网络技术实用丛书 Practical Network Technology

P2P 网络技术原理与C++开发案例

■ 张 文 赵子铭 主编
■ 杨天路 魏小康 等 编著

Technology Principle and C++ Development

■ 第9章中开发案例的源代码可到人民邮电出版社网站下载

■ 其他章节中的开发案例的源代码可到相应网站下载，下载地址在书中相应章节处已注明

人民邮电出版社
POSTS & TELECOM PRESS

CKOOK.COM
稀酷客
www.ckook.com

FORWARD

前 言

近几年来，基于 P2P 网络技术的网络应用迅速发展起来。最初，以 BT、Emule 等为代表的 P2P 网络下载软件，创造了飞速下载的纪录，较之以往的网络下载技术有了巨大的飞跃。近一两年，以 PPlive 等为代表而火热发展的网络视频技术依然离不开 P2P 网络的应用。许多调查显示，在因特网的流量中，有超过 50%的流量来自于 P2P 软件的使用。P2P 技术的这些特性使得 P2P 技术在普通的因特网用户中引起了极大的反响，从而为其带来了广泛的应用。

然而，P2P 技术的应用绝不仅限于此。研究人员认为，P2P 技术的应用将使网络上的资源得到充分利用和最大化的共享。P2P 技术在实时通信、协同工作、内容分发以及分布式计算等多个领域得到了应用。甚至有人认为，P2P 与其说是一种技术，更不如说是一种思想，有着改变整个因特网模式的思想。P2P 的一个重要特点就是改变因特网现在的以大网站为中心的状态，重返“非中心化”，并把资源共享的权力交还给用户。

目前，国内外已经从多个角度展开了对 P2P 技术的研究与应用，例如 P2P 路由寻址技术、P2P 文件共享技术、P2P 多媒体应用技术、P2P 流量监控技术及 P2P 安全技术等。自 2002 年起，美国国家科学基金会（NSF）提供了 1 200 万美元的资金启动了一个为期 5 年的研究项目 IRIS，该项目集中了 MIT 和 UC Berkeley 等 5 所美国著名高等院校的强大科研力量，为下一代大规模分布式应用研制基于 DHT 的新型基础设施。

基于 P2P 的网络应用系统容量大、扩散性强、计算资源利用率高，这些优点吸引了众多的研究人员和系统开发人员，以及大量对 P2P 网络技术感兴趣的工程技术人员。目前，关于 P2P 技术的英文书籍与论文很多，中文的相关资料却相对匮乏，这正是作者撰写本书的最重要原因之一。本书采用由浅入深以及技术原理与典型系统开发分析相结合的方式，向每一位对 P2P 技术感兴趣的读者全面地介绍了这一热门的研发领域。

本书分为基础理论篇与 C++开发案例篇两个部分。C++作为一种被广泛应用的编程语言，由于其面向对象的特性以及其语言本身相对的高效性，使其在 P2P 系统的开发中被大量使用。本书中介绍的所有案例都是以 C++为开发语言来实现的。

CKOOK.COM
稀酷客
www.ckook.com

基础理论篇（第 1～5 章）介绍了 P2P 网络的基本概念、网络结构、主流 P2P 应用系统架构、常见的路由搜索算法、P2P 应用所涉及的安全问题、P2P 应用所涉及的相关技术等 P2P 网络设计中的关键问题，学习这些内容可使读者基本掌握 P2P 网络理论，并为后续学习开发过程和系统分析扫除障碍。

C++开发案例篇（第 6～11 章）是本书的重点。这部分内容首先介绍了 P2P 应用开发的两个平台，然后针对目前应用比较广泛的几个 P2P 系统进行了详细的系统介绍和源码分析，其中包括 P2P 文件共享系统、P2P 即时通信系统、P2P 流媒体系统和 P2P 视频点播系统。

作为一本详细介绍并涉及 P2P 各热点研究方向的书籍，本书涵盖了 P2P 技术的基本概念、研究方向与主要研究热点，并给出了丰富的 C++开发案例供读者参考。

作者结合自己在 P2P 领域的研发经历并参考大量的国内外相关文献完成了本书的编写。本书由张文、赵子铭、杨天路、魏小康、胡春英等人共同完成。由于 P2P 网络技术所涉及的知识面极为广泛，而且技术的发展日新月异，加上作者学识水平有限，因此书中难免出现错误和疏漏，希望广大读者批评指正。

作　者

2008 年 5 月于北京

CKOOK.COM
稀酷客
www.ckook.com

CONTENTS

目 录

基础理论篇

第 1 章 P2P 基本概念 …… 3

1.1 P2P 网络的定义 …… 3

1.2 P2P 网络结构 …… 4

1.2.1 集中式 P2P 网络 …… 4

1.2.2 完全分布式非结构化 P2P 网络 …… 5

1.2.3 完全分布式结构化 P2P 网络 …… 6

1.2.4 混合式 P2P 网络 …… 7

1.2.5 P2P 网络和传统网络的对比 …… 8

1.3 P2P 网络的应用 …… 9

1.4 P2P 的发展 …… 11

1.4.1 起步 …… 11

1.4.2 发展 …… 11

1.4.3 高峰 …… 11

1.5 P2P 技术国内外研究现状 …… 12

1.5.1 国外相关研究 …… 12

1.5.2 国内研究现状 …… 12

1.6 本章总结 …… 13

1.7 练习题 …… 13

第 2 章 主流 P2P 系统 …… 15

2.1 文件共享类系统 …… 15

2.1.1 Napster …… 15

2.1.2 Gnutella …… 17

2.1.3 BitTorrent …… 19

2.1.4 eMule …… 20

2.1.5 Maze …… 22

2.2 即时通信类系统 …… 24

2.2.1 Skype …… 24

2.2.2 QQ …… 26

2.2.3 GTalk …… 27

2.3 流媒体类系统 …… 28

2.3.1 AnySee …… 28

2.3.2 PPLive …… 29

2.4 共享存储类系统 …… 30

2.4.1 OceanStore …… 30

2.4.2 Granary …… 32

2.5 对等计算类系统 …… 34

2.6 本章总结 …… 35

2.7 练习题 …… 35

第 3 章 P2P 网络的基础——搜索和路由算法 …… 36

3.1 传统搜索技术 …… 36

3.2 P2P 搜索技术的发展 …… 37

3.3 DHT 网络（结构化 P2P 网络）的搜索技术 …… 39

3.3.1 DHT 路由原理 …… 39

3.3.2 Chord …… 40

3.3.3 Pastry …… 44

3.3.4 CAN …… 46

3.3.5 Tapestry …… 48

3.3.6 Kademlia …… 50

3.3.7 小结 …… 53

CKOOK.COM
稀酷客
www.ckook.com

3.4 非结构化 P2P 网络的搜索技术 …… 53
3.4.1 Flooding …… 54
3.4.2 Modified-BFS …… 55
3.4.3 Iterative Deepening …… 55
3.4.4 Random Walk …… 56
3.4.5 Query Routing …… 57
3.4.6 Gnutella2 …… 59
3.4.7 移动 Agent …… 59
3.4.8 小结 …… 60
3.5 小世界（Small World）模型 …… 61
3.5.1 小世界模型概述 …… 61
3.5.2 聚类分布（CD）算法 …… 62
3.5.3 小世界网络的研究现状 …… 66
3.6 P2P 搜索技术研究的挑战 …… 66
3.7 本章总结 …… 67
3.8 练习题 …… 67

第 4 章 P2P 应用与安全 …… 69

4.1 P2P 应用面临的安全问题 …… 69
4.1.1 P2P 应用引发的版权问题 …… 69
4.1.2 P2P 应用对现有网络应用的威胁 …… 71
4.1.3 P2P 网络病毒与蠕虫 …… 71
4.1.4 结构化 P2P 网络的隐患 …… 72
4.2 P2P 应用与安全技术 …… 76
4.2.1 P2P 应用中的密码学技术 …… 76
4.2.2 P2P 应用中的网络安全技术 …… 78
4.2.3 利用 P2P 网络解决安全问题 …… 82
4.3 P2P 实例系统——Skype 的安全性分析 …… 88
4.3.1 Skype 简介 …… 89
4.3.2 Skype 安全机制分析 …… 93
4.3.3 Skype 流量识别 …… 96
4.4 本章总结 …… 98
4.5 练习题 …… 98

第 5 章 P2P 应用的相关技术 …… 100

5.1 P2P 与网络穿越 …… 100
5.1.1 NAT 网络概念 …… 100
5.1.2 STUN 协议与 NAT 穿越 …… 103
5.1.3 UDP 穿越 NAT …… 106
5.1.4 TCP 穿越 NAT …… 107
5.1.5 NAT 类型检测 …… 108
5.1.6 常见 NAT 穿越解决方案 …… 111
5.2 P2P 与 IMS 结合 …… 112
5.2.1 什么是 IMS …… 112
5.2.2 P2P 与 IMS 的网络融合 …… 112
5.3 VoIP 通信基础 …… 113
5.3.1 VoIP 系统概念 …… 113
5.3.2 H.323 协议 …… 115
5.3.3 SIP …… 115
5.3.4 RTP …… 117
5.3.5 RTSP …… 118
5.4 P2P 和 SIP 的结合 …… 119
5.4.1 P2P 和 SIP 结合方案分析 …… 120
5.4.2 基于 Pastry 设计 P2P-SIP 系统 …… 122
5.5 本章总结 …… 125
5.6 练习题 …… 126

C++开发案例篇

第 6 章 P2P 应用开发平台 …… 129

6.1 Windows Peer-to-Peer Networking 平台 …… 129
6.1.1 Windows Peer-to-Peer Networking 平台简介 …… 129
6.1.2 Windows Peer-to-Peer Networking 平台结构 …… 130
6.1.3 Windows Peer-to-Peer Networking 平台工作机制 …… 132
6.1.4 Windows Peer-to-Peer Networking 平台开发环境 …… 136
6.1.5 Windows Peer-to-Peer Networking 平台开发入门 …… 139
6.2 JXTA …… 161
6.2.1 JXTA 简介 …… 161

CKOOK.COM 稀酷客 www.ckook.com

6.2.2 JXTA 的平台结构 …… 162
6.2.3 JXTA 的基本概念 …… 163
6.2.4 JXTA 的网络架构 …… 166
6.2.5 JXTA 开发环境 …… 171
6.3 本章总结 …… 175
6.4 练习题 …… 175

第 7 章 P2P 文件共享系统开发实例——eMule 的设计与实现 …… 176

7.1 eMule 系统概述 …… 176
7.1.1 P2P 文件共享系统的背景 …… 176
7.1.2 eMule 的设计目标 …… 177
7.2 eMule 系统原理 …… 178
7.2.1 eMule 协议原理 …… 178
7.2.2 Kad 协议原理 …… 185
7.3 eMule 系统设计 …… 186
7.3.1 eMule 系统结构概述 …… 187
7.3.2 文件基础设施 …… 187
7.3.3 网络基础设施 …… 189
7.3.4 eMule 通信协议 …… 190
7.3.5 任务处理机制 …… 191
7.3.6 Kad 系统结构概述 …… 195
7.4 eMule 程序代码分析 …… 199
7.5 eMule 系统部署 …… 223
7.5.1 系统编译和运行 …… 224
7.5.2 系统安装与配置 …… 225
7.5.3 文件搜索和下载 …… 226
7.5.4 文件上传 …… 227
7.6 eMule 系统分析 …… 228
7.7 本章总结 …… 228
7.8 练习题 …… 228

第 8 章 P2P 文件共享系统开发实例二——BT 的设计与实现 …… 230

8.1 BitTorrent 系统概述 …… 230
8.2 BitTorrent 系统原理 …… 230
8.2.1 BitTorrent 协议原理 …… 231
8.2.2 BitTorrent Tracker 服务器原理 …… 234
8.3 BitTorrent 系统设计 …… 236
8.3.1 Arctic 客户端系统结构概述 …… 236
8.3.2 LibTorrent 库系统结构概述 …… 237
8.4 BitTorrent 程序代码分析 …… 238
8.4.1 Arctic 客户端代码分析 …… 238
8.4.2 LibTorrent 库代码分析 …… 260
8.5 BitTorrent 系统部署 …… 270
8.5.1 BitTorrent 服务器的部署 …… 271
8.5.2 BitTorrent 客户端的部署 …… 271
8.6 BitTorrent 系统分析 …… 271
8.7 本章总结 …… 272
8.8 练习题 …… 272

第 9 章 P2P 即时通信系统开发实例——Hermes 的设计与实现 …… 273

9.1 Hermes 系统概述 …… 273
9.1.1 P2P 即时通信系统的背景 …… 273
9.1.2 Hermes 系统设计目标 …… 274
9.2 Hermes 系统原理 …… 274
9.2.1 登录认证机制 …… 275
9.2.2 语音提取与播放机制 …… 277
9.3 Hermes 系统设计 …… 279
9.4 Hermes 程序代码分析 …… 280
9.4.1 语音处理模块代码分析 …… 280
9.4.2 通信模块代码分析 …… 295
9.5 Hermes 系统分析 …… 302
9.6 本章总结 …… 302
9.7 练习题 …… 302

第 10 章 P2P 流媒体系统开发实例——PeerCast 的设计与实现 …… 303

10.1 P2P 流媒体系统概念 …… 303
10.1.1 流媒体内容发布网络技术 …… 303
10.1.2 P2P 流媒体系统概述 …… 304
10.1.3 P2P 流媒体系统架构 …… 304
10.1.4 现有系统 …… 305
10.2 流媒体技术 …… 305
10.3 设计 P2P 音视频点播系统 …… 306
10.3.1 多媒体数据压缩 …… 306

www.ckook.com

10.3.2 应用层 QoS ····· 306
10.3.3 应用层多播技术····· 306
10.3.4 流媒体同步技术····· 307
10.3.5 PeerCast 实现分析····· 307
10.3.6 改造 BitTorrent 成为流媒体系统 ·····311
10.4 本章总结·····311
10.5 练习题·····311

第 11 章 P2P 视频点播系统开发实例——Myseelite 的设计与实现····· 313

11.1 Myseelite 系统概述 ····· 313
11.1.1 P2P 视频点播系统的背景····· 313
11.1.2 Myseelite 的设计目标····· 314
11.2 Myseelite 系统原理 ····· 315
11.2.1 基本概念····· 315
11.2.2 工作机制 ····· 316
11.2.3 ACE 简介 ····· 317
11.2.4 wxWidgets 简介····· 320
11.3 Myseelite 系统设计 ····· 320
11.3.1 Capture 子系统····· 320
11.3.2 Super Peer 子系统····· 321
11.3.3 Tracker 子系统 ····· 322
11.3.4 Client 子系统 ····· 324
11.4 Myseelite 程序代码分析 ····· 329
11.5 Myseelite 系统部署 ····· 353
11.5.1 系统编译 ····· 353
11.5.2 系统运行 ····· 353
11.5.3 轮播流程 ····· 354
11.5.4 直播流程 ····· 357
11.6 Myseelite 系统分析 ····· 359
11.7 本章总结 ····· 359
11.8 练习题 ····· 359

www.ckook.com

基础理论篇

www.ckook.com

www.ckook.com

CHAPTER 1

第 1 章　P2P 基本概念

要认识和理解 P2P 网络，就要理解 P2P 网络理论中的基本概念，这些概念和传统的服务器/客户端网络有很大的区别。

在学习了本章后，希望读者能够掌握以下几点：

（1）P2P 网络的概念和定义；

（2）P2P 网络的基本结构；

（3）P2P 网络的应用和发展。

1.1　P2P 网络的定义

P2P（Peer to Peer）即对等计算或对等网络，通常简称为 P2P，可以简单地定义成通过直接交换，共享计算机资源和服务。在 P2P 网络环境中，成千上万台彼此连接的计算机都处于对等的地位，整个网络一般来讲不依赖于专用集中服务器。网络中的每一台计算机既能充当网络服务的请求者，又能对其他计算机的请求做出响应，提供资源与服务。通常这些资源和服务包括信息的共享与交换、计算资源（如 CPU）的共享使用、存储资源（如缓存和磁盘空间）的使用等。

对于 P2P 概念的一种解释是 P2P 即 Peer to Peer。Peer 在英语里是“（地位、能力等）同等者”、“同事”和“伙伴”的意思。因此，P2P 也就可以理解为“端对端”的意思，或称为对等联网。

对于 P2P 的定义，不同的机构有着不同的理解，每种理解方式本质上并不矛盾，都从不同的侧面揭示了 P2P 网络的特点。本书仅列出比较典型的 5 种定义，如表 1-1 所示。

表 1-1　　P2P 多种定义

定 义 者	定　义
Intel 工作组	通过在系统之间直接交换来共享计算机资源和服务的一种应用模式
A.Weytsel	在因特网周边以非客户地位使用的设备
R.l.Granham	通过 3 个关键条件定义： ① 具有服务器质量的可运行计算机；

www.ckook.com

续表

定 义 者	定 义
R.I.Granham	② 具有独立于 DNS 的寻址系统； ③ 具有与可变连接合作的能力
C.Shirky	利用因特网边界的存储/CPU/内容/现场等资源的一种应用； 访问这些非集中资源意味着运行在不稳定连接和不可预知 IP 地址环境下，P2P 节点必须运行在 DNS 系统外边； 具备有效或全部的自治
Kindberg	独立生存的系统
D.J.Milojicic	给对等组提供或从对等组获得共享

1.2 P2P 网络结构

1.2.1 集中式 P2P 网络

集中式 P2P 网络形式上有一个中心服务器来负责记录共享信息以及回答对这些信息的查询。每一个对等实体对它将要共享的信息以及进行的通信负责，根据需要下载它所需要的其他对等实体上的信息。这种形式具有中心化的特点，但是它不同于传统意义上的 Client/Server（客户端/服务器）模式。传统意义上的 Client/Server 模式采用的是一种垄断的手段，所有资料都存放在服务器上，客户端只能被动地从服务器上读取信息，并且客户端之间不具有交互能力。其典型结构如图 1-1 所示。而集中式 P2P 网络则是将所有网上提供的资料都分别存放在提供该资料的客户端上，服务器上只保留索引信息，此外服务器与对等实体以及对等实体之间都具有交互能力。

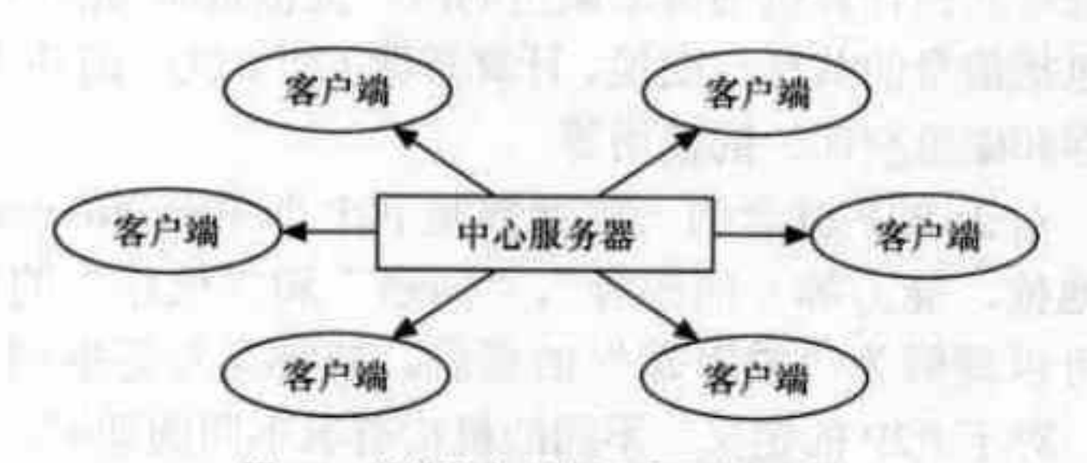


图 1-1 传统的客户端/服务器网络结构

采用集中式 P2P 形式的软件被称为第一代 P2P，其代表性软件为 Napster。Napster 的系统结构如图 1-2 所示。

这种形式有一个中央服务器，为用户提供共享和搜索文件服务。这就要求有一个连续运转的服务器，并且一旦该服务器被关闭，整个网络就会停止运行。此外，这样的服务器必须能够处理大量的用户连接，拥有足够的内存和磁盘空间来维护和搜索文件列表。虽然 Napster 流行一时，但是由于通过这个软件，大量的有版权的资料可以在网上轻易地获得，导致不久就被 RIAA（Recording Industry Association of America，美国唱片工业协会）告上法庭，最终

www.ckook.com

的判决结果是 Napster 停止运行，现在 Napster 已经停止运行了。之所以会有这样的结果是因为 Napster 的运行依赖于一个用来索引用户文件具有公司所有权的中心服务器。尽管 Napster 的使用与侵权、盗版等名词联系在一起，但我们不能因此就忽视 Napster 这一新兴领域所具有的潜力和广阔的应用前景，而阻止这一技术进步。为了解决由于外界的压力而有可能导致系统停止运行，产生了基于分布式的 P2P 形式。

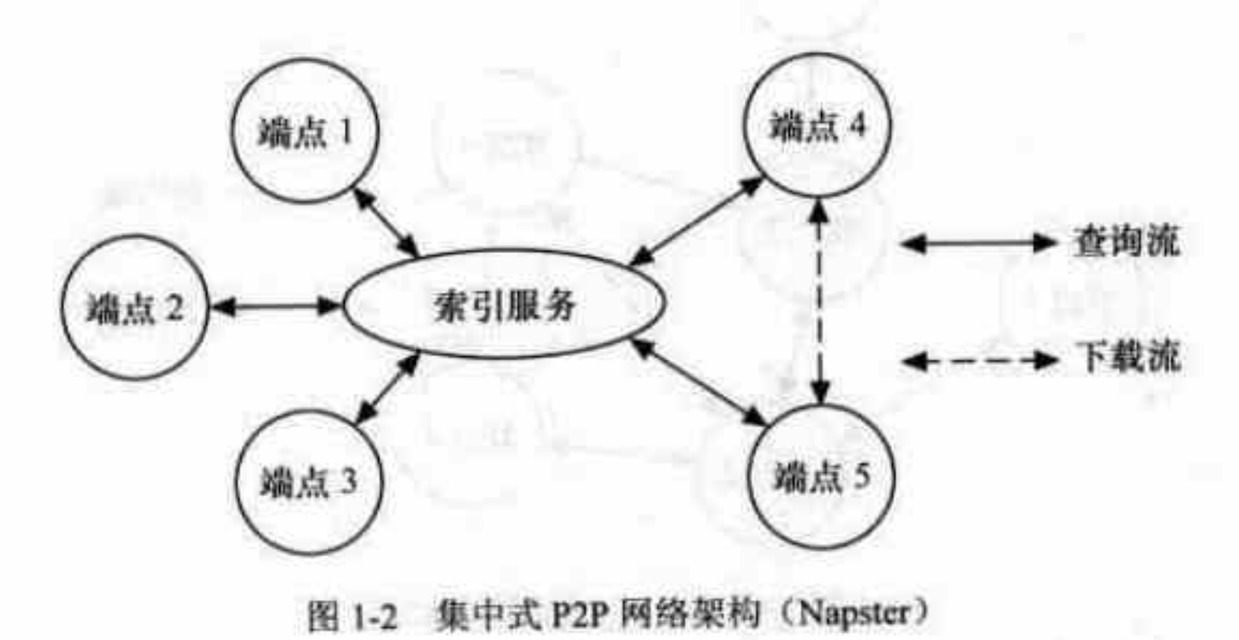


图 1-2 集中式 P2P 网络架构（Napster）

1.2.2 完全分布式非结构化 P2P 网络

完全分布式非结构化拓扑的 P2P 网络采用了随机图的组织方式来形成一个松散的网络，其典型网络结构如图 1-3 所示。这种结构对网络的动态变化有较好的容错能力，因此具有较好的可用性。同时，这种结构支持复杂查询，比如带有规则表达式的多关键字查询、模糊查询等。完全分布式非结构化拓扑的 P2P 网络的典型代表是 Gnutella。

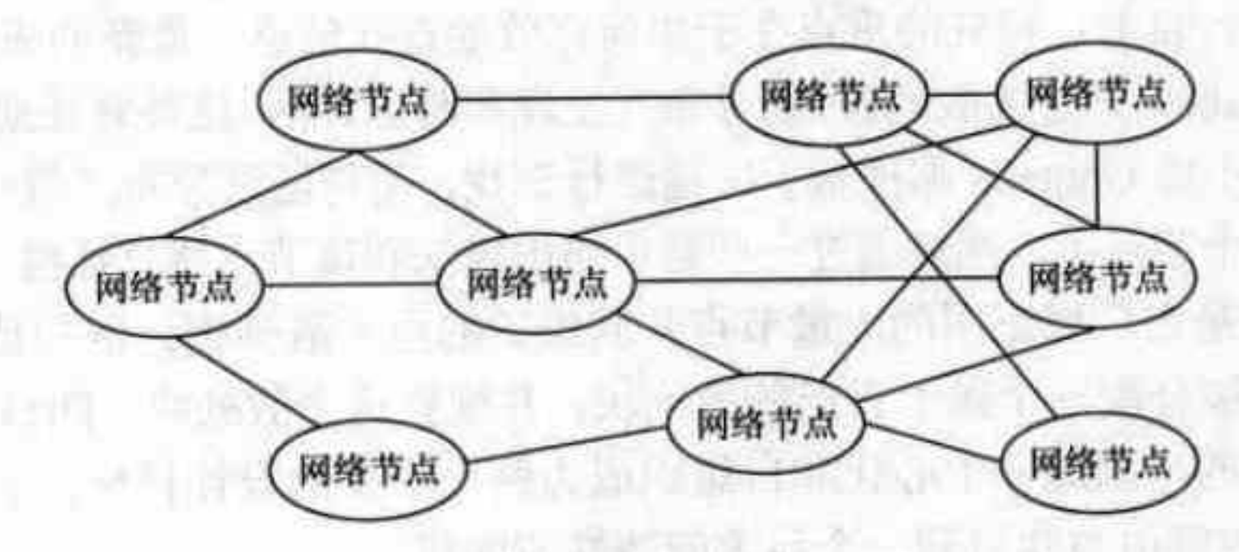


图 1-3 典型的分布式非结构化 P2P 网络

Gnutella 是一个 P2P 文件共享系统，它和 Napster 的最大区别在于 Gnutella 没有中心服务器，它采用了完全随机图的洪泛式搜索和随机转发机制。为了控制搜索消息的传输，Gnutella 网络采用类似 IP 数据包中 TTL 的机制来决定是否继续转发消息。Gnutella 的查询流程如图 1-4 所示。

在完全分布式非结构化拓扑的 P2P 网络模型中，每个节点都具有相同的功能，既是客户端又是服务器，因而节点也被称为对等点。

这种拓扑的优点是网络配置简单，不需要服务器的支持，在网络规模较小的时候具有很高的查询效率。但由于在这种拓扑的网络中多采用洪泛方式查询和定位资源，随着联网节点的增加，网络规模不断增大，从而给网络带来了沉重的网络负载。而且由于没有确定的拓扑，

www.ckook.com

这种形式的网络无法保证查找资源的确定性，即可能会漏过网络中的一些资源。另一个问题是由于采用了 TTL（Time To Live，生存时间）、洪泛、随机漫步或有选择转发算法，这个拓扑的网络直径不可控，可扩展性较差。

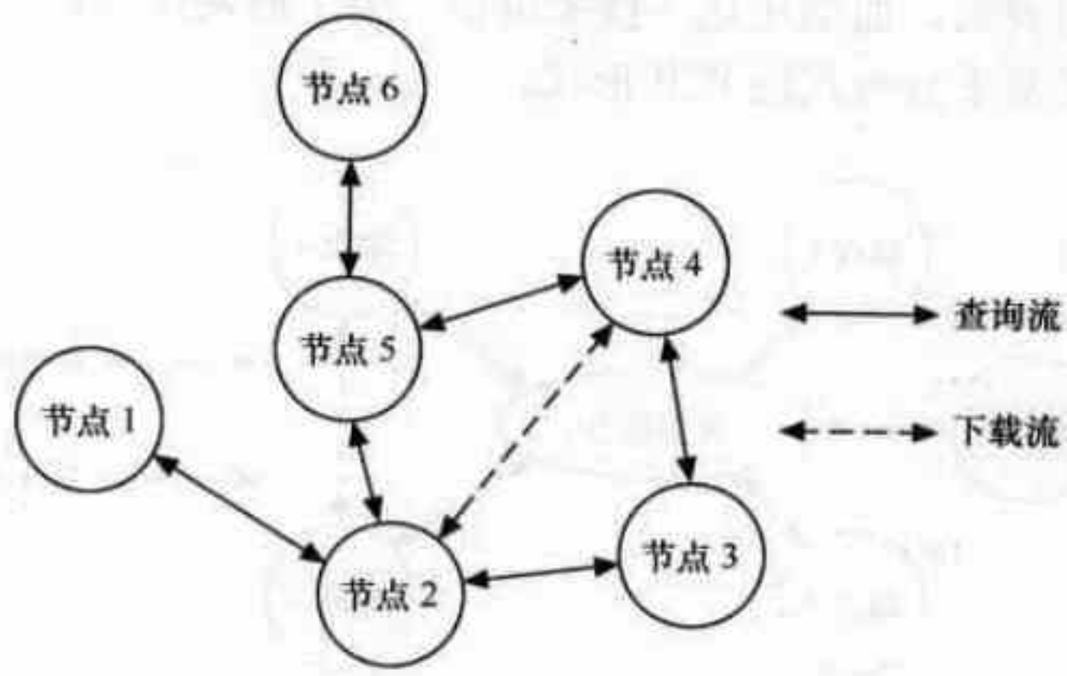


图 1-4　Gnutella 的查询流程

由于发现的准确性和可扩展性是完全分布式非结构化网络面临的两个重要问题，因此目前对这种结构网络的研究集中在改进发现算法和转发策略上。本书的后续章节将在这方面予以详细讲解。

1.2.3　完全分布式结构化 P2P 网络

由于非结构化拓扑的 P2P 网络中随机搜索造成的不可扩展性，人们开始研究如何构造一个高度结构化的系统。目前，研究的重点在于如何有效地查找信息，最新的成果就是基于 DHT（Distributed Hash Table，分布式散列表）的分布式发现和路由算法。这种算法避免了类似 Napster 的中心服务器，也不像 Gnutella 那样基于广播进行查找，而是通过分布式散列函数将输入的关键字唯一映射到某个节点上，然后通过一些特定路由算法和该节点建立连接。

分布式散列表是由广域范围的大量节点共同维护的巨大散列把。散列把被分割成不连续的块，每个节点都被分配一个属于自己的散列块，并维护这个散列块。DHT 的节点既是动态的，也是数量巨大的，因此非中心化和自组织成为两个重要的设计目标。通过散列函数，节点的 ID 还有资源的键值被映射到一个巨大的线性空间里。

在 DHT 技术中，网络节点按照一定的方式分配一个唯一节点标识符（Node ID），资源对象通过散列运算产生一个唯一的资源标识符（Object ID），且该资源将存储在节点 ID 与之相等或者相近的节点上。需要查找该资源时，采用同样的方法可定位到存储该资源的节点。

DHT 类结构能够自适应节点的动态加入/退出，有着良好的可扩展性、顽健性、节点 ID 分配的均匀性和自组织能力。由于重叠网络采用了确定性拓扑结构，DHT 可以提供精确的发现。只要目的节点存在于网络中 DHT 总能发现它，发现的准确性得到了保证。

DHT 类结构最大的问题是 DHT 的维护机制较为复杂，尤其是节点频繁加入退出造成的网络波动（Churn）会极大增加 DHT 的维护代价。DHT 所面临的另外一个问题是 DHT 仅支持精确关键字匹配查询，无法支持内容/语义等复杂查询。

目前已有的分布式结构化拓扑的 P2P 网络有 Pastry、Tapestry、Chord 和 CAN。本书的后

www.ckook.com

续章节将详细介绍这几个 P2P 网络的结构和路由算法等特性。

1.2.4 混合式 P2P 网络

集中式 P2P 形式有利于网络资源的快速检索，以及只要服务器能力足够强大就可以无限扩展，但是其中心化的模式容易遭到直接的攻击，分布式 P2P 形式解决了抗攻击问题，但是又缺乏快速搜索和可扩展性。混合 P2P 形式结合了集中式和分布式 P2P 形式的优点，在设计思想和处理能力上都得到进一步优化。它在分布式模式基础上，将用户节点按能力进行分类，使某些节点担任特殊的任务。Skype 即时通信软件就使用了混合式 P2P 网络模式。通常的混合式 P2P 系统结构图和查询流图分别如图 1-5 和图 1-6 所示。混合式 P2P 网络总共包含 3 种节点，分别是用户节点、搜索节点和索引节点。

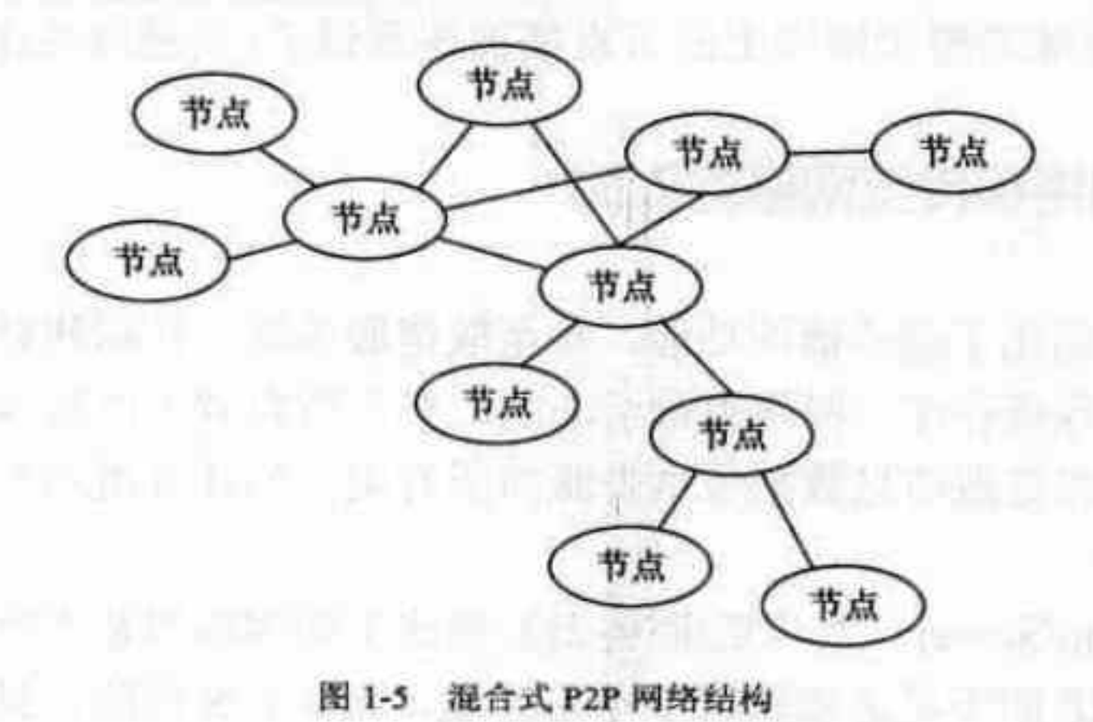


图 1-5 混合式 P2P 网络结构

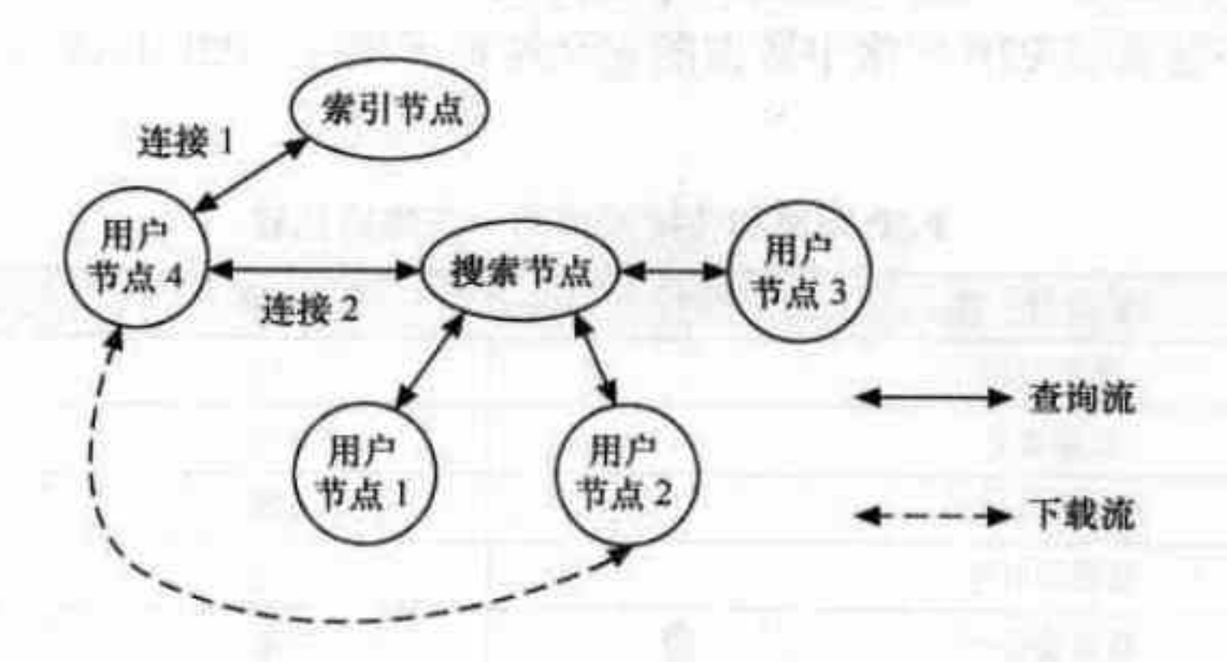


图 1-6 混合式 P2P 网络查询流程

（1）用户节点

普通的节点就是用户节点，它不具有任何特殊的功能。

（2）搜索节点

搜索节点处理搜索请求，从它们的孩子节点中搜索文件列表，这些节点必须有 128kbit/s 以上的网络连接速度。因此，最好用高性能的处理器。

（3）索引节点

连接速度快、内存充足的节点可以作为索引节点。索引节点保存可以利用的搜索节点信息、搜集状态信息以及尽力维护网络的结构。一个节点可以既是搜索节点又是索引节点。用

www.ckook.com

户节点可以选择 3 个搜索节点作为它的父节点，如果父节点接受该用户节点作为它的孩子的话，那么该用户节点就可以提交它要共享的列表给它的父节点。默认的情况下，搜索节点可以最多维护 500 个孩子节点。

在第三代 P2P 的软件体系结构中，采用了混合 P2P 形式。这种形式的关键之一是引入了索引节点，索引节点不会直接连接到有版权的资料，它就像 Yahoo！一样，只是搜索和所需资料相关的地址，至于用户到底连接下载了什么内容和它无关。这种形式的关键之二是引入搜索节点，搜索节点管理着所属用户的文件列表。用户节点通过索引节点获得搜索节点信息，之后用户节点就与获得的搜索节点相连，每一次查询都通过该搜索节点进行。当用户发出搜索请求后，如果和用户节点直接相连的搜索节点查询结果为 100 个（这里的 100 个搜索结果可以由用户自己来设定）就停止，如果不足 100 个，就向相邻的搜索节点发出请求，如果查询结果还不够，就继续向外快速发散，直到所有的搜索节点都被搜索到。而如果所有的搜索节点都被访问过，就意味着整个网络上的节点都被搜索到了，其速度要比纯 P2P 模式快得多。

1.2.5 P2P 网络和传统网络的对比

在 P2P 网络中，弱化了服务器的功能，甚至取消服务器，任意两台 PC 互为服务器/客户端，即使只有一个对等点存在，网络也是活动的，节点所有者可以随意地将自己的信息发布到网络上。P2P 技术将导致信息数据成本资源向所有用户的计算机均匀分布，即“边缘化”趋势。

首先，C/S（Client/Server，客户端/服务器）模式下的因特网是完全依赖于中心点服务器的，没有服务器的 P2P 的不足之处就在于不易管理，而对 C/S 网络，只需要在中心点进行管理，从而带来的不足就是 P2P 网络中数据的安全性难于保证。P2P 技术与 C /S 技术性能比较如表 1-2 所示。

表 1-2　　P2P 网络和传统网络的一些性能比较

性 能 比 较	P2P	C/S
数据发布	好	差
数据接受	中	好
数据互动性	好	差
数据即时性	好	差
数据安全性	差	好
数据更新	好	差
数据质量	中	好
数据覆盖率和数量	差	好
数据成本控制	好	差
数据管理方便性	差	好

在传统的 C/S 模式中，客户端之间要进行文件交换必须经过服务器，随着节点的增加，服务器的负担越来越重，并逐渐形成系统瓶颈，一旦服务器崩溃，整个网络也随之瘫痪。而在 P2P 网络中每个节点的地位都是对等的，每个节点既充当服务器，为其他节点提供服务，同时也充当客户端，享用其他节点提供的服务。同时，由于每个节点在工作时都在向网络贡

www.ckook.com

献资源（存储空间，CPU 周期等），因此对等点越多，网络的性能就越好。

其次，C/S 模式下的因特网完全依赖于中心点——服务器，没有服务器就根本无法工作，网络也就毫无意义可言。而在 P2P 网络中，每个节点都可以看作服务器，可以随意地将自己的信息发布到网络上，以供其他节点交换共享。

再次，在 C/S 模式下即使客户端有大量的闲置资源，如果没有得到服务器的响应，资源也无法被利用，资源利用率比较低下。而在 P2P 网络中，一切闲散资源都有机会得到利用，具有很高的资源利用率。C/S 模式与 P2P 模式的对比如图 1-7 和图 1-8 所示。

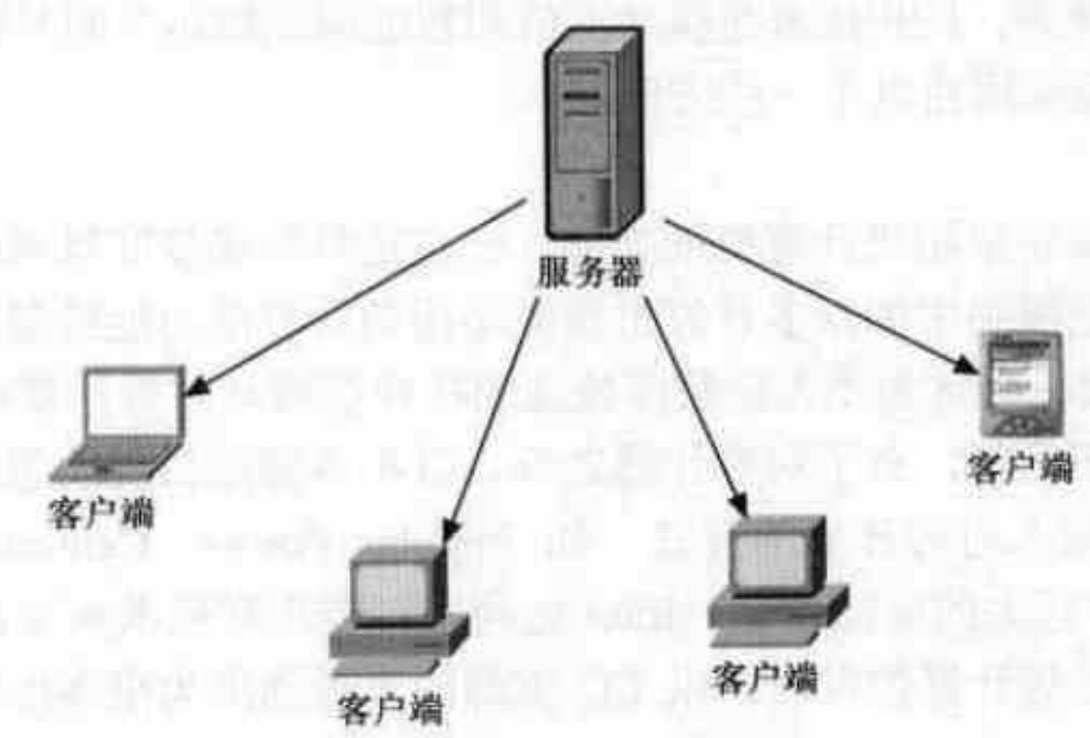


图 1-7　传统网络工作模式

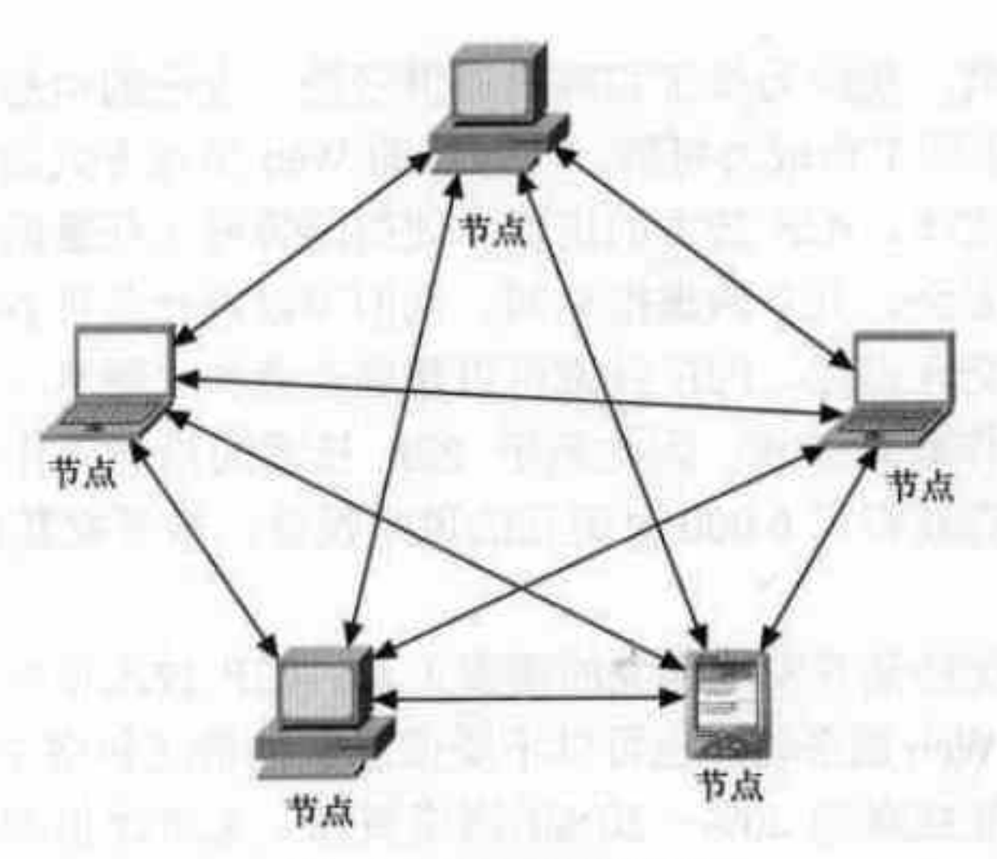


图 1-8　P2P 网络工作模式

1.3　P2P 网络的应用

Internet 最初产生和发展的一个主动力就是资源共享，也正是文件交换的需求直接导致了 P2P 技术的兴起，这是 P2P 最初也是最成功的应用之一，也正是针对这类应用的 Napster 使得人们在客户端/服务器模式下开始重新认识 P2P 思想对人们使用网络习惯的影响。

随着人们对 P2P 思想的理解和技术的发展，作为一种软件架构，P2P 还可以被开发出种

www.ckook.com

类繁多的应用模式，除了最初的文件交换之外，还出现了一些分布式存储、深度搜索、分布式计算、个人即时通信和协同工作等新颖应用。其中最著名的例子是基于分布式计算的搜索外星文明 SETI@home 科学实验，每个志愿参加者只需下载并运行类似屏幕保护的方式，就可以贡献自己闲置的计算能力，参与分析 Arecibo 射电望远镜的无线电磁波数据并回送计算数据，截至 2004 年 12 月，已有 528 万志愿者参加进来，获得了相当于 216 万年的 CPU 时间，仅一天的综合计算就相当于 67.46T 次浮点运算。另外，随着 SUN 公司将其 JXTA 协议扩展到诸如个人数字助理（PDA）和移动电话等手持终端上，并允许人们屏蔽具体的物理平台进行资料共享和文件交换等，P2P 技术在移动通信和智能网领域也开始呈现出较大应用前景。

P2P 网络技术主要应用在以下一些方面。

（1）对等计算

通过众多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，正是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、基因组的研究等，有了对等计算之后，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在投入对等计算的开发，如 Popular Power、Centrata、United Devices、Entropia 等，并获得了巨大的风险资金。Intel 也利用对等计算技术来设计其 CPU，并为其节省极大的费用，同时对等计算的发展是以 PC 资源的有效利用为根本出发点的，因此也受到 Intel 的极力推崇。从本质上讲，对等计算就是网络上 CPU 资源的共享。

（2）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、方便的消息和协作的工具变得日益重要。网络的出现，使协同工作成为可能。但传统的 Web 实现方式给服务器带来了极大的负担，造成了昂贵的成本支出。P2P 技术的出现，使得因特网上任意两台 PC 都可建立实时的联系，建立了这样一个安全、共享的虚拟空间，人们可以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 技术可以帮助企业和关键客户，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 技术的协同工作也受到了极大的重视。Lotous 公司的创始人奥奇获得了 6 000 万美元的风险投资，来开发其协同工作产品 Groove。

（3）搜索引擎

P2P 技术的另一个优势是开发出强大的搜索工具。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所有开放的信息资源）。以 P2P 技术发展的另一先锋 Gnutella 进行的搜索为例：一台 PC 上的 Gnutella 软件可将用户的搜索请求同时发给网络上另外 10 台 PC，如果搜索请求未得到满足，这 10 台 PC 中的每一台都会把该搜索请求转发给另外 10 台 PC，这样搜索范围将在几秒钟内以几何级数增长，几分钟内就可搜遍几百万台 PC 上的信息资源。可以说，P2P 为因特网的信息搜索提供了全新的解决之道，著名的搜索引擎公司 Google 也宣称要采用 P2P 技术来改进其搜索引擎，一家名为 Infrasearch 的新建公司也因为开发 P2P 技术的搜索引擎而获得了一笔巨额风险投资。

（4）文件共享

P2P 技术使在 Internet 上的任意两台计算机之间直接共享文档、多媒体和其他文件成为了

www.ckook.com

可能。利用 P2P 技术，网上计算机之间可以进行直接交互，而不需要使用任何一台中央服务器。可以说，对文件交换的需求直接引发了 P2P 技术热潮。在传统的 Web 方式中，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，用户再到某个网站搜索需要的文件，然后下载，这种方式对用户而言非常不方便。Napster 就是抓住了人们对 MP3 音乐的需求，其 MP3 交换直接引发了网络的 P2P 技术革命。在 P2P 网络中，对等机通过不同的查询机制定位含有所需资源的其他对等机后，将直接与其建立连接，并下载所需文件。

1.4 P2P 的发展

1.4.1 起步

P2P 基本技术的存在时间和 USENET（新闻组）、FidoNet（惠多网，一种 BBS 网络技术）这两种非常成功的分布式对等网络技术几乎是相同的，甚至更长些。USENET 产生于 1979 年，FidoNet 创建于 1984 年，它们都是一个分散、分布的信息交换系统。这些都是 P2P 技术的早期应用，可以看出 P2P 技术出现的时间是非常早的。

1.4.2 发展

P2P 正式步入发展的历史可以追溯到 1997 年 7 月，这也是因特网在中国起步的阶段。在一段介绍此时 P2P 技术的时间表中这样写着："Hotline Communications is founded, giving consumers software that lets them offer files for download from their own computers"。这句话的意思是：1997 年 7 月，Hotline Communications 公司成立，并且研制了一种可以使其用户从其他用户计算机中直接下载资料的软件。

早在 1998 年，美国东北波士顿大学的一年级新生、18 岁的肖恩·范宁为了能够解决他的室友的一个问题——如何在网上找到音乐而编写了一个简单的程序，这个程序能够搜索音乐文件并提供检索，把所有的音乐文件地址存放在一个集中的服务器中，这样使用者就能够方便地过滤上百的地址而找到自己需要的 MP3 文件。到了 1999 年，令他们没有想到的是，这个叫做 Napster 的程序成为了人们争相转告的"杀手程序"——它令无数散布在因特网上的音乐爱好者美梦成真，无数人在一夜之内开始使用 Napster。在最高峰时 Napster 网络有 8 000 万的注册用户，这是一个让其他所有网络望尘莫及的数字。这大概可以作为 P2P 软件成功进入人们生活的一个标志。

1.4.3 高峰

如今，P2P 技术的发展就得使用月，甚至日来记载了。现在，使用 P2P 技术的软件比比皆是，人们也在不知不觉中感受到了 P2P 作为高科技发展载体的快乐。应用 P2P 技术的 BT 软件的成功，将 P2P 的应用推到了一个顶峰。BT，这个被人戏称为"变态"的词几乎在大多

www.ckook.com

数人感觉中与 P2P 成了对等的一组概念，而它也将 P2P 技术发展到了近乎完美的地步。实际上 BitTorrent（BT，比特流）原先是指一个多点下载的 P2P 软件。它不像 FTP 那样只有一个发送源，BT 有多个发送源，当你在下载时，同时也在上传，使大家都处在同步传送的状态。应该说，BT 是当今 P2P 最为成功的一个应用。

与此同时，P2P 网络所特有的非中心化（Decentralization）、可扩展性、顽健性、高性价比、隐私保护及负载均衡等特点引发了世界各研究机构的兴趣。目前基于 P2P 的各种应用也在蓬勃的发展。具体内容将在各个章节中分别介绍。

1.5 P2P 技术国内外研究现状

1.5.1 国外相关研究

当前国外比较知名的 P2P 研究有麻省理工学院的 IRIS 计划，SUN 公司的 JXTA 平台，以及斯坦福大学的 P2P 研究。

（1）IRIS 计划

为了解决日益严重的因特网安全缺陷，美国国家科学基金会资助 5 所大学和研究机构合作开发一项安全的分布式数据存储系统，该计划名为 IRIS（Infrastructure for Resilient Internet Systems）。从本质上讲，IRIS 将是最终的 P2P 网络，但是其研究人员感兴趣的不是路由器、交换机和骨干网络，而是资源定位。IRIS 不仅仅是一个文件共享系统，还是独立于应用的平台，基于该平台可以方便地开发其他分布式应用。

IRIS 计划的核心技术是 DHTs（Distributed Hash Tables，分布式散列表）技术，MIT 的 chord 项目已经实现基于 DHT 的查询，Rice（莱斯）大学、NYU、UC Berkeley 都有相关的项目，分别是 Pastry、Kademelia、Ocean Store 和 CAN。资助 IRIS 计划的目的是深入研究 DHTs 技术在对等网络中的应用，大规模地验证已有的研究成果，最终建立安全的、顽健的分布式网络。

（2）斯坦福大学的研究

斯坦福大学主要是对 P2P 系统的搜索、资源分配和聚集以及安全问题进行了研究探讨，特别总结归纳了仍需要进一步研究的搜索和安全方面的开放性问题。搜索机制的关键性研究问题是要如何尽可能高效、高质、健壮地提供最优的表达力、复杂度和自组性。而安全性方面的关键问题是要确保网络的有效性、可靠性、匿名性和访问控制。更多的信息在斯坦福大学的 P2P 研究主页 http://www-db.stanford.edu/peers/上可以查找到。

1.5.2 国内研究现状

在国内，目前北京大学、清华大学与华中科技大学都开发出了相应的系统，详细介绍如下。

（1）北京大学——Maze

Maze 是北京大学网络实验室开发的一个中心控制与对等连接相融合的对等计算文件共

www.ckook.com

享系统，在结构上类似 Napster，对等计算搜索方法类似于 Gnutella。网络上的一台计算机，不论是在内网还是外网，可以通过安装运行 Maze 的客户端软件自由加入和退出 Maze 系统。每个节点可以将自己的一个或多个目录下的文件共享给系统的其他成员，也可以分享其他成员的资源。Maze 支持基于关键字的资源检索，也可以通过好友关系直接获得。

（2）清华大学——Granary

Granary 是清华大学自主开发的对等计算存储服务系统。它以对象格式存储数据。另外，Granary 设计了专门的节点信息收集算法 PeerWindow 的结构化覆盖网络路由协议 Tourist。

（3）华中科技大学——AnySee

AnySee 是华中科大设计研发的视频直播系统。它采用了一对多的服务模式，支持部分 NAT 和防火墙的穿越，提高了视频直播系统的可扩展性。同时，它利用近播原则、分域调度的思想，使用 Landmark 路标算法直接建树的方式构建应用层上的多播树，克服了 ESM 等一对多模式系统由联接图的构造和维护带来的负载影响。

在市场方面，国内 P2P 的市场基本从 2000 年底开始逐步启动。在基于 P2P 技术进行文件共享方面，国内有多家公司已经推出了自己的产品，目前主要的几款中文 P2P 软件有 POCO、PPLive、Ezpeer、Kuro、KuGoo 等。

P2P 技术在国内的应用还刚刚开始，多数人对 P2P 的认识还不完整，甚至还有很多的误解，例如盗版问题、管理性差等很容易与 P2P 联系起来。实际上每个具有划时代意义的创新在刚出现的时候往往可能被人们误读。P2P 也不例外，Napster 让人们开始关注 P2P，但也使得许多人认为 P2P 就是歌曲交换、文件交换，甚至认为 P2P 和盗版有着必然的联系。P2P 肯定会对原有的数字版权等体系造成很大的冲击，但这是技术创新的力量。我们不能因为现有的某些习惯、体制而扼杀技术创新带来的进步。而事实上，也正是每一个技术进步带动着社会的前进，带动着人们的生活方式、社会体制、法律法规的更新与进步。现在，我们又面临一场新的技术革命——P2P，我们对其应该有更为清醒的认识。

1.6 本章总结

本章从 P2P 的基本概念开始，首先着重介绍了 P2P 网络的拓扑结构，让读者对 P2P 网络可以有形象的认识，接着又介绍了 P2P 的发展、应用和国内外研究现状。目前，P2P 技术还是一个比较新的技术，各种各样的软件和标准层出不穷。但是每个 P2P 软件或者标准都具有本章所描述的一些特性，即要充分利用网络边界的资源。通过本章的学习，读者应该可以区别 P2P 网络和传统的集中式网络的差别，并能分别出 P2P 网络的结构分类。

1.7 练习题

（1）阐述 P2P 的概念。

（2）阐述 P2P 的几种网络结构。

（3）谈谈自己对 P2P 技术发展的看法。

www.ckook.com

参 考 文 献

[1] 郑纬民，等.对等计算研究概论．清华大学计算机科学与技术系高性能计算研究所，2005

[2] 罗杰文．Peer to Peer 综述．中科院计算技术研究所，2005

[3] 林卓林．P2P 原理与技术．矩阵通信技术论坛，2007-7-29

[4] 雷葆华，杨明川．P2P 技术的组网模式与业务模式探讨．北京：电信技术，2004（11）

www.ckook.com

CHAPTER 2

第 2 章　主流 P2P 系统

在读者对 P2P 网络的概念以及拓扑结构有了大概了解以后，本章将对当前主流的 P2P 系统进行分类介绍。

在学习了本章内容后，希望读者能够掌握以下两点：

（1）当前主流的 P2P 系统及应用领域；

（2）用 P2P 的思想去理解一些网络应用。

2.1　文件共享类系统

文件共享类系统是 P2P 应用的一个最主要的方面之一，目前已经有多种文件共享系统被广泛使用。虽然它们的共享方式各不相同，但都属于 P2P 网络的范畴。本节将对比较流行的几种文件共享系统做简单介绍。

2.1.1　Napster

Napster 系统是最早出现的 P2P 系统。1998 年，美国波士顿大学的一年级新生肖恩·范宁为了方便自己和室友之间共享 MP3 音乐，开发了一个局域网音乐共享程序。用户安装这个程序后，可以把自己拥有的音乐的相关信息（如歌曲名、拥有人地址等）发布到一个目录服务器上。其他用户可以在目录服务器上搜索自己需要下载的歌曲，找到提供下载这些歌曲的用户，然后就可以直接到对应用户去下载歌曲，这个共享系统被称作 Napster。系统示意图如图 2-1 所示。

目录服务器维护每首歌及共享者的列表，并针对用户的查询返回匹配的用户列表。比如，用户 E 想要下载《那些花儿》这首歌，他首先到目录服务器上进行查询，得到的回复是用户 D 那里可以提供下载，紧接着用户 E 就可以直接和用户 D 通信，把歌曲下载回来。

Napster 的运行模式更像是一个搜索引擎，服务器负责对用户共享的文

www.ckook.com

件简历索引并提供查询服务。这种方式实现了文件查询和文件下载的分离，极大地减轻了服务器的负担，比传统的客户端/服务器下载模式的效率要高得多。

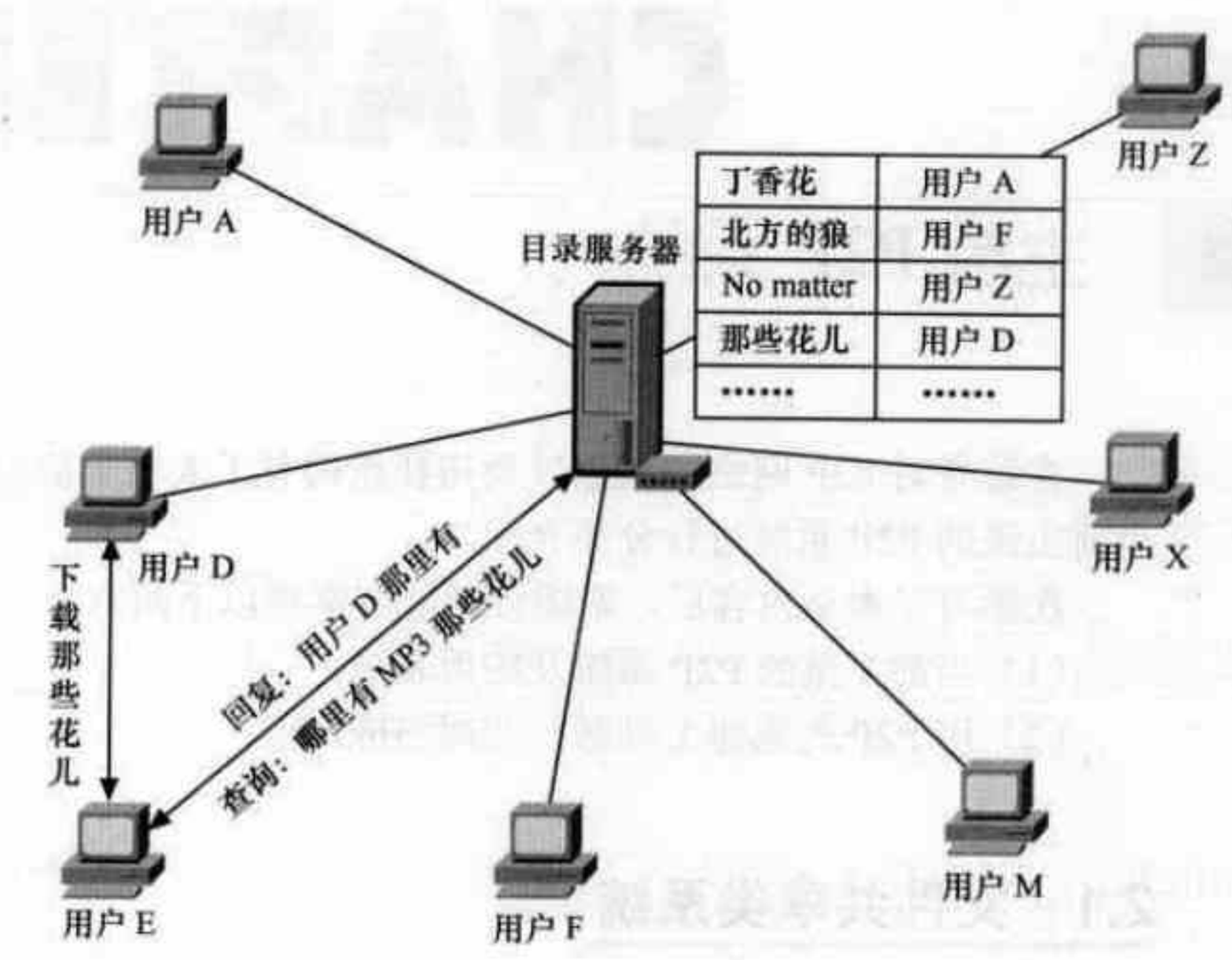


图 2-1 Napster 系统结构示意图

从结构上来看，Napster 属于中心化拓扑的 P2P 网络，整个系统由目录服务器和各个客户端构成。目录服务器由一个或一组高性能的服务器承担，主要负责所有活动客户端共享资源的管理，并提供资源查询服务；客户端安装在个人计算机上，可以动态地加入和离开网络。这种中心化的拓扑使得 Napster 容易部署，而且客户端容易开发。但是中心化拓扑不可避免的两个问题就是可扩展性和单点失效。当用户的数量迅速增大时，由于目录服务器不能很快地处理用户的查询请求，Napster 的系统性能会大大下降，而且目录服务器的瘫痪会造成整个系统的瘫痪。安全问题也是 Napster 系统的一个重大问题，目录服务器不对用户身份的合法性进行验证，有很大的安全隐患，而且中心化拓扑也不能很好地抵抗拒绝服务（DoS，Denial of Service）攻击。

困扰 Napster 的另一个比较重要的非技术问题是版权问题。由于用户可以自由地加入 Napster 并免费下载音乐文件，唱片公司的利益受到了威胁。1999 年 12 月，Napster 网站被多家唱片公司告上法庭，法庭判 Napster 网站败诉。

版权事件后，Napster 系统的发展就不再像以前那么迅速了，逐渐被后来的其他 P2P 下载系统所取代。然而，Napster 终究是最早的 P2P 系统，对 P2P 网络的发展具有非常重要的意义。目前，用户还可以从 www.napster.com 网站上下载 Napster 客户端软件，加入 Napster 系统。最新版的 Napster 软件界面如图 2-2 所示。最新的 Napster 软件提供了多种功能，用户除了可以搜索其余用户共享的音乐文件外，还可以在线收听、刻录 CD 等。

www.ckook.com

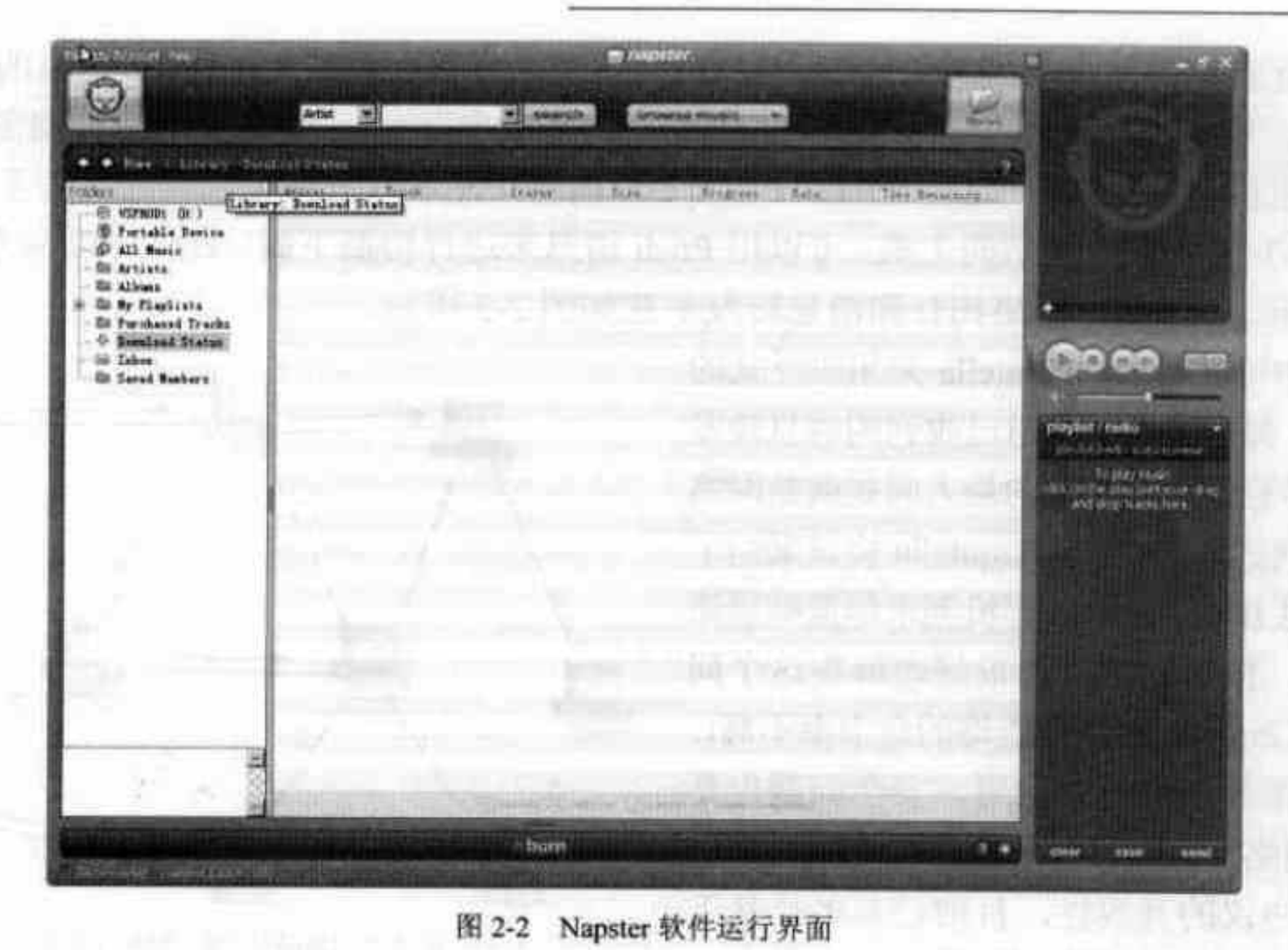


图 2-2 Napster 软件运行界面

2.1.2 Gnutella

Gnutella 是 Napster 的一个变种，它改变了 Napster 的拓扑结构，并且在很短的时间内就变得非常流行。严格来说，Gnutella 是一种能够智能发现节点，完全分布式的 P2P 通信协议。运行任何一个基于 Gnutella 协议开发的客户端软件都能加入 Gnutella 网络。

Gnutella 的协议非常简单，可以分为消息协议和下载协议。消息协议是 Gnutella 网络节点之间用来相互发现和搜索资源的，下载协议是两个节点之间用来传送文件的。

消息协议部分包含 5 种类型的消息，其类型和描述如表 2-1 所示。

表 2-1 Gnutella 协议的消息类型

消息类型	描述
Ping	用来动态地发现 Gnutella 网络的主机。当主机收到一条 Ping 消息时，应该回送一条 Pong 消息
Pong	Ping 消息的回送消息。包括已连接主机以及该主机可以向网络提供的共享数据的数量消息
Query	分布式 Gnutella 网络的查询消息。当一个主机收到一个 Query 消息时，将根据消息中的信息搜索本地数据列表，如果找到对应数据，就返回一条 QueryHit 消息
QueryHit	Query 消息的回应消息，包括获取查询到的数据信息
Push	一个用于允许防火墙中的客户端向网络提供基于文件的数据文件的机制

其中，Ping 和 Pong 消息用来发现邻居节点，Query 和 QueryHit 消息用来进行数据查询，Push 消息用来穿越防火墙的主机和数据探测。

在 Gnutella 网络中，每个节点既是服务器又是客户端。当一个节点需要查询消息时，会采用洪泛式的查询方式，即先把查询消息（Query）发送到自己的直接邻居节点。邻居节点首先查找自己的数据列表，如果发现要查询的数据，就回送一条确认信息（QueryHit），否则就把这条信息转发给自己的直接邻居。在转发的过程中，节点会检查并修改消息头的 TTL 字段和 Hops 字段，如果发现消息的 TTL 为 0，直接丢弃该消息。

www.ckook.com

当进行查询的节点收到 QueryHit 消息后，就可以直接到发送 QueryHit 消息的主机上去下载文件。下载协议采用的是标准的 HTTP 协议，下载方发送一个 GET 消息给服务器（提供该文件的主机），服务器回送一个 HTTP 200 OK 消息，然后双方就可以传送文件了。对于服务器在 NAT 或者防火墙后的主机，可以用 Push 消息来进行协商下载。具体可以参考 Gnutella 的协议规范。Gnutella 网络拓扑和消息转发示意如图 2-3 所示。

从图中可以看出，Gnutella 采用洪泛式的查询机制，每个节点都将自己收到的消息转发出去，这种查询方式带来的最大问题就是网络流量的几何式增长。虽然 Gnutella 协议采用了 TTL、消息 ID、路径 ID 等机制来尽量减缓流量的增长，但是仍然不能很好地解决这个问题。而且 Gnutella 网络中查询的命中率不高，不太适合作为一个商业应用。安全问题也是 Gnutella 网络没有加以考虑的问题。

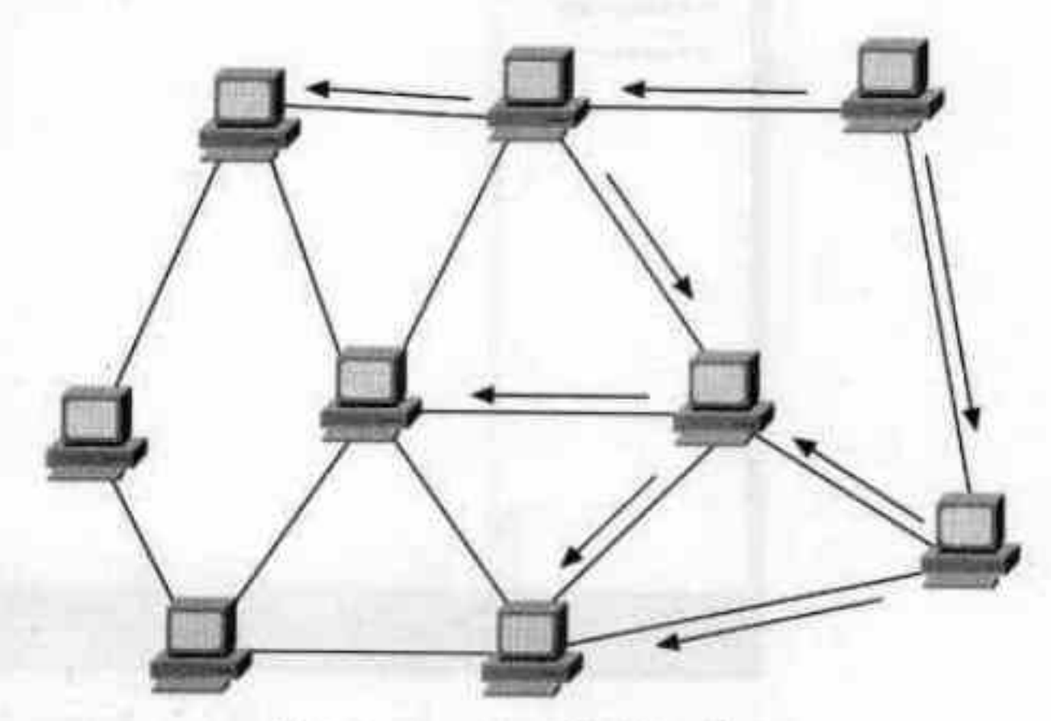


图 2-3　Gnutella 网络结构示意图

由于协议的开放性，目前已有多款基于 Gnutella 开发的客户端，如 Windows 平台下的 BearShare、Gnucleus、Morpheus、Swapper、XoloX Ultra、LimeWire、Phex 等，Linux/UNIX 平台下的 Gtk-Gnutella、Mutella、Qtella、LimeWire、Phex 等，Macintosh 平台下的 LimeWire、Phex 等。由于基于相同的协议，这些客户端之间可以相互通信，用户可以根据自己的操作系统以及爱好习惯选取任意一个客户端。下载网址为 www.gnutella.com。最新的 Windows 平台下 Phex 的运行界面如图 2-4 所示。

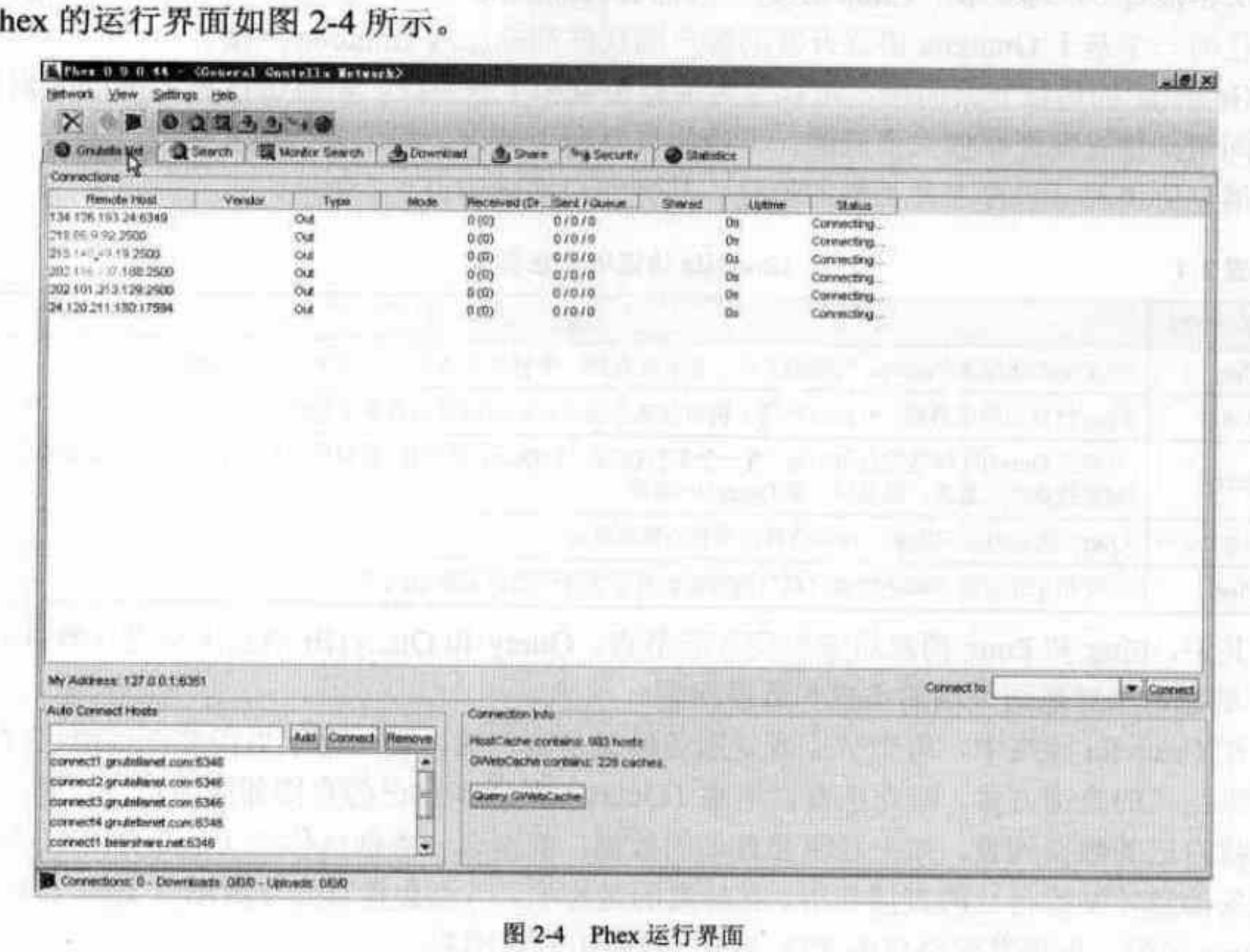


图 2-4　Phex 运行界面

www.ckook.com

2.1.3 BitTorrent

BitTorrent 的第一个可用版本是由 Cohen 于 2002 年 10 月推出的，简称 BT，是一个可以实现多点下载的文件分发协议。它的基本原理是每个人在下载的同时也为其他下载用户提供上传，所以，参与的人越多，“文件源”就越多，下载速度就越快。这也充分体现了 P2P 网络概念中“我为人人，人人为我”的思想。

当某个用户想要共享文件或目录时，首先要为该文件或目录生成一个“种子”文件，或者叫“元信息”文件（该文件中包含共享文件或目录以及用户的 URL 等信息），然后把这个“种子”文件上传到 BT 服务器上（又名 Tracker 服务器，如我们通常上的 bt.5qzone.net），等待别的用户来下载。需要下载某个文件的用户要首先到 BT 服务器上找到该文件的一个“种子”文件，然后根据“种子”文件提供的信息进行下载。BitTorrent 系统中把文件分割成相同大小的块，各个块可能会从不同的节点同时下载。BT 服务器及时收集每个下载者的信息，包括地址以及目前拥有的文件块信息等，然后从下载者的列表中随机选取一组告诉某个正在下载的节点。各个下载节点和 BT 服务器之间要及时进行信息交换，以确保下载顺利完成。BitTorrent 系统的示意图如图 2-5 所示。

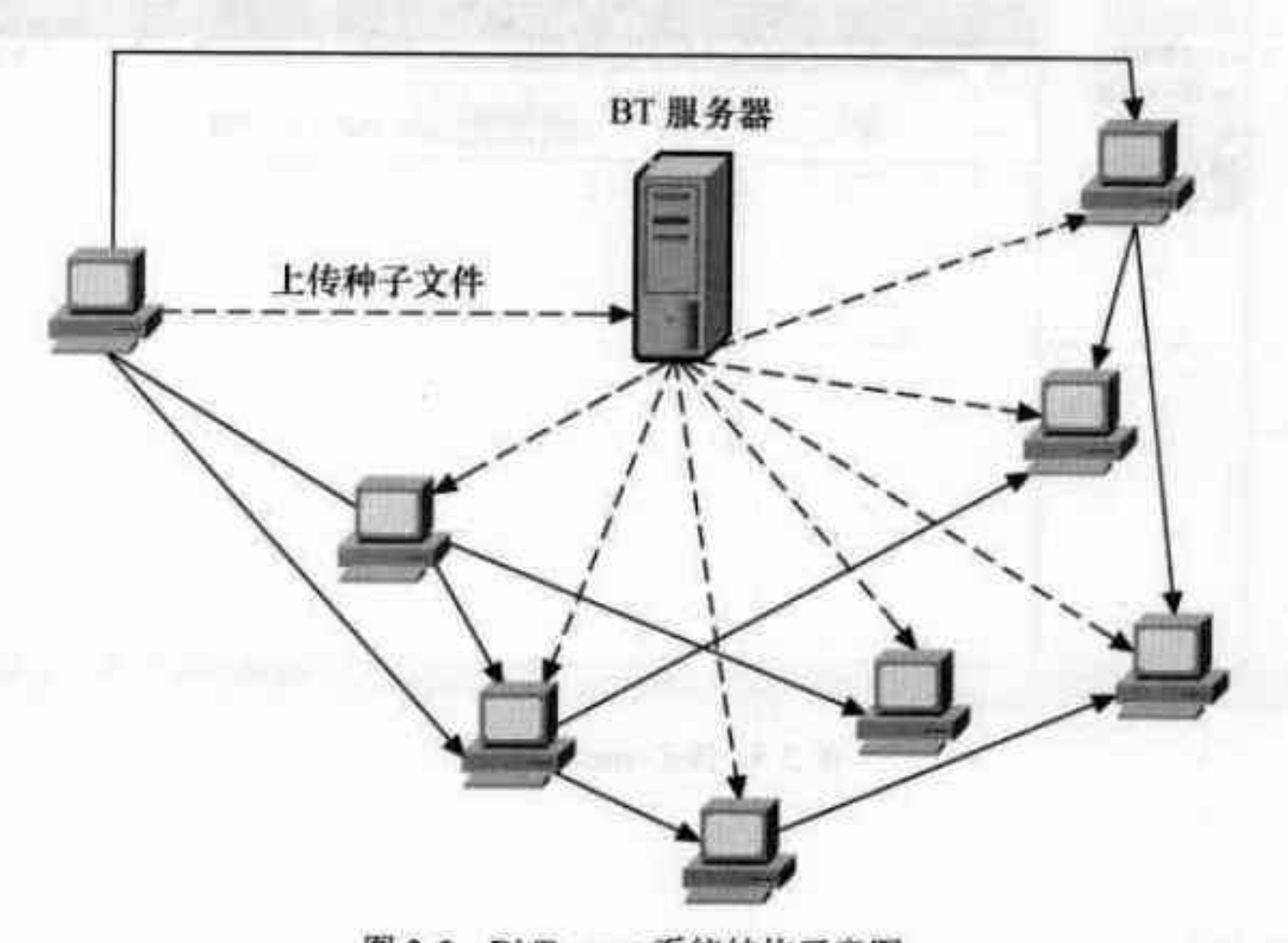


图 2-5 BitTorrent 系统结构示意图

图中的虚线表示种子文件的传送路径，实线表示文件块的传送路径。

在 BitTorrent 系统中，“种子”文件的上传下载以及各个节点和 BT 服务器之间的其他通信都是基于 HTTP 协议的，而各个节点之间的通信协议是由 Bittorrent 规范规定的，叫作 Bittorrent peer 协议。

各个文件之间交换文件时是把文件分割成固定大小的块来进行。为了保证最后所有的块可以顺利重组，BitTorrent 系统采用了 Hash（SHA1）机制，首先为整个文件生成一个校验值，然后为每个块也生成一个校验值。校验值可以保证文件块的唯一性，也可以作为文件块的标识。

www.ckook.com

从结构上来看，BitTorrent 系统属于中心拓扑的 P2P 网络，因此 BT 服务器同样存在着性能和单点失效问题，而且 BT 服务器没有针对用户的身份认证，在安全性方面也存在着漏洞。但是作为一个文件共享系统，BitTorrent 是一个非常成功的应用。特别是在带宽资源非常丰富的今天，BitTorrent 系统可以很方便地在用户之间交换文件。

BitTorrent 系统的协议也是公开的，任何人都可以开发服务器和客户端。国内比较流行的一个客户端是 BitComet，它是基于 Python 语言开发的。BitComet 软件运行环境如图 2-6 所示。

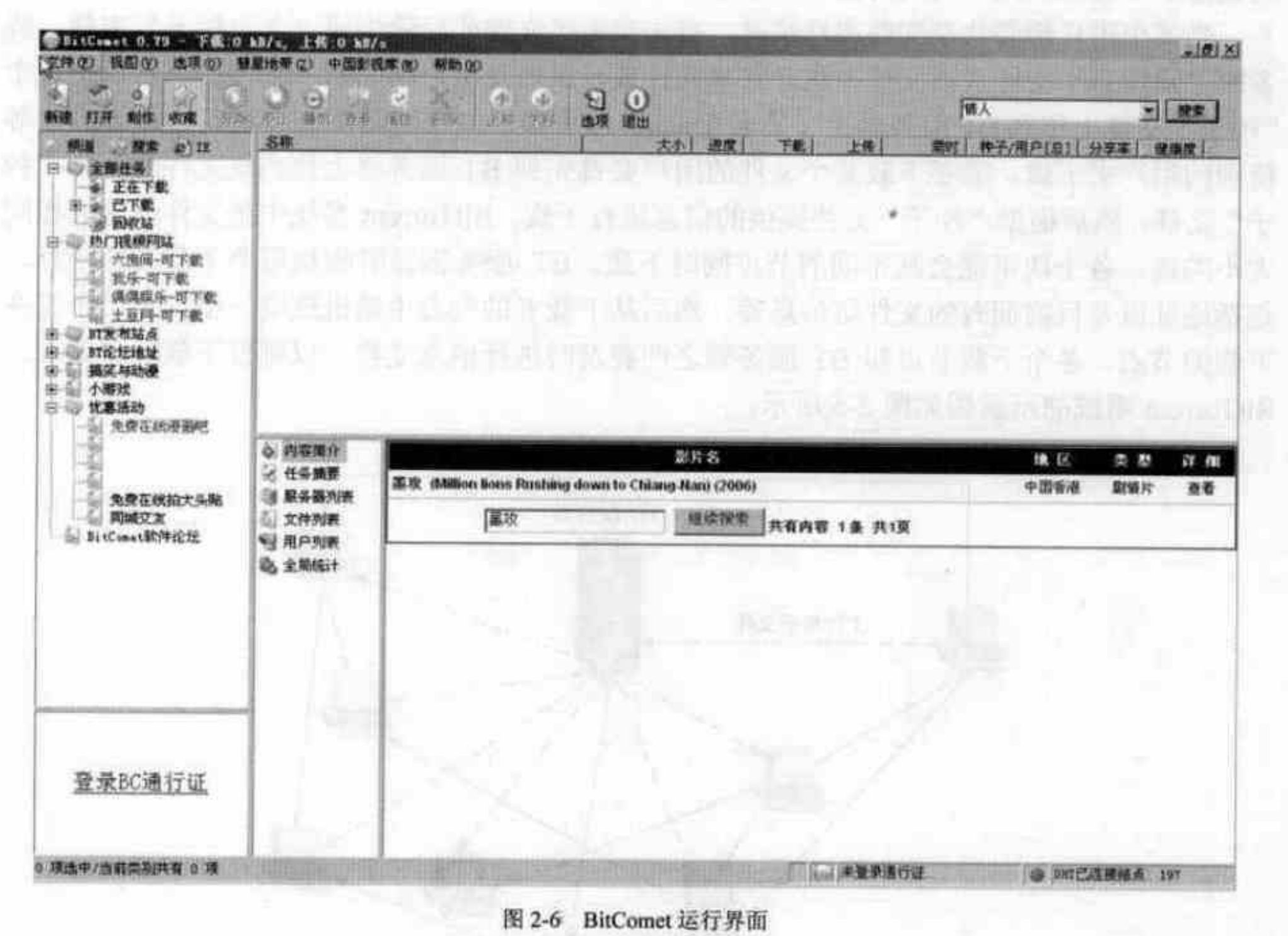


图 2-6 BitComet 运行界面

2.1.4 eMule

eMule 俗称“电骡”，是一个和 BT 非常类似的 P2P 下载工具，它源自于 eDonkey。eDonkey 由 Jed McCaleb 在 2000 年创立。该工具最重要的优点是可以同时从许多人那里下载同一个文件，并且采用了“多源文件传输协议”。eDonkey 的索引服务器并不是集中式的，而是由个人来运行，每个人都可以运行电骡服务器。在 eDonkey 协议中，定义了一系列文件传输、压缩和打包的标准，还有一套积分标准，可以根据用户上传的数据量来分配相应的带宽。积分越高，下载速度越快，这也解决了其他 P2P 系统中很多人只下载不上传的问题。在文件标识的问题上，eDonkey 采用了和 BT 一样的机制，用 Hash（MD5）函数对文件的计算结果作为该文件的唯一标识。文件的下载也通过分片来同时从多个地方

www.ckook.com

下载。

eMule 是 eDonkey 的改良版，而且是开源软件。eMule 采用了和 eDonkey 相同的网络协议，能直接登录 eDonkey 服务器，加入 eDonkey 网络。eMule 同时也提供了 eDonkey 不具有的功能，比如自动搜索网络中的服务器、缓存搜索结果、和其他节点交换服务器地址、首先下载文件头尾供用户预览等。

eMule 的系统结构示意图如图 2-7 所示。

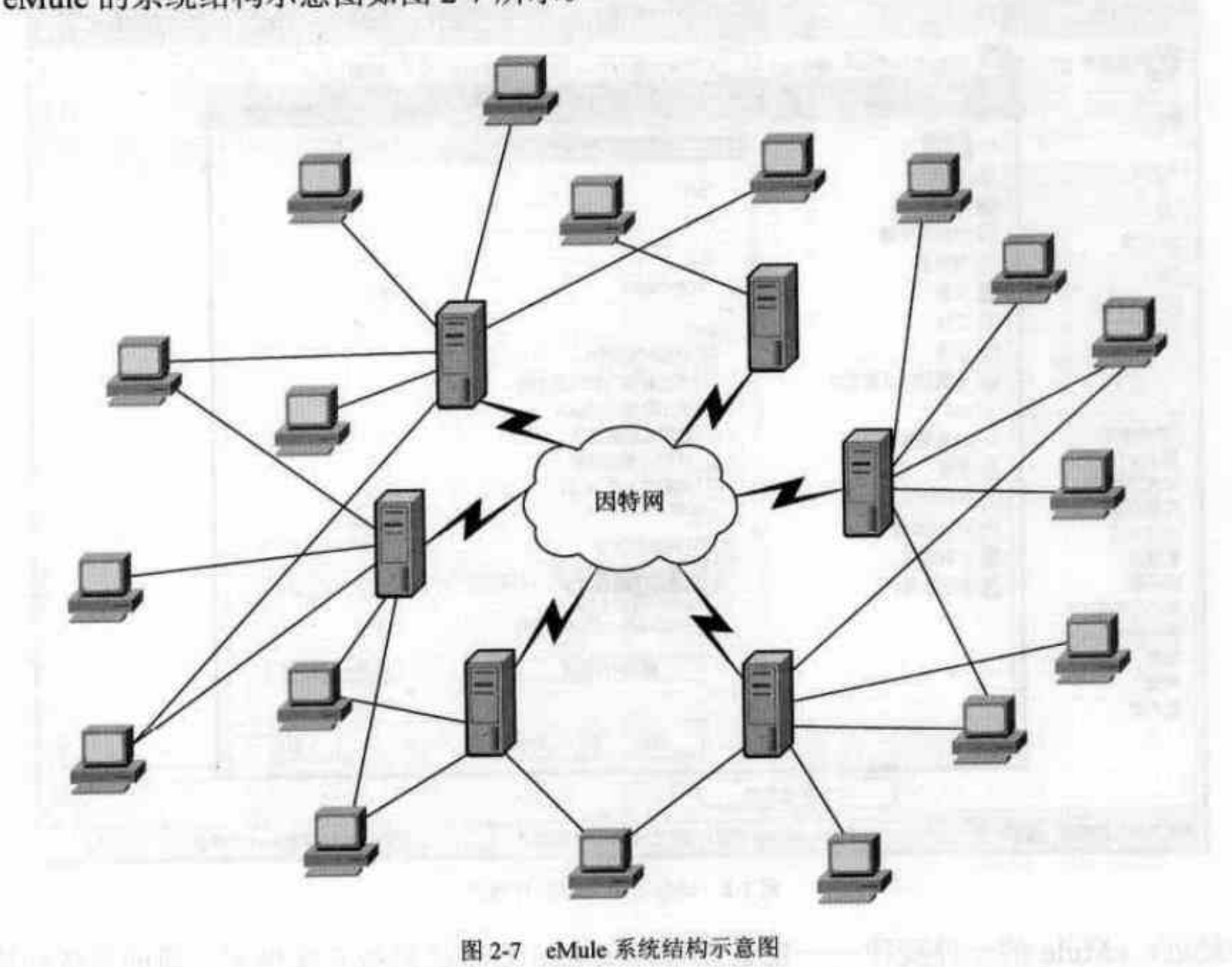


图 2-7 eMule 系统结构示意图

从图中可以看出，eMule 系统中的服务器分布在因特网的各个地方，而且客户端和服务器之间是多对多的关系。eMule 系统中的各个服务器之间是不进行相互通信的。

在客户端的服务器列表中，有一个是该客户端的主服务器。当客户端接入网络中后，先用 TCP 通信的方式把自己的共享文件信息以及地址信息发送给主服务器，随后用 UDP 通信的方式把相同的信息发送给其他服务器。每个服务器会记录登录到自己的所有客户端信息。当服务器收到查询请求后，会通过匹配记录把查询结果反馈给客户端；而客户端在搜索时，可以同时向多个服务器发出请求，这样大大提高了搜索的命中率。当客户端搜索到自己需要的文件时，就可以进行下载了。下载前，客户端要先从服务器上查询拥有这个文件的客户端列表，然后从不同的地方选择不同的块同时进行下载。下载到某个文件片断后，该片断可以同时被其他客户端下载，所以 eMule 也具有“下载的人越多，下载速度越快”的优点。

从结构上来看，eMule 是半中心化拓扑的。由于服务器分布在整个网络中，从一定程度上减少了单点失效发生的概率。而且由于 eMule 引入了强制共享机制，避免了其他

www.ckook.com

P2P 软件的低效率缺陷。不过这种拓扑结构也带来了安全隐患，由于每个人都可能成为服务器，而且客户端缺乏对服务器端的身份认证，使得恶意用户可以冒充服务器来控制 eMule 网络。

eMule 是开放源代码软件，读者可以通过协议和源码研究 eMule 网络更详细的运作机制。最新版的中文 eMule 的软件界面如图 2-8 所示。

图 2-8　eMule 中文版软件界面

最近，eMule 的一种变种——完全分布式 eMule 系统已经被开发出来。新的系统利用了 Kademlia 算法来实现节点间的发现和消息路由。Kademlia 也是开放源代码的，感兴趣的读者可以去研究一下它的实现机制。

2.1.5　Maze

2003 年暑假期间，北京大学网络实验室的几名研究生发现在众多的 FTP 站点里很难找到自己需要的资料，就开发了最早的 Maze 系统。Maze 最初是作为 FTP 资源搜索工具来使用的，后来安装到用户计算机上后，用户可以把自己硬盘上的一个区域划作共享区，其他用户就可以到这个共享区来下载东西，而不需要再让用户把自己的文件上传到服务器上。后来，Maze 又增加了许多新的功能，如添加好友、Maze 博客等，现在已经成为教育网内最流行的 P2P 文件共享软件。

从结构上来看，Maze 具有和 Napster 类似的结构，每个用户都把自己的共享数据提交到目录服务器上，以方便别人的搜索；从搜索策略上看，Maze 类似于 Gnutella，都是一种

www.ckook.com

广度优先搜索（当然了，Maze 也支持在服务器上进行搜索）。Maze 的系统结构示意图如图 2-9 所示。

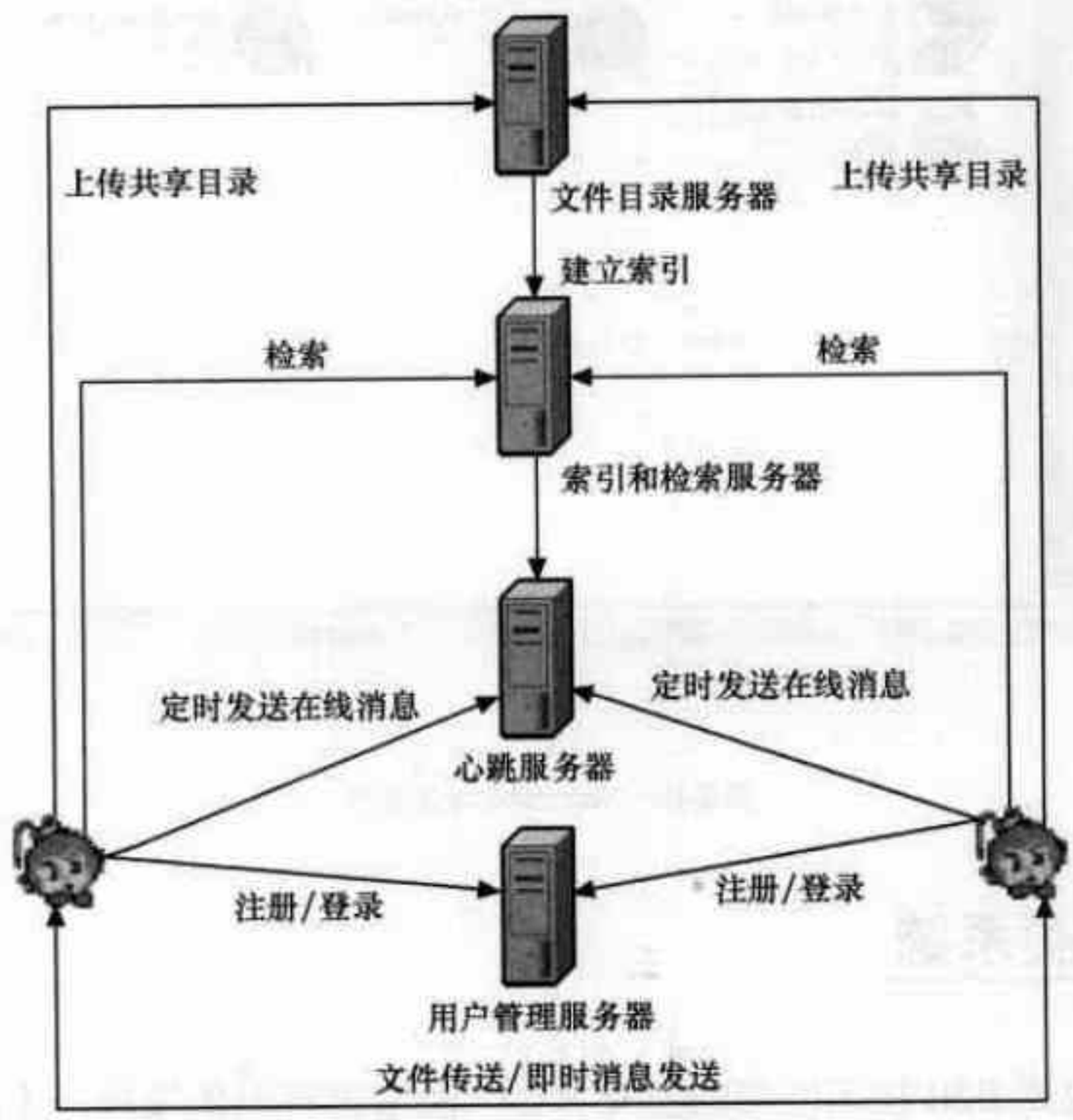


图 2-9 Maze 系统结构示意图

在这个系统中，每个节点（用户）相当于一个传统的 FTP 服务器和客户端结合体，节点的个数不受限制。除了节点外，还有众多的服务器来支持整个系统的运行，用户管理服务器负责用户的注册和认证，目录服务器发挥和 Napster 系统中目录服务器相同的功能，用来把每个节点的共享数据列表收集到数据库中，索引和检索服务器读取用户的共享数据库，建立基于 XML 接口的索引并提供检索服务，心跳服务器负责维护在线用户列表。在 Maze 系统中，每个在线节点每隔几秒钟就向心跳服务器发送自己的在线信息，同时要向目录服务器发送自己的共享目录信息。

由于在 Maze 系统中存在着中心服务器，所以性能问题和单点失效问题也是不可避免的。但是 Maze 充分利用了这种集中控制的优点，把整个系统的运营信息收集起来，可以供其他方面的研究利用，如社会学、网络资源分布等。同时 Maze 在安全性方面也前进了一步，利用分布式认证算法来保证用户认证的安全和有效性，支持节点间的身份认证。

Maze 从第 1 个版本到现在 4 年多的时间，注册用户已经超过了 600 万人，同时在线人数达到了 10 万，而且完成了从教育网到公众网的进军，发展速度堪称中国 P2P 文件共享领域内的第一。目前很多基于 Maze 的后续研究也在开展之中，如基于 Maze 的网格计算、基于 Maze 的网络观测平台以及数据挖掘等。

目前 Maze 的最高版本是 Maze2008，其运行界面如图 2-10 所示。

www.ckook.com

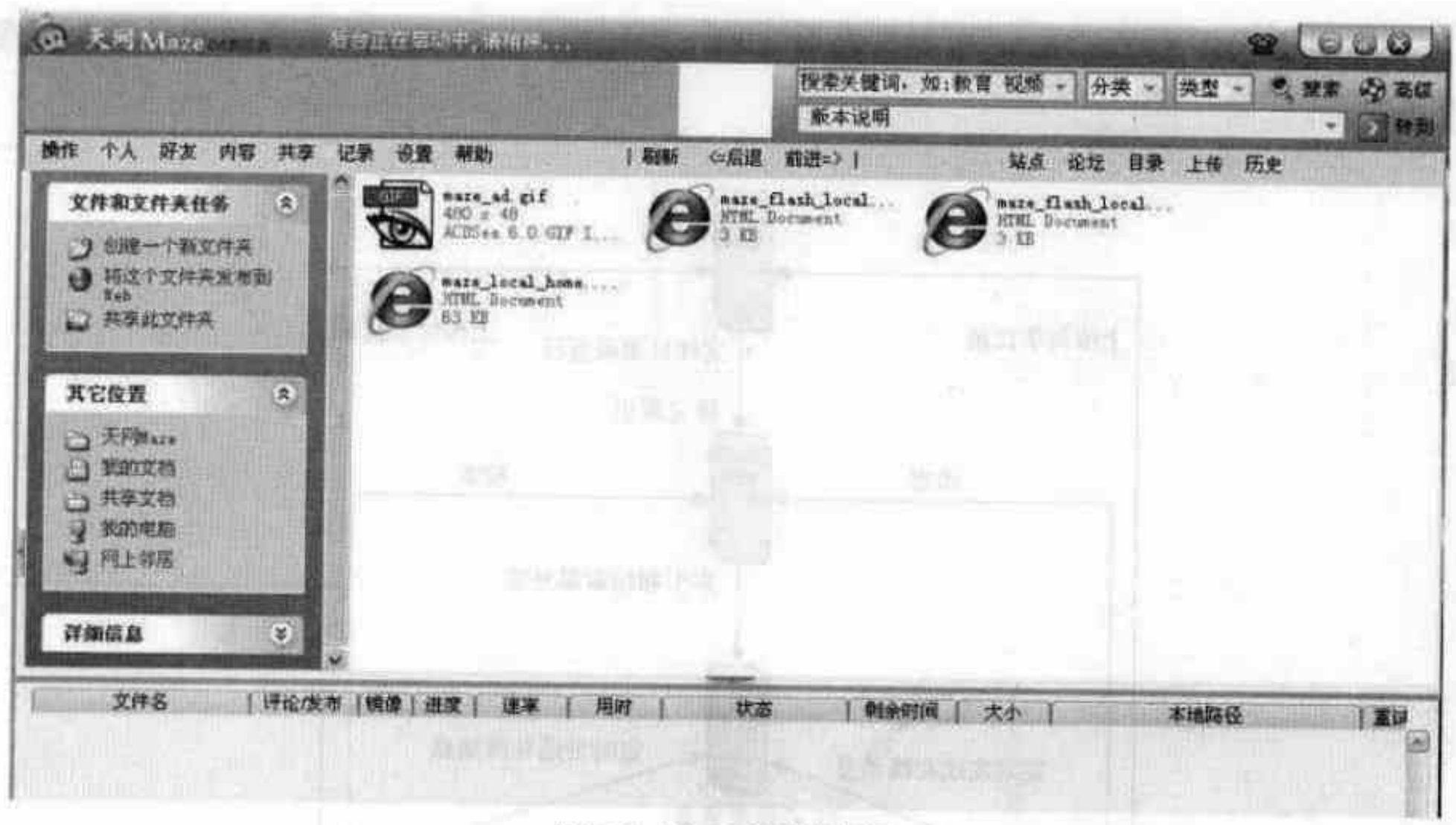


图 2-10　Maze2008 运行界面

2.2　即时通信类系统

即时通信系统也是 P2P 技术应用的主要方面。随着因特网的发展，人们不再满足于比较慢的网络交流方式如 E-mail 等，因此众多的即时通信系统被开发出来，最早的如 ICQ、MSN，到现在的 QQ、雅虎通、网易泡泡、Skype、GTalk 等。本节将对 Skype、QQ 和 GTalk 做简要介绍。

