述的变量LAR和LFS，以及一个存放已传出但尚未确认的帧的队列（sendQ）。发送方状态还包含一个计数信号量（counting semaphore），称为sendWindowNotFull，下面将介绍如何使用它。但一般来说，信号量是一个支持semWait和semSignal操作的同步原语。每次调用SemSignal操作，信号量加1；每次调用SemWait操作，信号量减1。如果信号量减小，导致它的值小于0，那么调用进程阻塞（挂起）。只要执行了足够的semSignal操作而使信号量的值增大到大于0，就允许恢复在调用semWait的过程中阻塞的进程。

对于协议的接收方，状态包含变量NFE。这是下一个希望接收的帧（next frame expected），正如本节前面所述，其序号比最后接收的帧（LFR）的序号大1。还有一个存放已收到的错序帧的队列（recvQ）。最后，虽然未显示，但发送方和接收方的滑动窗口的大小分别由常量SWS和RWS表示。

```
typedef struct {
    /* sender side state: */
    SwpSeqno    LAR;        /* seqno of last ACK
                               received */
    SwpSeqno    LFS;        /* last frame sent */
    Semaphore   sendWindowNotFull;
    SwpHdr      hdr;        /* pre-initialized header */
    struct sendQ_slot {
```

```
        Event   timeout;    /* event associated with send
                              -timeout */
        Msg     msg;
    }   sendQ[SWS];

    /* receiver side state: */
    SwpSeqno    NFE;        /* seqno of next frame
                               expected */
    struct recvQ_slot {
        int     received;  /* is msg valid? */
        Msg     msg;
    }   recvQ[RWS];
} SwpState;
```

SWP 的发送端是由 sendSWP 过程实现的，这个例程比较简单。首先，semWait 使这个进程在一个信号量上阻塞，直到它可以发送另一帧。一旦允许开始发送，sendSWP 设置帧首部中的序号，将此帧的副本存储在发送队列（sendQ）中，调度一个超时事件以便处理帧未被确认的情况，并将帧发送给称作 LINK 的下层协议。

值得注意的一个细节是恰巧在调用 msgAddHdr 之前调用 store_swp_hdr。该例程将存有 SWP 首部的 C 语言结构（state->hdr）转化为能够安全放在消息前面的字节串（hbuf）。这个例程（未显示）必须将首部中的每个整数字段转化为网络字节顺序，并且去掉编译程序加入 C 语言结构中的任意填充。7.1 节将详细讨论字节顺序的问题，但现在，假设这个例程将多字整数中的最高有效位放在最高地址字节就足够了。

这个例程的另一个复杂性是使用 semWait 和 sendWindowNotFull 信号量。sendWindowNotFull 被初始化为发送方滑动窗口的大小 SWS（未显示这一初始化）。发送方每传输一帧，semWait 操作将这个数减 1，如果减小到 0，则阻塞发送方进程。每收到一个 ACK，在 deliverSWP 中调用 semSignal 操作（如下所示）将此数加 1，从而解除等待中的发送方进程的阻塞。

```
static int
sendSWP(SwpState *state, Msg *frame)
{
    struct sendQ_slot *slot;
    hbuf[HLEN];

    /* wait for send window to open */
    semWait(&state->sendWindowNotFull);
    state->hdr.SeqNum = ++state->LFS;
    slot = &state->sendQ[state->hdr.SeqNum % SWS];
    store_swp_hdr(state->hdr, hbuf);
    msgAddHdr(frame, hbuf, HLEN);
    msgSaveCopy(&slot->msg, frame);
    slot->timeout = evSchedule(swpTimeout, slot,
        SWP_SEND_TIMEOUT);
    return send(LINK, frame);
}
```

在到达 SWP 的接收端之前，需要协调一下表面的不一致。一方面，我们一直声称高层协议通过调用 send 操作实现对低层协议服务的调用。因此期望通过 SWP 发送消息的协

议将会调用 send（SWP，packet）。另一方面，实现 SWP 发送操作的过程是 sendSWP，它的第一个参数是一个状态变量（SwpState）。怎么进行协调呢？答案是操作系统提供了将 send 的普通调用转换成一个协议特定的 sendSWP 调用的粘合代码。该粘合代码将 send 的第一个变量（神奇的协议变量 SWP）映射成两个指针：一个 sendSWP 的函数指针和一个 SWP 工作所需要的协议状态指针。我们让高层协议不直接通过普通的函数调用实现协议特定的函数调用的原因是想限制高层协议对低层协议编码的信息量，这将使得以后改变协议图配置变得容易。

现在来看 deliver 操作的 SWP 的特定协议实现，它在过程 deliverSWP 中给出。这个例程实际上处理两种不同类型的输入消息：本节点先发出帧的 ACK 和到达这个节点的数据帧。在某种意义上，这个例程的 ACK 部分是与 sendSWP 中所给算法的发送方相对应的。通过检验首部的 Flags 字段可以确定输入的消息是 ACK 还是一个数据帧。注意，这种特殊的实现不支持数据帧中捎带 ACK。

当输入帧是 ACK 时，deliverSWP 仅仅在发送队列（sendQ）中寻找与此 ACK 相应的位置，取消超时事件，并且释放保存在那个位置的帧。由于 ACK 可能是累积的，所以这项工作实际上是在一个循环中进行的。对于这种情况值得注意的另一个问题是调用子例程 swpInWindow。这个子例程在下面给出，它确保被确认帧的序号是在发送方当前希望收到的 ACK 的范围之内。

当输入帧包含数据时，deliverSWP 首先调用 msgStripHdr 和 load_swp_hdr 以便从帧中提取出首部。例程 load_swp_hdr 对应于前面讨论的 store_swp_hdr，它将一个字节串转化为保存 SWP 首部的 C 语言数据结构。然后 deliverSWP 调用 swpInWindow 以确保帧序号在期望的序号范围内。如果是这样，那么例程在已收到的连续帧的集合上循环，并通过调用 deliverHLP 例程将它们传给高层协议。它也要向发送方发回累积的 ACK，但是通过接收队列上的循环来实现（不用本节前面给出的在实际描述中使用的 SeqNumToAck 变量）。

```
static int
deliverSWP(SwpState state, Msg *frame)
{
    SwpHdr   hdr;
    char     *hbuf;

    hbuf = msgStripHdr(frame, HLEN);
    load_swp_hdr(&hdr, hbuf)
    if (hdr->Flags & FLAG_ACK_VALID)
    {
        /* received an acknowledgment---do SENDER side */
        if (swpInWindow(hdr.AckNum, state->LAR + 1,
            state->LFS))
        {
            do
            {
                struct sendQ_slot *slot;

                slot = &state->sendQ[++state->LAR % SWS];
```

```
                evCancel(slot->timeout);
                msgDestroy(&slot->msg);
                semSignal(&state->sendWindowNotFull);
            } while (state->LAR != hdr.AckNum);
        }
    }

    if (hdr.Flags & FLAG_HAS_DATA)
    {
        struct recvQ_slot *slot;

        /* received data packet---do RECEIVER side */
        slot = &state->recvQ[hdr.SeqNum % RWS];
        if (!swpInWindow(hdr.SeqNum, state->NFE,
            state->NFE + RWS - 1))
        {
            /* drop the message */
            return SUCCESS;
        }
        msgSaveCopy(&slot->msg, frame);
        slot->received = TRUE;
        if (hdr.SeqNum == state->NFE)
        {
            Msg m;

            while (slot->received)
            {
                deliver(HLP, &slot->msg);
                msgDestroy(&slot->msg);
                slot->received = FALSE;
                slot = &state->recvQ[++state->NFE % RWS];
            }
            /* send ACK: */
            prepare_ack(&m, state->NFE - 1);
            send(LINK, &m);
            msgDestroy(&m);
        }
    }
    return SUCCESS;
}
```

最后，swpInWindow 是一个简单的子例程，它检查一个给定的序号是否落在某个最大和最小序号之间。

```
static bool
swpInWindow(SwpSeqno seqno, SwpSeqno min, SwpSeqno max)
{
    SwpSeqno pos, maxpos;

    pos    = seqno - min;       /* pos *should* be in range [0,,MAX) */
    maxpos = max - min + 1;     /* maxpos is in range [0,,MAX] */
    return pos < maxpos;
}
```

4. 帧顺序和流量控制

滑动窗口协议可能是计算机网络中最著名的算法。然而，关于该算法易产生的混淆是，它可以用于三个不同的任务，第一个任务是本节的重点，即在不可靠链路上可靠地传输帧。（一般说来，该算法被用于在一个不可靠的网络上可靠地传输消息。）这是算法的核心功能。

滑动窗口算法的第二个任务是用于保持帧的传输顺序。这在接收方比较容易实现，因为每个帧有一个序号，接收方只要保证在它已经向下一个高层协议传递所有序号比当前帧序号小的帧之前，不向上传送帧。就是说，接收方缓存（即不传送）错序的帧。尽管我们也可以想象另一个版本，即接收方将帧传给下一个协议而不等待先前传送的所有帧都到达，但是本节描述的滑动窗口算法版本采用的方法是保持帧的顺序。我们应该问自己的一个问题是：是否确实需要滑动窗口协议来保持帧的顺序，或者说，在链路层实现这样的功能是否是不必要的。不幸的是，目前我们对网络体系结构的了解还不足以回答这个问题。我们首先需要理解的是，点到点链路序列如何由交换机连接成一条端到端的路径。

滑动窗口算法的第三个任务是，它有时支持流量控制（flow control）。这是一种接收方能够控制发送方的反馈机制，用于抑制发送方发送速度过快，即抑制传输比接收方所能处理的更多的数据。这通常通过扩展滑动窗口协议来完成，使接收方不仅确认收到的帧，而且通知发送方它还可接收多少帧。可接收的帧数对应于接收方有多大的空闲缓冲区空间。在按序传递的情况下，在将流量控制并入滑动窗口协议之前，我们应该确信流量控制在链路层是必要的。

结论 尚未讨论的一个重要概念是系统设计原理，我们称之为相关性分离（separation of concerns）。即必须仔细区别有时交织在一种机制中的不同功能，并且必须确信每一个功能都是必需的，而且是以最有效的方式得到支持。在这种特定的情况下，可靠传输、按序传输和流量控制有时组合在一个滑动窗口协议里，我们应该反思，在链路层这样做是否正确。带着这样的疑问，我们将在第3章（说明X.25网络如何用它实现跳到跳的流量控制）和第5章（描述TCP如何用它实现可靠的字节流信道）重新探讨滑动窗口算法。

2.5.3 并发逻辑信道

用在ARPANET网中的数据链路协议为滑动窗口协议提供了一种有趣的变换，虽然它仍采用简单的停止-等待算法，但它能保持管道满载。这种方法的一个重要结果是，在一个给定链路上传输的帧并不保持任何特定的顺序，该协议也没有流量控制。

我们称为并发逻辑信道（concurrent logical channel）的ARPANET协议的基本思想是，在一个点到点链路上多路复用多个逻辑信道，并且在每个逻辑信道上运行停止-等待算法。在任意逻辑信道上传输的帧之间没有任何关系，同时因为在每个逻辑信道上可以有一个不同的待确认帧，所以发送方可保持链路满载。

更准确地说，发送方为每一信道状态留有3个比特：一个布尔量，说明信道当前是否处于忙碌状态；1比特序号，说明下次在这个逻辑信道上发送的帧的序号；以及下一帧的序号，说明期望到达这一信道的帧的序号。当节点有一个帧要发送时，它使用序号最小的空闲信道，而其他方面它的表现就像停止-等待一样。

实际上，ARPANET网在每个地面链路上支持8个逻辑信道，在每个卫星链路上支持16

个逻辑信道。在地面链路中，每帧的首部包含了3比特的信道号和1比特的序号，总共4比特。当RWS=SWS时，这恰是滑动窗口协议在链路上最多支持8个待确认帧所需的比特数。

2.6 以太网和多路访问网络（802.3）[⊖]

毫无疑问，以太网是最近20年中最成功的局域网技术。它是20世纪70年代中期由施乐公司的帕洛阿尔托研究中心（PARC）开发的。以太网是一个正在使用的、更通用的带冲突检测的载波监听多路访问（CSMA/CD）局域网技术的例子。

正如CSMA的名字所表明的，以太网是一个多路访问网络，即一组节点通过一个共享链路发送和接收帧。因此你可以把以太网想象成一辆公共汽车，可以从多个站点进入。在CSMA/CD中，“载波监听”意味着所有节点可识别链路的忙或闲，“冲突检测”意味着当一个节点传输时要监听线路，因此可以检测到正在传输的帧与另一节点传输的帧发生干扰（冲突）。

以太网可以追溯到早期由夏威夷大学开发的一个分组无线网——Aloha，它支持横跨夏威夷岛的计算机通信。像Aloha网一样，以太网所面临的主要问题是如何调解对一个共享介质的公平而有效的访问（在Aloha中，介质是空气，而在以太网中，介质是一条同轴电缆）。也就是说，Aloha和以太网的中心思想是一个控制每个节点何时能传输的算法。

有趣的是，当今的以太网链路主要是点对点形式的，将主机连接到以太网交换机，或者交换机之间互相连接。因此，“多路访问技术”在今天的以太网中没有使用。同时，无线网络越来越流行，因此起源于Aloha的多路访问技术当前被广泛用于无线网络中，例如802.11（Wi-Fi）网络。这些网络将在2.7节探讨。

下一章中将介绍以太网交换技术。现在，我们关注一条以太网链路如何工作，多路访问以太网正在成为历史，多路访问网络的机制对于进一步的讨论非常重要，下面会介绍。

1978年，数字设备公司（DEC）和英特尔（Intel）公司联合施乐（Xerox）公司定义了10Mbps以太网标准。这个标准后来成为IEEE标准802.3的基础，它定义了以太网采用的物理介质的标准，包括100Mbps、1Gbps和10Gbps的版本。

2.6.1 物理特性

以太网段是在长度最高可达500m的同轴电缆上实现的（现在的以太网使用双绞线，通常使用“5类”线或光纤，有时一个网段的长度可超过500m）。这个电缆类似于有线电视所用的电缆类型。主机通过分接头连接到以太网段上。收发器（transceiver）是一种直接连接到分接头的设备，它检测线路何时空闲，并在主机发送时驱动信号。它也接收输入信号。收发器连到以太网适配器上，而适配器是插在主机上的。这种配置如图2-22所示。

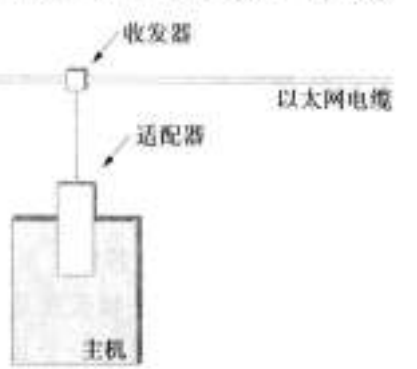


图2-22 以太网收发器和适配器

多个以太网段可由中继器（repeater）连接起来。中继器是一个转发数字信号的设备，很像转发模拟信号的放大器。中继器只理解比特而不理解帧。然而，在任一对主机之间最多可以安装四个中继器，这意味着典型的以太网总共只能达到2 500m。例如，在任一对主机间只

⊖ 参见“实验一”。

使用两个中继器支持类似于图 2-23 的配置，即建筑物垂直方向有一个主干网段，同时每层楼一个网段。

我们也可以使用多口的中继器，有时称为集线器（hub），如图 2-24 所示。对于从集线器的某个端口输入的信号，它将向其各个端口转发。



图 2-23 以太网中继器



图 2-24 以太网集线器

主机送到以太网上的任何信号都是向全网广播的，即信号在两个方向传播，并由中继器或集线器将信号转发到所有输出网段。连到每一网段末端的终结器吸收信号并阻止它反射和干扰后续信号。原先的以太网使用 2.2 节描述的曼彻斯特编码方案，而今天的高速以太网使用 4B/5B 编码方案或类似的 8B/10B 编码方案。

了解下面一点是很重要的：不管一个给定的以太网是只跨越一个网段，还是由中继器互联的一个线性网段序列，或者是由一个集线器将多个网段连接而成的星形配置，这种以太网上任意一台主机传输的数据都能够到达所有其他主机。这是好的一面，而不好的一面是，所有主机竞争访问同一条链路，结果处在同一个冲突域（collision domain）中。以太网的多路访问协议需要处理冲突域中链路的竞争问题。

2.6.2 访问协议

现在把注意力转向控制访问共享以太网链路的算法，这种算法一般称为以太网的介质访问控制（Media Access Control，MAC），通常在网络适配器上以硬件方式实现。我们并不描述硬件本身，而是考虑它实现的算法。下面首先描述以太网的帧格式和地址。

1. 帧格式

以太网帧格式如图 2-25 所示。64 位的 Preamble（前同步码）允许接收方与信号同步，它是一个 0 与 1 的交替序列，各用一个 48 位的地址标识源主机和目的主机。分组 Type（类型）字段用作多路分解密钥，即标识应该将这个帧传递给哪一个高层协议。每帧最多包含 1 500 字节数据，至少包含 46 字节数据，即使这意味着主机在传帧之前必须对它进行填充。规定最小帧长度是因为帧必须足够长才可能检测到冲突，下面将详细讨论这一点。最后，每一帧包含一个 32 位的 CRC。像 2.3.2 节中描述的 HDLC 协议一样，以太网是一个面向比特的组帧协议。注意，从主机的角度看，以太网帧有一个 14 字节的首部：两个 6 字节的地址和一个 2 字节的类型字段。发送适配器在发送之前加上前同步码、CRC 和后同步码，接收方适配器再去掉它们。

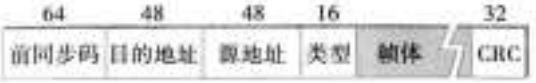


图 2-25 以太网帧格式

2. 地址

以太网上的每台主机——实际上世界上每个以太网的每台主机——都有一个唯一的以太网地址。从技术上讲，地址属于适配器而不是主机，通常固化在 ROM 中。以太网地址以可读的方式显示，即由冒号分隔的 6 个数。每个数对应于 6 字节地址的 1 个字节，并且由一对 16 进制数给出，每个 16 进制数对应 4 比特，而且去掉前导 0。例如，8:0:2b:e4:b1:2 是一个可读的以太网地址，表示为

00001000 00000000 00101011 11100100 10110001 00000010

为了保证每个适配器得到一个唯一的地址，需要对每个以太网设备制造商分配一个不同的前缀，这个前缀必须加到他们制造的每一个适配器的地址上。例如，先进设备公司（AMD）分配到的 24 位的前缀是 x080020（或 8:0:20）。然后，制造商必须保证其生产的加上后缀的地址是唯一的。

在以太网上传输的每一帧可由连到以太网上的所有适配器接收到。每个适配器将这些帧的地址与自己的地址比较，仅向主机传递发送给自己的帧。（可对一个适配器编程，使其以混杂（promiscuous）模式运行，在这种情况下，它向主机传递所有收到的帧，但这不是常规模式。）除了这些单播（unicast）地址外，一个由全 1 构成的以太网地址表示一个广播（broadcast）地址，所有的适配器向主机传递具有广播地址的帧。类似地，所有第一位为 1 但不是广播地址的地址称为多播（multicast）地址。一台给定的主机可对它的适配器编程以接收多播地址的某个集合。多播地址用于向以太网上某台主机的子集传送消息（例如所有文件服务器）。总之，一个以太网适配器接收所有帧，并且接收：

- 编址为它自己地址的帧。
- 编址为广播地址的帧。
- 编址为多播地址的帧，如果适配器在监听那个地址。
- 所有帧，如果适配器处于混杂模式。

以太网适配器只向主机传递它接收到的帧。

3. 发送器算法

正如我们已看到的，在以太网协议中，接收方一边是简单的，真正的难点是在发送方一边实现的，发送器算法定义如下。

当适配器有一帧要发送并且线路空闲时，它立即发送这一帧，不存在与其他适配器协商的问题。消息中 1 500 字节的上界意味着适配器只能以一段定长时间占用线路。

当适配器有一帧要发送并且线路忙时，则等待线路空闲，然后立即发送。[⊖] 因为要发送帧的适配器无论线路何时空闲，都以概率 1 发送，因此称以太网是 1-坚持（1-persistant）协议。一般来讲，p-坚持（p-persistant）的算法在线路变为空闲时，以概率 $0 \leqslant p \leqslant 1$ 发送，并以概率 $q=1-p$ 推迟发送。选 $p<1$ 的原因是，可能有多个适配器在等待线路成为空闲，

⊖ 更准确地说，在一帧结束后开始传输下一帧之前，所有适配器都会等待 9.6μs。对于第一帧的发送方和正在监听以等待线路变为空闲状态的节点而言均是如此。

我们不想让它们同时开始发送。比如，如果每一个适配器都以概率33%立即发送，那么最多有三个适配器在等待发送，但线路空闲时将只有一个适配器开始发送。尽管如此，以太网适配器在看到网络变为空闲时总是立即发送的，而且这样做效率很高。

对于 $p<1$ 时的 p-坚持协议的完整细节，你可能想知道掷硬币输掉（即决定推迟）的发送方在能够发送之前需要等待多久。最先开发这种类型协议的 Aloha 网的答案是，将时间分成离散的时间片，每个时间片对应于传输一个完整帧所用的时间。只要节点有一帧要发送，并且监听到一个空（空闲）的时间片，它就以概率 p 传输，或以概率 $q=p-1$ 推迟到下一个时间片。如果下一个时间片也是空的，那么节点再次分别以概率 p 和 q 决定传输或推迟。如果下一个时间片不是空的，即另外某个站已决定传输，那么该节点则需等待下一个空闲时间片并重复算法。

回到以太网的讨论，因为不存在中央控制，所以可能有两个（或更多）适配器同时开始传输，或是因为两者都发现线路空闲或两者都在等待线路变空闲。当这种情况发生时，称为两个（或更多）帧在网上冲突（collide）。因为以太网支持冲突检测，所以每个发送方能确定冲突的发生。当适配器检测出自己的帧与其他帧冲突时，它首先确保传输 32 位干扰序列，然后停止传输。因此，发送器在冲突的情况下将最少发送 96 位：64 位前同步码加 32 位干扰序列。

适配器只发送 96 位（有时称为残缺帧（runt frame））的一种情形是，两台主机彼此接近。如果两台主机相距较远，那么在检测到冲突之前，它们不得不传输较长时间，因而要发送更多的位。事实上，当两台主机处于以太网的两个相对端点时，就会出现最坏的情况。为了确切知道刚刚发送的帧没有与其他帧冲突，发送器可能需要发送 512 位。这并不是巧合，每个以太网帧的长度必须至少为 512 位（64 字节）：14 字节的首部加上 46 字节的数据加上 4 字节的 CRC。

为什么是 512 位？答案与以太网的另一个令人困惑的问题有关：为什么它的长度仅限制在 2 500m？为什么不是 10km 或 1 000km？这两个问题的答案都与下面的事实有关：两个节点相距越远，一个节点发送的帧到达另一个节点的时间越长，网络就越容易在这段时间内遭遇冲突。

图 2-26 表明了最坏的情形，其中主机 A 和主机 B 处于网络的相对两端。假设主机 A 在时刻 t 开始传输一个帧，如图 2-26a 所示。该帧到达主机 B 用了一个链路时延（时延用 d 表示）。这样，A 的帧的第一个比特在时刻 $t+d$ 到达 B，如图 2-26b 所示。假设主机 A 的帧到达前的一个时刻（即 B 仍看到一个空闲线路），主机 B 开始传输它自己的帧。B 的帧将立即与 A 的帧冲突，并且主机 B 将检测到这一冲突，如图 2-26c 所示。如上所述，主机 B 将发送 32 位的干扰序列。（B 的帧是一个残缺帧。）不幸的是，主机 A 直到 B 的帧到达时才知道发生了冲突，这将发生在一个链路时延之后，即在时刻 $t+2\times d$，如图 2-26d 所示。主机 A 必须继续传输直到

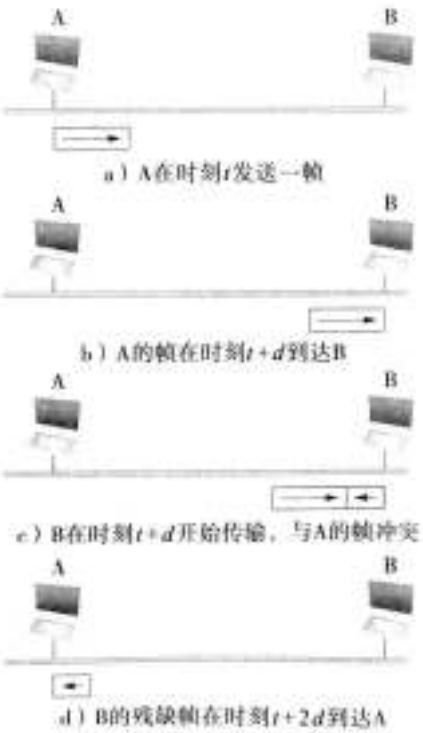


图 2-26 最坏的情形

这一时刻才能检测到冲突。换句话说，主机 A 必须传输 $2\times d$ 的时间以确保检测到所有冲突。考虑最大配置的以太网长度是 2 500m，并且两台主机之间最多可能有 4 个中继器。往返延迟是 51.2μs，在一个 10Mbps 以太网上对应于 512 位。考查这种情况的另一种方式是，为了使访问算法起作用，需要将以太网的最大时延限制为一个相当小的值（例如 51.2μs）。因此，以太网的最大长度必须是 2 500m 左右。

一旦适配器检测到冲突并停止传输，它会等待一定的时间后再尝试。在每次尝试发送但失败后，适配器把等待时间加倍，然后再试。每次使重传尝试间的延迟间隔加倍的策略，一般称为指数退避（exponential backoff）技术。更精确地说，适配器首先随机选择延迟 0 或 51.2μs。如果这次发送失败，那么它（随机选择）等待 0、51.2μs、102.4μs 或 153.6μs 之后再试，这个时间即是 $k\times 51.2$（对于 $k=0\sim 3$）。第 3 次冲突之后，它再随机选择等待 $k\times 51.2$（对于 $k=0\sim 2^3-1$）。一般来讲，算法在 0 和 2^n-1 之间随机选择一个 k 并等待 $k\times 51.2\mu s$，其中 n 是至今经历的冲突次数。适配器在尝试给定次数后将放弃尝试，并向主机报告传输错误。虽然退避算法在上面的公式中将 n 定为 10，但是适配器通常最多尝试 16 次。

2.6.3 以太网使用经验

由于以太网已存在很多年而且很流行，因此积累了大量的以太网使用经验。最重要的观测结果之一是以太网在轻载情况下工作很好。这是因为在重载下（在以太网上，通常超过 30%的利用率就被认为是重载），冲突使网络能力大量浪费且急剧下降。

幸运的是，大部分以太网是在比标准允许方式更保守的方式下使用的。例如，大部分以太网上连接的主机数少于 200，远小于最大数量 1024。（在第 4 章可能找到上限大约是 200 台主机的理由。）类似地，大部分以太网的距离远小于 2 500m，往返延迟接近 5μs 而不是 51.2μs。另一个以太网的实用因素是，即使以太网适配器没有实现链路层流量控制，主机通常会提供一种端到端的流量控制机制。因此，很少出现一台主机连续把帧送到网上的情况。

最后简要说明一下为什么以太网如此成功，这样我们就能够理解以太网具有的能与任何试图替代它的 LAN 技术相竞争的特性。首先，以太网极易管理和维护：不存在出故障的交换机，也没有需要不断更新的路由选择表或配置表，并且易于向网络增加一台新的主机。对管理者而言，很难想象更简单的网络了。其次，它的价格低廉：电缆便宜，唯一的其他成本是每台主机上的网络适配器。这些原因使得以太网的地位非常牢固，而任何试图取代以太网的其他基于交换技术的方法，都需要在网络适配器的成本之上附加额外的基础构件（交换机）投资。我们在下一章可以看到，采用基于交换技术的交换式以太网成功取代了多路访问以太网，一如既往的易于管理特性是其成功的关键。

2.7 无线[⊖]

无线网络技术在很多方面与有线网络不同，但也有许多共同的特性。在无线链路中，比特差错是必须要关注的，因为大部分无线链路都无法避免噪声环境。在无线网络中组帧和可靠性也必须要考虑。与有线网络不同，因为使用无线网络的通常是小型移动设备（如电话和传感器），因此其能量受限（例如电池容量较小）。而且，无线传输设备的功率也不

⊖ 参见“实验二”。

能太高，否则会和其他设备相互干扰，因此在给定频率的情况下设备的功率是受限的。

无线介质也采用多路访问方式，将无线信号仅传输给一个接收者是困难的，避免接收邻居的无线信号也是困难的。因此，在无线链路中介质访问控制是核心问题。在传输无线信号时，控制该信号的接收者是困难的，因此窃听问题也必须考虑。

它们现在在哪？

令牌环网[⊖]

多年来构建局域网一直有两种主要方式：以太网和令牌环网。令牌环网是由 IBM 提出的，标准化为 IEEE 802.5。令牌环网与以太网有许多类似之处：环中采用共享介质，依据分布式算法决定哪个节点何时能够传输帧，每个节点都能够看到其他所有节点传输的分组。

令牌环网和以太网最大的区别是拓扑结构。以太网是总线型拓扑结构，而令牌环网是环型拓扑结构。每个节点都连接着一对邻居，一个上游节点，一个下游节点。“令牌”是一个特定的比特序列，它在环中循环移动，每个节点接收令牌然后转发。当一个有帧要传输的节点看到令牌时，它把令牌从环上取下（即它不转发这个特定的比特模式），而将自己的帧插入环中。沿路的每个节点简单地转发帧，目的节点保存帧的副本，然后将消息转发给环的下一节点。当帧返回到发送方时，这个节点将帧取下来（而不是继续转发它），然后再插入令牌。以这种方式，某个下游节点将有机会发送一个帧。介质访问算法是公平的，因为令牌绕着环循环，每个节点都有机会发送帧。令牌环以轮转的方式为节点提供服务。

过去几十年中出现了许多令牌环网的变体，光纤分布式数据接口（Fiber Distributed Data Interface，FDDI）就是其中之一。最终，令牌环网逐渐被以太网淘汰，尤其是随着以太网交换技术和各种高速以太网（100M 比特和 100G 比特以太网）的出现。

无线网络技术多种多样，每种技术都有不同的特点。对不同技术进行分类的简单方法是依据它们所提供的数据传输速率和传输距离。其他区别包括其占用的电磁频谱（包括是否需要沉默期）和消耗的能量。本节中，我们探讨三种主要的无线技术：Wi-Fi（众所周知的 802.11）、蓝牙以及蜂窝无线标准中的第三代技术（3G）。表 2-4 给出了这些技术的基本情况及其简单对比。

表 2-4 主流无线技术概览

	蓝牙（802.15.1）	Wi-Fi（802.11）	3G 蜂窝
典型链路长度	10m	100m	数十公里
典型数据传输速率	2Mbps（共享）	54Mbps（共享）	数百 kbps（每连接）
典型应用	将外部设备连接到计算机	将计算机连接到有线接入点	将手机连接到有线基站
有线技术对比	USB	以太网	DSL

回想在 1.5 节中，带宽有时表示以赫兹为单位的频段宽度，有时表示链路的数据传输速率。因为这些概念在无线网络中经常遇到，在此我们使用带宽（bandwidth）表示其本来的含义，即频段的宽度，使用数据传输速率（data rate）表示在链路上每秒能够发送的比特数，正如表 2-4 中所示。

⊖ 参见“实验附录 A”。

由于所有无线链路共享同一链路，因而，它的挑战在于有效地共享资源，并且彼此不过度干扰。绝大多数无线链路是通过频率和空间的维度划分实现共享的。在一个特定地区的特定频率可能分配给一个特定的实体，例如一家公司。由于信号从信号源出发经过一定的距离后将减弱（attenuate），因而对一个电磁信号所覆盖的区域面积加以限制是可行的。降低发射器的功率即可缩小信号覆盖的范围。

无线信道的分配是由像美国联邦通信委员会（FCC）这样的政府机构决定的。特定的频段（频率范围）分配用作特定的用途。一些频段留作政府专用，其他频段用作 AM 广播、FM 广播、电视、卫星通信和移动电话。这些频段的特定频率允许某个地理区域的个体组织使用。最后有几个频段留作免许可之用——不需要许可。

利用免许可频率的设备仍然受到某些约束，否则它就是无约束的共享网络了。一个重要的限制是传输功率，这样便限制了信号的范围，使得它不太可能干扰其他的信号。例如，一部无绳电话（免许可设备）的有效范围大概是 100 英尺。

当频谱在许多设备和应用间共享时，一种解决思路是利用扩频（spread spectrum）技术。扩频的思想是在比正常范围更广的频段上传播信号，以减小来自其他设备的影响（扩频最初用于军事领域，因而这些“其他设备”经常试图干扰信号）。例如，跳频（frequency hopping）是一种以随机的频率序列传输信号的扩频技术，它的机制是首先以一个频率传输信号，接着以第二个频率传输，然后以第三个频率，依此类推。传输频率的序列不是真正的随机，而是由一个伪随机数生成器计算生成的。接收方利用与发送方相同的算法（且以相同的值初始化），因而能够与传输者的频率同步以便正确地接收数据帧。这种方式使得两个信号利用相同频率传输信息的可能性几乎为零，从而减小了干扰。

另外一种扩频技术称为*直接序列*（direct sequence），为更好地抗干扰而增加了冗余。每一个数据比特在信号传输中由多个比特位表示，如果一些传输的比特位由于干扰而被破坏了，通常有足够的冗余来恢复最初的位。对于发送者想要传输的每一位比特，实际上发送的是该比特和 n 个随机比特的异或值。在跳频技术中，随机比特序列是由发送者和接收者都知道的伪随机数生成器产生的。所传输的值是一个 n 比特切片编码（n-bit chipping code），它通过 n 倍于帧所需要带宽的频带传播信号。图 2-27 给出了一个 4 比特切片序列的示例。

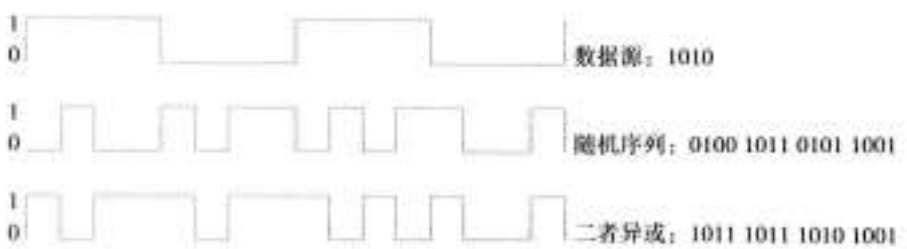


图 2-27 4 比特切片序列实例

电磁频谱的不同部分具有不同的属性，这样使得其中一些适合通信，另一些则不适合。例如，一些频谱能穿透建筑物而有些则不能。政府仅控制基本通信部分：无线电和微波范围。随着对基本频谱需求的增加，当模拟电视逐渐被数字电视淘汰时，人们开始关注一些重新成为可用资源的频谱。[⊖]

⊖ 由于视频编码和调制技术的发展，在数字视频传输中，每个电视频道所需的频谱减少了。

当今的无线网络中有两种不同类型的端节点。一种端节点称为基站（base station），通常不移动，但有线路（至少高带宽）连接到因特网或者其他网络，如图 2-28 所示。在链路另一端的节点（在此表现为客户节点）通常是移动的，并且依靠与基站的连接实现与其他节点的通信。

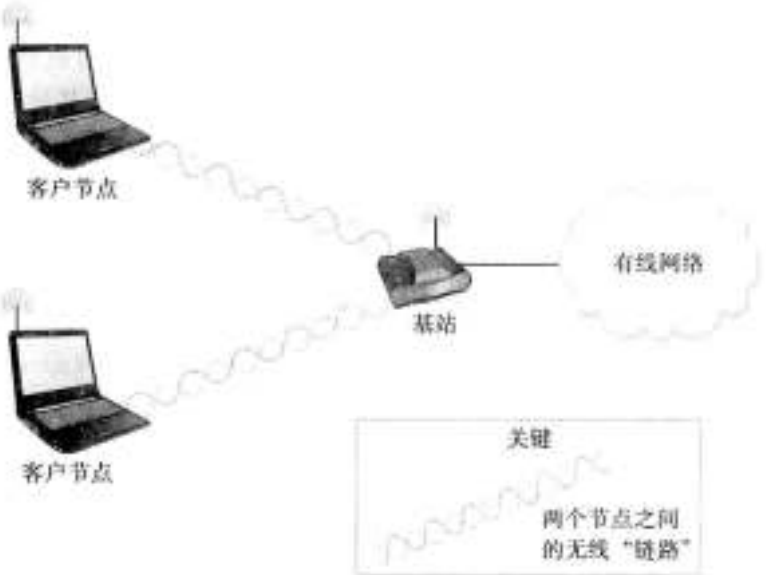


图 2-28 利用基站的无线网络

在图 2-28 中，利用一对波浪线表示在两个设备间（即在基站和一个客户节点之间）提供的无线“链路”。无线通信的一个有趣的特点是它自然支持点到多点的通信，这是因为由一个设备发送的无线电波能够同时被多个设备收到。然而，为高层协议创建一个点到点的链路通常是有用的，我们将在本节后面看到它的运作实例。

注意在图 2-28 中，非基站（客户）节点之间的通信是通过基站进行路由的。尽管由一个客户节点发出的无线电波可能会被其他的客户节点收到，但是普通的基站模型不允许客户节点之间的直接通信。

这种拓扑包含有三种不同级别的移动性。第一级是不移动，这种情况下接收方必须在一个固定位置接收来自基站的定向传输。第二级是在基站的范围内移动，对应的实例是蓝牙技术。第三级是基站之间的移动，蜂窝电话和 Wi-Fi 是对应的实例。

另一种引人注目的无线网络拓扑是网格（mesh）或自组织（ad hoc）网络。在无线网格中，节点是对等的（即没有特别的基站节点）。只要每个节点在前面节点的范围之内，消息就可以由对等节点组成的链向前发送。这种情形可由图 2-29 说明。这样便允许网络的无线部分延伸到一个发射器限制的范围之外。从技术之间竞争的角度，这样使得较短距离的无线网络通信技术得以延伸并可与较长距离的技术抗衡。网格还提供容错机制，这是通过提供获取从点 A 到点 B 消息传输的多种路由实现的。网格能够不断扩展，费用也随之不断增加。另一方面，网格需要的非基站节点的硬件和软件设计实现都具有一定的复杂性，潜在地增加了单位成本以及电力消耗，这是电池设备要考虑的重要内容。无线网格网络具有较强的理论研究价值（参见参考文献中的相关内容），但是与基站网络相比，它们依然很不成熟。无线传感器网络是另一种热门的无线技术，它通常可形成无线网格。



图 2-29 无线自组织网络或网格网络

现在我们已经简单介绍了一些常见的无线技术，下面我们进一步介绍这些无线技术的细节。

2.7.1 802.11/Wi-Fi

大部分读者都使用过基于 IEEE 802.11 标准的无线网络，通常称为 Wi-Fi[⊖]。Wi-Fi 在技术上是一个商标，由一个名为 Wi-Fi 联盟的商业组织拥有，确保其产品符合 802.11 标准。类似于以太网，802.11 用于有限的地理范围（家庭、办公室、校园），它的主要挑战是对共享通信介质的协调访问——在此情况下，信号通过空间传播。

1. 物理属性

802.11 定义了许多不同的物理层，其频段不同，提供的数据传输速率也不同，截至目前，802.11n 提供的数据传输速率最高，可达 600Mbps。

最初的 802.11 标准定义了两个基于无线电的物理层标准，一个利用跳频（在 79 个 1MHz 宽的频带上），另一个利用直接序列（使用一个 11 比特的切片序列）。两者都提供了 2Mbps 的速率。后来又补充了物理层标准 802.11b，利用不同的直接序列，802.11 可提供最高 11Mbps 的速率。这三个标准运行在电磁波谱 2.4GHz 的免许可频带。接着又制定了 802.11a 标准，利用正交频分多路复用（Orthogonal Frequency Division Multiplexing，OFDM）——FDM 的一种，可使发送速率达到 54Mbps。802.11a 运行在 5GHz 的免许可频段。一方面，该频段使用较少，因此干扰也较少。另一方面，很多信号被吸收了，且几乎都被限制在视线之内。接下来是 802.11g，802.11g 也使用 OFDM，传输速率可达到 54Mbps，它向后兼容 802.11b（可用 2.4GHz 频带）。

最新的标准是 2009 年提出的 802.11n（尽管先前的标准产品已经存在），802.11n 使用多路天线，允许更大的无线信道带宽，尽最大可能提高了数据传输速率。多路天线的使

⊖ 类比于 Hi-Fi（High Fidelity，高保真），Wi-Fi 是意为“无线保真”，还是仅为一个除 802.11 外别无所指的好记的名字，对此人们尚存争议。

用通常称为 MIMO，用于多路输入输出。

对于商业产品来说支持多种 802.11 标准是必要的，一些基站支持所有四种标准（a、b、g 和 n）。这不仅保证了支持任一标准的设备的兼容性，而且使得在特定环境中两个产品可以选择其最高的带宽。

值得注意的是，所有 802.11 标准都定义了其能够支持的最大比特传输率，它们在低比特率时工作得更好，例如，802.11a 支持的比特率为 6Mbps、9Mbps、12Mbps、18Mbps、24Mbps、36Mbps、48Mbps 和 54Mbps，在低比特率时，它更容易对受噪声干扰的传输信号进行解码。为了实现不同的比特率，需要用到不同的调制方法，此外，用于差错纠正编码的冗余信息的数量也是不同的（参见 2.4 节中关于差错检测码的介绍）。高冗余信息意味着低有效数据传输效率（因为有更多的传输比特是冗余的）。

系统希望在噪声环境下获得更优的比特传输速率，比特率选择算法可能很复杂（在"扩展阅读"一节中有个例子）。有趣的是，802.11 标准没有制定特定的方法，这些算法可以由供应商提供。选择比特率的基本方法是通过直接在物理层测量信噪比（SNR）来测试误比特率，或者通过度量分组被成功传输和确认的频率来测量 SNR。在有些方法中，发送方会通过高比特率发送一个或多个分组，看其是否成功，据此探测比特率。

2. 冲突避免

乍一看，无线网协议好像恰好遵循与以太网相同的算法（等到链路成为空闲后再传输，并且如果发生冲突则退避）。粗略地说，这正好是 802.11 所做的。然而，与无线网络不同，以太网中的节点在接收其他节点的传输时，能够同时发送数据，而无线网络中的节点不具备该特性，这使得在无线网络中的冲突检测非常复杂。无线网络中节点通常不能同时收发数据（以同样的频率），传输器的能量通常高于接收信号，一个无线节点可能因为太远而不能接收其他节点的传输，这种情况相当复杂，下面将进行详细描述。

考虑图 2-30 所示的情况，其中 A 和 C 都在 B 的有效范围内，但是 A 与 C 彼此间不能直接通信。假设 A 和 C 都想与 B 进行通信，因此都向 B 发送一个帧。A 和 C 都察觉不到对方，因为它们的信号不能传送那么远。这两个帧在 B 处互相冲突，但与以太网不同，A 和 C 都不知道这一冲突。此时 A 和 C 分别称为对方的隐藏节点（hidden node）。

一个相关的问题称为暴露节点问题（exposed node problem），在图 2-31 所示的情况下发生，其中 4 个节点中的每个节点都能发送和接收信号，这些信号只能到达紧靠它左右的

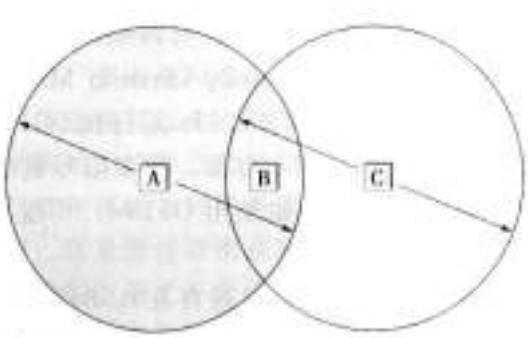


图 2-30 隐藏节点问题。虽然 A 和 C 相互隐藏，但它们的信号可能在 B 处冲突（B 的到达未显示）

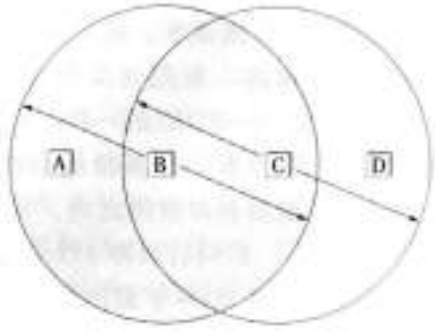


图 2-31 暴露节点问题。虽然 B 和 C 的信号彼此暴露，但 B 给 A 发送与 C 给 D 发送不会互相干扰（A 和 D 的到达未显示）

节点。例如，B能够与A和C交换帧，但不能到达D，而C能够到达B和D但不能到达A。假设B向A发送数据，节点C察觉到这一通信，因为它侦听到了B的传输。如果C只是因为听到B的传输就断定它不能向任何节点传输，这将是错误的。例如，假设C希望向节点D传输数据，因为C向D传输不会干扰A从B接收，所以这不成问题。(它将干扰A向B发送，但在本例中B正在发数据。)

802.11使用CSMA/CA解决这两个问题，这里CA表示冲突避免(avoidance)，不同于以太网CSMA/CD中的冲突检测(detection)。具体原理下面将进行介绍。

载波监听看起来似乎很简单：在发送分组前，监听信道中是否有信号在传输，如果没有就发送。然而，因为隐藏终端问题，仅仅没有监听到其他节点的信号并不能保证在接收端不会发生冲突。因此，在CSMA/CA中需要接收方给发送方明确发送一个ACK。如果分组被成功解码并通过了CRC校验，则接收方给发送方发送一个ACK。

需要注意的是如果冲突发生了，整个分组将变得无效。因此，802.11增加了称为RTS-CTS(请求发送-清除发送)的机制。这可以在一定程度上解决隐藏终端问题。发送方给期望的接收方发送一个短分组RTS，如果分组被成功接收，接收方会通过一个短帧CTS进行响应。即使RTS没有被隐藏终端听到，CTS也会被它听到。这会告诉在接收方信号范围内的节点在一段时间内不能发送分组，期望传输的时间长短包含在RTS和CTS分组中。该期望时间到达后，经过一个小的时隙，信道就变为可用，其他节点可以尝试发送。

当然，如果两个节点同时检测到一个空闲链路并试图发送一个RTS帧，那么它们的RTS帧将彼此冲突。当发送方在一段时间之后没有收到CTS帧时，它就知道发生了冲突，在这种情况下，它们都等待一段随机时间后再试。一个给定节点延迟的时间是由用在以太网上的相同的指数退避算法定义的(见2.6.2节)。

在一次成功的RTS-CTS交换后，发送方发送其数据分组，发送完后接收该分组的ACK。如果ACK超时，发送方使用上述方法试图再次请求信道。当然，这时其他节点可能也会试图访问信道。

3. 分布式系统

如上所述，802.11适合网格(自组织)的网络拓扑，针对网格网络的802.11s标准即将完成。然而，当前几乎所有802.11网络都使用面向基站的拓扑。

网络中所有节点并不是完全相同的，有些节点允许移动(例如便携式电脑)，有些节点连接到有线网络基础设施上。802.11称这些基站为接入点(Access Point，AP)，并且它们是通过一个所谓的分布式系统(distribution system)彼此连接的。图2-32给出一个连接三个接入点的分布式系统，每个接入点为某一区域的节点服务。每个接入点在其适当频率范围的信道上工作，并采用与它的邻居不同的信道。

分布式系统的细节在这个讨论中并不重要，例如，它可能是一个以太网。唯一重要的是分布式网络运行在链路层，即它与无线链路在同一个协议层运行，换言之，它不依赖于任何高层协议(如网络层)。

如果两个节点彼此可达，它们就能够直接通信，但这种配置的思想是，每个节点与各自的一个接入点相联系。例如，对于节点A与节点E通信，A首先向它的接入点(AP-1)发送一个帧，接入点通过分布式系统向AP-3转发这一帧，最后AP-3向E发送该帧。

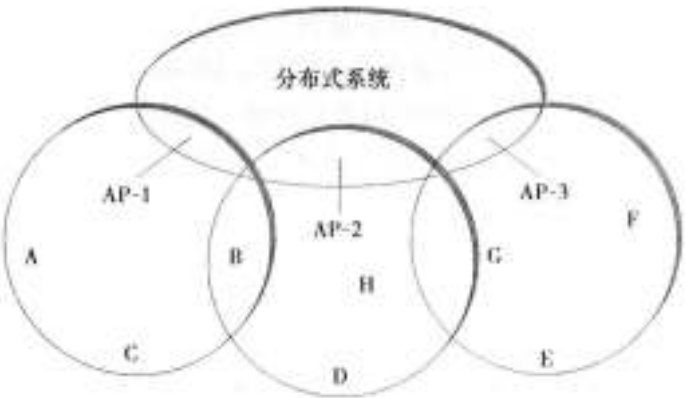


图 2-32 接入点连接到分布式系统

AP-1如何知道向 AP-3 发送消息超出了 802.11 的范围，它可以使用下一章（3.1.4 节）描述的桥接协议。802.11 详细说明了节点如何选择接入点，更有趣的是当节点从一个区域移动到另一个区域时该算法如何工作。

选择 AP 的技术称为*扫描*（scanning），它包含以下四步：

1）节点发送一个 Probe 帧。

2）所有可达的 AP 用一个 Probe Response 帧应答。

3）节点从中选择一个 AP，并向那个 AP 发送一个 Association Request 帧。

4）AP 用一个 Association Response 帧应答。

无论何时，当一个节点加入网络或者对当前 AP 不满意时，它就会用到这个协议。例如，当前 AP 的信号由于节点远离而变弱时就会发生这种情况。一个节点无论何时获得一个新的 AP，这个新的 AP 就通过分布式系统将变化通知给它的旧 AP（在步骤 4）中发生）。

考虑图 2-33 所示的情况，其中节点 C 从 AP-1 服务的区域移动到 AP-2 服务的区域。在移动时它发送 Probe 帧，最终收到来自于 AP-2 的 Probe Response 帧。在某一点上，C 宁愿选择 AP-2 而不是 AP-1，因此它将自己与 AP-2 的接入点联系起来。

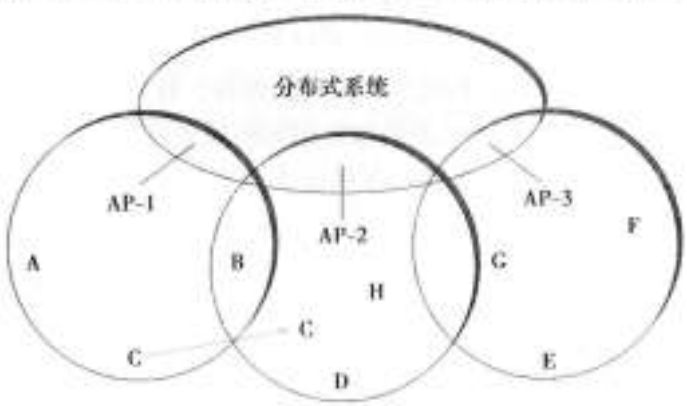


图 2-33 节点的移动性

刚刚描述的机制称为*主动扫描*（active scanning），因为节点在主动搜索接入点。AP 会定期发送一个通知 AP 能力的 Beacon 帧，其中包括 AP 支持的传输率，这样的情况称为*被动扫描*（passive scanning）：一个节点可以根据 Beacon 帧，只通过向接入点发回一个 Association Request 帧而关联到 AP。

4. 帧格式

大部分 802.11 帧格式如图 2-34 所示。这种帧包含源节点地址和目的节点地址（长度都是 48 比特），最多 2 312 字节数据，以及一个 32 比特的 CRC。Control（控制）字段包含三个重要的子字段（未显示）：一个 6 比特的 Type（类型）字段，指明帧携带的数据是一个 RTS 帧或 CTS 帧，还是由扫描算法使用；两个 1 比特的字段（称为 ToDS 和 FromDS）将在下面描述。

16	16	48	48	48	16	48	0–18,496	32
控制	持续时间	Addr1	Addr2	Addr3	SeqCtrl	Addr4	有效载荷	CRC

图 2-34 802.11 帧格式

802.11 帧格式独具特色之处是它包含四个地址而不是两个地址。如何解释这些地址取决于帧的 Control 字段中 ToDS 位和 FromDS 位的设置。考虑到帧必须通过分布式系统转发的可能性，这意味着原来的发送方不一定与最近的传输节点相同。同样也可以这样考虑目的地址。在最简单的情况下，当一个节点直接向另一个节点发送数据时，两个 DS 位都设为 0，Addr1 标识目的节点，Addr2 标识源节点。在最复杂的情况下，两个 DS 位都设置为 1，表明消息从一个无线节点进入一个分布式系统，然后从分布式系统到达另一个无线节点。两个 DS 位都置为 1 时，Addr1 标识最终目的站，Addr2 标识直接发送方（从分布式系统向最终目的站转发帧的节点），Addr3 标识中间目的站（从无线节点接收帧并通过分布式系统将其转发的节点），Addr4 标识最初的源节点。在图 2-32 给出的例子中，Addr1 对应于 E，Addr2 标识 AP-3，Addr3 对应于 AP-1，而 Addr4 标识 A。

2.7.2 蓝牙（802.15.1）

蓝牙技术填充了移动电话、PDA、笔记本电脑以及其他个人或外围设备之间的短距离通信的缝隙。例如，蓝牙技术可用于连接移动电话和耳机、笔记本电脑和耳机。概略地讲，蓝牙是利用电线连接两个设备的一种更为方便的替代。在这种应用中，不必提供很大的范围或带宽。这对于由电池供电的设备很有利，其中很重要的原因是耗电不多。这样就能够满足蓝牙的应用目的，多数蓝牙设备电池能量有限（例如无处不在的电话耳机），因此耗电少是非常重要的。[⊖]

蓝牙运行在 2.45GHz 的免许可频段，其链路典型带宽是 1～3Mbps，有效范围大约 10m。正因如此，加之通信设备通常属于个人或一个组织，因此蓝牙有时被归类为个人区域网络（Personal Area Network，PAN）。

有一个称为蓝牙专业因特网组织（Bluetooth Special Znternet Group）的工业联盟为蓝牙制定相关标准。它为一系列应用程序制定了一套完整的协议，超出了链路层对应用协议的定义，称为针对应用的应用协议轮廓（profile）。例如，用于同步 PDA 和个人计算机的应用协议轮廓。就 802.11 而论，另一个应用协议轮廓为移动计算机提供对有线 LAN 的访问，尽管这不是蓝牙的最初目的。IEEE 802.15.1 标准建立在蓝牙的基础上，但是它不包含应用协议。

⊖ 谁真的想在耳朵里放一个高能量的无线电耳机呢？

最基本的蓝牙网络配置称为皮可网（piconet），由一个主设备和7个从设备组成，如图2-35所示。所有通信都发生在主设备和从设备之间，从设备彼此之间不直接通信。由于从设备作用简单，因而它们的蓝牙硬件和软件简单且价格低廉。



图2-35 蓝牙皮可网

由于蓝牙在一段免许可的频段上运行，所以它需要利用一种扩频技术（见本节前面的叙述）来处理频率段中可能的干扰。它利用79个信道（channel）（频率）的跳频，每个每次使用625μs。这为蓝牙利用同步时分多路复用提供了一个自然的时间片。传输一帧占用1、3或5个连续的时间片。只有主设备可以在奇数时间片开始传输。从设备可以在偶数时间片开始传输，但仅回应在先前时间片中主设备的请求，因此不允许在从设备之间进行连接。

从设备可以被停置（parked）：设置其为不活动、低电量状态。停置设备不能与皮可网通信，它只能通过主设备激活。除了活动从设备之外，一个皮可网还可有255个停置设备。

除了蓝牙之外，在低功耗短距离通信领域还有一些其他技术，其中之一就是ZigBee，它由ZigBee联盟设计并被标准化为IEEE 802.15.4，ZigBee用于带宽需求较少和耗电很低以使电池使用时间较长的情况。它的设计宗旨是比蓝牙简单和便宜，其经济性使其适合于廉价设备，例如传感器。传感器是一种越来越重要的网络设备，这些廉价的小设备可以大量部署，用于监控温度、湿度以及建筑物中的能耗。

2.7.3 蜂窝电话技术

蜂窝电话技术起源于声音通信，基于蜂窝标准的数据服务正在变得越来越流行（这得益于移动电话和智能手机（smartphone）的快速增长）。其缺点之一是用户的使用成本高，某些原因是蜂窝使用许可的频谱（在若干年前被蜂窝电话运营商以天文价格买断），用于蜂窝电话（现在是用于蜂窝数据）的频段在世界各地是不同的。例如，在欧洲，用于蜂窝电话的主要频段是900MHz和1 800MHz，在北美，使用850MHz和1 900MHz的频段。

这种在频谱利用上的全球不一致性给那些周游世界的用户带来了一些问题，并为能在多种频率上运行的手机创造了市场（例如，一个三频段手机能在上面提到的四种频段中的三种频段上通信）。然而，与不断增多的一直折磨着蜂窝通信业务的不相容标准相比，该问题则略显苍白。直到最近，才出现了一些在少量标准集上的集中迹象。最后的问题是，绝大多数蜂窝技术是设计用于声音通信的，直至现在才开始适当地支持高带宽数据通信。

像 802.11 和 WiMAX 一样，蜂窝技术依赖对基站的使用，基站是有线网络的一部分。由一个基站天线负责的地理区域称为一个蜂窝（cell）。一个基站可以为一个蜂窝服务，或利用多个方向上的天线为多个蜂窝服务。蜂窝没有明确的边界，它们彼此重叠。在重叠之处，一部移动电话可以与多个基站通信。这一点与图 2-43 表示的 802.11 有些相像。然而，在任意时刻，一部电话仅能与一个基站通信，且受控于该基站。当移动电话离开一个蜂窝时，它移动到由一个或多个蜂窝重叠的区域。当前的基站感觉到该电话的信号变弱，将对该电话的控制交给收到最强信号的基站。如果此时该电话有一个呼叫，该呼叫必须由新基站传递，这称为切换（handoff）。

如上所述，蜂窝通信没有唯一的标准，而是以不同的方式、不同的传输速度支持数据通信的竞争技术集合。这些技术被松散的划分为代（generation），第一代（1G）是模拟的，因而数据通信能力有限。第二代标准转向数字通信，提供无线数据服务，而第三代（3G）标准支持高带宽并同时支持声音和数据传输。目前大部分移动电话网络支持 3G，并且 4G 也开始出现。由于每一代都拥有自己的标准和技术，因此对于一个特定的网络是属于 3G 还是其他经常会有争议（基于市场原因）。

第三代（3G）蜂窝通信的概念在 3G 技术实现之前就已经建立，旨在制定将提供高于 2G 的数据带宽的单一国际标准。不幸的是，单一的标准并没有实现，这种趋势似乎可能要延续到 4G。然而有趣的是，大部分 3G 标准都基于 CDMA（码分多路复用）的变体。

CDMA 使用一种扩频技术使来自多个设备的流量共享信道，每个发送方使用一个和数据传输速率相关的码片速率发送数据，每个发送方的码片速率对于接收方来说是已知的，例如，每个蜂窝网络中的基站会给当前与其关联的每个移动设备分配唯一的码片速率。当大量设备在同样的蜂窝频段中广播其信号时，所有传输看起来是互相混淆的。然而，知道给定发送方码片速率的接收方能够从混淆的信号中提取出它需要的数据。

相对于其他多路复用技术，CDMA 在针对突发数据方面有一些优势。它对于多少用户可以共享一个频段没有限制，只要确保它们都有唯一的码片速率即可。然而误比特率会随着并发发送的增加而增加，这使得它特别适合于有大量用户但是在某段时间内其中的大部分用户都不使用网络的情况，这种情况对于许多数据应用都是成立的，例如 Web 冲浪。实际系统中，许多移动用户同时发送信息的情况并不多见，所以 CDMA 相对于 TDMA 这样的多路复用技术来说具有较好的效率（例如，它能接近香农定理的理论上限值）。

相关主题

无线网络的跨层问题

近几年引起研究界巨大关注的无线网络的一个有趣的方面是其对分层协议体系结构形成了挑战。例如，802.11 标准能够建立链路抽象，以点到点的方式把一个节点连接到另一个节点，因此，任何高层协议都可以把它看作是点到点链路，但这是正确的方法吗？

例如，考虑三个节点 A、B、C 像图 2-30 所示那样排成一行的情况，如果想要从 A 到

C获取一个分组，常规的方法是A发送分组到B，B再发送分组到C。但实际上，给定节点能够发送分组的范围并不一定是个固定的圆圈，随着距离的增加信号会衰减。因此A发送分组到C可能有30%的成功率，而它发送分组到B可能有80%的成功率。所以有时候(30%的概率)可能不需要B把分组转发给C，分组有可能已经到C了。因此C告诉B"不要给我转发分组，我已经收到了"可能会更好些。麻省理工学院已经通过名为Roofnet的无线测试平台测试了该方法，结果显示其超越常规方法，吞吐量大增。但该方法也意味着A、B和C与通过简单链路把它们连接起来不一样了。对于这个情况有人说它违背了分层结构，然而也有人(比如本书作者)佩服这种创造力，因为它通过超越传统分层而提高了性能。

这样的例子数不胜数，从无线链路层向高层传递信息确实有很多好处，它也有助于将信息从物理层传递到链路层。这里有个很好的平衡。分层是工具，没有分层，构造大规模网络可能是很困难的。但是应该意识到，无论什么时候通过分层来隐藏信息，就可能失去那些本不该隐藏的信息。我们应该把分层(或者任何其他抽象形式)当作是工具，而不是不可侵犯的规则。

无线网络的安全性

与有线链路相比，无线链路的明显问题就是不能确定数据已经去了哪里。可能清楚数据已经到达了期望的接收者，但却不清楚有多少其他接收者可能也已经收到了所传输的数据。因此，如果关心数据的私密性，使用无线网络就是个挑战。

即使不关心数据的私密性，或者可能已经采取了一些其他方法(参见第8章中关于该主题的讨论)，但还是需要考虑一个未授权的用户在网络中注入数据的问题，这将会消耗网络资源，例如在住所和ISP之间有限的带宽。

因为这些原因，无线网络需要有一些机制来控制链路本身以及传输的数据。这些机制通常称为无线网络的安全性(wireless security)。安全是个大型主题，在第8章中会详细论述，在8.4.5节会介绍无线网络安全性的细节。

相关主题

卫星通信

无线通信在特定场合的一种应用是基于卫星的。卫星电话利用通信卫星作为基站，在国际上为卫星保留的频段上通信而且能在没有蜂窝基站的地方获得服务。因为服务代价较高，所以卫星电话很少用于蜂窝电话。由于卫星比蜂窝电话塔远，所以传输和接收需要经过更远的距离，卫星电话比现代蜂窝电话大而重。用于电视和无线广播的卫星通信利用了信号广播而不是点对点的优势，使用更为广泛。利用卫星的高带宽数据通信在商业上是可行的，但是它的相对高代价(设备和服务)将其使用范围限制在了没有其他可选通信方式的地区。

2.8 小结

本章介绍了各种各样的链路，它们可以将用户连接到现有的网络，并构造大规模网络。然而这些链路之间有着巨大的差异，存在许多问题和技术需要解决。有五个关键问题必须被解决，以便相互连接的两个或多个节点能够彼此交换报文。

首要问题是在源节点将组成二进制消息的比特编码成信号，然后在接收节点再将信号恢复成比特。这是编码问题，遇到的挑战是需要保持发送方和接收方的时钟同步。我们讨论了四种不同的编码技术：NRZ、NRZI、曼彻斯特和 4B/5B，它们在如何将时钟信息与被发送数据一起编码的问题上有很大区别。编码方案的关键属性之一是效率，即信号脉冲与编码比特的比率。

一旦可以在节点之间发送比特，下一步是确定如何将这些比特打包成帧。这是组帧问题，归结起来就是能辨认每一帧的开始与结束。同样，我们介绍了几种不同的技术，包括面向字节的协议、面向比特的协议和基于时钟的协议。

假设每个节点能够识别构成一个帧的比特集，那么第三个问题就是确定这些比特实际上是否正确，或者它们在发送中是否可能被破坏。这是差错检测问题，我们也介绍了三种不同的方法：循环冗余校验（CRC）、二维奇偶校验及校验和。自然，其中的 CRC 方法提供了最强的保证，并且是链路层上使用最广泛的方法。

如果某些即将到达目的节点的帧包含差错，则这些帧将被丢弃，下一个问题就是如何从这种丢失中恢复，目的是使链路看起来是可靠的。解决这个问题的一般方法称为自动请求重发（ARQ），它使用确认和超时相结合的方法。我们介绍了三种特别的 ARQ 算法：停止-等待、滑动窗口和并发信道。这些算法的吸引人之处在于为了保持信道满载，它们是如何有效使用链路的。

最后一个问题与点到点链路无关，但它是多路访问链路的主要问题：如何调解对一个共享链路的访问，以便使所有节点最终都有机会发送自己的数据。在这种情况下，我们介绍了一些介质访问协议——以太网和几种无线网络协议——它们已经用在局域网的实际建设中。由于无线传输范围有限，一些节点可能彼此隐藏，无线网络的介质访问较为复杂。现在，多数普通无线网络协议将一些节点指定为有线节点或基站节点，而其他移动节点与基站通信。随着网格网络（所有节点均作为对等实体通信）的涌现，无线网络标准和技术得以迅速发展。

接下来会发生什么："物联网"

随着处理能力和存储技术的发展，廉价的低能耗设备在持续增加，互联网中"主机"的概念发生了显著的变化。20 世纪 70 年代和 80 年代因特网中主要使用的是互相连接的复杂计算机，今天因特网中的主机可以是笔记本电脑或移动电话，更小的设备也变得越来越可行，例如传感器和执行器也可以作为因特网中的主机。这些设备虽小，但数量众多，由此出现了"物联网"概念——在物联网中的很多物体，从电灯开关到工厂中的仓储盒，都是互联网中可寻址的"主机"。

大量微小物体组成网络的概念听起来像科幻小说（也许是反乌托邦的小说），但确实有许多具体的实际应用。其中之一是通过网络应用控制各种设备能耗的想法。电灯开关等可以装上传感器（测量电力负荷、环境温度等）和执行器（控制设备何时激活，例如推迟洗衣机的使用，直到用电高峰期过去，这时电费较便宜）。这个概念常常出现在"智能电网"主题中，受今天的能源公司和设备供应商所追捧。

万亿个小型、低能耗、廉价、间歇连接的设备形成的网络对主机技术提出了挑战。举个简单的例子，在第 4 章中将会讨论的 IPv6 的设计要受到影响，需要寻址的数量远多于世界上传统计算机的数量。类似地，需要开发新的路由协议在设备间转发数据，它们有低

的能耗预算和不可靠的无线连接。甚至还需要开发新的操作系统，用于这些能量、CPU和存储资源都受限的微小设备。

“物联网”的未来我们拭目以待，但显而易见物联网已经超越了仅仅互联计算机的视角，互联亿万个智能设备的应用仅仅是个开始。

扩展阅读

在过去的20年中，计算机网络技术方面最重要的文献之一是Metcalf和Boggs撰写的介绍以太网的独创性论文（1976）。许多年之后，Boggs、Mogul和Kent报告了他们使用以太网的实际经验（1988），打破了多年来文献中的许多神话。这两篇文章都是必读的。第三篇文章讨论了包括802.11在内的无线网络的相关问题。

- Metcalf, R., and D. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM* 19 (7): 395-403, July 1976.
- Boggs, D., J. Mogul, and C. Kent. Measured capacity of an Ethernet. *Proceedings of the SIGCOMM'88 Symposium*, pages 222-234, August 1988.
- Bharghavan, V., A. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LANS. *Proceedings of the SIGCOMM'94 Symposium*, pages 212-225, August 1994.

有不计其数的教科书重点强调了网络层次结构的较低层，特别是远程通信（telecommunication）——从电话公司角度的联网。两本较好的书是Spragins等[SHP91]和Minoli[Min93]编写的。还有几本书集中讨论了各种局域网技术。自然，Stallings的书[Sta00]是最全面的，而Jain的书[Jai94]对光通信的低层细节做了很好的介绍。

无线网络是很活跃的研究领域，每年都有新颖且有趣的论文出现。Gupta和Kumar的论文[GK00]建立了关于无线网络容量的理论。Basagni等[BCGS04]对无线自组织网络进行了很好的介绍。Bicket等[BABM05]介绍了Roofnet无线网格网络的实验，Biswas和Morris[BM05]展示了运行在Roofnet上的ExOR。后者的论文是使用跨层信息改进无线网络性能的例子。Jamieson等[JB07]介绍了跨层技术的不同使用在面临比特差错的情况下改善吞吐量的内容。Wong等[WYLB06]关注于如何在无线信道中进行差错率和带宽的权衡，以便提高数据传输的准确率。Katti等[KRH+06]研究了使用网络编码提高无线网络性能的可行性。

最近Xiao等[XCL10]编写的书调研了传感器网络的许多方面。Vasseur和Dunkels[VD10]针对“物联网”如何使用因特网协议实现传感器与其他智能设备的互联提出了一种前向的观点。

针对信息理论，Blahut[Bla87]的书是一本很好的入门读物，同时可以阅读Shannon[Sha48]关于链路容量的论文。

针对关于比特差错的综合介绍，推荐Rao和Fujiwara[RF89]的书。对于循环冗余校验的详细讨论，以及硬件实现的问题，可参考Peterson和Brown的书[PB61]。

最后，我们推荐下列时常更新的网站作为参考：

- http://standards.ieee.org/：IEEE的各种与网络相关的标准的发展状况，包括以太网和802.11。

习题

1. 画出图 2-36 中所示比特模式的 NRZ、曼彻斯特和 NRZI 编码。假设 NRZI 信号从低电平开始。

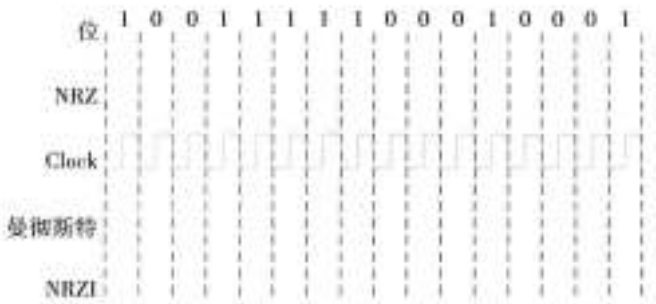


图 2-36 习题 1 图

2. 给出比特序列 1110 0101 0000 0011 的 4B/5B 编码，以及得到的 NRZI 信号。
3. 给出比特序列 1101 1110 1010 1101 1011 1110 1110 1111 的 4B/5B 编码，以及得到的 NRZI 信号。
4. 在 4B/5B 编码（见表 2-2）中，所使用的 5 比特编码中只有两个是以两个 0 结尾的。有多少个可能的 5 比特序列（被现存编码使用或不使用）使其满足最多有一个前导 0 和最多有一个末尾 0 这一较强的限制？所有的 4 比特序列都能映射到这样的 5 比特序列上吗？
5. 假设一个组帧协议使用比特填充，当帧包含比特序列 11010111111010111111010111111110 时，画出链路上传输的比特序列并标记填充的比特。
6. 假设一条链路上到达比特序列 11010111111010111111001011111110。给出去掉任意填充比特之后的帧。指出可能引入帧中的任何差错。
7. 假设一条链路上到达比特序列 01101011111101010011111111011001111110。给出去掉任意填充比特之后的帧，指出可能引入帧中的任何差错。
8. 假设用 BISYNC 组帧协议发送数据，数据的最后两个字节是 DLE 和 ETX。在紧邻 CRC 之前传送的字节序列是什么？
9. 对于下列每一种组帧协议，给出一个字节/比特序列，使它肯定不会出现在传输中。
   (a) BISYNC。
   (b) HDLC。
10. ★ 假设无论何时出现 1 比特，SONET 接收方就对它的时钟进行重同步；否则，接收方在它认为是该比特的时间片的中间位置对信号进行采样。
    (a) 为了能正确接收一行中的 48 个全 0 字节（一个 ATM 信元的值），要求发送方和接收方时钟的相对精确度是多少？
    (b) 考虑一个 SONET STS-1 线路上的转发站 A，它从下游端点 B 接收帧并将它们向上游重新传输。为了防止 A 每分钟积累多于一个的额外帧，要求 A 和 B 时钟的相对精确度是多少？
11. 说明为什么二维奇偶校验可以检测到所有的 3 比特错。
12. 给出一个 4 比特错的例子，使得如图 2-14 所示的二维奇偶校验无法检测。通常在哪些情况下检测不到 4 比特错？
13. 说明二维奇偶校验为接收方提供了足够的信息来纠正任意 1 比特错（假设接收方知道只有 1 比特是错的），但不能纠正任意 2 比特错。
14. 说明因特网校验和永远不会是 0xFFFF（即 sum 的最终值不会是 0x0000），除非缓冲区中的每一字节都是 0。（事实上，因特网规范要求校验和 0x0000 用 0xFFFF 传输，0x0000 值为省略的校验和保留。注意，在反码运算中，0x0000 和 0xFFFF 都表示 0 值。）
15. 证明除了在最后的校验和中的字节应该交换成正确的顺序外，书中的因特网校验和计算与字节顺序

(主机顺序或网络顺序)无关。特别是，以两种字节顺序中的任何一种计算16位字的校验和。例如16位字的反码和(表示为+')可表示为

$$[A, B] +'[C, D] +'\cdots +'[Y, Z]$$

下面交换后的和与上面的和一样：

$$[B, A] +'[D, C] +'\cdots +'[Z, Y]$$

16. 假设由因特网校验和算法使用的一个缓冲区中的一个字节需要减1(例如，首部的跳数字段)。给出一个算法，不需重新扫描整个缓冲区就能计算修正的校验和。你的算法应该考虑所讨论的字节是低字节还是高字节。

★ 17. 说明因特网校验和可以通过下列方式计算：首先对32位单元中的缓冲区求32位的反码和，然后对高半字和低半字求16位的反码和，最后同样对结果求反。(为了在32位的补码硬件上求32位的反码和，需要访问"溢出"位。)

18. 假设要传输消息11001001，并用CRC多项式x^3+1防止它出错。

(a) 使用多项式长除法确定应传输的消息。

(b) 假设由于传输链路上的噪声使得消息最左端的比特发生反转。接收方的CRC的计算结果是什么？接收方如何知道发生了一个差错？

√ 19. 假设要传输消息1011 0010 0100 1011，并用CRC多项式$x^5+x^2+x^1+1$防止它出错。

(a) 使用多项式长除法确定应传输的消息。

(b) 假设由于传输链路上的噪声使得消息最左端的比特发生反转。接收方CRC的计算结果是什么？接收方如何知道发生了一个差错？

20. 本章提出的CRC算法需要许多比特操作。然而，通过一种表驱动方法，可能一次取多个比特进行多项式长除法，这使得CRC的软件实现很有效。这里概述一次取3比特进行长除法的策略(见表2-5)；实际中，我们一次除8比特，表中将有256个条目。

令除数多项式$C=C(x)$为x^3+x^2+1，或1101。为了给C建立一个表，取每个3比特序列p，在其末尾附加3个0，然后求商$q=p\hat{}000\div C$，忽略余数。第3列是乘积$C\times q$，它的前3个比特应该等于p。

表2-5 表驱动计算CRC

p	$q=p\hat{}000\div C$	$C\times q$
000	000	000 000
001	001	001 101
010	011	010 ···
011	0··	011 ···
100	111	100 011
101	110	101 110
110	100	110 ···
111	···	111 ···

(a) 对于$p=110$，验证商$q=p\hat{}000\div C$和$q=p\hat{}111\div C$是相同的；就是说，末端比特是什么无关紧要。

(b) 填充表中缺少的条目。

(c) 利用该表，用C去除101 001 011 001 100。提示：被除数的前3比特是$p=101$，所以从表中看出，相应的商的前3比特是$q=110$。在被除数的第二个3比特上写下110，然后再从表中将被除数的前6比特减去$C\times q=101\ 110$。3比特一组继续进行。应该没有余数。

★ 21. 用1个奇偶校验位能够检测所有的1比特错。说明如下所示的推广至少有一种无效：

(a) 说明如果消息m的长度是8比特，那么不存在2比特的差错检测码$e=e(m)$能够检测所有2比特错。提示：考虑只有1位是1的所有8比特消息的集合M；注意，来自M的任何消息能够变成任何其他含2比特错的消息，并说明M中的某两个消息m_1和m_2一定有相同的差错码e。

(b) 找到一个N(不必是最小的)，使得应用于N比特块的32比特差错检测码不能够检测出改变到最多8比特的所有差错。

22. 考虑一个仅使用否定确认帧(NAK)但是没有肯定确认帧(ACK)的ARQ协议。描述需要如何安排超时。解释为什么一个基于ACK的协议通常比基于NAK的协议更可取。

23. 考虑在一条40km的点到点光纤链路上运行的ARQ算法。

(a) 计算链路的传播延迟，假设光纤中的光速是 2×10^8 m/s。

(b) 提出一个用于 ARQ 算法的合适的超时值。

(c) 给定这个超时值之后，为什么 ARQ 算法仍可能超时并重传帧？

24. 假设你为到月球的一条 1Mbps 点到点链路设计一个滑动窗口协议，单程时延是 1.25s，假设每帧携带 1 KB 数据，最少需要多少比特作序号？

25. 假设你为卫星站的一条 1Mbps 点到点链路设计一个滑动窗口协议，卫星在 3×10^4 km 的高度绕地球旋转。假设每帧携带 1KB 数据，在下述情况下，最少需要多少比特作序号？假设光速为 3×10^8 m/s。

(a) RWS=1。

(b) RWS=SWS。

26. 本章提出的滑动窗口协议可用于实现流量控制。这样做时我们可以想象接收方延迟几个 ACK，即直到有一个空闲的缓冲区存放下一帧才发送 ACK。这样，每一个 ACK 将同时确认收到最后一帧并告诉源方现在存在可用于保存下一帧的空闲缓冲区。解释为什么以这种方式实现流量控制不是一个好主意。

27. 图 2-17 的停止-等待图示中隐含的含义是，接收方收到重复的数据帧后就立即重发 ACK。假设改为接收方拥有自己的计时器，只有当期待的下一帧在超时间隔内没有到达后它才重传 ACK。假设接收方的超时值是发送方的 2 倍，画出说明图 2-17b～图 2-17d 中情形的时间线。假设接收方的超时值是发送方的一半，重画图 2-17c 的时间线。

28. 在停止-等待传输中，假设收到重复的 ACK 或数据帧后，发送方和接收方都立即重传上一帧。这一策略表面上是合理的，因为收到这种重复的 ACK 很可能意味着另一方超时了。

(a) 画一条时间线，说明如果第一个数据帧不知何故重复了，但没有帧丢失时会发生什么情况。重复过程将持续多久？这种情况称为“魔术师的学徒错误”。

(b) 假设像数据一样，如果在超时期间内没有响应，ACK 也被重传。同时假设双方使用相同的超时间隔。找出一种触发“魔术师的学徒错误”的合理情况。

29. 给出当接收方用完缓冲区空间时，如何通过让 ACK 携带额外的信息以减小 SWS 来扩充带有流量控制的滑动窗口协议的一些细节。假设初始 SWS 和 RWS 都是 4，链路速度是瞬时的，并且接收方能够以每秒一个的速率释放缓冲区（即接收方是瓶颈）。用一条传输时间线说明这个协议。说明在 $T=0$s，$T=1$s，…，$T=4$s 时会发生什么。

30. 描述一个将滑动窗口算法和可选 ACK 相结合的协议。该协议应该能及时重传，但如果一帧到达时只有一个或两个位置次序颠倒，则不重传。该协议也应该明确指出，如果几个连续的帧丢失了会发生什么。

31. 对于下列两种情况，画出 SWS=RWS=3 帧的滑动窗口算法的时间线图。使用的超时间隔大约为 2×RTT。

(a) 帧 4 丢失。

(b) 帧 4～帧 6 丢失。

32. 对于下列两种情况，画出 SWS=RWS=4 帧的滑动窗口算法的时间线图。假设接收方在未收到期望的帧时发送一个重复确认帧。例如，当它希望看到 FRAME [2] 却收到 FRAME [3] 时，便发送 DUPACK [2]。当接收方收到一系列帧时也发送一个累积的确认。例如，当它在收到 FRAME [3]，FRAME [4] 和 FRAME [5] 之后又收到丢失的 FRAME [2]，便发送 ACK [5]。使用的超时间隔大约为 2×RTT。

(a) 帧 2 丢失，超时之后重传（如通常一样）。

(b) 帧 2 丢失，在收到第一个重复确认帧或超时之后重传。这种方法能减少处理时间吗？注意为了快速重传，某些端到端协议（如 TCP 的变种）使用类似的方法。

33. 假设要运行 SWS=RWS=3 和 MaxSeqNum=5 的滑动窗口算法。因而，第 N 个分组 DATA [N] 在它的序号字段中实际包含 N mod 5。给出算法造成错乱的一个例子，即接收方期待 DATA [5] 却收

到 DATA [0]（它们具有相同的发送序号）。没有以错序到达的分组。注意，这意味着 MaxSeqNum ≥6 是充分必要条件。

34. 考虑 SWS=RWS=3 的滑动窗口算法，没有错序到达，并具有无限精度的序号。
    (a) 说明如果 DATA [6] 在接收窗口内，那么 DATA [0]（或一般而言，任意较早的数据）不可能到达接收方（因此，MaxSeqNum=6 应该是充分的）。
    (b) 说明如果能够发送 ACK [6]（或更精确地说，DATA [5] 在发送窗口内），那么收不到 ACK[2]（或更早的数据）。
    这些是对 2.5.2 节中给出的公式的一个证明，是对 SWS=3 情况的详述。注意，(b) 隐含说明前一个问题不能反过来包含无法区分 ACK [0] 和 ACK [5] 的情况。

35. 假设运行 SWS=5 和 RWS=3 的滑动窗口算法，并且没有错序到达。
    (a) 求 MaxSeqNum 的最小值。可以假设下述条件对于求最小的 MaxSeqNum 是充分的，如果 DATA [MaxSeqNum] 在滑动窗口内，那么 DATA [0] 不会到达。
    (b) 给出一个例子，说明 MaxSeqNum-1 不是充分的。
    (c) 给出最小化 MaxSeqNum 的通用规则，用 SWS 和 RWS 表示。

36. 假设通过一台中间路由器 R 将 A 连接到 B，如图 2-37 所示。A-R 链路和 R-B 链路在每个方向上每秒只接收并传输一个分组（因此两个分组占用 2s），并且两个方向独立传输。假设 A 使用 SWS=4 的滑动窗口协议向 B 发送。
    (a) 对于时间 $T=0, 1, 2, 3, 4, 5$，指出到达并离开每个节点的各个分组，或将它们标记在一条时间线上。
    (b) 如果链路上有一个 1.0s 的传输延迟，但立即接收到达的所有分组（即时延=1s 而带宽无限），那么会发生什么？

37. 假设通过一台中间路由器 R 将 A 连接到 B，如上题。A-R 链路是瞬时的，但 R-B 链路每秒只传输一个分组，一次一个（因此两个分组占用 2s）。假设 A 使用 SWS=4 的滑动窗口协议向 B 发送。对于时间 $T=0, 1, 2, 3, 4$，说明哪些分组到达 A 和 B 并从 A 和 B 发出。R 上的队列会变得多长？

38. 考虑上题中的情况，这次假设路由器队列长度为 1，即除了正在发送的分组外，它还能保存一个分组（在每个方向上）。令 A 的超时是 5s，SWS=4。说明从 $T=0$ 直到第一个满窗的所有 4 个分组都被成功传输的每一秒当中会发生什么。

39. 当在以太网上的两个主机共享一台硬件地址时会出现什么类型的问题？描述会发生什么情况以及为什么这些情况会成为问题。

40. 1982 年的以太网规范允许任意两个站点之间的同轴电缆最长可达 1 500m，其他点到点链路电缆长 1 000m，并有两个中继器。每个站点或中继器通过最长 50m 的“分接电缆”连接到同轴电缆。表 2-6 给出与每个设备相关的典型延迟（其中 c=真空中的光速=3×10^8 m/s）。由列出的设备源引起的以比特度量的最坏往返传播延迟是多少？（列表并不完全，其他延迟源包括检测时间和信号上升时间。）

表 2-6 各种设备相关的典型延迟 [习题 40]

项 目	延 迟
同轴电缆	传播速度 0.77c
链路/分支电缆	传播速度 0.65c
中继器	每个大约 0.6μs
收发器	每个大约 0.2μs

★ 41. 在同轴电缆以太网上，两个中继器之间的最大距离被限制在 500m，中继器重新产生 100%原振幅的信号。沿着一个 500m 的网段，信号将衰减到不低于初值的 14%（8.5dB）。那么，沿着 1 500m 的网段，衰减可能为 $0.14^3=0.3\%$。即使沿着 2 500m 的网段，这样一个信号仍然是可读的。那么，为什么要求每 500m 放置一个中继器？

42. 假设以太网的往返传播延迟是 46.4μs。这就会产生一个 512 位的最小分组尺寸（464 位传播延迟+48 位干扰信号）。
    (a) 如果延迟时间保持不变，并且发信号的速率增长到 100Mbps，那么最小分组尺寸会发生什么？

(b) 这么大的最小分组尺寸的缺点是什么?

(c) 如果兼容性不是问题，那么为了允许更小的最小分组尺寸，应该怎样制定规范?

★ 43. A 和 B 是试图在以太网上传输的两个站。每个站有一个准备发送的帧的稳定队列，A 的帧被编号为 A_1，A_2，等等，B 的帧类似。令 $T=51.2\mu s$ 是指数退避的基本单元。

假设 A 和 B 同时想发送帧 1，导致冲突，并分别选择退避时间 $0\times T$ 和 $1\times T$，这意味着 A 在竞争中获胜并传输 A_1 而 B 等待。在这次传输结束时，B 将试图重传 B_1 而 A 试图传输 A_2。这种首次尝试又会冲突，但现在 A 退避 $0\times T$ 或 $1\times T$，而 B 退避的时间等于 $0\times T$，…，$3\times T$ 中之一。

(a) 给出第一次冲突后 A 立即在第二次退避竞争中获胜的概率，就是说，A 第一次选择的退避时间 $k\times 51.2\mu s$ 小于 B 的退避时间。

(b) 假设 A 在第二次退避竞争中获胜，A 传输 A_2，当传输结束时，在 A 试图传输 A_3 而 B 试图再一次传输 B_1 时，A 和 B 又发生冲突。给出第一次冲突后 A 立即在第三次退避竞争中获胜的概率。

(c) 为 A 在所有余下的退避竞争中获胜的概率给出一个合理的下界。

(d) 然后帧 B_1 发生了什么?

这种情形称为以太网的捕获效用 (capture effect)。

44. 假设按如下方式修改以太网的传输算法：在每个成功传输完成后，主机等待一或两个时间片之后再尝试传输，否则采用常用方式退避。

(a) 解释为什么上题的捕获效用现在变得如此小。

(b) 说明上述策略现在如何导致一对主机捕获以太网，交替传输，并将第三台主机拒之门外。

(c) 提出一个可供选择的方法，例如，通过修改指数退避算法。一个站点的历史记录的哪些方面可被用作所修改的退避的参数?

45. 以太网使用曼彻斯特编码。假设共享以太网的主机没有完全同步，为什么在没有等到分组最后的 CRC 时就可以检测出冲突?

46. 假设 A、B、C 都进行第一次载波监听，作为传输尝试的一部分，而第 4 个站 D 正在传输。画出一条时间线，说明一个可能的传输、尝试、冲突和指数退避选择的序列。这条时间线也应该满足下列准则：(1) 初始传输尝试应该按 A、B、C 的顺序，而成功的传输应该按 C、B、A 的顺序；(2) 至少应该有 4 个冲突。

47. 重复前一道习题，现在假设以太网是 p-坚持的，$p=0.33$ (即线路成为空闲时，等待站立即用概率 p 传输，否则推迟一个 $51.2\mu s$ 时间片并重复该过程)。这条时间线应该满足准则：①初始传输尝试应该按 A、B、C 的顺序，而成功的传输应该按 C、B、A 的顺序；②说明在一条空闲线路上 4 个延迟时间片内至少有一个冲突，并且至少有一次发送成功。此外要注意，可能有许多解决方案。

48. 假设以太网物理地址是随机选择的 (使用真随机比特)。

(a) 在一个有 1 024 台主机的网络上，两个地址相同的概率是多少?

(b) 上述事件发生在 2^{20} 个网络的某一个或某几个上的概率是多少?

(c) (b) 中的所有网络中 2^{20} 台主机的某一对有相同地址的概率是多少?

提示：(a) 和 (c) 的解法就是用于解决所谓的生日问题的另一种形式：给定 N 个人，其中两个人的生日 (地址) 相同的概率是多少? 第二个人与第一个人的生日不同的概率是 $1-1/365$，第三个人与前两个人的生日不同的概率是 $1-2/365$，依此类推。因此，所有的生日不同的概率是

$$\left(1-\frac{1}{365}\right)\times\left(1-\frac{2}{365}\right)\times\cdots\times\left(1-\frac{N-1}{365}\right)$$

对于较小的 N 大约为

$$1-\frac{1+2+\cdots+(N-1)}{365}$$

49. 假设一个以太网上有 5 个站点在等待另外的分组传输结束。只要分组传输完毕，5 个站点就立刻开始传输，并导致冲突。

(a) 模拟这种情况，直到这 5 个等待站点之一成功传输。使用掷硬币或某种其他真实的随机源来确定退避时间。进行下列简化：忽略帧之间的间距，忽略冲突时间的可变性（因此，重传总是在 51.2μs 时间片的一个整数倍的时间后进行），并且假设每个冲突恰好用完一个时间片。

(b) 讨论模拟中的简化效果，与你可能在一个真实的以太网上遇到的行为作对比。

50. 写一个程序来实现上面讨论的模拟，这次有 N 个站点在等待传输。此外，用整数 T 模拟时间，单位是时间片，并且认为冲突占用一个时间片（因此，对于跟着 $k=0$ 的退避，其 T 时刻的冲突将导致 $T+1$ 时刻的重传尝试）。对于 $N=20$、$N=40$ 和 $N=100$，找出一个站点成功传输之前的平均延迟。你的数据支持"在 N 中延迟是线性的"这一概念吗？提示：对于每个站点，记录站的 NextTimeToSend 和 CollisionCount。到达时刻 T 时，只有一个站点的 NextTimeToSend $==T$，那么你的工作就完成了。如果没有这样的站点，T 增加 1。如果有两个或更多的站点，调度重传然后再试。

51. 假设 N 个以太网站同时试图发送，需要 $N/2$ 个时间片来确定下次由哪个站点传输。假设平均分组大小是 5 个时间片，将有效带宽表示成 N 的函数。

52. 考虑下面的以太网模型。传输尝试以 1 个时间片的平均时间间隔随机进行。特别地，连续的尝试之间的时间间隔是一个指数随机变量 $x=-\lambda\log u$，其中 u 是在间隔 $0\leqslant u\leqslant 1$ 中随机选择的。如果从 $t-1$ 到 $t+1$ 的范围内有另一个传输尝试，那么时刻 t 的尝试导致冲突，其中 t 是以 51.2μs 时间片为单位来度量的；否则该尝试成功。

(a) 对于一个给定的 λ 值，写一个程序，模拟一个成功传输之前所需的时间片的平均数量，称为争用间隔（contention interval）。找出争用间隔的最小值。注意，必须找出一个在一次成功传输之后的传输尝试，以便确定是否存在冲突。忽略重传，因为它可能不符合上面的随机模型。

(b) 以太网在争用间隔和成功传输之间交替。假设成功传输平均持续 8 个时间片（512 字节）。利用上面的争用间隔的最小长度，那么理论上的 10Mbps 带宽中有多少可用作传输？

53. 当两个无线节点的距离间隔大于其传播可达区域时，这两个节点如何才能通信？

54. 为什么在无线网络中做冲突检测要比在有线网络（如以太网）中复杂？

55. 在 802.11 网络中如何检测隐藏终端？

56. 为什么发生自然灾难时，通信中的无线网格拓扑优于基站拓扑？

57. 为什么对于传感器网络的每个节点利用 GPS 获取位置的方法不可行？请提供一个实用的方法。

网络互联

自然界似乎是通过多条漫长而曲折的道路才能到达其诸多终点。

——鲁道夫·洛策

问题：并不是所有网络都是直接相连的

在前面的章节中，我们看到了如何将一个节点连接到另一个节点或现有网络。许多技术可以用于构建“最后一英里”的链路或连接数量适中的节点，但如何建立全局规模的网络呢？单个的以太网可以互联不超过 1 024 台主机，而点对点链接只能连接两台主机。无线网络受限于波段可达的范围。为了构建全局网络，我们需要找到一种连接不同类型的互联链路和网络的方法。这种将不同类型的网络互联起来构建一个大型的全球性网络的概念是因特网的核心概念，通常称为网络互联（internetworking）。

我们可以将网络互联问题分解为一些子问题。首先，我们需要一种互联的途径。互联同种类型链路的设备通常被命名为交换机（switch），本章首先讨论这些设备。当前，交换机中最重要的一类是那种用来连接以太网段的交换机，这些交换机有时也称为网桥（bridge）。交换机的核心工作是使分组到达输入端并转发（forward）或交换（switch）到合适的输出端，使它们到达适当的目的地。交换机有很多种方法为一个分组确定“正确”的输出，这些方法大致可以分为无连接和面向连接两类。多年来，这两种方法都有各自的重要应用领域。

虽然已知各种不同的网络类型，但我们仍然需要一种途径来互联不同的网络和链接（如异构性处理）。曾被称为网关（gateway）的设备，现在更多地被称为路由器（router），用于处理不同类型的网络互联的协议，即网际协议（IP），是 3.2 节的主题。

一旦我们用交换机和路由器互联了所有的链接和网络，就可能获得很多不同的一个点到其他点的途径。通过网络找到合适的途径或路由（route）也成为网络互联的基础问题。例如，路径应该是有效的（比如最短）、无回环且可以处理非静态网络问题。出现这些问题是因为节点可能出错或重启，链接可能断裂，以及需要增添新节点或链接等。3.3 节将展望一些已有的处理此类问题的算法和协议。

一旦我们理解了交换和路由面临的问题，我们就需要一些设备来执行这些功能。本章最后总结了一些关于建立交换和路由的讨论。虽然许多分组交换机和路由器与通用计算机非常类似，但也有很多专门的设计。这一点在高端应用的情况下尤为突出，此时互联网的核心似乎永无止境地需要更大、更快的路由器来处理日益增加的传输负荷。

3.1 交换和桥接[1]

简单地说，交换是一种允许我们互联链路以形成一个更大规模网络的机制。交换机是

[1] 参见“实践三”。

一个多输入、多输出的设备，它能由一个输入端口传送分组到一个或多个输出端口。交换机把一个星形拓扑结构（见图 3-1）加到第 2 章中建立的点到点链路、总线（以太网）和环形拓扑结构上。星形拓扑结构具有以下优点：

- 尽管交换机有固定数目的输入端口和输出端口，限制了能直接连到单台交换机上的主机数，但是我们可以通过互联多台交换机来建立一个更大的网络。
- 我们可以通过点到点的链路把交换机互联，并把主机连接到交换机上。通常情况下，这意味着我们可以建立一个地理范围很大的网络。
- 在将一台新主机连到交换机上使之加入网络时，并不意味着已和网络相连的主机的性能会变差。

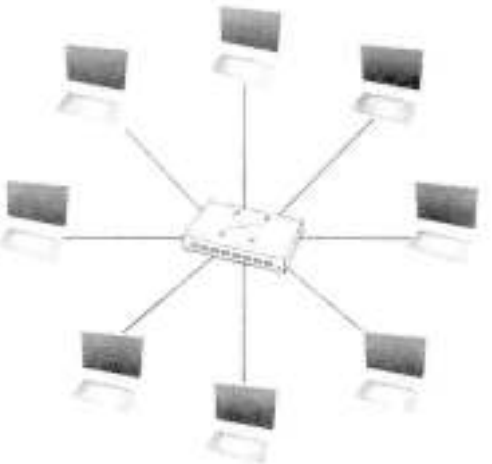


图 3-1 由交换机构成的星形拓扑

最后一点不适用于第 2 章中讨论的共享介质网络。例如，位于同一个以太网上的两台主机继续以 10Mbps 传输数据是不可能的，因为它们共享相同的传输介质。交换网中每台主机都有一条到交换机的链路，所以许多主机以全链路速度（带宽）传输数据是完全可能的，只要交换机的设计具有足够的总容量。交换机的设计目标之一就是提供高吞吐量，我们在下面再讨论这个问题。总之，交换网比共享介质网更具可扩展性（scalable），即具备增加更多节点的能力，因为交换网支持更多主机以全链路速度传输数据。

交换机被连接到许多链路上，为了与链路另一端的节点进行通信，每一条链路都运行相应的数据链路协议。交换机的主要工作就是在它的一条链路上接收输入分组，再把这些分组从其他的链路上传输出去。采用 OSI 体系结构中的术语，这种功能有时也称为交换（switching）或转发（forwarding），这是网络层的主要功能。

问题是交换机如何决定把每个分组放到哪一个输出链路上呢？一般的解决方法是交换机查看分组首部中的标识符，通过这个标识符来做出决定。使用这个标识符的方法多种多样，但是有两种常用的方法。第一种是数据报（datagram）或称无连接（connectionless）的方法，第二种是虚电路（virtual circuit）或称面向连接（connection）的方法。第三种方法不如前两种常用，称为源路由（source routing），但它最容易解释并且确实有一些有用的应用。

所有网络都需要采用一种方法来标识端节点，这种标识符通常称为地址（address）。在第 2 章中，我们已经见过几个地址的实例，例如以太网中使用的 48 位地址。对于以太网中的地址，唯一的要求就是网络中不存在具有相同地址的两个节点。这通过确保所有的以太网卡都有一个全局唯一（globally unique）标识符来实现。在下面的讨论中，我们假设每一台主机都有一个全局唯一的地址。以后，我们会考虑地址的其他有用特性，但作为讨论的起点，地址的全局唯一性就足够了。

我们需要做的另外一个假设就是有一些方法可用来标识每台交换机的输入和输出端口。至少有两种实用的端口标识方法：一种是给每个端口进行编号，另一种是用输入和输出端口所连接的节点（交换机或主机）的名字标识来识别端口。现在，我们使用给端口编号的方法。

3.1.1 数据报

数据报的思想非常简单：只需确保每个分组带有足够的信息，使得任何一台交换机都能决定怎样使它到达目的地。这就是说，每个分组都带有完整的目的地址。考查图 3-2 中的例子。在这个例子中，主机的地址为 A、B、C 等，要决定怎样转发一个分组，交换机需要查阅转发表（forwarding table）（有时称为路由表（routing table））。表 3-1 给出路由表的一个例子。这张特定的表显示交换机 2 在转发上例中的数据报时所需的转发信息。当你对这里描述的简单网络有了完全的了解后，就很容易画出这样的一张表，我们可以想象这张表是由网络管理员静态做出的。但是，对于有动态变化的拓扑结构和有多条路径可到达目的地的大型复杂网络，建立转发表就变得很困难。这个更难的问题称为路由（routing），是 3.3 节讨论的主题。我们可以把路由选择看作是在这种背景下发生的过程，当数据分组到达时，在转发表中能够获得正确信息来转发或交换分组。

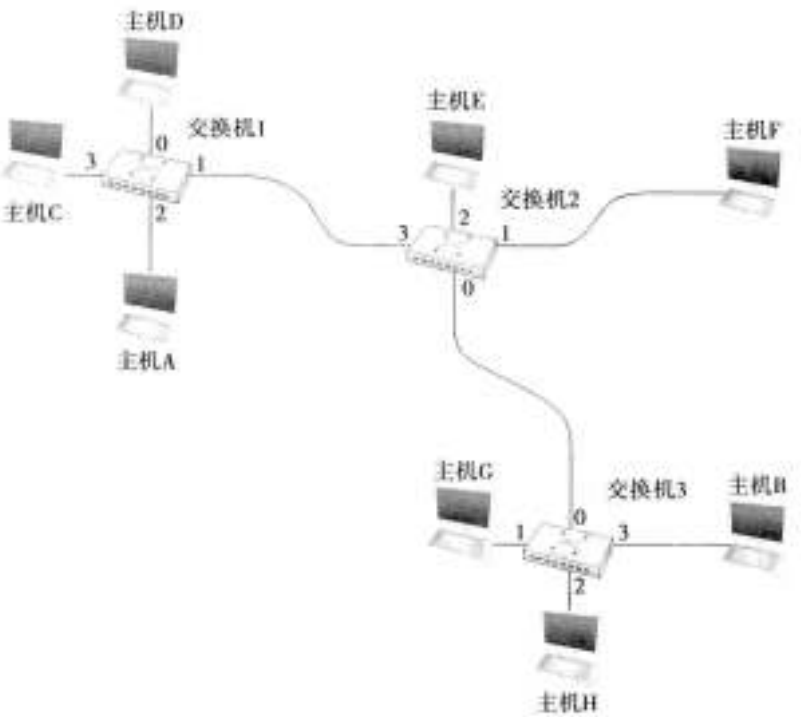


图 3-2 数据报转发的网络示例

数据报网络有以下特点：

- 一台主机无论何时都可以发送分组，因为任何到达交换机的分组都能立即转发（假设正确建立了转发表）。正因为如此，数据报网络也称为*无连接的*（connectionless）。这和*面向连接*（connection-oriented）的网络不同，因为面向连接网络在发送第一个数据分组之前需要建立某种*连接状态*（connection state）。
- 当一台主机发送一个分组时，主机无法知道网络是否可以传送该分组或目的主机是否可以接收。

表 3-1 交换机 2 的转发表

目的地	端 口
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

- 每个分组的转发都是独立于前面的分组的，哪怕这几个分组有可能传送到相同的目的地。这样，从主机 A 到主机 B 的两个连续的分组可能沿着完全不同的路径（也许是由于网络中某台交换机更改了转发表）进行传送。
- 当一台交换机或一段链路出现故障时，如果有可能在故障点周围找到一条可替代的路径，并相应地更新转发表，那么对通信并不会产生任何严重的影响。

最后一点对于数据报网络的历史发展尤其重要。因特网最重要的设计目标之一是具有应对错误的健壮性。历史证明数据报有效地达到了这个目标[⊖]。

3.1.2 虚电路交换

第二种明显不同于数据报模式的分组交换技术采用了*虚电路*（Virtual Circuit，VC）的概念。这种方法也称为*面向连接模式*（connection-oriented model），它要求在发送数据之前首先在源主机和目的主机之间建立一条虚连接。为了理解它的工作原理，考虑图 3-3，图中主机 A 想把分组发送到主机 B。我们可以把它看作一个两阶段的处理过程，第一个阶段是“建立连接”，第二个阶段是传输数据。我们分别予以考虑。



图 3-3 虚电路网络的例子

在建立连接阶段，需要在源主机和目的主机之间的每一台交换机上建立“连接状态”。连接状态由连接经过的每台交换机中的“VC 表”记录组成。交换机上的 VC 表中的一条记录包括：

- *虚电路标识符*（Virtual Circuit Identifier，VCI），在这台交换机上唯一标识连接，并且将放在属于这个连接的分组首部内传送。
- 从这个 VC 到达交换机的分组的输入接口。
- 从这个 VC 离开交换机的分组的输出接口。
- 用于输出分组的一个可能不同的 VCI。

这样一条记录的语义如下：如果分组在指定的输入接口到达并且首部包含指定的 VCI 值，那么先将这个分组的首部 VCI 替换成指定的输出 VCI 并将分组发送到指定的输出端口。

注意，交换机收到的分组的 VCI 和收到分组的接口的组合唯一标识这个虚连接。当然，在一台交换机上可能会同时建立许多虚连接。我们可以观察到，输入的 VCI 值和输

⊖ 再次声明：ARPANET 是为对抗核打击而构造的这一说法虽然没有得到其实际设计人员的证实，但应对各个构件错误的健壮性的确是其目标之一。

出的 VCI 值一般不同。所以，对于一个连接来讲，VCI 不是全局意义的标识符，而只是在一段给定链路上有意义，即它具有链路局部范围（link-local scope）。

当要建立一个新连接时，在连接所要经过的每段链路上分配一个新的 VCI 值。我们也要确保在一段链路上选定的 VCI 的值未被该链路上已存在的某个连接使用。

建立连接状态有两大类方法。一类是由网络管理员配置连接状态，这样的虚电路是"永久的"。自然，管理员也可以删除它，因此永久虚电路（Permanent Virtual Circuit，PVC）最好看作长期生存的或可管理配置的 VC。另一类是主机发送消息给网络以建立连接状态。发送的消息称为信令（signalling），这样建立的虚电路称为是交换的（switched）。一个交换的虚电路（SVC）的突出特性是主机可以动态地建立和删除这个虚电路，而不需要网络管理员的参与。注意，SVC 应该更准确地称作信令（signalled）虚电路，因为它使用信令（而不是交换）的方式，这正是 SVC 与 PVC 的区别。

我们假设网络管理员想手工创建一条从主机 A 到主机 B 的新的虚连接。首先，管理员要标识一个从 A 到 B 经过网络的路径，在图 3-3 的网络例子中只有一条路径，但在一般情况下可能不是这样。然后管理员在连接的每段链路上选择一个当前没有使用的 VCI 值。为达到例子的目的，假设从 A 到交换机 1 链路的 VCI 值选为 5，从交换机 1 到交换机 2 链路的 VCI 值选为 11。在这种情况下，交换机 1 在 VC 表中需要有一条配置成如表 3-2 中所示的记录。

表 3-2 交换机 1 的虚电路表条目

输入接口	输入 VCI	输出接口	输出 VCI
2	5	1	11

类似地，假设标识从交换机 2 到交换机 3 链路连接的 VCI 值选为 7，从交换机 3 到主机 B 链路的 VCI 值选为 4。这样交换机 2 和交换机 3 需要配置的 VC 表记录如表 3-3 所示。注意一台交换机的输出 VCI 值就是下一台交换机的输入 VCI 值。

一旦 VC 表建立，就可以进入数据传输阶段，如图 3-4 所示。对于每一个欲发送到主机 B 的分组，主机 A 将值为 5 的 VCI 放入分组首部并发送到交换机 1。交换机 1 在接口 2 上接收每个这样的分组，它使用这个接口号和分组首部中 VCI 的组合来找到相应的 VC 表记录。如表 3-2 所示，本例的表记录指示交换机 1 用接口 1 发送分组，并在发送分组时将分组首部的 VCI 赋值 11。这样，这个分组携带值为 11 的 VCI 从接口 3 到达交换机 2。交换机 2 在它的 VC 表中查找接口 3 和 VCI 值为 11 的记录（如表 3-3 所示），将分组首部的 VCI 值作相应的更改后发到交换机 3，如图 3-5 所示。此过程继续，直到分组携带值为 4 的 VCI 到达主机 B。主机 B 由此识别这个分组来自主机 A。

表 3-3 交换机 2 和交换机 3 的虚电路表条目

交换机 2 的 VC 表条目			
输入接口	输入 VCI	输出接口	输出 VCI
3	11	2	7
交换机 3 的 VC 表条目			
输入接口	输入 VCI	输出接口	输出 VCI
0	7	1	4

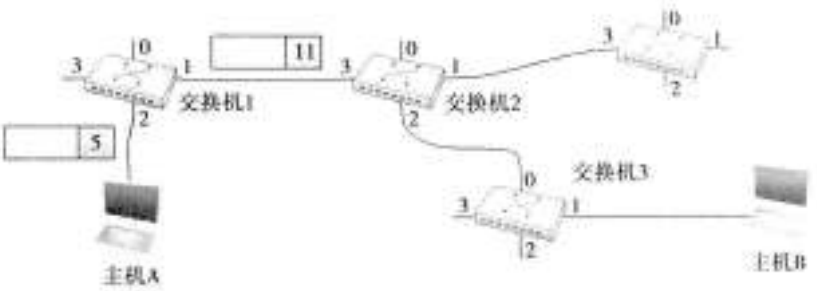


图 3-4 分组被发送到虚电路网络

在具有相当规模的真实网络中，使用上述过程在大量交换机中正确配置 VC 表，很快就会不堪重负。因此，我们几乎总是使用网络管理工具或某种信令方法，即使建立“永久”的 VC 也是如此。在 PVC 的情况下，信令由管理员发起，而 SVC 通常是使用其中一台主机的信令建立的。现在，我们考虑怎样用来自主机的信令建立刚才描述的 VC。

为了开始发信令的过程，主机 A 发送一个建立消息给网络，即发给交换机 1。这个建立消息中包含主机 B 的完整的目的地址。建立消息需要获得到达 B 的整个路径，以此建立路径上每台交换机中必需的连接状态。我们可以看到传送建立消息给主机 B 很像传送数据报给主机 B，其中交换机必须知道用哪个输出端口发送建立消息才能使它最终到达 B。现在，我们假设交换机知道足够的网络拓扑结构，使得建立消息在最终到达 B 之前依次经过交换机 2 和交换机 3。

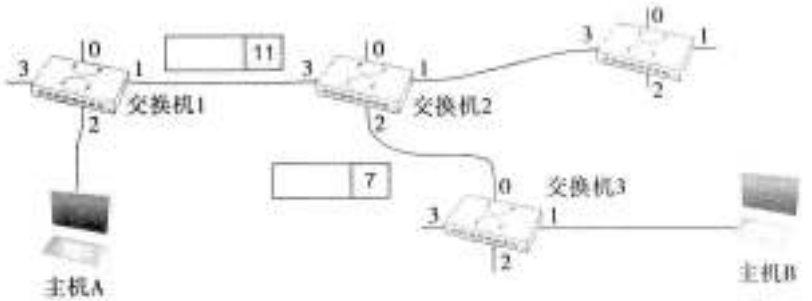


图 3-5 分组穿越虚电路网络

交换机 1 接收到连接请求时，除了将此请求发送给交换机 2 之外，还为这个新连接在它的虚电路表中创建一个新的记录。这个记录和前面表 3-2 所示的一样。主要区别是在接口上赋值一个未用 VCI 值的工作是由交换机完成的。在本例中，交换机取值为 5。当前虚电路表含有以下信息：“当到达端口 2 的分组标识符为 5 时，从端口 1 发送它们。”另一个问题在于，主机 A 需要知道将值为 5 的 VCI 放入欲发送给 B 的分组中。下面，我们将看到这个过程是怎样发生的。

在交换机 2 接收到建立消息后，它执行一个类似的过程，本例中选择 11 作为输入的 VCI 值，同样，交换机 3 选择 7 作为输入 VCI 的值。每台交换机可以给 VCI 取任意值，只要这个值是交换机端口上的其他连接不使用的。如前面提到的，VCI 具有“链路局部范围，就是说它们没有全局意义。

最后，建立消息到达主机 B。假设主机 B 正常且愿意与主机 A 建立连接，它也分配一个输入 VCI 值，本例中是 4。主机 B 用这个值识别所有来自主机 A 的分组。

现在，为了完成连接，每一个节点都需要知道它的下游相邻节点为此连接使用的 VCI 值。主机 B 给交换机 3 发送一个连接建立的确认消息，此消息包含它选取的 VCI 值（4）。现在交换机 3 可以完成此连接的虚电路表记录，因为它知道输出 VCI 的值必定是 4。交换机 3 发送确认消息给交换机 2，并指定 VCI 值为 7。交换机 2 发送消息给交换机 1，指定 VCI 值为 11。最后，交换机 1 将确认消息传递给主机 A，通知它这个连接使用的 VCI 值为 5。

至此，每个节点都知道从主机 A 到主机 B 通信的所有必需信息，每台交换机内都有与连接有关的完整的虚电路表记录。而且，主机 A 得到了肯定的确认，知道到达主机 B 的整条路径上的每件事都已准备好。这时，连接表记录已经存放在所有三台交换机内，正像上述例子中管理员配置的那样。但响应主机 A 发出的信令消息的整个过程是自动发生的。现在数据传输阶段可以开始了，而且和在 PVC 情况下的用法相同。

当主机 A 不想再发送数据给主机 B 时，它通过给交换机 1 发送一个撤销连接的消息来撤销连接。交换机 1 从虚电路表中删除相关记录，并把撤销信息转发给路径上的其他交换机，下一台交换机同样也删去表中相应记录。如果现在主机 A 再发送一个 VCI 为 5 的分组到交换机 1，这个分组就会被丢弃，就像连接根本不存在一样。

对于虚电路交换还要说明几点：

- 由于主机 A 在发送第一个数据分组之前，必须等待连接请求到达网络最远端并返回。所以在发送数据前至少有一个 RTT 的延迟[⊖]。
- 虽然连接请求包含主机 B 的完整地址（作为网络的全局标识符，这个地址也许会非常大），但每一个数据分组仅带有一个很小的标识符，这个标识符只在一段链路上是唯一的。这样，相对于数据报模式，分组首部引起的开销就会减少。
- 如果一个连接上有一台交换机或有一条链路出现故障，连接就会被破坏。这样就需要建立一个新的连接。同时需要撤销原来的连接，释放交换机中虚电路表的存储空间。
- 我们掩盖了一台交换机怎样决定在哪一条链路上传输连接请求的问题。其实这和数据报转发时建立转发表是同一个问题，需要某种路由算法（routing algorithm）。路由选择在 3.3 节描述，其中提到的算法对于路由建立请求和数据报是可通用的。

虚电路的优点之一是主机在发送数据前知道许多网络信息。例如，主机知道真正存在一条到接收方的路径，接收方愿意也能够接收数据，并且在建立虚电路时也可以把资源分配给虚电路。例如，X.25 网络是早期的（现在几乎废弃）一种基于虚电路的网络技术。X.25 网络采用的策略有以下三部分：

1）在初始化虚电路时，把缓存区分配给每一个虚电路。

2）在虚电路的每对节点之间使用滑动窗口协议（见 2.5 节）来进行流量控制，该协议是为了防止发送节点使接收节点分配的缓冲区溢出。

3）当连接请求消息传送到没有足够的缓冲区的某个节点时，建立虚电路的请求被拒绝。

在实施这三项策略的过程中，每个节点都应保证有足够的缓冲区，以供到达电路的分

⊖ 这并不是完全真实的。一些人提出的“高效”方法为，在收到一个连接请求时马上发送一个数据分组。然而，很多当前的系统实现是在发送数据之前先建立连接。

组排队所需。这种基本的策略称为逐跳流量控制（hop-by-hop flow control）。

比较起来，数据报网络没有建立连接阶段，且每台交换机独立处理每个分组，这使得数据报网络如何用有效方法分配资源的问题得以淡化。而每个到达的分组与其他分组竞争缓冲区空间的问题却突显出来。如果没有空闲的缓冲区，接收的分组必须被丢弃。但是我们注意到，即使在基于数据报的网络中，源主机也经常发送一组分组到同一目的主机。每台交换机可以根据源节点/目标节点对区分已排队的分组，这样交换机可以保证属于每个源节点/目标节点对的分组公平地共享缓冲区。我们在第6章对此做更深入的讨论。

在虚电路模式中，我们可以设想给每个虚电路提供不同的服务质量（Quality of Service，QoS）。在这种背景下，服务质量这个术语通常意味着网络可为用户提供多种与性能有关的保证，反过来也意味着交换机必须留出足够的资源来满足这种保证。例如，一个给定虚电路中的交换机把每个对应输出链路的带宽的百分之几分配给这段电路。另一个例子是一组交换机可以保证沿一条特定虚电路传输的分组的延迟（排队）不会超过一定的时间。我们在6.5节再讨论服务质量的问题。

相关主题

拥塞简介

交换机设计者必须要面对的一个重要问题是争用（contention）。当多个分组在一台交换机中排队时就会出现争用，因为它们在竞争相同的输出链路。我们在1.4节中讨论交换机如何处理这个问题。可以想象争用发生在独立分组达到的时间段里。而与之相反，拥塞发生在一段更长的时间段里。当交换机中排队的分组太多而占用了所有的缓冲区空间时，交换机不得不丢弃一些分组。我们在了解了网络体系结构的传输协议部分之后，在第6章中再讨论拥塞。但是目前我们看到，网络处理拥塞的方式是与采用虚电路还是采用数据报相关的。

一方面，假设每一台交换机分配足够的缓冲区来处理它支持的每一条虚电路上的分组，像X.25网络中所做的那样。这样，网络中就不存在拥塞的问题了，因为交换机不可能处于这样一种状态：排队的分组比交换机缓存空间还要多。因为在这种网络中，如果它不能给连接提供足够的资源来避免拥塞的发生，那么从一开始交换机就不允许该连接的建立。然而，这种方法的弊端在于极度保守——在同一时刻，所有的虚电路不可能都使用它们的缓冲区，因此，降低了交换机的利用率。

另一方面，数据报模式看上去会引起拥塞，因为直到缓冲区用尽以致于产生了拥塞，你才知道在交换机中有如此多的竞争。这时，要阻止拥塞已经太迟了，唯一的选择就是如何从拥塞中恢复。所幸的是，因为不用为不可能发生的最坏情况预留缓冲区，所以能够更好地利用交换机。

正如在很多常见的情况一样，没有严格的对与错。不产生拥塞（如X.25网络模式）和产生拥塞后再去处理（像简单的数据报模式）的方法各有利弊。我们在第6章中再研究这些设计方法。

近年来基于虚电路技术有很多很成功的技术，特别是X.25、帧中继和异步传输模式（ATM）。然而今天，虽然因特网的无连接模型获得了很大的成功，但并没有得到大面积的应用。近年来最普遍的虚电路应用是构造虚拟专用网（Virtual Private Network，VPN），这个主题将在3.2.9节中讨论。这种应用在当前基于因特网的技术中得到广泛

支持。

相关主题

光交换

对于2000年前后的网络行业观察者来说，光交换可能是最有趣的交换形式。由于多种因素的共同作用，光交换的确在20世纪90年代末成为一种重要的交换技术。一种因素是密集波分多路复用（Dense Wavelength Division Multiplexing，DWDM）设备的商用化，它同时以大量不同长度的光波（或颜色）传输信号，使一条光纤上能够发送大量数据。举例来说，你可以以100个或更多不同波长来发送数据，而每个波长可能携带高达10 Gbps的数据。

另一个因素是光放大器的商用化。光信号在通过光纤时会衰减，经过一段距离（大约40公里左右）后需要用某种方法使它们变得更强。在有光放大器之前，必须在通路上放置中继器来恢复光信号，把它转换成数字电信号，然后再转换回光信号。在把数据送入中继器前，必须用DWDM终端对其进行多路分解。这样，在长距离上就需要用很多DWDM终端来驱动一对光纤。不同于中继器，光放大器是模拟（analog）设备，它增强任何在光纤上发送的信号，即使这些信号是以100个不同波长发送的。光放大器使DWDM装置显得更有吸引力，因为如今一对相隔几百公里的DWDM终端也可以相互通信。而且，你可以在两端升级DWDM装置而不必改动路径中的光放大器，因为它们可以像放大50个波长那样轻而易举地放大100个波长。

有了DWDM和光放大器，就可以建造容量巨大的光网络。但至少还需要一种设备才能使这些网络工作，那就是光交换机（optical switch）。现今大多数所谓的光交换机都使用电子方式实现交换功能。从体系结构的角度来看，它们与电路交换机的共同之处要多于本章描述的分组交换机。一台典型的光交换机有大量能够理解SONET帧格式的接口，并且能够把来自一个输入端口的SONET信道交叉连接到一个输出接口。这样，即便不存在从A直接到B的光纤路径，也能够使用光交换机通过点C提供从点A到点B的SONET信道——只需要一条从A到C的路径、一台位于C点的交换机以及一条从C到B的路径。在这方面，光交换机和图3-3中的交换机存在某种关系，因为它制造了两点间有连接的假象，其实两点间没有直接的物理连接。然而，光交换机不提供虚电路，它们提供"真实的"电路（例如一条SONET信道）。甚至有一些新型光交换机使用微型镜把一台交换机端口的所有光束偏转到另一端口上，这样就可以产生一条从点A到点B的不受干扰的光信道。这种设备所使用的技术称为微机电系统（microelec tromechanical system）。

本书不会广泛探讨光网络，一部分是出于篇幅的考虑。从诸多现实目的出发，可以把光网络看成是一个能够让电话公司在你需要的时间和地点提供SONET链路或其他电路类型的基础设施。然而，有必要指出，下文讨论到的很多技术，例如路由协议和多协议标记交换，在光网络领域都有应用。

异步传输模式

异步传输模式（Asgn chronous Transfer Mode，ATM）几乎是最广为人知的基于虚电路的网络技术，尽管由于一些原因现在已经过了部署该网络的高峰期。由于种种原因，ATM在20世纪80年代和90年代初已成为一种相当重要的技术，其中一个原因是它被电话行业所追捧，而该行业历史上在数据通信中并没有那么活跃（相对于人们构建网络时选

用的链路提供方式而言）。而当许多计算机网络用户认为像以太网和令牌网这种共享介质网的速度太慢时，ATM 技术适时地作为一种高速交换技术出现了。在某些方面，ATM 是一种与以太网交换技术竞争的技术，也被看作是一种与 IP（网际协议）竞争的技术。

至少有一些因素使得 ATM 技术看起来是值得尝试的，如 ATM 分组格式（主结构如图 3-6 所示），通常称为 ATM 信元（ATM cell）。该结构中，我们可以忽略没有作用的通用流控制（GFC）位，重点从标志着 VPI（虚路径识别符，8 位）和 VCI（虚电路识别符，16 位）的 24 位开始看。如果把这些看作一个 24 位的字段，那么可以认为它们与一个虚电路标示符相关联。把该字段分为两部分的原因是为了允许层级的出现：所有电路都具有相同的 VPI，在某些情况下被视为一个组（一个虚路径），且只根据 VPI 就能一起进行交换，从而简化了交换过程。这个过程可以忽略所有 VCI 位且减小了 VC 表的大小。

4	8	16	3	1	8	384(48字节)
GFC	VPI	VCI	类型	CLP	HEC(CRC-8)	有效载荷

图 3-6 UNI 的 ATM 信元格式

跳到最后一个首部字节，我们发现一个 8 位的循环冗余校检位（CRC），称为首部差错校验（HEC）。它采用了 2.4.3 节中给出 CRC-8 多项式，并提供了在信元首部部分进行错误检测和单个位纠错的能力。保护信元首部非常重要，因为 VCI 中的错误将导致信元被误投。

对于 ATM 信元，也许其最重要的被称为信元而不是分组的原因是它仅有一个大小：53 字节。这样做的原因是什么呢？一个很重要的原因是促进实施硬件数据交换。在 20 世纪 80 年代中后期 ATM 刚创建时，10Mbps 的以太网在速度方面采用了最前沿的技术。为了更快速，大多数人都从硬件上考虑，而且，在电话行业，人们一想到交换机就认为非常大，因为电话交换机通常为成千上万的用户服务。当你想要建立一个快速的大规模交换机时，固定长度分组将会非常有用。这有两个主要原因：

1）构建硬件来做简单的工作比较容易。已知每个分组的长度后，处理分组的工作就更为简单。

2）如果所有分组都是相同的长度，那么，可以让多个交换单元以并行方式做同样的事，它们中每一个交换单元都花费相同的时间去完成自己的工作。

对第二个原因，实施并行方式极大地提高了交换机设计的可扩展性。如果说快速并行硬件交换机只能使用固定长度分组才能实现也许有些过分。但是，信元使得构造这样一种硬件的工作变得简单，这是千真万确的，并且在定义 ATM 标准时，我们已经有了大量如何在硬件上构造信元交换机的知识。事实证明，同样的原则在今天仍被许多交换机和路由器所采用，即使它们在处理可变长度的数据分组，也可以将那些分组切割为某种用于交换的信元。我们会在 3.4 节中讨论。

决定使用小的固定长度分组之后，下一个问题就是应将最合适的分组长度设为多少？如果长度太短，则在一个信元中携带的首部信息占的比例较大，所以传输真正数据的链路带宽的比例下降。甚至更严重的是，如果建造了一种设备，每秒可以处理某个最大数目的信元，那么当信元变短时，总的数据速率将随信元长度的减少而成比例降低。以网络适配器为例，它在把分组传送到主机之前把小的信元装配成更大的单元，这样设备的性能就直接取决于信元的大小。另一方面，如果信元的尺寸太大，为得到一个完整的信元，就需要

把传输的数据填充成完整的信元，这就引起带宽浪费的问题。如果信元的有效载荷是48字节而你只想发送1字节，那么还需填充47字节。如果这种情况多次发生，那么链路的利用率将非常低。相对较高的首部载荷比以及被部分填充的信元的发送频率的确在ATM网络中会导致一些明显的低效率。这种现象被反对者称为信元税（cell tax）。

事实证明，选取48字节的ATM信元载荷是一种妥协。使用更大还是更小的信元引起了激烈的争论，但几乎没有人赞同48字节，因为采用2的指数将更有利于计算机工作。

3.1.3 源路由

第三种交换方法称作源路由，这种方法既不使用虚电路也不使用传统数据报。这个名称来自于由源主机提供通过网络交换分组时所需的全部网络拓扑结构信息。

实现源路由有多种不同的方法。一种方法是给每台交换机的每个输出端口编号，并把编号放入分组的首部。那么交换机的功能就很简单：对于到达一个输入端口的每个分组，交换机读出它的首部中的端口号，并把分组发送到那个输出端口上。然而，由于发送主机和接收主机间的路径上一般没有多台交换机，分组的首部需要包含足够的信息，使得路径上的每个交换机都能够确定该把分组放置到哪个输出端口。一种方法就是在分组首部放置交换机端口的一个有序列表。该列表以某种方式旋转，使路径上的下一台交换机总是处在列表的首位。图3-7说明了这个方法。

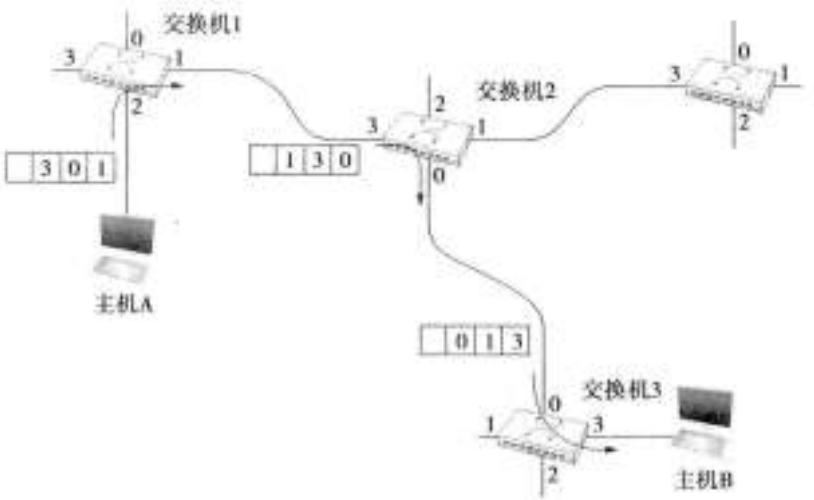


图3-7 交换网络中的源路由（交换机读取最右边的数字）

它们现在在哪？

ATM

在20世纪80年代末和90年代初的一段时间里，很多人认为ATM是准备接管这个世界的。主流电信公司都支持它，并承诺将语音、视频和数据高速网络平滑整合到一个通用网络势在必行。他们采用ATM改变所有使用可变长度分组的网络技术，如把以太网和IP作为"遗留"技术。然而今天以太网和IP仍处于主宰地位，而ATM则已过时。作为一种访问IP网络的方式，你仍然可以找到很多ATM部署方案。值得注意的是，很多数字用户线（DSL）接入网络是采用ATM构建的，所以一定数量的宽带网络接入是基于ATM链路的。虽然这些隐藏在DSL调制解调器之后，将以太网帧封装到接入网络的信元

当中。

关于为什么ATM没有在世界上普及这个问题仍在讨论。回顾这个问题，一种基本的重要因素是在ATM出现的时候，IP已经以自己的方式完全确立了牢固的地位。在20世纪80年代，还没有很多人接触过因特网，但IP已经实现全球互联并且每年连接的主机数量以两倍的速度增长。因为IP的核心是平滑连接各种网络，所以当ATM出现时，并没有像它的支持者想象的那样取代IP，而是很快被采纳为运行IP的另一种网络类型。因此，ATM更直接地面对着以太网的竞争而不是IP的竞争。便宜的以太网交换技术和无需昂贵的光交换机的百兆以太网捍卫了其作为主导局域网技术的地位。

在这个例子中，分组需要经过三台交换机才能从主机A到达主机B。在交换机1中，分组从端口1输出，在下一台交换机中从端口0输出，在第三台交换机中从端口3输出。这样，当分组离开主机A时，首部带有最初的端口表(3，0，1)。这里，我们假设每一台交换机读取列表中最右端的元素。为了确保下一台交换机能得到正确的信息，每台交换机在读出自己的记录之后对列表进行旋转。这样，当有分组离开交换机1到达交换机2时，首部列表是(1，3，0)，第二台交换机完成另外一次旋转，送出分组，此时其首部列表为(0，1，3)。尽管在图中没有显示出来，但是交换机3还要进行一次旋转，把分组的首部列表恢复到最初主机A发送时的状态。

这种方法还需要注意几个问题。第一，它假设主机A充分了解网络的拓扑结构，能形成分组的首部信息，其中包含路径中每一台交换机的所有正确输出方向。这同数据报网络中构造转发表的问题或在虚电路网络中算出向哪里发出建立分组的问题有某些类似之处。第二，注意我们不能预测一个分组的首部需要多大，因为它必须能够为路径上的每台交换机保留一个字的信息。这就说明分组的首部长度可能是可变的，而且没有上界，除非我们能确切地预测分组需要通过的交换机的最大数目。第三，这个方法还有一些变种。例如，每台交换机只是删掉其首部列表的第一个元素，而不是旋转首部信息。然而旋转要比删除首部信息有利，因为主机B可以得到分组首部的一份完整副本，这有助于主机B了解如何把数据回送到主机A。另外一个变种是，让分组的首部带有指向当前“下一个端口”的指针，这样每台交换机只需要更新指针而不用旋转分组的首部，便于高效的实现。我们在图3-8中说明这三种方法。在每种情况下，这台交换机需要读的记录值是A，下一台交换机需要读的记录值是B。

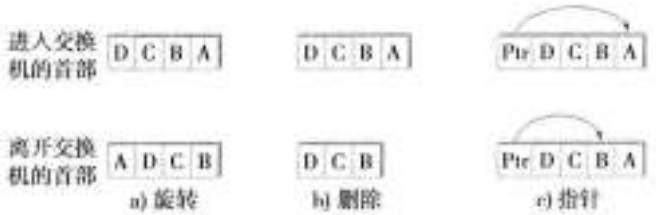


图3-8 源路由中处理首部的三种方法（记录值从右向左读）

源路由可以应用在数据报网络和虚电路网络中。例如，IP协议是一个数据报协议，包含一个源路由选项，允许选定的分组使用源路由选择，而多数情况采用传统的数据报交换。源路由选择也应用在一些虚电路网络中，用于获得从源到目的路径上的初始建立请求。

源路由有时用“严格”和“宽松”来分类。在“严格”源路由中，每个沿路径的节点

都必须声明，而“宽松”源路由可以仅声明一组要经过的节点，而并不确切地说明是否沿一个节点到下一个节点。宽松源路由可以理解为一组路点，而不是一个完整声明的路径。宽松选项有助于减小源用于创建源路由的信息量。在一些大型网络中，主机很难得到建立正确严格源路由所需要的到目的地的完整路径信息。但两种类型的源路由都可以在一些特定环境下得到应用，4.3 节中描述了一个应用。

3.1.4 网桥和局域网交换机[⊖]

在讨论过交换的某些基本思想后，现在更集中地讨论一些特殊的交换技术。我们先考虑用于在共享介质局域网（例如以太网）中转发分组的一类交换机。这样的交换机有时通称为局域网交换机，历史上曾称为网桥。

假设现有两个想要互联的以太网，那么必须做的第一件事是在它们之间放一个中继器，像第 2 章描述的那样。然而，当超出以太网的物理限制时，这并不是一个有效的解决办法。（回忆一下，任一对主机之间最多只能有两个中继器，并且总长度不能超过2 500m。）另一个办法是在两个以太网之间放一个节点，由节点来转发从一个以太网到另一个以太网的帧。此节点处于混杂模式，接收从任意以太网传来的所有帧，并将它们转发到另一个以太网。

我们所描述的节点通常称为网桥（bridge），由一个或多个网桥连接的 LAN 集合通常称为扩展局域网（extended LAN）。最简单的一种情况是，网桥仅在它们的输入端口上接收局域网的帧并在所有其他输出端口上将这些帧转发出去。这种简单的策略用于早期的网桥，但它一直在改进，使得网桥成为 LAN 集合互联的有效机制。本节其余部分将加入更有趣的细节。

注意，网桥符合前一节的交换机定义：多输入多输出设备，能够从一个输入传送分组给一个或多个输出。回顾一下，这提供了一个增加网络总带宽的方法。例如，当单个以太网段能够传输的总通信量是 100Mbps 时，一个以太网网桥能够传输 $100n$ Mbps，这里 n 是网桥上的端口（输入和输出）数。

1. 学习型网桥

注意，网桥不需要转发所有收到的帧，所以我们可以对网桥进行第一项优化。考虑图 3-9 中的网桥。从主机 A 到主机 B 的帧无论何时到达端口 1 都无需经网桥转发给端口 2。那么，问题是网桥如何得知各个主机在哪一端口上呢？

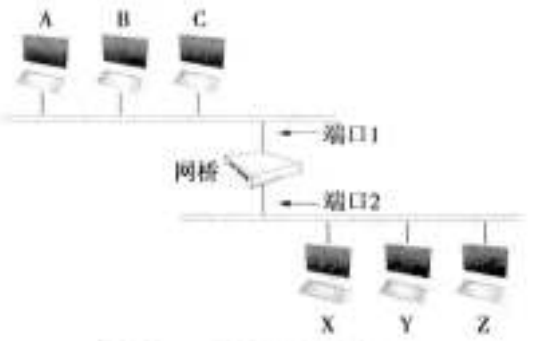


图 3-9 学习型网桥图解

一种选择是人为地在网桥内放置一个类似表 3-4 的表。这样，无论何时网桥在端口 1 上接收到给主机 A 的帧，都不必转发给端口 2，因为主机 A 可直接接收连接到端口 1 上的 LAN 的帧。任何时候在端口 2 收到发往主机 A 的帧，网桥都将转发给端口 1。

实际上没有人会采用这种方式来配置网桥，人工维护转发表是一个相当大的负担，特

⊖ 参见“实验四”。

别是考虑到网桥能使用简单的技巧得到这些信息时。所用方法的思想是每个网桥检查它所接收的帧的源地址。这样，当主机 A 向网桥任何一边的主机发送帧时，网桥接收到这一帧，并且记录下从主机 A 来的帧由端口 1 接收这一事实。用这种方法网桥可以建立起如表 3-4 所示的表。

表 3-4 通过网桥维护的转发表

主 机	端 口
A	1
B	1
C	1
X	2
Y	2
Z	2

注意，使用这种表的网桥能使用 3.1.1 节描述的转发数据报（或无连接）模式。每个分组带有一个全局地址，网桥通过查找表中的地址确定从哪个端口发出分组。

网桥首次启动时表是空的，表中的记录是随时间逐渐增加。而且，每条记录都有相应超时时间，超过一定时间，网桥便将其丢弃。这是为了适应主机从一个网络移动到另一个网络时的情况，这时其地址也随之发生移动。因此，这张表不需要包含全部主机地址。如果网桥接收到一个帧，而它要送达的主机地址不在当前表中，那么网桥会将这一帧从所有其他端口转发出去。换言之，此表只是可以过滤掉一些帧的简单优化，而没有正确性的要求。

2. 实现

实现学习型网桥算法的代码很简单，这里我们简单描述一下。BridgeEntry 结构定义网桥转发表中的一个记录，它们存储在一个 Map 结构（它支持 mapCreate、mapBind 和 mapResolve 操作）中，以便当表中已存在的源主机发来的分组到达时，能够快速地查找记录。常量 MAX_TTL 表示一个记录被丢弃前在表中保存的时间。

```
#define BRIDGE_TAB_SIZE   1024  /* max. size of bridging
                                   table */
#define MAX_TTL           120   /* time (in seconds) before
                                   an entry is flushed */

typedef struct {
    MacAddr     destination;    /* MAC address of a node */
    int         ifnumber;       /* interface to reach it */
    u_short     TTL;            /* time to live */
    Binding     binding;        /* binding in the Map */
} BridgeEntry;

int     numEntries = 0;
Map     bridgeMap = mapCreate(BRIDGE_TAB_SIZE,
                              sizeof(BridgeEntry));
```

当一个新的分组到达时更新转发表的例程由 updateTable 给出。传递的参数是包含在分组中的源 MAC 地址和接收分组的接口号。这里没有给出会被定时唤醒的另一个例程，它扫描转发表中的记录，并减少每个记录的 TTL（生存期）字段的值，丢弃 TTL 值为 0 的记录。注意，每次分组到达时更新一个已存在的表记录，TTL 被重置为 MAX _ TTL，能连到目标主机的接口被更新，以反映最新接收分组的情况。

```
void
updateTable (MacAddr src, int inif)
{
    BridgeEntry       *b;
```

```
    if (mapResolve(bridgeMap, &src, (void **)&b) == FALSE )
    {
        /* this address is not in the table, so try to add it */
        if (numEntries < BRIDGE_TAB_SIZE)
        {
            b = NEW(BridgeEntry);
            b->binding = mapBind( bridgeMap, &src, b);
            /* use source address of packet as dest. address in table */
            b->destination = src;
            numEntries++;
        }
        else
        {
            /* can't fit this address in the table now, so give up */
            return;
        }
    }
    /* reset TTL and use most recent input interface */
    b->TTL = MAX_TTL;
    b->ifnumber = inIf;
}
```

注意，在网桥转发表容量已满的情况下这个实现采用了一种简单的策略，即添加新地址失败。回顾一下，正确地转发不要求网桥表的完整性，它只是用来优化性能。如果表中的某条记录当前不被使用，最终会因超时而被删除，从而为新记录腾出空间。另一个方法是在发现表满时，调用某种高速缓存替换算法。例如，我们可以查找并删除具有最小TTL值的记录，以便安置新记录。

3. 生成树算法

如果扩展局域网内没有产生环，那么前面所讲的策略是很好的。环的产生可能造成帧永远在扩展局域网中循环这种可怕的故障。从图3-10描述的例子中容易看出这种情况，例如，网桥B1、B4和B6形成一个环。假设一个分组从以太网J进入网桥B4且其目的地并没有存在于任何一个网桥地址转发表内：B4发送一个分组拷贝到H和I。这时网桥B6转发该分组到能被B1看到的以太网G，然后转发该分组回到以太网H。B4的地址表中仍然没有目标地址，所以转发这个包到以太网I和J。没有办法能阻止双向连接的B1、B4和B6之间无休止的循环。

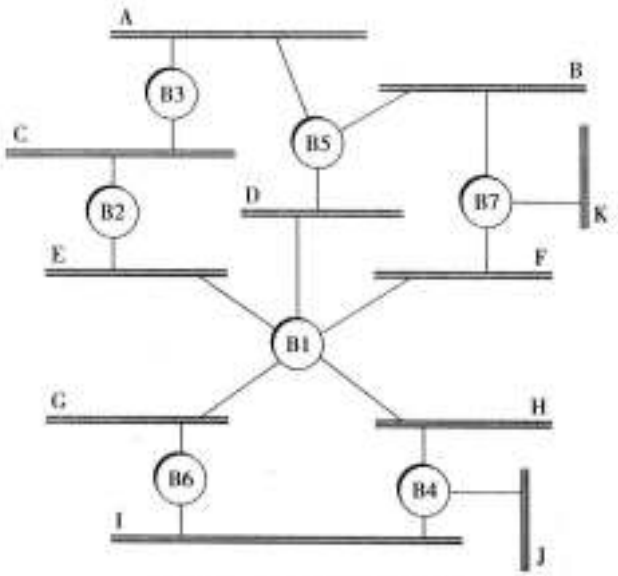
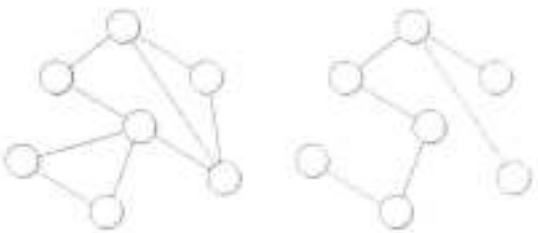


图3-10 带环的扩展局域网

为什么扩展局域网会有内部循环？一种可能是网络由多位管理员管理，比方说，因为网络跨越一个机构的多个部门。在这种情况下，可能没有人知道网络的整体配置，这就意味着可能添加了一个引起环的网桥而无人知道。另一种

很可能的情况是有目的地在网络中建立环，为网桥发生故障时提供冗余。毕竟，仅需要一个连接错误就能将一个没有循环的网络在分裂成两个独立的分区。

无论环是怎样产生的，网桥必须能正确处理环。让网桥运行分布式生成树（spanning tree）算法可以解决这个问题。如果将扩展局域网看作有环图，那么生成树是覆盖此图所有顶点的无环子图。就是说，生成树保留原图的所有顶点，却丢弃一些边。例如，图 3-11a 是一个有环图，图 3-11b 是可能的多个生成树中的一个。



a）有环图　　b）对应的生成树

图 3-11　示例图

生成树的想法很简单：在扩展网络中，网络拓扑结构子集没有环路并能到达所有的局域网。难点是所有的网桥如何协调决策并形成生成树的单一视图。因为通常一个典型拓扑是能够被多个生成树拓扑所覆盖的。这个问题的答案就在我们现在正在讨论的生成树协议中。

生成树算法是由数字设备公司（DEC）的 Radia Perlman 开发的，用于协调从一组网桥中为某个特定的扩展局域网产生生成树的协议。（IEEE 802.1 关于 LAN 网桥的规范就是基于此算法的。）实际上，这意味着每个网桥将决定它使用和不使用的转发帧的端口。从某种意义上来说，通过从扩展局域网的拓扑结构中去掉一些端口可使其退化成一棵无环树。当你看到我们有意在网络中加入环以提供冗余时可能会感到奇怪，因为有的网桥可能不参与转发帧。然而，算法是动态的，这意味着那些网桥随时准备在某个网桥出故障时重新配置以形成新的生成树。

图 3-12 给出对应于图 3-10 中扩展局域网的生成树。在这个例子中，B1 是根网桥，因为它的标识符最小。注意 B3 和 B5 都连接到 LAN A 上，但 B5 是指派网桥，因为它距离根更近。类似地，B5 和 B7 都连接到 LAN B 上，虽然从 B1 到两个网桥等距离，但是在这种情况下，B5 由于有较小的标识符而成为指派网桥。

观察图 3-10 给出的扩展局域网，完全可以根据上面给出的规则计算图 3-12 中的生成树。然而扩展局域网中的网桥无法看到整个网络的拓扑结构，更不用说窥视其他网桥内部的标识符了。因此，网桥必须彼此间交换配置消息，然后根据这些消息确定它们是根网桥或指派网桥。

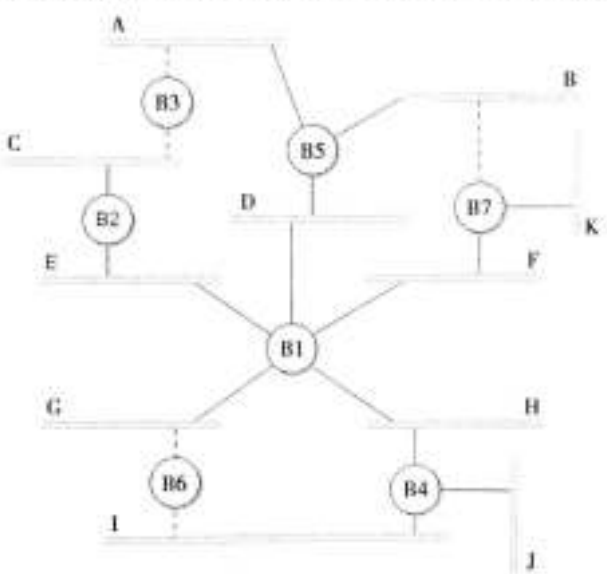


图 3-12　某些端口不被选择的生成树

配置消息包含以下三条信息：

1）发送消息的网桥的标识符。

2）发送网桥认定的根网桥的标识符。

3）从发送网桥到根网桥的按跳数度量的距离。

每个网桥记下在它的每个端口上看到的当前最优（best）（“最优”将在下面定义）配置消息，包括从其他网桥接收的和它自己发送的消息。

最初，每个网桥认为自己是根，并

从每个端口发出配置消息，标识自己是根并给出到根的距离为 0。网桥在某个端口接收到配置消息后，就检查这个新消息是否优于该端口记录的配置消息。如果满足以下条件，则认为新配置消息优于（better）当前记录的消息。

- 标识一个具有更小标识符的根。
- 标识一个带有相同标识符但具有更短距离的根。
- 根标识符和距离都相等，但发送这条消息的网桥具有更小的标识符。

如果新消息优于当前记录的消息，网桥则丢弃旧消息并保存新消息。然而，它首先将到根的距离字段加 1，因为这个网桥到根的距离比发送消息的网桥到根的距离远一跳。

当一个网桥接收到说明自身不是根网桥的配置消息时（即收到来自一个有更小标识符的网桥的消息），网桥终止生成自己的配置消息，而是先对来自其他网桥的配置消息中的距离字段加 1，然后转发。类似地，当一个网桥接收到说明自身不是某端口的指派网桥的配置消息时（即配置消息来自与根的距离更近或等于自己与根的距离但具有更小标识符的网桥），网桥停止在该端口发送配置消息。这样，当系统稳定时，只有根网桥仍然产生配置消息，而其余网桥仅在那些是指派网桥的端口上转发这些消息。从这点上说，一个生成树建立后，所有网桥需要为生成树协调其使用的端口，只有这些端口才能在扩展网络里转发数据分组。

下面来看一个网桥配置的实例。假设某网络所在大楼的电源刚刚恢复，所有网桥大约在同一时间启动，考虑在图 3-12 中会发生什么。所有网桥将开始发出自己是根的消息。我们将一个来自节点 X、与根节点 Y 的距离是 d 的配置消息表示为（Y，d，X）。以节点 B3 的活动为例，一系列事件将如下展开：

1）B3 接收（B2，0，B2）。

2）因为 2<3，所以 B3 接受 B2 为根。

3）B3 将 B2 通知的距离（0）加 1，并发送（B2，1，B3）给 B5。

4）同时，因为 B1 具有更小的标识符，所以，B2 接受 B1 为根，并发送（B1，1，B2）给 B3。

5）B5 接受 B1 为根并发送（B1，1，B5）给 B3。

6）B3 接受 B1 为根，且由于看到 B2 和 B5 都比它距离根近，因此 B3 停止在这两个端口上转发消息。

这使 B3 带有两个未被选择的端口，如图 3-12 所示。

即使系统稳定后，根网桥还会继续定期发送配置消息，其余网桥继续像前面描述的那样转发这些消息。如果某个网桥出现故障，下游的网桥将不能接收到这些配置消息，在等待一个指定的时间段后，它们会重新宣布自己是根，刚才描述的算法会再选一个新的根和新的指派网桥。

需要注意的一件重要的事情是，虽然无论何时某个网桥出错，算法都能重新构造生成树，但是不能为了绕开拥塞网桥而选择另一条路径来转发帧。

4. 广播和多播

之前讨论的重点是网桥如何从一个 LAN 转发单播帧到另一个 LAN。由于网桥的目标是透明地扩展局域网，并且由于大多数 LAN 都支持广播和多播，因此网桥也必须支持这两个特性。广播比较简单，每个网桥将带有目标广播地址的帧转发到除了接收它的端口以

外的其他活动（选择）端口。

多播可以按同样的方法实现，每台主机自己决定是否接收消息。实际应用中就是这样做的。然而，在扩展局域网中，并不是所有 LAN 都必须有一台作为某多播组成员的主机，因此情况可能会更好。特别是，可以扩展生成树算法来删除那些不需要转发多播帧的网络。考查帧由图 3-12 中 LAN A 的一台主机发往一个组 M。如果 LAN J 中的主机都不属于组 M，那么网桥 B4 就没有必要在网络上转发帧。另一方面，虽然 LAN H 中的任一主机都不属于组 M，但这并不意味着网桥 B1 不转发多播帧到 LAN H。这取决于在 LAN I 和 LAN J 中是否有组 M 的成员。

网桥如何知道是否需要通过某个给定的端口转发多播帧？网桥是通过观察从该端口接收到的源（source）地址得知的，这与网桥如何决定是否通过某个特定端口转发一个单播帧的方法一样。当然，组通常不是帧的源，所以我们的说法有点不妥。特别地，组 M 的每个成员主机都必须定时发送一个帧，在首部的源字段中携带组 M 的地址。这个帧的目标地址就是网桥的多播地址。

注意，以上描述的多播扩展方法虽然已经提出，但还没有广泛采纳。而且，在今天的扩展局域网上，多播与广播的实现方法是完全一样的。

5. 网桥的局限性

以上描述的基于网桥的解决方案只能用于一种非常有限的环境：连接少数相似的 LAN。当我们考虑到可扩展性和异构性问题时，这种局限性就变得很明显。

在可扩展性问题上，用网桥连接过多的 LAN 是不现实的，一般情况下不多于几十个。一个原因是生成树算法是线性扩展的，即没有为扩展局域网提供分层结构。另一个原因是网桥转发所有的广播帧。虽然在一种受限制的环境（如一个部门）下，对所有主机来说看到相互的广播消息是合理的，但一个更大范围（如一个大公司或一所大学）内的所有主机不可能都愿意受到相互广播消息的打扰。换言之，广播的规模不能太大，因此扩展局域网的规模不能太大。

增强扩展局域网的可扩展性的一种方法是虚拟 LAN（Virtual LAN，VLAN）。VLAN 允许将一个扩展局域网划分成几个看起来独立的 LAN，给每个 VLAN 赋一个标识符（有时称为颜色（color）），而且只有当两个网段有相同的标识符时，分组才能从一个网段传送到另一个网段。这样可以限制接收任何给定广播分组的扩展局域网上的网段数目。

我们通过例子来看 VLAN 是怎样工作的。图 3-13 给出在 4 个不同 LAN 网段上的 4 台主机。在没有 VLAN 的情况下，来自任何主机的广播分组将到达所有其他主机。现在我们假设把连接到主机 W 和主机 X 的网段定义为一个 VLAN，称为 VLAN 100。我们还定义连接到主机 Y 和 Z 的网段为 VLAN 200。这样做时，我们需要给网桥 B1 和 B2 的每个端口配置一个 VLAN 标识符。我们认为在两个 VLAN 中都包括 B1 到 B2 的链路。

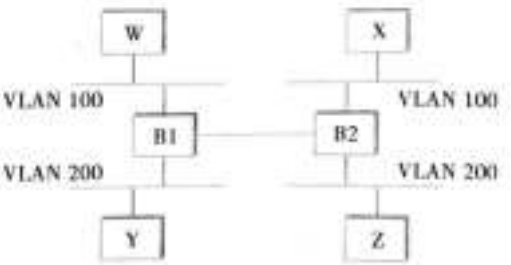


图 3-13 两个虚拟局域网共享一个主干

当主机 X 发送的分组到达网桥 B2 时，网桥观察到它来自配置为 VLAN 100 的一个端口。它在以太网的首部和有效载荷之间插入一个 VLAN

首部。我们感兴趣的 VLAN 首部部分是 VLAN 标识符；此时，这个标识符设置为 100。网桥现在按正常规则转发分组，附加的限制是这个分组不能发送给不属于 VLAN 100 的接口。这样，分组决不会发送到 VLAN 200 中连接主机 Z 的接口上，即使是一个广播分组。但是，分组被转发到网桥 B1，按照同样的规则，B1 可以转发分组给主机 W，但不转发给主机 Y。

VLAN 的一个有吸引力的特性是它能更改网络的逻辑拓扑结构，而不用移动任何线路或更改任何地址。例如，如果想使连接到主机 Z 的网段成为 VLAN 100 的一部分，并由此使 X，W 和 Z 在同一个虚拟 LAN 上，那么我们只需更改网桥 B2 的一条配置信息。

在异构性问题上，网桥完全受限于它们能够互联的网络的类型。特别是，网桥使用网络帧的首部，因此只能支持那些地址格式相同的网络。这样，网桥可以用来连接以太网与以太网、令牌网与令牌网以及 802.11 与另一个 802.11 网络。因为都支持相同的 48 比特地址格式，所以一个以太网与 802.11 网络之间就有了建立桥接的可能。然而，网桥不易推广到其他类型的网络，如 ATM。

尽管有这些局限性，但网桥仍然是整个网络中很重要的一部分。它的主要优点是允许多个 LAN 彼此透明地连接，就是说，网络可以连接起来而并不需要在终端主机上运行任何另外的协议（或者即使意识到需要这样做）。当主机希望在一个多点播送组中宣布它们的成员关系时可能存在一个潜在的异常，如 3.1.4 节所描述的情况。

然而，这种透明性可能是有危险的。如果一台主机（或更准确地说主机上运行的应用程序和传输协议）是在运行于单一 LAN 的假设下编程的，那么在源主机与目的主机之间插入网桥可能会出现意想不到的事。例如，如果一个网桥拥塞，它可能不得不丢弃帧，而单一以太网丢弃帧的情况是很少见的。又如，扩展局域网中任一对主机之间的时延变得更大且高度可变，而单一以太网的物理局限性使得时延很小且可预知。作为最后的例子，在一个扩展局域网中对帧重排序是可能的（尽管可能性不大），而在单一以太网中帧的顺序是不会搅乱的。结论是在运行于单一以太网段的假设下设计网络软件绝不会是安全的。网桥就是这种情况。[⊖]

3.2 互联网基础（IP）

在前一章中，我们看到可以使用网桥和局域网交换机建造相当大型的局域网，但是这些方法在扩展和处理异构问题上有局限性。本章将探索克服桥接网络局限性的方法，使得我们能够使用相当有效的路由建造大型的和高度异构的网络。我们称这样的网络为互联网（internetwork）。在下一章里，我们将继续讨论如何构建一个真正的全局互联网。当前，我们先了解一些基础知识。首先认真考虑一下互联网一词的含义。

3.2.1 什么是互联网？

我们用带小写“i”的“internetwork”或仅用“internet”指可提供某种主机到主机的分组传送服务的相互连接的任意网络集合。例如，一个有很多站点的公司可以租用电话公司的点到点链路将其不同站点的 LAN 互联成一个专用的互联网。当谈论应用广泛的目前已连接大部分网络的全球互联网时，我们用带大写“I”的“Internet”来称呼它。本书一贯注重基本原理，因此我们主要希望你了解“internet”的原理，但是用“Internet”中的

⊖ 参见“实验五”。

实例来阐明这些思想。

“网络”、“子网”和“互联网”是几个易混淆的术语。在3.2.5节以前，我们不谈子网。现在，我们用网络（network）来指前两章讨论的直接相连的网络或交换网络。这样的网络只使用一种技术，如802.5、以太网或ATM。互联网（internetwork）是这些网络互联的集合。有时，为了避免意义不明确，我们把互联的底层网络称为物理（physical）网络。互联网是由物理网络集合构成的逻辑（logical）网络。在这种情况下，仍然可以将由网桥或交换机相连的以太网集合看成一个单一的网络。

图3-14给出了一个互联网的例子。互联网通常称为“网际网”，因为它是由很多更小的网络组成的。此图中，我们可以看到以太网、FDDI环和点到点链路。这些网络中的每个网络采用的都是单一技术。将这些网络互联的节点称为路由器（router），有时也称为网关（gateway），但由于这个词还有其他含义，所以我们只使用路由器。

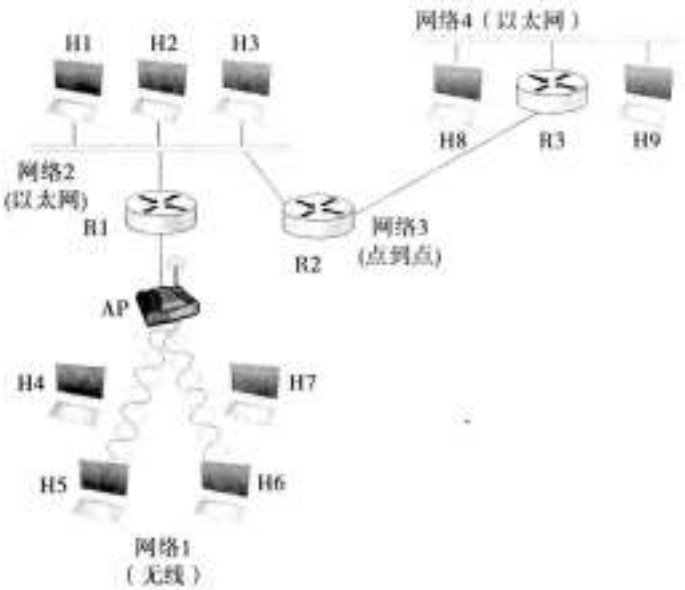


图3-14 一个简单的互联网。H_n=主机；R_n=路由器

网际协议（IP）是我们现今用来建造可扩展的异构互联网络的关键工具。它最早以其发明者的名字命名为Kahn-Cerf协议。[⊖] 可以认为IP运行于网络集合的所有节点（主机和路由器），并定义一种基础结构，使这些节点和网络具备一个逻辑互联网的功能。例如，图3-15给出了图3-14中的主机H5和H8是如何通过互联网逻辑相连的，包含每个节点上运行的协议图。注意，TCP和UDP等高层协议通常运行在主机的IP之上。

本章其余大部分将介绍有关IP的各个方面。当然，建造互联网也可以不使用IP，例如，Novell创建的互联网协议称为IPX，它的基础是施乐公司设计的XNS互联网。然而因特网的规模使IP成为最重要的协议，或者说，只有IP因特网是真正面对可扩展性问题的。因此，它为研究可扩展的互联网协议提供了最好的实例。

⊖ Robert Kahn和Vint Cerf因IP的设计而荣获2005年的图灵奖，这一奖项被认为是计算机领域的诺贝尔奖。

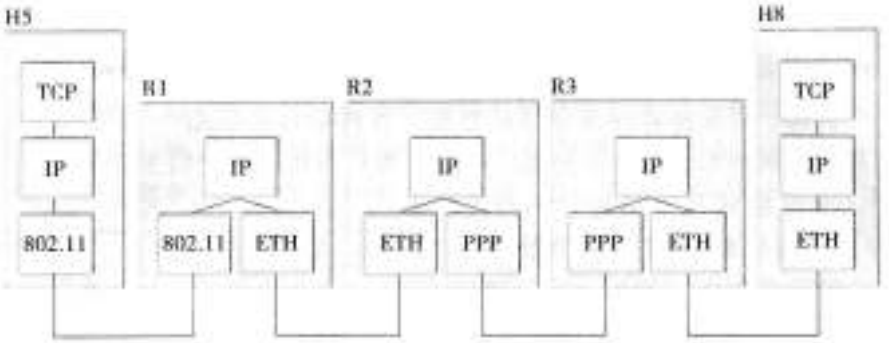


图 3-15 一个简单的互联网，说明用于连接图 3-14 中的 H5 和 H8 的协议层。ETH 是运行在以太网上的协议

3.2.2 服务模型

建造互联网时最好先定义服务模型（service model），即想要提供的主机到主机的服务。为互联网定义服务模型时主要关心的问题是：只有当每个底层物理网络都能提供一种主机到主机的服务时，我们才能提供这种服务。例如，如果底层网络技术可能随意延迟分组，那么互联网服务模型就不能保证在 1ms 之内传送每个分组。因此，定义 IP 服务模型的原则是使它的要求尽量少，在互联网中出现的任何技术都能提供必要的服务。

可将 IP 服务模型看成两部分：一是编址方案，提供标识互联网中所有主机的方法；二是传送数据的数据报（无连接的）模型。这种服务模型有时也称为尽力（best-effort）服务模型，这是因为尽管 IP 尽力传送数据报，但并不提供保证。我们推后编址方案的讨论，先来看数据传送模型。

1. 数据报传送

IP 数据报是 IP 的基础。3.1.1 节中已经提到，数据报是一种在网络中以无连接方式发送的分组类型。每个数据报携带足够的信息以使网络将分组传送到正确的目的地，不需要预先建立任何机制来告诉网络当分组到达时该怎么做。你只需发送它，网络就会尽力把它送到所希望到达的目的地。“尽力”服务的意思是：如果出现错误和分组丢失、损坏、误传送或因任何原因而没有到达目的地，网络什么也不做——它已经尽了最大努力，这就是它必须做的全部事情，而不采取从故障中恢复的任何行动。有时也将这种服务称为不可靠（unreliable）服务。

尽力的、无连接的服务大概是你能从互联网中得到的最简单的服务，并且这是很有用的服务。例如，如果你在一个提供可靠服务的网络上提供尽力服务，那么很好，你最终得到的尽力服务正好是总能传送分组的服务。反之，如果你在一个不可靠的网络上采用一个可靠的服务模型，那么你将不得不把很多额外的功能置于路由器上以弥补底层网络的不足。尽可能使路由器保持简单是 IP 最初的设计目标之一。

IP “在任何技术上运行”的能力经常被引述为其最重要的特性之一。值得注意的是，在发明 IP 时，现今 IP 赖以运行的很多技术还不存在。迄今，人们发明的所有网络技术对 IP 来说都具备一定的适应度，甚至有人声称 IP 可以运行在由信鸽传输消息的网络上。

尽力传送不仅仅意味着分组可能丢失。有时分组可能不按顺序传送，或者同一分组可能会传送不止一次。运行在 IP 之上的高层协议或应用需要知道所有可能的错误模式。

2. 分组格式

显然，IP 服务模型的一个关键部分是可携带的分组类型。像大多数分组一样，IP 数据报包含一个首部，后面接着许多字节的数据。首部的格式如图 3-16 所示。注意，我们采用与前面几章不同的形式来表示分组，这是因为后面几章在我们主要关注的互联网层及其上各层中，分组格式几乎都设计为按 32 位边界对齐，以简化软件对它们的处理任务。因此，一般表示方法（例如，用于因特网 RFC）就是将它们按 32 位字的序列取出。顶部的字先被传送，并且每个字最左边的字节先被传送。在这种表示方法中，可以很容易地识别出 8 位倍长的字段。特殊情况下，如果字段长度不是 8 位的偶数倍，还可以通过查看分组顶部标出的比特位置来确定字段的长度。



图 3-16 IPv4 分组的首部

查看 IP 首部的每个字段，我们发现尽力数据报传送的“简单”模式还有一些微妙的特性。Version（版本）字段说明 IP 的版本。当前 IP 的版本是 4，有时称为 IPv4。注意将这个字段置于数据报的开头会易于我们在后续版本中重新定义分组格式的其余部分。首部处理软件从查看版本开始，然后根据相应的格式分别处理分组的其余部分。下一个字段 HLen（首部长度）以 32 位字为单位指定首部的长度。在没有其他选项的时候，首部通常是 5 个字长（20 字节）。8 位的 TOS（服务类型）字段多年来有过很多不同的定义，但它的基本功能是允许根据不同的应用需求对分组进行不同的处理。例如，TOS 值可决定分组是否应放在一个特殊的接受低延迟的队列中。我们将在 6.4.2 节及 6.5.3 节中更详细地讨论这个字段的使用（和它的新名字）。

首部中接下来的 16 位是数据报的 Length（长度）字段，包括首部在内的长度。不同于 HLen 字段，Length 字段计字节数而不是字数。因此，IP 数据报的最大尺寸为 65 535 字节。然而，IP 运行的物理网络可能不支持如此长的分组。因此，IP 支持分片和重组。首部的第二个字包含分片信息，它的使用细节将在下面的“分片和重组”中读到。

在首部的第三个字中，第一个字节是生存期 TTL 字段。这个名字反映了它的历史含义而不是现在的使用方式。此字段的目的是捕捉在路由环路中转来转去的分组并丢弃它们，而不是让它们无限地消耗资源。最初，TTL 被设置为允许分组生存的一个指定的秒数，沿途的路由器将减小这个字段值直到为 0。然而，由于分组在路由器中的等待时间很少能达到 1 秒钟，并且路由器不是总能访问到一个公共时钟，因此大多数路由器都是在转发分组时将 TTL 字段值减 1。这样，它就变成按跳计数而不是一个计时器，这也是捕捉陷入路由循环的分组的一个绝好的方法。一个细节是发送主机对此字段的初始设置：设置太高，分组就会在被丢弃前转很多圈；设置太低，分组又可能无法到达目的地。64 是当前的默认值。

Protocol（协议）字段只是一个多路分解键，用于标识 IP 分组应被送至的高层协议。已定义了值的协议有 TCP（传输控制协议-6）、UDP（用户数据报协议-17）以及很多在协议图中位于 IP 之上的其他协议。

Checksum（校验和）字段通过将整个 IP 首部看作一个 16 位字的序列进行计算，使用二进制反码相加，并对结果取反码。这就是 2.4 节中描述的 IP 校验和算法。因此，如果首部的任一比特在传输中被损坏，那么收到分组的校验和将包含不正确的值。由于一个被损坏的首部可能在目的地址中包含错误（因此可能导致错误传送），因此应当丢弃校验和值不正确的分组。需要注意，校验和并不具备像 CRC 那么强的检错能力，但是它在软件中很容易计算。

首部所需的最后两个字段是分组的 SourceAddr（源地址）和 DestinationAddr（目的地址）。后者是数据报传送的关键：每个分组包含一个完整的目的地址，所以每台路由器能够做出转发决定。接收方根据源地址决定是否接收分组并做出应答。IP 地址在 3.2.3 节中讨论，现在重要的是要知道 IP 定义自己的全局地址空间，不依赖于在何种物理网络上运行。正如我们将看到的，这是支持异构性的关键之一。

最后，在首部的后端有多个选项。是否出现这些选项可通过检查 HLen（首部长度）字段确定。尽管选项很少被用到，但一个完整的 IP 实现必须能处理所有选项。

3. 分片和重组

在一个异构的网络集合中，提供统一的主机到主机服务模型需要面对的问题之一是每种网络技术都试图自己定义分组的大小。例如，以太网能接收最大长度为 1 500 字节的分组，而 FDDI 分组最长可达 4 500 字节。这就留给 IP 服务模型两种选择：确保所有 IP 数据报足够小，使其能适合任何网络技术的分组；或者当 IP 数据报对某一网络技术来说太大时，提供一种方法将分组分片和重组。后一种方法较好，特别是考虑到新的网络技术将不断出现，而 IP 需要在所有技术上运行，这使得选择一个合适的小范围数据报尺寸变得困难。这也就意味着主机没有必要发送小分组，因为这将使被发送数据的每个字节都需要更多的首部，会浪费带宽和消耗处理资源。例如，两个连接至 FDDI 网络的主机通过一条点到点链路互联时，没有必要发送足以适合以太网的小分组。

这里的中心思想是每种网络类型都有一个最大传输单元（Maximum Transmission Unit，MTU），这是一帧中所能携带的最大数据报。注意这个值小于网络上的最大分组尺寸，因为 IP 数据报需适合链路层帧的有效载荷（payload）。[⊖]

因此，当主机发送 IP 数据报时，它可以根据需要选择尺寸。一个合理选择是与主机直接相连的网络的 MTU。然后，只有当到目的地的路径中包含一个使用更小 MTU 的网络时才需要分片。然而，如果 IP 之上的传输协议发给 IP 一个大于本地 MTU 的分组，那么源主机必须将其分片。

