清华大学出版社计算机学术著作出版基金

评审委员会

中国计算机学会学术著作丛书

名誉主任委员：张效祥

主任委员：唐泽圣

副主任委员：陆汝钤

委　　员：（以姓氏笔画为序）

王　珊　吕　建　李晓明

林惠民　罗军舟　郑纬民

施伯乐　焦金生　谭铁牛

序

Preface

第一台电子计算机诞生于20世纪40年代。到目前为止，计算机的发展已远远超出了其创始者的想象。计算机的处理能力越来越强，应用面越来越广，应用领域也从单纯的科学计算渗透到社会生活的方方面面：从工业、国防、医疗、教育、娱乐直至人们的日常生活，计算机的影响可谓无处不在。

计算机之所以能取得上述地位并成为全球最具活力的产业，原因在于其高速的计算能力、庞大的存储能力以及友好灵活的用户界面。而这些新技术及其应用有赖于研究人员多年不懈的努力。学术研究是应用研究的基础，也是技术发展的动力。

自1992年起，清华大学出版社与广西科学技术出版社为促进我国计算机科学技术与产业的发展，推动计算机科技著作的出版，设立了“计算机学术著作出版基金”，并将资助出版的著作列为中国计算机学会的学术著作丛书。时至今日，本套丛书已出版学术专著近50种，产生了很好的社会影响，有的专著具有很高的学术水平，有的则奠定了一类学术研究的基础。中国计算机学会一直将学术著作的出版作为学会的一项主要工作。本届理事会将秉承这一传统，继续大力支持本套丛书的

出版，鼓励科技工作者写出更多的优秀学术著作，多出好书，多出精品，为提高我国的知识创新和技术创新能力，促进计算机科学技术的发展和进步作出更大的贡献。

中国计算机学会

2002 年 6 月 14 日

前言

Foreword

对等网络(Peer-to-Peer network,简称P2P网络)是分布式系统和计算机网络相结合的产物,它在网络协议的应用层,打破过去的"客户/服务器"模式,让所有网络成员享有"自由、平等、互联"的功能,不再有客户、服务器之分,任何两个网络结点之间都能共享文件、传递消息。对等网络起源于1999年风行一时的音乐文件共享软件Napster,随后则是一系列人们耳熟能详的网络软件:Gnutella*,KaZaA,BitTorrent,eDonkey/eMule,Skype,等等。虽然从产生到现在只有短短几年的历史,但是对等网络在应用领域和学术界获得了广泛的重视和成功,并占据了当前Internet超过一半的带宽资源,被称为"改变Internet的新一代网络技术"。

在应用领域,1999年出现的第一个P2P网络Napster,在美国6个月内即拥有5000万注册用户,成为网络时代的一个奇迹。此后许多计算机公司开始投资P2P网络的开发,如著名的无结构P2P网络Gnutella、KaZaA和eDonkey/eMule。2002年,家喻户晓的BitTorrent网络出现,其简称"BT"已成为"自由下载、文件共享"的代

---

* 注意Gnutella的发音为[nju: 'telə],同new-tella,字母G不发音。

名词。BT 在中国的应用尤为广泛，2006 年发布的中国互联网统计报告显示：中国 1.11 亿网民中有 27.8%的人使用过 BT 软件（超过 3000 万人），其流行程度由此可见一斑。除了文件共享，P2P 软件在其他多个应用领域也带来了冲击性的变革：网络语音电话软件 Skype 直接威胁到电话公司的利益，网络电视软件如 PPLive、TvAnts 等使得人们能通过 Internet 流畅地观看电视直播，诸如此类的应用还有很多。

在学术界，虽然 P2P 的思想起源很早并一直有人提及，但直到 2001 年“P2P 网络”这一概念才获得人们的共识并成为研究热点。计算机领域许多重要会议（如 SIGCOMM，SPAA，PODC，ICNP，INFOCOM，ICDCS 等）和刊物（如 IEEE/ACM Transactions on Networking，IEEE Transactions on Parallel and Distributed Systems，IEEE Journal on Selected Areas in Communications 等）从 2001 年开始发布和刊登 P2P 领域的论文，同年出现了 P2P 领域最著名的结构化网络模型：Chord，CAN，Tapestry 和 Pastry。P2P 领域自身的两个专业会议 IEEE P2P 和 IPTPS 分别于 2001 年、2002 年举行。人们提出了许多新颖、独特的 P2P 网络模型，开发了许多用以实现、完善 P2P 网络的核心机制及增强机制，其中不少理论成果已经转化为 Internet 上的实用软件。

到目前为止，P2P 还是一项比较新的技术，其中还存在很多问题，同时也存在着各种挑战和机遇。虽然有太多的问题和障碍，但 P2P 从无到有，从一个民间小软件发展到改变 Internet 面貌、改变人们交流方式的一项新技术，这一过程是值得人们回味的，其中蕴藏的力量和意义，值得我们付出勇气、热情、耐心去努力研究并发掘。

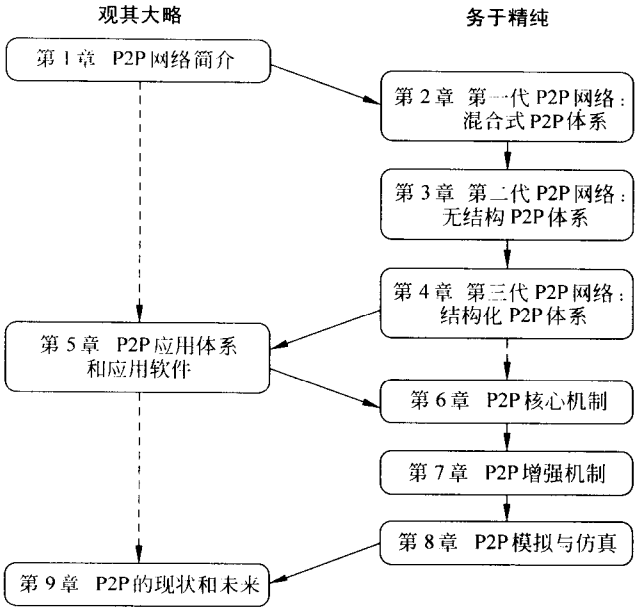
阅读本书的方法

追比圣贤，原是读书人的本意。《三国演义》用这样一段话来描述诸葛亮读书：“孔明与博陵崔州平、颍川石广元、汝南孟公威与徐元直四人为密友。此四人务于精纯，惟孔明独观其大略。尝抱膝长吟，而指四人曰：‘公等仕进可至刺史、郡守。’众问孔明之志若何，孔明但笑而不答。每常自比管仲、乐毅，其才不可量也。”

我们以为这段话，正说出了读书的要义。做学问不必执著于一事一物，对于大多数事情，只要能“观其大略”，明白是怎么回事就足够了；生命有限，青春短暂，我们要做的，是关注那些真正对自己有价值的东西，将精力集中于真正值得去研究的问题，努力“务于精纯”。

这也正是我们希望的读者阅读本书的方法。我们真心希望您付出时间之

后能有所得，所以建议读者按照如下线索来阅读本书：



给读者的鼓励

第一次读这本书时你或许不能完全理解，但不用担心，实际上作者在撰写本书时很多相关论文第一次也没看懂，比如 Freenet 的论文就至少看了 5 遍！读者完全可以跳过暂时看不懂的内容，因为我们在写作时已努力使各章各节尽量独立，当你往后看得更多时，前面感到困惑的内容很可能就豁然开朗了。

“事理因人言而悟者，有悟还有迷，总不如自悟之了了。”在阅读本书的时候，千万不要忘记您自己的批判和思考。无论是批评之言，还是建议之语，希不吝赐教。我们的信箱是：gchen@nju.edu.cn 或 lizhenhua@dislab.nju.edu.cn。

或许世上只有两种人能成为作家：痛苦的或快乐的，因为只有不一样的心情才能承受写作的漫漫孤寂。在作者心中，只能靠想象去构思他的读者空间，所以我们希望能看到、听到您任何形式的反馈，感受到您阅读时的欢欣、困惑与体会。只有这样，我们所构思的空间才能变得真实、饱满，让我们觉得写作时所有的单调、枯燥与艰辛，都是值得付出的。

愿本书带给您的，不仅仅是技术和知识！

参考文献来源

本书所引用的参考文献绝大多数来自于网络通信领域的重要国际会议和刊物，文献发表时间集中于2001—2007年之间，以保证内容的权威性和时效性；除此之外，我们也参考了大量Web资源，尤其是P2P相关网站上的资料。我们所关注的会议、刊物列举如下：

缩写名称	英文全称	中文名称
SIGCOMM	ACM Annual Conference of the Special Interest Group on Data Communication	ACM数据通信特别兴趣组年会
SPAA	ACM Symposium on Parallel Algorithms and Architectures	ACM并行算法和体系讨论会
PODC	ACM Symposium on Principles of Distributed Computing	ACM分布式计算原理讨论会
ICNP	International Conference on Network Protocols	国际网络协议会议
INFOCOM	IEEE Conference on Computer Communications	IEEE计算机通信会议
NOSSDAV	ACM International Workshop on Network and Operating System Support for Digital Audio and Video	ACM数字音频和视频的网络和操作系统支持讨论会
SOSP	ACM Symposium on Operating Systems Principles	ACM操作系统原理讨论会
OSDI	USENIX Symposium on Operating Systems Design and Implementation	USENIX操作系统设计和实现讨论会
USITS	USENIX Symposium on Internet Technologies and Systems	USENIX因特网技术和系统讨论会
ASPLOS	International Conference on Architectural Support for Programming Languages and Operating Systems	国际程序设计语言和操作系统的体系支持会议
ICDCS	International Conference on Distributed Computing Systems	国际分布式计算系统会议
ICPP	International Conference on Parallel Processing	国际并行处理会议
IPDPS	International Parallel and Distributed Processing Symposium	国际并行和分布式处理讨论会
IPTPS	International Workshop on Peer-to-Peer Systems	国际P2P系统讨论会
IEEE P2P	IEEE International Conference on Peer-to-Peer Computing	IEEE国际P2P计算会议
刊物名一般不缩写	IEEE/ACM Transactions on Networking	IEEE/ACM网络学报
	IEEE Transactions on Parallel and Distributed Systems	IEEE并行和分布式系统学报
	IEEE Journal on Selected Areas in Communications	IEEE通信精选领域杂志

致谢

本书的写作、出版得到两大项目的支持：一是2005年国家自然科学基金项目“实用化对等网络技术的研究”；二是2005年江苏省自然科学基金前期预研项目“新型P2P计算技术的基础研究”。香港科技大学倪明选教授和刘云浩教授对本书的写作给予了有益的指导，华中科技大学金海教授和国防科技大学肖侬教授通读了本书的初稿，并提出了许多有价值的建议。邱彤庆、谢军峰、袁瑞峰、叶懋、刘之育、陈欢、于南南等同学对部分章节提出了修改建议。在此谨表示衷心的感谢！

陈贵海　李振华

2007年5月

于南京大学鼓楼校区

目 录

Contents

第1章 P2P网络简介

1.1 什么是P2P网络

计算机是20世纪人类最伟大的发明之一，它的出现改变了人类几千年来对信息存储、表示、处理的方式，也极大地影响了人类生活与思考的方式。

计算机网络(computer network)是相互连接的多台计算机的集合体，人们使用计算机并通过网络互相交流信息，同时扩展计算机的功能。计算机网络的影响深入到人类生活的各个方面，它作为一种新的媒体改变了人类的交流方式，同时也改变了人们对计算机能力的评价——计算机的能力不仅仅在于它的处理速度和存储容量，更重要的是它们之间的连接方式。

因特网(Internet)是计算机网络中最耀眼的明星，它是世界上最大的广域网，连接了地球上几乎每个角落的计算机，是人类最广泛、最有效的信息交流平台。任何一台计算机只要接入因特网，它就潜在的拥有了世界上最大容量的信息仓库和最高速度的计算能力，从这种意义上说，因特网将单台计算机的能力扩展到了极致。

计算机网络自产生以来，其管理与控制就有两种不同的方式：集中式与分布式。集中式系统(centralized system)中的结点在功能上是不平等的，总会有一些通常是少数的管理

结点(server,服务器)在系统中占据中心、主导的地位,管理其他从属结点(client,客户)可执行的操作,控制其他结点之间的信息交换。分布式系统(distributed system)将管理和控制分散到它的各个成员结点中去,结点之间在功能上是平等的,没有谁拥有比其他人更多的特权。集中式系统的优势在于管理的集中化,能够对整个系统进行有效的控制,但它的优势也正是它的缺陷。由于所有结点之间的信息交换都要经过服务器,服务器本身成为限制系统工作效率和规模扩展的瓶颈;分布式系统刚好相反,由于管理的分散化,对整个系统的控制不像集中式那样强,但是由于信息交换的自由、平等,分布式系统常常拥有远远高于集中式系统的工作效率和规模的可扩展性。在实际情况中,很多网络系统并不是采用纯集中式或纯分布式的极端方式,而是兼有两方面的特性,称为混合式系统(hybrid system)。

因特网是最大的计算机网络,同样,它自诞生以来也一直存在着集中式与分布式两种不同的工作方式。客户/服务器模式(client/server mode,简称 C/S)是因特网最传统、最成熟的集中式工作模式,许多重要的因特网应用协议(如 HTTP、FTP、SMTP 等)采用了这一模式。在这种集中式的模式下,服务器将一直运行,被动地等待客户的主动接入,客户将请求发给服务器,服务器返回给客户所要的信息。客户/服务器模式在因特网的最初阶段工作得非常好,然而,随着因特网在规模上不断膨胀、在功能上不断扩展,服务器的负担越来越重,客户/服务器模式的低效率与难以扩展的缺陷暴露出来,它不再能适应需要极高效率与巨大规模的现代因特网。

"是人类的需求,真正推动了技术的革命。"每当人类所发明的技术不再能适应人类本身需求的时候,一定会有人提出新的思想、发明新的技术来解决现实与需求之间的矛盾。当传统的客户/服务器模式不再能适应现代因特网需求的时候,人们将目光重新放回到长久被忽视的分布式系统上,对等模式(peer-to-peer mode,简称 P2P)正是在这种情况下受到重视并很快成为研究热点。"Peer"一词在英文中的意思是"同等、对等的人",故而"peer-to-peer"译为"对等(计算)";国内很多人将 P2P 称作"点对点",这是不恰当的,因为"点对点"是"point-to-point",和 P2P 不是一回事。对等模式的本质思想在于打破传统的客户/服务器模式,让一切网络成员享有自由、平等、互联的功能,不再有客户、服务器之分,任何两个网络结点之间都能共享文件、传递消息。图 1.1.1 反映出从 C/S 到 P2P 的转变,peers 之间的逻辑连接构建在物理连接的基础上。

对等网络(peer-to-peer network,简称 P2P 网络)是分布式系统与计算机网络相结合的产物,是采用对等模式工作的计算机网络。图 1.1.2 描绘了随着分布式系统规模的扩展,分布式计算的模式相应发生的改变。在对等网络中,每个网络结点在行为上是自由的,在功能上是平等的,在连接上是互联的,所有结点分布式地自组织成一个整体网络,因此,它能够极大程度地提高网络效率,充分利用网络带宽,开发每个网络结点的潜力。

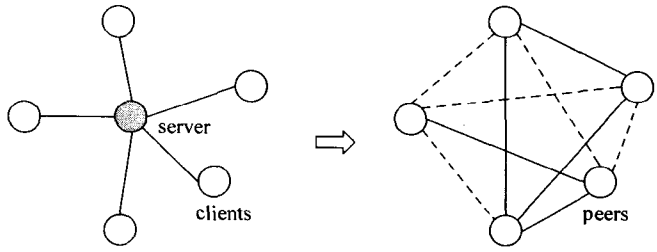


图 1.1.1 从 C/S 到 P2P
（实线表示物理连接，虚线表示逻辑连接）

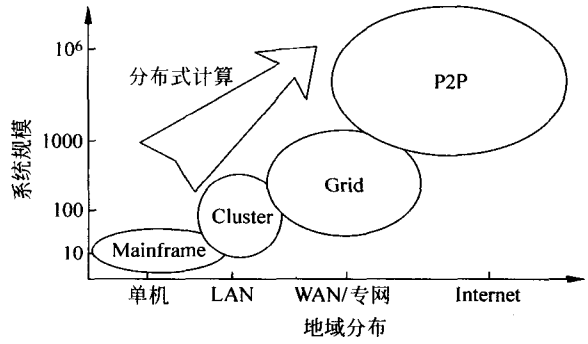


图 1.1.2 分布式计算模式与系统规模的关系
横轴：计算机网络在地域分布上的扩展；纵轴：分布式系统在规模上的膨胀；
斜箭头：适应于不同需求的分布式计算模式；Mainframe：主机计算；
Cluster：机群计算；Grid：网格计算；P2P：对等计算

P2P 的思想起源很早，我们用“Google 学术搜索”（http://scholar.google.com）找到最早提及 P2P 的文献发表于 1956 年，从那以后几乎每年都有与 P2P 相关的文章，但一直未成为热点。任何一种思想、理论的流行通常都需要一个杀手锏（killer application），以一种征服性的力量冲击人类的传统思维。P2P 的杀手锏正是出现于 1999 年的世界上第一个应用性对等网络 Napster，它创造了在半年时间里拥有 5000 万用户的网络奇迹，向整个世界展示了 P2P 优异的性能和巨大的潜力。

学术的脚步常常先于应用踏入某个领域，又往往在应用之后成为热点，P2P 和 Napster 的关系正是如此。在 Napster 之后，是一系列人们耳熟能详的 P2P 网络软件：Gnutella，KaZaA，BitTorrent，eDonkey/eMule，Skype，等等。虽然从 1999 年到现在只有短短几年，但是由于在工作模式上具有的优势和对于现代因特网的适应性，P2P 得以迅速从一个民间小软件发展为计算机网络的一项重要技术，在应用领域和学术界获得了广泛的重视和成功，并占据了当前 Internet 超过一半的带

宽资源，被认为是“改变 Internet 的新一代网络技术”。

对于已经应用或正处于理论研究阶段的各种 P2P 网络，国内外的研究者从多个不同的角度对它们进行了分类，包括从体系结构、出现时间角度和应用领域角度进行的分类，到目前为止尚未出现公认的、明确的分类方法。本书从 P2P 网络设计思想出发，兼顾体系结构和出现时间两个方面的考虑，将 P2P 网络分成三代：

第一代，混合式 P2P 网络，它是 C/S 和 P2P 两种模式的混合；

第二代，无结构 P2P 网络，它以分布、松散的结构来组织网络，故称“无结构”；

第三代，结构化 P2P 网络，它以准确、严格的结构来组织网络，并能高效地定位结点和数据。

对这三代 P2P 网络的讲解，是本书的一大重点，分别对应第 2、3、4 章的内容。读者需要注意的是：我们的分类仅出于本书行文、讲解的考虑，并非 P2P 领域明确的界定。

现代计算机网络均采用层次化的结构，以提供一个便于分析的模型和利于开发的技术接口，它具体地表现为一个网络通信从高层到低层的协议栈。这里面最为著名的是 ISO/OSI(国际标准化组织/开放系统互联)模型和 TCP/IP(传输控制协议/因特网协议)模型，前者细致、正式，后者更为实用。在本书中对于层次化的网络结构描述均采用 TCP/IP 模型，如图 1.1.3 所示。

图 1.1.3 中 TCP/IP 模型共分四层：网络接入层、网络层(IP 协议)、传输层(TCP/UDP 协议)和应用层。在此之前所说的集中式系统、分布式系统、客户/服务器工作模式以及对等计算模式，都是指四层中的最高层——应用层的工作方式，而下面的三层通常采用标准、单一的工作方式，本身并没有集中式与分布式之分，只是为应用层不同的工作方式提供底层的服务支持。

应用层	(C/S， P2P)
传输层	(TCP/UDP协议)
网络层	(IP协议)
网络接入层	(链接协议与物理协议)

图 1.1.3 TCP/IP 协议栈

P2P 网络的核心机制，是在应用层建立逻辑上的覆盖网络(overlay network)，封装下面的三层，让 P2P 网络的研究者和开发者不必关心下面三层是如何工作的，而仅仅去考虑应用层覆盖网络的工作情况，将精力集中于覆盖网络的设计、优化上。虽然如此，在对 P2P 网络做基础的研究和设计时，有时还是要考虑到下面层的工作情况，因为应用层建立的覆盖网与底层实际的物理网的工作情况不可能完全相同，在图 1.1.4 中，覆盖网上的一条逻辑连接 AE 对应物理网上三条物理连接：AC，CD 和 DE。所以从覆盖网看到的行为与底层物理网实际的行为并不一致。P2P 领域的研究者已经对这种不一致性做了大量的工作以尽可能减少两者之间的差异，提高整个网络的效率。简言之：P2P 网络工作于应用层，但兼顾网络底层。读者在阅读本书时应该注意到这一点，这一问题在以后的章节中会有详述。

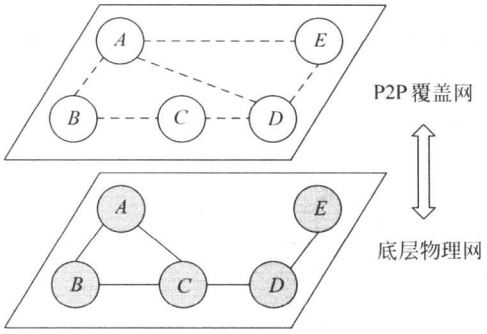


图 1.1.4 P2P 覆盖网和底层物理网的不一致性
（虚线表示逻辑连接，实线表示物理连接）

在 P2P 覆盖网上依靠 DHT(distributed hash table，分布式散列表)通常能准确、快速地路由消息和定位数据对象，这正是 P2P 查询的优势所在。“祸兮，福之所倚；福兮，祸之所伏。”正如计算机领域那条著名的“没有白吃的午餐定理”(no free lunch theorem)(Wolpert and Macready，1997)所述：任何方法对一类问题做得好，必然对另外某类问题做得不好，这就是代价。使用覆盖网和 DHT 的 P2P 网络，为追求性能和效率所付出的代价，是语义模糊查询的困难以及对动态网络环境中错误行为的容忍性下降。针对前者，语义模糊查询至今仍然是 P2P 领域的一个开放型问题，尤其对结构化 P2P 网络更为困难；针对后者，P2P 领域的研究者提出的方案各具特色，总体上分两类：①网络周期性主动更新；②在检测到错误后被动更新。前者机制简单而开销很大，后者开销较小而机制复杂，这就带来了两难问题。

前面描述了 P2P 的核心机制，有了它们，一个 P2P 网络能正常工作，但不见得“好用”，因此人们又提出了很多 P2P 的“增强机制”，以改善网络状况，让它更好地工作。这些增强机制不少是从分布式系统或者计算机网络领域借鉴过来的，如数据复制、缓存、分片等；当然，更多的增强机制来自 P2P 本身，如负载的均衡、异构性的开发、少数结点负担过重导致的“热点”问题、物理网与覆盖网不一致造成的“拓扑意识”问题、保护用户隐私的“匿名”问题、P2P 用户的“声誉”和“信任”问题以及最令人不放心的 P2P“安全”问题。

“纸上得来终觉浅，绝知此事要躬行。”理论是灰色的，生活之树常青。理论并不都现实可行，即使真的实现了，也未必好用，所以做研究注重理论结合实践，强调实践的作用。对 P2P 来说，由于其网络规模巨大，开发实际系统的软硬件开销巨大，因此 P2P 实验更侧重于“模拟”和“仿真”。

“图难于其易，为大于其细。”本章的后续部分，将从“大”而“易”的方面讲述 P2P 的历史、现状、缘由、特点、应用和著名模型；而本书后面的各章，则从“细”而

“难”的方面去具体地讲述三代 P2P 网络(第 2、3、4 章)，列举当前世界范围内的各种 P2P 应用体系和应用软件、介绍一些著名软件的使用(第 5 章)，研究 P2P 技术的核心机制、增强机制、模拟和仿真(第 6、7、8 章)，最后总结 P2P 的现状并展望 P2P 的未来(第 9 章)。我们给每位读者的建议是：把握本书的脉络，理清 P2P 的历史，有选择性地阅读，需要的时候回过头来参考。

1.2 P2P 网络的发展历程

1. 第一阶段：1999—2000 年，民间软件，锋芒初现

1999 年，18 岁的 Shawn Fanning 开发了世界上第一个应用性 P2P 网络软件 Napster，在半年时间里即拥有 5000 万注册用户。Napster 是第一代 P2P 网络——混合式 P2P 体系(hybrid P2P architecture)最杰出的代表，向整个世界传达了 P2P 优秀的思想，展现了 P2P 巨大的潜力。不久以后，Napster 网站因为版权问题被推上法庭，此后一直官司不断，在经历了约两年的法律纠纷之后，2001 年底，流星般的 Napster 最终关闭了。

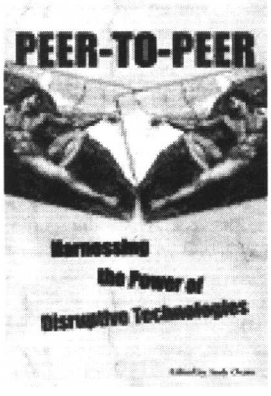
2000 年 3 月，第一个无结构 P2P 网络 Gnutella 诞生于 NullSoft 公司，它是第二代 P2P 网络——无结构 P2P 体系(unstructured P2P architecture)的代表。虽然发布之后不久，Gnutella 就因为其母公司担心法律问题而被关闭，但是 Gnutella 所代表的无结构、纯分布式 P2P 体系的思想却广泛流传开来，而 Gnutella 本身成为一种典型的无结构 P2P 网络协议。

几乎与 Gnutella 同时，以“自由、安全、匿名”著称的无结构 P2P 网络——Freenet 推出，它从一开始就有着很不同的理念：Napster、Gnutella 这类 P2P 系统的主要目的在于交换文件，而 Freenet 的目的是共享 Internet 上的计算机资源，组建一个不受限制、不受审查的信息发布和获取的平台。虽然 Freenet 拥有前卫的思想和高超的技术，但由于其使用的复杂和过于理想的自由主义理念，Freenet 在大多数国家被严格管制。

Gnutella 问世后不久还出现了新的无结构 P2P 应用系统 KaZaA，但与短命的 Gnutella 不同，KaZaA 从开始到现在，其用户群不断扩大，号称拥有超过 300 万的平均在线用户，KaZaA 网站上的统计数据称：截至 2007 年 3 月，KaZaA 软件已被下载超过 3.8 亿次，毫不逊色于当年的 Napster。KaZaA 基于著名的 FastTrack 协议，该协议是与 Gnutella 并列的典型的无结构 P2P 协议，其最大的改进在于引入“超结点”(supernode)，从而第一个有效地开发了 P2P 网络中的结点异构性(即结点之间在能力上的差异)。基于 FastTrack 协议的应用还有 KaZaA 的类似体 KaZaA-Lite，以及 Grokster、iMesh 等。

eDonkey(电驴)差不多和 Gnutella 同时出现(2000 年),它提供文件分块下载,从而每个文件可以从其他多个用户并行下载。eDonkey 网络结构很像 KaZaA,它采用"服务器"(不同于一般意义上的服务器)作为网络的核心部分,"服务器"的作用很像 KaZaA 中的"超结点",每个用户只需连接到特定的"服务器"来共享和获取文件。eDonkey 从一开始就吸引了很多 Internet 用户,到今天仍然非常流行。

2000 年 8 月,著名出版人 O'Reilly 组织了一次意义重大的 P2P 峰会,来帮助人们认识 P2P 的潜力并消除 Napster、Gnutella 造成的"P2P 是盗版技术"的负面影响。参与 P2P 峰会的既有 Napster、Gnutella、Freenet 的开发者,也有那些试图挖掘 P2P 分布计算能力的公司和组织如 Popular Power、SETI @ Home、Distributed. net 等。O'Reilly 认为目前 P2P 的状态类似于"盲人摸象",P2P 技术的领导者们每个人都看到了 P2P 这头"巨象"的一些特征,如果他们能够有机会交流思想,P2P 将会更快地发展。这次峰会主要有三个目的:①诠释 P2P,说明要从中得到什么;②描述 P2P 的作用,P2P 能解决什么样的问题;③形成一个提供给大众的关于 P2P 的信息,消除那些负面影响。在那次峰会后不久,O'Reilly 于 2001 年出版了目前所知最早的关于 P2P 网络的书籍 *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*(见右图)。这本书包含了有关 P2P 的诸多话题:P2P 的起源、Napster、SETI@ Home、Jabber、Gnutella、Freenet、Red Rover、Publius、Free Haven、元数据、缓存管理、信任机制、声誉、安全性等,它主要的不足是内容陈旧、缺乏代表性,因为那时对 P2P 网络的研究实际上还处于萌芽期。



同是 2000 年 8 月,Intel 公司宣布成立 P2P 工作组,正式开展 P2P 的研究。工作组成立以后,积极与应用开发商合作开发 P2P 应用平台。2002 年 Intel 发布了 .NET基础架构之上的 P2P Accelerator Kit(P2P 加速工具包)和 P2P 安全 API 软件包,从而使得利用微软. NET 开发软件的人员能够迅速地建立 P2P 安全 Web 应用。

IBM、HP 等公司也是 Intel 成立的 P2P 工作组中的成员。这两家公司在 2000 年 9 月共同推出了一种开放存储技术,这一存储技术利用了 P2P 技术,可以方便地从用户的硬盘向服务器上复制数据。HP 公司还把 P2P 的立足点放在打印技术上,该公司新推出的网络打印技术可使用户通过 P2P 网络共享打印机。

2. 第二阶段:2001—2003 年,步入正统,群雄逐鹿

如果说 1999 年属于第一代混合式 P2P 网络的辉煌、2000 年属于第二代无结

构 P2P 网络的成功，那么，2001 年则是第三代 P2P 网络——结构化 P2P 体系(structured P2P architecture)展现的舞台。这一年学术界真正开始关注和重视 P2P 网络，IEEE 成立 P2P 专业会议，ACM 在网络通信领域最具影响力的几个会议发表了多篇有关 P2P 的经典论文，P2P 领域最具代表性的经典模型和应用体系被提出，其中有：Chord、CAN、Tapestry、Pastry、CFS、OceanStore、PAST 等；同时，大多数知名的学术团体和技术组织成立或者完善了专门的 P2P 研究组，其中有：MIT 的 Chord/CFS 研究组、UC Berkeley 的 Tapestry/OceanStore 研究组，Microsoft Research 和 Rice University 的 Pastry/PAST 研究组、Stanford Peers 研究组等。

2001 年，Ray Ozzie(著名的 Lotus Notes 软件的开发者)创立了 Groove Networks 公司，开发 Groove Virtual Office(Groove 虚拟办公室)，此软件意在使用 P2P 技术营造一个 Internet 协同工作空间。2005 年 3 月，Groove 公司被软件巨人微软以 1.2 亿美元收购，而 Groove 的创始人 Ray Ozzie 成为微软 CTO(首席技术官)。实际上早在 2001 年，Groove 公司就曾接受过微软高达 5100 万美元的投资，所以 2005 年的收购并不值得奇怪。在微软最新推出的 Office 12 办公软件中，已整合了 Groove 软件：处于测试中的 Microsoft Office 12 共包含 14 个组件，而 Groove 占据其中 3 个。

2002 年，P2P 专业会议 IPTPS 首次召开，会议的规模不大，但影响力不小。SIGCOMM、SPAA、PODC、INFOCOM、ICDCS 等网络通信、分布式系统领域的重要会议继续关注 P2P，甚至有些设置了 P2P 专题讨论会，新的 P2P 模型如 Kademlia、Viceroy 等被提出，它们在理论上很有意义。

2002 年 5 月，由于不满意当时的 eDonkey 2000 客户端软件，并且坚信能做出更出色的类似客户端，Merkur 聚集了一批原本在其他领域有出色发挥的程序员到他的周围，开始了著名的“eMule 工程”。eMule“电骡”这个名称表明了它和 eDonkey 的关系——承继关系，但 eMule 更出色。他们没有想到的是，不久以后 eMule 的流行性甚至远超过它的前驱 eDonkey。

2002 年 10 月，新一代的混合式 P2P 网络 BitTorrent 推出，到 2003 年 BT(BitTorrent 简称)已在世界范围内(尤其是在中国)广泛流行。2006 年底，中国互联网络信息中心(CNNIC)发布的中国互联网统计报告显示：中国 1.11 亿网民中有 27.8% 的人使用过 BT 软件(超过 3000 万人)，其流行程度由此可见一斑。BitTorrent 使用基于文件的分散式服务器，共享同一文件的用户构成一个独立的子网，所以 BitTorrent 既不会因为一台服务器失效而影响整个网络，也不会像 Napster 那样被关闭网站(BT 网站太多且分散在世界各地)；同时，分片优化、阻塞控制等方法使得 BT 能够充分利用网络资源。

自 2003 年开始，P2P 的发展实际上进入一个稳定期。在解决了 P2P 网络最核心的问题后，学术界将重点放在其性能增强、安全问题和实用系统开发上。这里

值得一提的是由 MIT 的 Frans Kaashoek 教授领衔的研究小组，他们联合其他美国一流高校和研究机构进行的 IRIS 项目（Infrastructure for Resilient Internet System，容错的 Internet 系统架构），用 P2P 的方法去研究并建立新一代互联网络结构，得到了 2003 年美国 NSF（自然科学基金）在 IT 领域最大的一项基金资助。另一方面，商业领域更多地在改进过去的 P2P 应用软件，很多混合式、无结构的 P2P 网络将学术界结构化 P2P 体系的思想整合进来，如 BitTorrent 就整合了 Kademlia 的分布式散列表，这种融合体现了 P2P 学术界对商业界的影响，是难能可贵的。

2003 年，在无结构 P2P 方面 Gnutella 协议 0.6 版发布，它比 Gnutella 协议 0.4 版扩展了很多，比如以明确的形式建议“超 Peer”（UltraPeer）的使用。在结构化 P2P 方面，新的 P2P 模型 SkipNet、Koorde 等被提出，SkipNet 是第一个显式提供路由局部性、对象语义局部性的结构化 P2P 模型，Koorde 则在理论上具有很高的价值，证明了一些 P2P 领域悬而未决的结论。

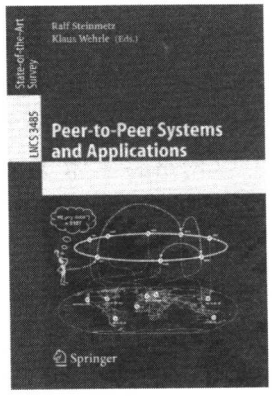
2003 年，商业领域诞生了一颗璀璨的新星——Skype 公司，它是全球第一家 P2P 即时通信公司，采用 P2P 技术为用户提供免费或廉价的语音通话服务，使用端到端的加密技术保证通信的安全可靠性。仅一年时间，Skype 用户就超过了 1300 万，迄今为止注册用户超过 2300 万，同时在线用户数高达 100 万，而且新用户以每天 6 万的速度增长，其迅猛的发展速度再度证明了 P2P 的巨大潜力。

软件领域的巨人微软公司向来不愿错过每一项热点技术。为了使 Windows 操作系统更好地与 P2P 应用协调，Microsoft 对 Windows XP 进行了强化，公布了“Windows XP P2P 软件开发包”，这个编程工具使软件开发商能够更容易地在 Windows XP 上编写 P2P 应用程序。2003 年，Microsoft 并购了 P2P 新公司 XDegrees，声称将把 XDegrees 的技术用于它的存储产品，随后，Microsoft 推出了一款面向年轻人的即时通信软件“Three degrees”，大刀阔斧地挺进 P2P 市场。微软在全球各地的研究院基本上都将 P2P 列入其工作重点并发表了多篇有价值的论文，这里面做得最好的是微软剑桥研究院（Pastry/PAST 系列），微软亚洲研究院的系统研究组在 P2P 方面也很不错。

3. 第三阶段：2004—2006 年，国际国内，风起云涌

2004 年，在 IPDPS 会议上基于 CCC 拓扑的 Cycloid 常数度 P2P 模型被提出，它兼有常数度、超立方体、环形多种属性，可以看成是对此前诸多结构化 P2P 模型的一个总结。从 Cycloid 的提出可以看出，P2P 网络的主要问题、核心机制、整体架构已经形成，P2P 研究者对这些重大问题已经形成共识，下一步要做的工作，应该主要放在更细节、更高效、更实用的方面。另外，如何将各种不同的 P2P 应用系统像 Web 那样整合起来（2005 年发表于 SIGCOMM 会议的 OpenDHT 服务体系

在这方面做了努力），甚至将 P2P 和 Web 整合起来，也是一件非常有意义的大事。



2005 年底，Springer 在其 LNCS（Lecture Notes in Computer Science）系列中出版了一本内容丰富、涵盖面非常广阔的 P2P 专著 *Peer-to-Peer Systems and Applications*。全书共 33 章，分 10 个部分，每章由不同作者编写，涉及 P2P 的概念、历史、结构、应用、自组织、检索、性能、移动环境应用、商业应用与市场等多方面内容，提供了对整个 P2P 领域权威、全面的总结、分析与展望。此外，P2P 的大师 Ion Stoica 特意为上述专著绘制了封面（见右图）并撰写了前言，使该书增加了影响力。

我们再将目光转到国内的 P2P，首先来看学术界的情况。国内一些高校、科研组织从 2003 年前后开始 P2P 的研究开发工作，其中有影响的是：北京大学网络实验室开发的“Maze 网络文件系统”、华中科技大学集群与网格计算实验室开发的“AnySee 视频直播系统”（Maze 和 AnySee 已投入应用，尤其在高校学生中相当流行）、清华大学高性能计算研究所开发的“Granary 广域网分布式存储系统”（处于开发优化进程中）。此外，仅以 2005、2006 年的国家自然科学基金资助项目为例，P2P 相关项目就有数十项，表 1.2.1 是一份不完全的统计。

表 1.2.1　2005、2006 年国家自然科学基金资助的 P2P 相关项目（不完全统计）

年份	批准号	项目名称	依托单位
2005 年	60503045	P2P 网络自主行为模型研究及其应用	上海交通大学
	60503047	基于语义链的对等网络模型及异构数据管理方法研究	中国科学院计算技术研究所
	60573053	P2P 系统的信任管理和匿名问题研究	中国科学院研究生院
	60573096	基于模式的高可扩展性 P2P 数据管理技术的研究	西北工业大学
	60573110	基于结构化对等网络的全文搜索技术	清华大学
	60573120	P2P 网络的关键安全问题研究	华中科技大学
	60573127	基于 P2P 思想的网格计算资源动态管理与任务调度研究	中南大学
	60573129	基于泛洪的非结构化 P2P 系统中分布式拒绝服务攻击防范方法的研究	电子科技大学
	60573131	实用化对等网络技术的研究	南京大学
	60573140	自适应的基于分布式索引缓存的无结构 P2P 快速搜索算法的研究	温州大学

续表

年份	批准号	项目名称	依托单位
2006年	60673071	P2P流媒体点播中的网络波动问题研究	清华大学
	60673156	IPv6环境下基于P2P的DDoS分布式防御研究	湖南大学
	60673166	非结构化对等网络高性能动态查询响应关键技术研究	上海交通大学
	60673167	基于对等模式的网格资源定位关键技术研究	中国人民解放军国防科学技术大学
	60673179	超级非结构化(Superpeer)P2P网络动态层次优化机制研究	浙江工商大学
	60673183	P2P文件共享系统中信誉机制的研究	北京大学
	60673184	基于对等网络技术的安全、高性能流媒体系统的理论研究	清华大学

作者所在的南京大学计算机科学与技术系GPS(Grid、P2P、Sensor)研究组，一直致力于P2P理论的钻研和P2P技术的开发工作。在理论方面，我们一直紧密关注P2P领域的新理论、新技术和发展趋势，重点跟踪SIGCOMM、SPAA、PODC、ICNP、INFOCOM、ICDCS、IPDPS、IPTPS这几大重要国际会议，并在不少国内国际的会议刊物上发表我们的研究成果。在技术开发方面，我们主要进行两个项目的工作：一是2005年国家自然科学基金项目“实用化对等网络技术的研究”，二是2005年江苏省自然科学基金前期预研项目“新型P2P计算技术的基础研究”。这些项目着眼于多项P2P新技术，其研究内容涉及新型P2P拓扑结构、可控散列函数、拓扑意识和一致性问题、容错性保证(崩溃点测量、覆盖网分割问题)、声誉与信任机制、安全检测与恢复等。

国内商业领域最近几年内出现了很多国产P2P应用软件，其中不少具有相当的流行性，这是国内P2P研究中非常可喜的现象。国产P2P文件共享软件如PP点点通、PoCo、百宝等都有很大的用户群，而国产P2P网络电视软件PPLive、TvAnts等在用户规模上也毫不逊色，从中可以看出国内计算机界在P2P方面所做的努力，以及国内计算机用户对P2P的认识与接受程度。虽然如此，目前众多国产P2P软件在技术上还很不完备，性能上也有很大的改善余地，与相对成熟的国外P2P软件相比还有不小的差距。

1.3 为什么需要P2P网络

P2P网络使得网络工作模式从集中式走向分布式，网络应用的核心从服务器走向每一个网络结点，从而使人们在网络上的信息交流被提升到了一个更高的层

次，使人们以更主动深刻的方式参与到网络中去。我们需要 P2P 网络，是因为它新颖的工作模式很好地适应了不断膨胀、不断扩展的 Internet 动态网络环境。P2P 网络的优点可归纳如下。

1. P2P 提高了网络工作效率

在 C/S 模式中，客户只能与服务器交换信息，如果两个客户之间要传送一个文件，通常是首先将文件传到服务器，然后由服务器传给另一个客户（或者说另一个客户从服务器下载文件），这在无形中增加了一个不必要的环节（最典型的如 FTP 文件传送）。如果服务器忙，文件传送将变得十分缓慢，更严重的情况下如果服务器故障，文件将没有办法传送，C/S 模式固有的问题影响了网络工作效率。P2P 模式与传统的 C/S 模式最大的区别在于没有集中式的控制，任意两个结点之间交换信息不需要经过一个固定的服务器，因此上面所说的问题对 P2P 网络来说是先天不存在的。

P2P 网络高效的另一重要原因是它在网络应用层构建了一个有（严格）拓扑结构（topology structure）的覆盖网，并且通常使用基于一致性散列函数（consistant hashing）的分布式散列表（DHT），将网络结点或数据对象高效、均匀地映射到覆盖网中。覆盖网与分布式散列表的结合，使得 P2P 网络具有很高的路由效率：比如对结构化 P2P 网络，任意两个结点间定位所要经过的覆盖网路由跳数在 $\log N$ 左右，N 为网络结点总数，这是一个非常理想的结果。虽然如此，正如 1.1 节所述，由于覆盖网与因特网的不一致性，覆盖网上一跳可能对应因特网上多跳，实际的 IP 路由跳数可能高于 $\log N$，许多研究者提出不同方案解决这一问题，努力缩小两者之间的差异，将实际的 IP 路由跳数控制在 $\log N$ 的常数倍范围之内，这样的路由效率仍然是很不错的。

2. P2P 充分利用了网络带宽

有个小故事说，硅谷有个做网络通信的腰缠万贯的商人，临死前竖着一根手指，他的子女不解就问他还有什么遗愿，他放下手指说了句“带宽”就断气了。对于 Internet 而言，带宽确实是最宝贵的资源；而在有限带宽的情况下，如何充分地利用带宽则更为关键。很多人都有这样的体会：明明网络带宽很高，下载某个文件的速度却低得出奇，就好像拿一个大木桶去等滴水的龙头，不是这头的桶不够大，而是那头存在瓶颈。

P2P 网络中没有 C/S 模式下的服务器，所以不存在 C/S 模式下服务器造成的“效率瓶颈”（efficiency bottleneck）——既没有不必要的中间环节，也不会因为服务器忙而不得不等待，更不会出现由于服务器“单点失效”（single point of failure）

导致整个网络通信中断的情形。在P2P网络中，任意两个对等结点之间平等地互联，在交换信息的过程中不受其他结点的控制与影响，数据传输速度通常只取决于网络带宽，因此它对网络带宽的利用是充分的。以最流行的P2P应用BT为例，当使用BT下载多个文件时，下载速度常常能达到或者接近网络带宽，而反过来使用HTTP和FTP则很少有这样的效果。

3. P2P开发了每个网络结点的潜力

一个网络结点对于网络潜在的贡献，是它可以提供的计算能力和存储容量。传统的C/S模式形成了以服务器为核心的网络——数据集中存储在服务器上，绝大多数的计算任务由服务器完成，客户对网络计算能力和数据存储的贡献微乎其微。P2P的出现将网络的核心从服务器转变为每一个网络结点——数据分散地存储在所有结点上，计算任务由各个结点分布、协同地完成，每个结点都是网络的主体和重要成员。

世界上绝大多数计算机都不会像专用服务器那样永久地连在网上，这些临时性的动态结点称为“网络边缘结点”(network edge node)，而它们所提供的计算能力(空闲处理器周期)和存储容量(剩余存储空间)称为“网络边缘资源”(network edge resource)，实际上这些临时性的网络边缘资源才是网络资源的真正主体。P2P网络的主体是网络边缘结点，它是第一个真正而又充分地利用网络边缘资源的计算机网络，P2P网络相对于传统网络的优势和对现代因特网卓越的适应性正是来源于此。所以说，P2P开发了每个网络结点的潜力，使得因特网的存储模式从“内容位于中心”转变为“内容位于边缘”，计算模式从“服务器集中计算”转变为“分布式协同计算”。

4. P2P网络具有非常高的可扩展性

计算机网络的可扩展性(scalability)通常用下面几个因素来衡量：当网络结点总数增加时，①结点负载如何改变；②为适应规模扩大而需要增加的额外设备的数量；③任意两个网络结点通信效率如何改变，尤其是路由效率。

C/S模式的缺陷在于它的难扩展性，当网络结点数目大量增加时，服务器的负载也随之线性地增加，虽然服务器通常都是高性能计算机，但是性能再高的超级计算机也没有办法应付网络规模的不断膨胀。当原先的服务器不堪重负时，必定要购买新的服务器来替代它或者分担它的负载，然而高性能服务器的价格是昂贵的，额外设备的开销会非常大。同时，由于连入同一服务器的客户数目增多，通信效率平均而言一定会下降，并且也增加了服务器过忙和故障的概率。

P2P模式具有非常高的可扩展性。首先，当网络结点数目大量增加时，随之增

加的通信开销被更多的结点分担，所以每个结点承担的负载并不会增加太多。其次，在网络规模扩大时 P2P 网络不需要增加额外设备。再次，P2P 网络路由跳数（针对结构化网络）的典型值为 $\log N$，所以随着 N 的增加路由跳数会增多，但是增量非常少，通信效率仍然保持在较高的水平。

5. P2P 网络具有良好的容错性

P2P 网络所构建的覆盖网和分布式散列表极大地提高了网络的工作效率和可扩展性，但是，覆盖网拓扑结构越严格，容错性（fault resilience）通常就越差，因为严格的拓扑结构对其成员结点不正常行为的容忍性相对较低。针对这一缺点，研究者提出了各种增强机制，最典型的是冗余方法（redundancy）和周期性检测（periodical detection）：前者通过保存适当的冗余信息，提供有效的替代，以空间换取容错；后者通过每个结点周期性地检测，来及时纠正错误，以时间换取容错。

为了实现高效的路由，P2P 网络中每个结点通常会保存一个路由表和其他辅助性的信息，记录它的网络邻居等，这些信息称作“结点状态”（node state），它在结点加入网络时被初始化。由于网络环境的动态性，网络结点不断地加入和离开，结点状态会变得陈旧，与实际网络不一致，这种不一致影响网络的工作效率。人们设计了很多方法来及时更新结点状态，这一机制称为结点的“自适应”（adaptability），它是保证网络容错性的基础。周期性探测是最典型的自适应操作，直接的办法是采用短周期频繁探测使得结点状态始终保持在最新，但是这样做开销很大；改进的办法是采用长周期松散探测，虽然不能保证结点状态最新和网络最高效，但是能保证结点在较新的状态下仍然能正确且比较高效地工作。除此之外，还有一些结点自适应的新方法，如 Kademlia 协议采用的消息“捎带确认”（piggybacking），将结点状态信息捎带在每条路由消息中发布，接收者由此确认自己的状态信息是否已过时。

1.4 P2P 网络的特点

P2P 网络的特点整体上可以概括为三个词：自由、平等、互联。

（1）自由　在 P2P 网络中，对等结点做什么事情、采取什么样的行为、与其他结点交换哪些信息，由其本身自由决定，而不受限于其他网络结点。另一方面，由于采用分布式散列表的 P2P 网络特有的匿名性，保护了发布者的信息，使得用户能够更加自由、没有后顾之忧地参与到网络中来。

（2）平等　平等是 P2P 网络最重要的特性，是“对等（计算）”名字的由来。平等意味着在一个系统中，虽然能力不尽相同（即结点间的“异构性”（heterogeneity）），但所有成员在功能、地位上都是平等的，没有谁拥有特权，没有谁能控制或限制其他

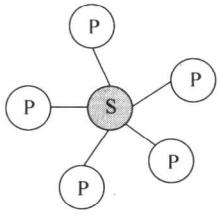
结点。就网络组织方式而言，平等指的是打破传统的客户/服务器模式，取消服务器这一特权结点的存在，让所有网络成员之间平等地交流信息。平等性是 P2P 网络的工作基础，P2P 网络对网络带宽的高效利用、对网络结点潜力的充分开发以及可扩展性等，都是基于平等性的。

（3）互联　互联的本质原因是 P2P 在应用层构建了覆盖网。P2P 网络中任意两个对等结点间都可以建立连接，这是覆盖网上的一条逻辑连接，它通常对应物理网上的一条 IP 路径（或者说是传输层的一个 TCP 连接）。在 C/S 模式下客户只能和服务器建立一条 Client-to-Server 的连接，而在 P2P 模式下任何两个对等结点间都可以建立一条 Peer-to-Peer 的连接。互联性源于平等性，它也是 P2P 网络高效率、高可扩展性的重要基础。

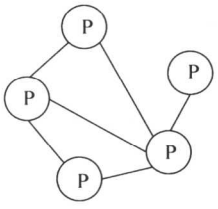
更具体地说，P2P 网络区别于其他系统的本质特点如下。

1. 网络拓扑结构严格

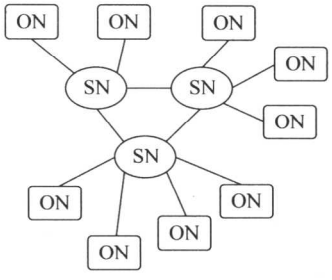
P2P 网络在网络应用层构建了一个有（严格）拓扑结构的覆盖网，覆盖网拓扑结构对于一个 P2P 网络具有基础性的意义，系统的其他许多机制如分布式散列表、路由、负载均衡、容错与自适应、自组织都以它为基础。具体来说 P2P 网络通常采用图 1.4.1 所示的几种拓扑结构。



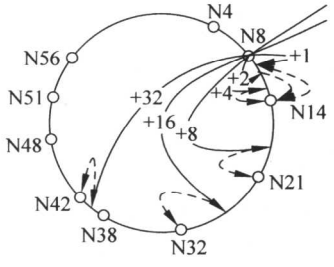
(1) 星形结构—— Napster、BitTorrent



(2) 随机图结构—— Gnutella、Freenet

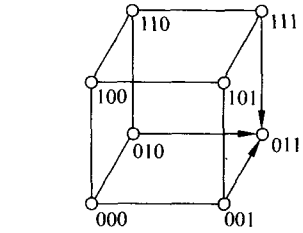


(3) 双层结构—— KaZaA 、 eDonkey/eMule

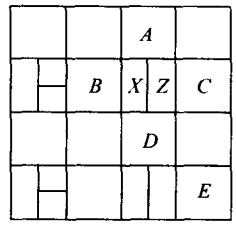


(4) 带弦环结构—— Chord 、 Kademlia
(图片来自 [Lua et al.,2004])

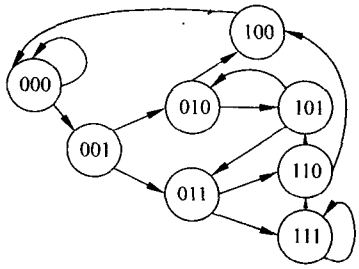
图 1.4.1　P2P 网络通常采用的拓扑结构图



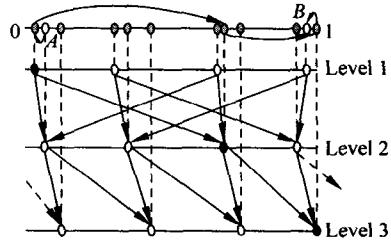
(5) 超立方体结构——Tapestry、Pastry
（实际上是变形的网格结构）



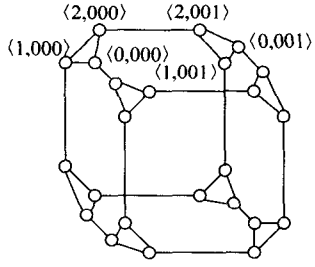
(6) 多维空间结构——CAN
（图片来自 [Lua et al.,2004]）



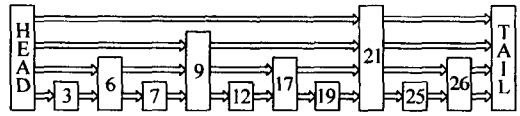
(7) de Brujin图结构——Koorde



(8) 蝴蝶结构——Viceroy
（图片来自 [Lua et al.,2003]）



(9) CCC结构——Cycloid



(10) 跳表结构——SkipNet（图片来自 [Harvey et al.,2003]）

图 1.4.1（续）

2. 结点和数据对象位置确定

分布式散列表(DHT)是P2P网络的核心设施。如图1.4.2所示，它通常基于一致性散列函数，提供对于任何一个结点、数据对象在覆盖网中的位置映射。这一点在结构化P2P网络中尤其重要，因为它保证了能够准确地定位到某个结点或者数据对象。具体地说，如果分布式散列表采用一致性散列函数 $H()$，对于某个网络结点(IP,Port,…)，该结点在覆盖网上将有唯一对应的“结点标识”nodeID=H(IP,Port,…)，IP为结点IP地址，Port为端口号，…表示其他属性；对于某个数据对象(Key,…)，它在覆盖网上也有唯一的“对象标识”objectID=H(Key,…)，Key为对象关键码，…表示其他属性。对于结点而言，nodeID确定了它的覆盖网

位置，对于数据对象而言，objectID 确定了它的索引信息在覆盖网上的存放位置。

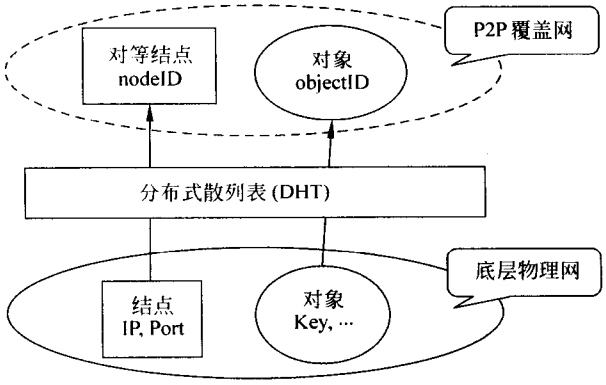


图 1.4.2 DHT 在 P2P 系统中的位置

3. 高效路由

基于 P2P 覆盖网与分布式散列表，P2P 网络通常有适合自己的路由算法，以保证高效路由。任意两个结点间定位所需的覆盖网路由跳数典型地为 TTL(无结构网络)或 $\log N$(结构化网络)，TTL(time to live)为跳数限制，N 为网络结点总数。因为覆盖网与物理网的不一致性，实际 IP 路由跳数会高于覆盖网路由跳数，但仍可以控制在 $\log N$ 的常数倍范围之内。具体地说，P2P 系统的路由方法大致有：

(1) 服务器路由——客户直接发送消息给服务器，服务器返回所需信息。

典型的系统有 Napster、BitTorrent；路由跳数为 $O(1)$。$O()$为渐进增长率符号，若函数 $f \in O(g)$，表示函数 f 渐进增长的速率不超过函数 g。数学上更严格地表述为：如果 $f \in O(g)$，那么 $\lim\limits_{N \to \infty} \dfrac{f(N)}{g(N)} = c$，$c$ 为某个有限常数。

(2) 无结构路由——结点以洪泛法或者类似的方法发送消息给自己的每个邻居，邻居收到消息后也这样做下去，直到定位成功，通常会有跳数限制 TTL 以控制路由范围。典型的系统有 Gnutella、Freenet；路由跳数为 $O(\mathrm{TTL})$。

(3) 双层路由——通常为双层结构的 P2P 网络所使用，“普通结点”直接发消息给“超结点”，“超结点”之间使用无结构路由。典型的系统有 KaZaA、eDonkey/eMule；路由跳数为 $O(1)+O(\mathrm{TTL})$。

(4) 数值邻近路由——这里的“数值”通常指结点 ID 值，路由过程中每一步，当前结点都在自己的路由表中选择与目的 ID 最邻近的结点作为下一跳。典型的系统有 Chord、Kademlia、SkipNet(NameID 路由)；路由跳数为 $O(\log N)$。

(5) 逐位匹配路由——逐位匹配路由基于层次化的路由表，每一步通常都能与目的 ID 多匹配至少一位。典型的系统有 Tapestry(后缀匹配)、Pastry(前缀匹配)、Koorde(de Bruijn 路由)、SkipNet(NumericID 路由)；路由跳数为 $O(\log_B N)$，B 为 nodeID 的基数(或称进制)。

(6) 位置邻近路由——每个结点的路由表记录自己在多维空间中的邻居，每次选择离目的结点最近的邻居作为下一跳。典型的系统有 CAN；路由跳数为 $O(d\sqrt[d]{N})$，d 为空间维数。

(7) 层次路由——不少 P2P 网络将结点组织到多个层次上，路由过程常常是先从底层爬到高层，再从高层爬到底层。典型的系统有 Viceroy、SkipNet(NumericID 路由)；路由跳数为 $O(\log N)$。

(8) 混合式路由——大多数结构化 P2P 网络的路由方式都不是单一的，如 Chord、Pastry、SkipNet、Viceroy、Koorde、Cycloid 中都使用了环形路由作为基础，但又结合了各自独特的路由方式。

4. 负载均衡

P2P 网络使用分布式散列表将结点、数据对象分布到覆盖网上，由于它通常使用一致性散列函数，所以这种分布是均衡的：所有结点大致均匀地分布在整个覆盖网中，所有数据对象的索引大致均匀地分布在所有结点中，即使有新结点加入、旧结点离开、新对象发布、旧对象删除，一致性散列函数都能保证高效的动态调整和调整后网络仍然保持很好的负载均衡。但上面所说的“均衡”只是一种“平均”，它不考虑结点之间能力上的不同，所以往往出现高能力结点空闲、低能力结点过于忙碌的情形；真正好的“负载均衡”(load balance)，应该是结点根据自己的能力“各尽所能”，实际上这体现了对结点“异构性”的开发。负载均衡是分布式系统努力追求的系统属性，它对于 P2P 系统的高效率、可扩展性、动态自适应具有重要意义。

5. 容错与动态自适应

在 P2P 网络产生之初，容错性一直是一个难题。虽然严格的拓扑结构和分布式散列表映射提高了系统的效率和可扩展性，却使得系统对其成员不正常行为的容错性下降了。随后，研究者提出各种增强容错性的机制，最典型的如冗余方法和周期性检测。到目前为止，大多数 P2P 模型和应用在不同程度上采用了这些方法，实现了良好的系统容错性。

P2P 网络是动态变化的，不断地有新结点加入、旧结点离开、新对象发布、旧对象删除，当这些发生以后，P2P 网络必须有很好的自适应性，做高效的调整，以保持网络的拓扑结构、位置映射、负载均衡和路由信息的更新。如上节所述，自适应最重要的是更新结点状态，自适应的方法有周期性探测、按需检测、捎带确认等。

6. 行为的自由性与匿名性

在C/S模式下，客户能做的事情，完全是服务器所提供的，客户不可能采取服务器不允许的行为，也不可能与服务器交换它不支持的信息。相反，P2P网络是一个自由、平等的网络，两个对等结点之间做什么事情、采取什么样的行为、交换哪些信息，完全由双方自由决定。比如现在最流行的BT网络，用户提供哪些文件给别人下载、发出对哪些文件的请求，甚至是上传文件给别人的速度、想与谁通信不想与谁通信，都是自由决定的。

另一方面，P2P网络中的用户是匿名的，因为分布式散列表采用安全散列函数将用户信息、数据对象信息映射到了一个表面上看起来没有任何意义的数值标识(identifier，ID)，这个ID唯一地代表用户和数据对象。由于安全散列函数的单向性与抗冲突性，不可能从此ID破解出它所代表的信息，匿名性(anonymity)正是基于这个原理实现的。

对于网络通信的自由性、匿名性，计算机领域乃至整个人类社会都存在着长期的争论，从来就没有得出过确定的结果。自由给人最大的活动空间，但是也给心存恶意的人提供了破坏系统秩序的基础；匿名使人可以做任何想做的事而不必关心后果，同时保护了发布者的重要信息和隐私，但是也会带来太多不负责任的行为。对于这种争论，在哲学上我们没有确定的态度；但是在技术上倾向于认为：尽管自由性和匿名性目前带来了很多困扰，然而，自由性、匿名性仍然是计算机网络和分布式系统发展的趋势，其中一个关键点是对“自由”、“匿名”本身含义的理解。

1.5 P2P网络的各种应用

在大多数传统的分布式系统和计算机网络的应用领域，都能采用P2P作为替代或者改进。此外，P2P也有很多独特的、不可替代的应用方式。P2P让每个互联网用户以更深刻、更广泛的方式参与进来，正如Internet2之父Doug Van Houweling所说：“下一代互联网用户将真正参与到网络中来，每个人都能为网络的资源和功能扩展作出自己的贡献。”我们相信这里的“下一代互联网”将会是P2P的。本书在第5章中还将按照下面的分类来详细讲述P2P网络的各种应用。

1. 文件共享

可以说文件共享的需求直接引发了P2P技术的产生与开发热潮。在传统的C/S模式下，两个网络用户要实现文件交换需要服务器的大力参与，通过将文件上传到某个特定的网站，用户再到那个网站搜索需要的文件然后下载，最典型的如FTP，这种方式的不便之处不言而喻。电子邮件方便了个人间的文件传递，却无法

解决大范围的交换问题，而且 E-mail 体系本质上还是依靠 Web 服务器的。Napster 正是在此情况下横空出世，它抓住人们对音乐 MP3 文件的需求，将文件共享的实际传输工作完全交给用户，而服务器只提供文件索引。Napster 对网络分工做出的小小的改变，直接引发了网络的 P2P 技术革命。在 Napster 之后，出现了多种基于不同结构的文件共享 P2P 网络，几乎每一种都得到了广泛的应用。文件共享是 P2P 应用的重点和主流，表 1.5.1 简单介绍了最具代表性的 P2P 文件共享系统。

表 1.5.1 最具代表性的 P2P 文件共享系统

Napster	世界上第一个应用性 P2P 网络，也是混合式 P2P 体系最有影响力的代表。因为版权的法律纠纷，原来的 Napster 已经不存在，下面是现在的 Napster 商业网站：[www. napster. com]
BitTorrent	简称 BT，是一个借助分散式服务器提供共享文件索引的混合式 P2P 网络，文件分片下载，并限定了用户在下载的同时必须上传。国外：[www. bittorrent. com]，国内：[bt. btchina. net]
Gnutella	第一个无结构 P2P 网络，网络中只有一种结点——对等结点，不再有服务器存在，对等结点分布式地松散组织，文件查询采用受限洪泛式的方法。商业网站：[www. gnutella. com]，技术网站：[rfc-gnutella. sourceforge. net]
FastTrack/ KaZaA	最大的特点是将网络结点组织成两类——超结点和普通结点，以此来开发结点异构性。FastTrack 系列中最著名的是 KaZaA。KaZaA 网站：[www. kazaa. com]
eDonkey/ eMule	eDonkey(电驴)将网络结点以类似 KaZaA 的方式组织成两层——服务器层和客户层，但 eDonkey 将文件分块从而提供多源下载机制。eDonkey 网站：[www. edonkey2000. com] eMule(电骡)这个名称表明了它和 eDonkey 的关系——后继但更出色，其流行性甚至远超过它的前驱 eDonkey。eMule 中文网站：[www. emule-project. net/home/perl/general. cgi]
Freenet	Freenet 从一开始就有着很不同的理念，其目的是共享 Internet 计算机资源组建一个自由、安全、匿名的信息发布和获取的平台，它是一个真正具有高度匿名性的 P2P 网络
Maze	北京大学网络实验室在 2003 年夏天开发了 Maze 这样一个混合式的 P2P 应用系统，它依托于北大天网，最初动机是要解决 FTP 服务器无法有效下载的缺陷。Maze 在教育网内应用广泛，是国内 P2P 网络的一个成功先驱。Maze 网站：[maze. pku. edu. cn]

2. 多媒体传输

自多媒体技术产生以来，针对多媒体文件(音频/视频/…)人们一直在寻找一个合适的网络传输载体，这源于多媒体文件特殊的要求：传输量大且要求传输速

率稳定。最初的多媒体传输采用C/S方式(FTP或HTTP最常见),所有发送带宽由服务器提供。很明显,当用户数急剧增加时,服务器不可能承受如此大的负担,所以Web网站采取的对策,要么是限制用户数目,要么是减少对每个用户的发送流量,但无论哪种对策都牺牲了性能。

P2P技术适应了多媒体传输对网络带宽的巨大需求,因为所需要的大量带宽被所有共享多媒体文件的用户分担,并且用户之间互相提供数据流量,所以当用户数急剧增加时通常发生的情况是:用户越多,传输越好!另一方面,一个或多个用户退出、掉线都不会对其他用户的正常传输产生明显或本质的影响。P2P多媒体传输最突出地表现在网络语音传输和"网络电视"(相当于网络视频传输),现在国内也开发了不少P2P网络电视软件,其中一些相当流行。表1.5.2列出了几种代表性的P2P多媒体传输软件。

表1.5.2 代表性的P2P多媒体传输软件

Skype	Skype是一个优秀的P2P网络语音传输工具,既提供高清晰的语音对话,还可以用来拨打国内国际电话。除此之外,Skype也提供网络聊天、文件传输等功能。Skype中文官方网站:[skype.tom.com]
PeerCast	PeerCast是一个不错的P2P广播软件,它帮助用户寻找各种格式的音频/视频流媒体资源,用户既可以播放一个频道,也可以创建一个频道。PeerCast网站:[www.peercast.org]
AnySee	AnySee是由华中科技大学集群与网格计算实验室P2P小组于2004年夏天开发的一个视频直播软件,使用P2P技术(应用层组播)解决教育网内网络电视服务器难以服务众多用户的问题,使更多的用户可以观看和利用网络电视频道。AnySee网站:[www.anysee.net]
Mercora	Mercora是一个非常好的"P2P电台",既能收听,又能广播。Mercora很简单,但很好用。Mercora网站:[www.mercora.com]
国内P2P网络电视软件	代表性的软件有PPLive、TvAnts、CCIPTV、CoolStreaming、QQ直播等,在第5章中将详细介绍它们的特点和使用方法

3. 实时通信

实时通信软件大概是每一个网络用户每天接触最多的了,比如国内最著名的QQ、PoPo,国外最著名的MSN Messenger、Skype、ICQ、Google Talk等。最初的实时通信系统都采用客户/服务器方式,假设用户A要发送消息给B,它首先发消息给服务器S,再由S发给B,虽然这样做多了一个不必要的中间环节,但对于只限于文本的低数据率传输仍然是合适的。然而今天,实时通信软件的功能早就不限于文字传送,还包括声音、视频、在线游戏等,它们对传输率、时延的高要求,注定了C/S方式被用户间直接建立连接的P2P方式所取代。最常见的P2P实时通信软件如表1.5.3所示,通常它们都支持C/S与P2P两种工作模式。

表 1.5.3 常见的 P2P 实时通信软件

QQ	深圳腾讯公司开发的 Internet 实时通信软件，目前支持在线聊天、视频电话、点对点断点续传文件、共享文件、网络硬盘、QQ 邮箱等多种功能，并可与移动通信终端设备相连。（对于中国的因特网用户而言，上面对 QQ 的解释或许多此一举）可以在下面的网站下载 QQ 客户端软件：[www. qq. com]
PoPo	由北京网易公司开发的一款实时通信软件，集实时聊天、手机短信、在线娱乐等功能于一体，还拥有许多特色功能如自建聊天室、网络文件共享、穿透防火墙的超大文件传输、视频聊天、语音聊天等。可以到著名的“网易”门户网站下载 PoPo 客户端：[popo. 163. com]
MSN Messenger	微软公司推出的实时消息软件，在世界范围内拥有巨大的用户群，现在它还支持网络搜索功能。使用 MSN Messenger 可以与他人进行文字聊天、语音对话、视频会议等即时交流。由于 MSN Messenger 的流行，它常被简称为 MSN。可以在下面的链接了解并下载 MSN：[messenger. msn. com]
ICQ	ICQ 是英文“I Seek You”或者“I See You”的连写，它是 QQ 的前驱，也可以看成是世界版的 QQ，功能与 QQ 差不多。ICQ 网站：[www. icq. com]
Jabber	由开放源码组织开发的实时消息传输平台，它基于 XML 语言，其目的在于建立一种让所有实时通信系统之间能够互操作的开放式协议。Jabber 网站：[www. jabber. org]
Google Talk	Google 公司基于 Jabber 的 XMPP 协议开发的一个用来进行语音呼叫和发送实时消息的软件，它最大的特点是简洁、易用（秉承 Google 一贯的风格）。可以在下面的链接了解到 Google Talk 并下载软件：[www. google. com/support/talk]

4. 协同工作

公司机构的日益分散，使得为员工和客户提供轻松、方便的协作工具变得更加重要；而网络的出现，则使协同工作真正成为可能。以传统的 Web 方式实现协同工作，给服务器带来极大的负担，造成昂贵的成本支出；P2P 技术的出现，使得互联网上任意两台计算机都可以建立实时的联系，从而建立了一个安全、共享的虚拟空间，供人们进行各种各样的活动，这些活动可以是同时进行，也可以交互进行。P2P 技术可以帮助企业与其客户以及合作伙伴之间建立起一种安全的网上工作联系方式，因此基于 P2P 技术的协同工作受到了极大的重视，最典型的 P2P 协同工作应用如表 1.5.4 所示。

表 1.5.4 典型的 P2P 协同工作应用

Groove	Groove 由 Ray Ozzie 在 2001 年创办，因开发著名的协同工作软件“Groove 虚拟办公室”而出名。Groove 系统以 P2P 的方式提供实时的文本消息、语音、视频传输，而提供这些功能的目的在于支持互联网上的协同工作。2005 年 3 月 Groove 被微软公司收购并成为新版 Office 的组件。Groove 网站：[www. groove. net]

5. 分布式数据存取

P2P的分布式数据存取系统本身包含文件共享的功能，但其目的与文件共享不同，它不像文件共享系统那样将数据传输率看成最重要的属性，而是以数据的可用性、持久性、安全性为目标，并且通常致力于广阔的领域和海量的数据。鉴于不同的目标，分布式数据存取系统采用的数据存取方法也不同，通常每个数据对象都带有自己的认证、鉴别信息，大多数系统中用户的存取都遵循严格的规则和权限来进行，为确保数据的可用性和持久性，往往采用分片、复制、缓存等方法。典型的P2P分布式数据存取系统如表1.5.5所示。

表1.5.5 典型的P2P分布式数据存取系统

CFS	CFS(cooperative file system，协同文件系统)是一个以Chord作为其DHT(分布式散列表)的P2P数据存取系统，不过它比Chord多了不少新的机制，比如文件分块。通过下面的链接可以了解到CFS：[pdos. csail. mit. edu/chord]
PAST	PAST是一个广域的P2P归档存储系统，以Pastry作为底层DHT，目的是提供Internet上安全、高可用、持久性的数据存取服务。PAST网站：[research. microsoft. com/～antr/PAST/default. htm]
OceanStore	OceanStore是一个以Tapestry作为底层DHT的分布式数据存取系统，其目标是提供全球范围内广域、持久性的数据存取服务，它使用了多种复杂机制提高系统性能，如层次化ID、数据分片冗余存储、一致性更新和内省优化等。OceanStore网站：[www. oceanstore. org]
Granary	Granary(谷仓)是清华大学高性能计算研究所开发的广域存储服务系统，它以对象格式存储数据，既可以基于Grid环境开发，也可以基于P2P环境开发。Granary设计了专门的结点信息收集算法以及结构化覆盖网路由协议。Granary系统项目主页：[hpc. cs. tsinghua. edu. cn/granary/granary. html]

6. 分布式计算

分布式计算将巨大的计算任务分解，交给许多台计算机分别执行，然后再将它们计算的结果进行归纳和整合，从而开发了每个网络结点的潜力，利用了它们空闲的CPU计算能力——重要的网络边缘资源之一。通过众多普通计算机来完成超级计算机的功能，一直是科学家梦寐以求的事情，采用P2P技术的对等计算，把网络中的众多计算机暂时不用的计算能力连结起来，使用积累的能力执行超级计算机的任务。任何需要大量数据处理的行业都可从对等计算中获利，如天气预报、动画制作、基因组研究等。有了对等计算之后，很多领域可以不再需要昂贵的超级计算机。P2P网络在分布式计算方面典型的应用如表1.5.6所示。

表 1.5.6 P2P 网络在分布式计算方面典型的应用

GPU	GPU(Gnutella 全球处理单元)的基本思想是在 P2P 网络(Gnutella 协议网)上共享 CPU 计算能力。相比过去的分布式计算系统,GPU 的计算任务分配发生在对等结点之间,而不是由一个服务器集中分配。GPU 主页:[gpu.sourceforge.net]
SETI@Home	由美国著名的计算机高校 UC Berkeley 建立的一项旨在利用连入 Internet 的成千上万台计算机的闲置计算能力搜索外星文明的实验性分布式计算系统。每个参加者下载客户端软件安装,此软件以屏幕保护程序的方式运行,对来自阿莱伯克射电望远镜采集的信号工作单元进行计算处理,再将结果返回给 UC Berkeley。 国外网站:[setiathome.ssl.berkeley.edu], 国内网站:[www.equn.com/seticn]
Entropia	通过使用其成员计算机的空闲处理器时间,Entropia 向客户提供"透明的、动态的、从一个到千万个处理器的可扩展性;包括实时资源类型和位置的重配置、处理器容错性和安全的网络数据通信。"Entropia 与 SETI@Home 在功能上很像,但它是赢利性的
Distributed.net	"这个创立于 1997 年的组织已经壮大到了全世界成千上万个用户,分布式计算的力量达到了相当于 16 万台奔腾 266MHz 的计算机不间断运行的水平。"用户只需到下面的网站下载一个客户端程序就可以参与到其中:[www.distributed.net]

7. P2P 搜索引擎

P2P 技术的另一个优势是开发出强大的搜索引擎。前面介绍过的 P2P 软件尤其是 P2P 文件共享软件大多支持 P2P 方式的专用搜索引擎(如 Gnutella、KaZaA、eMule 等),但这里要说的"P2P 搜索引擎"指的是能像 Google、百度、雅虎那样包罗万象、基于 Web 的通用搜索引擎。P2P 技术使用户能够深度搜索文档,而且这种搜索无需通过 Web 服务器,也不受信息文档格式和宿主设备的限制,可达到传统目录式搜索引擎(只能搜索到 20%～30%的网络资源)无可比拟的深度(理论上将包括网络上所有开放的信息资源)。可以说,P2P 为互联网的信息搜索提供了全新的解决之道,被很多人认为可能成为第三代搜索引擎的开发技术。不过目前的 P2P 搜索引擎离实际应用还有一些差距,很多都停留在理论阶段。已经出现的 P2P 搜索引擎如表 1.5.7 所示。

表 1.5.7 P2P 搜索引擎

Pandango	美国的新兴搜索引擎设计公司 i5 Digital 在 2002 年已正式推出其依据 P2P 理念开发的商业性搜索引擎 Pandango,不过并没有进入主流的搜索引擎阵容

8. 其他应用介绍

在以后的章节中，我们还会介绍更多 P2P 的应用，包括应用层多播、Web 缓存、事件发布、无线应用等。表 1.5.8 只是其中的一部分。

表 1.5.8 P2P 的其他应用

TinyP2P	TinyP2P 毫无疑问是世界上最小的 P2P 软件，它是用 15 行 Python 代码编写的！TinyP2P 创建它自己的私有和密码保护网络，不过实际上应该没人会用它。TinyP2P 的意义只在于告诉人们要写一个 P2P 软件并不困难。可以到下面的链接了解 TinyP2P：[www. freedom-to-tinker. com/tinyp2p. html]
Hamachi	Hamachi 为多台计算机提供一个安全的专有 P2P 网络。它可以连接任何两台联入 Internet 的计算机，连接是直接的并且可以绕过防火墙和 NAT(网络地址转换)。网站：[www. hamachi. cc]
迅雷	迅雷是一款基于 P2P 技术的多源下载软件。号称“宽带时期的下载工具”，迅雷针对宽带用户做了特别的优化，能够充分利用宽带上网的特点高速下载文件。实际中迅雷主要被用作 Web 插件集成到 Web 浏览器中。迅雷网站：[www. xunlei. com]
P2Pbazaar	顾名思义，“P2P 集市”提供了一个 P2P 方式的电子交易市场，在这里你可以浏览、搜索物品，而交易通过互发 E-mail 进行。“P2P 集市”目前还不支持信用卡，不过最大的问题是没多少人用它。网站：[www. p2pbazaar. com]
JXTA	Sun 公司 JXTA 项目的目的在于提供一个开放、通用、互操作的 P2P 开发平台，JXTA 的核心处使用 XML，这是它独立于语言和操作系统的重要原因。JXTA 封装了 P2P 网络底层，对用户而言只要使用其应用接口就可以进行 P2P 编程。JXTA 网站：[www. jxta. org/]
Bayeux	UC Berkeley 开发的 P2P 多播应用，基于 Tapestry 覆盖网提供高效、容错的应用层多播。可以在下面的网页找到 Bayeux 的信息：[www. cs. ucsb. edu/~ravenben/tapestry/html/bayeux. html]
SCRIBE	Microsoft Research 开发的通用、可扩展的组通信和事件发布系统，提供应用层多播和任播，它基于 Pastry 覆盖网。主页：[research. microsoft. com/~antr/SCRIBE/default. htm]
SQUIRREL	Microsoft Research 开发的分布式协同 Web 缓存，使得用户 Web 浏览器之间能共享缓存，它也是基于 Pastry 覆盖网的。主页：[research. microsoft. com/~antr/SQUIRREL/default. htm]

第2章 第一代P2P网络：混合式P2P体系

这一章讲述代表性的混合式P2P网络Napster和BitTorrent。“混合式”在这里指的是C/S与P2P的混合，它反映了网络工作模式从C/S到P2P的过渡，Napster正是如此。发展到后来的BitTorrent网络相当于分散化的多个Napster的集合，由此可见分布式的思想在混合式P2P网络里有着更深层次的渗透。

2.1 Napster——P2P网络的先驱

Napster是世界上第一个应用性P2P网络，也是混合式P2P体系最杰出的代表，它以不可抗拒的影响力向世界传达了P2P的独特思想，展现了P2P的巨大潜力，是P2P网络当之无愧的先驱。

2.1.1 Napster出现的背景和它创造的奇迹

1999年18岁的Shawn Fanning开发出Napster，它提供服务允许音乐迷们交流MP3文件。与传统的提供音乐下载的网站不同的是，Napster服务器里没有一首歌曲，它只是提供一个空间供音乐迷们将自己硬盘上的歌曲文件共享（用户发送歌曲索引信息到服务器）、搜索其他用户共享的歌曲文件

并到其他 Napster 用户硬盘上去下载歌曲。尽管 Napster 的工作原理听起来并不新奇，它所用到的软件技术也是计算机领域以前就有的，但是，它却是最先在互联网上让用户之间不经过服务器直接交换文件的应用体系。Napster 无意中采用了 P2P 模式，打破了客户/服务器模式的瓶颈，它背后隐藏的巨大的 P2P 世界也从此浮出水面。

Napster 发布后，在短短半年时间里吸引了 5000 万注册用户，最高时超过 6100 万用户，这在计算机网络领域是一个从未有过的奇迹。Napster 创造的奇迹同时也揭示了在互联网时代普通人也具有改变世界的能力——当 Shawn Fanning 最初在波士顿的东北大学校园开发 Napster 软件的时候，他只不过是想和弗吉尼亚的朋友共享 MP3 歌曲文件，而随后这个民间小软件影响了整个世界。

2.1.2 Napster 网络的工作原理

图 2.1.1 是 Napster 的工作原理图，Napster 网络由两个部分组成：Napster 网站＋Napster 用户。

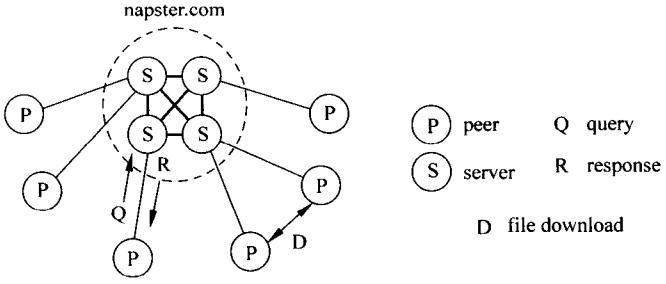


图 2.1.1 Napster 工作原理图
（图片来自[Saroiu et al.,2003]）

Napster 网站（图 2.1.1 中 napster.com）是一个服务器机群。每个服务器保存一部分用户的共享文件索引信息，所有的服务器互联、整合起来对网站外面的 Napster 用户提供统一的访问接口，在每个用户看来他们访问的都是同一个服务器。

每个 Napster 用户（图 2.1.1 中 peer）连接到机群中的一台服务器，他将愿意与其他用户共享的文件信息发送给服务器（图 2.1.1 中 server），服务器记录这些信息以及该用户的位置，并将它们做成一条索引添加到原有索引表中。当用户想要查询一个文件时，首先将“查询”消息发送给与其相连的服务器（图 2.1.1 中 Q：query），该服务器收到 Q 以后，与其他服务器协作处理查询消息 Q，处理完成后将“回复”消息（图 2.1.1 中 R：response）返回给用户，这条消息包含一个表单，列有所查到的所有匹配的文件索引。收到 R 以后，用户在表单中选择他想要的文件，根据文件索引中对应的位置与其他用户直接建立连接下载文件（图 2.1.1 中 D：file download）。

Napster 网站一方面维护所有 Napster 用户的共享文件索引，另一方面监控系统中每个用户的状态，比如跟踪记录用户所报告的连接带宽和用户已连入 Napster 网络的时间，以及发现哪些用户已经掉线等。这些信息对 Napster 系统的运作有重要的意义：首先，对于那些掉线的用户，必须在服务器中去掉该用户对应的文件索引，以保证文件索引的时效性；其次，当服务器收到用户的查询消息后，返回给该用户的回复消息中也包含了其他用户的连接带宽、连接时间等信息，这些信息对于该用户选择与哪些用户建立什么样的连接是很有帮助的。

2.1.3 Napster 的性能分析

文献[Saroiu et al.,2002；2003]对 Napster 做了详细的测量和分析，他们通过 Crawler(爬虫)工具来测量 Napster 网络。网络爬虫的工作原理大致如下[Sen and Wang,2004]："爬虫"首先作为一个结点加入 P2P 网络并与其他一些结点主动建立 TCP 连接，然后通过现有邻居获取更多结点信息，建立更多的连接，同时，"爬虫"结点记录下它与其他结点交换的信息以供后续分析。爬虫技术是"带宽密集型"的，因为它需要维持很多条 TCP 连接；同时，爬虫技术也是"侵犯型"的，因为爬虫自身参与到网络中来并对网络产生了影响，从而导致测量结果的不准确性(读者可以与物理学中著名的"测不准原理"相类比)。

根据他们的观察，Napster 网站机群由大约 160 台服务器组成，每个用户只和其中一台服务器建立连接，当这台服务器收到用户的查询消息时，它首先检索自己的索引以获得与它连接的所有"本地用户"的共享文件信息，然后检索其他服务器的索引以获得与其他服务器连接的"远程用户"的共享文件信息。

当新的用户加入 Napster 网络时，它告诉服务器自己的连接带宽，Napster 服务器记录这个值并向其他用户提供这个值，这对于系统的工作很有帮助，根据带宽信息用户可以选择与哪些用户建立什么样的连接。然而，许多 Napster 用户(大约 25%)加入网络时并不告诉服务器它的连接带宽，更坏的情况是，许多用户故意告诉服务器比其实际带宽低很多的连接带宽，从而阻碍别的用户从它下载文件。

虽然 Napster 将所有的用户看成平等的成员，但是 Napster 用户之间在能力上却有着巨大的差异，也就是 P2P 所讲的结点"异构性"。就网络连接设施而言，约 25%的用户使用低带宽的调制解调器(Modem,64Kbps)，约 50%的用户使用较高带宽的连接(Cable,DSL,T1 或 T3)，约 20%的用户使用高带宽连接(至少 3Mbps)，详见图 2.1.2。就用户连接时间而言，超过 50%的用户连接时间低于 1h，不到 10%的用户连接时间高于 6h。

在理想情况下，Napster 希望它的每个用户都能在下载的同时提供一些文件上传，这样的网络才高效、平衡。然而，在实际的 Napster 网络中，很多用户都是自私的，约 20%～40%的用户几乎从来不提供文件共享而只是下载别人的文件，即

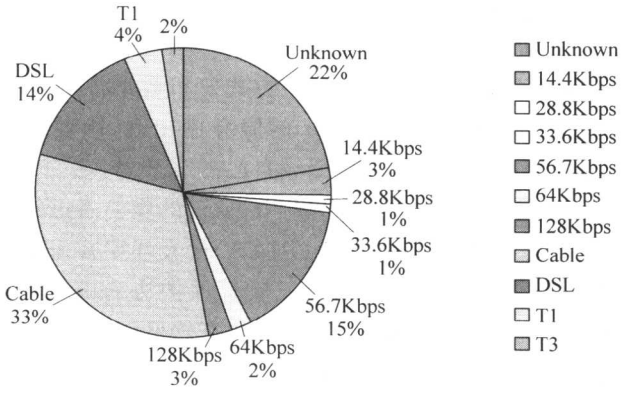


图 2.1.2　Napster 用户所报告的连接带宽和连接设备
（图片来自[Saroiu et al.,2003]）

使对于不自私的结点，绝大部分也只提供少量的文件共享。相反，Napster 中真正提供文件共享的，是网络中少数的约 1%的结点，它们才是 Napster 真正的文件提供者。上述现象在 P2P 领域中也被称为“Free Riding”（搭便车）。

对 Napster 网络性能，我们做如下总结：

（1）Napster 网络中的用户功能上平等，但实际能力差异很大，结点异构性表现在连接带宽、时延、连接时间、共享文件数等多个方面。

（2）许多 Napster 用户不报告或者错误地报告它的连接带宽给 Napster 服务器，这不利于整个网络的高效工作。

（3）Napster 网络中存在很多自私结点，它们只下载文件，几乎从来不上传。自私结点对于整个网络来说几乎是没有贡献只有消耗的。

基于上面的分析，Napster 或者类 Napster 的 P2P 网络在设计、优化时必须考虑这样三个问题：对结点异构性，系统应该有机制针对不同的结点能力采取不同的措施，让它们扮演不同的角色；对用户信息的报告，系统应该有方法鼓励正确的报告或者限制错误报告；对自私结点，系统必须有方法鼓励上传，有限制或禁止自私结点使用网络的权利。

2.1.4　Napster 的陨落和它的现状

Napster 创造了计算机网络领域从未有过的奇迹，然而，这个奇迹并没有持续太久。Napster 被多家唱片商以侵犯版权为由推上被告席而再次成为业界的焦点，最终以 Napster 的陨落作为落幕。下文描述了 Napster 的陨落。之所以要详细描述这一过程，其目的不仅在于 Napster 本身，更重要的是告诉人们 P2P 网络所面临的最大困境：版权问题的法律纠纷。

可能从来没有一个行业像唱片业这样，生存会因为一个小小的软件而受到如此沉重的威胁。1999 年 10 月 31 日，原告之一 BMG 公司和 Napster 达成和解协议，舆论认为这表明唱片业意识到通过数字方式发布音乐将是不可阻挡的潮流，消灭 Napster 也无法阻止其他模仿 Napster 的服务商出现，因此还不如与之合作改变 Napster，将 Napster 的 5000 万用户变成自己的客户，将之变成在线音乐销售的渠道。但随之而来的困境是如何防止那些 Napster 的模仿者继续免费提供歌曲。

1999 年 12 月 7 日，美国唱片业协会(RIAA)代表环宇音乐、索尼音乐、华纳音乐、百代唱片、BMG 等七大唱片公司以违反版权保护法为由把 Napster 公司推向法庭。他们称 Napster 向网民提供 MP3 文件共享软件侵犯了音乐版权，要求法院关闭该公司并判其赔偿损失 1 亿美元。

2000 年 2 月，美国旧金山第九巡回上诉法院的三名法官就音乐网站 Napster 版权纠纷案做出裁决，认为它侵害了各大唱片公司的版权，并且认为 Napster 一直明白它的用户在侵权却对这种侵权行为采取忽视和纵容的态度。但是三名法官并没有应唱片公司的要求，决定立即关闭 Napster 网站，而是把最初的判决送回给低一级的地方法院。法官们说，这项裁决内容过于复杂，需作进一步澄清。错综复杂的法律过程和长达 58 页的判决书反映了现在既缺乏与互联网相关的版权法，也缺乏相应的司法实践。

但是，不管 Napster 案的结果如何，都不能改变 Napster 背后的技术和思想给互联网带来的影响："魔鬼"已经钻出了"魔瓶"，而"魔瓶"也已经被打破了。对于唱片界来说，至少它们销售唱片的方式被彻底改变了。下一个可能是好莱坞的电影工业，因为压缩技术和宽带网络将使得人们能在网络上轻易地传输整部电影，视频和音频就数据量而言已不存在鸿沟。另一方面，网络上已经出现了许多试图推出"合法化的 Napster"模式的公司，譬如由 Napster 的创办人之一创办的 Lightshare.com 和 Flycode.com，利用一个集中的站点提供收费的音乐下载。传统方式下用户每次从网站下载的时候，网站都必须向电信部门交流量费。使用 P2P 则使得这部分费用不再存在了，因此网站获得更多的利润，而用户也可以得到更便宜的音乐。

2000 年 5 月 5 日，美国地区法官 Marilyn Hall Patel 作出判决，依据《美国 2000 年数字版权法》，Napster 的"安全港"资格被取消。

2000 年 6 月，美国唱片业协会(RIAA)和美国音乐出版协会(NMPA)向加利福尼亚州北地区联邦地方法院起诉 Napster 公司，请求法院禁止在社会上流通 Napster 公司的 MP3 文件交换软件"Napster"。

2000 年 7 月 11 日，参议院围绕 Napster 展开的诉讼召开听证会，无果而终。一些议员敦促国会立法，以澄清 Napster 公司是否违反了知识产权法；而支持 Napster 一方的人却认为国会不应该现在就介入双方的争端，以免影响新技术的

发展。7月26日，法官Patel同意美国唱片业协会的要求，做出初步判决，命令Napster立即停止服务。7月28日，美国第九巡回上诉法院暂缓了较低一级法庭的禁令，认为Patel的判决会引发大量问题。

2000年10月31日，Napster宣布同德国的媒体巨人贝塔斯曼集团结成伙伴关系，共同开发基于会员制度的音乐发放系统，通过这种方式，可以保证对艺术家的付款。根据双方的协议，贝塔斯曼同意撤销对Napster的起诉，Napster可使用贝塔斯曼的音乐。

2001年2月12日，美国第九巡回上诉法院作出决定，Napster必须终止其免费互联网服务，并且不得再向乐迷提供共享受版权保护的音乐服务。

2001年2月20日，Napster提出支付10亿美元的费用给唱片公司以结束诉讼，但是两天后这一提议被唱片公司拒绝。

2001年3月2日，Napster公司宣布，将开始阻止用户下载大约100万首受版权保护的音乐曲目，以遵守法官Patel随时可能下达的一项新的法律禁令。

2001年3月6日，美国地区法官Patel作出判决，责令Napster在5个工作日内删除所有存在争议的歌曲。

2001年6月25日，Napster和英国独立音乐协会、独立音乐合作协会签署了一个世界性的专利使用权转让协定，以给它新的订阅服务提供音乐。

2001年6月27日，电影艺术和科学研究院上诉Napster，控告Napster的在线服务允许用户下载奥斯卡电视广播期间艺员的表演录像。

2001年7月12日，法官Patel命令Napster继续关闭直到它可以证明能够有效地阻止对版权文件的使用。Metallica和Dr. Dre澄清了对Napster的法律争论，结束了双方之间所有的诉讼。

在最兴旺的时候，Napster拥有超过6000万使用网站服务下载歌曲的用户。但是在联邦法院判决文件交换服务违反版权法后，Napster在2001年关闭了服务，最终在2002年6月宣布破产。实际上当贝塔斯曼停止收购Napster时，Napster就已经意识到了这一点，当时的Napster网站上贴出了悲观的图片（见图2.1.3）。

2002年11月，消息传出：美国软件生产商Roxio将以超过500万美元的价格收购目前处于困境的音乐交换网站Napster的剩余资产。在得到特拉华州破产法庭的许可后，这笔交易迅速达成，Roxio以现金外加30万美元Roxio普通股完成交易，获得Napster的专利和品牌标签。但是Roxio不承担Napster的任何债务，Napster的其他硬件资源，诸如服务器、路由器、计算机等仍将是破产企业的一个组成部分，并于12月11日拍卖。对于Napster而言，作为一家软件公司而不是媒体集团，被Roxio收购后，其音乐P2P交换的生涯很难继续下去，实际上真正的Napster的历史已经结束。

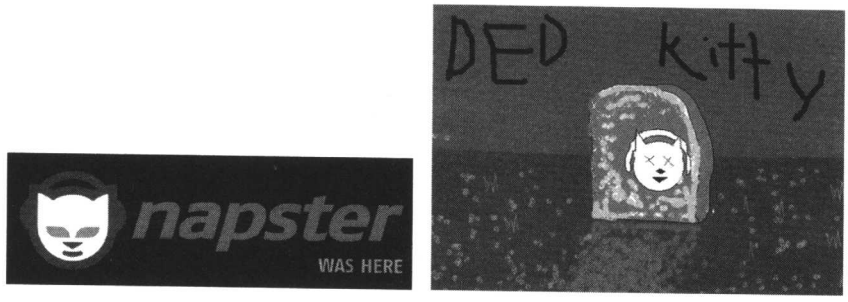


图 2.1.3 陨落的 Napster
（图片来自已关闭的 Napster 网站）

现在，Napster 变成了一个普通的付费音乐下载网站（见图 2.1.4）。虽然名字没有变，网址也没有变（仍然是 http://www.napster.com），但是，这个网站再也不是曾经的 Napster 了，它和 P2P 也再没有任何关系。

图 2.1.4 2006 年 Napster.com 网站页面

2.1.5 Napster 的缺陷和新的混合式 P2P 网络

1. Napster 的缺陷

（1）C/S 的残余：Napster 网络是 C/S 与 P2P 模式混合的结构，虽然本质的文件交换是 P2P 的，但是文件查询和系统维护都依靠 Napster 服务器，也就不可避免地带来了系统瓶颈、服务器单点失效、可扩展性低等问题。

（2）过于松散的组织管理：Napster 赋予它所有的用户平等的功能，却没有考

虑到它们能力上的差异，同时系统缺乏鼓励用户报告正确的信息和提供文件与别人共享的机制，使得 Napster 不能真正高效地工作。

(3) 版权问题：这是 Napster 和整个 P2P 领域面临的最大困境，也是导致 Napster 陨落的直接原因。虽然 Napster 网站服务器里没有任何音乐文件，只是提供文件索引，但是在漫长的法律纠纷之后，Napster 还是被判侵犯版权并从此消失。

2. 新的混合式 P2P 网络

Napster 像一颗流星，它短暂的生命放射出巨大的光芒，虽然陨落却照亮了整个 P2P 领域。在 Napster 之后，新的混合式 P2P 网络快速成长起来，其中最著名的是 BitTorrent，它也是一个以服务器为核心提供共享文件索引的 P2P 网络，所不同的是它提供了文件分片和散列函数映射，同时限定了用户在下载的同时必须上传，而网络和用户信息的更新，尤其是 BT 种子的维护，是依靠服务器中的 Tracker (追踪者)来完成的。下载同一文件的用户围绕 Tracker 形成一个独立的子网。BitTorrent 的服务器基于文件而不是基于整个网络，不同文件的 Tracker 通常在不同的服务器上，从而将服务器分散化了。BitTorrent 是 P2P 在中国最成功的应用，也拥有最为广大的用户群，鉴于此 2.2 节将对 BitTorrent 做深入的讲解。

2.2　BitTorrent——分片优化的新一代混合式 P2P 网络

2.2.1　BitTorrent 的曲折历史

BitTorrent 的第一个可用版本出现在 2002 年 10 月，然而，它的发明人，“BT 之父”Bram Cohen 那时却穷困潦倒。本书对人物的介绍很少，但在这里仍然要不吝篇幅贴出这位值得敬佩的 P2P 技术专家的肖像，见图 2.2.1。幸运的是，BitTorrent 引起了著名免费软件企业家 John Gilmore 的注意，他帮助 Cohen 解决了部分生活费用，使得 BitTorrent 免遭夭折。

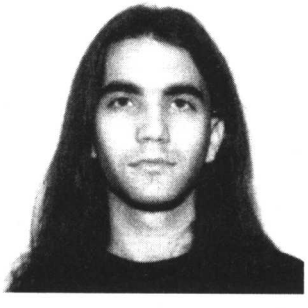


图 2.2.1　“BT 之父”Bram Cohen

BitTorrent 真正流行起来是在 2003 年初，它被用来发布一个新版的 Linux，与此同时，还有一些日本卡通的爱好者借助它来共享动画片。Internet2 主干网(又称 Abilene 主干，它连接起了美国 200 多所规模最大的大学，速度比现时的 ADSL 要快 3500 多倍)基础构造的主管 Steven Corbato 说：2003 年 5 月 BitTorrent 的流量开始激增，从 10 月开始 BitTorrent 的流量更是超过了

这个超高速网络总体流量的 10%；与之对比，其他文件交换系统的流量没有一个能超过 Internet2 总体流量的 1%。然而，这只是开始。

具有讽刺意味的是，尽管 BitTorrent 获得了如此的成功，它并没有为 Cohen 带来一分钱。而在 2003 年 9 月，他还在利用一张信用卡的免息期透支来填补另一张信用卡的账单来过活。

“每个人都有时来运转的时候。”不久，他的事情为 Valve 软件公司的常务董事 Gabe Newell 所获悉，尽管 Valve 正在开发令游戏玩家望眼欲穿的 Half-Life 2，但它同时也在建立一个名为 Steam 的在线分发网络。由于 Cohen 掌握这个领域的专门技术，所以 Valve 为他提供了一个职位，Cohen 从 10 月份起搬到西雅图，开始了他的工作。

今天，BitTorrent 既指一个混合式 P2P 网络以及它所对应的协议，同时又是一个支持该协议的应用软件（支持 BitTorrent 协议的应用软件还有 BitComet、BitSpirit、FlashBT 等），在 BitTorrent.com 可以下载到 BitTorrent 软件不同平台的版本，同时这个网站也是一个非常好的 .torrent 文件（BT 种子）搜索引擎，见图 2.2.2。

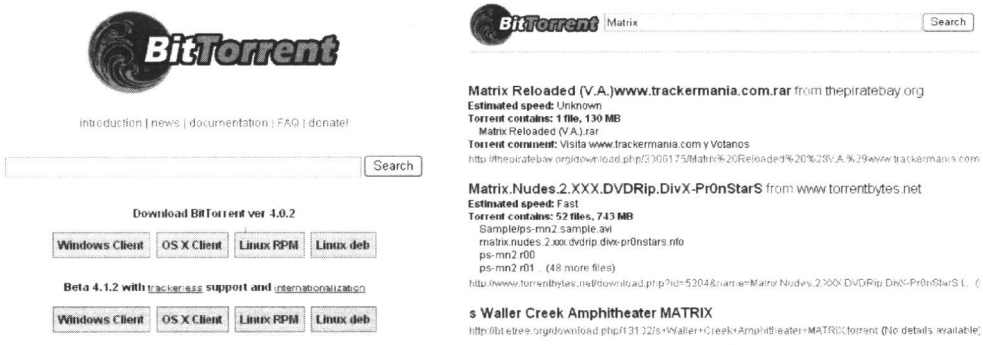


图 2.2.2 www.bittorrent.com 及其搜索“Matrix”的结果（2006 年 12 月）

BitTorrent 在中国非常流行。在国内所有的 P2P 应用体系中，BitTorrent 占据了大半的市场和用户群。2006 年底发布的中国互联网统计报告显示：中国 1.11 亿网民中有 27.8%的人使用过 BT 软件（超过 3000 万人），其流行程度由此可见一斑。BitTorrent 在中国的风靡一时，实际上创造了 P2P 领域新的奇迹，它在中国的影响力不亚于 P2P 的祖先 Napster 当年在美国带来的巨大冲击。国内常用的 BT 软件有 BitTorrent、BitComet、BitSpirit、FlashBT 等，BT 网站和搜索引擎已经很多并且一直在扩展中，目前最有影响力的是 BT@China 联盟（www.btchina.net）、冰鱼 BT 站（bt.icefish.org）、影视帝国（bt2.cnxp.com）、教育网 BT 总站（bt.5qzone.net）等。“好 123 网址之家”网站收录了很多 BT 网站链接：http://www.hao123.com/bt.htm。

2.2.2　BitTorrent 体系原理

这一小节对 BitTorrent 体系原理的讲述基于 Pouwelse 等人的论文[Pouwelse et al.,2004]，和他们在 IPTPS'05 上发表的论文[Pouwelse et al.,2005]。BitTorrent 网络由 4 个部分组成：BT 网站、.torrent 文件服务器、Tracker、BT 用户，如图 2.2.3 所示(BT 网站未画出)。

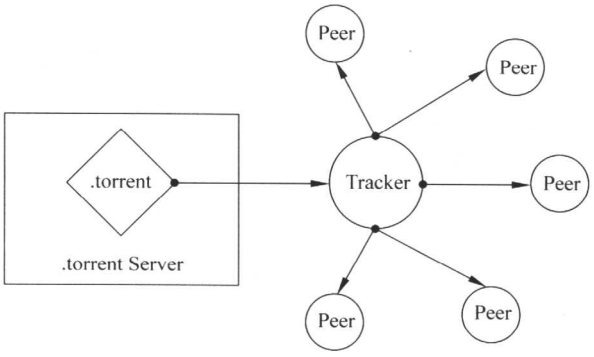


图 2.2.3　BitTorrent 体系原理示意图
(图片来自[Lua et al.,2004])

BT 网站是一系列提供 BT 种子文件(即.torrent 文件)搜索的服务器，从一个 BT 网站可以搜索到一部分种子文件(但这只是因特网 BT 种子文件的一个子集)。由于 BitTorrent 软件本身不提供文件搜索功能，BT 用户要下载文件必须首先到一个 BT 网站进行搜索，从 BT 网站返回的.torrent 文件列表(如图 2.2.4)中选择所要的种子文件。文件列表的含义如下(从左至右)：Added，.torrent 文件上传日期；Name，文件名；Filesize，文件大小；Seeds，种子数(这里"种子"指拥有整个文件的用户)；DLs，在 Tracker 注册的下载用户数(Tracker 跟踪所有下载同一个文件的用户)；Quality，软件运行的操作系统；Submitter，上传 .torrent 文件的用户；Info，到相关网页的连接。除了上面的内容，一个 .torrent 文件还包含它的 Tracker 的位置，以及所要下载文件的散列值用来验证数据完整性。

Added	Name	Filesize	Seeds	DLs	Quality	Submitter	Info
29-02	Castlevania (NES) completed in 12:23...	37 Mb	5	5	-	*anonymous*	link
17-03	Everquest2 Trailer	205 Mb	0	1	Windows	*anonymous*	-
07-03	FFX2 FMVS (Str6 FROM DVD)	919 Mb	1	14	PS2	Tidus_Beta	link
06-02	Partition Magic 8.0 español by Nitro...	50 Mb	4	2	N64	*anonymous*	-
30-11	Quake 1 beaten in 12 minutes, 23 sec...	166 Mb	8	5	Windows	haze	link
29-02	Rockman (NES) completed in 21:53 by ...	50 Mb	4	3	-	*anonymous*	link
15-11	S.T.A.L.K.E.R. Trailer 5 hi-res	47 Mb	0	1	Windows	Ezth Nahua	-

图 2.2.4　.torrent 文件列表
(图片来自[Pouwelse et al.,2004])

.torrent 文件服务器(图 2.2.3 中.torrent Server)保存.torrent 文件，它相当于一个小型的种子数据库，通过 BT 网站来搜索。在 BitTorrent 网络中，一个文件的共享开始于.torrent 文件的产生，通常它由文件的拥有者提交给 BT 网站(这个拥有者是该文件的第一个“种子”)，经过检查确保内容无误后被存放到.torrent 文件服务器中。

Tracker(跟踪者、跟踪服务器)是 BT 网络和用户信息的维护者，其职责是帮助用户相互发现对方，下载同一个文件的所有用户围绕 Tracker 形成一个独立的子网，它对一个文件的下载起着关键性的作用。Tracker 跟踪所有下载某个文件的用户，并实时地将这些用户信息发给其中的每一个，它控制着 BT 网络上某个文件(也可能是多个文件)的下载和用户之间的协同工作。Tracker 和用户之间使用一种很简单的基于 HTTP 的协议进行交互：用户告诉 Tracker 要下载的文件、自己使用的端口以及其他信息，Tracker 告诉用户下载同样文件的其他下载者的联系信息，用户可以利用这些信息相互之间建立连接；此外，一些关于下载和上传速率的信息被发送给 Tracker，Tracker 搜集这些信息用于统计。

在最近版本的 BitTorrent 软件中，即使没有 Tracker 也能定位到拥有文件的用户，这是通过基于 Kademlia 的分布式散列表来完成的(Kademlia 是一种实用、容错的结构化 P2P 网络，将在第 4 章对其进行描述)，在“用户列表”中以“DHT Network”的形式表示出来。但是，目前靠无 Tracker 的 DHT 方式定位文件的速度要远远低于有 Tracker 的服务器方式，因此它只能作为 BitTorrent 的一个辅助定位方法。

每个 BT 用户(图 2.2.3 中 Peer)可以同时下载多个文件。对于每一个文件而言，用户通常按照下述步骤来下载：

(1) BT 用户通过某个 BT 网站搜索文件(如 BT@China 联盟网站)，BT 网站将该搜索请求重定向到它的网站镜像(比如 BT@China 联盟就有很多镜像站点)，检索它所知的所有.torrent 文件服务器，返回给用户关于该文件的.torrent 文件列表。

(2) 用户选择列表中的一项或者多项，被选的每个.torrent 文件会启动一项下载任务。通常，BT 软件会根据.torrent 文件信息连接到该文件所对应的 Tracker，从 Tracker 获得当前也在下载该文件的用户信息(Tracker 一般只返回一定数量的用户而不是全部)。

(3) 用户根据上一步从 Tracker 获得的用户信息与他们建立直接的连接，从他们下载文件的一部分分片，同时也向他们提供上传。

(4) 为了保证 BT 用户的高速下载和整个网络的高效工作，一个用户并不总是和最初建立连接的那些用户互相交换文件，而是每隔一段时间从 Tracker 获得新的用户信息建立新的连接，同时采用“阻塞算法”主动地停止那些对自己来说没

有什么利益的连接、建立对自己更好的连接。

在上述步骤的第 1 步和第 2 步中，出于安全性的考虑，BT 网站可能还会要求用户做一项工作：输入网页中以图形表示的随机字符串，然后才能获得. torrent 文件列表或者连接到文件对应的 Tracker。在第 1 步输入随机字符串的目的，一方面是为了防止通信监听（此时随机字符串相当于文件加密的密码）；另一方面也有效地抵制了 DoS 攻击（denial of service，服务拒绝攻击），如果不要求输入字符串，恶意用户可以无限制地向 BT 网站发出查询请求从而使网站服务器过忙而拒绝提供正常服务。在第 2 步输入随机字符串的目的与第 1 步类似。上述 BT 安全机制虽然很不完善而且留下了很多可以攻击的漏洞，但仍然可以称得上是简单、有效的安全措施。对于混合式 P2P 网络而言，这样的安全机制是合情合理的。

2.2.3　BitTorrent 分片机制

本节所讲的 BitTorrent 分片机制，以及下一小节的 BitTorrent 阻塞算法，都来自 BT 发明人 Bram Cohen 的论文[Cohen，2003]。

为了跟踪每个 BT 用户都拥有些什么，BT 将文件分割为固定大小的分片（典型的大小是 256KB），每个用户必须通知其他下载者他拥有哪些分片。为了验证数据完整性，对每个分片都通过 SHA-1 散列函数计算出它的散列值保存在 .torrent文件中，用户只有在检查了分片的完整性之后，才会通知其他用户他拥有这个分片。

用户不断地从他能连接到的用户那里下载文件分片，当然，即使是建立了连接的用户，有的也并不包含想要的分片，或者还不允许他去下载（关于不允许其他用户从自己那里下载文件分片的策略，被称为阻塞算法，下一节将讲到）。关于文件分片有更高效、更容错的机制如 Erasure Coding（冗余编码）方法，但是也增加了复杂性；BT 简单地让用户公布它拥有的分片会导致不到 10% 的带宽开销，却可以有效、可靠地使用用户的上传能力。