2.2.1　Skype

Skype 是由著名的 KaZaA 软件的创始人 Niklas 推出的一款因特网即时语音通信软件，也是现在最流行的网络电话软件。基于 Skype 的各种辅助产品，如 Skype 语音网关、Skype 电话机等也陆续在推出。所有这些发展情况都源自于 Skype 的众多优点。

（1）高清晰音质

从理论上说，使用 Skype 可以听到所有人类可以听到的声音频率，而普通电话只能听到 300～3 400Hz 以内的声音。高的频率范围保证了高保真度的声音。

（2）高度保密性

Skype 终端之间传送的消息都是在发送前加密，发送后解密，加密算法采用最新的 AES（高级加密标准）算法，密钥长度为 256bit，是 AES 可选密钥长度里最长的，也是最安全的。密钥用 2 048bit 的 RSA（一种非对称加密算法）算法来生成，可以确保密钥的安全性。同时，用户登录服务器时会要求用户用私钥来进行身份验证。

（3）免费多方通话

Skype 支持最多 5 人的多方会议呼叫，而且所有的通话也是采用端到端加密，可以很方

www.ckook.com

便地进行商务会谈和其他会谈。

（4）跨平台性能

Skype 提供不同操作系统的发行版本，包括 Windows、Linux 以及 Mac OS 等。

在系统的设计上，Skype 借鉴了 KaZaA 的结构，采用了半分布式拓扑，节点之间按照不同的能力分为超级节点和普通节点。超级节点除了具有普通节点的功能外，还负责数据的转发。超级节点和普通节点之间没有具体的界限，一个超级节点可以变成普通节点，普通节点也可以成为超级节点。Skype 的系统结构示意图如图 2-11 所示。

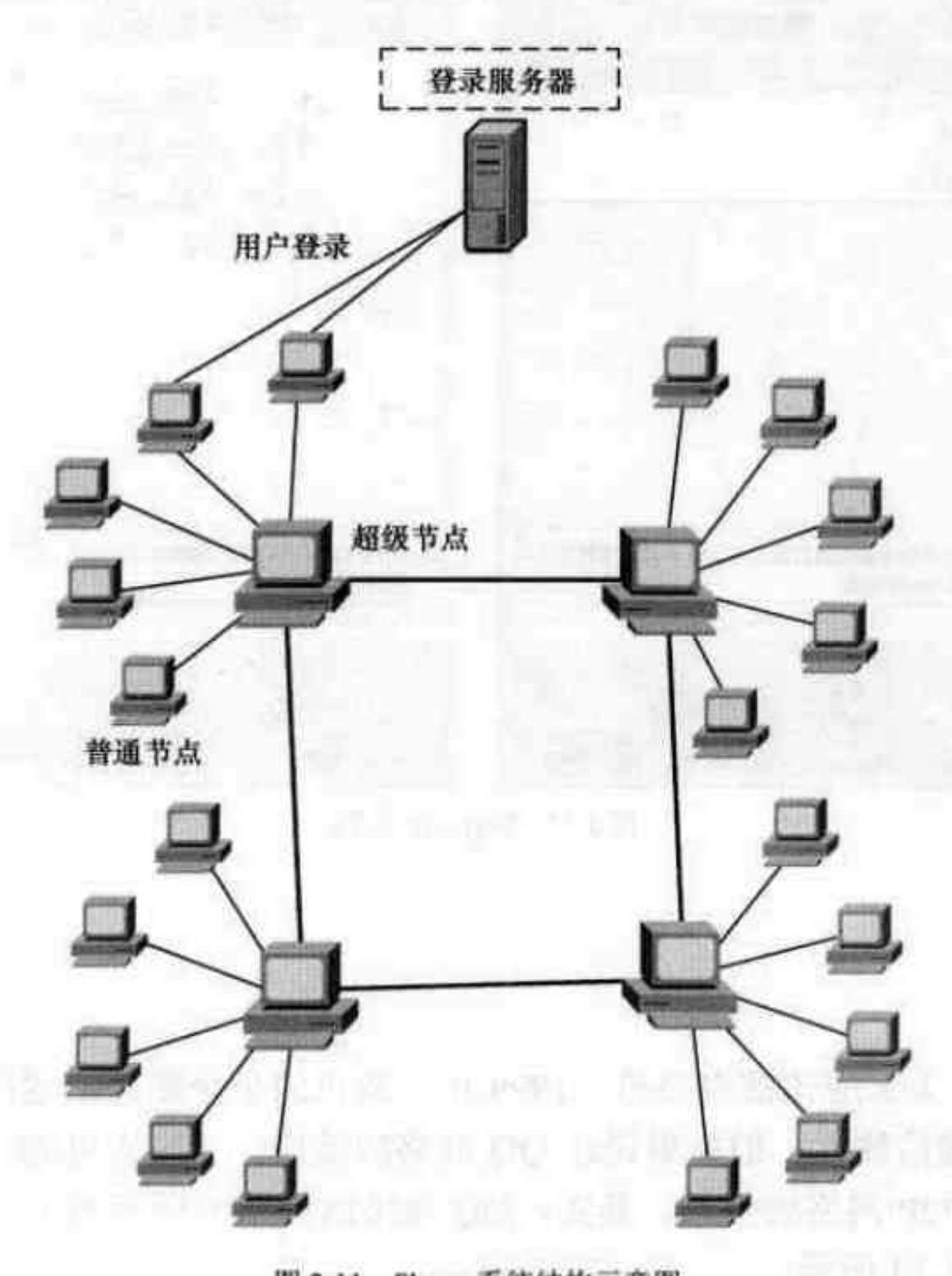


图 2-11　Skype 系统结构示意图

当某个节点启动时，首先要登录到服务器验证用户名和密码。初次注册的用户，服务器会分配给节点一个公钥/私钥对。然后服务器会告诉该节点它附近的超级节点列表，该节点选择一个超级节点连接，就可以加入 Skype 网络中了。当一个节点需要和另一个节点通信时，Skype 会利用一种全局索引的技术来进行分布式查找。当节点查找完成后，两个节点间就可以开始端到端通信了。利用超级节点，Skype 在穿越 NAT 和防火墙方面也具有非常好的性能。

由于 Skype 协议是私有协议，而且采用了基于 PKI 的身份认证以及最新的 AES 加密，使得 Skype 网络中传输的数据具有非常高的安全性。从结构上来看，Skype 采用了混合式的拓扑结构，结合了分布式拓扑和中心拓扑的优点，能适应网络的迅速变化。但由于超级节点在网络中的地位非常重要，负担也比较重，一旦发生变动，会对网络的负载均衡有很大的影响。

Tom-Skype 是由 Tom 在线和 Skype 公司联合推出的 Skype 中文版，在中国大陆地区发行。

www.ckook.com

Tom-Skype 最新版的软件界面如图 2-12 所示。

图 2-12 Skype 软件界面

2.2.2 QQ

说起腾讯 QQ，大家应该都很熟悉。1999 年，腾讯的小企鹅开始运作并以飞快的速度成为国内第一大即时通信软件。但如果说起 QQ 网络的结构，一般人可能并不清楚，特别是它和当前比较流行的 P2P 网络的关系。其实，QQ 网络结构属于标准的中心拓扑的 P2P 结构。其结构示意图如图 2-13 所示。

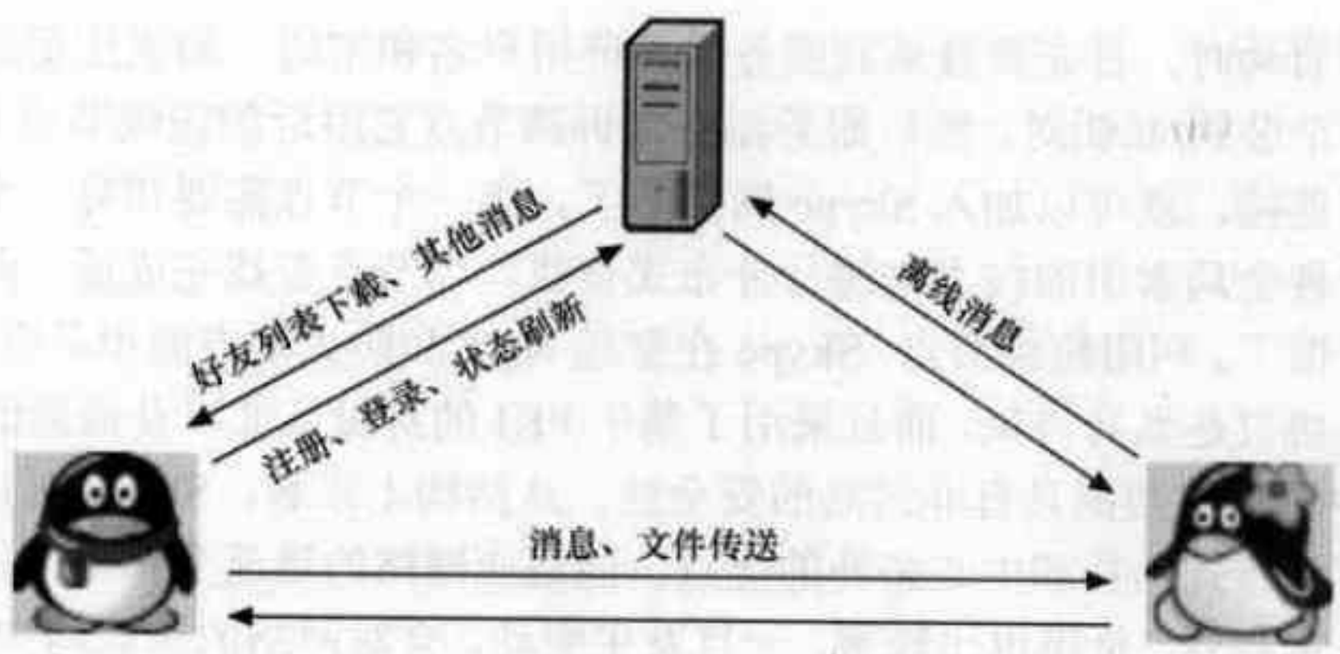


图 2-13 QQ 网络结构示意图

www.ckook.com

每个用户运行 QQ 客户端程序后，首先要输入自己的 QQ 号码和密码（初次登录用户可以通过很多方式申请号码，如在线申请、手机申请、邮箱申请等），然后向服务器发送登录消息，服务器对该 QQ 号码进行身份验证后就可以进入 QQ 网络了。顺利登录后，QQ 客户端会到服务器上下载自己的好友列表及好友在线消息，还有一些好友发送的离线消息。在线好友消息里会包含该好友的 IP 地址等消息，用户可以直接和在线好友进行点到点通信。如果通信的消息由于网络原因不能及时传送，QQ 客户端会选择经过服务器中转。如果用户要对某个离线好友发送消息，也可以直接发送，QQ 客户端会把该消息发送到服务器，等下次好友登录时服务器会把消息转送给该好友。

由于 QQ 采用了私有协议，因此对协议本身的分析只能采用黑盒分析的方法，通过抓数据包来实现。目前已经知道的是，QQ 的号码和密码发送是密文传送，而其他消息是明文发送，所以 QQ 聊天的内容并不安全。随着 QQ 的发展，QQ 号码已经变成了个人身份的标识，所以腾讯在保证号码安全方面做了很多工作，比如密码保护、号码激活等方式。申请了密码保护的用户在号码被盗后可以通过登录腾讯网站取回号码。当同一个号码在两个地方同时登录或者 QQ 异常退出时，腾讯都会要求用户进行号码激活等操作，以进一步保证用户号码的安全。

QQ 网络结构也属于中心拓扑，服务器对整个网络至关重要。为了尽可能地降低风险，腾讯采用分布式服务器的方式，在多个地方设置服务器。QQ 客户端登录时会尝试多种登录方式。不过这种方法只是减少了网络瘫痪的危险，并不能从根本上消除风险。

用过 Linux 的用户可能听说过 lumaQQ。lumaQQ 是由清华大学的研究生用 Java 和 SWT 开发的独立 QQ 客户端，遵循 GPL 协议。在 lumaQQ 的官方网站 http://lumaqq.linuxsir.org/main/ 上有 lumaQQ 的下载链接，同时有很多关于 QQ 协议分析的资料。对 QQ 协议感兴趣的读者可以到这个网站去查找进一步的资料。

2.2.3 GTalk

GTalk 是由美国 Google 公司最近几年推出的 IM 聊天软件。由于用户群以及市场的缘故，GTalk 并没有迅速流行起来。但是由于 GTalk 和 Gmail 之间密切的关系，使得 GTalk 有很多不同于其他 IM 软件的特点。

首先，GTalk 采用的是一个开放的标准。使用 Google GTalk 的用户实际上是连接到 Google GTalk 网络中的一个用户组，而这个网络是一个开放性的网络。这意味着 GTalk 用户可以和其他服务器提供商的用户进行对话。

其次，GTalk 和 Gmail 是一对一的关系。拥有 Gmail 的用户就同时拥有了 GTalk 的账户，两者之间可以互相通信。使用 GTalk 的用户可以收 Gmail 的电子邮件，而在 Gmail 电子邮件页面可以和 GTalk 的用户聊天。

最后，也是最重要的一点是 GTalk 的聊天记录可以存储在 Gmail 邮箱中，用户可以在任何连接到因特网的计算机上查看自己的聊天记录，而且用户可以利用 Google 强大的搜索功能对聊天记录进行搜索。

从网络架构上来说，GTalk 和 QQ 是一样的。用户首先需要登录到服务器去验证自己的身份同时取回自己的好友列表以及在线消息。和 QQ 不同的是，GTalk 是把聊天记录保存在服务器上，所以用户的聊天信息会往服务器上发送一份副本（除了多媒体消息及文件外）。而

www.ckook.com

且，GTalk 不支持离线消息，给离线好友发送消息是以邮件的形式发送的，但由于 GTalk 和 Gmail 绑定，这种邮件形式的离线消息会发送到用户的电子信箱里，有利于双方及时通信。GTalk 的网络结构示意图如图 2-14 所示。

图 2-14 中的 GTalk 用户既可以是使用 GTalk 客户端软件的用户，也可以是使用 Gmail 的用户（Gmail 中暂时不支持多媒体信息），还可以是在移动终端上访问 Gmail 的用户。

GTalk 的客户端运行界面如图 2-15 所示。

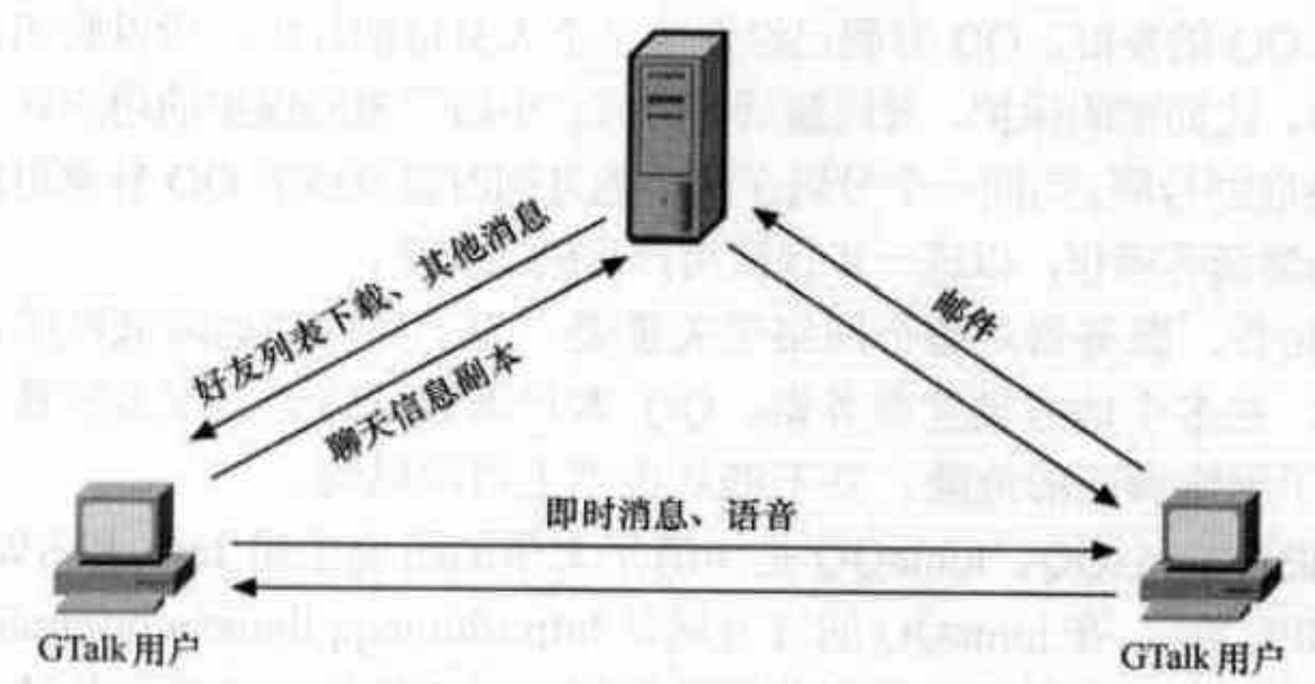


图 2-14　GTalk 网络结构示意图

图 2-15　GTalk 软件运行界面

2.3　流媒体类系统

随着因特网带宽的增加，网上电视等流媒体应用也逐渐出现。但是传统的客户端/服务器模式的应用不能满足多人在线看电视的需要，而 P2P 架构的网络电视就可以很好地解决这个问题，而且能达到人越多电视越流畅的效果。本节将分别介绍一个研究型流媒体系统 AnySee 和一个商业网络电视系统 PPLive。

2.3.1　AnySee

AnySee 是华中科技大学自主研发的视频直播系统，支持 NAT 和防火墙的穿越，而且很好地利用了近播思想，有较好的 QoS 保证。它的体系结构如图 2-16 所示。

在 AnySee 系统中，所有对等节点连接成一个以节目源节点为根的多播树。每个节点从树中的父节点接收多媒体数据，同时提供给自己的子节点。节目源节点与其他节点之间是一种对等关系。编码服务器把捕获到的视频信号编码为合适的流媒体格式，发送给节目源节点。黄页服务器收集节目源节点的相关信息，如频道名称、频道介绍、在线人数等，并把这些信息发送给每个参与者。历史信息服务器收集系统中每个节点的活动信息，如在线状态、收看频道等。AnySee 的这种树结构保证了媒体数据的一次路由，减少了网络负载，保证了系统的

www.ckook.com

可靠性和可扩展性。

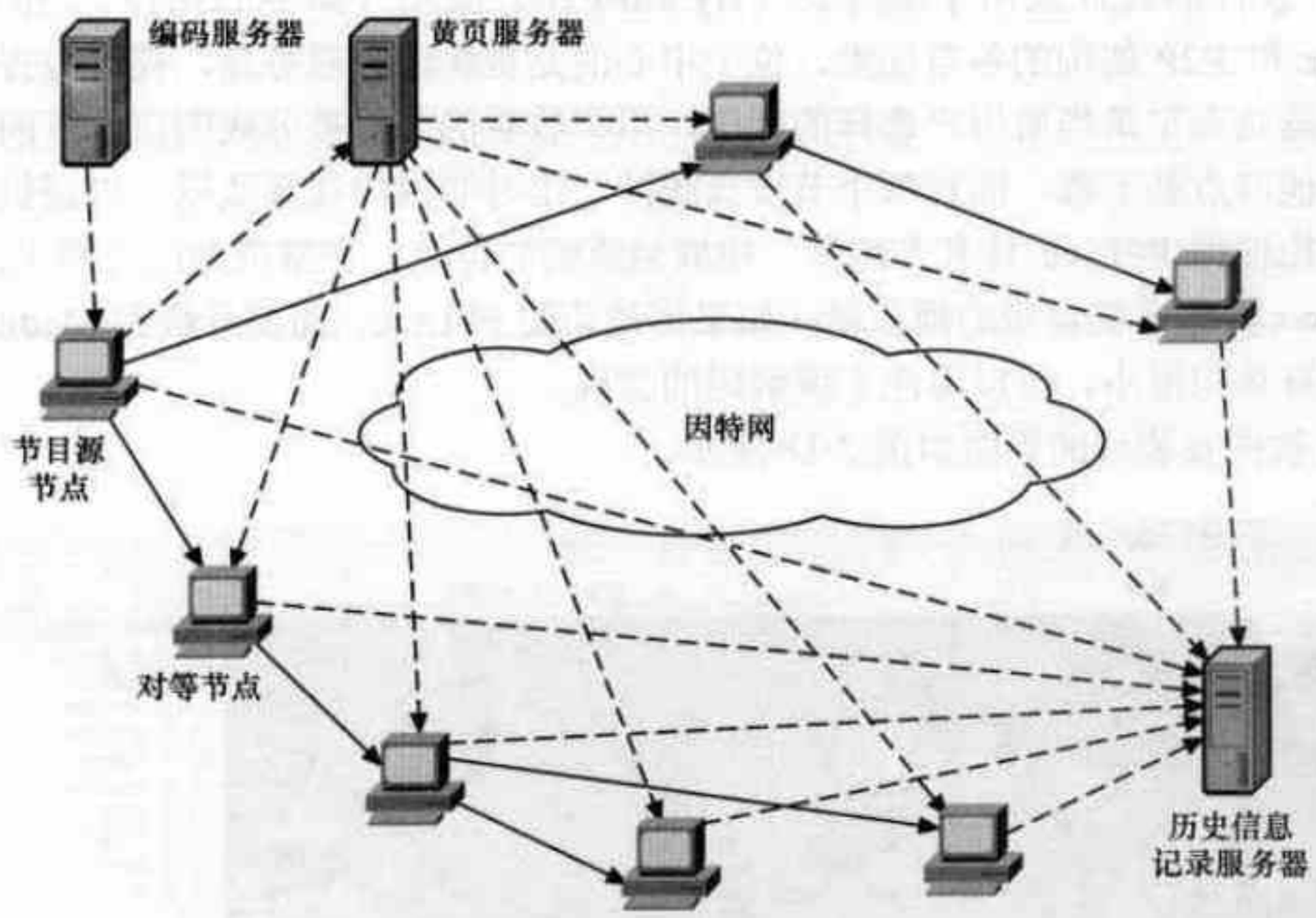


图 2-16 AnySee 系统结构示意图

AnySee 系统的实现有 3 个关键技术：应用层多播、Landmark 路标算法和数据缓冲机制。应用层多播和 Landmark 路标算法结合起来用来构建一棵稳健的多播树。当一个新的节点申请加入时，使用 Landmark 算法来把新节点就近加入到一棵已经存在的多播树中去。比如，在多播树中已经存在的节点集合为{A, B, C}，其中 A 在北京大学，B 在中科院，C 在北京邮电大学，这时在北京师范大学的节点 D 申请加入网络，D 会被调度到 C 的分支上，这样就可以保证 A 与 D 之间有较高的带宽。这种算法也可以保证在网络出现问题时只影响局部地区，不会造成大范围的网络瘫痪。数据缓冲机制是指每个节点都会缓存一定时间段内的流媒体数据，以保证质量。AnySee 的缓冲区管理机制主要有以下几条：（1）每个节点的数据缓冲区以时间间隔为单位进行管理；（2）缓冲区缓存最近 40s 长度内的媒体数据；（3）缓冲区占用的空间随时间动态变化；（4）对数据包头和数据内容分别进行管理；（5）每个节点缓冲的数据长度达到一定阈值才向子节点发送。

AnySee 系统曾经在 2004 年奥运会期间进行过网络视频直播，同年 11 月 11 日武汉地区的 896 个社区也使用了 AnySee 收看政务节目，试用效果良好。

AnySee 客户端是用 Java 开发的，运行时需要 JRE 的支持。软件运行时界面如图 2-17 所示。客户端软件只提供了简单的连接和配置功能，播放器用的是 Winmap 和系统自带的 Windows Media Player。

图 2-17 AnySee 软件运行界面

2.3.2 PPLive

PPLive 可以说是当前最流行、最热门的流媒体软件，是由华中科技大学的姚欣和上海聚

www.ckook.com

力传媒合作推出的。

PPLive 软件的设计采用了混合式（Hybrid-P2P）架构（即中心拓扑），继承和结合了 Client/Server 和 P2P 架构的各自优势。位于中心的是视频源和服务器，视频源提供多种视频服务，服务器负责记录当前用户选择的频道。用户收看的节目被分成固定长度的帧，可以从网络中的其他节点处下载，而且每个节点会缓冲一定时间内的视频数据，以提供给下一个用户。这种结构使得 PPLive 具有人越多、播放越流畅的特点。在播放器的选择上，PPLive 采用了 Windows 操作系统自带的播放器（如果用最新的 PPLive，需要升级到 Media Player10），使得软件本身体积很小，可以专注于编解码的实现。

PPLive 软件安装后的界面如图 2-18 所示。

图 2-18　PPLive 软件运行界面

界面左侧是播放器的主界面，右侧是以树状显示的节目列表。用户通过双击节目列表可以很方便地切换频道。如图 2-18 所示，PPLive 还提供了收藏频道等功能，让用户的选择更方便。

2.4　共享存储类系统

2.4.1　OceanStore

OceanStore 是由美国加州的计算机科学大师 John Kubiatowicz 提出的一个海量存储系统。

www.ckook.com

他的设想是人们利用这个系统可以从世界任何一个角落，通过任何一台计算机设备获得想要的任何资料。其主要设计构想如图 2-19 所示。

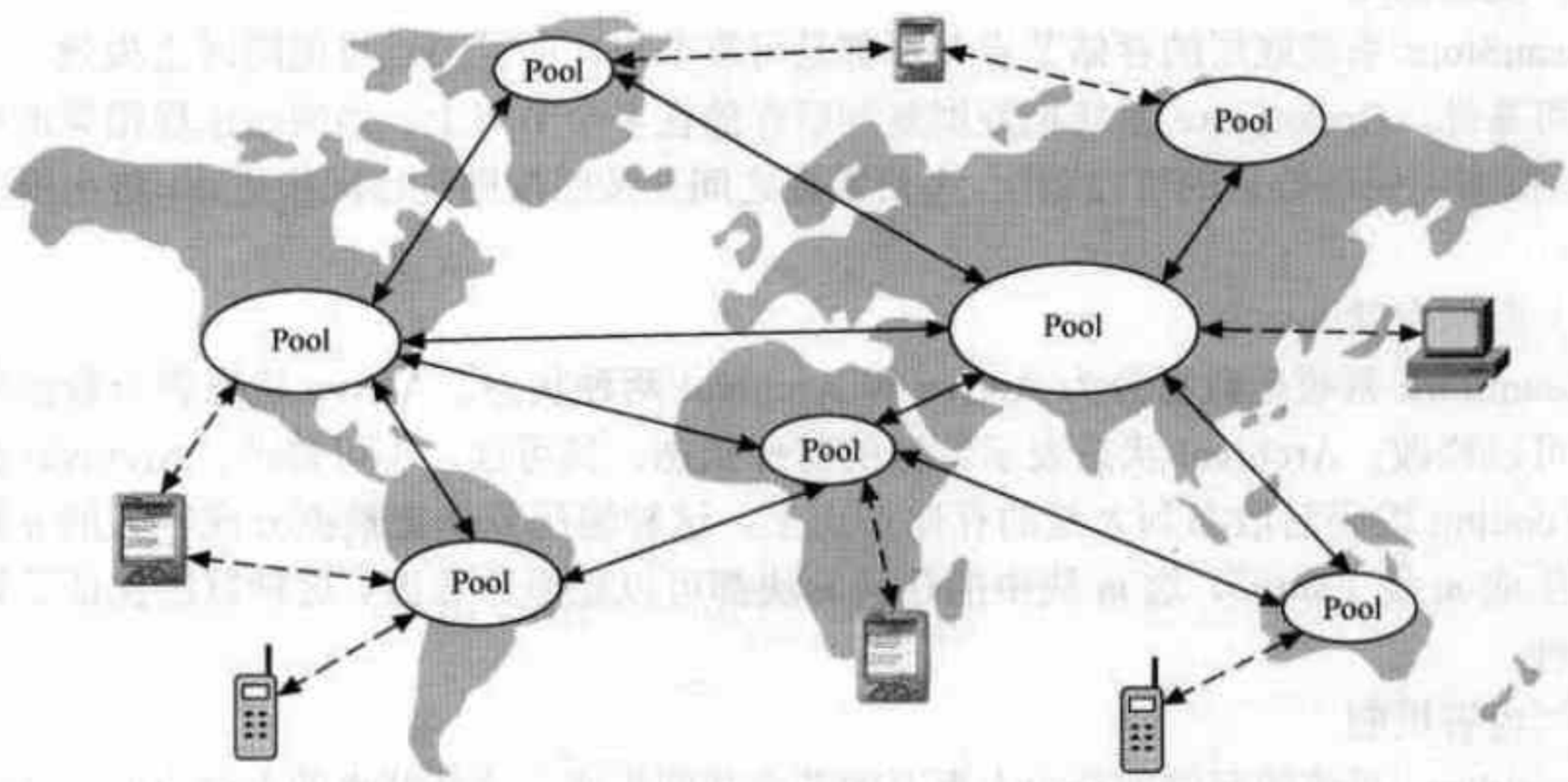


图 2-19 OceanStore 系统结构示意图

从图 2-19 中可以看出，OceanStore 是一个全球范围内的海量存储系统，它可以支持用户用一些嵌入式设备（如 PDA、手机等）来访问系统中的数据，真正随时随地接入。

OceanStore 核心的存储设备是由大量互相链接的存储节点共同组成的。这些节点多数是由经营存储服务的公司提供的，是专用节点，其他组织只要提供一定数量的存储节点和带宽能力也可以加入系统。用户是以付费的方式来享受存储服务的。这种框架也提供了一些可以商业化运营的模式。

OceanStore 系统的核心技术主要有以下几个方面。

（1）数据模式和访问接口

OceanStore 系统处理的基本数据是对象，对象的大小任意，由一个 128bit 的 Object ID 唯一标识。为了便于上层开发，OceanStore 给用户提供了一套完整的 API，用于对象数据的存储访问和控制。而且这些接口可以被再次封装，以满足从传统数据库到 OceanStore 的迁移。

（2）数据放置与路由

OceanStore 底层的结构化覆盖网为 Tapestry。Tapestry 是一个基于 DHT 来进行数据转发和网络连接的 P2P 协议，关于 Tapestry 的详细描述可以参考本书的后续章节。

OceanStore 并不直接使用 DHT 存储实际数据，而是仅仅利用 DHT 存储数据指针，也就是实际存储数据的节点的地址列表（通常一份数据会有多个副本）。由于指针比实际数据小很多，与直接使用 DHT 存储数据的方法相比，这种方法在节点进入、退出时由副本迁移所引发的带宽消耗会小很多。

（3）访问控制

由于要提供全球化的数据存储服务，数据的安全性和保密性显得异常重要。OceanStore 系统提供两种基本的访问控制：读控制和写控制。由这两种控制可以组合成更负责的控制策略。读控制是利用数据加密来实现的，所有的存储数据都是密文，密钥只发放给那些拥有读权限的用户。写控制是通过授权来实现的。数据的拥有者可以提供一份拥有写权限的用户列

www.ckook.com

表来实现写控制。除此之外，用户的所有写操作都必须要签名，这样就可以追踪所有的用户操作。

（4）复制策略

OceanStore 系统底层的存储节点并不都是可靠节点，而且节点可能随时会失效。为了保证数据可靠性，OceanStore 系统把数据复制后存储在多个节点上。当网络出现错误或被攻击时，能从概率上保证数据的可控制性。多份数据之间要按照规则进行同步更新，防止出现“脏”数据。

（5）归档存储

OceanStore 系统把数据分为 Active 和 Archival 两种状态。Active 状态表示数据的当前状态，可以修改；Archival 状态表示数据的曾经状态，只可读，不可修改。Archival 数据用 Erasure coding 编码后散布到大量的存储节点上。这种编码算法把数据分成等大的 n 块，通过编码生成 m 块（$m>n$），这 m 块中的任何 n 块都可以还原原数据。这种算法保证了数据的高可靠性。

（6）内省机制

OceanStore 系统的存储部分由上百万的节点共同组成，这些节点的存储空间、带宽和计算能力各不相同，而且各个节点不断地加入、退出或者失效。为了平衡这些负载，OceanStore 引入了内省机制，让系统对环境进行自适应。

目前，OceanStore 的一个原型系统 Pond 已经放在 sourceforge 网站上，包括相关的二进制文件、源代码和项目文档。这个原型系统是用 Java 开发的，可以运行在 Debian 或者 Red Hat 操作系统上。感兴趣的读者可以到 http://oceanstore.sourceforge.net 上下载相关资料。

2.4.2 Granary

Granary 系统是由清华大学自主研发的 P2P 存储系统。与已有的存储系统相比，Granary 系统以面向对象的方式提供数据存储和访问方式。这种思想和面向对象的程序设计概念类似，大大简化了上层的编码实现。上层的应用开发者可以直接将面向对象应用程序的对象数据存储到 Granary 中，随后通过查询直接获取数据，而不需要额外地进行应用程序和存储系统间的数据转换。Granary 系统的层次示意图如图 2-20 所示。

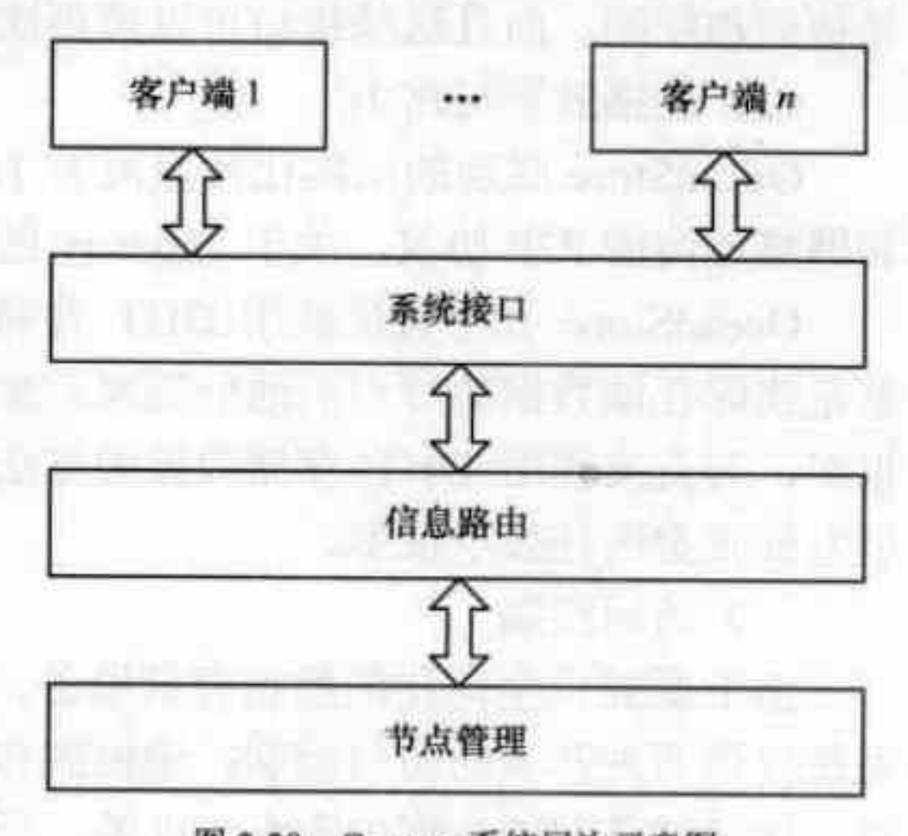


图 2-20 Granary 系统层次示意图

节点管理位于 Granary 的最底层，采用了 PeerWindow 节点收集协议来管理各个节点。每个 Granary 节点都拥有一个 128bit 的 Node ID，称为节点标识。节点标识在地址空间是均匀分布的（通过 Hash 函数来实现）。每个节点都会保存一个节点列表，列表中每一个值都是指向其他节点的指针，指针的内容由节点的地址、端口、级别等信息构成。级别由各个节点独立决定，设为 0、1、2…。规定一个 l 级的节点拥有所有 Node ID 的前 l 位与自己相同的节点的指针。这样，0 级节点知道系统中所有节点的信息，1 级节点知道一半节点的信

www.ckook.com

息，l 级节点知道大约 $1/2^l$ 个节点的信息。而且节点的级别可以在运行时自己调整。节点列表的维护通过“探测一多播”的机制来实现，每个节点加入、离开或者改变级别都有别的节点会探测到该事件并多播到网络中。

信息路由部分负责把信息传送到符合条件的节点上。Granary 采用了 Tourist 路由协议，它的基本思想是构建两个对称的 PeerWindow，一个常规的 PeerWindow 按正常方式运行，另一个为反转的 PeerWindow，它使用后缀匹配的算法，即 l 级节点维护所有 Node ID 的后 l 位与自己相同的节点的指针。利用这种结构，即使在 100 万个节点、平均生存周期为 1h 的系统中，Tourist 仍可以保证所有路由在两步之内完成，具有非常高的路由效率。

系统接口是 Granary 为上层提供的程序开发接口，使用户可以按照对象的格式存取数据。对象由一系列属性组成，其组成格式为<名称、类型、可见性、值>。名称唯一标识了该对象；类型是该对象的数据类型，可以为数值、字符或者二进制数据；可见性表示该对象的访问权限以及是否应该被加密；值是该对象的实际数据。用户对其存储的数据可以通过查询的方式来读取，如在存储的电影中搜索“国内影片”，系统会返回所有符合条件的列表。存储数据时，Granary 采取了多副本的方式，每类数据可以放置到多个节点上，以保证可靠性。

应用程序是基于 Granary 开发的外部程序，遵循存储管理的接口规范。

目前 Granary 系统已经向公众开放了，读者可以到 Granary 的官方网站 http://granary.net9.org/index.html 上下载客户端软件，然后用自己的电子邮件地址注册一个账号就可以使用 Granary 系统了。Granary 客户端 gclient 是用 Java 语言开发的，有 Windows 和 Linux 两个操作系统的版本，而且有支持 IPv6 的版本。gclient 运行界面如图 2-21 所示。

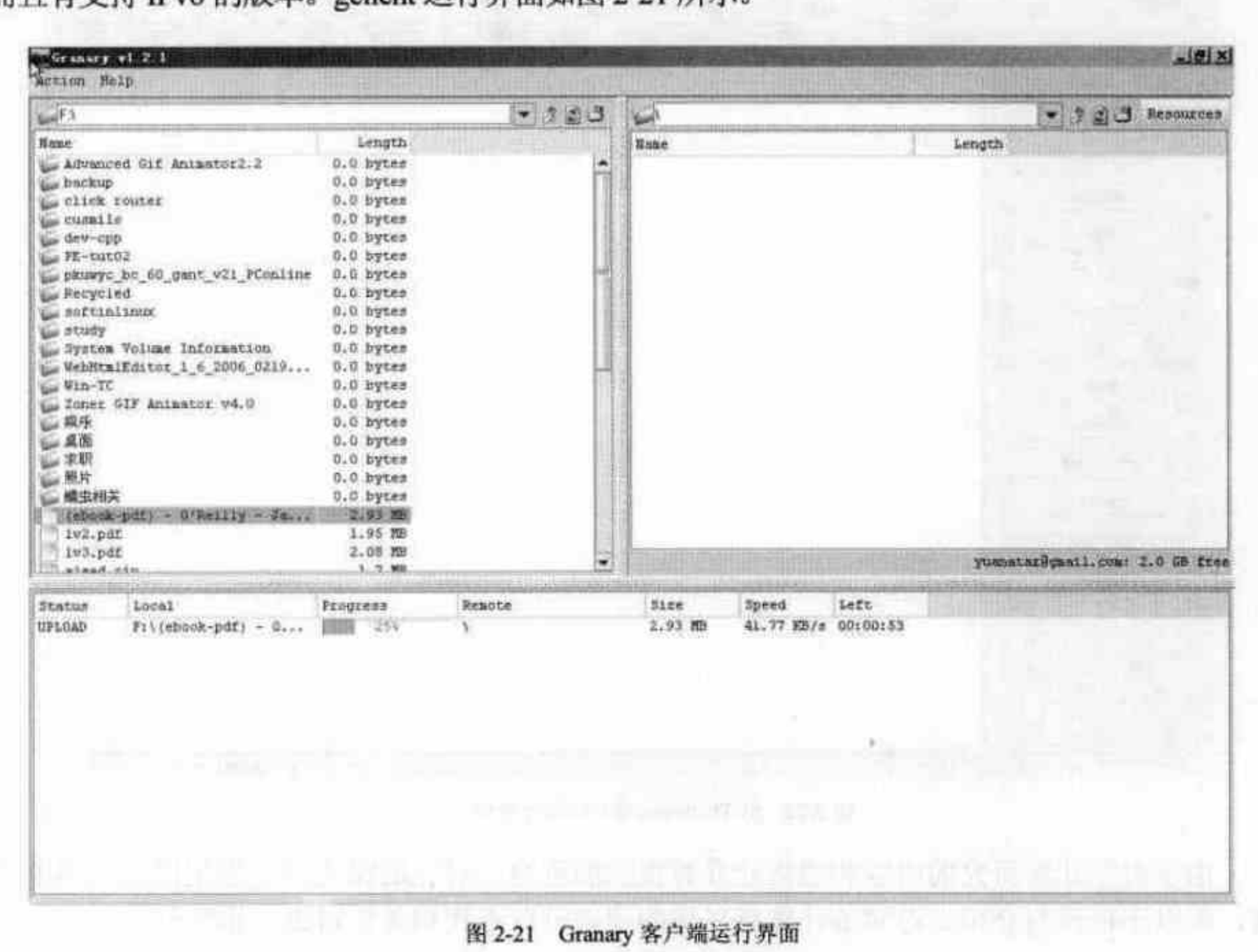


图 2-21 Granary 客户端运行界面

www.ckook.com

图中最下面的窗口区域是当前进行的操作动作显示，如上传、下载、进度显示以及剩余空间等。

2.5 对等计算类系统

对等计算的思想是利用因特网把全世界闲置的计算资源组合起来，形成一个非常巨大的计算能力来解决目前常规计算不能解决的问题。其中，比较具有代表性的就是 SETI@home 项目。

SETI@home 是 Search for Extraterrestrial Intelligence（搜寻外星智能）的缩写。该项目由美国加州大学伯克利分校的空间科学实验室主办，试图通过分析 Arecibo 射电望远镜采集的无线电信号，搜寻能够证实外星智能生物存在的证据。SETI@home 程序在用户的个人计算机上，通常在屏幕保护模式下或以后台模式运行。它利用的是多余的处理器资源，不影响用户正常使用计算机。目前，该项目在世界各地拥有近 500 万参与者，1999 年 5 月 17 日开始正式运行。到 2004 年 5 月，已经积累了近 200 万年的 CPU 运行时间，进行了近 5×10^{21} 次浮点运算，处理了超过 13 亿个数据单元。SETI@home 客户端软件运行界面如图 2-22 和图 2-23 所示。

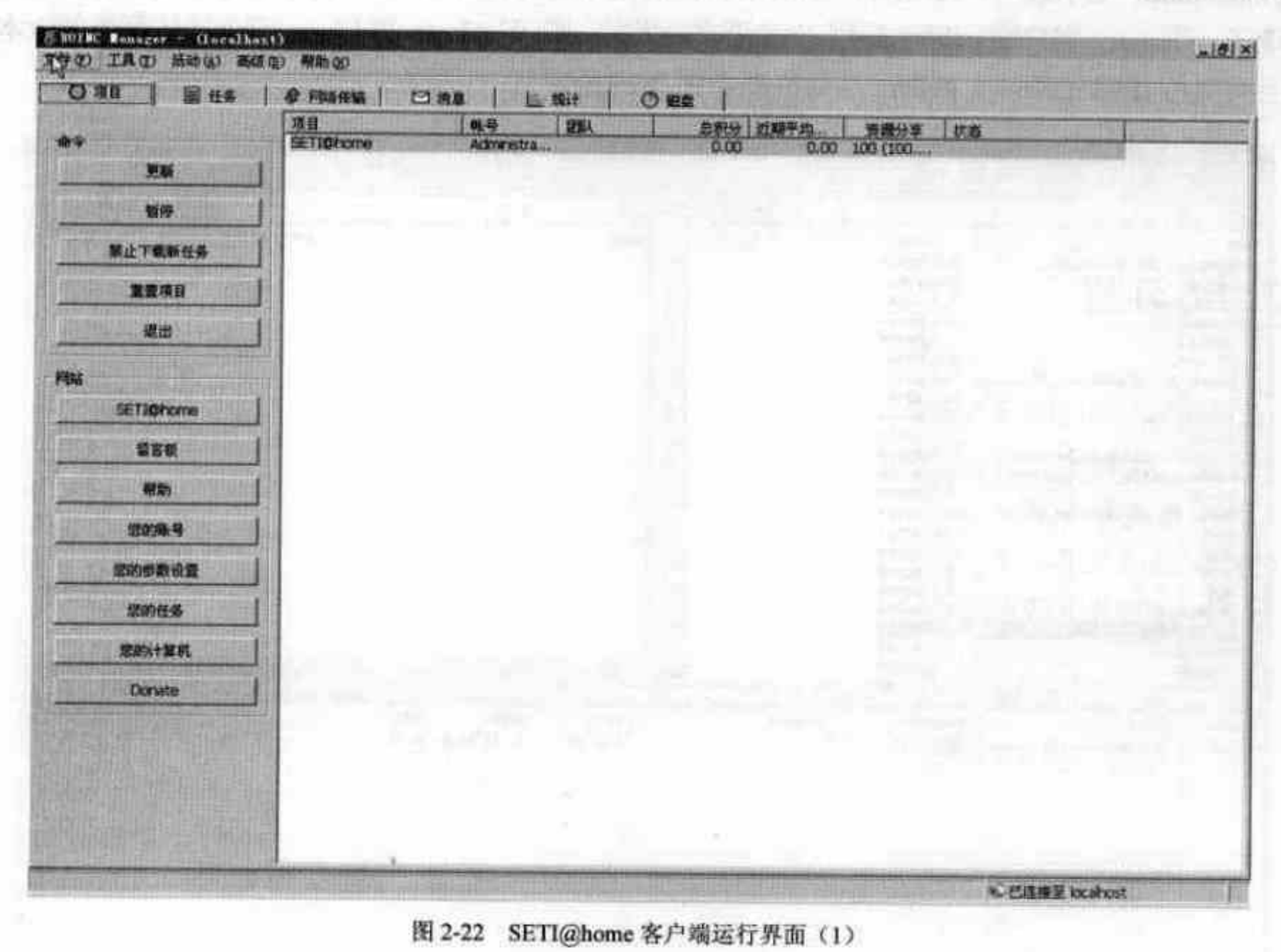


图 2-22 SETI@home 客户端运行界面（1）

由于对等计算研究的内容和网格计算有很大的重合，对它的深入讨论也超出了本书的范畴，所以不再详细介绍。对对等计算感兴趣的读者可以查找相关资料进一步学习。

www.ckook.com

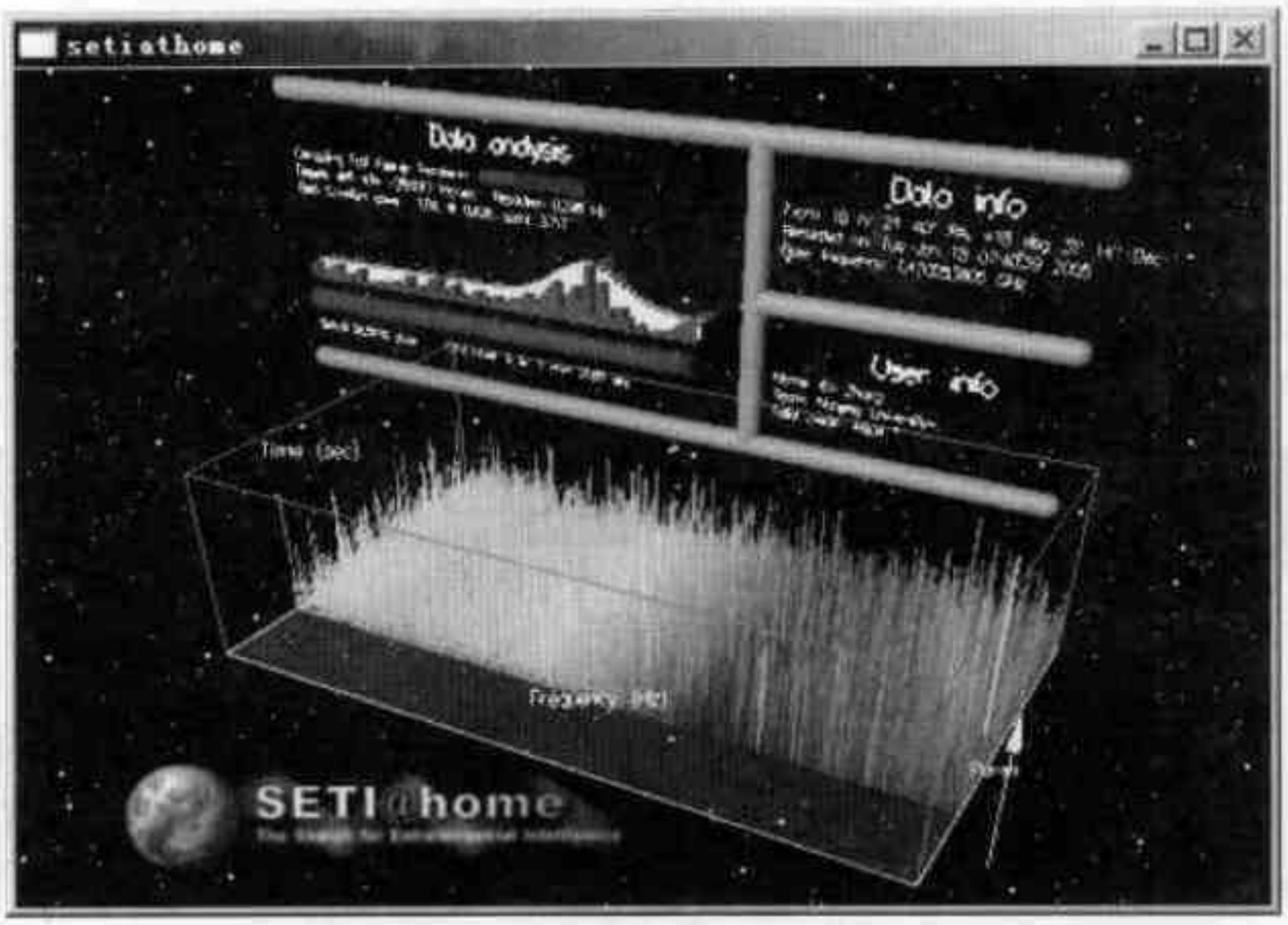


图 2-23 SETI@home 软件运行界面（2）

2.6 本章总结

本章是在读者对 P2P 基本概念有所了解的基础上，分类介绍了当前的主流 P2P 系统。按照用户，P2P 系统可以分为文件共享、即时通信、流媒体、分布式存储和对等计算 5 大类。本章对每一类系统都列举了丰富的例子，为读者进一步的学习和开发打好基础。

2.7 练习题

（1）按照不同的标准来对 P2P 系统进行分类。

注意：本章对 P2P 系统是按照用途来分类的，每一类都代表一种典型的应用。然而，P2P 系统的分类是有多种标准的。读者可以按照自己的理解或者查阅相关资料来重新对本章介绍的 P2P 系统以及其他 P2P 系统进行分类。

（2）请读者下载试用各种类型的 P2P 软件。

参 考 文 献

[1] 郑纬民，等. 对等计算研究概论.清华大学计算机科学与技术系高性能计算研究所，2005
[2] 罗杰文. Peer to Peer 综述. 中科院计算技术研究所，2005
[3] www.napster.com
[4] www.gnutella.com
[5] http://lumaqq.linuxsir.org/main/
[6] http://granary.netq.org/index.html

www.ckook.com

CHAPTER 3

第 3 章　P2P 网络的基础——搜索和路由算法

在 P2P 网络中，如何准确地定位节点是非常重要的，本章将从传统搜索技术与 P2P 搜索技术的对比开始，详细地介绍 P2P 网络的技术特点，着重描述分布式 P2P 网络的路由算法。

在学习了本章后，希望读者能够掌握以下几点：

（1）传统搜索技术与 P2P 搜索技术的区别；

（2）P2P 网络的类型及相应的技术特点；

（3）结构化的分布式 P2P 网络的路由原理；

（4）非结构化的分布式 P2P 网络的路由原理。

3.1　传统搜索技术

传统的搜索技术通常是指搜索引擎技术，搜索引擎以一定的策略在因特网中搜集、发现信息，对信息进行理解、提取、组织和处理，并为用户提供检索服务，从而起到信息导航的作用。

按照信息搜集方法和服务提供方式的不同，搜索引擎系统可以分为 3 大类，如表 3-1 所示。

表 3-1　搜索引擎系统分类

搜索引擎技术类别	搜索引擎技术描述
目录式搜索引擎	以人工方式或半自动方式搜集信息，由编辑员查看信息之后，人工形成信息摘要，并将信息置于事先确定的分类框架中，由于 Web 信息的海量性和人工处理能力、经济代价的限制，这类搜索引擎信息的即时性和全面性难以保证
机器人搜索引擎	由一个称为蜘蛛的机器人程序以某种策略自动地在因特网中搜集和发现信息，由索引器为搜集到的信息建立索引，由检索器根据用户的查询输入检索索引库，并将查询结果返回给用户，这类搜索引擎实现较为复杂，但能很好地实现信息的全面获取和即时性
元搜索引擎	这类搜索引擎没有自己的数据，而是将用户的查询请求同时向多个搜索引擎递交，将返回的结果进行重复排除、重新排序等处理后，作为自己的结果返回给用户，这类搜索引擎集合了多个搜索引擎的信息，并且加入新的排序和信息过滤，可以很好地提高用户满意度

www.ckook.com

在这 3 类搜索引擎技术中，第 2 类技术为目前的主流技术。它的实现主要包括 3 个关键点，如表 3-2 所示。

表 3-2 机器人搜索引擎实现关键点

关 键 点	细 节 描 述
利用网络蜘蛛获取网络资源	这是一种半自动化的资源获取方式。所谓半自动化，是指搜索器需要人工指定 URL（Uniform Resource Locator，统一资源定位系统），然后获取该 URL 所指向的网络资源，并分析该资源所指向的其他资源，然后获取。在实际的蜘蛛程序中，为了保证信息收集的全面性和及时性，还有多个蜘蛛程序的分工和合作问题，往往有复杂的控制机制
利用索引器抽取信息并建立索引表	当用网络蜘蛛获取资源后，需要对这些资源进行加工过滤，提取有用信息用一定的模型表示，使查询结果更为准确。其中信息的表示模型一般有布尔模型、向量模型、概率模型和神经网络模型等。Web 上的信息一般表现为网页，为每个网页生成摘要，此摘要将显示在查询结果的页面中，告诉查询用户各网页的内容概要。模型化的信息将存放在临时数据库中，由于 Web 数据的数据量极为庞大，为了提高检索效率，须按照一定规则建立索引
检索及用户交互	在信息索引库的基础上，接受用户查询请求，并到索引库检索相关内容，返回给用户。这部分的主要内容包括：用户查询理解，即最大可能贴近地理解用户通过查询串想要表达的查询目的，并将用户查询转换为后台检索使用的信息模型；根据用户查询的检索模型，在索引库中检索出结果集；通过特定的排序算法，对检索结果集进行排序

搜索引擎的评价指标主要包括响应时间、查全率、查准率和用户满意度等。

（1）响应时间是从用户提交查询请求到搜索引擎给出查询结果的时间间隔，响应时间必须在用户可以接受的范围之内。

（2）查全率是指查询结果集信息的完备性。

（3）查准率是指查询结果集中符合用户要求的数目与结果总数之比。

（4）用户满意度是一个难以量化的概念，除了搜索引擎本身的服务质量外，它还和用户群体及网络环境有关系。在搜索引擎可以控制的范围中，其核心是搜索结果的排序，即如何把最合适的结果排到前面。

目前，搜索引擎技术作为因特网搜索技术中应用最为广泛、用户数量最多的技术，已经趋于成熟，用户满意度也保持在一个可以接受的水平。当前搜索引擎服务提高的挑战主要来自于以下 3 方面。

（1）用户初始查询请求描述具有不准确性，用户往往难以清楚地表达其信息需求。对如何根据用户模糊的查询请求，较为准确、全面地理解用户的实际信息需求的研究，将是搜索引擎技术研究的一个重要方面。

（2）挖掘用户的个性化信息，将是一个能产生突破性成果的方法。根据用户的背景信息，如知识水平、专业方向、职业和兴趣等，将极大地促进用户初始查询理解的准确性和全面性。另外，对用户网络使用的日志分析，也能提供准确的用户网络行为，从而为用户信息需求分析提供重要依据。

（3）搜索引擎服务质量提高的另一个挑战是，查询结果集过于庞大，用户往往不会看到位于结果集后部的信息。这方面目前已经基本确定的解决方案是，根据分类体系组织结果集，通过类别选择缩小结果范围。它的实现难点是分类体系的确定和最终页面归属类别的确定。

3.2 P2P 搜索技术的发展

与传统搜索技术相对立的就是目前正在兴起的 P2P 搜索技术。在 P2P 网络中，用户共享

www.ckook.com

所有用户硬盘上的文件、目录乃至整个硬盘。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，可达到传统搜索引擎无可比拟的深度。

就网络模型而言，P2P 并不是新的概念，它可以说是因特网整体架构的基础。因特网最基本的协议 TCP/IP 协议簇并没有客户端和服务器的概念，所有的设备都是通信中平等的一端。在最早时期，所有因特网上的系统都同时具有服务器和客户端的功能。然而，受早期计算机性能、资源等因素的限制，随着因特网规模的迅速扩大，逐步形成了以少数服务器为中心的客户端/服务器（Client/Server）架构。WWW 的风靡，正是这一应用潮流的体现。

但是，随着因特网的发展，人们之间的联系日益紧密和深入，人们需要更直接、更广泛的信息交流。普通用户希望能够更全面地参与到因特网的信息交互中，而计算机和网络性能的提升也使其具有了现实的可能性。在此背景下，P2P 再一次受到了广泛的关注。

新一代的 P2P 网络搜索技术主要分为 4 类，如表 3-3 所示，它们的性能比较如表 3-4 所示。

表 3-3　　P2P 网络搜索技术

网络类别	技术特点及代表系统
集中式 P2P 网络	集中式 P2P 网络通过一个中心服务器来记录共享信息以及回答对这些信息的查询。每一个对等实体负责共享它的信息，下载它所需要的存储在其他对等实体上的信息。这种形式具有中心化的特点，但是它不同于传统意义上的 Client/Server 模式。所有信息都分别存放在提供该信息的客户端上，服务器上只保留索引信息，此外服务器与对等实体以及对等实体之间都具有交互能力。这类网络的代表系统是 Napster 和 eMule
结构化分布式 P2P 网络	结构化分布式 P2P 网络是一种纯 P2P 网络。这种网络不需要有中心服务器和中心路由器，其上的每一个 Peer 都作为对等实体，地位是完全平等的。每一个 Peer 既可以作为客户端又可以作为服务器，并且它们与相邻的 Peer 有相同的能力。这类网络中每一个节点被分配一个虚拟地址，同时用一个关键字来表示其可提供的共享内容。网络中各个节点分别存储自己的虚拟地址路由表进行路由。关键字存储在与关键字匹配的虚拟地址上。资源定位的时候，可以通过虚拟地址路由表快速查询到存储关键字的节点，从而获得共享内容的存储位置。这类网络的代表系统是 Chord、Pastry、CAN、Tapestry、Kademlia
非结构化分布式 P2P 网络	非结构化分布式 P2P 网络和结构分布式 P2P 网络一样，也是一种纯 P2P 网络，只是在网络拓扑结构上有所不同。这类网络中每个节点都拥有自己的邻居（Peer-group）。查询文件时，节点向自己所有的邻居发送查询数据包。每一个收到查询数据包的节点将检查在自己本地存储的文件是否满足查询要求。如果满足的话，该节点发送一个查询响应数据包给查询的初始发起者，节点间直接交换文件。不管满足与否，该用户都继续将查询数据包向自己的邻居节点转发。以此类推，查询消息像洪水（Flood）一样在网络中流动。这类网络的代表系统是 Gnutella
混合式 P2P 网络	混合式 P2P 网络是集中式 P2P 网络和分布式 P2P 网络的集合。集中式 P2P 网络有利于网络资源的快速检索，以及只要服务器能力足够强大就可以无限扩展，但是其中心化的模式容易遭到直接的攻击；分布式 P2P 网络具有良好的抗攻击性，但是又缺乏快速搜索的能力以及可扩展性。混合式 P2P 网络在分布式网络的基础上，将节点按能力进行分类，使某些节点担任特殊的任务，称为超级节点。超级节点分担了网络大部分的检索、路由和扩展功能，使普通节点的负载下降。这类网络的代表系统是 Skype

表 3-4　　P2P 网络性能比较

网络结构/比较标准	集中式 P2P 网络	结构化分布式 P2P 网络	非结构化分布式 P2P 网络	混合式 P2P 网络
可扩展性	差	好	差	中
可靠性	差	好	好	中
可维护性	最好	好	最好	中
查找算法效率	最高	高	中	中
复杂查询	支持	不支持	支持	支持

在以上 4 类 P2P 网络中，集中式 P2P 网络的技术特点相对比较简单，而分布式 P2P

www.ckook.com

网络由于没有服务器，完全依靠网络中的节点实现路由和网络拓扑，其搜索和路由算法相对比较复杂。本章的后面几节将着重对结构化 P2P 网络和非结构化 P2P 网络的搜索技术进行分析。

3.3 DHT 网络（结构化 P2P 网络）的搜索技术

结构化 P2P 网络指的是像 Chord、Pastry、CAN、Tapestry 之类的点对点网络。在这类网络中，每个节点都有固定的地址，整个网络具有相对稳定而规则的拓扑结构。依赖拓扑结构，可以给网络的每个节点指定一个逻辑地址，并把地址和节点的位置对应起来。对于给定的某个地址，拓扑结构保证只需要通过 $O(\log(n))$ 跳就能路由到具有相应地址的节点上去（n 是网络中的节点数）。在实际的系统中，P2P 网络的逻辑地址通常是由 Hash 函数得到的，每个节点都将保存一张 DHT（Distributed Hash Table）进行路由，所以结构化 P2P 网络通常也叫作 DHT 网络。

给定存储信息的索引，通过 DHT 能够高效率地定位到对应该索引的信息。但是 DHT 网络在全文检索方面，面临着巨大的挑战，目前没有很好的解决方案。另外，DHT 网络还面临本身固有的一些问题，如负载均衡不易，网络拓扑维护代价大，索引的同步维护困难等。

3.3.1 DHT 路由原理

在结构化 P2P 网络中，网络中的客户端主机称为节点，数据项目称为对象。名字空间指系统的一个名字域，在这个名字域中所有的名字是独一无二的。名字用来标识节点，一个节点可以有几个名字，每个名字基于不同的尺度。但总的来说，每个节点在一个名字空间中只能有一个名字。对于一个节点来说，典型的名字就是它的因特网 IP 地址。标识符是指在某个整数名字空间中的独一无二的整数。在 P2P 系统中，标识符可以通过对一个节点的名字进行散列来获得，例如标识符 $P=\mathrm{Hash}(IP)$。关键字（Key）是一个对象独一无二的标识符，它可以通过对对象名进行散列来获得，$K=\mathrm{Hash}(V)$。

散列函数可以将 Key 映射到整数，通常可以在一个更小的值的集合上获得一个离散分布。散列函数是不可逆的。散列表（Hash Table）是一个字典，在这个字典中 Key 通过一个散列函数被映射到一个数组位置上。如果有多个 Key 被映射到一个位置上叫做冲突。完美的散列函数是一个没有冲突的散列函数，每个不同的 Key 都被映射到不同的整数上。使用完美的散列函数的散列表没有冲突。最小的完美散列函数映射每个不同的 Key 到不同的整数，并且可能的整数的数目和可能的 Key 的数目相同。

在基于 DHT 的 P2P 系统中，文件被关联到 Key（由文件名的散列产生）；系统中的每个节点拥有一部分散列空间，它负责保存某个范围的 Key。在对一个 Key 进行查询后，系统将返回一个保存具有 Key 的对象的节点的标识符（如 IP 地址）。DHT 的功能允许节点基于文件的 Key 来存入和取出文件，这已经被证明是在分布式系统中的一种有用的底层基础。

基于 DHT 的搜索实现主要包括 4 个关键点，如表 3-5 所示。

www.ckook.com

表 3-5　　DHT 搜索实现关键点

关 键 点	细 节 描 述
散列表的建立	散列值的产生使用一个基本的散列函数，比如 SHA-1；节点的标识符采用节点名字（如 IP 地址）的散列值；对象的标识符采用对象名的散列值；每个节点存储一张散列表，存储节点标识符与节点物理地址的映射
内容的查找	内容查找通过<Key，Value>对来查找，Key 在这里对应着对象标识符，Value 在这里对应文件的名称、所在的节点的地址等信息
定位关键字所在的节点	将各个节点所具有的<Key，Value>对保存在与对象标识符相近的节点标识符的机器中，使搜索关键字代表的对象与节点地址关联上
<Key，Value>对的流动	当有新节点或新的<Key，Value>对出现时，将对应的<Key，Value>对转移到对应的节点上；当旧节点离开时，将其存储的<Key，Value>对转移到相邻的节点上

目前典型的 DHT 网络包括 Chord、Pastry、CAN、Tapestry、Kademlia 等网络，它们的主要区别在于采用了不同的 DHT 路由算法，这也决定了各类网络在逻辑拓扑上的不同。

3.3.2　Chord

Chord 在 2001 年由麻省理工学院提出，其核心思想就是要解决在 P2P 应用中遇到的基本问题：如何在 P2P 网络中找到存有特定数据的节点。Chord 专门为 P2P 应用设计，因此考虑了在 P2P 应用中可能遇到的特殊问题。Chord 在 CFS（Cooperative File System）中得到了应用。

1．路由算法

Chord 协议中规定使用的散列算法是 SHA-1，它在此基础上提供了优化的路由算法。

在 Chord 中，每个节点需要存储 m 个其他节点的信息，这些信息的集合被称为查询表（Finger Table）。在 Chord 中，表格中的节点间的间距以 2^{k-2} 的关系排列（k 表示表中的数组下标）。这样形成的节点之间路由关系实际上就是折半查找算法需要的排列关系。Chord 查询表如表 3-6 所示，其中 n 为网络中的最大节点数，m 为查询表级数（即 lb$n$），a 为本地节点的 ID。

表 3-6　　Chord 查询表

查 询 表 项	定　义
finger[k].start	$(a+2^{k-1}) \bmod 2^m$, $1 \leqslant k \leqslant m$
.interval	(finger[k].start, finger[k+1].start)
.node	first node $\geqslant$ a.finger[k].start
Successor	在环上离本地节点最近的后一个节点，也就是 finger[1].node
Predecessor	在环上离本地节点最近的前一个节点

在 Chord 中，关键字存储在与键值最接近的前向节点上，如表 3-1 所示，在这个 Chord 网络中，n 为 8，当前有 3 个节点，分别为节点 0、节点 1 和节点 2。键值为 1 的关键字信息存储在与它最接近的前向节点 1 上，键值为 2 的关键字信息存储在与它最接近的前向节点 3 上，键值为 6 的关键字信息存储在与它最接近的前向节点 0 上。

www.ckook.com

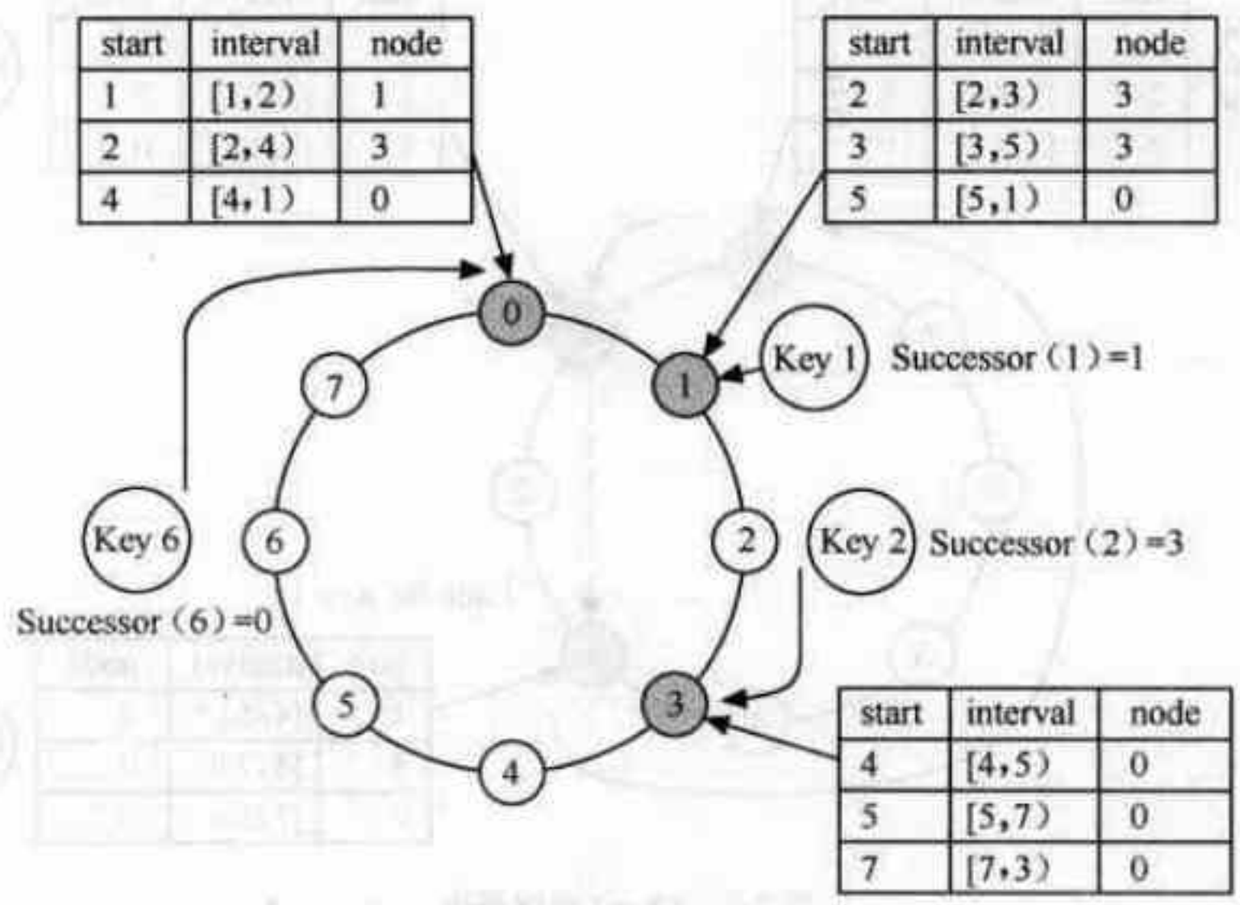


图 3-1 Chord 网络关键字存储

在查询的过程中，查询节点将请求发送到与键值最接近的节点上。收到查询请求的节点如果发现自身存储了被查询的信息，可以直接回应查询节点；如果被查询的信息不在本地，就根据查询表将请求转发到与键值最接近的节点上。这样的过程一直持续到找到相应的节点为止。不难看出，查询过程实际上就是折半查找的过程。如图 3-2 所示，节点 3 想要查询 Key1，它首先检查自己是否存储了 Key1，然后通过查询表确定与 Key1 键值最接近的节点 0，发送查询消息；节点 0 收到查询消息后，检查自己是否存储了 Key1，然后通过查询表确定与 Key1 键值最接近的节点 1；节点 1 收到查询消息后，确定 Key1 存储在本地，向节点 0 确认。Chord 查询的伪代码如下：

```
// 询问节点 a 查找与 ID 匹配的后向节点
a.find_successor(id)
   a' = find_predecessor(id);
   return a'.successor;

// 询问节点查找与 ID 匹配的前向节点
a.find_predecessor(id)
   a' = a;
   while (id ∉ [a', a'.successor])
         a' = a'.closest_preceding_finger(id);
   return a';

// 返回查询表中与 ID 最接近的节点
a.closest_preceding_finger(id);
   for i = m down to 1
      if (finger[i].node ∈ [a, id])
         return finger[i].node;
   return a;
```

www.ckook.com

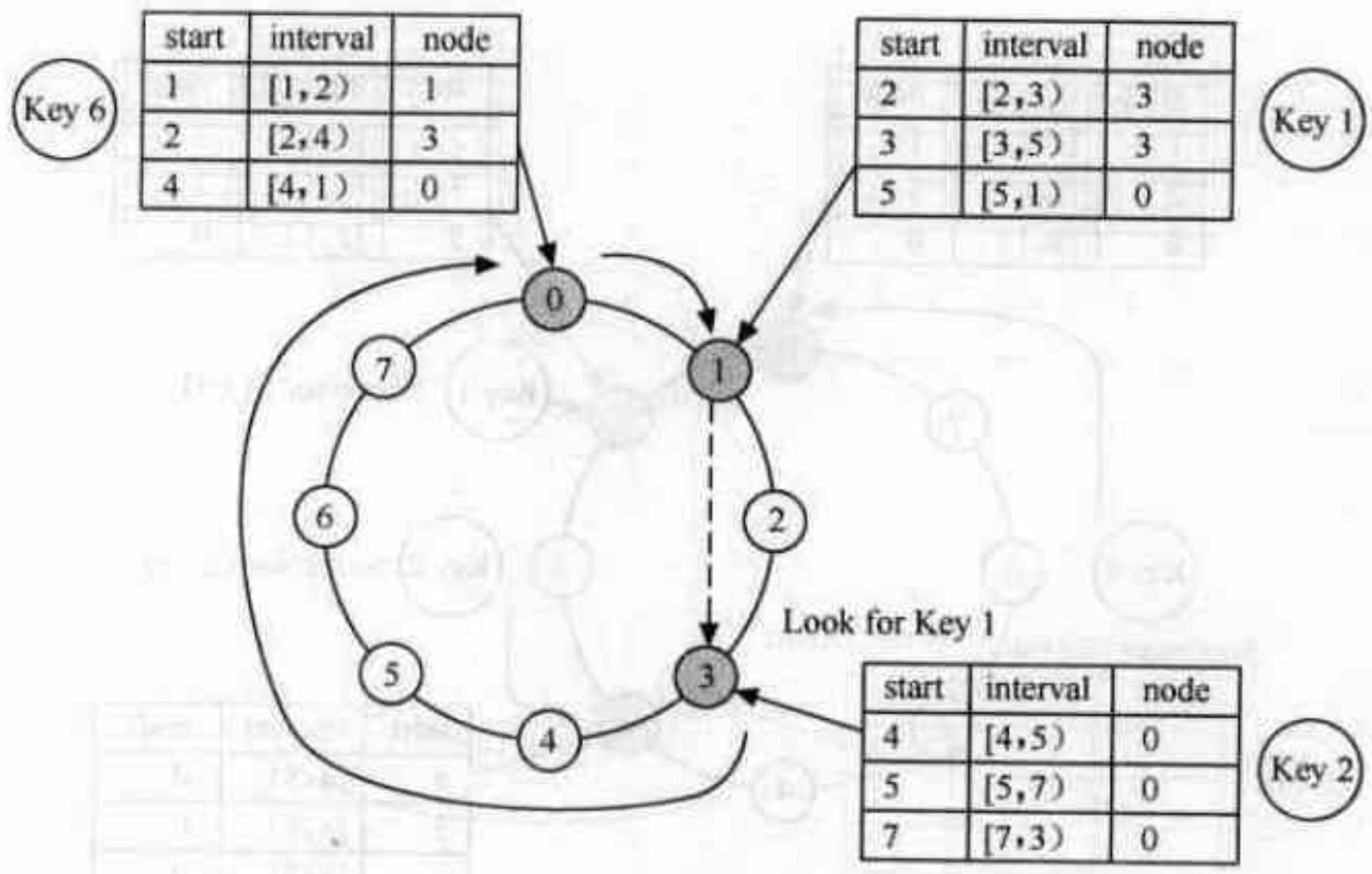


图 3-2　Chord 网络查询

在 Chord 中，查询需要的跳数为 O（log（n））。这样即使在大规模的 P2P 网络中（例如 n=100 000 000），查询的跳数也仅为 O（27），每个节点仅需存储 27 个（lb(100 000 000)）路由节点的信息。Chord 还考虑到多个节点同时加入系统的情况并对节点加入和退出算法作了优化。

2．节点的加入和离开

在 Chord 中，节点的加入分为 3 个阶段，如表 3-7 所示。

表 3-7　　节点加入阶段

加入阶段	细节描述
初始化本地节点查询表	节点 a 加入网络时必须知道网络中的某个节点 a'。这时，为了初始化 a 的查询表，a 将要求节点 a'为它查找查询表中的其他表项
更新其他节点查询表	节点加入网络后将通知其他节点，让其他节点即时地更新自己的查询表
转移关键字信息	其他节点将所有键值归属于 a 的关键字（即后继节点是 a 的关键字）转移到节点 a 上

整个节点加入的时间复杂度是 O（$\log^2$（n）），如果采用更加复杂的算法，可以把复杂度降低到 O（log（n））。Chord 节点加入的伪代码如下所示。

```
#define successor finger[1].node
// 节点 a 加入网络
// a'是网络中的任意一个节点
a.join(a')
  if (a')
      init_finger_table(a');
      update_others();
      // 将键值归属于 a 的关键字信息从其他后继节点转移到 a 上
  else // a 是网络中唯一的节点
      for i = 1 to m
          finger[i].node = a;
      predecessor = a;
```


www.ckook.com

```
// 初始化本地节点的查询表
// a'是网络中的任意节点
a.init_finger_table(a')
   finger[1].node = a'.find_successor(finger[1].start);
   predecessor = successor.predecessor;
   successor.predecessor = a;
   for i = 1 to m – 1
      if (finger[i + 1].start ∈ [a, finger[i].node])
         finger[i + 1].node = finger[i].node;
      else
         finger[i + 1].node = a'.find_successor(finger[i + 1].start);

// 更新其他与 a 相关的节点的查询表
a.update_others()
   for i = 1 to m
         // 寻找路由表的第 i 级可能为 a 的最后一个节点 p
         p = find_predecessor(a – 2^(i-1));
         p.update_finger_table(a, i);

// 判断 a 是否超前于 p 查询表中的第 i 级节点，如果是，用 a 替换 p 查询表的第 i 级节点
p.update_finger_table(a, i)
      if (a ∈ [p, finger[i].node])
         finger[i].node = a;
         p' = predecessor; // 获取 p 的第一个前向节点
         p'.update_finger_table(a, i);
```

在 Chord 中，当节点 a 退出时，所有在查询表中包括 a 的节点都必须把 a 替换成 a 的后继节点。同时，a 将自己负责的关键字信息转移到它的后继节点上。为了保证在退出时能维护正确的后继节点，每个节点都维护一张包括 r 个最近后继节点的后继列表。如果节点 a 注意到它的后继节点失效了，它就用后继列表中第一个正常节点替换失效节点。

3．算法分析

通过对 Chord 路由算法的分析，可以得出该协议具有以下几方面的优点，如表 3-8 所示。

表 3-8　　Chord 协议优点

协议优点	描　述
负载平衡	Chord 中采用一致性散列算法，该算法本身保证了其平衡性。所有节点以同等的概率分担系统负荷，从而可以避免某些节点负载过大的情况
分布性	Chord 是纯分布式系统，节点之间完全平等并完成同样的工作。这使得 Chord 具有很高的顽健性，可以抵御 DoS 攻击
可扩展性	Chord 协议的开销随着系统规模（节点总数 n）的增加而按照 O（log（n））的比例增加。因此 Chord 可以用于大规模的系统
可用性	Chord 协议要求节点根据网络的变化动态地更新路由表，因此能够及时地恢复路由关系，使节点查询能够可靠地进行
命名的灵活性	Chord 并未限制查询内容的结构，因此应用层可以灵活地将内容映射到名字空间，而不受底层协议的限制

www.ckook.com

尽管如此，Chord 协议还是存在一定的缺陷。首先，它没有考虑孤立网络的修复，一旦出现孤立网络，可能这个孤立网络会一直存在下去；其次，如果网络中存在一组恶意攻击者，可能会导致整个网络的路由不正确；再次，由于 Chord 网络中的相邻节点是通过虚拟地址定义的，虚拟地址与物理地址之间没有必然联系，导致实际的路由过程中存在“舍近求远”的隐患。

3.3.3 Pastry

Pastry 在 2001 年由位于英国剑桥的微软研究院和莱斯（Rice）大学提出，同 Chord 一样，它也是 DHT 网络的一个变种。Pastry 的典型应用包括 PAST 和 SCRIBE。

1．路由算法

在 Pastry 协议中没有规定具体应该采用的散列算法，但是定义了散列所得的键值是一个一维空间（实际上使用的是 128bit 的整数空间）。

在 Pastry 协议中，每个节点拥有一个 128 bit 的标识（Node ID）。为了保证 Node ID 的唯一性，一般由节点的网络标识（如 IP 地址）经过散列得到。Pastry 中的每个节点拥有一个路由表（Routing Table），一个邻居节点集合（Neighborhood Set）和一个叶子节点集合（Leaf Set），它们一起构成了节点的状态表。Pastry 网络主要有以下几个重要的参数。

（1）b：用 2^b 进制来处理 128bit ID，b 的值一般取 1，2，3，4。

（2）L：叶子节点集合的容量，一般取 $2b$ 或 $2b+1$。

（3）M：邻居节点集合的容量，一般取 $2b$ 或 $2b+1$。

（4）N：代表网络中存活的节点数。

状态表的定义如表 3-9 所示。

表 3-9　　Pastry 状态表定义

表　项	定　义
路由表	共 $\log_B N$（$B=2^b$ 为系统参数，典型值为 16，N 表示系统的节点总数），每行 $B-1$ 条数据，记录了邻居节点的信息（节点标识、IP 地址、当前状态等）。第 n 行存储的节点的 Node ID 与本地节点的 Node ID 的前 $n-1$ 位相同，且第 n 位与当前节点不相同。这样形成的路由表很类似 IP 路由中最长掩码匹配的算法。参数 b（或 B）的大小非常关键，它的选择反映了存储空间和路由效率之间的折中
叶子节点集合	容量为 L，用于保存在数值上最接近本地节点的 Node ID 的节点，即逻辑地址上的相邻节点，其中一半节点标识大于本地节点的 Node ID，另一半节点标识小于本地节点的 Node ID。$\lvert L \rvert$ 的典型值为 $2b$ 或者 $2\times 2b$
邻居节点集合	保存节点的直接邻居（如 Ping 值最小的 M 个节点），即物理地址上的相邻节点，它不是用来路由消息的，而是为配置网络服务的。Pastry 并未提供物理地址相邻的获取方法，而是假设应用层可以通过某种手段（人工配置或自动协商）得到信息并配置邻居节点集合。$\lvert M \rvert$ 的典型值为 $2b$ 或者 $2\times 2b$

Pastry 节点状态表如图 3-3 所示。在节点状态表中，节点本身的 ID 为 10 233 200。叶子集合中有 8 项。路由表共有 4×8 项。邻居集合中的节点 ID 由于来源于应用层，一般没有规律性可循。首先，路由查询消息中携带的被查询键值，即对象 ID（Object ID）。当收到路由消息时，节点首先检查该键值是否落在叶子节点集合的范围内。如果是，则直接把消息转发给叶子节点集合中节点标识和消息键值最接近的节点；否则就从路由表中根据最长前缀优先的原则选择一个节点作为路由目标，转发路由消息。如果不存在这样的节点，当前节点将会从其维护的所有邻居节点集合（包括路由表叶子集合及邻居集合中的节点）中选择一个距离该键值最接近的节点作为转发目标。例如，该节点要查询键值为 10 233 102 的关键字，检查

www.ckook.com

叶子集合，该对象 ID 不在集合中，然后查询路由表，确定 Node ID 为 10233100 的节点为最接近的节点，向其发送查询请求消息。

Key ID 10233102, Node ID 10233200

GlobalRouting 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	10233-1-00	2	
0		102332-2-0	
0			

Leaf Set

Smaller		Larger	
10233133	10233121	10233203	10233213
10233110	10233103	10233221	10233230

Neighborhood Set

13233030	12233023	20233103	30233131
0023011	00233001	12333121	10233131

图 3-3 Pastry 节点状态表

Pastry 路由的伪代码如下：

```
if(L_{-|l/2|} ≤ D ≤ L_{|l/2|})
    // D 在叶子节点集合的范围内
    forward to L_i , s.th. |D − L_i| is minimal; }
else
    // 使用路由表
    Let l = shl(D, A);
    if(R_l^{D'} ≠ null)
        forward to R_l^{D'};
    else
        // 特殊情况
        forward to T ∈ L ∪ R ∪ M , s.th.;
          shl(T, D) ≥ l,
          |T − D| < |A − D|;
```

2．节点的加入和离开

新节点加入时，首先应该获得本地的节点 A 的 ID（通过应用程序决定），并且得到一个网络中已存在的节点 B 的位置信息。节点 A 的加入过程主要包括两个部分：初始化本地节点的节点状态表和通知其他节点更新节点状态表。

节点 A 向节点 B 发送一条“加入”消息，消息的关键字就是 A 的节点号。节点 B 通过 Pastry 的路由算法找到与 A 最接近的节点 ID 的节点 C。在路由的过程中，节点 B 到节点 C 的路径上的所有节点（包括 B、C 两个节点）都会把自己的节点状态表传给 A，A 通过这些信息初始化自己的节点状态表。初始化完成后，A 将通知其他节点自己已经加入网络。

按照整个加入过程中的消息数量来衡量，节点加入操作的复杂度为 $O(\log_{2^b}(N))$。

Pastry 采用时间戳机制来处理节点并发加入系统的情况。当节点 X 向节点 A 发送自己的节点状态表信息时，它会在消息中附加一个时间戳 T_X。当节点 A 完成自己的节点状态表的初始化后，它在向 X 发出的通知消息中会附加上时间戳 T_X。节点 X 就可以通过时间戳来检查在 T_X 之后的这段时间内自己的节点状态表是否发生变化。如果发生了变化，则重新向 A 发出节点状态表信息，并通知 A 重新进行初始化。这种机制的前提条件是并发加入节点的数量很小。

www.ckook.com

如果同时有大量的节点加入系统，这种策略的性能如何有待进一步研究。

在 Pastry 中，当相邻节点不能和某个 Pastry 的节点通信时，就认为该节点已退出。

叶子节点集合中节点退出的处理机制为：本地节点要求当前叶子节点集合 L 中的 ID 最大的节点或 ID 最小的节点把它的叶子节点集合 L'发送过来（根据失效节点的节点 ID 判断，如果失效节点的节点 ID 比本地节点 ID 大，则用 ID 最大的节点，反之则用 ID 最小的节点）。如果 L'中存在 L 中没有的节点，则从中选择一个替代失效节点（在替代之前，需要验证该节点是否仍然有效）。

路由表中节点退出的处理机制为：本地节点选择退出节点所在的路由表行中的任意一个节点，要求它把自己的路由表中的对应的节点信息发送过来。如果本地节点的该路由表行中已经没有可用节点，则从路由表的下一行中选择任意一个节点。整个过程持续直到本地节点找到一个替代退出节点的节点，或者本地节点遍历路由表为止。

3．算法分析

通过对 Pastry 路由算法的分析，可以得出该协议具有以下几方面的优点：

（1）Pastry 利用了成熟的最大掩码匹配算法，在实现时可以利用很多现成的软件算法和硬件框架，可以获得很好的效率；

（2）与 Chord 相比，Pastry 引入了叶子节点和邻居节点集合，当应用层能够及时准确地获得这两个集合的节点信息时，路由查找速度可以大大提高，并且降低了因路由引起的网络传输开销。

但是在动态变化的 P2P 网络中，如何准确地获得叶子节点集合和邻居节点集合的信息有很大的难度。

3.3.4 CAN

内容寻址网络（CAN，Content-Addressable Network）在 2001 年由加州大学伯克利分校提出。CAN 也是 DHT 的一个变种。

1．路由算法

在散列算法上，CAN 与 Chord、Pastry 有所不同。在 Chord 和 Pastry 中，采用的是一致性散列算法，散列后得到的键值总是一维的。而在 CAN 中，散列后的结果是一个 d 维的笛卡尔空间。d 是一个由系统规模决定的常量。

在 CAN 中，整个 P2P 网络被映射到一个 d 维的笛卡尔空间，每个节点的位置由其自身 ID（经由散列后得到的 d 维向量）决定。CAN 中的节点只存储相邻节点表（在笛卡尔空间中相邻的定义为：在 d 维笛卡尔空间中，2 个节点的 d 维坐标中有 d–1 维是相等的，剩余的一维是相邻的，这样的节点称之为是相邻的）。由于 d 维的空间中最多有 $2d$ 个相邻的节点，因此 CAN 中节点的相邻节点表最多只存储 $2d$ 个表项，与整个 P2P 网络的节点数无关。

在查询的过程中，查询节点首先计算被查询内容的键值（也是一个 d 维向量），然后在相邻节点表中查找与该键值最为接近的相邻节点，找到后向该节点发送查询请求（这一策略被称为贪婪策略）。收到查询请求的节点如果发现被查询的信息存储在本地，则直接回应查询节点（这与 Chord、Pastry 是一致的）；如果发现被查询的信息不在本地，则根据相邻节点表转发请求。这样的过程一直持续到找到相应的节点为止。如图 3-4 所示，该 CAN 是映射在一个 2 维的笛卡尔空间上，节点 1 的坐标区间是（0.75～1，0.5～0.75），它查找的内容的键值为（0.45，0.15），它的相邻节点表的表项分别是（0.75～1，0.25～0.5）、

www.ckook.com

（0.75～1，0.75～1）、（0.5～0.75，0.5～0.75），经过 CAN 的路由算法后，最终找到节点 7，坐标区间是（0.25～0.5，0～0.25）。

如果查询节点或转发节点发现相邻节点表中无法找到可用的下一跳节点，则采用非结构化 P2P 网络中常用的扩展环搜索（Expanding Ring Search，使用无状态、受控的洪泛算法在单播的 CAN 中搜索）以找到合适的（符合贪婪策略）下一节点。

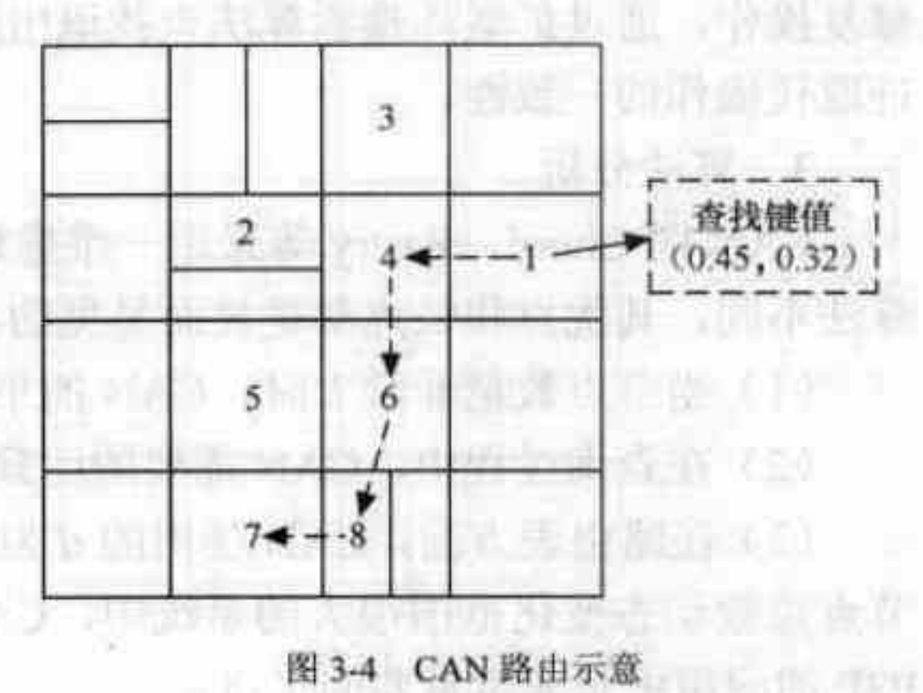


图 3-4 CAN 路由示意

CAN 路由算法的伪代码如下：

```
// 判断 D 是否存储在本地节点
if (D ∈ local_zone)
    return local_addr;
else
    // 判断相邻节点表中是否有与 D 更接近的节点
    if (find(next_zone))
        forward next_addr;
    else
        forward ERS_addr;
```

在 CAN 中，查询需要的跳数是均值为（$d/4$）（$n^{1/d}$）的随机值，考虑到 d 为常数，这一值可以表示为 O（$n^{1/d}$）或 O（$dn^{1/d}$）。

2．节点的加入和离开

CAN 是一种动态网络，一个新的节点加入 CAN 时，必须获得自己的坐标空间。这个过程是通过现有节点分裂自己的区域实现的。CAN 中的节点加入分为以下 3 步：

（1）新节点必须找到一个已经存在于 CAN 中的节点；

（2）新节点通过 CAN 的路由算法确定一个将要分裂自己区域的节点；

（3）执行分裂操作，通知原有区域的邻接区域的节点新的路由信息。

当节点离开 CAN 时，必须保证它的区域被 CAN 系统收回，也就是分配给其他仍然在系统中的节点。一般过程是由某个节点来接管这个区域和所有的（关键字，值）数据库。如果这个区域可以和相邻区域合并形成一个大的区域，那么 CAN 将执行合并操作。如果合并不能进行，那么该区域将交给其邻接节点中区域最小的节点。也就是说，这个节点将临时负责两个区域。

在 CAN 中，相邻节点周期性地交换更新消息。如果某个节点多次没有发送更新消息，则相邻节点认为该节点退出，执行取代操作。取代操作的具体步骤如下：

（1）启动一个时钟；

（2）当时钟超时后，节点向退出节点的所有相邻节点发送取代消息，消息中包括自己的区域面积信息；

（3）节点收到取代消息时，对比自己和取代节点的区域面积大小，面积最小的节点成为最终的取代节点。

在特殊情况下，可能出现多个相邻节点同时退出的情况。这时如果让某个节点取代某个失效节点，就有可能导致 CAN 中节点的路由信息不一致。在这种情况下，CAN 将首先执行

www.ckook.com

修复操作，通过扩展环搜索算法查找退出区域以外的区域的节点以获得足够的邻接状态来保证取代操作的一致性。

3．算法分析

CAN 同 Chord、Pastry 等采用一维虚地址空间的 P2P 网络相比，最主要的区别在于路由算法不同，其优点和缺点都是显而易见的。

（1）当节点数量非常大时，CAN 的平均查询跳数要比 Chord、Pastry 增加得更快。

（2）在查询过程中，CAN 需要的运算量也要高于 Chord、Pastry。

（3）在路由表方面，CAN 使用的 d 为预先设置的常量，与系统的节点数量无关。即使在节点总数动态变化范围很大的系统中，CAN 中节点的相邻节点表结构也能保持稳定，这在 P2P 的应用中是非常重要的优点。

3.3.5 Tapestry

Tapestry 在 2001 年由加州柏克莱大学提出，它的思想来源于 Plaxton，是利用节点 ID 之间前缀匹配的结果所形成的网状网络。Tapestry 的典型应用是 OceanStore。

1．路由算法

Tapestry 规定的散列算法与 Chord 和 Pastry 类似，可以采用 SHA-1，对象和节点通过散列算法被映射到一维的散列空间上。

Tapestry 具有自治、容错和负载平衡，对于负载具有弹性等特性。在 Tapestry 中，每个节点维护一个多层次的邻居映射（Neighbor Map），每一层的节点的数量为节点 ID 使用的进制数（称为 Base），在第 j 层的第 i 个条目存储的是与该节点 ID 拥有相同的后缀 $suffix(N, j\text{-}1)$，同时第 j 位为 i 的节点 ID 及其定位信息。例如，节点 3564（八进制）的邻居映射中的第 4 层第 7 个条目所指向的节点，是在网络距离上靠近 3564，并且节点 ID 以 7564 结尾的节点。Tapestry 的邻居映射如图 3-5 所示。

Tapestry 的路由过程与 Pastry 非常类似，同样采用的是多级匹配的算法，路由深度为 $O(\log_b N)$（其中 b 为节点 ID 使用的进制数）。如图 3-6 所示，节点 475BD 要寻找键值为 3AD29 的对象，通过 5 级路由，最后确定节点 3AD29。

节点 3564 邻居映射

0564	*064	**04	***0
1564	*164	**14	***1
2564	*264	**24	***2
3564	*364	**34	***3
4564	*464	**44	***4
5564	*564	**54	***5
6564	*664	**64	***6
7564	*764	**74	***7

图 3-5　Tapestry 邻居映射

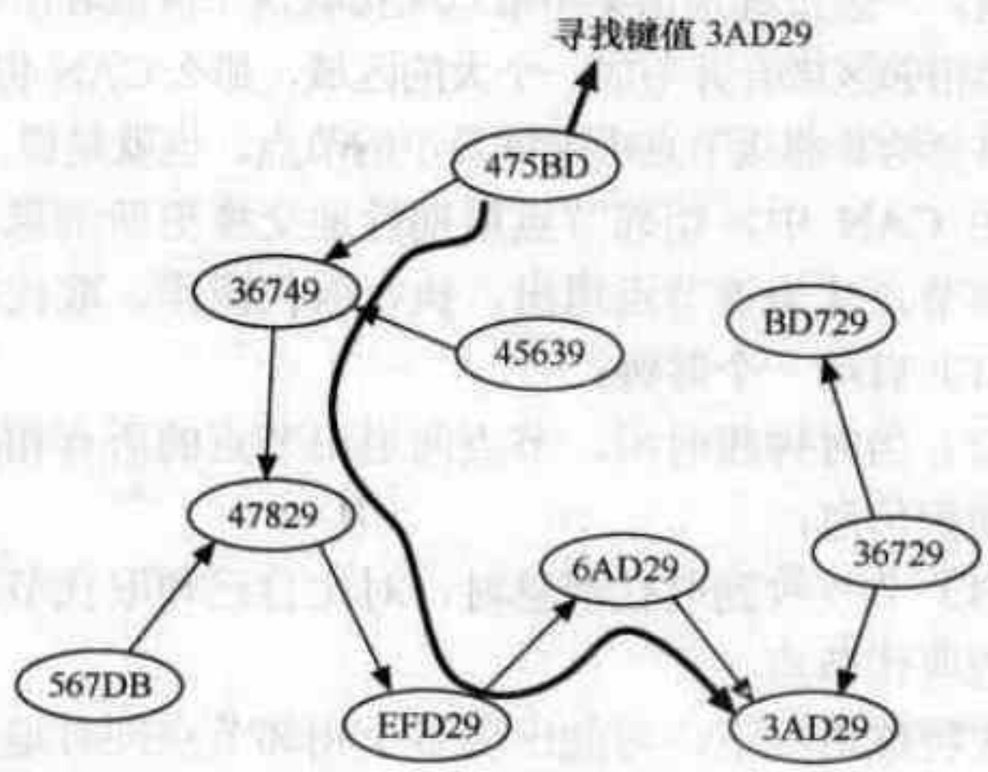


图 3-6　Tapestry 路由示意

www.ckook.com

2．节点的加入和离开

Tapestry 的节点加入算法和 Pastry 很类似。节点 *A* 在加入 Tapestry 网络之前，也需要确定一个已经存在网络中的节点 *B*。A 通过 B 发出路由请求，寻找与自己节点 ID 最接近的节点 *C*。节点 *A* 根据路由过程中所经过的节点的邻居映射来构造自己的邻居映射。构造过程中还需要进行一些优化工作。在构造完自己的邻居映射后，节点 *A* 通知网络中的其他节点更新自己的邻居映射。该通知消息只针对存在于 A 的邻居映射中的主邻居节点和二级邻居节点进行。Tapestry 的节点加入的伪代码如下所示。

```
H = G;
For (i = 0; H != NULL; i++)
    Grab ith level NeighborMap_i from H;
    For (j = 0; j < baseofID; j++)
        // 填充邻居映射的第 i 级节点
        While (Dist(N, NM_i(j, neigh) > min(eachDist(N, NM_i(j, sec.neigh))))
            neigh = sec.neighbor;
            sec.neighbors = neigh->sec.neighbors(i, j);
    H = LookupNextHopinNM(i + 1, new_id);
// 当访问到 NULL 节点时结束
```

对于节点的离开，Tapestry 采用以下两种机制处理。

（1）当节点正常地从网络中退出时，节点通过后向指针通知所有把它作为邻居的节点，这些节点会相应地调整邻居映射并通知对象服务器该节点已经退出网络；

（2）当节点非正常地从网络中退出时（主要原因是节点失效），它的邻居节点可以检测到它已经退出网络并可以相应地调整邻居映射。

在 Tapestry 中，检测链接和服务是否失败通常依赖于 TCP 超时设定。另外，每个 Tapestry 节点使用 backpointers 来发送周期的心跳（heartbeats）消息的 UDP 消息给邻居节点。通过这个消息来表明消息的源是一个有效节点。通过检测心跳消息的源，节点可以很快地检测到邻居映射中的失效节点。为了保证系统的顽健性，邻居映射每个条目之后还维护了两个后备邻居，当主邻居失效时，节点使用后备邻居进行路由。Tapestry 为失效节点提供了一个“二次机会”恢复，当在二次机会的时间内仍然失效的节点，将被后备节点替代。另外，为了避免查询时的单点失败，Tapestry 还采用了同一键值多个映射的方式，用冗余度换取可靠性。

3．算法分析

Tapestry 与 Chord 相比，在存储容量和最大路由跳数上基本一致，但是 Chord 对于网络最差情况的表现要差一些，因为它的逻辑地址与物理地址没有关联性，可能造成物理地址很近的节点在逻辑地址上相差很远。

Tapestry 在路由算法上和 Pastry 非常相似，它们之间的主要区别表现为共享对象发布上的不同。在 Pastry 中，节点发布共享信息是直接将对象的内容复制到与对象 ID 最接近的节点上，不受对象发布者的控制；在 Tapestry 中，节点发布共享信息是将 <对象，节点> 对信息发送到与对象 ID 最接近的节点上。因此，在 Pastry 中，定位的时间更快，而在 Tapestry 中，对象的保密性和一致性更好。

Tapestry 与 CAN 之间的区别主要表现为以下几个方面。

（1）Tapestry 的网络结构和每个节点的高输出导致它从不同节点到同一目标的转换都非

www.ckook.com

常快，而在 CAN 中查找本地节点比查找缓存中的位置快很多。

（2）由于 Tapestry 采取了 <对象，节点> 对的形式发布共享对象，使它可以迅速地完成对象的改变，而把对象的一致性问题交给应用层处理，而 CAN 假设对象是不变的，对象的值一旦改变就必须重新插入。

（3）CAN 同时也存在和 Chord 一样的问题，逻辑地址和物理地址不匹配。

（4）CAN 在路由信息存储上比 Tapestry 有优势，它更适合于动态变化频繁的网络。

3.3.6 Kademlia

Kademlia 在 2002 年由美国纽约大学的 PetarP. Maymounkov 和 David Mazieres.提出。它也是一种分布式散列表（DHT）技术，以异或算法（XOR）为距离度量基础。Kademlia 在 BitTorrent、BitComet 和 eMule 等软件中得到了应用。