通常当路由器接到想要在网络上转发的数据报，而这个网络的 MTU 比所接到的数据报小时，那么在路由器上将进行分片。为使这些分片在接收主机上能够重组，它们都在标识符（Ident）字段上携带同样的标识符。这个标识符由发送主机选择，并且对于所有可能在某个合理时段内从这个源主机到达目的主机的数据报来说是唯一的。由于原始数据报的所有分片都包含这个标识符，所以重组主机能够识别出这些汇聚到一起的分片。如果不是所有的分片都到达接收主机，那么主机将放弃重组进程并丢弃已到达的分片。IP 并不试图恢复丢失的分片。

⊖ 幸运的是，ATM 网络中的 MTU 要比一个单一信元大得多，因为 ATM 有自己的分片方式。在 ATM 中，链路层帧称为汇聚子层协议数据单元（CS-PDU）。

为了理解整个过程，考虑当图 3-14 所示的互联网例子中的主机 H5 向主机 H8 发送一个数据报时会发生什么。假设两个以太网和 802.11 网络的 MTU 是 1 500 字节，点到点网络的 MTU 是 532 字节，那么从 H5 发送的一个 1 420 字节的数据报（20 字节的 IP 首部加上 1400 字节的数据）在经过 802.11 网络和第一个以太网时不需要分片，但是在路由器 R2 上就必须被拆分为 3 个数据报。然后，这 3 个数据报经路由器 R3 转发，通过第二个以太网到达目的主机。这种情形在图 3-17 中给出。这个图还强调两个重点：

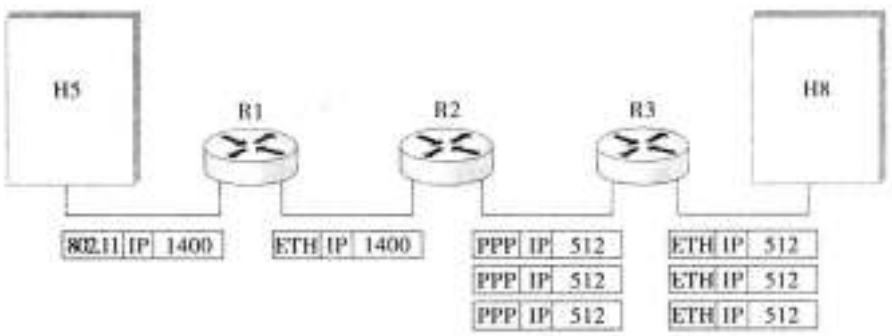


图 3-17 IP 数据报经过图 3-14 所示的一系列物理网络

1）每个分片本身就是一个在一系列物理网络上传送的独立的 IP 数据报，与其他分片无关。

2）每个 IP 数据报在它经过的每个物理网络上都要重新封装。

通过图 3-18 所示的每个数据报的首部字段可详细了解分片过程。图 3-18a 中未分片的分组有 1400 字节的数据和 20 字节的 IP 首部。当分组到达 MTU 为 532 字节的路由器 R2 时必须分片。一个 532 字节的 MTU 中，在 20 字节的 IP 首部之后留有 512 字节的数据，因此第一个分片包括 512 字节数据。路由器在 Flags（标志）字段中设置 M 位（见图 3-16），意思是后面还有其他的分片，并设置 Offset（偏移量）为 0，因为这个分片包含原始数据报的第一部分。第二个分片携带的数据从原始数据报的第 513 个字节开始，因此这个首部中的 Offset 字段设置为 64，即 512÷8。为什么除 8 呢？因为 IP 的设计者认定分片应该发生在 8 字节边界的地方，即 Offset 字段以 8 字节而不是以字节为单位计数。（我们把它作为练习，请你指出为什么要这样设计。）第三个分片中包含最后的 376 字节数据，偏移量为 2×512÷8＝128。由于这是最后一个分片，所以不设置 M 位。

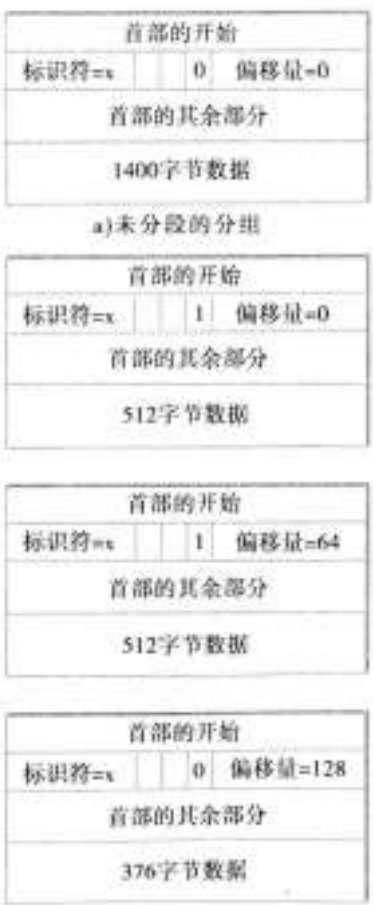


图 3-18 IP 分片中所使用的首部字段

注意，分片过程是以这样一种方式完成的，如果分片到达另一个有更小 MTU 的网络，那么分片过程可以重复地进行。分片产生更小的有效 IP 数据报，在接收方易于重组成原始的数据报，而与到达顺序无关。重组在

接收主机上而不在每个路由器上进行。

IP 重组绝非一个简单的过程。例如，即使一个分片丢失，接收方仍将试图重组数据报，而最终将放弃并不得不回收失败的重组中所使用的资源。[1] 由于这个原因，通常认为应该避免 IP 分片。现在，我们强烈鼓励主机执行“路径 MTU 发现”，这一过程通过发送足够小的分组，使其能通过从发送方到接收方路径上 MTU 最小的链路，从而避免分片。

3.2.3 全局地址

在上面 IP 服务模型的讨论中，我们曾提到过它提供的编址方案。毕竟，如果想把数据发送到任一网络的任一主机，就需要有一种方式来识别所有主机。因此，我们需要一个全局编址方案，其中任何两台主机的地址都不能相同。全局唯一性是一个编址方案所应提供的首要特性。[2]

以太网地址是全局唯一的，但仅此一点还不能满足一个大型互联网的编址方案要求。以太网的地址也是扁平的（flat），也就是说它们没有结构，且几乎不为路由协议提供线索。事实上，以太网地址的确有一个用于分配（assignment）目的的结构：前 24 比特标明制作者。但因为这个结构对于网络拓扑没有影响，所以它没有给路由协议提供任何有用信息。相比之下，IP 地址是分层的（hierarchical），即它们由对应于互联网某种层次结构的几个部分构成。更确切地说，IP 地址包括两部分：网络（network）部分和主机（host）部分。对于由多个相互连接的网络组成的互联网而言，这是一个非常合理的结构。IP 地址的网络部分指明主机连接到哪个网络，所有连到同一网络主机的 IP 地址的网络部分相同。IP 地址的主机部分唯一地标识特定网络中的每台主机。这样，在如图 3-14 所示的简单网络中，网络 1 中主机的地址有相同的网络部分和不同的主机部分。

注意图 3-14 中路由器被连接到两个网络。它们在每个网络上都需要一个地址，一个接口分配一个地址。例如，位于无线网和以太网之间的路由器 R1，在到无线网的接口上有一个 IP 地址，且与无线网中主机的网络部分相同，而到以太网接口上的 IP 地址则与以太网主机的网络部分相同。因此，记住路由器可以用具有两个网络接口的一台主机的方式实现，更确切地说，应把 IP 地址视为属于接口而不是属于主机。

现在，这些分层性地址看起来像什么？与其他某些分层地址格式不同的是，所有地址的两部分长度都不相同。如图 3-19 所示，IP 地址分为三种不同的类型，每种类型都定义不同长度的网络部分和主机部分。（还有将在 4.2 节中讨论的说明多播组的 D 类地址，而 E 类地址现在已经不再使用。）所有情况下，地址均为 32 位长。



图 3-19 IP 地址

IP 地址的类型由最高位的几个比特标识。如果第 1 位是 0，即为 A 类地址。如果第 1 位是 1、第 2 位是 0，则为 B 类地址。如果前 2 位是 1 而第 3 位是 0，则为 C

[1] 在第 8 章中我们将会看到，使主机占用不必要的资源可能会导致拒绝服务攻击。

[2] 不过，现代互联网中不再强调全局地址，理由涉及很多方面，详见 4.1 节。

类地址。这样，对大约40亿个可能的IP地址来说，1/2是A类，1/4是B类，1/8是C类。每一类都分配一定数目的位给地址的网络部分，而其余的留给主机部分。A类网络IP地址的网络部分有7位，主机部分有24位，意味着只能有126个A类网络（0和127保留），但每一类都能容纳最多$2^{24}-2$（大约1 600万）台主机（同样，有两个保留值）。B类地址分配14位给网络部分、16位给主机部分，意味着每个B类网络能容纳65 534台主机。最后，C类地址只将8位给主机部分、其余21位均为网络部分，所以，C类网络只可以有256个唯一的主机标识符，意味着只能连接254台主机（预留一台主机标识255用于广播，0是非法主机号）。然而，此编址方案支持2^{21}个C类网络。

表面上看，这种编址方案有着很大的灵活性，使得众多不同规模的网络可以非常有效地共存。最初的思想是因特网将包含一小部分广域网（A类网络）、相当数量站点（校园）规模的网络（B类网络）和大量局域网（C类网络）。然而，正如我们看到的，现在的编制方案并没有足够的灵活性。当前，IP地址通常是“无类的”，具体细节接下来解释。

在我们了解如何使用IP地址之前，先来看一些实际的事情，比如如何将IP地址写下来。按照惯例，IP地址被写成4个以点相隔的十进制（decimal）整数。每个整数表示一个十进制数值，包含在地址的一个字节中，从最高位开始。例如，键入这个句子的计算机的地址是171.69.210.245。

重要的是不要将IP地址与因特网的域名相混淆，域名也是分层的。域名是以点分隔的ASCII码字符串，如cs.princeton.edu，我们将在9.3.1节中讨论它。关于IP地址的重要事项是IP分组首部携带的信息，IP路由器正是使用其中的地址做出转发决定的。

3.2.4 IP数据报转发

我们现在来看互联网中IP路由器转发数据报的基本机制。回顾3.1节，转发（forwarding）是将从输入端口得到的一个分组通过适当输出端口发送出去的过程，而路由（routing）是建造一张表以决定分组的正确输出端口的过程。本节讨论的重点是转发，路由将在3.3节中讨论。

当我们讨论IP数据报的转发时，要记住以下几个要点：

- 每个IP数据报均包含目的主机的IP地址。
- IP地址的网络部分唯一地标识作为更大的因特网一部分的一个物理网络。
- 在连接到同一物理网络上的所有主机和路由器的地址中，其网络部分相同，因此可以通过在网络上发送帧而彼此通信。
- 每个作为因特网一部分的物理网络，按照定义至少有一台路由器同时至少连接到一个其他的物理网络，这台路由器可以与其中任一网络的主机或路由器交换分组。

因此，转发IP数据报可以按以下方法处理。一个数据报从源主机发往目的主机，沿途可能经过多台路由器。任何一个节点，无论是主机还是路由器，首先试图确定自己是否与目的主机连接在同一个物理网络上。为了做到这一点，它比较目的地址的网络部分和它的每一个网络接口地址的网络部分。（主机通常有一个接口，而路由器通常有两个或多个接口，因为它们一般连接在两个或多个网络上。）如果匹配，那么就意味着目的地址与接口位于同一网络中，分组可以在此网络中直接传送。3.2.6节将解释这个过程的一些细节。

如果没有被作为目的节点而连接到同一个物理网络上，就需要将数据报发往路由器。

一般而言，每个节点都有多台路由器可供选择，因此它需选择一个最佳的或至少是有较好机会使数据报更接近目标的路由器。所选择的路由器称为下一跳（next hop）路由器。该路由器通过查询它自己的转发表找到正确的下一跳。从概念上讲，转发表就是一个包括（NetworkNum，NextHop）（网络号，下一跳）对的表。（下面我们将看到，实际上转发表还包括和下一跳有关的额外信息。）通常还有一台默认路由器，当表上的条目与目标网络号都不匹配时将会使用默认路由器。对于主机来说，仅有一台默认路由器是完全可以接受的，它意味着当目的主机与发送主机不在同一个物理网络时所有数据报将通过默认路由器发出。

我们可将数据报转发算法描述如下：

```
if(目的节点的 NetworkNum = 我的一个接口的 NetworkNum)then
    经过那个接口传送分组到目的节点
else
    if(目的节点的 NetworkNum 在我的转发表中) then
        传送分组到 NextHop 路由器
    else
        传送分组到默认路由器
```

对于只有一个接口且转发表中只有一台默认路由器的主机来说，算法可以简化如下：

```
if(目的节点的 NetworkNum = 我的网络号)then
    直接传送分组到目的节点
else
    传送分组到默认路由器
```

我们来看这个算法在图 3-14 所示的互联网中是如何工作。首先，假设 H1 要发送一个数据报到 H2。由于它们在同一个物理网络中，H1 和 H2 的 IP 地址有相同的网络部分，于是 H1 推断可以在以太网上直接向 H2 发送数据报。这里有一个问题需要解决，即 H1 如何找到 H2 的正确以太网地址——这是 3.2.6 节描述的地址解析机制。

相关主题

网桥、交换机和路由器

网桥、交换机和路由器很容易被混淆。对于这样的混淆，很可能的一个原因是在某一层上，它们都是从一条链路把消息转发到另一条链路。人们根据分层对它们做出区分：网桥是链路层节点（它们在链路间转发帧以实现扩展的 LAN），交换机是网络层节点（它们在链路间转发分组以实现分组交换网络），路由器是互联网层节点（它们在网络之间转发数据报以实现互联网）。

网桥和交换机之间的区别正在迅速地消失。比如，我们已经可以看到多口网桥通常称为以太网交换机或者局域网交换机。正因如此，网桥和交换机通常编组为“第二层设备”。这里第二层是指“在物理层之上，在互联网层之下”。

历史上，局域网交换机（或网桥）与广域网交换机（比如基于 ATM 以及帧中继）之间曾有一些重要区别。局域网交换机传统上基于生成树算法，而广域网交换机通常运行路由协议，从而允许每台交换机学习整个网络的拓扑。这是一个很重要的区别，因为如果能了解整个网络的拓扑，就允许交换机辨别不同的路由器。相反，生成树算法只被限制在转发消息的单一树上。另外，生成树方法无法扩展。同样，这个区别受到的威胁是广域路由协议开始调整其工作方式并进入局域网交换机。

交换机和路由器之间又有什么区别？从内部技术上讲，两者看起来很相似（在讲路由器实现时将阐述这一点），关键区别是它们转发分组的种类：路由器转发 IP 数据报，而交换机转发第二层分组（以太网帧或者 ATM 信元）。

交换机构造的 ATM 网络和路由器构造的因特网之间的一个很大的区别是，因特网可以适应异构性，而 ATM 只包含同构链路。对异构性的支持是因特网得以广泛使用的关键原因之一。这也使得 IP 虚拟运行于其他网络协议之上（包括 ATM 和以太网），导致这些协议被看作第二层技术。

现假设 H5 要发送一个数据报给 H8。由于不在同一个物理网络中，它们有不同的网络号，因此 H5 推断需将数据报发送给路由器。R1 是唯一选择：默认路由器。所以 H5 通过无线网将数据报发给 R1。类似地，R1 知道不能直接将数据报发给 H8，因为 R1 的任何一个接口都不和 H8 在同一个物理网络中。假设 R1 的默认路由器是 R2，R1 通过以太网将数据报发给 R2。假设 R2 的转发表如表 3-5 所示，它找到 H8 的网络号（网络 4），并转发数据报至 R3。最后，由于 R3 与 H8 处于同一个物理网络中，R3 将数据报直接发送至 H8。

表 3-5 图 3-14 中路由器 R2 的转发表例子

网络号	下一跳
1	R1
4	R3

注意转发表中可能包含直接相连的网络的信息。例如，我们可以将路由器 R2 的网络接口标记为点到点链路（网络 3）中的接口 0 和以太网（网络 2）中的接口 1。那么 R2 将得到如表 3-6 所示的转发表。

表 3-6 图 3-14 中路由器 R2 的完整转发表

网络数	下一跳
1	R1
2	接口 1
3	接口 0
4	R3

因此，对于 R2 在一个分组中遇到的任何网络号，它都知道该怎么做。如果网络直接与 R2 相连，这时分组可通过网络被送达目的地；如果网络可经某个下一跳路由器到达，此时 R2 可经过所连接的网络到达此路由器。在任何一种情况下，R2 都使用下面描述的 ARP 来找到分组将要发送到的下一节点的 MAC 地址。

R2 使用的转发表很简单，可以用手工方式配置。然而，通常这些表都复杂得多，需要运行路由协议来建造，3.3 节描述了这样一个路由协议。实际中还需注意，网络号通常更长（如 128.96）。

我们现在能看到分层编址——将地址分为网络部分和主机部分——如何提高大型网络的可扩展性。路由器现在包含的转发表只列出一组网络号，而不是网络中的所有节点。在我们简单的例子中，R2 能够在一个 4 条记录的表中存储到达网络中所有主机（这里是 8 台主机）所需的信息。即使每个物理网络有 100 台主机，R2 也只需同样的 4 条记录。这是实现可扩展性的良好的第一步（虽然绝不是最后一步）。

结论 这里说明建造可扩展的网络的最重要原则之一：为达到可扩展性，需要减少存储在每个节点上以及在节点之间交换的信息量。最常用的方法是分层聚合（hierarchical aggregation）。IP 采用两层的层次结构，网络在上层，节点在下层。我们通过让路由器只处理如何到达正确的网络来聚合信息，将路由器传送一个数据报到给定网络中任一节点所需要的信息表示为一条聚合信息。

3.2.5 子网划分和无类地址

IP 地址的最初目的是希望其网络部分能够唯一明确确定一个物理网络。然而这种方法有两个缺点。假设一个大型校园中有很多内部网络，并决定连接到因特网上。对每个网络来说，不管多么小，都至少需要一个 C 类网络地址。对某些多于 255 台主机的网络来说，甚至需要一个 B 类地址。看起来这也许不是一个大问题，事实上在因特网的最初构想中也确实不是什么大问题，但是我们只有为数有限的网络号，并且 B 类网络地址比 C 类少得多。对 B 类地址往往会有很高的需求，因为你不知道网络是否会扩展到超过 255 个节点，因此一开始就使用 B 类地址比在超出一个 C 类网络的空间时再对每台主机重新编号要容易得多。这里我们关注的问题是地址分配的低效率：只有两个节点的网络使用一个完整的 C 类网络地址，那么就浪费了 253 个有用地址；一个只有稍多于 255 台主机的 B 类网络，则浪费 64 000 个以上的地址。

如果给每个物理网络分配一个网络号，耗尽 IP 地址空间的速度就比我们想象的快得多。虽然我们需要连接超过 40 亿台主机才能用完所有合法的地址，但是连接 2^{14}（约 16 000）个 B 类网络就能用完 B 类地址空间。因此，我们希望找到一个更有效地使用网络号的方法。

当你考虑路由时，就会发现分配多个网络号有另一个明显缺点。回忆一下，参与路由协议的节点中存储的状态数量是与其他节点数成正比的，同时，一个互联网中的路由包括建立转发表，这张表告诉路由器如何到达不同的网络。这样，使用的网络号越多，转发表就越大。大转发表增加了路由器的开销，并且在使用同一种技术时，大表中的查找速度比小表中的查找速度慢，因此降低了路由器的性能。这是另一个需要仔细分配网络号的原因。

划分子网（subnetting）提供了减少分配网络号总数的第一步。其思想是只用一个网络号，把具有这个网络号的 IP 地址分配给多个物理网络，这些物理网络叫作子网（subnet）。为使这种做法行之有效，需要做几件事：第一，子网应当彼此离得很近。这是因为从因特网远处的一个点上看，它们像是一个单一网络，只有一个网络号。这意味着一台路由器将只能选择一个路由来到达任何子网，因此它们最好在同一方向上。子网的最佳应用环境是在一个有多个物理网络的大型校园或公司中。从校园之外到达校园内任一子网时，你只需知道校园网连在因特网的什么地方。这通常是一个点，因此在转发表中只需一条记录就够了。即使校园网中有多个点连接在因特网的其余部分上，也最好先知道如何到达校园网的一个点。

在多个网络当中共享一个网络号的机制涉及使用子网掩码（subnet mask）配置每个子网中的所有节点。使用简单 IP 地址时，同一网络中的所有主机必须有相同的网络号。子网掩码使我们可以引入一个子网号（subnet number），同一物理网络中的所有主机将会有相同的子网号，这意味着主机可能处于不同的物理网络中，但共享一个网络号。这个概念的解释见图 3-20。

划分子网对主机来说意味着现在它是由 IP 地址和它所连接子网的掩码来配置的。例如，图 3-21 中的主机 H1 配置的地址为 128.96.34.15，子网掩码是 255.255.255.128。（在一个给定的子网中，所有主机都配置相同的掩码，即每个子网只有一个掩码。）将这两个数的按位与运算结果定义为此主机和同一子网内的所有其他主机的子网号。在这种情况

下，128.96.34.15 和 255.255.255.128 按位做与运算等于 128.96.34.0，这就是此图中最上端子网的子网号。



图 3-20 子网地址

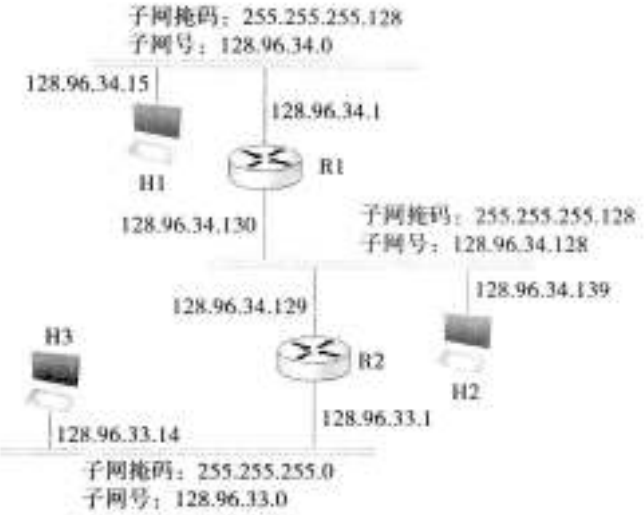


图 3-21 划分子网的例子

当主机要发送一个分组到一个特定的 IP 地址时，它所做的第一件事就是用它自己的子网掩码与目标 IP 地址做按位与运算。如果结果等于发送主机的子网号，那么它就得知目的主机在同一子网内，分组可以在子网中直接传送。如果结果不等于发送主机的子网号，就需要把分组发送给一台路由器以便转发到另一子网。例如，如果 H1 向 H2 发送，那么 H1 将它的子网掩码（255.255.255.128）和 H2 的地址（128.96.34.139）进行按位与运算得到 128.96.34.128。这与 H1 的子网号 128.96.34.0 不匹配，因此 H1 得知 H2 在另一个不同的子网中。由于 H1 不能直接在子网上传递分组给 H2，所以它将分组传送给它的默认路由器 R1。

当我们引入子网的概念后，路由器的转发表也发生了一点变化。回忆一下，我们刚才有一个转发表，这个表由成对形式的（NetworkNum，NextHop）{（网络号，下一跳）} 的记录组成。为了支持划分子网，表中各记录的形式现在必须保存（SubnetNum，SubnetMask，NextHop）{（子网号，子网掩码，下一跳）} 形式的记录。为了在表中找到正确的记录，路由器将分组的目的地址与每个记录的子网掩码依次进行按位与运算。如果结果与某一记录的子网号相匹配，那么这就是要使用的记录，然后将分组转发到指定的下一跳路由器。在图 3-21 所示的网络中，路由器 R1 的记录如表 3-7 所示。

表 3-7 图 3-21 中划分子网的转发表

子 网 号	子网掩码	下一跳
128.96.34.0	255.255.255.128	接口 0
128.96.34.128	255.255.255.128	接口 1
128.96.33.0	255.255.255.0	R2

继续看从 H1 向 H2 传送数据报的例子，R1 将 H2 的地址（128.96.34.139）与第一个记录的子网掩码（255.255.255.128）进行按位与运算，并将结果（128.96.34.128）与这一记录的网络号（128.96.34.0）进行比较。由于不匹配，所以继续比较下一记录。这一次出现匹配，所以 R1 通过接口 1 将数据报送往 H2，接口 1 与 H2 在同一网络中。

我们现在可将数据报转发算法描述如下：

D= 目标 IP 地址

```
对于每个转发表入口(SubnetNumber, SubnetMask, NextHop)
    D1= SubnetMask & D
    if D1= SubnetNumber
            if NextHop 是一个接口
                传送数据报到目标
            else
        传送数据报到 NextHop(一台路由器)
```

尽管这个例子中没有给出，但表中通常包含默认路由器，可在找不到明确的匹配时使用。需要指出，单纯地实现这种算法效率是很低的，将会重复地进行目的地址与子网掩码的按位与运算（而子网掩码可能并非每次都不相同）以及线性表搜索。

有关子网划分的一个重要结论是，互联网的不同部分将看到不同的东西。从我们假想的校园外部，路由器看到的是一个单一网络。在以上提到的例子中，校园外的路由器把图3-21的网络集合只看作网络128.96，并且转发表中有一条记录告诉它们如何到达这个网络。然而，校园内的路由器需要能够通过选择路由将分组传到正确的子网。因此，并不是互联网的所有部分都能看到同样的路由信息。这是一个聚合（aggregation）路由信息的例子，这些聚合信息是扩展路由系统的基础。后文将会介绍聚合的其他应用实例。

1. 无类地址

子网划分有一个同类，有时也称为超网（super netting），但通常称为无类域间路由（Classless Interdomain Routing，CIDR，发音为"cider"）。CIDR用子网划分的思路处理地址类划分从而得到其逻辑结论。那么为什么子网无法满足需要呢？本质上讲，子网只允许我们把一些类别的地址在多个子网内分拆，而CIDR允许我们将不同类的地址组成一个单独的"超网"。这样进一步解决了地址空间不足的问题，且可防止路由系统过载。

让我们看一下地址空间的效率与可扩展性问题是如何共存的。假设某公司的网络由256台主机构成。此时，要分配太多的C类地址，所以你会希望绑定一个B类地址。然而，将一个可以容纳65 535个地址的主干地址空间分给256台主机，其占用率仅仅是256/65 535=0.39%。即使划分子网可以使我们更细致地分配地址，但一个无法回避的事实是任何一个超过255台主机的组织，或现在没有但最终可能达到如此多主机数的组织仍然需要一个B类地址。

我们面对此类问题的第一种处理方法是拒绝给任何组织B类地址（除非有证据显示他们的需求接近64K个地址），而是给他们合适的C类地址以满足主机数目要求。由于我们现在每次以256个地址为单位分配空间，所以能够更精确地与组织消耗的地址空间数相匹配。对于任何一个至少有256台主机的组织来说，我们可以保证地址的利用率至少达到50%，且通常更高。

然而，这种解决方法会引起一个严重的问题：对路由器超量存储的需求。如果一个站点被分配了16个C类网络号，就意味着每个因特网主干网路由器的路由表中需要16条记录才能将分组传送到该站点，即使到这些网络的路径相同也是如此。如果我们给这个站点分配一个B类地址，同样的路由信息就可以存储为一条表记录。然而，地址分配的效率将只有16×255/65 536=6.2%。

因此，CIDR尝试在减少一台路由器所需知道的路由数的愿望与有效分配地址的需求之间取得平衡。为做到这一点，CIDR帮助我们聚合（aggregate）路由。就是说，它让我们仅使用转发表中的一条记录来知道如何到达多个不同的网络。你可能已经从名字猜到，

它通过打破地址分类之间的严格界限来做到这一点。为了理解 CIDR 如何工作，我们假设某组织有 16 个 C 类网络号。我们可以分配一块连续的（contiguous）C 类地址而不是随机地分配 16 个地址。假设我们分配的 C 类网络号为 192.4.16～192.4.31。可以看出，在此范围内，所有地址的高 20 位是一样的（11000000000000010000）。这样，我们就有效建造了一个 20 位的网络号，它所能支持的主机数界于 B 类网络号与 C 类网络号之间。换句话说，我们既得到以小于 B 类网络的块分配地址的高效率，又得到可在转发表中使用的单个网络前缀。可以看出，要让此方案正常工作，我们需要分发具有相同前缀的 C 类地址块，这意味着每块都必定包含数目为 2 的幂次方的 C 类网络。

CIDR 需要一种新型标注或者用已知的前缀（prefix）来表示网络号，因为前缀可以任意长。通常的做法是在前缀后放置一个“/X”，其中“X”是前缀的位长度。所以，在前面的例子里，在从 192.4.16～192.4.31 之间的网络中，20 位的前缀可以表示为 192.4.16/20。相反，如果需要表示单个 C 类网络号，因为其是 24 位长，所以记作 192.4.16/24。当前，随着 CIDR 成为规范，可以听到更多的人说起“/24”前缀表示的 C 类网络。注意，按这种方法表示网络地址类似于划分子网中用过的（mask，value）（〈掩码，值〉）方法，只要掩码是由从最高位开始的连续位组成即可（实际上这类问题都是这样）。

以我们刚刚给出的这种方法聚合路由的能力只是第一步。想象一个因特网服务提供商网络，它的基本任务是为大量公司和校园（客户）提供因特网连接。如果我们按照这种方法来给公司分配网络号，即所有连在此网络上的不同公司都共享一个公共地址前缀，那么我们能得到更大程度上的路由聚合。考虑图 3-22 所示的例子，提供商网络服务的 8 个客户被分配相邻的 24 位网络前缀，该前缀的前 21 位是相同的。由于所有客户均可经同一个提供商网络到达，因此可以通过通知它们共享的公用 21 位前缀来向它们通知一条路由，即使没有分配全部 24 位前缀也可以采用这种通知方式，只要提供商最终能够为客户分配那些前缀。做到这一点的一种方法是提前给提供商网络分配一部分地址空间，然后让网络提供商从此空间中给它的用户按需分配地址。注意，与这个简单的例子不同的是，没有必要使所有用户前缀都具有相同的长度。

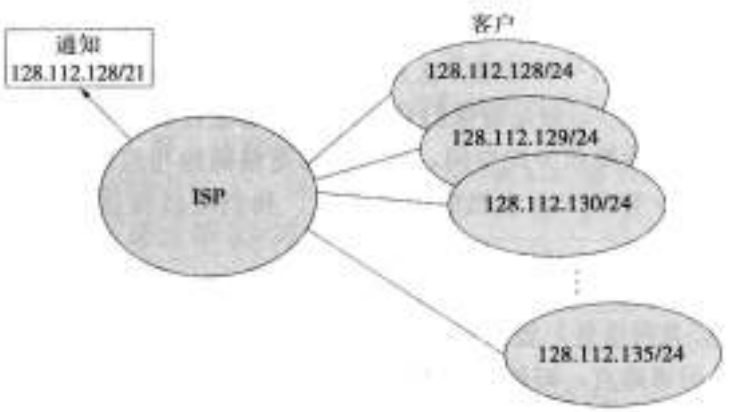


图 3-22 用 CIDR 进行路由聚合

2. IP 转发再讨论

至此，在 IP 转发讨论中，我们假设能够找到一个分组中的网络号，然后在转发表中查找该网络号。然而，现在我们引入了 CIDR，因此需要重新考虑这个假设。CIDR 意味

着前缀可以是2～32位的任意长度。而且，有时转发表中的前缀也可能重叠，也就是说，某个地址可能与不止一个前缀匹配。例如，在一台路由器的转发表中我们可以找到171.69（一个16位的前缀）和171.69.10（一个24位的前缀）。在这种情况下，一个目标为171.69.10.5的分组显然与两个前缀都匹配。处理这种情况的规则基于“最长匹配”原则，就是说，分组与最长的前缀匹配，此例中是171.69.10。另一方面，一个目标是171.69.20.5的分组将匹配171.69而不是171.69.10，如果与路由表中其他记录都不匹配，那么171.69将是最长的匹配。

在一个IP地址和转发表中的可变长前缀之间有效地寻找最长匹配的任务，近些年已经成为了一个富有成效的研究领域，本章的“扩展阅读”中给出了一些参考资料。最著名的算法使用了一种叫作PATRICIA树的方法，实际上它比CIDR得到了更好的研究。

3.2.6 地址转换（ARP）

上一节我们讨论了如何使IP数据报到达正确的物理网络，但是掩饰了数据报如何到达该网络上某一特定主机或路由器的问题。主要的问题是IP数据报包含IP地址，但是你想要将数据报传送到的主机或路由器上的物理接口硬件只理解特定网络的编址方案。这样，我们就需要将IP地址转换为这个网络所能理解的链路层地址（如一个48位的以太网地址）。然后，我们可以把数据报封装到包含该链路层地址的帧中，并发往最终目的地或发往一台可将数据报传向最终目的地的路由器。

将IP地址映射为物理网络地址的一个简单的方法是将主机的物理地址编码在IP地址的主机部分中。例如，一台主机的物理地址是00100001 01001001（前一个字节是十进制数33，后一个字节是十进制数81），它的IP地址是128.96.33.81。尽管这种解决办法已在一些网络中使用，但它仅限于那些网络物理地址不超过16位长的情况。在C类网络中，物理地址只有8位长。显然，这种方法不适用于48位长的以太网地址。

一个更普遍的解决方案是为每台主机保留一张地址对照表，也就是说，该表将IP地址映射为物理地址。虽然这个表可以由系统管理员集中管理并复制到网络中的各个主机上，但更好的方法是让每台主机通过网络动态地得到表的内容。这种方法可以用地址解析协议（Address Resolution Protocol，ARP）来实现。ARP的目标是使网络上的每台主机都建立一张IP地址和链路层地址间的映射表。由于这些映射可能会随时间而改变（例如，由于一台主机的以太网卡出现故障，而被另一个有新地址的卡替换掉），所以表中的条目周期性地超时并被删除。这种情况每15分钟发生一次。当前存储在主机上的映射集合称为ARP高速缓存或ARP表。

ARP充分利用很多链路层网络技术（如以太网）都支持广播这一优点。如果一台主机要发送一个数据报给已知为同一网络内的另一台主机（或路由器）（即发送和接收节点有同样的网络号），那么它首先检查缓存中的映射。如果映射不存在，它就需要调用网络上的地址解析协议。该调用是通过向网络广播一个ARP查询来实现的，查询中包含询问的IP地址（目标IP地址）。每台主机接收到这个查询并检查是否与自己的IP地址匹配。如果匹配，该主机发送一个包含它的链路层地址的应答信息给发出查询的源主机。源主机将此应答中包含的信息添加到它的ARP表中。

查询信息也包含发送主机的IP地址和链路层地址。这样，当一台主机广播一条查询信息时，网络上每台主机都会知道发送方的链路层地址和IP地址，并将此信息放入自己

的 ARP 表中。然而，并不是每台主机都把这一信息添加到 ARP 表中。如果某一主机的表中已经有了这台发送主机的条目，那么它将“刷新”这一条目，即重新设置时间长度直至丢弃这一条目。如果某主机是查询的目标，那么即使其表中没有发送主机的条目，它也把有关发送方的信息添加到自己的表中。这样做的原因是，源主机可能要向它发送一个应用级消息，并且它最终必须发一个应答或 ACK 给源主机，而这需要源主机的物理地址。如果某主机不是目标主机，并且在其 ARP 表中没有源主机的条目，那么它就不会将源主机的条目加入到自己的 ARP 表中。这是因为这台主机不需要源主机的链路层地址，没有必要在 ARP 表中保存这条信息。

图 3-23 给出了从 IP 地址到以太网地址映射的 ARP 分组格式。事实上，ARP 可以用于其他多种映射，主要区别在于地址长度。除了发送方和目标方的 IP 和链路层地址外，分组还包括：

- 一个 HardwareType（硬件类型）字段，说明物理网络的类型（如以太网）。
- 一个 ProtocolType（协议类型）字段，说明高层协议（如 IP）。
- HLen（“硬件”地址长度）和 PLen（“协议”地址长度）字段，分别说明链路层地址和高层协议地址的长度。
- 一个 Operation（操作）字段，说明是请求还是应答。
- 源硬件和目标硬件（以太网）地址和协议（IP）地址。



图 3-23 用于将 IP 地址和以太网地址进行映射的 ARP 分组格式

注意，ARP 进程的结果可以加在如表 4-1 所示的转发表中成为额外的一列。这样，例如，当 R2 需要转发一个分组到网络 2 时，它不仅能找到下一跳是 R1，而且能找到 MAC 地址，将其加在分组上并发往 R1。

结论 我们现在已经看到 IP 提供的处理异构性和可扩展性的基本机制。对于异构性问题，IP 首先定义一个尽力服务模型，对底层网络做最少的假设。更值得注意的是，这个服务模型基于不可靠的数据报。然后，IP 做两件重要的事情：提供一个公共的分组格式（分段和重组是使这个格式适应不同 MTU 网络的机制）；提供识别所有主机的全局地址空间（ARP 是使全局地址空间在具有不同物理地址编址方案的网络上运行的机制）。对于可扩展性问题，IP 使用层次化的聚合，减少转发分组所需的信息量。特别是，IP 地址被划分为网络和主机两部分，首先选择路由将分组发送到目的网络，然后再将分组传送给这个网络中正确的主机。

3.2.7 主机配置（DHCP）

在2.6节我们已经看到，以太网地址由制造商配置到网络适配器中，这样管理是为了保证这些地址是全球唯一的。显然，这是保证连接到一个以太网（包括扩展的LAN）的任意主机集合有唯一地址的充分条件。而且，唯一性是我们对以太网地址的全部要求。

与之相比，IP地址在一个给定的互联网中不仅必须是唯一的，而且必须反映互联网的结构。如上所述，它们包括网络部分和主机部分，且对于在同一个网络中的所有主机来说，网络部分必须相同。这样，IP地址不可能在主机制造时就一次配置好，因为那将意味着制造商知道哪台主机将要连到哪个网络，并且意味着一旦主机连接到某个网络，就再也不能移动至另一个网络上。因此，IP地址必须是可重新配置的。

除了IP地址之外，主机在开始发送分组之前还需要知道一些其他信息。其中最值得注意的是默认路由器的地址，即当分组的目的地址和发送主机不在同一网络时，主机能将分组发送到的地方。

大多数主机操作系统为系统管理员或用户提供一种方法，以手工方式配置主机所需的IP信息。然而，这样的人工配置有一些明显缺点。首先，在一个大型网络中直接配置所有主机工作量很大，尤其是当你意识到主机只有进行配置之后才能通过网络可达时。更重要的是，配置过程非常容易出错，因为它需要确保每台主机得到正确的网络号，并且任何两台主机不能有同样的IP地址。因此需要有自动配置方法，主要方法是使用动态主机配置协议（Dynamic Host Configuration Protocol，DHCP）。

DHCP依赖于DHCP服务器，DHCP服务器负责向主机提供配置信息。一个管理域中至少有一个DHCP服务器。在最简单的分层结构中，DHCP服务器的功能就像一个主机配置信息的中心库。例如，考虑在一个大公司的互联网中管理地址的问题。DHCP可以使网络管理员不必拿着一张地址清单和网络图走遍公司的每一台主机以手工方式配置它们。相反，每台主机的配置信息可以存储在DHCP服务器中，并当主机自举或与网络连接后，由每台主机自动获得。然而，管理员仍需选取每台主机接收的地址并将其存到服务器中。在这个模型中，每台主机的配置信息存储在一张表中，此表以某种形式的唯一客户标识作为索引，通常是硬件地址（如它的网络适配器的以太网地址）。

一种更成熟的DHCP用法使网络管理员不必为单个的主机分配地址。在这个模型中，DHCP服务器维护着一个由它按需分配给主机的可用地址的缓冲区。由于现在只需要给每个网络分配IP地址的范围（全部有相同的网络号），因此在很大程度上减少了管理员所必须做的配置数量。

由于DHCP的目标是使一台主机正常工作所需的人工配置量减至最小，如果每台主机都必须配置一个DHCP服务器地址的话，就达不到这个目的。因此，DHCP首先面临的就是找到服务器的问题。

为了与一个DHCP服务器相连，一台新自举或新连接的主机发送一条DHCPDISCOVER消息到一个特殊的IP地址（255.255.255.255）——广播地址。这意味着此条消息将被网络上的所有主机和路由器接收。（路由器不转发这样的分组到其他网络，以防止广播到整个因特网。）在最简单的情况下，这些节点中的某一个就是网络的DHCP服务器。然后该服务器应答产生这条发现消息的主机（所有其他的节点将忽略这条消息）。然而，并不是每个网络都需要一个DHCP服务器，因为这会潜在增加大量需要正确并一致

配置的服务器。因此，DHCP 使用中继代理（relay agent）的概念。每个网络上至少有一个中继代理，它只配置有一条信息：DHCP 服务器的 IP 地址。当中继代理接收到 DHCP-DISCOVER 消息时，它将这条消息单播到 DHCP 服务器并等待响应，然后将响应回传给发出请求的客户机。从一台主机中继一条消息到一个远程 DHCP 服务器的过程如图 3-24 所示。

图 3-25 给出了 DHCP 消息的格式。这条消息实际上是使用运行在 IP 之上的 UDP 协议（用户数据报协议）来发送的。下一章将讨论 UDP 的细节，这里，它所做的仅是提供一个多路分解键，表明“这是一个 DHCP 分组”。

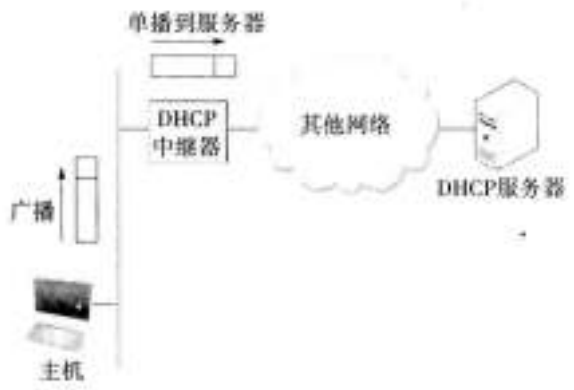


图 3-24 一个 DHCP 中继代理从一台主机收到一条广播 DHCPDISCOVER 消息并且发送一个单播 DHCPDISCOVER 给 DHCP 服务器

操作	硬件类型	硬件地址长度	跳数
传输ID			
客户端启动秒数		标志	
客户端前IP地址			
客户端IP地址（“你的”IP地址）			
服务器地址			
延迟代理地址			
客户端硬件地址（16字节）			
服务器名（64字节）			
文件（128字节）			
选项			

图 3-25 DHCP 分组格式

DHCP 协议起源于较早的 BOOTP 的协议，因此分组中的一些字段并不与主机配置严格相关。当客户端试图得到配置信息时，它把其硬件地址（如以太网地址）放到 chaddr（客户端硬件地址）字段中。DHCP 服务器填充 yiaddr（“你的” IP 地址）字段并发送给客户端作为回答。客户端使用的其他信息（如默认路由器等）包含在 options（选项）字段中。

显然，在 DHCP 为主机动态分配 IP 地址的情况下，主机不能无限期地保留地址，因为这样将导致服务器最终耗尽其地址空间。同时，不能依靠主机归还其地址，因为它可能已经崩溃、脱离网络或关闭。因此，DHCP 允许地址在一段时间内被租用。一旦租用期满，服务器就将地址回收。一个租用地址的主机，如果仍连在网络上且功能正常，显然需要定期重新租用地址。

结论 DHCP 阐明了可扩展性的一个重要方面：网络管理的可扩展性。可扩展性的讨论通常集中在使网络设备不要增长得太快这一点上，而关注网络管理复杂性的增长也十分重要。通过允许网络管理员给每个网络配置一个 IP 地址范围，而不必为每台主机配置一个 IP 地址，DHCP 改进了网络的可管理性。

注意，DHCP 也将更多的复杂性引入了网络管理，因为它使物理主机和 IP 地址之间的绑定更为动态化。这使得网络管理员的工作更加困难，例如，有必要定位一台出现故障的主机。

3.2.8 差错报告（ICMP）

下一个问题是因特网如何处理差错。当 IP 在数据报传送受阻而要将其丢弃时，例如，当路由器不知如何转发数据报或数据报的一个分片没有到达目的地时，它不能默默地归于失败。IP 总是和网际控制报文协议（Internet Control Message Protocol，ICMP）配置在一起，这个协议定义了当一台路由器或主机不能成功处理一个 IP 数据报时，向源主机发回的错误消息的集合。例如，ICMP 定义了目的主机不可达（可能是链路差错）、重组进程失败、TTL 为 0、IP 首部校验和出错等差错消息。

ICMP 也定义了路由器可发回源主机的少量控制信息。最有用的一种控制信息叫作 ICMP 重定向（ICMP-Redirect），它告知源主机有一条更好的到达目标的路由。ICMP 重定向用于以下情况。假设一台主机连接到一个连有两台路由器 R1 和 R2 的网络，主机使用 R1 作为默认路由器。若 R1 接到一个来自主机的数据报，而它根据转发表得知，对于某一特定目的地址来说 R2 会是更好的选择，于是它发回一个 ICMP 重定向给主机，指示主机对其余以该地址为目的地的数据报使用 R2。然后，主机将这个新路由加到转发表中。

ICMP 还为两个广泛使用的调试工具 ping 和 traceroute 提供了基础。ping 使用 ICMP 回应消息来判断一个节点是否可达且正在运行。traceroute 使用一种不太直接的技术来判断到目的节点的路径上的路由器集合，在本章最后的习题中有一道相关的习题。

3.2.9 虚拟网络和隧道

我们通过考虑一个也许你尚未预见但变得越来越重要的问题来结束对 IP 的介绍。至此，我们讨论的焦点是使不同网络上的节点能够以一种不受限制的方式进行通信。这通常是因特网的目标——人人都希望能将电子邮件发给任何人，一个新网站的创建者希望能有尽可能多的访问者。然而，更多的情况下需要更可控的连接。这种情况的一个重要例子就是虚拟专用网（Virtual Private network，VPN）。

术语 VPN 被大量地重复使用，并有各种各样的定义，但是，我们通过考查专用网的基本思想来直观地定义 VPN。拥有很多站点的公司经常通过从电话公司租用传输线路将网站互联起来建造专用网。在这样的网络中，通常出于安全性的考虑，通信被严格限制在本公司的网站间进行。为使专用网成为虚拟的（virtual），对于那些不与其他任何公司共享的租用传输线路，可用某种共享网络代替。一条虚电路（VC）是非常合适的租用线路替代品，因为它仍在公司的站点间提供逻辑的点到点连接。例如，如果公司 X 有一条从站点 A 到站点 B 的 VC，那么显然它能够在 A 和 B 之间发送分组。但是公司 Y 如果不事先建立它自己到站点 B 的虚电路，就不能将分组发送到 B，而在管理上可禁止建立这样的 VC 以防出现公司 X 和公司 Y 之间不希望的连接。

图 3-26a 给出了两个不同公司的专用网。在图 3-26b 中，它们都移植到一个虚电路网络中。真正的专用网中保持着有限的连接，但是由于专用网现在共享相同的传输设备和交换机，因此我们说建立了两个虚拟专用网。

在图 3-26 中，使用帧中继或 ATM 网络来提供站点之间的受控连接。使用 IP 网络——互联网——也可以提供类似功能的连接。然而，我们不能将不同公司的网站连到一个单一互联网上，因为那样将提供公司 X 与公司 Y 之间的连接，而这种连接是我们希望避免的。为解决这个问题，我们需要引入一个新的概念——IP 隧道（IP tunnel）。

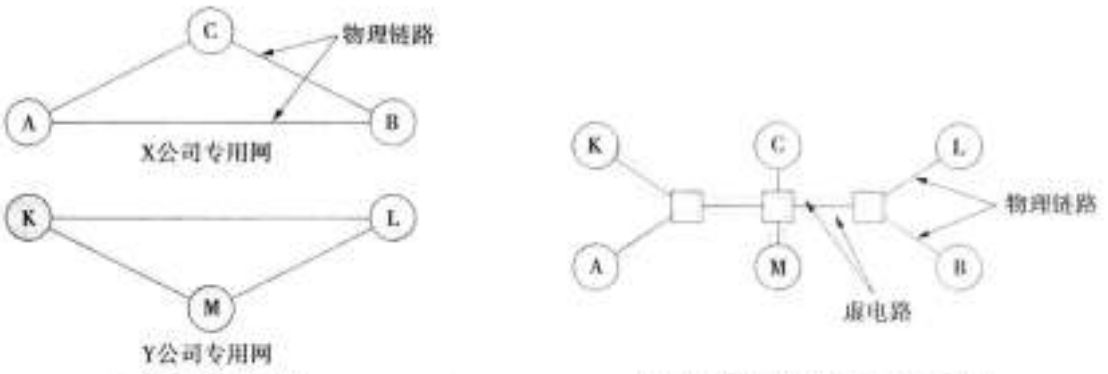


图 3-26 虚拟专用网络的例子

我们可以把 IP 隧道想象成一对节点间的一条虚拟的点到点链路，这对节点之间事实上可相隔任意多个网络。通过将隧道远端路由器的 IP 地址提供给虚链路，就可以在隧道入口处的路由器之间创建虚链路。无论何时当隧道入口处的路由器想要通过这个虚链路发送一个分组时，它就把分组封装在一个 IP 数据报中。IP 首部中的目的地址是隧道远端的路由器地址，而源地址是封装分组的路由器地址。

在隧道入口路由器的转发表中，这条虚链路更像一条普通的链路。例如，考虑图 3-27 中的网络。网络中已经配置了一条从 R1 到 R2 的隧道，并设置虚拟接口号为 0。R1 的转发表如表 3-8 所示。



图 3-27 贯穿互联网的隧道。18.5.0.1 是 R2 能够通过互联网到达 R1 的地址

表 3-8 图 3-27 中路由器 R1 的转发表

网络号	下一跳
1	接口 0
2	虚接口 0
默认	接口 1

R1 有两个物理接口。接口 0 连接网络 1；接口 1 连接一个大型互联网，并且是转发表中所有未确切说明的通信量的默认接口。另外，R1 有一个虚拟接口，即到隧道的接口。假设 R1 从网络 1 接收一个含有网络 2 中地址的分组，转发表认为这个分组应从虚拟接口 0 发出。为了从此接口发出分组，路由器接收这个分组，加上一个目标为 R2 的 IP 首部，然后就像刚刚接收到它一样继续转发这个分组。R2 的地址是 18.5.0.1，由于这个地址的网络号是 18，不是 1 或 2，因此目标是 R2 的分组将从默认接口转发到互联网。

一旦分组离开了R1，它就像一个目标是R2的普通IP分组一样被处理和转发。互联网中的所有路由器使用正常的方式来转发这个分组，直到它到达R2。当R2接收到这个分组时，发现它携带着自己的地址，因此将IP首部删除，并查看分组的有效载荷。它找到的是一个内部IP分组，其目的地址在网络2中。现在，R2像对待其他所有接收到的IP分组一样处理这个分组。由于R2直接与网络2相连，所以直接将该分组转发到网络2。图3-27给出了分组穿过网络时分组封装中的变化。

尽管R2充当隧道的末端点，但不影响它执行路由器的正常功能。例如，它可以接收非隧道的分组，但这些分组包含它们知道如何到达的网络的地址，然后它将按照正常方式进行转发。

也许你想知道为什么人们如此麻烦地建造一条隧道并且在分组穿过互联网时改变其封装。其中一个原因是安全性，我们将在第8章详细讨论。由于增加了加密，隧道可以变成一条穿过公用网络的专用链路。另一个原因是，R1和R2有一些其他参与网络所没有的能力，如多播路由。通过将这些路由器与隧道相连，我们可以建造一个虚拟的网络，其中所有有这种能力的路由器都好像直接相连一样。这事实上就是多播主干网（MBone）的建造过程，我们将在4.2节介绍它。建造隧道的第三个原因是传送非IP协议的分组穿过IP网络。只要隧道两端的路由器知道如何处理其他协议，IP隧道对它们来说就像一条能发送非IP分组的点到点链路。隧道也提供这样一种机制：即使一个分组被封装在隧道首部内的原首部表明它应传送到别处，我们也能强制把它传送到一个特定的地点。我们在4.4.2节考虑移动主机时将看到这种应用。因此，可以看出隧道是建造穿过互联网的虚拟链路的一种强有力的和非常普通的技术。

隧道技术也有缺点。第一，它增加了分组长度，对于短分组来说这可能意味着对带宽的严重浪费。其次，它也将影响隧道两端路由器的性能，因为它们将做一些超出正常转发的工作，如添加和删除隧道首部。最后，它还会增加一些管理实体的管理开销，这些实体负责设置隧道并确保它们能被路由协议正确处理。

3.3 路由

前面的讨论都假设交换机和路由器充分了解网络的拓扑结构，所以它们能为每个分组选择一个合适的输出端口。在虚电路中，路由仅仅是连接请求分组的问题，所有的后续分组都与连接请求分组经过同样的路径。在数据报网络（包括IP网络）中，每个分组都要进行路由。无论在哪种情况下，交换机或路由器都需要查看分组的目的地址，然后决定哪一个输出端口是将分组传送到那个目的地址的最好选择。就像我们在3.1.1节中看到的，交换机通过查询转发表来做这个决定。路由最基本的问题是，交换机和路由器如何获得转发表中的信息。

结论 我们重申转发（forwarding）和路由（routing）之间常被忽视的重要区别。转发过程包括：接收一个分组，查看它的目的地址，查询转发表，按表中决定的路径把分组转发出去。我们已经在前面一节见过几个转发的例子。路由是用于建立转发表的过程。我们还需注意，转发是在一个节点上本地执行的一个相对简单、定义良好的过程，而路由依赖于在网络发展过程中不断演进的、复杂的分布式算法。

虽然转发表（forwarding table）和路由表（routing table）这两个术语有时会交替使

用，但我们在此加以区别。在分组转发时使用转发表，因此转发表需要包含足够的信息来完成转发功能。这就意味着，转发表中的一行包括从网络号到发出接口的映射和一些MAC信息，如下一跳的以太网地址。另一方面，路由表作为建立转发表的前奏，是由路由算法建立的一张表，它通常包含从网络号到下一跳的映射。此外，它可能还包含如何得到这些信息的信息，以便路由器决定何时丢弃某些信息。

路由表和转发表是否是实际上独立的数据结构与实现时的选择有关，但保持它们彼此的独立性有很多原因。例如，构造转发表是为优化转发分组时查找网络号的过程，而优化路由表是为了计算拓扑结构的改变。有时，转发表甚至可以由某些特殊的硬件来实现，而路由表很少这样。表3-9给出每种表中的一行的示例。在这种情况下，路由表告诉我们网络前缀18/8是通过IP地址为171.69.245.10的下一跳路由器到达的，而转发表则包含如何确切转发一个分组到下一跳的信息：把它发送到MAC地址为8：0：2b：e4：b：1：2的接口号0。注意最后一条信息是由地址解析协议提供的。

表3-9 路由表和转发表一行的示例

a) 路由表		b) 转发表		
前缀/长度	下一跳	前缀/长度	接口	MAC地址
18/8	171.69.245.10	18/8	0	8:0:2b:e4:b:1:2

在了解路由的细节之前，需要回忆一下，每当我们试图为因特网建立一种机制时都应问的一个关键问题：这个解决方案可扩展吗？这一节描述的算法和协议将回答：不能。它们是为中等规模（即实际少于100个节点）的网络设计的。然而，我们描述的解决方案的确是今天因特网使用的层次性路由基础结构的一个构件。特别地，本节描述的协议统称为域内（intradomain）路由协议，或内部网关协议（interior gateway protocol，IGP）。为了理解这些术语，我们需要定义路由域（domain），路由域的一个不错的定义就是一个互联网，其中所有路由器都处于统一管理控制之下（例如，一所大学的校园网或一个因特网服务提供商的网络）。定义之间的相关性将在下一节讨论域间（interdomain）路由协议时变得更加清楚。至此，需要记住的一件重要的事情是我们是在中小型网络的范围内考虑路由问题，而不是因特网那么大的网络。

3.3.1 用图表示网络

路由本质上是图论问题。图3-28给出一个表示网络的图。图中标有A～F的节点可以是主机、交换机、路由器或网络。在开始讨论时，我们专注于考虑节点都是路由器的情况。图中的边对应于网络中的链路，每条边都有一个相应的开销（cost），表示希望通过这段链路发送的通信量。3.3.4节将讨论如何赋予边上的开销。[⊖]

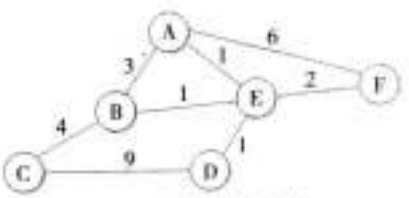


图3-28 用图表示网络

路由最基本的问题就是找出任意两个节点之间开销最小的路径，一条路径的开销等于组成这条路径的所有边上的开销之和。对于像图3-28中那样的简单网络，你

⊖ 在本章通篇采用的这个网络（图）例子里，我们采用无向边并为每条边分配一个开销。这的确是一种简化，更精确的方式是采用有向边，表明每个节点间都有一对边，它们都在各自的方向上且有自己的边开销。

可以想象计算出所有的最短路径，并将它们放入每一节点的某个非易失性存储器中。但这种静态方法有以下几个缺点：

- 它不处理节点或链路故障。
- 它不考虑新的节点或链路的增加。
- 它意味着边的开销不能改变，尽管我们可能有理由希望给一条重负载的链路临时指定一个高开销。

由于这些原因，在大多数实际的网络中，路由由运行在节点间的路由协议来实现。这些协议提供了一种分布式的、动态的方法来解决在链路和节点出现故障时寻找最小开销路径以及改变边上开销的问题。注意分布式（distributed）这个词，很难使集中式解决方案成为可扩展的，因此所有被广泛使用的路由协议都采用分布式算法。[⊖]

路由算法的分布式特性是人们对这个领域进行大量研究和开发的主要原因之一——在使分布式算法更好地工作方面还面临许多挑战。例如，分布式算法会产生这样的可能：在同一时刻，两台路由器对于到某个目的地的最短路径有不同的判断。事实上，每台路由器可能会认为另一个路由器到目标更近些，从而决定将分组发往另一台路由器。显然，这样的分组将会陷入循环，直到这两台路由器之间的矛盾得以解决，而且越早解决越好。这只是路由协议必须解决的问题中的一个例子。

在开始分析之前，我们假设已知网络中边的开销。下面我们将研究两类主要的路由协议：*距离向量*（distance vector）和*链路状态*（link state）。在 3.3.4 节我们回到以一种有针对性的方式计算边上开销的问题。

3.3.2 距离向量（RIP）[⊜]

距离向量算法的内在思想正如它的名字所揭示的那样[⊝]：每个节点构造一个包含到所有其他节点“距离”（开销）的一维数组（一个向量），并将这个向量分发给它的邻接点。对距离向量路由所作的初始假设是每个节点都知道到其直接邻接点的链路开销，到不相邻节点的链路开销被指定为无穷大。

要明白距离向量路由算法如何工作，考虑如图 3-29 所描述的例子是最容易的。在这个例子中，每条链路的开销设为 1，所以开销最小的路径就是包含跳数最少的路径（因为所有边都具有相同的开销，我们在图中没有标出开销）。我们可以将每个节点到所有其他节点的距离信息表示为一个表，如表 3-10 所示。注意，每个节点只知道表中一行的信息（左列标有其名字的那一行）。网络中任何一个节点都不能使用这里给出的网络全局视图。

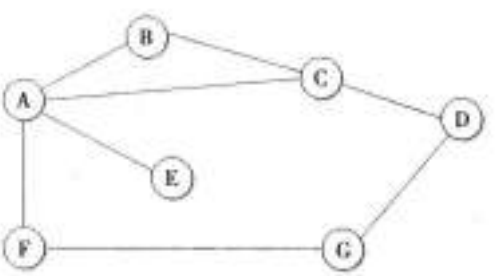


图 3-29 距离向量路由的网络例子

我们可以把表 3-10 中的每一行视作一个节点到所有其他节点的距离列表，代表这个节点的当前判断。最初，每个节点都把到直接邻接点的开销赋值为 1，到所有其他节点的

⊖ 这一广泛使用的假设近些年得到了再次检查——详见“扩展阅读”部分。

⊜ 参见“实验六”。

⊝ 此类算法的一个通用名字是 Bellman-Ford，即其发明者的名字。

表 3-10 存储在每个节点中的初始距离（全局视图）

存储在节点的信息	到每个节点的距离						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

开销赋值为无穷大。这样最初 A 点知道它可以经一跳到达 B，而 D 是不可达的。存储在 A 中的路由表反映了这样的信念，并且包括 A 用来到达其他任何可达节点的下一跳的名字。于是，最初 A 的路由表如表 3-11 所示。

距离向量路由的下一步是每个节点发送一个包含自己距离表的消息给其相邻节点。例如，节点 F 告诉节点 A 它可以到节点 G，开销为 1；A 也知道它能以开销 1 到达 F，因此二者相加就可以得到经 F 到 G 的开销。总开销 2 小于当前的开销无穷大，因此 A 记录它可经 F 到达 G，开销为 2。类似地，A 从 C 得知，C 能以开销 1 到达 D；A 将此与到 C 的开销 1 相加，决定可通过 C 以开销 2 到达 D，优于旧的开销无穷大。同时，A 从 C 得知，C 能以开销 1 到达 B，因此它推断经 C 到 B 的开销为 2。由于这比当前 A 到 B 的开销 1 大，因此新的信息被忽略。

至此，A 能够更新路由表中所有网络节点的开销和下一跳，结果如表 3-12 所示。

表 3-11 节点 A 的初始路由表

目的地	开销	下一跳
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

表 3-12 节点 A 的最终路由表

目的地	开销	下一跳
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

当拓扑结构不变时，每个节点只需与相邻节点之间交换少量信息即可得到完整的路由表。得到所有节点的路由信息的过程叫作收敛（convergence）。表 3-13 给出了路由收敛后从每个节点到其他节点的一组最终开销。我们必须强调，网络中没有任何一个节点有这张表的所有信息，每个节点只知道它自己的路由表的内容。像这样的分布式算法的优点就是它能使所有节点在没有任何集中授权的情况下取得对网络的一致视图。

表 3-13 存储在每个节点的最终距离（全局视图）

存储在节点的信息	到每个节点的距离						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

在结束有关距离向量路由的讨论之前，还要插进一些细节。首先，我们注意在两种情况下，一个给定的节点决定发送路由更新消息给它的相邻节点。一种情况是定期（periodic）更新。在这种情况下，即使没有任何变化，每个节点也总要时常自动发送更新消息。这使其他节点知道它仍在正常运行中。这样也可以确保当现有路由不可用时，它们仍能一直得到所需的信息。这些定期更新的频率随协议的不同而不同，但一般都是几秒到几分钟进行一次。第二种情况有时称为触发（triggered）更新，每当一个节点通知链路故障或从它的相邻节点接收到导致路由表中路由发生改变的更新时，便会引发这种更新。也就是说，当一个节点的路由表改变时，它就给其相邻节点发送一条更新消息，这可能引起相邻节点的路由表改变，使得这些相邻节点又给它们的相邻节点发送更新消息。

现在，我们考虑一下当链路或节点发生故障时会发生什么。首先注意到这个问题的节点发送新的距离列表给它的相邻节点，正常情况下，系统会很快平静地达到一种新的状态。至于一个节点如何探查故障，有两种不同的答案。一种途径是，节点通过发送控制分组持续地检测到另一节点的链路，并查看是否接收到确认。另一种途径是，如果节点在最近几次更新周期中接收不到预期的定期更新，则确定该链路（或链路另一端的节点）发生故障。

为了明白当一个节点检测到一条链路故障时将会怎样，我们来看当节点 F 发现它到 G 的链路发生故障时的情况。首先，F 设置到 G 的新距离值为无穷大，然后把这条信息传给 A。由于 A 知道自己到 G 的 2 跳路径经过 F，所以 A 也把它到 G 的距离设为无穷大。但是，从 C 送来的下一次更新中，A 将得知 C 有一个 2 跳路径到 G。这样，A 就知道它可以经过 C 以 3 跳的距离到达 G，这个开销小于无穷大，所以 A 相应地更新它的转发表。当 A 把更新消息通知 F 时，F 即知可以经过 A 以开销 4 到达 G，这个开销小于无穷大，从而系统将重新达到稳定。

遗憾的是，略有不同的情况就可能阻碍网络的稳定。例如，假设从 A 到 E 的链路出现故障。在下一次更新周期中，A 通知到 E 的距离为无穷大，但是 B 和 C 通知到 E 的距离为 2。根据事件发生的确切时序，可能发生如下事件：节点 B 一旦知道从 C 可以 2 跳的距离到达 E 时，就断定它可以 3 跳到达 E，并且把这条更新信息通知 A；节点 A 断定它能以 4 跳到达 E，并把这条信息通知 C；节点 C 断定它能以 5 跳到达 E；依此类推。只有当距离值达到一个足以被认为是无穷大的值时，循环才会停止。其间，没有一个节点真正知道 E 是不可达的，网络的路由表不能达到稳定。这种情况叫作计数到无穷（count-to-infinity）问题。

现有几种可使此问题得到部分解决的办法。第一种办法是使用一个相对较小的数作为无穷大的近似值。例如，我们可以推断穿过某个特定网络的最大跳数不会超过 16，因此可以选择 16 来表示无穷大。这至少可以限制计数到无穷大所花费的时间。当然，如果我们的网络增长到某些节点的距离大于 16 跳时，又会出现问题。

改进稳定路由时间的一种技术称为水平分割（split horizon）。其思想是当一个节点把路由的更新消息发送给相邻节点时，它并不把从各个相邻节点处学到的路由再回送给该节点。例如，如果节点 B 在其表中有路由信息（E，2，A），那么它就知道该路由一定是从节点 A 学到的。所以不论 B 什么时候给 A 发送更新消息时，其中都不包括路由（E，2）。水平分割的一种增强变形称为带反向抑制的水平分割（split horizon with poison reverse）。这种方法中 B 确实把来自 A 的路由回送给 A，但在该路由中加入否定信息来确保 A 最终

不会使用B到达E。例如，B把路由（E，∞）发送给A。这两种技术的问题在于它们只在涉及两个节点的路由循环中有效，对于更大的路由循环要求有更强的措施。继续以上的例子，如果B和C接收到A的链路故障后，在把路由通知给E之前等待一段时间，它们就会发现，实际上两者都没有到达E的路由。不幸的是这种方法延迟了协议的收敛，收敛速度是它的竞争对手链路状态路由（3.3.3节的主要内容）的主要优势之一。

1. 实现

实现这个算法的代码非常简单，在这里我们只给出了最基本的一些代码。结构Route定义了路由表中的每一条记录，常量MAX_TTL说明每一记录在被丢弃之前可以在表中保留多长时间。

```
#define MAX_ROUTES      128     /* maximum size of routing table */
#define MAX_TTL         120     /* time (in seconds) until route expires */

typedef struct {
    NodeAddr    Destination;    /* address of destination */
    NodeAddr    NextHop;        /* address of next hop */
    int         Cost;           /* distance metric */
    u_short     TTL;            /* time to live */
} Route;

int     numRoutes = 0;
Route   routingTable[MAX_ROUTES];
```

根据一条新路由来更新本地节点路由表的例程由mergeRoute给出。尽管没有显示出来，但定时器函数定期检查节点路由表中的路由列表，对每个路由中的TTL（生存期）字段进行递减运算，并丢弃生存期为0的所有路由。然而需要注意的是，一旦路由被相邻节点发送的更新消息重新确认，TTL字段就被重新设为MAX_TTL。

```
void
mergeRoute (Route *new)
{
    int i;

    for (i = 0; i < numRoutes; ++i)
    {
        if (new->Destination == routingTable[i].Destination)
        {
            if (new->Cost + 1 < routingTable[i].Cost)
            {
                /* found a better route: */
                break;
            } else if (new->NextHop == routingTable[i].NextHop) {
                /* metric for current next-hop may have
                   changed: */
                break;
            } else {
                /* route is uninteresting--just ignore
                   it */
                return;
            }
```

```
        }
    }
    if (i == numRoutes)
    {
        /* this is a completely new route: is there room
           for it? */
        if (numRoutes < MAXROUTES)
        {
            ++numRoutes;
        } else {
            /* can't fit this route in table so give up */
            return;
        }
    }
    routingTable[i] = *new;
    /* reset TTL */
    routingTable[i].TTL = MAX_TTL;
    /* account for hop to get to next node */
    ++routingTable[i].Cost;

}
```

最后，过程 updateRoutingTable 是主例程，它调用 mergeRoute 合并从相邻节点接收到的路由更新消息中的所有路由。

```
void
updateRoutingTable (Route *newRoute, int numNewRoutes)
{
    int i;

    for (i=0; i < numNewRoutes; ++i)
    {
        mergeRoute(&newRoute[i]);
    }
}
```

2. 路由信息协议（RIP）

在 IP 网络中，使用最广泛的路由协议之一就是路由信息协议（Routing Information Protocol，RIP）。它的广泛使用很大程度上依赖于 Unix 操作系统“伯克利软件发布版”（BSD）的一同发布，从 BSDUnix 中发展出了许多 Unix 的商用版本。RIP 还非常简单，它是建造在距离向量算法基础之上的路由协议的范例。

互联网中的路由协议与以上描述的理想化的图模型略有不同。在互联网中，路由器的目标是学会如何向不同的网络（network）转发分组。因此，路由器通知的是到达网络的开销，而不是到达其他路由器的开销。例如，在图 3-30 中，路由器 C 通知路由器 A 它可以以开销 0 到达网络 2 和网络 3（与它直接相连），以开销 1 到达网络 5 和网络 6，以开销 2 到达网络 4。

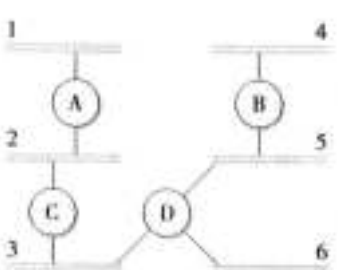


图 3-30 运行 RIP 的网络例子

我们可以从图 3-31 的 RIP（第二版）分组格式中看到这

一点。分组的主要部分为（address，mask，distance）（〈地址，掩码，距离〉）。然而，路由算法的原理正好是一样的。例如，如果路由器 A 从路由器 B 得知，经 B 到达网络 X 比经路由表中现有的下一跳到达网络 X 的开销更小，那么 A 将更新网络号的开销和下一跳信息。

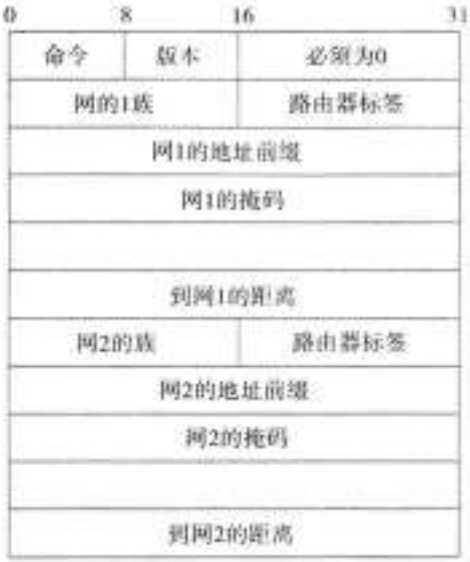


图 3-31 RIP（第二版）分组格式

事实上，RIP 是距离向量路由的一个相当简单的实现。运行 RIP 的路由器每 30s 发出一次通知，无论何时路由器收到来自其他路由器的引起转发表改变的信息，它都将发送一个更新消息。值得注意的一点是它支持多地址族，而不仅仅是 IP，这也是其通知中存在 Family（族）部分的原因。RIP 第二版（RIPv2）同样引入了一个像 RIP 第一版中服务于旧 IP 分类地址一样的子网掩码，这种掩码将在 3.2.5 节中介绍。

正如我们下面将看到的，在路由协议中可能使用多种不同的度量或开销来衡量链路。RIP 采取最简单的方法，使所有链路开销都等于 1，正如在上面的例子中所看到的。因此它总是尽力找到最少跳数的路由。有效距离为 1～15，16 表示无穷大。这也限制了 RIP 只可在很小的网络上运行，那些网络的路径长度不超过 15 跳。

3.3.3 链路状态（OSPF）[⊖]

链路状态路由是第二类主要的域内路由协议。链路状态路由的最初假设与距离向量路由的最初假设非常相似。假设每个节点都能找出到它的相邻节点（向上或向下）的链路状态以及每条链路的开销。我们还希望提供给每个节点足够的信息，使它能找出到达任一目标的最小开销路径。链路状态协议的基本思想非常简单：每个节点都知道怎样到达它的邻接点，如果我们确保这种信息被完整地传播到每个节点，那么每个节点都有足够的网络信息来建立一个完整的网络映象。显然，这是找到到达网络中任一点的最短路径的充分（但不必要）条件。因此，链路状态路由协议依靠两种机制：链路状态信息的可靠传播，根据所有积累的链路状态知识的总和进行的路由计算。

1. 可靠扩散

可靠扩散（Reliable Flooding）是确保参与路由协议的所有节点都能得到来自其他节点的一个链路状态信息副本的处理过程。正如术语扩散（flooding）所揭示的，它的基本思想是一个节点沿着所有与其直接相连的链路把其链路状态信息发送出去，接收到这个信息的每个节点再沿着所有与它相连的链路进行转发。这个过程一直继续，直到该信息到达网络中的所有节点。

更确切地说，每个节点创建一个更新分组，也叫链路状态分组（LSP），包含以下信息：

⊖ 参见“实验七”。

- 创建 LSP 节点的 ID。
- 与该节点直接相邻的节点列表，包括到这些相邻节点的链路开销。
- 一个序号。
- 这个分组的生存期（TTL）。

前两项用于路由计算，后两项用于将分组可靠地扩散到所有节点。可靠性包括确保拥有信息的最新副本，因为可能有来自于一个节点的多种不一致的 LSP 经过网络。保证可靠扩散是相当困难的（例如，在 ARPANET 网中采用的链路状态路由的一个早期版本就曾在 1981 年引起网络故障）。

扩散的工作方式如下。首先，两台邻接路由器之间的 LSP 传送使用确认和重传来保证可靠性，就像 2.5 节描述的可靠链路层协议一样。然而，还需要有更多的步骤来保证将一个 LSP 可靠地扩散到网络中的所有节点。

考虑节点 X 接收到一份来自节点 Y 的 LSP 副本的情况。注意节点 Y 可以是与节点 X 在同一个路由域中的其他任何一台路由器。X 检查是否已经存有一份来自 Y 的 LSP 副本。如果没有，它便存储这份 LSP。如果已经有了一份副本，则比较序号；如果新 LSP 序号更大，就被认为是最新的一份副本，并被保存用以替换旧的 LSP。较小的（或相等的）序号意味着这个 LSP 旧于（或不新于）已存储的 LSP，因此将被丢弃而无须采取进一步的行动。如果接收到的 LSP 是新的，那么 X 将发送这个 LSP 的副本给除了刚才发送 LSP 的那个相邻节点以外的所有相邻节点。不把 LSP 发回刚才发送它的节点，这使得 LSP 的扩散过程能够结束。由于 X 将这个 LSP 传向了它的所有邻居，邻居又转而做同样的事情，所以最终 LSP 的最新副本将到达所有节点。

图 3-32 表示一个 LSP 在一个小型网络中的扩散。每个节点在存储新的 LSP 后变成带阴影的形式。在图 3-32a 中，LSP 到达节点 X，在图 3-32b 中，X 将它发往邻居 A 和 C。A 和 C 不将它发回 X，但将它发往 B。由于 B 接收到这个 LSP 两个同样的副本，因此它将接收首先到达的副本而将第二个作为重复予以忽略。然后它将 LSP 传向 D，D 没有邻居需要扩散，因此过程结束。

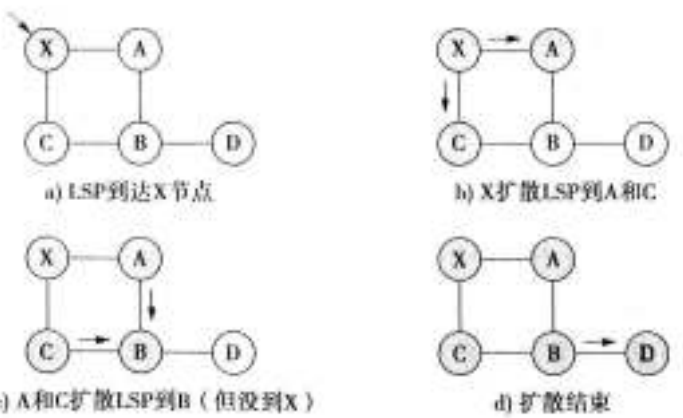


图 3-32 链路状态分组的扩散

与在 RIP 中一样，每个节点在两种情况下产生 LSP。周期性计时器超时或拓扑结构变化都将导致节点产生一个新的 LSP。然而，当与一个节点直接相连的链路或邻居出现故障时，才是节点产生一个 LSP 的基于拓扑的原因。链路故障有时可由链路层协议检测到。

邻居的“死亡”或对邻居的连接的丢失可由周期性的“hello”分组检测到。每个节点都在规定的时间间隔内向它的直接邻居发送这些信息。如果在一段足够长的时间内没有接收到来自于某个邻居的“hello”分组，通向那个邻居的链路将被宣告出现故障，并产生一个新的LSP来反映这一事实。

链路状态协议扩散机制的重要设计目标之一是必须将最新的信息以尽可能快的速度扩散到所有节点，同时旧信息必须从网络中删除而不允许在网络中循环。另外，我们显然希望减少网络中传送的路由的总通信量，毕竟，在那些实际把网络用于其应用程序的人看来，这就是系统开销。下面几段描述达到这些目标的一些方法。

减少系统开销的一种简单的方法是，除非有绝对必要，否则避免生成LSP。可以使用很长时间（通常是几个小时）的定时器周期性地生成LSP来做到这一点。如果当拓扑结构改变时扩散协议的确可靠，那么，可以放心假设不需要频繁发出“没有变化”消息的做法是安全的。

为了确保旧信息被新信息代替，LSP应携带序号。一个节点每产生一个新的LSP，就将序号增1。与协议中使用的大多数序号不同的是，这些序号不能重叠，所以需要很大的字段（如64比特）。如果一个节点出现故障后又恢复，那么它的序号从0开始。如果节点出现故障已很长一段时间，那么该节点所有旧的LSP都会超时（如下面所述）；否则，该节点最终会收到一份它自己的带有更大序号的LSP副本，然后它可以将序号加1并作为自己的序号使用。这将确保它的新LSP能够代替所有在节点出现故障之前遗留下来的旧LSP。

LSP还携带着一个生存期，用于确保旧的链路状态信息最终从网络中删除。一个节点在将新接收到的LSP扩散到其邻居之前，对其TTL减1。同时它也使节点中存储的LSP“衰老”。当TTL值等于0时，节点再次扩散TTL为0的这个LSP，这被网络中所有节点解释为删除这个LSP的信号。

2. 路由计算

一旦一个给定节点有了一份来自其他每个节点的LSP，它就能计算出完整的网络拓扑结构图，并可以根据这幅图决定到达每个目的地的最佳路由。然后，问题就是如何从这些信息中计算出路由。解决办法基于图论中的一个著名算法：Dijkstra的最短路径算法。

我们首先用图论的术语来定义Dijkstra算法。设想一个节点将它接收到的所有LSP信息构造成一个代表网络的图，其中N代表图中的节点集合，$l(i,j)$表示两个节点i，$j\in N$之间的边上的非负开销（权值），如果在i、j之间没有边相连，则$l(i,j)=\infty$。在下面的描述中，我们令s表示这个节点，$s\in N$，也就是执行算法寻找到达N中其他所有节点的最短路径的节点。此外，算法还维护以下两个变量：M代表到目前为止参与算法的节点集，$C(n)$表示从节点s到每个节点n的路径开销。给出这些定义后，算法的定义如下：

```
M=(s)
for N-(s)中的每个n
  C(n)=l(s,n)
while (n≠M)
  M=M∪{w}以使C(w)对于(N-M)中的所有w而言是最小的
  for (N-M)中的每个n
    C(n)=MIN(C(n),C(w)+l(w,n))
```

算法基本执行过程如下。首先从含有节点 s 的 M 开始，并使用到直接相连节点的已知开销初始化到其他节点的开销表 $C(n)$。然后寻找能以最小开销（w）到达的节点，并把这个节点加入 M 中。最后考虑用经过 w 到达节点的开销来更新开销表。在算法的最后一行中，如果从源点 s 到 w，加上从 w 到 n 总的开销小于已有的从 s 到 n 旧的路径的开销，那么我们就选择这条经 w 到达 n 的新路径。重复这个过程，直到所有节点都归入 M 中。

实际上，每台交换机使用一种称为向前搜索（forward search）的 Dijkstra 算法实现，从它收集的 LSP 中直接计算路由表。具体地说，每台交换机维护两张表：试探表（Tentative）和证实表（Confirmed）。每张表中有多条记录，每条记录包含（Destination，Cost，NextHop）（目的地，开销，下一跳）。算法如下：

1）用我的节点中的一条记录初始化证实表，这条记录中的开销为 0。

2）在前一步中加入证实表的那个节点，称为 Next（下一点）节点，选择它的 LSP。

3）对于 Next 节点的每个 Neighbor（相邻）节点，计算到达这些相邻节点的开销（Cost），也就是从我的节点到 Next 节点和再从 Next 节点到 Neighbor 节点的总开销之和。

a）如果相邻节点当前既不在证实表中也不在试探表中，那么把（Neighbor，Cost，NextHop）记录加入试探表中，其中 NextHop 是我到达 Next 节点所经的节点。

b）如果 Neighbor 节点当前在试探表中，且开销小于当前登记在表中的开销，那么用记录（Neighbor，Cost，NextHop）替换当前记录，其中 NextHop 是我到达 Next 节点所经的节点。

4）如果试探表为空，则停止。否则，从试探表中挑选开销最小的记录，把它移入证实表，并转回执行第 2）步。

我们看一个例子就更容易理解这个算法。考虑图 3-33 中描述的网络。注意，这幅图不像前几个例子，这个网络带有不同的边上开销。表 3-14 给出构造节点 D 的路由表的步骤。我们用与节点 D 相邻的节点 B 和 C 表示 D 的两个输出。注意，这个算法的开头看起来似乎方向有误（比如，将到达 B 的开销为 11 的路径第一个加入试探表），但最终得到了到达所有节点的最小开销路径。

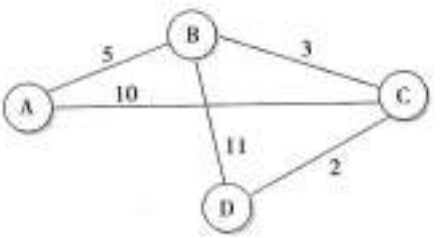


图 3-33 链路状态路由的网络示例

表 3-14 对节点 D（图 3-33）建立路由表的步骤

步骤	证实表	试探表	注 释
1	(D,0,—)		因为 D 是证实表中唯一的新成员，所以观察它的 LSP
2	(D,0,—)	(B,11,B) (C,2,C)	因 D 的 LSP 表明，我们可以以开销 11 通过 B 到达 B，比表中任何其他路径都好。因此把它加入试探表中，同理 C 也加入
3	(D,0,—)(C,2,C)	(B,11,B)	把试探表中开销最小的记录 C 加入证实表中。接着，检查证实表中新的成员 C 的 LSP
4	(D,0,—)(C,2,C)	(B,5,C) (A,12,C)	因为通过 C 到达 B 的开销是 5，所以替换记录（B，11，B），C 的 LSP 告诉我们可以以开销 12 到达 A
5	(D,0,—)(C,2,C) (B,5,C)	(A,12,C)	把试探表中开销最小的记录 B 加入证实表中，观察它的 LSP
6	(D,0,—)(C,2,C) (B,5,C)	(A,10,C)	因为可以经过 B 以开销 5 到达 A，所以替换试探表中的记录
7	(D,0,—)(C,2,C) (B,5,C)(A,10,C)		把试探表中开销最小的成员 A 移入证实表中，结束

链路状态路由算法有许多优点：它可以很快达到稳定状态，不产生过多的通信量，而且对拓扑结构改变或节点故障反应迅速。但缺点是每个节点存储的信息量（一个网络中其他所有节点的 LSP）可能非常大。这是路由的基本问题之一，也是可扩展性这个更一般性问题的一个例证。对于既能解决特殊性问题（每个节点可能需要的存储量）又能解决一般性问题（可扩展性）的一些方法，我们将在下一章讨论。

结论 距离向量算法和链路状态算法的区别可以总结如下。在距离向量算法中，每个节点只和直接相连的节点进行通信，但是它把所知的全部信息（即到所有节点的距离）都告诉它们。在链路状态算法中，每个节点和其余各个节点都进行通信，但只告诉它们自己确切知道的信息（即与其直接相连的链路状态）。

3. 开放最短路径优先协议（OSPF）

一个使用最广泛的链路状态路由协议是开放最短路径优先（Open Shortest Path First，OSPF）协议。第一个词“Open”指出它是一个开放的、非专有的和由 IETF 主持创建的标准。“SPF”部分来自于链路状态路由的另一个名字。OSPF 为以上描述过的基本链路状态算法增加了相当多的特性，这些特性包括：

- *路由消息的认证*：这是一个很好的特性，因为经常有一些错误配置的主机认为它能以 0 开销到达全世界的每台主机。当主机发布这件事时，周围相邻的每台路由器都更新它们的转发表以指向这台主机，也就是说这台主机接收到数量非常大的数据，其实它不知该如何处理。通常，它将所有数据丢弃，使网络停止。在很多情况下，这样的灾难可以通过要求认证路由更新消息来预防。OSPF 早期的版本使用一个简单的 8 字节口令进行认证。虽然这个认证尚不足以防止那些恶意用户，但是它减少了由错误配置引起的很多问题。（类似的一种认证形式也加到 RIP 第 2 版中。）8.3 节讨论的更强的密码认证是后来加上去的。
- *附加的层次*：分层是使系统具有更好的可扩展性的基本工具之一。OSPF 通过允许将域划分成区（area）给路由层次结构引入了另外一层。这意味着域内的路由器无须知道如何到达域内的每个网络，只需知道如何到达正确的区就足够了。这样，必须传送及存储在每个节点中的信息量将会减少。我们将在 4.1.1 节详细地讨论区。
- *负载平衡*：OSPF 允许到同一位置的多条路由有相同的开销，这样可以使通信量均匀地分布于这几条路由上。

OSPF 消息有多种不同类型，但是都以相同的首部开始，如图 3-34 所示。Version（版本）字段当前设为 2，Type（类型）字段可以从 1～5 取值。SourceAddr（源地址）指明消息的发送方，AreaId（区标识符）是节点所在区的 32 位标识。除了认证数据以外，整个分组使用和 IP 首部同样的算法（见 2.4 节）由 16 位的校验和来保护。若不使用认证，Authentication type（认证类型）为 0；否则为 1，表示使用一个简单的口令；或者为 2，表示使用 8.3 节中描述的一种加密认证校验和。在后一种情况下，Authentication



图 3-34 OSPF 首部格式

(认证)字段携带口令或加密校验和。

在5种OSPF消息类型中，类型1是“hello”消息，路由器将它发送给对等实体，表明自己仍是活动的且以上述方式连接。其余的消息类型用于链路状态消息接收的请求、发送和确认。OSPF中链路状态消息的基本构件叫作链路状态通知(LSA)。一条消息可包含多个LSA。这里我们提供一些有关LSA的细节。

像任何互联网路由协议一样，OSPF必须提供如何到达网络的信息。因此，OSPF必须比以上描述的简单基于图的协议提供更多的信息。特别是，一个运行OSPF的路由器可以产生链路状态分组，通知直接与其相连的一个或多个网络。另外，一个通过某条链路连接到另一台路由器上的路由器必须通知经此链路到达该路由器所需的开销。这两种类型的通知，对于一个域中的所有路由器确定到达该域中所有网络的开销及到达每个网络适当的下一跳都是必需的。

图3-35给出了类型1的链路状态通告的分组格式。类型1的LSA在路由器之间通知链路开销。类型2的LSA用来通知发出通知的路由器所连接的网络，而其他类型用于支持附加的层次结构，见下节所述。LSA中的很多字段应与前面讨论过的类似。LS Age(链路状态生存期)相当于生存期，不过它是递增计数的，当它达到一个定义的最大值时，LSA终结。Type(类型)字段告诉我们这是一个类型1的LSA。

链路状态生存期			选项	类型=1
链路状态ID				
通知路由器				
LS序号				
LS校验和			长度	
0	标志	0	链路数目	
链路ID				
链路数据				
链路类型		TOS值	度量值	
可选TOS信息				
其余链路				

图3-35 OSPF链路状态通告

在一个类型1的LSA中，Link-State ID(链路状态ID)和Advertising router(通知路由器)字段是一样的。每个字段都携带一个创建此LSA的路由器的32位标识符。尽管很多分配策略都可用于分配此ID，但最本质的是在一个路由域中ID必须唯一，并且一个给定的路由器必须一直使用同一个路由器ID。一种符合这些要求的选择路由器ID的方法是选择所有IP地址中最小的一个赋给路由器。(注意，一台路由器的每个接口都可能有一个不同的IP地址。)

LS sequence number(LS序号)的使用完全如上所述，用来检测旧的或重复的LSA。LSchecknum(LS校验和)类似于我们在2.4节和其他协议中讲过的某些其他校验和，用来验证数据未被破坏。它涉及分组中除LSAge之外的所有字段，因此每当LSAge加1后，不必重新计算校验和。Length(长度)是以字节为单位的整个LSA的长度。

现在我们来看实际的链路状态信息。因为TOS(服务类型)信息的存在使得这件事稍有一点麻烦，故暂且将其忽略，那么LSA中的每条链路都可由一个LinkID(链路ID)、一些Link Data(链路数据)和一个metric(度量值)来表示。这些字段中的前两项标识链路，一种普遍使用的方法是使用链路远端的路由器ID作为链路ID，并且如果需要的话，使用链路数据来区别多条并行链路。度量值当然是链路的开销。Type(类型)告诉我们有关链路的一些信息，例如，它是不是一条点到点的链路。

TOS信息的使用，允许OSPF基于它们的TOS字段值为IP分组选择不同的路由。依据数据的TOS值可为链路分配多个度量值而不是只分配一个度量值。例如，在网络中有一条链路专用于对延迟很敏感的通信，那么我们就可以用一个较低的度量值作为TOS值表示低延迟，而用一个较高的度量值表示其他。OSPF将为设置了那个TOS值的分组选

择不同的最短路径。值得注意的是，直到编写本书时，这项功能还未被广泛应用。

3.3.4 度量

在前面的讨论中，当我们执行路由算法时，都假设链路开销或度量标准是已知的。本节我们来看实际中一些有效的计算链路开销的方法。前面我们已经看到一个完全合理和十分简单的例子，即给所有链路分配的开销都是1，这样最小开销的路由也就是跳数最少的路由。然而这种方法有几个缺点。首先，它不是根据时延区分链路。这样，对路由协议来说，时延为250ms的卫星链路和时延为1ms的陆地链路是没有区别的。其次，它不是根据容量区分链路，这使得传输速度为9.6kbps的链路与传输速度为45Mbps的链路看起来一样好。最后，它也不是根据链路当前的负载情况区分链路，使其不可能绕过负载过重的链路。结果证明最后一个问题是最难的，因为你试图用单纯数量上的开销来获得复杂的和动态的链路特征。

很多不同的链路开销计算方法都以ARPANET网作为测试环境。（也正是在ARPANET网中展示了链路状态路由的稳定性胜过距离向量路由；最初的机制采用距离向量，但后来的版本使用链路状态。）下面的讨论研究ARPANET网路由度量标准的发展过程，同时，也研究了这个问题的一些细微方面。

最初的ARPANET网路由度量标准度量在每条链路上排队等待发送的分组数量，有10个分组排队等待发送的链路分配到的开销比有5个分组排队等待发送的链路大。然而，使用队列长度作为路由度量标准并不是个好办法，因为队列长度是负载大小的一个人为度量，它把分组移往最短的队列而不是目的地，这种情况很像有些人在超市收款台前从一个队列换到另一个队列。更确切地说，起初的ARPANET网路由机制容忍这样一个事实，即它不考虑链路的带宽，也不考虑链路的时延。

ARPANET网路由算法的第2版，有时称为*新路由机制*（new routing mechanism），既考虑了链路带宽，又考虑了链路时延，并使用延迟而不是队列长度作为负载的衡量标准。其实现方式如下：首先每个进入的分组以其到达路由器的时间（ArrivalTime）作为时标，同时记录它离开路由器的时间（DepartTime）；其次，当从链路另一端接收到链路层ACK后，节点计算分组的延迟为

$$\text{Delay} = (\text{DepartTime} - \text{ArrivalTime}) + \text{TransmissionTime} + \text{Latency}$$

其中TransmissionTime和Latency都是链路的静态特征，分别反映链路的带宽和时延。注意在这个公式中，DepartTime－ArrivalTime表示分组在节点中因装载而被延迟（排队等待）的时间量。如果ACK没有到达而是分组超时，就把DepartTime重新设定为分组的重传（retransmit）时间。这种情况下，DepartTime－ArrivalTime体现了链路的可靠性——分组被重传的次数越多，链路的可靠性越差，我们就越想避免这种情况。最后，分配给每条链路的权值是最近从这条链路发送的分组所经历的平均延迟。

虽然这个方法比原来的机制有所改进，但也存在不少问题。在较轻负载下，它可以正常工作，因为延迟的两个静态因素支配着开销。然而，在重负载下，拥塞的链路将开始通知一个很高的开销。这就使所有的通信量脱离该链路，从而使其空闲。然后，它又会通知一个低开销，因此，又引回所有通信量，如此反复。这种不稳定性产生的后果就是在重负载下，许多链路实际在空闲状态上浪费大量时间，这是重负载下最不希望发生的事情。

另一个问题是链路开销的范围会过大。例如，一条传输速度为9.6kbps的重负载链

路，其开销大约相当于传输速度为 56kbps 的轻负载链路的 127 倍。这就意味着，与传输速度为 9.6kbps 的 1 跳路径相比，路由算法宁愿选择一条由传输速度为 56kbps 的轻负载链路组成的 126 跳的路径。尽管从超负载的链路上减少一些通信量是一个好主意，但是这并没有什么吸引力，因为它失掉了超载的通信量。用 126 跳的链路去完成 1 跳的链路能完成的事，通常会极大地浪费网络资源。同样，对卫星链路而言也很不利，一条 56kbps 的空闲卫星链路要比一条 9.6kbps 的空闲陆地链路开销高得多，尽管前者可以为高带宽的应用提供更好的性能。

第三种称为“修正的 ARPANET 网路由度量标准”的方法解决了这些问题。主要的改进是大量缩减度量值的动态范围，说明链路的类型以及减缓度量值随时间的变化。

可通过以下几种措施来减缓度量值的变化。首先，将延迟度量换成链路的利用率，并将这个值和最近记录的利用率取平均值来减缓突然的改变。第二，对于从一个衡量周期到下一个周期中度量值可以改变多少要有严格的限制。通过减缓开销的变化，所有节点同时丢弃一条路由的可能性就大大降低了。

把测得的利用率、链路类型和链路速度输入到一个如图 3-36 所示的函数中，就得到动态范围的压缩。观察以下几点：

- 一个高负载链路的开销不可能大于其空闲时开销的 3 倍。
- 最昂贵的链路开销也仅仅是最廉价的链路开销的 7 倍。
- 高速的卫星链路比低速的陆地链路更有吸引力。
- 只有在中负载或高负载的情况下，开销才是链路利用率的函数。

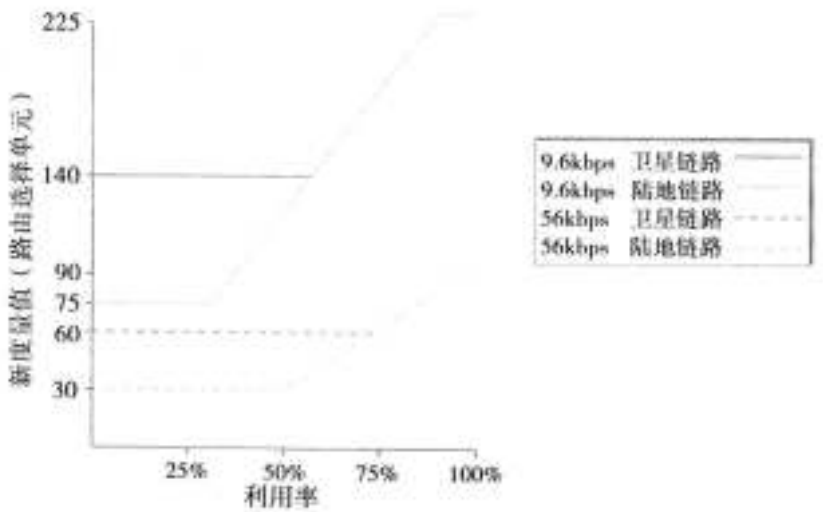


图 3-36 修正的 ARPANET 网路由选择度量值与链路使用率

这些因素意味着一条链路不可能完全被丢弃，因为开销增长 3 倍，对于一些路径来说，这条链路可能会成为不可取的，然而，对于其他路径来说却仍是最佳选择。图 3-36 中曲线的斜率、偏移量和分界点是通过反复试验获得的，并对其进行仔细地调整以提供良好的性能。

我们用现实来结束关于路由指标的讨论。在本书写作期间，大多数的实际网络度量变化非常少，且在网络管理员的控制之下，而不是像前面描述的那么自动化。部分原因是人们普遍认为动态变化的指标太不稳定，即使事实可能并非如此。也许更清楚的是，今天的

很多网络都缺乏比 ARPANET 更好的连接速度和延迟。因此，静态指标就是规范。一个设置指标的通用方法是采用常量乘以连接带宽的倒数。

相关主题

监控路由行为

鉴于穿过因特网这样规模的网络时路由分组的复杂性，我们也许想知道系统是如何正常工作的。我们有时候知道它工作正常，因为我们可以连到全世界的任何一个站点上。但我们怀疑它不是一直都正常工作，因为有时我们无法连接到某些站点上。真正的问题是当我们连接失败时，确定系统的哪一部分出了错，例如某种路由方式不能正常工作时，是由于远程服务器太忙还是某条链路或某台机器坏了？

这实际上是一个网络管理问题，虽然系统管理员可以用一些工具来监控网络，如 9.3.2 节描述的简单网络管理协议（SNMP），但网管问题对整个因特网来说仍然是未解决的一个主要问题。实际上，目前因特网已经增长到如此之大且如此复杂，即使它是由一些人工的和很大程度上确定的部分组成，但我们仍在很大程度上把它看作是一个活的有机体或自然现象去研究。也就是说，我们通过在因特网上进行一些实验试图了解它的动态行为，并提出模型来解释我们的观察。

此类研究的一个极好例子是由 Vern Paxson 做出的。1995 年，Paxson 使用 Unix 的 traceroute 工具研究了 37 个因特网站点之间的 40 000 条端到端的路由。他试图回答路由如何出故障，稳定的路由如何过时以及它们是否对称的问题。Paxson 发现用户遇到一个严重的端到端路由问题的可能性是 1/30，并且这样的问题通常持续约 30s。他还发现，有 2/3 的因特网路由持续存在几天或几周，且大约 1/3 的时间内，从主机 A 到主机 B 的路由与从主机 B 到主机 A 的路由相比，至少包含一个不同的路由域。最后 Paxson 得出结论，因特网路由正在变得越来越不可预测。

3.4 实现和性能

到目前为止，我们只讨论了交换机要做什么，却没有讨论它怎样去做。构造一台交换机有一个很简单的方法：买一台普通的工作站并为它配置多个网络接口。这台设备运行适当的软件就可以在它的一个接口上接收分组，执行前面描述的任一种交换功能，然后在它的另一个接口输出。实际上，当你想做像开发新路由协议这样的事时，这是一种建立实验交换机的流行方法，因为它提供非常好的灵活性和熟悉的编程环境。许多低端路由器的体系结构也没有脱离这种方法。

3.4.1 交换基础

交换与路由是相似的实现技术，所以我们将在这一节探讨它们的通用技术，然后转向一些在 3.4.4 节影响路由实现的特殊话题。在本节的大部分内容里，我们使用交换机（switch）来表示这两种设备，因为它们的设计是类似的（且总是说“交换机和路由器”很无聊）。

图 3-37 给出将一台有 3 个网络接口的工作站作为交换机的例子。图中显示一个分组可以从交换机的接口 1 到达然后从接口 2 输出的一条路径。在这里，我们假设工作站有一

种机制可以把数据从一个接口直接移动到它的主存储器，而不是被 CPU 直接复制，这种技术叫作直接内存访问（DMA）。一旦分组进入内存，CPU 就检查其首部来决定分组应该从哪个接口发送出去，然后，它用 DMA 把分组移动到相应的输出接口。注意，图 3-37 并没有画出分组进入 CPU，因为 CPU 只检查分组的首部，它并不读取分组中的每个数据字节。

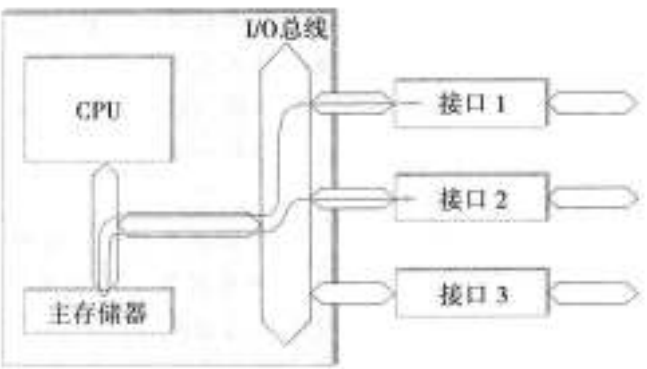


图 3-37 用作分组交换机的通用处理器

使用工作站作为交换机的主要问题是它的性能要受到限制，因为所有分组都必须通过一个的争用点。在上面提到的例子中，每个分组都要两次经过 I/O 总线，一次是写入内存，一次是从内存中读出。这样一个设备的总吞吐量（所有输入可支持的总的数据速率）的上界是内存带宽的一半，或者是 I/O 总线带宽的一半，取两者中较小的一个（通常是 I/O 总线带宽）。例如，一台有 133MHz、64 位 I/O 总线的工作站能够传输的数据峰值速率略高于 8Gbps。因为转发一个分组要经过总线两次，实际的速度就被限制在 4Gbps，它足以支持 100Mbps 的中端以太网接口卡。但在互联网中很难找到足够好的高端路由器。

此外这个上界还假设移动数据是唯一的问题；这种估算对长分组是很接近的，但当分组都较短时就相差很远。在后一种情况中，处理每个分组的开销，即分析它的首部信息及决定把分组从哪个输出链路传输出去，可能占开销的绝大部分。例如，假设一个工作站每秒能对 200 000 个分组完成所有必要的交换处理，有时称为分组每秒传输率（pps）。（这个数字代表了当今高档 PC 的处理能力。）如果平均分组较短，例如 64 字节，这就意味着

$$\text{Throughput} = \text{pps} \times (\text{BitsPerPacket}) = 2 \times 10^6 \times 64 \times 8 = 1024 \times 10^6$$

也就是 1Gbps 的吞吐量，大大低于当今网络用户要求的范围。记住这 1Gbps 是连接在这个交换机上的所有用户共享的，正如以太网中带宽被连接到共享介质上的所有用户所共享一样。那么，假如一台交换机有这样聚合吞吐量且带有 20 个端口，则每个端口就只能处理 50Mbps 的平均数据速率。

要解决这个问题，硬件设计者们设计出了大规模的交换机阵列，以减少争用的数量，提供较高的总吞吐量。注意，有一些争用是不可避免的：如果每个输入都有数据要发送到同一个输出上，那么它们不可能同时发送。然而，如果在不同输入上到达的数据要送到不同的输出上，一台设计良好的交换机能够以并行的方式把数据从输入送到输出上，这样就增大了总的吞吐量。

相关主题

定义吞吐量

很难准确地给交换机的吞吐量下定义。直观上，如果一台交换机有 n 个输入，而每个输入支持的链路速度为 s_i，那我们就认为吞吐量是所有 s_i 之和。这也许是实际中交换机所能提供的最大吞吐量。但事实上，没有任何交换机能保证这样的性能水平。原因很简单，假设在一段时间内，到达交换机的所有分组都要发送到同一个输出上，只要那个输出的带宽小于所有输入带宽之和，那么一些分组就将被送入缓冲区或丢弃。在这种特定的通信量模式下，交换机无法提供持续的而且速度高于那个输出链路速度的吞吐量。但是，如果分组平均分布在不同的输出端口上，交换机就能够控制所有输入端口以全链路速度进入的通信量，这种情况被认为是最优的。

另一个影响交换机性能的因素是到达输入端口的分组长度。对于一台 ATM 交换机来说，这通常不成问题，因为所有分组（信元）的长度相等。但对于以太网交换机或 IP 路由器来说，分组可能具有差异很大的长度。交换机必须执行的一些操作对每个分组都有固定的开销，因此，根据到达的所有分组都很短、都很长或有长有短的情况，交换机的执行可能有所不同。由于这个原因，所以对于转发变长分组的路由器和交换机来说，除了用比特每秒的吞吐量描述它们的性能外，还可以用分组每秒（packet per second，pps）的传输率来描述。pps 通常用最小长度的分组来测量。

对于上面的讨论，首先需要注意的一点是交换机的吞吐量是其通信量的函数。交换机的设计者们花费大量时间所做的事情之一就是试图提出一个能够模拟实际数据通信行为的通信量模型。事实证明获得精确的通信量模型是非常困难的。一个通信量模型试图解决的几个最主要问题是：①分组何时到达？②它们预定从哪个输出上发送出去？③它们有多大？

通信量建模是一门很成熟的科学，尤其在电话领域是非常成功的，它使电话公司能够设计网络以有效地传输预期的负载。部分原因是由于人们使用电话网络的习惯并不因时间的推移而变化：如打电话的频率、打一次电话的时间和人们多年来一直保持在母亲节打电话的习惯。相反，计算机通信的快速演变（如像 Bit Torrent 一样的新应用）几乎可以在一夜之间改变通信量模式，这就使有效地对计算机网络建模变得更加困难。不过，我们在本章的后面列出了有关这个主题的优秀书籍和文章。

为了让你对设计者必须关注的吞吐量范围有一个感性认识，我们来看一台写作时用在因特网上的高端路由器，它可支持 16 条 OC-768 链路，其吞吐量接近 640Gbps。一台 640Gbps 的交换机，如果处理 64 字节的稳定分组流，每秒分组传输率是

$$640\times10^9\div(64\times8)=1.25\times10^9\text{pps}$$

3.4.2 端口

大多数交换机和图 3-38 给出的交换机结构在概念上是相似的。它们由许多输入端口（input port）和输出端口（output port）以及一个交换结构（fabric）组成。通常至少有一个控制处理器负责整个交换，或者与端口直接通信，或者像这里所表示的那样通过交换结构进行交换。端口与外界通信，它们可能包含光纤的光接收器和激光发生器、存放等待交

换或传送分组的缓冲区以及大量能使交换机发挥作用的其他电路。交换结构的工作非常简单且有明确定义：当分组出现时，将其发送到正确的输出端口。

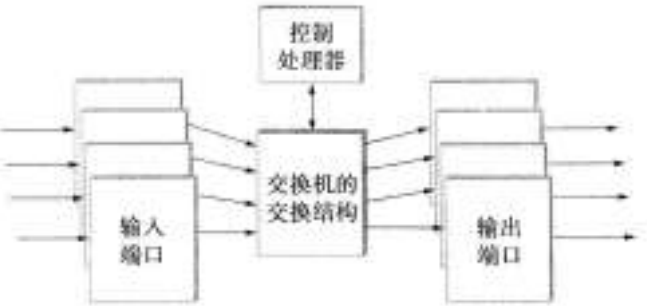


图 3-38 一台 4×4 的交换机

这样，端口的任务之一就是处理真实世界的复杂事物以便交换结构处理其中相对简单的工作。例如，假设交换机支持虚电路通信模式。3.1.2 节描述的虚电路映射表通常是在端口上。端口维护当前使用的虚电路标识符表，以及有关分组输出的信息，包括一个分组应该从每个 VCI 的哪个输出端口传输出去和怎样重新映射才能确保在输出链路上的唯一性。同样，以太网交换机的端口存储以太网地址和输出端口之间的映射表（如 3.1.4 节所述的网桥转发表）。通常，当一个分组从输入端口传送到交换结构时，端口已指出分组要送往哪里，端口通过传送某种控制信息相应地建立交换结构，或为分组附加足够的信息（如输出端口号）以便网状结构自动完成任务。只考察分组携带的信息进行分组交换的交换结构称作自选路由（self-routing），因为它们不需要额外的控制信息来为分组选择路由。下面讨论自选路由的交换结构的一个例子。

输入端口是第一个出现性能瓶颈的地方。输入端口必须接收一个稳定分组流，分析每个分组的首部信息来确定应发送到哪个（或哪些）输出端口，并传送分组到网状结构上。它执行的首部类型分析的范围，可以从一个简单的对某个 VCI 的表查找，到检查首部中很多字段的复杂匹配算法。正是这种操作类型在平均分组长度很小时可能会造成问题。例如，64 字节的分组到达连接 OC-48（2.48Gbps）链路的端口时，必须以

$$2.48\times10^{9}\div(64\times8)=4.83\times10^{6}\ \text{pps}$$

的速率处理分组。换句话说，当小的分组尽可能快地到达这个链路时（最坏的情况是多数端口都进行处理），输入端口用大约 200ns 处理每一个分组。

端口的另一关键功能是缓存。可以观察到，缓存可出现在输入端口，也可出现在输出端口，在网状结构中也可以有缓存（有时称为内部缓存（internal buffering））。简单的输入缓存方法有一些严重的局限性。考查一个用先进先出栈（FIFO）实现的输入缓冲区。当分组到达交换机时被放入输入缓冲区，然后交换机试图把每一个 FIFO 栈中的第一个分组转发到相应的输出端口。然而，如果几个不同输入端口上的第一个分组都要同时从一个输出端口输出，那么它们之中只能有一个被转发，其余的分组必须留在输入缓冲区[⊖]。

这种特性的缺点是那些滞留在输入缓冲区前面的分组阻碍缓冲区后面的分组传送到它们指定的输出上，尽管这些输出之间没有争用。这种现象称为队列头阻塞（head-of-line blocking）。图 3-39 给出一个简单的队列头阻塞的例子，其中我们看到竞争端口 2 的分组产生阻塞，阻碍了后面的一个希望从端口 1 输出



图 3-39 简单的队列头阻塞

⊖ 对于一个简单的输入缓冲交换机，一次只有一个分组能发送到给定的输出端口。可以设计多个交换机来实现一次转发多个分组到同一个输出端口，但基于更高的交换复杂性所带来的消耗，会对转发分组数量设置上限。

的分组。它表明当通信量均匀分布在各个输出时，队列头阻塞把输入缓存交换机中的吞吐量限制在最大理论值（交换机链路带宽的总和）的59%。因此，大多数交换机采用纯粹的输出缓存技术，或内部缓存和输出缓存混合的形式。这些依赖输入缓冲区的技术使用成熟的缓存管理方案来避免队列头阻塞。

缓冲区实际上执行更复杂的任务，而不仅仅是保存等待传输的分组。缓冲区是交换机时延的主要根源，也是由于缺少存放分组的空间而使分组多半会被丢弃的地方。因此，缓冲区是决定交换机服务质量特性的主要方面。例如，如果一个分组沿着有延迟保证的VC被发送出去，则它不能容许在缓冲区滞留太长时间。这意味着在一般情况下必须使用符合较宽QoS要求范围的分组调度和丢弃算法来管理缓冲区。我们在第6章进一步讨论这个问题。

3.4.3 交换结构

虽然有很多关于设计高效的和可扩展的交换结构的深入研究，但对于本节的目标来说，只了解一个交换机的交换结构的高级特性就足够了。交换机网状结构应当能用最小的延迟把分组从输入端口移到输出端口，并且是一种能满足交换机吞吐量目标的方式。这通常意味着交换结构呈现一定的并行度。有n个端口的高性能交换结构常常可以同时从它的每个输入端口把分组移到一个输出端口。交换结构类型的实例如下：

- *共享总线*（Shared-Bus）：像上面描述的，这是一种把传统工作站用作交换机时的一种交换结构。因为总线的带宽决定交换机的吞吐量，所以高性能的交换机通常有专门设计的总线而不是PC用的标准总线。
- *共享内存*（Shared-Memory）：在共享内存交换机中，分组由输入端口写进内存的一个地方，然后由输出端口从内存读出，这里存储器的带宽决定交换机的吞吐量。所以在这种设计中通常使用容量大和速度快的存储器。共享内存交换机和共享总线交换机相比，除了使用专门设计的高速内存总线代替I/O总线外，在原理上是相似的。
- *纵横式*（Crossbar）：纵横式交换机是一个能配置成任一输入端口和任一输出端口都有相连路径的矩阵。图3-40给出一个4×4纵横式交换机。最简单的纵横式交换机的主要问题是，需要每个输出端口能够立即从所有输入端口接收分组，这意味着每个端口要有一个带宽等于总交换吞吐量的存储器。在实际中，要解决这个问题一般需要更复杂的设计（例如，参见本章最后“扩展阅读”中的淘汰式（Knockout）交换机和McKeown的虚拟输出缓冲方法）。

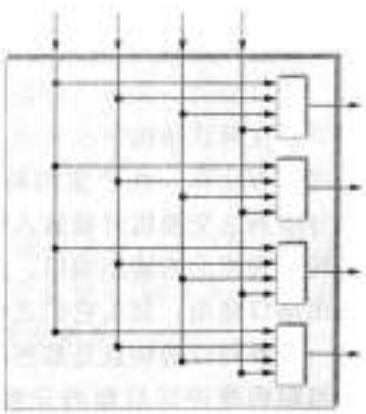
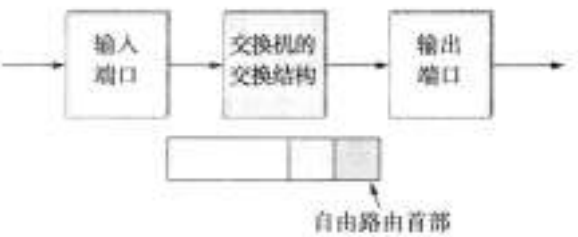


图3-40 一台4×4纵横式交换机

- *自选路由*（Self-routing）：前面提到，自选路由交换结构依赖于分组首部的某些信息将每个分组发往正确的输出端口。通常，在输入端口，在确定一个分组发往的输出端口后，一个特别的“自选路由首部”被加到分组上（如图3-41所示），这个额外的首部在分组离开交换机前被删除。自选路由交换结构通常由大量非常简单的2×2交换单元按规则的模式互联而成，如图3-42所示的榕树（banyan）交换网状结构。自选路由网状结构设计的例子参见本章最后的“扩展阅读”。



a) 分组到达输入端口



b) 输入端口携带自选路由分组首部并指引分组到正确的输出端口



c) 分组离开交换机时自选路由首部在输出端口被删除

图 3-41 为了使交换结构把分组发到正确的输出端口，一个自选路由首部在输入端口被加到分组上，最后被输出端口删除

自选路由交换结构存在于大多数可扩展性网状结构的设计方法中，而且对这个主题有大量的研究，“扩展阅读”中列出了一些。许多自选路由交换结构如图 3-42 所示，由 2×2 交换单元规则地互联组成。例如，在榕树网络中的 2×2 交换单元执行一个简单的任务：它们检查每个分组首部的一个二进制位，如果它是 0 就把分组送到上面的端口，是 1 就把分组送到下面的端口。显然，如果两个分组同时到达同一个榕树单元，并且它们这一位的值也相同，那么它们将被送到同一个输出端口，冲突就会发生。预防和处理这种冲突是自选路由交换机设计的一个主要挑战。榕树网络很巧妙地安排了一个 2×2 交换单元阵列，使得在分组按升序排列的情况下，它能把所有分组无冲突地发送到正确的输出端口。

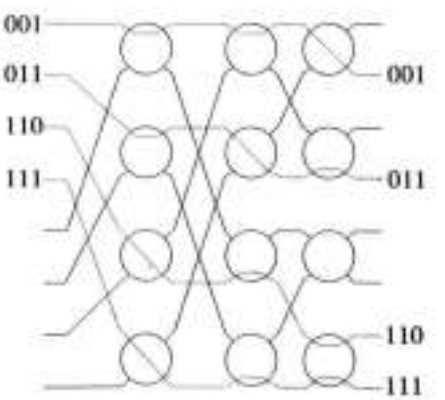


图 3-42 通过榕树网络进行分组路由。3 位的数字表示四个接收到的分组首部的值

从图 3-42 所示的例子中我们可以了解它是如何工作的，这里自选路由首部包含的输出端口号以二进制编码。第一列的交换机单元将检查位于自选路由首部中输出端口号的最高有效位，如果为 0 则向上路由分组，为 1 则向下路由分组。第二列的交换机单元检查首部中的第二位，最后一列的交换单元将检查最后一个有效位。从这个例子中可以看到，分

组被无冲突地传送到正确的目的端口。注意从第一列交换单元的上方输出的分组怎样到达网络的上半部分，这样，使用端口号为0～3的分组正确进入网络的一半部分，第二列又使分组正确进入网络的四分之一部分，最后一列使分组传送到正确的输出端口。巧妙之处是交换单元的结构编排避免了冲突。在网络开始部分的编排方式使用了“完全混合”的布线模式。要使用榕树网络建立完整的交换网状结构需要一个附加的部分，它用于在分组送到榕树网络之前对它们进行排序。Batcher-榕树交换机的设计是这种方法的一个著名的例子。Batcher网络也是由规则的2×2交换单元互联组成，用来把分组按降序排列。分组在离开Batcher网络时，将被送到正确的输出端口，使得它们在榕树网络中不发生冲突。

交换机设计中一个有意思的事情是用相同的基本技术可以制造出类型很不相同的交换机。例如，本章讨论的以太网交换机和ATM交换机，以及下一章讨论的因特网路由器，都是用本节给出的设计思想建立的。

3.4.4 路由器的实现

我们已经看到了建造一台交换机的多种方法，从一个有适当数目的网络接口的通用工作站到某些复杂的硬件设计。一般来说，建造路由器也有同样的选择范围，大多数情况如图3-43所示。控制处理器负责运行路由协议并通常充当路由器控制的核心。交换网状结构将分组从一个端口传送到另一个端口，就像在交换机中一样，而端口提供的功能是允许路由器与各种类型的链路（如以太网、SONET等）连接。

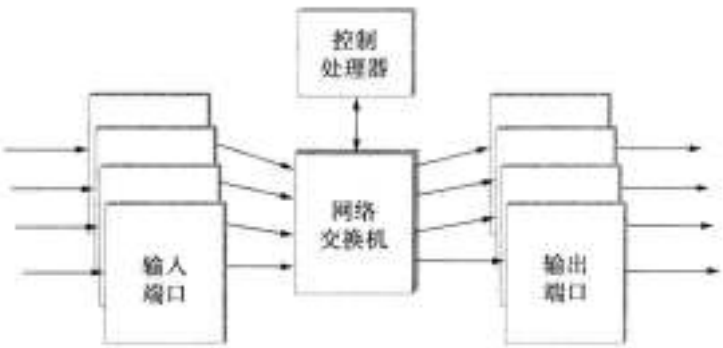


图3-43 路由器框图

关于路由器设计与交换机设计之间的不同之处，有几点值得注意。首先，路由器必须能够处理可变长度的分组，这个限制不适用于ATM交换机，但无疑适用于以太网或帧中继交换机。实际上很多高性能的路由器都被设计成基于信元的交换结构。在这种情况下，端口必须能够将可变长度的分组转换为信元，并再由信元转换为分组。这就是分割和重组（SAR），在ATM网络适配器中同样要面对这一问题。

可变长度的IP数据报所产生的另一个影响是使得描述路由器的性能比描述只转发信元的交换机的性能更为困难。路由器通常能在每秒钟之内转发固定数目的分组，这意味着，按比特/秒衡量的总吞吐量依赖于分组大小。路由器设计者通常需要选择在线速率（line rate）下支持的分组长度。也就是说，如果pps（分组/秒）是到达一个特定端口的分组能够被转发的速率，且linerate是按比特/秒计算的端口的物理速率，那么按比特计算的packetsize满足

$$packetsize \times pps = linerate$$

这就是在线速率下路由器能转发的分组大小。对较长的分组很可能维持线速率，但对较短的分组则不可能。有时候，设计者可能决定支持的合适分组尺寸为 40 字节，因为这是带有 TCP 首部的 IP 分组的最小尺寸。另一个选择可能是预期的平均分组尺寸，可通过研究网络通信量的轨迹来确定。例如，因特网主干网的衡量标准建议平均 IP 分组大约为 300 字节。然而，当遇到一长串短分组时，路由器就跟不上这个速度，并可能开始丢弃分组，从统计上看，这种情况时有发生，如果在路由器遭到主动攻击时（见第 8 章）则是很有可能的。此类设计在很大程度上依赖于成本考虑和路由器预期的应用。

在执行转发 IP 分组任务时，路由器可大致描述为具有集中式（centralized）或分布式（distributed）特征的转发模型。在集中式模型中（如本节先前所述），IP 转发算法在一个处理所有端口通信量的单一处理引擎中完成。在分布式模型中有多个处理引擎，可能每个端口一个，更常见的情况是每个线卡一个，这里一个线卡可以是一个或多个物理端口服务。每种模型各有其优缺点。在同等条件下，因为分布式转发模型在整体上有更强的处理能力，所以总的看来通过路由器每秒可转发更多分组。但是，分布式处理模型也使软件体系结构更加复杂，因为每个转发引擎通常都需要一份自己的转发表副本，这就使控制处理器需要保证始终及时地更新转发表。

路由器实现路由与交换机转发有很大的不同，另一方面的因素是 IP 转发算法本身。在网桥与大部分 ATM 交换机中，转发算法只包含在转发表中寻找固定长度的 ID（MAC 地址或者 VCI）、寻找正确的输出端口以及发送包到这个端口。我们在 3.2.4 节已经看到，IP 转发算法更复杂，部分原因是当转发分组大小是在 8 比特到 32 比特浮动而不是固定值时，转发一个分组需要进行相应的检验。

因为 IP 转发算法的高复杂性，IP 路由器的很多发展阶段都致力于突破基本性能限制。然而，当我们讨论本章“扩展阅读”的时候，仍然可以看到很多近年来提出的 IP 转发创新方法，在本书写作期间出现了可以每个接口转发 40Gbps IP 流量的商用路由器。在整合了很多高性能 IP 转发引擎和多种在 3.4 节介绍的可扩展交换网后，就可以建造具有 T（Terabits）级别吞吐量的路由器。这足以满足我们数年后的因特网流量增长情况。

另一种路由实现方面的有趣的技术是网络处理器（network processor）。网络处理器是可编程的标准工作站或 PC 处理器，但是针对网络任务做了更高的优化。例如，网络处理器可以有特别适于执行 IP 地址查找或计算 IP 数据报校验和的指令。这些设备可以用于路由器和其他网络设备中（如防火墙）。

关于网络处理器，人们正在争论的有趣问题之一是它能否比其他设备做得更好。例如，如果由于庞大工业的推动，传统处理器性能不断地显著提高，那么，网络处理器能跟上步伐吗？还有，一个追求通用性的服务设备，能比定制设计的 ASIC（特定应用集成电路，如只进行 IP 转发而不做其他工作）做得更好吗？对这些问题的部分回答取决于什么叫“做得更好”。例如，我们总是需要权衡硬件成本、推向市场的时间、性能和灵活性这些因素，灵活性是指路由器建造出来之后改变其所支持功能的能力。在本章余下部分及后面几章中，我们会看到对路由器功能的需求是如何变化的。在可预见的将来会存在广泛的路由器设计，而且网络处理器也会扮演某种角色。

3.5 小结

本章开始讨论通过在互联链接和网络中使用交换机和路由器来构造一个可扩展的和异构的网络时所涉及的问题。最常用的交换机使用方式是局域网互联，特别是以太网分割。局域网交换机或网桥使用源地址学习技术来改进转发效率，使用生成树算法避免回环。这些交换机被大量用于数据中心、学校及公司的网络中。

对于异构网络，网际协议（IP）的发明形成了今天的路由器。IP 为网络互联定义了一个简单通用的服务模型，这个模型基于尽力而为的 IP 分组传输而建立，致力于解决异构网络问题。这种服务模型的一个重要部分是全局地址问题，即为了相互间数据交换目的而使互联网中任意两个节点可以相互分辨。IP 服务模型的简单程度使其得到了所有已知的网络技术支持，且 ARP 机制也被用于全局 IP 地址到当地链路层地址的翻译工作。

网络互联的一项重要工作是决定互联网中到达任意目的地的有效路由问题。互联网路由算法用分布式方法解决这个问题。本章在应用实例（RIP 和 OSPF）中介绍两大类算法——链路状态算法和距离向量算法。

交换机和路由器都需要以一个较高的速率从输入端到输出端来转发分组，在有些情况下，交换机需要扩大容量来容纳几百个或几千个端口。由于争用问题，以可接受的代价建造大规模的高性能交换机是非常复杂的，因此交换机通常采用专用的硬件而不是通用的工作站来构造。

接下来会发生什么：未来因特网

因特网无疑是一个巨大的成功，很容易使我们忘记曾经有不存在因特网的时候。然而，囿于网络的可用程度，因特网的发明者只对其进行了部分开发。比如电路交换电话网并不适合因特网的需要。当前的因特网正如 20 世纪 60 年代的电话网一样被人为建立了。那么就可以问：互联网之后会是什么？

没有人当时就能知道问题的答案，但一些著名的研究正努力使某些“未来因特网”成为可能。现在很难想象当前的因特网会在很短的将来被某种东西取代（尽管如此，电话网络仍然存在，增加了传输流量并转移于互联网上）。如果能够超越当今网络增量部署思维的限制，就可以使一些创新成为可能，不然我们就会错过创新。在这种情况下非常流行谈论“清白”的研究。这种研究着眼于如果我们能从头开始的话会出现什么可能，并出于对未来的考虑而推迟部署。

例如，如果我们假设因特网上的每个节点都是移动的呢？我们可能会开始用一种不同的方式确定节点而不是 IP 地址，包括将什么样的网络节点连接到目前的信息。我们可能会使用一些其他形式的标识符。或者，作为另一个例子，我们可能会考虑把目前因特网建成一个不同的信任模型。当因特网最初被开发时，它似乎有理由默认每台主机应能发送信息给所有其他主机。但在如今这个垃圾邮件、钓鱼者和拒绝服务攻击者的世界里，人们可能会考虑不同的对新连接或未知节点具有更多受限初始能力的信任模型。这两个例子说明，正如当前我们知道一些在 20 世纪 70 年代没有出现的事情（如网络的移动性和安全的重要性）一样，我们可能要为互联网拿出非常不同的设计。

这里可以指出一些观点。首先，你不应该假设互联网已经完成了，其体系结构是可以灵活继承且将持续发展的。在下一章里我们将看到一些互联网演变的例子。另一点是有很

多种方法可用于网络研究；逐步开发可能的想法是很重要的，但正如因特网先驱大卫·克拉克的话，"设想未来，可以促进当前的进步"。

扩展阅读

对网桥尤其是对生成树算法的开创性文章是下面 Perlman 的一篇论文。不出意外地有大量关于互联网不同方面的文章，Cerl 和 Kahn 的文章是最早介绍 TCP/IP 体系结构并因其历史价值而值得阅读的。最后，关于诸多路由器设计文章中的一篇，McKeown 的文章描述一种内部使用信元的交换机设计方法，这种方法已成为商用的作为高性能路由器转发变长分组的基础。

- Perlman, R. An algorithm for distributed computation of spanning trees in an extended LAN. *Proceedings of the Ninth Data Communications Symposium*, Pages 44-53, September 1985.
- Cerf, V., and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications* COM-22 (5): 637-648, May 1974.
- McKeown, N. The iSLIP scheduling algorithm for input-queued switches. *IEEE Transactions on Networking* 7 (2): 188-201, April 1999.

在 Perlman 的另一著作 [Per00] 中可以发现有关网桥和路由器的精彩综述。这里也有丰富的关于 ATM 的文章，ATM 的先驱 Turner [Tur85] 最早提出将基于信元的网络用于集成服务模型。

很多当前因特网的核心技术和协议都在意见征求文档（RFC）中描述：子网划分在 Mogul 和 Poste [MP85] 中描述，CIDR 在 Fuller 和 Li [FL06] 中描述，RIP 第二版在 Malkin [Mal98] 中定义，OSPF 在 Moy [Moy98] 中定义。长达 200 页的 OSPF 规范是最长的 RCF 之一，但也包含众多不同寻常的关于如何实现一个协议的内容。避免 IP 碎片的原因被 Kent 和 Mogul [KM87] 验证，路径 MTU 发现技术由 Mogul 和 Deering [MD90] 描述。

一篇关于未来互联网的前瞻性研究文章是由 Clark 等人发表的 [CPB+05]。这篇文章与正在进行的研究工作都是围绕未来互联网的，下面我们将为此提供更多的参考。

有关交换机结构已发表了上千篇文章。早期很好地解释 Batcher 网络的一篇文章毫无疑问是 Batcher 自己写的 [Bat68]。排序网络是由 Drysdale 和 Young [DY75] 解释的，而一种有趣的纵横交换形式由 Yeh 等 [YHA87] 描述。Giacopelli 等 [GHMS91] 描述了"阳光"交换机，这篇文章对交换机设计中的流量分析提供了深入的见解，特别是"阳光"交换机设计师首先认识到信元有可能突发性到达交换机，从而影响了他们的设计思路。在 Robertazzi [Rob93] 中可以找到对不同交换机网状结构性能的出色综述。基于可变长度分组交换机设计的一个例子可以在 Gopal 和 Guerin [GG94] 中找到。

已经有大量的工作旨在开发可用于路由器的快速查找 IP 地址的算法（回想一下路由器需要匹配转发表的最长前缀的问题）。PATRICIA 树是最早应用于此类问题的算法 [Mor68]。更多最近的工作见 [DBCP97]、[WVTP97]、[LS98]、[SVSM98] 和 [EVD04] 中的报道。阅读 Partridge 等 [Par98] 的工作可以概览此类算法如何用于构建高速路由器。

光纤网本身就是一个丰富的研究领域，它有自己的期刊、会议等。我们推荐 Ramaswami 等 [RS01] 作为这个领域一篇很好的介绍性文章。

如果想了解有关网络性能的数学分析，那么就该读由 ARPANET 的先驱者之一

Kleinrock 写的一篇优秀著作 [Kle75]。关于分组交换中应用排队理论已发表了很多文章。我们推荐 Paxson 和 Floyd 的文章 [PF94]，重点介绍对因特网的重大贡献，并推荐由 Leland 等人写的一篇文章 [LTWW94]，介绍有关“远程依赖”的重要概念并说明许多通信量建模的传统方法的不足之处。

最后，我们推荐下列时常更新的网站作为参考：

- http://www.nets-find.net/：美国国家自然科学基金关于“未来互联网设计”研究的网站。
- http://www.geni.net/：描述 GENI 网络实验床的网站，允许创建一些之前提到的“清白”研究的地方。

习题

1. 使用图 3-44 给出的网络，在建立以下连接之后，给出所有交换机的虚电路表。假设连接的序列是累积的，即建立第二个连接时第一个连接仍未断开，以此类推。同时，假设 VCI 的分配始终选用每条链路上最低的未使用 VCI，从 0 开始。

(a) 主机 A 连接到主机 C。　　(b) 主机 D 连接到主机 B。

(c) 主机 D 连接到主机 I。　　(d) 主机 A 连接到主机 B。

(e) 主机 F 连接到主机 J。　　(f) 主机 H 连接到主机 A。

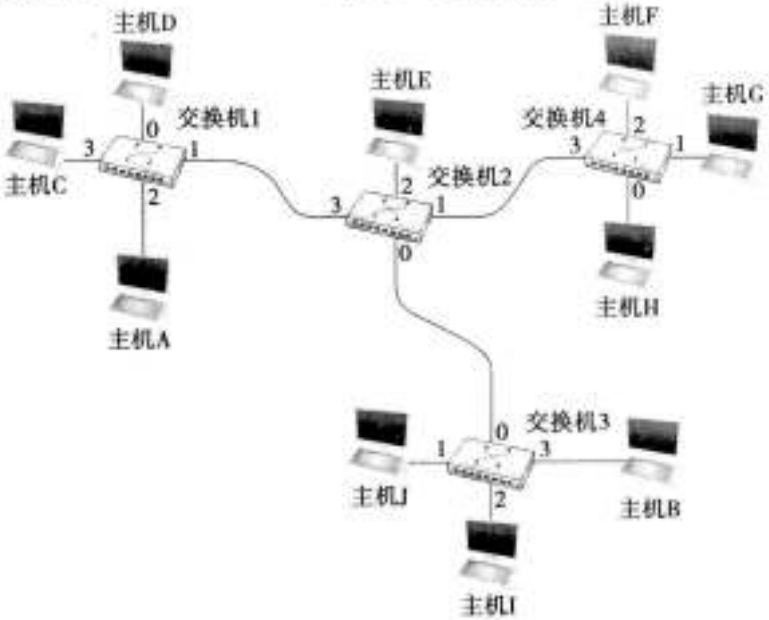


图 3-44　习题 1 和习题 2 的网络例子

2. 使用图 3-44 给出的网络，在建立以下连接之后，给出所有交换机的虚电路表。假设连接的序列是累积的，即建立第二个连接时第一个连接仍未断开，以此类推。同时，假设 VCI 的分配始终选用每条链路上最低的未使用 VCI，从 0 开始。

(a) 主机 D 连接到主机 H。　　(b) 主机 B 连接到主机 G。

(c) 主机 F 连接到主机 A。　　(d) 主机 H 连接到主机 C。

(e) 主机 I 连接到主机 E。　　(f) 主机 H 连接到主机 J。

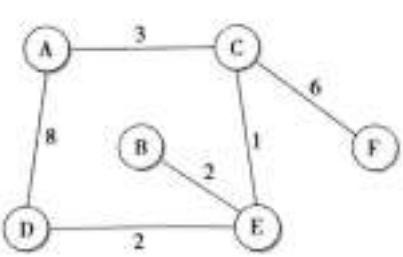


图 3-45　习题 3 的网络

3. 对图 3-45 中给出的网络，给出每个节点的数据报转发表。链路

上标有相应的成本，你的表应经由最小成本的路径把每个分组转发到目的地。

4. 给出图 3-46 中 S1～S4 交换机的转发表。每台交换机应该有一个默认路由记录，用来把未识别目的地址的分组转发到 OUT。任何与默认记录相重复的特定目的地表记录应被删除。

5. 考虑图 3-47 中的虚电路交换机。表 3-15 列出每台交换机的 VCI 表，即与其连接相关的（端口，VCI）对（或（VCI，接口）对）。连接是双向的。列出所有端点到端点的连接。

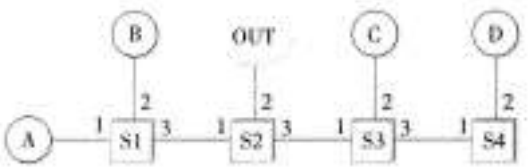


图 3-46 习题 4 框图

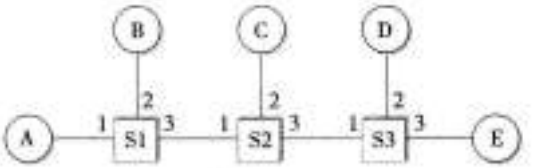


图 3-47 习题 5 框图

表 3-15 图 3-47 中交换机的 VCI 表

交换机 S1				交换机 S2				交换机 S3			
端口	VCI	端口	VCI	端口	VCI	端口	VCI	端口	VCI	端口	VCI
1	2	3	1	1	1	3	3	1	3	2	1
1	1	2	3	1	2	3	2	1	2	2	2
2	1	3	2								

6. 在 3.1.3 节的源路由选择的例子中，由 B 接收的地址是不可逆的，不能帮助 B 知道如何到达 A。提出一个对传送机制的修改方案使之可逆，机制中不应要求给出所有交换机的全局唯一名字。

7. 提出一个虚电路交换机可使用的机制，使得如果一台交换机丢失有关连接的所有状态信息，那么沿着经过那台交换机的路径上的分组的发送方被通知该交换机失效。

8. 提出一个数据报交换机可使用的机制，使得如果一台交换机丢失全部或部分转发表，受到影响的发送方会被通知交换机故障。

9. 3.1.2 节描述的虚电路机制假设每条链路是点到点的。在链路是共享介质（例如以太网）连接的情况下，扩展转发算法。

10. 假设在图 3-2 中已经添加了一条新链路，连接交换机 3 的端口 1（这里是 G）和交换机 1 的端口 0（这里是 D）。但没有向任何一台交换机通知这条链路，而且交换机 3 错误地认为主机 B 经由端口 1 是可达的。

(a) 使用数据报转发，如果主机 A 试图发送消息给主机 B，将会发生什么情况？

(b) 使用文中讨论的虚电路建立机制，如果主机 A 试图连接到主机 B，将会发生什么情况？

11. 给出路径经过某条链路两次的工作虚电路的例子，但是沿着这条路径发送的数据报不应产生无限循环。

12. 在 3.1.2 节中，每台交换机为输入链路选择一个 VCI 值。请说明每台交换机也可以为输出链路选择 VCI 值，并且同一个 VCI 值将被每种方法选择。如果每台交换机都选择输出 VCI，那么在数据被发送前，它还需要等待一个 RTT 吗？

13. 对图 3-48 中给出的扩展 LAN，指出哪些端口不会被生成树算法所选择。

14. 对图 3-48 中给出的扩展 LAN，假设网桥 B1 遭遇灾难性的故障。指出当经过恢复过程并形成新树后，哪些端口不会被生成树算法所选择。

15. 考虑图 3-49 中给出的学习型网桥的布局。假设全部网桥被初始化为空，给出经过下列传输后 B1～B4 的转发表：

- A 发送到 C。
- C 发送到 A。
- D 发送到 C。

用从某个端口直接到达的唯一的邻居来识别那个端口，就是说，B1 的端口可被标记为“A”和“E2”。

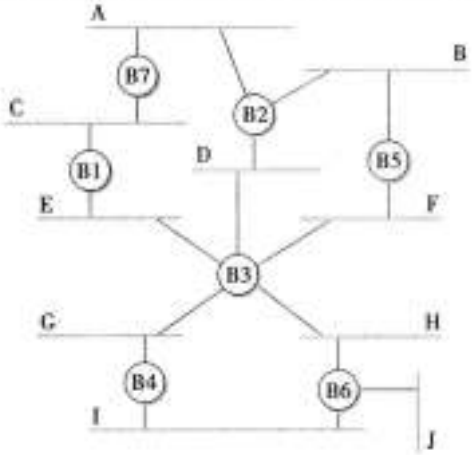


图 3-48 习题 13 和习题 14 的网络

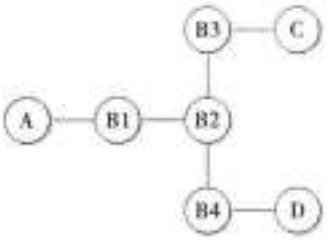


图 3-49 习题 15 和习题 16 的网络

16. 如前一道题，考虑图 3-49 中的学习型网桥。假设所有网桥被初始化为空，在经过以下的传输后，给出网桥 B1～B4 的转发表。
    - D 发送到 C。
    - C 发送到 D。
    - A 发送到 C。
17. 考虑图 3-50 中的主机 X、Y、Z、W 和带有初始化为空转发表的学习型网桥 B1、B2、B3。
    (a) 假设 X 发送到 W，哪个网桥知道 X 的位置？Y 的网络接口见到这个分组了吗？
    (b) 假设 Z 现在发送到 X，哪个网桥知道 Z 的位置？Y 的网络接口见到这个分组吗？
    (c) 假设 Y 现在发送到 X，哪个网桥知道 Y 的位置？Z 的网络接口见到这个分组吗？
    (d) 最后，假设 W 发送到 Y，哪个网桥知道 Z 的位置？W 的网络接口见到这个分组吗？
18. 给出图 3-51 中的扩展 LAN 的生成树，并讨论怎样解决任何平局问题。

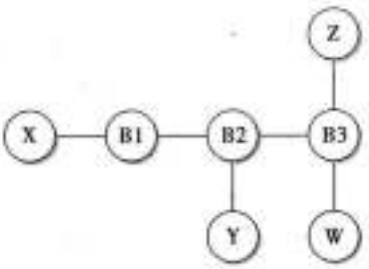


图 3-50 习题 17 框图



图3-51 习题 18 的扩展 LAN

19. 假设图 3-52 中的两个学习型网桥 B1、B2 形成一个环路，且不能实现生成树算法。每个网桥保留一个由〈地址，接口〉对组成的表。
    (a) 如果 M 发送到 L 会发生什么情况？
    (b) 假设在短时间后 L 回答 M。给出导致来自 M 的分组和来自 L 的分组以相反的方向在环路中循环的事件序列。
20. 假设在图 3-52 中 M 发送给自己（正常情况下这绝不会发生）。对于以下假设，说明将发生什么情况：
    (a) 网桥的学习算法是在搜索表中目的地址之前设置（或更新）

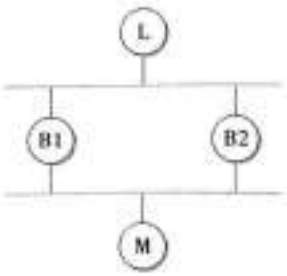


图 3-52 习题 19 和习题 20 的环

新的（源地址，接口）记录。

(b) 找到目的地址之后设置新的源地址。

21. 考虑图 3-10 中的扩展 LAN。在生成树算法中，如果网桥 B1 不参与，且在下述条件下会发生什么情况？

(a) 简单地转发所有生成树算法消息。

(b) 丢弃所有生成树消息。

22. 假设一些中继器（或集线器）而不是网桥被连接成一个环路。

(a) 当有节点传输时会发生什么情况？

(b) 为什么生成树机制很难或不可能在中继器上实现？

(c) 提出一种机制，中继器可以检测环路并关闭一些端口来切断环路。不要求你的解决方案 100%有效。

23. 假设一个网桥在同一个网络中连有两个端口。网桥怎样检测和纠正这种情况？

24. ATM 信元首部占 ATM 链路总带宽的百分比是多少？忽略信元填充或 ATM 适配层首部。

25. 信元转发方法（如 ATM）通常应用虚电路交换而不是数据报转发方式。给出解释这一问题的专题讨论（考虑过程问题）。

26. 假设一个工作站具有 800Mbps 的 I/O 总线速度和 2Gbps 的内存带宽。假设直接内存访问（DMA）用于转移数据进出内存，那么基于这个工作站的交换机能处理多少个 100Mbps 以太网链路接口？

27. 假设一个工作站的 I/O 总线速度为 1Gbps，内存带宽为 2Gbps。如果使用 DMA 对内存进行存取，那么基于这个工作站的交换机能处理多少个 100Mbps 以太网链路接口？

28. 假设一台以电脑工作站构建的交换机以每秒 500 000 个的速度转发分组，不考虑分组大小（在限制内）。假设 DMA 用于转移数据进出内存，内存带宽为 2Gbps，I/O 总线带宽为 1Gbps，分组长度为多少时总线的带宽将成为限制因素？

29. 假设一台交换机有输入和输出 FIFO 缓存。分组到达一个输入端口后被插入到它的 FIFO 的尾部。然后交换机试图转发每个 FIFO 首部的分组到适当输出端口的 FIFO 尾部。

(a) 解释在什么情况下，这样一台交换机会丢失发往一个 FIFO 为空的输出端口的分组。

(b) 这种行为叫作什么？

(c) 假设 FIFO 缓存可以被自由地重新分配，提出一个缓冲区改组过程以避免上面的问题，并解释为什么要那样做。

★ 30. 一个 $n \times n$ 的棉树网络的一级包括 $n/2$ 个 2×2 交换单元。第一级把分组正确输入到网络的一半，第二级把分组正确送到网络的 1/4 部分，如此下去，直到所有分组都被输送到正确的端口。推导构造一个 $n \times n$ 棉树网络需要多少个 2×2 交换单元的公式。用 $n=8$ 来验证你的答案。

★ 31. 描述 Batcher 网络是如何工作的（见“扩展阅读”一节）。解释怎样才能把 Batcher 网络和棉树网络结合来实现一个交换网状结构。

32. 假设在全部通信量介于一台服务器和 N 台“客户端”之间的环境中，用一台 10Mbps 交换机替换一台 10Mbps 以太网集线器（或中继器）。因为所有通信量仍需通过服务器-交换机的链路，所以名义上没有改进带宽。

(a) 你认为带宽会有任何改进吗？如果有，为什么？

(b) 交换机与集线器相比还有那些优缺点？

33. IP 地址的哪个方面使得有必要给每个接口分配一个地址，而不是给每台主机分配一个地址？根据你的答案，为什么 IP 能容忍点到点的接口有非唯一的地址或没有地址？

34. 为什么 IP 首部中的 Offset 字段要以 8 字节为单位来度量偏移量？（提示：回忆一下，Offset 字段长 13 位。）

35. 有些信令错误会导致一个分组中的所有比特被重写为全 0 或全 1。假设分组中的所有比特（包括因特网校验和）都被重写。一个全 0 或全 1 的分组是合法的 IPv4 分组吗？因特网校验和能捕获这样的错误吗？为什么？

36. 假设一条 TCP 消息包含 2 048 字节数据和 20 字节的 TCP 首部，这条 TCP 消息要传送给 IP，传送通

过因特网上的两个网络（即从源主机到一台路由器，再从路由器到目的主机）。第一个网络使用 14 字节的首部，并且有一个 1 024 字节的 MTU；第二个网络使用 8 字节的首部和 512 字节的 MTU。每个网络的 MTU 给出链路层帧能够承载的最大 IP 数据报尺寸。请给出传送到目的主机网络层的分段序列的尺寸和偏移量。假设所有 IP 首部的尺寸是 20 字节。

37. 路径 MTU 是 2 台主机之间当前路径上所有链路中最小的 MTU。假设我们可以找到上一题中那条路径的路径 MTU，并以此值作为所有路径分段的 MTU。请给出传送到目的主机网络层的分段序列的大小和偏移量。

38. 假设一个 IP 分组被分为 10 个分段，每个分片丢失的概率是 1%（独立的）。在合理的估计下，这意味着由于分片的丢失而丢失整个分组的概率是 10%。如果分组传输两次，那么整个分组丢失的概率是多少？

(a) 假设所有接收到的分片必须是同一次传送的一部分。

(b) 假设任何给定分片可能是任一次传送的一部分。

(c) 解释为什么在这里使用 Ident 字段是适当的。

39. 假设图 3-18b 中的分片都经过另一个路由器到达一条链路上，此链路具有 380 字节的 MTU，不算链路首部。请给出产生的分片。如果分组一开始就按此 MTU 分片，将产生多少个分片？

40. 最大带宽是多少时，一台 IP 主机能够在 60s 内发送 576 字节的分组且不使 Ident 字段出现回绕？假设 IP 的最大分片生存时间（MSL）是 60s，就是说，延迟的分组最多可延迟 60s 到达。如果超出此带宽将会怎样？

41. 为什么认为 IPv4 是在端点进行分片的重组，而不是在下一台路由器上进行分片的重组？为什么 IPv6 完全舍弃了分片？（提示：考虑 IP 层分片和链路层分片的区别。）

42. 将 ARP 表中各记录的超时设为 10～15 分钟是一个合理的折衷尝试。试描述超时值太小或太大时将会出现什么问题。

43. IP 当前使用 32 位的地址。如果我们重新设计 IP 使用 6 字节地址而不是 32 位地址，那么是否可以不使用 ARP？为什么？

44. 假设主机 A 和主机 B 在使用 ARP 的同一个以太网上被分配了相同的 IP 地址，B 在 A 之后启动。A 的现有连接会怎样？解释“自 ARP”（self-ARP）（启动时向网络查询自己的 IP 地址）如何解决这个问题。

45. 假设一个 IP 实现按如下算法接收目的 IP 地址为 D 的一个分组 P：

```
if (〈D 的以太网地址在 ARP 的高速缓存中〉)
  〈发送 P〉
else
  〈发送 D 的 ARP 查询〉
  〈得到应答回复后将 P 放入队列中〉
```

(a) 如果 IP 层接收到目的地址为 D 的突发分组，这个算法怎样才能不浪费资源？

(b) 草拟一个改进版。

(c) 假设当缓冲区查找失败时，就在发出一个查询后简单地丢弃 P。这个过程如何执行？（一些早期的 ARP 实现据称就是这样做的。）

46. 对于图 3-53 中给出的网络，给出当以下条件成立时的像表 3-10 和表 3-13 那样的全局距离向量表。

(a) 每个节点只知道到它直接邻居的距离。

(b) 每个节点将前一步中的信息告知了它的直接邻居。

(c) 步骤（b）再次发生。

47. 对于图 3-54 中给出的网络，给出当以下条件成立时的像表 3-10 和表 3-13 那样的全局距离向量表。

(a) 每个节点只知道到它直接邻居的距离。

(b) 每个节点将前一步中的信息告知了它的直接邻居。

(c) 步骤（b）再次发生。

48. 对于图 3-53 中给出的网络，试述链路状态算法如何建立节点 D 的路由表。

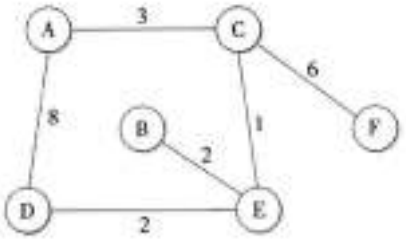


图 3-53 习题 46，习题 48 和习题 54 的网络

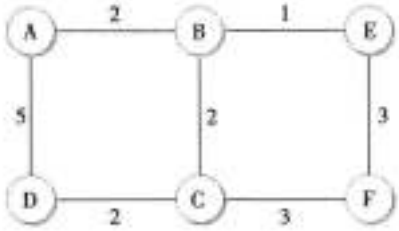


图 3-54 习题 47 的网络

49. 使用 Unix 实用程序 traceroute（Windows tracert）来确定从你的主机到因特网上其他主机（如 cs. princeton. edu 或 www. cisco. com）要经过多少跳。要想离开你的本地站点要经过多少个路由器？阅读 traceroute 手册或其他文档，并解释它是如何实现的。

50. 如果使用 traceroute 来寻找一个未指定地址的路径会出现什么情况？如果只是网络部分或主机部分未指定会怎样呢？

51. 一个站点如图 3-55 所示，R1 和 R2 是路由器，R2 连接外部网络。独立的 LAN 是以太网。RB 是网桥路由器（bridge-router），它将通信量路由给自己，并作为其他通信量的网桥。站点内部使用子网划分，每个子网中使用 ARP。不幸的是，主机 A 被错误配置且不使用子网。A 能到达 B、C、D 中的哪一个？

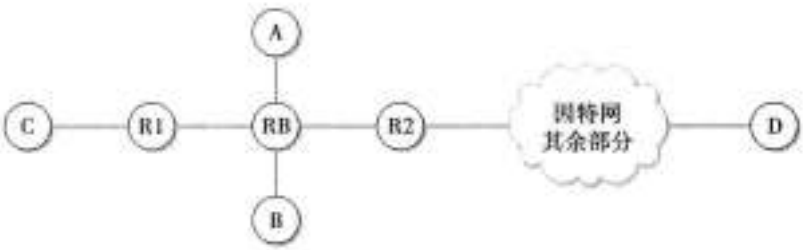


图 3-55 习题 51 的站点

52. 在一个所有链路开销均为 1 的网络中，假设我们有如表 3-16 所示的节点 A 和节点 F 的转发表，给出与这两个表相一致的最小网络的图示。

表 3-16 习题 52 的转发表

A			F		
节点	开销	下一跳	节点	开销	下一跳
B	1	B	A	3	E
C	2	B	B	2	C
D	1	D	C	1	C
E	2	B	D	2	E
F	3	D	E	1	E

53. 在一个所有链路开销均为 1 的网络中，假设有如表 3-17 所示的节点 A 和节点 F 的转发表，给出与这两个表相一致的最小网络的图示。

表 3-17 习题 53 的转发表

A			F		
节点	开销	下一跳	节点	开销	下一跳
B	1	B	A	2	C
C	1	C	B	3	C
D	2	B	C	1	C
E	3	C	D	2	C
F	2	C	E	1	E

54. 对于图 3-53 中给出的网络，假设按照习题 46 建立所有的转发表，然后 C—E 链路出错。给出：
(a) C 和 E 报告出错信息后，A、B、D 和 F 的表。
(b) A 和 D 在它们下一次互相交换信息之后的表。
(c) A 与 C 交换信息后，C 的表。

55. 假设一个路由器建立了如表 3-18 所示的路由表。这个路由器可以直接通过接口 0 和接口 1 传送分组，或者可将分组转发往路由器 R2、R3 或 R4。请描述当分组的目的地址为以下地址时，此路由器将怎么做。
(a) 128.96.39.10。 (b) 128.96.40.12。
(c) 128.96.40.151。 (d) 192.4.153.17。
(e) 192.4.153.90。

表 3-18 习题 55 的路由表

子网号	子网掩码	下一跳
128.96.39.0	255.255.255.128	接口 0
128.96.39.128	255.255.255.128	接口 1
128.96.40.0	255.255.255.128	R2
192.4.153.0	255.255.255.192	R3
<默认>		R4

√ 56. 假设一个路由器建立了如表 3-19 所示的路由表。这个路由器可以直接通过接口 0 和接口 1 传送分组，或者可将分组转发往路由器 R2、R3 或 R4。假设路由器实现最长的前缀匹配。请描述当分组的目的地址为以下地址时，此路由器将怎么做。
(a) 128.96.171.92。 (b) 128.96.167.151。
(c) 128.96.163.151。 (d) 128.96.169.192。
(e) 128.96.165.121。

表 3-19 习题 56 的路由表

子网号	子网掩码	下一跳
128.96.170.0	255.255.254.0	接口 0
128.96.168.0	255.255.254.0	接口 1
128.96.166.0	255.255.254.0	R2
128.96.164.0	255.255.252.0	R3
<默认>		R4

★ 57. 考虑图 3-56 中的简单网络，其中 A 和 B 互换距离向量路由信息。所有链路开销均为 1。假设 A-E 链路出错。



图 3-56 习题 57 的简单网络

(a) 给出导致 A 和 B 之间路由循环的路由表更新序列。
(b) 估计情况 (a) 的概率，假设 A 和 B 每次分割以同样的平均速率随机地发送路由更新消息。
(c) 如果 A 在发现 A-E 出错的 1s 内广播一个更新报告，而 B 一成不变地按每 60s 广播一次，估计形成循环的概率。

58. 考虑图 3-29 所示的网络，当 A-E 链路出错时产生路由循环的情况。列出 A、B 和 C 中所有与目的地 E 有关并导致循环的表更新序列。假设一次做一个表更新，所有参与者都遵照水平分割技术，并且，A 在 C 之前对 B 发送关于 E 不可达的初始报告。你可以忽略不会引起变化的更新。

59. 假设一组路由器都使用水平分割技术，这里我们考虑如果它们还使用反向抑制技术，那么在什么情况下会有所不同。
(a) 假设相关主机使用水平分割，说明反向抑制对 3.3.2 节描述的两个例子中路由循环的演变没有影响。
(b) 假设使用水平分割技术的路由器 A 和 B 由于某种原因到达了一种状态，在这种状态下，它们将一给定目标 X 的通信量互相转发。试分别描述在使用和不使用反向抑制时这种情况将如何发展。
(c) 给出即便使用反向抑制，仍会使 A 和 B 陷入如 (b) 的循环状态的事件序列。(提示：假设 B 和 A 经一个非常慢的链路相连。它们都经过第三个节点 C 到达 X，并同时向对方通知其路由。)

60. 抑制 (hold down) 是另一种避免距离向量循环的技术。主机在一段时间内忽略更新消息，直到有机会传播链路失败消息为止。考虑图 3-57 的网络，除了 E-D 链路的开销为 10 之外，其余所有链路的开销均为 1。假设 E-A 链路崩溃，并且 B 随后立即向 A 报告其环形的 E 路由 (这是一条经 A 的错误路由)。说明抑制解释的细节，并用它描述两个网络中路由循环的演变。在 EABD 网络中没有延迟发现可替代路由的情况下，怎样扩展抑制技术才能避免 EAB 网络中的循环？

61. 考虑如图 3-58 的网络，使用链路状态路由。假设 B-F 链路发生故障，并且依次发生以下事件：
(a) 节点 H 通过一个连接加到 G 的右端。
(b) 节点 D 通过一个连接加到 C 的左端。

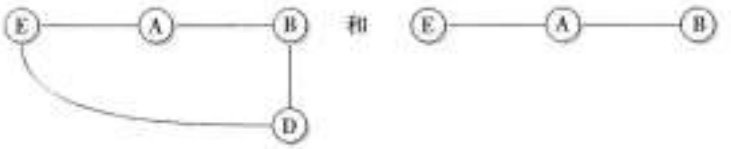


图 3-57 习题 60 的网络

(c) 增加一条新链路 D-A。

现在发生故障的 D-F 链路被恢复。请描述哪些链路状态分组将来回扩散。假设所有节点中的初始序号均为 1，且无分组超时，一条链路的两个端点在链路的 LSP 中使用同样的序号，并大于以前使用过的所有序号。

62. 在如图 3-59 所示的网络中，请给出前向搜索算法在建立节点 A 的路由数据库时如表 3-14 的步骤。

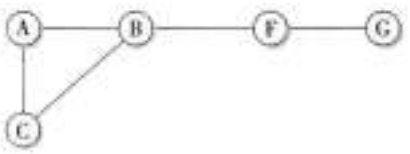


图 3-58 习题 61 的网络

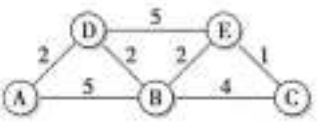


图 3-59 习题 62 的网络

63. 在如图 3-60 所示的网络中，请给出前向搜索算法在建立节点 A 的路由数据库时如表 3-14 那样的步骤。

64. 假设图 3-61 所示网络中的节点参与链路状态路由。C 接收到互相矛盾的 LSP：来自于 A 的声明 A—B 链路不可用，而来自于 B 的声明 A—B 链路可用。

(a) 这是怎么发生的？

(b) C 应怎么做？C 能够希望什么？

不假设 LSP 包含任何同步时间戳。

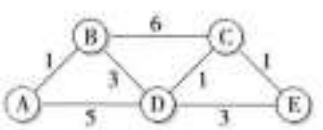


图 3-60 习题 63 的网络

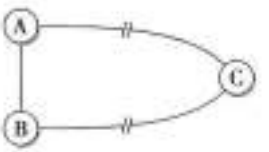


图 3-61 习题 64 的网络

65. 假设 IP 路由器学习有关 IP 网络和子网信息的方法和以太网网桥学习主机信息的方法一样：都是通过关注新主机的出现和它们的分组到达的接口。将这种情况与现有的距离向量路由器得知以下信息时的情况相比较：

(a) 一个叶子站点有到因特网的连接。

(b) 在不连到因特网上的一个组织的内部使用。

假设路由器只通过其他路由器接收新网络通知，且最初的路由器通过配置接收它们的 IP 网络信息。

66. 没有指定路由器的 IP 主机常常将地址错误地指向自己的分组丢弃，即使它们有能力正确转发这些分组。假如没有这个要求，如果一个地址指向 IP 地址 A 的分组被无意地在链路层广播，会出现什么情况？你认为这个要求的存在还有什么其他的正当理由吗？

67. 阅读 Unix/Windows 实用程序 netstat 的手册或其他文档。使用 netstat 显示你的主机上的当前 IP 路由表，解释每一条记录的目的。实际的最小记录数是多少？

68. 一个组织有一个 C 类网络 12.1.1/24，并希望建立 4 个部门的子网，其中部门 A 有 75 台主机、B 有 35 台主机、C 有 20 台主机、D 有 18 台主机，共有 148 台主机。

(a) 给出可以实现这个目标的子网掩码的一个可能的排列。

(b) 设想如果部门 D 增加到 32 台主机，对组织可以提出什么建议。

69. 假设主机 A 和 B 在一个有 C 类 IP 网络地址 200.0.0/24 的以太网 LAN 中。现在希望经过直接到 B 的连接将主机 C 连到网络上（见图 3-62）。试解释如何使用子网做这件事，并给出子网设置的实例。假设不能使用另外的网络地址。这将会对以太网 LAN 的大小有什么影响？

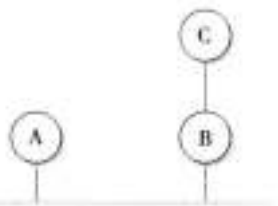


图 3-62 习题 69 的网络

70. 习题 69 中连接主机 C 的另一种方法是使用代理 ARP（proxy ARP）和路由：B 同意为去往 C 和来自 C 的通信量进行路由，并且也回答从以太网接收的对 C 的 ARP 查询。

(a) 当 A 使用 ARP 定位然后再发送一个分组到 C 时，给出所有携带物理地址的分组。

(b) 给出 B 的路由表。它必须具有什么特点？

71. 假设两个子网共享同一个物理 LAN，每个子网上的主机可看到另一个子网上的广播分组。

(a) 如果在共享 LAN 上并存两台服务器，每个子网上一个，那么 DHCP 将如何进行？将可能出现什么问题？

(b) 这种共享会影响 ARP 吗？

72. 表 3-20 是一个使用 CIDR 的路由表，地址字节为 16 进制。C4.50.0.0/12 中的“/12”表示网络掩码的前 12 位是 1，即 FF.F0.0.0。注意，最后 3 个记录覆盖每个地址，因此可代替一条默认路由。请说明下列地址将被传送到的下一跳各是什么：

(a) C4.5E.13.87。　(b) C4.5E.22.09。

(c) C3.41.80.02。　(d) 5E.43.91.12。

(e) C4.6D.31.2E。　(f) C4.6B.31.2E。

表 3-20 习题 72 的路由表

网/掩码长度	下一跳
C4.50.0.0/12	A
C4.5E.10.0/20	B
C4.60.0.0/12	C
C4.68.0.0/14	D
80.0.0.0/1	E
40.0.0.0/2	F
00.0.0.0/2	G

73. 表 3-21 是一个使用 CIDR 的路由表，地址字节为 16 进制。C4.50.0.0/12 中的“/12”表示网络掩码的前 12 位是 1，即 FF.F0.0.0。请说明下列地址将被传送到的下一跳各是什么：

(a) C4.4B.31.2E。　(b) C4.5E.05.09。

(c) C4.4D.31.2E。　(d) C4.5E.03.87。

(e) C4.5E.7F.12。　(f) C4.5E.D1.02。

表 3-21 习题 73 的路由表

网/掩码长度	下一跳
C4.5E.2.0/23	A
C4.5E.4.0/22	B
C4.5E.C0.0/19	C
C4.5E.40.0/18	D
C4.4C.0.0/14	E
C0.0.0.0/2	F
80.0.0.0/1	G

74. 一个 ISP 有一个/16 前缀（旧的 B 类地址），基于 CIDR 方法将一部分地址分配给一家新公司。新公司网络中三个部门的机器需要 IP 地址：工程部、市场部和销售部。这三个部门计划中的增长如下：工程部在第一年开始时有 5 台机器，此后每周增加一台；市场部最多需要 16 台机器；销售部的每两个客户需要一台机器。第一年开始时，公司没有客户，但是销售模式指出到第二年开始时，公司将有 6 个客户，并且，此后每周增加一个新客户的概率为 60%，失去一个客户的概率为 20%，或者以 20%的概率维持原数目不变。

(a) 如果市场部使用所有 16 位地址，并且销售部和工程部像计划预期的那样，那么，至少在 7 年内，支持此公司增长计划的地址范围是什么？

(b) 这样的地址分配可以维持多长时间？当公司的地址空间用完时，如何给三个部门分配地址？

(c) 如果在 7 年计划内不使用 CIDR 编址方法，那么，新公司还有什么得到地址空间的选择？

75. 试提出一个包含不同长度前缀的 IP 转发表的查找算法，要求不需要对整个表进行线性搜索就能找到最长匹配。

高级网络互联

任何表面的平等背后都隐藏着等级。

——梅森·库利

问题：扩展到数十亿节点

我们已经看到如何创建一个包含多种类型网络的互联网。这里，我们处理了异构性（heterogeneity）问题。网络互联中的第二个关键问题是规模（scale）——这可以说是所有网络的基础问题。为了理解规模问题，有必要考查因特网的发展过程，30 年来，因特网的规模几乎每年成倍地增长。这样的发展迫使我们面对很多挑战。

如何在有几百万个甚至几十亿个节点的网络中找到一条高效的路径？正如我们在本章看到的，很多解决路由扩展能力的方法都依赖于引入层次结构。我们可以在域内以区的形式引入层次，也可以利用层次结构扩展域间路由系统。促使互联网扩展为当前规模的域间路由协议是边界网关协议（BGP）。我们将讨论 BGP 的工作过程及 BGP 所面临的因特网持续扩展带来的挑战。

与路由可扩展能力密切相关的问题是编址。20 年前 IPv4 的 32 位地址即将用尽的问题已经显现出来。因为第 5 版本的 IP 已经在早期实验中用过，所以这也促成了新的第六版本（v6）IP 的确定。IPv6 不仅从根本上扩展了地址空间，同样也增加了很多新特性。其中一些新特性是从 IPv4 中改进而来。

随着因特网的持续增长，一些功能也随之发展。本章最后一节涵盖了一些显著的增强因特网能力的内容。首先是多播，这是一种基础服务模型的增强，即高效传输同类分组到一组接收端的能力。我们将介绍如何将多播融入互联网，并讨论多种支持多播的路由协议。第二个增强点是多协议标签交换（MPLS），它改变了 IP 网络的转发机制。此修改使 IP 的路由方式和 IP 网络提供的服务产生了一些变化。最后，我们讨论移动性对路由的影响，并描述为了支持移动主机和路由器而对 IP 进行的增强。在讨论各项增强内容的同时，可扩展性仍然不可或缺。

4.1 全球互联网

至此，我们已经知道如何将一些异构的网络相连构成互联网，以及如何使用简单的 IP 地址的层次结构在一个可进行一定扩展的互联网中进行路由。我们之所以说可进行“一定的”扩展，是因为即使每台路由器不需要知道连到互联网上的所有主机，但在至今描述的模型中，它还是有必要知道连在互联网上的所有网络。今天的因特网上连有几万个网络，我们讨论过的路由协议还不足以应付这样的规模。本节讨论几种大幅度提高可扩展性的技术，这些技术使因特网发展到当前的程度。

在介绍这些技术之前，我们首先要对全球因特网的形态有一个基本了解。它不只是以太网的随意互联，反之，它所呈现的形态反映出其中互联了许多不同的组织机构。图 4-1

给出了1990年时因特网状态的一个简单描述。从那以后，因特网的拓扑发展得越来越复杂，4.1.2节和图4-4更准确地描述了当前的因特网，但是现在我们还是使用图4-1来讲述。