分片流水作业：构架在 TCP 之上的应用层协议，例如 BT，很重要的一点是应该同时发送多个请求，以避免在两个分片发送之间的延迟，因为那样会严重影响传输速率。为了达到这个目的，BT 又进一步将每个分片划分为子分片（典型的大小是 16KB），同时，它一直保持几个请求（通常是 5 个）被流水式地同时发送。流水作业选择同时发送的请求数目的依据，是使大多数连接变得饱和以充分利用带宽。

分片选择：选择一个好的顺序来下载分片对提高性能非常重要。差的分片选择算法不仅低效，而且使用户之间相互等待浪费了宝贵的带宽资源。在不同阶段，BT 的分片选择策略有所不同：

（1）严格的优先级（一个分片的下载）

分片选择的第一个策略是：一旦请求了某个分片的子分片，那么该分片剩下的所有子分片优先于其他分片的子分片被请求，这样可以尽可能快地获得一个完整的分片。

（2）最少者优先（文件下载中间阶段/平稳期）

对一个下载者来说，在选择下一个要下载的分片时，通常选择的是他所知用户拥有数最少的那个分片，这就是所谓的“最少者优先”，通俗一点讲其实就是“物以稀为贵”。这种策略确保了每个下载者都拥有与他连接的用户们最希望得到的那些分片，从而一旦有需要，上传就可以开始；同时这也确保了越普通的分片越放在最后下载，从而减少了这样一种可能性：某个用户当前正提供上传，但随后他却没有任何被别人感兴趣的分片了（越稀少的分片通常越让别人感兴趣）。

BT 的分片选择充分考虑了经济学的因素，处处从整个系统的性能出发，参与者越多，系统越优化。如果在某次下载中只有一个种子，而且种子的上传能力比大多数下载者都要差，那么，不同的下载者从种子那里下载不同的分片，系统性能就会变得比较好，因为重复下载相同的分片浪费了种子发出更多信息的机会。“最少者优先”使得下载者只从种子处下载新的分片（也就是整个系统中其他用户都没有的分片），这得益于 BT 网络中下载者能够知道其他用户那里已经有了哪些分片（Tracker 的功劳）。

在某些下载中，原始的种子由于某些原因最终关闭，只好由剩下的这些下载者来负责上传，这样显然会带来一个风险：某些分片任何一个下载者都没有。“最少者优先”也很好地处理了这个问题：通过尽快地在用户中复制最稀少的分片，减少了由于当前种子停止上传所带来的分片缺失风险。

（3）随机的第一个片段（文件下载最初阶段）

“最少者优先”的一个例外是在下载刚开始的时候，此时下载者没有任何分片可供上传，所以需要尽快获取一个完整的分片。但此时最少的分片通常只有某一个用户拥有，所以，下载它可能比下载多个用户都拥有的那些分片要慢得多。因此，第一个分片是随机选择的，直到第一个分片下载完成，才切换到“最少者优先”策略。

（4）最后阶段模式（文件下载最后阶段）

有时候，可能会从一个速率很慢的用户那里请求一个分片，在下载的中间阶段这没有太大问题，但是在下载的最后阶段它却可能潜在的延迟下载的完成。为了避免这种情况，在最后阶段，下载者向他所连接的所有用户都发送某分片的子分片请求，一旦某个子分片到了，下载者就会向其他用户发送 Cancel 消息，取消对那些子分片的请求，以避免带宽浪费。实际上，这种方法消耗的带宽并不多，但文件的结束部分可以因此下载得非常快。

2.2.4 BitTorrent 阻塞算法

BT 并不是由 Tracker 服务器来集中分配资源，每个用户自己有责任来尽可能地提高自己的下载速率。下载者从他可以连接到的用户处下载文件，并根据对方提供的下载速率给予同等的上传回报（"tit-for-tat"，以牙还牙/针锋相对）。对于合作者，提供上传服务；对于不合作的，就"阻塞"对方。所以说，阻塞是一种临时的拒绝上传策略。虽然上传停止了，但是下载仍然继续，当不再阻塞的时候，也不需要重新建立连接。阻塞算法对提高 BT 性能是非常必要的。一个好的阻塞算法应该利用所有可用的资源，为所有下载者提供一致、可靠的下载速率，并适当惩罚那些只下载而不上传的自私用户。

（1）阻塞算法的经济学背景——帕累托有效

当一个系统中资源配置已达到这样一种境地：任何重新改变资源配置的方式，都不可能使一部分人在没有其他人受损的情况下受益，这一资源配置的状态称为"帕累托最优"状态，或称为"帕累托有效"（Pareto efficient）。在分布式系统领域，寻求帕累托有效是一种本地优化算法，BT 的阻塞算法用一种奉献与回报相对应的方式来试图达到帕累托最优。BT 用户对那些向他提供上传的用户给予同样的回报，目的是希望在任何时候都有若干个连接正在进行着双向传输。

（2）BitTorrent 的阻塞算法（choking algorithm）

从技术层面上说，BT 的每个用户一直对固定数量的其他用户保持疏通（通常是 4 个，疏通即指不阻塞），所以问题就变成了哪些用户应该保持疏通。这种方法要使得 TCP 的拥塞控制性能能够可靠地饱和上传容量，尽量让整个系统的上传能力达到最大。

对哪些用户保持疏通应该严格根据当前的下载速率来决定。原来的算法对一段长时间的网络传输进行总计，这种方法效果不好，因为根据资源可用或者不可用，带宽会变化得很快。为了避免因为频繁的阻塞和疏通造成的资源浪费，当前的实现是每隔 20 秒做一次轮询（10+10）：每隔 10 秒计算一次哪个用户要被阻塞，然后将阻塞状态保持到下一个 10 秒。时间间隔取 10 秒是因为这已经足够 TCP 来进行调整以使传输率达到最大。

（3）最优疏通（optimistic unchoking）

如果只是简单地为那些向自己提供最高的下载速率的用户提供上传，那么就没有办法来发现那些空闲的连接是否比当前正使用的连接更好。为了解决这个问题，在任何时候，每个用户都拥有一个称为"最优疏通"的连接，不管它的下载速率怎样。每隔 30 秒，用户重新计算一次哪个连接应该是"最优疏通"。30 秒足以让上传能力达到最大，下载能力也相应达到最大。

(4) 反对冷落(anti-snubbing)

某些情况下，一个下载者可能被他连接的所有用户都阻塞了，此时他将会保持较低的下载速率，直到通过“最优疏通”找到更好的用户来连接。为了缓解这个问题，如果一段时间过后，从某个用户那里一个分片也没有得到，那么下载者认为自己被对方“冷落”了，于是不再为对方提供上传(除非对方是“最优疏通”)。“反对冷落”常常会导致多个并发的“最优疏通”，从而使得下载速率恢复得更快。

(5) 仅仅上传

一旦某个用户完成了下载，它就不可能再通过下载速率来决定为哪些用户提供上传。目前采用的解决办法是：优先选择那些能从他这里得到更高上传速率的用户，或者选择那些此刻刚好被所有人阻塞的用户，以尽可能地利用上传带宽。

2.2.5 BitTorrent 性能分析

文献[Pouwelse et al.,2004; 2005]从 5 个方面分析了 BitTorrent 网络的性能，下面部分总结它们的分析结果。

1. 流行性

由于采用分片优化的多项机制，BT 系统是高效、高可扩展的，只要其核心部分——BT 网站、.torrent 文件服务器以及 Tracker 不发生故障，BT 网络的规模将不断扩大，它也会越来越流行。然而，目前上述三种服务器的故障率都比较高，限制了当前的 BT 规模(如图 2.2.5 所示)。可以明显地看到由于服务器故障导致的几处曲线下降(图片来自[Pouwelse et al.,2004])。

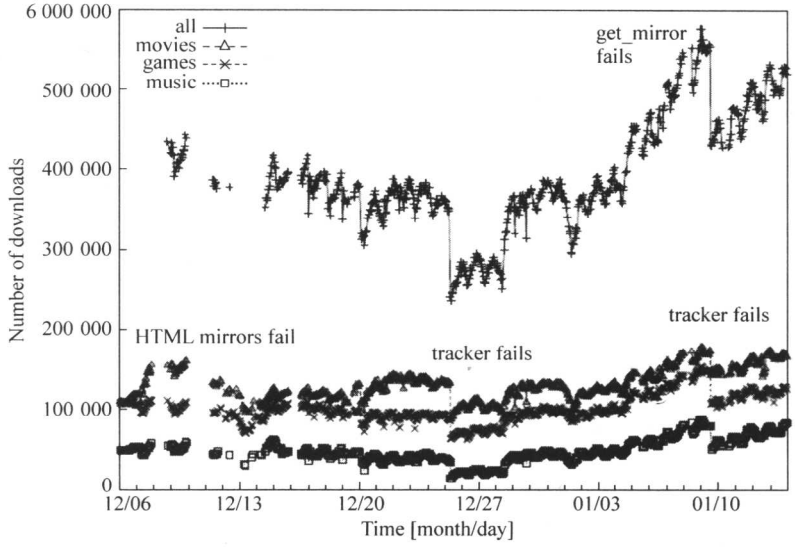


图 2.2.5 一个月的 BT 用户测量(基于 Suprnova 网站)

2. 可用性

BT 的可用性只取决于它的三个核心部分——BT 网站、.torrent 文件服务器以及 Tracker 的可用性，可是目前它们的可用性都不高。总体而言，Tracker 具有较高的可用性，常常可以保证文件被相当一部分用户获得；只有一半的 BT 网站镜像可以正常工作超过 2.1 天；.torrent 文件服务器则比前者更不可靠，如图 2.2.6 所示。

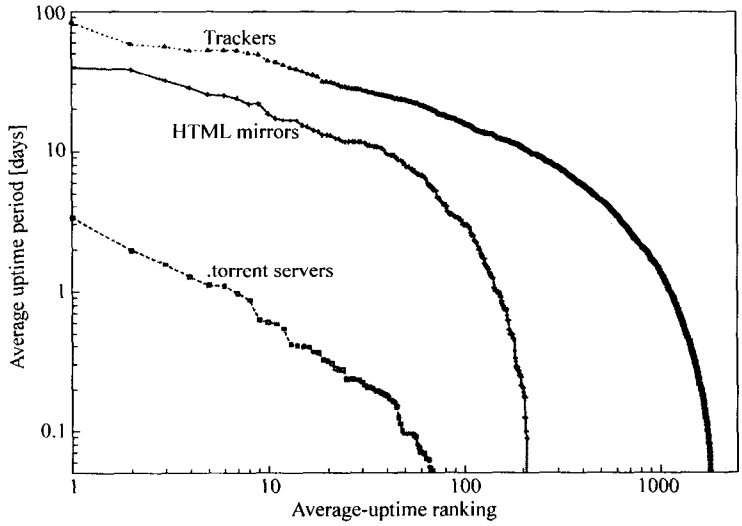


图 2.2.6　BT 网站(HTML mirrors)、.torrent 文件服务器和 Trackers 三者的平均正常工作时间
(图片来自[Pouwelse et al.,2004])

3. 下载性能

BT 用户平均的下载速度为 30KB/s，绝大多数低于 65KB/s，这个速度其实还是比较高的，它允许用户可以在一天内下载非常大的文件(比如高质量电影)。从图 2.2.7 可以看出，用户的下载速度和在该速度下的下载数目之间大致符合指数关系。

4. 文件生命期

文件生命期也就是该文件的种子生命期。由于 BT 网络中服务器的故障率较高以及用户(尤其是种子)上传的不确定性，BT 网络的文件生命期很难预测。在一次测量中，发现有 17%的用户在下载完成后做种时间超过 1 小时，仅有 3.1%的用户下载完成后做种时间超过 10 小时。

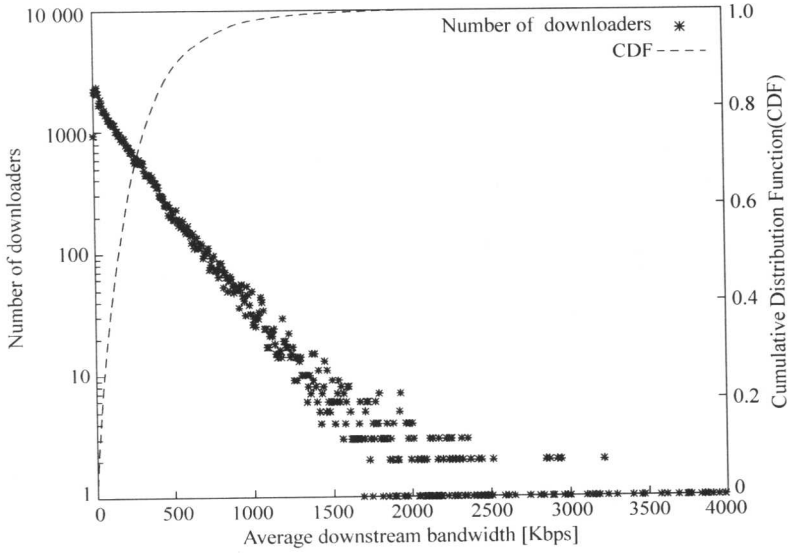


图 2.2.7 BT 用户平均下载速度和在那个速度下的下载文件数目

(图片来自[Pouwelse et al.,2004])

5. 污染等级

所谓"污染等级"是指 BT 用户插入到 BT 网络中的共享文件的真实性。优越于其他 P2P 应用,BT 有专门的"审查系统"(moderation system)来确保文件内容的真实性以降低污染等级。插入文件的 BT 用户被指派不同的角色:(a)审查者(moderator):审查别人提交的文件内容;(b)需要审查的提交者(moderated submitter):普通的不被系统信任的用户;(c)无需审查的提交者(unmoderated submitter):系统信任的无需审查即可插入文件的用户。如果(b)经常提交真实内容的文件,它可以被升级到(c),同样,如果(c)的表现很好也可以升级到(a),这非常类似于人类社会的审查制度。鉴于上述等级化的审查系统,BT 的污染等级一般较低,文件真实性是值得信任的。

2.2.6 BitTorrent 体系总结

(1) BitTorrent 是一个混合式结构的 P2P 网络,它以三个部分——BT 网站、.torrent文件服务器和 Tracker 为核心,控制和帮助 BT 用户共享文件。下载同一文件的用户围绕 Tracker 形成一个独立的子网。

(2) BitTorrent 限定用户在下载文件的同时必须提供上传,这既提高了网络效率,又杜绝了 P2P 网络中的自私结点现象。

(3) BitTorrent 将文件分片，分片又被划分成子分片，子分片流水作业，并且在文件下载的不同阶段有不同的分片选择策略以优化性能。这是 BT 最大的特点，也是它高效的最本质原因。

(4) BitTorrent 基于经济学规律的阻塞算法，优化了网络资源配置，增强了 BT 用户间的协作，它是 BT 网络高效的重要辅助工具。

(5) BitTorrent 对于每个插入的文件有等级化的审查系统，保证了文件的真实性；同时文件、分片都生成对应的散列值供验证，保证了文件的完整性。很多 BT 网站通过要求用户输入随机字符串的方法，有效地防止了通信监听和 DoS 攻击，提供了虽不完善但合情合理的安全机制。

(6) 现有 BT 系统的流行性虽然很高，但是远没有达到其高峰，最主要的原因是 BT 的核心部分——BT 网站、.torrent 文件服务器和 Tracker 故障率高，从而导致低可用性；另一个原因是 BT 中文件的可用性不可预见、不确定，因为这完全取决于"种子"用户的个人喜好，文件无持久性可言。

2.2.7 关于 BT 的一个重要事实澄清：BT 伤硬盘吗

"BT 下载是否伤硬盘?"这是因特网上一个被广泛讨论和关注的问题。因为讨论得多，各种观点鱼龙混杂，反而把一个本来简单的问题说复杂了。BT 下载对硬盘的影响，可以用一句话概括为：不采用磁盘缓存的 BT 下载伤硬盘，采用磁盘缓存的 BT 下载不伤硬盘；而目前的 BT 软件基本都采用磁盘缓存技术，所以通常都不会伤硬盘。

硬盘工作的原理，简单地说就是硬盘磁头读写盘片上的磁道，当磁头读写盘片太过频繁时，盘片可能会由于过热而损坏。最早的 BT 软件没有使用磁盘缓存技术，每下载到一些数据(比如文件的一个分片、一个子分片或者更小的单位)就会把它写入硬盘，当下载量很大时硬盘因为被频繁写入而过热损坏；当上传数据时，硬盘因为被频繁读数据同样会过热损坏。这是"BT 伤硬盘"的原理。

意识到上面的问题后，BT 软件的设计者改变了原有的对磁盘的读写方式，采用了磁盘缓存技术将磁盘读写频率降到非常低，即使 BT 用户高速下载、高速上传也不会伤害硬盘。BT 的"磁盘缓存"技术，对下载而言，每次下载到数据不是直接写入硬盘，而是写入缓存中(缓存通常是内存的一部分)，等到缓存中数据积累到一定容量再整体写入硬盘，因此写硬盘的频率是很低的；对上传而言，需要上传的数据要么可以在缓存中找到、要么可以从硬盘中一次读很多出来，因此读硬盘的频率也是很低的。剩下的问题就是：磁盘缓存应该设多大最合适？通常说来，缓存越大，读写盘次数就越少，也就越保护硬盘。但内存是有限而宝贵的资源，缓存不宜占用太多，否则其他软件相应不够用。以 BitComet 软件来说，缓存的默认值在(6～50)MB 之间动态调整，这样的缓存容量已经可以非常有效地保护硬盘了。

2.3 第一代 P2P 网络的特点

2.3.1 拓扑结构

第一代 P2P 网络，无论是 Napster 还是与之类似的 BitTorrent 等，都采用混合式体系结构，即星形拓扑结构，服务器仍然是整个网络的核心。

2.3.2 查询与路由

Napster 的查询机制简单而高效，用户询问服务器，服务器返回所要的文件索引。而 BitTorrent 本身不提供查询机制，BT 网站维护着大量的文件索引（即种子文件），用户到这些网站上去查询文件，本质上这和 Napster 查询差不多。在获得文件索引之后，用户根据文件索引直接建立与文件提供者的连接（在 BitTorrent 中，这之间还有一步与 Tracker 的连接），因此路由跳数是 $O(1)$ 的，即常数跳。

2.3.3 容错、自适应和匿名性

以服务器为核心的网络，其容错性只在于服务器的故障概率，如果使用多台服务器组成机群，并且提供冗余、替代机制使得一台服务器故障时它的任务可以被其他服务器所分担，那么这样的系统容错性将会非常高。然而，增加、升级服务器通常需要昂贵的支出，一般网站不愿付出此代价，这是 BitTorrent 网络服务器普遍故障率高、可用性低的主要原因。

第一代 P2P 网络的自组织、自适应基本上依靠服务器的监控，用户之间的协作建立在服务器监控的基础之上，因此只要服务器正常工作，网络和结点信息就能得到有效地维护。

可能是出于简单、高效的考虑，目前的混合式 P2P 网络基本上不提供匿名性，但这并不代表混合式 P2P 网络不能提供匿名性。实际上学术界已经有不少论文提出以服务器为核心的匿名方案，而且这些方案也是实际可行的。

2.3.4 增强机制

Napster 是第一代 P2P 网络的代表，但它是朴素的，留下了许多缺陷。在 Napster 的基础上，后起的混合式 P2P 系统都采用了一些增强机制来提高网络的效率，如 BitTorrent 提供文件分片机制，限定用户在下载的同时必须上传以杜绝自私结点的存在，这些都提高了网络工作效率，当然，也增加了网络复杂性。另一方面，在安全性上，BitTorrent 开始逐步采用一些简单、有效的机制防止常见的网络攻击，这相对于 Napster 也是一大进步。

第3章 第二代P2P网络：无结构P2P体系

这一章讲述4个著名的无结构P2P网络：Gnutella、KaZaA、eDonkey和Freenet。其中，Gnutella是最简单又最具代表性的，Freenet则要复杂得多。发展到后来的KaZaA和eDonkey通过使用“超结点”[①]来构建双层的P2P网络，而作为其核心的超结点层自组织成无结构网络，这是我们将KaZaA和eDonkey也归类为无结构P2P网络的原因所在。

3.1 Gnutella——纯分布式无结构P2P网络

3.1.1 Gnutella出现的背景

在P2P的祖先Napster出现后不久，Gnutella（注意首字母G不发音）——第一个无结构P2P网络，也是第二代P2P网络最好的代表，在2000年3月诞生于NullSoft公司。它的发明人也是著名的MP3播放软件WinAmp的设计者：

① 超结点：SuperNode或UltraPeer，此外还有众多别名，如DirectConnect网络称之为“集线器”（Hub），其他一些网络称之为“反射器”（Reflector）或“防卫器”（Defender）。

Justin Frankel 和 Tom Pepper。

鉴于 Napster 所带来的一连串版权问题、法律纠纷和社会影响，3 月 14 日，在 Gnutella 网站上公开 Gnutella 软件后，NullSoft 公司的母公司 AOL(America On Line，美国在线)担心该软件的普及可能产生预测不到的影响，在过了约一个半小时后就关闭了该网站。因为 AOL 不是一个单纯的 ISP(因特网服务提供商)，而是一家大型媒体企业，它不愿意步 Napster 的后尘。

虽然 Gnutella 是短命的，刚出生就被扼杀在摇篮中，但是，众多的 MP3 迷们早就翘首以盼 WinAmp 开发者推出“类似 Napster 的软件”，就在那宝贵的一个半小时内有几千个用户下载了该软件，而且其后事情的进展之快着实令人惊讶。人们在下载 Gnutella 后纷纷将其公开，还有人改造、克隆该软件，其中最有名的是 gnutella. wego. com，现在还有名为 www. gnutellaworld. net 的网站，在这里可以交换各种相关的信息。所以，当年的 Gnutella 虽然昙花一现，但是它所代表的纯分布式无结构 P2P 网络的思想，却广泛流传下来。

今天，Gnutella 更多地被当作一种典型的无结构 P2P 网络协议，而并非某个具体的应用软件。在技术上读者可以在 rfc-gnutella. sourceforge. net 获得关于 Gnutella 协议的文档，包括正式的 Gnutella 协议 0.4 版和改进后的 Gnutella 协议 0.6 版及 Gnutella 建议文档，Gnutella 协议 0.6 版最突出的改进是明确建议使用 UltraPeer(超结点)。在商业上有专门的网站 www. gnutella. com，这个网站相当于基于 Gnutella 协议的各种应用软件的集合和联盟。基于 Gnutella 协议开发的、适合于不同操作系统的无结构 P2P 应用有：

Windows
Gnutella Clients
BearShare
Gnucleus
Morpheus
Swapper
XoloX Ultra
LimeWire
Phex
Other Downloads

Macintosh
Gnutella Clients
LimeWire
Phex
Other Downloads

Linux/Unix
Gnutella Clients
Gtk-Gnutella
Mutella
Qtella
LimeWire
Phex
Other Downloads

3.1.2 Gnutella 体系的工作原理

图 3.1.1 是 Gnutella 体系的工作原理图，基于 Gnutella 协议 0.4 版。Gnutella 网络中只有一种结点——对等实体(图 3.1.1 中 P：peer)，不再有服务器存在。每个

Peer既是客户又是服务器，既能向其他Peer发送查询请求并获得查询结果，又能接收其他Peer发来的查询请求、返回所要的文件信息或者将此请求路由给其他的Peer，所以Gnutella开发者称Peer为Servent（Server＋Client）。除此之外，Gnutella中每个Peer还负责监控网络局部的通信状态，互相协作以保持整个网络的完整性与一致性。

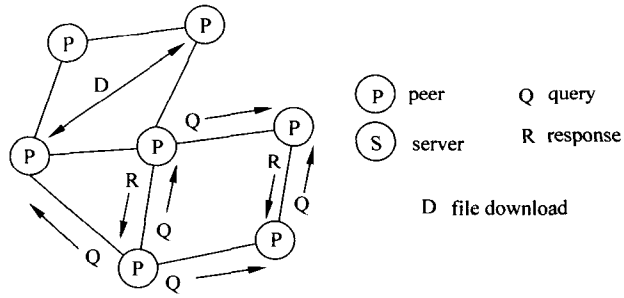


图3.1.1 Gnutella工作原理图

（图片来自[Saroiu et al.,2003]）

Gnutella在因特网上构建了一个应用层覆盖网，覆盖网上每个Peer对应实际的一台网络计算机，而覆盖网上每条连接对应因特网上一条点到点的链路。覆盖网上的连接是由每个Peer所保存的“邻居结点”信息确定的，有一个邻居结点就对应有一条边。

如果因特网上某台计算机要加入Gnutella网络，它必须首先连接到一个“众所周知”的几乎总是在线的Gnutella结点（这些结点也称为“自举”结点、“入口”结点），通过它连入Gnutella网络。Gnutella软件中通常提供一些这样的“众所周知”结点，或者也可以到一些网站上找，这些“众所周知”结点相当于一个中介，将网络外的用户介绍进网络来。“众所周知”结点一般有很多，并且功能都是等价的，所以并不破坏Gnutella纯分布式的特性。

在Gnutella网中消息可以被广播（broadcast）也可以被回播（back-propagate，沿广播的反向路径回传消息），Gnutella协议设计了一些特性以支持广播/回播机制：首先，每条消息具有一个随机产生的全局唯一标识符以互相区分（GUID，globally unique identifier，16字节的随机数）；其次，每个结点缓存最近路由的消息以支持回播和阻止不必要的重广播（re-broadcast，收到消息并将它再次广播到自己的邻居结点）；再次，每条消息都有一个TTL（time-to-live，生存时间、跳数限制），在覆盖网上每走过一跳TTL减1，TTL减到0就不再往前传，从而使得Gnutella洪泛式的广播不会无度地消耗网络资源。

Gnutella协议所设计的消息典型地有：

(1) 组成员消息：PING,PONG

PING：当一个新结点加入网络时它广播 PING 消息宣告它的到来，相当于“I'm here!”；同时 PING 消息也用来探测其他结点是否仍然存在，相当于“Are you there?”

PONG：这是对 PING 消息的回复。当一个结点收到 PING 消息时，它首先决定是否回播 PONG 消息，然后将 PING 广播给它的邻居结点(如果 TTL 没有减到 0)。它包含该结点的 IP 地址、端口号、共享文件的数量和大小等信息，相当于“Yes,I'm here。”

(2) 查询消息：QUERY,QUERY RESPONSE

QUERY：该消息用来查询所要的文件，它包含用户指定的查询内容和附加信息(如最小响应速度)，相当于“I'm looking for...”。每条 QUERY 消息有唯一的标识符，但是发出 QUERY 消息的源结点是不可知的。

QUERY RESPONSE：这是对 QUERY 消息的回复，它包含文件下载的必要信息——IP 地址、端口号、位置、带宽信息等，以及该结点的 nodeID。QUERY RESPONSE 消息沿原先 QUERY 消息传过来的路径回播，相当于“Your wanted file is here：...”。

(3) 文件传输消息：GET,PUSH

GET：该消息用来请求获得文件，它通常在发出 QUERY 的源结点收到 QUERY RESPONSE 之后，相当于“I'll get...from you。”

PUSH：一些网络结点可能位于防火墙之后，所以不能直接响应文件请求，PUSH 消息请求防火墙后的文件拥有者主动建立到自己的连接并将文件上传给自己，相当于“I can't get...from you,so you push it to me!”。虽然如此，如果双方都位于防火墙之后，PUSH 消息也无济于事。

为了查询文件，Peer 给它所有的邻居结点发送 QUERY 消息，这是一种控制性的洪泛路由方法，消息数量是指数递增的，TTL 不能很大，否则网络无法承载巨大的开销。收到 QUERY 消息后，Peer 首先检查是否有它自己共享的文件与 QUERY 消息中的查询请求相匹配，如果有就沿着 QUERY 消息传送过来的路径回播 QUERY RESPONSE 消息。不管是否有匹配的文件，只要 QUERY 消息的 TTL 没有减到 0，Peer 就会继续将它广播给邻居结点。Gnutella 查询方式有两大缺陷：①由于采用洪泛法路由，系统开销很大；②受 TTL 的限制，消息只能到达一定的范围，不能覆盖整个网络，因此即使文件存在，也不一定能被查询到。

Gnutella 网络具有很高的动态性，不断地有新结点加入、旧结点离开，为了保持覆盖网的一致性和 Peer 信息的及时更新，Gnutella 协议使用 PING、PONG 消息来帮助 Peer 发现、探测其他结点的存在与否。任何收到 PING 消息的 Peer 都可能沿着 PING 消息传过来的路径回播 PONG 消息(如果他愿意的话)，并且只要 PING 消息的 TTL 没有减到 0 就会被继续广播给 Peer 的邻居结点，这个方法和

上面的查询非常类似，也是依靠 TTL 来控制的洪泛式广播。通过 PING、PONG 消息，Gnutella 结点不断更新它们的邻居结点信息，保持了整个 Gnutella 网络较好的自组织和自适应性。

3.1.3 Gnutella 网络的性能分析

文献[Ripeanu，2001]、[Ripeanu et al.，2002]、[Saroiu et al.，2002；2003]、[Adar & Huberman，2000]对 Gnutella 从不同的角度作了测量、分析与总结，前面四篇论文通过 Crawler(爬虫)工具来测量 Gnutella 网络的结点连接带宽、时延、连接时间、共享文件分布、容错性、消息分布以及拓扑一致性等，最后一篇则通过运行基于 Java 的 Furi Client 软件来记录系统中流过的消息，从而测量 Gnutella 网络中的 Free-Riding 现象(类似 Napster 中的“自私结点”现象)。

在 Gnutella 网络中，用户的连接带宽只在 QUERY RESPONSE 消息中作为辅助信息返回给发出 QUERY 消息的用户。这与 Napster 不同。Napster 用户加入网络时就会告诉服务器它的连接带宽(当然也可以选择“不告诉”)。因此，Gnutella 网络中那些不共享文件的用户，或者其共享的文件与查询请求一直不匹配的用户，永远不会告诉别人他的带宽。

虽然在 Gnutella 网络中所有的结点都是功能平等的，但是它们之间在能力上却有着很大的差异，也就是所谓“Peer 异构性”。就网络连接设施而言，约 8%的用户使用低带宽的调制解调器(Modem，64Kbps)，约 60%的用户使用较高带宽的连接(Cable，DSL，T1 或 T3)，约 30%的用户使用高带宽连接(至少 3Mbps)，从中可以看出 Gnutella 用户在总体上比 Napster 用户的连接带宽要高很多。从经验上分析这可能来自两方面原因：①Gnutella 协议的洪泛式广播造成了很大的网络负载，这阻碍了低带宽结点的加入；②Gnutella 自产生之初，其用户群就更多地来自技术型、对网络有所理解的群体，他们通常拥有更高带宽的网络连接。

就 Gnutella 结点的时延而言，从图 3.1.2 可以看出，约 20%的结点时延低于 70ms，约 20%的结点时延超过 280ms——刚好是前者时延的 4 倍。从这里可以看出，在无结构的组织方式下，P2P 网络中会有相当一部分结点承受高时延。

就用户连接时间而言，与 Napster 类似，Gnutella 中超过 50%的用户连接时间低于 1 小时，不到 10%的用户连接时间高于 6 小时。就共享文件数而言，Gnutella 网络中高达 25%的用户不共享任何文件，约 75%的用户共享文件数低于 100，只有 7%的用户共享文件数超过 1000，但正是这 7%的用户，他们共享的文件数超过其余 93%用户共享文件数的总和。这正是文献[Adar & Huberman，2000]中所指出的 Gnutella 网中的 Free-Riding 现象。所以说，Gnutella 网中存在着大量自私的 Free-Riders(搭便车者)，这对网络的高效工作是很不利的。

虽然 Gnutella 网络没有严格的拓扑结构，只是一个看起来随机产生的图，然

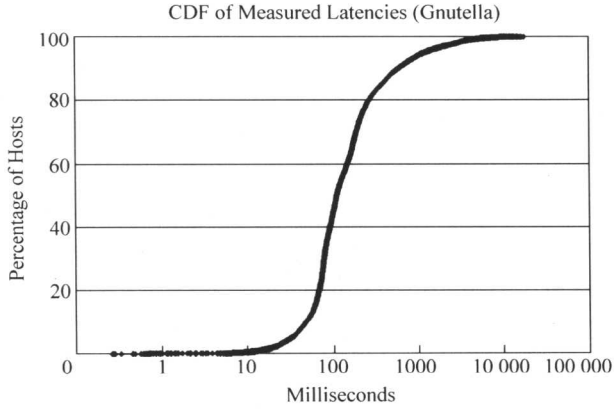
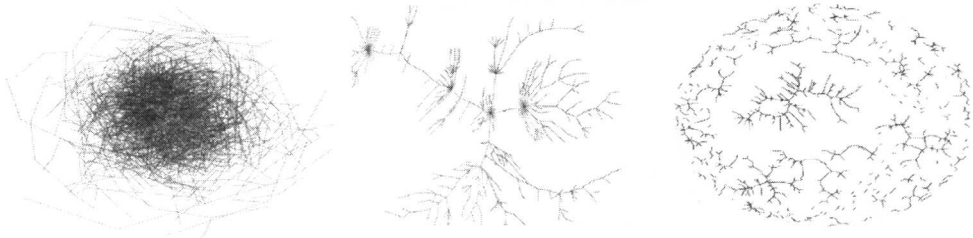


图 3.1.2 Gnutella 网络结点时延
（图片来自[Saroiu et al.,2003]）

而要看到的是，Gnutella 网络的背后是人类社会。过去有研究指出，人类社会自发地形成一个幂律(Power-Law)分布网络，在 Power-Law 网络里，拥有连接数 L 的结点占网络总结点数的份额，正比于 $L^{-\alpha}$，α 是一个取决于网络本身的常数因子，因此大多数结点连接数很少，少数结点拥有非常高的连接数。Gnutella 虽然不是一个纯粹的 Power-Law 网络，但它具有 Power-Law 结构的优点和缺点。Gnutella 大致符合因子 $\alpha=2.3$ 的 Power-Law 分布，而研究者指出因子 $\alpha<3$ 的 Power-Law 网络面对随机结点失效的容错性是非常高的；同时，Gnutella 网络中结点又带有常数性质的分布，大多数结点连接数都高于某个常数值，这使得它对于恶意攻击的容错性也较高。就 Gnutella 网络而言，我们最不希望看到的是因为结点失效而导致它分割成几个互不相交的子网，这是容错性的极端要求。图 3.1.3 展示了 Gnutella 覆盖网在不同的结点失效情况下所呈现出的状态。



(a) 正常的Gnutella覆盖网　(b) 随机移去30%结点后的Gnutella覆盖网　(c) 移去连接数量多的4%结点后的Gnutella覆盖网(它分裂了)

图 3.1.3 Gnutella 覆盖网的几种状态
（图片来自[Saroiu et al.,2003]）

Gnutella协议设计了多种消息以完成不同的功能，PING、PONG消息的主要用途是检测其他结点是否在线以维护网络连接和保持自适应，它们是辅助性的。然而，图3.1.4显示在早期的Gnutella中，仅仅PING消息在所有消息中就占了超过50%，这说明早期Gnutella的自适应机制是低效的；另一方面，对用户真正有用的QUERY消息只占36%。这一问题后来得到了解决，新的Gnutella应用做了很好的完善，使得QUERY消息占据消息总数的份额超过90%。

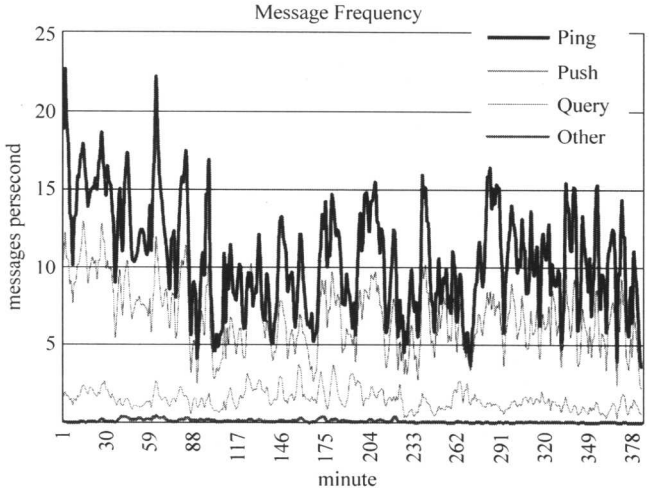


图3.1.4 早期Gnutella网络中消息分布(PING消息超过50%)
(图片来自[Ripeanu,2001])

拓扑一致性问题是几乎所有P2P网络的共同问题，Gnutella的覆盖网结构简单、松散，并且也没有采用什么机制来缩小覆盖网和物理网之间的差异，因此，Gnutella网络的拓扑一致性较低，这对它的工作效率也产生了不小的影响。

基于上面的性能分析，采用Gnutella协议的P2P网络在设计、优化时必须考虑下面几个问题：对结点异构性，系统应该有机制针对不同的结点能力采取不同的措施，让它们扮演不同的角色；对Free-Riding现象，系统必须有方法鼓励上传，有限制或剥夺Free-Rider的权利；为了保持高容错性，系统应当有高效的机制检测和恢复网络分割。虽然QUERY消息已经成为Gnutella网络的消息主体，但是查询机制的优化、TTL的取值，仍然是重要的研究课题；对于拓扑一致性问题，Gnutella也应当设计相应的方法来缩小覆盖网与物理网间的差异。

3.1.4 Napster与Gnutella的比较

1. Napster、Gnutella的共同点

(1) Napster、Gnutella分别是第一代、第二代P2P网络最好的代表，也是整个

P2P 领域的先驱。它们的开发有着相似的目的：使得接入因特网的广大用户之间查询文件、交流文件更加便利。

(2) 在两个系统中，文件都存储在用户计算机中，用户之间建立直接的连接来传递数据，文件交换采用 HTTP 风格的协议。

(3) 系统里所有的对等实体(Peer)之间是对称的：在功能上既是客户又是服务器，既提供下载又提供上传，正是这种对称性将 Napster 和 Gnutella 区别于它们之前的文件交换系统。

(4) 两者的可扩展性都不高，不过原因不同：Napster 是因为采用 C/S 结构的集中索引机制造成了网络瓶颈，Gnutella 则是因为消息的洪泛式广播造成了系统的巨大开销。

(5) Napster 和 Gnutella 网络中都存在普遍的结点异构性和 Free Riding 现象。对于异构性，两者都没有做有效的开发；对于 Free Riding 现象，两者都没有鼓励或惩罚的有效方法。

2. Napster、Gnutella 的不同点

(1) Napster 与 Gnutella 最大的不同在于前者有服务器而后者没有，所以两者网络结构的组织与查询、回复的机制完全不同。

(2) 就网络结构而言，Napster 是混合式星形结构、集中式与分布式相结合的网络，它混合了 C/S 模式与 P2P 模式；而 Gnutella 是无结构的、纯分布式的网络，它不带有 C/S 模式的残余，是纯粹意义上的 P2P 网络，并且 Gnutella 在因特网上构建了覆盖网.这是后来的 P2P 网络都会做的一项基础性工作。

(3) 就工作机制而言，Napster 中查询、回复和网络维护完全依靠服务器，是 C/S 的；而 Gnutella 则完全依靠 Peer 间的协作，是 P2P 的。在 Napster 中，只要共享文件存在一般就能查询到，而 Gnutella 则不然。

(4) 就容错性而言，Napster 只在服务器故障时出错，Gnutella 则可能因为结点信息过于陈旧导致的功能失效而出错，但最严重的问题是多个结点同时失效而导致网络分割成几个不相交的子网。

3.1.5 Gnutella 协议 0.6 版

Gnutella 协议 0.6 版于 2003 年发布，最先是作为一个建议性的测试版本，现在已为稳定版本。根据 Gnutella 协议 0.6 版构建的网络不再是纯分布式的，而是层次化的无结构 P2P 网络，相应地，查询算法、网络自组织算法等的主要机制也都发生了变化。Gnutella 协议 0.6 版将网络中的结点划分成两个层次：超结点(UltraPeer)和叶结点(LeafPeer)。超结点负责查询消息的路由，是整个 Gnutella 网络的骨干(backbone)；叶结点以超结点为代理连接到 Gnutella 网络。这种层次化的网络组织方式和下文中 KaZaA、eDonkey 的网络结构类似。

图3.1.5描绘了Gnutella协议0.6版构建的网络示例。网络中同时存在三类结点：①负责查询消息路由的超结点(UltraPeer)，这类结点在网络中占少数；②连接到超结点上、能力较弱的叶结点(LeafPeer)，叶结点将自己共享的文件信息上传到与其连接的超结点上，不参与查询消息的转发；③尚未实现超结点功能的遗产结点(LegacyPeer)，这类结点不能和叶结点相连，只能与网络中的超结点建立连接，它们可以看作旧版协议留下来的“遗产”。

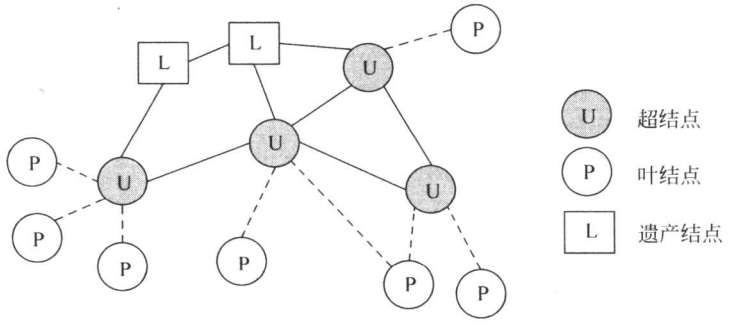


图3.1.5 基于Gnutella协议0.6版构建的网络

3.2 KaZaA——基于超结点的无结构P2P网络

3.2.1 KaZaA和FastTrack介绍

KaZaA是基于FastTrack协议的著名无结构P2P网络，诞生于2000年7月(也就是Gnutella出现后不久)，其公司创始人是来自斯堪的纳维亚的Niklas和他的丹麦伙伴Friis。Niklas是著名的P2P创业家，在KaZaA之后他相继创办了Joltid公司(开发推广P2P解决方案和P2P流量优化技术)、Altnet(世界上第一个完善而安全的应用性P2P网络，为消费者提供商业内容并发行数字版权管理许可证)、Skype公司(全球第一家P2P实时语音通信公司，本书5.3节将对Skype作详述)，并且都取得了成功。

与短命的Gnutella不同，KaZaA从产生到现在，其用户群不断扩大。KaZaA宣称它拥有超过300万的平均在线用户，共享超过5000TB的数据资源，可以说KaZaA构建了世界上最大的文件共享网络之一。

FastTrack是与Gnutella并列的典型的无结构P2P协议，它比Gnutella(0.4版)最大的改进在于引入“超结点”(SuperNode)，从而第一个很好地开发了P2P网络中的结点异构性；Gnutella(0.6版)虽然也引入了超结点，但比FastTrack要晚很多。基于FastTrack协议的应用除了KaZaA，还有KaZaA的类似体KaZaA-

Lite，以及 Grokster、iMesh 等。由于 KaZaA 是 FastTrack 协议最杰出的代表，下文将通过 KaZaA 使读者理解 FastTrack 协议，而不再对两者做明确的区分。

3.2.2 KaZaA 的工作原理

由于 KaZaA 是一个专有的协议，并且加密了其信号消息，因此对 KaZaA 的理解都是基于 P2P 研究者的测量和分析的结果（详见文献[Liang et al.,2004; 2005]）。

如图 3.2.1 所示，KaZaA 覆盖网是一个双层网络：上层由“超结点”（SuperNode，SN）组成，下层由“普通结点”（OrdinaryNode，ON）组成。P2P 研究者很早就指出 P2P 网络存在着极大的结点异构性问题，用户之间在带宽、处理能力、存储容量、NAT 访问方式（network address translation，即网络地址转换，如代理服务器、防火墙等）等方面存在着很大的差异性。KaZaA 是最先开发网络异构性的 P2P 网络，其方法就是将网络结点分成不同层次的两类：超结点、普通结点。超结点通常有高带宽、高处理能力、大存储容量、不受 NAT 限制，相比之下，普通结点通常有低带宽、低处理能力、小存储容量，常常受 NAT 限制。当普通结点加入 KaZaA 网络时，它会选择一个超结点作为其“父超结点”，并且与父超结点维持一条半永久的 TCP 连接，将它所共享的文件元数据信息（类似文件索引）上传给这个超结点。

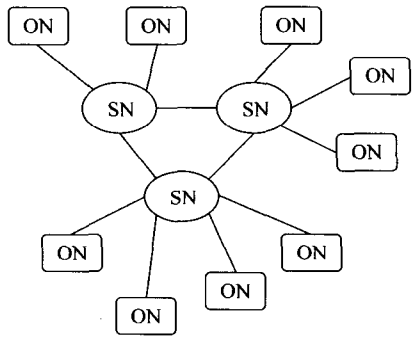


图 3.2.1 KaZaA 的双层覆盖网

KaZaA 采用“文件索引”将文件标识符映射到它所在的 IP 地址，这些文件索引分布在 KaZaA 的超结点中，每个超结点为其所有“子普通结点”共享的文件保存一个本地索引。从这里看 KaZaA 的超结点很像一台小的 Napster 服务器，但是与 Napster 不同的是，KaZaA 网络中的超结点不是专门的、永久的，它只是一个普通的拥有较高能力的网络结点而已，并且经常会由 KaZaA 普通结点转变而来。

每个普通结点向其父超结点上传自己所共享的文件元数据，元数据包括：文件名、文件大小、文件内容 Hash 值、文件描述符（如艺术家姓名、专辑名）等，其中文件描述符用来进行关键词匹配查询。文件内容 Hash 值的作用也很大：KaZaA 对每个文件生成它的 Hash 值，在 HTTP 风格的下载请求里 Hash 值是文件的唯一标识。如果从某个用户的下载失败了，文件内容 Hash 值使得 KaZaA 用户自动搜索特定的文件，而不必再做关键词查询。

当用户想要查询一个文件时，它向其父超结点发送带有文件关键词的查询消息，超结点在自己的数据库里寻找匹配的文件索引，返回给用户这些文件所在结点

的IP地址、服务端口号、文件元数据。每个超结点和其他的一些超结点间也保持着长期的TCP连接，从而构建了一个超结点覆盖网，当一个超结点收到查询消息时，它还会把这个消息发给与它连接的一些超结点，但查询只能发生在很小的一个超结点集里，因此只能返回与这个小的超结点集相连的所有普通结点共享的文件信息。从这点来看，KaZaA和Gnutella都具有文件查询的局部性。为了克服这个缺点，KaZaA采用了不断更新结点间连接关系的方法，下文将详细讲述。

3.2.3 KaZaA协议的语法和语义

KaZaA用户应当具有以下4个软件构件：

(1) KMD(KaZaA Media Desktop，KaZaA媒体桌面)。

(2) 存储在Windows注册表中的软件环境信息。软件环境信息包含一个有200个超结点信息(IP地址、端口等)的列表，称为“超结点列表缓存”。

(3) DBB文件。每个DBB文件包含用户希望共享的文件的元数据。当KaZaA软件运行以后，有一个进程永久地管理共享的用户本地文件夹，文件的增删都会在DBB文件中反映出来。

(4) DAT文件。每个DAT文件是一个部分下载的文件，一旦下载完成，DAT文件就被重命名成下载的原始文件。

每个KaZaA用户与网络中的其他用户之间进行4种不同的TCP通信：

(1) 信号通信。包括：为建立连接的握手通信，将DBB文件从普通结点上传到超结点，超结点列表更新、查询和回复。所有的信号通信都是加密的。

(2) 文件传输通信。这是指不经过中间的超结点、用户之间直接的文件传输。文件传输不加密，是以HTTP消息发送的。

(3) 商业广告。通过HTTP发送。

(4) 实时消息通信。采用Base64编码。

虽然KaZaA中普通结点和超结点之间、超结点和超结点之间的信号消息都被加密，但是著名的giFT项目(http://developer.berlios.de/projects/gift-fasttrack)反操作了KaZaA的加密算法，因此giFT-FastTrack的用户也可以从KaZaA网络中搜索和下载文件。另一方面，Sig2dat工具项目(http://www.geocities.com/vlaibb/tools.html)制造了一个工具可以获得KaZaA网络中任何文件的内容Hash值，这个工具被越来越多的KaZaA用户使用，将文件名和相应的Hash值放在网站或BBS上来帮助验证文件内容的正确性，从而有效地抵制文件污染攻击。

许多用户使用KaZaA-Lite软件来参与KaZaA网络，KaZaA-Lite与KaZaA软件类似，但是在查询过程中，KaZaA-Lite普通结点会将查询请求首先发给与它连接的超结点，在收到其父超结点返回的所有消息后，KaZaA-Lite普通结点自动断开与原来的父超结点的连接，同时连接到一个新的超结点，再向新的父超结点发

送查询请求。因此，在一次查询过程中，KaZaA-Lite 普通结点会连接到多个超结点，这大大增加了查询的回复内容，也很好地缓解了 3.2.2 节最后所说的查询局部性问题。

3.2.4 KaZaA 技术细节

1. 自适应

KaZaA 网络的自适应性是通过 KaZaA 结点之间频繁地交换超结点列表来保持的。每当一个普通结点连接到一个超结点，超结点立刻回送给这个普通结点一个超结点更新列表，包含 200 个超结点的 IP 地址、端口号、工作负载值，列表的第一项是发送列表的超结点自己。每当收到超结点更新列表时，普通结点会以超结点更新列表中的一些项替换掉它原有的超结点列表缓存中的项。另一方面，覆盖网中相邻的超结点之间也交换超结点更新列表。

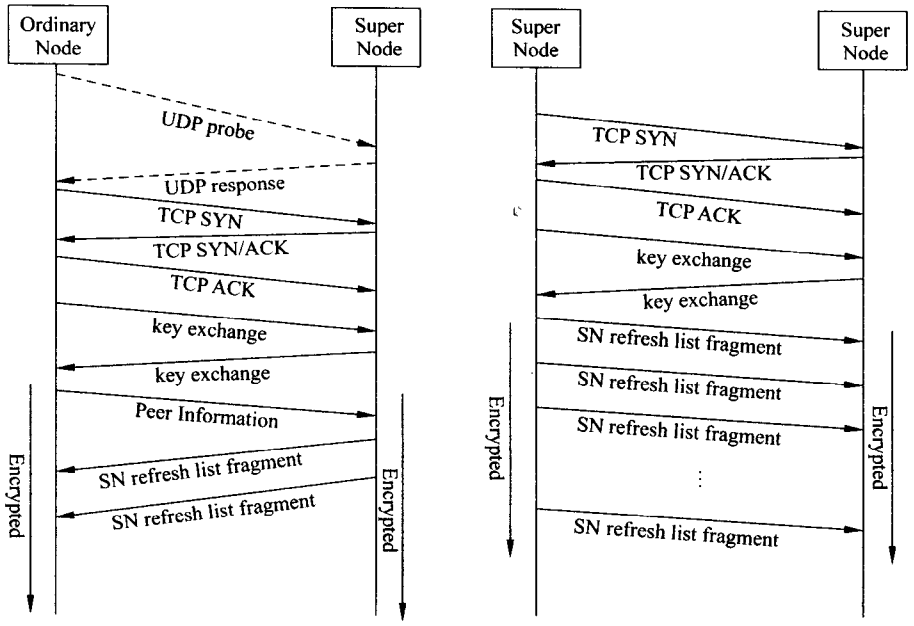
2. KaZaA 结点连接建立过程

当用户加入 KaZaA 网络时，它通常是一个普通结点，并且会得到一个原先保存的超结点列表缓存。它从缓存中选择一些候选超结点，给其中的每一个发送一个 UDP 数据包来探测(图 3.2.2(a)第 1 条，UDP probe)。用户将收到这些候选超结点中的某一些发来的 UDP 回复(图 3.2.2(a)第 2 条，UDP response)。对于每一个回复的候选超结点，普通结点都试图建立与它的 TCP 连接(图 3.2.2(a)第 3、4、5 条，TCP 三次握手)，对于每个这样的 TCP 连接，普通结点和超结点之间交换密钥(图 3.2.2(a)第 6、7 条，key exchange)，然后普通结点将用户信息发送给超结点(图 3.2.2(a)第 8 条，Peer Information)，超结点发给普通用户超结点更新列表(图 3.2.2(a)最下的两条，SN refresh list fragment)。用户信息包含用户的本地 IP 地址、端口号和用户名，注意是“本地 IP 地址”，所以当一个普通结点位于 NAT 之后时，本地 IP 地址是用户的私有地址，不同于经过 NAT 转换的地址，这一点对于 KaZaA 穿透防火墙或 NAT 的能力有重要意义，下文将会讲到。

做完上面的工作之后，普通结点会选择它认为最好的那个超结点建立连接，而不再和其他超结点连接，被选择的超结点就是它的“父超结点”。

3. KaZaA 的信号消息格式

如图 3.2.3 所示，KaZaA 用户之间发送的信号消息，由 5 字节的消息头和不定长的消息有效载荷(Payload)组成，每个消息以 1 字节的标识符(Packet ID)开头，后跟 2 字节的消息类型(Message Type)和 2 字节的有效载荷长度(Payload Length)，最后的 Payload 是真正的消息内容。



(a) 普通结点到超结点连接建立过程　　(b) 超结点之间连接建立过程

图 3.2.2　KaZaA 结点的连接

（图片来自[Liang et al.,2005]）

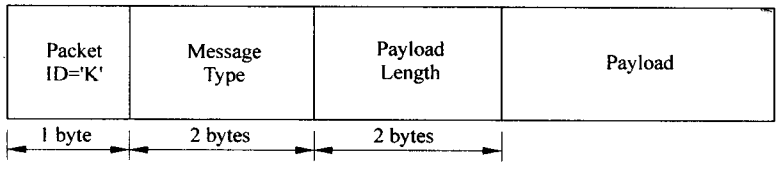


图 3.2.3　KaZaA 的信号消息格式

（图片来自[Liang et al.,2005]）

4. KaZaA 的防火墙和 NAT 穿透

早期的 KaZaA 软件使用默认端口 1214，因此防火墙可以很简单地封闭该端口以阻止 KaZaA 通信；后来的 KaZaA 软件都采用动态端口以穿透防火墙。每个用户随机选择他自己的端口，并把该端口告诉其他用户；更进一步，在用户之间发送的超结点更新列表中也包含超结点的端口。

NAT 穿透要复杂一些。如果用户 A 想从用户 B 那里下载文件，但 B 在 NAT 之后，那么 A 不能向 B 直接建立 TCP 连接（因为 A 发给 B 的连接请求将被 B 的

NAT 阻止），此时 KaZaA 的双层结构提供了一种方法部分地解决了这个问题。上述情况下，A 不是直接发送请求给 B，而是发送请求给 B 的父超结点 S，由 S 发送消息给 B，告诉 B 应该由 B 发出到 A 的连接请求、主动建立一条到 A 的 TCP 连接，然后 A 通过该连接从 B 下载文件。这种通过一个已经和 NAT 之后的结点建立连接的中间结点来建立结点间连接的技术，称为“连接反转”（connection reversal），它和 Internet 上俗称的“UDP 打洞”的技术原理相同。

3.2.5 KaZaA 性能分析

下文中对于 KaZaA 的性能分析均基于文献[Liang et al.,2004; 2005]的测量结果。测量结果显示：KaZaA 网络的超结点数平均在 25 000～40 000 之间，而超结点之间的连接是松散的，每个超结点平均与覆盖网中 40～60 个超结点建立连接，每个超结点平均连接 60～150 个普通结点。

（1）KaZaA 覆盖网动态性

KaZaA 覆盖网动态性通过连接保持时间来衡量，普通结点与超结点之间的连接保持时间平均在 34min，其中约 38%低于 30min；而超结点之间的连接平均在 11min，其中约 32%低于 30min。为什么连接时间如此短暂？用户本身离开 KaZaA 网是一方面，KaZaA 自适应过程中主动改变连接是更重要的原因：超结点之间的连接平均只有 11min，主要是因为超结点间频繁交换超结点列表、频繁改变邻居造成的。

（2）KaZaA 网络局部性

虽然没有办法得知 KaZaA 网络是否采用了提高局部性的方法，但是测量结果显示：当一个普通结点选择其父超结点，或者超结点间选择邻居时，确实考虑到了局部性因素。观察发现约 60%的超结点之间的连接 RTT（round trip time，往返时间）小于 50ms，40%的普通结点和超结点之间的连接 RTT 小于 5ms。另一方面，超结点返回给其子普通结点的超结点列表中，很高比例的超结点和这个普通结点有相似的 IP 前缀，可以推测这种现象不是偶然的，而是该普通结点的父超结点故意为之以增加局部性。

（3）KaZaA 索引管理

测量结果显示：KaZaA 网络中，约 13%的普通结点上传了超过 80%的元数据，说明索引上传负载并不平衡。另一方面测量结果也显示，虽然超结点间互相发送超结点更新列表，但它们之间不会交换索引信息，索引信息（元数据等）只在普通结点和其父超结点间传送。

3.2.6 KaZaA 网络总结

（1）KaZaA 是第一个显式开发网络结点异构性的 P2P 系统，它是通过将网络结点组织成两类——超结点和普通结点来开发这种异构性的，这是 KaZaA 网络最

大的特点。

(2) 为了缓解无结构P2P网络的查询局部性问题，并保持KaZaA网络的自适应性，KaZaA用户之间频繁地交换超结点更新列表，根据列表改变原有连接，这对提高KaZaA网络的性能有重要意义。

(3) 虽然不能确证，但测量结果显示KaZaA网络运作确实考虑了局部性因素。Free-Riding现象在KaZaA网络中依然存在。

(4) KaZaA通过使用动态端口和连接反转的方法，有效地穿透防火墙和NAT，使得KaZaA的适用范围大大拓宽。

3.3 eDonkey/eMule——分块下载的双层无结构P2P网络

3.3.1 eDonkey、eMule和Overnet介绍

eDonkey，中文名"电驴"，和Gnutella差不多同时出现(2000年)。eDonkey在很多方面与BitTorrent类似，比如文件分块[①]下载，从而每个文件可以从其他多个用户并行下载，使用文件内容散列值来验证数据完整性，以及采用"服务器"作为系统的核心。不同的是，BitTorrent中的服务器用来查询、搜索、跟踪用户，是基于文件的、真正意义上的服务器，而eDonkey中"服务器"的作用更像KaZaA中的"超结点"，是基于用户的。同时，eDonkey服务器之间构成的网络类似KaZaA的超结点网络，都是无结构的，这是本书将eDonkey/eMule归入第二代无结构P2P体系而非第一代混合式P2P体系的原因。eDonkey网站：http://www.edonkey2000.com。

Overnet是eDonkey所使用的分布式搜索网络，它本身是一个独立的应用，但eDonkey将其整合到自己的体系中。

eMule，这个名称"电骡"表明了它和eDonkey的关系，eMule是后继者，但更出色。2002年5月13日一个叫Merkur的人，由于不满意当时的eDonkey2000客户端软件，并且坚信他能做出更出色的类似客户端，于是便着手开发。他聚集了一批原本在其他领域有出色发挥的程序员到他的周围。eMule工程开始，其目标是保留eDonkey的优点和精华，加入新的功能以及优化图形界面。他们没有想到的是，不久以后eMule成了一个著名的P2P文件共享客户端软件，其流行性甚至远超过它的前驱eDonkey。今天，eDonkey、eMule所传输的网络通信量仅次于BitTorrent，显示了eDonkey类P2P体系良好的特性。eMule中文站：http://www.emule.org.cn。

---

① BitTorrent中称为"分片"，片：Piece，块：Chunk。

3.3.2 eDonkey 工作原理

eDonkey 网络由两部分组成，如图 3.3.1 所示：服务器层＋客户层。客户(图中 client)加入 eDonkey 网络以获取或者共享文件，服务器(图中 Server)的作用则是提供文件索引信息和服务器列表，它不传递实际的文件数据，每个客户都有可能成为服务器。每个客户连接到一个服务器进行文件查询或者服务器列表更新，服务器之间自组织成服务器层网络(图中 Server Layer)以交换文件索引和服务器列表信息。从上面的描述可以看出，eDonkey 网络非常像 KaZaA，本质上也是基于“超结点”开发 P2P 网络的结点异构性，只不过 eDonkey 中称“超结点”为“服务器”。

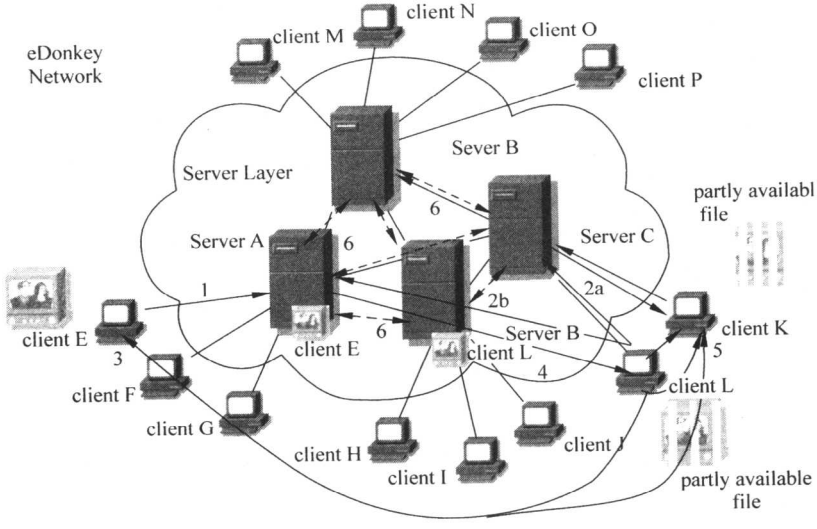


图 3.3.1 eDonkey 工作原理示意图

(图片来自[Tutschku,2004])

当 eDonkey 客户加入网络时，它首先要连接到某个服务器。这一般分两步完成：第一步，与过去保存的“入口服务器”列表中的服务器建立连接，从中选择离自己最近的那个(即时延最小)；第二步，通过所选的入口服务器获得一个普通服务器列表，从此表中选择最合适的服务器建立连接，同时断开与入口服务器的连接。

连接到某个服务器后，eDonkey 客户将他所共享的文件信息发送给服务器(图 3.3.1 中箭头 1)，服务器保存它所连接的所有客户的共享文件索引。如果客户要查询文件，他将查询请求发给服务器(图 3.3.1 中箭头 2a)，服务器返回相匹配的文件索引信息(文件名、文件大小、文件位置等)。客户也可以将该查询重新提交给另一个服务器(图中箭头 2b)，以获得更多的文件索引。在实现上，客户和服

务器间的通信的典型方式是采用4661端口上的TCP连接，而客户和客户间的通信采用4662端口上的TCP连接，而客户和服务器之间的额外通信，如深层查询，则采用4665端口上的UDP连接。

在自适应方面，鉴于P2P网络的动态性，每个下载者必须周期性地向上传者重发下载请求，周期一般为40s，否则原连接将被关闭。同时，eDonkey服务器之间周期性地交换服务器列表，不过这个周期通常比较长，eDonkey中服务器层的自适应更新非常像KaZaA中超结点层的超结点列表更新。

在eDonkey客户下载某个文件前，他首先通过查询获得文件提供者列表，但是由于种种原因，这个列表中的文件提供者很可能不能连接到，或者不能从他那里获得下载许可，因此该客户将向列表中的每个文件提供者请求“上传槽”(Upload Slot，图3.3.1中箭头3)。文件提供者在收到这样的请求后通常将它放进“上传队列”(Upload Queue)中的“等待列表”(图3.3.2中waiting list)，直到条件满足时才给该请求分配“上传槽”并将它放进“上传列表”(图3.3.2中upload list)，此时文件交换才真正开始：提供者向请求者发起建立TCP连接，决定要发送文件的哪些分块，然后发送数据。

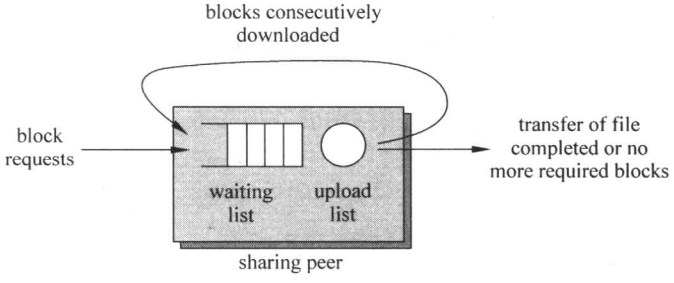


图3.3.2 eDonkey客户上传队列管理

(图片来自[Hoßfeld et al.,2004])

3.3.3 eDonkey文件分块

eDonkey协议将文件分成许多块，通常每块约10MB，只要某个客户收到一个完整的分块，它就可以与他人共享该文件(图3.3.1中箭头4)。通过分块，eDonkey实现了类似BitTorrent的“多源下载”机制(图3.3.1中箭头5)，这是eDonkey网络高效的重要原因。图3.3.3展示了分块的细节：文件首先被分块(chunk)，每块通常9500KB，最后一块通常小于这个值而被填充。每个分块分成多个片段(segment)，片段大小取决于“智能错误处理”(intelligent corruption handling，ICH)机制——该机制用来进行比分块更小的数据单元错误检查从而不必在检查出错误时丢弃整个分块。片段进一步分为小块(block)，每块通常

180KB，是 eDonkey 数据交换的最小单位。

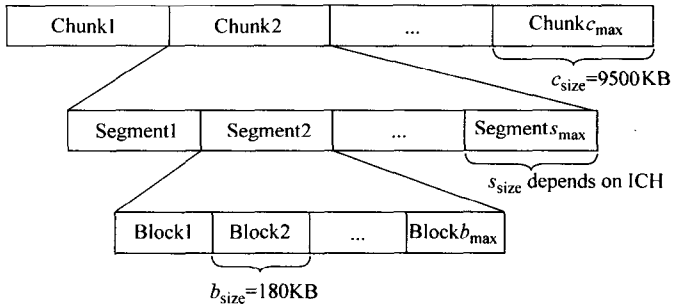


图 3.3.3 eDonkey 文件分块细节(Chunk，Segment，Block)
(图片来自[Hoßfeld et al.，2004])

3.3.4 eDonkey 性能分析

eDonkey 在很多地方类似 BitTorrent 或者 KaZaA，所以对于 eDonkey 的性能分析可以参考这两者，这里我们只做简单叙述。文献[Tutschku，2004]对 eDonkey 网络做了测量，下面的性能分析基于该文的测量结果，更细致的分析可以在文献[Tutschku，2004]和[Hoßfeld et al.，2004]中找到。

表 3.3.1 记录了测量环境和所观察到的各项参数：4662 端口 TCP 连接数(上文讲过 4662 端口是客户间的连接端口)、本地主机数、外部主机数、所有流传输总量、下载连接传输总量、下载连接数、入境下载连接数(从实验网外发起的 TCP 连接)、出境下载连接数(从实验网内发起的 TCP 连接)。

表 3.3.1 eDonkey 网络测量参数

Number of observed TCP connections on port "4662"	3 431 743	
Number of local hosts	25	
Number of foreign hosts	242 067	
Total transmitted volume in all flows	2.95×10^{11} bytes	
Total transmitted volume in download connections	2.08×10^{11} bytes	(70.5%)
Number of download connections	77 111	(2.24%)
Number of inbound download connections	21 344	(27.7%)
Number of outbound download connections	55 767	(72.3%)

这个表给我们最直观的印象是，下载连接传输总量占所有流传输总量的 70.5%，这个数值并不高，还有提高的可能。另一方面，在所建立的 TCP 连接中，只有 2.24%用来做下载连接，却承担了 70.5%的通信量，这很可能是 eDonkey 不及 BitTorrent 高效的重要原因。

表3.3.2记录了eDonkey通信量最大的7个自治系统(按地理位置)。

表3.3.2 eDonkey通信量测量记录，流量单位均为MB

Owner	Country	AS num	Total bytes	Bytes download		Bytes non-download	
				Outbound	Inbound	Outbound	Inbound
DTAG	.de	AS3320	50 258	15 890	4798	9231	20 331
Polish Tel.	.pl	AS5617	22 703	1761	574	9590	10 777
France Tel.	.fr	AS3215	10 527	2353	811	2910	4452
BTnet UK	.uk	AS2856	8992	1299	720	3197	3776
Verizon	.us	AS19262	6395	0.877	0.001	3196	3197
Arcor IP	.de	AS3209	5579	1133	415	1656	2373
NTL Grp. Ltd	.uk	AS5089	5224	1055	322	1557	2289

3.3.5 eDonkey网络总结

(1) eDonkey网络由两部分组成：服务器层＋客户层。客户加入eDonkey网络以获取或者共享文件，服务器则提供文件索引和交换服务器列表，其结构非常像KaZaA，目的是开发P2P网络中的结点异构性。

(2) eDonkey将文件分块，分块又分成片段，片段进一步分成小块，从而提供了类似BitTorrent的多源文件下载机制，并且对文件的完整性检查也有了更细的粒度。

(3) 为了适应动态的网络环境，eDonkey客户通过上传队列来管理客户间的连接，只有获得“上传槽”的连接才能真正传递数据，并且每个连接都被周期性地检测是否不断有请求发来，如果没有则关闭。队列机制对eDonkey高效工作很有帮助。

(4) eDonkey的自适应性是靠服务器之间周期性的更新服务器列表来维护的，虽然更新周期很长，但eDonkey在世界范围内的流行说明它的自适应性仍然是良好的。

3.4 Freenet——自由、安全、匿名的无结构P2P网络

3.4.1 Freenet出现的背景和发展历史

Freenet(自由网)的概念最初是在1999年提出的，当时还在苏格兰的爱丁堡大学读书的Clarke将这个想法发布到了网上，马上就迎来了不少的支持者。2000年3月Freenet的第一个版本推出，不过直到2001年8月才有第二次的更新，而随着网络泡沫的消退，Freenet似乎从那时开始就销声匿迹不为人知了，直到后来

的 Freenet 0.5 终于让人们看到 30 位开发者并没有放弃。下文对 Freenet 的讲解都基于 Clarke 的论文[Clarke et al.,2000]。

Freenet 从一开始就有着很不同的理念，Napster、Gnutella 和 KaZaA 这类 P2P 网络的主要目的在于交换文件，而 Freenet 的目的是共享 Internet 计算机资源，组建一个自由、安全、匿名的信息发布和获取的平台。Freenet 利用每个参加者作为一个结点，结点上的计算机划出一部分硬盘作为公用存储空间，在其中存放某个文件的某些数据，不过是全部经过加密的，所以即使是硬盘的主人也无法知道其中到底存放了什么内容。同时，不依赖于中央服务器的 Freenet 理论上能够不受限制地存在，而加密传输更让参加者的身份得以完全的保护，这也就是 P2P 网络从一开始就想要的用户“匿名”。

Freenet 匿名性的本质在于它安全、匿名的路由。在 Freenet 中，文件查询请求通过一条“代理结点链”从前一个结点传到后一个结点，路径上的每个结点只知道该请求的前一跳结点和由它决定的后一跳结点，对路径上的其他结点一无所知，这在网络领域也经常被称为“隧道路由”，是具有高安全性的路由方法。

Freenet 能够自适应地将使用到的文件存放到更接近使用者的结点上，也就是让网上的内容自动进行复制。这里有个 Clarke 经常讲的例子：“信息数据经常是在美国生产，在欧洲应用。使用 WWW 网，当欧洲要使用它时，必须每次都穿过大西洋，而使用 Freenet，这个文件将在第一次时穿过大西洋，而以后的欧洲用户请求就不会这么远了，他只需要到最近的欧洲结点上去读取它！”如果有很多的用户需要它，它会多次复制，分布到整个自由网中那些受欢迎的地区，这种效应 Clarke 称为“The Web's Slashdot effect”。当一段时间内大量用户同时需要某个 Web 网站的网页时，服务器处理速度通常会降低，而自由网却能够有效地利用网上无数用户的带宽。一旦文件被索取，就会在那里建立该文件的一个镜像（复本），从而充分利用带宽，提升读取速度，通常用户越多，反而速度会越快（这一点很像 BitTorrent 的宣传，但本质原因不同）。另一方面，Freenet 的复制过程通常是用户所不能感知的，即使用户有所察觉，他也不可能清楚地知道复制发生的过程，这在另一个层次上提供了 P2P 网络的数据对象匿名性。

由上所述可以看出，Freenet 更加关注的不是让用户传播受版权保护的音乐或者电影，而是让人们可以不受任何阻碍，隐蔽、自由地发布自己的观点，虽然同时也不能阻止那些有其他企图的人，利用 Freenet 从事一些不太光明的勾当。正是因为这一点，在很多国家 Freenet 已经成为一个具有杀伤力的危险关键词（见图 3.4.1），但还是

图 3.4.1 Freenet 的标志——红色小兔子
（图片来自已被关闭的 Freenet 网站）

有很多自由主义者在坚持他们的信念传播和使用它，实际上 Freenet 已经有了汉化版本。本书对 Freenet 的讲解只是出于研究计算机网络的学术目的，至于自由主义的哲学和信念，我们不作评判。在技术上，我们倾向于把 Freenet 比作一个锤子：它只是一个工具，至于到底是拿它来砸石头还是砸人，在于使用它的人是怎么想的。

Freenet 虽然有着强大的功能，但是一直以来都是技术界人士的特权，非常不友好的界面限制了使用它的人群。新版本的 Freenet 0.5 对此进行了很大的改进，改善了用户操作界面，努力让除了技术人员以外的用户能够方便地使用，如图 3.4.2 所示。

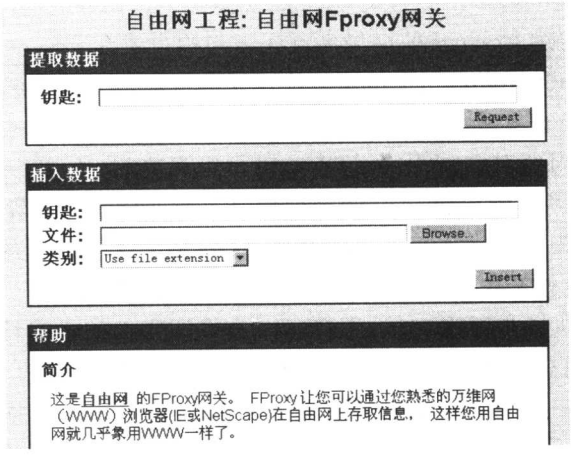


图 3.4.2 Freenet 软件界面(此软件已不可用)

像其他的 P2P 软件一样，Freenet 也曾经努力进入企业应用市场，出于此目的 Clarke 在 2000 年就创立了 Uprizer 公司，但随着新版本的 Freenet 出现，他离开这家公司重新建立了名为 Cematics 的公司。另外，作为 P2P 软件，越来越敏感的版权问题也同样缠绕在 Freenet 周围，可是商业问题和法律问题看来都不是 Clarke 关心的话题，他认为："归根结底，自由言论比现在的版权法重要得多。"

3.4.2 Freenet 的密码学基础

Freenet 涉及到很深的密码学知识。这一小节对 Freenet 密码学基础的讲述，只是为了让读者能大概明白这些密码学设施是做什么用的、对 Freenet 工作有什么意义，并不要求读者真正掌握它的原理，因为这里所提到的"对称加密"、"非对称加密"、"密钥"、"公钥"、"私钥"、"安全散列函数"、"数字签名"等都是密码学的专业术语，如果要在这里讲清楚它们可能需要的篇幅比讲 Freenet 还多。当然，如果读

者已掌握上述的密码学知识，下面的讲述可以让你更为深刻地理解 Freenet 的工作原理。

Freenet 中的文件是以其 Hash 值作为标识的，通常使用 SHA-1 安全散列函数来生成(SHA-1 散列值有 160bit，实际上目前它已不再安全，新的 SHA 算法已经出现，但这与我们所要讲的没有什么关系)。Freenet 使用三种不同类型的文件标识，分别完成不同的功能，它们是 Freenet 的核心设施。三种标识是互相独立的，但有时也可以合起来用。

1. KSK

KSK(keyword-signed key，关键词签名标识)是 Freenet 中最简单的文件标识，它的作用是构建 Freenet 的全局名空间。当用户要在 Freenet 网络中存储文件时，他必须给这个文件指定一个很短的描述性字符串，比如“pop stars / Chow Yun Fat / the Making of a Killer”。这个字符串有很重要的作用：一方面，该字符串作为输入来产生一对非对称密钥(公钥＋私钥)，公钥被 Hash 以产生文件标识，私钥用来给文件做数字签名，签名本身附带地提供了对文件的完整性检查；另一方面，字符串本身被作为密钥来加密原来的文件，这就是上文所提到的为什么即使拥有该文件的人(不包括文件的发布者)也不知道文件内容的原因。

为了让其他人获取某个文件，用户只需要将该文件的描述性字符串发布出来(它作为解密密钥总是和文件标识一起发送)，这也使得文件的 KSK 易于被其他人记住和交流。一个容易看出的问题是：KSK 所依赖的文件描述字符串是用户独立指定的，因此不同的用户很可能会碰巧选择相同的字符串从而产生相同的文件标识，这个问题是由下面所要讲的 SSK 来解决的。

2. SSK

SSK(signed-subspace key，签名子空间标识)的作用是构建 Freenet 的个人名空间，用户通过随机地产生一对非对称密钥(公钥＋私钥)来标识他自己的名空间，为了与 KSK 中的公钥/私钥区分，我们称 SSK 中的公钥/私钥为“名空间公钥/名空间私钥”。有了 SSK 以后，当用户要在 Freenet 网络中存储一个文件时，首先他要指定一个很短的描述性字符串，这与 KSK 的介绍相同；但随后该字符串和名空间公钥分别被 Hash，再将两个 Hash 值异或，得到的异或值再次 Hash 产生文件标识。比较 KSK 和 SSK 产生文件标识的不同就会发现，SSK 方法即使用户选择相同的字符串也几乎不可能产生相同的文件标识，这就解决了 KSK 中的问题。

与 KSK 类似，名空间私钥被用来给文件签名。由于名空间私钥是随机产生的，因此它比 KSK 中由描述性字符串产生的私钥安全得多。与 KSK 类似，SSK 也使用这个字符串作为密钥来加密原来的文件，原因与 KSK 相同。

为了让其他人获取某个文件，同样，用户只需要将该文件的描述性字符串公布出来（它作为解密密钥总是和文件标识一起发送），但是再也不会出现文件标识相同的情形了。

用户可以通过SSK管理他自己的名空间，他可以通过创建类似目录结构的文件来模拟一个层次化的文件结构，其中每个文件包含指向其他文件的指针。举例来说，用户可以使用“pop stars”作为密钥存储一个文件根目录，这个目录里包含一个密钥列表：“pop stars / Chow Yun Fat”、“pop stars / Fayewang”、……，分别指向下一级的文件，下一级的文件又可以这样递归地指下去。

3. CHK

CHK（content-hash key，内容散列标识）的作用是实现文件的更新和分割。将文件直接Hash得到的散列值就是它的CHK，因此CHK也是一种文件标识。与KSK、SSK不同的是，CHK使用一个随机产生的密钥来加密原来的文件，同样，为了让其他人获得文件，用户只需要将解密密钥和文件标识一起发布出来。

CHK和SSK一起使用通常更有意义。为了在Freenet网络中存储一个可更新的文件，用户首先使用CHK来存储它，然后使用SSK再存储一个间接文件，此文件的内容正是CHK。间接文件相当于一个指向真实文件的指针，这使得其他用户在得到文件的SSK以后可以分两步获得文件：先由SSK获得间接文件，再由间接文件里的CHK获得真正想要的文件。

过了一段时间用户有了新的文件（当然CHK也更新了），为了替换原来存储的文件，他首先使用新的CHK来存储新文件（这不会与旧文件冲突），然后使用原来的SSK再存储一个新的间接文件（其文件内容是新的CHK，因此它指向新文件）。因为SSK不变，在插入新的间接文件的过程中，通常会到达一个结点，在那里碰见了旧的间接文件从而导致文件标识SSK冲突，结点将检查新的间接文件的SSK签名，如果它确认该签名合法并且是更新的，它将以新的间接文件替代旧的间接文件。这样，其他用户根据原来他所知的文件SSK就可以分两步获得新文件，也可以根据原来所知的旧文件CHK直接获得旧文件（当然，这些旧文件最终可能会被清除）。综上所述，Freenet的文件更新，其实更新的不是文件，而是文件指针——间接文件，新旧文件可以在网络中共存，但间接文件只有一份。

CHK还可以用来将文件分割成多个部分。对大文件而言，分割可以减少对单个结点存储容量和带宽的需求；分割还可以将文件分成多个标准大小的部分以对抗通信量分析。文件分割是这样做的：首先用户将文件分割成多个部分，然后每个部分使用它自己的CHK来存储，最后存储一个间接文件（或者多层的间接文件）来指向文件的各个部分。

3.4.3 Freenet 中数据的查询与获取

为了获取一个文件，用户首先必须获得或者计算出它的文件标识（上节讲的三种文件标识中的一种或几种），同时设置好请求消息的跳数限制（hops-to-live，类似于 Gnutella 的 TTL），然后发送文件请求消息到 Freenet 网中。当一个结点收到请求消息时，首先它检查自己是否拥有该文件，如果有就回送消息，如果没有它会在自己的路由表里查找与该文件标识最接近的文件标识并把请求消息发送给与之相应的结点。如果最终查询成功文件被回送，这个结点除了将文件回传给请求者（注意：它只是回传文件给下一跳，并不知道文件的真正请求者，这里的“请求者”是安全路由所虚拟的），还会在它自己的数据库里缓存这个文件（上文所说的复制），并且在自己的路由表里创建一个新项指向该文件的数据源（注意：这里的文件源也是虚拟的，不一定是真正的文件源）。因此，如果以后该结点再收到对此文件的请求，那么它可以立刻返回自己缓存的文件；如果收到与此文件标识相近的文件标识，它会把请求消息传给路由表里所记录的文件源。从上面的讲解可以看出，Freenet 采用的是一种基于文件标识的“导向路由”方法。

文件缓存和自适应更新的路由表虽然提高了 Freenet 的工作效率，但是也潜在地带来了对安全性和匿名性的威胁，因此 Freenet 同意消息传送路径上的每个结点都可以单方面修改消息，将消息原来的“请求者”或者“文件源”改成它自己或者任意一个其他结点。这是 Freenet 安全路由的核心，也是其匿名性的关键所在。

如果某个结点不能将请求消息发送给它所选择的最好结点（这可能因为该结点此时不在线，或者是如果传送给它会导致循环路径），它将尝试它认为第二好的结点，如果这个结点再不行，再尝试第三好、第四好、……。如果所有的尝试都失败了，就回送失败消息给自己的前一跳，由前一跳结点来尝试新的下一跳，实际上，这就是计算机算法中经常讲的图的“深度优先”搜索。在路由过程中，如果跳数限制到了，结点也不会再做尝试，而只是回送失败消息给自己的前一跳。对于跳数限制有一点补充，就是在路由过程中每个结点都可以单方面减少跳数限制以减轻网络负担，甚至可以直接丢弃某个消息以释放存储容量。

图 3.4.3 描述了一次典型的 Freenet 数据查询和获取的消息路由过程。结点 a 发出对 data 的请求，此消息被发送给 b（箭头 1），b 又把它发送给 c（箭头 2）。由于 c 没有可尝试的下一跳所以只能回送失败消息给 b（箭头 3），b 做第二次尝试将消息发给 e（箭头 4），e 又发给 f（箭头 5）。f 将消息发给 b（箭头 6），但 b 检测到了循环，因此回送失败消息给 f（箭头 7），f 没有别的下一跳可以尝试只能回送失败消息给 e（箭头 8）。e 做第二次尝试将消息送给 d（箭头 9），而 d 正好拥有 data，它将 data 回送给 e（箭头 10），e 缓存 data 并回送给 b（箭头 11），b 缓存 data 并回送给 a（箭头 12），a 得到了它所要的 data，此次数据查询和获取成功完成。

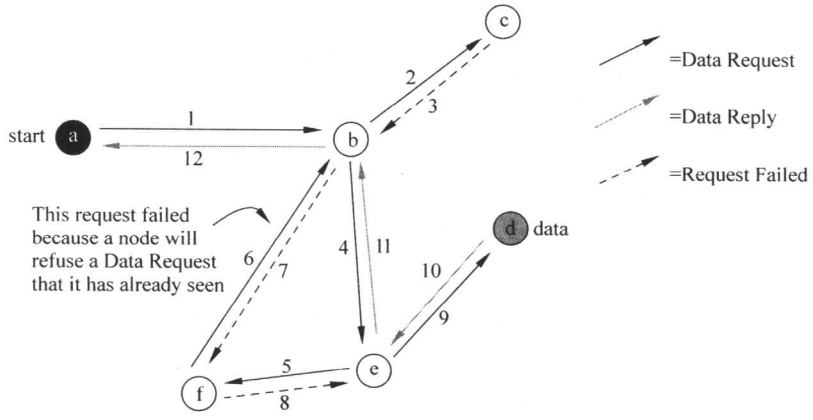


图 3.4.3 一次典型的 Freenet 数据查询和获取的消息路由过程(简化版)

(图片来自[Clarke et al.,2000])

上面所讲的 Freenet 的数据查询和获取机制，对于 Freenet 的工作效率有非常重要的影响。如果一个结点在其他结点的路由表中被列为是与某个文件相关的结点，那么它很可能会收到越来越多的对于那个文件的请求或者与那个文件相近的文件请求，因此它对于这类请求会越来越有“经验”，因为它既缓存了那些文件，又在自己的路由表中保存了大量的与那类文件相关的更好结点的信息，这称为 Freenet 的“标识集群”效应。随着时间的推移，这种“标识集群”效应会越来越明显：文件被复制得更多更优化，结点路由表表项也更多更“近”，使得 Freenet 网络越来越高效。

3.4.4 Freenet 中数据的存储与管理

为了在 Freenet 中存储数据，用户首先要使用文件标识中的一种(KSK，SSK，CHK)给文件计算标识，同时设置好请求消息的跳数限制，然后发送文件插入消息到 Freenet 网中。当一个结点收到文件插入消息时，它首先检查自己的数据库中是否有该文件标识，如果有就回送此文件，想插入文件的用户在收到此文件后就可以知道到底是该文件确实已经存在还是文件标识碰巧冲突了。如果是前者用户不需要再做插入，如果是后者用户将重新计算文件标识再做插入。如果收到插入消息的结点没有该文件标识，它将搜索自己的路由表中最近的文件标识项，将插入消息路由到该项指向的结点。

当已达到跳数限制(hops-to-live)并且没有检测到文件标识冲突时，“一切顺利”的消息将被回送给想插入文件的结点，这样用户就可以插入文件了。因为在检测过程中已经确定了路径，因此文件将按照该预定路径传送，路径上每个结点也会

在其路由表中增加与此文件标识相关的项(同上节一样，该文件标识项所指向的结点也是虚拟的文件源)。

Freenet 的文件存储机制对其网络性能是很有意义的。首先，文件通常被存储到已经拥有类似文件的结点上(这里“类似文件”指与该文件在标识上相近的文件)，这加强了上文所说的 Freenet 的“标识集群”效应；其次，新结点可以通过插入文件作为它向网络其他结点宣布自己存在的辅助方法；再次，如果恶意的用户想通过插入与已有文件标识相同的垃圾文件来“排挤”原文件，在 Freenet 中他不但很难得逞，反而将原文件的信息在网络中传得更广。

所有的信息存储系统都必须合理地分配有限的存储容量。Freenet 用户可以选择将自己的存储容量分配多少给 Freenet 用。结点的存储区按照 LRU(least recently used，最近使用优先)的方式来管理，越近的数据越靠前，所以越不会被新数据替换。当用户要保存新文件但存储区不够时，最不近的文件会被替换掉，但是被替换掉的文件在路由表中对应的项还会保存一段时间，因为结点有可能不久还会去获取这份文件。同样，路由表的容量也是有限的，当表项过多时一样会有替换发生，但由于表项本身非常小，所以替换的周期会非常长。

Freenet 的存储替换策略固然高效地利用了有限的存储区，但也有一个缺点，它没有办法保证某个文件在网络中至少有一份拷贝存在，因为替换都是结点局部决定的，不会与别人协作。如果某个文件不幸被所有结点丢弃了，但以后又有人希望获取它或者因为历史原因它重新变得有价值，再想获得它就非常难了。

上文讲过在 Freenet 中即使某个结点拥有某个文件，只要它不是文件的真正存储者，它就不可能知道文件内容，因为 Freenet 中保存的文件都是加密的，这也正是 Freenet 安全性的核心所在。加密的目的除了数据的保密性需求，更重要的意义是：Freenet 用户可以否认对任何关于其数据库中文件内容的了解，因为他所知的仅仅是文件标识，对文件内容根本不可能知道。从这点上说，Freenet 将网络中所有用户的存储区组织成了一个巨大的分布式系统，并且这个系统对于每个用户来说都是透明和不可知的。

3.4.5 Freenet 网络新结点加入

无结构 P2P 网络中新结点通常是通过其他已经存在于网络中的结点来加入网络的，Freenet 也不例外。为了向网络其他结点宣布自己的到来，每个结点必须有自己的标识，但这个标识的选取却并不简单：一方面，为了正确、高效地路由，这个标识应该是全局一致的；另一方面，出于安全性的考虑，不能让任何结点通过其他结点的某些属性计算出结点标识，否则恶意结点就可以利用别人的标识来做违法的事。因此，P2P 网络中常用的指定结点 ID 的全局一致性方法 nodeID＝Hash(IP，…)对 Freenet 并不适用。

Freenet采用加密协议来满足上面所讲的对结点标识的需求。新加入网络的结点选择一个随机的"种子"并且发送一条布告消息给某个现存的网络结点，布告消息里包含它的地址、种子的Hash值和一个跳数限制。当某个结点收到新结点的布告消息后，它也产生一个随机的种子，将这粒种子与收到的布告消息中的Hash值异或，将得到的异或值再Hash产生一个"承诺"(commitment)，然后它从自己的路由表中随机选择一个结点发给它新的布告消息(新的布告消息中包含发送者的"承诺")，这个过程一直继续下去形成一条链，直到布告消息中的跳数限制用完为止，最后一个收到布告消息的结点只产生自己的种子，不再做其他的事。上面的过程结束以后，这条链中所有的结点都公布他们各自的种子，所有种子的异或值就是新结点的标识！

对"承诺"的检查使得链中的每个结点能确认其他结点所公布的种子是真实的，因此，Freenet所采用的链式方法实际上产生了一个全局一致的随机的结点标识，并且这个标识不可能被某个恶意加入者影响。链上的每个结点会将新结点的标识加入到自己的路由表中。

3.4.6 Freenet协议细节

Freenet协议是面向数据包的并且使用了相互独立的消息，每条消息包含一个事务ID，因此结点可以追踪插入、查询消息的状态。上述设计是为了使Freenet可以灵活地选择传输层：TCP、UDP或者其他传输层协议。Freenet结点地址由其传输层协议＋传输层标识组成，比如"tcp/192.168.1.1：19114"，经常改变地址的结点也可以使用虚拟地址。

Freenet事务通常以从一个结点发送给另一个结点的Request.Handshake消息开始，这条消息包含发送者的地址(注意这个地址很可能被改变，因此是虚拟消息源)。如果接收结点对消息有回复，它将回送Reply.Handshake消息，该消息包含有结点所理解的协议版本；一旦握手成功，连接可以持续好几个小时，在这段时间里两个结点之间随后的事务不需要再次握手。

每条消息包含一个随机产生的64bit的事务ID、一个跳数限制(hops-to-live)和一个深度计数器。虽然事务ID不能保证不冲突，但在某个结点集内发生的事务，ID冲突的概率是极小的。跳数限制由消息最初的发送者指定，通常每走一跳会被减1，但是为了减少攻击者通过跳数限制所能获得的信息，在跳数限制减少到1以后消息仍然有继续往前传的概率(跳数限制一直为1)。消息每走一跳深度计数器通常加1，回复结点使用它来为回复消息设定合理的跳数限制。与跳数限制类似，为了减少攻击者通过深度计数器所能获得的信息，深度计数器为1的消息也会有继续往前传的概率但深度值一直保持为1。从上面可以看出，Freenet对跳数限制、深度计数器做了特殊处理，以增加其路由的安全性。

为了获取数据，用户发送 Request. Data 消息，它包含事务 ID、跳数限制、深度计数器和查询关键码，同时该用户会开启一个定时器并根据跳数限制来设定时值，如果定时值用完还是没有收到任何回复，就认为查询失败。当请求数据的消息被处理时，远处的结点可能会周期性地回发 Reply. Restart 消息，告诉消息的发送者消息因为等待的原因被延迟，从而发送者可以增加原先的定时值以容忍延迟。

如果请求成功，提供数据的结点会回送 Send. Data 消息，包含请求者所要的数据和数据提供者的地址（同样，这个地址很可能被改变因此也是虚拟的）。如果最终请求不成功，并且跳数限制也用完了，提供数据的结点会回送 Reply. NotFound 消息。如果请求不成功而跳数限制没有用完，这通常是因为消息走进了死角，找不到可行的非循环的路径，此时远处的结点将回送 Request. Continue 消息，包含剩下的跳数，请求消息的发送者收到此消息后将尝试新的下一跳。

为了存储数据，用户发送 Request. Insert 消息，它包含事务 ID、跳数限制、深度计数器和“建议”关键码（用户建议采用的文件标识），至于计时器以及 Reply. Restart 消息的使用，与获取数据时是类似的。如果插入导致文件标识冲突，远处结点可能返回 Send. Data 消息或者 Reply. NotFound 消息，前者返回现存的文件，后者是因为现存文件并没有找到但是路由表有所指向。如果插入消息没有碰到文件标识冲突，但在跳数限制用完之前就跑完了所有的结点，远处结点将回送 Request. Continue 消息，这代表插入失败，因为不能联系到足够多的结点。如果插入消息用完跳数限制并且都没有碰到文件标识冲突，远处结点将返回 Reply. Insert 消息，表示插入可以进行，收到该消息后文件插入者发送 Send. Insert 消息，包含要插入的文件，同前面所述，插入路径上的结点都会对此文件缓存。

3.4.7 Freenet 性能分析

1. 路由效率

由于 Freenet 中每个结点的文件缓存、路由表都是逐步构造出来的，因此一开始 Freenet 的路由效率会很低，路由路径跳数可能会非常大；但是随着时间的推移，Freenet 的文件缓存、路由表会越来越充实、完善，使得整个网络很好地组织起来，路由效率越来越高，最后趋于平稳。图 3.4.4 展示了这一过程（注意纵坐标是 log 型的）。

2. 可扩展性

P2P 网络中决定可扩展性的最重要因素是路由跳数 h 与网络结点总数 N 之间的关系，如果它们之间是对数关系，那么这样的 P2P 网络通常是高可扩展的。

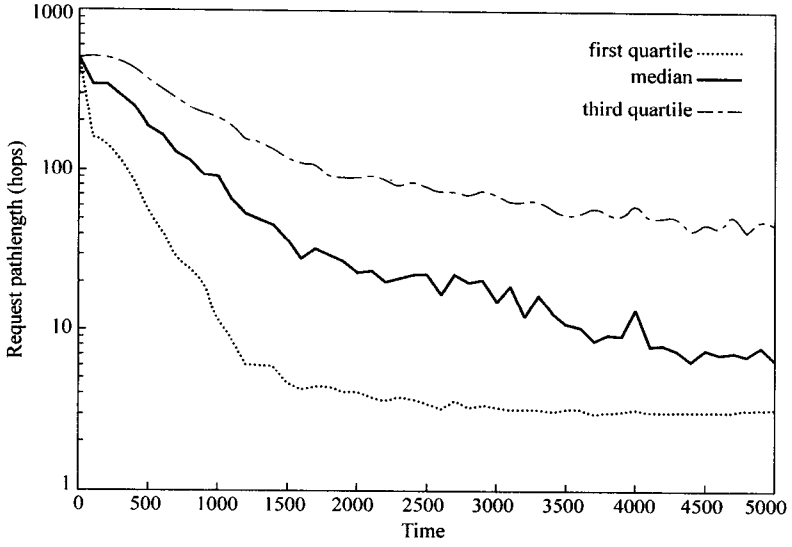


图 3.4.4 随着时间的推移 Freenet 的平均路由路径跳数变化

（图片来自[Clarke et al.,2000]）

图 3.4.5 展现了 Freenet 中路由路径跳数（稳定以后）与网络规模的关系，大致看来确实是服从对数关系的：路由路径跳数 $h \approx O(\log N)$。

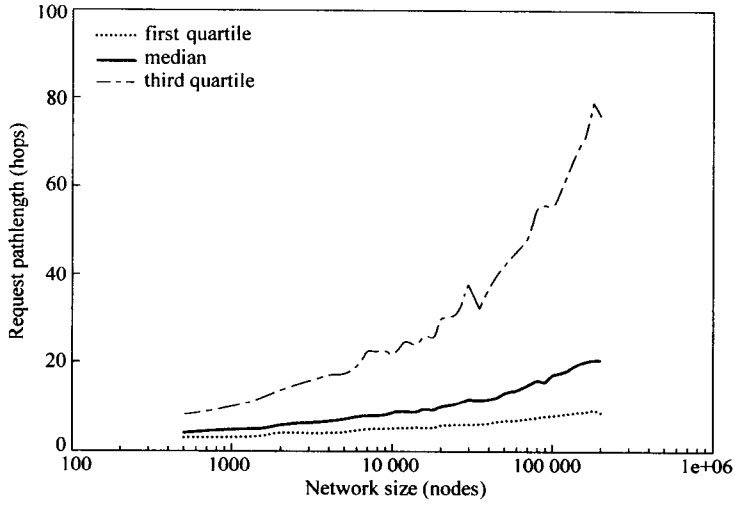


图 3.4.5 Freenet 中路由路径跳数与网络结点总数的关系

（图片来自[Clarke et al.,2000]）

3. Freenet 网络模型

通常无结构 P2P 网络符合小世界模型或者幂律模型。小世界模型(small-world model)是指网络中任意两个结点间的距离(即路由跳数)一般很短，并且对每个结点而言，其邻居结点相识(即互相连接)的概率很高，所以结点集群现象明显。幂律模型(power-law model)在 3.1.3 节中介绍过。从图 3.4.6 可以看出 Freenet 的结点连接数分布近似于幂律模型，同时也是符合小世界模型的(集群现象图中看不出来，不过从上文所讲的 Freenet“文件标识集群”效应可以推知)。

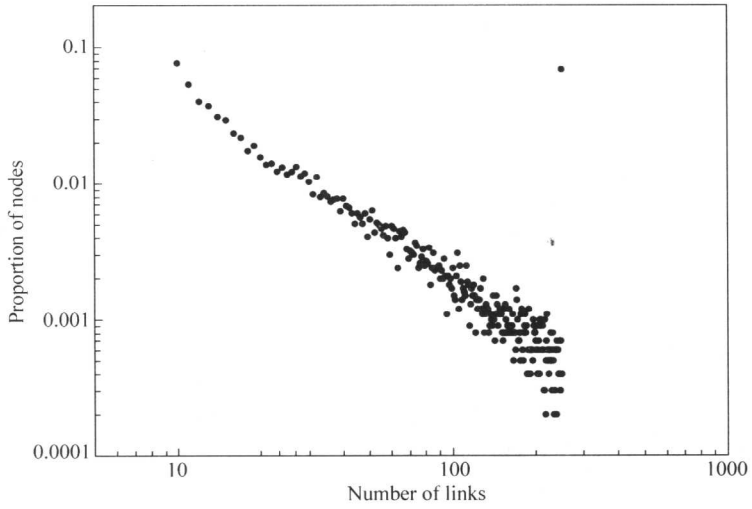


图 3.4.6 Freenet 中不同连接数的结点占结点总数的份额(注意坐标都是 log 型的)
(图片来自[Clarke et al.,2000])

3.4.8 Freenet 的安全性和匿名性分析

之所以要专门以一小节来分析 Freenet 的安全性和匿名性而不将它们合并到 3.4.7 小节对 Freenet 的分析中去，是因为 Freenet 与其他无结构 P2P 网络的最大区别就在于它高度的安全性与匿名性，这是 Freenet 设计的目标和存在的理由。

网络安全领域的研究者将一个系统的匿名性分为几种，比如发送者匿名、接收者匿名、文件标识匿名、关系匿名等。Freenet 提供了很好的发送者匿名和接收者匿名，这在上文中已经讲过；不幸的是，在基础的 Freenet 方案里，文件标识匿名是不可能的，因为 Freenet 依靠文件标识来路由所有的消息。基于文件标识匿名的考虑，有研究者提出以“预定路由”的方法辅助原先的 Freenet 路由，在这个方法里，消息被一系列的公钥加密，这些公钥实际上指定了被加密的消息所遵循的路

由，并且在这条路由路径上的结点既不能知道消息内容，也不能知道消息的真正发出者。当被加密消息到达预定路由终点后，它会被加入到普通的 Freenet 网中按照普通的路由去走，就好像预定路由的终点是消息发出的起点一样。

网络经常受到的另一个威胁是 DoS（denial of service，服务拒绝）攻击，Freenet 的攻击者同样可以使用它，通过向网络中插入大量的垃圾文件来破坏 Freenet 或者降低其工作效率。对付 DoS 攻击的一个常用方法是所谓的散列支付（hash cash），比如规定任何向 Freenet 网中插入文件的用户在插入之前必须做一次耗时的计算作为插入文件的支付代价，以此来降低垃圾文件的插入速度。另一种方法是将每个 Freenet 结点的存储区分成两部分，一部分用来存放旧文件，一部分用来存放新插入的文件。这样无论别人插入多少垃圾文件，最多只能占用它一半的存储区，所以结点至少还有一半存储容量可用。我们写到这里时，又想到了 BitTorrent 中使用的防止 DoS 攻击的方法也可以使用到 Freenet 中来，比如对于任何向 Freenet 中插入文件的用户，规定他事先必须人工输入一个动态产生的随机数，这样也可以很好地降低垃圾文件的插入速度。

3.4.9 Freenet 体系总结

（1）Freenet 是一个自由、安全、匿名的 P2P 网络，这是它最突出的特点。

（2）Freenet 利用每个参加者作为一个结点，每个结点划出一部分硬盘作为公用存储空间，其中存放某个文件的某些数据，不过是全部经过加密的，所以即使是硬盘的主人也无法知道其中到底存放了什么内容，这是 Freenet 安全的核心。

（3）Freenet 中文件查询请求通过一条“代理结点链”从前一个结点传到后一个结点，路径上的每个结点只知道该请求的前一跳结点和由它决定的后一跳结点，对路径上的其他结点一无所知，这种高安全的“隧道路由”是 Freenet 匿名的核心。

（4）Freenet 能够自适应地将使用到的文件缓存到更接近使用者的服务器上，也就是让网上的内容自动进行复制，这对于 Freenet 的工作效率有重要意义。

（5）Freenet 中消息是通过结点路由表中保存的文件标识来导向路由的，文件标识有三种类型：KSK、SSK 和 CHK，分别完成不同的功能。消息路由采用改造过的图的深度优先算法。

（6）Freenet 中插入文件之前先要检测是否存在文件标识冲突。结点存储区是按照 LRU 方法来更新的，文件可以分割成多份保存在多个结点中。每个新结点加入 Freenet 网都会被指定其结点标识，这个标识是由网络中现存的多个结点共同决定的。

（7）由于 Freenet 具有很高的路由效率，并且网络本身随着时间推移会形成更为优化的“文件标识集群”效应，所以 Freenet 具有很好的可扩展性。Freenet 网络总体上符合小世界模型和幂律模型。

(8) 虽然 Freenet 的安全、匿名性已经非常高，但它仍然不是完美的，针对这些缺陷或者可能的安全威胁，人们提出新的方法作为原来 Freenet 机制的辅助或者补充，以使得 Freenet 日趋完善。

3.5 无结构 P2P 网络的特点

3.5.1 覆盖网拓扑结构

“无结构 P2P 网络”这一名称的由来即是因为它的覆盖网“无结构”，这里“无结构”指的是覆盖网没有固定、严格的拓扑结构，而是一个随机生成、松散组织的普通图，理论上这张图可以是任何形状的。然而类似于今天的 Internet，虽然现有无结构 P2P 网络的拓扑结构都不严格遵循某种形状，但总是符合一定的规律——通常是小世界模型或者幂律模型，从本质上看这种规律是由 Internet 用户自发形成的。除此以外，后来的无结构 P2P 网络都发展成了基于超结点的双层拓扑结构，而超结点之间的连接方式往往也是符合小世界模型或幂律模型的。

1. 小世界模型

20 世纪 60 年代美国科学家做了这样一个实验：任何两个美国人，通过 5～6 层关系就能找到自己想找到的人。实验表明：短链普遍存在并且人们能自主地发现它们(没有全局的知识)，这就是“小世界现象”。小世界模型(small-world model)正是由“小世界现象”概括而来，1998 年 Watts 和 Strogatz 在 *Nature* 杂志上发表文章，引入了小世界网络模型，以描述从完全规则网络到完全随机网络的转变。就计算机网络来说，小世界模型是指任意两个网络结点间的距离(即路由跳数)一般很短，并且对每个结点而言，其邻居结点相识(即互相连接)的概率很高，所以结点集群现象明显。Gnutella 协议所支持的 P2P 网络大多可以用小世界模型来刻画其覆盖网拓扑结构，而对 Freenet 的测量结果说明它也是符合小世界模型的。

2. 幂律模型

1999 年，Barabási 和 Albert 在 *Science* 杂志上发表文章指出许多实际的复杂网络的连接度分布具有幂律形式；同是 1999 年，Michalis Faloutsos 等人在 SIGCOMM 会议上发表文章指出 Internet 拓扑结构符合幂律模型。幂律模型取得了计算机领域研究者广泛的认同和接受，并很快成为 Internet 拓扑建模的重要基础。简单地说，幂律模型(power-law model)是指网络中拥有连接数 L 的结点占网络结点总数的份额，正比于 $L^{-\alpha}$，α 是一个取决于网络本身的常数因子；因此网

络中大多数结点连接数很少，少数结点连接数很多。Gnutella 和 Freenet 等一系列无结构 P2P 网络虽然不符合严格的幂律模型，但都可以看成是它与其他模型的合成体，总体上都具有幂律模型的特点，如面对随机结点失效的高容错性等。

3.5.2 路由和定位方法

“路由”(routing)和“定位”(location)这两个词常常很难区分，有时候指的是一回事，有时候又不是。本书倾向于将定位过程中消息走过的路径上的每一跳选择称为一次“路由”，因此定位看成是由多次路由组成的。无结构 P2P 网络正因为“无结构”，不可能预先知道要找的数据到底在哪里，所以消息路由带有很大的随机性，通常以洪泛法为基础，通过 TTL 来限制洪泛的半径以减小网络负担。洪泛法很明显是低效的，因此人们提出很多新的路由方法。下面简单介绍无结构 P2P 网络最典型的四种路由方法。

1. 洪泛法

洪泛法(flooding)是无结构 P2P 网络最朴素、直接的路由方法，也是绝大多数现存无结构 P2P 网络实际采用的方法。每个收到查询消息的结点将消息发送给它的所有邻居结点(或者是其邻居结点中的一部分)，当然消息的上一跳结点除外；当消息到达目的地或者其跳数限制 TTL 用完时消息不再往前传送，因此洪泛法的路由覆盖范围是一个以 TTL 为半径的圆。洪泛法的网络开销是随着跳数限制 TTL 的增加而指数增加的，见图 3.5.1，因此 TTL 不可能很大，覆盖的范围也很有限，即使某个文件存在，可是由于它与查询者的距离大于 TTL，因此也不能被查到，这种查询局部性是无结构 P2P 网络的本质缺陷。

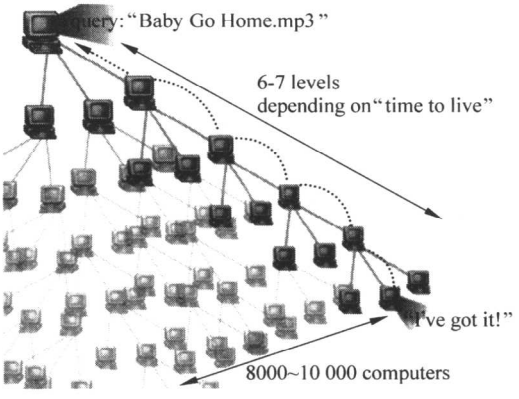


图 3.5.1 洪泛法路由的巨大开销
(图片来自[罗杰文,2005])

2. 扩展环

扩展环(expanding ring)实际上是一种试探性的洪泛法。既然无法预先知道数据离查询者到底多远，所以 TTL 也不要固定死。查询者先以一个小的 TTL 值做洪泛查询，如果查到则成功结束，否则查询者增加 TTL 值再做洪泛查询。这个方法一直继续下去直到查询成功(当然，TTL 值的增加还是会有一个上限的)，其覆盖范围是一个逐渐变大的圆，叫做“扩展环”，见图 3.5.2。扩展环方法相比洪泛法要高效一些，但是它洪泛的本质并没有改变，只能说是向前进了一步。

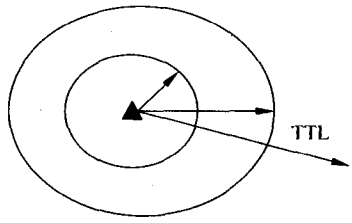


图 3.5.2 扩展环路由(TTL 逐步增加，路由范围也逐步变大)

3. 随机走

“随机走”(random walks)是分布式网络中常用的路由技术，它指的是结点收到查询消息时只随机选择一个邻居结点发送该消息，直到数据被找到或者跳数限制 TTL 用完，它有点类似物理学讲过的固体颗粒悬浮在液体中所做的“布朗运动”。随机走方法比起洪泛法最大的优势在于网络开销随着跳数增加而线性增加，因此 TTL 可以设得非常大。

在标准的随机走方法里，查询者只发出一条消息，这条消息随机往前走，直到结束条件满足，我们称这条消息为一个“行者”(walker)，见图 3.5.3。很明显，如果每次查询只发出一个“行者”，就必须要把 TTL 设得非常大以期望查询成功，但 TTL 太大必定带来过长的查询时延，因此，随机走方法通常是一次派出多个“行者”，每个“行者”走自己的路，这就是所谓的“k-Walker”方法。可以证明在通常的无结构网络中 k-Walker 方法在 T 步内所覆盖的范围与单行者方法在 kT 步内所覆盖的范围是等价的，因此通过 k-Walker 方法可以在不带来多少网络开销的情况下有效降低查询时延。新方法带来新问题，k 的选取是很关键的，k 太小时延就大，k 太大网络开销就大而且过大的 k 对查询来说收效甚微，文献[Lv et al.，2002]认为当 k 选取在 16～64 之间时，能带来最好的效果。

由于随机走方法中每个行者都是独立走的，如果某个行者很早就找到了所要的数据，那么其他行者再走就没有太大意义，因此有人提出“带检测的随机走”方法，也就是说当行者在往前走的过程中周期性地检测查询的发起者是否已经收到成功消息，如果收到它就不用再往前走了。新的问题接踵而来，这个周期应当选多大才能使得检测所带来的额外开销是值得的？文献[Lv et al.，2002]认为行者每走四步检测一次是一个很好的平衡点，但这只是实验结果。

细心的读者可能已经想到了下面的问题：如果同一查询发出的不同行者先后走过同一结点该怎么做才能让后者不去步前者的后尘？网上行者那么多又怎么知道两个行者属于同一个查询呢？对于这两个问题，最简单的方法也最实用，就是同一个查询的所有行者指定同样的ID，每个网络结点会记住它给自己的哪些邻居发送过哪些ID的行者，当具有相同ID的行者到来时，结点会把它发送给过去没到过的邻居，这样后到的行者就不会走以前行者的老路了。这个方法虽然简单，其效果却出人意料的好，因为它有效杜绝了行者之间的重复劳动。

采用随机走方法路由的无结构P2P网络在规模上是可扩展的，并且其跳数限制TTL可以比洪泛法高很多，也就能达到更大的范围；除此之外，文献[Gkantsidis et al.,2004]还指出随机走方法在查询重发时会非常高效。参数设置得很好的随机走方法可以让无结构P2P网络工作效率非常高，它是目前看来很有前途的无结构路由方法。

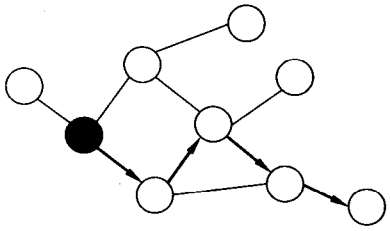


图3.5.3 随机走路由示例(单行者)。黑色结点为源结点，粗线箭头为随机走路线

4. 超结点路由

Gnutella 0.6版、eDonkey和FastTrack协议采用超结点路由(supernode routing)，见图3.5.4，网络的主体——普通结点，将所连接的超结点当作自己的“服务器”，查询直接发给超结点来代为完成。而网络的核心——超结点，自组织成一个超结点网络。由于超结点的数目相对很少，所以超结点网络即使采用洪泛法路由也仍然可行。基于eDonkey协议的各种软件的流行、FastTrack协议的代表KaZaA的流行，证明了超结点路由是实际可行并且高效的。

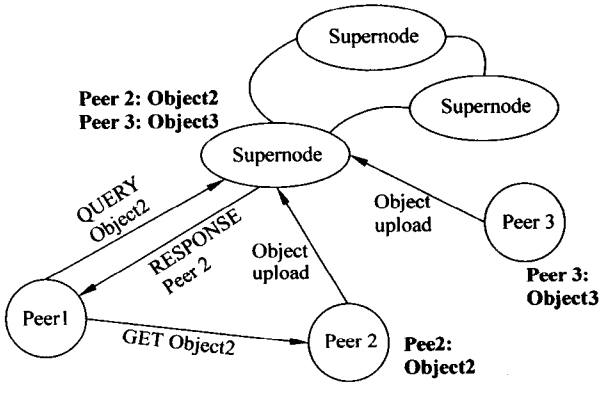


图3.5.4 超结点路由示例

(图片来自[Lua et al.,2004])

3.5.3 容错性与自适应

前面讲过无结构 P2P 网络通常符合幂律模型，即使是 KaZaA 那样的双层结构网络，其真正的 P2P 部分——超结点网络也是符合幂律模型的。幂律模型的一大特点就是对于随机结点失效的高容错性，因此无结构 P2P 网络在这方面也是高容错的。除非有恶意结点能够移去网络中大部分高能力、高结点度的结点（这在双层结构网络中相当于移去大部分超结点），否则无结构 P2P 网络一般不会有严重的性能下降，更不会被分割成多个子网。第 4 章将会讲到，结构化 P2P 网络很容易被分割成多个独立的子网，在这方面它远不及无结构 P2P 网络。

无结构 P2P 网络的自适应所要做的工作主要是检测自己的邻居是否还在线，因此只需要简单的 PING 消息探测就可以维持结点状态的最新。对于双层结构网络而言，自适应多了一项工作：超结点列表的定期更新。第 4 章将会讲到，结构化 P2P 网络的自适应是一件相当困难和繁重的任务。

3.5.4 可扩展性

上文在讲述无结构 P2P 网络路由和查询方法时已经讨论过了可扩展性问题，大致可以说：如果采用单纯的洪泛法，网络难扩展；如果采用更好的方法去改造洪泛法，网络的可扩展性会变高。比如采用超结点路由的 KaZaA，其网络结点数经常在 300 万左右，一样工作得很好。这说明只要方法恰当，无结构 P2P 网络一样是高可扩展的。

3.5.5 安全性与匿名性

通常说来，网络结构越严格，暴露给攻击者的信息就越多，网络的安全性与匿名性也就越差，这是结构化 P2P 网络的缺点，却是无结构 P2P 网络的优点。Freenet 是一个自由、安全、匿名的网络信息交流平台，它做到了数据存储的高安全性（通过数据加密后的透明存放）、数据发出者和接收者的高匿名性（通过隧道路由），同时对于恶意攻击有很高的抗力。Freenet 这一无结构 P2P 体系的出现是无结构 P2P 网络安全性与匿名性最强有力的证明。

3.5.6 增强机制——复制

文献[Cohen & Shenker，2002] 最早研究了无结构 P2P 网络中不同的复制方法，文献[Lv et al.，2002]在此基础上进一步分析了无结构 P2P 网络中数据查询的不同分布和数据复制的不同方法，并且将复制与查询结合起来分析它们组合的状况，下面作简略的介绍。

数据查询的分布典型的有两种：①均匀分布（uniform），所有数据对象被查询

的概率是均等的；②Zipf-like分布，假设将数据对象被访问的概率从高到低排序，那么这里第i个数据对象被访问到的概率正比于$i^{-\alpha}$，α是个常数因子，因此绝大多数对象被访问到的概率都比较低，只有少数对象非常高（它们就是所谓的“热点数据”）。对于Web、Napster和Gnutella等网络的测量都表明数据对象的查询遵循Zipf-like分布。

数据复制的方法典型的有三种：①均匀复制（uniform），所有数据对象复制份数相同；②比例复制（proportional），数据对象复制的份数正比于其被查询到的概率，越“热”的数据复制份数越多；③方根复制（square-root），数据对象复制的份数正比于其被查询到的概率的平方根，虽然越“热”的数据复制份数仍然越多，但是比比例复制要少得多。文献[Lv et al.,2002]从理论和实验两方面证明了：方根复制是无结构P2P网络最优的复制方案，对于现有的符合Zipf-like查询分布的无结构P2P网络来说，采用方根复制能使其效率达到最高，并且文献[Lv et al.,2002]的作者也提出了切实可行的实现方根复制的方案。

3.5.7 优势和缺陷

总体而言，无结构P2P网络最大的优势是“无结构”，最大的缺陷也是“无结构”，任何一个事物总是两面性的，无结构P2P网络就充分体现了这一点。

优势：（1）网络拓扑结构简单，开发和实现的难度小。

（2）高容错性和良好的自适应性。

（3）可以达到非常高的安全性和匿名性。

缺陷：（1）路由效率不高。即使采用新的路由方法，也不可能与结构化P2P网络相比。

（2）可扩展性不高。虽然KaZaA在这方面做得很好，也仍然没有办法比得上结构化P2P网络。

（3）数据无法准确定位。由于路由的局部性，无法保证网络中存在的数据一定能被查询到；相反，结构化P2P网络是可以准确定位数据的，只要存在就一定能被查询到而且会很快。

正是由于无结构P2P网络存在以上的缺陷，才有了结构化P2P网络的提出，这正是我们第4章要讨论的内容。

第4章 第三代P2P网络：结构化P2P体系

如果说1999年属于第一代混合式P2P网络的辉煌，2000年属于第二代无结构P2P网络的成功，2001年则是第三代结构化P2P网络展现的舞台，也是P2P历史上最重要的里程碑（至少在学术界是如此），因为这一年至少发生了几件P2P领域具有重大意义的事：

（1）学术界开始真正关注和重视P2P。IEEE成立P2P专业会议，ACM在通信领域最重要的会议SIGCOMM发表了多篇对结构化P2P有开创性意义的经典论文，此外绝大多数与网络通信、分布式系统有关的国际会议、刊物开始发表或者收录P2P方面的论文。

（2）提出了P2P领域最具代表性的几个经典模型和应用体系，其中包括Chord（in SIGCOMM'01，HotOS'01）、CAN（in SIGCOMM'01）、Tapestry（in UC Berkeley TechReport 2001）、Pastry（in Middleware'01）、CFS（in SOSP'01）、OceanStore（in ASPLOS'00）、PAST（in SOSP'01，HotOS'01）等，这些模型和体系是第三代P2P网络的代表作。

（3）许多知名的学术团体和技术组织成立或者完善专门的P2P研究组，其中包括MIT的Chord和CFS研究组、UC Berkeley的Tapestry和OceanStore研究组、Microsoft Research和Rice University的Pastry和PAST研究组以及

Stanford Peers 研究组等。

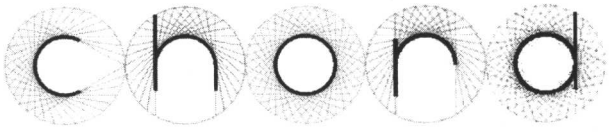
(4) 在商业领域，虽然不能明确知晓那时有多少商业机构开始关注或开发结构化 P2P 网络，但后来的事实证明结构化 P2P 网络对商业领域是很有影响的，比如基于异或度量的 Kademlia 网络后来被许多 P2P 文件共享体系或软件所使用，其中有非常流行的 BitTorrent、eDonkey/eMule 及其扩展版软件，还有 Azureus、eXeem 等。

2001 年只是结构化 P2P 网络的起点，在那以后 P2P 成为计算机网络领域的热点，而结构化 P2P 则成为 P2P 领域的热点。我们希望通过这一章的讲解，引领读者真正深入地进入 P2P 的世界。

4.1 Chord 与 CFS——简单、精确的环形 P2P 网络

4.1.1 Chord 介绍

Chord 是最简单、最精确的环形 P2P 模型。“Chord”这个单词在英文中指“弦”，在分布式系统中指“带弦环”，在 P2P 领域则指基于带弦环拓扑结构的分布式散列表(DHT)或者构建于其上的 P2P 网络。虽然 MIT 和 UC Berkeley 的研究者早在 2001 年之前就开发了 Chord 及其应用系统(见图 4.1.1)，但有关 Chord 的正式论文[Stoica et al., 2001]发表在计算机通信领域著名会议 ACM SIGCOMM'01 上，其他刊物和会议后来也有刊登 Chord 论文的，如[Stoica et al., 2003]。Chord 是结构化 P2P 领域最著名的理论模型，到 2006 年已被引用超过 3000 次，可以说，是凡研究 P2P 的人，没有不知道 Chord 的。



The Chord Project

FAQ | HOWTO | Downloads | Publications | Bugs | Funding | Contact us

The Chord project aims to build scalable, robust distributed systems using peer-to-peer ideas. The basis for much of our work is the Chord distributed hash lookup primitive. Chord is completely decentralized and symmetric, and can find data using only log(N) messages, where N is the number of nodes in the system. Chord's lookup mechanism is provably robust in the face of frequent node failures and re-joins.