1．路由算法

在 Kademlia 算法中，没有明确地描述产生节点 ID 和关键字键值的方法，只是假设每个节点能够随机地产生一个 160bit 的节点 ID。

在 Kademlia 算法中，定义了一个概念，那就是两标识符之间的距离的概念。给定两个标识符，x 和 y，Kademlia 定义两者的位异或（XOR）的结果作为两者的距离，$d(x,y)=x\oplus y$。我们注意到异或运算是一个有意义的运算，虽然它不是欧几里德运算。它具有下面的性质：

（1）$d(x,x)=0$；

（2）如果 $x\neq y$，则 $d(x,y)>0$；

（3）对于 $\forall x,y,d(x,y)=d(y,x)$；

（4）$d(x,y)+d(y,z)\geqslant d(x,z)$。

其中（4）之所以成立，是基于下面这个事实：$d(x,z)=d(x,y)\oplus d(y,z)$，并且 $\forall a\geqslant 0,b\geqslant 0,a+b\geqslant a\oplus b$。与 Chord 的顺时针循环运算类似，异或运算也是单向的。对于给定的一个点 x 以及距离 Δ，仅有一个点 y，使得 $d(x,y)=\Delta$。单向性确保所有对于相同的键值的查询将汇聚到相同路径中来，而不管是什么起源节点。因此，在查找路径上缓存 <Key, Value>对可以减少“撞车”的机会。

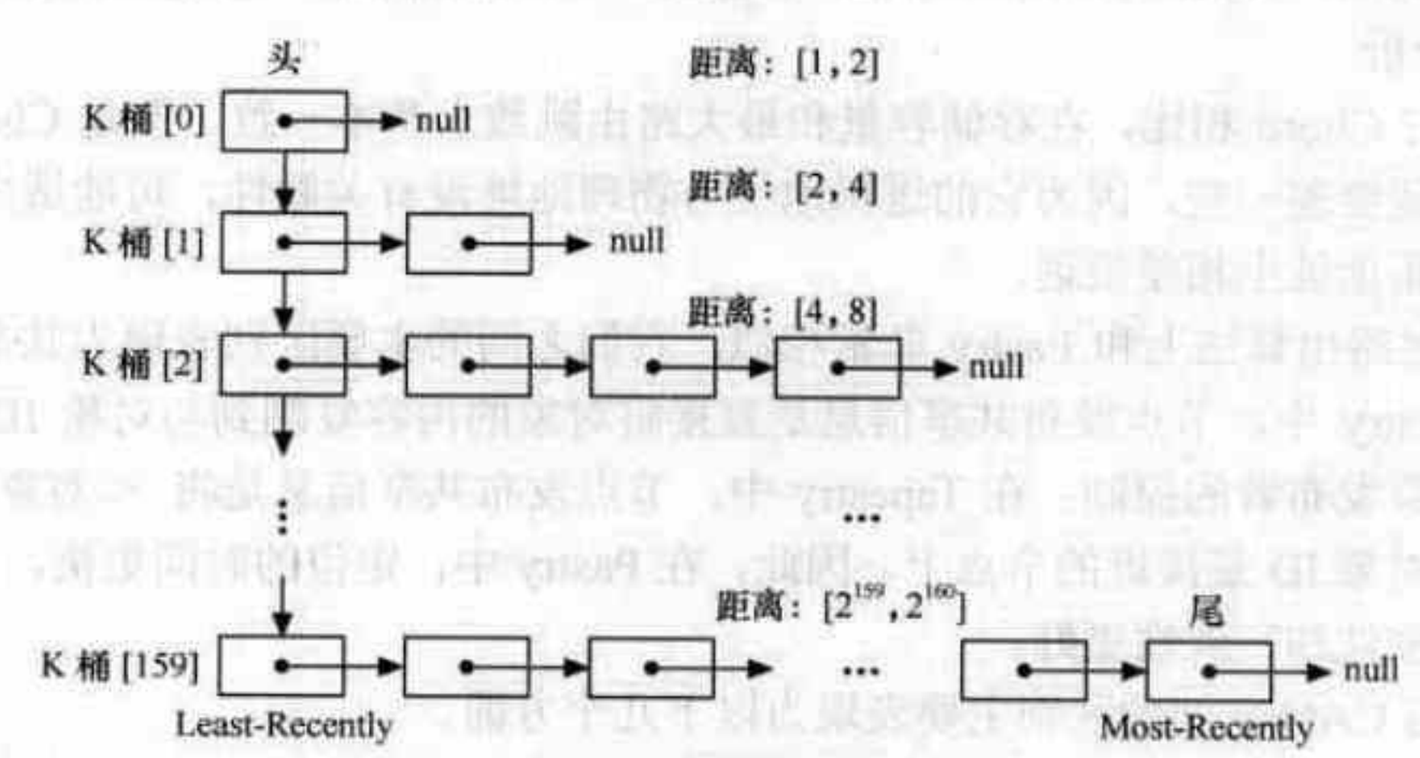


图 3-7 “K 桶”示意图

www.ckook.com

Kad 的路由表是通过“K 桶”的表格构造起来的，如图 3-7 所示。$\forall 0 \leqslant i < 160$，每个节点保存 k 个到本节点的距离为 2^i 到 2^{i+1} 之间的节点信息列表，包括<IP 地址，UDP 端口， 节点 ID>，我们把这些列表称为“K 桶”。“K 桶”的更新策略是：当节点收到来自其他节点的任何消息时，它将更新自己的一个“K 桶”，即发送节点 ID 对应的那个桶。如果发送节点已经存在于接收者的“K 桶”中，接收者会把它移到列表的尾部。如果这个节点还没有存在于对应的“K 桶”中并且这个桶少于 k 个节点，则接收者把发送者插入到列表的尾部。如果对应的“K 桶”已经满了，则发送者将向该“K 桶”中的最久未联系节点发送 Ping 命令测试是否存在，如果最久未联系节点没有回复，则把它从列表中移除，并把新的发送者插入到列表尾部。如果它回复了，则新的发送者信息会丢弃掉。“K 桶”的更新机制非常高效地实现了一种把最近看到的节点更新的策略，即在线时间长的节点具有较高的可能性继续保留在“K 桶”列表中。采用这种机制是基于对 Gnutella 网络上大量用户行为习惯的研究结果，即节点的有效概率和在线时长成正比关系，如图 3-8 所示，横坐标为在线时长（单位：min），纵坐标为有效概率。

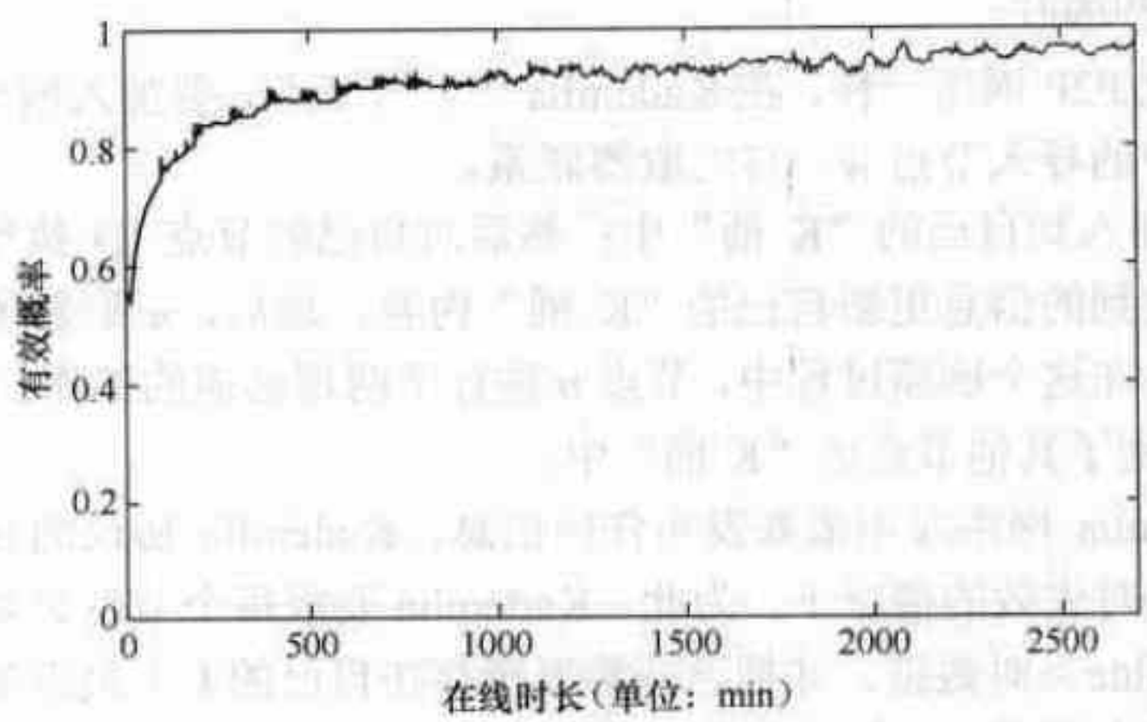


图 3-8 节点行为习惯分析

Kademlia 协议由 4 个远程过程调用（RPC，Remote Procedure Call）组成：PING、STORE、FIND_NODE、FIND_VALUE，如表 3-10 所示。

表 3-10 Kademlia 协议中的 RPC

RPC	功能描述
PING	测试节点是否存在
STORE	指示一个节点存储一个<Key, Value>对以用于以后的检索
FIND_NODE	将 160 bit 的 ID 作为变量，RPC 的接收者将返回 k 个它所知道的最接近目标 ID 的节点信息
FIND_VALUE	同 FIND_NODE 行为相似，返回 k 个它所知道的最接近目标键值的节点信息。如果 RPC 接收者拥有这个键的 STORE RPC，则只需要返回这个已存储的值

为了防止伪造地址，在所有 RPC 操作中，接收者都需要响应一个随机的 160bit 的 ID 值。另外，为了确定发送者的网络地址，PING 操作还可以附带在接收者的 RPC 回复信息中。

Kademlia 参与者必须做的最重要的工作是为一个给定的节点 ID 定位 k 个最接近节点。这个过程被称为节点查询。Kademlia 使用一种递归算法来进行节点查询。例如，当节点 x 要

www.ckook.com

查找节点 y 时，按照如下递归操作步骤进行路由：

（1）计算 x 到 y 的距离：$d(x,y)=x\oplus y$；

（2）从 x 的第 $\log d$ 个“K 桶”中选择 α 个节点，发送并行的、异步的 FIND_NODE RPC，如果该桶中的节点少于 α 个，则从附近多个桶中选择距离最接近 d 的 α 个节点；

（3）接收到查询请求的节点，如果发现自己就是 y，则回答自己是最接近 y 的，否则测量自己和 y 的距离，并且从其对应的“K 桶”中选择 k 个节点的信息发送给 x；

（4）x 从新接收到的 k 个节点中再次选择 α 个还没有询问过的节点发送 FIND_NODE RPC，如果经过一轮的 FIND_NODE 都没有返回一个比已知最接近的节点更接近的节点，则 x 将重新向所有 k 个未曾询问的最接近节点发送 FIND_NODE；

（5）直到 x 已经询问了 k 个最接近节点并且得到了响应，这个查询才结束。

这里 α 也是为系统优化而设立的一个参数，在 BitTorrent 实现中，取值为 $\alpha=3$。当 $\alpha=1$ 时，查询算法在消息开支和检测失效节点的时延上与 Chord 非常相似。然而，Kademlia 可以通过灵活地选择 k 个节点中的任意一个节点进行查询来保证低时延路由。

2．节点的加入和离开

同其他的结构化 P2P 网络一样，在 Kademlia 中，当节点 u 要加入网络时，它必须确定一个已经存在于网络中的导入节点 w，与之取得联系。

首先，u 将 w 插入到自己的“K 桶”中；然后对自己的节点 ID 执行一次 FIND_NODE 操作，并且根据接收到的信息更新自己的“K 桶”内容；最后，u 刷新所有比最接近的邻居节点更远的“K 桶”。在这个刷新过程中，节点 u 进行了两项必需的工作：既填充了自己的“K 桶”，又把自己插入到了其他节点的“K 桶”中。

节点离开 Kademlia 网络时不需要发布任何信息，Kademlia 协议的目标之一就是能够弹性工作在任意节点随时失效的情况下。为此，Kademlia 要求每个节点必须周期性地发布全部自己存放的<Key,Value>对数据，并把这些数据缓存在自己的 k 个最近邻居节点上，这样存放在失效节点的数据会很快被更新到其他新节点上。

3．算法分析

Kademlia 中基于异或拓扑的路由算法与 Pastry、Tapestry 的路由算法中的第一步和 Plaxton 的分布式搜索算法都非常的相似。然而，在后面这 3 个算法中，当路由到达路由表中最接近目标节点的级数后，再从这一级的路由表的 2^b 个节点中一次性地选择目标节点时，都会产生问题。因为没有采用 Kademlia 中的异或拓扑路由算法，这 3 个算法都还需要一个额外的算法结构，以从与目标节点拥有相同的前缀，但是接下来的 $b$bit 的数字不同的这 2^b 个节点中定位目标节点。这 3 个算法在解决这个问题上采取了不同的方法，每个都有其不足之处。它们在大小为 $O\left(2^b\log_{2^b}n\right)$ 的主路由表之外都另外需要一个大小为 $O\left(\log 2^b\right)$ 的次级路由表，这增加了自举和维护的开支，使协议变得更加复杂，而且对于 Pastry 和 Tapestry 来说，这妨碍了对正确性与一致性的分析。Plaxton 虽然有证据可以分析，但在类似于 P2P 网络这样极易失效的环境中不太适应。

相反，Kademlia 则非常容易地以不是 2 的基数被优化。通过配置桶表以使每一跳以 $b$bit 的速度接近目标节点。这就要求其满足一个条件，那就是任意的 $0<j<2^b$ 和 $0\leqslant i<160/b$，在与节点的距离为 $[j2^{160-(i+1)b},(j+1)2^{160-(i+1)b}]$ 的范围内就要有一个桶，这个有实际的项的总量预计不会超过 $\left(2^b-1\right)\log_{2^b}n$ 个桶。

www.ckook.com

此外，Kademlia 算法还具备一些独有的特点。它减少了节点必须发送的用来相互认识的配置消息的数量。在做键查询的同时，配置消息将会被自动传播。节点拥有足够的知识和灵活性来通过低时延路径发送查询请求。Kademlia 使用并行的、异步的查询请求来避免节点失效所带来的超时时延。通过节点记录相互的存在的算法可以抵抗某些基本的拒绝服务（DoS）攻击。

3.3.7 小结

以上介绍了 5 类基于 P2P 结构化的 P2P 分布式网络。通过这 5 类网络的对比可以看出，它们具有很多相似的特点，譬如都采用了虚拟地址的形式标识网络节点和数据对象，在处理节点的加入和退出时都采取了相似的步骤等。但是在路由机制方面，各类网络采用了不同的算法，从而导致了不同的网络特性。基于这些系统，研究人员还设计了具有更好性能的改进路由机制。这 5 类 P2P 分布式网络的性能对比如表 3-11 所示。

表 3-11　　P2P 结构化分布式网络性能对比

网络类型	节点加入复杂度	空间复杂度	查找复杂度	是否支持负载平衡
Chord	$O(\log^2(N))$	$O(N\log(N))$	$O(\log(N))$	支持
Pastry	$O(\log^2(N))$	$O(N\log(N))$	$O(\log(N))$	支持
CAN	$O(d)$	$O(d)$	$O(dN^{1/d})$	支持
Tapestry	$O(\log^2(N))$	$O(N\log(N))$	$O(\log(N))$	支持
Kademlia	$O(\log^2(N))$	$O(N\log(N))$	$O(\log(N))$	支持

当然，基于分布式散列表的路由机制也存在无法解决的问题，主要表现为以下两点。

（1）经过散列算法之后，节点的位置信息被破坏了，来自同一个子网的节点很可能节点 ID 相距甚远，这造成了虚拟地址和物理地址的不匹配，从而不利于查询性能的优化。

（2）基于散列表的系统不能利用上层应用本身的信息，许多应用（比如文件系统）的数据本身是按照层次结构组织的，而在使用散列函数后，这些层次信息就丢失了，另外使用散列算法也无法很好地与内容语义进行匹配。

3.4 非结构化 P2P 网络的搜索技术

非结构化的 P2P 网络指的是以 Gnutella 为典型代表的一类网络。在这类网络中，节点没有指定的逻辑地址，采用随机方法或者启发策略加入网络，网络拓扑随着节点的变迁和网络通信的进行而发生演变。

非结构化的 P2P 网络的搜索技术分为两类：不利用任何文档分布信息的盲搜索和利用网络中文档分布信息的搜索。前者可以抽象为如何从一个随机图中的任意一个节点出发定位目标点，使得整个过程遍历的点的个数最少；后者的特点是网络中的各个节点记录以前接收过的请求和应答，并将这种相关性作为对后续路由的启发。

非结构化的 P2P 网络为了保证较高的文本的召回率（Recall-Rate），路由时需要覆盖较多的节点，花费较大的通信代价和主机计算代价，特别是在对稀疏资源进行定位时。因此，发

www.ckook.com

现的准确性和网络的可扩展性是非结构化的 P2P 网络面临的两个重要问题。

3.4.1 Flooding

Flooding 算法是最早出现的非结构化 P2P 网络的路由算法，它的特点是路由时进行全网遍历的盲搜索，网络的拓扑没有固定的结构。

1．路由算法

在这种路由技术中，消息像洪水一样在网络中的各个节点间流动，所以被称作 Flooding 路由。从算法特性上看，它实际上是一个宽度优先搜索（BFS），算法首先遍历自己的相邻节点，然后再向下遍历。

Flooding 路由技术的典型网络代表是 Gnutella，它的具体路由方法如下。

在网络中，节点要查找某个文件时，首先向它的相邻节点传递查询信息。如果某个相邻节点拥有这个资源，就向查询节点返回一个 QueryHit 消息，否则，相邻节点就把消息转发给自己的相邻节点。在整个路由过程中，节点通过消息的 *TTL* 值来判断消息是否过期。

如图 3-9 所示，节点的查询消息最初设置 *TTL*=4，每传播一次后 *TTL* 减 1。如果当 *TTL* 减到 0 时还没有搜索到资源，则节点停止传递消息；如果搜索到资源，则按照原路径将目标机器的信息返回给查询节点。在消息传播的过程中可能出现循环，但是由于有 *TTL* 控制，因此这个循环不会永远进行下去，当 *TTL*=0 的时候自然结束。

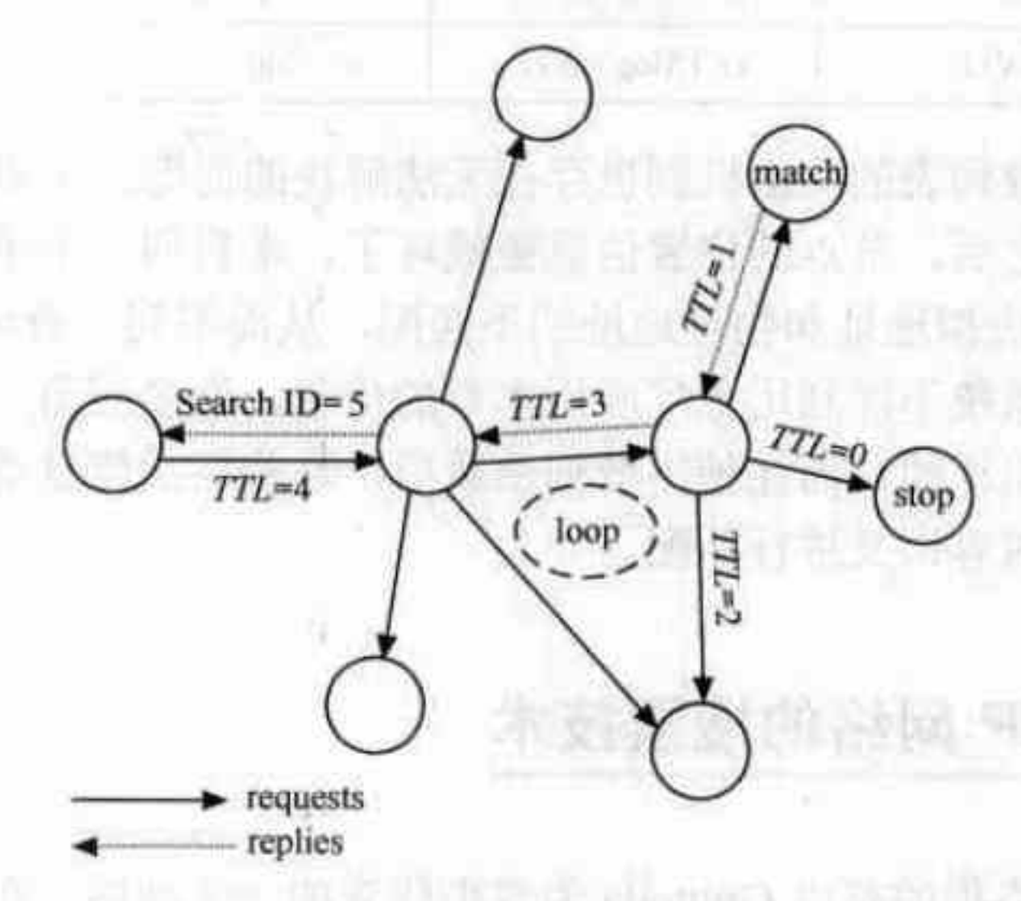


图 3-9 Flooding 路由算法

2．算法分析

Flooding 路由算法的优点和缺点都是非常明显的，主要表现在以下几个方面。

（1）路由算法比较简单，易于实现；

（2）路由算法和备份算法可以分离实施，两者之间没有必然联系，易于实现；

（3）每次路由都是在进行全网式的遍历，增加了网络的负担，搜索的效率不高；

（4）整个网络的可扩展性差；

（5）路由算法容易被攻击。

www.ckook.com

3.4.2 Modified-BFS

Modified-BFS 是 Flooding 路由算法的一种改进算法，它的改进点主要针对的是控制节点路由消息的扩散范围。

1．路由算法

这种路由算法的大部分路由机制都是与 Flooding 算法相同的，也是采用了全网遍历的盲搜索形式进行消息的路由。唯一的区别在于，在 Modified-BFS 中，节点在路由消息时不再向它的所有的相邻节点发送消息，而是按照一定的概率随机地选择部分的节点发送消息（概率根据系统的设计确定）。

2．算法分析

Modified-BFS 算法的特性同 Flooding 算法非常相似，区别在于以下两点。

（1）减少了网络中的路由消息，降低了网络的负载，提高了路由的效率；

（2）由于对整个网络的覆盖降低了，所以可能需要花费更长的时间才能定位到目标节点。

3.4.3 Iterative Deepening

Iterative Deepening 也是对 Flooding 算法的一种改进，与 Modified-BFS 不同的是，它的改进点主要针对的是控制节点路由消息的扩散程度。

1．路由算法

这种路由算法也是对 Flooding 路由的一种改进，其主要的特点是使用了变化的 *TTL* 值控制路由的深度。在策略上，Iterative Deepening 定义了两个系数 *P* 和 *W*。*P* 代表 *TTL* 的变化区间，例如 $P=\{a, b, c, d\}$；*W* 代表每次迭代的时间间隔。具体的路由方法如下。

首先，查询节点采用宽度优先搜索准则向相邻节点发送查询消息，消息的 $TTL=a$。消息传递到 $TTL=0$ 时停止，并且此时的边界节点临时缓存查询消息；查询节点如果在时间 *W* 内收到了匹配的响应，则停止查询，否则，在 *W* 时刻向相邻节点发送 Resend 消息，消息的 $TTL=a$；非边界节点在接收到 Resend 消息时，只是简单的转发消息，边界节点接收到 Resend 消息时，将开始缓存的查询消息的 *TTL* 设置为 $b-a$，继续向它的相邻节点发送；同理，在下一个 *W* 时刻到达时，如果查询节点没有收到匹配的响应，则发送 Resend 消息，消息的 $TTL=b$，再下一个 *W* 时刻发送的 Resend 消息的 $TTL=c$。

Iterative Deepening 的路由如图 3-10 所示。

2．算法分析

Iterative Deepening 路由算法改进了 Flooding 路由方法，使 *TTL* 没有在一开始就被赋予一个较大的值，减少了搜索的半径。这种路由算法的特点主要表现在：

（1）在网络中重复资源丰富时，它可以在不影响搜索质量的前提下减少网络的查询流量；

（2）在最坏情况时，它的搜索延迟将很大，因为 *TTL* 的逐步递增导致了开销的增加。

www.ckook.com

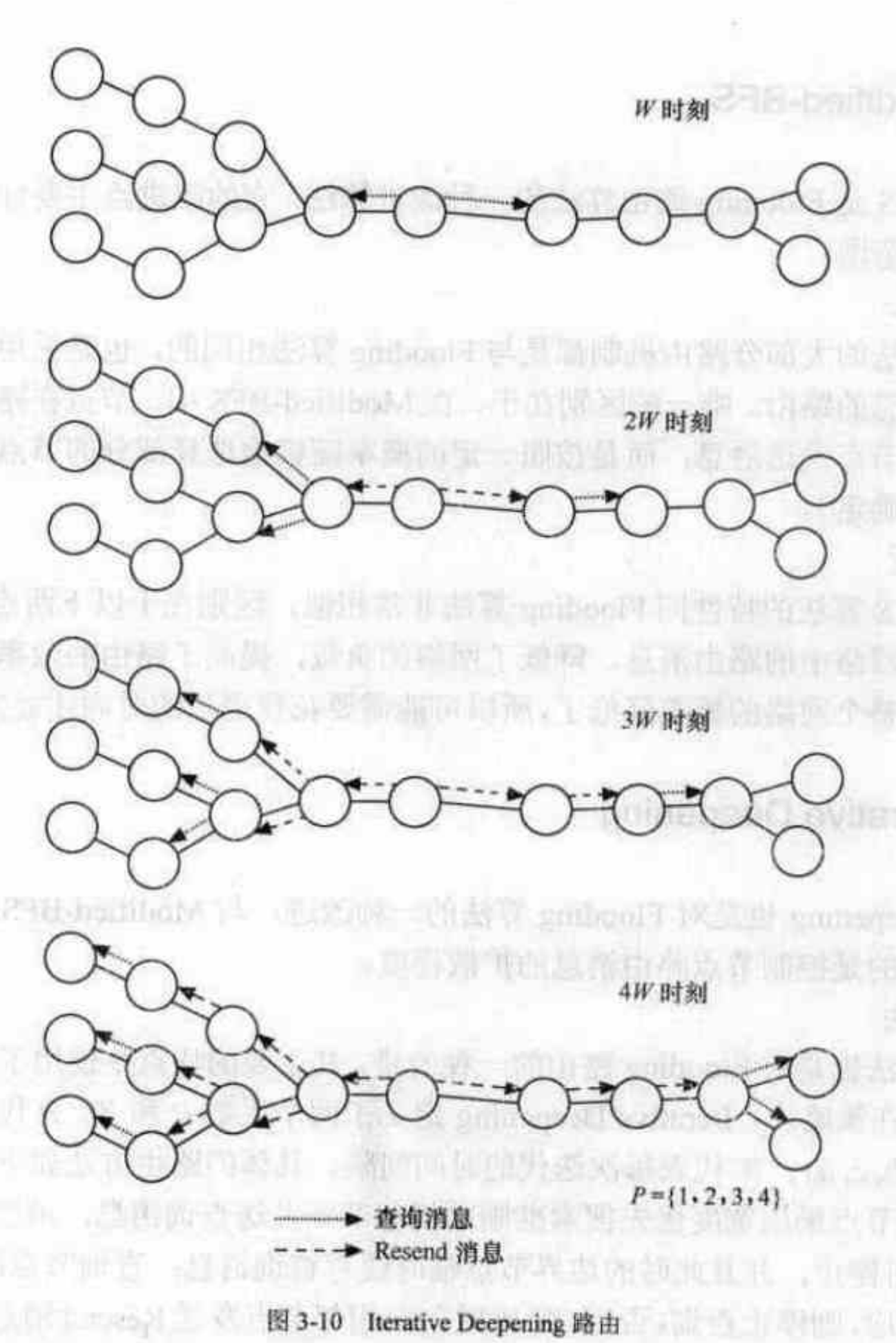


图 3-10 Iterative Deepening 路由

3.4.4 Random Walk

Random Walk 也是对 Flooding 算法的一种改进，同前面两种改进算法相比，这种算法对节点路由消息的控制更加明显，在路由消息的扩散范围和扩散程度两个方面都对算法进行了改进。

1．路由算法

这种路由算法的具体的路由方法如下：

查询节点将随机挑选 n 个相邻节点发送查询请求，中间节点每次随机选择自己的一个相邻节点转发消息，查询请求在路由过程中与请求节点保持联系。查询请求也被称为“walker”。

查询请求在以下 3 种情况下停止继续路由：

（1）成功找到查询的内容；

（2）当消息的 TTL=0 时；

（3）定期发送 checking 消息给查询节点，询问是否终止查询。

www.ckook.com

在 Random Walk 路由算法中，两个参数的取值对算法性能有很大的影响，分别是 walker 的数量 n 和发送 checking 消息的步长 S：

（1）如果 n 取值太大，将增大网络的负载，如果 n 取值太小，将增加查询的延时，一般 n 的取值为 16～64；

（2）如果 S 取值太大，将降低 checking 的作用，增大网络的负载，如果 S 取值太小，将造成查询节点出现阻塞，试验表明当 S=4 时，网络达到一个很好的均衡点。

Random Walk 的效果图如图 3-11 所示。

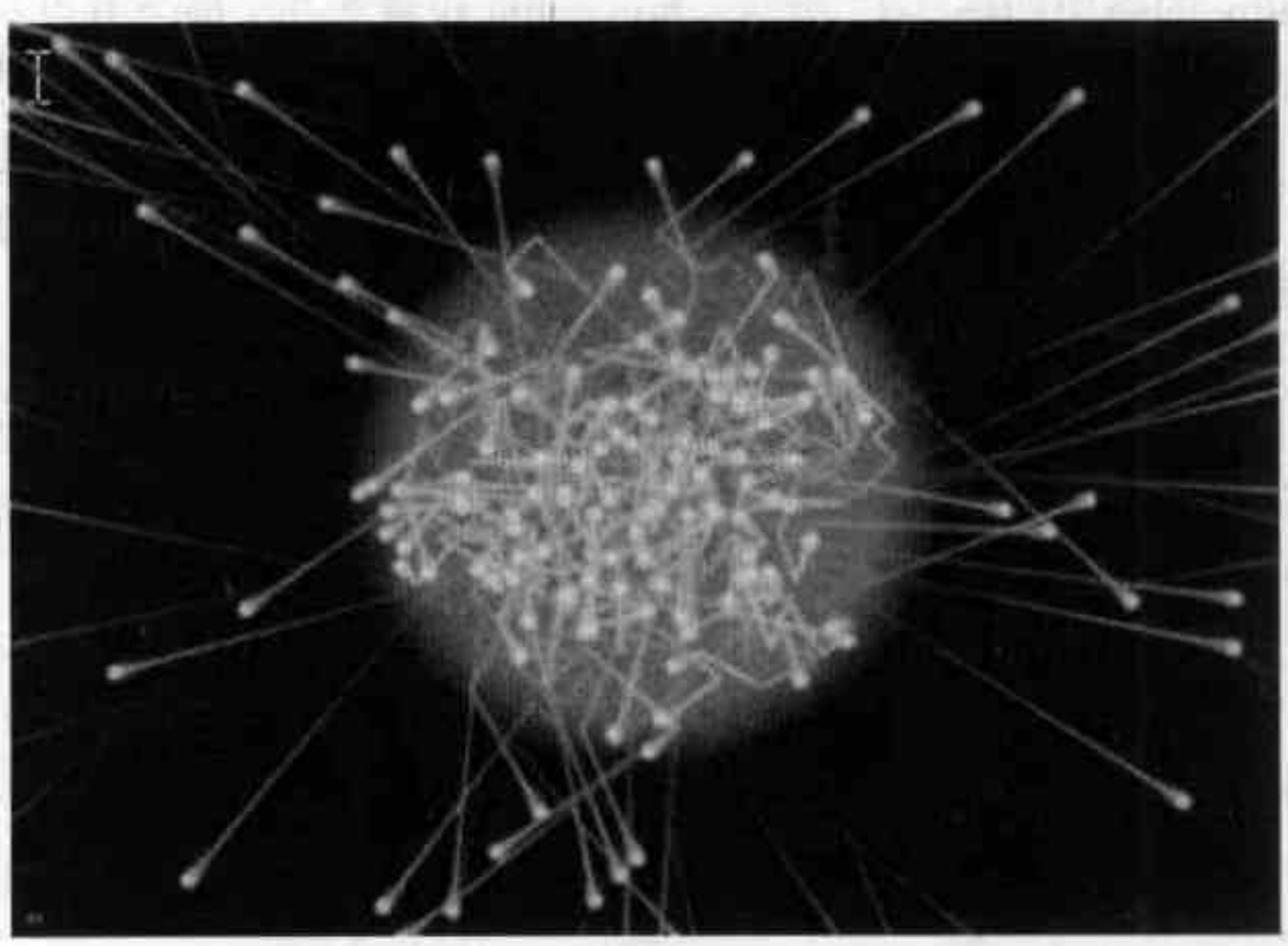


图 3-11 Random Walk 效果图

2．算法分析

Random Walk 算法的最大优势在于查询过程中路由消息的减少。即使是在最坏情况下，一次查询网络中也只会产生 $n \times TTL$ 条路由消息，这与所依赖的网络结构无关。另外，它也从一定意义上实现了本地节点的负载均衡，因为没有哪个节点在消息的转发过程中是热门节点。

Random Walk 算法的最大劣势在于表现的不稳定性上。查询的成功率在很大程度上取决于网络的拓扑结构和随机性的选择上，与查询节点和目标节点间的距离的相关性反而很小。这种算法的另一个缺陷在于它对查询对象不能动态地适应。它无法从上一次的查询中获得任何信息，对于热门对象或冷门对象都是同等对待。

3.4.5 Query Routing

Query Routing 也可以看作是对 Flooding 算法的一种改进，它改进的地方主要是体现在它对路由的节点进行了控制，不再是随机地向全部节点或者部分节点进行路由，而是有针对性地向关联节点进行路由。

1．路由算法

这种路由算法是一种启发式搜索算法。首先每个 Peer 都给本节点的资源做索引，并且记

www.ckook.com

录相邻节点的资源信息。当查询到达的时候，节点首先查询本节点中的资源情况，然后查询相邻节点是否也存在与该查询匹配的节点，如果存在，则向该节点发送消息，如果不存在，则直接返回查询结果。

对于 Query Routing 来说，算法的关键在于节点建立的索引。算法将设置一个参数 hop count 来确定索引的范围。该参数决定了节点维护索引的相邻节点离本节点最远的跳数。如果 hop count 设置得太大，那么节点的存储量将非常大，影响节点的性能。关于节点的索引如何建立，取决于系统的特性，本小节将不做描述。

Query Routing 的路由如图 3-12 所示，hop count 设置为 1，每个节点只记录与它相邻的节点的资源信息。当 P1 发起路由查询时，首先查询本地的索引信息，然后查询相邻节点的资源信息，发现只有 P2 可能提供被检索的数据，所以 P1 只向 P2 构造并发送路由消息，路由消息中还包括 P1 自己的索引信息。P2 接收路由消息后，首先检查自己本地的索引信息，然后再判断自己的相邻节点中 P3 可能提供被检索的数据，于是 P2 重新构造路由消息发送给 P3，其中包括 P1 和 P2 的搜索结果。以此类推，最后 P9 发现没有相邻节点可能提供被检索的数据，就将最终的结果返回给 P1。在 Query Routing 算法中，如果节点一开始就没有检测到任何相关的节点可能能够提供被检索的数据，那么系统可以考虑直接使用 Flooding 算法进行路由。

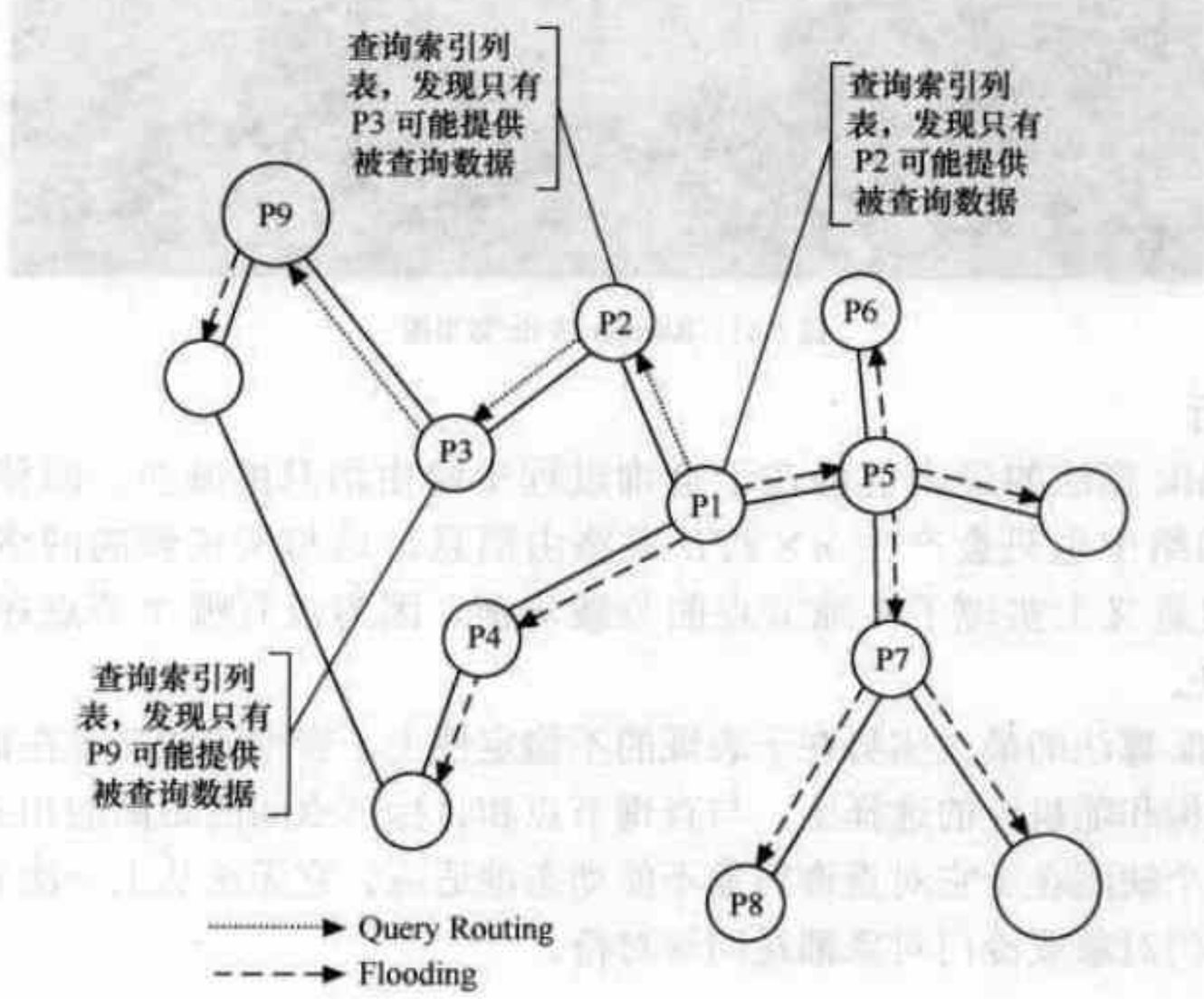


图 3-12　Query Routing 路由

2．算法分析

由于 Query Routing 在路由节点的选择上有很强的针对性，因此路由消息的冗余比 Flooding 算法减少了很多，在路由的时间和网络的负荷两方面都体现出明显的优势。但是这种对路由的有效关联是建立在每个节点都必须建立资源索引的基础上的，所以每个节点需要存储更多的信息，并且对这些信息进行有效的维护，形成系统的额外成本。如何在搜索的网络成本与节点成本上取得平衡，是该算法今后研究的关键点之一。

www.ckook.com

3.4.6 Gnutella2

Gnutella2 与 Flooding 算法有很大的区别，为了减少系统中的路由消息，这种算法采取了两级节点的方法，将系统分成了两级网络。

1．路由算法

这种路由算法中，节点分为两种类型，一种是超级节点，一种是叶子节点。叶子节点只与其归属的超级节点连接，超级节点与超级节点间再连通形成一个网络。超级节点间定期互相更新本地的存储数据表，以过滤不必要的通信。具体的路由算法如下：

当叶子节点发起查询的时候，首先在它所归属的超级节点的索引中寻找，如果找到匹配的对象，则返回结果，否则，超级节点将这个查询请求转发给它所连接的其他的超级节点，直到找到匹配的对象。

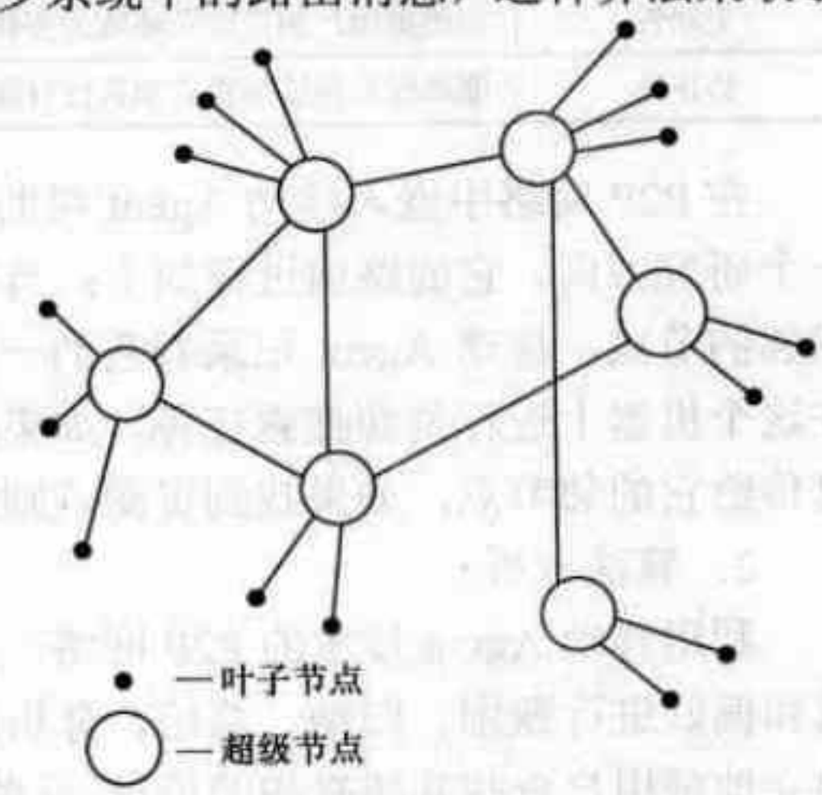


图 3-13 Gnutella2 网络拓扑

Gnutella2 的网络拓扑如图 3-13 所示。

2．算法分析

Gnutella2 与 Gnutella（Flooding 算法）在算法上存在很大的差异。Gnutella2 所构成的网络中，节点与节点间不再是完全平等的实体，超级节点负责了大部分的路由功能，普通的叶子节点的负载变得非常小。

这种路由算法的优势在于，路由过程中，网络中产生的流量负荷大大减少，普通叶子节点的负载大大降低，查询的延时很小。

但是这种算法的缺陷也是显而易见的。由于存在超级节点，网络的拓扑结构完全依赖于超级节点，当超级节点失效或者遭到攻击时，叶子节点将完全瘫痪。因此，对于超级节点的管理维护，在这种路由算法中是至关重要的。

3.4.7 移动 Agent

移动 Agent 将 P2P 路由与人工智能技术进行了结合，引入一种新的思想进行信息检索，使 P2P 网络的用户应用更加自动化和智能化。

1．路由算法

移动 Agent 是人工智能的一个重要分支，但目前还没有一个标准的定义。简单地说，移动 Agent 是一个能在异构网络中自主地从一台主机迁移到另一台主机，并可与其他 Agent 或资源进行交互的程序。它主要体现了表 3-12 所示的技术特性。

表 3-12 移动 Agent 的技术特性

技术特性	特性描述
智能性	具有丰富的知识和一定的推理能力，能揣测用户的意图，并能处理复杂的高难度的任务，对用户的需求能分析地接收，自动拒绝一些不合理或可能给用户带来危害的要求，而且具有从经验中不断学习的能力，适当地进行自我调节，提高处理问题的能力
代理性	在功能上是用户的某种代理，它可以代替用户完成一些任务，并将结果主动反馈给用户

www.ckook.com

续表

技术特性	特性描述
移动性	可以在网络上漫游到任何目标主机，并在目标主机上进行信息处理操作，最后将结果集中返回到起点，而且能随计算机用户的移动而移动
主动性	能根据用户的需求和环境的变化，主动向用户报告并提供服务
协作性	能通过各种通信协议和其他智能体进行信息交流，并可以相互协调共同完成复杂的任务

在 P2P 网络中嵌入移动 Agent 帮助用户完成信息检索的任务，是目前 P2P 分布式搜索的一个研究方向。它的路由过程如下：当有节点需要搜索的时候，它发送一个移动 Agent 给它相邻的节点，移动 Agent 记录着它的一些搜索的信息。当这个 Agent 到达一台新的机器后，在这个机器上进行资源搜索任务，如果这台机器上没有它想要的资源，则它把这些搜索的信息传给它的邻节点，如果找到资源，则返回给请求的机器。

2．算法分析

利用移动 Agent 技术的 P2P 网络，在用户的个性化方面有着先天的优势。对用户信息需求和偏好进行甄别、归纳、总结，分析用户的兴趣爱好，并借助已经学习到的规则，自动、独立地替用户查找其感兴趣的信息，这些都是移动 Agent 技术所独有的优点。但是，移动 Agent 技术也存在它的缺点，主要表现在以下 3 个方面。

（1）在系统实现方面，由于移动 Agent 技术是建立在人工智能技术基础上的，因此系统实现往往比较复杂；

（2）在节点负载方面，由于移动 Agent 技术要求在节点上运行 Agent，因此在节点负载的消耗上，会比其他 P2P 系统要求更多的资源；

（3）在搜索延时方面，由于移动 Agent 技术的搜索取决于节点 Agent 当前的学习情况，所以不同节点的 Agent 的搜索能力是不一致的，搜索的延时也是相差很大的。

3.4.8 小结

通过对上述几种网络的搜索技术的描述可以看出，对于非结构化 P2P 网络的可扩展路由技术而言，其本质就是通过一种方法尽可能少地覆盖网络中的节点，以达到尽可能快地遍历网络中的目标节点。

由于非结构化的 P2P 网络中没有整个网络的拓扑信息，因此在这样的网络中定位目标节点的唯一方法就是访问足够多的节点，以保证目标节点在这些节点中（从统计学的角度分析）。尽管如此，在达到覆盖目标节点的目的的过程中，还是必须注意以下几个方面。

（1）消息路由过程中的动态终止是非常重要的。依赖于 TTL 机制的路由技术是不具备动态终止能力的。在 Random Walk 中描述的 Checking 机制是路由消息动态终止的一个很好示例。在使用动态终止机制时，必须注意避免对源节点造成阻塞效应。

（2）消息的重复必须尽可能地减少。在最理想的情况下，每个查询消息只访问节点一次。重复访问对于整个网络来说，是一种资源的浪费。

（3）消息覆盖的粒度应该尽可能地小。在路由过程中，消息每一步的递增路由不应该导致巨大的新增被访问节点。在 Flooding 算法中，消息的路由每多增加一步，将导致指数级的新增被访问节点；在 Modified-BFS 算法中，通过设定概率减少每一步新增的被访问节点数；

www.ckook.com

在 Iterative Deepening 算法中，通过控制消息路由的步数减少网络中被访问的节点数；在 Random Walk 算法中，更是将每一步的新增被访问节点数控制为常数级。因为在非结构化 P2P 网络中，定位目标节点所需要的被访问节点数应该是一致的，所以多余的被访问的节点只是增加了网络的负担。

在保证以上各个方面的问题能很好地解决的基础上，路由算法应该尽可能地减少定位节点所需要的时间。

3.5 小世界（Small World）模型

小世界模型基于这样的原则，每个节点都表现出某些可以被捕捉到的兴趣，而兴趣相近的节点所保存的内容和提交的查询也呈现出一定的相关性。通过挖掘每个节点的兴趣，将节点按照它们所表现出的相关性组成网络，使得相关性高的节点在网络中比较近。这种按节点间的相关性所组成的网络表现出和社会网络相近的特性，即所谓的“小世界（Small World）”特性。这些特性被证明对于提高检索效率是非常有效的。

3.5.1 小世界模型概述

Stanley Milgram 实验发现，通过平均 6 人次的熟人传递就可以把社会中任意两个人联系起来，这种现象称为 Small World 现象。Milgram 实验揭示了两个发现：（1）短链效应普遍存在；（2）人们可以找到短链。第 2 个发现说明：当网络呈现某种拓扑结构时，仅利用局部信息就可以有效地找到短链。这个发现为分布式信息搜索提供了契机。

1．小世界模型的引入

在非结构化的 P2P 网络中，普遍采用洪泛算法进行路由，与 DHT 路由算法相比，可靠性差，并且对网络资源的消耗较大。最新的研究集中于两个方面展开：提供路由算法的可靠性和寻找随机图中的最短路径。而幂规律分布和小世界（Small World）特征证明，实际网络的拓扑结构既不是非结构化网络中的一个完全随机图，也不是结构化网络中的确定性拓扑图。

（1）幂规律分布指出，在实际网络中，有少数节点拥有较高的“度”，而多数节点的“度”较低。“度”代表了一个节点同其他节点的联系程度。因此，通过“度”较高的节点找到待查信息的概率也较高。

（2）小世界模型特征指出，网络拓扑具有高聚集度和短链的特性。在以小世界特征为基础的网络模型中，节点按照其自身的聚集度被划分为若干簇，在每个簇中至少存在一个“度”最高的节点作为中心节点。大量的研究表明，P2P 网络符合小世界的特征，这意味着网络中存在大量的高连通节点，部分节点之间存在“短链”现象。

通过对以上两点的分析，P2P 网络的路由算法的本质就发生了变化：从如何缩短路径长度的问题变成了如何找到“短链”的问题。尤其是在 DHT 路由算法中，如何产生和找到“短链”是路由算法设计的一个新的思路。小世界模型的引入对 P2P 路由算法产生了重大影响。

2．小世界网络拓扑

小世界的网络拓扑表明，只需要在规则网络上稍作随机改动，网络就同时具备随机性和

www.ckook.com

规则性两种特性。改动的方法如下：对规则网络中的每一个节点，以概率 p 断开与其连接的边，并从网络中的其他节点随机选择进行重新连接（如果所选的节点已经与此节点相连，则再随机选择别的节点进行重连）。当 p=0 时，该网络就是规则网络；当 p=1 时，该网络就是随机网络；当 $p\in$（0,1）时，将存在一个很大的区域，同时拥有较高的集聚程度和较小的最小距离。

小世界网络的几何性质示意如图 3-14 所示。整个网络的节点数 N=16，每个节点的连接边数 K=4，改动持续发生两圈。当 p=0 时，没有任何节点的边进行了重连；当 p=0.093 75 时，32 条边中有 3 条边进行了重连；当 p=1 时，所有的 32 条边都进行了重连。

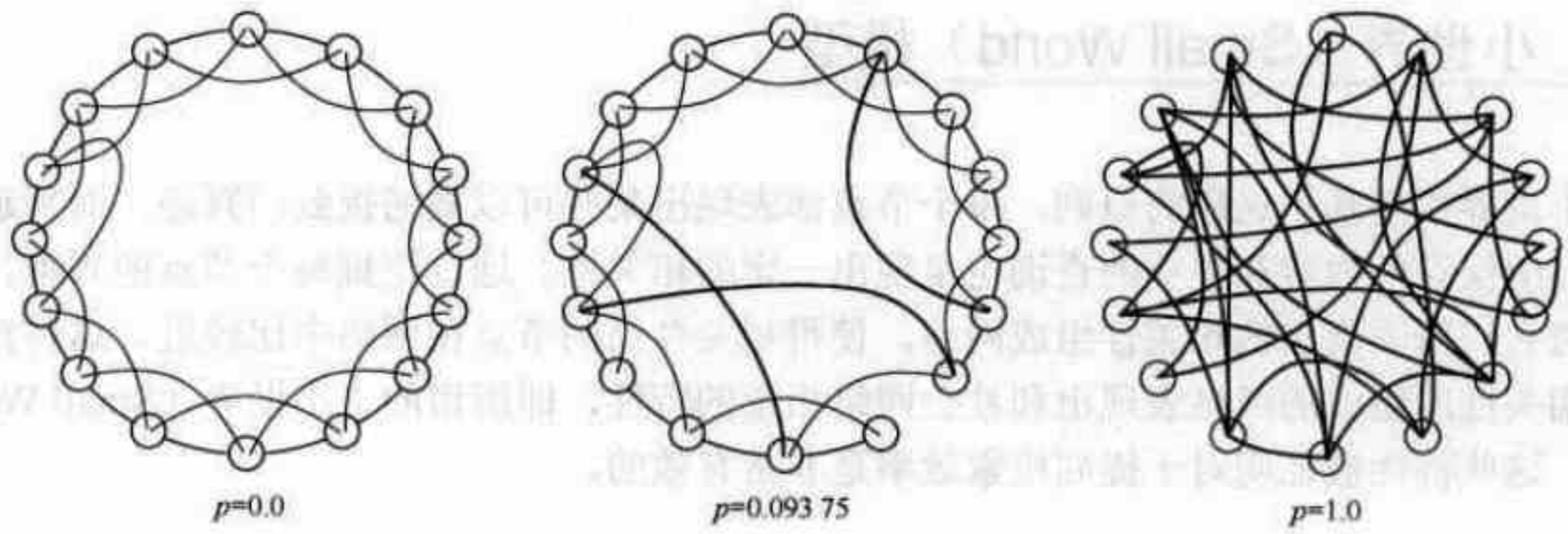


图 3-14　p 值不同的小世界网络

对这个网络的结构特性通过两个要素进行评价：特征路径长度 L（p）和簇系数 C（p）。特征路径长度衡量网络中两点之间的典型距离（全局特性），而簇系数衡量典型邻居的聚合度（本地特性）。在兴趣网络中，很多节点都只有稀疏连接，但稀疏程度又不至于导致与网络失去连接。具体的要求是 $N\gg K\gg\ln(N)\gg 1$，其中 $K\gg\ln(N)$ 保证了稀疏节点不会与网络失去连接。通过对图 3-14 示意的网络分析得出以下结论：在 p=0 的网络中，簇系数很高，特征路径长度随着节点数 N 的增加呈线性增长；在 p=1 的网络中，簇系数很低，特征路径长度随着节点数 N 的增加呈对数增长。这就意味着高聚合伴随着远距离，低聚合伴随着近距离。

但研究显示，存在这样一种临界状态：$L(p)$ 比较低，同时 $C(p)\gg C_{\text{random}}$。导致这种现象出现的原因是网络中存在一些“捷径”，这些“捷径”对 $L(p)$ 的影响是非线性的，而对 $C(p)$ 的影响却是线性的。所以网络在这种临界状态时，虽然特征路径长度缩短很多，但仍具有很高的簇系数。

3.5.2　聚类分布（CD）算法

本节将描述一种非结构化的 DHT 算法——聚类分布（CD, Clustering Distribution）算法，其基本原理是，将一个环状覆盖网络划分为两层——AUT 层和 HUB 层，AUT 负责数据管理，HUB 负责节点路由，每层在局部节点上以聚类方式组织信息。CD 算法初步解决了已有非结构化算法的数据无组织性和搜索盲目性。为了进一步缩短搜索路径长度，在 HUB 层中加入指向远距离节点的远程连接，使得严格聚类转为带有随机性的概率化聚类，从而将搜索路径长度控制在 $O\left(\log^2 N\right)$ 数量级。

1．Clustering Distribution 算法

Clustering Distribution 算法（简称 CD 算法）的索引采用了 DHT，利用高位散列函数将

www.ckook.com

查询词和文件的描述信息（如文件名或关键词等）映射为 Key。Key 按照升序排列，首尾相接，构成一个环形空间。Key 与 Key 的距离等于它们在环形空间上的最短距离。

各个节点拥有独自的聚类中心（Center，简称中心），节点从经过本地的消息中获得对其他节点的信息描述。根据搜索表现更新路由表内容，使内容向着中心进行聚类。各个节点的中心在 Key 空间中均匀分布。聚类中心由系统服务器分配，为了保证系统能够及时追踪时刻变化的查询热点，中央服务器会根据近期查询词的分布为节点分配中心，以保证整个系统的索引均衡性。每个节点维护一张长度为 K 的路由表，如表 3-13 所示。路由表分成 HUB 和 AUT 两部分，长度分别为 KH 和 KA（$KH+KA=K$）。AUT 表记录节点 ID 和代表其文件特征的 Key，HUB 表记录节点 ID 及其中心。从功能上看，AUT 层面从全局角度按 Key 的数值顺序重新分配索引记录，HUB 层面则按节点中心的数值顺序排列节点，形成有序的路由层。在更新过程中，两张表按照一定策略分别进行聚类。

表 3-13　　CD 算法路由表

Node 003　　Center = 4340		
HUB	Center	Node ID
	4616	218
	4855	82
	…	
AUT	Key	Node ID
	4217	264
	4209	318
	4332	403
	…	

搜索时，查询词若与本地文件匹配，则返回结果的同时，更新路径上各个节点的路由表；否则，在 AUT 表中查找查询词，若匹配，则发送消息至对应节点；若 AUT 表不匹配，则再从 HUB 表中寻找最接近查询词的节点（即节点中心与查询词接近），并发送查询词至该节点。

设某次成功的搜索的路径长度为 L，查询词为 q，从起始节点到目标节点依次是：$n_1, n_2, \cdots, n_L$。由于路径上的节点参与了对 q 的搜索，故可以认为它们与 q 的相关性较大，因此，在搜索成功后，CD 算法会对这些节点的路由表进行更新操作。路由表更新策略包括 HUB 表更新策略和 AUT 表更新策略。

（1）对 n_x（x=1，2，…，L−2），进行 HUB 表的更新。

① 令等待加入的新记录 $\mathrm{Key_{Ins}}=\mathrm{center}(n_{L-1})$，$\mathrm{center}(n_{L-1})$即为 n_{L-1} 的聚类中心，相应的记录为<$\mathrm{Key_{Ins}}$，n_{L-1}>。之所以选择 n_{L-1} 的中心为学习目标，是因为 n_{L-1} 是搜索路径的最后一个消息发送者，为搜索成功做出了最重要的贡献。

② 若 HUB 表未满，则加入新记录<$\mathrm{Key_{Ins}}$, n_{L-1}>，更新结束；若 HUB 表已满，则执行③。

③ 待删除的旧记录 $\mathrm{Key_{Del}}$ 为 n_x 的 HUB 表中距离 center（n_x）最大的记录，即 $\mathrm{Key_{Del}}=\max\{D(\mathrm{Key}_x, \mathrm{center}(n_x))\}$，$\mathrm{Key}_x$ 是 n_x 的 HUB 表中的任意一条记录。

④ 如果 $D(\mathrm{Key_{Ins}}, \mathrm{center}(n_x)) < D(\mathrm{Key_{Del}}, \mathrm{center}(n_x))$，则用新记录取代旧记录；否则，不做任何改动。

www.ckook.com

（2）对 n_x (x=1, 2, …, L−2)，进行 AUT 表的更新。

① 因为在 n_L 处搜索到 q，所以 n_L 的文件中必包含 q，路径上的节点将记录该信息。令等待加入的新记录 $Key_{Ins}=q$，相应的记录为<Key_{Ins}, n_L>。

② 若 AUT 表未满，则加入新记录<Key_{Ins},n_{L-1}>，更新结束；若 AUT 表已满，则执行③。

③ 待删除的旧记录 Key_{Del} 为 n_x 的 AUT 表中距离 center（n_x）最大的 Key，即 Key_{Del}=max{ $D(Key_x, center(n_x))$ }，Key_x 是 n_x 的 AUT 表中的任意一条记录。

④ 如果 $D(Key_{Ins}, center(n_x)) < D(Key_{Del}, center(n_x))$，则用新记录取代旧记录；否则，不做任何改动。

经过足够多的更新后，HUB 和 AUT 中的 Key 可以有效地聚类在中心处。

直观来讲，CD 算法的思想就是让各个节点在整个 Key 空间上各司其职，分工合作，如图 3-15 所示。AUT 负责管理 Key 空间内一段区域的 Key（但并不保证该区域内的所有 Key 都放在同一个节点上），该区域在 Key 空间的中心位置由节点中心决定。AUT 表的聚类可以在原来节点层面之上重建全局有序的管理层；HUB 记录节点中心，中心代表了节点的职能，通过 HUB，节点能够了解彼此的职能，聚类后的 HUB 可以在 AUT 层面上建立有序的路由层。

在图 3-15 中，一组同心圆表示节点的 3 个层面，由外向内依次为节点/文件层、AUT 层和 HUB 层，同一节点在多层中的逻辑位置处于一条经过圆心的直线上，如图 3-15 中的节点 A 和 B。在节点/文件层，Key 无规律地分布在各个节点上，在此层面上无法实现全局搜索；AUT 层重新调整了 Key 的分布，Key 分别向相关节点聚类，每个节点 AUT 分管中心附近的一段 key；HUB 层在 AUT 层之上建立了有序的路由层，每个节点 HUB 分管中心（本节点）附近的一段中心（其他节点）。

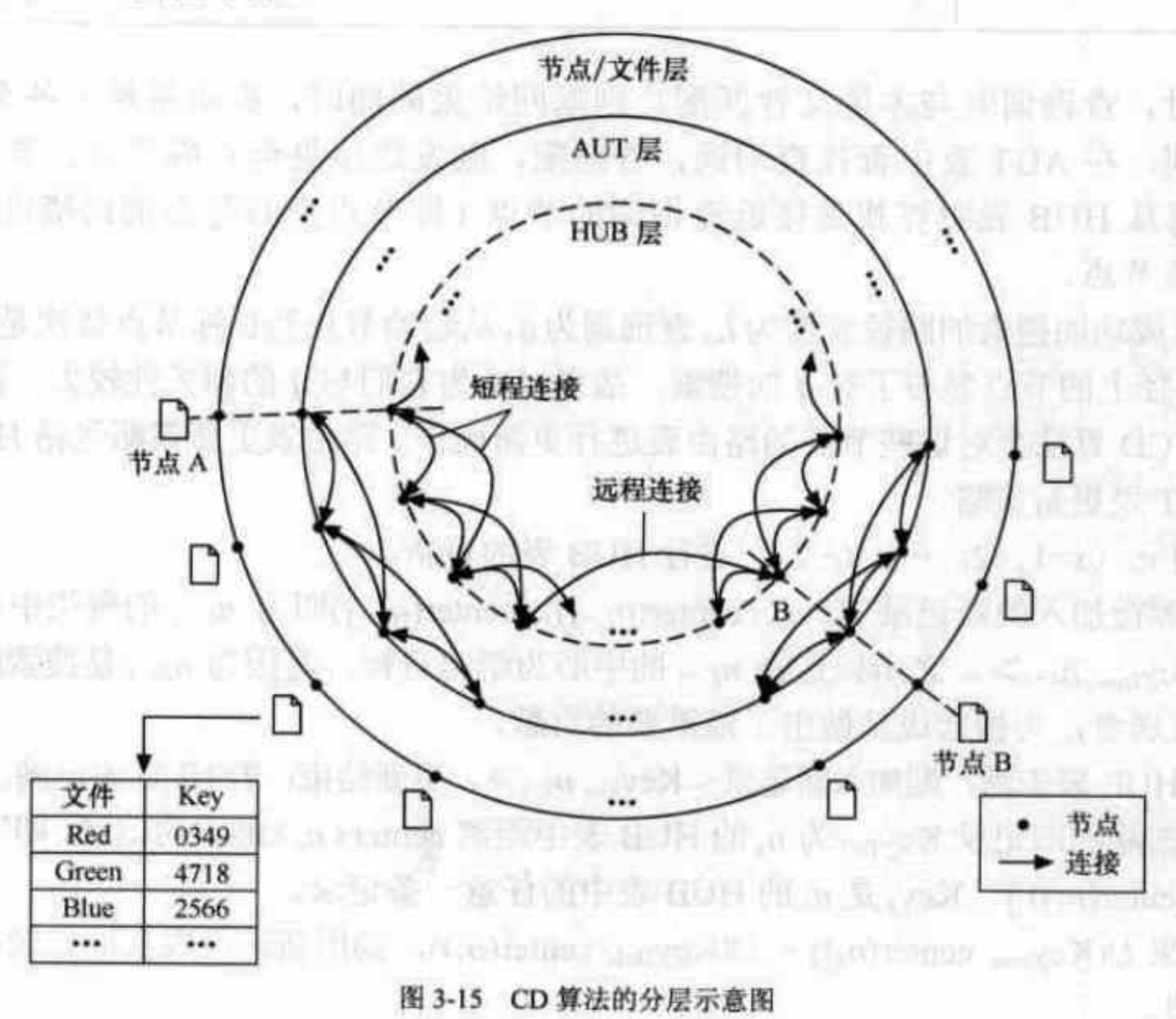


图 3-15　CD 算法的分层示意图

为了保证导航方向和搜索目标的一致，需 HUB 的聚类中心与 AUT 相同。HUB 的作用只是有机地将各个节点联系起来，形成全局的搜索层；而定位目标文件最终是要通过 AUT 表实现。如果 AUT 和 HUB 的中心不同，HUB 层的搜索将失去意义。

AUT 记录了其他节点文件的 Key，当查询词与 AUT 的某条记录匹配时，则表明距离目标节点仅一步之遥；而 HUB 确保了搜索沿着正确方向前进。AUT 可以看作是文件存储库（事实上距文件尚有一步），HUB 可以看作是导航信息库。这种双层结构的功能划分十分类似于因特网上的 Web 网站，网站既链接资源（如文本、媒体），又链接其他同类站点。

2．Enhanced Clustering Distribution 算法

在 CD 算法的基础上，通过对 HUB 表的更新策略进行改进，提出了 ECD（Enhanced Clustering Distribution）算法，其 AUT 表更新策略与 CD 算法相同。

ECD 算法对 n_x（x=1,2,⋯,L−2），进行 HUB 表的更新。

（1）令等待加入的新记录 $\mathrm{Key_{Ins}}$=center（n_{L-1}），center（n_{L-1}）即为 n_{L-1} 的聚类中心，相应的记录为<$\mathrm{Key_{Ins}}$,n_{L-1}>。

（2）若 HUB 表未满，则加入新记录<$\mathrm{Key_{Ins}}$, n_{L-1}>，更新结束；若 HUB 表已满，则执行（3）。

（3）计算 HUB 表中每条记录的删除概率，记 u=center（n_x），则 Key v 的删除概率 $P(v)=\dfrac{D(u,v)^{-1}}{\sum_{w} D(u,w)^{-1}}$，其中 w 为 HUB 表中的任意 Key。根据删除概率随机选择一个 Key 为待删除记录，记为 $\mathrm{Key_{Del}}$。

（4）重新计算 $\mathrm{Key_{Del}}$ 和 $\mathrm{Key_{Ins}}$ 的删除概率

$$P(\mathrm{Key_{Del}})=\frac{D(u,\mathrm{Key_{Del}})^{-1}}{D(u,\mathrm{Key_{Del}})^{-1}+D(u,\mathrm{Key_{Ins}})^{-1}}$$

$$P(\mathrm{Key_{Ins}})=\frac{D(u,\mathrm{Key_{Ins}})^{-1}}{D(u,\mathrm{Key_{Del}})^{-1}+D(u,\mathrm{Key_{Ins}})^{-1}}$$

根据删除概率随机确定删除对象，对 HUB 表做出相应调整。

可以看出，当 HUB 表填满时，ECD 算法至少引入一条远程连接。

3．算法分析

在 CD 算法的 HUB 层中，连接规则分布，从 A 到 B 需要通过邻居依次传递消息，路径长度正比于 D(center(A), center(B))；而 ECD 算法中存在着少数的远程连接，它们起到缩短路径的作用。但是，这并不等于说远程连接越多越好，短程连接和远程连接在搜索中都分别起着重要的作用。细粒度的短程连接保证搜索的获得性（Availability）和可靠性（Reliability），粗粒度的远程连接起到快速调度搜索消息的作用。

路由表的总容量固定，但 HUB 表和 AUT 表的容量比例并没有限制，从功能上看，二者分别起到了导航和管理 Key 的作用，缺一不可。如果 HUB 表比例过低，搜索的导航信息不足，成功率将严重下降；如果 HUB 表比例过高，AUT 表相对变小，所管理的 Key 数目过少，将直接影响搜索成功率。

www.ckook.com

节点失效是无组织系统必须要考虑的问题。当节点出现了失效情形时，除了失效节点的路由表不再发挥作用，工作节点的路由表中涉及到失效节点的记录也失效。试验显示 ECD 算法中即使存有少数 HUB 连接仍能保证有效搜索，说明 ECD 具有良好的容错性。同时，聚类的 AUT 表缩小了目标查找范围，节省了消息发送数量。相比之下，组织无序的路由表的搜索能力完全依赖于有效记录的数量，在节点失效时，搜索的盲目性会更加突出，成功率往往会随着失效率的增加而下降。

3.5.3 小世界网络的研究现状

目前针对小世界网络的主要研究是按照对用户检索行为的分析来划分用户兴趣的。这在一定程度上提高了检索的效率。但是检索行为只能反映用户部分兴趣，尤其对于提供大量信息的节点，只能展示它的一个很小的子集。对于共享内容的深度挖掘，成为一个十分迫切的问题。如何准确地定位节点的兴趣集合，动态地反映网络中的资源情况，公平地分担网络的负载，成为小世界网络研究所急需解决的关键点。

3.6 P2P 搜索技术研究的挑战

目前针对 P2P 搜索技术的研究中，最重要的两个成果分别是基于 DHT 的结构化搜索算法和基于 Small World 理论的搜索算法。特别是 DHT 的结构化搜索算法为资源的组织和查找提供了一种新的方法，成为近年来 P2P 研究领域的热点。

但是 P2P 系统在实际的应用中，一些其他方面的因素（例如物理网络的特性、复杂查询的需求等）开始影响 P2P 路由算法的效率，主要包括表 3-14 所示的几个方面。

表 3-14　　影响 P2P 路由算法效率的因素

影响因素	具体描述
网络节点的异质性	由于客户端/服务器模式在 Internet 和分布式领域多年的应用以及种类繁多的电子设备的普及（如便携式计算机、移动电话和 PDA），网络中节点在计算能力、存储空间和持续时间等方面有很大的差别。节点所承担的负载应该与节点的处理能力呈现一定的相关性，否则会影响路由算法的效率
网络节点的物理距离	由于实际的网络被路由器和交换机分割成不同的自治区域，体现出严密的层次性。节点的逻辑相邻应该与物理相邻呈现一定的相关性，否则会影响路由算法的效率
网络的波动程度	网络波动（Churn）是指包括节点的加入、退出、失败、迁移、并发加入过程、网络分割等行为。DHT 路由算法都是考虑在网络波动的最差情况下仍然能够正常运行，所以要求每个节点维护的拓扑尽可能地小，以便节点失效时造成的影响尽可能地小。同时要求节点定期更新自己的路由信息，以保证在动态的环境下路由成功地进行。但是，这样做也造成了路由算法相对的高延时和网络中冗余信息增加。如何平衡网络波动程度与节点的拓扑维护权重，是提高路由算法效率的重要因素
复杂查询的需求	作为资源组织与搜索技术，必然要求支持复杂的查询，如关键字、内容查询等。尽管信息检索和数据挖掘领域提供了大量成熟的语义查询技术，但是由于 DHT 具有精确关键字映射的特性，所以阻碍了 DHT 在复杂查询方面的应用。目前如何使 DHT 网络高效地支持多关键字查询，也是 P2P 搜索技术急需解决的关键问题之一

P2P 搜索技术一直是研究的热点，很多的文章都针对已有的路由算法提出了补充、完善和创新。但是，在资源搜索效率、资源准确定位、复杂查询等方面，P2P 搜索技术还存在很大的提升空间。另外，目前 P2P 搜索技术虽然在理论研究上比较热门，但是在具体的应用上

www.ckook.com

仍然与传统的搜索引擎技术（如 Google、百度等）具有很大的差距。如何将理论研究中 P2P 搜索技术所体现的优势转化为实际应用中用户所能感受到的服务，将是 P2P 搜索技术以后长期努力的方向。

3.7 本章总结

本章简要地介绍了传统搜索技术的基本情况，概括性地分析了 P2P 搜索技术的发展背景和技术特点，着重针对结构化的 P2P 网络和非结构化的 P2P 网络的特性进行了详细的阐述。在结构化的 P2P 网络中，本章集中介绍了 Chord、Pastry、CAN、Tapestry、Kademlia 等 5 类 DHT 网络，对每个网络的路由机制和算法特性进行了分析；在非结构化的 P2P 网络中，本章集中介绍了 Flooding、Modified-BFS、Iterative Deepening、Random Walk、Gnutella2、基于移动 Agent 的搜索技术、Query Routing 等搜索算法，对每个算法的路由机制和算法特性进行了描述。另外，本章还对目前处于起步阶段的小世界模型的网络拓扑进行了简单的分析，并且介绍了一种针对该模型的算法。

本章力图帮助读者梳理 P2P 网络的网络拓扑，希望读者通过本章能够对 P2P 网络的路由机制有一个基本的认识。在各类 P2P 网络的算法原理阐述和算法分析过程中，本章引用了大量的图表，希望读者能够对这些 P2P 网络建立一个直观的、生动的印象。

在本章的参考文献中，包括了本章所介绍的所有的 P2P 网络的详细介绍，如果读者希望对 P2P 网络的路由算法有进一步的认识，可以查阅这些资料进行深入的学习。在后续的练习题中，主要是对本章所涵盖的知识点的考查，另外还包括一部分启发性的思考题，希望读者可以通过这些练习题，整理自己的思路，加深对 P2P 网络的路由机制的了解和认识。

3.8 练习题

（1）请描述传统搜索技术与 P2P 搜索技术的区别。
（2）请描述结构化分布式 P2P 网络的优点和缺点。
（3）请描述 5 类结构化分布式 P2P 网络的路由算法。
（4）用 C++语言实现 Chord 网络的路由协议和关键字查找、节点加入、节点离开等基础操作。
（5）思考结构化分布式 P2P 网络中孤立网络合并的方法，以 Chord 网络或者 Pastry 网络为例。
（6）请描述 7 类非结构化分布式 P2P 网络的路由算法。
（7）用 C++语言实现 Flooding 的基础路由算法。
（8）比较 Flooding 和 Gnutella2 之间的异同。
（9）列举非结构化 P2P 网络路由过程所必须注意的要点。
（10）描述小世界网络的拓扑特性。

参 考 文 献

[1] I. Stoica R, Morris, D. Karger M Kaashoek F. Balakrishnan H. Chord: A scalable

www.ckook.com

peer-to-peer lookup service for Internet applications. In Proc. ACM SIGCOMM'01, San Diego, CA, Aug. 2001

[2] Druschel P, Rowstro A. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Submission to ACM SIGCOMM, 2001

[3] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable contentaddressable network. In Proc. ACM SIGCOMM'01, San Diego, CA, Aug. 2001

[4] Zhao B Y, Kubiatowicz J D, Joseph A D. Tapestry: An infrastructure for faultresilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001

[5] 徐恪，熊勇强，吴建平，等．对等网络研究综述．国家自然科学基金重点项目，编号：90104002、60203025，2005

[6] 罗杰文．Peer to Peer（P2P）综述．http://www.intsci.ac.cn/users/luojw/papers/P2P.htm.2005

[7] 吕建明，刘悦，丁林，许洪波，程学旗等．P2P 与信息检索．信息技术快报，第 3 卷第 2 期，2005

[8] Tsoumakos D, Roussopoulos N. A Comparison of Peer-to-Peer Search Methods. Proc. 6th Int'l Workshop Web and Databases, (WebDB 2003), 2003, pp. 61-66

[9] Kalogeraki V, Gunopulos D, Zeinalipour-Yazti D. A Local Search Mechanism for Peer-to-Peer Networks. In CIKM, 2002

[10] Lv C, Cao P, Cohen E, Li K, Shenker S. Search and Replication in Unstructured Peer-to-Peer Networks. In ICS, 2002

[11] Yang B and Garcia-Molina H. Improving Search in Peer-to-Peer Networks. In ICDCS, 2002

[12] 杨鲲，刘大有，郭欣，等．一个具有高安全性的移动 Agent 系统模板结构．国家 863 青年基金资助项目（863-306-QN2002-1），2000

[13] 黄镝．基于信息 Agent 技术的个性化学术资源信息系统．http://www.lib.sjtu.edu.cn/chinese/teaching&research/501.doc, 2006

[14] 刘艳．网络搜索引擎与智能代理技术．计算机工程，2002（3）：45-46

[15] Watts D J, Strogatz S H. Collective dynamics of small-world networks. Nature, 1998, 393: 440-442.

[16] 周晋，李衍达，等．基于 Small World 网络的非结构化 DHT 算法．国家自然科学基金项目（60003004），2005

[17] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. of the 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS 2002). 2002. 153-161

www.ckook.com

CHAPTER 4

第 4 章　P2P 应用与安全

本章将从 P2P 网络所引发的、所面临的安全问题以及它为网络所带来的新的安全保障 3 个方面对 P2P 网络与信息安全的关系进行阐述。

在学习了本章后，希望读者能够掌握以下几点：

（1）P2P 网络的出现所引发的安全问题；

（2）P2P 网络应用所需要解决的安全问题；

（3）P2P 技术为网络安全带来了哪些新的解决方案；

（4）P2P 技术的哪些特点决定了其目前的安全现状。

4.1　P2P 应用面临的安全问题

P2P 网络相对传统网络中的 C/S 结构而言是一种全新的变革，然而正是这样一种变革不可避免地带来了一系列的问题与挑战。其中，P2P 应用所面临的安全问题是最为突出的问题之一。只有这些问题顺利解决，才能为 P2P 网络的发展扫清障碍，提供更广阔的空间。

本节将就这些安全问题展开详细的论述。主要包括 P2P 应用引发的版权问题、P2P 应用对现有网络应用的威胁、P2P 网络病毒与蠕虫，结构化 P2P 网络的隐患等。

4.1.1　P2P 应用引发的版权问题

在 P2P 共享网络中普遍存在着知识产权保护问题。尽管目前 Gnutella、Kazaa 等 P2P 共享软件宣称其骨干服务器上并没有存储任何涉及产权保护的内容的备份，而仅仅是保存了各个内容在因特网上的存储索引，但毋庸置疑的是，P2P 共享软件的繁荣加速了盗版媒体的分发，增加了知识产权保护的难度。美国唱片工业协会（RIAA，Recording Industry Association of America）与这些共享软件公司展开了漫长的官司拉锯战，著名的 Napster 便是这场战争的第一个牺牲者。表 4-1 列出了 Napster 诉讼的大事记。

www.ckook.com

表 4-1　　Napster 诉讼的大事记

时　间	事　件	备　注
1999 年 5 月	由 Shawn Fanning 和 Sean Parker 共同创办的文件共享社区网站——Napster 正式成立	该公司最主要的服务是，为其用户提供一种便捷、易用的界面以实现媒体文件的搜寻及共享；同时还为音乐迷们提供相互交流的论坛，以及实时讯息、聊天室、用户书签等产品
12 月 7 日	美国唱片工业协会（RIAA）代表环宇音乐、索尼音乐、华纳音乐、百代唱片、BMG 等 7 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赔偿损失 1 亿美元
2000 年 4 月 13 日	重金属乐队 Metallica 起诉 Napster，称其侵犯了自己的版权，并涉嫌诈骗	此后不久，歌星 Rapper Dr. Dre 起诉 Napster；Metallica 乐队的击鼓手 Lars Ulrich 和律师则对多达 33.5 万名的 Napster 用户提起诉讼，这些用户通过 Napster 提供的服务非法共享该乐队的音乐
5 月 5 日	美国的地区法官 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	
6 月 12 日	美国唱片工业协会（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”
7 月 11 日	参议院就围绕 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介入双方的争端，以免影响新技术的发展
7 月 26 日	Patel 同意美国 RIAA 的要求，作出初步判决，命令 Napster 立即停止服务	
7 月 28 日	美国第九巡回上诉庭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	
10 月 31 日	Napster 宣布同德国的媒体巨人贝特斯曼集团组成伙伴关系，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，贝特斯曼同意撤销对 Napster 的起诉。Napster 可使用贝特斯曼的音乐	
2001 年 2 月 12 日	美国第九巡回上诉庭作出决定，Napster 必须终止其免费因特网服务，并不再向音乐迷提供共享版权保护音乐的服务	
3 月 2 日	Napster 公司宣布，该公司将开始阻止用户们下载大约 100 万首受版权保护的音乐曲目，以遵守 Patel 随时可能下达的一项新的法律禁令	
3 月 6 日	美国地区法官 Marilyn Hall Patel 做出判决，责令 Napster 在 5 个工作日内删除所有存在争议的歌曲	

吸取了 Napster 等 P2P 软件公司的教训，当今的 P2P 共享软件也迫切需要寻找到一条可以与媒体发布厂商共生的道路。只有解决了法律问题，才有可能使 P2P 共享软件真正地走上健康长久的发展道路。

网络社会与自然社会一样，其自身具有一种自发的在无序和有序之间寻找平衡的趋势。P2P 技术为网络信息共享带来了革命性的改进，而这种改进如果想要持续长期地为广大用户带来好处，必须以不损害内容提供商的基本利益为前提。这就要求在不影响现有 P2P 共享软件性能的前提下，在一定程度上实现知识产权的保护机制。目前，已经有些 P2P 厂商和其他公司一起在研究这样的问题。这也许将是下一代 P2P 共享软件面临的挑战性技术问题之一。

www.ckook.com

4.1.2 P2P 应用对现有网络应用的威胁

随着因特网技术和应用的深入发展，当今 P2P 的应用已经是非常普遍的事情，然而 P2P 的广泛应用也着实让电信运营商感到头疼，主要表现在以下几个方面。

（1）P2P 的传播方式决定了它对带宽的需求在理论上是无止境的。许多 P2P 客户端都采用了多点连接的下载方式，即每位下载者在获取数据的同时，还成为其他下载者数据的来源，这样下载的人数越多，下载的速度就越快。当用户需要在网络中下载数据时，会首先向邻接节点发出搜索请求，如果邻接节点的搜索结果不能满足要求，则会继续向外迅速扩散，直到达到要求或所有的搜索节点都被搜索到为止，它使得原来运行畅通的网络变得越来越拥塞。曾有调查显示，目前因特网上的 P2P 流量占据了整个网络流量的 60%左右，在高峰时甚至能达到 80%。

（2）P2P 网络的流量极大地改变了传统的网络流量模型，使运营商的带宽投入产出比下降。目前国内的宽带资费主要有全包月、半包月和计时等模式。这些资费模式和门限的设置多是参照普通因特网业务而设计的，普通因特网业务具有较大的忙时、忙日集中系数，同时流量占空比也较大，这使得对带宽资源的消耗相对较小。在这种情况下，资费和流量处于一种相对的平衡状态。但 P2P 的出现使得用户流量大幅度增加，产生的流量远远超过一般的因特网业务，也就是说，业务源的模型发生了变化，但资费仍然保持原有的模式，因此造成了运营商整体投入产出比的下降。这也是运营商对 P2P 网络流量排斥的主要因素之一。

（3）用户满意度下降。运营商 IP 网络的设计目前也是参考 Client/Server 模式进行的，考虑了普通因特网业务的强突发特性。但随着 P2P 增长带来的带宽需求规模的逐步扩大，网络的建设理念已经不适应流量的增长模式。对于数量占绝大多数的普通因特网业务而言，其所占的带宽资源正在被 P2P 应用侵蚀，致使因特网服务的质量受到影响，无法保持客户满意度。

（4）网络基础设施不堪重负。网络的接入设备，例如汇聚层的路由器、以太网络交换机、网桥、DSLAM、B-RAS 以及后台的认证系统等，是不可能具有无限的处理能力的。因此，当线路上用户使用量和通信状态数目急增时，运营商的现有网络基础设备将不能支持足够多的用户。

综合以上的问题，P2P 流量引起了运营商很大的重视。如何识别 P2P 流量、引导 P2P 流量，使其更加规范化，从而与现有的网络应用共同发展是包括运营商在内的许多业内人士共同关注的一个研究方向。至于详细的流量识别方法，会在 P2P 网络安全技术中进行介绍。

4.1.3 P2P 网络病毒与蠕虫

随着计算机网络应用的深入发展，计算机病毒对信息安全的威胁日益增加。特别是在 P2P 环境下，方便的共享和快速的选路机制，为某些网络病毒提供了更好的入侵机会。

由于 P2P 网络中逻辑相邻的节点，地理位置可能相隔很远，而参与 P2P 网络的节点数量又非常大，因此通过 P2P 系统传播的病毒，波及范围大，覆盖面广，从而造成的损失会很大。

在 P2P 网络中，每个节点防御病毒的能力是不同的。只要有一个节点感染病毒，就可以通过内部共享和通信机制将病毒扩散到附近的邻居节点。在短时间内可以造成网络拥塞甚至瘫痪，共享信息丢失，机密信息失窃，甚至通过网络病毒可以完全控制整个网络。

一个突出的例子就是 2003 年通过即时消息（Instant Message）软件传播病毒的案例显著

www.ckook.com

增多。包括 Symantec 公司和 McAfee 公司的高层技术主管都预测即时消息软件将会成为网络病毒传播和黑客攻击的主要载体之一。

随着 P2P 技术的发展，将来会出现各种专门针对 P2P 系统的网络病毒。利用系统漏洞，达到迅速破坏、瓦解、控制系统的目的。因此，网络病毒的潜在危机对 P2P 系统安全性和顽健性提出了更高的要求，迫切需要建立一套完整、高效、安全的防毒体系。

4.1.4 结构化 P2P 网络的隐患

P2P 系统以其能够充分利用网络资源、自适应性强、扩展性好、可靠性高等诸多优点在信息共享、分布式计算以及网络数据库等诸多领域得到了广泛的应用。但由于目前许多 P2P 底层技术还不很成熟，其安全问题，尤其是结构化 P2P 网络中分布式散列表（DHT）系统的安全问题越来越引起研究人员的重视。目前 P2P 的研究热点主要集中在如何提供有效的散列查找系统并基于它构建更复杂的系统。

查找的过程需要向一系列的节点发送路由查询请求，而每个收到查询请求的节点都需要通过查询自身维护的本地路由表来找到下一个节点，从而继续传送查询请求，直到找到真正被查询的节点为止。

当正常运行时，这些系统都能表现出高效的特性，但是其中的大多数系统都没有或很少考虑到恶意节点的问题。如果其散列查找系统受到攻击，整个系统的正常运行必然受到严重威胁。

大部分的 P2P 网络默认所有参与在网络中的节点都是可信的。在一个相对封闭的网络中，比如在某个防火墙之后，这种假设也许能成立。但当在一个开放的环境中时，比如在 Internet 中，这种假设显然会带来很多的安全隐患。有些 P2P 系统提出了建立认证中心的解决方案，并提出通过对每个 P2P 网络的用户收取费用来提高攻击者的攻击开销。但是在实际中，很多系统无法限制或不愿限制更多的用户参与到自己的 P2P 网络中。针对这种情况，就只能找出能够使系统在恶意节点的参与下仍能正常运行的解决方案。

基于 DHT 的查找系统具有良好的搜索性能，但 DHT 路由方法面临着如负载均匀不易、网络拓扑维护代价大、Key 的同步维护困难等固有问题，特别是 DHT 查找系统的安全问题已经非常突出，这不仅在 P2P 理论技术领域而且在 P2P 应用领域都引起了研究人员的高度关注。

下面列出了 P2P 网络的 DHT 查找系统面临的常见安全威胁及其解决方案。

1．路由攻击

路由攻击，指的是攻击者作为 DHT 查找系统的参与者，冒充正常节点把通过它进行的查找请求转发到不正确或不存在的节点，同时向其他节点发送不正确的路由信息来破坏其他节点的路由表，从而导致系统节点进行不正确的路由操作。

在查询协议中的路由包括：维护路由表，向路由表中的节点分发查询请求。在 DHT 中对于路由正确性的要求非常重要。然而，攻击者在这个方面可做的攻击有很多。

（1）错误的路由查询及解决方案

在 P2P 网络中，路由查询是通过网络中的普通节点来进行的，很难保证这些参与到路由查询中的节点都是正常的网络用户。如果恶意节点参与到路由的过程中，就可以将查询指向一个不正确的甚至根本不存在的节点。而且，一旦恶意节点参与到 P2P 网络中，并进而参与到路由过程中之后，会保持一直在线而不失效，这样，被重定向后的查询请求可能会被安排

www.ckook.com

指向另一个恶意节点。

幸运的是，随意的错误指向很容易被发现。发起查询的节点应该知道，查询过程的每一跳都是以更接近关键字的 ID 为目的的。遵循这一原则，对路由的每一跳进行检查就可以很容易地发现是否被攻击。如果发现了这种攻击，那么发起查询的节点可以返回到路由过程中正常的节点处，再重新选择下一跳路由。为了使得发起查询的节点能够进行这项检查，它必须能够知道查询过程中的每一跳。

另外，恶意节点也可以简单地宣称任意一个节点与某个 Key 相关联。由于发起查询的节点可能与目的节点有一定的距离，因此并不知道被宣称的节点是否真正地与该 Key 相关联。这样将会导致这个 Key 被存放在一个不正确的节点上或者无法被找到。

解决这个问题，需要在两个方面做出保证。

① 发起查询的节点应该能够验证目的节点确实是所查找的正确节点。

② 系统应该保证关键字与节点的绑定是可验证的。

有些 P2P 系统，比如 Chord，将节点的 IP 地址和端口通过加密的散列运算得到该节点的 ID。这样，节点的 ID 就是相对稳定且不可伪造的了。此外，有些系统希望通过公钥体制建立一个可长期存在的节点 ID。这种机制可使得系统对于消息的源以及消息内容的有效性进行验证。但是由于数字签名的费用过高，这种机制的应用受到一定程度的限制。

在某些 P2P 系统中，关键字被绑定在与其 ID 相近的节点上。也就是说，假如关键字 ID 是 10，那么这个关键字就只可能绑定在 ID 与 10 相近的节点上。如果有两个节点，ID 分别为 10 和 16，则该关键字就会和节点 ID 为 10 的节点进行绑定。这样，关键字与节点的绑定关系就是确定的。由于关键字和节点的 ID 都是通过散列函数产生的，因此是不可随意改变的。也就是说，恶意节点无法随意地宣称自己或某个节点与给定的关键字相关联。

（2）错误的路由更新

由于 P2P 网络中每个节点所保存的路由信息是通过对其他节点的询问得到的，所以一个恶意节点可以通过向其他节点发送虚假的路由更新请求来达到破坏各节点路由表的目的。这种攻击的效果可能导致一些无辜的节点也将查询请求错误地传给不正确的、恶意的，甚至是不存在的节点。

但是，如果系统能够识别出正确的路由更新请求，那么就可避免这种攻击。比如，在 Pastry 中，路由更新请求要求每一个新加入路由表的节点都有一个正确的前缀，而路由表中已有的节点必须被证明仍然可用之后，该节点的信息才能被更新。

另外，有些 P2P 系统中允许节点在备选的多条路径中，自由地选择一条最佳路径，比如，在 CAN 中的 RTT 优化设置。这本来是一种用于优化路由，并使系统更加灵活的机制，但却可以被攻击者所利用。一个恶意节点可以在路由选择过程中，提供一个不存在的节点、一个具有很大延时的节点甚至另一个恶意节点，从而达到其攻击目的。因此，P2P 协议的设计者应注意防止基于主机的路由选择被恶意节点所利用而带来的威胁。

（3）节点分割

节点分割指的是，恶意攻击者掌握了一定的网络资源后，构成一个“陷阱网络”，其中某些节点还可能同时是真实网络中的节点。当要加入 P2P 网络的新节点选取“陷阱网络”中的某个节点作为初始化节点以获取相应的初始化信息时，它就落入了由攻击节点组成的网络，从而和真正的网络分隔开了。

www.ckook.com

3．恶意攻击

（1）不一致的行为

当一个节点总是做出不恰当的表现时，诸如经过它查询到的路由节点总是不可用，或者向它发送的资源请求总是得不到响应时，P2P 网络就能够判定该节点为恶意节点。当一个节点被判定为恶意节点时，就会被网络所摒弃。但是，当一个恶意节点对网络中的一部分节点表现正常，而对其他的节点响应不正常时，问题就会比较复杂。因为对该节点的表现，网络中的一部分节点会给出肯定的结果，而另一部分节点的结论则恰恰相反。举个例子：假设一个恶意节点对其邻近的节点表现出正常的行为，而对远距离的节点请求做出非正常反应。这样，虽然远程的节点发现了该节点的异常，但是与其邻近的节点却找不到任何需要将该节点清除出路由表的理由。

当然，理想的情况是，远程的节点能够将恶意节点的不正常表现告知其邻近的节点，并使得这些临近的节点将该恶意节点清除出路由表。但是，这又引发了一个问题，那就是，如何验证一个来自远程节点的报告是真实的？也就是说，这份报告有可能是真的对恶意节点的揭发，但同时也有可能发送报告的远程节点本身是恶意节点，编造了一份假报告，从而试图将正常的节点排除于网络之外。那么对于邻近节点来说，如何验证报告的真实性就是又一个亟待解决的问题了。另外，如果每个节点都必须在它们的报告中加入由其公钥所生成的数字签名，这样一份恶意的报告就可被识别出来。如果没有采用报告的机制，每个节点就只能靠自己来对其他节点的行为进行判断。

（2）拒绝服务攻击

拒绝服务攻击即 DoS（Denial of Service）攻击。这是目前一种常见的网络攻击方法。以占用目标主机或目标服务器的 CPU、内存及网络带宽等资源，使其无法正常对外提供服务，或无法正常运作为目的。DoS 攻击主要方法为向目标发送大量的垃圾数据包，使其来不及响应；或者利用目标方的漏洞精心构造非正常数据包，造成目标方系统的崩溃。

对 P2P 网络的拒绝服务攻击与对传统网络的攻击在方法上并没有太大的区别。因此，P2P 网络对于拒绝服务攻击的防御措施也可参考传统 C/S 模式网络的做法，即采用冗余备份（将数据在网络中的多个节点上进行备份）以及负载均衡（将数据较为平均地存储于网络中的各个节点，尽量避免少数节点负荷过重的问题，否则这些少数节点很可能成为拒绝服务攻击的目标）等机制。

值得指出的是，由于 P2P 网络中在设计之初就考虑到了单个节点失效的问题，以及在网络中各节点地位的平等性，使得 P2P 网络不会因为某个节点的崩溃而崩溃。因此可以说，对于防御 DoS 攻击，P2P 网络有着先天的优势。

（3）频繁加入和退出

攻击者控制节点频繁地加入或退出系统，使系统重复进行分配节点 ID、更新节点路由表等执行过程，造成超额的数据传输和信息流量，消耗大量的网络带宽，大大降低了系统的效率和性能，甚至导致相应网段因超载而阻断。在这种攻击中，如果恶意节点能够避免大量的数据传送，则攻击效果会非常明显。

针对这种攻击，有效的办法就是让加入或离开网络的节点必须进行数据的传送，这样一个节点频繁的加入和退出网络时，对它自身的资源也会产生很快的消耗。这样就能有效地遏制恶意的攻击。

而且，即使不考虑恶意节点的攻击，对于加入与退出的处理也是每个 P2P 网络所要解决

www.ckook.com

的问题。因为 P2P 网络的主体——普通客户端，是一个很不稳定的群体，他们随时可能会离开网络甚至出现问题而失效。所以在 P2P 网络中，如果节点的加入与退出问题没解决好的话，很可能导致网络被分割甚至瘫痪。

（4）Sybil 攻击

很多大规模的 P2P 网络都面临着来自远程恶意节点或节点失效所带来的安全威胁。为了解决这些问题，许多系统引入了冗余备份机制。

一些系统通过将运算任务或存储任务在多个远程节点上进行备份，来保护数据的完整性，使得即使网络中的某个节点的数据不可用时，网络中仍然会有其他的节点保存了正确的数据以供使用。而另一些系统通过将一个完整的任务或数据进行分割后，分别存储在多个远程节点上，来保护数据的机密性，使得即使某个节点能破解出本地所存储的数据信息，但是由于数据并不完整，故仍无法窃取到全部信息。然而，这两种方法的实现，都必须要求节点能够识别出两个表面并不相关的节点在实际上到底是不是同一个节点。

如果一个节点无法直接获取远程节点的物理信息，那么它只能通过抽象信息，也就是我们所说的节点 ID 来区分远程节点。因此，P2P 系统必须保证每一个节点 ID 都唯一的标识了网络中的一个节点。否则，如果一个节点可对应多个节点 ID 时，很可能出现这样的情况，当一个节点在进行冗余备份时，表面上，它选择了网络中一组 ID 并不相同的节点，而实际上它有可能被恶意节点所欺骗，导致所选择的都是同一个节点，从而破坏了冗余备份的有效性。这种以一个节点来冒认多个节点 ID 的攻击方法就称为 Sybil 攻击。

针对 Sybil 攻击，最有效的方法就是保证网络中的每一个节点都对应一个唯一的节点 ID。有一种解决方案是，要求网络中的节点要为自己的账号付费。由于发起 Sybil 攻击，要求攻击者控制 P2P 网络中一定比例的节点，这样攻击者为了能够发起攻击就必须首先获取网络中一定比例的节点 ID，而其所消耗的资金就会随着网络规模的增长而增长。这种消耗可以大大地减小攻击发起的可能性。另一种解决方案是将节点 ID 与现实生活中的认证相绑定。但由于现实社会认证的复杂性，这种解决方案对认证机构的要求就更高。另外，值得注意的是，当 P2P 网络规模较小时，大部分已知的解决方案都不是很有效。因此，只能要求所有加入网络中的节点都是可信的。而当网络规模很大时，攻击者就很难在网络中控制一定比例的节点来组织攻击，这时，不可信节点的引入并不会给 P2P 网络带来很大的安全问题。

4.2 P2P 应用与安全技术

在 P2P 应用中，使用了大量的安全技术，包括基础的密码学技术、应用层的网络安全技术，此外，通过 P2P 应用还可以解决一些传统网络中很难解决或者无法解决的安全问题。本节将就这些问题进行详细的阐述。

4.2.1 P2P 应用中的密码学技术

P2P 应用中的安全技术，归根结底是对密码学技术的一种实际应用。因此可以认为，P2P 应用中包含了大量的密码学技术。本小节将对这些密码学技术进行简单的介绍，让读者对这

www.ckook.com

些技术有一个基础的认识。

1．对称密码技术

所谓对称密码，即双方使用相同的不公开的秘密，完成对明文数据的加密过程。对称算法有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，反过来也成立。在大多数对称算法中，加/解密密钥是相同的。这些算法也叫秘密密钥算法或单密钥算法，它要求发送者和接收者在安全通信之前，商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都能对消息进行加/解密。只要通信需要保密，密钥就必须保密。

对称算法的加密和解密表示为：

$$E_K(M)=C;D_K(C)=M$$

对称算法可分为两类。一类只对明文中的单个比特（有时对字节）运算的算法称为序列算法或序列密码。另一类是对明文的一组比特进行运算，这些比特组称为分组，相应的算法称为分组算法或分组密码。

好的对称密码系统的全部安全性只与密钥有关，和算法没有任何关系。这就是为什么密钥管理在密码学中如此重要的原因。对称密码技术存在下面的问题。

（1）密钥必须秘密地分配，它们比任何加密的信息更有价值，因为知道了密钥意味着知道了所有信息。

（2）如果密钥被暴露了，那么窃听者就能用该密钥去解密所有传送的信息，也能够假装是几方中的一方，产生虚假信息去愚弄另一方。

（3）假设网络中每对用户使用不同的密钥，那么密钥总数随着用户数的增加迅速增多。n 个用户的网络需要 $n(n-1)/2$ 个密钥。

在 P2P 应用中，很多地方都使用到了对称密码技术，它的主要应用场景是对数据包加密。当通信双方共享密钥后，就可以采用对称密码算法保证双方通信数据的机密性。例如在 Skype 系统中，它的端到端加密通信过程中就使用到了对称算法中的 AES 算法来对数据进行加密。

2．公开密钥密码技术

公开密钥（非对称）密码技术和对称密码技术不同，它并不要求加密和解密的密钥是相同的。公开密钥算法是这样设计的：用作加密的密钥不同于用作解密的密钥，而且解密密钥不能根据加密密钥计算出来（至少在合理假定的长时间内）。之所以叫做公开密钥算法，是因为加密密钥能够公开，即陌生者能用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。在这些系统中，加密密钥叫做公开密钥（简称公钥），解密密钥叫做私人密钥（简称私钥）。

用公钥 K_{public} 加密表示为 $E_{\text{public}}(M)=C$。

用私钥 K_{private} 解密表示为 $D_{K_{\text{private}}}(C)=M$。

但是在做数字签名时恰好相反，消息用私人密钥加密而用公开密钥解密，这时的运算可分别表示为：

$$E_{K_{\text{private}}}(M)=C;D_{K_{\text{public}}}(C)=M$$

在数学上，公开密钥实现的是一个单向陷门函数。加密是容易的，加密指令就是公开密钥，任何人都能加密信息。解密是困难的，它做得非常困难，以致于不知道这个秘密，即使用 Cray 计算机和几百万年的时间都不能解开这个信息。这个秘密或陷门就是私钥。持有这个秘密，解密就和加密一样容易。

www.ckook.com

公开密钥的获取一般是通过公钥证书。它包含了某人的公开密钥，由一个值得信赖的人签发。证书可用来防范用一个密钥替换另一个密钥的攻击。证书在公钥数据库中包含了比公钥更多的数据，包括关于证书持有者姓名、地址等信息，并且通常由证书机关或 CA 进行了签名。通过对证书中的密钥及其有关信息签名，CA 证实有关证书的信息是正确的，且公钥属于证书持有者而非其他人。证书在许多公钥协议中扮演了重要的角色。

而在实际的应用中，公开密钥算法并不能替代对称算法。其原因有以下两个。

（1）公开密钥算法比对称算法慢，对称算法一般比公开密钥算法快 1 000 倍。

（2）公开密钥密码系统对选择明文攻击是脆弱的。如果 $C=E(P)$，当 P 是 N 个可能明文集中的一个明文，那么密码分析者只需要加密所有 N 个可能的明文，并能与 C 比较结果（加密密钥是公开的）。用这种方法，不用恢复解密密钥，就能够确定 P。

一般情况下，公开密钥算法并不用于加密数据，而只用于加密密钥。把公开密钥密码技术用于密钥分配中很好地解决了密钥管理问题。对对称密码而言，数据加密密钥直到使用时才起作用。如果窃听者得到了密钥，那么他就能够解密用这个密钥加密的消息。而当需要对通信加密时，才产生会话密钥，不再需要时就销毁，这将极大地减少会话密钥遭到损害的风险。

在 P2P 系统中，也有很多地方使用到了公开密钥密码技术，它的主要应用场景是协商会话密钥。当通信双方拥有对方的公钥时，就可以通过公开密钥算法进行会话密钥的协商，保证会话密钥的安全性，从而保证了会话的安全性。例如在 Skype 系统中，它的私有会话建立协议中就使用到了公开密钥算法 RSA 算法进行会话密钥的交换。

3．单向散列函数

单向散列函数是现代密码学的中心，其数学定义是把一个可变输入长度串（叫做预映射）转换成固定长度输出串（叫做散列值）的一种函数。单向散列函数是工作在一个方向上的散列函数，从预映射的值很容易计算其散列值，但要产生一个预映射的值使其散列值等于某一个特殊值却是很难的。

单向散列函数是公开的，对处理过程不用保密。它的安全性是它的单向性，其输出不依赖于输入。预映射的值的单个比特的改变，统计地看，将引起散列值中一半的比特改变。已知一个散列值，要找到它的预映射的值，在计算上是不可行的。

消息鉴别码也叫数据鉴别码，它是带有秘密密钥的单向散列函数。散列值是预映射的值和密钥的函数。只有拥有密钥的人才能验证散列值。可以选用散列函数或分组加密算法产生消息鉴别码，也有专用于消息鉴别码的算法。

在 P2P 系统中，单向散列函数的使用是最为频繁的，特别是在 DHT 网络中。大部分的 DHT 网络中都是通过对客户端的 IP 地址、文件的关键字以及文件的内容采用单向散列函数映射到 128bit 的 ID 空间中，构建了 DHT 网络的虚拟地址。而通过单向散列函数的单向性，保证了这些 ID 不可伪造。