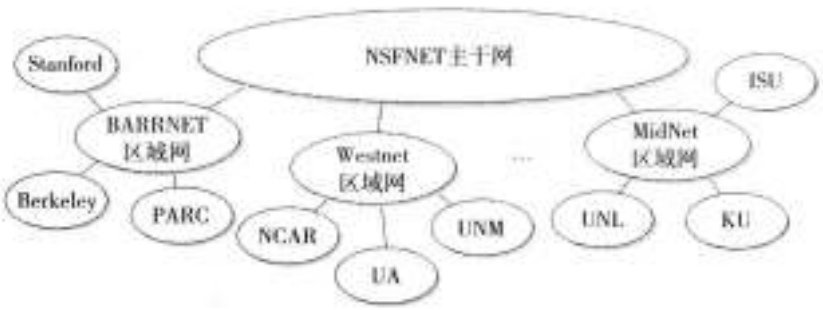


图4-1 1990年时因特网的树形结构

这个拓扑结构突出的特征之一是它由连接到服务提供商（ISP）网络（例如，BARRNET网是一个服务于旧金山湾区节点的提供商网络）的终端用户站点（例如斯坦福大学）组成。1990年，很多提供商都服务于有限的地理区域，因此称作区域网（regional network）。继而，区域网又由一个全国范围的主干网相连。1990年，美国国家科学基金会（National Science Foundation，NSF）提供资金建立了这个主干网，称之为NSFNET主干网（backbone）。尽管在图4-1上没有详细显示，但提供商网络通常由大量连接到路由器上的点到点链路（如以前的T1和DS3，今天的OC-48和OC-192 SONET链路）构成；类似地，每个终端用户站点通常不是一个单一网络，而是由多个通过路由器和网桥相连的物理网络组成。

注意，在图4-1中，每个提供商和终端用户都可能是管理上的独立实体，这就产生了一些有关路由的重要结果。例如，很可能不同的提供商对其网络中所使用的最佳路由协议以及如何给网络中的链路指定度量标准都有不同的看法。因为这种独立性，通常每个提供商网络就是一个*自治系统*（autonomous system，AS）。我们将在4.1.2节更精确地定义此术语，但是现在，可将一个AS看作一个管理上独立于其他AS的网络。

因特网有清晰可辨的结构，这有助于我们解决可扩展性问题。实际上，我们需要解决两个有关可扩展性的问题。第一个是路由的可扩展性问题。我们需要找到减少路由协议中携带的和路由器的路由表中存储的网络号数目的方法。第二个是地址利用问题，即确保IP地址空间不会被过快地消耗。

通过本书，我们将一次又一次地了解用来改进可扩展性的层次结构的原理。我们曾在前面章节里看到IP地址的层次化结构对路由可扩展性的改善，特别是无类别域间路由（CIDR）和子网。在下面两节里，我们可以看到在未来应用中利用层次化（以及聚集）来提供更强的扩展能力，包括域内及域间。最后一节考查新兴的IPv6标准，它的创建很大程度上是考虑可扩展性的结果。

4.1.1 路由区

作为第一个使用层次化扩展路由系统的例子，我们来看连接状态路由协议如何用于将一个路由域划分为称为区（area）的子域，如OSPF和IS-IS（不同协议中用的术语稍有不

同，这里，我们使用 OSPF）。通过在层次结构中加入这个额外的层，我们使单个域变得更大而不会使域内路由协议负担过重。

区是从管理上配置成相互交换链路状态信息的路由器的集合。主干网区是一个特殊的区，也称为区 0。一个划分成区的路由域的例子如图 4-2 所示。路由器 R1、R2 和 R3 是主干网区的成员，它们至少也是一个非主干网区的成员，R1 实际上是区 1 和区 2 的成员。一台既是主干网区也是非主干网区成员的路由器是区边界路由器（ABR）。注意这些与 AS 边界上的路由器有所区别。

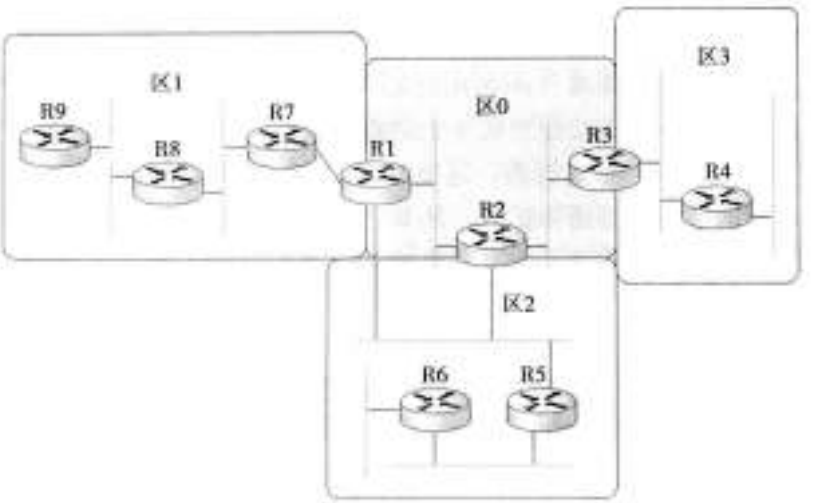


图 4-2 一个分为区的域

区中的路由如 3.3.3 节所述。区中的所有路由器彼此间发送链路状态通知，并由此发展出一个完整且一致的区映像图。然而，非区边界路由器的链路状态通知不会离开产生它们的区。这使扩散和路由计算进程具有更为显著的可扩展性。例如，区 3 中的路由器 R4 永远看不到区 1 中路由器 R8 的链路状态通知。结果，它不会知道除自己所在区以外的任何区的详细拓扑。

那么，区中的一台路由器如何确定一个目标为另一个区中网络的分组的正确下一跳？如果我们把从一个非主干网区到另一个非主干网区的分组路径分为三部分，答案就变得清楚了。首先，分组从源网络传输到主干网区，然后穿过主干网，再从主干网传输到目的网络。为了做到这一点，区边界路由器汇总从一个区中了解到的路由信息，使其能够在给其他区的通知中使用。例如，R1 收到从区 1 中所有路由器发来的链路状态通知，并因此能够确定到区 1 中任何网络的开销。当 R1 向区 0 发送链路状态通知时，它通知到达区 1 中网络的开销，就好像那些网络是直接连在 R1 上一样。这使区 0 中的所有路由器能够了解到达区 1 中所有网络的开销。然后区边界路由器汇总这些信息，并将其通知到非主干网区。这样，所有路由器都知道如何到达域中的所有网络。

注意在区 2 中有两个 ABR，因此区 2 中的路由器将不得不选择其中的一个用于到达主干网区。这很简单，由于 R1 和 R2 将通知到达不同网络的开销，因此，区 2 中路由器运行最短路径算法后，哪一台路由器是更好的选择就会变得清楚。例如，对于区 1 中的目的地来说，显然 R1 将是比 R2 更好的选择。

当把一个域划分为区时，网络管理员在可扩展性与路由优化性之间做出权衡。区的使

用使得所有从一个区到另一个区的分组必须通过主干网区行进，即使存在一条更短的可用路径。例如，即使 R4 和 R5 直接相连，分组也不能在它们之间传递，因为它们在不同的非主干网区。由此证实，可扩展性的需求通常比使用绝对最短路径的需求更为重要。

结论 这说明了网络设计中的一个重要原则。在某种优化性和可扩展性之间经常要做出权衡。当引入层次性后，信息对网络中一些节点是隐藏的，从而限制了它们做出完美优化选择的能力。然而，对可扩展性来说信息隐藏是必需的，因为这可以避免所有节点知道全局信息。在大型网络中，可扩展性永远是比完美优化性更迫切的设计目标。

最后，我们注意到网络管理员使用一个技巧来更灵活地决定哪一台路由器进入区 0。这个技巧使用路由器之间虚链路（virtual link）的思想。这条虚链路通过配置一个非直接连到区 0 上的路由器与一个直接连到区 0 上的路由器之间交换主干网路由信息来获得。例如，R8 和 R1 之间可配置一条虚链路，这使 R8 成为主干网的一部分。现在，R8 将参与和区 0 中其他路由器的链路状态通知扩散。从 R8 到 R1 的虚链路开销通过区 1 中发生的路由信息交换来确定。这项技术有助于提升路由的优化。

4.1.2 域间路由（BGP）[⊖]

本节一开始我们就引入了这样的概念，可按自治系统（AS）来组织因特网，每个自治系统（AS）在一个单独的管理实体的控制之下。一个复杂的公司内部网络可以是一个 AS，因特网的一个服务提供网络也可以是一个 AS。图 4-3 显示了有两个自治系统的简单网络。

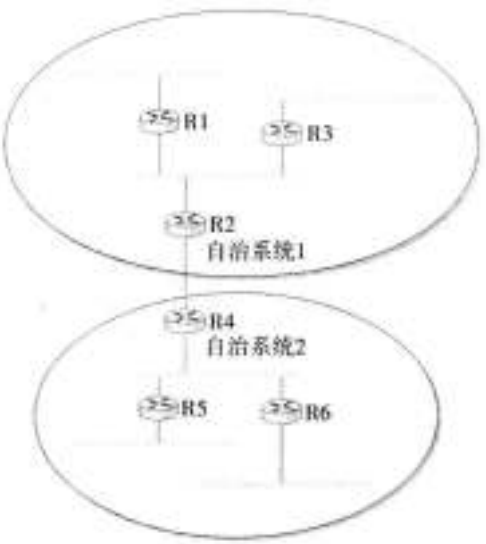


图 4-3 有两个自治系统的网络

自治系统的基本思想是提供将一个大型互联网中的路由信息进行分层聚合的一种补充方法，以提高可扩展性。现在，我们将路由问题划分为两部分：单个自治系统内的路由和自治系统间的路由。由于在因特网中自治系统的另一个名字是路由域（domain），因此我们称路由问题的两部分分别为域间路由和域内路由。此外，为了提高可扩展性，AS 模型将发生在各个不同 AS 中的域内路由分离。这样，每个 AS 都能够运行自己选择的任何域内路由协议。如果它希望的话，甚至可以使用静态路由或多协议。这样，域间路由问题就成为使不同 AS 彼此间共享可达性信息的问题，即共享通过给定的 AS 可达的 IP 地址配置信息。

1. 域间路由的挑战

域间路由当前所面临的最大挑战恐怕是每个 AS 都需要确定自己的路由策略（policy）。在一个特定的 AS 中，一个简单的路由策略实现实例为：只要可能，都优先通过 AS

⊖ 参见“实验八”。

"X" 而不是 AS "Y" 来发送流量，只有在 AS "Y" 是唯一路径的时候使用 AS "Y"，且从不允许从 AS "X" 到 AS "Y" 产生流量，反之亦然。当我们付费使用 AS "X" 和 AS "Y" 以将自己的 AS 连接到因特网，且 AS "X" 是优先连接的提供商而 AS "Y" 作为后备时，上述实例将成为一个典型的策略。因为当 AS "X" 和 AS "Y" 都作为提供商时（假设我们出资让它们扮演这样的角色），不希望跨过我们的网络来承载它们之间的流量（这叫作中转（transit）流量）。我们连接的 AS 越多，采用的策略就越复杂，特别是当所考虑的主干网提供商与几十个其他提供商或者成百的用户连接，且与每一个连接都采用不同的商业安排时（这些安排影响路由策略）。

域间路由的一个关键设计目标是支持类似上述例子的策略或者更为复杂的路由策略。使这个问题变得更困难的是我需要在没有任何其他自治系统帮助的情况下实现这样的策略，并且要面对可能的错误配置或者其他自治系统的恶意行为。除此以外，经常需要隐藏（private）策略，因为运行自治系统的个体（大部分是服务提供商）常相互竞争且不希望自己的商业安排被公开。

在因特网近些年的历史中，有两种主要的域间路由协议。第一种是外部网关协议（Exterior Gateway Protocol，EGP）。EGP 有很多局限性，其中最严重的可能是严重限制了因特网的拓扑结构。EGP 基本上要求因特网为树形拓扑结构，或者更确切地说，它是在因特网树形拓扑结构的基础上设计的，如图 4-1 所示。EGP 不允许更通用的拓扑结构。注意，在简单的树形结构中，只有一个主干网，并且与自治系统之间的连接是父子关系而不是对等关系。

代替 EGP 的是边界网关协议（Border Gateway Protocol，BGP），编写本书时 BGP 有了第 4 版（BGP-4）。BGP 比较复杂。本节给出 BGP-4 的要点。BGP 一直被认为是互联网最复杂的部分之一。我们在此对其中一些重要部分进行讨论。

与之前的 EGP 不同，BGP 并不假设自治系统的互联方式，它们可形成任意图形。这种模型显然是非常通用的，足以适应非树形结构的互联网络，像今天的多主干因特网一样，图 4-4 给出它的简化图示。（我们下面仍然会看到有一些互联网结构不像树这样简单，且 BGP 对此类结构不做任何假设。）

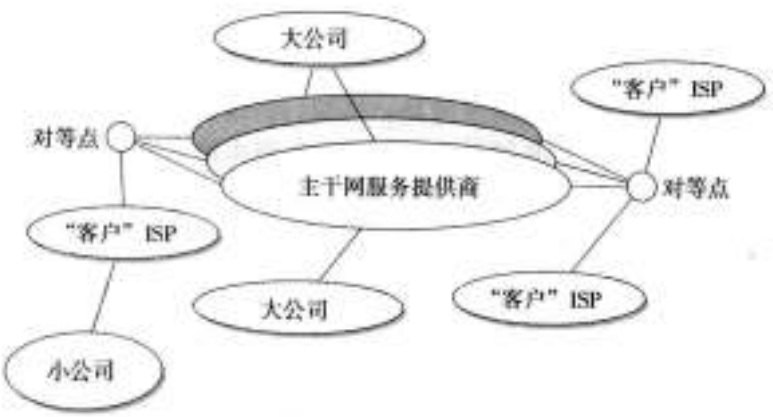


图 4-4 简单的多提供商因特网

与图 4-1 所示的简单树形结构的因特网不同，同样与图 4-4 的结构也不同，今天的因特网由互联的多个主干网组成，这些网络大多由一些私有公司而不是政府所运营。很多互

联网服务提供商（ISP）的存在主要是为了向“客户”（即家中有 PC 的个人）提供服务，而其他 ISP 提供的服务与传统的主干网服务更为相似，即连接到其他提供商或大型公司。通常，很多提供商在唯一的对等点（peering point）上彼此互联。

为了更加清楚地了解如何在复杂的自治系统互联中管理路由，我们从定义一些专有名词开始。我们将局部流量（local traffic）定义为在 AS 内的节点上起止的通信量，并且将中转流量（transit traffic）定义为穿过一个 AS 传送的通信量，并把 AS 分为三类：

- 桩 AS：与另一个 AS 只有一个连接，这样的 AS 只传送局部通信量。图 4-4 中的小公司就是一个桩 AS 的例子。
- 多连接 AS：与一个以上的其他 AS 有连接，但是拒绝传送中转通信量。例如，图 4-4 上部的大公司。
- 中转 AS：与一个以上的其他 AS 有连接，并能传送局部和中转两种通信量，如图 4-4 中的主干网提供商。

尽管 3.3 节路由讨论的焦点是根据最小化某种链路度量值来找到一条最优路径，但域间路由的目标则更为复杂。首先必须找到一条无环的通往预定目的地的路径，其次，路径必须兼容沿着路径的不同 AS 的策略。就像我们已经看到的，那些策略可能非常复杂。因此，当域内关注于具有良好定义的路径开销优化问题时，域间则关注于一个更为复杂的优化问题，就是寻找最好的策略兼容（policy-compliant）路径。

域间路由之所以困难，有这样几个原因。首先是可扩展性问题。因特网主干网路由器必须能够转发目标为因特网中任何地址的分组。这就需要有一张路由表以提供对任何合法 IP 地址的匹配。虽然 CIDR 有助于控制因特网的主干网路由中携带的不同前缀的数目，但是仍不能避免大量路由信息的传递——至编写此书时大约达到 300 000 个前缀[⊖]。

域间路由面临的进一步挑战来自于域的自治特性。注意，每个域可以运行它自己的内部路由协议，并可以使用任何它选用的路径度量值设置的方案。这就意味着计算穿过多个 AS 的有意义的路径开销是不可能的。一个值为 1 000 的开销对某个提供商来说可能是一条很好的路径，而对另一个提供商来说可能是很糟糕的。因此，域间路由只通知可达性（reachability）。可达性概念基本上可以描述为“你能通过这个 AS 到达这个网络”。这就意味着在域间路由中，选择一条最优路径是根本不可能的。

域间的自组织状况引发了信任的问题。提供商 A 可能不愿意相信来自提供商 B 的某些通知，担心提供商 B 通知错误的路由信息。例如，当提供商 B 通知到因特网上任何地方的一条成功路由时，如果提供商 B 错误地配置了它的路由器或者没有足够的容量承载通信量，那么信任它将是灾难性的。

与信任相关的问题与支持上述复杂策略的需要相关。例如，我们可能希望信任一个部分提供商仅当其通知可达特定的前缀时，因此可以采用这样的策略：使用 AS “X” 到达前缀 p 或 q，当且仅当 AS “X” 通知那些前缀可达。

2. BGP 基础

每个 AS 都有一个或多个边界路由器，通过它的分组进入或离开 AS。在图 4-3 所示的简单例子中，路由器 R2 和 R4 称为边界路由器。（多年来，路由器有时也称为网关（gate-

⊖ 查看本章结尾给出的网址来确定该值的当前估计值。

way)，因此有了协议 BGP 和 EGP 的名称。）边界路由器是一个简单的拿管在自治系统之间转发分组任务的 IP 路由器。

每一个参与 BGP 的 AS 必须至少有一个 BGP "代言人"，这个代言人是一台路由器，并与其他自治系统的 BGP 代言人交流。通常会发现边界路由器也是 BGP 代言人，但边界路由器并不是一定要成为 BGP 代言人。

BGP 并不属于 3.3 节描述的两类主要的路由协议（距离向量和链路状态协议）。与这些协议不同，BGP 将以 AS 枚举列表的形式通知到达某个特定网络的完整路径（complete path）。正因为这样，BGP 常被称为路径向量（path-vector）协议。这是根据一个特定 AS 的意愿而做出如上描述的那类策略决策所必需的。而且它使得路由循环很容易被检测出来。

我们以图 4-5 所示的网络来说明这个过程。假设提供商是中转网络，而客户网络是桩。提供商 A（AS2）的 BGP 代言人可通知分配给客户 P 和 Q 的网络号的可达性信息。实际上就是 "网络 128.96、192.4.153、192.4.32 和 192.4.3 可从 AS2 直接到达"。主干网收到这条通知后，再通知 "网络 128.96、192.4.153、192.4.32 和 192.4.3 可经路径（AS1，AS2）到达"。类似地，它还可以通知 "网络 192.12.69、192.4.54 和 192.4.23 可经路径（AS1，AS3）到达"。



图 4-5 一个运行 BGP 的网络的例子

BGP 一项重要的任务是防止建立带环路径。例如图 4-6 中的网络与图 4-5 不同的地方在于，在 AS2 和 AS3 间多了一个连接，但带来的影响在于自治系统形成了闭环。假设 AS1 得知它可经 AS2 到达网络 128.96，因此，它将此事实通知 AS3，AS3 又通知 AS2。AS2 现在可断定应将目标是 128.96 的分组发往 AS3，AS3 又将分组发往 AS1，AS1 再发回到 AS2，它们将无限循环下去。这种现象可通过在路由消息中携带完整的 AS 路径来避免。这种情况下，AS2 从 AS3 接收的通往 128.96 的通知将包括一条 AS 路径〈AS3，AS1，AS2，AS4〉。AS2 看到它自己在路径中，因而得出这是一条无用路径的结论。

为了使这种闭环预防技术生效，BGP 中携带的 AS 号应该唯一。例如，在上面的例子中，如果其他 AS 不以同样方法标识自己，AS2 就只能在 AS 路径中识别出自己。AS 号是由中心授权机构分配的保证唯一性的 16 位数字。由于 16 位只允许约 65 000 个 AS，看起来似乎并不多，因此我们注意到桩 AS 不需要唯一的 AS 号，这覆盖了绝大多数非提供商网络[⊖]。

⊖ 32 位的 AS 号也已经被定义并在 2009 年左右开始使用，这样就可以确保 AS 号空间不会成为稀缺资源。

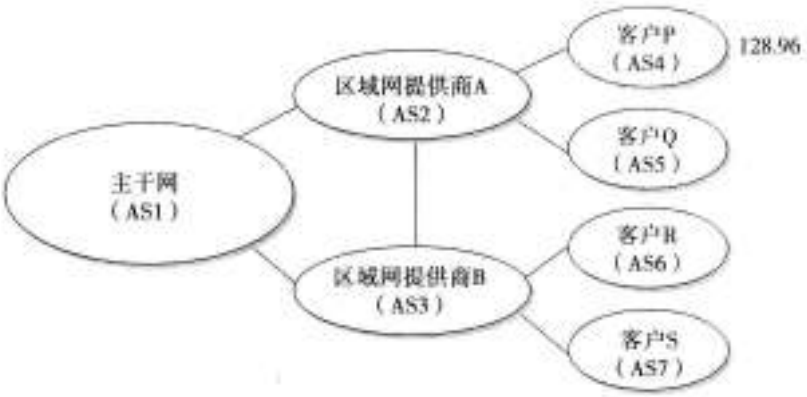


图 4-6 一个自组织系统闭环的例子

我们应该注意，一个给定的 AS 仅通知自认为是够好的路由。也就是说，如果 BGP 代言人有多条到达同一目的地的路由可供选择，那么它将根据自己的本地策略来选择最好的一条，然后这就是它通知的路由。而且，即使 BGP 代言人有一条到达某个目的地的路由，它也没有通知的义务。这就是 AS 如何实现不提供中转的策略：拒绝通知去往那些未包含在这个 AS 前缀的路由，即使它知道怎么到达这些地址。

如果一条路径上的链路出现故障或策略改变，BGP 代言人需要能够取消之前通知的路径。这由一种称为撤销路由（withdrawn route）的负通知形式来完成。正的和负的可达性信息都携带在一个 BGP 更新消息中，格式如图 4-7 所示。（注意图中的字段长度是 16 比特的倍数，这与本章中其他的分组格式不同。）



图 4-7 BGP-4 更新分组格式

不像前面章节中描述的路由协议，BGP 被定义为运行在 5.2 节描述的可靠传输协议——TCP 之上。因为 BGP 代言人可以依赖 TCP 而变得可靠，这意味着从一个代言人向另一个代言人发出的任何信息不需要重发。因此，只要不出现变化，实际上一个 BGP 代言人就可以只是偶尔发送一个“keep alive”消息，表示“我还在这里，无任何变化”。如果这台路由器崩溃，就将停止发送这类消息，从它那里获取路由的其他路由器就会知道那些路由不再有效。

3. 常见 AS 关系和策略

之前提到的策略可能非常复杂，事实证明还是有一些常见策略能够反映自治系统的关系。最常见的关系如图 4-8 所示，与之相关的三种常见关系和策略如下：

- 提供商-客户。提供商需要把客户接入网络。客户可能是一个公司，或是小规模的 ISP（他们自己本身也有客户）。因此常见策略是向客户通知已知的所有路由，并将从客户那里获取的路由信息通知所有路由器。
- 客户-提供商。在另一个方向上，客户想要通过提供商获得发送给他（或他的客户，如果他拥有客户）的流量，并且想通过提供商发送数据流量

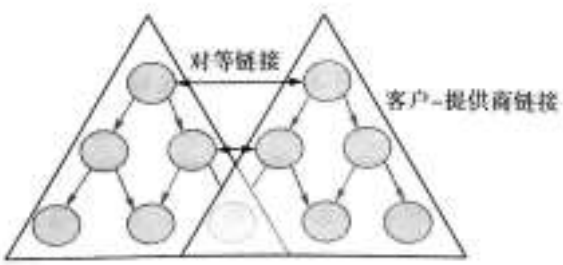


图 4-8 常见 AS 关系

到其他网络。因此这种情况下的常见策略就是向提供商通知自己的前缀和从客户那里学到的路由信息，向客户通知从提供商那里学到的路由，但是并不将从一个提供商学到的路由通知给另一个提供商。最后一步是确认客户自己没有从一个提供商向另一个提供商传递流量，因为这是支付网络流量费用的人所不愿看到的。

- 对等。第三种选择是自治系统间的对等关系。两个提供商是对等的且为对方的客户提供接入且不用向对方付费。这里的典型策略是向对等提供商通知从客户学到的路由，向客户通知从提供商学到的路由，但并不向其他提供商通知从对等方学到的路由，反之亦然。

图中有一点值得注意，即它向明显无结构的网络加入一些结构的方式。在这个层次的底层有桩网络，它是一个或多个提供商的客户，沿着层次往上走，我们可以看到其他一些提供商被作为客户的提供商。最上层的提供商有客户和对等方，但它不是其他服务商的客户。这些提供商称为 Tier-1 提供商。

结论 让我们回到现实的问题：这些方法如何帮助我们建造可扩展的网络？首先，参与 BGP 的节点数目与 AS 的数目是同一数量级，而 AS 数目比网络数目小得多。其次，找到一条好的域间路由即是找到一条通往正确的边界路由器的路径，而每个 AS 只有少数几个边界路由器。这样，我们就将路由问题简单地划分为可管理的几部分，再一次使用一个新的层次来增加可扩展性。域间路由的复杂性取决于 AS 的数量，而域内路由的复杂性取决于一个 AS 内的网络数目。

4. 集成域间路由和域内路由

前面的讨论说明了 BGP 代言人如何了解域间路由信息，却没有说明域中所有其他路由器如何得到这个信息的问题。解决此问题有几种方法。

让我们从一种非常简单而且也非常普遍的情况开始讨论。对于一个只有一点与其他 AS 连接的桩 AS 来说，显然边界路由器是 AS 外部所有路由的唯一选择。这样的路由器可以将一个默认路由（default route）注入域内路由协议。实际上，它表明任何一个在域内协议中没有被明确通知的网络都可以通过边界路由器到达。回忆一下 4.1 节对于 IP 转发的讨论，转发表中的默认项在所有指定项之后，并且与任何匹配失败的项匹配。

下一步是使从外部 AS 了解到的特定路由注入边界路由器。例如，考虑连接到客户 AS 的提供商 AS 的边界路由器。此路由器通过 BGP 或通过已经配置在其中的信息，知道网络前缀 192.4.54/24 位于该客户 AS 中。它可以将到此前缀的路由注入提供商 AS 运行的路由协议。这将是这样的一种通知："我有一条到 192.4.54/24 的开销为 X 的链路"。这将使提供商 AS 中的其他路由器知道可将目标为那个前缀的分组发往此边界路由器。

最后一级复杂性出现在主干网中，主干网从 BGP 了解到如此多的路由信息，以至于因开销太大而不能将其注入域内协议。例如，如果一个边界路由器想要注入从另一个 AS 了解到的 10 000 个前缀，它将不得不发送非常大的链路状态分组到 AS 中的其他路由器，并且最短路径的计算也将变得很复杂。因此，主干网中的路由器使用 BGP 的一种称为内部 BGP（iBGP）的变化形式，有效地将 AS 边界上 BGP 代言人了解到的信息再次分发到 AS 中所有其他路由器上。（前文讨论过 BGP 的另一种变化形式，即运行在自治系统之间的外部 BGP（eBGP）。）iBGP 使 AS 中任何路由器在发送分组到任何地址时都能够获知最好的边界路由器以供使用。同时，AS 中每台路由器明白如何在没有注入信息的情况下，

使用常规的域内协议到达每台边界路由器。通过将两类信息合并，AS 中的每台路由器都能够确定所有前缀相应的下一跳。

在图 4-9 中给出了一个简单网络来表示单个 AS，看看它是如何工作的。这里有三台边界路由器，分别是 A、D 和 E。路由器向其他 AS 发出 eBGP，并且学习如何到达不同的前缀。这三台边界路由器通过在 AS 所有路由器中建立 iBGP 会话网格来相互连接，包括与内部路由器 B 和 C 建立连接。让我们把目光集中在路由器 B 如何建立其向具有其他前缀的目标进行转发的完整视图。先看一下图 4-10 中最左侧的表格，里面显示了路由器 B 从其 iBGP 会话中学到的信息。它学到了一些经过路由器 A、D 和 E 有最佳路线的相应前缀目标。同时，所有 AS 中的路由器都在运行一些域内协议，比如路由信息协议（RIP）或开放最短路径优先（OSPF）。（域内协议中的一个典型术语是 IGP——内部网关协议。）通过完整的分段协议，B 学会了如何到达如最左侧表中所列的那些域内的其他节点。比如，要到达路由器 E，B 需要向路由器 C 发送分组。最后，在最右侧的表中，B 同时放置了所有描述，将从 iBGP 学到的外部前缀信息合成起来。这些 iBGP 信息包含从 IGP 中得到的内部路由器到边界路由器的信息。因此，如果一个像 18.0/16 这样的前缀是通过边界路由器可达的，而且最佳内部路径是通过 C 到达 E，那么接下来所有目标为 18.0/16 的分组将首先发向 C。利用这种方法，所有 AS 中的路由器可以为所有通过其他 AS 中边界路由器可达的前缀建立完整的路由表。

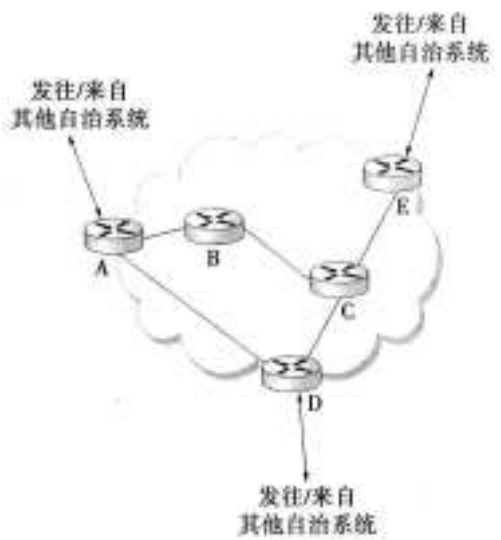


图 4-9 域内和域间路由举例。所有路由器运行 iBGP 和 IRP，边界路由器（A、D、E）运行 eBGP 到达其他 AS

前缀	BGP下一跳
18.0/16	E
12.5.5/24	A
128.34/16	D
128.69./16	A

AS的BGP表

路由器	IGP路径
A	A
C	C
D	C
E	C

路由器B的IGP表

前缀	IGP路径
18.0/16	C
12.5.5/24	A
128.34/16	C
128.69./16	A

路由器的合成表

图 4-10 路由器 B 的 BGP 路由表、IGP 路由表和合成表

4.1.3 IP 版本 6（IPv6）

从很多方面来看，建立新版 IP 的动机都很简单：解决 IP 地址耗尽的问题。划分子网和 CIDR 有助于抑制因特网地址空间的消耗，也有助于控制因特网路由器中所需的路由表信息的增长。然而，这些技术终究有一天无法再满足需要。特别是，实际上不可能达到 100%的地址利用率，因此在第 40 亿台主机连到因特网之前，就可能已经用完了地址空

间。即使我们能够使用所有 40 亿个地址，也不难想象编号耗尽的情况，现在 IP 地址不仅分配给典型的计算机，还分配给手机、电视和其他家庭应用设备。所有这些可能性都说明最终需要一个比 32 位提供的更大的地址空间。

1. 历史观点

IETF 从 1991 年开始就看到了 IP 地址空间扩展的问题，并提出了几种替代方法。由于 IP 地址携带在每个 IP 分组的首部中，因此地址尺寸的增长使得 IP 首部必须改变。这意味着需要新版 IP，并随之需要用于因特网中每台主机和路由器的新软件。显然，这并不是一件小事，而是需要深思熟虑的大改变。

定义新版本 IP 的成果称为下一代 IP 或 IPng。随着工作的进展，一个正式的版本号被指定，因此 IPng 现称为 IPv6（IP 版本 6）。注意，本章到目前为止讨论的 IP 版本是第 4 版（IPv4）。编号不连续的原因是版本号 5 用于几年前的一个实验协议。

新版本 IP 的显著变化引起了滚雪球现象。网络设计者一般认为，如果所设计的网络规模有可能改变，那么最好尽可能同时调整 IP 中的其他内容。因此，IETF 做了一项有关希望在新版 IP 中加入特性的问卷调查。除了提供可扩展的路由和编址的需求外，还希望 IPng 有如下特性：

- 支持实时服务。
- 安全性支持。
- 自动配置（即主机自动地配置自己的 IP 地址和域名等信息的能力）。
- 增强路由功能，包括支持移动主机。

有趣的是，在设计 IPv6 时，IPv4 不具备上述特性中的许多特性，但近年来 IPv4 对这些特性的支持已取得了进展，而且两种协议通常使用类似的技术。可以说，将 IPv6 看成一张白纸的自由促进了 IP 新功能的设计，并且这些功能继而被改装成 IPv4 的功能。

除了以上列出的特性外，IPng 的另一个无可非议的特性是从当前的 IP 版本（IPv4）到新版本必须有一个过渡计划。因为因特网变得如此之大并且没有中央控制，完全不可能有这样一个“国旗纪念日”，让每个人都在这一天关闭主机和路由器来安装新版的 IP。因此，可能有一个很长的过渡期，有些主机和路由器运行 IPv4，有些同时运行 IPv4 和 IPv6，而另一些只运行 IPv6。

IETF 任命一个名为 IPng 理事会的委员会来收集所有 IPng 需求，并且评价将变成 IPng 协议的所有建议。这个委员会收到很多建议，其中一些建议与另一些建议合并，最终由理事会选出一个建议作为 IPng 的基础。这个建议叫作简单因特网协议扩充（Simple Internet Protocol Plus，SIPP）。SIPP 最初要求将 IP 地址扩大一倍到 64 位。当理事会选择了 SIPP 时，他们规定了几种改变，其中一项是将地址再扩大一倍到 128 位（16 字节）。这次，设置版本号为 6。本节的其余部分将描述 IPv6 的一些主要特性。至撰写本书时，IPv6 的大多数关键规范在 IETF 中正处于建议或草拟标准的阶段。

2. 地址和路由

首先，与 32 位的 IPv4 不同，IPv6 提供 128 位的地址空间。因此，如果地址分配有效性能达到 100%，在 IPv4 中可以最多编址 40 亿个节点，而 IPv6 可编址 3.4×10^{38} 个节点。但是，正如我们所看到的，100%的地址分配效率是不可能的。在其他编址方案的某些分

析中，如法国和美国电话网络以及 IPv4 的编址方案的分析，都可找到一些地址分配效率的经验数据。基于从这项研究得出的最悲观的效率估计，预测 IPv6 地址空间在地球表面的每平方英尺上提供 1 500 多个地址。看起来它似乎可以很好地为我们服务，即使全星上的烤面包机也有 IP 地址。

3. 地址空间分配

受益于 IPv4 中 CIDR 的使用，IPv6 地址同样不分类，但是地址空间仍然基于前导比特的不同划分方式。前导比特说明 IPv6 地址的不同用处而不是说明不同的地址分类。当前 IPv6 的地址前缀分配如表 4-1 所示。

表 4-1 IPv6 的地址前缀分配

前 缀	用 途
00...0 (128 位)	未分配
00...1 (128 位)	回环
1111 1111	多播地址
1111 1110 10	链路局部单播
其他	全局单播

这种地址空间的分配需要一些讨论。首先，IPv4 的三种主要地址类（A、B 和 C）的全部功能包含在“其他”范围内。我们很快会看到，全局单播地址很像无类的 IPv4 地址，只是长了很多。在此我们主要感兴趣的是，这种重要的地址形式占据 IPv6 所有地址空间的 99%。（在撰写本书时，IPv6 单播地址的分配从 001 开头的块开始，剩余的地址空间大约占 87%，保留给未来使用。）

多播地址空间用于多播，因此与 IPv4 中 D 类地址的作用类似。注意，多播地址是很容易区分的，它们开头的一个字节全是 1。我们将在 4.2 节看到如何使用这些地址。

链路局部使用地址的思想是使主机构造一个地址，能够适用于它所连接的网络，而不必关心全局地址唯一性问题。我们下面将看到，这一点对自动配置很有用。类似地，站点局部使用地址预定为允许在一个未连在更大因特网上的站点（如专用社团网）上构造有效地址，并且不需考虑全局地址唯一性问题。

在全局单播地址空间中有一些重要的特殊类型的地址。可以通过将 32 位的 IPv4 地址前面加 0 扩展到 128 位，将一个“兼容 IPv4 的 IPv6 地址”分配给一个节点。一个只能理解 IPv4 的节点可以通过对 IPv4 的 32 位地址加上 2 字节全为 1 的前缀，再在前面加 0 直至扩展到 128 位，分配一个“映射 IPv4 的 IPv6 地址”。这两种特殊的地址类型用于 IPv4 到 IPv6 的转换（关于本主题的讨论见相关主题）。

相关主题

IPv4 到 IPv6 的过渡

从 IPv4 到 IPv6 过渡背后的最重要思想是因特网太大且无法集中管理，所以不可能有指定的“某一天”让每台主机和路由器都从 IPv4 升级到 IPv6。因此，IPv6 需要按照这样的方式逐渐部署：只理解 IPv4 的主机和路由器可以继续运行尽可能长的时间。理想情况下，IPv4 节点应该能和其他的 IPv4 或某些有 IPv6 兼容能力的节点对话。而且，IPv6 主机应该能和其他 IPv6 节点不定期地对话，即使它们之间的某些基础结构只支持 IPv4。已定义了两种主要机制帮助实现这种过渡：双栈操作（dual-stackoperation）和隧道技术（tunneling）。

双栈的思想非常简单：IPv6 节点既运行 IPv6 也运行 IPv4，并且使用 Version（版本）字段来决定哪一个栈应处理到达的分组。在这种情况下，IPv6 地址可以与 IPv4 地址无关，或者可以是本节前面描述的“映射 IPv4 的 IPv6 地址”。

基本隧道技术在4.1节描述过，它将IP分组作为另一个IP分组的有效载荷（payload）进行发送。为了向IPv6过渡，隧道用于在只理解IPv4的网段发送IPv6分组。这意味着IPv6分组被封装在一个IPv4的首部内，首部中有隧道端点的地址，穿过只支持IPv4的网段，然后在端点解去封装。端点可以是一台路由器或主机，无论哪种情况，它必须有支持IPv6的能力以处理解开封装后的IPv6分组。如果端点是一个有映射IPv4的IPv6地址的主机，那么通过从IPv6地址中抽出IPv4地址，并用它形成IPv4首部，就可以自动地使用隧道技术。否则，隧道必须进行人工配置。在这种情况下，封装节点需要知道隧道另一端的IPv4地址，因为它不能从IPv6首部中得到。从IPv6的角度，隧道的另一端看起来像是一个正规的只在一跳跨度之外的IPv6节点，尽管在隧道的两个端点之间可能存在IPv4基础设施的许多跳点。

4. 地址的符号表示

和使用IPv4一样，书写IPv6地址时要用到一些特殊的符号。标准的表示方式是x:x:x:x:x:x:x:x，其中每个“x”都是一个16位的地址段的16进制表示。例如

47CD:1234:4422:AC02:0022:1234:A456:0124

任何IPv6地址均可用这种符号表示。由于IPv6地址中有一些特殊类型，因此在特定情况下需要一些特殊的符号表示。例如，有多个连续0的地址可以表示为删掉所有这些0的紧凑形式。因此

47CD:0000:0000:0000:0000:0000:A456:0124

可以写成

47CD::A456:0124

显然，为了避免二义性，这种缩写形式只可用于一个地址中连续多个0的集合。

由于两类包括嵌入IPv4地址的IPv6地址有自己的特殊符号，使得提取IPv4地址更为容易。例如，一台主机的IPv4地址是128.96.33.81，那么其带IPv4映射的IPv6地址可以写成

::FFFF:128.96.33.81

就是说，后32位按IPv4的符号表示书写，而不是以冒号相隔的一对16进制数。注意最前面的双冒号表示前导0。

5. 全局单播地址

至此，编址最重要的事情是IPv6必须提供普通的传统单播编址。为做到这一点，它必须支持新主机在因特网中的快速增加，并在因特网中的物理网络数增长时，允许按可扩展的方式进行路由。因此，IPv6的核心是单播地址分配计划，即确定那些前缀为001的地址如何分配给服务提供商、自治系统、网络、主机和路由器。

事实上，所提出的IPv6单播地址分配计划非常类似于IPv4中CIDR的地址分配计划。为了明白它如何工作以及如何提供可扩展性，我们先定义一些新的术语。我们可以将一个非中转AS（即桩或多连接AS）看作一个用户（subscriber），并且可将一个中转AS看作一个提供商（provider）。而且，我们可进一步将提供商划分为直接（direct）和非直接（indinect）两种。前者直接与用户相连。后者主要连接其他提供商，并不直接与用户相连，通常叫作主干网（backbone network）。

有了这些定义，我们看到因特网并不只是 AS 的任意互联集合，它有一些内在层次。困难在于，在没有发明基于层次性进行工作的新机制的情况下，如何使用这种层次性，就像 EGP 中发生的那样。例如，当一个用户连接到主干网上，或当一个直接提供商开始连接其他多个提供商时，直接的和非直接的提供商之间的区别就会变得模糊。

和 CIDR 一样，IPv6 地址分配计划的目标是提供路由信息的聚合以减少域内路由器的负担。另外，关键思想是使用地址前缀（即在地址高端的一系列连续的比特）来聚合到大量网络甚至是大量 AS 的可达性信息。做到这一点的主要方法是为直接提供商分配一个地址前缀，然后给它的用户分配一个以此前缀开始而比之更长的前缀。这正是我们在图 3-22 中看到的。这样，一个提供商可以向它的所有用户通知一个前缀。

当然，缺点是当站点想要改变提供商时，需要得到一个新的地址前缀，并重新对站点内的节点编号。这是一项很繁重的任务，足以阻止多数人不断更换提供商。因此，人们正在研究其他编址方案（如地理编址），其中站点的地址是其位置的函数而不取决于它连接的提供商。然而现在基于提供商的编址对于有效地完成路由来说是必需的。

注意，虽然 IPv6 的地址分配本质上等价于引入 CIDR 的 IPv4 地址分配方案，但是 IPv6 有一个显著的优点，即不需要让大量前期已分配的地址适应此方案。

问题是在层次结构中其他层进行层次聚合是否有意义？例如，所有提供商都应该从所连接的主干网的前缀中获取它们的地址前缀吗？如果大多数提供商连接在多个主干网上，这可能就变得无意义了。而且，由于提供商数比站点数少得多，在这层中进行聚合的好处就更小了。

进行聚合的意义在于国界或洲界。各大洲的边界形成因特网拓扑中的自然划分，例如，如果欧洲的地址都有一个共同的前缀，那么就可以完成大量的聚合。因此其他洲的大多数路由器只需一条路由表记录来表示所有带欧洲前缀的网络，欧洲的提供商将会选择以欧洲前缀开始的前缀。使用这种方案，一个 IPv6 地址看起来可能如图 4-11 所示。RegistryID（注册号）可以是一个分配给欧洲地址注册的标识符，其他洲或国家也分配不同的 ID。注意，这种情况下前缀将有不同的长度。例如，客户较少的提供商的前缀会比客户较多的提供商的前缀长（并因此有较少的总体可用地址空间）。

3	m	n	o	p	125-m-n-o-p
010	注册号	提供商号	订户号	子网号	接口号

图 4-11 基于提供商的 IPv6 单播地址

当一个用户连接在多个提供商上时，情况较难处理。这个用户应该为其站点使用哪个前缀？此问题没有完美的解决办法。例如，假设一个用户连接在两个提供商 X 和 Y 上。如果用户从 X 得到前缀，那么 Y 不得不通知一个和它的其他订户无关的前缀，因而无法聚合。如果用户以 X 的前缀为其部分 AS 的编号而以 Y 的前缀为另一部分 AS 的编号，那么当一个提供商的连接切断时，就要冒一半站点不可达的危险。当 X 和 Y 有多个共同用户时，效果较好的一种办法是让它们之间有三种前缀：一种用于只属于 X 的用户，一种用于只属于 Y 的用户，另一种用于既是 X 也是 Y 的用户的站点。

6. 分组格式

尽管 IPv6 在多个方面扩展了 IPv4，但它的首部格式实际上更为简单。这种简单性是由于从协议中删除了不必要的功能。图 4-12 给出了 IPv6 的分组首部。（为了与 IPv4 比较，请对照图 3-16 所示的首部格式。）

与很多首部一样，此首部格式也以 Version（版本）字段开始，因为是 IPv6，所以设为 6。IPv6 和 IPv4 的 Version 字段都在首部的开始位置，使得首部处理软件能够立即决定要寻找哪种首部格式。TaffieClass（通信类别）和 FlowLabel（流标签）字段都与服务质量问题有关，将在 6.5 节中讨论。



图 4-12 IPv6 分组首部

PayloadLen（有效载荷长度）字段给出不包括 IPv6 首部在内的分组的长度，按字节计数。NextHeader（下一个首部）字段明确地代替 IPv4 中的 IP 选项和 Protocol（协议）字段。如果需要选项，那么它们被携带在 IP 首部之后的一个或多个特殊首部中，这由 NextHeader 字段中的值指出。如果没有特殊首部，NextHeader 字段是识别运行在 IP 之上的更高层协议（如 TCP 或 UDP）的多路分解密钥，即它和 IPv4 的 Protocol 字段的作用相同。同时，分段被作为一个选项首部来处理，即 IPv4 中有关于分段的字段不包括在 IPv6 首部中。HopLimit（跳数上限）字段也就是 IPv4 的 TTL（生存期）字段，重新命名是为了反映它的实际使用方式。

最后，首部的大部分被源地址和目的地址所占据，各为 16 字节（128 位）长。这样，IPv6 首部长度总是 40 字节。考虑一下，IPv6 地址的长度是 IPv4 地址长度的 4 倍，而首部的差异尤为明显，IPv4 的首部若没有选项只有 20 字节长。

IPv6 处理选项的方法比 IPv4 有很大改进。在 IPv4 中，如果存在选项，每台路由器都得解析整个选项字段，看选项是否相关。这是因为选项作为〈type，length，value〉（〈类型，长度，值〉）的无序集合隐藏在 IP 首部的末尾。相比之下，在 IPv6 中，如果选项存在的话，IPv6 将它看作是必须以特定顺序出现的扩展首部（extension header）。这就意味着每台路由器能够很快地确定是否有选项与它相关，在多数情况下它们是不相关的。通常，这可以从 NextHeader 字段上判断出来。这样的结果是，IPv6 中选项处理的效率较高，这是路由器性能的一个重要因素。另外，将选项作为扩展首部这种新格式意味着它们可以为任意长度，而在 IPv4 中则被限制为最多 44 字节。下面我们来看如何使用其中一些选项。

每个选项有它自己的扩展首部类型。每个扩展首部的类型由前面首部中的 NextHeader 字段值来标识，并且每个扩展首部也包含一个 NextHeader 字段来标识跟在它后面的首部。最后一个扩展首部后面跟着一个传输层首部（如 TCP），这时 NextHeader 字段的值与 IPv4 首部中 Protocol 字段的值相同。因此，NextHeader 字段有双重职责：它既可以标

识随后的扩展首部类型，又可在最后的扩展首部中作为多路分解键标识运行在 IPv6 之上的更高层协议。

考虑如图 4-13 所示的分片首部示例。此首部提供的功能类似于 3.2.2 节描述的 IPv4 首部中的分片字段，但是它只在需要拆分时出现。假设它是唯一出现的扩展首部，那么 IPv6 首部的 NextHeader 字段值将是 44，这个值被分配用来表示分片首部。分片首部的 NextHeader 字段自身包含一个描述其后跟随首部的值。此外假设不存在其他扩展首部，那么下一个首部可能就是 TCP 首部，这将导致 NextHeader 字段的值为 6，就像 Protocol 字段在 IPv4 中那样。如果分片首部后面还跟着一个首部（比如认证首部），那么分片首部的 NextHeader 字段值将为 51。



图 4-13 IPv6 的分片扩展首部

7. 自动配置

尽管因特网以惊人的速度增长，但阻碍这项技术被更快接受的一个因素是连入因特网通常需要一定的系统管理专业知识。特别是，连在因特网上的每台主机至少都需要配置一些信息，如合法的 IP 地址、其所连接链路的子网掩码以及名字服务器的地址。这样，就不可能将一台新开箱的计算机不预先配置就连到因特网上。因此，IPv6 的目的之一是提供自动配置支持，有时叫作即插即用（plug-and-play）操作。

如我们在 3.2.7 节所看到的，IPv4 也可以进行自动配置，但要看是否存在一个能将地址和其他配置信息分发到 DHCP 客户端的服务器。IPv6 中更长的地址格式有助于提供一种有用的新的自动配置形式，叫作无状态（stateless）自动配置，它不需要服务器。

回忆一下，IPv6 的单播地址是分层的，最低有效部分是接口号。这样，我们可以将自动配置问题划分为两部分：

1）获取主机所连接的链路上具有唯一性的接口号。

2）获取此子网的正确地址前缀。

第一部分比较容易实现，因为一条链路上每台主机必须有唯一的链路层地址。例如，以太网上的所有主机有一个唯一的 48 位以太网地址。它可以通过加上表 4-1 中适当的前缀（1111 1110 10），并在后面补上足够的 0 将其变为一个 128 位的合法链路局部使用地址。对某些设备来说，例如打印机或一个小型无路由器且不与其他任何网络相连的网络中的主机，这个地址可能就足够了。那些需要一个全局合法地址的设备依赖同一条链路上的路由器定期向链路通知适当的前缀。显然，这需要路由器配置有正确的地址前缀，并且此前缀是按照保证在末尾有足够的空间（如 48 位）来添加一个合适的链路层地址的方法选择的。

把 48 位长的链路层地址嵌入 IPv6 地址的能力，是选择如此长的地址长度的原因之一。128 位不仅允许嵌入，而且为我们上面讨论的多层编址留有足够的空间。

相关主题

网络地址转换

尽管 IPv6 的引入是出于对 IP 地址使用的增长将导致地址空间耗尽的顾虑，但另一种技术作为节省 IP 地址空间的一种方法正在流行起来。这种技术是网络地址转换（Network Address Translation，NAT），它的广泛使用将可能在很大程度上延缓部署 IPv6 的需求。人们通常认为 NAT 是“结构不规范的”，但是它也是网络发展中不容忽视的事物。

NAT 的基本思想是因特网上所有可能彼此通信的主机不需要全球唯一的地址。一台主机可以设置一个无需全球唯一的“私有地址”，它只要求在一定的范围内唯一即可，例如在主机所在的公司网络内唯一。A 类网络号 10 通常用于此目的，因为这个网络号曾分配给 APPANET 网，所以不再作为全球唯一的地址使用。只要一台主机仅与公司网络内的其他主机进行通信，局部唯一地址就足够了。如果它希望与公司网络以外的主机进行通信，则要经过 NAT 盒（box），这种设备能将主机使用的私有地址转换为分配给 NAT 的某个全球唯一地址。公司内一小部分主机可能同时需要 NAT 盒的服务，因此 NAT 盒需要一个小的全球唯一地址池，其容量远比公司内每台主机都需要一个全球唯一地址的数目小。

因此，我们可以想象一个 NAT 盒接收来自于公司内部主机的 IP 分组，并将 IP 源地址从私有地址（如 10.0.1.5）转换为全球唯一地址（如 171.69.210.246）。当有来自于远程主机的目的地址为 171.69.210.246 的分组到达时，NAT 盒将此目的地址转换为 10.0.1.5，并把分组转发到主机。

NAT 的主要缺点是违反了 IP 服务模型的一个关键假设——所有节点都有全球唯一地址。事实证明很多应用和协议都依赖这个假设。特别是，很多运行在 IP 上的协议（如 FTP 之类的应用协议）在消息中携带 IP 地址。如果高层协议工作正常，这些地址也需要由 NAT 盒来转换，因此 NAT 盒就变得比简单的 IP 首部翻译器更复杂，同时还要求它能理解越来越多的高层协议，因此又产生了部署新应用的障碍。

更严重的问题是，NAT 使得外部设备很难向 NAT 中的私有设备发起连接。由于 NAT 设备缺少映射机制，因此没有公共地址可供连接请求发送。这种情况使得 IP 语音等应用的部署更为复杂。

几乎可以肯定地说：如果不使用 NAT，网络的情况会更好，但是 NAT 看起来似乎不可能消失。IPv6 的广泛部署可能会改善这一状况，但现在的 NAT 因为其他因素，其流行范围超越了最初的设计。例如，如果整个因特网具有（私有的）IP 地址，且和提供商的地址空间无关，那么就可以很轻易地更换服务提供商。然而，NAT 盒并不是一个针对安全威胁的真正的解决方案，将地址放在 NAT 盒中并不能表明其具有抵抗简单攻击的保护等级。随着未来 IPv6 部署步伐的加快，看看 NAT 如何发展将会是非常有趣的事。

---

8. 高级路由能力

IPv6 的另一个扩展首部是路由首部。如果没有这个首部，IPv6 的路由与在 CIDR 下 IPv4 的路由没有多大区别。路由首部包括一个 IPv6 地址表，描述分组到达目标途中所经

过的节点或拓扑区域。例如，一个拓扑区域可以是一个主干网提供商网络。在基于分组接分组的传送机制中，实现提供商选择的一种方法是指定分组必须经过该网络。这样，一台主机可以让一些分组经过一个便宜的提供商，一些分组经过提供高可靠性的提供商，还有一些分组经过主机认为能提供安全性的提供商。

为了提供指定拓扑实体而不是单个节点的能力，IPv6 定义了一个任播（anycast）地址。一个任播地址被分配给一组接口，发往这个地址的分组将到达其中“最近的”一个接口，哪个最近由路由协议来决定。例如，可以给主干网提供商的所有路由器分配一个任播地址，在路由首部中将使用它。

我们也希望任播地址和路由首部能够用于提供对移动主机的高级路由支持。提供这项支持的详细机制还在制定过程中。

9. 其他特性

正如在本节一开始所提到的，开发 IPv6 的基本动机是支持因特网的持续增长。然而，一旦 IP 首部不得不因地址而变化，就到了我们做更多改变的时候，包括我们刚才描述的自动配置和源定向路由。IPv6 包含几个附加特性，大部分将在本书的其他章节介绍，4.4.2 节讨论移动性，网络安全是第 8 章的主题，为因特网提出的一种新服务模型在 6.5 节描述。值得注意的是对于上述方面，IPv4 和 IPv6 的能力几乎没有什么本质区别，因此 IPv6 的主要驱动力仍然是满足更长地址的需求。

4.2 多播

正如我们在第 2 章中看到的，像以太网这样的多点访问网络用硬件实现多播。然而，许多应用需要在互联网上具有一种广泛的多播能力。比如，当一个电台通过因特网广播时，同样的数据必须被发送到有用户打开收音机并调到这个台的所有主机上。在这个例子里，连接方式是一到多。其他一到多的应用包括传输同一段新闻、实时股票价格以及软件升级给多台主机等。还有很多应用连接模式是多对多，比如多媒体远程会议、在线多用户游戏以及分布式仿真。在这些示例中，一个组的成员从多个发送者那里得到信息，基本上是每个发送者那里。对任一个发送者来说，它们都能收到同样的信息。

在正常的网络连接中，每个分组必须包含地址，并被发送到一个单独的主机，因此不能很好地满足每一个应用程序。如果一个应用程序可以将数据发送到一个组，那么它将发送独立的具有认证数据的分组给组的每个成员。这些冗余流量占用了比必需流量更大的带宽。而且，冗余流量并非均匀分布，而是围绕发送主机，并可能轻易超过发送主机及附近网络和路由器的通信能力。

为了更好地支持多对多和一对多的连接，IP 提供了一种 IP 级模拟多播用于多点访问网络，如第 2 章中提到的以太网。现在我们介绍 IP 多播的概念，同时也需要用一个术语来描述之前讨论的传统一对一 IP 服务，这种服务称为单播（unicast）。

基本的 IP 多播模型是基于多播组（group）的多对多模型，每个组都有自己的 IP 多播地址（multicast address）。组里的主机收到任何的分组拷贝都会发送到组的多播地址。一台主机可以在多个组里，利用下面将要讨论的协议也可以通过局部路由器自由地加入或者离开组。因此，如果我们将单播地址关联于一个节点或者一个界面，那么多播地址就关联于一个抽象的组，组的成员随时间而变化。而且，原始多播服务模型允许任何主机向组

发送多播流量，主机没必要成为一个组的成员，但可能有任意数量的发送者加入到一个已知的组。

利用 IP 多播发送认证分组到组内的每个成员时，主机发送一个分组的单个拷贝给组的多播地址。发送主机不需要知道各个组成员的单播 IP 地址，因为正如我们知道的，这些知识存在于互联网上的路由器中。同样，发送主机不需要发送多个分组拷贝，因为路由器无论何时都会在需要的时候将分组转发给多个链接。相比于使用单播 IP 传送相同的分组给多个接收者，IP 多播更可测，因为它消除了一些需要在同一条链路上发送多次的冗余流量，特别是靠近发送主机的链路。

IP 的原始多对多多播已经增强为可支持一对多的多播形式。在一对多多播模型中，即特定源多播（SSM），接收主机指定一个多播组和特定发送主机。接收主机仅将从特定主机收到的多播地址发送给特定的组。许多因特网多播应用（如无线电广播）适合 SSM 模式。为了与 SSM 对比，IP 原始多对多模型有时被称为任意源多播（ASM）。

主机通过使用特定协议与本地路由器通信，向局部路由器发送加入或退出多播组的信号。在 IPv4 中，该协议是因特网组管理协议（IGMP）；在 IPv6 中，它是多播侦听发现（MLD）。路由器有责任使多播行为相对主机来说是正确的。因为一台主机可能退出多播组失败（如遇到冲突或其他错误时），路由器定期选出局域网以确定哪些组仍然对所关联的主机有兴趣。

4.2.1 多播地址

IP 地址将一个子空间保留给多播地址。在 IPv4 中，这些地址被分配在 D 类地址空间中，在 IPv6 中也有一部分地址空间（见表 4-1）是保留给多播组地址的。一些多播区域中的子空间是留给域内多播的，因此，它们可以由不同的域单独重用。

因此，当我们忽略为所有多播地址共享的前缀时，IPv4 中有 28 位可能的多播地址。这带来了一个问题，即何时可以在局域网里使用硬件多播。就拿以太网来说，当我们忽略共享前缀时，以太网多播地址只有 23 位。换句话说，要利用以太网多播，必须将 28 位的 IP 多播地址映射到 23 位以太网多播地址。实施的时候采用了低 23 位的 IP 多播地址作为其以太网多播地址，而忽略了高 5 位。因此，32 位（2^5）IP 地址就映射到了以太网地址。

当以太网的一台主机上加入一个 IP 多播组时，它需要配置以太网接口以接收由相应的以太网多播地址发来的分组。不幸的是，如果主机被路由到那个以太网，将导致接收主机不仅接收了其想得到的多播流量，而且也会收到映射到相同的以太网地址的其他 31 个 IP 多播组的流量。因此，接收主机必须检查 IP 报头的所有多播分组，以确定该分组是否真正属于那个多播组。总之，不匹配的多播地址大小意味着即便主机对流量要到达的多播组没有兴趣，多播流量也会对该主机产生负担。幸运的是，在某些交换网（如交换式以太网），利用路由器对不需要的分组进行识别并抛弃的策略，这个问题就可以得到缓解。

一个令人困惑的问题是如何使发送方和接收方学习将哪些多播地址放在首位。这通常需要一些超出现有网络约束的手段，有一些相当复杂的工具可在因特网上通知组地址。其中一个例子是 sdr，将在 9.2.1 节讨论。

4.2.2 多播路由（DVMRP、PIM、MSDP）

对于任何IP地址而言，路由器的单播转发表显示了哪些连接可以用来转发单播分组。为了支持多播，路由器必须有额外的多播转发表，基于多播地址来显示哪些（一般都多于一个）连接可以用来转发多播分组（如果要通过多个连接来转发的话，路由器将复制这些分组）。因此，如果单播转发表指明了一套路径，那么多播转发表就指明了一套树结构：*多播分发树*（multicast distribution tree）。而且，为支持特定源多播（事实上的结果是，对于任意源多播的某些类型），多播转发表必须根据多播地址和（单播）源IP地址的组合，来确定哪些链接可用，并且重新定义一个树集合。

多播路由是一个多播分发树的决策过程，更具体地说，是一个多播转发表的建立过程。就像单播路由，一个组播路由协议是不够的。它还必须在网络发展的时候控制合理的规模，而且必须适应不同路由域的自治。

1. DVMRP

我们在3.3.2节讨论过的用于单播的距离向量路由可以扩展为支持多播。相应的协议叫作*距离向量多播路由协议*（DVMRP）。DVMRP是第一个有望广泛使用的多播路由协议。

回忆一下，在距离向量算法中，每台路由器保存着一张（Destination，Cost，NextHop）（（目的地，开销，下一跳））的表，并与跟它直连的相邻节点交换成对的（Destination，Cost）（（目的地，开销））表。扩展这个算法以支持多播分两个阶段处理。首先，我们需要设计一种广播机制，允许把分组转发到互联网的所有网络上。其次，我们需要完善这一机制，删除那些没有主机属于多播组的网络。因此，DVMRP可称为*洪泛剪枝*（flood-and-prune）协议的多播路由协议。

给出一个单播路由表，每台路由器都知道到达给定Destination的当前最短路径要经过的NextHop。因此，无论何时一台路由器接收到源S的一个多播分组，当且仅当分组通过在到S的最短路径上的链路到达（即分组来自于路由表中与S相关的NextHop）时，路由器才将分组在所有输出链路上转发（除了分组到来的这条链路）。此策略有效地将分组由S向外扩散，但是并不循环回到S。

此方法有两个主要缺点。首先，虽然的确是以扩散方式发往网络，但无法避开那些没有多播组成员的LAN。我们将在下面解决这个问题。第二个缺点是连到一个LAN上的每台路由器都会将一个给定的分组在这个LAN上转发。这是因为采用向除了分组的来路以外的所有链路进行扩散的转发策略，而没有考虑这些链路是否属于以源为根的最短路径树。

第二个缺点的解决办法是删除那些由连接在给定LAN上的多台路由器产生的重复广播分组。方法之一是为与源相关的每条链路指定一台路由器作为父（parent）路由器，只有父路由器可以在LAN中转发从源而来的多播分组。选择到源S路径最短的路由器作为父路由器，若两台路由器到源等距，则选择有较小地址的路由器。一个给定的路由器能够根据与相邻节点交换的距离向量消息知道它是否是某个LAN中的父路由器（再次同每个可能的源相关）。

注意，这就要求每台路由器对它的每个源来说，都要为每条相关的链路保留一位，指

出它是否是那个源/链路对的父路由器。记住在互联网的背景之下，源是一个网络而不是一台主机，因为一台互联网路由器只对网络间转发分组感兴趣。由此产生的机制有时叫作*逆向路径广播*（Reverse-Path Broadcast，RPB）或者*逆向路径转发*（Reverse Path Forwarding，RPF）。路径是反向的，因为我们在考虑什么时候发出转发指令的时候就已经认识到了到源的最短路径，就像在单播路由中我们用最短路径来决定目的地一样。

RPB机制只实现了最短路径广播。现在，我们想要剪枝接收目的地址为组G的各个分组的网络，而把那些不含有组G成员主机的网络排除在外。这个过程可以分两步实现。首先，我们需要识别没有组成员的叶子（leaf）网络。确定一个网络是否是叶子网络十分容易，如果RPB中描述的父路由器在网络中是唯一的路由器，那么此网络就是叶子网络。确定组的成员是否在网络中，要看组G的成员主机是否定期经过这个网络发出通知，就像我们在链路状态多播中所描述的那样。然后，路由器用此信息来决定是否将地址为G的多播分组在这个LAN上转发。

第二步是将“这里没有组G成员”的信息沿最短路径树上传。要实现这样的功能，路由器需要在向相邻节点发送的（Destination，Cost）对中增加一些内容，说明本叶子网络希望接收哪些组的多播分组。接着，这个信息从一台路由器传播到另一台路由器，这样，对于一台给定的路由器来说，它知道在每条链路上应该转发哪些组的多播分组。

注意在路由更新中包含这些信息的代价相当高。因此，在实际情况中，这些信息只在某个源开始往那个组发送分组时才被交换。换句话说，该策略将使用RPB，即在基本距离向量算法中增加了少许开销，直到某个多播地址被激活。这时，不希望接收目的地址为那个组的分组的路由器进行声明，然后将信息传向其他路由器。

2. PIM-SM

协议无关多播（Protocol Independent Multicast，PIM）是为解决当前多播协议的可扩展性问题而开发的。特别是人们认识到在一小部分路由器希望接收某个特定组的通信量的环境下，当前协议的可扩展性并不是很好。例如，如果大多数路由器从一开始就不希望接收某通信量，那么将该通信量广播到所有路由器，直到它们被明确要求从分发中删除，就并不是一个好的设计选择。这种情况很常见，因此PIM将问题分为*稀疏模式*（sparse mode）和*稠密模式*（dense mode），其中稀疏和稠密是指路由器所期待的多播的比例。PIM稠密模式（PIM-DM）采用了洪泛剪枝算法，就像DVMRP一样，并且还要忍受同样的扩展问题。PIM稀疏模式（PIM-SM）已经成为了域多播路由协议，这里将重点讨论。顺便说一下，和DVMRP等早期协议不同，PIM“协议无关”指的是PIM不依赖于任何特定的单播路由。就像我们下面可看到的，它可以用于任何单播路由。

在PIM-SM中，路由器使用称为Join（加入）的PIM协议消息加入多播分布树。注意这与DVMRP的先创建广播树再剪枝无关路由的方式不同。由此引起的问题是这些Join消息发向何处，毕竟任何路由器（任何数量的路由器）都可以发送消息到多播组。为了解决这一问题，PIM为每个组指定一个称为*汇集点*（rendezvous point，RP）的路由器。通常，一个域中的很多路由器都被配置为候选RP，PIM-SM定义了一系列过程，通过这些过程，域中所有路由器能够一致同意某一台路由器作为一个特定组的RP。这些过程很复杂，因为它们必须处理各种各样的情况，如一台候选RP出现故障以及一个域由于很多链路或节点的故障而划分为两个独立的网络等。后面的讨论中假设域中所有路由器都知道某

个给定组的 RP 的单播地址。

多播转发树是路由器向 RP 发送 Join 消息的结果。PIM-SM 允许构造两种类型的树：共享（shared）树，所有发送方都可以使用；和特定源（source-specified）树，只允许一个特定的发送主机使用。操作的标准模式首先创建共享树，然后，如果有足够的通信量保证还可建造一个或多个特定源树。因为建树时沿树设置路由器的状态，所以默认一个组只有一棵树而不是一个组中每个发送方有一棵树是很重要的。

当一台路由器向组 G 的 RP 发送一条 Join 消息时，它使用标准 IP 单播方式传送。如图 4-14a 所示，路由器 R4 发送一条 Join 到某个组的汇集点。初始 Join 消息是“通配的”，即它适用于所有发送方。显然，一条 Join 消息必须经一台路由器序列到达 RP（如 R2）。沿途的每台路由器都看到了 Join，并在转发表中创建一条共享树的记录，称为（*，G）记录（“*”表示“所有发送方”）。为了创建转发表记录，它查看 Join 到达的接口，并把这个接口标记为用于转发这个组的数据分组的接口。然后它决定使用哪个接口将 Join 转发到 RP。这个接口将成为发往本组的输入分组的唯一可接收的接口。然后它向 RP 转发 Join。最后，消息到达 RP，完成树分支的建立。这样建立的共享树如图 4-14a 中从 RP 到 R4 的粗实线所示。

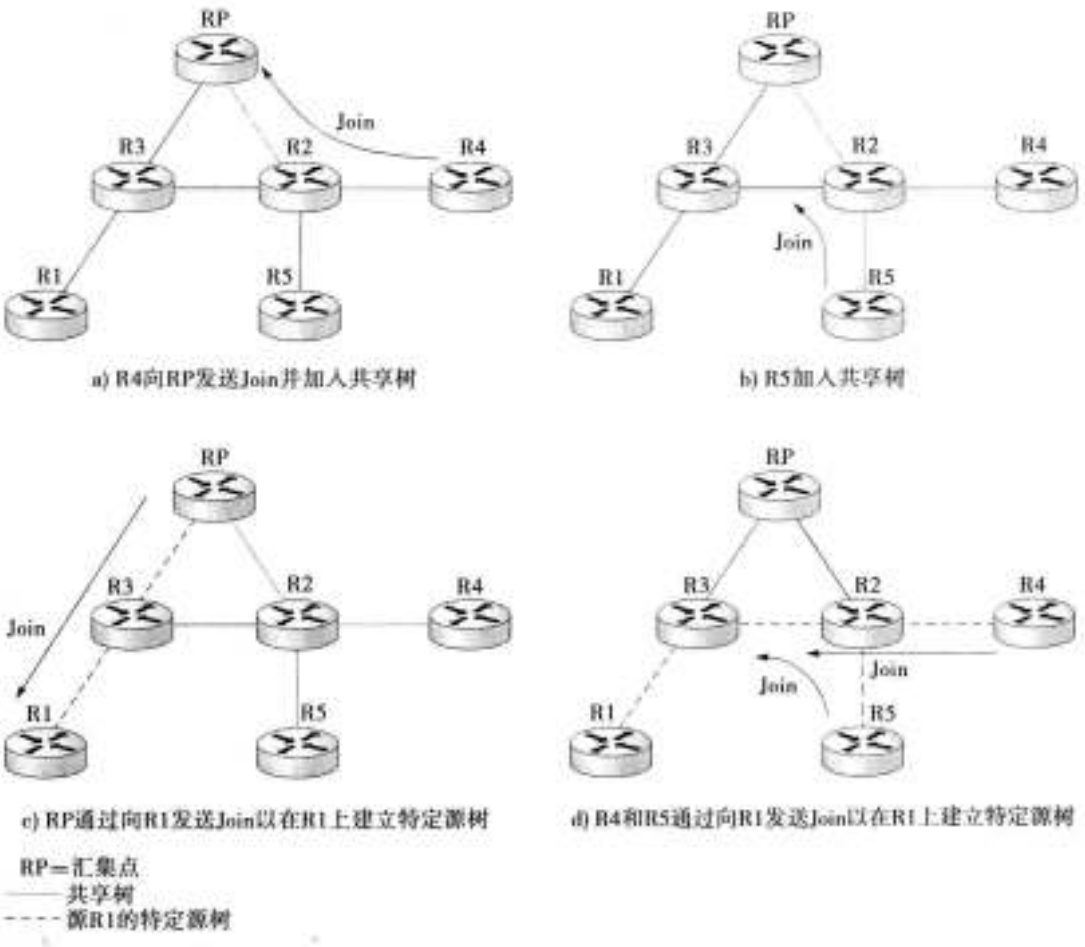


图 4-14 PIM 操作

当有更多的路由器向 RP 发出 Join 时，将导致新的分支添加到树上，如图 4-14b 所

示。注意在这种情况下，Join 只需要传送到 R2，R2 在为这个组创建的转发表记录中简单地增加一个新的输出接口，即可将新分支添加到树上。R2 不需要转发 Join 到 RP，还需注意此过程的最终结果是创建了一棵以 RP 为根的树。

现在，假设一台主机希望发送一个消息到这个组。为此，它构造一个目标为适当的多播组地址的分组，并将其发往一个称为指派路由器（Designated Router，DR）的局域网路由器。假设 DR 是图 4-14 中的 R1。这时 R1 和 RP 之间没有这一多播组的状态，因此，R1 使用隧道（tunnel）将多播分组传送到 RP，而不是简单地转发它。即 R1 将多播分组封装在一个 PIM Register（注册）消息中，发向 RP 的单播 IP 地址。就像 3.2.9 节描述的隧道端点一样，RP 接收指向它的分组，并通过注册信息查看它的有效载荷，找到其中地址为该组的多播地址的 IP 分组。当然，RP 知道如何处理这样一个分组，它会将此分组发往以 RP 为根的共享树。在图 4-14 的例子中，这意味着 RP 将分组发往 R2，R2 能将分组转发到 R4 和 R5。从 R1 到 R4 和 R5 的一个分组的完整传送过程如图 4-15 所示。我们可以看到，从 R1 到 RP 经隧道传送的分组带有包含 RP 单播地址的附加 IP 首部，然后，目的地址为 G 的多播分组沿共享树到达 R4 和 R5。

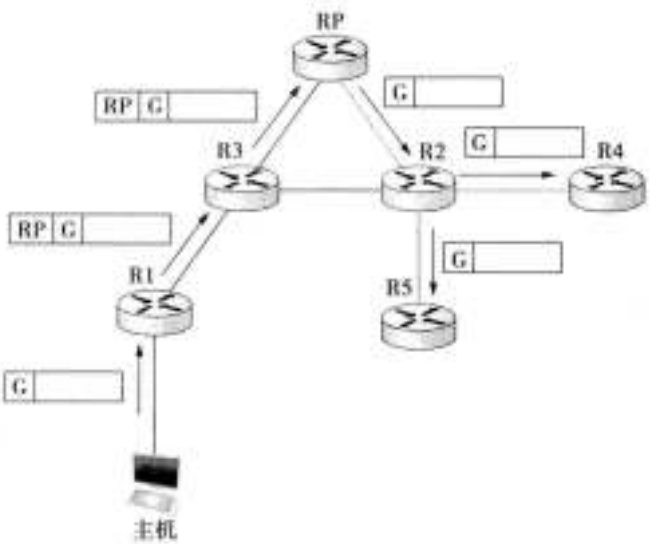


图 4-15 分组沿共享树的传送过程。R1 将分组经隧道传到 RP，RP 再沿共享树将分组转发到 R4 和 R5

现在，我们可能想宣布成功，因为所有主机都可按这种方式向接收方进行发送。然而，在分组发往 RP 时的封装和解封装过程中，存在一些带宽无效和处理开销，因此，RP 可以选择将有关组的信息告之相关路由器以避免使用隧道。它将向发送主机发送一个 Join 消息（见图 4-14c）。当此 Join 向主机传送时，将使沿途的路由器（R3）得知有关组的信息，因此 DR 就能将分组作为原始的（native）（即非隧道的）多播分组向组发送。

本阶段需要注意的一个重要细节是由 RP 发给发送主机的 Join 消息仅限于该发送主机接收，而之前由 R4 和 R5 发送的 Join 消息则适用于所有发送方。这样，新 Join 的结果是创建确定的源与 RP 之间的路由器的特定发送方（sender-specific）状态。这种状态称为（S，G）状态，因为它用于一个发送方到一个组的情况；与之相对照，接收方与 RP 之间的（*，G）状态用于所有发送方。因此，在图 4-14c 中，我们看到从 R1 到 RP 的特定源

路由（以虚线表示）和对所有发送方均有效的从 RP 到接收方的树（以粗线表示）。

下一步可能的优化是以特定源树代替整个共享树。因为从发送方经 RP 到接收方的路径也许比可能的最短路径长得多，也可能因为从某个发送方观察到的高数据率而触发这种优化。在这种情况下，在树的下游的路由器，如例中的 R4，向源发送特定源的 Join 消息。当此消息按最短路径传向源时，沿途路由器创建树的（S，G）状态，结果是得到以源为根而不是以 RP 为根的树。假设 R4 和 R5 都换成特定源树，我们最终将得到如图 4-14d 所示的树。注意这棵树不再包括 RP。我们从图中删掉了共享树以便简化该图，但是实际上，所有连有某组接收方的路由器必须留在共享树中，以防止新的发送方出现。

现在我们明白了为什么 PIM 是协议无关的，因为它建造和维护树的所有机制都得益于单播路由而不依赖于域中使用的任何特殊的单播路由协议。树的构成完全由 Join 消息所走的路径决定，而这个路径由单播路由的最短路径决定。因此，准确地说，与本节描述的从链路状态或距离向量路由中导出的其他多播路由协议相比较，PIM 是“与单播路由协议无关的”。注意，PIM 与 IP 协议密切相关，即从网络层协议看，它并不是协议无关的。

PIM-SM 的设计再一次显示出建造可扩展性网络的挑战，以及可扩展性有时是如何与某种优化性相对立的。共享树当然比特定源树具有更大的可扩展性，因为它将路由器的全部状态化简为组数的规模而不是成倍于组数的发送方数的规模。然而，特定源树很可能是达到更高效的路由所必需的。

3. 域间多播（MSDP）

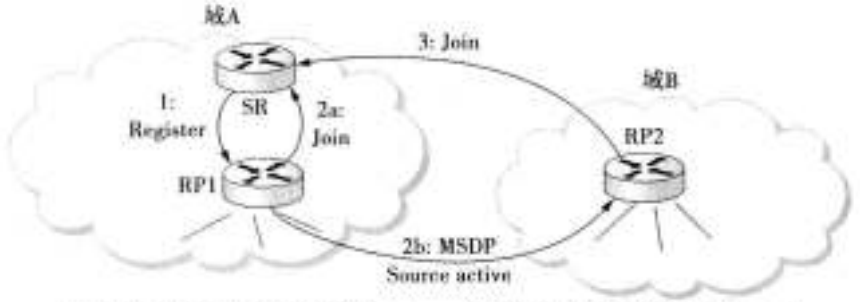
在域间多播时，PIM-SM 有很多大的缺陷。特别是一个组中单个 RP 的存在违反了域本身是匿名的原则。对于一个给定的多播组，所有参与的域都依赖于 RP 所在的域。更为甚者，如果一个参与的多播组中发送方和接收方共享一个单独的域，那么无论域是否有多播组的 RP，多播流量都会从发送方到接收方。因此，PIM-SM 协议不是一个域间协议，而是一个域内协议。

为了使 PIM-SM 可以用于域间多播，人们设计了多播源发现协议（Multicast Source Discovery Protocol，MSDP）。MSDP 用于连接不同的域，其中的每个域都运行 PIM-SM 并拥有自己的 RP。MSDP 可将不同域的 RP 连接起来。每个 RP 在其他域中都有一个或多个 MSDP 对等 RP。每对 MSDP 对等通过 TCP 连接（见 5.2 节）穿越运行 MSDP 协议的区域。同时，给定的多播组中所有 MSDP 对等形成一个用于广播网络的松散网格。MSDP 消息通过该对等 RP 网格利用反路径广播算法进行广播。该算法我们在介绍 DVMRP 内容时讨论过。

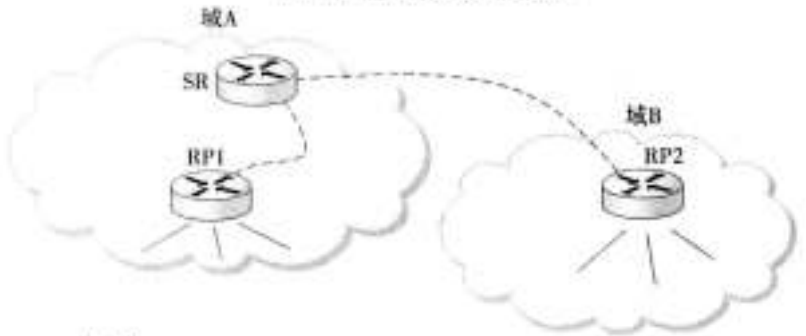
MSDP 通过 RP 网格广播了什么信息呢？肯定不是组成员信息。当一台主机加入一个组时，大部分的信息都流向自己域内的 RP。MSDP 广播的是源——多播发送方——信息。每个 RP 都知道自己域中的源，因为每当一个新的源加入都会接收到一个 Register（注册）消息。每个 RP 定期地使用 MSDP 来广播 Source Active（源活动）消息给对等方，给出源的 IP 地址、多播组地址和原始 RP 的 IP 地址。

如图 4-16a 所示，如果一个 MSDP 对等 RP 接收到多播组有活动的接收方的广播，那么为了它自己的利益，它将发送一个特定源的 Join 消息给源主机。如图 4-16b 所示，Join 信息为这个 PR 建立一个特定源树。结果是每个 MSDP 网络成员并拥有特定多播组的活动接收方的 RP，都加入到新源的特定源树中。当一个 RP 从源中接收多播时，RP 就利用其

共享树在域中给接收方转发多播。



a）源SR向域RP（即RP1）发送Register，然后RP1向SR发送特定源Join，并向在域B的MSDP对等方（即RP2）发送MSDP Source Active，然后RP2向SR发送特定源Join



b）至此，RP1和RP2都加入了源SR的特定源树

图 4-16 MSDP 操作

4. 特定源多播（PIM-SSM）

就像早期的多播协议一样，最初的 PIM 服务模型是一个多对多的模型。接收方加入一个组，而且任何主机都可以向这个组发送消息。然而，在 20 世纪 90 年代末，人们重新认识到 PIM 中可以加入一对多模型。毕竟，很多多播应用只有一个合法发送方，就像一个正在因特网上发送的会议的发言者一样。我们已经看到 PIM-SM 在使用原始共享树后，可以创建出优化的特定源最短路径树。在原始的 PIM 设计中，这种优化对于主机是不可见的——只有路由器加入特定源树中。然而，一旦人们认识到一对多服务模型的需求，PIM-SM 的特定源路由能力就被确定为主机可用。事实上，这需要改变 IGMP 和类似于 IPv6 的 MLD（多播侦听发现），而不是改变 PIM 本身。这种新近的能力称为 PIM-SSM（PIM 特定源多播）。

PIM-SSM 引进了一个新的概念——信道（channel）。信道是一个源地址 S 和组地址 G 的合成。组地址 G 就像一个普通的 IP 多播组，而且 IPv4 和 IPv6 已经分配了 SSM 的多播地址空间的子范围。为使用 PIM-SSM，主机要指出在发送给本地路由器的 IGMP 成员报告消息中的组和源。这台路由器随后发送一个 PIM-SM 特定源 Join 消息给源，因此在特定源树上给自己增加一个分支，就像之前描述的“普通”PIM-SM 一样，但绕过了整个共享树阶段。既然树中结果是针对特定源的，那么只有指定的源在树中可以发送分组。

PIM-SSM 的引入带来了一些明显的好处，特别是因为对一对多播有着更高的要求：

- 多播更直接地传向接收方。
- 信道地址更有效地由多播组地址加上源地址组成。因此，给定一个特定多播组地址范围有利于 SSM 的使用，多个域可以独立而没有冲突地使用同一个多播组地址，就像它们在自己域中的源上使用一样。
- 既然只有特定源可以向 SSM 组发送消息，所以几乎不存在被恶意主机使用伪造的多播流量来淹没主机和接收方从而被攻击的风险。
- PIM-SSM 可以像用于域内一样用于域间，而不会依靠任何其他类似于 MSDP 的协议。

所以，SSM 对于多播服务模型来说是一个有用的补充。

5. 双向树（BIDIR-PIM）

我们利用另一种针对 PIM 进行增强的称为双向 PIM（Bidirectional PIM）的方法来完善对于多播的讨论。BIDIR-PIM 是 PIM-SM 的变种，可很好地适用于域内的多对多多播，尤其是当组内的发送方同时也是接收方时（比如在多方视频会议的情境下）。在 PIM-SM 中，潜在的接收方通过发送 IGMP 成员报告消息（一定不是特定源消息）加入组，共享 RP 的树从而转发多播分组给接收者。然而，和 PIM-SM 不同，共享树有逆向承的分支。这对于 PIM-SM 的单向树没有任何影响，但 BIDIR-PIM 的树是双向的——从下行流分支中收到多播分组的路由器可以将分组向下或向上沿树进行转发。传输一个分组到特定接收方的路由在向下分支到达接收方之前，必须经过同样远的向上分支。作为例子，请看图 4-17b 中的 R1 到 R2 之间的多播路由。R4 转发一个下行多播分组到 R2 的同时，转发一个相同分组的拷贝到上行的 R5。

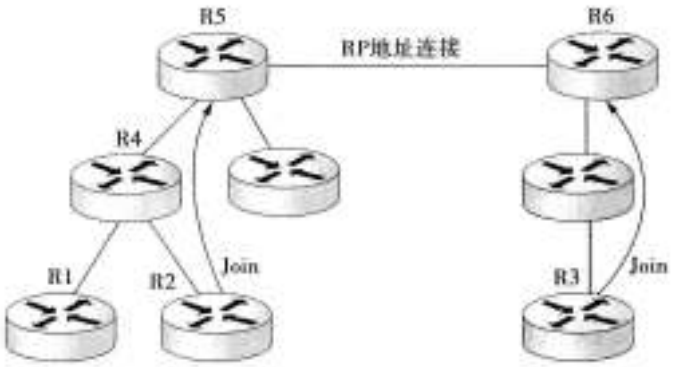
BIDIR-PIM 令人吃惊的表现之一是它不需要 RP。它所需要的仅仅是一个可用路由地址，我们称之为 RP 地址，尽管它完全没必要是一个 RP 的地址。这意味着什么？来自接收方的 Join 被转发到 RP 地址直到到达其台路由器，这台路由器具有链接的接口，而这个链接包含 Join 消息终止位置的 RP 地址。图 4-17a 显示了一个从 R2 到 R5 的 Join 消息，和另一个从 R3 到 R6 的 Join 消息。多播分组的上行流同样流向 RP 地址，直到到达一台具有 RP 终止地址所在链路接口的路由器。之后是上行转发的最后一步，路由器在此链路上转发多播分组，保证此链接上的所有其他路由器可以接收到该分组。图 4-17b 显示了从 R1 出发的多播流量。

BIDIR-PIM 至今无法用于域间。另一方面，与域中用于多对多多播的 PIM-SM 相比，BIDIR-PIM 还是有很多优势的。

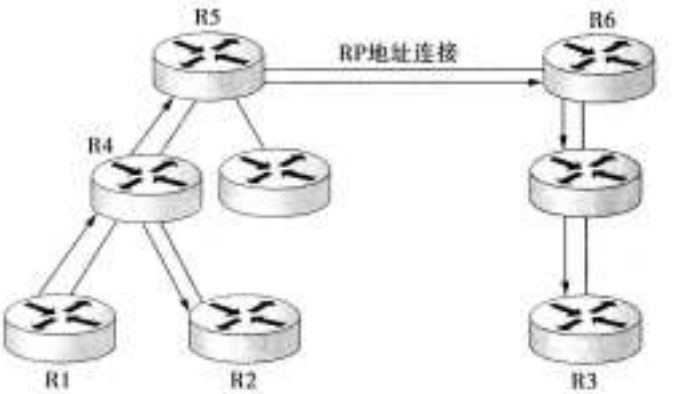
- 没有源注册过程，因为路由器已经知道如何向 RP 地址路由一个多播分组。
- 该路由比使用 PIM-SM 共享树的路由更直接地到达，因为该路由只按需要沿树上行而不是遍历整个 RP 路径。
- 双向树使用比 PIM-SM 特定源树更少的状态，因为这里没有任何特定源状态（换句话说，路由将长于特定源树）。
- RP 不会成为瓶颈，而且确实不需要真正的 RP。

结论产生于这样一个事实，多播中很多不同的方法是与 PIM 有关的，而在多播空间里寻找优化方案是个非常困难的问题。在你为这个任务选择最好的多播模式之前，需要决定用哪些标准进行优化（使用带宽、路由器状态、路径长度等）以及支持哪种应用（一对

多、多对多等)。



a) R2和R3向RP地址发送Join，并在到达一个RP地址链路上的路由器时终止



b) R1发出的多播分组转发上行流到RP地址链路，当与某个成员分支相交时转发下行流

图 4-17 BIDIR-PIM 操作

它们现在在哪?

多播协议的命运

自从 1991 年 Steve Deering 的博士论文 "Multicast Routing in a Datagram Network"（数据分组网络中的多播路由）发表之后，许多 IP 多播协议被抛弃。在很多情况下，这些协议都仍然有很多可惜鉴的地方。最成功的早期多播协议是 DVMRP，我们已经在本节开始的时候讨论过了。多播开放最短路径优先（MOSPF）协议是基于 OSPF 单播路由协议的。PIM 稠密模式（PIM-DM）与 DVMRP 类似，同样使用了洪泛剪枝方法，同时在独立单播路由协议中的应用又与 PIM-SM 类似。这些协议更适合稠密域（比如，一个有高转发率的路由器会对多播有兴趣）。在扩展性的挑战充分显现之前，这些早期多播协议都显示出了一定的相关性。即使它们仍然在一个多播组域内有意义并期望得到更多的关注，但它们现在已经很少使用了，一部分因素是路由器通常必须支持 PIM-SM。

核基树（Core-based tree，CBT）是另外一种多播方法，这种方法是与 PIM 同时期出现的。IETF 一开始无法在这两种方法中进行选择，而且 PIM 和 CBT 都是先进的 "实验性" 协议。然而，PIM 在行业中被人们更广泛地采用了，而且 CBT 的主要技术——共享

树和双向树——分别被完全合并到 PIM-SM 和 BIDIR-PIM 中。

边界网关多播协议（BGMP）使用了双向共享树的概念。然而，在 BGMP 中，树的节点是域，域中的一个是根。换句话讲，BGMP 像 MSDP 一样连在一起支持域间多播。与 MSDP 不同的是，域可以自由地选择域内协议。BGMP 是 IETF 提出的协议，几年前人们还希望 BGMP 取代 MSDP 作为主要的域间路由协议。然而 BGMP 非常复杂，它需要一个给域指派多播地址的协议，以使 BGMP 知道哪个域是给定地址的根。因此，在写出该协议的时候，并没有对 BGMP 加以实现。

4.3 多协议标签交换（MPLS）

我们继续讨论使 IP 性能增强的体系结构，这些结构运用广泛，但在终端用户一侧通常是隐藏的。一种提高方式称为多协议标签交换（Multi-protocol Label Switching，MPLS），它将虚电路的一些特点与数据报的灵活性和健壮性结合在一起。一方面，MPLS 与基于 IP 数据报的体系结构有密切关系，即它依靠 IP 地址和 IP 路由协议来工作。另一方面，支持 MPLS 的路由器也通过检查相对短的、固定长度的标记来转发分组，这些标记有一个局部范围，就像在虚电路网络中一样。也许正是这两种表面对立的技术的结合使 MPLS 在因特网工程界被各方所接受。

在讨论 MPLS 如何工作之前，很自然会问："它有什么好处？"关于 MPLS 有很多说法，但是现在只提及以下三个主要方面：

- 使不具备按正常方式转发 IP 数据报能力的设备支持 IP。
- 按显式路由（预先计算的路由）转发 IP 数据报，而无需匹配普通 IP 路由协议选择的路由。
- 支持特定类型的虚拟专用网服务。

值得注意的是，最初目的之一——提高性能反而没有提到。近些年，为了提高性能，人们对 IP 路由器的转发算法做了很多工作，并考虑了很多决定性能的首部处理之外的复杂因素。

了解 MPLS 如何工作的最佳方法是来看一些实际使用的例子。在以下三节中，我们将用例子分别讲述以上提到的三种 MPLS 的应用。

4.3.1 基于目的地的转发

最早介绍为 IP 分组附加标签这一思想的文章之一是 Chandranmenon 和 Varghese 写的论文，其中描述了一种称为链式索引（threaded index）的思想。与之非常类似的思想现在实现在支持 MPLS 的路由器上。下面的例子说明这种思想是如何工作的。

考虑如图 4-18 的网络。最右侧的两台路由器（R3 和 R4）每一台都连着一个网络，前缀分别为 18.1.1/24 和 18.3.3/24。其他两台路由器（R1 和 R2）有路由表，指出当路由器要转发分组到那两个网络之一时使用哪个发送接口。

当一台路由器能够支持 MPLS 时，它给路由表中的每个前缀都分配一个标签，并将标签和所表示的前缀通知相邻路由器。此通知的分发由标签分发协议（Label Distribution Protocol，LDP）携带。如图 4-19 所示，路由器 R2 给前缀 18.1.1 分配的标签值是 15，给前缀 18.3.3 分配的标签值是 16。这些标签可由分配路由器来选择，可以看作是路由表的

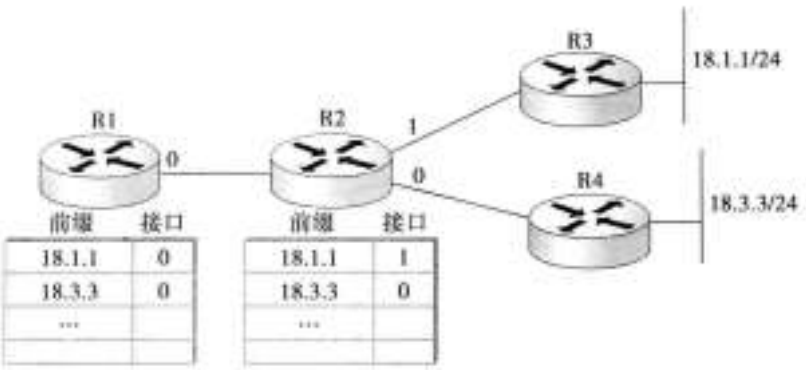


图 4-18 示例网络中的路由表

索引。分配标签之后，R2 将标签绑定通知给相邻的节点，在此例中，可以看到 R2 将标签 15 和前缀 18.1.1 之间的绑定通知给 R1。事实上，这样的一个通知就相当于 R2 在说“请将那些发给我的目标前缀为 18.1.1 的分组全都附着标签 15”。R1 将标签存在一个表中，旁边是其前缀，表示它是发往那个前缀的任何分组的远程或输出标签。

在图 4-19c 中，我们看到路由器 R3 将另一个前缀 18.1.1 的标签通知 R2，R2 将从 R3 处得到的远程标签放入表中的适当位置。

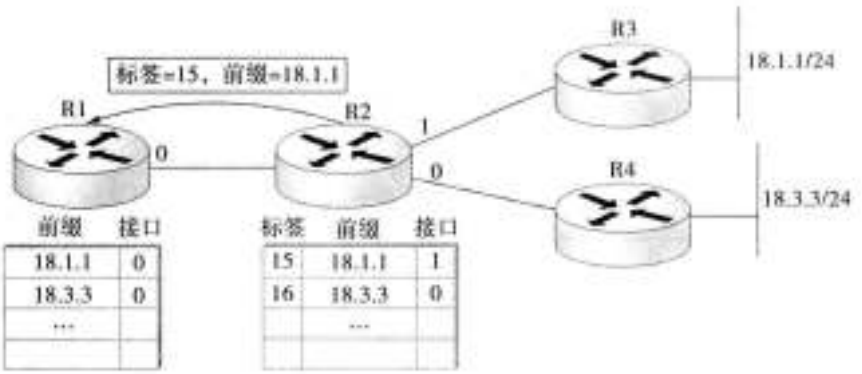
现在，我们可以来看一个分组在这样的网络中转发时的情况。假设一个目的 IP 地址为 18.1.1.5 的分组从左边传送到路由器 R1。在这种情况下，R1 叫作标签边缘路由器（Label Edge Router，LER），LER 对到达的 IP 分组进行完全的 IP 查找，然后用它们的标签作为查找的结果。在这种情况下，R1 发现 18.1.1.5 与转发表中的前缀 18.1.1 匹配，并且这条记录中还包含一个输出接口和一个远程标签值。因此 R1 将远程标签 15 附加到这个分组上，然后发送。

当分组到达 R2 时，R2 只查看分组中的标签，而不查看 IP 地址。R2 中的转发表指示到达的携带标签值 15 的分组应从接口 1 发出，并应携带路由器 R3 通知的标签值 24。因此，R2 重写或是交换标签，并把它转发到 R3。

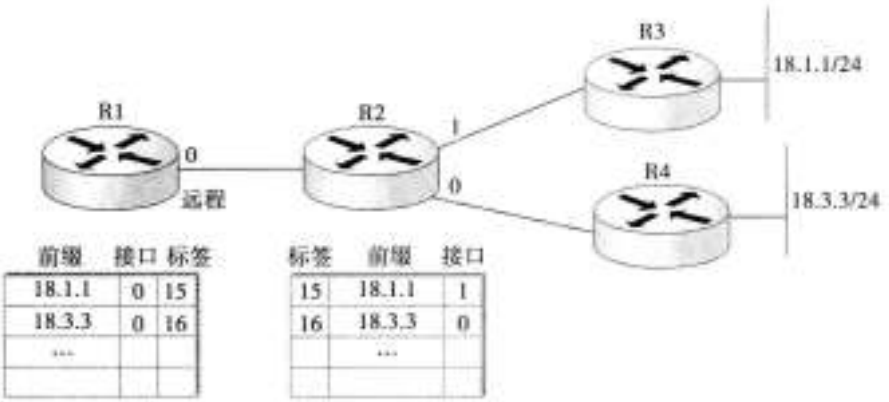
这些应用和标签交换完成了什么工作？来看此例中 R2 转发分组时的情况。其实，它根本不需要检查 IP 地址。R2 只检查输入的标签即可。因此，我们用标签查找代替正常的 IP 目的地址查找。为了理解这种做法的重要性，回忆一下，虽然 IP 地址的长度总是相同的，但是 IP 前缀是变长的，且 IP 目的地址查找算法需要查找最长匹配（longest match），即与将要转发的分组中的 IP 地址高比特部分相匹配的最长前缀。相反，刚才描述的标签转发机制是一种精确匹配（exact match）算法。例如，通过使用标签作为数组的索引，其中数组的每个单元是转发表中的一行，就可以实现一个非常简单的精确匹配算法。

注意，当转发算法从最长匹配变为精确匹配时，路由算法可以是任何一种标准的 IP 路由算法（如 OSPF）。在此环境中，分组经过的路径就是在不使用 MPLS 时分组走的那条路径，即 IP 路由算法选择的路径。改变的只是转发算法。

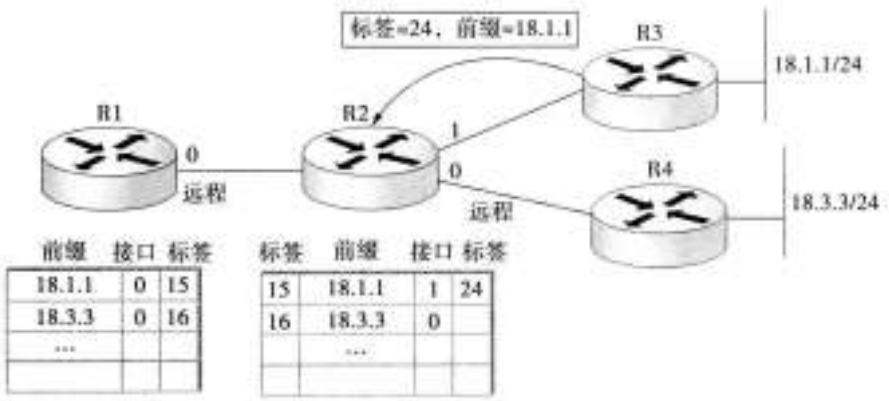
MPLS 重要的一个概念可以用这个例子来解释。每个 MPLS 标签与一个转发等价类（forwarding equivalence class，FEC）相关联——某台路由器接收同一转发处理的分组集合。在这个例子里，每个路由表中的前缀都是一个 FEC。所有满足前缀 18.1.1 的分组——无论低序列字节的 IP 地址是什么——都沿同样长的路线转发。因此，每台路由器



a）R2给R1分配了一些标签并通知绑定



b）R1将收到的标签存储在表中



c）R3通知另一个绑定，而R2将收到的标签存储在表中

图 4-19

都能分配一个标签用来映射到 18.1.1，而且每一个分组包含的 IP 地址中，只要高位和前缀想匹配便可以用这个前缀来转发。

就像我们将在子序列样例中看到的，FEC 是非常强大而灵活的概念。FEC 可以形成几乎所有标准。比如，所有与特殊用户相关的分组都被认为在同一个 FEC 中。

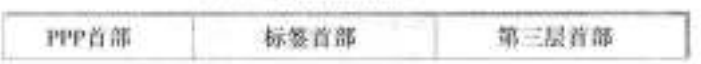
回到眼前的例子，我们发现将转发算法从普通 IP 转发改为标签交换产生了一个重要的结果：以前不知道如何转发 IP 分组的设备可以用来在 MPLS 网络里转发 IP 流量。早期最值得称道的满足此要求的是 ATM 交换机，它们可以不做任何硬件变化就支持 MPLS。

ATM 交换机支持之前所描述的标签交换转发算法。如果将 IP 路由协议（IRP）和发布标签绑定的方法提供给这些交换机，就可以将它们变成标签交换路由器（LSR）——该设备可以运行 IP 控制协议但使用标签交换转发算法。最近，相同的思想被应用于在 3.1.2 节介绍的光交换机中。

在我们考虑将一台 ATM 交换机变成一台 LSR 的好处之前，先来看一些尚未解释清楚的问题。我们说将标签“附加”在分组上，但是，到底附加在哪儿呢？答案要看分组是在何种链路上传送的。在分组上携带标签有两种常用方法，如图 4-20 所示。当 IP 分组作为完整的帧传送时，它们在大多数链路类型上，如以太网和 PPP，标签将作为一个“夹片”插在第二层首部和 IP（或其他第三层）首部之间，如图中较低位置所示。然而，如果一台 ATM 交换机用来完成 MPLS LSR 的功能，那么标签应该放在交换机能使用的地方，即需要放在 ATM 信元首部，确切地说，通常在我们寻找虚拟电路标识（VCI）和虚拟路径标识（VPI）字段的地方。



a) ATM封装分组标签



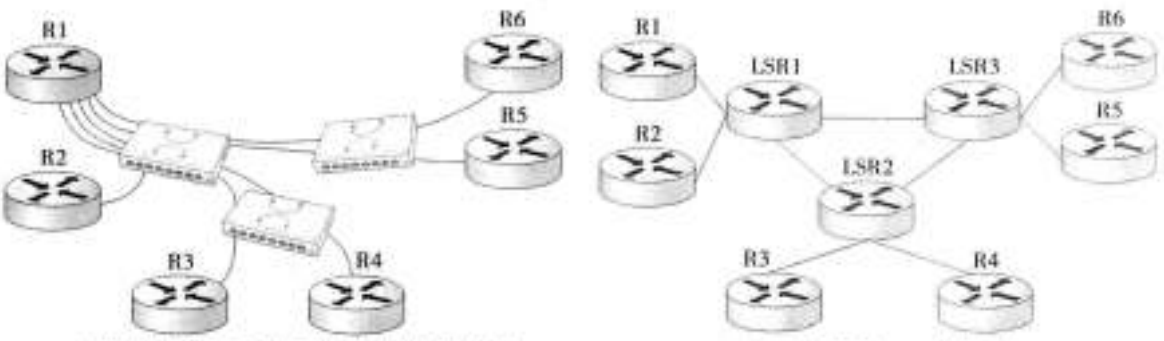
b) 帧封装分组标签

图 4-20

现在通过这个设计方案 ATM 交换机能完成 LSR 的功能，那么我们从中得到了什么？需要注意的一点是，现在我们可以建造一个混合使用传统 IP 路由器、标签边缘路由器和完成 LSR 功能的 ATM 交换机的网络，同时它们都可以使用相同的路由协议。为了理解使用相同协议的好处，来看两种情况。在图 4-21a 中，我们看到一组路由器通过 ATM 网络上的虚电路互联，这种配置叫覆盖（overlay）网络。过去经常建造这种类型的网络，因为商用 ATM 交换机比路由器支持的总吞吐量高。现在，此类网络不那么普遍了，因为路由器的性能已经提高，甚至超过了 ATM 交换机。然而，这种网络仍然存在，因为在主干网中已经安装了很多 ATM 交换机，某种程度上也是因为 ATM 交换机有能力支持一些性能，如电路模拟服务和虚电路服务。

在覆盖网络中，每台路由器都可经虚电路连接其他所有路由器，但是在这里，为了表示得更清晰，我们只给出了从 R1 到其他所有对等路由器的电路。R1 有 5 个路由邻居，并需要跟所有邻居交换路由协议消息——我们说 R1 有 5 个路由邻接点。与之相对照，在图 4-21b 中，ATM 交换机被 LSR 代替，不再有虚电路与路由器连接。因此，R1 只有一个邻接点，即 LSR1。在大型网络中，在交换机上运行 MPLS 将使每台路由器必须维护的邻接点数目大幅减少，并可大幅减少路由器为了将拓扑改变通知到其他每台路由器而必须做的工作量。

在边缘路由器和 LSR 上运行相同路由协议的第二个好处是可使边缘路由器对网络拓扑有全面的了解。这就意味着如果网络中的某条链路或某个节点发生故障，那么边缘路由



a）路由器通过重叠虚电路与其他路由器连接　　b）路由器通过LSR对等连接

图 4-21

器就会有更好的机会选择一条好的新路径，而不像在边缘路由器没有这些知识的情况下，ATM 交换机需要变更受影响 VC 的路由。

注意将 ATM 交换机用 LSR "替换" 的步骤实际上是通过改变运行在交换机上的协议来实现的，但通常不需要改变转发硬件。也就是说，一台 ATM 交换机总是可以通过只升级软件而转换为一台 MPLS LSR。而且，MPLS LSR 在运行 MPLS 控制协议的同时，可以继续支持标准 ATM 功能，这称为 "混合转换器模式"。

最近，在原本不能转发 IP 分组的设备上运行 IP 控制协议的思想已经扩展到了光交换机和 STDM 设备上，如 SONET 多路复用器。这通常称作通用 MPLS（generalized MPLS，GMPLS）。采用 GMPLS 的部分动机是为路由器提供光网络的拓扑知识，正如 ATM 中的情况。更重要的是以前没有控制光设备的标准协议，因此，看起来 MPLS 自然可以承担这个工作。

相关主题

MPLS 位于哪一层？

关于 MPLS 究竟属于 1.3 节给出的分层协议体系结构的哪一层还存在很多争论。由于 MPLS 首部通常位于分组的第三层和第二层首部之间，所以它有时被称为 2.5 层协议。有些人争辩说，由于 IP 分组被封装在 MPLS 首部内，因此 MPLS 必须在 IP "之下"，使它成为第二层协议。而另一些人认为，由于 MPLS 的控制协议的很大一部分与 IP 使用同样的协议（MPLS 使用 IP 路由协议和 IP 编址），因此 MPLS 必须和 IP 在同一层（即第 3 层）。如 1.3 节所述，分层体系结构是有用的工具，但是并非总能准确地描述真实世界，MPLS 就是一个很好的例子，反映出严格的分层观点可能很难与现实相一致。

4.3.2 显式路由

在 3.1.3 节，我们介绍了源路由的概念。IP 有一个源路由选项，但是有几个原因使其并未得到广泛使用，包括它只能说明有限数目的跳数，并且它通常在大多数路由器上的 "快速路径" 之外进行处理。

MPLS 提供了一种方便的方法将类似于源路由的能力添加到 IP 网络中，尽管这种能力通常叫作**显式路由**（explicit routing）而不叫作**源路由**（source routing）。做此区别的一个原因是它通常不是选择此路由的分组的真正源，在更多情况下它是服务提供商网络中的

一台路由器。图 4-22 所示的是如何使用 MPLS 显式路由的例子。这种网络通常叫作鱼（fish）形网络，因为它的形状像鱼（路由器 R1 和 R2 是尾，R7 是头）。

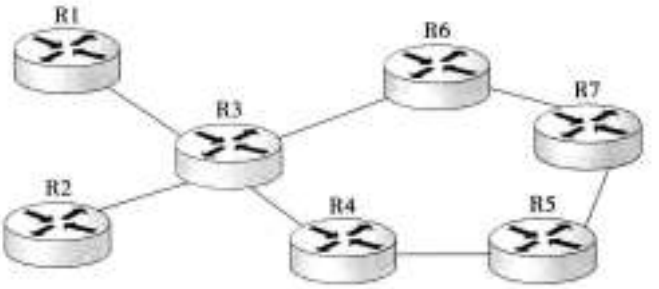


图 4-22 一种需要显式路由的网络

假设在图 4-22 的网络中，操作者已经决定任何从 R1 到 R7 的通信量都应经过路径 R1-R3-R6-R7，而任何从 R2 到 R7 的通信量都经过路径 R2-R3-R4-R5-R7。这样选择的一个原因是充分利用从 R3 到 R7 之间两条不同路径的可用容量。正常的 IP 路由无法轻易做到这一点，因为 R3 做出转发决定时并不看通信量来自何处。

因为 MPLS 使用标签交换来转发分组，所以如果路由器能使用 MPLS，就能比较容易地达到想要的路由。如果 R1 和 R2 在将分组发送到 R3 之前为它们附加了各自的标签，那么 R3 就可以将来自 R1 和 R2 的分组转发到不同的路径。由此产生的问题是，网络中的所有路由器使用何种标签达成共识，并且如何转发有特别标签的分组？显然，我们不能使用上节描述的步骤来分发标签，因为那些步骤建立的标签会使分组按照正常 IP 路由的路径走，而这正是我们希望避免的。所以，需要一种新的机制。完成此任务的协议叫作资源预留协议（RSVP）。我们将在 6.5.2 节讨论有关这个协议的更多内容，这里，只说明它能够沿显式说明的路径（如 R1-R3-R6-R7）发送一个 RSVP 消息，并据此建立沿这条路径的所有标签转发表项。这非常类似于 3.3 节中描述的建立一条虚电路的过程。

显式路由的应用之一是通信量工程（traffic engineering），它是指保证网络中有足够的可用资源满足需求的任务。确切地控制通信量流经哪一条路径是通信量工程的一个重要部分。显式路由也有助于使网络在面对故障时更容易恢复，这一功能称为快速重选路由（fast reroute）。例如，可以预先计算一条从路由器 A 到路由器 B 的明显避开某条特定链路 L 的路径。当链路 L 出错时，路由器 A 就可将所有目标是 B 的通信量经预先计算的那条路径发送。将预先计算的备份路径与对沿路径分组进行显式路由的结合，意味着 A 不需要等待路由协议分组穿过网络或等待网络上的其他各种节点执行路由算法。在特定情况下，这样可以在很大程度上减少分组为绕过出错点而重选路由所用的时间。

有关显式路由需要注意的最后一点是，显式路由不需要由网络操作者来计算，如上例所述，路由器可以用很多算法来自动计算显式路由。其中最常见的是约束最短路径优先（Constrained Shortest Path First，CSPF）算法，类似于 3.3.3 节描述的链路状态算法，但是要考虑一些约束。例如，如果需要找到一条从 R1 到 R7 的路径，能够承担要求的 100Mbps 负载，那么，我们就可以说约束是每条链路必须至少有 100Mbps 的可用容量。CSPF 解决这类问题。有关 CSPF 的更多细节和显式路由的应用在本章末尾的“扩展阅读”

中介绍。

4.3.3 虚拟专用网和隧道

我们在3.2.9节先讨论了虚拟专用网（VPN），并且我们注意到，建造它们的方法之一是使用隧道。事实证明，MPLS可以看作是建造隧道的一种方法，这使它适合于建造各种类型的VPN。

最简单形式的MPLS VPN可以理解为第二层的VPN。在这种类型的VPN中，MPLS作为第二层数据（如以太网的帧或ATM信元）的隧道穿过能用MPLS路由器的网络。回忆一下，在3.2.9节中，使用隧道的一个原因是在网络中提供一些路由器不支持的某种网络服务（如多播）。这里使用同样的逻辑：IP路由器不是ATM交换机，因此在传统的路由器网络中不能提供ATM虚电路服务。然而，如果有经隧道互联的一对路由器，那么就可经隧道发送ATM信元，并仿真一条ATM电路。这种技术在IETF中称为伪线仿真（pseudo wire emulation）。图4-23描述了这种思想。

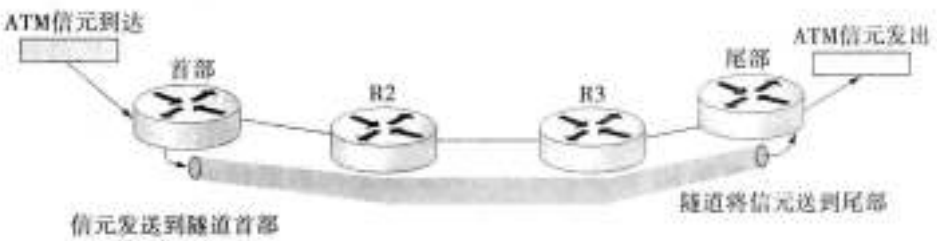


图4-23 一条由隧道仿真的ATM电路

我们已经知道IP隧道是如何建造的：在隧道入口的路由器把将要用隧道传送的数据封装在一个IP首部（隧道首部）内，它表示在隧道远端的路由器地址，然后像发送其他IP分组一样发送数据。接收方路由器收到首部中带有它自己地址的分组后，将隧道首部剥离，找到经隧道发送的数据，然后进行处理。它对数据所做的具体处理依赖于数据是什么。例如，如果它是另一个IP分组，就像普通IP分组那样将它转发出去。然而，只要接收方路由器知道怎么处理非IP分组的话，那么它也可以不是一个IP分组。我们即将讨论如何处理非IP数据的问题。

MPLS隧道与IP隧道没有太大不同，只是隧道首部中包含MPLS首部而不是IP首部。看图4-19中的第一个例子，我们看到路由器R1为每个发往前缀18.1.1的分组附加一个标签（15）。这样的分组将通过路径R1-R2-R3传送，路径中的每台路由器只检查MPLS标签。因此，我们看到R1沿此路径不仅可以发送IP分组，只要是能够封装到MPLS首部的任何数据都可以通过这条路径，因为相关路由器只看MPLS首部而不关心其他。在这一点上，MPLS首部和IP首部一样。[⊖]沿隧道（MPLS或其他）发送非IP通信量的唯一的问题是：当非IP通信量到达隧道端点时，我们该做什么？常用的解决办法是在隧道负载中携带某种多路分解标识符，告诉隧道端点的路由器怎么做。事实证明MPLS标记非常适合做这样的标识符。可以用一个例子把这一点解释清楚。

假设在一个支持MPLS的路由器网络上，我们想要使用隧道将ATM信元从一台路

⊖ 注意，IP的首部长度为20字节，而MPLS只有4字节，这意味着使用MPLS更节省带宽。

由器传送到另一台路由器，如图 4-23 所示。而且，假设我们的目标是仿真一条 ATM 虚电路，就是说，信元到达带有某个 VCI 的输入端口的隧道入口或首部，然后从某个输出端口的隧道的尾部和另一个 VCI 处离开。为做到这一点，可如下配置首部和尾部路由器：

- 首路由器需要配置输入端口、输入 VCI、仿真电路的多路分解标签以及隧道端路由器的地址。
- 尾部路由器需要配置输出端口、输出 VCI 和多路分解标签。

一旦路由器有了这些信息，我们就可以看到 ATM 信元是如何转发的，其步骤如图 4-24 所示。

1）一个 ATM 信元到达带有合适 VCI 值（本例中是 101）的指定输入端口。

2）首部路由器添加用来识别仿真电路的多路分解标签。

3）然后，首部路由器添加第二个标签，这是将使分组到达尾部路由器的隧道标签。此标签通过 4.3.1 节描述的机制获得。

4）首部和尾部之间的路由器转发只使用隧道标签的分组。

5）尾部路由器删除隧道标签，找到多路分解标签，并识别出仿真电路。

6）尾部路由器将 ATM VCI 修改为正确的值（本例中是 202），然后将它从正确的端口发出。

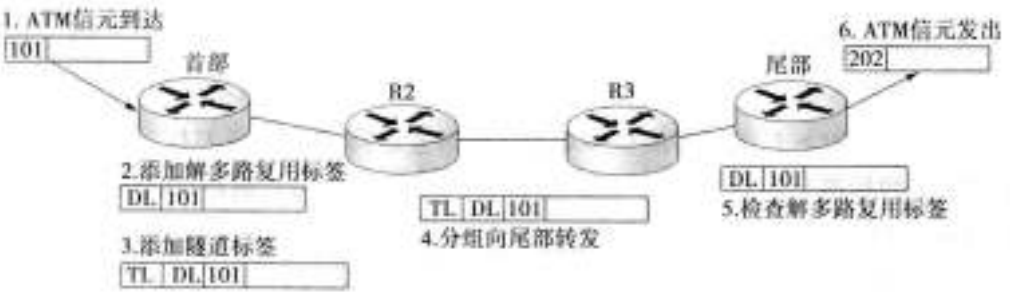


图 4-24 沿隧道转发 ATM 信元

在这个例子中，可能令人感到奇怪的一点是分组附加了两个标签。这是 MPLS 中一个有趣的特点，标签可以在一个分组中堆叠到任意深度。这提供了某些有用的可扩展的能力。在此例中，这使得一个隧道可能包含大量的仿真电路。

这里描述的技术可以用来仿真很多其他第二层服务，包括帧中继和以太网。值得注意的是，也可以使用 IP 隧道来提供本质上相同的能力。这里，MPLS 的主要优点是隧道首部较短。

在 MPLS 被用于建立隧道第二层的服务之前，它也用于支持第三层的 VPN。我们不在这里解释第三层 VPN 的细节，因为它很复杂，更多信息的来源参见“扩展阅读”，但是我们将指出，它们代表当今 MPLS 最流行的一种应用。第三层 VPN 也使用 MPLS 标签栈使分组经隧道在 IP 网络中传送。然而，使用隧道传送的分组本身就是 IP 分组，因此叫第三层 VPN。在第三层 VPN 中，一个服务提供商运营着支持 MPLS 的路由器网络，为任何数目的不同客户提供“虚拟专用”IP 网络服务。也就是说，提供商的每个客户都有多个站点，服务提供商给每个客户都造成了网络上没有其他客户的错觉。客户看到 IP 网络只与它自己的站点互联，而不连接其他站点。这意味着每个客户在路由和编址上与其他客户

都是隔离的。客户 A 不能直接发送分组到客户 B，反之亦然。[⊖] 客户 A 甚至可以使用客户 B 使用过的 IP 地址。其基本思想表示在图 4-25 中。像在第二层 VPN 中一样，MPLS 用于将分组通过隧道从一个站点发送到另一个站点。然而，隧道的配置是通过对 BGP 的某些相当复杂的运用自动完成，这一点超出了本书的范围。

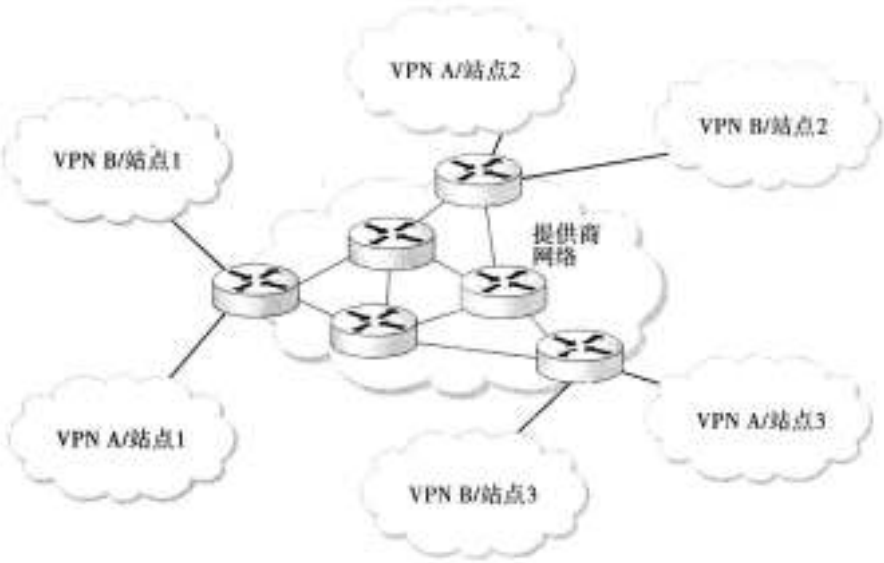


图 4-25 一个第三层 VPN 的例子。客户 A 和客户 B 从一个提供商得到虚拟专用 IP 服务

总而言之，MPLS 是用于解决很多不同的网络互联问题的通用工具。它将通常与虚电路网络相联系的标签交换转发机制与 IP 数据报网络的路由和控制协议结合起来，产生一类具有两者折中性质的网络。这样就能扩展 IP 网络的能力，包括使路由的控制更为精确，并且支持一系列 VPN 服务。

它们现在在哪？

部署 MPLS

MPLS 最初的设想是隐形于当今的互联网中，对大多数客户及学者用户透明，而只提供独立服务提供商进行网络内操作的技术。然而，现在几乎是强制性地在高端路由器制造商的产品中包含 MPLS 功能，并在服务提供商中广泛应用。至少对于关注公共互联网的学生和研究人员而言，MPLS 技术的广泛成功得益于良好的保密工作。

MPLS 的两个主要应用说明了 MPLS 的部署。在本节介绍的第三层 VPN 应用是 MPLS 的重量级应用。现在世界上几乎每一个服务提供商都提供基于 MPLS 的第三层 VPN 服务。由于第三层 VPN 服务是向一些公司提供给专门的 IP 服务而不是提供全球因特网连接，所以通常运行于与因特网分离的路由器上。然而，一些提供商通过一个共同的骨干网运行互联网服务和 VPN 服务。

⊖ 事实上，客户 A 通常可以通过受限途径发送数据给客户 B。最有可能的是客户 A 和客户 B 具有到整个因特网的连接，且客户 A 还可以发送电子邮件，比如发给客户 B 网络内部的邮件服务器。VPN 提供的隐私服务可阻止客户 A 非受限访问客户 B 网络中的所有机器和子网。

MPLS的第二大用途是显式路由，无论是流量工程或快速重路由，或者两者兼而有之。不同于明确地销售给终端客户的第三层VPN服务，显式路由是提供给提供商的提高其网络的可靠性和降低成本的内部能力。提供商通常不公开他们的内部网络的设计细节，因此很难确定有多少提供商实际使用此技术。很显然，使用MPLS的显式路由功能的提供商比使用VPN功能的要少很多，但显示路由仍然有充分的应用，特别是当带宽昂贵的时候或者需要维持低拥塞需求的时候（例如支持实时服务）。

4.4 移动设备之间的路由[⊖]

面对移动设备对因特网体系结构提出的挑战时我们不必过于吃惊，因为因特网是在计算机体型庞大、不可移动的时候设计的，即使因特网设计者意识到移动设备可能在未来出现，也大可不必将兼容这些设备作为首要任务。当然，今天手提电脑到处都是，尤其是在笔记本电脑和具备IP功能的手机，且还在以更多的其他形式出现，比如传感器。在本节中，我们将着眼于一些由移动设备的出现而带来的挑战，以及一些应对挑战的方法。

4.4.1 移动网络的挑战

本书的大部分读者可能已经使用过移动网络设备，对许多人来说，移动设备已经成为常态。所以有理由认为移动网络是一个已经解决的问题。当然，今天很容易发现一个无线热点并使用802.11或一些其他的无线网络协议与之连接，从而获得很好的网络服务。使热点成为可能的关键技术是DHCP（详见3.2.7节）。你能坐在咖啡馆，打开你的笔记本并获取IP地址，使笔记本与一台默认路由器和一个域名系统（DNS）服务器（见9.3.1节）连接，运行应用程序获取需要的东西。

然而如果我们仔细观察，对于某些应用场景来说，每次移动时只是由DHCP得到一个新的IP地址很明显是不够的。假设你正在用笔记本或智能手机打IP电话，你会从一个热点转到另一个，甚至从802.11转换到3G无线网络连接。

显然，当你从一个接入网络移动到另一个时，你需要一个对应新网络的新的IP地址。但是，与你通信的另一端的计算机或电话不能立即知道你已移动以及你的新IP地址。因此，如果缺少相应的机制，会使分组继续发送到你之前使用的是地址，而不是你现在的地址。这个问题如图4-26所示，当移动节点从4.26a中的802.11网络移动到4.26b中的蜂窝网络时，通信节点（correspondent node）分组需要找到通向移动节点所在的新网络的途径。

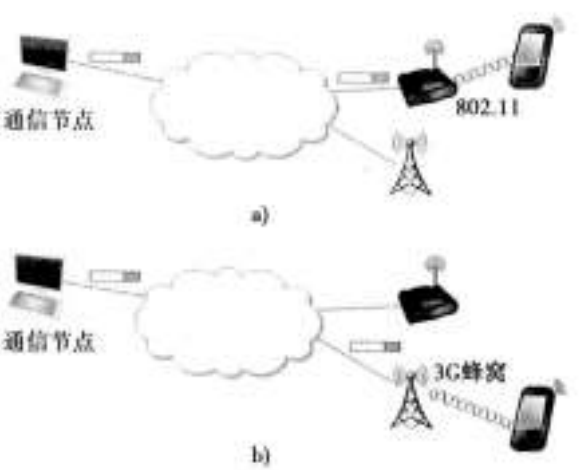


图4-26 从一个通信节点向一个移动节点转发分组

现在有许多不同的方式来解决刚才描述

⊖ 参见"实验九"。

的问题，我们会在下面讨论这些方法。假设有一些方法来重定向分组以便它们到达你的新地址而不是旧地址，随之而来的问题将涉及安全性。例如，如果有一种机制使得我可以说“我的新 IP 地址是 X”，那么如何防范攻击者在没有得到我许可的情况下发布这样的消息，从而导致其接收我的分组或重定向转发到一些不知情的第三方？由此可见，安全性和移动性之间存在密切的关系。

上述讨论强调的重点是 IP 地址实际上承担了两个任务。它们不但作为终端的标识符（identifier），也用于定位（locate）终端。我们将标识符看作终端的长期命名，而将定位器看作有关如何将分组路由至终端的可能的临时信息。只要设备不动或不经常移动，两个任务都使用一个单一的地址似乎很合理。但是，一旦设备开始移动，你宁愿要一个不随移动而变化的标识符，这种标识符可以称为*终端标识符*（endpoint identifier）或*主机标识符*（host identifier）。另外还有分离的*定位器*（locator）。将定位器与标识符分离的想法已经存在了很长时间，下面所描述的大多数处理移动问题的方法都提供了某种形式的分离。

IP 地址不会改变的假设存在于不同的地方。例如，我们将在下一章讨论的像 TCP 这样的传输协议都假设 IP 地址在连接的整个生命周期中保持不变，因此在移动世界操作的传输协议需要重新评价这个假设。

虽然我们都熟悉移动终端，但值得注意的是路由器也可以移动。这当然比今天的移动终端少见，但在很多环境中移动路由器可能是有意义的。一个例子是应急小组试图在某些自然灾害已经破坏了所有固定基础设施之后部署网络。当网络中的所有节点都是移动的，而不止是终端时，我们还需要考虑其他问题。我们将在 4.4.2 节讨论这个主题。

正如很多技术一样，移动性支持引发了增量部署问题。在网络发展的前几十年中，因特网全部由不移动的节点组成，而且假设在可预见的未来有很多路由器和主机仍然如此是很合理的。因此，移动解决方案需要处理增量部署。相反，IPv6 可以在一开始时就将移动性作为其能力的一部分而加入，从而具有诸多优势。

在我们开始探索支持移动设备的方法之前，需要澄清几点。通常人们会对无线网络和移动产生混淆。虽然移动和无线因为显而易见的原因通常会同时出现，但无线连接其实是在没有连线的情况下从 A 到 B 获得数据，正如第 2 章讨论的那样。这里移动性是指当一个节点在通信过程中移动的时候要处理的问题。当然很多使用无线信道的节点并不具有移动性，且有时移动节点会使用有线通信（虽然这种状况不一定常见）。

最后我们在本章很可能会关注称为*网络层移动性*（network-layer mobility）的问题，也就是关注如何处理从一个网络转移到另一个网络的节点。正像我们在 2.7 节看到的，对于在同一个 802.11 网络中从一个接入点转移到另一个接入点的情况，可以用特定于 802.11 的机制处理，当然移动电话网也同样能解决移动性问题。但对于像因特网这样的异构系统，我们就需要对更广泛网络间的移动性提供支持。

4.4.2 路由到移动主机（移动 IP）

移动 IP 是当前互联网体系结构解决移动主机分组路由的基础机制。它引入了一些新功能，但不需要对非移动主机和大多数路由器做任何改变，从而解决之前提出的增量部署问题。

移动主机是假设有一个称为*本地地址*（home address）的永久 IP 地址，这个地址拥有与它的*本地网络*（home network）相同的网络前缀。其他主机首先发送分组给移动主机时

使用这个地址。因为这个地址不会改变，所以在主机漫游时，长期运行的应用可以使用它。我们可以将之看作主机的长期标识符。

当主机从本地网络移动到一个新的外地网络时，通常会使用类似 DHCP 的方式获得一个新的网络地址。[⊖] 这个地址在主机每次漫游时都会改变，因此我们可以认为它更像是主机定位器，但重要的是要注意主机在申请一个外地网络的新 IP 地址时并不丢弃其本地地址。正如我们下面将要看到的，本地地址决定了主机在移动时维持通信的能力。

虽然大部分路由器保持不变，但支持移动性需要至少有一台路由器添加一些新的功能，这台路由器称作移动节点的本地代理（home agent），位于移动主机的本地网络。在某些情况下还需要另一个增强功能的路由器作为外地代理（foreign agent），它位于移动节点离开本地网络后连接到的那个网络。我们先考虑使用外地代理时移动 IP 的操作。一个具有本地代理和外地代理的网络的例子如图 4-27 所示。



图 4-27 移动主机和移动代理

本地代理和外地代理周期性的采用关联的通知信息公布其存在。一个移动主机也会在关联一个新网络时发布通知信息。本地代理所发的通知使一个移动主机可以在离开其本地网络前学到本地代理的地址信息。当移动主机关联到一个外地网络时，外地代理则联系本地代理，提供转交地址（care-of-address）。这个地址通常是外地代理的 IP 地址。

在这一点上，我们可以看到，任何主机试图发送一个分组到移动主机时会发送一个与该节点本地地址相同的目标地址。正常的 IP 转发会导致分组到达移动节点的本地代理所设定的本地网络。因此，我们可以将传递数据分组到移动节点问题分为三部分：

1）本地代理如何截取目标为移动节点的分组？

2）本地代理如何传送此分组到外地代理？

3）外地代理如何将此分组传送到移动节点？

第一个问题可能比较简单，如图 4-27 所示，本地代理是从发送主机到内部网络的唯一途径，所以一定能收到以移动节点为目标的分组。但如果发送（通信）节点在网络 18，或者另一个连接到网络 18 的路由器试图不通过本地代理传送分组呢？对于这个问题，本地代理使用了一种称为 ARP 代理（proxy）的技术把自己模拟为移动节点。除了本地代理在 ARP 消息中插入移动节点的 IP 地址而不是自己的地址以外，这项工作很像在 3.2.6 节描述的地址解析协议（ARP）。由于使用自己的硬件地址，所有同一个网络的节点需要关联移动节点的 IP 地址和本地代理的硬件地址。这个过程中微妙的是事实上 ARP 消息可能由其他网络的节点缓存。为确保该缓存时序无效，当移动节点在外地代理注册的同时，本地代理发送了一个 ARP 消息。因为此 ARP 消息并不是通常 ARP 需求的响应，所以称为免费（gratuitous）ARP。

第二个问题是传送截取的分组到外地代理。这里使用的是在 3.2.9 节描述的隧道技术。本地代理简单地封装一个具有目标为外地代理的 IP 首部的分组且分发到互联网。所

⊖ 由于 DHCP 与移动 IP 几乎同时开发，所以原始的移动 IP 标准不需要 DHCP。不过，今天 DHCP 已经无处不在。

有途经的路由器只能看到以外地代理的IP地址为目标的IP分组。换个角度看，IP隧道建立在本地代理和外地代理之间，且本地代理只将目标为移动节点的分组放入隧道。

当一个分组最终抵达外地代理时，被封装的额外IP首部被去除而露出移动节点的目标本地代理的IP分组。显然外地代理不能处理旧的IP分组，因为这将导致分组被发送回本地代理。外地代理会识别出已注册的移动节点的地址，然后转发分组到移动节点的硬件（hard ware）地址（比如以太网地址）。这个过程是注册过程的一部分。

对于这些过程，可以看到的一种情况是外地代理和移动节点被放在一起。移动节点本身可以实现外地代理功能。此时移动节点必须能动态申请一个外地网络的IP地址（比如使用DHCP）。此地址被作为转交地址使用。在本例中，这个地址的网络号为12。此种方法具有允许移动节点归属于没有外地代理的网络的良好特性。因此，只需增加本地代理和移动节点上的一些新软件就可以获得移动性（假设外部网络使用DHCP）。

那么发送到其他方向的流量如何处理呢（比如由移动节点到固定节点）？这比之前讨论的要更容易。移动节点仅仅在IP分组的源区域放置永久地址，同时放置固定节点的IP地址到目标地址，则分组即可转发到固定节点。当然，如果两个节点都是移动的，这个过程在单个方向上分别采用。

1. 移动IP路由优化

上述方法有一个重大缺陷：从相关节点到移动节点的路由明显不是最优的。一个极端的例子是当一个移动节点和发送节点在同一个网络中，但移动节点的本地网络在互联网远端的情况。发送方通信节点将所有分组地址设为本地网络地址，这些分组需要通过互联网到达本地代理，然后通过隧道穿过互联网到达外地代理。如果通信节点可以确认移动节点在同一个网络且能直接传输分组的话，这样显然很好。但在大多数情况下，目标是尽可能直接地从通信节点而不通过本地代理传送分组。很多情况下这被认为是一个三角路由问题（triangle routing problem），因为从通信节点通过本地代理到移动节点经过了一个三角形的两条边，而第三条边是最直接的途径。

解决三角路由的基本思路是使通信节点获取移动节点的转交地址。通信节点可以创建自己的通向外地代理的隧道，且这被认为是优化解决方法。如果发送者安装了必要的软件来学习转交地址并创建自己的隧道，则路由可以被优化。反之，分组采用次优的路由。

当一个本地代理看到一个分组且此分组目标为自己支持的移动节点时，本地代理可以推断出发送者是否采用优化路由。因此，本地代理除了转发数据分组到外地代理外，还向源发送“绑定更新”消息。源如果具备此能力，则使用此绑定更新创建一条绑定缓存（binding cache）记录，其中存放从移动节点地址到转发地址的映射列表。下次这个源要发送数据分组给移动节点时，就能从缓存中找到该绑定且通过隧道直接发送分组到外地代理。

这个方案存在一个明显的问题，就是当移动主机转到新网络的时候，绑定缓存可能过时了。如果过时信息被利用，外地代理将从隧道中收到发往移动节点的分组，但该分组已不在此网络注册。此种情况下，移动主机将发送一个绑定警告（binding warning）消息给发送者并告知其不再使用此缓存记录。这个方案只在外地代理不是移动节点本身时使用。所以，缓存记录需要在一段时间后删除，确切的时间长度在绑定更新消息中提供。

如上所述，移动路由提出了一些有趣的安全挑战，现在我们已经看到移动IP是如何工作的，这些挑战变得更清晰了。比如，一个攻击者希望截取发送到互联网其他节点的分

组时，可能以自己是该节点新外地代理的身份联系该节点的本地代理。因此，很明显这里需要一些认证机制。我们将在第 8 章中讨论该机制。

2. IPv6 的移动性

IPv4 和 IPv6 对移动性的支持存在很大差别。最重要的是，可以从头开始在 IPv6 标准中创建移动性支持，这减轻了一部分增量部署问题（或者更准确地说，IPv6 是一个大增量部署问题，如果得到解决，则可将移动支持作为分组的一部分）。

由于具有 IPv6 功能的主机在接入外地网络时可以获得一个地址（采用 IPv6 内核规范中定义的多种机制），所以移动 IPv6 去除了外地代理且使得每台主机具备扮演外地代理角色的能力。

IPv6 作为移动 IP 的另一个有趣方面是包含了一组灵活的扩展首部，正如在 4.1.3 节介绍的那样。这可用于上述优化路由的场景中。不同于通过转交地址建立隧道将分组发送到移动节点，IPv6 可以通过包含在路由首部的本地地址发送 IP 分组到转发地址。这个首部会被中间节点忽略，但移动节点会把分组当作发送给本地地址一样来处理，可继续用固定 IP 来传递给高层协议。采用扩展首部而不是隧道技术可以使带宽占用和处理更经济。

最后，我们介绍一些移动网络中的开放性问题。移动设备的电源消耗管理越来越重要，这样具有有限电量且体积更小的设备才能被制造出来。移动自组（ad hoc）网也存在很多特殊挑战（见相关主题），这种网络在没有任何固定节点时使一组移动节点组成网络。还有一些移动网络、传感器网络（sensor network）类的挑战曾在以前提起过。传感器一般体积小、昂贵且由电池供电，这意味着需要考虑很多低能耗和有限处理能力的问题。另外，既然无线通信和移动性通常会齐头并进，持续的无线技术发展将使移动网络产生新的挑战和机会。

相关主题

移动自组网

本节的大部分篇幅中会假设只有终端节点（主机）是移动的，这可以和当今网络中我们处理的大部分情况吻合。我们的笔记本电脑和手机四处移动且和固定的网络设施相连接，比如通过固定链路连接到互联网骨干网的发射塔和 802.11 接入点。然而，很多现代路由器做得很小且足以移动，可以用在移动网络环境，比如在移动车辆之间构建网络。因为路由协议是动态的，所以可以想象移动路由不应成为问题，而且这大致是正确的。然而，如果所有或者大部分网络节点都是移动的呢？极端逻辑下，存在一个完全没有固定设施的网络，只有一些移动节点，其中一些或全部用作路由器。标准的路由协议能在这样的环境下工作吗？

这种没有固定设施且所有设备都是移动的环境被命名为移动自组网（MANET 是解决该问题的 IETF 工作组的名字）。为了理解为什么移动自组网需要特殊的解决方案，我们可以想象以下状况：不像固定网络，任何给定的自组网路由器的邻居因为节点移动而经常变化。既然任何邻居关系变化都需要一个路由协议消息发送并计算新的路由表，因此很容易发现必须考虑使用一个对环境没有优化的协议。加剧这个问题的事实是通信是无线的，要消耗电量，而且很多移动设备可能会耗尽电池电量。链接带宽可能也是有限的。因此，减少发送路由协议消息以及重新洪泛给所有邻居引起的开销是自组路由需考虑的关键问题。

在撰写本书的时候，很多移动自组网的优化路由方法被提出。这些方法可以分为被动方法和主动方法。优化链路状态路由（Optimized Link State Rount，OLSR）是主流的主动方法，且通过其名字能够感觉出它的内容。OLSR 将传统链接状态协议（如 OSPF，见 3.3.3 节）和很多减少洪泛路由消息的优化方法组合在一起。被动协议包括自组织按需距离向量（Ad Hoc On-Demand Distance Vector，AODV）和动态按需移动自组网（Dynamic MANET On Demand，DYMO），两者都是基于 3.3.2 节介绍的距离向量协议。这些方法专注于仅在需要时才建立路由来减少路由协议开销总量，比如当一个给定节点向特定目的地发送消息时，有丰富的解决方案空间可供人们权衡选择，而且这个空间处于持续的探索当中。

4.5 小结

本章的主题是处理因特网的持续增长问题。因特网持续连接更多的用户，且每个用户发送更多的应用流量（如视频流等），因此需要更多带宽。虽然因特网被证明是一个具有很强扩展性的系统，但新的可扩展性问题也持续地需要解决方案。除了可扩展性，因特网还需要不断进化来支持新的能力和服务。

当前主要的扩展性问题是地址空间的有效使用和路由表随互联网增长而增长的问题。层次化 IP 地址格式，包括网络部分和主机部分，为我们管理可扩展性提供了一个层次。路由区则提供了另外一个层次。自治系统允许我们将路由问题划分为两个部分，域内路由和域间路由，每一种路由都比总体的路由问题小很多。因特网的域间路由协议 BGP 在处理因特网增长方面取得了极大的成功。

尽管采用了多个步骤来扩展 IPv4，但显然很快就需要一个新的更长的地址格式。这需要新的 IP 数据报格式和一个新的协议。新协议被命名为 IPv6，开始时被称为下一代 IP（IPng）。IPv6 提供 128 位地址，且具有基本类似 CIDR 的寻址和路由方法。虽然 IPv6 发布了诸多新能力，但其主要优势在于支持超大数量可寻址设备。

最后，因特网不但要解决可扩展性问题，还需要功能方面的进化。在这方面，我们看到原始 IP 数据报模型上三个方面的增强。首先是多播，使同一组数据可以高效地传输给一组接收者。正如单播一样，多播面临的很多挑战都与可扩展性相关，人们开发了大量不同协议和多播模型，用以优化不同环境下的扩展性及路由。第二是 MPLS，将虚电路网络中的一些理念引入 IP 且广泛应用于扩展 IP 能力。MPLS 的应用范围覆盖了从流量工程到支持因特网上的虚拟专用网等。最后是移动性支持，这个特性与最初 IP 设计者的想法相去甚远，但这个特性随着可移动联网设备（包括主机和路由器）的增多而变得越来越重要。

接下来会发生什么：部署 IPv6

自从 IPv4 地址空间不足变得非常严重以至于需要一个新版的 IP 以来，已经过去了大约 20 年的时间。最初的 IPv6 规范也存在了超过 15 年。现在有很多支持 IPv6 的主机操作系统，并且主要的路由器生产商也在他们的产品中对 IPv6 提供不同程度的支持。至写本书时止，IPv6 还没有在因特网上进行真正意义的部署。什么时候开始进行实实在在的部署以及由什么原因引发这种部署，这是值得人们关注的。

IPv6 并不那么急需的原因之一是 NAT（网络地址转换，本章前面有所描述）的广泛使用。因为提供商将 IPv4 地址看作稀有资源，所以只将很少的一部分分配给用户，或者按使用的地址数量收费；客户的很多设备被隐藏到 NAT 设备之后，只使用一个 IPv4 地址。例如，这就像有多个使用 IP 地址的设备的家庭网络，在网络中使用某种 NAT 来节约地址。因此，可能促使部署 IPv6 的一个因素是那些使用 NAT 时不能很好工作的应用。对于客户端-服务器应用，当客户地址隐藏在 NAT 盒之后工作得非常好，而对等应用则稍差一些。例如，多人游戏和 IP 电话就是不使用 NAT 可以工作得更好，并因此从更宽松的地址分配策略中受益的应用实例。然而，即使这些应用已经可以不使用 NAT，但 NAT 技术已广泛存在。

近些年来，获得成块的 IPv4 地址已经变得更加困难，特别是在美国以外的其他国家尤为明显。随着困难变得越来越大，提供商为客户提供 IPv6 地址的动机也随之出现。同时，对于现有的提供商来说，提供 IPv6 是很大的附加开销，因为当他们开始提供 IPv6 时并不能终止对 IPv4 的支持。这就意味着提供商路由表的尺寸可能会增长，因为它们必须携带所有现有的 IPv4 前缀，再加上 IPv6 前缀。

现在，IPv6 的部署几乎只发生在研究性网络上。一小部分服务提供商开始提供 IPv6（通常带有一些政府行为的动机），特别是在美国以外。商业路由器和主机操作系统不同程度地支持 IPv6。可以肯定，IPv6 的部署会加快，但似乎很可能绝大多数主机和网络将至少在很多年中都只运行 IPv4。

扩展阅读

由 Bradner 和 Mankin 撰写的 RFC 是扩展阅读的首选，它综述了高速增长的因特网如何对原有体系结构的扩展能力施压，并最终产生了 IPv6。Paxson 的文章描述了路由器在因特网上的行为研究，虽然写于 15 年之前，但该文章仍然被高频率地引用且给出了很好的研究人员如何学习因特网动态行为的例子。最后一篇文章讨论多播，解释了 MBone 上最初使用的多播方法。

- Bradner, S. , and A. Mankin. The recommendation for the next generation IP protocol. *Request for Comments* 1752, January 1995.
- Paxson, V. End-to-end routing behavior in the Internet. *SIGCOMM'96*, pages 25-38, August 1996.
- Deering, S. , and D. Cheriton. Multicast routing in datagram internetworks and extended LANS. *ACM Transactions on Computer Systems* 8 (2): 85-110, May 1990.