图 4.1.1 Chord 项目主页

(http://pdos.csail.mit.edu/chord)，2006 年面貌

如果仅仅将 Chord 看成一个分布式散列表，那么它只支持结构化 P2P 最简单的功能：将结点和数据对象映射到覆盖网中。Chord 基于安全的一致性散列函数来分配结点 ID 和对象 ID。在一个有 N 个结点的网络中，每个 Chord 结点保存 $O(\log N)$ 个其他结点的信息，查询数据对象需要的覆盖网路由跳数也是 $O(\log N)$，当结点离开或者加入网络时，为了维持网络结构、保持自适应性所需要的消息数在 $O(\log^2 N)$。作为分布式散列表，Chord 具有几乎最优的路由效率、确定性的对象查询、负载均衡、高可扩展性以及良好的容错性与自适应；但最重要的一点是：Chord 简单、优美，这是它最为经典的直接原因。

如果将 Chord 看成一个 P2P 网络，那么它是一个基于带弦环拓扑结构的分布式系统，提供数据对象的存储、查询、复制、缓存，在它之上还可以架构更高层的分布式数据存取系统如 CFS(cooperative file system，协同文件系统)，这将在本节后面讲述。

4.1.2 Chord 基础工作原理

Chord 通过安全散列函数(如 SHA-1)给每个网络结点和每个数据对象分配唯一的 ID，即 nodeID=H(node 属性)，H 为散列函数，node 属性可以是结点 IP 地址、端口号、公钥、随机数或者它们的组合；objectID=H(object 属性)，object 属性可以是数据对象的名称、内容、大小、发布者等或者它们的组合。SHA 系列散列函数的 Hash 值长度 m 通常大于等于 160，如此长的位数保证了 ID 几乎不可能出现重复，所以认为是唯一的。

Chord 按照如下方法将数据对象(只是其索引)分配到网络结点中：首先，所有的结点按照 nodeID 从小到大顺时针排列在一个环上，数据对象 k(k 是 objectID)被分配到环上顺时针方向紧随 k(包括与 k 相等)的第一个结点，该结点也称为对象 k 的后继，记做 successor(k)。如图 4.1.2 所示，在 m=3 的 Chord 环上，共有 8 个位置，当前有 3 个结点，nodeID 分别为 0，1，3，successor(1)=1，因此 objectID=1 的对象索引将被放置在结点 1，同理 successor(2)=3，successor(6)=0，对象 2 将被放置在结点 3，对象 6 将被放置在结点 0(下文不再区分对象、对象 ID、对象索引)。

每个 Chord 结点也有其后继，结点 n 的后继是在环上紧随 n(不等于 n)的第一个结点，记做 n.successor，请注意结点后继和对象后继的不同！

为了保持正确、一致的对象放置，当 Chord 中有新结点 n 加入时，原本由 n 的后继结点负责的对象，其中一部分必须分配给 n；当 Chord 中有旧结点 n 离开时，原本由 n 负责的所有对象，必须分配给 n 的后继。除此之外，对象不需要再做移动，这正是一致性散列函数所追求的性质。举例来说，图 4.1.2 中，如果结点 7 加入，那么对象 6 将从结点 0 被重分配到结点 7，此时 successor(6)=7。

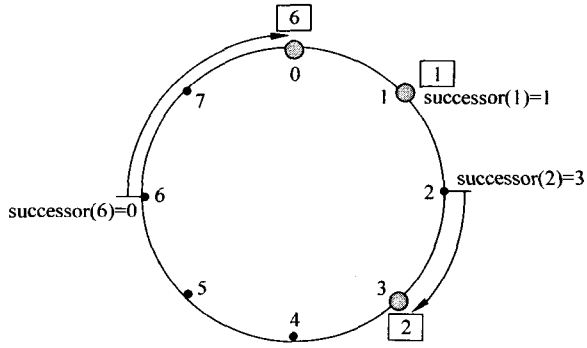


图 4.1.2 一个简单的 Chord 环($m=3$)
(图片来自[Stoica et al.,2001])

有了上述的后继关系，就有了可以工作的 Chord 环，但它是非常低效的，因为每次定位对象都要沿着环一个一个后继地走过 $O(N)$ 个结点，N 为总结点数。为了高效地定位，每个 Chord 结点维护一个额外的路由表，其作用只是为了加速定位过程，通常不影响 Chord 正确工作，因为 Chord 能否正确定位只取决于后继的正确性。

每个 Chord 结点维护一个有 m 项的路由表(m 是 ID 位数)，也称为“指向表”(finger table)。结点 n 的路由表中，第 i 项指向结点 s，$s=\text{successor}(n+2^{i-1})$，$1\leqslant i\leqslant m$，因此 s 是在顺时针方向到 n 的距离至少为 2^{i-1} 的第一个结点，记做 n. finger[i]. node。一个路由表项通常包含相关结点的 nodeID 和其 IP 地址、端口号。容易看出，n. finger[1]. node 也就是 n. successor。

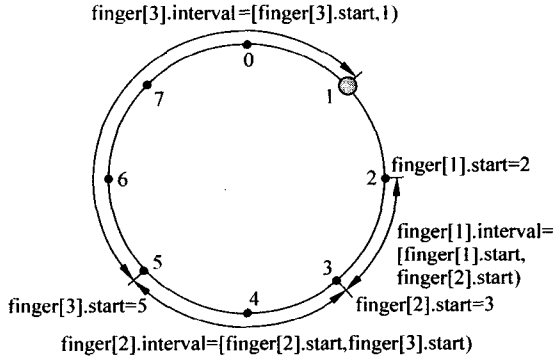
上述的 Chord 路由表设计有两个特点：第一，每个结点只保存很少的其他结点信息，并且对于离它越远的结点所知越少；第二，Chord 结点不能从自己的路由表中直接看出对象 k 的后继。

为了确定对象 k 的后继，如果结点 n 能找到一个比自己离 k 更近的结点，那么那个结点一定知道更多的关于对象 k 所在区域的信息，因为上面已经说过，Chord 路由表的特点即是对越近的结点所知越多。因此，结点 n 在自己的路由表中寻找在 k 之前且离 k 最近的结点 j，让 j 去找离 k 最近的结点，如此递归下去，n 将获得越来越多离 k 越来越近的结点信息，最终可以找到对象 k 的前驱(在 k 之前不等于 k，离 k 最近的结点)，记做 predecessor(k)；类似地，结点 n 的前驱记做 n. predecessor，指在 n 之前不等于 n、离 n 最近的结点。与后继不同，对象、结点的前驱很像，通常不做太多区分。

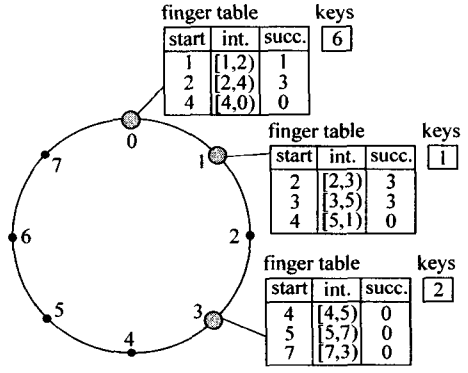
表 4.1.1 总结了 Chord 结点 n 的路由表各项属性及其定义。图 4.1.3 是 Chord 路由表的简单例子。

表 4.1.1 Chord 结点 n 的路由表各项属性及其定义

属　性	定　义
finger[k]. start	$(n+2^{k-1}) \bmod 2^m$，$1 \leqslant k \leqslant m$
. interval	[finger[k]. start，finger[$k+1$]. start)
. node	$\geqslant n$. finger[k]. start 的第一个结点
successor	后继结点，即 finger[1]. node
predecessor	前驱结点



(a)



(b)

图 4.1.3 Chord 路由表的简单例子。图中标出的内容，请对照表 4.1.1 中 Chord 路由表各项属性含义来理解

（图片来自[Stoica et al.，2001]）

4.1.3 Chord对象定位算法

Chord的对象定位算法准确、简单、优美，通过图4.1.4的三个函数即可完成。

```
//请求结点 n 寻找 id 的后继
n. find_successor(id)
    n′=find_predecessor(id);
    return n′. successor;

//请求结点 n 寻找 id 的前驱
n. find_predecessor(id)
    n′=n;
    while(id∉(n′,n′. successor])
        n′=n′. closest_preceding_finger(id);
    return n′;

//返回 id 之前最近的 finger
n. closest_preceding_finger(id)
    for i=m downto 1
        if(finger[i]. node∈(n,id))
            return finger[i]. node;
    return n;
```

图4.1.4 Chord定位函数伪代码。其中 $n.f()$ 表示函数 $f()$ 被结点 n 调用执行，远程调用和远程变量引用前都加了 n'. 标志，本地调用和本地变量引用前不加结点标志

第一个函数find_successor(id)首先调用函数find_predecessor(id)寻找对象id的前驱 n'，然后获得结点 n' 的后继，也就是对象id的后继。

第二个函数find_predecessor(id)通过联系一系列结点，在Chord环上朝id走得越来越近，最终可以到达对象id的前驱结点。

第三个函数closest_preceding_finger(id)是在定位过程中真正被多次调用执行的过程。它的作用是：在结点自己的路由表中，从后往前找到在id前且与id最近的结点，返回给调用它的find_predecessor(id)函数。

举例来说，考虑图4.1.3中简单的Chord网，假设结点3要找到对象1的后继。因为在结点3的路由表中，1属于3. finger[3]. interval即[7,3)，因此结点3让3. finger[3]. node即结点0去找1；结点0同样从自己的路由表中找，它发现自己的后继1恰好是对象1的后继，因此将1返回给结点3，结点3就知道对象1放在结点1中。

从Chord路由表的构造和Chord定位算法可以看出：每次调用函数closest_preceding_finger(id)，新找到的结点离对象id的距离通常比原来少一半，因此通

常情况下最多调用 closest_preceding_finger(id)函数 log$N$ 次就可以定位成功，这正是前面所说 Chord 具有 $O(\log N)$定位跳数的原因所在。

4.1.4 Chord 结点加入算法

在动态的网络环境中，结点可以在任何时刻加入或者离开，对于 P2P 网络来说这要求在动态环境下仍然能够正确地定位每个数据对象。为了达到自适应的目的，Chord 需要保持两个不变的属性：①每个结点的后继始终正确；②对每个对象 k，结点 successor(k)始终负责 k 的索引。另外，为了使得定位能快速完成，结点的路由表最好能不断调整到最新或者比较新的状态。

为了保持上述的两个不变属性，当新结点 n 加入时 Chord 必须完成三个任务：①初始化 n 的前驱和路由表项；②更新网络其他结点的前驱和路由表项以反映 n 的加入；③告诉其后继将应该由 n 负责的数据对象索引传递给 n。

假设新结点 n 通过某种方法联系到一个 Chord 现存结点 n'(n'通常是众所周知结点)，n 将首先通过 n'来初始化自己的状态信息，并将自己加入到 Chord 网络中，这是通过调用函数 join(n')来完成的，如图 4.1.5 所示。而 join(n')内部首先调用 init_finger_table(n')来初始化结点路由表，其次调用 updata_others()来更新其他结点信息，最后将其后继中应该由自己负责的对象移动到自身。

```
#define successor finger[1].node
//结点 n 加入网络；
//n'是网络中任意结点
n.join(n')
    if(n')
        init_finger_table(n');
        update_others();
        //将其后继结点中 id 位于(predecessor,n]的对象移动到自身
    else //n 是网络中唯一的结点
        for i=1 to m
            finger[i].node=n;
        predecessor=n;
```

图 4.1.5 Chord 结点加入函数 join(n')

init_finger_table(n')函数通过现存结点结点 n'初始化 n 的路由表，如图 4.1.6 所示。n 请求 n'帮自己查找后继，从而更新自己的前驱和 n'的前驱，然后再通过多次调用 n'的后继查找函数来初始化自己的路由表。初始化自身路由表时，对表中每一项，结点 n 首先检查第 i 项是否也是合适的第 $i+1$ 项，如果不是才会通过 n'来完成，这个简单的措施具有重要的意义：它将 init_finger_table(n')的算法复杂度(这里理解为传递的消息个数)从 $O(m\log N)$降到了 $O(\log^2 N)$。

```
//初始化本地结点的路由表
//n'是网络中任意现存结点
n. init_finger_table(n')
    finger[1]. node=n'. find_successor(finger[1]. start);
    predecessor=successor. predecessor;
    successor. predecessor=n;
    for i=1to m-1
        if(finger[i+1]. start∈[n, finger[i]. node))
            finger[i+1]. node=finger[i]. node;
        else
            finger[i+1]. node=n'. find_successor(finger[i+1]. start);
```

图 4.1.6 Chord 结点初始化路由表函数 init_finger_table(n')

update_others()函数更新其他结点的状态信息以反映 n 的加入，如图 4.1.7 所示。当且仅当下面两个条件满足时，结点 n 将成为结点 p 路由表的第 i 项：①结点 p 在 n 之前至少 2^{i-1}；②结点 p 路由表的当前第 i 项结点在 n 之后。满足这两个条件的第一个结点 p 是结点($n-2^{i-1}$)的前驱，因此，update_others()首先找到该前驱，然后调用函数 update_finger_table(n,i)，递归地更新 Chord 网中所有需要更新路由表第 i 项的结点信息。这个递归过程是通过沿 Chord 环逆时针走、即不断走向前驱结点来完成的，因为如果结点 p 需要更新其路由表第 i 项，那么 p 的前驱很可能也需要，如此递归下去；反过来，如果结点 p 不需要更新其路由表第 i 项，那么 p 的前驱也不需要更新，这正是函数 update_finger_table(s,i)的递归结束条件。通常情况下，一个新结点加入 Chord 网后，需要更新信息的结点数为 $O(\log N)$，所以寻找和更新这些结点信息的时间复杂度(理解为消息总跳数)为 $O(\log^2 N)$。

```
//更新所有那些路由表应该引用 n 的结点状态
n. update_others()
    for i=1 to m
        p=find_predecessor(n-2^(i-1));
        p. update_finger_table(n,i);

//如果 s 是 n 的路由表的第 i 项，则用 s 更新 n 的路由表
n. update_finger_table(s,i)
    if(s∈[n,finger[i]. node))
        finger[i]. node=s;
        p=predecessor;
        p. update_finger_table(s,i);
```

图 4.1.7 Chord 结点加入时，更新其他结点信息的函数 update_others()和它所调用的子函数 update_finger_table(s,i)

4.1.5 Chord 自适应算法

上面所介绍的 Chord 算法虽然完备、细致，但是却有两个不能解决的困难：①并发操作；②不正常操作，如不做通知的结点退出。在这两种情况下，光靠上面的算法不能保证 4.1.4 节中所说的 Chord 正确工作所需要的两个不变属性，并且由于路由表出错太多，工作效率也会急剧下降。鉴于此，Chord 采用称为“稳定化”(stabilization)的自适应算法来保持结点后继的正确性。上文说过，只要后继关系正确，Chord 就一定能正确定位，虽然定位效率可能很低，但至少保证了正确性。在后继正确的前提下，Chord 进一步使用后继信息来验证和更新路由表项，这样定位可以恢复到和正确时一样快。

图 4.1.8 是 Chord 自适应算法的代码，它包括三个部分：①并发的结点加入函数 join(n')；②稳定函数 stabilize()；③路由表更新函数 fix_fingers()。

```
n. join(n')
    predecessor=nil;
    successor=n'. finde_successor(n);

//周期性地修正 n 的后继，并通知其后继修正前驱
n. stabilize()
    x=successor. predecessor;
    if(x∈(n,successor))
        successor=x;
    successor. notify(n);

//n'认为 n 可能是自己的前驱
n. notify(n')
    if(predecessor is nil or n'∈(predecessor,n))
        predecessor=n';

//周期性地修正路由表项
n. fix_fingers()
    i=random index > 1 into finger[];
    finger[i]. node=find_successor(finger[i]. start);
```

图 4.1.8 Chord 自适应算法，包括三个部分：①并发的结点加入函数 join(n')；②稳定函数 stabilize()及其子函数 notify(n')；③路由表更新函数 fix_fingers()

新的 join(n')函数比过去简单得多，它只是通过现存结点 n'来找到 n 的后继，其他什么事都不做。每个 Chord 结点周期性地调用稳定函数 stabilize()和路由表更新函数 fix_fingers()，前者首先修正自己的后继，然后通过调用子函数 notify(n')来通

知其后继修正前驱；后者在前者的基础上随机修正自己的路由表项，虽然这不能保证路由表最新，但对提高定位效率是很有意义的。路由表的正确度、定位效率能提高多少，取决于它被调用执行的周期长短。

4.1.6 Chord容错性和复制、缓存

上文多次提到过，在Chord中正确的后继关系是Chord一切工作的基础。然而单后继始终是脆弱的，不管机制多么完善，网络的动态性和不确定性都可以导致单后继的失效。鉴于此，实际的Chord给每个结点维护一个后继列表(successor list)，其中保存了该结点在Chord环上的r个后继，除非r个后继都失效，否则结点的后继关系就可以修正。典型地，取$r=O(\log N)$，假设每个结点失效概率为1/2，在这样动态的网络中，通常find_successor()函数仍然能在$O(\log N)$的跳数内找到正确的后继。

后继列表还能帮助Chord复制数据。典型的复制方式，是将结点n所保存的数据对象复制到n的r个后继中，复制对于数据的可用性、持久性有重要意义。

除了复制，Chord还提供了缓存机制。缓存的作用主要在于提高获取数据的速度，典型地采用路径缓存，即在Chord定位过程中的每个中间结点缓存所获得的数据对象。这样当后来再定位此数据时，可以在中途就得到数据而不需要走到底。

采用后继列表、复制、缓存后，Chord的容错性得到了本质的提高，数据可用性及定位效率也比以前高得多。通常情况下，Chord是一个高容错、高效率、高可用的结构化网络，除非有恶意结点攻击迫使Chord环分割为多个自闭、不交的子环。当然，分割问题并不是不可解决的，本书后面将讲到这一点。

4.1.7 Chord实验分析

1. 负载均衡

负载均衡是使用一致性散列函数的结构化P2P网络的共同属性。对于Chord而言，由于数据对象被分配到其后继中，而数据对象、结点的ID都是随机、均匀产生的，因此每个结点所负担的数据对象数也应该大致均衡。此外，Chord还采用了“虚拟服务器”(virtual server，也可称“虚拟结点”)的方法，在一台计算机上运行多个Chord结点，这可以使得结点各尽所能：结点能力越大，其上运行的虚拟服务器数越多。图4.1.9是一个实验Chord网的负载分布图，实验中使用了1万个结点、50万个数据对象。可以看出，结点所负担的对象数目基本上集中在50左右，这正是负载平均数(50万/1万=50)。

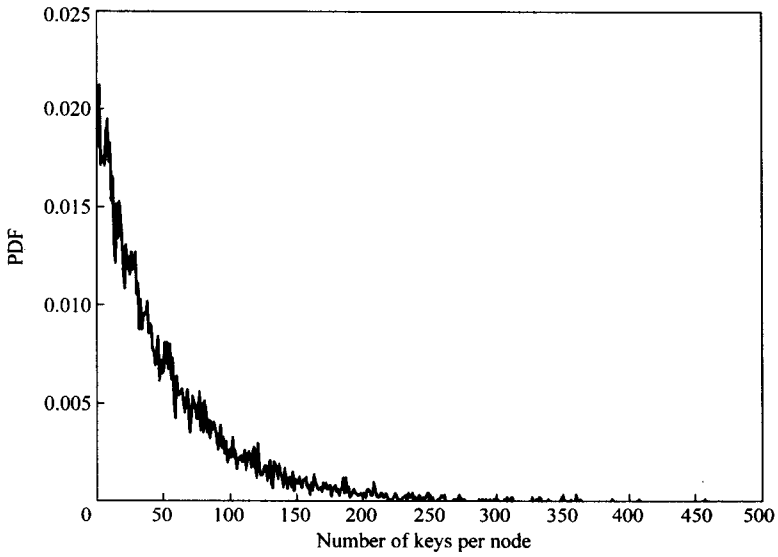
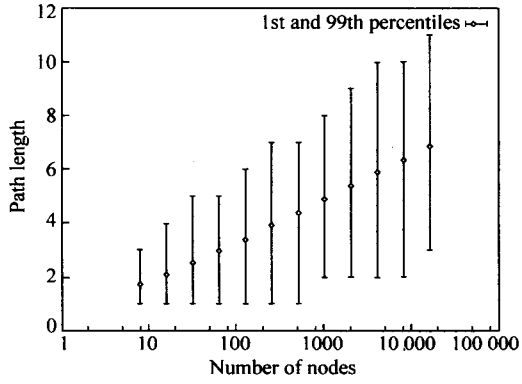


图 4.1.9 Chord 负载分布图

（图片来自[Stoica et al.,2001]）

2. 定位路径长度

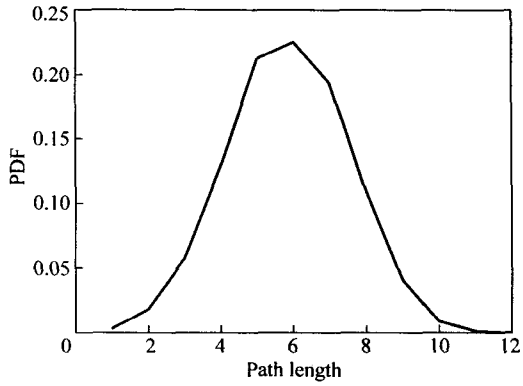
前面已在理论上说明 Chord 定位路径长度为 $O(\log N)$跳。实验中网络结点数取 $N=2^k$，而数据对象数取 100×2^k，k 从 3 取到 14，每次独立地实验测量定位路径长度（即覆盖网跳数）。测量的结果如图 4.1.10 所示：路径长度平均约 $\log N/2$，是



(a) 定位路径长度与Chord结点数的关系图

图 4.1.10 Chord 定位路径长度及其概率分布

（图片来自[Stoica et al.,2001]）



(b) 定位路径长度的概率分布图(网络结点数为2^{12}，刚好6占的概率最大)

图 4.1.10(续)

预计值 log$N$ 的一半。这并非偶然，因为 Chord 路由表的指数构造，它每次查找都可以将目的 ID 与当前结点 ID 之间的差距减小至少一半，将这个差距值用二进制表现出来：如果最高位(第 i 位)是 1，那么只要下一跳沿着其路由表第 i 项走，差距值最高位就可以变为 0；如果下一位(第 $i-1$ 位)还是 1，同样走第 $i-1$ 项就可以将第 $i-1$ 位变为 0；但如果第 $i-1$ 位本来就是 0，那么直接走第 $i-2$ 项就可以了。依此类推，从概率的角度讲，目的 ID 与当前结点 ID 间的差距的二进制值中 0，1 应该是各占一半，因此平均路径长度刚好是 log$N/2$。

4.1.8 Chord 总结

(1) Chord 采用带弦环拓扑结构，通过一致性散列函数将结点、数据对象映射到覆盖网上，数据对象(的索引)由其后继结点负责，简单、精确正是 Chord 最大的特点。

(2) 每个 Chord 结点维护一个很小的路由表，其中第 i 项保存着到自身的距离至少为 2^{i-1} 的第一个结点的位置信息。后继关系是 Chord 定位的基础，路由表则起着加速定位的作用，正是它使得 Chord 定位路径长度为 $O(\log N)$跳。

(3) Chord 需要保持两个不变的属性才能正确工作：①每个结点的后继始终正确；②对每个对象 k，k 的后继结点始终负责 k 的索引。另外，为了能快速完成定位，结点的路由表最好能不断调整到最新或者较新的状态。Chord 通过自适应的结点加入算法完成这两个功能，其算法复杂度在 $O(\log^2 N)$。

(4) 为了应对动态的网络环境和并发、非常规的操作，作为对(3)的补充，Chord 采用了周期性的“稳定”算法和路由表更新算法来检查和修正后继关系及路由表项，它们对于 Chord 网络来说是不可或缺的。

(5) 为了保持高容错性，Chord 采用了后继列表的方式来避免单后继的脆弱。同时，基于后继列表的复制对 Chord 的数据可用性有很大的提高，而路径缓存则进一步加速了 Chord 的定位。

综上所述，Chord 是一个简单、精确的环形 P2P 网络，它具有对数级的定位效率和良好的容错性、自适应性，再加上复制、缓存等机制的辅助，使 Chord 成为一个高可扩展、高效率的实用结构化 P2P 模型。

4.1.9 CFS 介绍

CFS(cooperative file system，协同文件系统)是以 Chord 作为其基础的 P2P 只读文件存储系统。依靠 Chord 作为其分布式散列表，CFS 提供高效、容错、负载均衡的文件存储和获取。

CFS 中结点有“服务器”和“客户”之分，但它们不同于传统 C/S 模式中的服务器和客户，同一个结点可以既是服务器又是客户。文件被分块(block)存储在 CFS 服务器中，客户通过 CFS 存储和获取文件。CFS 由三层构件组成：①FS，File System，文件系统；②DHash，分布式数据块散列表；③Chord，底层定位散列表，其功能如下所示：

FS	高层，从 DHash 层获得数据块，将这些块转换为文件，给更高的应用提供文件系统接口
DHash	中间层，分布和缓存数据块以平衡负载，复制数据块以容错，并且通过服务器选择来减少时延；使用 Chord 来定位数据块
Chord	底层，维护路由表，定位数据块所在的服务器

CFS 服务器、客户及其与 FS、DHash、Chord 三层的关系如图 4.1.11 所示，垂直箭头代表本地过程调用，水平箭头代表远程过程调用(RPC，remote procedure call)。

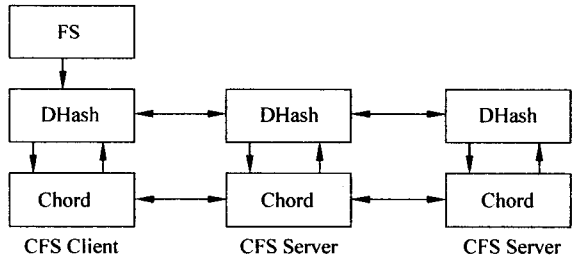


图 4.1.11 CFS 系统架构图
(图片来自[Dabek et al.,2001])

CFS所提供的“分块”功能对其性能是有重要意义的，DHash层则专门负责对这些“分块”的操作。尤其是对于那些流行的大容量文件，将其分块存储在多个网络结点中，既减少了单个结点的巨额负载，将负载平衡到多个结点中，又提高了获取文件的速度和文件可用性。对于小文件而言，分块对它没有太多影响（因为分不了几块），CFS是通过路径缓存提高其定位速度的。除此之外，CFS还提供了“预取”（pre-fetching）功能重叠文件查询和获取来减小下载时延，将每个数据块复制到一定数目的网络结点中以提高容错性与持久性，在查询路径上缓存数据块以提高定位效率。另外，为了防止结点滥用CFS系统，CFS限制每个用户插入系统的数据低于某个限额。

目前CFS的实现使用了SFS文件系统工具包，可以运行于Linux、OpenBSD、FreeBSD操作系统。其软件原型的实验结果显示：CFS将数据传递到客户的速度和FTP（file transfer protocol，文件传输协议）一样快，这样的传输速度是可喜的。

4.1.10 CFS文件系统结构

CFS文件系统的组织结构类似于UNIX V7文件系统，但使用了DHash数据块、块标识来替换UNIX中的磁盘块、块地址。如图4.1.12所示，每一块要么是文件的一部分，要么是文件系统元数据的一部分，比如目录。块之间有包含关系，通常父块包含其子块的标识。

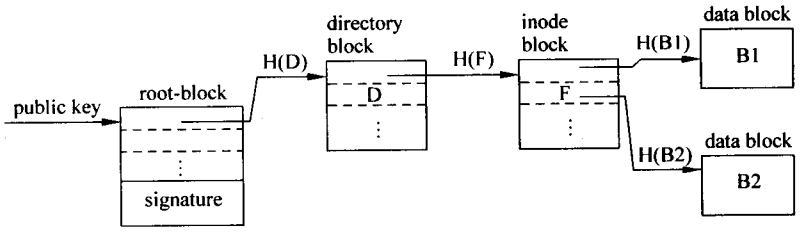


图4.1.12 CFS文件结构示意图。图中文件根块（root-block）由发布者的公钥来标识，并被相应的私钥签名。其他块被它们内容的安全散列值标识
（图片来自[Dabek et al.,2001]）

当发布者要向CFS系统插入一个文件时，首先这个文件将被分成很多大小相同的块，每块由其内容的安全散列值标识。类似UNIX目录结构，CFS允许插入元数据块（如块目录）以组织这些块，但文件根目录一定放在根块（root-block）中，根块由发布者的公钥来标识，并被相应的私钥签名。这样，客户可以通过上述公钥获取文件根块，并通过签名验证发布者是否真实。获得根块后客户通过根块中保存的子块标识一层层类似目录地走下去，从而找到所有的文件块。对这些文件块，客户都可以通过其标识来验证数据完整性。

CFS 虽然自称“只读”文件系统，但实际上文件也是可“写”的，在 CFS 中“写”表现为文件发布者对原文件的更新操作。文件更新是通过发布新的文件根块来完成的，新根块与旧根块用同样的私钥签名，也以同样的公钥标识，本质上只是更新了一些“文件块指针”，因此其他块只要不需要更新，都可以不做变动，从而节省了更新开销。另一方面，根块中附加的时间戳既标明了谁更新，又防止了攻击者使用旧根块来做更新的“重放”攻击（replay attack，截取通信过程中的旧消息，以后重新发送它）。

CFS 中没有明显的“删除”操作，而是采用“协议有限间隔”（agree-upon finite interval）的方法，对于每个数据块，在间隔时间内，只要存储结点没有收到发布者新的保持请求，就可以自主删除该块，所以 CFS 实际上提供了一种隐式的删除方案。

4.1.11 CFS 对 Chord 的改进：前驱列表定位、服务器选择和结点 ID 认证

原来的 Chord 定位中，最重要的是通过调用 find_predecessor(id)函数返回对象 id 的直接前驱，CFS 对此做了改进（如图 4.1.13 所示）：该函数每次通过调用 preceding_node_list(id)子函数获得一个前驱列表，然后从这些前驱中选出最大的前驱 n'，再通过 n'来获得 id 前驱列表，这个过程循环执行下去，直到最终找到 id 的直接前驱。虽然改进后的方法每次返回的信息比过去多，但定位速度、容错性比以前要好得多。

```
//请求结点 n 寻找 id 的前驱
n. find_predecessor(id)
    n'=n;
    while(id∉(n',n'. successor])
        l=n'. preceding_node_list(id);
        n'=max n''∈l s. t. n'' is alive
    return n';

//请求结点 n 返回一组结点，这组结点在 n 的路由表中，
//或者是 n 的后继列表中位于 id 之前的结点
n. preceding_node_list(id)
return {n'∈{fingers∪successors} s. t. n'∈(n,id]}
```

图 4.1.13 改进后的 Chord 前驱寻找函数、获得前驱列表的子函数

为了减少定位时延，CFS 中在其 Chord 层加入了服务器选择，即定位过程中选择下一跳时，尽量选择那些时延小、在底层物理网中靠近的结点。以上面的前驱

寻找函数为例，假设某一步结点 m 返回给结点 n 它的 id 前驱列表，列表中除了结点位置信息，还有每个结点到 m 的时延（这些时延在 m 初始化、更新自己的路由表时获得）。基于这些时延，n 对列表中每个结点进行评价，选取它认为最合理的作为下一跳，这也就是所谓的“服务器选择”。设 $C(n_i)$ 表示 n 取 n_i 作为下一跳的预计开销，Chord 采用了如下公式评价 $C(n_i)$，从而选择最小的 $C(n_i)$ 作为下一跳：

$$C(n_i) = d_i + d' \times H(n_i)$$

$$H(n_i) = \text{ones}((n_i - \text{id}) \gg (160 - \log N))$$

其中，d_i 表示结点 m 在前驱列表中告诉 n 的它到 n_i 的时延，d' 表示 n 所取的每跳平均时延（通常基于 n 过去的经验计算）；$H(n_i)$ 表示 n 取 n_i 作为下一跳后估计剩余的 Chord 跳数；N 表示 n 对 Chord 网总结点数的估计（通常基于 n 的邻近结点密度），因此 $\log N$ 是当前 Chord 网中真正决定跳数的高比特位数；≫为位运算右移符号。结点 ID 使用 160 位散列值，ones(x) 函数计算 x 中有多少位为 1，因此 (n_i－id) 被右移 (160－log$N$) 位后，ones() 函数实际上计算了 n_i 到 id 间近似的跳数 $H(n_i)$。

在计算出 $H(n_i)$ 后，将 $H(n_i)$ 乘以每跳平均时延 d'，即得到取 n_i 作为下一跳后的估计剩余时延，加上到 n_i 本身的时延，就是取 n_i 所要的总时延的预计值 $C(n_i)$。这里我们假设了一个前提：n 到 id 的定位时延，大致等于 n 到 n_i 的时延加上 n_i 到 id 的定位时延，即所谓的“时延传递性”。通常情况下，因特网时延是近似符合这个特点的，因此采用服务器选择的 Chord 定位一般能工作得较好，从而有效地降低时延。

结点 ID 认证的目的是防止恶意结点伪造 ID 或者虚报 IP 地址等信息。当一个新的结点 n 加入网络时，一些现存的结点会考虑是否将它加入到自己的路由表中。每个这样的现存结点首先发送一条消息到 n 所宣称的 IP 地址，并附加一个“现时”(Nonce，一个很难预测、伪造的随机数，仅用来标识本次事务)。如果该 IP 地址的结点承认它拥有所宣称的 ID，并且将此 ID、收到的“现时”以及其他信息（如虚拟结点号）回发给现存结点，那么现存结点首先对比自己发出的“现时”与收到的“现时”，结果相同证明 n 没有虚报 IP 地址（否则 n 不可能收到该“现时”值），然后计算 n 的 IP 地址和其他信息的散列值，如果该散列值与结点 n 宣称的 ID 相同，证明 n 没有伪造 ID，此时现存结点才将 n 的 ID、IP 地址等信息加入自己的路由表。

4.1.12 CFS 中的复制、缓存和负载均衡

在 CFS 中复制、缓存和负载均衡都是基于数据块的，因此都由 DHash 层负责。DHash 将每个数据块复制 k 份放到 k 个服务器中以提高数据可用性，并在服务器不断加入、离开的情况下自动维护 k 个副本。典型地，DHash 将数据块的 k 个副本放在 Chord 环上数据后继结点的 k 个后继上，这样 DHash 就可以很容易地

找到它们，因为 Chord 每个结点都维护一个后继列表，并且通常后继列表项 r 大于 k，同时，这也使得当服务器加入、离开时 k 个副本的维护变得简单、方便。复制除了提高数据可用性外，还可以提高系统的容错性与工作效率：其原因在于副本虽然放在相邻 ID 的结点上，但这些结点在物理上（或者说 IP 地址上）是分散的。从物理上同时破坏这些副本是很难做到的事，因此提高了容错性，其次，客户每次查询都可以获得 k 个副本信息，它可以从中选出离自己最近、时延最小的结点来获取数据，因此提高了工作效率。

DHash 缓存数据块以避免那些热点数据的后继结点负载过重。当一个 CFS 客户查询数据块时，它将依次走过 $O(\log N)$ 个结点，在中途每一步，客户查询消息都会问当前结点是否有所需数据的缓存，如果有就不需要再往前走，消息直接回传给查询的客户。客户获得所需数据后，会给它刚才路由过程中的每个中间结点发送一份数据副本作为缓存，也就是所谓的“路径缓存”。当以后其他客户再查询该数据时，它很可能会经过过去客户查询路径的一部分，因此就可以直接从缓存中获得数据。每个结点分配给缓存的存储空间通常是有限的，到达限度后 DHash 采用 LRU(least-recently-used，最近使用优先）替换策略以新数据替换最不近使用的数据。

图 4.1.14 是 CFS 的 DHash 复制、缓存的简单例子。

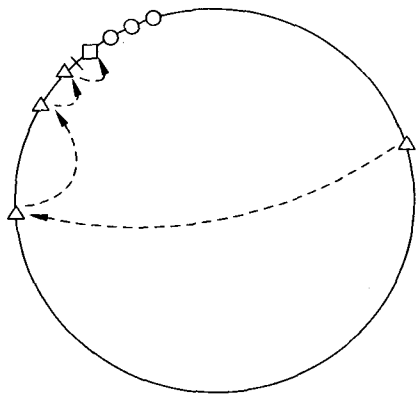


图 4.1.14 DHash 后继复制和路径缓存。图中细线代表数据 ID，菱形代表其后继结点，圆圈代表数据后继结点的 k 个后继存放副本，三角代表查询路径结点存放缓存
（图片来自[Dabek et al.,2001]）

CFS 负载均衡采用了多种辅助手段：首先是 Chord 中已经采用的“虚拟服务器”方法；其次是“限额”方法，限定每个客户可以向系统中插入的数据总量。此外，缓存、复制都能在一定程度上提高负载均衡性。

4.1.13 CFS总结

(1) CFS是以Chord作为其基础的P2P(只读)文件存储系统。文件被分块存储在CFS服务器中，客户通过CFS存储和获取文件。

(2) CFS由三层构件组成：①FS，从DHash层获得数据块，将这些块转换为文件提供给客户；②DHash，负责对数据块的分布、复制、缓存和预取等；③Chord，为DHash层提供数据块定位。

(3) CFS文件系统类似UNIX文件目录结构，只是以根块代替根目录、以元数据块代替子目录、以数据块代替文件，而以块标识代替文件地址。

(4) CFS对Chord做了很多改进，如采用前驱列表定位以提高定位容错性，使用服务器选择减少定位时延，对结点ID认证以防止ID伪造和IP虚报。

(5) CFS对数据块采用后继复制以提高数据可用性，同时减少了客户获取数据的时延；路径缓存则提高了系统工作效率，同时避免热点数据的后继结点负载过重；CFS负载均衡采用了多种办法，如“虚拟结点”和“限额”，复制和缓存对负载均衡也有促进作用。

4.2 CAN——简单、容错的多维空间P2P网络

4.2.1 CAN介绍

CAN(content addressable network，内容可寻址网络)采用多维空间拓扑结构，更严格地说是一种多维Torus(环面)结构。如最常见的二维空间网格，可以想象成解析几何中的笛卡儿平面，而CAN给结点的编址与坐标平面给点的编址也有很大的相似之处，所以CAN的思想简单、直观，比其他结构化P2P模型容易理解。除此之外，CAN高效、容错、可扩展，也是结构化P2P的一个经典代表。CAN正式的论文[Ratnasamy et al., 2001]发表在计算机通信领域著名会议ACM SIGCOMM 2001上，与Chord同年同会，它是结构化P2P领域仅次于Chord的最著名理论模型，经常与Chord相提并论。

CAN的多维空间被动态地分配给其网络结点，每个结点占有一个属于自己的方块并负责该方块中所有的“点”(如图4.2.1所示)，而每个数据对象通常被唯一映射到多维空间中的一个“点”，由负责该点的结点来存储数据对象(的索引)。每个结点维护一个路由表，其中记录它在多维空间上的邻居信息，如图4.2.1中结点D可以记录B、C、E的ID和地址。与Chord类似，CAN的定位也是逐步完成的，每一步挑选当前结点路由表中离目的结点最“近”的邻居作为下一跳(这个“近”是指多维空间上的近)。与大多数分布式系统定位一样，CAN定位算法也是贪心、局

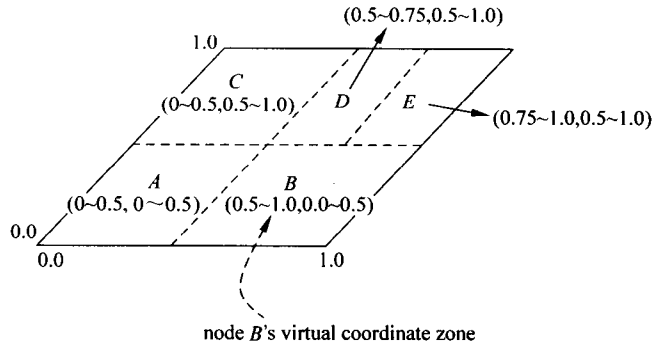


图 4.2.1 有 5 个结点的二维 CAN 示意图
（图片来自[Ratnasamy et al.,2001]）

部的，对一个 d 维 CAN 来说，如果维护一个有 $2d$ 项的路由表，其定位效率通常在 $O(d\sqrt[d]{N})$，这个值看上去似乎不符合结构化 P2P 的经典路由效率 $O(\log N)$，但只要取 $d=\log N$，$O(d\sqrt[d]{N})$ 也就成了 $O(\log N)$，因此 CAN 的定位效率和其他的结构化 P2P 网络本质上是一样的。

每个新结点加入 CAN 以后，它必须拥有自己的空间。以 2 维 CAN 为例，每个新结点被映射到一个点，它加入之后，该点原来所在的方块通常将一分为二，一半分给新结点负责，一半留给原来负责的结点（如图 4.2.1 中 D 和 E），相应的数据对象也会重分配。与上面的方块分割对应，当旧结点离开 CAN 后，它的某个邻居必须接管它原来负责的区域，相当于方块合并。

本节标题上讲 CAN 是“容错的”，这体现在 CAN 路由选择的灵活性上。首先，CAN 不需要像 Chord 那样维护一些严格的不变属性（如结点后继关系），每个邻居对 CAN 结点来说都是同等地位的；其次，在 CAN 中任意两个点（方块）间存在多条路径，即使很多邻居信息失效，仍然能以较快的速度定位目的结点。CAN 优秀的容错性，实际上是源于其多维空间的拓扑结构。

虽然到目前为止 CAN 只是停留在理论模型阶段，并没有基于其上的具体应用系统被开发出来，但至少作为一个良好的底层分布式散列表，CAN 完全可以作为诸多 P2P 应用系统的底层定位设施，替代原来的方案。

4.2.2 CAN 网络构建

CAN 网络构建从新结点加入开始。一个新结点加入 CAN 分三步完成：①自举；②寻找区域；③加入路由表。下面具体讲述这三步。

JOIN STEP1：自举

自举（bootstrap）指新结点寻找一个已在 CAN 中的结点来加入 CAN（“自举”

这一名称表示“自我举荐”之意）。假设CAN有一个相关的DNS(domain name system)域名，比如说一个公开的网络地址，新结点通过它来获得一个自举结点（也称为入口结点）的IP地址（此时自举结点也就是众所周知结点）。自举结点向新结点提供一个列表，其中包含了一些CAN现存结点的信息。

JOIN STEP2：寻找区域

在自举之后，新结点 n 随机地选择CAN空间中的一个点 P 并向 P 发送一个加入请求消息。这条消息可以通过自举获得的任意一个现存结点发送到CAN网络中，然后消息被逐步路由到点 P 所在的区域，并最终到达负责 P 所在区域的结点 n'。n' 按照某个规则将其负责的区域一分为二，以2维CAN为例，区域可以先按照横坐标分，再按照纵坐标分，使得区域尽量规整。n' 将原区域分一半给 n 后，相应的数据对象也要传递给 n。

JOIN STEP3：加入路由表

获得自己的区域后，结点 n 从 n' 那里获得其邻居的IP地址等信息（当然 n' 也是 n 的邻居），并通知每个邻居更新其路由表以反映 n 的存在。由于P2P网络的动态性，很多时候区域分割、合并消息并不能到达它应该到的地方。为了更好地更新结点状态，CAN也采用了自适应的周期性方法，每隔一段时间每个结点自动向其邻居发送自己所负责的区域和自己的路由表信息。如果邻居发现自己的路由表与当前收到的消息不一致，则更新相应的路由表项。由于每次结点插入、离开或者做周期性更新，只需要通知其邻居结点，而每个结点的路由表记录 $O(d)$ 个邻居，因此CAN的自适应开销是 $O(d)$ 的，当维数 d 较低时这个开销相比Chord和大多数P2P系统要小得多。

图4.2.2是一个简单的CAN结点加入的例子，7号结点通过自举结点 (x,y) 加入，它选择1号区域中的点，因此1号结点从横坐标方向分割为两部分，将右半部分交由7负责。7号结点加入以后，它将通知其邻居更新路由表。图中仅写出了1、7的路由表变化。

与结点加入相反，当某个结点离开CAN后，它原先负责的区域需要由其他结点来接管。正常情况下，离开的结点会显式地将其区域及所负责数据交给一个邻居，如果该邻居可以将离开结点区域与自己的区域合并成一个规整的单区域，区域合并就做完了；但如果不能合并成一个规整的单区域，离开结点只能将其区域交给占有最小区域的邻居，由它来暂时负责两块区域，但并不合并。

然而，在动态的网络环境下，很多结点离开都是突然的，不会去通知邻居自己的离开，因此，每个CAN结点周期性地检测其邻居是否在线，一旦发现不在，就开始接管掉线的结点区域。这个方法带来一个问题：假定结点 n 突然掉线，在掉线后的周期中它的多个邻居都检测到了 n 掉线，每个邻居就会独立地开始做 n 的接管工作，这必定会导致冲突。CAN的方法是：让每个邻居做完接管工作以后，向 n

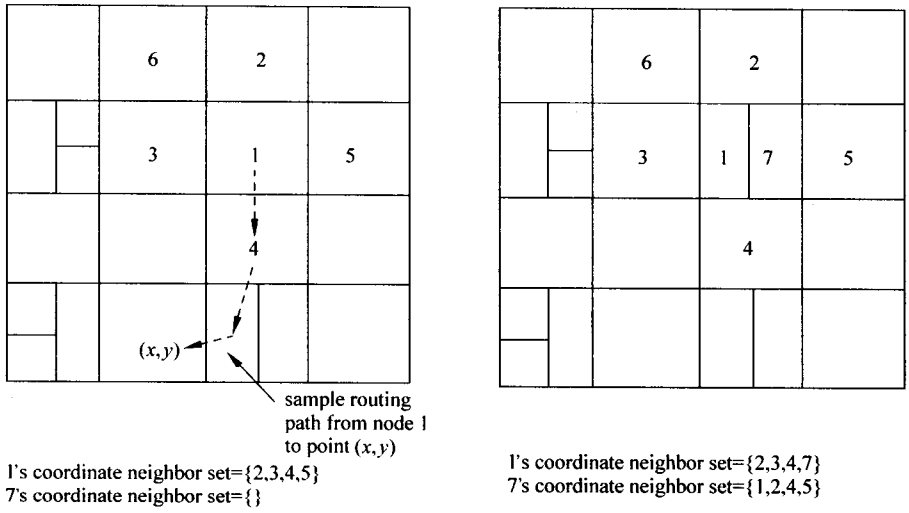


图 4.2.2 简单的 CAN 结点加入举例

（图片来自[Ratnasamy et al.,2001]）

的其他邻居发送 TAKEOVER 消息，该消息中还包括消息源的区域信息；收到 TAKEOVER 消息后，其他邻居比较消息源的区域和自己的区域，如果前者大它就回发 TAKEOVER 消息表明由自己接管更合适，如果后者大就取消正在进行的接管工作。这个自适应的方法保证了只有 n 的最小区域邻居才能接管原来 n 负责的区域。

明白 CAN 的网络构建方法后，一个很自然的问题是：随着结点不断加入、离开，CAN 网络的区域划分将变得支离破碎，而且由一个结点负责多个结点的情况将越来越多，直到负载超过结点能力上限。鉴于此，CAN 采用了“背景区域重分配”(background zone reassignment)方法来合并那些支离破碎的区域，使它们变得规整，并尽量让一个结点只负责一块区域，具体的方法见 CAN 论文[Ratnasamy et al.,2001]，这里不做详述。

4.2.3 CAN 增强机制：多维、多空间、多散列

1. 多维

前面一直以 2 维 CAN 作为例子，但实际上，如果 CAN 适当增加维度 d，可以有效地减少定位路径长度 $O(d\sqrt[d]{N})$，而路由表项数(通常 $2d$)不会增加多少。同时，维度增高也可以提高容错性，因为两点间的路径选择比以前要多很多。当然，维度 d 并非越大越好，因为自适应的开销，带来了对 d 的限制。

2. 多空间

一个CAN空间只是实际网络的一个逻辑映射，因此，完全可以使用多个不同的CAN空间，每个空间都是实际网络的完整映射，一个空间也常常称为一个“现实”(reality)。一个真实的网络结点在每个CAN空间中都会被分配一块区域，负责不同的数据对象，反过来，同一个数据对象(的索引)在每个空间中都会被分给一个结点，因此实际上它相当于被“复制”了多份，这种隐式的复制方案提高了数据可用性。另一方面，多空间提高了CAN的定位效率，因为在选择下一跳时，结点可以去比较它在每个空间中的邻居，从所有空间的所有邻居中选出离目的结点最近的，通常要更好。

3. 多散列

在多空间的情况下，一个数据对象可以被分配到多个区域中；实际上即使在单空间的情况下，采用多散列方法也一样能将数据对象分配到多个区域，这同样“复制”了数据。所谓“多散列”，就是采用多个散列函数对同一对象映射出多个值，或者散列函数只用一个，但对同一对象映射时加入多个“盐值”(salt)以映射出多个值，两种方法效果差不多。

4.2.4 CAN的“区域超载”

CAN中每个区域通常被分配给一个结点负责，或者在区域合并、分割后会出现一个结点负责多个区域的情况。“区域超载”则恰恰相反，是将一个区域分配给多个结点负责。

使用“区域超载”方法后，一个结点除了维护原来的路由表，还需要维护一个“区域超载列表”，其中保存着和自己共同负责同一区域的结点信息。另外，使用“区域超载”以后，当新结点A加入CAN后，如果它所映射到的点原先由结点B负责，B不一定要分割自己的区域将一半给A，而是检查负责该区域的结点数是否超过上限，如果没有超过就不需要分割区域，而是将完整的区域也交给A负责，同时A从B那里获得“区域超载列表”；但如果超过上限，区域还是要被分割的，负责原区域的那些结点也要随之分割，通常也是分成相等的两份分别负责分割后的两块区域。

采用“区域超载”方法至少可以带来性能上三方面的改进：

(1) 减少定位跳数。让多个结点负责同一区域等效于减少系统结点数。

(2) 减少每跳时延。在选择下一跳时，由于邻居区域由多个结点负责，可以从这多个结点中选出时延最短的作为下一跳。

(3) 提高容错性和可用性。首先一个区域只有在负责它的所有结点都失效时才不可达，其次该区域的数据此时可以“复制”到负责该区域的每个结点中，这已经是CAN的第三种“隐式复制”方法了。

4.2.5 CAN 中的复制与缓存

前面已经讲过 CAN 的三种“隐式复制”方法，对于普通数据而言这已经足够了，但对热点数据，CAN 采用显式的复制，通常将它复制到邻居区域中，从而分担负载。

CAN 的缓存与 Chord 类似，也是采用路径缓存方法，在定位路径上放置缓存的副本，不同之处只在于 CAN 通常只对热点数据做缓存，这里不再赘述。

4.2.6 CAN 总结

(1) CAN 采用多维空间拓扑结构，其思想简单、直观，比其他结构化 P2P 模型容易理解。CAN 空间被动态分配给其网络结点，每个结点负责一块，每个数据对象被映射到一个点，由负责该点所在区域的结点来保存。

(2) 每个 CAN 结点维护一个路由表，记录它在多维空间上的邻居信息，CAN 的定位正是基于该路由表逐步完成的。对 d 维 CAN 来说，其定位效率通常为 $O(d\sqrt[d]{N})$跳。

(3) CAN 是高容错的。这体现在其路由选择的灵活性上：CAN 中任意两个结点间存在多条路径，即使很多邻居信息失效，仍然能以较快的速度定位目的结点。

(4) 一个新结点加入 CAN 分三步完成：①自举，②寻找区域，③加入路由表。在加入过程中相应区域将一分为二，其中一半分给新结点负责。与此相应，当旧结点离开后，它负责的区域也将被其邻居接管，接管过程中可能形成很不规整的区域分布，CAN 采用“背景区域重分配”方法来调整、合并它们。

(5) CAN 采用了多种增强机制来提高系统性能，如多维度、多空间、多散列；而“区域超载”技术则对 CAN 有多方面的影响。CAN 中除了显式的复制与缓存，还有三种隐式的复制方法，这些都提高了数据可用性。

(6) 综上所述，CAN 是一个简单、容错的多维空间 P2P 模型，它具有优秀的定位效率和良好的容错性、自适应性，再加上多种增强机制的辅助，使 CAN 成为一个可扩展、高效率、高可用的结构化 P2P 模型。

4.3 Tapestry 与 OceanStore——广域的超立方体结构P2P 网络

4.3.1 Tapestry 简介

Tapestry 是一个面向广域分布式数据存取、容错的超立方体结构 P2P 模型。实际上，Tapestry 的覆盖网并不是严格的超立方体结构，而是基于 Plaxton Mesh (见文献[Plaxton et al.,1997])的网格形结构。本书中我们将 Tapestry 称为“超立

方体结构”只是出于简单、形象的考虑，也就是用“超立方体结构”近似地代表“Plaxton Mesh 结构”。

2000 年 3 月 UC Berkeley 以 Ben Y. Zhao 等为首的研究者开始 Tapestry 的开发工作，与 Tapestry 差不多同期，其他高校和组织也开始类似模型的开发，如以前讲过的 UC Berkeley 和 MIT 的 Chord、AT &T Centre for Internet Research 的 CAN，以及 4.3.2 节要讲的 Microsoft Research 和 Rice University 的 Pastry。这四个结构化 P2P 领域最著名的模型都提供了最基础的结点和对象定位功能，Tapestry、Pastry 与 Chord、CAN 的不同之处，首先在于它们都是基于 Plaxton Mesh 的覆盖网结构，其次在于它们在构建覆盖网时就考虑了拓扑一致性问题，因此定位具有很好的局部性，而 Chord 与 CAN 在构建网络时并没有考虑这些，只是在工作期间采取辅助的方法来提高局部性。Tapestry 与 Pastry 也有不同，不同之处在于数据对象的定位与放置：Tapestry 帮助用户寻找最邻近的数据副本，而 Pastry 则通过主动复制数据、将副本随机存放在网络中来提高数据可用性，并期望通过复制来使用户获得较近的副本。

2003 年 6 月 UC Berkeley 的 Tapestry 研究组发布了 Tapestry 2.0 版本软件，它使用 Java 编写，面向 Linux 操作系统，提供分布式应用的底层覆盖网络或者作为其分布式散列表。Tapestry 2.0 可以在 http://www.cs.ucsb.edu/~ravenben/tapestry 下载。

Tapestry 应用广泛，其中最著名的例子是 UC Berkeley 的 OceanStore 广域存储系统，本节下面的内容将做详细介绍。此外，基于 Tapestry 的应用还有 Bayeux、Brocade、Cashmere、SpamWatch、Warp 等，表 4.3.1 简述了这些应用。

表 4.3.1 Tapestry 的应用

Bayeux	提供高效、容错的应用层多播 [http://www.cs.ucsb.edu/~ravenben/tapestry/html/bayeux.html]
Brocade	提供界标路由(Landmark Routing) [http://www.cs.ucsb.edu/~ravenben/tapestry/html/brocade.html]
Cashmere	提供匿名路由 [http://www.cs.ucsb.edu/~ravenben/cashmere]
Fault-Tolerant Overlay Routing	基于 Tapestry 开发路由的冗余性，从而提供容错的覆盖网路由 [http://www.cs.ucsb.edu/~ravenben/tapestry/html/fault.html]
OceanStore	提供全球范围内广域的、持久性数据存取服务 [http://www.oceanstore.org]
SpamWatch	基于 Tapestry，使用基于内容相似度的搜索引擎，来提供分布式的 Spam-Filtering 系统 [http://www.cs.berkeley.edu/~zf/spamwatch/]
Warp	通过类型重定向提供快速移动服务架构 [http://www.cs.ucsb.edu/~ravenben/tapestry/html/warp.html]

4.3.2 Tapestry 路由和定位

Tapestry 的路由和定位机制来源于 Plaxton，但比 Plaxton 更为容错、优化，适应动态的网络环境。在 Tapestry 中，每个结点有它在覆盖网中的标识 nodeID，每个数据对象有其标识 objectID，objectID 也常常被称为 GUID(globally unique ID，全局唯一标识)，每条消息有其特定应用的 AID(application ID)，用以区分应用类型，类似于 TCP 协议中的端口号。与 Chord、CAN 类似，Tapestry 为每个数据对象分配一个负责它的结点，这个结点称为该对象的“根”(root)，意味着它对于这个对象特殊的重要性。如果恰好有一个结点的 nodeID 等于对象的 objectID，那么 root(objectID)＝nodeID，但通常并不存在恰好相等的 nodeID，此时 Tapestry 将对象分配给 nodeID 与 objectID 最接近的结点负责，即 root(objectID)＝最接近 objectID 的 nodeID。类似前面 CAN 的增强机制，Tapestry 也可以采用多个散列值，从而给一个对象指定多个根结点，以提高对象可用性和持久性，这里不做详述。

与 Plaxton 一样，Tapestry 采用逐位匹配的后缀路由，每一跳匹配更多的后缀，比如：

xxx8 —》xx98 —》x598 —》4598　[x 表示通配符]

上面的路由采用后缀匹配，但这与前缀匹配路由并无区别(实际上在后来的文章[Zhao et al.,2004]中 Tapestry 的设计者是采用前缀匹配的)。为适应后缀匹配路由，每个 Tapestry 结点维护一个层次化的“路由表”(routing table，也称为“邻居表”)，其中每一层代表与自身 nodeID 匹配一定位数后缀的结点。如图 4.3.1 所示，一个 8 进制 4 位的路由表，该路由表属于结点 0642，ID 有 4 位，每位是一个 8 进制数，每一列称为一“层”(level)，因为 4 位所以 4 层，从右往左分别为第 1 层、第 2 层、第 3 层、第 4 层，表示后缀与当前 nodeID 匹配位数分别为 0 位、1 位、2 位、3 位。因为采用 8 进制 ID，所以每层有 8 项，第 i 层第 j 项表示与当前 nodeID 后缀匹配位数为 $i-1$ 位、并且以 j 开头的结点(注意这里“开头”的含义是指匹配后缀的紧前一位)，如第 3 层第 5 项“x542”表示与当前结点 0642 后缀匹配两位(42)、并且以 5 开头的结点。

基于上述路由表，Tapestry 的后缀匹配路由变得高效、容错，路由的第 n 跳所到达的结点通常与目的结点 ID 至少匹配 n 位后缀，图 4.3.2 是一个简单的例子。假设已到达结点与目的结点 ID 后缀匹配刚好 n 位，为了找到下一跳结点，需要在当前结点的路由表第 $n+1$ 层中，查找与目的结点 ID 匹配更多位后缀的结点作为下一跳。通常可以找到这样的结点，但如果实在找不到匹配更多位的结点，就找数值上更近的，这常常意味着定位即将完成。

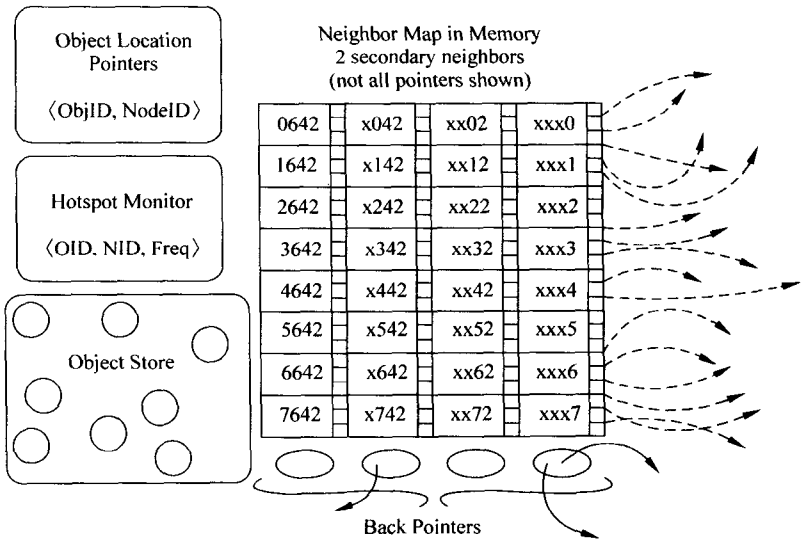


图 4.3.1　结点 0642 的状态信息，包括其对象索引、热点数据管理器、对象存储空间、路由表（其中包含主项、次要项、反向指针）

（图片来自[Zhao et al.,2001]）

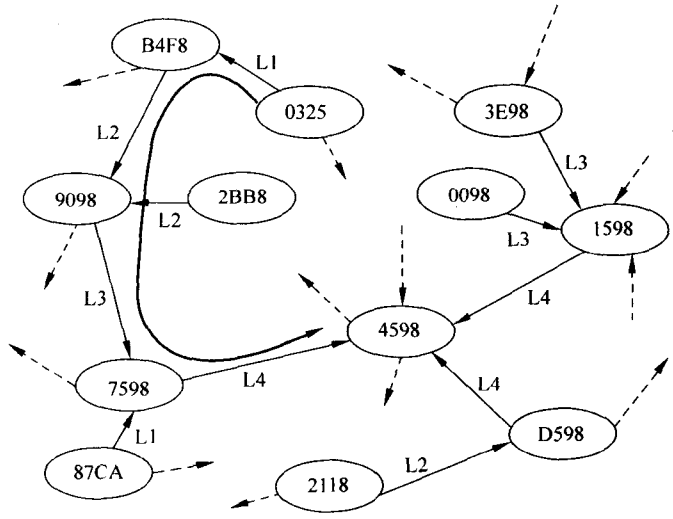


图 4.3.2　Tapestry 路由的例子，结点 0325 要发送一条消息给结点 4598，粗线标明了路由的每一跳

（图片来自[Zhao et al.,2001]）

到这里我们可以看出：Tapestry 的路由机制可以保证在一个有 N 个结点的网络中，任何一次定位一般都能在 $\log_B N$ 跳之内完成，其中 B 为 ID 编码的进制（也称“基”，Base），如 8 进制则 $B=8$。Tapestry 的路由表项数相比 Chord、CAN 要略多一些，因为它有 $\log_B N$ 层，每层 B 项，故共需要 $B * \log_B N$ 项，虽然多了一个倍数 B，但路由表项数仍然是很少的。同时，Tapestry 路由保证了路由的容错性，因为与 CAN 相似、两个结点间的路径可以有很多条，就好比在超立方体中，从一个点要走到另一个点，如果按照后缀逐位匹配的方法走，可以有很多条不同的路径作为候选，这是我们称 Tapestry 为“超立方体结构”P2P 模型的主要原因之一。不过要注意的是：Tapestry 路由本质上不同于超立方体路由，其限制要比超立方体路由多很多，两者只是类似而已。

如果某个 Tapestry 结点 S 要向网络插入一个数据对象 O，那么最重要的是，它要把该对象的索引放到 O 的根结点上（上文讲过 O 的根结点就是网络中与 O 的 ID 相等或最近的结点）。为了找到根结点，S 向网络中发送一条以 objectID 为目的地的消息，其中还包含有对象索引信息如〈objectID(O)，serverID(S)〉。路由路径上的每个结点收到该消息后会将该消息送给下一个与 objectID 更匹配的结点，直到某一步没有办法更匹配，实际上也就是到了与 objectID 最近的结点，即找到了对象 O 的根。在上面过程中，消息所走过路径上的所有结点，当然也包括对象 O 的根结点，都会保存 O 的索引信息，这类似于前面在 Chord、CAN 中所讲的“路径缓存”。另一方面，由于复制的使用，同一个对象 O 可能有多个副本放置在网络中，共享相同的根结点，因此一个 Tapestry 结点可能会保存 O 的多个副本的索引，指向不同的位置，以后当用户查询 O 到达该结点时，可以从 O 的多个索引中找自己认为最近、最合适的来获取数据，这是上文所述 Tapestry 自动帮助用户获取最邻近副本的原因所在。

如果某个 Tapestry 结点要查询一个数据对象 O，它也发送一条以 objectID 为目的地的定位消息，该消息按照后缀匹配的方法逐步路由，在每一步都会检查当前结点是否已经保存了 O 的索引，如果是则定位结束。如果中间结点都没有找到 O 的索引，最后消息必将到达 O 的根结点，所以说，根结点的功能在于确保对象定位成功，这也是它对 O 有特殊重要性的原因。根结点常被称为对象 O 的“代理”结点，意在说明根结点负责了该对象的定位。

4.3.3 Tapestry 动态结点算法

为了保持路由表的更新和定位的容错性，Tapestry 路由表中有一种称为“反向指针”（back pointer）的项，它指向那些把自己作为路由表项的结点，利用它来周期性地发送“心跳”（Heartbeat）消息给“反向结点”，告诉它们自己的存在。如果在某个周期内“反向结点”没有收到“心跳”消息，就认为该路由表项失效。此外，

Tapestry路由表中每项并不只保存一个结点，而是保存一个“主项”和两个“次要项”，当主项结点失效时，就使用次要项结点。还有一个特殊的设计是：Tapestry发现路由表某项失效后通常并不立即替换它，而是过一段时间再检测一次它是否在线，如果还不在才替换，这被称为“二次机会”，以适应网络中常常出现的结点突然离开但很快又加入的“闪断”现象。

新结点 N 加入 Tapestry 网络需要做三项工作：①初始化自己的路由表；②告诉其他结点自己的到来，更新它们的路由表；③该由 N 负责的对象索引必须从原来结点移交给 N。

JOIN STEP1：*初始化路由表*

假设新结点 N 通过某种途径联系到一个 Tapestry 现存结点 G，N 首先通过 G 发送以 N 自己为目的地的消息。在消息路由的过程中，假设第 i 步到达结点 Hi，初始的 $H0=G$，那么根据 Tapestry 的后缀匹配路由算法，N 和 Hi 应该共享长度 i 的后缀，所以 N 从 Hi 那里获得路由表的第 $i+1$ 层项是合适的。在将 Hi 的第 $i+1$ 层项复制到 N 自己的路由表后，N 要做一个优化：首先对每一项而言，如果次要项结点对 N 来说比主项结点更近，那么次要项成为主项；随后 N 查找新主项结点的路由表相关项，比较 N 到每项的距离，看是否有更好的选择；上述优化过程会重复进行直到收敛，也就是说做到不再有更好的选择为止。

JOIN STEP2：*更新其他结点路由表*

N 通过 G 发送往 N 自己的消息，在 $\log N$ 跳之内将最终到达 N 的根结点 R。R 收到该消息后，为了通知其他结点更新路由表，首先计算它和 N 匹配的后缀位数 p，然后通过自己路由表中的“反向指针”，告诉那些与 R 也匹配 p 位后缀的反向结点：如果 N 对它们是合适的，那么它们在自己的路由表中添加 N 或者用 N 替换原来的项。

JOIN STEP3：*对象索引移交*

在最初设计 Tapestry 时，当 N 发送往 N 自己的消息在 $\log N$ 跳之内最终到达 N 的根结点 R 时，认为 R 正是需要移交对象索引给 N 的结点，所以 R 将自己的对象索引中该由 N 负责的部分移交给 N，对象索引移交即结束。然而，Tapestry 不同于 Chord，它没有 Chord 那样严格的结构和必须维持的不变属性，因此，仅让根结点 R 移交对象索引给 N 是不够的。在后来的 Tapestry 设计中，将对象索引移交放在 JOIN STEP2 中完成：对于 STEP2 中 R 所通知的那些反向结点，它们不仅用 N 更新自己的路由表，还将自己的对象索引中应由 N 负责的那部分移交给 N。很明显，新方法比旧方法要好得多。

如果结点 N 正常离开，它会告诉自己的反向结点在路由表中去掉 N。同时，N 要把自己负责的对象索引分别移交给它们的新根结点。

如果结点 N 不正常离开，则通过这一小节开头所讲的周期性检测方法来检查

到结点失效。确认结点 N 失效后，将通过与结点加入过程中类似的多播方法来更新路由表，而对于对象索引，则采用“软状态重发布”的方法：所谓“软状态”(soft state)指对象索引都是暂时性的，而“重发布”指让数据对象的拥有者(服务器)定期重新发布自己的对象索引信息，实际上这个方法最主要的目的是为数据对象更新根结点。

4.3.4 Tapestry 体系架构

图 4.3.3 展示了 Tapestry 的体系架构，下面从底层到高层描述各个部分的含义：

(1) 传输协议(Transport Protocols)：封装了网络传输层，相当于覆盖网与物理网的中间层，典型地可以使用 TCP 或者 UDP 协议。

(2) 邻居链接管理(Neighbor Link Management)：向上层提供安全但不可靠的数据报服务，比如长消息的分片和组合；另外该层还负责持续的邻居链接管理和更新，如周期性的邻居结点失效检测、时延估计等，当检测到状态变化时通知上层来处理。

(3) 路由(Router)：管理 Tapestry 结点路由表(Routing Table)和对象指针数据库(Object Pointer Database)。它检查所收到消息的目的地，决定路由的下一跳；在新结点加入、旧结点离开时更新对象索引信息。

(4) 应用接口/提供给上层的 API(Application Interface/Upcall API)：这是 Tapestry 提供给其高层应用的接口。三个最重要的接口函数分别是：

① DELIVER(ID, AID, Msg)——接收到要发往 ID 的消息 Msg，而当前结点

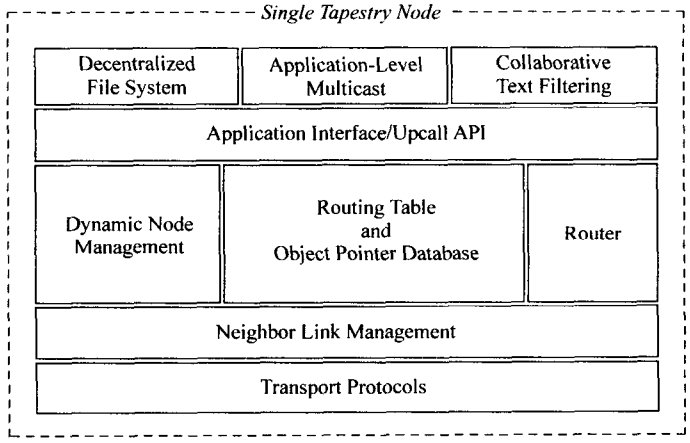


图 4.3.3 Tapestry 组件架构图

(图片来自[Zhao et al., 2004])

正是路由的终点(最后一跳)，此时 DELIVER(...)被调用。

② FORWAED(ID,AID,Msg)——将以 ID 为目的地的消息 Msg 往前传，它通过调用下面的 ROUTE(...)函数来完成。

③ ROUTE(ID,AID,Msg,NextHopNode)——将以 ID 为目的地的消息 Msg 路由到下一跳结点 NextHopNode。

其中 ID 指一个 nodeID 或者 objectID，AID 代表某种应用类型。

(5) 分布式文件系统/应用层多播/协同文本过滤(Decentralized File System/Application-Level Multicast/Collaborative Text Filtering)：基于 Tapestry 的各种上层应用，实际上并不限于这三种。

4.3.5 Tapestry 总结

(1) Tapestry 是一个面向广域分布式数据存取、容错的超立方体结构 P2P 模型，它在构建网络时即考虑了拓扑一致性。Tapestry 最具特色的功能在于帮助用户寻找最邻近的数据副本。

(2) Tapestry 中每个结点、每个数据对象、每条消息(更确切地说是每种应用)都有一个全局唯一的 ID，每个数据对象有一个负责它的结点，称为该对象的“根”，它是网络中 nodeID 与 objectID 最匹配的结点。

(3) 每个 Tapestry 结点维护一个路由表，其中第 i 层第 j 项表示与当前 nodeID 后缀匹配位数为 $i-1$ 位、并且以 j 开头的结点，Tapestry 正是基于这样的路由表实现了效率为 $O(\log N)$跳的后缀匹配路由，并且其路由具有高容错性。

(4) Tapestry 路由表还维护一种称为“反向指针”的项，很多重要操作如结点加入、离开、失效检测和修复都用到了它。新结点加入时，首先从其定位路径上的所有结点那里逐层获得路由表，然后通过反向指针通知其他结点更新路由表项，那些被通知到的结点还会将相关的对象索引移交给新结点负责。

(5) Tapestry 体系架构分为 5 层，分层有助于高层应用的开发和各层的优化完善。基于 Tapestry 已经开发出了多种应用，如分布式文件系统 OceanStrore、应用层多播 Bayeux、协同文本过滤 SpamWatch 等。

4.3.6 OceanStore 简介

OceanStore 是一个基于 Tapestry 的分布式数据存取系统，其目标是提供全球范围的广域、持久性数据存取服务。任何一台计算机都可以加入到 OceanStore 系统中来，向系统贡献它自己的存储空间，同时获得他人存储的内容。用户只需要向 OceanStore“服务提供者”(相当于一个客户端软件)提交请求，服务提供者之间会自动完成该请求，对于用户来说这个过程是透明的。

OceanStore 对数据提供传统的复制、缓存功能，以提高存取速度和可用性，同

时带来的“通信局部化”也有效地减少了网络拥塞。OceanStore 建立在一个广域、动态、不可靠的网络基础上，所以它假设系统中每个服务器都不可靠，都有泄露信息或者成为恶意结点的可能，因此它对所有数据、元数据等都提供了加密或者认证的功能。由于多个副本的存在，且在 OceanStore 中数据更新是经常有的，为了保持副本间的强一致性，OceanStore 采用了“拜占庭式容错提交协议”(Byzantine Fault Tolerant Commit Protocol)，这个协议会带来很大的网络开销，因此 OceanStore API 允许用户减弱一致性要求以换取性能。

OceanStore 的数据持久性是通过基于版本的深度归档存储方案(version-based deep archive storage)来实现的：每个数据对象的每个版本都以持久、只读的方式存储在系统中，并且以“冗余编码”(erasure code)的方式分片冗余地存储在网络多个结点中，这些分片中只要一部分就可以重构原文件。因此，OceanStore 中数据是高可用、高持久性的。

在广域、动态的网络中提供数据存取，最重要的需求是存取性能和容错性。OceanStore 通过称为“内省”(introspection)的机制来辅助系统工作：内在的事件管理器收集和分析诸如使用模式、网络行为、资源可用性这样的信息，从而 OceanStore 可以通过预先迁移数据到可用区域，以适应那些突发事件的影响，如地域性断开(regional outage)、DoS 攻击等。

UC Berkeley 的研究者早在 2000 年之前就开始了 OceanStore 的研究工作(图 4.3.4)，其构想可能比 Tapestry 还要早，虽然它以 Tapestry 为底层基础。OceanStore 的正式论文[Kubiatowicz et al.，2000]发表于 ASPLOS'00 会议，虽然这篇文章只提出了广域、持久性数据存取的构想，但它对 P2P 的应用产生了很深的影响，此外还有两篇关于 OceanStore 的论文也很重要：[Bindel et al.，2000]和[Wells，2002]。起初 OceanStore 的各个功能构件都是独立实现的，2003 年 OceanStore 研究组在 FAST'03 会议上发表了 OceanStore 的原型 Pond 的论文

The OceanStore Project

Providing Global-Scale Persistent Data

UC Berkeley Computer Science Division

- Project overview
  A brief overview of the OceanStore project.
- People
  Faculty, staff, and students working on the OceanStore project.

图 4.3.4 OceanStore 网站[http://www.oceanstore.org/]，2006 年

[Rhea et al.,2003]，Pond 实现了 OceanStore 所预想的大多数功能。2004 年 6 月 OceanStore 在 SourceForge 上发布了原型软件及其源代码，可以链接 http://oceanstore.sourceforge.net。

4.3.7 OceanStore 的命名机制和存取控制

在 OceanStore 中数据对象是最基本的单元，它类似于文件系统中的一个文件。数据对象以只读文件版本的方式按序保存在系统中，原则上每个对象的每个版本都是永久保存的，虽然只有最新版通常才有意义。“版本”方法给 OceanStore 的数据对象复制、缓存带来了很大的帮助，我们只要知道哪个是新版就可以，没有必要去替换旧版保持一致性，而且旧版的存在在某些情况下也是很有用的。

图 4.3.5 描述了 OceanStore 的数据对象组织架构。每个对象的每个版本包含着该版本数据和元数据（如目录）以及指向其前一个版本的指针。每个版本都有自己的标识 $VGUID_i$（Version GUID，版本全局唯一标识，i 表示是对象的第 i 版）。对象的所有版本通过“反向指针”（图中 backpointer，注意跟 Tapestry 中的“反向指针”含义不同）连成一个流，这串序列合起来有一个标识 AGUID（Active GUID，有效全局唯一标识），AGUID 唯一标识了 OceanStore 中一个有效的数据对象。每个数据对象版本由许多块组成，每块有自己的标识 BGUID（Block GUID，块全局唯一标识），这些块自顶向下组织成一棵类似 B 树的结构：树根叫做“根块”（root block），其中保存着该版本的元数据 M 和向下的指针，根块的 BGUID 通常被作为该版本的 VGUID。中间块叫做“间接块”（indirect blocks），其中只保存向下的指针，可能指向新的中间块或者底层的数据块。树底层的叶子叫做“数据块”（data

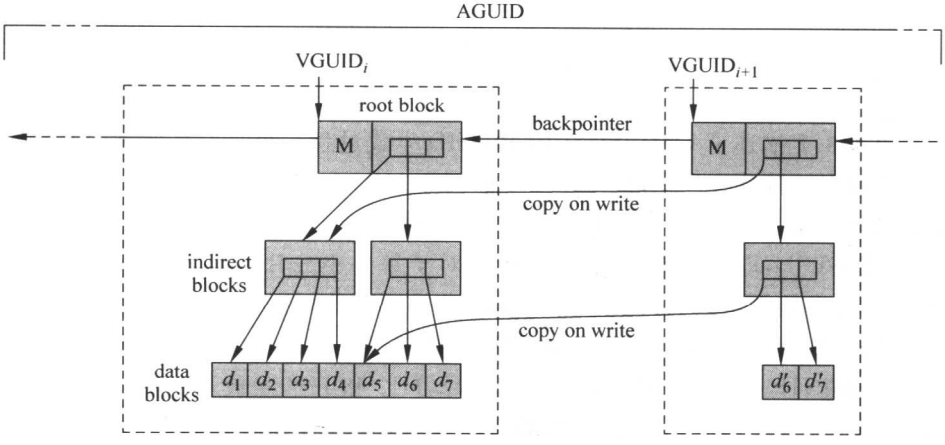


图 4.3.5 OceanStore 数据对象组织架构

（图片来自[Rhea et al.,2003]）

block)，保存真正的数据。作为索引的根块和中间块里的“指针”，实际上是其子块的 BGUID，因为在 OceanStore 中，任何数据只要有了它的 ID 就能找到并获得它。另外，块的存在使得对象版本之间可以共享数据，只要保存数据块的 BGUID 就可以了(图中 copy on write)。

图 4.3.6 描述了 OceanStore 有效对象的结构，AGUID 是由对象拥有者的公钥和可读对象名拼接起来的安全散列值(图中 AGUID 表示成了“GUID”)，这种生成方法既避免了对象名冲突，又起到了完整性检查的作用。有效对象结构中的其他部分将在下文中陆续讲到。

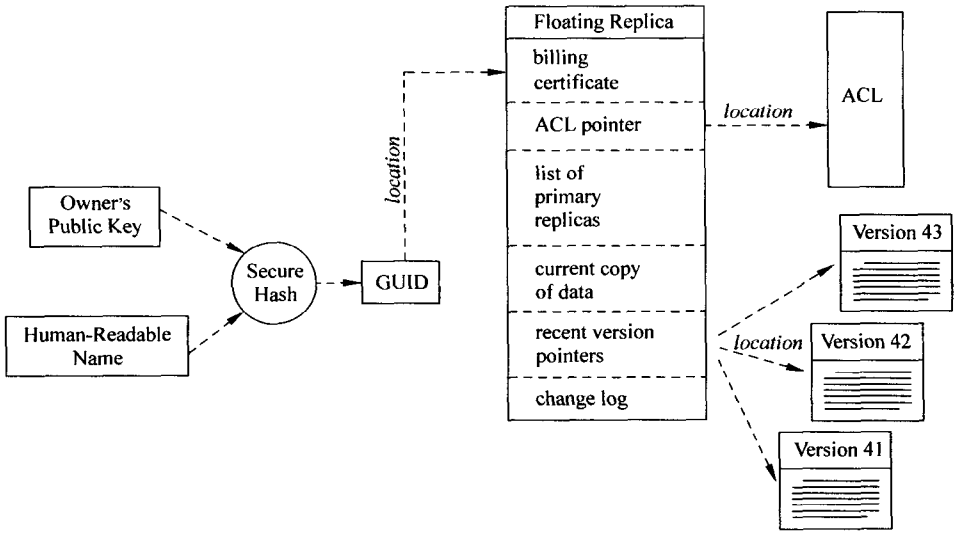


图 4.3.6 OceanStore 有效对象结构

(图片来自[Bindel et al.,2000])

图 4.3.7 描述了 OceanStore 的 BGUID 生成过程。每个数据块被分片存储，如图中数据块被分 4 片，每片产生一个散列值 H1、H2、H3、H4，然后 H1 和 H2 生成 H12，H3 和 H4 生成 H34，H12 和 H34 生成 H14，H14 和数据块本身的散列值 Hd 生成 GUID，即该数据块的 BGUID。对于每一个分片而言，除了存储分片数据，还需要存储它从底层到顶层所需要的兄弟散列值(约 $\log N$ 个，N 为分片数)，如图中分片 1 存储 H2、H34 和 Hd，这样如果要验证分片 1 的数据完整性，只要计算其散列值 H1，与 H2 合起来可以生成 H12，H12 和 H34 生成 H14，如果最后一步 H14 和 Hd 生成的散列值等于 BGUID，那么就证明该分片数据是完整、真实的。

因为 OceanStore 中同一数据对象有多个版本共存，因此必须有一种机制来从对象的 AGUID 安全映射到其最新版本的 VGUID。这样的映射方法有两种：方

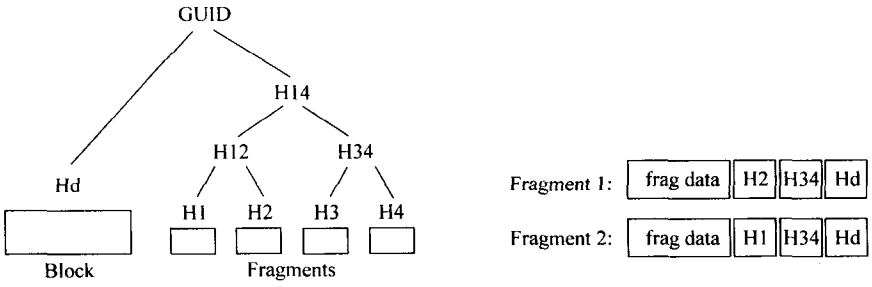


图 4.3.7 OceanStore 中 BGUID 的生成
（图片来自[Wells,2002]）

法一，每个数据对象在系统中对应一个“主环”(primary ring)，由多台服务器组成，它们之间使用“拜占庭一致性协议”来维护 AGUID 到最新 VGUID 的映射，并将用户发出的对象更新操作序列化执行。方法二，作为对方法一的补充，如果某个对象没有这样的主环，AGUID 到最新 VGUID 的映射将被存储在称为“墓碑”(tombstone)的结构里。“墓碑”这个名字源于“主环”在“死亡”前将映射放到其中。图 4.3.8 描述了“墓碑”的结构，我们从上往下看：①最顶层是 AGUID(这里写成 Archive GUID，其实就是 Active GUID)，正如前文所说它是②用户公钥和对象名的安全散列值；③中间层是负责方(RP，responsible party)的公钥和 GUID；④次底层是对象主环(PR，primary ring)的公钥和私钥密文；⑤最底层是对象最新的 VGUID。图中“Signature”表示数字签名，以用来逐层检验信息的真实性。

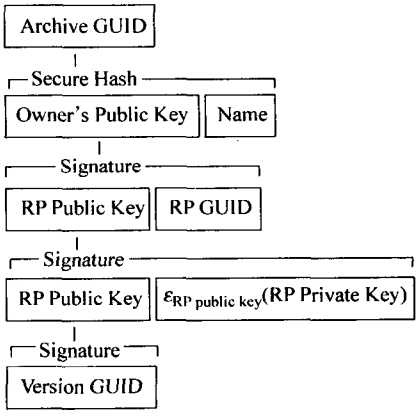


图 4.3.8 OceanStore“墓碑”结构
（图片来自[Wells,2002]）

类似文件系统的“读写控制”，OceanStore 对系统中数据对象提供两种原始类型的“存取控制”：读者限制和写者限制。更复杂的存取控制如“工作组方式”都基于这两者。

读者限制：为了阻止不合法的读者，OceanStore 中所有非完全公开的数据都被加密，密钥分发给那些允许读的用户。如果要撤销原来发给的读允许，对象拥有者要么删除数据对象，要么用新密钥加密原对象，通常后者更为可取。

写者限制：为了阻止不合法的写者，OceanStore 要求所有的写操作都必须签名，这样，那些行为良好的服务器和客户就可以通过存取控制链(ACL，access

control list)来验证写操作。对象的存取控制链是由对象拥有者所签名的、授予特定用户对该对象的操作特权。

4.3.8 OceanStore 的路由和定位算法

OceanStore 采用了两种路由和定位算法，前一种称为“概率查询”，它是局部性的，因此非常快，但不保证成功；后一种称为“全局查询”，采用 Tapestry 后缀匹配路由算法，是全局性的，速度较慢，但保证成功。OceanStore 通常先尝试“概率查询”，如果失败，再做确定性的“全局查询”，从长远看来这种方法是很有利的。(这两种方法是否让你回想到，计算机读取数据时，是先尝试从高速缓存中找，如果找不到，再到内存或外存中找。)

OceanStore 的“概率查询”能够快速定位局部性的邻近数据对象，它采用一种称为“Attenuated Bloom Filters”的数据结构。一个深度为 D 的 Attenuated Bloom Filter 可以看成是一个含有 D 个普通 Bloom Filter 的数组，其中第 1 个 Bloom Filter 是结点本地对象的记录，第 i 个 Bloom Filter 是在任意路径上离当前结点距离为 i 的所有结点的 Bloom Filters 的并集。OceanStore 为网络中每条有向边保存一个 Attenuated Bloom Filter 以表示沿该边可以定位到的对象信息，查询消息在 Bloom Filter 的指导下沿着有向边路由。图 4.3.9 展示了一个“概率查询”的例程：n_1 查找对象 X，n_1 的本地 Bloom Filter 显示它没有所要的对象(箭头 1)，但是它的有向边的 Filter 显示 n_2 可能是一个路由到 X 的中间结点(箭头 2)，因此查询请求被发往 n_2，n_2 的 Bloom Filter 显示 n_2 没有所要的对象(箭头 3)，而其邻居 n_4 也没有(箭头 4a)，但邻居 n_3 可能有(箭头 4b)，查询请求又被发给 n_3，刚好找到了对象 X(箭头 5)，概率查询成功。

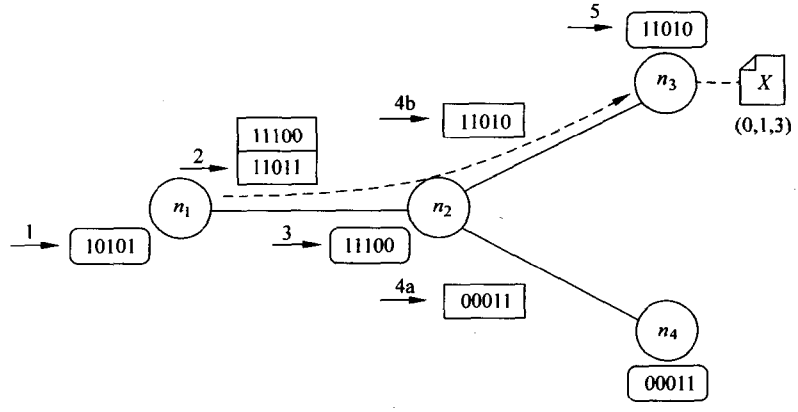


图 4.3.9 OceanStore 的概率查询过程

(图片来自[Kubiatowicz et al.,2000])

OceanStore 的全局查询采用 Tapestry 后缀匹配路由算法，这在 Tapestry 中已经详细讲过，这里不再赘述。

4.3.9　OceanStore 的更新模型

在 OceanStore 中，任何一个数据对象都对应一个“主副本环”(primary replica，也称为“主环”或者“内环”)，它是数据对象归档存储、更新、最新版本获得的核心设施。如果用户要更新对象，它首先发出更新请求，通过 Tapestry 底层网络将请求发到对象主环，主环服务器之间序列化所收到的更新请求并执行它们。然后，主环服务器将新数据对象深度归档存储起来(4.3.10 节将详细讲述“深度归档存储”)，并通过“分发树”(dissemination tree)将更新分发到对象的“次要副本”(secondary replica)服务器以更新缓存。前一项深度归档存储工作为的是提供持久可用的数据对象，后一项分发操作则为了加快数据定位和获取速度。更新过程如图 4.3.10 和图 4.3.11 所示。

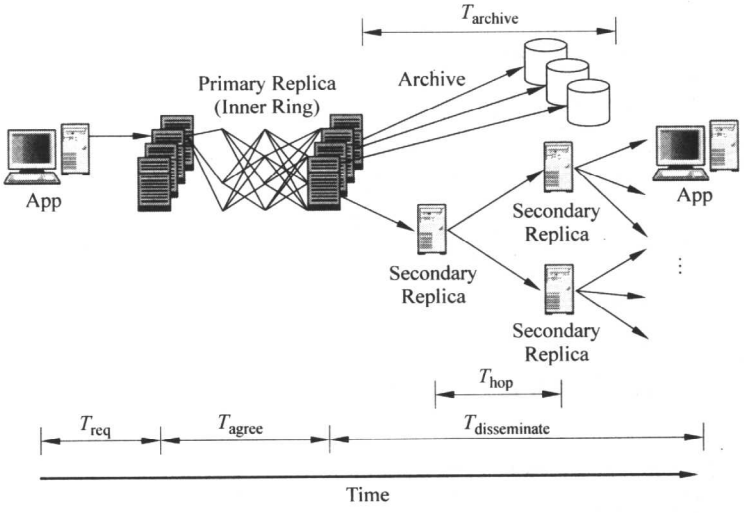
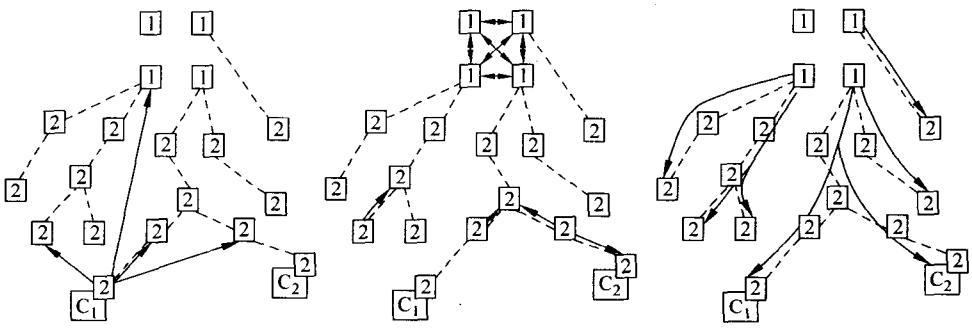


图 4.3.10　OceanStore 更新过程

(图片来自[Rhea et al.，2003])

上文多次提到“拜占庭协议”[Lamport et al.，1982]，这里简单地描述一下。所谓“拜占庭协议”是一个分布式的决策过程，假定决策由多个不可靠的人来协商决定，那么只要其中超过 2/3 的人遵循协议就可以最终达到相同的决策。也就是说，如果有 $3f+1$ 个人来决定一件事，为了达成一致其中不能有超过 f 个人不遵循协议。“拜占庭协议”虽然保证了决策的一致性，但它需要发送很多消息，通常是决策者总数的平方级。对 OceanStore 而言，主副本环服务器如果太多，传统的“拜



(a) C1发送数据更新请求　(b) 主副本环服务器之间通过"拜占庭一致性协议"执行更新　(c) 主副本环服务器通过"分发树"分发更新到次要副本服务器

图 4.3.11 OceanStore 数据更新路径-更为具体的例子。其中 1 表示主副本环服务器，2 表示次要副本服务器

(图片来自[Kubiatowicz et al.,2000])

占庭协议"就不合适，因此，OceanStore 设计了多种类似的协议来取代它，其中之一便是"Castro and Liskov 算法"。这个算法有两个版本：一种是使用公钥机制进行认证，另一种是使用基于对称加密的 MAC(message authentication codes，报文鉴别码)机制来认证。OceanStore 结合了算法的两个版本：对于主环服务器更新采用 MAC 机制，虽然它需要更多的密钥，但它的计算要快得多；而对于与其他服务器的更新则采用公钥机制，虽然计算量大一些，但需要的密钥很少。除了"Castro and Liskov 算法"，OceanStore 还设计了称为"Proactive Threshold Signatures"的一致性协议，详见 OceanStore 原型 Pond 的论文[Rhea et al.,2003]。

由于 OceanStore 工作于不可靠的网络，每个结点都有成为恶意结点的可能，因此所有数据块都被加密存放。尽管在没有解密密钥的情况下，没有办法看到数据块的内容，但是仍然可以对数据块进行操作，典型的操作如：块替换(replace-block)、块插入(insert-block)、块删除(delete-block)、块附加(append-block)。以最基本的块插入和块删除为例，OceanStore 将块分成两类："数据块"和"索引块"，前者包含真实数据，后者只相当于一个指针。如图 4.3.12 所示，原来有 3 块，要插入新块 Block41.5，它应该在 Block41 与 Block42 之间。首先 Block41.5 被附加到最后一块 Block43 之后，而原来的 Block42 位置被替换成一个索引块，Block42 自己则被附加到 Block41.5 之后，替换的索引块中保存两个指针，一个指向 Block41.5，另一个指向 Block42。整个过程中没有用到块的内容，但仍然保留了块的基本顺序。相应地，删除一个块只需要将要删除的块替换成一个含空指针的索引块即可。

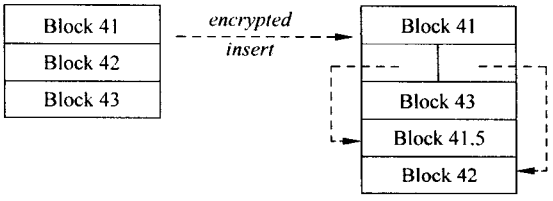


图 4.3.12　基于密文的数据块插入操作
（图片来自[Kubiatowicz et al.,2000]）

4.3.10　OceanStore 的深度归档存储

深度归档存储是 OceanStore 提供持久性数据存储的核心机制，它使用“冗余编码”(erasure code)的方式将数据对象分片冗余地存储在网络的多个结点中，只要获得这些分片的一部分就可以重构原文件。

“冗余编码”(erasure code)是一种提高数据可用性的优秀的数学编码方法。假设编码前数据被分成互相独立的 n 片，“冗余编码”将这 n 片转换成更多相关的分片，比如 $2n$ 片、$4n$ 片，原来分片数与编码后分片数的比称为“冗余编码率”(erasure code rate)。这些冗余的、相关的分片散布到网络中后，任何时刻只要用户能取得其中任意 n 片，就可以重构原来的对象。

采用“冗余编码”比采用简单的复制方法对数据可用性、持久性的提高要好得多：假设系统中共有 n 个结点，某个时刻有 m 个结点失效，f 是对象的分片数，r_f 是最大容许的不可用分片数(所谓“最大容许”指丢失掉的分片数不足以导致对象不能重构)，那么对象可用的概率符合下面公式：

$$P = \sum_{i=0}^{r_f} \frac{\binom{m}{i}\binom{m-n}{f-i}}{\binom{n}{f}}$$

假定 n 取 100 万，m 取 10 万，即 10%的结点失效，简单复制方法提供的对象可用性是 99%；而采用“冗余编码率”为 1/2 的冗余编码，在消耗相同存储容量的前提下，对象可用性将达到 99.9994%，如果“冗余编码率”取 1/4，对象可用性将比这个值高更多。

4.3.11　OceanStore 的内省优化

OceanStore 内省优化基于两点考虑：首先是因为网络动态性，保持结点状态的自适应更新；其次是因为网络异构性，利用复制、集群等方法充分利用高能力的结点。如图 4.3.13 所示，OceanStore 内省优化由三个循环的操作组成：①观察

(observation)，监控系统活动并记录下活动信息；②优化(optimization)，利用观察得到的信息来调整下面的计算；③计算(computation)，这代表实际的系统工作，如通信、数据交换、本地计算等。

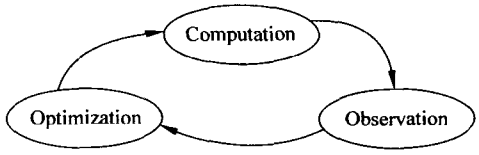


图 4.3.13 OceanStore 内省过程
(图片来自[Kubiatowicz et al.,2000])

更具体地，上面的内省优化过程可以描述成图 4.3.14 所示的架构。结点事件流(Event Stream)不断发给事件处理器(Event Handlers)，事件处理器将事件信息总结、提炼放到数据库(DB)中。周期性的深度分析(Periodic In-Depth Analysis)每隔一段时间取出数据库中的事件信息，一方面基于历史和经验信息发出本地优化命令(Issue Local Optimizations Commands)以做局部的内省操作，另一方面将其事件信息进一步总结发给更广范围内的结点做进一步处理(Forward Aggregates for Further Processing)。

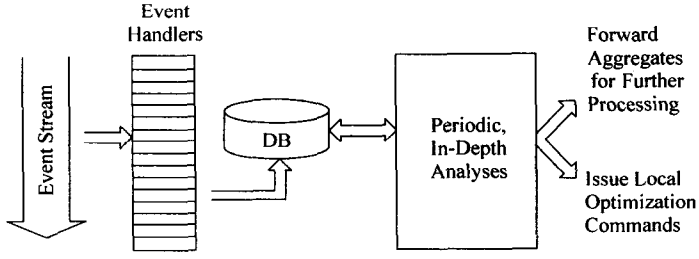


图 4.3.14 OceanStore 内省架构
(图片来自[Kubiatowicz et al.,2000])

"集群"对于分布式的 P2P 系统来说通常都是有益的，OceanStore 通过称为"集群识别"(cluster recognition)的方法来识别紧密相关的数据并将它们集群。每个结点每次存取数据都会自动启动事件处理器，该事件处理器不断构建一张代表数据间语义距离的图。集群算法每隔一段时间运行一次，它使用这张图来检测数据间可能的集群，最后将检测到的集群信息发给一个全局的分析层，由它来发布和描述已有集群的信息。