4.2.2 P2P 应用中的网络安全技术

在 P2P 应用中，使用了大量的网络安全技术，包括身份认证技术、匿名技术、流量识别技术、安全路由技术以及信任机制等。本小节将就这些技术进行详细的介绍。

1．身份认证技术

身份认证技术就是在信息网络中通过认证系统的验证、核实，来鉴别用户身份真实性和

www.ckook.com

有效性的技术。采用身份认证技术，其目的在于识别合法用户和非法用户，从而阻止非法用户访问系统。认证机制是安全系统的第一道防线，用来确保安全机制的有效性、访问控制的有效性，以及交换或存储数据的真实性等，是避免攻击者假冒、篡改等主动攻击的有效手段。认证系统的目的有两个。第一，信源的识别即验证发信人确实不是冒充的；第二，检验发送信息的完整性，也就是要验证信息在传送过程中是否被篡改、重放或延迟。

认证技术的实现往往依赖于一个安全的认证系统的建立。在安全的认证系统中，发送者通过一个公开信道将消息发送给接收者，接收者不仅要接收消息本身，而且还要验证消息是否来自合法的发送者，是否该消息被篡改过。

由于P2P网络大大弱化了服务器在网络中的作用，使得网络中缺乏一个权威的认证实体，认证问题变得比较困难。在P2P网络中的身份认证问题，可以看成是在一个无中心的环境中如何选择可靠的资源，即如何认证节点，并建立节点之间的信誉问题。

通过前面章节的介绍，可以看到在P2P网络中，恶意节点通过冒充或利用P2P网络的匿名性，可做成形形色色的攻击，并为P2P网络带来极大的安全隐患。可以说，身份认证在P2P网络的安全性上起了举足轻重的作用。

2．匿名技术

匿名问题，主要指的是基于P2P的隐蔽通信与隐私保护问题。利用P2P无中心的特性可以为隐私保护和匿名通信提供新的技术手段。

匿名性和隐私保护在很多应用场景中是非常关键的。在使用现金购物，或是参加无记名投票选举时，人们都希望能够对其他的参与者或者可能存在的窃听者隐藏自己的真实身份。在另外的一些场景中，人们又希望自己在向其他人展示自己身份的同时，阻止其他未授权的人通过通信流分析等手段发现自己的身份，例如为警方检举罪犯的目击证人。事实上，匿名性和隐私保护已经成为了一个现代社会正常运行所不可缺少的一项机制，很多国家已经对隐私权进行了立法保护。

然而在现有的Internet世界中，用户的隐私状况却一直令人堪忧。目前Internet网络协议不支持隐藏通信端地址的功能。能够访问路由节点的攻击者可以监控用户的流量特征，获得IP地址，使用一些跟踪软件甚至可以直接从IP地址追踪到个人用户。SSL之类的加密机制能够防止其他人获得通信的内容，但是这些机制并不能隐藏是谁发送了这些信息。

P2P技术为解决Internet隐私问题开辟了一条新的可行方案。P2P系统要求每个匿名用户同时也是服务器，为其他用户提供匿名服务。这意味着经过一个节点的消息可能是源于该节点，也可能是源于其他节点，很难决定是这两种情况中的哪一种。P2P系统的另一个特点是攻击者不易找到明确的攻击目标，在一个大规模的环境中，任何一次通信都可能包含许多潜在的用户。另外，P2P系统具有较好的可扩展性和柔性，可以在节点之间进行负载均衡，不存在单失效点等优点。

但P2P系统也面临着许多挑战。首先是加入控制（Admission Control），系统很难知道加入的节点是否是恶意节点，是否被攻击者控制等。这样，一个能力强的攻击者，可以向系统中插入大量被其控制的节点，以进行通信流量分析。而对加入系统的节点进行身份认证又与匿名性这一目标相违背。其次是P2P系统的动态性很强，许多节点在网络中的时间并不长，它们频繁地加入和离开系统。当一个节点加入系统时，它需要为系统中其他节点形成匿名路径，这可能会带来一些安全问题。当一个节点离开系统时，该节点所在匿名路径上的那些用

www.ckook.com

户必须等待新的匿名路径的形成。节点加入和离开系统的另一个问题是节点必须知道网络中的其他一些节点。而 P2P 系统不断变化的匿名集又给这一问题增添了困难。最后，P2P 系统中各个节点的性能有差异，尤其是在一个开放的环境中。这导致了一些问题，例如，一个性能差的节点会降低其所在匿名路径的效率，即使匿名路径上其他节点的性能很好。性能差异还可能有利于攻击者进行时间分析，因为攻击者可以从一条路径上的不同的延迟获得一些相关的信息等。

另外，匿名通信技术如果被滥用将导致很多因特网犯罪而无法追究到匿名用户的责任。所以提供强匿名性和隐私保护的 P2P 网络必须以不违反法律为前提。而在匿名与隐私保护和法律监控之间寻找平衡又将带来新的技术挑战。当然，前提是相关的法律法规必须进一步完善。

3．流量识别技术

流量识别是一个不断发展变化的过程。最初的 P2P 流可通过其固定服务器 IP 地址及端口号加以识别，但是很快就出现了大量采用可变端口及伪装端口的分布式 P2P 应用软件。由于 Internet 上存在着众多的 P2P 应用软件，使得 P2P 流量的识别与监控必须采用一种或多种手段协作进行。

（1）IP 地址识别。该方法可以分类出集中式 P2P 中的目录服务器以及混合式 P2P 中的 Super-peer 以及某些恶意的 Peer 用户。

（2）TCP/UDP 端口识别。该方法可以分类出采用固定端口的 P2P 应用，以及提示未知端口的出现。

（3）数据报深层扫描（DPI）。由于 P2P 软件引入动态端口，只能通过扫描高层协议来探知 P2P 数据报。如对于 Kazaa 系统，我们只能深入 HTTP 内部获取 Kazaa 特征代码。对于其他 P2P 应用，有时甚至要通过几个特征代码才能判断其为 P2P 流。通过 DPI 扫描可识别 Gnutella、e-donkey、Kazaa、BitTorrent 4 类主流 P2P 软件。实验证明该方法的识别准确度可以达到 95%，然而 DPI 扫描技术无法识别加密的 P2P 流。

（4）基于会话（Session）的分类。TCP/UDP 端口可能存在于任何一个数据包中，高层协议的特征代码却只能存在于一个会话包的头几个数据报中。因此，当在一个会话包的第一个数据报中发现 P2P 特征代码时，该会话包的其余数据报也就可以判断为 P2P 数据报。有时 P2P 软件甚至使用多个会话包，这就需要系统软件能关联匹配这多个会话包进行 P2P 判定。

（5）双向识别。当某个方向的流（五元组定义的流）被识别为 P2P 流，则其反方向流必然也是 P2P 流。

（6）流统计状态的识别。在 IP 层通过统计流量特征的方式识别 P2P 流。研究表明，P2P 流量具有长时固定连接的特点，因此理论上基于流状态的统计识别方式可以识别一切大规模的 P2P 流量。当然该方法无法精确地判断出该 IP 流采用的 P2P 协议类型。

对于采用可变端口及伪装端口的分布式 P2P 应用软件产生的流量，当前最为有效的识别手段是 DPI 扫描，通过扫描应用层数据报的关键字加以识别。然而，DPI 扫描方式的缺点是无法识别加密的 P2P 流，可预见 P2P 软件的下一发展趋势是加密化。对此，国外一些研究人员进行了有益的尝试，试图通过在传输层考察固定时长内 IP 地址数目与端口号数目的对应关系区分出 P2P 流，依据是 Web 服务通常针对同一 IP 地址同时使用多个端口号进行下载，而 P2P 流则对一个 IP 地址仅使用一个端口号连接。然而，该方案在区分 P2P 流与游戏、视频流等其他数据流时，仅能采取简单的排除法。所以，研究 P2P 流的统计特性（连接时长、连接

www.ckook.com

速率、连接突发度等），利用流统计状态信息准确区分 P2P 流将是下一步 P2P 流量控制研究的主要方向。

4. 安全路由技术

首先需要指出的是本节所提到的安全路由技术主要指的是基于 DHT 结构化 P2P 网络的安全路由技术。

结构化的 P2P 网络为建立大规模、无中心的网络应用（例如分布式存储、计算等）提供了基础。这种 P2P 网络有很多优点，比如只需很少的几跳路由便定位到所查询的节点，而每个节点的本地路由表中只需存储有限的几个节点的路由信息，而且这种 P2P 系统可发展为大规模网络，容错性好，并能提供有效的负载均衡。但现在的结构化 P2P 网络并不安全。即使是一小部分恶意节点也可能阻断正常的消息转发。那么如何保证结构化 P2P 网络的路由安全呢？下面就针对结构化 P2P 网络的路由安全技术进行一些探讨。

安全路由的实现可分为以下 3 个部分。

（1）节点 ID 分配的安全。它指的是节点无法控制分配给自己的 ID 号。否则，攻击者就有机会控制某个给定数据的备份，或者阻断从目标节点发出或目标节点接收到的数据包。

为了实现节点 ID 分配的安全，最典型的一种解决方案就是建立一个中心的、可信的认证实体。CA（Certification Authority）主要负责将为网络中的每个节点分配一个公钥和节点 ID。CA 应保证所分配的节点 ID 是随机选择的，以防止节点伪造节点 ID。同时，CA 还应为 P2P 网络提供一个公钥认证体系，即为网络中的节点之间相互通信建立一个加密与认证的渠道。

许多 P2P 网络中，节点 ID 的分配都与节点的 IP 地址甚至是端口相关。这是因为，在网络中，节点的 IP 地址可以在一段时间内保持不变，在这段时间内，该信息与该节点是唯一对应的。在节点 ID 的分配中引入节点 IP 地址，可进一步帮助 P2P 网络来防范 Sybil 攻击。当然，在节点 ID 分配时，需要防范节点进行 IP 地址的伪造。

（2）路由表保存的安全。路由表安全，应保证在一个正常节点的本地路由表中，恶意节点所占的比例不应大于在整个网络中的恶意节点所占的比例。否则，攻击者只需要控制为数不多的几个恶意节点，就有可能阻断该节点的正常消息传递。

为了实现节点的本地路由表的安全，需要严格控制其路由表中所保存的节点信息。也就是说，当路由表中的某个节点离开网络或失效时，该节点就会被从路由表中删除掉，而同时路由表中应重新引入一个新的节点。这时，对新加入路由表的节点应该采取严格的控制措施。比如，节点在填充本地路由表时，要求只有节点 ID 与某个特定的节点 ID 最相近的节点能够加入本地路由表。

（3）消息转发的安全。保证了节点在查询某个 Key 时，很有可能至少找到保存该 Key 的多个备份节点中的一个节点。

5. 信任机制

虽然 P2P 领域可能看起来是令人激动的和全新的，但在分布式环境中安全计算的要素却还是原样。信任是通过综合以下 3 个标准要素来建立的。

（1）认证：决定某些实体实际上是否就是它们自称的人或东西的过程。在实际中，认证有两种形式。第一种形式涉及在一个网络（如因特网）中对等点向其他对等点认证它们自己。第二种形式涉及一个 P2P 应用的用户向该应用认证它们自己。在一些 P2P 应用中，这两者是一回事。

www.ckook.com

（2）授权：授予一个认证过的实体实施某些行为或访问某些资源的许可的过程。在一个P2P应用中，一个对等点可能被认证为只可以访问另一个对等点的部分资源。

（3）加密：把易理解的信息（明文）转化为一种对未授权的个人和系统来说难理解的形式（密文）的过程。解密是这一过程的逆过程。在一个P2P应用中，加密可以扮演许多角色。加密的一个明显用途就是保护在一个不安全网络（如因特网）中的对等点间流动的信息。这与每个对等点的安全认证结合在一起，确保了交换数据不会在通信中被窃听。如果信息是被数字签名过的或在其中加入了MAC（消息认证码），那么双方还能确定该信息没有被修改过。

在P2P网络中，使用到的信任机制主要包括以下两类。

（1）对等实体的认证和授权

一旦P2P应用发展到一个引人关注的程度，信任和安全问题就出现了。在用户间互相了解的小型应用中，信任和安全很少会成为问题。可是，常规的P2P应用很少会保持这么小的规模。

信任在每个具有一定规模的分布式应用（包括对等应用）中都是一个问题。在一个分布式应用中，信任的等级就是衡量我们确信程度的尺度，即我们正与之通信的人是否是我们以为的那个人，以及我们正访问的资源是否是我们以为的那些。

在实体间都互相了解的小型网络中，要建立信任是很容易的。在小型网络中，实体是建立在互相熟悉的基础上的，信任就可以由在真实世界中运作的同样的社会力量来维持。当一个网络应用扩展到常规的社会力量不再满足要求时，建立信任的难度就会变得大起来。网络的确切大小显然依赖于应用，但当网络中的任何实体不再只期望和认识的实体进行相互作用的时候，上述的扩展过程中的问题就出现了。

这里有两个问题：对等的认证和授权。首先，实体不能再假定其他实体就是它们自己所声称的实体，这是认证问题。其次，实体不能再简单地允许其他实体不加选择地访问它们提供的功能和它们管理的资源，这是授权问题。

无论一个相互作用是在两个以前从没遇见过的实体间的，还是在两个老相识的实体间的，相关实体都必须保证他们间的相互作用是安全的。

（2）共享资源的认证

关于信任，在许多P2P应用中，还有一个决定性的方面没有得到应有的重视。对等认证无疑是重要的，但通常同样重要的是对共享资源的认证——特别是内容。没有对共享内容的完整性和同一性的保证，一个P2P应用就会引入许多安全漏洞的风险。

例如，节点搜索“物种起源”，收到了3个响应回复。响应一是真正由Charles Darwin写的《物种起源》；响应二虽然仍是Charles Darwin所写的，但其中关键的几页被篡改了；响应三则只是引用了《物种起源》的其他作品。有些人认为P2P系统的发展将极大地依赖于是否能够提供一种新技术来确保节点接收到可靠保证的资源。

要确保信任，一个网络应用和它的基础结构必须作出一些保证。首先，实体间的连接必须是安全的。基础结构还必须可以使以下条件成为可能：能够准确地识别其他实体，或至少能够肯定地断言这种识别是不可能的。最后，通过应用来管理或交换的资源也必须满足同样的要求。

4.2.3 利用P2P网络解决安全问题

一个开放的网络，如Internet，主要涉及到的安全问题如下。

www.ckook.com

（1）数据的机密性：防止通信的内容被窃听。

（2）数据的完整性：确保接收方所收到的消息与发送方所发出的消息一致。

（3）隐私性："擦除"同一用户的不同会话信息，防止这些会话信息被关联起来用于流量分析、市场分析等。

（4）认证：验证网络实体的身份。

（5）访问控制：对敏感信息或资源进行严格控制，只允许授权用户的访问。

近年来，许多研究提出了甚至已经应用了 P2P 网络来解决这些棘手的问题。由于 P2P 网络独有的与 C/S 网络的众多不同之处，使得一些曾经困扰 C/S 网络的安全问题，在引入了 P2P 网络后，出现了一些新的解决方案。可以说，在某些特定情况下，C/S 网络通过引入 P2P 网络，可得到一些新的安全解决方案。

其中，P2P 网络最有效的应用领域，也是引起人们最多关注的领域就是隐私性，即匿名性。"E-mail Mixes"也许是 P2P 网络最早被用于提供匿名邮件服务的应用。这种方法通过 E-mail 服务器所组成的网络，成功的将该邮件转发到下一个节点。这样对于发送 E-mail 的用户信息就可实现隐蔽的要求。

另外，P2P 网络可用来隐藏通信流量模式，增加攻击者的工作量。在数据的机密性、认证以及访问控制方面，也有 P2P 网络的应用。本小节就侧重介绍利用 P2P 网络来建立的匿名通信系统，以及基于 P2P 网络防御 DDoS 攻击的解决方案。

4.2.3.1 利用 P2P 网络的匿名通信系统

Internet 已经被越来越多地用于个人和商业通信中，如众多市民使用网络来处理银行事务、购物和纳税等重要的商业交易，信息安全逐渐成为一个潜在的巨大问题。因此信息加密技术成为信息安全的一种重要保障手段。对信息本身的保密已有相对成熟的保护信息的数据加密算法。

但 HTTP、TCP/IP、MIME、FTP 等开放式协议所需的报文头部信息例如源地址、目的地址、报文长度等很难通过加密算法来隐藏。攻击者可以利用协议存在的漏洞发动窃听与流量分析攻击来获取通信双方的地址，进而推断出一些有价值的信息。侦听是指攻击者对通信链路进行非法窃听，获取重要信息。而流量分析是指攻击者可以对链路中信息的时间、速率、源及目的地等信息进行分析，从而窃取通信双方的地址以及相关重要信息。

在 P2P 网络中，由于信息的传输分散在各节点之间进行而无需经过某个集中环节，用户的隐私信息被窃听和泄漏的可能性大大缩小。此外，目前解决 Internet 隐私问题主要采用了中继转发的技术，从而将通信的参与者隐藏在众多的网络实体之中。在传统的一些匿名通信系统中，实现这一机制依赖于某些中继服务器节点。而在 P2P 中，所有参与者都可以提供中继转发的功能，因而大大提高了匿名通信的灵活性和可靠性，能够为用户提供更好的隐私保护。

1．匿名通信的概念

匿名性（Anonymity）是指通信实体的一种状态，即该实体无法从一个实体集合——匿名集（Anonymity Set）中被识别出来。

匿名通信是指通过一定的方法将业务流中的通信关系加以隐藏，使窃听者无法直接获知或推知双方的通信关系或通信的一方。匿名通信的一个重要目的就是隐藏通信双方的身份或通信关系，从而实现对网络用户的个人通信隐私及涉密通信的更好的保护。匿

www.ckook.com

名通信主要研究网络通信中用户的标识信息的保密问题。这一类标识信息包括用户主机的 IP 地址。

2. 典型匿名通信系统

下面将介绍两个典型的匿名通信系统：Chaum 混淆网络和洋葱路由技术。

（1）Chaum 混淆（Mixes）网络

在 1981 年 David Chaum 提出 MIX 网络的概念。Chaum 定义了一个经过多个中间节点转发数据的多级目标路径，为隐蔽接收者，发送者可选定 N 个连续目标，其中之一为真正接收者。窃听者在一段链路上获取真正接收者的概率为 $1/N$，并且中间节点在传送消息时可采取重新排序、延迟和填充手段使获取真正目标的概率更低，从而加大攻击者进行流量分析的难度。MIX 网络结构示意图见图 4-1。

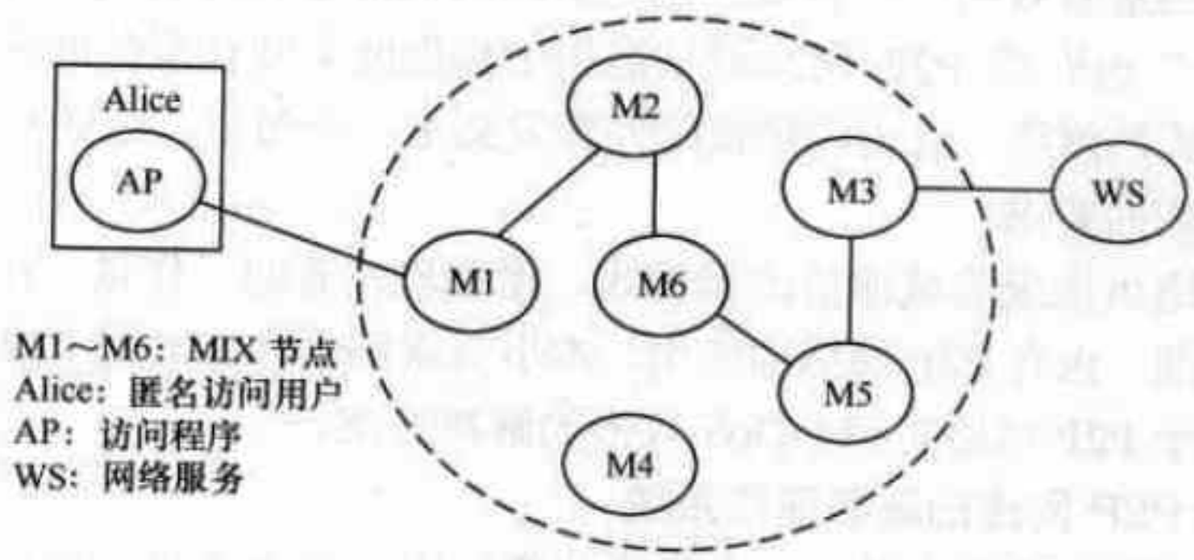


图 4-1　MIX 网络结构示意图

为了和 WS 进行匿名通信，Alice 的 AP 首先选择网络中的某些 MIX（图中选择了 M1、M2、M6、M5、M3），并在其上建立一条匿名通道（图中为实线段）。AP 与 WS 的交互通过通道完成，AP 首先将发送的数据交给 M1，然后由 M1 传递给 M2，依次经过通道中的每个 MIX，最终到达 WS，WS 的响应数据则以相反的方向传递。

为实现上述混淆，必须满足以下两个条件：①所选目标能可靠地完成上述工作且彼此间存在安全通道；②中间节点必须知道整个路径结构才能重新排序。再者，上述延迟虽然可以防止敌手的跟踪，但也可能为攻击者进行攻击提供时间。在 MIX 概念的基础上，发展了一系列比较可行的匿名系统，使基于 Internet 的匿名技术得到了局部应用。

（2）洋葱路由（OR，Onion Routing）技术

洋葱路由技术是美国海军研究实验室的研究者们采用多次混淆的办法，提出的一种新的匿名通信技术，可以在公开的计算机网络中隐蔽网络的结构，对在互连网上进行的跟踪、窃听和流量分析有很强的抵抗作用，通信双方用洋葱包代替通常的 TCP/IP 数据包，利用代理技术实现与目标系统间的连接。这种洋葱路由代理技术的实现过程如图 4-2 所示。

洋葱路由技术的缺陷如下。

① 路径信息可以隐蔽，但代理路由器 X 和 W 易成为攻击的重点，尤其是主机 A 至 X，W 至主机 B 之间是以明文传输，虽然采用应用层加密，但传输层的端口地址和 IP 层的 IP 地址是公开的，缺乏端点的安全机制，一旦攻破 X 或 W 节点，则路由器上的链路状态和节点信息将全部公开了。

② 两路由节点通信数据采用了公钥密码机制，当主机节点增多且数据流量增大时，

www.ckook.com

路由节点解密的时间长，占用大量的 CPU 资源，路由节点的转发效率下降，路由节点将成为网络的瓶颈。

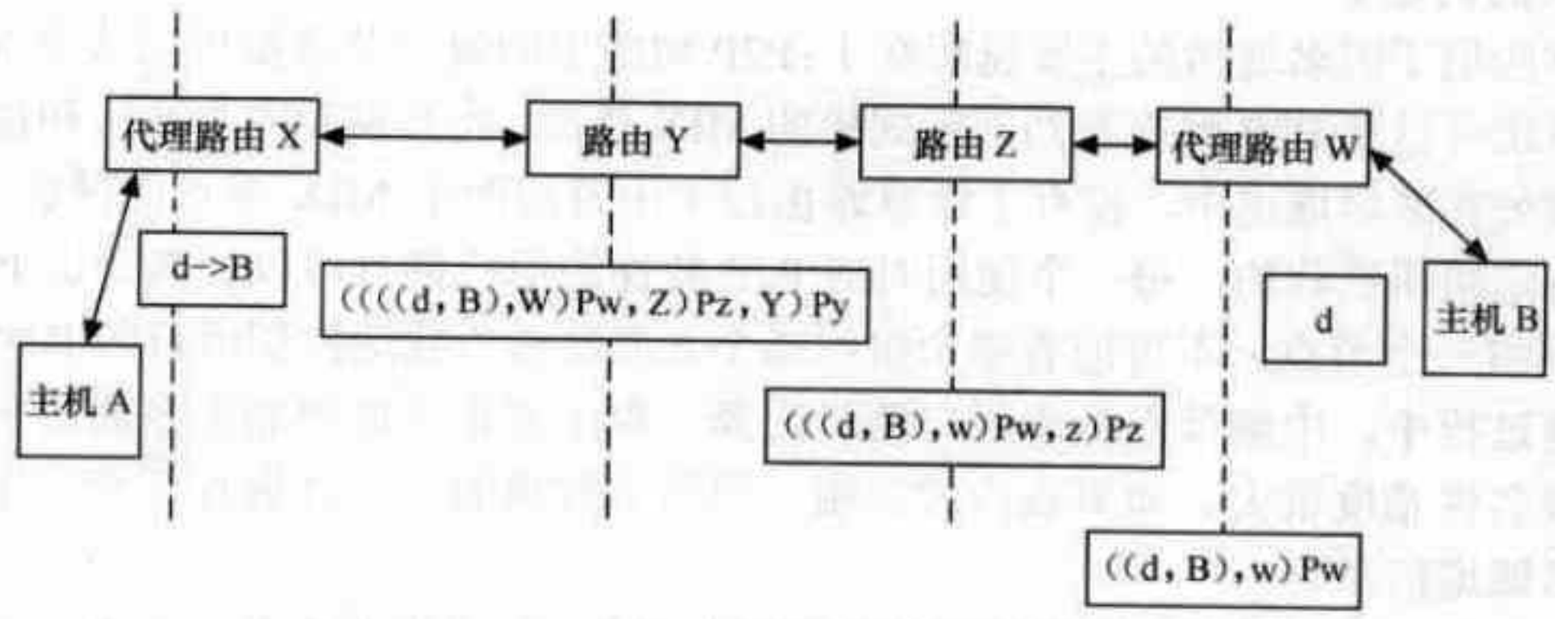


图 4-2 洋葱路由技术示意图

③ 所有节点必须把公私钥、认证码、可信标识、连接的链路信息、存取控制策略、链路当前状态等信息存放在数据库中，当节点数 N 很大时，维护和管理链路的数据库变得庞大，存取路由信息的时延也加大。因此要满足可接受的效率，N 的规模要受到限制。

④ 单一类型的链路效率低，由于葱头路由是面向连接的技术，而实际的应用是各不相同的。有些连接是短暂的，如 HTTP，而有些连接是长时间的，如 RLOGIN 和 Telnet。

3．基于传统 C/S 模式的匿名通信系统

MIX 网络的概念提出之后，出现了一些传统的基于 C/S 结构的 MIX 系统，如 mixmaster 系统、Onion Routing、Freedom、WebMixes、Anonymity Network。它们在用途、规模上各有差异，但具有一个相同的特征——系统中的 MIX 由相对固定的、由特定组织经营的知名站点组成。伴随此特征，它们都具有一些天生的优缺点。

优点可以归纳为可用性、易认证和可靠性 3 个方面。在可用性方面，由于使用公开的站点作为 MIX，且这些站点通常 24h 在线，用户可以很容易地获得全部 MIX 列表，并在其中选择几个，架设匿名通道并通信；在认证方面，因为每个 MIX 都有确定的域名，用户可以通过数字签名技术对其进行认证；在可靠性方面，由于中继节点是基本固定不变的，可以在它们之间协商以达到一定的负载平衡。

当然它们的缺点同样不可忽视。第一，缺乏规模可扩展性。它们的规模局限于经营者拥有的站点数目，这种规模很难让 Internet 中普通用户更广泛地使用匿名通信。第二，既然传统的 MIX 系统中的 MIX 是公开的站点，那么攻击者同样能够很容易找到它们，并对其进行攻击。第三，系统内部攻击。前面提到，攻击者若能明确地控制整条通道，就能够获知用户的一举一动。而传统 MIX 系统在特定组织控制之下，若攻击者出现在经营者之中，则对于用户来说就毫无匿名性可言。

通过对传统 MIX 系统的分析能够得出一个结论：若要广泛应用匿名通信，就必须借助新的 Internet 网络结构，P2P 就是一个很好的选择。

4．基于 P2P 的匿名通信系统

P2P 网络和传统的客户端/服务器（C/S）结构相比较，P2P 网络中节点的关系是平等的（称为对等点 Peer），改变了客户端在 C/S 结构中是被动的角色，使得客户端在 P2P

www.ckook.com

网络中成为同时具有服务器和客户端功能的 Peer；减轻了服务器的负载并且减少了网络拥塞。P2P 网络的体系结构能够解决 C/S 结构中一些由本身的结构引起而无法在本身的框架中解决的问题。

将 P2P 应用于匿名通信的主要思想在于：P2P 网络中的每个节点都可以成为 MIX 节点，匿名通信系统可以从 P2P 网络得到同等规模的 MIX 资源，而且发起匿名通信和接收匿名通信的节点都处在匿名通道中，没有了洋葱路由技术中首尾两个 MIX 节点的薄弱环节。P2P 网络不是特定组织经营的，每一个使用对应 P2P 软件的用户拥有可以加入到该 P2P 网络，成为 P2P 网络一个节点，不可能有哪个组织或个人能随心所欲地控制所有的 P2P 节点。匿名通道构建过程中，中继节点是由用户随机选择，攻击者很难能控制匿名通道中的所有节点，破坏匿名性难度很大。如果攻击者控制了相当多数量的节点并进行联合攻击，也可能会窃取匿名通道信息。

将 P2P 应用于匿名通信能获得较好的规模可扩展性，但 P2P 的非集中性和动态性会带来新的问题。在此对基于 P2P 的 MIX 系统的特性进行分析。

第一，可用性。基于 P2P 的 MIX 系统不再由特定的组织经营，系统中的 MIX 通常不是知名站点，用户需要增加在 P2P 网络内搜索可用 MC 的过程。只要加入到 MIX 网络中的节点达到一定的数目，就能保证每个用户有足够的 MIX 可选择。

第二，可靠性。由于系统中的 MIX 很多，对于 MIX 访问是随机和分散的，不会出现瓶颈，也不会使整个 MIX 系统崩溃。但同时由于 Peer 不是 24h 在线，因此可能出现某条通道中的 MIX 突然失效，这需要软件具有中途改道的能力。

第三，规模可扩展性。P2P 网络天然具有良好的规模可扩展性。

第四，对 MIX 的攻击方面。MIX 都不是知名的站点，对手的攻击只能是盲目的。

第五，内部攻击方面。在基于 P2P 的 MIX 系统中，不可能有哪个组织或个人能随心所欲地控制所有的 MIX 中继节点由用户随机选择，攻击者不可能控制匿名通道中的所有节点，也就不能破坏匿名性。

第六，认证。网络中的 Peer 都是动态加入 MIX 系统的，将基于 P2P 的 MIX 系统建立在对所有的 Peer 进行认证的基础上是不切合实际的。

建立基于 P2P 的匿名系统是可能的，但需要更多的条件，包括大规模的 P2P 网络，快速高效的搜索算法，MIX 的选择最优化，通道断裂重架算法。

4.2.3.2 利用 P2P 网络防御 DDoS 攻击

随着 Internet 的快速发展，全新的沟通交流方式使得我们的生活大大改变。WWW 技术以及 P2P 技术让普通用户可以随时地在网络中查找到他们所需要的消息、文件、多媒体、实时应用等多元化的信息。人们越来越依赖这些技术所带来的便捷，因此对这些服务的稳定性也就提出了越来越高的要求。

而与此同时，这种依赖性也激发了某些别有用心者的破坏欲望。DDoS（Distributed Denial of Service，分布式拒绝服务）攻击就是这样一种攻击。它通过向目标发送大量的有缺陷的数据包来占用网络的处理能力与带宽，以达到干扰、破坏或降低网络服务质量的目的。这种攻击只需要消耗攻击者很少的资源，但对于被攻击网络却会造成严重的破坏，并且需要较长时间的恢复。而 Internet 无论是现在的这种形式还是未来将要演进的形式，都将保持其开放性的特点。这种特点决定了任何单个实体都无法对这种攻击进行有效的管理与控制。

www.ckook.com

为了有效地防御 DDoS 攻击，国内外许多研究者都提出了一定的解决方案。本小节就哥伦比亚大学提出的一种利用 P2P 技术来有效防御 DDoS 攻击的方案进行较为详细的介绍。

1．解决方案

为了保护授权用户与服务器之间的通信不会遭受到 DDoS 攻击，哥伦比亚大学的 Angelos D. Keromytis 教授提出了一个框架：SOS（Secure Overlay Services），如图 4-3 所示。这个框架实际上就是一个 P2P 的网络。它的目的就是保障授权用户与服务器之间的通信能防御 DDoS 攻击，其中授权用户指的是已经通过了网络认证的用户，即合法用户。Angelos D. Keromytis 教授在这个框架中定义了一系列的网络实体，通过这些实体的合作，形成了一个防御 DDoS 攻击的有效屏障。

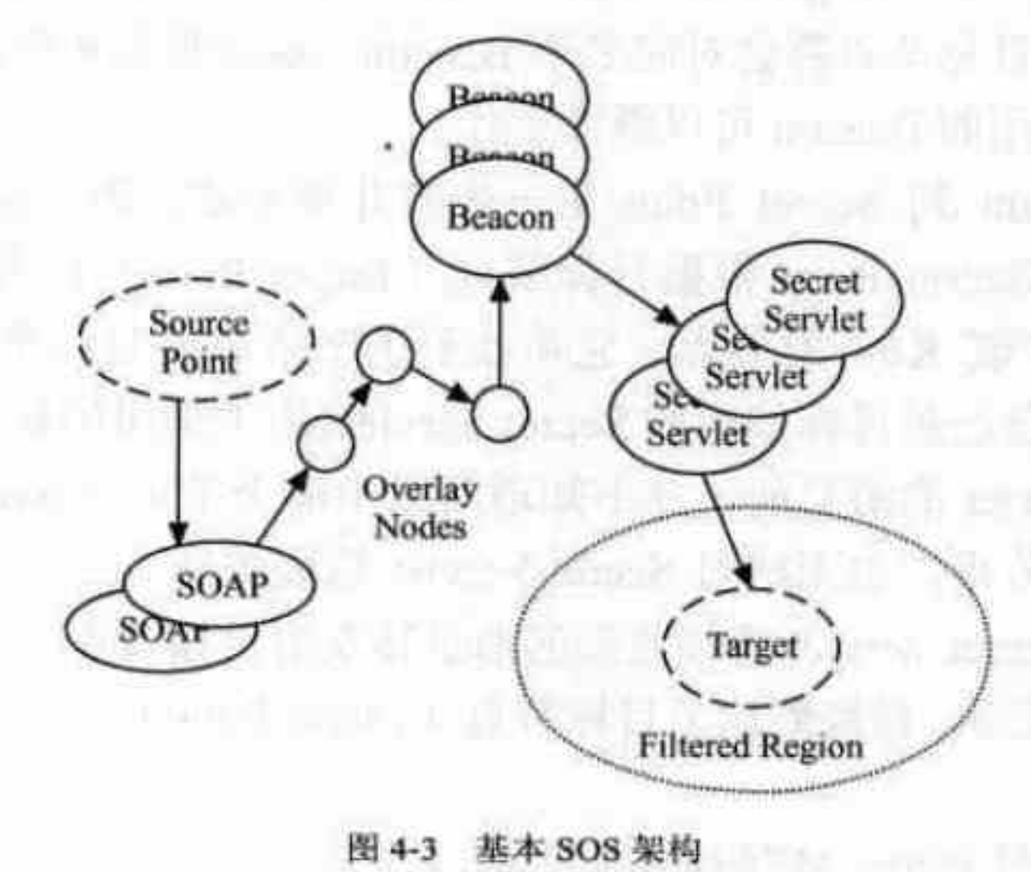


图 4-3 基本 SOS 架构

该系统中主要包括以下各实体。

（1）Targets：目标节点。是 DDoS 攻击者的攻击目标，也是 SOS 框架所要保护的目标。我们希望它能够且仅能够接收到从合法用户处发送的数据。在其周围，由其他的网络节点组成一个“Filtered region”，预先对发送到目标节点的数据包进行过滤处理。之所以将对数据包的过滤放在目标节点的周围，是因为对于攻击者来说，控制一个位于网络中心位置的节点远远比控制一个位于网络边缘的节点困难得多，所以在目标节点附近进行数据包过滤，就比在网络的边缘进行数据包过滤要安全得多。

（2）Secret Servlets：SOS 框架中的一种节点，作为与目标节点建立通信所必经的节点。它们在 P2P 网络中的标识是保密的。

（3）Beacons：SOS 框架中的一种节点，接收到发送至目标节点的数据包后，验证是否合法，并将合法数据包转发给 Secret Servlet。因此，Beacons 应知道 Secret Servlet 在网络中的标识。

（4）Overlay Access Point（OAP）：位于 SOS 框架边缘的一种节点，从经过认证的节点处获取消息。

（5）Source Point：向目标节点发送消息与请求的节点，不一定属于 SOS 的框架之中。这种节点是经过认证的。

（6）Attack Point：攻击节点。试图发起 DDoS 攻击的节点。

该系统的防御机制如下。

（1）在网络接入时，合法客户端选择任意一个接入点（即前面提到的 Access Point）Na，

www.ckook.com

作为连接其和目标节点（Target Point）的第一步，当 Na 同意作其接入节点之前，首先要确认该客户端有连接目标节点的权利，而后，Na 与客户端之间的通信可通过 IPSec 等进行。当 Na 被恶意攻击者所攻击而失效时，客户端可以选择另一个 Na，这样，针对某一个 Na 的 DDoS 攻击不会起太大作用。

（2）目标节点（Target Point）的 IP 地址通过散列运算后得到一个 Key，而保存了这个 Key 的节点就成为该目标节点的 Beacon。这样，接入节点 Na 在接收到消息之后，可根据消息的接收方，即目标节点（Target Point）的 IP 地址算出对应的 Key，再在网络中查找保存了该 Key 的节点，它所找到的节点就是目标节点（Target Point）的 Beacon。由于 P2P 网络的冗余备份机制，通过目标节点（Target Point）的 IP 地址所算出的 Key 会在网络中的多个节点中进行保存，因此，每个目标节点都会对应多个 Beacon。这种机制使得即使某个 Beacon 被恶意节点攻击后，仍有备用的 Beacon 可以继续工作。

（3）由 Beacon Point 到 Secret Point 这一步有几种方法，而最简单的是，与接入节点（Access Point）相同，Secret Point 根据目标节点（Target Point）IP 地址算出对应的 Key，再在网络中查找保存了该 Key 的节点，它所找到的节点就是目标节点（Target Point）的 Beacon，然后通知他们自己是目标节点的 Secret Servlet。由于采用的反向连接的方式，Beacon Point 在收到 Secret Servlet 的消息前，并不知道网络中哪个节点为 Secret Servlet。因此，即使 Beacon 节点为恶意节点，也无法对 Secret Servlet 造成破坏。

（4）最后，再由 Secret servlet 将接收到的消息转发给目标节点。至此从 Source Point 发出的消息经过 SOS 框架后，最终到达了目标节点（Target Point）。

2．安全性分析

对于该系统的安全性分析主要包括以下几点。

（1）当接入节点（Access Point）被恶意攻击者所攻击而失效时，客户端可以选择另一个接入节点（Access Point），这样，针对某一个接入节点（Access Point）的 DDoS 攻击不会起太大作用。

（2）由于 SOS 框架实际上是一个 P2P 网络，因此，其消息路由具有不确定性，从而给 DDoS 攻击者的路由定位造成很大困难。另外，该网络中所有节点的地位都是相同的，因此不会由于某一个节点的失效而影响消息的正常转发。

（3）由于 P2P 网络的备份机制，使得即使某个 Beacon Point 被攻击，也有其他的 Beacon Point 继续工作。

（4）即使当 Secret Servlet 被攻击而无法接收数据时，它仍可以通知目标节点，这样目标节点就可以重新在 SOS 框架中选择新的 Secret Servlet。

（5）即使当攻击者发现了 Secret Servlet，并且控制了它向目标节点（Target Point）发动 DDoS 攻击时，目标节点仍然可以选择新的 Secret Servlet。这样原有的 Secret Servlet 发出的数据包就会被 Filter Region 过滤掉，而无法再继续攻击目标节点（Target Point）。

4.3 P2P 实例系统——Skype 的安全性分析

本节将针对一个 P2P 实例系统 Skype 进行详细的安全性分析，重点包括 Skype 系统的安全机制分析和流量识别分析。

www.ckook.com

4.3.1 Skype 简介

相信很多读者都曾听说过或者使用过 Skype 软件，这是一款网络电话软件。依据 Skype 软件发布公司的定义，Skype 是一个免费软件，采用了最新的 P2P 技术，旨在带给全世界人们购买得起的、高质量的语音通信。它主要具有网络电话、实时传讯、网站浏览、语音视讯、档案传输、搜寻用户等功能，更有突破防火墙限制的通信技术。

由于 Skype 融合了当前热门的两大技术——VoIP 技术和 P2P 技术，它的语音质量、编码技术都远远优于其他同类系统。在 2003 年首次推出之后，Skype 就迅速地占领了 P2P 软件的绝大多数市场，并对 VoIP 软件的市场也形成了惊人的冲击。随着 Skype 潜在用户的增多，对 Skype 软件的安全问题的探讨也日益提上了日程。

作为一款商用软件，Skype 采用了私有协议，对其系统中所传送的所有数据都进行了加密处理，并且在反破解方面下了很大的工夫。因此，虽然 Skype 从 2003 年就开始推出市场，并引起了很多黑客、学者的兴趣，但直到现在，也没有人能完整地讲出 Skype 的协议内容及源码细节。

1．基本概念

以下是 Skype 系统涉及的一些基本概念。

（1）SkypeClient。Skype 软件的客户端，简称为 SC。

（2）SuperNode。超级节点，简称为 SN。在 Skype 中是个非常重要的概念，它的作用就像因特网中的核心路由器的作用。每个 SC 根据其内存、带宽以及上线时间为条件，都有可能被选为 SN，SN 一般是具有开放、可达的 IP 地址的 SC。而每个 SN 都会与数百个左右的 SC 连接。

（3）LoginServer。登录服务器，简称为 LS。它本身虽然不是一个 Skype 节点，但它是 Skype 网络中一个非常重要的实体。登录服务器存储着用户的用户名和密码，负责用户登录时的认证过程，同时它还要负责用户名全局唯一。

（4）HostCache。以下简写为 HC，是一个 SN 的 IP 地址和端口对的列表，这些列表由 SC 建立并经常更新。它是 Skype 操作的最紧要的部分，必须至少有一个有效的接入点（即在线的 Skype 节点的 IP 地址和端口），HC 最多可以拥有 200 个接入点。在旧版本中，HC 被保存在安装主机的注册表如下键值中，HKEY_CURRENT_USER/SOFTWARE/SKYPE/PHONE/LIB/CONNECTION/HOSTCACHE。而在新版本中，HC 被保存在安装主机的如下目录文件中，C：\Documents and Settings\<loggedinuserin O/S>\Application Data\Skype\shared.xml。

（5）BaddyList。好友列表，在旧版本中，Skype 存储它的好友信息在注册表中，好友列表被存放在本机，而不是存放在一种中心服务器中。但在新版本中，这个机制已经改变，好友列表被存放在中心服务器中。这样，如果用户在不同的机器上登录，不需要重新建立好友列表。好友列表经过了数字化签名和加密处理。

（6）Encryption。加密处理，Skype 采用了 256bit 的 AES 加密算法，同时，为了对每个 Skype 呼叫或即时消息进行加密，Skype 采用 1 536～2 048bit 的 RSA 算法来协商对称的 AES 密钥。用户的公钥在登录时被 Skype 服务器确认。

（7）Codecs。编码方式，作为 GlobalIPSound 公司的合作伙伴，可以猜测，Skype 采用了

www.ckook.com

该公司实现的宽带编码技术 iLBC 和 iSAC。这两种编码方式许可频率在 50～8 000Hz 的语音通过。

（8）Port。采用端口，SC 开放了 TCP 和 UDP 监听端口，这些端口值在 SC 的连接对话框设置。在安装的时候，SC 随机选择一个端口号。除此之外，SC 也在 80（HTTP 端口）和 443（HTTPS 端口）打开 TCP 监听端口。

（9）NAT/Firewall。SC 采用了各种 STUN 和 TURN 协议来决定它在哪种类型的 NAT 和防火墙之后。

2．Skype 网络架构

Skype 的网络架构如图 4-4 所示。

Skype 是一种重叠的 P2P 网络结构。这种重叠的网络中包含有两种类型的节点：普通主机（Ordinary host）和超级节点（SN）。普通主机是一个可以发起呼叫和发送即时消息的运行中的 Skype 应用程序。超级节点则是普通主机接入 Skype 网络的端节点。任何配置了公共 IP 地址的主机，如果它同时拥有足够的 CPU、内存和网络带宽都有可能被选为超级节点。普通主机必须要与超级节点建立连接并向登录服务器成功的注册、登录才能接入 Skype 网络。

在 Skype 网络中，除了登录服务器以外再没有处于中心地位的服务器。用户的上线、下线信息的存储和传播以及用户查询请求都是以一种分散的方式进行的。

作为一种基于 P2P 的网络架构，Skype 只有很少量的数据存放在中心服务器上。如图 4-4 所示，Skype 客户端连接到超级节点上，而超级节点互相连接，整个网络作为一个整体，可以看作是用于全球定位的服务器。

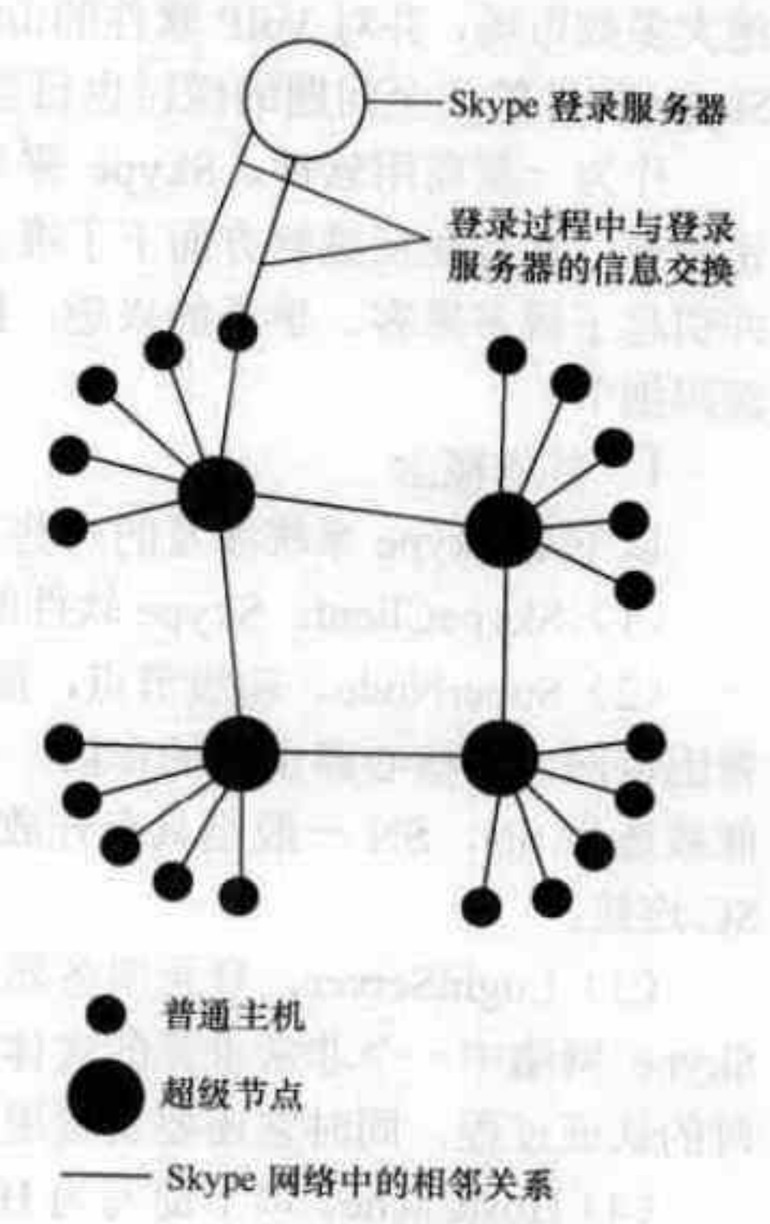


图 4-4　Skype 网络架构

当 Skype 用户第一次登录时，它会连接几个硬编码的 IP 地址，可能是 Skype 或它们的合作者所管理的服务器，这些服务器，提供了一个动态的 SN 列表，SC 会存储下来，并在将来进行连接。通过这种方式，Skype 公司不需要管理许多设备提供地址目录，而只需要提供初始化的几个连接即可。

通常，每个 SC 会建立 4～5 个可信任的连接，并且动态地选择带宽最好或者时延最小的一条路径。

3．Skype 协议分析

本节将具体地针对 Skype 的协议进行分析。

（1）启动过程

当 SC 安装之后第一次运行，它发送 HTTP1.1GET 请求给 Skype 服务器（Skype.com）。这个请求的第一行包含了关键字“installed”。

在随后的启动过程中，SC 只发送一个 HTTP1.1GET 请求给 Skype 服务器（Skype.com）

www.ckook.com

来决定是否需要版本的更新。这个请求的第一行包含了关键字“getlastestversion”。

（2）注册算法和登录过程分析

Skype 注册算法如图 4-5 所示。

登录过程具体说明如下。

登录时，如果 Skype 客户端位于 NAT 或防火墙后，则需要与某个超级节点建立连接，如果连接不成功则无法登录。

登录时，需要与注册服务器交互用户名和密码，向其他节点广播其在线信息，确定其 NAT 和防火墙的类型，搜索有公网 IP 的在线 Skype 客户端，这些节点用来在本节点与超级节点的连接不可用时保持连接。

由于初次登录时，HC 中没有数据，所以初次登录过程与之后的登录过程有所不同。

初次登录开始时，Skype 客户端首先向几个已知的超级节点发送 UDP 包，如果有回应的，则与其建立 TCP 连接，有可能会有几个回应，超级节点会保留至少一个，而断开其他连接。当通过 TCP 与超级节点交换几个数据包之后，可能会请求注册服务器，然后与之建立 TCP 连接。进行登录认证后，关闭该连接。

第一次登录之后，就会存一些可用的主机 IP 和端口在 HC 中。之后的登录过程 Skype 客户端直接从 HC 中选择一台主机，并建立 TCP 连接即可。

（3）用户查找

Skype 用它的全局索引（简称 GI）技术来搜索用户。Skype 声称这种查找是分布式的，并且保证如果一个用户存在，不论以公网还是私网地址登录，只要登录的时间累计 72h 以上就一定能找到它。而随后的深入测试也证明了这一点。

以下从 3 个不同的网络架构来讨论 Skype 的用户查找技术。

网络架构一：通话双方都是公网 IP 地址。

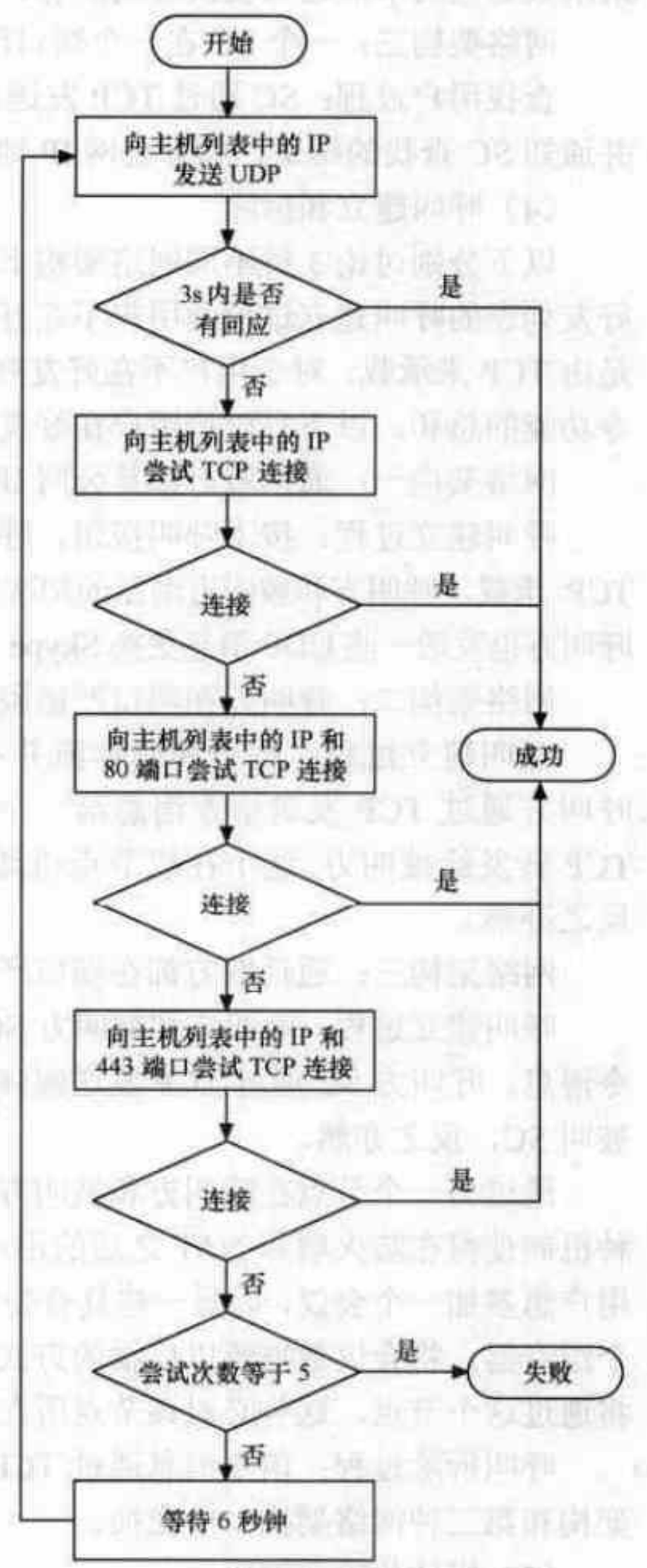


图 4-5 Skype 注册算法

查找用户过程：在输入了 Skype 用户 ID 并按下查找按钮之后，SC 开始查找特定的用户。SC 发送一个 TCP 数据包给它的 SN。SC 在查询用户时，先向 SN 发出请求，SN 告诉 SC 4 个主机的 IP 和端口号，SC 再向这 4 个主机发送 UDP 查询请求数据包，如果未查到，SC 通过 TCP 数据包告诉 SN，则 SN 再告诉 SC 8 个主机的 IP 和端口号，继续查询。如此直到查到为止。查询的消息会被中间机器缓存。从试验中可以看到：当 B 节点查询 A 节点时，第一次用了 8s，但是 B 节点卸载重装后（即本地缓存中没有该查询信息），再次查

www.ckook.com

询 A 的结果比第一次要快，所以证明中间机器确实保留了查询结果。

网络架构二：一个 SC 在一个端口严格受限的 NAT 后。

查找用户过程：和前面不同的是，SC 和 SN 以及一些在 SC 登录的时候回应 SC 的 UDP 请求数据包的节点之间交换查询信息。

网络架构三：一个 SC 在一个端口严格受限的 NAT 和一个 UDP 严格受限的防火墙之后。

查找用户过程：SC 通过 TCP 发送查找请求数据包给它的 SN。然后 SN 执行了查找请求并通知 SC 查找的结果，和有公网 IP 地址的 SC 的查找过程不同，SC 并不联系其他的节点。

（4）呼叫建立和拆除

以下分别讨论 3 种不同网络架构下的呼叫建立过程，对于每种网络架构，又分为用户在好友列表的呼叫建立过程和用户不在好友列表的呼叫建立过程。需要注意的是，呼叫信令总是由 TCP 来承载。对于用户不在好友列表的情况，呼叫建立的过程无非是用户查找和呼叫信令功能的总和。以下只讨论用户在好友列表的呼叫建立过程。

网络架构一：通话双方都是公网 IP 地址、在线，且在彼此的好友列表中。

呼叫建立过程：按下呼叫按钮，呼叫方 SC 和被叫方 SC 建立 TCP 连接，信令消息通过 TCP 承载，呼叫方和被叫方消息的初次交换，必须通过挑战—应答机制。在呼叫建立过程中，呼叫方也发送一些 UDP 消息更换 Skype 节点，这些节点是一些在登录的过程中发现的在线节点。

网络架构二：呼叫方在端口严格限制的 NAT 之后，而被呼叫方在公网 IP 地址之后。

呼叫建立过程：信令和媒体流并不直接在呼叫方和被叫方之间传送。取而代之的是，呼叫方通过 TCP 发送信令消息给一个在线的 Skype 节点，然后这个节点将信令消息通过 TCP 转发给被叫方。这个在线节点也通过 UDP 数据包路由语音数据包从呼叫方到被叫方，反之亦然。

网络架构三：通话双方都在端口严格限制的 NAT 以及 UDP 严格限制的防火墙之后。

呼叫建立过程：呼叫方和被叫方 SC 都通过 TCP 依赖于另一个在线 Skype 节点来交换信令消息。呼叫方 SC 通过 TCP 发送媒体流给在线 Skype 节点，然后由这个节点转发媒体流给被叫 SC，反之亦然。

通过另一个节点在呼叫方和被叫方路由语音数据包，有许多的好处：首先，它提供了一种机制使得在防火墙和 NAT 之后的用户可以互相通话；其次，如果在防火墙和 NAT 之后的用户想参加一个会议，而另一些具有公网 IP 的用户也想参加这个会议，这个节点可以作为一个混音器，将会议数据流以广播的方式发给各个参与方。负面的影响是：有很多的网络流量将通过这个节点，这势必对该节点所在的网络带宽和自身的资源占用造成影响。

呼叫拆除过程：信令消息通过 TCP 在呼叫方、被叫方以及它们相应的 SN（第二种网络架构和第三种网络架构）中交换。

（5）媒体传输和编码

网络架构一：Skype 通话双方都有公网 IP 地址。

媒体流直接通过 UDP 在双方之间传输。用来传输媒体流的 UDP 端口必须通过用户手动配置。语音数据包的长度是 67Byte，这是 UDP 负载的长度。

网络架构二：呼叫方或者被叫方或者两者都在端口严格受限的 NAT。

它们通过另一个在线的 Skype 节点利用 UDP 发送语音流。这个节点的主要作用是：代理语音流并且转发语音流从呼叫方到被叫方，反之亦然。语音数据包的长度仍然是 67Byte，这

www.ckook.com

是 UDP 负载的长度。

网络架构三：呼叫双方都在端口严格受限的 NAT 以及 UDP 严格受限的防火墙之后。

呼叫双方通过另一个在线的 Skype 节点利用 TCP 发送和接收语音数据流。TCP 负载的长度为 69Byte。

综上所述，Skype 协议看起来尽量使用 UDP 来承载语音流。

静音抑制。Skype 不支持静音抑制。即使通话双方不再通话，它们之间仍然有静音数据包交换。理由是：首先，它保持了和 NAT 的 UDP 绑定；其次，这些数据包可以作为一些背景音存在。在语音流采用 TCP 承载的情况下，静音数据包仍然传送。

呼叫挂起。Skype 许可节点挂起一个呼叫。

编码率范围。经测试表明，Skype 编码许可通过的编码率范围为 50～8 000Hz。

（6）上线以及存活通知

上线通知。在建立 Skype 账号时，SN 会自动建立使用者清单资料。当登录 Skype 时，SN 会自动更新目前线上使用者清单，而其他使用者在登录 Skype 时，则能得知谁已经上线。

存活通知。SC 每隔 60s，向 SN 发送信息，从而确定其是否还在线。

4.3.2 Skype 安全机制分析

由于 Skype 采用了一套加密的私有协议，到目前为止，还没有任何研究机构或个人能够完全理解其运行的细节问题。可以说 Skype 是目前众多的 P2P 软件中，对于安全方面考虑得最为全面的系统之一。虽然也有一些 Skype 的漏洞被公布出来，但是总体而言，攻击者还无法针对 Skype 软件进行有效的攻击，甚至连运营商也无法非常准确地识别出 Internet 网络中的 Skype 流量。如果 Skype 真的能够被证明是安全的，这无疑将为 P2P 网络安全技术的发展带来一片新的天地。

那么 Skype 到底是如何实现其通信机制的呢？这种机制又是否安全呢？这些问题引起了许多人的关注，也有很多研究者已经投入到了 Skype 的研究中，并有所收获。下面本节就对 Skype 的安全性进行分析，希望能够引发读者进一步的思考。

1．Skype 所面临的安全威胁

值得指出的是，Skype 的安全从以下几个方面分析是复杂的。首先，Skype 通话的安全取决于多个因素，包括了 Skype 运行其上的计算机的安全，以及通过 Skype 语音流的网络的安全；其次，因为 Skype 协议是专有的和秘密的，所以可以获取的唯一信息来源只有该公司对其的安全陈述以及对该软件的反汇编技术；再次，因为 Skype 采用了 P2P 架构，Skype 通话的安全可能被网络中的第三方所影响；最后，因为 Skype 程序在运行时能够不断地更新自己，Skype 系统的安全可能在未收到任何提示的情况下动态更新。

众所周知，安全不是一个可以被独立加以分析的抽象标准。所以，评估 Skype 的安全必须从考虑特定的安全威胁入手，然后决定 Skype 的设计和操作是否能避免这些安全威胁。

针对 Skype 安全问题主要来自于对 Skype 客户端的威胁，以下分别加以具体说明。

（1）文件传输劫持。为了截取文件，攻击者需要有访问进行文件传输的一方或双方主机的权限。因为文件传输采用了链路加密的技术，所以即使在线截获了文件传输的数据包，也

www.ckook.com

不一定能够破译。尽管如此，攻击者仍然可以窃取最终存储在本地硬盘的传输文件。

（2）即时消息劫持。为了截取即时消息，攻击者同样需要有访问进行文件传输的一方或双方主机的权限，或者其他能够获取 Skype 应用程序客户端界面的方式，因为如果能够远程查看和控制 Skype 应用程序客户端界面，在某种意义上说就相当于劫持了即时消息。和文件传输一样，即时消息在传输的时候不大可能被劫持。但是如果即时消息被显示在桌面或存储在如下的历史即时消息存放目录，C:\Documents and Settings\<loggedinuserinO/S> \Application Data\Skype\<Skype applicationUsername>\IMHistory，那么攻击者就可以实现即时消息的劫持了。安装在靶机的抓取屏幕和键盘记录软件可以被用来完成这一功能，从而使得 Skype 的即时消息功能的安全性能大打折扣。

（3）语音窃听。窃听语音比劫持文件和即时消息更为复杂。语音的实时加密技术往往使得攻击者无功而返。但是，通过分析，Skype 系统可能对加密密钥做如下 3 种处理：①它可能在语音信道中传输加密密钥；②它可能采用其他的密钥来加密这个密钥，然后传输；③它可能存储这一密钥到用户的硬盘空间中。基于以上的分析，攻击者或许可以通过一些反向工程技术获悉 Skype 软件采用的加密算法、整个加密流程乃至加密密钥，然后在这个基础上对劫持的加密语音信息进行攻击。

此外，虽然 Skype 的设计者一再强调由于加密的存在，这种攻击是不可能的。但是从一些其他的加密协议看，任何系统都不可能完美无瑕，都或多或少存在着设计上的缺陷，如果攻击者可以发现这些设计缺陷，那么相应的攻击也就水到渠成了。

2．Skype 所采用的反跟踪技术

为了防止恶意攻击者采用反向工程的方法对软件进行破解，Skype 采用了许多反跟踪技术，所以当使用编译器进行跟踪时，程序总会抛出一系列异常。如果要对其进行反向工程的研究，必须首先研究 Skype 采用了哪些反跟踪技术，找出有效的解决办法，才能深入到对该软件内部的语音流加密机制的处理分析。以下我们对 Skype 可能采用的反跟踪技术进行总结。

（1）检测跟踪法。当解密者利用各种跟踪调试软件对加密系统分析执行时，势必会造成许多与正常执行加密系统不一致的地方，如运行环境、中断入口和时间差异等。这种方法是在反跟踪技术中对不一致的地方采取一定的措施，可以起到保护加密系统的目的。

Skype 中就采用了检测跟踪法，在程序启动时，多次对程序的运行环境进行判断，如果计算机中安装了类似 SoftICE 一类的反编译软件，则 Skype 就不会运行。

（2）抑制跟踪中断。Debug 的 T 和 G 命令分别要运行系统的单步中断和断点中断服务程序，在系统向量表中这两个中断的中断向量分别为 1 和 3，中断服务程序的入口地址分别存放在 0000:0004 和 0000:000C 起始的 4Byte 中。因此，当这些单元中的内容被修改后，T 和 G 命令将无法正常执行。

在对 Skype 的反汇编中，可以在程序中下断点进行调试，但是在程序抛出异常后，查看其异常链时，发现无法下断点，所以不排除程序中部分使用抑制跟踪中断。

（3）对程序分块加密执行。为了防止加密程序被反汇编，加密程序最好以分块的密文形式装入内存，在执行时由上一块加密程序对其进行译码，而且在某一块执行结束后必须立即对它进行清除，这样在任何时刻内不可能从内存中得到完整的解密程序代码。这种方法除了能防止反汇编外还可以使解密者无法设置断点，从而从一个侧面来防止动态跟踪。

www.ckook.com

Skype 中使用了该方法。在对其跟踪中发现，程序初始时，首先要对其中部分代码进行解密替换工作。要想下断点，必须首先搞清楚程序对代码替换的起始位置，否则只能单步调试。除此之外，还有一段隐代码，即把一段代码隐藏到了数据段中，如果使用静态反汇编就很难发现。而这段代码又是非常重要的一部分，它对后面的执行产生了较大影响。

（4）程序自生成技术。程序的自生成是指在程序的运行过程中，利用上面的程序来生成将要执行的指令代码，并在程序中设置各种反跟踪措施的技术。这样可以使得反汇编的指令并非是将要执行的指令代码，同时还可以隐蔽关键指令代码，但由于实现代价较高，一般只对某些关键指令适用。

这种方法在 Skype 中也被使用。在跟踪中能看到，在运行过程中，前一段指令执行后，会自动更改替换后续的指令。但这种行为是否是因为程序发现被跟踪而导致的，目前尚不清楚。

（5）迷惑技术。为了迷惑和拖垮解密者，可以在加密系统中多多设置专门针对解密者的“陷阱”，这样既可以消耗解密者的时间和精力，还可以消磨解密者的斗志和毅力。

设置大循环：程序越简单，就越易读，跟踪也就越方便，鉴于这个原因，在加密系统中设置大循环，可以在精力上消耗解密者，延长跟踪破译加密系统的时间。这种反跟踪技术已经被广泛应用，而且取得了较好的效果。它的具体实现方法是，在加密程序中设置多重循环，并使上一层循环启动下一层循环，下一层循环启动下下一层循环，如此循环，而且还可以频繁地调用子程序，还要保证不能有一层循环被遗漏不执行。

废指令法：在加密程序中设置适当的无用程序段，而且在这其中设置如大循环等程序，这种方法在反跟踪技术中被称为废指令法。要实现废指令法有三点要保证：①废指令要精心组织安排，不要让解密者识破机关，这是废指令法应具备的基本点，因为它的目的是诱导解密者去研究破解自身，并在破解过程中拖垮解密者，所以废指令法本身的伪装十分重要；②所用的废指令应大量选用用户生疏的指令或 DOS 内部功能的调用，以最大程度地消耗解密者的精力和破译时间；③要确保不实现任何功能的废指令段不能被逾越，这是废指令法要注意的一个重要问题，因为它如果能被轻易逾越，那么就说明加密系统所采取的废指令法是失败的反跟踪技术。

这两种方法，在 Skype 中用到的较多。

（6）利用 SEH 链。在对 Skype 的跟踪中，发现程序总是抛出异常中止运行，但对 SEH 链下断点却无法拦截，这样就很有可能是程序进行了一些异常处理。

除此之外，还有很多程序反跟踪技术，如设置堆栈指针法、对程序段校验法、指令流队列法、封锁键盘输入以及设置显示器的显示性能等。这些技术在分析的过程中暂时未发现，但是不排除 Skype 采用的可能性。

综上所述，虽然在理论上，采用方向工程的思路具有较大的可行性，但是由于 Skype 在反跟踪方面做了较多有效的处理，使得对其进行反向工程的工作量比较大，需要花费较多的时间与精力。

3．Skype 的身份验证

为解决用户身份验证的问题，Skype 网络中引入了中心注册服务器，当 Skype 客户端第一次登入 Skype 网络中时，需要向中心服务器提交注册信息，以获取登录的用户名及密码。在以后登录时，需要通过向网络提供用户名与密码，进行身份验证。通过这次验证所产生的数字签名就是 Skype 认证机制的基础。当会话开始时，Skype 节点也需要这项来对它们的通信方进行身份验证。

www.ckook.com

有论文对 Skype 注册及身份验证的原理进行了详细的分析，简述如下：

核心的组成部分是 Skype 中心服务器的私钥 S_s，其相应的公钥 V_s，以及在 Skype 客户端所保存的对这对密钥的标识。

当新的用户注册时，他可以选择一个用户名 A 和密码 P_A。这个用户的 Skype 客户端会生成一对 RSA 算法密钥，私钥 S_A 和公钥 V_A，以及密码的散列值 $H(P_A)$，存储在客户端本地。下面，客户端就可以与中心服务器建立一个通过 256bit AES 算法加密的会话。会话的密钥是通过客户端本地系统所生成的一个随机数。在这个过程中，客户端也会对服务器的身份进行验证。通过该会话，客户端向服务器传送了 A、$H(P_A)$ 和 V_A。

服务器会验证 A 是否唯一或是否符合命名规则。如果符合要求的话，服务器就会将 $(A, H(H(P_A))$ 存入数据库，并为 A 形成认证标识 IC_A，然后将其返回给客户端。

4. 加密的私有会话建立协议

Skype 采用了不公开的私有协议建立会话，该协议的目的是防止重放攻击、验证节点身份以及让通信双方协商会话密钥。之后，通信双方就可以使用它们已经协商好的会话密钥来进行保密通信。

在该密钥交换协议中，通信双方处于平等的地位，都需要接受对方的验证。为了防止重放攻击，通信双方采用挑战—应答的方式向对方发送 64bit 的随机数，收到该随机数的一方需要对该随机数进行修改，并用自己的私钥进行签名，最后将结果返回给对方。这样就完成了整个认证过程。

Skype 宣称它的系统中引入了 RSA 加密机制作为其密钥交换算法。但是，Skype 并没有公开其所使用的密钥交换算法的具体实现。一个较差的实现很可能使得整个系统处于一种危险的境地之中。因此，Skype 密钥交换协议的安全与否目前尚无定论。

如果 A 和 B 两个节点第一次建立通信，就需要建立一个新的会话以及属于该会话的 256bit 的会话密钥，SK_{AB}。只要 A 和 B 之间还有通信流量，那么这个会话就不会结束。即使会话结束，SK_{AB} 也会一直存在于内存中，直到 Skype 客户端关闭为止。

5. 端到端加密

在数据的传输过程中，Skype 采用了端到端加密的方式，使得中间节点无法对会话中的数据进行窃听或篡改。在会话中所有的数据包都会通过 256bit 的 AES 算法进行加密。

4.3.3 Skype 流量识别

随着 Skype 市场的快速发展，越来越多的人开始关注这一特殊的 VoIP 软件，针对 Skype 的流量识别研究也越来越多。下面就针对当前的一些研究进行简单的介绍。

在前面的章节中曾介绍过 P2P 软件的流量识别技术，对 Skype 的流量识别属于其中的一个特例。本节的讨论主要就端口分析和消息流分析来展开。

1. 端口分析

Skype 客户端在通信时同时使用了 UDP 和 TCP。但是每个 Skype 客户端在通信时所使用的 TCP 和 UDP 端口都是可变的，如果用户没有特别的指定，则 Skype 客户端会随机选择一个端口作为其通信的端口。但是仍然有一个特殊的端口是值得注意的，即端口 33033。当 Skype 客户端第一次安装使用时，在客户端本地保存了一些固定的 Skype 网络中的超级节点信息，

www.docin.com
www.ckook.com

Skype 客户端如果想要登录到网络中，必须通过这些超级节点。而这些超级节点的通信端口就是 33033。因此，初次登录时，Skype 客户端总会发送出一系列的 UDP 数据包到几个特殊 IP 的 33033 端口。

需要指出的是，虽然 Skype 客户端会建立 TCP 和 UDP 通信，但是 UDP 通信并不是必须的。当 UDP 通信被阻止时，Skype 客户端仍然能够顺利的登录到网络中。

Skype 客户端总是试图通过高于 1024 的端口来与其他的主机建立 TCP 连接。但如果防火墙禁止了这些端口的通信，则 Skype 会尝试 443 端口，如果再失败，则它会尝试通过 80 端口来初始化连接。

2．消息流分析

Skype 客户端在登录后，会进行 3 个行为：UDP 探测、TCP 握手和 TCP 认证。每一个行为都有其独有的特征。下面将对这 3 个行为逐个进行分析。

（1）UDP 探测

在这期间，Skype 客户端会向某个超级节点发送两个请求，并接收到该超级节点的两次回应。顺序及方向如图 4-6 所示。

由于 U1 是发起通信的数据包，因此该数据包的前两个字节为本次会话的会话标识，同样的会话标识依然出现在 U2、U3 中。

在本次会话中，SN 与 SC 进行了 IP 地址信息的交互。U2 的第 4 到第 7 个字节包含了 SC 的 IP 地址，而 U3 的第 9～12 个字节包含了 SN 的 IP 地址。

（2）与 SN 的 TCP 握手

在 UDP 探测后，Skype 客户端会发现一个或几个可用的 SN，并与其中的一个 SN 建立 TCP 连接。连接的端口号与之前的 UDP 连接的端口号相同。

如果 TCP 连接断开或超时，Skype 客户端会重新选择一个之前 UDP 探测出的可用的 SN，并与之建立 TCP 连接。

如果 TCP 连接建立成功，Skype 客户端和 SC 就可以交换信息。

一般情况下，该连接会保持到 Skype 客户端退出网络，或 SN 无法提供正常服务为止。当一个 TCP 连接断开后，Skype 客户端会以同样的方式再寻找下一个 SN，并与之建立 TCP 连接。

（3）TCP 认证

在 TCP 认证过程中，在 SC 与登录服务器（LS）之间一般会有 4 个数据包，如图 4-7 所示。L1 和 L2 的长度为 5 字节，L3、L4 的长度会变化。

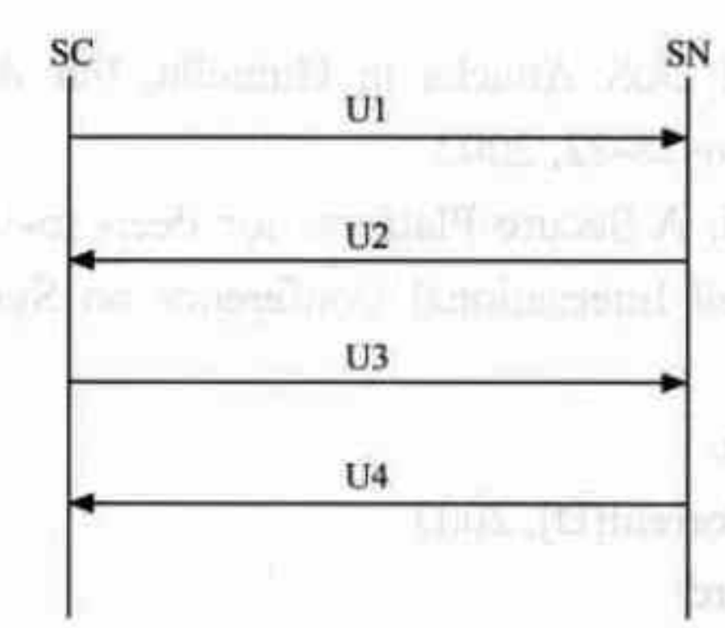


图 4-6　UDP 探测数据包

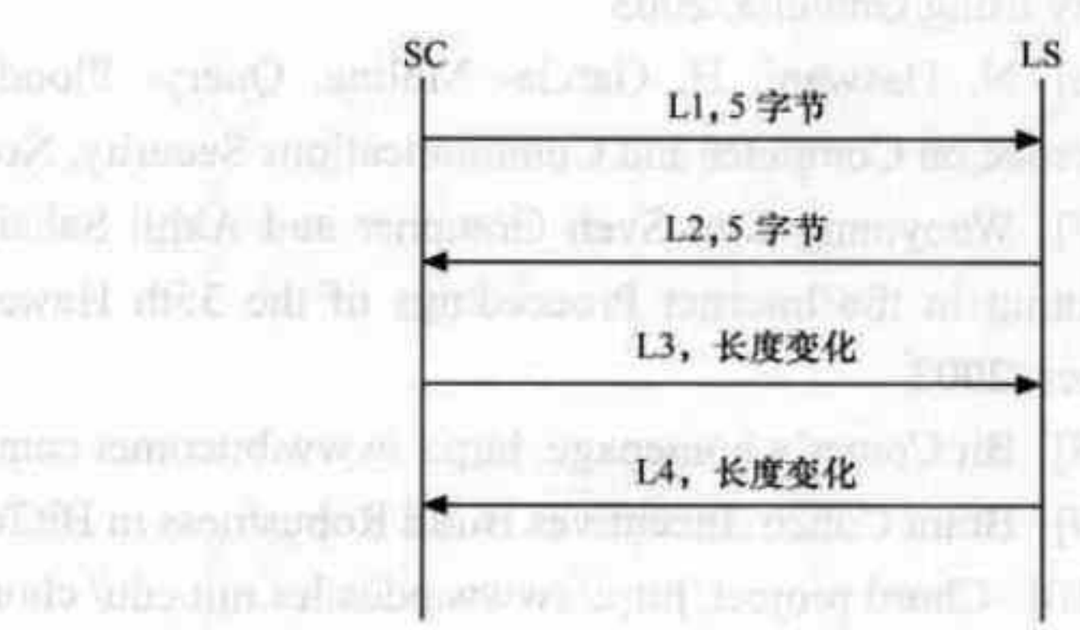


图 4-7　TCP 认证

www.docin.com

当然，有的 Skype 客户端保留了用户的登录信息，在下一次启动时，就不用用户再次输入登录的用户名、密码。这种情况下，用户的登录信息很可能保存在 C:\Documents and Settings\<username\Application Data\Skype \user\ config.xml 中。这样，SC 就不需要再联系登录服务器，因此 L1～L4 就不会再出现。

4.4 本章总结

本章着重介绍了 P2P 网络与信息安全，主要从 P2P 应用面临的安全问题、P2P 应用与安全技术等两个方面进行了分析，最后以 Skype 为具体实例分析了其所采用的安全机制。

希望通过本章的学习使读者对 P2P 网络的安全问题有一个较为全面的认识，并了解国内外在这个领域的一些研究动向。

4.5 练习题

（1）P2P 网络所涉及到的安全问题主要有哪些？

（2）P2P 网络与普通的 C/S 网络的哪些区别使得其在安全方面具有独特的优势与缺陷？

（3）P2P 网络中的路由安全应如何保障？

（4）用什么有效的方法来抑制 P2P 网络病毒与蠕虫的传播？

（5）Skype 所采用的最有效的安全机制有哪些？

参 考 文 献

[1] Napster' s homepage. http:/ /www.napster.com/

[2] Gnutella' s homepage. http:/ /www.gnutella.com/

[3] R. Schollmeier. A Definition of Peer- to- Peer Networking for the Classification of P2P Architectures and Applications [J]. IEEE, 2001

[4] Jonaslund. Authentication in peer- to- peer systems[D] ,2002

[5] Dimitri DeFigueiredo, AntonioGarcia, Bill Kramer. Analysis of Peer- to- Peer Network Security using Gnutella, 2003

[6] N. Daswani, H. Garcia- Molina. Query- Flood DoS Attacks in Gnutella, 9th ACM Conference on Computer and Communications Security, Nov 18-22, 2002

[7] Wooyoung Kim Sven Graupner and Akhil Sahai. A Secure Platform for Peer- to- Peer Computing in the Internet Proceedings of the 35th Hawaii International Conference on System Sciences, 2002

[8] Bit Comet' s homepage. http:/ /www.bitcomet.com/

[9] Bram Cohen. Incentives Build Robustness in BitTorrent[D], 2003

[10] Chord project. http:/ /www.pdos.lcs.mit.edu/ chord/

[11] 李旭华，叶飞跃，蒙德龙．P2P 网络中基于代理合作的匿名传输机制．计算机应用，

www.ckook.com

2006（1）

[12] 朱娜斐，陈松乔，睦鸿飞，陈建二．匿名通信概览．计算机应用，2005（11）

[13] Bruce Schneier. Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, 1996

[14] Salman A Baset，Henning Schulzrinne.An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol，IEEE Infocom 2006

www.docin.com

CHAPTER 5

第 5 章　P2P 应用的相关技术

在学习了本章后，希望读者能够掌握以下几点：

（1）穿越 NAT 实现 P2P 通信；

（2）P2P 在 IMS 中的应用；

（3）P2P 与 VoIP 技术的结合。

5.1　P2P 与网络穿越

5.1.1　NAT 网络概念

1．NAT 产生的背景

NAT 概念产生的一个主要原因就是网络规模的超高速增长，如表 5-1 所示。

www.docin.com

表 5-1　　网络规模的增长

网络发展事件	网络增长规模
1983 年 1 月 1 日 TCP/IP 协议簇成为 ARPANET 上唯一的正式协议	ARPANET 上连接的网络，机器和用户快速增长，特别是 NSFNET 和 ARPANET 互连以后，数目更以指数增长。很多的地区网络开始加入，并且开始与加拿大、欧洲和太平洋地区的网络连接
20 世纪 80 年代中期， Internet 出现	网络规模增长保持指数级
20 世纪 90 年代，高速发展	到了 1990 年已经拥有了 3000 个网络和 20 万台计算机。1992 年，已有 100 万台主机与 Internet 相通。1995 年已经有了多条主干网，数百个中级（地区）网，上万个 LAN，数百万主机和上千万用户，规模大约每年翻一番

因特网业务的发展需要它能够提供大量 IP 地址来满足企业和个人的访问需求。但是人们开始制定网络协议的时候，没有考虑到网络的规模能够增长得如此迅速，所以造成了现有网络协议 IP 地址的短缺。人们提出了大量的解决方案，例如提出了 IPv6 协议来扩大 IP 地址的大小，改变 IP 地址结构，甚至提出了废弃 IP 地址。不幸的是，这些方案要么改变路由器的结构，要么改变 PC 结构或者同时改变这两者，所以这些方案不能够马上

www.ckook.com

实施。一种新的解决方案——地址复用产生了。

这种解决方案是基于这样的事实：在任意给定的时刻，私网域中只有很小比例的主机和公网中的主机进行通信。基于这种思想，在私网内部没有必要为每台主机都分配一个公网的 IP 地址。这样整个网络就被分成了私网域和公网域。在私网域内部，主机之间的通信地址完全可以使用可以重复利用的私网地址。如果私网内部的主机需要和公网的主机进行通信，只需要动态地把 NAT 设备的公网 IP 地址分配给需要和公网进行通信的主机，NAT 作为中间转发设备，起到桥接作用，实现二者之间的通信。

1993 年由 Paul Francis 和 Tony Eng 在 Computer Communication Review（CCR，计算机通信评论）杂志发表了一篇关于地址复用的论文，在这篇论文中首次提到了地址复用的思想和实现地址复用的 NAT 设备。IETF（Internet Engineering Task Force，因特网工程任务组）在 1994 年 5 月提出了 NAT 协议，当时主要是为了解决网络地址短缺的问题；IETF 又在 1996 年 2 月提出了私有网络的概念。在 1999 年 8 月 IETF 提出了 NAT 中使用的相关术语。目前 NAT 设备已经广泛应用于各级网络之中。

2．NAT 概述

NAT 的全称是 Network Address Translation，可翻译为网络地址转换或者网络地址翻译。NAT 是在 IPv4 地址日益短缺和路由表不断增大的情况下产生的，部署 NAT 的主要目的就是为了能够重用 IP 地址。Windows 操作系统中用于多台主机共享上网的“Internet 连接共享”使用的就是 NAT 技术，还有许多因特网网关设备也都使用 NAT。

NAT 技术通过将局域网上的主机地址映射为 Internet 上的有效 IP 地址，从而实现了网络地址的复用。NAT 技术不仅隐藏了内部网络地址信息，使外界无法直接访问内部网络设备，保护内部网络，同时也暂时解决了 IPv4 地址不足的问题。NAT 允许任何内部主机访问 Internet 上的任何服务。NAT 网络示意图如图 5-1 所示。

因特网地址分配组织规定以下的 3 个网络地址保留作私有地址，见表 5-2。

www.docin.com

表 5-2　　私有 IP 地址空间

私有 IP 地址空间	A 类	10.0.0.0～10.255.255.255 （/8 前缀或 255.0.0.0 子网）
	B 类	172.16.0.0～172.31.255.255 （/16 前缀或 255.255.0.0 子网）
	C 类	192.168.0.0～192.168.255.255 （/24 前缀或 255.255.255.0 子网）

这 3 个区间的网络地址不会在因特网上被分配，但可以在局域网内部重复使用。不同的子网，它们的内部地址可以相同。当内部网络主机访问因特网或与外部网络的主机通信时，需要用到地址转换。

如图 5-2 所示，当内部网络要访问外部网络时，就通过 NAT 将自己的 IP 地址和端口号转换为外部的 IP 地址和端口号；当外部主机要访问内部网络时，可以通过配置内部网络中特定的应用服务器和 NAT 转换来访问内部服务。

注意：这里只是说配置特定的服务，比如 HTTP 和 Telnet。因为这些应用的端口是默认的，配置的时候可以将公网端口对应到这些熟知的应用端口。但是这并不适用于随机的端到端直连。

www.ckook.com

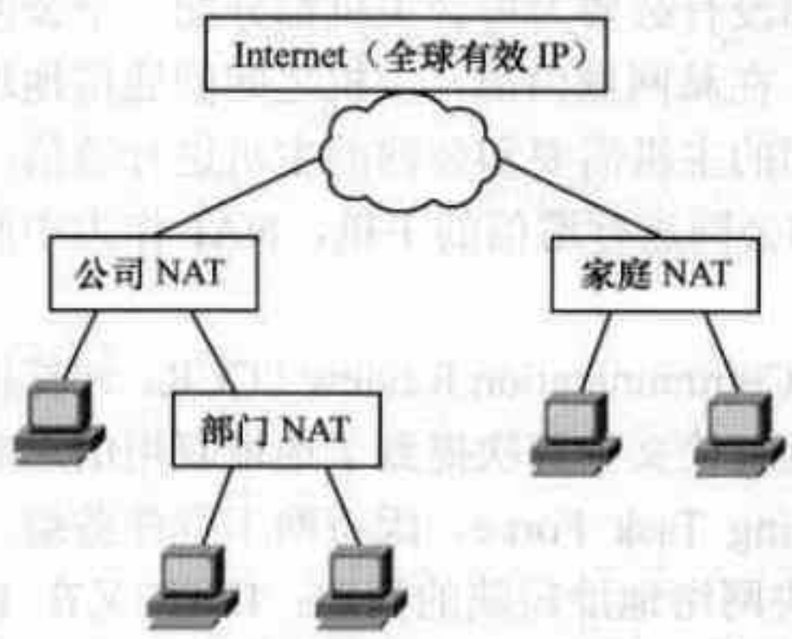


图 5-1　NAT 网络示意图

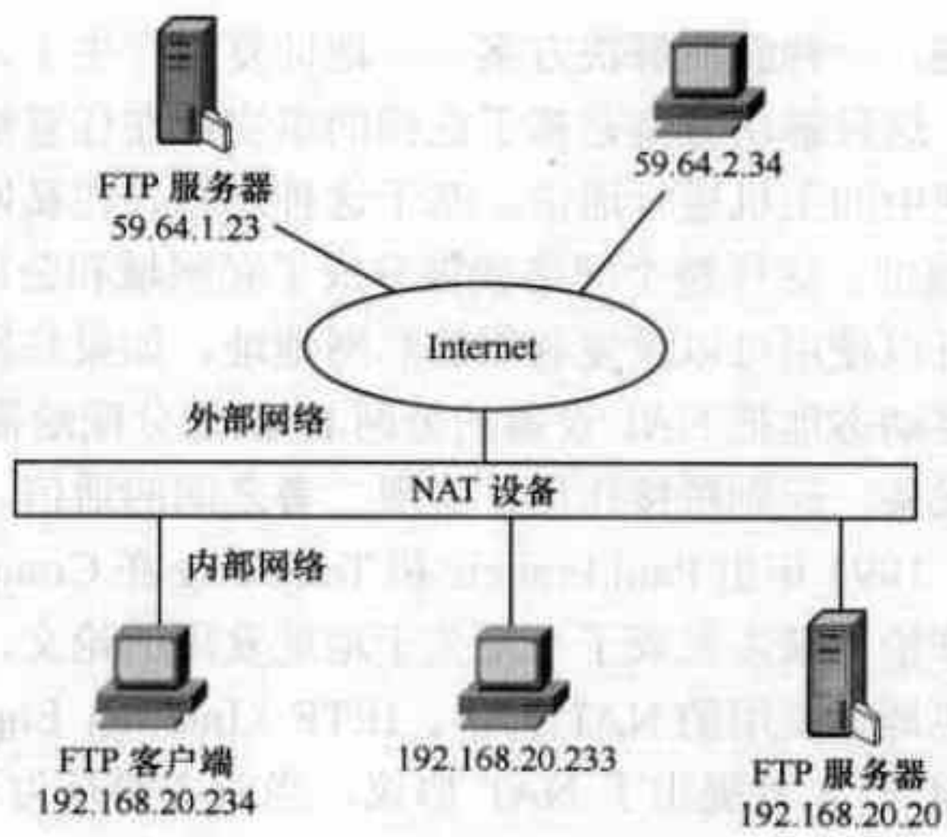


图 5-2　NAT 功能示意图

3．NAT 的作用

在网络中引入 NAT 设备能够暂时解决地址空间不足等问题，但同时可能带来一些负面影响。表 5-3 详细说明了 NAT 在网络中的作用和可能的不足。

表 5-3　　NAT 的作用和不足

编号	NAT 的作用	可能带来的问题
（1）	解决 IPv4 网络地址短缺问题，实现地址复用	只是在一定程度上缓解现行 IPv4 地址短缺问题。长期来看，它仍是一种权宜之计，即使不考虑架构和部署问题，IPv4 地址空间本身仍是有限的，如果所有的网络都隐藏在 NAT 设备之后，IPv4 地址空间不久仍将会耗尽。IPv6 才是最根本的解决方案
（2）	保护网络安全。实现企业 LAN 内部结构隐藏，能够在一定程度上防范外部网络攻击	如果攻击者获得对 NAT 设备的控制，整个网络将会变得非常脆弱。因此，尽管 NAT 在网络保护方面有其可取之处，但却不能作为替换防火墙的方案使用，只能作为一种用于家庭与小型办公网络的辅助性安全措施。而且由于 NAT 改变了源地址和目的地址，因此使用 NAT 后，防火墙的安全策略也需要随之调整，从而可能会相应地带来部分安全漏洞。其次，部署了 VPN 的企业 LAN 在采用 IPSec 进行安全加密时，若 NAT 设置不当，还会破坏 VPN 功能
（3）	便于网络管理	用户网络在使用 NAT 时，由于每个数据包都需要在 NAT 设备中翻译成正确的 IP 地址后发往下一级，给处理器带来了额外的负担。多专网整合可能带来问题

如图 5-3 所示，内网 IP 为 192.168.20.1 的 NAT 设备外网 IP 为 183.1.2.3。

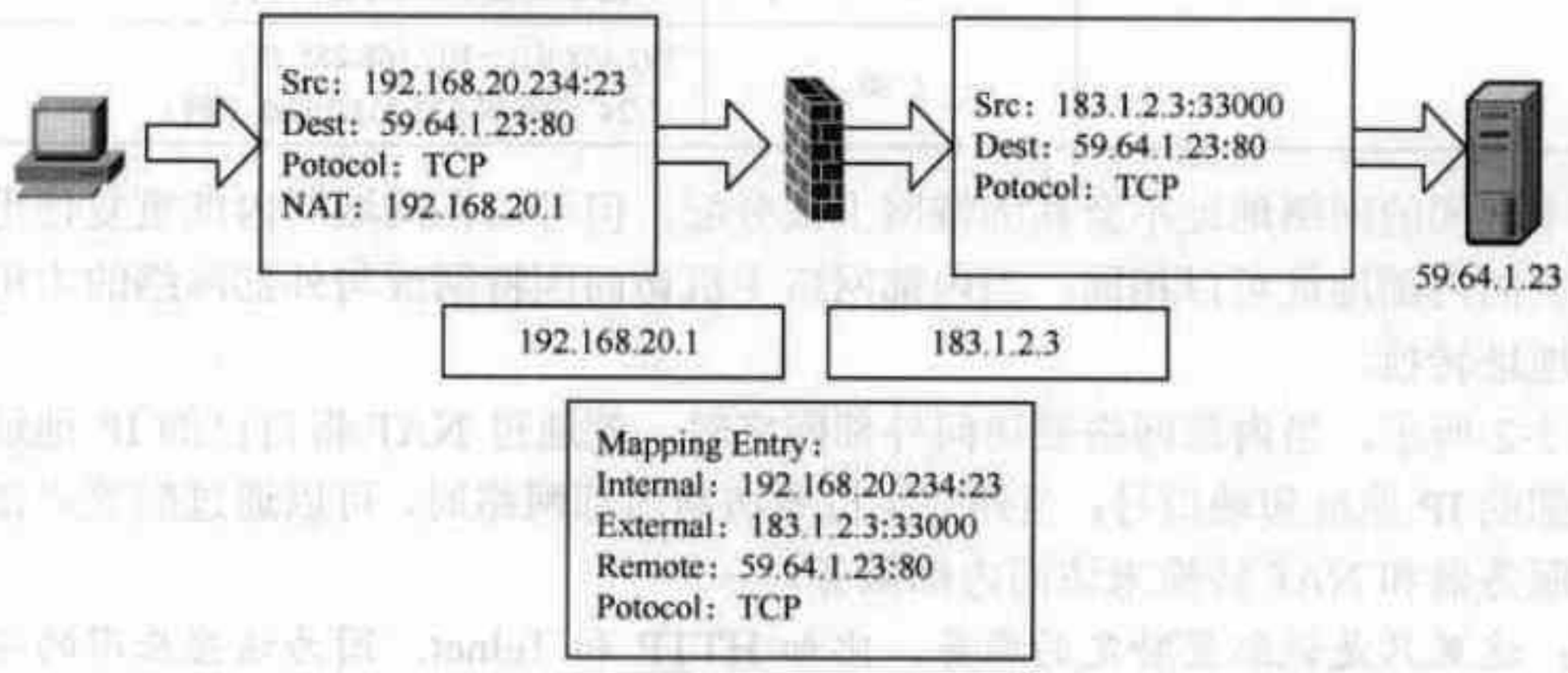


图 5-3　NAT 数据包转换

内网主机 192.168.20.234 想访问外网服务器 59.64.1.23，内网主机向外网服务器发送数据

www.ckook.com

包：192.168.20.234:23→59.64.1.23:80。

当此数据包到达 NAT 时，NAT 检测出此数据包是要发送到外网的，NAT 会修改此数据包的源地址和端口，并在映射表中添加 IP 和端口的映射关系条目，然后将修改后的数据包转发出去：183.1.2.3:33000→59.64.1.23:80。

外网服务器发送回应数据包：59.64.1.23:80→183.1.2.3:33000。

NAT 收到外网服务器发送的数据包，检查 IP 和端口映射关系表，发现 183.1.2.3:33000 与 192.168.20.234:23 对应，于是 NAT 修改数据包，并将数据包转发：59.64.1.23:80→192.168.20.234:23。

4．NAT 对 P2P 应用的影响

虽然 NAT 技术使得局域网的用户可以共享一个 IP 地址访问 Internet，节省了 IP 地址资源，同时也起到了防火墙的作用，但 NAT 的存在给 P2P 应用也带来了问题。因为 NAT 不允许外部主机主动访问内部主机，这样虽然 NAT 使得内部网络有安全性高的优点，但同时这种局限性却不适合 P2P 应用的需要，因为 P2P 中无论是内网的主机还是外部网络的主机的地位都是平等的，它们都可以相互建立直接的连接，实现信息共享。但由于 NAT 的存在，使得外部网络的主机无法首先主动发出到内网主机的连接。这些问题体现在以下几个方面。

（1）内网主机的 IP 地址是私有地址，它们在 Internet 上是无效的，从而使得内网主机对外网通常是不可见的。

（2）即使内网主机的 IP 地址已知，但 NAT 仍然阻止外网主机主动访问内网主机。

（3）内网主机可以作为客房端访问外网，但却不能作为服务器向外网提供服务。

上述这些限制使得存在 NAT 的网络中，外网主机无法穿过 NAT 主动与内网主机进行通信，P2P 网络各主机无法彼此发现对方进行对等信息交换，特别是位于不同 NAT 之后的不同内网中的计算机，更是无法相互连接。如果不能够穿过 NAT，P2P 网络就无法实现真正的对等，也就无法实现所有信息的交换或共享。

随着基于 Internet 的 P2P 网络技术的广泛应用，更多的内网主机需要参与到 P2P 中来。NAT 之后的主机 IP 地址在 Internet 上是不可见的，Internet 上的主机不能主动访问这些 NAT 之后的主机，而位于不同 NAT 之后的主机间更是无法相互识别而不能直接交换信息。但是 P2P 网络要求任何主机之间都能够直接对等交换信息，这就使得 P2P 网络应用必须解决穿越 NAT 实现双向对等通信的问题。

5.1.2 STUN 协议与 NAT 穿越

1．STUN 协议

解决 NAT 穿越问题的一种方法是在用户终端进行干预。用户通过某种机制预先得到其地址对应在出口 NAT 上的对外地址，然后在报文负载中所描述的地址信息就直接填写出口 NAT 上的对外地址，而不是私网内用户的私有 IP 地址，这样报文负载中的内容在经过 NAT 时就无需被修改了，只需按普通 NAT 流程转换报文头的 IP 地址即可，负载中的 IP 地址信息和报文头地址信息又是一致的。STUN 协议就是基于此思路来解决 NAT 穿越问题的。

www.ckook.com

STUN 协议由 IETF RFC 3489 定义，它的全称是 Simple Traversal of User Datagram Protocol Through Network Address Translators（NATs），即 UDP 对 NAT 的简单穿越方式，它是一个轻量级的协议。STUN 协议在 NAT 穿透中的作用，主要是允许应用程序确定在其本身和公共因特网之间是否使用了 NAT 和防火墙，并且判断出 NAT 和防火墙的具体类型，但是其本身并不提供对 NAT 的穿越方案。STUN 协议适用于现有的许多种 NAT，不需要对 NAT 进行任何特殊的配置。所以，它允许许多应用程序通过现有的 NAT 体系进行工作，并且支持同时使用多个 NAT 的情况。典型的 STUN 系统配置如图 5-4 所示。

STUN 中定义了两类不同的实体，分别是 STUN Client 和 STUN Server，详见表 5-4。

表 5-4　　STUN 实体定义

实　体	定　义
STUN Client	STUN Client 被定义为一个产生 STUN 请求的实体，在网络终端（比如用户 PC）或网络元件（比如 Conferencing Server，会话服务器）上运行
STUN Server	STUN Server 被定义为一个接受 STUN 请求并产生 STUN 响应消息的实体，通常连接在外部 Internet 上

如图 5-4 所示，其中 STUN Client 可以位于一层或多层 NAT 之后，STUN Server 则直接连接在外部 Internet 上。在 STUN 协议中，应用程序（即 STUN Client）向 NAT 外的 STUN Server 通过 UDP 发送请求 STUN 消息，STUN Client 可以通过设置不同的请求消息属性值，比如 CHANGE-REQUEST 来告知 STUN Server 进行相应的动作。STUN Server 收到请求消息，产生响应消息，响应消息中包括请求消息的源地址和端口，即 STUN Client 在 NAT 上对应的外部地址和端口号。响应消息通过 NAT 转发给 STUN Client，STUN Client 通过响应消息体中的 SOURCE-ADDRESS 属性便可以得知其在 NAT 上对应的外部地址以及端口号，此外，通过查看响应消息体中的其他属性值可以得知所处 NAT 设备的更多相关特性，从而采取适当的策略穿越 NAT，和通信节点建立连接。

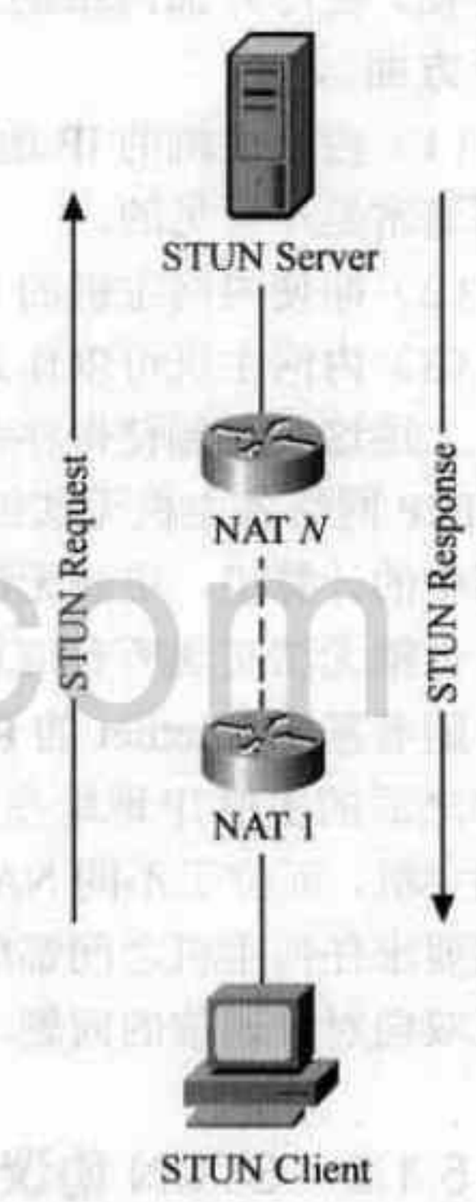


图 5-4　典型的 STUN 系统配置

STUN 协议是一个简单的客户端/服务器协议。客户端向服务器发送请求，服务器返回响应。有两种类型的请求：使用 UDP 发送的绑定请求（Binding Requests）和通过 TCP 使用 TLS 发送的共享秘密请求（Shared Secret Requests）。共享秘密请求希望服务器返回一个临时的用户名和密码。这个临时的用户名和密码，用于后面的绑定请求、绑定响应中的鉴权和检查消息的完整性的功用。Binding Request 用于确定 NAT 分配的绑定地址。客房端使用 UDP 向服务器发送 Binding Request，服务器检查请求的源地址和端口号，并将它们复制到将要发送回客房端的响应中。客户端可以通过请求中的几个参数，来选择服务器发送响应的目标地址或者使用的源地址和端口号。

2. NAT 类型

在 STUN 协议中，把 NAT 分为以下 4 种类型，在利用 STUN 进行 NAT 穿越时，将根据不同的 NAT 类型采取不同的穿越方法。

www.ckook.com

Full Cone 型 NAT：在 Full Cone 型 NAT 中，来自相同内部 IP 地址和端口号的请求消息，将被映射为相同的外部地址和端口号。同时，一个 NAT 以外的外部节点，可以通过向 NAT 之后节点的映射地址和端口发送数据包的方式，向 NAT 之后的节点发送数据包，从而和任意内部节点进行通信。示意图如图 5-5 所示。

NAT 会将客房端地址（X，y）转换成公网地址（A，b）并绑定；任何包都可以通过地址（A，b）送到客户主机的地址（X，y）上。

Restricted Cone 型 NAT：在该类型 NAT 中，来自相同内部 IP 地址和端口号的请求消息将被映射为相同的外部地址和端口号。同时，一个 NAT 以外的外部节点（其 IP 地址为 X）可以向 NAT 之后的节点发送数据包，当且仅当在这之前该内部节点曾经发送过数据报到外部 IP 地址 X。示意图如图 5-6 所示。