一些有趣的关于因特网路由行为的实验研究在 Labovitz 等 [LAAJ00] 进行了解释。另外一些关于 BGP 稳定性的有用文章可见 Gao 和 Rexford [GR01]。

IPv6 相关的 RFC 集合可见 Bradner 和 Mankin [BM95]，一些新的 IPv6 规范可见 Deering 和 Hinden [DH98]，还有很多其他的 IPv6 相关 RFC。

协议无关多播（PIM）在 Deering 等 [DEF+96] 和 Fenner 等 [FHHK06] 进行了描述，PIM-SSM 在 [Bha03] 进行了描述。[Wil00] 和 [HC99] 都是很好的介绍多播历史细节的读物。

MPLS 相关协议和开发可参考 Chandranmenon 等 [CV95]，Rekhter 等 [RDR+97] 和

Davie 等 [DR00]。后面的文献描述了 MPLS 的一些应用，如流量工程、网络故障的快速恢复以及虚拟专用网等。[RR06] 提供了 MPLS/BGP VPN 的规范，它是 MPLS 网络上能够提供的第三层 VPN 的一种形式。

最后，我们推荐下列时常更新的网站作为参考：

http://www.isoc.org/internet/history/：一个与互联网历史相关的链接集合，包括最初构建互联网的研究人员写的文章。

http://bgp.potaroo.net/：许多互联网路由表增长数据，包括 IPv6 部署。

习题

1. 如图 4-28 所示，网络中水平线表示传输提供商，几条垂直线为提供商之间的链接。

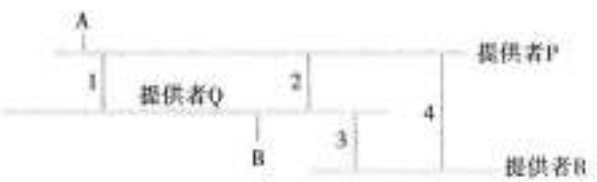


图 4-28 习题 1 的网络

   (a) 提供商 Q 的 BGP 代言人可收到几条到 P 的路由？

   (b) 假设 Q 和 P 采用的策略是将输出流量路由至最接近目标提供商的链接，以此实现最小化开销。那么从主机 A 到主机 B 和从主机 B 到主机 A，分别会采用哪条路径？

   (c) 为使 B→A 的传输使用更近的链路 1，Q 应如何做？

   (d) 为使 B→A 的传输经过 R，Q 应如何做？

2. 给出一个将路由器组成自治系统的例子，使从 A 点到 B 点的最少跳路径通过同一个 AS 两次。解释 BGP 此时该如何做。

★ 3. 令 A 为因特网中的自治系统的数目，D（直径）为 AS 路径长度的最大值。

   (a) 分别给出 D 为 $\log A$ 阶和 $\sqrt{A}$ 阶时的连通性模型。

   (b) 假设每个 AS 号为 2 字节，每个网络号为 4 字节，试估计一个 BGP 代言人为了解到每个网络的 AS 路径而必须接收的数据量。将答案表示为 A、D 和网络数 N 的表达式。

4. 为 IPv6 提出一个合理超位数运行的方案，尤其是提供如图 4-11 所示的数据报，可具有额外的 ID 字段，使之可多于 128 位，同时可调整各个字段的尺寸。你可以假设字段按字节边缘划分，且 InterfaceID 是 64 位（提示：可认为字段只有在异常情况下才会以最大空间分配）。如果 InterfaceID 是 48 位呢？

5. 假设 P、Q 和 R 是网络服务提供商，各自的 CIDR 地址分别为 C1.0.0.0/8、C2.0.0.0/8 和 C3.0.0.0/8。每个提供商的客户最初接收的地址分配是提供商地址的一个子集。P 有如下客户：

   PA，分配地址 C1.A3.0.0/16

   PB，分配地址 C1.B0.0.0/12

   Q 有如下客户：

   QA，分配地址 C2.0A.10.0/20

   QB，分配地址 C2.0B.0.0/16

   假设没有其他提供商和客户。

   (a) 给出 P、Q 和 R 的路由表，假设每个提供商都和另外两个提供商连接。

   (b) 现在假设 P 连接 Q，Q 连接 R，但 P 和 R 不直接连接。给出 P 和 R 的路由表。

   (c) 假设除现有链路外，客户 PA 需要一条到达 Q 的直接链路，且 QA 需要一条到达 P 的直接链路。给出 P 和 Q 的路由表，忽略 R。

6. 上一道题目中，假设每个提供商都与另外两个连接。假设客户 PA 转到提供商 Q 且客户 QB 转到提供商 R。应用 CIDR 最长匹配规则给出路由表，使所有三个提供商允许 PA 和 QB 交换且无需重编号。

7. 假设大部分因特网使用某种形式的地理编址，但一个大型国际组织拥有一个 IP 网络地址且在自己的链路中路由内部流量。

(a) 解释该组织在此种状况下输入流量路由的低效性。

(b) 解释如何解决输出流量问题。

(c) 用解决问题 (b) 的方法解决输入流量，会发生什么？

(d) 假设该组织将各地办公室的地址更换为分离的地理地址，如果仍需要在组织内路由内部流量的话，他们的内部路由体系结构必须是怎样的？

8. 电话系统使用地理编址。解释你不认为这是理所当然的因特网体系结构的原因。

9. 假设一个小 ISP X 向大 ISP A 付费以接入因特网，同时从另一个 ISP B 处购买了备用链路以便在与 A 的链接断开时使用。如果 X 通过 ISP A 学习到通往一些前缀的路径，需要将这些路径通知 B 吗？为什么？

10. 假设站点 A 是多连接的 (multihomed)。它有从不同的提供商 P 和 Q 到因特网的两个连接。使用习题 5 中的基于提供商的编址方法，且 A 从 P 得到其地址分配。Q 有一条 A 的 CIDR 最长匹配路由记录。

(a) 描述哪些输入流量流入 A-Q 连接。考虑 Q 使用 BGP 将 A 和不将 A 向全网通知的情况。

(b) 如果 P-A 链路出现故障，为使所有输入流量都能经 Q 到达 A，Q 必须向 A 至少通知哪些路由？

(c) 如果 A 使用这两条链路来传送输出流量，必须克服哪些问题？

★ 11. 假设在一个大型组织 A 中的网络 N，除了现有的经 A 的连接外，还有它自己的到因特网服务提供商的直接连接。设 R1 为连接 N 到其提供商的路由器，R2 为连接 N 到 A 的其余部分的路由器。

(a) 假设 N 仍是 A 的子网，R1 和 R2 应如何配置？对于 N 使用它自己的独立连接，仍将存在什么限制？A 不能使用 N 的连接吗？说明你的配置，包括 R1 和 R2 应通知什么，以及使用哪些路径进行通知。假设可使用像 BGP 这样的机制。

(b) 现在假设 N 有自己的网络号，你在 (a) 中得出的答案将如何变化？

(c) 描述当 A 自己的链路发生故障后，允许它使用 N 的链路的一台路由器配置。

12. 路由器如何确定接收到的 IP 分组为多播？针对 IPv4 和 IPv6 给出答案。

13. 假设一个多播组希望被一个特定多播域隐藏，那么一个被指派到该组的 IP 多播地址如何能够在没有与其他多播域协商的情况下避免冲突？

14. 在何种情况下一台非路由器主机在以太网上可以从未加入的多播组接收 IP 多播分组？

15. 考虑如图 4-29 所示的因特网的例子，其中源主机 D 和 E 向多播组 G 发送分组，除 3D 和 E 其余主机全部是组 G 的成员。给出每个源的最短路径多播树。

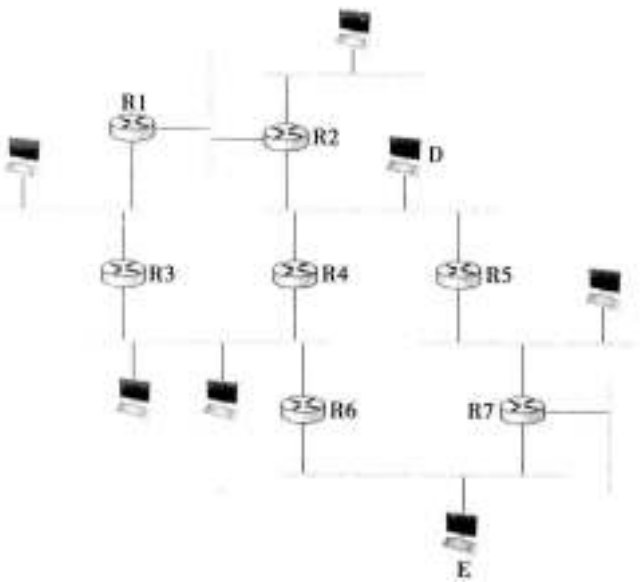


图 4-29 习题 15 的示例网络

16. 考虑如图 4-30 所示的因特网的例子，其中源主机 S1 和 S2 向多播组 G 发送分组，除了 S1 和 S2 其余主机全部是组 G 的成员。给出每个源的最短路径多播树。

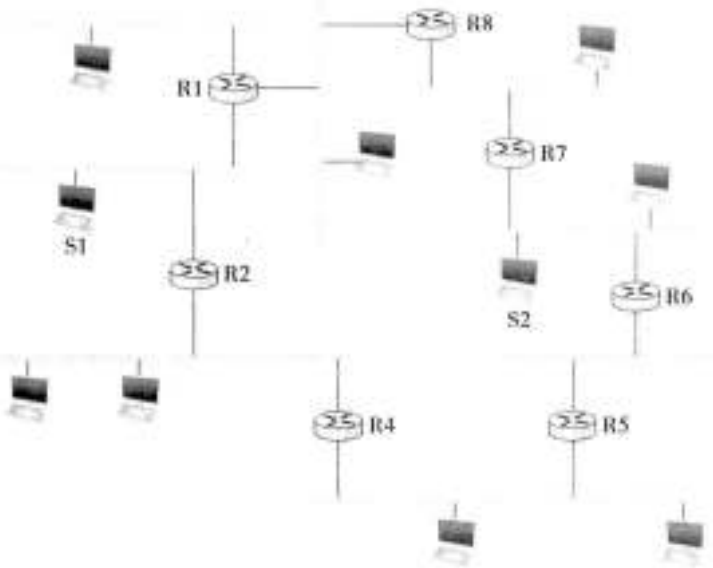


图 4-30 习题 16 的示例网络

17. 假设主机 A 正在向一个多播组发送消息。接收方是以 A 为根的树上的叶子节点，树的深度为 N，且每个非叶子节点有 k 个子女，因此有 k^N 个接收方。
    (a) 如果 A 向所有接收方发送一条多播消息，会涉及多少条独立的链路传输？
    (b) 如果 A 向每个接收方发送单播消息，会涉及多少条独立的链路传输？
    (c) 假设 A 向所有接收方发送，但是有一些消息丢失且需要重传。仅考虑链路重传方面，向接收方哪部分的单播重传等价于向所有接收方的一次多播重传？
18. 现有的因特网在很大程度上依赖于参与者成为高素质的“网络公民”——共同坚持操作所有甚至更多的标准协议。
    (a) 在 PIM-SM 方案中，谁决定什么时候创建特定源的树？这在什么情况下出现问题？
    (b) 在 PIM-SMM 方案中，谁决定什么时候创建特定源的树？为什么可以假设不存在这个问题？
19. (a) 画一个互联网的例子，在该网络中从一个源路由器到组成员路由器的 BIDIR-PIM 路由的长度比 PIM-SM 特定源的路由要长。
    (b) 画一个两种路由相同的例子。
20. 判断以下 IPv6 地址的表示是否正确。
    (a) ::0F53:6382:AB00:67DB:BB27:7332。
    (b) 7803:42F2:::88EC:D4BA:B75D:11CD。
    (c) :::4BA8:95CC::DB97:4EAB。
    (d) 74DC::02BA。
    (e) ::00FF:128.112.92.116。
21. MPLS 标签一般长 20 位。请解释当 MPLS 用于基于目的地址的转发时，为什么这个长度能提供足够的标签。
22. MPLS 有时被认为改进了路由器的性能。请解释为什么，以及为什么实际中可能不是这样。
23. 假设每个 MPLS 标签占 32 位，并作为图 4-20b 中曾使用过的“垫”首部添加到一个分组中。
    (a) 通过隧道传送分组时，如果使用 4.3.3 节描述的 MPLS 技术，需要多少附加字节？

(b) 通过隧道传送分组时，如果使用3.2.9节描述的附加IP首部，最少需要多少附加字节？

(c) 当平均分组长度为300字节和64字节时，分别计算如上两种方法的带宽利用率。带宽利用率定义为（传送的有效载荷字节数）÷（传送的总字节数）。

24. RFC791描述了IP协议，并包括源路由的两个选项。请说出与使用MPLS的显式路由相比，使用IP源路由选项的三个缺点。（提示：包括选项的IP首部可能最多15个字长。）

25. DHCP允许计算机在任意时刻移动到新的子网时请求一个新的IP地址。为什么这样无法满足移动主机的通信需求？

26. 请求将传送到一个移动节点的流量先发送给本地代理的主要缺点是什么？

27. 移动IP允许本地代理通知相关通信节点一个移动节点的新的转交地址。如何利用此机制盗取流量？如何将之用于执行对其他节点的一次洪泛流量攻击？

第5章
Computer Networks: A Systems Approach

端到端协议

胜利是美丽灿烂的花朵，运输是枝干，没有它胜利之花是不会开放的。

——温斯顿·丘吉尔

问题：进程间如何通信

前面三章描述了可用于把多台计算机连接在一起的各种技术，从简单的以太网和无线网到覆盖全球的互联网。下面要考虑的问题是从这种主机到主机的分组传递服务转向进程到进程的通信信道，这正是网络体系结构中传输（transport）层的任务，由于它支持在终端节点上运行的应用程序之间的通信，因此传输层协议有时也称为端到端（end-to-end）协议。

两种因素促成了端到端协议的形成。从其上层看，需要使用传输层服务的应用层进程有一些特定的需求。下面列出了上层协议希望传输层能提供的一些常用的特性：

- 确保消息成功传输。
- 消息按序传输。
- 最多传送每个消息的一个副本。
- 支持任意大的消息。
- 支持发送方与接收方之间的同步。
- 允许接收方对发送方进行流量控制。
- 支持每台主机上的多个应用进程。

注意，这个列表并没有包括应用进程要求网络提供的全部功能。例如，没有包括诸如身份验证或加密这类通常由传输层之上的协议来提供的安全特性。

从其下层看，传输层协议赖以运行的下层网络所能提供的服务能力有某些限制。其中比较典型的是下层网络可能会：

- 丢弃消息。
- 使消息乱序。
- 传送一个消息的多个副本。
- 限制消息的大小。
- 在任意长延迟后才发送消息。

这样的网络称为是提供尽力而为（best-effort）的服务，因特网就是这种网络的一个实例。

因此，问题的关键是设计出各种算法，把下层网络低于要求的特性转变成应用程序所需的高级服务。不同的传输层协议应用这些算法的不同组合。本章在以下四种有代表性的服务环境中考察这些算法：简单异步多路分解服务、可靠字节流服务、请求/应答服务和用于实时应用的服务。

对于多路分解服务和字节流服务，分别以因特网中的用户数据报协议（User Data-

gram Protocol，UDP）和传输控制协议（Transmission Control Protocol，TCP）为例来阐述在实际应用中如何提供这些服务。对于请求/应答服务，我们讨论它在远程过程调用（Remote Procedure Call，RPC）服务中的作用及其特征，讨论将围绕着两个广泛使用的RPC协议SunRPC和DCE-RPC来展开。

实时应用对传输层有特定的需求，例如要求音频或视频能够及时播放，我们将关注该类应用对传输层协议的要求，以及广泛用于该目的的协议：实时传输协议（Real-time Transport Protocol，RTP）。

5.1 简单多路分解（UDP）

可能的最简单的传输协议是把下层网络的主机到主机的传递服务扩展到进程到进程的通信服务。任何主机上都可能运行多个进程，因此该协议至少需要增加一个多路分解功能，以便每台主机上的多个进程能够共享网络。除此之外，传输协议不再向下层网络提供的尽力而为服务增加任何其他功能。因特网提供的用户数据报协议就是这样的传输协议。

在这样的协议中，唯一值得注意的问题是用来标识目的进程的地址形式。虽然可以用操作系统赋予的进程标识符（pid）使进程之间直接相互识别，但这样的方法只可能在一个封闭的分布式系统中有实际价值，即在所有主机上只运行一个操作系统，这个唯一的操作系统给每个进程分配唯一的标识符。一种更通用的也是被UDP采用的方法，是使用一个称为*端口*（port）的抽象定位器，使进程之间能够间接（indirectly）相互识别。其基本思想是，源进程向端口发送消息而目的进程从端口接收消息。

实现该多路分解功能的端到端协议的首部通常包含消息的发送方（源）和接收方（目的）的标识符（端口）。例如，图5-1给出了UDP的首部结构。注意，UDP端口字段只有16位。这意味着最多有64K个可能的端口，显然不够用来标识因特网上所有主机的全部进程。幸运的是，端口只对单台主机有效，而不是在整个因特网上都有效。也就是说，进程实际是通过特定主机上的某个端口（即一个（主机，端口）对）标识的。实际上，这个（主机，端口）对构成了UDP协议的多路分解密钥。

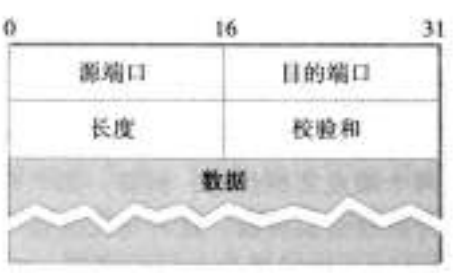


图5-1 UDP首部格式

接下来的问题是一个发送进程如何知道接收进程的端口号。典型情况下，一个客户进程发起与服务器进程的消息交换。一旦客户进程建立了与服务器进程的联系，服务器就能获得客户进程的端口号（包含在消息首部的SrcPort（源端口）字段中），并能对客户进程进行应答。因此，真正的难题是客户进程如何首先知道服务器进程的端口号。通常的做法是，服务器进程在一个*知名端口*（well-known port）接收消息。就是说，每个服务器进程在某个固定的广为公布的端口接收消息，就像美国紧急电话服务可以用众所周知的号码911一样。例如，在因特网上，域名服务器（DNS）总是在端口53接收消息，电子邮件服务在端口25监听消息，UNIX上的talk程序在端口517接收消息，等等。这种服务与端口的对照表定期在RFC上公布，并可以在大多数UNIX系统的/etc/services文件中得到。有时，知名端口仅仅是通信的起点：客户端和服务器用这个端口达成一致，并在另一个端口进行后续的通信，以便释放知名端口给其他客户进程使用。

另一种策略是推广这种思想：知名端口只有一个，就是端口映射（port mapper）服务接收消息的端口。客户端先给端口映射服务程序的知名端口发送消息，询问“无论什么”服务时应使用的端口，而端口映射服务程序返回相应的端口。这种策略使更改各种服务的端口和对每台主机使用不同端口提供相同的服务变得容易。

如上所述，端口纯粹是一种抽象。实际上，它的具体实现在不同的计算机系统中是不同的，或者更准确地说，在不同的操作系统中是不同的。例如，第1章描述的套接字API是端口的一种实现。一般来说，端口是由一个消息队列实现的，如图5-2所示。当消息到达时，协议（例如UDP）会把该消息加到队列的末尾。如果队列满了，消息被丢弃。UDP中没有让发送方减慢发送速度的流量控制机制。当应用程序进程需要接收消息时，就从队列前端移出一条消息。如果队列是空的，进程就阻塞直到有消息可用。

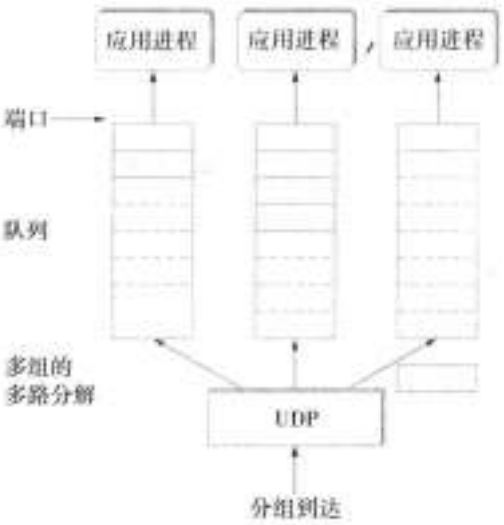


图5-2 UDP消息队列

最后，虽然UDP没有实现流量控制或可靠的/有序的传输，但它除了把消息多路分解给某个应用进程外，还提供了另一种功能——通过使用校验和来确保消息的正确性（UDP校验和在IPv4中是可选项，但在IPv6中是强制性的）。基本的UDP校验和算法与在2.4.2节定义的IP协议校验和算法一致——即将一组16位字以补码形式相加，然后再对相加和取补。但用于校验的输入数据有点不直观。

UDP计算UDP首部、消息体内容和某些场合称为伪首部（pseudoheader）的校验和。伪首部由IP首部的三个字段（协议号、源IP地址和目的IP地址）加上UDP长度字段组成（UDP长度字段在校验和计算中被使用两次）。用伪首部的原因是验证消息已在正确的两个端点之间传输。例如，如果在分组的传递过程中修改了它的目的IP地址，就会造成分组被错误传输，这种情况会被UDP校验和检查出来。

5.2 可靠字节流（TCP）[⊖]

与UDP这样简单的多路分解协议相比，一种更复杂的传输协议提供了可靠的、面向连接的字节流服务。事实证明，这种协议对于众多的各类应用程序是有用的，因为它使应用程序从数据丢失和失序的顾虑中解脱出来。因特网的传输控制协议是这类协议中使用最广泛的协议，也是协调性最精确的协议。因此，本节将详细讨论TCP。在本节最后，我们还指出并讨论了其他的设计选择。

按照本章开始的问题中给出的传输协议的特性，TCP能保证可靠的、有序的字节流传输。它是全双工协议，也就是说每个TCP连接支持一对字节流，每个方向一个字节流。对这两个字节流中的每个流，它还包含流量控制机制，允许接收方限制发送方在给定时间内发送的数据量。另外，像UDP一样，TCP支持多路分解机制，允许任何主机上的多个应用程序同时与它们各自的对等实体进行对话。此外，TCP也实现了一个高度协调（而

⊖ 参见“实验十”。

且仍在发展）的拥塞控制机制。这种机制的思想是控制 TCP 发送方发送数据的速度，其目的不是为了防止发送方发出的数据超出接收方的接收能力，而是防止发送方发出的数据超出网络的容量。关于 TCP 拥塞控制机制的说明将在第 6 章给出，那里我们将在如何公平分配网络资源这一更大的范围内讨论它。

结论 由于很多人混淆了拥塞控制与流量控制，所以在此我们再次说明它们之间的区别。流量控制（flow control）防止发送方发出的数据超出接收方的接收能力，拥塞控制（congestion control）防止过多的数据注入网络而造成交换机或链路超载。因此，流量控制是一个端到端的问题，而拥塞控制是主机如何与网络交互的问题。

5.2.1 端到端问题

TCP 的核心是滑动窗口算法。虽然这与 2.5.2 节介绍的基本算法相同，但因为 TCP 是在整个因特网上而不是在一个点到点链路上运行，所以它们存在着很多重要的差别。本节指出这些差别，并解释这些差别怎样使 TCP 复杂化。下面几节将描述 TCP 如何处理这些复杂情况。

首先，尽管 2.5.2 节描述的滑动窗口算法运行在总是连接两台计算机的一条物理链路上，但 TCP 仍然支持运行在因特网中任意两台计算机上的进程之间的逻辑连接。这就是说，TCP 需要有明确的连接建立阶段，使连接的双方同意相互交换数据。这个不同点类似于需要拨号到对方，但是没有专用电话线路。TCP 也有一个明确的断开连接阶段。在连接建立阶段发生的事件之一，是双方建立某种共享状态使滑动窗口算法开始运行。连接断开阶段是必要的，因为只有这样双方主机才知道可以释放这种状态。

其次，尽管总是连接两台相同计算机的一条物理链路具有固定的往返时间（RTT），但 TCP 连接很可能具有差异很大的往返时延。例如，在旧金山的一台主机和在波士顿的一台主机之间有一个 TCP 连接，它们相隔数千公里，RTT 的值可能是 100ms，而在同一个房间的两台主机之间也可能有一个 TCP 连接，它们只相距几米，RTT 的值可能只有 1ms。因此，同样的 TCP 协议必须支持这两种连接。更糟糕的是，旧金山和波士顿两地主机之间的 TCP 连接可能在凌晨 3 点时的 RTT 值是 100ms，而在下午 3 点时的 RTT 值变成 500ms。甚至一个 TCP 连接在持续几分钟后，RTT 值就可能发生变化。这对滑动窗口算法而言，意味着触发重传的超时机制必须具有适应性（当然，点到点链路的超时值必须是可以设置的参数，但不必为了某对节点而调整上述定时器）。

第三个差别是分组通过因特网时可能重排序，这在点到点的链路上是不可能的，因为在链路一端先发送的分组一定先到达另一端。分组的轻度失序不会引起问题，因为滑动窗口算法能用序号将分组正确地重新排序。真正的问题是失序的分组多长时间能到达，换句话说，分组多晚才到达目的地。最坏的情况下，分组在因特网中被延迟，直到 IP 的生存期（TTL）字段过期，此时分组被丢弃（因此不存在分组迟到的危险）。已知 IP 在分组的 TTL 过期后就会丢弃分组，TCP 假设每个分组有一个最大的生存期。这是一种设计选择，称为*最大报文段生存期*（Maximum Segment Lifetime，MSL），当前的推荐值为 120s。注意，IP 并不直接强制使用这个 120s 的值，它只是 TCP 对一个分组可能在因特网上生存多久所做的保守估计。这样做的含义很明显，就是 TCP 不得不为很早以前发出的分组突然出现在接收方而做准备，因为这种分组可能会搅乱滑动窗口算法。

第四，连接到点到点链路的计算机通常都支持这种链路。例如，如果一个链路的延迟带宽积为8KB，就意味着窗口的大小在给定的时间内允许最大8KB的数据不被确认，这样可以认为链路任一端的计算机能缓存至多8KB数据。不这样设计系统是愚蠢的。另一方面，几乎任何类型的计算机都能连到因特网上，这使得用于TCP连接的资源数量变化很大，尤其是考虑到任何一台主机都可能同时支持几百个TCP连接。这意味着TCP必须包含一种机制，使连接的每一端用它“了解”另一端有什么资源（比如多少缓冲区空间）用于连接。这就是流量控制问题。

第五，因为一个直连链路的发送方不能以超出链路带宽所允许的速率发送数据，而且只有一台主机向链路注入数据，所以它不可能不知道链路拥塞。换句话说，链路的负载情况是以发送方的分组队列形式显现的。相比之下，TCP连接的发送方并不知道经过什么链路传送到目的地。例如，发送方计算机可能直接连到相对较快的以太网，它能以100Mbps的速率发送数据，但是在网络中的某个地方，必须通过一段1.5Mbps的T1链路。而且更糟糕的是，从很多数据源产生的分组可能都要通过这段低速网络链路。这就会导致网络拥塞问题。我们在第6章讨论这个主题。

下面通过对比用于提供可靠/有序传输服务的TCP方法与X.25网络使用的方法来结束端到端问题的讨论。在TCP中，下层的IP网络被认为是不可靠的，而且会使传递消息错序，TCP在端到端的基础上利用滑动窗口算法提供可靠/有序的传送。相比之下，X.25网络在跳到跳的基础上在网络内部使用滑动窗口协议。对这种方法的假设是，如果一条消息在沿源主机到目的主机路径上的每对节点之间都能可靠而有序地传输，那么端到端服务也能保证可靠/有序的传输。

后面这种方法的问题在于，一系列跳到跳的保证不一定能叠加为端到端的保证。首先，如果一个异构的链路（例如以太网）加在路径的末端，那么无法保证这一跳能维持与其他跳同样的服务。其次，滑动窗口协议只保证一个消息从节点A到节点B正确传递，从节点B到节点C也能正确传递，但它不能保证在节点B本身不出错。例如，我们已经知道，网络节点在把一个消息从输入缓冲区传到输出缓冲区时有可能会出现错误，也知道它们有时改变消息的顺序。正是由于这些小窗口的脆弱性，所以仍然需要提供真正的端到端检测以保证可靠/有序的服务，即使系统的低层已实现了这种功能。

结论 本节讨论的目的在于阐述系统设计中最重要的原则之一，即端到端理论。简而言之，端到端理论说明一种功能（在我们的例子中是提供可靠/有序的传递）不应该在系统的较低层提供，除非能在低层完全正确地实现。因此，这条原则有利于TCP/IP方法。但是该原则并不是绝对的，有时为了性能优化的需要也允许在较低层提供一些不完全的功能。这就是为什么在跳到跳基础上进行差错检测（如CRC）的原因，检测并重传经过一跳的单个损坏的分组要优于端到端重传整个文件。

5.2.2 报文段格式

TCP是面向字节的协议，这就是说发送方向一个TCP连接写入字节，接收方从这个TCP连接读出字节。虽然用“字节流”描述TCP提供给应用进程的服务，但是TCP本身并不在因特网上传送单个字节。实际上，在源主机上的TCP收集发送进程交付的字节，存到缓冲区中，积累到足够的数量，将其一起放入一个大小适宜的分组，再发送给目的主

机上的对等实体。目的主机上的 TCP 把这个分组的内容存入接收缓冲区，接收进程在空闲时从这个缓冲区读出字节。图 5-3 是这种情况的图解，为简单起见，只显示了一个方向的数据流。但通常情况下 TCP 连接支持字节流双向流动。

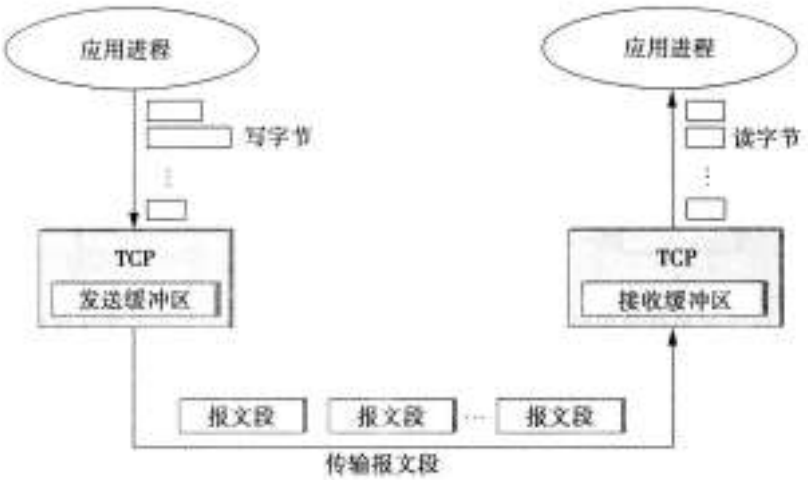


图 5-3 TCP 字节流

图 5-3 中，由于 TCP 对等实体之间交换的每个分组都携带一段字节流，所以将这些分组称为报文段（segment）。每个 TCP 报文段包含如图 5-4 所示的首部。图中绝大多数字段的相关内容将在本节说明，现在简单地介绍一下。

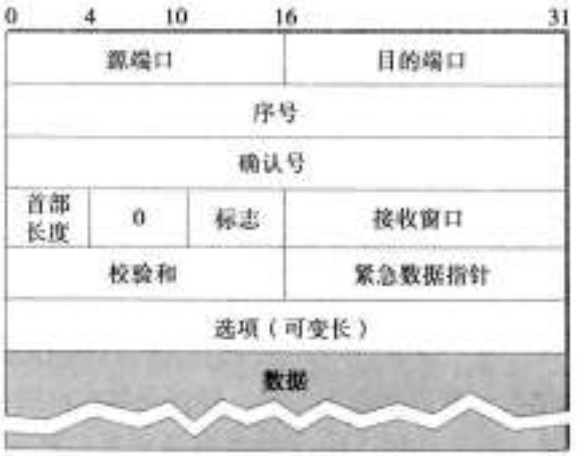


图 5-4 TCP 首部格式

与 UDP 首部一样，SrcPort 和 DstPort 字段分别表示源端口和目的端口。这两个字段加上源 IP 地址和目的 IP 地址，组合成每个 TCP 连接的唯一标识。也就是说，TCP 的多路分解密钥由四元组给出：

(SrcPort, SrcIPAddr, DstPort, DstIPAddr)

注意，因为 TCP 连接有始有终，所以有可能在某一对端口间建立了一个连接，并用它发送和接收数据，然后关闭。接着在一段时间后，第二个连接又使用同一对端口。有时把这种情况称为相同连接的两个不同实例（incarnation）。

Acknowledgment（确认号）、SequenceNum（序号）和 AdvertisedWindow（接收窗

口）字段都在 TCP 的滑动窗口算法中使用。因为 TCP 是面向字节的协议，所以数据的每个字节都有序号，SequenceNum 字段包含报文段携带数据的第一个字节的序号。Acknowledgment 和 AdvertisedWindow 字段携带反方向数据的信息。为了简化讨论，我们忽略数据可以双向流动的事实，只关心有特定 SequenceNum 值的数据向一个方向流动，而 Acknowledgment 和 AdvertisedWindow 向相反方向流动，如图 5-5 所示。这三个字段将在 5.2.4 节中更详细地描述。



图 5-5 TCP 过程的简单描述（只考虑单向），数据流在一个方向而确认在相反方向

6 比特的 Flags（标志）字段用来在 TCP 对等实体间传递控制信息。可能的标志位有 SYN、FIN、RESET、PUSH、URG 和 ACK。SYN 和 FIN 标志分别在建立和终止 TCP 连接时使用。它们的用法在 5.2.3 节描述。ACK 标志在每次 Acknowledgment 字段有效时设置，意指接收方应对 Acknowledgment 字段加以注意。URG 标志意味着本报文段包含紧急数据。当这个标志被置位时，UrgPtr（紧急数据指针）字段指明本报文段的非紧急数据从什么地方开始。紧急数据包含在报文段段体的前部，直到 UrgPtr 值所指的字节数为止。PUSH 标志说明发送方调用了 push 操作，这向 TCP 的接收方表明它应该把这个事实通知给接收进程。我们将在 5.2.7 节对后两个特性进行更多的讨论。最后，RESET 标志说明接收方已经出现混乱，例如，因为它收到了并不希望收到的报文段，所以它想要终止连接。

最后，CheckSum（校验和）字段与 UDP 中的用法完全相同，它是通过计算整个 TCP 首部、TCP 数据以及由 IP 首部的源地址、目的地址和长度字段构成的伪首部得到的。在 IPv4 和 IPv6 中，TCP 都要求有校验和字段。而且，由于 TCP 首部的长度是可变的（在必选项之后紧跟可选项），所以在其首部中包含一个 HdrLen（首部长度）字段，该字段以 32 位字为单位给出首部的长度。该字段也称为 Offset（偏移量）字段，因为可以用它衡量从分组的开始位置到数据开始位置的偏移量。

5.2.3 连接建立与终止

一个 TCP 连接从客户端（呼叫方）向服务器（被呼叫方）执行一个主动打开操作开始。假设服务器事先已经执行了被动打开操作，那么双方就交换建立连接的消息（回顾第 1 章，想要建立连接的一方执行主动打开操作，而接受连接的一方执行被动打开操作）。只有在连接建立阶段完成以后，双方才开始发送数据。同样，当其中一方发送完数据，就会关闭一个方向的连接，这就使 TCP 开始一轮终止连接的消息。注意，尽管连接的建立是一个非对称的活动（一方执行被动打开而另一方执行主动打开），但是连接的断开则是对称的活动（每一方必须独立地关闭连接）[⊖]。因此，有可能一方已经完成了关闭连接，意味着它不再发送

⊖ 更准确地说，连接建立实际上是对称的，每一方都试图在同一时间打开连接，但通常情况下，一方是主动打开，而另一方则是被动打开。

数据，但是另一方却仍保持双向连接的另一半为打开状态并且继续发送数据。

1. 三次握手

TCP 使用的建立和终止连接的算法称为三次握手（three-way handshake）。我们首先描述基本算法，然后说明 TCP 如何使用它。三次握手是指客户端和服务器之间要交换三次消息，如图 5-6 中的时间线所示。

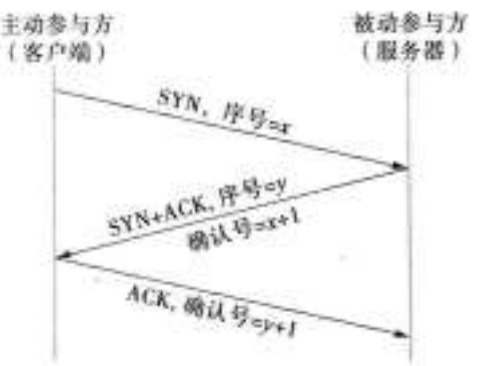


图 5-6 三次握手算法的时间线

算法的思想是双方需要商定一些参数，在打开一个 TCP 连接的时候，参数就是双方打算为各自的字节流使用的开始序号。通常，参数可以是每一方希望另一方了解的任何情况。首先，客户端（主动参与方）发送一个报文段给服务器（被动参与方），声明它将使用的初始序号（Falgs=SYN，SequenceNum=x）。服务器用一个报文段响应，确认客户端的序号（Flags=ACK，Ack=$x+1$），同时声明自己的初始序号（Flags=SYN，SequenceNum=y）。也就是说，第二个报文段的 Flags 字段的 SYN 和 ACK 位被置位。最后，客户端用第三个报文段响应，确认服务器的序号（Flags=ACK，Ack=$y+1$）。每一端的确认序号比发送来的序号大 1 的原因是 Acknowledgment 字段实际指出“希望接收的下一个序号”，从而隐含地确认前面所有序号。前两个报文段都使用计时器，虽然在图中的时间线上没有显示，而且如果没收到所希望的应答，就会重传报文段。

你也许会问，为什么在连接的建立阶段客户端和服务器必须相互交换初始序号？如果双方只从已知的序号（比如 0）开始会比较简单。实际上，TCP 规范要求连接的每一方随机地选择一个初始序号。这样做的原因是防止同一连接的两个实例过快地重复使用同一个序号，也就是说，仍旧有可能出现以前的连接实例的一个数据段干扰后来的连接实例的情况。

2. 状态转换图

TCP 非常复杂，以至于在它的规范中包含了一个状态转换图，如图 5-7 所示。这个图只显示打开一个连接时的状态转换（ESTABLISHED 的上面部分）和关闭一个连接时的状态转换（ESTABLISHED 的下面部分）。当连接打开后执行的操作（即滑动窗口算法的操作）隐含在 ESTABLISHED 状态中。

TCP 的状态转换图相当容易理解。每个矩形框代表一个状态，TCP 连接的每一端都能在其中找到自己的位置。所有连接开始于 CLOSED 状态。随着连接的进行，连接沿弧线从一个状态转移到另一个状态。每个弧线用事件/操作（event/action）的形式标记。这样，如果一个连接处于 LISTEN 状态且收到一个 SYN 报文段（带有 SYN 标志置位的报文段），那么连接就转换到 SYN_RCVD 状态，并且执行用 ACK+SYN 报文段应答的操作。

注意有两类事件会触发状态转换：来自对等实体的一个报文段（例如，从 LISTEN 到 SYN_RCVD 弧线上的事件），或本地应用进程调用一个 TCP 操作（例如，从 CLOSED 到 SYN_SENT 弧线上的主动打开（active open）事件）。换句话说，TCP 的状态转换图有效地定义了其对等实体之间的接口和服务接口的语义（semantic），如 1.3.1 节中的定义。这两种接口的语法（syntax）分别由报文段的格式（见图 5-4）和一些应用程序接口（1.4.1 节中有一个例子）给出。

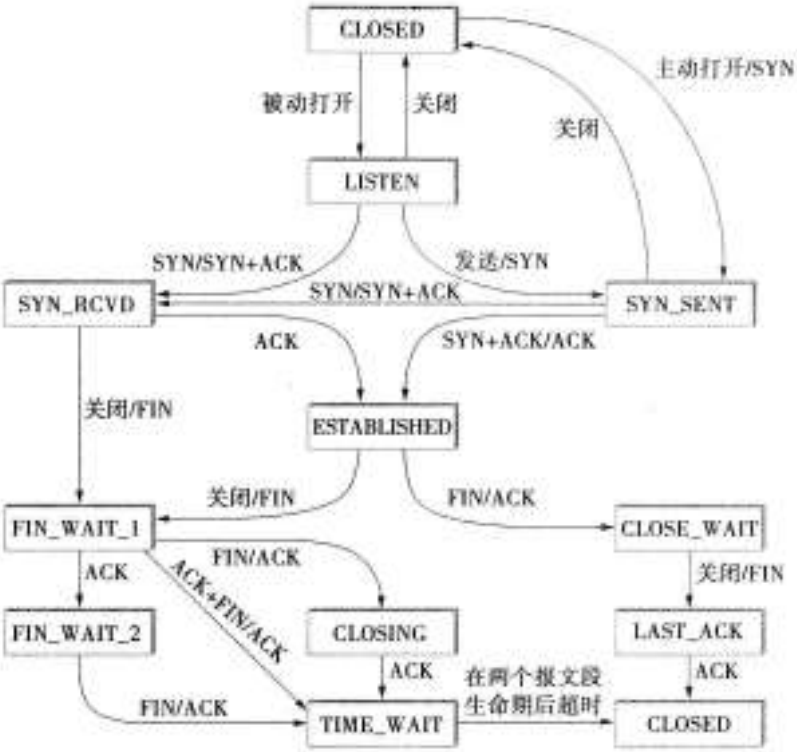


图 5-7 TCP 状态转换图

现在跟踪图 5-7 中发生的典型转换。注意在连接的每一端，TCP 会执行不同的状态转换。当打开一个连接时，服务器先执行一个被动的 TCP 打开操作，这使 TCP 转移到 LISTEN 状态。在一段时间后，客户端执行主动打开操作，向服务器发送一个 SYN 报文段并转移到 SYN _ SENT 状态。当 SYN 报文段到达服务器时，它就会转移到 SYN _ RCVD 状态并用 SYN＋ACK 报文段响应。这个报文段到达客户端后，会使客户端转移到 ESTABLISHED 状态并向服务器发回一个 ACK 报文段。当这个 ACK 报文段到达后，服务器最后转移到 ESTABLISHED 状态。到此为止，我们跟踪了整个三次握手过程。

关于状态转换图中的连接建立阶段，有三种情况需要注意。第一，如果客户端到服务器的 ACK 报文段（相应于三次握手的第三次）丢失，连接仍能正常工作。这是因为客户端已经处于 ESTABLISHED 状态，所以本地应用进程可以开始向另一方发送数据。每个报文段都有 ACK 标志置位，而且在 Acknowledgment 字段中包含正确的数值，所以当第一个报文段到达服务器时，它就会转移到 ESTABLISHED 状态。这实际上是 TCP 的重点之一，即每个报文段报告发送方希望看到的下一个序号，即使这个序号与以前的一个或多个报文段包含的序号重复。

关于状态转换图需要注意的第二种情况是，只要本地进程调用一个 TCP 的发送（send）操作，LISTEN 状态就会发生一个有趣的状态转换。也就是说，一个被动参与方有可能识别出连接的双方（即它自己和想与它连接的远端参与方），然后改变为主动建立连接。据我们所知，还没有应用进程真正利用 TCP 的这个特性。

关于转换图要注意的最后一种情况是有一些弧线在图中没有给出。特别是，一方向另一方发送报文段的同时会调用一个超时机制，如果没有出现期望的响应，最终会导致重发这个报文段。这些重传没有在状态转换图中给出。如果在几次重发后仍没有得到期望的响

应，TCP 就会放弃重传并回到 CLOSED 状态。

现在我们来考虑终止一个连接的过程，这里需要注意的一件重要的事情是，连接双方的应用进程必须独立地关闭自己一方的连接。如果仅一方关闭连接，就是说，它不再发送数据，但它仍能接收另一方发来的数据。这就使状态转换图复杂化，因为必须考虑到双方可能同时调用关闭（close）操作，也可能其中一个先调用关闭操作，间隔一段时间后另一个再调用关闭操作。这样，连接的任何一方从 ESTABLISHED 状态到 CLOSED 状态有三种转换组合：

- 一方先关闭：ESTABLISHED→FIN_WAIT_1→FIN_WAIT_2→TIME_WAIT→CLOSED。
- 另一方先关闭：ESTABLISHED→CLOSE_WAIT→LAST_ACK→CLOSED。
- 双方同时关闭：ESTABLISHED→FIN_WAIT_1→CLOSING→TIME_WAIT→CLOSED。

事实上，还存在第四种极少出现的到达 CLOSED 状态的转换顺序，它沿着从 FIN_WAIT_1 到 TIME_WAIT 的弧到达。我们把它作为习题，请你找出导致第四种可能性的情况组合。

关于断开连接需要考虑的主要事情是，直到等待一个 IP 数据报在因特网上可能存活的最大时长的 2 倍时间（即 120s）后，连接才能从 TIME_WAIT 状态转移到 CLOSED 状态。这是因为当连接的本地一方已经发出一个 ACK 报文段响应对方的 FIN 报文段时，它并不知道这个 ACK 报文段是否成功传递。因此，另一方可能又重传一个 FIN 报文段，而这第二个 FIN 报文段可能在网络中被延迟。如果允许连接直接转移到 CLOSED 状态，那么可能会有另一对应用进程打开同一个连接（即使用同一对端口号），而前面连接实例中被延迟的 FIN 报文段这时就会立即使后来的连接实例终止。

5.2.4 滑动窗口再讨论

现在讨论 TCP 滑动窗口算法的变体，它服务于这样几个目的：①保证数据的可靠传递；②确保数据的有序传递；③增强发送方和接收方之间的流量控制。在这三种功能的前两种情况下，TCP 对滑动窗口算法的使用与我们在 2.5.2 节看到的相同。TCP 与以前算法的不同之处在于它增加了流量控制功能。特别是 TCP 并不使用固定尺寸的滑动窗口，而是由接收方向发送方通知（advertise）它的窗口尺寸。这是通过使用 TCP 首部的 AdvertisedWindow 字段完成的。此后，发送方在任意给定时刻未被确认的字节数都不能超过 AdvertisedWindow 的值。接收方根据分配给连接的用于缓存数据的内存数量，为 AdvertisedWindow 选择一个合适的值。其思想是不使发送方发送的数据超过接收方缓冲区的限度。下面更深入地讨论这个问题。

1. 可靠和有序的传输

图 5-8 给出了 TCP 发送方和接收方如何相互作用以实现可靠和有序的传输。发送方的 TCP 维护一个发送缓冲区，用来存储那些已被发出但未被确认的数据和已被发送应用程序写入但尚未发出的数据。在接收方，TCP 维护一个接收缓冲区，用于存放到达的错序数据和按正确顺序到达（即字节流中没有丢失前面的字节）但应用进程无暇读出的数据。

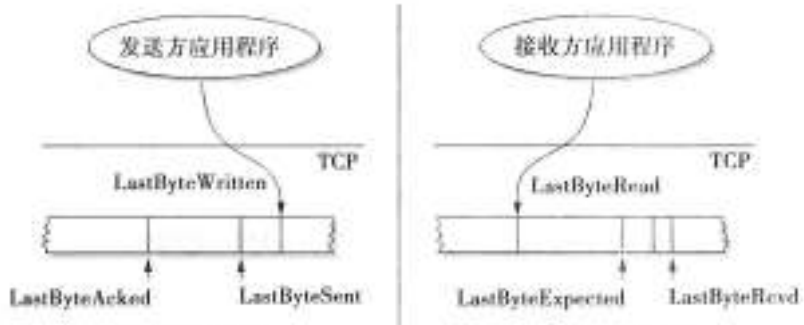


图 5-8 TCP 发送缓冲区和接收缓冲区的关系

为了简化讨论，先忽略以下事实：缓冲区和序号都是有限数量的，因此最终会回绕。而且我们也不区分数据中某个特定字节在缓冲区存储位置的指针与字节的序号。

首先看发送方，它维护发送缓冲区的三个指针：LastByteAcked、LastByteSent 和 LastbyteWritten。显然

$$\text{LastByteAcked} \leqslant \text{LastByteSent}$$

因为接收方不可能确认未发出的字节。并且

$$\text{LastByteSent} \leqslant \text{LastByteWritten}$$

因为 TCP 不能发送应用程序没写入的字节。还要注意，不必在缓冲区中保留 LastbyteAcked 左边的字节，因为它们已经被确认了；也没有必要缓存 LastByteWritten 右边的字节，因为它们还没产生。

在接收方维护着一组类似的指针（序号）：LastByteRead、NextByteExpected 和 LastByteRcvd，然而因为传输的错序问题，不等式不那么直观。第一个关系式

$$\text{LastByteRead} < \text{NextByteExpected}$$

成立，因为只有一个字节及其前面的所有字节都被接收后，它才能被应用程序读出。为了满足这一准则，NextByteExpected 指向紧接最后一个字节的那个字节。其次，

$$\text{NextByteExpected} \leqslant \text{LastByteRcvd} + 1$$

成立，因为如果数据按正确的顺序到达，NextByteExpected 指向 LastByteRcvd 之后的那个字节，但若数据到达是错序的，那么 NextByteExpected 将指向数据中的第一个间隙的开始处，如图 5-8 所示。注意，LastByteRead 左边的字节不必再保存在缓冲区中，因为它们已经被本地应用程序读取，而 LastByteRcvd 右边的字节也不必缓存，因为它们还没到达。

2. 流量控制

以上讨论的绝大部分与 2.5.2 节类似，唯一的区别是，发送应用进程和接收应用进程分别填充和清除它们的本地缓冲区，下面会详述。之前的讨论忽略了这样的事实，上游节点到来的数据填入发送缓冲区，向下游节点发送数据后清除接收缓冲区。

你必须确定在继续讨论之前对这一问题已经理解，因为现在要讨论这两个算法更大的不同之处。接下来，再次说明收发双方的缓冲区具有有限的大小，分别用 MaxSendBuffer 和 MaxRcvBuffer 表示，但是我们并不关心它们具体是如何实现的。换句话说，我们只对被缓存的字节数感兴趣，而不管这些字节实际上存储在什么地方。

回想在滑动窗口协议中，窗口的大小决定可以被发出而不必等待接收方确认的数据

量。这样，接收方通过给发送方通知一个不大于它所能存放数据量的窗口，就能控制发送方的发送速率。可以看出，在接收方的 TCP 必须保持

$$\text{LastByteRcvd} - \text{LastByteRead} \leqslant \text{MaxRcvBuffer}$$

才能避免缓冲区溢出。因此它通知的窗口大小为

$$\text{AdvertisedWindow} = \text{MaxRcvBuffer} - ((\text{NextByteExpected} - 1) - \text{LastByteRead})$$

这个数值代表其缓冲区中剩余的可用空间数量。当数据到来时，只要它前面的字节已经到达，接收方就会对它确认。另外，LastByteRcvd 向右移动（增加），这也意味着通知窗口可能缩小。通知窗口是否缩小依赖于本地应用进程处理数据的快慢。如果本地进程读取数据的速率与数据到达的速率相同（使 LastByteRead 和 LastByteRcvd 以相同的速率增加），那么通知窗口就保持打开状态（即 AdvertisedWindow=MaxRcvBuffer）。然而，如果接收进程的速率可能因为它对读到的每个字节要进行费时的操作而落后，那么随着每个报文段的到来，通知窗口就会变得很小，直到最终变成 0。

发送方的 TCP 必须遵守从接收方得到的通知窗口。这就意味着在任何时刻，它必须确保

$$\text{LastByteSent} - \text{LastByteAcked} \leqslant \text{AdvertisedWindow}$$

换个方式说，发送方计算一个有效窗口，用来限制它可以发送多少数据：

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

显然，只有 EffectiveWindow 大于 0，发送方才可以发送更多的数据。因此，有可能一个报文段到达而确认 x 字节，从而允许发送方把 LastByteAcked 增加 x，但是由于接收进程没有读取任何数据，所以通知窗口比先前小了 x。在这种情况下，发送方可以释放缓冲区空间，但不能再发送任何数据。

在整个过程中，发送方还必须始终保证本地应用进程不使发送缓冲区溢出，也就是

$$\text{LastByteWritten} - \text{LastByteAcked} \leqslant \text{MaxSendBuffer}$$

如果发送进程试图向 TCP 写入 y 字节，但是

$$(\text{LastByteWritten} - \text{LastByteAcked}) + y > \text{MaxSendBuffer}$$

那么，TCP 就会阻塞发送进程，不让它再产生数据。

现在应该可以理解一个慢速接收进程是如何使一个快速发送进程最终停止下来的。首先，接收缓冲区填满，这意味着通知窗口缩小到 0。通知窗口为 0 意味着发送方不能发送任何数据，即使它以前发送的数据早已被成功确认。最后，不能传输任何数据意味着发送缓冲区填满，最终使 TCP 将发送进程阻塞。接收进程重新开始读取数据，接收方 TCP 就能够打开它的窗口，允许发送方 TCP 把数据从它的缓冲区传送出去。当这个数据最终被确认，LastByteAcked 随着增加时，就把保存这个被确认数据的缓冲区空间释放，发送进程解除阻塞并允许继续执行。

现在只剩下一个细节需要解决，即发送方如何知道通知的窗口不再是 0？如上所述，TCP 总是发送一个报文段对接收到的报文段做出响应，这个响应包含 Ackonwledge 和 AdvertisedWindow 字段的最新值，即使这两个值自上次发送以来没有改变。问题正在于此。一旦已经通知接收方窗口变为 0，就不允许发送方发送任何数据，这就意味着它没有办法发现在将来的某个时刻通知窗口不再是 0。接收方的 TCP 不会自发地发送不包含数据的报文段，它只在响应到达的报文段时发送它们。

TCP 按下述方式处理这种情况。当对方通知的窗口变为 0 时，发送方仍坚持不停地发送一个只有 1 字节的报文段。它知道这个报文段有可能不被接收，但它还是要尝试，因为

每个这样的1字节报文段会触发包含当前通知窗口的响应。最终，某个1字节的探测报文段会触发一个报告非0通知窗口的响应。

结论 注意，发送方周期性地发送探测报文段的原因是：TCP被设计成使接收方尽可能地简单，即它只响应从发送方来的报文段，而它自己从不发起任何活动。这是公认的（尽管并不是通用的）协议设计规则的一个例子，因为没有较好的名称，我们称其为聪明的发送方/笨拙的接收方（smart sender/dumb receiver）规则。在2.5.2节讨论NAK的用法时，我们见过这条规则的另一个例子。

3. 防止回绕

本节和下节考虑SequenceNum字段和AdvertisedWindow字段的大小以及它们对TCP正确性和性能的影响。TCP的SequenceNum字段长32位，AdvertisedWindow字段长16位，也就是说TCP无疑已经满足滑动窗口算法的要求，即序号空间是窗口空间的两倍：$2^{32} \gg 2 \times 2^{16}$。然而，这个要求对这两个字段并不重要。下面依次考虑每个字段。

32位序号空间的相关问题是，某个连接使用的序号可能会回绕，即具有序号x的一个字节在某个时刻被发送出，一段时间后，第二个具有序号x的字节也有可能被发送出。再次假设一个分组在因特网上的生存期不超过MSL的建议值。这样，当前的任务是确保序号在这120s的期限内不会回绕。这种情况是否会发生依赖于数据在因特网上传输的速度，也就是说，32位的序号空间多快被用完（这里的讨论假设尽可能快地消耗序号空间，当然只要让管道在传输进行中保持满载就能如此）。表5-1显示了具有不同带宽的网络使序号回绕的时间。

可见，32位的序号空间对当今的网络是足够的，但是对于当前存在于因特网主干网上的OC-192链路，在单个TCP连接以622Mbps或更高速率（现多数服务器都具备千兆以太网口）运行时这个长度是不够的。幸运的是，IETF已经完成了对TCP的扩展工作，通过有效地扩展序号空间来防止序号回绕。这个工作以及相关的扩展在5.2.8节描述。

4. 保持管道满载

与16位AdvertisedWindow字段相关的问题是，它必须足够大以使得发送方能够保持管道满载。显然，接收方可以不把窗口开放到AdvertisedWindow字段所允许的最大值。我们只关心接收方是否有足够的缓冲区空间可以处理AdvertisedWindow所允许的最大数据量的情况。

在这种情况下，网络带宽和延迟带宽积共同决定AdvertisedWindow字段应有的大小。窗口必须开放得足够大，使得数量为延迟×带宽的全部数据能被传输。假设有100ms的RTT（穿越美国大陆的连接的典型数字），表5-2给出了几种网络技术的延迟带宽积。

表5-1 32比特序号空间的回绕时间

带 宽	回绕时间
T1（1.5Mbps）	6.4小时
以太网（10Mbps）	57分钟
T3（45Mbps）	13分钟
快速以太网（100Mbps）	6分钟
OC-3（155Mbps）	4分钟
OC-12（622Mbps）	55秒
OC-48（2.5Gbps）	14秒

表5-2 100ms RTT所需的窗口大小

带 宽	延迟×带宽
T1（1.5Mbps）	18KB
以太网（10Mbps）	122KB
T3（45Mbps）	549KB
快速以太网（100Mbps）	1.2MB
OC-3（155Mbps）	1.8MB
OC-12（622Mbps）	7.4MB
OC-48（2.5Gbps）	29.6MB

可以看出，TCP 的 AdvertisedWindow 字段比 SequenceNum 字段处于更糟糕的境况，它的大小甚至不足以处理横穿美国大陆的 T3 连接，因为 16 比特的字段只允许 64KB 的通知窗口。TCP 扩展（参见 5.2.8 节）提供了一种有效增加通知窗口大小的机制。

5.2.5 触发传输

接下来考虑一个微妙的问题：TCP 怎样决定传输一个报文段。如前所述，TCP 支持一种字节流抽象，即应用程序把字节写到流里，而由 TCP 决定字节数是否达到足以发送一个报文段的要求。支配这个决定的因素是什么呢？

如果忽略流量控制的可能性，即假设窗口是敞开的，就像一个连接刚开始时的情形，那么 TCP 有三种机制触发一个报文段的传输。第一种机制，TCP 维护一个变量，称为最大报文段长度（MSS），一旦 TCP 从发送进程收集到 MSS 字节，它就发送一个报文段。通常把 MSS 设置成 TCP 能发送而且不造成本地 IP 分段的最大报文段长度。也就是说，MSS 被设置成直接连接网络的最大传输单元（MTU）减去 TCP 和 IP 首部的大小。第二种触发 TCP 发送一个报文段的机制是，发送进程明确要求 TCP 发送一个报文段。特别是 TCP 支持 push 操作，发送进程调用这个操作能使 TCP 将缓冲区中所有未发送的字节发送出去。最后一种触发 TCP 发送一个报文段的机制是定时器激活，结果报文段中包含当前缓冲区中所有需要发送出去的字节。然而，我们很快就会看到，这个“定时器”并不完全如所期望的那样。

1. 傻瓜窗口症状

当然，流量控制不能被忽略，它对控制发送方起着显而易见的作用。如果发送方有 MSS 字节数据要传输而窗口至少打开了那么大，那么发送方就传输一个满报文段。但是，如果发送方正在积累要发送的字节，而窗口是关闭的。现在假设有一个 ACK 到达，使窗口开到足以让发送方传输，比如说 MSS/2 字节。那么发送方是发出一个半满的报文段还是等待窗口开大到 MSS？最初的 TCP 规范并未对这一点进行说明，但早期的 TCP 实现决定进行发送，并传输一个半满的报文段。毕竟，无法得知多长时间以后窗口才会进一步开放。

事实证明，一味地利用任何可用窗口的策略会导致现在称作傻瓜窗口症状（silly window syndrome）的情形。图 5-9 有助于想象发生的情况。如果把 TCP 流看作是一个传送带，把“满载”的容器（报文段）向一个方向移动，空的容器（ACK 段）向相反方向移动，那么 MSS 大小的报文段就对应大容器而 1 字节的报文段就对应很小的容器。只要发送方发送 MSS 大小的报文并且接收方一次接收一个 MSS 大小的数据，那么一切都没问题（见图 5-9a）。但是，如果接收方必须减少窗口导致发送方在一段时间内不能完整地发送一个 MSS 大小的数据将会发生什么？如果只要有小于 MSS 的空容器到达，发送方就一味地填充，接收方将会确认它，因此任何引入系统的小容器就可能不定期地留在系统中。也就是说，在每一端小容器会立刻被填充或被清除，而从不会联合毗邻的容器来创建一个更大的容器，参见图 5-9b。当早期的 TCP 实现定期地发觉自己用很小的报文段充满网络时，这种“傻瓜窗口症状”就出现了。

注意，只有在发送方传输小报文段或接收方打开小窗口时才会出现傻瓜窗口症状。如果这两种情况都未发生，那么小容器就永远不会被引入数据流中。禁止发送小报文段是不可能的。举例来说，应用程序可以在只发出一个字节后就执行 push 操作。但是防止接收



a) 如果发送方发送MSS大小的报文段，同时接收方接收MSS的报文段，系统运转顺利



b) 如果发送方发送小于MSS大小的报文段或者接收方接收小于MSS的报文段，那么这个小“容器”就会进入系统并一直存在

图 5-9 傻瓜窗口症状

方引入小的容器（即打开小窗口）是可能的。其规则是接收方在通知一个大小为零的窗口后，必须等到有 MSS 大小的空间才能再通知打开窗口。

由于不能排除小容器被引入数据流的可能性，所以需要采用把它们合并起来的机制。接收方可以通过延迟发送 ACK 做到这一点（发送一个合并的 ACK 而不是几个小 ACK），然而这只解决了部分问题，因为接收方无法知道在等待下一个报文段到达或应用程序读出更多的数据（因此而打开窗口）时，等待多久才是安全的。最终解决问题的重任落到发送方，这又回到了最初的问题：发送方什么时候决定传输一个报文段？

2. Nagle 算法

回到 TCP 的发送方，如果有数据要发送但是打开的窗口小于 MSS，那么在发出可用数据之前可能要等待一段时间，但问题是等待多久？如果等待太久，就会有损于像 Telnet 这样的交互式应用程序。如果等待的时间不够长，又会面临发出很多小分组而陷入傻瓜窗口症状的风险。答案是引入一个定时器，时间到了就传输数据。

虽然可以引入一个基于时钟的定时器，比如每 100ms 激活一次，但是 Nagle 引入了一种完美的自计时（self-clocking）方案。其思想是只要 TCP 发出了数据，发送方终究会收到一个 ACK。可以把这个 ACK 看成激活的定时器，触发传输更多的数据。Nagle 算法提供了一条决定何时传输数据的简单统一的规则：

```
当应用产生要发送的数据时
    if 可用数据和窗口≥MSS
      发送满载的报文段
    else
      if 有正在传输的报文段
        缓存新数据直到 ACK 到达
      else
        发送所有新数据
```

换句话说，如果窗口大小允许，那么就可以发出一个满载的报文段；如果当前没有处于传输中的报文段，也可以立即发出一个小报文段；但是如果有传输的报文段，发送

方就必须等待有 ACK 到达才可传输下一个报文段。这样，像 Telnet 这类每次写一个字节的交互式应用程序将能以每 RTT 一个报文段的速率发送数据。有些报文段只有一个字节，而其他报文段将包含用户在一个 RTT 的时间内能输入的所有字节。因为有些应用程序不能容忍每次 TCP 连接写操作的延迟，所以套接字接口允许应用程序通过设置 TCP_NODELAY 选项来关闭 Nagle 算法。设置这个选项意味着数据被尽可能快地传输。

5.2.6 自适应重传

由于 TCP 保证可靠的数据传送，所以如果在一定的时限内没有收到 ACK，那么它就会重传每个报文段。TCP 把这个超时设置成它期望的连接两端的 RTT 的函数。不幸的是，即使给出因特网上任意一对主机之间 RTT 可能的范围，也给出同一对主机之间 RTT 随时间的变化，选择一个合适的超时值也并不容易。为了处理这个问题，TCP 使用一种自适应重传机制。下面描述这种机制以及它是怎样随着因特网业界使用 TCP 获得的经验而发展起来的。

1. 原始算法

我们从计算一对主机之间超时值的简单算法开始。这是最初在 TCP 规范中描述的算法（下面用规范中的术语描述），但是它适用于任何端到端协议。

算法的思想是，维持一个 RTT 的平均运行值，并把超时值作为 RTT 的一个函数计算。特别地，每次 TCP 发送一个数据报文段，它便记录发送时刻。当那个报文段的 ACK 到达时，TCP 再次读取时间，然后把这两次时间的差作为 SampleRTT。接着 TCP 利用以前的估计值和这个新的样本值计算出 EstimatedRTT 作为加权平均值。即

$$\text{EstimatedRTT} = \alpha \times \text{EstimatedRTT} + (1-\alpha) \times \text{SampleRTT}$$

选择参数 α 是为了平滑 EstimatedRTT。较小的 α 值可以跟踪 RTT 的变化，但是它可能受瞬时波动的影响过于严重。另一方面，一个大的 α 值更稳定但不能迅速适应真正的变化。原始 TCP 规范建议 α 值设置在 0.8～0.9 之间。TCP 用 EstimatedRTT 以较保守的方式计算超时值：

$$\text{TimeOut} = 2 \times \text{EstimatedRTT}$$

2. Karn/Partridge 算法

在因特网上应用几年以后，人们在这个简单算法中发现了一个明显的缺陷。问题是 ACK 实际上并不确认传送，而是确认数据的接收。换句话说，无论何时重传一个报文段，然后一个 ACK 到达发送方，它都不可能为测量样本 RTT 确定这个 ACK 是针对第一个报文段还是第二个重发的报文段。发送方有必要知道这个 ACK 是针对哪一次发送的，以便计算一个精确的 SampleRTT。如图 5-10 所示，如果错把针对第二个报文段的 ACK 当成针对第一个报文段的，SampleRTT 就会过大（见图 5-10a），相反，如果错把与第一个报文段关联的 ACK 当成第二个报文段的 ACK，那么 SampleRTT 就会过小（见图 5-10b）。

解决办法相当简单。当 TCP 重传一个报文段时，它停止计算 RTT 的样本值，即它只为仅发送一次的报文段测量 SampleRTT。这个算法就是著名的以其发明者命名的 Karn/Partridge 算法。他们提出的修正法还包括一个对 TCP 超时机制较小的改动。每次 TCP 重传时，它设置下次的超时值为上次的两倍，而并不以上次的 EstimatedRTT 为基础。也就

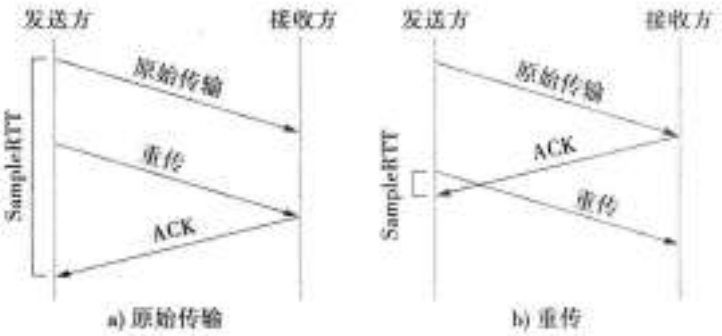


图 5-10 与原始传输和重传关联的 ACK

是说，Karn 和 Partridge 提出 TCP 使用指数回退算法，就像以太网中一样。使用指数回退算法的动机很简单：拥塞最可能导致报文段丢失，这表明 TCP 源主机对超时的反应不应该太主动。实际上，连接超时次数越多，源主机越会小心谨慎。在第 6 章中会看到这一思想在更复杂的机制中再次体现。

3. Jacobson/Karels 算法

Karn/Partridge 算法提出时，正逢因特网经受严重网络拥塞的时期。该方法用于解决一些拥塞问题，虽然有一定改进，但拥塞并未消除。几年以后，另外两位研究人员 Jacobson 和 Karels 提出对 TCP 进行进一步的改进来对付拥塞问题。这个建议的重要内容在第 6 章描述。这里只关注与决定何时超时并重传报文段有关的提议。

另外，必须清楚超时机制是怎样与拥塞相关的。如果超时太快，可能不必要重传报文段，那样只会增加网络的负载。在第 6 章会看到，需要准确超时值的另一个原因是超时被用来暗示发生了拥塞，它会触发拥塞控制机制。最后要注意，Jacobson/Karels 关于超时的计算并没有什么特别针对 TCP 的东西，它可以用于任何端到端协议。

原始计算的主要问题是没有考虑到 RTT 样本的变化。直觉上，如果样本变化小，则 EstimatedRTT 的值就更为可信，没必要把这个值乘以 2 来计算超时值。另一方面，样本变化很大说明超时值应该远不止是 EstimatedRTT 的两倍。

在新方法中，发送方与以前一样测量一个新的 SampleRTT，并把这个新样本按如下方法包含到超时计算中：

$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$

$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta \times \text{Difference})$$

$$\text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation})$$

其中，δ 是 0～1 之间的小数。也就是说，计算 RTT 的平均值以及该平均值的变化。

接着 TCP 把超时值作为 EstimatedRTT 和 Deviation 的函数计算：

$$\text{TimeOut} = \mu \times \text{EstimatedRTT} + \phi \times \text{Deviation}$$

其中，根据经验，μ 通常设为 1，ϕ 设为 4。这样，当变化小时，TimeOut 与 EstimatedRTT 接近，而大的变化会使 Deviation 项决定计算结果。

4. 实现

关于 TCP 中超时的实现，有两点需要注意。第一点，可以不用浮点算术实现 EstimatedRTT 和 Deviation 的计算。相反，将 δ 选作 $1/2^n$，整个计算以 2^n 的比例增加。这

使得能够用移位实现乘法和除法的整数运算，因此获得较高的性能。下面的代码段给出了相应的计算方法，其中 $n=3$（即 $\delta=1/8$）。注意 EstimatedRTT 和 Deviation 以按比例放大的形式存储，而代码开始处的 SampleRTT 和结尾处的 TimeOut 是真正的未放大的值。如果认为代码难以理解，那么可以代入一些真实数值，验证它的结果与上面的公式一致。

```
{
    SampleRTT -= (EstimatedRTT >> 3);
    EstimatedRTT += SampleRTT;
    if (SampleRTT < 0)
        SampleRTT = -SampleRTT;
    SampleRTT -= (Deviation >> 3);
    Deviation += SampleRTT;
    TimeOut = (EstimatedRTT >> 3) + (Deviation >> 1);
}
```

需要注意的第二点是，Jacobson 和 Karels 的算法实际上相当于读取当前时间的时钟。在典型的 UNIX 实现中，时钟粒度大到 500ms，显然比横穿美国的在 100～200ms 之间的平均 RTT 值大得多。更糟的是，在典型的 UNIX 实现中，TCP 的实现只在每个 500ms 时刻检查是否发生超时，而且只对每个 RTT 取一次往返时间样本。这两个因素的组合意味着一个报文段传输 1s 后发生超时。因此，TCP 的扩展还包含一个使 RTT 计算更准确的机制。

目前讨论过的所有重传算法都基于确认机制，通过超时来表明报文段可能已经丢失。然而，超时并没有告诉发送方在丢失报文段之后发送的报文段是否成功到达，因为 TCP 的确认是累积确认，它只确认所收到的没有间隔的最后一个报文段。在间隔之后接收的报文段导致了更大的窗口，如果能够对间隔之后正确接收的报文段进行确认，那么发送方就具有更高的智能性，能够了解到更实际的拥塞状态，从而对 RTT 有更准确的估计。5.2.8 节将要描述的 TCP 扩展机制能够做到这一点。

5.2.7 记录边界

由于 TCP 是面向字节流的协议，因此每次发送方写入的字节数未必与接收方读出的字节数相等。例如，应用程序向一个 TCP 连接写入 8 字节，接着写入 2 字节，然后又写入 20 字节；而在接收方，应用程序每次只读 5 字节，循环读 6 次。TCP 不在第 8 和第 9 个字节以及第 10 和第 11 个字节之间插入记录的边界。这与面向数据报的协议（如 UDP）相反，在面向数据报的协议中发送的消息与接收到的消息长度完全相同。

虽然 TCP 是面向字节流的协议，但它有两个不同的特性可被发送方用来在字节流中插入记录边界，因而通知接收方如何把字节流分成记录（例如，在许多数据库应用中，能标记出记录的边界是有用的）。这两个特性最初由于完全不同的原因被包含在 TCP 中，但它们一直未被使用，直到用于这个目的。

第一个机制是紧急数据特性，在 TCP 首部用 URG 标志和 UrgPtr 字段实现。最初，紧急数据机制被设计成允许发送应用程序向其对等实体发送带外（out-of-band）数据。用“带外”是为了与正常数据流中的数据相区别，如用命令中断正在进行的操作。在报文段中，这个带外数据用 UrgPtr 字段标识，一到达就立即传递给接收进程，即使这意味着在

传输具有更早序号的数据之前传输它。然而，这个特性一直没有得到应用，所以人们不再用它来表示“紧急”数据，而是渐渐地用它来表示“特殊”数据，比如一个记录标记。与push操作一样，这个用法发展起来是因为接收方的TCP必须通知应用程序有紧急数据到达。也就是说，紧急数据本身并不重要，重要的是发送进程能有效地向接收方发送信号。

第二个向字节流插入记录结束标记的机制是push操作。最初设计这种机制是用来使发送进程能告诉TCP，不管它收集了多少字节，应立刻把它们发送给对等实体。push可以用来实现记录分界，因为TCP规范说明，当应用程序要求push时，不管源端缓冲区有多少数据，TCP必须将它们发送出去。如果目的端TCP支持这个选项，那么无论何时接收到有PUSH标志的报文段，都要通知应用程序。因此，如果接收端支持这个选项（套接字接口不支持），就可以用push操作将TCP流分割成记录。

当然，应用程序总是不依靠TCP的任何帮助就能随意插入记录边界。例如，它可以发送一个字段指明随后的记录长度，或向数据流中插入它自己的记录边界标记。

5.2.8 TCP扩展

我们已在本节四个不同的地方提到，目前对TCP的扩展有助于缓和其面临的由于底层网络变得更快速而带来的一些问题。这些扩展被设计成对TCP协议的影响尽可能地小。特别是在实现时，它们作为可选项添加到TCP的首部。（虽然先前未加说明，但TCP首部有一个HdrLen字段的原因是其首部的长度是可变的，而TCP首部的可变长度部分包含已经加入的可选项。）把这些扩展功能添加为可选项而不是改变TCP基本首部的重要性在于，即使没有实现这些选项，主机之间仍然可以使用TCP进行通信。然而，对于实现这些可选的扩展功能的主机，就可以利用它们。在TCP连接的建立阶段，连接的双方可以对是否使用扩展功能达成一致。

第一个扩展功能有助于改进TCP的超时机制。TCP不使用粗粒度事件来测量RTT，而是在即将发送一个报文段时，读取实际的系统时钟，并把这个时间值（可以把它想象成一个32位的时标放到该报文段的首部。接收方在其确认中把这个时标返回给发送方，而发送方从当前时间中减去这个时标就可以测出RTT。实际上，这个时标选项为TCP提供了一个理想的场所来存储某个报文段是何时被传输的，它把时间存放在报文段中。注意，连接中的两端并不需要进行时钟同步，因为时标是在连接的同一端被写入和读取的。

第二个扩展解决TCP的32位SequenceNum字段在高速网络上回绕过快的问题。TCP并不定义一个新的64位的序号字段，而是使用刚描述的32位时标有效地扩展序号空间。换句话说，TCP根据一个64位的标识符决定接收或丢弃一个报文段，这个标识符由低32位的SequenceNum字段和高32位的时标构成。由于时标一直在增加，所以可以用它来区分有相同序号的两个不同的报文段。注意，以这种方式使用时标只是为了防止序号回绕现象，它并不作为序号的一部分而用于数据的排序和确认。

第三个扩展功能允许TCP通知更大的窗口，因此允许它填充可能由高速网络形成的更大的延迟×带宽管道。这个扩展功能有一个选项，它为通知窗口定义一个扩展因子（scaling factor）。也就是说，不把出现在AdvertisedWindow字段中的数值解释为发送方允许有多少字节不被确认，这个选项允许TCP连接的双方同意AdvertisedWindow字段计算更大的数据块（例如，发送方可以有多少个16字节数据单元不被确认）。换句话说，这个窗口扩展选项说明，在用AdvertisedWindow字段中的内容计算有效窗口之前，每一方

应将它左移多少位。

第四个扩展功能扩充了 TCP 的累积确认功能，方法是对收到的与先前已收到的报文段不连续的报文段进行可选确认，这称为可选确认（selective acknowledgment）或 SACK。使用 SACK 选项时，接收方继续确认报文段，确认字段的确认方式不变，但也使用首部的选项字段来确认接收到的其他数据块。这使得发送方能够根据可选确认来重传确实已经丢失的报文段。

如果没有 SACK，发送方只有两个合理的策略。悲观的策略使得发生超时后 TCP 不仅重传超时的报文段，而且还要重传该报文段之后可能并没有发生超时的报文段。该策略实际上做了最坏的假设，即所有报文段都丢失了。该策略的缺点是它可能不必要地重传了那些已经被成功接收的报文段。另一个策略是比较乐观的，它仅重传那些发生了超时的报文段，它认为只有一个报文段丢失了。这种乐观策略的缺点是运行比较慢，当拥塞严重时，可能会丢失连续的一系列报文段，但只有收到对先前报文段重传的确认时，才会发现报文段丢失。因此对每个报文段消耗了一个 RTT，直到重传了丢失报文段序列中的所有报文段。使用 SACK 选项，可以使发送方采用一种更好的策略：只重传那些能够填充可选确认报文段间隔的报文段。

顺便说一下，这些延伸并不是所有需求。在下一章讨论 TCP 如何处理拥塞时会看到更多的延展。互联网编号分配机构（IANA）跟踪所有被定义为 TCP（和许多其他互联网协议）的选项。大约有 30 个 TCP 的选项是在编写的时候定义的（尽管大部分是实验性的或过时的）。请参阅在本章末尾参考资料中 IANA 注册协议编号的链接。

5.2.9 性能

回忆在第 1 章介绍的评估网络性能的两个度量标准：时延和吞吐量。正如在那个讨论中提到的，这些度量值不仅受底层硬件的影响（例如传播延迟和链路的带宽），而且还受软件开销的影响。现在我们有一个完整的基于软件的协议图，它包含可供选择的传输协议，我们可以讨论如何有效地测量它的性能。这种测量的重要性在于它们代表了应用程序所看到的性能。

我们按照实验结论报告的方式开始描述实验方法，包括实验中用到的设备。我们的实验设备为：每个工作站双核 CPU，2.4GHz Xeon 处理器，Linux 操作系统。为了使速度达到 1Gbps 以上，每台机器使用一对以太网适配器（标记的网卡，用于网络接口卡）。以太网的跨度是一个机房，所以传播不是问题，这使本实验只测量处理器/软件的开销。一个运行在套接字接口上的测试程序只是简单地尝试尽快发消息。图 5-11 说明了该实验的设置。

你可能注意到，由于硬件、连接速度和操作系统的原因，该实验设置并不理想。本节的重点不是演示一个特定协议能够运行得多快，而是描述测量和报告协议性能的通用方法。

对不同大小的消息使用称为 TTCP 的基准测试工具进行吞吐量测试。吞吐量测试的结果如图 5-12 所示。在这个图中要注意的关键事情是，吞吐量随报文长度的增加而增加。这是讲得通的，因为每个报文包含一定的开销，所以较大的报文意味着将这个开销分摊到更多的字节上。吞吐量曲线在大约 1KB 的位置变平，从这一点开始，每个报文的开销与协议栈处理的大量字节相比变得无关紧要。

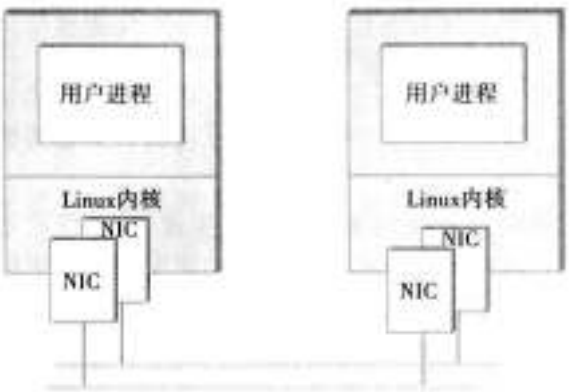


图 5-11 测试系统：两个 Linux 工作站和一对 Gbps 级以太网链路

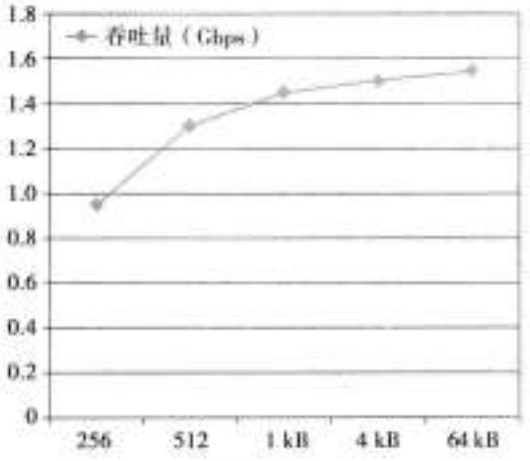


图 5-12 对不同报文长度，用 TCP 测量的吞吐量

值得注意的是，最大吞吐量小于 2Gbps，是这个设置中的有效链接速度。对结果的进一步测试和分析需要找出瓶颈（或者是否有多个因素影响）。例如，CPU 负载指标可能指出瓶颈是 CPU 还是内存带宽、适配器性能或者其他问题。

我们同时还注意到在这个测试中网络基本上是“完美”的。基本上没有延时和丢失，唯一影响性能的就是 TCP 的实现机制和工作站的软硬件环境。相比之下，大多数时候我们应对的是不够完善的网络：特别是带宽受限、最后一英里连接和易丢包的无线连接。在了解这些连接如何影响 TCP 性能之前，我们需要理解 TCP 如何解决拥塞，这正是 6.3 节的主题。

自从有网络以来，网络链路的增速已经无数次影响到网络发送给应用程序的数据。例如，美国在 1989 年启动一项浩大的研究工作——构建千兆以太网。目的不仅是使链路和交换机速度达到甚至超过 1Gbps，而是希望将这样的吞吐量直接传送给单个应用程序进程。这遇到很多实际的问题（比如，网络适配器、工作站的体系结构以及操作系统都必须在设计时考虑到网络应用程序的吞吐量），同时证明了很多难题也没那么复杂。所有问题中最让人关心的是现存的传输协议，特别是 TCP 可能达不到千兆位操作。

事实证明，TCP 紧跟高速网络和应用不断增加的需求，为解决高带宽延迟积而引入滑动窗口就是一个重要的例子。不过，TCP 的理论性能和实际表现还是有很大差异。简单的问题也会导致性能降低，比如数据从网络适配器传送到应用程序的过程中多次不必要地复制数据，又如当带宽延迟积很大时缓存空间不足。同时，TCP 的动态变化也很复杂（在下一章会更加明显）。网络行为和应用行为中微小的相互作用以及 TCP 协议自身原因都能使其突然改变性能。

对于我们的目的来说，TCP 在网速增加的同时表现良好这一点应该引起注意。当 TCP 遇到一些限制时（正常来说是拥塞、增加的带宽延迟积或者两者都有），研究者急于找到解决方案。我们在本章已经探讨过其中的一些，在下一章我们会研究更多方案。

5.2.10 其他设计选择

尽管已经证明 TCP 是一个健壮的协议，能够满足广泛的应用需要，但传输协议的设

计空间是相当大的。在此设计空间上，TCP 绝不是唯一的有效选择，我们通过考查另一些设计选择来结束对 TCP 的讨论。虽然对 TCP 的设计者为什么做这些选择给出了解释，但我们要观察针对另一些选择而设计的传输协议和未来可能出现的传输协议。

第一，本书的第 1 章就提出，至少有两类令人感兴趣的传输协议：像 TCP 一样的面向流的协议和像 RPC 一样的请求/应答协议。换句话说，我们已经隐含地将设计空间分为两半且将 TCP 放在面向流的那一半。我们可以进一步将面向流的协议分为两组：可靠的和不可靠的。前者包括 TCP，后者更适于交互式的视频应用，它宁可丢弃帧也不引发与重传相关的延迟。

建立传输协议分类学是非常有趣的，并可以继续越分越细，但世界并不是我们喜欢的那样黑白分明。例如，考虑 TCP 作为传输协议对请求/应答应用的适用性。TCP 是一个全双工的协议，所以很容易打开客户端和服务器间的 TCP 连接，在一个方向发送请求消息，而在另一个方向发送应答消息。然而，存在两种复杂情况。第一种情况是 TCP 为面向字节（byte）的协议而不是面向数据报（message）的协议，但请求/应答应用总是处理报文。（我们马上就仔细探讨字节和数据报的问题。）第二种情况是，在请求消息和应答消息都能包含在一个网络分组中的情况下，一个设计良好的请求/应答协议仅需要两个分组来完成交换，而 TCP 至少需要 9 个分组，其中 3 个用于建立连接，2 个用于消息交换，4 个用于断开连接。当然，如果请求和应答消息大到足以需要多个网络分组（例如，需要用 100 个分组来发送 100 000 字节的应答消息），那么建立和断开连接的开销就无关紧要了。换句话说，并不总是某个协议不能支持某一特定功能这样一种情况，而有时是在特定情况下某种设计比另一种设计更有效。

第二，正如刚提到的，你可能会问为什么 TCP 选择提供可靠的字节（byte）流服务，而不提供可靠的数据报（message）服务，数据报应该是要进行记录交换的数据库应用的自然选择。对这个问题有两个答案。第一个是，面向数据报的协议必须通过定义确立一个报文的长度上界。毕竟，一个无限长的报文就是一个字节流。但是对于协议选择的任何一个长度上界，总会有某个应用程序想要发送更大的报文，使得传输协议无用，并且迫使应用程序实现它自己的类似传输的服务。第二个是，虽然面向数据报的协议显然更适用于彼此间要发送记录的应用，但它可以很容易地在字节流中插入记录边界来实现这一功能，如 5.2.7 节所述。

在 TCP 设计中做出的第三个决定就是应用程序按序传输字节流。这意味着，它可以从网络中接收乱序字节流，等待补齐丢失的字节。这对于很多应用很有帮助，但对于能够自己处理乱序字节的应用就没多大用处。举个简单的例子，包含多个嵌入图像的网页不需要在所有图像按序收到后才开始显示在页面上。还有一类应用希望在应用层来处理乱序数据，以便当分组丢失或乱序时更快地获取数据。为支持这些应用而产生了另一种 IETF 标准传输协议，如流控制传输协议（Stream Control Transmission Protocol，SCTP）。SCTP 协议提供部分有序传输服务，而不是严格有序的传输服务。（SCTP 还有一些与 TCP 不同的设计决策，包括消息定向和单一会话支持多 IP 地址。详见“扩展阅读”。）

第四，TCP 选择实现显式的建立/断开阶段，但这不是必需的。对建立连接来说，它显然可以与第一个数据消息一起发送全部必要的连接参数。TCP 之所以选择更保守的方法是要让接收方在任何数据到达之前有机会拒绝连接。对于断开连接来说，虽能简单地关闭一个很长一段时间不活动的连接，但这对像 Telnet 那样一次连接要保持几个星期的应

用来说会使问题复杂化，这样的应用将被强制要求发送带外的“存留”消息以防止另一端的连接状态消失。

最后，TCP 是一个基于窗口的协议，但这不是唯一的可能性。可供选择的有基于速率（rate-based）的设计，接收方通知发送方愿意接收的输入数据的速率，速率用每秒字节数或分组数表示。例如，接收方可能通知发送方它每秒能处理 100 个分组。在窗口和速率间存在一个有趣的对偶性，因为窗口中的分组（字节）数除以 RTT 恰好就是速率。例如，一个 10 个分组长的窗口大小和一个 100 ms 的 RTT 意味着允许发送方以每秒传输 100 个分组的速率传送。通过增大或减小窗口的尺寸，接收方能有效地提高和降低发送方的传输速率。在 TCP 中，这个信息在每个数据段的 ACK 的 AdvertisedWindow 字段中被反馈给发送方。基于速率的协议的关键问题之一是每隔多久将期望的速率（一段时间后可能改变）传达给源主机：是对每个分组或每隔一个 RTT 回送，还是仅当速率改变时回送？刚才虽然在流量控制的环境中讨论了窗口和速率，但在第 6 章将要讨论的拥塞控制环境下它们将是一个更有争议的问题。

5.3 远程过程调用

正如在第 1 章中所讨论的，应用程序使用的通用通信模式是请求/应答模式，也称为消息事务：客户端向服务器发送一条请求消息，服务器用一条应答消息响应，而客户端阻塞（挂起执行）等待这个应答。图 5-13 说明了在这样的消息事务中客户端和服务器的基本交互活动。

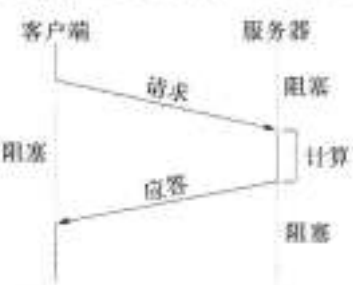


图 5-13 RPC 的时间线

支持请求/应答模式的传输协议不只是在一个方向上传递 UDP 报文，在另一个方向上也传递 UDP 报文。它需要正确处理在远程主机上的标识进程，并建立起请求和应答之间的联系。它还要克服本章开始的问题中概述的底层网络的所有限制。虽然 TCP 通过提供一个可靠的字节流服务克服了这些限制，但它也不能很好地配合请求/应答模式，因为只为交换一对消息就要创建 TCP 连接看起来似乎矫枉过正。本节描述第三种传输协议，称为远程过程调用（Remote Procedure Call，RPC），它与涉及请求/应答报文交换的应用的需求更匹配。

5.3.1 RPC 基础

实际上 RPC 不仅仅是一个协议，它是构造分布式系统的一种流行机制。RPC 之所以流行，是因为它基于本地过程调用语义：应用程序首先调用一个过程，不管它是本地调用还是远程调用，接着阻塞直到调用返回结果。应用程序开发者可能没有意识到调用是本地的还是远程的，因此任务被大大简化了。在面向对象的语言中，当调用过程是远程对象方法时，RPC 被认为是远程方法范例（RMI）。虽然这听起来可能比较简单，但有两个主要问题使得 RPC 比本地过程调用更复杂：

- 调用进程和被调用进程间的网络比一台计算机的情况有更复杂的特性。例如，它可能限制消息的大小，并且有丢失和重排消息的可能。
- 运行调用和被调用进程的计算机可能有明显不同的体系结构和数据表示格式。

这样，一个完整的 RPC 机制实际上包含两个主要部分：

1）协议，用于管理客户端与服务器进程之间发送的消息以及处理底层网络的潜在不合理特性。

2）编程语言和编译程序，它支持在客户端把参数封装进一个请求报文，并在服务器上把报文翻译成原来的参数，对返回值也能做类似的处理（RPC 机制中的这部分通常称为桩编译器（stub compiler））。

图 5-14 描述了当一个客户端调用一个远程过程时发生的事情。首先，客户端调用过程的本地桩，并把过程需要的参数传递给它。桩把参数翻译为一个请求消息，接着调用 RPC 协议把这个请求消息发送给服务器，这样桩隐藏了过程是在远端的事实。在服务器端，RPC 协议把这个请求消息传递给服务器桩（有时称为框架（skeleton）），由它负责把消息翻译为过程的参数并调用本地过程。服务器过程完成后，它把执行结果送给服务器桩，服务器桩将结果封装到应答消息中，并把它原封不动地送到 RPC 协议以便发送回客户端。客户端的 RPC 协议将收到的消息上传给客户端桩，客户端桩将消息翻译成返回值送给客户端程序。

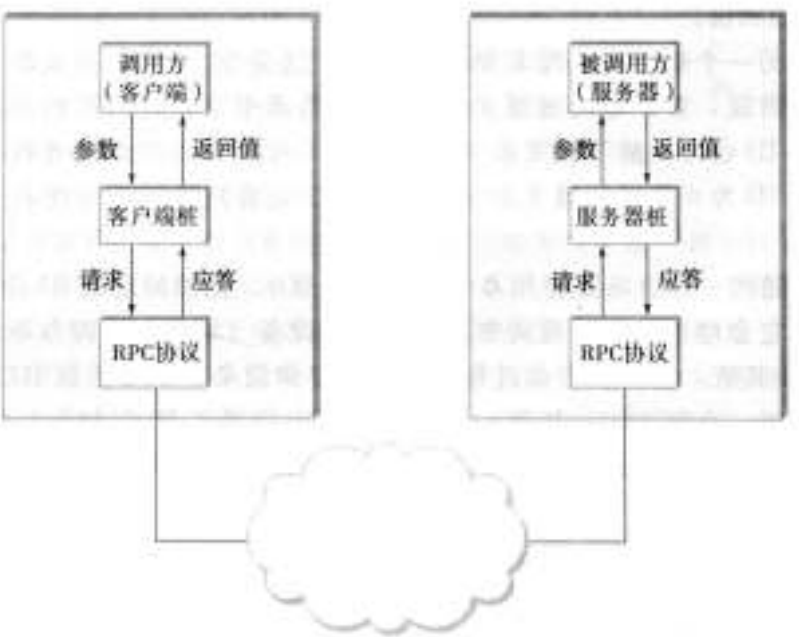


图 5-14 完整的 RPC 机制

本节只考虑 RPC 机制与协议相关的方面。也就是说，忽略桩而着眼于客户端和服务器之间传送消息的 RPC 协议，有时叫请求/应答协议。参数和消息之间的相互转换将在第 7 章描述。

由于 RPC 这个术语只是指一类协议，而不是像 TCP 那样是一个特定的标准，因此 RPC 协议会随它所执行的功能而有所不同。TCP 是可靠的字节流协议，而 RPC 不是，没有一个绝对可靠的 RPC 协议。因此，本节将讨论不同于前面的其他设计选择。

1. RPC 中的标识符

任何 RPC 协议必须执行两个功能：

- 提供一个命名空间来唯一标识被调用的过程。

• 将每一个应答报文与请求报文相匹配。

第一个问题类似于在网络中识别节点的问题，正如在前面章节中所介绍的一样（例如第4章中的IP地址）。进行识别的一种设计选择是命名空间是扁平的还是分层的。扁平的命名空间仅仅为每个过程分配一个唯一的、无结构的标识符（例如一个整数），这个数字标识符被装载到RPC请求报文的一个字段中。这就需要一个中心协调者以避免为两个不同的过程指派相同的过程标识符。此外，协议也可以实现一个分层的命名空间，类似于文件路径名，只需要在同一个文件目录下具有唯一的文件名即可。这种方法简化了过程命名，能够确保过程名的唯一性。RPC分层的名字空间可以通过在请求报文中定义一组字段来实现，在两层或三层的命名空间中每层都有一个名字。

将应答报文与请求报文匹配的关键是使用报文ID字段来识别请求-应答对。应答报文的ID字段与请求报文的ID字段具有相同的值。当客户端RPC模块收到应答报文时，它使用报文ID来搜索相应的请求。为了使RPC事务对调用者来说像本地过程调用一样，在收到应答报文前调用者被阻塞（例如使用信号量）。接收到应答报文时，基于应答报文中的请求号来识别被阻塞的调用者，从应答报文中获取远程过程的返回值，将调用者解除阻塞，以便它能返回值。

RPC中的另一个挑战是处理非预期的响应，这是和报文ID相关联的。例如，考虑下面的实际情况，客户端发送报文ID为0的请求报文，之后死机并重启，然后发送另一个报文ID也为0的不相关报文。服务器不可能知道客户机死机并重启了，因此它收到报文ID为0的重复报文后将其丢弃，于是客户端就不可能收到对于第二个请求的响应。

解决该问题的一种方法是使用启动ID（boot ID）。机器的启动ID是数字，每次机器重新启动后它会增加，启动后从非易失性存储设备（如磁盘或闪存驱动器）中读取该ID号，增加其值，然后在启动过程中写回到存储设备中，之后该ID号会被放在该主机所发送的每一个报文中。如果一个报文具有旧的报文ID但却有新的启动ID，则会被识别为一个新的报文。这样，报文ID和启动ID组合起来构成了每一个事务的唯一ID。

相关主题

RPC位于哪一层？

"这是哪一层"的问题再次显露出来。对很多人来说，特别是那些持协议体系结构应该严格分层观点的人，RPC是在传输层协议（通常是UDP）之上实现的，所以它自己（按照定义）不能算是一个传输层协议。但是因特网需要有RPC协议，因为它提供了一个与TCP和UDP所提供的服务根本不同的进程到进程的服务。然而，对这一建议通常的反应是因特网体系结构并不禁止网络设计人员在UDP之上实现他们自己的RPC协议。不管因特网是否应该具有你所支持的RPC协议，重要的是在因特网体系结构上实现RPC的方法与RPC是否是一个传输层协议无关。

有趣的是，另有一些人认为RPC是世界上最有意义的协议，而TCP/IP只不过是当你想到"脱离站点"时才做的事。这是操作系统领域中的主流观点，人们已经构造了无数分布式系统的操作系统内核，这些系统只包含一个运行在网络设备驱动程序之上的协议——你可以

借到就是 RPC 协议。

我们的观点是，任何提供进程到进程服务而不是节点到节点或主机到主机服务的协议，都可称作传输层协议。这样，RPC 就是一个传输层协议，但它可以在其他传输层协议之上来实现。

2. 克服网络局限性

RPC 协议通常执行额外的功能来应对网络信道不顺畅的情况，两个主要功能是：

- 提供可靠报文传输机制。
- 通过分片和重组支持大尺寸的报文。

RPC 协议需要提供可靠性，因为其下层协议（例如 UDP/IP）没有提供可靠性，或许还要快速或有效地从故障中恢复，否则故障最终会由下层协议修复。RPC 协议通过类似于 TCP 中的确认和超时机制来实现可靠性，其基本的算法很直观，如图 5-15 所示。客户机发送一个请求报文，服务器对其进行确认，过程执行完后，服务器发送一个应答报文，客户机对该应答进行确认。

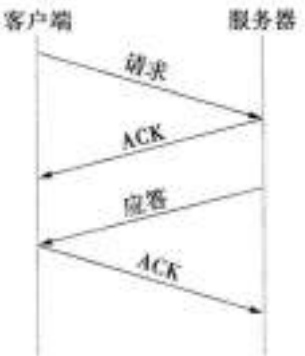


图 5-15 可靠 RPC 协议的简单时间线

承载数据（请求报文或应答报文）或 ACK（确认）的报文在网络中可能会丢失。为了应对这种可能性，客户端和服务器对它们所发送的每一个报文都会保留一份副本，直到与其相应的 ACK 已经收到。每一方也都会维持一个 RETRANSMIT（重传）定时器，超时后就重传报文。在放弃和释放报文之前双方会重新启动定时器并再一次协商定时器的取值。

如果 RPC 客户端收到了一个应答报文，那么服务器一定已经收到了相应的请求报文，因此应答报文本身就是一个隐式的确认，无需服务器再专门发送确认报文。类似地，请求报文可以隐式地确认先前的应答报文——假设协议要求请求-应答事务是连续的，即一个事务在下一个事务开始之前必须已经完成。不幸的是，这种连续性会严重限制 RPC 的性能。

对于 RPC 协议来说，摆脱这种困境的一种方法是使用信道（channel）抽象。在一个给定的信道中，请求/应答事务是连续的（在给定信道的某一时刻只有一个事务是活跃的），但同时可以有多个信道。每个报文包含一个信道 ID 字段来指明其所属的信道。如果先前未被确认过的话，则在给定信道中的一个请求报文隐式地确认该信道中先前的应答报文。如果应用程序同时有多个请求/应答事务需要和服务器交互的话（应用程序有多个线程），那么它可以打开和服务器之间的多个信道。如图 5-16 所示，应答报文对请求报文进行确认，接下来的请求报文对先前的应答报文进行确认。注意，这里我们看到了称为并发逻辑信道的类似方法，在 2.5.3 节中该方法用于改善停止-等待可靠传输机制的性能。

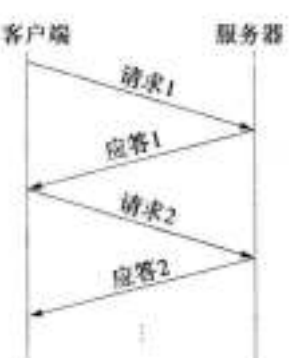


图 5-16 使用隐式确认的可靠 RPC 协议的时间线

RPC 必须考虑的另一个复杂问题是服务器可能需要花

费很长时间来得出计算结果，也可能在计算过程中出错，这样在产生应答之前可能会崩溃。这里所说的时间段是在服务器对请求发送了确认之后，但在发送应答之前。为了使客户端能够对响应慢的服务器和已死机的服务器进行区分，RPC 客户端周期性地向服务器发送“你是活跃的吗?”报文，服务器用 ACK 对该报文进行响应。另一种办法是，服务器向客户端发送“我仍然活跃着”报文，而无需客户端首先发送报文。客户端首先发出询问的方法具有更好的扩展性，因为它把该任务（管理定时器）放在了客户端。

RPC 的可靠性包括最多一次语义（at-most-once semantics）。这意味着对于客户端发送的每个请求报文，最多只有一个报文副本被传递到服务器。客户端每次调用远程过程时，服务器上的过程最多被调用一次。我们说“最多一次”而不是“正好一次”，因为总有可能网络或者服务器出故障，这时即使传递请求报文的一个副本也是不可能的。

为了实现最多一次语义，RPC 服务器端必须识别重复请求并忽略之，因此它必须维护一些识别先前请求的状态信息。一种方法是使用序列号识别请求，这样服务器只要记住最近的序列号就可以了。不幸的是，这样就限制了 RPC 同时只能给一个服务器发送一个请求，因为在带有下一个序列号的请求被传输之前先前的请求必须已经完成。信道提供了一种解决办法，通过记住每个信道的当前序列号，服务器能够识别重复的请求，而不必限制客户端同时只能发送一个请求。

正如“最多一次”听起来一样明显，并不是所有 RPC 协议都支持这种行为。一些协议支持所谓的零次或多次（zero-or-more）语义，也就是说，客户端的每次调用导致远程过程被调用零次或多次。不难理解，这种方式会引起问题，因为远程过程的每次调用都可能会改变一些本地的状态变量（例如，增加计数器的值）或造成外部可见的副作用（例如，发射一枚导弹）。另一方面，如果远程过程的调用是幂等（idempotent）的，即多次调用产生的效果和一次调用一样，那么 RPC 机制就不必支持最多一次语义，用一个更简单（可能更快速）的实现就足够了。

作为可靠传输协议，RPC 需要实现报文分片和重组有两个原因：一是下层协议栈没有提供该功能；二是通过 RPC 协议来实现该功能会更有效。考虑 RPC 在 UDP/IP 之上依赖于 IP 实现分片和重组的情况，如果在特定的时间段内报文的一个分片传输失败，那么 IP 丢弃该报文的所有分片，从而造成该报文的丢失，最终 RPC 协议（假设它实现了可靠性）发生超时并重传该报文。与此相反，考虑 RPC 协议自身实现分片和重组的情况，每个分片都有各自的 ACK（确认）或 NACK（否认），丢失的分片会被更快地检测到并重传，这样就只有丢失的分片需要重传，而不必重传整个报文。

3. 同步协议与异步协议

描绘协议特性的一种方式是说明它是同步的（synchronous）还是异步的（asynchronous）。这两个术语的具体含义依赖于它们用于协议层次结构中的哪一层。在传输层，最好将它们看成是一些可能性的范围，而不是两种可能性之一，这些可能性中的任意一个的关键属性是发送报文的操作返回后，发送方知道多少信息。换句话说，如果假设一个应用程序调用了传输协议上的 send 操作，那么问题是，send 操作返回时，应用程序对操作是否成功知道多少?

在这个范围的异步一端，当 send 操作返回时，应用程序绝对不知道任何事情。它不

仅不知道报文是否被对等方接收，甚至不能确切知道报文是否成功地离开本地机器。在这个范围的同步一端，send 操作通常返回一个应答报文。也就是说，应用程序不仅知道发送的报文被对等方接收，而且知道对等方返回一个应答。这样，同步协议实现了请求/应答的抽象，而如果发送方想要传输很多报文而不必等待响应，则使用异步协议。应用这个定义，RPC 协议显然是一个同步协议。

在这两个极端间存在一些值得注意的问题，尽管在本章我们没有讨论它们。例如，传输协议可以实现 send 后阻塞（不返回），直到报文成功地被远程机器接收，但在发送方的对等方实际处理完毕并响应之前返回。有时这称为可靠数据报协议（reliable datagram protocol）。

5.3.2 RPC 实现（SunRPC、DCE）

现在讨论 RPC 协议实现的例子，这将突出协议设计者的不同设计决定。第一个例子是 SunRPC，这是一个被广泛使用的 RPC 协议，也称为开放网络计算 RPC（Open Network Computing RPC，ONCRPC）。第二个例子是 DCE-RPC，它是分布式计算环境（Distributed Computing Environment，DCE）的一部分。DCE 是构造由开放软件基金会（Open Software Foundation，OSF）定义的分布式系统的一套标准和软件，OSF 是由一些计算机公司组成的联盟，最初包括 IBM、DEC 和 HP 公司，现在 OSF 更名为开放组（The Open Group）。这两个例子在 RPC 解决方案中非常具有代表性。

1. SunRPC

SunRPC 已经成为一个事实上的标准，这得益于它与 Sun 工作站一起被广泛应用及其在流行的 Sun 网络文件系统（Network File System，NFS）中所起的中心作用。IETF 采纳了 SunRPC，将其作为一个标准的因特网协议，命名在 ONC RPC 之下。

SunRPC 在多个不同的传输层协议之上实现，图 5-17 描述了在 UDP 之上实现 SunRPC 的协议图。正如本章前面所提到的，严格的层次结构主义者可能会反对在传输层协议之上运行传输层协议的想法，或者会因为 RPC 出现在传输层之上而争论 RPC 不是传输层协议。然而，正如下面将要讨论的，将 RPC 运行在现有的传输层之上的设计决定有很多考虑因素。

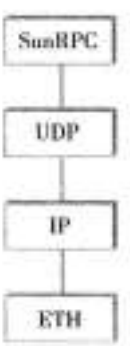


图 5-17 在 UDP 之上的 SunRPC 协议图

SunRPC 使用两层标识符来识别远程过程：一个 32 位的程序号和一个 32 位的过程号（还有 32 位的版本号，在下面的讨论中我们将其忽略）。例如，NFS 服务器已经被赋予程序号 x00100003，在这个程序中，getattr 是过程 1，setattr 是过程 2，read 是过程 6，write 是过程 8，等等。程序号和过程号被放置在 SunRPC 请求报文的首部来传输，其首部字段如图 5-18 所示。对于调用特定程序的特定过程而言，支持多个程序号的服务器是尽责的。SunRPC 请求实际上代表了一种请求，该请求是调用特定机器上特定程序和过程的请求，相同的程序号可以用在相同网络中的不同机器上。这样，服务器的地址（如 IP 地址）就成了 RPC 地址中的第三层地址。

不同程序号可能属于同一台机器上的不同服务器，这些不同的服务器具有不同的传输层标识（例如 UDP 端口），其中大部分都是被动态指派的非知名端口，这些标识



图 5-18 SunRPC 首部格式

称为传输层选择器（transport selector）。一个想要与特定程序交互的 SunRPC 客户端如何决定应该使用哪一个传输层选择器到达相应的服务器呢？解决办法是给远程机器上的一个程序指派知名地址，让该程序告诉客户端哪个传输层选择器能够到达该机器上的其他程序。该 SunRPC 程序的最初版本称为端口映射（Port Mapper），它仅支持下层协议为 UDP 或 TCP 的情况。它的程序号是 x00100000，知名端口号是 111。RPCBIND 由端口映射程序演化而来，它支持任意的下层传输层协议。当 SunRPC 服务器启动时，它调用位于服务器宿主机上的 RPCBIND 注册过程来注册其支持的传输层选择器和程序号，之后远程客户端就能调用 RPCBIND 过程查询对应于特定程序号的传输层选择器。

为了便于理解，考虑在 UDP 上使用端口映射的例子，为了给 NFS 的 read 过程发送请求报文，客户端首先要给 UDP 端口 111 上的端口映射程序发送一个请求报文，请求调用过程 3 以把程序号 x00100003 映射为当前 NFS 程序所在的 UDP 端口。客户端接着向该 UDP 端口发送程序号为 x00100003 过程号为 6 的 SunRPC 请求报文，在该端口监听的 SunRPC 模块调用 NFS 的 read 过程。客户端也保存程序号到端口号的映射，所以不必在每次想与 NFS 对话的时候都要返回到端口映射程序。

为了将应答报文与相应的请求报文相匹配，以便 RPC 能返回到正确的调用者，请求报文和应答报文的首部都包含有 XID（事务 ID）字段，如图 5-18 所示。XID 是唯一的事务标识符，仅用于一次请求和相应的应答中。在服务器对给定的请求进行了成功的应答后，它不会记得该 XID。因此，SunRPC 不能保证最多一次语义。

SunRPC 语义的细节依赖于下层的传输层协议，它没有实现自己的可靠性，因此仅当下层传输层协议可靠时它才是可靠的（当然，任何运行在 SunRPC 之上的应用程序也可以选择在 SunRPC 之上实现其自身的可靠机制）。发送比网络 MTU 大的请求和应答报文的能力也依赖于下层的传输层。换句话说，在可靠性和报文大小方面，SunRPC 相对于下层传输层来说并没有做任何改进，因此 SunRPC 能够运行在许多不同的传输层协议之上，其

有相当大的灵活性，没有给 RPC 协议的设计增加任何复杂性。

回到图 5-18 所示的 SunRPC 的首部格式，请求报文包含变长的 Credentials（证书）和 Verifier（校验）字段，它们都被客户端用来向服务器认证自己，也就是说，给出客户端有权访问服务器的证据。客户端如何向服务器认证自己是一个普遍的问题，它必须由任何希望提供合理安全级别的协议来解决。关于该问题的更详细的讨论见下一章。

2. DCE-RPC

DCE-RPC 是 DCE 系统核心部分的 RPC 协议，也是微软 DCOM 和 ActiveX 中 RPC 机制的基础。它可以和第 7 章描述的网络数据表示（Network Data Representation，NDR）桩编译器一起使用，但它也可以作为公用对象请求代理体系结构（Common Object Request Broker Architecture，CORBA）的基础 RPC 协议，CORBA 是一个构建分布式、面向对象系统的行业标准。

像 SunRPC 一样，DCE-RPC 被设计为运行在包括 UDP 和 TCP 在内的多个传输层协议上。与 SunRPC 类似，它定义了一个两层编址方案：传输层协议多路分解到正确的服务器，DCE－RPC 分发到由该服务器输入的特定过程，客户端查阅“端点映射服务”（类似于 SunRPC 的端口映射）以获得如何到达特定服务器的途径。然而，与 SunRPC 不同的是，DCE-RPC 实现最多一次调用语义（事实上，DCE-RPC 支持多个调用语义，包括类似于 SunRPC 的幂等语义，但最多一次语义是其默认行为）。在下面的段落中将突出两种方法的区别之处。

图 5-19 给出了典型报文交换的时间线，其中每个报文由它的 DCE-RPC 类型标明。客户端发送一个 Request 报文，服务器最终用一个 Response 报文来应答，然后客户端确认（Ack）这个响应。然而，在 DCE-RPC 中服务器并不确认请求报文，而是由客户端定期向服务器发送一个 Ping 报文，服务器用一个 Working 报文响应表示远程过程仍在处理中。如果服务器的应答很快被接收，则不需要发送 Ping 报文。尽管图中没有显示，但 DCE-RPC 也支持其他报文类型，例如，客户端能够向服务器发送一个 Quit 报文，要求服务器终止一个仍在处理的早期调用，服务器响应一个 Quack（退出确认）报文。服务器也可以用一个 Reject（表示一个调用已被拒绝）报文响应 Request 报文，并且能够用一个 Nocall 报文（表示服务器未曾接收到调用方的请求）响应 Ping 报文。

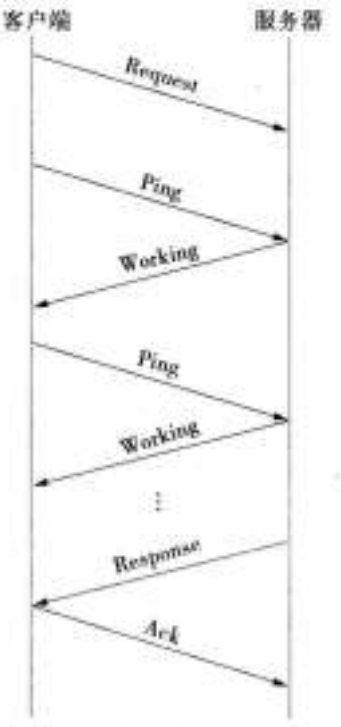


图 5-19 典型的 DCE-RPC 报文交换

DCE-RPC 中的每个请求/应答事务都发生在*活跃期*（activity），活跃期是一对参与方之间的逻辑请求/应答信道。在任何时刻，在给定信道中只有一个报文事务是活跃的。就像在 2.5.3 节中所描述的并发逻辑信道方法一样，如果应用程序同时想有多个请求/应答事务，那么就必须打开多个信道。报文所属的活跃期通过报文的 ActivityId 字段来识别，SequenceNum 字段用来区分同一活跃期中的不同调用，它与 SunRPC 中的 XID（事务 ID）字段具有相同的作用。与 SunRPC 不同，DCE-RPC 跟踪特

定活跃期中最后使用的序号，以此保证最多一次语义。为了在服务器重启之前或之后对应答进行区分，DCE-RPC 使用 ServerBoot 字段来保存机器的启动 ID。

DCE-RPC 中另外一个不同于 SunRPC 的设计选择是在 RPC 协议中支持分段和重组。正如上面所提到的，即使下层协议（例如 IP）提供了分段/重组功能，当分段丢失时，作为 RPC 的一部分来实现的该算法能够进行快速恢复，减少带宽消耗。FragmentNum 字段唯一地识别构成给定请求或应答报文的每个分段。每个 DCE-RPC 分段被指派了一个唯一的分段号（0、1、2、3 等）。客户端和服务器都实现了一种选择性的确认机制，其执行过程如下。（从客户端给服务器发送一个分段请求报文的角度来描述这个机制，当服务器给客户端发送一个分段响应时，采用同样的机制。）

首先，构成请求报文的每个分段都包含一个唯一的 FragmentNum 和一个指示该分组是一个调用的分段（frag）或是一个调用的最后分段（last_frag）的标志，容纳一个单独分组的请求报文携带一个 no_frag 标志。当服务器接收到具有 last_frag 标志的分组而且分组的分段号没有间断时，它就知道已接收了一个完整的请求报文。其次，为了响应每个到达的分段，服务器发送一个 Fack（分段确认）报文给客户端。该确认标明服务器已经成功收到的最大分段号。换句话说，确认是累积确认，很像 TCP 中的确认。然而，服务器同时也选择地确认它错序接收的较大分段号。它使用位向量来实现这一功能，位向量标识相对于按序接收到的最大号分段的那些乱序分段。最后，客户端用重传丢失的分段来响应。

图 5-20 解释了整个工作过程。假设服务器已成功接收到直到序号为 20 的各个分段，以及序号为 23、25 和 26 的分段。服务器用一个 Fack 响应，Fack 标识分段 20 是最大有序分段，并加上一个第 3（23＝20＋3）位、第 5（25＝20＋5）位和第 6（26＝20＋6）位为 1 的位向量（SelAck）。为了支持一个（几乎）任意长的比特向量，向量的长度（以 32 位字度量）在 SelAckLen 字段中给出。

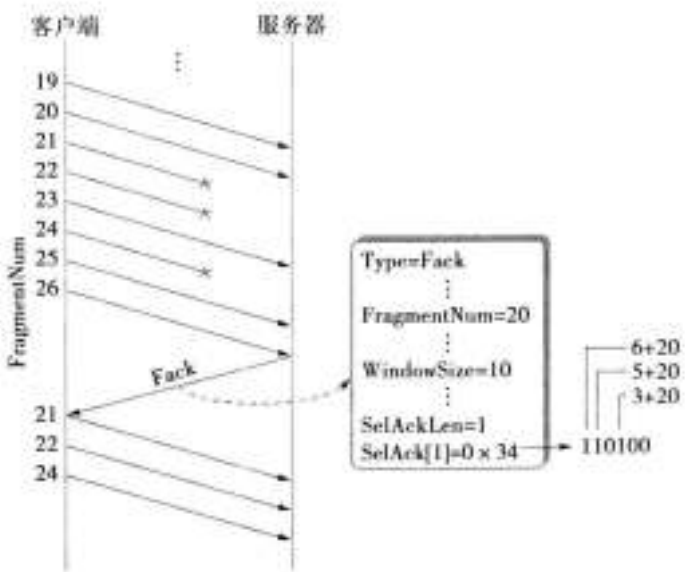


图 5-20 带有选择性确认的分段

如果 DCE-RPC 支持很大的报文——FragmentNum 字段长度为 16 位，这表示它能够

支持 64K 个分段——那么尽可能快地处理掉组成一个报文的所有分段对该协议来说是不适合的，因为这样做可能超出接收方的限度。相反，DCE-RPC 实现了一个类似于 TCP 的流量控制算法。尤其是，每个 Fack 报文不仅仅确认接收到的分段，同时也通知发送方现在可以发送多少个分段。这正是图 5-20 中的 WindowSize 字段的目的，它与 TCP 的 AdvertisedWindow 字段的目的类似，但它计数的是分段而不是字节。DCE-RPC 也实现了一个类似于 TCP 的拥塞控制机制，我们将在第 6 章中讨论。鉴于拥塞控制的复杂性，一些 RPC 协议通过避免分段来避免拥塞。

总之，在设计一个 RPC 协议时，设计者有比较大的选择范围。SunRPC 能够实现最低限度的功能，对下层的传输层增加了相关功能，能够定位正确的过程并识别报文。DEC-RPC 增加了更多的功能，在更复杂的环境下改善了性能。

5.4 实时应用程序传输（RTP）

在分组交换的早期阶段，大部分应用程序涉及的都是数据的传输：访问远程计算资源、传输文件、发送电子邮件等。然而，早在 1981 年，在分组网络上就已经开始了传输实时流量的实验，例如传输数字化音频。当应用程序对及时传输信息有强烈需求时，该应用程序就被称为是实时的。因特网电话或 IP 电话（VoIP）是实时应用程序的典型例子，因为在一次通话过程中延迟 1s 以上才得到对方的响应是不能容忍的。我们将会看到，实时应用程序对传输层协议有特殊要求，这些要求是本章中已经讨论过的协议所无法满足的。



图 5-21 vat 音频会议软件的用户界面

多媒体应用（涉及视频、音频和数据）可以分为两类：*交互式*（interactive）应用和*流式*（streaming）应用。交互式应用程序早期流行一时的例子是 vat，这是一个音频会议工具，通常用于支持 IP 多播的网络上。图 5-21 显示了一个典型 vat 会议的控制面板。因特网电话也是一个交互式的应用，在本书写作时它可能是最广泛使用的交互式应用。另一个交互式应用程序的例子是基于互联网的多媒体会议，参见第 1 章的图 1-1。现代的即时消息应用程序也使用实时音频和视频。这些都是对实时需求最迫切的应用程序。

流式应用程序主要是从服务器向客户端传输音频或视频流，其代表是像 Real Audio® 这样的商业产品。以 YouTube 为代表的视频流式应用程序已经成为流量互联网上的主要形式。流式应用由于缺乏人与人的交互，所以对下层协议的实时需求并不是很迫切。然而，假如你希望按下播放键后就能立刻观看视频但由于分组延迟而导致缓冲或视觉延迟，这时实时就显得很重要。虽然这些应用程序对实时性的要求并不严格，但它们与交互式应用程序一样，需要在通用协议之上考虑类似的问题。

用于实时应用和多媒体应用的传输层协议的设计者们在定义满足不同应用的需求方面面临着非常现实的挑战，他们必须要注意不同应用之间的交互性，例如，音频和视频流的同步。我们将会看到这些因素是如何影响实时传输协议（Real-time Transport Protocol，RTP）的设计的，RTP 也是当今的主流传输协议。

RTP 大部分来源于早期 vat 应用中的协议功能，vat 设计之初运行在 UDP 上，设计者发现处理实时音频对话的一些技术特性适用于其他许多应用程序，于是他们定义了包含这些特性的协议，后来演变为 RTP。RTP 能运行在许多底层协议之上，但通常都运行在 UDP 之上，这样其协议栈就如图 5-22 所示。注意到这里是在传输层协议之上运行传输层协议，这并没有违反规则[⊖]，事实上可以有多种理解，因为 UDP 提供了这样一个最低限度的功能层，基于端口号的基本的多路分解正是 RTP 所需要的。优于在 RTP 中重新创建端口号，RTP 利用了 UDP 的多路分解功能。



图 5-22 使用 RTP 的多媒体应用的协议栈

5.4.1 需求

通用多媒体协议的最基本需求是允许类似的应用程序之间交互操作，例如，允许两个独立实现的音频会议应用程序之间相互交谈。这就要求应用程序使用相同的音频编码和压缩方法，否则，接收方将无法理解发送方发送的数据。因为有许多不同的音频编码方法，每种方法在质量、带宽需求、计算代价等方面都有自己的协定，因此只使用一种方法并不是一个好主意。相反，我们的协议应该提供一种方法，使得发送方能够告诉接收方它想要使用的编码方法，双方可能会进行协商，以便找到双方都可接受的方法。

与音频一样，目前也有许多不同的视频编码方法，因此，RTP 的首要功能是提供选择编码方法的能力。注意这也用于识别应用的类型（例如，音频或视频），一旦知道了使用的编码算法，也就知道了被编码的数据的类型。

RTP 的另一个重要需求是使得数据流接收方能够判断已接收数据中的时间关系，在 6.5 节我们将介绍，实时应用程序需要将接收到的数据放入播放缓冲区（playback buffer），以便消除在网络传输过程中可能被引入的数据流中的抖动。因此，数据需要有时间戳（也称时标），以便接收方能在合适的时间进行播放。

与单个媒体流相关的时间是多媒体会议的同步事件，最明显的例子是同步来自同一发送方的音频和视频流。正如下面将会看到的，对单个流来说，这是比播放时间更复杂的问题。

需要提供的另外一个重要的功能是分组丢失指示。注意具有潜在限制的应用程序通常不能使用像 TCP 这样的可靠传输协议，因为对丢失数据的重传可能会使分组到达得太晚，从而无法使用。因此，应用程序必须能够处理丢失的分组，处理的第一步就是通知分组已经丢失。例如，当分组丢失时，使用 MPEG 编码的视频应用程序可能需要采取不同的行动，这依赖于分组是来自 I 帧、B 帧或 P 帧。

分组丢失也暗示网络发生了拥塞。多媒体应用程序通常不运行在 TCP 上，因此无法利用 TCP 的拥塞避免特性（6.3 节介绍）。然而许多多媒体应用程序能够对拥塞进行响应，例如，通过改变编码算法的参数来减少带宽消耗。显然，为了完成该工作，接收方需要通知发送方丢失事件的出现，以便发送方能够调整其编码参数。

多媒体应用程序的另外一个通用功能是帧边界的指示，帧是依赖于特定应用的，例如，通知视频应用程序一个特定的分组集对应某一帧可能是有帮助的。在一个音频应用程

⊖ 但这却使人们对 RTP 是否为真正的传输层协议产生困惑。

序中标记“交谈”的开始可能也是有帮助的，这里所说的交谈是指在沉默之后发出的声音或单词集合。接收方据此就能识别在交谈之间的沉默，并使用这段时间来移动播放点。对于用户来说，交谈时声音之间的空隙发生微小的减少或延长是不易察觉的，然而声音的减少或延长不但是易于察觉的，而且是很令人讨厌的。

想放入协议中的最后的功能是识别发送方的方法，这比 IP 地址更为友好，如图 5-21 所示，音频和视频会议应用能在控制面板上显示字符串，例如 Joe User（user@domain.com），这样应用协议应该支持这样的字符串和数据流的关联。

除了协议所要求的功能外，需要注意一个额外的需求，即它应该尽可能有效地利用带宽。换句话说，我们不希望长首部格式中许多额外的比特随着每个分组被发送，因为音频分组是最常见的多媒体数据类型，它应该尽可能小，以便减少构造分组的时间。长的音频分组意味着长的响应时间，对交谈质量有负面影响（在选择 ATM 信元长度时这是一个考虑因素）。由于数据分组本身比较小，较大的首部意味着相对多的连接带宽被用于首部，这就减少了实际数据所能利用的带宽。我们将会看到设计 RTP 的好几个方面都受到了保持短首部的影响。

我们可以讨论实时传输协议中描述的各个特性存在的必要性，也可以添加更多的特性。关键在于为应用程序开发者描述一系列有用的抽象并为他们构建应用程序块，使他们的工作更轻松。例如，通过在 RTP 中增加时间戳机制，每个实时应用程序开发者不必开发自己的时间戳机制，同时还增加了两个实时应用程序互相交互的机会。

5.4.2 RTP 设计

我们已经看到了用于多媒体的传输层协议有相当多的需求，下面来看看满足这些需求的协议的细节。RTP 协议是由 IETF 开发的，目前应用广泛。RTP 标准实际上定义了一对协议，即 RTP 和实时传输控制协议（Real-time Transport Control Protocol，RTCP），前者用于多媒体数据的交换，后者用于周期性地发送与特定数据流相关联的控制信息。当在 UDP 上运行时，RTP 数据流与相关联的 RTCP 控制流使用连续的传输层端口。RTP 数据使用偶数端口号，RTCP 控制信息使用相连的下一个（奇数）端口号。

因为 RTP 用于支持各种各样的应用，因此它提供了一种灵活的机制，使用该机制不需要重复修订 RTP 协议本身就可以开发新的应用程序。对于每种应用类型（如音频），RTP 定义了一个轮廓（profile）和一个或多个格式（format）。轮廓提供了一个信息范围来确保对各种应用类型中 RTP 首部字段的理解，当我们查看首部细节时会更明白这一点。格式定义了跟随在 RTP 首部之后的数据如何被解释。例如，RTP 首部之后可能只有一个字节序列，每个字节表示一个单一的音频样本。相比较而言，数据的格式可能更为复杂，例如，一个用 MPEG 编码的视频流需要有许多结构来表示各种不同类型的信息。

结论 RTP 的设计包含了一个称为应用层框架（Application Level Framing，ALF）的体系结构原则，该原则是 Clark 和 Tennenhouse 在 1990 年提出的，这是一个为多媒体应用设计协议的新方法。他们认识到这些新应用无法得到像 TCP 这样的现有协议的良好支持，而且可能也无法得到任何“固定大小”的协议的良好支持。该原则的核心是应用程序最理解它自己的需要。例如，MPEG 视频应用程序知道如何最好地恢复丢失的帧，若 I 帧或者 B 帧丢失如何进行不同的处理。同样的应用程序也理解如何将传输的数据进行分段，例如，在不同数据报中最好

发送来自不同帧的数据，以使丢失的分组仅仅影响一个帧。由于该原因 RTP 在相应应用的轮廓和格式中保留了许多协议细节。

首部格式

图 5-23 显示了 RTP 的首部格式，前 12 个字节总是存在，然而仅在特定环境下才使用贡献源标识符。首部之后是扩展首部，下面将有相关描述。之后是 RTP 有效载荷，其格式由具体应用来决定，它包含着可能被多种不同应用所使用的字段，对于不同的应用来说其字段不同，这有利于 RTP 仅仅承载特定应用的相关载荷。



图 5-23 RTP 首部格式

前两位是版本号，本书写作时 RTP 的版本号是 2。协议设计者认为 2 位将足以包含将来所有 RTP 版本号，这些位在 RTP 首部是额外开销，针对各种不同应用的轮廓使用使得对于基本 RTP 协议的修订不会太多。在任何情况下，如果 RTP 需要有超越版本 2 的下一个版本，可以考虑改变首部格式，以便支持将来的多个版本。例如，在版本字段具有值 3 的新的 RTP 首部可能在首部包含一个子版本号字段。

下一位是填充（padding，P）位，当由于某种原因使 RTP 载荷被填充时需要使用该字段。例如，由于加密算法的需要，RTP 数据可能需要填充一些内容。在这种情况下，下层协议（例如，UDP）将传输整个 RTP 首部、数据以及填充内容，填充的字节将会被忽略，见图 5-24。注意这种填充方法使得 RTP 首部不需要长度字段（使得首部尽可能短），在没有填充的情况下，长度可由下层协议推出。

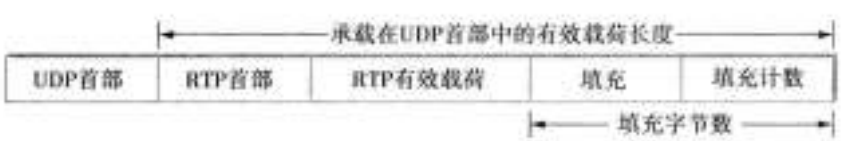


图 5-24 RTP 分组的填充

扩展（extension，X）位用于指示扩展首部的存在，特定的应用会定义扩展首部，它跟随在基本首部之后。扩展首部很少使用，对于特定的应用来说，可能会针对特定载荷定义扩展首部，以说明特定载荷的格式。

X 位之后是一个 4 位的字段，用来说明包含在首部中的贡献源（contributing source）的数量。关于贡献源将会在下文讨论。

根据上文我们注意到需要有对各种各样帧的指示，这由有特殊用途的标记位来提供，例如，在语音应用中标记交谈的开始。接下来是 7 位的有效载荷类型字段，它指示在分组

中承载的是什么类型的多媒体数据。该字段的一种可能用途是基于网络资源可用性或应用质量的反馈，使应用从一种编码方法切换到另一种方法。标记位的确切使用方式和有效载荷类型由应用的配置文件来决定。

有效载荷类型通常不用作多路分解来指示不同应用的数据（或者单一应用中的不同流，例如，视频会议的音频或视频流）多路分解密码，这是因为多路分解功能典型地由下层来提供（例如 UDP，正如 5.1 节中所描述的）。这样，使用 RTP 的两个媒体流将使用不同的 UDP 端口号。

序号用于使 RTP 流的接收方检测丢失或失序的分组，发送方对每个传输的分组递增序号的值，当检测到丢失分组时 RTP 不做任何事情，这一点与 TCP 不同，TCP 纠正丢失（通过重传）并将其理解为拥塞指示（从而可能减小窗口大小）。确切地说，当分组丢失时，由应用来决定该采取什么措施，因为该决定可能是高度依赖应用的。例如，对视频应用来说，最好的决定是当分组丢失时，播放正确接收到的最后的帧。一些应用也可能会修改其编码算法，以减少响应丢失的带宽需求，但这不是 RTP 的功能。对 RTP 来说，决定应该被减少的发送率是不明智的，因为这可能使得应用变得无法使用。

时间戳字段的功能是使接收方在合适的时间间隙内播放样本，并使不同的媒体流同步。因为不同的应用可能需要不同的时间间隔，因此 RTP 本身并没有指定衡量时间的单位。相反，时间戳只是一个计数器，计数的时间间隔依赖于所使用的编码方法。例如，音频应用使用 125ms 作为时钟单位。时钟粒度在 RTP 轮廓或应用的载荷格式中指定。

分组中时间戳的值是表示所产生分组中第一个样本的时间的数字，时间戳不反映实际时间，只反映不同时间戳间的差别。例如，如果样本间隔是 125ms，分组 n 中的第一个样本产生 10ms 后分组 $n+1$ 中的第一个样本产生了，那么这两个样本间的样本实例数是

$$\text{TimeBetweenPackets} \div \text{TimePerSample} = (10\times10^{-3}) \div (125\times10^{-6}) = 80$$

假设时钟间隔与样本间隔是相同的，那么分组 $n+1$ 中的时间戳比分组 n 中的大 80。注意，由于沉默检测这样的压缩技术，小于 80 个的样本可能已经被发送了，然而时间戳允许接收方根据正确的时间关系来播放样本。

同步源（SSRC）是一个 32 位的数字，它唯一地标识 RTP 流的单个源。在一个给定的多媒体会议中，每个发送方随机选择一个 SSRC，并在两个不同的源选择了相同的值时解决冲突。通过使用源标识符，而不是源的网络地址或传输地址，RTP 能够确保与下层协议独立。它也使得具有多个源（例如，好几个摄像头）的单一节点能够对源进行区分。当单一节点产生了不同的媒体流时（例如，音频和视频），在每个流中不需要使用相同的 SSRC，因为在 RTCP（后面将会描述）中有机制能够允许媒体同步。

仅当许多 RTP 流通过混频器时才使用贡献源（CSRC）。混频器通过从许多源接收数据并作为单一流发送来减小会议的带宽需求。例如，来自好几个并行演讲者的音频流能够被解码并重新编码为一个单一的音频流。在这种情况下，混频器将其自己作为同步源，并作为贡献源——演讲者的 SSRC 值贡献给当前分组。

5.4.3 控制协议

RTCP 提供了与多媒体应用数据流相关联的控制流，该控制流提供了三个主要功能：

1）关于应用和网络性能的反馈。

2）来自同一发送方的不同媒体流的关联与同步方法。

3）传输发送方身份以便显示在用户界面上的方法（例如，显示在图 5-21 中的 vat 界面）。

第一个功能对于拥塞的侦测和响应来说是有用的，例如，一些自适应速率的应用可能会通过性能数据来决定使用更好的压缩方法以降低拥塞，或者当拥塞比较小时发送高质量的流。回馈的性能数据在诊断网络故障时也可能用到。

你可能会认为 RTP 同步源（SSRC）ID 已经提供了第二个功能，但事实上没有。正如已经提到过的，在同一节点上的多个摄像头有不同的 SSRC 值，并且来自同一节点的音频和视频流并没有使用同一 SSRC 的需求。因为 SSRC 的值可能会出现冲突，因此改变一个流的 SSRC 值是必要的。为了处理这个问题，RTCP 使用了规范名（CNAME）的概念，它被指派给一个发送方，然后与使用 RTCP 机制的发送方所使用的 SSRC 值相关联。

关联两个流只是媒体同步问题的一部分。因为不同的流可能有完全不同的时钟（具有不同的时钟间隔，甚至不同的错误量），因此需要有一种方法来精确地同步各个流。RTCP 通过传递关联实际时间的定时信息和 RTP 分组所携带的依赖时钟速率的时间戳来解决该问题。

RTCP 定义了许多不同的分组类型，包括：

- 发送方报告，它使得活动的发送方能够产生会话来报告传输和接收的统计量。
- 接收方报告，接收方用来报告接收统计量。
- 源描述，承载 CNAME 和其他发送方描述信息。
- 特定应用的控制分组。

这些不同类型的 RTCP 分组通过下层协议来发送，正如之前提到过的，典型的是使用 UDP。多个 RTCP 分组可以被封装在一个下层协议的 PDU 中。在每个下层 PDU 中至少要有两个 RTCP 分组：一个是报告分组，另一个是源描述分组。也可能会包含其他分组，其大小只要满足下层协议的限制即可。

在进一步查看 RTCP 分组的内容之前，注意周期性发送控制流量的多播组的每个成员都有潜在的问题。除非采用一些步骤来进行限制，否则控制流量有可能成为带宽的重要消耗者。例如，在音频会议中，两个或三个发送方可能在任何情况下发送音频数据，无法对其进行控制，但是对每个发送控制流没有群体限制，在有成千上万参与者的会议中将是一个严重的问题。为了处理这个问题，RTCP 有一套机制，根据参与者增加的数量来测量报告频率。这些规则是比较复杂的，但基本目标是：将 RTCP 流量的总量限制为 RTP 数据流量的一个比较小的比例（典型值为 5%）。为了实现这个目标，参与者需要知道可能会使用多少数据带宽（例如，发送三个音频流的数量）以及参与者的数量。他们通过 RTP 来知道前者（通过本节末的会话管理），通过其他参与者的 RTCP 报告来知道后者。因为 RTCP 报告可能会以很低的速率发送，因此可以获得接收方当前容量的可能数量，但这就足够了。而且，在假设多数参与者喜欢看报告（例如，找出谁正在讲话）的基础上，推荐给活动的发送方分配较多的 RTCP 带宽。

一旦参与者决定了它能够消耗多少 RTCP 流量带宽，就会以适当的比率周期性地发送报告。发送方报告和接收方报告的区别仅在于前者包括一些关于发送方的额外信息。两种类型的报告都包含最近的报告周期内从所有源接收的数据的相关信息。

发送方报告中的额外信息包括：

- 包含产生报告的实际时间的时间戳。

- 相应于产生报告的时间的 RTP 时间戳。
- 发送方开始传输时分组和字节的累计计数。

注意前两个值用于来自同一源的不同媒体流的同步，即使这些流在其 RTP 数据流中使用了不同的时间间隔，因为它可以将实际时间转换为 RTP 时间戳。

发送方和接收方报告都包含每个源从最后的报告中获取的数据块，每个数据块包含源的如下统计量：

- SSRC。
- 自最后报告发送后丢失的该源的数据分组片段（通过比较接收的分组数量与期待的分组数量来计算得到，最后的值可由 RTP 序号来决定）。
- 自听到的第一个时间后从该源丢失的分组的总数量。
- 从该源接收到的最大序号（扩展到 32 位来避免序列号的回绕）。
- 该源预计的到达间隔空间（通过比较接收到的分组空间和传输中的分组空间来计算得到）。
- 通过该源的 RTCP 接收到的最后的确切时间戳。
- 从通过该源的 RTCP 接收到的最后的发送方报告开始的延迟。

正如你可能想象到的，信息的接收方能够知道关于会话状态的各种信息。特别是它们能知道是否别的接收方正在从一些发送方获得更好的质量，这可能指示需要进行资源预留，或者网络中存在问题。此外，如果发送方注意到许多接收方正在经历分组的大量丢失，它可能会降低发送速率或使用一种对丢失分组更有弹性的编码方案。

要考虑的 RTCP 的最后一个方面是源描述分组，该分组至少应包含发送方的 SSRC 和 CNAME。规范名源于这样的方法，对于产生需要同步的媒体流的所有应用程序（例如，同一用户独立产生音频和视频流），即使它们选择了不同的 SSRC 值，它们也将选择同样的 CNAME。这使得接收方能够识别出媒体流来自同一发送方。CNAME 的最常见格式是 user@ host，这里 host 是发送机器的完整域名。这样，运行在机器 cicada. cs. princeton. edu 上由用户名为 jdoe 的用户安装的应用程序将使用字符串 jdoe@ cicada. cs. princeton. edu 作为其 CNAME。用于这种表示法中的大量且可变长度的字符串对于 SSRC 来说是不好的选择，因为 SSRC 随着每个数据分组被发送，并且必须实时处理。允许 CNAME 与周期性 RTCP 报文中的 SSRC 值关联，使得 SSRC 有一个紧凑且有效的格式。

在源描述分组中可能会包含其他内容，例如用户的真实名字和电子邮件地址。这些信息用在用户界面显示和联系参与者中，但是对于 RTP 操作来说，它们没有 CNAME 重要。

像 TCP 一样，RTP 和 RTCP 是相当复杂的一对协议。这种复杂性很大一部分来自使应用程序设计者的生活更轻松的愿望。因为应用程序是多种多样的，所以设计传输协议的挑战是让这个协议更通用以满足各种应用的不同需求，而并不是设计出无法实现的协议。在这方面 RTP 协议成功构造了互联网上众多实时多媒体通信的基础。

5.5 小结

本章描述了四种完全不同的端到端协议。第一类协议是一个简单的多路分解协议，以 UDP 为代表。所有这样的协议所做的事情是将报文分发给基于端口号的相应的应用程序进程。它并没有以任何方式增强底层网络的尽力而为传输服务模型，或者换个方式说，它为应用程序提供了一个不可靠的、无连接的数据报服务。

第二类是一个可靠的字节流协议，这类协议的具体例子是 TCP。这样的协议面临的问题是恢复网络丢失的报文，按照发送的顺序递交报文，以及允许接收方对发送方进行流量控制。TCP 使用带有通知窗口的基本滑动窗口算法来实现这些功能。对这个协议另一个要注意的事项是其精确的超时/重传机制的重要性。有趣的是，虽然 TCP 是单个协议，但它至少采用了五种不同的算法——滑动窗口、Nagle、三次握手、Karn/Partridge 和 Jacobson/Karels——每种算法对于端到端协议都是有价值的。

第三类传输协议是请求/应答协议，它是 RPC 的基础。这类协议必须将请求分发给正确的远程过程，并将应答与相应的请求相匹配。它们可能提供可靠性，如最多一次语义，也可能通过报文的分段和重组来支持大尺寸的报文。

最后一类传输协议针对的是涉及多媒体数据（例如音频和视频）并且对实时传输有需求的应用程序类，这样的传输协议需要考虑单个媒体流的时间信息以及多媒体流的同步。它也要给上层（例如，应用层）提供关于数据丢失的信息（因为通常没有足够的时间来重传丢失的分组），以便使用合适的针对特定应用的恢复方法和避免拥塞的方法。已经被开发出来的能够满足这些需要的协议是 RTP，它包括一个称为 RTCP 的控制协议。

接下来会发生什么：传输协议多样性

读完本章应该清楚的是，设计传输协议是一件复杂的事情。正如我们看到的，首先正确实现一个传输协议是足够困难的，但变化的环境使这一问题更加复杂。有些变化并不是太难预料（例如，网络不断变快），但有时一类新应用的出现将改变对传输层服务的要求。问题在于找到适应这些变化的方法。有时现有协议可以进行调整，如为了处理新问题 TCP 选项的不断演化。有时又可以创建新的传输协议，如 RTP 和 SCTP。重用现有协议还是创建新协议，这是一个开放的问题。

除了对现有协议稍作修改，以适应不断变化的网络环境或新应用需求，有时我们也在现有协议下调整应用程序来适应变化。例如，需要 RPC 语义的应用程序在 RPC 协议下工作得最好，而对应用程序稍作修改也能在 TCP 协议下工作，只不过效率有一些损失。在网页应用程序需要某些 RPC 选项的今天，这种情况并不少见。网络中 TCP 无处不在，它能胜任各种具有挑战性的环境，如穿过 NAT 设备和防火墙。因此，有时在 TCP 上构建并运行仿 RPC 的机制更为简单，即使严格来说它不是最有效的方法。

同样，似乎 RTP 应该是传输视频流的首选协议，它也仍然被广泛使用着，但使用 TCP（和 HTTP）来提供流媒体视频内容的趋势也很强烈。采用这种方式来传输娱乐视频（如在网页浏览器上观看的电视节目和电影）的做法渐渐流行。说该方法将走向终点还言之尚早，但不难想象 TCP 是未来用于视频传输的主要传输协议。

回顾第 1 章讨论的互联网“沙漏”结构，很明显，传输协议的多样性正符合互联网设计师的预期。但是，现如今 TCP 几乎成为沙漏中“腰”的一部分，因为 TCP 提供无所不在的服务，而大多数应用程序又很依赖这些服务。

扩展阅读

TCP 无疑是一个复杂的协议，实际上在本章还有一些细节没有阐述。因此本章的推荐阅读中包括 TCP 的原始规范。推荐这个规范的目的不是为了补充缺少的细节，而是向你展示一个真正好的协议规范是什么样子。由 Birrell 和 Nelson 写的论文是 RPC 的开山之

作。Clark 和 Tennenhouse 的论文是关于协议体系结构的，介绍对设计 RTP 产生了灵感的应用层框架（Application Layer Framing）的概念，这篇论文在设计随应用需求而改变的协议方面非常有见地。

- USC-ISI. Transmission Control Protocol. *Request for Comments* 793, September 1981.
- Birrell, A., and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems* 2 (1): 39-59, February 1984.
- Clark, D., and D. Tennenhouse. Architectural considerations for a new generation of protocols. *Proceedings of the SIGCOMM'90 Symposium*, pages 200-208, September 1990.

除了协议规范，关于 TCP 最完整的描述，包括它在 Unix 中的实现，可以在 Stevens 的著作［Ste94b］和［SW95］中找到。在 Comer 和 Stevens 的 TCP/IP 系列丛书的第 3 卷中，描述了在 TCP 和 UDP 上用 Posix 套接字接口［CS00］、Windows 套接字接口［CS97］和 BSD 套接字接口［CS96］怎样编写客户端/服务器应用程序。在帮助概述［OY02］中描述了 SCTP 作为可靠传输协议与 TCP 在设计方面的不同，并在［Ste07］中进行了详细说明。

多篇论文非常详细地评估了不同传输协议的性能。例如，Clark 等人的文章［CJRS89］测量了 TCP 层的处理开销，Mosberger 等人的文章［MPBO96］探讨了协议处理开销的局限，Thekkath 和 Levy 的文章［TL93］以及 Schroeder 和 Burrows 的文章［SB89］都非常详细地考察了 RPC 的性能。

原有的 TCP 超时计算在 TCP 规范中有描述（见上面），Karn/Partridge 算法在［KP91］中描述，Jacobson/Karels 算法在［Jac88］中描述。由 Jacobson 等［JBB92］定义了对 TCP 的扩展，而 O'Malley 和 Peterson［OP91］论述了以这样的方式扩展 TCP 不是解决问题的正确方法。

有多种分布式操作系统已经定义了它们自己的 RPC 协议。著名的例子包括由 Cheriton 和 Zwaenepoel［CZ85］描述的 V 系统，由 Ousterhout 等人［OCD+88］描述的 Sprite 系统，以及由 Mullender［Mul90］描述的 Amoeba 系统。

在 RFC 3550［SCFJ03］中描述了 RTP，在众多的其他 RFC（例如，RFC 3551［SC03］）中描述了使用 RTP 的各种应用的轮廓。McCanne 和 Jacobson［MJ95］描述了 vic，这是使用 RTP 的一种应用。

最后，下列时常更新的参考资料提供了大量关于传输协议和其他协议的信息：

- http://www.iana.org/protocols/：包括所有选项、常量、端口号等各种已定义的互联网协议的信息。除此之外，你还可以在这里找到 TCP 头标志、TCP 选项，以及 TCP 和 UDP 的知名端口号。

习题

1. 如果一个 UDP 数据报从主机 A 的端口 P 发送到主机 B 的端口 Q，但主机 B 没有监听端口 Q 的进程，那么主机 B 返回一个 ICMP 端口不可达的报文给 A。与所有 ICMP 报文一样，这个报文作为一个整体发给主机 A，而不是 A 上的端口 P。

(a) 给出何时一个应用程序可能希望接收到这样的 ICMP 报文的一个例子。

(b) 指出在你选择的操作系统上，应用程序要接收到这样的报文需要做些什么。

(c) 为什么将这样的报文直接发送回主机 A 上的原始端口 P 不是一个好方法?

2. 考虑一个用于请求文件的基于 UDP 的简单协议（基于不太严格的普通文件传输协议或 TFTP）。客户端发送一个初始文件请求，服务器用第一个数据分组回答（如果文件可以被发送）。然后客户端和服务器使用停止-等待传输机制继续文件传输。

(a) 描述客户端请求一个文件却得到另外一个文件的情况，可以允许客户应用程序突然退出并在同一端口重新开始。

(b) 对协议提出改进，使得这种情况的发生率大为减少。

3. 设计一个用于从服务器检索文件的基于 UDP 的简单协议。不必提供认证，可以使用数据的停止-等待传输。你的协议应该解决下列问题：

(a) 第一个分组的拷贝不应再建立一个"连接"。

(b) 最后一个 ACK 的丢失不应使服务器怀疑传输是否成功。

(c) 来自过去连接的迟到分组不应被解释为当前连接的一部分。

4. 本章解释了 TCP 连接断开期间状态转换的三种顺序。还可能有第四种顺序，即它经过另外一个弧（没有显示在图 5-7 中），从 FIN_WAIT_1 到 TIME_WAIT 且标有 FIN+ACK/ACK。解释导致这第四种状态转换顺序的环境。

5. 关闭一个 TCP 连接时，为什么从 LAST_ACK 到 CLOSED 的转换不必有两个段生存期（two-segment-lifetime）的超时?

6. 接收 0 通知窗口的一个 TCP 连接的发送方定期检查接收方，以便发现窗口什么时候变为非 0。为什么当接收方负责报告通知窗口变为非 0（即发送方不检查）时，它需要一个额外的定时器?

7. 阅读 Unix 或 Windows 应用程序 netstat 的使用手册。使用 netstat 查看本地 TCP 的连接状态。找出关闭连接在 TIME_WAIT 中花费的时间。

8. TCP 首部的序号字段长 32 位，足以处理 40 亿字节的数据。即使在一个连接上有许多序号的数据从未被传送过，为什么序号仍旧可能从 $2^{32}-1$ 回绕到 0?

9. 假设你受雇设计一个使用滑动窗口的可靠的字节流协议（如 TCP）。这个协议将运行在 1Gbps 的网络上。网络的 RTT 是 100ms，且数据段的最大生存期是 30s。

(a) 协议首部的 AdvertisedWindow 字段和 SequenceNum 字段中应包含多少位?

(b) 怎样确定上述数值，哪个值可能不太确定?

10. 假设你正在设计一个使用滑动窗口的可靠字节流协议（如 TCP）。这个协议运行在 1Gbps 的网络上。网络的 RTT 是 140ms，且最大段生存期是 60s。协议首部的 AdvertisedWindow 字段和 SequenceNum 字段各应包含多少位?

11. 假设一台主机想要通过发送分组并测量接收的百分比的方法建立一个可靠的链路。例如由路由器来做这件事。阐述在 TCP 连接上做这件事的困难。

12. 假设 TCP 运行在一个 1Gbps 的链路上。

(a) 假设 TCP 能持续利用全部带宽，TCP 序号完全回绕最快需要多长时间?

(b) 如果在上面算出的回绕时间内，一个附加的 32 位时间戳字段增量 1 000 次。这个时标回绕需要多长时间?

13. 假设 TCP 运行在一个 40Gbps 的 STS-768 链路上。

(a) 假设 TCP 能持续利用全部带宽，TCP 序号完全回绕需要多长时间?

(b) 如果在你上面算出的回绕时间内，一个附加的 32 位时标字段增量 1 000 次。这个时标回绕需要多长时间?

14. 如果主机 A 从同一端口接收远程主机 B 的两个 SYN 分组，第二个 SYN 分组可能是前一个 SYN 分组的重传，或是在 B 崩溃并重启动后一个全新的连接请求。

(a) 描述主机 A 看到的这两种情况的区别。

(b) 给出 TCP 层在接收一个 SYN 分组时需要做的事情的算法描述。考虑上面的重传/新连接的情况。

以及不监听目标端口的可能性。

15. 假设 x 和 y 是两个 TCP 序号。写一个函数确定 x 是在 y 之前到达（RFC 793 的表示方法是 "$x<y$"）还是在 y 之后到达，你的解决方案应在序号回绕的情况下也有效。

16. 假设套接字 A 和 B 间有一个空闲的 TCP 连接。第三方已经窃听并知道两端的当前序号。

(a) 假设第三方向 A 发送一个伪造的看起来发自 B 的分组，并带有 100 字节的新数据。此时会发生什么情况？（提示：查阅 RFC 793，了解当接收到一个不是 "可接受的 ACK" 的 ACK 时，TCP 怎么处理。）

(b) 假设第三方向每一端都发送一个伪造的看起来发自另一端的分组。这时会发生什么情况？如果 A 后来发送 200 字节的数据给 B 会发生什么情况？

17. 假设 A 方经无线网连接到因特网上，使用 DHCP 来分配 IP 地址。A 打开多个 Telnet 连接（使用 TCP），随后断开无线网连接。接着 B 方拨号进入并被赋予和 A 相同的 IP 地址。假设 B 能够猜到 A 已连接到什么主机，描述一个探测序列，能够使 B 获得足够的状态信息以继续 A 的连接。

18. 诊断程序一般能够用来记录对某个（主机，端口）的每个 TCP 连接的前 100 个字节。简述对每个接收到的 TCP 分组 P 必须做什么，才能确定它是否包含对主机 HOST，端口 PORT 的连接的前 100 个字节数据。假设 IP 的首部是 P.IPHEAD，TCP 首部是 P.TCPHEAD，且首部字段命名如图 3-16 和图 5-4 所示。（提示：为得到初始化序号（ISN），你必须检查每个分组的 SYN 位的设置。忽略序号最终被重用的事实。）

19. 如果源地址为 B 的分组到达主机 A，那么它将很容易被任何第三方主机 C 伪造。但是如果 A 接受来自 B 的 TCP 连接，那么在三次握手过程中，A 向 B 的地址发送 ISNA 并且接收它的一个确认。如果 C 不在能窃听 ISNA 的位置，则 C 不能伪造 B 的响应。

然而，选择 ISNA 的算法确实给其他不相关主机提供了一个猜测它的平等机会。尤其是 A 基于连接时的时钟值选择 ISNA。RFC 793 指出这个时钟值每 4ms 增加 1，通用的 Berkeley 实现曾对此简化为每秒钟一次增加 250 000（或 256 000）。

(a) 给出这个简化的每秒增加一次的实现，解释任意主机 C 至少在打开 TCP 时如何假扮 B。你可以假设 B 不对 A 因受骗发送给它的 SYN + ACK 分组做响应。

(b) 假设实际的 RTT 可以估计为 40ms 以内，使用非简化的 "每 4ms 增加一次" 的 TCP 实现，你期望实现 (a) 中策略要尝试多少次？

20. 在绝大多数 TCP 的实现中都内置有 Nagle 算法，它要求发送方不发送不满整段的数据（即使是被 PUSH 操作发送的），直到积累够一个完整的数据段或最近未确认的 ACK 到达。

(a) 假设通过 RTT 为 4.1s 的 TCP 连接发送字母串 abcdefghi，每秒发送一个字母。请画出时间线表明何时发送每个分组，以及其中包含什么。

(b) 如果在一条全双工 Telnet 连接上发送以上内容，用户会看到什么？

(c) 假设通过这条连接发送鼠标位置的变化。若在每个 RTT 期间发送多次鼠标位置变化，那么在用 Nagle 算法和不用 Nagle 算法的情况下，用户将会看到鼠标是如何移动的？

21. 假设客户端 C 不断地通过 TCP 连接服务器 S 上的给定端口，并且每次由 C 起动关闭。

(a) 在 C 使它的所有可用端口处于 TIME_WAIT 状态之前，每秒钟可以建立多少个 TCP 连接？假设客户端的短期端口的范围为 1024~5119，并假设 TIME_WAIT 持续时间是 60s。

(b) Berkeley 的 TCP 实现通常允许如果旧连接实例使用的最高序号比新实例使用的 ISN 小，那么在 TIME_WAIT 到期之前，一个处于 TIME_WAIT 状态的套接字可以再次打开，这解决了旧数据被当作新数据接收的问题。但是，TIME_WAIT 也用于处理迟到的最后 FIN 的目的。这样的实现必须怎么做才能解决这个问题，并仍然完全满足 TCP 的要求，即 FIN 可以在一个连接的 TIME_WAIT 状态收到同一响应之前或期间的任何时刻被发送？

22. 解释为什么如果服务器先于客户端启动关闭，则 TIME_WAIT 会是很严重的问题。描述一种会发生这种问题的情况。

23. Karn 和 Partridge 提出指数增长超时值的理由是什么？特别要说明为什么不用线性（或更慢）增长？

★ 24. Jacobson/Karels 算法将 TimeOut 设为 4 倍平均偏差而不是平均值。假设单个分组的往返时间遵循统计正态分布，即 4 倍平均偏差都是 σ 标准偏差。使用统计表计算一个分组会超时到达的概率是多少。

25. 假设一个窗口大小为 1 的 TCP 连接，每隔一个分组丢弃一个分组。到达的分组的 RTT＝1s。考虑下面两种情况各会发生什么？对 TimeOut 会发生什么？

(a) 收到分组后，从停止的地方开始，假设 EstimatedRTT 被初始化为预定的超时值，而且 TimeOut 是它的 2 倍。

(b) 收到分组后，重新将 TimeOut 初始化为上次超时间隔使用的指数退避的值。

以下 4 道习题中的计算用电子表格是很简单的。

26. 假设在 TCP 的自适应重传机制中，EstimatedRTT 在某时刻为 4.0s，其后测得的 RTT 全为 1.0s。按照 Jacobson/Karels 算法的计算，TimeOut 要多长时间才会变得小于 4.0s？假设请你给出一个合理的 Deviation 初始值，你的答案对这个选择的敏感度如何？使用 $\delta=1/8$。

√ 27. 假设在 TCP 的自适应重传机制中，EstimatedRTT 在某时刻为 90，其后测得的 RTT 全为 200。按照 Jacobson/Karels 算法的计算，TimeOut 要多长时间才会变得小于 300？假设 Deviation 初始值为 25，使用 $\delta=1/8$。

28. 假设 TCP 测得的 RTT 值除了每第 N 个 RTT 是 4.0s 以外，其他都是 1.0s。最大的 N 大约是多少才不会在稳定状态（即 Jacobson/Karels 算法中的 TimeOut 保持大于 4.0s）时发生超时？使用 $\delta=1/8$。

29. 假设 TCP 正在测量的 RTT 一直是 1.0s，平均偏差是 0.1s，突然 RTT 跳到 5.0s，没有偏差。比较原始算法和 Jacobson/Karels 算法在计算 TimeOut 时有什么不同？特别地，每种算法各遇到多少次超时？计算得到的最大的 TimeOut 是多少？使用 $\delta=1/8$。

30. 假设一个 TCP 分段被发送多次，我们取 SampleRTT 为最初发送和 ACK 之间的时间，如图 5-10a 所示。请说明如果一个 TCP 连接有一个分组大小的窗口，它每隔一个分组就丢弃一个分组（即每个分组传送两次），那么 EstimatedRTT 会增加到无穷大。假设 TimeOut＝EstimatedRTT，正文中描述的两种算法会把 TimeOut 设置得更大。（提示：EstimatedRTT＝EstimatedRTT＋β×(SampleRTT－EstimatedRTT)。）

31. 假设当一个 TCP 分段被发送多次时，取 SampleRTT 为最近一次传送和 ACK 之间的时间，如图 5-10b 所示。假设 TimeOut＝2×EstimatedRTT。描述一种情况，其中没有分组丢失，但 EstimatedRTT 收敛到真正的 RTT 的 1/3，并画出最后稳定状态的示意图。（提示：真正的 RTT 是以一个突然跳跃开始，这个跳跃刚好超过建立的 TimeOut 值。）

32. 查阅 RFC 793，找出如果一个 FIN 或 RST 到达但序号不是 NextByteExpected 时，TCP 应该如何响应？考虑序号在接收窗口内和不在接收窗口内的两种情况。

33. TIME_WAIT 的作用之一是处理很晚到达的第一个连接实例的数据分组被当作第二个实例的数据收下的情况。

(a) 对于在没有 TIME_WAIT 时发生的这种情况，解释为什么连接的主机在延迟的分组被发出以后但未被传递之前不得不顺序交换几个分组。

(b) 提出一种可以导致这种延迟传递的网络情况。

34. 提出一种对 TCP 的扩展，使得连接的一端可以将它这一端转交给第三台主机；也就是说，如果原来 A 与 B 连接，A 将它的连接转交给 C，那么这以后 C 将与 B 连接，而 A 将不再保持这个连接。请说明在 TCP 状态转换图中所需的新状态和新转换，以及用到的任何新分组类型。可以假设各方都会理解这个新选项。A 在转交之后应该进入什么状态？

35. TCP 的同时打开性质是极少用到的。

(a) 改变 TCP，使之禁用同时打开。指出状态图（如果必要也包括不在图中的事件响应）应做哪些改变。

(b) TCP 可以禁用同时关闭吗？

(c) 改变 TCP，使两台主机同时交换 SYN 会导致两个独立的连接。指出这要求状态图做什么改变，同时首部有必要做什么改变。注意现在这意味着在一个给定的（主机，端口）对上存在多个连接。(可以查阅 RFC 1122 第 87 页的第一个“讨论”项。)

36. TCP 是一个非常对称的协议，但是客户端/服务器模型不是对称的。考虑一个非对称的 TCP 式的协议，其中只有服务器方被赋予一个可被应用层看到的端口号。客户端的套接字只是能连到服务器端口的抽象。

(a) 提出一种首部数据和连接的语义以支持这种情况。你将用什么代替客户端的端口号？

(b) 现在 TIME_WAIT 采取什么形式？通过编程接口如何看到这一点？假设客户端套接字现在可以任意多次地重复连接到给定的服务器端口和许可的资源上。

(c) 查阅 rsh/rlogin 协议。为什么上面的设想将导致其不可用？

37. 下面的习题与 TCP 的状态 FIN_WAIT_2 有关（见图 5-7）。

(a) 描述客户端怎样使一个适当的服务器无限期地处于状态 FIN_WAIT_2。对这种情况，服务器的协议需要有什么特性？

(b) 在某种现有的服务器上试验这种情况。可以写一个桩客户程序，或者用一个现成的可以连到任意一个端口的 Telnet 客户程序。使用 netstat 工具验证服务器处于状态 FIN_WAIT_2。

38. 在 RFC 1122 中关于 TCP 有以下陈述：

“一台主机可以实现一个“半双工”的 TCP 关闭顺序，因此一个已经调用 CLOSE 的应用程序不能继续从连接读数据。如果这样一台主机在 TCP 中仍有接收到的数据时发出 CLOSE 命令，或者调用 CLOSE 后 TCP 仍收到新数据，那么它的 TCP 应该发出一个 RST 说明数据丢失。”

描述包含以上陈述的一种情景，其中被正在关闭的主机发送（而不是发向它）的数据丢失。你可以假设远程主机在收到 RST 时会丢弃已接收到缓冲区但还未被读出的所有数据。

39. 当 TCP 发出（SYN，SequenceNum＝x）或（FIN，SequenceNum＝x）时，随后的 ACK 有 Acknowledgment＝$x+1$；也就是说，SYN 和 FIN 在序号空间中各占用一个单位。这样有必要吗？如果有必要，请给出一个相应的 Acknowledgment 是 x 而非 $x+1$ 时会发生混乱的例子。如果没有必要，请解释为什么。

40. 从 RFC 793 中查出 TCP 首部选项的一般格式。

(a) 概述一种策略，它可以扩展选项可用的空间以超出当前 44 字节的限制。

(b) 提出一种 TCP 扩展，允许选项的发送方说明在接收方不理解选项含义时应做什么。列出这样几种可能有用的接收方动作，并试着为每个动作的应用给出一个例子。

41. TCP 的首部没有引导 ID 字段。当 TCP 连接的一端崩溃并重启，然后发送一个具有先前使用过的 ID 的报文时，为什么不会出现问题？

42. 假设使用提供零次或多次语义的不可靠 RPC 协议实现远程文件系统安装。如果收到报文应答，就改进为至少一次语义。定义 read(n) 返回指定的第 n 块，而不是按顺序的下一块；这种方式读一次相当于读两次，而且至少一次语义也就等同于正好一次语义。

(a) 对于文件系统的哪些操作，至少一次语义和正好一次语义没有区别？考虑 open、create、write、seek、opendir、readdir、mkdir、delete (aka unlink) 和 rmdir。

(b) 对余下的操作，哪些能改变语义使至少一次语义和正好一次语义等价？文件系统的什么操作不能与至少一次语义相容？

(c) 假设 rmdir 系统调用的语义是：如果给定的目录存在则删除它，否则什么也不做。那么怎样写一个能区别这两种情况的删除目录的程序？

43. 基于 RPC 的 NFS 远程文件系统有时被认为具有比期望的 write 慢的性能。在 NFS 中，服务器的 RPC 对客户端 write 请求的应答意味着数据被物理写入服务器的磁盘，而不只是放入队列中。

(a) 如果客户端通过一条单独的逻辑 CHAN 信道发送所有 write 请求（即使有无限带宽），解释我们可以预料到的瓶颈，并解释为什么使用一个信道池会有所帮助。(提示：你需要知道一点磁盘控

制器的知识。）

(b) 假设服务器的应答只意味着数据被放入磁盘队列。解释为什么不使用本地磁盘会导致数据丢失。注意，忽略数据刚进队列系统就崩溃的情况，因为在这种情况下用本地磁盘也会引起数据丢失。

(c) 一个选择是让服务器立即确认 write 请求，稍后发送它自己独立的请求确认物理写操作的完成。请提出能达到相同效果的不同的 RPC 语义，但是用一个逻辑请求/应答。

44. 考虑一个使用包含信道抽象和引导 ID 的 RPC 机制的客户端和服务器。

(a) 给出一个涉及服务器重启的方案，在该方案中 RPC 请求被客户端发送两次，被服务器执行两次，但只有一个 ACK。

(b) 客户端如何意识到事件的发生？客户端能确信事件已经发生了吗？

45. 假设 RPC 请求正好形成了“磁盘块 N 中字段 X 的值以 10%递增”，对正在执行的服务器给定一种机制，即使在操作过程中服务器发生崩溃也能保证一个到达的请求恰好执行一次。假设单独的磁盘块写操作完成或块没有改变，也可以假设一些指派的“撤销日志”块是可用的。你的机制应该包括 RPC 服务器在重启时会有什么操作。

46. 考虑一个向服务器发送请求的 SunRPC 客户端。

(a) 在什么情况下，客户端能够确信其请求恰好被执行了一次？

(b) 假设希望把至多一次语义加到 SunRPC 中，应做什么改变？解释为什么向现有的首部添加一个或多个字段是不够的。

47. 假设用 TCP 完成一个 RPC 协议的底层传输，每个 TCP 连接传输一个连续的请求和应答流。TCP 需要有以下什么字段（如果有的话）：

(a) 信道 ID。　　(b) 报文 ID。

(c) 引导 ID。　　(d) 请求报文类型。

(e) 应答报文类型。　　(f) 确认报文类型。

(g)“你仍活跃着吗”报文类型。

其中哪些是上层 RPC 协议必须提供的？存在某些相似的隐式确认吗？

48. 写一个测试程序，用套接字接口在一对被某种 LAN（例如以太网、802.11）连接的 UNIX 工作站间发送报文。用这个测试程序完成以下实验。

(a) 测量不同报文大小情况下（例如，1 字节，100 字节，200 字节，…，1 000 字节）TCP 和 UDP 的往返时延。

(b) 测量 1KB，2KB，3KB，…，32KB 报文的 TCP 和 UDP 的吞吐量。绘出测得的吞吐量和报文大小的函数关系图。

(c) 测量 TCP 从一台主机向另一台主机发送 1MB 数据的吞吐量。通过循环发送一定大小的报文来实现，例如每次发送 1KB，重复 1 024 次，用不同大小的报文重复这个实验，并画出结果的图形。

49. 找出 RTP 应用程序可以完成以下功能的情形：

- 在需要不同时间戳但本质上是同一时间内发送多个分组。
- 在需要相同时间戳的不同时间内发送分组。

结果，在一些情况下，RTP 时间戳必须由应用程序来提供（至少是间接提供）。（提示：考虑发送速率和播放速率可能不匹配的情况。）

50. 帧时间或声音样本时间的时间戳时钟计量单位应该有最小的分辨能力，以确保精确的播放，但时间单位通常是相当小的，这样做的目的是什么？

51. 假设想要从接收方返回 RTCP 报告，其总量不超过流出的 RTP 流的 5%。如果每个报告是 84 字节，RTP 流量是 320kbps，有 1 000 个接收容器，那么接收方多久能够获得一个报告？如果有 10 000 个接收容器呢？

52. RFC 3550 指明接收方 RTCP 报告间的时间间隔包括一个随机因素以避免所有接收方在同一时间发送。如果所有接收方在其应答时间间隔的 5%的子间隔内发送，到达的上游 RTCP 流量将与下游的

RTCP 流量竞争。

(a) 视频接收方在发送报告时可以适当地等待发送，直到处理并显示一帧的高优先权任务被完成了。这可能意味着它们的 RTCP 传输在帧边界是同步的。这是值得重点关注的吗？

(b) 有 10 个接收方，它们都在一个特定的 5%子间隔内发送的概率是多少？

(c) 有 10 个接收方，它们中半数都在一个特定的 5%子间隔内发送的概率是多少？接收方增加 20 倍后，它们中半数都在同样的 5%子间隔内发送的概率是多少？（提示：在 10 个接收方中选择 5 个有多少种方法？）

53. 服务器如何处理分组丢失速率数据和接收方报告中的抖动数据？

54. 提出一种决定何时报告 RTP 分组丢失的机制。将该机制与 5.2.6 节中的 TCP 自适应重传机制进行比较。

第6章
Computer Networks, A Systems Approach

拥塞控制与资源分配

造物主给你美貌，也给你美好的品格。

——威廉·莎士比亚

问题：分配资源

到目前为止，为了理解数据如何在异构网络的进程之间传输，我们已经学习了组成网络协议层次结构的足够多的层。现在我们转到一个贯穿整个协议栈的问题：当用户竞争资源时，如何有效和公平地分配这些资源。共享资源包括链路带宽以及路由器或交换机上的缓冲区，所有分组都在缓冲区中排队等待发送。分组在路由器中争用（contend）链路的使用权，链路上每个参与争用的分组都被放在队列中等待通过链路顺序发送。当过多的分组争用同一条链路时，队列就会溢出，分组将不得不被丢弃。如果常常发生分组丢失的情况，就称作网络拥塞（congested）。大多数的网络提供拥塞控制（congestion-control）机制来处理这种情况。

拥塞控制和资源分配是同一事物的两个方面。一方面，如果网络在资源分配方面采取积极的措施（例如，在一个特定的时间周期内调度某一条虚电路使用指定的物理链路），就可以避免拥塞，从而没有必要进行拥塞控制。然而，以任意的精度来分配网络资源都是相当困难的，因为我们讨论的资源分布在整个网络，需要调度连接着一系列路由器的多条链路。另一方面，也可以允许发送方想发多少数据就发多少数据，然后当拥塞发生时将其恢复。这种方法比较容易实现，但它也具有一定的破坏性，因为在拥塞控制前网络可能会丢弃许多分组，并且在网络恰好发生拥塞时，即资源相对于需求来说变得缺乏时，竞争用户会最强烈地感觉到对资源分配的急切需求。此外也有一些折衷的解决方案，先做一些不太精确的分配决定，当然拥塞依然会发生，因此还需要采取一些从拥塞中恢复的措施。把这种混合解决方案称为拥塞控制还是资源分配都无关紧要，从某种意义上来说，它是二者兼而有之。

拥塞控制和资源分配与主机和网络设备（如路由器）都有关。在网络设备中可用各种排队规则控制分组传送的顺序和丢弃哪些分组。排队规则也可以隔离通信量，使一个用户的分组不会过分地影响另一个用户的分组。在端主机上，拥塞控制机制协调源端发送分组的速度。拥塞控制机制首先要尽力采取措施避免拥塞的出现，拥塞一旦发生，则要尽快消除拥塞。

本章首先概述拥塞控制和资源分配。然后讨论网络内部路由器上可实现的不同排队规则，接着描述一个在主机上由 TCP 提供的拥塞控制算法。6.4 节探讨与路由器和主机都相关的各种技术，目的是在拥塞成为一个问题之前予以避免。本章最后考察服务质量（quality of service）涉及的广泛领域。我们考虑不同应用对网络中资源分配的不同需求级别，同时描述应用请求资源和网络满足请求的各种方法。

6.1 资源分配问题

资源分配和拥塞控制是复杂的问题，从设计第一个网络以来它们就一直是许多研究的主题。如今它们依然是活跃的研究领域。使这些问题如此复杂的因素之一是，它们不能被划分

到协议层次结构的某一层上。资源分配有一部分在网络中的路由器、交换机和链路上实现，另一部分在端主机上运行的传输层协议中实现。端系统可能使用信令协议向网络节点发送资源请求，网络节点将可用资源的信息反馈给它。本章的一个主要目标是定义一个框架，使得这些机制在其中是可以理解的，同时对这些机制中有代表性的实例给出有关细节。

在继续讨论之前，我们先解释几个术语的含义。资源分配（resource allocation）是指一个过程，网络设备通过它来尽量满足应用对网络资源的竞争需求，这里的资源主要指链路带宽和路由器或交换机上的缓冲区空间。当然，它常常不能满足所有需求，即一些用户或应用获得的网络资源可能比它们需要的少。部分资源分配问题在于何时说不以及对谁说不。

我们用拥塞控制（congestion control）这个术语来描述网络节点为防止和响应过载状态所做的工作。由于拥塞对每个人来说一般都不是好事情，所以当务之急是使拥塞下降，或者首先防止它的出现。说服少数主机停止发送从而改善其他每台主机的状况，就可以简单地实现拥塞控制。但是对于拥塞控制机制而言，或许更注重公平的观念，换言之，它们试图让所有用户分担困难，而不是只让几个用户承担很大的困难。于是我们看到现在许多拥塞控制机制中都引入了资源分配的概念。

理解流量控制和拥塞控制的区别也是非常重要的。正如我们在 2.5 节看到的，流量控制是为了使快速的发送方不能超出慢速的接收方。与之相反，拥塞控制是考虑到网络某些点上资源的缺乏，因而阻止一组发送方向网络中发送过多的数据。这两个概念经常混淆，然而正如我们将看到的，它们也会共用某些机制。

6.1.1 网络模型

首先我们定义网络体系结构的三个主要特性。在很大程度上，这是对前几章提到的有关资源分配问题的总结。

1. 分组交换网

我们来考虑在由多链路和交换机（或路由器）组成的分组交换网（或互联网）中的资源分配。本章描述的大多数机制是为因特网上的使用而设计的，因此最初的定义都是用路由器而不是交换机，在我们的讨论中都用路由器（router）这一术语。不论是网络还是互联网，问题基本上是相同的。

在这样的环境中，一个给定源可能在立即输出链路上有足够的容量来发送分组，但到了网络中间的某处，它的分组会遇到一条正被许多不同通信源使用的链路。图 6-1 描述了这种情况：两条高速链路正传输给一条低速链路。这与像以太网和无线网这样的共享访问网络形成对比，在共享访问网络中，源可以直接观察网络上的通信量并决定是否发送一个分组。我们已经见过用于在共享访问网络上分配带宽的算法（第 2 章）。从某种意义上说，这些访问控制算法与交换网中的拥塞控制算法类似。

结论 注意拥塞控制与路由选择不同。虽然对拥塞的链路确实可由路由传播协议赋予一个大的边权值，其结果是路由器将绕过该链路，但是“绕过”拥塞链路基本没有解决拥塞的问题。为清楚这一点，我们只需看图 6-1 所示的简单网络，图中所有通信量都必须流经同一路由器到达目的地。尽管这是一个极端的例子，但某台路由器不能被绕过的情况是常见的。[⊖] 这台路由器会变成拥塞的，而没有任

⊖ Internet 上的路由是非常复杂的，最好的情况就是获得合理的路由方向，且没有形成路由回路。绕过拥塞的路由相当于是蛋糕上的糖衣。

何路由机制可以解决这个问题。这种拥塞的路由器有时称为瓶颈（bottleneck）路由器。

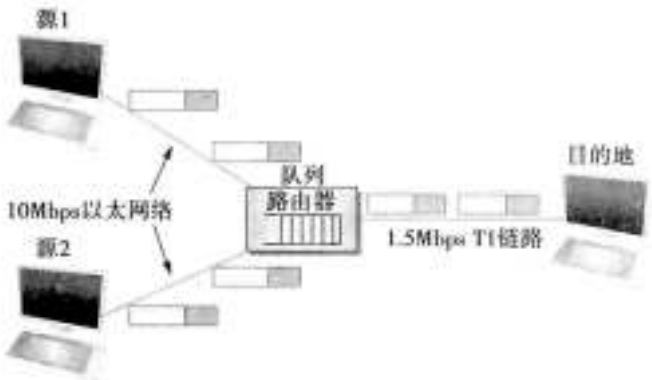


图 6-1 一个潜在的瓶颈路由器

2. 无连接流

在我们讨论的大多数情况下，假设网络基本上是无连接的，任何面向连接的服务均在端主机上运行的传输协议中实现（我们稍后解释“基本上”的含义）。这正是因特网的模型，其中 IP 提供无连接数据报传输服务，而 TCP 实现端到端的连接抽象。请注意这一假设不包括像 X.25（见 3.1.2 节）这样的虚电路网络。在建立虚电路时，一个连接建立报文将穿越网络。这个建立报文为每台路由器的连接保留一系列缓冲区，从而提供一种拥塞控制形式——只有在每台路由器上都能分配足够的缓冲区时，连接才能建立。这种方法的主要缺点是导致资源的不充分利用——为某个特定的电路保留的缓冲区即使在空闲时也不能被其他传输使用。本章重点讨论的是在一个互联网中使用的资源分配方法，因而我们主要关注无连接网络。

因为我们将网络分类为无连接和面向连接时是过于严格的，在两者之间有一段灰色区域，所以我们需要界定*无连接*（connection-less）这一术语。特别是无连接网络过强地假定所有数据报是完全独立的。虽然数据报独立地进行交换，但是在某对特定的主机间的数据报流通常要经过一系列特定的路由器。所谓*流*（flow）是指在源和目的主机对之间发送的一系列分组，它们沿相同的路由经过网络。在资源分配问题中，流是个十分重要的抽象概念，本章将会用到。

流抽象的最大优势之一是它可以有不同粒度的定义。例如，流可以是主机到主机的（即有相同的源和目的地主机地址），或是进程到进程的（即有相同的源和目的地主机/端口对）。在后一种情况中，流与我们在本书中一直使用的信道的概念基本上是相同的。我们引进流这个新术语是因为对网络中的路由器而言，流是可见的，而信道只是一个端到端的抽象概念而已。图 6-2 描述了多个流通过一组路由器的过程。

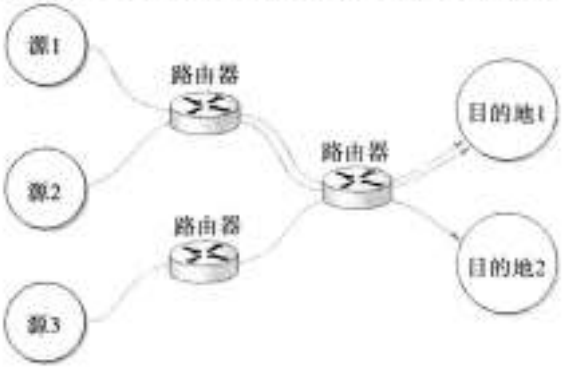


图 6-2 多个流通过一组路由器

因为有多个相关分组流经过每台路由器，所以有时为每个流保存一些状态信息是

有意义的，可以用这些信息对属于这个流的分组做资源分配决定。这些状态有时称为软状态（soft state），软状态和硬状态的主要差别是软状态并不总是需要用信令显式地创建或删除。纯粹的无连接网络在路由器上不维护任何状态，而纯粹的面向连接网络在路由器上维护硬状态，软状态则处于两者之间。一般情况下，网络的正确操作并不依赖于当前的软状态（每个分组不考虑软状态时，依然能正确地路由），但是如果路由器正在维护某个流的软状态信息，而某个分组又恰好属于这个流的话，那么路由器最好能处理这个分组。

注意一个流既可以隐式地定义，也可以显式地建立。对于前者，路由器检查每个经过的分组的首部中的地址，若观测出有相同的源和目的地的分组，就将这些分组当作属于同一个流来对待，以便进行拥塞控制。对于后者，源在网上发送一个建立流的报文，声明将启动一个分组流。然而对于显式建立的流和面向连接网络中的一个连接是否相同，还是有争议的。我们之所以提到这一点，是因为即使是显式建立的流也并不意味着任何端到端的语义，尤其是它并不意味着可靠而有序的虚电路传输。它的存在仅仅是用于资源分配的目的。在本章我们会看到隐式的流和显式的流。

3. 服务模型

本章前面的部分将集中讨论采用因特网尽力而为服务模型的机制。在尽力而为服务模型中，以完全相同的方式对待每个分组，端主机没有机会要求网络对某个流提供特殊保证或优先服务。定义一个支持某种优先服务或特殊保证的服务模型，比如保证视频数据流所需带宽，将是 6.5 节讨论的主题。这种服务模型能提供多种服务质量（Qualities of Service，QoS）。正如我们将看到的，实际上有一个可能性的范围，这个范围从纯粹的尽力而为服务模型到保证每个流能获得定量 QoS 的模型。最大的挑战之一就是定义一种服务模型，它可以满足许多领域的应用需求，甚至能够用于将来创建的应用。

6.1.2 分类方法

资源分配机制千差万别，进行彻底的分类是很难的。下面我们介绍可以表示资源分配机制特征的三个方面，并在本章中对其更细微的差异进行讨论。

1. 以路由器为中心和以主机为中心

资源分配机制可分为两大类：一类是在网络内部（即在路由器或交换机上）解决问题，另一类是在网络边缘（即在主机上，或许是在传输协议内）解决问题。由于网络中的路由器和网络边缘上的主机均参与资源分配，所以真正的问题是哪一个承担主要责任。

在以路由器为中心的设计中，由每台路由器决定什么时候转发分组，以及选择丢弃哪些分组，同时通知网上正在产生通信量的主机允许它发送的分组数目。在以主机为中心的设计中，端主机观测网络状态（如成功通过网络的分组数），并相应调整它们的行为。注意这两类设计并不是相互排斥的。例如，把拥塞管理的负担主要放在路由器上的网络仍然希望端主机能支持路由器发送的建议报文，而使用端到端拥塞控制的网络中的路由器仍然用某种策略（不论多么简单）决定当路由器队列发生溢出时要丢弃哪些分组。

2. 基于预定方式和基于反馈方式

有时资源分配机制也根据使用预定（reservation）还是反馈（feedback）进行分类。在基于预定的系统中，一些实体（如端主机）为流的分配向网络申请一定的容量。然后每台路由器分配足够的资源（包括缓冲区及链路带宽的百分比）以满足这一请求。如果某些

路由器由于自身资源不足而不能满足该请求，那么路由器会拒绝这个预定。这就像打电话时遇到忙音一样。在基于反馈的系统中，端主机在未预定任何容量的情况下开始发送数据，然后根据收到的反馈信息调整发送速率。反馈信息可以是显式的（explicit）（即发生拥塞的路由器向主机发送“请减慢速度”的消息），或隐式的（implicit）（即端主机根据外部可观察到的网络行为（如分组丢失）来调整发送速率）。

注意，基于预定的系统总是意味着采用以路由器为中心的资源分配机制。这是因为每台路由器都必须负责了解它当前的可用容量并判断能否接受新的预定，同时确信每台主机处在它预定的范围内。如果主机发送数据速率超过它所预定的容量，那么当路由器发生拥塞时，该主机的分组就是丢弃的候选对象。另一方面，基于反馈的系统意味着既可以采用以路由器为中心的机制，也可以采用以主机为中心的机制。通常，如果反馈是显式的，那么至少在某种程度上资源分配方案与路由器有关。如果反馈是隐式的，则几乎所有负担都落在端主机的身上，路由器的任务只是在发生拥塞时悄悄把分组丢掉。

预定不必由端主机发出，也可由网络管理员给流或聚集流量分配资源，见6.5.3节。

3. 基于窗口方式和基于速率方式

表示资源分配机制特征的第三种方法是依据基于窗口（window based）还是基于速率（rate based）。正如上面提到的，这只是多个领域中的一个，其中类似的机制和术语还用于流量控制和拥塞控制。流量控制和资源分配的机制都需要一种表达方式以向发送方传达允许其发送的数据量。传达这一信息通常用两种方法：窗口（window）和速率（rate）。我们已经见过基于窗口的传输协议，如TCP，其中接收方向发送方通知一个窗口。这个窗口对应于接收方的缓冲区大小以及对发送方传输数据量的限制，就是说，它支持流量控制。一个类似的机制——窗口通知——可以用于在网络内预定缓冲区空间（即它支持资源分配）。6.3节描述的TCP拥塞控制机制是基于窗口的。

也可以用速率控制发送方的行为，也就是接收方或网络每秒能接收多少比特。基于速率的控制机制对以某平均速率产生数据和需要保证最小吞吐量的多媒体应用程序往往是有用的。例如，在7.2.3节中描述的该类视频编解码器以1Mbps的平均速率和2Mbps的峰值速率产生视频。在本章后面我们会看到，在支持不同服务质量的基于预定的系统中，流的基于速率的特性是一个合乎逻辑的选择：发送方对每秒如此多的比特数做出预定，同时路径上的每台路由器在为其他流提供空间的情况下，确定是否能支持这一速率。

4. 资源分配分类小结

如上所述，资源分配有三种分类方法，每种又分为两类，共有八种不同的策略。虽然这八种方法都是可能实现的，但我们指出，实际上有两种策略看起来最为流行，这两种策略与网络的基本服务模型有关。

一方面，尽力而为服务模型不允许用户预定网络容量，因此它通常意味着使用反馈方式。这就意味着拥塞控制的责任大部分落在端主机身上，或许路由器会提供某些辅助。在实践中，这样的网络使用基于窗口的信息。它是因特网中采用的一般策略，也是6.3节和6.4节的中心内容。

另一方面，基于服务质量的服务模型可能包含某种预定形式。对这些预定的支持主要依赖路由器的参与，例如，将分组放入不同的队列依赖于它们要求的预定资源的级别。而且，由于窗口与用户所需的网络带宽仅是间接相关，自然就需要按速率表示预定的资源。

我们将在 6.5 节中讨论这个问题。

6.1.3 评价标准

最后我们讨论如何判断一种资源分配机制的优劣。回忆在本章开头的问题中我们提出的网络如何有效地和公平地分配资源的问题。也就是说，评价一种资源分配方案的优劣至少有两大标准。下面我们依次考虑每一种标准。

1. 有效的资源分配

评价网络资源分配方案的有效性通常从考虑网络的两个主要度量标准开始，即吞吐量和延迟。显然，我们希望有尽可能大的吞吐量和尽可能小的延迟。不幸的是，这些目标通常相互对立。对于资源分配算法而言，增加吞吐量就是要允许尽可能多的分组进入网络，使所有链路的利用率趋近 100%。这样做是为了尽量避免链路空闲的可能，因为空闲的链路会降低吞吐量。这种策略的问题在于，增加网络中的分组数就要增加每台路由器队列的长度，从而导致分组在网络中的延迟加大。

为描述这种关系，一些网络设计者提出使用吞吐量和延迟的比值作为衡量资源分配方案有效性的度量标准。这个比值有时称为网络的能力（power）[⊖]：

$$\text{Power} = \text{Throughput}/\text{Delay}$$

注意，不能明显看出能力是判断资源分配有效性的恰当标准。原因有两点：第一，能力这一度量标准是基于 M/M/1 排队网络理论的，[⊖] 这一理论假设有无限个队列；而实际网络的缓冲区是有限的，有时不得不丢弃一些分组。第二，能力这一度量标准的定义通常与单一连接（流）有关，如何把它扩展到多个有竞争的连接还不清楚。尽管有相当严格的限制，但是到目前为止还没有其他的度量标准被广泛接受，因此现在仍在使用能力这一度量标准。

我们的目标是使这个比率达到最大，这个比率是关于网络负载的函数。而负载是由资源分配机制来设置的。图 6-3 给出了典型的能力曲线，理想情况下，资源分配机制可以使其达到曲线的最高点。如果在最高点的左侧，表明这一机制过于保守；就是说，它允许发送的分组太少，未能充分利用链路。在最高点的右侧表明这一机制允许进入网络中的分组过多，以致由排队导致的延迟的增加量开始超出吞吐量的微小增益。

有趣的是，能力曲线非常类似分时计算机系统的系统吞吐量曲线。系统吞吐量随进入系统的作业数而增加，当在系统中运行的作业数达到某一点时，系统性能开始下降（系统花大量的时间进行内存页面交换），同时系统吞吐量也开始下降。

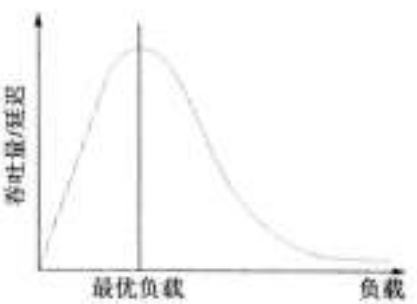


图 6-3 吞吐量与延迟的比值和负载的函数关系

在本章后几节中可以看到，许多拥塞控制方案只能很粗略地控制负载。也就是说，它们实在不可能实现只把“旋钮”转一点并只允许少量的额外分组进入网络。因此，网络设计者需要考虑系统在高负载下工作时将会

⊖ 确切的定义是 $\text{Power} = \text{Throughput}^{\alpha}/\text{Delay}$，其中 $0<\alpha<1$，$\alpha=1$ 时能力取极大值，吞吐量的单位是每秒的数据单元（例如比特），延迟的单位是秒。

⊖ 因为本书不是针对排队理论的，因此仅给出了 M/M/1 排队的简要描述，1 表示有一台服务器，M 表示分组到达和服务时间的分配依据“Markovian”，即指数方式。

出现的情况，即图 6-3 所示曲线到达最右端的情况。在理想的情况下，我们希望避免由于系统来回摆动而使系统吞吐量趋于零的情况。用网络术语来说，我们希望有一个稳定的（stable）系统——即使在网络负载很大的情况下，其中的分组依然可以在网络中继续传送。如果机制是不稳定的，网络将可能出现拥塞崩溃（congestion collapse）。

2. 公平的资源分配

网络资源的有效利用并不是衡量资源分配方案的唯一标准，还必须考虑公平性的问题。但是当试图定义公平资源分配的确切组成时，就会陷入一头雾水。例如，基于预定的资源分配方案提供显式的方法来产生可控制的不公平性。在这样的方案中，可以使用预定方式让一个视频流在某一链路上获得 1Mbps 的吞吐量，而同时让另一个文件传输在同一链路上却仅获得 10kbps 的吞吐量。

相反，如果没有显式的信息，当多个流共享某一条链路时，希望每一个流都获得相等的带宽份额。这种定义假设带宽的公平（fair）分配就是带宽的平均（equal）分配。然而即使不是预定方式，平等分配也不等同于公平分配。我们是否也应比较流的路径长度？比如，如图 6-4 所示，1 个跳数为 4 的流和 3 个跳数为 1 的流竞争时，如何体现公平？

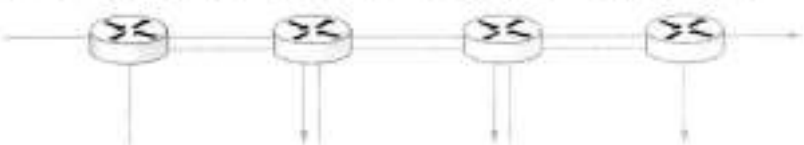


图 6-4 1 个跳数为 4 的流和 3 个跳数为 1 的流的竞争

假设公平就意味着平等，而且所有路径有相同的长度，网络研究员 Raj Jain 提出了一种评价拥塞控制机制公平性的度量标准，他的公平指数定义如下。给定一个流吞吐量的集合（x_1，x_2，…，x_n）（使用统一的单位，如 b/s），下列函数定义这些流的公平性指数：

$$f(x_1, x_2, \cdots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}$$

公平性指数总是产生 0～1 之间的一个数，1 表示最公平。为了更直观地了解这一度量标准，考虑 n 个流都获得每秒 1 个单位数据吞吐量的情况。它的公平性指数为

$$\frac{n^2}{n \times n} = 1$$

现在假设有一个流获得了 1+Δ 的吞吐量，则公平性指数为

$$\frac{((n-1)+1+\Delta)^2}{n((n-1)+(1+\Delta)^2)} = \frac{n^2+2n\Delta+\Delta^2}{n^2+2n\Delta+n\Delta^2}$$

注意，分母比分子大 $(n-1)\Delta^2$。因此不论这个流的吞吐量比其他流的吞吐量大还是小（正的或负的 Δ），公平性指数都降到 1 以下了。另一种简单的情况是考虑 n 个流中只有 k 个流有相等的吞吐量，而其余 $n-k$ 个流的吞吐量为 0，那么公平性指数降为 k/n。

6.2 排队规则[⊖]