OceanStore 中的数据对象除了持久性的归档，更多地以所谓"流动副本"(floating replica)的方式存在，它虽然不稳定，但对它的存取比对数据归档要快得

多。OceanStore的副本管理必须不断调整“流动副本”的数目和位置，以取得更高的数据存取效率。前面所讲的“事件管理器”监控客户数据请求和服务器负载，如果发现某个区域的客户多次请求某数据对象，就发送一个新副本到那个区域；如果发现服务器负载过重，则创建新副本到服务器的邻居以分担负载。除了上面所说的功能，副本管理还能主动地删除不被用到的副本，或者在检测到某个区域结点失效状况严重时，预先将副本迁移到别处。

4.3.12 OceanStore总结

(1) OceanStore是一个基于Tapestry的分布式数据存取系统，其目标是提供全球范围的广域、持久性数据存取服务。

(2) 在OceanStore中数据对象是最基本的单元，以只读文件版本的方式按序保存在系统中。为了标识和验证数据，OceanStore使用AGUID来标识可用对象，VGUID标识对象的各个版本，BGUID标识数据块或元数据块，这些ID之间以类似B树的方式组织起来。

(3) OceanStore采用了两种路由和定位算法：前一种称“概率查询”，是局部的，因此非常快，但不保证成功；后一种称“全局查询”，采用Tapestry后缀匹配路由算法，速度慢，但保证成功。通常先尝试“概率查询”，如果失败再做“全局查询”。

(4) 在OceanStore中，任何一个数据对象都对应一个“主副本环”，它是数据对象归档存储、更新、最新版本获得的核心设施。用户要更新对象，首先发出请求到对象主环，主环服务器之间序列化所收到的更新请求，然后，一方面将新数据深度归档存储，另一方面将更新分发到对象的“次要副本”服务器以更新缓存。

(5) OceanStore建立在一个广域、动态、不可靠的网络基础上，系统中每个结点都不可靠，因此它对所有数据提供加密或者认证。由于多个副本的存在，为了保持副本间的强一致性，OceanStore采用了“拜占庭式容错提交协议”，这个协议会带来很大的网络开销，因此OceanStore API允许用户减弱一致性要求以换取性能。

(6) OceanStore的数据持久性是通过深度归档存储方案来实现的，数据以“冗余编码”的方式分片冗余地存储在网络中多个服务器中，只要有这些分片中的一部分就可以重构原文件，因此数据是高可用、高持久性的。

(7) 为提高存取性能和自适应性，OceanStore通过称为“内省”的机制来辅助系统工作。内在的事件管理器收集和分析网络信息，从而对紧密相关的数据对象集群，并管理数据对象的“流动副本”，通过调整其数目、位置来提高存取效率，或者预先迁移对象以应对网络失效。

最后，我们以下面这幅图来描述OceanStore的理想。图4.3.15中“pool”表示分布在世界各地的数据“池”，而不管用户在哪里，使用的是什么设备(个人电脑、手机、PDA等)，连接到哪个数据池，通过OceanStore的组织架构，他都能访问到

全球任意位置的数据。

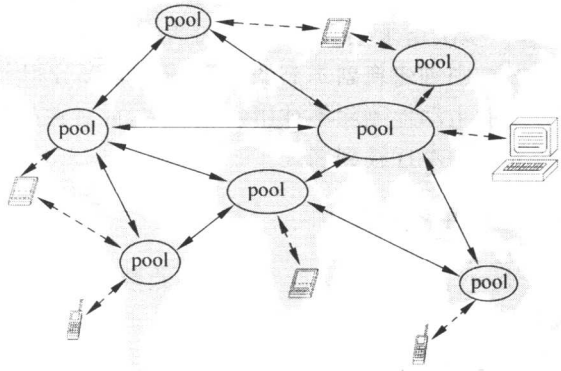


图 4.3.15 OceanStore 的理想——跨越全球的分布式持久数据存取
（图片来自[Kubiatowicz et al.,2000]）

4.4 Pastry 与 PAST——容错的混合式结构 P2P 网络

4.4.1 Pastry 简介

Pastry 是一个容错、高效、可扩展的混合式结构 P2P 网络，同时它也是众多分布式应用的底层架构。Pastry 结合了环形结构与超立方体结构（实际是 Plaxton Mesh）的优点，提供高效的查询路由、确定性的对象定位和独立于具体应用的负载平衡，此外，它还提供了复制、缓存、错误复原等增强机制以提高系统性能。

Microsoft Research 和 Rice University 的 Antony Rowstron 与 Peter Druschel 等人大致在 2000 年开始 Pastry 的设计及其应用的开发，Pastry 正式的论文[Rowstron & Druschel,2001a]发表在 Middleware'01 会议上，其知名度仅次于 Chord 和 CAN，也是非常有影响力和实用的 P2P 模型。4.3 节已经提到过 Tapestry 与 Pastry 的相似之处：两者的覆盖网结构和路由方法都来源于 Plaxton Mesh，但都做了一些改进，并且在构建覆盖网时都考虑了局部性因素。两者不同之处主要在于副本的放置：Tapestry 的目的是尽可能找到最近的副本，Pastry 则希望副本能均匀、分散地放置，从而在全局上有好的效果。还有其他不同之处，比如 Pastry 结点状态除了包含类似 Tapestry 的路由表外，还有叶集 L 和邻居集 M，叶集维护 ID 邻近的结点，邻居集则维护物理上邻近的结点。就增强机制而言，Pastry 采用了传统的 ID 邻近复制和路径缓存，这和 Chord、CAN 很像。

Pastry 的两个可用版本目前可以上网下载到：一个是 Rice University 开发的 FreePastry，另一个是 Microsoft Research 开发的 SimPastry/VisPastry。

Pastry 和 Tapestry 一样有着广泛的应用，见图 4.4.1，其中最著名的是 PAST 归档存储系统，本节下面将详细讲述。表 4.4.1 简单描述了 Pastry 的各种应用。

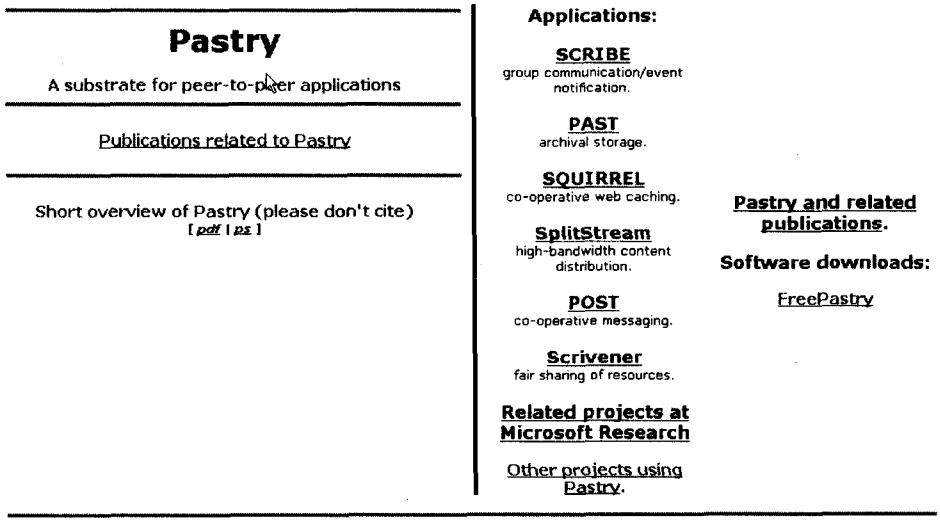


图 4.4.1　Pastry 网站[http://research.microsoft.com/~antr/Pastry]，2006 年

表 4.4.1　Pastry 的各种应用

SCRIBE	通用、可扩展的组通信和事件发布系统，提供应用层多播和任播 [http://research.microsoft.com/~antr/SCRIBE/default.htm]
PAST	广域、安全的 P2P 归档存储系统 [http://research.microsoft.com/~antr/PAST/default.htm]
SQUIRREL	分布式的协同 Web 缓存，使得用户 Web 浏览器之间能共享缓存 [http://research.microsoft.com/~antr/SQUIRREL/default.htm]
SplitStream	基于 Pastry 的高带宽内容流动/分布系统 [http://research.microsoft.com/~antr/SplitStream/default.htm]
POST	提供通信、协同的消息架构，可用来支持安全 E-mail、安全实时消息、分布式协同应用等 [http://www.cs.rice.edu/CS/Systems/POST/default.htm]
Scrivener	强调 P2P 系统资源公平共享的架构 [http://www.cs.rice.edu/CS/Systems/Scrivener/default.htm]
其他 Pastry 项目	(1)PASTA：剑桥大学计算机实验室开发的类似 PAST 的文件系统，但它提供可变、分布式、等级化的名空间 (2)Herald：Microsoft Research 开发的出版/订阅事件发布服务 (3)Pastiche：密西根大学开发的 P2P 备份系统 (4)DPSR：普度大学开发的 DPSR 项目，其目的是开发有拓扑意识的结构化 P2P 架构(如 Pastry)与移动 Ad Hoc 网多跳路由协议之间的协同等

4.4.2 Pastry 路由

每个 Pastry 结点被分配一个 128 位的 nodeID，用来标识其覆盖网位置，nodeID 通过安全散列函数来生成，可以认为是均匀、唯一、独立于结点属性的。每个数据对象被分配一个类似的 objectID，对象索引由 nodeID 与 objectID 最近的结点来负责。

Pastry 路由非常像 Tapestry，只是它采用前缀匹配，而 Tapestry 采用后缀匹配，两种方法并无本质的差别。如图 4.4.2 所示，每个 Pastry 结点维护一个路由表(Routing table)、一个叶集(Leaf set)和一个邻居集(Neighborhood set)，其中，路由表 R 与 Tapestry 的路由表非常类似，也是分层的。图 4.4.2 的中间是一个 4 进制、8 层的路由表，从上到下为第 0 行、第 1 行、……、第 7 行，分别代表与当前结点 nodeID(10233102)匹配 0 位、1 位、……、7 位前缀的结点，每行有 4 项(因为采用 4 进制)，从左到右依次以 0、1、2、3 开头(这里“开头”是指前缀中第一个不匹配的位)，与当前结点 nodeID 在该位恰好相等的项标为阴影，通常是空指针。图 4.4.2中路由表每项结点 ID 都以 X-Y-Z 的形式表示以便于观看(第 0 行没有 X)，其中 X 标识匹配的前缀，Y 表示第一个不匹配位，Z 则是结点 ID 的后几位。另外还可以发现，路由表越往下空缺项越多，因为越往下要求与当前结点 nodeID 前缀匹配越多，也就越不容易找到满足条件的结点。空缺项并不影响 Pastry 正确路由，只是在路由速度上可能会稍慢一些。

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

图 4.4.2 Pastry 结点状态：上面是叶集，中间是路由表，下面是邻居集
(图片来自[Rowstron & Druschel, 2001a])

图4.4.2中Pastry结点状态中还有两个重要的部分：叶集L和邻居集M。叶集L中包含$|L|$个与当前结点ID最邻近的“叶结点”，这些叶结点又分为两部分：$|L|/2$个比当前结点ID小的叶结点，和$|L|/2$个比当前结点ID大的叶结点。叶集的作用在于保证Pastry路由的正确性，它很像Chord中的后继列表。

邻居集M中包含$|M|$个在网络物理层与当前结点邻近的结点，其作用在增强Pastry工作的局部性，路由过程中通常不使用M。综上所述，Pastry结点状态$=L+R+M$，表中结点项数$=|L|+B*\log_B N+|M|$，B为进制(图4.4.2中$B=4$)，N为网络结点总数，它比Tapestry路由表多出叶集和邻居集的$|L|+|M|$项。

基于上述路由表，Pastry采用前缀逐位匹配的路由算法。通常每一步至少比前一步多匹配一位前缀，直到前缀无法再匹配更多位，此时的下一跳结点为ID与目的地最邻近的结点，更具体说，是当前结点的叶集L中与目的ID最接近的结点。具体的Pastry核心路由算法见图4.4.3，其中符号含义如下(A表示当前结点nodeID)：R_l^i：路由表R中第l行第i项；L_i：叶集L中离当前结点nodeID第i近的结点；D_l：D的第l位，D为目的结点ID；$shl(A,B)$：A、B匹配的前缀长度。

```
(1) if(L_{-⌊|L|/2⌋} ≤ D ≤ L_{⌊|L|/2⌋}){
(2)     //D位于叶集L的范围内
(3)     forward to L_i, s. th. |D-L_i| is minimal;
(4) }else{
(5)     //利用路由表
(6)     Let l=shl(D,A);
(7)     if(R_l^{D_l} ≠ null){
(8)         forward to R_l^{D_l};
(9)     }
(10)    else{
(11)        //极少情况下
(12)        forward to T∈L∪R∪M, s. th.
(13)            shl(T,D) ≥ l,
(14)            |T-D| < |A-D|
(15)    }
(16) }
```

图4.4.3 Pastry核心路由算法

(引自[Rowstron & Druschel,2001a])

图4.4.3的算法中，第(1)～(3)行首先检查目的结点D是否在当前结点的叶集L的范围内。如果在，那么消息被直接送到与D最近的结点(包括相等)，它就是D实际对应的结点；否则使用路由表，消息通常被路由到前缀至少多匹配一位

的结点(第(5)～(9)行)。但是某些情况下，也会出现路由表该项空缺或者不可达，此时没有办法匹配更多位，所以消息被路由到与目的结点 D 更接近的表项(第(10)～(15)行)，该项在 L、R、M 中选取。

从上面的算法可以看出：Pastry 路由要么多匹配至少一位 ID，要么能找到比当前结点离目的结点 D 更近的 ID。所以，通常 Pastry 定位跳数为 $O(\log_B N)$，这是结构化 P2P 网络典型的高路由效率，与 Tapestry 相似；同时由于叶集 L 的存在，Pastry 路由通常比 Tapestry 更快、更容错。

为了提高安全性，防止恶意结点的破坏，Pastry 采用“随机路由”(randomized routing)来减少路由的确定性。如对上述的 Pastry 核心路由算法，当多个结点都符合作为下一跳的条件时，不一定非要选择最优的，而是随机选择一个，这样，即使恶意结点了解路由表、叶集中的项，它要完全阻断或者监听路由也是不可能的。当然，采用随机的路由选择后，路由性能一般会有所下降——“安全总是以牺牲一部分性能为代价的”。

4.4.3 Pastry 自组织和自适应

与前面讲到的结构化 P2P 网络一样，Pastry 结点加入网络也需要做三项工作：①初始化自己的路由表、叶集和邻居集；②通知其他结点自己的到来；③从现存结点那里获得需要负责的数据。

JOIN STEP1：*初始化路由表、叶集和邻居集*

假设新结点 X 要加入 Pastry 网络，它首先通过某种方式联系到一个现存结点 A，这里所谓“某种方式”可以是前面讲过的“扩展环”IP 多播，也可以是 Web 网站提供众所周知的结点，不管哪种方式，通常 A 与 X 在物理上会比较近。

联系到 A 之后，X 通过 A 发送一条以 X 为目的地的消息。虽然 X 结点过去并不存在，但这不影响消息路由，消息会按照上面所述的 Pastry 前缀匹配路由算法逐步走下去，最终到达网络中 nodeID 离 X 最近的结点 Z。X 路由表的初始化与 Tapestry 非常类似：加入消息所走过路径上的每个结点将它们的路由表信息发给 X，X 检查这些信息，必要的时候也可能通过这些信息去联系更多的结点，获得更多的路由表信息。总体上，X 是从第 i 跳结点 B_i 那里获得自己路由表的第 i 行，其原因和 Tapestry 是类似的：因为 B_i 与 X 通常匹配 i 位前缀，所以 B_i 的路由表第 i 行是 X 的路由表第 i 行的合理选择。

X 的叶集和邻居集的构造是比较简单的，因为 Z 与 X 在 nodeID 上最近，所以 X 只要从 Z 那里获得叶集作为自己的，然后做必要的修正就可以。前面已说过，A 通常是在物理网上离 X 很近的结点，所以，X 只要从 A 那里获得邻居集作为自己的邻居集就行。

JOIN STEP2：通知其他结点自己的到来

由于叶集和邻居集的存在，Pastry更新其他结点状态要比Tapestry相对简单：X只需要把自己的结点状态信息发给它自己的路由表、叶集、邻居集中的结点即可，剩下的工作由收到更新消息的结点自己去做——去选择用X替换原来的表项。

JOIN STEP3：新结点获取需要负责的数据

Pastry最经典的论文中并没有专门讲述这一步的做法，但是应该和Tapestry差不多——从ID最接近的结点那里获取数据。由于叶集的存在，这一步工作要比Tapestry简单，因为Pastry只要让X从其叶集结点那里获取需要负责的数据即可。

综上所述，Pastry的结点加入算法与Tapestry很像，能达到相同的效果（容错、局部），但更简单、更快，需要的消息数为$O(\log_B N)$，是所有结构化P2P网络中最优的。

Pastry结点的离开，可能是主动的，也可能是突然的。前一种方式易于处理，因为结点离开前只要告诉它所知的结点自己要离开即可。下面我们所讲的是后一种情况，所谓“结点失效”的处理。

FAIL STEP 1：修正叶集

如果结点发现其叶集中某个结点失效，那么它会向其叶集中未失效的最“远”结点获取叶集以修正自己的叶集。所谓最“远”结点，分两种情况：如果失效叶集项的ID＜当前结点ID，那么最“远”结点就是$L_{-|L|/2}$（其含义上面说过）；同理，如果失效叶集项的ID＞当前结点ID，那么最“远”结点就是$L_{-|L|/2}$。假设获取的叶集为L'，那么L'与原叶集L会有很多项重叠，但总是能找到最合适的结点来修正失效的叶集项。叶集对Pastry正确工作有基础性的作用，因此Pastry对叶集的修正是严格、频繁和高要求的。

FAIL STEP 2：修正路由表

路由表修正的要求比叶集要低很多，因为首先路由表错误并不影响Pastry正确工作，只是路由效率会下降；其次路由表项通常比叶集多很多，失效也是经常有的事，如果更新过于频繁反而会带来过重的负担。

如果结点发现路由表中第l行第d项失效，它会发送消息给第l行未失效的第i项结点（$i \neq d$），获得它的路由表第l行第d项作为替代。假如在某些特殊情况下，路由表第l行所有项均失效，那么就联系路由表第$l+1$行结点来获取替代项（这一行也可能所有项均失效），这个过程会迭代地做下去，通常总是会成功的。

FAIL STEP 3：修正邻居集

邻居集在Pastry路由中通常不使用，因此对它的修正比对路由表更为松散：每隔一个周期，结点检查其邻居集看是否有失效的情况，如果有就发消息给未失效

的邻居集结点，获取它们的邻居集，从中找到合适的替代。

4.4.4 Pastry 的局部性

Pastry 的局部性分两步来实现：第一步体现在路由表的构造上，但路由表中的局部性并不准确，只是有一些局部性的意义；在路由表初始化以后，第二步，新结点通过邻居集 M 来修正路由表项，以邻居集 M 为参考标准，用物理网上离自己确实很近的结点来替换原有项。第二步对 Pastry 的局部性有本质的意义，因此也可以说 Pastry 的局部性是源于邻居集 M 的。

上面讲过 Pastry 路由表的构造：新结点 X 从定位路径上第 i 跳结点 B_i 那里获得自己路由表的第 i 行，直到 nodeID 与 X 最近的结点 Z 为止。因为 A 与 X 在物理网上很近，用数学归纳法，我们假设 Pastry 在 X 加入之前，每次为新结点构造路由表时都考虑了局部性（包括邻居集 M 的使用），那么在三角关系近似满足的情况下（三角关系可以说成：如果 A、B 很近，B、C 很近，那么 A、C 也很近）可以推出：当 X 从 A 中复制第 0 行作为自己的路由表第 0 行、从 A 中复制邻居集 M 作为自己的邻居集、并用 M 来修正过第 0 行后，X 的路由表第 0 行结点跟 X 在物理上都是邻近的。

随着跳数的增加，局部性将变差，三角关系也越来越不成立，因此 X 离其路由表中第 1 行结点会比较远，离第 2 行结点更远，依次类推。这样的局部性关系反映在图 4.4.4 中。

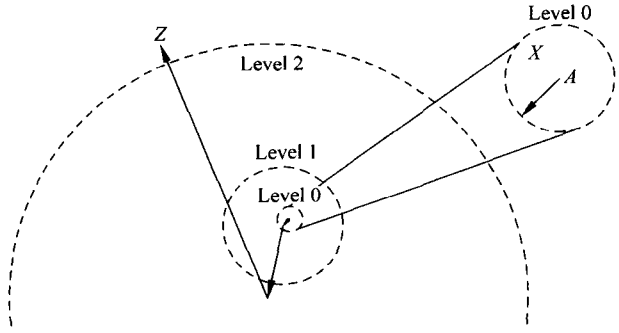


图 4.4.4 Pastry 路由表局部性关系。X 为加入结点，A 为其自举结点，Level 表示路由表行数或路由跳数，圆弧和半径表示实际距离
（图片来自[Rowstron & Druschel, 2001a]）

Pastry 采用传统的 ID 邻近复制，同一数据对象被复制到与其 ID 相近的 k 个结点上。这实际上也提供了一定的数据存取局部性：因为 Pastry 中结点 ID 与结点本身属性无关，因此 ID 邻近的结点通常分散在整个网络中，而 Pastry 定位对象时主要依靠 ID，如果要查询某个对象，定位机制通常会在大致相当的时间里找到

这些结点(其实只需要一个叶集 L 就可以都找到了)，然后，由查询者自身来决定到底要哪个副本。由于 k 个副本通常均匀分散，所以总会有一两个离查询者很近。

4.4.5　Pastry 实验分析

图 4.4.5 描述了 Pastry 平均路由跳数与网络结点数的关系，并与 $\log N$ 曲线做了比较。实验中取 $B=16$，$|L|=16$，$|M|=32$。可以看出 Pastry 路由跳数曲线比 $\log N$ 曲线要略低，这样的路由性能是优秀的。

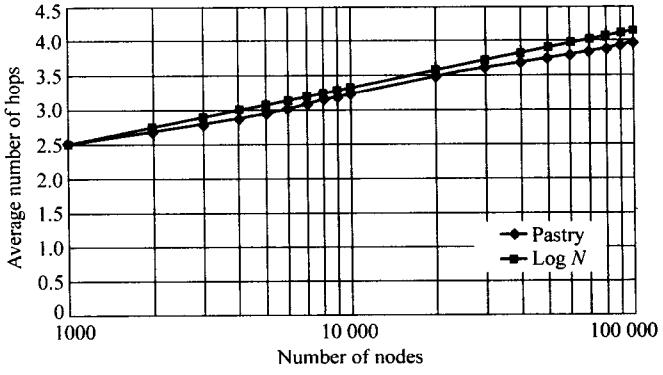


图 4.4.5　Pastry 平均路由跳数与网络结点数的关系
(图片来自[Rowstron & Druschel,2001a])

图 4.4.6 描述了 Pastry 路由路径长度(以跳数来衡量)的相对距离(relative distance)与网络结点数的关系。下面的黑线是一种理想的最优路由(采用完全路由表)，它走的总是最短路径，因此相对距离恒为 1。Pastry 虽然采用贪心的局部

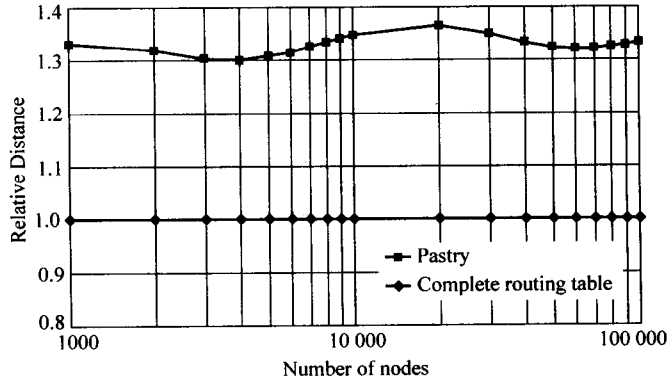


图 4.4.6　Pastry 路由相对距离与网络结点数的关系
(图片来自[Rowstron & Druschel,2001a])

路由方法——前缀匹配路由，但路径长度的相对距离仅在 1.3～1.4 之间，略高于最优情况，这证明 Pastry 路由算法是非常先进的。

图 4.4.7 描述了 Pastry 平均路由跳数与结点失效之间的关系。实验中取 $B=16$，$|L|=16$，$|M|=32$，网络中共有 5000 个结点，做了 20 万次查询。第一条柱状图表示没有结点失效，平均路由跳数为 2.73；第二条柱状图表示有 500 个结点失效(10%)、但不修复路由表时，平均路由跳数上升为 2.96；第三条柱状图表示有 500 个结点失效(10%)、但修复路由表时，平均路由跳数为 2.74，仅比无结点失效时的 2.73 高 0.01，这说明 Pastry 的路由表修复方案实用并且优秀。这张图也体现了 Pastry 的高容错性。

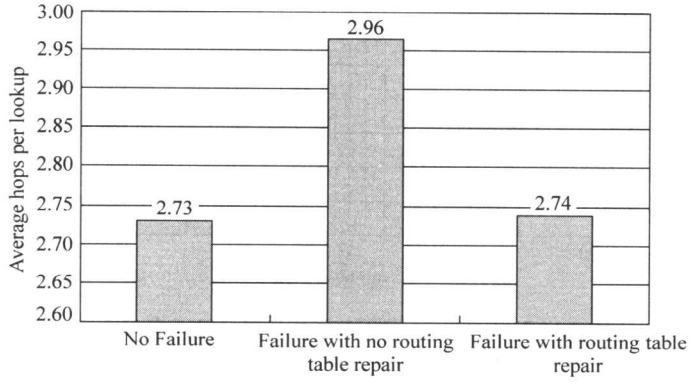


图 4.4.7 在无结点失效、10%失效且不修复路由表、10%失效但修复路由表三种情况下 Pastry 平均路由跳数的变化

(图片来自[Rowstron & Druschel,2001a])

4.4.6 Pastry 总结

(1) Pastry 是一个容错、高效、可扩展的混合式结构 P2P 网络，结合了环形结构与超立方体结构(实际是 Plaxton Mesh)的优点。它在因特网上构造了一个分布式、自组织、容错的覆盖网，提供高效的路由、确定性的对象定位和独立于具体应用的负载平衡。此外，它还提供了复制、缓存、错误复原等增强机制以提高系统性能。

(2) Pastry 的定位、路由与 Tapestry 非常像：每个结点、每个数据对象分配一个唯一、随机的 ID，采用来自 Plaxton 的层次化路由表，不过采用前缀匹配路由。但 Pastry 结点除了路由表，还有叶集 L 和邻居集 M。L 维护 ID 邻近的结点，确保路由正确性，M 则维护与当前结点在物理上邻近的结点。

(3) Pastry 的结点加入、失效算法与 Tapestry 类似，但由于叶集 L 和邻居集

M 的存在，Pastry 能更简单、更快地达到同样的自适应效果。

(4) Pastry 在构建网络时即考虑了局部性因素，这分两步来实现：第一步是路由表的构造，第二步以邻居集 M 为参考来修正路由表，用物理网上离自己确实很近的结点来替换原有表项。第二步对 Pastry 的局部性有本质的意义，是局部性的真正来源。

(5) Pastry 有着广泛的应用：最著名的是 PAST 归档存储系统，此外还有 SCRIBE 组通信和事件发布系统(提供应用层多播和任播)、SQUIRREL 分布式协同 Web 缓存等。

4.4.7 PAST 简介

PAST 是一个广域的 P2P 归档存储系统，它以 Pastry 作为底层架构，目的是提供 Internet 上安全、高可用、持久性的数据存取服务。每个用户向系统贡献自己的一部分带宽和存储空间，同时可以获取系统中的文件，或者将自己的文件归档存储到系统中。下文对 PAST 的讲述基于这两篇论文：[Rowstron & Druschel, 2001b]和[Druschel & Rowstron, 2001]。

因为以 Pastry 为基础，所以 PAST 中每个结点、每个对象都被分配一个唯一、随机的 ID，文件的副本则存放在与其 ID 最接近的多个结点中，因此总体上看来，整个系统的负载是均匀分布的。然而，负载均匀并不总是好的，因为网络中结点的能力差异很大，单就带宽而言，慢到 56Kbps 的电话线，快到超过 10Gbps 的光纤；另一方面，文件容量也是大小不一，小到不足 1KB，大到很多 GB。所以，很多时候“负载不均衡”反而是需要的，比如当某个文件被频繁访问而成为“热点”时，PAST 将它缓存到网络其他结点的空闲空间中，这些“负载不均衡”的方法下文将讲到。

4.4.8 PAST 操作

PAST 向其用户提供下面几个基本的操作：

(1) 插入：fileID=Insert(name,owner-credentials,k,file)

插入操作允许用户插入一个名为 name 的文件 file 到系统中，并要求系统为其保存 k 个副本，用户证书(owner-credentials)通常是用户的公钥。插入操作返回数值 fileID，以在系统中标识该文件。fileID 常常是函数四个参数的安全散列值，或者还可能在其中加入一个“盐值”(salt)以增加随机性。

每当一个用户向系统插入一个文件，所需的存储空间(文件大小×k)就被记入该用户的存储限额(storage quota)，存储限额被用完的用户不能再插入文件，这有效地限制了恶意用户对系统的滥用。同时，该用户给想插入的文件生成一个“文件证书”(file certificate)，并用其私钥给证书签名，以证明颁发者的真实性。证书中包含的信息有：fileID、文件内容散列值、副本数目 k、盐值、创建时间和其他可选元

数据信息等。文件证书和文件一起通过 Pastry 网络路由，当到达合适的结点时（nodeID 与 fileID 最近的 k 个结点中的一个），那个结点首先验证文件证书是否是文件拥有者所发，然后重新计算文件内容散列值与证书中的做比较，来验证数据完整性。如果所有信息都通过验证，此结点将负责存储这个副本，并将插入请求路由到其他负责存储文件的结点（这些结点大多会在其叶集 L 中）。

一旦 k 个结点都各自保存了一个副本，插入操作成功，但还要向插入者回发一条确认消息。每个负责存储副本的结点都在这条消息中附加自己的“存储收据”（storage receipt），插入者验证这些存储收据，以确认 k 个副本确实被插入到系统的不同结点中。

（2）查询：file=lookup(fileID)

查询操作能够可靠、安全地获得标识为 fileID 的文件（通常还附加该文件的证书），只要 PAST 系统还存在该文件的一个可达副本。由于每个文件通常都有多个副本存在，这些副本均匀地分散在网络中，同时文件还可能被缓存在许多地方，因此查询操作常常能获得很近的副本，这对系统高效工作是很有帮助的。

目前 PAST 不支持关键码查询，系统中的文件也没有目录结构，查询只能基于 fileID，这是结构化 P2P 网络共同的缺点。

（3）回收：Reclaim(fileID,owner-credentials)

回收操作删除以 fileID 标识的文件，用户证书（owner-credentials）一般是用户的公钥。通常回收者还提供“回收证书”用来认证回收的合法性，因为只有文件的插入者有权删除它。收到回收消息后，负责存储文件的结点回发“回收收据”（reclaim receipt）给回收者，回收者收到后可以恢复插入该文件耗去的“存储限额”。需要注意的是：回收操作只是在概念上“删除”文件，并不真正删除它，所以文件即使删除了，仍然可能定位到；只有当存储文件的结点主动删除它时，文件才真正从系统中消失。

4.4.9 PAST 安全机制

每个 PAST 用户都有一个“智能卡”（smartcard），每个“智能卡”拥有一对公钥/私钥，其中的公钥还带有“智能卡发行者”（smartcard issuer）的数字签名以验证公钥真实性，“智能卡发行者”通常是权威的可信机构。“智能卡”是 PAST 安全机制的核心部件，它的作用包括：产生和验证各种各样的证书，以及维护结点的“存储限额”。

虽然 PAST 假设其用户都有成为恶意结点的可能，但认为：①系统中大多数用户仍然是可信的；②攻击者不能控制“智能卡”这一核心部件。上面两个假设是 PAST 安全工作的前提，通常它们都能得到满足。

“智能卡”保证了 nodeID 和 fileID 的完整性，从而防止了攻击者控制 nodeID

邻近的多个结点以完全控制某个文件所有副本的可能。上面讲过的“存储收据”防止了攻击者欺骗文件插入者已存储 k 个副本、但实际上副本数远远不足 k 的可能。而“文件证书”则使得存储文件的结点和获取文件的结点都能验证文件插入者的真实性和文件内容的完整性。最后，“文件证书”和“回收证书”两者配合起来，实现了PAST的“存储限额”机制。在必要的情况下，PAST用户可以先将文件加密，然后再插入到系统中。

PAST安全机制的另一方面在于其底层架构Pastry的安全性。如上文所说的“随机路由”方法有效降低了路由确定性，使得用户攻击Pastry路由过程十分困难；其次，对于每个路由表项，或者叶集、邻居集中的结点，Pastry可以要求相应结点签名后做验证、验证通过才加入表中的方法，从而防止了恶意结点伪造表项的可能。

4.4.10 PAST存储管理

PAST存储管理的目的是：①在系统存储利用率逐步达到100%的过程中，不断平衡网络结点剩余的空闲存储空间；②维护PAST的复制不变属性——每个数据对象的 k 个副本被存放在nodeID相近但不同的 k 个结点上。为了达到这两个目的，首先，PAST采用了两种称为“转移”(diversion)的方法：一种称为“副本转移”(replica diversion)，一种称为“文件转移”(file diversion)，它们将在4.4.11节讲述。其次，基于结点异构性，PAST通过比较的方法控制“每结点存储容量”(per-node storage)的分布。每当新结点 N 加入网络，PAST将 N 所公布的存储能力(N 只允许公布其真实磁盘空间的一部分)与其叶集 L 中结点的平均存储能力做比较。如果 N 的存储能力很高，它就被“分割”，以多个nodeID的形式加入网络(这类似于Chord中的“虚拟服务器”)；反过来如果 N 的存储能力太低，它可能会被系统拒绝。

PAST存储管理的一个不足之处是它对文件不分片，因此大文件常常需要通过“副本转移”或者“文件转移”的方法来存放，而过大的文件则经常没有办法放下，只好被系统拒绝。当然，不分片也有不分片的好处，一是实现简单，二是避免了分片带来的开销和维护的复杂性。

4.4.11 PAST副本转移和文件转移

副本转移：通常，PAST让nodeID与fileID最近的 k 个结点保存文件的 k 个副本，但如果这 k 个结点并不全适合，比如其中某些结点带宽、空间严重不足但文件又很大，此时，“副本转移”允许这 k 个结点叶集中的结点来代替地保存副本。比如 A 是 k 个结点中的一个，但 A 不能容纳文件 f，于是 A 选择一个可以容纳 f 的叶集结点 B 来代替它保存 f，而 A 只保存一个到 B 的指针，B 代替 A 保存好 f 之后，A

像原先一样回发“存储收据”，这个过程就称“A 将 f 的一个副本转移到了 B 中”。

副本转移虽然有效地利用了系统资源，但也带来了很多相关的问题。在 A 将 f 的副本转移到 B 之后，必须确保两点：①B 的失效将自动导致一个新副本的创建；②A 的失效不会导致 B 上的副本不可达。条件①是结合 Pastry 来完成的，因为 Pastry 保证了 A 能及时检测到 B 的失效从而再做一次新的副本转移。条件②通过在另外一个结点 C 的文件表里保存一个 B 上副本指针的方法来保证，C 是 nodeID 离 fileID 第 $k+1$ 近的结点。如果 A 失效则 C 自动成为离 fileID 最近的 k 个结点中的一个，从而 C 所保存的 B 上副本的指针刚好能替代原来 A 上的工作，系统不需要再做调整。

文件转移：仅有副本转移并不能解决所有的问题，比如 PAST 让 nodeID 与 fileID 最近的 k 个结点保存文件的 k 个副本，可是这些结点不能完成该任务，于是它们想通过“副本转移”来完成，可是很不巧，它们的叶集结点还是不能容纳该文件，“副本转移”无法成功，此时只好回发给文件插入者一个“消极确认”(negative acknowledgement)。收到“消极确认”后，插入者并不放弃，他使用新的“盐值”(salt)为该文件产生新的 fileID，并重新尝试将它插入，上述过程被称为“文件转移”。文件转移如果一次不成功会再重复做第二次、第三次，直到第四次尝试失败，插入者才放弃插入操作，这通常说明文件实在太大，或者运气实在太差。

4.4.12 PAST 缓存管理

采用缓存的目的是尽量减小文件查询时延、增加系统查询吞吐量和平衡系统的查询负担。对于“热点”数据而言，仅维护 k 个副本不足以分担对数据的频繁查询及获取，因此，PAST 采用路径方法来缓存它的副本，这里“路径”可以是文件插入的路径，也可以是文件查询的路径。

与复制不同，缓存的副本是临时的，结点需要空间时可以自主替换缓存。PAST 的缓存替换方法基于“GreedyDual-Size”GD-S 策略，该策略原来是被开发以缓存 Web 代理的。具体地说，GD-S 为每个缓存的文件 d 维护一个“权重”(weight)$H(d)$，每当缓存命中(文件 d 被插入或查询)时，$H(d)$被设为 $c(d)/s(d)$，其中 $c(d)$代表和文件 d 相关联的某种开销，$s(d)$代表文件大小即 size(d)。当结点需要空间时，它替换所有缓存中 $H(d)$最小的文件，假设为 v；v 被替换后，当前的每个缓存文件的 $H(d)$被减去 $H(v)$。GD-S 策略的优点在于它可调节，比如将文件开销设为固定的 1，GD-S 策略就最大化了文件命中率的影响。

4.4.13 PAST 总结

(1) PAST 是一个广域的 P2P 归档存储系统，它以 Pastry 作为底层架构，目的是提供 Internet 上安全、高可用、持久性的数据存取服务。

(2) PAST 向其用户提供几个基本的操作：插入、查询、回收。这些操作都需要相应的“证书”以检验其合法性，并且插入、回收时还需要“收据”来做确认。

(3) 每个 PAST 用户都有一个“智能卡”，它由权威的可信机构发行，是 PAST 安全机制的核心部件。其作用包括：产生和验证各种各样的证书以及维护结点的“存储限额”。

(4) PAST 存储管理有两个目的：①不断平衡网络结点剩余的空闲存储空间；②保持每个数据对象有 k 个副本被存放在 nodeID 相近但不同的 k 个结点上。为了达到这两个目的，PAST 采用了“副本转移”、“文件转移”、“路径缓存”等多种方法。

4.5 其他著名结构化 P2P 网络——Kademlia、SkipNet 等

4.5.1 其他著名结构化 P2P 网络简介

前面四节讲述了四个最著名的结构化 P2P 网络及其应用系统：Chord 和 CFS、CAN、Tapestry 和 OceanStore、Pastry 和 PAST，它们是结构化 P2P 的先驱和最杰出的代表。在它们之后，更多的结构化 P2P 网络模型被提出，其中不乏理论上的优秀者和实践上的成功者。

在理论上，出现了一种有特殊性质的新型结构化 P2P 模型——常数度 P2P 网络，它们的路由、定位、自组织方式与过去的模型区别并不大，但每个结点的“度”(即连接数)是固定的，不随网络规模而变。常数度 P2P 网络在保持每个结点维护固定路由表项数的同时，仍然能达到 $O(\log N)$ 跳的指数定位效率，保持了系统的高可扩展性；同时，由于路由表项数固定，网络自适应的开销也相应减少了。最著名的常数度 P2P 模型有 Viceroy(2002)、Koorde(2003)、Cycloid(2004)等，4.5.2 节将详细介绍它们。

在实践上，研究者从新的角度、新的度量上去看结构化 P2P 网络，提出了更加容错、实用的结构化 P2P 系统，比如发表于 IPTPS'02 的 Kademlia 系统，其路由方式类似 Chord，但采用基于“异或”(XOR)的距离量度，并将网络结构配置信息融合到每条消息中，从而构建了一个高容错、自适应的 P2P 信息系统。继 Kademlia 之后，2003 年 Microsoft Research 和华盛顿大学的研究者提出了 SkipNet 模型：一个基于“跳表”(SkipList)数据结构、很多方面类似 Chord 的结构化 P2P 网络，它最大的特点在于提供结点路由、对象语义两方面的局部性，这是以前的结构化 P2P 网络都没有做到的。

4.5.2 Kademlia——基于异或度量的 P2P 信息系统

1. Kademlia 概要及其应用

Kademlia 最大的特点在于使用“异或”(XOR)来度量网络结点之间的距离，这样的度量方式不影响 P2P 网络的工作效率和可扩展性，却提供了更好的容错性和灵活性。因为“异或”是对称的，所以 Kademlia 结点能从路由消息中获得有用的网络配置信息，这种新颖的“捎带更新”方法使得 Kademlia 以很小的开销获得了很高的自适应性。反过来，对最著名的 P2P 模型 Chord 而言，由于其距离度量不是对称的，路由消息不能反映多少有用的网络结构信息。此外，Chord 路由表必须是严格的，尤其是后继关系必须为不变属性；而 Kademlia 则没有这样严格的限制，结点可以发送消息给其路由表任意一“段”(interval)中的每一个结点，让它们基于时延选择路由下一跳，甚至让它们发送并行的异步消息。

Kademlia 被许多文件共享系统(或软件)使用，表 4.5.1 简单介绍了这些应用(“开始于 xxx 版”指该应用整合 Kademlia 的时间)。

表 4.5.1 Kademlia 的应用

Overnet 网络	Overnet	现已被整合到 eDonkey2000 中
	eDonkeyHybrid	混合式 eDonkey 软件
	mlDonkey	运行于多平台、多网络的 eDonkey 扩展版软件，除了支持多个操作系统平台，还支持许多协议，如 BitTorrent、DirectConnect、eDonkey、FastTrack、Gnutella2、FTP/HTTP、Kad 网络、Overnet、OpenNap、SoulSeek 等 [http://www.nongnu.org/mldonkey/]
Kad 网络	eMule	在第 2 章已详细讲过，开始于 0.40 版
	mlDonkey	同上，开始于 2.5-28 版
	aMule	aMule 即“all-platform eMule”，它是 eMule 的扩展版，能运行于大多数操作系统平台，开始于 2.10 版 [http://www.amule.org/]
RevConnect		基于 DirectConnect 协议的 P2P 文件共享软件，以 Kademlia 作为分布式散列表，具有多源下载、部分文件共享、安全用户认证、自动资源回收、增强的自动搜索等功能(开始于 0.403 版) [http://www.revconnect.com/]
KadC		用以在 Overnet 网络中发布、获取信息的 C 语言库 [http://kadc.sourceforge.net/]
Azureus		开始于 2.3.0.0 版，使用 Kademlia 网络作为 BitTorrent Trackers 失效时的替代定位方法

续表

BitTorrent	在第2章已详细讲过，其Beta 4.1.0版已拥有一个Kademlia网络，为那些无Tracker的torrents服务
BitSpirit	基于BitTorrent协议的一个客户端，开始于3.0版
eXeem	基于BitTorrent网络的一个P2P文件共享软件，但其目的是取代BitTorrent中原有的Trackers（开始于Beta 0.25版）[http://www.exeems.com/]

注：以上信息来自于http://en.wikipedia.org/wiki/Kademlia。

2. Kademlia的异或度量、结点状态和自组织

同经典的结构化P2P网络一样，Kademlia给每个结点分配一个唯一、随机的nodeID，每个对象分配一个类似的objectID（又称key），使用160位的SHA-1函数来生成ID。对象索引由nodeID最“接近”objectID的结点来负责（注意这里的“接近”是以异或距离来度量的）。

假设Kademlia中有两个结点，其nodeID分别为x,y，那么它们之间的距离$d(x,y)=x\oplus y$，比如$x=1011,y=0010$，则$d(x,y)=1011\oplus 0010=1001$。虽然异或距离是非欧的（非欧几里得几何度量），但它却是合理的。很明显$d(x,x)=0$；$d(x,y)>0$，if $x\neq y$，并且对于任何x,y有：$d(x,y)=d(y,x)$，即所谓的“对称性”。在过去的P2P模型中，CAN、Tapestry和Pastry都具有对称性，但Chord不具有。除此之外，异或距离还具有“三角属性”：$d(x,y)+d(y,z)\geqslant d(x,z)$，三角属性公式来自下面两个引理：①$d(x,z)=d(x,y)\oplus d(y,z)$；②对任意的$a\geqslant 0$，$b\geqslant 0$，$a+b\geqslant a\oplus b$。

与Chord的顺时针环形度量一样，Kademlia的异或度量也是“单向”的，所谓“单向”，指的是对于任意点x和距离$d>0$，系统中有且仅有唯一点y满足$d(x,y)=d$。单向性保证了所有对相同数据对象的定位最终将会聚于相同的路径，且越往后走会聚的可能性越高，因此，采用“路径缓存”对于提高Kademlia的定位效率、缓解热点问题很有帮助。

每个Kademlia结点维护一个称为“k-buckets”的路由表。以采用160位ID为例，对每个$i(0\leqslant i<160)$，结点都保存一个链表（一个链表称为一个k-bucket），其中记录了到自己的异或距离在2^i与2^{i+1}之间的一些结点，每个链表项是形如〈IP地址，UDP端口，nodeID〉的三元组。每个链表以最近访问时间来排序，最不近看到的结点排在表头，最近看到的排在表尾。对于很小的i来讲，其链表通常为空，因为与本地结点ID异或距离如此近的结点一般很难找到；但对于大的i来讲，i越大其链表中需要记录的结点数越多，因为每一“段”（interval）的长度是随i指数递增的。为了限制链表中的项数，Kademlia规定链表长度以k为上限（这也

是为什么称 Kademlia 路由表为“k-buckets”的原因）。

每当 Kademlia 结点收到来自其他结点的消息，它就用消息发送者的 nodeID 来更新相应的 k-bucket，这称为“捎带确认”（piggybacking，也称为捎带更新）：如果该 k-bucket 中已有此 nodeID，原项将被移到表尾；如果链表长度不到 k，它将被附加到表尾；如果链表长度已为 k，此时结点首先去联系表头结点，如果能联系到，则将表头结点移到表尾其他项不变，如果联系不到，则删除表头结点，再将发送者的 nodeID 插入到表尾。Kademlia 的链表项替换策略除了符合 LRU（least recently used）规则，还能有效地防止 DoS 攻击：攻击者没有办法用特定的 nodeID 填满其他结点的路由表，因为 Kademlia 只有检测到旧结点失效才会用新结点来取代它。Kademlia 的“捎带更新”自适应方法新颖、独特、实用，并且带来的网络开销很小。

3. Kademlia 协议

Kademlia 协议由四个 RPC（remote procedure call，远程过程调用）组成：PING、STORE、FIND_NODE 和 FIND_VALUE。对任何一个 RPC 消息，接收者都必须回复一个 160 位的 RPC ID，以抵制地址伪造。

PING：用来探测一个结点，看它是否在线。

STORE：指示一个结点存储一个〈key，value〉对，以便于将来的数据获取。key 为对象散列值，即 objectID，value 为真正的数据对象（或其索引）。

FIND_NODE：以 ID 为参数，FIND_NODE RPC 的接收者返回他所知的离目的 ID 最近的 k 个结点，以三元组〈IP 地址，UDP 端口，nodeID〉的形式。

FIND_VALUE：以 key 为参数，寻找 key 对应的 value。它与 FIND_NODE 非常像，也是返回类似的三元组，但有一个例外：如果 FIND_VALUE RPC 的接收者已收到过相应 key 的 STORE 消息，他只返回对应 key 的 value。出于缓存的目的，成功找到 value 的用户会将〈key，value〉对存储到它所知的离 key 最近但没有返回 value 的结点中。

Kademlia 中最重要的过程，是定位与目的 ID 最近的 k 个结点，这个过程又称“结点查询”，Kademlia 是通过一个递归的过程来完成的。查询发起者 I（Initiator）首先从他自己的 k-buckets 中找到离目的 ID 最近的 α 个结点（α 是一个系统并行参数），然后发送并行、异步的 FIND_NODE RPC 给这 α 个结点。如果 I 没有收到某个结点的回复，它就在自己的 k-buckets 中删除那个结点。

如果一轮 FIND_NODE RPCs 都没有能返回比 I 所知离目的 ID 更近的结点，查询发起者 I 将重发 FIND_NODE RPC 给离目的 ID 最近、但过去未查询过的 k 个结点。当 I 查询它所知离目的 ID 最近的 k 个结点并收到它们的回复时，“结点查询”完成。

如果某个结点要向网络中存储〈key，value〉对，它首先使用“结点查询”操作定位到离 key 最近的 k 个结点，然后给它们发送 STORE RPC，让它们存储 value，这实际上就是“ID 邻近复制”。此外，上述的 k 个结点每过一个小时就重新发布〈key，value〉对，以保证数据可用；同时，还要求最初的〈key，value〉对发布者每过 24 小时重新发布它，否则所有〈key，value〉对都会在 24 小时后过期。

为了保持存储、查询的一致性，当任何一个结点 w 发现，存在另一个结点 u 离保存在 w 上的〈key，value〉对更近时，w 会复制这个〈key，value〉对到 u，但并不删除自己数据库中的〈key，value〉对。

为了保持路由表（k-buckets）的更新，如果某个 k-bucket 的所有结点在一个小时中未被查询，那么就从中任选一个 ID 对它做“结点查询”，以刷新这个 k-bucket。

如果新结点 u 要加入 Kademlia 网络，它首先联系到一个网络现存结点 w（自举结点），将 w 加入自己的 k-buckets，然后通过 w 做一次以 u 为目的地的“结点查询”，从而初始化自己的 k-buckets，然后将自己的到来告诉其他结点，以更新它们的状态。

4.5.3 SkipNet——基于跳表、提供显式局部性的 P2P 模型

1. SkipNet 概要

SkipNet 是一个通过使用“跳表”（SkipList）数据结构提供显式局部性的结构化 P2P 模型，它提供的显式局部性表现在两个方面：①内容局部性，即语义上可控的对象放置，能够显式地指定数据对象的放置位置或者放置范围；②路由局部性，同一个组织中结点间消息路由的路径一定位于该组织内部。虽然以前的 P2P 模型或多或少都提供了一些隐式的局部性方法，但 SkipNet 是第一个提供显式、明确的局部性的 P2P 模型，这是 SkipNet 最大的亮点。

因为 SkipNet 提供显式的局部性，取消或者说弱化了一致性散列函数的影响，因此它不能像 Chord、Pastry 等 P2P 网络那样提供非常好的负载均衡。实际上，“局部性”和“负载均衡”这两个属性本身就是矛盾的。SkipNet 通过折中的办法来平衡这对矛盾，称为“受限负载均衡”：即在某个范围内数据分布是均衡的，但从网络总体上看未必成立。

SkipNet 语义局部性能够缓解一些在其他系统中非常严重的问题，比如“网络分割”问题：Chord、Pastry 这样的 P2P 网络一旦被分割，性能就急剧下降，甚至无法工作陷入瘫痪；SkipNet 却不同，由于它保证了同一个组织中结点间消息路由的路径一定位于该组织内部，所以即使这个组织与外界隔离，它还能自闭地正常工作，当检测到网络分割之后，在恢复与外界连接的同时，组织内部通信不受影响。实际上，路由局部性对于 SkipNet 网络安全也带来很多影响：从正面讲，首先由于

组织内通信是自闭的，外界很难监听到，其次路由局部性使得对该组织的控制、管理很方便；从反面讲，既然合法的管理者能方便地控制该组织，技术高超的“黑客”自然也能控制它。所以说局部性是一把双刃剑，什么人用、怎么用是关键。

SkipNet 采用了两套独立而又相关的 ID：NameID(名标识)和 NumericID(数值标识)。结点名和对象名被直接映射到 NameID，NumericID 则是通过“层次化分环”过程来获得(下文会讲到这个过程)，与结点名、对象内容无关。在折中的方法里，SkipNet 将结点名或者内容标识用“!”分成两部分，前一部分直接作为 NameID，后一部分的散列值作为 NumericID，从而在“局部性”与“随机性”之间获得一个平衡。

2. SkipNet 组织结构

我们首先介绍 SkipNet 的理论来源——跳表(SkipList)。跳表是一个有序的链接表，其中一些指针指向跳表中距离很远的结点。每个指针有它的“层”(level)，层越高指得越远。在一个“完全跳表”中，第 h 层指针指向距离 2^h 的结点(如图 4.5.1 所示)。通常每隔 2^i 个结点，才会有一个 i 层指针(注意最底层为第 0 层)。“完全跳表”支持 $O(\log N)$跳的关键码搜索，N 为跳表结点总数。

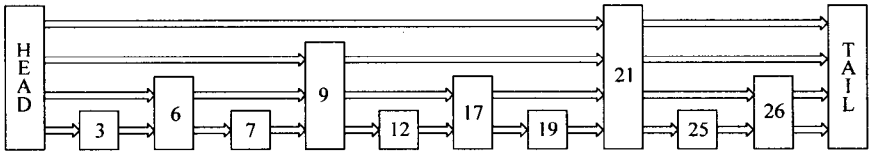


图 4.5.1 完全跳表示例

(图片来自[Harvey et al.,2003a])

对于 SkipNet 而言，使用“完全跳表”不适合，因为网络是动态变化的，而“完全跳表”结构过于严格，插入、删除的开销非常大。鉴于此，SkipNet 使用了“概率跳表”这样一个变形，在“概率跳表”中，每个结点按照概率来选取自己具有哪层指针，而不是按照位置。0 层指针是必选的，选择第 i 层指针的概率为 $1/2^h$。“概率跳表”仍然支持 $O(\log N)$跳的关键码搜索。图 4.5.2 是“概率跳表”的例子。

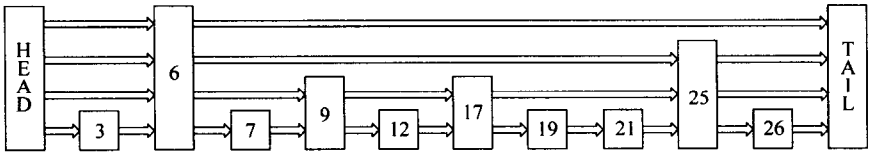


图 4.5.2 概率跳表的例子

(图片来自[Harvey et al.,2003a])

SkipNet 基于“概率跳表”，只不过用网络结点代替了跳表结点，将跳表头尾相接组成一个环，并且每个指针变为双向的。因此结点路由表(R-Table)项数为 $2\log N$，其中第 h 层指针指向距离当前结点 2^h 跳的两个结点，一个沿顺时针方向，一个沿逆时针方向。图 4.5.3 中，根环上结点之间按照结点名或者其数据对象关键码来排列，结点名或对象关键码称为结点的 NameID(名标识)。

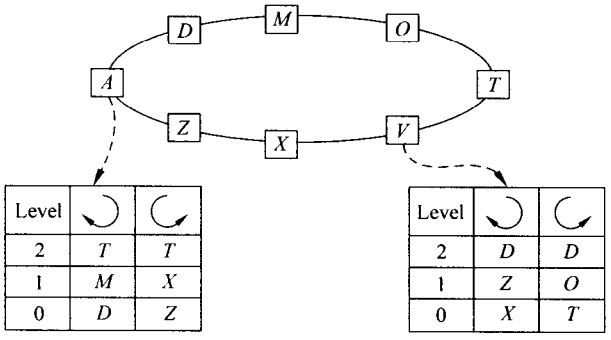


图 4.5.3 SkipNet 根环(0 层主环)及结点 A、V 的路由表
(图片来自[Harvey et al.,2003a])

图 4.5.4 更详细地描述了图 4.5.3 中 SkipNet 的结构。在最底层(0 层)所有结点依靠 0 层指针相接成一个“根环”(Root Ring)，而 1 层指针将根环分成两个环，因为 1 层指针是隔一个结点指的。同样，第 i 层指针将第($i-1$)层环分成两个 i 层环，也就是将根环分成 2^i 个 i 层环。当然，图 4.5.3 和图 4.5.4 所示的 SkipNet 结构过于简单、规整，实际的 SkipNet 网比它要复杂、不规则很多：当第 i 层环被分成 $i+1$ 层环时，i 层环中每个结点以 1/2 的概率选择加入哪个 $i+1$ 层环，这样，虽然环的拆分可能是不等的，但是松散的组织方式有利于插入、删除操作，并且$O(\log N)$的定位跳数仍然能保持。

在上述环拆分过程中，我们可以看到第 i 层环有自己的环标识(RingID)，这个标识有 i 位。根环没有环标识，它被分成两个 1 层环，左边的为 Ring 0，右边的为 Ring 1；Ring 0 又被分成两个 2 层环，左边的为 Ring 00，右边的为 Ring 01；依次类推递归地做下去，最后会分到某层环，它只有一个结点，此时的环标识就是结点的 NumericID(数值标识)，如图 4.5.4 中结点 M 的 NumericID 是 010。因此，结点 NumericID 的前 h 位决定了它在第 h 层的环关系：它由哪个 $h-1$ 层环拆分成的，它又能分成哪两个 $h+1$ 层环。从上述内容可以看出：NumericID 与 NameID 没有关系，它不反映结点或者数据对象的属性，其作用类似于一个散列函数，也是随机、均匀的。另外，就 ID 排布而言，根环上的结点以 NameID 顺序串成一个环，而最高层的单结点环是以 NumericID 排序的。

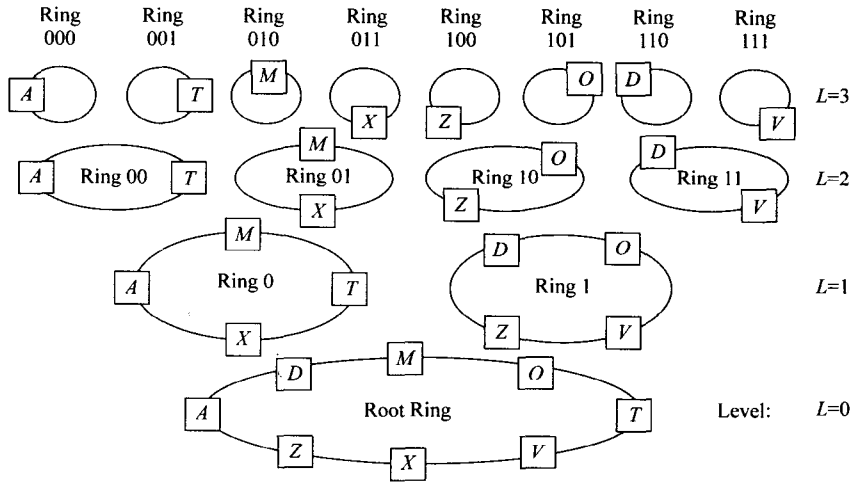


图 4.5.4 一个 8 结点的 SkipNet 规整的环拆分示例
（图片来自[Harvey et al.,2003a]）

除了路由表(R-Table)，每个 SkipNet 结点还维护一个“叶集”L，这与 Pastry 的叶集非常像，其中 $L/2$ 个叶集结点是环上顺时针方向离当前结点最近的，另外 $L/2$ 个叶集结点是逆时针方向离当前结点最近的。叶集对于 SkipNet 的路由效率、容错性都有很大的影响，这在 Pastry 中已经讲过，这里不再累述。

3. SkipNet 路由：NameID 路由＋NumericID 路由

SkipNet 的路由分为两种：第一种是 NameID 路由，总体上与 Chord 很像，是数值邻近路由，或者说是“Name 邻近路由”；第二种是 NumericID 路由，总体上与 Pastry 很像，是前缀匹配路由。

NameID 路由算法与 Chord 类似，下一跳总选择路由表中不超过目的结点 NameID 的最高层结点，直到 NameID 与目的结点 NameID 最近(包括相等)的结点为止。NameID 路由跳数通常为 $O(\log N)$。

NumericID 路由算法跟 Pastry 很像：首先查找根环，直到发现一个结点与目的结点的 NumericID 匹配第一位，此时通过这个结点“爬升”到它的 1 层环，再在 1 层环上查找与目的结点 NumericID 匹配前两位的结点。依次类推，每次上升一层多匹配一位，直到不能匹配更多位、又不能找到 NumericID 更接近的结点为止，此时到达的就是目的结点。SkipNet 的 NumericID 路由跳数通常也为 $O(\log N)$。

4. SkipNet 结点加入和离开算法

SkipNet 新结点加入很像一个“爬树”的过程，这里的“树”指的是 SkipNet 层

环，新结点从“叶”爬到“根”（最高层环），再从“根”爬到“叶”。具体来说，假设新结点为 N，它首先必须找到与其 NumericID 相应的最高层环，这个过程是通过使用 NumericID 路由算法，发送以 N 为目的地的消息来做到的。当 N 爬到最高层环后，它使用 NameID 算法找到在这层环上自己的邻居，然后通过该邻居，在下一层环上寻找邻居，依次类推，直到根环。做完上述过程后，N 通知它所在每层环上的邻居结点将 N 加入路由表。其他诸如数据转移等操作，SkipNet 与 Chord、Pastry 类似，这里不再累述。通常情况下，SkipNet 新结点加入网络，需要发出的消息数为 $O(\log N)$。

SkipNet 的结点离开、失效算法与 Chord 很像：对于 SkipNet 而言，根环是最重要的，它保证了定位的正确性，很像 Chord 的主环，每当结点离开必须进行修正，而结点失效在主环上也必须能很快检测到；另一方面，SkipNet 的高层环用来加速定位，不影响正确工作，好比 Chord 的“Finger Table”，所以对它的修正要松散很多。只要主环正确，可以保证高层环正确修复。

和 Chord 的“Stabilization”一样，SkipNet 也有类似的“背景修复”操作。整个过程开始于根环，每个结点周期性地发送消息给根环上自己的一些邻居（通常在叶集中找），来检测它们是否可达，如果不可达就进行修复，找新的邻居替换不可达结点。修复好根环邻居以后，就可以进而修复 1 层环邻居，它基于正确的根环，因此能保证 1 层环修复的正确性。下面依次类推，每次修复 $i+1$ 层环都是基于前一步已经修复好的 i 层环。除了自底向上的方法，SkipNet 背景修复还有一个辅助性的局部方法：在某层环内，每个结点周期性地发送消息给它该层的邻居，告诉邻居：“我认为我是你的第 n 层环左（右）邻居”，邻居收到消息后，如果发现事实果真如此，那么不加回复，但如果事实不是这样，就回复消息说：“我认为我的左（右）邻居是 xxx，并不是你”，这就产生了一个“冲突”。结点知道“冲突”后，会进行一个互相协作的过程来修复自身状态，从而达到一致。

5. SkipNet 的局部性

SkipNet 的内容局部性和路由局部性是易于理解的，这两者基于 NameID 标识和 NameID 路由。在 SkipNet 的实现中，SkipNet 的设计者为 NameID 使用了 DNS 风格的名字，不过在适当的地方对 DNS 名反向以使结构更统一，比如 john.microsoft.com 写成 com.microsoft.john，从而所有在 microsoft.com 域中的结点拥有共同的前缀。这样的命名方案正是服务于 SkipNet 的内容局部性和路由局部性。

前文已说过，局部性和负载均衡是一对矛盾，如果完全按照上面的 DNS 风格 ID 来组织网络，这样的网络将会负载极不均衡。鉴于此，SkipNet 采用“受限负载均衡”（constrained load balance，CLB）的方法，在两者之间取一个折中，将数据对象名分成两个部分：第一部分是域名，它保留原有 DNS 名的前缀，也称 CLB 域

(CLB domain)；第二部分是原有 DNS 名后缀的散列值，也称 CLB 后缀(CLB suffix)。使用这样的对象名划分，既通过 CBL 域保留了局部性，又通过 CLB 后缀在 CLB 域内部保持了负载均衡，即所谓“受限负载均衡”。在 SkipNet 中，符号“!”被插入到原有 DNS 名中，以分隔前缀和后缀。

在“受限负载均衡”机制下，查询一个数据对象通常分两步来完成：首先，使用 NameID 路由算法找到一个结点所在的 CLB 域；然后，使用 NumericID 路由算法在这个 CLB 域内寻找带有 CLB 后缀散列值的结点。第一步的 NameID 路由算法与上文所讲的没有区别，但第二步的 NumericID 路由算法与上文的有所不同，因为此时的 NumericID 路由是在 CLB 前缀匹配的情况下做的，每当遇到 ID 界限时(这个界限由 CLB 域所界定)，查询必须反向走以确保没有结点被“漏掉”。两步加起来，路由跳数仍为 $O(\log N)$，但要比原来单纯的 NameID 路由或者 NumericID 路由跳数多不少。

除了内容局部性和路由局部性，SkipNet 也提供 Pastry 那样的物理网局部性，或者说拓扑一致性，这是依靠 SkipNet 结点维护一张 P-Table(Proximity Table，邻近表)和一张 C-Table(原论文中未解释此名称的含义)来实现的。P-Table 的作用很像 Pastry 中的邻居集 M，用来作为物理网上邻近结点的参照，不过 P-Table 的结构比 Pastry 的邻居集 M 要复杂：它将表分成多个指数级的“间隔”(intervals)，每个间隔保存一个邻近结点，作为物理邻近的参照。当新结点 N 加入 SkipNet 时，它发送消息(P-Table join message)给物理上离自己很近的一个结点(称为“种子结点”，seed node)，该消息为每个“间隔”维护一个常数项的链表，能容纳多个“候选结点”(比如 10 个)。“种子结点”收到消息后，使用自己的 P-Table 项来填充消息中的“间隔”作为“候选结点”，然后将消息发给自己的 P-Table 中离 N 最“远”的结点作为下一跳(这里的最“远”是就 ID 而言)，继续填充消息中的“间隔”，依次类推。当消息中每个“间隔”都有至少一个“候选结点”后，消息被回送给新结点 N 以初始化 N 的 P-Table。上述过程需要的消息数通常为 $O(\log N)$。

C-Table 的作用是辅助 NumericID 路由时的物理邻近度量。由于 SkipNet 的“CLB 路由”既使用了 NameID 又使用了 NumericID，所以它既使用 P-Table 又使用 C-Table；但由于 C-Table 的使用是在“CLB 路由”第二步，所以 C-Table 中在 CBL 域外的项此时不起作用。C-Table 是基于 NumericID 的，因此它与 Pastry 的路由表很像，由许多“层”组成，每层多匹配一位前缀。C-Table 的构建与 Pastry 的路由表构建几乎一样，这里不再累述。

6. SkipNet 增强机制

“松散路由表”vs“密集路由表”：

前面讲述 SkipNet 的 ID 时，都是二进制的，每个 i 层环分 $i+1$ 层环时也是分

成两个环，但实际上，完全可以采用更高的进制，比如三进制，每次分成三个环，在这种情况下根环仍然只有一个，但有3个1层环、9个2层环。假设采用B进制，随着B增加，R-Table中的指针数将会减少到$O(\log_B N)$，但路由跳数将会增加到$O(\log_B N)$。我们称这样的R-Table为“松散路由表”(sparse R-Table)。

对应于“松散路由表”，还有一种“密集路由表”(dense R-Table)，它是指在R-Table每层两个方向各额外存储$B-1$个邻近结点指针，类似Chord的“后继列表”。“密集路由表”减少了路由跳数，但增加了路由表项数。

路由表到底“松散”好，还是“密集”好，是很难界定的。SkipNet设计者实现时采用$B=8$，他们认为这是一个很好的平衡点。

重复指针删除：

因为在SkipNet中，h层环上相邻的两个结点在$h+1$层环上很可能也相邻，此时结点的h层指针和$h+1$层指针就是“重复”的。如果将重复指针中的一个删除，而替换成同样符合条件的邻近结点，将能改善路由表的性能。SkipNet的设计者们的实验证明，这样的“小技巧”使得路由性能提高了大约25%。

7. SkipNet对网络分割问题的解决

虽然由于路由局部性的存在，“网络分割”(network partition)问题对于SkipNet的影响要远远小于对其他结构化P2P网络，但此问题终归是要解决的。SkipNet网络分割算法的开销大概在$O(S*\log M)$，其中S为网络分割成的“子网”数目，M可以认为是“子网”中结点数的平均值。SkipNet解决网络分割问题分如下三步完成。

Recovery Step 1：Discovery Techniques(发现技术)

为了让每个SkipNet结点能及时发现自己所在的“子网”是被分割出来的，SkipNet设计者将每个组织内的结点分成一些“片段”(segments)，然后在每个片段中选择一些结点作为“众所周知的”。如图4.5.5所示，Microsoft可以将它自己分成两个片段：com.hotmail和com.microsoft。组织内的每个结点维护一个“众所周知结点”列表，使用它们作为不同片段之间的“连接点”。

因为每个SkipNet结点维护一个叶集，所以如果某个结点发现它的叶集某一半可达，而另一半却完全不可达，就表示发生了网络分割，而此结点自己正是分割的“边界点”。该“边界点”周期性地连接那些不可达的叶集结点，只要其中一个能连接到，“边界点”就可以开始进行片段的合并过程。合并分为两步，分别Step 2和Step 3中描述。

Recovery Step 2：Connecting Root Ring Segments(连接根环片段)

合并片段的第一步是连接根环片段，方法是：通过一个片段中的结点路由消息给另一个片段中的结点(采用NameID路由，因为是根环)，这个过程反复做下

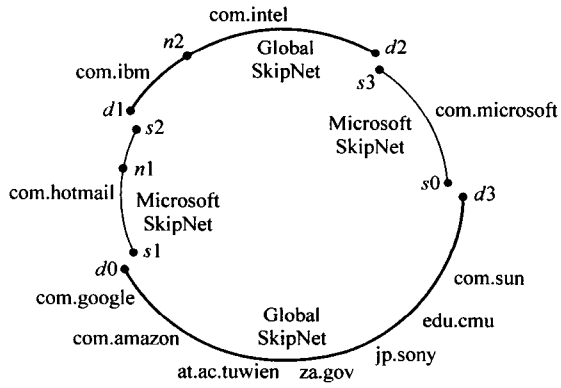


图 4.5.5 两个需要合并的 SkipNets 示例(一个是 Microsoft SkipNet,另一个是 Global SkipNet,每个 SkipNet 又包含两个片段)

(图片来自[Harvey et al.,2003a])

去,直到找到所有的片段"边界点",然后进行合并。以图 4.5.5 为例,整个 SkipNet 被分成两个 SkipNets,一个是 Microsoft SkipNet,另一个是 Global SkipNet,每个 SkipNet 又包含两个片段。为了合并 Microsoft SkipNet 和 Global SkipNet,首先 Microsoft SkipNet 中的结点 $n1$ 联系到了 Global SkipNet 中的结点 $n2$,然后 $n1$ 发送消息给 $n2$(通过 IP 方式直接发送),让 $n2$ 在 Global SkipNet 中路由一条消息到 $n1$。由于原有 SkipNet 已经被分割,$n2$ 路由的这条消息是不能到达 $n1$ 的,而将到达 $d1$,$d1$ 通过 IP 方式直接发送回复消息给 $n1$,告诉它 $d1$ 和其邻居 $d0$ 的存在。然后,$n1$ 在 Microsoft SkipNet 中向 $d0$ 路由一条消息,以相同的方式发现了"边界点"$s1$ 和 $s0$。这个过程递归地在 $s0$、$d0$ 中做下去,又发现了"边界点" $s2$、$s3$、$d2$、$d3$。在发现了所有的"边界点"之后,Microsoft SkipNet 和 Global SkipNet 的根环合并就很方便了。

Recovery Step 3: Repairing Routing Pointers following Root Ring Connection(通过根环连接修理路由指针)

根环合并好以后,SkipNet 路由已经能保证正确,剩下的是修理路由指针,也就是合并高层环。因为对路由指针的要求不像对根环指针那样严格,这可以通过周期性的"背景修复"操作来完成,但太慢了,所以需要用下面的方法。SkipNet 路由指针修理的方法是:使用根环来修理 1 层环指针,然后使用 1 层环指针来修理 2 层环指针,依次类推。

如图 4.5.6 所示,在两个 SkipNet 片段界限(segment boundary)处,根环上只需要连接两个"边界点"就可以了,但 1 层环上需要连接两对结点,一对用来合并 0

环，一对用来合并 1 环。类似地，2 层环上需要连接 4 对结点，分别用来合并 11 环、00 环、01 环、10 环。

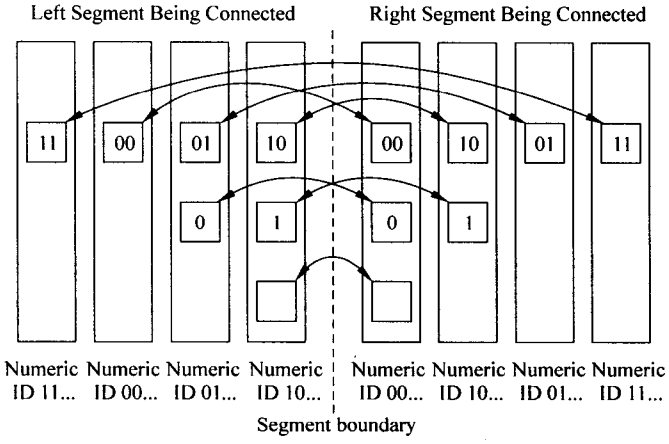


图 4.5.6　两个 SkipNet 片段界限处需要连接的指针
（图片来自[Harvey et al.,2003]）

4.6　常数度 P2P 模型——Viceroy、Koorde 和 Cycloid 等

4.6.1　常数度 P2P 模型概要

常数度 P2P 模型出现在经典的结构化 P2P 网络 Chord、CAN、Tapestry、Pastry 之后（对于 d 维 CAN 而言，如果 d 固定，实际上它也是常数度的）。最早的常数度模型是 2002 年发表于 PODC 的 Viceroy，然后是 2003 年发表于 IPTPS 的 Koorde，而 2004 年发表于 IPDPS 的 Cycloid 常数度模型，则可以看成是对此前众多 P2P 模型的一个总结。

常数度 P2P 模型目前并未得到具体的应用，其作用更多体现在理论上：它们有独特的性质——常数度。每个结点的“度”（即连接数）是固定的，不随网络规模而变；但它们的路由、定位、自组织方式与过去的模型区别并不大。常数度 P2P 网络在保持每个结点维护固定路由表项数的同时，仍然能达到 $O(\log N)$ 跳的定位效率，保持了系统的高可扩展性。同时，由于路由表项数固定，网络自组织、自适应的开销也相应减少了。

4.6.2 Viceroy——基于蝴蝶结构的常数度 P2P 模型

1. Viceroy 概要

"Viceroy"在英文中的意思是"副王峡蝶：一种橘色和黑色相间的北美蝴蝶(北美副王蛱蝶 杨蛱蝶属)，像但略小于大蝴蝶。"这个名称代表了 Viceroy 模型的最大特征：基于"蝴蝶"拓扑结构(butterfly topology)。

Viceroy 是最早出现的常数度 P2P 模型，它的提出基于对前面所讲的可变度结构化 P2P 模型在两个方面的改进：①经典的可变度结构化 P2P 模型通常维护 $O(\log N)$项的路由表，也就是保持 $O(\log N)$的结点度，虽然 N 的对数一般很小，但还是会随 N 变大，常数度模型则不会；②由于可变度结构化 P2P 模型自组织、自适应的开销通常和结点度紧密相关，要么是线性关系，要么是其平方，所以采用常数度模型减少了结构化 P2P 网络自组织和自适应的开销。在拥有上面两个新属性的前提下，Viceroy 仍然能保持 $O(\log N)$跳的定位效率，正是通过其底层"蝴蝶"拓扑结构来实现的。

除了"蝴蝶"结构，Viceroy 还将其覆盖网结点组织成一个与 Chord 非常类似的环形，每个 Viceroy 结点、对象都被分配一个随机、唯一的 ID，不过 ID 区间为 $[0,1)$。每个结点有一个逆时针方向的"前驱"、一个顺时针方向的"后继"，每个数据对象(的索引)由其"后继"负责，这些和 Chord 是一样的。因此，严格地说，Viceroy 是一个"蝴蝶"和"环形"相结合的 P2P 网络模型。

2. Viceroy 的拓扑结构

每个 Viceroy 结点被分配一个 nodeID，设为 x，$x\in[0,1)$。结点维护一个 7 项的路由表(正所谓"常数度"7)，前两项是 x 在环上的前驱和后继，分别记为 x. predecessor 和 x. successor。x 还有一个额外的属性"层"(level)，它表示结点 x 在"蝴蝶网"上的位置(如图 4.6.1 所示)，记做 x. level。结点 x 的 level 最初是一个随机选择的正整数，在$[1,\log N]$之间，N 为网络结点总数，随着网络的改变 x. level 也可能改变，下文会具体讲到层的选择算法。假设 x. level$=l$，就称 x 为一个"l 层结点"，它首先维护两条到 $l+1$ 层结点的连接：一条称为"右下边"(down-right edge)，连接到距离 x 大约 $1/2^l$ 且最近的那个 $l+1$ 层结点，记为 x. right；另一条称为"左下边"(down-left edge)，连接到离 x 最近的那个 $l+1$ 层结点，记为 x. left，"右下边"、"左下边"是路由表第 3、4 项。此外，x 还维护一条"上边"(up edge)，连接到离 x 最近的那个 $l-1$ 层结点(如果 $l=1$，该结点没有"上边")，记为 x. up，"上边"是路由表第 5 项。最后，x 维护两条"同层环边"，指向与 x 同在 l 层，但在环上离 x 顺时针、逆时针方向最近的两个结点，分别记为 x. nextonlevel 和

x.prevonlevel,“同层环边”是路由表第6、7项。图4.6.1描绘了一个理想的“蝴蝶”拓扑结构，出于简化的目的，只画出了每个结点的“左下边”和“右下边”。

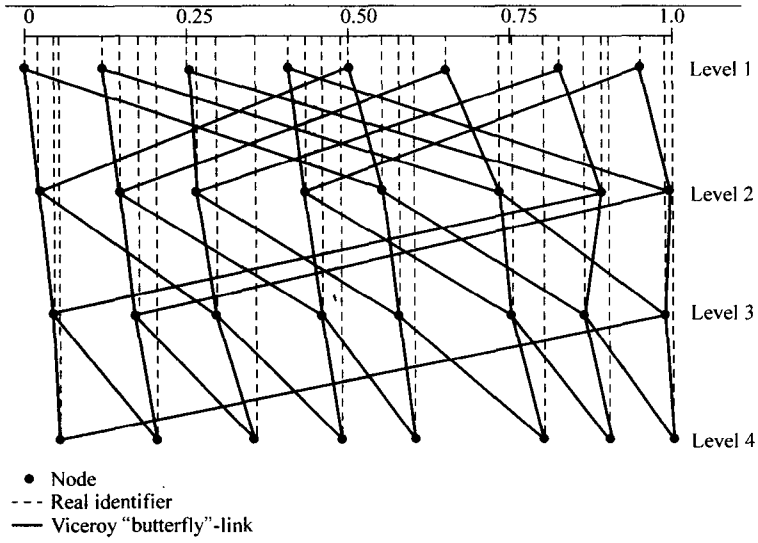


图4.6.1 一个理想的“蝴蝶”网。图中只画出了每个结点的“左下边”和“右下边”（图片来自[Malkhi et al.,2002]）

下面定义几个符号以便于下文的讲述：

$\mathrm{NLEVEL}_i(x)$：环上顺时针方向离结点 x 最近的 i 层结点。

$\mathrm{PLEVEL}_i(x)$：环上逆时针方向离结点 x 最近的 i 层结点。

NEXTONNEVER(x)：环上顺时针方向离结点 x 最近的，且与 x 同层的结点。

PREVONNEVER(x)：环上逆时针方向离结点 x 最近的，且与 x 同层的结点。

$d(x,y)$：环上顺时针方向从结点 x 到结点 y 的距离。

$q(x,y)$：环上顺时针方向位于(x,y)之间的结点数目。

$q_i(x,y)$：环上顺时针方向位于(x,y)之间的 i 层结点数目。

有了上面的符号以后，结点 x 的“左下边”即为 $\mathrm{NLEVEL}_{l+1}(x)$，“右下边”即为 $\mathrm{NLEVEL}_{l+1}(x+1/2^l)$，“上边”即为 $\mathrm{NLEVEL}_{l-1}(x)$。

由上文所述的路由表可以看出，Viceroy实际上构建了三个“连接集”：①“主环”(general ring)，通过“前驱”和“后继”连接形成；②“层环”(level rings)，通过“同层环边”，每一层的结点被连成一个环；③“蝴蝶网”(butterfly network)，任何一个 l 层的非叶结点通过两条“下边”连接到 $l+1$ 层的两个结点，并通过“上边”连

接到 $l-1$ 层的一个结点。正是这三个相互独立又相互联系的“连接集”，使得 Viceroy 既有经典的可变度 P2P 模型的对数定位效率，又有较好的容错性与自适应性。

3. Viceroy 层选择和结点加入算法

由于 Viceroy 是一个分布式的 P2P 网络，所以没有哪个结点能知道具体的网络结点总数，所以，在选择$[l,\log N]$之间的某个值作为其 level 时，这个 N 只能基于结点的局部估计。Viceroy 采用一个简单的方法估计 N：结点 x 认为 $N=1/d(x,\mathrm{SUCC}(x))$，$l$ 是 ID 空间范围，$d(x,\mathrm{SUCC}(x))$是 x 到其后继的距离，x 认为 $d(x,\mathrm{SUCC}(x))$是 Viceroy 环上两个邻接结点间距离的平均值。用这个方法计算 N 固然不准确，但还是合理的。

Viceroy 结点加入算法共分 5 步，前 3 步和 Chord 几乎一样，下面具体来讲：

JOIN STEP 1：生成 nodeID

使用一致性散列函数生成 nodeID，设为 s。

JOIN STEP 2：加入主环

使用查询子程序（查询将在下文讲到）找到 s 的后继，更新 s. successor 和 s. predecessor，然后再更新 s 的前驱的后继和 s 的后继的前驱，从而 s 将自己插入到 Viceroy 环中。

JOIN STEP 3：获得负责数据

s 从其后继结点中获得应当由自己负责的数据。

JOIN STEP 4：加入层环

s 使用开头讲过的方法选择自己的层 l，然后沿着 Viceroy 主环找到结点 $s'=$ NEXTONLEVEL（s）和 $s''=$ PREVONLEVEL（s），更新 s. nextonlevel、s. prevonlevel、s''. nextonlevel、s'. prevonlevel，从而在 Viceroy“层环”上插入自己。

JOIN STEP 5：加入蝴蝶网

s 沿着 Viceroy 主环找到左下边 $\mathrm{NLEVEL}_{l+1}(s)$，把它赋给 s. left；然后使用查询子程序找到顺时针方向离 $s+1/2^l$ 最近的结点，记为 s'，再沿着 Viceroy 主环找到 $\mathrm{NLEVEL}_{l+1}(s')$，把它赋给 s. right；最后沿着 Viceroy 主环找到 $\mathrm{NLEVEL}_{l-1}(s)$，把它赋给 s. up。

4. Viceroy 查询算法

我们首先介绍 Viceroy 的简化查询算法，它没有用到“同层环边”（即 x. nextonlevel 和 x. prevonlevel）。查询函数记为 LOOKUP(x,y)，y 表示发起查询的结点，x 表示目的结点的 ID，查询函数要做的是找到沿顺时针方向离 x 最近（或相等）的结点。

查询算法分3步，其中cur表示当前结点，初始时cur=y：

LOOKUP STEP 1：Proceed to root(走到根)

如果cur.level=1，跳到第二步Traverse tree；

否则：如果cur.up存在，令cur=cur.up，即往上层走；如果cur.up不存在，则令cur=cur.successor。这个过程循环下去，直到走到最上层(即第一层)的"根"。

LOOKUP STEP 2：Traverse tree(遍历树)

如果$d(\text{cur},x)<1/2^{\text{cur.level}}$并且cur.left存在，那么令cur=cur.left，即往左下走。

如果$d(\text{cur},x)\geqslant 1/2^{\text{cur.level}}$并且cur.right存在，那么令cur=cur.right，即往右下走。

如果上面两种情况所需要的"下边"不存在，或者存在但超过了目的地x，那么跳到第三步Traverse ring；如果所需"下边"存在且没有超过目的地x，则重复第二步操作。

LOOKUP STEP 3：Traverse ring(遍历主环)

如果cur已经是顺时针方向离x最近(或相等)的结点，则查询成功返回。

否则，令cur=cur.successor或者cur=cur.predecessor，哪个离目的地x更近选哪个。重复这个过程，直到查询成功。

Viceroy论文中证明了：上述简单查询算法中，前两步通常要走$O(\log N)$跳，N为网络结点总数，而第三步很可能要走$O(\log^2 N)$跳。为了将第三步开销降到$O(\log N)$跳，Viceroy的设计者改进了查询第三步，使用层环、主环遍历相结合以加快定位过程：

LOOKUP STEP 3′：改进版Traverse ring(遍历层环和主环)

如果cur已经是顺时针方向离x最近(或相等)的结点，则查询成功返回。

否则：如果cur.nextonlevel属于区间(cur,x)，那么cur=cur.nextonlevel；如果cur.prevonlevel属于区间(cur,x)，那么cur=cur.prevonlevel；如果上面两条都不能满足，令cur=cur.successor或者cur=cur.predecessor，哪个离目的地x更近选哪个。重复这个过程，直到查询成功。

4.6.3 Koorde——整合Chord、de Bruijn图的常数度P2P模型

1. Koorde概要

"Koorde"这个名字来源于荷兰语，数学上的"弦"(chord)在荷兰语中为"Koorde"，由此可以看出Koorde和著名P2P模型Chord的关系——Koorde将de Bruijn图(德布罗意图)嵌入到Chord中，从而既继承了Chord的简单、优美，又实现了比Chord

更好的、理论上接近最优的性能：①如果每个结点度为常数 2，其定位效率仍能达到 $O(\log N)$跳；②如果每个结点度为 $O(\log N)$，其定位效率能达到 $O(\log N/\log\log N)$跳。

与 Chord 一样，Koorde 使用一致性散列函数给结点和数据对象分配 ID，并将这些 ID 组织在一个环上。数据对象 k 由其后继结点 successor(k)负责，后继结点的定义与 Chord 相同：环上顺时针方向离 k 最近(或相等)的结点。

2. de Bruijn 图及其路由

Koorde 在 Chord 环上嵌入一张 de Bruijn 图来加速定位。de Bruijn 图将每个 ID 映射到一个 de Bruijn 结点，其中每个结点 m 有两条出边(从自己发出的有向边)：一条指向结点 $2m(\mathrm{mod}\ 2^b)$，另一条指向结点 $2m+1(\mathrm{mod}\ 2^b)$，其中 b 为 ID 位数，这正体现了 Koorde 的“常数度”属性。换个角度看，结点 m 指向的两个结点，一个是将 m 左移一位后最低位补 0，另一个是将 m 左移一位后最低位补 1。如果用符号 o 表示“模 2^b 的拼接”操作，则 m o $0=2m(\mathrm{mod}\ 2^b)$，m o $1=2m+1(\mathrm{mod}\ 2^b)$。图 4.6.2 是一个 $b=3$ 的 de Bruijn 图。

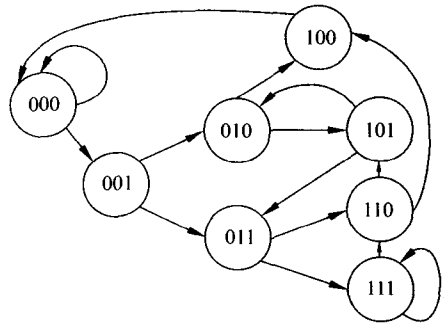


图 4.6.2 $b=3$ 的 de Bruijn 图

我们首先介绍 de Bruijn 图路由。假设一个理想化的网络，ID 空间中每个 ID 都对应一个结点，因此图中共有个 2^b 结点，de Bruijn 路由算法如图 4.6.3 所示。

```
procedure m. lookup(k,kshift)
    if k=m then return(m) / * m owns k * /
    else{
        t=m o topBit(kshift) / * o 表示拼接 * /
        return(t. lookup(k,kshift<<1))
    }
```

图 4.6.3 de Bruijn 算法

de Bruijn 路由算法中，m 为当前结点，k 为目的结点。kshift 初始时为 k，路由过程中它不断被取首位比特，再左移，用来缩小 m.与 k 的差异。为了到达 k，每一步(指一次递归调用 lookup()函数)m 被拼接上 kshift 的首位比特值(topBit(kshift))成为 t，然后再调用 t 的 lookup()函数来查找 k(参数 kshift 被左移一位)。这个过程递归下去，直到当前结点 $m=k$。实际上，上述过程中 t 离 k 将越来越近，每一步它们之间差异的比特位数少 1，因此 de Bruijn 路由算法将在 $O(b)$即 $O(\log N)$跳内到达目的地，而每个结点度仅为 2。

3. Koorde 路由

上述 de Bruijn 路由算法虽然非常高效，甚至可以说是最优的，但它只是一个理想情形。在实际的 P2P 网络中，ID 空间中不可能每个 ID 都对应一个实际结点，因此，Koorde 路由算法将 Chord 环形路和 de Bruijn 路由结合起来。具体地说，每个 Koorde 结点 m 除了指向两个 de Bruijn 邻居 $2m$ 和 $2m+1$ 外，还指向其环上的后继结点 m. successor 和环上 $2m$ 的前驱结点 d = predecessor($2m$)，容易看出，d 通常也是 $2m+1$ 的前驱。这样，实际的 Koorde 中每个结点度为常数 4。

图 4.6.4 所示即 Koorde 路由算法，其中 m，k，kshift 与 de Bruijn 路由算法中含义相同，但多了一个虚拟结点 i，它是 Koorde 路由算法的“主线”，引导着路由总体上按照 de Brujin 路由来走，而 d 与 successor 则不断弥补 m 与 i(现实结点与虚拟结点)之间的差异。

```
procedure m. lookup(k, kshift, i)
    if k∈(m, successor] then return(successor)
    else if i∈(m, successor] then return(
        d. lookup(k, kshift<<1, i o topBit(kshift))) /* o 表示拼接 */
    else return(successor. lookup(k, kshift, i))
```

图 4.6.4 koorde 算法

为了不断缩小当前结点 m 与目的结点 k 之间的差异，Koorde 引入了虚拟结点 i，其作用很像 de Bruijn 路由中的 m。kshift 初始时为 k，路由过程中它不断被取首位比特，再左移，用来缩小 i 与 k 的差异。为了到达 k，每一步首先检查 i 是否落在(m，successor]之间，如果是，则 i 被拼接上 kshift 的首位比特值，而 kshift 左移一位，然后再递归调用 d. lookup()函数；如果 i 不在(m，successor]之间，则调用 successor. lookup()函数，以 successor 平衡 m 与 i 之间的差异。上述过程递归下去，直到 k 在(m，successor]之间，表示定位成功。

Koorde 作者已经在其论文中证明：Koorde 路由算法定位跳数通常至多为 $3b$，即 $3\log N$，仍然符合 $O(\log N)$跳的定位效率。

4. Koorde 的容错性和自适应

Koorde 的容错性有些像 Chord。我们可以将 Koorde 中的 de Bruijn 指针类比 Chord 中的路由指针 fingers，在 Chord 中，即使 fingers 大量失效，但只要后继关系成立，定位就仍然能成功。类似地，Koorde 中 de Bruijn 指针也只是用来加速定位，不影响定位正确性。因此，Koorde 可以采用类似 Chord 的结点加入、离开算法，还可以采用 Chord 中的“Stabilization”自适应方法。此外，Koorde 还可以采用类似 Chord 的“后继列表”来保存多个后继，以增强系统容错性。

5. 增强的 k 度 Koorde

原来的 Koorde 使用两个 de Bruijn 指针，因此失效的概率就很高。虽然 de Bruijn 指针失效并不影响定位正确性，但定位效率却会因而非常低。鉴于此，“k 度 Koorde”将 2 度 de Bruijn 图扩展为 k 度 de Bruijn 图：每个结点 m 维护 k 个 de Bruijn 指针，分别指向 $km, km+1, \cdots, km+(k-1)$。结点变为 k 度后，一方面增强了 de Bruijn 指针的容错性，另一方面 de Bruijn 路由每次可以匹配 k 位，这就将定位效率从原来的 $\log N$ 跳提高到 $\log_k N$ 跳。

k 度 Koorde 对上述 k 个 de Bruijn 指针稍微做了调整：让结点 m 指向 km 的前驱结点和环上 km 前驱结点的 k 个后继。这和 k 度 de Bruijn 图本质上是一致的，因此定位效率仍然是 $O(\log_k N)$ 跳。

在 Koorde 论文中作者还证明了：对于一个网络而言，假设其中每个结点失效概率为 1/2，那么为了使网络以恒定的概率保持连通，其中某些结点度必须为 $\Omega(\log N)$（$\Omega()$ 为渐进增长率符号，若函数 $f \in \Omega(g)$，表示函数 f 渐进增长的速率超过或等于函数 g，数学上更严格地表述为：如果 $f \in \Omega(g)$，那么 $\lim_{N\to\infty} \frac{f(N)}{g(N)} > 0$，包括极限为无穷大的情形）。基于此，$k$ 度 Koorde 通常令 $k = \log N$，再通过类似 Chord 的后继列表维护算法，来保持网络的高容错性，同时，定位效率达到了 $O(\log_{\log N} N) = O(\log N / \log\log N)$，这正是本节开头就讲到的。

4.6.4 Cycloid——基于 CCC 的常数度 P2P 模型

1. Cycloid 概要

“Cycloid”意指“圆环”，其思想是用圆环去代替超立方体的每个结点，形成每个结点具有常数度的“带环立方体”（CCC，cube-connected-cycles）。假设原立方体是 d 维的，那么去取代原立方体每个结点的圆环也含有 d 个结点，并且每个结点度为 3：其中两条边用来维护与环上前驱、后继结点的连接，第 3 条边用来维护

与超立方体上其他环的连接(实际的 Cycloid 结点度为 7，这 3 个 CCC 邻居是其中一部分)。通过使用 CCC 这一特殊的拓扑结构，并采用与 Pastry 类似的定位、路由、自组织机制，Cycloid 在保持结点常数度的情况下，达到了 $O(d)$跳的定位效率。因为 CCC 结点数 $N=d\times 2^d$，因此 $O(d)$也就是 $O(\log N)$，所以说 Cycloid 拥有与经典的常数度 P2P 模型 Viceroy、Koorde 一样好的性能，某些方面甚至更好。

2. CCC 及 Cycloid 关键码分配

一个 d 维 CCC 是每个结点都被用一个包含 d 个结点的圆环所取代的 d 维超立方体，因此每个结点度为 3，任意两个结点之间的最大距离为 $2k-1+[k/2]$，其中[ ]表示下取整。每个结点用一个标识对$(k,a_{d-1},a_{d-2}\cdots a_0)$来表示，其中 k 为“环标识”，$a_{d-1}a_{d-2}\cdots a_0$ 为“立方体标识”，每一位取值从 0 到 $d-1$。图 4.6.5 描绘了一个 3 维 CCC 结构。

CCC 的基础是超立方体，而基于超立方体结构的著名 P2P 模型 Pastry 已被公认具有良好的性能，所以 Cycloid 设计时很多方面都参照了 Pastry。如图 4.6.6 所示，每个 Cycloid 结点维护一个路由表和两个叶集：“内叶集”、“外叶集”，加起来共 7 项，所以每个结点具有常数度 7。图 4.6.6 是结点(4,101-1-1010)的路由表和叶集，nodeID 有 8 位说明基于 8 维 CCC，x 代表通配符。路由表中维护一个“立方体邻居”(cubical neighbor)和两个“环邻居”(cyclic neighbor)，这将在下文中具体讲到。“内叶集”维护结点在自己所在环上的前驱和后继结点；而“外叶集”维护结点到“前驱环”和“后继环”上结点的指针，“前驱环”指环标识比当前结点环标识小但最接近的环，“后继环”则指比当前结点环标识大但最接近的环。

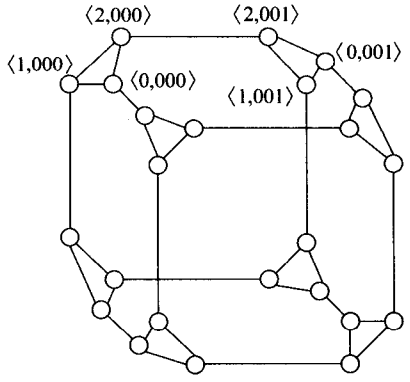


图 4.6.5 3 维 CCC 结构

NodeID(4,101-1-1010)	
Routing table	
cubical neighbor: (3, 101-0-xxxx)	
cyclic neighbor: (3, 101-1-1100)	
cyclic neighbor(3,101-1-0011)	
Leaf Sets(half smaller, half larger)	
Inside Leaf Set	
(3,101-1-1010)	(6,101-1-1010)
Outside Leaf Set	
(7,101-1-1001)	(6,101-1-1011)

图 4.6.6 Cycloid 结点(4,101-1-1010)的路由表和叶集

读者可能已经观察到了 Cycloid 立方体标识中的“-”，它的含义与 Pastry 中的“-”是不同的：假设一个 Cycloid 结点 nodeID 为$(k, a_{d-1}a_{d-2}\cdots a_k\cdots a_0)(k\neq 0)$，其“立方体邻居”应为$k-1, a_{d-1}a_{d-2}\cdots\bar{a}_k$xx…x，x 为通配符，$\bar{a}_k$ 表示 a_k 比特取反；而其“环邻居”应为$k-1, b_{d-1}b_{d-2}\cdots b_0$ 和 $k-1, c_{d-1}c_{d-2}\cdots c_0$，前者是以 $k-1$ 作为环标识、并且立方体标识比当前结点大但最接近的结点，后者是以 $k-1$ 作为环标识、并且立方体标识比当前结点小但最接近的结点。如果 nodeID 中 $k=0$，那么该结点没有立方体邻居和环邻居，其路由表为空。从上面的讲述可以看出：环标识 k 不仅代表了结点在环中的位置，更重要的是代表了路由表中立方体邻居与它的前缀匹配位数，k 为几则匹配几位，这在下面讲 Cycloid 路由时很重要。

对于 Cycloid 中每一个环上的结点而言，它们按环标识顺序串接起来，具有最大环标识的那个结点称为该环的“主结点”(primary node)；而环之间则通过环标识组织起来。

Cycloid 路由表中，立方体邻居使得路由过程中，可以从左到右逐步匹配目的 ID 的立方体标识，这和 Pastry 前缀匹配路由很像。相应地，环邻居的作用在于匹配环标识。Cycloid 叶集的作用是辅助路由，它帮助提高路由效率、检查路由终止条件并避免超过目的地。图 4.6.7 是结点(4,101-1-1010)的一种可能联结方式。

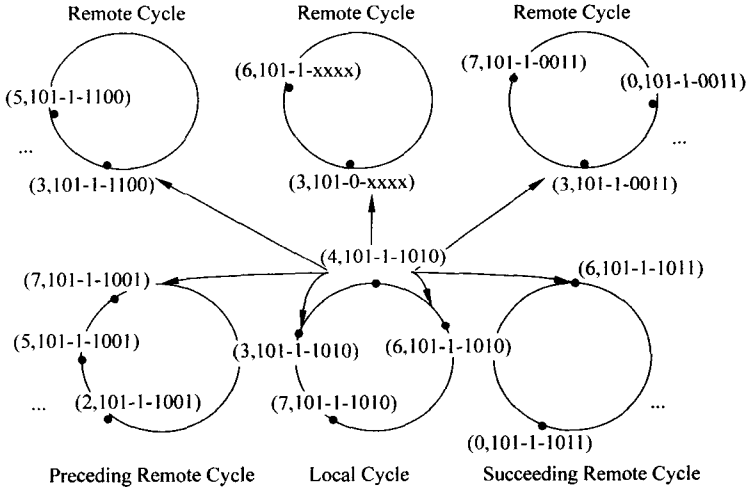


图 4.6.7 Cycloid 结点(4,101-1-1010)的路由表和叶集结点图示

最后，与 Pastry 类似，Cycloid 也使用一致性安全散列函数将结点、数据对象映射到覆盖网中，给它们分配随机、唯一的 ID，唯一的不同在于 Cycloid 标识是一对：(环标识，立方体标识)，通常环标识是将散列值模 d，立方体标识是将散列值除以 d。假设数据对象的 objectID 为 k，它被分配到结点时，首先考虑立方体标识接近，然后再考虑环标识接近。

3. Cycloid 路由

Cycloid 路由算法分三步。下文中定义 MSDB(most significant different bit)为当前结点 ID 和目的 ID 不匹配的位数。第一步“上升”的目的是使当前结点的环标识 $k \geqslant$MSDB，上文讲过，k 代表了当前结点路由表中立方体邻居与它的前缀匹配位数，所以这一步辅助地增加了当前结点 ID 和目的 ID 的匹配程度。第二步“下降”的目的是逐步匹配目的 ID 的立方体标识。第三步“遍历环”的目的是在叶集的局部范围内找到最终的目的结点。

Routing STEP 1：Ascending(上升)

当结点收到路由请求时，如果它的环标识 $k <$MSDB，那么它将路由请求发给外叶集中某个结点。上述过程一直做下去，直到 $k \geqslant$MSDB。

Routing STEP 2：Dscending(下降)

如果 $k =$MSDB，路由请求被发给当前结点的立方体邻居；

如果 $k >$MSDB，路由请求被发给当前结点的环邻居或内叶集结点，哪个最近选哪个；

做完第二步以后，收到路由请求的结点还会从第一步做起。

Routing STEP 3：Traverse Cycle(遍历环)

如果目的 ID 在当前结点叶集范围内，那么路由请求被发给叶集内离目的 ID 最近的结点。上述过程一直做下去，直到当前结点为最近结点，路由成功。

图 4.6.8 描述了 Cycloid 路由的一个例子：

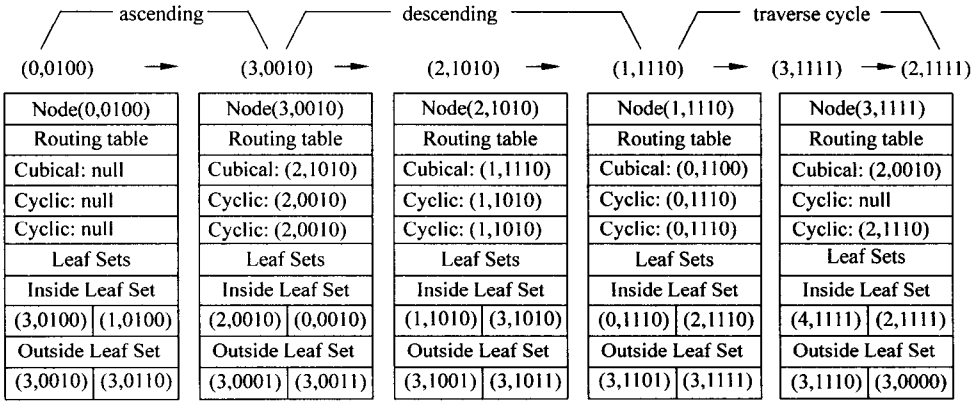


图 4.6.8 Cycloid 路由的例子，每个箭头代表一步

(1) 结点(0,0100)要发消息给(2,1111)，起初的 MSDB=3，因为 0100 与 1111 有 3 位不匹配，而(0,0100)的 $k=0$，显然 $k <$MSDB，所以要做第一步“上升”，消息

被发给外叶集结点(3,0010)；

(2) (3,0010)的 $k=3$，此时 MSDB=3，k=MSDB，所以要做第二步“下降”，消息被发给立方体邻居(2,1010)；

(3) (2,1010)的 $k=2$，此时 MSDB=2，k=MSDB，所以还是做第二步“下降”，消息被发给立方体邻居(1,1110)；

(4) 对于(1,1110)而言，(2,1111)在其叶集范围内，因此消息被发给距离(2,1111)最近的叶集结点(3,1111)；

(5) 对于(3,1111)而言，(2,1111)刚好是其叶集结点，它将消息直接发给目的结点，路由成功。

Cycloid 路由中每个步骤，走过的跳数都为 $O(d)$，因此整个路由过程的跳数也为 $O(d)$。上文已经说过对 Cycloid 而言，$O(d)$ 即等价于 $O(\log N)$，因此 Cycloid 路由是高效的。

4. Cycloid 自组织

结点加入：

与大多数结构化 P2P 网络一样，如果新结点 X 要加入 Cycloid 网络，它首先要联系到一个“自举”结点 A，通过 A 将以 X 为目的地的加入消息路由到离 X 最近的结点 Z，Z 的叶集复制给 X 作为 X 叶集的基础，这和 Pastry 结点加入几乎是一样的。然后，如果 X 和 Z 属于同一个环，那么 Z 的外叶集即成为 X 的外叶集。如果 Z 是 X 的后继，那么 Z 的前驱和 Z 本身成为 X 内叶集的左右结点，否则 Z 本身和 Z 的后继成为 X 内叶集的左右结点。

如果 X 是其所在环中仅存的结点，那么 X 和 Z 当然不同环，此时 X 的内叶集左右结点只能是 X 自己。如果 Z 所在环是 X 所在环的后继环，那么 Z 的外叶集左结点和 Z 所在环的“主结点”分别成为 X 外叶集的左右结点；否则，Z 所在环的“主结点”和 Z 的外叶集右结点分别成为 X 外叶集的左右结点。(这里有点像绕口令，注意体会)

路由表的初始化使用“先本地后远程”的方法：X 首先在自己所在环中按环标识降序搜索邻居；如果没找到，X 就搜索自己的邻居环(使用外叶集)，搜索方向取决于当前结点立方体标识的第 k 位——如果 $a_k=1$ 搜索方向是逆时针，如果 $a_k=0$ 搜索方向是顺时针，其目的在于增加找到邻居的可能性和速度。

结点 X 初始化好自己的叶集与路由表后，首先通知内叶集结点更新状态；然后，如果 X 的外叶集结点是所在环的“主结点”，X 也通知它们更新状态。外叶集结点更新了自己的状态后，还需要通知它们自己的内叶集结点更新状态。上述过程一直做下去，直到需要的更新完毕。

结点离开：

结点 X 离开之前，它应该首先通知内叶集中的结点自己将离开；如果 X 是所在环的“主结点”，它还要通知外叶集中的结点自己将离开。外叶集中的结点收到 X 离开的消息后，需要通知它所在环的所有结点 X 将离开，当然这可以很快完成，因为环上最多有 d 个结点。上面这些工作，只能更新那些以 X 作为其叶集项的结点的状态，不能更新那些以 X 作为其路由表项的结点的状态。后面这部分结点只能通过类似 Chord 的“Stabilization”自适应方法才能更新状态。

4.7 结构化 P2P 网络的特点与分析

4.7.1 覆盖网拓扑结构

结构化 P2P 网络最大的特点即在于它们都有一个严格的覆盖网拓扑结构，这也是它们区别于前两代 P2P 网络——混合式 P2P 和无结构 P2P 的主要特征。下面我们简单总结结构化 P2P 网络的几种主要拓扑结构：

(1) 带弦环　所有结点被组织在一个环上，环只提供两种功能——取得当前结点的前驱和后继(如果是单向环只提供后继，如 Chord)，只要后继关系正确，它就能保证正确定位。为了加速定位，往往在环上加入“弦”，即每个结点维护一个路由表，指向环上离自己很远的结点。采用环形结构的 P2P 网络有 Chord、Pastry、Kademlia、Cycloid 等，其中 Chord 是最纯粹的带弦环，其“弦”是指数间隔的，下标越大的弦指得越远；Pastry 混合了环形和超立方体，其“叶集”反映了环属性，当然也可以把 Pastry 路由表理解成一种“弦”；Kademlia 是基于异或度量的带弦环，其中的“弦”也是指数间隔的，但其指向很松散，没有 Chord 那样严格的要求；Cycloid 是带环的立方体结构(CCC)，首先每个结点都在一个小环上，这通过“内叶集”来维护，其次每个小环有其“前驱环”、“后继环”，从而将小环也串成了一个大环，这通过“外叶集”来维护。

(2) 多维空间　所有结点被组织在一个多维笛卡儿空间里，每个结点有自己在空间中的邻居。采用多维空间的 P2P 网络是 CAN，多维空间结构可以更严格地说成是“多维环面(Torus)”结构。

(3) 超立方体(或者 Plaxton Mesh)　所有结点被组织在一个超立方体中，采用此结构的有 Tapestry、Pastry 和它们的前驱 Plaxton。实际上，这三种结构化 P2P 网络都不能算作严格的超立方体结构，因为它们的覆盖网采用了称为 Plaxton Mesh 的拓扑结构，每层维护匹配 nodeID 不同长度前缀(或后缀)的结点，基于这样的路由表采用的前缀(或后缀)匹配路由很像超立方体路由(但并不相同)，因此得名。Cycloid 的基础 CCC 是基于超立方体改造的，如果将其中的小环看成结点，Cycloid 结点组织倒是很像超立方体。

(4) 蝴蝶形　蝴蝶网中，每个结点有它的“层”(level)，每层的结点通常维护两个下边、一个上边以及两个同层边。采用蝴蝶结构的 P2P 网络有 Viceroy 等，它主要基于蝴蝶网，不过每个 Viceroy 结点还保存一个前驱、一个后继，从而又组织成了一个全局的环结构。

(5) de Bruijn 图　de Bruijn 图中，每个结点 m 有两条出边：一条指向结点 $2m \pmod{2^b}$，另一条指向结点 $2m+1 \pmod{2^b}$，其中 b 为 ID 位数。Koorde 将 de Bruijn 图嵌入到 Chord 环中，提高了路由效率。

(6) CCC　一个 d 维 CCC 是每个结点被一个包含 d 个结点的圆环所取代的 d 维超立方体，因此，每个结点度为常数 3。Cycloid 是基于 CCC 结构的常数度 P2P 模型。

(7) 其他形状(如跳表)　除上述几种最经典的拓扑结构外，还提出了很多种适合于结构化 P2P 网络的拓扑结构，比如基于“跳表”(SkipList)的 SkipNet 等。

4.7.2　分布式散列表

所有的结构化 P2P 网络都使用分布式散列表(DHT)来将结点、数据对象映射到覆盖网中。为了使这种映射唯一、均匀、随机，分布式散列表都使用安全的一致性散列函数，其中最著名、也是被大多数 P2P 系统采用的安全散列函数是 SHA-1(安全散列算法)，它能产生均匀、随机、与输入无关的 160 位散列值，并且散列值冲突的概率极小(注：山东大学的王小云教授分别于 2004 年、2005 年破解了 MD-5、SHA-1 散列算法，因此它们不再是“安全”的，很多系统采用了更新版本、更长散列值的 SHA 算法，但这与 P2P 网络关系并不大，因为替换散列函数是一项并不复杂的工作)。

4.7.3　路由和定位

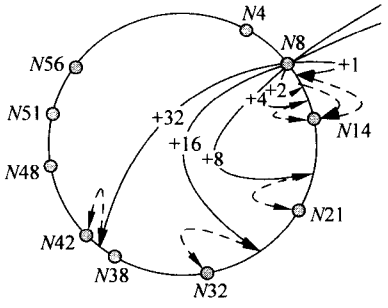
路由和定位的方式通常取决于两个因素：①覆盖网拓扑结构，②路由表结构。基于这两个因素，结构化 P2P 网络通常都维护一个比较小的路由表(可变度或者常数度)，采用分布式、局部性的贪心路由算法，逐步缩小当前结点与目的结点之间的 ID 差异。通常定位效率为 $O(\log N)$跳，并且能保证定位成功，单就覆盖网而言此定位效率接近最优。结构化 P2P 网络主要的路由方式有如下几种。

1. 数值邻近路由

这里的“数值”通常指结点 ID 值。路由过程中每一步，当前结点都在自己的路由表中选择与目的 ID 最邻近的结点作为下一跳，因此，路由路径中每一跳的结点 ID 与目的 ID 的差距会越来越小，直至最接近目的 ID 的结点为止。广义上讲，几乎所有的结构化 P2P 路由算法，都可以算是“数值邻近”的，只不过路由表构造不

同(比如 Chord 使用指数间隔的“Finger Table”，而 Tapestry、Pastry 使用来自 Plaxton 的层次化路由表)，或者是对于“邻近”的度量不同(比如 CAN 的“邻近”是指空间位置邻近，而 Kademlia 则基于异或值来度量距离)。

Chord 是最典型的数值邻近路由(如图 4.7.1 所示)，它的路由表(Finger Table)中每个表项指向的结点到当前结点的距离是指数递增的，因此 Chord 路由的每一步能将前一步距离缩短至少一半，从而获得了 $O(\log N)$ 跳的定位效率。Kademlia 路由表(k-buckets)很像 Chord，每个 k-bucket 链表的覆盖范围也是指数递增的，不同点首先在于 Kademlia 以异或值度量距离，其次每个路由表项是一个含有多个结点的链表，再次这些结点的选取是松散的、不断更新的，不像 Chord 那样严格，所以 Kademlia 路由更容错、更实用。



Finger Table

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

图 4.7.1 Chord 路由举例。结点 N8 发出一条往 N56 的消息，实线箭头代表右图路由表的左项，虚线箭头代表路由表右项(实际结点)。基于数值邻近的路由算法，N8 将选择 N42 作为下一跳，因为 N42 离 N56 最近

(图片来自[Lua et al.,2004])

2. 逐位匹配路由

逐位匹配路由出自 Plaxton，其后继 Tapestry、Pastry 继承并扩展了它，前者采用后缀匹配，后者采用前缀匹配，其实并没有什么差别。基于层次化的路由表，每一步通常都能与目的 ID 多匹配至少一位，因此逐位匹配路由的效率等价于 ID 的位数，也就是 $O(\log N)$。

Koorde 采用的 de Bruijn 路由，每次将当前结点 ID 左移再拼接上未匹配的一位，实际上也是逐位匹配路由。

3. 位置邻近路由

CAN 是最典型的位置邻近路由(如图 4.7.2 所示)，每个结点的路由表记录自己在多维空间中的邻居，每次选择离目的结点最近的邻居作为下一跳，其定位

效率等价于多维空间的直径 $O(d\sqrt[d]{N})$，如果取 $d=\log N$，$O(d\sqrt[d]{N})$ 也就成了 $O(\log N)$，与其他的结构化路由方法是一样的。

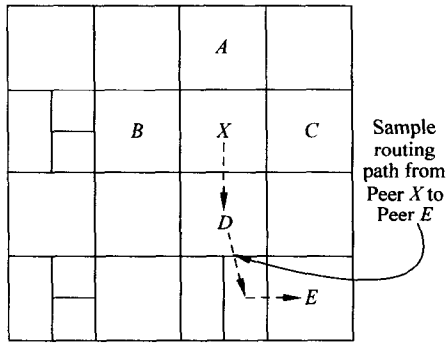


图 4.7.2 CAN 路由例子，X 要发送消息给 E 所走的路由路径
（图片来自[Lua et al.,2004]）

4. 层次路由

不少 P2P 网络将结点组织到多个层次上，路由过程常常是先从低层爬到高层，再从高层爬到底层。Viceroy 采用了蝴蝶网层次路由，通过其路由表中的“上边”、“下边”，首先沿“上边”走到“根”即 1 层结点，然后从 1 层结点出发沿“下左边”或者“下右边”往下一层走，就好像从一棵树的根走到叶子。SikpNet 的 NumericID 路由也是层次化进行的：首先查找“根环”，直到发现一个结点与目的结点的 NumericID 匹配第一位，再通过这个结点“爬升”到它的 1 层环，在 1 层环上查找与目的结点 NumericID 匹配前两位的结点，依次做下去，每次上升一层多匹配一位，直到不能匹配更多位、又不能找到 NumericID 更接近的结点为止。Viceroy 和 SkipNet 中结点都被组织到 $\log N$ 个层次，所以路由效率都是 $O(\log N)$ 跳的。

5. 混合式路由

大多数结构化 P2P 网络的路由方式都不是单一的，如 Chord、Pastry、Viceroy、Koorde、Cycloid 中都使用了环形路由（即沿着环一个一个走，因为太简单不专门分类写出）作为基础，但又结合了各自独特的路由方式。Cycloid 是一个典型的混合式路由的例子，它使用了类似 Pastry 的超立方体路由（属于逐位匹配路由），又结合了两种环形路由：一是主环上的环形路由，二是每个小环上的环形路由。不管采用多少种方式结合，结构化 P2P 网络的路由效率在 $O(\log N)$ 跳是不会变的。

4.7.4 动态结点算法(自组织、自适应)

1. 结点加入

几乎所有结构化P2P网络的结点加入算法都大同小异，无非是分如下三步：

JOIN STEP 1：新结点 N 以某种方式找到一个网络现存结点 G，这也称为“自举”。

JOIN STEP 2：N 通过 G 发送以 N 为目的地的消息，该消息最终到达ID与 N 最接近的结点 Z，N 从 Z 或者从消息路径的每个结点中获取路由表信息以及应由自己负责的数据，之后再做修正和优化。

JOIN STEP 3：通知网络其他结点更新路由表以反映 N 的到来。这里的“其他结点”通常是 N 根据其路由表中的邻居来选择的。

结构化P2P网络的结点加入算法要远比混合式、无结构P2P网络复杂，原因就在于要保持覆盖网拓扑结构。通常结点加入的开销为 $O(\log N)$ 或者 $O(\log^2 N)$。

2. 结点离开和失效

对照上面的结点加入算法，不难想象结构化P2P网络中结点离开时要做的工作，其实也就是通知那些把自己当邻居的结点更新它们的路由表，同时把自己负责的数据交给相应的结点。结点离开的开销和结点加入差不多，因为它就像加入算法中STEP3的“逆过程”。

如果每个结点都是合法离开的，结构化P2P网络实现起来就远不像现在这么复杂。复杂的根源在于网络的动态性和不确定性：很多情况下结点离开是突然的、不合法的，这称为“失效”，此时其他结点没有办法知道失效的结点是谁，因此也无法做相应的更新，网络拓扑结构不能维持。处理失效的第一步是检测失效，现实的方法通常是周期性地联系路由表中的邻居，看是否可达，如果不可达再进行第二步：路由表修复。

对结点失效的处理是结构化P2P网络最大的开销所在。

4.7.5 容错性与安全性

结构化P2P网络的容错性、安全性通常不及无结构P2P网络，因为拓扑结构越严格、定位越准确，维护起来就越复杂，暴露给攻击者的弱点也就越多。

为了提高容错性，结构化P2P网络的设计者多从路由表上做工作：Chord使用“后继列表”来取代单后继；Pastry的“叶集”则进一步，既保存多个后继又保存多个先驱；Kademlia的方法与前两者皆不同，它只松散地保存一些邻居，并且仅根据收到的消息就能“捎带更新”路由表，所以根本不需要维护后继关系的正确性。

另一方面，某些拓扑结构本身就具有很高的容错性，比如 CAN、Tapestry 和 Pastry，定位路径原本就有多个选择，即使很多结点失效也一样能定位成功。

结构化 P2P 网络的安全性难以完备，也更复杂。现有系统都实现了某个方面的安全机制，但通常是很不够的，也没有出现像无结构 P2P 网络中 Freenet 那样安全、匿名的系统。三个最著名的结构化 P2P 应用系统——CFS、OceanStore 和 PAST 都通过散列方法来保证数据完整性，尤其是 OceanStore 中使用了层次化的 AGUID、VGUID、BGUID 来逐层认证。除了数据完整性，路由表项的真实性也是容易受攻击的，CFS 中使用“现时”(Nonce)回馈的方法防止恶意结点伪造 ID 或虚报 IP。PAST 系统中使用称为“智能卡”的设施来提供安全性：“智能卡”产生和验证各种各样的证书以及维护结点的“存储限额”(防止恶意结点滥用网络资源)。“智能卡”虽然很好，但必须由可信的权威方来发行，这就引入了集中式的办法，破坏了 P2P 系统分布式的工作模式。

“覆盖网分割”问题是结构化 P2P 网络的难题，也可以说是 P2P 领域的一大难点，因为对于无结构 P2P 网络，分割问题也是很难解决的。结构化 P2P 网络比起无结构 P2P 网络更容易被分割，并且一旦被分割受到的损害要严重得多，原因就在于它们结构严格，所以攻击者只要让某些关键性的结点失效(比如 DoS 攻击)，就可以有效地阻碍整个网络的工作。虽然人们提出很多方法来解决网络分割问题，但并没有一个公认的高效、实用方法，“覆盖网分割”依旧是 P2P 领域的一个开放性问题。

常数度 P2P 模型在保持 $O(\log N)$ 路由效率的同时，每个结点只需要维护常数个邻居，从而自适应开销比可变度 P2P 网络小很多。然而，“常数度”是以其他性能的损失为代价的，最明显的就是容错性和安全性：在容错性上，由于路由表项数少，所以一旦邻居失效，要修复通常比较困难，而且在修复的这段时间里网络工作效率会有明显的下降；在安全性上，常数度 P2P 模型要更容易受攻击，比如由于网络总体边数比较少，所以更容易被分割成多个不交的子网。

4.7.6 局部性

局部性的目的通常在于提高覆盖网与物理网的一致性，从而减少通信时延。CFS 在其 Chord 层加入了服务器选择方法，Pastry 则通过每个结点维护“邻居集”来选择物理网上真正邻近的结点，从而减少时延。Tapestry 的方法与前两者不同，它将每个数据对象复制多份放到网络中，在查询时帮助用户自动定位到很近的副本，从而在“概率”上减小了时延。

除了上述的网络局部性，语义局部性也是很有意义的，如 2003 年提出的 SkipNet 网络，基于“跳表”数据结构，提供路由、数据内容两个方面的语义局部性。

4.7.7 增强机制：复制、缓存和分片

结构化P2P网络普遍采用“ID邻近复制”，即将数据对象复制到ID邻近的k个结点上。这样做有两个原因：一是ID邻近的结点能很快找到，二是ID邻近的结点往往均匀散布在网络中，它们同时失效的可能性很小，因此提高了数据可用性。

缓存的目的与复制不同，虽然也可以提高数据可用性，但主要是为了提高定位速度（缓存对象索引）和获取速度（缓存对象本身）。结构化P2P网络基本上使用“路径缓存”，将数据缓存到查询路径上，其原因就在于结构化P2P网络由于对象位置确定、拓扑结构严格，不同结点查询同一对象的路由路径往往会有重叠，并且越接近对象保存的结点，路由重叠的可能性越大。

CFS中将每个文件分块，而使用DHash中间层来分布和获取这些分块。OceanStore中的数据分片与CFS不同，其目的是为了将数据归档存储以提供持久性，它采用了“冗余编码”方法（erasure code）将数据对象分片冗余地存储在网络多个结点中，只要获得一部分分片就可以重构原文件，因此数据具有高可用性。PAST没有提供分片方法，而是使用“副本转移”和“文件转移”的方法应付文件过大或者结点能力过低的情况。

4.7.8 P2P网络各项属性总结

在本章的最后，我们列表总结第2、3、4章所讲过的各种P2P网络的各项属性，从而对它们做一个深入的比较（参考论文[Lua et al.,2004]和[Wang & Li,2003]）。其中TTL表示跳数限制，N表示网络结点总数，d表示维度，B表示ID的进制，$|L|$表示叶集大小，$|M|$表示邻居集大小。

分类	P2P网络	拓扑结构	路由算法	路由效率	路由表项（结点度）	结点加入/离开的开销	容错性	安全性
混合式P2P网络	Napster	星形	服务器	$O(1)$	$O(1)$	$O(1)$	服务器单点	无
	BitTorrent	星形	服务器	$O(1)$	$O(1)$	$O(1)$	服务器单点	一般
无结构P2P网络	Gnutella	随机图	洪泛法	TTL	$O(1)$	$O(1)$	很好	低
	KaZaA	双层结构（随机图、星形）	超结点、洪泛法	$O(1)$+TTL	$O(1)$	$O(1)$	良好	低
	eDonkey/eMule	双层结构（随机图、星形）	服务器、洪泛法	$O(1)$+TTL	$O(1)$	$O(1)$	良好	一般
	Freenet	随机图	基于对象的洪泛法	HTL（Hops To Live）	$O(1)$	$O(1)$	很好	匿名

续表

分类	P2P 网络	拓扑结构	路由算法	路由效率	路由表项（结点度）	结点加入/离开的开销	容错性	安全性
结构化 P2P 网络	Chord	带弦环	数值邻近路由	$O(\log N)$	$\log N$	$\log_2 N$	一般	低
	CAN	多维空间	位置邻近路由	$O(d\sqrt[d]{N})$	$2d$	$2d$	良好	一般
	Tapestry	超立方体（Plaxton Mesh）	后缀匹配路由	$O(\log_B N)$	$B*\log_B N$	$\log_B N$	良好	一般
	Pastry	超立方体（Mesh）、环形	前缀匹配路由	$O(\log_B N)$	$B*\log_B N+\lvert L\rvert+\lvert M\rvert$	$\log_B N$	很好	一般
	Kademlia	基于异或的带弦环	数值邻近路由	$O(\log N)$	$O(\log N)$	$\log N$	很好	一般
	SkipNet	跳表	数值邻近路由、前缀匹配路由	$O(\log N)$	$O(\log N)$	$O(\log N)$	很好	局部可控
	Viceroy	蝴蝶网、环形	蝴蝶路由、环形路由	$O(\log N)$	7	$\log N$	一般	低
	Koorde	de Bruijn 图、环形	de Bruijn 路由、环形路由	$O(\log N)$	4	$\log N$	一般	低
	Cycloid	CCC、环形	CCC 路由、环形路由	$O(\log N)$	7	$\log N$	一般	低

第5章 P2P应用体系和应用软件

“执古之道，以御今之有。”在书的前半部分，也就是第1、2、3、4章，回顾了P2P网络从出现到今天的历史——

第一代：混合式P2P网络到

第二代：无结构P2P网络，再到第三代：结构化P2P网络。

历史的作用并不在于让人们“知道过去发生了什么”，而是要告诉人们“今天我们拥有什么、又可以做什么?”第5章的意义正在于此。在本章中，将侧重于“应用”二字，其目的是向每一个人(懂不懂计算机都没有关系)讲明白：在学术界和商业界对P2P做了如此多的工作之后，到底存在哪些现实可用的P2P网络，到底有哪些我们现在就可以从网上下载并使用的P2P应用软件，当然最重要的是——这些软件能做什么、能给我们带来什么、又到底如何使用?