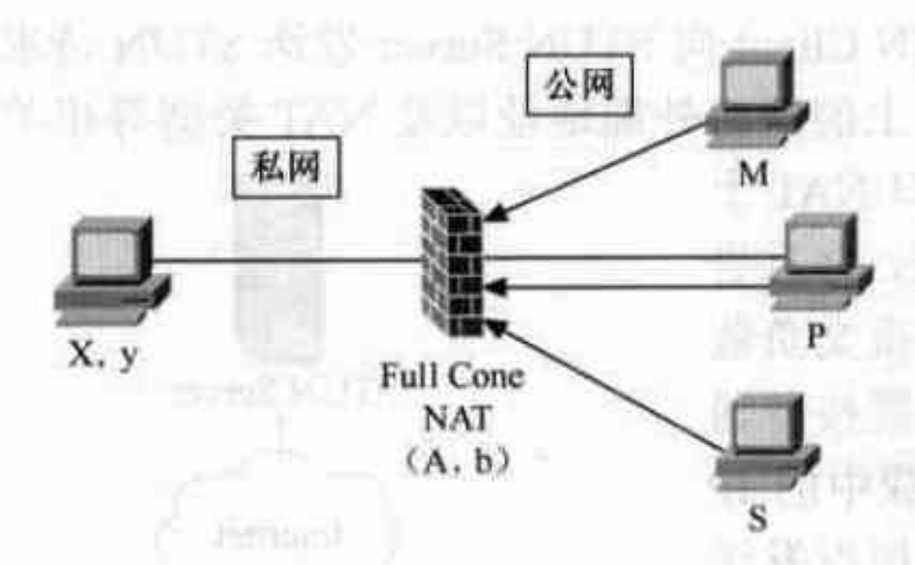


图 5-5 Full Cone NAT

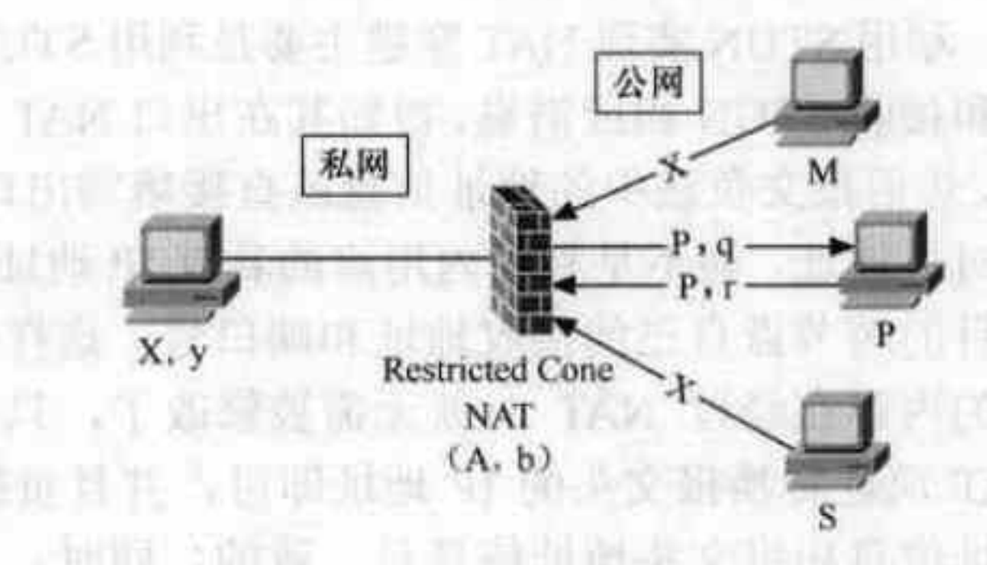


图 5-6 Restricted Cone NAT

NAT 会将客房端地址（X，y）转换成公网地址（A，b）并绑定，只有来自主机 P 的包才能和主机（X，y）通信。

Port Restricted Cone 型 NAT：在该类型 NAT 中，来自相同内部 IP 地址和端口号的请求消息将被映射为相同的外部地址和端口号。同时，一个 NAT 以外的外部节点（其 IP 地址为 X，端口号为 P）可以向 NAT 之后的节点发送数据包，当且仅当在这之前该内部节点曾经发送过数据报到外部 IP 地址 X 和端口 P。示意图如图 5-7 所示。

NAT 会将客房端地址（X，y）转换成公网地址（A，b）并绑定，只有来自主机（P，q）的包才能和主机（X，y）通信。

对称型 NAT（Symmetric NAT）：在对称型 NAT 中，只有来自相同内部 IP 地址和端口号，并且发送往同一个目的地址和端口的请求消息，才将被映射为相同的外部地址和端口号。同时，一个 NAT 以外的外部节点可以向 NAT 之后的节点发送数据报，当且仅当在这之前外部节点曾经接收过该内部节点发送给它的 UDP 数据包。示意如图 5-8 所示。

NAT 会将客房端地址（X，y）转换成公网地址（A，b）并绑定为（X，y）⇔（A，b）⇔（P，q）；NAT 只接收来自（P，q）的包，将它转给（X，y）；每次客房端请求一个不同的公网地址和端口，NAT 会新分配一个端口号（C，d）。

除了 NAT 类型之外，STUN 中还详细定义了 STUN 消息的各种属性，STUN 请求消息和响应消息的格式，以及 STUN 错误信息和相应的 STUN 错误码。当错误发生的时候，可以通过检查错误码来判断发生错误的类型。

www.ckook.com

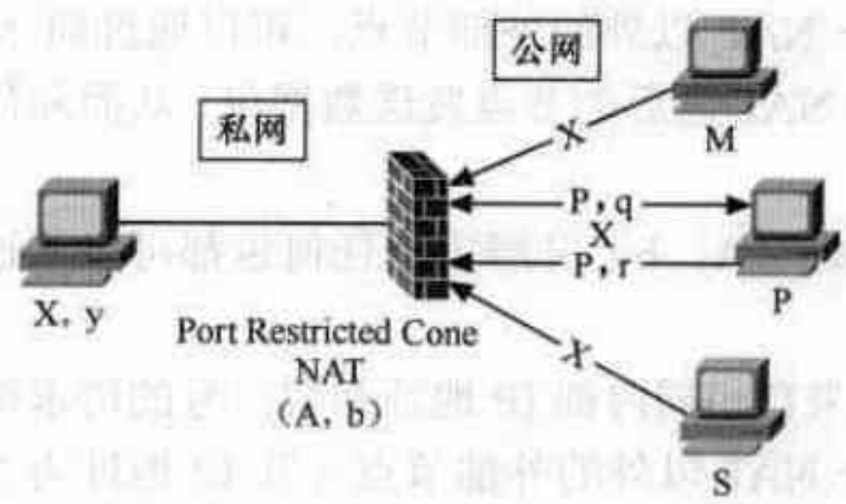


图 5-7 Port Restricted Cone NAT

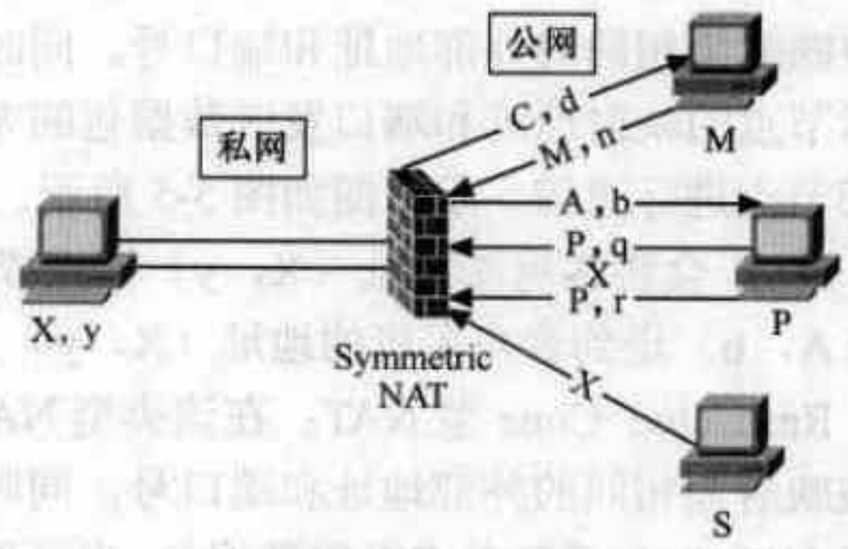


图 5-8 Symmetric NAT

3．利用 STUN 进行 NAT 穿越

利用 STUN 实现 NAT 穿越主要是利用 STUN Client 向 STUN Server 发送 STUN 请求消息和接收 STUN 相应消息，得知其在出口 NAT 上的映射外部地址以及 NAT 类型等相关信息，然后报文负载中的地址信息就直接填写出口 NAT 上的对外地址，而不是私网内用户的私有 IP 地址，并且告知目的端节点自己的接收地址和端口号。这样报文负载中的内容在经过 NAT 时就无需被修改了，只需按普通 NAT 流程转换报文头的 IP 地址即可，并且负载中的 IP 地址信息和报文头地址信息是一致的。同时，根据发送端和接收端所处的 NAT 类型，采取适当的策略来完成对 NAT 的穿透。

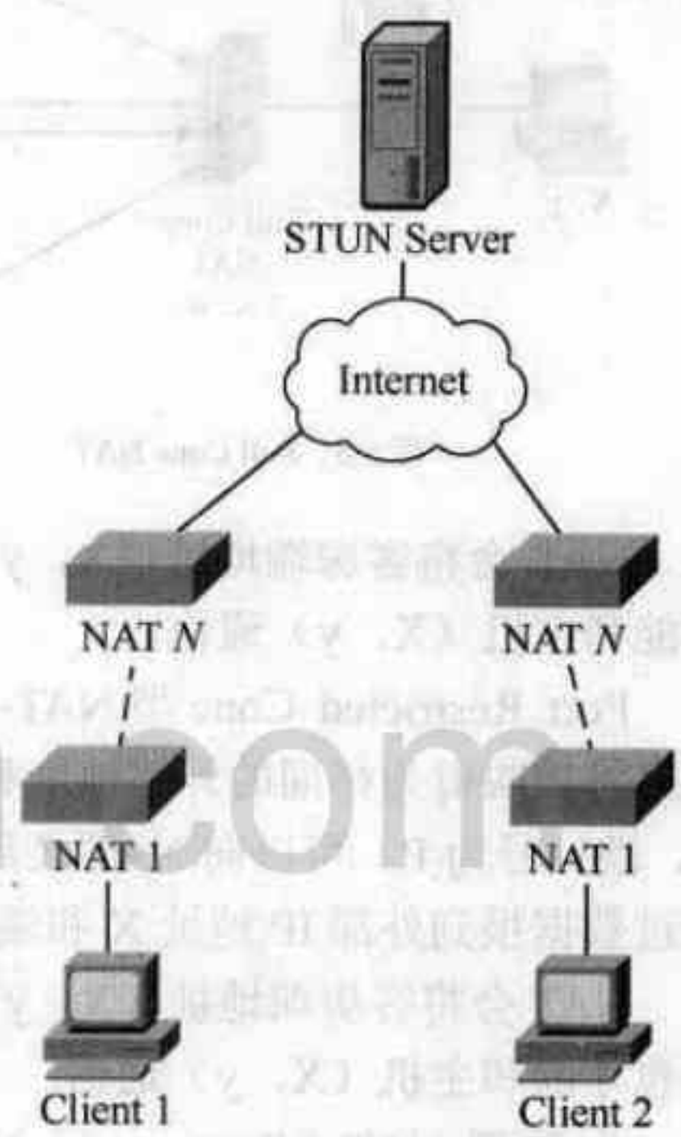


图 5-9 利用 STUN 进行 NAT 穿越

利用 STUN 方式进行 NAT 穿越的网络系统如图 5-9 所示。其中发送节点和接收节点可以处于一个 NAT 之后，也可以处于多个NAT之后，也可以直接连接在外部Internet上。STUN Server 连接在外部 Internet 上。

如图 5-9 所示，根据 STUN 标准，发送端节点（STUN Client 1）和接收端节点（STUN Client 2）可以通过向 STUN Server 发送请求消息，从 STUN Server 的响应消息中得知所处 NAT 的具体类型。

当节点位于多层 NAT 之后时，NAT 类型为所有 NAT 中限制级别最高的 NAT 类型。例如，当节点位于 3 层 NAT 之后，3 个 NAT 的类型分别是 Full Cone 型、限制型和对称型，那么节点所处 NAT 的类型就是对称型。

5.1.3 UDP 穿越 NAT

如果 P2P 网络中的两台主机都位于 NAT 之后，那么双方都不知道对方的公网 IP 地址，从而双方都无法主动发出连接请求。要解决这个问题需要借助双方都熟知的公网服务器来协助建立连接，也就是使用 UDP 在 NAT 上打洞的技术（UDP Hole Punching）。

假设 P2P 网络中两台主机的网络位置情况如图 5-10 所示。

www.ckook.com

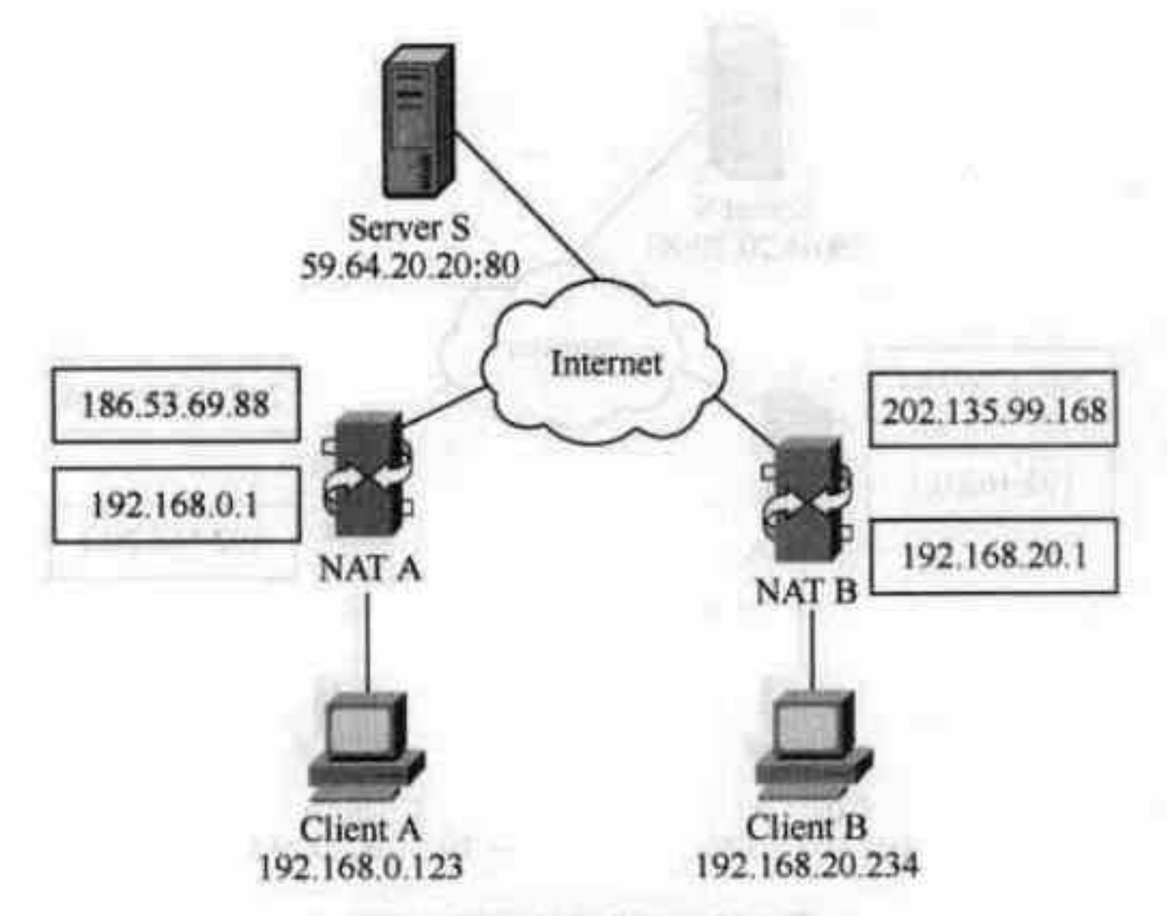


图 5-10 两台主机位于不同 NAT 后

在 Client A 和 Client B 之间建立 UDP 直连的过程如下：

（1）Client A 登录 Server S，NAT A 为这次的 Session 分配了一个端口 55000，那么 Server S 收到的 Client A 的地址是 186.53.69.88:55000，这就是 Client A 的外网地址。

（2）Client B 登录 Server S，NAT B 给此次 Session 分配的端口是 44000，那么 Server S 收到的 B 的地址是 202.135.99.168:44000。

此时，Client A 与 Client B 都可以与 Server S 通信了。如果 Client A 此时想直接发送 UDP 数据包给 Client B，那么 Client A 可以从 Server S 获得 B 的公网地址 202.135.99.168:44000，但是 Client A 还不能使用该公网地址直接和 Client B 通信，因为此时 NAT B 会将 Client A 主动发来的信息丢弃，因为此数据包是没有会话记录的。

（3）现在需要在 NAT B 上打一个方向为 186.53.69.88（即 Client A 的外网地址）的洞，那么 Client A 发送到 202.135.99.168:44000 的信息，Client B 就能收到了。

（4）Server S 负责向 Client B 发送打洞指令。

（5）Client B 向 Client A 的公网地址发送一个 UDP 报文，虽然此数据包会被 NAT A 丢弃。但是 NAT B 建立了会话记录，不会丢弃 Client A 的包了。

（6）Server S 通知 Client A“洞”已打好，Client A 可以发连接包给 Client B 了。

（7）Client A 向 Client B 发送 UDP 包。

（8）至此双方可以进行 UDP 直连。

注意：以上过程只适合于 Cone NAT 的情况，如果是 Symmetric NAT，那么当 Client B 向 Client A 打洞的端口已经重新分配时，Client B 将无法知道这个端口。

5.1.4 TCP 穿越 NAT

如图 5-11 所示，Client A 和 Client B 进行 TCP 直连过程如下：

（1）Client A 登录 Server S，NAT A 为这次的 Session 分配了一个端口 55000，那么 Server S 收到的 Client A 的地址是 186.53.69.88:55000，这就是 Client A 的外网地址。

www.ckook.com

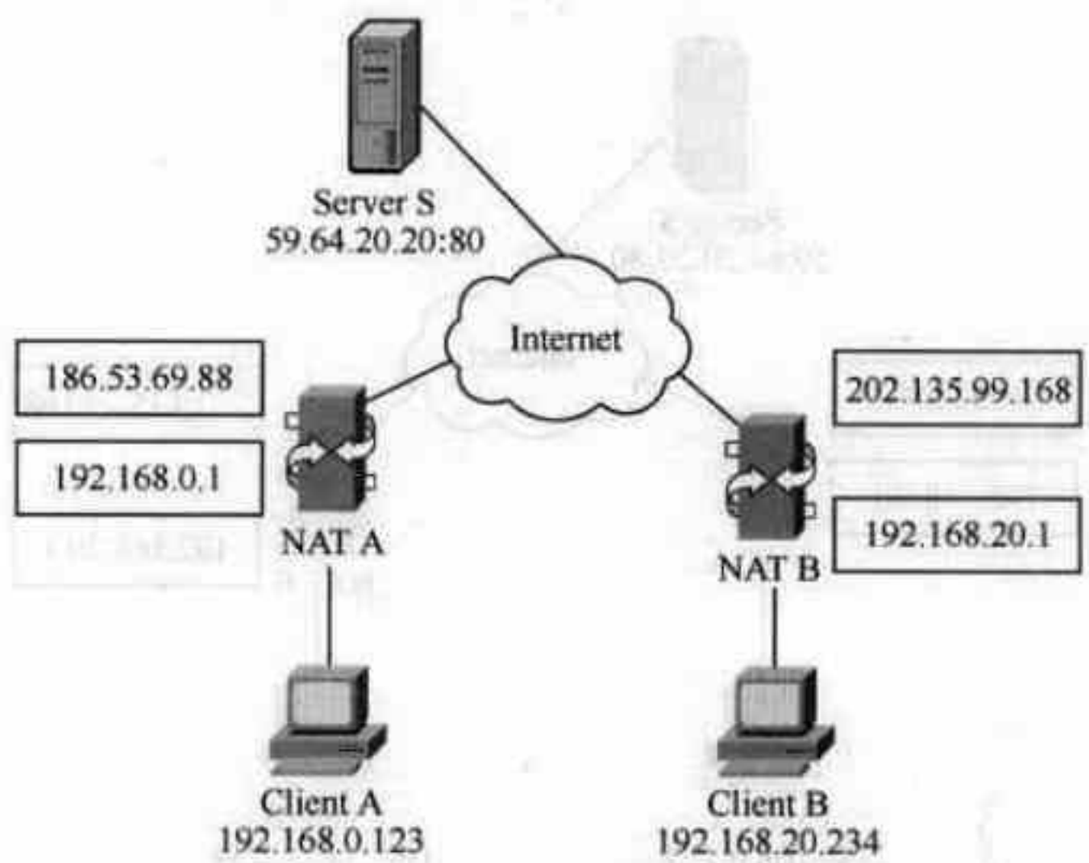


图 5-11　TCP 直连网络示意图

（2）Client B 登录 Server S，NAT B 给此次 Session 分配的端口是 44000，那么 Server S 收到的 B 的地址是 202.135.99.168:44000。

此时，Client A 与 Client B 都可以与 Server S 通信了。如果 Client A 此时直接发送 TCP 连接包给 Client B，那么 Client A 可以从 Server S 获得 B 的公网地址 202.135.99.168:44000，但是 Client A 还不能使用该公网地址直接和 Client B 通信，因为此时 NAT B 会将 Client A 主动发来的信息丢弃，因为此数据包是没有会话记录的。

（3）Client A 向 Client B 发送数据包 TCP[SYN]192.168.0.123:1234→202.135.99.168:44000，经过 NAT A 后转换为 TCP[SYN]186.53.69.88:55000→202.135.99.168:44000。从而由 202.135.99.168:44000 发来的 TCP 包可以通过 NAT A 了，因为 NAT A 中有了相关的会话映射关系。

（4）Client B 向 Client A 发送数据包 TCP[SYN]192.168.20.234:1234→186.53.69.88:55000，经过 NAT B 后转换为 TCP[SYN]202.135.99.168:44000→186.53.69.88:55000。从而由 186.53.69.88:33000 发来的 TCP 包可以通过 NAT B 了，因为 NAT B 中有了相关的会话映射关系。

（5）双方三次握手建立 TCP 直连。

5.1.5　NAT 类型检测

1．Natcheck 程序

从前面的章节可以看出要有效进行直连通信，关键的一点就是对端点所处的网络环境进行准确的判断，这样才能对症下药。

在开源项目网站 http://midcom-p2p.sourceforge.net/上有专门关于 NAT 穿越和 NAT 类型检测的资料，该网站提供了开源程序 natcheck 作为一个实现示例。

Natcheck 程序可以检测 NAT 设备对 P2P 通信协议的兼容性。目前已经发展到了第 3 个版本，同时支持 TCP 和 UDP 协议。

Natcheck 程序的编译和运行：如果是 64 位计算机，可以添加编译参数“-DHAVE_STDINT”或者直接修改程序的#define；如果在 Windows 平台上运行 NATCheck，首先安装 Windows

www.ckook.com

上的 GCC 环境 MinGW，然后使用-lws2_32 编译选项编译即可。

使用者可以将自己的检测结果上传到指定的统计网站上，但是目前使用 natcheck 程序得到的统计数据网站上不再提供了，可以在相关论文中查阅统计结果。

使用 natcheck 程序需要自己设置服务器，并且修改源代码指向这些服务器。以前的版本当中还有一个 natbouncer 程序，现在此程序已经合并进 natserver 程序当中，通过指定不同的参数来确定运行的方式。在主服务器上运行“natserver 1”和“natserver 2”，在协同服务器上运行“natserver 3”。

这个程序使用的技术简单而且有效，已经被许多在线游戏和其他 P2P 应用广泛地使用。

2．Winstun 检测程序

（1）STUN 测试 NAT 类型流程

客户端会预先编制 4 种测试规则，分别如下：

测试 1 如图 5-12 所示。

STUN Client 发送空的 CHANGE REQUEST，用于获取客户端公网 IP 和端口。

测试 2 如图 5-13 所示。

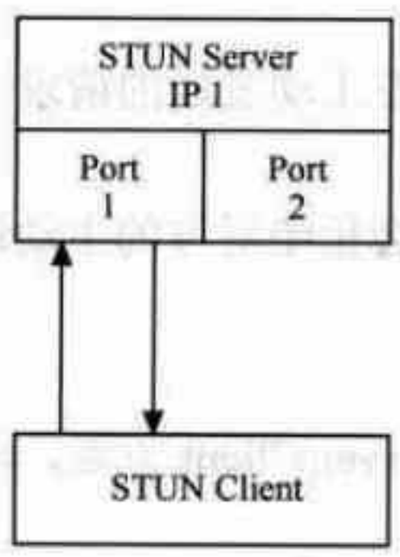


图 5-12　STUN 测试 1

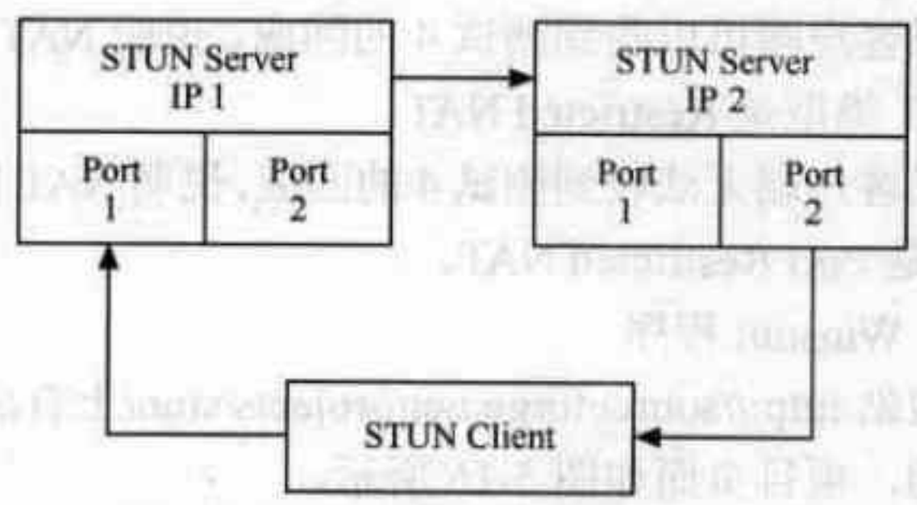


图 5-13　STUN 测试 2

STUN Client 发送“change ip”和“change port”消息，并要求服务器发送更改了 IP 和端口的回应。

测试 3 如图 5-14 所示。

STUN Client 发送空的 CHANGE REQUEST，用于获取客户端公网 IP 和端口。

测试 4 如图 5-15 所示。

STUN Client 发送“change port”要求服务器发送更改了端口的回应。

在测试开始后，客户端首先向服务器发送没有任何附带要求的 CHANGE-REQUEST，如果发送失败，或没有接到回应，说明客户端所在的网络不支持 UDP。

如果接收到服务器的回应，如 BINDING-RESPONSE，客户端应该根据服务器的客户端公网 IP 地址与当前本机上的 IP 地址进行对比。这时，无论结果是否相同，都要继续发送 TEST II，并根据服务器的回应进行后继判断。

如果本机 IP 等于公网 IP，并且服务器关于测试 2 的回应也可以收到，那么客户端目前就在公网上，当前的 IP 地址就是公网 IP。但如果收不到对测试 2 的回应，说明客户端在防火墙后，外部主动发送的请求会被防火墙拒绝。

如果本机 IP 不等于公网 IP，但是可以收到对测试 2 的回应，说明用户在 Full Cone NAT 后面，客户端的内网 IP 和端口与 NAT 上外网 IP 和端口相互绑定，即使服务器改用另一个 IP

www.docin.com

地址，仍可以正常通过。

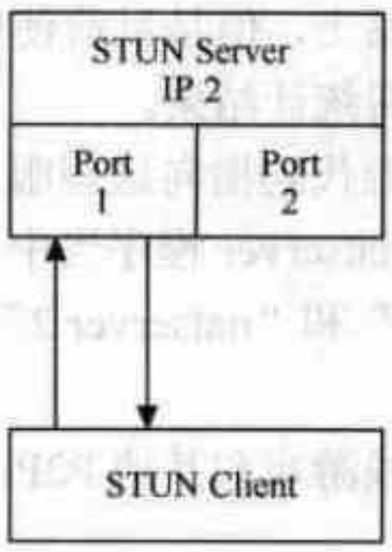


图 5-14　STUN 测试 3

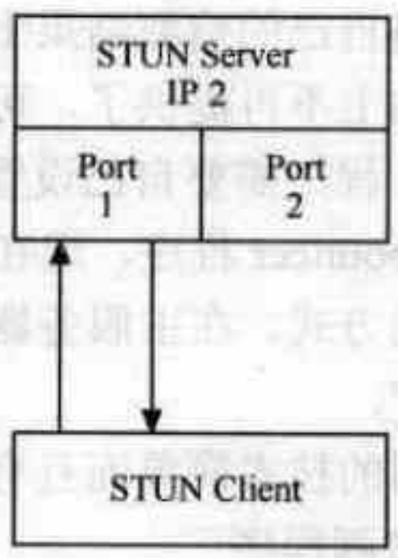


图 5-15　STUN 测试 4

如果无法收到测试 2 的回应，客户端则应顺次发送测试 3，并判断回应的 IP 和端口与之前测试 1 返回的公网 IP 端口是否相同。如果不同，说明用户在对称 NAT 后，这种类型的 NAT 对于每次连接会话都会更改所分配的端口。

如果对第二次测试 1 的回应 IP 地址和端口与第一次相同，则客户端发送测试 3，即要求服务器更改端口。

如果客户端可以收到测试 4 的回应，说明 NAT 并不会封堵对方主动发送的请求，当前使用的 NAT 类型是 Restricted NAT。

如果客户端无法收到测试 4 的回应，说明 NAT 像防火墙一样，会拒绝对方的主动请求，当前 NAT 是 Port Restricted NAT。

（2）Winstun 程序

在网站 http://sourceforge.net/projects/stun/上有成熟的 STUN Server/Client 方案，这个项目是开源的，项目页面如图 5-16 所示。

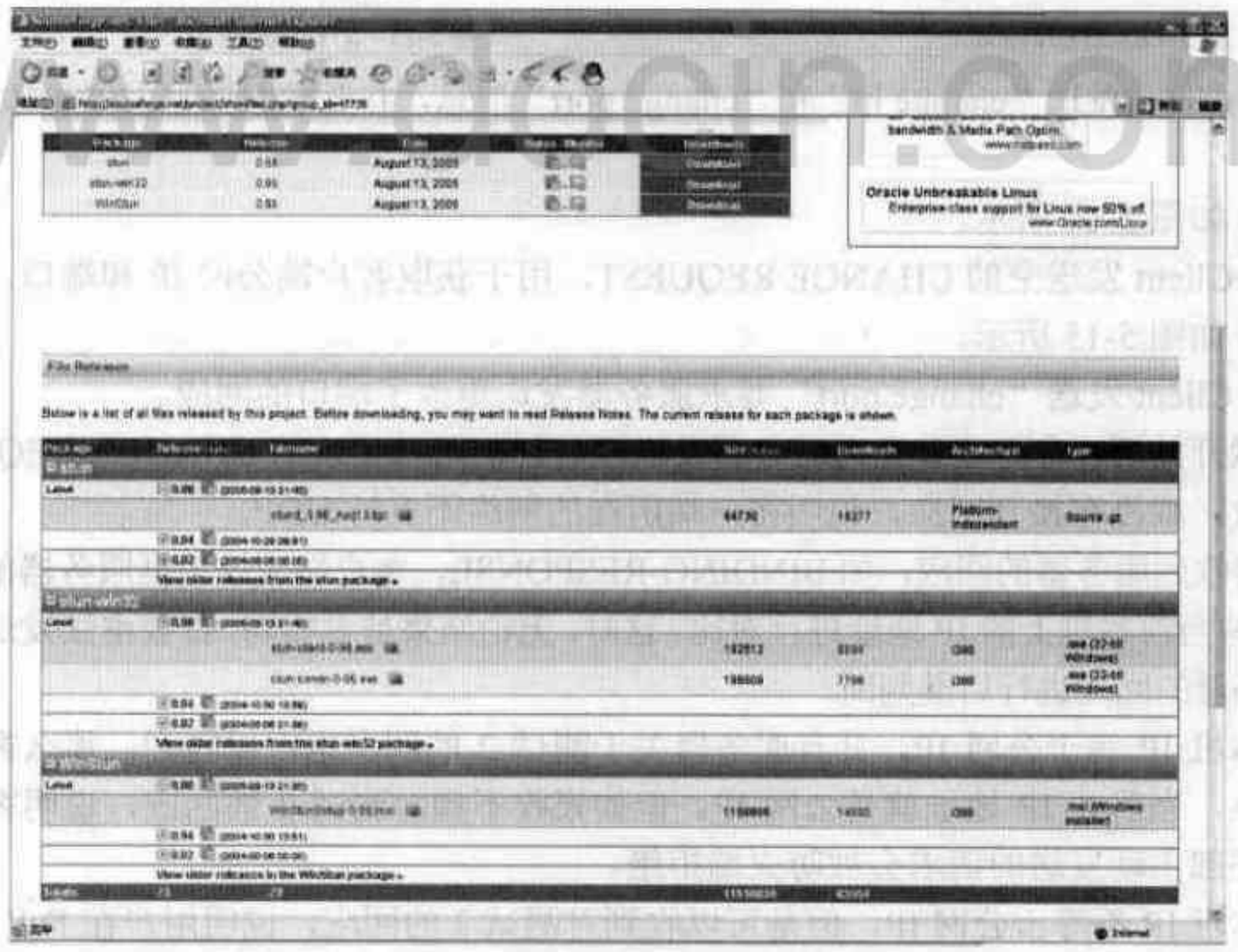


图 5-16　开源 STUN 项目

www.ckook.com

下载 WinStun 安装文件 WinStunSetup-0-96.msi，并安装。运行后界面如图 5-17 所示。点击“Run Test”按钮，获得主机所处的 NAT 类型信息，如图 5-18 所示。

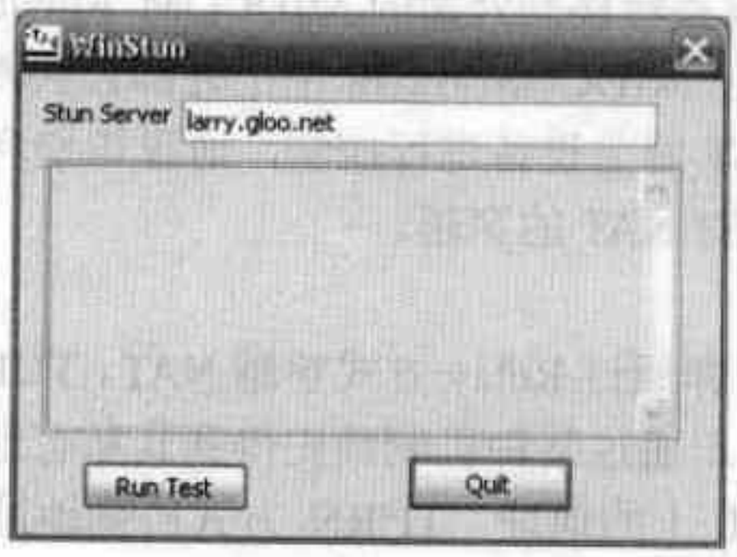


图 5-17 WinStun 启动界面

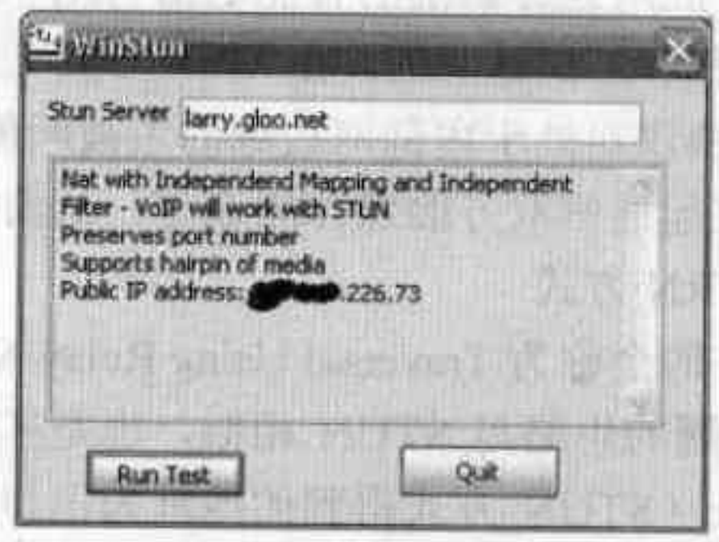


图 5-18 WinStun 检测结果

5.1.6 常见 NAT 穿越解决方案

1. UPnP 方式

UPnP 规范，即 Universal Plug and Play，通用即插即用是一种用于智能设备的对等网络连接的体系结构。UPnP 使用 Internet 标准和技术，使支持 UPnP 的设备彼此可以互联协同工作。

读者可能熟悉计算机上常见的即插即用设备，UPnP 是对这种传统的技术进行了扩展，扩展到了网络上。这个技术简化了智能设备的联网过程。

UPnP 应用如图 5-19 所示。当一台主机接入 UPnP 网络后，便可以自动地和所有支持 UPnP 的设备通信。

UPnP 相当于建立了一个标准的设备控制协议。此协议基于已有的成熟的 Internet 协议，其通用性能非常强。UPnP 设备可以自动配置网络地址。

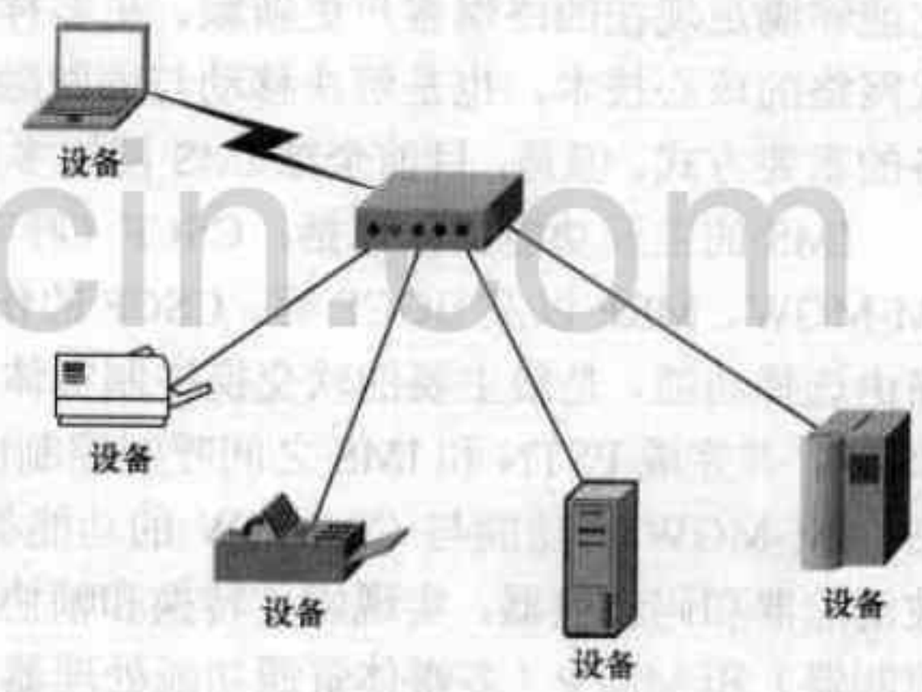


图 5-19 UPnP 应用

UPnP NAT 穿越解决方案允许网络应用程序对它们是否位于一个具有 UPnP 能力的 NAT 设备之后进行检测。然后这些程序将获得共享的全球有效 IP 地址，并且配置端口映射从而可以将来自 NAT 外部端口的数据包转发到应用程序使用的内部端口。

2. ALG 方式

ALG（Application Level Gateway，应用层网关），是一种可以识别指定 IP 协议的设备。ALG 不是简单地查看包头信息来决定数据包是否可以通过，而是更深层地分析数据包负载内的数据，也就是应用层的数据。普通 NAT 是通过修改报文头部的地址信息实现地址的转换的，但对于多媒体通信和 P2P 应用来说，在 TCP/UDP 负载中也附带地址信息。一般的方法，应用程序在负载中填写的是其自身地址，此地址信息在通过 NAT 时被修改为 NAT 上对外的地址。

www.docin.com

www.ckook.com

3．MIDCOM 方式

MIDCOM 方式采取了由可信的第三方（MIDCOM Agent）对 NAT 网关设备进行控制的框架结构，应用业务识别的智能也由 NAT 网关设备转移到外部的 MIDCOM Agent 上，因此应用协议对 NAT 是透明的。在 MIDCOM 方式中，MIDCOM Agent 位于外部公共网上，通过 MIDCOM 协议对处于内部网边缘的主被叫侧 NAT 网关设备进行控制，通过识别和解析应用协议，建立起通信双方的媒体流通道，从而实现对 NAT 的穿越。

4．TURN 方式

TURN 的全称为 Traversal Using Relay NAT，即通过 Relay 方式穿越 NAT。TURN 方式解决 NAT 问题的思路与 STUN 相似，也是基于用户通过某种机制预先得到其私有地址对应在公网的地址（STUN 方式得到的地址为出口 NAT 上的地址，TURN 方式得到地址为 TURN Server 上的地址），然后在报文负载中所描述的地址信息就直接填写该公网地址的方式，实际应用原理是一样的。

5.2 P2P 与 IMS 结合

5.2.1 什么是 IMS

IMS（IP Multimedia Subsystem）是 IP 多媒体子系统，是一种全新的多媒体业务形式，它能够满足现在的终端客户更新颖、更多样化多媒体业务的需求。目前，IMS 被认为是下一代网络的核心技术，也是解决移动与固网融合，引入语音、数据、视频三重融合等差异化业务的重要方式。但是，目前全球 IMS 网络多数处于初级阶段，应用方式也处于业界探讨当中。

IMS 的主要功能实体包括：CSCF（呼叫控制服务器）、MGCF（媒体网关控制服务器）、IM-MGW、MRF 以及 BGCF 等。CSCF 的作用是完成入呼叫网关功能、呼叫业务触发功能和路由选择功能，是最主要的软交换控制实体；MGCF 的作用是根据被叫号码和来话情况选择 CSCF，并完成 PSTN 和 IMS 之间呼叫控制协议转换，以及控制 IM-MGW 媒体通道的呼叫状态；IM-MGW 的功能与 CS-MGW 的功能类似，与 MGCF 一起完成资源控制，以及通过回波消除器和码转换器，实现媒体转换和帧协议转换功能；MRF 分为 MRFC（多媒体资源功能控制器）和 MRFP（多媒体资源功能处理器），分别完成媒体流的控制和承载功能；BGCF 的主要作用是在与 PSTN 通信时，完成信令转发功能。

5.2.2 P2P 与 IMS 的网络融合

2007 年 11 月 15～16 日，IMS 技术峰会暨第二届 IMS 技术发展高层咨询会在北京召开。本次峰会由中国通信学会主办，云集了业界各主流运营商及通信设备制造商。中兴通讯在会上做了将 P2P 与 IMS 技术相结合的主题演讲，引起与会者的极大关注。

3GPP 提出 IMS 后，网络融合引起了人们广泛的关注，P2P 在因特网的广泛应用和大量用户也极大地吸引了运营商。P2P 网络与 IMS 网络融合成为一种趋势，很多人在探索相应的

www.ckook.com

运作方式和赢利模式。图 5-20 所示是参考文献[3]提供的一种网络融合模式，其中采用了混合式 P2P 结构，并且借鉴了 BitTorrent 的原理。

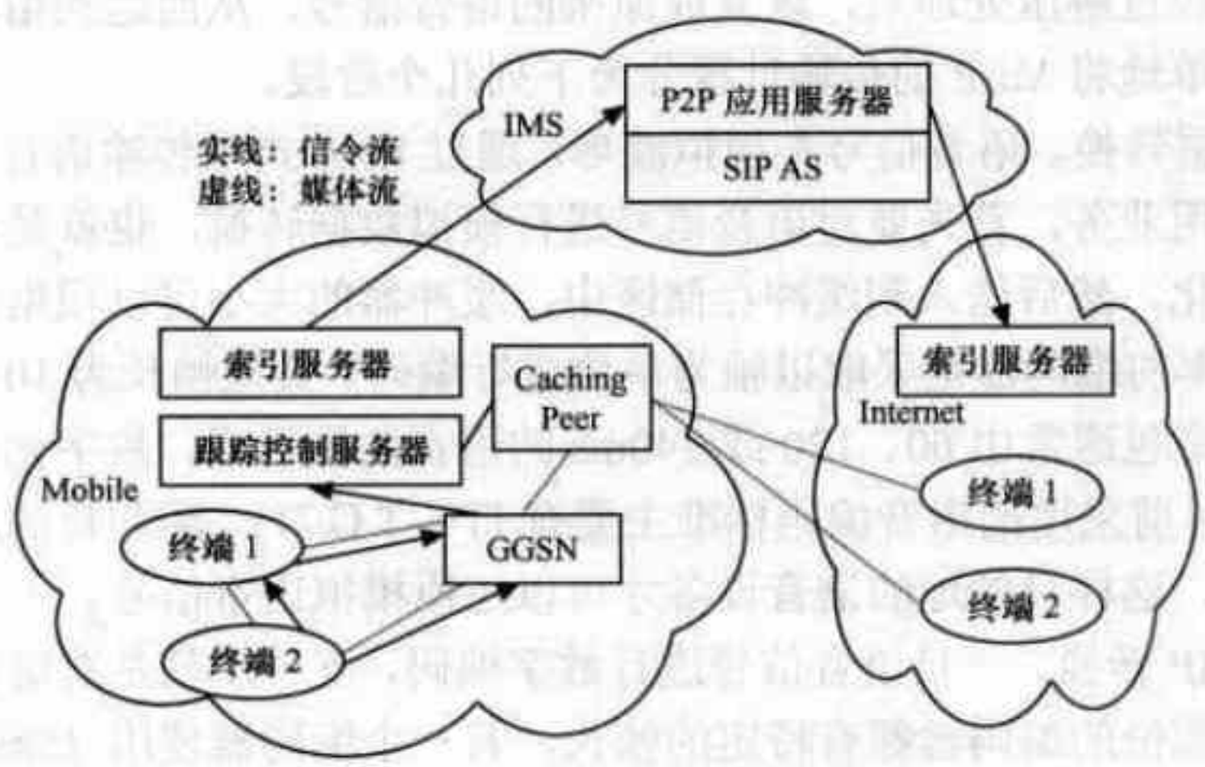


图 5-20 基于 P2P 共享应用的 IMS 方案

5.3 VoIP 通信基础

VoIP（Voice over Internet Protocol）是一种以 IP 电话为主，并推出相应的增值业务的技术。 VoIP 最大的优势是能广泛地采用 Internet 和全球 IP 互连的环境，提供比传统业务更多、更好的服务。VoIP 可以在 IP 网络上传送语音、传真、视频和数据等业务，如统一消息、虚拟电话、虚拟语音/传真邮箱、查号业务、Internet 呼叫中心、Internet 呼叫管理、电视会议、电子商务、传真存储转发和各种信息的存储转发等。本节将就 VoIP 的基本概念以及与其相关的一些协议做一个初步的介绍。

www.docin.com

5.3.1 VoIP 系统概念

为了在一个 IP 网络上传输语音信号，要求具备几个元素和功能。最简单形式的网络由两个或多个具有 VoIP 功能的设备组成，这一设备通过一个 IP 网络连接。VoIP 模型的基本结构图如图 5-21 所示。

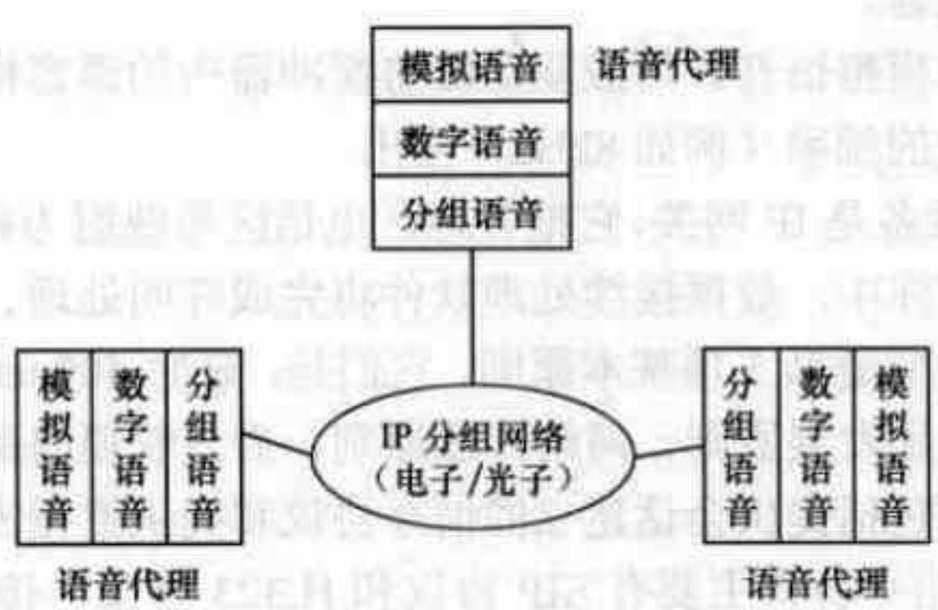


图 5-21 VoIP 的模型结构

www.ckook.com

VoIP 的基本原理是：通过语音的压缩算法对语音数据编码进行压缩处理，然后把这些语音数据按 TCP/IP 标准进行打包，经过 IP 网络把数据包送至接收地，再把这些语音数据包按照顺序组合起来，经过解压处理后，恢复成原来的语音信号，从而达到由因特网传送语音的目的。因此可以简单地将 VoIP 的传输过程分为下列几个阶段。

（1）语音—数据转换。语音信号是模拟波形，通过 IP 方式来传输语音，不管是实时应用业务还是非实时应用业务，首先要对语音信号进行模拟数据转换，也就是对模拟语音信号进行 8 位或 6 位的量化，然后送入到缓冲存储区中，缓冲器的大小可以根据延迟和编码的要求选择。许多低比特率的编码器是采取以帧为单位进行编码。典型帧长为 10～30ms。考虑传输过程中的代价，语音包通常由 60、120 或 240ms 的语音数据组成。数字化可以使用各种语音编码方案来实现，目前采用的语音编码标准主要有 ITU-T G.711。源和目的地的语音编码器必须实现相同的算法，这样目的地的语音设备才可以还原模拟语音信号。

（2）原数据到 IP 转换。一旦语音信号进行数字编码，下一步就是对语音包以特定的帧长进行压缩编码。大部份的编码器都有特定的帧长，若一个编码器使用 15ms 的帧，则把每次 60ms 的语音包分成 4 帧，并按顺序进行编码。每个帧合 120 个语音样点（抽样率为 8kHz）。编码后，将 4 个压缩的帧合成一个压缩的语音包送入网络处理器。网络处理器为语音添加包头、时标和其他信息后通过网络传送到另一端点。语音网络简单地建立通信端点之间的物理连接（一条线路），并在端点之间传输编码的信号。IP 网络不像电路交换网络，它不形成连接，它要求把数据放在可变长的数据报或分组中，然后给每个数据报附带寻址和控制信息，并通过网络发送，一站一站地转发到目的地。

（3）传送。在这个通道中，全部网络被看成一个从输入端接收语音包，然后在一定时间 t 内将其传送到网络输出端。t 可以在某个范围内变化，反映了网络传输中的抖动。网络中的每个节点检查每个 IP 数据附带的寻址信息，并使用这个信息把该数据报转发到目的地路径上的下一站。网络链路可以是支持 IP 数据流的任何拓扑结构或访问方法。

（4）IP 包—数据的转换。目的地 VoIP 设备接收这个 IP 数据并开始处理。网络级提供一个可变长度的缓冲器，用来调节网络产生的抖动。该缓冲器可容纳许多语音包，用户可以选择缓冲器的大小。小的缓冲器产生延迟较小，但不能调节大的抖动。其次，解码器将经编码的语音包解压缩后产生新的语音包，这个模块也可以按帧进行操作，完全和解码器的长度相同。若帧长度为 15ms，60ms 的语音包被分成 4 帧，然后它们被解码后还原成 60ms 的语音数据流送入解码缓冲器。在数据报的处理过程中，去掉寻址和控制信息，保留原始的数据，然后把这个数据提供给解码器。

（5）数字语音转换为模拟语音。播放驱动器将缓冲器中的语音样点（480 个）取出送入声卡，通过扬声器按预定的频率（例如 8kHz）播出。

VoIP 的核心与关键设备是 IP 网关，它把各地区电话区号映射为相应地区网关的 IP 地址。这些信息存放在一个数据库中，数据接续处理软件将完成呼叫处理、数字语音打包、路由管理等功能。VoIP 系统建设应遵循 5 项基本原则，它们是：延时 400ms 的基本原则、99.999 9% 可靠电信原则，多媒体应用发展原则、网络开放原则、后方管理的保障原则。

在 VoIP 系统中，主要包括提供会话建立的信令协议和提供数据流传输的传输协议两类协议。目前 VoIP 系统中的信令协议主要有 SIP 协议和 H.323 协议，传输协议主要有 RTP 协议和 RTSP 协议，本节将在后续小节中作详细的介绍。

www.ckook.com

5.3.2 H.323 协议

H.323 是 ITU 多媒体通信系列标准 H.32x 的一部分，目的是为现有的分组网络 PBN（如 IP 网络）提供多媒体通信标准。

从整体上来说，H.323 是一个框架性建设，它涉及到终端设备、视频、音频和数据传输、通信控制、网络接口方面的内容，还包括了组成多点会议的设备。H.323 的基本组成单元是“域”，是一个由关守管理的网关、多点控制单元（MCU）、多点控制器（MC）、多点处理器（MP）和所有终端组成的集合。一个域最少包含一个终端，而且必须有且只有一个关守。H.323 还包括了 H.323 终端与其他终端之间的、通过不同网络的、端到端的连接。H.323 的体系结构如图 5-22 所示。

H.323 为基于网络的通信系统定义了 4 个主要的组件：终端、网关、关守和多点控制单元。终端是分组网络中能提供实时、双向通信的节点设备，也是一种终端用户设备，可以和网关、多点接入控制单元通信。所有终端都必须支持语音通信，视频和数据通信可选。

H.323 协议栈从系统的总体框架（H.323）、视频编解码（H.263）、音频编解码（H.723.1）、系统控制（H.245）、数据流的复用（H.225）等方面作了比较详细的规定。其中系统控制协议包括 H.323、H.245 和 H.225.0，Q.931 和 RTP/RTCP 是 H.225.0 的主要组成部分。系统控制是 H.323 终端的核心。整个系统控制由 H.245 控制信道、H.225.0 呼叫信令信道和 RAS（注册、许可、状态）信道提供，音频编解码协议包括 G.711（必选）、G.722、G.723.1、G.728、G.729 等协议。编码器使用的音频标准必须由 H.245 协议协商确定。H.323 终端应具备对本身所具有的音频编解码能力进行非对称操作的能力，如以 G.711 发送，以 G.729 接收。视频编解码协议主要包括 H.261 协议（必选）和 H.263 协议。H.323 系统中视频功能是可选的。数据会议功能也是可选的，其标准是多媒体会议数据协议 T.120。H.323 协议栈的结构如图 5-23 所示。

www.docin.com

图 5-22 H.323 体系结构

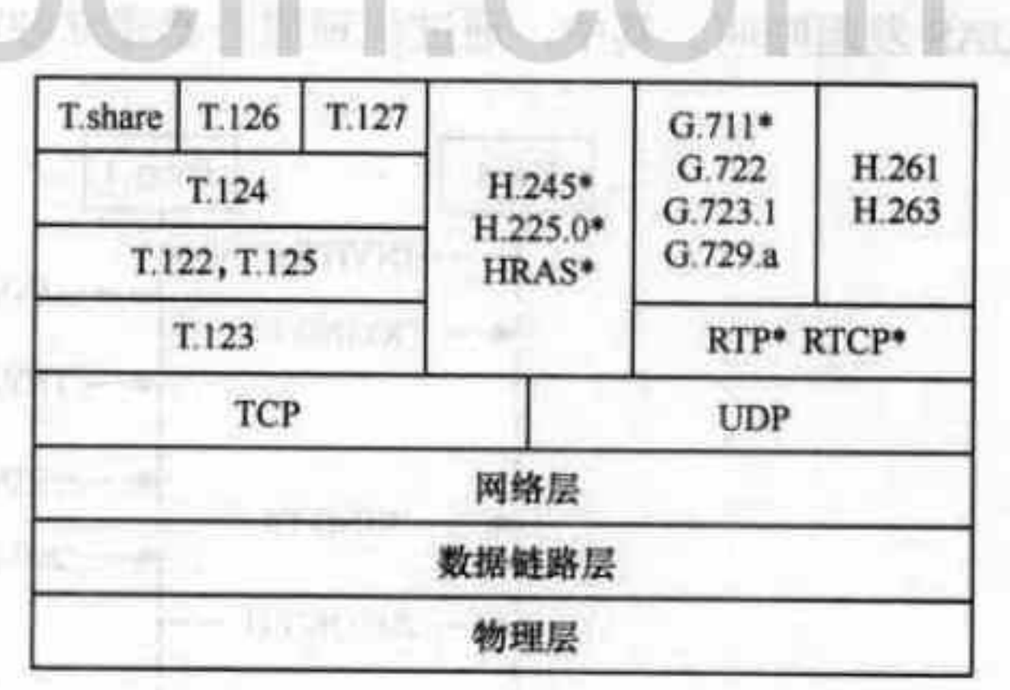


图 5-23 H.323 协议栈

5.3.3 SIP

SIP（Session Initiation Protocol）是由 IETF 定义，基于 IP 的一个应用层控制（信令）协议。

www.ckook.com

由于 SIP 是基于纯文本的信令协议，可以管理不同接入到网络上的会话等。参与会话的成员可以通过多播方式、单播联网或者两者结合的形式进行通信。SIP 协议主要提供以下功能。

（1）名字翻译与用户定位，保证无论主叫方在何处呼叫都能到达被叫方；

（2）特征协商，它允许与呼叫有关的终端在所支持的呼叫的特征上达成一致；

（3）呼叫参与者管理，呼叫中的参与者能够引入其他用户加入呼叫或取消到其他用户的连接；

（4）呼叫特征改变，用户能够在呼叫过程中改变呼叫的特征。

在 SIP 的协议中，定义客房端和服务器两类元素。客房端是指为了向服务器发送请求而与服务器建立连接的应用程序，服务器是指用于向客房端的请求提供服务并回复响应的应用程序。

而在 SIP 系统中，主要包括两类实体：SIP 用户代理和 SIP 网络服务器。用户代理是 SIP 系统中的终端实体，网络服务器是 SIP 系统中处理与多个呼叫相关联信令的网络实体。一般来说，SIP 中的服务器主要有 4 类，如表 5-5 所示。客房端主要存在于用户代理（User Agent）和代理服务器（Proxy）中。代理服务器分为两类，有状态代理服务器和无状态代理服务器。有状态代理服务器记住它接收的请求、回复的响应和它转发的请求，而无状态代理服务器在转发请求后就忘记所有的信息。

表 5-5　　SIP 中的服务器

服务器类型	功能描述
用户代理服务器	当接到 SIP 请求时它联系用户，并代表用户返回响应
代理服务器	代表其他客房端发起请求，既充当服务器又充当客房端的媒介程序。在转发请求之前，它可以改写原请求消息中的内容
重定向服务器	它接到 SIP 请求，并把请求中的原地址映射成零个或多个新地址，返回给客房端
注册服务器	它接收客房端的注册请求，完成用户地址的注册

在 SIP 协议中，支持 3 种呼叫方式：由用户代理客房端（UAC）向用户代理服务器（UAS）直接呼叫、由 UAC 在重定向服务器的协助下进行重定向呼叫和由代理服务器代表 UAC 向 UAS 发起呼叫。其中，通过代理服务器建立呼叫的流程如 5-24 所示。

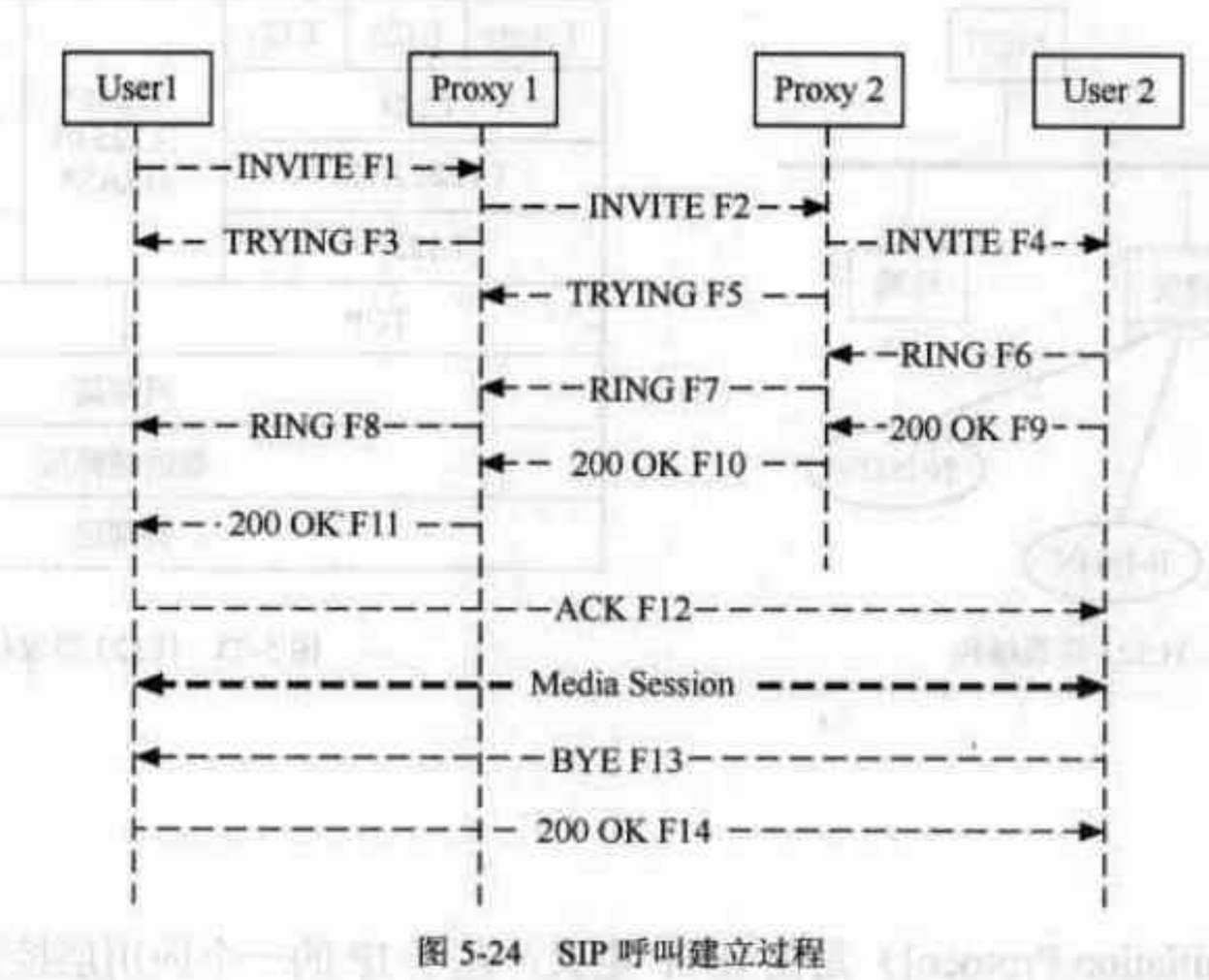


图 5-24　SIP 呼叫建立过程

www.ckook.com

SIP 消息包括两种类型：请求消息和响应消息。请求消息是由客房端发送到服务器，响应消息是由服务器发送到客房端。在请求消息中包含 3 个元素：请求行、消息头和消息体。在响应消息中也包含 3 个元素：状态行、头和消息体。

在 SIP 消息中定义了 6 种方法，分别如下：

（1）INVITE，邀请用户加入呼叫；

（2）BYE，终止呼叫中的两个用户之间的呼叫；

（3）OPTIONS，请求关于服务器能力的信息；

（4）ACK，确认客房端已经接收到对 INVITE 的最终响应；

（5）REGISTER，提供地址解析的映射，让服务器知道其他用户的位置；

（6）INFO，用于会话中的信令。

H.323 和 SIP 分别是通信领域与因特网两大阵营推出的建议，他们支持的业务基本相同，也都利用 RTP 作为媒体传输的协议，其主要区别如表 5-6 所示。总地来说，H.323 沿用的是传统的实现电话信令模式，比较成熟，而 SIP 遵循了因特网一贯坚持的简练、开放、兼容和可扩展等原则，比较简单。

表 5-6　　H.323 与 SIP 的区别

区 别 点	详 细 描 述
协议的编码规则	H.323 采用基于 ASN.1 和压缩编码规则的二进制编码，SIP 采用基于文本的协议的编码
协议的系统结构	H.323 由多点控制单元（MCU）集中执行会议控制功能，所有参加会议终端都向 MCU 发送控制消息，它不支持多播功能；SIP 设计上是分布式的呼叫模型，并且支持分布式的多播功能
协议的扩展业务	H.323 中定义了专门的协议用于补充业务，如 H.450.1、H.450.2 和 H.450.3 等；SIP 没有专门定义协议用于补充业务，但是它可以很方便地支持补充业务或智能业务
协议的呼叫建立过程	在 H.323 中，呼叫建立过程涉及 3 条信令信道：RAS 信令信道、呼叫信令信道和 H.245 控制信道，通过这 3 条信道的协调才使得 H.323 的呼叫得以进行，呼叫建立时间很长；在 SIP 中，会话请求过程和媒体协商过程等一起进行，只需要 1.5 个回路时延来建立呼叫

5.3.4 RTP

RTP（Real-time Transport Protocol）是由 IETF 开发的实时传输协议，可以在面向连接或无连接的下层协议上工作，通常和 UDP 协议一起使用。RTP 的工作机理与 RSVP 不同，主要实现一种端到端的多媒体流同步控制机制，既不需要事先建立连接，也不需要中间节点的参与，为其保留资源。在网络带宽充足的情况下，RTP 具有一定的带宽调控能力，保证端到端的多媒体流同步。在网络带宽不足时，RTP 的带宽调控能力将受到一定的限制。

RTP 定义了两种报文：RTP 报文和 RTCP 报文，RTP 报文用于传送媒体数据（如音频和视频），它由 RTP 报头和数据两部分组成，RTP 数据部分称为有效载荷（payload）；RTCP 报文用于传送控制信息，以实现协议控制功能。RTP 报文和 RTCP 报文将作为下层协议的数据单元进行传输。如果使用 UDP，则 RTP 报文和 RTCP 报文分别使用两个相邻的 UDP 端口，RTP 报文使用低端口，RTCP 报文使用高端口。如果使用其他的下层协议，RTP 报文和 RTCP 报文可以合并，放在一个数据单元中一起传送，控制信息在前，媒体数据在后。通常，RTP

www.ckook.com

是由应用程序实现的。RTP 的报文格式如表 5-7 所示。

表 5-7　　RTP 报文格式

V	P	X	CC	M	PT	序 列 号
时戳（Timestamp）						
同步信源（SSRC）标识符						
特约信源（CSRC）标识符						

在 RTP 协议中，通过 RTCP 报文提供了基于无连接传输协议的端到端控制机制，这是一种基于接收者反馈的网络传输 QoS 监测机制。在 RTCP 的接收报告中包含了当前网络传输 QoS 有关信息，如报文丢失率、报文丢失累计、接收到的最高序列号、平均延迟抖动以及用于计算发布接收报告往返所需时间的时间标签等。发送者可通过这些信息监测和评价网络传输 QoS 状况，并可采取适当的策略实施同步控制。

RTP 协议规定，每个 RTP 系统必须实现 RTCP 的控制功能，由内部功能模块定期自动执行。RTCP 报文是轻载信息，其信息量与最低的数据通信量相平衡，它所产生的通信量只是数据通信量的 5%左右。

5.3.5　RTSP

RTSP（Real Time Streaming Protocol）是由 RealNetworks 和 Netscape 共同提出的，该协议定义了一对多应用程序如何有效地通过 IP 网络传送多媒体数据。RTSP 在体系结构上位于 RTP 和 RTCP 之上，它使用 TCP 或 RTP 完成数据传输。RTSP 是应用级协议，控制实时数据的发送。RTSP 协议提供了一个可扩展框架，使实时数据，如音频与视频的受控、点播成为可能。数据源包括现场数据与存储在剪辑中的数据。该协议目的在于控制多个数据发送连接，为选择发送通道（如 UDP、多播 UDP 与 TCP）提供途径，并为选择基于 RTP 上发送机制提供方法。

RTSP 协议建立并控制一个或几个时间同步的连续媒体流。尽管连续媒体流与控制流交叉是可能的，通常它本身并不发送连续流，即 RTSP 充当多媒体服务器的网络远程控制。RTSP 连接没有绑定到传输层连接，如 TCP。在 RTSP 连接期间，RTSP 用户可打开或关闭多个对服务器的可靠传输连接以发出 RTSP 请求。RTSP 操作并不依赖用于携带连续媒体流的传输机制。RTSP 协议在语法和操作上与 HTTP/1.1 类似，因此 HTTP 的扩展机制大都可加入 RTSP。

RTSP 在功能上与 HTTP 有重叠，与 HTTP 相互作用体现在与流内容的初始接触是通过网页。但是，RTSP 与 HTTP 的本质差别在于数据发送以不同协议进行。HTTP 是不对称协议，用户发出请求，服务器作出响应。而在 RTSP 中，媒体用户和服务器都可发出请求，且其请求都是无状态的；在请求确认后很长时间内，仍可设置参数，控制媒体流。

RTSP 协议的特性如表 5-8 所示。

表 5-8　　RTSP 协议特性

协 议 特 性	特 性 描 述
可扩展性	新方法和参数很容易加入 RTSP
易解析	RTSP 可由标准 HTTP 或 MIME 解析器解析
安全	RTSP 使用网页安全机制

www.ckook.com

续表

协议特性	特性描述
独立于传输	RTSP 可使用不可靠数据报协议（UDP）、可靠数据报协议（RDP），如要实现应用级可靠，可使用可靠流协议
多服务器支持	每个流可放在不同服务器上，用户端自动同不同服务器建立几个并发控制连接，媒体同步在传输层执行
记录设备控制	协议可控制记录和回放设备
流控与会议开始分离	仅要求会议初始化协议提供，或可用来创建唯一会议标识号。特殊情况下，SIP 或 H.323 可用来邀请服务器入会
适合专业应用	通过 SMPTE 时标，RTSP 支持帧级精度，允许远程数字编辑
演示描述中立	协议没强加特殊演示或源文件，可传送所用格式类型；然而，演示描述至少必须包含一个 RTSP URI
代理与防火墙友好	协议可由应用和传输层防火墙处理。防火墙需要理解 SETUP 方法，为 UDP 媒体流打开一个“缺口”
HTTP 友好	RTSP 明智地采用 HTTP 观念，使现在结构都可重用。结构包括 Internet 内容选择平台（PICS）。由于在大多数情况下控制连续媒体流需要服务器状态，RTSP 不仅仅向 HTTP 添加方法
适当的服务器控制	如用户启动一个流，他必须也可以停止一个流
传输协调	实际处理连续媒体流前，用户可协调传输方法
性能协调	如基本特征无效，必须有一些清理机制让用户决定那种方法没生效。这允许用户提出适合的用户界面

5.4 P2P 和 SIP 的结合

上一节对 VoIP 和 SIP 的相关概念做了简要的描述，本节从一个全新的角度来向读者介绍 SIP 的应用。凭借协议本身的简单性和易扩展性，SIP 已经成为 VoIP 的两个协议之一（另一个是 H.323），而 VoIP 是 NGN（下一代网络）的一个重要组成部分，因为它可以提供丰富的业务和低廉的费用。根据 IResearch 的资料显示，VoIP 服务市场规模在 2004 年已经达到 35 亿美元，而且还在飞速增长，预计到 2008 年能达到 210 美元。

在高速增长的同时，SIP 也开始暴露出一些问题。传统的 SIP 系统采用的是客房端/服务器的模式，随着用户数量的增大，单点失效和性能瓶颈显得越来越突出。解决单点失效问题一般是通过冗余服务器来实现的，但当冗余服务器数量不足时，解决单点失效问题并不是很有效，冗余服务器数量过多时却会使各个服务器之间的同步变得非常复杂，这就带来了另一个问题——性能瓶颈。如何在单点失效和性能瓶颈之间折中是一个比较困难的问题。客房端/服务器的模式使得部署一个 SIP 系统需要比较多的花费，而且需要大量的资源去维护服务器，对运营商来说是一个很大的负担。

Skype 在这些方面做得很好，它使用 P2P 模式解决了 SIP 所不能解决的问题。在 Skype 网络中，网络节点被划分为超级节点和普通节点两种，中心服务器只负责用户的注册、登录和分配超级节点。超级节点之间通过 P2P 的机制互联，普通节点连接到超级节点，利用超级节点来查找通信的另一端。当超级节点失效时，该节点管理的普通节点会被转移到其他超级节点上，从而很好地避免了单点失效问题；而且超级节点和普通节点之间可以根据网络的负载相互转化，不存在性能瓶颈。Skype 的这种网络结构使得它的运营商只要维护很少的中心服务器的运行就可以管理整个网络。据统计，Skype 网络的同时在线人数已经突破了 700 万，但通信质量仍旧能得到很好的保证。

www.ckook.com

Skype 的成功使得 IETF 开始考虑利用 P2P 技术来改造 SIP。P2P 的优点是网络中不存在中心服务器或者中心服务器只负责一些网络的附属功能。这个特征使得 P2P 网络的节点之间可以比较均匀地负担负载，通过增加节点就可以增加系统容量，而且这种增加节点所需要的操作几乎是“零花费”的。利用具有高扩展性、高顽健性和高容错性的 P2P 网络来改造 SIP，可以很显著地提高 SIP 的性能。

5.4.1 P2P 和 SIP 结合方案分析

P2P 和 SIP 结合，有下面两种方式可以选择。

（1）P2P over SIP

通过扩展 SIP 信令来构建 P2P 网络，并且利用已有的 SIP 信令来简化 P2P 节点之间的消息传递。例如使用 SIP 的注册请求来加入 P2P 网络，使用其他请求和回应来在 P2P 网络中进行通信。威廉玛丽学院的 David A. Brayn 就提出了扩展 SIP 的 Register 请求，将其用作 P2P 的信令。不过 P2P over SIP 会增加 SIP 协议本身的负载和延迟，不利于系统的扩展。

（2）SIP over P2P

以 P2P 网络来承载 SIP 消息的传输。传统的 SIP 消息是承载在 TCP 或者 UDP 数据包里，使用 DNS 服务来进行定位。而近些年研究较多的基于 DHT 的 P2P 网络可以提供很好的资源定位和数据传送服务，所以一些学者也开始研究基于 SIP over P2P 的模式。美国哥伦比亚大学的 Kundan Singh 和 Henning Schulzrinne 在《Peer to Peer Internet Telephony Using SIP》一文中就提出了一个基于 Chord 和 SIP 的 IP 电话系统。

从上面两种方式对比来看，P2P over SIP 需要对 SIP 协议做较多的扩展，而且基于 SIP 的 P2P 系统和当前的 P2P 系统之间很难相互兼容，实现这样的系统比较困难；而 SIP over P2P 只需要把 SIP 消息封装在已有 P2P 系统的消息里传送，而且可以和已有的 SIP 系统兼容。鉴于这些原因，我们选择使用 SIP over P2P 来实现 IP 电话系统。由于基于 DHT 的 P2P 网络可以提供很好的资源定位服务，所以是和 SIP 结合的最好选择。本章以后提到的 P2P 均指基于 DHT 技术的 P2P。

在介绍 SIP 协议的时候我们提到，SIP 网络包含两类组件：用户代理和服务器。用户代理是给是 SIP 系统的客户端，用来发起呼叫和对呼叫作出响应；服务器为用户代理提供注册、认证、鉴权和路由等服务。SIP 系统的所有消息都是作为 TCP 或者 UDP 的数据包进行传送的，要把 SIP over TCP/UDP 改为 SIP over P2P，只需要用 P2P 网络提供的传输功能来替代底层的 TCP 和 UDP。

从网络结构上来说，SIP over P2P 可以有 3 种可行的运行方案。

第一种方案是只把所有的 SIP 服务器用基于 DHT 的 P2P 网络连接起来，用户代理只要知道一个服务器的地址就可以接入网络。注册时，每个用户和传统的 SIP 系统一样把自己的信息发送给服务器，服务器接着用 Hash（根据具体的 P2P 网络而定）算法计算该用户的 ID,记为 Key_{user}，然后以 Key_{user} 作为键值把该用户的信息发送给适当的服务器（一种可行方案是服务器的 ID 和 Key_{user} 之间差值最小）。这种方案不需要对客户端作出改动，只需要运营商重新配置服务器之间的连接关系，以 DHT 代替传统的 DNS 服务。此方案的结构示意图如图 5-25 所示。

虽然这种方案采用了 P2P 网络来管理服务器，但是并没有对现有网络作出显著的改变，因而对提升系统的吞吐量没有太大的帮助。在 SIP 系统中，服务器之间本来是一种松耦合的关系，每个服务器只负责自己单独的域，各个域服务器之间通过 DNS 服务来定位。如果采用

www.ckook.com

上述方案，各个服务器之间的耦合度反而有所增加，不利于系统的扩展。因此这种方案不太可能成为未来的发展趋势。

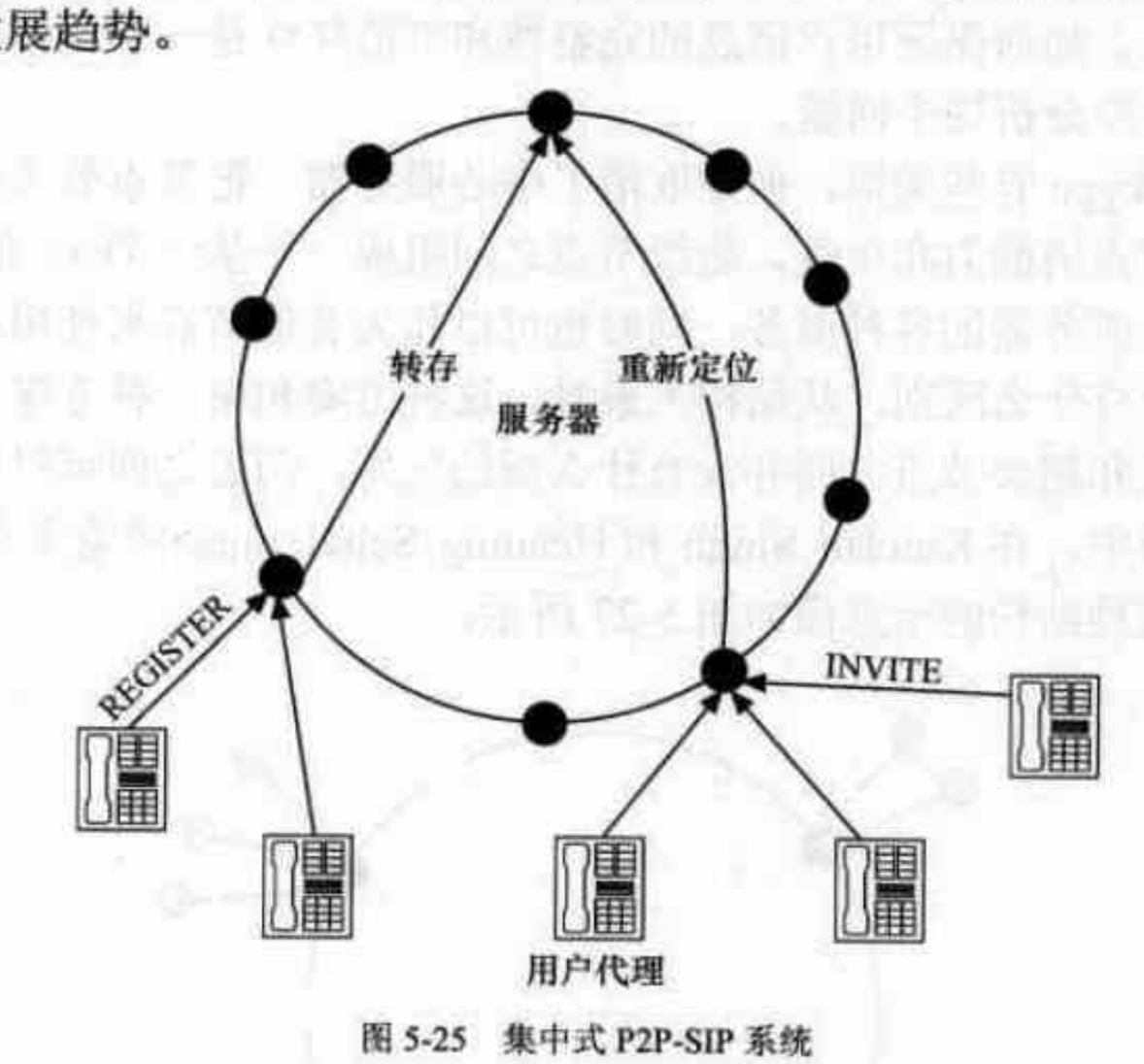


图 5-25 集中式 P2P-SIP 系统

第二种方案不再区分服务器和用户代理，它颠覆了传统的 SIP 系统的结构。这种方案中，每个 P2P-SIP 节点既承担服务器的功能，给其他节点提供注册等服务，同时也承担用户代理的功能，构成一个纯 P2P 结构的 SIP 系统。注册时，每个用户都用固定的 Hash 算法计算出自己用户名散列值，记为 Key_{user}，然后以 Key_{user} 为键值根据底层 P2P 网络的规则把自己的注册信息存储在某个节点上。当某个用户要邀请别的用户聊天时，首先用同样的算法得到被邀请方的 Key_{user}，然后把 INVITE 消息发送给存储被邀请用户信息的节点，存储被邀请用户信息的节点要负责把 INVITE 消息转发给被邀请用户的节点，即存储被邀请用户信息的节点充当了 SIP 服务器的角色。这种方案充分利用了 P2P 网络高扩展性、高可靠性和高容错性的特点，易于实现，是未来的一个发展方向。此方案结构的示意图如 5-26 所示。

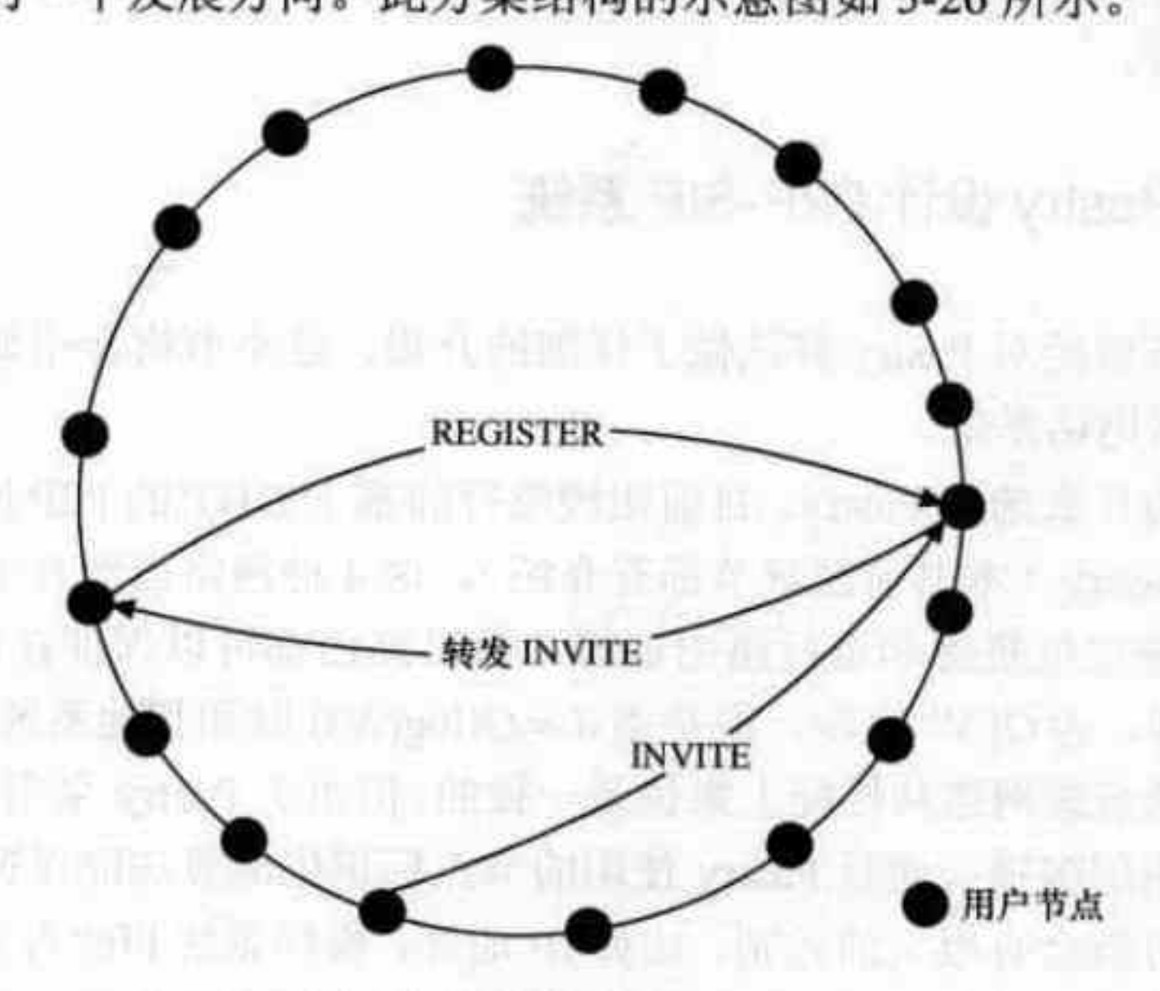


图 5-26 完全分布式 P2P-SIP 系统

www.ckook.com

不过完全的 P2P 结构带来了新的问题，那就是身份认证。由于每个用户需要把自己的信息保存在别的节点上。如何保证用户信息的完整性和可信任性是一个比较有挑战的问题。本节的后续部分将进一步分析这个问题。

第三种方案和 Skype 有些类似，但是取消了中心服务器，把节点分为超级节点和普通节点，划分的标准是节点的能力和负载。超级节点之间组成一个基于 DHT 的 P2P 网络，负责给普通节点提供 SIP 服务器的各种服务，同时也可以作为普通节点来使用；普通节点和现有 SIP 系统的客户端没有什么区别。从结构上来看，这种方案和第一种方案有些类似，但最大的不同在于普通节点和超级节点之间并没有什么质的区别，两者之间可以相互转化，可以说是对前两种方案的折中。在 Kundan Singh 和 Henning Schulzrinne 的论文里提到的 IP 电话系统就是这种结构。这种结构的示意图如图 5-27 所示。

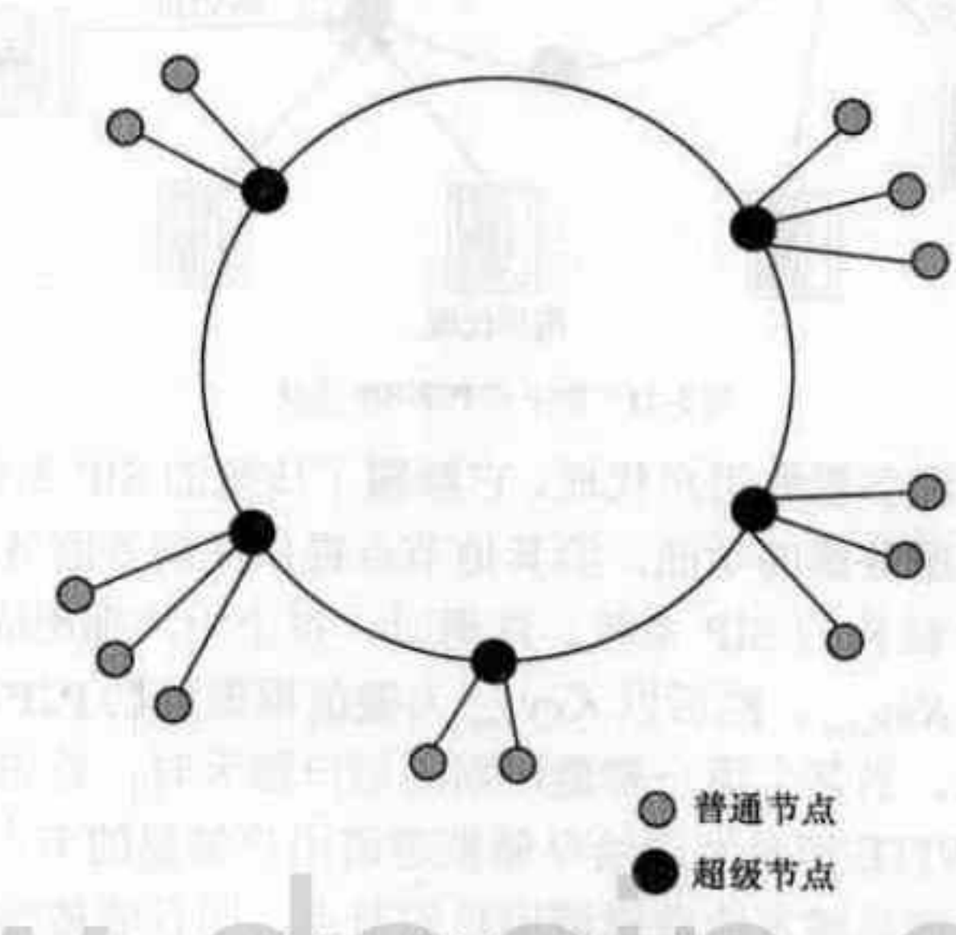


图 5-27 混合式 P2P-SIP 系统

www.docin.com

这种方案的优点在于可以充分利用节点的性能，但这同时增加了网络的复杂度，实现上不如第二种方案容易。

5.4.2 基于 Pastry 设计 P2P-SIP 系统

本书的前面章节曾经对 Pastry 算法做了详细的介绍，这小节将介绍如何基于 Pastry 来构建一个 P2P-SIP 网络电话系统。

首先说明一下为什么选用 Pastry。目前比较流行的基于 DHT 的 P2P 网络有 4 种：Pastry、CAN、Chord 和 Tapstery（本书前面章节都有介绍）。这 4 种网络虽然有不同的设计思想，但都使用 DHT 算法来定位资源和进行路由选择，而且路由都可以保证在 $O(\log(N))$ 步内完成（CAN 有些不太一样，为 $O(N^{1/d})$ 步，但是当 $d=O(\log(N))$ 时和其他系统一样）。因此，选择哪个网络作为 SIP 的承载网络从性能上来说是一致的。但由于 Pastry 采用了前缀匹配的算法，能很方便地实现应用层多播；而且 Pastry 使用的节点标识生成算法能保证在标识空间里相邻的两个节点在底层可能会有很大的差别，比如 IP 地址、操作系统和所有者权限等方面，这可以提供很好地容错性能。关于 Pastry 详细的介绍参见本书的前面章节。这里给出一个典型的

www.ckook.com

Pastry 路由表示意图，如图 5-28 所示。图中的一个圆点代表一个 Pastry 节点，一个节点和另一个点之间有线连接，代表这个节点的路由表中有到另一个节点的路由项。

从图 5-28 中还可以看出一个 Pastry 节点可以有到全球各地节点的路由项，而且在路由项中到较近节点的项比较多，这说明 Pastry 的路由项既有广泛性，也具有良好的局部特性。

1．用 Pastry 来支持 SIP 的基本功能

前面提到传统的 SIP 系统中有用户代理和服务器之分，而如果要构建纯 P2P 的 SIP 系统，就需要以合适的方式把服务器完成的功能分布到整个网络中。基于 Pastry，我们可以构建一个类似分布式存储的系统，把每个用户的注册信息存储在特定的一个或者多个节点上。所有的节点都可能会存储别的用户信息，因而每个节点都要具有 SIP 服务器的功能。节点 A 第一次启动时，首先计算用户输入用户名的 Hash 值（在 Pastry 系统中，可以用 SHA-1 算法来计算），记为 Key_{user}，然后用 Pastry 的 API route（msg, key），以 Key_{user} 作为 key 参数，以用户的注册消息作为 msg，把用户的注册消息发送到节点 ID 和 Key_{user} 最接近的节点（依据 Pastry 的规则）B。B 节点收到 A 节点的注册消息后，就开始充当 A 的注册服务器，负责处理目的地为 A 节点的呼叫请求和回应。节点 C 如果要向 A 发起通话邀请，首先要通过计算得到节点 A 用户的 Key_{user}，然后以 Key_{user} 为 key 参数，以 INVITE 消息为 msg 参数， 把邀请发送到 B 节点，B 节点接着把邀请转发给 A 节点，并给 C 回送临时回应消息。A 节点的用户如果接受此次邀请，就把 ACK 消息发送到 B 节点然后由 B 节点转发给 C 节点，然后 C 节点再回送 ACK 消息。这样就完成了 SIP 连接的三次握手，再往后 A 和 C 节点就可以按照约定的规则进行通信了，这个过程的示意图如图 5-29 所示（为了更好地描述 SIP 信令发送的流程，图中没有画出 Pastry 网络消息路由的过程）。

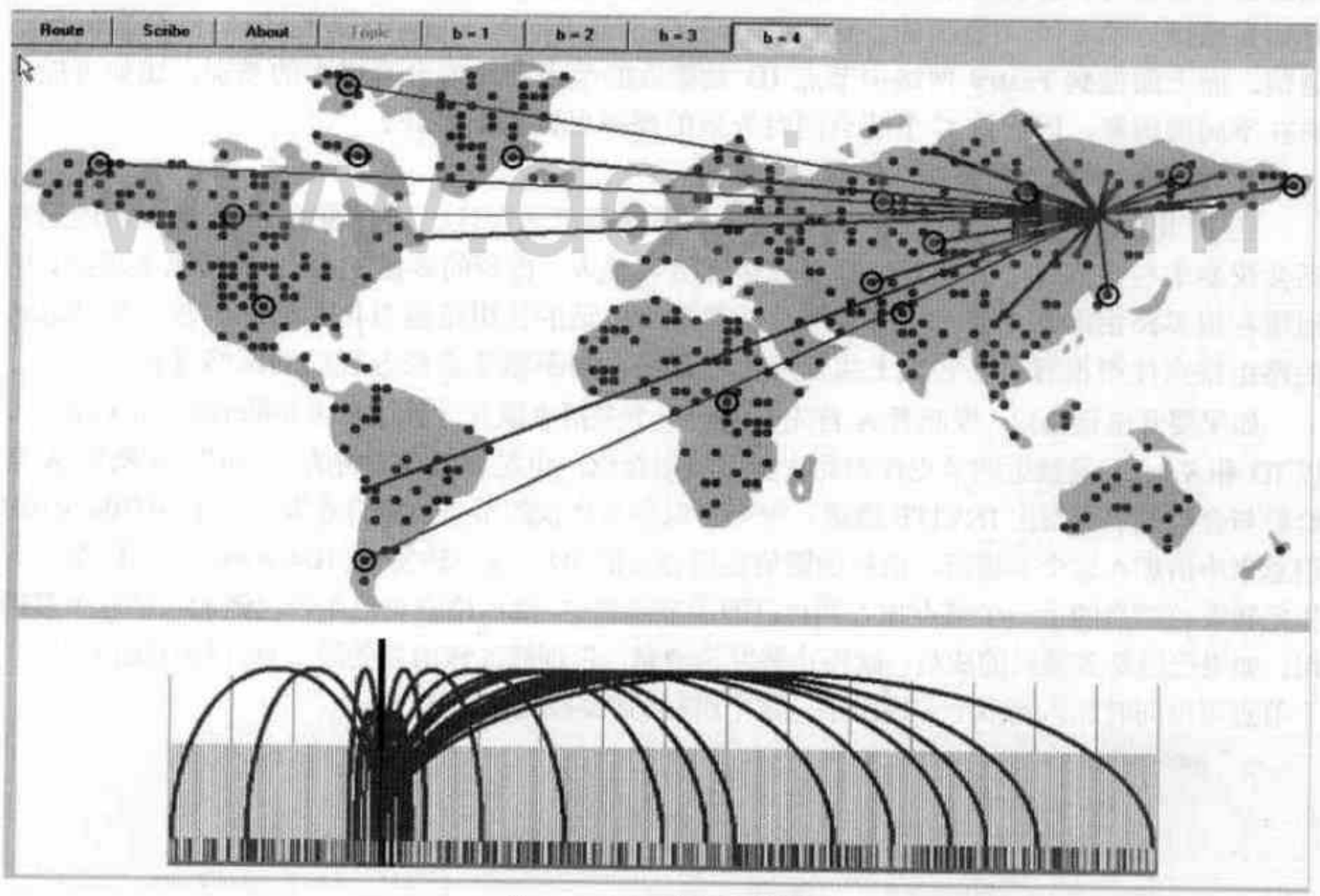


图 5-28　Pastry 路由项示意图

www.ckook.com

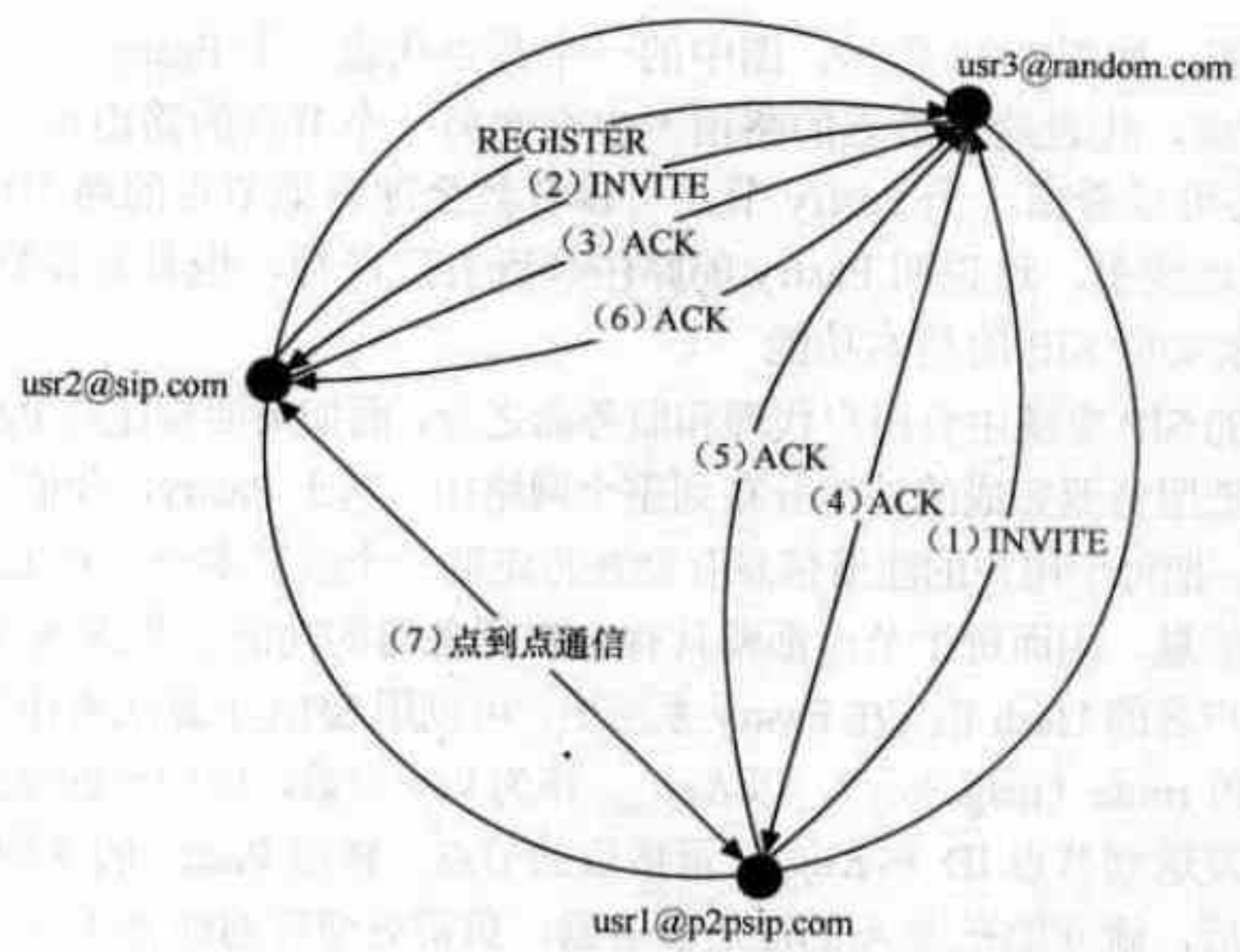


图 5-29 基于 Pastry 的 P2P-SIP 系统建立连接示意图

图中 usr@sip.com 是 A 节点的用户名，usr1@p2psip.com 是 C 节点的用户名，usr3@random.com 是 B 节点的用户名。

这种注册方式有一个潜在的问题就是如果 B 节点失效，A 和 C 之间的通信就没有办法正常进行，类似于“单点失效”。P2P 网络中的节点是可以随时加入或离开网络的，因此上面的问题会经常出现。解决这个问题可以采用类似冗余备份的方式，节点注册时是把自己的信息存储在最接近的 C 个节点上的，这样除非这 C 个节点同时失效才可能造成 A 和 C 之间无法通信。而上面提到 Pastry 网络中节点 ID 最接近的节点在底层会有很大的差别，比如可能分布在不同的国家，因而这 C 个节点同时失效的概率可以忽略不计。

2. 用 Pastry 来支持多播

在 IP 电话中有一项用途很广泛的功能，那就是电话会议。和普通的 IP 电话不同的是电话会议要求与会的每个人的发言都要被传送给其他人。传统的多播是由 IP 多播来实现的，然而现在很多路由器都不支持 IP 多播，使得网络电话的使用范围有很大的局限性。而 Pastry 的路由特点使得很容易在它之上实现应用层多播，而不需要去修改底层的网络设备。

如果要开电话会议，发起者 A 首先要创建一个多播主题并计算这个主题的 Hash 值 Key_{topic}，以 ID 和 Key_{topic} 最接近的节点作为此次会议的集合点，也是会议多播树的“root”，紧接着 A 要给参与会议的节点发出 INVITE 邀请。每个可以参与会议的节点给集合点发送一个 SUBSCRIBE 消息来申请加入这个多播组，消息的键值是集合点的 ID。每个转发 SUBSCRIBE 消息的节点首先把转发该消息的上一个节点加入到自己的子节点组中，然后检查自己是否已经加入了这个多播组，如果已经是多播组的成员，就停止转发该消息，否则修改该消息的源节点并转发该消息。一个节点可以同时加入到多个多播组中。这个过程的代码表示如下所示。

```
forward(msg, key, nextId)
    switch msg.type is
        SUBSCRIBE :
            if !(msg.topic ∈ topics)
                topics = topics ∪ msg.topic
```


www.ckook.com

```
            msg.source = thisNodeId
            route(msg,msg.topic)      //把消息的源节点修改后发送
            topics[msg.topic].children ∪ msg.source
        nextId = null //停止转发原来的消息
```

topics 是这个节点目前加入的所有的多播组主题的集合，每个 SUBSCRIBE 消息里也有一个主题的标识（msg.topic）。

经过这个过程，一个多播树很快就建立了起来。观察这个建立过程，会发现有很多没有收到邀请的节点也加入了这个多播组，我们把这些节点称为 Router。它们只在底层负责多播组的消息传播功能，并不向上层发送数据。一个多播树的示意图如图 5-30 所示。

图中所有画圆圈的点都属于这个多播组的成员(左上角那个除外，它是空主题的集合点)。图中创建的主题是“P2P 技术”。边上标有“Router”的点就是那些被动加入多播组的节点，即节点只负责转发消息，并不向上层发送数据。

考虑到 P2P 网络中的节点随时可能失效，这个多播树也可能会出现不同程度的断裂。为了保持状态，父节点要定时向子节点发送心跳信息。如果某个节点过了一段时间没有收到父节点的心跳信息，就自动生成一个 SUBSCRIBE 请求，重新加入多播组，这样多播树就重新组织了。在多播树的修复期间，可能会有一些消息丢失，但考虑到语音通信对丢包不是很敏感，这点是可以容忍的。