无论资源分配机制的其余部分是简单还是复杂，每台路由器都必须应用某种排队规则来

⊖ 参见“实验十一”。

决定如何缓存等待发送的分组。可以把排队算法想象为分配带宽（哪些分组被传送）和分配缓冲区（哪些分组被丢弃）。它通过确定分组等待传输的时间直接影响分组的延迟。本节介绍两种常用的排队算法——FIFO（先进先出）和 FQ（公平排队）法，以及几个变种算法。

6.2.1 FIFO

FIFO（先进先出）排队也称为先来先服务（FCFS）排队，它的思想很简单：先到达路由器的分组先被发送。图 6-5a 所示为一个带“槽口”的 FIFO 队列，最多可容纳 8 个分组。假设每台路由器中缓冲区的空间是有限的，当有分组到达而队列（缓冲区）已满时，路由器将丢弃这个分组，如图 6-5b 所示。这种做法不会考虑这个分组属于哪个流，也不考虑它有多重要。由于到达 FIFO 队尾的分组被丢弃，所以有时也称为*队尾丢弃*（tail drop）。

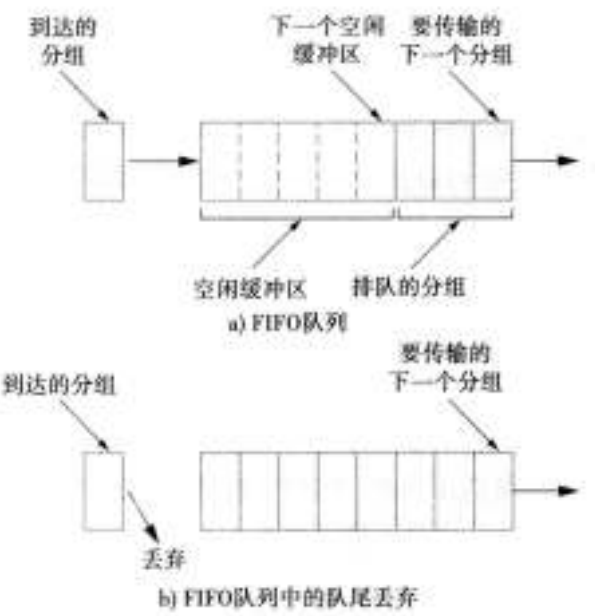


图 6-5

注意，队尾丢弃与 FIFO 是两个不同的概念。FIFO 是调度规则（scheduling discipline），它决定分组传送的顺序。队尾丢弃是*丢弃策略*（drop policy），它决定哪个分组被丢弃。因为 FIFO 和队尾丢弃分别是最简单的调度规则和丢弃策略，因此有时它们被看成一种组合排队实现。不幸的是，这一组合常常被简单地称作 FIFO 排队（FIFO queuing），实际上应更确切地称之为*带队尾丢弃的 FIFO*（FIFO with tail drop）。6.4 节提供了其他丢弃策略的一个例子，它采用比“是否有空缓冲区?”更复杂的算法决定何时丢弃分组。这种丢弃策略可以与 FIFO 或更复杂的调度规则一起使用。

带队尾丢弃的 FIFO 作为所有排队算法中最简单的算法，在本书写作时是因特网路由器中使用最广泛的算法。这种简单的排队方法将拥塞控制和资源分配的所有责任推到网络的边缘。这样，就假设目前因特网中流行的拥塞控制方式不能从路由器获得任何帮助：TCP 负责检测和处理拥塞。我们将在 6.3 节中看到它是如何运作的。

基本 FIFO 排队的一个简单变种是优先排队。它的思想是给每个分组一个优先级标志，这个标志可以被传送到如 6.5.3 节描述的 IP 首部中。路由器然后采用多个 FIFO 队列，每个队列对应一个优先级。路由器总是先发送完非空的最高优先级队列中的分组，然后再移到下一个优先级队列。在每个优先级队列中，分组都采用 FIFO 方式管理。这种方式与尽力而为传送模型有一点不同，但它还没有优化到对任意特定的优先级提供保证。它仅允许高优先级的分组插在队列的前面。

当然，优先排队的问题是高优先级队列可能使其他所有队列“挨饿”。也就是说，只要在高优先级队列中有一个高优先级分组要发送，那么低优先级队列就不能获得服务。要想让它成为可行的，必须对插入高优先级队列中的高优先级通信量进行严格的限制。显然，我们不能允许用户不受限制地将其分组设置为高优先级，必须防止他们一起这样做，

或者提供某种方式将一些用户的要求“驳回”。显然，我们可以用经济手段来实现：传送分组的优先级越高，网络费用就越高。但是在像因特网这样分散化的环境中，实现这样的方案还有很多问题。

在因特网中，使用优先级排队的一种情况是为了保护那些最重要的分组——通常指在网络拓扑结构改变后为保证路由表的稳定性所必需的路由更新分组。对于这样的分组，常常有一个特殊队列，它可以由 IP 首部中的差分服务代码点（Differentiated Services Code Point，DSCP）以前的 TOS 字段标识。事实上，这是 6.5.3 节中讲述的“区分服务”思想的一个简单例子。

6.2.2 公平排队

FIFO 排队的一个主要问题是不能区分不同的通信源，按前几节中的说法，就是它不能按分组所属的流分离它们。这个问题有两个不同的层面。一个层面是不能确定任何一个完全在源上实现的拥塞控制算法都能在路由器几乎不提供帮助的情况下控制拥塞。我们暂时停止对这一点的判断，留到下一节讨论 TCP 拥塞控制时再讨论。另一个层面，因为整个拥塞控制机制在源上实现，而且 FIFO 排队不提供监视这些源是否遵循这一机制的手段，因此一些恶意操作的源（流）就可能占用网络任意多的容量。我们再来考虑因特网，某个给定的应用程序完全可以不用 TCP，从而就能绕过端到端拥塞控制机制。（像当今因特网电话这样的应用程序就是这么做的。）这样的应用程序能够以其分组无限制地充斥路由器，从而使其他应用程序的分组被丢弃。

公平排队（FQ）是为解决这一问题而提出的算法。其思想是为路由器当前处理的每个流维护一个独立的队列。路由器以轮转方式为这些队列服务，如图 6-6 所示。当某个流发送分组过快时，它的队列被填满。当队列达到某一长度时，属于这个流队列的多余分组就被丢弃。用这种方式，一个源就不能以牺牲其他流为代价任意增加它占用的网络能力的份额。

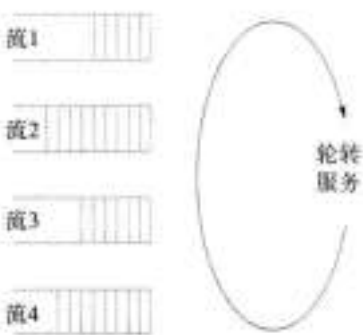


图 6-6 一台路由器上 4 个流的轮转服务

注意，FQ 既不涉及路由器将其状态通知通信源，也不以任何方式限制给定源传送分组的速率。换言之，FQ 依然与端到端拥塞控制协同使用。它简单地隔离各通信以使恶意的通信源不会影响那些忠实执行端到端算法的源。对于在良好拥塞控制算法管理下的流的集合，FQ 也对它们施加公平性。

尽管基本概念很简单，但你仍需了解相当多的细节。问题的复杂性主要在于路由器处理的分组长度不一定相同。要真正以公平的方式分配输出链路的带宽，就有必要考虑分组的长度。例如，路由器正在管理两个流，一个流有 1 000 字节的分组，另一个流有 500 字节的分组（或许由于来自这台路由器上游的分段），那么对每个流队列的分组进行的轮转服务将把链路带宽的 2/3 分给第一个流，而分给第二个流的仅是链路带宽的 1/3。

我们真正想实现的是按位轮转，也就是路由器从流 1 传送 1 位，再从流 2 传送 1 位，以此类推。显然，从不同的分组中按位交叉传送是不可行的。因此 FQ 机制采用如下方法来模拟这一行为：先判断对于按位轮转方式发送的分组何时传送完毕，然后根据完成时间对要传送的分组进行排序。

为了理解近似按位轮转的算法，考虑单个流的行为并设想一个时钟，每当所有活动流都传送 1 位时，时钟滴答 1 次。（流是活动的是指它有数据在队列中。）对于这个流，用 P_i 表示分组 i 的长度，用 S_i 表示路由器发送分组 i 的起始时间，用 F_i 表示路由器发送分组 i 的结束时间。如果用 P_i 表示发送分组 i 所用的时钟滴答数（记住流每传 1 位，时钟滴答数加 1），那么容易看出 $F_i=S_i+P_i$。

何时开始传送分组 i 呢？这取决于分组 i 是在路由器传完这个流的第 $i-1$ 个分组之前到达还是之后到达。如果是之前到达，那么逻辑上分组 i 的第 1 位将在分组 $i-1$ 的最后 1 位发完后立即发送。另一种情况，可能当路由器发完第 $i-1$ 个分组很久之后第 i 个分组才到达，也就是说在一段时间里这个流的队列是空的，这时轮转技术将不从这个流传送任何分组。若用 Ai 表示分组 i 到达路由器的时间，那么 $S_i=\max(F_{i-1},A_i)$，于是我们可以计算

$$F_i=\max(F_{i-1},A_i)+P_i$$

现在来讨论多个流的情况，同时我们发现确定 A_i 的一个要领。在分组到达时，我们不可能正好读出墙上的时钟。正如上面提到的，我们希望按位轮转时，每当所有活动流都传送 1 位时，时钟加 1 个滴答，因此需要时钟在流越多时增加的速度越慢。尤其是，如果有 n 个活动流，那么每当传送 n 个比特时，时钟就必须加 1 个滴答。这样的时钟将用于计算 A_i。

现在，用上面的公式来计算每个流中到达的每个分组的 F_i。然后把所有 F_i 当作时间戳，而下一个被传送的分组总是时间戳值最小的那个分组——基于上述原因，该分组应最先被传送完。

注意这表明一个流可能有一个分组到达，同时因为它比一个已经在队列中等待发送的其他流的分组更短，所以它可能被插在队列中比它长的分组前面。然而并不意味着新到的分组能抢先当前正在传送的分组。由于没有这种抢先机制，如上所述的 FQ 的实现方法不能精确地模拟试图达到的近似按位轮转技术。

为了更好地理解公平排队的实现是如何工作的，我们来看图 6-7 给出的例子。在图 6-7a 中显示了两个流的队列，算法在流 2 队列之前从流 1 中选择两个被传送的分组。在图 6-7b 中，当流 1 的分组到达时，路由器已经开始发送流 2 的分组。如果采用完全的按位发送公平排队，虽然流 1 的分组在流 2 发完之前到达，但在实现时流 1 的分组不能抢先于流 2 的分组发送。



图 6-7 实现公平排队的例子

在公平排队中有两个问题需注意。第一，只要队列中至少有一个分组，链路就绝不空闲。任何具有这一特点的排队方案称作工作保持（work-conserving）方案。工作保持产生的效果之一就是：如果我的流与其他流共享一条链路，而其他流此时不发送任何数据，那么我的流就可以使用全部链路能力。然而，一旦其他的流开始发送数据，它们将会使用自

已的那一份链路能力，而我的流可获得的能力将下降。

第二，当链路满负荷且有 n 个流在发送数据时，我所使用的能力不能超过链路带宽的 $1/n$。如果我试图超越这一界限，那么我的分组将会被赋予更大的时间戳值，使它们为了等待传送而在队列里等更长的时间。最终队列会溢出，但究竟是我的分组还是别人的分组被丢弃不是由我们正在使用的公平排队决定的。这是由丢弃策略决定的，FQ 是一个调度算法，和 FIFO 一样，它可以和任何丢弃策略结合。

因为 FQ 是工作保持的，所以一个流不用的带宽可自动地被其他任何流使用。例如，如果有 4 个流正在通过路由器，并且都是发送分组，那么每个流能获得带宽的 1/4。但是如果它们中有 1 个流已空闲了足够的时间，它的所有分组都被送出路由器的队列，那么可用带宽将被其余的 3 个流共享，于是每个流都会获得带宽的 1/3。这样就可以认为 FQ 给每个流提供的是有保证的最小份额，如果有的流不用它们的份额，别的流就可获得比它的保证多的带宽。

可以实现 FQ 的一个变种称为加权公平排队（weighted fair queuing，WFQ），它允许为每个流（队列）指定一个权值。这个权值逻辑上用来描述路由器为该队列服务 1 次所传送的比特数，它可以有效地控制流可获得的链路带宽的百分比。简单的 FQ 给每个队列赋权值 1，这意味着逻辑上每轮转 1 次为每个队列发送 1 位。当有 n 个流时，这导致每个流获得带宽的 $1/n$。但是在 WFQ 中，第一个队列的权值可以是 2，第二个队列的权值可以是 1，而第三个队列的权值可以是 3。假设每个队列总是有分组等待发送，那么第一个流获得可用带宽的 1/3，第二个流获得可用带宽的 1/6，而第三个流获得可用带宽的 1/2。

虽然我们已经用流描述了 WFQ，但要注意它可以按通信量的类别（class）实现，其中类别用其他方法定义，不同于在本章开始介绍的简单的流。例如，可以使用 IP 首部中的某些字段标识类别，并给每一类分配一个队列和权值。这正是 6.5.3 节中描述的区分服务体系结构的一部分。

注意，执行 WFQ 的路由器必须从某处获得每个队列指定的权值，或者通过手工配置，或者通过这些源的某种信令来获得。在后一种情况下，我们转向于基于预定的模型。仅给队列指定一个权值的方式提供一种弱的预定形式，因为这些权值只和流获得的带宽间接相关。（例如，流可获得的带宽还依赖于有多少流和它共享链路）。我们将在 6.5.2 节中看到如何使用 WFQ 作为基于预定的资源分配机制的组成部分。

结论 最后我们可以看到，整个队列管理的讨论说明了一个称为策略和机制分离（separating policy and mechanism）的重要系统设计原则。其思想是把每一种机制看作一个黑箱，它提供可由一组旋钮控制的多层面服务。策略规定这些旋钮的一组特定设置，但它并不知道（或不关心）黑箱如何实现。这种情况下讨论的机制是排队规则，而策略是哪个流获得何种级别服务的特殊设置（如优先级或权值）。我们将在 6.5 节讨论 WFQ 机制可能使用的一些策略。

6.3 TCP 拥塞控制

本节讲述当今用于端到端拥塞控制的主要实例，它是通过 TCP 实现的。TCP 的基本策略是把分组发送到没有预定的网络上，然后对出现的可观察事件做出反应。TCP 假设网络路由器只支持 FIFO 排队，但用公平排队也能工作。

TCP 拥塞控制是在 20 世纪 80 年代后期由 Van Jacobson 引入因特网的，大约是在

TCP/IP 协议栈已经实施 8 年之后。在此之前，因特网正遭受拥塞崩溃的困扰——主机按通知窗口允许的速度向因特网发送分组，在一些路由器上会发生拥塞（使一些分组被丢弃），同时主机会因超时而重传分组，引起更严重的拥塞。

一般而言，TCP 拥塞控制的概念是为每个源确定网络中有多少可用能力，从而知道它可以安全完成传送的分组数。一旦某个源有这么多分组在传送，它用确认（ACK）信号的到达表明它的一个分组已经离开网络，因而它不用增加拥塞级别就可以安全地向网络插入一个新的分组。通过使用确认信息测定分组的传送，人们称 TCP 是自同步（self-clocking）的。当然，在一开始就确定可获得的能力并非易事。因为其他连接时连时断，可获得的带宽不断随时间变化，这意味着任何指定的源必须能调整传送分组的数目，这使问题更加复杂。本节描述 TCP 解决这些问题以及其他问题所使用的算法。

注意，虽然我们一次描述 TCP 拥塞控制的一种机制，从而给人一种印象，好像我们在谈论三个独立的机制，但只有将它们作为一个整体考虑时才有 TCP 拥塞控制。此外，虽然我们要从称为标准 TCP（standard TCP）拥塞控制的变种说起，但我们将看到相当多的变种都在使用当中，研究人员不断探索新的方法来解决这一问题。稍后讨论其中一些新的方法。

6.3.1 加性增/乘性减

TCP 为每个连接维护一个新的状态变量，称为 CongestionWindow（拥塞窗口），源用它来限制给定时间内允许传送的数据量。拥塞控制窗口与流量控制的通知窗口相对应。TCP 做如下修改：允许未确认数据的最大字节数为当前拥塞窗口和通知窗口的最小值。这样，用 5.2.4 节中定义的变量，对 TCP 的有效窗口修订如下：

$$\text{MaxWindow} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$

$$\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

也就是说，在 EffectiveWindow 的计算中用 MaxWindow 代替 AdvertisedWindow。这样，允许 TCP 源发送分组的速率不超过网络或目标主机可接受的速率中的最小值。

当然，问题是 TCP 如何得到一个合适的 CongestionWindow 的值。与 AdvertisedWindow 不同，AdvertisedWindow 的值是由连接的接收方送出的，没有任何一个源向 TCP 的发送方发送一个合适的 CongestionWindow 值。答案是 TCP 源根据它所获得的网络中存在的拥塞级别来设定 CongestionWindow。当拥塞级别上升时减小拥塞窗口，而当拥塞级别下降时加大拥塞窗口。这种把两者合在一起的机制通常称为加性增/乘性减（additive increase/multiplicative decrease，AIMD），在下面的讨论中，会明白用这样一个拗口的名字的原因。

那么，关键问题是源如何确定网络拥塞并且如何减小拥塞窗口？答案是基于这样的观察，分组不能被传送和导致超时的主要原因在于拥塞造成分组被丢弃。因为传输错误而造成丢弃分组的情况是很少的。因此，TCP 认为超时是发生拥塞的标志，并据此降低正在传输的速率。需要说明的是，每发生一次超时，源就将 CongestionWindow 设为当前值的一半，这种对于每次超时将 CongestionWindow 值减半的做法正是对应于该 AIMD 机制中所指的“乘性减”。

尽管 CongestionWindow 是按字节定义的，但如果按整个分组来考虑成倍减少是最容易理解的。例如，假设当前 CongestionWindow 被设为 16 个分组。若检测到一个分组丢

失，CongestionWindow 就被设置为 8。（通常，发生超时就是检测到分组丢失，但在下面将会看到，TCP 还有检测丢弃分组的其他机制。）如果检测到另外的分组丢失，CongestionWindow 的值将减到 4，然后就减少到 2，直到减至一个分组的长度。不允许 CongestionWindow 低于一个分组的长度，在 TCP 术语中，一个分组的长度称为最大报文段长度（Maximum Segment Size，MSS）。

只减小窗口大小的拥塞控制策略显然过于保守，同时也需要能增大 CongestionWindow 以充分利用网络新增的能力。这就是机制中谈到的“加性增”部分，它的工作原理如下：每当源成功地发送 CongestionWindow 设定的分组数——也就是说，每个发出的分组都在最近的往返时间（RTT）时间内获得确认，源就将等于一个分组长度的值加到 CongestionWindow 上。这种线性增加的过程如图 6-8 所示。注意，在实际应用中 TCP 不会等待整个窗口的确认值收到之后才给拥塞窗口增加一个分组的值，而是随着到达的每一个确认增加一个小的值。具体地讲，每次收到一个确认后，拥塞窗口按如下公式增加：

$$\text{Increment} = \text{MSS} \times (\text{MSS}/\text{CongestionWindow})$$

$$\text{CongestionWindow} += \text{Increment}$$

也就是说，不是在每个 RTT 内将 CongestionWindow 增加整个 MSS 值，而是每收到一个 ACK，CongestionWindow 就增加 MSS 值的一部分。假设每个 ACK 确认收到 MSS 字节，那么 MSS 值的这一部分等于 MSS/CongestionWindow。

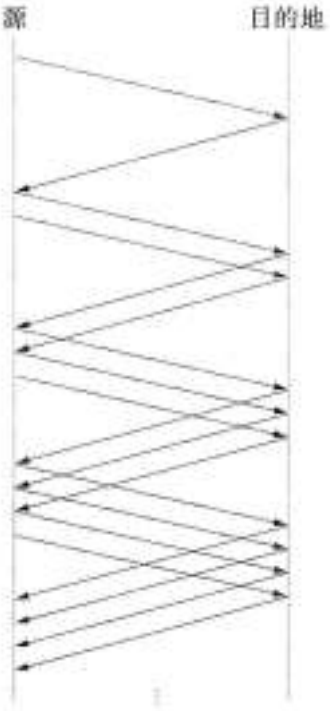


图 6-8 在加性增期间传输的分组，每个 RTT 中增加一个分组

连续增加或减少拥塞窗口的模式将贯穿连接的生命期。事实上，如果用 CongestionWindow 作为时间的函数画出当前的值，将得到如图 6-9 所示的锯齿型图案。理解有关加性增/乘性减的重要概念是：源减小它的拥塞窗口的速度比加大这个窗口要快得多。这与加性增/加性减策略形成鲜明对比，后一种策略是指每收到一个 ACK，窗口值增加 1 个分组，同时每发生一次超时，窗口值减少 1 个分组。事实证明加性增/乘性减是拥塞控制机制达到稳定的一个必要条件（参考“扩展阅读”）。大幅度减少窗口与谨慎增加窗口的一个直观原因是，窗口过大的后果要比窗口过小严重得多。例如，当窗口过大时，被丢弃的分组要被重传，导致拥塞更加严重，因而迅速从这种状态中摆脱出来非常重要。

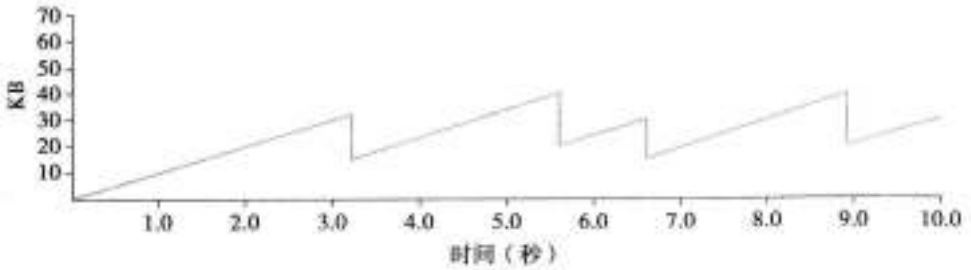


图 6-9 典型的 TCP 锯齿图案

最后，由于超时是拥塞发生的标志，并触发窗口大小的成倍减少，所以 TCP 需要提

供最精确的超时机制。TCP 的超时机制在 5.2.6 节中已谈到，在此不再重复。但要记住该机制的两个要点：①超时值被设置为平均 RTT 和该平均值的标准偏差值的一个函数。②由于用精确时钟测量每次传输的开销太大，因此 TCP 只用粗粒度（500 ms）时钟在每个 RTT（而不是每发一个分组）对往返时间进行一次采样。

相关主题

丢包不等于拥塞：无线网络中的 TCP

TCP 拥塞控制有一种严重的失败倾向。当链路因位错误产生高频率的丢包时，TCP 误认为这是拥塞的信号，这种现象在无线链路上非常常见。结果是 TCP 发送端降低其发送速率，但对位错误率没有影响，所以这种丢包情况会延续到发送端窗口只剩一个分组为止。这时，TCP 的吞吐量会降低到每个往返时间一个分组，这远远小于无拥塞的网络中的正常速率。

鉴于这种情况，你可能想知道 TCP 如何在无线网络中运转。幸运的是，有许多解决这个问题的方法。最常见方法的是，在链路层采取一些措施来减少或隐藏由于位错误导致的分组丢失。例如，802.11 网络对所发送的报文提供了一种前向纠错（FEC）的方法，使一些错误能够在接收方进行纠正。另一种方法是链路层重传，这样，即使一个分组被损坏或丢失，它最终都可以被成功发送，TCP 永远不会察觉初始损失。每一种方法都有它的缺点：FEC 浪费了一些带宽，有时仍会纠错失败，而重传不仅增加了连接的 RTT 及其变化情况，而且会导致更糟糕的表现。

在某些情况下使用的另一种方法是将 TCP 连接切分为无线和有线段。这个方法有很多延伸，但基本方法是将有线报文段上的丢失视为拥塞信号，而将无线报文段上的丢失视为由位错误引起的。这种技术已经在卫星网络中使用，其 RTT 时间过于冗长，因此人们无法忍受更长时间的拥塞处理。不像链路层方法，这种方法是一种协议中端到端操作的基本变换。这也意味着，在连接的正向和反向路径必须经过相同的“中间盒”，这个中间盒就是用来切分连接的。

另一种方法试图智能地区分拥塞和位错误两种丢失的差异。线索表明拥塞导致丢包，例如 RTT 的增加和后续的丢包间的相关性。显式拥塞通知（ECN）标记（参见第 6.4.2 节）也可以提供一个拥塞即将来临的迹象，所以随后的一系列损失更可能与拥塞有关。显然，如果你能发现这两种丢失之间的差异，那么 TCP 不需要因位错误有关的丢失而降低其窗口。不幸的是，这种方法很难 100%判定，同时这一领域也是研究的热点。

6.3.2 慢启动

当源的操作接近网络可达到的能力时，使用上述加性增机制是正确的，但是当源从头开始时，则要用很长时间才能延续一个连接。因此 TCP 提供第二种机制，讽刺性地称为慢启动（slow start）[⊖]，它从较小的值开始迅速增加拥塞窗口，以指数方式而不是线性方式有效增加拥塞窗口。

具体说来，源开始将 CongestionWindow 设置为 1 个分组，当这个分组的确认到达源

⊖ 尽管最初的论文中将其称为“slow-start”，但今天“slow start”更常用，因此此处忽略了连字符。

时，TCP将CongestionWindow加1，然后发送两个分组。当收到两个相应的确认后，TCP将CongestionWindow加2（即每个确认加1），然后发送4个分组。最终结果是TCP在每个RTT内将传送的分组数加倍。图6-10显示了慢启动期间发送分组数的增长情况，可与图6-8所示的累次增加的线性增长情况相比较。



图6-10 慢启动期间传输的分组

开始时我们会对为什么把指数机制称为“慢速”有些迷惑，但如果放在特定的历史背景中就可以解释了。我们并不需要将慢启动与前一小节的线性机制相比，而是与TCP的最初行为相比。考虑当建立一个连接和源开始发送分组（也就是当它目前没有分组发送）时会发生什么事情。若源发送通知窗口允许的分组数（这恰是提出慢启动前TCP所做的工作），那么即使网络有大量的带宽可用，路由器也可能处理不了这种一连串的分组。这完全取决于路由器上可用的缓冲区空间的大小。因此慢启动用于将分组隔开，使分组突然增多的情况不会发生。换句话说，尽管指数增长比线性增长快，但慢启动比起立即发送整个通知窗口的数据量要“慢”得多。

实际上慢启动能够在两种不同情况下运作。第一是在刚开始连接时，源不知道它在给定时间内能发送多少分组。（记住，TCP能运行在从9 600bps～2.4Gbps的链路上，因此源无法知道网络的能力。）在这种情况下，慢启动在每个RTT时间内不断将CongestionWindow加倍直至发生分组丢失，这时超时引起CongestionWindow的成倍减少（除以2）。

使用慢启动的第二种情况更为微妙，这发生在连接停止而等待超时发生的时候。回想一下TCP滑动窗口算法的原理，当一个分组丢失时，源最终达到某一点，在此点上它已经发送通知窗口允许的分组数量，因此当它等待一个不会到达的确认信息时阻塞。最终发生超时，但在这段时间没有分组在传送，也就是源收不到确认以“同步”新分组的传送。源将收到一个要求重新打开整个通知窗口的一个累积的ACK，但如上所述，源于是用慢启动重新启动数据流，而不是立即将整个窗口值的数据都送到网上。

尽管源再次使用慢启动，但现在所得到的信息比连接开始时要多。尤其是，源知道CongestionWindow的当前（并且是有用的）值，这是最近一个分组丢失前存在的CongestionWindow的值，将其除以2作为丢失时的值。可以把它看作目标（target）拥塞窗口。用慢启动将发送速率快速增长到这个值，超过该值后，则用累次增加的方法。注意，要考虑一个小的管理操作问题，我们希望能记住由于成倍减少而产生的目标拥塞窗口以及慢启动所用到的实际（actual）拥塞窗口。为解决这一问题，TCP引入一个临时变量存储目标窗口，一般称为CongestionThreshold（拥塞阈值），它的值等于由于成倍减少而产生的CongestionWindow值。然后变量CongestionWindow被重置为一个分组，以后每收到一个ACK增加一个分组，直至达到CongestionThreshold，在这一点每一个RTT增加一个分组。

换句话说，TCP用如下代码段定义拥塞窗口：

```
{
    u_int   cw = state->CongestionWindow;
    u_int   incr = state->maxseg;

    if (cw > state->CongestionThreshold)
        incr = incr * incr / cw;
    state->CongestionWindow = MIN(cw + incr, TCP_MAXWIN);
}
```

其中 state 表示某个特定 TCP 连接的状态，TCP_MAXWIN 定义允许拥塞窗口增长到的上界。

图 6-11 描绘了 TCP 的 CongestionWindow 随时间增减的轨迹，同时说明慢启动和加性增/乘性减的相互作用。这一轨迹取自一个实际 TCP 连接，并显示 CongestionWindow 随时间变化的当前值。

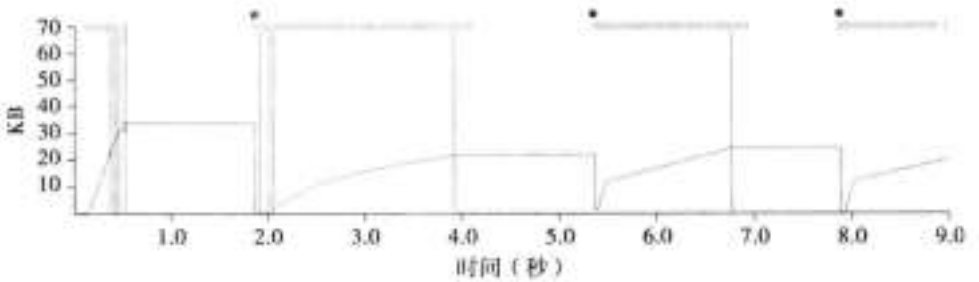


图 6-11 TCP 拥塞控制行为。曲线＝随时间变化的 CongestionWindow 的值；图上方的黑点＝超时；图上方的细线＝每个分组被传送的时刻；竖条＝最终要被重传的分组首次传送的时刻

有关这一轨迹需注意以下几件事情。首先拥塞窗口在连接开始时迅速增加，这对应于初始慢启动过程。慢启动阶段持续到大约进入连接 0.4s 后几个分组丢失，此时 CongestionWindow 保持在约 34KB 处（为什么许多分组在慢启动过程中丢失将在下面讨论）。拥塞窗口不变是由于几个分组丢失而导致没有 ACK 到达。事实上，在这段时间里没有发送新的分组，如图 6-11 中上方没有小条的那一段。最后大约在 2s 时发生超时，此刻拥塞窗口减半（即从大约 34KB 减到 17KB 左右），同时 CongestionThreshold 也被设置为该值。然后慢启动将 CongestionWindow 重设为 1 个分组并开始由此增加。

在这一轨迹中没有足够的细节可以精确地看出在刚过 2s 时丢失一对分组后发生的事情，所以直接来看在 2～4s 间拥塞窗口的线性增加。这对应于本节中的加性增。大约从 4s 开始，由于丢失分组 CongestionWindow 变平坦。现在，大约在 5.5s：

1）发生超时，使拥塞窗口减半，从大约 22KB 降至 11KB，同时 CongestionThreshold 也设置为该值。

2）CongestionWindow 被重设为 1 个分组，随之发送方进入慢启动。

3）慢启动使 CongestionWindow 按指数增加，直至 CongestionThreshold。

4）CongestionWindow 开始线性增加。

大约在 8s 时又一次超时发生，重复上述过程。

现在我们回到慢启动开始时为什么会有许多分组丢失的问题。TCP 在此试图了解网络上有多少可用带宽，这是一项很困难的任务。如果源在这个阶段不积极，例如如果它仅仅线性增加拥塞窗口，那么要确定有多少可用带宽需用很长时间。这对连接可达到的吞吐

量可能产生严重影响。另一方面，若源在这一阶段比较积极，随着 TCP 采用指数增长，源可能有窗口值一半的分组被网络丢弃。

要了解在指数增加过程中的情况，考虑这样一种情形：源通过网络刚好能成功发送 16 个分组到目的地，这使拥塞窗口增至 32。但是假设网络碰巧只有支持从这个源发送的 16 个分组的能力，那么在新的拥塞窗口之下发送的 32 个分组中的 16 个可能被网络丢弃。实际上，这只是最坏的情形，因为在某些路由器上还可以缓存一些分组。这个问题会随着网络的延迟带宽积的增加而变得日益严重。比如说，延迟带宽积为 500KB，这意味着在每个连接开始时，每个连接最多可能丢弃 500KB 的数据。当然，这是假设源和目的地都实行"大窗口"扩充。

一些协议设计者已提出了替换慢启动的方法，源试图通过更复杂的方式来估计可用带宽。一个最近的例子是正在 IETF 进行标准化的快启动（quick-start）机制。其基本思想是 TCP 发送方能够在 SYN 分组的 IP 选项中放置请求速率来请求一个比慢启动初始发送速率高的初始发送速率。沿途路由器可以检查选项，在该流经过的链路上估计拥塞程度，决定是否该速率是可接受的，是否一个较低的速率是可接受的，或者是否应该使用标准的慢启动。当 SYN 到达接收方时，它将包含路径上所有路由器可接受的速率以及路径上不支持快启动请求的一台或多台路由器的指示。在前一情况下，TCP 发送方使用该速率开始传输；在后一情况下，它返回到标准的慢启动。如果允许 TCP 以较高的速率启动，那么会话能够更快地进入传输流，而不必等待多个往返时间。

显然如此改进 TCP 面临的挑战之一是相对于标准 TCP 而言的，它需要路由器进行更充分的协调。如果路径中的一台路由器不支持快启动，那么系统只能返回到标准的慢启动。因此，这种改进要用在因特网中还需要相当长的时间。目前，这种方法主要使用在受控制的网络环境中（例如，研究性的网络）。

6.3.3 快速重传和快速恢复

至今描述的机制只是将拥塞控制加至 TCP 中的最初方案的一部分。人们很快发现，TCP 超时的粗粒度实现方法导致连接在等待一个计时器超时时，有很长一段时间连接无效。因此，一种称为快速重传（fast retransmit）的新机制被加入到 TCP 中。快速重传是一种启发式的机制，有时它触发对丢失分组的重传比常规超时机制更快。快速重传机制并不能代替常规超时机制，它只是增强功能。

快速重传的思想简单明了。每当数据分组到达接收方时，接收方会用 ACK 作为响应，即使这个序号已被确认过。这样，当分组没有按正常顺序到达，或当 TCP 由于它前面的数据还没有到达而不能确认分组中的数据时，TCP 会将它上次发过的确认信息再发一次。同一个确认的第二次重传称为重复确认（duplicate ACK）。当发送方发现一个重复确认时，它就知道接收方必定收到一个未按正常顺序到达的分组，表明它前面的分组可能丢失。由于前面的分组有可能只是延迟到达，并没有丢失，所以发送方等待，直至看到一定数量的重复确认，然后才重传丢失的分组。实际应用中，TCP 等待直到看到三个重复确认，才开始重传分组。

图 6-12 说明了重复确认如何导致快速重传的过程。在此例中，目的地址收到分组 1 和分组 2，但分组 3 在网络中丢失。这样，目的地址在分组 4 到达时为分组 2 发送一个重复确认，当分组 5 到达时又发一个，依此类推。（为简化这个例子，我们按分组 1、2、3

等来考虑，而不必考虑每个字节的序号。）当发送方看到分组 2 的第三个重复确认时（发送第三个确认是因为接收方已收到分组 6），它就重传分组 3。注意当被重传的分组 3 的拷贝到达目的地时，接收方给源发送一个累积确认，用于确认已收到包括分组 6 的所有分组。

图 6-13 说明了带有快速重传机制的一个 TCP 版本的行为。如果和图 6-11 中的轨迹（其中未实现快速重传）相比，你会发现一个有趣的现象：拥塞窗口保持不变且没有分组发送的那段很长的时间已经被消除。通常，在一次典型的 TCP 连接中，这种技术可以消除大约一半的粗粒度的超时，从而使吞吐量比其他方法提高 20%左右。但要注意，快速重传策略并不能消除所有粗粒度的超时。这是因为对于小的窗口，没有足够多的分组传送引起足够多的重复确认。假设丢失足够多的分组（例如，在初始慢启动阶段所发生的），滑动窗口算法最终阻塞发送方直至超时发生。假设当前最大通知窗口为 64KB，在实际应用中，TCP 的快速重传机制的每个窗口最多能检测到三个被丢弃的分组。

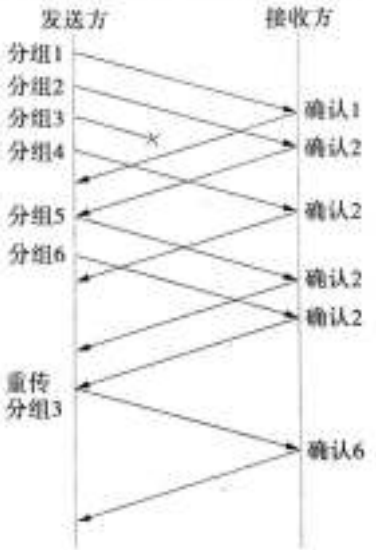


图 6-12 基于重复 ACK 的快速重传

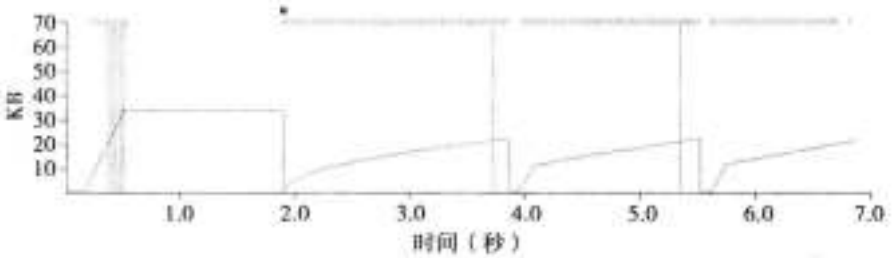


图 6-13 带快速重传的 TCP 轨迹。曲线＝拥塞窗口；黑点＝超时，细线标记＝每个分组传送的时刻；竖条＝最终被重传的分组的首次传送时刻

最后，我们可以再做一个改进。当快速重传机制发出拥塞信号时，不把拥塞窗口退回 1 个分组并启动慢启动，而是可能利用还在管道中的 ACK 去同步分组的发送。这种称为快速恢复（fast recovery）的机制有效地消除了在快速重传检测到一个丢失分组并开始加性增之间的慢启动过程。例如，快速恢复避免了图 6-13 中 3.8～4s 之间的慢启动过程，直接将拥塞窗口减半（从 22KB 减至 11KB）并重新开始加性增。换句话说，慢启动仅用于连接开始和发生粗粒度超时的时刻。在其他时间，拥塞窗口遵循加性增/乘性减模式。

相关主题

快速 TCP?

在过去的 20 年中，有关 TCP 能够运行得多快的争论相当多。首先有声明说 TCP 太复杂了，因此随着网络向 G 比特的范围发展，它不能在主机软件上运行得太快。该声明多次受到驳斥。最近，有重要理论表明标准 TCP 在高带宽环境下的性能是受限制的。TCP 的拥塞控制行为分析显示出，在稳定状态下，TCP 的吞吐量大约是：

$$\text{Rate} = \left(\frac{1.2 \times \text{MSS}}{\text{RTT} \times \sqrt{p}}\right)$$

在具有100ms RTT和10Gbps链路的网络上，如果分组丢失率在每50亿个分组丢失1个以下的话——等价于每100分钟出现一个拥塞事件，那么一个TCP连接只能达到接近于链路速率的吞吐量。虽然光纤上比特错误率很低，分组丢失率很小，但也比这个要求的分组丢失率高得多，因此不可能用单一TCP连接填充管道。人们提出了许多在高带宽网络上改进TCP行为的方案，增加了其范围。受限于MSS，已经提出的一个简单的改进是增加分组大小。不幸的是，增加分组大小也增加了给定分组出现比特错误的机会，因此单纯增加MSS不是足够的。IETF提出的其他方案对TCP避免拥塞的方式进行了改进，试图使TCP能够更好地利用可用带宽。这里的挑战对于标准TCP实现来说是公平的，对于避免导致当前TCP行为的拥塞事件来说也是公平的。

目前探索性的RFC提出了高速TCP方案，这对于在高带宽环境下并且不和其他大量流量竞争情况下的TCP来说是有效的。实际上，当拥塞窗口变得很大时，高速TCP以比标准TCP大得多的拥塞窗口来启动。在拥塞窗口相对比较小的一般环境下（大约40×MSS），高速TCP与标准TCP是无法区分的。在“扩展阅读”中列出的部分其他方案也是如此。注意，在Linux操作系统中默认的TCP行为现在是基于一个叫作CUBIC的TCP变种，它急剧扩大了在高延迟带宽积制度下的拥塞窗口。同时在带宽受限的情况下，保持与旧TCP变种的兼容性。

上面提到的是快启动方案，它改变了TCP的启动行为。因为它使TCP连接能够更快地提高发送速率，因此当连接时间比较短或应用程序周期性地停止发送数据使TCP返回到慢启动阶段时，它对TCP性能的影响是非常明显的。

然而另一个方案FAST TCP采取了类似TCP Vegas（在下一章中描述）的方法。其基本思想是预测拥塞的发生并避免它，从而在减少拥塞窗口时避免性能损失。

好几种方案涉及了对TCP的更大改变，甚至用新开发的协议对其进行了替换。这些方案对于在高带宽环境下快速并公平地填充管道有相当大的好处，但也面临着需要更高配置的挑战。本章末将向读者介绍该领域的最新工作进展。

6.4 拥塞避免机制

TCP的策略是一旦发生拥塞时就控制它，而不是从一开始就试图避免拥塞的发生。理解这一点是重要的。事实上，TCP在努力找到拥塞发生点的过程中不断地给网络增加负载，然后从这一点回退。换言之，TCP需要制造丢失分组来发现连接的可用带宽。另外一种富有吸引力但还未被广泛接受的策略，就是预测拥塞将在何时发生，然后在分组刚要被丢弃前降低主机发送数据的速率，我们称这种策略为拥塞避免（congestion avoidance），以便与拥塞控制（congestion control）相区别。

本节描述三种不同的拥塞避免机制。前两种方法很相似：它们在路由器上增加少量附加功能来协助拥塞中的端节点。第三种机制和前两种截然不同，它试图纯粹从端节点避免拥塞。

6.4.1 DECbit

第一种机制用于数字网络体系结构（Digital Network Architecture，DNA），DNA是使用面向连接的传输层协议的一个无连接网络。因此这种机制可用于TCP和IP。如上所述，这种机制的思想是更均匀地将拥塞控制的责任分摊给路由器和端节点。每台路由器监视当前负载并且在拥塞将发生时明确地通知端节点。这种通知是通过在流经路由器的分组

中设置1个二进制拥塞位来实现的，这个位因此叫作DECbit。然后目标主机将这一拥塞位复制到它传送回源的ACK中。最后源调整发送速率来避免拥塞。下面的讨论描述更多算法的细节，从路由器中发生的情况开始。

一个拥塞位被加到分组首部。在分组到达时，如果平均队列长度大于或等于1，那么路由器就在分组中设置这一位。平均队列长度用最近一次忙+闲（busy+idle）周期加上当前忙周期的时间间隔来测量。（当路由器传送分组时称为忙，否则称为闲。）图6-14显示了路由器上的队列长度是时间的函数。本质上讲，路由器计算曲线下的区域，并以此值除以这段时间间隔来计算平均队列长度。用队列长度为1作为设置拥塞位的触发点是对于有效排队（因此提高吞吐率）和增加空闲时间（因此降低延迟）的一种折衷方案。换言之，长度为1的队列看起来优化了能力函数。

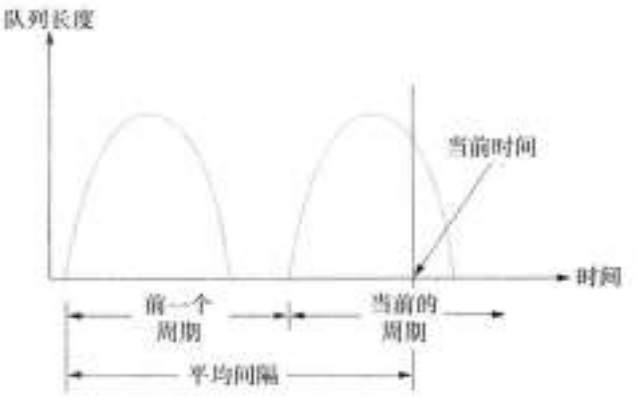


图6-14 在路由器上计算平均队列长度

下面我们再来看看这种机制中主机部分需做的工作。源记录有多少个分组引起路由器设置拥塞位。实际应用中，源维护一个拥塞窗口（如同在TCP中一样），并观察最近一个窗口中引起设置拥塞位的分组数的比例。若这个比例小于50%，源就将其拥塞窗口增加1个分组。若这个比例大于或等于50%，源就将其拥塞窗口减至原有值的0.875，选择50%作为阈值，是基于对能力曲线中峰值的分析。选择“增加1个分组，减至原拥塞窗口值的0.875”的原则是因为加性增/乘性减会使这种机制更加稳定。

6.4.2 随机早期检测（RED）

第二种机制叫作随机早期检测（Random Early Detection，RED），它与DECbit很相似，是在每台路由器上编程监视自己的队列长度，当检测到拥塞即将发生时，就通知源调整拥塞窗口。RED是在20世纪90年代早期由Sally Floyd和Van Jacobson发明的，它与DECbit方案主要有两点不同。

首先，RED不是显式地发送一个拥塞通知消息给源，RED最常见的实现方式是通过丢弃一个分组隐式地通知源发生拥塞。因此，源实际上是通过随之而来的超时或重复确认得到通知。RED被设计为与TCP配合使用，目前通过超时方式检测拥塞（或者通过其他方式检测丢失分组，例如重复确认）。至于RED名称中的“早期”一词，是指网关会在它不得不丢弃一个分组之前就提前把它丢弃，以此通知源应提早减少拥塞窗口。换言之，路由器在它的缓冲区被完全填满之前就提前丢弃少量的分组，以此使源放慢发送分组的速率，作为表明以后不必再丢弃大量分组的希望。需要说明的是，RED只要简单地将丢弃（dropping）分组改为标记（marking）分组，就可以很容易地让它与显示的反馈方案一同工作，详细讨论见下面的显式拥塞通知。

相关主题

显式拥塞通知（ECN）

由于目前的RED总是通过丢弃分组发出拥塞信号，因此最近人们已经在关注采用显

式通知是否是一个更好的策略。这就导致了制定因特网的显示拥塞通知（Explict Congestion Notification，ECN）标准。

基本争论是，丢弃分组当然可以作为发生拥塞的信号，而且对于那些长时间的大块传输来说可能是正确的做法，但是这样做会损害那些对延迟和丢失几个分组敏感的应用。例如像Telnet 和 Web 浏览这样的交互式应用，通过显式通知来获知拥塞将对这类应用更为合适。

ECN 在技术上需要用 2 比特位，建议标准使用 IP TOS 字段的第 6 位和第 7 位。其中一位是由信源设置的 ECN 位，表明源端能够处理拥塞通知。另一位由端到端路径上的路由器在发生拥塞时设置，由目的主机返回给源主机。源主机上运行的 TCP 将对 ECN 位的设置值做出响应，响应方式与对被丢弃分组的响应一样。

正像任何好主意一样，最近 ECN 引起的关注已经让人们停下来并开始思考，如何使网络从在网络边缘的主机与网络中间的路由器之间以 ECN 方式交换数据分组上捎带的信息这一方式中获益。这种通用的策略有时称为活动队列管理（active queue management），而且近期的研究似乎表明，这对那些具有较大延迟-带宽的 TCP 流尤其有价值。有兴趣的读者可以进一步阅读本章最后给出的相关参考文献。

RED 和 DECbit 的第二个不同点是在决定何时丢弃一个分组和丢弃哪个分组的细节上。为理解其基本思想，考虑一个简单的 FIFO 队列。它不是等队列完全排满以后再将每个到达的分组丢弃（6.2.1 节的队尾丢弃策略），而是当队列长度超过某个丢弃级别（drop level）时，按照某个丢弃概率（drop probability）将到达的分组丢弃。这种思想称为早期随机丢弃（early random drop）。RED 算法定义如何监视队列长度和何时丢弃分组的细节。

在下面的段落中，我们描述由 Floyd 和 Jacobson 最初提出的 RED 算法。我们注意到，还有发明人和其他研究者提出的几种改进算法，本章的“扩展阅读”将讨论其中的一些改进算法。然而，其基本思想与下面描述的相同，当前的实现也接近于下面的算法。

第一，RED 用加权动态平均值来计算平均队列长度，这个加权动态平均值和最初 TCP 超时计算中用到的权值类似。就是说，AvgLen 用如下公式计算

$$\text{AvgLen} = (1-\text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}$$

其中 $0 < \text{Weight} < 1$ 且 SampleLen 是样本测量时的队列长度。在大多数软件实现中，每当一个新的分组到达网关时，就测量一次队列长度。在硬件中，它可以在某个特定的采样间隔计算。

使用平均队列长度而不是瞬间队列长度，是因为平均队列长度能更准确地捕获拥塞的动向。由于因特网通信量的突发性，队列会突然排满然后又变空。如果队列大部分时间都是空的，那么得出结论路由器拥塞并告知主机放慢速度可能并不合适。于是，加权动态平均值的计算试图通过滤出队列的短期变化来检测长期存在的拥塞，如图 6-15 的右半部分所示。可以把动态平均值想象成一个低通滤波器，其中 Weight（权值）决定滤波器的时间常量。如何选择这个时间常量的问题将在下面讨论。

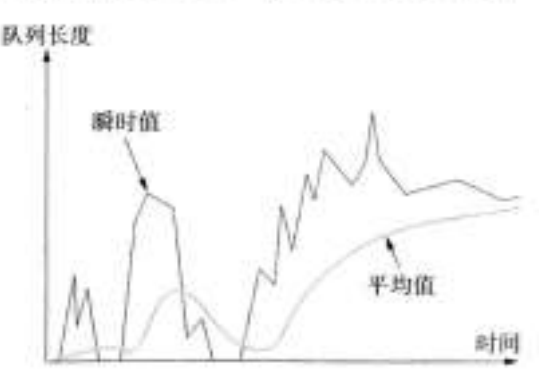


图 6-15 加权动态平均队列长度

第二，RED 有两个队列长度阈值用于触发某种特定的活动：MinThreshold（最小阈值）和 MaxThreshold（最大阈值）。当分组到达网

关时，RED将当前的AvgLen同这两个阈值比较，采用下面的规则：

如果 AvgLen≤MinThreshold

→分组入队列

如果 MinThreshold<AvgLen<MaxThreshold

→计算概率 P

→按概率 P 丢弃到达的分组

如果 MaxThreshold≤AvgLen

→丢弃到达的分组

也就是说，如果平均队列长度比下阈值小，那么无需采取任何措施；如果平均队列长度大于上阈值，那么分组总是被丢弃。如果平均队列长度在两个阈值之间，那么按某个概率P将新到的分组丢弃。图6-16描绘了这种情况。P和AvgLen的关系大致如图6-17所示。注意，当AvgLen的值在两个阈值之间时，丢弃分组的概率缓慢增长，在上阈值处达到MaxP，在那一点直接跳到1。其理由是：当AvgLen达到上阈值时，温和的方法（丢弃一些分组）已经不起作用，因此要使用激进的方法，即丢弃所有到达的分组。有一些研究者提出建议，从随机丢弃到完全丢弃应有一个较平滑的过渡，而不应用这里所示的突变方法。

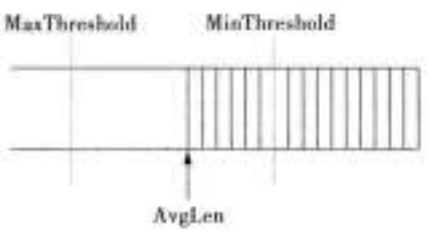


图6-16 在FIFO队列中的RED阈值

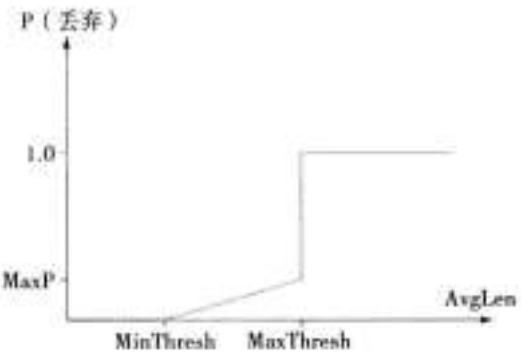


图6-17 RED的丢弃概率函数

尽管图6-17显示了丢弃概率只是AvgLen的函数，但实际情况要复杂一些。事实上，P是Avglen和上个分组被丢弃后距当前时间的函数。具体地说，它的计算公式如下：

$$TempP = MaxP \times (AvgLen - MinThreshold)/(MaxThreshold - MinThreshold)$$

$$P = TempP/(1 - count \times TempP)$$

TempP是图6-17中画在y轴上的变量。当Avglen在两个阈值之间时，count记录有多少到达的分组已经排入队列（未丢弃）。P随count值的增加而缓慢增加，由此使得丢弃最近一个分组后，丢弃分组的可能性随着时间的增加而增加。这使得间隔较小的丢弃比间隔较大的丢弃要少一些。计算P的这一额外步骤是由RED的发明者引入的，他们观察到，没有这一步，分组的丢弃就不能很好地按时间分布，而是趋于成群发生。由于来自某一连接的分组到达可能是突发式的，所以这种成群的丢弃可能使单条连接上的丢弃倍增。这不是我们所期望的，因为在每个RTT内只丢一个分组就足以让连接减少窗口尺寸，而丢弃多个分组则会让连接回退到慢启动。

作为例子，假设设置MaxP为0.02且count初始化为0。如果平均队列长度位于两个阈值中间，则TempP及P的初始值将是MaxP的一半（0.01）。当然，到达的分组有99%的可能进入队列。随着每个后续的分组不被丢弃，P缓慢增加，当50个分组都无丢

弃地到达时，P将加倍到0.02。在未必出现的99个分组无丢失到达的事件中，P达到1，保证下一个分组被丢弃。算法部分的重点是它确保丢弃大致随时间均匀分布。

当AvgLen超过MinThreshold时，如果RED仅丢弃了一小部分分组，那么将导致一些TCP连接减小它们的窗口尺寸，从而降低分组到达路由器的速率。如果一切正常，AvgLen将随之减少且拥塞得以避免。队列长度可能保持较短，且由于很少出现分组丢弃而使吞吐量保持较高水平。

注意，由于RED的操作是在一个随时间平均的队列长度上，瞬间队列长度可能会远大于AvgLen。在这种情况下，如果一个分组到达而没有地方放置，那么它将不得不被丢弃。一旦发生这种情况，RED将按队尾丢弃模式操作。RED的目标之一是只要可能就避免队尾丢弃操作。

RED的随机性赋予算法一个有趣的特性。由于RED随机地丢弃分组，所以RED决定丢弃某一特定流的分组的概率和这个流在路由器上获得的带宽份额成比例。这是因为当一个流发送分组的数量较多时，它就提供更多可供随机丢弃的候选分组。这样在RED中就有了某种公平资源分配的感觉，尽管这绝不是准确的。

结论 注意有大量的分析在探究如何设置各种RED参数，如MaxThreshold、MinThreshold、MaxP和Weight，即所有用于优化能力函数（吞吐量与延迟的比率）的参数。通过模拟对这些参数的性能进行了确认，结果表明算法对这些参数不是过分敏感的。但重要的是要记住，所有分析和模拟都是以网络工作负载的特性为转移的。RED真正的贡献是给出一种机制，通过这种机制，路由器可以更精确地管理它的队列长度。精确地定义构成最佳队列长度的因素取决于通信量的结合，而且仍是当前的研究主题，人们正在收集因特网中RED操作方案的实际信息。

考虑两个阈值MaxThreshold和MinThreshold的设置。如果通信量相当大，那么应将MinThreshold的值设置得足够大，使链路的利用率维持在可接受的高级别上。而且两个阈值的差值应大于在一个RTT中算出的平均队列长度的典型增长量。对当今因特网已知的通信量结合情况，将MaxThreshold设置为MinThreshold的两倍看来是合理的。此外，我们希望高负载期间平均队列的长度徘徊在两个阈值之间，应有多于MaxThreshold的足够空闲缓冲区来容纳因特网中自然产生的突发性分组，从而避免路由器强迫进入队尾丢弃模式。

上面我们提到，Weight决定动态平均低通滤波器的时间常量，它为我们如何选择一个合适的Weight提供线索。回想RED在拥塞期间通过丢弃分组向TCP流发送信号的情况。假设路由器丢弃了来自某一连接的一个分组，然后立即转发来自同一连接的更多的分组。当这些分组到达接收方时，接收方开始向发送方发出重复确认。当发送方看到足够的重复确认后，它将减少其窗口尺寸。于是从路由器丢弃一个分组开始到同一路由器减少其窗口尺寸而使相关连接的压力有所减轻为止，必须至少占用这个连接的一次往返时间。出现路由器对拥塞的响应时间远小于通过路由器时连接的往返时间的情况可能并不多。如前面指出的那样，100 ms是因特网上平均往返时间的一个不错的估计值。因此Weight应该这样选择：能滤出队列长度在时间上远小于100ms的变化。

由于RED通过给TCP流发信号告知它们减速的机制工作，因此你可能会想到，如果这些信号被忽略掉了会出现什么情况？这通常称为*无应答流*（unresponsive flow）问题，多年来人们一直在关注这个问题。无应答流使用超出自己公平份额的网络资源，如果这样的流足够多，将引起拥塞崩溃，就像在对TCP进行拥塞控制之前所发生的情况一样。6.5

节描述的一些技术通过把某些类型的通信量与其他通信量相区别来缓解这个问题。也可能使用一个 RED 的变形丢弃更多对它所发送的初始暗示不做应答的流，这将依然是一个活跃的研究领域。

6.4.3 基于源的拥塞避免

和前面两种依赖路由器上新机制的拥塞避免方案不同，我们现在描述的策略是从终端主机上检测拥塞的初始阶段（在丢失分组发生前）。首先我们对使用不同信息检测早期拥塞的一系列相关机制做一个回顾，然后再详细描述一种特别的机制。

这些技术的基本思想是从网络中观察这样的一些迹象：某台路由器的队列正在增加，如果不采取措施很快会发生拥塞。例如，源可能注意到随着网络中路由器分组队列的增大，可以测量到它发送的每个后继分组的 RTT 都会有增加。利用这种观察的一个特殊算法如下：拥塞窗口的正常增长就像在 TCP 中那样，但是每隔两个往返时间延迟，算法检查当前的 RTT 是否大于迄今为止所测到的最大和最小 RTT 的平均值。如果大于，那么算法将拥塞窗口减少 1/8。

第二种算法与上一种相似。是否改变当前窗口大小是根据 RTT 和窗口大小两个因素的变化而决定的。每隔两个往返时间延迟，根据以下乘积将窗口调整一次。

$$(\text{CurrentWindow}-\text{OldWindow})\times(\text{CurrentRTT}-\text{OldRTT})$$

若结果为正，源将窗口值减少 1/8；如果结果为负或为 0，源将窗口值增加 1 个最大分组长度。注意：在每次调整期间窗口均会改变，也就是说，它围绕最佳点来回摆动。

相关主题

Tahoe、Reno 和 Vegas

"TCP Vegas" 这个名称最早出现在以 4.3 BSD Unix 各种版本发布的 TCP 实现中，这些版本称为 Tahoe 和 Reno（同 Las Vegas 一样，Tahoe 和 Reno 都是美国内华达州的地名），同时 TCP 的版本也因 BSD 版本而知名。TCP Tahoe 也称为 BSD 网络版 1.0（BNR1），对应于 Jacobson 的拥塞控制机制的最初实现，它包括 6.3 节中描述的除快速恢复外的所有机制。TCP Reno 也称为 BSD 网络版 2.0（BNR2），加入快速恢复机制，以及一种称为分组首部预测（header prediction）的优化措施，用于优化段按序到达的常见情况。TCP Reno 也支持延迟确认（delayed ACK）——隔一个段确认一次而不是每个段确认一次，但这是可选的，有时会被关闭。在 4.4 BSD Unix 版本中使用的 TCP 版本加入了 5.2 节所述的 "大窗口" 扩展。

从 Linux 操作系统日益普及和研究人员数量的激增来看 TCP 拥塞控制，形势已变得相当复杂。Linux 提供了一系列 TCP 拥塞控制设置，包括 Vegas 选项和默认的 TCP 新变种 CUBIC 选项。用地名来命名 TCP 变种已成为风潮（参见 TCP-Illinois 和 TCP-Westwood）。

在对 TCP 家族的讨论中，应提出的一点是近几年中 TCP 成为变化很多的协议，尤其在拥塞控制机制方面。事实上，对于在哪个版本中引进哪一种技术，甚至无法完全达成一致，这是考虑了代码中间版本的可用性和补丁上面加补丁的结果。

可以肯定的是，任意两个遵循最初规范的 TCP 实现尽管应该能互操作，但不一定执行得很好。识别 TCP 变种间互操作的操作含义是一个难题。换句话说，你可以认为 TCP 不再用规范定义，而由一个实现来定义。唯一的问题是用哪个实现？

随着网络接近拥塞，可以看到另一种变化：发送速率趋于平缓。第三种方案利用了这一点。每个 RTT 内，它将窗口大小增加一个分组，同时将获得的吞吐量与加一个分组前的吞吐量进行比较。如果它们的差小于只有一个分组传送时（即和刚建立连接时一样）所达到的吞吐量的一半，那么算法将窗口减一个分组。这个方案通过用网络中未发送完的字节数除以 RTT 的值来计算吞吐量。第四种机制，我们将更详细地说明，类似上一种算法，它注重吞吐量的改变，更具体地说注重发送速率的改变。但和第三种算法中计算吞吐量的方法不同，它不查看吞吐量坡度的变化，而是将测量到的吞吐量变化率与理想的吞吐量变化率作比较。这种算法称为 TCP Vegas，虽然在因特网中并未广泛使用，但这种算法所使用的策略仍在继续研究中。（更多信息请参看“扩展阅读”。）

从图 6-18 中给出的标准 TCP 的轨迹中可直观地看出 Vegas 算法的思路。（对 TCP Vegas 命名的解释参见相关主题“Tahoe、Reno 和 Vegas”。）图 6-18 的顶部显示连接的拥塞窗口的变化，它与本节早先给出的轨迹表示的信息相同。图中间和底部描绘了新的信息：中间图显示在源上测量到的平均发送速率，底部图显示在瓶颈路由器上测到的平均队列长度。这三个图在时间上是同步的。在 4.5～6s 之间（阴影部分），拥塞窗口增加（顶部图），我们希望看到吞吐量也随之增加，然而吞吐量保持平缓（中部图）。这是因为吞吐量不可能超过可用的带宽。若超过这个点，窗口大小的增加只会导致分组在瓶颈路由器上占用缓冲空间（底部图）。

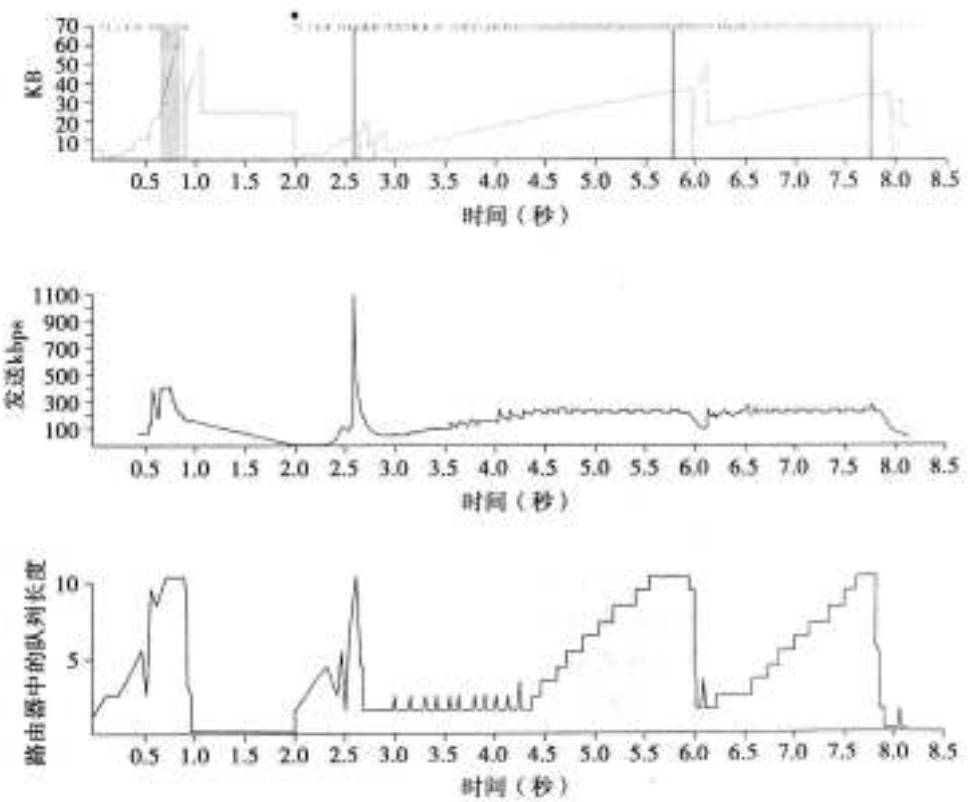


图 6-18 拥塞窗口与观测到的吞吐率（三幅图是同步的）。上图：拥塞窗口；中图：观测到的吞吐量；下图：路由器上占用的缓冲区。曲线＝拥塞窗口；黑点＝超时；细线标记＝每个分组被传送的时刻；竖条＝最终将被重传的分组第一次传送的时刻

有一个有用的比喻，说明图 6-18 所示的现象是在冰上驾驶。里程表（拥塞窗口）或许告诉你当前时速为 30 英里/小时，但从车窗向外看，人们是以步行的速度（测到的发送速率）通过你，你才得知你的时速最多 5 英里/小时。额外的能量被汽车轮胎（路由器缓冲区）吸收了。

TCP Vegas 用这种思想测量和控制连接正在传送的额外数据量，所谓“额外数据”，是指源为了让数据量与当前可用带宽匹配而发送的数据，而这些数据本不应该发送。TCP Vegas 的目标是将网络中的额外数据量维持在“正常的”标准上。显然，若源正在发送的额外数据量太多，则会引起长时间的延迟，并可能导致拥塞。其次，若源正在发送的额外数据过少，就不能快速响应可用网络带宽的增长。TCP Vegas 的拥塞避免行为除了根据丢弃分组外，还根据网络中额外数据的估计值的变化，现在我们详细地描述算法。

首先，定义一个指定流的 BaseRTT 为流无拥塞时分组的 RTT 值。实际应用中，TCP Vegas 将 BaseRTT 设为所有测得的往返时间的最小值，它通常是在路由器队列由于这个流产生的通信量而增加之前由连接发送的第一个分组的 RTT。如果假设连接尚未溢出，则所期望的吞吐量为

$$\text{ExpectedRate}=\text{CongestionWindow}/\text{BaseRTT}$$

其中 CongestionWindow 为 TCP 拥塞窗口，为讨论方便，假设它等于传送中的字节数。

第二，TCP Vegas 计算当前发送速率 ActualRate。做法如下：记录一个不同分组的发送时间，记下从发送该分组到收到确认这段时间内传送的字节数，当它收到确认信息时，计算该分组的样本 RTT，最后用传送字节数除以样本 RTT。上述计算过程每个往返时间内执行一次。

第三，TCP Vegas 比较 ActualRate 和 ExpectedRate，并相应地调整窗口。令 Diff=ExpectedRate−ActualRate。注意按照定义，Diff 的值应大于或等于 0，因为 ActualRate>ExpectedRate 意味着需将 BaseRTT 改变为最近采样的 RTT 值。再定义两个阈值 $\alpha<\beta$，它们分别大致对应网络中有过少和过多的额外数据。当 Diff$<\alpha$ 时，TCP Vegas 在下一个 RTT 中线性增加拥塞窗口，而当 Diff$>\beta$ 时，TCP Vegas 在下一个 RTT 内线性减少拥塞窗口。当 $\alpha<$Diff$<\beta$ 时，TCP Vegas 保持拥塞窗口不变。

从直观上我们可以看出，实际吞吐量和期望吞吐量的差距越大，网络中的拥塞越多，这意味着发送速率应当下降。由阈值 β 触发窗口减小。另一方面，当实际吞吐量与期望吞吐量过于接近时，连接处于不能充分利用可用带宽的危险中。这时阈值 α 触发窗口增加。总之，目标就是将网络中的额外字节数保持在 α 和 β 之间。

图 6-19 显示了 TCP Vegas 拥塞避免算法的过程。上图描绘拥塞窗口的变化过程，与本章给出的吞吐量的其他过程图显示相同的信息。下图显示期望吞吐率和实际吞吐率，它们控制如何设置拥塞窗口。下图是对算法工作原理的最好说明，上面的线表示 ExpectedRate，下面的线表示 ActualRate，宽的阴影带表示阈值 α 和 β 之间的区域；阴影带的顶部线与 ExpectedRate 相距 αKBps，阴影带的底部与 ExpectedRate 相距 βKBps。目标是将 ActualRate 保持在两个阈值之间，也就是在阴影区域内。当 ActualRate 在阴影区之下（即离 ExpectedRate 太远）时，TCP Vegas 减少拥塞窗口以免在网络中缓存太多的分组。同样，当 ActualRate 在阴影区域之上（即离 ExpectedRate 太近）时，TCP Vegas 增加拥塞窗口以免网络不能被充分利用。

如上所述，因为算法将实际吞吐率与期望吞吐率之间的差和阈值 α 与 β 比较，所以这两个阈值是通过 KBps 定义的。但若通过网络中一个连接占用多少额外的缓冲区定义或许更为准确。例如，某个连接的 BaseRTT 为 100ms，一个分组长度为 1KB，若 α= 30KBps

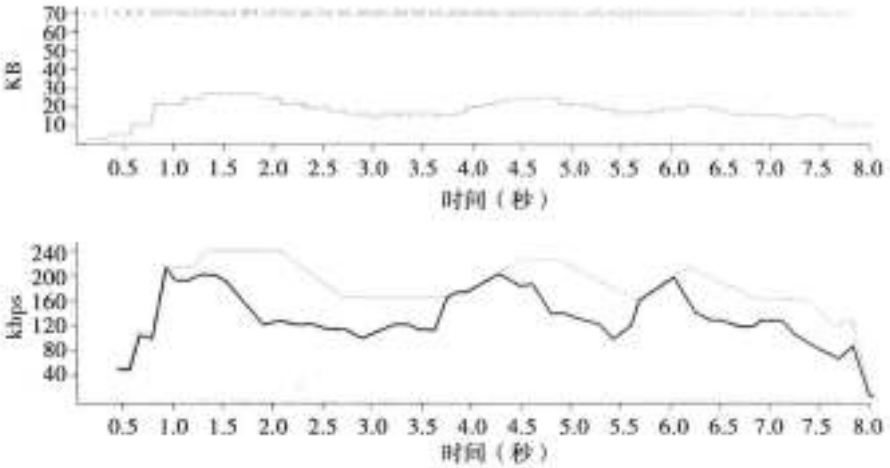


图 6-19 TCP Vegas 拥塞避免机制的图示。上图：拥塞窗口；
下图：期望吞吐量（上面的线）和实际吞吐量（下面的线）。
阴影区是在 α 和 β 阈值中间的区域

和 β=60KBps，那么我们可认为 α 说明连接在网络中至少应占用三个额外缓冲区，而 β 说明连接在网络中占用的额外缓冲区应不超过六个。在实际应用中，将 α 设为一个缓冲区和将 β 设为三个缓冲区较好。

最后，你会发现 TCP Vegas 线性减少拥塞窗口，看起来似乎与需要成倍减少以确保稳定性的规则冲突。对此的解释是，TCP Vegas 在发生超时时确实会使用成倍减少，而刚描述的线性减少应该是发生在拥塞出现和分组开始被丢弃之前的拥塞窗口中的及早减少。

相关主题

评价一种新的拥塞控制机制

假设你要开发一种新的拥塞控制机制并且想评估它的性能。例如，你或许想将它与当前用于因特网的机制进行比较。如何测量和评估你的机制？尽管曾有一段时间因特网的主要目的是支持网络研究，但是现在它已成为一个大型的实用网，因此完全不适于进行一项控制机制的实验。

如果你的方法是纯粹端到端的，即假设因特网中只有 FIFO 路由器，那么在几台主机上运行拥塞机制并测量连接能获得的吞吐量是可能实现的。但是在这里我们要加上一个警告。发明一个拥塞控制机制极其容易，它可以获得 5 倍于因特网的 TCP 吞吐量。你能以很高的速率简单地一下把大量的分组放入因特网，并由此引发拥塞。所有运行 TCP 的其他主机检测到这一拥塞并降低它们发送分组的速率。然后你的机制就可以开心地占用所有带宽。这种策略很快，但是它一点都不公平。

直接在因特网上做实验，即使非常小心，也会因你的拥塞控制机制涉及改变路由器而使实验失败。为了评估一种新的拥塞控制算法而改变运行在成千上万台路由器上的软件是完全不实际的。在这种情况下，网络设计者被迫在模拟网络或私用测试网络上测试他们的系统。例如，本章中用到的 TCP 轨迹是由运行在网络模拟器上的 TCP 应用程序产生的。无论是用模拟器还是用测试平台，提出一个代表真实因特网中的拓扑结构和通信量的典型实例都是一项挑战。

6.5 服务质量

多年以来，分组交换网一直许诺要支持多媒体应用程序，即结合音频、视频和数据的应用程序。音频和视频信息一旦数字化，终归要像其他数据格式一样作为比特流传输。要求高带宽的链路一直是实现这一许诺的障碍。然而，近来编码技术的发展降低了音频和视频应用程序对带宽的要求，同时链路的速度也提高了。

然而要在网络上传输音频和视频，除了提供足够的带宽外，还有更多的事情要做。例如，电话中的谈话双方希望能够这样交谈：一个人既能回答对方的说话又能让对方立即听到自己的说话。这样，传输的及时性就非常重要。我们称对数据的及时性敏感的应用程序为实时应用程序（real-time application）。音频和视频应用程序就是典型的例子，但是还有其他的例子，如工业控制——你想使发送给机器人手臂的一条命令在它撞向某物体之前到达。甚至文件传输应用程序也可能有及时性的限制，例如有这样的需求，第二天要用数据，应在头一天晚上将数据库中的更新全部完成。

关于实时应用程序的一个显著特点是：它们需要来自网络的某种保证，即数据可以按时（用某种“按时”定义）到达。反之，非实时应用能够使用端到端重传策略来确保数据正确到达，这种策略不能提供及时性：如果数据较晚到达，重传只是增加了总延迟。及时到达必须由网络本身（路由器）提供，而不是在网络边缘（主机）提供。因此我们得出结论，尽力服务模型对于实时应用程序来说是不够的，这种模型试着传输数据但不做任何许诺，并将清除操作留给边缘。我们需要的是一种新的服务模型，其中应用程序可以向网络要求更高的保证。然后网络可能通过提供它会做得更好的保证或者说明不能比此刻做得更好做出响应。注意，这样一个服务模型是当前模型的超集：在尽力而为服务下运行很好的应用程序应该能够使用新的服务模型，它们的需求不太严格。这意味着网络将区别对待一些分组，这是在尽力而为服务模型中没有做的。对于提供不同级别服务的网络，经常被说成是支持服务质量（QoS）的。

此刻你或许疑虑“互联网还没有支持实时应用程序吗？”我们多数尝试过像 Skype 这样的网络电话应用，它运转正常的原因一部分是尽力而为服务时常运行得很好。（Skype 公司还特别做了一些聪明的事情，试图解决互联网中缺乏的 QoS。）这里的关键词是“时常”。如果你需要实时应用程序使用的服务足够可靠，那么尽力而为服务（它不作任何保证）并不够可靠。我们稍后会探讨 QoS 的必要性。

6.5.1 应用需求

在考虑用于给应用程序提供服务质量的各种协议和机制之前，我们先试图了解这些应用程序需要什么。首先我们把应用程序分为两类：实时的和非实时的。后者有时称为传统数据（traditional data）应用程序，因为在传统的数据网络上它们一直是主要的应用程序，包括大多数流行的应用程序，如 Telnet、FTP、email、Web 浏览，等等。这些应用程序不需要保证数据的实时传输也能正常运行。非实时类应用程序的另一个术语是弹性（elastic），因为当它们面临延迟增加时，能有很好的伸展余地。注意，这些应用程序能够从更短的延迟中获益，而当延迟增加时它们也不会变得不可用。还要注意，它们的延迟要求从像 Telnet 这样的交互式应用变化到像 email 一样更加异步的应用，中间有像 FTP 这样的交互式大容量传输。

1. 实时音频实例

作为实时应用程序的一个具体例子，考虑类似于图 6-20 所示的一个音频应用程序。通过从麦克风采样而生成数据，并且用模数（A→D）转换器将它们数字化。将数字样本放在分组中，分组是通过网络传输的，在另一端接收分组，接收主机必须以适当的速率播放（playback）数据。例如，如果语音样本是以每 125μs 的速率被收集的，那么应该以相同的速率对它们进行播放。这样，我们可以认为每个样本都有一个特定的播放时刻（playback time）：在接收主机上恰好需要在这个时刻来播放。在这个语音的例子中，每个样本的播放时刻比前一个样本晚 125μs。如果数据在它相应的回放时刻之后到达，无论是因为在网络延迟，还是因为被丢弃并重传，它基本上都是无用的。实时应用程序的特点是迟到的数据会完全失去价值。在弹性应用程序中，如果数据按时到达可能会很好，但如果不是这样我们仍然能使用它。



图 6-20 一个音频应用

使我们的语音应用程序能工作的一种方法是，确保所有样本用完全相同的时间通过网络。那么，由于样本以每 125μs 一个的速率输入，所以它们将以相同的速率出现在接收端，为播放做好准备。然而，通常很难保证所有通过分组交换网络的样本恰好经历相同的延迟。分组遇到交换机或路由器中的队列，这些队列的长度随着时间而变化，这意味着延迟趋向于随时间而变化，结果是音频流中每个分组的延迟都可能不同。在接收方处理这一问题的方法是缓存一定量的预备数据，因此总是为等待在正常时刻播放的分组提供存储。如果一个分组延迟很短一段时间，那么它会进入缓冲区直到它的播放时刻到来。如果它延迟很长时间，那么在播放之前，它不需要在接收方的缓冲区中存储很长时间。这样，我们已经有效地给所有分组的播放时间增加了一个常数偏移量作为一种保证方式。我们称这个偏移量为播放点（playback point）。我们遇到的唯一的麻烦是，如果分组在网络中延迟了过长的时间以至于它们在其播放时刻之后才到达，这将使得播放缓冲区被用尽。

播放缓冲区的操作如图 6-21 所示。左边的斜线表明以固定速率产生的分组。波浪线表明分组什么时候到达，即它们被发送后的时间变化量，取决于网络中实际情况。右边的斜线表明在进入播放缓冲区一段时间之后以固定速率播放的分组。只要播放线在时间轴上足够靠右，应用程序就绝不会注意到网络延迟的变化量。然而，如果我们将播放线向左移动一点，那么一些分组将会由于到达太晚而变得无用。

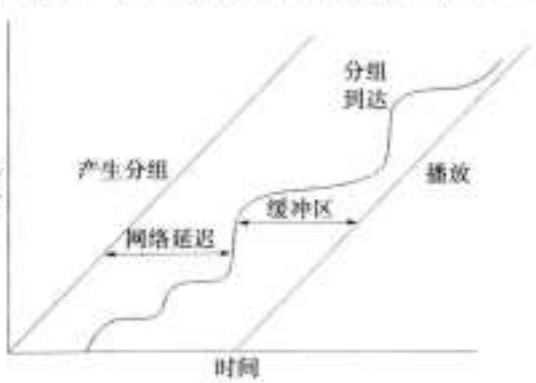


图 6-21 回放缓冲区

对于音频应用程序，我们能对播放数据延迟多久设置限制。如果你说话的时刻与你的听众听到的时刻之间超过 300ms，那么谈话就很难继续下去。因此，这种情况下我们想

从网络得到的保证是我们的所有数据在300ms之内到达。如果数据早到达，我们将它放到缓冲区直到恰当的播放时刻到来再播放它。如果数据晚到，那么它对我们没有用而必须将它丢弃。

为更好地评价网络延迟是怎样变化的，图6-22显示了某一天的一段时间里在因特网的某条路径上测量到的单向延迟。尽管确切的数值会随路径和日期有所变化，但是这里的关键因素是延迟的可变性，它在任何路径上的任何时间里都可以看得到。正如图上部的累积百分比所表示的，97%的分组有100ms或小于100ms的延迟。这意味着如果我们的音频应用实例把播放点设在100ms，那么平均说来，每100个分组中有三个分组将因到达太晚而变得无用。关于此图要注意的重要一点是曲线的尾部（即它向右延伸的长度）非常长。我们必须把播放点设在200ms以确保所有分组及时到达。

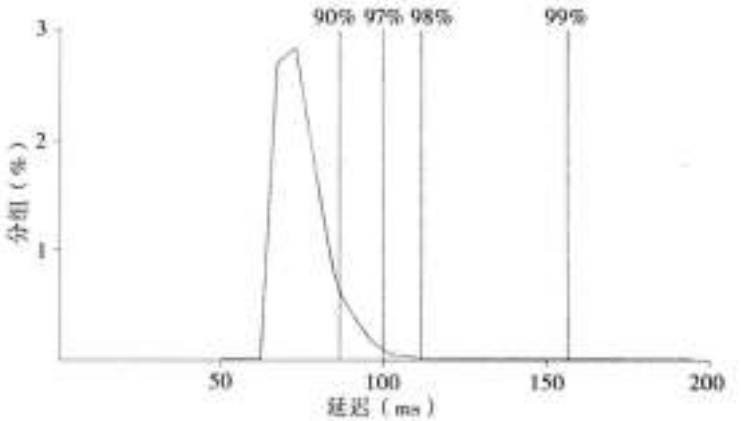


图6-22 一个因特网连接延迟分布的例子

2. 实时应用分类

至此我们对实时应用如何工作有了一个具体的认识，接下来可以看看一些不同类型的应用，它们能帮助我们进一步认识服务模型。下面的分类方法大部分要归功于Clark、Braden、Shenker和Zhang的工作，他们有关这一问题的文章可以在本章的“扩展阅读”中找到。在图6-23中总结了应用的分类。

我们所依据的第一个对应用程序进行分类的特性是它们对丢失数据的容忍度，其中发生“丢失”可能是由于一个分组到达太晚而不能播放，或是由于网络中正常的原因引起的。一方面，一个丢失的音频样本可以根据周围的样本插值，这对感觉到的音频质量只有很小的影响。仅当越来越多的样本丢失时，质量会下降到使语音变得不可理解的程度。另一方面，机器人控制程序可能是一个不能容忍丢失的实时应用的例子——丢失了包含在指示机器人手臂停止命令中的分组是不能接受的。这样，我们就能根据实时应用是否能够容忍偶尔的丢失，把它们分成容忍型（tolerant）或非容忍型（intolerant）。（附带指出，注意许多实时应用比非实时应用更能忍受偶然的丢失。例如，将我们的音频应用和FTP进行比较，文件中一位不



图6-23 应用分类

正确的丢失就可能使整个文件变得完全无用。）

第二种方法是根据适应性对实时应用进行分类。例如，一个音频应用可能适应分组通过网络时经历的延迟量。如果注意到分组几乎总是在发送后300ms之内到达，那么就能相应地设置播放点，给少于300ms到达的分组分配缓冲区。假设随后看到所有分组都是在发送后100ms之内到达。如果把播放点提前到100ms，那么应用的用户可能感觉到改进。这个改变播放点的过程实际上需要我们在某段时间之内以递增的速率将样本播放完。对于语音应用程序，只是通过缩短单词间的无声时间，就能以一种勉强可理解的方式做到这一点。于是，在这种情况下，播放点调整就相当容易，并且在几个语音应用程序中已经被有效地实现，例如像vat这样的音频电话会议程序。注意，播放点调整能够在任一方向上发生，但是实际上这样做会使调整期间被播放的信号失真，并且这种失真的影响将在很大程度上取决于终端用户怎样使用数据。通常，非容忍型应用不能容忍这种失真，就像它们不能容忍分组的丢失一样。

观察一下，如果依据所有分组将在100ms之内到达的假设设置我们的播放点，就会发现一些分组稍晚一些到达，我们就不得不丢弃它，而如果把播放点保留在300ms，就不必丢弃它们。这样，只有当它具有明显的优点，并且有证据表明迟到分组的数目小到能被接受的程度时，才可以把播放点提前。当有观察到的近期历史记录或有来自于网络的某种保证，就可以这样做。

我们称能够调节播放点的应用为延迟可适应的（delay-adaptive）应用。另一类可适应的应用是速率可适应的（rate-adaptive）应用。例如，许多视频编码算法能够权衡比特速率与质量。这样，如果发现网络能够支持某一带宽，就能相应地设置编码参数。如果以后有了更多的可用带宽，则可以改变参数以提高质量。

3. 支持QoS的方法

考虑到各种应用需求的广阔空间，我们所需要的是一个应用范围更广的服务模型，以满足任何一种应用需求。这促使我们提出一个服务模型，不仅提供一种类型的服务（尽力而为服务），而是具有多类服务，其中的每一类都能满足某个应用集合的需求。为了达到这个目标，我们现在准备考察人们为了提供一定范围的服务质量而开发出来的一些方法。这些方法可以分为两大类：

- 细粒度（fine-grained）方法，它给单独的应用程序或流提供QoS。
- 粗粒度（coarse-grained）方法，它给多类数据或成块通信量提供QoS。

在第一类里我们发现综合服务（Integrated Services），一种IETF开发的QoS体系结构，常常与资源预定协议（Resource Reservation Protocol，RSVP）相关。ATM的QoS方法也属于这一类。第二类中有区别服务（Differentiated Services），写本书的时候，它可能是最广泛配置的QoS机制。接下来的两节我们将依次讨论它们。

最后，正如本节的开端所述，给网络中增加QoS支持对于实时应用程序来说并不是必须的。作为讨论的结束语，我们再回到这个问题：在不依赖于广泛使用的QoS机制（如综合服务或区分服务）的情况下，为了更好地支持实时流，端主机可能做什么。

6.5.2 综合服务（RSVP）[⊖]

术语综合服务（常常简称为IntServ）是指在1995～1997年前后由IETF所做的主要

⊖ 参见"实验十二"。

工作。IntServ 工作组开发了大量的服务类（service classes）规范来满足上述一些应用类别的需要。它还定义了如何利用 RSVP 通过这些服务类来做预定。下面概要地介绍这些规范及实现它们所用的机制。

1. 服务类

有一个服务类是为非容忍型应用设计的。这些应用要求分组永远不推迟到达，网络应该保证任意分组所经历的最大延迟具有某一特定值，这样应用程序就能设定它的播放点，以便没有分组会在它的播放时刻之后到达。我们假设总能通过缓存方法处理早到的分组。这种服务通常称为有保证的（guaranteed）服务。

除了有保证的服务，IETF 还考虑了其他几种服务，但最终确定了一种满足容忍型、可适应型应用需求的服务。这种服务称为受控负载（controlled load），它的产生是因为我们观察到现存的这种类型的应用在负载不重的网络中运行得相当好。例如，音频应用程序 vat，随着网络延迟的变化调整它的播放点，只要丢失率保持在小于或等于 10%的水平，就能产生相当好的音频质量。

受控负载服务的目的是为请求这种服务的应用仿真一个轻负载的网络，即使网络作为一个整体的实际负载可能是很重的。它的技巧在于使用像 WFQ（见 6.2 节）这样的排队机制将受控负载通信量和其他通信量分开，同时使用某种容许控制的形式来限制一条链路上受控负载通信量的总量，从而保持合理的低负载。下面我们更详细地讨论容许控制。

显然，这两种服务类型只是可能提供的所有服务类型的一个子集。事实上，其他服务从未标准化为 IETF 工作的一部分。迄今，上述两个服务（包括传统的尽力而为服务）已被证明足够灵活，可满足广泛的应用需求。

2. 机制概述

既然我们已经用几个新的服务类扩充了尽力而为服务模型，下一个问题就是怎样实现能给应用提供这些服务的网络。本节概括其关键机制，阅读本节时要记住，所描述的机制仍有待因特网设计团体进行认真的推敲，这里只是对上面概括的支持服务模型涉及的各个部分做一个总体的讨论。

首先，尽力而为服务只能告诉网络我们想让分组去哪里并把它留在那里，而实时服务更多的是告诉网络一些关于我们所要求服务的类型信息。我们可以给它定性信息，如“使用受控负载服务”，或定量信息，如“我需要的最大延迟为 100ms”。除了描述我们想要的服务类型，还需要告诉网络我们打算将什么注入网络，因为低带宽应用比高带宽应用需要更少的网络资源。我们提供给网络的信息的集合称为流说明（flowspec）。这个名字来自于这样的想法：与单一的应用相联系的并具备共同需求的分组集合称为流（flow），这与 6.1 节中所用的术语相一致。

第二，当我们请求网络提供一个特殊的服务时，网络需要判断实际上它是否能够提供该服务。例如，如果 10 个用户请求一个服务，其中每个用户将持续使用 2Mbps 的链路容量，而且所有用户都共享一条 10Mbps 容量的链路，那么网络就不得不拒绝某些用户。决定何时拒绝的进程称为容许控制（admission control）。

第三，我们需要一种机制，通过这种机制网络用户与网络自身的组件之间互相交换信息，如请求服务、流说明和容许控制的决定。这在 ATM 中称为信令（signalling），但由于这个词有多种含义，我们称这个进程为资源预定（resource reservation），通过资源预定

协议来完成。

最后，当已经描述流和它们的需求并且也做出了容许控制的决定时，网络交换机和路由器需要满足流的需求。对于在交换机和路由器中的传输，满足这些需求的关键部分是管理分组排队和调度的方式。这个最后的机制是分组调度（packet scheduling）。

3. 流说明

可将流说明分为两个部分：描述流的通信量特性的部分（TSpec）和描述向网络要求的服务部分（RSpec）。RSpec 是一种特有的服务，并且相对来说容易描述。例如，对于受控负载服务，RSpec 是很平常的：应用程序要求受控负载服务时只要求不附加参数。对于有保证的服务，你能够指定延迟目标或界限。（在 IETF 有保证的服务的规范中，你指定的不是延迟而是另一个用来计算延迟的量。）

TSpec 要复杂一些。正如上面的例子所表明的，需要给网络提供关于流所使用的带宽的足够信息，以允许做出智能的容许控制决定。然而，对于大部分应用，带宽不仅仅是一个数字，它是一个不断变化的东西。例如，一个视频应用在场景快速变化时比静止时每秒产生更多的比特。下面的例子表明，只知道长期平均带宽是不够的。假设我们有 10 个流到达独立输入端口上的交换机且都留在同一个 10Mbps 链路上。假设经过某一合适的时间间隔，每个流预期能够发送不超过 1Mbps 的数据。你可能以为这没有出现问题。然而，如果这些是像压缩视频这样的可变比特速率的应用，那么它们的发送速率偶尔会高于平均速率。如果足够的源端以高于它们的平均速率发送，那么数据到达交换机的总速率将大于 10Mbps。这种额外的数据在发送到链路上之前会被放在队列中。这种情况持续越久，队列就会越长。分组可能不得不被丢弃，而且即便没有分组丢弃，位于队列中的数据也会延迟。如果分组被延迟的时间足够长，那么将不能提供所请求的服务。

下面我们讨论的是如何准确地管理队列，以控制延迟和避免丢弃分组。然而，这里要注意，我们需要知道源端的带宽怎样随着时间而改变。一种描述源端带宽特性的方法称为令牌桶（token bucket）过滤器。这种过滤器由两个参数描述：令牌速率 r 和桶深度 B。它的工作过程如下。为了能够发送一个字节，必须有一个令牌。为了发送长度为 n 的分组，就需要 n 个令牌。开始没有令牌，然后以每秒 r 的速度累积令牌，且累积的令牌不超过 B 个。这意味着，能够以尽可能快的速度向网络连续发送 B 个字节，但是经过足够长的时间间隔，每秒发送的字节就不能超过 r 个。结果表明，当试图指出它是否能接纳一个新的服务请求时，这个信息对于容许控制算法是非常有帮助的。

图 6-24 说明了令牌桶怎样刻画流的带宽需求。为简单起见，假设每个流能够按单独的字节而不是按分组发送数据。流 A 以 1MBps 的稳定速率产生数据，因此可以用速率 r=1MBps 和桶深为 1 字节的令牌桶过滤器来描述它。这意味着流 A 以 1MBps 的速率接收令牌，但它存储的令牌不能多于 1 个——它立即将它们用掉。流 B 也长期以平均 1MBps 的速率发送，但它是以 0.5MBps 的速率发送 2s，然后以 2MBps 的速率发送 1s。因为令牌桶速率 r 在某种意义上是一个长期的平均速率，所以流 B 能够用速率为 1MBps 的令牌桶来描述。然而，与流 A 不同，流 B 至少需要 1MB 的桶深 B，使它在以 2MBps 发送时能够使用以小

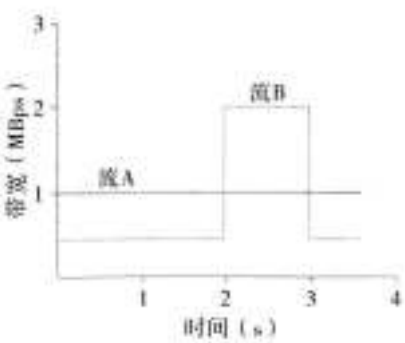


图 6-24 具有相同平均速率和不同令牌桶描述的两个流

于1MBps的速率发送时储存起来的令牌。在本例中的前2s，它以1MBps的速率接收令牌，但却仅仅以0.5MBps的速率来消耗，所以它能够储存2×0.5=1MB的令牌，然后在第三秒以2MBps的速率发送时消耗它们（以及在这1s中继续增加的新令牌）。第三秒结束时，它已经消耗了额外的令牌，又以0.5MBps的速率发送，再次开始储存令牌。

值得注意的是单个流能够用许多不同的令牌桶描述。作为一个小例子，流A能够用与流B相同的令牌桶描述，桶B具有1MBps的速率和1MB的桶深。它实际上从不需要累积令牌，这一事实并没有使它成为一个不准确的描述，但这确实意味着我们没有向网络传达一些有用的信息——实际上流A与它的带宽需求高度一致。总的来说，应用的带宽需求越清楚越好，这样可以防止网络资源的过度分配。

4. 容许控制

容许控制的思想很简单：假设给定当前的可用资源，当某个新的流想要接收一个特殊级别的服务时，容许控制查看流的TSpec和RSpec，并试图确定所要求的服务是否能得到所需数量的通信量，在当前可利用的资源条件下，不会使任何前面被容许的流接收到比它所要求的更差的服务。如果它能够提供服务，则流被容许；否则被拒绝。难点在于指出何时说是，何时说否。

容许控制完全依赖于请求服务的类型和路由器中使用的排队规则，在本节的后面将讨论后一个问题。对于有保证的服务，你需要用一个好算法做出明确的是/否容许的决定。如果每台路由器都使用6.2节讨论的加权公平排队法，决定将是相当容易的。对于受控负载服务，决定可能基于试探法，例如“上次我允许带有这一TSpec的流进入这个类，该类的延迟超出了可接受的界限，所以我最好说否”或者“我当前的延迟离界限还很远，我容许另一个流应该不会有困难”。

容许控制不应该与策略制定（policing）相混淆。前者是对每个新的流做出的是否容许的决定。后者是应用在每一个分组基础上的功能，保证一个流与用于实现预定的TSpec相一致。如果一个流与它的TSpec不一致——例如，它每秒发送的字节是它所说要发送字节的两倍——那么，这很可能干扰向其他流提供的服务，而且必须进行改正。有多种改正方式，显然可以丢弃违规的分组。然而，另一种方式是检验分组是否真的干扰其他流的服务。如果它们不干扰，就将分组加标记后继续发送，标记为“这是一个不一致的分组，如果你需要丢弃分组，先丢弃它。”

容许控制与策略（policy）这一重要问题密切相关。例如，网络管理员可能希望允许其公司的首席执行官所做的预定被接纳，而下层雇员所做的预定被拒绝。当然，如果首席执行官请求的资源不可用，那么他的预定请求也可能失败，所以我们看出，当做出容许控制的决定时，规则和资源可用性问题可能都是需要考虑的。在写本书时，互联网技术中策略的应用正成为一个备受关注的领域。

5. 预定协议

虽然面向连接的网络总需要某种创建协议以便在交换机中建立必要的虚电路状态，但是像因特网这种无连接的网络就没有这种协议。然而，正如本节所指出的，当我们想从网络中获得实时服务时，我们需要向它提供更多的信息。尽管已经为因特网提出了大量的创建协议，但当前最受关注的是RSVP。它格外吸引人，因为它与传统的面向连接网络的信令协议本质上截然不同。

基础RSVP的一个关键假设是，它不应该损害在当今无连接网络的健壮性。因为无连接的网络很少或不依赖于网络本身存储的状态，所以在端到端的连接依然保持的情况下，路由器崩溃或重启以及链路通或断都是可能的。RSVP试图通过在路由器中使用软状态（soft state）的思想来维持这种健壮性。当不再需要软状态（相对于面向连接网络中的硬状态）时，不需要明确地删除它们。相反，如果它没有被周期性刷新，那么就会在某个相当短的时间（如1分钟）后超时。我们将在后面看到这怎样有助于健壮性。

RSVP的另一个重要特性是支持多播流，使其像单播流一样有效。这并不奇怪，因为多播应用（如vat和vic）显然是较早受益于实时服务的应用。RSVP设计者观察到多播应用的接收方比发送方多得多，典型的例子是只有一个讲演者和大量听众的演讲。并且，接收方可能有不同的需求。例如，一个接收方可能想要接收仅来自一个发送方的数据，而其他接收方可能想接收来自所有发送方的数据。与其让发送方记录潜在的大量接收方，不如让接收方记录它们自己的需要更有意义。这就意味着RSVP采用面向接收（receiver-oriented）的方法。与之相反，面向连接的网络通常将资源预定留给发送方，正如在电话网中，一般情况下是由打电话的人引发资源分配的。

RSVP的软状态和面向接收的性质给予它许多良好的特性，特性之一是非常直接地增加或减少提供给接收方的资源分配级别。由于每个接收方定期发送刷新消息以保持相称的软状态，所以很容易发送一个请求新级别资源的新的预定。此外，软状态可以从容地处理网络或节点的潜在故障。万一主机崩溃，主机分配给一个流的资源会自然地超时并释放。为了看出在路由器或链路失败时会发生什么，我们需要仔细研究进行预定的机制。

首先，考虑一个发送方和一个接收方试图为它们之间的通信量获得预定的情况。在接收方能够进行预定之前需要发生两件事。第一，接收方需要知道发送方可能发送的通信量，以便能够进行相应的预定。即它需要知道发送方的TSpec。第二，它需要知道分组从发送方到接收方遵循什么路径，以便能在路径上的每台路由器建立一个资源预定。可以通过从发送方向包含TSpec的接收方发送一条消息来满足这两个需求。显然，这将TSpec给了接收方。所发生的另一件事是当这条消息（称为PATH消息）通过时，每台路由器查看它，并确定反向路径（reverse path），用于在尝试获得路径中每台路由器预定的过程中，发送从接收方返回给发送方的预定。建立多播树首先是通过在4.2节中描述的机制实现的。

收到PATH消息后，接收方对RESV消息中的多播树向上发回一个预定。这条消息包含发送方的TSpec和描述接收方请求的RSpec。路径上的每台路由器查看预定请求并分配必要的资源来满足它。如果可以做预定，那么RESV请求被传给下一台路由器。如果不能，则向发出请求的接收方返回一个出错消息。如果一切正常，那么就在发送方和接收方之间的每台路由器上安装了正确的预定。只要接收方想保留预定，它就大约每30s发送一次相同的RESV消息。

现在我们可以看看当路由器或链路失败时会发生什么。路由选择协议将适应这种失败并生成一条从发送方到接收方的新路径。PATH消息大约每30s发送一次，如果路由器在它的转发表中检测到有变化发生，可能还会更快地发送一个PATH消息。因此新路由稳定后的第一个PATH消息将通过新路径到达接收方。接收方的下一个RESV消息将遵循新的路径，如果一切顺利，接着在新的路径上建立新的预定。同时，不在新路径上的路由器将停止获得RESV消息，并且它们的预定将超时并释放。这样只要路由变化不过于频繁，RSVP就能很好地处理拓扑结构中的变化。

我们需要考虑的下一件事是怎样处理多播，其中可能有多个发送方组成一个组并有多个接收方。图 6-25 说明了这种情况。首先，让我们处理对一个发送方的多个接收方。当一个 RESV 消息对多播树向上传时，很可能碰上树的一个其他接收方已经建立的预定。可能有这样的情况：这一点的上游所预定的资源足以为这两个接收方服务。例如，如果接收方 A 已经为小于 100ms 的有保证的延迟进行了预定，并且来自接收方 B 的新的请求是小于 200ms 的延迟，那么就不需要新的预定。另一方面，如果新的请求是小于 50ms 的延迟，那么路由器将首先需要看一看它是否能够接受这个请求。如果能，则再向上游发送请求。下次接收方 A 请求最小值为 100ms 的延迟，路由器就不需要传递这个请求。一般说来，可以以这种方式合并预定以满足合并点下游的所有接收方的需要。

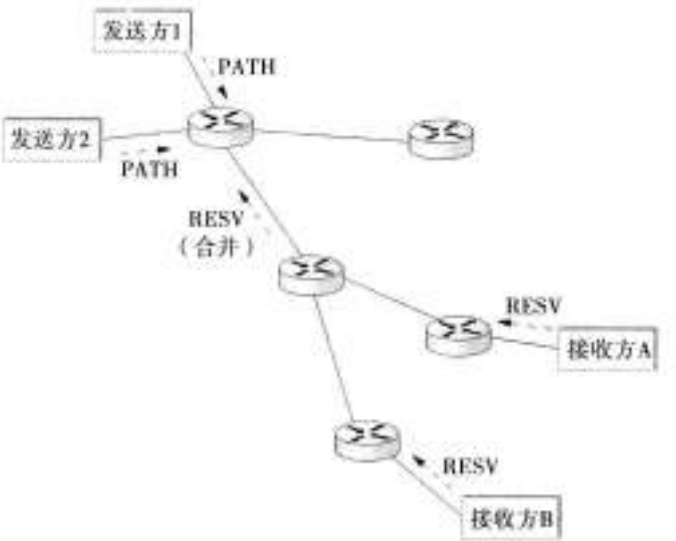


图 6-25　在一个多播树上进行预定

如果树上还有多个发送方，那么接收方需要收集所有发送方的 TSpec，并进行足够大的预留以容纳所有发送方的通信量。然而，这可能并不意味着需要把 TSpec 累加起来。例如，在一个 10 人的音频会议中，并不是多点分配足够资源来传送 10 个音频流，因为 10 个人同时讲话的结果是不能理解的。这样，我们能够设想一个足以容纳最多两个讲话者的预定。从所有发送方的 TSpec 计算出正确的总的 TSpec，显然要根据具体应用而有所不同。而且，我们可能只有兴趣听到所有可能讲话者中一部分人的话。RSVP 在处理下列选项时有不同的预定“风格”：“为所有讲话者预定资源”，“为任意 n 个讲话者预定资源”，“只为讲话者 A 和 B 预定资源”。

6. 分组的分类和调度

一旦我们已经描述了通信量和想要的网络服务并且为路径上的所有路由器安装了合适的预定，路由器便只剩下一件工作——向数据分组实际提供所要求的服务。其中需要做两部分工作：

- 将每个分组与适当的预定相关联以便正确处理它，这个进程称为分类（classifying）分组。
- 管理队列中的分组以便它们能接收所要求的服务，这个进程称为分组调度（scheduling）。

第一部分是通过检查分组中的最多 5 个字段（源地址、目的地址、协议号、源端口和目的端口）进行的。（在 IPv6 中，分组首部中的 FlowLabel 字段可能用于根据一个较短关

键字进行查找。）根据该信息，能够将分组放在适当的类中。例如，它可能被归类到受控负载的各个类中，或者可能是需要独立于所有其他有保证的流而单独处理的有保证流的一部分。简而言之，存在一个从分组首部中流说明信息到一个确定如何在队列中处理分组的类标识符的映射。对于有保证的流，这可能是一个一对一的映射，而对于其他服务，它可能是多对一的映射。分类的细节与队列管理细节紧密相关。

应该明确，路由器中像FIFO队列这样简单的排队机制不足以提供许多不同的服务和每个服务中不同级别的延迟。6.2节讨论了几个更复杂的队列管理规则，而且它们的某种组合很可能用在路由器中。

理想状况下，分组调度的细节不应该在服务模型中说明。相反，这是实现者能够尝试进行创造性活动以有效实现服务模型的一个领域。在有保证服务的情况下，已经可以使加权公平排队规则（其中共享链路一定份额的每个流都有自己单独的队列）提供一个容易计算的有保证的端到端延迟界限。对于受控负载，可以使用更简单的方案。一种可能性包括：把所有受控负载通信量看作单个聚集的流（就涉及的调度机制而论），该流的权值可以根据可控负载中允许的通信总量来设置。如果你在一台路由器中考虑它，这个问题就更困难，许多不同的服务很可能以并发的方式提供，这些服务的每一个服务都可能需要一个不同的调度算法。这样，需要用某个综合的队列调度算法管理不同服务之间的资源。

它们现在在哪？

RSVP和综合服务配置

在本书写作时，RSVP和综合服务体系结构并没有被广泛配置，其主要原因是本节末所描述的可扩展性问题。实际上，它被认为是一种“死”的技术。然而，认为RSVP和综合服务已经死亡却为时过早。

与IntServ分离开，作为一个为流量工程（正如4.3节中的描述）建立MPLS路径的协议，RSVP已经被广泛配置了。由于该原因，因特网中的大多数路由器有一些RSVP实现。然而，这可能是在本书写作时因特网中RSVP配置的全部内容。对RSVP的这种使用方式完全独立于IntServ，但它至少说明协议本身是可配置的。

有证据表明RSVP和IntServ在它们首次提出后10年多的时间里可能得到一次被广泛应用的机会。例如，IETF正在对RSVP进行标准化扩展，以便使其支持集合预定——在过去RSVP和Intserv的扩展性已经提升了。在商业产品中RSVP作为一个资源预定协议正在逐渐得到支持。

多种因素会导致RSVP和IntServ在不久的将来被更多地采纳。首先，需要QoS的应用，例如IP上的语音、实时视频会议和娱乐视频，比10年前要广泛得多，因此就更需要复杂的QoS机制。第二，容许控制——使得当资源不足时网络能够对应用程序说“不”——对于除非资源足够否则不能很好运行的应用程序来说是一个好的机制。例如，大部分IP电话用户宁可听到忙信号，也不愿意进行无法接受的坏质量的呼叫。网络操作者宁可给一个用户发送一个忙信号，也不愿意给大量用户提供差的质量。第三个因素是新应用程序的大量资源需求，例如高精度的视频传输，因为它们需要高带宽才能工作好。因此去建立偶尔能说“不”的网络比提供足够带宽去满足所有可能的应用需求具有更高的性价比。然而这是一个复杂的平衡，关于容许控制的价值，以及将RSVP和IntServ作为工具来提供的争论可能会持续一些时间。

7. 可扩展性问题

虽然综合服务体系结构和 RSVP 象征着 IP 的尽力而为服务模型的重大发展，但是因特网服务提供商们觉得这并不是他们想要采用的正确模型。这种保留和 IP 基本的设计目的之一（即可扩展性）有关。在尽力而为服务模型中，因特网中的路由器几乎或根本不存储任何流经它们的各个流的状态。这样，随着因特网的增长，路由器要跟上这种增长所需做的唯一事情就是每秒传送更多的比特和处理更大的路由表。但是 RSVP 增大通过路由器的每个流要做相应预定的可能性。为了理解这个问题的严重性，假设在一条 OC-48 (2.5Gbps) 的链路上的每个流都是 64kbps 的音频流。这样的流的数量为

$$2.5\times10^9/64\times10^3=39\ 000$$

这些预定中的每一个都需要某个量的状态，这些状态存储在内存中，并且定时刷新。路由器需要对其中的每个流进行分类、控制和排队。每当这样的流请求一个预定时，就需要做一次容许控制决定。而且需要一些“拒绝”用户的机制，使他们不能做长时间的任意大的预定。[⊖]

在写本书时，可扩展性涉及的这些问题已经阻碍了综合服务的广泛采用。由于这些问题，人们已经开发了不需要做这么多“每个流”状态的其他方法。下一节将讨论一些这样的方法。

6.5.3 区分服务 (EF、AF)[⊖]

综合服务体系结构给单独的流分配资源，而区分服务模型（常常简称为 DiffServ）给少数几类通信量分配资源。实际上，已经提出的几个区分服务方法简单地把通信量分为两类。这是一个非常明智的方法：如果你考虑一下网络操作员在试图保持尽力而为服务在互联网平稳运行方面的困难，那么在服务模型上做少量增加是有意义的。

假设我们决定通过仅增加一个称之为“加价”的新类来改进尽力而为服务模型。显然，我们需要一些方法指出哪些分组是加价分组，哪些分组是常规的尽力而为服务类型。与其让 RSVP 这样的协议来告诉所有路由器某一个流正在发送加价分组，不如只让这些分组在到达时向路由器表明身份，这样要简单得多。当然，通过使用分组首部的 1 位就可以做到：如果这一位设置为 1，表示分组是加价分组；如果为 0，表示分组是尽力而为服务类型。记住这一点，需要解决两个问题：

- 谁设置加价位？在什么环境下设置？
- 当路由器看到设置该位的分组时，做哪些不同的处理？

对第一个问题有许多可能的回答，但最常用的方法是在管理的边界设置这一位。例如，在一个因特网服务提供商的网络边缘的路由器可能对与某一特定公司网络相连的接口上到达的分组设置这一位。因特网服务提供商可能这样做是因为，这个公司已经为比尽力而为服务更高级别的服务付费。也可能并不是所有分组都被标识为加价，例如，可以配置路由器，让其将分组标识为加价直至某一最大速度，同时使所有额外分组都保持尽力而为服务。

⊖ 为每个预定付费是“拒绝”用户的一种方法，这与针对每次呼叫来付费的电话模型是类型的。这不是“拒绝”用户的唯一方法，为每个呼叫付费是电话网络中的主要计费方式。

⊖ 参见“实验附录 B”。

假设分组已经用某种方法进行标识，那么路由器将如何处理遇到的标识过的分组？这里又有很多答案。实际上，IETF 的区分服务工作组正在对一系列应用于标识分组的路由器行为进行标准化。它们被称为每跳行为（Per-Hop Behaviors，PHB），这个术语表明它定义的是单台路由器的行为而不是端到端服务的行为。因为新行为不止一个，所以在分组首部中需要不止一位来告诉路由器用哪个行为。IETF 已经决定从 IP 首部中去掉还未广泛使用的 TOS 字节，并重新定义它。这个字节中的 6 位已分配给区分服务代码指针（Diff-Serv Code Points，DSCP），其中每个 DSCP 是一个 6 位的值，标识用于一个分组的特定 PHB。（ECN 使用其余 2 位，参见 6.4.2 节。）

1. 加速转发（EF）PHB

要说明的最简单的 PHB 之一是加速转发（Expedited Forwarding，EF）。对标识为 EF 分组的处理应当是路由器以最少的延迟和丢失转发。使路由器能向所有 EF 分组都保证这点的唯一方法是：严格限制 EF 分组到达路由器的速率小于路由器转发 EF 分组的速率。例如，有 100Mbps 接口的路由器需要确保目标为该接口的 EF 分组的到达速率绝不超过 100Mbps。可能还需要确保这个速率比 100Mbps 稍低一些，以便路由器还可以偶尔有时间发送其他分组（如路由更新）。

通过在管理域的边界上配置路由器，容许进入域中的 EF 分组达到某一最大速率，可以获得对 EF 分组的速率限制。一个虽然保守但简单的方法是，确保进入域中的所有 EF 分组的速率的和低于域中最慢链路的带宽。这就能确保即使在最糟的情况下（所有 EF 分组都集中在最低速的链路上）也不会超载，同时能提供正确的行为。

对于 EF 行为，有几个可能的实现策略。一个策略就是给 EF 分组赋予严格高于其他所有分组的优先级。另一个策略是在 EF 分组和其他分组之间执行加权平均排队，将 EF 的权值设置得足够大，使所有 EF 分组可以快速传输。比起严格的优先级，这种方法有一个优点：即使在 EF 通信量过多的情况下，仍可以确保非 EF 分组获得一定的链路访问。这可能意味着 EF 分组不能准确地获得指定的行为，但是它能在 EF 通信量过载时，防止基本的路由通信量被阻止在网络之外。

2. 确保转发（AF）PHB

确保转发（Assured Forwarding，AF）基于界内界外 RED（RED with In and Out，RIO）或加权 RED 方法。这两种方法是对 6.4.2 节中基本 RED 算法的改进。图 6-26 显示了 RIO 是如何工作的，和图 6-17 一样，我们看到随着 x 轴上平均队列长度的增加，y 轴上的丢弃概率也增加。但是现在，对于两类通信量，我们有两条单独的丢弃概率曲线。很快就会明白，RIO 为什么称这两类为“界内”和“界外”。因为“界外”曲线的 MinThreshold 比“界内”曲线的小，所以很显然在低级别拥塞时，只有标志“界外”的分组被 RED 算法丢弃。如果拥塞变得更严重，将有更多的“界外”分组被丢弃，然后如果平均队列长度超过 Minin 时，RED 就开始连“界内”分组也丢弃了。

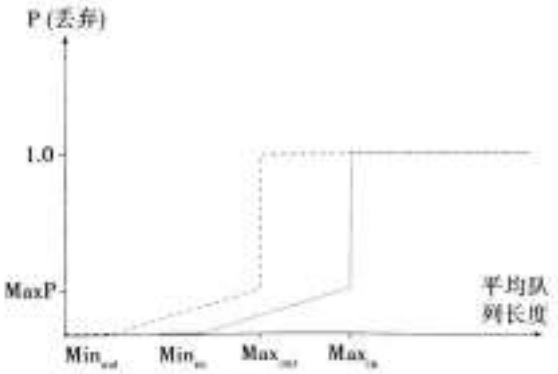


图 6-26 具有“界内”和“界外”丢弃概率的 RED

之所以称两类分组为“界内”和“界外”，是源于分组的标识方式。我们已经注意到，标识分组由管理域边缘的路由器来执行。可以把这台路由器看作是在网络服务提供商和该网络某一客户之间的边界上。客户可能是其他的网络，例如，某个公司的网络或是另一个网络服务提供商的网络。客户和网络服务提供商在确保服务的某种描述问题上达成一致（或许是客户为这种描述向网络服务提供商付费）。这个描述可能是这样的：“允许客户 X 最多发送 y Mbps 可确保的通信量”，也可能要复杂得多。不论描述是什么，边缘路由器都会清楚地将来自该客户的分组标识为“界内描述”或“界外描述”。在上面提到的例子中，只要客户的发送速率低于 y Mbps，那么它的所有分组都标识为“界内”，但是如果它超过了这个速率，额外的分组将被标识为“界外”。

在边缘的轮廓仪和在服务提供商网络中所有路由器上的 RIO 都应尽力确保（但不是保证）客户的分组能在其轮廓下传送。实际上，如果大部分分组是“界外”分组的话（其中包括由那些没有为建立描述而额外付费的客户发送的分组），RIO 机制会尽量保持拥塞较低以便使“界内”分组很少被丢弃。显然，必须有足够的带宽，使得单独的“界内”分组不会让链路拥塞到 RIO 开始丢弃“界内”分组的地步。

和 RED 一样，RIO 这种机制的有效性一定程度上依赖于正确的参数选择，对于 RIO 来说，需要设置更多的参数。在写本书时，这种方案在现有的网络中如何工作还是一个存在争议的问题。

RIO 的一个重要的性质是它不改变“界内”和“界外”分组的顺序。例如，如果一个 TCP 连接正将分组送给一个轮廓仪，于是一些分组被标识为“界内”，而另一些分组被标识为“界外”，这些分组将在路由器队列中获得不同的丢弃概率，但它们会按与发送相同的顺序到达接收方。这对于大多数 TCP 实现都很重要，尽管它们能处理无序到达的分组，但是当分组按序到达时，它们执行起来要好得多。还要注意，当分组到达顺序改变时，像快速重传这样的机制可能会被错误地触发。

可以推广 RIO 思想使它提供两条以上的丢弃概率曲线，这种思想基于一个称为加权 RED（WRED）的方法。在这种情况下，DSCP 字段的值用于在多条丢弃概率曲线中选择一条曲线，以便能提供多种不同类别的服务。

它们现在在哪？

区分服务的低调成功

在 2003 年，许多人断言区分服务已经走向死亡。在那年的 ACM SIGCOMM 年会（最有影响的网络研究会议）上，一个具有醒目名字“RIPQoS”的小组成立了——该小组的官方名字是“重访 IP 服务质量”，在小组声明中清楚地暗示 QoS 可能准备安详地休息了。然而，正如马克吐温讽刺他的死亡报道太夸张了，似乎 IP 服务质量的死亡，尤其是区分服务的死亡也太夸张了。

导致人们对区分服务持悲观看法的主要原因是它没有被因特网服务提供商配置到任何重要的领域。不仅如此，而且没有任何 QoS 机制使 IP 电话和视频流这样的实时应用在因特网上工作得很好，这使人怀疑是否需要任何 QoS。一定程度上这是许多因特网服务提供商提供的高带宽链路和路由器的配置，特别是在 20 世纪 90 年代后期网络蓬勃发展的那几年。

为了清楚区分服务成功在哪里，需要看看 ISP 的骨干网。例如，已经配置了 IP 电话的公司——本书写作时有数以千万计的企业级 IP 电话在使用中——通常为音频媒体分组使用 EF 行为，以确保在和其他流量共享链路时不会有延迟。同样的方法用于许多基于 IP

的住宅语音服务：仅仅在住宅之外的上行链路获得优先权（例如，DSL链路较慢的方向）。通常对于语音端点为EF建立DSCP，对于连接到宽带链路上的用户的路由器使用区分服务，以便对那些分组有较低的延迟。甚至有一些大的国家电话公司也已经把传统的语音服务移植到了IP网络上，区分服务提供了保护语音服务质量的方法。

除受益于区分服务的语音之外，还有一些其他应用，特别是商业数据服务。毋庸置疑，在未来几年中基于IP的视频将逐步成熟，这将提供又一个驱动因素。总的来说，两个因素使区分服务的配置非常重要：应用程序对QoS保证的高要求，以及链路带宽足以为所有流量提供QoS保证的缺乏。了解区分服务是非常重要的，像任何其他QoS机制一样，它不能创造带宽——它所能做的只是确保带宽被优先分配给有更大QoS需求的应用程序。

提供区分服务的第三种方法是在由6.2.2节描述的加权公平排队调度程序中用DSCP的值决定将分组放入哪个队列中。正如一个非常简单的例子所示，我们可以用一个代码指点指示尽力而为服务（best-effort）队列，用第二个代码点选择加价（premium）队列。然后我们需要为加价队列选一个权值，使加价分组获得比尽力而为服务分组更好的服务。这依赖于加价分组提供的负载。例如，如果令加价队列的权值为1，令尽力而为服务队列的权值为4，确保加价分组可获得的带宽为

$$B_{\text{premium}}=W_{\text{premium}}/(W_{\text{premium}}+W_{\text{best\_effort}})=1/(1+4)=0.2$$

就是说，我们有效地为加价分组预定了链路的20%。这样如果加价通信量提供的负载平均为链路的10%，那么加价通信量就好像运行在一个负载非常轻的网络中，因而服务也会非常好。特别是，因为在这种情况下WFQ试图让加价分组一到达就被发送，所以加价分组的延迟可以一直保持较低。另一方面，如果加价通信量的负载为30%，那么它活动时就好像运行在一个负载很重的网络中，而加价分组的延迟将非常高，甚至比所谓的尽力服务分组更糟。因此，了解提供的负载和谨慎地设置权值对这类服务而言很重要。但是要注意，安全的方法在为加价队列设置权值时非常保守。如果这种权值设置比期望的负载高得多，就有可能出错，并且也不能避免尽力而为服务通信量使用已被加价预定但未被加价分组使用的那部分带宽。

和在WRED中一样，我们可以推广基于WFQ的方法，允许由不同代码点表示的两个以上的类。而且可以把一个队列选择器和一种丢弃选择结合起来。例如，用12个代码点，我们可以有4个带不同权值的队列，每个队列有3种丢弃选择。这正是IETF在"确保服务"的定义中所做的事情。

相关主题

ATM服务质量

当前ATM技术的重要性不如10年以前，但其实际贡献是在QoS领域。在一些方面，ATM具有相当强的QoS能力的事实激励着IP中QoS的发展。这也有助于早期人们对ATM的采纳。

在很多方面，ATM网络提供的服务质量性能与使用综合服务的IP网络提供的服务质量性能是类似的。但是与IETF的三种服务类型相比，ATM标准化组织共提出五种服务类型。[⊖]这五种ATM服务类型是：

⊖ 我们把尽力而为看作是一种服务类型，具有可控制的负载和有保证的服务。

- 固定比特速率（CBR）。
- 可变比特速率——实时（VBR-rt）。
- 可变比特速率——非实时（VBR-nrt）。
- 可利用比特速率（ABR）。
- 未指明的比特速率（UBR）。

大部分 ATM 和 IP 的服务类型是相当类似的，但其中一类 ABR，在 IP 中没有真正与之相对应的类。下面我们将详细地解释这个类。VBR-rt 和 IP 综合服务中的有保证服务类非常相似。用于建立 VBR-rt VC 的确切参数和用于做有保证服务预定的参数略有不同，但基本思想是相同的。源端产生的通信量由令牌桶表示，同时指定穿过网络需要的最大总延迟。

除了希望 CBR 通信量的源端以不变的速率传送以外，CBR 也类似于有保证的服务。注意事实上这是 VBR 的一个特例，其中源端的峰值速率和传送的平均速率相等。

VBR-nrt 和 IP 的受控负载服务有些类似。此外，源通信量由令牌桶描述，但在延迟的保证方面，它不像 VBR-rt 和 IP 有保证服务那么严格。UBR 是 ATM 的尽力而为服务。

最后，我们来看 ABR。它不仅是一个服务类型，还定义了一系列拥塞控制机制。它是比较复杂的，所以我们这里只涉及其中几个要点。

ABR 机制在虚电路上运作是通过在 VC 的源和目的之间交换一种称为资源管理（RM）信元的特殊 ATM 信元。发送 RM 信元的目的是将网络中有关拥塞状态的信息回送给源端，使它能以合适的速率发送通信量。就这方面而言，RM 信元是一种明确的拥塞反馈机制。这与 DECbit 类似（见 6.4.1 节），但与靠丢弃分组来检测拥塞的 TCP 隐式反馈机制相反。它也类似于 6.3.2 节中描述的用于 TCP 的新的快启动机制。

最初，源端向目的端发送信元，其中包含它想发送数据信元的速率。路径上的交换机查看该请求速率，并决定在保证其他正在传送的通信量基础上，是否有足够的可用资源来处理这一速率。如果有足够的资源可用，RM 信元将不加改动地被传送；否则，在该信元传送前将减少请求的速率。在目的节点，RM 信元将回转并发还给源端，以便让源知道它能以多大的速率发送。

ABR 允许源增加或减少其分配率，增加还是减少由当下条件决定。即以高于或低于要求的速率的周期性地发送 RM 信元。如果不使用源规定的发送速率，它就会随着时间流逝而衰减。通常，交换机中的特定算法决定 RM 信元穿过交换机时的速率。该算法借鉴了各种信息（如当前缓冲占用率）测得当前活跃 VC 的到达速率。这些算法是正规的拥塞控制算法，力求最大限度地提高吞吐量，并保持低延时和低损耗。

目前在实际的网络中 ATM 使用得并不多，关于 ATM 服务质量人们感兴趣的问题是在不同技术之上有多少机制。ATM 和 IP 服务质量中的机制包括允许控制、调度算法、令牌桶、显式拥塞反馈机制等。

6.5.4 基于等式的拥塞控制

我们通过回顾 TCP 拥塞控制的全过程来结束对 QoS 的讨论，但这一次是在实时应用的背景下。回想一下，TCP 调整发送方的拥塞窗口（从而调整发送速率）以响应 ACK 和超时事件。这种方法的优点在于，它不需要网络上路由器的配合；它是一种纯粹基于主机

的策略。这种策略补充了我们所考虑的 QoS 机制，这是因为：①在 QoS 机制广泛采用之前，各类应用可以使用目前的这种基于主机的解决方案；②即使区分服务被全部采用，路由器队列仍有可能超出预定，如果发生这种情况，我们还是希望实时应用以合理的方式做出反应。

虽然我们想要利用 TCP 的拥塞控制算法，但是 TCP 本身并不适合实时应用。一个原因是 TCP 是可靠的协议，而实时应用经常不能忍受由重传引入的超时。然而，如果将 TCP 从它的拥塞控制机制中分离出来，也就是说，给一个不可靠的协议（像 UDP）加入类似 TCP 的拥塞控制，那么会发生什么情况？实时应用能够利用这样的协议吗？

一方面，这是一个有吸引力的想法，因为它会引发实时流与 TCP 流的公平竞争。目前所用的一种替代策略是视频应用使用 UDP，不用任何形式的拥塞控制，结果是 TCP 在拥塞时让路，允许 UDP 侵占 TCP 流的带宽。另一方面，TCP 拥塞控制算法的锯齿行为（见图 6-9）不适合实时应用：锯齿意味着应用传输的速率总是上上下下。相反，实时应用若在相当长的一段时间内能够维持一个平稳的传输速率，则工作得最好。

是否可能达到两个方面的完美结合：一方面为了公平面与 TCP 拥塞控制兼容，一方面为了应用而维持平稳的传输速率？最近的工作表明，答案是肯定的。尤其是，已经提出了多种所谓的 TCP 友好的拥塞控制算法。这些算法有两个主要目标。第一个目标是缓慢适应拥塞窗口。这一点是通过采用相对长的时间段（如一个 RTT）而不是基于每个分组来实现，使得传输速率得以平滑。第二个目标为在公平竞争 TCP 流的意义下是 TCP 友好的。通过把流的行为附加到建模 TCP 行为的等式中可以保证这一特性得到满足。因此，这种方法有时称为基于等式的拥塞控制（equation-based congestion control）。

在 6.3 节中我们看到过一个关于 TCP 吞吐率的简单等式，有兴趣的读者可以参阅本章末参考文献中的论文以获取关于其完整模型的细节。对我们来说，关注下面这个通用形式的等式就足够了：

$$\text{Rate} \propto \left(\frac{1}{\text{RTT} \times \sqrt{p}} \right)$$

这个等式表明，为了使 TCP 友好，传输速率必须与往返时间（RTT）和丢弃速率（p）的平方根成反比。换言之，为了从这个关系中构造一个拥塞控制机制，接收方必须定期地向发送方报告当时的丢弃速率（例如，接收方可能报告最后 100 个分组中有 10%没有收到），然后发送方上下调整发送速率，保证等式关系继续成立。当然，这还取决于应用对这些可用速率变化的适应性，但是正如我们将在下一章看到的，许多实时应用的适应性是相当强的。

6.6 小结

就像我们刚才看到的，资源分配问题不仅是计算机网络的重点，也是一个很难的问题。本章讨论了资源分配的两个方面。第一个方面是拥塞控制，它考虑的是当主机申请的资源超过网络可以提供的资源时防止全面的服务降级。第二个方面是给不同的应用提供不同的服务质量，这些应用需要的保证比尽力而为服务模型提供的保证要多。

大多数拥塞控制机制是针对当今因特网的尽力而为服务模型设计的，其拥塞控制的主要职责落到网络的端节点上。典型情况下，源使用反馈（从网络中隐式地获得或由路由器显式发送）来调整它加在网络上的负载。这正是 TCP 的拥塞控制机制所做的工作。

路由器实现一种控制哪些分组可以传输而哪些分组被丢弃的排队规则，这完全不依赖于端节点正在做的工作。有时这个排队算法可以复杂到能隔离通信量（例如 WFQ），而在其他情况下，路由器试图监视它的队列长度，然后在拥塞将发生时发送信号给源主机（例如，RED 网关和 DECbit）。

时下的服务质量方法所做的工作比拥塞控制多得多。它们的目标是使对延迟、丢失和吞吐量有着各种需求的应用程序所提出的请求都能通过网络中的新机制得到满足。综合服务（IntServ）方法允许单独的应用流使用显式的信令机制向路由器说明它们的需要（RSVP），而区分服务（DiffServ）将分组指定到路由器上接收区分处理的多种不同的类中。虽然区分服务是目前使用较为广泛的方法，但公网的部署中还是鲜见区分服务和综合服务。

接下来会发生什么：网络重构

也许我们应该提出的更大的问题是，我们可以对网络有哪些期望和最终有多少责任落到端主机上。在提供更多不同服务质量方面，基于预定的策略明显优于现今使用的基于反馈的策略，能提供不同质量的服务是将更多的功能放在网络路由器上的主要原因。是不是这就意味着像 TCP 这样的端到端拥塞控制的日子到了屈指可数的地步？当然不是。TCP 和使用 TCP 的应用地位都很牢固，而且在很多情况下它们不需要从网络中获得更多的帮助。进一步说，在因特网这样世界范围内的异构网络中，所有路由器都准确地实现相同的资源预定机制是不可能的。最终，看来网络中的端节点至少在一定程度上必须靠自己。毕竟，我们不能忘记支撑因特网的正确设计原则是在路由器上做最简单的事，而把所有复杂的事放在你可以控制的边缘上。除了资源分配问题，观察上述方面在未来的几年中如何发展将会非常有趣。

从某种意义上来讲，区分服务方法代表网络中的绝对最少智能和综合服务网络中需要的高智能（同时存储状态信息）之间的折衷方法。综合服务和 RSVP 的部署只在非常有限的环境（主要是在企业网络）中才能找到。一个重要的问题是区分服务方法是否能够满足更苛刻的应用需求。例如，如果一个服务提供商试图提供在 IP 网络上大范围的视频点播服务，区分服务技术是否足以达到传统的电视观众所期望的服务质量？随着网络中各种不同智能程度的变化，看来似乎需要开发更多的 QoS 选项。

在公共互联网上，一些趋势似乎指向分管其 QoS 需求的终端系统。例如，Skype 的成功一部分是靠免费服务使用户愿意忍受无 Qos 保证。从技术的角度来看，速度自适应（Skype 有可变比特率语音编解码器）和覆盖路由（第 9 章中谈到的技术）的结合虽然缺乏网络支持，却已经能够提供有效的传输质量。同时，在用户更为苛刻的企业环境中，部署网络级 QoS 支持的发展日趋复杂。

扩展阅读

这一章的推荐阅读清单很长，反映了人们在拥塞控制和资源分配方面做了很多有意义的工作。它包括本章讨论的引进各种机制的原始论文。如果想要了解上述机制的具体细节，包括关于这些机制的有效性和公平性的完整分析，这些文章是不可不看的，因为它们给出了有关拥塞控制的各种相互影响的观点。另外，第一篇文章给出了一个很好的关于本主题的早期工作的概述，而最后一篇则是关于在因特网上开发 QoS 功能的重要文章之一。

- Gerla, M., and L. Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications* COM-28 (4): 553-573, April 1980.
- Demers, A., S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. *Proceedings of the SIGCOMM'89 Symposium*, pages 1-12, September 1989.
- Jacobson, V. Congestion avoidance and control. *Proceedings of the SIGCOMM'88 Symposium*, pages 314-319, August 1988.
- Floyd, S., and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1 (4): 397-413, August 1993.
- Clark, D., S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *Proceedings of the SIGCOMM'92 Symposium*, pages 14-26, August 1992.
- Parekh, A. and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Transactions on Networking* 2 (2): 137-150, *April* 1994.

除了上面推荐的文章，还有大量关于资源分配的有价值的资料。对于初学者，Kleinrock [Kle79] 和 Jaffe [Jaf81] 这两篇早期的文章奠定了使用能力作为衡量拥塞控制有效性度量的基础。另外，Jain [Jai91] 对和性能评估有关的各种问题给出了彻底的讨论，包括 Jain 的公平性指标的描述。

有关 TCP Vegas 的更多细节可以在 Brakmo 和 Peterson [BP95] 的文章中找到，后续的工作在 Low 等 [LPW02b] 的文章中有所描述。关于 TCP 拥塞控制算法的改进在下列文章将继续探讨。高延迟带宽积下的 TCP Vegas 的扩展 FAST TCP 在 [WJLH06] 文章中讨论。Ha 等人 [HRX08] 的文章描述了 Linux 下的 TCP 变种 CUBIC。Floyd 的文章中详述了一种实验 IETF 标准：HighSpeed TCP 的规范。类似 6.4 节介绍的各种拥塞避免技术可以在 Wang 和 Crowcroft [WC92, WC91] 以及 Jain [Jai89] 的文章中找到，第一篇文章还给出一个很好的拥塞避免方法的综述，它是基于人们对网络接近拥塞时如何变化的共识给出的。一些人对 RED 算法提出了改进，这包括 Lin 等人 [LM97] 的文章中描述的流 RED (FRED)。

Ramakrishnan, Floyd 和 Black 在 [RFB01] 中详细说明了 ECN 建议标准。Stoica 等人的 [SSZ98]，Low 等人的 [LPW+02a] 以及 Katabi 等人的 [KHR02] 以活动队列管理的形式推广了上述思想。Katabi 的文章介绍了 XCP，这是一个新提出的传输层协议，在高带宽环境下改善了 TCP 的吞吐量。

更近的关于分组调度的工作扩展了上面引用的最初的公平排队的文章。很好的例子参见 Stoica 和 Zhang [SZ97]，Bennett 和 Zhang [BZ96]，以及 Goyal, Vin 和 Chen [GVC96] 的文章中。

许多已出版的其他文章是关于综合服务体系结构的，这些文章包括 Braden 等人的 [BCS94] 的概述和 Zhang 等人的 [ZDE+93] 关于 RSVP 的描述。讨论区分服务这一主题的第一篇文章是 Clark 的 [Cla97]，这篇文章介绍 RIO 机制和区分服务总的体系结构。其后 Clark 和 Fang 的 [CF98] 给出一些模拟结果。Blake 等人的 [BBC+98] 定义了区别服务体系结构，Davie 等人的 [DCB+02] 定义了 EF 每一跳的行为。想了解遵循互联网上尽力而为服务时实时应用程序的耗费，我们推荐 Chen [CHHL06] 等人撰写的 Skype 用户

满意度这篇论文。

最后，除了上述针对传统数据应用程序特性而改进 TCP 的工作，人们又提出了多个 TCP 友好的拥塞控制算法，并为适应实时应用做了修改。在 Floyd 等人的［FHPW00］，Sisalem 和 Schulzrinne 的［SS98］，Rhee 等人的［ROY00］以及 Rejaie 等人的［RHE99］中包括这些算法。这些算法建立在 Padhye 等人的［PFTK98］早期提出的基于等式模型的 TCP 吞吐量基础上。一个 TCP 友好的吞吐量控制协议已经被 IETF［HFPW03］定义了，在 UDP 中增加拥塞控制的想法已经导致了数据报拥塞控制协议（DCCP）［KHF06］的产生。

习题

1. 可以基于主机到主机或基于进程到进程定义流。
   (a) 讨论每种方法对应用程序的含义。
   (b) IPv6 包含一个 FlowLabel 字段，用于向路由器提供有关单独流的线索。源主机将在这个字段放一个用于标识流的其他所有字段的伪随机散列值，这样路由器可以使用这些位中的任意子集作为散列值来快速查找流。FlowLabel 字段基于这两种方法的哪一个？
2. TCP 使用以主机为中心的、基于反馈的和基于窗口的资源分配模型。如何设计 TCP 以替代下列模型？
   (a) 以主机为中心的、基于反馈的和基于速率的模型。
   (b) 以路由器为中心的和基于反馈的模型。
3. ★ 对于以太网，如第 2 章中的习题 51，平均分组大小为 5 个时间段，而且当有 N 个站点试图发送时，直到有一个站点发送成功的平均延迟为 $N/2$ 个时间段。分别画出以吞吐量、延迟和能力作为负载函数时的曲线图。吞吐量按其最大值的百分比计量，负载按在任意时刻同时准备发送数据的站点的数目（N）计量（有些不真实）。注意，这意味着总有一个站点准备发送（除非 $N=0$，但这种情况可忽略）。假设每个站点在一个时刻只有一个分组要发送。
4. 假设两台主机 A 和 B 通过一台路由器 R 相连。A 到 R 的链路带宽为无限大，R 到 B 的链路每秒可以发送一个分组。R 的队列为无限大。负载按每秒从 A 到 B 发送的分组数计量。画出吞吐量对负载和延迟对负载的图；如果画不出图，解释原因。计量负载是否有更恰当的方法？
5. TCP Reno 是否可能到达这样的状态：拥塞窗口尺寸远大于 RTT×带宽（例如它的 2 倍）？这可能吗？
6. 考虑图 6-27 中主机 H 和路由器 R 及 R1 的排列。所有链路都是全双工的，而且所有路由器都比它们的链路快。说明路由器 R1 不会出现拥塞，同时对于其他的任何路由器 R，我们可以找到一种只让该路由器拥塞的通信模式。

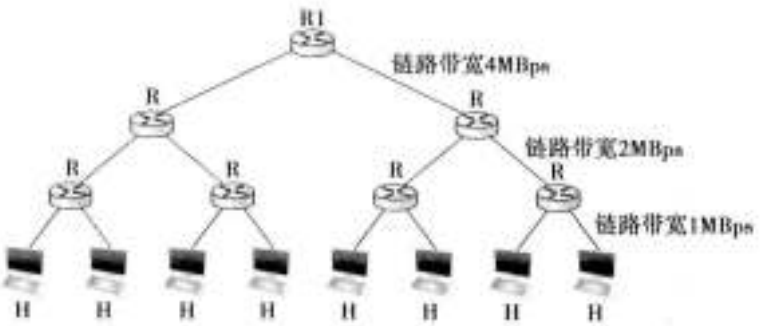


图 6-27 习题 6 的图示

7. 假设一个拥塞控制方案可以使一组竞争的流获得下列的吞吐率：200KBps，160KBps，110KBps，95KBps 和 150KBps。

(a) 计算这种方案的公平性指数。

(b) 在这些流中再加入一个吞吐率为1 000KBps的流，重新计算公平性指数。

8. 在公平排队中，值 F_i 被解释为一个时间戳：第 i 个分组完成传送的时刻。给出加权公平排队中的 F_i 的解释，并根据 F_{i-1}、到达时间 A_i、分组大小 P_i 以及为该流指定的权值 w_i，给出 F_i 的公式。

9. 举例说明公平排队实现中的非抢先策略如何导致与按位轮转服务不同的分组传送顺序。

10. 假设一台路由器有三个输入流和一个输出。它收到表6-1所列的分组，它们几乎在同一时间内以所列顺序到达，其间输出端口忙，但所有队列是空的。在下列情况下，给出分组传送的顺序：

(a) 公平排队。

(b) 加权公平排队，流2的权值为4，其他两个流的权值为1。

11. 假设一台路由器有三个输入流和一个输出。它收到表6-2所列的分组，它们几乎在同一时间内以所列顺序到达，其间输出端口忙，但所有队列是空的。在下列情况下，给出分组传送的顺序：

(a) 公平排队。

(b) 加权公平排队，流2的权值是流1的2倍，流3的权值是流1的1.5倍。注意，按照流1、流2、流3的顺序。

表6-1 习题10的分组

分 组	大 小	流
1	100	1
2	100	1
3	100	1
4	100	1
5	190	2
6	200	2
7	110	3
8	50	3

表6-2 习题11的分组

分 组	大 小	流
1	200	1
2	200	1
3	160	2
4	120	2
5	160	2
6	210	3
7	150	3
8	90	3

12. 假设路由器的丢弃策略是，当队列满的时候丢弃开销最大的分组，其中分组“开销”的定义是分组长度与它在队列中的剩余（remainning）时间的乘积。（注意，在计算开销时，用前面的分组长度之和代替剩余时间，结果是等价的。）

(a) 与队尾丢弃相比，这一策略有哪些利弊？

(b) 给出排队分组序列的一个实例，其中丢弃最大开销分组的顺序不同于丢弃最大分组的顺序。

(c) 给出一个例子，其中两个分组随着时间的伸延其相应开销的顺序发生互换。

13. 两个用户，一个用Telnet而另一个用FTP发送文件，都通过路由器R将其通信量送出。R的输出链路太慢以至于两个用户的分组一直留在R的队列中。如果R对这两个流使用如下的排队策略，讨论Telnet用户看到的相对性能。

(a) 轮转服务。

(b) 公平排队。

(c) 修改后的公平排队，其中我们只计算数据字节的开销，不计算TCP和IP首部的开销。

只考虑输出通信量。假设Telnet分组有1字节数据，FTP分组有512字节数据，并且所有分组有40字节的首部。

14. 考虑一台路由器，它管理着三个流，在下列时钟时刻每个流都有固定大小的分组到达：

流A：1，2，4，6，7，9，10。

流B：2，6，8，11，12，15。

流C：1，2，3，5，6，7，8。

所有三个流共享同一条输出链路，路由器在该链路上每个时间单元只能传送一个分组。假设有一个无限大的缓冲区空间。

(a) 假设路由器实现公平排队。每个分组被路由器传送时，给出挂钟时间。到达时间的关系将按 A、B、C 的顺序解决。注意，挂钟时间 $T=2$ 就是公平排队时钟时间 $A_i-1.5$。

(b) 假设路由器实现加权公平排队，其中对流 A 和流 B 给出相等的容量份额，对流 C 给出的容量是流 A 的 2 倍。每个分组被传送时，给出挂钟时间。

15. 考虑一台路由器，它管理着三个流，在下列挂钟时刻每个流都有固定大小的分组到达：

流 A：1，3，5，6，8，9，11。

流 B：1，4，7，8，9，13，15。

流 C：1，2，4，6，7，12。

所有三个流共享同一条输出链路，路由器在链路上每个时间单元只能传送一个分组。假设有一个无限大的缓冲区。

(a) 假设路由器实现公平排队。每个分组被路由器传送时，给出挂钟时间，到达时间的关系将按 A、B、C 的顺序解决。注意，挂钟时间 $T=2$ 就是公平排队时钟时间 $A_i-1.333$。

(b) 假设路由器实现加权公平排队，其中对流 A 和流 C 给出的容量份额相等，对流 B 给出的容量是流 A 的 2 倍。每个分组被传送时，给出挂钟时间。

16. 假设 TCP 实现一种扩展，允许窗口大小远大于 64KB。假设你用这一扩展 TCP 在一条延迟为 50ms 的 1Gbps 链路上传送一个 10MB 的文件，而且 TCP 接收窗口为 1MB。如果 TCP 发送 1KB 的分组（假设无拥塞，无丢失分组）：

(a) 当慢启动打开发送窗口达到 1MB 时用了多少 RTT？

(b) 发送该文件用了多少 RTT？

(c) 如果发送文件的时间由所需的 RTT 的数量与链路延迟的乘积给出，传输的有效吞吐量是多少？链路带宽的利用率是多少？

17. 考虑一个简单的拥塞控制算法：使用线性增加和成倍减少，但不用慢启动，以分组而不是以字节为单位，启动每个连接时拥塞窗口的值为一个分组。给出这一算法的详细描述。假设延迟只有传输时延，而且发送一组分组时，只返回一个 ACK。在分组 9、25、30、38 和 50 丢失的情况下，画出拥塞窗口作为往返时间的函数图。为了简单起见，假设有一个完美的超时机制，它在恰好传送一个 RTT 后检测到一个丢失的分组。

18. 在前一个问题给定的条件下，计算这个连接可获得的有效吞吐量。假设每个分组包含 1 KB 数据，且 RTT = 100ms。

19. 在线性增加期间，TCP 用下面的公式计算拥塞窗口的增量：

$$\text{Increment}=\text{MSS}\times(\text{MSS}/\text{CongestionWindow})$$

解释为什么每个 ACK 到达时计算这个增量可能得不到正确的增量值。对这个增量给出一个更精确的定义。（提示：一个给定的 ACK 可以表示已收到多于或少于一个 MSS 值的数据。）

20. 在什么环境下，TCP 中即使使用快速重传机制仍然可能发生粗粒度超时？

21. 假设在 A 和 B 之间有路由器 R。A 到 R 的带宽无限大（即分组无延迟），但 R 到 B 的链路引入每秒一个分组的带宽延迟（即两个分组用 2s，依此类推）。但是从 B 到 R 的确认可以立即发送。A 通过一个 TCP 连接给 B 发送数据，用慢启动但窗口尺寸任意大。R 除了发送中的分组外，有长度为 1 的一个队列。在每一秒中，发送方先处理到达的 ACK，再响应任何超时。

(a) 假设固定的 TimeOut 为 2s，在 $T=0, 1, \cdots, 6$s 时，发送和收到的分别是哪些分组？由于超时链路会空闲吗？

(b) 如果 TimeOut 是 3s，有哪些变化？

22. 假设 A，R，B 如前一题所述，本题中所不同的是除了发送中的分组外，R 有长度为三个分组的队列。A 用慢启动启动一个连接，接收窗口为无限大。在第二个重复 ACK 时（即同一分组的第三个 ACK 时），进行快速重传，TimeOut 间隔为无限大。忽略快速恢复。当一个分组丢失时，设置窗口大小为 1。用表格表示在前 15s 内 A 收到什么和 A 发送什么，以及 R 发送什么，R 的队列和 R 丢弃

什么。

23. 假设上一题中 R 到 B 的链路从带宽延迟变为传播延迟，这样用 1s 可发送两个分组。列出在前 8s 内发送了什么和接收了什么。假设静态超时值为 2s，在超时时使用慢启动，而且在几乎同一时间发送的 ACK 被合并。注意现在 R 的队列长度是无关的（为什么？）。

24. 假设主机 A 通过路由器 R1 和 R2 到达主机 B：A-R1-R2-B。不用快速重传，且 A 以 2×EstimatedRTT 计算 TimeOut。假设 A 到 R1 及 R2 到 B 的链路有无限大的带宽，但是 R1 到 R2 的链路为数据分组（但不是 ACK）引入每分组 1s 的带宽延迟。描述这样一种情况，其中即使 A 总有数据要发送，R1 到 R2 的链路利用率也没有达到 100%。（提示：假设 A 的 CongestionWindow 从 N 增加到 N+1，其中 N 是 R1 的队列长度。）

25. 你是一个因特网服务提供商，你的客户主机直接连到你的路由器上。你知道一些主机正在使用实验性的 TCP，而且猜想有些主机可能正在使用无拥塞控制的“贪婪的”TCP。你可以在路由器上做哪些测量来确定一个客户根本没有使用慢启动？如果一个客户只在启动时使用慢启动，但不是超时发生后，你能检测到吗？

26. 使 TCP 拥塞控制机制失败通常需要发送方的显式协作。然而，考虑大数据传输的接收方，使用改进为尚未到达的 ACK 分组的 TCP。这么做可能是因为并不需要数据的全部内容，或者是因为丢失的数据能够在后续的一次独立传输中被恢复。这种接收方的行为对会话中的拥塞控制特性有什么影响？你能设计出一种修改 TCP 的方法，以避免发送方以这种方式被利用吗？

27. 考虑图 6-28 中的 TCP 轨迹。分别确定在启动时进行的慢启动的时间区间。超时后的慢启动的时间区间以及线性增加的拥塞避免的时间区间。解释 T=0.5～1.9 期间将发生什么事情。产生这种轨迹的 TCP 版本包含一个图 6-11 的 TCP 中没有的性质，这个性质是什么？这个轨迹和图 6-13 中的轨迹都缺乏一个性质，这个性质是什么呢？

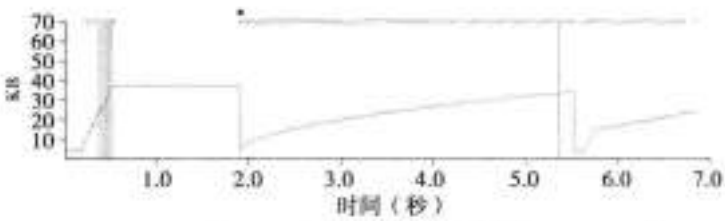


图 6-28 习题 27 的 TCP 轨迹

28. 假设你正在一条 3KBps 的电话链路上下载一个大文件，软件上显示一个每秒平均字节的计数器。TCP 拥塞控制和偶尔的分组丢失如何使计数器产生波动？假设只有总 RTT 的 1/3 花在电话链路上。

29. 假设 TCP 用于一条存在丢失的链路，该链路平均每 4 个数据段中丢失一个。假设延迟带宽积窗口尺寸远大于 4 个数据段。

    (a) 当我们启动一个连接时会发生什么？我们到达过拥塞避免的线性增加阶段吗？

    (b) 不使用来自路由器的显式反馈机制，TCP 有办法区分拥塞丢失和链路丢失吗（至少在短时间内）？

    (c) 假设 TCP 发送方确实可靠地从路由器获得显式的拥塞控制指示。假设上面的链路是普通的，要支持远大于 4 个数据段的窗口尺寸是可行的吗？TCP 必须做些什么呢？

30. 假设两个 TCP 连接共享一条经过路由器 R 的路径。路由器的队列长度为 6 个数据段，每个连接有固定的 3 个数据段的拥塞窗口。这些连接不使用拥塞控制。第三个 TCP 连接现在也试图通过路由器 R，且不用拥塞控制。描述这样一种情景，至少在一小段时间内，第三个连接没有得到任何可用的带宽，前两个连接继续各占用带宽的 50%。如果第三个连接使用慢启动会发生问题吗？就前两个连接部分而言，完全的拥塞避免怎样才有助于解决这个问题？

31. 假设一个 TCP 连接有大小为 8 个报文段的窗口，一个 RTT 为 800ms，发送方以每 100ms 一个报文段

的固定速率发送报文段，接收方以同样的速率无延迟地返回 ACK。一个报文段丢失了，快速重传算法根据收到的第三个重复 ACK，检测到这一丢失。当重传分组的 ACK 最终到达时，在下列情况下，发送方总共损失了多少时间（与无丢失传输相比）？

(a) 发送方在滑动窗口再次向前滑动之前等待重传丢失分组的 ACK。

(b) 发送方用相继到达的每个重复 ACK 作为它可以将窗口向前滑动一个数据段的指示。

32. 本章中叙述了加性增是使拥塞控制机制稳定的必要条件。简述如果所有增加都是指数级的，就可能引起特殊的不稳定性，就是说，TCP 在 CongestionWindow 增长到超过 CongestionThreshold 后，继续使用慢启动。

33. 讨论标记一个分组（如在 DECbit 机制中）和丢弃分组（如在 RED 网关中不执行 ECN）各自的利弊。

34. 考虑一个 RED 网关，其平均队列长度在两个阈值中间且 MaxP = 0.01。

(a) 分别计算 count = 1 和 count = 100 时的丢弃概率 P_{count}。

(b) 计算前 50 个分组无丢弃的概率。注意这个概率等于 $(1-P_1) \times \cdots \times (1-P_{50})$。

35. 考虑一个 RED 网关，其平均队列长度在两个阈值中间且 MaxP = p。

(a) 计算前 n 个分组不被丢弃的概率。

(b) 求 p，使得满足前 n 个分组不被丢弃的概率是 a。

36. 解释在 RED 网关中设置 MaxThreshold = 2 × MinThreshold 的直观原因。

37. 在 RED 网关中，解释为什么事实上 MaxThreshold 比可用缓冲池的实际长度小。

38. 解释容忍突发性与控制网络拥塞之间的基本冲突。

39. 为什么 RED 网关的丢弃概率 P 不是简单地从 MinThresh 时的 P = 0 线性增长到 MaxThresh 时的 P = 1？

40. 在 RFC 3168 中定义的显式拥塞通知（ECN）要用 1 比特指示端点是否支持 ECN。试分析没有这 1 比特的不利影响。

★ 41. 在 TCP Vegas 中，用一个 RTT 间隔内传送的数据量除以 RTT 的长度计算 ActualRate。

(a) 说明对于任何 TCP，如果窗口尺寸保持不变，那么一旦整个窗口的数据被发送，则在一个 RTT 间隔内传送的数据量也是不变的。假设发送方收到一个 ACK 后就立即传送每个数据段，分组没有丢失且按顺序传送，所有数据段的长度相同，且该路径上的第一条链路不是最慢的链路。

(b) 给出时间线图示，表示上面每个 RTT 传送的数据量可能小于 CongestionWindow。

42. 假设一个 TCP Vegas 连接测量第一个分组的 RTT，并将 BaseRTT 的值设置为测量到的 RTT 值，但是后来一条链路发生故障，于是所有后续的通信量被路由选择到了另一条 RTT 为原 RTT 两倍的路径上。TCP Vegas 会对此做何反应？CongestionWindow 的值有什么变化？假设没有实际的超时发生，而且 β 远小于 ExpectedRate 的初始值。

43. 考虑引起 1s 网络延迟的下面两个原因（假设 ACK 立即返回）：

- 一个中间路由器，它的带宽延迟为每个分组用 1s 输出，且无竞争的通信量。
- 一个中间路由器，它的带宽延迟为每个分组用 100ms 输出，且固定地将 10 个分组（来自其他源）补充到队列中。

(a) 通常，传输协议可能怎样区分这两种情况？

(b) 假设 TCP Vegas 在上述连接上发送，CongestionWindow 的初始值为三个分组。在每种情况下 CongestionWindow 会发生什么变化？假设 BaseRTT = 1s 且 β 等于每秒一个分组。

44. 许多实时视频应用因为不能容忍重传时延，因此运行在 UDP 上面不是 TCP 上。然而，这就意味着视频应用不受 TCP 拥塞控制算法的限制。这会对 TCP 流量需求什么影响？该因果关系是明确的吗？好在这些视频应用通常使用 RTP（见 5.4 节），它导致 RTCP 接收方报告被发送到源。这些报告被周期性地发送（例如，每秒钟一次）并且包含在最后报告期间成功接收分组的百分比。描述在 TCP 兼容的方法中源如何使用这些信息来调整其发送速率。

45. 讨论为什么拥塞控制问题在互联网层比在 ATM 层更好管理，至少在互联网仅有一部分是 ATM 时是这样的。在专用的 IP over ATM 网络中，拥塞控制在信元层还是在 TCP 层更好管理？为什么？

46. 考虑图 6-23 中的分类。
    (a) 给出一个非容忍型/速率自适应型（intolerant/rate adaptive）实时应用程序的例子。
    (b) 解释为什么你可能希望一个容忍丢失的应用至少有某种速率自适应。
    (c) 尽管有（b）中的解释，仍要给出一个可看作是容忍型/非自适应型（tolerant/nonadaptive）应用的例子。（提示：容忍即使再小的丢失，也是容忍丢失的应用；你需要将速率自适应解释为适应重大带宽变化的能力。）

47. 一个给定流的传输时间表（见表 6-3）列出每秒中发送分组的数目。该流必须保持在令牌桶过滤器的范围内。对于下列令牌速率，流需要的令牌桶深度是多少？假设桶开始时是满的。
    (a) 每秒 2 个分组。
    (b) 每秒 4 个分组。

表 6-3 习题 47 的传输时间表

时间（s）	分组发送
0	8
1	4
2	1
3	0
4	6
5	1

48. 一个给定流的传输时间表（见表 6-4）列出每秒中发送分组的数目。该流必须保持在令牌桶过滤器的范围内。求作为令牌速率 r 的函数必需的桶深度 D。注意，r 只能取正整数值。假设桶开始时是满的。

49. 假设一台路由器已经接收了 TSpecs 流，如表 6-5 所示，用令牌速率为每秒 r 个分组且桶深度为 B 个分组的令牌桶过滤器来描述它。所有流都在同一个方向，路由器每 0.1s 转发一个分组。
    (a) 一个分组可能面临的最大延迟是多少？
    (b) 假设第三个流一直以最大速率均匀地发送分组，路由器在 2.0s 内能从该流发送分组的最小数目是多少？

表 6-4 习题 48 的传输时间表

时间（s）	分组发送
0	5
1	3
2	1
3	0
4	6
5	1

表 6-5 习题 49 的 TSpecs

r	B
1	10
2	4
4	1

50. 假设一台 RSVP 路由器突然丢失了它的预定状态，但仍在运行。
    (a) 如果路由器用单个 FIFO 队列处理预定的和非预定的流，已预定的流会出现什么情况？
    (b) 如果路由器用加权公平排队隔离预定的和非预定的通信量，已预定的流会出现什么情况？
    (c) 最终，这些流的接收方会请求重新开始它们的预定。给出这些请求被拒绝的一种情况。

端到端数据

在获得数据之前就构建理论是大错特错的。

——阿瑟·柯南道尔爵士

问题：我们用数据做什么？

从网络的观点看，应用程序间彼此发送消息，每条消息只是一个未解释的字节串。然而，从应用程序的观点看，这些消息包含各种类型的数据（data）——整型数组、视频帧、文本行、数字图像，等等。换句话说，这些字节是有含义的。我们现在来考虑这样一个问题，怎样更好地对应用程序要转换成字节串的各种不同类型的数据进行编码。从许多方面来看，这类似于我们在2.2节见到的把字节串编码为电磁信号的问题。

回顾第2章对编码的讨论，实质上涉及两个问题。第一是接收方能从信号中提取出与传送方发送的消息相同的消息，这就是组帧的问题。第二是尽可能地提高编码效率。把应用程序数据编码为网络消息时都存在这两个问题。

为了让接收方提取发送方发来的消息，就出现了发送方和接收方要统一消息格式的问题，通常称为表示格式（presentation format）。例如，发送方要给接收方发送一个整型数组，那么双方必须协商每个整数是什么样的表示格式（比如，长度为多少字节，采用怎样的排列方式，以及最高有效位最先到达还是最后到达），以及在数组中有多少元素。7.1节讲述传统计算机数据的各种编码，如整数、浮点数、字符串、数组和结构体。还有一些为多媒体数据制定的格式，例如视频一般是用运动图像专家组（Moving Picture Experts Group，MPEG）所创建的系种格式传输，而静止图像通常用联合图像专家组（Joint Photographic Experts Group，JPEG）格式传输。在多媒体数据编码中出现的具体问题将在7.2节讨论。

我们需要考虑多媒体类型数据的表示和压缩（compression）两个方面。我们熟知的传输格式、音频存储格式、视频处理格式都有这些问题：确保什么内容被记录、被拍照或者被听到，以便接收方可以正确解读发送方发送的信息，而且这么做在某种程度上可以使网络不被大量的多媒体数据所淹没。

压缩及更普遍的编码效率问题由来已久，可追溯到20世纪40年代香农关于信息论的早期工作。实际上，人们朝着两个相反的方向努力。一方面，我们希望在数据中加入尽可能多的冗余，以便即使消息出现了错误，接收方仍能提取出正确的数据。我们在2.4节的差错检测和纠错码中所看到的给消息添加多余信息的做法正是为了这一目的。另一方面，我们希望尽可能从数据中删掉更多的冗余，以便能用更少的比特进行编码。鉴于我们的感官和大脑处理视觉和听觉信号的方式，多媒体数据为压缩提供了大量的机会。我们听不见高频声音也听不到低频声音，就像我们不能像注意一张大图片一样注意图像中的所有细节，尤其是图像移动时。

我们有足够的理由证明压缩对网络设计者是重要的，原因之一是我们很难保证处处

都有带宽充裕的网络。例如，我们设计压缩算法的方式影响着对丢失和延迟数据的敏感度，从而影响资源分配机制和端到端协议的设计。反之，如果在一个视频会议期间底层网络不能保证一定量的带宽，那么我们就可以选择设计能适应网络条件变化的压缩算法。

最后，表示格式化和数据压缩的一个重要特征就是需要发送主机和接收主机处理消息数据中的每个字节。为此，表示格式化和压缩有时称为数据操纵（data manipulation）功能。与我们至今看到的大多数协议不同，这些协议处理信息时不考虑其包含的内容。由于必须对消息中数据的每个字节进行读、计算和写，所以数据操纵会影响网上端到端的吞吐量。实际上，这些操纵可能就是影响吞吐量的一些因素。

7.1 表示格式化

最常见的网络数据转换就是应用程序使用的表示法和适合网络传输的格式之间的相互转换。通常将这种转换称为表示格式化（presentation formatting）。如图7-1所示，发送程序把要传输的数据由其内部表示形式转换成可以在网上传输的消息，也就是说，数据被编码（encoded）成消息。在接收端，应用程序把接收到的消息再转换成它可以处理的表示形式，也就是说，消息被解码（decoded）。编码有时也称为参数排列（argument marshalling），解码有时也称还原（unmarshalling）。此术语来自RPC领域，客户端认为它调用了一个带着一组参数的过程，这些参数"按适当的、有效的方法排列和组织"[⊖]，从而构成网络消息。



图7-1 包含编码和解码应用程序数据的表示格式化

你也许会问，编码问题为什么如此复杂，以致于要用到像排列这样的名称。原因之一是计算机用不同的方法表示数据。例如，有些计算机用IEEE标准754格式表示浮点数，而有些仍使用它们自己的非标准格式。即使是像整型数那么简单的数据，不同体系结构也要使用不同的长度表示（例如16位、32位、64位）。更糟的是，一些计算机将整型数表示成高端字节序（big-endian）形式（一个字的最高有效位在高地址字节中），而另一些则将整型数表示成低端字节序（little-endian）形式（一个字的最高有效位在低地址字节中）。MIPS和PowerPC处理器是高端字节序体系结构的例子，而Intel x86系列则是低端字节序体系结构的例子。图7-2中，分别给出整数34 677 374的高端字节序表示法和低端字节序表示法。

使参数排列变得困难的另一个原因是应用程序用不同的语言编写，而且，即使你只使用一种语言，也可能有不止一个编译程序。举例来说，编译程序在内存中怎样存放结构（记录），如组成结构的字段之间填充多少位，就有很大的自由度。因而，即使计算机的体系结构和写程序的语言都相同，你也不能简单地把结构从一台计算机传给另一台计算机，因为目标机上的编译程序也许对结构中字段的位置做了不同的调整。

⊖ 这是《韦氏新大学词典》中对排列的定义。

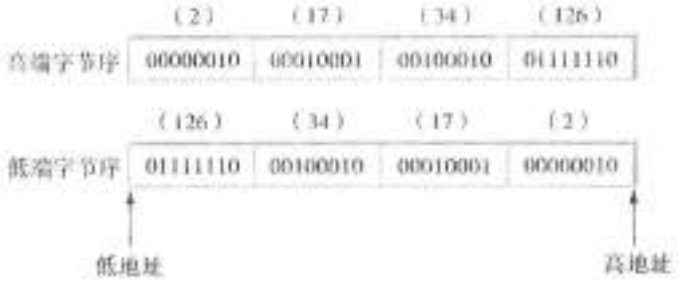


图 7-2 整数 34 677 374 的高端字节序和低端字节序的字节顺序

7.1.1 分类方法

尽管任何做过参数排列的人都会告诉你，参数排列不会将火箭科学这样难的理论包含进去，它只是摆弄比特位的一桩小事，但你需要在许多设计方案中做出选择。我们首先给出参数排列系统的简单分类方法。以下虽不是唯一可行的分类方法，但它足以涵盖大多数重要的可供选择的方案。

1. 数据类型

第一个问题是系统打算支持什么样的数据类型。通常，我们可以将由参数排列机制支持的类型分为三级，每一级都使参数排列系统面对更复杂的任务。

在最低级，参数排列系统对基本类型（base type）的某个集合进行操作。通常，基本类型包括整数、浮点数和字符。系统还可以支持序数类型和布尔型。如上所述，基本类型集合的含义是指，编码进程必须能将每一基本类型从一种表示法转换为另一种表示法，如把整型从高端字节序表示转换为低端字节序表示。

向上一级是扁平类型（flat type）：结构和数组。扁平类型初看起来并不会使参数排列系统复杂化，实则不然。问题是，为了按字的边界对齐字段，编译程序在编译应用程序时习惯于在组成结构的字段之间加入填充。参数排列系统通常将结构压缩（pack）使得它们不含填充。

在最高级，参数排列系统必须处理复杂类型（complex type）：使用指针建立的类型。也就是说，一个程序要发送给另一个程序的数据结构可以不包含在某个单一的结构中，而可能包含从一个结构指向另一结构的指针。树就是包含指针的复杂类型的一个很好的例子。显然，数据编码器必须为网上传输准备好数据结构，因为指针是通过内存寻址实现的，而且，驻留在一台计算机上某个内存地址的结构并不意味着在另一台计算机上有相同的驻留地址。换句话说，参数排列系统必须串行化（serialize）（展开）复杂数据结构。

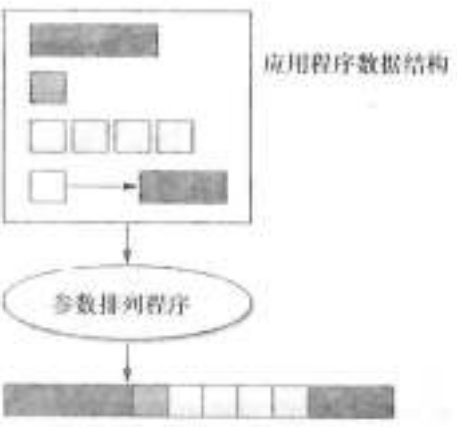


图 7-3 参数排列：转换、压缩和线性化

结论 总之，参数排列与类型系统的复杂化程度有关，它的任务通常包括基本类型转换、结构压缩、复杂数据结构线性化，它们共同构建了一个可以在网上传输的连续消息。这个任务如图 7-3 所示。

2. 转换策略

类型系统一旦建立，接下来要讨论的问题就是参数排列程序采用什么样的转换策略。一般有两个选择：标准中间形式（canonical intermediate form）和接收方调整（receiver-makes-right）。接下来，我们依次讨论每一种转换策略。

标准中间形式的概念就是要确定每一种类型的所有外部表示法。在发送数据前，发送主机将数据由其内部表示转换成这种数据的外部表示。而在接收数据的过程中，接收主机又把这种数据的外部表示转换成其本地的表示。为了说明这种思想，我们来考虑整型数据，其他数据类型按类似的方法处理。你可以把整型数说明为要用于外部表示的高端字节序形式。发送主机必须把它发送的每个整数翻译成高端字节序形式，而接收主机必须把高端字节序形式的整数翻译成它所使用的一种表示方法。（这正是在因特网中对协议首部做的工作。）当然，发送主机也许已经使用了高端字节序形式，在这种情况下就不需要再转换了。

另一个选择有时称为接收方调整，它允许发送方用其内部格式传输数据；发送方不进行基本类型的转换，但通常要压缩和展开较复杂的数据结构。然后接收方负责把数据从发送方的格式翻译成其本地的格式。用这种策略的问题是，每台主机必须准备好转换来自所有其他体系结构的数据。这在联网技术中被认为是一个 $N \times N$ 解决方案（N by-N solution）：N 个体系结构中的每一台计算机都必须能处理所有 N 个体系结构的数据。相反，在一个使用标准中间形式的系统中，每台主机只需要知道它自己的表示法和另一特定的表示法（一种外部表示法）之间怎么转换。