5.1 P2P应用清单

表5.1.1给出一个清单，它既是本章各小节的提纲，也是当前存在的P2P应用体系和应用软件的分类总结。读者可以把它看成当前世界的一份不完全但很具代表性的“P2P应用清单”，或许有些名字你曾听说过，也说不定哪个软件你正在使用呢。

表 5.1.1 P2P 应用清单

P2P 应用领域	P2P 应用体系/应用软件		简　介
文件共享	Napster		Napster 是世界上第一个应用性 P2P 网络，也是混合式 P2P 网络的代表作，正是它向世界宣告了 P2P 的到来。不幸的是，真正的 Napster 已经不存在了。
	BitTorrent（简称 BT）	ABC	BitTorrent 是以分散化的服务器为核心提供文件共享的 P2P 网络，它提供文件分片、多源下载，同时限定用户在下载的同时必须上传。而网络和用户信息的更新，尤其是 BT 种子的维护，是依靠服务器中的 Tracker 来完成的，下载同一文件的用户围绕 Tracker 形成一个独立的子网。
		Azureus	
		BitComet	
		BitSpirit	
		BitTorrent	
		…… 太多支持 BT 的软件	
		eXeem	eXeem 在很多方面非常像 BT，关键的不同在于 Tracker 的处理：eXeem 有它自己的分散网络，每个用户可以自动成为 Tracker；而 BT 中对于一个文件而言 Tracker 是固定的。
	Gnutella	Gnutella	Gnutella 是无结构 P2P 网络的先驱和代表。Gnutella 中只有一种结点——Peer，不再有服务器存在，每个 Peer 既能向其他 Peer 发送查询请求并获得查询结果，又能接收其他 Peer 发来的查询请求，返回所要的文件信息，或者将此请求转发给其他的 Peer。
		BearShare	
		Gnucleus	
		Gtk-Gnutella	
		LimeWire	
		Morpheus	
		Mutella	
		Phex	
		Qtella	
		Swapper	
		Xolox Ultra	
	FastTrack	KaZaA	FastTrack 最大的特点是将网络结点分成两类：超结点和普通结点，以此来开发 P2P 网络的异构性。超结点之间周期性地交换信息是 FastTrack 工作的核心，而普通结点仅仅连接到超结点。
		KaZaA Lite	
		Mammoth	
	eDonkey/Overnet	eDonkey（电驴）	eDonkey(电驴)将网络结点组织成两层：服务器层和客户层。eDonkey 将文件分块，分块又分成片段，片段进一步分成小块，从而提供多源文件下载机制和更细粒度的数据完整性检查。Overnet 是 eDonkey 所使用的分布式搜索网络，它本身是一个独立的应用，但 eDonkey 将它整合到自己的体系中。
		mlDonkey	
		eMule(电骡)	
		aMule	
		xMule	
		MobileMule	
		Iphant	

续表

P2P应用领域	P2P应用体系/应用软件		简　介
文件共享	Ares	Ares P2P	Ares是一个非常不错的P2P文件共享体系，2005年度的“Google搜索进步排行”第2名。Ares的优点在于能快速地搜索和下载文件，并具有自设代理服务器、自订聊天社群、内建多媒体播放器、内建浏览器等功能。
		Ares Galaxy	
		Warez P2P	
	Direct Connect	DC	与FastTrack、eDonkey类似，Direct Connect将网络结点组织成两层：Hub层和客户层。客户先连接到Hub，再获得文件信息列表
		DC++	
		RevConnect	
	Maze		北京大学网络实验室在2003年夏天开发了Maze这样一个混合式的P2P个人信息中心(PIC)文件系统。它最先依托于北大天网，目的是为了解决FTP服务器无法有效下载的缺陷。今天Maze有广泛的应用，可以看成是国内P2P一个成功的先驱。
	国内P2P文件共享软件(企业开发)	PP点点通	国内有很多优秀的P2P文件共享软件，其中一些颇有影响力。如前文所说，它们之所以难以达到像BT、eDonkey这样的流行，原因并不在软件或者体系本身，而是源于这样一条“先人为主、因果互导”的P2P软件准则：“用户越多，则越流行；越流行，则越好用；而越好用，则用户越多。”
		PoCo	
		百宝	
		“ZCOM智通”P2P网络杂志	
		卡盟	
		酷宝	
		酷狗	
		“百兆”P2P浏览器	
		百度下吧	
		SouGood	
	多网络中间件	giFT	提供一个中间层P2P软件，以支持多个P2P体系，或者说不限定使用哪种P2P网络，相当于整合了多个P2P网络的一个搜索引擎。
		iMesh	
		iSwipe	
		Jubster	
		Kiwi Alpha	
		MLDonkey	
		Shareaza	
		Trusty Files	
		Xolox	

续表

P2P 应用领域	P2P 应用体系/应用软件		简　介
文件共享	Freenet(自由网)		Freenet 的目的是共享 Internet 计算机资源组建一个自由、安全、匿名的信息发布和获取平台。它利用每个参加者作为一个结点，每个结点划出一部分硬盘作为公用存储空间，其中存放某个文件的某些数据，不过是全部经过加密的，所以即使是硬盘的主人也无法知道其中到底存放了些什么内容。
	Mute	Mute MFC Mute NapShare	Mute 最大的特点是：用户不是直接相连，而是通过一系列超结点来连接。为确保安全、匿名，通信用 RSA 算法加密，使用无规律的虚拟 IP 地址而非实际的，因此对于文件真正的发送者、接受者，用户一无所知，“Mute”(沉默的)的含义也正在于此。
	Waste		Waste 来自于著名的 NullSoft 公司，不幸的是它已经关闭了。从 Waste 的宣言：“匿名、安全、加密共享”就可以明白它是干什么的、为什么关闭。
	FileTopia		FileTopia 是一个安全、高效的文件共享网络，它的搜索效率值得称道，同时保护用户的隐私和安全(如 IP 地址编码)，此外还具有聊天等功能。或许 FileTopia 唯一的问题在于用户太少。
	FolderShare(文件夹共享)		FS 让用户创建他自己的 P2P 网络，从而和自己的朋友们共享文件夹。
	Open Media Network(开放媒体网络)		OMN 的作者是著名的 Netscape 软件的设计者，OMN 被设计用来只发布合法的资源：用户共享资源的同时，保护资源生产者的版权。
	Furthur		Furthur 仅仅用来共享音乐，并且它保证了只有经音乐所有者授权的音乐才可以被下载，从而保护了版权。Furthur 的目的或许在于告诉音乐圈：P2P 并不是天生与版权为敌的，虽然 Napster 被认为这么做了。

续表

P2P应用领域	P2P应用体系/应用软件		简　介
多媒体传输	PeerCast(对等广播)		PeerCast是一个不错的P2P广播网络，它帮助用户寻找各种格式的音频/视频流媒体资源。用户既可以播放一个频道，也可以创建一个频道。
	AnySee		Anysee是由华中科技大学集群与网格计算实验室P2P小组于2004年夏天开发的一个视频直播软件，使用P2P技术(应用层组播)解决教育网内网络电视服务器难以服务众多用户的问题，使更多的用户可以观看和发布网络电视频道。
	Mercora		Mercora是一个非常好的“P2P电台”，既能收听，又能广播。Mercora很简单，但很好用。
	Skype		Skype是一个优秀的网络语音传输工具，既提供高清晰的语音对话，还可以用来拨打国内国际电话。除此之外，Skype也提供网络聊天、传文件等功能。Skype流行的另一大原因是它简单、通用。
	国内P2P网络电视软件	PPLive TvAnts CCIPTV CoolStreaming QQ直播 ……	“网络电视”并不是一个新课题，但P2P真正适应了它对网络带宽的巨大需求，以C/S方式播放网络电视给服务器造成的负担是无法承受的。虽然国内P2P网络电视软件使用起来并不尽如人意，但这一趋势是不可逆转的。
实时通信	Skype		Skype不仅是一个优秀的网络语音传输工具，还是一个优秀的P2P实时通信工具。
	Jabber	Google Talk	Jabber是由开放源码组织开发的实时消息传输平台，它基于XML语言，其目的在于建立一种让所有实时通信系统之间能够互操作的开放式协议。Google Talk正是基于Jabber的XMPP协议开发的。
	支持C/S、P2P两种模式的实时通信软件	QQ PoPo MSN Messenger ICQ	这些耳熟能详的实时通信软件似乎不需要我们做更多的描述。它们最初都是客户/服务器(C/S)模式的，因为这对于文本消息传输已足够了；但今天，实时通信的功能早就不限于文字传送，还包括声音、视频、在线游戏等，它们对传输率、时延的高要求，注定了C/S方式被用户间直接建立连接的P2P方式所取代。
协同工作	Groove		Groove早在1997年由Lotus Notes公司创办，因开发著名的协同工作软件“虚拟办公室”而出名。Groove系统以P2P的方式提供实时的文本消息、语音、视频传输，而提供这些功能的目的在于支持互联网上的协同工作。2005年3月Groove被微软公司收购。

续表

P2P 应用领域	P2P 应用体系/应用软件	简　介
分布式数据存取	CFS(协同文件系统)	CFS(cooperative file system)是一个以结构化 P2P 网络——Chord 作为其分布式散列表的 P2P 数据存取系统，不过它比 Chord 多了很多新的机制，如文件分块。
	PAST	PAST 是一个广域的 P2P 归档存储系统，它以结构化 P2P 网络——Pastry 作为底层架构，目的是提供 Internet 上安全、高可用、持久性的数据存取服务。
	OceanStore	OceanStore 基于 Tapestry，其目标是提供全球范围的广域、持久性数据存取服务，它使用了多种机制提高系统性能，如层次化 ID、数据分片冗余存储、一致性更新和内省优化等。
	Granary	Granary(谷仓)是清华大学高性能计算研究所开发的广域存储服务系统，它以对象格式存储数据，既可以基于 Grid 环境开发，也可以基于 P2P 环境开发。Granary 设计了专门的结点信息收集算法以及结构化覆盖网络路由协议。
分布式计算	GPU(Gnutella 全球处理单元)	GPU 的基本思想是在 P2P 网(Gnutella 网)上共享 CPU 计算能力。相比过去的分布式计算系统，GPU 的计算任务分配发生在 Peer 之间，而不是由一个服务器集中分配。
	SETI@ Home (Search for Extraterrestrial Intelligence @ Home)	SETI@Home(在家里搜索外星智能)，是由美国著名高校 UC Berkeley 所建立的一项旨在利用连入 Internet 的成千上万台计算机的闲置计算能力搜索外星文明的实验性分布式计算系统。每个参加者下载客户端安装，此软件以屏幕保护程序的方式运行，对来自阿莱伯克射电望远镜采集的信号工作单元进行计算处理，再将结果返回给 UC Berkeley。
	Distributed. net	这个创立于 1997 年的组织已经壮大到了全世界成千上万的用户，分布式计算的力量到了相当于 16 万台奔腾 266MHz 的计算机不间断运行的水平。
	Entropia	通过使用其成员计算机的空闲处理器时间，Entropia 向其客户提供“透明的、动态的、从一个到千万个处理器的可扩展性，包括实时资源类型和位置的重配置、处理器容错性和安全的网络数据通信。”Entropia 与 SETI@Home 功能上很像，但它是赢利性的。
P2P 搜索引擎	Pandango	美国的新兴搜索引擎设计公司 i5 Digital 在 2002 年已正式推出其依据 P2P 理念开发的商业性搜索引擎 Pandango，不过并没有进入主流的搜索引擎阵容。

续表

P2P应用领域	P2P应用体系/应用软件		简　介
其他应用	TinyP2P（微型P2P）		TinyP2P毫无疑问是世界上最小的P2P应用体系/软件，它是用15行Python代码编写的！TinyP2P创建它自己的私有和密码保护网络，不过实际上应该没人会用它。TinyP2P的意义只在于告诉人们要写一个P2P软件并不困难。
	Peer2Mail		如果你有大的Web邮箱储存空间，就能够使用P2M软件在它里面储存文件。P2M分割你要存储的文件，可选地加密它，一旦上传了所有文件段，你能够下载它们且合并这些段得到原始文件。
	Hamachi		Hamachi为多台计算机提供一个安全的专有P2P网络。它可以连接任何两台联入Internet的计算机，连接是直接的，并且可以绕过防火墙、路由器甚至NATs。
	Aimini		Aimini是一个集成了诸多“小玩意”的P2P应用体系，如直接连接、FTP下载、VoIP、网络聊天、货物交换、用户查询、地址名册等。或许正是功能太多，导致Aimini从未被广泛使用过。
	Dijjer		Dijjer的发明人是“BT之父”——Bram Cohen，它与BT非常像，只是不需要torrent(种子)文件，而只需要在规定的Web站点放上文件链接。实际上Dijjer更多地被当作Web插件来用，比如FireFox浏览器中集成了Dijjer插件，从而在浏览器中就可以P2P的方式下载文件，而用户感觉不到。
	迅雷		“迅雷”是一款基于P2P技术的多源下载软件。号称“宽带时期的下载工具”，迅雷针对宽带用户做了特别的优化，能够充分利用宽带上网的特点高速下载文件。与Dijjer相似，迅雷主要被用作Web插件集成到Web浏览器中。
	P2PBazaar(P2P集市)		顾名思义，“P2P集市”提供了一个P2P方式的电子市场，在这里你可以浏览、搜索物品，而交易通过互发E-mail进行。“P2P集市”目前还不支持信用卡，不过最大的问题是没多少人用它。
	JXTA		Sun公司JXTA项目的目的在于提供一个开放、通用、互操作的P2P开发平台。JXTA的核心处使用XML，这是它独立于语言和操作系统的重要原因。JXTA封装了P2P网络底层，对用户而言只要使用其应用接口就可以进行P2P编程。
	结构化P2P网络的部分代表性应用	Bayeux	UC Berkeley开发的P2P多播应用，基于Tapestry覆盖网提供高效、容错的应用层多播。
		SCRIBE	Microsoft Research开发的通用、可扩展的组通信和事件发布系统，提供应用层多播和任播，它基于Pastry覆盖网。
		SQUIRREL	Microsoft Research开发的分布式协同Web缓存，使得用户Web浏览器之间能共享缓存，它也是基于Pastry覆盖网的。

在这份“P2P 应用清单”里，你可以看到太多的有相似功能的 P2P 软件，实际上其中很多非常优秀(包括不少国产软件)。但它们中很少能达到像 BT、eDonkey 这样的流行，其原因并不全在软件本身，而是源于这样一条“先入为主、因果互导”的 P2P 软件准则——“用户越多，则越流行；越流行，则越好用；而越好用，则用户越多。”

5.2 文件共享

文件共享是 P2P 应用的重点和主流，在本章开头的“P2P 应用清单”里列举了世界范围内绝大多数有影响力、代表性的 P2P 文件共享体系/软件，这里不可能对其中每一个都具体介绍。下文将详细讲述两种最流行的 P2P 应用软件——BitComet 和 eMule 的使用方法。

5.2.1 BitTorrent 的使用

BT 在中国的名气，比 P2P 还要大，实际上很多人是先听说 BT，后知道 P2P 的(包括作者本人)。中国互联网络信息中心(CNNIC)2006 年 1 月 17 日发布的第 17 次中国互联网发展状况统计报告显示：中国内地网民总人数为 1.11 亿，而截至 2005 年末，中国网民中有 27.8%的人使用过 BT 软件，总规模约为 3085.8 万人，如图 5.2.1 所示。

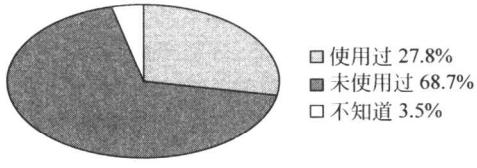


图 5.2.1 中国内地 1.11 亿网民 BT 软件使用比例

本书中已多次谈及 BT，第 2 章专设一节“BitTorrent——分片优化的新一代混合式 P2P 网络”详细介绍了 BT 的历史、设计者、搜索网站、体系原理、分片机制、阻塞算法和性能分析，读者可以看到客观上 BT 的成功和主观上作者对 BT 的推崇。历史、理论不再多说，下面的部分只讲 BT 软件具体的使用。今天，支持 BT 体系的软件琳琅满目，功能上都大同小异，作者最经常使用的是 BitComet、BitSpirit 和 BitTorrent。下面以 BitComet 0.61 版为例，讲述 BT 的使用步骤，这里会讲得很详细，因为其中不少步骤对于其他软件的使用也是适合的。

STEP1：下载并安装 BitComet 0.61 版

可以在很多 Web 站点下载到 BitComet 0.61 版软件，比如到号称“P2P 行业

第一中文门户站"的"P2P 中国"网站如下链接 http://www.ppcn.net/n2323c16.aspx，可以看到"BitComet V0.61 简体中文稳定版"的文字说明，在文字下面的链接 http://download.ppcn.net/p2psoft/bittorrent/BitComet_0.61.exe 上击右键"目标另存为..."，选定安装程序保存路径即可开始下载。文件大小近 3MB，属于很小的软件。

下载完成后双击 BitComet_0.61.exe 图标，安装过程开始。首先选择安装语言，然后会有一个"许可证协议"界面，如果接受协议条款则点击"我接受"按钮；之后选择要安装的组件，如图 5.2.2 所示。

图 5.2.2　选择要安装的组件

除了必选的 BitComet 主程序，下面四项都属个人爱好，不影响 BitComet 使用性能。我们通常选择"关联.torrent 文件"，这样以后使用起来很方便。

下一步选择 BitComet 软件的安装文件夹，虽然默认安装路径为"C:\Program Files\BitComet"（这也是绝大多数软件的默认安装路径），但强烈建议修改安装路径不要放在 C 盘，因为网上下载的文件有可能携带病毒，放在 C 盘与诸多系统文件在一起最不安全。

STEP2：配置 BitComet 0.61 版选项（可选）

如果你的计算机不通过代理上网，这一步通常可以不做，因为 BitComet 0.61 版选项的默认设置基本上合适。不过还是强烈建议执行这一步骤，了解已有选项的设置情况，如图 5.2.3 所示。

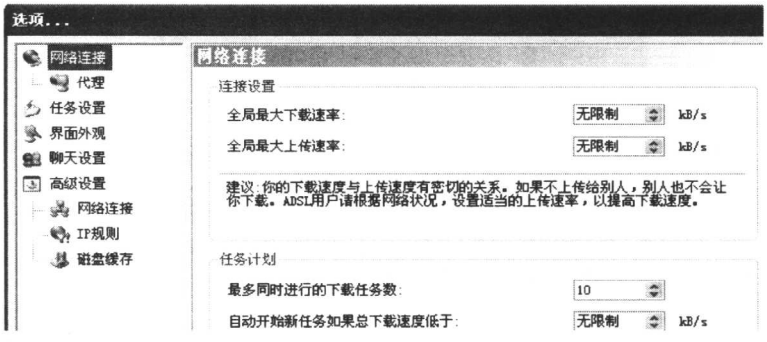


图 5.2.3　配置 BitComet 0.61 版选项

这些选项中，最重要的是“网络连接”下的“全局最大下载速率/全局最大上传速率”。如果带宽有限，可以适当降低上传速率，但不宜太低。从用户自己的角度讲，上传速率太小的用户不能获得高下载速率，这是 BT 系统所设定的；从整个系统的角度讲，只有每个 BT 用户都贡献自己的一部分带宽来上传，整个 BT 网络才能更好地工作。

一个重要的设置是“高级设置”下的“磁盘缓存”。磁盘缓存的目的是使用内存以避免磁盘因频繁读写而损坏。默认情况下磁盘缓存在最小值 6MB～最大值 50MB 之间动态调整，这个值很合理，建议不要修改，因为改小了可能使磁盘负担过重，改大了多占内存而下载效果也不会明显改善。

STEP3：设置代理选项

很多计算机是通过代理服务器上网的（通常学校、公司、政府组织都是这样做的），它相当于一个中转站，你的计算机发出的一切消息都经过它的处理发到因特网上，而你的计算机收到的一切消息都是代理服务器先从因特网上收到消息后经处理转发给你的。代理协议有多种类型，最常见的是 HTTP 代理、Socks 代理。BitComet 软件既支持 Socks 代理（Socks4、Socks4a、Socks5 三种），也支持 HTTP 代理（HTTP1.1），如图 5.2.4 所示。

图 5.2.4 代理服务器设置

BitComet 对 Socks 代理支持较好，但对 HTTP 代理支持得不好，常常不可用，因此如果你的代理服务器属于 HTTP 代理类型，而且在 BitComet 中填了此代理后总是不能下载，就需要通过某个代理协议转换软件将 HTTP 代理转成“虚拟”的 Socks 代理使用。要做这样的转换，首先从网上下载一个代理协议转换软件，这种软件很多，以作者常用的 CCProxy 为例，下载和安装不再详述，只讲其设置，如图 5.2.5 所示。

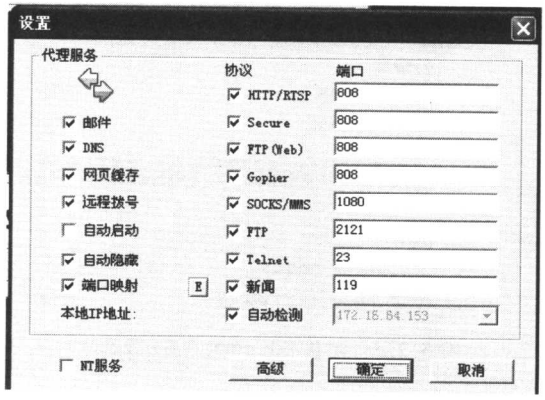


图 5.2.5 设置界面

在图 5.2.5 的设置界面中，建议不要更改协议端口(注意 Socks 代理协议端口为 1080，这个下面有用)，点击“高级”按钮，显示如图 5.2.6 所示界面。

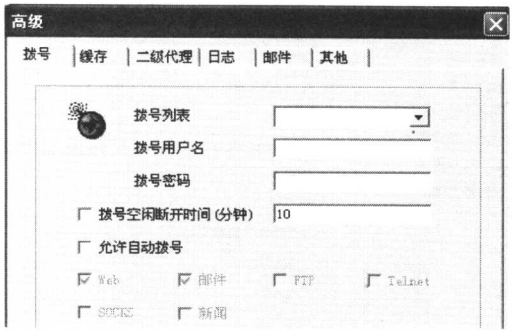


图 5.2.6 “高级”界面

图 5.2.6 所示“高级”界面里，需要填的只是“二级代理”(其实就是代理服务器)，如图 5.2.7 所示。

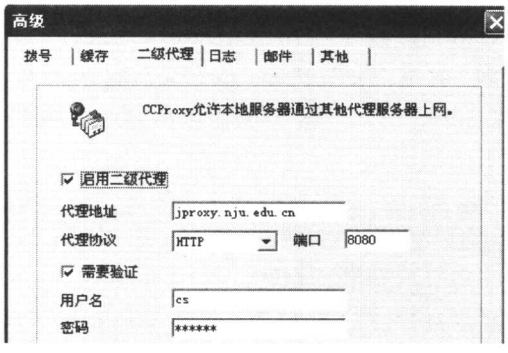


图 5.2.7 “二级代理”界面

在二级代理界面中填入代理服务器的地址、协议、端口，如果需要的话还要填入用户名、密码。一个需要提醒的问题是，这个界面中的“二级代理”实际上填的就是代理服务器的信息，请读者注意这一点。

CCProxy 配置完成后运行，按图 5.2.8 所示填写 BitComet 的“代理”设置(其中“服务器 127.0.0.1”就是指本机，“端口 1080”正是 SOCKS 协议端口，通过这样的转换，BitComet 将 BT 消息以 SOCKS 协议格式发送到本机的 1080 端口，再由 CCProxy 将 SOCKS 协议格式的消息转换成 HTTP 协议格式的消息发送给 HTTP 类型的代理服务器)，就完成了 STEP3 代理选项设置，BitComet 一般可以正常下载。

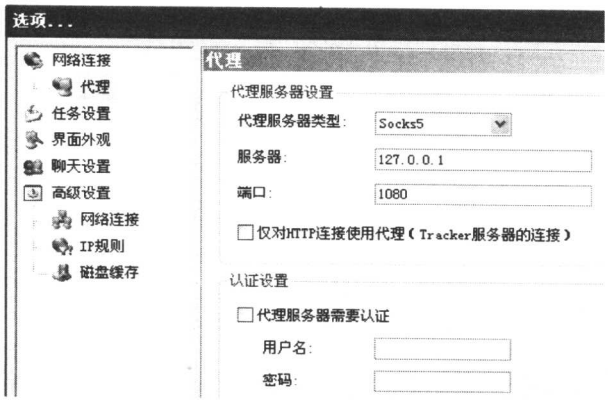


图 5.2.8 “代理”设置

STEP4：到 BT 网站搜索文件，下载种子(.torrent 文件)，设置任务属性

因为 BT 软件本身不提供文件搜索功能，所以必须到某个 BT 网站搜索文件。Web 上的 BT 网站很多，推荐 BT@China 联盟：http://www.btchina.net，这里的文件搜索结果相对比较全。在网站搜索栏里输入文件关键字，通常选择按照“发布内容”搜索，单击“开始搜索”按钮，一般几秒后显示搜索到的种子(.torrent 文件)列表。图 5.2.9 是 BT@China 搜索“周润发 上海滩”的结果(2006 年 3 月 17 日)。

日期	类别	名称 (安装BT后点击下载torrent文件)	大小	种子	下载	完成	发布者
01-21 22:23	内地连续剧	【画影论坛@BT发布】【国产】【刀锋1937】【国语】【VCD-RMVB】[相关讨论] [相同]	3577MB	94	485	999+	3487320@CN5566
09-22 10:08	电影	[推荐]-> 【BT影视天堂原创】【周润发电影32部】【RMVB/11.81G/国语/粤语】[相关讨论] [相同]	12095MB	31	364	999+	bn524@BTPIG&BTGOD
02-09 19:48	港台连续剧	超级经典，周润发的上海滩全25集RMVB，非常清晰，大家一起加速吧 [讨论] [相同]	4142MB	18	146	999+	黑暗中独行
06-19 19:50	港台连续剧	回顾经典 周润发 上海滩 [相关讨论] [相同]	4142MB	18	146	999+	letmedie@CNXP
09-14 12:57	港台连续剧	[推荐]-> 周润发上海滩.rmvb(国语+字幕)+MTV+正版VCD封面+MP3(叶丽仪+周润发现场版) [讨论] [相同]	1919MB	6	33	999+	yang211
09-14 18:05	港台连续剧	周润发上海滩.rmvb(国语+字幕)+MTV+正版VCD封面+MP3(叶丽仪+周润发现场版) [讨论] [相同]	1919MB	6	33	999+	yang211
09-15 01:46	港台连续剧	==正版== 上海滩 - 周润发 [讨论] [相同]	1919MB	6	33	999+	uoa
09-15 04:14	港台连续剧	==正版==>> 上海滩 << ==周润发== [讨论] [相同]	1919MB	6	33	999+	uoa
09-15 15:07	港台连续剧	==正版=>>上海滩<<=周润发~= [讨论] [相同]	1919MB	6	33	999+	uoa

图 5.2.9 搜索到的种子文件例

图 5.2.9 中，最重要的是文件“种子”数的多少：种子数越多通常下载越快并且文件能完全下载，种子数为 0 则文件不能完全下载。其次看文件“下载”用户数的多少：下载用户越多，通常下载越快，这正是 P2P 文件共享的特点所在。

从种子列表中选择最合适的(通常是名称符合且种子、用户数排在最前的)种子，点击种子名称下载种子(有些 BT 网站为了防止恶意下载种子造成服务器负担

过重，在下载种子之前可能会要求输入网页临时生成的验证码）。

如果 STEP1 安装 BitComet 时选择了关联. torrent 文件，则一旦种子下载完成即自动运行 BitComet 并进入“任务属性”界面，设置该文件下载任务的属性：文件保存位置、文件选择、高级设置、任务链接。其中最重要的是文件选择，因为一个种子所关联的多个文件常常并不都是你想要的，有些是广告甚至病毒。

文件选择完毕，一般就可以开始下载任务。如果要针对该任务做进一步设置，在“高级设置”界面中，可以选择“允许使用公用 DHT 网络”（这是从结构化 P2P 网络中借鉴过来的 DHT 网络技术，但就目前而言，BT 对公用 DHT 网络的使用是甚为低效的）、“允许用户来源交换”，还可单独设置任务参数：最小保证上传值、最大允许上传值。

STEP5：任务进行和完成

图 5.2.10 是 BitComet 0.61 版下载任务界面，在右上的任务队列中显示了所有进行中的任务（下载或者上传），而右下的任务信息中提供了 BitComet 工作时各项重要的统计信息。

图 5.2.10 中各项信息的含义如表 5.2.1 所示。

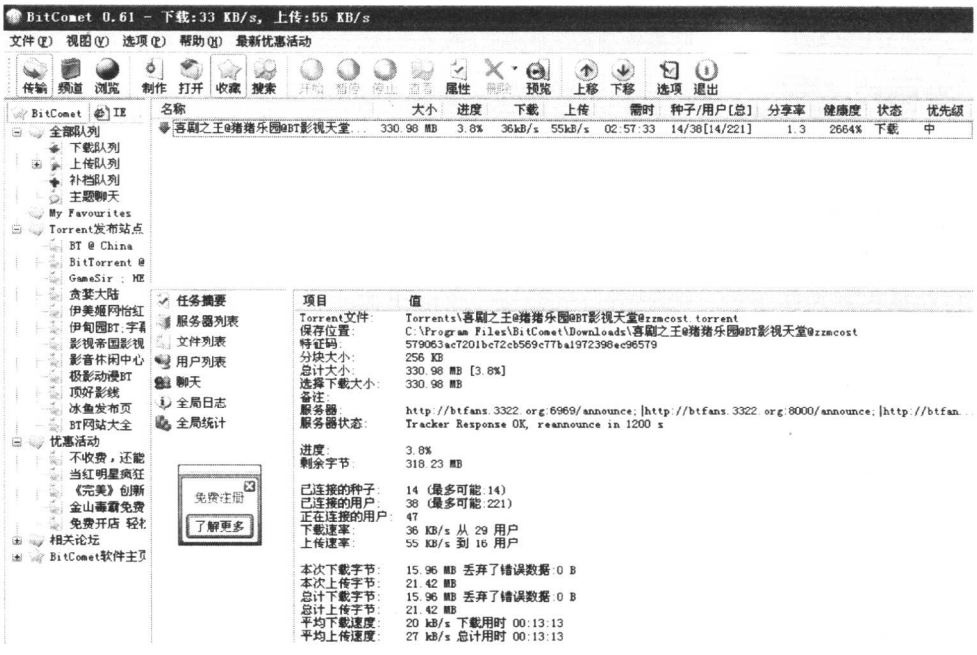


图 5.2.10 下载任务界面

表 5.2.1 图 5.2.10 中各项信息的含义

右上任务属性	名称	下载(或上传)文件名称
	大小	文件大小
	进度	文件已下载多少百分比
	下载	文件下载速度
	上传	文件上传给其他用户的速度
	需时	文件下载共要多长时间，根据当前下载速率而定
	种子/用户[总]	已连接到的种子/用户数[该文件总共的种子/用户数]
	分享率	表示自己的分享程度，数字越大表示自己的贡献越大、人品越好。以下载而言，分享率＝总上传量/总下载量
	健康度	种子数越多，健康度越高，只有健康度＞100%文件才能完全下载
	状态	任务当前所处的状态：下载、上传、暂停、停止、连接服务器、连接客户端
	优先级	分高、中、低三级，反映了用户对该任务的重视程度
右下统计信息	任务摘要	一个任务最重要的各项统计信息
	服务器列表	文件 Tracker 服务器列表
	文件列表	任务文件列表，反映了 STEP4 中“文件选择”的结果
	用户列表	连接到的所有用户列表，包含每个用户的详细信息
	聊天	可以与其他 BT 用户聊天
	全局日志	一些任务日志，通常不会用到
	全局统计	BitComet 运行的全局统计信息，是重要的历史记录

到这里，大致讲完了使用 BitComet 的主要步骤、方法，其中包括一些相关技术如穿越代理和防火墙。实际上，BitComet 使用中还有许多未谈及的方面，比如文件的发布、种子的制作等，这些留给读者自己去摸索。

BT 网上共享最多的是影视文件，音乐文件数量也不少，游戏、软件的共享情况比较稳定。就下载速度而言，BT 在种子、下载数较多时，加速过程很快，达到稳定后下载速率非常高，常常只受限于物理带宽。

5.2.2 eDonkey 的使用

eDonkey(电驴)体系的特征在于将网络结点组织成两层：服务器层和客户层(注意这里的“服务器”、“客户”与 C/S 模式中的含义不同)，客户连接到服务器，而真正的网络信息交换(客户信息、文件索引)发生在服务器之间。eDonkey 将文件分块，分块又分成片段，片段进一步分成小块，从而提供了类似 BitTorrent 的多源文件下载机制，并且对文件的完整性检查也有了更细的粒度。

eDonkey与BitTorrent有着本质的不同：eDonkey属于双层无结构P2P网，虽然用户要连接到服务器，但谁连接到谁是任意的、可变的；BitTorrent属于混合式P2P网，用户要下载某个文件必须连接到固定的Tracker；另一方面从软件使用上来说，eDonkey自带搜索工具，搜索是完全分布式的，共享资源不受任何外来的集中式限制(但这使得非法内容的文件得以广泛传播)，而BitTorrent必须借助BT搜索网站来集中式地完成，外界可以通过审核BT网站中的种子(.torrent文件)来限制共享资源的内容。

eMule(电骡)这个名称表明了它和eDonkey的关系——后继但更出色，它的开发源于2002年5月Merkur对当时的eDonkey2000客户端软件的不满，其目标是保留eDonkey的优点和精华，加入新的功能以及优化图形界面。出乎意料的是，不久以后eMule成了一个著名的P2P文件共享客户端，其流行性甚至远超过它的前驱eDonkey。在今天，eDonkey、eMule所传输的网络通信量仅次于BitTorrent，显示了eDonkey类P2P体系良好的特性。

除了最著名的eDonkey、eMule，还有很多软件支持eDonkey网，如aMule、xMule、MobileMule、Iphant等。本书第3章专设一节“eDonkey/eMule——分块下载的双层无结构P2P网络”详细讲述了eDonkey的历史、工作原理、文件分块和性能分析，而下文仅以eMule 0.46c版软件为例，讲述eDonkey网络的使用。

STEP1：下载并安装eMule 0.46c版

可以在很多地方下载到eMule 0.46c版软件，如eMule中文网站http://www.emule.org.cn/，或者P2P中国网站eMule专版http://emule.ppcn.net/。

STEP2：配置eMule 0.46c版

第一次运行eMule需要配置一些信息。首先是用户名，用以在eDonkey网络中标识你自己；其次是端口设置(如图5.2.11所示)，通常采用默认的TCP：4662、UDP：4672就可以，其中TCP端口必须打开以确保eMule的主要功能，UDP端口被Kad无服务器网络使用(见4.5.2节，Kademlia结构化P2P网络)，可以关闭。

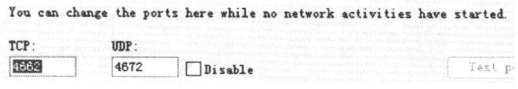


图5.2.11 端口设置

端口设置后是eMule下载/上传优先权设置，两个选项都是为了优化非热门文件(rare files)的传输。然后选择上传时是否整块传输(full chunk transfer)，选择使用哪些网络：eD2K(eDonkey2000网络)还是Kad(Kademlia无服务器网络)。

在最后一个界面，设置你的操作系统类型、同时下载数，并告诉 eDonkey 网络你的连接类型（网络连接设备、下载/上传带宽），连接类型对于 eD2K 网配置你的计算机有重要意义，所以最好别填“Unknown”。

在完成上述设置后，如果你的计算机有防火墙，或者说属于公司、组织内网用户（在 eD2K 网中，内网用户的 eMule 具有 LowID，而公网用户具有 HighID，LowID 用户之间不能直接建立连接），还需要设置代理，这些在 5.2.1 节 BitComet 使用步骤 STEP3 中已经详细讲过，这里不再赘述。

最后，eMule 还有一项非常重要的“目录”选项，在此界面中设置“下载文件”路径、“临时文件”路径、“共享目录”。其中下载文件夹保存已完全下载的文件，临时文件夹用来保存处于下载中但尚未完成的文件，“共享目录”是你提供给 eDonkey 网络其他用户可以访问到的，所以务必小心设置，不要暴露隐私和机密文件，如图 5.2.12 所示。

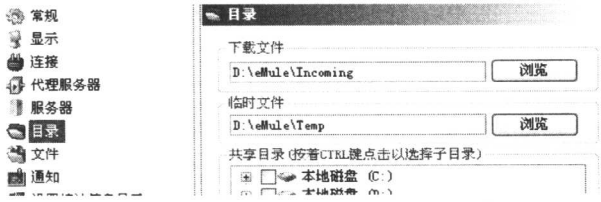


图 5.2.12 “目录”选项

STEP3：连接服务器

使用 eMule 的第一步是连接到服务器。第一次打开 eMule 时，“服务器”界面会显示十来个静态的“入口服务器”，它们是 eD2K 网最高层、最核心、通常永久在线的服务器。这些“入口服务器”中的一些（通常是 Ping 值最小，也就是离用户最近的著名服务器）会将用户连接到很多“普通服务器”（下文简称“服务器”）。这些服务器才是用户 eMule 工作时真正依赖的服务器，并且服务器列表是动态更新的。图 5.2.13 是一张典型的 eD2K 网服务器列表（注意最上两个是“入口服务器”）。

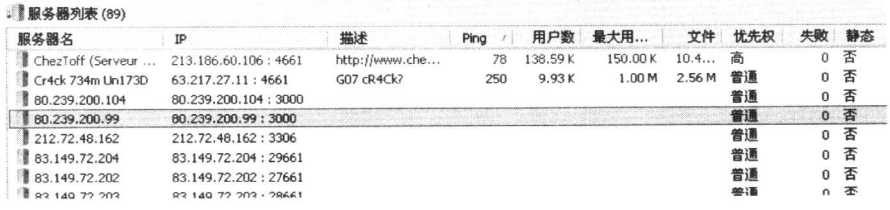


图 5.2.13 eD2K 网服务器列表例

STEP4：搜索并下载文件

连接服务器后，就可以正常使用 eMule 了。虽然可以在 Web 网站直接点击 eMule 文件链接下载文件，但绝大多数情况下文件是先搜索后下载的（这才是 eMule 的特长所在）。只要在 eMule“搜索”界面中输入文件关键字，几秒后 eMule 即返回搜索到的文件列表，可见 eMule 搜索之简单易用。从操作上讲，eMule 搜索比 BT 搜索要简单得多，但其搜索结果一般远不如 BT 理想，并且 eMule 搜索结果依赖于用户所连接到的服务器，服务器不同搜索结果往往不同。需要注意的是，eMule 搜索结果中“可用源数”类似 BT 的“下载”用户数，但“完成来源”指的是“可用源数”中拥有完整文件的用户百分比，与 BT 的“健康度”含义完全不同。图 5.2.14 是写作此书时 eMule 对于电影“无极”的搜索结果。

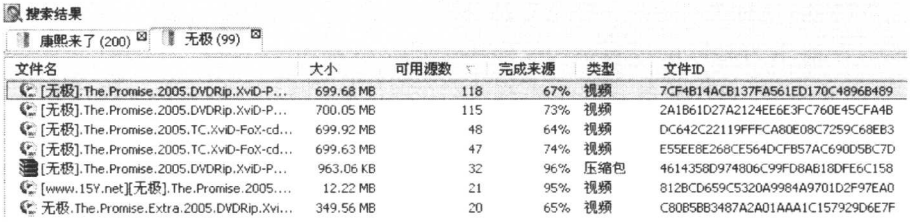


图 5.2.14 搜索结果例

在搜索到的文件列表中点击右键选择“下载”，eMule 即开始下载文件。图 5.2.15 所示是 eMule 文件传输界面，其中有 9 个下载任务在同时执行，注意“进程”条的颜色：黑色表示文件的这个部分已经有了；红色表示所有已知的来源都缺少这个部分；不同变化的蓝色表示这个部分来源的可得性，深蓝表示高可得；黄色表示已下载中的部分；显示在上面的绿色条表示下载的进度总计；当进程条全绿时，文件下载完成。

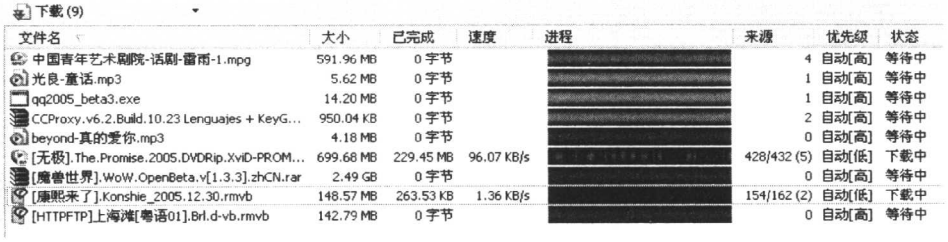


图 5.2.15 文件传输界面

双击某个下载任务可以看到该文件所有下载源的情况，这里最重要的属性是队列“优先级”QR，表示你在某个下载源那里排队的次序，QR 越大，排在你前面的

人越多，你需要等待的时间就越长，也就是说能在此下载源获得数据的概率越小，如图 5.2.16 所示。

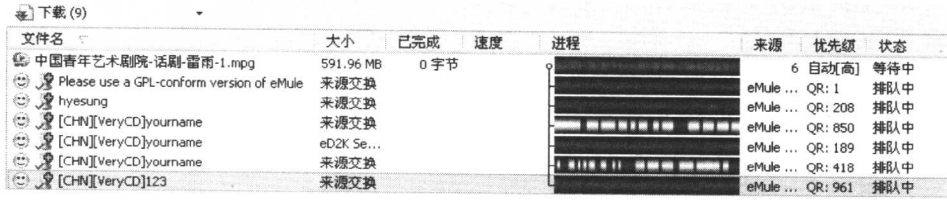


图 5.2.16 文件下载源情况例

到这里，大致讲完了 eMule 最基本的使用步骤。实际上 eMule 还有很多其他方面的功能，比如文件共享、发送消息、IRC(Internet 在线聊天)功能等，这些就留给读者在使用 eMule 的过程中去慢慢体会吧。

eDonkey 网上共享最多的是影视文件，并且热门影视的可用源数比较多，非热门影视可用源数很少，常常只有一两个，这与无结构网络的洪泛法查询有直接联系。在影视文件之后，音乐、软件的共享情况比较稳定。除此之外，使用 eDonkey 网下载文件加速过程非常慢，通常需要很多小时才能有稳定流量，并且稳定流量比起 BT 来说也要小很多(开玩笑讲，所谓“电驴”、“电骡”，可以这么理解：驴子和骡子都是慢性子的动物，所以用电驴、电骡下载文件也要有慢性子、急不得)。

5.2.3 百宝——优秀的国产 P2P 音乐共享软件

“百宝”的名字来源于小叮当(机器猫)的百宝袋，希望使用了百宝就可以像是用小叮当的百宝袋，需要什么就能够拿出什么。实际上，百宝是一款国产 P2P 文件共享软件，标题中之所以将百宝说成“优秀的国产 P2P 音乐共享软件”，是因为我们在实际注册、登录、运行、测试百宝之后，发现百宝对于音乐文件(尤其是 MP3)的搜索、下载效果特别好，对于其他类型文件(如影视、软件等)则差强人意，经常一个都搜索不到。百宝软件主界面如图 5.2.17 所示。

图 5.2.17 百宝软件主界面(v2.0 Beta1 版)

可以在“P2P中国”网站的百宝专区 http://100bao.ppcn.net/下载到百宝 v2.0 Beta1 版软件。使用百宝的第一步是要注册用户，注册以后获得一个 ID 号，以后用此号及密码登录百宝。要注意的是：百宝只支持 Socks5 代理，所以如果你是使用 HTTP 代理上网的内网用户，在百宝的“代理设置”里不能直接填 HTTP 代理，而是首先运行 CCProxy 软件（这在 5.2.1 节 BitComet 使用步骤 STEP3 中已经详细讲过），然后在百宝代理设置里按图 5.2.18 所示填写。

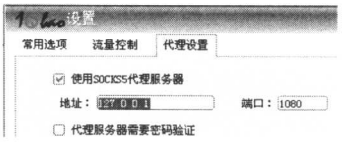


图 5.2.18 “设置”界面

用百宝搜索、下载音乐文件，尤其是 MP3，方便快捷，只要在搜索栏中输入关键字，回车即可，如图 5.2.19 所示。

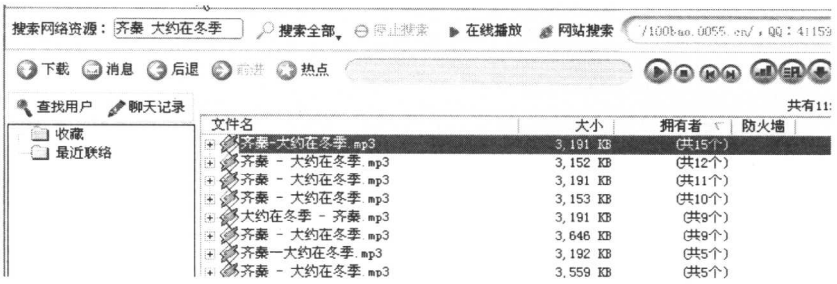


图 5.2.19 用百宝搜索例

双击文件项或者右键单击选择“开始下载”，百宝即开始下载文件。在“资源下载”界面中可以看到下载进度以及下载的源用户，如图 5.2.20 所示。

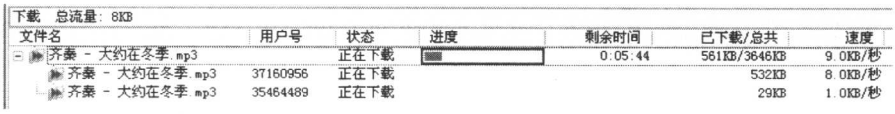


图 5.2.20 下载例

除了音乐文件共享，百宝还具有其他一些功能，但并不是百宝所长。用一句话来说：“如果你需要音乐，请用百宝！”

5.2.4 Maze 文件共享系统

Maze 最初是北京大学网络实验室的几位硕士研究生在 2003 年暑假的开发成果，我们在其主页 http://maze.pku.edu.cn/上可以看到 Maze 的开发目的是“解决 FTP 服务器的缺陷以及它所导致的在 FTP 搜索引擎内找到资源却无法有效下

载的问题，为广大网友提供一种文件共享的新方法、文件下载的新途径。”据统计，2004 年 10 月 Maze 拥有近 41 万用户、同时在线用户数超过 1 万，能搜索到超过 1.5 亿文件。

在 Maze 经典版之后，Maze 工作组又开发了 ngMaze 系统，其目标在于“扩展原 Maze 系统以支持 IPv6 用户，并利用 CNGI 上部署的弹性重叠网络（即覆盖网络）对 P2P 系统支撑与管控的中间件系统，提升 Maze 系统的性能以及考虑未来 CNGI 运营商的需求，让系统具有较好的可管理，可控制和可运营特性。”

Maze 系统的组织类似 Napster，属于典型的混合式 P2P 结构，带有集中式、基于机群的搜索引擎，提供基于关键词的查询和基于网络局部性（按照 IP 地址前缀匹配程度来衡量）的下载优化机制。此外，基于“朋友的朋友更可能成为朋友”这样的想法，Maze 将其用户按照朋友关系组织成一个“社会化网络”（social network）[Chen et al.,2004]：用户通过 Maze 客户端不仅可以看到其朋友，还能看到其朋友的朋友（如图 5.2.21 所示），这加强了 Maze 系统中的用户关联，有助于 Maze 系统向“小世界网络”的进化，同时也分担了 Maze 服务器的部分任务，有利于系统的分布性。

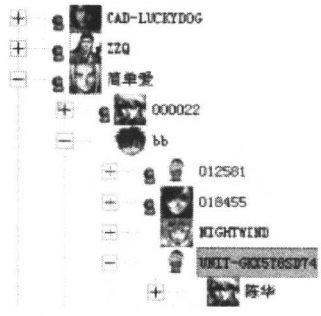


图 5.2.21

[刘翰宇等，2006]采用聚类技术分析 Maze 系统日志，将 Maze 用户以量化的方式分为 4 类：NAT 型用户、活跃型用户、客户端用户以及惰性用户，并指出占用户总数仅 0.77%的活跃型用户具有服务器特性、对文件共享系统的可用性具有重要意义。[Yang et al.,2005]详细分析了 Maze 系统中广泛存在的“搭便车用户”，并设计了一套激励机制以促进 Maze 用户分享其资源。

由于 Maze 一直是学术研究团体以非盈利性的目的开发，所以在功能上远不及商业软件完备，比如 Maze 客户端软件不支持代理，损失了大量潜在的内网用户。

5.2.5 国产 P2P 文件共享软件评点

国产 P2P 文件共享软件中不乏优秀之作，上文仅介绍了“百宝”作为代表。实际上国内一大批比较流行的 P2P 文件共享软件，如 PP 点点通、PoCo、ZCOM 智通、卡盟、酷狗、酷宝等，无论从使用方式、网络结构、界面设计上都与“百宝”类似，只是针对内容各有侧重。限于篇幅不再对这些软件详细介绍，仅对其中部分列表做一评点。

P2P 软件	PP 点点通	
主页	http://www.pp365.com	
简介	PP 点点通是国产 P2P 软件中用户最多、影响力最大的，其网站宣称已拥有累计注册用户 3000 多万，每天新增用户 8 万以上。其名字“PP 点点通”正是“点对点”(Peer-to-Peer)的含义，虽然这个翻译并不正确。	
优点	PP 点点通使用简单，界面美观，文件搜索和下载效果都还不错。此外 PP 点点通与 PoCo 已实现兼容。如果你是公网用户，推荐一用，支持国产！	
缺点	不支持代理，内网用户难以使用。(2005 Build 0112 版)	
P2P 软件	PoCo	
主页	http://www.poco.cn/	
简介	POCO 在很多方面和 PP 点点通很像，也是一款具有较大影响力的国产 P2P 软件，其网站宣称已拥有累计注册用户 1500 多万。其名字“PoCo”意为“People Connect”，反映了 PoCo 侧重于连接因特网用户的思想。	
优点	PoCo 各方面与 PP 点点通很类似。上面已说过 PP 点点通与 PoCo 已实现兼容，所以也推荐一用。	
缺点	不支持代理，内网用户难以使用。(0.4455.0.0 版)	
P2P 软件	ZCOM 智通 (P2P 网络电子杂志)	
主页	http://www.zcom.com/	
简介	ZCOM 的设计思想为“彻底颠覆平面杂志的阅读习惯，令读者的阅读方式更时尚新颖”。它收集了互联网上几乎所有的免费电子杂志。ZCOM 宣称已拥有大中华地区最大的宽带娱乐用户群(1200 万人)。	
优点	使用 ZCOM 可以获得非常多的网络电子杂志，如果您很喜欢看电子杂志，ZCOM 是首选。	
缺点	软件设计粗糙，点击很多功能按钮都自动转到 ZCOM 网站上。(3.04 版)	
P2P 软件	卡　　盟	
主页	http://www.kamun.com/	
简介	卡盟是网络科技时代集在线点播、文件共享、资源搜索和即时聊天等众多功能于一身的娱乐平台，实际上其主要功能就是一个 P2P 文件共享软件。	
优点	与大多数国产 P2P 软件类似，卡盟使用简单，界面美观，文件搜索和下载效果都还不错。此外，卡盟提供了一些方便的手机短信功能，或许值得一试。	
缺点	只支持 Socks 代理。(3.1.0.0 版)	

P2P 软件	酷宝	
主页	http://www.kubao.com/	
简介	酷宝为华语用户提供了一个适合中文环境的多媒体资源共享平台，实际上其主要功能就是一个 P2P 文件共享软件。	
优点	与大多数国产 P2P 软件类似，酷宝使用简单，界面美观，文件搜索和下载效果都还不错。	
缺点	不支持代理，内网用户难以使用。(1.0.0.8 版)	
P2P 软件	酷狗	
主页	http://www.kugoo.com/	
简介	酷狗是基于中文平台的专业 P2P 音乐及文件传输软件。	
优点	酷狗的特长在于音乐文件的搜索和下载，与“百宝”很像。	
缺点	只支持 Socks 代理。(3.202.0.0 版)	
P2P 软件	百兆 P2P 浏览器	
主页	http://www.baizhao.com/	
简介	百兆由一群热爱网络的年轻人开发，是集 P2P 音乐下载、P2P 在线音乐试听、百兆网络电台、百兆翻唱平台、浏览器、百兆域名、即时通信合为一体的国产 P2P 软件。	
优点	百兆的特长在于音乐文件的搜索和下载，此外百兆网络电台和翻唱平台对于感兴趣的用户推荐一试。	
缺点	不支持代理，内网用户难以使用。(1.0.0.0 版)	

5.3 多媒体传输

自多媒体技术产生以来，多媒体文件(音频/视频/……)就一直在寻找一个合适的网络传输载体，这源于多媒体文件特殊的要求：传输量大且要求传输速率稳定。P2P 真正适应了多媒体传输对网络带宽的巨量需求，因为所需要的巨大带宽被所有共享多媒体文件的用户分担，当用户数增加时传输质量通常是变好而非变差。这一节介绍著名的 P2P 网络语音传输工具 Skype，以及国内的两款 P2P 网络电视软件 PPLive 和 TvAnts。

5.3.1 Skype——优秀的网络语音传输工具

Skype，也称“TOM-Skype”，是 TOM 在线和 Skype Technologies S. A. 联合推出的互联网语音传输工具，当然它也能传输视频。Skype 使用 P2P 技术提供非常清晰的语音通话效果，其网络结构类似于 KaZaA，依靠“超结点”构造通信骨

干网；使用端对端的加密技术，保证通信的安全可靠。Skype 中文官方网站：http://skype.tom.com。

Skype 最吸引眼球的应用在于可以使用 SkypeOut 实现 PC2Phone 的功能（即网上拨打座机电话），价格低廉而且音效很好，属于 VoIP（Voice of IP，IP 语音电话）。Skype 网络电话服务的开通，已对全球网络、电信领域造成很大冲击。

Skype 创始人 Niklas Zennstrom（见右图）是一位著名的 P2P 创业家。2000 年他和丹麦伙伴 Friis 创办了著名的双层无结构 P2P 软件 KaZaA，后来 KaZaA 成为互联网历史上下载量最大的软件之一（下载次数多达 3.7 亿次）。在 KaZaA 之后他相继创办了 Joltid 公司（开发推广 P2P 解决方案合和 P2P 流量优化技术）和 Altnet（世界上第一个完善而安全的 P2P 网络，为消费者提供商业内容并发行数字版权管理许可证）。2003 年，Niklas 创立了闻名全球的 Skype 公司。

Skype创始人简介

在很多方面 Skype 与国内现有的即时通信工具 QQ、MSN Messenger、PoPo 很像，比如它们都需要用户注册、登录从而构建一个 Internet 网上社群，都具有添加好友、查找好友、语音聊天、文件传输等功能，界面、操作、选项设置等也非常相似。而 Skype 真正超越它们的地方，从品质上说是清晰的语音通话效果，从功能上说是特有的电话拨号、多方通话服务。下面图 5.3.1 的左图为 Skype 主界面，右图为 Skype 电话拨号界面（2.0.4.45 版）。

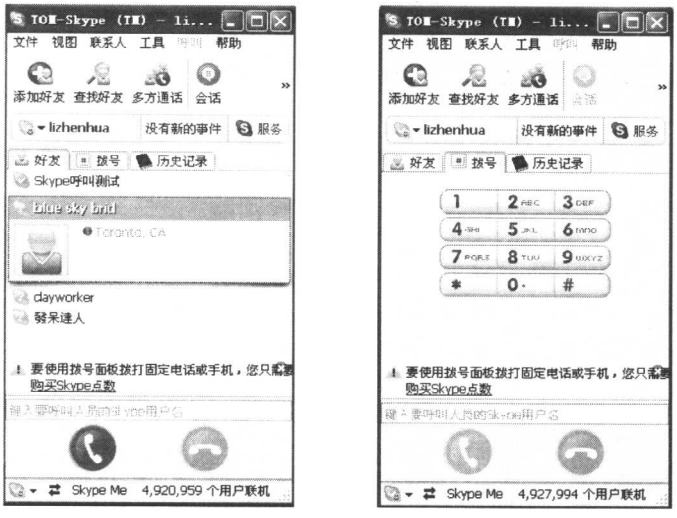


图 5.3.1　Skype 界面

对于 Skype 的使用这里不再多说，因为只要你会用 QQ、MSN Messenger 这样的软件，就肯定会用 Skype。唯一要提醒内网用户的是，虽然 Skype 宣称穿越防火墙、NAT 的能力极强，但肯定还是要设置代理的。

最后用一句话来评价 Skype："对于网络语音传输而言，Skype 是首选！"

5.3.2 PPLive——不错的国产 P2P 网络电视软件

PPLive 是一款用于互联网上大规模视频直播的 P2P 多媒体传输软件。PPLive 使用网状模型，有效解决了当前网络视频点播服务的带宽和负载有限问题，用户越多，播放越流畅，是国产 P2P 网络电视软件中最流行和最具代表性的。PPLive 官方网站：http://www.pplive.com。

写作本书时我们使用的是 PPLive 1.1.0.4 版（如图 5.3.2 所示）。从软件界面和使用上来说，PPLive 简洁、易用，各项功能一目了然，并且提供的频道列表非常之多，每个频道后给出了当前该频道的播放质量（百分比越高通常播放质量越高）。

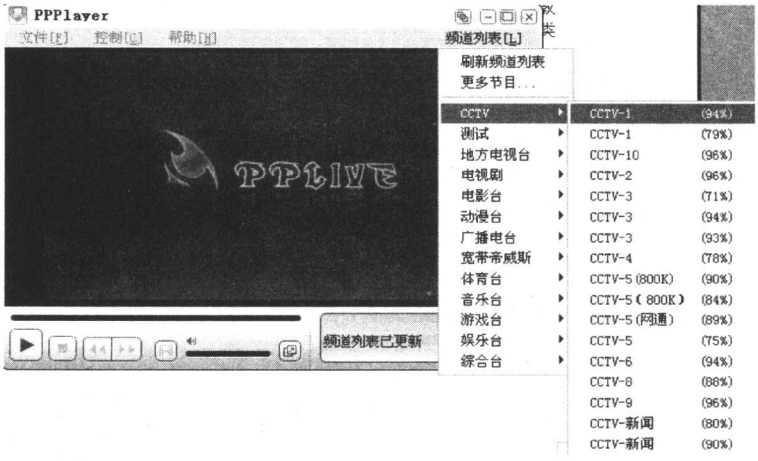


图 5.3.2 PPLive 界面

PPLive 播放的电视节目的码率一般为 500Kbps，有些频道已经开始提供 800Kbps 的码率。电视节目播放的码率如果很高，有可能会在播放的流畅度上造成问题，播放电视节目的时候会产生停顿现象。

PPLive 目前的两大问题，一是对于不同的公网用户服务质量差别很大，二是不支持代理，内网用户难以使用。

目前就国内而言，公网用户大致分为三类：①电信 ADSL，②教育网，③铁通、网通和其他各种类型专有网。由于 PPLive 现在有大量的教育网和电信 ADSL 用户，因此这两类用户目前使用情况比较好，可以流畅地播放节目；但是像铁通、网

通之类的专有网用户就有可能短暂性地播放不正常。

对内网用户而言，由于目前 PPLive 还不支持代理，所以基本上不能使用。PPLive 网站提供的帮助文档中指出：内网用户要想使用 PPLive，必须首先进行端口映射(比如使用 CCProxy)，然后开启操作系统和路由器中的 UPNP(通用即插即用)功能，并且这还建立在路由器支持 UPNP 功能且用户操作系统是 Windows XP 的前提下。

总体上来讲，PPLive 对于公网用户来说，是值得一用的 P2P 网络电视软件。

5.3.3　TvAnts——支持内网的 P2P 电视蚂蚁

诸如 PPLive、CCIPTV、CoolStreaming 这样的国产 P2P 网络电视软件，虽然已经拥有不少用户并且能提供不错的播放质量，但它们都有一个共同的缺点：只针对公网用户。实际上在中国(世界范围内也大致如此)，公司、组织内通过代理服务器上网的内网用户占相当一部分比例，并且他们所拥有的网络带宽往往要高于公网用户，这部分用户难于使用流行的国产 P2P 网络电视软件，是一个很大缺失。

TvAnts(网页主页：http://www.tvants.com)在国产 P2P 网络电视软件中远不及 PPLive 出名，用户数和电视频道也要少很多，但它对内网的支持很不错，而且既支持 Socks 代理，又支持 HTTP 代理，弥补了国产同类软件的缺失，也形成了自己在众多同类软件中的亮点，图 5.3.3 是 TvAnts 的代理设置。

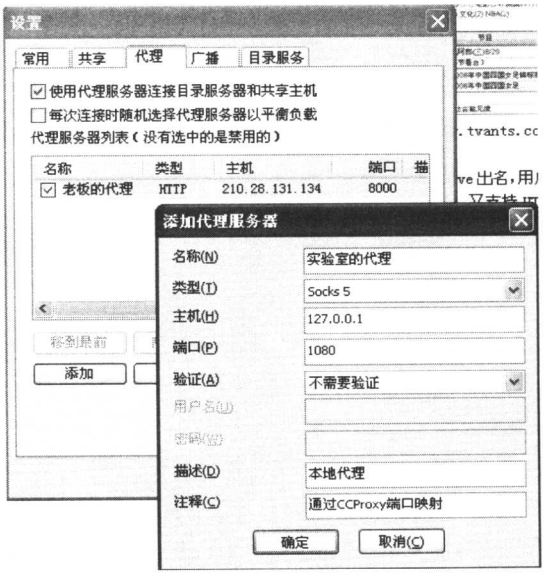


图 5.3.3　TvAnts 的代理设置

写作本书时我们用的是 TvAnts 1.0.0.55 版(如图 5.3.4 所示)，软件界面很朴素，使用方法简单、直观。第一次运行 TvAnts 需要首先搜索一次，然后在搜索结果中选择喜欢的电视频道。TvAnts 提供的频道不多，通常排在前面的在线用户数多的频道播放效果较好。TvAnts 为每个频道设有一个聊天室和一张节目单，这是一个非常人性化的设置。

图 5.3.4 TvAnts 的界面设置

点击“进入聊天室”按钮以后，电视开始播放(这样的设置不太合理，看电视似乎不必非要和别人聊天)。图 5.3.5 正是图 5.3.4 中“电视剧：再见阿郎(三)”的播放画面截图，我们是通过 HTTP 代理使用 TvAnts 的，个人感觉 TvAnts 的播放效果还不错，并且很少有停顿现象。

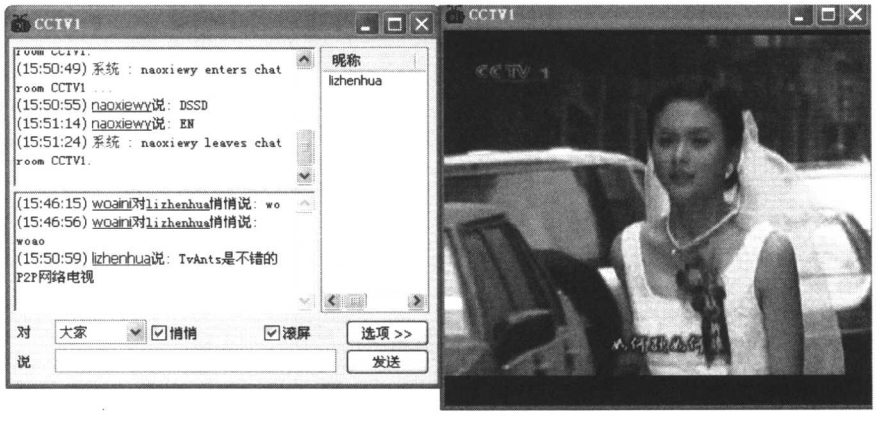


图 5.3.5 搜放画面例

5.3.4　AnySee 视频直播系统

AnySee 是由华中科技大学集群与网格计算实验室 P2P 小组于 2004 年夏天开发的一个视频直播软件，使用 P2P 应用层组播技术解决教育网内网络电视服务器难以服务众多用户的问题，使更多的用户可以观看和发布网络电视频道。AnySee 曾被用来在教育网(CERNET)内直播 2004 年的奥运会和两个国际会议 GCC'04、NPC'04，目前在国内 40 多所高校拥有超过 6 万用户。图 5.3.6 展示了 AnySee 系统超级结点的分布情况，图片来自 AnySee 主页 http://www.anysee.net/，2007 年 3 月。

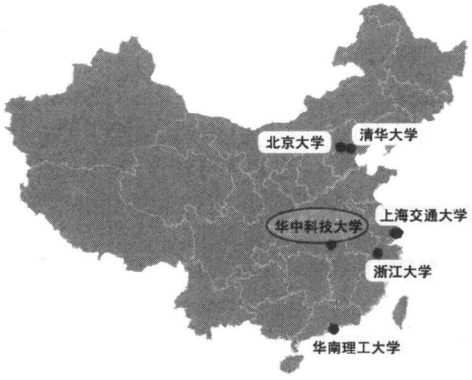


图 5.3.6　AnySee 系统超级结点的分布情况

AnySee 在实现过程中考虑了多种策略以不断改善视频服务质量：

(1) 采用基于 Landmark 的 NearCast 策略，使得系统中结点的逻辑位置尽量和实际的物理位置保持一致，以提高服务质量；

(2) 健壮的多播树维护策略[Liao et al.,2006]，使得整个系统可以适应 P2P 网络高度的动态性；

(3) 根据结点性能采用不同的服务策略，即使性能比较差的结点也可以加入系统享受服务；

(4) 特殊的 NAT 结点处理策略，尽量将 NAT 结点调整成多播树的叶结点，以克服 NAT 结点对多播树的性能影响。

5.4　实时通信和协同工作

5.4.1　P2P 实时通信软件评点

支持 P2P 模式的实时通信软件大多为网络用户所熟知，并且这些软件各方面都非常成熟，故而在此不做具体介绍，仅列表以评点之。

P2P 软件	QQ	
主页	http://www.qq.com	
简介	深圳腾讯公司开发的基于 Internet 的实时通信软件，目前支持在线聊天、视频电话、点对点断点续传文件、共享文件、网络硬盘、QQ 邮箱等多种功能，并可与移动通信终端设备相连。（对于中国的因特网用户而言，上面对 QQ 的解释或许完全是多此一举）	
优点	界面漂亮、美观，具有巨大的国内用户群。文本、语音、视频聊天各项功能都很不错。支持 HTTP、Socks 代理。	
P2P 软件	PoPo	
主页	http://popo.163.com	
简介	由北京网易公司开发的一款实时通信软件，集实时聊天、手机短信、在线娱乐等功能于一体，还拥有许多特色功能如自建聊天室、网络文件共享、穿透防火墙的超大文件传输、视频聊天、语音聊天等。	
优点	界面简洁，软件占用系统资源很少，消息传输时延短，靠新颖的手机短信服务吸引了大批国内用户。支持 HTTP、Socks 代理，穿透防火墙的能力很强，非常适合内网用户使用。	
P2P 软件	MSN Messenger	
主页	http://www.msn.com.cn/	
简介	微软公司推出的实时消息软件，它在世界范围内拥有庞大的用户群。使用 MSN Messenger 可以与他人进行文字聊天、语音对话、视频会议等即时交流，现在它还支持网络搜索功能。	
优点	界面正规、经典，在世界范围内拥有庞大的用户群，各方面功能都比较成熟。支持 HTTP、Socks 代理，不过内网用户使用效果并不尽如人意。	
P2P 软件	Google Talk	
主页	http://www.google.com/talk/intl/zh-CN/	
简介	Google 公司基于 Jabber 的 XMPP 协议开发的一个用来进行语音呼叫和发送实时消息的软件，它最大的特点是简单、易用。	
优点	界面简洁，而性能良好，秉承 Google 一贯的作风。支持 HTTP、Socks 代理。	

5.4.2 Groove 虚拟办公室——优秀的 P2P 协同工作空间

Groove Virtual Office 为美国 Groove Networks 公司的产品，该公司由著名的

Lotus Notes的开发者Ray Ozzie创立，原来的产品名称是“Groove Workspace”，意在用Groove虚拟办公室营造一个Internet协同工作空间。图5.4.1是Groove网站主页。Groove客户端软件之间不通过服务器就能直接进行数据交换，这正是使用P2P技术的Groove区别于传统“群件”（协同工作软件）的标志。对于Groove虚拟办公室而言，某个公司或者多个公司之间为了合作开发一个项目，伙伴企业的人员以及正在出差的公司职员即便不能进入公司内部网络也可以像在公司内部一样使用。用Groove网站自己的话说：“Groove Virtual Office is the fastest way to get everybody working on the same page!”（Groove虚拟办公室是让所有人在同一页面工作的最快方法！）

图5.4.1 Groove网站主页：http://www.groove.net/home/index.cfm，2006年1月

2005年3月Groove公司被软件巨人微软以1.2亿美元收购，而Groove创始人Ray Ozzie也成为微软CTO（首席技术官）。实际上早在2001年，Groove公司就曾接受过微软高达5100万美元的投资，所以2005年的收购并不值得奇怪。在微软最新推出的Office 12办公软件中，已整合了Groove软件：处于测试中的Microsoft Office 12共包含14个组件，而Groove是其中的3个组件：

（1）Groove Management Services，Beta-1 build；

（2）Groove Data Bridge，Beta-1 build；

（3）Groove Relay Server，Beta-1 build。

Groove网站上提供了一个Groove虚拟办公室的flash演示（你可以进入Groove网站并在线观看），它向我们展示了Groove虚拟办公室的三个方面用途：

①共享文件；②管理工程；③创建解决方案。实际上这三个用途合起来，正提供了一个优秀的协同工作空间。

由于 Groove 虚拟办公室这样一个 P2P 群件需要透过防火墙来交换信息，所以有的企业因担心安全问题而迟迟不愿引入。为此，Groove Virtual Office v3.0 强化了安全功能，与各种各样的防病毒软件结合起来，每进行一次文件拷贝都要对其内容进行检查。

除安全功能外，Groove Virtual Office v3.0 还新配备了可现场通知每名成员的临场控制功能、使用个人电脑的 VoIP(IP 语音电话)功能等，并对程序代码进行了修改，运行速度提高到了原来的 2.5～3 倍。

在 Groove 网站上提供了一张 Groove 软件现有客户列表，其中包含了很多著名的公司、组织(如图 5.4.2 所示)，其流行程度可见一斑。

Enterprise

APS Healthcare
BAE Systems
Beckman Coulter
Bertelsmann AG
Boeing
Capgemini
Carlson Restaurants
CH2M Hill
Chevron/Texaco
Dell Computer
Electronic Data Systems (EDS)
GlaxoSmithKline
HewlettPackard
Hitachi Data Systems
Intermec
Mattel
Microsoft Corp.
Neutrogena
Novartis
Pfizer
Praxair
PriceWaterhouseCoopers, LLP
Siemens Medical Solutions USA, Inc.
Sharp Electronics
Unilever
Verizon

Government

Defense Advanced Research Projects Agency (DARPA)
Department of Homeland Security (DHS)
General Dynamics
Office of the Coalition Provisional Authority (OCPA)
Syntek
US Department of State

图 5.4.2 Groove 现有客户列表，2006 年 1 月

5.5 分布式数据存取

P2P 的分布式数据存取本身包含文件共享的功能，但其目的与文件共享不同：它不像文件共享系统那样将传输率看成最重要的属性，而是以数据的可用性、持久性、安全性为目标，并且通常致力于广阔的领域和海量的数据。鉴于不同的目标，分布式数据存取系统采用的数据存取方法也不同，通常每个数据对象都带有自己的认证、鉴别信息，大多数系统中用户的存取都遵循严格的规则和权限来进行，为

确保数据的可用性、持久性，往往采用分片、复制、缓存等方法。

本书4.2、4.4、4.5三节已详细地讲述过最具代表性的三种P2P分布式数据存取应用——CFS、OceanStore和PAST，它们都基于这几年P2P理论界的著名结构化模型，上文也提及清华大学开发的面向对象的广域存储系统Granary。从理论上讲，分布式数据存取具有广泛的用途和良好的前景，但实际上到目前为止，即使是CFS、OceanStore和PAST这三个最经典的应用都还停留在实验室阶段。这种现象一方面源自系统本身设计的复杂性与实现的困难性，另一方面也因为P2P网络高动态性的本质属性，给系统开发带来相当大的难度。

Granary广域存储服务系统

Granary(谷仓)是清华大学高性能计算研究所开发的广域存储服务系统，项目主页为http://hpc.cs.tsinghua.edu.cn/granary/granary.html，目前正在教育网(CERNET和CERNET2)上开发和优化，尚未投入应用。在用户看来，Granary将是一个巨大的存储池，用户付费使用Granary的存储空间，进行个人信息存储，或利用Granary作为大型应用系统的后台存储。

Granary提供基于对象(object)的存储方式。对象的属性可以是浮点数(float)、字符串(string)或二进制数据块(blob)三种，对象隶属于某"类"(class)，类是对象的模式信息描述，类不含有方法(method)。客户端在Granary提供的Client-API基础上编写，具体的操作主要包括创建类、删除类、创建对象、删除对象、读写对象、基于对象的数据查询、订阅事件响应等。

Granary为其应用构思了两种不同的环境：Grid环境和P2P环境。在Grid环境下，参与结点数目较少(几千左右)，但大多是强大、稳定的服务器，此时Granary能提供高效率、高可用、高可靠的存储服务。在P2P环境下，参与结点数目较多(百万规模)，并以普通个人电脑为主，此时Granary的存储服务性能将下降很多，但系统不会因为环境原因而不可用。

Granary的分布式散列表ConDHT系统包含了数百个结点，每个结点都是比较稳定的服务器，相当于上面所说的Grid环境，并且完善了对象的复制方案以增强数据可用性。ConDHT提供三种简单而完备的接口：put(key,value,version)(存储)、get(key)(读取)、delete(key,version)(删除)，每个数据副本都有一个版本号(version)，ConDHT利用数据版本号管理数据更新。与一般DHT系统不同的是，ConDHT使用单步路由策略而非多跳路由，这一方面是基于当前系统规模并不庞大并且结点多是稳定的服务器的特点，无需多跳路由，另一方面是为了简化路由表维护，因为即使发生了路由不一致，单步策略更便于调试和修复。

5.6 分布式计算

分布式计算将巨大的计算任务分解，交给许多台计算机分别执行，再将它们计算的结果归纳和整合，从而利用了它们空闲的 CPU 计算能力——重要的网络边缘资源之一。有了对等计算之后，很多领域就不再需要昂贵的超级计算机了。

在硅谷现在有许多公司正投入对等计算的开发，如 Popular Power，Centrata，United Devices，Entropia 等，并获得了高额的风险资金。硬件巨头 Intel 也在利用对等计算技术来设计改进其 CPU，因为对等计算的发展是以 PC 机资源的有效利用为根本出发点的，所以 P2P 方式的分布式计算受到 Intel 的推崇在情理之中。就本质而言，对等计算就是网络上 CPU 资源的共享。

5.6.1 GPU——Gnutella 全球处理单元

GPU 是一个真正的 P2P 分布式计算系统，它使用经典的无结构 P2P 网络——Gnutella 来分布计算任务、共享计算能力。每个 GPU 用户创建一个计算机别名来加入 GPU 体系，通过 Gnutella 网络来提供自己的 CPU 计算能力，同时获得系统中别人的 CPU 计算能力。GPU 主页：http://gpu.sourceforge.net，其客户端界面如图 5.6.1 所示。

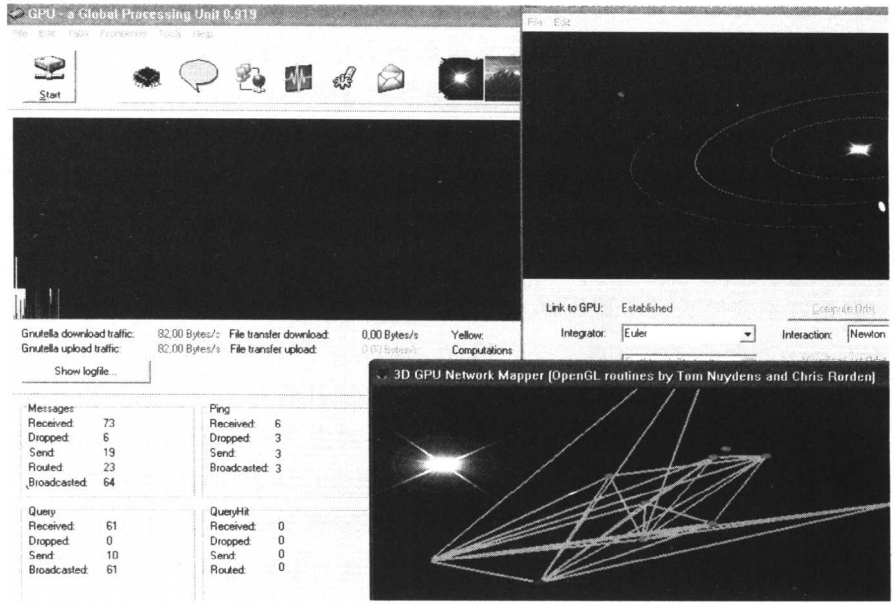


图 5.6.1 GPU 0.919 版客户端界面

GPU软件支持插件以扩展其功能，在0.919版时有三个插件可下载（现在的0.928版提供更多的插件）：

（1）景观产生器（landscape generator，或Terragen）：产生虚拟的空间景观。

（2）搜索引擎（search engine）：是一个P2P形式的“Web爬虫”。“Web爬虫”是搜索引擎经常使用的技术，比如Google就是通过“Web爬虫”来搜集网上信息的。

（3）文件分配器（file distributor）：它让用户能发送文件给其他计算机处理。

目前GPU的流行程度还远不及传统的分布式计算系统（如SETI@Home），一个任务的网络客户群成员数目通常在5～15左右。GPU当前的另一大问题是它使用起来确实很不容易，让很多用户对它复杂的使用方式望而生畏。不过有一点值得肯定的是：GPU至少以Gnutella的方式宣告了以P2P技术进行分布式计算的潮流的开始。

5.6.2 SETI@Home——分布式计算缘起

SETI@Home（Search for Extraterrestrial Intelligence at Home，在家里搜索外星智慧）是由美国著名高校UC Berkeley所建立的一项旨在利用连入Internet的成千上万台计算机的闲置计算能力搜索外星文明的实验性分布式计算系统。SETI@Home国外网站：setiathome.ssl.berkeley.edu，国内网站：http://boinc.equn.com/seti/。

自从最早发现无线电波以来，人们认为存在检测出来自宇宙其他文明社会电波信号的可能性。检测来自外人类的电波信号看起来好像是一个简单的信号处理任务，但实际上需要巨大的计算能力才能完成，主要因为：①外来信号的参数是不可知的；②对宇宙智能搜索的灵敏度极大地依赖于可用的处理能力。

在电波频谱的每一个极小部分执行大量的计算，需要比现在最快的超级计算机还强的计算能力。所幸的是，通过无线电波望远镜获取的信号数据流是一个容易分解的分布式任务，能够根据频段对数据进行分块，这些分块在本质上是相互独立的。另外，对太空一个位置的观察得到的结果和另外一个位置得到的结果是相互独立的，这就使得能够把很大的数据集分成大量的小块，每一个计算机能够比较快地分析出其中的一块，从而可以把工作分配给自愿贡献空闲CPU周期的多台计算机来处理。上面对任务可分性的解释，正是UC Berkeley的SETI@Home项目的工作基础。

在这个项目中，SETI@Home通过Arecibo望远镜观察了可见太空的大部分，SETI@home系统需要存储总共39Tb级的大量数据，需要1100盘磁带，每盘磁带

存储 15.5 个小时数据。图 5.6.2 展示了 SETI@Home 的工作原理：

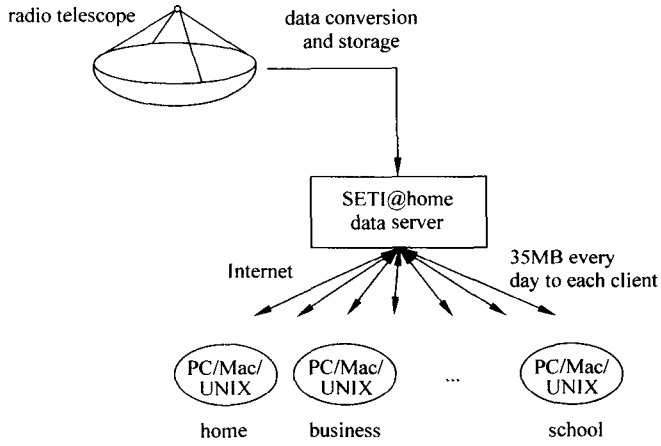


图 5.6.2 SETI@Home 工作原理示意图

（图片来自 HP 实验室 Milojicic 等人的综述性论文 Peer-to-Peer Computing，July 2003）

SETI@Home 的主服务器由三台 Sun 企业级 450 系列计算机组成。其中一台作为用户数据库，包含了几百万 SETI@Home 志愿者的信息（完成的工作量数目、上一次连接的时间，成员关系），用户数据库还包含由每一类微处理结构以及 SETI@Home 所支持每一种操作系统完成的工作量信息。

第二台服务器作为科学数据库，具有不断扩大的磁盘阵列（每一阵列是 432GB 的 RAID 0+1），它包含每一个工作单元所需处理的时间、空间坐标数据、频率数据、被 SETI@Home 用户处理的工作单元的次数以及接收的结果等。科学数据库最大的存储部分是被 SETI@Home 自愿者检测出来的可能的信号参数（比如某空间坐标处的信号能量、频率、到达时间等）。

第三台服务器包含工作单元的存储、对工作单元进行处理的分发和返回结果的存储。

SETI@Home 当前能对 47 种不同 CPU 和操作系统分发客户端软件，客户可从 SETI@Home 网站下载客户端软件。对于微软的 Windows 和苹果的 Macintosh，客户端软件安装后是作为一个屏保程序运行的，只有当该屏保程序被激活的情况下，才能处理数据。对于其他平台，客户端运行在文本方式下，这些平台的用户基本上是在后台运行该客户端程序的。

图 5.6.3 给出了 SETI@Home 的客户端软件例。

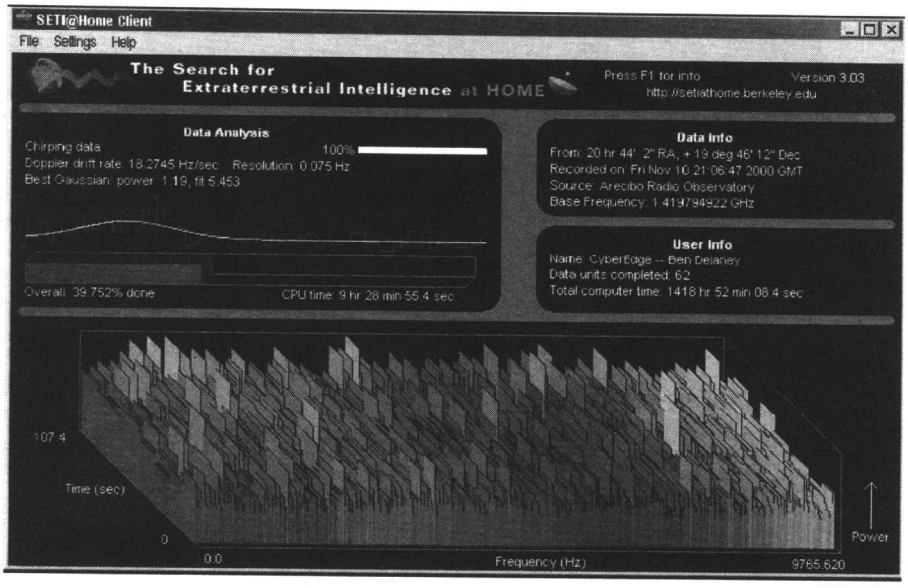


图 5.6.3 SETI@Home 客户端软件界面，软件版本 3.03

5.7 P2P 搜索引擎

前面介绍过的 P2P 软件尤其是 P2P 文件共享软件大多支持 P2P 方式的搜索引擎(如 eMule、百宝等)，但这里说的“P2P 搜索引擎”指的是能像 Google 那样包罗万象、基于 Web 网页的通用搜索引擎。P2P 技术使用户能够深度搜索文档，而且这种搜索无需通过 Web 服务器，也可以不受信息文档格式和宿主设备的限制，可达到传统目录式搜索引擎(只能搜索到 20%～30%的网络资源)无可比拟的深度(理论上将包括网络上的所有开放的信息资源)。可以说，P2P 为互联网的信息搜索提供了全新的解决之道，被很多人认为是第三代搜索引擎的开发思想。当然，目前的 P2P 搜索引擎离实际应用还有一些差距，很多都停留在理论阶段。

美国的新兴搜索引擎设计公司 i5 Digital 在 2002 年已正式推出其依据 P2P 理念开发的商业性搜索引擎 Pandango，不过并没有进入主流的搜索引擎阵容，在此不再详述。

5.8 其他应用介绍

本节介绍 P2P 的一些其他应用，虽然未将它们独立归为某类，但并不代表这些应用并非主流。它们中有些很“有趣”，有些很“宏远”，希望能对读者有一些帮助

或者启发。

5.8.1 TinyP2P——15 行代码的 P2P 软件

TinyP2P 是一个只用 15 行 Python 代码写成的 P2P 文件交换软件，但它兼有服务器与客户端两方面功能，是名副其实的“世界上最小的 P2P 应用”。TinyP2P 创建它自己的私有和密码保护网络，可以供一群人（但不能太多）交换文件，不过实际上应该没人会用它。E. W. Felten 编写此程序的目的只在于告诉人们要写一个 P2P 软件并不困难。

下面给出 TinyP2P 的源代码。不可否认这段代码有严重的“代码堆积”现象，并且由于将代码紧凑地堆到一起严重地影响了其可读性，不过毕竟 E. W. Felten 做到了每行代码不超过 80 个字符（第一行代码是程序注释不算在内）：

```
# tinyp2p.py 1.0(documentation at http://freedom-to-tinker.com/tinyp2p.html)
1  import sys,os,SimpleXMLRPCServer,xmlrpclib,re,hmac #(C) 2004,E.W. Felten
2  ar,pw,res=(sys.argv,lambda u: hmac.new(sys.argv[1],u).hexdigest(),re.search)
3  pxy,xs=(xmlrpclib.ServerProxy,SimpleXMLRPCServer.SimpleXMLRPCServer)
4  def ls(p=""): return filter(lambda n: (p=="")or res(p,n),os.listdir(os.getcwd()))
5  if ar[2]!="client": # license: http://creativecommons.org/licenses/by-nc-sa/2.0
6  myU,prs,srv=("http://"+ar[3]+":"+ar[4],ar[5:],lambda x: x.serve_forever())
7  def pr(x=[]): return([(y in prs) or prs.append(y) for y in x] or 1) and prs
8  def c(n): return((lambda f: (f.read(),f.close()))(file(n)))[0]
9  f=lambda p,n,a: (p==pw(myU))and(((n==0)and pr(a))or((n==1)and
   [ls(a)])or c(a))
10 def aug(u): return((u==myU) and pr()) or pr(pxy(u).f(pw(u),0,pr([myU])))
11 pr() and [aug(s) for s in aug(pr()[0])]
12 (lambda sv: sv.register_function(f,"f") or srv(sv))(xs((ar[3],int(ar[4]))))
13 for url in pxy(ar[3]).f(pw(ar[3]),0,[]):
14 for fn in filter(lambda n: not n in ls(),(pxy(url).f(pw(url),1,ar[4]))[0]):
15 (lambda fi: fi.write(pxy(url).f(pw(url),2,fn)) or fi.close())(file(fn,"wc"))
```

下面对 TinyP2P 程序的说明来自 E. W. Felten 本人：

“程序创建一个小规模的网络，一群朋友或商业伙伴可以用它共享文件。这个程序在非常大的网络不能很好的工作；作为替代，许多小网络能够共存。每一个网络都被密码保护；网络仅能被知道密码的人访问。（不过如果有攻击者能窃听通信，则网络是不安全的）

程序使用标准通信协议：HTTP 和 XML-RPC。HTTP 就是 Web 浏览器获取页面时使用的协议，XML-RPC 则在提供 Web 服务时广泛使用。

程序可以选择作为服务器模式或客户端模式运行。当运行在服务器模式时，程序连接到其他服务器的网络，并使当前目录下所有文件对于网络中的用

户下载有效。(当服务器运行时,存放到目录的文件对其他的用户立即变得有效)键入下面命令将使程序运行在服务器模式:

python tinyp2p. py *password server hostname portnum* [*otherurl*]

这里 password 是网络密码,hostname 和 portnum 被用来构建服务器将侦听的 URL(http: //*hostname*: *portnum*),otherurl(可选项)是其他已经作为网络一部分运行的服务器的 URL(otherurl 用来将你的服务连接到网络,如果你不提供 otherurl,程序会假设你的服务开始一个新的网络)。

键入下面命令将程序运行在客户端模式:

python tinyp2p. py *password client serverurl pattern*

这里 password 是网络密码,serverurl 是属于网络的服务器的 URL,pattern 是字符串。命令查看网络中每个服务器共享的文件,当以下两件事情为真时:(a)文件名中包含'模式'子串;(b)当前目录不存在同名文件,文件将被下载并存放在当前目录。'模式'实际上是 Python 正则式,能够被 Python 的 re. search 库用来匹配文件名。

注意,如果你将客户程序与服务器程序运行在同一个目录,客户端下载的文件将自动被服务器程序重发布。依赖于你的环境,也许如你所想,也许不。同样要注意的,你能在同一个目录运行多个属于不同网络的服务器。"

5.8.2 JXTA——开放式 P2P 开发平台

JXTA 开始于 Sun 公司在 Bill Joy 和 Mike Clary 领导下的一个研究项目,它是一组开放、通用的 P2P 协议,允许任何可联入 Internet 的设备(手机、PDA、个人电脑、服务器)基于 JXTA 平台进行互相通信和合作。

2001 年 4 月 25 日 Bill Joy 通过在线 Web 广播向公众开放了 JXTA 网站(http://www. JXTA. org/),这也标志着 Sun 公司正式向计算机领域发布 JXTA 这一开放式的 P2P 开发平台。JXTA. org 网站包含了在 JXTA 平台 Peers 之间发起通信、协作的规范、实现和架构。

JXTA 的 Peers 在 Internet 上创建了一个虚拟网络,在这个网络中,任何 Peer 可以和其他 Peer 直接通信并交流资源,即使有某些 Peers 或资源位于防火墙或 NAT 之后。JXTA 网站上明确地指出了 JXTA 项目的目标:

(1) 互操作性:JXTA 跨越不同的 P2P 系统和社区;

(2) 平台独立性:支持多种语言、系统和网络;

(3) 普适性:每个设备都可以成为一个 Peer(从原文的"数字心跳"意译而来)。

同时,JXTA 网站也构想了 JXTA 项目可能的功能:

(1) 在网络上寻找 Peers 和资源,即使有防火墙存在;

(2) 可以和网络中的任何人共享文件;

（3）跨越不同的网络创建你自己的设备 Peers 组；

（4）在公共网上同 Peers 安全通信。

JXTA 在网上已被下载超过 100 万次，这说明有非常多的程序员对 JXTA 感兴趣，其中很多也许已经在 JXTA 平台上开发出了自己的 P2P 应用程序。

对于 JXTA 的第一个目标——互操作性，值得注意的是：虽然 JXTA 平台提供互操作层，帮助开发者设计的应用之间能够互操作，但 JXTA 本身并不保证互操作性。JXTA 和 TCP/IP 在应用的时候是类同的，正如虽然 FTP 和 HTTP 都是建立在 TCP/IP 基础之上的，但你不能用 FTP 客户端去访问网页。在 JXTA 中也一样：假如有两种应用都是建立在 JXTA 之上的，并不说明它们之间一定能够具有互操作性。互操作性由开发者自己设计实现。

对于 JXTA 的第二个目标——平台独立性，不容置疑，提供统一的基础协议的第一步，就是采用一种可以在大多数平台上标示的语言形式——XML 是一种完美的选择。JXTA 的开发者意识到 XML 很快就会成为数据交换的默认标准，它提供了一种全局的、具有语言独立性和平台独立性的数据表示形式，同时 XML 也可以很容易地转变为其他编码形式，所以 JXTA 使用 XML 定义所有的协议。

对于 JXTA 的第三个目标——普适性，实际上这来源于 Sun 公司一贯的风格：对各种设备的全面支持。比如 Sun 著名的 Java 语言就提供了三个面向不同领域的部件：J2ME 面向嵌入式设备，J2SE 面向普通程序员，J2EE 则面向企业项目开发。

JXTA 涉及大量的 P2P 术语，阅读它们对于读者理解 P2P 系统很有意义，表 5.8.1给出了解释。

表 5.8.1 JXTA 术语的解释

JXTA 术语	解　　释
Peers(对等点)	任何实现一个或多个 JXTA 协议的实体。一个 Peer 可以是从大型机到移动电话，甚至是一个移动传感器的任何设备。Peer 是独立存在的，并且与其他 Peer 的通信都是异步的。
Peer Group(对等点组)	具有相同兴趣的 Peers 可以集结成 Peer Group(对等点组)。对等点组可能跨越多种物理网络。
Messages(消息)	JXTA 网络中的所有通信都通过接收和发送消息(Messages)的方式来实现。这些 Messages 叫做 JXTA 消息，这些消息具有标准的格式，这也是实现互操作性的关键。
Pipes(管道)	管道在 JXTA 环境中建立虚拟的通信通道。Peer 使用管道来发送和接收 JXTA 消息。管道是一种虚拟的概念，Peer 不需要知道它们所在的实际网络地址就可以使用管道，这是一种重要的抽象。