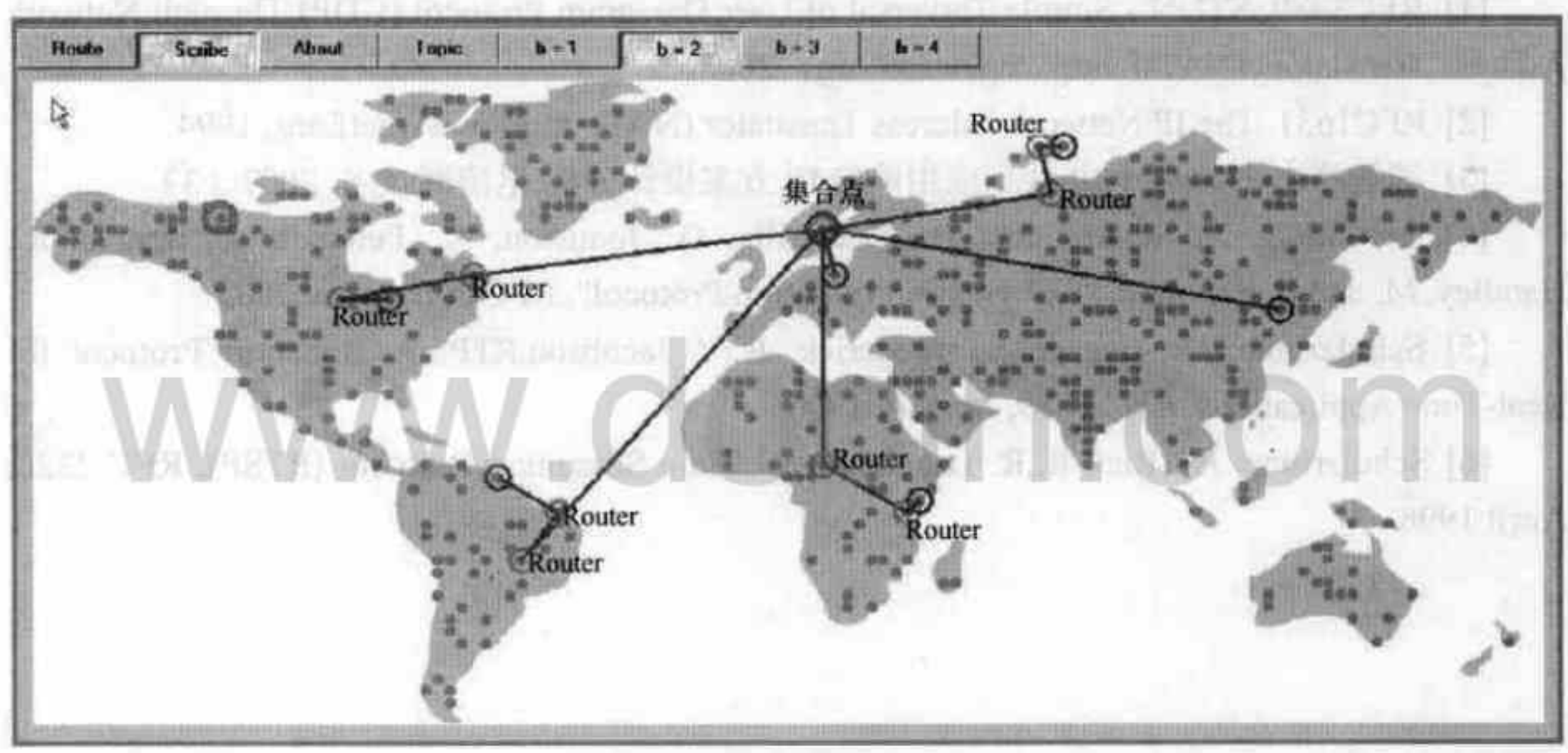


图 5-30 多播树示意图

多播树建立好以后，电话会议就可以开始了。每个发言者都把消息发送到集合点，然后由集合点把消息向整个多播树传播。Pastry 良好的局部特性能使得每次消息转发都选择最短的路径，这点可以最大限度地保证语音流的实时性。

5.5 本章总结

穿越 NAT 实现 P2P 通信是实现 P2P 应用要解决的关键问题。本章讲解了 NAT 网络的基

www.ckook.com

本原理，了解了 NAT 网络的基本原理就可以很清楚为什么这样可以穿越 NAT 网络。重点讲述了 STUN 协议在 NAT 穿越中的应用，讲述了 NAT 分类的原则，并给出了穿越 NAT 实现 TCP 和 UDP 的两个例子。然后简述了几个常见的 NAT 穿越方案。最后简述了 P2P 技术与 IMS 和智能技术的结合。

5.6 练习题

（1）阐述在 NAT 网络中内网主机访问外网的过程。
（2）STUN 协议将 NAT 分为哪几种类型？
（3）怎样穿越 NAT 实现 TCP 通信？
（4）怎样穿越 NAT 实现 UDP 通信？
（5）简述 P2P 在 IMS 中的应用。
（6）简述 P2P 与 VoIP 技术的结合。

参 考 文 献

[1] RFC3489. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). http://www.ietf.org，2003

[2] RFC1631. The IP Network Address Translator (NAT). http://www.ietf.org, 1994

[3] 姜智峰.一种基于 P2P 共享应用的 IMS 方案设计分析.电信网技术, 2007（3）

[4] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002

[5] Schulzrinne, H., Casner, S., Frederick, R.,V. Jacobson.RTP: A Transport Protocol for Real-Time Applications, RFC 1889, January 1996

[6] Schulzrinne, H., Rao, R.,R. Lanphier.Real Time Streaming Protocol (RTSP), RFC 2326, April 1998

www.docin.com

www.ckook.com

C++开发案例篇

www.docin.com

www.ckook.com

www.docin.com

www.ckook.com

CHAPTER 6

第 6 章　P2P 应用开发平台

Windows Peer-to-Peer Networking 是微软公司推出的一套 C/C++语言 P2P 应用开发平台，而 JXTA 是 SUN 公司推出的一个基于 Java 的 P2P 应用开发平台。本章将分别介绍 Windows Peer-to-Peer Networking 平台、JXTA 规范和 Java 平台的平台结构及基本概念。本章还会通过 Windows Peer-to-Peer Networking 平台与 JXTA 平台的实现例子来向读者简要介绍两个平台的开发。

在学习了本章后，希望读者能够掌握以下几点：

（1）WPPN 和 JXTA 的系统框架；

（2）WPPN 和 JXTA 平台中的基本概念；

（3）搭建 WPPN 和 JXTA 开发环境；

（4）利用 WPPN 和 JXTA 开发简单的 P2P 应用程序。

6.1　Windows Peer-to-Peer Networking 平台

www.docin.com

6.1.1　Windows Peer-to-Peer Networking 平台简介

随着 P2P 技术的发展，越来越多的公司开始注意到 P2P 技术的巨大威力。微软公司也在不久前推出了视窗系列平台下的点对点的网络接口平台——Windows Peer-to-Peer Networking（以下简称 WPPN）。WPPN 是可以在众多视窗平台下使用的一组点对点开发平台，它支持 Windows XP with Service Pack 2，Windows XP Professional x64 Edition，Windows XP with Service Pack 1 and the Advanced Networking Pack for Windows XP，or Windows Vista。本章将介绍 WPPN 的相关概念、结构与工作机制，并解析 WPPN 开发的代码实例。

1．什么是 Windows Peer-to-Peer Networking 平台

微软公司对于 WPPN 的目标定义为：使得人们可以在不依靠于一个中心服务器的情况之下安全地通信或者共享信息，但是如果网络中存在一个中心服务器则通信或者信息的共享会更加地顺畅的一套开发库。Windows

www.ckook.com

点对点网络平台本质上提供了一系列的 API 函数，这些函数不仅功能强大而且相当地灵活。这其中主要的模块包括：

（1）点对点名字解析模块；

（2）点对点图模块；

（3）点对点组模块；

（4）点对点身份管理模块。

在后面的章节中我们会对这些模块作更详细的介绍。

2．为什么选择 Windows Peer-to-Peer Networking 平台

WPPN 为开发者提供了一套完整的开发点对点网络应用程序的解决方案。P2P 应用程序开发者可以很方便地使用点对点名字解析、点对点图、点对点组、点对点身份管理等 API 函数创建新的、令人兴奋的、健壮的点对点网络应用程序。WPPN 最大的优点在于，它使得开发者可以集中精力实现他们的想法而不是把精力花费在具体的网络问题上。

WPPN 有以下几个特色使其成为开发者的选择。

（1）安全性：即使在有网络错误或者网络攻击的情况下，WPPN 平台也保证了点对点网络的顽健性。

（2）可扩展性：WPPN 平台适用于从小型的 Ad hoc 网络到整个 Internet 网络等各种规模的网络环境。

（3）无服务器结构：WPPN 平台结构中没有中心服务器，所以不会存在对中心服务器的依赖。

（4）自适应性：WPPN 程序能够自己根据不同的网络情况调整。

（5）自修复性：当网络存在错误时，WPPN 程序能够自己修复和修正错误。

（6）共享性：WPPN 实现了各个节点间的资源共享。

采用 WPPN 平台可以方便地开发以下类型的点对点网络应用程序。

（1）实时的网络通信与聊天工具；

（2）需要网络协作的程序；

（3）资源共享程序；

（4）分布式处理程序。

www.docin.com

6.1.2 Windows Peer-to-Peer Networking 平台结构

1．平台结构

如图 6-1 所示，WPPN 平台是构建在 Windows 网络协议栈之上的开发平台，WPPN 平台的各个模块都需要下层的 WinSock API、Crypto API 以及 TCP/IP 协议栈提供支持。WPPN 平台通过 Win32 API 的方式向上层提供开发接口。

WPPN 平台包括以下一些模块。

图（Graphing）：图在 WPPN 平台中是负责维护对等点的集合。从图 6-1 中可以看出，图模块包含了洪泛与同步、存储、图维护 3 个子模块。

组（Grouping）：组模块是构建在图之上的一个安全层，这个安全层定义了一个组在建立、邀请以及连接时的安全模型。另外，组模块使用 PNRP 作命名解析，这样多个 P2P 应用程序

www.ckook.com

可以分享同一个图。组模块包括组安全、组安全服务等两个子模块。

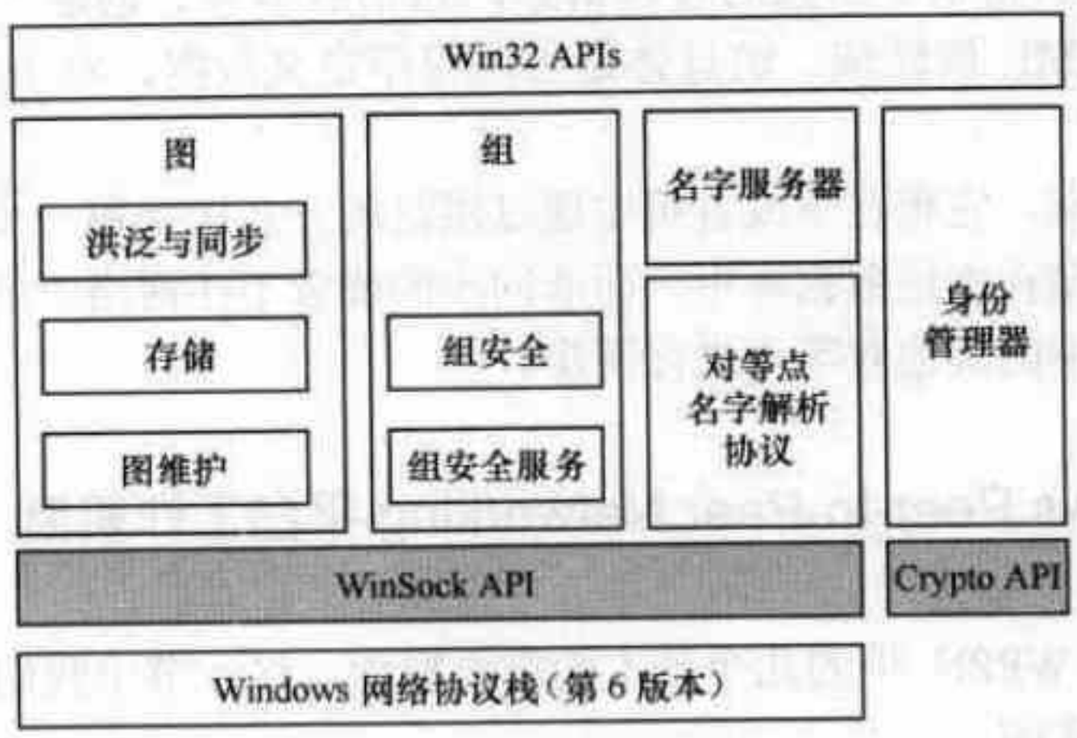


图 6-1 WPPN 平台结构示意图

NSP：The Name Service Provider，名字服务器，此模块提供了访问任意的名字服务器的机制。在 WPPN 平台中，P2P 的应用程序使用 NSP 的接口来访问 PNRP。

PNRP：PNRP 模块提供点对点的名字解析。

身份管理器（Identity Manager）：身份管理器管理着点对点网络中的身份创建、管理。

2．基本概念

介绍了 WPPN 平台的结构之后，下面再介绍 3 个 WPPN 中重要的概念，熟悉这些概念便于读者理解 WPPN 的工作机制。

（1）对等点名称（Peer Names）

对等点名称是 PNRP、身份管理器都会使用到的一个概念。一个对等点名称用于标识一个资源，例如一台计算机、一个用户、一个组或者一项服务。PNRP 使用对等点名称在一个 P2P 网络中寻找一个确定的节点。一个对等点名称的长度为 128bit。

WPPN 中一个对等点名称有这样的格式：Authority.Classifier。其中 Authority 的值取决于这个对等点名称是否是安全的，对于一个安全的对等点名称来说，Authority 值是对等点的名字经过一个散列运算获得，对于一个不安全的对等点名称 Authority 的值则取 0。Classifier 值是一个包含不大于 15 个字符的字符串，采用 Unicode 编码。下面是 3 个合法的对等点名称的例子。

① 0.MyUnsecuredPeer.Name

② 0.JohnDoe.Games

③ 6520c005f63fc1864b7d8f3cabebd4916ae7f33d.JohnDoe.secure

（2）域（Clouds）

域是点对点图模块与组模块中使用到的一个概念。PNRP 中定义域为可以在同一个 IPv6 环境下通信的对等点的集合，以下是域的一些特点。

① 域通过名字来标识，可访问的域可以通过 WSALookupServiceBegin 函数来列取。

② 如果计算机接入到因特网中，则是全局域的一部分，使用“Global_”标识。

③ 可以通过 PNRP 在特定的域中注册和解析对等点名称。

（3）消息（Records）

www.ckook.com

消息是图中各个节点或者组中各个成员相互通信的单位。一个消息包括两个部分：消息头、消息体。消息头记录了消息的各种信息，包括版本号、创建人、消息体的类型。消息头还包含了一个 XML 属性域。消息体是应用程序定义数据，对于 P2P 网络平台来说是不透明的。

当一条消息被发送，它将在图或者组中通过洪泛的方式送到每一个节点。同步的对等点会接收消息并把它存储在本地数据库中，而非同步的或者不在网络中的对等点会在下一次接入到网络时与图和组中的其他对等点进行同步。

6.1.3 Windows Peer-to-Peer Networking 平台工作机制

上一节中介绍了 WPPN 中的几个基本的重要概念，这一节中我们结合这些概念来分析 WPPN 平台是如何工作的。

1．对等点名字解析协议

为了能与其他的节点相互通信，点对点网络中的节点必须能发掘其他节点的存在并能够通过其名称或者其他标识解析出它的网络地址、端口等信息。在点对点网络中，这一过程是不可能通过中心服务器来完成的，例如 DNS。事实上，WPPN 是通过对等点名字解析协议（PNRP）来完成这一过程的。

PNRP ID 是 PNRP 中的基本概念，它是从前一节我们介绍的对等点名称派生出来的。如图 6-2 所示，PNRP ID 为 256bit，其中高 128bit 为对等点名称，低 128bit 称作服务位置标记。在 WPPN 中，对等点名字（Peer Names）标识的是一台计算机或者网络设备，而 PNRP ID 可以标识运行在计算机上的一个服务，因为服务位置标记的不同，同一台计算机可能只有一个对等点名称，但是却可以有多个 PNRP ID。

www.docin.com

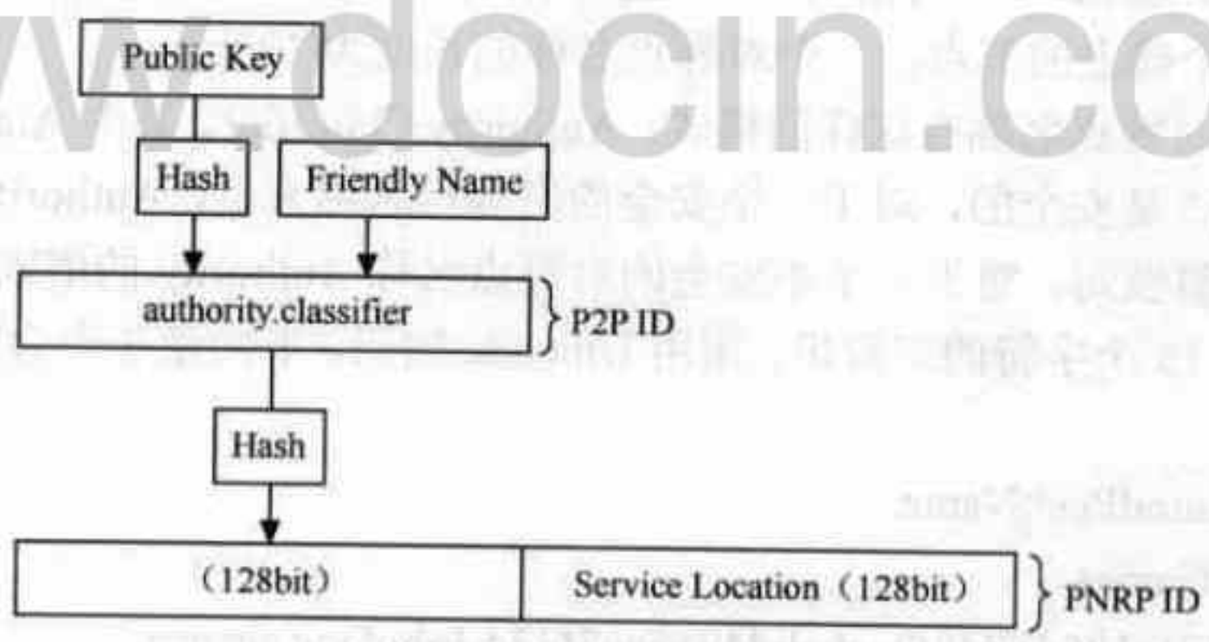


图 6-2　PNRP ID 结构图

当一个对等点知道运行在另一台计算机或者网络设备上的一个服务的 PNRP ID，并且希望参与到这项服务当中时，它必须与运行这个服务的设备取得联系，简单说就是它需要获得这台设备的 IP 地址和这项服务的端口号。在 WPPN 中，这一过程也是通过离散散列表的方式来完成的。每一台网络设备都保存着一个列表，这个列表中保存着它自己运行着的点对点程序的 PNRP ID，以及某些其他计算机上运行的点对点程序的 PNRP ID 以及对应的 IP 地址。当一个网络设备想连接另一个计算机上的点对点服务时，首先检查它想连接的服务的 PNRP

www.ckook.com

ID 是否存在于它本机保存的列表中，如果存在，它就可以直接连接目标设备。如果不存在，它就向与那个服务的 PNRP ID 最接近的计算机发出解析请求，请求它协助自己完成地址解析。通过这样递归的方式，一台计算机就能解析出它所请求服务所在的目标主机 IP 地址，而不需要 DNS 的帮助。

以图 6-3 作为例子，对等点 A 提供一项服务的 PNRP ID 为 200，同时它保存的列表中有 B 所运行的服务 450 和 C 所运行的服务 500，在图中，我们使用带箭头的线段表示某一个对等点在列表中包含另一个对等点提供的服务的位置信息。当对等点 A 想连接服务 800 时：

（1）首先 A 查找自己的列表，发现 800 不在列表中，而表中的 500 最接近于 800，于是 A 向运行服务 500 的 C 主机发出一个 PNRP 请求，请求 C 协助 A 寻找到运行服务 800 的主机；

（2）C 收到 A 的请求消息之后检查自己的列表，发现自己的列表中没有任何项，于是 C 只能给 A 回复一条消息，告诉 A 自己找不到运行服务 800 的主机，也找不到任何运行服务代号接近 800 的主机；

（3）A 只能再向服务代号第二接近 800 的运行服务 450 的主机 B 发出相同的请求；

（4）B 查找自己的列表，发现 E 运行着服务 800，于是它将 E 的 IP 地址发送给 A；

（5）然后 A 直接发送一个 PNRP 请求给 E；

（6）E 回复一个确认消息给 A。

图 6-4 描绘了 A 查找服务 800 所在计算机 IP 地址的流程。

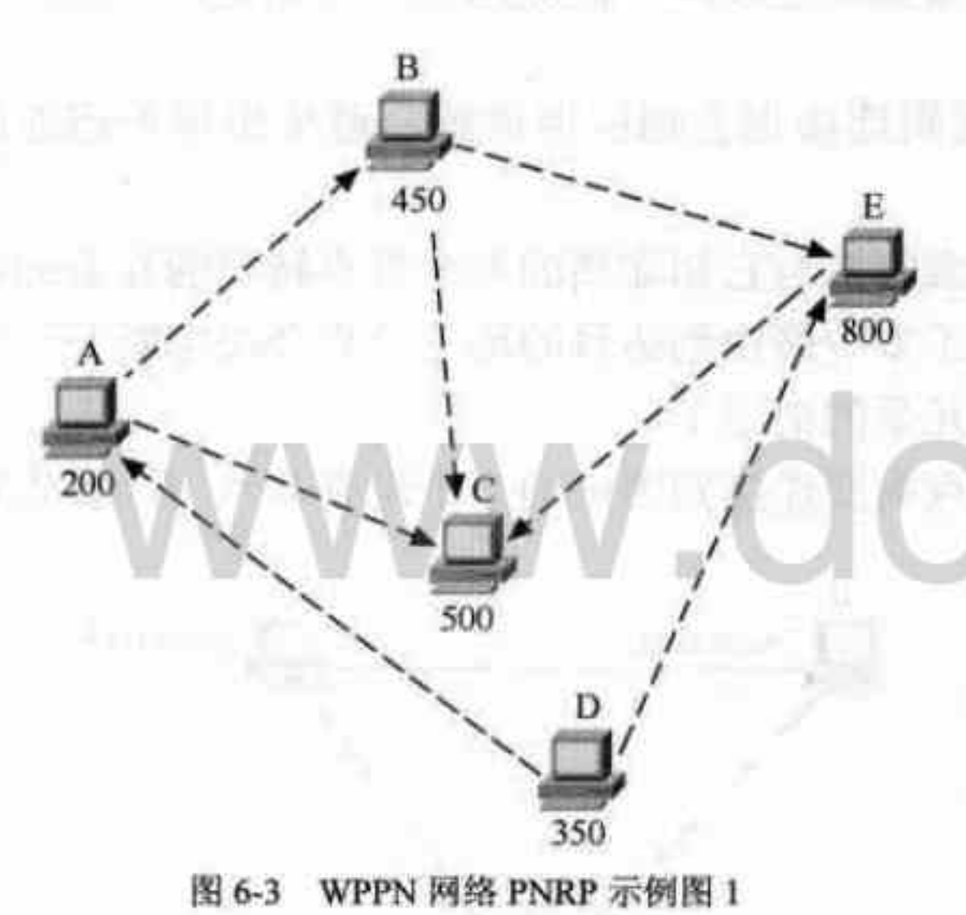


图 6-3　WPPN 网络 PNRP 示例图 1

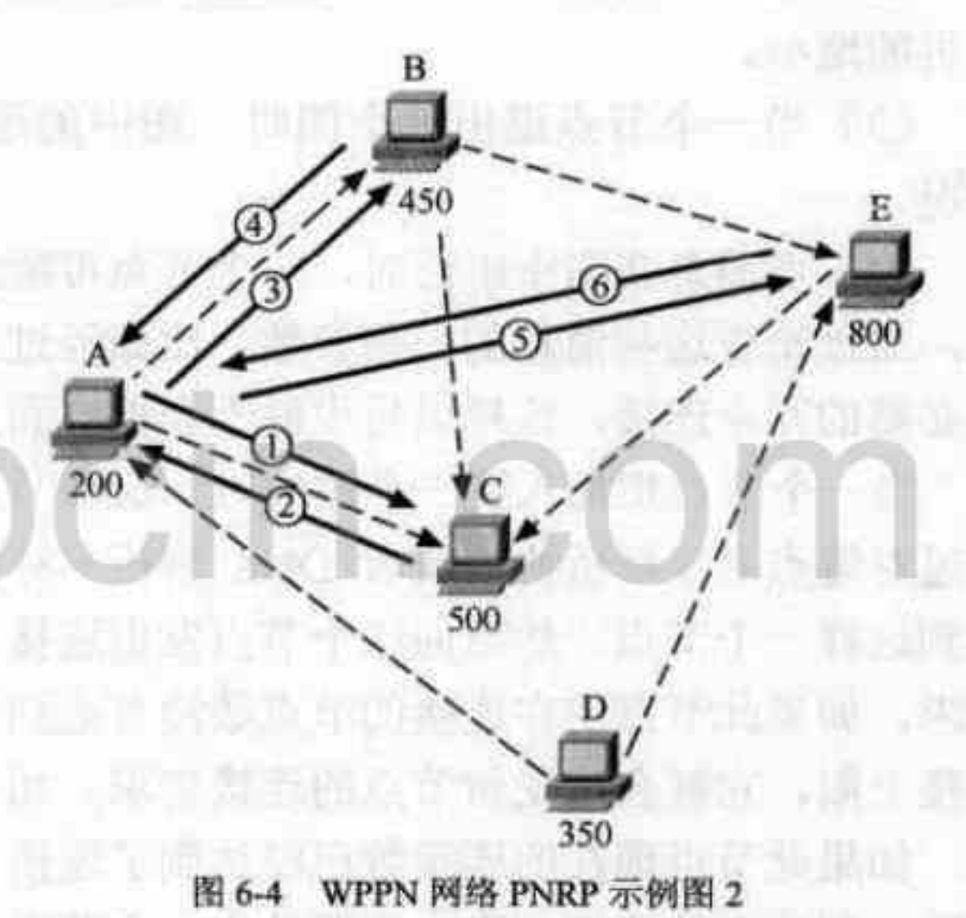


图 6-4　WPPN 网络 PNRP 示例图 2

www.docin.com

2．图模块

一个图（Graph，又被称作对等点图，Peer Graph）是一个相互连接的节点的集合，图中的各个节点可以通过逻辑上的邻居来转发请求或者消息，最后实现任意两点之间的连接。WPPN 开发平台中对等点图的建立是基于洪泛算法，洪泛是一种把消息传播给所有用户的途径，它的效率并不高但是实现简单。在 WPPN 开发平台中，洪泛算法被用于以下用途。

（1）向所有的节点发送增加新记录的消息；

（2）向所有节点发送更新记录的消息；

（3）向所有节点发送删除记录的消息。

为了实现以上功能，每条洪泛出来的消息都通过全局唯一标识符（GUID）来标识。一个

www.ckook.com

建立好的图有以下特点。

（1）连通性好，在图中的任意两个节点都有一条通路连接。

（2）半径小，图中的两个逻辑上相连的节点（具有直接的 TCP 连接的两个节点）在物理上也比较接近。半径小的图有利于消息在图中快速地传播。

（3）顽健性强，当图中某些节点消失时，此图仍然保持连通。

在一个图中，逻辑上相连的两个节点叫作邻居，这些逻辑上相连的节点之间可能相隔多个路由器。节点 ID 是一个节点在加入到一个图时随机产生的一个数据，在一个图中每一个节点的 ID 必须是唯一的。而一个图是通过图签名来唯一标识的，这个图签名就是此图中 ID 号最小的那个节点的 ID。例如有一个图中假设存在 3 个节点，它们的 ID 分别是 5217、6749、3490，则这个图签名为 3490，这个号就标识了这一个图，当然实际的 ID 号远远比这个要大。

下面我们介绍一下图是如何维护的。图中消息的传输是通过洪泛算法，而图的建立与维护则是通过图维护协议来完成的。图维护协议定义了一个图是如何保持健壮的连接和较小的半径的。

（1）图通过一个图签名来标识，如果一个图被分割成几个小块，每一个小块形成新的图，而这些新的图签名就会不同，通过比较各个图签名就可以很容易地发现图被分割了。在一个图中，图签名通过联系人（Contact）来维护，在节点加入到一个图时它可能被随机地选做联系人，一个图可能存在着多个联系人。

（2）当一个新的节点加入到图时，图中的逻辑连接可能会相应作出调整，使得图的半径尽可能地小。

（3）当一个节点退出一个图时，图中的逻辑连接也会相应地调整，避免出现不连通的情况。

（4）当消息在图中洪泛时，一个节点可能会收到与它相连接的多个节点转发的冗余的消息，通过检查这些消息的一些参数，比如经过了多少路由到达目的地，节点会决定断开一些不必要的冗余连接，这样以后也就不会再收到冗余的消息了。

当一个节点想加入到一个图中时，这个节点需要连接到图中的一个现有节点上，首先它通过对等点名字解析协议或者 DNS 解析名称找到这样一个节点。然后向这个节点发出连接请求，如果此节点现在连接的节点数没有达到连接上限，它就会接受新节点的连接请求。相反，如果此节点现在的连接数已经达到了连接上限，则从它的连接列表中随即抽出一个节点的地址发送给这个想加入的节点。

当一个节点从图中退出时，它会发出一个退出消息。因为节点的退出可能会导致图的分割，所以节点所发出的退出消息中会包含于它自己连接的所有节点的列表。收到退出消息的节点会尝试与列表中的其他节点取得逻辑连接。

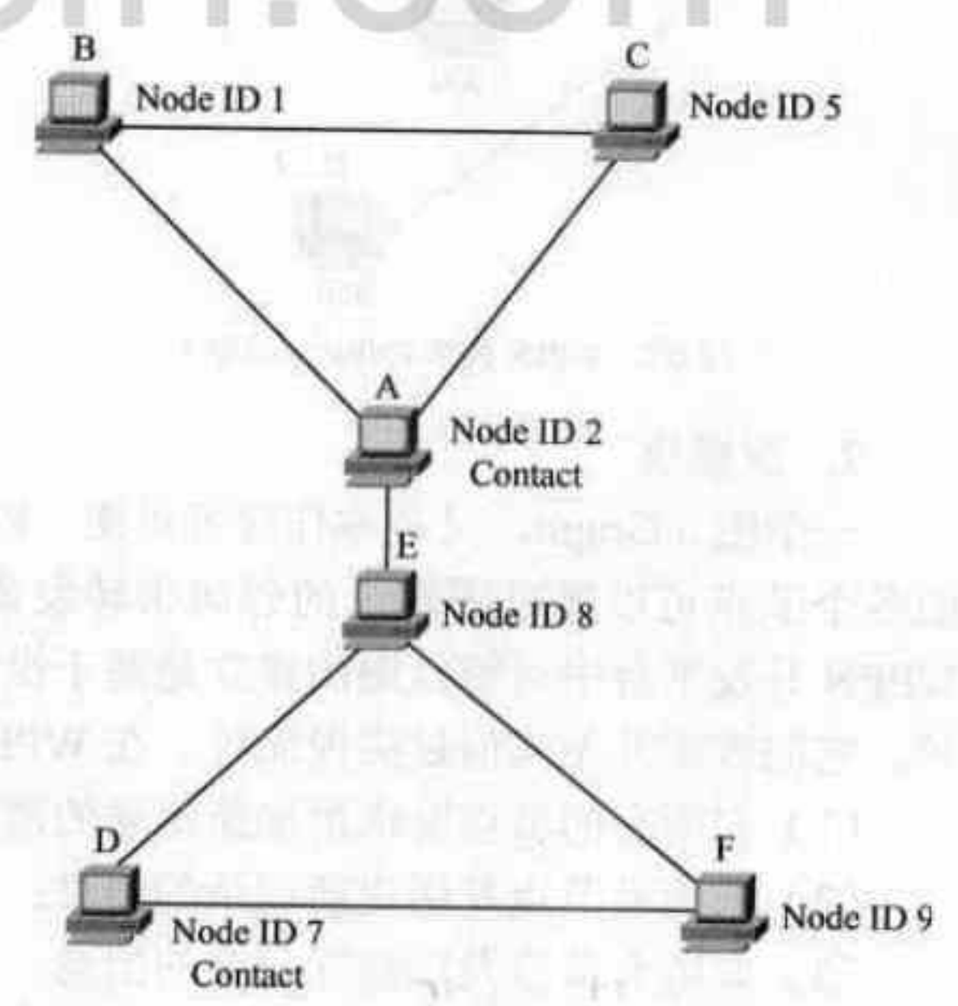


图 6-5 “图”创建示意图

下面详细描述一下一个图创建的过程。

我们把图 6-5 所示的这个图称为图 X，图

www.ckook.com

X 由节点 A 创建，所以开始时图 X 中只有一个节点 A，它的 ID 号为 2，节点 A 首先会创建一个签名记录（Signature Record）和一个联系人记录（Contact record），签名记录为它自己的 ID 号 2，联系人记录的内容为节点 A 的 ID 号及相关信息。然后节点 A 使用这个签名记录通过签名算法对联系人记录进行签名。然后节点 B 加入到图 X 中，节点 B 的 ID 号为 1，因为此时图 X 中只有 A、B 两个节点，所以 A 与 B 之间建立起一条连接，同时 B 获取 A 创建的签名记录和联系人记录。B 比较自己的 ID 和签名记录，发现自己的 ID 比签名记录更小，便要对图重新生成签名记录。节点 B 把新的签名记录也就是自己的 ID 号 1 广播到图中，当然这个时候图 X 中其实只有 A、B 两个节点，节点 A 收到 B 传来的新的签名记录时会更新自己数据库中的签名记录。对比到新收到的签名记录与之前的签名记录不一致时，节点 A 会使用新的签名记录也就是节点 B 的 ID 号对联系人记录重新签名。完成新的签名之后，节点 A 再把新的联系人记录发送给 B，这样节点 B 完成了加入图 X 的全部过程。此时图 X 中的两个节点 A、B 都有了相同的数据，它们都有通过 ID 1 签名的联系人记录，联系人记录中有着节点 A 的 ID 号和其他信息。注意这种方式中，B 加入到图中，但是 B 并不是图的联系人。通过相同的方式其他节点也可以加入到图 X 中。

3．WPPN 与安全

在点对点网络中不存在中心服务器，也就不存在关于安全的中心数据库，而在普通的客户端/服务器网络中，这些中心服务器、安全数据库提供着典型的安全服务，比如鉴权、认证、授权。而在无服务器的点对点网络中，对等点自己必须提供一种鉴权机制。在 WPPN 中，鉴权是通过自签名的证书来完成的，证书可以满足 X.509 证书格式。X.509 的认证方式类似于一个分层的 PKI（公钥基础设施）系统，其中有许多 CA（认证实体）。对等点网络用于每一个节点像一个 CA 一样工作，每一个节点自己生成一个公私钥对，并且通过私钥签名证书。这些通过签名的证书可以用于对等点相互之间的鉴权认证。

4．IPv6 与 NAT

WPPN 使用 IPv6 作为其网络层，在 IPv6 下不存在地址短缺问题。而在 IPv4 下，通常需要通过 NAT（网络地址转换）来解决地址短缺的问题，NAT 延长了 IPv4 的生命期，但是却破坏了端到端的通信。通常有一个关于 IPv6 的误解，许多人以为现在的 IPv4 系统必须通过升级才能使用 IPv6，其实这是不正确的，IPv6 在设计之初就考虑到了这些问题，所以 IPv6 中有一系列的技术可以保证 IPv4 网络与 IPv6 网络的顺畅交互。

在 Windows 平台下，有两种 IPv4 到 IPv6 的过渡技术。

（1）内网自动隧道协议（ISATAP）

ISATAP 是在 IPv4 的内网中在 IPv6 的用户之间进行单播地址转换的技术，详细资料可以查看 RFC 4214。

（2）IPv6 到 IPv4

IPv6 到 IPv4 的地址转换，可以与 ISATAP 交换使用，详细资料可以查看 RFC 3056。

对于 IPv4 因特网中的 IPv6 连接，“IPv6 到 IPv4”是一项被微软所提倡的技术。但是“IPv6 到 IPv4”技术需要一个对一个私有网络中的计算机分配一个公共的 IPv6 地址，然后利用这台计算机做 IPv6 到 IPv4 转换的路由器。Windows XP 和 Windows Server 2003 可以通过因特网连接共享（ICS）来自动实现这样一个作为 IPv6 到 IPv4 转换的路由器的功能。但是许多用于连接小型办公网络到因特网的 NAT 并不兼容这样的 IPv6 到 IPv4 的转换功能，

www.ckook.com

况且许多小型办公网络连接到因特网并不仅仅通过一个 NAT。NAT 另外存在的一个问题是它们不支持 TCP 和 UDP 以外的其他协议的中继功能。

6.1.4 Windows Peer-to-Peer Networking 平台开发环境

1．安装运行支持

安装 Windows Peer-to-Peer 运行支持是极其方便的，对于 Window XP SP1 的用户只需要在微软官方网站上下载免费的 Advanced Networking Pack for Windows XP。而对于 Windows XP SP2、Windows XP Professional x64 Edition、Windows Vista 的用户只需要按照图 6-6 所示的步骤就能安装 Windows Peer-to-Peer 开发平台。

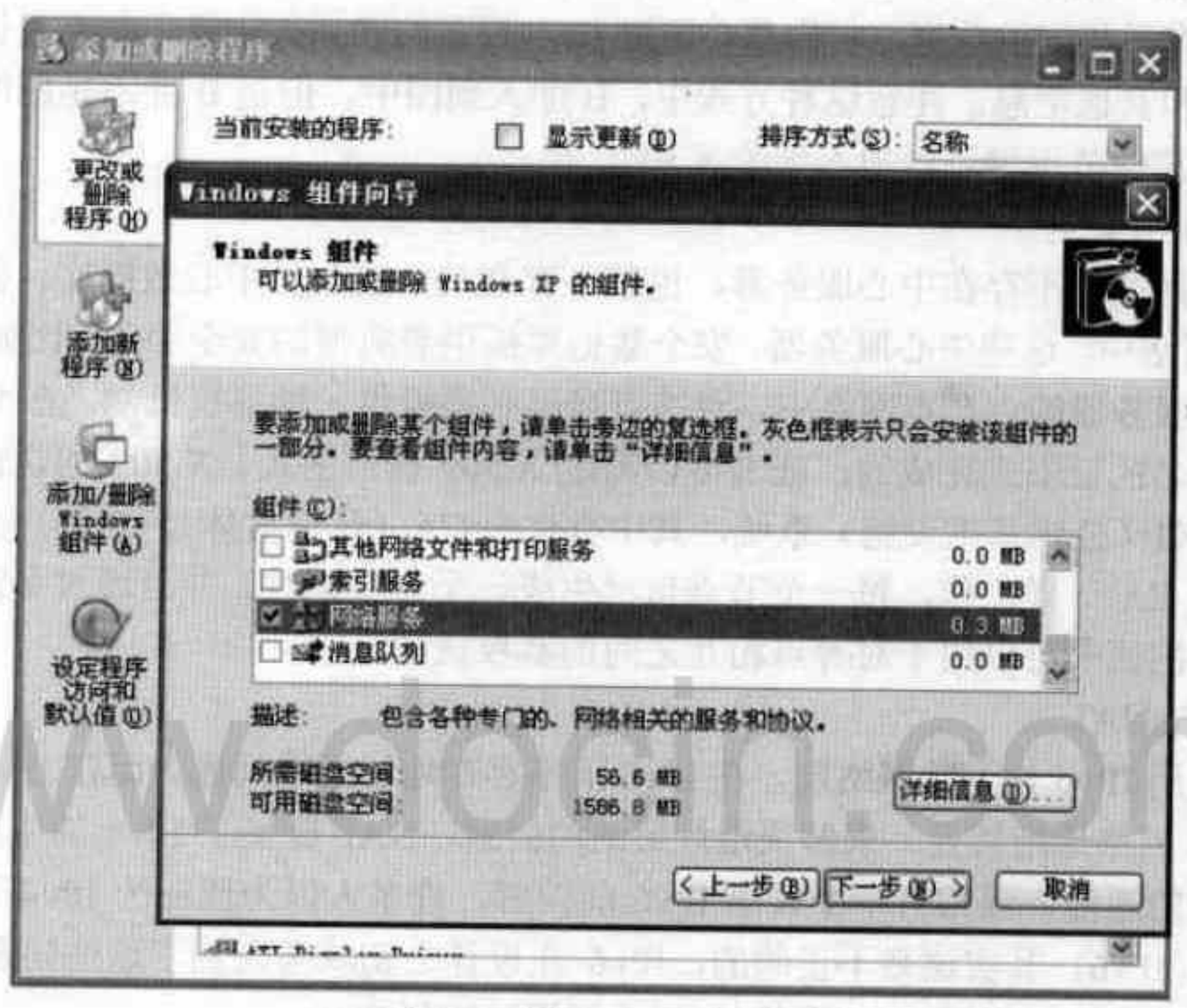


图 6-6 安装 Windows 组件

单击"开始"中的"控制面板"，进入"添加/删除程序"中的"添加/删除 Windows 组件"，双击"网络服务"一项，进入图 6-7 所示的菜单。

选中"点对点"复选框，单击"确定"按钮，系统会自动完成安装。在安装运行支持之后，就可以运行利用视窗点对点平台开发的应用程序了。

2．安装 Visual Studio

Microsoft Visual Studio 是一个完整的开发工具套装软件，它给企业提供创建解决方案的易用工具。Visual Studio 专业版使开发者能完全利用 Microsoft Windows 操作系统和 Web 开发的功能。现在 VS 可用的版本包括 Visual Studio 6.0、Visual Studio .Net 2003、Visual Studio .Net 2005 等。读者可以根据自己的情况选择。图 6-8 所示是 Visual Studio .Net 2003 典型的运行界面。

www.ckook.com

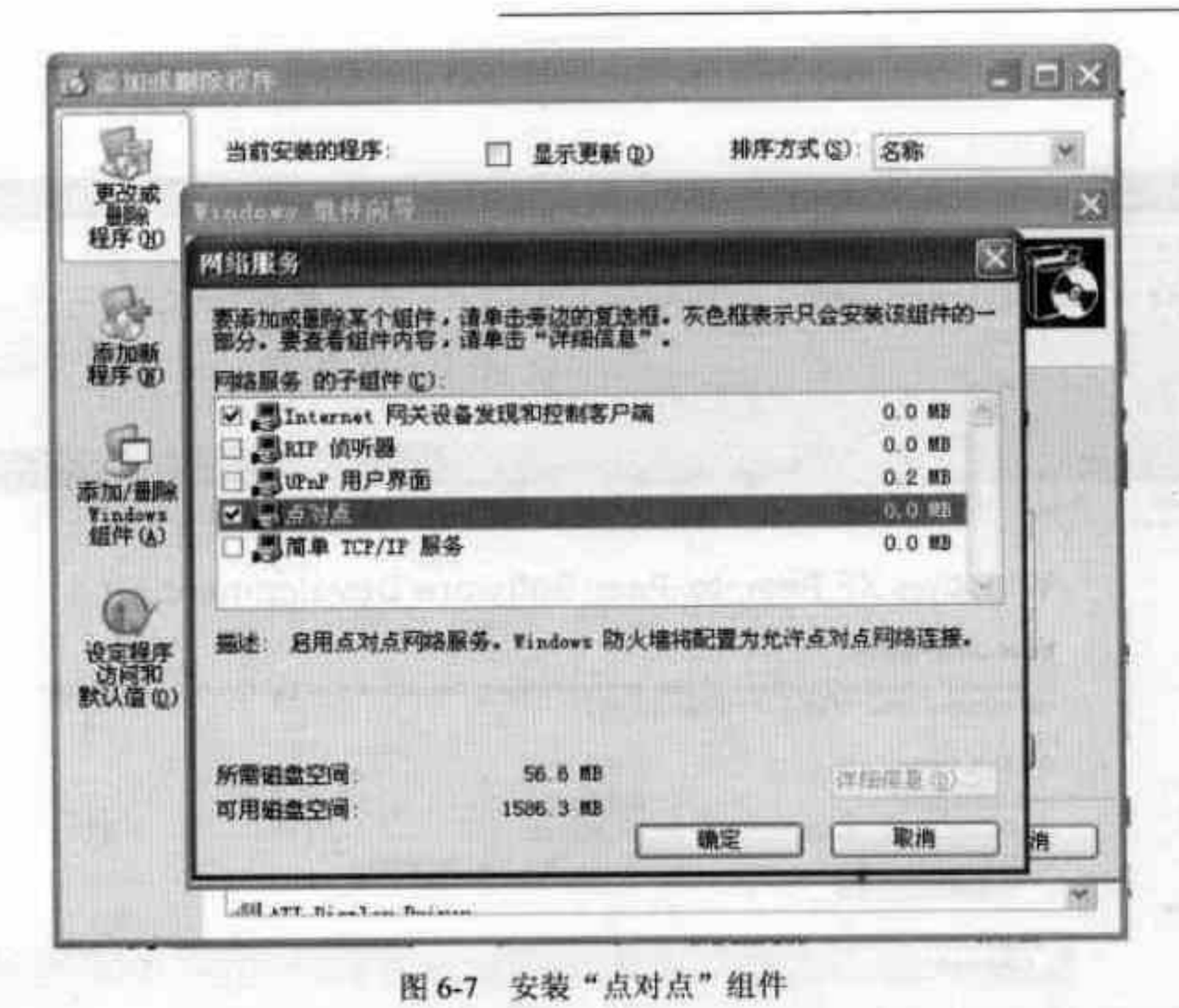


图 6-7 安装"点对点"组件

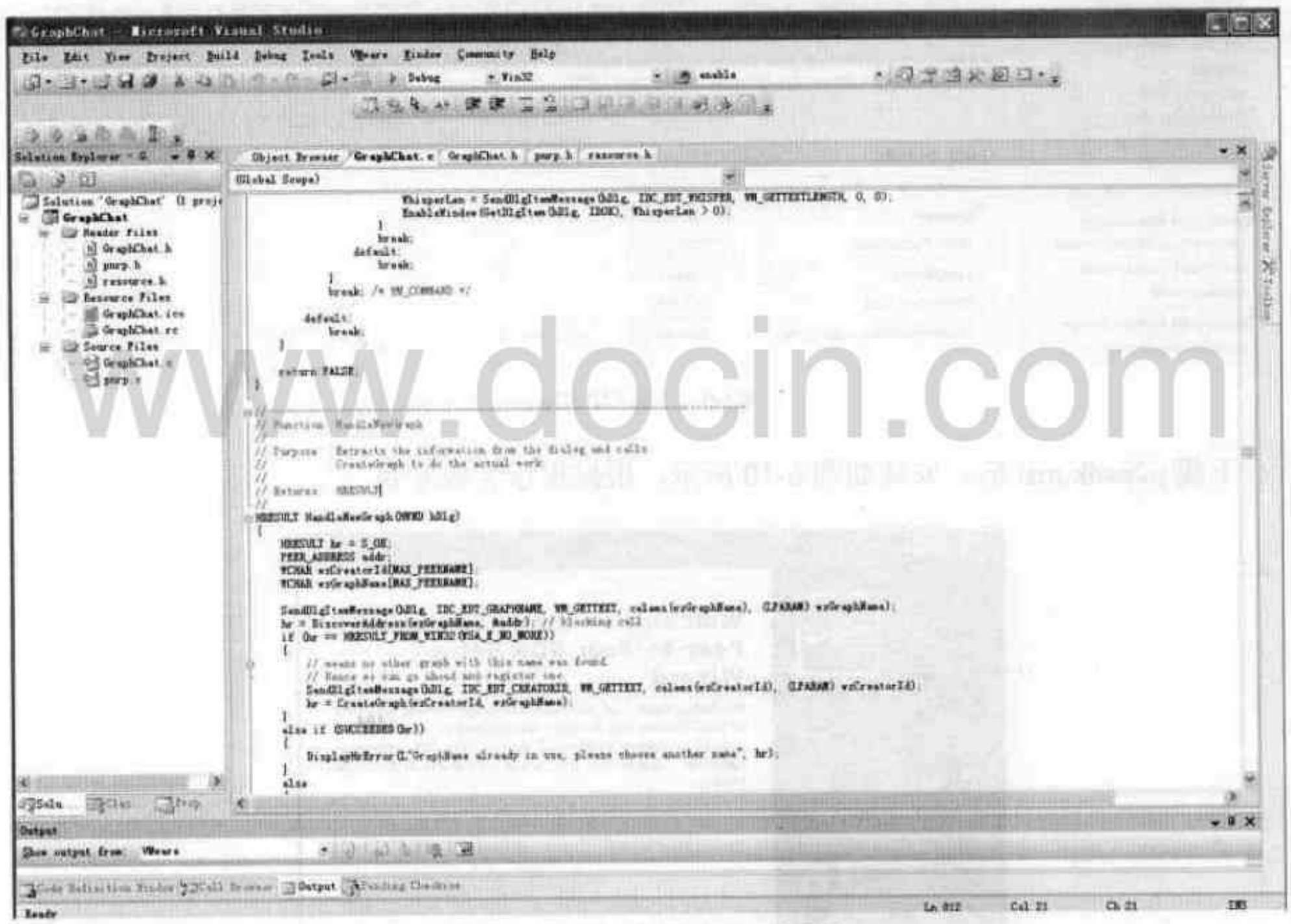


图 6-8 Visual Studio.Net 2003 运行界面

3．安装点对点开发包

除此之外，你仅需要从微软官方网站下载 P2P SDK，网址为 http://www.microsoft.com/downloads/details.aspx?familyid=5116a614-a487-4dff-b384-829cd8ce977d&displaylang=en，如图

www.docin.com

6-9 所示。

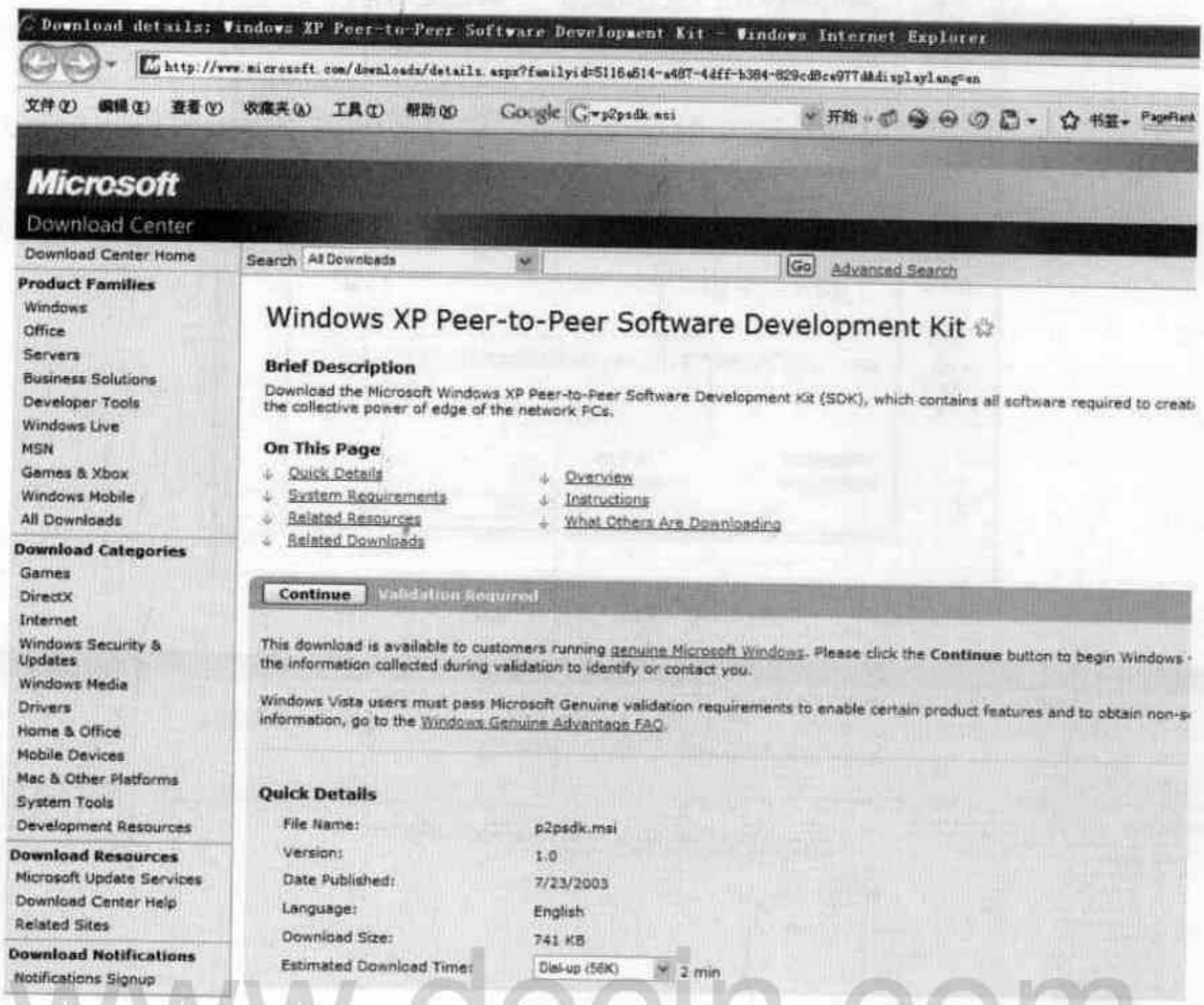


图 6-9 下载 P2P SDK

www.docin.com

下载 p2psdk.msi 后，安装如图 6-10 所示，根据提示完成安装。

图 6-10 安装 P2P SDK

www.ckook.com

6.1.5 Windows Peer-to-Peer Networking 平台开发入门

1. Windows Peer-to-Peer Networking 平台开发接口

WPPN 平台包含了一系列的 API 功能强大、使用灵活的函数，这些函数大致可以分为以下 4 类。

（1）点对点图模块 API，此类 API 可以使对等点高效、可靠地在图之间传递信息。使用这些 API 可以方便地创建管理对等点图，以及向图中节点发送数据。

① PeerGraphStartup

```
HRESULT WINAPI PeerGraphStartup(
WORD wVersionRequested,                    // PEER_GRAPH_VERSION 的版本号
PPEER_VERSION_DATA pVersionData            // PEER_VERSION_DATA 结构地址，接受本机安装的
P2P 平台版本
);
```

此函数是每一个使用 Windows Peer-to-Peer Networking 平台的程序用来初始化平台图模块的函数，也是使用图模块的程序必须调用的第一个图模块的函数。与此函数相对应的还有 PeerGraphShutdown 函数，用于释放图模块所使用的资源。

② PeerGraphCreate

```
HRESULT WINAPI PeerGraphCreate(
  PPEER_GRAPH_PROPERTIES pGraphProperties,          //创建图的一些参数
  PCWSTR pwzDatabaseName,                           //用于保存记录数据库文件
  PPEER_SECURITY_INTERFACE pSecurityInterface,      //安全属性
  PHGRAPH phGraph                                   //句柄指针，用于返回图的句柄
);
```

此函数用于创建一个对等点图，通过此函数一个应用程序可以详细地设置这个图的信息、安全级别。此函数如果执行成功，将通过最后一个参数返回一个指向这个图的句柄，但是真正的网络连接并没有建立。

③ PeerGraphRegisterEvent

```
HRESULT WINAPI PeerGraphRegisterEvent(
  HGRAPH hGraph,                                    //图的句柄
  HANDLE hEvent,                                    //待触发的事件句柄
  ULONG cEventRegistrations,
  PPEER_GRAPH_EVENT_REGISTRATION pEventRegistrations,
  HPEEREVENT* phPeerEvent
);
```

函数 PeerGraphRegisterEvent 将节点注册到一个对等点图中，当对等点图中触发事件的时候，对等点图会通过参数 hEvent 触发本地事件。对于对等点的创建者在成功执行 PeerGraphCreate 后调用此函数，对于其他图成员在调用 PeerGraphOpen 之后调用此函数。

④ PeerGraphOpen

```
HRESULT WINAPI PeerGraphOpen(
  PCWSTR pwzGraphId,                                //图的 ID 号
```

www.ckook.com

```
    PCWSTR pwzPeerId,                                    //调用此函数的节点 ID 号
    PCWSTR pwzDatabaseName,                              //用于保存记录数据库文件
    PPEER_SECURITY_INTERFACE pSecurityInterface,         //安全属性
    ULONG cRecordTypeSyncPrecedence,                     //记录类型的数量
    GUID* pRecordTypeSyncPrecedence,
    PHGRAPH phGraph                                      //句柄指针，用于返回图的句柄
);
```

此函数用于打开先前 PeerGraphCreate 建立的一个图，这个图可以是本地的也可以是远程的。

⑤ PeerGraphListen

```
HRESULT WINAPI PeerGraphListen(
    HGRAPH hGraph,                                       //图的句柄
    DWORD dwScope,
    DWORD dwScopeId,
    WORD wPort                                           //端口号，设置为 0 时系统动态分配端口号
);
```

调用此函数后节点开始监听对等点图中连接请求。

⑥ PeerGraphConnect

```
HRESULT WINAPI PeerGraphConnect(
    HGRAPH hGraph,                                       //图的句柄
    PCWSTR pwzPeerId,                                    //对等点 ID
    PPEER_ADDRESS pAddress,                              //指向对等点的地址的结构体
    ULONGLONG* pullConnectionId                          //用于标识连接的 ID
);
```

使用函数 PeerGraphConnect 建立与图中某一个对等点的连接。此函数的调用是异步的，连接是否成功需要等待 PEER_GRAPH_EVENT_NEIGHBOR_CONNECTION 事件。

⑦ PeerGraphClose

```
HRESULT WINAPI PeerGraphClose(
    HGRAPH hGraph                                        //图的句柄
);
```

调用此函数，节点断开所有和图中节点的连接，从图中退出。

（2）点对点组模块 API，此类 API 是 PNRP 协议与图模块的结合。使用此类 API 可以创建和管理安全的对等点组，并且与组中的成员通信。

① PeerGroupStartup

```
HRESULT WINAPI PeerGroupStartup(
    WORD wVersionRequested,
    PPEER_VERSION_DATA pVersionData
);
```

与图模块类似，要使用组模块，应用程序必须先初始化平台，调用函数 PeerGroupStartup。

② PeerGroupCreate

```
HRESULT WINAPI PeerGroupCreate(
    PPEER_GROUP_PROPERTIES pProperties,
    HGROUP* phGroup
);
```


www.docin.com
www.ckook.com

与 PeerGraphCreate 类似，PeerGroupCreate 创建一个组。

③ PeerGroupJoin

```
HRESULT WINAPI PeerGroupJoin(
    PCWSTR pwzIdentity,     //表示身份的字符串
    PCWSTR pwzInvitation,   //邀请书
    PCWSTR pwzCloud,
    HGROUP* phGroup
);
```

调用此函数的节点将发出一个加入已存在的组的请求。

④ PeerGroupConnect

```
HRESULT WINAPI PeerGroupConnect(
    HGROUP hGroup
);
```

此函数会使用 PNRP 协议查找一个对等点组，并尝试着连接这个对等点组，此函数若返回成功，节点就可以和对等点组中的节点进行通信。

注意：此函数不同于对等点图中的 PeerGraphConnect 函数，前者连接对象是一个组，后者连接对象是图中一个节点。对等点组还提供其他很多 API 函数，这里就不一一介绍了。

（3）点对点名字解析 API，此类 API 提供了无服务器的名字解析服务，是 PNRP 协议的实现，使用此类 API 可以管理、注册、注销以及解析对等点名称。其实 WPPN 是使用 Winsock 2 的名字空间提供商 API 来实现 PNRP 的功能的。

① WSASetService

```
INT WSASetService(
    LPWSAQUERYSET lpqsRegInfo,
    WSAESETSERVICEOP essOperation,
    DWORD dwControlFlags
);
```

PNRP 使用此函数进行注册和注销对等点名称。

② WSALookupServiceBegin

```
INT WSALookupServiceBegin(
    LPWSAQUERYSET lpqsRestrictions,
    DWORD dwControlFlags,
    LPHANDLE lphLookup
);
```

要解析一个对等点名称，首先需要调用函数 WSALookupServiceBegin，此函数会返回一个查询句柄供以后使用。

③ WSALookupServiceNext

```
INT WSALookupServiceNext(
    HANDLE hLookup,
    DWORD dwControlFlags,
    LPDWORD lpdwBufferLength,
    LPWSAQUERYSET lpqsResults
);
```

www.ckook.com

调用 WSALookupServiceBegin 之后，程序再使用其返回的句柄，并调用 WSALookup ServiceNext 函数来解析对等点名称。

④ WSALookupServiceEnd

```
INT WSALookupServiceEnd(
  HANDLE hLookup
);
```

最后查询结束后，程序调用 WSALookupServiceEnd 释放资源，完成名字解析。

（4）点对点身份管理 API，此类 API 可以用于建立 PNRP 可使用的安全的对等点名称。

① PeerIdentityCreate

```
HRESULT WINAPI PeerIdentityCreate(
  PCWSTR pwzClassifier,                    //合法的对等点标识符
  PCWSTR pwzFriendlyName,                  //昵称
  HCRYPTPROV hCryptProv,                   /加密使用的密钥
  PWSTR* ppwzIdentity                      //创建的新对等点名称的指针
);
```

此函数将创建一个新的对等点名称，此函数返回的结果是其他身份管理函数的基础。

② PeerIdentityDelete

```
HRESULT WINAPI PeerIdentityDelete(
  PCWSTR pwzIdentity                       //准备删除的对等点名称
);
```

此函数永久地删除一个对等点名称，包括移除这个对等点相关的认证信息、私钥信息等。

③ PeerIdentityImport

```
HRESULT WINAPI PeerIdentityImport(
  PCWSTR* ppwzImportXML,                   //导入对等点信息使用的 XML 文件
  PCWSTR pwzPassword,                      //导出信息时使用的加密密码
  PCWSTR* ppwzIdentity                     //导入的对等点标识
);
```

此函数导入一个已创建好的对等点名称。比如用户在主机 A 上创建了对等点，在主机 B 上需要使用此函数来导入先前创建的对等点。

④ PeerIdentityExport

```
HRESULT WINAPI PeerIdentityExport(
  PCWSTR pwzIdentity,                      //需要导出的对等点标识
  PCWSTR pwzPassword,                      //导出信息要使用的加密密码
  PCWSTR* ppwzExportXML                    //导出对等点信息使用的 XML 文件
);
```

此函数导出一个已创建好的对等点名称。

2．Windows Peer-to-Peer Networking 平台开发实例

本节我们会解析一个使用 WPPN 图模块开发的点对点聊天程序。**该程序的源代码可以在 6.1.4 节中讲到的 SDK 中找到**。此程序运行的界面如图 6-11 所示。在界面的右侧有所有在线用户的列表，界面的左侧有两个框。上面的框显示系统信息以及所有的聊天记录，下面的框是用户用作输入的对话框。当用户完成输入之后只需要单击“Send”按钮就可以发送消息。

www.ckook.com

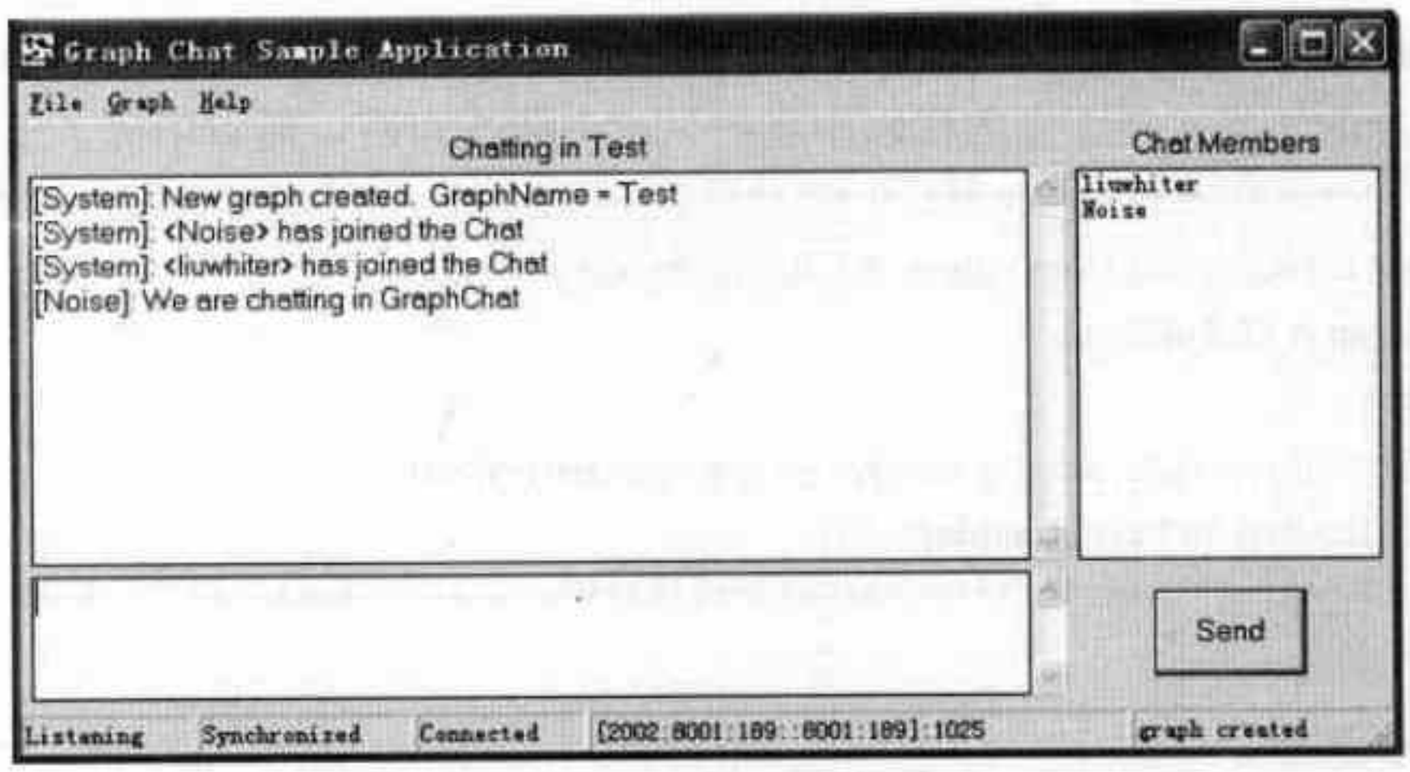
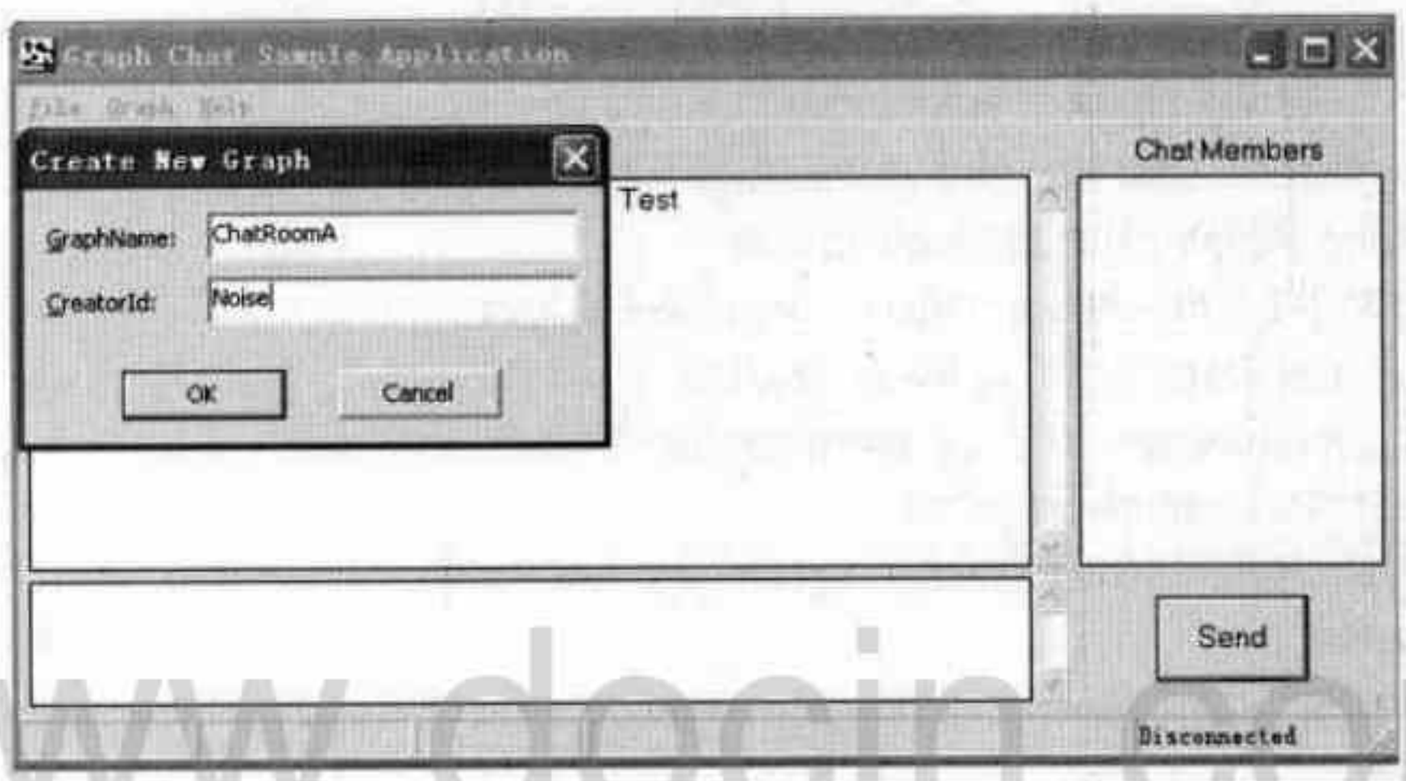


图 6-11　Graph Chat 运行界面 1

如果用户想要创建一个新的 Graph，只需要选择“Graph”→“Create New Graph”命令，程序就会创建一个新的 Graph 用于聊天，如图 6-12 所示。



www.docin.com

图 6-12　Graph Chat 运行界面 2

同样如果一个用户知道一个已存在的 Graph 的名称，他可以直接加入这个 Graph，如图 6-13 所示。

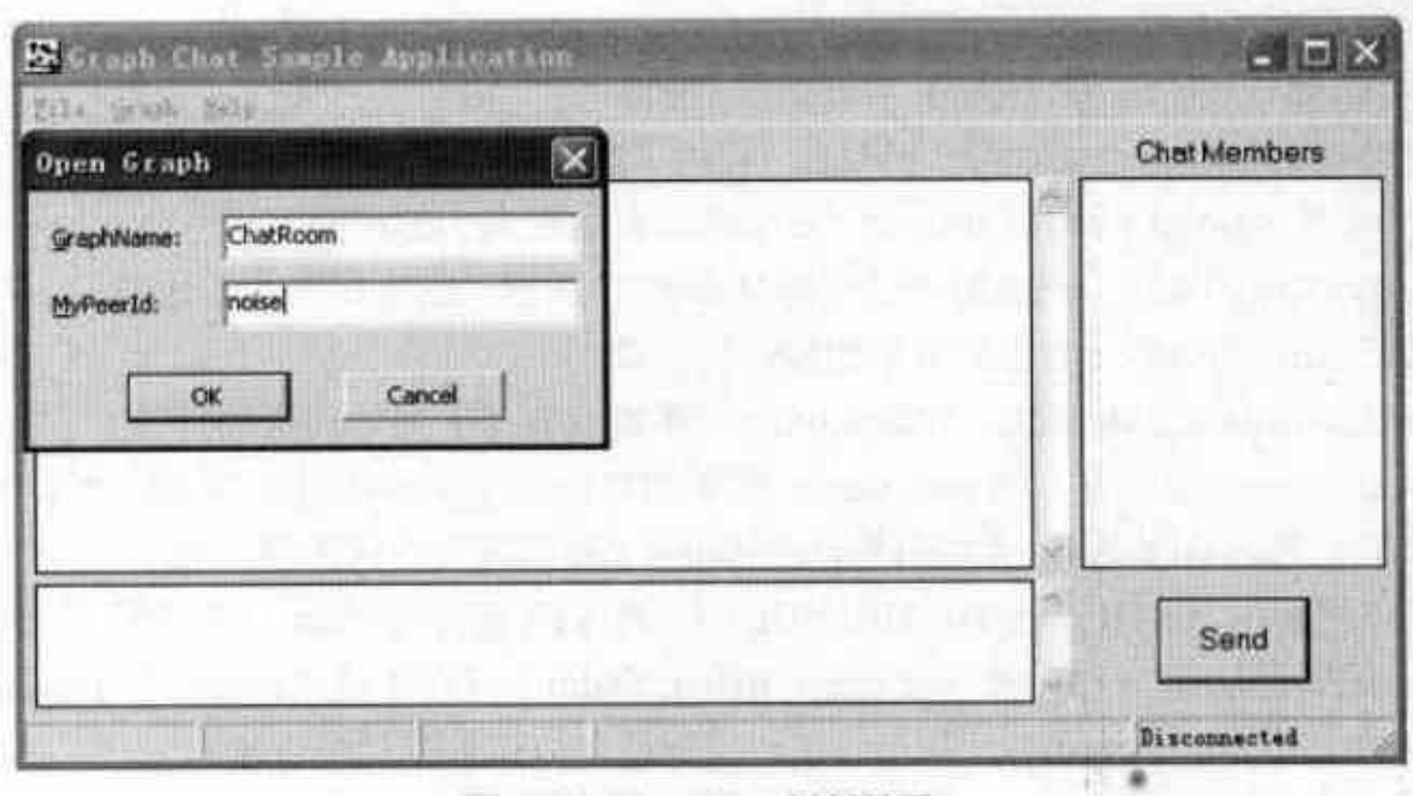


图 6-13　Graph Chat 运行界面 3

www.ckook.com

GraphChat 的代码并不庞大，但是由于篇幅的限制，我们主要分析它的 P2P 模块的实现。对于其界面部分的实现，在下面的代码中我们只给出函数声明，而省略掉了函数定义。

```
/*****************************************************************************
Copyright (c) Microsoft Corporation. All Rights Reserved.
Module Name: GraphChat.c
Abstract:
    This C file includes sample code for ga chat application built
    with the Peer-to-Peer Graphing API.
*****************************************************************************/
#include "GraphChat.h"
#include "pnrp.h"

HINSTANCE g_hInst = NULL; //应用程序句柄
HWND g_hwndMain, g_hwndText, g_hwndMsg, g_hwndMembers, g_hwndChatLabel;
HWND g_hwndSend, g_hwndMemberLabel, g_hwndStatus; //窗口句柄
WCHAR g_wzGraphId[MAX_PEERNAME], g_wzPeerId[MAX_PEERNAME]; //peer 的 Id
WCHAR g_wzWsprId[MAX_PEERNAME];
BOOL g_fSynchronized = FALSE, g_fPnrpRegistered = FALSE;
HGRAPH g_hGraph = NULL; //graph 的句柄
HPEEREVENT g_hPeerEvent = NULL; //PeerEvent 的句柄
HANDLE g_hEvent = NULL, g_hWait = NULL;
PWSTR g_pwzIdentity = NULL, g_pwzWsprMsg = NULL;
ULONGLONG g_ullConnection = 0;
GUID CHAT_MESSAGE_RECORD_TYPE = {0x4d5b2f11, 0x6522, 0x433b, {0x84, 0xef, 0xa2, 0x98,
0xe6, 0x7, 0x57, 0xb0}};
GUID WHISPER_MESSAGE_TYPE =  {0x4d5b2f11, 0x6522, 0x433b, {0x84, 0xef, 0xa2, 0x98, 0xe6,
0x7, 0xbb, 0xbb}};

//////////////////////////////////////////////////
// 以下是 UI 和用户输入相关的函数
//////////////////////////////////////////////////
HRESULT InitUI(void); //此函数用于初始化主窗口
BOOL ProcessSpecialKeys(MSG * pMsg); //设置 Tab 键和 Enter 键处理的函数
void ProcessResizing(LPRECT pRect); //处理改变对话框大小的函数
void ResizeMainWindow(int dxMain, int dyMain);//设置窗口的指定大小的函数
void AddControls(void); //为主窗口添加控件的函数
void EnableDisconnectMenu(); //使"Disconnect"按键有效或者无效的函数
void DisplayMessage(PCWSTR pwzName, PCWSTR pwzMsg); //在 chat 显示区内显示消息的函数
void SetStatusPart(int sbPart, PCWSTR pwzStatus); //此函数设置状态栏的文字
void MsgErrHr(PCWSTR pwzMsg, HRESULT hr, PCSTR pszFunction); //此函数显示错误消息
void UpdateParticipant(WCHAR* wzPerson, BOOL fAdd); //从在线用户列表控件中添加删除在线用户
void ClearParticipantList(void); //从在线用户列表控件中添加删除所有用户
#define DisplaySysMsg(wzMsg)   DisplayMessage(L"System", wzMsg)
```


www.ckook.com

```
#define SetStatus(wzMsg) SetStatusPart(SB_PART_MESSAGE, wzMsg)
#define DisplayHrError(pwzMsg, hr)     MsgErrHr(pwzMsg, hr, __FUNCTION__)

// 主窗口的消息处理函数
LRESULT CALLBACK MainProc(HWND hWnd,UINT message,WPARAM wm,LPARAM lm);
// About 窗口的消息处理函数
LRESULT CALLBACK AboutProc(HWND hDlg, UINT message, WPARAM wm, LPARAM lm);
// ‘创建 graph 对话框’的消息处理函数
LRESULT CALLBACK NewGraphProc(HWND hDlg, UINT message, WPARAM wm, LPARAM lm);
// ‘打开存在 graph’的对话框消息
LRESULT CALLBACK OpenGraphProc(HWND hDlg, UINT message, WPARAM wm, LPARAM lm);

// 此函数获取消息文本并发送
void ProcessSendButton(void)
{
    if (FAILED(AddChatRecord(wzMessage)))
        return;
}

// 此函数是应用程序的入口函数
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow)
    {
        BOOL bAcclKeySet = TRUE;
        g_hInst = hInstance;
        SystemParametersInfo(SPI_SETKEYBOARDCUES, 0, &bAcclKeySet, 0);
        if (SUCCEEDED(InitSystem(nCmdShow)))
        {
            MSG msg;
            HACCEL hAccel = LoadAccelerators(g_hInst, (LPCTSTR)IDC_ACCEL);
            while (GetMessage(&msg, NULL, 0, 0))//主消息循环
            {
                HWND hwndTranslate;
                if ((msg.hwnd == g_hwndMain) || IsChild (g_hwndMain, msg.hwnd))
                    hwndTranslate = g_hwndMain;
                else
                    hwndTranslate = msg.hwnd;

                if (!TranslateAccelerator(hwndTranslate, hAccel, &msg))
                    if (!ProcessSpecialKeys(&msg))
                    {   TranslateMessage(&msg);
                        DispatchMessage(&msg);}
            }
```

www.ckook.com

```
        }
        else
            MessageBox(NULL, L"Initialized error", L"Error", MB_OK | MB_ICONWARNING);

        CleanupGraph();//程序退出前析构各种资源
        DeleteIdentity();
        WSACleanup();
        PeerGraphShutdown();
        return 0;
    }

    ////////////////////////////////////////////////////////
    // 以下是 P2P 模块相关的函数
    ////////////////////////////////////////////////////////
    //初始化 P2P 模块的函数
    HRESULT InitSystem(int nCmdShow)
    {
        HRESULT hr = S_OK;
        WORD wVersionRequested = MAKEWORD(2,2);
        WSADATA wsaData;
        // 初始化 WinSock,用于使用 PNRP
        DWORD nRetVal = WSAStartup(wVersionRequested,&wsaData);
        if (nRetVal != 0)
            hr = HRESULT_FROM_WIN32(GetLastError());;

        // 初始化 P2P
        if (SUCCEEDED(hr))
        { PEER_VERSION_DATA peerVersion;
            hr = PeerGraphStartup(PEER_GRAPH_VERSION, &peerVersion);}

        // 创建一个 identity 为 PNRP 使用
        if (SUCCEEDED(hr))
            hr = CreateIdentity();

        if (SUCCEEDED(hr))
            hr = InitUI();

        if (SUCCEEDED(hr))
        { ShowWindow(g_hwndMain, nCmdShow);
            UpdateWindow(g_hwndMain);}
        return hr;
    }

    // 从当前连接的 graph 中断开的函数
```

www.docin.com

www.ckook.com

```
void CmdDisconnect()
{
    CleanupGraph();
    SetStatusPart(SB_PART_ADDRESS, L"");
    ProcessStatusChangeEvent(0);
    SetStatus(L"Disconnected");
}

// 处理秘密消息的对话框消息处理函数
LRESULT CALLBACK WhisperProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    LRESULT WhisperLen;
    HRESULT hr = S_OK;
    switch (message)
    {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
                case IDOK:
                    hr = HandleDirectConnection(hDlg);
                    if (SUCCEEDED(hr))
                        SetStatus(L"Direct msg sent");
                    else if (PEER_E_CONNECT_SELF == hr)
                        DisplayHrError(L"Cannot Whisper to Self", hr);
                    else
                        DisplayHrError(L"Direct Communication Failed", hr);
                    EndDialog(hDlg, IDOK);
                    break;

                case IDC_EDT_WHISPER:
                    if (HIWORD(wParam) == EN_CHANGE)
                    {   WhisperLen = SendDlgItemMessage(hDlg, IDC_EDT_WHISPER, WM_
GETTEXTLENGTH, 0, 0);
                        EnableWindow(GetDlgItem(hDlg, IDOK), WhisperLen > 0);}
                    break;
                default:
                    break;}
            break; /* WM_COMMAND */
        default:
            break;}
    return FALSE;
}
```

www.ckook.com

```
// 此函数从对话框获取信息并调用 CreateGraph 函数
HRESULT HandleNewGraph(HWND hDlg)
{
    HRESULT hr = S_OK;
    PEER_ADDRESS addr;
    WCHAR wzCreatorId[MAX_PEERNAME], wzGraphName[MAX_PEERNAME];
    SendDlgItemMessage(hDlg, IDC_EDT_GRAPHNAME, WM_GETTEXT, celems(wzGraphName),
(LPARAM) wzGraphName);
    hr = DiscoverAddress(wzGraphName, &addr); // blocking call
    if (hr == HRESULT_FROM_WIN32(WSA_E_NO_MORE))
    { SendDlgItemMessage(hDlg, IDC_EDT_CREATORID, WM_GETTEXT, celems(wzCreatorId),
(LPARAM) wzCreatorId);
        hr = CreateGraph(wzCreatorId, wzGraphName);}
    else if (SUCCEEDED(hr))
        DisplayHrError(L"GraphName already in use, please choose another name", hr);
    else
        DisplayHrError(L"Unable to Create Graph", hr);

    return hr;
}

// 此函数从对话框获取信息并调用 OpenGraph 函数
HRESULT HandleOpenGraph(HWND hDlg)
{
    HRESULT hr = S_OK;
    WCHAR wzGraphId[MAX_PEERNAME], wzPeerId[MAX_PEERNAME];
    SendDlgItemMessage(hDlg, IDC_EDT_GRAPHID, WM_GETTEXT, celems(wzGraphId),
(LPARAM) wzGraphId);
    SendDlgItemMessage(hDlg,IDC_EDT_PEERID,WM_GETTEXT,celems(wzPeerId),(LPARAM)
wzPeerId);
    hr = OpenGraph(wzPeerId, wzGraphId);
    if(FAILED(hr))
        DisplayHrError(L"Unable to Open Graph", hr);
    return hr;
}

// 此函数从对话框获取信息并调用 OpenDirectConnection 函数
HRESULT HandleDirectConnection(HWND hDlg)
{
    LRESULT WhisperLen = 0;
    int iItem = ListBox_GetCurSel(g_hwndMembers);
    if (iItem < 0)
        return E_FAIL;
```

www.dcnf.com

www.ckook.com

```
        ListBox_GetText(g_hwndMembers, iItem, g_wzWsprId);
        WhisperLen = SendDlgItemMessage(hDlg, IDC_EDT_WHISPER, WM_GETTEXTLENGTH, 0, 0);
        g_pwzWsprMsg = (WCHAR*)malloc( sizeof(WCHAR) * (WhisperLen+1) );
        if (g_pwzWsprMsg == NULL)
            return E_OUTOFMEMORY;
        SendDlgItemMessage(hDlg,IDC_EDT_WHISPER,WM_GETTEXT,WhisperLen+1,(LPARAM)
g_pwzWsprMsg);
        return OpenDirectConnection();
    }

    // 此函数打开与 peer 的直接连接
    HRESULT OpenDirectConnection()
    {
        HRESULT hr = S_OK;
        HPEERENUM hPeerEnum;
        PEER_NODE_INFO ** ppNodeInfo = NULL;
        ULONG cItem = 0;
        hr = PeerGraphEnumNodes(g_hGraph, g_wzWsprId, &hPeerEnum);
        if (SUCCEEDED(hr))
        { cItem = 1; // wanted only one matching result.
            hr = PeerGraphGetNextItem(hPeerEnum, &cItem, (PVOID*) &ppNodeInfo);
            PeerGraphEndEnumeration(hPeerEnum);}

        if (SUCCEEDED(hr))
        {
            if ( (cItem == 0) || (NULL == ppNodeInfo) )
                hr = E_FAIL;
            else if ((*ppNodeInfo)->cAddresses == 0)
                hr = E_FAIL;}

        if (SUCCEEDED(hr))
        {
            hr = PeerGraphOpenDirectConnection(g_hGraph, g_wzWsprId,&(*ppNodeInfo)->pAddresses
[0], &g_ullConnection);
            if (SUCCEEDED(hr))
            { WCHAR wzMsg[MAX_PATH];
                wsprintf(wzMsg, L"ID: %08X", g_ullConnection);
                SetStatusPart(SB_PART_MESSAGE, wzMsg);}}

        if (NULL != ppNodeInfo)
        { PeerGraphFreeData(ppNodeInfo);
            ppNodeInfo = NULL;}
```

www.docin.com

```
        return hr;
}

// 此函数创建一个 identity 供 PNRP 使用
HRESULT CreateIdentity()
{
return PeerIdentityCreate(L"GraphChat",L"GraphChatMember",(HCRYPTPROV)NULL,&g_pwzIdentity);
}

// 此函数删除一个 identity
HRESULT DeleteIdentity()
{
    HRESULT hr = S_OK;
    if (g_pwzIdentity)
    { hr = PeerIdentityDelete(g_pwzIdentity);
        g_pwzIdentity = NULL;}
    return hr;
}

// 此函数创建一个 graph
HRESULT CreateGraph(PWSTR wzCreatorId, PWSTR wzGraphId)
{
    HRESULT hr = S_OK;
    PEER_GRAPH_PROPERTIES props = {0};
    WCHAR wzMessage[MAX_CHAT_MESSAGE] = {0};
    CleanupGraph();
    PeerGraphDelete(wzGraphId, wzCreatorId, L"SampleChatGraph");

    props.dwSize = sizeof(props);
    props.pwzFriendlyName = L"Sample Graph";
    props.pwzGraphId = wzGraphId;
    props.pwzCreatorId = wzCreatorId;
    props.cPresenceMax = -1;
    wcsncpy(g_wzGraphId, wzGraphId, celems(g_wzGraphId));
    wcsncpy(g_wzPeerId, wzCreatorId, celems(g_wzPeerId));
    hr = PeerGraphCreate(&props, L"SampleChatGraph", NULL, &g_hGraph);
    if (SUCCEEDED(hr))
    { _snwprintf(wzMessage, celems(wzMessage), L"New graph created.  GraphName = %s",
wzGraphId);
        DisplaySysMsg(wzMessage);
        hr = PrepareToChat(wzGraphId, FALSE);}
    else
        DisplayHrError(L"Failed to create a new Graph", hr);
```

www.dosin.com

www.ckook.com

```
    if (SUCCEEDED(hr))
    {
        g_fSynchronized = TRUE;
        hr = PeerGraphListen(g_hGraph, PEER_GRAPH_SCOPE_GLOBAL, 0, GRAPHING_PORT);
        if (SUCCEEDED(hr))
        {   UpdateParticipant(wzCreatorId, TRUE);
            hr = RegisterAddress(g_wzGraphId);
            if (FAILED(hr))
                DisplayHrError(L"Error attempting to Register Graph ID.",hr);}
        else
            DisplayHrError(L"Failed call to PeerGraphListen", hr);}
    return hr;
}

// 此函数打开一个存在的 graph
HRESULT OpenGraph(WCHAR *wzPeerId, WCHAR *wzGraphId)
{
    HRESULT hr = S_OK;
    HRESULT hrOpen = S_OK;
    CleanupGraph();      // 释放先前的 graph 资源
    wcsncpy(g_wzGraphId, wzGraphId, celems(g_wzGraphId));
    wcsncpy(g_wzPeerId, wzPeerId, celems(g_wzPeerId));
    hrOpen = PeerGraphOpen(wzGraphId, wzPeerId, L"SampleChatGraph", NULL, 0, NULL, &g_hGraph);

    if (SUCCEEDED(hrOpen))
    {
        hr = PrepareToChat(wzGraphId, TRUE);
        if (SUCCEEDED(hr))
            UpdateParticipant(wzPeerId, TRUE);

        if (hrOpen == PEER_S_GRAPH_DATA_CREATED)
        {
            if (hr == HRESULT_FROM_WIN32(WSA_E_NO_MORE))
                MessageBox(GetWindow(g_hwndMain, GW_ENABLEDPOPUP), L"Nobody could
be found to connect to. Please try again later.", L"Info", MB_OK | MB_ICONWARNING);
            else if(FAILED(hr))
                DisplayHrError(L"Node could not connect to graph", hr);}
        else
        {
            hr = PeerGraphListen(g_hGraph,PEER_GRAPH_SCOPE_GLOBAL, 0, GRAPHING_PORT);
            if (SUCCEEDED(hr))
            {   UpdateParticipant(wzPeerId, TRUE);
                hr = RegisterAddress(g_wzGraphId);
```

www.docin.com

```
                if (FAILED(hr))
                        DisplayHrError(L"Error attempting to Register Graph ID.",hr);
            }
            else
                DisplayHrError(L"Error attempting to listen", hr);}
    }
    else
    { hr = hrOpen;
        DisplayHrError(L"Failed to open Graph", hr);}
    return hr;
}

// 在成功创建或者连接一个 graph 后调用的初始化函数
HRESULT PrepareToChat(WCHAR *wzGraphId, BOOL fConnect)
{
    PEER_ADDRESS addr;
    HRESULT hr = RegisterForEvents();
    if (SUCCEEDED(hr) && fConnect)
    {
        hr = DiscoverAddress(wzGraphId, &addr);
        if (SUCCEEDED(hr))
            hr = PeerGraphConnect(g_hGraph, NULL, &addr, NULL);
        if (FAILED(hr))
            DisplayHrError(L"Couldn't find/connect to Graph", hr);}
    return hr;
}

// 注册事件回调函数
HRESULT RegisterForEvents()
{
    HRESULT hr = S_OK;
    PEER_GRAPH_EVENT_REGISTRATION regs[] = {
        { PEER_GRAPH_EVENT_RECORD_CHANGED, &CHAT_MESSAGE_RECORD_TYPE },
        { PEER_GRAPH_EVENT_NODE_CHANGED,     0 },
        { PEER_GRAPH_EVENT_STATUS_CHANGED, 0 },
        { PEER_GRAPH_EVENT_DIRECT_CONNECTION, 0 },
        { PEER_GRAPH_EVENT_INCOMING_DATA, &WHISPER_MESSAGE_TYPE },
        { PEER_GRAPH_EVENT_CONNECTION_REQUIRED, 0 },
    };

    g_hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (g_hEvent == NULL)
    { DisplayHrError(L"CreateEvent call failed", E_OUTOFMEMORY);
```

www.dogn.com

www.ckook.com

```
        hr = HRESULT_FROM_WIN32(GetLastError());;}
    else
    {
        hr = PeerGraphRegisterEvent(g_hGraph,g_hEvent,celems(regs),regs, &g_hPeerEvent);
        if (FAILED(hr))
        {   DisplayHrError(L"PeerGraphRegisterEvents failed", hr);
            CloseHandle(g_hEvent);
            g_hEvent = NULL;}
    }
    if (SUCCEEDED(hr))
    {
        if (!RegisterWaitForSingleObject(&g_hWait, g_hEvent, EventCallback, NULL, INFINITE,
WT_EXECUTEDEFAULT))
        {
            hr = HRESULT_FROM_WIN32(GetLastError());
            DisplayHrError(L"Could not setup event callback", hr);
            if (g_hPeerEvent != NULL)
            {   PeerGraphUnregisterEvent(g_hPeerEvent);
                g_hPeerEvent = NULL;}
            CloseHandle(g_hEvent);
            g_hEvent = NULL;}
    }
    return hr;
}

// 处理由 graphing infrastructure 激活的事件的函数
VOID CALLBACK EventCallback(PVOID lpParam, BOOLEAN reason)
{
    HRESULT hr= S_OK;
    PEER_GRAPH_EVENT_DATA *pEventData = NULL;
    while (TRUE)
    {
        hr = PeerGraphGetEventData(g_hPeerEvent, &pEventData);
        if (FAILED(hr) || (hr == PEER_S_NO_EVENT_DATA) || (NULL == pEventData) )
            break;

        switch (pEventData->eventType)
        {
            case PEER_GRAPH_EVENT_RECORD_CHANGED:
                ProcessRecordChangeEvent(pEventData);
                break;
            case PEER_GRAPH_EVENT_STATUS_CHANGED:
                ProcessStatusChangeEvent(pEventData->dwStatus);
```

www.ckook.com

```
            break;
        case PEER_GRAPH_EVENT_NODE_CHANGED:
            ProcessNodeChangeEvent(pEventData);
            break;
        case PEER_GRAPH_EVENT_DIRECT_CONNECTION:
            ProcessDirectConnectionEvent(pEventData);
            break;
        case PEER_GRAPH_EVENT_INCOMING_DATA:
            ProcessIncomingDataEvent(pEventData);
            break;
        case PEER_GRAPH_EVENT_CONNECTION_REQUIRED:
            ProcessConnectionRequiredEvent(pEventData);
            break;
        default:
            break;}
    PeerGraphFreeData(pEventData);}
}

// 处理 PEER_GRAPH_EVENT_NODE_CHANGED 事件的函数
void ProcessNodeChangeEvent(PEER_GRAPH_EVENT_DATA *pEventData)
{
    HRESULT hr = S_OK;
    if (pEventData->nodeChangeData.changeType == PEER_NODE_CHANGE_CONNECTED)
        UpdateParticipant(pEventData->nodeChangeData.pwzPeerId, TRUE);
    else if (pEventData->nodeChangeData.changeType == PEER_NODE_CHANGE_DISCONNECTED)
        UpdateParticipant(pEventData->nodeChangeData.pwzPeerId, FALSE);
    else if ((pEventData->nodeChangeData.changeType == PEER_NODE_CHANGE_UPDATED)
&&(pEventData->nodeChangeData.ullNodeId == 0))
    {
        SetStatus(L"Address Change Occured.");
        hr = RegisterAddress(g_wzGraphId);
        if (FAILED(hr))
            DisplayHrError(L"Error attempting to Register Graph ID.",hr);}
}

// 处理 PEER_GRAPH_EVENT_STATUS_CHANGED 事件的函数
void ProcessStatusChangeEvent(DWORD dwStatus)
{
    PWSTR pwzMsg = L"";
    PEER_GRAPH_PROPERTIES * pGraphProperties = NULL;
    WCHAR wzChatTitle[MAX_PEERNAME + 30] = {0};
    HRESULT hr = S_OK;
    wsprintf(wzChatTitle, L"Offline");
```


www.ckook.com

```
hr = PeerGraphGetProperties(g_hGraph, &pGraphProperties);
pwzMsg = L"";
if (SUCCEEDED(hr))
    if (pGraphProperties != NULL)
        if (dwStatus & PEER_GRAPH_STATUS_LISTENING)
        {   pwzMsg = L"Listening";
            wsprintf(wzChatTitle, L"Waiting to chat in %s", pGraphProperties->pwzGraphId);}

SetStatusPart(SB_PART_LISTENING, pwzMsg);
  pwzMsg = L"";
  if (SUCCEEDED(hr))
      if (pGraphProperties != NULL)
          if (dwStatus & PEER_GRAPH_STATUS_HAS_CONNECTIONS)
          {   pwzMsg = L"Connected";
              wsprintf(wzChatTitle, L"Chatting in %s", pGraphProperties->pwzGraphId);}

  PeerGraphFreeData(pGraphProperties);
  SetStatusPart(SB_PART_CONNECTED, pwzMsg);
  pwzMsg = L"";
  if (dwStatus & PEER_GRAPH_STATUS_SYNCHRONIZED)
  {
      if (!g_fSynchronized)
      {
          g_fSynchronized = TRUE;
          if (!(dwStatus & PEER_GRAPH_STATUS_LISTENING))
          {
              HRESULT hr = PeerGraphListen(g_hGraph, PEER_GRAPH_SCOPE_GLOBAL, 0,
GRAPHING_PORT);
              if (SUCCEEDED(hr))
              {   hr = RegisterAddress(g_wzGraphId);
                  if (FAILED(hr))
                      DisplayHrError(L"Error attempting to Register Graph ID.",hr);}
              else
                  DisplayHrError(L"Error attempting to listen", hr);
          }
      }
      pwzMsg = L"Synchronized";
  }
  SetStatusPart(SB_PART_SYNCHRONIZED, pwzMsg);
  SetWindowText(g_hwndChatLabel, wzChatTitle);
  EnableDisconnectMenu();
}

// 处理 PEER_GRAPH_EVENT_RECORD_CHANGED 事件的函数
```

www.ckook.com

```
void ProcessRecordChangeEvent(PEER_GRAPH_EVENT_DATA *pEventData)
{
    if (pEventData->recordChangeData.changeType == PEER_RECORD_ADDED)
        if (IsEqualGUID(&pEventData->recordChangeData.recordType, &CHAT_MESSAGE_RECORD_TYPE))
        {
            PEER_RECORD *pRecord = NULL;
            HRESULT hr = PeerGraphGetRecord(g_hGraph, &pEventData->recordChangeData.recordId,
&pRecord);
            if (SUCCEEDED(hr))
            {   DisplayMessage(pRecord->pwzCreatorId, (PCWSTR) pRecord->data.pbData);
                PeerGraphFreeData(pRecord);}
            else
                DisplayHrError(L"Error adding chat record", hr);
        }
}

// 处理 PEER_GRAPH_EVENT_DIRECT_CONNECTION 事件的函数
void ProcessDirectConnectionEvent(PEER_GRAPH_EVENT_DATA *pEventData)
{
    switch (pEventData->connectionChangeData.status)
    {
        case PEER_CONNECTED:
            SetStatusPart(SB_PART_MESSAGE, L"DC Connected");
            if (g_ullConnection == pEventData->connectionChangeData.ullConnectionId)
                SendDirectData();
            break;
        case PEER_CONNECTION_FAILED:
            g_ullConnection = 0;
            SetStatusPart(SB_PART_MESSAGE, L"DC Failed");
            break;
        case PEER_DISCONNECTED:
            g_ullConnection = 0;
            SetStatusPart(SB_PART_MESSAGE, L"DC Disconnected");
            break;
        default: break;}
}

// 处理 PEER_GRAPH_EVENT_INCOMING_DATA 事件的函数
void ProcessIncomingDataEvent(PEER_GRAPH_EVENT_DATA *pEventData)
{
    WCHAR wzSentFrom[MAX_PEERNAME + 20] = {0};
    HPEERENUM hPeerEnum = NULL;
    HRESULT hr;
```

www.ccin.com

www.ckook.com

```
    ULONG cItem;
    g_ullConnection = (ULONGLONG)pEventData->incomingData.ullConnectionId;
    hr = PeerGraphEnumConnections(g_hGraph, PEER_CONNECTION_DIRECT, &hPeerEnum);
    if (SUCCEEDED(hr))
    {
        PEER_CONNECTION_INFO ** ppConnections = NULL;
        hr = PeerGraphGetItemCount(hPeerEnum, &cItem);
        if (SUCCEEDED(hr))
            hr = PeerGraphGetNextItem(hPeerEnum, &cItem, (PVOID*) &ppConnections);

        if (SUCCEEDED(hr))
        {
            ULONG i;
            for (i=0; i < cItem; i++)
            {
                PEER_CONNECTION_INFO * pConnectionInfo = ppConnections[i];
                if (g_ullConnection == pConnectionInfo->ullConnectionId)
                {   wsprintf(wzSentFrom, L"Whisper from %s", pConnectionInfo->pwzPeerId);
                    DisplayMessage(wzSentFrom, (PCWSTR) pEventData->incomingData.data.pbData);
                    break;}
            }
            PeerGraphFreeData(ppConnections);
        }
        PeerGraphEndEnumeration(hPeerEnum);
    }
    PeerGraphCloseDirectConnection(g_hGraph, g_ullConnection);
    g_ullConnection = 0;
}

// 处理 PEER_GRAPH_EVENT_CONNECTION_REQUIRED 事件的函数
void ProcessConnectionRequiredEvent(PEER_GRAPH_EVENT_DATA *pEventData)
{
    HRESULT hr = S_OK;
    PEER_ADDRESS addr;
    hr = DiscoverAddress(g_wzGraphId, &addr);
    if (SUCCEEDED(hr))
        hr = PeerGraphConnect(g_hGraph, NULL, &addr, NULL);
}

// ProcessDirectConnectionEvent 会调用此函数发送消息
void SendDirectData()
{
    ULONG    cb = 0;
```

www.ccnr.com

```
        HRESULT hr = S_OK;
        WCHAR   wzSentTo[MAX_PEERNAME + 20];
        cb = (ULONG)(sizeof(WCHAR) * (wcslen(g_pwzWsprMsg)+1));
        hr = PeerGraphSendData(g_hGraph, g_ullConnection, &WHISPER_MESSAGE_TYPE, cb,
g_pwzWsprMsg);
        if (SUCCEEDED(hr))
        { wsprintf(wzSentTo, L"Whisper to %s", g_wzWsprId);
            DisplayMessage(wzSentTo, g_pwzWsprMsg);}
        else
            DisplayHrError(L"unable to send data",hr);

        if (g_pwzWsprMsg)
        { free(g_pwzWsprMsg);
            g_pwzWsprMsg = NULL;}
    }

    // 此函数添加一条聊天消息给 graph
    HRESULT AddChatRecord(PCWSTR pwzMessage)
    {
        HRESULT        hr = S_OK;
        PEER_RECORD record = {0};
        GUID           idRecord;
        FILETIME       ftExpire;
        GetSystemTimeAsFileTime(&ftExpire);

        hr = PeerGraphUniversalTimeToPeerTime(g_hGraph, &ftExpire, &ftExpire);
        if (SUCCEEDED(hr))
        { *((ULONGLONG UNALIGNED*)&ftExpire) += 600000000;   // Now + 60 seconds
            record.dwSize = sizeof(record);
            record.data.cbData = ( ((ULONG) wcslen(pwzMessage)) +1) * sizeof(WCHAR);
            record.data.pbData = (PBYTE) pwzMessage;
            record.ftExpiration = ftExpire;
            record.type = CHAT_MESSAGE_RECORD_TYPE;
            hr = PeerGraphAddRecord(g_hGraph, &record, &idRecord);
            if (FAILED(hr))
                DisplayHrError(L"Failed to add a chat record to the graph.", hr);}
        else
            DisplayHrError(L"Failed to add a chat record to the graph.", hr);

        return hr;
    }

    // 此函数删除 graph 使用的资源、全局变量
```


www.ckook.com

```
void CleanupGraph()
{
    if (g_fPnrpRegistered)
        UnregisterAddress(g_wzGraphId);
    if (g_hPeerEvent != NULL)
    { PeerGraphUnregisterEvent(g_hPeerEvent);
        g_hPeerEvent = NULL;}
    if (g_hWait != NULL)
    { UnregisterWaitEx(g_hWait, INVALID_HANDLE_VALUE);
        g_hWait = NULL;}
    if (g_hEvent != NULL)
    { CloseHandle(g_hEvent);
        g_hEvent = NULL;}
    if (g_hGraph != NULL)
    { PeerGraphClose(g_hGraph);
        g_hGraph = NULL;}
    if (g_pwzWsprMsg)
    { free(g_pwzWsprMsg);
        g_pwzWsprMsg = NULL;}

    ClearParticipantList();
    g_fSynchronized = FALSE;
}

HRESULT DiscoverAddress(WCHAR *wzGraphId, PEER_ADDRESS *pPeerAddress)
{
    HRESULT hr;
    HCURSOR hCursor;
    PWSTR pwzUnsecuredName = NULL;
    if(wzGraphId != NULL)
        hr = PeerCreatePeerName(NULL,wzGraphId,&pwzUnsecuredName);
    if(FAILED(hr))
    { DisplayHrError(L"PeerName could not be created", hr);
        return hr;}

    ZeroMemory(pPeerAddress, sizeof(PEER_ADDRESS));
    pPeerAddress->dwSize = sizeof(PEER_ADDRESS);
    hCursor = SetCursor(LoadCursor(NULL, MAKEINTRESOURCE(IDC_WAIT)));
    hr = PnrpResolve(pwzUnsecuredName, NULL, &pPeerAddress->sin6);
    SetCursor(hCursor);
    PeerFreeData(pwzUnsecuredName);
    return hr;
}
```

www.docin.com

www.ckook.com

```
HRESULT RegisterAddress(WCHAR *wzGraphId)
{
    HRESULT hr;
    PEER_NODE_INFO *pNodeInfo = NULL;
    PWSTR pwzUnsecuredName = NULL;
    if(wzGraphId != NULL)
        hr = PeerCreatePeerName(NULL,wzGraphId,&pwzUnsecuredName);
    if(FAILED(hr))
    { DisplayHrError(L"PeerName could not be created", hr);
        return hr;}

    hr = PeerGraphGetNodeInfo(g_hGraph, 0, &pNodeInfo);
    if (SUCCEEDED(hr))
    {
        hr = PnrpRegister(g_pwzIdentity, pwzUnsecuredName, NULL, pNodeInfo);
        if (SUCCEEDED(hr))
        {
            if (pNodeInfo->cAddresses >= 1)
            {   WCHAR wzTemp[512];
                SOCKADDR_IN6 * pAddress = &pNodeInfo->pAddresses->sin6;
                FormatAddress(pAddress, wzTemp, celems(wzTemp));
                SetStatusPart(SB_PART_ADDRESS, wzTemp);}
            g_fPnrpRegistered = TRUE;
        }
        else
            DisplayHrError(L"Failed to register node in PNRP", hr);

        PeerGraphFreeData(pNodeInfo);
    }
    else
        DisplayHrError(L"Could not get local node info", hr);
    PeerFreeData(pwzUnsecuredName);
    return hr;
}

HRESULT UnregisterAddress(WCHAR *wzGraphId)
{
    HRESULT hr;
    PWSTR pwzUnsecuredName = NULL;
    if(wzGraphId != NULL)
        PeerCreatePeerName(NULL,wzGraphId,&pwzUnsecuredName);

    hr = PnrpUnregister(g_pwzIdentity, pwzUnsecuredName, NULL);
```

www.ccin.com

www.ckook.com

```
    if (FAILED(hr))
        DisplayHrError(L"Failed to unregister node in PNRP", hr);
    else
        g_fPnrpRegistered = FALSE;

    PeerFreeData(pwzUnsecuredName);
    return hr;
}
```