使用共同的外部格式是正确的做法吗？当然这是网络界过去 25 年来的习惯看法。然而，答案并不总是肯定的。事实证明对各种基本类型不存在那么多不同的表示法，或者说 N 不会那么大。另外，更普遍的情况是同一类型的两台计算机彼此通信。在这种情形下，如果把一种体系结构表示的数据翻译成一些另外的外部表示，而在接收方又必须反过来把数据翻译成相同体系结构的表示，似乎是愚蠢的。

第三种选择是（尽管我们知道不存在使用它的系统）：如果发送方知道接收方有同样的体系结构就使用接收方调整法；如果两台计算机采用不同体系结构，就使用某种标准中间形式。这时，发送方如何了解到接收方的体系结构？它可以从一台名字服务器或者首先使用一个简单的测试例子看是否产生相应的结果来获得该信息。

3. 标记

参数排列中的第三个问题是接收方如何知道它接收的消息中包含什么类型的数据。有两种常用的方法：带标记（tagged）数据和不带标记（untagged）数据。带标记的方法十分直观，所以我们首先来介绍它。

标记是指包含在消息中的任何附加信息（除了基本类型具体表示之外的信息），它有助于接收方解码消息。有几种标记可以包含在消息中。例如，每个数据项可以增加一个类型（type）标记。类型标记说明其后跟着的值是整数、浮点数或其他类型。另一个例子就是长度（length）标记。这样的标记用于说明一个数组元素的个数或一个整型数的长度。第三个例子是体系结构（architecture）标记，它可以与接收方调整策略相结合，说明生成消息中所含数据的体系结构。图 7-4 描述了如何用

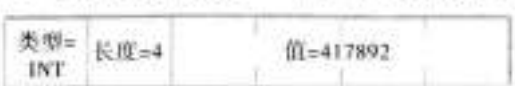


图 7-4 用带标记消息编码的 32 位整数

带标记消息的方法编码一个简单的 32 位整数。

当然，另一个方法就是不用标记。那么，在这种情况下，接收方怎样知道如何解码数据呢？它知道如何解码是因为程序中已经指定。换句话说，如果你调用一个带参数的远程过程，其参数为两个整数和一个浮点数，那么这个远程过程就不必检查标记以了解正在接收的类型数据，它只假设消息包含两个整数和一个浮点数，并做相应的解码。对于大多数情况，这样做是有效的，只在发送可变长数组时例外。在这种情况下，一般用长度标记来说明数组的长度。

不带标记数据的方法也是值得关注的，它意味着表示格式化是真正端到端的。对于一些中间代理而言，如果不标记数据，就不可能解释消息。你也许会问，中间代理为什么需要解释消息？这是因为为解决系统设计不能处理的问题而提出的特定解决方案往往会导致奇怪的事情发生。不良网络设计的问题不在本书讨论的范围。

4. 桩

桩是实现参数排列的一段代码。一般桩是用来支持 RPC 的。在客户端，桩把过程参数排列成可以通过 RPC 协议传输的消息。在服务器端，桩反过来把这个消息转换成一组用来调用远程过程的参数。桩或者被解释或者被编译。

在基于编译的方法中，每个过程都有一个定制的客户桩和服务器桩。虽然桩可由人工设计，但它们通常是由桩编译程序根据过程接口的描述产生的。这个情形如图 7-5 所示。由于桩是编译的，所以通常它的效率很高。在基于解释的方法中，系统提供通用的客户桩和服务器桩，它们都由过程的接口描述来设置参数。因为这种描述很容易修改，所以解释的桩优点是具有灵活性，而编译的桩在实际中更常见。



图 7-5 桩编译程序把接口描述作为输入并输出客户桩和服务器桩

7.1.2 例子（XDR、ASN.1、NDR）

现在，我们按照分类方法简要地描述三种流行的网络数据表示法，并用整数基本类型来说明每个系统是如何工作的。

1. XDR

外部数据表示（External Data Representation，XDR）是用在 SunRPC 上的网络格式。在刚才介绍的分类方法中，XDR 支持以下内容：

- 除函数指针外的整个 C 类型系统。
- 定义一个标准中间形式。
- 不使用标记（除了说明数组的长度外）。
- 使用编译的桩。

一个 XDR 整数是编码一个 C 整数的 32 位数据项。用两个辅助记号表示它。这两个辅助记号是：C 整数的最高有效字节在 XDR 整数中的首字节，C 整数的最低有效字节在 XDR 整数中的第 4 字节。就是说，XDR 对整数使用高端字节序格式。就像 C 代码一样，XDR 既支持有符号整数也支持无符号整数。

XDR 表示可变长数组时，首先给出说明数组中元素个数的无符号整数（4 个字节），再跟随一些相应类型的元素。XDR 按结构分量在结构中的说明顺序来编码。对数组和结构的每一个元素或分量的大小都用 4 个字节的倍数表示。较小的数据类型用全 0 填充到 4 个字节。“填充到 4 个字节”的规则有一个例外，对于字符类型，每个字节编码一个字符。

下面代码段给出一个 C 结构体（item）以及编码/解码此结构的 XDR 例程（xdr_item）的例子。图 7-6 是 XDR 关于这个结构体的实际表示，在这个结构体中字段 name 是 7 个字符长，数组 list 中有 3 个值。

```
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
   int     count;
   char    name[MAXNAME];
   int     list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
    return(xdr_int(xdrs, &ptr->count) &&
       xdr_string(xdrs, &ptr->name, MAXNAME) &&
       xdr_array(xdrs, &ptr->list, &ptr->count, MAXLIST,
              sizeof(int), xdr_int));
}
```

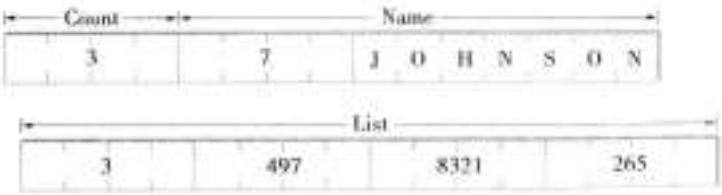


图 7-6 XDR 中一个结构体编码的例子

在这个例子中，xdr_array、xdr_int 和 xdr_string 是 XDR 分别为编码和解码数组、整数和字符串提供的基本函数。参数 xdrs 是一个环境变量，XDR 用它跟踪消息处理过程中

的位置，它包含一个标志来说明这个例程是用于编码消息还是用于解码消息。换句话说，子程序 xdr_item 既可以用在客户端又可以用在服务器端。注意，应用程序员既可以手工编写 xdr_item 例程，也可以使用称为 rpcgen（未显示）的桩编译器产生这个编码/解码例程。在后一种情况中，rpcgen 用定义数据结构 item 的远程过程作为输入，并输出相应的桩。

当然，XDR 到底怎样执行依赖于数据的复杂性。在整型数组的简单情况下，其中每个整数必须按一个字节到另一个字节的顺序转换，每个字节平均需要 3 条指令，意味着整个数组的转换受存储器限制。每个字节进行更复杂的转换时需要更多指令，这会导致其受 CPU 限制且因此而以低于内存带宽的数据率工作。

2. ASN.1

抽象语法标记 1（Abstract Syntax Notation One，ASN.1）是一个 ISO 标准，它定义网上发送数据的一种表示方法。ASN.1 的专用表示方法部分称为基本编码规则（Basic Encoding Rules，BER）。ASN.1 支持无函数指针的 C 类型系统，定义标准中间形式，并使用类型标记。它的桩或者被编译或者被解释。ASN.1 BER 为人们所知的原因之一是它被用于因特网标准的简单网络管理协议（Simple Network Management Protocol，SNMP）。

ASN.1 用三元组的形式表示每个数据项：

(tag，length，value)

虽然 ASN.1 允许定义多字节的标记，但 tag 通常是一个 8 位的字段。length 字段说明 value 要占多少字节。下面详细讨论 length。复合数据类型（比如结构体）可以通过嵌套基本数据类型来构造，如图 7-7 所示。

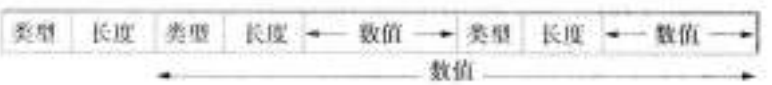


图 7-7 在 ASN.1/BER 中用嵌套建立的复合类型

如果 value 小于或等于 127 字节，那么就用单字节来指定 length。因此，如图 7-8 所示，一个 32 位整数被编码为 1 字节 type、1 字节 length 以及 4 字节编码整数。在一个整数的情况下，正如 XDR 表示法那样，value 本身用 2 的补码表示而且用高端字节序形式。记住即使整数的 value 在 XDR 和 ASN.1 中用完全相同的方法表示，但 XDR 表示中既没有与整数相关的 type 标记也没有 length 标记。这两个标记在消息中都占一定空间，更重要的是，它们都需要在参数排列和还原的过程中处理。这是 ASN.1 不如 XDR 效率高的一个原因。另外，每个数据值的前面有一个 length 字段，这就意味着数据值不可能正好落在自然字节的边界上（例如，一个整数在一个字的边界开始）。这使编码/解码处理更加复杂。

如果 value 大于或等于 128 字节，那么用多个字节来说明其 length。这时你也许要问，为什么一个字节能说明至多 127 个字节而不是 256 个字节的长度？原因是 length 字段中的 1 位用来指示 length 字段有多大。第 8 位为 0 指示 length 字段为 1 字节。为了说明更长的 length，将第 8 位置为 1，其他 7 位说明再加上多少字节组成 length。图 7-9 说明一个单一的 1 字节 length 和一个多字节 length。

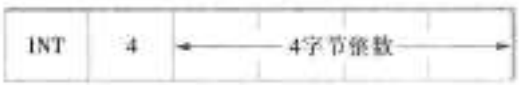


图 7-8 4 字节整数的 ASN.1/BER 表示

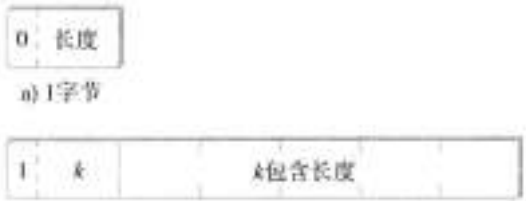


图 7-9 长度的 ASN.1/BER 表示

3. NDR

网络数据表示法（Network Data Representation，NDR）是用于分布式计算环境（Distributed Computing Environment，DCE，我们在 5.3 节中会介绍）的数据编码标准。与 XDR 和 ASN.1 不同，NDR 使用接收方调整方式。NDR 表示法在每条消息前插入一个体系结构标记，而对单个数据项是不带标记的。NDR 使用编译程序生成桩，这个编译程序用接口定义语言（Interface Definition Language，IDL）写出程序描述并生成必要的桩。IDL 看起来和 C 非常相似，所以，基本上支持 C 类型系统。

图 7-10 说明包含在每个 NDR 编码消息中前面 4 字节的体系结构定义标记。第一个字节含两个 4 位的字段。第一个字段 IntegrRep（整数格式）定义消息中所有整数的格式：0 表示高端字节序整数，1 表示低端字节序整数。CharRep（字符格式）字段说明所使用的字符格式：0 表示 ASCII（美国国家信息交换标准代码），1 表示 EBCDIC（一个更早的由 IBM 定义的不同于 ASCII 的编码）。接下来，FloatRep（浮点数格式）字段定义使用哪一种浮点数表示法：0 表示 IEEE 754，1 表示 VAX，2 表示 Cray，3 表示 IBM。最后两个字节保留将来使用。注意像整型数组这样的简单情况，NDR 做的很多工作都与 XDR 相同，所以它也能达到同样的性能。

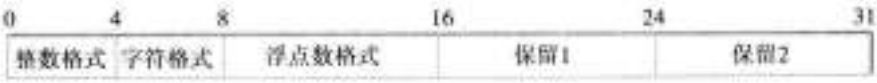


图 7-10 NDR 的体系结构标记

7.1.3 标记语言（XML）

尽管我们从 RPC 的观点讨论了表示格式化的问题，也就是说，如何编码基本数据类型和复合数据结构使它们能从客户程序被发送到服务器程序，但同样的问题在其他环境中也存在。例如，Web 服务器如何描述一个 Web 页面才能使不同的浏览器知道在屏幕上该显示什么？在这种特定的情况下，答案就是超文本标记语言（HyperText Markup Language，HTML），它指示某些字符串应该用粗体或斜体显示，应该使用什么类型的字体和字号，以及图像应该放在什么位置。

由于 Web 非常流行，并且 Web 上存在各种各样的应用和数据，这就导致了不同的 Web 应用之间需要互相通信和理解对方数据的情况。例如，一个电子商务网站需要和一个物流公司的网站沟通，以便允许客户不用离开电子商务网站就可以跟踪一个包裹。实际上这看起来很像 RPC，如今 Web 采用的服务器之间的通信方法是基于可扩展标记语言（Extensible Markup Language，XML）的。

标记语言，如 HTML 和 XML，使用标签数据来达到目的。数据被表示为文本，而且被称为标记（markup）的文本标签与数据文本交织起来表示数据信息。事实上 HTML 只实现了如何将文本显示出来，而其他的标记语言可以标记数据类型和数据结构。

XML 实际上是一个框架，用于为不同的数据定义不同的标记语言。例如，XML 可用来定义一种大致等同于 HTML 的标记语言，叫作可扩展超文本标记语言（Extensible Hyper Text Markup Language，XHTML）。XML 为标记和数据文本相混合定义了基本语法，但特殊标记语言的设计者必须命名并定义其标记。一种普遍的做法是将基于 XML 的语言简单地称作 XML，不过在本书中，我们仍要强调其中的区别。

XML 语法看起来很像 HTML。例如，一段用假设基于 XML 的语言记载的雇员记录如下面的 XML 文档（document）所示，而且可以存储在 employee.xml 文件中。第一行说明所用的 XML 版本，其余几行指定构成雇员记录的 4 个字段，其中最后一个字段（hiredate）包含三个子字段。换句话说，XML 允许用户指定成对的标记/值的嵌套结构，这种结构可以等价于一个表示数据的树状结构（把 employee 作为根节点）。这类似于 XDR、ASN.1 和 NDR 表示复合类型的能力，但是，它使用的是一种既能由程序处理且具有可读性的格式。更重要的是，一些程序（如解析器）可以用来跨越不同的基于 XML 语言，因为这些不同语言的定义都可以表示为数据并提交给这些程序进行分析。

```
<?xml version="1.0"?>
<employee>
  <name>John Doe</name>
  <title>Head Bottle Washer</title>
  <id>12345678</id>
  <hiredate>
    <day>5</day>
    <month>June</month>
    <year>1986</year>
  </hiredate>
</employee>
```

虽然文档中的数据和标记都很容易理解，但还是要由雇员记录语言的定义来决定哪些标记是合法的、这些标记代表什么意思、实现什么数据类型等。没有这些标准定义的标记，程序员（或计算机）就不能确定 year 字段中的 1986 是字符串、整数、非符号整数还是浮点数。

特定的基于 XML 语言的定义由大纲（schema）给出，而模式是一个说明怎样解释数据集合的数据库术语。现在有很多定义 XML 的大纲语言。我们现在把关注点放在具有主导地位的 XML Schema 语言上。一个由 XML Schema 定义的大纲称为 XML 大纲定义（XML Schema Definition，XSD）。下面是一个用于 employee.xml 的 XSD 例子，换句话说，它定义了示例文档应符合的语言。这些代码被存放在一个名为 employee.xsd 的文件中。

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="employee">
    <complexType>
      <sequence>
        <element name="name" type="string"/>
        <element name="title" type="string"/>
        <element name="id" type="string"/>
        <element name="hiredate">
          <complexType>
            <sequence>
              <element name="day" type="integer"/>
              <element name="month" type="string"/>
              <element name="year" type="integer"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

这个 XSD 看起来有点像我们的例子文档 employee.xml，这是因为 XML 大纲本身就是基于 XML 的语言。所以这个 XSD 和上面定义的 employee.xml 文档有着很明显的联系。例如：

```
<element name="title" type="string"/>
```

引号内的标记 title 被解释为一个字符串。通过 XSD 里的行序列和嵌套可以得出 title 字段为一个雇员记录的第二个元素的结论。

不同于其他大纲语言，XML Schema 提供了多种数据类型，如字符串型、整型、小数型和布尔型。就像在 employee.xsd 里展示的一样，允许各种数据类型的组合和嵌套以创建复合数据类型。所以，XSD 定义的不仅仅是语法，还定义自己的抽象数据模型。一个符合 XSD 的文档表示一个符合特定数据模型的数据集合。

XSD 的意义在于定义了一个抽象数据模型，而不仅仅是语法模型，因此采用 XML 之外的表示方式的数据也可符合该模型。然而，XML 作为一种在线表示形式也具有一些缺点：相对于其他数据表示法它并不紧凑，并且解析速度会比较慢。还有一些可选择的采用二进制表示的方法。当 W3C 做出了高效 XML 交换（EXI）草案的时候，国际标准化组织（ISO）已经发表了一个名为“Fast Infoset”的草案。二进制表示法牺牲了可读性，但方便了更大程度的压缩和快速解析。

XML 命名空间

XML 必须处理的一个普遍问题就是名字冲突[⊖]。之所以会出现这个问题是因为模式语言（比如 XML Schema）支持模块化，采用模块化是考虑到一个模式可以由另外一个模式重用。假设两个 XSD 都定义了一个标记并命名为“idNumber”。或许一个 XSD 使用这

⊖ 命名对于计算机科学家来说是一个重要而常见的问题。我们将在 9.3.1 节中考察 DNS（因特网主机的命名机制）时更详细地讨论命名空间。

个名字来识别公司的雇员，而另一个 XSD 使用这个名字来识别公司所拥有的笔记本电脑。我们可能重用这两个 XSD 到第三个 XSD 中用来描述笔记本电脑与雇员的关联。但如果想这样做就得有一些机制来区分雇员号和笔记本电脑编号。

XML 用来解决这个问题的方法是 XML 命名空间（XML namespace）。命名空间是名字的集合。每个 XML 命名空间可以用统一资源标识符（uniform resource identifier，URI）来识别。URI 的具体细节会在 9.1.2 节中介绍。现在，我们需要知道的是这是一种全局统一的标识符。[⊖] 简单的标记名字（比如 idNumber）只要在命名空间中是唯一的就可以加入命名空间。因为命名空间是全局唯一的，而且简单的名字在命名空间中也是唯一的，所以这两个全局唯一的限定的名字（qualified name）不会发生冲突。

XSD 通常使用如下的命令行定义一个目标命名空间（target namespace）：

```
targetNamespace="http://www.example.com/employee"
```

其中，http://www.example.com/employee 是一个 URI，标识一个假设的命名空间（简单地说，我们的 XSD 例子并没有说明一个目标命名空间）。所有 XSD 中定义的新标记都属于这个命名空间。

现在，如果一个 XSD 想引用已经在其他 XSD 中定义的名字，它可以用命名空间来预定义那些名字。从 XSD 转回文档，一个文档需要在使用命名空间时分派一个短命名空间前缀。例如下面一行关联 emp 为 employee 的命名空间前缀：

```
xmlns:emp="http://www.example.com/employee"
```

任何命名空间标记都可以通过 emp 前缀来进行确认，比如下面这行中的 title：

```
<emp:title>Head Bottle Washer</emp:title>
```

emp：title 是个受限的名字，它不会和其他命名空间的名字产生冲突。

值得一提的是，XML 当前应用非常广泛，从 Web 服务中的 RPC 样式通信到办公工具再到即时消息。我们在第 9 章中会看到它的更多应用。它现在无疑是因特网上层所依靠的核心协议之一。

7.2 多媒体数据

根据多方估计，包括音频、视频和静态图像的多媒体数据已经占据了因特网的大部分流量。这是一个相对近期的发展——现在也许很难相信，在 2005 年以前是没有 YouTube 的。使多媒体数据在互联网上广泛传播的部分原因可能是压缩技术的进步。因为多媒体数据大多是靠人类的视觉和听觉感官来接收的，然后经过人类大脑的处理，所以压缩这些数据要面临很多独特的挑战。你试图保留最重要的信息，同时摆脱任何不会改善视觉或听觉感官体验的信息。因此，计算机科学和人类感知的学习都会发挥作用。在本节中，我们来看在表述和压缩多媒体数据方面人们所做出的主要努力。

压缩技术的使用当然不仅仅局限于多媒体技术上——例如，在互联网上发送文件或者下载后解压缩文件时，你很可能会用到一个实用的压缩工具，像 zip 或者 compress。事实证明用于压缩数据的技术都是无损的（lossless），因为大多数人不喜欢文件里的数据丢

⊖ HTTP URL 是 URI 的特殊形式。

失，这些技术也可以作为多媒体数据压缩方案的一部分。与此相反，有损压缩（lossy compression）技术通常应用在多媒体数据上且并不能保证接收的数据与发送的数据一模一样。如上面所提到的，这是因为多媒体数据中通常只包含一小部分对接收方有用的数据。我们的感官和大脑只能察觉到这么多细节，我们很擅长在看或者听的时候自动填补漏掉的内容，甚至能自动纠正其中的错误。有损算法通常会得到比其他算法更好的压缩比，其数值比无损压缩好一个数量级或更多。

为了了解压缩技术对于网络多媒体的传播有多么重要，考虑下面的例子。一个1 080像素×1 920像素的高清电视屏幕，每个像素有24位的颜色信息，所以每帧是

$$1\,080\times1\,920\times24=50\text{Mb}$$

所以如果你想每秒发送24帧，那就会超过1Gbps。这比大多数因特网用户上网的速度还要快很多。相比之下，现代压缩技术可以将速度降到10Mbps左右，得到足够清晰的HDTV信号，减少两个数量级并满足很多宽带用户的要求。类似的压缩技术可以应用于像YouTube视频剪辑这样的低质量视频，如果没有压缩技术使所有有趣的视频适应今天的网络带宽，它就不会拥有现在这样的人气。

近年来，压缩技术应用到多媒体上已经形成了一个大的创新领域，尤其是有损压缩。然而无损压缩技术也发挥了很重要的作用。事实上，大多数有损压缩技术包含的一些步骤是无损的，所以我们从对无损压缩的概述开始讨论。

7.2.1 无损压缩技术

在许多方面，压缩都是不能与数据编码分开的。就是说，在考虑如何用一组比特去编码一块数据时，我们同时要考虑如何实现用最少的比特编码数据。例如，如果有一个由A到Z的26个字母组成的数据块，假设所有符号在你要编码的数据块中出现的机会相等，那么，最好用5个比特编码每个符号（因为$2^5=32$是大于26的2的最小次幂）。然而，如果符号R出现的次数为50%，那么使用比编码其他符号更少的比特去编码符号R就是一个很好的主意。通常，如果你知道每个符号出现在数据中的相对概率，那么，你就可以用某种方法为每个可能的符号指定不同数量的比特，即用最少的比特数去编码给定的数据块。这就是*赫夫曼编码*（Huffman code）的基本思想，也是早期在数据压缩方面最重要的发展之一。

相关主题

什么时候进行压缩？

因为网络传递压缩数据比传递未压缩数据用的时间少，所以似乎在发送前压缩数据总是个好主意。然而，并不一定如此，压缩/解压缩算法常常是需要耗时的计算。你必须考虑给定主机处理器的速度和网络带宽等因素，花时间压缩/解压缩数据是不是值得。特别是，如果B_c是数据通过（顺序地）压缩程序/解压缩程序的平均带宽，B_n是未压缩数据的网络带宽（包括网络处理耗费的带宽），r是平均压缩率，而且，如果我们假设在任何传输前，所有数据都是压缩的，那么发送x个字节未压缩数据所用时间是

$$x/B_n$$

而压缩和发送压缩数据的时间是

$$x/B_c+x/(rB_n)$$

这样，如果

$$x/B_c + x/(rB_n) < x/B_n$$

压缩是有益的，它等价于

$$B_c > r/(r-1) \times B_n$$

例如，压缩率为2，B_c必须大于$2\times B_n$压缩才有意义。

对于许多压缩算法，开始传输前不要求压缩整个数据集合（如果要求这样做的话，视频会议就是不可能的），但首先必须收集一定量的数据（也许是几个视频帧）。在这种情形下，“填充管道”所需的数据总量就是上述方程中x的值。

当然，讨论有损压缩算法时，处理资源并非唯一的因素。对于不同的应用程序，用户非常愿意在带宽（或延迟）和压缩导致的信息丢失程度之间进行不同的权衡。例如，放射线学者看乳房X光片时不可能容忍图像质量有任何重要的损失，但他却可能会容忍在网上等几小时检索图像。相反，多数人显然都能容忍电话交换机中不可靠的音频质量以便能在全球范围内自由地拨打电话（边开车边打电话的质量就更不用说了）。

1. 行程编码

行程编码（Run Length Encoding，RLE）是一种极其简单的压缩技术。其思想是对连续出现的一个符号，只用此符号的一个副本加上符号出现的次数来代替，所以起名为行程。例如，AAABBCDDDD串就被编码为3A2B1C4D。

事实证明，RLE对于压缩一些类型的图像很有用。它可以通过比较相邻的像素值，然后只编码有变化的符号来压缩数字图像。对于有很大相似区域的图像来说，这个技术是相当有效的。例如，对扫描的文本图像，它的效果很显著，能达到压缩率为8∶1的量级。RLE对这样的文件处理很有效是因为这种文件常常包含大量可以被删除的空白。实际上，RLE过去是用来发送传真的关键压缩算法。然而，对局部变化程度很小的图像，其压缩实际上会明显增加图像的字节数，因为，当一个符号不重复出现时，要用两个字节来表示单个符号。

2. 差分脉码调制

另一个简单的无损压缩算法是差分脉码调制（Differential Pulse Code Modulation，DPCM）。它的思想是首先输出一个参考符号，然后输出数据中的每个符号与参考符号的差。例如，使用符号A作为参考符号，字符串AAABBCDDDD将编码为A0001123333，因为A和参考符号相同，B与参考符号的差为1，依次类推。注意这个简单的例子并不能说明DPCM的真正好处，即当差较小时，它们能用比符号本身更少的比特去编码。在这个例子中，差的范围为0~3，每个符号用两个比特表示，而完全表示这些字符则需要用7或8个比特。一旦差变的很大，就选择一个新的参考符号。

对大多数数字图像来说，DPCM比RLE效果更好，因为它利用相邻像素值通常是相似的这个事实。由于这种关系，相邻像素值之间差分的动态范围可能远比原始图像的动态范围小，而且这个范围因此能用更少的比特来表示。使用DPCM，我们已经测量到对数字图像压缩率为1.5∶1。DPCM也适用于音频，因为相邻的音频波形在值上可能是相近的。

另一种略有差别的方法称为delta编码（delta encoding），简单地把一个符号编码为与前一符号的差。比如，AAABBCDDDD可以表示为A001011000。注意，delta编码很适合

编码相邻像素相似的图像。delta 编码后还可以再进行 RLE（行程编码），因为如果每个符号后都有许多类似的符号，我们就会找到一长串 0。

3. 基于字典的方法

我们讨论的最后一个无损压缩法是基于字典的方法，其中最著名的是 Lempel-Ziv (LZ) 压缩算法。Unix 的 compress 和 gzip 命令使用 LZ 算法的一个变种。

基于字典的压缩算法，其思想是为你希望在数据中查找的可变长字符串（把它们看作常用短语）建一个字典（表）。当这些串出现在数据中时，用相应的字典索引替代每个串。例如，在文本数据中不是处理单独的字符，你可以把每个词作为一个串来对待并为每个词输出其在该字典的索引。下面举个例子详细说明，"compression" 一词在特定的词典中的索引为 4978，因为在/usr/share/dict/words 文件中它是第 4978 个词。要压缩一个文本的正文，每次当这个串出现时，就会用 4978 来代替。由于在这个特定的字典中只有 25 000 多个词，需用 15 个比特来编码这个索引，意味着串 "compression" 可以用 15 个比特而不是 7 比特 ASCII 所要求的 77 比特来表示。这样压缩率为 5∶1！在另一个数据点上，当我们把压缩命令应用到本书所描述的那些协议的源代码时，我们可能会得到 2∶1 的压缩率。

当然，剩下的问题是字典从哪里来。一个选择是定义静态字典，最好是为要压缩的数据定制字典。LZ 压缩法使用的更一般的解决方案，即定义基于压缩数据内容的自适应字典。然而，在这种情况下，构造的字典在压缩期间必须和数据一起发送，以便算法的解压部分能完成它的工作。如何正确地建立自适应字典已经成为一个广泛的研究课题。在这一章的结尾我们讨论关于这个主题的重要文章。

7.2.2 图像表示和压缩（GIF、JPEG）

在过去的几年中数字图像的使用增加了。这种增加是由于图形显示器的发明而非高速网络，对用于数字图像数据的标准表示格式和压缩算法的需求也越来越紧迫了。为了满足这个需求，ISO 定义了一个称为 JPEG 的数字图像格式，其名称来自设计它的联合图像专家组（Joint Photographic Experts Group，JPEG）。JPEG 中的 "Joint" 代表它是 ISO/ITU 联合的成果。如今 JPEG 格式是静态图片中应用最广泛的格式。这种格式定义的核心是一个压缩算法，我们下面会描述它。很多在 JPEG 中应用的技术也在 MPEG 中应用，即由运动图像专家组（MPEG）创建的一组视频压缩和传送标准。

在深入研究 JPEG 格式的细节之前，我们注意到，从数字图像到可以被接收者接受的可传送、可解压缩、可正确显示的压缩图像之间有很多的步骤。你可能知道数字图像是由像素组成的（因此，在数码相机的广告中引入兆像素），每个像素代表二维网格中的一个位置，以此来组成图像。对于彩色图像，每个像素有一些代表一个颜色的数值。有很多方式来表示颜色，称为颜色空间（color space），大多数人都熟悉的一个颜色空间是 RGB（红，绿，蓝）。你可以将颜色看成一个三维量，通过红、绿、蓝的不同数值而合成出任何颜色。在一个三维空间中，有很多不同的有效的方式来表述一个给定的点（例如笛卡尔坐标和极坐标）。同样，很多方法用三个数量来描述一个颜色，最常替代 RGB 的是 YUV。Y 是亮度，即像素的整体亮度，U 和 V 包含色度或者颜色信息。容易混淆的是，也有很多 YUV 颜色空间的不同变体。下面会更详细地讲述这个方面。

这个讨论的意义是彩色图像（静态或者动态）的编码和传送需要颜色空间两端之间达

成协议。否则，你最后肯定会得到与发送方不同的颜色。因此，在颜色空间定义上达成一致（也许只是使用的特定命名空间的一种沟通的方式）是任何图像格式或者视频格式定义的一部分。

让我们看看 GIF（Graphical Interchange Format）格式的例子。GIF 使用 RGB 颜色空间，用 8 位代表颜色三维变量中的一个变量，一共需要 24 位。GIF 首先将 24 位彩色图片降到 8 位彩色图片，而不是每个像素 24 位。这是通过识别一幅图片中已用到的颜色来实现的，颜色总量通常都会大大小于 2^{24} 这个值，然后再从 256 种颜色中挑选最接近图片中用到的颜色。也许会多于 256 种颜色，然而，诀窍就是在 256 种颜色中挑选的时候不要让颜色失真过多，这样颜色中的像素就不会改变太大。

256 种颜色存储在一个表里，这个表可以用一个 8 位的数索引，这样每个像素的值都对应一个适当的索引。注意这是一个针对任何多于 256 种颜色的图片的有损压缩的例子。然后，GIF 在结果上运行一个 LZ 变量，将常见的像素作为组成字典的字符串进行处理，这是个无损的操作。通过这种方法，GIF 有时可以达到 10∶1 的压缩比，不过只在图像包含相对比较少的离散颜色的情况下。例如一个图标，GIF 就可以处理得很好。而自然场景的图像通常包含连续光谱上的颜色，用 GIF 就不能压缩到这个压缩比了。在某些情况下，对于 GIF 引起的颜色损失从而导致的扭曲，人眼也不太能察觉出来。

JPEG 格式更适合摄影图像，因此我们以创建它的组织为之命名。JPEG 不会像 GIF 一样减少颜色的量。相反，JPEG 首先将 RGB 颜色（就是通常从数码相机中得到的颜色）转换到 YUV 颜色空间，原因与眼睛感知图像的方式有关。眼睛里有亮度的接收器，并且对每种颜色有单独的接收器。因为我们非常善于感受亮度的变化，所以使用更多数据位传输亮度信息是很有意义的。既然 YUV 中的 Y 表示像素中的整体亮度信息，因此我们就可以单独压缩这个值，而不必过分关注其他两个（色度）值。

如上所述，YUV 和 RGB 是用于描述三维空间中一个点的两种可互换的方式，而且使用一个线性方程从一个颜色空间转换到另一个颜色空间也是可能的。YUV 颜色空间通常用来表示数字图像，方程是：

$$Y=0.299R+0.587G+0.114B$$
$$U=(B-Y)\times 0.565$$
$$V=(R-Y)\times 0.713$$

这里常量的准确值不重要，只要编码器和解码器在此取得一致即可。（为了显示图像，解码器必须要应用逆转换来恢复 RGB 分量。）然而，常量就是根据人类感知的颜色仔细挑选出来的。你可以看到 Y（亮度）是红、绿、蓝各部分的总和，而 U 和 V 分量表示颜色的差异。U 表示平均亮度和蓝色亮度的差异，V 表示平均亮度和红色亮度的差异。你可能注意到，将 R、G、B 设置成它们的最大值（8 位表示 255），会使得 Y 的值等于 255，而此时 U 和 V 都是 0。也就是在 RGB 颜色空间中全白像素是（255，255，255），在 YUV 颜色空间中是（255，0，0）。

一旦图像转换成 YUV 颜色空间，我们就可以将颜色的三个部分单独压缩。我们想将 U 和 V 部分压缩得更小，因为人眼对它们比较不敏感。压缩 U 和 V 的一个方法就是对它们进行下采样（subsample）。下采样的基本思想就是取出一组相邻的像素值，计算这组像素的 U 或 V 平均值，然后用这个值代替这组里的所有像素值进行传输。图 7-11 说明了这一点。亮度（Y）没有下采样，所以所有像素的 Y 值都会被传输，显示在 16×16 像素网

格的左面。而在U和V的情况下，我们将四个相邻的像素设成一组，计算这一组的平均值，然后进行传输。因此，我们最终是传输一个8×8的U和V的像素网格。因此，在这个例子中，对于每四个像素，我们传输6个值（4个Y值，1个U值，1个V值）而不是最开始的12个值（每个部分各4个值），这样可以减少50%的信息。

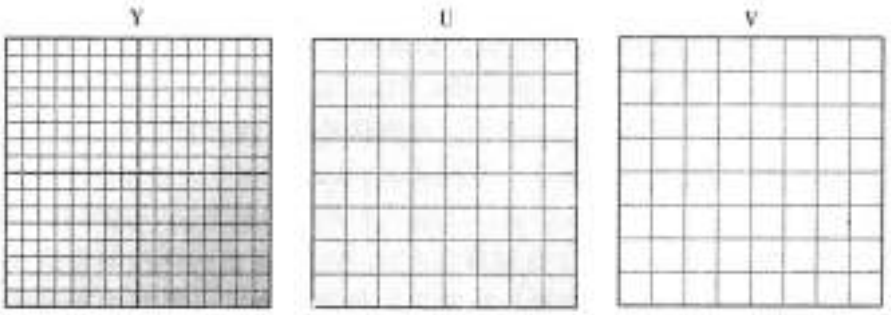


图7-11 对图像中的U值和V值进行下采样

值得注意的是，下采样的程度可以适当加深，相应地会增加压缩比和降低图像质量。在这里所示的下采样方法中，色度是在水平和垂直两个方向进行两倍采样（记为4∶2∶0），正好可以匹配常用于JPEG和MPEG的方法。

一旦下采样完成，我们就有三个网格的像素要处理，每一个都可以单独处理。如图7-12所示，JPEG压缩分三个阶段完成。在压缩端，以每次一个8×8数据块让图像经过这三个阶段。第一阶段对这个数据块进行离散余弦变换（Discrete Cosine Transform，DCT）。如果你把图像看作空间域中的一个信号，那么DCT把这个信号变换成空间频率（spatial frequency）域中一个等价的信号。这是一个无损运算，但必须在进行下一步有损变换前完成。在DCT之后，第二阶段将产生的信号进行量化，并且，在量化过程中丢失信号所包含的最低有效信息。第三阶段编码出最终的结果，但在编码过程中，为前两个阶段完成的有损压缩增添一个无损压缩的成分。解压同样包含这三个阶段，但顺序相反。

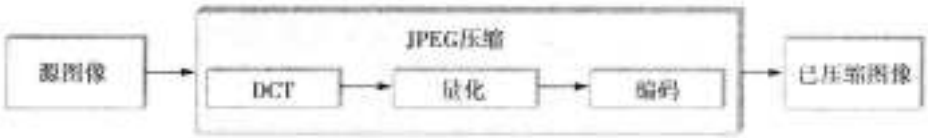


图7-12 JPEG压缩框图

1. DCT阶段

DCT是一种与快速傅里叶变换（Fast Fourier Transform，FFT）紧密相关的转换。它取8×8像素值矩阵为输入，并输出一个8×8频率系数矩阵。你可以把输入矩阵看作定义在二维空间（x和y）的一个64点信号，DCT把这个信号分割成64个空间频率。为了获得空间频率的直观感觉，想象你自己沿x方向横着移过一张图片的过程。你会看见每个像素值按x的某个函数变化。如果随着x的增加这个值慢慢变化，它就有较低的空间频率，然而，如果它的值迅速地变化，它就有较高的空间频率。所以低频对应图片总体特性，而高频对应图片更细微的变化。DCT思想就是要分离那些观看图像所必需的总体特性与不太必要的且有些情况下人眼几乎感觉不到的细微特性。

解压缩期间完成 DCT 的逆过程，DCT 由以下公式来定义：

$$\mathrm{DCT}(i,j)=\frac{1}{\sqrt{2N}}C(i)C(j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathrm{pi.xel}(x,y)\times\cos\left[\frac{(2x+1)i\pi}{2N}\right]\cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$\mathrm{pixel}(x,y)=\frac{1}{\sqrt{2N}}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}C(i)C(j)\mathrm{DCT}(i,j)\times\cos\left[\frac{(2x+1)i\pi}{2N}\right]\cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

$$C(x)=\begin{cases}\dfrac{1}{\sqrt{2}} & x=0\\ 1 & x>0\end{cases}$$

其中 pixel(x,y) 是被压缩的 8×8 块中位置为 (x, y) 的像素的灰度值，此时 $N=8$。

第一个频率系数在输出矩阵的 (0, 0) 位置，称为 DC 系数 (DC coefficient)。我们可以很直观地看出，DC 系数是 64 个输入像素的平均值。输出矩阵的其他 63 个元素称为 AC 系数 (AC coefficient)。它们为这个平均值加上较高的空间频率信息。这样，当从第一个频率系数向前走到第 64 个频率系数时，你就从低频信息移到高频信息，图像从粗糊变得越来越细腻。这些高频系数对感觉到的图像质量越来越不重要。JPEG 的第二个阶段是判断去掉哪一部分系数。

2. 量化阶段

在 JPEG 的第二个阶段，压缩是有损的。DCT 本身不丢失信息，它只是把图像变换成更容易知道什么信息可以删除的形式。(尽管本身是无损的，但由于使用定点运算当然会有一些精度的损失。) 量化是容易理解的：它只是丢弃频率系数中可忽略的那些比特。

为了清楚量化阶段是怎样完成的，可以想象你要压缩小于 100 的某些数字，例如，45、98、23、66 和 7。如果你判定将这些数截断成最接近的 10 的倍数就足以达到目的，那么你可以使用整型算术用量程 10 除每个数，得出 4、9、2、6 和 0。每一个这样的数可以用 4 个比特编码而不像原始数据需要 7 个比特编码。

就像下面公式中给出的那样，JPEG 使用一个量化表给出用于每个系数的量程，而不是对所有 64 个系数使用同样的量程。你可以把这个表 (Quantum) 看成一个参数，对它进行设置可以控制丢失的信息量，相应地控制压缩比。实际上，JPEG 标准声明了在压缩数字图像中已证明是有效的一个量化表集合，表 7-1 是一个量化表的例子。在这样的表中，较低的系数有接近 1 的量程 (意味着几乎没有低频信息丢失)，而高系数有较大的值 (意味着更多高频信息丢失)。值得注意的是，由于这样的量化表，许多高频系数量化后最终被置为 0，这为它们进入第三阶段进一步压缩做好准备。

表 7-1 JPEG 量化表示例

$$\mathrm{Quantum}=\begin{bmatrix}3&5&7&9&11&13&15&17\\5&7&9&11&13&15&17&19\\7&9&11&13&15&17&19&21\\9&11&13&15&17&19&21&23\\11&13&15&17&19&21&23&25\\13&15&17&19&21&23&25&27\\15&17&19&21&23&25&27&29\\17&19&21&23&25&27&29&31\end{bmatrix}$$

基本量化方程为

$$\mathrm{QuantizedValue}(i,j)=\mathrm{IntegerRound}(\mathrm{DCT}(i,j)/\mathrm{Quantum}(i,j))$$

其中

$$\mathrm{IntegerRound}(x)=\begin{cases}[x+0.5] & x\geqslant 0\\ [x-0.5] & x<0\end{cases}$$

解压缩则简单地定义为

$$DCT(i,j)=QuantizedValue(i,j)\times Quantum(i,j)$$

例如，假设某个特定块的 DC 系数（即 DCT（0，0））等于 25，那么，用表 7-1 量化这个值的结果为

$$\lfloor 25/3+0.5 \rfloor=8$$

解压缩过程中，这个系数被恢复为

$$8\times 3=24$$

3. 编码阶段

在 JPEG 最后的阶段，用一种压缩格式编码量化的频率系数。这就导致了另一次压缩，但这次压缩是无损的。DC 系数从（0，0）位置开始，如图 7-13 所示，各个系数按 Z 字形进行处理。沿着这个 Z 字形使用行程编码格式，RLE 只应用于 0 系数，它的效果显著，因为后面的许多系数都是 0。单个系数值则用赫夫曼码来编码。（JPEG 标准允许使用算术编码代替赫夫曼码。）

另外，因为 DC 系数包含来自源图像 8×8 块的很大比例的信息，而且图像从一块到另一块变化较缓慢，因此每个 DC 系数编码成与前一个 DC 系数的差。这是 7.2.1 节介绍的 delta 编码方法。

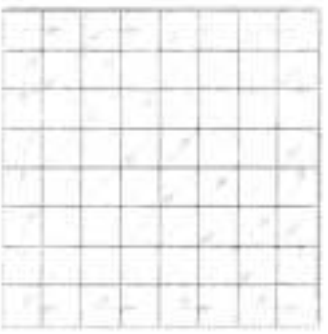


图 7-13 量化频率系数的 Z 字形遍历

JPEG 包含许多变数，用来控制压缩率和逼真度。比如，通过使用不同的量化表可以完成这些工作。这些变数，再加上不同图像有不同特性的因素，因此 JPEG 不可能达到任何精度的压缩率。30∶1 的压缩比是比较常见的，更高的压缩比当然也有可能，但是在更高的压缩比中构件的失真（artifact）（由于压缩而明显失真）会更严重。

7.2.3 视频压缩（MPEG）

现在我们把注意力转到 MPEG 格式上，其名称来自定义它的运动图像专家组（Moving Picture Experts Group，MPEG）。粗略地说，运动图像（即视频）是简单地以某个速率连续显示的静止图像（也称为帧（frame）或图片（picture））。每个帧可以使用与 JPEG 中用到的基于 DCT 的相同技术来压缩。当然这个问题不能到此为止，因为它不能删除视频序列中帧到帧的冗余。例如，在一个场景中，如果没有许多动作，两个连续的视频帧会包含几乎相同的信息，所以没必要发送两次相同的信息。即使有动作的时候，由于运动对象从一帧到下一帧可能没有变化，也会有大量的冗余信息；有些时候，只是位置发生了变化。MPEG 考虑到了帧之间的这种冗余。同时，MPEG 还为伴随视频的音频信号定义一个编码机制，但在这一节我们只考虑 MPEG 的视频方面。

1. 帧类型

MPEG 接收一个视频帧序列作为输入，然后将其压缩成三种类型的帧，分别称为 I 帧（内部图像）、P 帧（预测图像）和 B 帧（双向预测图像）。每个输入的帧被压缩成这三种类型之一。I 帧可以作为参考帧，它们是独立的，既不依赖前面的帧也不依赖后面的帧。粗略地说，I 帧是视频源对应帧的 JPEG 压缩形式。P 帧和 B 帧不是独立的，它们定义了

相对某个参考帧的差。更明确地说，P帧说明与前一个I帧的差，而B帧给出前一个I帧或P帧与后一个I帧或P帧之间的插值。

图7-14说明一个有7个视频帧的序列。由MPEG压缩后，产生一个由I帧、P帧和B帧组成的序列。两个I帧是独立的，可以在接收方解压缩，不依赖任何其他帧。P帧依赖前一个I帧，只有在前一个I帧到达后，它才能在接收方被解压缩。每个B帧既依赖前面的I帧或P帧又依赖后面的I帧或P帧。MPEG在解压缩B帧重现原始视频帧之前，这些参考帧必须到达接收方。

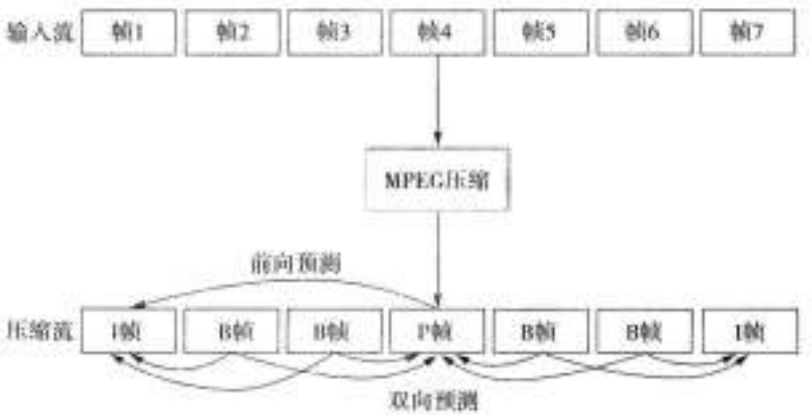


图7-14 MPEG产生的I帧、P帧和B帧的序列

值得注意是，因为每个B帧依赖序列中的后面一帧，因而压缩帧不按顺序传输。图7-13中所示的IBBPBBI顺序就变成IPBBIBB的传输顺序。此外，MPEG不定义I帧与P帧和B帧的比率，这个比率可能依压缩要求及图像的质量而变。例如，若只允许传I帧，这就类似于使用JPEG压缩视频图像。

与前面JPEG的讨论不同，下面我们要集中讨论MPEG流的解码（decoding）。它的描述稍微容易一些，也是当今网络系统中最常执行的操作，这是由于MPEG的编码很费时，以至于它通常是脱机完成的（即不是实时的）。例如，在视频点播系统中，视频提前被编码并储存在磁盘上。当点播者想要观看视频时，MPEG流就被传送到点播者的计算机上，在这台计算机上实时解码并显示。

让我们更仔细地看看三种类型的帧。如上所述，I帧近似等于源帧的JPEG压缩形式。其主要区别是MPEG以16×16宏块（macroblock）为单位工作。对一个用YUV表示的彩色视频，每个宏块的U和V分量下采样变成8×8的块。也就是说，宏块中每个2×2的子块由一个U值和一个V值给出，即4个像素值的平均值。该子块仍然有4个Y值。能够这样做是因为人对颜色不如对亮度敏感，所以不太精确地传输U和V分量对人们观看图像不会有明显的干扰。帧和宏块之间的对应关系在图7-15中给出。

P帧和B帧也是以宏块为单位处理的。直觉上，我们可以看到每一宏块携带的信息捕捉了视频图像的动作，也就是说，它显示宏块相对参考帧在什么方向，移动了多远。下面描述解压缩期间如何用B帧重构一个帧，也可以用类似的方法处理P帧，只不过P帧仅依赖一个参考帧而不是两个参考帧。

在详述B帧如何解压缩之前，我们首先指出，在B帧中的每一宏块没有必要像上面提出的那样相对前一个和后一个帧都做定义，可以只相对前一帧或后一帧来说明。事实上，

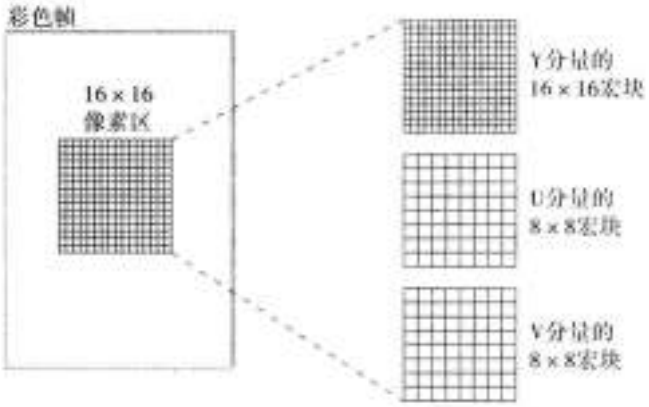


图 7-15 每个帧作为宏块的集合

B帧中某一特定的宏块可以使用和I帧中同样的内部代码。存在这种灵活性是因为如果运动图像变化太快，有时给出内部图像编码比推算向前或向后编码更合适。这样，在B帧中的每一宏块包含一个类型字段，说明宏块使用哪一种编码。在以下的讨论中，我们只考虑在宏块中使用双向预测编码的一般情况。

在这种情况下，B帧中的每个宏块用一个4元组表示：宏块在帧中的坐标，相对前一个参考帧的运动向量，相对后一个参考帧的运动向量。宏块中每个像素的增量 δ（即每个像素相对于两个参考像素变化了多少）。对宏块中的每个像素，第一个任务是在过去的和未来的参考帧中查找对应参考像素。这是使用两个与该宏块有关的运动向量来完成的。然后，把该像素的增量 δ 与两个像素的平均值相加。更准确地说，如果我们分别用 F_p 和 F_f 表示过去的和未来的参考帧，并且由 (x_p, y_p) 和 (x_f, y_f) 给出过去和未来的运动向量，那么当前帧（表示为 F_c）中在坐标 (x, y) 处的像素由如下公式计算：

$$F_c(x,y) = (F_p(x+x_p, y+y_p) + F_f(x+x_f, y+y_f))/2 + \delta(x,y)$$

其中 δ 是B帧中说明的那个像素的增量。这些增量的编码方法与编码I帧像素的方法相同。就是说，它们通过DCT，然后量化。由于增量通常很小，所以大多数DCT系数量化后为0，因此它们可能被有效地压缩。

从前面的讨论中我们已经非常清楚如何进行编码，但有一个例外。压缩期间生成一个B帧或P帧时，MPEG必须决定把宏块放在什么位置。回想一下P帧中的每个宏块，比如，它相对于I帧中的一个宏块来定义，但P帧中的那个宏块不必和I帧中对应的宏块在帧的同一部位，其位置的差由运动矢量给出。你可能想选取一个运动矢量，使P帧中的宏块尽可能类似于I帧中对应的宏块，使该宏块的增量尽可能小。这就意味着，你必须计算出将该图片中的对象从一帧移到了下一帧的什么位置。这是一个运动估计（motion estimation）问题，而且已有解决这个问题的多种技术（启发式方法）。（在本章末尾“扩展阅读”中我们将讨论研究这个问题的文章。）这个问题的难点正是在同样的硬件上，MPEG编码比解码所花费的时间长的原因之一。如上所述，MPEG并不指定任何特定的技术，它只定义将信息编码到B帧和P帧中的格式以及解压缩期间重构像素的算法。

2. 效率和性能

尽管高达150∶1的MPEG压缩比并非前所未闻，但一般情况下MPEG的压缩比为

90：1。就帧的类型而言，对 I 帧来说压缩比大约可达到 30：1（这与先将 24 位颜色减到 8 位颜色时用 JPEG 完成的压缩比是一致的），而 P 帧和 B 帧通常的压缩比要比 I 帧高 3～5 倍。如果不先将 24 位颜色减到 8 位颜色，用 MPEG 可达到的压缩比一般在 30：1～50：1 之间。

MPEG 涉及非常耗时的计算。在压缩端通常脱机完成，在为视频点播服务准备影片方面，这并不是一个问题。如今可以使用硬件实时压缩视频，但是软件实现正在很快弥合这种差距。在解压缩的一端，可以利用廉价 MPEG 视频解压缩卡，但它所做的与 YUV 色彩查找没什么差别，这恰好解决了开销最大的一步。多数现用的 MPEG 解码是由软件完成的。在最近几年，当纯粹用软件解码 MPEG 流时，处理器已经快到足够保持每秒 30 帧的视频速率——现代处理器甚至可以解码高分辨率的 MPEG 视频流（HDTV）。

3. 其他视频编码标准

我们在结束讨论时指出 MPEG 不是唯一可用于编码视频的标准。例如，为编码实时多媒体数据，ITU-T 也定义了“H 系列”标准。H 系列通常包括视频标准、音频标准、控制标准和多路复用技术标准（例如，将音频、视频以及数据混合到单个比特流上）。在这个系列中，H.261 和 H.263 是第一代和第二代的视频编码标准。与以 1.5Mbps 比特率为目标的 MPEG 的早期版本不同，H.261 和 H.263 用于更低的速率。它们用于综合服务数字网（Integrated Services Digital Network，ISDN）标准，支持以 64kbps 为增量的可用带宽链路上的视频。大体上看，H.261 和 H.263 都与 MPEG 有许多类似的地方：它们都使用 DCT、量化和中间帧压缩。H.261/H.263 与 MPEG 只是在细节上有一些差别。事实上，比较新的 H.264 标准也是 MPEG—4 标准的一部分。随着视频被越来越多的设备所支持，从连接低带宽蜂窝无线网的小屏幕设备到连接光缆高带宽的大型电视机，在这个领域可能会有更多的创新要求和更多的标准。

7.2.4 在网上传输 MPEG

正像本章前面提到的那样，MPEG 不只是定义怎样压缩视频，还定义 MPEG 压缩的视频格式。同样，JPEG 和 GIF 定义静止图像的格式。先看一下 MPEG，首先记住它定义视频流格式，而并不指明如何将这个流拆成网络分组。这样，MPEG 不但可用于存储在磁盘上的视频，也可用于在一个面向流的网络连接上传输的视频，类似 TCP 提供的连接。稍后，我们将更详细地说明如何将一个 MPEG 流拆成网络分组。

MPEG 格式是本书中讨论的协议中最复杂的一种。其复杂的原因在于要求编码算法对给定的视频流编码有各种可能的选择。复杂的原因还在于随着时间的推移，标准不断地发展（即 MPEG-1 和 MPEG-2）。以下描述的是 MPEG-2 视频流的主型（main profile）。你可以把 MPEG 的“型”看作是与某种“版本”类似，但在 MPEG 的首部中并不明确说明；接收方必须综合它所看到的首部字段，推断出视频流的型。

如图 7-16 所示，MPEG-2 流的主型具有嵌套结构。（注意这个图隐藏了许多琐碎的细节。）在最外层，视频包含一个由 SeqHdr 分隔的图片组（GOP）序列。该序列由 SeqEndCode（0xb7）结束。SeqHdr 位于每个 GOP 前，说明 GOP 中每个图片（帧）的大小（以像素和宏块为单位测量）、图片间隔期（以 ms 为单位测量）以及这个 GOP 内宏块的两个量化矩阵——一个用于帧内编码宏块（I 块），另一个用于帧间编码宏块（B 块和 P 块）。

由于这个信息针对每个 GOP 给出而不是如你可能希望的那样针对整个视频流做出说明，所以可以在整个视频的 GOP 边界上改变量化表和帧频。正如我们下面讨论的，这就使它能适应视频流随时间的变化。

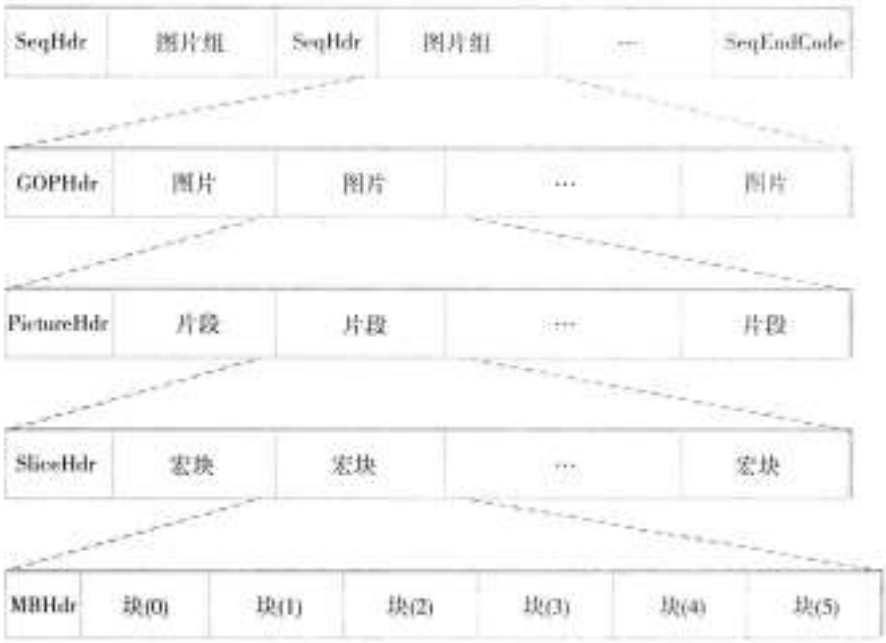


图 7-16 压缩的 MPEG 视频流格式

每个 GOP 由一个 GOPHdr 给出，后面跟着构成这个 GOP 的一组图片。GOPHdr 说明这个 GOP 内图片的数目，以及这个 GOP 的同步信息（即当 GOP 播放时，相对于这个视频的开始）。每个图片依次由 PictureHdr 和构成这个图片的一组片段（slice）给出。（一个片段是这个图片的一个区域，例如一条水平线。）PictureHdr 标识图片的类型（I、B 或 P），除此之外还定义特定图片的量化表。SliceHdr 给出片段的顶点位置，并给出一次改变量化表的机会——通过乘一个常量比例因子而不通过给出一个全新的量化表。接着，在 SliceHdr 后面跟着一系列宏块。最后，每个宏块包含一个首部，用来说明块在图片内的地址，以及这个宏块内的 6 个数据块：1 个 U 分量，1 个 V 分量和 4 个 Y 分量。（回忆一下，Y 分量是 16×16 的，而 U 分量和 V 分量是 8×8 的。）

显然 MPEG 格式的功能之一就是它能给编码程序一个随时修改编码的机会。它能更改帧速率、分辨率、定义 GOP 的混合帧类型、量化表和用于单个宏块的编码。因此，通过权衡网络带宽与图片质量，它就能适应在网上传输的视频的速率。网络协议怎样利用这种适应性是当前的一个热门研究课题（见“相关主题”）。

在网上发送视频流的另一个重要问题是如何把流恰好拆成分组。如果在 TCP 的连接上发送视频流，拆成分组是不成问题的，TCP 判定何时有足够多的字节发送下一个 IP 数据报。但使用交互式视频时，极少在 TCP 上传输，因为 TCP 对丢失数据段的重传会产生不可接受的延迟。如果我们用 UDP 传输视频，就要谨慎地选择流的拆分点，例如，可以选择宏块的边界。这是因为我们想把一个丢失的分组造成的影响限制在一个宏块中，而不会因为一个丢失的数据段而同时损坏几个宏块。这就是一个应用层组帧的实例，5.4 节已经讨论过这一主题。

拆分视频流只是在网上发送 MPEG 压缩视频的第一个问题。下一个复杂的问题是处理分组的丢失。一方面，如果 B 帧被网络丢弃，那么它可能简单地重放前一帧，不会严重地影响视频播放。30 帧中丢失 1 帧不是大问题。另一方面，丢失 I 帧会有严重的后果——没有它，随后的 B 帧和 P 帧就不能处理。因此，丢失 I 帧会导致丢失若干视频帧。虽然你可以重传故障的 I 帧，但因此产生的延迟在实时视频会议中大概是不可接受的。这个问题的一个解决方案是使用 6.5.3 节介绍的区分服务技术，将含有 I 帧的分组标记为比其他分组小的丢失概率。

最后注意怎样选择编码视频不只取决于可用网络带宽，还取决于应用程序对延迟的限制。像视频会议这样的交互式应用程序要求延迟很小。关键的因素是在 GOP 中 I 帧、P 帧和 B 帧的组合。考虑下列 GOP：

IBBBBPBBBBI

这个 GOP 使视频会议应用程序产生的问题在于，发送方必须延迟 4 个 B 帧的传输，直至得到它们后面的 P 帧或 I 帧。这是因为每个 B 帧依赖于后续的 P 帧或 I 帧。假设视频以 15 帧/s 的速率显示（即 1 帧/67ms），这就意味着第一个 B 帧被延迟 4×67ms，大于 0.25s。这个延迟不包括由网络产生的传播延迟。0.25s 远远大于人眼可感知的 100ms 的最低限度。为此许多视频会议使用 JPEG 编码视频，JPEG 常常被称为运动-JPEG。（由于所有帧都能独立，因此运动-JPEG 还能解决丢失参考帧的问题。）但是要注意，一个只依赖于先前的帧而不依赖后面的帧的中间帧的编码不会成为问题。这样，一个形如

IPPPPI

的 GOP 会在交互式视频会议中工作正常。

相关主题

自适应视频编码

我们已经讲过，使用 MPEG 的视频编码允许在图像的质量和消耗的带宽之间进行权衡。相反，显然在某些质量级别上操作的视频压缩算法的输出带宽一般不会是常量，而是根据视频流中细节和运动的数量随时间变化的。这些事实引发了一些重要的问题：如何设计一个在分组交换网上传输压缩视频的系统。

假设我们有一个视频编解码器，它以 R bps 的平均速率输出压缩的视频流，但是偶然会暴涨到 $3R$ bps。假使我们让视频流通过一个“平滑缓冲区”平滑掉瞬时传输速率中的峰值，我们就可以在容量为 R 的固定带宽管道（如一条租用线路或 CBR 电路）上传输这个视频流。现在，可能在某一时刻平滑缓冲区发生溢出，也许是由于电影中一系列很长的动作导致视频编码解码器在较长一段时期内的高速率输出。这时，我们可以暂时增加压缩数量，由此降低数据速率（和图片质量），容许腾空平滑缓冲区。当平滑缓冲区接近空时，我们再提高编码质量。

我们在分组交换网上能做很多相同的工作，但是不用平滑缓冲区。假设我们有办法测量可用容量以及沿一条路径的拥塞程度，比如，使用 6.5.5 节讲述的基于等式的拥塞控制算法。在可用带宽波动时，我们可以把信息反馈到编解码器以便调整编码参数，使其在拥塞期间缩减，且在网络空闲时更积极地（用较高图片质量）发送。这类似 TCP 的行为，除了传输视频的情况，我们宁愿实际修改发送的总数据量也不愿意修改发送定量数据需要花费的时间，因为我们不想把延迟引入视频应用程序。

如果我们把一个视频流多播（multicasting）到许多接收方就会出现一个有趣的问题。由于视频流可能经历程度不同的拥塞，那么如何为每个接收方选择恰当的速率？解决这个问题的巧妙方案是将被传输的视频流拆分为一些“层”。第一层包含图片所需的基本细节，而后续的层加入更多由更高频信息组成的细节。然后每一层发送到不同的多播组地址，这样每个接收方可以决定要加入多少层。如果接收方A经历严重拥塞，那么它就只参加携带基本层的多播组，而接收方B可能加入所有层。接收方A可以定期尝试参加下一层的细节看看是否有足够带宽可用。这种方法称为接收方驱动的分层多播（receiver-driven layered multicast，RLM）。一个值得研究的问题是，如何建立一组最励使接收方参加恰当数目的组而不参加所有组，因为参加太多的组会引发不必要的网络拥塞。

7.2.5 音频压缩（MP3）

MPEG不仅定义如何压缩视频，它还定义了压缩音频的标准。这个标准可用于压缩电影的音频部分（此时，MPEG标准定义在一个MPEG流中压缩音频与压缩视频如何交错），或压缩独立的音频（例如音频CD）。

为了清楚音频压缩，我们必须从数据开始。CD质量的音频实际上是高品质音频的数字表示，它是以44.1kHz频率采样的（即大约每23ms采样一次）。每个样本16比特，它意味着一个立体声（2声道）音频流产生一个如下的比特速率：

$$2\times44.1\times1\,000\times16=1.41\text{Mbps}$$

比较起来，电话质量的声音是以8KHz频率采样的，具有8比特的样本，产生64kbps的比特速率，并非巧合，这正是ISDN链路的速度。

显然，在容量为128kbps的一对ISDN数据/声音线上传输CD质量音频需要一定量的压缩。尤其糟糕的是，同步和错误校正开销需要用49比特编码每个16比特样本，这使得实际的比特速率为

$$49/16\times1.41\text{Mbps}=4.32\text{Mbps}$$

MPEG通过定义三级压缩来解决这种需求，表7-2中给出其定义。其中，Layer III（更广为人知的名称为MP3）是最常用的。

表7-2 MP3的压缩率

编 码	比特速率	压缩因子
Layer I	384kbps	4
Layer II	192kbps	8
Layer III	128kbps	12

为了达到这样的压缩率，MP3使用MPEG压缩视频所使用的类似技术。首先，它把音频流拆分为某些频率的子波段，大致类似于MPEG分别对视频流分量Y、U和V的处理方法。其次，每个子波段被分成一系列的块，除了其长度可以在64～1 024个样本之间变化外，它类似于MPEG的宏块。（编码算法可以根据某些失真效果改变块的大小，音响失真效果超出我们讨论的范围。）最后，就像对MPEG视频那样，每个块用改进的DCT算法进行变换、量化和赫夫曼编码。

MP3的诀窍在于选择使用多少子波段以及为每个子波段分配多少比特，要记住这就是尝试产生目标比特速率允许的最佳质量的音频。如何准确完成这些分配是由音质模型来控制的，这超出本书的讨论范围。但为了说明这个思想，考虑在压缩男声时分配较多比特给低频子波段，而在压缩女声时分配较多比特给高频子波段，这是有道理的。在操作上，MP3动态修改每个子波段的量化表，使每个子波段达到理想的效果。

一旦进行压缩，子波段就被打包成定长的帧，而且附上一个帧首部。这个帧首部不但

包括同步信息，而且还包括解码器为确定编码每个子波段使用多少比特所需要的比特分配信息。如上所述，这些音频帧就能与视频帧交替形成完整的 MPEG 流。值得说明的一点是，在可能发生拥塞的网络中，虽然丢弃 B 帧可行，但经验告诉我们，丢弃音频帧不是一个好主意，因为与劣质音频相比用户更能容忍劣质视频。

7.3 小结

本章描述了网络分组中的应用程序数据是如何编码的。与本书前面介绍的网络协议不同，那里你可以把协议看作处理消息（message），而这里的转换是处理数据（data）。多媒体数据类型，例如视频、静态图像、音频，都推动着这个领域的发展。

第一个问题是表示格式化，其中难点是格式化应用程序计算的不同数据类型：整数、浮点数、字符串、数组以及结构。这既涉及计算机和网络之间字节顺序的转换，也涉及复合数据结构的线性化。我们概述了表示格式化的设计空间，以及在这个设计空间中处于不同位置的四种特殊机制：XDR、ASN.1、NDR 以及越来越重要的 XML。

第二个问题是压缩，它所关心的是减少传输不同类型的数据所需的带宽。压缩算法可以是无损压缩或者是有损压缩，有损压缩算法最适合图像和视频数据。JPEG、MPEG 和 MP3 是有损压缩协议的例子，分别用于静态图像、视频和音频数据。MPEG 系列等视频压缩和编码格式要继续发展，以在有限的可用宽带内满足更高的质量需求。

接下来会发生什么：无处不在的视频

有一点几乎不用强调，视频是现在因特网流量的主要部分，它具有三屏幕（three-screen）能力，即可以交付给电视、电脑和手机，这是计算、通信和娱乐业的首要目标。这引发了很多有趣的问题。问题之一是这些视频都会影响因特网所需宽带。今天从互联网有足够的配置能力到把娱乐视频传播给每一个看电视或租 DVD 的人，我们还有很长一段路要走。这不仅推动了人们对更大网络容量的需求，也促进了“内容为中心网络”等新型网络体系结构的提出。

与本章内容更直接相关的是，在这个新的视频格式无处不在的环境中，需求聚集在编码和表示格式上。作为这个部分所面临的挑战的一个例子，考虑这样一个事实，即视频流中的三个主要开发者（Microsoft®、Adobe®和 Apple®）都开发出了自己的从网站到浏览器的不同视频流协议。用户可以在浏览器上安装恰当的插件来解决大多数的不兼容情况，但是在一些不灵活的设备上，比如手机，就会出现明显的混乱。随着网络电视和机顶盒的出现，不兼容视频格式似乎就变成了一个持续让用户烦恼的来源。

HTML5 的新兴标准试图确保至少有一个被所有浏览器都支持的通用的视频解码器和格式，以此来成为它们最低的共同标准，但是这个过程至今还没能实现。专利问题导致了参与标准制定过程的人犹豫是否要确定一个格式，因为他们害怕一旦专利被侵犯，它们的不知名的所有者就会露面表示不满。

由于 IP 视频无处不在而成为人们关注焦点的另一个问题是设备的方便配置和管理。尽管现在一些因特网用户对于配置网络参数得心应手（我的 ISP 应该使用 DHCP 还是 PPoE?），但是对于一般的电视买家来说，他们不太可能想进行比调台按钮更复杂的配置。对于 IP 设备的即插即用配置仍然是一个很重要的目标，这样即便不是网络专家也有能力排除这些设备的故障。这是管理家庭设备这个更大的问题的一部分，目前已成为网络上的

热门话题之一。

扩展阅读

我们本章的推荐阅读列表的前两篇文章分别给出 JPEG 标准和 MPEG 标准的概述。它们的主要价值是阐明形成标准的各种因素。我们还推荐一篇有关接收方驱动的分层多播的文章，作为系统设计方法的一个优秀实例，它包含多播问题、拥塞控制问题和视频编码问题。

- Wallace, G. K. The JPEG still picture compression standard. *Communications of the ACM* 34 (1): 30-44, April 1991.
- Le Gall, D. MPEG: A video compression standard for multimedia applications. *Communications of the ACM* 34 (1): 46-58, April 1991.
- McCanne, S., V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *Proceedings of the SIGCOMM'96 Symposium*, pages 117-130, September 1996.

遗憾的是，没有一篇文章给出表示格式的详尽论述。除了 XDR、ASN.1/BER 和 NDR 的规范（见 Eisler [Eis06]、CCITT 建议标准 [CCITT92a, CCITT92b]，以及开放软件基金会 [OSF94]）之外，还有另外三篇文章包含有关表示格式化的主题：O' Malley 等 [OPM94]、Lin [Lin93] 和 Chen 等 [CLNZ89]。这三篇文章都讨论有关性能的问题。

关于压缩的话题，最好从赫夫曼编码开始。赫夫曼编码最初定义在 [Huf52] 中。最初的 LZ 算法是在 Ziv 和 Lempel [ZL77] 中介绍的，并且其改进算法也是由同一作者发表在 [ZL78] 中。这些文章都是纯理论的。把 LZ 方法引入广泛的实践领域的工作可在 Welch [Wel84] 中找到。关于压缩问题更全面的概述，我们推荐 Nelson [Nel92]。你还可以从最新的一些关于多媒体的书中学习有关压缩的知识。我们推荐 Witten 等 [WMB99]，它有很高的科学价值，且很少有天花乱坠的宣传，而 Buford [Buf94] 汇集了跨越多媒体各个主题的一些重要章节。对 MPEG 标准的全面描述见 Mitchell 等 [MPFL96]。对 MP3 的描述见 Noll [Nol97]。

最后，我们推荐下列时常更新的网站作为参考：

- http://www.w3.org/TR/REC.xml/：最新的 XML 标准。
- http://mpeg.chiariglione.org/：MPEG 主页，很多 MPEG 资料的来源。

习题

1. 考虑如下的 C 代码：

```
#define MAXSTR 100

struct date {
    char month[MAXSTR];
    int  day;
    int  year;
};

struct employee {
    char  name[MAXSTR];
```

```
    int     ssn;
    struct date *hireday;
    int     salary_history[5];
    int     num_raises;
};

static struct date date0 = {"DECEMBER", 2, 1998};
static struct date date1 = {"JANUARY", 7, 2002};

static struct employee employee0 = {"RICHARD", 4376,
                                    &date0,180000, 85000,
                                    90000, 0, 0}, 21};
static struct employee employee1 = {"MARY", 4377,
                                    &date1, 190000,
                                    150000, 0, 0, 0}, 1};
```

其中，num_raises+1 对应于数组 salary_history 中有效条目的个数。请给出由 XDR 生成的 employee0 的实际表示法。

2. 对于上题，请给出由 XDR 生成的 employee1 的实际表示法。
3. 对习题 1 给出的数据结构，请给出对这个结构进行编码/解码的 XDR 例程。如果你有可用的 XDR，运行这个例程并测试它对一个 employee 结构的实例进行编码和解码所花费的时间。
4. 使用库函数 htonl 和 Unix 的 bcopy 或 Windows 的 CopyMemory 实现一个例程，该例程产生习题 1 中给定结构的实际表示与 XDR 产生的完全相同。如果可能，比较这个“手写”的编码/解码器与相应的 XDR 例程之间的性能差别。
5. 使用 XDR 和 htonl 分别对有 1 000 个元素的整型数组进行编码。测量并比较每种方法的性能。对于读和写 1 000 个元素的整型数组的简单循环如何进行比较？分别在一个本地字节顺序和网络字节顺序相同的计算机上以及一个本地字节顺序和网络字节顺序不同的计算机上进行实验。
6. 写出你自己的 htonl 实现。使用你的 htonl 和（如果低端字节序表示法硬件可用）标准库版本，进行恰当的实验来确定字节交换整数比只拷贝它们多花多长时间。
7. 给出如下 3 个整数的 ASN.1 编码。注意 ASN.1 整数与 XDR 中的一样，长度为 32 位。
   (a) 101。
   (b) 10 120。
   (c) 16 909 060。
8. 给出下列 3 个整数的 ASN.1 编码。注意 ASN.1 整数与 XDR 中的一样，长度为 32 位。
   (a) 15。
   (b) 29 496 729。
   (c) 58 993 458。
9. 给出习题 7 中整数的高端字节序和低端字节序表示。
10. 给出习题 8 中整数的高端字节序和低端字节序表示。
11. 用 XDR 对图 5-18 所示的 SunRPC 协议首部进行编码和解码。XDR 的版本是由 RPCVersion 字段决定的。这样做可能有什么困难？用新版本的 XDR 能否转变成低端字节序整型格式？
12. 表示格式化过程有时被看作是与应用程序分开的独立协议层。假设如此，为什么在表示层包含数据压缩不是一个好主意？
13. 假设你有一台字长为 36 位的计算机，串被表示为每个字包含 5 个压缩的 7 位字符。为了让这台计算机能与其他计算机交换整数和串数据，必须解决什么表示法问题？
14. 选择一种能支持用户定义的自动类型转换的程序设计语言，定义类型 netint 并提供在 int 和 netint 之间进行赋值和相等比较的转换。推广这种方法可以解决网络参数排列问题吗？
15. 不同体系结构对位顺序以及字节顺序有不同约定，例如，字节的最低有效位是第 0 位还是第 7 位。

[Pos81]（在附录B中）定义标准的网络位顺序。为什么位顺序与表示格式无关？

★16. 在网络中，令 $p \leqslant 1$ 是用高端字节序表示的分数，$1-p$ 则用低端字节序表示。假设我们随机选择两台计算机并把 int 型数据从一台计算机发送到另一台计算机。对于 $p=0.1$、$p=0.5$ 和 $p=0.9$，给出用于高端字节序表示法网络字节顺序格式和接收方调整格式所需的字节顺序转换的平均数。（提示：两个端点都使用高端字节序表示法的概率是 p^2，两个端点都使用不同字节顺序的概率是 $2p(1-p)$。）

17. 描述一个能让 XML 文档更简短且更有效的 XML 表示形式。

18. 用一个压缩实用程序（如 compress、gzip 或 pkzip）做实验。你可以获得什么样的压缩率？能否产生一些数据文件，你对它可以获得 5：1 或 10：1 的压缩率？

★19. 假设一个文件包含字母 a、b、c 和 d。名义上我们要求这样的文件中的每个字母用两个比特储存。

(a) 假设字母 a 出现的概率为 50%，b 出现的概率为 30%，而 c 和 d 出现的概率均为 10%。给出一种每个字母为 2 比特串的编码，这种编码提供优化的压缩。（提示：对 a 使用 1 个比特。）

(b) 你提出的编码方案达到的压缩比是多少？（这是对每个字母达到的平均压缩比，由字母的频率加权。）

(c) 假设字母 a 和 b 出现的频率均为 40%，c 出现的频率为 15%，d 出现的频率为 5%，重做此题。

★20. 假设有一个压缩函数 c，将一个比特串 s 压缩后为比特串 $c(s)$。

(a) 说明对任何整数 N，一定有一个长度为 N 的串 s，满足 $\text{length}(c(s)) \geqslant N$，就是说，所进行的是无效的压缩。

(b) 压缩一些已经压缩过的文件（试用同样的实用程序依次压缩几次）。文件的长度发生什么变化？

(c) 已知如 (a) 所述的压缩函数 c，给出一个函数 c'，对所有比特串 s，$\text{length}(c'(s)) \leqslant \min(\text{length}(c(s)), \text{length}(s))+1$，就是说，在最坏的情况下，用 c' 压缩只使长度扩大 1 比特。

21. 给出一个行程编码算法，要求只用单个字节表示不重复的符号。

22. 在一个给定的文本文件中，编写一个程序构造一部包含所有“词”的字典，词定义为连续非空格串。我们可以通过把每个词表示为词典的索引来压缩该文件（忽略空格信息的丢失）。下载包含 [Pos81] 的 rfc791.txt 文件，并在其上运行你的程序。首先假设每个词用 12 比特编码（这应该是足够的），而 128 个最常用的词用 8 比特编码，其余的词用 13 比特编码，给出压缩后文件的大小。假设字典本身能按每个词占 length（word）+1 字节存储。

★23. 除了丢弃第二个变量（j 或 y）和第二个余弦因子外，一维离散余弦变换（DCT）类似于二维变换。我们也丢弃 DCT 逆变换的前导系数。对 $N=8$ 实现 DCT 变换和逆变换（用电子表格完成，虽然用支持矩阵的语言可能更好）并回答下列问题：

(a) 如果输入数据为 (1,2,3,5,5,3,2,1)，哪些 DCT 系数接近 0？

(b) 如果输入数据为 (1,2,3,4,5,6,7,8)，我们必须保留多少个 DCT 系数，才能使得 DCT 逆变换后的值都在其原值的 1%或 10%以内？假设丢弃的 DCT 系数用 0 代替。

(c) 令 s_i 是在 i 为 1 和 j 时为 0 的输入序列，其中 $1 \leqslant j \leqslant 8$，$j=i$。让我们对 s_i 应用 DCT，令最后三个系数为 0，然后对其应用 DCT 逆变换。结果中哪个 i 位置引起的错误最小？哪个 i 位置引起的错误最大？其中 $1 \leqslant i \leqslant 8$。

24. 比较 JPEG 格式的全白图像与相同大小的普通照片图像。在 JPEG 压缩过程的哪个阶段或哪些阶段，制作白图像会小于普通照片？

对下面的三道习题，可以用实用程序 cjpeg 和 djpeg，并可从 http://www.ijg.org/站点下载。也可以使用其他 JPEG 转换程序。对于手工建立和检验图像文件，推荐 pgm 可移动灰度格式，参见 Unix pgm(5)/ppm(5) 手册。

25. 建立一个由 8×8 网格、首列为垂直黑线组成的灰度图像。压缩成 JPEG 格式后再解压缩。与默认质量设置下产生的字节相差多少？如何描述引入的视觉偏差？什么样的质量设置恰好能恢复文件？

26. 建立一个由有 64 个 ASCII 字符的文本串组成的 8×8 灰度图像。只使用小写字母，不用空格和标点。压缩成 JPEG 格式然后再解压缩。其结果作为文本可识别的程度如何？为什么加入空格事情会变得更

槽？若质量设置为 100，这会是一个合理的文本压缩方法吗？

27. 使用浮点算术，写一个程序实现正向 DCT 和反向 DCT。在一个样本图像上运行该程序。由于 DCT 是无损的，因此由程序输出的图像应该与输入图像相匹配。修改你的程序使其将一些高频分量变为 0，并查看输出图像会受到什么影响。这与 JPEG 的做法有何区别？

28. 用各个 pixel(x,y) 的平均值来表示 DCT(0，0)。

29. 考虑一下人们希望视频标准能够提供哪些合理的功能，例如快进、编辑能力、随机存取等。（参见本章“扩展阅读”中给出的 Le Gall 的论文“MPEG：A video compression standard for multimedia applications”。）根据这些性质解释 MPEG 的设计。

30. 对 MPEG 流，假设你希望实现快进和快退。假如限制你的装置只显示 I 帧，运行过程中会出现什么问题？如果不做限制，显示快进序列中一个给定的帧，在原序列中你必须解码的最大帧数是多少？

31. 使用 mpeg_play 播放一个 MPEG 编码的视频。试验各选项，特别是-nob 和-nop，它们分别用于省略流中的 B 帧和 P 帧。省略这些帧会出现什么可视效果？

32. mpeg_stat 程序可以用来显示视频流的统计数据。对多个流使用该程序，确定：

(a) I、B 和 P 帧的数量和顺序。

(b) 整个视频的平均压缩率。

(c) 每种帧的平均压缩率。

33. 设想我们有一个在黑背景上两个白点以均匀速率相向移动的视频。我们借助 MPEG 对其进行编码。在一个 I 帧中两个点分别为 100 像素，在下一个 I 帧中它们已合并了。最终合并的点刚好位于 16×16 宏块的中心。

(a) 描述如何对插入的 B 帧（或 P 帧）的 Y 分量进行最佳编码。

(b) 假设点是彩色的，而且当点移动时色彩缓慢变化。描述 U 和 V 可能如何编码。

第8章
Computer Networks: A Systems Approach

网络安全

将上帝的保护与人的脆弱性相结合才是难能之举。

——塞内加

问题：安全攻击

计算机网络是一种典型的共享资源，被很多代表不同利益的应用程序所使用。因特网就是一种被广泛共享的资源，相互竞争的商业对手、相互敌对的政府以及投机的犯罪分子都在使用它。如果不采用安全措施，攻击者可能会危及网络会话或分布式应用的安全。

考虑一些对安全使用网络（如万维网）的威胁。假设你是一个使用信用卡从网站订购商品的顾客。一个很明显的威胁是攻击者可能会窃听网络通信，读取消息来获得你的信用卡信息。那么如何实现这种窃听呢？在广播网络（如以太网）上实现窃听是很容易的，因为任何节点都可以通过配置来接收网络上的所有流量。对于无线通信，则可以在没有物理连接的情况下实现窃听。更复杂的窃听方法包括搭线窃听和在任何相关节点上安装间谍软件。只有在非常极端的情况下（如国家安全）才会采用严格的措施来阻止窃听，而因特网则不属于这种情况。但是，通过加密消息来防止攻击者读取消息的内容是可能而且可行的。采取了这种措施的协议称为提供了机密性（confidentiality）。更深一步，隐藏通信的数量和目的地称为流量机密性（traffic confidentiality）——因为有时仅仅知道向某个地址发送的通信量对攻击者来说也是有帮助的。

对于网站的顾客来说，即使保证了保密性，仍然存在其他威胁。攻击者虽然不能读取加密消息的内容，但仍能修改其中的一些位以便构成另一个有效的订单，如完全不同的商品或1 000个同样的商品。有些技术虽然不能阻止这种篡改，但可以检测到篡改。能够检测篡改的协议称为提供了数据完整性（data integrity）。攻击者也可能利用重放攻击（replay attack）把你的消息再重新发送一遍。对于网站来说，好像是你又订购了一份与第一次相同的商品。能够检测重放攻击的协议称为提供了原始性（originality）。然而，原始性并不能阻止攻击者拦截你的订单，并且等待一段时间后再传输该订单——实际上是推迟你的订单。攻击者可以安排在你外出度假的时候把商品送到你家门口，此时他就可以很容易地窃取你的商品。能够检测这种延迟攻击的协议称为提供了实效性（timeliness）。数据完整性、原始性和实效性被看作是一般属性——完整性（intergrity）的不同方面。

对顾客的另一个威胁是在不知情的情况下被重定向到一个错误的网站。这可能是由于域名系统（DNS）攻击造成的，即错误信息被输入到域名服务器或顾客计算机的域名服务缓存中。这会导致将正确的URL翻译成错误的IP地址——错误网站的地址。能够确保你的确在与一个你认为正在与之通话的人通话的协议称为提供了认证（authentication）。认证与完整性是紧密联系的，因为如果消息已经不是原始消息了，那么确认该消息来自特定的人是没有意义的。

网站的拥有者也可能被攻击。有些网站的外观已经被破坏，网站文件在未授权的情况下

被远程访问并修改。这就是访问控制（access control）问题：强制实施用于规定谁可以做什么的规则。网站也会受到拒绝服务（Denial of Service，DoS）攻击。网站被攻击期间，顾客不能访问网站，因为网站已经被假请求淹没。确保一定程度的访问称为可用性（availability）。

除了这些问题，因特网曾被大量用于部署恶意代码，它们利用了端系统中的漏洞。蠕虫（worm）是在网络上自我复制的代码段，它已经出现了几十年，并在继续制造麻烦，与之相关的病毒（virus）也是这样，病毒是通过传输被感染的文件来传播的。被感染的计算机就可以被部署成僵尸网络，实施进一步的破坏，如发起 DoS 攻击。

虽然因特网提供冗余来应对诸如链路或路由器损坏之类的问题，但它的原始设计中并没有提供我们所讨论的安全性。因特网安全机制实质上是补丁。如果真的要对因特网进行全面的重新设计，那么集成安全性将会是最重要的推动因素。这可能使得本章更加切题。

现在有很多用于保护联网系统安全的工具，从各种密码术到专用设备，如防火墙。本章将介绍这些工具，并重点关注使用密码学方法来增强网络安全性。增强网络安全性仍然是一个快速变化且需要大量研究工作的领域。

8.1 密码基础

我们将一步一步地介绍以密码学为基础的安全概念。第一步是密码算法——密码和密码族列——在本节中介绍。密码算法本身不能作为安全解决方案，而是构建安全方案的基础。密钥（key）是密码算法的参数，8.2 节解决分发密钥的问题。下一步（见 8.3 节），我们将描述如何将密码模块集成到协议中以便为拥有正确密钥的参与者提供安全的通信。最后，8.4 节分析了几个当前正在使用的完整的安全协议和系统。

8.1.1 密码原理

加密对消息作变换，使得任何不知道如何做逆变换的人都不能理解该消息。发送方把加密（encryption）函数应用于原始明文（plaintext）消息，使其变为密文（ciphertext）消息，再发送到网络上，如图 8-1 所示。接收方应用一个秘密的解密（decryption）函数——加密函数的逆函数——恢复出原始明文。如果窃听者不知道解密函数，那么他就不能理解网络中传输的密文。由加密函数及相应的解密函数所表示的转换称为密码（cipher）。

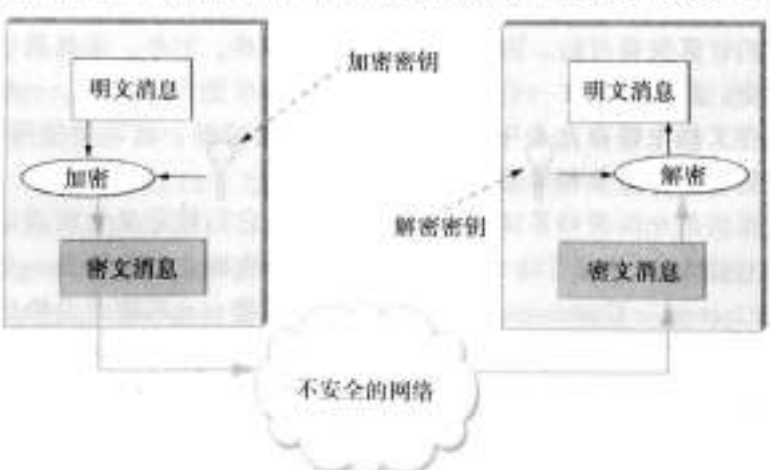


图 8-1 对称密钥加密和解密

密码学家最早在1883年就提出了上述原理。加密和解密函数应该以一个密钥（key）为参数，并且函数应该可以公开——只有密钥应该保密。因此，针对给定明文消息所产生的密文依赖于加密算法和密钥。采用该原理的一个原因是，如果你依赖于密码算法的保密，那么当你认为它不再保密时，就必须放弃这个算法（而不光是密钥）。这意味着要频繁地修改密码算法，而开发一个新的算法要做大量工作，因此这样做是有问题的。而且，想知道某个密码算法是否有效的最好的方法是长时间使用它——如果没有人能破解它，它可能就是安全的。（幸好，许多人试图破解算法，并且当他们成功时会让尽量多的人知道，所以一般来说，没有消息就是好消息。）因此，开发一个新的算法会有相当大的开销和风险。最后，用密钥将密码算法参数化实际上为我们提供了一个非常大的密码算法族，更换密钥实质上是更换密码算法，从而限制密码分析者（cryptanalyst，即密码破译者）能用于破译密钥/密码算法的数据总量，以及成功破译后能够解密的数据总量。

对加密算法的基本需求是：它能用一种方法把明文转换成密文，只有指定的接收者——即解密密钥的持有者——才能把密文恢复成明文。这意味着，没有解密密钥者的人无法解密消息。

当一个潜在攻击者接收到一段密文时，他可能知道更多的信息，而不仅仅是密文本身。认识到这一点很重要。例如，他们可能知道明文是用英文写的，这意味着在明文中字母e会比其他字母出现得更频繁，其他一些字母和常见字母组合出现的频率同样可以预测。这个信息可以极大地简化寻找密钥的工作。同样，他可能知道消息的一些有关内容，比如，“login”一词很可能出现在一个远程登录会话的开始。这可能引起一次已知明文（known plaintext）攻击，它比唯密文（ciphertext only）攻击有高得多的成功率。更好的攻击是选择明文（chosen plaintext）攻击，这种攻击向发送方加入一些有可能会被传送的信息，例如，在战时就发生过这样的事情。

因此，最好的密码算法可以防止攻击者在同时知道明文和密文时推测出密钥。这使得攻击者除了测试所有可能的密钥（穷举，“蛮力”搜索）外别无选择。如果密钥为n位，那么有2^n个可能的密钥（n位中的每一位可以是0或1）。攻击者可能非常幸运，能马上测试到正确的密钥，也可能非常不走运，需要测试所有2^n个可能的值。发现正确密钥值的平均猜测次数介于这两种极端情况之间，即$2^n/2$。可以通过选择足够大的密钥空间以及使检测密钥的运算的代价足够高，使得这种搜索在计算上不可行。但由于计算速度不断提高，使得以前不可行的计算变得可行，因此这种措施也很困难。另外，虽然我们关注数据在网络上传输时的安全，即数据只在一个较短的时间间隔内可能受到威胁。一般来说，人们应该考虑对于需要在文档中保存几十年的数据可能受到的威胁。这需要使用相当大的密钥。另一方面，大密钥也使得加密和解密更慢。

大部分密码算法是分组密码算法（block cipher），它们被定义为将固定长度的明文分组作为输入，典型值为64位或128位。用分组密码算法独立地加密每一个分组——常称为电码本模式（Electronic Codebook（ECB）mode）加密——其缺点是给定的明文分组总是产生相同的密文分组。因此从密文中重复重现的分组能够识别出明文中的重复分组，这使得密码分析者更容易破译密码。

为了避免上述问题，分组密码通常使分组的密文根据上下文的不同而变化，从而增加强度。增强分组密码强度的方法称为操作模式（modes of operation）。一种常用的操作模式是密码分组链（Cipher Block Chaining，CBC）模式。在CBC模式下，每一个明文分组

在被加密前都与前一分组的密文异或。结果是每一个密文分组都部分地依赖前面的所有分组（即依赖于上下文）。因为第一个明文分组前没有其他分组，因此与一个随机数异或。这个随机数称为初始化向量（Initialization Vector，IV），与一系列密文分组一起发送以便明文的第一分组能被解密。图 8-2 说明了这种模式。另一种操作模式是计数器模式（counter mode），计数器的连续的值（如 1，2，3，…）被结合到连续明文分组的加密过程中。

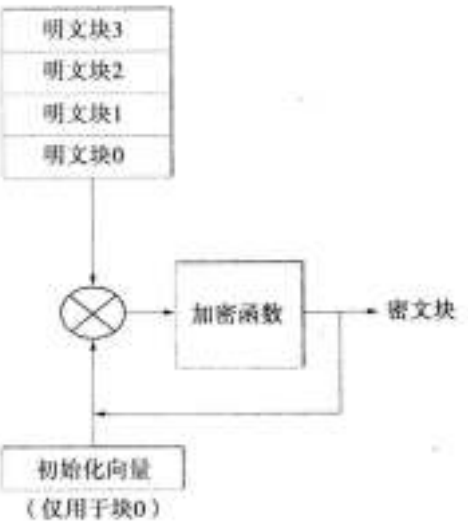


图 8-2 密码分组链（CBC）

8.1.2 对称密钥密码

在对称密钥密码中，两个参与者[⊖]共享一个秘密密钥。换句话说，如果用一个特定的密钥加密了一条消息，也必须要使用相同的密钥来解密该消息。如果图 8-1 描述的是一个对称密钥密码，那么加密和解密密钥应该是相同的。对称密钥密码也称为秘密密钥密码，因为共享的密钥一定是只有参与者才知道。（我们将在后面看到替代方案——公钥密码。）

美国国家标准与技术研究所（NIST）已经颁布了一系列对称密钥密码标准。数据加密标准（Data Encryption Standard，DES）是其中的第一个，它经受住了时间的考验，还没有发现比蛮力攻击更好的密码分析攻击。然而，蛮力攻击的速度已经变得很快。DES 的密钥有 56 个独立位（虽然密钥的总长度是 64 位，但每一个字节的最后一位是奇偶校验位），这对于目前的处理器来说太小了。正如前面提到的，平均需要搜索 2^{56} 个可能的密钥空间的一半来找到正确的密钥，即 $2^{55}=3.6\times10^{16}$ 个密钥。这看起来好像很多，但这种搜索可以高度并行化，因此将可以控制的尽可能多的计算机都用于执行该任务是可能的，而现在很容易找到数千台计算机（例如，Amazon.com 可以以每小时几美分的价格向你出租计算机）。到 20 世纪 90 年代后期，在几小时内搜索到 DES 密钥已成为可能。因此，NIST 在 1999 年更新了 DES 标准，指出 DES 应该只用于遗留系统中。

NIST 还标准化了三重 DES（Triple DES，3DES），它实际上通过增加密钥长度来有效地抵抗对 DES 的密码分析。3DES 密钥有 168（$=3\times56$）个独立的位，被用作三个 DES 密钥，分别称为 DES-key1、DES-key2 和 DES-key3。用 3DES 加密明文分组时，先用 DES-key1 对该分组做 DES 加密，然后用 DES-key2 对结果分组做 DES 解密，最后用 DES-key3 对上一步的结果做 DES 加密。解密包括用 DES-key3 解密，然后用 DES-key2 加密，最后用 DES-key1 解密。[⊜]

虽然 3DES 解决了 DES 的密钥长度问题，但它继承了其他一些不足。DES/3DES 的软件实现较慢，原因是 IBM 最初设计 DES 时是为了用硬件实现。另外，DES/3DES 使用

⊖ 我们使用术语参与者（participant）代表参与一个安全通信的实体，因为我们曾在整本书中使用该术语来表示一个信道的两端，在安全领域，他们一般称作主体人（principal）。

⊜ 3DES 加密时使用以 DES-key2 为密钥的 DES 解密的原因是为了与遗留 DES 系统互操作。如果一个遗留 DES 系统使用单个密钥，那么一个 3DES 系统将那个密钥用作 DES-key1、DES-key2 和 DES-key3，就能够实现相同的加密功能。在前两步，我们用相同的密钥先加密再解密，能够产生原始明文，然后我们再加密。

64 位的分组大小，而更大的分组会更高效且更安全。

3DES 正在被 NIST 在 2001 年发布的高级加密标准（Advanced Encryption Standard，AES）所替代。该标准所选择的密码算法（经过一些较小的改动）最初根据发明者的名字 Daemen 和 Rijmen 而命名为 Rijndael，发音类似于 "Rhine Dahl"。AES 支持 128 位、192 位或 256 位的密钥长度，分组长度为 128 位。AES 用硬件和软件都可以快速实现。它不需要太多内存，因此适用于小型移动设备。AES 具有一些数学上已证明的安全属性，并且截至到撰写本书时，还没有已知的成功攻击。[⊖]

8.1.3 公钥密码

对称密码的一个替代方案是非对称或公钥密码。公钥密码使用一对相关的密钥，一个用于加密，另一个用于解密，而不是在两个参与者之间共享一个密钥。这对密钥只属于一个参与方。密钥拥有者要保密解密密钥，因此只有密钥拥有者能够解密消息。这个密钥被称为私钥（private key）。密钥拥有者将加密密钥公开，因此任何人都可以为密钥拥有者加密消息，这个密钥被称为公钥（public key）。显然，为了使该方案奏效，必须保证不能从公钥推算出私钥。因此，一个参与方能够得到公钥并发送加密后的消息给公钥的拥有者，并且只有私钥的拥有者可以解密消息。图 8-3 描述了这种情况。

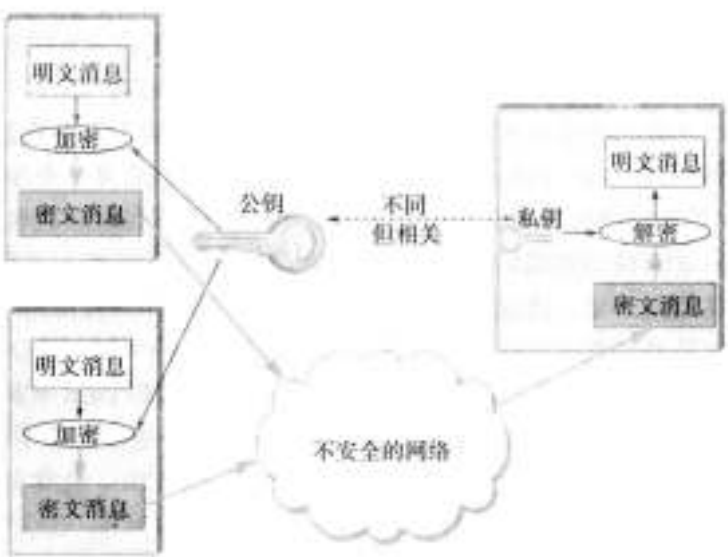


图 8-3 公钥加密

因为上述方案不太直观，因此我们强调公共的加密密钥对解密消息是没有用的——如果你没有私有的解密密钥，你甚至不能解密你自己加密的消息。如果你将密钥看作在参与者之间定义一个通信信道，那么公钥密码和对称密码的另一个区别是信道的拓扑。对称密码的密钥提供的是参与者之间的双向信道，每一个参与者都有相同的（对称的）密钥，任何一方都能在任何一个方向上加密或解密消息。与之相比，公钥/私钥对提供了一个从有公钥的每个人到（唯一的）私钥拥有者之间的多对一的单向信道，如图 8-3 所示。

⊖ 任何比纯粹的蛮力付出更小的计算开销而能够发现明文的行为在技术上都被归类成攻击，现在已经发表了一些针对 AES 的攻击。虽然这些攻击比蛮力好，但它们的计算量仍然很大。

公钥密码的另一个重要特性是私有“解密”密钥可以与加密算法一起使用来加密消息，因此这些消息只能用公共的“加密”密钥来解密。这个特性显然对机密性不起作用，因为有公钥的任何人都能够解密这样的消息。（确实，对于两个参与者之间的双向机密性，每一个参与者都需要自己的密钥对，并且每一方都用另一方的公钥加密消息。）。然而，这个特性对于认证有用，因为它告诉消息的接收者这样的消息只能由密钥的拥有者创建（有关假设会在后面讨论）。图 8-4 描述了这种情况。从图中可以清楚地看到，任何有公钥的人能够解密被加密的消息，假设解密后的结果与期望的结果匹配，那么可以断定加密是用私钥完成的。这种操作如何用于提供认证是 8.3 节的主题。正如我们将要看到的，公钥密码主要用于认证和秘密地分发对称密钥，而消息的保密则依赖对称密码。

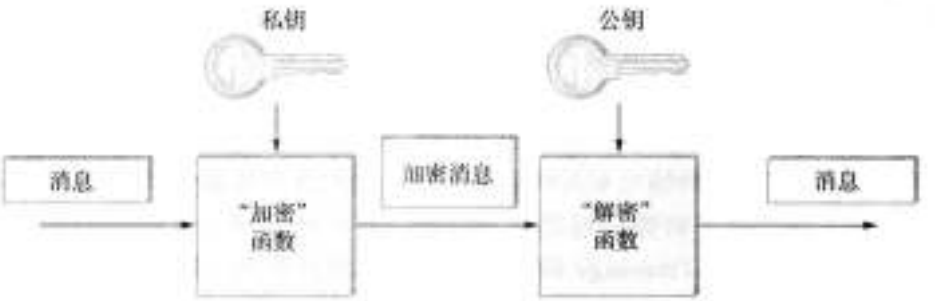


图 8-4 使用公钥的认证

一段有趣的历史：公钥密码的概念最早由 Diffie 和 Hellman 在 1976 年发表。然而，有资料证明英国的通信电子安全小组（Communications-Electronics Security Group）在 1970 年就发现了公钥密码，而美国国家安全局（NSA）则声明他们在 20 世纪 60 年代中期就发现了。

最著名的公钥密码是 RSA，根据其发明者 Rivest、Shamir 和 Adleman 的名字命名。RSA 依赖于大数因子分解的极大计算开销。找到分解大数的有效方法是在 1978 年 RSA 出现以前的很长时间内一直没有被数学家成功解决的问题，因此 RSA 对密码分析者的抵抗力进一步增强了人们对其安全性的信心。不幸的是，RSA 需要相对较大的密钥来保证安全，至少 1 024 位。这比对称密码的密钥要大，因为通过分解产生密钥对的大数来破解 RSA 私钥比通过穷举所有密钥空间来破解要快。

另一公钥密码是 ElGamal。就像 RSA 一样，ElGamal 也依赖于数学问题——离散对数问题，至今还没有找到有效的解决方法，并且需要至少 1 024 位的密钥。当输入是椭圆曲线时，就出现了离散对数问题的一个变形，一般认为这个问题更难计算。基于这个问题的密码机制称为*椭圆曲线密码*（elliptic curve cryptography）。

不幸的是，公钥密码比对称密码慢好几个数量级。因此，对称密码用于绝大部分加密，而公钥密码则用于认证（见 8.1.4 节）和密钥建立（见 8.2 节）。

8.1.4 认证码

加密本身不能提供数据完整性。例如，仅仅随机修改一条密文消息可能解密出看上去有效的明文，在这种情况下，接收方无法检测到篡改。加密本身也不能提供认证。如果消息已经不是原始消息了，而确认该消息来自特定的人是没有意义的。从某种程度上讲，完整性和认证在本质上是不可分割的。

认证码（authenticator）是一个值，包含在被传输的消息中，可用于同时验证消息的真实性和数据完整性。我们将在 8.3 节讨论认证码在协议中的使用。在这里，我们侧重于认证码的生成算法。

你应该还记得我们在 2.4.3 节讲了校验和以及循环冗余校验（CRC）——在原始消息上添加一些信息并一起发送出去——是检测消息因比特差错而被无意中修改的方法。类似的概念也适用于认证码，对于某些人故意破坏消息而不想被检测到的情况增加了挑战。为了支持认证，认证码包含一些证据来证明创建该认证码的人知道一个只有所声称的消息发送方才知道的秘密。例如，这个秘密可能是一个密钥，证据则是用这个密钥加密的值。冗余信息的形式与秘密证据的形式是相关的。我们会讨论几种可行的结合方式。

我们先假设原始消息不需要保密——被传输的消息由原始消息明文加上认证码组成。然后，我们会考虑需要保密的情况。

有一类认证码将加密与密码散列函数（cryptographic hash function）结合。密码散列算法被当作公开信息，就像密码算法一样。密码散列函数（也称为密码校验和（cryptographic checksum））是一种输出有关消息的足够冗余信息来发现任何篡改的函数。就像校验和或 CRC 能发现由噪声链路引起的比特差错，密码校验用于发现攻击者的蓄意破坏。它输出的值称为消息摘要（message digest），就像传统校验和一样，被追加在消息后。不管原始消息有多长，一个给定的散列函数产生的所有消息摘要都具有相同的位数。因为所有可能的输入消息空间比可能的消息摘要空间大，就会出现不同输入消息产生相同消息摘要的情况，就像散列表中的碰撞。

通过加密消息摘要能够创建认证码。接收方计算消息明文部分的摘要，并与解密后的消息摘要对比。如果二者相等，那么接收方可以断定该消息的确来自所声称的发送方（因为该消息一定曾经被正确的密钥加密过）并且没有被篡改。攻击者无法发送一条伪造消息和与之匹配的伪造摘要并能逃脱检测，因为他没有正确加密伪造摘要所需的密钥。然而，攻击者能够监听得到原始的明文消息及其被加密的摘要。然后攻击者计算原始消息的摘要（散列函数是公开的），并产生替换消息，从中寻找具有相同消息摘要的消息。如果找到这样的一条消息，就能将新消息与旧认证码一起发送而不被检测到。因此，要保证安全性，就要求散列函数具有单向性（one-way）：攻击者找到与原始消息具有相同摘要的明文消息在计算上是不可行的。

对于能够满足要求的散列函数，其输出必须均匀地随机分布。例如，如果摘要长度为 128 位，并且随机分布，那么为了找到与给定消息摘要匹配的另一条消息，平均需要测试 2^{127} 条消息。如果输出不是随机分布的——即某些输出比其他输出更有可能——那么对于某些消息，更容易找到具有相同摘要的另一条消息，这会降低该算法的安全性。如果你只是试图找到碰撞（collision）——产生相同摘要的两条消息——那么你平均要计算 2^{64} 条消息的摘要。这个惊人的事实正是“生日攻击”的基础——更多细节请参考习题。

最常用的密码散列算法包括消息摘要 5（Message Digest 5，MD5）和安全散列函数 1（Secure Hash Algorithm 1，SHA-1）。MD5 输出 128 位的摘要，SHA-1 输出 160 位的摘要。人们发现 MD5 的弱点已经有一段时间了，因此建议将 MD5 换成 SHA-1。就在最近，研究者发现了比蛮力攻击高效一些的 SHA-1 碰撞发现技术，但该技术在计算上还不可行。虽然碰撞攻击（collision attack——以发现碰撞为基础的攻击）不像原像攻击（preimage attack——以发现与给定消息碰撞的另一条消息为基础的攻击）的风险大，但都是严重的

脆弱点。NIST 已提出到 2010 年停止使用 SHA-1，鼓励使用由 SHA 的四种变形组成的 SHA-2。目前正在上演设计新散列 SHA-3 的竞争。

在生成加密消息摘要时，摘要的加密可以使用对称密钥密码或公钥密码。如果使用公钥密码，用发送方的私钥加密（私钥一般被认为用于解密），接收方或其他人可以用发送方的公钥解密摘要。

通过公钥算法用私钥加密的摘要称为数字签名（digital signature），因为它像手写签名一样提供了不可否认性。接收方收到带有数字签名的消息后，能够向第三方证明发送方的确发送了该消息，因为第三方能够使用发送方的公钥自己做验证。（用对称密钥加密的摘要不具备该特性，因为只有两个参与者知道密钥；另外，因为两个参与者都知道密钥，所谓的接收方有可能自己创建了该消息）。任何公钥密码都可被用于数字签名。数字签名标准（Digital Signature Standard，DSS）是一种已被 NIST 定义为标准化的数字签名格式。DSS 签名可以使用三种公钥密码中的任何一种，一种基于 RSA，另一种基于 ElGamal，第三种称为椭圆曲线数字签名算法（Elliptic Curve Digital Signature Algorithm）。

另一类认证码与上述认证码类似，但不加密散列值，而是使用一个类散列函数，以一个秘密值（只有发送方和接收方知道）为参数，如图 8-5 所示。该函数输出一个认证码，称为消息认证码（Message Authentication Code，MAC）。发送方将 MAC 附加到明文消息后，接收方用明文和秘密值重新计算 MAC，并将重计算的 MAC 与接收到的 MAC 相比较。



图 8-5 计算 MAC 与 HMAC

MAC 的一种常见变形是对明文消息和秘密值的串接应用密码散列（如 MD5 或 SHA-1），如图 8-5 所示。所得到的摘要称为散列的消息认证码（Hashed Message Authentication Code，HMAC），因为其本质上是 MAC。附加到明文消息上的是 HMAC 而不是秘密值。只有知道秘密值的接收方才能够计算出正确的 HMAC，并与接收到的 HMAC 比较。如果不是因为散列函数具有单向特性，那么攻击者就可能找到产生该 HMAC 的输入并将它与明文消息相比较以便确定该秘密值。

到目前为止，我们假设消息是不保密的，因此原始消息可以以明文形式传输。为了对带有认证码的消息提供机密性，只要加密整个消息及其认证码——MAC、HMAC 或被加密的摘要就足够了。记住，在实际中，机密性是用对称密钥密码实现的，因为它们比公钥密码快得多。此外，将认证码包含在加密过程中而增加的开销很小，且增强了安全性。一

种常见的简化方法是加密消息及其（原始）摘要，这样摘要只需加密一次，在这种情况下，整个密文消息被当作认证码。

虽然认证码看上去好像解决了认证问题，但我们在8.3节将会看到这只是解决方案的基础。然而，我们首先要解决的是参与者如何获得密钥的问题。

8.2 密钥预分发

要使用密码和认证码，通信参与者需要知道使用什么密钥。对于对称密钥密码来说，如何让一对参与者得到所共享的密钥？对于公钥密码来说，参与者如何知道公钥属于哪个特定的参与者？这两个问题的答案根据密钥是短期的会话密钥（session keys）还是长期的预分发密钥（predistributed keys）而不同。

会话密钥用于保护持续时间相对较短的单个通信（一个会话）的安全。一对参与者之间的每一个不同会话使用一个新的会话密钥。为了加快速度，会话密钥一般是对称密钥。参与者通过协议——会话密钥建立协议——决定使用哪个会话密钥。会话密钥建立协议本身应该是安全的（只有这样，攻击者才不能得到新的会话密钥），其安全性是以长期的预分发密钥为基础的。

之所以将会话密钥与预分发密钥分离，有如下几个原因：

- 限制密钥总的使用时间能够减少进行计算密集型攻击的时间，减少密码分析者可用的密文，同时也减少了当密钥被破解时泄漏的信息。
- 对对称密钥进行预分发是有问题的。
- 公钥密码更适合认证和会话密钥分发，但如果用于加密整个消息来提供机密性则太慢了。

本节将说明预分发密钥是如何分发的，8.3节将解释会话密钥是如何建立的。在下文中我们用“Alice”和“Bob”来代表参与者，这种方法在密码学文献中很常见。必须要记住，虽然我们用拟人化的术语来标识参与者，但经常要考虑的情况是软件或硬件实体之间的通信，如客户端和服务器，它们经常与参与的人没有直接关系。

8.2.1 公钥预分发

用于产生相匹配的公钥/私钥对的算法是公开的，其软件也是公开的。因此，如果Alice想使用公钥密码，她可以产生自己的公钥/私钥对，并隐藏私钥而公开公钥。但是，她如何公开自己的公钥呢——声明该公钥属于她——其他参与者也能够确保该密钥的确属于她。不能通过电子邮件或Web来公开密钥，因为攻击者能够伪造一个足可以乱真的声明，宣布密钥 x 属于Alice，而事实上 x 属于攻击者。

用于证明公钥与身份之间的绑定关系（哪个密钥属于谁）的一个完整的方案称为公钥基础设施（Public Key Infrastructure，PKI）。PKI先验证身份，然后以带外方式将身份绑定到密钥上。我们用“带外”表示网络以及组成该网络的计算机以外的东西，就像下面描述的情况。如果Alice和Bob是相互认识的个体，那么他们可以在同一个房间见面，Alice可以把她的公钥交给Bob，也许会写在一个名片上面。如果Bob是一个组织，而Alice作为个体可以出示传统的身份信息，可能包括一张照片或指纹。如果Alice和Bob是同一个公司拥有的计算机，那么系统管理员能够为Bob配置Alice的公钥。

带外建立密钥的方式看起来扩展性不强，但对于启动PKI已经足够了。Bob知道Al-

ice 的密钥是 x，此信息能够通过数字签名与信任概念相结合的可扩展方式广泛传播。例如，假设你已经在带外收到了 Bob 的公钥，那么你完全可以在密钥和身份方面信任 Bob。那么 Bob 可以给你发送一条消息，声明 Alice 的密钥是 x，因为你知道 Bob 的公钥，因此能够认证该消息来自 Bob。（记住，为了对声明进行数字签名，Bob 会追加一个用他的私钥加密的消息的密码散列值。）因为你相信 Bob 会说实话，所以你知道 Alice 的密钥是 x，虽然你从来没有见到过她，也没有与她交换过消息。使用了数字签名，Bob 甚至不需要向你发送消息，他可以只是简单地创建并发布一个被数字签名的声明，说明 Alice 的公钥是 x。这种关于公钥绑定关系的带有签名的声明称为**公钥证书**（public-key certificate），或简称为**证书**。Bob 可以向 Alice 发送该证书的一份拷贝，或者将它发布在网站上。当有人需要验证 Alice 的公钥时，只要他们相信 Bob 并知道他的公钥，他们就能得到该证书的拷贝，或者从 Alice 那儿直接获得。可以看到，从寥寥几个密钥（在本例中，只有 Bob 的公钥）开始，随着时间的推移，你能够建立起可信密钥的大集合。Bob 在本例中扮演的角色通常称为**认证机构**（Certification Authority，CA），而且当前因特网的安全主要依赖于 CA。Versign® 是一个著名的商业 CA。我们会在下面继续讨论本主题。

主要的证书标准之一是 X.509。该标准的很多细节都是开放的，只是规定了一个基本结构。证书必须包含：

- 被证明的实体的身份。
- 被证明的实体的公钥。
- 签发者的身份。
- 数字签名。
- 数字签名算法标识符（使用了哪个密码散列函数和密码算法）。

一个可选的组件是证书的过期时间。我们会在下面看到该特征的一个具体应用。

因为证书创建了身份与公钥之间的绑定关系，所以我们应该更深入地研究“身份”的含义。例如，一个证书的内容是“该公钥属于 John Smith”，那么，如果你不能区分该证书所说的是成千上万个 John Smith 中的哪一个，那么该证书的用处就不大。因此，证书必须对所要证明的身份使用有明确定义的命名空间，例如，经常为电子邮件地址和 DNS 域颁发证书。

PKI 能够使用不同方法来形式化信任表示，我们讨论两种主要的方法。

1. 认证机构

在该信任模型中，信任是二元的，你要么完全信任某个人或者完全不信任他。与证书结合就能创建**信任链**（chains of trust）。如果 X 证明某个公钥属于 Y，而且 Y 进一步证明另一个公钥属于 Z，那么，尽管 X 和 Z 从未见过面，仍然存在一条从 X 到 Z 的证书链。如果你知道 X 的密钥——你必须信任 X 和 Y——那么你可以信任那个给出 Z 的密钥的证书。换句话说，你所需要的只是一个证书链，所有证书都是由你信任的实体签名的，只要该证书链能够回到某个已知的实体即可。

认证机构（certification authority/certificate authority，CA）是一个（被某些人）声明为用于验证和颁发证书的可信实体。有商业 CA、政府 CA 甚至免费 CA。为了使用 CA，你必须要知道其密钥。如果你可以得到从某个已知其密钥的 CA 开始的签名证书链，那么你就能得到该 CA 的密钥。然后，你就可以相信由该 CA 签名的任何证书。

构建这种链的流行方法是用分层的树形结构进行排列，如图 8-6 所示。如果每个人有根 CA 的公钥，那么任何一个参与者可以向另一个参与者提供一个证书链，并且他知道这足以为那个参与者建立一条信任链。

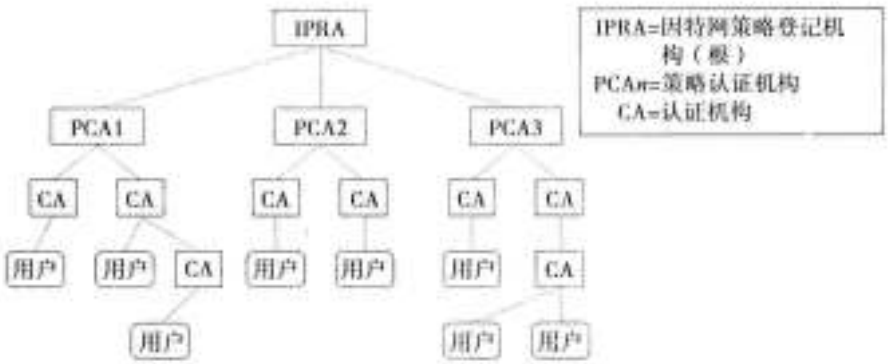


图 8-6 认证机构的分层树形结构

关于构建信认链还有一些重要问题。最重要的是，即使你确信拥有根 CA 的公钥，也必须保证根 CA 下面的每个 CA 正常工作。如果只有一个 CA 愿意为没有验证身份的个体颁发证书，那么一条看上去有效的证书链就变得没有意义了。例如，根 CA 可能向一个二层 CA 颁发了一个证书，证明证书中的名字与该 CA 的企业名称一致，但这个第二层 CA 可能希望向任何提出请求的人出售证书，而且不需要验证身份。信任链越长，这个问题就越严重。X.509 证书提供了一个选项，能限制证书主体可以依次证明的实体集合。

一个证书链可能有多个根，例如，这种情况在保护当前 Web 交易的安全时就很常见。Web 浏览器，如 Firefox 和 Internet Explorer，已经预装了一个 CA 集合的证书。事实上，浏览器的生产者已经确信这些 CA 以及它们的密钥是可信的。用户可以向浏览器认为可信的实体追加证书。安全套接字层/传输层安全（SSL/TLS）接受这些证书，SSL/TLS 协议经常用于保护 Web 事务的安全，我们将在 8.4.3 节进行讨论。（如果你感到好奇，你可以随便看一下浏览器的偏好设置，找到“查看证书”选项来查看你的浏览器被设置成信任多少个 CA。）

2. 信任网络

另一种信任模型是以良好隐私（Pretty Good Privacy，PGP）为例的信任网络（web of trust），该模型将在 8.4.3 节深入讨论。PGP 是用于电子邮件的安全系统，因此与公钥绑定的是电子邮件地址，用于证书签名的也是电子邮件地址。PGP 的初衷是抵抗政府的入侵，因此没有使用 CA。相反，每个个体能够决定信任谁，以及信任他们的程度。在该模型中，信任是一个程度问题。另外，公钥证书可以包含一个信任级别来指明签发者对证书中公钥绑定的信任程度。因此，一个用户可能在信任一个密钥绑定前不得不获得多个证明该绑定的证书。

例如，假设你有一个由 Alice 提供的针对 Bob 的证书，你可能给该证书指定一个中等信任级别。但是，如果你还有其他由 C 和 D 提供的针对 Bob 的证书，而且他们都是中等可信的，那么就可以大大增加你对该公钥是 Bob 的有效公钥的信心。简而言之，PGP 认为建立信任的问题完全是个人的事情，为用户提供原始资料，并让他们做出自己的决定，而不是假设他们都自愿信任 CA 的单一层次结构。引用 PGP 的开发者 Phil Zimmerman 的

话来说，“PGP 是为喜欢自己打包降落伞的人设计的。”

PGP 在网络界已经相当流行，PGP 密钥签名聚会成为 IETF 会议的一个特点。在这样的聚会上，个人可以：

- 从他知道身份的其他人那里收集公钥。
- 把自己的公钥提供给其他人。
- 获得由其他人签署的自己的公钥，从而收集对不断扩大的人群更有说服力的证书。
- 签署另外一些人的公钥，从而帮助他们建立证书集合，用户分发他们的公钥。
- 从自己足够信任的签署密钥的其他人那里收集证书。

这样，随着时间的推移，一个用户会收集一套具有不同信任程度的证书。

3. 撤销证书

由证书引发的一个问题是如何撤销或取消一个证书。这个问题为什么重要呢？假设你怀疑某人已发现了你的私钥。而在世界上可能有任意多个证书宣称你是与那个私钥对应的公钥的拥有者。这样，发现你的私钥的人拥有了假冒你所需的一切东西：有效的证书和你的私钥。要解决这个问题，最好能取消旧的被泄露的密钥与你的身份绑定，使得假冒者不能再说服其他人相信他就是你。

对这个问题的基本解决方案十分简单。每个认证机构可以发行一张证书撤销列表（Certificate Revocation List，CRL），它是一个经过数字签名的被撤销的证书的列表。CRL 定期更新并公开。因为它是经过数字签名的，因此可以张贴在网站上。这样，当 Alice 接收到 Bob 的证书并希望验证该证书时，Alice 会首先查阅 CA 最近公布的 CRL。只要证书未被撤销，它就有效。注意，如果所有证书都有无限的有效期，CRL 就会变得越来越长，因为担心可能要使用被撤销的证书的某些拷贝，你永远不会从 CRL 中去掉一个证书。因此，通常会在颁发证书时附上有效日期。这样，我们就能限制被撤销的证书需要保留在 CRL 上的时间。只要它过了最初设定的有效日期，就可以从 CRL 中删除它。

8.2.2 对称密钥预分发

如果 Alice 想用秘密密钥密码与 Bob 通信，那么她就不能只选择一个密钥并发送给他，因为如果事先没有密钥，他们就不能加密该秘密密钥以便保密，而且也不能相互认证。就像公钥一样，需要某种预分发机制。对称密钥的预分发比公钥的预分发困难，有两个主要原因：

- 虽然每个实体只需要一个公钥就能够满足认证和保密的需要，但想要通信的每一对实体必须有一个对称密钥。如果有 N 个实体，那么需要 $N(N-1)/2$ 个密钥。
- 与公钥不同，秘密密钥必须要保密。

总而言之，需要分发更多的密钥，而且你不能使用任何人都可以读的证书。

最常见的解决方法是使用密钥分发中心（Key Distribution Center，KDC）。KDC 是一个与每个实体都共享一个秘密密钥的可信实体。这就将密钥数量降低到 $N-1$ 个，因此更好管理，对一些应用来说完全可以在带外建立这些密钥。当 Alice 希望与 Bob 通信时，其通信内容并不经过 KDC。KDC 只是参与认证 Alice 和 Bob 的协议——使用 KDC 已经与他们分别共享的密钥——并为他们生成新的会话密钥。然后 Alice 和 Bob 直接使用他们的会话密钥通信。Kerberos（见 8.3.3 节）是一个基于该方法的应用广泛的系统。

8.3 认证协议

我们在8.1节描述了如何加密消息和创建认证码，在8.2节介绍了如何预分发必要的密钥。要保护协议安全，看起来好像只要在每条消息后追加认证码并在需要保密的时候加密消息就可以了。

实际上并没有这么简单，有两个原因。首先，存在重放攻击（replay attack）的问题：攻击者重新传送一条以前发送过的消息。例如，如果该消息是你在网站上下的订单，那么重放的消息对网站来说好像是你又订购了一份同样的东西。虽然这条消息不是最初的那条消息，但认证码依然有效；毕竟该消息是你创建的，并且没有被篡改。这种攻击的一种变形叫作隐藏重放攻击（suppress-replay attack），攻击者可能只是推迟你的消息（通过劫持消息并在以后重放该消息来实现），从而使该消息在不合适的时候被接收到。例如，攻击者可以将你的股票订单从一个合适的时间推迟到一个你并不想买入的时间。虽然这条消息从某种意义上说还是原始的，但它不具有时效性。原始性和时效性可以看作完整性的不同方面。要保障这两个方面，在大部分情况下需要一个重要的反复的协议。

另一个我们还没有解决的问题是怎样建立会话密钥。会话密钥是在通信过程中生成的对称密码密钥，只用于一次会话，参见8.2节。这同样也包含一个重要的协议。

这两个问题的共同点是认证。如果一条消息不是原始的且及时的，那么从实用的角度出发，我们希望将它看作一条不真实的消息，并不来源于它所声称的实体。另外，很显然，当你安排与某人共享一个新会话密钥时，你希望知道你确实与正确的人共享了密钥。通常，认证协议也同时建立会话密钥，这样在协议结束时，Alice和Bob已经认证了对方并有了一个新的对称密钥。如果没有新会话密钥，协议只能在某个时间点认证Alice和Bob，而会话密钥允许他们有效地认证后续的消息。一般情况下，会话密钥建立协议执行认证（一个值得注意的例外是Diffie-Hellman，见8.3.4节）。因此，术语认证协议（authetication protocol）和会话密钥建立协议（session key establishment protocol）几乎是同义的。

在认证协议中有一套确保原始性和时效性的核心技术。在讨论具体协议前，我们先描述这些技术。

8.3.1 原始性和时效性技术

我们已经看到只用认证码并不能使我们检测到非原始的或不及时的消息。一种方法是在消息中包含时间戳。显然，时间戳本身必须是防篡改的，因此认证码必须涵盖它。时间戳的主要缺点是它们需要分布式的时钟同步。因为我们的系统会依赖于同步，因此时钟同步除了常见的挑战外，也需要被保护以抵抗安全威胁。另一个问题是分布式的时钟只能在一定程度上保持同步——一定的误差幅度。因此，时间戳提供的时序完整性的精确度只能与同步的程度相当。

另一种方法是在消息中包含一个当前值（nonce）——只使用一次的随机数。参与者能够通过检查该当前值是否曾使用过来检测重放攻击。不过，这需要记录以前使用的当前值，这会积累大量的数据。一种解决方法是将时间戳和当前值结合起来，只需要当前值在一定的时间范围内保持唯一。这样就能以便于管理的方式保证当前值的唯一性，而且只需要时钟的不精确同步。

解决时间戳和当前值缺点的另一种方案是在挑战-响应（challenge-response）协议中使用时间戳和当前值。假设我们使用时间戳。在挑战-响应协议中，Alice 向 Bob 发送一个时间戳，让 Bob 在响应消息中加密该时间戳（如果他们共享一个对称密钥）或者对该时间戳进行数字签名（如果 Bob 有一个公钥，如图 8-7 所示）。被加密的时间戳就像一个同时证明了时效性的认证码。Alice 能够很容易地检查来自 Bob 的响应消息中时间戳的时效性，因为该时间戳来自 Alice 自己的时钟——不需要分布式的时钟同步。再假设协议使用当前值。那么 Alice 只需要跟踪响应消息正在显示的当前值以及很长时间内没有被显示的当前值，任何带有无法识别的当前值的响应都是伪造的。

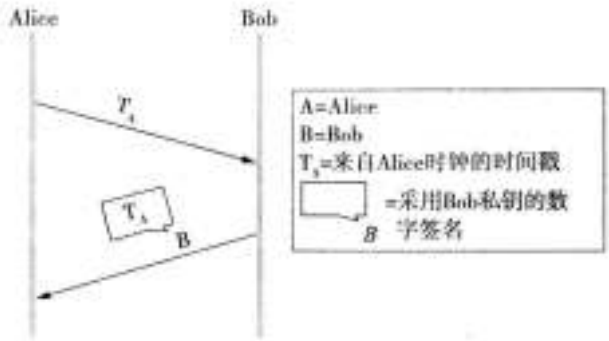


图 8-7 挑战-响应协议

挑战-响应的优点是它将时效性和认证结合起来，因为只有 Bob 知道用于加密这个从未出现过的时间戳或当前值的密钥（如果是对称密钥密码，Alice 也知道这个密钥），否则协议就会显得异常复杂。在下面描述的大部分认证协议中都使用时间戳或当前值。

8.3.2 公钥认证协议

在讲述公钥认证协议前，我们假设已经通过某些诸如 PKI 的方法（见 8.2.1 节）预分发了 Alice 和 Bob 的公钥。这也包括 Alice 在她发送给 Bob 的第一条消息中包含自己证书的情况以及 Bob 在收到 Alice 发送的第一条消息后搜索 Alice 证书的情况。

第一个协议（见图 8-8）依赖于 Alice 和 Bob 的时钟同步。Alice 向 Bob 发送一条包含明文时间戳及其身份和数字签名的消息。Bob 用数字签名来认证消息，用时间戳来验证消息是否是新消息。Bob 回送一条消息，包含明文时间戳和他的身份，以及一个用 Alice 的公钥加密的新会话密钥（为了保密），并且所有内容都被数字签名了。Alice 能够验证消息的真实性和时效性，以便确定这个新会话密钥是否可信。为了解决时钟同步不精确的问题，可以将时间戳与当前值结合使用。

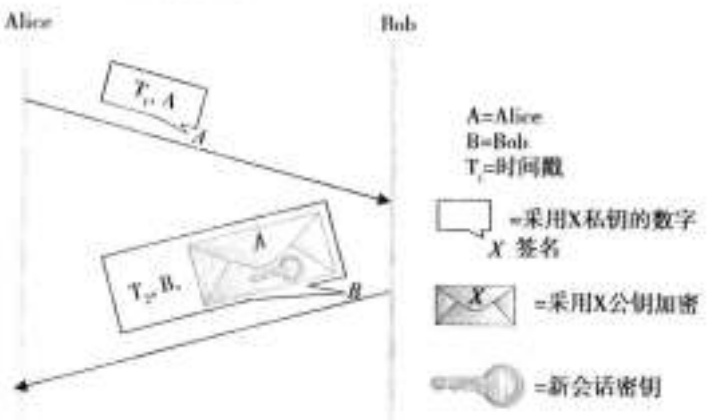


图 8-8 依赖于同步的公钥认证协议

第二个协议（见图 8-9）与前一个类似，但不依赖于时钟同步。在这个协议中，Alice 还是向 Bob 发送一条包含时间戳及其身份的经数字签名的消息。因为他们的时钟并不同步，因此 Bob 不能确定该消息是否是新消息。Bob 回送一条经数字签名的包含 Alice 的原始时间戳、他自己的时间戳以及他的身份的消息。Alice 可以通过将她的原始时间戳与她的当前时间相比较来验证消息的时效性。然后她再向 Bob 发送一条经数字签名的包含 Bob 的原始时间戳以及用 Bob 的公钥加密的新会话密钥的消息。Bob 能够验证消息的时效性，因为该时间戳来自他的时钟，以便确定这个新的会话密钥是否可信。时间戳实际上充当了一种便利的当前值，并且这个协议的确可以使用当前值。

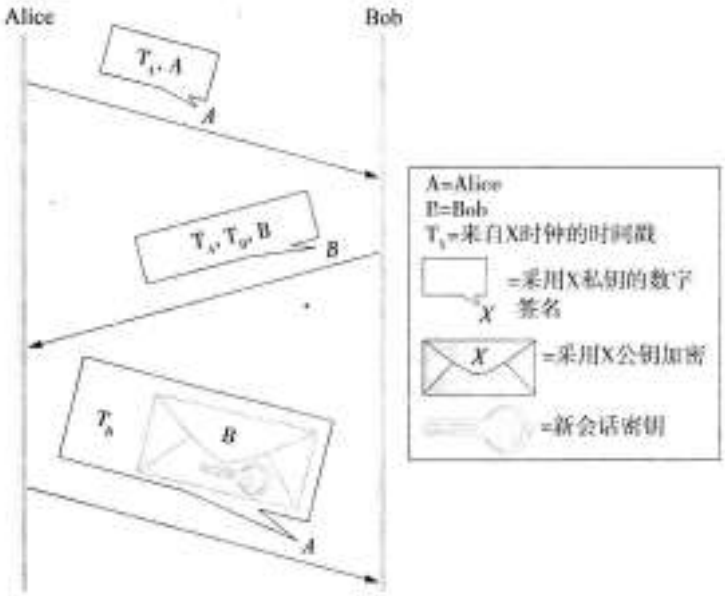


图 8-9 不依赖于同步的公钥认证协议。Alice 只将时间戳与她自己的时钟比对，Bob 也只做类似的比对

8.3.3 对称密钥认证协议

正如 8.2.2 节所说的那样，只有在比较小的系统中为每一对实体预分发密钥才是可行的。在本书中，我们重点讨论大系统，其中每个实体都有自己的与 KDC 共享的主密钥（master key）。在这种情况下，基于对称密钥的认证协议包含三个实体：Alice、Bob 和 KDC。该协议最终的输出是 Alice 和 Bob 之间共享的会话密钥，他们可以使用该密钥直接通信，而不需要与 KDC 交互。

图 8-10 描述了 Needham-Schroeder 认证协议。注意，KDC 实际上并不认证 Alice 的第一条消息，而且根本不与 Bob 通信。KDC 只是用它所知道的 Alice 和 Bob 的主密钥来构造一条对于除 Alice 以外的人都没有用的响应消息（只有 Alice 可以解密该消息），消息中包含了 Alice 和 Bob 用于执行认证协议剩余步骤所需要的资料。

前两条消息中的当前值用于让 Alice 确认 KDC 的响应消息是新的。第二条和第三条消息包含新的会话密钥以及 Alice 的身份，并且都用 Bob 的主密钥加密。这是一种对称密钥版本的公钥证书，它实际上是一种由 KDC 签名的声明（因为 KDC 是除 Bob 以外唯一知道 Bob 主密钥的实体），该声明说明了所包含的会话密钥属于 Alice 和 Bob。虽然最后两条消

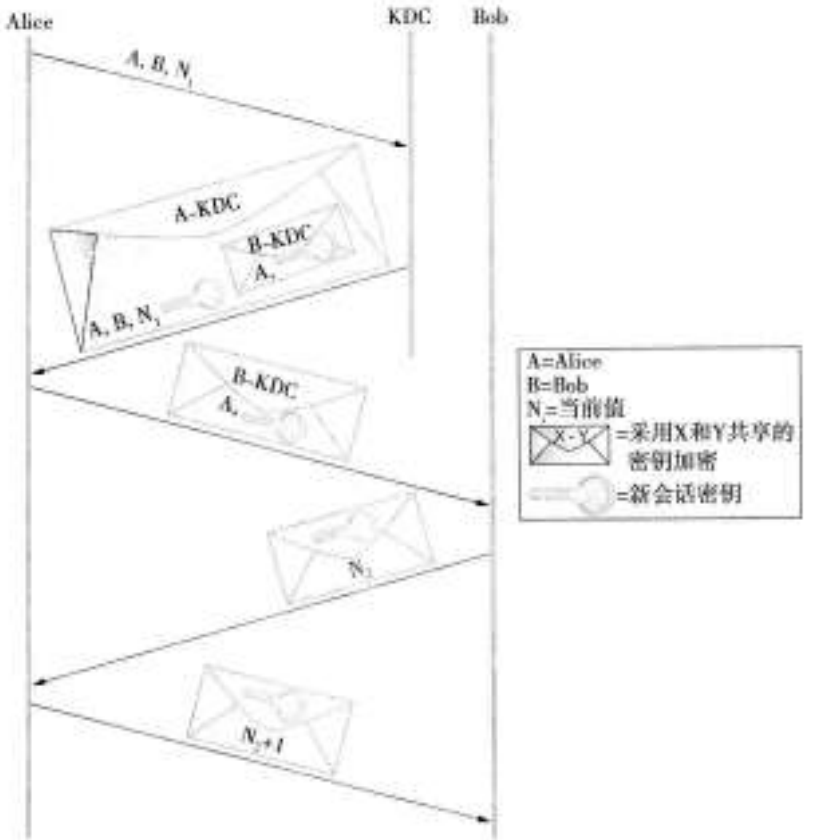


图 8-10 Needham-Schroeder 认证协议

息中的当前值用于让 Bob 确认第三条消息是真的，但该推理有一个缺陷——见习题 4。

Kerberos

Kerberos 是一个基于 Needham-Schroeder 协议的认证系统，专门用于客户端-服务器环境中。Kerberos 最初是由 MIT 开发的，目前是一个 IETF 标准，既有开源产品，又有商业产品。在这里，我们关注 Kerberos 的一些有趣的创新。

Kerberos 的客户一般通过口令来认证自己。Alice 与 KDC 共享的主密钥是由她的口令推算出的——如果你知道口令，你就能计算出密钥。Kerberos 假设任何人都能够通过物理介质访问任何客户端，因此不但要在网络中，而且在 Alice 登录的任何计算机上，都要尽量减小 Alice 的口令被泄露的可能性。Kerberos 利用 Needham-Schroeder 来实现这一点。在 Needham-Schroeder 中，Alice 唯一一次使用口令是解密来自 KDC 的响应。Kerberos 的客户端软件一直等待 KDC 响应的到来，然后提示 Alice 输入她的口令，计算主密钥，解密 KDC 的响应，并删除所有有关口令和主密钥的信息以便将泄漏口令的可能性降到最低。还要注意，用户意识到正在使用 Kerberos 的唯一标记是当用户被提示输入口令时。

在 Needham-Schroeder 中，KDC 发给 Alice 的响应有两个作用：为她提供了一种证明其身份的方法（只有 Alice 能解密该响应），为她提供了一种能够展现给 Bob 的对称密钥证书或“票据”——用 Bob 的主密钥加密的会话密钥和 Alice 身份。在 Kerberos 中，这两个功能以及 KDC 本身都被一分为二（见图 8-11）。称为认证服务器（Authentication Server，AS）的可信服务器完成 KDC 的第一个功能，即为 Alice 提供一种证明身份的方法

(不是向 Bob 证明，而是向称为票据授予服务器（Ticket Granting Server，TGS）的第二个可信服务器证明)。TGS 完成 KDC 的第二个功能，向 Alice 回送一个提交给 Bob 的票据。该方案的优点是，如果 Alice 需要与多个服务器通信，而不仅是 Bob，那么她能够从 TGS 为每一个服务器获得一个票据，而不需要访问 AS。

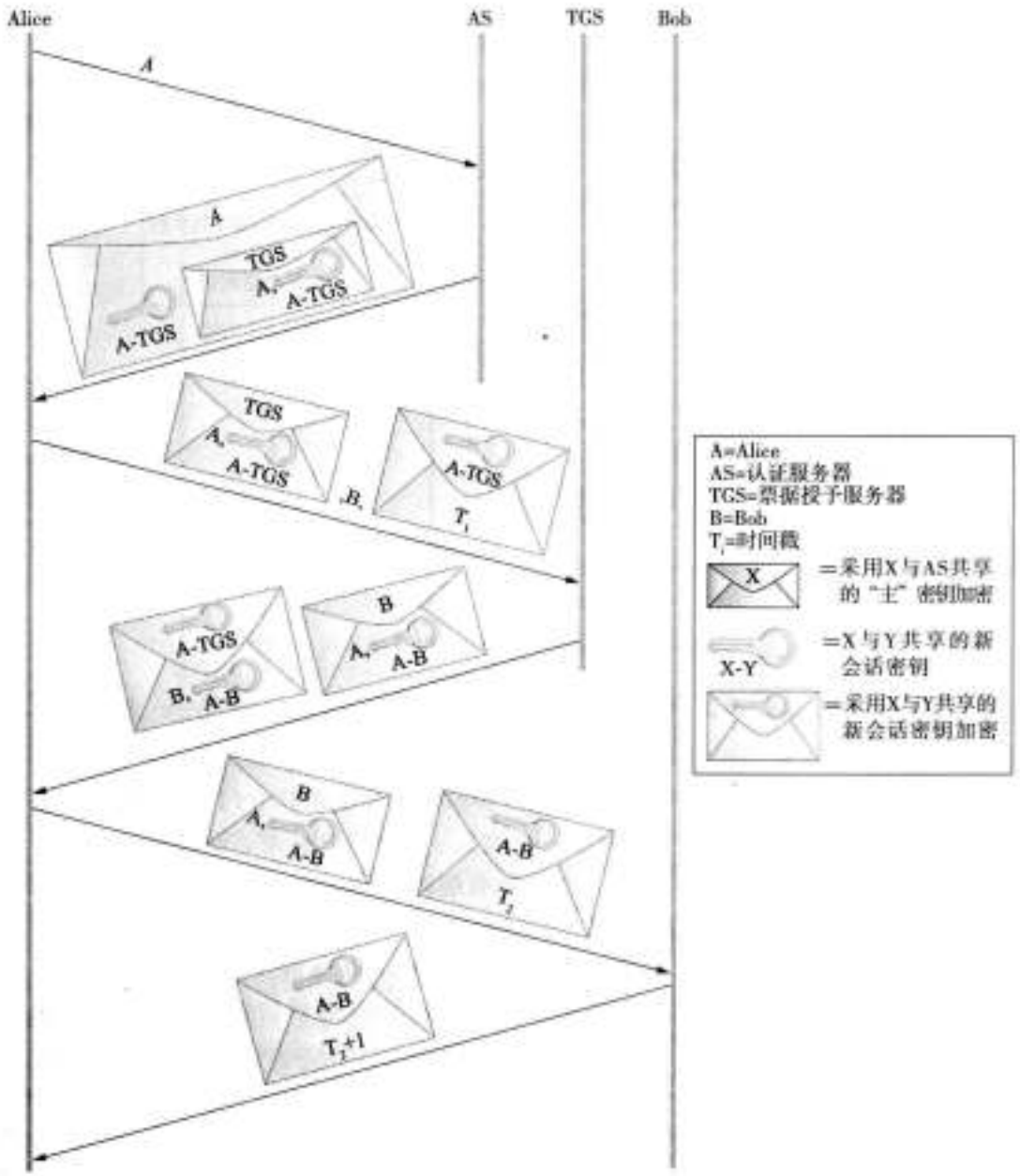


图 8-11 Kerberos 认证

在使用 Kerberos 的客户端-服务器应用领域中，假设一定程度的时钟同步是合理的。这允许 Kerberos 使用时间戳和生命期而不是 Needham-Schroeder 的当前值，因此可以避免习题 4 中讨论的 Needham-Schroeder 的安全弱点。Kerberos 支持的密码算法包括散列函数 SHA-1 和 MD5，对称密钥密码 AES、3DES 和 DES。

8.3.4 Diffie-Hellman 密钥协商

Diffie-Hellman 密钥协商协议用于建立会话密钥，而不需要预分发密钥。Alice 与 Bob 之间交换的消息能够被窃听，但是窃听者不知道 Alice 和 Bob 最终计算出的会话密钥。另一方面，Diffie-Hellman 并不认证参与者。因为在不能确认与谁通信的情况下进行安全通信几乎没有意义的，因此通常以某种方式对 Diffie-Hellman 进行补充来提供认证。

这个协议有两个参数 p 和 g，两个都是公共参数而且可以被系统中的所有用户使用。参数 p 必须是一个素数。整数 mod p（modulo p 的缩写）后的结果在 $0 \sim p-1$ 之间，因为 $x \bmod p$ 是 x 被 p 整除后的余数，这就形成了数学家所说的乘法群（group）。参数 g（一般称为生成元）必须是 p 的本原根（primitive root）：对 $1 \sim p-1$ 之间的每个数 n，都存一个值 k，使得 $n=g^k \bmod p$。例如，如果 p 是素数 5（实际系统中使用的素数大得多），那么我们可以选择 2 作为生成元 g，因为：

$$1 = 2^0 \bmod p$$
$$2 = 2^1 \bmod p$$
$$3 = 2^3 \bmod p$$
$$4 = 2^2 \bmod p$$

假设 Alice 和 Bob 想协商一个共享对称密钥。Alice 和 Bob 以及所有其他人都已经知道了 p 和 q 的值。Alice 生成一个随机私有值 a，Bob 生成一个随机私有值 b。a 和 b 都是从 $\{1, \cdots, p-1\}$ 这组整数中选取出来的。Alice 和 Bob 推算出相应的公开值——他们将要发送给对方的未加密的值，如下所示。Alice 的公开值是

$$g^a \bmod p$$

Bob 的公开值是

$$g^b \bmod p$$

然后交换他们的公开值。最后，Alice 计算

$$g^{ab} \bmod p = (g^b \bmod p)^a \bmod p$$

Bob 计算

$$g^{ba} \bmod p = (g^a \bmod p)^b \bmod p$$

现在 Alice 和 Bob 将 $g^{ab} \bmod p$（与 $g^{ba} \bmod p$ 相等）作为他们共享的对称密钥。

任何窃听者都能知道 p、g 以及两个公开值 $g^a \bmod p$ 和 $g^b \bmod p$。如果窃听者能够确定 a 或 b，她才能够很容易地计算出密钥。然而，对于足够大的 p、a 和 b，从这些信息中得出 a 或 b 在计算上是不可行的，这称为离散对数问题（discrete logarithm problem）。

另一方面，Diffie-Hellman 还存在缺少认证的问题。一种可以利用该问题的攻击是中间人攻击（man-in-the-middle attack）。假设 Mallory 是一个能够截获消息的攻击者。Mallory 已经知道了 p 和 g，因为它们都是公开的。她还产生了两个随机私有值 c 和 d，分别用于与 Alice 和 Bob 通信。当 Alice 和 Bob 向对方发送了他们的公开值时，Mallory 截获这些值并发送她自己的

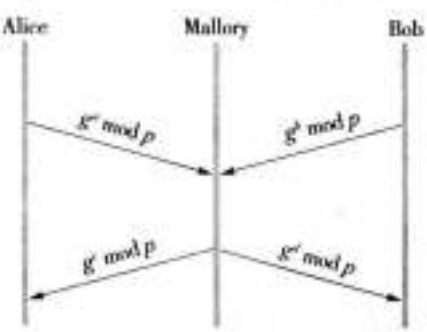


图 8-12 中间人攻击

公开值，如图 8-12 所示。结果是 Alice 和 Bob 在不知情的情况下分别与 Mallory 而不是与对方共享了一个密钥。

Diffie-Hellman 的一种称为修补的 Diffie-Hellman（fixed Diffie-Hellman）的变形提供了对一方或两方参与者的认证。它依赖于与公钥证书类似的证书，但它用于证明实体的 Diffie—Hellman 公共参数。例如，这种证书可能会声明 Alice 的 Diffie-Hellman 参数是 p、g 和 $g^a \bmod p$（注意，a 的值仍然只有 Alice 知道）。这种证书能够让 Bob 确信执行 Diffie-Hellman 协议的另一方是 Alice——其他参与者不能计算出秘密密钥，因为她不知道 a。如果两个参与者都有 Diffie-Hellman 参数证书，他们便可以互相认证。如果只有一个参与者有证书，那么只有这个参与者能够被认证。这在一些情况下是有用的，例如，当一个参与者是一台 Web 服务器，另一个参与者是任意的客户，该客户能够在向 Web 服务器发送信用卡卡号前认证该服务器并建立一个用于保密的会话密钥。

8.4 系统实例

截至目前，我们已看到了构造某种安全性所需要的组件。这些组件包括密码算法、密钥预分发机制和认证协议。这一节我们分析一些使用这些组件的完整系统。

可以根据这些系统工作的协议层粗略地进行分类。工作在应用层的系统包括良好隐私（Pretty Good Privacy，PGP），它提供电子邮件安全和安全外壳（Secure Shell，SSH）——一个安全的远程登录工具，在传输层，有 IETF 的传输层安全（Transport Layer Security，TLS）标准和它所派生于的较早的安全套接字层（Secure Socket Layer，SSL）协议。IPsec（IP 安全）协议，顾名思义，工作在 IP（网络）层。802.11i 为无线网络的链路层提供了安全性。本节将阐述每个方法的主要特性。

你可能会疑惑为什么必须在如此多不同的层提供安全性。一个原因是不同的威胁需要不同的防御措施，并且这通常会对应于保护不同的协议层。例如，如果你主要考虑在相邻建筑中的一个人窥探你的笔记本电脑与 802.11 接入点之间的通信，那么，你可能需要链路层安全。然而，如果你希望确信你连接到了银行的网站，并且阻止因特网服务提供商的某个好奇的雇员读取你发送给银行的所有数据，那么提供通信安全的正确位置应该是你的计算机与银行服务器之间整个通信信道的某层，如传输层。情况常常如此，不存在适用于所有情况的解决方案。

下面所描述的安全协议都能够改变所使用的密码算法。将安全系统设计为与算法独立的思想是非常好的，因为你不知道什么时候你最喜欢的密码算法会被证明对你的目标来讲不够安全。如果你能在不需要改变协议规范或协议实现的情况下很快地切换到新算法，这将是非常好的一件事。注意，与在不改变算法的情况下改变密钥类似，如果你的一个密码算法被发现有缺陷，而你的整个安全体系结构不需要马上重新设计，这是极好的。

8.4.1 良好隐私（PGP）

良好隐私（Pretty Good Privacy，PGP）是一种广泛用于电子邮件安全的方法。它提供了认证、机密性、数据完整性和不可否认性。PGP 最早是由 Phil Zimmerman 发明的，并且已经发展成为 IETF 的标准，称为 OpenPGP。正如我们在 8.2 节所看到的，PGP 适用于使用“信任网络”模型来分发密钥而不是用树型的分层结构。

PGP 的机密性和接收方认证依赖于电子邮件的接收方拥有一个发送者知道的公钥。为

了提供发送方认证和不可否认性，发送方必须拥有一个接收者知道的公钥。这些公钥是通过 8.2.1 节中描述的证书和信任网络 PKI 进行预分发的。这些证书可以进一步指明支持哪些密码算法或者密钥拥有者推荐哪些密码算法。

思考下面的例子，PGP 被用于提供发送方认证和机密性。假设 Alice 有一条消息通过电子邮件传递给 Bob。Alice 的 PGP 应用程序执行图 8-13 所示的步骤。首先，Alice 对消息进行数字签名，MD5、SHA-1 和 SHA-2 协议族都是可以用于数字签名的散列函数。然后，她的 PGP 应用程序为仅有的这条消息产生一个新的会话密钥，所支持的对称密钥密码包括 AES 和 3DES。数字签名后的消息通过该会话密钥被加密，会话密钥本身被 Bob 的公钥加密后追加到消息后。Alice 的 PGP 应用程序会提示她以前赋给 Bob 的公钥的信任程度，这是根据她所拥有的 Bob 的证书的数量以及对每个证书的签名者的信任度来确定的。最后，对消息进行 base64 编码（如 9.1.1 节所述），转换成一种 ASCII 兼容的表示形式。这不是为了安全，而是因为电子邮件消息必须以 ASCII 形式发送。通过电子邮件接收到 PGP 消息后，Bob 的 PGP 应用程序按逆序执行上述过程来得到原始的明文消息，并确认 Alice 的数字签名——同时提示 Bob 他对 Alice 的公钥的信任程度。

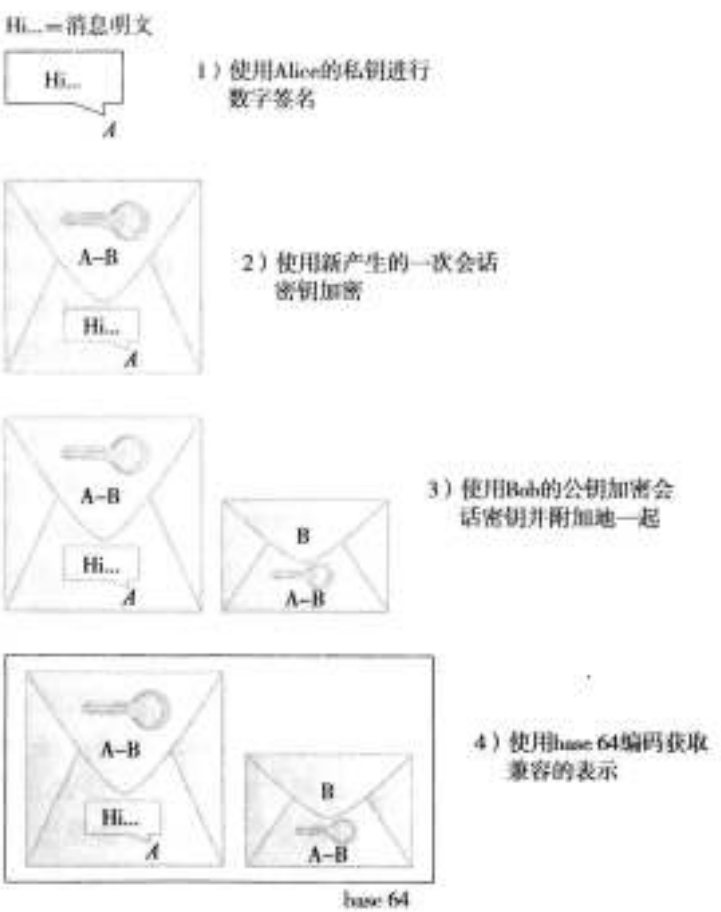


图 8-13 PGP 准备由 Alice 通过电子邮件发送到 Bob 的消息的步骤

电子邮件有一个不同寻常的特性，它允许 PGP 在这个单条消息数据传输协议中嵌入适当的认证协议，从而避免提前进行任何消息交换（避免前面在 8.3 节中描述的一些复杂步骤）。Alice 的数字签名足以认证她。虽然无法证明消息是及时的，但传统的电子邮件也

不是及时的。另外，也无法证明消息是原始的，但 Bob 作为一个电子邮件用户，通常能够从电子邮件副本中恢复信息（这在正常操作情况下也不是不可能的）。Alice 可以确定只有 Bob 能够读取消息，因为会话密钥是用他的公钥加密的。虽然该协议没有向 Alice 证明 Bob 确实存在并收到了电子邮件，但从 Bob 回送给 Alice 的经过认证的电子邮件能够提供证明。

上述讨论给出了一个很好的示例，说明了为什么应用层安全机制很有用。只有全面了解应用的工作原理，你才能做出正确的选择，即防御哪些攻击（如伪造的电子邮件）以及忽略哪些攻击（如被延迟的或被重放的电子邮件）。

8.4.2 安全外壳（SSH）[⊖]

安全外壳（Secure Shell，SSH）协议用于提供一种远程登录服务，其目的是取代早期在因特网上使用的安全性较差的 Telnet 和 rlogin 程序。（SSH 还能用于远程执行命令和传输文件，分别与 Unix 的 rsh 命令和 rcp 命令类似，但我们重点要关注的是 SSH 怎样支持远程登录。）SSH 最常用的功能是提供强客户端/服务器认证/消息完整性——其中 SSH 的客户端运行在用户的台式机上，而 SSH 的服务器端运行在用户想要登录的某台远程主机上——但 SSH 也支持机密性。Telnet 和 rlogin 则不提供任何这样的功能。注意，“SSH”既指 SSH 协议，也指使用 SSH 的应用，你需要从上下文中区分出指的是什么。

为了更好地理解 SSH 对现今因特网的重要性，考虑几个应用 SSH 的场景。例如，远程通勤人员一般付款给那些提供高速电缆调制解调器或 DSL 服务的 ISP，他们通过这些 ISP 或其他 ISP 组成的链来连接到雇主运行的计算机。这就意味着在登录时，他们的密码以及他们收发的数据很可能穿越任意多个不可信网络。SSH 提供一种方法来加密在这些连接上发送的数据以增加他们登录时使用的认证机制的强度。SSH 的一种类似的用法是远程登录到路由器，可能是更改配置或读取日志文件。显然，网络管理员希望确保他能够安全登录到路由器，并且未授权实体既无法登录，也不能截取发送给路由器的命令或返回给管理员的输出信息。

SSH 的最新版本（版本 2）由三个协议组成：

- SSH-TRANS：传输层协议。
- SSH-AUTH：认证协议。
- SSH-CONN：连接协议。

我们关注前两个协议，它们与远程登录有关。我们在本节的最后简要讨论 SSH-CONN 的用途。

SSH-TRANS 在客户端和服务器之间提供了一条加密信道。这条信道运行在一个 TCP 连接上。当一个用户使用 SSH 登录一台远程主机时，第一步就是要在两台主机间建立一条 SSH-TRANS 信道。两台计算机建立这条安全信道的方法是先让客户使用 RSA 认证服务器。认证完成后，客户和服务器就建立一个会话密钥，并用这个密钥加密信道上发送的所有数据。这种高层次描述忽略了几个细节，其中包括这样一个事实，SSH-TRANS 协议包含双方对将要使用的加密算法的协商。例如，一般会选择 AES。SSH-TRANS 还会对信道上交换的所有数据进行消息完整性检查。

⊖ 参见“实验十三”。

我们不能忽略的一个问题是客户端如何获得认证服务器所需的服务器的公钥。听起来可能有些奇怪，服务器在连接时告诉客户端它的公钥。当客户首次连接到一台特定主机时，SSH 应用程序告诫用户他从未同这台主机对过话并询问用户是否要继续。尽管有风险，用户还是经常回答“是”，因为 SSH 应用不能有效认证服务器。于是 SSH 应用就记住服务器的公钥，并在下一次用户连接到同一机器时把保存的密钥与服务器回应的密钥进行比较。如果它们是相同的，服务器就通过 SSH 的认证。但如果不同，SSH 会再次警告用户有错误，然后给用户中止连接的机会。作为替代方法，谨慎的用户可以通过带外机制来获得服务器的公钥，并保存到客户端，这样就不用冒“首次”连接的危险。

一旦 SSH-TRANS 信道存在，下一步就是用户实际登录到目标主机上，更确切地说，就是让服务器对用户身份进行认证。SSH 允许通过三种不同机制来完成这件事。第一种机制，因为两台计算机是在一条安全的信道上通信，所以用户只要把口令发送给服务器即可。对于 Telnet 来说，这样做是不安全的，因为口令是用明文发送的，但在使用 SSH 的情况下，口令在 SSH-TRANS 信道中是加密的。第二种机制使用公钥加密，这种机制要求用户将其公钥事先放在服务器上。第三种机制称为基于主机的认证（host-based authentication），基本思想是任何声称来自一组可信主机的用户将自动被认为是该服务器上的同样的用户。基于主机的认证要求客户主机（host）在首次连接时向服务器认证自己。标准的 SSH-TRANS 默认仅认证服务器。

通过这段讨论，你主要应该明白 SSH 是我们在本章所见的协议和算法的一个比较直接的应用。然而，用户需要建立和管理多个密钥，这有时让 SSH 显得很难懂，而且实际使用的接口依赖于操作系统。例如，在大多数 Unix 机上运行的 OpenSSH 包都支持 ssh-keygen 命令，用于创建公钥/私钥对。然后这些密钥被存储在用户主目录下的 .ssh 目录的不同文件中。例如，文件/.ssh/known_hosts 记录用户已登录的所有主机的密钥，文件/.ssh/ authorized _ keys 包含用户登录本机时认证他所需的公钥（即它们在服务端被使用），文件/.ssh/identity 包含认证远程机器上的用户时需要的私钥（即它们在客户端被使用）。

最后，SSH 已被证明是一个保护远程登录安全的有用系统，它已经被扩展以支持其他应用，如发送和接收电子邮件。其思想是在一条安全的“SSH 隧道”上运行这些应用程序。这种功能称为端口转发（port forwarding），它使用的是 SSH-CONN 协议。图 8-14 展示了这种思想，我们看到主机 A 上的客户程序通过使用一条 SSH 连接来转发通信量，间接地与主机 B 上的服务程序通信。这种机制称为端口转发是因为当消息到达服务器上的知名 SSH 端口时，SSH 首先把内容解密，然后把数据转发到服务器实际监听的端口上。

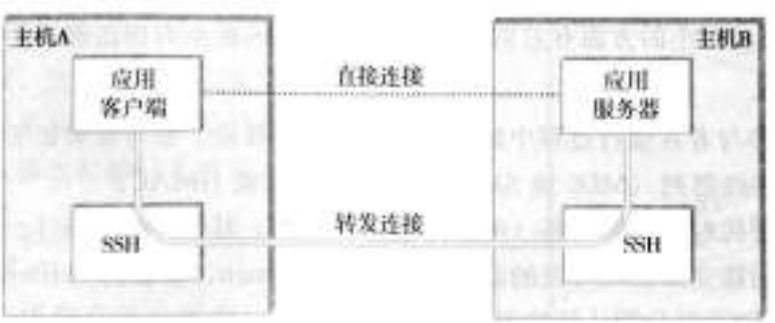


图 8-14 用 SSH 端口转发保护其他基于 TCP 的应用的安全

这仅是另外一种隧道，即3.2.9节中介绍的类型，在本例中提供了机密性和认证。以这种方法使用隧道可以提供一种虚拟专用网络（Virtual Private Network，VPN）。

8.4.3 传输层安全（TLS、SSL、HTTPS）

为了了解IETF正在开发的传输层安全（Transport Layer Security，TLS）标准以及TLS的基础安全套接字层（Secure Socket Layer，SSL）的设计目标和需求，考虑它要解决的主要问题之一是有帮助的。当WWW流行起来而且商业企业开始对它感兴趣时，很显然，对于Web上的交易，某些安全等级将是必不可少的。这种情况的典型例子是通过信用卡购物。当你把信用卡信息发送到Web上的一台计算机时需要考虑几个问题。首先，你可能会担心信息在传输中被截获，并被他人用于非法购物。你也许还担心交易的有关细节被修改，如改变购买数量。而且你一定想知道信用卡信息到达的计算机是属于商家的计算机而不是其他什么人的计算机。这样，我们立即看到Web交易对机密性、完整性和认证的需求。对这个问题的第一个广泛使用的解决方法是SSL，最初是由Netscape开发的，后来成为IETF的TLS标准的基础。

SSL和TLS的设计者认识到这些问题并非Web交易（即那些使用HTTP的应用）所特有的，因而要在应用协议（例如HTTP）和传输协议（例如TCP）之间建立一个通用的协议。将这个协议称为“传输层安全”的原因在于，从应用的观点看，这个协议层看起来就像一个普通的传输协议，只是它是安全的。也就是说，发送方可以打开连接并发送要传输的字节，而安全传输层就会把它们传送给接收方，并保证必需的机密性、完整性和认证。在TCP上通过运行安全传输层，TCP的所有一般特性（可靠性、流量控制、拥塞控制等）也会被提供给应用程序。图8-15描述了协议层的布局。



图8-15 应用层与TCP层之间插入的安全传输层

当以这种方式使用HTTP时，它被称为是HTTPS（安全的HTTP）。事实上，HTTP本身无变化。它简单地把数据传递到SSL/TLS层或从SSL/TLS层接收数据，而不是在TCP层。为方便起见，为HTTPS分配了默认的TCP端口443。就是说，如果你试图连接到一台服务器的TCP端口443，可能会发现你正与SSL/TLS协议会话，只要能顺利进行认证和解密，它就会将数据传给HTTP。虽然存在独立的SSL/TLS实现，但更常见的实现方法是与需要它的应用绑定，主要是Web浏览器。

在下面的关于传输层安全的讨论中，我们关注TLS。虽然SSL和TLS非常不幸地不能互操作，但只在很小的方面有差别，因此几乎对TLS的所有描述都适用于SSL。

1. 握手协议

一对TLS参与者在运行过程中协商要使用的密码算法。参与者对如下选择进行协商：

- 数据完整性散列（MD5或SHA-1等），用于实现HMAC。
- 用于保密的对称密钥密码（可能的选择有DES、3DES和AES.）。
- 会话密钥建立方法（可能的选择有Diffie-Hellman、修正的Diffie-Hellman和使用RSA或DSS的公钥认证协议。）

有趣的是，握手协议还可以协商使用压缩算法，不是因为它能提供安全性方面的好

处，而是因为在你进行其他协商时很容易实现压缩算法协商，并且你已经决定对数据进行一些开销较大的以字节为单位的操作。

在 TLS 中，保密密码使用两个密钥，每个方向一个，类似地也需要两个初始向量。同样，HMAC 为两个参与者使用不同的密钥。这样，无论选择什么密码和散列函数，一个 TLS 会话都需要六个密钥。TLS 从一个共享的主密钥（master key）中推导出所有密钥。主密钥是一个 384 位（48 字节）的值，则是部分地由 TLS 会话密钥建立协议产生的"会话"密钥中推导出来的。

TLS 中协商选项和建立共享主秘密的部分称为握手协议（handshake protocol）。（实际的数据传输是由 TLS 的记录协议（record protocol）完成的）。握手协议本质上是一个会话密钥建立协议，它建立一个主秘密而不是会话密钥。TLS 支持对会话密钥建立方法的选择。这些都需要相应的不同协议。另外，握手协议支持选择两个参与者的相互认证、仅对单个参与者的认证（这是最常见的情况，例如，认证一个 Web 站点而不是用户）或者无认证（匿名 Diffie-Hellman）。这样，握手协议就把几个会话密钥建立协议交织在一个协议中。

图 8-16 显示了握手协议的高层描述。客户首先发送一个它所支持的密码算法组合的列表，以偏好递减的顺序排列。服务器返回它从客户列表中选择的一个密码算法组合。这些消息还包含一个客户当前值（client-nonce）和一个服务器当前值（server-nonce），当前值将被用于以后产生主秘密。



图 8-16 用于建立 TLS 会话的握手协议

到此，协商阶段结束。现在服务器根据所协商的会话密钥建立协议发送其他消息。这可能包括发送一个公钥证书或者一组 Diffie－Hellman 参数。如果服务器要求认证客户，它就会发送一个单独的消息来说明这一点。然后，客户用所协商的密钥交换协议的相应部分作为响应。

现在，客户和服务器都拥有了产生主秘密所必需的信息。它们所交换的会话密钥事实上并不是一个密钥，而是 TLS 中所谓的预主秘密（pre-master secret）。主秘密是由预主秘密、客户当前值和服务器当前值计算得出的（使用一个公开的算法）。然后，客户用从主秘密推导出的密钥发送一条消息，它包含了前面所有握手消息的散列值，服务器会以一条类似的消息作为响应。这使得它们能够检测到所发送的消息与接收到的握手消息之间的任何区别，例如，一个中间人攻击通过修改初始的未加密客户消息来减弱对密码算法的选择。

2. 记录协议

在用握手协议建立的会话中，TLS 的记录协议为下层的传输服务增加了机密性和完整性。从应用层

向下传递的消息：

1）为执行后续步骤将消息分段或合并成适当大小的块。

2）压缩（可选）。

3）使用 HMAC 保护完整性。

4）使用对称密码加密。

5）被传送到传输层（一般是 TCP），进行传输。

记录协议使用一个 HMAC 作为认证码。HMAC 使用参与者协商的任何散列函数（MD5 或 SHA-1 等）。在计算 HMAC 时，客户和服务器使用不同的密钥，这使得破解更加困难。另外，每条记录协议消息被分配一个序列号，计算 HMAC 时将该序列号包括在内，但该序列号并不显式地出现在消息中。这个隐含的序列号能够防止消息重放或重排序。这一点是需要的，因为虽然 TCP 保证了正常情况下没有重复消息，但没有考虑能够截获消息并发送伪造消息的攻击者。另一方面，TCP 的确保传输使得 TLS 可以依赖拥有下一个隐含序列号的合法 TLS 消息实现按序传输。

TLS 协议的另一个对 Web 交易相当有用的有趣特性是恢复会话的功能。为了了解这种功能的动机，理解 HTTP 如何使用 TCP 连接是有帮助的（HTTP 的细节在 9.1.2 节给出）。每个 HTTP 操作，如从服务器获取文本网页或图像，都需要打开一个新的 TCP 连接。获取有许多嵌入图形对象的单个页面会占用许多 TCP 连接。回想一下 5.2 节，数据开始传输之前打开 TCP 连接需要三次握手。一旦 TCP 连接准备接收数据，客户就需要启动 TLS 握手协议，应用程序数据真正发送前会话至少还要两个 RTT（并消耗一些处理资源和网络带宽）。TLS 的恢复功能可缓解这个问题。