续表

JXTA术语	解　释
Services(服务)	Peer和Peer Group都可以提供服务。由Peer提供的服务属于个人级服务,叫做"Peer服务",这种服务的方式和集中式服务相同。其他Peer不能提供该Peer的"Peer服务"。 Peer Group提供的服务叫做"Peer Group服务"(对等组服务)。和"Peer服务"不同的是,这种服务不只是针对某一个特殊用户,而是针对这个Group中的多个用户。"Peer Group服务"更容易被使用,原因是当某一个用户不可用的时候,其他的Peer仍然会提供同样的服务。
Codats(代码/数据)	在JXTA中,这个词说明内容可以是代码或是数据。Codats可以被发布、发现,或是在必要时被取代。
Advertisements(广告)	广告用来发布和接收JXTA网络用的资源,例如Peer、Peer Group、管道或是Codats。广告是以XML文件的形式表现的。
Identifiers(标识符)	在JXTA环境中,标识符起到了非常关键的作用。JXTA使用标识符来识别资源,而不是使用物理网地址。JXTA的标识符被定义为URN(Uniform Resource Name,统一资源名)。一个URN本质上就是一个URI(Uniform Resource Identifier,统一资源标识符),这个URI在全局范围内唯一存在,并且即使资源已经不再存在,它的URI还是存在的。
World Peer Group(世界对等点组)	每一个JXTA的Peer都缺省地属于World Peer Group,并能够加入这个Peer Group。即使当Peer无法在网络上找到任何其他Peer的情况下,甚至在Peer没有连接到网络的情况下,所有的Peers也都是属于这个Peer Group的。
Net Peer Group(网络对等点组)	在一个本地网络中,网络管理员可以创建一个任何Peer都可以加入的Peer Group,这样的Peer Group就是一个Net Peer Group。这种Group类似于DHCP服务。一个Net Peer Group可以在管理员的限制下为Peer提供全局接入的服务。
Rendezvous Peer(集结点)	一个Rendezvous Peer是一个通过缓存其他Peer广告而保存有这些节点信息的特殊Peer。因此,Rendezvous Peer可以帮助其他Peers互相发现。Rendezvous Peer同样可以将收到的发现请求(discovery request)转发给其他的Rendezvous Peer。
Endpoint(终结点)	Endpoint是网络的目的地结点,它可以使用网络地址来表示。Peers一般不直接使用Endpoint,它们间接地通过Pipe来使用Endpoint,Pipe是建立在Endpoint之间的。
Routers(路由器)	将数据包在JXTA网络上传送的节点称之为JXTA路由器。并不是所有的Peer都需要成为一个路由器。任何不是路由器的Peer都需要找到一个路由器来传递它的消息。

在术语之后,我们来看JXTA定义的协议:JXTA目前定义了6个协议,这些协议可以被用作应用程序的基础,协议被设计为低开销的,并且和使用它们的应用程序所在的网络拓扑结构无关。并不要求每个JXTA Peer都实现全部的6个协

议，Peer 实现协议的数量取决于这个 Peer 的能力，一个 Peer 甚至可以只实现一个协议。Peer 也能够根据自身的需求扩展任意一个协议。表 5.8.2 说明了这 6 个协议。

表 5.8.2 JXTA 协议的解释

JXTA 协议	解　释
Peer Discovery Protocol(PDP)对等点发现协议	Peer 使用这个协议来发现被发布出来的 JXTA 资源。由于广告就代表着发布的资源，所以 PDP 的主要工作是帮助 Peer 来接收其他 Peer 的广告。作为最低级别的发现协议，PDP 提供发现的最基本机制。应用程序可以选择使用其他更高级别的发现机制。PDP 可以作为低级别的协议为其他高级别协议提供服务。
Peer Resolver Protocol(PRP)对等点解析协议	在通常情况下，Peer 向其他 Peer 发送查询消息来定位服务或者内容。PRP 会将查询的格式标准化。使用这个协议，Peer 可以发送通用的查询并接收回复。
Peer Information Protocol(PIP)对等点信息协议	PIP 可以被用来在 JXTA 环境中对一个 Peer 发出 Ping 消息。当一个 Peer 接收到一条 Ping 消息时可以有几种选择：可以选择给出一个简单的回复，回复当中仅包含 Peer 的运行时间；也可以选择发送一个完全的回复，当中包含它自身的广告信息；或者也可以选择忽略这条 Ping 消息。
Peer Membership Protocol(PMP)对等点成员协议	Peer 使用 PMP 来加入和离开 Peer Group。这个协议识别 Peer 使用的 4 个分散的步骤，并为每一步的动作定义 JXTA 消息： (1) Apply：一个 Peer 如果想加入一个 Group，可以向这个 Group 的成员验证者提出申请。 (2) Join：在申请之后，Peer 可以选择加入这个 Group。 (3) Renew：如果要更新 Group 的成员信息，Peer 发送 Renew 消息。 (4) Cancel：Peer 可以选择取消它在 Peer Group 中的成员资格。
Pipe Binding Protocol(PBP)管道绑定协议	在 JXTA 环境中，Peer 使用管道来连接服务，一个 Peer 可以动态地用绑定 Pipe 的一端来连接服务。Peer 可以新建 Pipe，把它绑定到现存 Pipe 上，或者取消对 Pipe 的绑定。在这些情况下，Peer 使用管道绑定协议。
Endpoint Routing Protocol(ERP)终结点路由协议	这个协议帮助 Peer 将消息路由至目的地。ERP 帮助 Peer 路由器查询其他 Peer 路由器，用于传递消息的有效路由。

在 JXTA 这一小节最后，我们简单地提一下 JXTA Shell。JXTA Shell 是一个建立在 JXTA 的 Java 实现上的范例应用程序，它以命令行方式为用户提供了一种交互接入到 JXTA 的方式。JXTA Shell 使用和 Unix Shell 相似的界面，并且 JXTA 的 Shell 命令和 Unix 的 Shell 命令也很相像(如 Man、Clear、Cat、Exit 等)。用户可以从 JXTA 网站的 Shell Homepage 下载到 JXTA Shell，它的安装和配置并不复杂。

第6章 P2P核心机制

“工欲善其事，必先利其器。”这一章讲述 P2P 的核心机制，也就是让一个 P2P 系统能正常运行所需要的最核心、最基础的部件和方法。它们对于一个 P2P 网络有至关重要的意义，是每个 P2P 系统的设计者、程序员都应该首先考虑的问题。

P2P 网络是一个构建在底层物理网（如 IP 网）上的覆盖网，所以第一步需要考虑的问题就是覆盖网的拓扑结构（6.1 节），拓扑结构对 P2P 网络本身的工作性能和 P2P 网络其他各方面的设计机制有决定性的作用。

确定了使用哪种拓扑结构的覆盖网以后，第二步要做的是：将物理网上的计算机映射到覆盖网，为每个物理网结点分配一个覆盖网结点。为了保持匿名性，不能将计算机的 IP 地址直接作为 P2P 覆盖网上结点的标识（nodeID）。到目前为止，主流的方法都是通过一致性安全散列函数来实现这一映射，即 nodeID=H(IP,…)，H 表示散列函数（常用 SHA 系列函数），省略号表示可能加进其他一些信息（如服务端口号）。除了结点的映射，每个数据对象也需要分配一个 objectID，即 objectID=H(objectName,…)，注意数据对象映射的作用在于将数据对象的索引信息放到 nodeID 与 objectID

最近的结点上，所有对此数据对象的定位都需要首先到此结点上找到对象索引，再从索引中得到对象实际的保存位置。上面的映射工作正是由 P2P 网络的另一基础设施——DHT（分布式散列表）来完成的（6.2 节）。鉴于散列函数的随机性，可以看出：分布式散列表在实现物理网到覆盖网映射的同时，损失了物理网上结点之间的邻近关系，也就破坏了两者的一致性。这个问题会在第 7 章增强机制中的“拓扑意识和一致性问题”中讲到。另一方面，由于对同一数据对象的定位都要首先访问 nodeID 与 objectID 最近的结点，这既给该结点造成巨大负担，也影响定位效率，所以第 7 章增强机制中提出“复制与缓存”等方法来改进。

有了覆盖网，又依靠分布式散列表建立了映射以后，就可以开始 P2P 网络最基本的功能：定位（6.3 节）。所谓“定位”，包括两种情况：①结点 A 想确定结点 B 的位置；②结点 A 想确定数据对象 O 的位置。不管哪种情况，P2P 网络定位都是通过一步一步的“路由”来完成的。路由的作用在于将消息送给离目的地更“近”的结点，这里的“近”有多个角度的衡量，因此也就有了各种各样的 P2P 路由方法。

通过路由与定位，能找到想要的结点和数据。但是这里的“想要”，含义是模糊的：大多数情况下，用户并不清楚自己确切“想要”什么，好比 Google 上人们只是打入几个关键词，希望得到一些东西。因而，P2P 网络需要“查询和搜索”机制（6.4 节）。本章中认为：“查询”的意义在于根据指定的内容来定位，可以指定结点、数据对象的确切名，也可以像 Google 那样仅指定几个关键词——这就是所谓“搜索”或称“模糊查询”。精确查询是结构化 P2P 网络的优势，但模糊查询是结构化 P2P 网络的劣势，原因就在于“结构化”三字：结构过于严格，分布式散列表又将数据对象映射得毫无规律。到目前为止，如何合理可行地实现结构化 P2P 网络的模糊查询，还是一个开放型问题。与结构化 P2P 网络相对，无结构 P2P 网络模糊查询的效果要好得多，不过受制于洪泛法中 TTL 的局部性还是始终存在的；精确查询虽然效率不高但通常能查到，但最大的问题是：无结构 P2P 网络没有办法保证找到指定的东西，也就无法确定网络中“没有”这个东西。

P2P 网络的最大特点和难点，在于它极大的动态性：不断有新结点加入、旧结点离开、结点失效等情况发生。面对这样一个动态环境，P2P 网络需要一套健全、可行的方法来处理这些情况：在新结点加入时通知其他结点新成员的到来，在旧结点离开时通知其他结点老成员的离去，而最难的是需要高效、合理地检测到结点失效并及时修复 P2P 网络。对于这三者的处理，我们统称其为“动态结点算法”（6.5 节）。

没有人是不犯错误的，年轻的 P2P 网络更易犯错。这些错误从客观上来说可能来源于结点失效或结点负担过重的“热点”问题，从主观上来说可能来源于恶意攻击所导致的安全问题、无权威认证情况下的用户信誉问题或者是 P2P 网络设计者的疏忽。对如此多错误的避免或者补救，我们统称其为“容错性”（6.6 节）。本

章中讲容错性，我们只对结点失效的情况进行研究（这也称为“自适应”），而其他诸如“热点”问题、信誉问题、安全问题等，将在第 7 章“P2P 增强机制”里分 3 节详述。

到这里，总算可以松一口气了：因为读者从上文中差不多能够想明白，设计或实现一个 P2P 网络的核心机制是什么、为什么、从何而来。虽然实现了核心机制的 P2P 网络未必能工作得很好，但最起码，现在它能正常运行起来了。

6.1 覆盖网拓扑结构

P2P 网络在应用层构建了一个有（严格）拓扑结构的覆盖网，覆盖网拓扑结构对于一个 P2P 系统具有基础性的意义，系统的其他机制如分布式散列表、路由与定位、负载均衡、自适应、自组织、容错性等，都是基于它的。到目前为止，已出现了几十种不同拓扑结构的 P2P 网络，实际上，一个 P2P 模型给人留下的最直观、深刻的印象，常常就是其几何上的拓扑结构。第 2、3、4 章已经对很多经典的 P2P 覆盖网拓扑结构做了介绍，并在第 4 章最后列表比较了它们的各项参数、优缺点。本节侧重对各种拓扑结构更深层次的分析，主要内容来源于 SIGCOMM'03 上的论文[Gummadi et al.,2003]。

1. 环和带弦环（ring and chordal ring）

环是最简单的线性拓扑，但纯粹的环结构太脆弱，也太低效，所以一般都在环上加入弦以使其容错、高效定位（即减小其直径），Chord 正是代表性的例子，它让结点 a 维护 $\log N$ 个邻居信息（fingers），其中第 i 个 finger 指向顺时针方向离 $a+2^i$ 最近的结点，从而达到在 $\log N$ 跳内成功定位的目的。然而，Chord 并不是带弦环唯一的形式，并且即使就 Chord 来说，它的第 i 个 finger 也没有必要指向顺时针方向离 $a+2^i$ 最近的结点；实际上，只要让第 i 个 finger 指向$[a+2^i, a+2^{i+1}]$区间中任意一个结点，同样能实现 $\log N$ 跳的定位。对后一种情况而言，路由表的第 i 项将有 2^i 个选择，所以可选的路由表构造将有 $\prod_{i=1}^{\log N} 2^i = 2^{1+2+\cdots+\log N} \approx 2^{\frac{\log N}{2}\log N} = N^{\frac{\log N}{2}}$ 种，提供了非常灵活的路由表构造。

如果使用上面所说的高灵活性的路由表来定位，假设结点 a、b 的 ID 相差 $O(N)$并且由 a 来发起定位，那么对路由的第一跳来说，a 的路由表中大约有 $\log N$ 个邻居可以作为下一跳，对第二跳来说，大约有$(\log N-1)$个邻居可以作为下一跳。依此类推，总共算起来，可选的路由路径共有$(\log N)\times(\log N-1)\times\cdots\times 2=(\log N)!$ 条，提供了非常灵活的路由路径选择。当然，上面的推导只是一种极端情形，实际定位时的路径选择并没有这么多。

2. 树和 Plaxton Mesh

树是一种非常流行的数据结构，它的层次化组织结构在很多场合非常适用，实际上，作为 Tapestry、Pastry 网络基础的 Plaxton Mesh，也是一种变形的树结构。按照前缀匹配或者后缀匹配的形式，网络被组织成深度为 $\log N$ 的二叉树，每个结点是一个树叶，两个结点间的距离是他们的最小公共子树的高度。每个结点维护 $\log N$ 个邻居，其中第 i 个邻居到自身的距离是 i。在构造路由表时，第 i 个邻居有 2^{i-1} 种选择，这正对应于 ID 与自身匹配前 $(\log N-i)$ 位并且第 $(\log N-i+1)$ 位不同的结点个数。所以可选的路由表构造将有 $\prod_{i=1}^{\log N} 2^{i-1} = 2^{0+1+2+\cdots+(\log N-1)} \approx 2^{\frac{\log N}{2}\log N} = N^{\frac{\log N}{2}}$ 种，这同带弦环结构非常类似，同样提供了非常灵活的路由表构造。

然而，树结构的路由路径灵活性却远远不及环。实际上在树结构中，路由每一跳只有一个邻居可以减短当前结点与目的结点的距离，也就是说只有一个结点可以比上一跳匹配更多的前缀或后缀，所以严格的树结构几乎不能提供路由路径的灵活性。实际的 Tapestry 或者 Pastry 都不是严格的树结构，路由也并不是每一步都非要匹配更多的前缀，有时候只要 ID 更接近就可以了，所以灵活性相对高很多。

3. 环面和超立方体(torus and hypercube)

在第 4 章讲 CAN 时，我们将其拓扑结构归入“多维空间”，实际上多维空间指的是多维环面(d-Torus)。当取 $d=\log N$ 时，CAN 的路由表正构成了一个 $\log N$ 维的超立方体，结点 a 的每一个邻居和 a 仅有一维属性不同，a 的第 i 个邻居和 a 的 ID 仅第 i 比特不同，a 和 b 之间的距离等于两者 ID 不同比特的位数。由此可以看出，超立方体结构中结点的路由表构造是相当严格的，几乎没有什么灵活性。

超立方体结构和树结构的本质不同在于，超立方体路由只要能在任意位上多匹配一个比特就可以了，而树结构路由要求严格的前缀匹配或者后缀匹配。因此，超立方体的路由路径选择比树结构要灵活得多，第一跳约有 $\log N$ 个选择，第二跳约有 $(\log N-1)$ 个选择，总共算起来，可选的路由路径共有 $(\log N)\times(\log N-1)\times\cdots\times 2=(\log N)!$ 条，和带弦环是一样的。

从 2、3 可以看出，树结构和超立方体结构在路由表构造的灵活性与路由路径选择的灵活性两方面，刚好相反，都是在一方面很强，另一方面很弱。所以不管是 Tapestry、Pastry，还是 CAN，都不严格采用两者中哪一方，所以本书对它们的拓扑结构归类也并不严格，“会意”的作用更大。

4. 蝴蝶(butterfly)

传统的蝴蝶拓扑并不能直接用于 P2P 网络的动态性环境中，所以 Viceroy 对

它做了很多改进。在第 4 章我们讲到 Viceroy 的路由分三步完成：第一步，逐层往上走直到第一层；第二步，从第一层逐层往下走直到到达目的结点所在的层；第三步，仅依靠前驱/后继信息在层环上走直到到达目的结点。这三步中，第三步的路由选择是最严格的，几乎没有什么灵活性；前两步的路由灵活性相对好些，但由于 Viceroy 是常数度网络，相比前面讲的 log$N$ 度网络来说，路由灵活性还是要差很多，这不能归结为 Viceroy 设计上的不足，而是常数度 P2P 网络固有的缺陷。

5. 异或(XOR)

也许“异或”不能算一种拓扑，但 Kademlia 确实通过它组织成了一种拓扑结构。由于异或距离明显强调了左边比特的权重，实际上 Kademlia 的路由表构造同树结构或者说 Plaxton Mesh 是类似的，可选的路由表构造将有 $N^{\frac{\log N}{2}}$ 种。

然而，Kademlia 的路由路径选择比起树结构要灵活得多，因为采用 ID 的异或值度量距离，所以路由时可以跳过已匹配前缀后的几位先匹配后面的比特位，而后面匹配的比特位在随后的路由中也并不需要保持，总之只要离目的结点异或距离更短就可以路由。

6. 混合式(hybrid)

绝大多数 P2P 网络混合了多种拓扑结构，所以它们的路由灵活性往往兼有多种拓扑带来的特性，比较难以分析。一般来说，混合式的拓扑结构所提供的路由灵活性通常都比较好。

6.2 分布式散列表

分布式散列表(DHT，sistributed hash table)是 P2P 网络的一个核心设施，它基于一致性散列函数，提供对于任何一个结点、数据对象在覆盖网中的位置映射。这在结构化 P2P 网络中尤其重要，因为它保证了能够准确地定位到某个结点或者数据对象。除此之外，分布式散列表还提供了匿名性这一 P2P 网络独特的属性。鉴于上述基础性的作用，人们也常将 Chord、Pastry、Kademlia 这样的 P2P 网络称为 DHT，但本书中我们一般认为 DHT 是 P2P 网络的一个部件而非全部。

具体地说，如果分布式散列表采用一致性散列函数 $H()$，对于某个网络结点(IP，Port)，该结点在覆盖网上将有唯一对应的标识 nodeID=H(IP，Port，…)，IP 为结点 IP 地址，Port 为端口号，…表示其他属性；对于某个数据对象(Key，…)，它在覆盖网上也有唯一的标识 objectID=H(Key，…)，Key 为对象关键码，…表示其他属性。对结点而言，nodeID 确定了它的覆盖网位置；对数据对象而言，objectID 确定了它的索引信息在覆盖网上的存放位置。

图 6.2.1 所示的 P2P 体系架构反映了 DHT 在 P2P 网络中的地位：分布式散列表位于结构化 P2P 应用和 P2P 覆盖网之间，它组织覆盖网中的结点，向上层提供三个最基本的接口：

(1) Put(Key,Value)：向网络中存储具有标识 Key 的数据 Value。

(2) Remove(Key)：在网络中删除具有标识 Key 的数据对象。

(3) Value=Get(Key)：从网络中获取具有标识 Key 的数据保存在 Value 中。

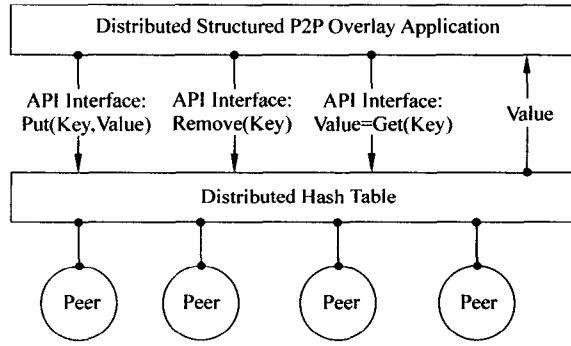


图 6.2.1 P2P 体系架构及其应用接口
(图片来自[Lua et al.,2004])

在本书所介绍的各种 P2P 网络中，每个都自己定义了 Key 与 Value 之间的映射关系，提供了适用于自身的数据存储、获取方法。然而，P2P 是一个全世界共通的领域，如果能够像 Web 那样无缝地整合不同地方、不同应用的服务到一个系统中来，那样 P2P 的影响力和对计算机网络的贡献将得到质的改变。Dabek 等人在 IPTPS'03 会议上概括描述了结构化 P2P 网络公用 API 的规划[Dabek et al., 2003]。而 Karp 等人在 IPTPS'04 会议上提出 OpenHash 体系[Karp et al., 2004]，来整合各种各样的 DHT 到一个通用的 DHT 中：它只提供两个服务接口 put()/get()，同时提供一个公用的路由平台。实际上，OpenHash 可以看作是对 IPTPS'03 上公用 API 规划的一个实现。Rhea 等人在 SIGCOMM'05 会议上发布的 OpenDHT 服务[Rhea et al.,2005]，已经对前面所说的 P2P 公共路由平台做了详细的规划和初步的实现，这项工作如果成功，将成为 P2P 或者说 DHT 历史上的里程碑。

"所有事物都带有两面性。"分布式散列表的使用并非只有利而无害。早在 IPTPS'02 会议上，Ratnasamy 等人发表的论文[Ratnasamy et al.,2002]中，就谈到了有关分布式散列表的诸多开放型问题。因为散列函数屏蔽了输入内容的原本属性，这带来了两个问题：一是拓扑一致性问题，带来通信时延的增加；二是数据对象的语义查询变得十分困难。"语义查询"可以说是结构化 P2P 网络最迫切需要

解决的问题，这些内容在第 7 章再做讨论。下面我们由浅入深地介绍分布式散列表最基础的部分：散列函数→安全散列函数→一致性散列函数。

6.2.1 散列函数

散列函数，又称“哈希”函数(hash function)，它在计算机领域有着广泛的应用，尤其是分布式数据存储(使用一致性散列函数)和数据安全(使用安全散列函数)。

设有函数 $y=H(x)$，H 为散列函数当它具有下面两条性质：

(1) 随机性：H 在值空间上分布足够随机；

(2) 均匀性：H 在值空间上分布足够均匀，以保证负载均衡。

最常用的散列函数有：

(1) MD5 报文摘要算法　散列值 128 位，由于内部缺陷已经被攻破(2005 年)；

(2) SHA-1 安全散列算法　散列值 160 位，比 MD5 要安全一些，是 P2P 领域最常使用的散列函数，但是也由于内部缺陷已经被攻破(2005、2006 年，王小云、姚期智等)，下一步需要使用更高位数的 SHA 算法如 SHA-2(256 位、384 位、512 位)；

(3) RIPEMD-160 报文摘要算法　散列值 160 位，设计用来改进 MD5 的性能；

(4) HMAC 散列报文摘要算法　相当于散列函数和 MAC(报文鉴别码)的结合使用。由于使用了共享的密钥值，攻击者没有办法对它像上述三种情况那样离线攻击，因此，即使 HMAC 中的“H”使用最不安全的 MD5 算法，其攻击难度也是巨大的。当然，安全性的代价是双方共享一个对称密钥，密钥的分配在没有中央控制的系统中成了问题。

6.2.2 安全散列函数

安全散列函数(secure hash function)最初应用于网络安全领域，用来生成报文摘要以保证数据完整性，或者再用私有密钥对报文摘要加密(即“数字签名”)以保证不可抵赖性。P2P 系统使用安全散列函数除了安全性的考虑，还有匿名性：用结点散列值(即 nodeID)在网上标识结点以保护用户信息。

设有散列函数 $y=H(x)$，H 为安全散列函数当它还具有下面几条性质：

(1) 单向性：给定散列值 h，寻找 x 使得 $H(x)=h$ 在计算上不可行；

(2) 弱抗冲突性：给定 x，寻找 y 使得 $H(y)=H(x)$在计算上不可行；

(3) 强抗冲突性：寻找任意一对(x,y)，使得 $H(x)=H(y)$在计算上不可行。

从上到下需求越来越强：强抗冲突必定弱抗冲突，弱抗冲突必然单向。

人们常常将“安全散列函数”简称为“散列函数”，因为对安全性的要求太普遍所以常常省略，实际上 6.2.1 节讲到的四种散列函数 MD5、SHA-1、RIPEMD-160、HMAC 都是安全散列函数。

下面简单地介绍和分析散列函数可能受到的攻击。

(1) 强行攻击　强行攻击破坏散列函数的弱抗冲突性。除非使用高超的数学知识去理解散列函数内部机制并发现其漏洞，否则一般只能采用“暴力破解”逐个尝试的方法。设散列值为 n 位，强行攻击的强度为 2^n 数量级，当 n 较大时，除了碰运气这在计算上几乎是不可能的。

(2) 生日攻击　生日攻击破坏散列函数的强抗冲突性。其原理等同于这样一个有些令人意外的事实：“虽然任意两个人生日相同的概率很小；但是只需要 23 个人，则其中存在两个人生日相同的概率大于 0.5。”

设散列值为 n 位，则可能的散列值有 $N=2^n$ 个。选取包含 k 个随机报文的集合，其中至少有两个报文散列值相等的概率为：$P(N,k)=1-\frac{N!}{(N-k)!\ N^k}$。令 $P(N,k)>0.5$，$k\approx 1.18\sqrt{N}$，即 $k\approx 1.18\times 2^{n/2}$，因此生日攻击的强度为 $2^{n/2}$ 数量级，相当于强行攻击强度的平方根。对于 MD5，$n=128$，攻击强度为 2^{64}；对于 SHA-1，$n=160$，攻击强度为 2^{80}，这两者都非常的大，所以生日攻击在计算上仍然不可行。

6.2.3　一致性散列函数

一致性散列函数是几乎所有的 P2P 系统分布式散列表的基础，这一小节我们仅定性介绍其性质，更为严格的描述见[Karger et al.,1997]。

从数学上看，一致性散列函数是一种动态散列函数，它的值空间范围是动态变化的，因此不同的系统用户看到和用到的值空间也不同。对 P2P 网络来说，每个数据对象(或其索引)都被分布式散列表放到相应的结点中，但由于网络结点不断加入、离开，所以结点集这一“散列值空间”在不断变化。将值空间中的每个位置称为一个“桶”(在 P2P 网络中“桶”就是一个结点)，将每个用户所看到和用到的桶的集合称为它的“视图”(view)，H 为一致性散列函数(consistent hash function)当它具有下面三条性质：

(1) 平滑性(smoothness)：当增加或者减少桶后，需要移动到新桶的对象数目最少。更具体地，要求加入新桶以后，原来的对象只可能从旧桶移到新桶，不可能从旧桶移到旧桶，这也称为单调性(monotonicity)。

(2) 伸展性(spread)：由于每个用户对桶的视图不一样，所以要求同一个对象在所有视图中映射到的桶数尽可能少。

(3) 负载均衡(load)：要求每个桶所装载的对象数目尽可能少且尽可能均衡。

一致性散列函数广泛地应用于分布式的数据存储，它保证了在分布式的动态环境下，散列函数的映射空间将随之平缓地变化并仍然保持负载均衡，从而保证了系统的可用性和工作效率。P2P 网络也是一种动态变化的分布式系统，结点不断

地加入或者离开使得散列值空间处于不断变化之中，它依靠一致性散列函数保证其定位、查询、数据存储与复制的高效性与负载均衡。

P2P 网络所使用的一致性散列函数，多是基于安全散列函数，最常用的是 SHA-1(secure hash algorithm，安全散列算法)。虽然不久前，山东大学的王小云教授破解了 MD-5 和 SHA-1 散列算法，因此它们不再是“安全”的，很多系统采用了更新版本、更长散列值的 SHA 算法，但这与 P2P 网络关系并不大，因为替换散列函数并不是一项复杂的工作。最后，安全散列函数只是一致性散列函数的基础，广义的“一致性散列函数”并不仅仅是一个数学函数，还包括对象分配、移动的规则。

6.3 路由和定位

“路由”和“定位”这两个词在计算机网络中有时很难区分，但对于 P2P 而言，有必要将两者区分开来。所谓“定位”，是指确定位置的过程，它包括两种情况：①结点 A 想确定结点 B 的位置；②结点 A 想确定数据对象 O 的位置。不管哪种情况，P2P 网络定位都是通过一步一步的“路由”来完成的，路由的作用在于将消息送给离目的地更“近”的结点，这里的“近”有多个角度的衡量，因此也就有了各种各样的 P2P 路由方法。所以说，“路由”是“定位”的基本步骤。

路由和定位的方式通常取决于两个因素：①覆盖网拓扑结构；②路由表结构。基于这两个因素，结构化 P2P 网络通常都维护一个比较小的路由表(可变度或者常数度)，采用分布式、局部性的贪心路由算法，逐步缩小当前结点与目的结点之间的 ID 差异，通常定位效率为 $O(\log N)$跳，并且能保证定位成功。对无结构 P2P 网络而言，通常使用“洪泛法”或者其变形方法来路由，跳数限制在 TTL 之内，路由效率不高，并且由于洪泛法带有很大的随机性和局部性，无法确保定位成功(即无法确定某个对象“不存在”)。

P2P 网络的路由方式有很多种，但大多基于同样的思想。概括算起来，P2P 路由方法实际上只有少数几种，并且已经趋于成熟，下面我们概括成几类来叙述。在讲完路由算法后，我们分析两个重要的理论问题：一是 P2P 网络中路由跳数的理论下限，二是结点度和网络直径的折中关系对路由算法的影响。

6.3.1 混合式 P2P 网络的路由和定位方法

混合式 P2P 网络依靠中心服务器来工作，路由方法相当简单：不管是 Napster 还是 BitTorrent，都采用服务器路由。如图 6.3.1 所示的 Napster 路由中，用户只要向服务器提交查询(query)，服务器即给出回复(response)，回复中包含文件拥有者的信息，用户得到回复消息后直接同文件拥有者建立连接，进行文件

下载(file download)。所以星形路由的跳数是 $O(1)$即常数的，路由的性能通常只取决于或者主要取决于服务器。

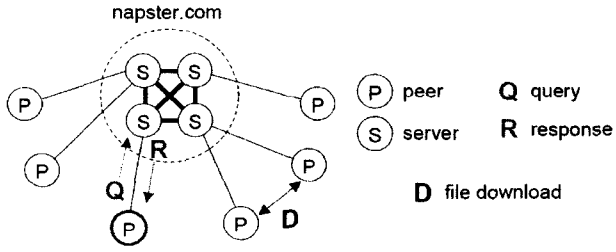


图 6.3.1 Napster 路由和定位示意图
(图片来自[Saroiu et al.,2003])

6.3.2 无结构 P2P 网络的路由和定位方法

无结构 P2P 网络正因为“无结构”，不可能预先知道要找的数据到底在哪里，所以消息路由带有很大的随机性，通常以洪泛法为基础，通过 TTL 来限制洪泛的半径，以减小网络负担。很明显，洪泛法是低效的，所以人们提出很多新的无结构路由方法。在 3.5.2 节我们总结了无结构 P2P 网络最典型的四种路由方法：洪泛法、扩展环、随机走和超结点路由，这里介绍一种无结构查询的新方法：导向路由(guided routing)。

所谓“导向路由”，即路由消息时尽量选择可能包含相关内容的结点作为下一跳。这要求路由表按照数据对象的内容来构造，比如表项中保存文件标识，而此文件标识指向可能包含它的结点。如果说“导向路由”是按照数据对象内容来引导路由的，那么过去介绍的这四种路由方法则是按照结点本身来引导路由的。

Freenet 是“导向路由”的代表，它的路由表就是按照文件来组织的，每个表项记录一个文件标识以及文件标识所指向的结点，此结点很可能包含该文件，或者至少离文件更近。当文件被找到沿原路返回给请求者时，定位路径上的每一跳结点都会在路由表中添加关于此文件的项，如果以后再收到对此文件的请求，定位速度就快多了。所以在 Freenet 中，一个结点对一些文件保存的路由信息会越来越多、越来越好，所以它对于定位这些文件的“经验”就越来越丰富，这种现象称为“标识集群”效应。

图 6.3.2 描绘了 Freenet 中一次典型的“导向路由”过程，本质上它相当于图的“深度优先搜索”，是一个不断探测、回溯，直到成功定位的过程。每个箭头的具体含义，在 3.4.3 节已详细讲解过，这里不再赘述。6.4 节要讲的“路由索引”，本质上也是一种导向路由。

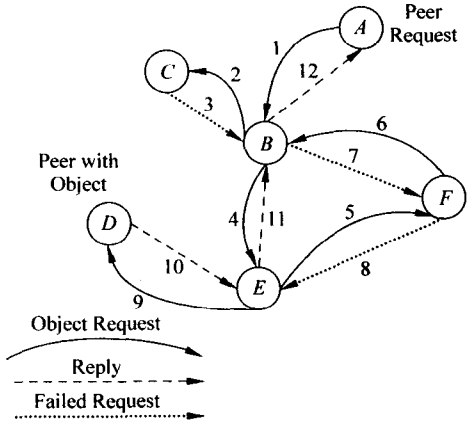


图 6.3.2 Freenet 中的"导向路由"过程示例

(图片来自[Lua et al.,2004])

6.3.3 结构化 P2P 网络的路由和定位方法

结构化 P2P 网络的路由和定位方法与其覆盖网拓扑结构、路由表结构密切相关,虽然看上去方法众多、各有所长,但本质却是一致的:采用分布式、局部性的贪心路由算法,逐步缩小当前结点与目的结点之间的 ID 差异。几乎所有的结构化 P2P 路由方法,都能用 $O(\log N)$ 跳成功定位,下一小节将证明这样的定位效率接近理论上的最优值。我们在 4.7.3 节中总结了结构化 P2P 网络主要的几种路由方法:(1)数值邻近路由;(2)逐位匹配路由;(3)位置邻近路由;(4)层次路由;(5)混合式路由。

6.3.4 P2P 网络定位至少需要多少跳

在发表于 IPTPS'03 的论文[Kaashoek & Karger.,2003]中,得出了下面的关于 P2P 网络路由的数学结论:

定理:假设某个分布式系统中共有 N 个结点,并且结点最大度为 d,那么在最坏情况下定位至少需要 $(\log_d N-1)$ 跳,而平均路由跳数为 $(\log_d N-O(1))$。

证明:因为结点最大度为 d,通过数学归纳法容易看出:从任一个结点 A 出发,与其距离在 h 跳范围内的结点最多有 $d^{h+1}/(d-1)$ 个。令 $d^{h+1}/(d-1)=N$,h 近似为 $(\log_d N-1)$,因此系统内必然存在结点距离 A 至少有 $(\log_d N-1)$ 跳,这是最坏情况的例子,$(\log_d N-1)$ 跳也称为"Moore Bound"(摩尔界限)。

在平均情况下,想象以结点 A 为中心发散出来的结点集群,并假设在每个结点度都接近 d 的情况下,其中大多数结点应当分布在与 A 距离 $(\log_d N-1)$ 跳左

右，因此平均路由跳数为($\log_d N - O(1)$)。如果结点平均度并不接近 d，平均路由跳数还是近似符合上述规律的，除非太多结点度远小于 d。

发表在 INFOCOM'07 会议上的论文[Guo et al.,2007]设计了一个基于"不完备 Kautz 有向图"(incomplete Kautz digraph)的常数度 P2P 模型"Moore"——该名称暗示了它已逼近上述"Moore Bound"，能达到 $\log_d N$ 跳的平均路由跳数。

6.3.5 结点度和网络直径的折中关系对路由算法的影响

上面我们得出了 P2P 网络中路由跳数的理论下限，但仅有此下限是明显不够的，这一小节我们来分析结点度和网络直径的折中关系，以及这种折中关系对路由算法的影响(参考[Xu et al.,2003])。

结构化 P2P 网络中的定位建立在确定性拓扑结构的基础上，从而表现出对网络中路由的指导性和网络中结点与数据管理的较强控制力；但是，确定性的拓扑结构实际上又限制了路由算法效率的提升。研究者分析目前基于 DHT 的结构化 P2P 网络的路由算法，发现算法的两个最重要参数：结点度(表示结点邻居数、路由表项数)和网络直径(路由算法的平均路径长度或跳数)，两者之间存在折中关系，如果用图形描绘这种关系，就表现为一条"渐进折中曲线"。

如图 6.3.3 所示，在 N 个结点的 P2P 网络中，图中直观显示出当度数为 $O(N)$ 时，路由算法的直径为 $O(1)$，即常数跳；而当每个结点仅维护一个邻居状态时，路由算法的直径为 $O(N)$，相当于纯粹的环网路由。上述是度数和直径关系的两种极端情况，其他非极端情况在图中各点有标出。最后从图中可以看出，$O(d)$ 度和 $O(d)$ 直径的路由算法是不可能的。

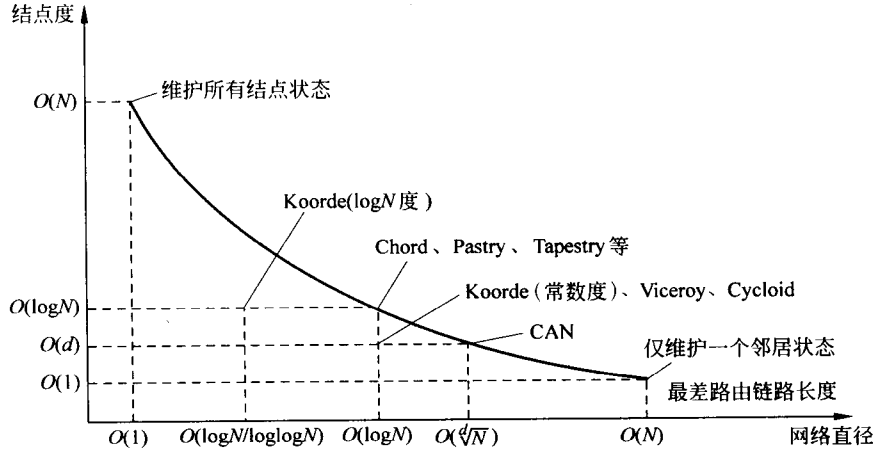


图 6.3.3 结点度和网络直径的折中关系对路由算法的影响

6.4 查询和搜索

6.3 节讲了 P2P 网络的路由与定位，Peer 通过一步步路由能定位到想要的结点和数据。但是这里的“想要”，含义是模糊的：大多数情况下，用户并不清楚自己确切想要什么，好比 Google 上人们只是打入几个关键词，希望从而得到一些东西。因而，P2P 网络需要查询和搜索机制。建立像 Google 那样的一个集中式搜索引擎的方法对 P2P 网络并不适用，原因之一是建立集中式搜索引擎的巨大开销，之二是 P2P 网络中结点和数据的高动态性。

“查询”与“定位”这两个词也常常会被混淆，这里做一个区分，认为“查询”的意义在于根据指定的内容来定位，可以指定结点、数据对象的确切名，也可以像 Google 那样仅指定几个关键词——这就是所谓的“搜索”或称“模糊查询”。精确查询与定位常常是等同的过程，它是结构化 P2P 网络的优势；但模糊查询却比定位复杂得多，是结构化 P2P 网络的劣势，原因就在于“结构化”三字：结构过于严格、分布式散列表又将数据对象映射得毫无规律。目前为止，如何合理可行地实现结构化 P2P 网络的模糊查询，还是一个开放型问题。

由于精确查询一般等同于定位，所以有了 6.3 节这里就不必多说了，本节重点讲“搜索”，也就是“模糊查询”方面的进展。

与结构化 P2P 网络相对，无结构 P2P 网络模糊查询的效果要好一些，不过受制于洪泛法，由 TTL 带来的局部性还是始终存在的。无结构的精确查询虽然效率不高，但通常能查到，不过有一个问题，就是无结构 P2P 网络没有办法确证所指定的东西，也就是说无法确定网络中“没有”这个东西。目前已提出的无结构查询方法可分为三类[Zhu & Hu,2004]：

(1) 随机走，这在 3.5 节已经讲过。

(2) 导向搜索(guided search)，即路由消息时尽量选择可能包含相关内容的结点作为下一跳。上节讲过的 Freenet“导向路由”和下文要讲的“路由索引”，都是一种导向搜索方法。

(3) 基于相似内容组的搜索(similar content group-based search)，将 P2P 网络中的结点组织到包含相似内容的多个组中(“物以类聚，人以群分。”差不多就是这个道理)，查询时首先找到相应的组，这样不仅查询成功率高而且也能做得更快。

结构化 P2P 网络中的模糊查询方法可分两类[Zhu & Hu,2004]：

(1) 关键码搜索(keyword search)：查询请求中包含几个关键码，要返回包含关键码的文件。常用的方法是预先为每个可能的关键码建立索引，说明哪些文件中包含它。下文在讲基于 DHT 的 P2P 网络复合查询时，所提到的“散列索引”就属于此类方法。

(2) 语义搜索(semantic search)：是基于内容的全文本搜索，查询以自然语言的形式表达，而不是仅要求几个关键码的匹配。语义查询首先需要从文件中提取出其代表性的部分，如摘要、标题、关键码，或者更为复杂、抽象的成分。文献[Tang et al.,2003]和[Zhu et al.,2003]都设计了一种结构化 P2P 网络的语义搜索系统，它们都使用了新型的信息检索算法，如向量空间模型(VSM，vector space model)和潜在语义索引(LSI，latent semantic index)，这里不做介绍，有兴趣的读者可参考相关文献。

下面三小节我们重点讲三种 P2P 模糊查询的方法：第一种“路由索引”是很有影响力的导向搜索方法，第二种“基于 DHT 的 P2P 网络复合查询”属于关键码搜索方法，而第三种“前缀散列树”则是一种用来做 P2P 搜索的特殊组织结构。

6.4.1 路由索引

“路由索引”(routing indices)从本质上说，是一种基于结点内容的无结构 P2P 查询方法，不过这里说“无结构”是指查询本身而言，它对结构化 P2P 网络一样是可使用的。

现有 P2P 系统的查询机制，要么需要一个昂贵且脆弱的中央服务器索引，要么是指数开销的洪泛法查询，要么是结构化的精确查询。“路由索引”方法则不同，它允许结点将查询请求发送给更可能包含要查询文件的邻居，这里邻居的选择是基于结点维护的“路由索引”的，所以它既比随机选择邻居合理，又比邻居全选高效。发表在 ICDCS'02 的“路由索引”论文[Crespo & Garcia-Molina,2002]上共设计了三种路由索引方案：复合索引(CRI)、跳数索引(HRI)和指数索引(ERI)，并对它们做了模拟和性能比较。实验结果表明：路由索引方法查询比起洪泛法查询的性能要提高一到两个数量级。

“路由索引”(RI)是 P2P 网络中每个结点维护的一个数据结构(和相关算法)，它的作用在于当结点收到一条查询请求时，返回一个邻居列表，列表中的邻居按照它们对此次查询的“好处”(goodness)来排序。“好处”的衡量可以有所不同，但总是能反映邻近结点包含所要查询文件数目大概的多少。

在路由索引中，结点还有一个“本地索引”(local index)，以快速找到本地资源。此外，以复合路由索引(CRI)方案为例，结点还维护每条路径上的文件数目和关于每个兴趣主题的文件数目。下面是一张简单的路由索引示意图(图 6.4.1)，其中标出了结点 A,D,I,J 的路由索引，A 的路由索引中还特别标出了本地索引的位置。有四个文件主题：DB(database)、N(networks)、T(theory)、L(languages)。第一列的 # 号表示经由此结点可以访问到的文件总数目，而后面的列则表示经由此结点可以访问到的该主题文件数目。举例来说，我们看 A 的路由索引中第 3 行“C|1000 0 300 0 50”，可以看到：A 通过 C 能访问到 1000 个文件，而其中 300 个

关于“Networks”,50 个关于“Languages”。

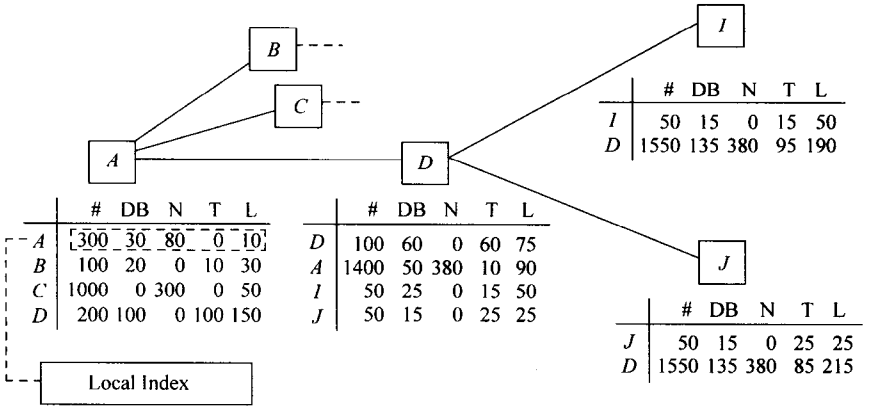


图 6.4.1 一张简单的复合路由索引示意图

(图片来自[Crespo & Garcia-Molina,2002])

有了路由索引后,我们来看如何根据它计算一个结点对于一次查询的“好处”(goodness)。以复合路由索引(CRI)为例,我们基于经由此结点可访问到的文件数来计算“好处”,严格的定义如下:

$$\text{Goodness} = \text{NumberOfDocuments} \times \prod_i \frac{\text{GRI}(S_i)}{\text{NumberOfDocuments}}$$

其中 $\text{CRI}(S_i)$ 表示复合路由索引中第 i 个话题 S_i 的文件数。下面举例解释这个公式:假设结点 A 收到一条关于“Database”和“Languages”的查询请求,那么可以期望经由 B 能访问到的文件数是 $100\times\frac{20}{100}\times\frac{30}{100}=6$,经由 C 能访问到的文件数是 $1000\times\frac{0}{1000}\times\frac{50}{1000}=0$,经由 D 能访问到的文件数是 $200\times\frac{100}{200}\times\frac{150}{200}=75$,所以 B,C,D 的“好处”分别为 6,0,75,基于这个衡量准则,很明显 A 将会把查询请求发给 D。

上面所讲的“复合路由索引”(CRI)的缺点是没有考虑到距离,也就是“跳数”的影响,CRI 只看到了“经由”某结点能访问到的文件数,但没有考虑到达那个文件到底要几跳,而事实上,相隔跳数很大的文件,其“好处”是明显不及相隔跳数很少的文件的。上面的问题正是“跳数路由索引”(HRI)方案的出发点。图 6.4.2 是一个跳数路由索引的例子,它将不同跳数访问到的文件分开保存,很明显,HRI 比 CRI 的信息更详细,但要消耗更多的存储空间,更新起来也更复杂。

“指数路由索引”(ERI)的构建方式比“跳数路由索引”更复杂,但消耗的存储

Node	1 Hop					2 Hops				
	#	DB	N	T	L	#	DB	N	T	L
X	60	13	2	5	10	20	10	10	4	17
Y	30	0	3	15	12	50	31	0	15	20
Z	5	2	0	3	3	70	10	40	20	50

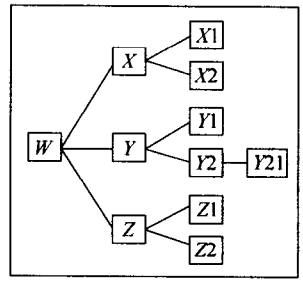


图 6.4.2 左边表格是右边网络中结点 W 的“跳数路由索引”
（图片来自[Crespo & Garcia-Molina,2002]）

空间变少了。实际上，“指数路由索引”可以看成“复合路由索引”与“跳数路由索引”的一个折中，它将文件数目与到达跳数两个因素整合到了一起。这里对 ERI 不做详述。

关于路由索引的查询方法，我们认为它是适用于 P2P 网络的、很有启发意义的查询和搜索机制，而这一小节只讲了其基本原理，很多更深、更细的技术都未涉及。

6.4.2 基于 DHT 的 P2P 网络复合查询

Harren 等人在 IPTPS'02 会议上发表的论文[Harren et al.,2002]中，提出了一个在基于 DHT 的 P2P 网络上构建复合查询设施的设想。他们设计了一个三层的复合查询架构，并对每一层所需做的工作做了描述。下面将介绍这一设想的各个方面。

1. 文本检索和散列索引

文本检索(text retrieval)和散列索引(hash index)的知识是 Harren 等人工作的重要背景知识。计算机领域过去已有研究[Witten et al.,1999]指出：可以使用“散列索引”这一精确匹配设施作为“文本检索”的底层架构，它既支持精确的子串匹配，又支持更模糊的匹配。基本的散列索引方案是将每个字符串拆开，然后按照“n-grams”(长度为 n 的不同子串，我们翻译为“n 长子串”)来索引。举例来说，有一个文件，其 ID 为 I，文件名为“Beethovens 9th”，那么文件名字符串可以被拆成 12 个“3 元子串”(trigrams)：Bee，eet，eth，tho，hov，ove，ven，ens，ns_，s_9，_9t，9th，其中‘_’表示空格符。对于每个“n 长子串”g_i，向散列索引中插入一个以 g_i 标识的有序对(g_i，I)，从而可以通过 g_i 找到原文件的 ID=I。一个自然的问题是：用“n 长子串”构建散列索引时，n 取多少为宜？这个问题的答案要视具体情况而定，但通常$n=2$ 或 3 是典型的取值。

依照上面方法构造的散列索引，可以有效地支持关键码匹配查询，其效果就好比数据库中下面的 SQL 查询语句：

```
SELECT   H. fileID, H. fileName
  FROM   hashtable H
 WHERE   H. text IN(〈list-of-n-grams-in-search〉)
GUOUP BY H. fileID
HAVING   COUNT( * ) >=〈#-of-n-grams-in-search〉
   AND   H. fileName LIKE 〈substring expresssion〉
```

2. 问题：为什么不构建 P2P 数据库来实现模糊查询

数据库技术已出现好多年了，并且发展得非常成熟、完备，在数据库上可以做多种查询，查询的要求几乎是可以由用户任意指定的（比如 SQL 查询语句）。那么，为什么不构建一个 P2P 数据库来实现模糊查询呢？原因有好几方面。

首先，P2P 的优势在于其去掉中心服务器，带来了对等和分布的特性，如果使用 P2P 数据库来模糊查询，那么又重新引入了中心服务器，将已发展 6 年多的 P2P 领域再一次带回到最初的 Napster，这是思想上的倒退。

其次，P2P 网络中包含的文件数目常常是巨量的，构造这样一个数据库需要庞大的存储容量和处理速度。即便使用超级计算机构建了一个这样的数据库，对于数据库中如此多的元组，进行一次模糊查询的开销将是非常大的，而且该数据库从此也成了整个网络的瓶颈。

最后，P2P 网络极具动态性，因而其中的文件也极具动态性，这就要求 P2P 数据库能以非常高的速率检测到文件或结点失效，并做及时的更新，这两项工作都很困难。

3. 复合查询的体系架构

[Harren et al.,2002]中说，他们构建基于 DHT 的 P2P 复合查询架构的目标在于两点：①可以被广泛、实际地使用；②对 DHT 已提供的 API 做最小化的扩展，不引入太多的操作。图 6.4.3 是他们设计的复合查询三层体系架构（单个结点）。

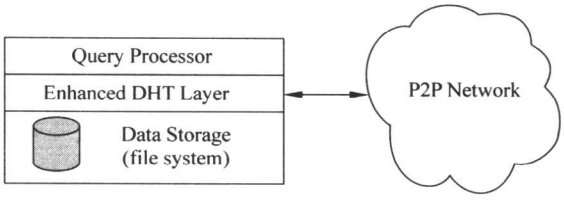


图 6.4.3 基于 DHT 的 P2P 复合查询三层体系架构

图 6.4.3 中，最底层是结点的本地数据存储（文件系统），它应该支持下列 API：

（1）一个迭代器（Iterator），可以用来扫描本地数据存储中所有的对象；

（2）对每个对象而言，提供对其本地标识（localID）和内容（contents）的访问接口；

（3）一个元数据接口，用来查找有关本地数据存储中对象的其他属性；

（4）对其他属性的访问接口。

请注意上述 API 都是只读的。

中间层是增强的 DHT 层，它本身支持 put()/get()接口，同时还提供下列 API 作为“增强”：

（1）一个称为“lscan”的迭代器，可以用来扫描本地计算机上所有的 DHT 项。由一个更高层操作引发的多个“lscan”可以在多台计算机上并发扫描；

（2）一个称为“新数据”（newData）的“回调”（callback）方法，当向 DHT 的本地部分插入新项时通知更高层。

最高层是查询处理层，它支持多种查询操作（如 Join、Selection、Grouping 等）的并发实现、指定查询请求和扫描查询结果。Harren 等人计划在这一层支持两个查询 API：

（1）一个“图-脚本”（graph-scripting）接口，用来指定显式的查询计划；

（2）一个简化的 SQL 接口，以支持布告式的查询（declarative querying）。普通的查询类型（比如关键码搜索）可以由此 SQL 接口来支持。

6.4.3 前缀散列树

使用基于一致性散列函数的分布式散列表的结构化 P2P 网络虽然有很多优点，比如定位准确、高效等，但却有一个共同的缺陷：只支持精确查询。这个缺陷来源于一致性散列函数的本质特点——随机性和均匀性，正是由于散列值和原来的对象没有任何关系，语义上相似的对象被分配到网络中毫不相关的结点，所以不支持关键码模糊查询。但实际上，模糊查询却是所有搜索引擎（如 Google、Yahoo!、百度等）都必须首先支持的重点工作。在第 4 章分析结构化 P2P 网络的特点时，已经提到结构化 P2P 网络之所以不能广泛应用的一个重要原因，就是只支持精确查询。

混合式、无结构 P2P 网络大多支持关键码模糊查询，前者基于服务器之间的协同，后者则基于洪泛式的路由方案，不过它们的模糊查询都有限制：混合式 P2P 网络的查询低效，因为服务器间协同是一项不小的工作；无结构 P2P 网络的模糊查询受限于跳数限制 TTL，只能在局部区域内查询。结构化 P2P 的优点在于准确、高效的定位，如果能采用某些方法结合 DHT 的特点，但同时又支持模糊查询，

将是一项非常有意义的工作。下面介绍的前缀散列树(prefix hash tree,PHT)是一个构建在 DHT 之上的中间件,它封装了 DHT,将数据对象按照二进制形式的关键码组织成一棵前缀树,对上层应用提供模糊查询接口。不过,前缀散列树所提供的"模糊查询"只是一定程度上的,远远没有达到 Web 模糊查询的水平,甚至有时还不及无结构 P2P 网络。文献[Ramabhadran et al.,2004a]是关于前缀散列树的简要介绍,更详细的内容在文献[Ramabhadran et al.,2004b]中。

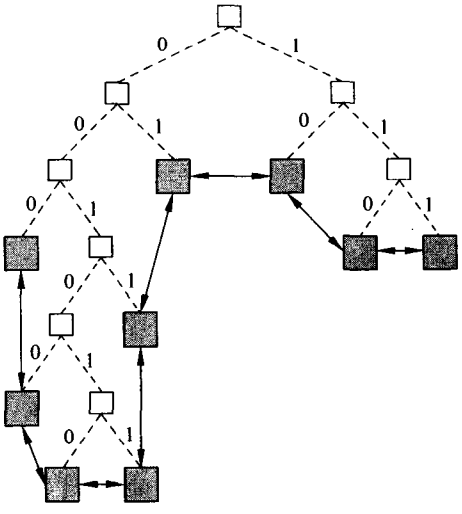
1. 前缀散列树组织结构

在前缀散列树方案中;数据对象关键码都以二进制形式表示,假设共有 N 个对象,采用 D 比特的关键码,D 只要大于 $\log N$ 就可以了。图 6.4.4 左边是一棵简单的前缀散列树,我们结合它来描述 PHT 的特点:

(1) 每个结点要么有两个子结点(一左一右),要么没有子结点,没有子结点的结点称为"叶结点",有两个子结点的结点称为"内部结点"。

(2) 数据对象(的关键码 K)只保存在叶结点中,称为 leaf(K)。每个叶结点最多保存 B 个关键码,每个内部结点的子树中至少保存($B+1$)个关键码。因此,随着关键码的插入或删除,前缀散列树可能会在必要的地方进行分割或者合并。

(3) 每个叶结点维护两个指针(每个指针都是双向的),一个指向其左边关键码最近的叶结点,一个指向其右边关键码最近的叶结点。这个方法将所有的叶结



(a) 前缀散列树结构

Leaf nodes	Keys
000*	000001 000100 000100
00100*	001001
001010*	001010 001010 001010
001011*	001011 001011
0011*	
01*	010000 010101
10*	100010 101011 101111
110*	110000 110010 110011 110110
111*	111000 111010

(b) 关键码分配

图 6.4.4 前缀散列树例

(图片来自[Ramabhadran et al.,2004b])

点按照关键码串成了一条链。

(4) 每个结点都有一个标识(label)。假设父结点的标识为 1，则其左、右子结点的标识分别为 10、11，也就是刚好多一个比特。

(5) 关键码 K 沿着前缀散列树往下走，直到走到其叶结点 leaf(K)，由 leaf(K)负责保存它，如图中 leaf(000001)=000。

(6) 树结点和 P2P 网络结点之间的映射：假设树结点标识为 1，P2P 网络的 DHT 采用的散列函数为 H，那么树结点 1 对应 nodeID 与 $H(1)$最近或相等的覆盖网结点。前缀散列树通过这样的方法封装 DHT。

2. 前缀散列树的线性查询和二分查询

前缀散列树的"线性查询"实际上是对树的自顶向下遍历。假设要查询的关键码为 K，为了找到负责 K 的叶结点 leaf(K)，自树根向下不断匹配 K 的前缀，直到 leaf(K)为止。线性查询算法的效率为 $O(D)$，实际上就是 $O(\log N)$，因为 D 取 $\log N$ 就足够了。线性查询还可以并行来做，也就是将图 6.4.5 的 for()循环变成并行过程，同时查找匹配不同长度的前缀，其中必定有一个会返回 leaf(K)，这增加了查询负担，但减小了时延。

```
Algorithm: PHT-LOOKUP-LINEAR
input: A key K
output: leaf(K)
for i←0 to D do
  /* P_i(K) denotes prefix of K of length i */
  node←DHT-LOOKUP(P_i(K));
  if(node is a leaf node) then return node;
end
return failure;
```

图 6.4.5 前缀散列树"线性查询"算法

(图片来自[Ramabhadran et al.,2004b])

虽然线性查询算法已经很高效了，但下面的"二分查询"算法更高效(图 6.4.6)：它不是从前缀散列树自顶向下逐层匹配 K 的前缀查找 leaf(K)，而是用一个二分的尝试性方法，将关键码 K 的一半拿出来，直接寻找其对应的 DHT 结点。如果该结点刚好对应一个前缀散列树的叶结点，那么查询成功，否则将 K 的后一半再二分，取前缀(即关键码 3/4 的长度)重复上述过程，依次递归做下去，必定可以在 $\log(D+1)$ 即 $\log(\log N)$的调用次数内完成查询。虽然二分查询比线性查询要快，但它只能顺序地来做，而线性查询可以并行做。

```
Algorithm: PHT-LOOKUP-BINARY
input: A key K
output: leaf(K)
lo←0;
hi←D;
while(lo≤hi) do
   mid←(lo≤hi)/2;
   /* P_mid(K) denotes prefix of K of length mid */
   mode←DHT-LOOKUP(P_mid(K));
   if(mode is a leaf node) then return node;
   else
      if(node is an internal node) then lo←mid+1;
      else hi←mid−1;
   end
end
return failure;
```

图 6.4.6 前缀散列树“二分查询”算法

(图片来自[Ramabhadran et al.,2004b])

下面是一个应用前缀散列树的例子(图 6.4.7)。它支持“范围查询”(range query):即给定关键码 L、$H(L \leqslant H)$,要找到所有介于$[L, H]$范围的关键码 K。前缀散列树有两种方法支持这样的范围查询:前一种串行(sequential),后一种并行(parallel)。

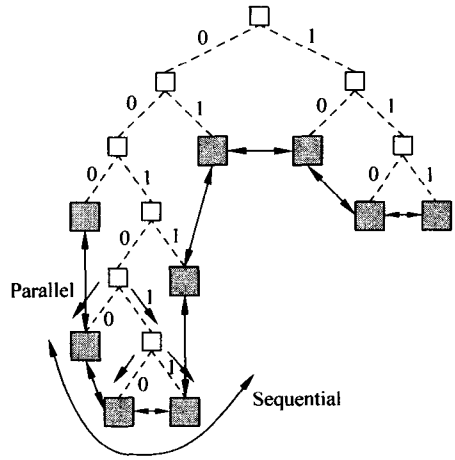


图 6.4.7 前缀散列树“范围查询”示意图

(图片来自[Ramabhadran et al.,2004b])

前一种方法简单直接，首先使用 PHT 查询算法找到 leaf(L)，然后沿着叶结点链逐个遍历到 leaf(H)，返回中间经历的所有叶结点信息。如果 L 与 H 之间的 K 很多，遍历的过程会很慢，带来很大的时延。

后一种方法首先从前缀散列树找一个内部结点，其子树刚好能覆盖[L，H]范围，然后从这个内部结点出发，在它的子树中并行地再做范围查询，直到叶结点为止。比如 L=001001，H=001011，那么首先定位到它们共同的前缀 0010 *，也就是找到内部结点 0010 *，然后从内部结点 0010 * 出发向下递归地并行查找，这涉及范围[L，H]的不断分割，不过分割规则并不复杂。

6.5 动态结点算法

P2P 网络的最大特点和难点，在于它极大的动态性：不断地有新结点加入、旧结点离开、结点失效等情况发生。面对这样一个动态环境，P2P 网络需要一套健全、可行的方法来处理各种动态问题，在新结点加入时通知其他结点新成员的到来，在旧结点离开时通知其他结点老成员的离去，而最难的是需要高效、合理地检测到结点失效并及时修复 P2P 网络。对于这三者的处理，我们统称其为“动态结点算法”。

6.5.1 混合式 P2P 网络的动态结点算法

混合式 P2P 网络中结点加入、离开、失效的处理都由中心服务器来完成，所以它的动态结点算法非常简单：

(1) 加入：用户连接到服务器(这里的服务器通常是众所周知并且稳定在线的)，告诉服务器自己的信息(IP 地址、端口、可以被共享的文件等)，服务器将其保存下来，按照需要或者周期性地把新用户信息发给其他用户；

(2) 离开：用户告诉服务器自己即将离开，服务器从数据库中删除该用户的信息，并按照需要或者周期性地告诉其他用户该用户已经离开(因为其他用户可能有缓存记录)；

(3) 失效：失效情况通常依靠服务器的周期性检测来发现，一旦发现某个用户失效，处理的方法与(2)中用户离开的情形相同。

6.5.2 无结构 P2P 网络的动态结点算法

1. Gnutella 动态结点算法

Gnutella 是最早、最简单，也最经典的无结构 P2P 网络，我们来大概地看一下它是如何处理结点加入、离开、失效的。

如果某台计算机要加入 Gnutella 网络，它首先连接到一个众所周知的 Gnutella 结点（可以从 Gnutella 网站上获得），通过此结点连入 Gnutella 网络，并获得最初的邻居表。然后，新的 Peer 发送消息给其他 Peer 以获得信息和更新别人的信息。这里所用的“消息”就是 Gnutella 组成员消息 PING 和 PONG。

(1) PING：当一个新结点加入网络时，它广播 PING 消息来宣告自己的到来，相当于“I'm here!”；同时 PING 消息也用来探测其他结点是否仍然存在，相当于“Are you there?”

(2) PONG：这是对 PING 消息的回复。当一个结点收到 PING 消息时，它首先决定是否回播 PONG 消息，然后将 PING 广播给它的邻居结点（如果 TTL 没有减到 0）。它包含该结点的 IP 地址、端口号、共享文件的数量和大小等信息，相当于“Yes，I'm here。”

PING、PONG 消息不仅仅用于 Gnutella 结点加入，同样也适用于结点离开和失效的情形。Peer 通过周期性发送 PING 消息、回复 PONG 消息来维护其路由表，路由表的正确度取决于发送 PING 消息的周期，而这又决定了 Gnutella 网络的自适应开销。早期的 Gnutella 网络中 PING 消息超过消息总数的 50%，说明协议中的动态结点算法很低效；改进以后的 Gnutella 网络中 PING 消息不超过消息总数的 10%，说明优化了很多。

2. KaZaA 动态结点算法

KaZaA 是一个双层的无结构 P2P 网络，普通结点不管加入还是离开，都是通过连接超结点来完成的。对失效而言，KaZaA 网络的自适应是通过结点间频繁交换超结点列表来保持的。每当一个普通结点连接到一个超结点后，超结点立刻回送给这个普通结点一个超结点更新列表，其中包含 200 个超结点的 IP 地址、端口号、工作负载值。每当收到超结点更新列表后，普通结点会以列表中的一些项替换掉它原有的超结点列表缓存中的项。

另一方面，KaZaA 超结点之间也交换超结点更新列表。通过相互之间频繁地交换超结点更新列表，KaZaA 的上层“超结点网络”能够不断地重新组织自己、优化自己，保持很高的自适应性。

3. eDonkey 与 Freenet 的动态结点算法

eDonkey 与 KaZaA 的工作原理很像，所以动态结点算法本质上也是类似的。而对 Freenet 而言，动态结点算法就要复杂得多，因为要始终通过加密方法保证安全、匿名，比如 Freenet 中新结点加入就需要一个复杂的多方协商过程，这些在 3.4.5 节中有详述。

6.5.3 结构化 P2P 网络的动态结点算法

最经典的结构化 P2P 网络有 Chord、CAN、Tapestry、Pastry 等，在 4.1 节～4.4 节可以找到它们详细的动态结点算法。实际上，虽然众多结构化 P2P 网络在方方面面各有其独特之处，但它们的结点加入、离开、失效算法本质上却差不多，这里我们做一个概括。

1. 加入

新结点 N 加入结构化 P2P 网络通常需要做三步工作：①初始化自己的路由表；②告诉其他结点自己的到来，更新它们的路由表；③应由新结点 N 负责的对象索引必须从现存结点移交给 N。

对第一步，初始化路由表需要首先通过某种方式连接到一个入口结点 G(通常是某个众所周知的结点)，通过 G 发送消息给离 N 最近的结点(nodeID 最近)。此后有两种方式得到 N 的路由表：一种如 Chord，直接从离 N 最近的结点那里获得路由表；另一种如 Tapestry 或 Pastry，从路由过程中走过的每一跳结点那里获取自身路由表的一部分。

对第二步，更新其他结点的路由表，通常是新结点 N 发送更新消息给路由表中的结点，让它们加以修正。这个过程常常是递归的，路由表中结点还可能通知更多的结点，直到过程收敛或者达到限制条件。这一步是新结点加入中开销最大的，需要发送非常多的消息。

对第三步，对象索引移交，实际上常常可以和第一步合起来做。比如 Chord 中是由 N 的后继结点将部分索引移交给 N，而 Tapestry 中则是路由过程中走过的每一跳结点都可能将部分索引移交给 N，就像给 N 发送路由表一样。

上面三步只是一个总的框架，并不一定足够。比如在 CAN 中，还会进行区域的分裂；而对其他结构化 P2P 网络，都有一些特定于本身的额外操作。

2. 离开

旧结点离开结构化 P2P 网络所要做的工作，可以看成加入时所做工作的逆过程：首先移交对象索引，然后通知其他结点(通常是路由表中的结点)在路由表中删除自己，最后离开网络。离开过程相对简单，不做详述。

3. 失效

结点失效是 P2P 网络工作的大敌，因为它不像结点加入、离开那样是主动发生并且通知他人的，而是被动发生、不可预料的。前文已说过对待失效问题的两种传统方法：一种是按需的，当某个操作发现有结点失效时再做处理；另一种是周

期性的，每隔一段时间检测一次，如果有失效情况再做修复。前者开销小但滞后性大，因为已经犯错了才做处理，属“亡羊补牢”；后者可以提前修正一些错误，但开销很大，大部分周期性检测的工作属“徒劳无功”。实际中多采用后者周期性检测方法，因为它相对简单，易实现。除了这两种传统的方法，Kademlia 还提出了一种全新的“捎带更新”的方法，即每个结点在收到每条路由消息时都捎带更新自己的路由表，这种方法不仅简单而且实用，但并不通用。

Chord 采用称为“稳定化”(stabilization)的周期性检测方法来处理失效问题，检测到失效后，先修理后继关系，再修理路由表。Tapestry 也采用周期性方法，检测到失效后修理路由表，同时 Tapestry 还采用“软状态重发布”的方法让结点定期重新发布自己的对象信息，这些“软状态”信息更重要的作用是为数据对象更新根结点。Pastry 相当于 Chord 与 Tapestry 的结合，同样是周期性检测，发现失效后修复叶集和路由表。CAN 对失效的处理多了一个“背景区域重分配”的过程，这是因为 CAN 的区域划分方法可能导致网络变得支离破碎。

结构化 P2P 网络还有很多种，所以失效时的处理方法也就有很多种，各有其独特之处，读者可以在第 4 章找到对它们的详述。总体而言，结点失效带来的自适应问题，是 P2P 网络尤其是结构化 P2P 网络工作的一大难题，虽然现有方法能满足基本的要求，但更好、更实用的方法仍然是这个领域所期待的。

6.6 容错性

P2P 网络工作在一个极具动态性和成员不可靠的环境下，加之这个年轻的领域在太多地方有不足，因此 P2P 网络工作中可能犯很多错。这些错误从客观上来说，可能来源于结点失效、结点负担过重，从主观上来说，可能来源于恶意攻击所导致的安全问题、无权威认证情况下的用户信誉问题或者是 P2P 网络设计者的疏忽。对如此多错误的避免或者补救，统称为“容错性”。本节讨论容错性时，只针对结点失效，而“热点”问题、信誉问题、安全问题等，将在第 7 章详述。

“没有人能不犯错。对待错误的态度，不是过分苛求，而是尽早补救。”——对于结点失效的容错，正是采取了这个原则。互联网不是绝对可靠的，用户也不是绝对遵守规则的，我们没有办法让互联网不出故障，更没有办法要求用户永远按照规则合理退出 P2P 网络，所以结点失效没有办法避免，只能补救。

“自适应”和“容错性”常常不能很好地区分，这里我们认为：“自适应”指的是对于结点失效的“容错性”，所以本节的话题实际上是“自适应”。为了实现高效的路由，P2P 网络中每个结点会保存一个路由表和其他辅助性的信息，记录它的网络邻居、远程指针等，这些信息称作“结点状态”，它在结点加入网络时被初始化。但是，由于网络环境的动态性，网络结点不断地加入、离开、失效，数据对象也被不断

地插入、删除、废弃，结点状态变得陈旧、与实际网络不一致，这种不一致影响网络工作的效率和正确性。

最典型的容错性机制，是冗余方法和周期性检测：前者通过保存适当的冗余信息，提供有效的替代，以空间换取容错；后者通过每个结点周期性地检测，来及时纠正错误，以时间换取容错。

6.6.1 为了保证容错，结点度至少需要多少

在发表于 IPTPS'03 的论文[Kaashoek & Karger, 2003]中，得出了下面的关于 P2P 网络为了保证容错所需要结点度的数学结论：

定理：对于一个网络而言，假设其中每个结点失效概率为 1/2，那么为了使网络以常数概率保持连通，其中某些结点度必须为 $\Omega(\log N)$，N 为网络结点总数。

证明：先使用反证法来不严格地简单证明此定理。假设网络最大结点度 $d<\log N-2\log\log N-1$，那么任意一个结点被分离（即失去了与它所有邻居的连接）的概率至少是 $(1/2)^{d+1}\geqslant\frac{2^{2\log\log N}}{2^{\log N}}=\frac{\log^2 N}{N}$。网络中共有 N 个结点，所以被分离的结点个数的期望值至少是 $N\times\frac{\log^2 N}{N}=\log^2 N$，也就是说当网络最大结点度 $d<\log N-2\log\log N-1$ 时，是无法期望网络保持连通的。

到这里，看起来定理似乎已经得到了证明，但实际上不是，因为上面的证明过程假定了“结点被分离”事件之间是互相独立的。这个假设并不合理，因为如果一个结点没有被分离，那么它就至少连到了一个“活着”的邻居，而这减少了其他结点被分离的概率。

所以我们换一个思路来证明，仍然用反证法。假设网络最大结点度是 d 且 $d<\log N$，那么每个结点至多有 d 个邻居、d^2 个距离为 2 的邻居。下面我们从 N 个结点中取一个子集 S，S 中共有 $\frac{N}{d^2}$ 个结点，并且 S 中的任意两个结点没有共同的邻居（子集 S 可以用贪心算法构造：在网络中取一个结点，把它放进 S，删除与它距离为 2 的所有邻居，按照这个步骤重复做下去）。对于 S 中的结点而言，因为它们之间没有共同的邻居，所以“结点被分离”事件之间是互相独立的。从而，S 中的结点没有一个被分离的概率至多为 $(1-(\log^2 N)/N)^{N/d^2}<1/e$，e 为自然对数，基于下面的不等式：对于任意正数 x，$(1-x)^{1/x}<1/e$，注意 $d<\log N$。因此，在 $d<\log N$ 的情况下，S 中存在结点被分离的概率大于 $(1-1/e)$，也就是说将以常数概率被分离，整个网络自然无法以常数概率保持连通。综上所述，为了使网络以常数概率保持连通，$d\geqslant\log N$ 是必须的，也就是说其中某些结点度必须为 $\Omega(\log N)$。

6.6.2 容错的传统方法——冗余

“冗余”是容错的传统方法。最简单地说，假定某个部件出错的概率为 p，则它正常工作的概率就为 $1-p$，如果多一个冗余部件，则正常工作的概率就为 $1-p^2$。比如 $p=1/2$，则前者 $1-p=0.5$，后者 $1-p^2=0.75$，如果再多一个冗余，正常工作的概率 $1-p^3=0.875$。

混合式 P2P 网络是以服务器为核心的网络，其容错性只在于服务器的故障概率，如果使用多台服务器组成机群，并且提供冗余、替代机制，使得一台服务故障时它的任务可以被其他服务器所分担，那么很显然，这样的系统容错性将会非常高，这称为“硬件冗余”。然而，增加、升级服务器通常需要昂贵的支出，Napster 只有一个服务器机群，可以接受这样的支出，但 BitTorrent 网络中有太多的服务器，不可能承受这样的开支，所以 BitTorrent 服务器的故障率普遍较高。

无结构 P2P 网络和结构化 P2P 网络中一般没有服务器（双层结构的 P2P 网络如 KaZaA、eDonkey 中的“服务器”是软件概念的，并不是真正的硬件服务器），所以“硬件冗余”是没有必要的，它们都采用“软件冗余”。最常用的软件冗余方法有：路由表项冗余或者数据对象复制。前者指的是一个路由表项中存放多个结点，平时仅使用“主结点”，当“主结点”失效时使用“从结点”来替代，典型应用如 Chord 中使用“后继列表”（Successor List）来取代单后继。后者“复制”方法我们在 7.2 节中重点讨论。

6.6.3 容错性的分类

这里我们所要分类的容错性是狭义的，也就是仅针对“结点失效”的容错性，从此狭义角度我们将 P2P 网络的容错性分成两类：

（1）静态容错性（static resilience）：在保持路由表不变、不做修复（如“稳定化”）的情况下，仅仅删除网络中那些失效结点的相关项，所看到的 P2P 网络的查询成功率；

（2）路由恢复下的容错性（routing recovery resilience）：它与静态容错性的不同在于还进行路由表的修复。

我们先看静态容错性，Gummadi 等人曾在论文[Gummadi et al.,2003]中提到影响静态容错性的最关键因素是网络的“灵活性”（flexibility）。所谓“灵活性”，指的是在确定了源结点和目的结点的情况下，网络中有多少个可选的邻居结点和多少条可选的路由路径。我们细看一下与“灵活性”相关的两种选择：

（1）邻居选择（neighbor selection）：P2P 网络中每个结点都维护一个路由表，某些算法在路由表中使用确定性的邻居（比如 Koorde），而其他一些算法则允许路由表中维护多个可选的邻居，这些邻居的选择原则不仅仅基于 nodeID，常常还基

于其他一些原则(比如 Tapestry 中考虑了时延)。

(2) 路由选择(route selection)：在给定一组邻居和一个目的结点的情况下，路由算法决定了下一跳的选择。虽然如此，当决定了的下一跳失效时，还有多少其他邻居可选作为下一跳？如果这样的可选下一跳很少，路由算法就很可能失效。

Chord、Kademlia、Kelips[Gupta et al.,2003]既提供邻居选择灵活性，又提供路由选择灵活性；Tapestry 只提供路由选择灵活性；而 Koorde 既不提供邻居选择灵活性，又不提供路由选择灵活性。在下文中可以看到灵活性对这些 P2P 网络协议容错性的影响。

下面再看路由恢复下的容错性，我们归纳出路由恢复的三种不同策略：

(1) 按需恢复(on demand recovery)：直到需要的时候才进行恢复，比如邻居结点告诉你它将要离开，或者在实际路由时发现某个结点失效了。

(2) 稳定化恢复(stabilization recovery)：这是一个做周期性失效检测并加以路由表修复的策略，是大多数 P2P 网络所采用的。

(3) 捎带恢复(piggybacked recovery)：某些协议(如 Kademlia、Kelips)使用接收到的每条消息中的信息来捎带更新路由表。

按需恢复是最高效、开销最小的，能适应普通的动态性环境，但在高动态性环境下它无能为力。稳定化恢复是网络开销最大的，但是对高动态性环境的适应性很好。Kademlia、Kelips 之所以能使用捎带恢复，是因为它们的网络路由路径是对称的。

6.6.4 网络动态性的分类

我们经常说 P2P 系统工作在一个极具“动态性”的网络环境下，但是到底什么是网络“动态性”呢？粗略地讲，可以将“动态性”理解为 P2P 网络中的“搅动”(churn)：结点频繁加入、离开或者失效的现象。从更细致的角度看，“搅动”是有程度之分的，我们将其分成两类：

(1) 普通搅动(ordinary churn)：结点一个接一个地串行加入 P2P 网络，或者合理地离开网络，离开前通知其邻居更新路由表。这些搅动事件发生的频率不高，并且网络有足够的时间来做修复，以保持 P2P 网络应有的结构和连通性。

(2) 高搅动(high churn)：P2P 网络中有大量的结点频繁并且并发地加入、离开或者失效。

上面对搅动的分类对应着 P2P 网络动态性的分类：普通动态性、高动态性。可以看出，普通动态性环境和高动态性环境所对应的事件、搅动程度、处理方法是完全不同的，有着质的差别——能够处理普通搅动的方法(如周期性的“稳定化”)在处理高搅动时很可能无效，而在普通动态性环境下能工作得很好的 P2P 网络，在高动态性环境下很可能“崩溃”(“崩溃”的概念将在下文中讲到)。

高动态性环境并非想象出来的，在真实生活中它也会不时出现。比如一些非常热门的电影电视，可能导致在一段很短的时间内有巨大的用户群加入、离开网络；又比如人为设计的大量受控恶意结点并发加入、离开。

高动态性环境给 P2P 网络容错性提出了真正严格但合理的要求，一个“高容错”的 P2P 网络可以理解为一个在高动态性环境下仍然能有效恢复并基本上正常工作的网络。所以说，高动态性环境给 P2P 网络提出了极端的要求，是一块 P2P 真正的“试金石”。

6.6.5 容错性的重要参数——崩溃点

对于一个 P2P 网络而言，结点度越高，通常容错性越好，因为高的结点度意味着网络更强的连通性和灵活性。虽然如此，结点度并非决定 P2P 网络容错性的唯一因素，很多其他因素，比如“集群效应”也在很大程度上影响容错性（这一点我们在讲无结构 P2P 网络的“小世界模型”时就已说过）。因此，通过仔细的设计系统，即使是常数度网络也可以拥有良好的容错性。现在的问题是：如果已知 P2P 网络中结点加入和离开（失效）的频率，以及所要求的容错性，在众多的 P2P 网络中，我们到底应该选择哪一种？

为了回答这个问题，我们做了多方面的工作：首先提出了“崩溃点”这一重要参数作为 P2P 网络容错性的度量[Liu et al.,2004]，其次在统一的 P2P 模拟平台 p2psim 上公平地实验比较了 Chord、Tapestry、Kelips、Kademlia、Koorde 等著名 P2P 协议在高动态性环境下的容错性（基于“崩溃点”来衡量）[Liu et al.,2006a]。

首先我们定义“崩溃点”(crash point)：如果一个 P2P 网络中有 x 比率的结点并发离开（或失效），导致整个网络中 50%的（随机）查询失败，那么 x 就定义为该 P2P 网络的崩溃点。之所以这样定义“崩溃点”，基于下面三个原因：

(1) 结点并发加入、离开 P2P 网络都会造成路由表错误，但是为了实验的方便，不失一般性，仅考虑结点并发离开（失效）的情形。并发离开的结点比率 x 代表着网络动态性的程度。

(2) 这个定义忽略了 P2P 网络不同种类的“崩溃”：一种是单个结点被分离，由它发出或者接收的所有查询都失败；另一种是整个网络被分割成不连通的多个子网，子网之间的所有查询都失败，而子网内部的查询还能成功。忽略了不同种类的崩溃情形之后，我们可以从一个更普适的角度来以同样的准则比较各种 P2P 网络。

(3) 查询成功的比率是易于测量的，所以查询失败的比率也就易得，并且它们和网络连通度有着密切联系。实验中我们发现一旦查询失败率超过 50%，网络基本上再也不可能恢复到整体连通的状态了。

问题：为什么以 50%查询失败作为崩溃点？

答：50%的查询失败率，也就等价于50%的查询成功率，它和网络的连通度有着密切联系。我们假定一个包含 N 个结点的 P2P 网络被分割成了两个等大的子网，每个包含 $N/2$ 个结点，那么此时，可以认为每个子网内 $\frac{N}{2}\times\frac{N}{2}$ 次查询仍然成功，但子网间的查询都将失败，所以平均的查询成功率：

$$\text{SucRatio}=\frac{\frac{N}{2}\times\frac{N}{2}\times 2}{N\times N}\times 100\%=50\%$$

上面假定网络被分割成两个等大的子网是有原因的，因为这对应查询成功率的极小值。我们假设网络被分割成两个不等的子网，包含结点比例分别为 x，y，$x+y=1$，那么此时平均的查询成功率即为：

$$\text{SucRatio}'=\frac{xN\times xN+yN\times yN}{N\times N}\times 100\%=(x^2+y^2)\times 100\%$$

根据不等式 $x^2+y^2\geqslant\frac{1}{2}(x+y)^2$ 可知，SucRatio′≥50%。可见不等分割的查询成功率总是高于均等分割的。

上述内容正是选择50%查询失败作为崩溃点的原因。因此，如果查询成功率 SucRatio 远超过50%，有理由认为网络整体仍然连通，反之如果查询成功率 SucRatio 远低于50%，有理由认为网络被分割，要么已经崩溃，要么离崩溃也不远了。

图 6.6.1 是对 Tapestry 的崩溃点实验结果，它清楚地证明了选择50%的合理性。从图中可以看到，当50%的结点失效时，查询成功率 SucRatio 也降到了50%，而网络从此以后再没有能恢复；而当不超过40%的结点失效时，网络恢复得很好。

图 6.6.2 是5种 P2P 网络崩溃点实验测量结果图。

从图 6.6.2 我们可以看到，虽然 Kelips 通过牺牲存储容量维护了很多的邻居，但其崩溃点并不比 Chord 好，都在70%左右。而 Tapestry 和 Koorde 的崩溃点都在50%左右，其原因可能是 Tapestry 提供了邻居选择灵活性，而 Koorde 虽然没有邻居选择或路由选择的灵活性，但它所基于的环结构提供了比 Tapestry 树结构更好的灵活性。Kademlia 的崩溃点介于50%到70%之间，但是我们认为它在网络规模更大的情况下能工作得更好、更容错，因为它优秀的捎带更新方法。

另外我们观察到的一个现象是，对于 Koorde，70%结点失效时的查询成功率反而高于60%结点失效时的查询成功率，这是因为结点少了，查询也少了，查询路径变短了。虽然如此，因为结点过少、查询过少，测量变得不准确，也没有太多意义，这正是我们在70%结点失效处停止实验的原因。

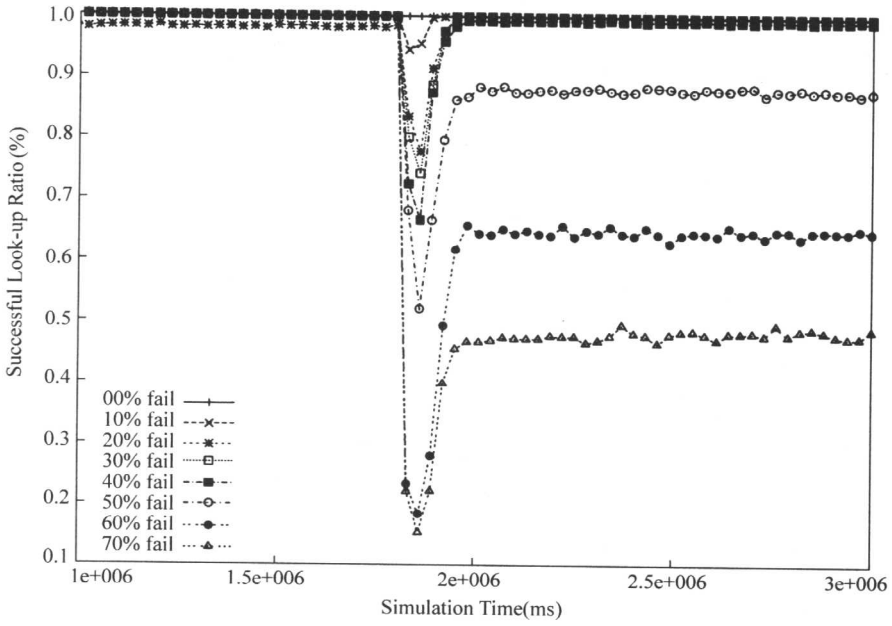


图 6.6.1 Tapestry 崩溃点实验测量结果

(图片来自[Liu et al.,2006a])

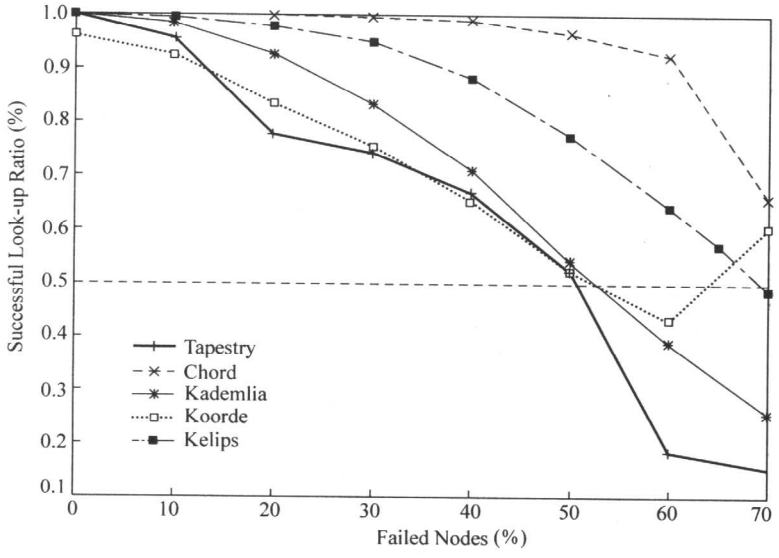


图 6.6.2 5 种 P2P 网络崩溃点实验测量结果

(图片来自[Liu et al.,2006a])

6.6.6 P2P 覆盖网分割问题

“覆盖网分割”(overlay partitioning)，也常称为“网络分割”，一直是 P2P 领域的一大难题。有两种事可能导致覆盖网分割：其一，是粗糙的覆盖网设计，没有考虑到维护连通性；其二，是那些不规则的、意料之外的结点行为。P2P 网络中的结点可能通过事先通知相应结点来“优雅”地离开，也可能由于故障或恶意攻击、在不做任何通知的情况下“唐突”地离开，不管哪种情况，只要离开的结点构成一个“点割集”或其相关边构成一个“边割集”(“割集”的概念可以在图论的相关文献中找到)，覆盖网就会被分割。

1. 覆盖网分割问题解法的初步探讨

一个 P2P 网络对覆盖网分割的容错性，与其本身的连通性密切相关。这里我们将 P2P 覆盖网看成无向图 $G=(V,E)$，V 代表结点集合，E 代表边(逻辑连接)集合，一条逻辑连接可能对应底层物理网上多条连接，但我们不考虑覆盖网与物理网间的匹配关系，仅假设物理网总保持连通，因为如果物理网分割了，构建于其上的覆盖网必定分割。

图论中有两个基本的因素来衡量图 G 的连通性：点连通度 $\kappa(G)$ 和边连通度 $\lambda(G)$，前者是指为使图 G 分割所要移走的最少结点数，后者是指为使图 G 分割所要移走的最少边数。从路由的角度看，点连通度 $\kappa(G)$ 还意味着在图 G 中，任意两个结点间都存在 $\kappa(G)$ 条“独立”的路径，这里“独立”是指路径间没有公共结点；而边连通度 $\lambda(G)$ 意味着在图 G 中，任意两个结点间都存在 $\lambda(G)$ 条“边独立”的路径，这里“边独立”是指路径间没有公共边，但可能有公共结点。

文献[Saroiu et al.,2003]通过细致的测量来研究无结构 P2P 网络 Gnutella 的容错性，其实验结果表明：Gnutella 近似符合 $\alpha=2.3$ 的幂律分布，对随机结点失效有高容错性，但是对恶意攻击的抵抗力则非常弱，尤其是高连接度的结点失效后，容易导致覆盖网被分割。文献[Ripeanu & Foster,2001]指出：如果无结构 P2P 网络中每个结点的连接度能尽量高于某个常数下限，就能有效提高对抗恶意攻击的能力。这里所说的“常数下限”使得我们想到衡量图连通性的第三个因素——最小结点度 $\delta(G)$，源于图论中一个简单而重要的不等式：$\kappa(G)\leqslant\lambda(G)\leqslant\delta(G)$，所以最小结点度实际上确定了点连通度和边连通度的上限，也就间接确定了图的连通性上限。联想到实际的 P2P 网络，诸如 Chord、Pastry 这样的 $\log N$ 度 P2P 模型的 $\delta(G)$ 一般都为 $\log N$，而诸如 Koorde、Viceroy 这样的常数度 P2P 模型的 $\delta(G)$ 则为很小的常数，所以常数度 P2P 网络自然要比 $\log N$ 度 P2P 网络易分割得多。

文献[Loguinov et al.,2005]研究了一些结构化 P2P 覆盖网(Chord、CAN、de Bruijn 图)的图论属性：路由性能、集群属性、等分带宽等，并提出了一种基于最优

直径 de Bruijn 图的网络架构。这里我们尤其关注“等分带宽”(Bisection Width)这一属性：图 G 的等分带宽 $bw(G)$是指将 G 分成结点数相等的两部分(分割方法是任意的)，这两部分之间所连接边数的最小值，由此可以看出等分带宽 $bw(G)$确定了通过删边使图 G 分割成两个均等部分的难度。为什么要注重将图分割成两个“均等部分”？这要追溯到 6.6.5 节已证明的结论：在 P2P 网络被分割成两个部分的诸多情形中，均等分割是查询成功率下降最大的情形。文献[Loguinov et al.，2005]已证明：Chord 的等分带宽为 N(N 为结点总数)，d 维 CAN 的等分带宽为 $2N^{(d-1)/d}$，说明 Chord 和 CAN 的等分带宽都非常大。

通过上文的分析，我们归纳出度量 P2P 覆盖网连通性时需要考虑的四个重要因素：点连通度 $\kappa(G)$、边连通度 $\lambda(G)$、最小结点度 $\delta(G)$和等分带宽 $bw(G)$。

2. 缓解覆盖网分割问题的各种方法

对于 P2P 领域业已提出的缓解覆盖网分割问题的各种方法，我们从两个正交的角度将它们分成四类。第一个角度考虑该方法是主动避免还是被动修复，“主动避免”意味着采取措施主动避免可能出现的覆盖网分割以尽可能保证覆盖网一直连通，“被动修复”意味着当发现覆盖网分割时被动地修复以合并分裂的多个子网。第二个角度考虑该方法是事件驱动还是周期检测，“事件驱动”意味着仅当某些事件(如结点加入或离开)发生时才采取必要的操作，“周期检测”意味着周期性地采取一些维护性的操作。从这两个角度出发，就自然有了四种组合，对应我们分成的四类。

(1) 主动避免和事件驱动

[Pandurangan et al.，2001]将 P2P 网络中结点的加入和离开看成一个“泊松过程”(Poisson process)，提出了一种集中式的方法来维护 P2P 覆盖网的连通性。他们使用集中式的服务器来引导新加入覆盖网的结点与哪些现存结点建立连接、离开覆盖网的结点如何更新信息，前提是要求所有结点的加入和离开都必须预先通知服务器。很明显，这违反了 P2P 的思想，并且要求结点离开时通知服务器是不现实的。

(2) 主动避免和周期检测

文献[Liu et al.，2006b]最先从割点的角度研究覆盖网分割问题，他们设计了一种称为 CAM(connection adjacency matrix)的分布式算法，通过周期性地检测覆盖网割点，并通过割点主动加边以将自身调整成普通结点，来尽量避免割点的存在，从而缓解无结构 P2P 覆盖网的分割问题。CAM 算法可以有效地检测到结点间绝大多数的可达关系(所谓可达，即结点间能互相定位到)，从而较为准确地检测到覆盖网中的割点。下面讲解 CAM 检测割点的这一过程。

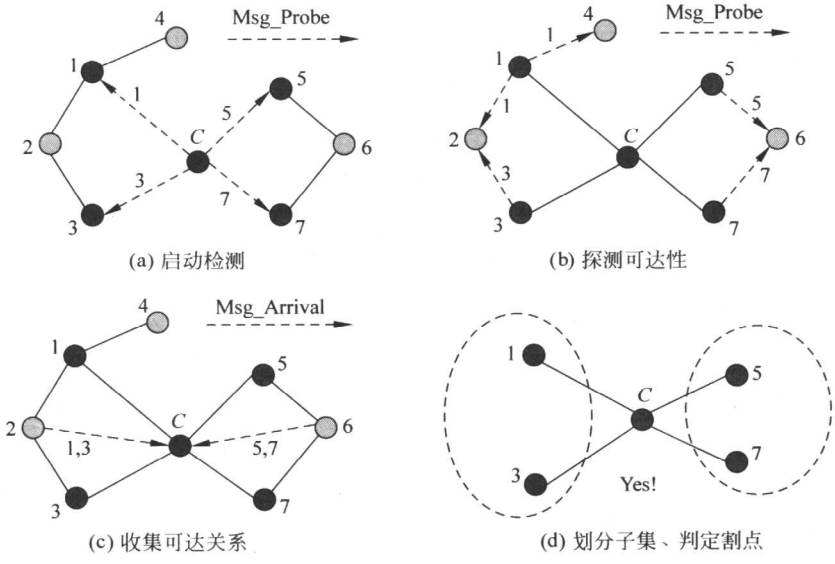
对于无结构 P2P 网络中任意一个结点 C 而言，为了检测自身是否为割点，就必须在 C 不参与消息路由的前提下，探测其邻居之间的可达关系，再依据可达关

系将邻居集分成互不可达的子集；如果子集只有一个，那么 C 不是割点，否则，C 是割点。这就是 CAM 检测割点的中心思想。具体说来，结点 C（或称候选割点）为了检测自己是否为割点，分下面几步进行：

① 启动检测　C 给每个邻居 $N_1, N_2, \cdots, N_n$ 发送探测消息 Msg_Probe（见图 6.6.3(a)），此消息中包含结点 C 的地址、跳数限制 TTL 和邻居号 N_i。网络中每个结点维护一张“连接表”，每个候选割点在连接表中都有其对应的一项，形如〈候选割点地址，邻居号 1，邻居号 2，……〉。结点 N 的连接表中候选割点 C 所对应的项表明 C 的哪些邻居是互相可达的。

② 探测可达性　当某个结点收到 Msg_Probe 消息时，如果 Msg_Probe 中的邻居号是新的，它就把该消息发送给每个邻居（除了消息发送者）（见图 6.6.3(b)），并且将消息中的邻居号 N_i 添加到候选割点 C 的连接表项中。如果连接表项的邻居号不止 1 个，就给候选割点发送 Msg_Arrival 消息，此消息包含该表项中所有的邻居号，实际上就是告诉候选割点哪些邻居可达（见图 6.6.3(c)）。候选割点给其不同邻居发出的探测消息所到达的范围是不重叠的（只在相交的那一点回发 Msg_Arrival 消息报告邻居间的可达关系），避免了重复的可达性探测。

③ 划分子集　C 不断收集 Msg_Arrival 消息，计算其邻居之间的可达关系，将可达的结点划分到同一个子集（这一步实际上是在计算等价类）。鉴于 P2P 网络的动态性，少数 Msg_Arrival 信息可能会丢失，所以 C 可以设置一个超时值 Timeout，在 Timeout 后认为得到了最终的划分。



(a) 启动检测　(b) 探测可达性

(c) 收集可达关系　(d) 划分子集、判定割点

图 6.6.3　CAM 割点检测的过程

④ 判定割点 完成子集的划分后，C 根据子集的数目来判定自己是否为割点：如果所有邻居归到同一个子集，那么 C 不是割点。见图 6.6.3(d)。

CAM 的设计者已在其论文中证明：当跳数限制 TTL 为无穷大时，CAM 能准确检测到所有割点。然而实际网络中不可能设定 TTL＝∞，文献[Liu et al., 2006b]通过模拟实验指出 TTL 取 3 或 4 即可使检测的准确率超过 95%。假设无结构 P2P 网络中共有 n 个结点，c 为平均结点度，t 为跳数限制 TTL，那么每个候选割点检测的开销为 $\min(O(c^t), O(nc))$，所以整个网络检测的总开销(即所需消息数)为 $\min(O(nc^t), O(n^2c))$，当 c，t 均为较小的常数时，总开销近似为 $O(n)$。

需要指出的是，对 P2P 网络来说，CAM 适用的范围是较小的，因为它只能检测割点，不能检测“点割集”，而实际情况下很多网络分割是因为割集的形成。对结构化 P2P 网络而言，点连通度 $\kappa(G)$一般较大，所以并不存在割点，因此 CAM 也不适用。

(3) 被动修复和事件驱动

很多结构化 P2P 网络的工作是基于环结构的，如果大环被分割成多个独立的小环(简称为“分环”)，就不能正常工作。文献[Mahajan et al.,2003]设计了一种能有效检测和修复分环的方法，并对 Pastry 做了具体的实现：当结点 N 发现其“叶集”(leaf set)中某一侧的结点全部失效时，N 就从其“路由表”(route table)中选择在失效一侧最接近 N 的结点作为“种子”(seed)，让种子结点 S 返回其路由表中位于 S 到 N 之间最接近 N 的结点(这里所说的“接近”都指 nodeID 接近)，这一过程循环做下去，直到找到最接近的结点放到 N 的叶集中失效的那一侧，整个过程开销大约为 $\log N$。

发表于 IPTPS'03 会议的文献[Harvey et al.,2003b]专门针对 SkipNet 的分环问题做了研究，但其设计的方法只适用于 SikpNet 这样的提供数据语义、消息路由双重局部性的特殊网络，不具有通用性。

(4) 被动修复和周期检测

文献[Sit & Morris,2002]提出了一种“交叉验证”(cross-check)路由表的方法来缓解覆盖网分割问题：每个结点每隔一段时间随机地向其路由表中的邻居发送校验查询消息，让其邻居执行此查询，然后比较自己查询的结果和邻居返回的结果，如果两者一致，可以近似地认为网络没有被分割；否则，就需要启动相应的修复机制。“交叉验证”方法的主要缺点在于它的随机性和不确定性。

经典的结构化 P2P 网络 Chord 和 Pastry 中都有谈及覆盖网分割问题。在 Chord 中，一个结点 N 周期性地要求其他结点查询自己，如果其他结点不能正确地找到 N，就认为可能存在网络分割。而在 Pastry 中，结点周期性地使用 IP 多播，以扩展环路由的方式来搜索邻近的结点，如果存在网络分割，那么被分离开的结点之间很有可能互相搜索到，那时再来做子网合并。Chord 和 Pastry 的方法同样存在着很大的随机性。

第7章 P2P增强机制

第6章讲述了P2P的核心机制，在具有这些最核心、最基础的部件之后，P2P网络就可以正常工作了。然而，“正常工作”不是我们研究、设计P2P的终点，实际上，它只是个起点。我们所真正追求的，是一个高效率、高可用、高安全的分布式网络，一个不仅“能用”而且“好用”的P2P网络。为了做到这一点，我们需要去分析P2P网络所追求的性能，发现所存在的问题，研究可采用的方法，设计并实现合理的软件。这些工作合起来，我们称其为“P2P增强机制”。

P2P增强机制中不少是从分布式系统或者计算机网络领域借鉴过来的，如复制、缓存和文件分片；当然，更多的增强机制来自于P2P领域本身，如结点负担过重导致的“热点”问题、物理网与覆盖网不一致造成的“拓扑意识”问题、保护用户隐私的“匿名”问题、P2P用户的“声誉”问题和最令人不放心的P2P“安全”问题。我们在本章各节中，将针对这里的每一个问题，做细致的描述。

一个人做一件事，是一个过程，而最终能做到什么程度，是一个结果。“过程与结果，到底哪个更有价值?”绝大多数情况下，期望的结果是驱使我们努力去做的动力。所以本章一开始，专门用一节(7.1节)讨论P2P系统的各项性能，只有在明白了P2P系统到底需要追求哪些性能之后，我们的努力才

有了一个明确的方向。

“复制”已经是计算机领域一个耳熟能详的技术(7.2节),它简单、传统、经典,而且又非常实用,可算是一个“放之四海而皆准”的增强方法。P2P中复制技术所追求的,通常在效率、可用、容错这三个方面:从效率性上讲,通过复制数据对象到多个结点,可以有效地减少获取该对象所需走的路由跳数,从而减少了时延;从可用性和容错性这两方面讲,复制本身是一种冗余技术,一个东西有冗余的替代成分,其可用性和容错性的增强是明显的。除了“复制”,7.2节还讲了“缓存”,从技术的角度讲“缓存”与“复制”并无多大区别,不同点只在于目的与效果:“复制”的效果一般是持久的,其目的既在效率,又在可用和容错;相反,“缓存”的效果一般是临时的,其目的通常只是追求效率。“不同的目的决定了不同的对待方法。”所以P2P网络中,同一个数据对象,一般会被复制在与其有密切联系的结点上,但缓存在查询该对象的路由路径上。

“分片”也是人们总能想到的一种增强技术(7.3节),只是因为它不像复制那么简单,所以总排在“复制”与“缓存”之后。传统的分片方法是把一个文件分成多个等大的片,将这些分片分散存储或者分开传输,从而获得并发效果以提高效率,BitTorrent就是一个最好的例子。但分片的作用远不仅在此,新的分片技术,如著名的“冗余编码”(erasure code),首先将文件分成互相独立的n片,然后将这n片转换成更多相关的分片,比如$2n$片、$4n$片,这些冗余相关的分片散布到网络中后,只要用户能取得其中任意n片,就可以重构原来的对象,所以采用“冗余编码”比采用简单的复制方法对数据可用性、容错性的提高要好得多。

“负载均衡”(7.4节)是所有基于分布式散列表的P2P网络都希望具有的属性,因为它能使得整个网络更高效地工作、更充分地利用每个网络成员的资源。但是对“负载均衡”的理解是个问题:“均衡”应理解成“平均”,还是“各尽所能”呢?这是“异构性”(7.4节)所要解决的问题。P2P网络中,不仅结点间能力差异巨大,网络连接也有明显不同,对“异构性”的开发,正是为了使网络结点各尽所能。

P2P网络,尤其是结构化P2P网络中,一个数据对象可能存放在任何一个结点,但其索引(位置信息)却只能存放在一个特定的结点,通常是nodeID与objectID最近的结点(该结点也称为对象的“根节点”),这在分布式系统领域中称为“Home-Based”,我们可以形象地把那个存放索引的根结点看成对象的“家”。这种方法使得对象位置确定、能快速定位,但带来了一个问题:如果一个数据对象特别热门,那么将有太多的用户试图去获取它,也就会有太多的消息在短时间内一起涌到对象的“根结点”去获取对象索引,这称为“热点”问题(7.4节)。像上述情形,热点问题带来的结果无非两种:①“根结点”由于缓存溢出或者无法处理巨量消息而崩溃;②“根结点”忽略蜂拥而来的巨量消息,绝大多数用户无法获取对象索引,也就得不到想要的对象。“热点”问题是P2P网络中客观存在的,很难避免,也很

难预料，只能想方法缓解，比如 7.3 节所讲的“缓存”就是解决“热点”问题的常用方法。实际上 P2P 网络“异构性”的开发对“热点”问题的解决也很有帮助。

“拓扑意识和一致性问题”(7.5 节)在本书中已经提过多次了，这个问题是与 P2P 在物理网上构建应用层覆盖网并存的。解决一致性问题的方法很多：有从最底层散列函数做起，比如 6.2 节中所讲的“可控散列函数”方法；有从物理层与应用层之间结合的角度做起，比如 Pastry 中每个结点都有一个“邻居集”，其中保存着物理层邻近的结点，这些结点的作用在于指导邻居选择或路由选择向时延更小、拓扑更一致的方向发展；也有直接从应用层覆盖网着手的，比如交换两个 Peer 的 nodeID，让物理网和覆盖网更一致。

“匿名”(7.6 节)并不是 P2P 最初设计的目的，我们可以把它看成 P2P 的一个附带产物。匿名的根源可以追溯到“安全散列函数”，这在 6.2 节有详述。“存在即合理”，不管匿名是对还是错，它作为一种工具的力量是不可否认的，至少它保护了发布者的重要信息和隐私，虽然同时可能带来太多不负责任的行为。在技术上我们倾向于认为：自由、匿名的网络是计算机网络和分布式系统发展的趋势和最终形态。

“声誉”(7.6 节)可以看成对“匿名”的一种克制。好比在人类社会中，“声誉”既是鼓励人奋斗的动力，又是约束人行为的准则；在 P2P 网络中引入声誉机制，既鼓励 P2P 用户为网络贡献一部分能力从而获得有利的声誉，又约束了他们不合法的行为。P2P 网络中的“信任”(7.7 节)常常是基于“声誉”的，或者至少是一个用户对另一个用户的某种行为判定或衡量。马克思说：“经济基础决定上层建筑。”诸如“匿名”、“声誉”、“信任”、“安全”等问题，都可以看成 P2P 的上层建筑，虽然一直有这些方面的研究，但它之所以未成为 P2P 研究的主流，是因为还有更多 P2P 基础性的问题没有得到解决。

“安全”问题(7.7 节)在无论哪个领域中都是复杂、头疼、难以两全的问题。安全机制好比学生宿舍中的管理员：他的存在能让宿舍更安全，但是总要你登记又让你觉得很烦。P2P 领域最初涉及、也持续至今的最大安全问题是“版权问题”，这个问题在计算机其他领域并不存在，因为它不只是计算机本身的问题，而是 P2P 独有的与公司利益、知识产权之间的纠纷。除了版权问题，因为 P2P 本身是分布式系统的一个极端情况，所以分布式系统领域的绝大多数安全问题，在 P2P 网络中都存在，并且常常表现得更严重。许多 P2P 安全问题，到目前为止都是开放性的。

到这里，我们讲到了诸多 P2P 增强机制，其目的无非在于一点：让 P2P 网络更“好用”。古人讲“精益求精”，人的追求是没有止境的，对于 P2P 的进一步完善，对于 P2P 增强机制的研究和实现，今天，只是一个开始。

7.1　P2P 系统的性能

为什么在本章一开始，就用一节专门讨论 P2P 系统的性能？因为只有在明白了 P2P 系统到底需要追求哪些性能之后，我们下面的努力才有了一个明确的方向，才知道到底要“增强”P2P 网络的哪些东西。

表 7.1.1 描述了 P2P 系统的各项主要性能。

表 7.1.1　P2P 系统的各项主要性能

性　能	描　述
结点度	通常为路由表项数，一个结点连接到的其他结点数，也称“邻居数”。结点度决定了每个 P2P 结点所要维护的路由表项数，也就决定了结点自适应工作的开销，并且对于整个网络的连通度有基础性作用
定位效率	定位任意一个结点或者数据对象所需要走过的覆盖网路由跳数，或理解为任意一条消息从源结点出发要到达目的结点所走过的覆盖网跳数。这是 P2P 网络工作时表现出来的最重要的一项外在性能，经常拿它来衡量 P2P 网络的工作效率
定位时延	定位所需要的时间，它是定位过程的真实时延，反映了真正的物理网工作情况。出于简化的目的，很多时候可用物理网跳数(如 IP 跳数)来近似代表时延
Stretch(伸展性)	这是描述 P2P 覆盖网与物理网之间一致性程度的主要参数，它的定义并不统一，一种定义为：$\text{Stretch}=\frac{\text{物理网跳数}}{\text{覆盖网跳数}}$。如果要求更准确，可以用时延来定义 Stretch
负载均衡	数据对象在网络结点中分布得是否均匀，以及每个结点为其他结点消息路由所承担的负载是否平衡。负载均衡是由 P2P 网络的分布式散列表所决定的。此外，对“均衡”的理解，也存在两种态度：“平均”还是“各尽所能”
异构性	异构性反映了 P2P 网络中结点间能力的差异，和网络连接的不同。开发异构性的目的，是为了使网络结点各尽所能
查询负载	为了查询一个数据对象，P2P 网络需要付出的开销，也就是说一共需要路由多少消息。查询负载也是 P2P 网络工作时表现出来的一项重要的外在性能，经常用它来衡量 P2P 查询(定位)算法的效率
自适应开销	随着新结点不断加入、旧结点不断离开或失效，为了维护网络拓扑结构和保持结点状态更新所需要的消息。经常用自适应开销来衡量一个 P2P 网络动态结点算法的工作效率
可扩展性	当 P2P 网络规模增大、结点增多时，是否还能正常工作，其他各项性能发生的变化是否仍然合理、可以接受？可扩展性是衡量一个分布式系统性能的关键属性，在系统规模剧增时尤其重要
可用性	可用性通常表示 P2P 网络中一个数据对象能被获取到的概率。比如当某个数据对象本身不能被获取时，如果它的副本还能得到，那么这个数据对象仍然是“可用”的