6.2 JXTA

6.2.1 JXTA 简介

JXTA 是由 SUN 公司推出的一组开放的通用 P2P 协议，它允许任何接入网络的设备作为对等点进行通信和协同工作，如移动电话、PDA（个人数字助理）、PC（个人计算机）、服务器等。JXTA 并不依赖于某种编程语言，事实上，我们可以利用任何一种平台来实现 JXTA 协议。我们把某种平台的实现称为绑定。在本节中我们以 J2SE 平台绑定为例子，介绍 JXTA 的相关概念。

1. JXTA 是什么

JXTA 是专门用于做 P2P 计算的开放性网络计算平台，它的目的是为开发新的应用程序提供基本的模块和服务。

JXTA 是 Juxtapose 的缩写，意思是“并列，并置”。JXTA 不同于传统的分布式应用模式，如客户端/服务器或者浏览器/服务器。

JXTA 给开发 P2P 应用程序提供了一整套开放协议和开源实现。同时，JXTA 对 P2P 节点的共同行为做了标准化，包括节点发现、自组织、发现和通告网络服务、相互通信以及监视其他节点等。

JXTA 协议独立于编程语言和传输协议。Java、C/C++、Perl 以及其他语言都可以用来实现 JXTA 协议，同时 JXTA 传输可以构建在 TCP/IP、HTTP、Bluetooth、HomePNA 或者其他传输协议上。

开发者可以利用 JXTA 开发相互操作的 P2P 服务或应用。因为协议的独立性，异构网络的设备可以使用不同的协议栈相互通信。

2. 为什么选择 JXTA

目前网络应用在内容和接入设备方面持续增长，使得 P2P 技术也变得越来越流行。基于 P2P 的日常软件涵盖了多个方面，如文件共享、分布式计算、即时通信以及分布式存储等。虽然这些软件有着不同的功能，但它们在节点发现、内容搜索、数据传送和文件传送等方面有很多的相似点。不过目前的应用开发却是非常低效的，开发者们不停地在解决同样的问题，或者说在“重新造轮子”——几乎所有的应用都局限在单一的平台上，相互之间不能进行通信或者共享资源。

www.docin.com

SUN 公司推出 JXTA 的首要目的就是提供一个平台，在这个平台上有 P2P 网络所必需的基本功能。除此之外，JXTA 还致力于解决已有 P2P 系统的一些问题。

（1）协同性

JXTA 使得对等点能提供多种 P2P 服务，用于对等点之间的相互通信。

（2）平台独立性

JXTA 独立于编程语言、传输协议和开发平台。

（3）通用性

JXTA 允许任何数字设备接入，而不仅仅局限于 PC 或者某一特定平台。

在 P2P 网络中，节点往往都处在因特网的边缘。由于没有固定的接入时间和网络地址，所以标准的 DNS 服务器并不会覆盖这些节点。JXTA 设计了一个和传统的地址服务不一样的机制来覆盖所有的节点。通过使用 JXTA 定义的 ID，网络节点就可以在网络中穿行、更改传输协议、更改网络地址等，即使节点暂时不在线也能被其他节点找到。

6.2.2 JXTA 的平台结构

1．总体描述

基于 JXTA 的软件架构可以分为 3 层，层次结构示意图如图 6-14 所示。

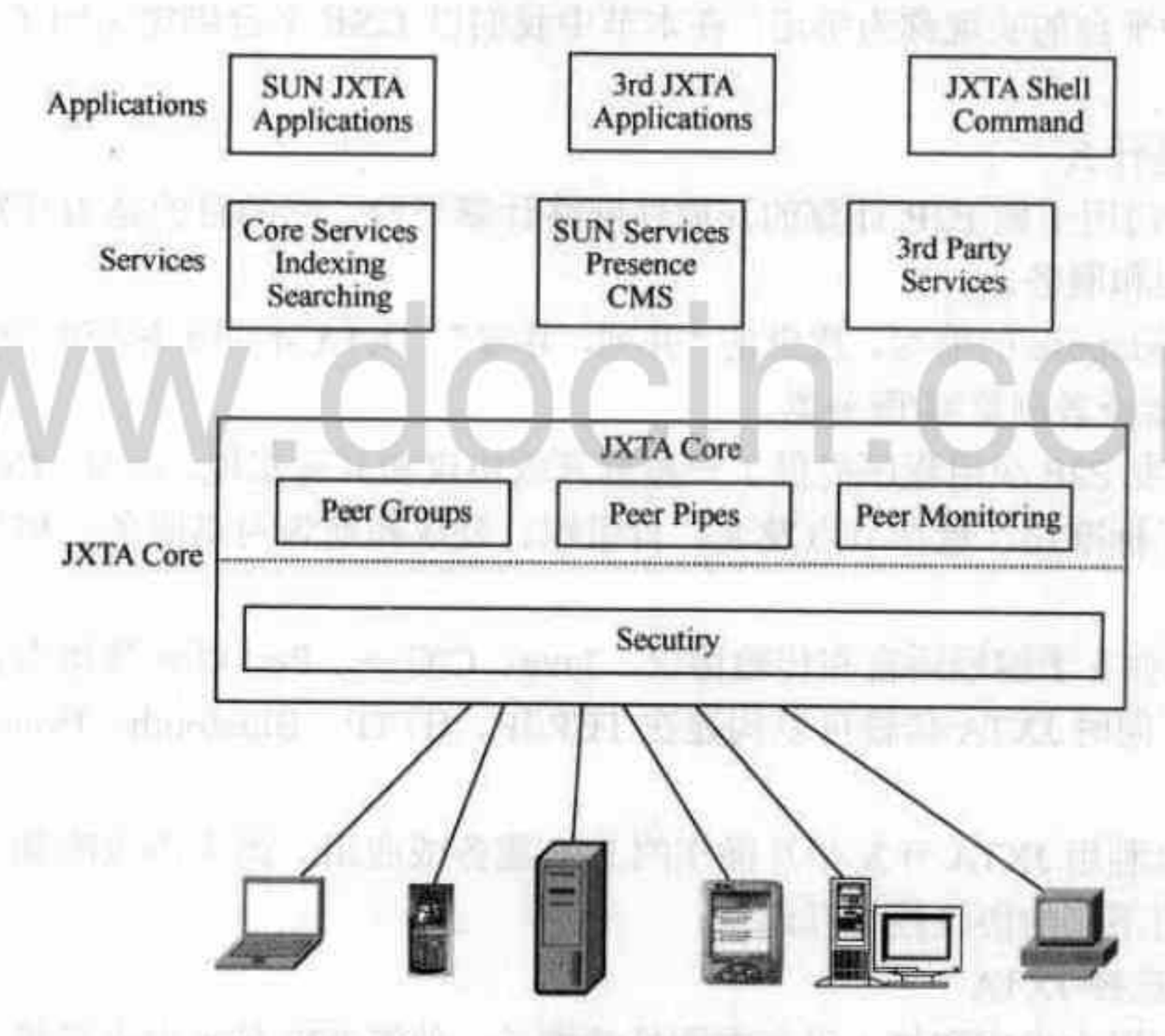


图 6-14 JXTA 软件架构示意图

核心层（JXTA Core）封装了 P2P 网络的一些最基础的要素，包括为 P2P 应用程序提供关键机制的模块。这些关键机制包括发现、传输、节点加入、创建组还有相关的安全原语。

服务层（Services）包含一些网络服务。这些服务可能并不是 P2P 网络所必不可少的，但却非常有用。这些服务包括搜索和索引、目录、存储系统、文件共享、分布式文件系统、资

www.docin.com
www.ckook.com

源聚合和租借、协议转换、验证以及PKI服务等。

应用层（Applications）包括一些应用的实现，比如P2P即时消息、文件和资源共享、娱乐内容管理和转发、P2P电子邮件系统、分布式验证系统等。

服务层和应用层之间并没有严格的界限。一个客户的应用可能是另一个客户的服务。整个系统是按照模块化来设计的，开发者可以选择服务和应用来满足需求。

2．JXTA的组件

JXTA网络是由很多相互连接的对等点组成的。对等点之间可以自由组成点组来对外提供服务，如文件共享、聊天等。

JXTA对等点把提供的服务以XML文档的格式发布出去，这些文档称为通告。通过使用通告，网络中的其他对等点能知道如何连接和访问对等点的服务。

JXTA对等点之间使用管道来交换消息。管道是一种异步、单向的消息传输机制。消息是封装了路由、摘要以及信任书等的XML文档。管道在使用时要绑定到端点，如IP地址和TCP端口。本章将陆续对这些概念进行详细介绍。

3．JXTA结构的关键特征

JXTA有3个关键特征，这些特征使得JXTA和其他分布式网络模型有所区别。

（1）使用XML文档对资源进行描述。

（2）使用从管道到对等点，从对等点到端点的抽象地址映射机制，不依赖于类似DNS的中心命名机制。

（3）采用了一种全局通用的对等点命名机制，每个对等点都有可以唯一标识的ID。

6.2.3 JXTA的基本概念

本节将介绍JXTA的关键技术和基本概念。

www.docin.com

1．对等点

对等点（节点）是实现了一个或多个JXTA协议的任意一个网络设备，这些设备可以是传感器、电话、PDA、PC、服务器甚至是大型机。每个对等点都是独立运行的，各个对等点有自己的唯一标识PeerID，而且相互之间可以进行通信。

对等点发布一个或多个可以由JXTA协议使用的网络接口。每个网络接口都是一个对等点的端点，并可以由端点唯一标识。对等点之间通过端点来建立点到点连接。

不过对等点之间并不是必须要有点到点的真实连接。被物理网络或者网络配置隔离（如NAT、防火墙、代理等）的对等点之间可以通过中间节点来转发消息。

通常将多个对等体配置成能够在网络中容易相互发现，具有短期或长期的联系，这样的对等点组织叫做点组。

2．点组

点组是多个对等点的集合，用来提供一组服务。对等点是自发地组成点组的，每个点组都有自己的唯一标识。点组的成员关系策略由点组自己确定，可以是开放的，也可以是高度安全和受限的。开放的点组每个对等点可以自由加入，而高度安全和受限点组需要加入的对等点有足够的信任度。

一个对等点可以同时加入多个点组。对等点默认都会加入NetPeerGroup点组，而且所有

www.ckook.com

的对等点都属于这个点组。除此之外，对等点可以选择加入其他点组。

JXTA 协议描述了对等点如何发布、发现、加入和监视点组，并没有规定什么时候因为什么原因创建点组。

引入点组是基于下面几个原因。

（1）创建一个安全的环境

点组创建了一个可以强制实行特定的安全策略的局部受控区域。安全策略的形式可以是多种多样的，最简单的可以是以文本形式交换的用户名和密码，复杂点的可以是公钥加密算法。点组边界使得成员可以访问和发布受保护的内容。点组形成一个逻辑区域，来限制外界对点组资源的访问。

（2）创建一个特定服务的环境

可以通过点组建立一个具有特定服务功能的局部区域。比如，对等点可以组合到一起来实现一个文件共享或者 CPU 共享的网络。点组可以把网络划分一些专门提供特定服务的抽象区域。点组边界规定了查找该组资源时的范围。

（3）创建一个监控的环境

点组允许对等点以任何目的来检测组内的对等点。点组同时形成一个有层次的父亲——孩子关系，每个点组有一个父亲。搜索请求在组内传播，而点组通告除了在组内发布外还会发布到父亲点组中。

点组提供了一系列服务，称为点组服务。JXTA 定义了一组核心的点组服务。额外的服务可以根据需求来开发。比如两个对等点加入同一个点组才能通过服务进行交流。

JXTA 提供的核心点组服务包括以下几种。

（1）发现服务

发现服务是对等点用来查找点组资源的，资源可以是对等点、点组、管道或服务等。

（2）成员关系服务

成员关系服务允许当前的点组成员拒绝或接受一个新的点组成员。如果一个对等点想要加入某个点组，首先要找到该点组的一个成员，然后发送加入请求。当前点组的成员可以拒绝或接受这个请求。成员关系服务可能以强制成员投票或者指定一个代表的方式来处理新成员的加入请求。

（3）访问服务

访问服务用于验证从一个对等点到另一个对等点的请求。收到请求的对等点向请求节点提供信任度和关于访问请求是否被允许的信息。

注意：并非点组内所有行为都要经过访问服务的检查，只有那些被限制于特定对等点的行为才会被检验。

（4）管道服务

管道服务用来创建和管理点组成员间的管道通信。

（5）解析服务

解析服务用来给其他对等点发送普通的查询请求。对等点可以定义和交换查询来发现它们所需要的任何信息，比如某个服务的状态或者管道端点的状态。

（6）监控服务

监控服务用来允许一个对等点监控点组内其他对等点的行为。

www.ckook.com

不过，不是每个点组都要实现上面提到的所有服务。一个点组可以实现它认为有用的服务，而且可以依赖于默认的 NetPeerGroup 点组来提供某些非关键核心服务的实现。

3．网络服务

对等点之间通过相互协作和通信来发布、发现和调用网络服务。一个对等点可以发布多个服务。对等点通过对等点发现协议来找到网络服务。

JXTA 协议里定义了下面两个层次的网络服务。

（1）对等点服务

对等点服务只能通过访问发布它的对等点来访问服务。如果发布服务的对等点失效了，该服务也就随之失效。服务的多个实例可以运行在不同的对等点上，但每个实例只发布它自己的服务通告。

（2）点组服务

点组服务是由多个运行在点组成员上的服务实例组成的，每个实例都可能和其他实例之间进行交互。如果一个对等点失效，点组服务不受影响，因为在其他点组成员上运行的服务仍然可以被访问。点组服务是作为点组通告的一部分发布的。

网络服务可以预先安装在对等点上，也可以从网络上加载。为了运行一个服务实例，对等点需要找到符合对等点运行环境的一个服务实现。从网上查找、下载及安装一个服务的过程和从 Internet 上搜索一个 Web 页、获得并安装所需插件程序的过程是相似的。

4．模块

JXTA 中的模块是一个抽象概念，代表在 JXTA 网络里实现了某个“行为”的一段“代码”。最常见的“行为”的例子就是能在对等点上实例化的网络服务。模块本身并不指定这个“代码”是什么，它可能会是一个 Java 类、一个 JAR 文件、一个动态链接库、一组 XML 消息或者一段脚本等。模块行为的实现留给了模块的实现者。比如，模块可以用来代表网络服务不同平台上的实现，如 Java 平台、微软的 Windows 平台或者 SUN 的 Solaris 平台。

模块提供一个对等点实例化新“行为”的抽象表示。当对等点浏览或者加入一个新点组时，它可能会发现需要实例化的新“行为”。比如当加入一个点组时，对等点可能需要学习一个仅在这个点组中使用的查找技术。为了加入该点组，对等点必须实例化这个新的搜索服务。

模块框架能以和平台无关的方式来描述和通告“行为”，并允许对等点以任何语言和方式来实例化“行为”。比如一个对等点可以选择用 Java 或者 C 语言来实例化某个特定的“行为”。

JXTA 能描述和发布与平台无关的“行为”，这一点对支持由异构节点组成的点组来说很重要。模块通告使得对等点可以以平台无关的方式来描述一种“行为”。JXTA 平台就是使用模块通告来描述自己的。

模块包含模块类、模块规范和模块实现。

（1）模块类

模块类主要用来通告“行为”的存在。模块类的定义描述了模块期望的行为和期望的绑定平台。每个模块类通过一个唯一的 ID 来标识，称为 ModuleClassID。

（2）模块规范

模块规范主要用来访问一个模块。它包含了访问或调用该模块的所有必需信息。比如，以服务为例，模块规范包含用来和服务通信的管道通告。

一个模块规范就是一种说明该模块类所包含的功能的方式。对于一个给定的模块类可以

www.ckook.com

有多个模块规范。每个模块规范通过一个 ID 来标识，为 ModuleSpecID。ModuleSpecID 包含了 ModuleClassID（ModuleClassID 嵌入在 ModuleSpecID 中），用来指明相关的模块类。

模块规范暗含了网络的兼容性。一个模块规范的所有实现必须使用相同的协议，而且要相互兼容，即使它们可能是用不同的语言开发的。

（3）模块实现

模块实现是给定模块规范的实现。一个模块规范可能会有多个实现。每个实现都包含了 ModuleSpecID，指明它所实现的模块规范。

模块可以用在点组服务中，也能用在对等点服务中。JXTA 使用模块来标识服务的存在（模块类）、服务的规范（模块规范）和服务的实现（模块实现）。模块的每个部分都有相关的通告，可以发布给其他 JXTA 对等点。

举个 JXTA 发现服务的例子。发现服务有一个 ModuleClassID，该 ID 表明它从抽象功能上来说是个发现服务。每个发现服务可以有多个规范，相互之间可能不兼容。某个点组可能会采用组的大小和网络状况裁剪过的发现策略，而其他组可能会使用别的策略。每个规范有一个 ModuleSpecID，该 ID 中包含了发现服务的 ModuleClassID。对于每个规范，可能会有多个实现，每个实现都包含了相同的 ModuleSpecID。

总之，每个模块类可以有多个模块规范，模块规范之间可能互相不兼容。不过，每个模块规范的多个实现之间必须兼容。

5．消息

消息是 JXTA 对等点之间相互传送的对象，是通信的基本单元。消息通过管道服务和端点服务发送和接收。一般情况下，应用程序使用管道服务创建、发送和接收消息，而不需要直接使用端点服务。如果要使用端点服务，应用程序必须要知道 JXTA 的网络拓扑。

一个消息是由串有序的消息元素组成的，消息元素就是一对名称和内容。因此，消息本质就是一组键/值对，它的内容可以是任何类型。

JXTA 协议就是关于对等点之间交换消息的规范。每个平台绑定都要描述如何在消息和一个原始数据结构之间转换。原始数据结构可能是 Java 对象或者 C 结构体。

消息有两种类型：XML 格式和二进制格式。JXTA 的 J2SE 平台绑定使用二进制格式来封装消息，以减少消息的开销。服务可以选择使用最合适的消息格式。如果服务要求消息紧凑时，可以使用二进制格式来封装消息，其他服务可以使用 XML 格式。XML 格式中的二进制数据可以使用 BASE64 进行编码。

用 XML 格式的消息来定义协议，可以使不同的对等体参与到这个协议中。因为数据是有标签的，所以每个对等体可以选择和自己能力、角色最匹配的方式来实现协议。如果一个对等点只需要消息的一个子集，通过使用 XML 数据标签，对等点就可以取出自己感兴趣的部分。比如，一个对等点资源受限，不能处理全部数据，它可以通过使用标签从数据中抽取出自己能处理的部分并忽略掉其他部分。

6.2.4 JXTA 的网络架构

1．组织形式

JXTA 网络是由多个对等点组成的 Ad hoc、多跳的自适应网络。网络连接可能是短暂的，

www.ckook.com

对等点之间的消息路由也可能不确定，对等点可以随时加入或离开网络，路由可能会比较频繁地改变。

只要对等点能使用 JXTA 协议进行通信，它们可以是任何形式的。JXTA 网络的组织形式并不是由 JXTA 框架决定的。在 JXTA 网络中，有 4 类典型的对等点。

（1）最小边缘点

最小边缘点可以发送和接收消息，但是不能缓存通告或者路由消息。运行在 PDA 或者蜂窝电话等资源受限设备上的对等点就可能会是最小边缘点。

（2）完整边缘点

完整边缘点可以发送和接收消息，而且通常会缓存通告。完整边缘点在自己缓存的通告里来发现消息，而且不转发任何发现消息。绝大多数对等点都是完整边缘点。

（3）集合点

集合点可以是任意一个对等点，它维护一个通告的缓冲池。集合点通过转发发现请求来帮助其他对等点发现资源。当对等点加入一个点组时，它会自动去寻找一个集合点。如果没有发现集合点，它就变成该点组的一个集合点。每个集合点维护一个列表，列表中有它所知道的其他集合点以及把它作为集合点的其他对等点。

每个点组维护自己的一组集合点，集合点的数量可以尽可能地多。只有点组的集合点才能看到该点组的特定查询请求。

边缘点把查询和发现请求发送给集合点，集合点再把这些请求转发到其他集合点。发现请求过程持续到某个对等点回应或者请求失效。消息中有个默认的 TTL 字段，最大跳数为 7。JXTA 通过在消息中的路径上保存对等点列表来避免回路。

（4）路由点

路由点维护着其他路由点的信息，而且把消息转发到对等点。一个对等点首先在它的本地缓冲区里查找路由信息。如果没有查到路由信息，对等点通过给路由点发消息来询问路由信息。路由点可以把不同的物理或逻辑网络桥接起来，给那些不能直接访问其他对等点的节点转发消息（处在 NAT 内）。任何对等点都可以实例化路由点或者集合点所需要的服务。

2．分布式共享资源索引（SRDI）

JXTA 的 JS2E 绑定支持分布式共享资源索引服务，简称 SRDI 服务。SRDI 可以提供查询请求传播机制。采用 SRDI 的集合点维护一个由边缘点发布的通告索引。当边缘点发布新的通告时，它们使用 SRDI 把通告索引传送给它们的集合点。通过这种集合点—边缘点的层次关系，查询只在集合点之间传播，可以很明显地减少在通告查询中涉及的对等点。

每个集合点维护点组内它所知道的集合点列表，列表可能从启动时预先配置的参数中获取。每个集合点周期性的选择任意个知道的集合点，然后把它们发送给自己知道的其他集合点。同时，集合点也周期性地删除没有回应的集合点。通过这种方式，集合点之间形成了一个松散的网络结构。

当一个对等点发布新的通告时，只有通告的索引被发送给集合点，这使得需要存储在集合点上的数据量最小。同时，集合点也把索引发送给其他集合点。

3．查询过程

JXTA 网络用的一个示例配置如图 6-15 所示。对等点 A 是一个边缘点，被配置成使用对等点 R1 作为集合点。当对等点 A 初始化一个发现或者查询请求后，首先发送到集合点 R1，

www.ckook.com

同时通过多播发送给子网中的其他对等点。

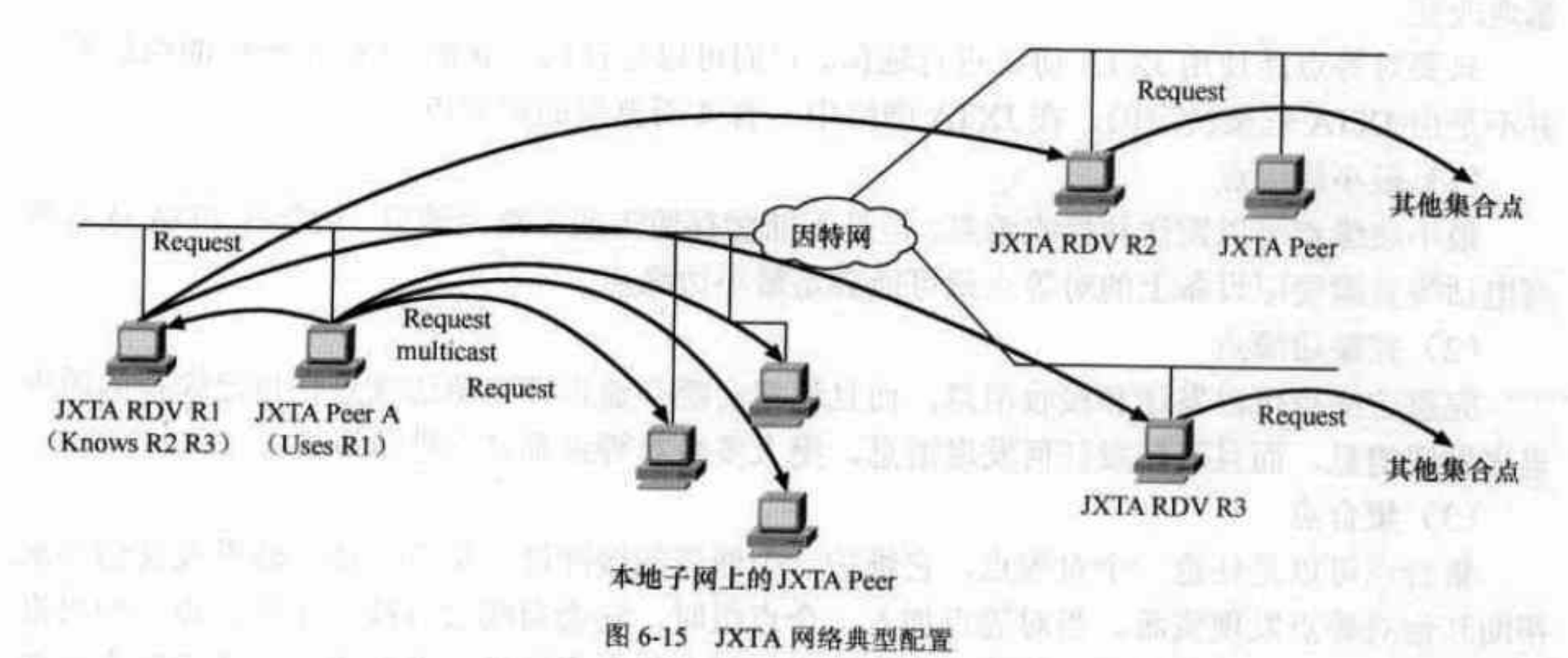


图 6-15　JXTA 网络典型配置

局部的查询可以使用多播或者广播发送给邻居节点。收到查询请求的对等点如果在它们的本地缓冲区找到符合查询条件的信息就直接给发送请求的对等点发送回应消息。

超出本地的查询被发送给集合点。集合点首先在它自己的缓冲区里查看是否有满足请求条件的信息。如果有，就直接回应发起请求的对等点，而且不再传播该请求。而如果它在 SRDI 里查找到符合请求信息的索引，就会通知发布资源的对等点，然后该对等点会直接回应发送请求的对等点。

如果集合点没有找到需要的信息，它会把请求转发给其他集合点，其他集合点重复相同的过程。发送请求中有一个跳数计数器，用来指定最大转发次数。

图 6-16 展示了 SRDI 服务是如何运行的。对等点 2 发布一个新的通告，把通告的索引存储在集合点 R3 上，同时可能被送给点组中的其他集合点。现在，对等点 1 给它的集合点 R1 发送一个查询请求。集合点 R1 首先检查它的本地缓存，如果没有发现就传播查询。当查询到达 R2 时，R2 通知对应对等点直接回应对等点 1。

4. JXTA 协议

JXTA 定义了一系列 XML 格式的消息来使对等点之间相互通信。JXTA 用这些消息来发现对等点、通告、网络资源等，也用来通信和路由。

目前有 6 种 JXTA 协议，下面分别予以描述。

（1）对等点发现协议（PDP，Peer Discovery Protocol）

对等点使用对等点发现协议来通告它们的资源（如对等点、点组、管道或服务），同时也用来发现其他对等点的资源。每个对等点的资源都用通告来描述和发布。

（2）对等点信息协议（PIP，Peer Information Protocol）

对等点使用对等点信息协议获取其他对等点的状态信息（上线时间、状态、最近的流量）。

（3）对等点解析协议（PRP，Peer Resolver Protocol）

对等点使用对等点解析协议来发送通用查询，同时接收该查询的回应。该查询可以发送给点组的所有成员或者特定的部分成员。对等点解析协议允许对等点定义和交换它们需要的任意信息，而不局限于 PDP 和 PIP 所定义的特定信息。

www.ckook.com

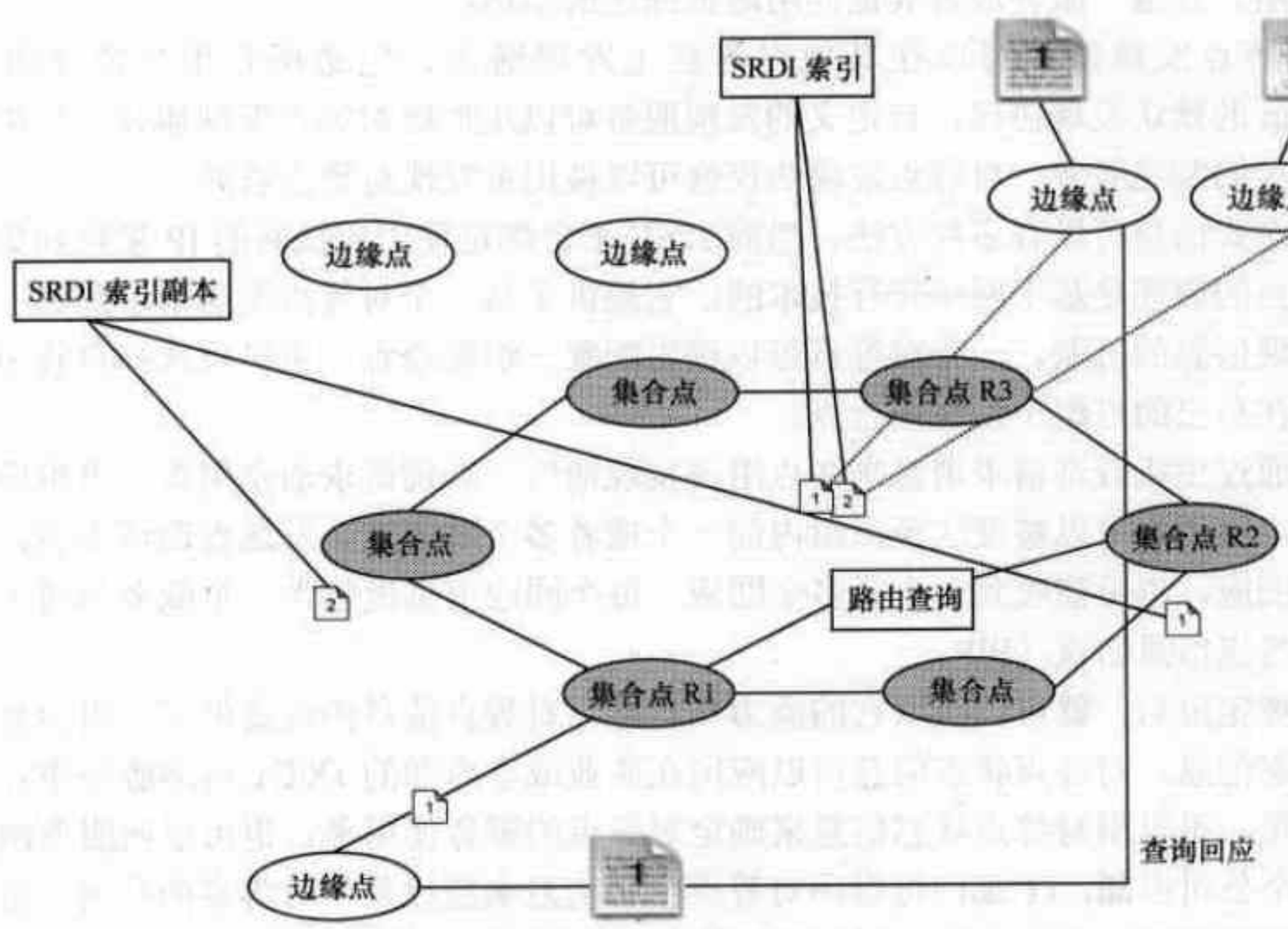


图 6-16 SRDI 服务运行示意图

（4）管道绑定协议（PBP，Pipe Binding Protocol）

对等点使用管道绑定协议在相互之间建立虚拟的通信信道，可以在管道上绑定两个或者更多的端点。

（5）端点路由协议（ERP，Endpoint Routing Protocol）

对等点使用端点路由协议来查找到目的地址的路由。路由信息包括一系列排序的路由点的 ID，通过这些排序的路由点可以给目的地发送消息。比如，消息通过路由 A 到路由 B 到最终目的地。

（6）集合点协议（RVP，Rendezvous Protocol）

集合点协议是 JXTA 用来预定或者传播服务的。在一个点组内，对等点可以成为集合点，或者只是使用集合点。集合点协议允许一个对等点给服务的所有实例发送消息。对等点解析协议和管道绑定协议用集合点协议来传播消息。

所有的 JXTA 协议都基于请求/应答模式，属于异步协议。JXTA 对等点使用某个协议来给点组中的一个或多个对等点发送查询请求，可能收不到回应，也可能收到一个或多个回应。比如，某个对等点使用管道绑定协议来给 NetPeerGroup 点组中所有对等点发送一个查询请求。随后，可能有多个对等点会回应这个请求。或者该对等点发送查询请求来查找名字为“aardvark”的管道。如果这个管道不存在，对等点就收不到任何回应。

JXTA 网络的对等点并不需要实现所有的 6 个协议，它们只需要实现它们可能会使用的。当前的 J2SE 平台绑定支持所有的 6 个协议。可以使用 Java 语言的 API 来访问 JXTA 协议支持的行为，比如发现对等点或者加入点组等。

（1）对等点发现协议（PDP）

对等点发现协议用来发现任何发布了的对等点资源。资源是用通告来描述的，它可能是

www.docin.com

对等点、点组、管道、服务或者其他能用通告描述的实体。

使用对等点发现协议可以在其他对等点上发现通告，它是所有用户定义组和默认NetPeerGroup的默认发现协议。自定义的发现服务可以用扩展对等点发现协议。如果一个点组没有它自己的发现服务，对等点发现协议就可以被用来发现对等点资源。

发现分布式信息可以有多种方法。当前J2SE平台绑定使用局域网的IP多播和集合点来实现。集合点的原理是基于网络爬行技术的，它提供了从一个对等点发送请求到另一个对等点来动态发现信息的机制。一个对等点可以预先配置一组集合点，也可以选择自我引导，通过动态定位在自己的点组中发现集合点。

对等点通过生成查询请求消息来在点组内发现通告。查询请求消息包含了点组信任书和对等点标识。该消息可以被发送至点组内的一个或者多个对等点。发送查询请求后，对等点可能收不到回应，也可能收到一个或多个回应。每个回应消息里包含一个或多个通告。

（2）对等点信息协议（PIP）

对等点被定位后，就可以获取它的能力和状态。对等点信息协议提供了一组消息用来获取对等点状态信息。对等点状态信息可以应用在商业或者内部的JXTA网络服务中。比如，在商业应用里，可以用对等点状态信息来确定对等点的服务使用率，也可以向服务消费者收费。而在一个公司内部，IT部门可以用对等点状态信息来监控某个对等点的行为，也可以对公司的网络流量重新做路由安排，以增强网络的总体性能。对等点状态信息还可以扩展到别的东西上面。

对等点信息协议使用Ping消息来获取目标对等点的状态。Ping消息里可以指定目标对等点只是简单的返回应答还是返回自己的全面状态信息。Ping 消息里包含了发送者的信任书、源ID、目的ID、生存时间以及对等点通告等。

（3）对等点解析协议

对等点解析协议使对等点能给其他对等点发送通用查询请求并接收回应。查询请求可以被发送给特定的对等点或者通过集合点服务传播到整个点组。对等点解析协议使用集合点服务把查询分发给多个对等点，使用单播消息给特定的对等点发送查询请求。

对等点解析协议是支持通用查询的基础协议。对等点发现协议和对等点信息协议都建立在对等点解析协议的基础上，只提供特定的查询和回应。对等点信息协议用来查询特定的状态信息，对等点发现协议用来发现对等点资源。对等点解析协议可以用来发送任何通用查询。比如，对等点可以利用解析协议来查找服务信息或管道端点状态等。

解析查询消息（Resolver Query Message）是点组的一个成员给另一个成员发送的查询请求。该消息包括了发送者的信任书、查询ID、特定的服务句柄和查询内容。每个对等点可以在点组解析服务中注册一个句柄来处理查询解析消息并生成回应消息。解析查询回应消息用来对一个解析查询消息作出回应。解析查询回应消息包括了发送者的信任书、查询ID、特定的服务句柄和回应内容。对等点可以发送多个解析查询请求，发送请求后，该对等点可能收不到回应，也可能收到一个或多个回应。

对等点也可以参与到分布式共享资源索引（SRDI）服务中。SRDI 是一个通用机制，在这种机制下JXTA服务可以和其他对等点一起使用分布式资源索引服务，这些对等点应该是比较有能力而且属于一个点组的，比如说集合点。使用索引可以把查询请求传送到最可能回应节点，而且可以重传对等点感兴趣的消息。

www.ckook.com

（4）管道绑定协议

点组成员使用管道绑定协议把管道端点绑定到管道通告上。基于虚拟连接的管道可以运行在任何物理传输网络上，比如说 TCP/IP。管道的每个端点保持这个虚拟连接，并在需要的时候通过查找管道的当前绑定端点来重新建立管道。

可以把管道看作一个抽象的命名队列，该队列支持创建、打开和解析、关闭、删除、发送以及接收等操作。管道实现可能会有所不同，但所有正确的实现都使用管道绑定协议把管道绑定到端点上。对等点通过抽象的创建操作把管道端点绑定到管道传送上。

对等点通过发送管道查询消息来查找和自身端点绑定到同一个管道通告上的对等点。为了获取最新信息，查询消息可能会忽略缓冲区的消息。在查询消息中也可以指定目的 ID，这样只有指定的对等点会回应该查询。

每个绑定到对应管道的对等点都会给发起请求的对等点发送回应消息。回应消息包含了管道 ID、创建了输入管道的对等点和一个布尔值，布尔值标识输入管道是否存在。

（5）端点路由协议

端点路由协议定义了一组请求和回应消息，主要用来发现路由消息。当一个对等点给另一个对等点发送消息时就需要路由消息。源对等点发送消息时，首先在本地缓冲区来查找是否有到目的地的路由信息。如果没有，源对等点发送一个路由查询请求给已知的路由点，询问路由信息。当路由点收到路由查询时，首先检查自己是否知道该路由信息，如果知道，路由点以集合的方式返回路由信息。

任何对等点都可以查询路由点来获取路由信息，而且点组的任何成员都可以成为路由点。路由点会缓存已知的路由信息。

路由信息包括源对等点 ID、目的对等点 ID、生存字段和一个路由点的排序序列。路由点 ID 序列可能并不完整，但至少存在一个路由点 ID。

对等点使用路由查询请求获取路由信息，在请求中可以指定是否动态查询新的路由。

路由应答消息是路由点回应路由查询请求的。消息里包含了目的 ID、路由器的 ID 和路由器的对等点通告。该路由器有到目的地的路由信息。

（6）集合点协议

集合点协议负责一个点组内消息的传播，不同的点组有不同的传播消息的方法。集合点协议只定义了以下简单的协议标准。

① 可以让对等点连接（传播消息和接收传播的消息）。

② 控制消息的传播（TTL，环路探测）。

对等点解析协议和管道绑定协议都用集合点协议来传播消息。

6.2.5 JXTA 开发环境

本节将以 JXTA 的 J2SE 平台绑定为例，给读者介绍如何搭建一个 JXTA 应用的开发环境。开发基于 J2SE 平台的 JXTA 应用需要安装的软件包括 JDK、Netbeans 和 JXTA 开发库。

1. 安装 JDK

JDK 是许多 Java 专家最初使用的开发环境，尽管目前已经有很多第三方开发工具即 JDK 仍然是 Java 开发的重要工具。JDK 包含了所有编写、运行 Java 程序所需要的工具，即

www.ckook.com

Java 基本组件、库、Java 编译器、Java 解释器等。其核心 API 是一些预定义的类库，开发人员使用这些类来访问 Java 语言的基本功能。

JDK 的最新版本可以在 Sun 的官方网站 http://Java.sun.com 上下载，目前用得比较多的是 jdk1.5。下载后就可以进行安装，双击运行“jdk-1_5_0-windows-i586.exe”文件，将出现 JDK 的安装界面，如图 6-17 所示。

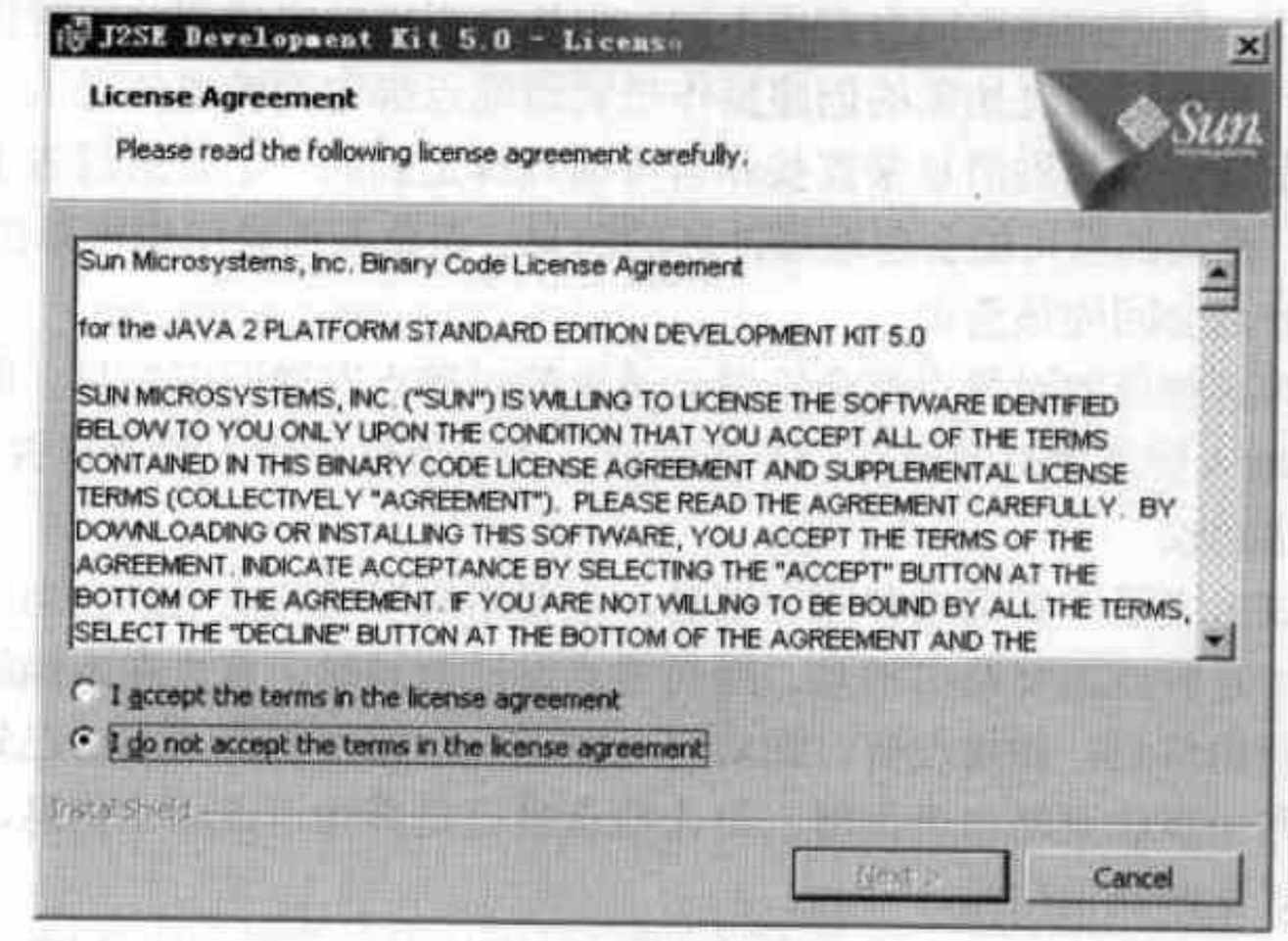


图 6-17　JDK 安装界面 1

接受安装协议，单击“Next”按钮，这时会让用户选择安装组件和安装目录，如图 6-18 所示。

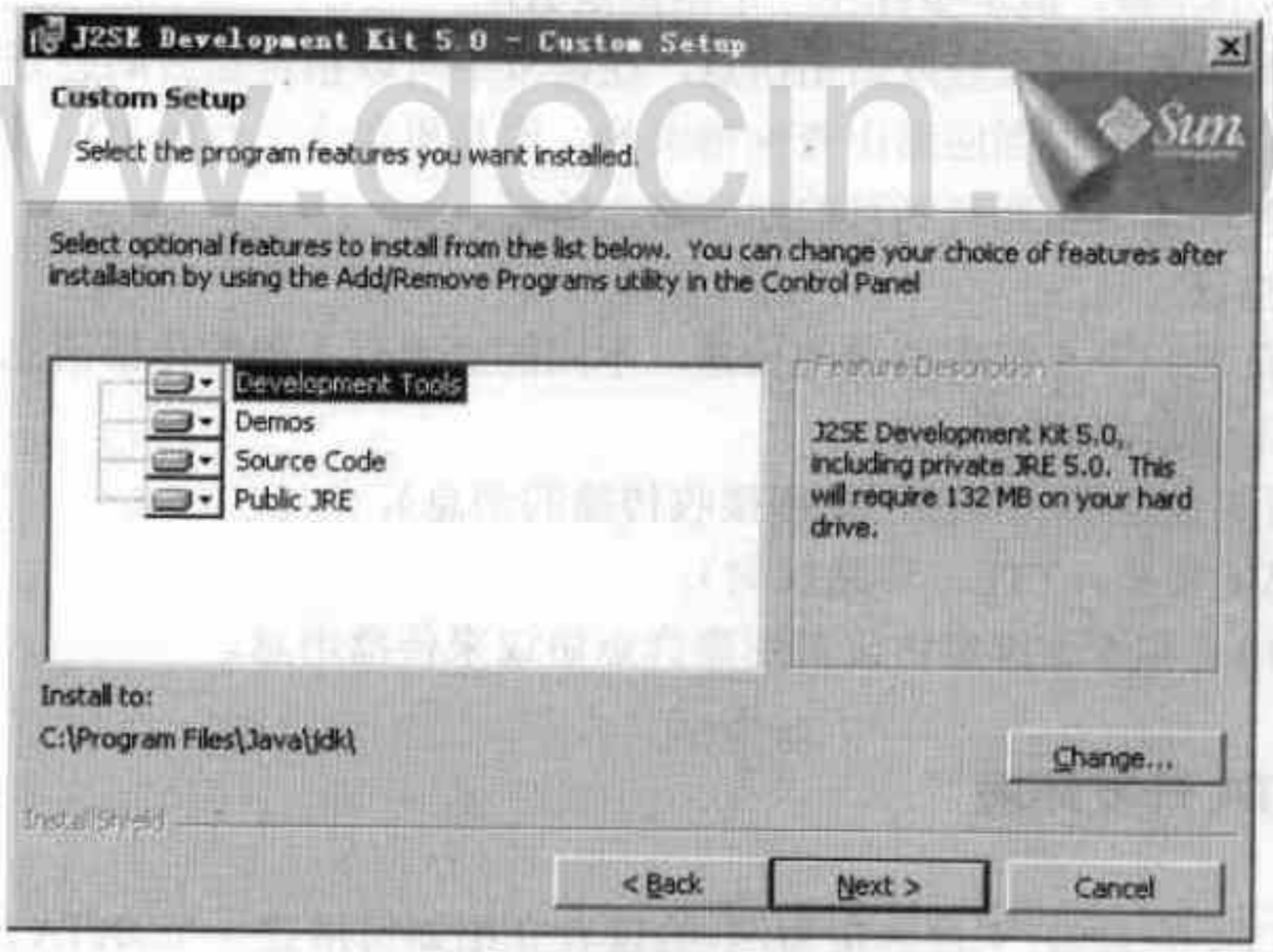


图 6-18　JDK 安装界面 2

选择所有的组件，然后指定 JDK 的安装目录，默认是 C:\Program Files\Java\jdk-1_5_0，修改为 C:\Program Files\Java\jdk，然后单击“Next”按钮，开始安装 JDK。JDK 安装完毕后，会弹出对话框让用户选择是否安装 JRE（Java 运行时环境），如图 6-19 所示。

www.docin.com
www.ckook.com

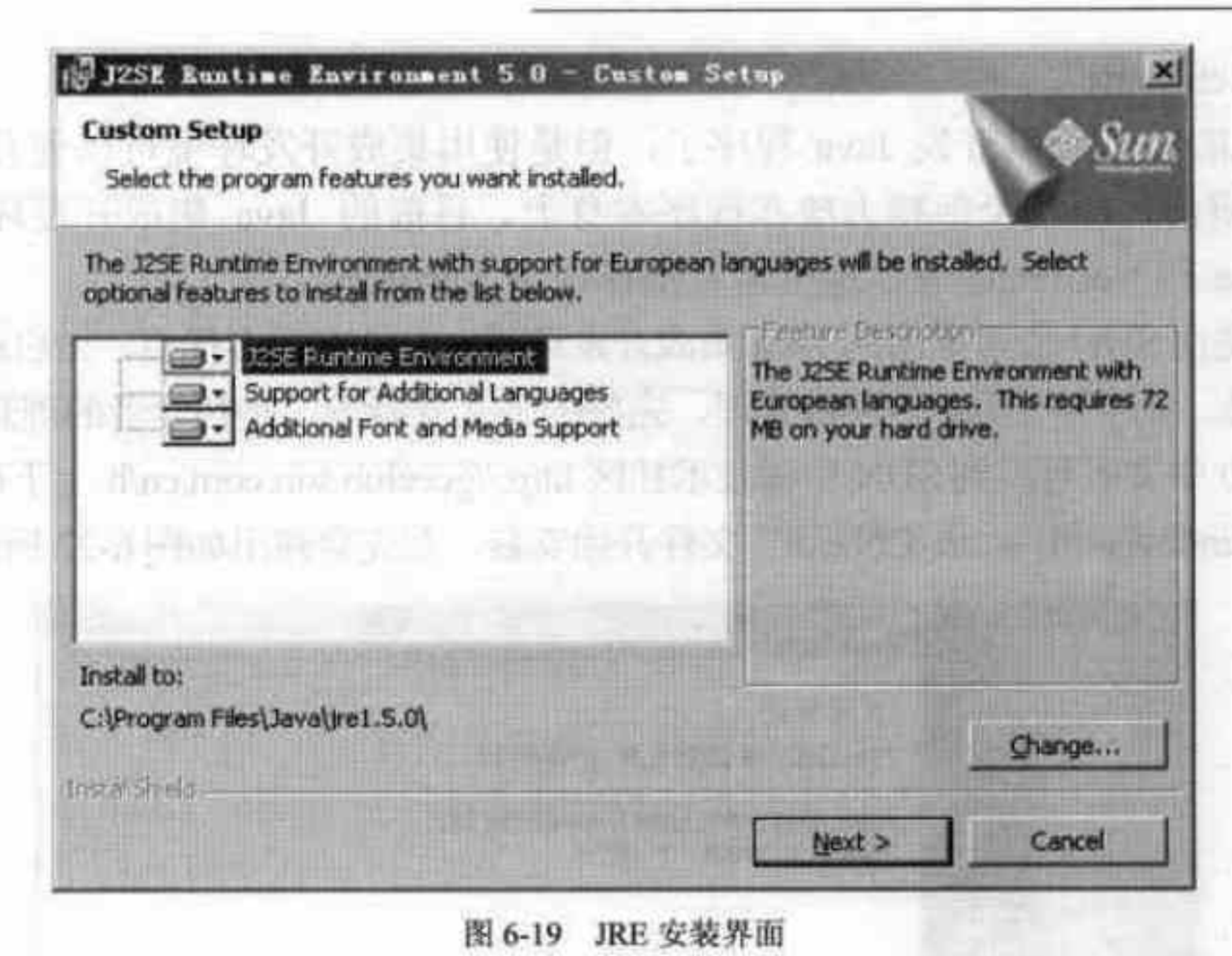


图 6-19 JRE 安装界面

把 JRE 的安装目录修改为 C:\Program Files\Java\jre，单击“Next”按钮完成所有的安装。

安装完 JDK 后，要在环境变量中加入 JDK 的安装路径，以方便程序的调用。在控制面板里双击“系统”图标，出现如图 6-20 所示的对话框。选择其中的“高级”标签，单击“环境变量”按钮，出现如图 6-21 所示的对话框。选择其中的 path 变量，单击“编辑”按钮，在 path 变量后面添加“C:\Program Files\Java\jdk\bin”，然后新建一个 classpath 变量，变量值设置为“C:\Program Files\Java\jre\lib”，然后单击“确定”按钮，JDK 的环境变量就设置好了。读者可以在“cmd”窗口里键入“Javac”命令来测试 JDK 是否能正常使用。

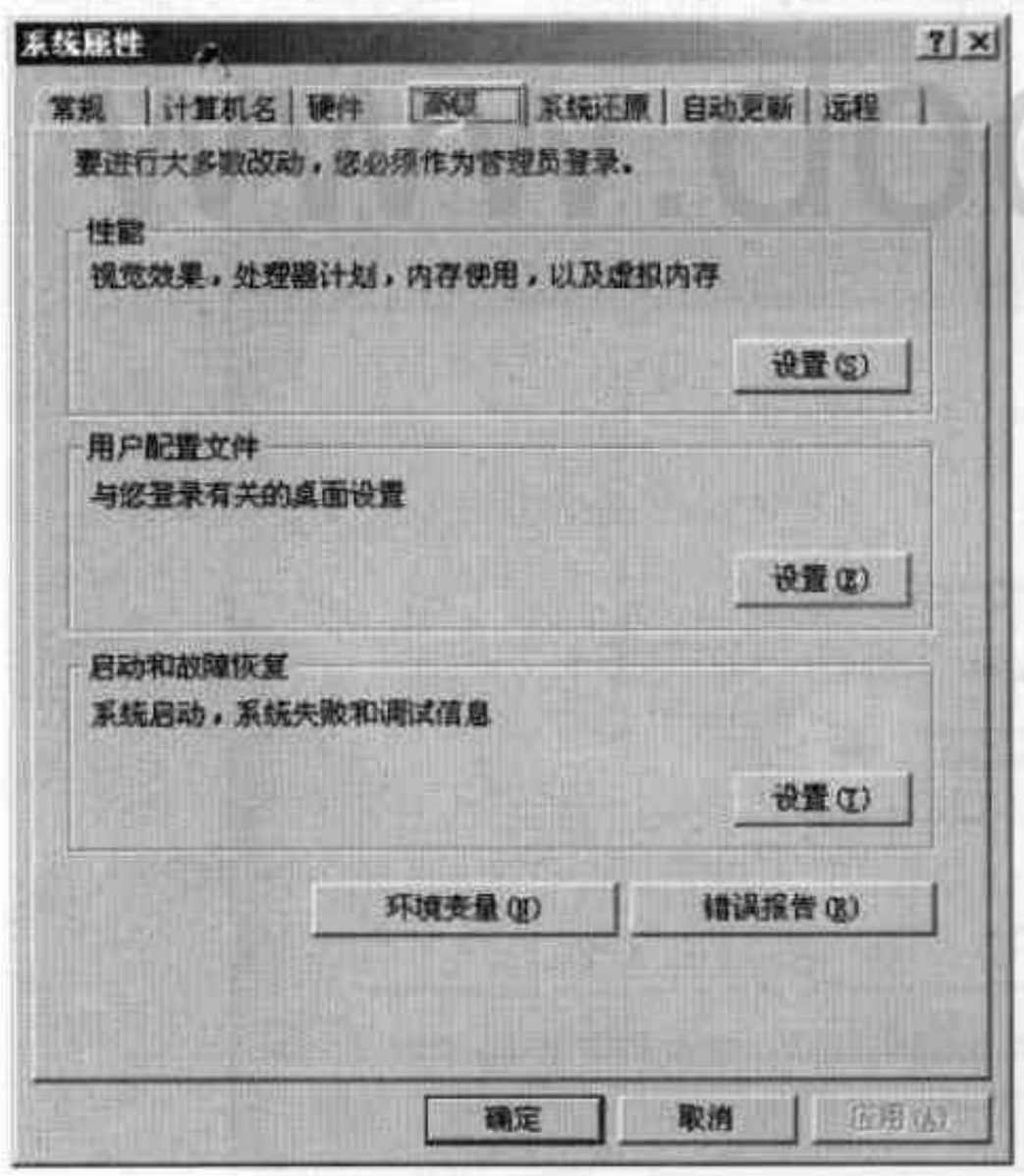


图 6-20 JDK 环境变量设置 1

图 6-21 JDK 环境变量设置 2

www.ckook.com

2．安装 NetBeans

虽然使用 JDK 就可以开发 Java 程序了，但是使用集成开发环境可以使开发者不用去关注一些琐碎的事情，而把全部精力放在程序本身上。目前的 Java 集成开发环境有很多，如 jbuilder、eclipse 和 NetBeans 等，这里笔者推荐使用 NetBeans。

NetBeans 是由 SUN 公司推出的 Java 集成开发环境，它具有很多优点，如启动速度快、丰富的代码补全功能、完全基于 ant 的项目管理、完整的中文支持等，是初学者的理想开发工具。

NetBeans5.0 中文版可以到 SUN 中国技术社区 http://gceclub.sun.com.cn/h 上下载，下载后双击“netbeans-5_0-windows-ml_ja_zh_CN.exe”文件开始安装。首先会弹出如图 6-22 所示的安装界面。

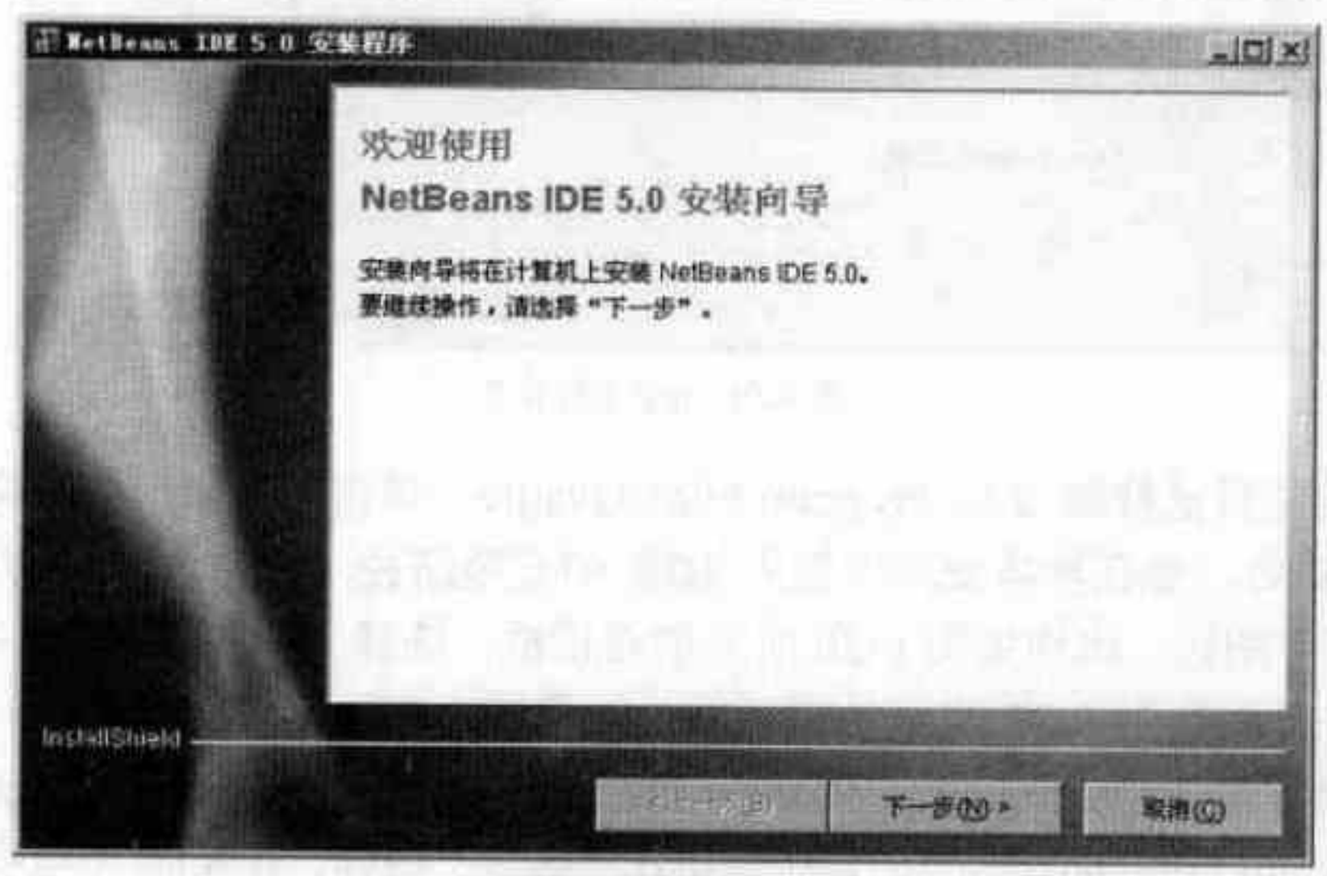


图 6-22　NetBeans 安装界面

单击“下一步”按钮开始安装，所有的选择都选择默认选项。安装完成后就可以运行 NetBeans，典型的运行界面如图 6-23 所示。

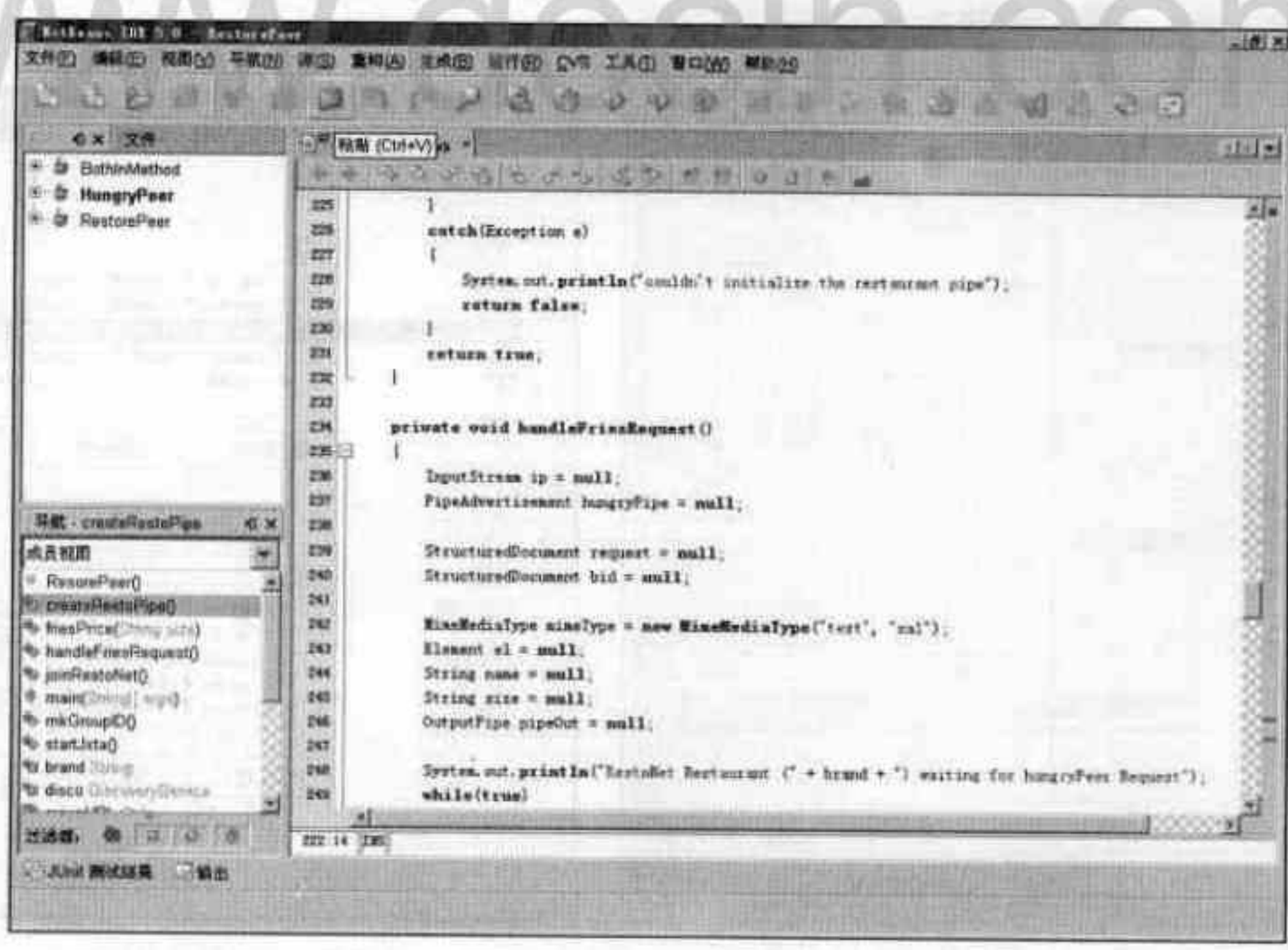


图 6-23　NetBeans 运行界面

www.ckook.com

3．安装 JXTA 开发库

JXTA 开源组织提供了 JXTA 的多个语言的实现版本，读者可以到 www.jxta.org 上下载对应库，这里应该选择下载 J2SE 平台的绑定，最新版本是 2.4.1b。把文件下载到本地后解压缩到 C 盘根目录，这时 C 盘目录下会出现 jxta-lib-2.4.1b 子目录，目录下就是开发 JXTA 程序所必需的库文件，后缀名为 jar。

下载库文件后，读者还应该下载对应的文档，以方便开发时查询库函数的功能。

6.3 本章总结

本章首先介绍了 WPPN 平台、JXTA 平台的基本概念和协议，在此基础上分别介绍了如何搭建 WPPN 和 JXTA 应用程序的开放环境，并给出了几个简单的例子和例子代码讲解。

6.4 练习题

（1）在网上追踪 WPPN 的最新资料。
（2）编写程序，熟悉 WPPN 的各种 API。
（3）下载 JXTA 的协议规范，查看 JXTA 的集合点之间是如何相互发现的。
（4）用 JxtaBiDiPipe 实现两个对等点之间通信的程序。

参 考 文 献

[1] SUN 公司．JXTA v2.3.x:Java™ Programmer's Guide．www.jxta.org，2005

[2] Oaks S, Travesrsat B, Gong L，著．技桥，译．JXTA 技术手册．北京：清华大学出版社，2004

[3] Brookshier D，等，著．常晓波，李静，译．Java P2P 程序设计．北京：中国电力出版社，2003

[4] Petzold C. Programming Windows．Microsoft Press, 1998

www.docin.com

www.ckook.com

CHAPTER 7

第 7 章　P2P 文件共享系统开发实例一——eMule 的设计与实现

eMule 是目前世界上最大的，并且最可靠的 P2P 文档共享的客户端软件，它遵循通用公共许可证（GNU，General Public License）开发，任何人都可以永久免费安装使用，并且可以在符合 GPL（General Public License，通用公共许可证）复制、散布与修改的条款与条件下发布 eMule DownCD 版。本章将针对 eMule 系统的运行原理、系统结构进行介绍，并且对部分模块的代码进行详细的分析。

在学习了本章后，希望读者能够掌握以下几点：

（1）了解该系统的原理；

（2）了解该系统的架构；

（3）掌握该系统的使用方法；

（4）了解该系统的优缺点。

7.1　eMule 系统概述

www.docin.com

eMule 系统作为一个 P2P 文件共享系统，其目的就是打破传统搜索引擎技术的壁垒，建立一个以 P2P 网络为基础的文件共享系统。本节将针对该系统的系统背景、系统设计目标等进行简要的阐述，使读者对 eMule 系统有一个初步的认识。

7.1.1　P2P 文件共享系统的背景

现在的社会是一个信息的社会，因特网技术的蓬勃发展，为信息的交流与共享提供了更为高速便捷的平台。然而，对于用户来说，并不是所有的信息都有价值，即使是有价值的信息也并不都是同等重要。面对如此纷繁的信息时，如何能够快速高效地找到有效信息变成了当今 IT 领域一个亟待解决的问题。有这样一句经典的话，“如果一条信息无法被人找到，那么它相当于并不存在。”由此可见，随着当今社会信息的高速增长，检索技术变得越来越重要。信息技术越是发达，就越是离不开先进的

www.ckook.com

检索技术。

在最初的因特网中，网民通过各个网站提供的链接寻找资源，自主权和选择权受到很大的制约。其后，出现了专门提供搜索服务的搜索引擎网站，如 Yahoo!、Lycos、Google、百度等，使得网民可以利用搜索引擎去查找获取自己所需要的信息。但是传统的搜索引擎技术在资源的搜索深度和信息的有效定位上存在很大的局限性。正是在这样的背景下，P2P 技术以一种全新的思路为文件共享提供解决方案，它将信息共享的主动权重新交还给了用户。一方面，用户可以将自己收藏的有价值的文件在网络上任意共享；另一方面，用户可以从网络上精确地查找自己所需要的资源并快速获取。以 eMule、BT 等为代表的 P2P 文件共享软件，利用 P2P 网络中丰富的资源，实现了共享文件的快速定位、网络备份以及高速下载，开创了文件共享的新时代。

7.1.2 eMule 的设计目标

在 eMule 的官方首页上写着：2002 年 05 月 13 日，一个叫做 Merkur 的人，他不满意原始 eDonkey2000 的客户端，并且坚信他能够做得更好，所以他开始制作。他聚集了其他开发人员在他的周围，eMule 工程就此诞生。他的目标是将 eDonkey 的优点及精华保留下来，并加入新的功能以及使图形界面变得更好。

eMule 与其他 P2P 软件相比，其客户端使用了多种途径搜索下载的资料源，通过 ED2K、来源交换、Kad 共同组成一个可靠的网络结构。其设计目标如表 7-1 所示。

表 7-1 eMule 的设计目标

设计目标	详细描述
激励机制	通过排队机制和积分上传系统，激励用户共享并上传自己的资源，以使自己更容易、更快速地下载自己想要的资源
文件下载智能管理	通过对每个下载文件的自动检查，确保文件的正确性；通过智慧损坏控制，快速修复文件的损坏部分
自动优先权及来源管理	允许用户一次下载许多个资源而无须监视它们
预览功能	允许用户在视频文件下载完成之前查看文件
Web 服务特性	通过自身的 Web 服务特性和 Web 服务器，允许用户快速地向网络存取资料
过程管理	允许用户在下载的过程中通过类别对文件进行组织和管理
全面的资源搜索	通过服务器搜索（本地和全球）、基于 Web 搜索和 Kad 网络，共同为用户提供资源搜索，还允许用户使用非常复杂的布林搜索使搜索更为灵活
好友管理	通过信息及好友系统，用户能传送信息到其他的客户端并可将他们加为好友，有好友上线的话，用户就能在自己的好友列表中看到
聊天功能	通过内建的 IRC 客户端，用户能和全世界其他的共享者聊天

eMule 是完全免费的软件，官方版 eMule 也完全没有任何的广告软件，其宣扬的开发精神是为了乐趣及知识，而不是为了金钱。

eMule 遵循通用公共许可证开发。该协议的主要原则包括：

（1）确保软件自始至终都以开放源代码形式发布，保护开发成果不被窃取用作商业发售。任何一套软件，只要其中使用了受 GPL 协议保护的第三方软件的源程序，并向非开发人员发布时，软件本身也就自动成为受 GPL 保护并且约束的实体。也就是说，此时它必须开放源代码。

（2）GPL 大致就是一个左侧版权（Copyleft，或译为“反版权”、“版权属左”、“版权所无”、“版责”等）的体现。你可以去掉所有原作的版权信息，只要你保持开源，并且随源代

www.ckook.com

码、二进制版附上 GPL 的许可证就行，让后人可以很明确地得知此软件的授权信息。GPL 精髓就是，使软件在完整开源的情况下，尽可能地让使用者得到自由发挥的空间，使软件得到更快更好的发展。

（3）无论软件以何种形式发布，都必须同时附上源代码。例如在 Web 上提供下载，就必须在二进制版本（如果有的话）下载的同一个页面，清楚地提供源代码下载的链接。如果以光盘形式发布，就必须同时附上源文件的光盘。

（4）开发或维护遵循 GPL 协议开发的软件的公司或个人，可以对使用者收取一定的服务费用。但还是一句老话——必须无偿提供软件的完整源代码，不得将源代码与服务做捆绑或任何变相捆绑销售。

7.2 eMule 系统原理

eMule 系统作为一个 P2P 的文件共享系统，主要支持两类网络协议：eMule 和 Kad。本节将针对这两类网络协议的原理进行详细的描述，使读者对 eMule 系统的构建基础有一个全面的了解。

7.2.1 eMule 协议原理

eMule 是一个基于 eDonkey 协议的流行的文件共享软件。eMule 网络由几百个 eMule 服务器和上百万个 eMule 客户端共同组成。eMule 客户端通过连接 eMule 服务器获得网络服务，该连接在运行过程中一直打开。eMule 服务器执行集中式的索引服务，不和其他服务器进行通信。

eMule 服务器使用一个内部数据库，存储客户端和文件的信息。eMule 服务器并不存储实际的文件，仅仅提供文件位置信息的集中索引。此外，eMule 服务器还提供穿越防火墙的客户端和无法接收输入连接的客户端之间的桥梁作用。eMule 协议通过使用 UDP 来削弱 eMule 客户端对 eMule 服务器和其他 eMule 客户端的依赖性。eMule 的系统示意如图 7-1 所示。

www.dzsh.com

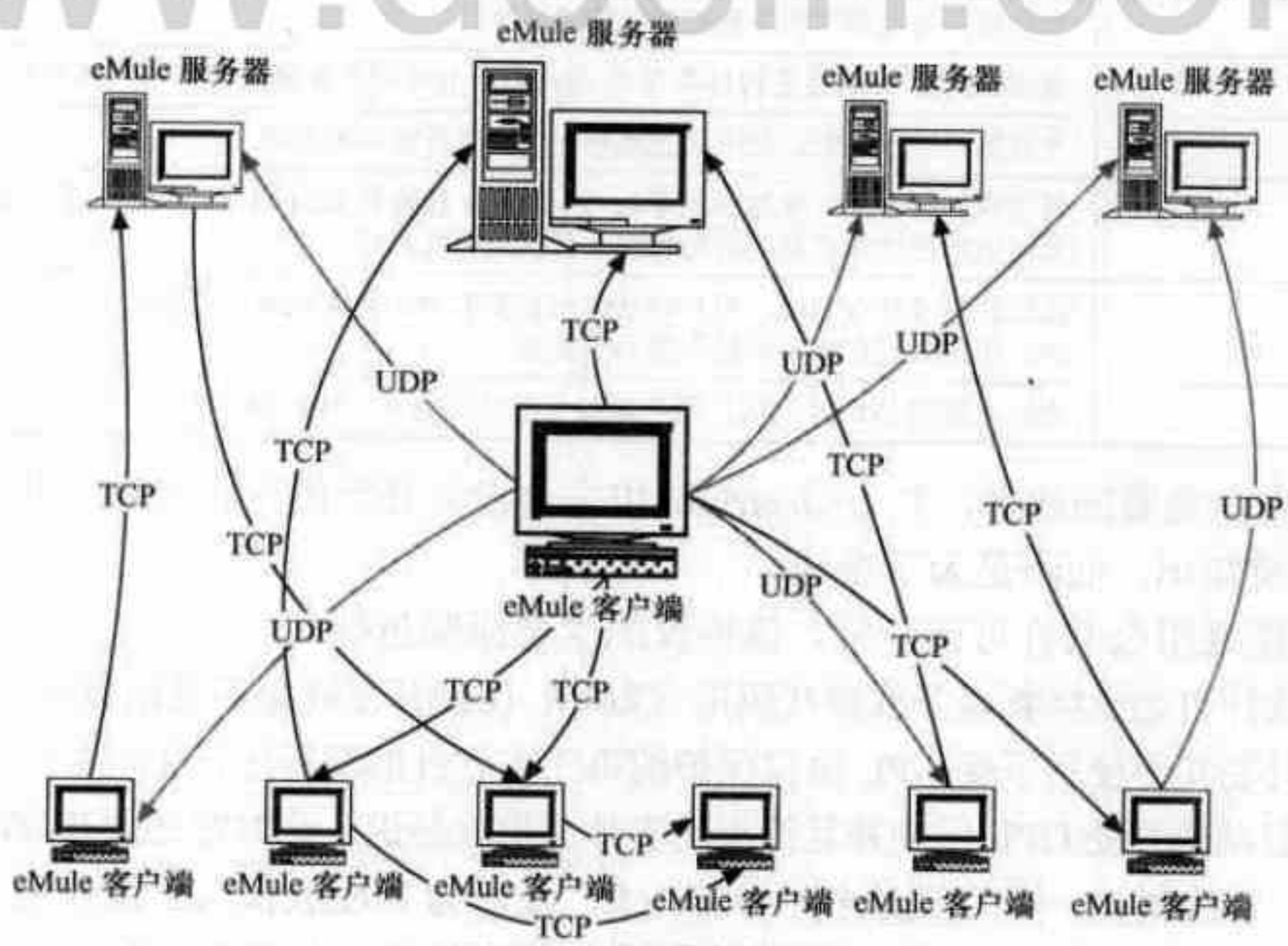


图 7-1 eMule 系统示意图

www.ckook.com

在 eMule 协议中定义了客户端 ID、用户 ID 和文件 ID 3 个名词，其意义如表 7-2 所示。

表 7-2 eMule 中的名词解释

名词定义	名词解释
客户端 ID	客户端 ID 是在 eMule 客户端与 eMule 服务器建立握手的时候由服务器分配的 4Byte 的标识，它仅在该客户端和服务器的 TCP 连接的生命周期内有效。客户端 ID 有低 ID 和高 ID 之分。通常情况下，eMule 服务器给那些不能接受输入连接的客户端分配低 ID。低 ID 限制了客户端对 eMule 网络的使用。高 ID 对客户端使用 eMule 网络没有限制
用户 ID	用户 ID 是一个 128bit 的 GUID，由串联的随机数组成，其中第 6Byte 和第 15Byte 不是随机产生的，它们的值是 14 和 111。用户 ID 是唯一的，并且在 eMule 客户端与 eMule 服务器会话过程中用来表示客户端。用户 ID 在信誉系统中扮演着重要的角色
文件 ID	文件 ID 用来唯一地标识网络上的文件。此外，在文件损坏检查和恢复时也会用到文件 ID。文件 ID 的主要目的是为了唯一地标识和索引文件，它是通过散列文件的内容来计算。文件 ID 主要有两种：第一种用于产生唯一的文件 ID，使用 MD4 算法来基于文件的数据计算；第二种用于损坏检查和恢复，使用 SHA1 算法来为每个部分计算

当 eMule 服务器不能打开 eMule 客户端端口的 TCP 连接时，该客户端将得到一个低 ID。这通常是由于该客户端安装了防火墙，拒绝输入连接。此外，当 eMule 客户端通过 NAT 或者代理服务器连接时，或者当 eMule 服务器忙时，eMule 客户端也可能会被分配一个低 ID。eMule 客户端的高 ID 是基于客户端的 IP 地址，按照以下方式计算出来的：

假定主机 IP 是 X.Y.Z.W，ID 将是 $X+2^8\times Y+2^{16}\times Z+2^{24}\times W$。低 ID 始终小于 16777216（0x1000000）。

1．客户端与服务器的 TCP 通信

eMule 客户端在启动的时候使用 TCP 连接向某个 eMule 服务器注册，服务器给客户端分配一个 ID，用于标识在其后会话过程中的该客户端。客户端不能同时连接几个服务器并且不能在没有用户的干预下动态改变服务器。eMule 客户端向服务器注册的流程分为高 ID 注册、低 ID 注册、注册拒绝 3 种情况，分别如图 7-2、图 7-3 和图 7-4 所示。

www.docin.com

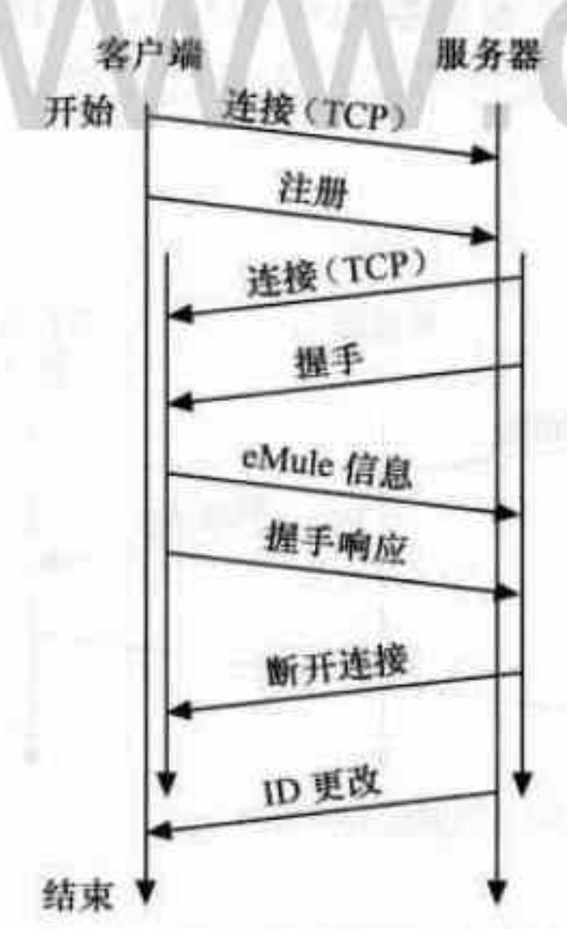


图 7-2 高 ID 客户端注册序列图

图 7-3 低 ID 客户端注册序列图

在注册成功后，客户端发送其共享的数据给服务器，服务器将这些信息存储到内部数据库中。客户端同时还将其希望下载的文件列表发送给服务器。服务器以一系列消息来响应，

www.ckook.com

每一个消息对应下载列表中的每个文件，直到所有的源列表都被下载到客户端为止。其流程如图 7-5 所示。

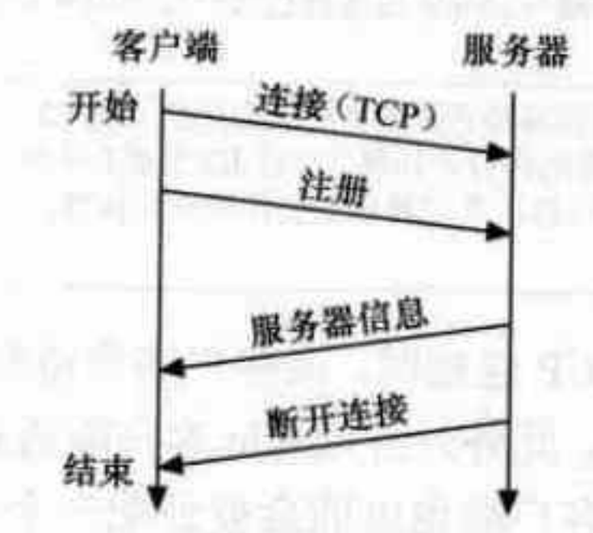


图 7-4　注册拒绝序列图

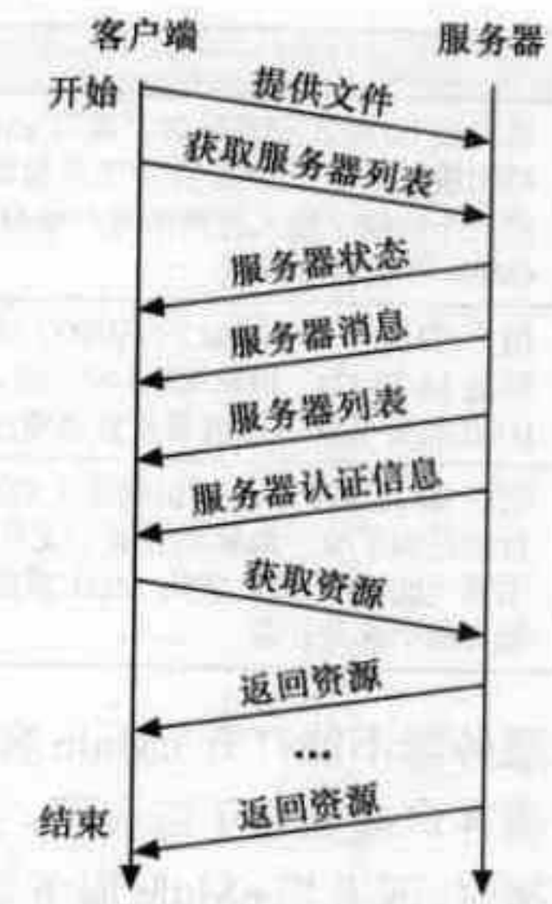


图 7-5　启动信息交互序列图

当客户端需要搜索文件时，它将一个搜索请求发送到服务器，服务器以相应的搜索结果响应。客户端选择下载一个或多个文件，向服务器请求选中的文件的源，服务器返回每个请求文件的源的列表。服务器的状态信息可能在服务器找到文件的源之前发送给客户端，该信息包括了当前服务器的用户数和服务器支持的文件信息。客户端搜索文件的流程如图 7-6 所示。

eMule 还设计了回调机制，用于克服低 ID 客户端由于不能接收输入连接请求而无法共享文件的缺陷。该机制的流程是：假定客户端 A 和 B 连接到同一个服务器，B 是低 ID，无法接收输入连接请求，A 向服务器发送一个回调请求，通过服务器已有的与 B 的 TCP 连接向 B 提供 A 的 IP 地址和端口，通知 B 主动连接 A，将它的文件不经过服务器发送给 A。回调机制的流程如图 7-7 所示。

www.ccun.com

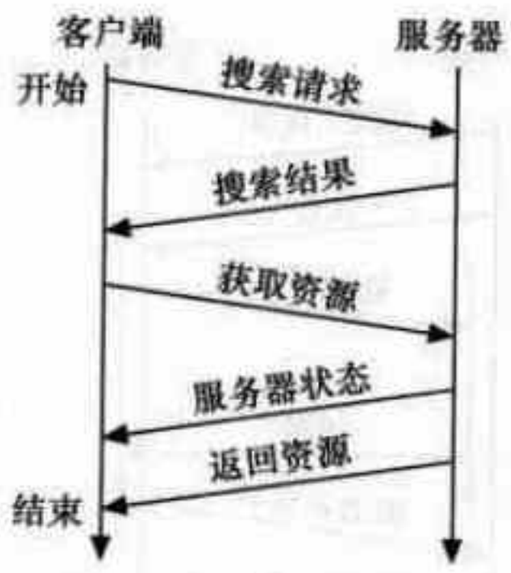


图 7-6　文件搜索序列图

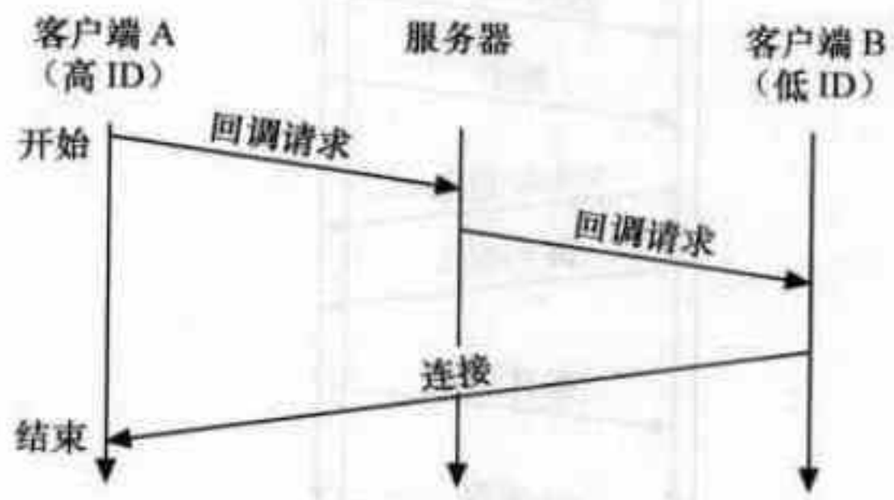


图 7-7　回调机制序列图

2．客户端与服务器的 UDP 通信

eMule 客户端通过 UDP 与除已经连接的服务器之外的服务器进行通信，其目的是为了增强文件查找、增强源搜索和保持服务器列表有效。UDP 包通过定时器每 100ms 出发一次进行发送。

www.ckook.com

eMule 客户端通过 UDP 服务器状态请求和 UDP 服务器描述请求两种消息完成对服务器列表状态的定期验证。当检查一个 eMule 服务器的状态时，eMule 客户端首先向服务器发送一个状态请求消息，然后是在每两个状态请求消息之后发送一个服务器描述请求消息，如图 7-8 所示。eMule 客户端发送的状态请求消息中包含一个随机数，服务器在响应该消息时需要将随机数返回。如果随机数与请求消息中的不一致，响应消息将被丢弃。客户端每次发送状态请求消息时将自加一个“尝试计数器”，收到服务器的响应消息时将重置该计数器。当计数器达到设定值时，客户端将认为该服务器无法提供服务，将它从服务器列表中删除。服务器的响应消息中，包括了服务器的当前用户数、服务器上的文件数，以及服务器的软件、硬件限制。该流程如图 7-9 所示。

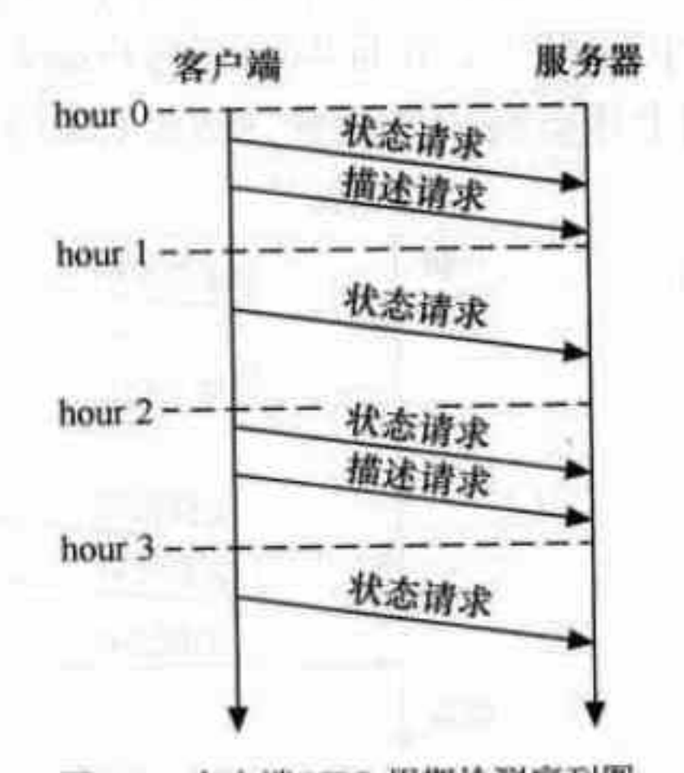


图 7-8 客户端 UDP 周期检测序列图

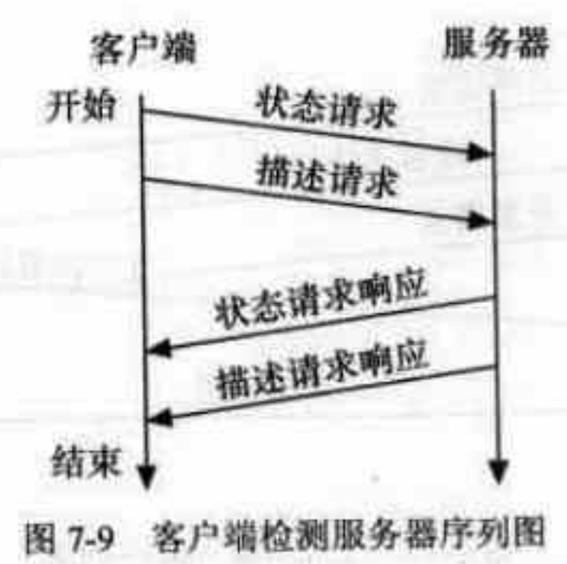


图 7-9 客户端检测服务器序列图

eMule 客户端可能通过配置使用 UDP 来增强文件搜索功能。UDP 的搜索请求格式和 TCP 几乎相同。服务器仅在有搜索结果时响应请求。

当 eMule 客户端的下载队列中的某个文件的源的数量低于设定的限值时，客户端定期向服务器列表中的服务器发送 UDP 包以获得该文件的更多的源。由于这类 UDP 包有可能每秒都会发送，因此它在 eMule 客户端发送的 UDP 数据包中占据着最可观的比例。

3．客户端与客户端的 TCP 通信

eMule 为每个<客户端，文件>提供一个专一的 TCP 连接，以支持 eMule 客户端的文件下载。为了保证合理的下载速率，在客户端能够提供最小允许速率（这是一个硬编码，当前设置为 2.4kbit/s）之前，eMule 不允许客户端下载文件。

当 eMule 客户端间建立 TCP 通信时，首先是进行一个初始化的对称握手过程，双方交互彼此的信息，包括标识、版本和容量等。初始化握手过程如图 7-10 所示。

当连接建立后，eMule 客户端立即发送若干请求消息，其中包括了它所希望下载的文件。其基本信息交换由 4 类消息组成：文件请求消息、请求文件 ID 消息、文件请求响应消息、文件状态消息。在 eMule 的扩展协议中，还添加了两条消息：资源请求消息和资源响应消息。该扩展协议主要是为了将源端的资源传递给下载端（若源端也正处于下载文件的过程中）。该流程如图 7-11 所示。

如果被请求客户端没有被请求的文件，被请求客户端将在收到请求文件 ID 消息之后立即向请求客户端发送没有找到文件的响应消息。该流程如图 7-12 所示。

www.ckook.com

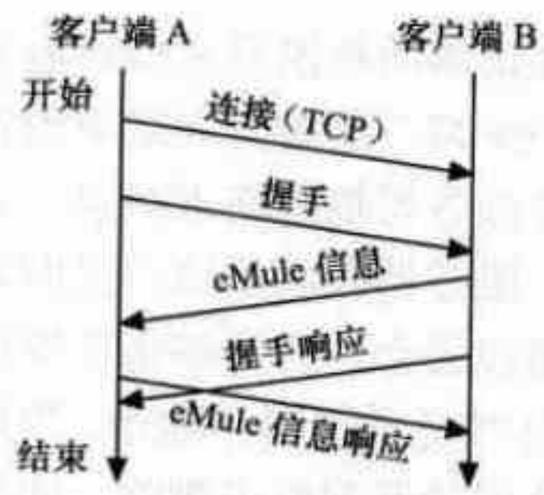


图 7-10　客户端初始化握手序列图

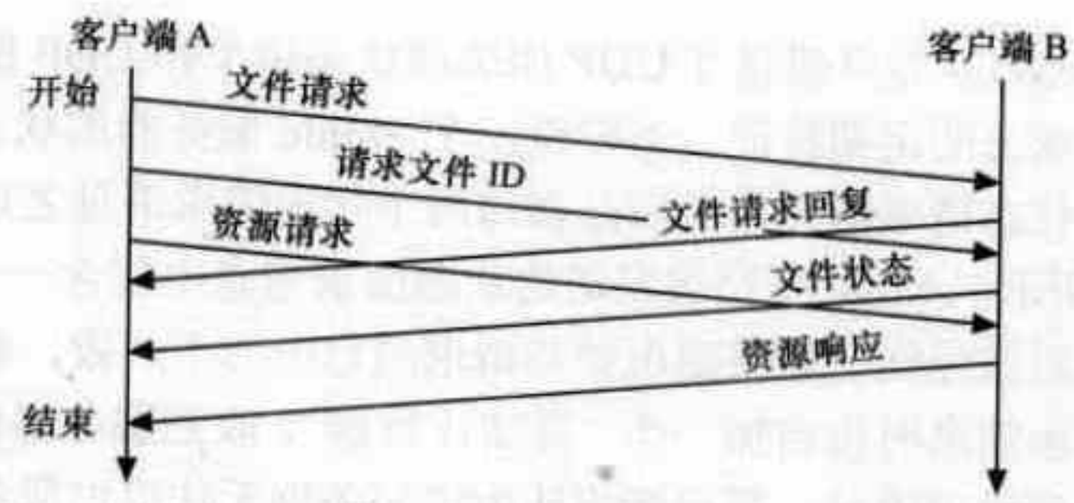


图 7-11　文件请求序列图

如果被请求客户端有被请求的文件，但是当前该客户端正在给其他客户端传送该文件，并且它的上传队列不为空，即很有可能还有其他请求客户端正在等待下载该文件，在完成初始化握手后，被请求客户端将请求客户端加入到它的上传队列中，并且向请求客户端发送队列等级消息，该消息告诉请求客户端它在被请求客户端的上传队列中的位置。该流程如图 7-13 所示。

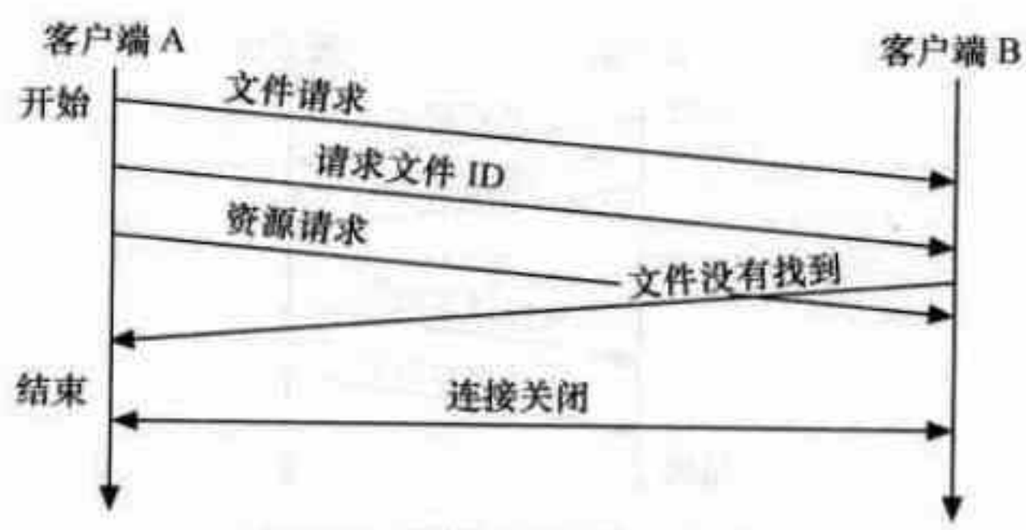


图 7-12　文件请求失败序列图

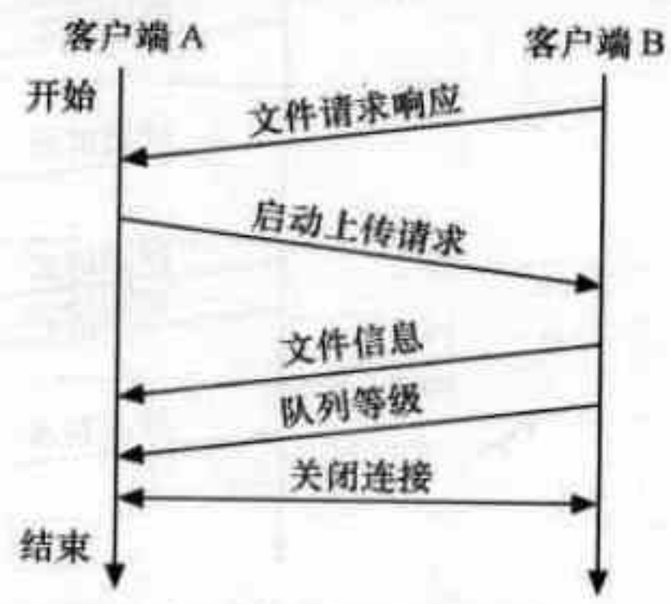


图 7-13　文件请求排队序列图

eMule 客户端为每个上传的文件维护一个上传优先级队列，在队列中的每个客户端的优先级按照客户端在队列中的时间和优先级修饰符号来确定。排在队列顶部的客户端可以下载文件。一个客户端在以下情况发生的时候停止下载文件：

（1）上传的客户端主动中止文件上传；

（2）下载的客户端已经下载完成其所需文件的所有部分；

（3）另一个客户端的优先级高于当前的客户端。

当请求客户端排到被请求客户端的队列的顶端时，被请求客户端将主动连接请求客户端，完成初始化握手过程，然后向请求客户端发送接受上传请求消息。这时候，请求客户端可以发送请求部分消息选择继续下载文件，或者发送取消传输消息取消下载文件。该流程如图 7-14 所示。

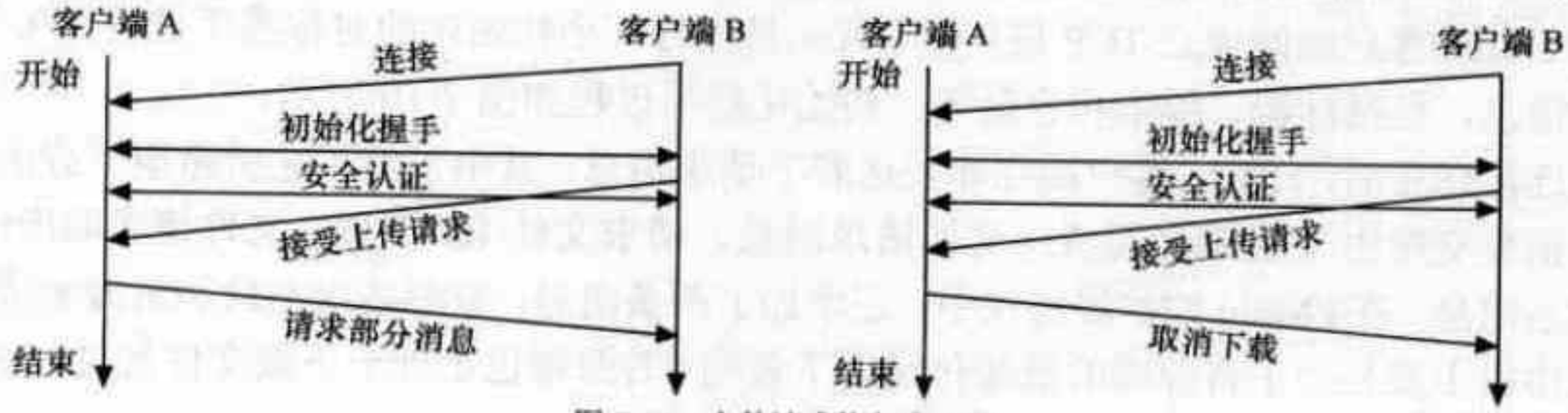


图 7-14　文件请求恢复序列图

发送和接收文件片是 eMule 网络活动的最主要部分。一个文件片的大小可以在 5 000～15 000Byte 之间变化。为了避免出现分片的情况，一个文件片消息被分成若干块发送，每一

www.ckook.com

块都是一个独立的 TCP 包。该包的最大长度是 1 300Byte。其中，控制消息在一个单独的 TCP 包内发送，有时也和其他消息一起发送，数据消息被分割成若干个 TCP 包。第一个 TCP 包中包含了发送文件片的消息头，其他 TCP 包都是该文件片的数据。当文件片不是 TCP 包大小的整倍数时，多余的数据将在第一个包中一起发送。该流程如图 7-15 所示。

在文件请求响应之后，传输流程正式开始。请求客户端发送启动上传请求消息，被请求端发送接受上传请求消息进行响应。之后，请求客户端开始请求文件片，被请求端发送对应的文件片。注意，一次请求最多可以请求 3 个文件片。该流程如图 7-16 所示。