会话恢复是对握手的一种优化，可以用在客户和服务器已经建立了一些共享状态的情况下。客户简单地在初始握手消息中包含以前所建立会话的会话 ID。如果服务器发现它仍然有那个会话的状态，并且在会话建立时协商使用恢复选项，那么服务器就可以用一个成功的指示响应客户，然后使用先前协商的算法和参数开始数据传输。如果会话 ID 不与服务器缓存的任何会话状态相匹配，或者如果不允许恢复会话，那么服务器就会返回到标准的握手过程。

8.4.4 IP 安全（IPsec）

也许将安全性集成到因特网中的最具雄心的工作发生在 IP 层。这种体系结构称为 IPsec，对 IPsec 的支持在 IPv4 中是可选的，但在 IPv6 中必须支持。

IPsec 的确是能够提供本章讨论的所有安全服务的框架（与单个协议或系统相反）。IPsec 提供三个自由度等级。第一，它是高度模块化的，允许用户（或更可能是系统管理员）从各种加密算法和专用安全协议中进行选择。第二，IPsec 允许用户从一个大的安全服务菜单中进行选择，包括访问控制、完整性、认证、原始性和机密性。第三，IPsec 可以用来保护“窄小”的流（比如在一对主机间发送的属于特殊 TCP 连接的分组）或“宽大”的流（比如一对路由器之间流过的所有分组）。

从高层看，IPsec 包括两部分。第一部分是实现可用的安全服务的一对协议。它们是提供访问控制、无连接消息完整性、认证和反重发保护的认证首部（Authentication Header，AH）以及同样支持这些服务再加上机密性的封装安全有效载荷（Encapsulating Security Payload，ESP）。因为 AH 很少使用，因此我们在此重点讨论 ESP。第二部分是

对密钥管理的支持，它置身于所谓的因特网安全关联和密钥管理协议（Internet Security Association and Key Management Protocol，ISAKMP）的保护之下。

把这两部分绑定在一起的抽象就是安全关联（Security Association，SA）。SA 是具有一个或多个可用安全特性的单工（单向）连接，保护一对主机间的双向通信，例如对应一个 TCP 连接，需要两个安全关联，每个方向有一个安全关联。虽然 IP 是一个无连接协议，但其安全性依赖于连接状态信息，如密钥和序列号。在创建 SA 时，由接收方的计算机为其指定一个安全参数索引（Security Parameters Index，SPI）。这个 SPI 和目标 IP 地址的组合唯一标识一个 SA。ESP 首部包含 SPI，以便使接收方主机能确定输入分组属于哪个 SA，进而确定对分组用什么算法和密钥。

SA 的建立、协商、修改和删除是通过 ISAKMP 实现的。它定义了交换密钥生成和认证数据的分组格式。这些格式并非很有用，因为它们只提供一个框架，确切的密钥和认证数据的格式取决于所使用的密钥生成技术、密码算法和认证机制。此外，尽管 ISAKMP 建议将因特网密钥交换（Internet Key Exchange，IKE）作为一个可能的协议，并且 IKE 也是实际中使用的协议，但它并不指定特定的密钥交换协议。

ESP 是用于在已建立的 SA 上安全地传输数据的协议。在 IPv4 中，ESP 首部跟在 IP 首部之后；在 IPv6 中，它是一个扩展首部。其格式使用一个首部和一个尾部，如图 8-17 所示。SPI 字段帮助接收方主机识别分组所属的安全关联。SeqNum（序号）字段防止重发攻击。分组的 PayloadData（有效载荷数据）字段包含由 NextHdr（下一个首部）字段描述的数据。如果选择保密，那么就用与 SA 关联的任何算法加密数据。PadLength（填充长度）字段记录给数据加入多少填充。填充有时是必要的，例如，因为加密算法要求明文是某个字节数的倍数，或确保结果密文以 4 字节边界结束。最后，AuthenticationData（认证数据）包含认证码。



图 8-17 IPsec 的 ESP 格式

IPsec 支持隧道模式（tunnel mode），也支持一种更直接的传输模式（transport mode）。每个 SA 工作在其中的一种模式下。在传输模式 SA 中，ESP 的载荷数据被简单地作为像 UDP 或 TCP 一样的高层数据。在这种模式下，IPsec 作为一个中间协议层，很像 SSL/TLS 在 TCP 与高层协议之间的作用。当接收到 ESP 消息后，它的载荷被传送到高层协议。

然而，在隧道模式 SA 中，ESP 的载荷数据本身就是一个 IP 分组，如图 8-18 所示。内部 IP 分组的源地址和目的地址可能与外层的 IP 分组不同。当接收到 ESP 消息时，它的载荷被当作一个正常的 IP 分组转发。使用 ESP 的最常见的方式是在两台路由器（典型情况是防火墙）之间建立一个“IPsec 隧道”。例如，一个公司希望使用因特网连接两个站

点，在一个站点的路由器与另一个站点的路由器之间打开一对隧道模式 SA，就像 3.2.9 节描述的那样。从一个站点发出的 IP 分组在发送路由器上成为发送给另一台路由器的一条 ESP 消息的载荷。接收路由器会拆开 IP 分组的载荷，并转发到真正的目的地。



图 8-18 包含一个嵌套的用隧道模式 ESP 加密的 IP 分组的 IP 分组。注意，内层分组和外层分组具有不同的地址

这些隧道也可以配置为使用 ESP 来提供机密性和认证，从而防止对跨越该虚拟链接的数据的非授权访问，并确保隧道的另一端不会接收到伪造的数据。另外，隧道能够提供流量机密性，因为多个数据流通过单个隧道会使得两个特定端点间传输的数据量信息变得模糊。这种隧道组成的网络能够用于实现一个完整的虚拟专用网络（VPN，见 3.2.9 节）。在 VPN 上通信的主机甚至不需要知道 VPN 的存在。

8.4.5 无线安全（802.11i）

由于介质方面缺少任何物理安全措施，所以无线链路（见 2.7 节）是非常容易受到安全威胁的。虽然 802.11 的便捷使得无线技术得到广泛支持，但缺乏安全性是一个经常遇到的问题。因为无线电波能够穿越大部分墙壁，如果无线接入点缺少安全措施，那么攻击者就能在办公楼外访问公司网络。与此类似，办公楼内带有无线网络适配器的计算机也可以连接到办公楼外，并有可能遭到攻击，而如果该计算机有一个以太网连接，那么公司网络中的其他计算机也可能会遭到攻击。

因此，保护无线链路的安全需要做相当多的工作。令人惊讶的是，用于 802.11 的早期安全技术之一，有线等效隐私（Wired Equivalent Privacy，WEP）被发现有严重缺陷，并且很容易被攻破。

IEEE 802.11i 标准在链路层为 802.11（Wi-Fi）提供了认证、消息完整性和机密性。Wi-Fi 保护接入 2（Wi-Fi Protected Access 2，WPA2）经常作为 802.11i 的同义词，但是，从技术上来讲，WPA2 是认证符合 802.11i 产品的 Wi—Fi 联盟商标。

为了向后兼容，802.11i 包括了第一代安全算法的定义——包括 WEP——现在已经知道这些算法有严重的安全缺陷。我们在这里关注 802.11i 的新的更健壮的算法。

802.11i 认证支持两种模式。对每一种模式，认证成功的最终结果是一个共享的主密钥对。个人模式（personal mode）也称为预共享密钥模式（preshared key mode，PSK），提供了较弱的安全性，但对于像家庭 802.11 网络这样的环境来说更方便且更经济。无线设备和接入点（AP）被预先配置一个共享的密码短语口令（passphrase）——本质上是一个非常长的口令，通过密码学方式从该口令可推导出主密钥。

802.11i 的更强的认证模式基于 IEEE 802.1X 局域网访问控制框架，它使用了一台认证服务器（AS），如图 8-19 所示。AS 和 AP 必须通过安全信道相连，甚至可以在同一台主机上，但它们在逻辑上是分离的。AP 在无线设备与 AS 之间转发认证消息。用于认证的协议是可扩展认证协议（Extensible Authentication Protocol，EAP）。EAP 支持多种认

证方法——智能卡、Kerberos、一次性口令、公钥认证等——以及单方认证和双方认证。因此，与其说 EAP 是一个协议，不如说它是一个认证框架。大量与 EAP 兼容的协议称为 EAP 方法（EAP methods）。例如，EAP-TLS 就是一个基于 TLS 认证的 EAP 方法（见 8.4.3 节）

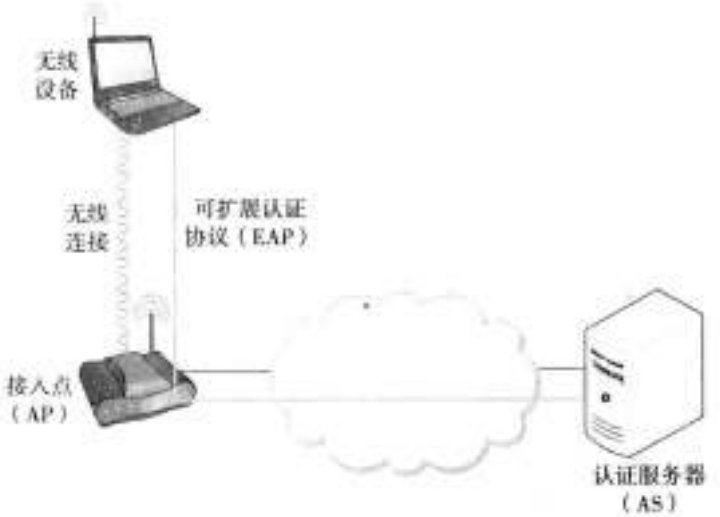


图 8-19 在 802.11i 中使用认证服务器

802.11i 对于将什么 EAP 方法作为认证基础没有做任何限制。然而，它的确要求一种能实现双向（mutual）认证的 EAP 方法，因为我们不但要防止攻击者通过我们的 AP 访问网络，而且希望防止攻击者用一个伪造的恶意 AP 愚弄我们的无线设备。成功认证的最终结果是无线设备与 AS 之间共享的主密钥对，然后 AS 把这个密钥传递给 AP。

较健壮的基于 AS 的模式与较弱的个人模式之间的主要区别之一是前者很容易支持每个客户有一个唯一密钥。这就使得更改能够自认证的客户集（例如，撤销对一个客户的访问）变得更容易，不需要改变存储在每个客户端的秘密信息。

有了主密钥对以后，无线设备和 AP 执行一个称为四次握手的会话密钥建立协议来建立短期密钥对。这个短期密钥对实际上是一个密钥集合，其中包括称为临时密钥（temporal key）的会话密钥。会话密钥被协议 CCMP 使用，为 802.11i 提供数据保密和完整性。

CCMP 代表使用带有消息认证码的密码分组链接（Cipher-Block Chaining with Message Authentication Code，CBC-MAC）协议的 CTR（计数器模式）。CCMP 使用计数器模式的 AES 进行加密来提供机密性。回忆一下计数器模式加密，来自计数器的连续值被集成到连续明文块的加密中（见 8.1.1 节）。

CCMP 使用一个消息认证码（MAC）作为认证码。虽然 CCMP 在机密性加密时不使用 CBC，该 MAC 算法是基于 CBC（见 8.1.1 节）的。事实上，在执行 CBC 时，不传输任何 CBC 加密块，而只是将最后一个 CBC 加密块作为 MAC（实际上只使用它的前 8 个字节）。初始向量是由特殊构造的第一个块充当的，该块包含一个 48 位的分组号（这个分组号也用于机密性加密以及抵抗重放攻击）——一个序列号。然后，MAC 和明文一起被加密以便抵抗生日攻击，生日攻击依赖于找到具有相同认证码的不同消息（见 8.1.4 节）。

8.5 防火墙

虽然本章的大部分内容集中在使用密码学来提供诸如认证和机密性等安全特性，但存在很多无法用密码学方法解决的安全问题。例如，蠕虫和病毒通过利用操作系统及应用程序的错误（有时是利用人类容易受骗的弱点）来传播，没有多少密码学方法能够帮助没有对漏洞打补丁的计算机。因此，需要经常使用其他方法来阻止不同形式的潜在有害流量。防火墙是最常见的方法之一。

防火墙是一种处在它所保护的站点与网络其他部分之间的某个点上的系统，如图 8-20 所示。它通常是作为一台“设备”或路由器的一部分来实现的，虽然“个人防火墙”可以在终端用户计算机上实现。基于防火墙的安全依赖于防火墙是从外部接入站点的唯一连接点，通过其他网关、无线连接或拨号连接来旁路防火墙是不可能的。在网络环境中用“墙”来做比喻有些误导，因为有大量流量穿过防火墙。一种理解防火墙的方法是它在默认情况下阻止流量，除非该流量被专门允许通过。例如，它可能会过滤掉所有到达特定 IP 地址或特定 TCP 端口号的输入消息。

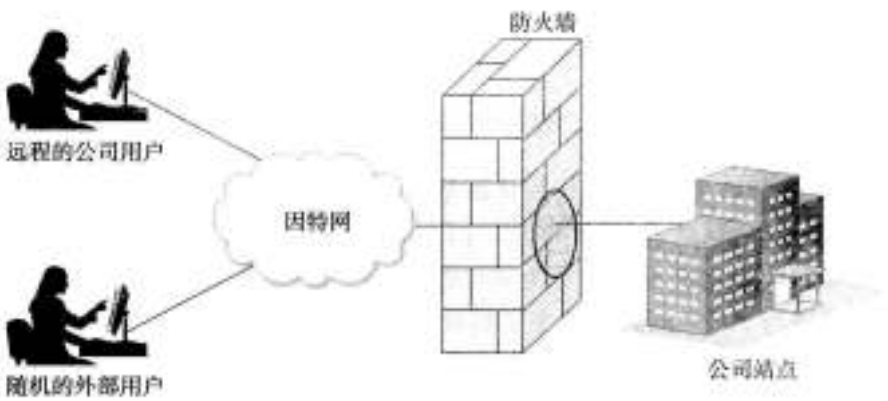


图 8-20 防火墙过滤站点与因特网其他部分之间的分组

事实上，防火墙将网络划分为一个较可信的区域和一个不太可信的区域[⊖]这对于不想让外部用户访问站点内部一个特定的主机或服务的情况是有用的。而问题的复杂性来自于你希望允许不同的外部用户使用不同的访问，外部用户的范围从一般公共用户到业务合作伙伴到组织内的远程成员。防火墙还可以对外出的流量施加限制，以便预防一些攻击，并且当攻击者成功获得防火墙内的访问权后限制造成的损失。

防火墙可以用于创建多个信任区域（zones of trust），就像一个信任度逐渐增加的信任区域组成的层次结构。一种常见的布局包括三个信任区域：内部网络、DMZ（非军事区）以及因特网的其他部分。DMZ 用于放置供外部访问的服务，如 DNS 和电子邮件服务器。内部网络和外部网络都可以访问 DMZ，但 DMZ 中的主机不能访问内部网络。因此，如果攻击者成功攻击了 DMZ 中的一台主机，他们依然不能访问内部网络。DMZ 可以定期重装，以便维持“干净”状态。

⊖ 防火墙的位置通常也是全局可寻址区域与使用本地地址的区域之间的分界线。因此，网络地址转换（Network Address Translation，NAT，见 4.1.3 节）功能和防火墙功能通常都能在同一设备中找到，虽然它们在逻辑上是分离的。

防火墙根据 IP、TCP 和 UDP 等信息进行过滤，配置过程使用一个能表示要转发和不转发分组特点的地址表。但就地址而言，我们不仅仅是指目标 IP 地址，虽然这是一种可能性。通常，表中的每一条记录是一个 4 元组：它给出源和目标的 IP 地址以及 TCP（或 UDP）端口号。

例如，一个防火墙可以被配置成过滤掉（不转发）与下列描述匹配的所有分组：

<192.12.13.14，1234，128.7.6.5，80>

这个模式指出过滤掉从地址为 192.12.13.14 的主机的 1234 端口发送到主机 128.7.6.5 的 80 端口的所有分组。（端口 80 是众所周知的用于 HTTP 的 TCP 端口。）当然，列出每个希望过滤其分组的源主机的名字常常是不实际的，所以，模式可以包含通配符。例如，

<*，*，128.7.6.5,80>

指出过滤发送到 128.7.6.5 上 80 端口的所有分组，不管是什么源主机或端口发送的分组。注意，像这些地址模式要求防火墙除了基于第三层的主机地址外，还要基于第四层的端口号做出转发/过滤的决定，因此网络层防火墙有时也称为*第四层交换机*（level 4 switches）。

在前面的讨论中，防火墙会转发所有分组，除非被特别设置为过滤掉某类分组。防火墙也可以过滤掉所有分组，除非被显式设置为转发该分组。也可以混合使用这两种策略。例如，防火墙可以被设置为只允许访问在一个特定邮件服务器上的 25 端口（SMTP 邮件端口），而不是阻塞访问 128.7.6.5 主机上的 80 端口，例如，

<*，*，128.19.20.21,25>

但要阻塞所有其他通信量。经验表明，防火墙经常会设置错误，而允许不安全的访问。一个问题是过滤规则会以复杂的方式重叠，使得系统管理员很难正确地表达过滤意图。使安全性最大化的一个设计原则是将防火墙配置为丢弃所有分组，除非该分组被显式地允许通过。当然，这意味着某些有效的应用可能会被意外阻止，这些应用的用户最终会注意到问题并让系统管理员进行适当的更改。

很多客户端-服务器应用动态地为客户分配端口。如果防火墙内的一个客户发起了到外部服务器的访问，该服务器的响应会被寻址到这个动态分配的端口。这会引起一个问题：如何设置防火墙，使之允许来自服务器的任何响应分组，但是要阻止非客户请求的类似分组？这对于单独处理每一个分组的*无状态防火墙*（stateless firewall）是不可能实现的，因此需要采用*有状态防火墙*（stateful firewall），它会记录每个连接的状态信息。一个被寻址到动态分配端口的输入分组会被允许通过的唯一条件是，它是该端口上连接的当前状态的一个有效响应。

现代防火墙也能够理解并根据很多特定的应用层协议（如 HTTP、Telnet 或 FTP）进行过滤，它们使用特定于那个协议的信息（如 HTTP 中的 URL）来决定是否丢弃消息。

防火墙的优点和缺点

防火墙只是保护网络免受来自因特网其他部分的不期望的访问，它不能为防火墙内部网络之间以及防火墙外部网络之间的合法通信提供安全性。相比之下，本章所描述的基于密码的安全机制则能够为处在任何位置的参与者之间提供安全的通信。既然情况如此，为什么防火墙还是这么普遍？一个原因是部署防火墙时只需要用成熟的商业产品在一方部署，而基于密码的安全需要通信的两端都提供支持。防火墙处于主导地位的一个更本质的原因是它们在一个中心位置封装了安全性，实际上将安全性从网络的其他部分剥离。系统

管理员能够通过管理防火墙来提供安全性，使得防火墙内的用户和应用不需要考虑安全问题——至少不需要考虑某些安全问题。

遗憾的是，防火墙有严重的缺陷。因为防火墙不限制防火墙内主机之间的通信，所以能够控制站点内代码运行的攻击者可以访问所有本地主机。攻击者如何进入防火墙呢？攻击者可能是一个拥有合法访问权限的不满的员工，或者攻击者的软件被隐藏在通过CD安装或从Web下载的软件中，或者攻击者能够通过无线通信或电话拨号连接来旁路防火墙。

另一问题是，任何被允许通过防火墙的实体（如商业伙伴或位于外部的员工）都可能成为安全缺陷。如果他们的安全性不像你的那么好，那么攻击者能够通过穿透他们的安全防护从而穿透你的安全防护。

防火墙最严重的问题之一是容易被防火墙内部计算机上的缺陷破坏。这些缺陷经常被发现，因此管理员不得不一直监视相关公告。但管理员经常做不到这一点，因为防火墙安全违规经常利用存在了一段时间而且有简单解决方案的漏洞。

术语恶意软件（malicious software，malware）是指被设计成以一种计算机用户不期望的并且不易发觉的方式运作的软件。病毒、蠕虫和间谍软件是恶意软件的常见类型。（病毒（virus）有时与恶意软件通用，但我们以狭义方式使用它，只是指特定的一类恶意软件）。恶意代码不一定是可执行的目标码，它也可能是解释代码，如脚本或微软Word中使用的可执行的宏。

病毒（virus）和蠕虫（worm）的特点是能够制造并散布自身的备份，它们之间的区别是：蠕虫是完整的程序，而病毒是插入（插入它自己的备份）到另一个软件中或一个文件中的一段代码，因此，当软件执行或打开文件的时候，病毒也被执行了。典型情况下，病毒和蠕虫会造成因试图传播自身的备份而引起的网络带宽消耗。更糟糕的是，它们也可能会以不同方式蓄意破坏系统或毁坏其安全性。例如，它们可以安装一个后门（backdoor），后门是不经过正常认证就允许远程访问系统的软件。这可能会导致防火墙暴露一个本身应该提供认证过程而被后门破坏的服务。

间谍软件是未经授权的收集和传输有关计算机系统及其用户隐私信息的软件。通常，间谍软件被秘密地嵌入到一个本来有用的程序中，并随用户安装该程序的备份时被传播。对防火墙来说，问题是这些私有信息的传输看上去像合法通信。

一个很自然被提出的问题是，防火墙（或密码学安全）能否首先将恶意软件排除在系统之外。绝大部分恶意软件实际上是通过网络传输的，虽然也可能通过可移动存储设备传输，如CD和记忆棒。当然，这是支持"阻止没有被显式允许的所有信息"，很多管理员都在配置防火墙时采用该方法。

检测恶意软件的一种方法是查找来自已知恶意软件的代码段，有时称为特征（signature）。聪明的恶意软件会用不同的方式调整其表示方式，因此这种方法受到了限制。对进入网络的数据进行如此详细的审查可能会影响网络性能。密码学安全也不能消除这个问题，虽然它的确提供了一种认证软件来源并检测篡改（例如病毒插入自身的备份）的方法。

与防火墙相关的是入侵检测系统（Intrusion Detection System，IDS）和入侵防御系统（Intrusion Prevention System，IPS）。这些系统试图寻找异常行为，如发向给定主机或给定端口号的不正常的大量流量，并向网络管理者生成警告甚至直接采取行动限制可能的攻击，虽然现在在该领域有商业产品，但它仍处在发展阶段。

8.6 小结

像因特网这样的网络是由利益互相冲突的实体所共享的，这种情况在网络互联的早期是完全无法预见的。网络安全的任务就是防止一些用户监视或影响其他用户的网络使用。机密性是通过加密消息来得到的，数据完整性可以通过使用密码散列来确保，两种技术结合起来可以确保消息的真实性。

对称密钥密码，如 AES 和 3DES，在加密和解密时使用相同的秘密密钥，因此发送方和接收方必须共享相同的密钥。公钥密码，如 RSA，使用公钥进行加密，使用一个秘密的私钥进行解密。这意味着任何实体都能用该公钥加密一条消息，使得消息只对私钥的持有者才可读。已知最快的破解已建立的密码（如 AES 和 RSA）的方法是在可能的密钥空间中进行蛮力搜索，通过使用大密钥能够使得这种方法在计算上不可行。大部分用于保密的加密使用对称密钥密码，因为它们有非常出众的速度，而公钥密码通常用于认证和会话密钥建立。

认证码是附加到消息后用于验证消息真实性和数据完整性的一个值。生成认证码的一种方法是加密由密码散列函数（如 MD5 或 SHA-1 散列族中的一个函数）输出的消息摘要。如果消息摘要是用公钥密码的私钥加密的，所得出的认证码被认为是数字签名，因为公钥可以被用于验证只有私钥的拥有者能够产生该签名。另一类认证码是消息认证码，它是由类散列函数以一个共享的秘密值为参数而产生的。散列 MAC 是将密码散列应用于明文消息与该秘密值的串接后计算出的 MAC。

会话密钥被用于保护相对较短的一段通信。会话密钥的动态建立依赖于长期存在的预分发密钥。特定实体对预分发密钥的所有权可以通过由可信实体数字签名的公钥证书来证明。公钥基础设施是用来验证这种绑定的完整机制，它依赖于信任网络或信任链。用于对称密钥密码的密钥的预分发方法是不同的，因为无法使用公钥证书，并且每一对参与者都需要唯一的密钥。密钥分发中心与每一个参与者都共享一个预分发密钥的可信实体，因此参与者之间可以使用会话密钥，而不能使用预分发密钥。

认证和会话密钥建立要求协议确保消息的时效性和原始性。时间戳或当前值能够用于标识消息的时效性。我们看了两个使用公钥密码的认证协议，一个要求时钟同步，而另一个不需要。Needham-Schroeder 是认证两个参与者的协议，每个参与者都与密钥分发中心共享一个主对称密码密钥。Kerberos 是一个基于 Needham-Schroeder 协议的认证系统，专门用于客户端/服务器环境。Diffie-Hellman 密钥协商协议在不预分发密钥和不认证的情况下建立会话密钥。

我们介绍了几个使用这些密码算法和协议提供安全性的系统。在应用层，PGP 可以用来保护电子邮件消息，SSH 可以用于安全地连接到一台远程计算机。在传输层，TLS 可用于保护 WWW 上的商务交易。在网络层，IPsec 体系结构可用于因特网上任何主机和网关集合之间的安全通信。

防火墙对它所保护的站点与网络其他部分之间传输的消息进行过滤。防火墙根据 IP、TCP 和 UDP 地址进行过滤，也根据一些应用协议中的字段进行过滤。有状态防火墙跟踪每个连接的状态，以便允许将有效的响应传送到动态分配的端口。虽然防火墙安全有一些重要的局限，但它的优点是将一些安全责任从用户和应用转移到系统管理员。

接下来会发生什么：面对安全

如果你问任何因特网研究者这样一个问题："如果我们能够重新建造因特网，什么会是未来因特网应具备的最重要的特点？"，答案很可能会涉及更好的安全性。思考这个问题的一个角度是因特网是由一个很小的团体设计的，当时他们只是想访问对方的计算机，而当今的因特网被一个巨大的全球性的团体所使用，包括相当数量的犯罪分子希望访问其他大量计算机。因此，这种默认公开访问的设计不太适合当今社会。

有大量理论研究试图改善这种状况。当然，一个问题是因特网是日常生活中如此重要的部分，以至于我们无法想象用一个新的重新设计的版本来替换它。但是，的确有大量"从零开始"的研究正在进行，其理论基础工作在不受增量部署问题影响的未来的互联网上，这可能会带来一些新的见解，在将来用于更新当前的因特网。（参考第3章中的"扩展阅读"。）

短期展望似乎是继续玩猫捉老鼠的游戏。防火墙、入侵检测系统和DoS缓解系统变得更复杂，攻击者寻找新方法来对付这些系统的防御，这些系统则变得更擅长抵抗新攻击。有利的一面是，很多安全系统效果很好，如果不是因为传输层安全（TLS）和所有密码学方法提供有效的支持，Web上不会有如此多的电子商务系统。

曾用于描述因特网未来愿景的词是"一个值得社会信任的互联网"。显然，实现这个愿景是巨大的挑战，保护网络安全将会是未来一段时间内的一个研究和创新的领域。

扩展阅读

前篇有关安全的文章，共同对本主题给出了很好的综述。Lampson等人的文章包含安全的形式化论述，而Satyanarayanan的文章对在实际中怎样设计一个安全系统进行了详细描述。第三篇文章给出IPsec安全体系结构的概述，而且它是全面了解当今因特网安全状态的很好的起点。

- Lampson, B. et al. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems* 10 (4): 265-310, November 1992.
- Satyanarayanan, M. Integrating security in a large distributed system. *ACM Transactions on Computer Systems* 7 (3): 247-280, August 1989.
- Staniford, S., V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. *USENIX Security Symposium* 2002, pp. 149- 167. San Francisco, CA, August 2002.

有几本好书全面介绍了网络安全领域的各个方面。我们推荐Schneier [Sch95]、Stallings [Sta03] 和Kaufman等 [KPS02] 的书。前两本对本主题给出全面的分析，而最后一本就这个主题给出非常易读的概述。IPsec的完整的体系结构定义在一系列RFC中：[Ken05a]、[Eas05]、[MG98a]、[MG98b]、[MD98]、[Ken05b]、[Kau05]。OpenPGP标准的定义在 [cal07] 中，最新的TLS标准的定义在 [DR08] 中。Barrett和Silverman的书 [BS01] 给出SSH的完整描述。Menezes等 [MvOV96] 的书是一本全面的密码学参考书（在下面的URL中能够免费下载该书的拷贝）。

对于辨别和防范拒绝服务攻击这一问题的讨论，可以在Moore等 [MVS01]、Spatscheck和Peterson [SP99] 以及Qie等 [QPP02] 的书中看到。用来辨别攻击源的最新技

术可以在 Savage 等［SWKA00］和 Snoeren 等［SPS+01］的几篇论文中找到。Garber［Gar00］和 Harrison［Har00］讨论了 DDoS 攻击带来的越来越大的威胁。Park 和 Lee 的论文［PL01］介绍了早期用来防范这些攻击的一种方法。Yang 等［YWA08］的 TVA 方法是一种新的 Dos 防御方法，属于“从零开始”的研究类型。

最后，我们推荐下列网址供实时参考：

- http://www.cert.org/：计算机紧急事件响应组（CERT）的网址，CERT 是关注计算机安全问题的组织。
- http://www.cacr.math.uwaterloo.ca/hac/：［MvOV96］的可下载拷贝，这是一本全面的密码学参考书。

习题

1. 在你的系统中查找或安装一个加密程序（例如，Unix 的 des 命令或 pgp 命令）。阅读其文档并用它进行实验。测试用它对数据加密和解密有多快。加密和解密的速率相同吗？试比较一些使用不同密钥长度的计时结果，例如，比较 AES 和三重 DES。
2. 画出类似 8.1.1 节中描述的图来表示密码分组链。
3. 学习一个密钥托管方案（例如，Clipper）。密钥托管的优点和缺点是什么？
4. 一个好的密码散列算法应该产生随机输出。换言之，任何给定散列值的概率应该与随机选取的输入数据对应的散列值的概率大致相同。如果所使用的散列算法的输出不是随机的，会产生什么后果呢？例如，考虑这种情况，一些散列值出现的可能性是其他散列值的 2 倍。
5. 假设 Alice 使用 8.3.3 节中描述的 Needham-Schroeder 认证协议来发起一个与 Bob 的会话。进一步假设攻击者能够窃听认证消息，并在会话完成后很久才能发现（解密）会话密钥。攻击者如何欺骗 Bob 使他将攻击者认证为 Alice。
6. 在口令认证中抵制重放攻击的一个机制是使用一次性口令（one-time password）：准备一个口令列表，一旦 password［N］被接受，服务器就对 N 减 1，并在下次提示输入 password［N−1］。当 N=0 时，需要新列表。描述一个机制，使得用户和服务器仅需要记住一个主口令 mp，并且有在本地计算 password［N］=f(mp，N) 的可用方法。提示：设 g 是一个合适的单向函数（如 MD5），并设 password［N］=g^{N}(mp) =g 对 mp 应用 N 次。解释为什么知道 password［N］并不会有助于揭示 password［N−1］。
7. 假设用户使用上述的一次性口令（或者可重用的口令），但是口令的传送足够慢。
   (a) 说明窃听者在一定数量的猜测后可获得对远程服务器的访问。（提示：原用户输入口令的除最后一个字符外的所有字符后，窃听者开始猜测。）
   (b) 一次性口令的用户可能遇到什么样的其他攻击？
8. Diffie-Hellman 密钥交换协议容易受到“中间人”攻击，如 8.3.4 节和图 8-12 所示。简述怎样扩展 Diffie-Hellman 才能让它防范这种可能的攻击。
9. 假设我们有一个非常短的秘密 s（例如，一个比特或者一个社会保障码），而且我们希望发送给其他人一条消息 m，其中不会暴露 s，但随后能用于验证我们确实知道 s。解释为什么利用 RSA 加密的 m=MD5(s) 或者 m=E(s) 不是安全的选择，并给出一个更好的选择。
10. 假设两个人想在网络上打扑克。为了发牌，他们需要一个在他们之间公平选择随机数 x 的机制，如果一方能不公平地影响 x 的选择，则另一方输了。描述这样的一个机制。（提示：假设如果两个比特串 x_1、x_2 之一是随机的，则二者的异或 $x=x_1 \oplus x_2$ 也是随机的。）
11. 估算找到两条具有相同 MD5 校验和的消息的概率，假设消息的总数分别是 2^{63}、2^{64} 和 2^{65}。（提示：这又是一个生日问题，就像第 2 章的习题 48 那样，而且第 $k+1$ 个消息有不同于前 k 个消息的校验和的概率是 $1-k/2^{128}$。但是，在那道习题的提示中简化了积的近似值，这一点现在不适用了。所以，每一端分别取对数，并使用近似值 $\log(1-k/2^{128}) \approx -k/2^{128}$。）

12. 假设我们想用 3DES 加密一个 Telnet 会话。Telnet 发送很多 1 字节的消息，而 3DES 一次加密 8 字节的分组。解释怎样在这个场景中安全地使用 3DES。
13. 考虑下列用于下载文件的简单 UDP 协议（大体上以 TFTP 为基础，即 RFC 1350）：
    - 客户端发送一个文件请求。
    - 服务器用第一个数据分组响应。
    - 客户端发送 ACK，双方以停止－等待协议进行交互。

    假设客户端和服务器分别拥有密钥 K_c 和 K_s，并且彼此知道对方的密钥。
    (a) 用这些密钥和 MD5 扩展文件下载协议，提供发送方认证和消息完整性。你的协议也应是可抵抗重放攻击的。
    (b) 在你修改协议时所增加的额外信息怎样防止出现晚到的来自之前连接的分组以及序列号的回绕。
14. 利用你选择的浏览器，找到你的浏览器对 HTTPS 默认配置的是什么认证机构。你信任这些代理机构吗？当你设置为不信任这些认证机构中的一些或全部时，会发生什么？
15. 使用 OpenPGP 的一个实现（如 GnuPG）来完成下列练习。注意，不包括电子邮件——你只操作一台计算机上的文件。
    (a) 产生一个公钥/私钥对。
    (b) 用公钥加密一个文件，以便安全存储，然后用私钥解密。
    (c) 用你的密钥对来数字签名一个未加密的文件，然后假设你是别人，用你的公钥验证你的签名。
    (d) 假设第一个公钥/私钥对属于 Alice，为 Bob 产生第二个公钥/私钥对。扮演 Alice，加密并签名一个发给 Bob 的文件（确保以 Alice 签名，而不是以 Bob）。然后，扮演 Bob，验证 Alice 的签名，并解密该文件。
16. 考虑 8.2.1 节中所描述的证书分层结构。根 CA 为一个二层 CA 签发了一个证书，该二层 CA 为 Alice 签发了一个证书。Bob 有根 CA 的公钥，所以他能够验证二层 CA 的证书。为什么 Bob 仍然可能不相信 Alice 是证书中公钥的拥有者？
17. PuTTY（发音为 "putty"）是一个流行的免费 SSH 客户端——实现 SSH 连接的客户端应用程序——用于 Unix 和 Windows，可以在 Web 上访问它的文档。
    (a) PuTTY 如何处理对以前没有连接过的服务器的认证？
    (b) 客户端是如何被服务器认证的？
    (c) PuTTY 支持多种密码。它如何确定每一个特定的连接使用哪一种密码？
    (d) PuTTY 支持像 DES 这样的密码，这在某些或所有情况下都被认为太脆弱。为什么？PuTTY 如何判断哪些密码脆弱？它是如何利用这些信息的？
    (e) 对于一个给定的连接，PuTTY 允许用户指定所传输数据的最长时间和最大传输数据量，然后 PuTTY 开始建立一个新的会话密钥，这个过程在文档中称为密钥交换（key exchange）或密钥重发（rekeying）。这个特性的目的是什么？
    (f) 使用 PuTTY 密钥生成器 PuTTYgen 为 PuTTY 所支持的一种公钥密码产生一个公钥/私钥对。
18. 假设你希望防火墙阻止所有进入的 Telnet 连接，但允许出去的 Telnet 连接。一种方法是阻止到指定的 Telnet 端口（23）的所有进入的分组。
    (a) 我们可能也希望阻止到其他端口的进入分组，但为了不妨碍出去的 Telnet，必须允许哪些进入的 TCP 连接？
    (b) 现在假设允许你的防火墙除端口号之外还可使用 TCP 首部的 Flags 位。解释怎样达到你所希望的 Telnet 效果，同时不允许进入的 TCP 连接。
19. 假设防火墙被配置为允许出去的 TCP 连接，但仅允许到指定的端口的进入的连接。现在 FTP 协议提出一个问题：当一个内部客户联系外部服务器时，出去的 TCP 控制连接可以正常打开，但 TCP 数据连接传统上是进入的。

(a) 查找 FTP 协议，例如在 RFC 959 中查找，清楚 PORT 命令如何工作，讨论如何写客户端才能限制防火墙允许访问的端口的数量。这种端口的数量可以限制为 1 吗？

(b) FTP PASV 命令如何用于解决防火墙问题？

20. 假设过滤路由器的设置如图 8-21 所示：主防火墙是 R1。解释如何配置 R1 和 R2，使外界能远程登录到网络 2，但不能登录到网络 1 上的主机。为了避免“跃过”闯入网络 1，也不许从网络 2 远程登录到网络 1。

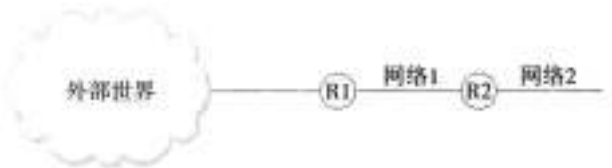


图 8-21 习题 20 图

21. 为什么因特网服务提供商可能想阻止某些外出通信量？

22. 据说 IPsec 不一定能与网络地址转换（Network Address Translation，NAT）（RFC 1631）一起工作。然而，IPsec 可不可以与 NAT 一起工作取决于我们使用 IPsec 和 NAT 的哪个模式。假设我们使用真正的 NAT，只转换 IP 地址（不转换端口）。IPsec 和 NAT 能在下列各种情况下工作吗？解释为什么。

(a) IPsec 使用 ESP 传输模式。

(b) IPsec 使用 ESP 隧道模式。

(c) 如果我们使用端口地址转换（Port Address Translation，PAT）会怎样？PAT 也称为 NAT 中的网络地址/端口转换（Network Address/Port Translation，NAPT），其中除了对 IP 地址进行转换外，还要对端口号进行转换，以便在专用网络外部共享一个 IP 地址。

第9章
Computer Networks: A Systems Approach

应　　用

现在不是结束，甚至不是结束的开始，但可能是开始的结束。

——温斯顿·丘吉尔

问题：应用需要自己的协议

本书开始谈论了人们想在计算机网络上运行应用程序，从 Web 浏览器到视频会议工具，可谓包罗万象。在中间的各章，我们循序渐进地展开了构建这些应用所需的网络体系结构。我们现在循环回到原点，返回到网络应用。这些应用可部分地看作是网络协议（从它们与其他计算机上的对等实体交换消息的角度来看）。部分看作是传统应用程序（从它们与窗口系统、文件系统以及最终和用户之间相互作用的角度来看）。本章探讨现今一些最流行的网络应用。

考察应用能够使我们理解在整本书中一直强调的系统方法（system approach）。因此，构建高效联网应用的最好方法是理解网络能够提供的构建模块以及这些模块是如何互相交互的。例如，一个特定的网络应用可能需要利用可靠的传输协议、认证和保密机制以及底层网络的资源分配能力。通常在应用开发者知道如何充分利用这些工具时（也有大量的没有很好地利用可用网络能力的反面应用实例），应用才工作得最好。应用一般也需要它们自己的协议，在很多情况下使用我们在研究低层协议时已经看到的同样的原则。因此，本章我们关注如何将已经描述的思想和技术组合在一起来构建高效联网应用。换句话说，如果你想象自己编写一个网络应用，那么根据定义你就成为一个协议设计者（和实现者）。

我们将研究各种熟悉和不熟悉的网络应用，包括交换电子邮件、Web 冲浪、跨业务整合应用、管理网络元素的集合、像 vic 和 vat 那样的多媒体应用，以及新兴的对等网和内容分发网络。这些罗列虽显粗略，但它举例说明了设计和构建应用的要领。应用需要在网络中或主机协议栈中的其他层选择适当的组件，然后增强那些低层服务以提供应用所需要的精确的通信服务。

9.1　传统应用

我们以最流行应用中的两个例子（万维网（Word Wide Web）和电子邮件）开始对应用的讨论。广义地讲，这两个应用都使用请求/响应模式——用户向服务器发出请求，服务器随之响应。我们称它们为传统应用，因为它们代表着自从早期计算机网络时代就已存在的应用种类。（虽然 Web 比电子邮件新很多，但其基础是早期的文件传输。）与之对照，后面的章节将讨论最近才变得可行的一类应用：流应用（即像视频和音频之类的多媒体应用）和各种基于覆盖的应用。（注意，这些分类之间的界限有点模糊，因为你当然可以通过 Web 访问流媒体数据，但现在我们把重点放在 Web 的一般用法，即请求网页、图片等。）

在详细讨论每个应用之前，我们需要清楚三个要点。第一点非常重要，即区分应用程

序（program）与应用协议（protocol）。例如，超文本传输协议（HyperText Transport Protocal，HTTP）是一个用于从远程服务器上获取网页的应用协议。有许多不同的应用程序（如 Internet Explorer、Netscape、Firefox 和 Safari 那样的 Web 客户端）提供给用户不同的外观和感觉，但是它们都使用同样的 HTTP 协议在因特网上与 Web 服务器通信。当然，事实上该协议已公开并已成为标准，允许很多不同公司和个人开发的应用程序与所有 Web 服务器（有很多变种）实现互操作。

本节着重于两个使用非常广泛并且已成为标准的应用协议：

- SMTP：简单邮件传输协议（Simple Mail Transfer Protocol），用于交换电子邮件。
- HTTP：超文本传输协议（HyperText Transport Protocol），用于 Web 浏览器和 Web 服务器之间的通信。

我们还会研究个人应用协议是如何在 Web 服务（Web Service）框架中定义的。

第二点，由于在这一节中描述的所有应用协议都遵循相同的请求/响应通信模式，所以我们期望它们都建立在 RPC 传输协议之上。然而并非如此，它们全都实现在 TCP 之上。实际上，每个协议是在一个已有的可靠传输协议之上重新改造一个简单的类似 RPC 的机制。我们说“简单”，是因为每个协议都没有设计成支持 5.3 节中讨论的任意远程过程调用，而是设计为发送和响应一个特定的请求消息集合。

最后，我们发现很多应用层协议，包括 HTTP 和 SMTP，都有一个配套协议来说明能交换的数据格式。这是这些协议相对简单的一个原因：大部分复杂的事情在这个配套文档中管理。例如，SMTP 是一个交换电子邮件消息的协议，但 RFC 822 和多功能因特网邮件扩展（Multipurpose Internet Mail Extensions，MIME）定义邮件消息的格式。类似地，HTTP 是获取 Web 页面的协议，但超文本标记语言（HyperText Markup Language，HTML）是一个配套规范，它定义这些网页的格式。

9.1.1 电子邮件（SMTP、MIME、IMAP）

电子邮件是最早的网络应用之一。毕竟，在你刚获准在一条跨国链路上运行时，有什么比希望向链路另一端的用户发送一条消息更自然的呢？令人惊讶的是，ARPANET 的先驱并没有料到在网络建成后（那时，远程访问计算机资源是主要的设计目标），电子邮件会成为一个关键应用，但是电子邮件变成了很有用的应用并且一直非常流行。

如前所述，有两点很重要：①区分底层的消息传输协议（如 SMTP 或 IMAP）与用户接口（即邮件阅读器），②区分这个传输协议（SMTP）与定义交换消息格式的配套协议（RFC 822 和 MIME）。我们将从考查消息格式开始。

1. 消息格式

RFC 822 将消息定义为两个部分：首部（header）和主体（body）。两部分都用 ASCII 文本来表述。最初，消息主体被设想为简单的文本。现在情况仍然如此，尽管 RFC 822 已被 MIME 扩充以允许主体支持所有种类的数据。这些数据仍用 ASCII 文本表示，但因为它可能是已编码的版本（如 JPEG 图像），所以对用户来说不一定是可读的。稍后将更多地提到 MIME。

消息首部是一系列以〈CRLF〉终止的行。（〈CRLF〉代表回车+换行，是一对 ASCII 控制符，常常用于指示文本行的结尾。）首部与主体之间用一个空行分隔。每个首部行包

括了由冒号分隔的类型和值。许多首部行是用户所熟悉的，因为当他们写电子邮件消息时要求填写这些行。例如，To：首部用来识别消息的接收者，而 Subject：首部用来说明这个消息的目的。其他的首部由低层的邮件递送系统填写。例如 Date：(消息传送时间)，From：(发送消息的用户) 和 Received：(处理这个消息的每个邮件服务器)。当然，还有许多其他的首部行，感兴趣的用户可参考 RFC 822。

RFC 822 在 1993 年被扩充（此后又多次更新），以便允许邮件消息携带许多不同的数据类型：音频、视频、图像、PDF 文档等。MIME 包括三个基本部分。第一部分是一个首部行的集合，它扩充了由 RFC 822 定义的原始集合。这些首部行以各种方式描述了放到主体中的数据。它们包括 MIME-Version：(正在使用的 MIME 版本)，Content-Description：(关于消息中内容的可读性描述，类似于 Subject：行)，Content-Type：(消息中包含的数据类型) 和 Content-Transfer-Encoding：(主体中的数据是如何编码的)。

第二部分是一组内容类型（和子类型）的定义。例如，MIME 定义两种不同的静态图像类型，表示为 image/gif 和 image/jpeg，每种表示的含义都显而易见。作为另一个例子，text/plain 表示简单文本，你可能会一种 822 式消息中找到它，而 text/richtext 表示消息包含“标记”文本（如使用特殊的字体、斜体等的文本）。作为第三个例子，MIME 定义一个 application 类型，其子类型对应不同应用程序的输出（如 application/postscript 和 application/msword）。

MIME 也定义了一个 multipart 类型，说明如何构造一个带有多种数据类型的消息。这就像编程语言定义基本类型（如整数和浮点数）和组合类型（如结构和数组）。一种可能的 multipart 子类型是 mixed，它说明消息包含一组指定顺序的独立数据块。每一块有自身描述本块类型的首部行。

第三部分是编码各种类型数据的方法，以便使数据可以在一个 ASCII 电子邮件消息中传递。问题是，对于一些数据类型（如 JPEG 图像），在图像中任何给定的 8 比特字节都可能包含 256 个不同值中的一个，而只有这些值的一个子集是有效的 ASCII 字符。邮件消息中仅包含 ASCII 是很重要的，因为它们可能通过一系列中间系统（就像下面将要描述的网关）。中间系统假设所有邮件都是 ASCII 字符，如果其中包含非 ASCII 字符将破坏该消息。为解决这个问题，MIME 使用了一种简单地将二进制数编码到 ASCII 字符集中的方法。这个编码方法称为 base64。其思路是将原始二进制数据的每 3 个字节映射到 4 个 ASCII 字符。具体方法是将二进制数以 24 比特为单位分组，并将每个分组分成 4 个 6 比特的块。每 6 比特映射成 64 个有效的 ASCII 字符之一。例如，0 映射到 A，1 映射到 B 等。如果你看到采用 base64 方案编码的消息，你将注意到仅有 52 个大小写字母，10 个 0～9 的数字及特殊字符“+”和“/”。这些正是 ASCII 字符集中最前面的 64 个值。

另一方面，为了让那些坚持使用纯文本邮件的用户阅读邮件时尽可能不费力气，一个由常规纯文本组成的 MIME 消息只能使用 7 比特 ASCII 编码。对大多数 ASCII 数据还有其他可读的编码。

综上所述，一条包含一些普通文本、一幅 JPEG 图像和一个 PostScript 文件的消息，看上去如下所示：

```
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="--------417CA6E2DE4A8CAF8C5"
From: Alice Smith <Alice@cisco.com>
To: Bob@cs.Princeton.edu
Subject: promised material
Date: Mon, 07 Sep 1998 19:05:19 -0400

----------417CA6E2DE4A8CAF8C5
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Bob,

Here's the jpeg image and draft report I promised.

--Alice

----------417CA6E2DE4A8CAF8C5
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
... unreadable encoding of a jpeg figure
----------417CA6E2DE4A8CAF8C5
Content-Type: application/postscript; name="draft.ps"
Content-Transfer-Encoding: 7bit
... readable encoding of a PostScript document
```

在这个例子中，消息首部中的 Content-Type 行指出这个消息包含不同的块，每一块用一个并不出现在数据本身中的字符串标明。并且每个块拥有它自己的 Content-Type 和 Content-Transfer-Encoding 行。

2. 消息传输

很多年来，从主机到主机传递的大部分电子邮件都只使用 SMTP。虽然 SMTP 仍然扮演着核心角色，但现在它只是多个协议中的一种，因特网消息访问协议（Internet Message Access Protocol，IMAP）和邮局协议（Post Office Protocol，POP）成为另外两个重要的读取邮件消息的协议。我们的讨论从 SMTP 开始，后面会转向 IMAP。

为了把 SMTP 放在正确的环境中，我们需要标识关键的角色。首先，当用户编写、存档、检索和阅读邮件时，他们要与邮件阅读器（mail reader）交互。有无数可用的邮件阅读器，就像有很多可选择的 Web 浏览器一样。在因特网的早期，用户一般登录到其邮箱（mailbox）所在的主机，他们所调用的邮件阅读器是一个从文件系统中读取消息的本地应用程序。当然，现在用户通过笔记本电脑或智能手机远程访问他们的邮箱，而并不是先登录到存储邮件的主机（邮件服务器）。从邮件服务器远程下载邮件到用户设备时使用另一个邮件传输协议，如 POP 或 IMAP。

第二，在每台承载邮箱的主机上运行一个邮件后台处理程序（mail daemon）（或进程）。你可以想象这个处理程序扮演着邮局的角色，也称为消息传送代理（message transfer agent，MTA）：邮件阅读器将它们想发送给其他用户的消息交给邮件后台处理程序，这个邮件后台处理程序使用运行在 TCP 之上的 SMTP 传输消息到另一台计算机上的邮件后台处理程序，并且邮件后台处理程序将收到的消息放入用户的邮箱中（那里，用户的邮

件阅读器随后能发现消息)。由于 SMTP 是一种任何人都能实现的协议，因此从理论上讲可以有许多不同的邮件后台处理程序的实现。然而，事实是只有少数流行的实现，其中来自 Berkeley UNIX 和 postfix 的老的 sendmail 程序是传播最广的。

虽然发送方计算机上的 sendmail 程序建立一个到接收方计算机上的 sendmail 程序的 SMTP/TCP 连接的确是可能的，但在很多情况下邮件要穿越从发送方主机到接收方主机的路径上的一个或多个邮件网关（mail gateway）。像端主机一样，这些网关也运行一个 sendmail 进程。这些中间节点称作网关并非偶然，因为它们的工作就是存储和转发邮件消息，很像“IP 网关”(就是我们所说的路由器）存储和转发 IP 数据报。它们仅有的差别是邮件网关通常把消息缓存在磁盘上并且在几天之内不断尝试再传送到下一台计算机上，而 IP 路由器把数据报缓存在内存中并且只在转瞬之间再尝试传送它们。图 9-1 说明一个从发送方到接收方的两跳的路径。

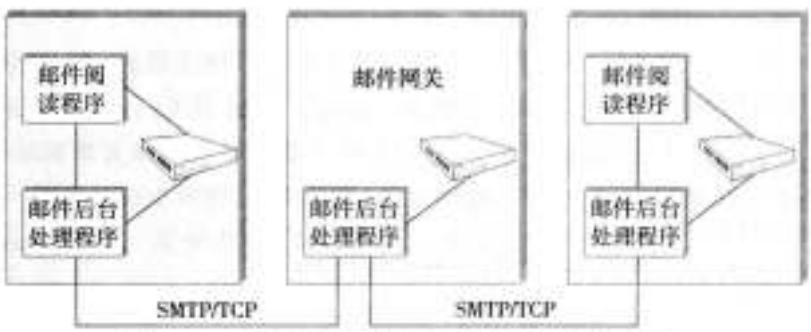


图 9-1 一系列邮件网关存储和转发电子邮件消息

你可能会问，邮件网关是必需的吗？为什么发送方主机不能直接将消息发送到接收方主机？一个原因是接收方不想把读邮件的特定主机包含在他的地址中。另一个原因是，在大型组织中，邮箱服务受到许多不同计算机的控制。例如，递交给 Bob@cs. princeton. edu 的邮件，首先被送到普林斯顿大学计算机科学系的一个邮件网关上（即名为 cs. princeton. edu 的主机)，然后被转发（这涉及第二个连接）到 Bob 今天用于阅读邮件的那台计算机上。转发网关维护一个数据库，这个数据库将用户映射到他们当前想要接收邮件的计算机上，发送方不必知道这台计算机的名字（消息首部行中的 Received: 将帮助你跟踪一个指定消息经过的邮件网关)。另一个原因是接收方计算机可能不总是开机的，在这种情况下邮件网关将保留消息直到它能被转发出去。

在每对主机之间使用一条独立的 SMTP 连接，以使消息逐步接近接收方，与路径上有多少个邮件网关无关。每个 SMTP 会话涉及在两个邮件后台处理程序之间的对话，一个充当客户而另一个充当服务器。在一次会话期间两台主机可能会传送多条消息。因为 RFC 822 使用 ASCII 作为基本表达形式来定义消息，所以得知 SMTP 也是基于 ASCII 的就不足为奇了，这意味人们假扮 SMTP 客户程序是可能的。

理解 SMTP 的最好方法是通过一个简单的例子。下面是在发送主机 cs. princeton. edu 和接收主机 cisco. com 之间的一次交换。在此情况下，普林斯顿大学的用户 Bob 尝试发送一个邮件给 Cisco 公司的 Alice 和 Tom。由 cs. princeton. edu 发出的行用黑体显示，而由 cisco. com 发出的行用斜体显示，添加额外的空白行是为了使对话更易读。

```
HELO cs.princeton.edu
250 Hello daemon@mail.cs.princeton.edu [128.32.169.24]

MAIL FROM:<Bob@cs.princeton.edu>
250 OK

RCPT TO:<Alice@cisco.com>
250 OK

RCPT TO:<Tom@cisco.com>
550 No such user here

DATA
354 Start mail input; end with <CRLF>.<CRLF>
Blah blah blah...
...etc. etc. etc.
<CRLF>.<CRLF>
250 OK

QUIT
221 Closing connection
```

正如你能看到的，SMTP 在客户和服务器之间包含一系列交换。在每一次交换中客户提交一个命令（如 HELO、MAIL、RCPT、DATA、QUIT），服务器以一个代码来响应（如 250、550、354、221）。服务器也返回对代码的可读性解释（如 No such user here）。在这个特例中，客户首先用 HELO 命令向服务器标识自己。它把自己的域名作为参数，服务器验证这个域名是否对应于 TCP 连接所使用的 IP 地址。你将注意到服务器会向客户声明该 IP 地址。然后客户询问服务器是否愿为两个不同的用户接收邮件，服务器通过对一个说“YES”而对另一个说“NO”来响应。然后客户发送消息，并使用只有一个圆点（“.”）的行作为消息的结束。最后，客户终止连接。

当然还有很多其他的命令和返回代码。例如，服务器可用代码 251 来响应客户的 RCPT 命令，它表明用户在该主机上没有邮箱，但服务器允诺将该消息转发到另一个邮件后台处理程序。换句话说，主机行使邮件网关的职能。另一例子，客户能发出 VRFY 操作命令来验证用户的电子邮件地址，但实际却没给该用户发送消息。

其他应注意的是 MAIL 和 RCPT 操作的参数。例如，分别有 FROM：<Bob@cs.princeton.edu>和 TO：<Alice@cisco.com>，这些看上去很像 822 的首部字段，而且在某种意义上也就是如此。实际情况是邮件后台处理程序解析消息，以抽取运行 SMTP 所需的信息。抽取的信息构成消息的信封（envelope）。SMTP 客户使用该信封作为与 SMTP 服务器交换的参数。从历史上看，sendmail 变得如此流行的原因是没有人想重新实现消息解析功能。虽然今天的电子邮件地址看上去非常好用（如 Bob@cs.princeton.edu），但情况并不总是如此。在因特网流行之前，一个诸如格式 user%host@site!neighbor 的电子邮件地址并不罕见。

3. 邮件阅读器

最后一步是用户从邮箱中实际接收消息，阅读、回复并可能保存一份副本以备将来参考。用户通过与邮件阅读器的交互来完成这些行为。如前所述，这个阅读器最初只是一个与用户邮箱运行在同一台计算机上的程序，在这种情况下，它只是简单地读写实现邮箱的

文件。这是在笔记本电脑时代之前的常见情况。现在，用户多数从远程机器上访问邮箱，但要使用其他协议，如 POP 或 IMAP。讨论用户邮件阅读器的接口形式超出本书的范围，但讨论访问协议一定在我们的范围之内。我们特别考虑 IMAP。

IMAP 在许多方面类似于 SMTP。它是一个运行在 TCP 之上的客户端/服务器协议，这里客户端（运行在用户的桌面计算机上）发出以〈CRLF〉结束的 ASCII 文本行命令，而邮件服务器（运行在维护用户邮箱的计算机上）做出响应。交互以客户认证自己的身份开始，并确定他要访问的邮箱。这可以表示成如图 9-2 所示的简单状态转换图。在这个图中，LOGIN、AUTHENTICATE、SELECT、EXAMINE、CLOSE 和 LOGOUT 是客户能发出的命令的例子，OK 是服务器的一种可能的响应。其他的常见命令包括 FETCH、STORE、DELETE 和 EXPUNGE，其含义不言而喻。另外的服务器响应包括 NO（客户没有权利执行该操作）和 BAD（命令的格式非法）。



（1）没有预认证的连接（OK问候）
（2）经过预认证的连接（PREAUTH问候）
（3）拒绝连接（BYE问候）
（4）LOGIN或AUTHENTICATE命令成功
（5）SELECT或EXAMINE命令成功
（6）CLOSE命令、失败的SELECT或EXAMINE命令
（7）LOGOUT命令，服务器关闭或连接关闭

图 9-2 IMAP 的状态转换图

当用户请求 FETCH（读取）一条消息时，服务器就以 MIME 格式返回它，然后由邮件阅读器解码。除消息本身之外，IMAP 也定义一组消息属性（attribute），这些属性作为其他命令的一部分进行交换，独立于传输消息本身。消息属性包括消息大小等信息，更有意思的是消息相关的各种标志（flag），如 Seen、Answered、Deleted 和 Recent。这些标志用于保持客户与服务器的同步，就是说，当用户在邮件阅读器上删除一条消息时，客户需要向邮件服务器报告这一事实。随后，如果用户决定清除所有被删除的消息，客户发出一个 EXPUNGE 命令给服务器，服务器便知道从邮箱中彻底删除所有之前被删除的消息。

最后，注意当用户回复一条消息或发送一条新消息时，邮件阅读器并不使用 IMAP 从客户向邮件服务器转发消息而使用 SMTP。这意味着用户的邮件服务器实际上是作为从桌面到接收方邮箱的传输路径上的第一个邮件网关。

9.1.2 万维网（HTTP）

万维网已经取得很大的成功并且使得如此多的人都能访问因特网，以至于万维网有时看起来像是因特网的同义词。事实上，形成 Web 的系统设计大约从 1989 年开始，远远晚于因特网成为广泛部署的系统的时间。Web 的最初目标是找到组织和抽取信息的方法，借

用超文本的思想——相互链接的文档（这种思想至少在20世纪60年代就存在了）[⊖]。

一个有助于理解Web的思路是：Web是一组相互协作的客户端与服务器，所有成员说的是同一种语言：HTTP。大多数人接触Web是通过图形化的客户程序或Web浏览器，例如Safari、Chrome、Firefox或Internet Explorer。图9-3展示了使用Firefox浏览器显示来自普林斯顿大学的一页信息。



图9-3　Firefox Web浏览器

显然，如果想将信息组织成由相互链接的文档或对象组成的系统，你需要从获取一个文档开始。因此，任何一个Web浏览器都有允许用户打开URL的功能。URL对大多数人来说是如此熟悉，以至于容易忘记它们不是一直存在的。URL提供有关Web上对象位置的信息，它们看起来有如下形式：

http://www.cs.princeton.edu/index.html

如果你打开这个特定的URL，你的Web浏览器将打开一个到称为www.cs.princeton.edu的计算机上的Web服务器的TCP连接，立刻获取并显示名为index.html的文件。Web上的多数文件包含图像和文本，有些还有音频或视频片段等。它们也经常包含指向其他文件的URL。而Web浏览器会通过某种办法使你能够识别这些URL对象并要求浏览器打开它们（一般通过高亮方式或在文本下加下划线）。这些嵌入的URL称为**超文本链接**（hypertext link）。当你要求Web浏览器打开这些嵌入的URL之一（即用鼠标指向它并单

⊖　万维网联盟所提供的Web简史可追溯到1945年的一篇描述缩微胶片文档之间链接的文章。

击）时，它将打开一个新的连接并取回和显示一个新文件。这称为跟随链接（following a link）。因此在网络范围内很容易从一台计算机跳到另一台计算机，并得到各种类型的信息。一旦你设法在一个文档中嵌入一个链接，并允许用户跟随该链接来得到另一个文档，你就有了超文本系统的基础。

当你选择查看一个网页时，你的浏览器（客户端）使用运行在 TCP 之上的 HTTP 从服务器上获取网页。像 SMTP 一样，HTTP 是面向文本的协议。本质上，每个 HTTP 消息有通用的格式

```
START_LINE <CRLF>
MESSAGE_HEADER <CRLF>
<CRLF>
MESSAGE_BODY <CRLF>
```

和从前一样，〈CRLF〉代表回车＋换行。第一行（START_LINE）指出这是一个请求消息还是一个响应消息。实际上，它识别一个要执行的“远程过程”（在请求消息的情况下），或识别一个请求的状态（status）（在响应消息的情况下）。下面的若干行说明一些限定请求或响应的选项和参数。可以有零个或多个 MESSAGE _ HEADER 行，它们以一个空白行结束，每行看起来都像电子邮件消息中的首部行。HTTP 定义了许多可能的首部类型，它们中的一些属于请求消息而另一些属于响应消息，还有一些属于消息主体中携带的数据。我们只给出几个有代表性的例子而不是给出所有可能的首部类型。最后，在空白行之后是请求的内容（MESSAGE _ BODY），请求消息中的这一部分通常是空项。

HTTP 为什么运行在 TCP 上？并非必须如此，但 TCP 确实能很好地匹配 HTTP 的需求，即提供了可靠传输（谁希望得到丢失了数据的网页呢）、流量控制和拥塞控制。然而，就像我们将在后面看到的，在 TCP 上构建一个请求/响应协议会引起一些问题，特别是如果你忽略了应用层协议与传输层协议之间交互的微妙之处。

1. 请求消息

HTTP 请求消息的首行说明三件事：应完成的操作、应完成操作所针对的网页和所用的 HTTP 版本。虽然 HTTP 定义了分类广泛的可能的请求操作，包括允许把一个网页投递到服务器上的写（write）操作，但最常用的两种操作是 GET（获取指定的网页）和 HEAD（获取指定网页的状态信息）。前者显然在浏览器想取回和显示一个网页时使用。后者用来测试一个超文本链接的合法性或者查看一个特定网页在上次浏览器获取后是否被修改过。表 9-1 总结了全部操作。POST 命令引起了因特网上的很多恶作剧（包括垃圾邮件），这听起来很无辜。

表 9-1 HTTP 请求操作

操 作	描 述
OPTIONS	请求关于可用选项的信息
GET	获得由 URL 标识的文档
HEAD	获得由 URL 标识的文档的状态信息
POST	递交信息（如注释）到服务器
PUT	在指定的 URL 下存储文档
DELETE	删除指定的 URL
TRACE	回送请求消息
CONNECT	由代理使用

例如，START_LINE

```
GET http://www.cs.princeton.edu/index.html
HTTP/1.1
```

说明客户想要让主机 www. cs. princeton. edu 上的服务器返回一个名为 index. html 的网

页。这个特例使用一个绝对的（absolute）URL。也有可能使用一个相对的（relative）标识并在 MESSAGE-HEADER 行中指定主机名，比如，

```
GET index.html HTTP/1.1
Host: www.cs.princeton.edu
```

其中 Host 是可能的 MESSAGE_HEADER 字段之一，这些字段中更有趣的是 If-Modified-Since，它给出一种使客户有条件地请求一个网页的办法，即只有在首部行指定的时间以后有人修改过该网页时，服务器才返回该网页。

2. 响应消息

像请求消息一样，响应消息也以一个单一的 START_LINE 行作为开始。在这种情况下，此行说明所使用的 HTTP 版本，三位代码指示请求是否成功，并且用一个文本串给出这种响应的原因。例如：START_LINE

```
HTTP/1.1 202 Accepted
```

指出服务器能够满足要求，而

```
HTTP/1.1 404 Not Found
```

指出因为网页没有找到而不能满足该请求。响应代码有五种通用类型，代码的第一位指明其类型。表 9-2 总结了五种代码。

表 9-2　HTTP 的五类结果代码

代　码	类　型	原 因 示 例
1xx	信息	接收请求，继续处理
2xx	成功	行为被成功接收、理解和接受
3xx	重定向	为完成请求所需的进一步的行为
4xx	客户错误	请求语法错或请求不能实施
5xx	服务器错误	服务器不能响应一个显然有效的请求

就像 POST 请求消息那些非预期的结果一样，有时人们会惊讶于实际中如何使用不同的响应消息。例如，通过将请求重定向到附近的缓存，请求重定向（代码是 302）成为在内容分发网络（Content Distribution Networks，CDN）中扮演重要角色的强大机制（见 9.4.3 节）。

与请求消息类似，响应消息也能包含一个或多个 MESSAGE _ HEADER 行。这些行传递返回给客户的附加信息。例如 Location 首部行说明所请求的 URL 在另一个位置是可用的。因此，如果普林斯顿大学计算机科学系的网页已经从 http://www.cs. princeton.edu/index.html 移到了 http://www.princeton.edu/cs/index.html，则原来地址的服务器可以响应如下：

```
HTTP/1.1 301 Moved Permanently
Location: http://www.princeton.edu/cs/index.html
```

通常情况下，响应消息也将携带请求的页面。这个页面是一 HTML 文档，但是因为它可以带有非文本数据（如 GIF 图像），所以它使用 MIME 编码（见 9.1.1 节）。某些 MESSAGE_HEADER 行给出页面内容的属性，包括 Content-Length（内容的字节数）、Expires（内容失效的时间）和 Last-Modified（内容在服务器上被修改的最后

时间）。

3. 统一资源标识符

HTTP 将 URL 作为地址，它是统一资源标识符（Uniform Resource Identifier，URI）的一种。URI 是一个标识资源的字符串，资源可以是具有标识的文档、图像或服务等的任何内容。

URI 的格式允许将多种更专用的资源标识符集成到标识符的 URI 空间中。URI 的第一部分是大纲（scheme），它命名表示特定种类的资源的特定方法，如将 mailto 用于电子邮件地址，将 file 用于文件名。URI 的第二部分与第一部分用分号隔开，是特定于大纲的部分（scheme-specific part）。它是符合第一部分中大纲的资源标识符，例如下面的 URI

```
mailto:santa@northpole.org
```

和

```
file:///C:/foo.html
```

资源不必可获取或可访问。我们看一个 7.1.3 节中的例子——可扩展标记语言（XML）命名空间是由类似 URL 的 URI 标识的，但严格地讲，它们并不是定位符（locator），因为它们并没有告诉你如何定位任何内容，它们只是为那个命名空间提供一个全球唯一的标识符，并不要求作为 XML 文档的目标命名空间的 URI 获取任何内容。我们将在 9.2.1 节看到另一个非 URL 的 URI 实例。

4. TCP 连接

HTTP 的最初版本(1.0）分别为每一个从服务器接获取数据项建立一个单独的 TCP 连接。不难看到这是一种多么低效的机制：即使客户想做的只是检验是否有一个网页的最新拷贝，也不得不在客户端和服务器之间交换建立连接和拆除连接的消息。所以，接收一个包括一些文本和 12 个图标或其他小图像的网页将导致建立和关闭 13 个单独的 TCP 连接。图 9-4 显示了获取只有一个嵌入对象的页面的事件序列。粗线表示 TCP 消息，而细线表示 HTTP 请求和响应。（有些 TCP ACK 没有显示出来，以防止图变得混乱。）你能看到，两个往返时间用于建立 TCP 连接，而另外两个（至少）用于获取页面和图像。除了延迟的影响，服务器上也需要用于处理额外 TCP 连接建立和终止的处理开销。

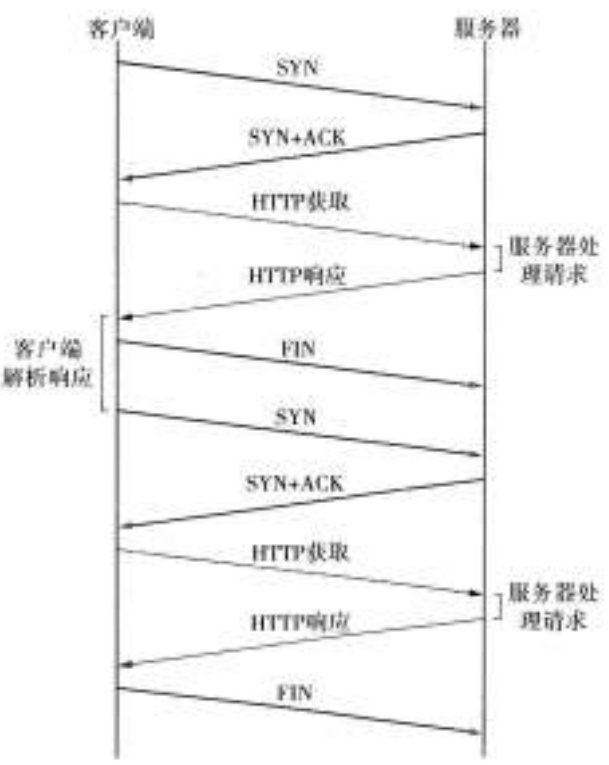


图 9-4 HTTP 1.0 的行为

为了改变这种情况，HTTP 版本 1.1 引入了持久连接（persistent connection），即客户端和服务器能够在同一个

TCP 连接上交换多个请求/响应消息。持久连接许多优点。第一，它明显地消除了连接建立的开销，从而降低因额外的 TCP 分组所引起的服务器负载和网络负载以及用户可察觉的延迟。第二，由于用户可在一个 TCP 连接上传送多个请求消息，所以 TCP 拥塞窗口机制也能更有效地工作，这是因为它不需要为每个页面经历慢启动阶段。图 9-5 显示了图 9-4 中的事务在连接已经打开（这可能是由于前面访问过同一台服务器）的情况下使用一个持久连接。

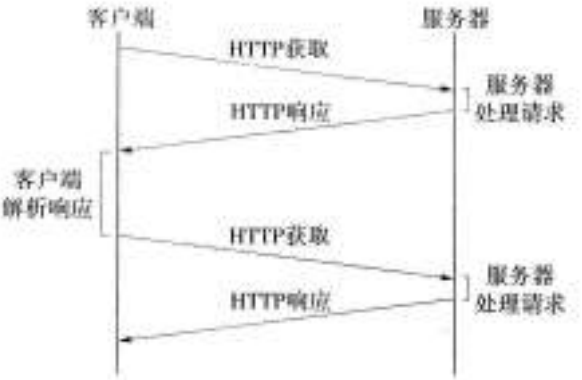


图 9-5　HTTP 1.1 的持久连接行为

然而，持久连接并不是无代价的，问题是客户端和服务器都不一定知道一个特定 TCP 连接需要保持多长时间。这对于服务器来说更为关键，它可能被要求为上千个客户保持打开的连接。解决方案是服务器必须设一个时间间隔，并关闭那些在一定时间内没有收到请求消息的连接。另外，不管是客户端还是服务器，都必须观察另一端是否已选择了关闭连接，而且它们必须使用该信息作为关闭本端连接的信号。（回忆一下，两端必须都关闭了一个 TCP 连接之后，TCP 才会完全终结。）对这个问题的考虑增加了复杂性，这可能是没有一开始就使用持久连接的原因，但现在人们普遍认为持久连接的优点比缺点多。

5. 缓存

现在，因特网的一个最具活力的研究（和企业技术）领域是如何高效地缓存 Web 网页。使用缓存有许多好处。从客户的角度看，从邻近的缓存中得到并显示网页比从世界的另一端获取要快得多。从服务器的角度看，用缓存截取和满足请求将降低服务器的负载。

可在不同地方实现缓存。例如，用户浏览器可缓存近来访问过的页面，如果用户再次访问该网页，则简单地显示缓存的拷贝。另一个例子是网站可支持一种单一网站范围的缓存，它允许用户利用其他用户以前下载的网页。ISP 靠近因特网的中间，也能缓存页面。注意，在第二种情况下，网站内的用户很可能知道哪台计算机正在为网站缓存网页，他们可配置自己的浏览器直接连接到缓存主机。这种节点有时称为代理（proxy）。相反，连接到 ISP 的网站可能并不知道 ISP 正在缓存网页，这可能发生在来自各个网站的 HTTP 请求恰好通过一个公共 ISP 路由器的情况下。路由器快速查看流经的请求消息，并注意所请求网页的 URL。如果网页已在缓存中，则返回它；如果不在，将请求转发到服务器，并监视反方向而至的响应。当它到来时，路由器将保存它的拷贝以希望能满足以后的请求[⊖]。

无论将网页缓存在哪里，重要的是缓存网页的能力，因此 HTTP 的设计目标之一是使缓存更加容易。技巧是缓存需要确保不用过期版本的网页响应。例如，服务器为每个送回客户（或在服务器和客户间的缓存）的网页指定一个截止日期（Expires 首部字段）。缓存记住这一日期，并且知道在到期前每次请求都不需要再检查网页。到期后（或者如果没有设置该首部字段），缓存能使用 HEAD 或有条件的 GET 操作（有 If-Modified-Since 首

⊖ 这种缓存存在很多问题，从技术问题到管理问题。技术挑战的一个例子是当发给服务器的请求与发给客户的响应不使用相同的路由器序列时，非对称路径将产生的影响。

部行的 GET）去验证它有没有该网页的最新拷贝。更常见的情况是，在整个请求/响应链上，所有缓存机制必须遵守一组缓存指示（cache directives），这些指示说明一个文档是否能被缓存、可缓存多长时间、一个文档必须有多新，等等。我们将在 9.4.3 节讨论 CDN（CDN 是高效的分布式缓存）的相关问题。

9.1.3 Web 服务

截至目前，我们已经研究的大部分应用都包含人类与计算机之间的交互。例如，一个人用 Web 浏览器与一台服务器交互，交互是通过对来自用户的输入（如单击链接）给出响应而进行的。然而，对计算机到计算机的直接交互的需求越来越多。并且，就像前面讲到的应用需要协议一样，直接通信的应用程序之间也需要协议。在本节中，我们将着眼于构建大量应用到应用协议所面临的挑战以及已提出的一些解决方案。

使应用程序之间直接通信的主要推动力来自商业领域。历史上，企业（商业或其他组织）之间的交互曾包含一些手工步骤，如填订单或打电话确认某个商品是否有库存。即使是在一个企业内，因独立开发而造成不能直接交互的软件系统之间包含手工步骤也是很常见的。这种手工交互正渐渐地被应用之间的直接交互所取代。企业 A 的一个订单应用可能会发送一条消息到企业 B 的配货应用，配货应用会立即给出回应来说明是否能够满足订单。如果 B 无法满足该订单，A 的应用会立即从另一个提供商那里订货或请多个提供商竞标。

这里给出一个有关我们正在讨论话题的简单例子。假设你从一个在线零售商，如 Amazon.com，买一本书。一旦你的书被寄出以后，Amazon 会用电子邮件给你发送一个跟踪号码，然后你就会转向航运公司的网站（也许是 http://www.fedex.com）并跟踪包裹。然而，你也可以直接通过 Amazon.com 网站来跟踪包裹。为了实现这个功能，Amazon 必须用联邦快递（Fedex）能够理解的格式向 Fedex 发送查询，然后解释结果，并在一个可能包含订单信息的网页中显示该结果。在用户能够立即在 Amazon.com 网页上得到有关订单的所有信息的背后是 Amazon 与 Fedex 必须有一个用于交换跟踪包裹所需要信息的协议——称为包裹跟踪协议（Package Tracking Protocol）。应该清楚的一点是对于很多潜在的这类协议，我们最好有一些工具来简化定义及构建它们的任务。

网络应用甚至跨越组织边界的应用并不是新事物——我们在前一节中就已经看到了一些例子。这个问题的一个新着眼点是规模，不是网络大小的规模，而是不同网络应用的种类数的规模，诸如电子邮件和文件传输的传统应用的协议规范和实现一般是由网络专家小组开发的。为了迅速开发大量潜在的网络应用，有必要推出一些能够简化和自动化应用协议设计和实现的技术。

人们已经提出了两个体系结构作为该问题的解决方案。这两个体系结构都称为 Web 服务（Web Services），这个名字取自一个术语，这个术语指的是为客户应用提供远程访问服务从而构成网络应用的单个应用[⊖]。用于区分这两种 Web 服务体系结构的两个非正式的简写是 SOAP 和 REST（就像在“SOAP 与 REST 的争论”中一样）。我们将简要讨论那些术语的技术含义。

SOAP 体系结构解决该问题的方法是至少在理论上使得为每一个网络应用生成定制协

⊖ 不幸的是，Web 服务听起来太通用了，以至于很多人会错误地认为它包括与 Web 相关的任何服务。

议成为可能。这个方法的关键组件包括一个协议规范框架、根据协议规范自动产生协议实现的软件工具，以及可以在协议间重用的模块化的部分规范。

REST 体系结构解决该问题的方式是将单个 Web 服务当作万维网资源——用 URI 标识，通过 HTTP 访问。从本质上讲，REST 体系结构只是 Web 体系结构。Web 体系结构的长处包括稳定性和可证明的可扩展性（在网络大小的意义上）。HTTP 不能很好地适应调用远程服务的一般过程性模式或面向操作的模式，这可以认为是它的一个弱点。然而 REST 倡导丰富的服务能够通过使用 HTTP 更适用的更面向数据的模式或文档传递模式进行发布。

虽然目前对这两种结构的应用都比较活跃，但它们还是太新，还没有现实使用的实验数据。二者竞争的方式能够教给我们一些意义深远的关于网络应用的知识。一种体系结构也许会占据主导地位，或者它们以某种方式融合，或者我们可能发现一种体系结构更适合某些类的应用而另一种体系结构则更适合其他类的应用。

1. 定制应用协议（WSDL、SOAP）

非正式地称为 SOAP 的体系结构基于 Web 服务描述语言（Web Services Description Language，WSDL）和 SOAP[⊖]。这两个标准都是由万维网联盟（World Wide Web Consortium，W3C）发布的。这就是人们通常在没有任何限定语的情况下使用术语 Web 服务时所表示的体系结构。这些标准现在依然在迅速发展，我们在这里的讨论实际上是一个快照。

WSDL 和 SOAP 分别是用于说明和实现应用协议和传输协议的框架。它们通常一起使用，虽然并非必须如此。WSDL 用于说明与应用相关的细节，如支持什么操作、用于调用或响应那些操作的应用数据的格式，以及操作是否包含响应。SOAP 的作用是使定义具有所期望的（诸如可靠性和安全性方面）特征语义的传输协议变得简单。

WSDL 和 SOAP 主要是由协议规范语言组成。它们都基于 XML（见 7.1.3 节），并兼顾对诸如桩编译器和目录服务等的使用。在有很多定制协议的环境中，对自动产生实现的支持是很关键的，以避免手动实现每一个协议。支持软件一般采用第三方厂商开发的工具包和应用服务器的形式，这就允许单个 Web 服务的开发者能够更多地专注于他们需要解决的业务问题（如跟踪客户所购买的软件包）。

2. 定义应用协议

WSDL 选择应用协议的过程操作模型。抽象的 Web 服务接口由一组命名的操作组成，每一个操作代表客户与 Web 服务之间的简单交互。操作类似于 RPC 系统中的远程调用过程。来自 W3C 的 WSDL 入门中的一个例子是旅馆预订 Web 服务，包括 CheckAvailability 和 MakeReservation 两个操作。

每一个操作描述了给出消息传输序列的消息交换模式（message exchange pattern，MEP），包括当有错误打断了消息流后要发送的默认消息。已经预定义了几个 MEP，也可以定义新的用户 MEP，但在实际中好像只有两个 MEP 正在使用：In-Only（从客户发到服务器的一条消息）和 In-Out（来自客户的一个请求和来自服务器的响应）。这些模式应该非常类似，但也暗示支持 MEP 灵活性的代价可能超过了其带来的好处。

⊖ 虽然名字 SOAP 最初是一个首字母缩写词，但它不再正式代表任何东西。

MEP 是具有占位符而不是特定的消息类型或格式的模板，因此操作定义中的一部分要指明用哪些消息格式映射到模板中的占位符。消息格式的定义不像我们已经讨论的协议一样定义在比特级，而是定义为使用 XML 大纲（见 7.1.3 节）的抽象数据模型。XML 大纲提供了一个基本数据类型集以及定义复合数据类型的方法。符合 XML 大纲定义格式（抽象数据模型）的数据能够用 XML 具体表示，或者使用另一种表示方法，如“二进制”表示 Fast Infoset。

WSDL 巧妙地将能够抽象描述的协议部分（操作、MEP、抽象消息格式）与必须具体定义的部分分开。WSDL 的具体部分定义了一个低层协议、MEP 如何映射到协议，以及消息在线上传输时使用什么比特级的表示。这部分协议称为绑定（binding），虽然将其描述为一个实现或一个到实现的映射更好。WSDL 已经为 HTTP 和基于 SOAP 的协议预定义了绑定，也包括一些参数，以便协议设计者可调节到协议的映射。WSDL 有一个用于定义新绑定的框架，但 SOAP 协议占主导地位。

WSDL 在减轻定义大量协议的工作量方面的关键是复用基本的规范模块。Web 服务的 WSDL 规范可能由多个 WSDL 文档组成，单个 WSDL 文档可用于其他 Web 服务规范。这种模块化使得开发规范更简单，并且能够很容易地确认两个规范是否有相同的组件（例如，它们可以由相同的工具来支持）以及那些组件是否真的相同。这种模块化以及 WSDL 的默认规则也有助于避免规范对于协议设计者来说太冗长。

对于开发过中等规模软件的人来说，应该很熟悉 WSDL 模块化。一个 WSDL 文档没有必要是一个完整的规范，例如，它可以定义一个消息格式或者一个特定 WSDL 接口的绑定。部分规范通过 XML 命名空间（见 7.1.3 节）被唯一标识，每一个 WSDL 文档指明一个目标命名空间（target namespace）的 URI，文档中的任何新定义都在那个命名空间上下文中命名。一个 WSDL 文档可以通过包含（including）第二个有相同目标命名空间的文档或导入（importing）有不同目标命名空间的文档来集成其中的组件。

3. 定义传输协议

虽然 SOAP 经常称为一个协议，但最好将其看成一个协议族的基础或者一个定义协议的框架。正如 SOAP 1.2 规范中的解释，“SOAP 提供了一个简单的消息框架，其核心功能是提供可伸缩性。”SOAP 使用许多与 WSDL 相同的策略，包括用 XML 大纲定义消息格式、到低层协议的绑定、MEP 和用 XML 命名空间标识的可重用规范组件。

SOAP 用于定义具有支持特定应用协议所需特性的传输协议。SOAP 的目标是使得通过可复用组件定义很多协议变得可行。每个组件捕获实现一个特定特性时的首部信息和逻辑。为了定义一个具有特定特性集的协议，只需要组合相应的组件。当然，也没有那么简单。让我们更进一步看看 SOAP 的这个方面。

SOAP 1.2 引入了特性（feature）抽象，其描述如下：SOAP 特性是 SOAP 消息框架的扩展。SOAP 并没限制潜在特性的范围，特性的例子包括“可靠性”（reliability）、“安全性”（security）、“相互关系”（correlation）、“路由”（routing）和消息交换模式（message exchange patterns，MEP），诸如请求/响应、单向和对等会话。一个 SOAP 特性规范必须包括：

- 一个标识该特性的 URI。
- 抽象描述的状态信息和处理过程，每个 SOAP 节点实现该特性时需要这些信息。

- 传递到下一个节点的信息。
- 如果该特性是 MEP，那么将交换消息的生命周期和临时关系/因果关系（例如，响应跟在请求之后，并且被发送到请求的发起者）。

注意，这种协议特性概念的形式化是非常低层的，它几乎就是一个设计。

对给定的一个特性集合，有两种策略可定义实现这些特性的 SOAP 协议。一种是分层：以派生特性的方式将 SOAP 绑定到低层协议，例如，我们通过将 SOAP 绑定到 HTTP 来获得一个请求/响应协议，将 SOAP 请求放在 HTTP 请求中，将 SOAP 响应放在 HTTP 响应中。因为这是一个非常常见的例子，因此 SOAP 已经预定义为与 HTTP 绑定，新的绑定可以通过使用 SOAP 协议绑定框架来定义。

另一种更灵活的实现特性的方式包含首部块（header block）。SOAP 消息由信封和主体组成，信封包含由首部块组成的首部，主体包含发给最终接收者的载荷。该消息的结构在图 9-6 中进行了说明。



图 9-6　SOAP 消息的结构

现在，某些首部信息对应特定的特性应该是一个熟悉的说法。数字签名用于实现认证，序列号用于可靠性，校验和用于检测消息错误。SOAP 首部块的目的是封装对应特定特性的首部信息。这种对应不总是一对一的，因为一个特性可能涉及多个首部块，或者一个首部块用于多个特性。SOAP 模块（module）是针对一个或多个首部块的语法和语义的规范。每一个模块用于提供一个或多个特性，并且必须声明它所实现的特性。

SOAP 模块的深层目标是能够通过简单地包含每个相关的模块规范来组合一个具有一组特性的协议。如果协议至多需要一个语义和认证，那么就把对应的模块包含在规范中。这展示了一种协议服务模块化的巧妙方法，是我们在整本书中看到的协议分层方法的替代方法。它有点像以结构化的方法将一系列协议层扁平化为一个协议。至于 SOAP 1.2 中引入的 SOAP 特性和模块在实践中工作的如何还有待考察。这种方案的一个主要弱点是模块之间可能会互相干扰。模块规范需要指明所有与其他 SOAP 模块的已知（known）交互，显然这对于解决问题没有太大帮助。另一方面，提供了最重要特性的核心特性及模块集合可能足够小而被熟知和充分理解。

4. Web 服务协议的标准化

WSDL 和 SOAP 不是协议，它们是用于规范（specifying）协议的标准。对于相互协作来实现 Web 服务的不同企业，只是在使用 WSDL 和 SOAP 来定义它们的协议这一点上达成一致还是不够的，它们必须在具体协议上达成一致——标准化。例如，继续讨论在本节开头提到的简单包裹跟踪示例，你可以想象在线零售商和航运公司可能希望标准化一个它们用来交换信息的协议。标准化对于工具支持和互操作都是很关键的。然而，这个体系结构中的不同网络应用至少在消息格式和所使用的操作方面存在差异。

对于标准化与定制之间的矛盾，正在通过建立称为轮廓（profile）的部分标准来解决。轮廓是一个指南集合，用于在定义协议时缩小或限制所参考的 WSDL、SOAP 和其他标准的选择范围。它们同时也会解决那些标准之间的含混和差距。在实际中，轮廓通常标准化正在出现的事实标准。

应用最广和最多的轮廓是 WS-I 基本轮廓（WS-I Basic Profile）。该轮廓是由 Web 服务互操作组织（Web Services Interoperability Organization，WS-I）提出的，它是一个行业联盟，而 WSDL 和 SOAP 是由万维网联盟（World Wide Web Consortium，W3C）定义的。基本轮廓解决了定义 Web 服务时所面临的最基本的选择。最值得注意的是，它要求 WSDL 只能与 SOAP 绑定，SOAP 只能与 HTTP 绑定，并且使用 HTTP 的 POST 方法。它还指定了必须使用的 WSDL 和 SOAP 的版本。

WS-I 基本安全轮廓（WS-I Basic Security Profile）通过规定如何使用 SSL/TLS 层以及要求遵循 WS 安全（WS-Security，Web 服务安全）来为基本轮廓增加安全限制。WS 安全说明了如何使用各种不同的现有技术，如 X.509 公钥证书（见 8.2.1 节）和 Kerberos（见 8.3.3 节）为 SOAP 协议提供安全特性。

WS 安全只是一套由行业联盟结构化信息标准发展组织（Organization for the Advancement of Structured Information Standards，OASIS）建立的不断扩大的 SOAP 级标准中的第一个。这些标准统称为 WS-*，包括 WS-Reliability、WS-ReliableMessaging、WS-Coordination 和 WS-AtomicTransaction。

5. 一个通用的应用协议（REST）

WSDL/SOAP Web 服务体系结构的基本假设是，集成网络应用的最好方法是使用为每一个应用定制的协议。这个体系结构的设计目标是使得定义和实现那些协议成为可能。与之相比，REST Web 服务体系结构的基本假设是，集成网络应用的最好方法是应用万维网体系架构的模型（见 9.1.2 节）。这个模型是由 Web 体系架构师 Roy Fielding 发表的，称为表述性状态转移（REpresentational State Transfer，REST）。没有必要为 Web 服务定义一个新的 REST 结构——现有的结构是适用的，虽然可能需要做一些扩展。在 Web 体系结构中，单个 Web 服务被看作用 URI 标识并通过 HTTP 访问的资源——带有单一通用寻址方案的单一通用应用协议。

WSDL 有用户定义的操作，而 HTTP 只有一个很小的方法集，如 GET 和 POST（见表 9-1）。那么这些简单方法如何为丰富的 Web 服务提供接口？通过使用 REST 模型，复杂性从协议转移到载荷。载荷是资源抽象状态的表示。例如，GET 会返回资源当前状态的表示，POST 则会发送资源的一个期望状态的表示。

资源状态的表示是抽象的，它没有必要模仿一个特定的 Web 服务实例实际上是如何实现该资源的。没有必要在每一条消息中传递完整的消息状态。可以通过只传递感兴趣的（如，只是正在被修改的部分）状态来减小消息的大小。并且，因为 Web 服务与其他 Web 资源共享一个协议和地址空间，所以部分状态能够通过引用（通过 URI）传递，即使这些状态属于其他 Web 服务。

这种方法最好被总结为与过程模式相对的面向数据或文档的传递模式。在这种体系结构中定义应用协议包括定义文档结构（即状态表示）。XML 和更轻载的 JavaScript 对象表示法（JavaScript Object Notation，JSON）是最常用的状态表示语言（见 7.1 节）。互操作依赖于 Web 服务与其客户对状态表示的协商。当然，在 SOAP 中也是这样的，Web 服务和其客户必须在载荷格式上达成一致。不同的是，在 SOAP 结构中，互操作性还依赖于对协议的协商。在 REST 结构中，协议总是 HTTP，因此互操作问题的源头被消除了。

REST 的卖点之一是它利用了用于支持 Web 的基础设施。例如，Web 代理能够实施

安全机制或缓存信息。现有的内容分发网络（CDN）可用于支持 REST 应用。

与 WSDL/SOAP 相比，Web 已经历经了一段时间来使得标准变得稳定并证明它具有良好的可扩展性。它还以 SSL/TLS 的形式提供了安全性。Web 和 REST 在可发展性上也有优势。虽然 WSDL 和 SOAP 框架（framework）在定义协议时能够添加新特性，且在绑定方面高度灵活，但一旦协议被定义，这种灵活性就不相关了。例如，HTTP 的标准化协议在设计时就要以后向兼容的方式提供可扩展性。HTTP 本身的可扩展性以首部、新方法和新内容类型的方式提供。使用 WSDL/SOAP 的协议设计者需要在每一个定制协议的设计中提供可扩展性。当然，REST 体系结构中的状态表示设计者还要在设计中提供可发展性。

WSDL/SOAP 可能具有优势的领域是修改或包装之前的传统应用，使之符合 Web 服务。这是非常重要的一点，因为至少在不久的将来，大部分 Web 服务还是基于传统应用的。这些应用通常具有过程化的接口，更容易映射为 WSDL 操作而不是 REST 状态。REST 与 WSDL/SOAP 的竞争可能与为单个 Web 服务修改 REST 风格接口的难易息息相关。我们可能会发现有些 Web 服务用 WSDL/SOAP 更好，而其他的则用 REST 更好。

在线零售商 Amazon.com 恰好是 Web 服务的一个早期（2002 年）采纳者。有趣的是，Amazon 通过这两种 Web 服务体系结构来提供系统的公开访问，并且这几年中有大约 80%的使用是通过 REST 接口。当然，这只是一个可能很好地反映了特定于 Amazon 因素的数据点[⊖]。

9.2　多媒体应用

正如前一节的传统应用一样，像音频和视频会议那样的多媒体应用需要自己的协议。在设计多媒体应用协议方面的许多最初经验来自 MBone 工具集，诸如开发用于 MBone 的 vat 和 vic 之类的应用。MBone 是支持 IP 多播来实现多方会议的覆盖网络。开始时，每个应用实现自己的协议（或协议族），但显然，许多多媒体应用有共同的需求。这最终导致需要开发用于多媒体应用的一些通用协议。

我们已经看到了很多多媒体应用使用的协议。实时传输协议（Real-time Transport Protocol，RTP）（在 5.4 节中描述）提供了多媒体应用的共有功能，如传递定时信息、标识应用的编码机制和媒体类型。

资源预留协议（Resource Reservation Protocol，RSVP）（参见 6.5.2 节）用于在网络上请求分配资源，以便能够为一个应用提供所期望的服务质量（QoS）。我们将在9.2.2 节看到资源分配如何与多媒体应用的其他方面交互。

除了用于多媒体传输和资源分配的这些协议，许多多媒体应用还需要会话控制（session control）协议。例如，假设我们希望打一个跨越因特网的电话（Voice over IP，VoIP）。我们需要某种机制去通报本次呼叫的预定接收方，即我们想要与她通话，比如，通过向某个多媒体设备送出一个消息使之发出铃声。我们也希望支持类似于呼叫转移和三方呼叫等特性。会话初始化协议（Session Initiation Protocol，SIP）和 H.323 就是解决会话控制问题的协议的例子，我们将分析这些协议作为讨论多媒体应用的开始。

⊖ 参见“实验十四”和“实验十五”。

9.2.1 会话控制和呼叫控制（SDP、SIP、H.323）

为了理解会话控制的相关问题，考虑以下问题。假设你想在某个时间为广大参与者举办一次视频会议，也许你已决定使用 MPEG-2 标准为视频流编码，使用多播 IP 地址 224.1.1.1 传输数据，并且在端口号为 4000 的 UDP 之上使用 RTP 发送信息。你将如何把这些信息通知所有预期的与会者呢？一种方法是将所有信息放在电子邮件中送出，但理想的情况是有一种传播这种信息的标准格式和协议。IETF 已经定义了用于此目的的协议，包括

- 会话描述协议（Session Description Protocol，SDP）。
- 会话发布协议（Session Announcement Protocol，SAP）。
- 会话初始化协议（Session Initiation Protocol，SIP）。
- 简单会议控制协议（Simple Conference Control Protocol，SCCP）。

你可能认为这是为看似简单的任务准备的许多协议，但是这个问题有多个方面和不同的情况需要解决。例如，MBone 上将举行的某次会议会话（这可以用 SDP 和 SAP 来完成）和试图在一个特定的时间与某个用户发起因特网电话呼叫（这可以用 SDP 和 SIP 来完成）之间存在差别。对于前者，一旦你将所有会话信息以标准格式送到一个众所周知的多播地址，就可以认为你的工作完成了。而对于后者，你需要定位一个或多个用户，给他们发一条消息来通知你想通话的愿望（类似于拨通他们的电话），并且可能要在各方中协商一个合适的音频编码。我们首先看 SDP，许多应用都使用它，然后再看 SIP，它正渐渐被很多交互式应用所广泛使用，比如因特网电话。

1. 会话描述协议（SDP）

会话描述协议（SDP）是一个相当通用的协议，能用于多种场景，通常与一个或多个其他协议（如 SIP）联合使用，它传达如下信息：

- 会话的名字和目的。
- 会话的开始和结束时间。
- 会话包含的媒体类型（如音频、视频）。
- 接收会话所需的细节信息（如数据将被送到的多播地址、所用到的传输协议、端口号、编码方案）。

SDP 使用一系列 ASCII 格式的文本行来提供这些信息，每一行的形式为〈type〉=〈value〉（〈类型〉=〈值〉）。以下用一条示例消息来解释 SDP 的要点。

```
v=0
o=larry 2890844526 2890842807 IN IP4 128.112.136.10
s=Networking 101
i=A class on computer networking
u=http://www.cs.princeton.edu/
e=larry@cs.princeton.edu
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
```

注意，SDP 像 HTML 一样很容易阅读，但其严格的格式化规则使计算机能清晰地解释数据。例如，SDP 规范定义所有允许出现的信息“类型”、信息呈现次序以及每种被定

义的类型的格式和保留字。

首先注意到每种信息“类型”由单个字符标识。例如，行 v=0 告诉我们“版本”号的值是 0（就是说，这个消息的格式符合 SDP 版本 0 规范）。下一行提供会话的“源”，它含有足够多的信息以唯一标识该会话。其中，larry 是会话创建者的用户名，128.112.136.10 是他的计算机的 IP 地址。跟随在 larry 后面的数字是会话标识符，它对于那台机器来说是唯一的。接下来是 SDP 通知的“版本”号，如果会话信息被后来的消息更新，版本号将增加。

接下来的三行（s、i 和 u）——会话的名字、会话描述和会话的统一资源标识符（URI，见 9.1.2 节），这些信息有助于用户决定是否参与本次会话。这样的信息将显示在用户的会话目录工具界面上，以表明现在和即将来临的已经用 SDP 通知的事件。下一行（e =...）含有某个与本次会话有关的人的电子邮件地址。图 9-7 显示了一种称为 sdr 的会话目录工具的屏幕画面以及在截取画面时已经通知的若干会话的描述。



图 9-7　一个会话目录工具显示的从 SDP 消息中摘录的信息

下面我们将进入使一个应用程序能够参与会话的技术细节。以 c = ... 开始的行提供这个会话的数据将要发往的 IP 多播地址，用户需要加入这个多播组才能接收会话。接下来我们看到的是会话的开始和结束时间（根据网络时间协议 NTP 被编码为整数）。最后得到有关本次会话的媒体信息。本次会话有三种可用的媒体类型——音频、视频以及称为“wb”的共享白板应用。对每种媒体类型都有如下格式的信息行：

```
m=<media> <port> <transport> <format>
```

媒体类型是自解释的，而端口号在多种情况下是指 UDP 端口。当我们看“传输”字段时，能看出 wb 应用直接运行在 UDP 之上，而音频和视频使用“RTP/AVP”传输。这意味着它们运行在 RTP 上并且使用称为 AVP 的应用轮廓（application profile）（见 5.4 节中的定义）。这个轮廓为音频和视频定义一些不同的编码方案，这里我们能看到，音频使用编码 0（使用一种 8kHz 的采样频率和每样本 8 比特的编码），视频使用编码 31，代表 H.261 编码方案。这些编码方案的“神奇的数字”在定义 AVP 轮廓的 RFC 中定义。在 SDP 中也有可能描述非标准的编码方案。

最后我们来看“wb”媒体类型的描述。这种数据的所有编码信息都是特定于 wb 应用的，因此在格式字段提供应用的名字就足够了。这类似于将应用 application/wb 放入一条

MIME 消息中。

现在我们已经知道了如何描述会话，可以继续关注如何初始化会话。一种办法是使用 SDP 通过向一个众所周知的多播地址送出 SDP 消息以通告多媒体会议。显示在图 9-7 中的会话目录工具负责将其加入多播组，并且从接收到的 SDP 消息中收集和显示信息。SDP 也用于传输 IP 娱乐视频（通常称为 IPTV），用于提供每个电视频道中视频内容的相关信息。

SDP 也在与会话初始化协议（SIP）的合作中扮演重要角色。由于 IP 语音（即在 IP 网络上支持类似电话的应用）和基于 IP 的视频会议的广泛普及，因此 SIP 现在已经成为 IP 互联网协议族的重要组成部分。

2. SIP

SIP 是一个与 HTTP 有点类似的应用层协议，基于相似的请求/响应模式。然而，记住它是为不同种类的应用设计的，因而它提供与 HTTP 截然不同的能力。SIP 所提供的能力可以分成五类：

- 用户定位：确定与特定用户通信的恰当设备。
- 用户可用性：确定用户是否自愿或能够参加一个特定的通信会话。
- 用户能力：确定采用哪种媒体和编码方案。
- 会话建立：建立会话参数，诸如通信各方所使用的端口号。
- 会话管理：包括传递会话（如实现“呼叫转发”）和修改会话参数等功能。

这些功能中的大部分都很容易理解，但定位问题需要进一步讨论。SIP 与 HTTP 之间的一个重要差别是 SIP 主要用于人与人的通信。因此，重要的是能定位独立的用户（user）而不是计算机。与电子邮件系统不同，只定位用户今后某个日期访问并转储消息的服务器是不够的。如果想实现实时通信，我们就需要知道现在用户在什么地方。这一功能更复杂的方面在于用户可以选用一系列不同设备进行通信的事实，例如，在办公室时使用桌面 PC 而旅行时使用手持设备。多种设备可以同时参与并且能力上差别巨大（如一个文字数字型的寻呼机和一个基于 PC 的视频“电话”）。理想情况是，其他用户可以在任何时间定位适当的设备并与之通信。此外，用户必须还能控制何时、何地和接收何人的呼叫。

为了使用户对呼叫行使适当级别的控制，SIP 引入代理的概念。SIP 代理可以看作用户的联系点，用户可向它送出通信初始化请求。这些代理代表呼叫者完成各种功能。我们通过一个例子来看看代理如何以最佳的方式工作。

考虑图 9-8 中的两个用户。首先需要注意的是每个用户有一个 user@domain 格式的名字，很像电子邮件地址。当用户 Bruce 想要初始化与 Larry 的会话时，他向他所在域的本地代理 cisco.com 发送他的初始化 SIP 消息。这条初始化消息含有一个 SIP URI——格式如下的统一资源标识符：

```
SIP:larry@princeton.edu
```

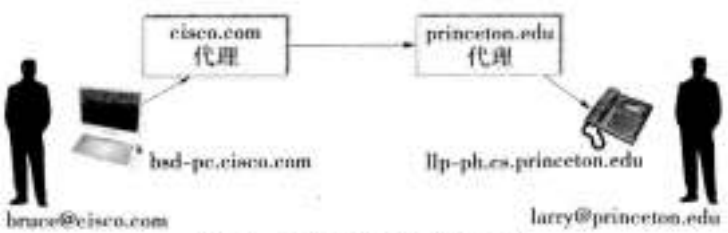


图 9-8 通过 SIP 代理建立通信

SIP URI 提供了用户的完整标识，但（不像 URL）不提供他的位置，因为位置可能随时变化。我们将马上看到如何确定用户的位置。

在接收到来自 Bruce 的初始化消息的基础上，代理 cisco.com 查看 SIP URI 并且推断这一消息应该被送到代理 princeton.edu。现在，我们假设代理 princeton.edu 已经访问了某些数据库，得到了从名字 larry@princeton.edu 到目前 Larry 希望用于接收消息的一台或多台设备的 IP 地址的映射。代理能将消息转发到 Larry 选择的设备上。发送消息到多台设备上称为分支（forking）并且可以并行或串行方式完成（例如，如果他的座机不应答就发送到他的手机上）。

从 Bruce 到 Larry 的初始化消息很可能是一条 SIP invite 消息，它看上去与下述形式相似：

```
INVITE sip:larry@princeton.edu SIP/2.0
Via: SIP/2.0/UDP bsd-pc.cisco.com;branch=z9hG4bK433yte4
To: Larry <sip:larry@princeton.edu>
From: Bruce <sip:bruce@cisco.com>;tag=55123
Call-ID: xy745jj210re3@bsd-pc.cisco.com
CSeq: 271828 INVITE
Contact: <sip:bruce@bsd-pc.cisco.com>
Content-Type: application/sdp
Content-Length: 142
```

第一行标识的内容包括：将要执行的功能的类型（invite）；执行功能所需的资源，即被调用方（sip：larry@princeton.edu）；以及协议的版本（2.0）。其后的首部行可能看上去眼熟，因为它们与电子邮件消息中的首部行有类似之处。SIP 定义了大量的首部行字段，这里只描述其中的一部分。注意本例的 Via：首部标识了发送这一消息的设备。首部 Content-Type：和 Content-Length：描述首部之后的消息内容，与 MIME 编码的电子邮件消息相同。在本例中，内容是一条会话描述协议（SDP）消息。这条消息将描述诸如 Bruce 与 Larry 想要交换的媒体类型（音频、视频等）及他所支持的其他编解码类型等会话属性。注意 SIP 的 Content-Type：字段提供使用任何协议来完成任务的能力，尽管 SDP 是使用最普遍的协议。

再来看这个例子，当 invite 消息到达代理 cisco.com 时，代理不仅向 princeton.edu 转发消息，它也响应 invite 的发出者。像在 HTTP 中一样，所有响应都有一个响应代码，并且代码的组织也类似于 HTTP，如表 9-2 所示。在图 9-9 中，我们能看到一系列 SIP 消息和响应。

在图 9-9 中的第一个响应消息是临时响应 100 trying，它表明消息已经被呼叫代理无差错接收。一旦 invite 被传送给 Larry 的电话机，它将提醒 Larry 并且用 180 ringing 消息进行响应。这一消息到达 Bruce 的计算机后产生一个“振铃音”。假设 Larry 想要并且也能与 Bruce 通信，他就摘下电话，这引起消息 200 OK 被送出。Bruce 的计算机以 ACK 响应，并且这时媒体（如一个 RTP 封装的音频流）开始在两个参与者之间流动。注意这时参与者知道彼此的地址，所以 ACK 能绕过代理而直接被发送。此时在呼叫中将不再涉及代理。注意，媒体一般将采取与原始信令消息不同的路径穿越网络。此外，即使一两个代理在此时瘫痪，呼叫仍能正常维持。最后，当一个参与者希望终止会话时，它送出 BYE 消息，在正常情况下引发 200 OK 响应。

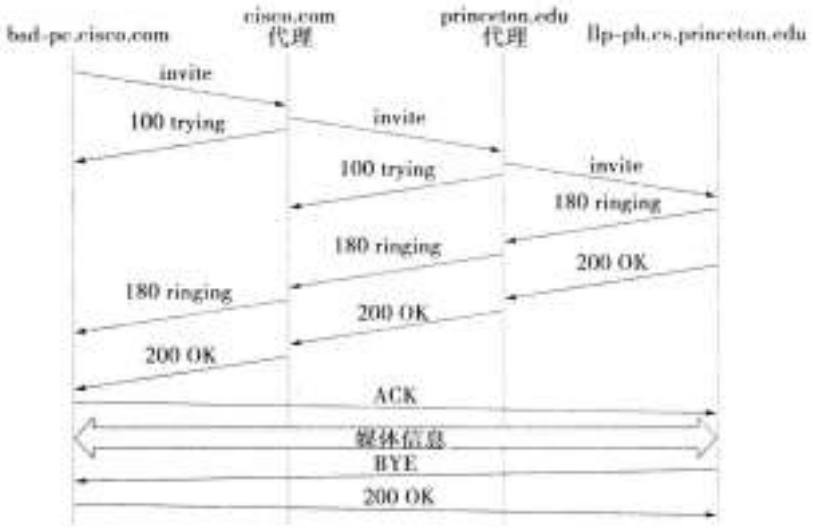


图 9-9 一个基本 SIP 会话的消息流

这里我们掩盖了一些细节。一个是会话特性的协商。也许 Bruce 本来想用音频和视频通信，但 Larry 的电话机仅支持音频。因而 Larry 的电话机在考虑了 Bruce 的 invite 中建议的选项后在 200 OK 中送出一个描述 Larry 和他的设备能接受的会话属性。通过这种方法，在媒体流开始前，对彼此可接受的会话参数达成一致。

我们掩盖的另一个大问题是定位 Larry 现在的设备。首先，Bruce 的计算机必须把它的 invite 送到代理 cisco. com。这可能是计算机内已配置的信息，或从 DHCP 得知。然后代理 cisco. com 必须找到代理 princeton. edu。这可以用某种特殊类型的 DNS 查询并返回 princeton. edu 域的 SIP 代理的 IP 地址。（我们在 9.3.1 节讨论了 DNS 如何实现这一点。）最后代理 princeton. edu 必须找到联系 Larry 所使用的设备。典型的做法是，代理服务器访问用一些方法建立的位置数据库。手工配置是一种选择，但更灵活的选择是使用 SIP 的注册（registration）功能。

用户可通过发送一条 SIP register 消息到他域中的“注册处”注册一个位置服务。这条消息产生一个记录地址（address of record）与联系地址（contact address）之间的绑定。记录地址看上去是像 SIP URI 这样的知名地址（如 sip：larry@princeton. edu），而联系地址是当前能够找到用户的地址（如 sip：larry@ llpph. cs. princeton. edu）。在我们的例子中，这种绑定正是代理 princeton. edu 所需要的。

注意一个用户可以在多个位置注册并且多个用户可以注册在同一台设备上。例如，你能想象一组人步入装备一部 IP 电话的会议室，并且所有人都注册到这部电话上，以使他们能在电话上接收呼叫。

SIP 是一个非常丰富和灵活的协议，它能支持大量不同的复杂呼叫情况，也能支持那些与电话几乎无关的应用程序。例如，SIP 支持将呼叫转移到“音乐保持”或语音邮件服务器的操作。如何将其用于像即时通信那样的应用也是显而易见的。在写本书时，针对这些应用的 SIP 扩展的标准化工作正在进行。

3. H. 323

国际电信联盟（ITU）在呼叫控制领域也非常活跃，因为它和电话业相关，因此这并

不令人奇怪，因为ITU的传统领域就是电话业。幸运的是，在这方面IETF和ITU之间已经进行了大力协作，所以各种协议在某种程度上能互操作。ITU对分组网络上的多媒体通信的主要建议称为H.323，它捆绑了许多其他建议，包括呼叫控制协议H.225。H.323包含的全部建议长达好几百页，而且这个协议以其复杂性而著称，所以这里只可能给出简短的概述。

H.323作为因特网电话的一个协议而流行，这里我们考虑它的应用。一个发起或终止呼叫的设备称为H.323终端，它可以是运行因特网电话应用程序的工作站，也可能是特别设计的设备——例如一种具有网络软件和以太网端口的类似电话的设备。H.323终端能直接相互对话，但呼叫通常由一种叫作网闸（gatekeeper）的设备来协调。网闸完成一系列功能，比如在电话呼叫使用的各种地址格式之间转换以及控制在给定的时间内所发生的呼叫次数，从而限制H.323应用程序所用的带宽。H.323也包括网关（gateway）的概念，它将H.323网络连接到其他类型的网络。网关最常见的用法是将H.323网络连接到公共交换电话网（PSTN），如图9-10所示。这使得在一个计算机上运行H.323应用程序的用户能与在公共电话网上使用传统电话的人通话。由网闸完成的一个有用功能是帮助终端找出网关，可能要在一系列候选设备中找出与呼叫最终目的地最接近的网关，这对于传统电话数目远大于基于PC机的电话数目的世界，好处是显而易见的。当一个H.323终端呼叫一个传统电话的端点时，网关变成H.323呼叫的有效端点，并且负责对需要在电话网上传输的信令信息和媒体流进行适当的转换。

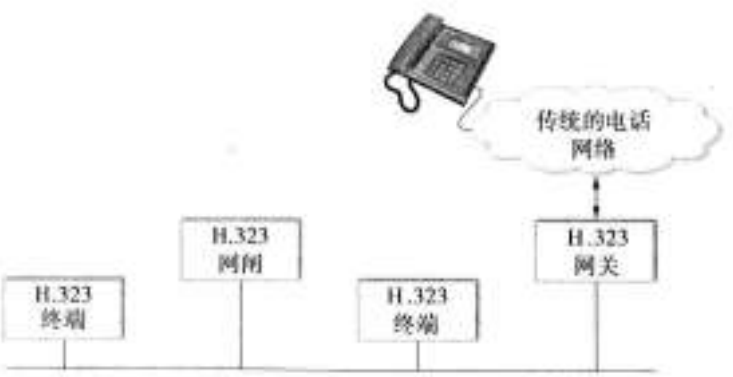


图9-10　H.323网络中的设备

H.323的一个重要部分是H.245协议。H.245用来协商呼叫的特性，有点类似于上面描述的SDP的用法。H.245消息可以列出许多它所支持的不同音频编码标准，并且呼叫的远端终点将答复一个它所支持的编码列表。然后两端选取一个双方都认可的编码标准。H.245还能发信令通知本次呼叫中将由RTP和实时控制协议（Real Time Control Protocol，RTCP）使用的用于媒体流（或多个流，例如一次呼叫可以包含音频和视频）的UDP端口号。一旦完成这些操作，呼叫就能继续，用RTP传送媒体流并用RTCP承载相应的控制信息。

9.2.2　多媒体应用的资源分配

正如我们已经看到的，像SIP和H.323这样的会话控制协议能够用于在多媒体应用中初始化和控制通信，而RTP为应用的数据流提供了传输层功能。使多媒体应用正常工

作的最后一步是确保在网络内分配了合适的资源，以便满足应用的服务质量要求。我们在第6章展示了多种资源分配方法。开发这些技术的推动力主要是对多媒体应用的支持。因此，应用如何利用底层网络的资源分配能力呢？

值得注意的是，很多多媒体应用在诸如公共因特网的“尽力而为”网络上成功地运行。大量商业IP语音服务（如Skype）证明了这样一个事实，当资源不充足时，你只需要关心资源分配——在当今的因特网上，资源匮乏是很常见的。

像RTCP（见5.4节）这样的协议能够通过为应用提供有关网络中所提供服务的质量信息来帮助尽力而为网络中的应用。回忆一下，RTCP在多媒体应用的两个参与者之间传送关于分组丢失率和延迟特性的信息。应用能够使用这些信息来改变其编码方案——例如，当带宽较小时改用低位率的编码。注意，虽然在分组丢失率高时，改变成发送附加的冗余信息的编码方式很诱人，但这并不能满足要求，它类似于在出现丢失分组时增加TCP的窗口大小，这与所期望的避免拥塞崩溃完全相反。

正如在6.5.3节讨论的那样，区分服务（DiffServ）可用于为应用提供非常基础和可扩展的资源分配。多媒体应用可以在它产生的分组的IP首部设置区分服务码点（DSCP），以努力确保媒体分组和控制分组都能得到适当的服务质量。例如，将声音媒体分组标记为“快速转发”（EF），将使得它们被路径上的路由器放置在一个低延迟或高优先级的队列中，而呼叫信令（如SIP）分组则通常标记为某种“确保转发”（AF）以便能够使得它们与尽力而为流量分别排队，从而减小丢失的风险。

当然，如果网络设备（如路由器）关心DSCP，那么标记发送主机或设备中的分组才有意义。一般来讲，公共因特网上的路由器忽略DSCP，而为所有分组提供尽力而为的服务。然而，企业或公司网络有能力为它们内部的多媒体流量使用区分服务，而且经常这样做。另外，甚至是因特网的住宅用户也可以通过在他们因特网连接的出网方向上使用区分服务来提高VoIP或其他多媒体应用的质量，如图9-11所示。这种方法是有效的，因为很多宽带因特网连接都是不对称的：如果出链路的速度比入链路低很多（即更多的资源限制），那么在那条链路上使用区分服务的资源分配能够为延迟和丢失敏感的应用弥补质量上的差异。



图9-11 用于VoIP应用的区分服务。区分服务排队仅用于从客户路由器到ISP的上行链路

虽然区分服务在简单性方面很有吸引力，但显然它不能满足应用在所有条件下的需

要。例如，假设图 9-11 中的上行带宽只有 100kbps，客户试图部署两个 VoIP 呼叫，每一个都使用 64kbps 的编码。显然，上行链路现在超过了 100%负载，这会引起很大的排队延迟和分组丢失。客户路由器有再多聪明的排队方法也不能解决这个问题。

很多多媒体应用的特性不是试图将很多呼叫塞入一个过窄的管道，而是阻塞一个呼叫而让另一个呼叫继续。也就是，最好让一个人成功完成一次会话，而让另外一个人听到忙信号，而不是让两个呼叫者都同时经历不可接受的音质。我们有时把这种应用称为具有陡峭的效能曲线（steep utility curve），意思是当网络提供的服务质量降低时，应用的效能（作用）会迅速下降。多媒体应用通常有这个特性，而很多传统应用则没有。例如，如果延迟达到数小时，电子邮件依然能工作得很好。

具有陡峭效能曲线的应用通常能很好地适应一些准入控制。如果你不能确定总是有充足的资源可用于支持应用需要的负载，那么准入控制提供了一种对一些应用说“不”而允许其他应用得到所需资源的方法。

我们在 6.5.2 节看到一种用 RSVP 实行准入控制的方法，在此简短地回顾一下，但使用会话控制协议的应用提供了一些其他的准入控制选项。这里需要注意的一个关键点是像 SIP 或 H.323 这样的会话控制协议经常在呼叫或会话的开始包括在一个端点和另一个实体（SIP 代理或 H.323 网关）之间的某种消息交换。这提供了一种很方便的方式来对不能得到足够资源的新呼叫说“不”。

作为一个例子，考虑图 9-12 中的网络。假设从分公司到总公司的广域网链路具有足以满足三个并发的使用 64kbps 编码的 VoIP 呼叫的带宽。每一部电话在发出一个呼叫时都需要已经与本地 SIP 代理或 H.323 网关进行了通信，因此对于代理/网关来说在链路满负载时回送一条消息告诉 IP 电话播放忙信号是很容易的。代理或网关甚至能够处理一个特定的 IP 电话可能同时发出多个呼叫的情况，并且可能使用不同的编码速度。然而，这种方案仅在没有其他设备能够在与网关或代理通话以前就使链路超载的情况下才能工作。区分服务排队可以确保用于文件传输的 PC 不会干扰 VoIP 呼叫。但是假设在远程办公室里有一些不需要先与网关或代理通话的 VoIP 应用，这样的应用如果能够对其分组设置适当标志并放入与现有 VoIP 流量相同的队列中，那么很显然，由于没有来自代理或网关的反馈，会导致链路达到超载点。

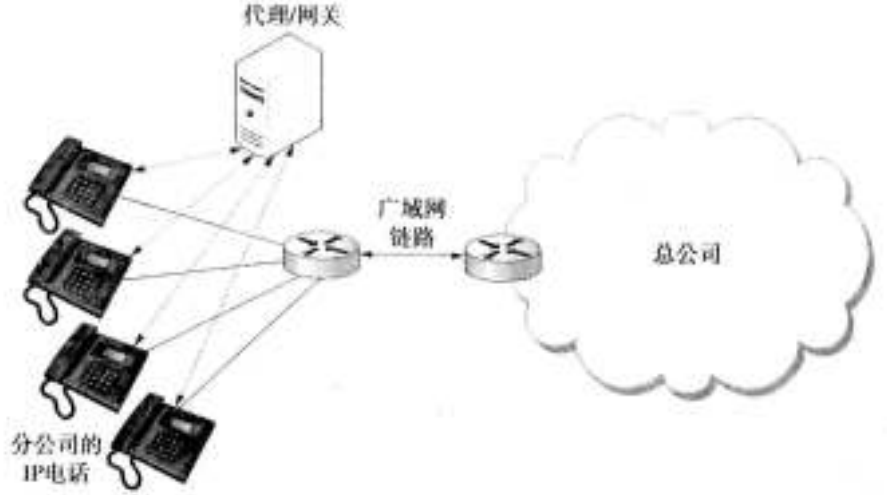


图 9-12　使用会话控制协议的准入控制

刚才描述的方法的另一个问题是它依赖于网关或代理知道每一个应用将使用的路径。在图 9-12 所示的简单拓扑中，这不是一个大问题，但是在更加复杂的网络中，这将很快变得不可管理。我们只需要想象这样一种情况：远程办公室有两个到外部世界的不同连接，我们则要求代理或网关不但要理解 SIP 或 H.323，还要理解路由、链路失败和当前网络状况。这将很快变得不可管理。

我们将刚刚描述的这种准入控制称为路径外的（off-path），因为做出准入控制决策的设备不在需要分配资源的数据路径上。一个显然的替代是路径上的（on-path）准入控制，在 IP 网络中提供路径上的准入控制的协议的标准示例是 RSVP。我们在 6.5.2 节看到了 RSVP 如何用于确保沿着一条路径分配足够的资源，在本节所描述的那些应用中使用 RSVP 也是顺理成章的。还有一个需要加入的细节是准入控制协议如何与会话控制协议交互。

协调准入控制（或资源预留）协议与会话控制协议的行为不是火箭科学（很困难的事），但仍然需要注意一些细节。作为一个例子，考虑两个实体之间的简单电话呼叫。在进行预留前，你需要知道该呼叫将使用多少带宽，意思是你需要知道将使用什么编码。这也意味着你需要先做一些会话控制以便交换关于两个电话所支持的编码的信息。然而，你不能先完成所有会话控制，因为你不想让电话在准入控制决策前就响铃，以防准入控制失败。图 9-13 说明了这种情况，其中 SIP 用于会话控制，RSVP 用于准入控制决策（在此例中成功执行）。

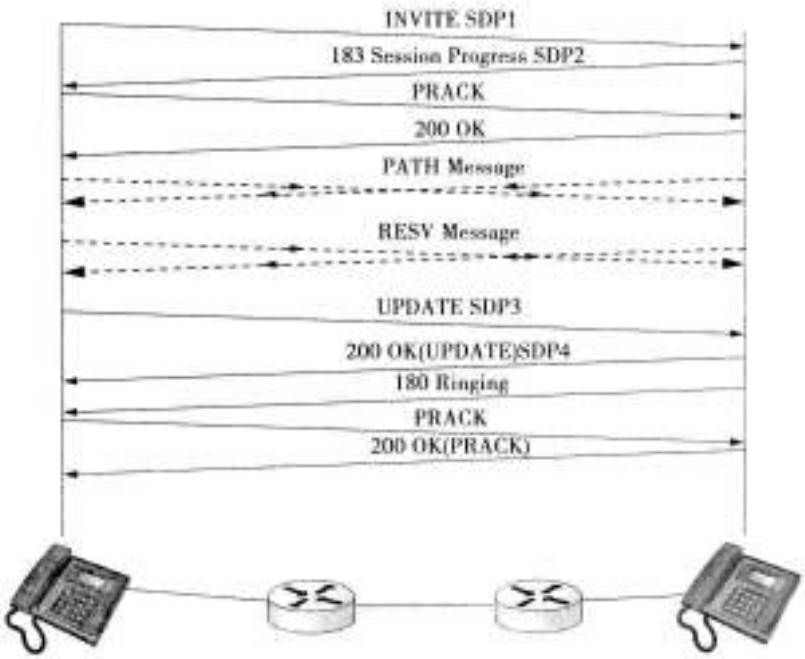


图 9-13 SIP 信令和资源预留的协作

这里需要注意的主要问题是会话控制与资源分配任务的交叉。实线代表 SIP 消息，虚线代表 RSVP 消息。注意，在该例中，SIP 消息直接在电话和电话之间传送（即我们没有显示任何 SIP 代理），而 RSVP 消息也被中间的路由器处理以检查是否有足够的资源来允许该呼叫。

我们先看一下前两条 SIP 消息中交换的编码信息（回忆一下，SDP 用于列举所有可用编码）。PRACK 是一个“预确认”。一旦交换了这些消息，包含对所需要资源总量的描述的 RSVP PATH 消息就可以作为在呼叫的两个方向上进行资源预留的第一步而被发送。接下来，可以返回 RESV 消息来实际预留资源。一旦主叫电话接收到 RESV 消息，它可

以发送一条更新的 SDP 消息来报告已经在一个方向上预留了资源这样一个事实。当被叫电话收到这两条消息以及从另一个电话发来的 RESV 时，它就可以振铃了，并告诉另外一个电话现在两个方向上都已经预留了资源（用 SDP 消息），还要通知主叫电话它正在振铃。从现在起，正常的 SIP 信令和媒体流将继续下去，如图 9-9 所示。

我们又一次看到构建应用要求我们理解不同构建模块（在本例中是 SIP 和 RSVP）之间的交互。SIP 的设计者实际上对协议做了一些修改，使得完成不同工作的协议之间能够实现功能交叉。因此，我们再次强调本书关注整个系统，而不是孤立于系统的其他部分来看一层或一个组件。

9.3 基础设施服务

有一些协议对于因特网的平稳运行是至关重要的，但并不完全符合严格的分层模型。其中一个是域名系统（DNS）——不是一个用户通常显式调用的应用，而是几乎所有其他应用都依赖的一种服务。这是因为名字服务用于将主机名翻译成主机地址，这种应用的存在使得其他应用的用户可以通过名字而不是地址来访问远程主机。换句话说，名字服务一般被其他应用使用，而不是人类使用。

另一个关键功能是网络管理，虽然普通用户对它不熟悉，但它却是系统管理员经常进行的操作。网络管理普遍被认为是联网的难题之一，并继续成为大量研究的重点。下面看一下相关问题和解决方法。

9.3.1 名字服务（DNS）

到目前为止，我们在本书中一直使用地址来标识主机。地址虽然非常适合路由器的处理，但对用户并不十分友好。因此，通常也为网络中的每台主机分配一个唯一的名字（name）。在本节中，我们已经看到 HTTP 的应用层协议使用诸如 www.princeton.edu 的名字。本节描述如何开发一个命名服务系统以便把对用户友好的名字映射成对路由器友好的地址。由于这种服务可使其他应用自由地通过名字而不是地址来识别主机，所以它往往是在网络上实现的第一个应用程序。名字服务有时也称为中间件（middleware），因为它填补了应用程序与底层网络之间的间隙。

主机名在两个重要的方面不同于主机地址。第一，它们通常是可变长且容易记忆的，因此很容易被人们记住。（与之对照的固定长度的数字地址则更易于路由器的处理。）第二，主机名通常不包含能够帮助网络定位主机的信息（即分组的路由走向）；与之相比，地址有时则有内嵌的路由选择信息。扁平（flat）地址（没有将地址划分成各个组成部分）属于例外。

在详细介绍在网络中如何命名主机之前，我们首先介绍一些基本术语。首先，*命名空间*（name space）定义所有可能的名字的集合。命名空间可以是*扁平的*（flat）（名字中不划分各个部分），也可以是*分层的*（hierarchical）（UNIX 文件名是一个明显的例子）。第二，命名系统维护一组名字到值的*绑定*（binding）。当我们提供一个名字后，命名系统可以返回一个我们希望的值，在多数情况下是一个地址。最后，*解析机制*（resolution mechanism）是这样一个过程：当以一个名字调用它时，返回相应的值。*名字服务器*（name server）是网络中可用的解析机制的一个特定的实现，并能用发送消息的方式来查询它。

因为因特网很大，所以它有一个开发得非常好的命名系统——域名系统（Domain

Name System，DNS)。因此，我们用 DNS 作为讨论命名主机问题的框架。注意因特网并不是一直使用 DNS。早期，因特网仅仅有几百台主机，那时有一个称为网络信息中心(Network Information Center，NIC) 的中央权威机构维护着一个名字到地址绑定的扁平表，此表称为 hosts.txt。无论何时任何站点想要加一个新主机到因特网上，站点管理者都会将新主机的名字/地址对用电子邮件发给 NIC。这个信息被人工加到表（hosts.txt）中，每隔几天后就把修改过的表发给各个站点一次，然后每个站点的系统管理员将表安装到站点的每台主机上。名字解析简单地用一个过程来实现，即在表的本地备份中找到主机名，然后返回相应的地址。

毫无疑问，随着因特网主机数目的增长，使用 hosts.txt 方法的名字服务不再有效。因此在 20 世纪 80 年代中期，域名系统开始投入使用。DNS 采用分层的命名空间而不是扁平的命名空间，并且实现此命名空间的绑定表被划分为不相交的分布在因特网中的子表。可通过网络查询名字服务器上的可用子表。

因特网中发生的事情是：用户提供一台主机名给应用程序（可能嵌入在一个复合名字中，如电子邮件地址或 URL)，这个应用程序启用命名系统将名字翻译成一台主机地址。然后应用程序通过某个传输协议（如 TCP）根据主机的 IP 地址打开一个到这台主机的连接。这种情况如图 9-14 所示（发送电子邮件的情况)。虽然该图中的名字解析看上去很简单，但我们将看到它还包括更多内容。

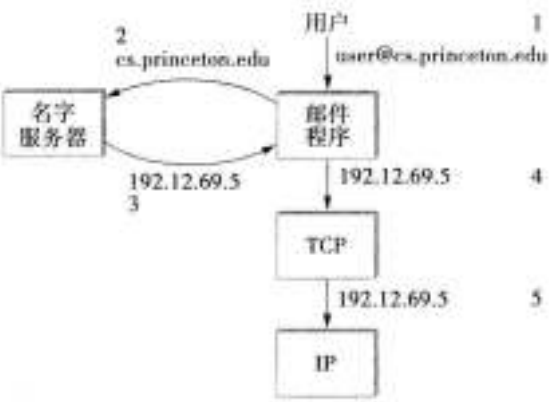


图 9-14 名字翻译成地址，其中数字 1~5 显示处理过程中步骤的顺序

1. 域名的分层结构

DNS 为因特网对象实现了一个分层的命名空间。与 UNIX 文件名不同的是（UNIX 文件名是将名字的组成部分从左到右用斜线分隔)，DNS 名字从右向左进行处理，并用圆点分隔。(尽管域名是从右向左处理，但人们“读”域名时仍然是从左向右。) 例如，一个主机域名的例子是 cicada.cs.princeton.edu。注意，我们说过域名用于命名因特网对象，意思是 DNS 并不严格地用来将主机名映射到主机地址，更精确地说，DNS 将域名映射为值。目前，我们假设这些值是 IP 地址，本节的稍后部分将重新讨论这个问题。

和 UNIX 的文件分层结构一样，DNS 的分层结构也可看作一棵树，树中的每个节点对应一个域，树的叶子对应被命名的主机。图 9-15 给出域名分层结构的一个例子。注意，我们并不指定术语域（domain）的任何语义，只是简单地认为它是定义其他名字的一个上下文。[⊖]

事实上，当域名分层结构刚发展到采用什么惯例来支配从分层结构的顶部分发名字时，曾引发了大量的讨论。不必详细讨论，我们就可以注意到层次结构的第一层的范围并不大。有每个国家的域，再加上“6 个大域”：.edu、.com、.gov、.mil、.org 和 .net。这 6 个域都以美国为基础（因特网和 DNS 是在美国发明的)。例如，只有美国的教育机构

⊖ 令人困惑的是词语域（domain）也用在因特网路由中，但其含义与在 DNS 中的含义不同，大体上与术语自治系统（autonomous system）等价。

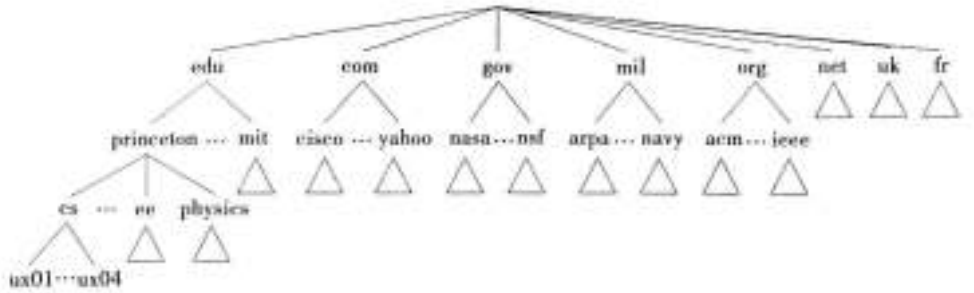


图 9-15　域名层次结构示例

可以注册一个 .edu 域名。最近，顶级域已经被扩展了，部分原因是为了满足 .com 域名的大量需求。新的顶级域包括 .biz、.coop 和 .info。另一个近期发展是支持用非拉丁字符集表示的域名，如阿拉伯文和中文。

2. 名字服务器

完全的域名分层结构只是一种抽象。我们现将注意力转向分层结构在实际中如何实现这个问题。第一步将分层结构分为子树，叫作区域（zone）。图 9-16 显示了如何将图 9-15 中的分层结构划分为区域。每个区域可看作对应于负责分层结构中一部分的某个管理权威。例如，分层结构的顶层可以形成一个区域，由因特网名字与编号分配机构（Internet Corporation for Assigned Name and Numbers，ICANN）来管理。在它下面是一个对应于普林斯顿大学的区域。在这个区域内，有些系不想承担管理分层结构的责任（因此它们留在大学级区域中），而另一些系，如计算机科学系，管理它们自己的系级区域。

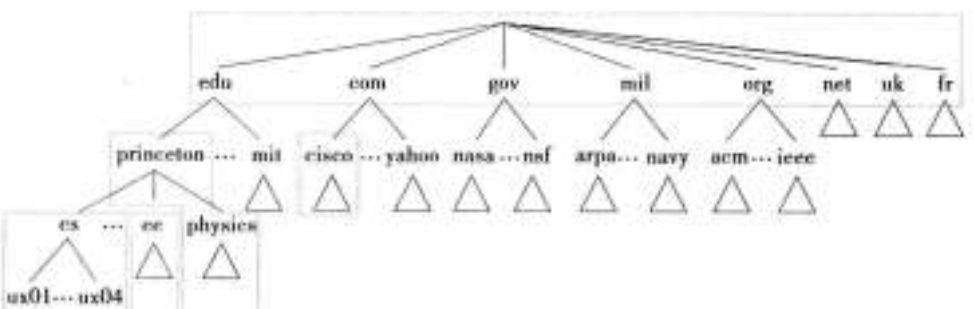


图 9-16　划分为区域的域名分层结构

区域的相关性是它对应于 DNS 实现的基本单元——名字服务器。特别是，每个区域中包含的信息被实现在两个或多个名字服务器上。而每个名字服务器是通过因特网可访问的一个程序。客户向名字服务器发出请求，名字服务器响应所请求的信息。有时，响应信息包含用户想要得到的最终回答，而有时响应信息包含指向另一个服务器的指针，客户端由此继续向下一个服务器提出请求。这样，从实现的角度来看，更确切地说，DNS 可看作一个名字服务器的分层结构，而不是域的分层结构，

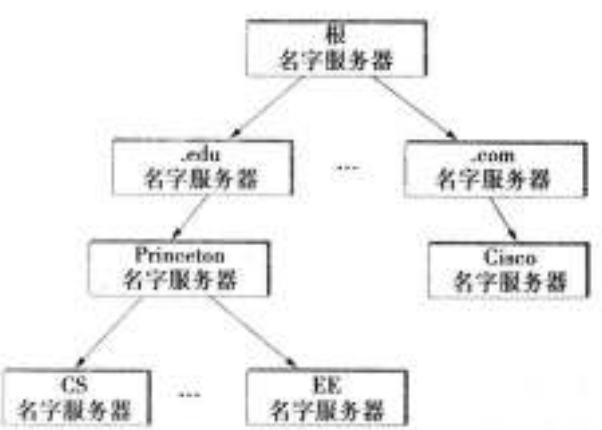


图 9-17　名字服务器的分层结构

图 9-17 给出了解释。

注意，为了实现备份，每个区域在两个或多个名字服务器上实现，这样，即使一个名字服务器出了故障，仍然可以获得信息。另一方面，一个给定的名字服务器可以实现多个区域。

每个名字服务器以资源记录（resource record）集合的形式来实现区域信息。本质上，一个资源记录是一个名字到值的绑定，或者更确切地说，是一个包括以下字段的 5 元组：

(Name, Value, Type, Class, TTL)

Name 和 Value 字段的内容顾名思义，而 Type 字段说明值应如何解释。例如，Type＝A 表明 Value 是一个 IP 地址。所以，A 类型记录实现了我们所设想的名字到地址的映射。其他记录类型包括：

- NS：Value 字段给出运行名字服务器的一台主机的域名，该主机知道如何解析指定域中的域名。
- CNAME：Value 字段给出一个特定主机的规范名，用于定义别名。
- MX：Value 字段给出运行邮件服务器的主机的域名，该服务器接收来自指定域的消息。

Class 字段允许除 NIC 以外的其他实体定义有用的记录类型。至今，唯一广泛使用的 Class 是由因特网使用的 Class，记为 IN。最后，TTL 字段指出这个资源记录的有效时间。它被服务器用来缓存来自其他服务器的资源记录，当 TTL 到期后，服务器必须从它的缓存中清除这个记录。

为了更好地理解资源记录如何表示域层次中的信息，考虑下列从图 9-5 的分层结构中抽取的一些例子。为了简化这个例子，我们忽略 TTL 字段并且只给出实现这个区域的名字服务器中的相关信息。

首先，根名字服务器为每个顶级域名服务器（Top-Level Domain，TLD）包含一个 NS 记录，它可以标识为 DNS 分层结构中的一部分（本例中的 .edu 和 .com）提供查询解析的服务器。它还有一个 A 记录用来将该名字翻译为相应的 IP 地址。一起使用这两条记录可有效地实现一个从根名字服务器到每个顶级域名服务器的指针。

```
(edu, a3.nstld.com, NS, IN)
(a3.nstld.com, 192.5.6.32, A, IN)
(com, a.gtld-servers.net, NS, IN)
(a.gtld-servers.net, 192.5.6.30, A, IN)
...
```

沿分层结构向下走一层，a3. nstld. com 服务器有 .edu 域的记录，如下：

```
(princeton.edu, dns.princeton.edu, NS, IN)
(dns.princeton.edu, 128.112.129.15, A, IN)
...
```

在这种情况下，我们得到分层结构中负责 princeton. edu 部分的名字服务器的 NS 记录和 A 记录。这个服务器可能会直接解析一些查询（如，对 email. princeton. edu 的查询），而把其他查询重定向到甚至位于分层结构中另一层中的一个服务器（如，对 penguins.

cs. princeton. edu 的查询）。

(email.princeton.edu, 128.112.198.35, A, IN)
(penguins.cs.princeton.edu, dns1.cs.princeton.edu, NS, IN)
(dns1.cs.princeton.edu, 128.112.136.10, A, IN)
⋮

最后，第三级名字服务器，例如由域 cs. princeton. edu 管理的名字服务器，包含有域内所有主机的 A 类记录。它也可以为每一台主机都定义一个别名（CNAME 记录）集合。别名有时只是便于计算机处理的（如更短的）名字，但它们也能用来提供一个间接层。例如，www. cs. princeton. edu 是名为 coreweb. cs. princeton. edu 的主机的别名，这允许站点的 Web 服务器在不影响远程用户的情况下转移到其他机器上。远程用户可简单地继续使用该别名而不考虑当前哪台计算机运行着该域的 Web 服务器。邮件交换（MX）类记录对邮件应用有同样的用途，它允许管理者改变代表本域接收邮件的主机而不必改变每个人的邮件地址。

(penguins.cs.princeton.edu, 128.112.155.166, A, IN)
(www.cs.princeton.edu, coreweb.cs.princeton.edu, CNAME, IN)
(coreweb.cs.princeton.edu, 128.112.136.35, A, IN)
(cs.princeton.edu, mail.cs.princeton.edu, MX, IN)
(mail.cs.princeton.edu, 128.112.136.72, A, IN)
⋮

注意，虽然可以为任意类型的对象定义资源记录，但是通常 DNS 用于命名主机（包括服务器）和站点。它不用于命名个人或其他像文件和目录之类的对象，通常用其他命名系统来识别这样的对象。例如，X. 500 是一个 ISO 命名系统，设计它的目的是为了更容易识别个人。它允许你通过给出一组属性：名字、头衔、电话号码、邮政地址等来命名个人。X. 500 被证明过于繁琐，并且在某种意义上，它已经被现存的 Web 上强有力的搜索引擎所取代，但它最终发展成轻量目录访问协议（Lightweight Directory Access Protocol，LDAP）。LDAP 是 X. 500 的一个子集，最初被设计为 X. 500 的 PC 前端。今天，它作为一个了解用户信息的系统主要流行于企业中。

3. 名字解析

给出名字服务器的分层结构后，我们现在来考虑客户如何让这些服务器解析域名的问题。为了解释其基本思想，假设客户想要解析与上一小节给出的服务器集合有关的名字 penguins. cs. princeton. edu。客户首先发送一个包含这个名字的查询给根服务器（你将在下面看到，这在实际中是很少发生的，但对于目前解释基本操作来说，已经足够了）。根服务器不能匹配整个名字，只返回一个它所能提供的最佳匹配，即指向 TLD 服务器 a3. nstld. com 的 edu 的 NS 记录。服务器也返回与此记录相关的所有记录，在这种情况下，返回的是 a3. nstld. com 的 A 类记录。客户没有得到想要的答案，接着发送同样的查询给 IP 主机为 192. 5. 6. 32 的名字服务器。这个服务器也不能匹配整个名字，因此返回 princeton. edu 域的 NS 类记录和相应的 A 类记录。客户再一次发送同样的查询给 IP 主机为 128. 112. 129. 15 的服务器，这次得到 cs. princeton. edu 域的 NS 类记录和相应的 A 类

记录。此时到达了能够完全解析该查询的服务器。最后，向服务器 128.112.136.10 发送查询，返回 penguins.cs.princeton.edu 的 A 类记录，客户得知对应的 IP 地址是 128.112.155.166。

这个例子仍留下两个有关解析处理的问题没有回答。第一个问题是，客户如何首先确定根服务器的位置，或者说，怎样解析那台知道如何解析这些名字的服务器的名字？在任何命名系统中，这都是一个最基本的问题，答案是必须以某种方式启动系统。在这种情况下，一个或多个根服务器的名字到地址的映射是众所周知的，就是通过某种名字系统以外的方式发布。

然而实际中，并不是所有客户都知道根服务器。相反，运行在因特网的每一台主机上的客户程序在初始化时都设置了一个本地（local）名字服务器地址。例如，Princeton 大学计算机科学系的所有主机都知道名字服务器是 dns1.cs.princeton.edu。这个本地名字服务器又有一个或多个根服务器的资源记录，例如：

```
('root',a.root-servers.net,NS,IN)
(a.root-servers.net,198.41.0.4,A,IN)
```

这样，解析名字实际涉及客户查询本地服务器，这个本地服务器再作为一个客户端为原来的客户查询远程服务器。这种客户/服务器之间的交互结果如图 9-18 所示。这种模式的一个优点是因特网中的所有主机不必保留最新的根服务器的位置信息，而只是服务器必须知道根服务器的情况。第二个优点是本地服务器可以看到由所有本地客户发出查询而返回的应答。本地服务器缓存（cache）这些应答，有时候不用出网就能够解决未来的查询。由远程服务器返回的资源记录的 TTL 字段指出每个记录能被可靠缓存的时间期限。这种缓存机制也可以用于分层结构的上层，以便减小根服务器和 TLD 服务器的负荷。

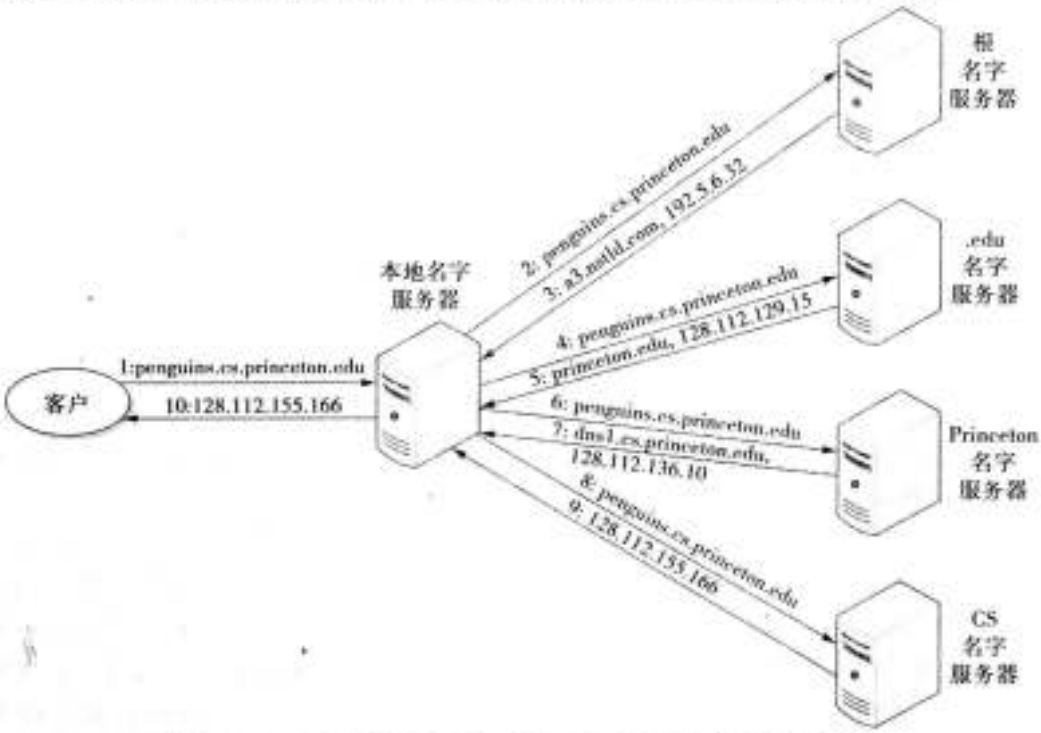


图 9-18 实际中的名字解析，数字 1—10 显示处理步骤的顺序

第二个问题是当用户提交一个部分名字（如 penguins）而不是完整域名（如 penguins. cs. princeton. edu）时系统如何工作。答案是将客户程序配置为主机所在的本地域（如 cs. princeton. edu），在发出查询之前将其追加到任何一个简单名字之后。

结论 现在，我们清楚了三个不同层次的标识符（域名、IP 地址和物理网络地址）并且一层标识符到另一层标识符的映射发生在网络体系结构的不同点上。首先，当用户与应用程序交互时指明域名。第二，应用程序令 DNS 将这个名字翻译为一个 IP 地址，放在每个数据报中的是 IP 地址而不是域名。（另外，这个翻译过程涉及通过因特网发送 IP 数据报，但是这些数据报寻址到运行名字服务器的主机，而不是最终目标）。第三，IP 在每台路由器上执行转发，这常常意味着将一个 IP 地址映射为另一个 IP 地址，即将最终目标地址映射为下一跳路由器的地址。最后，IP 使用 ARP 将下一跳路由器的 IP 地址翻译成机器的物理地址，下一跳可能是最终目标也可能是中间路由器。在物理网络上发送的帧的首部中有这些物理地址。

相关主题

命名惯例

我们对于 DNS 的描述着重于基本机制（mechanism），即多个服务器上的分层结构如何划分及解析过程如何工作。同样值得注意但没什么技术性的问题是决定这些机制中使用的命名惯例（convention）。例如，所有美国大学都在 edu 域内是一个惯例，而英国的大学则在 uk（United Kingdom，联合王国）域内的 ac（academic，学术）子域内。

要知道有时候惯例的定义并没有任何人做明确的决定。例如，按惯例一个站点将为其提供邮件交换服务的主机隐藏在 MX 记录后。另一个可采用的惯例是将邮件送给 user@mail. cs. princeton. edu，很像我们期望在 ftp. cs. princeton. edu 上找到站点的一个公共 FTP 目录和在 www. cs. princeton. edu 上找到它的 WWW 服务器。

惯例也存在于本地级别的命名中，组织根据某种统一的规则集合为其计算机命名。在因特网中，主机名 venus、saturn 和 mars 非常普通，找到公共的命名惯例并不难。然而，一些主机的命名惯例更有想象力。例如，一个站点将它的计算机命名为 up、down、crashed、rebooting 等，结果会造成像 "rebooting has crashed" 或 "up is down" 等让人迷惑的说法。当然，也有些名字没有什么想象力，比如那些以整数命名的机器。

9.3.2 网络管理（SNMP）

无论从网络所涉及的节点数来说，还是从能够运行在任何一个节点上的协议族来说，网络都是一个复杂的系统。即使只关心在单一管理域中的节点，比如一个校园网，那也可能有几十台路由器和成百甚至上千台主机需要跟踪。如果你想对任何一个节点上的诸如地址转换表、路由表、TCP 连接状态等状况都进行维护和操作，管理所有这些信息的前景很快会变得令人沮丧。

我们很希望了解在不同节点上各种协议的状态。例如，你可能想要监测重组失败的 IP 数据报的数目，以便决定是否需要调整碎片收集程序用于收集部分组装的数据报的超时时间。在另一个例子中，你可能想要追踪各种节点上的负载（如发送或接收分组的数目），

以便决定是否需要在网络中加入新的路由器或链路。当然，你也必须监视硬件故障和软件异常反应。

刚才我们描述了网络管理的问题，它遍及整个网络体系结构。由于我们想要跟踪的节点是分布式的，所以我们仅有的实际选择是使用网络管理网络。这意味着我们需要一种协议，它允许读（可能还需要）写不同网络节点上的各种状态信息。用于此目的的使用最广泛的协议是简单网络管理协议（Simple Network Management Protocol，SNMP）。

SNMP 本质上是一种专用的请求/响应协议，它支持两种请求消息：GET 和 SET。前者用于从某个节点上获取状态，而后者用于把新状态存储到某个节点上。（SNMP 也支持第三种操作 GET-NEXT，我们会在下面解释它。）下面重点讨论 GET 操作，因为它是使用最频繁的一种操作。

SNMP 的使用方法很简单：系统管理员与显示网络信息的客户程序进行交互。这个客户程序通常有图形界面。你可以认为这个界面扮演着与 Web 浏览器相同的角色。只要管理员选定了他想看的特定信息，客户程序就使用 SNMP 向被询问的那个节点请求所要的信息（SNMP 运行在 UDP 之上）。运行在那个节点上的 SNMP 服务器接收请求，找到恰当的信息，并将它返回到客户程序，然后再显示给用户。

这种特别简单的场景中仅存在一个复杂的问题：客户怎样正确指明它想要哪些信息，同样，服务器如何知道读出内存中的哪个变量来满足这个请求？答案是 SNMP 依赖于一个称为管理信息库（Management Information Base，MIB）的配套规范。这个 MIB 定义了一些特殊的信息（MIB 变量（variable）），你可以从网络节点上检索它们。

当前的 MIB 版本称为 MIB-II，它将变量组织在 10 个不同的组（group）中。你将发现大部分组与本书描述的某个协议对应，并且每组内定义的几乎所有变量看上去都很熟悉。例如：

- System：整个系统（节点）的通用参数，包括节点定位在何地、它已启动多长时间和系统的名字。
- Interfaces：关于连接到这个节点上的所有网络接口（适配器）的信息，比如每个接口的物理地址、每个接口上已发送和接收了多少分组。
- Address translation：关于地址解析协议（Address Resolution Protocol，ARP）的信息，特别是它的地址转换表的内容。
- IP：与 IP 相关的变量，包括它的路由表、它已成功转发了多少数据报和重组数据报的统计。它还包括因某种原因而使 IP 丢弃数据报的次数。
- TCP：关于 TCP 连接的信息，如被动打开和主动打开连接的次数、重置次数、超时次数、默认超时设置等。只要连接存在，每个连接的信息就持续存在。
- UDP：关于 UDP 通信量的信息，包括已发送和接收到的 UDP 数据报的总数。

还有 ICMP、EGP 和 SNMP 自身的组。第 10 组用于不同的媒体。

再回到这样一个话题，即客户需要确切指出它想从节点得到什么信息。MIB 变量列表仅完成了一半的工作，这留下两个问题。第一，我们需要精确的语法，使客户用来说明想要取哪些 MIB 变量。第二，我们需要一个对服务器返回值的精确表示。这两个问题都使用 ASN.1 来解决。

首先考虑第二个问题。正如我们已经在第 7 章中看到的，ASN.1/BER 为不同的数据类型定义表示法，比如整数。MIB 定义每个变量的类型，然后当它要在网上传输时，用

ASN.1/BER 对包含在这个变量中的值进行编码。至于第一个问题，ASN.1 也定义了一种对象识别方案，这种识别系统没有在第 7 章中描述。MIB 使用该识别系统为 MIB 中的每一个变量指定了一个全局唯一的标识符。这些标识符用圆点“.”记法给出，类似域名。例如：1.3.6.1.2.1.4.3 是 IP 相关的 MIB 变量 ipInReceives 的唯一 ASN.1 标识符，该变量记录本节点接收的 IP 数据报的数目。在本例中，前缀 1.3.6.1.2.1 标识 MIB 数据库（记住，ASN.1 对象 ID 适用于世界上所有可能的对象），4 对应于 IP 组，最后的 3 表示该组中的第三个变量。

这样，网络管理按下述步骤进行工作。SNMP 客户将需要的 MIB 变量的 ASN.1 标识符放入请求消息，并发送此消息到服务器。服务器则将该标识符映射到一个本地变量（即到一个存储着该变量值的内存位置），取回该变量的当前值，并且使用 ANS.1/BER 编码该值后返回给客户。

还有最后一个细节。许多 MIB 变量是表或者结构，这样的组合型变量说明了使用 SNMP GET-NEXT 操作的理由。当该操作应用到一个特定的变量 ID 时，返回变量的值和下一个变量的 ID，如表的下一项或结构中的下一个字段。这有助于客户“遍历”表或结构的所有元素。

9.4　覆盖网络

起初，因特网采用了清晰的模型，即网络中的路由器负责从源向目的地转发分组，而应用程序运行在连接到网络边缘的主机上。本章前两节讨论的客户/服务器的应用范例正符合这种模型。

然而在最近几年，*分组转发*（packet forwarding）和*应用处理*（application processing）之间的区别变得不太清晰。新的应用被分布到因特网上，在许多情况下，这些应用做出自己的转发决定。有时，通过扩展传统的路由器和交换机支持一定数量的特定应用处理，可以实现这些新的混合型应用。例如，所谓的*七层交换机*（level-7 switch）位于服务器群前面，并根据请求的 URL 将 HTTP 请求转发至特定服务器。然而，*覆盖网络*（overlay network）作为把新功能引入因特网的可选机制迅速兴起。

你可以把一个覆盖看成一个在底层网络之上实现的逻辑网络。根据这个定义，因特网本身就是建立在老电话网链路之上的覆盖网络，事实的确如此。图 9-19 描述了一个实现在底层网络之上的覆盖。覆盖中的每个节点也存在于底层网络中，它按一种特定于应用的方法来处理和转发分组。连接覆盖节点的链路作为通过底层网络的隧道来实现。多个覆盖网络能存在于同一个底层网络之上（每个实现它们自己的特定应用行为），并且覆盖可以嵌套，一个搭建在另一个之上。例如，本节讨论的所有覆盖网络示例都将今天的因特网视为底层网络。

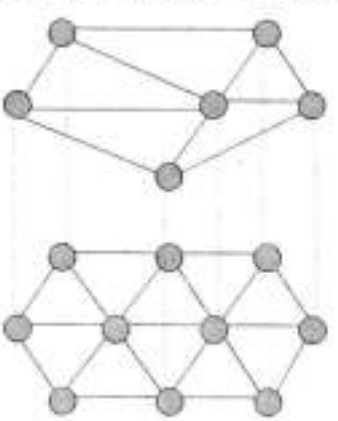


图 9-19　在物理网络之上的覆盖网络

我们看过隧道化的例子，例如用于实现虚拟专用网（VPN）。简要复习一下，隧道两端的节点视它们之间的多条路径为一个单一的逻辑链路，节点根据分组外部的首部通过隧道转发分组，而从不知道端节点已经附加了内部首部。例如，图 9-20 显示了由一对隧道连接的三个覆盖节点

(A、B和C)，覆盖节点B根据内部首部（IHdr）为来自A的分组做出转发到C的决定，然后附加一个外部首部（OHdr），把C标识为底层网络的目的地。节点A、B和C能解释内部和外部首部，然而中间的路由器仅能理解外部首部。类似地，A、B和C在覆盖网络和底层网络中都有地址，但是不一定相同。例如，它们的低层地址可以是32位的IP地址，而它们的覆盖地址可以是实验性的128位地址。事实上，覆盖根本不需要使用传统地址，而可以根据URL、域名、XML查询甚至分组的内容来决定路由。

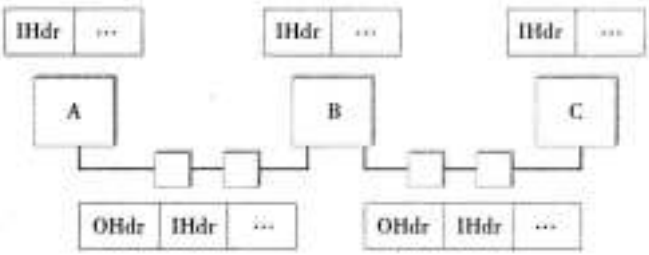


图9-20 穿过物理节点的覆盖节点隧道

相关主题

覆盖和因特网的僵化

由于因特网的流行和广泛使用，使得人们容易忘记它曾经是一个为研究人员实验分组交换网络的实验设施。然而，因特网在商业上越是成功，作为提供新思想的平台的作用就越小。今天，商业利益限制了因特网的持续发展。

事实上，来自美国国家研究委员会（Nation Research Council）2001年的报告指出，不管在智能（与现行标准的兼容性的压力抑制了其进一步创新）方面还是基础结构本身（研究人员影响核心基础结构几乎是不可能的）方面，因特网都是僵化的。同时，一整套可能需要新解决方法的新挑战也正在出现。报告指出，进退两难的局面在于：

> "……成功和广泛被采用的技术容易僵化，使得新功能很难引进，或者如果继续采用现行技术，很难用更好的技术取而代之。现有的业界人士一般不积极开发或部署创新性的技术……"

寻找引进创新技术的正确途径是令人感兴趣的问题。这样的创新能把某些事情做得非常好，但整体上它们落后于其他重要领域的现有技术。例如，为了在因特网中引进一种新的路由策略，你必须构造一种路由器，它不仅支持这一新策略，同时也能与市场已有的产品在性能、可靠性、管理工具集等方面进行竞争。这是一个过分的要求。创新者需要的是一种方法，允许用户利用新思想，而又不必编写为支持基本系统所需的成千上万行的代码。

覆盖网络恰好提供了这样的机遇。可通过编程使覆盖节点支持新的功能或特性，并且依赖传统节点提供低层连接。随着时间的推移，如果在覆盖中所用的思想被证明是有用的，就可能有经济动力把这些功能移植到基本系统中，即把它加入到商用路由器的功能特性集合中。另一方面，这些功能可能相当复杂，足以使覆盖层保持原状。

9.4.1 路由覆盖

最简单的覆盖是纯粹用来支持路由的一种替代策略，在覆盖节点上没有执行额外的应

用层处理。可以把虚拟专用网（见 4.1.8 节）看作一个路由选择覆盖的例子，与其说它定义一个替代策略或算法，不如说定义由标准 IP 转发算法处理的替代路由表项。在这种特殊情况下，覆盖使用“IP 隧道”，而大多数商用路由器支持利用这些 VPN 的能力。

然而假设你打算使用商用路由器提供商不想在其产品中提供的路由算法，那么如何处理这种情况？你可以简单地在一些端主机上运行你的算法并且以隧道方式穿过因特网路由器。这些主机的行为像是覆盖网络上的路由器：作为主机它们可能仅通过一条物理链路连接到因特网，但作为覆盖节点它们通过隧道连接到多个邻居。

因为覆盖根据定义几乎是引入独立于标准化处理的新技术的一种途径，所以我们不能把哪个标准覆盖作为例子。相反，我们通过描述一系列近来由网络研究人员提出的实验系统来说明路由选择覆盖的一般思想。

1. IP 实验版本

覆盖对于部署你希望最终将占领世界的 IP 实验版本来说是理想的。例如，IP 多播（见 4.2 节）开始是作为 IP 的一种扩展，但直到今天仍然有很多因特网路由器不支持它。多播主干（MBone）是在因特网单播路由的基础上实现的 IP 多播的覆盖网络。Mbone 上开发并部署了大量多媒体会议工具，例如，IETF 会议（它持续一周并且吸引了成千上万的参与者），多年来都在 MBone 上广播。

像 VPN 一样，MBone 同时使用 IP 隧道和 IP 地址，但与 VPN 不同的是，MBone 实现一个不同的转发算法——它转发分组到最短路径多播树中的所有下游邻居。作为覆盖，支持多播的路由器建立穿过传统路由器的隧道，同时希望有朝一日它将不再使用传统路由器。

6-BONE 是一个类似的覆盖，它用于逐步部署 IPv6。像 Mbone 一样，6-BONE 通过 IPv4 路由器用隧道转发分组。然而与 MBone 不同的是，6-BONE 节点不是简单地提供 IPv4 的 32 位的新解释，而是根据 IPv6 的 128 位地址空间转发分组。此外，6-BONE 也支持多播。

2. 端系统多播

虽然 IP 多播在研究者和因特网社团的特定部分很流行，但它在全球因特网上的部署受到很大的限制。因此，像视频会议那样的基于多播的应用近来已经转向另外一种策略，称为*端系统多播*（end system multicast）。端系统多播的概念是承认 IP 多播将不会无处不在，而代之以让参与特定多播应用的端主机实现自己的多播树。

在描述端系统多播如何工作之前，重要的是先理解它不像 VPN 和 MBone，端系统多播假设只有因特网主机（而不是因特网路由器）参与覆盖。此外，这些主机通常是通过 UDP 隧道而不是 IP 隧道相互交换消息，使它容易作为常规的应用程序来实现。这使得有可能将底层网络看成是一个全连通图，因为在因特网中的每台主机都能向所有其他主机发送消息。因此，可以抽象地说端系统多播解决了下列问题：从表示因特网的全连通图出发，目标是找出其中跨越全体组成员的内嵌多播树。

既然我们把底层的因特网看成是全连通的，一种简单的解决方案是使每一个源直接与每个组成员连接。换句话说，端系统多播能够通过由每个节点发送单点播送消息到每个组成员来实现。为看清这其中的问题，特别是与在路由器中实现 IP 多播相比较，考虑图 9-21中的示例拓扑图。图 9-21a 描绘了一个物理拓扑的例子，其中 R1 和 R2 是由低带

宽的横跨大陆的链路所连接的路由器；A、B、C 和 D 是端主机，链路的延迟作为边的权值。假设 A 想要发送多播消息到其他三台主机，图 9-21b 显示了简单的单点播送通信是如何工作的。同样的消息必须穿越 A-R1 链路三次，且消息的两个副本穿越 R1-R2，显然这是不希望发生的。图 9-21c 描述了通过 DVMRP 构造的 IP 多播树，显然，这一方法排除了多余的消息。然而，没有路由器的支持，你能希望得到的最好的端系统多播是一个类似于图 9-19d 显示的树。端系统多播定义了一个构造此树的体系结构。

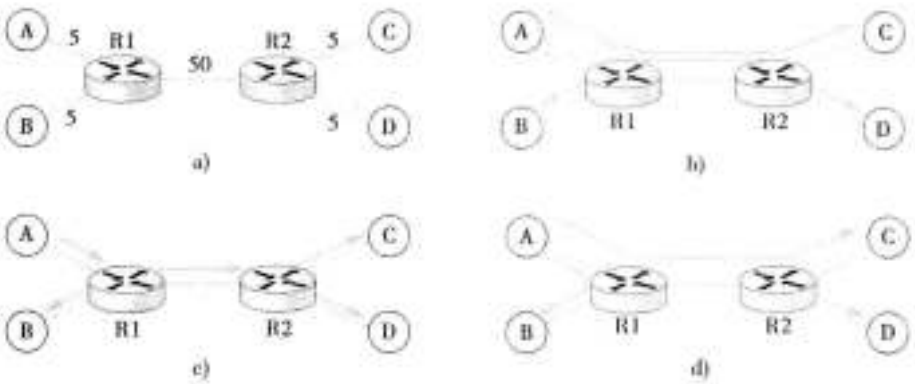


图 9-21 映射到一个物理拓扑上的可选的多播树

一般的做法是支持多层的覆盖网络，每一层覆盖网络从下一层覆盖网络中抽取一个子图，直到我们已经选择了应用所期望的子图。特定的端系统多播发生在两个阶段：首先我们在全连通的因特网顶部构建一个简单的网格（mesh），然后我们在此网格内选择一个多播树。这一思想在图 9-22 中说明，我们再次假设四个端主机 A、B、C 和 D。第一步是关键性的：一旦我们选择了一个合适的网格覆盖，我们就在其上简单地运行一个标准的多播路由算法（如 DVMRP）来构造多播树。我们还奢望忽略因特网范围的多播所面对的可扩展性问题，因为可以选择仅包括那些想要参与特定多播组的节点的中间网格。

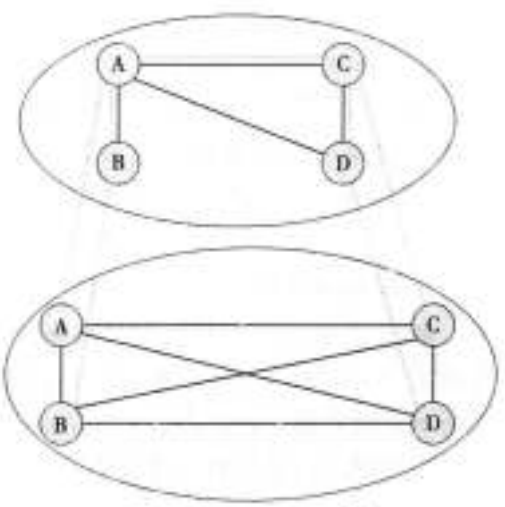


图 9-22 内嵌到覆盖网络中的多播树

构建中间网格覆盖的关键是选择一个大致对应于下层因特网物理拓扑的拓扑图，但我们不得不在没有人告诉我们下层因特网实际看上去如何的情况下去做，因为我们仅在端主机上运行而不是在路由器上。一般的策略是为端主机测量到其他节点的往返延迟，并且决定仅当出现它们喜欢的链路时才把它加到网格中。这一工作过程如下。

首先，假设网格已经存在，每个节点与它直接连接的邻居交换所有其他被认为是网格成员节点的列表。当某个节点从邻居接收到这样一个成员列表后，它将信息加入自己的成员列表中，并且将得到的列表转发到它的邻居。这个信息最终传遍网格，很像距离向量路由协议。

当一台主机想要加入多播覆盖时，它必须知道至少一个覆盖中的其他节点的 IP 地址。

然后它发送一条“加入网格”消息到这个节点。这样就通过到已知节点的一条边将新节点连接到网格。通常，新节点可能将消息发给多个当前节点，从而通过多重连接加入网格。一旦节点由一组链路连接到网格，它就周期性地发送“保活”消息到其邻居节点，让邻居知道它仍想作为组的一部分。

当一个节点离开组时，它发送一条“离开网格”消息到与它直接相连的邻居，并且这一信息通过上面描述的成员列表传播到在网格中的其他节点。另外，节点可能发生故障，或仅仅是决定要默默地退出组，在这种情况下，邻居测定出它不再送出“保活”消息。一些节点的离开对网格的影响很小，但节点应检测到网格由于离开的节点而被分区，它通过给另一个区域的节点发送“加入网格”的消息，建立一条到那个节点的新边。注意多个邻居能同时判定在网格中已经出现分区，导致多个跨区域的边被加入到网格中。