续表

性　能	描　述
容错性	“容错性”是一个含义很广的概念，粗略地讲它表示 P2P 网络对错误行为的容忍程度
匿名性	P2P 网络能否保护用户的隐私和重要信息，如用户计算机的 IP 地址、文件的发布者、接收者等
安全性	P2P 网络在一个不可靠的环境中能否正常工作？在受到恶意结点攻击时各项工作是否还能安全进行？有没有应对攻击的合理措施？实际上，安全性在任何领域都是一个异常复杂、困难的问题，它涉及的因素非常多

表 7.1.1 列举的 P2P 系统性能，基本上是从宏观、外在来看，并且是定性而没有从数学上严格定量描述的。下面的各节中，会从微观、内在的角度，定义更为细致、严格、数学化的性能，但从本质上来说，它们与本节所列出的各项性能是一致的。

7.2 复制与缓存

“复制”(replication)技术简单、传统，又非常实用，是计算机领域经常为人们所想到的一种增强方法。P2P 系统中复制技术所追求的，在效率、可用、容错三个方面：从效率性上讲，通过复制数据对象到多个结点，可以有效地减少获取该对象所需要的路由跳数，从而减少了时延；从可用性和容错性上讲，复制本身是一种冗余技术，而冗余正是提高可用性、容错性的经典方法。

“万物皆有两面性。”复制也并非有利而无害。对于复制而言，除了冗余本身的代价，最大的问题是副本之间的“一致性”问题：假设一个对象被复制了 m 份分布在网络中，当一份修改时，需要立即通知其他各份修改吗？如果需要，首先怎么找到其他各个副本，其次如何修改其他各个副本，都是要解决的问题；而如何减少网络开销、高效地完成这一过程，更是不容易的。

除了“复制”，本节还讲了“缓存”(Cache)。从技术的角度讲，“缓存”与“复制”并无多大区别，都是一种冗余技术。它们的不同在于目的与效果：“复制”的效果一般是持久的，其目的既在效率、又在可用和容错；相反，“缓存”的效果一般是临时的，其目的通常只是追求效率。目的不同，方法也不同，所以 P2P 网络中，同一个数据对象，一般会被复制在与其有密切联系的结点上(如“ID 邻近复制”)，但缓存在查询该对象的路由路径上(即“路径缓存”)。

7.2.1 关于复制的一个简单而有用的结论

复制是有好处的，但是不是复制越多越好呢？显然不是。过多的副本占用的存储容量大，更新起来更为困难，实际上不仅没什么好处，反而是对资源的浪费。

那么复制多少份最合适呢？我们从对象可用性的角度简单地推导一下：

假设要求某个数据对象的可用性为 α，单个副本的可用性为 p，那么副本数 r 应该满足

$$\alpha = 1-(1-p)^r$$

将上式变形，副本数 r 应该为

$$r = \frac{\log(1-\alpha)}{\log(1-p)}$$

所以一个对象只要复制 $\frac{\log(1-\alpha)}{\log(1-p)}$ 份，就可以满足可用性的要求。举例来看，如果 $p=0.9$，要求可用性 $\alpha=0.999$，那么复制 3 份就足够了。

7.2.2 无结构 P2P 网络复制的理论模型

发表在 SIGCOMM'02 上的论文[Cohen & Shenker,2002]最早从理论上讨论了无结构 P2P 网络的复制，而 Lv 等人的论文[Lv et al.,2002]则给 Cohen 等人的结论做了很好的总结，并加上不少有意义的扩展内容。这一小节我们基于上述两篇论文，从分析数据查询的分布方式开始，从数学上定义一系列与 P2P 网络复制密切相关的变量、参数，然后将无结构 P2P 网络中的复制方案归为三类：均匀复制、比例复制、方根复制，从数学上推导出哪种复制方案最好(方根复制)。最后设计了一种方根复制的实现方法。实际上其中的许多定义、结论，对结构化 P2P 网络是同样成立的。

假设网络中共有 n 个结点、m 个数据对象，q_i 表示对第 i 个数据对象的查询概率，显然

$$\sum_{i=1}^{m} q_i = 1$$

数据查询的分布典型地有两种：

(1) 均匀分布(uniform)：所有数据对象被查询的概率是均等的，即

$$q_i = 1/m$$

(2) Zipf-like 分布：假设将数据对象被访问的概率从高到低排序，那么这里第 i 个数据对象被访问到的概率正比于 $i^{-\alpha}$，因此绝大多数对象被访问到的概率都比较低，只有少数对象非常高(它们就是所谓的“热点数据”)。对于 Web、Napster、Gnutella 等网络的测量都表明：网络中数据对象的查询遵循 Zipf-like 分布。即：

$$q_i \propto 1/i^{\alpha}$$

假设对象 i 被复制到 r_i 个结点，网络中所有对象副本总数为 R，即

$$\sum_{i=1}^{m} r_i = R$$

那么，数据复制的方法典型地有三种：

（1）均匀复制（uniform）：所有数据对象复制份数相同，即

$$r_i = R/m$$

（2）比例复制（proportional）：数据对象复制的份数正比于其被查询到的概率，越“热门”的数据复制份数越多，即

$$r_i \propto q_i$$

（3）方根复制（square-root）：数据对象复制的份数正比于它被查询到的概率的平方根，虽然越“热门”的数据复制份数仍然越多，但是比“比例复制”要少得多，即

$$r_i \propto \sqrt{q_i}$$

设数据对象 i 的平均查询距离（average search size）为 A_i，即查询对象 i 所走过的平均跳数。在假设副本随机放置的情况下，A_i 应该为

$$A_i = n/r_i$$

整个网络的平均查询距离设为 A，它是反映复制性能的关键参数：

$$A = \sum_{i=1}^{m} q_i A_i = \sum_{i=1}^{m} q_i \frac{n}{r_i} = n \sum_{i=1}^{m} \frac{q_i}{r_i}$$

如果采用均匀复制（uniform），即 $r_i = R/m$，并设网络中每个结点平均负责的副本数为 ρ，显然 $\rho = R/n$，那么 A 应该为：

$$A_{\text{uniform}} = \sum_{i=1}^{m} q_i \frac{m}{\rho} = \frac{m}{\rho}$$

说明均匀复制的情况下，平均查询距离 A 与查询分布无关。

如果采用比例复制（proportional），即 $r_i \propto q_i$，这里取 $r_i = Rq_i$，那么此时 A 为

$$A_{\text{proportional}} = n \sum_{i=1}^{m} \frac{q_i}{Rq_i} = \frac{m}{\rho}$$

说明比例复制和均匀复制具有一样的平均查询距离 A，并且 A 与查询分布无关。

到底什么样的复制策略能产生最小的平均查询距离 A 呢？文献[Cohen & Shenker, 2002]证明了是方根复制（square-root），即 $r_i \propto \sqrt{q_i}$。更具体地说，$r_i = \lambda \sqrt{q_i}$ 时，其中 $\lambda = \dfrac{R}{\sum_{i=1}^{m} \sqrt{q_i}}$，有最优的 A：

$$A_{\text{optimal}} = \frac{1}{\rho} \left(\sum_{i=1}^{m} \sqrt{q_i} \right)^2$$

关于复制的另一个重要参数是“利用率”（utilization rate），数据对象 i 的利用率

$$U_i = R\frac{q_i}{r_i}$$

利用率 U_i 反映了对象 i 的一个副本平均被查询到的次数，它与 A_i 很不一样。比如均匀复制和比例复制，两者的 A 是一样的，但均匀复制时 U_i 正比于 q_i，也就是说数据越“热门”利用率越高；而比例复制时所有对象的 U_i 相同，具有相等的利用率，是查询负载在理论上最平衡的。对于方根复制，它的 U_i 介于均匀复制与比例复制之间，查询负载不及比例复制平衡，但比均匀复制要好得多。

表 7.2.1 总结了三种复制方案的各方面性能。

表 7.2.1 三种复制方案的不同性能

方 案	A	r_i	$A_i=n/r_i$	$U_i=Rq_i/r_i$
均匀复制	$\rho^{-1}m$	R/m	$\rho^{-1}m$	$q_i m$
比例复制	$\rho^{-1}m$	$q_i R$	$(\rho q_i)^{-1}$	1
方根复制	$\rho^{-1}\left(\sum_i \sqrt{q_i}\right)^2$	$R\sqrt{q_i}/\sum_j \sqrt{q_j}$	$\rho^{-1}\sum_j \sqrt{q_j}/\sqrt{q_i}$	$\sqrt{q_i}\sum_j \sqrt{q_j}$

既然方根复制能产生最小的平均查询距离 A，而其利用率介于均匀复制、比例复制之间又是可以接受的，因此，我们可以不严格地说：总体上，方根复制方法是最好的。

既然方根复制最好，一个自然的问题是：怎么实现方根复制呢？文献[Cohen & Shenker，2002]设计了一种方根复制的实现方法：每当一次查询完成后，对象被复制到一定数目的网络结点上，这里“一定数目”指正比于 A_i，即每次查询对象 i 后，$c\frac{n}{r_i}$ 个副本将被放置到网络不同结点中，其中 c 是一个常数。以符号 r_i' 来表示 r_i 的导数，有

$$r_i' = q_i c\frac{n}{r_i}$$

对上式积分，可以看出 $r_i \propto \sqrt{q_i}$。从而说明这种方法确实能达到方根复制的效果。

7.2.3 结构化 P2P 网络中的复制和缓存

结构化 P2P 网络普遍采用“ID 邻近复制”，即将数据对象复制到 ID 邻近的 k 个结点上。这有两个原因：一是 ID 邻近的结点能很快找到，二是 ID 邻近的结点往往均匀分布在网络中(物理网)，它们同时失效的可能性很小，因此提高了数据可用性。

缓存的目的与复制不同，虽然也可以提高数据可用性，但主要是为了提高定位速度（缓存对象索引）和数据获取速度（缓存对象本身）。结构化 P2P 网络基本上使用“路径缓存”，将数据缓存到查询路径上，原因就在于：结构化 P2P 网络由于对象位置确定、拓扑结构严格，不同结点查询同一对象的定位路径往往会有重叠，并且越接近对象保存的结点，路由重叠的可能性越大。

我们首先来看最经典的结构化 P2P 网络 Chord 是怎样进行复制和缓存的。在 Chord 中，正确的后继关系是一切工作的基础，然而单后继是脆弱的，无论机制多么完善，网络的动态性和不确定性都可以导致单后继的失效。鉴于此，实际的 Chord 系统给每个结点维护一个“后继列表”，其中保存了结点在 Chord 环上的 r 个后继，除非 r 个后继都失效，否则结点的后继关系就可以修正。后继列表还帮助 Chord 复制数据，将结点 n 所保存的数据对象复制到 n 的 r 个后继中。Chord 采用典型的“路径缓存”，即在 Chord 定位过程中的每个中间结点缓存所获得的数据对象，这样当后来再定位此数据时，可以在中途就得到数据而不需要走到底。

同 Chord 一样，CAN 也提供了类似的显式复制与缓存方法。但除此之外，CAN 还提供三种“隐式”的复制方法：多空间、多散列、区域超载（详见 4.2 节）。采用这些“隐式”的复制方法可以有效减少定位跳数和每跳时延，同时也提高了系统容错性和可用性。

基于 Tapestry 的 OceanStore 可以算是结构化 P2P 应用中最复杂的了，它的复制方案自然也不同寻常。在 OceanStore 中，任何一个数据对象都对应一个“主副本环”（primary replica，也称为“主环”），它是数据对象复制的关键部分，也是维护副本一致性的核心设施。如果用户要更新对象，它发出的更新请求通过 Tapestry 网络送到对象的主环，主环服务器之间序列化所收到的更新请求并执行它们。然后，主环服务器使用“冗余编码”（erasure coding）技术将新数据对象深度归档存储起来（图 7.2.1 中 Archive 箭头），“冗余编码”属于分片技术（7.3 节会讲到），但这种分片是有冗余的，所以可以看成是一种“复制”。同时，主环通过“分发树”（dissemination tree）将更新逐层发布到对象的“次要副本”（secondary replica）服务器，从而更新了过去的副本，保持了副本的一致性。整个过程如图 7.2.1 所示。

PAST 是基于 Pastry 的 P2P 应用系统，但它的复制方案却比 Pastry 要复杂得多。Pastry 采用传统的“ID 邻近复制”，而 PAST 采用了两种称为“转移”（diversion）的方法来弥补“ID 邻近复制”的不足：一种称为“副本转移”（replica diversion），另一种称为“文件转移”（file diversion），详见 4.4 节。这种特殊的复制方法，实际上是与 PAST 中文件不分片密切相关的。

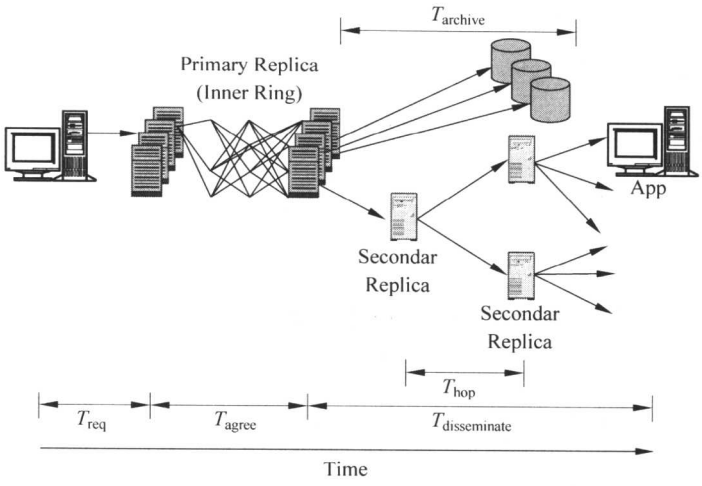


图 7.2.1　OceanStore 副本更新过程
（图片来自[Rhea et al.,2003]）

7.3　分片

“分片”也是计算机领域普遍使用的增强技术，只是因为它实现起来并不简单，所以总排在“复制”与“缓存”之后。有一种著名的算法思想叫“分而治之”，分片技术可以看成这种思想的应用：将整体分解为多个部分，从而使操作的粒度变得更小更灵活，并且往往可以在各个部分间并发进行。

传统的分片方法把一个文件分成多个等大小的片，将这些分片分散存储或者分开传输，从而获得并发效果，以提高效率。BitTorrent 和 eDonkey 都是传统分片的成功案例。

结构化 P2P 应用系统 CFS 和 OceanStore 中的分片是层次化的，前者类似于 UNIX 目录结构，后者则要复杂得多，下文会讲到。

分片的作用并不只在提高效率。新的分片技术，如著名的“冗余编码”(erasure code)方法，能以适度的存储开销获得可用性、容错性的大幅提升。对“冗余编码”的原理讲解及其性能分析、横向比较，是本节的重点内容。

7.3.1　实用的传统分片技术

1. BitTorrent 分片

BitTorrent 的发明人 Bram Cohen 在论文[Cohen,2003]中详细讲述了 BT 的分片技术。实际上，BitTorrent 所提供的是一种传统的分片技术，但已经取得了非

常好的效果。在本书 2.2.3 节中我们已细致地说明了 BT 的分片机制。

2. eDonkey 文件分块

eDonkey 协议将文件分成许多“块”，其实“分块”与“分片”本是一回事。通常每块约 10MB，只要某个客户收到一个完整的分块，它就可以与他人共享该文件。通过分块，eDonkey 实现了类似 BitTorrent 的“多源下载”机制，这是 eDonkey 网络高效的重要原因。3.3.3 节描述了 eDonkey 分块的具体方法。

7.3.2 结构化 P2P 网络中的层次化数据分块

1. CFS 文件分块

Frank Dabek 等人在 SOSP'01 会议上发表的[Dabek et al.,2001]论文中，讲述了基于 Chord 的 CFS(协同文件系统)中层次化的文件分块。CFS 文件系统的组织结构类似于 UNIX V7 文件系统，但使用了 DHash 数据块、块标识来替换 UNIX 中的磁盘块、块地址。每一块要么是文件的一部分，要么是文件系统元数据的一部分，比如目录。块之间有包含关系，通常父块包含其子块的标识。关于 CFS 文件分块的更多内容，可以参阅 4.1.10 节。

2. OceanStore 数据对象分块

UC Berkeley 的 OceanStore 研究组在其论文[Bindel et al.,2000]、[Kubiatowicz et al.,2000]、[Wells 2002]和[Rhea et al.,2003]中讲述了 OceanStore 复杂的数据对象分块方法，以及采用“冗余编码”技术的深度归档存储方案，前者在下面一段文字中回顾，而后者我们会在下面几小节中详细讲解。

在 OceanStore 中，数据对象是最基本的单元，它类似于文件系统中的一个文件。数据对象以“只读文件版本”的方式按序保存在系统中，原则上每个对象的每个版本都是永久保存的，虽然只有最新版通常才有意义。“版本”方法给 OceanStore 的数据对象复制、缓存带来很大的帮助，用户只要知道哪个是新版就可以，没有必要去替换旧版保持一致性(这节省了很多工作量)，旧版的存在在某些情况下也是很有作用的。关于 OceanStore 数据对象分块的更多内容，可以参阅 4.3.7 节。

7.3.3 冗余编码的数学原理

“冗余编码”(erasure code)是一种提高数据可用性、容错性的优秀的数学编码方法。假设编码前数据被分成互相独立的 n 片，“冗余编码”将这 n 片转换成更多相关的分片，比如 $2n$ 片、$4n$ 片，原来分片数与冗余分片数的比称为“冗余编码比

率”(erasure code rate)。这些冗余相关的分片散布到网络中后，只要用户能取得其中任意 n 片，就可以重构原来的对象，所以假设将这些冗余分片广泛地散布到网络中去，除非发生巨大的网络灾难，否则一定可以找到 n 片来恢复原数据对象。

Luigi Rizzo 于 1997 年发表在 ACM Computer Communication Review 上的论文[Rizzo,1997]是关于冗余编码最经典的资料，下面我们对冗余编码数学原理的讲解，正是基于这篇论文，其中涉及不少线性代数和数论的知识，限于篇幅不做详述，读者需参阅相关资料。

1. 总体过程

Luigi Rizzo 起初设计冗余编码是出于可靠的计算机通信协议的考虑：为了在不可靠的通信环境下将源数据(sourece data)可靠地传送给接收方，发送方首先将数据分成 k 份，然后使用冗余编码将 k 份转换成 n 份通过网络传送，接收方只要能收到其中任意 k 份，就能重构原数据。图 7.3.1 形象地描绘了这一过程。

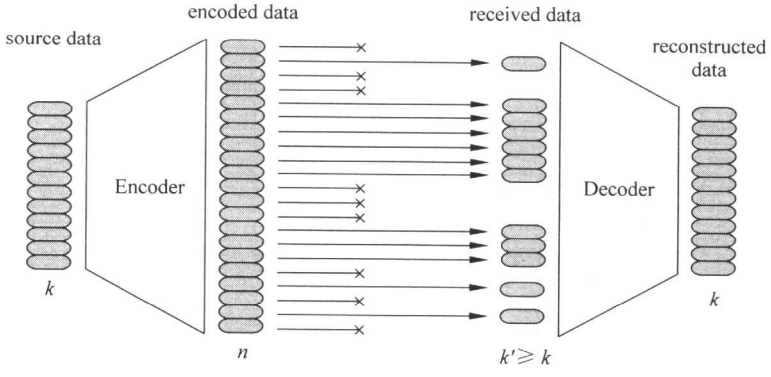


图 7.3.1　使用冗余编码进行数据传输的图示
(图片来自[Rizzo. ,1997])

冗余编码的方式有很多种，其中比较简单、实用的是“线性编码”(linear codes)，这个名称是由于编码采用了线性代数的方法。首先用向量 $\boldsymbol{x}=x_0\cdots x_{k-1}$ 表示源数据(k 为源数据块数)，用向量 $\boldsymbol{y}=y_0\cdots y_{n-1}$ 表示编码后的数据(n 为编码后的块数)，而编码器(encoder)是一个 $n\times k$ 的矩阵 G，那么“(n,k)线性编码”可以表示成下式：

$$\boldsymbol{y}=\boldsymbol{G}\boldsymbol{x}$$

如果用方程组来表示上面的“(n,k)线性编码”，则其中一共有 n 个方程。假定含有 n 个元素的向量 $\boldsymbol{y}$ 中仅有 k 个元素被成功获得(残缺的 $\boldsymbol{y}$ 向量设为 $\boldsymbol{y}'$)，因为 $\boldsymbol{G}$ 是已知的，$\boldsymbol{G}$ 中对应 $\boldsymbol{y}'$ 的那部分矩阵设为 $\boldsymbol{G}'$(它是 $k\times k$ 的)，那么根据 $\boldsymbol{y}'$ 和 $\boldsymbol{G}'$ 可

建立 k 个方程，由此 k 个方程即可解出含有 k 个元素的向量 $\boldsymbol{x}$，也就是重构出了源数据。需要说明的是，上述过程有一个前提：这 k 个方程要互相独立。更具体地说，是要求从 $n\times k$ 的矩阵 $\boldsymbol{G}$ 中任意取出一个 $k\times k$ 的矩阵 $\boldsymbol{G}'$，$\boldsymbol{G}'$都必须可逆（在线性代数中，矩阵可逆等价于其行列式不为 0）。

“线性编码”更为特殊的情形，是编码后向量 $\boldsymbol{y}$ 中包含 $\boldsymbol{x}$，此时的冗余编码称为“系统性编码”（systematic code）。更具体地说，“系统性编码”要求编码矩阵 $\boldsymbol{G}$ 中包含单位矩阵 $\boldsymbol{I}_k$。“系统性编码”的优点在于：当数据错误率或丢失率较低时，它简化了源数据的重构过程。图 7.3.2 描述了“系统性编码”的基本原理。

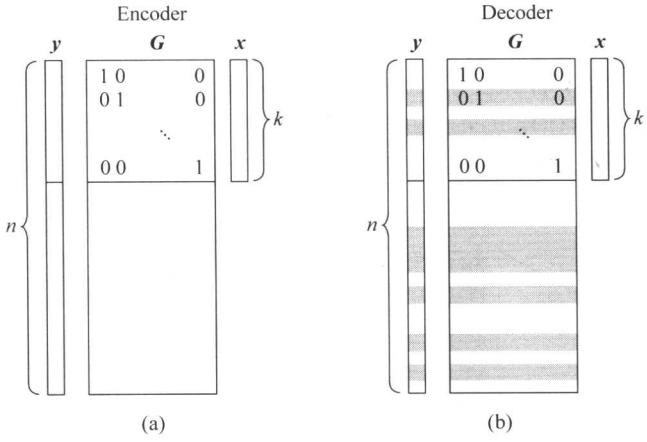


图 7.3.2 “系统性编码”的原理示意图

（图片来自[Rizzo.，1997]）

（(a) 编码矩阵 $\boldsymbol{G}$ 最上面部分是单位矩阵 $\boldsymbol{I}_k$；(b) 阴影部分对应实际收到的数据 $\boldsymbol{y}'$，和对应 $\boldsymbol{y}'$的那部分矩阵 $\boldsymbol{G}'$）

如果上面的编码过程看懂了，那么下面对冗余编码中数据恢复的讲解就显得非常简单了：

$$\boldsymbol{y}' = \boldsymbol{G}'\boldsymbol{x} \rightarrow \boldsymbol{x} = \boldsymbol{G}'^{-1}\boldsymbol{y}'$$

从左式推导出的右式，就是冗余编码的源数据恢复操作。严格地讲，冗余编码恢复源数据时还需要额外的信息：$\boldsymbol{y}$ 中各个块的标识。不过这些额外信息可以其他方式获取，不必显式传送，而且实际上，相比冗余编码中数据块的传送而言，传送块标识信息所消耗的网络资源少到可以忽略。

2. 冗余编码的核心构件：编码矩阵 $\boldsymbol{G}$

冗余编码方案中，对 $n\times k$ 的编码矩阵 $\boldsymbol{G}$ 的要求是：从 $\boldsymbol{G}$ 中任意取出一个 $k\times k$ 的矩阵 $\boldsymbol{G}'$，$\boldsymbol{G}'$都必须可逆，或者说从 $\boldsymbol{G}$ 中任意取出 k 个行向量，这些向量必须互

相独立。因此，$\boldsymbol{G}$ 的每一列至多有$(k-1)$个 0 元。对于“系统性编码”而言，由于其中包含的单位矩阵 $\boldsymbol{I}_k$ 已经消耗了所有的 0 元，所以 $\boldsymbol{G}$ 中其他行都不能含有 0 元。

冗余编码有很多种方法，每种都对应其特殊的编码矩阵 $\boldsymbol{G}$。下面我们讲一种非常简单的构造 $\boldsymbol{G}$ 的方法：$\boldsymbol{G}$ 中的元素 $g_{ij}=x_i^{j-1}$，其中 x_i 是“扩展域”$GF(p^r)$中的元素。这样构造出来的 $\boldsymbol{G}$ 被称为“范德蒙矩阵”(Vandermonde matrices)，其行列式

$$|\boldsymbol{G}|=\prod_{i,j=1,\cdots,k,i<j}(x_j-x_i)$$

上面我们说到“扩展域”$GF(p^r)$，这是数论中的一个概念，其中 p 代表一个素数，$q=p^r$ 是此“扩展域”中的元素个数，$r>1$。下面我们从“域”开始，讲到“素域”，再讲到“扩展域”。

“域”(field)是一个集合，在此集合上我们可以对元素进行加、减、乘、除操作，就好像在整数集上一样。

“素域”(prime field)也称“Galois 域”(Galois field，GF)，写作 $GF(p)$，p 为素数。$GF(p)$中含 p 个元素：从 0 到 $p-1$。在素域 $GF(p)$中的加、乘操作要模 p 进行。

“扩展域”(extension field)写作 $GF(p^r)$，其中含有 $q=p^r$ 个元素，p 为素数，$r>1$。扩展域上的操作比素域要复杂得多，这里不做详述，有兴趣的读者可以参考数论资料。

在说了“扩展域”的概念后，我们回到先前构造的“范德蒙矩阵”$\boldsymbol{G}$。计算 $\boldsymbol{G}$ 的行列式$|\boldsymbol{G}|$的公式上面已写出，所以当构造 g_{ij} 的 x^i 互不相同时，$|\boldsymbol{G}|\neq 0$，矩阵 $\boldsymbol{G}$ 可逆。拿这样的矩阵 $\boldsymbol{G}$ 再加以单位矩阵 $\boldsymbol{I}_k$ 的扩展，就生成了一个“系统性编码”所需的编码矩阵。

上面对冗余编码数学原理的讲解，对没有相关知识的读者，理解起来会很困难；不过这没有什么关系，因为我们讲解冗余编码数学原理的真正意图，只是让读者大概了解一下冗余编码的背景、应用和概要过程。对 P2P 而言，只要知道冗余编码能用来做什么，就足够了。

7.3.4 冗余编码的优点

发表在 IPTPS'02 会议的论文[Weatherspoon & Kubiatowicz，2002]讨论了冗余编码对数据可用性的提高程度，并从理论上比较了冗余编码和复制在多方面的性能和开销，这一小节讲述冗余编码的优点，正是基于他们的结论。

使用冗余编码所获得的高可用性，是对大量数据块的“统计学稳定性”(statistical stability)开发的结果。首先定义如下符号：

P：源数据可用性

n：编码后的数据块数

k：重构源数据需要的块数

N：网络中计算机(结点)总数

M：当前不可用的计算机(结点)总数

基于上述符号，使用冗余编码后源数据可用性符合如下公式：

$$P=\sum_{i=0}^{n-k}\frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}$$

举例来说，如果网络中共有 100 万台计算机，其中 10%不可用(失效)，即 $N=100$ 万，$M=10$ 万。假使对源数据只以复制方法存储 2 个副本，那么其可用性是 $1-0.1\times0.1=0.99$；假使用冗余编码，在消耗同样存储容量的情况下，冗余编码率应该为 $r=1/2$(因为复制用了 2 个副本)，如果将源数据编码成 64 块($n=64$)，需要其中 32 块来重构($k=32$)，代入上式可计算出可用性 $P=0.999\,999\,998$。从这个例子我们可以清楚地看到在数据可用性方面，冗余编码相对于复制的优越性。

文献[Weatherspoon & Kubiatowicz,2002]从三个角度比较了冗余编码和复制的性能和开销，虽然他们的比较是理论层次的，并且忽略了不少现实上的考虑，结果过于乐观，但仍然是很有意义的，其结论概括如下：

(1) 固定 MTTF 和修理间隙　在固定 MTTF(mean time to failure，平均失效时间)和修理间隙(repair epoch)的前提下，复制方法比冗余编码方法所消耗的带宽、存储容量和磁盘搜索次数多一个数量级。

(2) 固定存储开销和修理间隙　使用上面的例子，复制方法中存储 2 个副本，冗余编码率 $r=1/2$，此时两种方法有相同的存储开销。假定将源数据编码成 64 块($n=64$)，就需要其中 32 块来重构($k=32$)，从这个例子计算出复制方法的 MTTF=74 年，而冗余编码方法的 MTTF=10^{20}年！可见在固定存储开销和修理间隙的前提下，冗余编码比复制方法的 MTTF 高很多个数量级。

(3) 固定 MTTF 和存储开销　在固定 MTTF 和存储开销的前提下，冗余编码所需要的修理间隙比复制要大很多，所以为数据修理所耗的带宽要少得多。

最后补充一点，也是文献[Weatherspoon & Kubiatowicz,2002]中所提及的，有关冗余编码的安全问题：在冗余编码方法中，一旦某个分片被恶意修改，而同时它又能被其他人成功获得，那么重构的源数据将是错误的，其危害性要远胜于复制中副本被修改！这个问题解决起来并不困难，只要在每个分片中加上验证码(比如分片散列值)来保证其完整性就可以了，由此所带来的额外的存储、带宽和验证计算的开销也是相对非常少的。

7.3.5　冗余编码 vs. 复制

冗余编码和复制这两种方法，从本质上说都属于“冗余”技术，P2P 领域有不少

论文进行了这两者的比较，有从理论上，有从实验上。7.3.4 节说到，文献[Weatherspoon & Kubiatowicz,2002]认为冗余编码从绝大多数角度来看都远远优越于复制方法；但几年之后在同样的会议上，有人提出了相反的意见，认为冗余编码也有其自身相当多的缺点，在很多情况下使用冗余编码的效果反而不及简单复制。下面我们就来看发表在 IPTPS'05 会议上的论文[Rodrigues & Liskov, 2005]中对冗余编码和复制的比较。

文献[Rodrigues & Liskov,2005]首先对冗余编码和复制方法的"冗余度"(redundancy level)进行比较，也就是比较在相同可用性的情况下，两种方法的存储开销(或者说副本数目)。在 7.2.1 节已推导出复制方法中，为了达到数据可用性 α 而需要的副本数目 $r = \frac{\log(1-\alpha)}{\log(1-p)}$(其中 p 为单个副本可用性)。文献[Rodrigues & Liskov,2005]中使用近似模型得出了冗余编码中需要的存储开销公式，但此公式过于复杂，写出来令读者望而生畏，所以下面我们仅引用他们的实验结果来比较两者的"冗余度"。

从图 7.3.3 可以看出：在服务器可用性越低、所要求的数据可用性越高时，冗余编码的效果越好；但当服务器可用性较高、所要求数据可用性并不太高时，冗余编码的效果并不比复制好多少，而同时它又付出了实现上复杂的代价，所以这个时候，使用复制方法更可取。

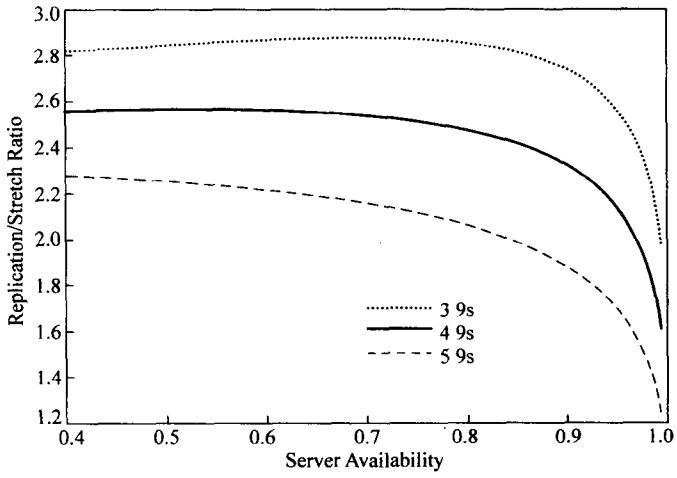


图 7.3.3 横坐标为服务器可用性(即单个结点可用性)，纵坐标为"Replication/Stretch Ratio"，Replication 为复制的存储开销(即副本数)，Stretch Ratio 为冗余编码的存储开销(编码率的倒数)。三条曲线分别表示可用性为 0.999,0.9999,0.999 99

(图片来自[Rodrigues & Liskov,2005])

Rodrigues 和 Liskov 在其论文的最后概括了冗余编码方法的几大缺点：

(1) 冗余编码方法给系统实现带来了很大的复杂性，软件开发和网络通信处理的难度加大，所以除非真的有必要，否则冗余编码方法并不可取。

(2) 在像 Internet 这样的结点间时延相差很大的异构性网络中，冗余编码重构源数据时，从多个结点下载多个分块所要的时延可能很大(取决于最远的那个分块)；相反，复制方法中，常常可以从多个副本中选取最近的那一个来下载，所以时延一般较小。

(3) 有些系统中，重要操作都在服务器端进行，比如关键码搜索；冗余编码不适合这种系统，因为文件都被“拆碎”了。

(4) 一个数据对象不是永远不变的，它可能会更新。在复制方法中，副本更新尚且是一个开销大且不容易的过程，对于冗余编码而言，数据更新难上加难。

最后简单谈一下作者对于“冗余编码”和“复制”这两种 P2P 领域主导性的冗余技术的看法：从实用、可行的角度来说，我们认为在实际开发一个 P2P 系统时，除非对可用性要求极高、并且网络环境非常不可靠时考虑使用冗余编码，否则，简单的复制方法是正确的选择。文献[Wu et al.,2005]中设计了一种将复制和冗余编码混合起来使用的方法，既考虑到用户的文件下载行为，又保证了文件的高可用性，是一个折中方案。

7.4 负载均衡、异构性与热点问题

“负载均衡”是所有基于分布式散列表(DHT)的 P2P 网络都希望具有的属性，因为它能使得整个网络更高效地工作，更充分地利用每个网络成员的资源。

但是对“负载均衡”的理解是个问题：“均衡”应理解成“平均”(每个结点承担同样多的负载)，还是“各尽所能”(能力越强的结点承担负载越大)呢？这是“异构性”所要解决的问题。P2P 网络中，不仅结点间能力差异巨大，网络连接也有明显不同，对“异构性”的开发，正是为了使网络结点各尽所能。还记得《蜘蛛侠》上的一句话吗？——“能力越大，责任就越大。”

P2P 网络，尤其是结构化 P2P 网络中，一个数据对象可能存放在任何一个结点，但其索引(位置信息)却只存放在一个特定的结点，通常是 nodeID 与 objectID 最近的结点(称为对象的“根结点”)，这在分布式系统领域中称为“Home-Based”，我们可以形象地把那个存放索引的结点看成对象的“家”。这种方法使得对象位置确定、能快速定位，但带来了一个问题：如果一个数据对象特别热门，那么将有太多的用户试图去获取它，也就会有太多的消息在短时间内一起涌到对象的“根结点”去获取对象索引，这称为“热点”(flash crowds)问题。下文既讲到了 P2P 网络热点问题的解决，还换一个角度，用 P2P 技术来解决 C/S 模式中的热点问题。

7.4.1 负载均衡

P2P 网络的负载均衡本质上来源于其分布式散列表，因为散列函数能将数据对象随机、均匀地分布到网络结点中，这是其基本属性。但这种分布的“均衡”只是一种“平均”，它不考虑结点之间能力上的不同（即下文的“结点异构性”），往往出现高能力结点空闲、低能力结点过于忙碌的情况，所以这实际上并不是一种好的分布效果。真正好的“负载均衡”，应该是结点根据自己的能力“各尽所能”，这正是对“异构性”开发的目的。

最初意识到该问题并加以解决、应用的 P2P 网络是 KaZaA 和 eDonkey，它们让高能力结点充当网络主干——“超结点”，而让低能力结点连接到“超结点”成为“普通结点”，事实证明这种方法是可行的、有效的。

结构化 P2P 网络中最先提出各尽所能的“负载均衡”方法的是 Chord，它让每个实际的网络结点负责多个“虚拟服务器”（virtual server），由于每个虚拟服务器都负责 Chord 环上连续的一小段，所以每个实际的网络结点负责了 Chord 环上多个不连续的小段，也就是承受了多个不同的负载。当某个结点负载过重时，它会把自己的负载转移一部分给那些负载较轻的结点，负载转移的基本单位正是“虚拟服务器”。所以说：“虚拟服务器”的数目既反映了结点当前承受的工作负载，又反映了结点能力的高低，正是“各尽所能”的。

Rao 等人在 IPTPS'03 上发表的论文[Rao et al.,2003]扩展了 Chord 中“虚拟服务器”的应用，他们设计了静态环境下的三种负载平衡方案：

（1）最简单的方案被称为“一对一”：让每个负载较轻的结点 v 周期性地联系一个随机结点 w，如果 w 负载很重，则 w 将一部分虚拟服务器转移给 v。

（2）第二个方案被称为“一对多”：让负载很重的结点首先联系一个随机的“目录结点”（其中含有一些负载较轻的结点信息），从目录结点那里得知多个负载较轻的结点，然后将自己的一部分虚拟服务器转移给这些结点。

（3）第三个方案被称为“多对多”：在此方案中，每个目录结点包含一些负载较轻和负载较重的结点信息，每个目录结点运行设计好的算法，在负载较轻和负载较重的结点之间重新分配虚拟服务器。这种方案比前两个方案更加“集中化”，性能也更好，只是更复杂。

随后，Godfrey 等人在 INFOCOM'04 上发表了一篇论文[Godfrey et al., 2004]，它在[Rao et al.,2003]的基础上，设计了一种动态环境下的负载平衡算法，并宣称此算法能使系统利用率提高超过一个数量级。

7.4.2 异构性

早在本书第 3 章介绍 KaZaA 和 eDonkey 网络时，我们就说到了 P2P 网络中

存在巨大的异构性，而 KaZaA 和 eDonkey 通过使用“超结点＋普通结点”的双层无结构网络，开发了这种异构性。这一小节的重点在细致分析 P2P 网络中的异构性，并描述解决异构性问题的种种方法。

首先对于“异构性”做更细的分类。实际上 P2P 网络中的“异构性”可以分成两类：①网络异构性；②结点异构性。首先，在像 Internet 这样的大规模网络中，任意两个结点之间的连接，在传输时延和带宽上各不相同，而在覆盖网中，每条逻辑连接是由多条物理连接组成的，所以差别就更大，这就是所谓“网络异构性”。其次，网络中的结点能力千差万别，包括带宽、存储容量、处理速度等，这是所谓“结点异构性”。

“异构性”的存在本身既不是好事，也不是坏事，关键看怎么处理这一问题。对网络异构性来说，如果我们对异构性置之不理，那么在构建路由表时很可能会选择“不好”的邻居，从而带来很大的路由时延；对结点异构性来说，如果所有结点负责的数据项数差不多，那么高能力结点将长期空闲，而低能力结点可能无法承受其工作负载，成为系统的“瓶颈”。因此，如果能开发 P2P 网络的异构性，在路由时选择好的邻居，在负载分配时考虑结点能力，让它们各尽所能，则可以使 P2P 网络工作得更高效、网络资源使用得更充分。7.4.3 节讲述在无结构 P2P 网络的异构性开发方面很有影响力的模型——Gia。

7.4.3 Gia——异构性自适应的无结构 P2P 网络模型

Gia 是在 SIGCOMM'03 会议上，由 P2P 领域的多位专家提出的无结构 P2P 网络理论模型[Chawathe et al.,2003]，它对 Gnutella 做了多方面的改进：首先是采用拓扑自适应算法开发无结构 P2P 网络的结点异构性，实际上 Gia 论文中已提到这是受 KaZaA 影响所设计的。其次，Gia 采用了基于“令牌”的流控制来防止任何一个结点“过热”或过负载。最后，为了提高无结构 P2P 网络的查询效率，Gia 使用了“一跳复制”，并且 Gia 查询协议使用称为“带偏见的随机走”(biased random walk)方法来路由。下面详细介绍 Gia 的每个部分。

1. 拓扑自适应

拓扑自适应算法是 Gia 的核心设施，其作用在于不断调整无结构 P2P 网络的组织方式，让高能力结点拥有高的连接度，形成“集群效应”，以开发网络异构性。

新结点加入 Gia 网络时，首先以类似 Gnutella 的方法，通过一个网络现存结点(“自举”结点)获得其他结点的信息作为自身结点状态的基础。每个 Gia 结点维护一个“主机缓存”(host Cache)列表，其中保存着其他 Gia 结点的位置、能力等。缓存列表应当不断检测和更新：除了周期性地检测结点是否可达，Gia 结点间还周期性地交换缓存列表，以对自己的列表进行优化，优化的目的是形成“集群效应”以开发网络异构性。具体地说，是使得高能力结点拥有高的连接度，低能力结

点拥有低的结点度并且离高能力结点很近，这个过程实际上形成了一个隐式的“超结点网络”。

为了达到拓扑自适应的目标，每个 Gia 结点独立地计算自己的满意度 S(level of satisfaction)。S 在 0 到 1 之间，S 越大说明越满意，只要 S 没有达到 1，拓扑自适应过程就会一直搜索更合适的邻居以提高“集群效应”。图 7.4.1 的算法具体地讲述了 Gia 的拓扑自适应过程。

```
Let C_i represent capacity of node i
if num_nbrsx+1≤max_nbrs then {we have room}
   ACCEPT Y; return
{we need to drop a neighbor}
subset←i ∀ i ∈ nbrsx such that C_i≤C_Y
if no such neighbors exist then
   REJECT Y; return
candidate Z←highest-degree neighbor from subset
if(C_Y>max(C_i ∀ i∈nbrsx)) {Y has higher capacity}
or(num_nbrsz>num_nbrsY+H) {Y has fewer nbrs}
then
   DROP Z; ACCEPT Y
else
   REJECT Y
```

图 7.4.1 Gia 拓扑自适应算法(邻居选择)

(图片来自[Chawathe et al.,2003])

图 7.4.1 中，结点 X 想要接受结点 Y 到自己的邻居表中，如果邻居数未到上限，可以直接将 Y 加入其中；如果邻居表已满，就需要扔掉其中一个，当然被扔掉的也可能是 Y。X 首先从自己的邻居中找到所有能力低于 Y 的结点放到 subset 中，如果没有结点能力比 Y 低，Y 自然被拒绝；不过通常 subset 不会为空，于是从中取出目前所连结点度最高的候选结点 Z，在 Y，Z 中只留一个。如果 Y 的能力高于 X 的所有邻居的能力，或者 Z 所连结点度比 Y 所连结点度多出超过 H(H 是一个常数阈值)，则以 Y 取代 Z，否则拒绝 Y。这个算法的好处在于：①尽量接受能力高的邻居，因为高能力的邻居很可能担当“超结点”的角色；②尽量接受目前连接度比较低的邻居，因为连接度低的邻居在同等能力下可以给自己更好的服务；③这个局部性的算法能在总体上形成网络“集群效应”。

2. 流控制

Gia 流控制的目的是防止任何一个结点“过热”或过负载。采用“流控制”方法后，如果结点 A 想要发送消息给邻居 B，必须首先拥有 B 的“许可”。“许可”可以

量化成“令牌”(token)的形式：每个 Gia 结点周期性地给其邻居分配一定数目的令牌，这通常正比于结点自身的能力，高能力结点发给邻居的令牌也会较多。只有拥有对方的令牌，结点才能发送消息给对方，令牌消耗完了就不能再发送，而只能等新令牌分配，从而避免了过负载问题。

3. 一跳复制

为了提高 Gia 网络查询效率，每个结点动态地维护其邻居结点所拥有数据的索引，通过这样的索引能提高查询速度(下文中将说到它还能缓解“热点”问题)。因为每个数据的索引被“复制”到了离自己“一跳”的邻居中，所以称为“一跳复制”(one-hop replication)。

4. 查询协议

Gia 查询协议使用称为“带偏见的随机走”(biased random walk)方法来路由。“随机走”路由在第 3 章已经讲过，而 Gia 所谓的“带偏见”是指：每个结点在选择下一跳时，不像传统“随机走”那样随机决定，而是去找邻居中结点能力最高的，当然，前提是必须要拥有那个邻居的“令牌”。

无结构路由算法都会有 TTL 跳数限制，Gia 也不例外。但 Gia 在查询过程中使用了“簿记”(book-keeping)技术来避免重复路径，实际上相当于更好地利用了 TTL。每条查询消息有自己唯一的 GUID，Gia 结点会记录下它过去已经给哪条消息路由到哪个邻居，所以，如果具有相同 GUID 的消息又到达这个 Gia 结点时，它就把消息送给与上次不同的邻居。特殊情况是，这个 Gia 结点可能已经把某条消息路由给过所有的邻居，可是这条消息又到了，此时结点就刷新过去的“簿记”，重新开始上面的过程。

无结构 P2P 应用中，很多用户发出的查询请求都是模糊的，希望能找到多个与查询请求比较匹配的数据对象。鉴于这样的需求，Gia 给每条查询消息附加一个“最大响应”(MAX_RESPONSES)参数，每当查询找到一个匹配的对象，MAX_RESPONSES 就被减 1，如果减到 0 查询就终止，即使 TTL 还剩余。

7.4.4 热点问题

“热点”在英文中为“hot spots”，但网络领域中经常称“flash crowds”，可能这样的名称比较形象地描述了某个网络资源在极短的时间(flash，闪光、一瞬)内，对它的获取请求发生了不可预料的、大量的、快速的增长，这种突然爆发的流行性导致网络某个结点造成难以应付的拥挤(crowds，群集、拥挤)。

本节开头已讲到过 P2P 网络，尤其是结构化 P2P 网络中“热点”问题的根源：一个对象的索引只能放在某个特定结点上，造成此“根结点”的负载过重。“缓存”

是解决热点问题的常用方法：在结构化 P2P 网络中，由于定位对象的路径是确定的，所以路由过程中越接近对象索引时，路径重复的概率越大，在此情况下采取“路径缓存”，将对象索引信息缓存到定位路径上，这个简单的方法却能很好地缓解索引的“热点”问题。

解决了对象索引的热点问题之后，对象本身的热点问题接踵而至，对它的处理方法还是传统的“复制”或者“缓存”。结构化 P2P 网络中，保存该数据对象的结点将它复制到 nodeID 邻近的结点，同时对获取请求做“重定向”，告诉获取请求到邻近结点去拿副本。无结构 P2P 网络通常将数据对象复制或缓存到邻居结点，由于获取请求常常先到达邻居结点，所以在那时就可以发现并直接传送副本给需要的用户，根本不用再往前走，也就谈不上“重定向”了。文献[chawathe et al.,2003]中 Gia 采取的“一跳复制”，就是这种方法的典型例子。

7.4.5 有用的引申——用 P2P 技术来解决 C/S 热点问题

7.4.4 节讲述了 P2P 网络本身存在的热点问题及其传统解决方法，本小节我们将 P2P 技术引申到 Web 中，通过构建 P2P 网络来解决 C/S 模式中服务器的“热点”问题。传统的解决 Web 热点问题的方法是将 Web 站点分布到多台服务器中，这些服务器组成一个机群或网络，当出现“热点”现象时复制热点数据对象到多台服务器中，并采取“重定向”方法将获取请求分散开，这种方法需要支付购买多台服务器和网络搭建的费用。而下面我们将讲述的两种方法，前者构建无结构 P2P 网络，后者构建结构化 P2P 网络，给 Web 热点问题的解决带来了新启发。

1. PROOFS

Stavrou 等人构建了一个称为“PROOFS”的无结构 P2P 网络将网络客户(clients)组织起来，以解决网络服务器(server)的“热点”问题[Stavrou et al., 2004]。“PROOFS”即“P2P Randomized Overlays to Obviate Flash-crowd Symptoms”(为消除热点现象而构建的 P2P 随机覆盖网)，是一种使用随机覆盖网构建和随机、局部性查询相结合的简单、轻量的 P2P 方法，它能在大量用户同时要求获取某个对象时，有效地定位和传输对象。

PROOFS 由两个协议组成：①构建覆盖网协议；②定位对象协议。前者构建并维护覆盖网，将网络客户连接、组织起来，这个覆盖网是无结构的，并且其中的结点连接是随机的，结点之间还会周期性地交换邻居。后者类似于 Gnutella 的洪泛式查询协议，但查询过程更随机化、局部化。Stavrou 等人的模拟实验表明：PROOFS 覆盖网中即使有 70%的成员拒绝传送查询消息和数据对象，对热点对象的查询成功率仍然超过 95%；另一方面，PROOFS 对那些非热点的对象查询成功率不高，但这并不在“热点”问题的范围内。

PROOFS 网络中包含两种结点：客户(client)和“自举服务器”(bootstrap server)，客户相当于 P2P 覆盖网中的普通结点，而自举服务器维护一个有限大小的、最近加入网络的客户信息缓存，通过该缓存客户之间可以互相联系和获取信息。

如果一个结点要加入 PROOFS 网络，它将运行构建覆盖网协议(Construct-Overlay)。协议首先联系一个自举服务器来获得初步的邻居列表，从此列表中选取一部分作为结点自己的邻居(并实际地建立连接)，该邻居集也就是构件覆盖网协议所要维护的结点状态信息。

PROOFS 覆盖网会不断通过“混洗”(shuffle)操作交换两个结点的部分邻居来调整覆盖网。具体的“混洗”过程我们不做详述，仅以图 7.4.2 为例来表示其大概的效果。

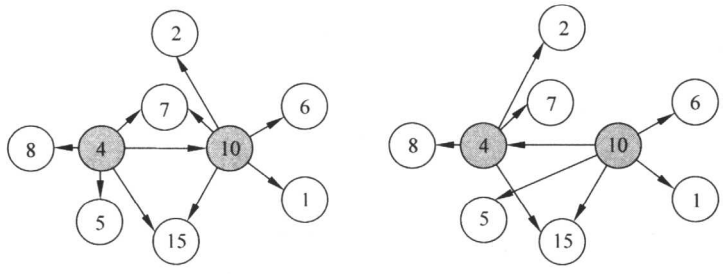


图 7.4.2 “混洗”操作的一个简单例子，左图为混洗前，右图为混洗后
(图片来自[Stavrou et al.,2004])

PROOFS 结点运行定位对象协议(LocateObject)来查询数据对象。查询消息包含四个部分：①对象描述，即查询对象的相关信息；②跳数限制(TTL)，限制查询消息跳数，从而限制洪泛半径；③扇出数(fanout)，限制一个结点能将查询消息发送给多少个邻居；④返回地址(return address)，最先发出此次查询的结点地址。从消息格式可以看出，PROOFS 通过跳数限制(TTL)和扇出数(fanout)来共同限制洪泛法查询的消息数。图 7.4.3 直观地展现了 PROOFS 定位对象协议的查询方式相对于普通 C/S 查询的优越性。

上面的文字只讲述了 PROOFS 的主要工作原理，实际上还有很多细节没有提及，比如用来增强 PROOFS 系统容错性的各种机制。感兴趣的读者可以参考论文[Stavrou et al.,2004]。

2. BackSlash

发表在 IPTPS'02 会议上的论文[Stading et al.,2002]构建了一个称为“BackSlash”的结构化 P2P 网络来主动缓存“热点对象”，并通过“重定向”方法将

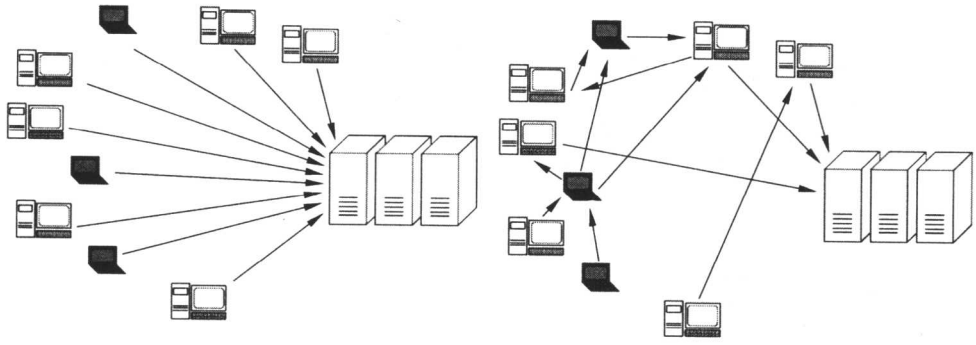


图 7.4.3　左图：在普通 C/S 模式中，每个客户都向 Web 服务器发出查询请求，导致“热点”状况；右图：在 PROOFS 系统中，客户通过“定位对象协议”在无结构 P2P 网中查询对象，分散了热点

（图片来自[Stavrou et al.,2004]）

大量的查询请求分散到缓存的多个副本中。BackSlash 与 PROOFS 的区别，一在它们构建的 P2P 网络，前者是结构化的，后者是无结构的；二在 BackSlash 是构建 P2P 网络来组织属于同一 Web 站点的多个服务器（在 Web 服务器这里分散热点），而 PROOFS 则构建 P2P 网络来组织多个客户（在 PROOFS 客户间分散热点）。

一个 BackSlash 结点对应 Web 站点的一个服务器，在不出现“热点”状况时它们的工作和普通 Web 服务器没有差别，但是当对某个对象的查询请求急速增加时，BackSlash 结点将切换工作状态到两种特殊的模式：①“过负载前模式”（pre-overload mode）；②“过负载模式”（overload mode）。如果结点发现自己的负载短时间内变得很重，但仍然可以应付，就切换到“过负载前模式”，在此模式下结点正常处理已收到的查询请求，但将随后收到的查询请求转移到其他 BackSlash 结点中。而如果结点发现自己的负载几乎无法应付，就切换到“过负载模式”，将所有的查询请求重定向到“替代 BackSlash 结点”（surrogate BackSlash nodes）中。更深入地说，BackSlash 结点的“重定向”是通过改写 Web 站点中文件的 URL（uniform resource locator，统一资源定位符）来实现的。至于 URL 如何改写，这里不做介绍，可参照原论文。

一旦某个“替代 BackSlash 结点”收到关于数据对象的查询请求，它就开始充当 Web 客户和 BackSlash 的“协同镜像部分”（collaborative mirroring portion）之间的“网关”（gateway），而构成此“协同镜像部分”的多个服务器是使用结构化 P2P 网络组织起来的。目前的 BackSlash 使用 CAN 网络来组织“协同镜像部分”，但实际上，可以用任何一个结构化 P2P 网络来替换 CAN。

每个 BackSlash 结点都分配一些存储容量用作“缓存”（广义的概念，实际上也

包括了复制），这部分存储容量被分成两部分："副本空间"和"临时缓存空间"。当在 BackSlash 的镜像 P2P 网络中插入某个对象时，副本也被创建，并且副本将长期存在；而临时缓存则是在收到该对象的查询请求后创建，以加速随后的查询，并且临时缓存是暂时性存在的，如果以后不被使用到则会被替换。

"缓存"在 BackSlash 的镜像 P2P 网络中有两种可取的传播方式：第一种方式称为"本地传播"（local diffusion），每个含有对象副本或临时缓存的结点监测它所收到的对象查询请求速率，如果该速率达到某个预定的"推出阈值"（push threshold），则结点将对象的一份新的临时缓存往前"推出"一跳，也就是将此临时缓存放置到离发出查询的结点更近的前一跳结点中。我们觉得 BackSlash 的这种"本地传播"方法和文献[Chawathe et al.,2003]中设计的 Gia 模型的"一跳复制"方法有本质的相似。

第二种传播方式称为"目录传播"（directory diffusion）。在此方式下，镜像 P2P 网络中的结点将保存一个"指针目录"，其中的"指针"指向其他对象的副本，而不是他们自己的副本（这与结构化 P2P 网络的对象索引保存方法是本质上一样的）。当某个结点 A 收到一个新插入对象时，它为该对象创建一个目录，然后在 BackSlash 的镜像 P2P 网络中随机选取一个结点 B，在结点 B 上为对象存储一个副本，并在 A 的目录中记下结点 B 的地址（这就是一个"指针"）。目录结点监测它所收到的对象查询请求速率，如果速率较低，则它返回与此对象相关的目录项；如果查询请求速率达到某个预定的"阈值"（threshold），则目录结点向发出查询的结点回复一个"请帖"，请求它成为新的临时缓存结点。

关于 BackSlash 就讲到这里。虽然要在 Web 站点中真正构建出 BackSlash 所设计的镜像 P2P 网络并不是一件容易的事，但 BackSlash 所体现的"用 P2P 新技术来解决计算机网络领域老问题"的思想，是很有启发意义的。

7.5 拓扑意识和一致性问题

7.5.1 处理一致性问题的三种传统方法

处理 P2P 网络拓扑意识和一致性问题的方法有很多，有的从底层散列函数（ID 编码）做起，有的从物理层与应用层结合的角度做起，有的直接从应用层覆盖网着手。处理一致性问题的传统方法有如下三种[Gummadi et al.,2003]：

（1）邻近路由选择（proximity route selection，PRS） 邻近路由选择是指在路由过程中，下一跳的选择并不仅仅根据哪个邻居结点在逻辑上（比如 nodeID）离目的地最近，还依据邻居结点的时延信息。综合上述两方面的考虑，选择一个在逻辑上离目的地更近，并且时延相对较小的"折中"结点。

(2) 邻近邻居选择(proximity neighbor selection,PNS) 邻近邻居选择是指在结点创建、更新路由表时,即维护了一定数目的"邻近"邻居,这里的"邻近"是指物理上邻近、时延很小。在著名的结构化 P2P 网络 Pastry 中,每个结点维护三张表——路由表、叶集和邻居集,前两个表是路由的主要依据,第三张表邻居集则维护了一些与当前结点时延很短的结点,用来辅助路由。所以 Pastry 是采用"邻近邻居选择"方法来提高一致性的典型。

(3) 邻近 ID 选择(proximity identifier selection,PIS) 邻近 ID 选择也常称为地理规划(geographica layout)。通常的 P2P 分布式散列表都将结点映射到与其自身属性无关的 ID,"地理规划"方法反其道而行,试图让映射到的 ID 能反映结点的地理信息或物理网位置信息。最先使用"地理规划"方法的是 CAN 的改进版本——"有拓扑意识的 CAN"(Topologically-aware CAN),它能有效地减少 CAN 的路由时延。有研究者指出:因特网时延可以被合理地假设成一个 d 维的几何空间($d \geqslant 2$),这对 CAN 刚好适用。但问题是:绝大多数 P2P 系统的 ID 空间都是一维的,能不能在采用一维 ID 空间的情况下实现"地理规划"方法?下面要讲的希尔伯特编码方法为此做出了努力。

7.5.2 希尔伯特编码——邻近 ID 选择

希尔伯特编码(Hilbert coding)(文献[Zhou et al.,2003a;2003b])是指:用一条曲线无交叉地穿过一个多维空间的所有单元格,沿着曲线走向,从小到大给每个格子编码,每个大格子里的小格子编码是紧邻的。这条曲线称为"希尔伯特曲线"。图 7.5.1 是一个简单的例子。

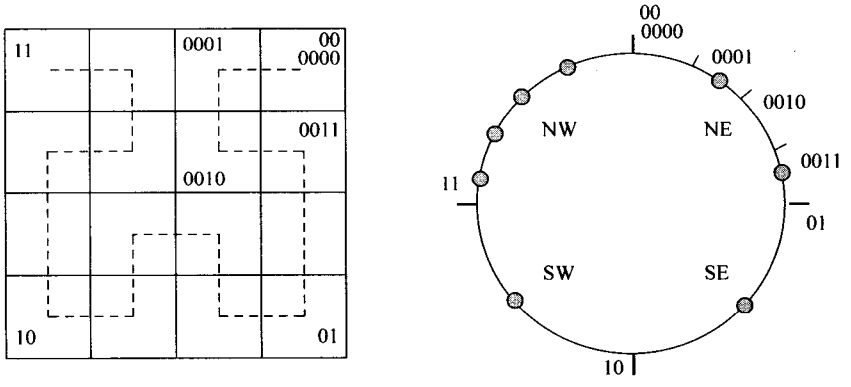


图 7.5.1 左图为穿过 2 维空间的希尔伯特曲线;右图为用希尔伯特曲线贯穿左图形成的数值环

(图片来自[Zhou et al.,2003])

图 7.5.1 的左图可以看成是一张地图(或者一张 IP 地址分布图)，使用希尔伯特曲线对它编码，这样的编码方法有如下性质：

(1) 如果两个结点编码值(nodeID)相差 1，它们在地理上一定是紧密相邻的。

(2) 两个结点的编码值相差越小，即它们的 nodeID 前缀比特匹配越多，那么它们在物理上越邻近。注意：反之不然。

(3) 如果要从编码值完整地反映出结点的邻近关系，必须有一个算法对编码值反运算，从而得到结点的地理位置。

(4) 希尔伯特编码是可调节的：编码越长则划分越细，反映出的位置信息越多。

图 5.7.1 的右图将左图中希尔伯特曲线的首尾相连构成一个环，它相当于一个不使用 SHA 而使用希尔伯特编码来作为散列函数的环结构。希尔伯特编码与 SHA 相比虽然不安全，但是它能保证在环上邻近的结点在地理上也是邻近的，从而使得覆盖网与物理网之间保持较高的一致性。

均衡希尔伯特编码则是指：虽然地理位置是均衡分布的，但是网络结点(或者说 IP 地址)在地理上的分布却是不均衡的，因此需要人为地划分地理区域，使网络结点分布越密的区域其区域编码越短，这类似于“霍夫曼编码”：概率越大，编码越短。均衡希尔伯特编码的优点是：让包含同样多结点的区域，其区域编码等长，因此结点 ID 也是等长的。图 7.5.2 是均衡希尔伯特编码的一个例子。

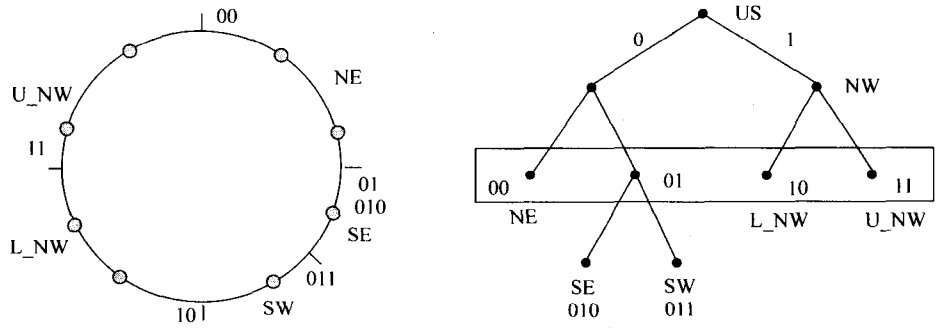


图 7.5.2 左图为反映区域结点密度的均衡编码环；右图为区域按结点密度均衡分割后的编码(图片来自[Zhou et al.,2003])

7.5.3 CFS 的下一跳选择——邻近路由选择

为了减少定位时延，CFS 中在其 Chord 层加入了邻近路由选择，即定位过程中选择下一跳时，尽量选择那些时延小、在底层物理网中靠近的结点。以 CFS 的前驱寻找函数为例，假设某一步结点 m 返回给结点 n 它的前驱列表，列表中包含

每个前驱到 m 的时延。基于这些时延，n 对列表中每个结点进行评价，选取它认为最合理的作为下一跳。设 $C(n_i)$ 表示 n 取 n_i 作为下一跳的预计开销，CFS 采用如下公式评价 $C(n_i)$：

$$C(n_i) = d_i + d' \times H(n_i)$$

$$H(n_i) = \text{ones}((n_i - \text{id}) \gg (160 - \log N))$$

其中，d_i 表示结点 m 在前驱列表中告诉 n 的它到 n_i 的时延，d' 表示 n 所取的每跳平均时延（通常基于 n 过去的经验计算），$H(n_i)$ 表示 n 取 n_i 作为下一跳后估计剩余的 Chord 跳数，N 表示 n 对 Chord 网总结点数的估计（通常基于 n 的邻近结点密度），因此 $\log N$ 是当前 Chord 网中真正决定跳数的高比特位数，$\gg$ 为位运算右移符号，结点 ID 使用 160 位散列值，ones(x) 函数计算 x 中有多少位为 1，因此 $(n_i - \text{id})$ 被右移 $(160 - \log N)$ 位后，ones() 函数实际上计算了 n_i 到 id 间近似的跳数 $H(n_i)$。

计算出 $H(n_i)$ 后，将 $H(n_i)$ 乘以每跳平均时延 d'，即得到取 n_i 作为下一跳的估计剩余时延，加上到 n_i 本身的时延，就是取 n_i 所要的总时延的预计值 $C(n_i)$。这里我们假设 n 到 id 的定位时延，大致等于 n 到 n_i 的时延加上 n_i 到 id 的定位时延，即所谓的“时延传递性”。通常情况下，因特网时延是近似符合此假设的，因此 CFS 采用邻近路由选择下一跳后，一般能有效地降低时延。

7.5.4 Pastry 的路由表构造——邻近邻居选择

本书 4.4.4 节中讲到过 Pastry 的“局部性”。实际上，“拓扑意识和一致性”在某些情况下可以说成“局部性”，因为如果 P2P 路由能局限在某个区域内，物理网和覆盖网之间的差别就不会太大，也就具有一定程度的“一致性”。Pastry 的局部性是通过“邻近邻居选择”来做到的。

4.4 节中讲过 Pastry 路由表的构造：新结点 X 从定位路径上第 i 跳结点 B_i 那里获得自己路由表的第 i 行，直到 nodeID 与 X 最接近的结点 Z 为止（最初 X 是通过“自举”结点 A 发送定位消息的）。借用数学归纳法的思想，我们假设 X 加入前，每次为新结点构造路由表时都做了“邻近邻居选择”，那么在三角关系近似满足的情况下（三角关系可以说成：如果 A，B 很近，B，C 很近，那么 A，C 也很近）可以推出：当 X 从 A 中复制第 0 行作为自己的路由表第 0 行、从 A 中复制邻居集作为自己的邻居集、并用邻居集修正过第 0 行后，X 的路由表第 0 行结点跟 X 在物理上都是邻近的。

随着跳数的增加，三角关系越来越不成立，“邻近邻居选择”的效果也越发减弱。从图 7.5.3 可以看到：X 离其路由表中第 1 行结点会比较远，离第 2 行结点更远，……。

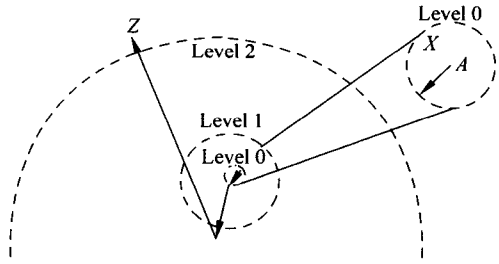


图 7.5.3 Pastry 的“邻近邻居选择”。X 为新加入结点，A 为其自举结点，Level 表示路由表行数或路由跳数，圆弧和半径表示物理网距离或时延

（图片来自[Pastry,2001]）

7.5.5 LTM——动态邻近邻居选择

上面几小节讲解了处理一致性问题的三种传统方法及其典型例子，这一小节要讲的“LTM 技术”(location-aware topology matching technique，位置意识的拓扑匹配技术)[Liuyl et al.,2004; 2005]，在维持原查询范围和减少查询回复时间的前提下，通过主动断开低效连接和冗余连接并选择物理上更接近的结点作为覆盖网邻居，来缓解无结构 P2P 覆盖网与其底层物理网之间的不一致性，尽量减少不必要的通信开销。因此总体上看，LTM 属于邻近邻居选择，但与 7.5.4 节所讲的 Pastry 邻近邻居选择不同的是，LTM 对邻居的选择是动态的，只要结点工作这种选择就一直会进行，而并非只发生在构造路由表时。

1. 无结构 P2P 网络低效的三个原因

文献[Liuyl et al.,2005]分析了无结构 P2P 网络低效的三个原因。首先，在无结构 P2P 覆盖网中，结点选择其覆盖网邻居通常不考虑底层物理网的因素，所以覆盖网上的消息路由常常在物理网上走过太多重复甚至完全不必要的路径，这既导致通信开销增加，又导致通信时延变长。其次，在无结构 P2P 网络中，同样的查询消息经常因为洪泛路由的原因，沿多条路径到达同样的目的地，实际上这些路径中只有一条是有意义的。最后，对两个邻居结点而言，假设它们已经收到了同样的查询 Q 但还没来得及发给对方，在它们收到来自对方的查询 Q 前，必定已经互发了 Q，而互发的 Q 明显是多余的。LTM 的设计动机，正出于对上述三个原因的缓解。

2. LTM 使用的三种主要操作

LTM 的第一种操作，是让每个结点 S 周期性地洪泛广播一种“两跳探测消

息"(TTL2-Detector)。S 在此探测消息中填入自己的 IP 地址和发送时间戳 T_2 并设定 TTL=2；当 S 的邻居 N_i 收到该消息时，N_i 记录 T_2 从而计算出 $d(S,N_i)$(d 表示时延，并假设时延是近似对称的)，修改探测消息的 TTL=1，并在其后附加上 N_i 的 IP 地址和发送时间戳 T_1；当 N_i 的邻居 P(也就是 S 的"两跳邻居")收到探测消息时，P 根据 T_1 计算出 $d(N_i,P)$，并根据 T_1-T_2 计算出 $d(S,N_i)$。这里要注意的关键问题是：结点 S 的两跳邻居 P 既可能同时是 S 的直接邻居，也可能同时与 S 的多个邻居相邻(见图 7.5.4)，因此 P 很可能计算出到 S 的多重时延；LTM 正是通过这种多重关系比较出覆盖网连接的合理性，从而主动断开低效连接和冗余连接，建立高效连接。

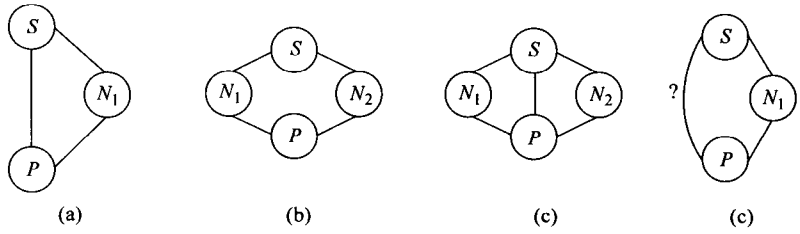


图 7.5.4 S 与其邻居 N_1，N_2 及其两跳邻居 P 的四种关系
(图片来自[Liuyl et al.,2005])

LTM 的第二种操作，是针对图 7.5.4 所示的前三种情形来主动断开低效和冗余连接。对图 7.5.4(a)的情形，在 SP、SN1、PN1 三条连接中，如果 SP 或 PN1 时延最大，就考虑断开 SP 或 PN1("考虑断开"并不是真的断开，而是放入一个"will-cut list"中以后再做决定)；如果 SN1 时延最大，那么 S 不用做什么，但此时对 N_1 的探测而言，N_1 将考虑断开 SN1。对(b)的情形，在四条连接中，如果 PN1 或 PN2 时延最大，就考虑断开 PN1 或 PN2；如果 SN1 或 SN2 时延最大，那么 S 不用做什么，但此时对 N_1 或 N_2 的探测而言，它们将考虑断开 SN1 或 SN2。对(c)的情形，S 将选取 N_1 或 N_2 的一侧转换为情形(a)来处理。

LTM 的第三种操作，是针对图 7.5.4 所示的第四种情形(d)：P 没有收到直接来自 S 的探测消息，并且 P 仅通过 S 的一个邻居 N_1 收到探测消息。此时，P 首先检查"will-cut list"来获取它到 S 的时延，如果表中没有 S，P 就主动探测它到 S 的时延 PS，如果发现 PS 的时延比 PN1、SN1 大，则断开 PS，否则主动创建 PS 连接。

从 LTM 的三种主要操作可以看出，LTM 并未减小无结构 P2P 网络的洪泛查询范围，但通过主动断开低效连接和冗余连接、主动建立高效连接，一定程度上缓解了覆盖网与其底层物理网之间的不一致性，从局部性的角度看确实减少了重复的通信开销。然而应该注意到的是，虽然 LTM 本身的开销较小，但频繁断开和新

建连接，也同时加剧了 P2P 网络的动态性。

7.6 匿名、声誉和信任

在 P2P 网络中，“匿名”与“声誉”，“信任”之间，是互相依存的矛盾。

“匿名”并不是 P2P 最初设计的目的，而是 P2P 的一个附带产物。匿名的根源可以追溯到“安全散列函数”，这在 6.2 节有详述。“存在即合理”，不管在道义上匿名是对还是错，但它作为一种工具的力量是不可否认的，至少它保护了发布者的重要信息和隐私，虽然同时可能带来太多不负责任的行为。

更细致地说，网络安全领域的研究者将一个系统的匿名性分为几种，如发送者匿名、接收者匿名、文件标识匿名、关系匿名等。著名的匿名 P2P 网络 Freenet 提供了很好的发送者匿名和接收者匿名，但在基础的 Freenet 方案里，文件标识匿名是不可能的，因为 Freenet 依靠文件标识来路由所有的消息。

“声誉”可以看成对“匿名”的一种克制。好比在人类社会中，“声誉”既是鼓励人奋斗的动力，又是约束人行为的准则；在 P2P 网络中引入声誉机制，既鼓励了 P2P 用户为网络贡献一部分能力从而获得有利的声誉，又约束了他不合法的行为。

P2P 网络中的“信任”常常是基于“声誉”的，或者至少是一个用户对另一个用户的某种行为判定或衡量。在很多地方，对这两个概念我们不加区分。

本章引言中说到：“匿名”、“信任”之所以未成为 P2P 研究的主流，是因为还有更多的 P2P 的基础性问题没有得到解决；但这不等于说关于声誉、信任的研究目前并不需要，或者没有意义。实际上，早在混合式 P2P 网络 BitTorrent 中，声誉（信任）机制就已经存在了，只是那被称为“阻塞算法”。近来，有越来越多的研究者开始将注意力转移到“声誉”、“信任”的研究上来，这是一个可喜的趋势。

7.6.1 匿名的各种方法

发表在 CCS'02 会议上的论文[Freedman & Morris，2002]分析了威胁匿名性的三种实体，并概括了实现匿名性的三种方法，下面引述他们的结论。

对于绝大多数网络实体来说，少有人会关心其他实体的信息，尤其是重要信息和隐私，因为他们既没有兴趣也没有能力来获取；但有三种实体却有这样的兴趣和能力，从而构成了对匿名性的三种威胁：

（1）好奇的个人或者群体，他们可能会加入网络以监听通信；

（2）某些组织可能会通过网络进入其他人的计算机；

（3）那些更大的、很有能力的组织可能会监控 Internet 主干。

针对这些威胁，计算机安全领域出现了大致三种实现匿名性的方法：

（1）最简单的集中式方法。用户通过匿名代理（如 Anonymizer. com）发送消

息给服务器，由此代理隐藏用户信息（如 IP 地址）。这种方式的缺点在于：一旦匿名代理泄漏了用户信息，或者某个黑客能获取代理中的信息，则整个系统的匿名机制就没有意义了。此外，匿名代理本身成为系统瓶颈，如果受到 DoS（服务拒绝）攻击，则整个系统将瘫痪。

（2）“mix-net”（混合中继网）方法。在这种方法中，用户通过一组“混合中继”（mix relays）结点连接到服务器，代表性的系统有 The Anonymous Remailer System、Onion Routing 和 Zero-knowledge's Freedom，它们通过一组为数不多的、固定的核心中继结点来提供匿名性。“mix-net”方法比集中式方法要好得多，但是它仍然有不少缺陷：比如假定有一个核心中继结点是恶意的，那么只要它收到一条来自非核心中继结点的消息，它就可以立刻判断出此消息的来源；此外，那些共谋的入口和出口中继结点可以使用“计时分析”（timing analysis）来找到消息的源和目的地。

（3）使用随机中继的方法，即中继结点是随机、不确定的。第 3 章中讲过的 Freenet 和下文将要讲述的 Tarzan，都属于这种情况。Freenet、Tarzan 将所有结点组织成一个无结构 P2P 网络，每个结点都可以作为中继。下面我们讲解 Tarzan 实现匿名性的方法。

7.6.2 Tarzan——P2P 匿名网络层

“Tarzan”在英文中的意思为“泰山”（记得电影《人猿泰山》吗?），Freedman 和 Morris 在其论文中并未讲过这个名称的缘由，我们猜测这表示 Tarzan 系统像人猿“泰山”一样灵活、健壮。

Tarzan 是一个 P2P 的匿名 IP 网络覆盖，它通过数据多层加密和消息多跳路由来实现匿名性。Tarzan 将 7.6.1 节中讲到的“mix-net”混合中继网方法扩展到 P2P 环境中，结点之间通过中继结点序列（这一序列结点构成一条“隧道”）来通信，任何一个 Peer 都有可能成为中继结点。同时，Tazan 使用了一种可扩展和实用的技术来进行通信量掩盖以对抗通信量分析。

匿名性分很多种，Tarzan 除了提供“发送者匿名性”和“接收者匿名性”外，还提供了较弱的“关系匿名性”（relationship anonymity）：一对结点之间互相通信的关系不会被其他结点发现。

Tarzan 系统涉及的技术非常多，这里我们只介绍最基础的网络模型和体系架构，而技术细节、分析和实验，读者可参考论文[Freedman & Morris, 2002]；Freedman 等人还在同一年的 IPTPS'02 会议上发表了一篇精简版的 Tarzan 论文[Freedman et al., 2002]，它更短也更易懂。

1. Tarzan 网络模型

Tarzan 结点之间通过一系列结点构成的“隧道”(tunnel)来通信，如果这条“隧道”中有一些结点是恶意的，那么 Tarzan 所设想的路由匿名性就不能实现。

还有一种情况也是对 Tarzan 非常不利的，就是某台计算机可能拥有多个 IP 地址，从而虚拟地操纵多个 Tarzan 结点。为了对抗上述问题，Tarzan 设计者从一个简单的观察结果(一台计算机通常只能控制一个连续区间内的 IP 地址段，典型地是某个局域网中的路由器)出发，定义了“域”(domain)概念，以此标识被某个恶意结点所控制的子网。

一个域到底应该包括多少个 IP 地址？也就是说域的划分粒度多大才合适？根据经验，一个局域网的 IP 掩码通常为“255.255.255.0”或者“255.255.0.0”，也就是说网络地址通常占有 IP 地址的前面 24 位或 16 位，所以 Tarzan 的域划分首先使用 16 位掩码，然后使用 24 位掩码。图 7.6.1 是一个简单的 Tarzan 网络模型。

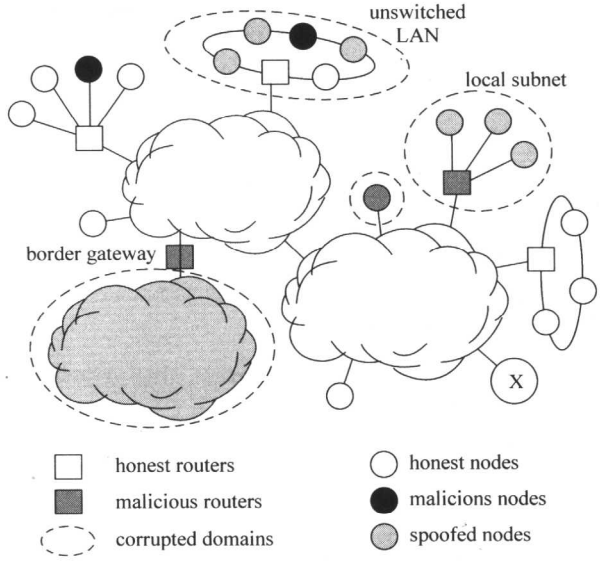


图 7.6.1 Tarzan 网络模型。可以清楚地看到：恶意的路由器控制了整个域(子网)，而一般的恶意结点则不能控制整个域，但它能监听域内其他结点的通信
(图片来自[Freedman & Morris,2002])

2. Tarzan 体系架构

我们以一次典型的 Tarzan 消息传送为例来讲解 Tarzan 体系架构。Tarzan 结点为了匿名发送消息，它首先选择一组结点来构成一条穿过 Tarzan 覆盖网的路

径；然后使用这组结点来创建一条“隧道”，这一步包含会话密钥的分配；最后，它通过这条“隧道”来发送消息。“隧道”的出口点(exit point)是一个 NAT(network address translator，网络地址转换器)，它将匿名的消息传给目的服务器。这些服务器并没有意识到 Tarzan 的存在，它们将回复消息传给 NAT，NAT 沿着“隧道”将回复消息反向路由到出发点。

图 7.6.2 中，最左边的大框表示一个 Tarzan 结点，该结点中的应用程序(client app)想要发送一条 IP 消息(src：App，dst：Dest)，系统内核将此消息转交(kernel divert)给“隧道创建器”(initiator)，隧道创建器一方面为消息传送构建“隧道”，另一方面将消息的源地址(App)转换成一个私有地址(Priv)，并将消息多层加密后封装。封装后的消息以 UDP(用户数据报)的方式发送给隧道中的第一个中继结点，此中继结点基于消息包中的流标号来解密出第一层消息，并将第一层消息发给第二个中继结点，依次做下去，原来多层加密封装的消息被一层层解密。上文已说过，“隧道”的出口点是一个 NAT(图中的 PNAT)，此 PNAT 解密出消息的最后一层，从中提取出源消息(src：Priv，dst：Dest)，将此消息的源地址(Priv)转换为 PNAT 的公开地址(PNAT)，最后发送给目的服务器。服务器回复消息反向传输的过程在上文已经说过，读者也很容易想象出。

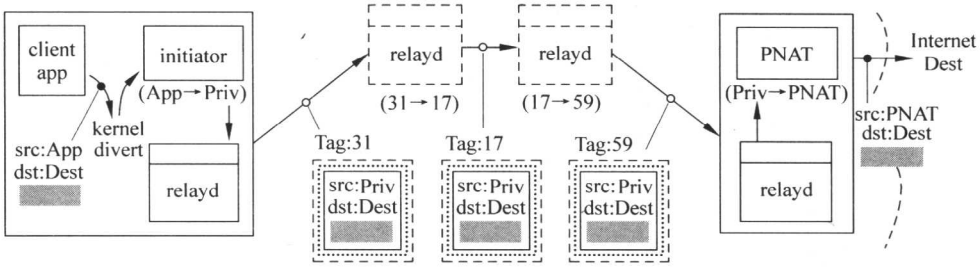


图 7.6.2 Tarzan 体系架构图(以一次消息发送为例)
(图片来自[Freedman & Morris.，2002])

从理论上说，Tarzan 体系复杂、完备、细致，具有高度的匿名性；但是如果真要实现 Tarzan 这样一个 P2P 匿名网络层，却是一项极其困难而浩大的工程。

7.6.3 P2P 声誉、信任涉及的问题和解决方法

在计算机领域，声誉管理一直是受人重视的。一个典型的例子是电子交易平台 eBay(相信大多数人对 eBay 并不陌生)。在 eBay 的声誉系统中，买者和卖者在每次交易后可以互相评价，用户的声誉是他最近 6 个月来所获得评价的和。eBay 的声誉机制很简单，并且是基于服务器的，所以对于 P2P 这样的分布式系统来说，并不适合。

Kamvar 等人在论文[Kamvar et al.,2003]中提出了设计 P2P 声誉系统所要考虑的 5 个问题，另外，Aberer and Despotovic 在论文[Aberer & Despotovic,2001]中也提出了 3 个涉及 P2P 信任管理系统的问题。下面我们总结一下设计 P2P 声誉、信任系统所涉及的问题：

(1) 此系统必须是自管辖的(self-policing)。也就是说系统本身为其用户定义了公有的行为准则和声誉/信任衡量，即使在没有集中式认证或权威第三方的情况下，系统用户也能总体上遵循并加强这些准则。

(2) 此系统必须是匿名的。一个用户的声誉应该同一个不透明的 ID 相关联(如 P2P 系统中通过分布式散列表生成的 nodeID)，而不是像 IP 地址这样的显式标识。

(3) 此系统不应该给予新来者任何额外的利益。用户的声誉必须通过他在多次事务中所表现出来的行为来衡量，恶意结点无法通过更换 ID 成为新来者而获得声誉。

(4) 此系统应该尽量最小化声誉/信任机制带来的额外开销，这包括通信、存储、计算、体系架构等多方面的负担。

(5) 此系统应该对恶意结点有较强的容错性。恶意结点单方面的行为没有办法不合理地改变他自己或者其他人的声誉。

从 2001 年 P2P 概念被正式提出，关于 P2P 声誉/信任机制的研究就一直在进行，并且近来有成为领域热点的趋势，这反映了 P2P 系统逐渐走向成熟。

Philippe Golle 等人在 2001 年的 ACM 电子商务会议上发表的论文[Golle et al.,2001]，构建了 P2P 网络的一个正式的“博弈理论模型”(game theoretic model)，并分析了此模型在多种支付机制(payment mechanism)下用户策略之间的均衡(equilibria)。但是这个模型所对应的声誉机制建立在集中式服务器的基础之上，由中心服务器来管理文件索引(就像 Napster 那样)。

发表在 CIKM'01 会议上的论文[Aberer & Despotovic,2001]讨论了 P2P 系统中信任管理的诸多问题，并设计了一个分布式的 P2P 信任管理方案，这差不多是最早全面分析 P2P 声誉、信任的文献。

发表在 CCS'02 会议上的论文[Damiani et al.,2002]，提出了一种基于声誉的、在 P2P 网络中选择可靠资源的方法。每个 Peer 在下载资源前，通过分布式的投票算法(polling algorithm)来评价资源的可靠性，从而限制恶意资源在 P2P 网络中的传播。Damiani 等人声称他们的方法可以在现存 P2P 协议上直接以捎带确认(piggybacking)的方法实现，所以对原协议的修改非常少。

发表在 2003 年的 ACM 电子商务会议上的论文[Xiong & Liu,2003]，设计了一个服务于 P2P 电子商务社群的、基于声誉的信任模型“PeerTrust”。此模型基于事务回馈(transaction feedback)来量化和比较 Peer 的可信任性(trustworthiness)。

但仅仅考虑回馈并不准确，实际上她们的模型总共考虑了 5 个因素来评价 Peer 的可信任性：①以满意度表示的回馈，由一个 Peer 从其他 Peers 那里通过事务得到；②回馈范围，比如在社群中一个 Peer 和其他 Peers 之间发生的事务总数；③回馈资源的信用度，表示一个 Peer 对另一个 Peer 所做回馈的信任程度；④事务上下文，比如该事务的大小、重要性；⑤社群上下文，比如社群对 Peer 的奖励、惩罚等。她们将这五个因素整合到一个模型 PeerTrust 中，并做了许多实现方面的工作。有关这些因素的具体含义和此模型的细节介绍，读者可参考原论文，这里不作详述。实际上在 2004 年的 *IEEE Transactions on knowledge and Data Engineering* 上，她们又发表了大体相同的论文[Xiong & Liu,2004]，比前一篇更为细致。

在 2.2 节我们讲过 BitTorrent 的阻塞算法，这实际上是一种隐式的声誉方法，它非常简单，但十分有效，唯一的不足是阻塞算法只基于本次下载而不考虑历史行为。

下面将要详细介绍的 EigenTrust 声誉管理，在 P2P 声誉研究方面有很大的影响力，它对以前的工作做了很好的概括和数学总结，而其所设计的 EigenTrust 算法，用分布式的方法去近似获得全局性、历史性的用户声誉，更为准确、合理。

7.6.4 EigenTrust 算法——完备的 P2P 声誉管理

斯坦福大学的 Kamvar 等人在 WWW'03 会议上发表的论文[Kamvar et al., 2003]中，提出了一个显式地管理 P2P 网络中用户声誉的方法：EigenTrust 算法。“EigenTrust”这个名称可以理解成“特征信任”，因为它使用了用户间满意度矩阵的特征向量来计算信任值。

1. 衡量声誉(满意度)的三种方法

在一个分布式环境中，每次事务之后，用户要互相评价，这是声誉最基础、最底层的来源。比如在 P2P 网络中，每次当用户 i 从用户 j 那里下载一个文件后(也可能下载失败)，用户 i 会以一个信任值 $tr(i,j)$ 评价这次事务：如果成功下载到所要的文件，则 $tr(i,j)=1$；如果下载失败或者下载的文件并非所要的，则 $tr(i,j)=-1$。用户 i 很可能从用户 j 那里下载很多文件，这样就需要一个用户 i 对 j 历史性的评价(称为满意度 s)，我们记为 s_{ij}。

满意度的衡量方法有很多种，这里我们介绍常用的三种。第一种方法最简单，基于上段所讲的信任值 $tr(i,j)$，直接令 $s_{ij}=\sum tr(i,j)$。第二种方法与前者本质上等价，它分别记录用户 i 对用户 j 满意的事务数 $\mathrm{sat}(i,j)$ 和不满意的事务数 $\mathrm{unsat}(i,j)$，那么 $s_{ij}=\mathrm{sat}(i,j)-\mathrm{unsat}(i,j)$。前两种方法固然简单易行，却不准确，因为这里的声誉衡量太过局限、偶然，而不是来自一个大的范围中多个用户的

评价；另一方面，如果全局性地收集 s_{ij} 再求和，则能获得准确的声誉信息，但造成的网络通信开销又非常大，也不合理。第三种方法正是 EigenTrust 采用的“传递信任值”(transitive trust)方法，它以一种自然的、低开销的方式收集网络中所有用户的信任值，用户 i 信任那些给他提供正确下载的用户，所以也信任这些用户所提供的信任值，故称“传递信任”。EigenTrust 算法分布式地计算传递信任值，其实验结果声称：即使网络中 70%的结点是恶意的，EigenTrust 也能有效地减少不满意的下载次数。

2. EigenTrust 收集、计算信任值的方法

EigenTrust 不是直接使用上面所说的满意度 s_{ij}，而是将它们规范化(normalize)后使用，规范化的优点在于可以有效地避免恶意结点给予其他结点太高或太低的评价。规范化后的信任值记为 c_{ij}，EigenTrust 采用了一种很简单的方法做规范化：$c_{ij}=\dfrac{\max(s_{ij},0)}{\sum_j \max(s_{ij},0)}$。

EigenTrust“传递”式地收集信任值，用户 i 第一步所做的就是去问其邻居对于其他用户的看法。当然，对于每个邻居发给用户 i 的信任信息，用户 i 不是同等对待的，i 自己越信任哪个邻居，他给这个邻居发过来的信任向量赋予越大的权重，也就是：

$$t_{ik}=\sum_j c_{ij}c_{jk}$$

其中，t_{ik} 表示用户 i 对用户 k 的信任，它是通过用户 i 询问邻居得到的，邻居 j 发过来的 j 对 k 的信任 c_{jk} 被乘上了用户 i 自己对邻居 j 的信任 c_{ij}(这就是权重)。上式可以写成线性代数中矩阵和向量的形式，如果我们定义 $\boldsymbol{C}$ 为矩阵 $[c_{ij}]$，$\boldsymbol{t}_i$ 表示包含 t_{ik} 的向量(注意 k 是变量)，那么 EigenTrust 收集、计算信任值的第一步可以表示为下式：

$$\boldsymbol{t}_i=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{c}_i$$

其中，$\boldsymbol{C}^{\mathrm{T}}$ 表示矩阵 $\boldsymbol{C}$ 的转置，而 $\boldsymbol{c}_i$ 表示包含 c_{ij} 的向量(注意 j 是变量)。很明显地，$\boldsymbol{t}_i$ 仅仅反映了用户 i 和它邻居的经验，为了得到更广泛的评价，用户 i 第二步要去询问其邻居的邻居，此时 $\boldsymbol{t}_i'=(\boldsymbol{C}^{\mathrm{T}})^2\boldsymbol{c}_i$，这相当于“传递”了一次信任值。同样，用户可以继续上述过程，再往前多走一跳、两跳，而 $\boldsymbol{t}_i^{(n-1)}=(\boldsymbol{C}^{\mathrm{T}})^n\boldsymbol{c}_i$，每多做一步，用户 i 得到的信任值就越准确，当然所需要的网络开销和计算量也相应增加了。

既然步数越多、n 越大，信任值越准确，而同时收集、计算的负担也越大，那么如何才能计算出一个相当近似的信任值向量 $\boldsymbol{t}_i$ 来呢？这个问题的答案，正是 EigenTrust 算法名称的由来：可以证明，当 n 很大时，每个用户 i 的信任值向量 $\boldsymbol{t}_i$ 都将趋向于矩阵 $\boldsymbol{C}$ 的“左主特征向量”(left principal eigenvector)$\boldsymbol{e}$，换句话说，在

EigenTrust 模型中 t 是一个全局特征向量，它的每个元素 t_j 代表了整个系统赋予用户 j 的信任值。下面是一个简单、非分布式的 EigenTrust 算法：

$$
\begin{aligned}
&t^{(0)} = e;\\
&\text{repeat}\\
&\quad t^{(k+1)} = C^{\mathrm{T}} t^{(k)};\\
&\quad \delta = \| t^{(k+1)} - t^{k} \|;\\
&\text{until } \delta < \varepsilon;
\end{aligned}
$$

实际的分布式 EigenTrust 算法比上面的要复杂很多，这里不做进一步讲述，读者可参考论文[Kamvar et al.,2003]。

3. 安全的 EigenTrust 算法

上面所讲的 EigenTrust 算法是不安全的，恶意用户可以发布错误的信任值来破坏整个系统，所以 EigenTrust 设计了两个基本的安全机制：①一个用户当前的信任值不应该由他自己计算，也不应该放在他自己那里，而是放在他的“得分管理者”(score manager)那里，并由此管理者计算。管理者的分配使用了分布式散列表，下面会讲到。②一个用户的“得分管理者”也不应该唯一，因为“得分管理者”也是不可信的，所以 EigenTrust 使用多散列方法来为一个用户分配多个得分管理者。

在 EigenTrust 中，用户的“得分管理者”是通过 CAN 散列函数来分配的，分配可表示为如下映射：

$$\text{用户 ID(如 IP 地址、端口号)} \xrightarrow{H_1, H_2, \cdots} \text{CAN 空间上多个点}$$

其中每个散列函数将用户映射到一个点。图 7.6.3 是一次简单的“得分管理者”分配的例子。

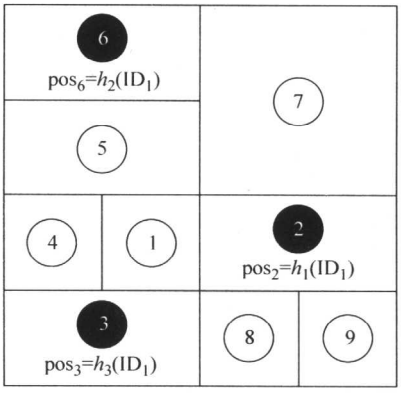


图 7.6.3　EigenTrust 使用二维 CAN 来做“得分管理者”分配

(图片来自[Kamvar et al.,2003])

图 7.6.3 中，用户 1 的标识 ID_1 被三个散列函数 h_1，h_2，h_3 分别映射到用户 2，3，6 所负责空间上的点，因此，用户 2，3，6 都成为用户 1 的“得分管理器”。

EigenTrust 使用分布式、低开销的方法合理地计算出 P2P 网络中用户之间的信任值，并且此信任是历史、全局地衡量的，所以 EigenTrust 应该是一个完备、实用的 P2P 声誉、信任方案。

7.7 P2P 安全问题

“安全问题”在无论哪个领域中都是复杂、头疼、难以两全的问题，“安全机制”就好比学生宿舍中的管理员，他的存在能让宿舍更安全，但是总要你登记又让你觉得很烦。P2P 领域最初涉及、且持续至今的最大安全问题是“版权问题”，这个问题在计算机其他领域并不存在，因为它不只是计算机本身的问题，而是 P2P 独有的与公司利益、知识产权之间的纠纷。除了版权问题，因为 P2P 本身是分布式系统的一个极端情况，所以分布式系统领域的绝大多数安全问题，在 P2P 网络中都存在，并且常常表现得更严重。许多 P2P 安全问题，到目前为止都是开放性的。

从三代 P2P 网络的角度来看，第一代混合式 P2P 网络的安全性是最容易保证的，因为它含有中心服务器，也就有了一个权威的认证机构，从而 C/S 模式下很多传统的安全方案都能应用到混合式 P2P 网络上来。第二代无结构 P2P 网络的安全性相比而言要困难得多，因为所有用户平等，没有权威的第三方存在，不过由于无结构网络组织松散，没有严格的“结构”，所以暴露给攻击者的信息也很少，增加了攻击者的难度。7.6 节中讲“匿名”时提到的 Freenet，实际上已经是一个非常安全的无结构 P2P 网络。第三代结构化 P2P 网络的安全性最为困难，因为网络结构越严格，暴露给攻击者的信息就越多，也就越容易受攻击。现有系统都实现了某个方面的安全，但通常是很不够的，这个问题注定需要 P2P 领域研究者付出大量的努力。

7.7.1 P2P 网络中的攻击方式和安全对策

是凡网络安全，没有普适的方法，必定针对特定的网络攻击设计特定的安全防卫机制，也就是“兵来将挡，水来土掩。”威胁 P2P 网络安全的攻击方式及其对应防卫策略可分以下几类：

1. 监听、截获

1.1 目的：监听网络通信，截获数据包，破坏数据机密性。

1.2 特点：被动攻击方式，难以检测，但易于防止。

1.3 策略：数据加密（对称加密如 DES、公开加密如 RSA）。

1.4 应用：如果采取对称加密(symmetic encryption)，难点在于密钥的分配，这在纯分布式的系统中要更难，因为缺乏可信的权威第三方。所以有两种选择存在：一是引入可信第三方，如 Kerberos 机制(应用层)、IP Sec 的 ESP 协议(封装安全有效载荷)(IP 层)，实际上基于 Pastry 协议的 P2P 应用系统——PAST 所采用的"智能卡"安全机制就是依赖于集中式的权威第三方的；二是采用类似电子邮件的 PGP 协议(pretty good protocol)，这是一个纯分布式的协议，以公开密钥机制来传递对称密钥。PGP 协议非常适合于 P2P 应用，是目前为止所知的非常适应分布式体系结构的安全方案。

如果采取公开加密(public encryption)，可用的方法也很多，比如最经典的 RSA 公开加密算法。唯一的不足在于公开加密巨大的计算开销。其实对 P2P 而言，最合理的方法是上面所说的、以公开加密机制传递对称密钥的 PGP 混合式方案。

2. 中断

2.1 目的：阻断网络通信。

2.2 特点：主动攻击，易于检测，难于防止。

2.3 策略：本地观察(通过 Ping 命令就可以检测出)。

2.4 应用：要在无结构 P2P 网络上阻断特定结点之间的数据包是极其困难的，因为攻击者无法也不可能得知数据包的路由信息。但是在结构化 P2P 网络上，由于系统高度的结构化和确定性的路由方式，攻击者可以主动地采取某种方式，在特定结点之间的路由路径上占据重要位置，从而截断两者之间的通信。只要 P2P 网络是高度结构化、确定性的，中断问题就很难解决；在不少结构化 P2P 系统如 Tapestry、Pastry 中，消息路由的下一跳都有一定的随机性，这能在很大程度上遏制中断攻击。

3. 篡改

3.1 目的：修改数据包，破坏数据完整性。

3.2 特点：主动攻击，易于检测，难于防止。

3.3 策略：报文鉴别 MAC、散列函数、数字签名。

3.4 应用：保护数据的完整性相对简单，只要在数据报后附上其鉴别码即可。最常用的鉴别码是原数据报的散列值，SHA-1 几乎成为所有现存 P2P 系统的选择，但是随着 MD-5 和 SHA-1 被攻破，散列方法的安全性已经受到质疑。MAC 报文鉴别码的破解难度比散列方法要大得多，但是它需要双方共享一个密值，密值的安全分配对 P2P 网络来说又是一个难以解决的问题。

数字签名(digital signature)方案用来保护数据安全性太过浪费(中国有句古

话："杀鸡何用牛刀?")。首先，无论 RSA 还是 DSA 数字签名，都需要相当大的计算开销。即使容忍这一点，公钥的认证也是一个大问题，如果采用 X.509 标准的 CA 认证，固然完备，但是又引入了可信的权威第三方，破坏了 P2P 的分布式本质；如果采用 PGP 协议的信任链方式分配公钥，首先此方式本身有一定风险，其次它暴露了足够的缺陷来受"女巫攻击"[Douceur，2002]和"拜占庭攻击"[Lamport et al.，1982]的威胁。

总体而言，防止数据篡改，还是散列方法最为可取。虽然 MD-5 和 SHA-1 被攻破，但是应该很快会发布新的标准，而且目前要在现实上攻破 SHA-1 还是很难的。下文中讲到 OceanStore 中数据分块、层次化的完整性验证，是防止篡改的更严格方法，当然实现起来也更复杂。

4. 伪造

4.1 目的：构造虚假数据包，攻击网络缺陷。

4.2 特点：主动攻击，检测的难度视攻击而定，难于防止。

4.3 策略：视特定攻击而定。

4.4 应用：伪造的形式有多种：

(1) DoS(服务拒绝)+IP 欺骗：攻击者将它收到的所有请求发给被攻击者，使得被攻击者通信阻塞无法提供新的服务，从而攻击者伪造被攻击者的 IP 数据报，进行 IP 欺骗。DoS 是一种传统、普遍的网络攻击，应对它的方法有很多，大多数 P2P 协议在设计时都有这方面的考虑(下文将讲到)。

(2) ID 伪造：在没有可信第三方认证的情况下，P2P 网络一般采用散列方法产生结点 ID，无论是 Hash(IP)、Hash(公钥)还是其他方法，都难以避免恶意结点的 ID 伪造。ID 伪造有两种：一是"女巫攻击"[Douceur，2002]，即一个结点伪装成多个 ID，操纵、破坏、分割网络；二是"拜占庭攻击"[Lamport et al.，1982]，指多个恶意结点的联合欺骗，这很难做到，但是一旦做到则极具破坏力，且更难检测。下文介绍的"Chord 的结点 ID 认证"是一种应对 ID 伪造的策略。

(3) 源站文件伪造：在纯分布式的 P2P 网络中，当结点对自己提供的文件不需要负任何责任时，它可能提供伪造的文件。下载者终究会知道他被欺骗，但是它对提供者几乎没有任何办法。在 P2P 网络中引入"声誉"/"信任"机制可以有效解决源站点文件伪造问题，这在 7.6 节中有详述。

5. 重放

5.1 目的：将过去截取的数据包沿信道重发，破坏正常通信。

5.2 特点：主动攻击，易于检测，难于防止。

5.3 策略：给消息附加时间戳。

5.4 应用："重放攻击"的目的在于打断通信，制造网络混乱，应对重放攻击最有效的策略是给消息附加时间戳。但是在分布式系统中，各个成员的时间无法一致，所以如果使用物理时钟来加时间戳，必定带来不少问题。分布式系统中经常通过"逻辑时间戳"来同步，这固然很好，但系统的设计要因而复杂很多。一些折中的方案，比如使用 IP Sec 安全协议、在 IP Sec 首部中加入序号以及采用滑动窗口，可以非常有效地抑制重放。

6. 抵赖

6.1 目的：抵赖已发出的消息。

6.2 特点：主动攻击，易于检测，难于防止。

6.3 策略：数字签名。

6.4 应用：对于抵赖一向是采用数字签名，其难点在于公钥的认证。如果采取集中式结构认证公钥，必然引入可信第三方，如 X.509 协议的 CA 认证方式，这种层次化认证结构与采用"超结点"的双层 P2P 网络结构看上去似乎能适应，但并不能确证。如果采取分布式结构，最适合采用的是 PGP 协议，以"信任链"方式认证公钥，上文已说过，PGP 认证本身有一定风险，适合普通的应用场合，一般不能进行商业应用。

上面所说 6 个方面的安全威胁，只是从安全攻击的底层技术角度来看；如果从表现方式来看，P2P 网络中的安全威胁种类就非常多了，而且它们使用的攻击方式通常远不止一种。文献[Sit & Morris, 2002]对 P2P 网络中的安全性做了多方面的考虑，归纳了很多可能的威胁，并总结了 6 条初步的 P2P 安全设计原则：

(1) 定义可验证的系统不变属性并验证它们(如 Chord 中的后继关系)；

(2) 允许查询的发起者观察查询的路由过程；

(3) 以可验证的方式给结点分配 ID；

(4) 路由过程中的"下一跳结点选择"不可滥用；

(5) 使用随机查询来"交叉校验"路由表；

(6) 避免单点负责(single points of responsibility)。

7.7.2 P2P 面临的非技术性问题

1. 版权问题

本书 2.1 节详细地描述了 P2P 的祖先 Napster 陷入版权纠纷后陨落的过程。之所以要把这样一个看似非计算机领域的社会问题讲得如此细致，目的正在于引导人们去思考版权问题：这个 P2P 过去、现在、将来都面对的最大威胁。

就像 Napster 的出现冲击着唱片公司的利益一样，大多数 P2P 服务都将不可

避免地与知识产权发生冲突。Napster 在几年的法律纠纷之后，已经变成了一个普通的音乐收费下载网站，实际上可以说：Napster 从关闭的那天起就已经不存在了。而后来的 P2P 应用，如 BitTorrent、eDonkey、KaZaA，都一直官司不断、争议不休，从来没有一天真正被社会完全接纳过。尽管美国唱片业协会等一些组织在寻找新的方式来保护知识产权，而事实上也确实出现了一些 P2P 应用能确保版权被保护，但这都是远远不够的。每一个提供文件共享服务的 P2P 公司，都不得不认真审视 P2P 网络面临的版权问题。

2. 管理困难

P2P 网络的精髓在于其“乌托邦”式的自由管理方式，这种方式给了用户更多的自由，但是也陷入了“无政府主义”的困境。可以想象，缺乏管理的 P2P 网络将会成为病毒、色情内容以及非法交易的温床。许多 P2P 公司打算通过 P2P 网络开展电子商务，但是付费问题、流量计算、商品价值的验证等都是一时很难克服的困难。

3. 垃圾信息

由于 P2P 网络的用户众多，当某个用户进行搜索时，自然会得到大量的搜索结果，而除了少数有用的信息以外，其他大多数信息可能都属于垃圾信息（使用过 BT 的人应当有此经验）。在缺乏统一管理的情况下，P2P 网络很难对搜索结果进行排序，所以用户常常不可避免地接收到垃圾信息。现在已经有公司尝试着将人工智能技术、专家数据库技术引入 P2P 网络中，希望能够克服垃圾信息的困扰。

虽然作为一个年轻的领域，P2P 存在众多技术上的难题，也面临着不少非技术性的问题，但是，P2P 并不是“先天缺钙”的，作为一个刚刚崭露头角的新技术，P2P 还需要一个发展过程。我们有理由相信：通过后天的不断完善，P2P 技术会越来越完备，并最终成熟起来。

第8章 P2P模拟与仿真

美国史学大师斯塔夫里阿诺斯在其巨著《全球通史》中写到："绝大多数理论是错误的，因为它们的依据是信念而不是理性。各种时髦理论走马灯般换来换去，每一种都反映了当时的见识和偏见。"由此可见理论研究的主观性与片面性，也更强调了做研究应注重理论结合实践，强调实验的作用。对P2P网络来说，由于其规模巨大，实际构建如此大规模的分布式网络带来的硬件、软件开销通常是难以付出的，因此P2P领域侧重"模拟"与"仿真"两个方面。按照国际标准化组织(ISO)的定义，模拟(simulation)是指"选取一个物理的或抽象的系统的某些行为特征，用另一系统来表示它们的过程"，而仿真(emulation)是指"用另一数据处理系统，主要是硬件，来全部或部分地模拟某一数据处理系统，以至于模仿的系统能像被模仿系统一样接受同样的数据，执行同样的程序，获得同样的结果"。由上述定义可以看出，"仿真"的要求要比"模拟"高很多，实现起来的难度也相应大得多。

本章我们从P2P模拟开始，首先提出P2P模拟器的设计准则和意义(8.1节)，并分析为什么P2P模拟器一直不被P2P领域重视的原因。然后回顾几个经典的网络模拟器(NS-2)和拓扑产生器(GT-ITM、BRITE)(8.2节)，并介绍几个有影响力

的 P2P 模拟器 p2psim、3LS、GnutellaSim、PeerSim(8.3 节)。最后介绍著名的全球网络服务仿真平台 PlanetLab(8.4 节)。

8.1 P2P 模拟器的设计意义和准则

8.1.1 P2P 模拟器的设计意义

为什么我们要用将近一章内容来讨论 P2P 模拟器(P2P simulator)？在计算机网络或者分布式系统的研究中，人们往往青睐实际的网络实验，因为它真实、可靠，不含过多的理论假设；然而，这并不是 P2P 实验的正确选择，因为具有大规模、高动态性的实际 P2P 网络构建起来相当困难，并且需要昂贵的开销。所以我们相信，“模拟”(simulation)是 P2P 领域现实可行的系统测试、评价、比较和认证机制。P2P 模拟器的开发为 P2P 领域的研究者带来了实验上的便利，人们只需要将精力集中于那些想要关注的性能，而不必花费大量时间去编写实验工具。