综上所述，我们将得到一个网格，它是原有全连通因特网的子图，但它的性能可能是次优的，因为①初始的邻居选择向拓扑中加入了随机的链路，②分区的修复可能增加当时重要但不是长期有用的边，③组成员可能由于动态的加入和离开而变化，④底层网络条件可能变化。因此需要系统计算每条边的值，导致不断对网格加入新边和删除边。

为增加新边，每个节点 i 周期性地探测那些目前没有连接到网格中的任意成员 j，测量边 (i, j) 的往返时延，然后评估加入此边的效用。如果效用超过某个阈值，链路 (i, j) 被加入到网格。评估加入边 (i, j) 的效用的过程如下：

```
EvaluateUtility(j)
  utility = 0
  对于每个不等于i的成员m
    CL = 通过网格路由到节点m的当前时延
    NL = 在网格中增加边(i, j) 后到节点m的新时延
    if (NL < CL) then
      utility += (CL - NL)/CL
  return utility
```

决定删去一条边与之类似，只是每个节点 i 计算到当前邻居 j 的每条链路的代价的过程如下：

```
EvaluateCost(j)
  Cost_i = 以i作为i的下一跳的成员的数量
  Cost_j = 以i作为j的下一跳的成员的数量
  return max{Cost_i, Cost_j}
```

然后挑出一个代价最低的邻居，并且如果代价低于某个阈值就丢弃它。

最后，因为维护网格所本质上使用的是距离向量路由协议，所以运行 DVMRP 在网格中找出适当的多播树是轻而易举的。注意，尽管不可能证明刚才描述的协议可形成最优网格，但允许 DVMRP 选择可能最佳的多播树，仿真和大量的实践经验显示它的效果不错。

3. 弹性覆盖网络

另一种正在流行的路由覆盖是为传统的单播应用找出可替代的路由。这类覆盖利用了因特网中三角不等式不成立的观测现象。图 9-23 说明了这个问题。在因特网中找出这样的三个站点不足为奇（把它们称为 A、B、C），A 与 B 之间的时延大于 A 与 C 和 C 与 B 之间时延的和，就是说，有时通过中间节点间接发送分组比直接将其发送到目的地要好一些。



图 9-23 三角不等式在网络中不一定成立

怎么会这样呢？BGP 从来不许诺它能找出任意两个站点间的最短（shortest）路由，它只能尝试找出某条（some）路由。更糟糕的是，BGP 路由受政策问题影响严重，比如谁向谁付费以承载流量。这是经常发生的，例如在重要的主干网 ISP 之间的对等点上。简言之，不应该再对因特网中三角不等式不成立感到惊奇。

我们如何利用这一观测现象呢？第一步是认识到在可扩展性与路由算法的最优性之间有一个基本的平衡。一方面，BGP 适合非常大的网络，但常常不选择可能的最佳路由，并且对网络故障的适应是很慢的。另一方面，如果你仅关心在少数站点之中找到最佳路由，那么你在监视可能使用的每条路径的质量方面应该做得好得多，因此允许选择在任一时刻的最佳可能路由。

一个称为弹性覆盖网络（Resilient Overlay Network，RON）的实验性覆盖正好完成这一工作。因为 RON 使用 $n \times n$ 的策略，在每对站点之间密切监视（通过主动探测）路径质量的三个方面——时延、可用带宽和分组丢失率，它的规模仅能达到几十个节点。这既能在任一对节点间选择最佳路由，也能在网络条件变化时快速更改路由。实践表明，RON 能为应用带来适度的性能改进，但更重要的是，它能从网络故障中更快地恢复。例如，2001 年在 64 小时的时间内，一个运行在 12 个节点的 RON 实例检测到持续达 30 分钟的 32 次故障，并且它也能在平均 20 秒内从中恢复。这次实验也表明，仅通过一个中间节点转发数据，通常也就足以从因特网的故障中恢复。

因为 RON 规模不能扩展，所以使用 RON 去帮助任意主机 A 与任意主机 B 通信是不可能的。A 和 B 必须事先知道它们可能通信，然后加入到同一个 RON。然而，RON 在某些情况下似乎是一个好主意，如当连接几十个分布在因特网中的公司站点时，或允许你和你的 50 个朋友为了运行某个应用而建立你们的私有覆盖时。但实际问题是当每人都开始运行他们自己的 RON 时将会发生什么？上百万的 RON 主动探测路径是否会使网络不堪重负？并且当许多 RON 竞争同样的路径时，是否有任何一个 RON 会获得改进的性能？这些问题仍然无法回答。

结论 一般而言，这些覆盖说明了计算机网络的一个核心概念：虚拟化（virtualization）[⊖]。就是说，在由物理资源构建的物理网络之上用抽象（逻辑）资源构建

⊖ 术语虚拟化现在更多地用在数据中心计算的背景下，指的是使用超级管理程序或类似技术进行服务器的虚拟化，但它其实是一个更广泛的概念。

一个虚拟网络是可能的。而且，这些虚拟化的网络可以堆叠，多个虚拟网络可共存在同一层上。每个虚拟网络又为一些用户、应用或高层网络提供有价值的新功能。

9.4.2 对等网

像 Napster® 和 KaZaA 这样共享音乐的应用已将术语“对等”的概念引入流行的行话里。但是对一个系统而言，“对等”究竟意味着什么？当然，在共享 MP3 文件的环境中，它意味着不必从中心站点下载音乐，而是能够直接从因特网中其他存放音乐拷贝的机器上下载。更宽泛地讲，我们可以说对等网允许一个用户团体把他们的资源（内容、存储、网络带宽、磁盘带宽、CPU）放到一个缓冲池中，从而提供对更大的档案库、更大的视音频会议、更复杂的搜索和计算等的访问，这是任何用户不能单独提供的。

通常，当讨论对等网时，一提到像非集中（decentralized）和自组织（self-organizing）这样的属性就意味着单个节点在没有任何集中式协商的情况下，把它们自己组织成一个网络。这些术语也能用于描述因特网本身。然而具有讽刺意味的是，依照这个定义来衡量，Napster 算不上一个真正的对等系统，因为它依赖于已知文件的中心注册目录，用户为了找到提供某个指定文件的机器，必须查找这个目录。只有最后一步（真正下载文件）才发生在两个用户的计算机之间，但这并没比传统的客户/服务器交互多做什么工作。唯一的不同是服务器属于某个像你一样的客户而不是一家大公司。

因此，我们回到最初的问题：就对等网而言什么最吸引人？一种回答是定位一个感兴趣的对象和把它下载到你的计算机上的过程都不必与同一个中央权威机构联系，同时系统能够扩展到几百万个节点。能够以非集中方式完成这两项任务的对等系统原来就是一个覆盖网，其中节点就是那些愿意共享感兴趣的对象（例如音乐和其他各类文件）的主机，而连接这些节点的链路（隧道）代表为了追踪那些你想要的对象而必须访问的机器序列。我们看完下面两个例子之后，这种描述就会变得更加清晰。

1. Gnutella

Gnutella 是一个早期的对等网，它试图区别交换音乐（可能侵犯某人的版权）和一般的共享文件（这应该是件好事，因为我们从 2 岁起就学习共享了）。Gnutella 的吸引人之处在于它是最早不依赖于对象集中式注册目录的系统之一。相反，Gnutella 的参与者把自己安排在一个类似于图 9-24 所示的覆盖网上。就是说，运行 Gnutella 软件（即实现 Gnutella 协议）的每个节点知道同样运行 Gnutella 软件的其他计算机。“A 和 B 相互知道”的关系对应于这个图的边。（后面我们会谈到这幅图是如何形成的。）

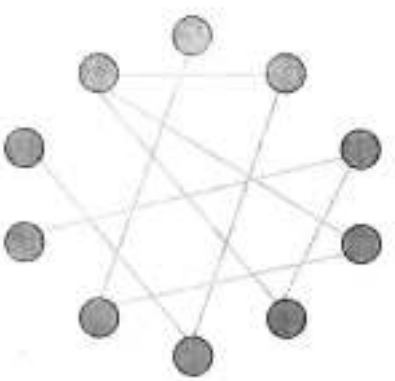


图 9-24　Gnutella 对等网的拓扑示例

每当一个给定节点上的用户想找一个对象时，Gnutella 为它发出一条 QUERY 消息（例如指出文件的名字）给它的邻居。如果一个邻居有此对象，它就用 QUERY RESPONSE 消息回答发出查询的节点，指出从哪里（例如一个 IP 地址和 TCP 端口号）可以下载想要的对象。那个节点随后使用 GET 或 PUT 消息访问对象。如果节点不能解析这个查询，它将 QUERY 消息转发给它的每个

邻居（除发出查询的节点外），然后这个过程重复进行。换句话说，为了定位对象，Gnutella 在覆盖上扩散。Gnutella 为每一个查询设置一个 TTL，所以这种扩散不会无休止地进行下去。

除了 TTL 和查询串，每一条 QUERY 消息包含唯一的一个查询标识符（QID），但是它不包含初始消息源的标识。相反，每个节点维护近期看到的 QUERY 消息的一个记录：QID 和给它发 QUERY 的邻居。它以两种方式使用这一历史记录，首先，如果节点收到的 QUERY 的 QID 与它最近看到的 QID 相匹配，那么它不转发这条 QUERY 消息。这样可以比使用 TTL 更快地结束转发循环。第二，每当节点从下游邻居收到一个 QUERY RESPONSE，它都知道应将响应转发到最初给它发送 QUERY 消息的上游节点。这样，响应会返回初始节点，而没有一个中间节点知道最初是谁想要这个特定的对象。

回到如何生成图的问题，当一个节点加入 Gnutella 覆盖时，当然必须至少知道一个其他节点。新节点至少通过一个这样的链路连到覆盖。然后，节点通过 QUERY RESPONSE 消息了解其他节点，既包括它请求的对象，也包括响应恰好通过的节点。节点自由决定要把所发现的哪一个节点作为邻居。Gnutella 协议提供 PING 和 PONG 消息，通过这两个消息，节点分别探测一个给定的邻居是否仍然存在以及它是否响应。

应该明白，Gnutella 不是一个特别聪明的协议，其后的系统已在努力改进它。可能改进的一个方面是如何传播查询。扩散有一个很好的特性，它保证可以在最少可能的跳数内找到期望的目标，但是它的扩展性不好。它可能随机转发查询，或者根据以往结果的成功概率转发查询。第二个方面是竭力复制对象，因为给定对象的拷贝越多，就越容易找到一个拷贝。另外，你也可能提出一个完全不同的策略，这是我们下面将要考虑的问题。

2. 结构化覆盖

在文件共享系统已经奋力填补 Napster 留下空白的同时，研究团体也为对等网探索了一种替代的设计。与 Gnutella 网络本质上的随机（非结构化）演化方式相对照，我们把这样的网络称为结构化的（structured）。类似于 Gnutella 的非结构化覆盖使用简单的覆盖构造和维护算法，但它们最多提供不可靠的、随机的搜索。相反，为了与一个特定图的结构保持一致，结构化覆盖被设计为允许可靠的和有效的（用概率方式限定延迟）对象定位，但是在覆盖构造和维护中增加了复杂性。

如果你从更高的层次上想一下我们要做什么，那么需要考虑两个问题：①我们如何将对象映射到节点？②我们如何把请求路由到负责存储给定对象的节点？我们从第一个问题开始，这个问题有一种简单的陈述：我们如何将名为 x 的对象映射到某个能为它提供服务的节点 n 的地址上？传统的对等网对于哪个节点存储对象 x 没有控制权，而如果我们能控制对象在网上如何分布，那么以后我们就更容易找到这些对象。

将名字映射到地址的一种常用技术是散列表

$$\text{hash}(x) \rightarrow n$$

意味着首先将对象 x 放到节点 n 上，此后，想要定位 x 的客户只要计算 x 的散列值就能确定它在节点 n 上。基于散列的方法有一个非常好的特性，它易于将对象均匀分布到一组节点上，但是简单的散列算法有一个致命的弱点：我们应该允许 n 有多少个可能的值？（用散列法的术语，应该有多少桶？）我们可以简单地决定，例如有 101 个可能的散列值，使

用模运算的散列函数，即

```
hash(x)
  return x % 101
```

不幸的是，如果有多于 101 个节点愿意存储对象，那么我们就不可能利用到所有节点，另一方面，如果我们选择的模大于最大可能节点数目，那么 x 的某些值将散列到一个不存在的节点的地址。这也是将散列函数转换返回值作为一个实际 IP 地址时不可小觑的问题。

为了解决这个问题，结构化的对等网使用一种称为一致散列法（consistent hashing）的算法，它可以在一个大的 ID 空间上均匀地散列一组对象 x 的散列值。图 9-25 用一个圆圈表示一个 128 位的 ID 空间，其中我们使用算法把对象

hash(object_name)→objid

和节点

hash(IP_addr)→nodeid

放置在这个圆圈上。由于 128 位的 ID 空间是很大的，所以一个对象与一台计算机的 IP 地址不会得到同样的 ID。为了解决这种问题，每个对象由在圆圈上最接近（closest）它的 ID 号对应的节点维护。换句话说，这里的做法是使用高质量的散列函数将节点和对象都映射到同一个大的稀疏 ID 空间，然后根据它们各自标识符数值的接近程度将对象映射到节点上。像一般的散列计算一样，此方法可以将对象均匀分布在节点上，但与一般散列计算不同的是，当一个节点（散列桶）加入或离开时，仅有少量对象需要移动。

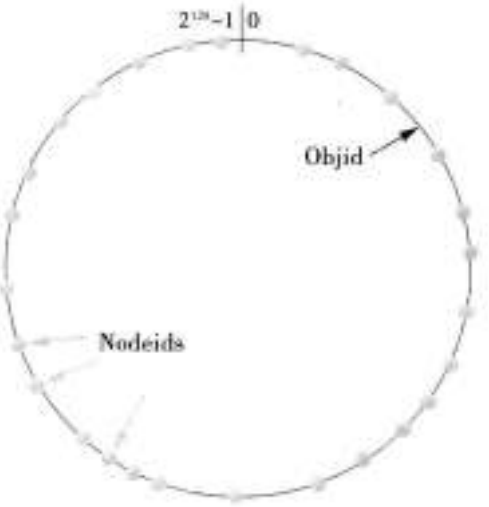


图 9-25　节点和对象都映射（散列）到 ID 空间，对象由空间中最近的节点来维护

现在我们考虑第二个问题：一个想要访问对象 x 的用户如何知道空间中哪个节点的 ID 最接近 x 的 ID？一个可能的答案是每个节点保存一张完整的节点 ID 与相关的 IP 地址的表，但这对于一个大型网络来说并不实用。另一种方法是将消息路由到这个节点上（route a message to this node）！这正是结构化对等网所使用的方法。换句话说，如果我们以一种聪明的方法构建覆盖——与我们需要聪明地为节点的路由表选择表项是一样的——那么我们就可以通过简单地路由到一个节点而发现它，有时也将此方法称为分布式散列表（distributed hash table，DHT），因为从概念上讲，散列表分布在网络中的所有节点上。

图 9-26 展示一个简单的 28 比特的 ID 空间会发生什么。为了使讨论尽量具体，我们考虑名为 Pastry 的特殊对等网所使用的方法，其他系统以相似的方式工作。（更多的例子可以参见

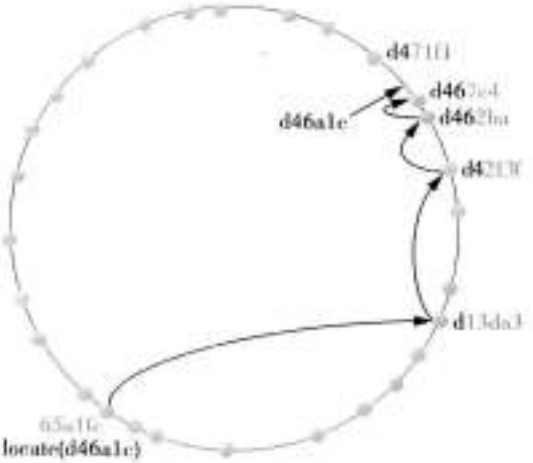


图 9-26　通过对等覆盖网络中的路由来定位对象

"扩展阅读"介绍的文章。)

假设你在ID为65a1fc(16进制)的节点上,而且你想要定位ID为d46a1c的对象。你发现你的ID与对象的ID没有任何共同之处,但是你知道一个至少共享前缀d的节点。那个节点在128位的ID空间中比你更接近对象,所以你将消息转发给它。(我们不给出所转发的消息格式,但你可以把它想象成"定位对象d46a1c"。)假设节点d13da3知道另一个与对象共享更长前缀的节点,它就继续转发消息。这个过程一直继续,直到到达一个不知道更近节点的节点为止,根据定义,这个节点正是存储对象的节点。记住,随着我们在"ID空间"的逻辑移动,消息实际上是通过底层的因特网从一个节点转发到另一个节点。

每个节点维护一个路由表(下面有更多解释)和一小组数字更大或更小的节点ID对应的IP地址。这称为节点的叶集(leaf set),它的意义在于一旦消息被路由到同一叶集中的任一节点时,此节点可以直接转发消息到最终目的地。从另一个角度说,即使存在多个与对象ID共享最长前缀的节点,叶集也能正确而有效地将消息传递到数字上最接近它的节点。此外,因为叶集中的任何节点都可以路由消息,如同同一集合中的任何其他节点一样,所以使路由选择更健壮。这样,如果一个节点不能进一步路由一条消息,那么叶集中的其他邻居也许能。总之,路由过程定义如下:

```
Route(D)
  if D在我的叶集范围内
    转发到叶集中数字上最接近的节点
  else
    令l = 共享前缀的长度
    令d = D的地址中第l位数字的值
    if RouteTab[l,d]存在
      转发到RouteTab[l,d]
    else
      转发到至少具有共享前缀长度且数字上更接近的已知节点
```

路由表RouteTab是一个二维数组。对于ID中的每个十六进制数字(在128位的ID中有32个这样的数字)有一行,对于每个十六进制值(显然有16个这样的值)有一列。i行中的每一项和这个节点共享长度为i的前缀,并且在这一行中,j列中的项在第$i+1$个位置有十六进制值j。图9-27显示了节点65a1fcx的示例路由表的前三行,其中x表示一个未指定的后缀。该图显示了表中每一项所匹配的ID前缀。图中没有显示这个表项的实际值——路由到的下一个节点的IP地址。

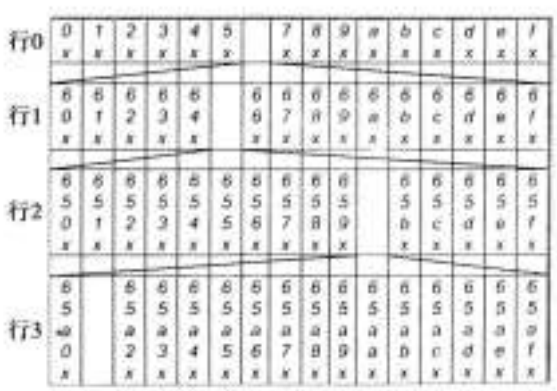


图9-27 ID为65a1fcx的节点的示例路由表

向覆盖增加一个节点就像路由“定位对象消息”到对象一样。新节点必须知道至少一个现有成员。它要求这个成员将“增加节点消息”路由到最接近加入节点 ID 的节点，如图 9-28 所示。通过这个路由过程，新节点可以知道其他共享前缀的节点，并开始填写路由表。随着时间的推移，更多的节点加入到覆盖中，现有节点也可以选择是否在自己的路由表中加入这些节点的信息。当新节点增加一个比现有节点当前表中前缀更长的前缀时，现有节点才做这个选择。叶集中的邻居节点也相互交换路由表，这意味着随着时间的推移，路由信息将遍布覆盖网络。

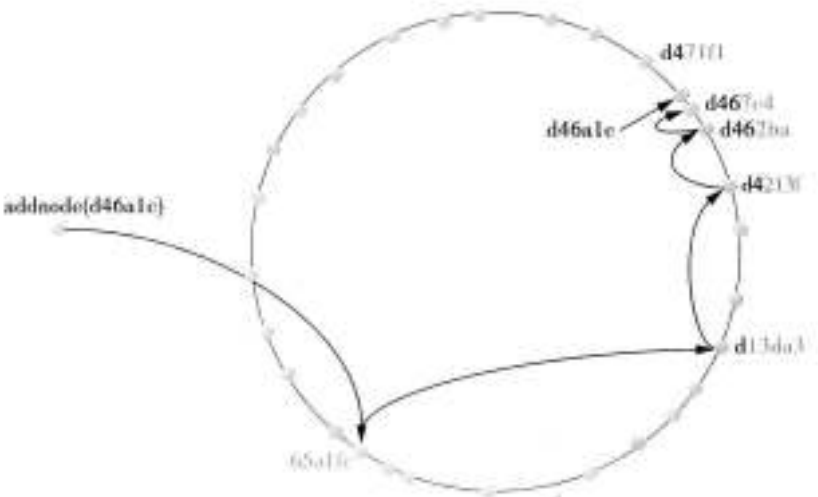


图 9-28　向网络增加一个节点

你可能已经注意到，尽管结构化覆盖对定位一个给定对象时所需的路由跳数给出概率范围——Pastry 中的跳数被限定为 $\log_{16}N$，其中 N 是覆盖上的节点数——每一跳都可能引入潜在的延迟。这是因为每个中间节点可能位于因特网的一个随机位置上。（在最坏的情况下，每个节点在不同的大陆上！）事实上，在世界范围内使用上述算法的覆盖网上，每一跳的预计延迟是因特网上任一对节点间延迟的平均值！幸运的是实际中能做得更好。其思想是在选择每一个路由表项时，在具有适合于该项的一个 ID 前缀的所有节点中，选择在底层物理网中较近的节点。这样做所得到的端到端路由延迟已被证明可能小于源和目的之间的延迟。

最后，目前的讨论集中在对等网上定位对象的一般问题。给定这样一个路由选择架构可以建立不同的服务，例如一个使用文件名作为对象名的文件共享服务。为了定位文件，首先计算名字的散列值，得到对应的对象 ID，然后将一条“定位对象消息”路由到这个 ID。为了提高可用性，系统也可以把它复制到多个节点上。将多个备份存储在给定文件通常被路由到的节点的叶集上，就是这样做的一种方法。记住，即使这些节点在 ID 空间中是邻居，它们也可能在物理上散布于因特网上。因此，虽然一座城市断电可能会在物理上关闭传统文件系统中的文件备份，但在对等网上发生这样的故障时，就可能会保全一个或多个备份。

除文件共享以外的其他服务也能建立在分布式散列表之上，例如多播应用不是从网格构建多播树，而是用结构化覆盖的边构建它，从而可以在几个应用和多播组之间分摊覆盖构建和维护的开销。

3. BitTorrent

BitTorrent 是 Bram Cohen 设计的对等文件共享协议。它是以复制文件或复制被称为片（piece）的文件段为基础的。通常，任何一个片都能从多个对等方下载，即使只有一个对等方有整个文件。BitTorrent 的主要优点是避免了因一个文件只有一个源而引起的瓶颈。考虑到一些计算机通过因特网链路提供文件服务时速率有限，特别是大多数宽带网络的非对称性造成速率极低，此时 BitTorrent 的优势更加有用。BitTorrent 的精华在于复制是下载过程的一个自然的副产品：当一个对等方下载一个特定片时，它就成为该片的另一个源。下载该文件的片的对等方越多，片的复制就越多，从而均衡地分配负载。片的下载是以随机顺序进行的，从而避免所有对等方都发现缺少同一个片集合的情况。

每一个文件都通过它自己的独立 BitTorrent 网络被共享，该网络被称为群（swarm）。（一个群可能潜在地共享一组文件，但我们为了简单只描述一个文件的情况。）一个典型群的生命周期如下。群初始时是一个有文件完整备份的对等方。一个想下载该文件的节点加入该群，称为第二个成员，并开始从第一个对等方下载文件的片。在下载过程中，它成为已下载片的另一个源，即使此时它还没有下载完整个文件。（事实上，对等方在完成下载后立即离开群的现象很常见，虽然鼓励它们待得再长一些。）其他节点加入该群并开始从多个对等方下载片，而不只是第一个对等方（见图 9-29）。

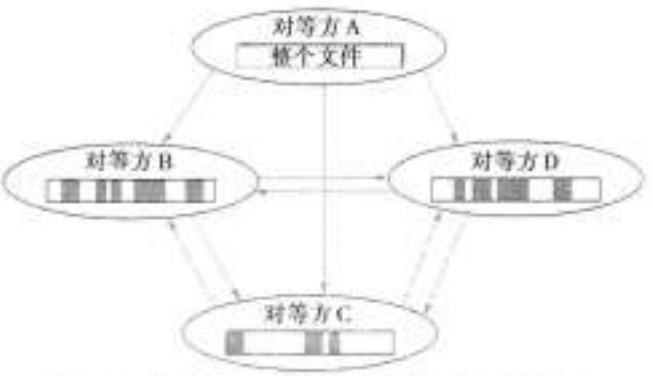


图 9-29 BitTorrent 群中的一个对等方从其他可能还没有整个文件的对等方下载文件

如果该文件保持很高的需求量，那么一群新对等方会替代那些离开的对等方，因此群会永远保持活力；否则，群会萎缩到只包括第一个对等方的情况，直到有新对等方加入该群。

既然我们已经总览了 BitTorrent，我们可以问一下请求是如何被路由到拥有给定片的对等方的。为了提出请求，一个想要成为下载者的节点必须先加入群。它要从下载一个包含文件和群的维护信息的 .torrent 文件开始。.torrent 文件很容易复制，它一般是从 Web 服务器下载的，并通过跟踪网页中的链接来发现。它包含：

- 目标文件的大小。
- 片的大小。
- 为每个片预计算的 SHA-1 散列值（见 8.1.4 节）。
- 群跟踪器（tracker）的 URL。

跟踪器是一个跟踪群的当前成员关系的服务器。我们会在后面看到可以将 BitTorrent 扩展成不需要这个集中点，因为它的存在会引起潜在的瓶颈或故障。

想要成为下载者的节点通过向跟踪器发送一条包含其网络地址以及为自己产生的一个随机对等方 ID 的消息来加入群，成为一个对等方。该消息还包括 .torrent 文件主要部分的 SHA-1 散列值，该值被用作群的 ID。

将新对等方称为 P。跟踪器用一个包含部分对等方 ID 和网络地址的列表来回应 P，P

与其中的一些对等方建立 TCP 连接。注意，P 只是与群的一个子集直接相连，虽然它可能决定联系更多的对等方，甚至向跟踪器请求更多的对等方。为了在建立了 TCP 连接之后与特定对等方建立 BitTorrent 连接，P 发送自己的 ID 和群 ID，而对等方会以自己的 ID 和群 ID 回应。如果群 ID 不匹配或者回应对等方的 IP 不是 P 所期望的 ID，该连接就终止。

所得到的 BitTorrent 连接是对称的：每一端都能从另一端下载文件。每一端开始都会给对方发送一个位图来报告自己拥有的片，因此每一个对等方都知道对方的最初状态。每当下载者 D 下载完一个片时，它就向与它直接相连的对等方发送一条标识该片的消息，这样那些对等方就能更新有关 D 的状态的内部表示。这就是对下载请求如何被路由到拥有该片的对等方这个问题的最终答案，因为它意味着每个对等方都知道哪个直接相连的对等方拥有该片。如果 D 需要一个在所有连接中都不存在的片，那么它会连接到更多的或不同的对等方（它能从跟踪器得到更多的对等方），或者先处理其他片并希望它的一些连接能够从它们的连接得到那个片。

如何将对象——在本例中是片——映射到对等方节点？当然，每个对等方最终得到所有片，因此该问题实际上是在对等方拥有所有片以前的某个给定时间，它拥有哪些片，或者也可以等价描述为对等方下载片的顺序。答案是它们以随机顺序下载片，以防止它们拥有任何其他对等方的片的严格的子集或超集。

目前所描述的 BitTorrent 使用一个集中的跟踪器，它会成为群的故障点，并最终可能成为性能瓶颈。同时，提供跟踪器对于想通过 BitTorrent 提供文件的人来说是一件烦人的事。新版本的 BitTorrent 还支持使用基于 DHT 实现的无跟踪器群。具有无跟踪器能力的 BitTorrent 客户端软件不但要实现一个 BitTorrent 对等方，还要实现一个我们称为对等方探测器（peer finder，相应的 BitTorrent 术语就是一个简单的节点）的部分，用于帮助对等方发现其他对等方。

对等方探测器构成它们自己的覆盖网络，用它们自己的协议在 UDP 上实现一个 DHT。另外，对等方探测器网络包含相关对等方属于不同群的对等方探测器。换句话说，每个群形成了 BitTorrent 对等方的不同网络，而对等方探测器网络跨越多个群。

对等方探测器随机产生自己的探测器 ID，与群 ID 的长度相同（160 位）。每个探测器维护一个适度的表，其中包含具有与自己的 ID 接近的主要探测器（和它们的相关对等方），再加上一些 ID 较远的探测器。下面的算法确保了具有与给定群 ID 接近的探测器很可能知道来自该群的对等方，这个算法同时提供了一种查询方法。当探测器 F 需要找到来自特定群的对等方时，它就向表中具有与群 ID 接近的探测器发送一个请求。如果所联系的探测器知道来自该群的任何对等方，它就以对等方的联系信息作为回应。否则，它以表中接近该群的探测器的联系信息作为回应，这样 F 就能以迭代方式查询那些探测器。

搜索完成后，因为没有接近群的探测器，所以 F 将自己的联系信息和相关对等方插入到与群最接近的探测器。实际结果是一个特定群的对等方进入与该群接近的探测器的表中。

上述方案假设 F 已经是探测器网络的一部分，也就是说它已经知道如何与其他探测器联系。这种假设对于以前运行的探测器安装是正确的，因为它们会保存其他探测器的信息，甚至在执行过程中也会保存。如果群使用跟踪器，它的对等方能够告诉它们的探测器关于其他探测器的信息（反转对等方和探测器的角色），因为 BitTorrent 对等协议扩展后能够交换探测器联系信息。但是一个新安装的探测器如何发现其他探测器？用于无跟踪器

群的.torrent文件包含一个或多个探测器的联系信息，而不是跟踪器的URL，以便适应那种情况。

BitTorrent的一个非同寻常的方面是它正面解决公平问题或好“网络公民”问题。协议通常依赖于每一个用户的良好行为但并不能强制。例如，一个肆无忌惮的以太网对等方使用一种比指数回退算法更贪婪的回退算法来得到更好的性能，或者一个肆无忌惮的TCP对等方通过不参与拥塞控制来得到更好的性能。

BitTorrent所依赖的良好行为是对等方为其他对等方上传片。因为典型的BitTorrent只是想尽快下载文件，因此实现一个试图下载所有片而只上传尽量少的片的对等方——这是一个坏对等方——是很有诱惑力的。为了阻止坏行为，BitTorrent协议包含了一种机制，允许对等方互相奖励或惩罚。如果一个对等方因为没有很好地为另一个对等方上传片而造成行为不当，第二个对等方可以阻塞（choke）这个坏对等方：它可以决定至少暂时不向这个坏对等方上传片，并发送一条消息告诉它。还有一种消息类型用于告诉一个对等方它已经被解阻塞了。阻塞机制也被对等方用来限制活动BitTorrent连接的数量，以便保持良好的TCP性能。有很多可能的阻塞算法，而设计一个好的算法是一种艺术。

9.4.3 内容分发网络

我们已经看到了运行在TCP上的HTTP是怎样让Web浏览器从Web服务器上获得网页的。然而，任何无休止地等待一个网页返回的人都知道这样的系统绝非完美。考虑到目前因特网的主干建立在OC-192（10Gbps）链路上，出现这种情况的原因并不明显。普遍的共识是当你下载网页时，系统中存在四个潜在的瓶颈：

- 第一英里：因特网中可能有高容量链路，但当你通过56kbps的调制解调器或性能很差的3G无线链路连接时，它不能帮你更快地下载一个网页。
- 最后一英里：将服务器与因特网相连的链路可能因请求太多而超载，虽然链路的带宽很高。
- 服务器本身：服务器具有有限的资源（CPU、内存、磁盘带宽等），会因为过多的并发请求而超载。
- 对等点：少数共同实现因特网主干的ISP可能在其内部有高带宽管道，但是没有什么推动力使它们对其对等点提供高容量连接。如果你连接到ISP A而服务器连接到ISP B，那么你请求的页面可能会在A和B相互对等的点被丢弃。

要解决第一个问题，除了你之外别人做不了什么。但利用备份技术有可能处理其他问题。做这件事的系统通常称为内容分发网络（Content Distribution Network，CDN）。Akamai可能是最著名的CDN。

CDN的思想是在地理上分布一组服务器代理（server surrogate）来缓存那些通常在一组后端服务器（backend server）上维护的网页。于是在大型新闻事件发生时，不必让几百万人为了连接www.cnn.com而无休止地等待——这种情形称为突发聚集（flash crowd）——把这样的负载分布到多台服务器上是有可能的。此外，不必遍历多个ISP才到达www.cnn.com。如果这些代理服务器分散在所有主干网ISP上，那么就有可能到达其中一台服务器而不需经过对等点。显然对于一个想要为其网页提供更高效访问的网站，在因特网上维护数以千计的代理服务器的代价太昂贵了。商用CDN为很多网站提供这项服务，从而使这个代价由很多客户分担。

尽管我们把它们称为代理服务器，事实上，把它们看作是高速缓存也是正确的。如果没有客户所请求的页面，它们会向后端服务器请求。然而在实践中，后端服务器提前把它们的数据复制给代理，而不是等代理在需要时请求。同样，只有与动态内容相反的静态网页才分布到代理上。用户需要到后端服务器上获取所有频繁变化的内容（例如体育比分和股票报价）或是由某种计算结果给出的内容（例如一条数据库查询）。

拥有地理上分布的一个庞大的服务器集合并不能完全解决问题。为了完整起见，CDN还需提供一组能够把客户请求转发给最合适的服务器的重定向器（redirector），如图9-30所示。重定向器的主要目标是为每个请求选择能为客户提供最短响应时间（response time）的服务器，其次是让整个系统在底层硬件（网络链路和网页服务器）能够支持的情况下每秒钟处理尽可能多的请求。在一段给定时间内所能满足的请求的平均数量——称为系统吞吐量（system throughput）——是系统负载很重时的主要问题，例如，当突发聚集访问少数网页或分布式拒绝服务（DDoS）攻击者瞄准某个网站时，就像2000年2月CNN、Yahoo和其他几个著名的网站遇到的情况一样。

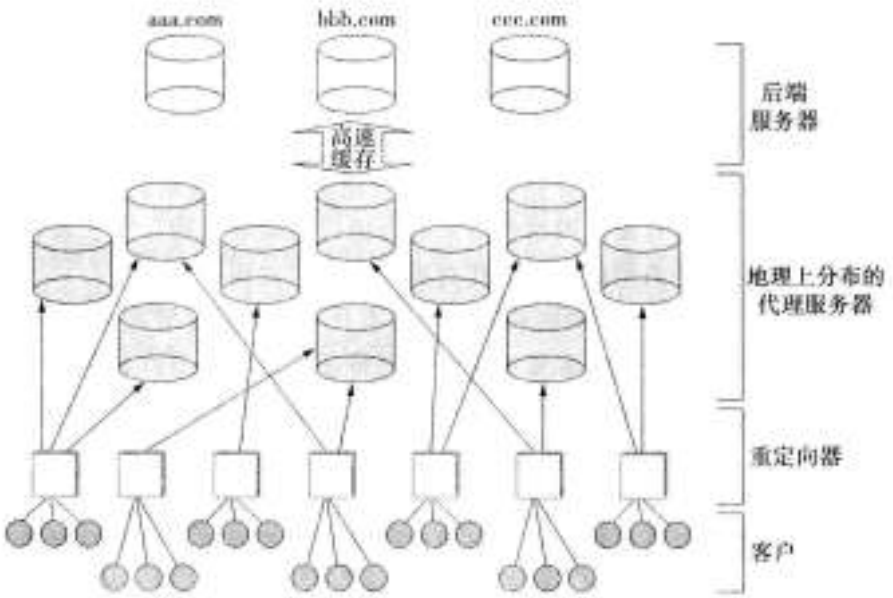


图9-30 一个内容分发网络（CDN）的组成部分

CDN根据多个因素来决定如何分发客户请求。例如，为了减少响应时间，重定向器可能根据网络接近程度（network proximity）来选择服务器。相反，为了提高系统的总吞吐量，希望在一组服务器上均匀地平衡（balance）负载。如果分发机制把位置（locality）作为考虑因素就能提高吞吐量并缩短响应时间，也就是说，选择一个在它的高速缓存中可能已包含被请求网页的服务器。CDN应采用的因素组合是公开争论的问题。本节考虑一些可能性。

1. 机制

在目前的描述中，重定向器只是一个抽象的功能，尽管这听起来像要求路由器做的事，因为从逻辑上讲它像路由器转发分组一样转发请求消息。事实上，有多种机制可以用来实现重定向。注意，为方便讨论，我们假设每个重定向器都知道所有可达服务器的地址。（从这里开始我们丢弃“代理”这个词而只称其为一组服务器。）在实践中，服务器会

出现和消失，可以通过某种形式的带外通信使这些信息保持最新。

首先，可以通过调整DNS给客户返回不同服务器地址的方法实现重定向。例如，当客户要求解析域名www.cnn.com时，DNS服务器会返回保存CNN的Web网页的已知具有最轻负载的服务器IP地址。另一种方法是，对于给定的一组服务器，DNS服务器可能只是以轮转的方式返回地址。注意基于DNS的重定向机制的粒度通常是网站级（例如cnn.com）而不是某个特定的URL（例如http://www.cnn.com/2002/world/europe/06/21/william.birthday/index.html）。然而，当返回一个嵌入式链接时，服务器会重写URL，这样就有效地将客户指向最适合那个特定对象的服务器。

商用CDN本质上结合使用URL重写和基于DNS的重定向。出于可扩展性原因，高级DNS服务器先指向一个能返回实际服务器地址的区域级DNS服务器。为了对变化做出快速响应，DNS服务器把它们返回的资源记录的TTL改成一段很短的时间，比如20秒。这样做是必要的，使得客户不再缓存结果，并因此不能返回DNS服务器查找最新的URL到服务器的映射。

另一种可能性是使用HTTP的重定向特性：客户向服务器发一条请求消息，服务器响应一个为获得网页的客户所应联系的新的（更好的）服务器。不幸的是，基于服务器的重定向会导致在因特网上的额外往返时间，更糟糕的是，服务器容易因为重定向任务而使本身超载。然而，如果客户附近有一个节点——例如一个本地Web代理——知道哪些服务器可用，它就可以拦截请求消息并指示客户从合适的服务器请求网页。在这种情况下，要么重定向系统需要处在所有离开网站的请求都会通过的咽喉要道，要么客户将不得不通过显式定位代理（就像一个典型的非透明代理那样）的方式协作。

到了这里，你可能在想CDN到底和覆盖网络有什么关系，尽管把CDN看成是覆盖有一点夸大其词，但它们的确有一个非常重要的共同特征。像覆盖节点那样，一个基于代理的重定向器会做出应用级路由选择决定。它不会根据地址以及它对网络拓扑的了解来转发分组，它会根据URL以及它对某组服务器的方位和负载的了解来转发HTTP请求。现今的因特网体系结构并不直接支持重定向——“直接”的意思是指客户把HTTP请求发给重定向器，由重定向器转发给目标——所以通常让重定向器返回合适的目标地址并让客户自行联系服务器来间接实现重定向。

2. 策略

现在我们来考虑一些重定向器转发请求时可能用到的典型策略。实际上，我们已经提出了一个简单的策略——轮转策略。一个类似的方案是简单地在可用服务器中随机选择一台服务器。两个方法都能够很好地把负载均匀散布到CDN上，但它们并没有很好地缩减客户感受到的响应时间。

很明显，这两个方案都没有把网络接近程度考虑在内，同时，它们还忽略了位置。也就是说，对同一URL的请求被转发给不同服务器，使得由所选服务器内存中的缓存来提供网页的可能性变小了。这使服务器不得不从磁盘甚至后端服务器上取回网页。一组分布在各地的重定向器如何在不经过全局协调的情况下把对同一网页的请求发往同一服务器（或一小组服务器）？答案出奇的简单：所有重定向器都使用某种形式的散列法把URL确定地映射到一个小的取值范围。这个方法的主要好处是不需要为了协调操作而进行任何重定向器之间的通信，不管哪个重定向器收到URL，散列过程都能产生相

同的输出。

那么什么是好的散列方案？经典的取模（modulo）散列方案——用散列法处理每个URL时，以服务器数目作模数——不适合这种环境。这是因为如果服务器数目改变，取模运算的结果会使得越来越少的网页能保持分配到相同服务器。虽然我们不希望服务器集合频繁变化，但在集合中添加新服务器将引起人们不愿看到的大规模重新分配。

另一个办法是使用9.4.2节中讨论过的一致散列（consistent hashing）算法。具体来讲，每个重定向器先把所有服务器散列变换到单位圆中，然后对于每个到达的URL，重定向器把URL也散列变换成单位圆上的一个值，最后URL被指定给圆上离它的散列值最近的服务器。如果这个方案中的一个节点出现故障，其负载将被转移给它的邻居（在单位圆上）。这样添加或移走一台服务器只会使请求分配在局部发生变化。注意，与对等网的情况不同的是，对等网上一条消息从一个节点路由至另一节点为的是找到其ID离目标最近的服务器，每个重定向器知道这组服务器是如何映射到单位圆上的，所以它们可以各自独立地选择“最近”的服务器。

这种策略很容易通过扩展以考虑服务器负载。假设重定向器知道每个可用服务器当前的负载，这些信息不一定是最新的，但我们可以想象重定向器只对前几秒内它向某个服务器转发请求的次数进行计数，并把这个计数作为该服务器当前负载的估计值。当收到一个URL时，重定向器散列变换这个URL加上每个可用的服务器，并将结果值排序。这个排序列表有效地定义了重定向器所考虑的可用服务器的顺序。于是重定向器顺序访问列表直到它找到负载低于某一阈值的服务器。与普通的一致散列法相比，这种方法的好处在于每个URL都有不同的服务器顺序与之对应。所以如果一台服务器发生故障，它的负载将被均匀地分布到其他机器上。这种方法是高速缓存阵列路由协议（Cache Array Routing Protocol，CARP）的基础，以下是其伪代码表示：

```
SelectServer(URL,S)
  for 服务器集合S中的每台服务器S,
    weight,←hash(URL,address(s,))
  排序 weight
  for 递减序列 weight, 中的每台服务器s,
    if Load(s,)<threshold then
      return s,
  return weight 最高的服务器
```

随着负载的增加，这个方案从只使用列表上的第一台服务器变为将请求分散到多台服务器上。一些通常由“繁忙”服务器处理的网页也将由相对不太忙的服务器处理。因为这个过程的依据是服务器负载的聚集而不是单个网页的受欢迎程度，保存某些受欢迎网页的服务器可能找到更多分担其负载的服务器而不是共同保存不受欢迎网页的服务器。在此过程中，一些不受欢迎的网页会在系统中被复制，这仅仅是因为它们恰好主要被存储在繁忙的服务器上。同时，如果一些网页变得非常受欢迎，可以想象，所有系统中的服务器都可能有责任提供它们。

最后，可以用至少两种不同的方法把网络接近程度引入等式中。第一种方法是通过监视服务器响应请求的时间长短并把这个测量值作为“服务器负载”参数运用到前面的算法中，这使得服务器负载和网络接近程度之间的区别变得模糊。这种策略倾向于选择邻近的或负载轻的服务器而不是远处的或负载重的服务器。第二种方法是在较早的阶段限制候选的服务器集合使其只包括附近的服务器，这样就使接近程度成为决定因素之一。更难的问

题是如何在可用的很多服务器中确定哪个离得较近。一种方法是只选择与客户在相同 ISP 上的可用服务器。一种稍微复杂一点的方法是查看由 BGP 生成的自治系统图，并且只选择那些从客户出发，在一定跳数内可达的服务器作为候选者。在网络接近程度和服务器缓存位置之间寻找平衡是正在研究的课题。

9.5 小结

我们已经看到了两种应用最广泛的基于客户/服务器的应用协议：用于交换电子邮件的 SMTP，用于在万维网中漫游的 HTTP。我们看到了应用到应用通信如何驱动诸如 SOAP 和 REST 等新的协议开发框架的发明。我们还研究了会话控制协议，如 SIP 和 H.323，它们用于控制多媒体应用，如 IP 语音。除了这些应用协议，我们还看到了一些关键的支撑协议：用于域名系统的 DNS 协议，用于为网络管理查询远程节点的 SNMP 协议。最后，我们看到了新兴的应用——包括覆盖、对等网和内容分发网络——以创新的方式融合应用处理和分组转发。

应用协议也是很奇怪的。在许多方面，传统的客户/服务器应用像是另一层传输协议，不同的是它们有内嵌的特定应用的知识。你可以认为它们不过是特殊化的传输协议，并且这些传输协议层层叠加直至产生应用所需要的准确服务。类似地，覆盖和对等网协议可以看作是提供一种可选的路由体系结构，但同样也是为一个特定应用的需求而定制的。我们从这个观察结论所得到的实际经验是设计应用层协议确实与设计核心网络协议没有什么区别，而且对核心网络协议了解得越多，那么对设计应用层协议越有帮助。我们还发现系统方法——理解功能及组件是如何交互来构建一个完整系统的——在设计应用时应该与在设计联网其他方面时用得一样多。

接下来会发生什么：新的网络体系结构

在应用协议领域正确指出一个特定的开放性问题是困难的，因为每天都出现新的应用，故而整个领域都是开放的，并且这些应用的网络需求是与应用相关的。对网络设计者的真正挑战是清楚随网络变化的应用的需求是什么，以及这些变化如何促使我们开发传输协议，以及在网络路由器中加入功能。

开发新的传输协议是一个相对容易处理的问题。你也许不能让 IETF 认可你的传输协议等同于 TCP 或 UDP，但没有什么能阻止你设计出世界上最好的、与新的运行在 UDP 之上的端到端协议相结合的多媒体应用，就像 RTP 所做的那样。

另一方面，将特定应用的知识放入网络中（放入路由器中）是一个更困难的问题。这是因为要使一个特定应用生效，需要将所有新的网络服务或功能装入许多（如果不是全部）因特网路由器中。覆盖网络提供一种无需全部（或者甚至任一个）路由器的协作就可将新的功能引入到网络中的方法。但从长远看，我们可以期望改变底层网络体系结构以适应这些覆盖。我们通过 RON 看到了这个问题——RON 和 BGP 路由选择如何相互交互——并且随着覆盖网络的日趋盛行我们预料它会成为一个普遍的问题。

一种可能性是不会出现另一种固定的（fixed）体系结构，而下一代网络体系结构将会有特别强的适应性。在这样的前提下，并不是定义一个仅传送数据分组的体系结构，网络体系结构可能允许分组不但携带数据，也将携带代码（或可能是指向代码的指针）以便告诉路由器如何处理分组。这样的网络引发许多问题，而不仅仅是在应用程序能有效地对路

由器编程的情况下如何加强网络世界安全的问题。另一种可能性是网络的虚拟化成为规范，一些“片”提供健壮易懂且经过全面调试的服务，而其他的则用于实验功能，这也是研究团体正在追求的方向。

扩展阅读

我们的第一篇文章提供了有关万维网早期设计和实现的有趣的综述，这是由万维网的发明者在万维网席卷全球前写的。Mockapetris 和 Dunlap 对 DNS 的发展做出了很好的描述。覆盖 CDN 和对等网络在近几年中得到了充分研究，最后六篇研究论文对了解这些问题提供一个很好的起点。

- Berners-Lee, T., R. Caillia, A. Luotonen, H. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM* 37 (8), pages 76-82, August 1994.
- Mockapetris, P., and K. Dunlap. Development of the Domain Name System. *Proceedings of the SIGCOMM'88 Symposium*, pages 123-133, August 1988.
- Karger, D. et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *Proceedings of the ACM Symposium on Theory of Computing*, Pages 654-663, May 1997.
- Chu, Y., S. Rao, and H. Zhang. A case for End System Multicast. *Proceedings of the ACM SIGMETRICS'00 Conference*, pages 1-12, June 2000.
- Andersen, D. et al. Resilient overlay networks. *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131-145, October 2001.
- Rowstron, A., and P. Druschel. Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility. *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 188-201, October 2001.
- Stoica, I. et al. Chord: A scalable peer-to-peer lookup service for Internet applications. *Proceedings of the ACM SIGCOMM Conference*, pages 149-160, August 2001.
- Ratnasamy, S. et al. A scalable content-addressable network. *Proceedings of ACM SIGCOMM'01*, pages 161-172, August 2001.

SMTP 最早在 RFC821 [Pos82] 中定义，当然，RFC 822 是 RFC 822 [Cro82]。它们曾经是 IETF 术语，分别被 [Kle01] 和 [Res01] 废弃。MIME 在一系列 RFC 中定义，最新的版本在 RFC 2045 [FB96] 中定义，还有几个附加 RFC 补充细节。

HTTP 的 1.0 版本在 RFC 1945 [BLFF96] 中说明，最新版本是 1.1，在 RFC 2616 [FGM+99] 中定义。Mogul [Mog95] 解释了 HTTP 1.1 的持久连接。有很多关于 Web 性能（尤其是 Web 缓存）的文章。其中的一个好例子是 Danzig 关于 Web 通信量及高速缓存有效性方面的论文 [Dan98]。Roy Fielding 的博士论文 [Fie00] 是 REST 的最佳参考。

SIP 在 RFC 3261 [SCJ+02] 中定义，其中包含了很有帮助的指南部分以及该协议的详细规范，就像 MIME 一样，有很多其他的 RFC 扩展了 SIP 协议。

有大量讨论命名以及资源发现（首先找出存在什么资源）问题的文章。对命名的一般研究可在 Terry [Ter86]、Comer 和 Peterson [CP89]、Birrell 等 [BLNS82]、Saltzer [Sal78]、Shoch [Sho78] 和 Watson [Wat81] 中找到；基于属性的（描述）命名系统在 Peterson [Pet88] 和 Bowman 等 [BPY90] 中描述；资源发现是 Bowman 等 [BDMS94]

的主题。

网络管理是 IETF 全力投入的广泛而重要的领域。有 100 多个 RFC 文档描述 SNMP 和 MIB 的各种问题。然而，两篇关键的参考文献是 RFC 2578 [MPS99]（它定义了 SNMP 版本 2（SNMPv2）的管理信息格式）和 RFC 3416（它定义了 SNMPv2 的协议操作）。与 SNMP/MIB 相关的许多其他 RFC 定义对 MIB 变量核心集的扩展，例如那些针对特定网络技术或特定厂商产品的变量。Perkins 等 [PM97] 提供了对 SNMP 和 MIB 的一个很好的介绍。

美国国家科学研究委员会关于因特网僵化的报告可以在 [NRC01] 中找到。另外，使用覆盖网络引入创新技术的建议是由 Peterson，Anderson，Culler 和 Roscoe [PACR02] 提出的。Savage，Hoffman，Snell 和 Anderson [SGH+99] 提出支配 BGP 路由的最早的例子。使用 DNS 均衡服务器负载的思想参见 RFC1794 [Bri95]。关于 Web 高速缓存与重复服务器的问题在 Rabinovich 和 Spatscheck 的书 [RS02] 中有详尽的论述。Wang、Pai 和 Peterson [WPP02] 研究了重定向器的设计。

最后，我们推荐下面的实时参考来帮助读者跟踪万维网的飞速发展以及有关 Web 服务的丰富信息。

- http://www.w3.org：万维网联盟。

习题

1. 讨论你将如何重写 SMTP 或 HTTP 以利用一个启发式的通用请求/响应协议。能将一个类似的持久连接从应用层协议移至这样的传输层协议中吗？还有哪些其他的应用任务可以移至这个协议中？
2. 大部分 Telnet 客户能连接到端口 25（SMTP 端口）而不是 Telnet 端口。使用这样一个工具，连接到 SMTP 服务器并给你自己（或获得许可的其他人）发送某个伪造的电子邮件。然后检查首部以找出消息不真实的证据。
3. 为了让 SMTP 和一个像 sendmail 那样的邮件后台处理程序提供一些措施以提防上道习题所说的邮件伪造者，可以使用（或增加）SMTP 的什么特性？
4. 找出 SMTP 主机如何处理来自另一方的未知命令，特别是这种机制如何允许协议的改进（如“扩展的 SMTP”）。你可以读 RFC，也可以像习题 2 那样连接一个 SMTP 服务器并测试它对不存在命令的响应。
5. 正如书中描述的，SMTP 包含几条小消息的交换。在大多数情况下，服务器的响应不影响客户的后续发送。客户因而可以实现命令管道（command pipelining）：在单一的消息中发送多条命令。
   (a) 对于哪些 SMTP 命令，客户需要注意服务器的响应？
   (b) 假设服务器用 gets() 或等价的函数读取每一条客户消息，它将读取一个串直到遇见一个换行符（LF）为止。当它检测到客户已使用命令管道之后，需要做些什么？
   (c) 然而管道可被某些服务器中断，研究客户怎样协商它的使用。
6. 像 MIME 这样的协议所面对的中心问题之一是数量巨大的可用数据格式。参考 MIME RFC 找出 MIME 是怎样处理新的或特定系统的图像和文本格式的。
7. MIME 使用 multipart/alternative 语法来支持对相同内容的多个表示。例如，文本能以 text/plain，text/richtext 和 application/postscript 的形式发送。虽然把本来的格式放在普通文本格式之前会使实现更容易，但为什么你仍认为普通文本格式是首选？
8. 查看有关 MIME 的 RFC，找出 base64 编码是如何处理长度不是 3 字节的偶数倍的二进制数据的。
9. POP3 邮局协议只允许客户使用口令认证方法获取电子邮件。通常，客户发送电子邮件就是简单地将其送到服务器，然后期望邮件被转发。

(a) 解释为什么邮件服务器经常不再允许来自任意客户的此类转发。

(b) 提出一个 SMTP 选项来认证远程客户。

(c) 找出可用于解决这一问题的现有方法。

10. 在 HTTP 1.0 版本中，一个服务器通过关闭连接来标记传输的结束。从 TCP 层的角度解释为什么这样做会引起服务器的问题。找出 HTTP1.1 版本中怎样避免这一问题。一个通用的请求/响应协议会怎样解决这个问题？

11. 找出如何配置 HTTP 服务器以便消除 404 not found 消息，取而代之的是返回一个默认的（并且希望是更友善的）消息。判断这样的特性是否是协议的一部分或实现的一部分，或者在技术上协议是允许这样做的。（有关 apache HTTP 服务器的文档可在 www.apache.org 上找到。）

12. 为什么 HTTP GET 命令

```
GET http://www.cs.princeton.edu/index.html HTTP/1.1
```

含有所连接的服务器的名字？服务器还不知道它的名字吗？像习题 2 那样使用 Telnet 连接到一台 HTTP 服务器的端口 80，并且找出如果省略主机名后会发生什么事情？

13. HTTP 在连接的服务器端启动一个 close() 函数，然后它必须在 TCP 状态 FIN-WAIT-2 等待客户关闭另一端。TCP 协议中的什么机制能帮助 HTTP 服务器处理不合作的或实现很糟糕的不在本端关闭连接的客户。如果可能的话，找出这种机制的编程接口，并指明 HTTP 服务器可以怎样使用它。

14. 假设一个非常大的 Web 站点需要一种机制，能使客户按照适当的度量来访问多个 HTTP 服务器中"最接近"的 HTTP 服务器。

(a) 讨论在 HTTP 内开发这种机制。

(b) 讨论在 DNS 内开发这种机制。

比较两种方法。两种方法都能在不升级浏览器的前提下工作吗？

15. 像 FTP 和 SMTP 这样的应用协议是从零开始设计的，并且它们看上去工作得相当好。那么需要 Web 服务协议框架的商家到商家以及企业应用集成协议工作得怎么样呢？

16. 选择一个有等同 REST 和 SOAP 接口的 Web 服务，就像 Amazon.com 提供的那些。比较等同的操作是如何以两种风格被实现的。

17. 得到某些 SOAP 风格的 Web 服务的 WSDL，并选择一个操作。对实现该操作的消息，辨识各个字段。

18. 假设一个大型会议中某些接收者能够以比其他人高得多的带宽接收数据。可以通过什么实现来解决这个问题？（提示：同时考虑会话公告协议（Session Announcement Protocol，SAP）和使用第三方混频器的可能性。）

19. 怎样编码两个分组内的音频（或视频）数据，使得当其中一个分组丢失时，分辨率只是降低到只有一半带宽时的分辨率？解释如果使用 JPEG 类型的编码，为什么这样做要困难得多。

20. 解释统一资源定位符（URL）和统一资源标识符（URI）之间的关系。给出一个 URI 不是 URL 的例子。

21. 找出 DNS MX 记录除了提供邮件服务器的别名之外，还提供哪些其他特性，毕竟服务器的别名也可由 DNS CNAME 记录提供。MX 记录用于支持电子邮件，一个类似的 Web 记录可用于支持 HTTP 吗？

22. ARP 和 DNS 都依赖高速缓存；ARP 高速缓存记录生存期一般是 10 分钟，而 DNS 高速缓存记录生存期是几天。解释为什么会存在这种差别。DNS 高速缓存记录的生存期太长会产生什么不希望的后果？

23. IPv6 通过允许硬件地址作为 IPv6 地址的一部分来简化 ARP。这如何使 DNS 的工作复杂化？这将怎样影响你找到本地 DNS 服务器。

24. DNS 服务器也允许反向查询。给出一个 IP 地址 128.112.169.4，它被逆转换成一个文本串 4.169.112.128.in_addr.arpa，并且使用 DNS 的 PTR 记录查找（PTR 的域层次结构类似于地址的

域层次结构)。假设你想要根据主机名鉴别分组的发送方，并确信源 IP 地址是真的。解释下列情况下的不安全性：将源 IP 地址转换到上述主机名，然后将名字与可信任主机的给定列表进行对比。(提示：你信任谁的 DNS 服务器?)

25. 一个域名(如 cs. princeton. edu)和一个 IP 子网号(如 192.12.69.0)之间有什么关系?必须用相同的名字服务器识别一个子网中的所有主机吗?像上一道习题中的反向查询情况如何?

26. 假设一台主机选择一台不在它的组织内的名字服务器做地址解析。对在任何 DNS 高速缓存中都找不到的查询，与用本地名字服务器相比，这样做何时不增加总通信量?这样做何时会得到一个较高的 DNS 高速缓存命中率和较少的总通信量?

27. 图 9-17 显示了名字服务器的分层结构。如果一个名字服务器服务于多个区域，怎样表示这种分层结构?在这种设置下，如何将名字服务器的分层结构联系到区域的分层结构?怎样处理每个区域可以有多个名字服务器的情况?

28. 使用 whois 工具/服务找出谁负责你的站点，至少是在 InterNIC 涉及的范围。通过 DNS 名字和通过 IP 网络号查询你的站点，对于后者可能不得不尝试另一个 whois 服务器(如 whois-h whois. arin. net. .)。也试一试 princeton. edu 和 cisco. com。

29. 许多较小的机构让第三方维护它们的 Web 站点。怎样使用 whois 发现是否有这样的情况?如果有，怎样使用 whois 发现第三方的身份?

30. 现存 DNS. com 分层结构的特性之一是它非常"宽阔"。
    (a) 提出一个比 .com 分层结构更加分层化的重组方案，你能预料到这一提议会遇到什么反对意见吗?
    (b) 当大部分 DNS 的域名含有 4 层或 4 层以上时，与现有的两层域名相比结果怎样?

31. 假设是另外一种情况，我们抛弃所有 DNS 分层结构，并且简单地将所有 .com 项移至根名字服务器：www. cisco. com 可能变成 www. cisco，或可能就是 cisco。从总体上讲，这将如何影响到根名字服务器的通信量?对一种把 cisco 这样的名字转换成 Web 服务器地址的场景，它是怎样影响通信量的?

32. 在改变 IP 地址(如 Web 服务器的主机名的 IP 地址)时，涉及 DNS 高速缓存方面的问题是什么?怎样使这些问题的影响最小?

33. 采用一个适当的 DNS 查询工具(如 dig)并且禁止递归查询特性(如用 +norecursive)，这样当你的工具向一个 DNS 服务器发送一个询问，并且服务器不能根据自己的记录完全回答请求时，服务器回送在查询序列中的下一台 DNS 服务器，而不是自动转发查询到下一台服务器。然后实现如图 9-18 所示的手工方式名字查找，用主机名 www. cs. princeton. edu 尝试一下。列出所接触的每一台中间名字服务器。你可能也需要指明查询是针对 NS 记录而不是一般的 A 记录。

34. 找出是否有一个可用的 SNMP 节点来回答你发送的查询。如果找到了，找一些 SNMP 工具(如 ucd-snmp 工具集)并且尝试做下面的事情：
    (a) 使用类似

```
snmpwalk nodename public system
```

    的操作获取整个 system 组。也尝试在上面的操作中用 1 来代替 system。
    (b) 使用多个 SNMP GET-NEXT 操作(如使用 snmpgetnext 或等价的操作)手工遍历 system 组，每次取出一条记录。

35. 使用前一道习题中的 SNMP 设备和工具获取 tcp 组(组编号是 6)或某个其他组。然后执行某种操作使一些组的计数器有所改变，再重新获取这些组以显示其变化。尝试用这样一种方法做这件事，确保是你的行动引起记录的改变。

36. SNMP 提供的什么信息对计划发起第 5 章习题 17 的 IP 欺骗攻击的人可能是有用的?还有什么其他 SNMP 信息被认为是敏感的。

37. 如何设计只使用 HTTP 302 重定向或只使用 DNS 的 CDN 重定向机制?这种方法有什么限制?将两

种机制结合起来可行吗？

38. 如果一个基于DNS的重定向机制想根据当前负载信息选择一台合适的服务器，那么它会遇到什么问题？

39. 想象一种情况，有多个CDN希望互相成为对等方（类似于自治系统在IP层成为对等方），以便将内容传送给大量的终端用户。例如，CDN A可能代表一个内容提供商集合来提供内容，而CDN B可能代表另一个内容提供商集合，A和B都有一个物理覆盖区使得它们能够将内容传送给不同的终端用户集合。概述CDN A和B如何结合使用DNS重定向和HTTP 302重定向将来自CDN A的内容提供商的内容发送给CDN B的终端用户（或反过来）。

40. 想象一个CDN被配置成缓存分层结构，终端用户从边界缓存中访问内容，边界缓存则在缓存失效时从父缓存获取内容，依次向上直到根缓存，根缓存最终从原始服务器获取内容。什么指标会引起如下配置决策：

(a) 向一个特定的缓存加入更大的存储量。

(b) 向缓存分层结构加入另一层。

41. 多播覆盖有效地将流式内容从单个源推送到多个目的地，在中间节点不缓存流内容。CDN有效地从一个缓存分层结构中拉取内容（包括视频），并在中间节点缓存内容。举例说明如何将二者看作彼此的对偶。解释为什么可以认为CDN与异步多播等价。（提示：考虑TiVo。）

42. 考虑下面的简化BitTorrent环境。在回答本问题期间，有一个有2^n个对等方的群，并且没有对等方加入或离开该群。对等方需要1个时间单元来上传或下载一个片，并且在一个时间内只能做一件事。开始时，有一个对等方有完整的文件，而其他对等方什么也没有。

(a) 如果该群的目标文件只由1个片组成，那么所有对等方得到该文件至少需要多长时间？忽略除上传/下载以外的其他时间。

(b) 设x是你对上一个问题的答案。如果群的目标文件由两个片组成，那么所有对等方在不到$2x$个时间单元内得到该文件是可能的吗？为什么？

习题选答

第1章

4. 当最后一个数据比特到达目的地时，我们认为传输完成。

(a) 1.5MB=12582 912 比特，两个初始 RTT(160ms)+12 582 912/10 000 000bps(传输)+RTT/2(传播)≈1.458s。

(b) 请求的分组数=1.5MB/1KB=1 536。对于上面的值，我们加上 1 535 个 RTT 的时间（分组 1 到达和分组 1 536 到达之间的 RTT 个数），总共是 1.458+122.8=124.258s。

(c) 1 536 个分组除以 20 是 76.8，这将占用 76.5 个 RTT（第 1 组到达用半个 RTT，加上第 1 组和第 77 组之间的 76 个 RTT），再加上初始的两个 RTT，一共是 6.28s。

(d) 我们在握手后立即发送一个分组，握手后的一个 RTT 后我们发了两个分组，n 个 RTT 之后，我们已经发了 $1+2+4+\cdots+2^n=2^{n+1}-1$ 个分组。在 $n=10$ 时，我们已经发了所有 1 536 个分组，最后一组在 0.5 个 RTT 后到达。总的时间是 2+10.5RTT，即 1s。

6. 传播延迟是 50×10^3m/（2×10^8m/s）= 250μs，800 比特/250μs = 3.2 Mbps。对于 512 字节的分组，这个值提高为 16.4Mbps。

14. (a) 链路上传播延迟是（55×10^3）/（3×10^8）=184s。因此 RTT 是 368s。

(b) 链路的延迟带宽积是 $184\times128\times10^3=2.81$ MB。

(c) 拍摄完一张图片后必须马上在链路上发送，并且在任务控制中心能解释它之前要传播完。5MB 数据的传输延迟是 41 943 040 比特/128×10^3=328s，因此所需的总时间是传输延迟+传播延迟=328+184=512s。

17. (a) 对每一条链路，要花 1Gbps/5kb=5μs 时间在链路上发送分组，然后最后 1 比特穿过链路还需要 10μs。因此对只有一个交换机的 LAN，交换机仅在收到整个分组后才开始转发，总的传送延迟是两个传输延迟+两个传播延迟=30μs。

(b) 对于 3 台交换机及 4 条链路，总延迟是 4 个传输延迟+4 个传播延迟 = 60μs。

(c) 对于“直通”转发，交换机在转发之前仅需要解码前 128 位数据，这需要 128ns，这个延迟替换了前一答案中的传输延迟，所以总延迟为：1 个传输延迟+3 个直通解码延迟+4 个传播延迟=45.384μs。

27. (a) 1 920×1 080×24×30=1 492 992 000≈1.5Gbps。

(b) 8×8 000=64kbps。

(c) 260×50=13kbps

(d) 24×88 200=216 800≈2.1Mbps。

第2章

3. 给定比特序列的 4B/5B 编码如下。

11011 11100 10110 11011 10111 11100 11100 11101

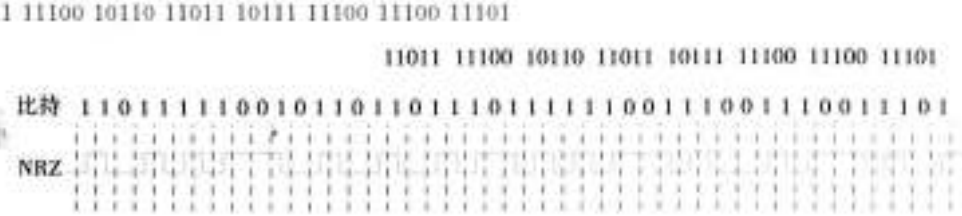


图 1

7. 用标记"∧"填充的0被移去的位置，当检测到连续7个1时表示出现一个错误（err），在该比特序列的末端检测到帧结束（eof）。

$$01101011111\wedge10100111111\underline{1}_{err}\ 0110\ \underline{01111110}_{eof}$$

19. (a) 将消息1011 0010 0100 1011附加8个0，然后用1 0000 0111（$x^8+x^2+x^1+1$）去除。余数是1001 0011。我们将原始消息附加上这个余数一起发送，结果为：

1011 0010 0100 0011 1001 0011

(b) 将上述消息的第一位反转得到0011 0010 0100 1011 1001 0011，用1 0000 0111（$x^8+x^2+x^1+1$）去除，得到余数1011 0110。

25. 链路的单程时延是100ms，带宽×往返延迟大约是125分组/s×0.2s或者25个分组。SWS应该这么大。

(a) 如果RWS=1，必须的序号空间是26，因此需要5比特。

(b) 如果RWS=SWS，序号空间必须覆盖SWS的两倍，即达到50，因此需要6比特。

32. 图2给出第一个例子的时间线，第二个例子减少了大约1个RTT的交易时间。

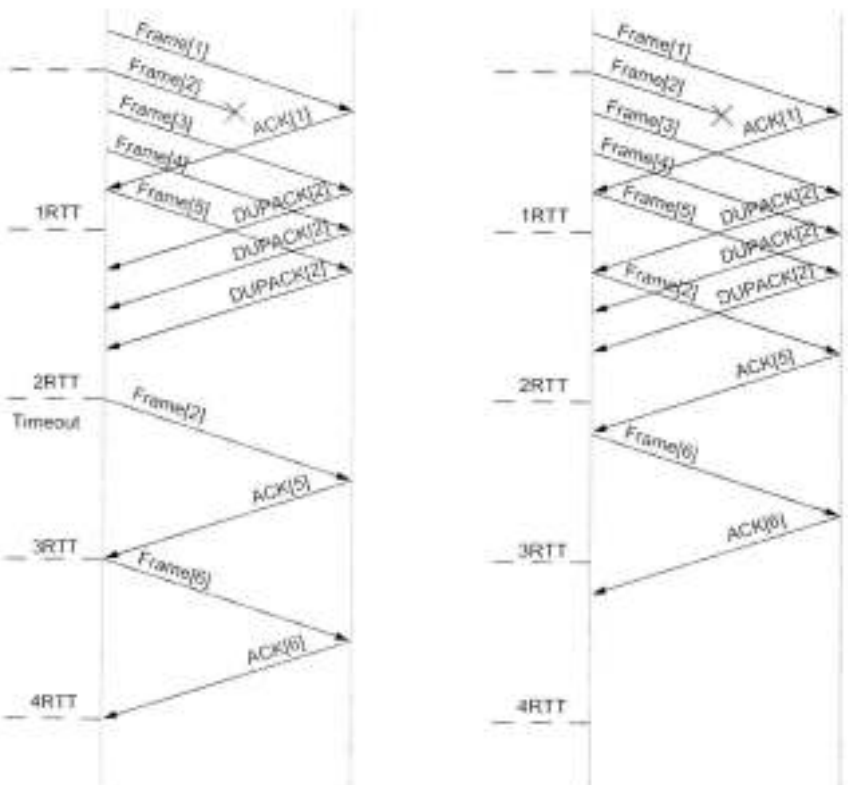


图 2

第3章

2. 表1是累积的，VCI表的每一个部分既包含当前记录也包含以前的所有记录。注意，在（d）部分我们假设交换机4端口0的VCI0不可重用（它在（a）部分中用于连接H）。这与通常情况下的双向VCI更加一致。

表 1

习题	交换机	输入		输出	
		端口	VCI	端口	VCI
(a)	1	0	0	1	0
	2	3	0	1	0
	4	3	0	0	0
(b)	2	0	0	1	1
	3	3	0	0	0
	4	3	1	1	0
(c)	1	1	1	2	0
	2	1	2	3	1
	4	2	0	3	2
(d)	1	1	2	3	0
	2	1	3	3	2
	4	0	1	3	3
(e)	2	0	1	2	0
	3	2	0	0	1
(f)	2	1	4	0	2
	3	0	2	1	0
	4	0	2	3	4

14. 下面给出了 LAN 和它们的指派网桥之间的映射关系：

B1 dead

B2 A，B，D

B3 E，F，G，H

B4 I

B5 idle

B6 J

B7 C

16. 所有的网桥能看到从 D 到 C 的分组。只有 B3、B2 和 B4 看到从 C 到 D 的分组。只有 B1、B2、B3 看到从 A 到 C 的分组。

B1 A-接口：A　　B2-接口：D（不是 C）

B2 B1-接口：A　　B3-接口：C　　B4-接口：D

B3 C-接口：C　　B2-接口：A，D

B4 D-接口：D　　B2-接口：C（不是 A）

27. 由于 I/O 总线速度低于内存带宽，所以它是瓶颈。因为每个分组穿过 I/O 总线两次，所以能提供的有效带宽是 1 000/2Mbps。因此接口数目是 (500/100)－5。

37. 根据定义，路径 MTU 为 576 字节，最大 IP 载荷是 576－20＝556 字节。我们需要传输的 IP 载荷为 1 024＋20－1044 字节，可将它分为两个分片，第一个分片的大小是 552 字节（因为分片应该是 8 字节的整数倍，因此不能是 556。）第二个分片大小为 1 044－552－492 字节。如果用路径 MTU，一共需要两个分片。上一题中我们需要三个分片。

47. (a) 见表 2。

表 2

存储在节点中的信息	至到达节点的距离					
	A	B	C	D	E	F
A	0	2	∞	5	∞	∞
B	2	0	2	∞	1	∞
C	∞	2	0	2	∞	3
D	5	∞	2	0	∞	∞
E	∞	1	∞	∞	0	3
F	∞	∞	3	∞	3	0

(b) 见表 3。

表 3

存储在节点中的信息	至到达节点的距离					
	A	B	C	D	E	F
A	0	2	4	5	3	∞
B	2	0	2	4	1	4
C	4	2	0	2	3	3
D	5	4	2	0	∞	5
E	3	1	3	∞	0	3
F	∞	4	3	5	3	0

(c) 见表 4。

表 4

存储在节点中的信息	至到达节点的距离					
	A	B	C	D	E	F
A	0	2	4	5	3	6
B	2	0	2	4	1	4
C	4	2	0	2	3	3
D	5	4	2	0	5	5
E	3	1	3	5	0	3
F	6	4	3	5	3	0

53. 图 3 是一个网络拓扑的示例：

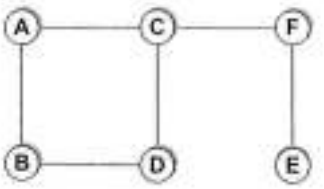


图 3

56. 应用每一个子网掩码，如果相应的子网号与对应列匹配，那么在下一跳中使用该表项。

(a) 应用子网掩码 255.255.254.0，我们得到 128.96.170.0，用接口 0 作为下一跳。

(b) 应用子网掩码 255.255.254.0，我们得到 128.96.166.0（下一跳是 R2）。应用子网掩码 255.255.252.0，我们得到 128.96.164.0（下一跳是 R3）。但是 255.255.254.0 是一个更长的前缀，所以使用 R2 作为下一跳。

(c) 没有子网号匹配，使用默认路由器 R4。

(d) 应用子网掩码 255.255.254.0，我们得到 128.96.168.0，用接口 1 作为下一跳。

(e) 应用子网掩码 255.255.252.0，我们得到 128.96.164.0，用 R3 作为下一跳。

63. 见表 5。

表 5

步骤	已证实的	试探性的
1	(A, 0, −)	
2	(A, 0, −)	(B, 1, B) (D, 5, D)
3	(A, 0, −) (B, 1, B)	(D, 4, B) (C, 7, B)
4	(A, 0, −) (B, 1, B) (D, 4, B)	(C, 5, B) (E, 7, B)
5	(A, 0, −) (B, 1, B) (D, 4, B) (C, 5, B)	(E, 6, B)
6	(A, 0, −) (B, 1, B) (D, 4, B) (C, 5, B) (E, 6, B)	

73. (a) F (b) B (c) E (d) A (e) D (f) C

第 4 章

15. 图 4 描述了源 D 和 E 的多播树。

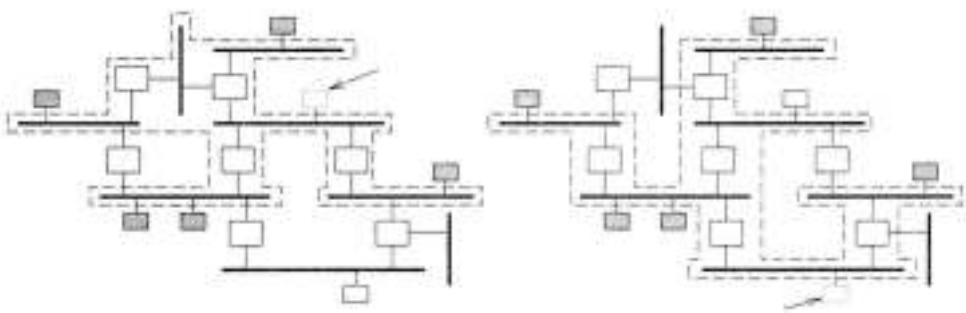


图 4

第 5 章

10. 通知窗口应该足够大以保证管道满载，延迟（RTT）×带宽是 140ms×1Gbps=10Mb=17.5MB，对于 AdvertisedWindow 字段，需要 25 比特（2^{25} = 33 554 432）。序号空间在最大的段生存期内必须不回绕。在 60s 内，能发送 7.5GB，33 比特允许使用一个 8.6GB 的序号空间，并且在 60s 内不回绕。

13. (a) 2^{32}B / (5GB) = 859ms。

(b) 在 859ms 中 1 000 次确答就是每 859μs 一次，意味着在 3.7Ms 或大约 43 天出现回绕。

27. 初始 Deviation=50，反复 20 次 TimeOut 才会低于 300.0。见表 6。

表 6

反复次数	SampleRTT	EstRTT	Dev	diff	TimeOut
0	200.0	90.0	50.0	—	—
1	200.0	103.7	57.5	110.0	333.7
2	200.0	115.7	62.3	96.3	364.9
3	200.0	126.2	65.0	84.3	386.2
4	200.0	135.4	66.1	73.8	399.8
5	200.0	143.4	66.0	64.6	407.4
6	200.0	150.4	64.9	56.6	410.0
7	200.0	156.6	63.0	49.6	408.6
8	200.0	162.0	60.6	43.4	404.4

（续）

反复次数	SampleRTT	EstRTT	Dev	diff	TimeOut
9	200.0	166.7	57.8	38.0	397.9
10	200.0	170.8	54.8	33.3	390.0
11	200.0	174.4	51.6	29.2	380.8
12	200.0	177.6	48.4	25.6	371.2
13	200.0	180.4	45.2	22.4	361.2
14	200.0	182.8	42.0	19.6	350.8
15	200.0	184.9	38.9	17.2	340.5
16	200.0	186.7	36.0	15.1	330.7
17	200.0	188.3	33.2	13.3	321.1
18	200.0	189.7	30.6	11.7	312.1
19	200.0	190.9	28.1	10.3	303.3
20	200.0	192.0	25.8	9.1	295.2

第 6 章

11. (a) 首先我们计算完成时间 F_i。由于对所有分组我们可以令 $A_i=0$，所以在此不必担心时钟速度，F_i 就变成每个流的累积大小，即 $F_i=F_{i-1}+P_i$。如表 7 所示，现在我们以 F_i 的升序来发送：分组 3，分组 1，分组 6，分组 4，分组 7，分组 2，分组 5，分组 8。

(b) 给定流 1 的权值为 2，我们将每个 F_i 除以 2，即 $F_i=F_{i-1}+P_i/2$。给定流 2 的权值 4，我们对每个 F_i 除以 4，即 $F_i=F_{i-1}+P_i/4$。给定流 3 的权值 3，我们对每个 F_i 除以 3，即 $F_i=F_{i-1}+P_i/3$。再次使用没有等待时间这个事实。如表 8 所示，以加权 F_i 的升序发送分组：分组 3，分组 4，分组 6，分组 1，分组 5，分组 7，分组 8，分组 2。

表 7

分组	大小	流	F_i
1	200	1	200
2	200	1	400
3	160	2	160
4	120	2	280
5	160	2	440
6	210	3	210
7	150	3	360
8	90	3	450

表 8

分组	大小	流	加权的 F_i
1	200	1	100
2	200	1	200
3	160	2	40
4	120	2	70
5	160	2	110
6	210	3	70
7	150	3	120
8	90	3	150

15. (a) 对一个给定流上第 i 个到达的分组，通过公式 $F_i=\max\{A_i,F_{i-1}\}+1$ 计算估计的完成时间 F_i，其中用于测量到达时间 A_i 的时钟以活动队列的数目为因子缓慢增加。A_i 的时钟是全局的，按上述方法计算的 F_i 序列对每个流来说是局部的。

表 9 列按时钟顺序列出所有事件，我们用分组所属的流和到达时间标识分组，因此分组 A4 是在时钟为 4 时列出到达的流 A 的分组，即第三个分组。最后 3 列是后续时间间隔中每个流的队列，包括当前正在发送的分组。活动队列的数目决定了 A_i 在下一行上的增加量。如果多个分组的 F_i 值相同，那么它们出现在同一行上；当 $F_i=F_{i-1}+1$（相对于 $F_i=A_i+1$）时，F_i 值用斜体表示。

表 9

时钟	A_i	到达	F_i	发送	A 队列	B 队列	C 队列
1	1.0	A1, B1, C1	2.0	A1	A1	B1	C1
2	1.333	C2	3.0	B1		B1	C1, C2

（续）

时钟	A_i	到达	F_i	发送	A队列	B队列	C队列
3	1.833	A3	3.0	C1	A3		C1，C2
4	2.333	B4	3.333	A3	A3	B4	C2，C4
		C4	4.0				
5	2.666	A5	4.0	C2	A5	B4	C2，C4
6	3.0	A6	5.0	B4	A5，A6	B4	C4，C6
		C6	5.0				
7	3.333	B7	4.333	A5	A5，A6	B7	C4，C6，C7
		C7	6.0				
8	3.666	A8	6.0	C4	A6，A8	B7，B8	C4，C6，C7
		B8	5.333				
9	4	A9	7.0	B7	A6，A8，A9	B7，B8，B9	C6，C7
		B9	6.333				
10	4.333			A6	A6，A8，A9	B8，B9	C6，C7
11	4.666	A11	8.0	C6	A8，A9，A11	B8，B9	C7
12	5	C12	7.0	B8	A8，A9，A11	B8，B9	C7，C12
13	5.333	B13	7.333	A8	A8，A9，A11	B9，B13	C7，C12
14	5.666			C7	A9，A11	B9，B13	C7，C12
15	6.0	B15	8.333	B9	A9，A11	B9，B13，B15	C12
16	6.333			A9	A9，A11	B13，B15	C12
17	6.666			C12	A11	B13，B15	C12
18	7			B13	A11	B13，B15	
19	7.5			A11	A11	B15	
20	8			B15		B15	

(b) 对于流B的加权公平队列，

$$F_i=\max\{A_i,F_{i-1}\}+0.5$$

对流A和C，F_i同前，可得表10。

表 10

时钟	A_i	到达	F_i	发送	A队列	B队列	C队列
1	1.0	A1，C1	2.0	B1	A1	B1	C1
		B1	1.5				
2	1.333	C2	3.0	A1			C1，C2
3	1.833	A3	3.0	C1	A1		C1，C2
4	2.333	B4	2.833	B4	A3	B4	C2，C4
		C4	4.0				
5	2.666	A5	4.0	A3	A3，A5		C2，C4
6	3.166	A6	5.0	C2	A5，A6		C2，C4，C6
		C6	5.0				
7	3.666	B7	4.167	A5	A5，A6	B7	C4，C6，C7
		C7	6.0				
8	4.0	A8	6.0	C4	A6，A8	B7，B8	C6，C7
		B8	4.666				
9	4.333	A9	7.0	B7	A6，A8，A9	B7，B8，B9	C6，C7
		B9	5.166				
10	4.666			B8	A6，A8，A9	B8，B9	C6，C7
11	5.0	A11	8.0	A6	A6，A8，A9，A11	B9	C6，C7
12	5.333	C12	7.0	C6	A8，A9，A11	B9	C6，C7，C12
13	5.666	B13	6.166	B9	A8，A9，A11	B9，B13	C7，C12

（续）

时钟	A_i	到达	F_i	发送	A队列	B队列	C队列
14	6.0			A8	A9，A11	B13	C7，C12
15	6.333	B15	6.833	C7	A9，A11	B13，B15	C12
16	6.666			B13	A9，A11	B13，B15	C12
17	7.0			B15	A11	B15	C12
18	7.333			A9	A11		C12
19	7.833			C12	A11		C12
20	8.333			A11	A11		

35. (a) 我们有

$$\text{TempP}=\text{MaxP}\times(\text{AvgLen}-\text{MinThreshold})/(\text{MaxThreshold}-\text{MinThreshold})$$

AvgLen 是 MinThreshold 和 MaxThreshold 的平均值，这意味着比例为 1/2，所以 TempP＝MaxP/2＝p/2。现在我们有

$$P_{count}=\text{TempP}/(1-\text{count}\times\text{TempP})=1/(x-\text{count})$$

此处，$x=2/p$，因此

$$1-P_{count}=(x-(\text{count}+1))/(x-\text{count})$$

求乘积

$$(1-P_1)\times\cdots\times(1-P_n)$$

得出

$$\frac{x-2}{x-1}\cdot\frac{x-3}{x-2}\cdot\cdots\cdot\frac{x-(n+1)}{x-n}=\frac{x-(n+1)}{x-1}$$

其中，$x=2/p$。

(b) 从前一问题的结果可得

$$a=\frac{x-(n+1)}{x-1}$$

因此

$$x=\frac{(n+1)-a}{1-a}=2/p$$

相应地有

$$p=\frac{2(1-a)}{(n+1)-a}$$

48. 在每时每刻，桶的容量必须非负，对给定的桶的深度 D 和令牌速率 r，我们能计算 t 秒时的桶容量 $v(t)$，并且强制 $v(t)$ 非负。

$$v(0)=D-5+r=D-(5-r)\geqslant 0$$
$$v(1)=D-5-5+2r=D-2(5-r)\geqslant 0$$
$$v(2)=D-5-5-1+3r=D-(11-3r)\geqslant 0$$
$$v(3)=D-5-5-1+4r=D-(11-4r)\geqslant 0$$
$$v(4)=D-5-5-1-6+5r=D-(17-5r)\geqslant 0$$
$$v(5)=D-5-5-1-6-1+6r=D-6(3-r)\geqslant 0$$

我们定义函数 $f_1(r)$，$f_2(r)$，…，$f_6(r)$ 如下：

$$f_1(r)=5-r$$
$$f_2(r)=2(5-r)-2f_1(r)\geqslant f_1(r)\ (1\leqslant r\leqslant 5)$$
$$f_3(r)=11-3r\leqslant f_2(r)\quad (r\geqslant 1)$$
$$f_4(r)=11-4r<f_3(r)\quad (r\geqslant 1)$$
$$f_5(r)=17-5r$$
$$f_6(r)=6(3-r)\leqslant f_5(r)\quad (r\geqslant 1)$$

首先对 $r\geqslant 5$，$f_i(r)\leqslant 0$ 的所有 i，这意味着令牌速率比每秒 5 分组要快，任何正的桶深度都满足（即 $D\geqslant 0$）。对 $1\leqslant r\leqslant 5$，由于其他的函数小于 $f_2(r)$ 和 $f_1(r)$，我们仅需要考虑它们就可以了，你很容易发现 $f_2(r)-f_1(r)=3r-7$。因此桶的深度 D 必须满足下列公式

$$D\geqslant\begin{cases}f_1(r)=17-5r & r=1,2\\ f_2(r)=2(5-r) & r=3,4,5\\ 0 & r\geqslant 5\end{cases}$$

第 7 章

2. 每个串的前面是它的长度计数，工资数组的前面是元素个数计数。最终被发送的整数和 ASCII 字符序列是：

4 M A R Y 4377 7 J A N U A R Y 7 2002 2 90000 150000 1

8.

表 11

INT	4	15
INT	4	29496729
INT	4	58993458

10. 15 be 00000000 00000000 00000000 00001111
15 le 00001111 00000000 00000000 00000000

29496729 be 00000001 11000010 00010101 10011001
29496729 le 10011001 00010101 11000010 00000001

58993458 be 00000011 10000100 00101011 00110010
58993458 le 00110010 00101011 10000100 00000011

术　语

3DES：三重 DES。使用了三个密钥的 DES 版本，有效增加了密钥大小和加密的健壮性。

3G：第三代移动无线网，是一种基于码分多路复用（CDMA）的蜂窝无线技术。

4B/5B：在光纤分布数据接口（FDDI）中使用的一种比特编码方案。其中每 4 比特数据作为一个 5 比特序列传输。

4G：第四代无线技术，正在逐步普及的一套用于支持比 3G 更高的数据传输率的标准。

802.3：IEEE 以太网标准。

802.5：IEEE 令牌环网标准。

802.11：IEEE 无线网络标准。

802.17：IEEE 弹性分组环标准。

822：RFC 822，定义因特网电子邮件格式。见 SMTP。

AAL（ATM Adaptation Layer，ATM 适配层）：是一个配置在 ATM 之上的协议层。为数据通信定义了两个 AAL：AAL3/4 和 AAL5。每个协议层提供一种机制，在发送方将大分组分段为信元，而在接收方将信元重组为大分组。

ABR：①可用的比特率（Available Bit Rate），它是一种专门为 ATM 网络开发的基于速率的拥塞控制方案。ABR 可以根据来自网络中的交换机的反馈，允许数据源增加或减少所分配到的速率。对比 CBR、UBR 和 VBR。②区域边界路由器（Area Border Router），是处在一个链路状态协议中区域边界上的路由器。

ACK：acknowledgment（确认）的缩写。由接收数据的接收方发送一个确认，向发送方表明数据传输成功。

additive increase/multiplicative decrease（加性增/乘性减）：TCP 使用的拥塞窗口策略。TCP 以线性速率打开拥塞窗口，但是当由于拥塞导致数据丢失时就将窗口减小一半。事实表明加性增/乘性减是保持拥塞控制机制稳定的必要条件。

AES（Advanced Encryption Standard，高级加密标准）：用来取代 DES 的密码。

AF（Assured Forwarding，确保转发）：它是为区分服务提出的每跳行为之一。

ALF（Application Level Framing，应用层框架）：它是一种协议设计原则，表明与通用的传输协议相比，应用程序能更好地理解它们的通信需求。

AMPS（Advanced Mobile Phone System，先进的移动电话系统）：它是一种模拟蜂窝电话系统，现在正被数字系统（称为个人通信服务，PCS）所取代。

ANSI（American National Standards Institute，美国国家标准协会）：它是非官方的标准化组织，参与 ISO 标准化过程，负责 SONET。

API（Application Programming Interface，应用程序编程接口）：它是应用程序用来访问网络子系统（通常是传输协议）的接口，通常特定于操作系统。Berkeley Unix 的套接字 API 是一个广泛使用的例子。

area（区域）：在链路状态路由选择的上下文中，它是彼此共享全部路由信息的邻接路由器集合。为提高可扩展性，一个路由选择域被划分成区域。

ARP（Address Resolution Protocol，地址解析协议）：因特网体系结构的协议，用于将高层协议地址翻译成物理硬件地址，一般用于在因特网上将 IP 地址映射为以太网地址。

ARPA（Advanced Research Projects Agency，高级研究计划署）：美国国防部的研究与开发组织之一，负责资助 ARPANET 和可促使 TCP/IP 因特网发展的研究项目。也称为 DARPA，D 代表国防部（Defense）。

ARPANET：一个由 ARPA 资助的试验性广域分组交换网络，开始于 20 世纪 60 年代末，成为发展中的因特网的主干网。

ARQ（Automatic Repeat Request，自动请求重发）：它是在不可靠链路上可靠地发送分组的通用策略。如果发送方在一段特定的时间后没有收到分组的 ACK，它就假定此分组没有到达目的地（或者到达了但是有比特错误），并重发该分组。

停止-等待与滑动窗口是 ARQ 协议的两个例子。对比 FEC。

ASN.1（Abstract Syntax Notation One，抽象语法标记 1）：它与基本编码规则（BER）一起作为 OSI 体系结构的一部分，是由 ISO 制定的格式化表示的标准。

ATM（Asynchronous Transfer Mode，异步传输模式）：它是一种面向连接的网络技术，使用小的定长分组（称为信元）传输数据。

ATM Forum（ATM 论坛）：一个建立 ATM 标准的重要组织。

authentication（认证）：一种安全协议，存疑的双方通过它相互证实它们符合其所声称的身份。

autonomous system（AS，自治系统）：一组网络和路由器，它们属于同一机构，使用相同的域内路由协议。

bandwidth（带宽）：一种对链路或连接的容量的度量，通常以比特/秒为单位。

Bellman-Ford：一种距离向量路由选择算法的名称，以两位发明者的名字命名。

BER（Basic Encoding Rule，基本编码规则）：对 ASN.1 中定义的数据类型进行编码的规则。

best-effort delivery（尽力而为的传送）：当前因特网体系结构中使用的服务模型。消息传输是尽力而为的，不提供保证。

BGP（Border Gateway Protocol，边界网关协议）：一种域间路由选择协议，自治系统可通过它交换可达性信息。最新版本是 BGP-4。

BISYNC（Binary Synchronous Communication，二进制同步通信）：由 IBM 在 20 世纪 60 年代末开发的面向字节的链路层协议。

bit stuffing（比特填充）：一种在比特级区分控制序列和数据的技术，在高级数据链路控制（HDLC）协议中使用。

block（阻塞）：操作系统术语，用来描述一个进程因等待某一事件而暂停执行的状态，如等待信号量（semaphore）状态的改变。

bluetooth（蓝牙）：用于短距离连接计算机、移动电话、外围设备的无线标准。

bridge（网桥）：一种将链路层的帧从一个物理网络转发到另一个网络的设备，有时也称局域网交换机。对比中继器（repeater）和路由器（router）。

broadcast（广播）：一种将一个分组传到一个特定网络或互联网上的每一个主机的方法，可以用硬件实现（如以太网），也可以用软件实现（如 IP 广播）。

CA（Certification Authority，认证机构，也称为证书管理机构（Certification Authority））：一个负责签署安全证书的实体，保证包含在证书中的公钥属于证书中命名的实体。

CBC（Cipher Block Chaining，密码分组链）：一种加密模式，在加密前，每个明文分组与前一个密文分组进行异或。

CBR（Constant Bit Rate，恒定比特率）：是 ATM 中的一类服务，它们保证以一种恒定的比特率传输数据，从而模拟一条专用的传输链路。对比 ABR、UBR 和 VBR。

CCITT：现在已不复存在的国际电报电话咨询委员会（Consultative Committee of International Telegraph and Telephone），它是联合国的国际电信联盟（ITU）的分会。该组织已经被 ITU-T 取代。

CDMA（码分多址）：用于无线网络的多路复用方法。

CDN（内容分发网络）：分布在因特网上的 Web 服务器代理的集合，它们代替服务器响应 Web HTTP 请求。广泛分布代理服务器的目的是一旦有某个代理靠近客户时，使它能更快地响应请求。

cell（信元）：一个 53 字节的 ATM 分组，能够传输最多 48 字节的数据。

certificate（证书）：由一个实体数字签名后的文档，其中包含另一个实体的名称和公钥。用于发布公钥。参见 CA。

channel（信道）：本书使用的一个通用的通信术语，用来描述一个逻辑的进程到进程的连接。

checksum（校验和）：通常指对一个分组的所有或部分字节的反码求和，由发送方计算并附加在分组的后面。接收方重新计算校验和，并将它与消息中携带的校验和比较。校验和用于检查分组中的差错，也用于验证分组被传送到了正确的主机。术语校验和有时（不严密地）用于通指检错码。

chipping code（片码）：与数据流进行异或（XOR）操作，用于实现扩频直接序列技术的随机比特序列。

CIDR（Classless Interdomain Routing，无类域间路由）：一种聚合路由的方法，将一块连续的 C 类

IP 地址看作一个网络。

circuit switching（**电路交换**）：一种通过网络交换数据的通用策略。它需要在数据源和目的地之间建立一个专用路径（电路）。对比分组交换（packet switching）。

client（**客户**）：分布式系统中的服务请求者。

cLNP（Connectionless Network Protocol，**无连接网络协议**）：ISO 中对应于因特网 IP 协议的协议。

clock recovery（**时钟恢复**）：在串行传输的数字信号中导出有效时钟的过程。

concurrent logical channels（**并发逻辑信道**）：在单条点到点链路上多路复用多个停止-等待逻辑信道。没有强制的传递顺序。这种机制用在 ARPANET 的 IMP-IMP 协议中。

congestion（**拥塞**）：一种由于过多分组竞争有限的资源（如路由器或交换机上的链路带宽和缓冲区空间）而导致的状态。这种状态可能会使路由器（交换机）被迫丢弃分组。

congestion control（**拥塞控制**）：它是能够缓和或避免拥塞出现的任何网络资源管理策略。拥塞控制机制可以在网络中的路由器（交换机）上实现，也可以在网络边缘的主机上实现，或者是两者结合来实现。

connection（**连接**）：一般地，在使用一个信道前要先建立它（如通过传输一些设置信息）。例如，TCP 提供了一种连接抽象，支持可靠有序的字节流传递。面向连接的网络（如 ATM）常常被称作提供了一个虚电路（virtual circuit）抽象。

connectionless protocol（**无连接协议**）：一种不需要预先建立连接就能发送数据的协议。IP 就是这种协议的一个例子。

context switch（**上下文切换**）：一种操作系统挂起一个进程的执行并开始执行另一个进程的操作。上下文切换包括保存前一个进程的状态（如所有寄存器的内容）和装入后一个进程的状态。

controlled load（**受控负载**）：因特网综合服务体系结构中的一种服务类型。

CRC（Cyclic Redundancy Check，**循环冗余校验**）：一种检错码，由网络硬件（如以太网适配器）对组成分组的字节计算得到，并将它附加到分组上。循环冗余校验提供比简单校验和更强大的差错检测功能。

crossbar switch（**纵横式交换机**）：一种简单的交换机，其中每个输入都与每个输出直接相连，而输出端口负责解决争用。

CSMA/CD（Carrier Sense Multiple Access with Collision Detect，**带冲突检测的载波侦听多路访问**）：CSMA/CD 是网络硬件的一种功能。“载波侦听多路访问”的意思是多个站可以侦听链路并检测链路何时处于使用或空闲状态。“冲突检测”是指如果两个或多个站同时在链路上传送数据，它们将会检测到信号间的冲突。以太网是使用 CSMA/CD 技术的最广为人知的例子。

cut-through（**直通快速转发**）：交换或转发的一种方式，在一个分组完全被交换节点接收之前，它就被传送到输出接口。这样能减少分组通过节点的时延。

datagram（**数据报**）：因特网体系结构中的基本传输单元。数据报中含有将它传递给目的地所需的所有信息，与美国邮政系统的信件类似。数据报网络是无连接的。

DCE（Distributed Computing Environment，**分布式计算环境**）：支持分布式计算的基于 RPC 的协议和标准的集合，由 OSF 定义。

DDCMP（Digital Data Communication Message Protocol，**数字数据通信报文协议**）：数字设备公司的 DECnet 中采用的一种面向字节的链路层协议。

DDoS（Distributed Denial of Service，**分布式拒绝服务**）：一种拒绝服务攻击，由一组节点发起攻击。每个进行攻击的节点只对目标机投放少量负载，但来自所有攻击节点的总负载量会使被攻击的目标机不堪重负。

DECbit：一种拥塞控制方案，其中路由器通过将被转发的分组首部中的一位置为 1，通知端节点即将到来的拥塞。当端节点收到的该位置为 1 的分组达到一定百分比时，端节点将降低它们的发送速率。

decryption（**解密**）：加密过程的逆过程，从被加密的报文中恢复数据。

delay bandwidth product（**延迟带宽积**）：网络的 RTT 与带宽的乘积，给出网络能传送多少数据的度量。

demultiplexing（**多路分解**）：利用分组首部包含的信息，将其导向一个协议栈。例如，IP 用其首部的 ProtNum 字段决定一个分组属于哪个高层协议（即 TCP 或 UDP）。而 TCP 使用端口号将 TCP 分组分解到正确的应用进程。对比多路复

用（multiplexing）。

demultiplexing key（多路分解键）：分组首部中能使多路分解得以进行的字段（如 IP 的 ProtNum 字段）。

dense mode multicast（密集模式多播）：大多数路由器或主机需要接收多播分组时使用的 PIM 模式。

DES（Data Encryption Standard，数据加密标准）：一种基于 64 比特秘密密钥的数据加密算法。

DHCP（Dynamic Host Configuration Protocol，动态主机配置协议）：一种当主机自举时使用的协议，用来获得各种网络信息，如 IP 地址。

DHT（Distributed Hash Table，分布式散列表）：一种根据对象的名字把消息路由到一个支持特定对象的计算机的方法。该对象被散列到唯一的标识符，每一中间节点沿路由转发消息到一个与该 ID 共享较大前缀的节点。DHT 通常用在对等网络中。

Differentiated Services（区分服务）：它在因特网上提供一种比尽力服务更好的新式服务体系结构，已经被提议用作综合服务的替代选择。

direct sequence（直接序列）：将数据流与称为码片的随机比特序列进行异或操作的一种扩频技术。

distance vector：（距离向量）：用于路由选择的最低开销路径算法。每个节点将其可达性信息及相关开销通告给直接相邻的节点，并用收到的更新信息构造转发表。路由选择协议 RIP 使用距离向量算法。对比链路状态（link state）。

DMA（Direct Memory Access）：直接存储器存取。一种将主机与 I/O 设备相连的方法，设备可以直接从主机的内存读取数据或直接将数据写入主机内存。参见 PIO。

DNA/DECNET（Digital Network Architecture，数字网络体系结构）：一种基于 OSI 的体系结构，支持无连接的网络模型和面向连接的传输协议。

DNS（Domain Name System，域名系统）：因特网的分布式命名系统，用来将主机名（如 cicada. cs. princeton. edu）解析为 IP 地址（如 192. 12. 69. 35）。DNS 是通过名字服务器的分层结构实现的。

domain（域）：既可以指在分层 DNS 名字空间中的环境（如 "edu" 域），也可以指为了分层路由选择而被看作单一实体的一个因特网区域。后者相当于自治系统（autonomous system）。

DoS（拒绝服务）：在这一场景中，一个攻击节点向目标节点注入大量负载（大量分组），它有效地使合法用户不能访问该节点。因此它们被拒绝服务。

DS3：由电话公司提供的一种 44. 7Mbps 的传输链路服务，也称为 T3。

DSL（Digital Subscriber Line，数字用户线）：在双绞电话线上以每秒几兆比特的速度传输数据的一系列标准。

duplicate ACK（重复确认）：TCP 确认的重传。重复的 ACK 不确认任何新数据。多次收到重复的 ACK 会触发 TCP 的快速重传（fast retransmit）机制。

DVMRP（Distance Vector Multicast Routing Protocol，距离向量多播路由选择协议）：Mbone 中的大多数路由器所使用的多播路由协议。

DWDM（Dense Wavelength Division Multiplexing，密集波分复用）：在一条单一的物理光纤上多路复用多个光波（多个颜色）。就可以支持大量光波长的意义来说，这个技术是 "密集的"。

ECN（Explict Congestion Notification，显式拥塞通知）：路由器通过在转发的分组中设置一个标志来通知端主机有关拥塞情况的一种技术。与 RED 这样的活动队列管理算法结合使用。

EF（Expedited Forwarding，加速转发）：为区分服务提出的每跳行为之一。

EGP（Exterior Gateway Protocol，外部网关协议）：因特网早期的域间路由选择协议。自治系统的外部网关（路由器）用它与其他自治系统交换路由选择信息。已被 BGP 取代。

encapsulation（封装）：一种由低层协议完成的操作，将高层协议传下来的消息加上特定协议的首部或尾部。当一个消息下传到协议栈时，它收集一连串首部，最外层的首部对应栈底的协议。

encryption（加密）：对数据应用转换功能的动作，其目的是希望只有数据的接收方（在应用相反的解密（decryption）功能以后）才能读懂它。加密通常依赖于一个由收发双方共享的密钥或依赖于一对公开/私有密钥。

Ethernet（以太网）：一种使用 CSMA/CD 的流行的局域网技术，它具有 10Mbps 的带宽。以太网本身只是一条无源线路，所有与以太网传输有

关的功能完全由主机的适配器实现。

exponential backoff（指数后退）：一种重传策略，某个分组每次重传会使超时值加倍。

exposed node problem（暴露节点问题）：在无线网络中发生的一种情况，其中两个节点从同一个源接收信号，但它们都能到达不接收该信号的其他节点。

extended LAN（扩展局域网）：由网桥连接的局域网的集合。

fabric（网状结构）：交换机中真正完成交换的部分，即将分组从输入移到输出。对比端口（port）。

fair queuing（FQ，公平排队法）：一种基于循环的排队算法，能防止行为不良的进程任意占用很多网络容量。

fast retransmit：快速重传。一种由 TCP 使用的策略，旨在避免在发生分组丢失时出现超时现象。当 TCP 收到 3 个连续的重复 ACK，并确认（不包括）那个分段之前的所有数据后，就重传该分段。

FDDI（Fiber Distributed Data Interface，光纤分布式数据接口）：一种运行在光纤上的高速令牌环网络技术。

FEC：①前向纠错（Forward Error Correction），一种不用重传分组而恢复数据分组中的比特差错的通用策略。每个分组中含有冗余信息，接收方可用来确定哪些比特是不正确的。对比 ARQ。②转发等价类（Forwarding Equivalence Class），在路由器上接收相同转发待遇的分组，MPLS 标签通常与 FEC 相关联。

Fibre Channel（光纤信道）：一种常用于将计算机（通常是超级计算机）和外设相连的双向链路协议。光纤信道具有 100MBps 的带宽，最长可达 30m。

firewall（防火墙）：一个被配置成过滤（不转发）来自特定源的分组的路由器。用于实现一种安全策略。

flow control（流量控制）：数据的接收方调节发送方传输速率的一种机制，以便使数据不要到达太快以至于来不及处理。对比拥塞控制（congestion control）。

flowspec（流说明）：说明一个流所需的网络带宽和延迟以便建立预留，与 RSVP 一起使用。

forwarding（转发）：路由器对每个分组执行的一种操作，在某个输入端收到分组，决定将分组从哪个输出端发出去，并将分组发送到那个输出端。

forwarding table（转发表）：路由器中维护的一个表，能使路由器决定如何转发分组。建立转发表的过程称为路由选择，所以转发表有时也称为路由表（routing table）。在有些实现中，路由表和转发表是独立的数据结构。

fragmentation/reassembly（分片和重组）：传输消息的大小超过网络 MTU 时采用的方法。消息由发送方分成较小的片，由接收方将其重新组装。

frame（帧）：分组的另一个名称，通常用于指在一条链路上而不是在整个网络上发送的分组。关于帧的一个重要问题是接收方如何检测帧的开始和结束，即组帧问题。

Frame Relay（帧中继）：电话公司提供的一种面向连接的公用分组交换服务。

frequency hopping（跳频）：在频率的随机序列上传输数据的一种扩频技术。

FTP（File Transfer Protocol，文件传输协议）：用于在两个主机间传输文件的因特网体系结构的标准协议，建立在 TCP 之上。

GMPLS（Generalized Multi-Protocol Label Switching，通用多协议标签交换）：允许 IP 运行在光交换网络上。

gopher：一种早期的因特网信息服务。

GPRS（通用分组无线通信服务）：蜂窝无线网络提供的一种分组传输服务。

GSM（Global System for Mobile Communication，全球移动系统）：普遍使用的数字蜂窝电话系统。

H.323：通常用于因特网电话的会话控制协议。

handle（句柄）：在程序设计中用来存取一个对象的标识符或指针。

hardware address（硬件地址）：用于标识局域网中的主机适配器的链路层地址。

HDLC（High-Level Data Link Control protocol，高级数据链路控制协议）：一种链路层协议的 ISO 标准，使用比特填充来解决组帧问题。

hidden node problem（隐藏节点问题）：在无线网络中发生的一种情况，其中两个节点向同一个目标发送信息，但每个节点都不知道另一个节点的存在。

hierarchical routing（层次路由）：一种多层路由选

择方案，使用地址空间的分层结构作为转发决策的基础。例如，分组首先被路由到目标网络，然后送到网络中的特定主机。

host（主机）：连接到一个或多个网络的计算机，支持用户并运行应用程序。

HTML（Hyper Text Markup Language，超文本标记语言）：一种用于构建万维网网页的语言。

HTTP（HyperText Transport Protocol，超文本传输协议）：一种基于请求/响应模式的应用层协议，用于万维网。HTTP 使用 TCP 连接传输数据。

IAB（Internet Architecture Board，因特网体系结构工作委员会）：审查因特网体系结构协议的开发和标准化的主要机构。以前称为因特网法案。

iBGP（Interior BGP，内部边界网关协议）：用于在同一区域的路由器间交换域间路由信息的协议。

ICMP（Internet Control Message Protocol，网际控制报文协议）：该协议是 IP 的一部分。它允许路由器或目标主机与源主机通信，通常报告 IP 数据报处理过程中的错误。

IEEE（Institute for Electrical and Electronics Engineers，电气和电子工程师协会）：工程师的专业协会，也定义网络标准，包括 802 系列局域网标准。

IETF（Internet Engineering Task Force，因特网工程任务组）：负责开发因特网标准和最佳实践的主要组织。

IMAP（Internet Message Access Protocol，因特网消息访问协议）：允许用户从邮件服务器读取其邮件的一个应用层协议。

IMP-IMP：在最早的 ARPANET 中使用的面向字节的链路层协议。

Integrated Services（综合服务）：通常指能同时有效支持传统的计算机数据以及实时音频和视频的一个分组交换网，也指用来替代现有的尽力服务模型的一种因特网服务模型。

integrity（完整性）：在网络安全的环境中确保收到的消息与发送的消息相同的一种服务。

interdomain routing（域间路由）：在不同的路由选择域之间交换路由信息的一种服务。BGP 是域间路由协议的例子。

internet（互联网）：通过路由器互联起来的分组交换网的集合（可能是异构的）。也称为 internetwork。

Internet（因特网）：基于 TCP/IP 体系结构的全球互联网，连接世界各地数百万台主机。

interoperability（互操作性）：指异构的硬件以及多厂商软件通过正确地交换消息进行通信的能力。

interrupt（中断）：告诉操作系统停止当前活动并采取某种行动的（通常由硬件设备产生的）一种事件。例如，有一种中断是用来通知操作系统有一个分组已经从网络上到达。

intradomain routing（域内路由选择）：在一个域或一个自治系统内部交换路由选择信息。RIP 和 OSPF 是域内路由选择协议的实例。

IP（Internet Protocol，网际协议）：在因特网上提供一种无连接的、尽力传送数据报服务的协议。

IPng（下一代 IP，也叫作 IPv6）：IP 的新版本，提供更大更多的层次性地址空间和其他新的特性。

IPsec（IP 安全）：用于认证、保密和消息完整性的一种体系结构，是因特网的安全服务之一。

IRTF（Internet Research Task Force，因特网研究任务组）：IAB 的一个部门，负责制定因特网的研究和发展方向。

ISDN（Integrated Services Digital Network，综合业务数字网）：由电话公司提供的一种数字通信服务，由 ITU-T 标准化。ISDN 将语音连接与数字数据服务合并在单一的物理介质中。

IS-IS（Intermediate System to Intermediate System）：一种链路状态路由选择协议，类似于 OSPF。

ISO（International Standards Organization，国际标准化组织）：该组织设计了七层 OSI 体系结构以及一整套在商业上未获成功的协议。

ITU-T：国际电信联盟的附属委员会，为国际模拟和数字通信的所有领域制订技术标准的国际组织。ITU-T 制订电信标准，包括引人注目的 ATM 标准。

jitter（抖动）：指网络时延的变化。较大的抖动会对视频和音频应用的质量产生负面影响。

JPEG（Joint Photographic Experts Group，联合图像专家组）：通常是指由联合图像专家组开发的一种广泛用于压缩静止图像的算法。

Kerberos：MIT 开发的基于 TCP/IP 的认证系统，其中两台主机使用可信第三方相互认证。

key distribution（密钥分发）：用户之间通过交换数字签名的证书获得对方公钥的一种机制。

LAN（Local Area Network，局域网）：一种基于物理网络技术的网络，最多可跨越几公里（例如

以太网或 FDDI)。对比 SAN、MAN 和 WAN。

LAN switch (局域网交换机)：网桥（bridge）的另一个术语，通常用于具有多个端口的网桥。如果它支持的链路技术是以太网，也称为以太网交换机。

latency (时延)：衡量一个比特从链路或信道的一端到达另一端所需的时间。时延严格使用时间来度量。

LDAP (Lightweight Directory Access Protocol，轻量级目录访问协议)：最近流行的一种用户信息目录服务 X.500 的一个子集。

LER (Label Edge router，标签边缘路由器)：位于 MPLS 边缘的路由器，对到达的 IP 分组执行完整的 IP 查询，然后用标签作为查询结果。

link (链路)：网络中两个节点之间的物理连接。它可以是铜线或光缆，也可以是无线链路（例如人造卫星）。

link-level protocol (链路层协议)：负责在直接相连的网络（例如以太网、令牌环或点到点链路）上传送帧的协议。也称为 link-layer protocol。

link state (链路状态)：用于路由选择的最小开销路径算法。直接连接的邻居的信息和当前链路的开销被扩散给所有路由器，每台路由器用这个信息构造网络图，并以此图作为转发决策的依据。开放最短路径优先（OSPF）路由选择协议使用一种链路状态算法。对比距离向量（distance vector）。

LSR (Label-Switching Router，标签交换路由器)：运行 IP 控制协议的路由器，但使用 MPLS 标签交换转发算法。

MAC (Media Access Control，介质访问控制)：用于控制访问诸如以太网和光纤分布数据接口（FDDI）的共享介质网络的算法。

MACA (Multiple Access with Collision Avoidance，带冲突避免的多路访问)：用于调解共享介质访问的分布式算法。

MACAW (Multiple Access with Collision Avoidance for Wireless，用于无线网络的带冲突避免的多路访问)：是通用的 MACA 算法的一种改进，能更好地支持无线网络。已由 802.11 采用。

MAN (Metropolitan Area Network，城域网)：使用新网络技术的网络，这些新技术以高速运行（可达几个 Gbps）并且跨越的距离足以覆盖一个城市的区域。对比 SAN、LAN 和 WAN。

Manchester (曼彻斯特编码)：一种比特编码方案，传输的是时钟与非归零（NRZ）编码数据的异或值，用于以太网。

MBone (Multicast Backbone，多播主干网)：构建在因特网上的一个逻辑网络，其中具有增强的多播功能的路由器使用隧道技术跨越因特网转发多播数据报。

MD5 (Message Digest version 5，报文摘要版本 5)：一种有效的密码校验和算法，常用于验证消息的内容未被改动。

MIB (Management Information Base，管理信息库)：定义一个网络节点上可以读写的与网络相关的变量的集合。MIB 同简单网络管理协议（SNMP）一起使用。

MIME (Multipurpose Internet Mail Extensions，多功能因特网邮件扩展)：将二进制数据（如图像文件）转换成 ASCII 文本，然后通过电子邮件传送的规范。

Mosaic：一种以前流行的免费图形界面 WWW 浏览器，由位于伊利诺斯州大学的美国国家超级计算应用中心开发。

MP3 (MPEG Layer 3，MPEG 第三层)：与 MPEG 一起使用的音频压缩算法。

MPEG (Moving Picture Experts Group，运动图像专家组)：它通常用来指由 MPEG 开发的压缩视频流的算法。

MPLS (Multiprotocol Label Switching，多协议标签交换)：在第二层（如 ATM）交换机上有效实现 IP 路由的技术集合。

MSDP (多播源发现协议)：用于实现域间多播的协议。

MTU (Maximum Transmission Unit，最大传输单元)：在一个物理网络上可以传输的最大分组的大小。

multicast (多播)：广播的一种特殊形式，分组被传递到网络主机的一个指定子集。

multiplexing (多路复用)：将多个不同的信道合并到一个低层信道。例如，分离的 TCP 和 UDP 信道被多路复用到一条主机到主机的 IP 信道上。其逆操作是多路分解（demultiplexing），在接收主机上执行。

name resolution (名字解析)：将主机的名字（易于读取）翻译成对应的地址（机器可读）。参见 DNS。

NAT（Network Address Translation，**网络地址转换**）：一种扩展 IP 地址空间的技术，在站点或网络边界的本地唯一地址和全球理解的 IP 地址之间进行转换。

NDR（Network Data Representation，**网络数据表示**）：在分布式计算环境（DCE）中使用的数据编码标准，由开放软件基金会定义。NDR 使用接收方保证正确的策略并将体系结构标志插入到每个消息的前部。

network-level protocol（**网络层协议**）：在交换网络上运行的协议，直接在链路层之上。

NFS（Network File System，**网络文件系统**）：一种由 Sun 微系统公司开发的流行的分布式文件系统，基于 Sun 公司开发的一种 RPC 协议 SunRPC。

NIST（National Institute for Standards and Technology，**美国国家标准技术协会**）：美国官方的标准化组织。

node（**节点**）：用于表示构成网络的单个计算机的通用术语。节点包括通用计算机、交换机和路由器。

NRZ（Non-Return to Zero，**非归零编码**）：一种比特编码方案，将 1 编为高信号，0 编为低信号。

NRZI（Non-Return to Zero Inverted，**非归零反转编码**）：一种比特编码方案，在当前信号跳变时编码为 1，在当前信号不变时编码为 0。

NSF（National Science Foundation，**美国国家科学基金会**）：美国政府资助其国内科学研究的机构，包括对网络和因特网基础设施的研究。

OC（Optical Carrier，**光载波**）：它是 SONET 光传输的各种速率的前缀。例如，OC-1 指在光纤上传输速率为 51.84Mbps 的 SONET 标准。OC-n 信号与 STS-n 信号的差别，仅在于 OC-n 信号是用于对光传输进行优化的。

ONC（Open Network Computing，**开放网络计算**）：被因特网采纳为标准的 SunRPC 的一个版本。

OSF（Open Software Foundation，**开放软件基金会**）：一个计算机厂商的联合会，为分布式计算定义了多种标准，包括国家数据表示（NDR）格式。

optical switch（**光交换机**）：一种能将来自输入端口的光波不需转换成电子信号就能转发到输出端口的交换设备。

OSI（Open Systems Interconnection，**开放系统互联**）：由 ISO 开发的七层网络参考模型，指导 ISO 和 ITU-T 协议标准的设计。

OSPF（Open Shortest Path First，**开放最短路径优先**）：由 IETF 为因特网体系结构开发的路由协议。OSPF 是基于链路状态（link-state）的算法，其中每个结点构造因特网的一个拓扑图并根据该图做出转发决定。

overlay（**覆盖**）：运行在已存在物理网络上的一个虚拟（逻辑）网。覆盖节点彼此通过隧道而不是物理链路进行通信。由于覆盖不要求现有网络基础设施的协作，因此通常用来部署新的网络服务。

packet（**分组**）：在分组交换网上发送的数据单元。参见帧（frame）和分段（segment）。

packet switching（**分组交换**）：在网络上交换数据的通用策略。分组交换使用存储转发机制交换称为分组的离散数据单元，并包含统计多路复用（statistical multiplexing）。

participants（**参与者**）：用于表示相互发送消息的进程、协议或主机的通用术语。

PAWS（Protection Against Wrapped Sequence numbers，**防止序号回绕**）：设计具有足够大序号空间的传输协议，以防止在一个分组可能被长时间延迟的网络上出现序号回绕。

PDU（Protocol Date Unit，**协议数据单元**）：分组或帧的另一个名称。

peer（**对等实体**）：指一个主机上的协议模块为实现某些通信服务而与之交互的另一个主机上的对等体。

peer-to-peer networks（**对等网**）：将应用逻辑（例如文件存储）与路由选择集成的一类通用应用。有名的例子包括 Napster 和 Gnutella。研究原型通常使用分布式散列表。

PEM（Privacy Enhanced Mail，**隐私增强邮件**）：对因特网电子邮件扩展，支持保密性和完整性保护。参见 PGP。

PGP（Pretty Good Privacy，**良好隐私**）：公用域软件的一个集合，它们用 RSA 提供保密性和认证能力，并使用公钥分发信任网络进行密钥分发。

PHB（Per-Hop Behavior，**每跳行为**）：区分服务体系结构中单个路由器的行为。AF 和 EF 是两种建议的 PHB。

physical-level protocol（**物理层协议**）：OSI 协议栈的最低层。它的主要功能是将比特编码为可在

物理传输介质上传输的信号。

piconet（皮可网）：短距离（如 10m）的无线网络。可以不使用电缆而连接办公室的计算机（笔记本电脑、打印机、PDA、工作站等）。

PIM（Protocol Independent Multicast，协议无关多播）：可以建造在不同的单播路由协议上的多播路由协议。

Ping：用于测试因特网上到不同主机的 RTT 的 UNIX 工具。Ping 发送一条 ICMP ECHO_REQUEST 消息，然后远程主机发回一个 ECHO_RESPONSE 消息。

PIO（Programmed Input/Output，可编程输入/输出）：一种连接主机到 I/O 设备的方法，CPU 从 I/O 设备读数据并向 I/O 设备写数据。参见 DMA。

poison reverse（毒性逆转）：和水平分割（split horizon）结合使用，是距离向量路由选择协议中避免路由环路的一种启发式技术。

port（端口）：用于表示一个网络用户连接到网络结点的通用术语。在交换机上，端口表示接收和发送分组的输入或输出。

POTS（Plain Old Telephone Service，简易老式电话服务）：用于说明现在的电话服务与 ISDN、IP 电话（VoIP）或电话公司现在及将来提供的其他技术截然不同。

PPP（Point-to-Point Protocol，点到点协议）：描述通常通过拨号线路连接计算机的数据链路协议。

process（进程）：由操作系统提供的一种抽象，允许不同的操作并发执行。例如，每个用户应用程序通常在自己的进程内进行，而各种操作系统功能则在其他进程内进行。

promiscuous mode（混杂模式）：网络适配器的一种操作模式，适配器接收网络上传送的所有帧，而并不仅仅是发送给它的那些帧。

protocol（协议）：运行在不同计算机上的模块之间的接口规范，以及那些模块实现的通信服务。这个术语也用来表示满足规范的模块实现。为了区别这两种用法，接口通常叫作协议规范（protocol specification）。

proxy（代理）：位于客户与获取消息并提供服务的服务器之间的一个智能体。例如，代理可以充当服务器的“替身”来响应客户的请求，可能使用它缓存的数据，而不必联系服务器。

pseudoheader（伪首部）：IP 首部中的字段的子集，向上传递给传输层协议 TCP 和 UDP，以便在它们计算校验和时使用。伪首部包括源和目的 IP 地址以及 IP 数据报长度，因此能够用来检测这些字段是否被破坏或一个分组是否被传送到不正确的地址。

public key encryption（公钥加密）：一种加密算法（如 RSA），其中每个参与者有一个私钥（不与任何人共享）和一个公钥（每人都可以获得）。一条保密消息使用用户的公钥加密数据后发送给用户。由于需要用私钥解密消息，因此只有接收方可以读消息。

QoS（Quality of Service，服务质量）：由网络体系结构提供的分组传送保证。通常与性能保证有关，如带宽和延迟。因特网提供尽力传输服务，即尽力传输一个分组，但不确保正确传输。

RED（Random Early Detection，随机早期检测）：路由器的一种排队规则，当预测到拥塞时，随机丢弃分组，警告发送方放慢发送速率。

rendezvous point（汇集点）：PIM 所使用的允许接收方了解发送方的路由器。

repeater（中继器）：一种将电信号从一个以太网电缆传播到另一个以太网电缆的设备。一个以太网的任何两台主机之间最多可能有两台中继器。中继器转发信号，而网桥（bridge）转发帧（frame），路由器（router）和交换机（switch）转发分组（packet）。

REST（Representational State Transfer，表述性状态转移）：使用 HTTP 作为通用应用协议构建 Web 服务的一种方法。

RFC（Request For Comments，请求评论）：因特网报告，其中包含像 TCP 和 IP 这样的协议规范。

RIP（Routing Information Protocol，路由信息协议）：Berkeley Unix 提供的域内路由协议。每个运行 RIP 的路由器动态地建立基于距离向量（distance-vector）算法的转发表。

router（路由器）：连接两个或多个网络的网络节点，把分组从一个网络转发到另一个网络。对比网桥（bridge）、中继器（repeater）和交换机（switch）。

routing（路由）：一个进程，节点通过它交换用于构建正确转发表的拓扑结构信息。参见转发（forwarding）、链路状态（link state）和距离向量（distance vector）。

routing table（路由表）：参见转发表（forwarding

table)。

RPB (Reverse Path Broadcast, **逆向路径广播**)：用于消除重复广播分组的一种技术。

RPC (Remote Procedure Call, **远程过程调用**)：很多客户/服务器交互使用的同步请求/应答传输协议。

RPR (Resilient Packet Ring, **弹性分组环**)：主要用于城域网的一种环网类型。参见 802.17。

RSA：一种公共密钥加密算法，根据发明者 Rivest、Shamir、Adleman 的名字命名。

RSVP (Resource Reservation Protocol, **资源预留协议**)：在网络中预留资源的协议。RSVP 使用路由器中软状态 (soft state) 的概念，将预留资源的责任放在接收方而不是发送方。

RTCP (Real-time Transport Control Protocol, **实时传输控制协议**)：与 RTP 相关的控制协议。

RTP (Real-time Transport Protocol, **实时传输协议**)：有实时性约束的多媒体应用所使用的端到端协议。

RTT (Round-Trip Time, **往返时间**)：信息的一个比特从链路或信道的一端到另一端再返回所用的时间，换句话说，它是信道时延的两倍。

SAN (System Area Network, **系统区域网络**)：跨越一个计算机系统不同部分（如显示器、摄像头、磁盘）的网络。有时代表存储区域网 (storage area network)，包括像 HiPPI 和光纤信道这样的接口。对比 LAN、MAN 和 WAN。

schema (**大纲**)：说明如何构造和解释一组数据的规范，是为 XML 文档定义的。

scrambling (**扰频**)：在传输前将一个信号与伪随机比特流进行异或操作的进程，以便能实现时钟恢复所需的足够的信号转变。扰频用于 SONET 中。

SDP (Session Description Protocol, **会话描述协议**)：用于获悉音频/视频信道可用性的应用层协议。它记录会话的名称和目的、开始和结束的时间、媒体类型（例如音频、视频）以及接收会话的详细信息（例如多播地址、传输协议和使用的端口号）。

SHA (Secure Hash Algorithm, **安全散列算法**)：一系列加密散列算法中的一个。

segment (**报文段**)：一个 TCP 分组。一个报文段包含的正在由 TCP 发送的字节流的一部分。

SELECT：用于构建远程过程调用协议的一个同步多路分解协议。

semaphore (**信号量**)：用于支持进程间同步的一个变量。通常，一个进程加塞 (block) 在一个信号量上，等待另一个进程给此信号量发信号。

server (**服务器**)：客户/服务器分布式系统中的服务提供者。

signalling (**信令**)：在物理层，它表示经过某种物理介质的信号的传输。在 ATM 中，信令是指建立虚电路的进程。

silly window syndrome (**傻瓜窗口症状**)：TCP 中发生的一种情况。发送方发送一个小数据段就能填满窗口，如果每次接收方打开接收窗口很小时就会发生这种情况，结果是产生很多小数据段，不能充分使用带宽。

SIP (Session Initiation Protocol, **会话初始化协议**)：用于多媒体应用的应用层协议。它确定与特定的用户通信的恰当设备，确定用户是否愿意或能够参加一个特定的通信会话，确定选择使用哪种介质和编码方案，并建立会话的参数（如端口号）。

sliding window (**滑动窗口**)：允许发送方在接收到一个确认之前发送多个（最多是窗口的尺寸）分组的算法。当窗口中先被发出的那些分组的确认返回后，窗口就“滑动”，然后可以发送更多的分组。滑动窗口算法将可靠传输与高吞吐量结合起来。参见 ARQ。

slow start (**慢启动**)：TCP 的一个防止拥塞的算法，尝试协调输出数据段的速度。对于每个返回的 ACK，再发送两个分组，使得未传输的数据段的数目呈指数增长。

SMTP (Simple Mail Transfer Protocol, **简单邮件传输协议**)：因特网的电子邮件协议。参见 822。

SNA (System Network Architecture, **系统网络体系结构**)：IBM 的专有网络体系结构。

SNMP (Simple Network Management Protocol, **简单网络管理协议**)：一个允许监视主机、网络和路由器的因特网协议。

SOAP：用于定义和实现应用协议的 Web 服务框架的组件。

socket (**套接字**)：由 Unix 提供的对 TCP/IP 应用程序的应用编程接口 (API) 的抽象。

soft state (**软状态**)：包含在路由器中的与连接有关的信息，它被缓存一段有限的时间，而不是通过连接建立阶段被明确建立（并需要

明确解释）。

SONET（Synchronous Optical Network，同步光纤网络）：在光纤上进行数字传输的基于时钟的里程标准，定义电话公司如何在光纤网络上传输数据。

source routing（源路由）：指在分组被发送之前，由源端来做出路由选择决定。路由包括分组到目的端途中经过的节点列表。

source-specific multicast（特定源多播）：一种多播方式，其中一个组可以只有一个发送方。

sparse mode multicast（稀疏模式多播）：当相对的少数主机或路由器需要接收某个组的多播数据时在 PIM 中使用的模式。

split horizon（水平分割）：在距离向量路由算法中打破路由循环的一种方法。当一个节点发送一条路由更新信息给它的邻居时，并不把从每个邻居处得知的路由再发还给那个邻居。水平分割和毒性逆转一起使用。

spread spectrum（扩频）：一种编码技术，在比需要的更宽的频率上扩展一个信号，使得干扰的影响降至最小。

SSL（Secure Socket Layer，安全套接字层）：运行在 TCP 之上的一个协议层，提供连接的认证和加密，也称为传输层安全（TLS）。

statistical multiplexing（统计多路复用）：在一条共享链路或信道上多数据源的基于请求的多路复用。

stop-and-wait（停止-等待）：一种可靠的传输算法，发送方发送一个分组，然后在发送下一个分组之前等待应答。与滑动窗口（sliding window）和并发逻辑信道（concurrent logical channels）比较。参见 ARQ。

STS（Synchronous Transport Signal，同步传输信号）：SONET 的不同传输速率的前缀。例如，STS-1 是指用于 51.84Mbps 传输的 SONET 标准。

subnetting（划分子网）：用一个 IP 网络地址表示多个物理网络，子网内的路由器使用子网掩码来找到分组应被转发到哪个物理网络。划分子网实际上是在两层结构的 IP 地址中引入另一层。

SunRPC：由 Sun 微系统公司开发的远程过程调用协议。SunRPC 用于支持 NFS。参见 ONC。

switch（交换机）：基于每个分组中的首部信息将分组从输入端转发到输出端的网络节点。与路由器的主要区别是它们通常不用来互联不同类型的网络。

switching fabric（交换机构）：将分组从输入端引导到正确输出端的交换机的部件。

T1：等价于 24 条 ISDN 电路或 1.544Mbps 的一种标准电话载波服务，也称为 DS1。

T3：等价于 24 条 T1 电路或者 44.736Mbps 的一种标准电话载波服务，也称为 DS3。

TCP（Transmission Control Protocol，传输控制协议）：因特网体系结构的面向连接的传输协议，提供可靠的字节流传输服务。

TDMA（Time Division Multiple Access，时分多路复用）：用于蜂窝无线网络的一种多路复用形式，也是一个特定无线标准的名字。

Telnet：因特网体系结构的远程终端协议。Telnet 可以使用户与一个远程系统交互，就好像用户终端直接连在那台计算机上一样。

throughput（吞吐量）：当发送数据通过一个信道时观察到的速率。这个术语有时与带宽（bandwidth）互换使用。

TLS（Transport Layer Security，传输层安全）：可以放在像 TCP 这样的传输协议上的安全服务，通常被 HTTP 用来完成万维网上的安全事务，起源于 SSL。

token bucket（令牌桶）：描述或管理一个流所用带宽的一种方法。从概念上讲，进程随时间积累令牌，并且必须用掉一个令牌来传输数据的一个字节，然后在没有剩余令牌时必须停止发送。这样，总的带宽受到某种突发性调节的限制。

token ring（令牌环）：将主机连成一个环的物理网络技术，一个令牌（比特模式）绕环路循环，一个给定的节点在被允许传输之前必须拥有令牌。802.5 和 FDDI 是令牌环网络的例子。

transport protocol（传输协议）：使不同主机上的进程进行通信的端到端协议。TCP 是一个典型的示例。

TTL（Time to Live，生存期）：它通常是一个 IP 数据报在被丢弃之前能够经历的跳数（路由器）的一个度量值。

tunneling（隧道）：使用运行在与分组同一层上的协议封装一个分组。例如，多播 IP 分组封装在单点播送 IP 分组中，经隧道穿过因特网实现 Mbone。隧道也可用于从 IPv4 到 IPv6 的转换。

two-dimensional parity（二维奇偶校验）：将字节从概念上排列成一个矩阵的奇偶校验方案，同时

计算行和列的奇偶性。

Tymnet：一个早期的网络，在一个路由器集合间维护的一个虚电路（virtual circuit）。

UBR（Unspecified Bit Rate，未指明比特率）：ATM中的“不提供不必要服务”的一类服务，提供尽力的信元传送。对比ABR、CBR和VBR。

UDP（User Datagram Protocol，用户数据报协议）：因特网体系结构的传输协议，提供应用层进程之间的无连接的数据报服务。

UMTS（Universal Mobile Telecommunications System，通用移动通信系统）：基于宽带CDMA的蜂窝无线标准，提供较高的数据传输率。

unicast（单播）：指发送一个分组给一个单独的目的主机。对比广播（broadcast）和多播（multicast）。

URI（Uniform Resource Identifier，统一资源标识符）：推广的URL。例如，与SIP联合使用建立音频/视频会话。

URL（Uniform Resource Locator，统一资源定位符）：用于识别因特网资源位置的一个文本串。一个典型的URL形如http://www.cisco.com。在这个URL中，http是一个协议，用于访问位于主机www.cisco.com上的资源。

vat：因特网上使用的运行在RTP之上的音频会议工具。

VBR（Variable Bit Rate，可变比特率）：ATM中的服务类型之一，用于带宽需求随时间变化的应用，如压缩的视频图像。对比ABR、CBR和UBR。

VCI（Virtual Circuit Identifier，虚电路标识符）：它是分组首部中的一个标识符，用于虚电路交换。在ATM的情况下，VPI和VCI一起标识端到端的连接。

vic：基于Unix的使用RTP的视频会议工具。

virtual circuit（虚电路）：它是由面向连接的网络（如ATM）提供的抽象。要想在参与者之间交换消息，就必须在数据发送之前建立一条虚电路（并可能为虚电路分配资源）。对比数据报（datagram）。

virtual clock（虚拟时钟）：一种服务模型，允许源端使用其需要的基于速率的描述在路由器上预留资源。虚拟时钟不属于当前因特网的尽力传输服务。

VPI（Virtual Path Identifier，虚路径标识符）：ATM首部中的一个8比特或12比特字段。VPI可用于将穿过一个网络的多个虚连接隐藏为一条虚“路径”，从而减少交换机所必须维护的连接状态的数量。参见VCI。

VPN（Virtual Private Network，虚拟专用网）：处在某些现有网络之上的逻辑网络。例如，一个公司在世界范围内有一些站点，它可以在因特网的顶部建一个虚拟网络，而不必租用每个站点之间的线路。

WAN（Wide Area Network，广域网）：指可以跨越远距离（如穿过国家）的任何物理网络技术。对比SAN、LAN和WAN。

well-known port（周知端口）：它通常是一个由某个特殊服务器使用的端口号。例如，域名服务器在每个主机的众所周知的UDP和TCP端口53上接收消息。

weighted fair queuing（WFQ，加权公平排队法）：公平排队法（fair queuing）的一个变种，可以为每个流给定一个网络容量的不同份额。

WSDL（Web Service Description Language，Web服务描述语言）：用于定义和实现应用协议的Web服务框架组件。

WWW（World Wide Web，万维网）：因特网上的一种超媒体信息服务。

X.25：ITU的分组交换协议标准。

X.400：ITU的电子邮件标准，与因特网体系结构中的SMTP相对应。

X.500：ITU的目录服务标准，定义一种基于属性的命名服务。

X.509：ITU的一个数字证书标准。

XDR（External Data Representation，外部数据表示）：Sun微系统公司的与机器无关的数据结构的标准。对比ASN.1和NDR。

XML（Extensible Markup Language，可扩展标记语言）：用于描述因特网应用程序之间传递的数据的一种语法规则。

XSD（XML Schema Definition）：XML大纲定义，用于定义XML对象格式和解释的模型语言。

zone（区域）：域名分层结构的一个划分，对应于负责分层结构中一部分的一个管理机构。每个区域必须至少有两台名字服务器用于答复区域中的DNS请求。

参考文献

[Barr01] Barrett, D. and Silverman, R. *SSH: The Secure Shell*. O'Reilly, Sebastopol, CA, 2001.

[BCGN04] Basagni, S., Conti, M., Giordano, S., and Stojmenovic, I., Eds. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, Hoboken, NJ, 2004.

[Bat68] Batcher, K.E. Sorting networks and their applications. *Proceedings of the AFIPS Spring Joint Computer Conference*, 32:307–314, 1968.

[BZ96] Bennett, J. and Zhang, H. Hierarchical packet fair queueing algorithms. In *Proceedings of the SIGCOMM '96 Symposium*, pages 143–156, August 1996.

[BLFF96] Berners-Lee, T., Fielding, R., and Frystyk, H. Hypertext Transfer Protocol — HTTP/1.0. *Request for Comments* 1945, May 1996.

[BDSZ94] Bharghavan, V., Demers, A., Shenker, S., and Zhang, L. MACAW: a media access protocol for wireless LANs. In *Proceedings of the SIGCOMM '94 Symposium*, pages 212–225, August 1994.

[Bha03] Bhattacharyya, S. An Overview of Source-Specific Multicast. *Request for Comments* 3569, July 2003.

[BABM05] Bicket, J., Aguayo, D., Biswas, S., and Morris, R. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking*, Cologne, Germany, 2005.

[BLNS82] Birrell, A., Levin, R., Needham, R., and Schroeder, M. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25:250–273, April 1982.

[BM05] Biswas, S. and Morris, R. ExOR: Opportunistic multi-hop routing for wireless networks. In *Proceedings of the ACM SIGCOMM 2005 Conference*, Philadelphia, PA, August 2005.

[BG93] Bjorkman, M. and Gunningberg, P. Locking effects in multiprocessor implementations of protocols. In *Proceedings of the SIGCOMM '93 Symposium*, pages 74–83, September 1993.

[Blah87] Blahut, R.E. *Principles and Practice of Information Theory*. Addison-Wesley, Reading, MA, 1987.

[BBC+98] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An Architecture for Differentiated Services. *Request for Comments* 2475, December 1998.

[Boo95] Boorsook, P. How anarchy works. *Wired* 3(10):110–118, October 1995.

[BDMS94] Bowman, C.M., Danzig, P.B., Manber, U., and Schwartz, M.F. Scalable Internet Resource Discovery: Research Problems and Approaches. *Communications of the ACM* 37(8):98–107, August 1994.

[BP90] Bowman, C.M., Peterson, L.L., and Yeatts, A. Univers: An attribute-based name server. *Software—Practice and Experience* 20(4):403–424, April 1990.

[BCS94] Braden, R., Clark, D., and Shenker, S. Integrated Services in the Internet Architecture: An Overview. *Request for Comments* 1633, September 1994.

[BM95] Bradner, S. and Mankin, A., Eds. *IPng: Internet Protocol Next Generation*. Addison-Wesley, Reading, MA, 1995.

[BP95] Brakmo, L.S. and Peterson, L.L. TCP Vegas: End-to-end congestion avoidance on a global internet. *IEEE Journal of Selected Areas in Communication (JSAC)* 13(8):1465–1480, October 1995.

[Bri95] Brisco, T. DNS Support for Load Balancing. *Request for Comments* 1794, April 1995.

[Buf94] Buford, J.F. *Multimedia Systems*. ACM Press/Addison-Wesley, Reading, MA, 1994.

[Cal07] Callas, J. et al. OpenPGP Message Format. *Request for Comments* 4880, November 2007.

[CV95] Chandranmenon, G.P. and Varghese, G. Trading packet headers for packet processing. In *Proceedings of the SIGCOMM '95 Symposium*, pages 162–173, October 1995.

[CHH+06] Chen, K.-T., Huang, C.-Y., Huang, P., and Lei, C.-L. Quantifying Skype user satisfaction. In *Proceedings of the ACM SIGCOMM 2006 Conference*, pp. 399–410, September 2006.

[CLNZ89] Chen, S.K., Lazowska, D., Notkin, D., and Zahorjan, J. Performance implications of design alternatives for remote procedure call stubs. In *Proceedings of the Ninth International Conference on Distributed Computing Systems*, pages 36–41, June 1989.

[CZ85] Cheriton, D.R. and Zwaenepoel, W. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems* 3(2):77–107, May 1985.

[Cla97] Clark, D.D. Internet Cost Allocation and Pricing. In *Internet Economics*. MIT Press, Cambridge, MA, 1997, pages 215–252.

[Cla82] Clark, D.D. Modularity and Efficiency in Protocol Implementation. *Request for Comments* 817, July 1982.

[Cla88] Clark, D.D. The design philosophy of the DARPA internet protocols. In *Proceedings of the SIGCOMM '88 Symposium*, pages 106–114, August 1988.

[Cla85] Clark, D.D. The structuring of systems using upcalls. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 171–180, December 1985.

[CF98] Clark, D. and Fang, W. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking* 6(4):362–373, August 1998.

[CJRS89] Clark, D.D., Jacobson, V., Romkey, J., and Salwen, H. An analysis of TCP processing overhead. *IEEE Communications* 27(6):23–29, June 1989.

[CPB+07] Clark, D.D., Partridge, C., Braden, R., Davie, B.S., Floyd, S., Jacobson, V., Katabi, D., Minshall, G., Ramakrishnan, K.K., Roscoe, T., Stoica, I., Wroclawski, J., and Zhang, L. Making the world (of communications) a different place. *Computer Communication Review* 35(3):91–96, 2005.

[CHC+01] Clausen, T., Hansen, G., Christensen, L., and Behrmann, G. The optimized link state routing protocol, evaluation through experiments and simulation. In *IEEE Symposium on Wireless Personal Mobile Communications*, September 2001.

[Com95] Comer, D.E. *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture*, 5th ed. Prentice Hall, Upper Saddle River, NJ, 2005.

[CP93] Comer, D.E. and Peterson, L.L. Understanding naming in distributed systems. *Distributed Computing* 3(2):51–60, May 1989.

[CS96] Comer, D.E. and Stevens, D.L. *Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, BSD Socket Version*, 2nd ed. Prentice Hall, Englewood Cliffs, NJ, 1996.

[CS97] Comer, D.E. and Stevens, D.L. *Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, Windows Sockets Version*. Prentice Hall, Englewood Cliffs, NJ, 1997.

[CS00] Comer, D.E. and Stevens, D.L. *Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications, Linux/Posix Sockets Version*. Prentice Hall, Upper Saddle River, NJ, 2000.

[CCITT92a] Comité Consultif International de Telegraphique et Telephonique. Open systems interconnection: Specification of abstract syntax notation one (ASN.1). CCIT Recommendation X.208, 1992.

[CCITT92b] Comité Consultif International de Telegraphique et Telephonique. Open systems interconnection: Specification of basic encoding rules for abstract syntax notation one (ASN.1). CCIT Recommendation X.209, 1992.

[Cro82] Crocker, D. Standard for the Format of ARPA Internet Text Message. *Request for Comments* 822, August 1982.

[Dan98] Danzig, P. NetCache architecture and deployment. *Computer Networks and ISDN Systems* 30(22–23):2081–2091, November 1998.

[DCB+02] Davie, B., Charny, A., Bennett, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and Stiliadis, D. An Expedited Forwarding PHB (Per-Hop Behavior). *Request for Comments* 3246, March 2002.

[DD00] Davie, B. and Rekhter, Y. *MPLS: Technology and Applications*. Morgan Kaufmann, San Francisco, CA, 2000.

[DEF+96] Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C., and Wei, L. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking* 4(2):153–162, April 1996.

[DH98] Deering, S. and Hinden, R. Internet Protocol, Version 6 (IPv6) Specification. *Request for Comments* 2460, December 1998.

[DBCP97] Degermark, M., Brodnik, A., Carlsson, S., and Pink, S. Small forwarding tables for fast routing lookups. In *Proceedings of the SIGCOMM '97 Symposium*, pages 3–14, October 1997.

[DR08] Dierks, T. and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. *Request for Comments* 5246, August 2008.

[DP93] Druschel, P. and Peterson, L.L. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 189–202, December 1993.

[DS75] Drysdale, R.L. and Young, F.H. Improved divide/sort/merge sorting networks. *SIAM Journal on Computing* 4(3):264–270, September 1975.

[Eas05] Eastlake 3rd, D. Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). *Request for Comments* 4305, December 2005.

[EVD04] Eatherton, W., Varghese, G., and Dittia, Z. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Computer Communication Review* 34(2):97–122, 2004.

[Eis06] Eisler, M. XDR: External Data Representation Standard. *Request for Comments* 4506, May 2006.

[FHHK06] Fenner, B., Handley, M., Holbrook, H., and Kouvelas, I. Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised). *Request for Comments* 4601, August 2006.

[Fie00] Fielding, R.T. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, 2000 (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm).

[FGM+99] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol—HTTP/1.1. *Request for Comments* 2616, June 1999.

[Flo03] Floyd, S. High speed TCP for Large Congestion Windows. *Request for Comments* 3649, December 2003.

[FHPW00] Floyd, S., Handley, M., Padhye, J., and Widmer, J. Equation-based congestion control for unicast applications. In *Proceedings of the SIGCOMM 2000 Symposium*, pages 43–56, August 2000.

[FB96] Freed, N., and Borenstein, N. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. *Request for Comments* 2045, November 1996.

[FL06] Fuller, V. and Li, T. Classless Inter-Domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. *Request for Comments* 4632, August 2006.

[GR01] Gao, L. and Rexford, J. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking* 9(6):681–692, December 2001.

[Gar00] Garber, L. Technology news: Denial-of-service attacks rip the Internet. *Computer* 33(4):12–17, April 2000.

[GHMS91] Giacopelli, J.N., Hickey, J.J., Marcus, W.S., and Sincoskie, W.D. Sunshine: A high performance self-routing broadband packet switch architecture. *IEEE Journal of Selected Areas in Communication (JSAC)* 9(8):1289–1298, October 1991.

[GG94] Gopal, I. and Guerin, R. Network transparency: The plaNET approach. *IEEE/ACM Transactions on Networking* 2(3):226–239, June 1994.

[GVC96] Goyal, P., Vin, H., and Chen, H. Start-time fair queueing: A scheduling algorithm for Integrated Services packet switching networks. In *Proceedings of the SIGCOMM '96 Symposium*, pages 157–168, August 1996. Also see OSDI '96.

[GK00] Gupta, P. and Kumar, P.R. The capacity of wireless networks. *IEEE Transactions on Information Theory* 46(2):388–404, 2000.

[HRX08] Ha, S., Rhee, I., and Xu, L. CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review* 42(5):64–74, July 2008.

[HC99] Handley, M. and Crowcroft, J. Internet multicast today. *The Internet Protocol Journal* 2(4), December 1999.

[HFPW03] Handley, M., Floyd, S., Padhye, J., and Widmer, J. TCP Friendly Rate Control (TFRC): Protocol Specification. *Request for Comments* 3448, January 2003.

[Har00] Harrison, A. Cyber assaults hit Buy.com, eBay, CNN and Amazon. In *Computerworld*, February 9, 2000.

[HP95] Holzmann, G.J. and Pehrson, B. *The Early History of Data Networks*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[Huf52] Huffman, D.A. A method for the construction of minimal-redundancy codes. *Proceedings of the IRE* 40(9):1098–1101, September 1952.

[HMPT89] Hutchinson, N., Mishra, S., Peterson, L., and Thomas, V. Tools for implementing network protocols. *Software—Practice and Experience* 19(9):895–916, September 1989.

[HP91] Hutchinson, N. and Peterson, L. The *x*-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering* 17(1):64–76, January 1991.

[Jac88] Jacobson, V. Congestion avoidance and control. In *Proceedings of the SIGCOMM '88 Symposium*, pages 314–329, August 1988.

[JBB92] Jacobson, V., Braden, R., and Borman, D. TCP Extensions for High Performance. *Request for Comments* 1323, May 1992.

[Jaf81] Jaffe, J.M. Flow control power is nondecentralizable. *IEEE Transactions on Communications* COM-29(9):1301–1306, September 1981.

[Jai89] Jain, R. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review* 19(5):56–71, October 1989.

[Jai94] Jain, R. *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*. Addison-Wesley, Reading, MA, 1994.

[Jai91] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and*

Modeling. John Wiley & Sons, New York, 1991.

[JK07] Jamieson, K. and Balakrishnan, H. PPR: Partial packet recovery for wireless networks. In *Proceedings of the SIGCOMM 2007 Conference*, pages 409–420, August 2007.

[JPA04] Johnson, D., Perkins, C., and Arkko, J. Mobility Support in IPv6. *Request for Comments* 3775, June 2004.

[KP91] Karn, P. and Partridge, C. Improving round-trip time estimates in reliable transport protocols. *ACM Transactions on Computer Systems* 9(4):364–373, November 1991.

[KHR02] Katabi, D., Handley, M., and Rohrs, C. Congestion control for high bandwidth-delay product networks. In *Proceedings of the ACM SIGCOMM 2002 Conference*, pages 89–102, August 2002.

[KRH+06] Katti, S., Rahul, H., Hu, W., Katabi, D., Médard, M., and Crowcroft, J. XORs in the air: Practical wireless network coding. In *Proceedings of the SIGCOMM 2006 Conference*, pages 243–254, September 2006.

[Kau05] Kaufman, C. Internet Key Exchange (IKEv2) Protocol. *Request for Comments* 4306, December 2005.

[KPS02] Kaufman, C., Perlman, R., and Speciner, M. *Network Security: Private Communication in a Public World*. Prentice Hall, Englewood Cliffs, NJ, 2002.

[Ken05a] Kent, S. IP Authentication Header. *Request for Comments* 4302, December 2005.

[Ken05b] Kent, S. IP Encapsulating Security Payload (ESP). *Request for Comments* 4303, December 2005.

[KM87] Kent, C. and Mogul, J. Fragmentation considered harmful. In *Proceedings of the SIGCOMM '87 Symposium*, pages 390–401, August 1987.

[Kle79] Kleinrock, L. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications*, pages 43.1.1–43.1.10, June 1979.

[Kle75] Kleinrock, L. *Queuing Systems, Volume 1: Theory*. John Wiley & Sons, New York, 1975.

[Kle01] Klensin, J. Simple Mail Transfer Protocol. *Request for Comments* 2821, April 2001.

[KHF06] Kohler, E., Handley, M., and Floyd, S. Datagram Congestion Control Protocol (DCCP). *Request for Comments* 4340, March 2006.

[LAA+00] Labovitz, C., Ahuja, A., Bose, A., and Jahanian, F. Delayed internet routing convergence. In *Proceedings of the SIGCOMM 2000 Symposium*, pages 293–306, August 2000.

[LS98] Lakshman, T.V. and Stiliadis, D. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of the SIGCOMM '98 Symposium*, pages 203–214, September 1998.

[LMKQ89] Leffler, S.J., McKusick, M.K., Karels, M.J., and Quarterman, J.S. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA, 1989.

[LTWW94] Leland, W., Taqqu, M., Willinger, W., and Wilson, D. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking* 2:1–15, February 1994.

[Lin93] Lin, H.-A. Estimation of the optimal performance of ASN.1/BER transfer syntax. *Computer Communications Review* 23(3):45–58, July 1993.

[LS97] Lin, D. and Morris, R. Dynamics of random early detection. In *Proceedings of the SIGCOMM '97 Symposium*, pages 127–136, October 1997.

[LPW+02a] Low, S., Paganini, F., Wang, J., Adlakha, S., and Doyle, J. Dynamics of TCP/AQM and a scalable control. In *Proceedings of IEEE INFOCOM*, June 2002.

[LPW02b] Low, S., Peterson, L., and Wang, L. Understanding TCP Vegas: A duality model. *Journal of the ACM* 49(2):207–235, March 2002.

[MD98] Madson, C. and Doraswamy, N. The ESP DES-CBC Cipher Algorithm with Explicit IV. *Request for Comments* 2405, November 1998.

[MG98a] Madson, C. and Glenn, R. The Use of HMAC-MD5-96 within ESP and AH. *Request for Comments* 2403, November 1998.

[MG98b] Madson, C. and Glenn, R. The Use of HMAC-SHA-1-96 within ESP and AH. *Request for Comments* 2404, November 1998.

[Mal98] Malkin, G. RIP Version 2. *Request for Comments* 2453, November 1998.

[Mas86] Mashey, J. RISC, MIPS, and the motion of complexity. In *UniForum 1986 Conference Proceedings*, pages 116–124, 1986.

[MJ95] McCanne, S. and Jacobson, V. VIC: A flexible framework for packet video. In *Proceedings of the Third ACM International Conference on Multimedia*, pages 511–522, November 1995.

[MP599] McCloghrie, K., Perkins, D., and Schoenwaelder, J. Structure of Management Information Version 2 (SMIv2). *Request for Comments* 2578, April 1999.

[MD93] McKenney, P.E. and Dove, K.F. Efficient demultiplexing of incoming TCP packets. In *Proceedings of the SIGCOMM '92 Symposium*, pages 269–280, August 1993.

[MvOV96] Menezes, A.F., van Oorschot, P.C., and Vanstone, S.A. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996.

[Min93] Minoli, D. *Enterprise Networking: Fractional T1 to SONET, Frame Relay to BISDN*. Artech House, Norwood, MA, 1993.

[MPFL96] Mitchell, J.L., Pennebaker, W.B., Fogg, C.E., and LeGall, D.J. *MPEG Video: Compression Standard*. Chapman Hall, London, 1996.

[Mog95] Mogul, J. The case for persistent-connection HTTP. In *Proceedings of the SIGCOMM '95 Symposium*, pages 299–313, August 1995.

[MD90] Mogul, J. and Deering, S. Path MTU Discovery. *Request for Comments* 1191, November 1990.

[MP85] Mogul, J. and Postel, J. Internet Standard Subnetting Procedure. *Request for Comments* 950, August 1985.

[MVS01] Moore, D., Voelker, G., and Savage, S. Inferring internet denial of service activity. In *Proceedings of 2001 USENIX Security Symposium*, August 2001.

[Mor68] Morrison, D. PATRICIA—a practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM* 15(4):514–534, October 1968.

[MPBO96] Mosberger, D., Peterson, L., Bridges, P., and O'Malley, S. Analysis of techniques to improve protocol latency. In *Proceedings of the SIGCOMM '96 Symposium*, pages 73–84, August 1996.

[Moy98] Moy, J. OSPF Version 2. *Request for Comments* 2328, April 1998.

[Mul90] Mullender, S. Amoeba: A distributed operating system for the 1990s. *IEEE Computer* 23(5):44–53, May 1990.

[NYKT94] Nahum, E.M., Yates, D.J., Kurose, J.F., and Towsley, D. Performance issues in parallelized network protocols. In *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI)*, pages 125–137, November 1994.

[NRC01] National Research Council. *Looking Over the Fence at Networks*. National Academy Press, Washington, D.C., 2001.

[NRC94] National Research Council, Computer Science and Telecommunications Board. *Realizing the Information Future: The Internet and Beyond*. National Academy Press, Washington, D.C., 1994.

[Nel92] Nelson, M. *The Data Compression Book*. M&T Books, San Mateo, CA, 1992.

[Nol97] Noll, P. MPEG digital audio coding. *IEEE Signal Processing Magazine*, pages 59–81, September 1997.

[OP91] O'Malley, S. and Peterson, L. TCP Extensions Considered Harmful. *Request for Comments* 1263, October 1991.

[OPM94] O'Malley, S.W., Proebsting, T.A., and Montz, A.B. Universal stub compiler. In *Proceedings of the SIGCOMM '94 Symposium*, pages 295–306, August 1994.

[ORG02] Ong, L. and Yoakum, J. An Introduction to the Stream Control Transmission Protocol (SCTP). *Request for Comments* 3286, May 2002.

[OSF91] Open Software Foundation. *OSF DCE Application Environment Specification*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[OCD+88] Ousterhout, J.K., Cherenson, A.R., Douglis, F., Nelson, M.N., and Welch, B.B. The Sprite network operating system. *IEEE Computer* 21(2):23–36, February 1988.

[PFTK98] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM '98 Symposium*, pages 303–314, September 1998.

[Pad85] Padlipsky, M.A. *The Elements of Networking Style and Other Essays and Animadversions on the Art of Intercomputer Networking*. Prentice Hall, Englewood Cliffs, NJ, 1985.

[PG94] Parekh, A. and Gallagher, R. A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE/ACM Transactions on Networking* 2(2):137–150, April 1994.

[PL01] Park, K. and Lee, H. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proceedings of the ACM SIGCOMM 2001 Conference*, pages 15–26, August 2001.

[Par+98] Partridge, C. et al. A 50 Gb/s IP router. *IEEE/ACM Transactions on Networking* 6(3):237–247, June 1998.

[PF94] Paxon, V. and Floyd, S. Wide-area traffic: The failure of Poisson modeling. In *Proceedings of the SIGCOMM '94 Symposium*, pages 257–268, August 1994.

[Per02] Perkins, C., Ed. IP mobility support for IPv4. *Request for Comments* 3344, August 2002.

[PM97] Perkins, D. and McGinnis, E. *Understanding SNMP MIBS*. Prentice Hall, Upper Saddle River, NJ, 1997.

[Per00] Perlman, R. *Interconnections: Bridges, Routers, Switches and Internetworking Protocols*, 2nd ed. Addison-Wesley, Reading, MA, 2000.

[Pet88] Peterson, L.L. The Profile naming service. *ACM Transactions on Computer Systems* 6(4):341–364, November 1988.

[PAC+03] Peterson, L., Anderson, T., Culler, D., and Roscoe, T. A blueprint for introducing disruptive technology into the Internet. *ACM SIGCOMM Computer Communication Review* 33(1):59–64, January 2003.

[PB61] Peterson, W.W. and Brown, D.T. Cyclic codes for error detection. *Proceedings of the IRE*, 49:228–235, January 1961.

[Pie84] Pierce, J. Telephony—a personal view. *IEEE Communications* 22(5):116–120, May 1984.

[Pos81] Postel, J. Internet Protocol. *Request for Comments* 791, September 1981.

[Pos82] Postel, J. Simple Mail Transfer Protocol. *Request for Comments* 821, August 1982.

[Pre02] Presuhn, R. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). *Request for Comments* 3416, December 2002.

[QPP02] Qie, X., Pang, R., and Peterson, L. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, pages 45–60, December 2002.

[RS02] Rabinovich, M. and Spatscheck, O. *Web Caching and Replication*. Addison-Wesley, Reading, MA, 2002.

[RFB01] Ramakrishnan, K., Floyd, S., and Black, D. The Addition of Explicit Congestion Notification (ECN) to IP. *Request for Comments* 3168, September 2001.

[RS01] Ramaswami, R. and Sivarajan, K. *Optical Networks: A Practical Perspective*, 2nd ed. Morgan Kaufmann, San Francisco, CA, 2001.

[LC89] Rao, T.R.N. and Fujiwara, E. *Error-Control Coding for Computer Systems*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[RHE99] Rejaie, R., Handley, M., and Estrin, D. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. *INFOCOM* 3:1337–1345, March 1999.

[RDR+97] Rekhter, Y., Davie, B., Rosen, E., Swallow, G., Farinacci, D., and Katz, D. Tag switching architecture overview. *Proceedings of the IEEE* 82(12):1973–1983, December 1997.

[Res01] Resnick, P. Internet Message Format. *Request for Comments* 2822, April 2001.

[RXV00] Rhee, I., Ozdemir, V., and Yi, Y. TEAR: TCP Emulation at Receivers—Flow Control for Multimedia Streaming. Technical Report, Department of Computer Science, North Carolina State University, April 2000.

[Ste94] Stevens, W.R. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, MA, 1994.

[SW95] Stevens, W.R. and Wright, G.R. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, Reading, MA, 1995.

[Rit84] Ritchie, D. A stream input-output system. *AT&T Bell Laboratories Technical Journal* 63(8):311–324, October 1984.

[Rob93] Robertazzi, T.G., Ed. *Performance Evaluation of High Speed Switching Fabrics and Networks: ATM, Broadband ISDN, and MAN Technology*. IEEE Press, Piscataway, NJ, 1993.

[RR06] Rosen, E. and Rekhter, Y. BGP/MPLS IP Virtual Private Networks (VPNs). *Request for Comments* 4364, February 2006.

[Sal78] Saltzer, J. Naming and binding of objects. In *Operating Systems—An Advanced Course*, Bayer, R. et al., Eds., Lecture Notes on Computer Science, Vol. 60, pages 99–208. Springer-Verlag, Heidelberg, 1978.

[SK09] Saltzer, J.H. and Kaashoek, M.F. *Principles of Computer System Design: An Introduction*. Morgan Kaufmann, San Francisco, CA, 2009.

[SRC84] Saltzer, J., Reed, D., and Clark, D. End-to-end arguments in system design. *ACM Transactions on Computer Systems* 2(4):277–288, November 1984.

[SCH+99] Savage, S., Collins, A., Hoffman, E. Snell, J., and Anderson, T. The end-to-end effects of Internet path selection. In *Proceedings of the ACM SIGCOMM '99 Conference*, pages 289–299, September 1999.

[SWKA00] Savage, S., Wetherall, D., Karlin, A., and Anderson, T. Practical network support for IP traceback. In *Proceedings of the ACM SIGCOMM 2000 Conference*, pages 295–306, August 2000.

[Sch95] Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, New York, 1995.

[SB89] Schroeder, M.D. and Burrows, M. Performance of Firefly RPC. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 83–90, December 1989.

[SCJ+02] Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. Sip: Session Initiation Protocol. *Request for Comments* 3261, June 2002.

[SC03] Schulzrinne, H. and Casner, S. RTP Profile for Audio and Video Conferences with Minimal Control. *Request for Comments* 3551, July 2003.

[SCFJ03] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. RTP: A Transport Protocol for Real-Time Applications. *Request for Comments* 3550, July 2003.

[Sha48] Shannon, C. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 623–656, 1948.

[Sho78] Schoch, J. Inter-network naming, addressing, and routing. In *Proceedings of the Seventeenth IEEE Computer Society International Conference (COMPCON)*, pages 72–79, September 1978.

[SS98] Sisalem, D. and Schulzrinne, H. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. In *Proceedings of Workshop on Network and Operating System Support for Digital Audio and Video, 1998*, July 1998.

[SPS+01] Snoeren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Kent, S.T., and Strayer, W.T. Hash-based IP traceback. In *Proceedings of the ACM SIGCOMM 2001 Conference*, pages 3–14, August 2001.

[SP99] Spatscheck, O. and Peterson, L.L. Defending against denial of service attacks in Scout. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 59–72, February 1999.

[SHP91] Spragins, L., Hammond, J., and Pawlikowski, K. *Telecommunications: Protocols and Design*. Addison-Wesley, Reading, MA, 1991.

[SVSW98] Srinivasan, V., Vargese, G., Suri, S., and Waldvogel, M. Fast scalable level four switching. In *Proceedings of the SIGCOMM '98 Symposium*, pages 191–202, September 1998.

[Sta03] Stallings, W. *Cryptography and Network Security*, 3rd ed. Prentice Hall, Upper Saddle River, NJ, 2003.

[Sta07] Stallings, W. *Data and Computer Communications*, 8th ed. Prentice Hall, Upper Saddle River, NJ, 2007.

[Sta00] Stallings, W. *Local and Metropolitan Area Networks*, 6th ed. Prentice Hall, Upper Saddle River, NJ, 2000.

[SPW02] Staniford, S., Paxson, V., and Weaver, N. How to own the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.

[Ste07] Stewart, R., Ed. Stream Control Transmission Protocol. *Request for Comments* 4960, September 2007.

[SZ97] Stica, I. and Zhang, H. A hierarchical fair service curve algorithm for link-sharing and priority services. In *Proceedings of the SIGCOMM '97 Symposium*, pages 29–262, October 1997.

[SSZ98] Stoica, I., Shenker, S., and Zhang, H. Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocation in high-speed networks. In *Proceedings of the ACM SIGCOMM '98 Symposium*, pages 33–46, August 1998.

[Tan07] Tanenbaum, A.S. *Modern Operating Systems*, 3rd ed. Prentice Hall, Upper Saddle River, NJ, 2007.

[Ter86] Terry, D. Structure-free name management for evolving distributed environments. In *Sixth International Conference on Distributed Computing Systems*, pages 502–508, May 1986.

[TL93] Thekkath, C.A. and Levy, H.M. Limits to low-latency communication on high-speed networks. *ACM Transactions on Computer Systems* 11(2):179–203, May 1993.

[Tur85] Turner, J.S. Design of an integrated services packet network. In *Proceedings of the Ninth Data Communications Symposium*, pages 124–133, September 1985.

[VL87] Varghese, G. and Lauck, T. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 25–38, November 1987.

[VD10] Vasseur, J.P. and Dunkels, A. *Interconnecting Smart Objects with IP*. Morgan Kaufmann, San Francisco, CA, 2010.

[WVTP97] Waldvogel, M., Varghese, G., Turner, J., and Plattner, B. Scalable high speed routing lookups. In *Proceedings of the SIGCOMM '97 Symposium*, pages 25–36, October 1997.

[WC91] Wang, Z. and Crowcroft, J. A new congestion control scheme: Slow start and search (Tri-S). *ACM Computer Communication Review* 21(1):32–43, January 1991.

[WC92] Wang, Z. and Crowcroft, J. Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm. *ACM Computer Communication Review* 22(2):9–16, April 1992.

[WPP02] Wang, L., Pai, V., and Peterson, L. The effectiveness of request redirection on CDN robustness. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 345–360, December 2002.

[Wat81] Watson, R. Identifiers (naming) in distributed systems. In *Distributed System—Architecture and Implementation*, Lampson, B., Paul, M., and Siegert, H., Eds., pages 191–210. Springer-Verlag, New York, 1981.

[WM87] Watson, R.W. and Mamrak, S.A. Gaining efficiency in transport services by appropriate design and implementation choices. *ACM Transactions on Computer Systems* 5(2):97–120, May 1987.

[WJLH06] Wei, D., Jin, C., Low, S., and Hegde, S. FAST TCP: Motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking* 14(6):1246–1259, December 2006.

[WWD07] Weise, M. and Weynand, D. *How Video Works: From Analog to High Definition*. Focal Press, Boston, MA, 2007.

[Wel84] Welch, T. A technique for high-performance data compression. *IEEE Computer* 17(6):8–19, June 1984.

[Wil00] Williamson, B. *Developing IP Multicast Networks, Volume I*. Cisco Press, Indianapolis, IN, 2000.

[WMB99] Witten, I.H., Moffat, A., and Bell, T.C. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 1999.

[WYLB06] Wong, S., Yang, H., Lu, S., and Bharghavan, V. Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, pages 146–157, September 2006.

[XCL10] Xiao, Y., Chen, H., and Li, F., Eds. *Handbook on Sensor Networks*. World Scientific, Singapore, 2010.

[YWA08] Yang, X., Wetherall, D., and Anderson, T.E. TVA: A DoS-limiting network architecture. *IEEE/ACM Transactions on Networking*, 16(6):1267–1280, 2008.

[YHA87] Yeh, Y.-S., Hluchyj, M.B., and Acampora, A.S. The knockout switch: A simple, modular architecture for high-performance packet switching. *IEEE Journal of Selected Areas in Communication (JSAC)* 5(8):1274–1283, October 1987.

[ZDE+93] Zhang, L., Deering, S., Estrin, D., Schenker, S., and Zappala, D. RSVP: A new resource reservation protocol. *IEEE Network* 7(9):8–18, September 1993.

[ZL77] Ziv, J. and Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23(3):337–343, May 1977.

[ZL78] Ziv, J. and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 24(5):530–536, September 1978.