P2P 网络通常在规模上能够扩展到很大，而网络异构性却很高，因为加入 P2P 网络的计算机在带宽、存储容量、计算能力上有巨大的差异。并且，P2P 网络的工作涉及用户计算机、物理网络、应用层覆盖网、P2P 协议和最高层的用户，多层的复杂性导致：如果要在真实环境中实际运行、测试一个 P2P 网络需要付出巨大的努力，通常是不可取的；反而，使用模拟的方法忽略掉一些不必要或不关心的细节，在可控的环境中测试、评价一个 P2P 网络，不仅方便、易实现，且结果常常能更公平(可以基于同一模拟器测试多种 P2P 协议)。

虽然如此，长久以来 P2P 模拟器的开发并未得到这个领域的研究者们应有的重视，[Liu,2003]分析了其中一些原因，我们总结这种不重视的原因如下：

第一，P2P 大规模、动态性的覆盖网络，是其独有的特点，也是过去的网络研究所没有遇到过的，这使得 P2P 模拟器的设计、开发变得十分困难。

第二，现有的经典网络模拟器的存在(最著名的如 NS-2)，使得 P2P 领域的研究者不愿意去花费大量劳动开发专有的 P2P 模拟器，而宁可基于传统的网络模拟器或者对其加以改进以节省工作量。这种表象使得专有 P2P 模拟器的开发看起来并不必要(实际并不是这样，比如 NS-2，它模拟的网络层次很低，最高达到传输层，但 P2P 却是工作在应用层的)。

第三，从 Napster 开始，P2P 就一直是一种“民间的”Internet 应用，P2P 软件的开发者们往往将精力聚集到如何使它迅速地扩展、吸引用户群，而很少有人真正去仔细研究它。

第四，到目前为止，P2P领域的研究者对于大规模分布式应用开发中的模拟工作的效果、目标和过程，都未能达成共识，缺乏一个指引人们去实现的标准。

再多的原因，都不应成为放弃的理由，因为只要是真正有价值的东西，就值得我们去付出努力。后面几节将讲到的p2psim、3LS、PLP2P、Peersim等通用P2P模拟器，是P2P模拟器领域的先驱，它们的出现给了我们设计、开发P2P模拟器的信心。

8.1.2 P2P模拟器的设计准则

在这里，通过分析P2P网络实际的特点、需求，观察传统网络模拟器的设计思想，再结合已有的一些P2P模拟器实现，我们来归纳一下P2P模拟器基本的设计准则：

(1) 简洁易用　简洁、易用是一个模拟器最基本的设计准则。一个复杂、难用的模拟器，即使设计再合理、功能再完备，也很难被人们接受。

(2) 通用性　P2P领域并不缺乏那些着眼于特定应用的模拟器，而是需要一个通用的模拟器，来为各种P2P协议的测试、比较提供一个共同的平台。

(3) 可扩展性　P2P网络一般具有高可扩展性，结点规模既能很小，又能扩大到上千万台计算机，所以一个好的P2P模拟器也应该能模拟巨大规模的网络。

(4) 分层，可组合　由于P2P既工作在网络层(物理网)，又工作在应用层(覆盖网)，所以模拟时应该将不同的层次分开，每层都能提供多个不同组件，它们之间还可组合使用。现有的通用P2P模拟器p2psim、3LS和GnutellaSim都是分层可组合的，说明这个准则完全可以满足。

(5) 能产生各种必要的事件　P2P模拟器至少要能产生P2P网络中最基本的各种事件：路由、查询、结点加入、结点离开或失效。

(6) 合理的时间、空间开销　“时间开销”指的是P2P模拟器运行一次模拟的时间不应太长，尤其是网络规模很大时，模拟时间最好能线形增长。“空间开销”指的是模拟时消耗的存储容量不宜太大，最好所有的信息都放在内存中进行。

(7) 合理的同步机制/无死锁　在分布式系统中，同步常常是重中之重，是整个系统能正常工作的核心，所以P2P模拟器也必须具备合理的同步机制，保证系统不会发生死锁。p2psim的同步是通过一个事件队列和线程配合使用来实现的，这个方法简单、经典，且避免了死锁；3LS则使用了全局的时间步时钟来集中同步。

(8) 时延设定合理　P2P网络中有很多细节因素，除了时延，还有拥塞队列延迟、带宽、数据包丢失等，求全责备地让P2P模拟器能一下子模拟所有这些因素是不合理的，也不一定有必要。在这些细节因素中，时延是一个不容忽略的值，P2P

模拟器必须为网络(拓扑)设定合理的时延。在 p2psim 中网络的时延由拓扑提供，而其值基于对 Internet 主机的真实测量和分析。

(9) 新协议实现简单　通用 P2P 模拟器本身通常提供了一些已实现的协议，但这是远远不够的，用户需要基于该模拟器实现新协议。所以新协议的实现要尽可能简单，尽可能侧重算法，这要求 P2P 模拟器对 P2P 协议中的许多细节做合理的抽象。

(10) 模拟信息记录　模拟器必须能够记录必要的信息，如协议参数、时间、消息数、查询成功率等，这些信息对以后的分析有重要意义。此外，P2P 模拟器的信息记录部分最好能让用户修改，从而按照用户的要求记录所关心的信息。

8.2　经典的网络模拟器与拓扑产生器

计算机网络已经发展了好几十年，并且已趋向成熟、完善。有很多经典的网络模拟器与拓扑产生器已经得到了计算机领域的公认，所以将它们用到 P2P 的模拟上来，虽然不一定完全适合(比如最经典的网络模拟器 NS-2，它模拟的网络层次很低，最高达到传输层，但 P2P 却是工作在应用层的)，但经过改进以后，模拟结果常常是合理的、为 P2P 领域所接受的，因为一方面这些经典工具经受了时间和研究者的考验，各方面功能已经很完备，另一方面 P2P 网络也是一种计算机网络，所以使用经典的网络模拟器来模拟 P2P 协议是可取的。

8.2.1　经典的网络模拟器 NS-2

NS(The Network Simulator，网络模拟器)是开发于 UC Berkeley 的著名网络模拟器，它是一个面向对象的、离散事件驱动的网络环境模拟器。虽然 NS 设计目标也包含仿真功能，但一直不完善，所以一般都用 NS 来做模拟。NS 网站：http://www.isi.edu/nsnam/ns/。

NS 起源于 1989 年 REAL 网络模拟器的变形，在过去的这些年里它发展、进化了很多。1995 年 NS 开发受 DARPA 的 VINT(Virtual InterNetwork Testbed，虚拟因特网试验台)项目支持，参与这个项目的组织有 LBL、Xerox PARC、UC Berkeley 和 USC/ISI。目前 NS 的开发受 DARPA 的 SAMAN (Simulation Augmented by Measurement and Analysis for Networks，由网络测量和分析所增强的模拟)项目和 NSF 的 CONSER(Collaborative Simulation for Education and Research，教育和研究的协同模拟)项目支持。NS 不是一个完成的产品，而是一个不断努力中的项目。最初的 NS 发布版本为 NS-1，而目前为 NS-2，它包含了许多新的功能，且很多方面做了优化。

NS-2 可以模拟各种 IP 网络环境。具体地说：首先，NS 实现了对许多网络协

议的模拟，如著名的 TCP、UDP 协议，和数据源发生器，如 FTP、WWW、Telnet、Web、CBR 和 VBR 等；其次，NS 模拟了路由队列的管理机制，如 Drop Tail、RED 和 VBR，实现了 Dijkstra 和其他多种路由算法；再次，NS 还实现了各种网络多播协议和一些应用于局域网模拟的 MAC 层协议。

NS-2 由 C++ 和 Otcl 两种程序设计语言实现，Otcl 是加上面向对象特性的 TCL 脚本程序设计语言，由 MIT 开发。之所以要用两种语言来实现 NS，是因为 NS 模拟器有两方面的工作要做[杨璐，2002]：

一方面，具体协议的模拟和实现，需要一种程序设计语言，它需要很有效率地处理字节、报头等信息，需要应用合适的算法在大量的数据集合上进行操作。为了实现这个任务，程序内部模块的运行速度是非常重要的，而运行模拟环境的时间、寻找和修复 bug 的时间、重新编译和运行的时间就显得不很重要了。C++ 正是适合此要求的编程语言。

另一方面，许多网络中的研究工作，都是围绕着网络组件和环境具体参数的设置和改变而进行的，需要在短时间内快速地开发和模拟出所需要的网络环境，并且方便修改和发现、修复程序中的 Bug。在这种任务中，run-around time 就显得很重要了，因为模拟环境的建立和参数信息的配置只需要运行一次。Otcl 正是适合此要求的脚本语言。

NS-2 功能完备、成熟，它的模拟效果得到计算机网络领域的公认。但是 NS-2 体系架构复杂，需要相当长时间才能理解并使用它。

8.2.2 Transit-Stub 模型与 GT-ITM 拓扑产生器

Internet 是这个世界上最大、最重要的计算机网络，所有基于 Internet 的模拟器，都必须首先解决一个最基础、最核心的问题：给 Internet 的拓扑结构建模。Internet 的拓扑结构不是预先规划的，而是逐渐形成的，并处在不断变化中，而且这种形成过程往往是人为的、不确定的，所以在理论上对 Internet 根本无法准确建模。虽然如此，就像 NP 难问题一样，“没有最好的方法不代表没有近似的好方法”，Transit-Stub 模型就是一个得到计算机领域公认的 Internet 拓扑模型，它由 Kenneth L. Calvert 等人在 1997 年提出，论文[Calvert et al.,1997]发表在 *IEEE Communications Magazine* 上。到现在为止的 9 年时间里，Transit-Stub 模型一直是最经典、最实用的 Internet 拓扑模型，而基于它开发的 GT-ITM 软件 http://www-static.cc.gatech.edu/projects/gtitm/，也一直是最权威的 Internet 拓扑产生器。

Transit-Stub 模型认为：现代 Internet 可以看成一系列互联的“路由区域”(routing domain)的集合，每个路由区域是一组结点(路由器、交换机或者主机)，这组结点可以看成是受到统一管理、共享路由信息和路由规则。Internet 上的路

由区域可以分为两类：要么是 Stub 区域，要么是 Transit 区域。一个 Stub 区域只传输那些从区域内发起的消息或终止于本区域的消息，而 Transit 区域则无此限制。实际上，Transit 区域的作用在于有效地连接 Stub 区域，如果没有 Transit 区域，任意两个 Stub 区域间都得有连接。Stub 区域通常对应一个校园网或者其他形式的局域网（LAN），而 Transit 区域通常对应广域网（WAN）或城域网（MAN）。图 8.2.1 是一个经典的 Internet 区域结构的例子。

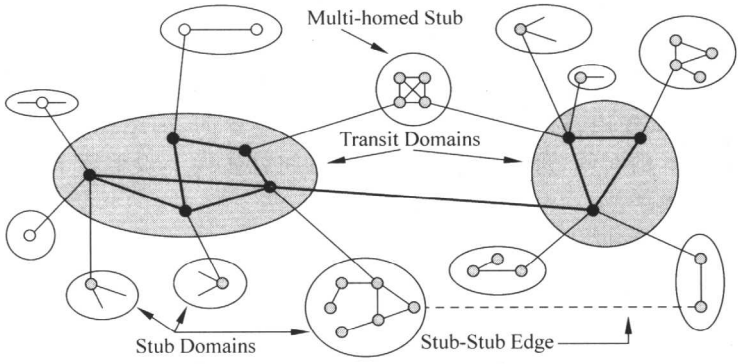


图 8.2.1 Internet 区域结构的例子
（图片来自[Calvert et al.,1997]）

一个 Transit 区域由一组“中枢结点”（backbone node）构成，每个中枢结点可以通过 Stub 区域的“网关结点”（gateway node）连接到一些 Stub 区域。图 8.2.1 中，深黑色的实心点表示中枢结点，读者一定能看出哪些是网关结点。

Stub 区域又可进一步分为 Single-homed 或者 Multi-homed：Single-homed Stub 区域仅连接到一个 Transit 区域，而 Multi-homed Stub 区域则连接到多个 Transit 区域。某些 Stub 区域还有到其他 Stub 区域的连接。此外，Transit 区域自身也可被层次化组织。

GT-ITM（Georgia Tech Internetwork Topology Models）是第一个很好地实现了 Transit-Stub 模型的网络模拟器（GT-ITM 也同时支持其他模型），它实际上是一组用来产生和分析 Internet 图模型（拓扑结构）的程序包。GT-ITM 是今天计算机领域公认的 Internet 拓扑产生器，可以在 http://www-static.cc.gatech.edu/projects/gtitm/中找到关于 GT-ITM 的详细资料。GT-ITM 最先在 Sun 公司的 Solaris 操作系统下开发，后来发布了 Linux 系统版本，在 Windows 操作系统下通过 cygwin 软件模拟 Linux 环境也能使用。

GT-ITM 通过反复地生成新图来产生互联的子图。产生新图的根据是边数和图的连通度，不连通的图被放弃。这个方法保证了最终产生的互联子图是从所

有可能的随机连通图中选择的。这个方法通常需要不少时间，因为如果边数较少，产生连通图的概率也会很小。

从 Stub 区域到 Transit 结点的边是由 GT-ITM 随机选择 Stub 区域和 Transi 结点然后添加的。除此之外，GT-ITM 还随机添加 Stub 区域到 Stub 区域的边。

作为对最基础的拓扑结构的增强，结点和边都被关联一些信息。每个结点有一个字符串标签，它标识了结点的属性：ID 说明此结点是一个 Stub 结点还是一个 Transit 结点，全局 ID 说明它属于哪个区域，还有一个区域内 ID。对于 Stub 结点而言，这个标签还说明了 Stub 区域所连接到的主 Transit 结点。对于边而言，每条边有它的路由策略权值(Routing Policy Weight)，用来帮助找到最短路由。

GT-ITM 使用了 Stanford Graph Base(SGB)来表达和操作图形，它包含了来自 SGB 的必要的库，因此用户不需要下载 SGB 代码或者去弄明白这些图(网络拓扑)是如何生成的，只要使用 GT-ITM 就可以产生和分析得到领域公认的网络图或拓扑结构。

8.2.3 通用拓扑产生器 BRITE

BRITE(Boston University Representative Internet Topology gEnerator)是美国波士顿大学开发的通用 Internet 拓扑产生器，其通用性体现在 3 个方面：

(1) 代表性：能反映真实 Internet 拓扑结构的多方面属性，如层次结构、结点度分布等；

(2) 广含性：能够组合多种模型产生网络拓扑，如层次模型、水平模型、幂律模型等；

(3) 互操作性：给许多广泛使用的网络模拟或可视化工具提供接口，如前面讲过的 NS-2。

BRITE 工作在类 UNIX 操作系统下，其功能部件有 C++ 和 Java 两种实现方式，用户可任选其一，如果希望使用图形界面则需要 JDK1.3 或更高版本的支持，因为 BRITE 图形界面是用 Java 开发的。

如图 8.2.2 所示，BRITE 首先从配置文件(1)中读取所要产生的网络拓扑参数，配置文件既可以手工写入，也可以通过 BRITE 的图形界面自动生成。BRITE 可以从其他拓扑产生器中导入拓扑数据(2)，如前面讲过的 GT-ITM，也可以导入直接从 Internet 上收集的拓扑数据。BRITE 组合导入的拓扑数据，输出 BRITE 格式的拓扑数据(3)或者其他格式(如 NS 格式)的拓扑数据。此外，BRITE 的开发者还提供 BRITE 分析引擎(称作“BRIANA”)来分析网络拓扑的各方面属性(4)。

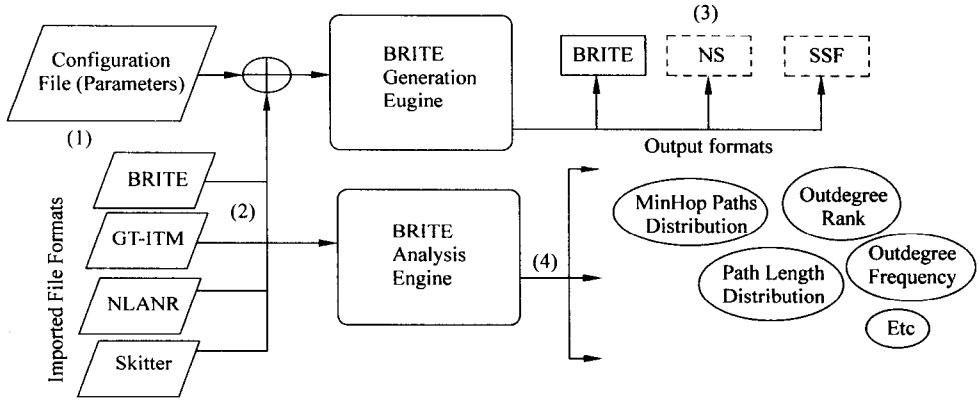


图 8.2.2 BRITE 架构示意图
（图片来自[Medina et al.,2001]）

8.3 P2P 模拟器

到目前为止，有影响力的 P2P 模拟器为数甚少，并且由于开发时间短，还没有非常成熟、公认的软件。这里我们介绍 4 个有代表性的 P2P 模拟器：p2psim，3LS，GnutellaSim 和 PeerSim，并着重讲解通用 P2P 模拟器 p2psim。

8.3.1 通用 P2P 模拟器 p2psim

p2psim 可以说是 P2P 模拟器中做得最好的，它是一个用来评价、研究、分析 P2P 协议的免费的、多线程的、离散的事件模拟器，其设计目标是要使理解 P2P 协议源码变得简单，使各种 P2P 协议的比较变得方便，并且要使模拟器本身具有合理的性能。p2psim 目前可以运行在一些类 UNIX 的操作系统中，它是 MIT 的 IRIS(Infrastructure for Resilient Internet Systems，容错因特网系统架构)项目的一部分，IRIS 项目的目标是基于分布式散列表(DHT)开发新型的分布式架构，以使得新一代的大规模分布式应用成为可能。下文对 p2psim 的讲述基于论文[Gil et al.，p2psim]和网站 http://pdos.lcs.mit.edu/p2psim/。

p2psim 支持一种网络延迟模型，该模型是通过对 Internet 主机的真实测量来构建的。要注意的是，p2psim 只给网络时延建立了模型，而对于网络队列延迟、带宽或者数据包丢失并未建模，这对于那些以时延为瓶颈的实验已经足够了。

2004 年 7 月 28 日，p2psim-0.2 版发布，这是最早正式发布的 p2psim 软件。2004 年 11 月 11 日建立了 p2psim mailing list(p2psim 邮件列表)，P2P 领域的研究者开始针对 p2psim 进行通信和讨论。2005 年 4 月 15 日，p2psim-0.3 版发布，

这是到目前为止 p2psim 软件的最新版本。p2psim 对网络协议做了抽象，去掉了网络协议中许多并不重要的细节，只留下需要考虑的核心 P2P 算法；此外，用户可以简化 P2P 协议实现的许多方面，比如可以让一个结点知晓所有其他结点的信息，以此简化 P2P 协议中的自适应操作。

目前的 p2psim-0.3 版实现的 P2P 协议有 Chord、Accordion、Koorde、Kelips、Tapestry 和 Kademlia。这些协议的实现是特定于 p2psim 的，协议代码的行数要远远少于协议的真正实现。

除了支持上述经典的 P2P 协议，p2psim 允许用户在特定的网络拓扑结构上以特定的工作负载运行不同的 P2P 协议。因为 p2psim 是一个 P2P 模拟器而非某种具体的网络，所以用户在用 p2psim 模拟实验时，可以拥有比实际网络实验更多的环境控制，从而可以通过 p2psim 研究多种协议在一些非常极端环境下的性能（比如我们对“崩溃点”的实验，就是使用 p2psim 制造高动态性的网络环境，而这在实际的 P2P 网络中是很难出现的）。最后，在 p2psim 实验中将结点总数扩展到很大的规模是容易的，因为这只是时间问题，而在实际 P2P 网络中做实验时，要扩展规模是非常困难的事。

开发 p2psim 的人员全部来自 MIT，他们是：Thomer M. Gil、Frans Kaashoek、Jinyang Li、Robert Morris 和 Jeremy Stribling，其中多数为在读博士。

读者可以在图 8.3.1 所示的 p2psim 的网站上找到更多关于 p2psim 的信息。

图 8.3.1　p2psim 主页 http://pdos.csail.mit.edu/p2psim/index.html，2007 年 3 月

1. p2psim 的主要构件与工作原理

p2psim 的主要构件包括：结点（Nodes）、网络（Network）、拓扑（Topology）、查询产生器（Lookup Generator）、搅动产生器（Churn Generator）。图 8.3.2 给出

了这些构件的基本图示和它们之间的关系。

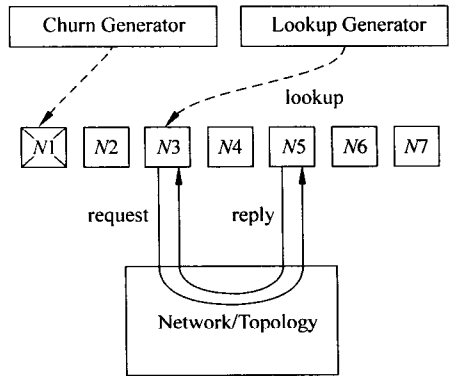


图 8.3.2 p2psim 的主要构件

（图片来自论文[Gil et al.,p2psim]）

一个结点（$N1$，$N2$，…，$N7$，…）模拟了 P2P 网络中的一台计算机，它们通过网络（Network）来互相发送消息（request，reply），而网络（Network）使用拓扑（Topology）来决定两个结点之间的时延。

搅动产生器（Churn Generator）模型化了 P2P 网络的动态性本质，它周期性地向网络中加入结点（通过调用结点的 join 方法），或者移除结点（通过破坏它）。在图 8.3.2 中，搅动产生器移除了结点 $N1$。

查询产生器（Lookup Generator）通过周期性地发出查询请求来模拟 P2P 网络的工作负载。虽然从原则上讲搅动产生器、查询产生器是不同的，但出于简化，p2psim 实际上是将两者作为一个构件来实现的。图 8.3.2 显示查询产生器向结点 $N3$ 发出了一个查询（lookup），$N3$ 收到后通过网络将查询（request）发送给了结点 $N5$，而 $N5$ 对此查询请求做了回复（reply），在此过程中 $N3$ 记录了此次查询完成的时间，以用于后来的用户分析。

在 p2psim 的一次模拟中，所有结点运行同样的 P2P 协议。对每个新协议，用户必须实现基础的 join 和 lookup 方法：join 方法初始化一个结点的路由表，并且联系一个或多个现存结点来使得自身成为 P2P 网络的成员，这个过程也称为“自举”（Bootstrap）；调用 lookup 方法的结点运行该协议的查询算法，来找到 ID 空间中与给定对象标识 K 最近的结点 N。结点可以周期性地运行“稳定化”程序来刷新路由表。

结点之间的通信通过 RPC（remote procedure call，远程过程调用）来进行，这些 RPC 实际表现为穿越网络的消息包。每当收到一个消息包时，网络（Network）询问拓扑（Topology）发送者和接收者之间的时延，此时延决定了消息包被送到目

的地的“模拟时间”。如果 RPC 被发送给一个“死亡的”结点，网络将模拟一次“超时”操作，在 RPC 发出以后 n 个 RTT(round trip time，回环时间)后若收不到回复(n 是一个用户配置的值)，网络发出特殊的“错误-RPC-回复”(error-RPC-reply)。

拓扑(Topology)给网络时延建模，它知晓网络中任意两个结点之间的时延。p2psim 支持多种不同的拓扑，包括一个基于因特网主机真实测量值的模型，该数据集是通过调整 King 项目中的技术来测量 1900 个域名服务器之间的时延值得到的。King 数据集提供了一个合理的时延数据集，并且获取了多个结点之间的时延关系，因为它直接测量了真实的因特网主机。King 测量技术依靠对递归的 DNS 查询的计时，为了决定两个域名服务器 A，B 之间的距离，其中 A 的服务域是 D_a，B 的服务域是 D_b，我们让 A 来解析 D_b 中的名字并给此操作计时，得到的计时值减去测量位置到 A 的时延，就认为是 A，B 之间的时延。

2. p2psim 中的线程

p2psim 使用线程来支持并发操作以提高性能，并使得源码易于理解。p2psim 的控制流设计目标是最大化性能并发性，最小化同步需要，并且避免死锁。下面我们来看看这些目标是怎么达到的。

p2psim 的主线程不断地从事件队列中取出队首的事件，并开一个新的线程执行此事件。事件队列中的事件是按照时间戳排序的，p2psim 模拟器的当前时间就是最近从队列中取出的那个事件的时间戳。当主线程为事件创建新线程后，主线程让出 CPU。直到再没有其他线程处于运行中为止，主线程才重新占有 CPU，从队列中取出下一个事件。

查询产生器(Lookup Generator)、搅动产生器(Churn Generator)、网络(Network)、结点(Nodes)都可以向事件队列中插入事件：查询产生器插入的是 lookup 查询事件，而搅动产生器插入的是结点加入事件和结点崩溃事件。

当主线程从事件队列中取出结点 N 的 lookup 或 join 事件后，它为此事件创建一个新线程 T，T 执行结点 N 的 handler，此时，T 运行在结点 N 的上下文(Context)中。T 发出 RPC，向网络发出一个消息包，然后阻塞以等待 RPC 回复(实际上 p2psim 也支持异步 RPC)。网络询问拓扑相应的时延，并向事件队列中插入一个“消息包传递事件”。当主线程从队列中取出该“消息包传递事件”时，它创建一个新线程来执行 RPC handler，这与 lookup 或 join 事件的处理相似。当最终收到 RPC 回复时，T 不再阻塞。线程在它所执行的 handler 方法返回后结束。一个结点也可以给它自己规划一个回调(Callback)，有时这是必要的，比如周期性地做“稳定化”操作就需要回调。

3. p2psim 的使用方法

p2psim 最新的版本是 2005 年 4 月 18 日发布的 0.3 版。在 p2psim 网站的 http://pdos.csail.mit.edu/p2psim/howto.html 页面上可以下载到 p2psim 的 0.1 版、0.2 版和 0.3 版。p2psim 目前只可以运行在类 UNIX 的操作系统中，并且其开发者建议使用 GCC 2.95.3 来编译。下面是下载、解压、配置、编译 p2psim 的典型步骤：

```
$ wget http://pdos.lcs.mit.edu/p2psim/p2psim-0.3.tar.gz
$ tar xvfz p2psim-0.3.tar.gz
$ cd p2psim-0.3
$ ./configure
$ make
```

要运行 p2psim 进行一次模拟，用户需要首先创建三个文件：协议文件、拓扑文件和事件文件。在 example/protocol.txt 中可以看到协议文件的语法和典型例子，在 example/topology.txt 中可以看到拓扑文件的语法和典型例子，在 example/events.txt 中可以看到事件文件的语法和典型例子。

创建好三个文件以后，就可以运行 p2psim 了，比如输入如下命令：

```
p2psim/p2psim  example/protocol.txt  example/topology.txt  example/events.txt
```

输出可能是：

```
average RTT=30ms
# 1: k
# 2: k_tell
# 3: alpha
# 4: stabilize_timer
# 5: refresh_timer
# 6: learn
# 7: rcache
#
...
FAILED_LOOKUPS::lookup_10th: 0 lookup_mean: 0.000 lookup_median: 0
lookup_90th: 0 stretch_10th: 0.000 stretch_mean: 0.000 stretch_median: 0.000
stretch_90th: 0.000 hops_10th: 0 hops_mean: 0.000 hops_median: 0 hops_90th: 0
numlookups: 0
OVERALL_LOOKUPS::lookup_10th: 8 lookup_mean: 19.273 lookup_median: 22
lookup_90th: 26 stretch_10th: 1.000 stretch_mean: 2.182 stretch_median: 1.000
stretch_90th: 1.000 hops_10th: 0 hops_mean: 0.091 hops_median: 0 hops_90th: 0
```

```
numlookups: 11
TIMEOUTS_PER_LOOKUP::time_timeout_10th: 0 time_timeout_mean: 0.000
time_timeout_median: 0 time_timeout_90th: 0 num_timeout_10th: 0.000
num_timeout_mean: 0.000 num_timeout_median: 0.000 num_timeout_90th: 0.000
WORST_BURST::in: 0 out: 0
<-----ENDSTATS----->
```

对上面输出结果中的各种参数，这里我们不做解释，因为它随协议而不同，并且输出什么、输出格式都是用户可以修改的。您可以在 p2psim 网站的 faq 中找到这些输出的具体含义。

除了上面所说的命令运行方式，实际上，在需要大量实验、多次更改参数的情况下，最好的方式是以脚本文件运行 p2psim(支持 Perl 脚本)。下面是一个运行脚本的例子：

```
scripts/run-simulations.pl --protocol Chord \
                           --topology your_favorate_topology_file \
                           --logdir. --args examples/chord_args
```

脚本可以节省大量的实验时间和精力，避免了多次键入命令的重复劳动，我们强烈推荐以脚本方式运行 p2psim。

8.3.2 三层 P2P 模拟器 3LS

3LS 也是一个有特色的 P2P 模拟器，只是不及 p2psim 那样流行和得到 P2P 领域的肯定，所以很少被用到，但它的不少思想方法是值得研究、借鉴的。

3LS(3-level simulator，3 层模拟器)。是一个基于时间步的 P2P 模拟器，它使用一个中央步时钟(step-clock)来模拟系统时间。3LS 将 P2P 网络看成如下 3 层：

用户层(user level)
协议层(protocol level)
网络层(network level)

协议层处在 3LS 的中间，它既负责模拟 P2P 协议和应用，也提供了网络层与用户层之间的服务接口。正是因为 3LS 将这三层分开，所以使对各种网络拓扑结构、P2P 协议、P2P 应用和用户的模拟、比较成为可能。

当打开 3LS 模拟器时，既可以选择为 3 层创建模型，也可以在 3LS 提供的库中选择一个合适的模型组合。随着模拟的进行，事件被输出到命令显示屏上；模拟结束后，所有的模拟数据被保存到文件以备将来分析。3LS 选择 Java 语言作为 P2P 模型的创建语言，因为 Java 语言简单、易用、功能完备，且在 P2P 研究者中非常流行。

网络模拟时的可视化是受人青睐的，3LS 的可视化是在工具 AiSee 的辅助下实现的（其网站为 http://www.aisee.com/），之所以选择 AiSee 是因为它简单、易用，能在多个操作系统下运行，在表达上的功能和性能也很好。每当 3LS 模拟网络的信息被保存到文件后，就可以使用 AiSee 的图形定义语言（GDL，graph description language）来将它图形化。图 8.3.3 是一个使用 AiSee 图形化 3LS 的例子。

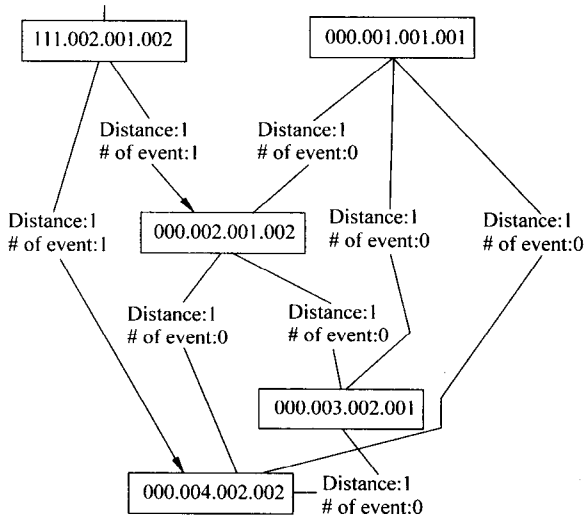


图 8.3.3 使用 AiSee 图形化 3LS 的例子

（图片来自[Ting & Deters,2003]）

8.3.3 数据包层 Gnutella 模拟器 GnutellaSim

GnutellaSim 是一个可扩展的数据包层（packet-level）Gnutella 模拟器，它用一个细致的网络模型来对 Gnutella 系统进行全面的评价。GnutellaSim 基于其开发者所设计的“数据包层 P2P 系统模拟”架构[He et al.,2003]，该架构的特色在于功能上的隔离（functional isolation）和协议中心化的结构（protocol-centric structure），以及可被扩展以实现其他协议的 P2P 系统。

GnutellaSim 是基于 NS-2.27 版和 PDNS 开发的。NS 在 8.3.4 节会讲到，而 PDNS 是 NS 的分布式版本：如果要模拟的结点数目很多，在单机上模拟速度很慢时，可以使用 PDNS 把模拟工作分布到多台计算机上以并发执行。目前的 GnutellaSim 已实现了 Gnutella 协议 0.4 版和 0.6 版，也就是说遗产结点（legacy peer）、超结点（ultra peer）、叶结点（leaf peer）都被包含进来。

为了支持 GnutellaSim 的模拟工作，其开发者扩展了 NS-2 中的 TCP 实现以

使其更接近真实的 TCP，附加的特色有：接受者通告窗口（receiver advertised window）、发送者缓存、类似 Socket 的 API、TCP 的动态连接建立、真实的有效载荷传输（real payload transfer）。在 GnutellaSim 主页可以找到关于它的更详细的资料：http://www-static.cc.gatech.edu/computing/compass/gnutella/。

GnutellaSim 是分层的，共分 3 层：应用层（application layer）、协议层（protocol layer）和套接字适配层（socket adaption layer）。GnutellaSim 不提供网络层，而是要求用户为其设定网络层模拟器，比如 8.3.4 节将讲到的 GT-ITM。图 8.3.4 展示了“数据包层 P2P 系统模拟”的体系架构，而 GnutellaSim 正是按照此架构开发的。

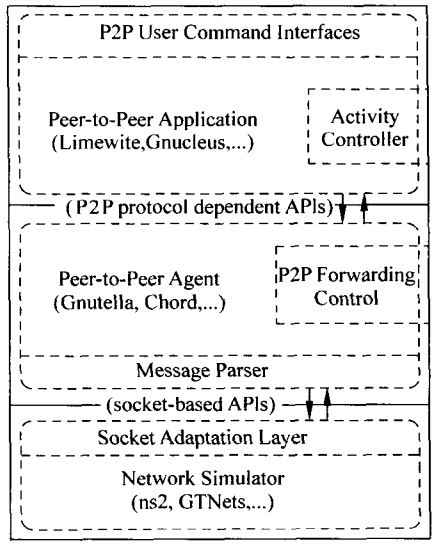


图 8.3.4 “数据包层 P2P 系统模拟”的体系架构
（图片来自[He et al.,2003]）

在最上面的应用层中，活动控制器（Activity Controller）是同每个 Peer 相关联以控制其行为的、基于计时器的实体。在中间的协议层中，Peer-to-Peer Agent 的工作是进行特定协议的消息传输和处理，在 Peer-to-Peer Agent 和 Peer-to-Peer Application 之间的 APIs 通常对应特定 P2P 系统的消息类型。消息解析器（Message Parser）代表为特定 P2P 协议的消息编码、解码的实体。最下面的套接字适配层（Socket Adaption Layer）用 Socket 接口包装了特定网络模拟器（Network Simulator）所提供的传输服务，因为 Socket 接口对真实 P2P 应用的开发者来说是最为熟悉的。

8.3.4 PeerSim 模拟器简介

PeerSim 的开发目标是高可扩展性和动态环境适应性。它由两个模拟引擎组成：一个是基于循环的简化版，另一个是事件驱动的。这些引擎由许多简单、可扩展的插件所支持。

基于循环的模拟引擎使用了一些简化的假设（比如忽略了传输层细节），所以可扩展性很好。事件驱动的模拟引擎效率较低，但它更符合实际情况（比如支持传输层模拟）。后者比前者的适用面更广，所以基于循环的协议也可以在事件驱动的模拟引擎上执行。

PeerSim 是 BISON 项目（http://www.cs.unibo.it/bison/）的一部分，它用 Java 语言编写，是独立于操作系统的开源软件。2005 年 11 月 PeerSim 1.0 版发布，可以到下面的网页下载：http://sourceforge.net/projects/peersim/。关于 PeerSim 的更多资料可参见其网页：http://peersim.sourceforge.net/。

8.4 全球网络服务仿真平台 PlanetLab

PlanetLab 是支持全球范围（因特网）新型网络服务开发、测试的开放式研究平台，自 2003 年创办以来，已有来自各个顶级学术和工业研究机构的超过 1000 名研究者使用 PlanetLab 来开发分布式存储、覆盖网络、P2P 系统、分布式散列表（DHT）和查询处理等多个领域的新技术。截至 2007 年 4 月 11 日，PlanetLab 拥有全球范围内 378 个地方的 775 个结点（见图 8.4.1），这些结点共同构成了一个跨越全球的独立覆盖网络，并且每个结点都是一台性能强劲、工作稳定的服务器。

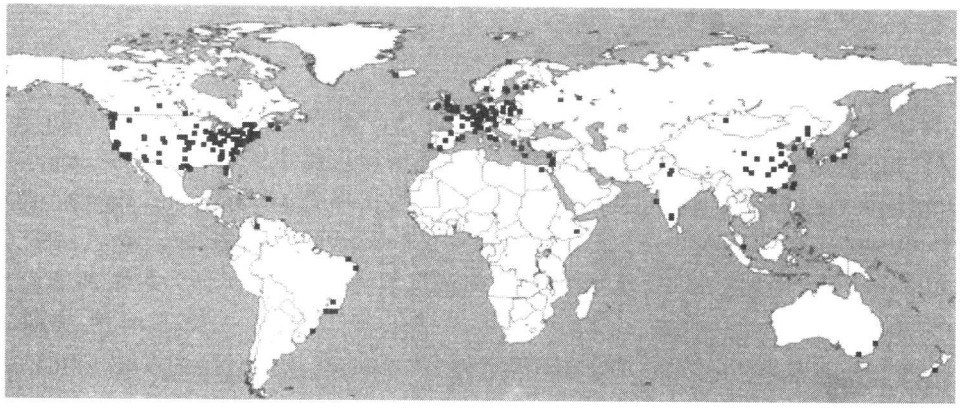


图 8.4.1 PlanetLab 结点分布图，2007 年 4 月

有关 PlanetLab 的历史可以追溯到 2002 年 3 月，来自普林斯顿大学的 Larry Peterson 和 UC Berkeley 的 David Culler 组织了一个非正式的全球网络服务研究者会议，讨论构建一个用来开发、评测全球范围网络服务的分布式共享平台。这次会议由 Intel 的 Berkeley 研究院做东，共有来自 MIT、CMU 等多所美国计算机名校的 30 名研究者参加。这次会议构思了这样一个图景：在全球范围内部署一个计算机集合，其中每台计算机(包括其网络带宽)都由参与的组织贡献，作为回报，其成员将有权使用所有计算机构成的这一平台——PlanetLab。Intel 研究院为 PlanetLab 提供了最开始的 100 台计算机，分布到 40 所高校中，并提供 PlanetLab 平台的操作支持直到 2004 年 1 月，在那之后，PlanetLab 的维护任务将转交位于普林斯顿大学的 PlanetLab 联盟。

PlanetLab 的设想一经提出，即获得网络领域研究者广泛重视。2002 年 8 月，也就是 SIGCOMM'02 会议召开前一天，大约 80 位网络系统研究者在 CMU 集合，讨论 PlanetLab 的应用及其架构设计。2002 年 10 月，PlanetLab 1.0 版软件发布，软件开发者为 Intel 的 Berkeley 研究院、分布式系统实验室和普林斯顿大学。2003 年 9 月，美国国家自然科学基金(NSF)向 PlanetLab 的设计、开发者颁发了 450 万美元的奖金，以进一步支持和增强 PlanetLab。2004 年 1 月，PlanetLab 2.0 版软件发布。

2004 年 12 月，中国教育和科研计算机网(CERNET)加入 PlanetLab，国内 25 所大学成为 PlanetLab 在中国的首批成员(可以在这个网页靠前的位置找到这些大学的名单：https://www.planet-lab.org/db/pub/sites.php)。此次 CERNET 的加入是“PlanetLab 中国项目”的开始，首先在中国 20 个城市的 25 所大学中设立 50 个 PlanetLab 结点(这些结点是由 HP 和 Intel 所捐助)，这使得 CERNET 成为亚洲第一个地区性 PlanetLab 研究中心。

PlanetLab 软件目前(2007 年 4 月)正升级到 4.0 版，大约有 600 个项目正运行于 PlanetLab 平台之上，可以在这个网页找到正在 PlanetLab 上运行的项目列表及项目信息：https://www.planet-lab.org/db/pub/slices.php。

第9章 P2P的现状和未来

今天的P2P，已从一个民间小软件，发展到改变Internet面貌、改变人们交流方式的一项重要技术。有句话说："今天的世界，网络上每传输两个比特，就差不多有一个是P2P的！"，这种说法并不夸张。

从商业应用来看，目前P2P业界存在两种趋势：扩张与合并。最著名的BitTorrent、eDonkey/eMule虽然已很流行，但仍然存在扩张的可能；优秀的网络语音传输软件Skype是P2P应用的一颗新星，其扩张规模令人惊叹。国内开发的P2P文件共享软件PP点点通、PoCo、百宝、卡盟等，都处在积极的扩张中，试图拥有更大的用户群，而P2P多媒体传输软件，尤其是网络电视软件如PPLive、CCIPTV、TvAnts等，已经相当流行，并且具有很好的市场前景。合并的例子在P2P领域并不少见，最著名的是2005年3月，以开发P2P协同工作软件"虚拟办公室"而闻名的Groove公司被软件巨人微软公司收购，随后Groove被整合到微软Office 12办公软件中。

在学术领域，2001年提出的四大结构化模型：Chord、CAN、Tapestry和Pastry的经典地位并没有动摇，仍然被P2P的研究者广泛引用和分析。后几年设计的新型P2P模型如Kademlia、SkipNet，常数度模型如Viceroy、Koorde、Cycloid等，影响力不断扩大。从计算机领域的各大会议、刊

物特别是P2P专业会议发表的论文来看，P2P的研究重点已经从核心机制逐渐转向增强机制，目前的研究热点集中在几个方面：P2P网络中的语义模糊查询、容错性、拓扑意识与一致性问题、声誉/信任和安全性问题。

应当承认，P2P技术还很年轻，离成熟、完善还有相当长的路要走。从P2P的祖先Napster开始就一直存在的“版权问题”，到今天仍然纠缠着大多数P2P的应用；而使用P2P来传播非法、色情内容的行为，一直没有得到有效控制，尤其在eDonkey这样的无结构网络中更为突出。学术界的研究者过分重视结构化P2P网络，而对简单实用的混合式、无结构P2P网络关注较少；过分看重理论的提出、论文的发表，而忽视了实际的软件开发、网络构建，也是一种不良的趋势。

虽然有太多的问题和障碍，但P2P在这几年中的发展、成长是有目共睹的。我们相信P2P的未来会是一片光明，人们在P2P上付出的努力不会白费。

9.1 P2P的主要研究组织

P2P的主要研究组织，包括世界计算机领域最有影响力的几大组织：ACM、IEEE和USENIX等，研究P2P的著名高校：UC Berkeley、MIT、Stanford、CMU、UIUC和Rice University等，研究P2P的著名公司：Microsoft、IBM、Intel、Sun和HP等，此外还有一些组织，下面会做介绍。

9.1.1 世界计算机领域最有影响力的组织

1. ACM

世界计算机领域最有影响力的科学、教育性组织，也是美国的重要专业学会，其宗旨在于不断推动计算机科学与技术的发展。

ACM(Association for Computing Machinery，(美国)计算机协会)成立于1947年，即世界第一台电子数字计算机(ENIAC)问世的第二年。它是世界上第一个计算机组织，所以ACM一直被公认为“The First Society in Computing”。图9.1.1是ACM网站的首页。

ACM的主要活动包括：①出版杂志、学报、书刊；②举行专业年会；③对有杰出贡献的计算机科学家、工程师、教育家和专业人员颁发多种奖，其中最著名的是世界计算机领域最高奖——图灵奖(A. M. Turing Prize)。

ACM的组织成员今天已达到9万人之多，其中大部分是计算机专业人员，大多数ACM成员又属于一个或多个SIG(special interest group，特别兴趣组)。ACM就像一个伞状的组织，为其所有的成员提供最新的有关科学、技术、应用的信息，它是世界计算机领域最新、最权威信息的重要源泉。

ACM 设有多份刊物和会议发表与 P2P 相关的论文，但没有专门针对 P2P 的，详见 9.2 节。

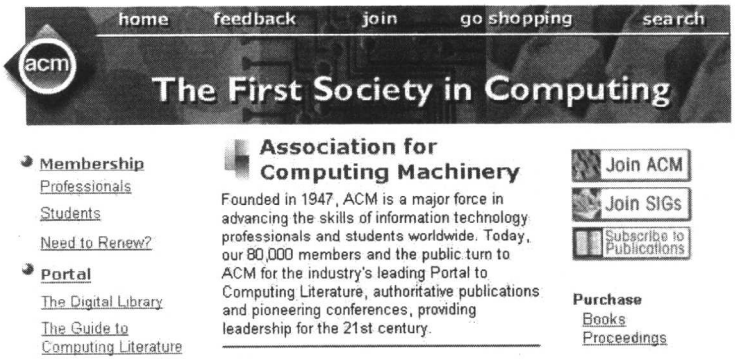


图 9.1.1 ACM 网站(http://www.acm.org)

2. IEEE

IEEE(Institute of Electrical and Electronics Engineers，电气电子工程师协会)是世界电气电子领域最有影响力的组织，也是世界计算领域最有影响力的科学、技术性组织，致力于推动电气电子技术在理论方面的发展和应用方面的进步。

IEEE 的前身 AIEE(American Institute of Electrical Engineers，美国电气工程师协会)和 IRE(Institute of Radio Engineers，无线电工程师协会)成立于 1884 年，1963 年 1 月 1 日 AIEE 和 IRE 正式合并为 IEEE。图 9.1.2 是 IEEE 网站的首页。

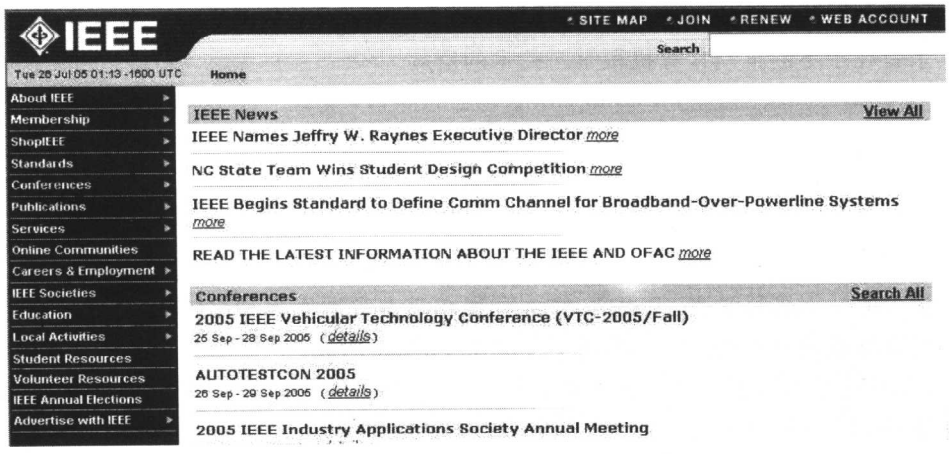


图 9.1.2 IEEE 网站(http://www.ieee.org)

IEEE是一个非营利性科技学会，拥有全球近175个国家36万多名会员。透过多元化的会员来源，它在计算机、电信、生物医学、太空、电力及消费性电子产品等领域都是主要的权威。在电气及电子工程、计算机及控制技术领域中，IEEE发表的文献占了全球将近30%。IEEE主要的活动也是出版杂志、学报、书刊，和举办专业会议，不过在数量上要远超过ACM。此外，还有一些IEEE专业社团(类似于ACM的SIG)，如IEEE Computer Society、IEEE Communications Society等。

IEEE设有多份刊物和会议发表与P2P相关的论文，并有专门的P2P会议，详见9.2节。

3. USENIX

1975年6月Mel Ferentz在美国曼哈顿举办了第一届USENIX会议(有来自20个协会的40个人参加了这次会议)，标志着USENIX的创立。当时它并不叫这个名字，而是一个“UNIX用户组”(UNIX Users' Group→USENIX)。

USENIX(Advanced Computing Systems Association，高级计算系统协会)将工作在世界计算机领域前沿的工程师、系统管理者、科学家、技师集合到一起，提供一个讨论计算机技术的中立论坛，以促进其技术不断革新和不断超越。USENIX是世界开源组织的成员。图9.1.3是USENIX网站的首页。

USENIX主办的几个会议(如USITS、OSDI、USENIX年会等)都收录了与P2P相关的论文，影响力都很大。

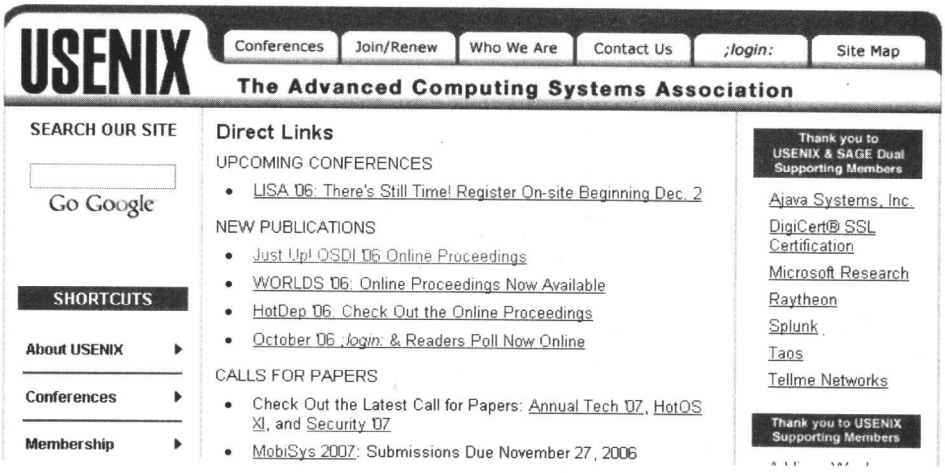


图9.1.3 USENIX网站(http://www.usenix.org)

9.1.2 研究 P2P 的著名高校

UC Berkeley——University of California at Berkeley 加利福尼亚大学伯克利分校（美国）

UC Berkeley 是加利福尼亚大学 9 所分校中历史最久、声望最高的一所，并且 UC Berkeley 的计算机系在全世界的高校中是排在最前的，正是它设计开发了 UNIX BSD 版操作系统。UC Berkeley 的 Ion Stoica 在 2001 年提出了最早、最经典的结构化 P2P 模型 Chord，Ratnasamy 在同年同会议（SIGCOMM）上提出了著名的多维空间 P2P 模型 CAN，而 Ben Zhao 等人则在同年提出了广域的超立方体结构 P2P 模型 Tapestry。UC Berkeley 在 P2P 领域的地位和造诣由此可见是非常突出的。

MIT——Massachusetts Institute of Technology，麻省理工学院（美国）

MIT 是美国或者说是世界上最佳的理工科大学之一。MIT 的计算机研究也是全世界首屈一指的。MIT 的 Robert Morris 等人参与了 Chord 的设计，Dabek 等人开发了基于 Chord 的协同文件系统 CFS，而 Kaashoek 等人在 IPTPS'03 上提出了著名的常数度 P2P 模型 Koorde，并对 P2P 领域的一些重要理论问题作了解答。值得一提的是：由 Kaashoek 教授领衔的研究小组，联合其他美国一流高校和研究机构进行的 IRIS 项目（Infrastructure for Resilient Internet System，容错的因特网系统架构），用 P2P 的方法去研究并建立新一代互联网络结构，于 2003 年得到了美国 NSF（自然科学基金）在 IT 领域最大的一项基金资助。由此可见 MIT 在 P2P 领域的研究毫不逊色于 UC Berkeley。

Stanford（University of Stanford，斯坦福大学（美国））

斯坦福大学与哈佛、耶鲁齐名，是美国也是世界一流大学，被奉为经典中之经典的《计算机编程艺术》的作者、计算机宗师 Donald E. Knuth 正工作于此。斯坦福大学的研究者（如 Stanford Peers 研究组）在 P2P 领域做了大量工作，发表了很多有价值的论文。

CMU（Carnegie Mellon University，卡内基·梅隆大学（美国））

从综合实力上看，CMU 在美国肯定排不进前十位，但 CMU 的计算机系历史悠久、贡献卓越，一直走在世界计算机科学的最前沿。

UIUC（University of Illinois at Urbana-Champaign，伊利诺伊大学香槟分校（美国））

UICU 一直被列为美国最优秀的理工大学之一，其势头直逼大名鼎鼎的

MIT,UIUC的计算机系亦毫不逊色,在P2P领域做了大量有价值的工作。

Rice University(赖斯(Rice)大学(美国))

Rice University在美国的大学中并不如前面几个那样出名,它的计算机系也不及前面提到的各所大学。在P2P领域,2001年Rice University的Peter Druschel参与设计了著名的混合式超立方体结构P2P模型Pastry和基于Pastry的应用系统PAST。

9.1.3 研究P2P的著名公司

Microsoft——微软公司(美国)

微软这个神奇的"软件巨人"就不必再介绍了。在P2P领域,2001年微软研究院的Antony Rowstron提出了著名的混合式结构P2P模型Pastry,开发了基于Pastry的应用系统PAST;2003年微软研究院的Harvey等人设计了基于跳表结构的新型P2P网络SkipNet。实际上,微软研究院在世界各地的多个分部都有P2P的专业研究组,Pastry与SkipNet仅是他们工作中最出名的那部分。2005年微软收购了以开发P2P协同工作软件"虚拟办公室"而闻名的Groove公司,并将Groove整合到最新的Microsoft Office 12办公软件中。

Intel——英特尔公司(美国)

如果说微软是"软件巨人",那么称英特尔为"硬件巨人"毫不过分。2000年8月,Intel宣布成立P2P工作组,正式开展P2P研究。工作组成立以后,积极与应用开发商合作,开发P2P应用平台。2002年Intel发布了.NET基础架构之上的Accelerator Kit(P2P加速工具包)和P2P安全API软件包,从而使得微软.NET开发人员能够迅速地建立P2P安全Web应用程序。上面所说的其实只是Intel在P2P领域所做工作的一部分。

Sun——Sun公司(美国)

人们常把Sun公司与Java语言联系在一起,实际上,在开发Java之前Sun公司在计算机领域已经很有影响力了。Sun在P2P方面做了大量工作,有学术性的,也有商业性的,最著名的是JXTA项目,其目的在于提供一个开放、通用、互操作的P2P开发平台。

ACIRI——AT&T Center for Internet Research,AT&T因特网研究中心(美国)

AT&T在计算机和电子通信领域应该算是家喻户晓了,它的起源可以追溯到

1876 年贝尔发明电话。AT&T 因特网研究中心的 Ratnasamy、Francis 等人在 2001 年 SIGCOMM 会议上提出了著名的多维空间 P2P 模型 CAN。

IBM——International Business Machines Corporation，国际商用机器公司（美国）

我们说微软是“软件巨人”，英特尔是“硬件巨人”，那么 IBM 呢？IBM 是名副其实的“计算机巨人”，是世界计算机领域真正的先驱和前辈。IBM 在 P2P 领域的工作侧重于商业和技术，它是英特尔成立的 P2P 工作组的成员之一，并且 IBM 和 HP 这两家公司在 2000 年 9 月共同推出了一种开放存储技术，这一存储技术利用了 P2P 技术，可以方便地从用户的硬盘向服务器上复制数据。

HP——惠普公司（美国）

或许 HP 谈不上“巨人”的称号，也没有什么非常闪亮的产品，但 HP 确实是一家涉足多个计算机领域的著名公司。HP 在 P2P 方面做的工作，既有学术性的，也有商业性的。HP 也是英特尔成立的 P2P 工作组的成员之一，上文已说过 IBM 和 HP 这两家公司 2000 年 9 月共同推出的利用 P2P 技术从用户硬盘向服务器上复制数据的开放存储技术。此外 HP 公司还把 P2P 的立足点放在打印技术上，该公司新推出的网络打印技术可使用户通过 P2P 网络共享打印机。

9.1.4 研究 P2P 的其他组织

P2PWG 和 GGF——P2P Work Group & Global Grid Forum，P2P 工作组和全球网格论坛

P2P 工作组成立的主要目的是希望加速 P2P 基础设施的建立和相应的标准化工作。P2P 工作组成立之后，对 P2P 中的术语进行了统一，也形成相关的草案，但是在标准化工作方面工作进展缓慢。目前 P2PWG 已经和 GGF 合并，由 GGF 负责网格计算和 P2P 计算等相关的标准化工作。

下面是一些重要的 P2P 网站：

网 站 名	注 释	链 接
P2P 中国 (PPCN. NET)	P2P 行业第一中文门户网站	http://www. ppcn. net
P2P 天空 (P2PSKY. COM)	也是著名的 P2P 中文门户网站	http://www. p2psky. com/
JXTA 网站	JXTA 开发社区	http://www. jxta. org
P2P INFO	这里提供了非常多的 P2P 信息	http://p2p. info. pl/

续表

网 站 名	注 释	链 接
OpenP2P 网站	O'REILLY 开办的关于 P2P 的开放式网站，有非常全面的 P2P 信息	http://openp2p.com/
P2P Weblog	P2P 的 Web 博客专栏网站	http://p2p.weblogsinc.com/

9.2 P2P 的重要国际会议和刊物

与 P2P 相关的重要国际会议既包括 P2P 的专业会议，也包括与 P2P 相关的会议。但到目前为止，没有 P2P 专业的国际刊物，与 P2P 有关的论文多发表在与分布式系统或计算机网络相关的刊物上。实际上，由于 P2P 技术发展很快，新技术不断产生，所以会议对 P2P 来说更为重要也更有影响力，而刊物由于审稿时间长，对 P2P 来说，刊物的作用更多地表现在肯定那些过去很有价值的东西。

9.2.1 有关 P2P 的重要国际会议

1. P2P 专业国际会议

IPTPS——International Workshop on Peer-to-Peer Systems，国际 P2P 系统讨论会

P2P 领域最高级的专业国际会议，2002 年第一次召开，每年召开一次，其赞助方有 Microsoft、Intel、IBM 等国际著名计算机公司。IPTPS 虽然到目前为止只举办了 4 届，但对 P2P 的影响是巨大的，比如发表在 IPTPS'02 上的 Kademlia 模型、IPTPS'03 上的 Koorde 模型。实际上 IPTPS 上发表的许多论文，都成为 P2P 领域的指导性文章，提出了亟待解决的问题，引领着 P2P 领域的研究方向。

IEEE P2P——IEEE International Conference on Peer-to-Peer Computing，IEEE 国际 P2P 计算会议

IEEE 举办的 P2P 专业国际会议，2001 年第一次召开，每年召开一次。IEEE P2P 上发表的论文数目较多，对 P2P 的发展作出了一定的贡献。

2. 与 P2P 相关的国际会议

SIGCOMM——ACM Annual Conference of the Special Interest Group on Data Communication，ACM 数据通信特别兴趣组年会

SIGCOMM 可以说是网络通信领域最高层次的会议，它每年召开一次，已经

有很多年的历史了。SIGCOMM 会议上发表的论文，基本上都是网络通信领域的经典、权威之作，对整个网络通信领域有深远的影响。对 P2P 而言，SIGCOMM 同样有着举足轻重的作用，比如最著名的 P2P 模型 Chord、CAN 就发表在 SIGCOMM'01 上。

SPAA——ACM Symposium on Parallel Algorithms and Architectures，ACM 并行算法和体系讨论会

SPAA 是由 ACM 举办的并行算法和体系方面的高层会议，1989 年创办，每年召开一次。Plaxton 等人在 SPAA'97 上提出的 Plaxton Mesh 结构，成为 P2P 中的两个著名模型 Tapestry 和 Pastry 的基础。

PODC——ACM Symposium on Principles of Distributed Computing，ACM 分布式计算原理讨论会

PODC 是由 ACM SIGACT（算法和计算理论特别兴趣组）和 SIGOPS（操作系统特别兴趣组）支持开办的分布式计算领域年会，已有很多年历史，其赞助方有 Microsoft、Intel、Google 等国际著名计算机公司。PODC 上发表了不少有关 P2P 的重要论文，比如最早的常数度 P2P 模型 Viceroy 发表在 PODC'02 上。

ICNP——International Conference on Network Protocols，国际网络协议会议

ICNP 是 IEEE 举办的网络协议高层次会议，每年召开一次，收录网络协议各个层次、各个方面的研究论文，当然也包括 P2P 领域的。

INFOCOM——IEEE Conference on Computer Communications，IEEE 计算机通信会议

INFOCOM 是 IEEE 举办的计算机通信领域高层次会议，每年召开一次，已经有很多年的历史（始于 1982 年）。INFOCOM 会议上每年发表的论文数非常多，所以分设很多专题研究组（Workshop），近几年都有 P2P 的 Workshop。有很多 P2P 领域的重要论文发表在这些会议上。

NOSSDAV——ACM International Workshop on Network and Operating System Support for Digital Audio and Video，ACM 数字音频和视频的网络和操作系统支持讨论会

NOSSDAV 关注多媒体应用、网络、操作系统等领域的新观点和未来发展方向，其范围已拓展到网络游戏、传感器网络、多媒体界面和 P2P 网络。NOSSDAV 由 ACM 的 SIGMultimedia 研究组于 1990 年开办，每年召开一次，是一个很有影

响力的会议。

SOSP——ACM Symposium on Operating Systems Principles，ACM 操作系统原理讨论会

SOSP 是由 ACM SIGOPS(操作系统特别兴趣组)支持举办的操作系统领域的高层次会议，1967 年第一次召开，每两年召开一次，是一个非常老的会议。SOSP 也收录为数不多的 P2P 论文，但影响力非常大，如著名的结构化 P2P 应用 CFS 和 PAST 发表在 SOSP'01 上。

OSDI——USENIX Symposium on Operating Systems Design and Implementation，USENIX 操作系统设计和实现讨论会

1994 年 USENIX 创办了 OSDI 会议，此后每两年召开一次，OSDI 是操作系统领域的高层次会议，影响力非常大。

USITS——USENIX Symposium on Internet Technologies and Systems，USENIX 因特网技术和系统讨论会

USITS 是由 USENIX 开办的网络领域高层次会议，1997 年创办，每两年召开一次，虽然历史很短，但影响力很大。USITS 也收录 P2P 论文，如 SkipNet 模型发表在 USITS'03 上。

ICDCS——International Conference on Distributed Computing Systems，国际分布式计算系统会议

ICDCS 是由 IEEE Computer Society Technical Committee on Distributed Processing(TCDP)在 1979 年举办的，一开始每 18 个月召开一次，从 1984 年开始每年召开一次，在分布式系统领域很有影响力。

ICPP——International Conference on Parallel Processing，国际并行处理会议

ICPP 创办于 1972 年，每年召开一次，是一个非常古老的会议。ICPP 收录的论文数较多，所以下设多个会议专题，最近几年都有 P2P 的专门会议(Session)。

IPDPS——International Parallel and Distributed Processing Symposium，国际并行和分布式处理讨论会

IPDPS 是 IEEE 举办的并行和分布式处理领域的重要会议，每年召开一次。IPDPS 会议上发表的论文数一般较多，下设很多专题研究组，最近几年都有 P2P 的讨论会(Workshop)。

9.2.2 P2P 的重要国际刊物

P2P 是一个很新、发展很快的领域，新思想不断涌现、新方法不断产生，所以实际上会议对 P2P 来说更为重要，也更具影响力，能很快推广新东西。刊物由于审稿时间长、发表要求高，所以对 P2P 领域来说，刊物的作用多在肯定那些过去很有价值的东西，因此，这里我们只列出对 P2P 来说最重要的几种权威国际刊物。

IEEE/ACM Transactions on Networking——IEEE/ACM 网络学报

由 IEEE Communications Society、IEEE Computer Society 和 ACM SIGCOMM(数据通信特别兴趣组)支持创办的通信网络领域的权威学报。

IEEE Transactions on Parallel and Distributed Systems——IEEE 并行和分布式系统学报

IEEE 在并行和分布式系统领域的权威学报，其中刊登了不少关于 P2P 的重要论文。

IEEE Journal on Selected Areas in Communications——IEEE 通信精选领域杂志

这份通信领域颇有影响的杂志收录 IEEE 在通信领域的重要论文。

9.3 P2P 的主要商业模式

本节我们总结 P2P 的各种商业模式，看它们是如何产生、运作的，遇到了什么样的问题，对商业领域关心 P2P 的人有何启发。

商业的目的本质上只有一条——赢利，有句话说："没有永远的朋友，也没有永远的敌人，只有不变的利益。"这个残忍的断言说出了一种事实：利益是真正驱使这个世界运作的原动力，金钱在大多数情况下有决定性的发言权。

上述事实在 P2P 领域得到了坚实的肯定，最主要表现在 P2P 的商业赢利模式、商业纠纷、企业运作上。实际上，即使是在不那么功利化的学术界，政策性的项目资助或者企业技术投资，对学术界的 P2P 研究者也是至关重要的，毕竟，不管哪个人，总是先要吃饭，然后才能思考的。

有人说，每六个月，硅谷都会为一种新技术的诞生而激动。媒体疯狂炒作，大大小小的公司便蜂拥而上，腰缠万贯的风险投资商也将钞票大把大把地撒在上面。但结果呢？个别幸运的公司经过拼搏看到了一线希望的曙光，个别公司被某个大公司收购，而其他所有的公司都会土崩瓦解，同一些新名词、新概念一道随风而逝。

从"网络泡沫"的破灭，到 2000 年以电子商务为代表的新经济全面溃败，经过

大风大浪的洗礼，渴望一夜暴富的投资者开始变得谨慎和理智，所以相比“Peer-to-Peer”的说法，人们更愿意把P2P称为“Path to Profit”（赢利之路）。对于这条赢利之路，人们不再像互联网早期那样盲目和狂热，而开始仔细地规划它、实现它。在多年的探索中，人们总结出了P2P有代表性的各种商业模式。

1. Napster模式

Napster这个像流星一样耀眼而短暂的P2P的先驱，代表了一种简单而实用的P2P赢利模式：网站提供服务器，给用户查询和存放文件索引，而真正的文件交换直接发生在用户之间。虽然可以列举出这种模式的诸多缺点：服务器瓶颈、单点失效等，但Napster模式确实是成功的：公司组织人员开发一个并不复杂的P2P软件，提供服务器（和网站），争取到足够多的用户，赢利就开始了（广告是主要的赢利方式）。

打倒Napster的并不是它技术上的缺点，而是版权问题带来的法律纠纷，因为对Napster模式，只要关掉它的服务器，一切就全完了！

Napster的陨落并不意味着Napster模式没有市场。实际上，我们看到世界上大多数P2P应用所采用的还是Napster模式，比如国内的PP点点通、PoCo、百宝等一大批P2P文件共享软件都使用了Napster模式，但它们都很流行，而且到目前为止运作得还算好。

2. BitTorrent模式

BitTorrent是现在最流行的P2P应用，在国内P2P市场更占统治地位。实际上BT也是基于服务器的，只不过服务器很多，分散在网络中，所以BitTorrent模式相当于多个Napster模式分散的组合。现在因特网上有数不清的BT网站提供BT种子下载和搜索，每个网站都像一个小型的Napster。

BitTorrent也一直面临版权问题，它之所以没有消失反而一直工作得很好，原因就在于BT网站太多、太分散，没有办法去逐个起诉、关闭，这种情况好像中国的一句谚语——“法不施众”。

但是，BT网站不被关闭并不代表着不被监管，现在国内最著名的几家提供种子搜索的BT网站，他们提供的种子文件中极少含非法、色情内容，可见其服务器中的种子文件必定经过了检查和过滤。

3. eDonkey模式

eDonkey模式代表了一种典型的双层无结构P2P网络赢利模式，它也是基于多个分散在因特网中的服务器的，但与BitTorrent本质的不同在于：这些服务器中绝大多数是随机、动态的，不属于哪个固定的网站。所以eDonkey不仅不可能

被起诉、关闭，甚至连监管都是极难办到的。

虽然如此，eDonkey 还是有一点把柄可以被抓住的，就像《荷马史诗》中的希腊英雄阿基里斯，虽然全身无敌但还是在脚踵留下了缺陷：eDonkey 用户加入网络需要通过为数不多的众所周知服务器（也称“入口服务器”），这些服务器一般是不变的，所以只要关闭了它们，eDonkey 也活不长了。

eDonkey 涉及的社会问题非常多，除了版权问题，还有非法、色情内容的传播，因为 eDonkey 网络中传播内容基本不可能被监管。

4. Gnutella 模式

Gnutella 模式是典型的无结构 P2P 网络赢利模式，它没有服务器，是纯分散式的。Gnutella 模式的缺陷与 eDonkey 模式一样，也是用户加入时必须用到的那些入口服务器。

Gnutella 刚产生，就因为其母公司害怕卷入法律纠纷而被关闭，但 Gnutella 模式流传了下来。今天，诸如 BearShare、Gnucleus、LimeWire 等一大批 Gnutella 模式的 P2P 应用都有一定的流行性。Gnutella 模式一直没有达到 Napster 模式、BitTorrent 模式、eDonkey 模式这样的成功，原因在于纯粹的无结构 P2P 网络工作效率不高。

5. Lightshare 模式（P2P 电子商务）

作为一个刚成立不久的 P2P 公司，Lightshare 的赢利方法是在 P2P 网络上开展电子商务。用户可以在 P2P 网络上购买或出售商品，可以搜寻待售商品信息或者列出自己欲售商品的信息。待售的商品保留在出售者的计算机里，中心服务器动态地保存该商品的相关信息，以供其他用户查询。

在 Lightshare 开展的服务中，初期的交易商品只包括数字文件，但是以后将逐渐扩展到其他领域。这种赢利模式目前存在着管理、计费、安全等方面的困难。

6. GPU 模式（P2P 分布式计算）

GPU 是一个真正的 P2P 分布式计算系统，它使用经典的无结构 P2P 网络——Gnutella 来分布计算任务、共享计算能力。每个 GPU 用户创建一个计算机别名来加入 GPU 体系，通过 Gnutella 网络来提供自己的 CPU 计算能力，同时获得系统中别人的 CPU 计算能力。

很明显，GPU 模式可以用来交易 CPU 计算能力，这是它赢利的基础。

7. Skype 模式(P2P 语音传输)

在 VoIP 之前,语音传输是电信电话的专利;而 Skype 的出现不仅冲击了电信电话,更挑战了 VoIP。Skype 以其清晰的语音传输质量、加密的通信过程、高效可靠的服务从众多网络通信工具中脱颖而出,并且随着各国政府或电信部门对 Skype 使用的逐步开放,Skype 的影响力将越发广泛。

使用 Skype 通话必须购买相应的"电子通话货币",但其价格比打电话要便宜得多。

8. PPLive 模式(P2P 网络电视)

Internet 网络电视过去一直是收费的在线直播网站的市场,后来有了 IPTV,将这个市场分解;而 P2P 网络电视既瓜分市场,又挑战 IPTV。国内的 PPLive、CCIPTV、TvAnts 等 P2P 网络电视软件已非常流行。

9. 综合模式

综合模式是将上述的两种或多种模式结合起来,取长补短,提供更好的 P2P 服务。也可以把 P2P 同其他的商务模式(如 B2B、ASP 等)或者网络技术(如搜索引擎技术、网格计算等)相结合,探索新的赢利模式。

除此以外,还有一些无法预见或无法定论归类的赢利模式,因为作为一项新技术,人们对 P2P 的认识都很不完整。有人把人们现在对 P2P 的认识比作"盲人摸象":每个人都只看清了其中的一个部分,而完整的认识还需要在这个领域的探索中逐渐形成。

除了单纯的赢利之外,对投资者来说,P2P 的吸引力还在于它低成本的起步、病毒式的传播方式。比如 P2P 的祖先 Napster,几乎从未做过广告,但是其用户数量却得以指数级暴涨,这和以前的网络应用大不相同,也是其他网络服务无法企及的。

9.4 P2P 与其他领域的融合

9.4.1 P2P 与无线网络的融合

无线网络是一个研究很早、范围很广的领域,包括最传统的无线 Ad hoc 网络(也称 mobile Ad hoc 网络,简称 MANET)、最近几年逐渐成为热点的无线传感器网络(wireless sensor network,简称 Sensornet)和无线 Mesh 网络(简称 WMN)。这里我们主要关注 P2P 技术在无线 Ad hoc 网络中的应用。

无线 Ad hoc 网络是一组可移动的无线网络结点的集合，结点间只依靠无线通信和互相协作来自组织成一个临时网络，不依赖于任何底层体系结构或集中式管理的支持。由于结点间通过无线通信来连接，而无线通信的范围一般较小、不可能覆盖整个网络，所以两个结点间通信一般要依靠其他中间结点来路由消息。又由于结点的移动性，无线 Ad hoc 网络的拓扑结构几乎总是处于变化中，如何在高动态性的环境下保持无线 Ad hoc 网络的高性能，这是研究无线 Ad hoc 网络的最大挑战所在。

针对无线 Ad hoc 网络目前已提出了许多种路由算法，其中最简单而又最具影响力的三个是：DSDV，AODV 和 DSR。DSDV（Dynamic Destination-Sequenced Distance Vector Routing，动态目标序列化距离向量路由）发表在 SIGCOMM'94 会议上[Perkins & Bhagwat，1994]，它将传统的 Bellman-Ford 距离向量路由算法改进以适应 MANET 的需求，而这种改进是依靠使用基于目标结点的序列号来反映路由信息的时效性、同时避免回路的形成。DSDV 通过周期性的洪泛广播来维护路由表，这种方法常称为 Proactive（前摄的、主动的）；与此相反，AODV、DSR 只在需要时才创建路由，平时不做周期性的路由表维护，所以常称为 Reactive（反应的、被动的）。

AODV（Ad-hoc on-demand distance vector routing，Ad hoc 按需距离向量路由）[Perkins & Royer，1999]也是基于 Bellman-Ford 算法，只是取消了周期性的路由维护，仅当需要路由时才通过洪泛广播的方式创建（见图 9.4.1）；同时 AODV 也使用了 DSDV 的序列号（细心的读者应该已经发现 DSDV、AODV 的设计者都有 Perkins）。DSR（dynamic source routing，动态源站选路）[Johnson & Maltz，1996]的特点是每条路由消息中都包含已经建立的整条路径，消息中包含更多的信息自然带来了很多好处，同时开销也必然增加。总体上说，不管哪种 Ad hoc 路由算法，要么属于 Proactive，要么属于 Reactive，要么是将两者结合。Proactive 方法的优点是可以随时快速地建立路由，代价是周期性维护；Reactive 方法的优点是无需周期性维护，代价则是建立路由的时延相对较长。

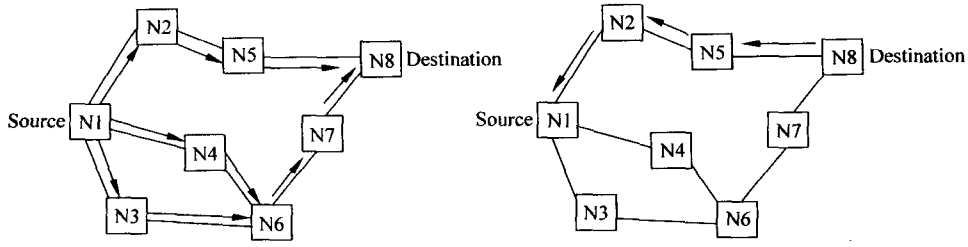


图 9.4.1 AODV 的路由（左图是洪泛广播的过程，右图是路由建立的过程）
（图片来自[Royer & Toh，1999]）

看到这里读者容易发现，P2P 网络和无线 Ad hoc 网络之间有相当多的共性，比如不包含集中式的管理、高动态性的网络环境、网络自组织的必要性和重要性等。另一方面，两者之间又有着本质的不同，使得 P2P 和 MANET 领域的技术不能互相照搬，最明显的不同之处表现在两方面：①P2P 网络是一个应用层覆盖网，构建在 Internet 网络层之上，覆盖网的路由实际上依靠 Internet 的 IP 路由来实现，而 MANET 本身就是网络层，路由直接发生在 MANET 中；②P2P 网络的动态性主要表现在结点的加入和离开，而 MANET 的动态性则主要表现在结点的移动性，从而决定了两者自组织方式的不同。

鉴于 P2P 与 Ad hoc 之间的共性与区别，已经出现了一些建议性的方案，试图将两者结合起来。这方面较早的是发表于 IPTPS'03 会议上的 PeerNet [Eriksson et al.,2003]，它将 MANET 中的结点 ID 和其位置分离，通过额外的查询服务来实现两者间的映射，并且将网络结点层次化地组织成从大到小的区域从而构成一棵树，同一小区域内的结点匹配更多的 ID 前缀，路由过程一般是先爬树、再下树（如图 9.4.2 所示）。为了维持树的平衡，PeerNet 需要基于一些准则动态迁移结点。总体看来，PeerNet 用到的机制比较复杂，整体开销也很大，对 MANET 来说却并未取得什么实质性的好处，所以只能算一个雏形，但它所倡导的“将 P2P 从网络协议栈的高层下压到低层”的思想，很有启发意义。

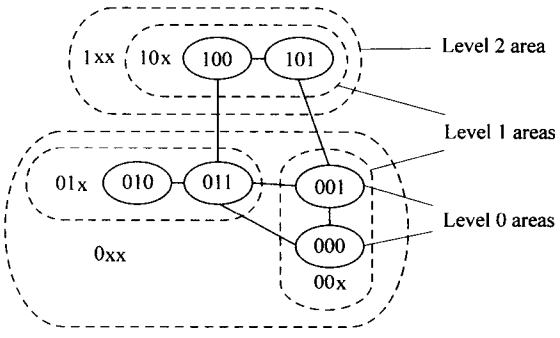


图 9.4.2 一个简单的 3 层 PeerNet 组织结构图

（图片来自[Eriksson et al.,2003]）

发表于 HotOS'03 会议上的 DPSR [Hu et al.,2003]以及 2005 年提出的 CrossRoad [Delmastro,2005]和 MADPastry [Zahn & Schiller,2005]都继承了 Pastry 的一些特性。在 P2P 网络中，Pastry 路由表中的每一项代表一个 IP 地址，而 DPSR 将每一项换成了一条路由路径，这条路径是通过 DSR 的洪泛广播得到的。MADPastry（Mobile Ad hoc Pastry）削减了 Pastry 的 Route Table 和 Leaf Table，路由的创建使用 AODV 算法。

这方面最受好评的是发表于 SIGCOMM'06 会议上的 VRR（virtual ring

routing，虚拟环路由)[Caesar et al.，2006]，很多学者对它给出了相当高的评价，并认为 VRR 将实用化地给 MANET 领域带来影响(现在微软研究院已经开发出了 VRR 的 Windows XP 实现，可以到下面的链接下载软件：http://research.microsoft.com/vrr/)。不管怎么说，VRR 确实是将 DHT 思想应用到 MANET 的成功代表。

如图 9.4.3 所示，VRR 给 MANET 中每个结点随机分配一个 ID，按照 ID 顺序将所有结点在逻辑上组织成一个“虚拟环”(virtual ring)，每个结点在虚拟环上维护 $r/2$ 个顺时针邻居和 $r/2$ 个逆时针邻居，合起来称为 vset，很明显图 9.4.3 中 VRR 的 $r=4$。作为一切路由的基础，每个结点还维护其物理上的邻居集 pset，依靠无线广播即可做到。Virtual Ring 上的逻辑连接通过类似距离向量的方法来维护，比如对结点 8F6 而言，它保存到 vset 中每个邻居的下一跳结点，如果 8F6 要走到 90E，它将循着这些下一跳结点逐渐走到 90E(图中共走过 5 跳)，这样一条路由路径称为一条 vset-path。为使得 vset-path 的构建成为可能，每个结点的路由表中还保存着其他结点间穿过它的路由路径，这一点非常重要。

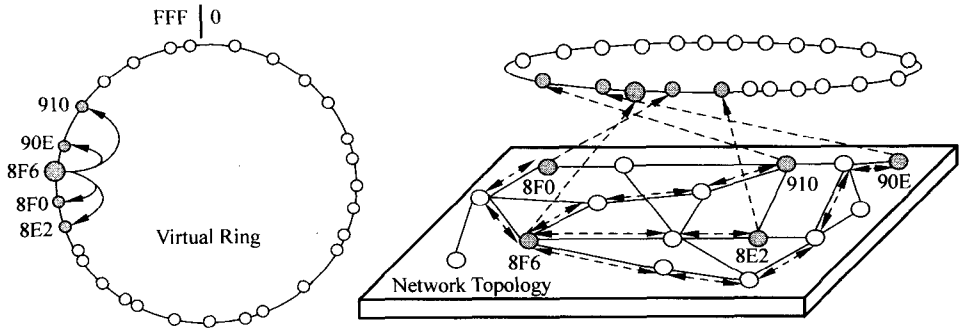


图 9.4.3 VRR 的“Virtual Ring”和 MANET 网络拓扑的对应关系
(图片来自[Caesar et al.，2006])

看到这里读者也许体会不到 VRR 的独特之处，并且很容易认为 Virtual Ring 只不过是 Chord 或者 Pastry 的一个简化版本。然而事实并非如此，VRR 的独特之处在于：它完全工作在网络层，本身就是一个网络层路由协议，Virtual Ring 所对应的 vset-paths 完全靠 VRR 自身来建立和维护，这和 P2P 网络中逻辑连接靠底层 IP 网络支持有本质区别。另一方面与传统的 Ad hoc 路由相比，VRR 的优势在于：路由的建立、维护不需要洪泛广播，也不需要额外的机制来提供结点 ID 和其地址间的转换。对于 VRR 具体的路由表维护、结点加入、连接失效、分环合并的处理方法，限于篇幅这里不做详述，读者可参考原论文。

9.4.2 P2P与网格计算的融合

网格计算(grid computing)同P2P类似,也是近些年伴随着Internet迅速发展起来的一种分布式系统工作模式。网格利用Internet把分散在不同地理位置的计算机(也称"结点")组织成一台虚拟的超级计算机,而整个计算是由成千上万个结点组成的一张网格。以网格方式组织成的虚拟的超级计算机有两个优势:一个是数据处理能力超强;另一个是能充分利用网上的闲置处理能力即"网络边缘资源"。

网格计算的领军人物、美国阿冈国家实验室的资深科学家、Globus项目(下文将讲到)的领导人Ian Foster在1998年出版了《网格:21世纪信息技术基础设施的蓝图》一书,标志着网格以明确的形态进入计算机领域。2002年7月,Ian Foster在《什么是网格?判断网格的三个标准》一文中限定"网格"必须同时满足三个条件:①在分布式的环境中协同使用资源;②使用标准的、开放的和通用的协议和接口;③提供多样化的服务质量。这三个条件明确地划定了网格的界限。

2002年8月,IBM宣布投入数十亿美元研发网格计算,与Globus合作开发开放的网格计算标准,于是网格计算和Globus一起从幕后走到前台,受到前所未有的关注。Globus是美国阿冈国家实验室的研发项目,全美国有12所大学和研究机构参与了该项目。Globus对资源管理、安全、信息服务及数据管理等网格计算的关键理论进行研究,开发能在各种平台上运行的网格计算工具软件(Toolkit),帮助规划和组建大型的网格试验平台,开发适合大型网格系统运行的大型应用程序。IBM与Globus共同发布的"开放网格服务体系"OGSA(Open Grid Services Architecture)勾勒了Globus Toolkit 3.0的蓝图。OGSA主要是将Web Services、数据库存取、J2EE等技术规范纳入网格计算。

从上面对网格计算的介绍可以看出:网格计算和对等计算以不同的方法组织大规模、分布式的资源(包括计算能力、存储资源和带宽等),两者的研究领域有不少共通和重叠之处(比如都在Internet之上构建了覆盖网),又有很好的互补性,正如"网格之父"Ian Foster所说:"网格计算建立了体系结构,但没有解决结点失效问题;对等计算解决了结点失效问题,但没有建立体系结构。"[Foster & Iamnitchi,2003]所以P2P与网格计算的融合,很可能是一个双赢的趋势。此外我们也应看到,P2P与Grid之间的区别仍然很多,所以要真正实现两者的融合还需要细致的设计与分析、远非一蹴而就的易事。表9.4.1总结了Grid与P2P的区别。

表 9.4.1 Grid 与 P2P 的区别

目　　的	Grid	P2P
	共享虚拟的超级计算机	共享 Internet 边缘资源
资源特性	同构	异构
网络环境	稳定、高速	动态、混杂
成员特点	正规、可信	混杂、不可信
网络规模	中等规模	大规模
体系结构	层次化	扁平、层次化
资源管理	集中式	分布式
协议接口	开放、标准、通用	封闭、私有、专用

9.5 P2P 的未来

9.5.1 P2P 未来的商业应用趋势

从 1999 年 P2P 的先驱 Napster 推出音乐共享软件算起，到后来的 Gnutella、KaZaA、BitTorrent、eDonkey 等一系列流行的 P2P 文件共享软件，再到 P2P 语音传输软件 Skype、P2P 网络电视软件 PPLive、P2P 协同工作软件 Groove 等，我们看到 P2P 的应用正越来越广、越来越好。

BitTorrent 在文件共享领域的统治地位短期内不会动摇，提供 BT 种子下载和搜索的网站还会不断出现，但很快将达到饱和。eDonkey 在文件共享领域仍然会比较流行，但如果监管不力，eDonkey 所引起的社会问题很可能成为它自己的定时炸弹，一个迫切的要求是 eDonkey 应该提供方法限制非法、色情内容的传播。国内一大批采用 Napster 模式的 P2P 文件共享软件和 P2P 网络电视软件，还有很大的发展空间，但如果它们一味追求用户群的扩张，而忽视了本身功能的完善，其前途是令人担忧的。优秀的 P2P 语音传输软件 Skype 将能拥有更多的用户，它对传统的电信领域将产生多大的冲击还很难说，也许行政性的干涉会牵扯进来。

商业领域大规模的收购和合并在 P2P 领域还会出现，而且不止一两个案例，另一种趋势是不同 P2P 网络之间的协议兼容，但目前看来这方面进展缓慢。商品化的 P2P 搜索引擎很可能出现，或者 P2P 技术会应用到目前已有的 Web 搜索引擎(如 Google)上。到目前为止，P2P 软件的编写、开发多是自发、无关联的，人们可能会发布某个成熟、完善的 P2P 开发平台，并逐步推广基于此平台来开发 P2P 应用的方法。

9.5.2 P2P 未来的学术研究趋势

从 P2P 产生 6 年来计算机领域的各大会议、刊物特别是 P2P 专业会议发表的

论文上看，P2P的研究重点已经从核心机制逐渐转向增强机制，目前的研究热点集中在以下几个方面：P2P网络中的语义模糊查询、容错性、拓扑意识与一致性问题、声誉和安全性问题。除此之外，对于P2P模拟、仿真的重视程度也在日渐增加。

“语义模糊查询”仍然是P2P最重要的研究课题之一。“拓扑意识和一致性问题”仍然是P2P的一项研究重点，因为它是影响P2P网络性能的重要因素。“声誉”/“信任”机制开始成为P2P领域的一个热点，它对P2P网络的效率、安全都很有意义；同样地，P2P的安全性将一直为人们所关心。从更宽的范围来看，P2P技术和其他技术的结合，P2P领域和其他领域（如网格计算、无线网络等）的融合，是一个必然的趋势。

从更实际的角度看，结构化P2P网络最迫切需要的，是向公众推出一个实用化的软件，一个有说服力、有征服力的“杀手锏”(Killer Application)。在学术界日益膨胀、日益浮躁、论文泡沫化的今天，很多学者的研究脱离实际应用而变成“面向论文”的，其研究常常是“先造桥、后找河”，或者忽视开发实现的难度而变成堆叠理论、不切实际的空中楼阁。学术界对P2P长期以来的理论研究绝大多数停留在论文层次上，并且对很多问题一直是众说纷纭难有定论，这种状况很像《庄子》中的一句话：“天下大乱，圣贤不明，道德不一。天下多得一察焉以自好。”我们认为这种混乱状况的本质原因正在于P2P研究者过分看重理论成果，而忽视了实际开发；人们希望看到的，是从理论成果中转化出来的现实、可用的软件产品，是希望P2P真的做到让网络“Peer-to-Peer”，而非“Paper-to-Paper”。

9.5.3 结束语：探讨P2P的未来

《菜根谭》首句讲：“欲做精金美玉的人品，定从烈火中煅来。”P2P是一个年轻的领域，也是一个充满问题、充满挑战，也就充满机遇的领域。不应挑剔它哪些还没有做到，而应肯定它已经做到了什么。虽然有太多的问题和障碍，但是从Napster到BitTorrent，从KaZaA到eDonkey，从Freenet到Skype，再到Chord、CAN、Tapestry、Pastry等，P2P从无到有，从一个小软件到改变Internet面貌、改变人们交流方式的一大领域，这一过程是值得欣慰的；其中蕴藏的力量和意义，值得我们鼓起勇气去大胆探索，值得我们怀着热情去深入思考，值得我们付出耐心去努力开发。

在本书的最后，作者希望引用康德先生《纯粹理性批判》第二版序言的结尾作为本书的结束语——“如果一个理论本身具有持久性，那么，最初给它带来极大危险的作用和反作用随着时间的推移就只会有助于磨平其不平整之处，而如果是无偏见、有洞察力、真正享有盛名的人来从事这一工作，则也可以在短时间内使它获得所要求的完美。”

参考文献

[Ab erer & Despotovic,2001]Karl Aberer,Zoran Despotovic. 2001. Managing Trust in a Peer-2-Peer Information System. In: Proc Tenth Int'l Conf on Information and Knowledge Management(CIKM01),ACM Press,310-317

[Adar & Huberman,2000]Eytan Adar,Bernardo A. Huberman. 2000. Free Riding on Gnutella. First Monday,5(10),October,134-139

[Bindel et al., 2000] David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi,Sean Rhea, Hakim Weatherspoon, Westley Weimer, Christopher Wells, Ben Y. Zhao,John Kubiatowicz. 2000. OceanStore: An Extremely Wide-Area Storage System. UC Berkeley Technique Report

[Caesar et al., 2006] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Sheal, Antony Rowstron. 2006. Virtual Ring Routing: Network Routing Inspired by DHTs. In: SIGCOMM'06,351-362

[Calvert et al., 1997] Kenneth L. Calvert, Matthew B. Doar, Ellen W. Zegura. 1997. Modeling Internet Topology. IEEE Communications Magazine,35(6),160-163

[Chawathe et al., 2003] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, Scott Shenker. 2003. Making Gnutella-like P2P Systems Scalable. In: SIGCOMM,03,407-418.

[Chen et al.,2004]Hua Chen,Mao Yang,et al.,2004. Maze: A Social Peer-to-peer Network. In: IEEE Int'l Conf on e-Commerce Technology for Dynamic e-Business (CEC-EAST'04), Beijing,China,290-293

[Clarke et al., 2000] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong. 2000. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In: Proc Int'l Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, Berlin: Springer,46-66

[Cohen & Shenker, 2002] Edith Cohen, Scott Shenker. 2002. Replication Strategies in Unstructured Peer-to-Peer Networks. In: SIGCOMM'02,84-95

[Cohen, 2003]Bram Cohen. 2003. Incentives Build Robustness in BitTorrent. In: Workshop on Economics of P2P Systems, Berkeley, CA, USA, Vol. 6

[Crespo & Garcia-Molina, 2002]Arturo Crespo, Hector Garcia-Molina. 2002. Routing Indices for Peer-to-Peer Systems. In: ICDCS'02, 23-32

[Dabek et al., 2001]Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica. 2001. Wide-area cooperative storage with CFS. In: SOSP'01, 202-215

[Dabek et al., 2003]Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, Ion Stoica. 2003. Towards a Common API for Structured Peer-to-Peer Overlays. In: IPTPS'03, 33-44

[Damiani et al., 2002]Ernesto Damiani, De Captitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, Fabio Violante. 2002. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In: Proc 9th ACM Conf on Computer and Communication Security(CCS'02), New York: ACM Press, 207-216

[Delmastro, 2005]F. Delmastro. 2005. From Pastry to CrossROAD: Cross-layer ring overlay for ad hoc networks. In: Proc 3rd IEEE Int'l Conf on Pervasive Computing and Communications Workshops (PerCom)2005, 60-64

[Douceur, 2002]John R. Douceur. 2002. The Sybil Attack. In: IPTPS'02, 251-260

[Druschel & Rowstron, 2001]Peter Druschel, Antony Rowstron. 2001. PAST: A large-scale, persistent peer-to-peer storage utility. In: Proc 8th ACM Symp on Operating Systems principles(HotOS'01), New York: ACM Press, 188-201

[Eriksson et al., 2003] Jakob Eriksson, Michalis Faloutsos, Srikanth Krishnamurthy. 2003. PeerNet: Pushing Peer-to-Peer down the Stack. In: IPTPS'03, 268-277

[Foster & Iamnitchi, 2003] Ian Foster, Adriana Iamnitchi. 2003. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: IPTPS'03, LNCS 2735, Berlin: Springer, 118-128

[Freedman & Morris, 2002]Michael J. Freedman, Robert Morris. 2002. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In: Proc 9th ACM Conf on Computer and Communications Security(CCS'02), New York: ACM Press, 193-206

[Freedman et al., 2002] Michael J. Freedman, Emil Sit, Josh Cates, Robert Morris. 2002. Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer. In: IPTPS'02, 121-129

[Gil et al., p2psim] Thomer M. Gil, M. Frans Kaashoek, Jinyang Li, Robert Morris, Jeremy Stribling. p2psim: A Simulator for Peer-to-Peer Protocols. http://pdos. csail. mit. edu /p2psim/

[Gkantsidis et al., 2004]Christos Gkantsidis, Milena Mihail, Amin Saberi. 2004. Random Walks in Peer-to-Peer Networks. In: INFOCOM'04, 120-130

[Godfrey et al., 2004]Brighten Godfrey, Karthik Lakshminarayanan, Richard Karp, Ion Stoica. 2004. Load Balancing in Dynamic Structured P2P Systems. In: INFOCOM'04, 46-50

[Golle et al., 2001] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov. 2001. Incentives for Sharing in Peer-to-Peer Networks. In: Proc 3rd ACM Conf on Electronic Commerce (CEC'01), New York: ACM Press, 264-267

[Gummadi et al., 2002] Krishna P. Gummadi, Stefan Saroiu, Steven D. Gribble. 2002. King: Estimating Latency between Arbitrary Internet End Hosts. In: Proc SIGCOMM Internet

Measurement Workshop(IMW 2002), Marseille, France, November, 5-18

[Gummadi et al., 2003] Krishna P. Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, Ion Stoica. 2003. The Impact of DHT Routing Geometry on Resilience and Proximity. In: SIGCOMM'03, Karlsruhe, 381-394

[Guo et al., 2007] Deke Guo, Jie Wu, Honghui Chen, Xueshan Luo. 2007. Moore: An Extendable Peer-to-Peer Network Based Incomplete Kautz Digraph with Constant Degree. In: INFOCOM'07, 821-829

[Gupta et al., 2003] Indranil Gupta, Ken P. Birman, Prakash Linga, Al J. Demers, Robbert van Renesse. 2003. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In: IPTPS'03, 160-169

[Harren et al., 2002] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, Ion Stoica. 2002. Complex Queries in DHT-based Peer-to-Peer Networks. In: IPTPS'02, 242-250

[Harvey et al., 2003a] Nicholas J. A. Harvey, John Dunagan, Michael B. Jones, Stefan Saroiu, Marvin Theimer, Alec Wolman. 2003. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In: Proc 4th USENIX Symp on Internet Technologies and Systems (USITS'03), Seattle, WA, 113-126

[Harvey et al., 2003b] Nicholas J. A. Harvey, Michael B. Jones, Marvin Theimer, Alec Wolman. 2003. Efficient Recovery from Organizational Disconnects in SkipNet. In: IPTPS'03, 183-196

[He et al., 2003] Qi He, Mostafa Ammar, George Riley, Himanshu Raj, Richard Fujimoto. 2003. Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. In: Proc 11th Int'l Symp on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems(MASCOTS'03), 71-78

[Hoßfeld et al., 2004] Tobias Hoßfeld, Kenji Leibnitz, Rastin Pries, Kurt Tutschku, Phuoc Tran-Gia, Krzysztof Pawlikowski. 2004. Information Diffusion in eDonkey Filesharing Networks. In: Australian Telecommunication Networks and Applications Conference(ATNAC 2004), Sydney, Australia, December 2004

[Hu et al., 2003] Y. Charlie Hu, Saumitra Das, Himabindu Pucha. 2003. Exploiting the Synergy Between Peer-to-Peer and Mobile Ad Hoc Networks. In: Hot-OS IX, Lihue, HI, USA, 37-42

[Johnson & Maltz, 1996] David B. Johnson, David A. Maltz. 1996. Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing, 353: 153-181

[Kaashoek & Karger, 2003] M. Frans Kaashoek, David R. Karger. 2003. Koorde: A Simple Degree-optimal Distributed Hash Table. In: IPTPS'03, 640-651

[Kamvar et al., 2003] Sepandar D. Kamvar, Mario T. Schlosser, Hector Garcia-Molina. 2003. The EigenTrust Algorithm for Reputation Management in P2P Networks. In: Proc 12th Int'l Conf on World Wide Web(WWW'03), 640-651

[Karger et al., 1997] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahy. 1997. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In: Proc ACM Symp on Theory of Computing, 654-663

[Karp et al., 2004] Brad Karp, Sylvia Ratnasamy, Sean Rhea, Scott Shenker. 2004. Spurring

Adoption of DHTs with OpenHash, a Public DHT Service. In: IPTPS'04, 195-205

[Kubiatowicz et al., 2000] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, Ben Y. Zhao. 2000. OceanStore: An Architecture for Global-Scale Persistent Storage. In: ASPLOS'00, 190-201

[Lamport et al., 1982] Leslie Lamport, Robert Shostak, Marshall Pease. 1982. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3): 382-401

[Liang et al., 2004] Jian Liang, Rakesh Kumar, Keith W. Ross. 2004. Understanding KaZaA. http://cis.poly.edu/~ross/papers/UnderstandingKaZaA.pdf

[Liang et al., 2005] Jian Liang, Rakesh Kumar, Keith W. Ross. 2005. The KaZaA Overlay: A Measurement Study. Computer Networks Journal, 50(6): 842-858

[Liao et al., 2006] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, Dafu Deng. 2006. AnySee: Peer-to-Peer Live Streaming. In: INFOCOM'06, 1-10

[Liu XM et al., 2006] Xiaomei Liu, Li Xiao, Andrew Kreling, Yunhao Liu. 2006. Optimizing Overlay Topology by Reducing Cut Vertices. In: NOSSDAV'06, 99-104

[Liu YH et al., 2004] Yunhao Liu, Xiaomei Liu, Li Xiao, Lionel M. Ni, Xiaodong Zhang. 2004. Location-aware Topology Matching in P2P Systems. In: INFOCOM'04, 2220-2230

[Liu YH et al., 2005] Yunhao Liu, Li Xiao, Xiaomei Liu, Lionel M. Ni, Xiaodong Zhang. 2005. Location Awareness in Unstructured Peer-to-Peer Systems. IEEE Transactions on Parallel and Distributed Systems, 16(2): 163-174

[Liu ZY et al., 2004] Zhiyu Liu, Guihai Chen, Chunfeng Yuan, Sanglu Lu, Chengzhong Xu. 2004. Fault Resilience of Structured P2P Systems. In: WISE 2004, LNCS 3306, 736-741

[Liu ZY et al., 2006] Zhiyu Liu, Ruifeng Yuan, Zhenhua Li, Hongxing Li, Guihai Chen. 2006. Survive under High Churn in Structured P2P Systems: Evaluation and Strategy. In: Int'l Conf on Computational Science(ICCS 2006), Reading, UK, May 28-31

[Loguinov et al., 2005] Dmitri Loguinov, Juan Casas, Xiaoming Wang. 2005. Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience. IEEE/ACM Transactions on Networking, 13(5): 1107-1120

[Lua et al., 2004] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim. 2004. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE Communications Survey and Tutorial, March, 72-93

[Lv et al., 2002] Qin Lv, Pei Cao, Edith Cohen, Kai Li, Scott Shenker. 2002. Search and Replication in Unstructured Peer-to-Peer Networks. In: Proc 16th Int'l Conf on Supercomputing(ICS'02), 84-95

[Mahajan et al., 2003] Ratul Mahajan, Miguel Castro, Antony Rowstron. 2003. Controlling the Cost of Reliability in Peer-to-Peer Overlays. In: IPTPS'03, 21-32

[Malkhi et al., 2002] Dahlia Malkhi, Moni Naor, David Ratajczak. 2002. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In: ACM PODC'02, 183-192

[Maymounkov & Mazieres, 2002] Petar Maymounkov, David Mazieres. 2002. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In: IPTPS'02, 53-65

[Medina et al., 2001] Alberto Medina, Anukool Lakhina, Ibrahim Matta, John Byers. 2001. BRITE: Universal Topology Generation from a User's Perspective. Boston University, BUCS-TR-2001-003, April 12

[Pandurangan et al., 2001] Gopal Pandurangan, Prabhakar Raghavan, Eli Upfal. 2001. Building low-diameter P2P networks. In: 42nd IEEE Symp on Foundations of Computer Science (FOCS 2001), 492-499

[Perkins & Bhagwat, 1994] Charles E. Perkins, Pravin Bhagwat. 1994. Highly Dynamic Destination-Sequenced Distance Vector-Routing (DSDV) for Mobile Computers. In: SIGCOMM'94, 234-244

[Perkins & Royer, 1999] Charles E. Perkins, Elizabeth M. Royer. 1999. Ad-hoc On-Demand Distance Vector Routing. In: Proc 2nd IEEE Workshop on Mobile Computing Systems and Applications, 90-100

[Plaxton et al., 1997] C Gred Plaxton, Rajmohan Rajaraman, Andrea W. Richa. 1997. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: SPAA'97, 314-326

[Pouwelse et al., 2004] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, H. J. Sips. 2004. A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System. In: Parallel and Distributed Systems Group, Delft University of Technology, The Netherlands

[Pouwelse et al., 2005] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, H. J. Sips. 2005. The BitTorrent P2P File Sharing System: Measurements and Analysis. In: IPTPS'05, 205-216

[Ramabhadran et al., 2004a] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, Scott Shenker. 2004. Prefix Hash Tree: An Indexing Data Structure over Distributed Hash Tables. http://berkeley.intel-research.net/sylvia/pht.pdf

[Ramabhadran et al., 2004b] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, Scott Shenker. 2004. Brief Announcement: Prefix Hash Tree. In: Proc 23rd ACM SIGACT-SIGOPS Symposium, 368

[Rao et al., 2003] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, Ion Stoica. 2003. Load Balancing in Structured P2P Systems. In: IPTPS'03, 68-79

[Ratnasamy et al., 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. 2001. A Scalable Content Addressable Network. In: SIGCOMM'01, 161-172

[Ratnasamy et al., 2002] Sylvia Ratnasamy, Scott Shenker, Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. In: IPTPS'02, 45-52

[Rhea et al., 2003] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y. Zhao, John Kubiatowicz. 2003. Pond: The OceanStore Prototype. In: Proc 2nd USENIX Conf on File and Storage Technologies (FAST'03), 1-14

[Rhea et al., 2005] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, Harlan Yu. 2005. OpenDHT: A Public DHT Service and Its Uses. In: SIGCOMM'05, 73-84

[Ripeanu, 2001] Matei Ripeanu. 2001. Peer-to-Peer Architecture Case Study: Gnutella Network. In: IEEE P2P'01, 99-100

[Ripeanu et al., 2002] Matei Ripeanu, Adriana Iamnitchi, Ian Foster. 2002. Mapping the Gnutella Network Properties of large scale peer-to-peer systems and implications for system design.

IEEE Internet Computing,6(2): 50-57

[Rizzo, 1997]Luigi Rizzo. 1997. Effective Erasure Codes for Reliable Computer Communication Protocols. ACM Computer Communication Review,27(2): 24-36

[Rodrigues & Liskov, 2005] Rodrigo Rodrigues, Barbara Liskov. 2005. High Availability in DHTs: Erasure Coding vs. Replication. In: IPTPS'05,226-239

[Rowstron & Druschel, 2001a] Antony Rowstron, Peter Druschel. 2001. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In: Proc 18th IFIP/ACM Int'l Conf on Distributed Systems Platforms(Middleware'01),LNCS 2218, Springer-Verlag,329-350

[Rowstron & Druschel, 2001b] Antony Rowstron, Peter Druschel. 2001. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In: SOSP'01, 188-201

[Royer & Toh,1999]Elizabeth M. Royer, Chai-Keong Toh. 1999. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. IEEE Personal Communications, 6(2): 46-55

[Saroiu et al.,2002]Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble. 2002. A Measurement Study of Peer-to-Peer File Sharing Systems. In: Proc Multimedia Computing and Networking(MMCN'02),156-170

[Saroiu et al.,2003]Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble. 2003. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. Multimedia Systems, 9(2): 170-184

[Sen & Wang, 2004] Subhabrata Sen, Jia Wang. 2004. Analyzing Peer-to-Peer Traffic Across Large Networks. IEEE/ACM Transactions on Networking,12(2),219-232

[Shen et al., 2004] Haiying Shen, Cheng-Zhong Xu, Guihai Chen. 2004. Cycloid: A Constant-Degree and Lookup-Efficient P2P Overlay Network. In: IPDPS'04, New Mexico, USA

[Sit & Morris, 2002] Emil Sit, Robert Morris. 2002. Security Considerations for Peer-to-Peer Distributed Hash Tables. In: IPTPS'02,261-269

[Stading et al.,2002]Tyron Stading, Petros Maniatis, Mary Baker. Peer-to-Peer Caching Schemes to Address Flash Crowds. In: IPTPS'02,203-213

[Stavrou et al., 2004] Angelos Stavrou, Dan Rubenstein, Sambit Sahu. 2004. A Lightweight, Robust P2P Systems to Handle Flash Crowds. IEEE Journal on Selected Areas in Communications,22(1): 6-17

[Stoica et al., 2001] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, Hari Balakrishnan. 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: SIGCOMM'01,149-160

[Stoica et al.,2003]Ion Stoica, Robert Morrisz, David Liben-Nowell, David R. Kargerz, M. Frans Kaashoekz, Frank Dabekz, Hari Balakrishnanz. 2003. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking,11(1): 17-32

[Tang et al., 2003] Chunqiang Tang, Zhichen Xu, Sandhya Dwarkadas. 2003. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: SIGCOMM'03, 175-186

[Ting & Deters, 2003] Nyik San Ting, Ralph Deters. 2003. 3LS—A Peer-to-Peer Network Simulator. In: IEEE P2P'03, 212-213

[Tutschku 2004] Kurt Tutschku. 2004. A Measurement-based Traffic Profile of the eDonkey Filesharing Service. In: Proc 5th Passive and Active Measurement Workshop(PAM'04), 12-21

[Weatherspoon & Kubiatowicz, 2002] Hakim Weatherspoon, John D. Kubiatowicz. 2002. Erasure Coding vs. Replication: A Quantitative Comparison. In: IPTPS'02, LNCS 2429, Springer-Verlag, 328-338

[Wells, 2002] Chris Wells. 2002. The OceanStore Archive: Goals, Structure, and Self-Repair. UC Berkeley Technique Report

[Witten et al., 1999] Ian H. Witten, Alistair Moffat, Timothy C. Bell. 1999. Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd ed. San Francisco: Morgan Kaufmann

[Wolpert & Macready, 1997] David H. Wolpert, William G. Macready. 1997. No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, 1(1): 67-82

[Wu et al., 2005] Fan Wu, Tongqing Qiu, Yuequan Chen, Guihai Chen. 2005. A Redundancy Schemes for High Availability in DHTs. In: ISPA2005, Nanjing, China. LNCS 3758, 990-1000

[Xiong & Liu, 2003] Li Xiong, Ling Liu. 2003. A Reputation-Based Trust Model for Peer-to-Peer eCommerce Communities. In: Proc ACM Conf on Electronic Commerce, 275-284

[Xiong & Liu, 2004] Li Xiong, Ling Liu. 2004. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. IEEE Transactions on Knowledge and Data Engineering, 16(7): 843-857

[Xu et al., 2003] Jun Xu, Abhishek Kumar, Xingxing Yu. 2003. On the Fundamental Tradeoffs Between Routing Table Size and Network Diameter in Peer-to-Peer Networks. In: INFOCOM'03, 2177-2187

[Yang et al., 2005] Mao Yang, Zheng Zhang, Xiaoming Li, Yafei Dai. 2005. An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. In: IPTPS'05, 182-192

[Zahn & Schiller, 2005] Thomas Zahn, Jochen Schiller. 2005. MADPastry: A DHT Substrate for Practicably Sized MANETs. In: Proc 5th Workshop on Applications and Services in Wireless Networks(ASWN'05)

[Zhao et al., 2001] Ben Y. Zhao, John Kubiatowicz, Anthony D. Joseph. 2001. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. UC Berkeley Technical Report: UCB/CSD-01-1141

[Zhao et al., 2004] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, John D. Kubiatowicz. 2004. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. IEEE Journal on Selected Areas in Communications, 22(1): 41-53

[Zhou et al., 2003a] Shuheng Zhou, Gregory R. Ganger, Peter Steenkiste. 2003. Locality-based Node IDs: Enabling Explicit Locality in DHTs. Technical Report of CMU-CS-03-171

[Zhou et al., 2003b] Shuheng Zhou, Gregory R. Ganger, Peter Steenkiste. 2003. Balancing Locality and Randomness in DHTs. Technical Report of CMU-CS-03-203

[Zhu & Hu, 2004] Yingwu Zhu, Yiming Hu. 2004. Semantic Search in Peer-to-Peer Systems. In: J. Wu, ed. Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-

to-Peer Networks, CRC Press

[Zhu et al., 2003] Yingwu Zhu, Honghao Wang, Yiming Hu. 2003. Integrating Semantics-based access Mechanisms with P2P File Systems. In: IEEE P2P'03, 118-125

[刘翰宇等, 2006]刘翰宇, 肖明忠, 代亚非, 李晓明. 2006. 活跃型用户对 P2P 文件共享系统可用性的影响. 软件学报, 17(10): 2087-2095

[罗杰文, 2005]罗杰文. Peer to Peer(P2P)综述. 2005 年 11 月 3 日. http://www.intsci.ac.cn/users/luojw/papers/p2p.htm

索　引

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z