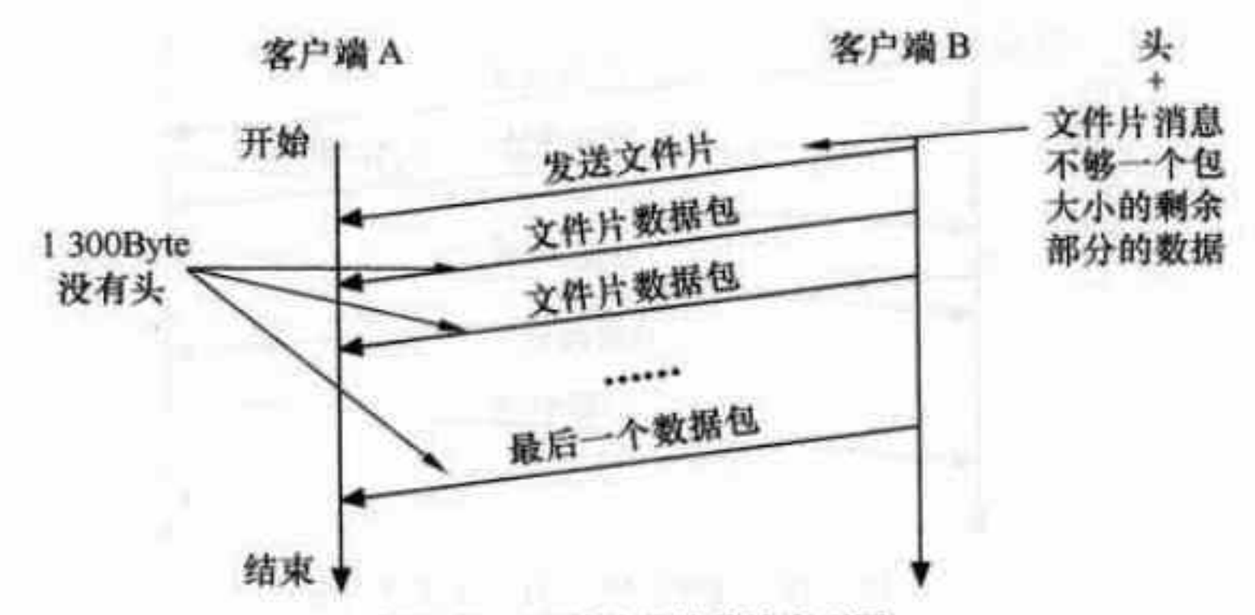


图 7-15 文件片分块传输序列图

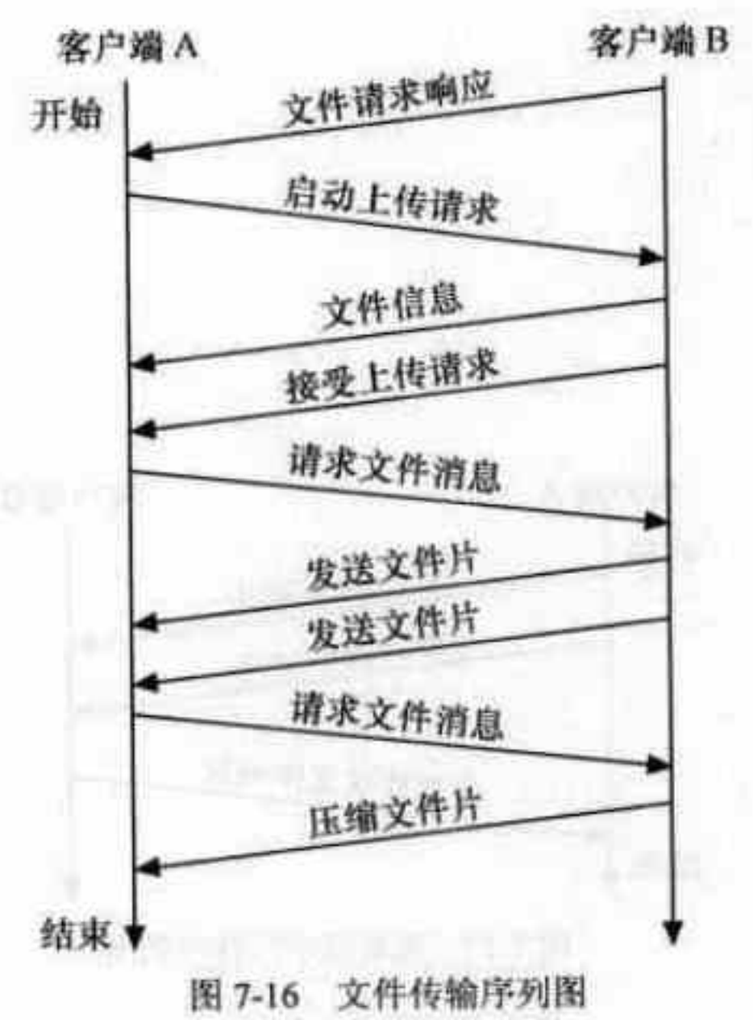


图 7-16 文件传输序列图

eMule 有选择性地选择文件片的下载顺序，目的是使整个网络的吞吐量和共享性最优化。文件片的下载顺序是由请求文件片的客户端决定的。请求客户端可以在任何时间从任意源上下载一个文件片，并且从同一个源上下载的数据块存储在一个位置。下载文件片时对下载片等级有以下几个原则。

（1）数据块的可用频率，目的是使少的数据块尽量快地下载，以使下载客户端成为新的源；

（2）预览文件片，用于预览和检查文件；

（3）请求状态，尝试向每个源询问其他的数据块，该询问在所有源之间传播；

（4）完成，在开始另一个数据块下载前应该局部地完成本数据块的获取。

频率的定义标准有 3 个域：非常少、少和普通。对于各个域，都有标准定义的权重，用于计算文件片的等级值，等级值低的文件片首先下载。等级值变化如表 7-3 所示。

表 7-3　文件片等级值范围

等级制范围	定　义
0～9 999	未被请求和被请求的非常少的文件片
10 000～19 999	未被请求的少的文件片和预览文件片
20 000～29 999	未被请求的最接近完成的普通文件片
30 000～39 999	被请求的少的文件片和预览文件片
40 000～49 999	被请求的未完成的普通文件片

www.ckook.com

浏览客户端的共享文件及文件夹有两类处理流程。第一类流程是通过发送查看共享文件消息实现，发生在初始化握手之后。该消息由查看共享文件回复消息来响应。该流程如图 7-17 所示。第二类流程是通过发送查看文件夹列表消息开始，该消息由共享文件夹列表消息来响应，然后对于响应消息中的每个文件夹发送一个查看共享文件夹内容的消息，该消息由内容列表消息来响应。该流程如图 7-18 所示。如果客户端被配置为阻止文件或文件夹共享请求，它将回复一个请求共享拒绝消息。该流程如图 7-19 所示。

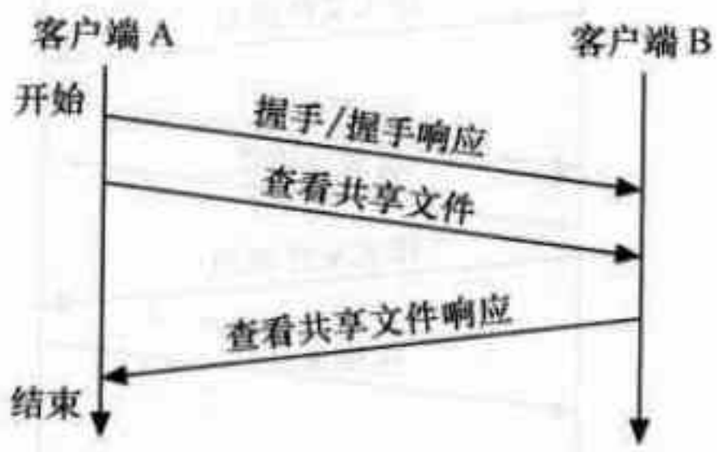


图 7-17　查看共享文件序列图

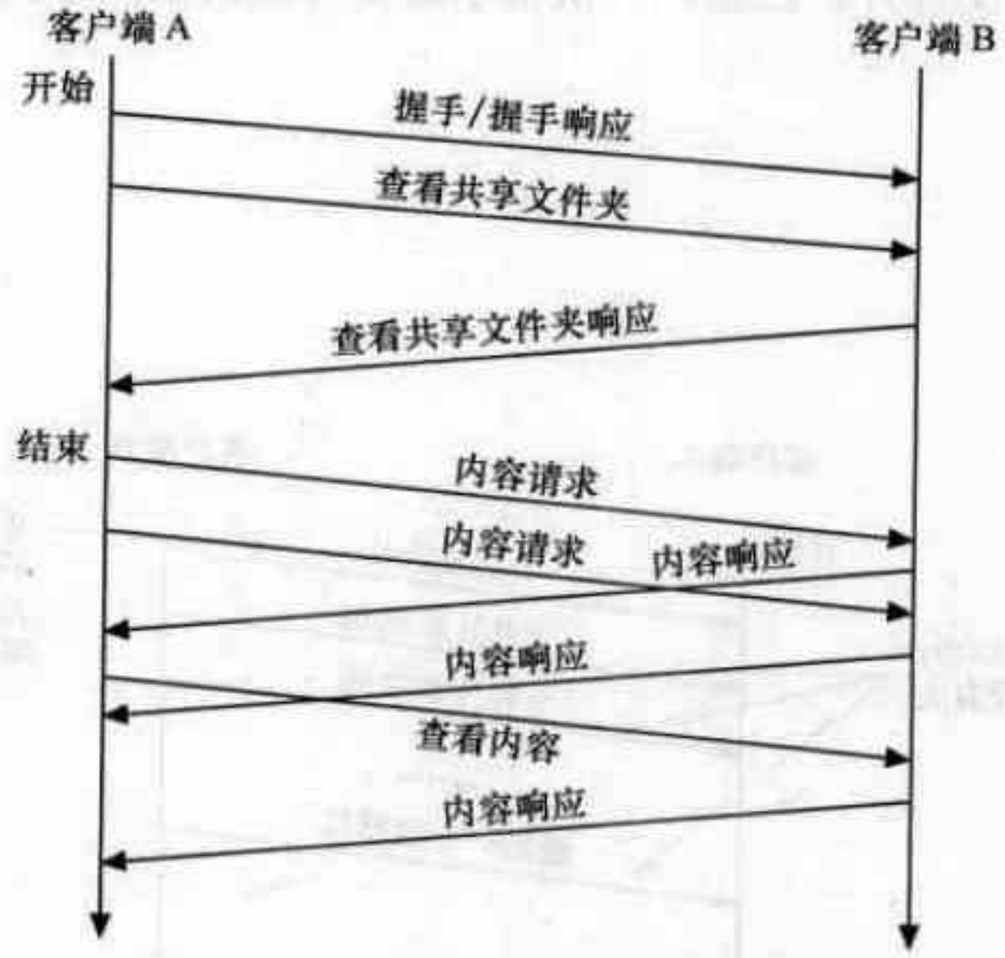
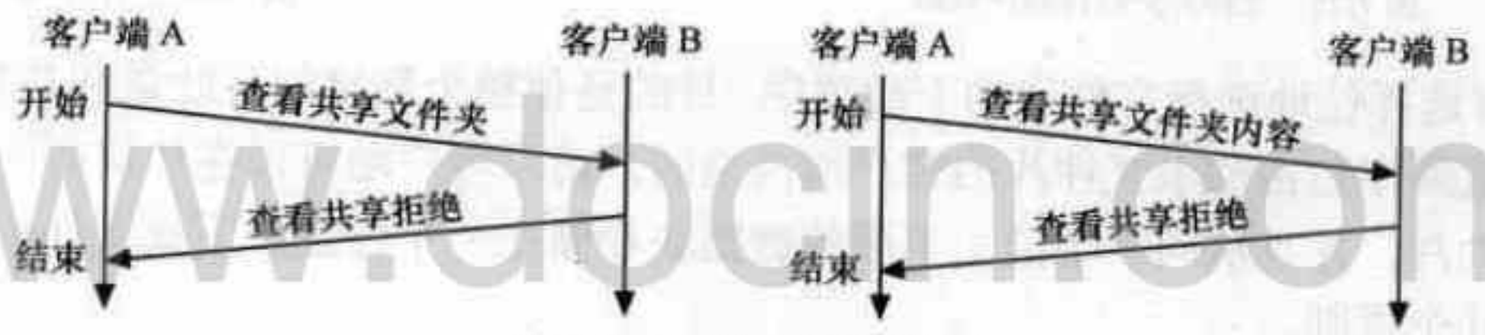


图 7-18　查看共享文件/文件夹序列图



www.docin.com

图 7-19　查看共享拒绝序列图

eMule 客户端通过发送散列设置请求消息，能够得到文件片的散列值，这个请求通过散列设置回复消息来响应，该消息中包含了文件中每个文件片的散列设置，该流程如图 7-20 所示。

eMule 客户端可以通过向其他客户端请求以预览一个共享文件。该预览功能在程序上是独立的，并且根据文件的类型不同而变化。该流程包括两种消息，预览请求消息和预览回复消息，如图 7-21 所示。

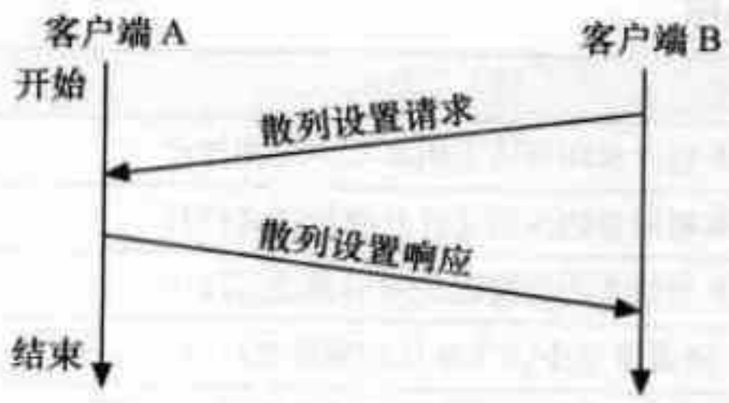


图 7-20　散列设置请求序列图

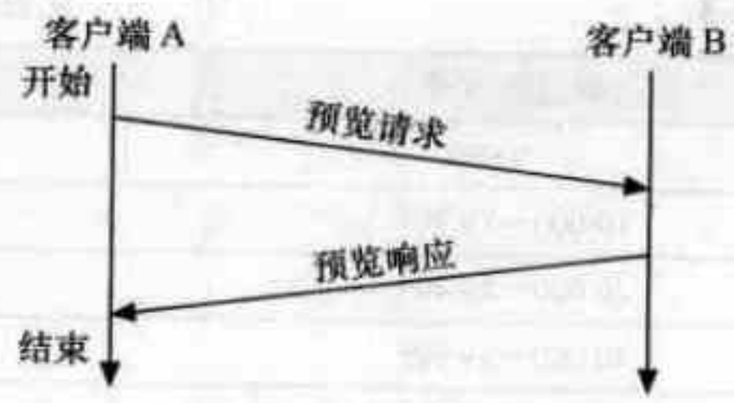


图 7-21　文件预览序列图

www.ckook.com

4．客户端与客户端的 UDP 通信

eMule 客户端定期通过 UDP 数据包，向它的下载队列中的客户端发送重新询问文件消息，用以询问自己在其队列中的位置。该消息在客户端将其他客户端加入到自己的下载队列后每间隔 20min 发送一次。该消息存在以下 3 种可能的回复。

（1）队列等级，表明请求客户端在发送端的队列中的位置；

（2）队列已满，表明发送端的队列已满；

（3）文件未找到，表明请求客户端请求的文件并不在发送端的列表中。

该流程如图 7-22 所示。

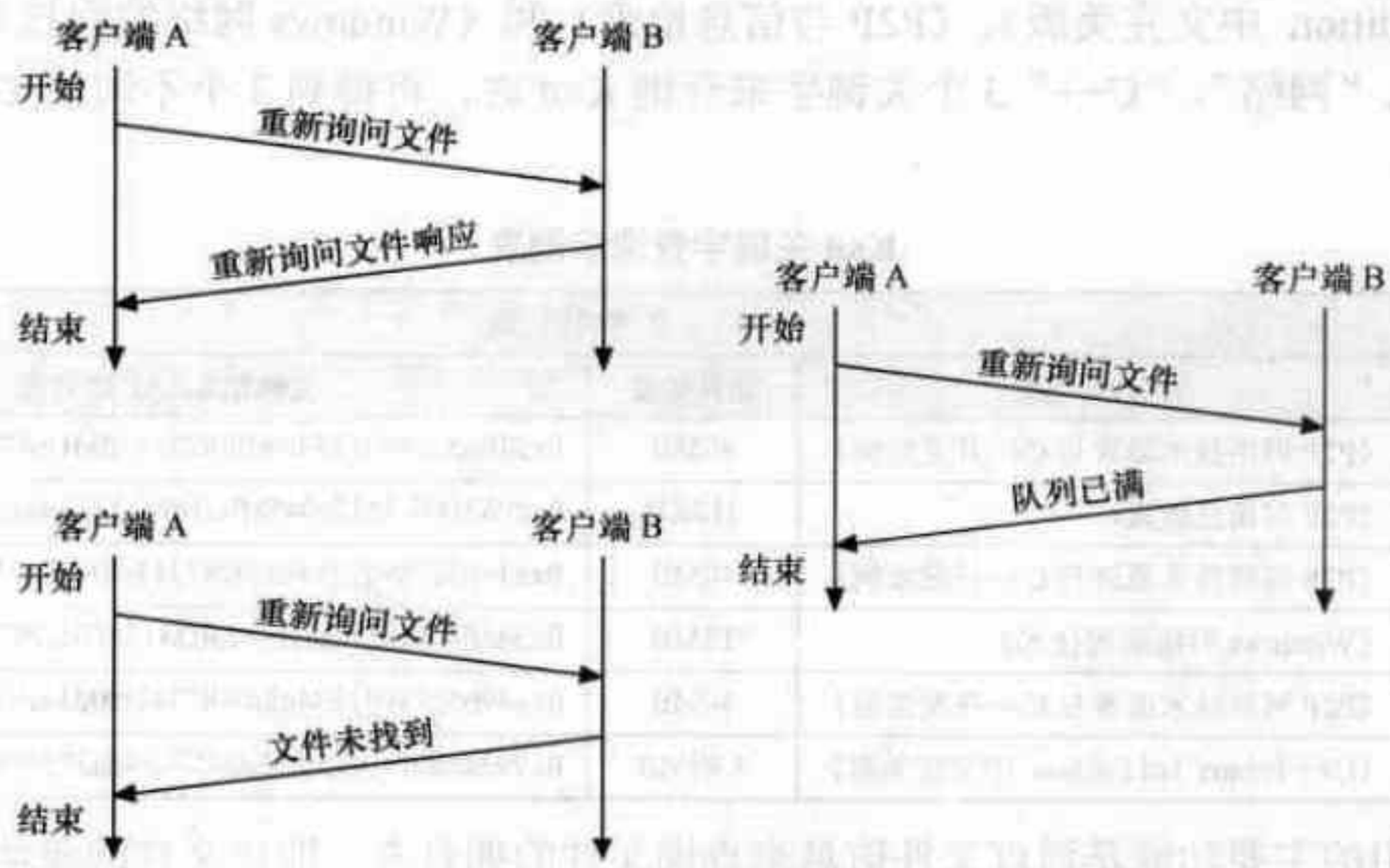


图 7-22 重新询问文件序列图

7.2.2 Kad 协议原理

www.docin.com

Kad 是 Kademlia 的简称，eMule 的官方网站在 2004 年 2 月 27 日正式发布的 eMule v0.42b 中，Kad 开始正式内嵌成为 eMule 的一个功能模块，可以说从这个版本开始 eMule 便开始支持 Kad 网络了。

1．Kad 网络概述

Kad 是一个完全分布式的网络，它完全不需要服务器，仅通过自身的 DHT 算法进行路由，与 eMule 协议有着本质的区别。但 Kad 实现的功能却和 eMule 协议是相同的，即搜索和寻找文件的源。它的具体的路由算法在 3.3.6 节中有详细的介绍。在使用 eMule 的时候打开 Kad，会有两个明显的特点：下载速度会加快和下载文件的源会增加。这对于 eMule 中的低 ID 用户和下载源在国外的文件的用户效果特别明显，在 eMule 网络中找不到的源在 Kad 网络中一般都能找到。

Kad 网络的主要目标是实现无服务器和改善可测量性。传统的 eMule 协议每个服务器只能处理一定数量的客户端，并且当连接服务器的客户端过多时，会严重地影响网络。而 Kad 具有自组织的特性，并且能够通过自我调节使使用者数量以及他们的连接效果达到最佳。因此，Kad 具有更好的网络特性。这种无服务器模式的 P2P 网络，被认为更加贴近 P2P 的定义，

www.ckook.com

是 P2P 网络发展的一种趋势。

2. Kad 的数据字典

一个 Kad 网络通过字典条目的形式，可以同时存储多张分布式散列表。在 eMule 中，Kad 网络主要存储并维护了两类分布式散列表：关键字字典和文件索引字典。

关键字字典的主要功能是通过给出的关键字查询其对应的文件名称及相关文件信息。其中，Key 是通过 SHA1 算法计算的关键字字符串的 160bit 散列值，而其对应的 Value 是一个列表，包含了所有拥有该关键字的文件的信息，如文件名、文件长度、文件的 SHA1 散列值等。譬如，假定目前网络中存在 4 个文件：《P2P 网络技术原理与 C++开发实例》、《C++ Primer 3rd Edition 中文完美版》、《P2P 与信息检索》和《Windows 网络编程技术》。客户端通过“P2P”、“网络”、“C++”3 个关键字来查询 Kad 时，将得到 3 个不同的文件列表，如表 7-4 所示。

表 7-4　　**Kad 关键字查询示例表**

查询关键字	文件列表		
	文件名	文件长度	文件的 SHA1 散列值
P2P	《P2P 网络技术原理与 C++开发实例》	40MB	0xa49b5c78901234e8f0987343ef0d1ea678902637
	《P2P 与信息检索》	312KB	0xc983f45b7d37e0458f0a3096e1f98c0de64593a0
网络	《P2P 网络技术原理与 C++开发实例》	40MB	0xa49b5c78901234e8f0987343ef0d1ea678902637
	《Windows 网络编程技术》	13MB	0x34ef930dc238ae3f9043073134f5de793a4b3478
C++	《P2P 网络技术原理与 C++开发实例》	40MB	0xa49b5c78901234e8f0987343ef0d1ea678902637
	《C++ Primer 3rd Edition 中文完美版》	4.49MB	0x793e8d0f914729f98ea457c14da39238fb3dee04

文件索引的主要功能是通过文件信息来查询文件的拥有者，即该文件的源地址。其中，Key 是通过 SHA1 算法计算的所需文件的散列值（注意，这是对文件的内容进行的散列计算，基本上可以唯一地标识内容相同的文件），而其对应的 Value 同样是一个列表，包含了当前所有拥有该文件的节点的网络信息，如源地址 IP、源地址端口号、源地址的节点 ID。客户端在获得这些信息后，就可以到源地址下载该文件。

3. Kad 的 ID 计算方法

在 Kad 网络中，客户端的 ID 计算更加关注于是否 open 和 freely，其方法与 eMule 协议中的不太一致，一个 IP 地址为 X.Y.Z.W 的客户端的 ID 值计算如下所示。

$$ID = 2^{24} \times X + 2^{16} \times Y + 2^{8} \times Z + W$$

Kad 的 ID 计算方法为那些 W 为 0 的用户使用 eMule 网络提供了途径。但是内网用户在使用的时候依旧无法达到内网用户完全穿透网络的效果。

Kad 本身有一个 nodes.dat 的文件，即节点文件，里面存放了客户端在 Kad 网络中的邻居节点，客户端通过这些节点进入 Kad 网络。这些节点在分布式 P2P 网络中也被称为“导入节点”。

7.3　eMule 系统设计

eMule 工程是个非常庞大的工程，其中包括 100 多个类，2 000 多个文件。对它的源代码

www.ckook.com

的详尽分析，足可以写一本书。本节将就其主要的几个模块的结构以及接口进行分析和描述，为读者学习其源代码提供一些思路。

7.3.1 eMule 系统结构概述

eMule 系统是一个典型的 MFC 程序，它的图形界面等，已经和 MFC 紧紧融合到了一起。因此，通常情况下它只能在 Windows 平台下运行。通过 Rational Rose 对 eMule 工程进行逆向软件工程，得到其系统的静态结构图，如图 7-23 所示。通过这个静态结构图可以看出 eMule 工程的庞大性。但是 eMule 的代码结构非常合理，各个功能模块之间的划分都非常清楚，并且部分模块实现的功能还可以考虑在其他程序中进行复用。

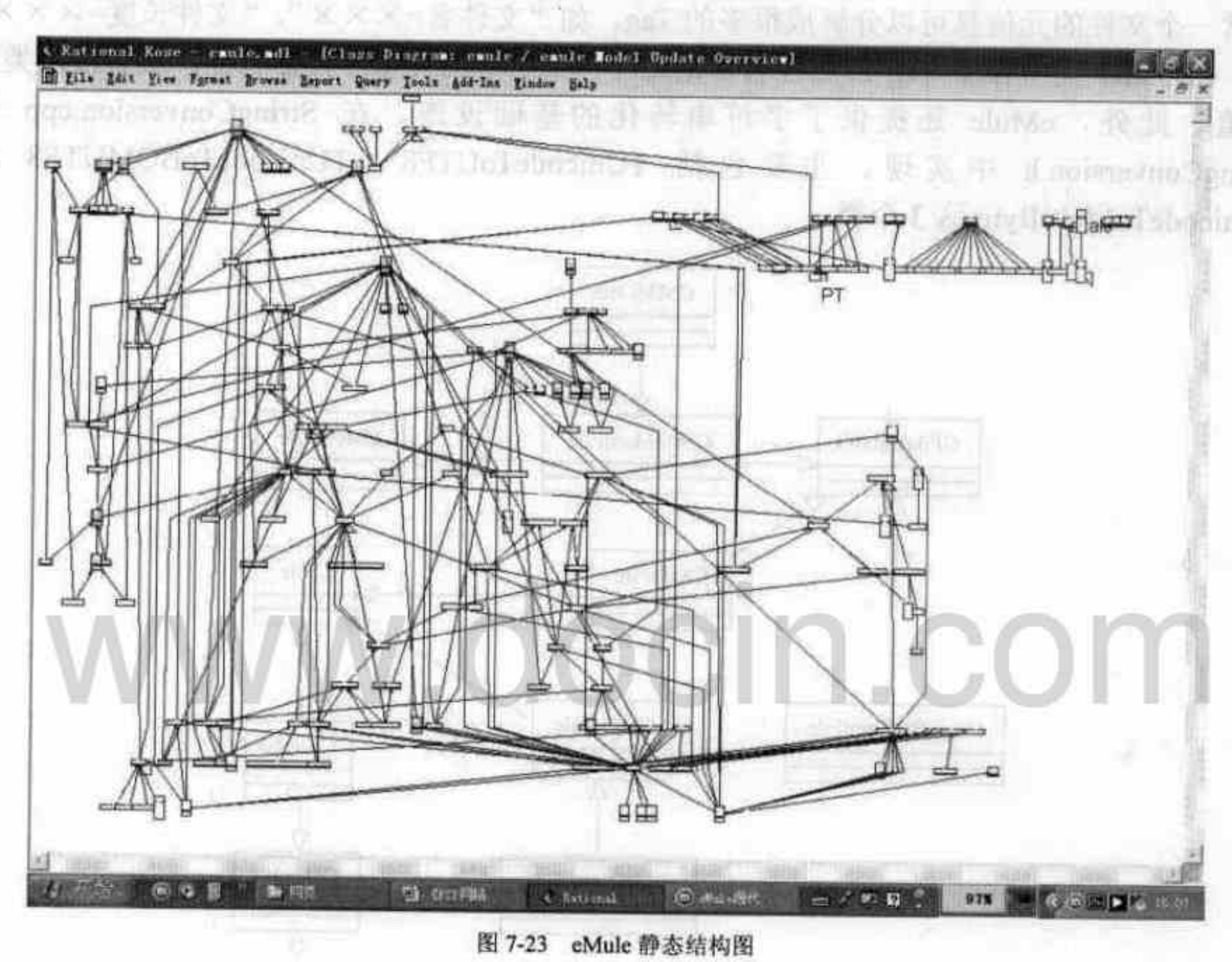


图 7-23 eMule 静态结构图

总的来说，eMule 系统包括文件基础设施、网络基础设施、eMule 通信协议、任务处理机制、Kademlia 子系统几个主要组成部分。

7.3.2 文件基础设施

在 eMule 中，有两个比较重要的类，CPreferences 和 CStatistics。前者掌握着程序的大部分配置数据，而后者则进行各种统计信息。这两个类的特点是都有很多的静态成员变量，以保证其属性的唯一性。这两个类提供了大量的方法对其属性进行获取和修改。这两个类的唯

www.ckook.com

一实例分别是 thePrefs 和 theStats。

在 eMule 中，要读取的配置文件数量很多，各类配置文件都有自己的定义格式。为了便于对这些文件进行操作，eMule 设计了一个很重要的基础设施类，它能够很方便地处理一些常用数据类型的读写，并且带有一定的安全保护机制。在 kademlia\io 目录下，提供了另外一种方法实现该功能。这两种基础设施类如图 7-24 所示。这两种方法的实现都把数据操作的行为和数据操作的对象分割开。首先定义一个基类，提供所有逻辑操作数据的行为，然后通过一个纯虚函数提供物理上操作数据的方法。这样，继承类就可以很方便地通过纯虚函数的实现来完成各种格式的配置文件的读写。这两个基类分别是 SafeFile.h 和 SafeFile.cpp 中实现的 CFileDataIO 类以及 DataIO.h 和 DataIO.cpp 文件中实现的 CDataIO 类。在 CDataIO 类中，还实现了按照 Tag 进行读写的功能。Tag 定义了文件的某个属性的取值，通常一个文件的元信息可以分解成很多的 Tag，如“文件名=×××”、“文件长度=×××”等。CDataIO 类中存储 Tag 的属性名都是先存一个字节的类型，再存名称，最后按照类型存值。此外，eMule 还提供了字符串转化的基础设施，在 StringConversion.cpp 和 StringConversion.h 中实现，主要包括 TUnicodeToUTF8、TUnicodeToBOMUTF8 和 TUnicodeToMultiByte 这 3 个类。

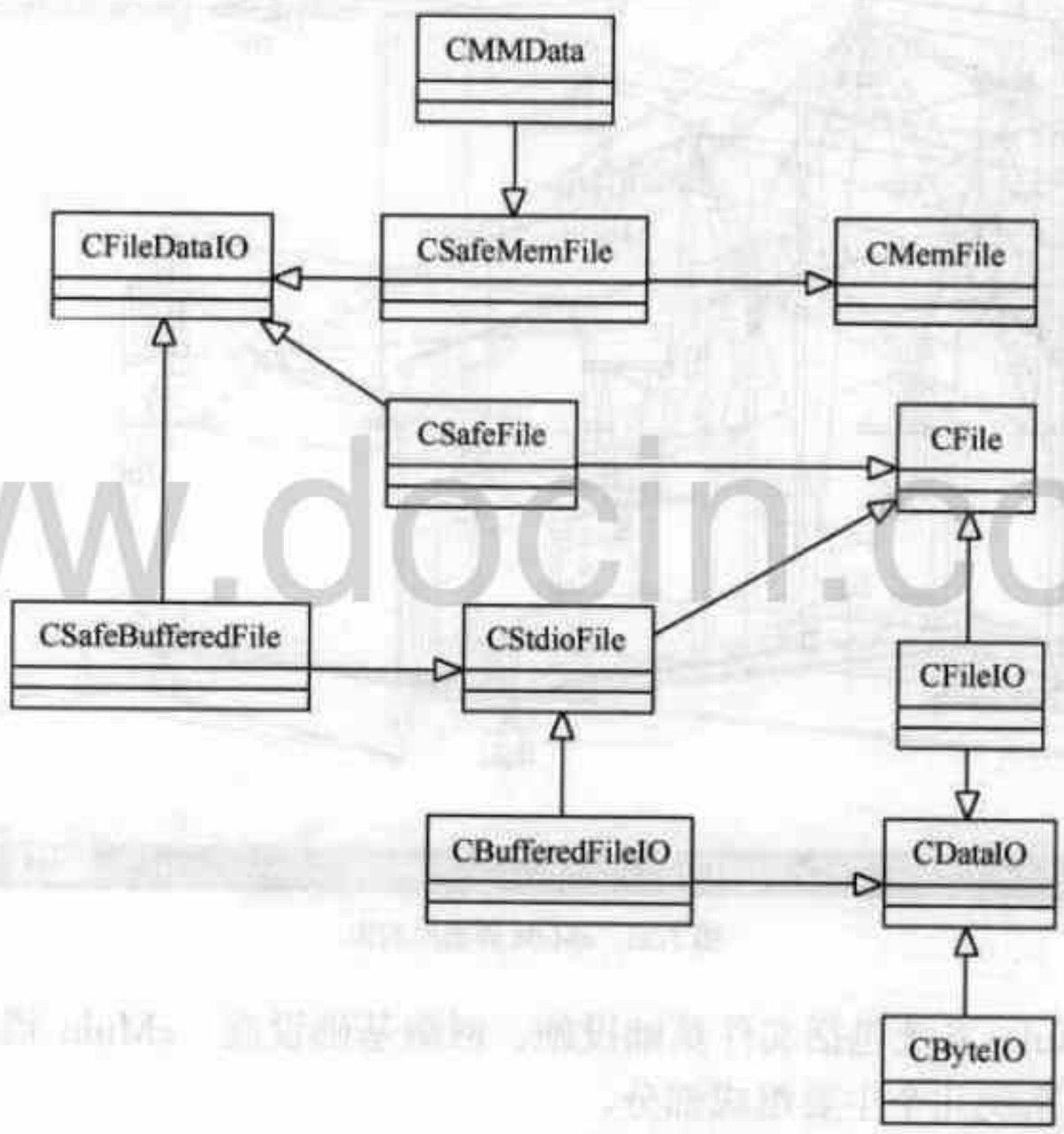


图 7-24　eMule 文件基础设施类图

eMule 通过 CKnownFileList 类管理自己共享的文件信息。它使用了 MFC 的 CMap 类来维护内部的散列表，包括两部分，已知文件列表和取消的文件列表。该类提供了对文件列表的各类操作，但并不提供对具体文件的操作。而对某个特定文件的信息的操作由 CKnownFile 类实现。该类继承了 CAbstractFile 类。它将读到的文件信息保存成一个一个的 Tag，并且能够随时掌握目前该文件的下载情况，如图 7-25 所示。

www.docin.com
www.ckook.com

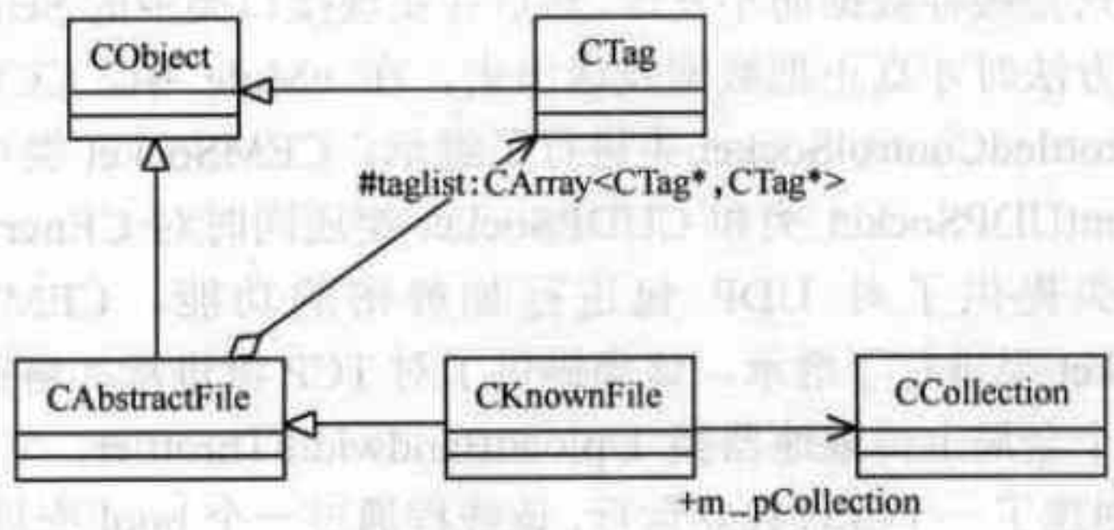


图 7-25 CKnownFile 类图

7.3.3 网络基础设施

在 eMule 的设计中，为了能够更加高效地开发网络相关的代码，构建了一些网络基础设施类，这些类的代码有很高的复用价值。它的网络基础设施类如图 7-26 所示。

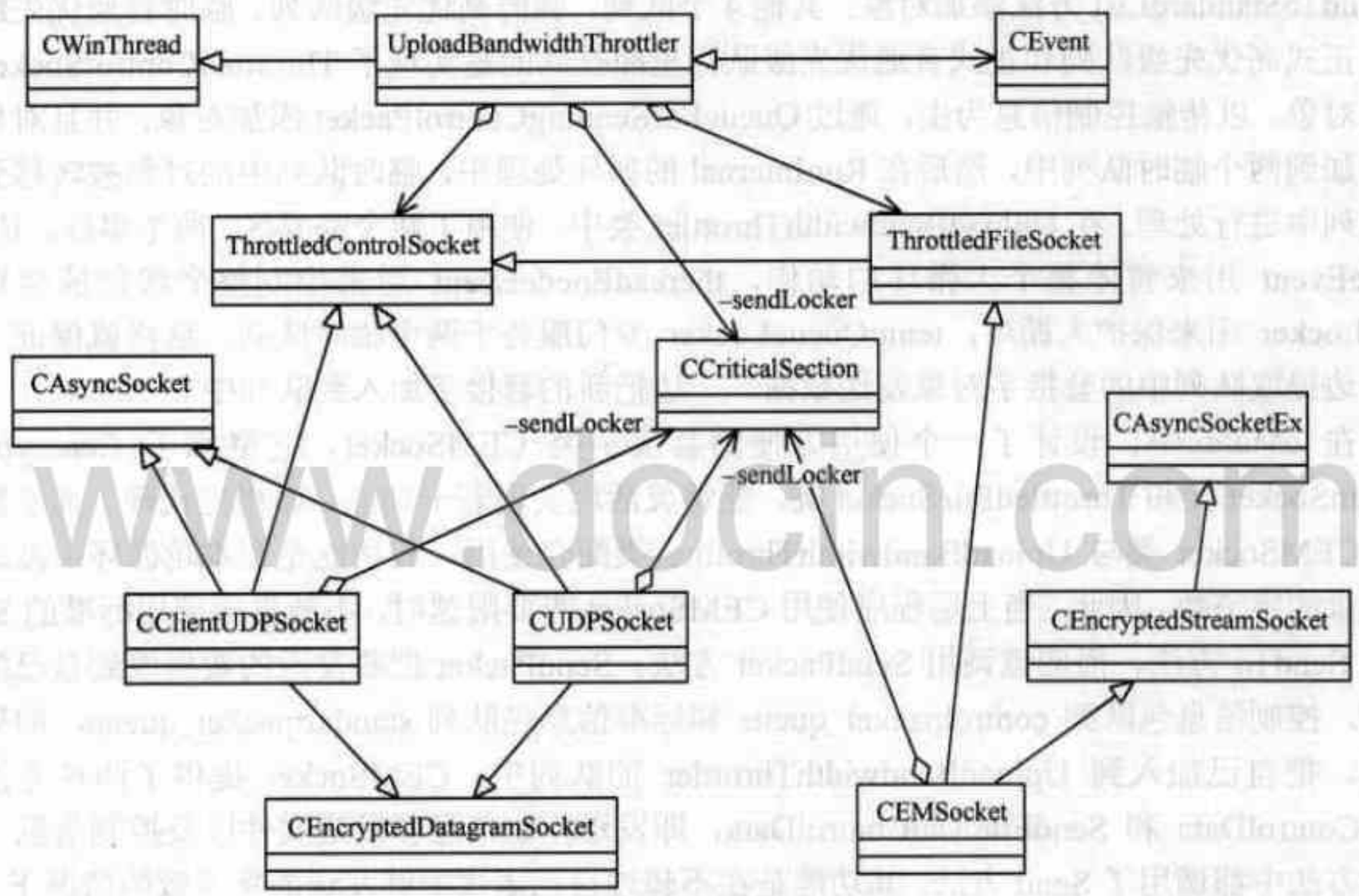


图 7-26 eMule 网络基础设施类图

首先，eMule 中设计了一个 CAsyncSocketEx 类以替代 MFC 中的 CAsyncSocket 类，它提供了所有 CAsyncSocket 类实现的方法，并且提供了一些增强方法。如它在效率上更高，主要是在消息的分发机制上，它通过定义一系列的消息类型提高了消息的分发效率；CAsyncSocketEx 类支持通过实现 CAsyncSocketExLayer 类的方式，将一个 Socket 灵活地分成若干层，从而可以方便地实现许多网络功能，如代理或者 SSL 加密等。

其次，eMule 中设计了两个接口类，ThrottledControlSocket 类和 ThrottledFileSocket 类，以实现限速功能。其他网络套接字类在需要实现限速功能时，只需要继承这两个接口类，在

www.docin.com

其默认的发送函数中只是缓存数据而不发送，然后在实现接口类中的 SendFileAndControlData 或 SendControlData 方法时才真正把数据发送出去。在 eMule 中，CClientUDPSocket 类和 CUDPSocket 类对 ThrottledControlSocket 类进行了继承，CEMSocket 类对 ThrottledFileSocket 类进行了继承。CClientUDPSocket 类和 CUDPSocket 类还同时对 CEncryptedDatagramSocket 类进行了继承，该类提供了对 UDP 包进行加解密的功能。CEMSocket 类也同时对 CEncryptedStreamSocket 类进行了继承，该类提供了对 TCP 流进行加解密的功能。

eMule 提供了一个全局上传限速器类 UploadBandwidthThrottler，它继承了 CWinThread 类。该类的构造函数创建了一个线程独立运行，该线程通过一个 bool 变量判断是否结束工作。为了避免线程的目标静态函数 RunProc 无法调用 this 指针，在该函数中调用了一个成员函数 RunInternal 来完成实际的工作。在该函数中，设计了一个大循环来负责处理工作线程的日常操作，包括计算本次循环中能够发送的字节数，以安排调度；计算本次循环应该休眠的时间，以进行限速；操作标准队列和控制信息队列，发送队列中的数据。在 UploadBandwidthThrottler 类中，保存了 5 个套接字队列。其中，标准队列（m_StandardOrder_list）里面排队的是实现了 ThrottledFileSocket 接口的对象，这些对象既能够传输文件内容，也可以传输控制信息，通过 AddToStandardList 方法添加对象；其他 4 个队列，临时高优先级队列、临时普通优先级队列、正式高优先级队列和正式普通优先级队列里面排队的是实现了 ThrottledControlSocket 接口的对象，以传输控制信息为主，通过 QueueForSendingControlPacket 添加对象，并且对象首先添加到两个临时队列中，然后在 RunInternal 的循环处理中，临时队列中的对象被转移到正式队列中进行处理。在 UploadBandwidthThrottler 类中，使用了两个临界区，两个事件。其中，pauseEvent 用来暂停整个大循环的动作，thereadEnedeEvent 用来控制整个线程的结束，sendLocker 用来保护大循环，tempQueueLocker 专门服务于两个临时队列。这样就保证了能够一边提取队列中的套接字对象发送数据，一边把新的套接字加入到队列中。

在 eMule 中，设计了一个使用方便的套接字类 CEMSocket，它继承了 CencryptedStreamSocket 类和 ThrottledFileSocket 类，能够灵活地实现若干功能，如指定代理、加密数据等。CEMSocket 类与 UploadBandwidthThrottler 类配合使用，其功能是在本轮循环中发送指定数量的字节数。因此，当上层程序使用 CEMSocket 需要限速时，不能直接调用标准的 Send 或者 SendTo 方法，而应该调用 SendPacket 方法。SendPacket 把要发送的数据放到自己的队列中，控制信息包队列 controlpacket_queue 和标准信息包队列 standartpacket_queue。如果有必要，把自己加入到 UploadBandwidthThrottler 的队列中。CEMSocket 提供了两种方法：SendControlData 和 SendFileAndControlData，即发送控制信息和发送文件以及控制信息。这两种方法中都调用了 Send 方法，其功能是在不超过自己本次能够发送的字节数的情况下，从队列中将数据取出并发出去。因此，当 CEMSocket 与 UploadBandwidthThrottler 结合使用时，就形成了一个两层的队列，即所有套接字组成了一个发送队列，在 UploadBandwidthThrottler 的控制下保证了对速度的限制，而每个套接字即将发送的数据包又组成了一个队列，保证了每次进行数据发送时都能满足 UploadBandwidthThrottler 的要求。

7.3.4 eMule 通信协议

在第二节中介绍的 eMule 协议是由 Packet 类实现的。其中，在 Opcodes.h 中定义了 eMule

www.ckook.com

协议中的各字段，在 Packets.h 和 Packets.cpp 中定义了协议的构造和解析。Packet 类中提供了多个构造函数，使上层应用可以以不同的方式来创建 Packet。此外，Packet 类还在内部实现了压缩和解压的方法，通过 PackPacket 和 UnPackPacket 两个成员函数，该方法直接调用 zlib 库中的压缩方法。但 Packet 类中并没有提供对 eMule 协议中各字段的设置方法和访问方法，所以上层应用在填充和读取 eMule 协议中各字段时，必须知道协议字段的大小及填充顺序。这种设计虽然可以提高程序的效率（直接对数据的内存区进行操作），但同时也造成了上层应用与协议之间的强耦合性。可以考虑将 Packet 类设计得更为完善，将 eMule 协议进行更加完整的封装，并且提供对内部各个字段操作的外部成员函数，这也是很多协议在设计实现中所采用的方式。两种设计思路孰优孰劣，读者可以自己思考。

eMule 协议中分为 TCP 通信和 UDP 通信，如图 7-27 所示。在 TCP 通信中，数据流被划分成一个一个的包，这个工作由 CEMSocket 类完成，对于包的处理分别由 CEMSocket 类的两个继承类 CServerSocket 类和 CClientReqSocket 类完成，通过纯虚函数 PacketReceived 来实现。其中，CServerSocket 类负责处理客户端与服务器之间 TCP 连接发送的数据包，CClientReqSocket 类负责处理客户端之间 TCP 连接发送的数据包。eMule 协议中的 TCP 协议分为协议头和协议体，其中协议头是固定长度的 6Byte，如表 7-5 所示。在 UDP 通信中，CClientUDPSocket 类负责处理客户端与客户端之间的 UDP 数据包，CUDPSocket 类负责处理客户端与服务器之间的 UDP 数据包。此外，CKademliaUDPListener 类负责处理与 Kademilia 协议有关的 UDP 数据包。由于 UDP 通信本身就是基于包的单位进行传输的，因此 eMule 协议中的 UDP 协议的协议头只有一个协议类型字段和消息类型字段，不再包括消息长度字段。

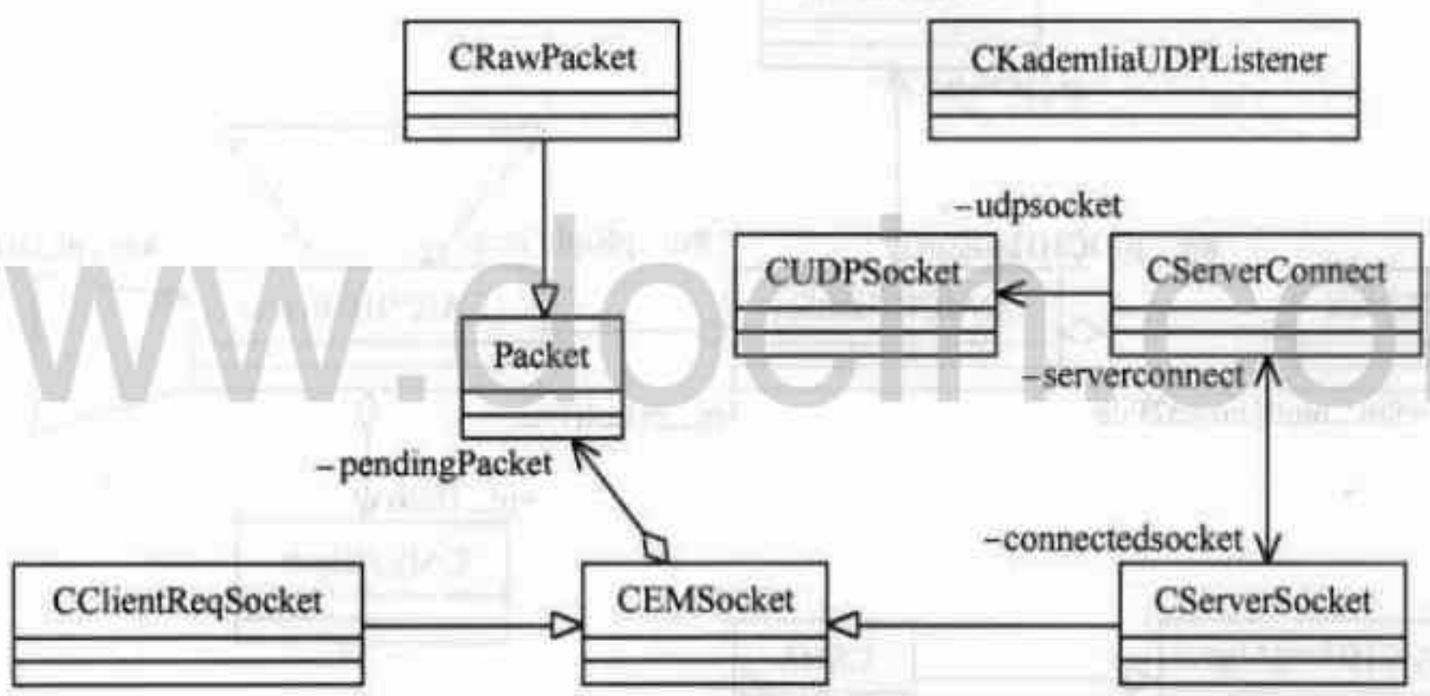


图 7-27 eMule 通信协议类图

www.dooin.com

表 7-5 **eMule 协议中 TCP 协议头描述**

字段名	大小	字段描述
协议类型	1Byte	表示协议的类型，包括 eMule 协议、Kad 协议等
消息长度	4Byte	表示协议的长度，不包括协议头
消息类型	1Byte	表示协议体的消息类型，包括注册消息、拒绝消息、获取服务器列表消息、提供文件消息等

7.3.5 任务处理机制

在本节中，将向读者简要地介绍 eMule 中的任务分块机制、信息集、信誉机制和任务队

www.ckook.com

列等几部分的内容。

1. 分块机制

eMule 在文件的分发上使用了文件分块机制，以加快文件传输的速度以及保证文件传输的正确性。eMule 的文件分块中，有两种不同大小的分块方式，9 500kByte 和 180kByte。对文件的分块处理发生在 eMule 获取共享文件的信息时。首先，eMule 将一个很大的共享文件分割成若干个 9 500kByte 的块，把这些块组织成一棵树；然后，每一个这样的块又被分解成若干个 180kByte 的块（52 个 180kByte 的块加上 1 个 140kByte 的块），仍然组织成树的结构。

由 CKnownFile 类中的 CreateFromFile 函数实现。在这个函数中，主要涉及了以下几个类。CAICHHash 类，负责存储文件块的散列值，提供赋值、比较等运算符重载操作。CAICHHashAlgo 类，是一个计算散列值的算法接口类，具体的散列算法由继承类来实现。在 eMule 中，CSHA 类继承了该接口类，实现了 SHA1 散列算法。CAICHHashTree 类实现了一棵散列树，它有两个成员变量指针，m_pLeftTree 和 m_pRightTree，分别指向其左子树和右子树。这是一种典型的树的实现方法。CAICHHashSet 是 CKnownFile 类的一个成员变量，CKnownFile 类通过它执行文件分块操作。CreateFromFile 函数通过 CAICHHashTree 的 FindHash 函数找到或创建对应的树枝节点，即一个指向 CAICHHashTree 的指针。这样，在对文件进行分块操作的时候，整个 CAICHHashTree 也建立起来了，每个分块的散列值都保存在对应的节点上。此外，CKnownFile 类中还有一个 hashlist 的成员变量，用以存储以 9 728 000Byte 为单位的整个文件的 MD4 算法的散列值。这部分代码的类图如图 7-28 所示。

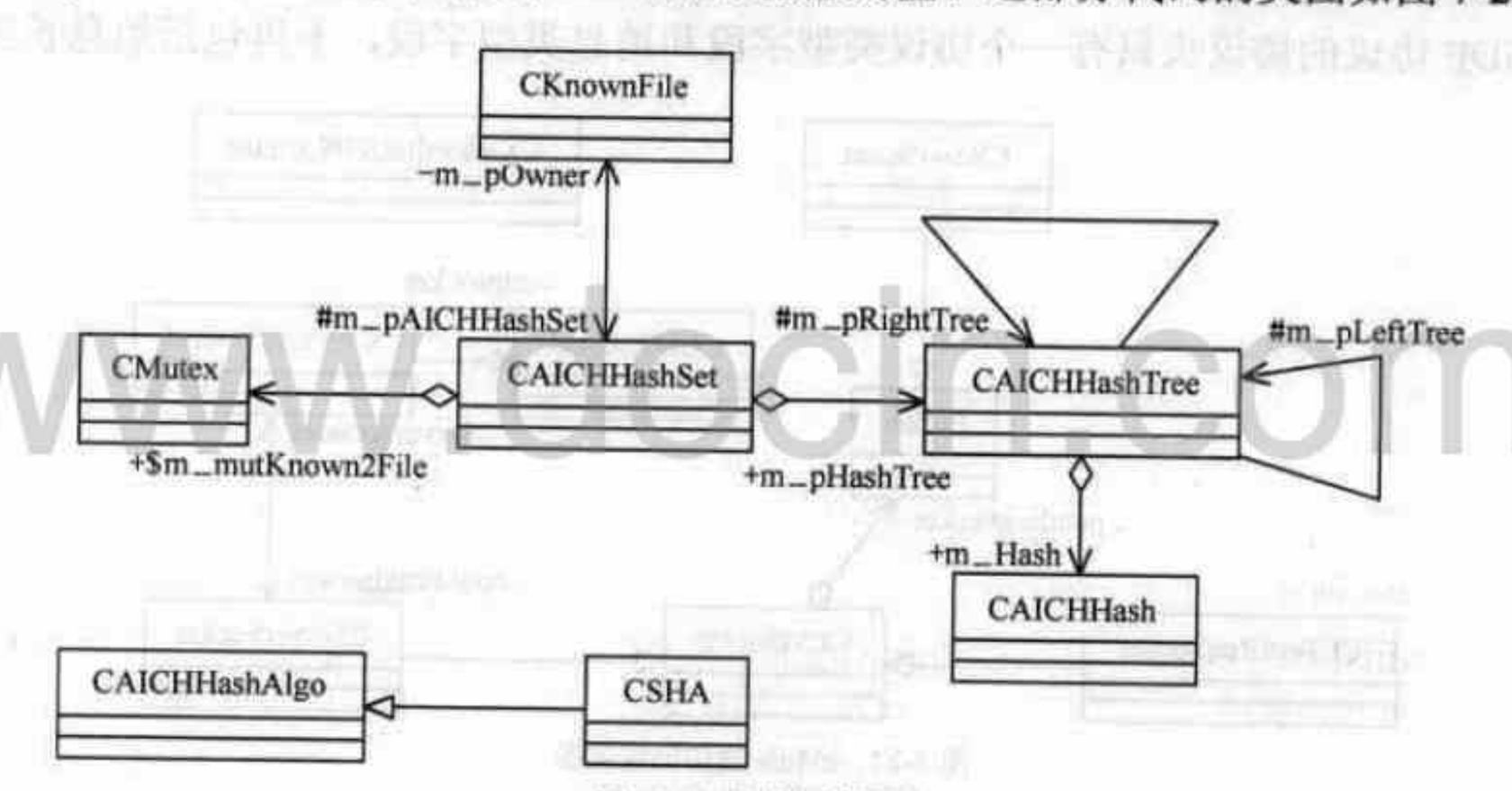


图 7-28 分块机制类图

2. 信息集

在 eMule 中，CSearchList 负责掌管所有搜索请求，CServerList 负责管理所有服务器列表，CClientList 负责管理所有客户端列表。这几个类的对象都在 CemuleApp 的 InitInstance 函数中实例化，其类图如图 7-29 所示。

CSearchList 的主要任务就是维护一个类型为 CSearchFile 的列表，提供方法供外部对这个列表进行添加、删除、查询和修改等操作。CSearchList 通过两个 CMap 的对象，m_foundFilesCount 和 m_foundSourcesCount，分别记录了每次搜索中目前搜索到的文件个数以及源的个数，通过每次搜索的独有的 32bitID 进行标识。当有一个新的搜索提交时，通过

www.ckook.com

CSearchList 的 NewSearch 函数建立一个新的搜索项，添加到 m_foundFilesCount 和 m_foundSourcesCount 中。当有搜索结果返回时，函数 ProcessSearchAnswer 或者 ProcessUDPSearchAnswer 能够直接对数据进行处理，创建对应的搜索文件信息对象，并且加入到自己的列表中。CSearchList 中的元素是 CSearchFile，代表了每次搜索的相关信息。CSearchFile 同 CKnownFile 一样，继承了 CAbstractFile，保存了某个文件与搜索相关的信息，而不是该文件本身的信息，包括：哪些机器上有这个文件；哪个服务器上搜到这个文件；向搜索文件添加预览等。CSearchFile 中定义了两个结构体，SServer 和 SClient，并且有两个对应该结构体的成员变量数组，m_aClients 和 m_aServers，用以存储该搜索文件的可能来源，包括服务器或者其他客户端。CSearchFile 提供了对这两个数组的简单的操作方法，包括添加和获取。此外，CSearchFile 中还有一个二级列表，m_list_parent，用以保存同一个散列的文件的多个源。

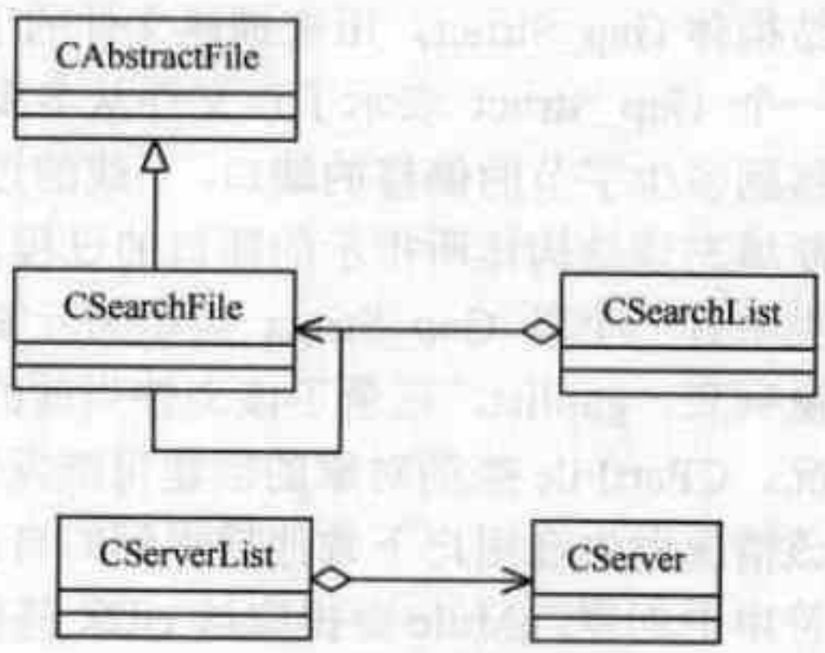


图 7-29 信息集类图

CServerList 的主要任务是维护一个类型为 CServer 的列表，提供方法供外部对这个列表进行添加、删除、查询和修改等操作。服务器的信息列表是需要长期保留的，因此 CServerList 提供了将其所有信息保存到一个文件以及从该文件中读取其信息的功能。此外，CServerList 还提供了一些统计信息的功能，譬如所有的服务器的用户数、共享的文件数等。CServerList 的元素是 CServer，保存了服务器的各种信息。它可以通过 IP 地址和端口号来创建，也可以通过结构体 ServerMet_Struct 来创建。ServerMet_Struct 包含了 IP 地址、端口号以及属性的个数，用以从文件中读取。CServer 的其他属性在保存到文件中时，均采用 Tag 方式。

3．信誉机制

eMule 中采用了 tit-for-tat 的激励机制，即“你对别人好，别人也对你好”，以使用户更加愿意共享自己的资源。这在 P2P 系统中是非常重要的。在实现中，CClientCreditsList 类和 CClientCredits 类负责 eMule 中的信誉机制。和信誉相关的信息是需要永久保存的，因此 CClientCreditsList 提供了 LoadList 和 SaveList 的方法，从文件中获取或者保存信誉相关的信息。CClientCreditsList 中有一个以 CClientCredits 类型为基础元素的 CMap 类的成员变量。CClientCredits 类提供的构造函数中，包括使用 CreditStruct 结构来创建的方法，这个结构主要包括了静态信息，如上传量和下载量等。

为了保证信誉机制的可靠性，eMule 中采用了数字签名，通过 Crypto++库提供其与数字签名验证相关的功能。CClientCreditsList 在创建时，在 InitalizeCrypting 中会装载或者创建自己的公私钥对。CClientCreditsList 中包含的有效的信息都是经过其他人数字签名的，因此在实际使用中，这些信息和自己的私钥应该注意保存，以保留自己积攒的信誉。

4．任务队列

在 eMule 中，当开始下载一个文件时，会创建两个临时文件，以 3 位数字为名的.part 文件以及与其同名的.part.met 文件。其中前者会创建得和原始文件大小一样，而后者用以存储文件的元信息，如该文件的原名、修改日期和已下载完成的部分等。这部分的类图如图 7-30 所示。

www.ckook.com

eMule 中的 CPartFile 类用来描述下载任务，它继承了 CKnownFile 类。该类中，使用了一个结构体 Gap_Struct，用来描述文件的下载情况。一个 Gap_Struct 表示了该文件从多少字节的偏移到多少字节的偏移的缺口。下载的过程就是不断填充该结构体所指示的缺口的过程。CPartFile 类中有一个以 Gap_Struct 类型为元素的成员变量列表，gaplist，记录了该文件当前的缺口的情况。CPartFile 类的对象的创建可能发生在几种情况下：①从搜索文件 CSearchFile 中创建，该情况发生在用户下载他搜索到的自己需要的文件时发生。②从一个包含了 ed2k 链接的字符串中创建，eMule 会提取该 ed2k 链接中的信息，用以创建 CPartFile 类的对象。③当 eMule 程序启动时，恢复上次关闭时没有完成的下载任务，从下载目录中寻找.part 文件和.part.met 文件，创建 CPartFile 类的对象。此外，CPartFile 类还要处理不断得到的下载数据，通过以结构体 Requested_Block_Struct 为元素的成员变量列表 requestedblocks_list 进行缓存，以减少磁盘开销。CPartFile 类通过一个以 CUpDownClient 指针为元素的成员变量列表 srclist，用以保存该文件的来源信息。CPartFile 提供 Process 方法，处理一些和日常运行有关的事情，如检查该文件所关联的各个客户端的状态、向它们发送下载请求等。

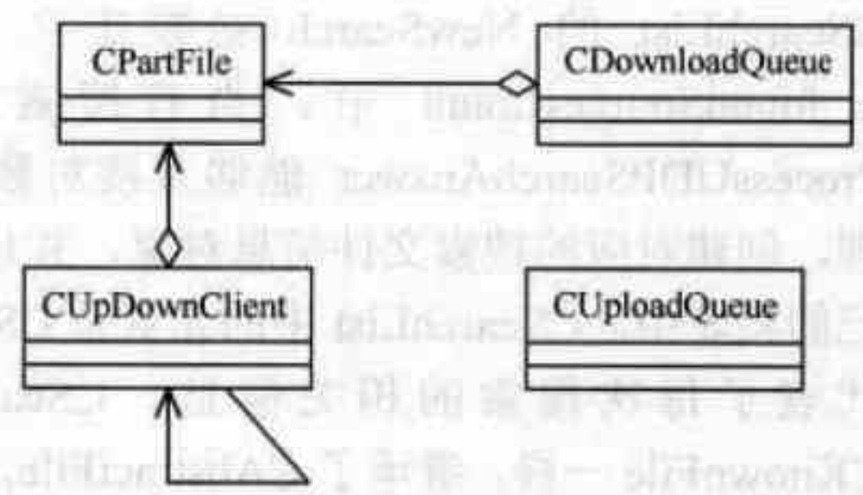


图 7-30 任务队列类图

eMule 中的 CDownloadQueue 类负责处理下载队列，这个队列中的项目是 CPartFile 的指针。它提供对这个列表中的元素进行增加、查询、删除以及统计等功能。由于每个 CPartFile 类的对象对应两个临时文件，所以在 CDownloadQueue 初始化时，需要寻找所有可能的下载路径，找到所有的.part.met 文件，生成 CPartFile 类的对象，并且添加到自己的列表中；在退出时，CDownloadQueue 也需要把所有的下载任务的元信息分别保存到各自对应的.part.met 文件中。CDownloadQueue 中提供 Process 方法遍历调用其列表中的 CPartFile 对象的 Process 方法，此外，一些关于下载情况的统计信息也是在每次 Process 后进行更新的。

eMule 中的 CUploadQueue 类负责处理上传队列，这个队列中的项目是 CUpDownClient 的指针。由于 CUploadQueue 队列中保存的信息是上传的客户端，所以该信息不需要持久化。CUploadQueue 类中有两个成员变量列表，waitinglist 和 uploadinglist，分别代表排队列表和上传列表。当收到一个新的下载请求时，首先将对应的客户端添加到排队队列中，然后根据情况，把排队队列中的客户端不断添加到上传队列中去。CUploadQueue 中提供 Process 方法向上传队列中的所有客户端依次发送数据，以及完成一些关于上传方面的统计信息。在 CUploadQueue 的构造函数中设置了一个定时器，该定时器将触发一个 UploadTimer 的函数。在该函数中，完成了以上所有 Process 方法的调用。

CUpDownClient 类是 eMule 中代码量最大的类，其作用是从逻辑上表示一个其他的客户端的各种信息。它的定义是在 UpDownClient.h 文件中，实现是在 BaseClient.cpp、DownloadClient.cpp、PeerCacheClient.cpp、UploadClient.cpp 和 URLClient.cpp 中。其中，在 BaseClient.cpp 文件中，主要实现了该类的一些基本功能，如各种状态的获取、设置，以及按照要求处理和发送各种请求等；在 DownloadClient.cpp 文件中，实现了与下载相关的功能，包括各种下载请求的发送以及相应数据的接收等；在 UploadClient.cpp 文件中，实现了与上

www.ckook.com

传相关的功能，包括接收下载请求，生成相应的文件块发送出去等；在 PeerCacheClient.cpp 文件中，实现了与 PeerCache 技术相关的功能，该技术允许从 ISP 提供的一些快照服务器上快速地上传或者下载一些文件，将部分本来需要在骨干网上传输的流量转移到 ISP 的内部，以减少骨干网络的带宽消耗；在 URLClient.cpp 文件中，实现了利用 HTTP 协议对原有的 eMule 协议的包装，以使其尽可能地穿越更多的网络防火墙。CUpDownClient 类实现了一个 A4AF（ask for another file）的机制。因为一个客户端在同一个时间内只能向另外一个客户端请求同一个文件。而对于很多的下载任务，有可能出现其源是相同的情况。这样，当有下载任务从这个源下载时，其他下载任务就不能从这个源下载了。eMule 中允许用户对其手动进行控制，将源从一个下载任务切换到另一个下载任务，这就是 A4AF 机制。

7.3.6 Kad 系统结构概述

1．Kad 系统总体结构

在 eMule 网络中，实现了一个完全的 P2P 网络——Kad 网络。该网络中不存在中心节点，所有的用户即是客户端，也是服务器，有关 Kad 网络的算法详见 3.3.6 节。在 eMule 中，CKademlia 类是整个 Kad 网络的主控类，负责开始或停止 Kad 网络，处理日常事务；CPrefs 类负责处理本地的与 Kad 相关的信息；CRoutingZone、CRoutingBin 和 CContact 这 3 个类组成了每个客户端节点所了解的联系信息及由这些信息所组成的数据结构；CKademilaUDPListener 类负责处理网络信息；CIndexed 类负责处理本地存储的索引信息；CSearch 和 CSearchManager 这两个类负责处理和搜索相关的操作，其中前者表示的是一次单任务的搜索，后者负责对所有搜索任务的处理；CUInt128 实现了一个表示 128bit 的整数的类，功能十分完善，可以进行一些算术操作，并且可以进行比较。这部分代码的类图如图 7-31 所示。

www.docin.com

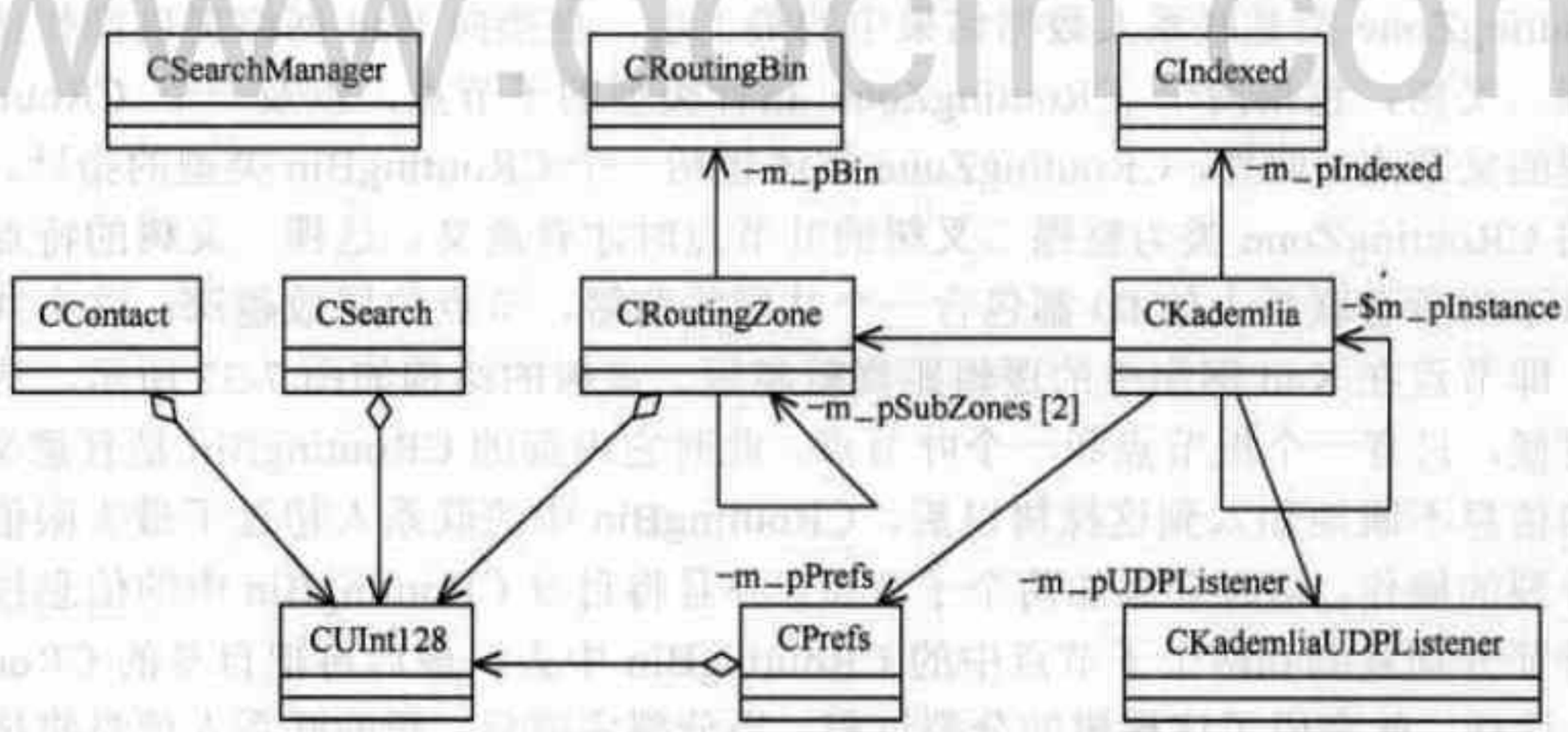


图 7-31 Kad 类图

2．Kad 系统的基础设施类

Kad 的主控类是 CKademlia，它通过 Start 方法来创建该类唯一的静态对象指针，完成对 Kad 网的启动，通过 Stop 方法来销毁该类唯一的静态对象指针，完成对 Kad 网的关闭。CKademlia 还提供了其他的静态方法，实现对 Kad 网的控制功能。其中，Process 函数负责处

www.ckook.com

理与 Kad 网络相关的事务，如定期检查某个区间是否因为节点数过少而需要添加新的节点、定期检查邻居节点等。此外，Process 函数还负责调度所有搜索任务的日常处理以及向 eMule 其他模块返回 Kad 网络的一些统计信息。

CPrefs 类是 Kad 实现中的另一个基础设施类，其作用与 eMule 中的 CPreferences 类类似，只是它只保留了与 Kad 网络相关的、需要长期保存的本地信息。该类的唯一实例在 CKademlia 启动时创建，提供了大量的方法对其属性进行获取和修改。

CUInt128 类是 Kad 实现中的又一个基础设施类，它实现了对 128bit ID 的各项操作，包括获取、设置、比较、加减、异或以及很多算术符重载等。仔细地学习 CUInt128 类中的代码，可以学到很多在编写这种自定义的抽象数据类型 ADT（Abstract Data Type）时应该注意的问题。

3．Kad 网络中联系人的数据结构

CRoutingZone、CRoutingBin 和 CContact 这 3 个类组成了客户端列表的数据结构，它既要满足在 Kad 网络中搜索到目标的时间要求，又要满足数据结构所占用的空间要求。

在 CContact 类中，包含了一个联系人的信息，包括联系人的 IP 地址、ID、TCP 端口、UDP 端口、Kad 版本号以及其健康程度，这些都可以从该类的成员变量中找到。CContact 类提供了大量对联系人信息进行读取和设置的方法。其中，健康程度分为 5 个等级，分别从 0 到 4。联系人初始化时健康程度设置为 3。系统经常会与联系人进行联系，以判断其健康状况。经常能够联系上的联系人，健康程度会慢慢减少到 0；很久没有联系上的联系人，健康程度会慢慢增加到 4，如果再过一段时间还是未能联系上，该联系人将从联系人列表中被剔除。

CRoutingBin 类包含了一个 CContact 类的列表，提供了获取联系人对象指针的方法。所有的联系人信息都由 CRoutingBin 类进行管理和维护，添加和查找联系人必须通过该类。另外，一个 CRoutingBin 类的对象中能够包含的联系人的数量也是有限制的。

CRoutingZone 类是联系人数据结果中的最上层，直接向 Kad 网络提供操作接口。该类是一棵二叉树，包括两个 CRoutingZone 指针类型的子节点，以及一个 CRoutingZone 指针类型的父节点。此外，CRoutingZone 类还包括一个 CRoutingBin 类型的指针，该指针在当前的 CRoutingZone 类为整棵二叉树的叶节点时才有意义。这棵二叉树的特点是：每个节点以下的所有联系人的 ID 都包含一个共同的前缀，节点的层数越深，这个共同前缀就越长，即节点在 Kad 网络中的逻辑距离就越短。该树的结构如图 7-32 所示。这棵树刚开始的时候，只有一个根节点和一个叶节点，此时它内部的 CRoutingBin 是有意义的。当联系人的信息不断地加入到这棵树以后，CRoutingBin 中的联系人超过了最大限值，就进行一个分裂的操作。这时会添加两个子节点，并且将自身 CRoutingBin 中的信息按照节点前缀的特征分别复制到两个子节点中的 CRoutingBin 中去，最后再把自身的 CRoutingBin 删除掉。这样，就完成了这棵树的分裂过程。当分裂完成后，新的联系人信息将按照节点前缀的特征加入到最后的叶子节点中。但是并不是所有的这种情况节点都会发生分裂，因为如果允许任意分裂的话，本地所需存储的节点信息数量就会急剧上升。这里，自身 ID 的作用就体现了。只有当自身 ID 和当前准备分裂的节点有共同前缀时，这个节点才会分裂，而如果判断到一个节点不能分裂，而它的 CRoutingBin 又满掉了，那么就会拒绝添加联系人信息。

www.ckook.com

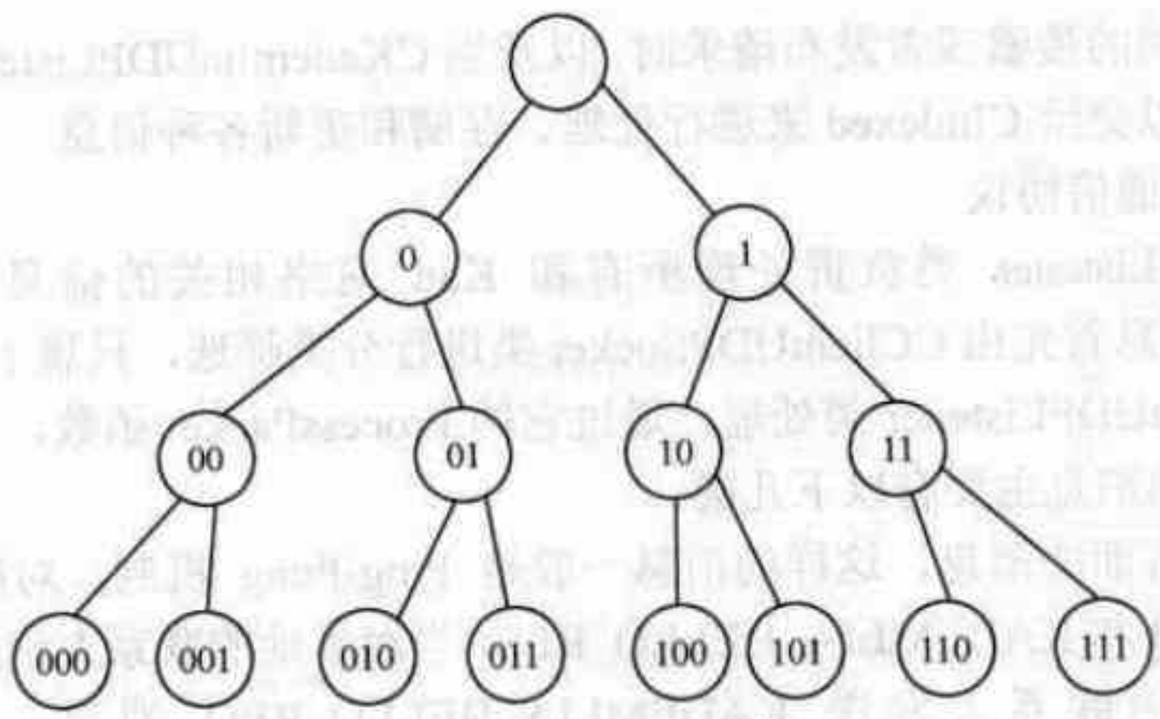


图 7-32 CRoutingZone 树

在这种节点添加策略中，离自身 ID 逻辑距离越近，即共同前缀越长的联系人信息越有可能被加入成功，因为它对应的节点越有可能因为分裂而获得更多的子节点，也就对应了更多的存储容量。这样，在 Kad 网络中，每一个参与者掌握的离自己逻辑距离越近的节点的数量比例越多。Kad 网络搜索算法的时间代价是 $O(\log N)$，而从这个二叉树的结构来看，由于只有部分节点会分裂，所以实质上存储所需要的空间代价也是 $O(\log N)$。实际上 CRoutingZone 在实现时和理论上的 Kademlia 协议也有一些区别。它从根节点开始，有一个最低的分裂层数 KBASE，在这个分裂层数的范围内，都可以进行分裂，这样它知道的其他地区的联系人信息就能够稍微多一些。

4. Kad 网络中文件信息的数据结构

Kad 网络的最大特点是无中心服务器，它把 eMule 协议中存储在中心索引服务器中的信息分散存储到各个客户端中。一个文件源的信息，对应着一个文件内容的散列值与拥有这个文件的客户端的 IP 地址、端口号以及其他信息。一个关键字的信息，对应着该关键字以及它相关的文件。在关键字的信息中，它所对应的文件信息要更加详细，通常包括该文件的文件名、文件大小、文件内容的散列值，如果是 MP3 或者其他媒体文件，还会包括作者、生产时间、媒体文件播放时间长度、流派等 Tag 信息。

在 eMule 中，CIndexed 类正是为存储这些信息所设计的类。该类的对象在 CKademlia 的 Start 中实例化。在 CIndexed 类中，设计了 4 个 CMap 类的对象来存储信息，分别存储文件源信息 m_mapSources、关键字信息 m_mapKeyword、文件评论信息 m_mapNotes 以及负载信息 m_mapLoad。其中，文件评论信息是不用长久保存的，而其他的信息都会在系统结束时写到文件中，下次系统重新启动时再从文件中读入。此外，负载信息不是由其他联系人发布的，而是根据文件源信息和关键字信息的发布情况自行动态调整的。每一次收到发布信息时，对应的 ID 的负载就会增大。

CIndexed 中的信息会经常进行检查，每隔 30min 它会把自己存储的信息中太旧的信息清除掉。其中，文件源信息的保存时间是 5h，关键字信息的保存时间是 24h，文件评论信息的保存时间是 24h。因此，文件和关键字的发布也要周期性地反复进行。这对于整个 Kad 网络的稳定性也是有好处的，因为每一次联系都会试图把对方添加到自己的联系人列表中，或者在联系人列表中标注上一次见到对方的时间。

CIndexed 类为其他部分的代码提供了它们所需要的增加信息和搜索信息的接口，这样当

www.ckook.com

从网络中获取到相关的搜索或者发布请求时，以及当 CKademliaUDPListener 完成对收到的消息的解析后，就可以交给 CIndexed 来进行处理、存储和更新各种信息。

5．Kad 网络的通信协议

CKademliaUDPListener 类负责处理所有和 Kad 网络相关的信息。发给 CkademliaUDPListener 类的消息首先由 CClientUDPSocket 类进行分类筛选，只属于 Kad 网络相关的消息转发给 CKademliaUDPListener 类处理，通过它的 ProcessPacket 函数。

Kad 网络相关的消息主要有以下几类：

（1）健康检查方面的消息，这样的消息一般是 Ping-Pong 机制，对应的消息有 KADEMLIA_HELLO_REQ 和 KADEMLIA_HELLO_RES。当对本地的联系人信息列表进行检查时，会依次对列表中的联系人发送 KADEMLIA_HELLO_REQ 消息，然后处理收到的 KADEMLIA_HELLO_RES 消息。

（2）节点搜索方面的消息，对应的消息有 KADEMLIA_REQ 和 KADEMLIA_RES。在 Kad 网络中，进行节点搜索是最重要的行为之一，实现的方式是迭代式的搜索。当开始搜索某个 ID 时，eMule 首先在本地的联系人列表中寻找距离最近的联系人，向它们发送搜索请求，然后再由这些联系人寻找距离更近的联系人信息，继续发送搜索请求。通过这样不断地逼近目标节点，就能够得到距离目标 ID 最近的那些联系人的信息。

（3）内容发布方面的消息，对应的消息有 KADEMLIA_PUBLISH_REQ 和 KADEMLIA_PUBLISH_RES。eMule 中存储在 Kad 网络中的信息主要有 3 类：文件源信息、关键字信息和文件的评论信息。其中，文件源对应的是每一个具体的文件，其标识是该文件内容的散列值。文件源信息表明了某人拥有某个特定文件的这样一个事实。关键字对应的是文件的搜索特征，其标识是该关键字内容的散列值。eMule 允许用户对某个文件发表评论，评论的信息单独保存，其标识是该文件内容的散列值。一个关键字可能会对应多个文件，而一个文件也可能对应多个文件源。以散列算法作为关键字和文件的索引，使得搜索和发布都变得非常简单。每个 eMule 客户端都拥有自己共享文件的详细信息。在 Kad 网络中，它们首先将文件名进行分词，提取关键字，然后计算出关键字的散列值，并且把这些关键字信息发布到 ID 与关键字散列值距离最近的联系人那里，把文件源信息发布到 ID 与文件内容散列值距离最近的联系人那里。提取关键字的方法是，从文件名中寻找有分割符含义的字符，如下划线等，然后把文件名分成一个个的关键字。

（4）文件搜索方面的消息，对应的消息有 KADEMLIA_SEARCH_REQ 和 KADEMLIA_SEARCH_RES。当用户使用 Kad 网络进行搜索并且下载文件时，首先查找希望搜索的文件的关键字。由于使用的是同样的散列算法，只需要找到 ID 值和计算出来的散列值距离最近的联系人信息后，向它们发送搜索特定关键字的请求即可。在得到返回信息后，搜索人就知道了该关键字对应的文件，然后从这些文件中挑选自己需要的文件。这时，启动针对文件的搜索，找到 ID 值和计算出来的文件内容的散列值距离最近的联系人的信息，向它们发送搜索特定文件的请求。在得到该文件的文件源信息后，eMule 按照这些信息去连接对应的客户端，使用传统的 eMule 协议去和这些客户端协商下载文件。

在 eMule 的实现中，还有一个 Kademlia2 的协议，其原理和 Kademlia 协议是一样的，只有协议代码和具体的消息格式不一样，其在具体的消息中包含了比 Kademlia 协议更加丰富的一些信息。此外，还有一个对于已发出的请求的追踪的特性，该信息保存在一个以

www.ckook.com

TrackPackets_Struct 类型为元素的列表中。这里面详细地记录了什么时间曾经对哪个 IP 发出过哪种 opcode 所对应的请求。其目的是为了防止针对 DHT 算法的一种路由污染攻击。因为在搜索联系人的时候，被搜索到的一些联系人的信息，eMule 会试图将它先加入到本地的联系人信息列表中。这样如果有人想恶意攻击某个 eMule 客户端，它只需要向该客户端发送 KADEMLIA_RES 消息，并且在消息的内容中包含大量的虚假联系人信息，该客户端的 Kad 网络就基本瘫痪了。通过 TrackPackets_Struct 列表的这个特性，eMule 客户端就可以过滤掉那种没有发送请求消息就收到的回应消息，从而避免了这种攻击带来的影响。

6．Kad 网络的搜索

Kad 网络的搜索是由 CSearch 类和 CSearchManager 类来完成的。其中，CSearch 类的对象表示了一个具体的搜索任务（搜索文件源、搜索关键字或者发布任务等），它包括了一个搜索任务从创建到结束的全部过程。而 CSearchManager 类通过一个以 CSearch 类的对象指针为元素的 CMap 对象管理着所有的搜索任务，其键值是每个搜索任务所对应的目标 ID。通过该目标 ID，使用 Map 映射，可以快速地查找到每一个具体的搜索。CSearchManager 的所有方法都是静态的，使得全局只有一个唯一的对象。

CSearch 的对象只能由 CSearchManager 创建，通过它的 PrepareFindKeywords、PrepareLookup 和 FindNode3 个成员函数。在创建的过程中，CSearch 类需要对创建的类型 m_uType 进行赋值（NODE、NODECOMPLETE、FILE、KEYWORD、NOTES、STOREFILE、STOREKEYWORD、STORENOTES、FINDBUDDY 或者 FINDSOURCE），明确其任务的类型。CSearch 主要提供了以下几个方法支持查询，如表 7-6 所示。此外，CSearch 类能够处理各种搜索结果，然后向调用它的代码返回处理好的结果。需要注意的是，Go、SendFindValue 和 JumpStart 方法都是 CSearch 类的私有方法，只有 CSearch 类的友元类 CSearchManager 可以访问。

表 7-6　CSearch 类的主要方法

方 法 名	方 法 描 述
Go	启动整个搜索过程，它会开始第一次从本地的联系人列表中寻找候选的联系人，然后发动搜索
SendFindValue	向某个联系人发送一个搜索拥有某 ID 的联系人的信息的请求
JumpStart	在搜索进行到某个阶段时，如果得到了一些中间结果，开始进行下一步的行动，下一步的行动可能是 SendFindValue，也可能认为搜索到的联系人离目标已经足够近了，而开始实质性的请求
StorePacket	实质性的请求，例如在一个以发布文件源为任务的 CSearch 中，它会向目标联系人发送 KADEMLIA2_PUBLISH_SOURCE_REQ（如果不支持 Kademlia2，那么是 KADEMLIA_PUBLISH_REQ）

Kad 网络的其他部分代码直接访问 CSearchManager 类，通过 CSearchManager 类的接口间接地使用 CSearch 类的方法。此外，CSearchManager 类对外提供创建各种新的搜索任务的接口，作用类似于设计模式中的 Factory，其他部分的代码只需要说明需要开始一个什么样的搜索任务即可，CSearchManager 来完成相应的创建 CSearch 的任务。

7.4　eMule 程序代码分析

本节将针对 eMule 庞大程序中的文件分块处理模块的代码进行详细的代码分析和程序实

www.ckook.com

例，希望可以帮助读者深入了解 eMule 的源代码。该部分的代码主要包括 CAICHHash 类、CAICHHashAlgo 类、CAICHHashTree 类、CAICHUntrustedHash 类、CAICHRequestedData 类和 CAICHHashSet 类，实现在 SHAHashSet.h 和 SHAHashSet.cpp 两个文件中。**eMule 的源代码可以在 http://www.emule.org.cn/download 上找到。**

其中 CAICHHashAlgo 类是一个算法的接口类，它定义了散列算法的接口，包括重置、添加数据、算法完成、获取散列等，其代码如下所示。

```
////////////////////////////////////////////////////////////////////////////////
///CAICHHashAlgo
class CAICHHashAlgo
{
public:
    virtual void   Reset() = 0;
    virtual void   Add(LPCVOID pData, DWORD nLength) = 0;
    virtual void   Finish(CAICHHash& Hash) = 0;
    virtual void   GetHash(CAICHHash& Hash) = 0;
};
```

CAICHUntrustedHash 类和 CAICHRequestedData 类是两个数据结构，分别表示不信任的散列值和被请求的数据，其代码如下所示。

```
////////////////////////////////////////////////////////////////////////////////
///CAICHUntrustedHashs
class CAICHUntrustedHash {
public:
    CAICHUntrustedHash&   operator=(const CAICHUntrustedHash& k1) {
        m_adwIpsSigning.Copy(k1.m_adwIpsSigning); m_Hash = k1.m_Hash ;
        return *this;
    }
    bool    AddSigningIP(uint32 dwIP);

    CAICHHash               m_Hash;
    CArray<uint32, uint32>  m_adwIpsSigning;
};

////////////////////////////////////////////////////////////////////////////////
///CAICHUntrustedHashs
class CAICHRequestedData {
public:
    CAICHRequestedData()  {m_nPart = 0; m_pPartFile = NULL; m_pClient= NULL;}
    CAICHRequestedData& operator=(const CAICHRequestedData& k1) {
        m_nPart = k1.m_nPart;
        m_pPartFile = k1.m_pPartFile;
        m_pClient = k1.m_pClient;
        return *this;
```


www.ckook.com

```
    }

    uint16              m_nPart;
    CPartFile*          m_pPartFile;
    CUpDownClient*      m_pClient;
};
```

CAICHHash 类，负责存储文件块散列值的数据结构。它提供了赋值、比较等运算符重载操作，以及数据写入和读出等操作，其代码如下所示。

```
////////////////////////////////////////////////////////////////////////////////
///CAICHHash
class CAICHHash
{
public:
    ~CAICHHash() {;}
    CAICHHash() { ZeroMemory(m_abyBuffer, HASHSIZE); }
    CAICHHash(CFileDataIO* file) { Read(file); }
    CAICHHash(uchar* data) { Read(data); }
    CAICHHash(const CAICHHash& k1) { *this = k1; }
    CAICHHash&      operator=(const CAICHHash& k1) { memcpy(m_abyBuffer, k1.m_abyBuffer,
HASHSIZE); return *this; }
    friend bool operator==(const CAICHHash& k1,const CAICHHash& k2) { return
memcmp(k1.m_abyBuffer, k2.m_abyBuffer, HASHSIZE) == 0;}
    friend bool operator!=(const CAICHHash& k1,const CAICHHash& k2) { return !(k1 == k2); }
    void        Read(CFileDataIO* file);                // 从文件中读入散列值
    void        Write(CFileDataIO* file) const;         // 向文件写入散列值
    void        Read(uchar* data) { memcpy(m_abyBuffer, data, HASHSIZE); }
                                                        // 读入散列值
    CString     GetString() const;                      // 获取基于 BASE32 编码的散列值
    uchar*      GetRawHash() { return m_abyBuffer; }
                                                        // 获取原始散列值
    static int  GetHashSize() { return HASHSIZE;}       // 获取散列值大小
private:
    uchar m_abyBuffer[HASHSIZE];
};
```

CAICHHashTree 类实现了一棵散列树，它的成员变量如下所示。

```
CAICHHash m_Hash;               // 散列值
uint64    m_nDataSize;          // 该散列覆盖的数据大小
uint64    m_nBaseSize;          // 最小散列的块大小
bool      m_bIsLeftBranch;      // 树的左右分支标识
bool      m_bHashValid;         // 散列的合法性标识
```

CAICHHashTree 类对外提供的公有函数如下所示。

```
////////////////////////////////////////////////////////////////////////////////
///CAICHHashTree
```

www.docin.com

```
CAICHHashTree::CAICHHashTree(uint64 nDataSize, bool bLeftBranch, uint64 nBaseSize){
	m_nDataSize = nDataSize;
	m_nBaseSize = nBaseSize;
	m_bIsLeftBranch = bLeftBranch;
	m_pLeftTree = NULL;
	m_pRightTree = NULL;
	m_bHashValid = false;
}

CAICHHashTree::~CAICHHashTree(){
	delete m_pLeftTree;
	delete m_pRightTree;
}

// 设置块的散列值
void CAICHHashTree::SetBlockHash(uint64 nSize, uint64 nStartPos, CAICHHashAlgo* pHashAlg){
	ASSERT ( nSize <= EMBLOCKSIZE );
	CAICHHashTree* pToInsert = FindHash(nStartPos, nSize);
	if (pToInsert == NULL){ // 健全性
		ASSERT ( false );
		theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Critical Error: Failed to Insert
SHA-HashBlock, FindHash() failed!"));
		return;
	}

	// 健全性
	if (pToInsert->m_nBaseSize != EMBLOCKSIZE || pToInsert->m_nDataSize != nSize){
		ASSERT ( false );
		theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Critical Error: Logical error on
values in SetBlockHashFromData"));
		return;
	}

	pHashAlg->Finish(pToInsert->m_Hash);
	pToInsert->m_bHashValid = true;
	//DEBUG_ONLY(theApp.QueueDebugLogLine(/*DLP_VERYLOW,*/ false, _T("Set ShaHash for
block %u - %u (%u Bytes) to %s"), nStartPos, nStartPos + nSize, nSize, pToInsert->m_Hash.GetString()) );

}

// 使用递归，计算丢失的散列值，并且可以覆盖存在的散列值
bool CAICHHashTree::ReCalculateHash(CAICHHashAlgo* hashalg, bool bDontReplace){
```

www.docin.com

www.ckook.com

```
        ASSERT ( !( (m_pLeftTree != NULL) ^ (m_pRightTree != NULL)) );
        if (m_pLeftTree && m_pRightTree){
            if ( !m_pLeftTree->ReCalculateHash(hashalg, bDontReplace) || !m_pRightTree-> ReCalculate
Hash(hashalg, bDontReplace) )
                return false;
            if (bDontReplace && m_bHashValid)
                return true;
            if (m_pRightTree->m_bHashValid && m_pLeftTree->m_bHashValid){
                hashalg->Reset();
                hashalg->Add(m_pLeftTree->m_Hash.GetRawHash(), HASHSIZE);
                hashalg->Add(m_pRightTree->m_Hash.GetRawHash(), HASHSIZE);
                hashalg->Finish(m_Hash);
                m_bHashValid = true;
                return true;
            }
            else
                return m_bHashValid;
        }
        else
            return true;
    }

    // 检验散列树，并且可以删除错误的分支
    bool CAICHHashTree::VerifyHashTree(CAICHHashAlgo* hashalg, bool bDeleteBadTrees){
        if (!m_bHashValid){
            ASSERT ( false );
            if (bDeleteBadTrees){
                delete m_pLeftTree;
                m_pLeftTree = NULL;
                delete m_pRightTree;
                m_pRightTree = NULL;
            }
            theApp.QueueDebugLogLine(/*DLP_HIGH,*/ false, _T("VerifyHashTree - No masterhash available"));
            return false;
        }

        // 计算丢失的散列，不覆盖任何数据
        if (m_pLeftTree && !m_pLeftTree->m_bHashValid)
            m_pLeftTree->ReCalculateHash(hashalg, true);
        if (m_pRightTree && !m_pRightTree->m_bHashValid)
            m_pRightTree->ReCalculateHash(hashalg, true);

        if ((m_pRightTree && m_pRightTree->m_bHashValid) ^ (m_pLeftTree && m_pLeftTree->m_bHashValid)){
```

www.ckook.com

```
            // 一个分支不能被检验
            if (bDeleteBadTrees){
                delete m_pLeftTree;
                m_pLeftTree = NULL;
                delete m_pRightTree;
                m_pRightTree = NULL;
            }
        theApp.QueueDebugLogLine(/*DLP_HIGH,*/ false, _T("VerifyHashSet failed - Hashtree incomplete"));
        return false;
    }
    if ((m_pRightTree && m_pRightTree->m_bHashValid) && (m_pLeftTree && m_pLeftTree->
m_bHashValid)){
            // 检验本地散列值与子节点的散列值是否匹配
            CAICHHash CmpHash;
            hashalg->Reset();
            hashalg->Add(m_pLeftTree->m_Hash.GetRawHash(), HASHSIZE);
            hashalg->Add(m_pRightTree->m_Hash.GetRawHash(), HASHSIZE);
            hashalg->Finish(CmpHash);

            if (m_Hash != CmpHash){
                if (bDeleteBadTrees){
                    delete m_pLeftTree;
                    m_pLeftTree = NULL;
                    delete m_pRightTree;
                    m_pRightTree = NULL;
                }
                return false;
            }
            return m_pLeftTree->VerifyHashTree(hashalg, bDeleteBadTrees) && m_pRightTree->Verify
HashTree(hashalg, bDeleteBadTrees);
        }
        else
            // 已经是分支中最后的散列值
            return true;
    }

    // 寻找散列值
    CAICHHashTree* CAICHHashTree::FindHash(uint64 nStartPos, uint64 nSize) {
        uint8 buffer = 0;
        return FindHash(nStartPos, nSize, &buffer);
    }
```

CAICHHashTree 类内部的私有函数如下所示。

www.ckook.com

```
// 递归函数，一级一级往下寻找散列值
CAICHHashTree* CAICHHashTree::FindHash(uint64 nStartPos, uint64 nSize, uint8* nLevel){
    (*nLevel)++;
    if (*nLevel > 22){ // 健全性
        ASSERT( false );
        return false;
    }
    if (nStartPos + nSize > m_nDataSize){ // 健全性
        ASSERT ( false );
        return NULL;
    }
    if (nSize > m_nDataSize){ // 健全性
        ASSERT ( false );
        return NULL;
    }

    if (nStartPos == 0 && nSize == m_nDataSize){
        // 该散列是目标值
        return this;
    }
    else if (m_nDataSize <= m_nBaseSize){ // 健全性
        // 已经是最低层，不能再往下搜索了
        ASSERT( false );
        return NULL;
    }
    else{
        uint64 nBlocks = m_nDataSize / m_nBaseSize + ((m_nDataSize % m_nBaseSize != 0 )? 1:0);
        uint64 nLeft = ( ((m_bIsLeftBranch) ? nBlocks+1:nBlocks) / 2)* m_nBaseSize;
        uint64 nRight = m_nDataSize - nLeft;
        if (nStartPos < nLeft){
            if (nStartPos + nSize > nLeft){ // 健全性
                ASSERT ( false );
                return NULL;
            }
            if (m_pLeftTree == NULL)
                m_pLeftTree = new CAICHHashTree(nLeft, true, (nLeft <= PARTSIZE) ?
EMBLOCKSIZE : PARTSIZE);
            else{
                ASSERT( m_pLeftTree->m_nDataSize == nLeft );
            }
            return m_pLeftTree->FindHash(nStartPos, nSize, nLevel);
        }
        else{
```

www.ckook.com

```
                nStartPos -= nLeft;
                if (nStartPos + nSize > nRight){ // 健全性
                    ASSERT ( false );
                    return NULL;
                }
                if (m_pRightTree == NULL)
                    m_pRightTree = new CAICHHashTree(nRight, false, (nRight <= PARTSIZE) ?
EMBLOCKSIZE : PARTSIZE);
                else{
                    ASSERT( m_pRightTree->m_nDataSize = = nRight );
                }
                return m_pRightTree->FindHash(nStartPos, nSize, nLevel);
            }
        }
    }

    // 创建恢复的数据块
    bool CAICHHashTree::CreatePartRecoveryData(uint64 nStartPos, uint64 nSize, CFileDataIO*
fileDataOut, uint32 wHashIdent, bool b32BitIdent){
        if (nStartPos + nSize > m_nDataSize){ // 健全性
            ASSERT ( false );
            return false;
        }
        if (nSize > m_nDataSize){ // 健全性
            ASSERT ( false );
            return false;
        }

        if (nStartPos == 0 && nSize == m_nDataSize){
            // 该数据块是目标数据块，将所有的数据分组写入该数据块中
            // 该层的散列标识将被调整
            return WriteLowestLevelHashs(fileDataOut, wHashIdent, false, b32BitIdent);
        }
        else if (m_nDataSize <= m_nBaseSize){ // 健全性
            // 已经是最低层，不能再往下搜索了
            ASSERT( false );
            return false;
        }
        else{
            wHashIdent <<= 1;
            wHashIdent |= (m_bIsLeftBranch) ? 1: 0;

            uint64 nBlocks = m_nDataSize / m_nBaseSize + ((m_nDataSize % m_nBaseSize != 0 )? 1:0);
```

www.dcgn.com

www.ckook.com

```
        uint64 nLeft = ( ((m_bIsLeftBranch) ? nBlocks+1:nBlocks) / 2)* m_nBaseSize;
        uint64 nRight = m_nDataSize - nLeft;
        if (m_pLeftTree == NULL || m_pRightTree == NULL){
            ASSERT( false );
            return false;
        }
        if (nStartPos < nLeft){
            if (nStartPos + nSize > nLeft || !m_pRightTree->m_bHashValid){ // 健全性
                ASSERT ( false );
                return false;
            }
            m_pRightTree->WriteHash(fileDataOut, wHashIdent, b32BitIdent);
            return m_pLeftTree->CreatePartRecoveryData(nStartPos, nSize, fileDataOut, wHashIdent,
b32BitIdent);
        }
        else{
            nStartPos -= nLeft;
            if (nStartPos + nSize > nRight || !m_pLeftTree->m_bHashValid){ // 健全性
                ASSERT ( false );
                return false;
            }
            m_pLeftTree->WriteHash(fileDataOut, wHashIdent, b32BitIdent);
            return    m_pRightTree->CreatePartRecoveryData(nStartPos,    nSize,    fileDataOut,
wHashIdent, b32BitIdent);
        }
    }
}

// 向文件中写入散列值
void CAICHHashTree::WriteHash(CFileDataIO* fileDataOut, uint32 wHashIdent, bool b32BitIdent) const{
    ASSERT( m_bHashValid );
    wHashIdent <<= 1;
    wHashIdent |= (m_bIsLeftBranch) ? 1: 0;
    if (!b32BitIdent){
        ASSERT( wHashIdent <= 0xFFFF );
        fileDataOut->WriteUInt16((uint16)wHashIdent);
    }
    else
        fileDataOut->WriteUInt32(wHashIdent);
    m_Hash.Write(fileDataOut);
}

// 将最底层的散列值写入文件，按照从左向右的顺序，没有使用标识
```

www.docin.com

www.ckook.com

```
bool CAICHHashTree::WriteLowestLevelHashs(CFileDataIO* fileDataOut, uint32 wHashIdent, bool
bNoIdent, bool b32BitIdent) const{
    wHashIdent <<= 1;
    wHashIdent |= (m_bIsLeftBranch) ? 1: 0;
    if (m_pLeftTree == NULL && m_pRightTree == NULL){
        if (m_nDataSize <= m_nBaseSize && m_bHashValid ){
            if (!bNoIdent && !b32BitIdent){
                ASSERT( wHashIdent <= 0xFFFF );
                fileDataOut->WriteUInt16((uint16)wHashIdent);
            }
            else if (!bNoIdent && b32BitIdent)
                fileDataOut->WriteUInt32(wHashIdent);
            m_Hash.Write(fileDataOut);
            //theApp.AddDebugLogLine(false,_T("%s"),m_Hash.GetString(), wHashIdent, this);
            return true;
        }
        else{
            ASSERT( false );
            return false;
        }
    }
    else if (m_pLeftTree == NULL || m_pRightTree == NULL){
        ASSERT( false );
        return false;
    }
    else{
        return m_pLeftTree->WriteLowestLevelHashs(fileDataOut, wHashIdent, bNoIdent,
b32BitIdent)
            && m_pRightTree->WriteLowestLevelHashs(fileDataOut, wHashIdent, bNoIdent, b32BitIdent);
    }
}

// 通过文件输入的数据，恢复最底层所有的散列值，散列值被假定为从左向右的顺序，没有使用标识
bool CAICHHashTree::LoadLowestLevelHashs(CFileDataIO* fileInput){
    if (m_nDataSize <= m_nBaseSize){ // 健全性
        // 已经到达最底层，读入散列
        m_Hash.Read(fileInput);
        //theApp.AddDebugLogLine(false,m_Hash.GetString());
        m_bHashValid = true;
        return true;
    }
    else{
        uint64 nBlocks = m_nDataSize / m_nBaseSize + ((m_nDataSize % m_nBaseSize != 0 )? 1:0);
```


www.ckook.com

```
            uint64 nLeft = ( ((m_bIsLeftBranch) ? nBlocks+1:nBlocks) / 2)* m_nBaseSize;
            uint64 nRight = m_nDataSize - nLeft;
            if (m_pLeftTree == NULL)
              m_pLeftTree = new CAICHHashTree(nLeft, true, (nLeft <= PARTSIZE) ? EMBLOCKSIZE :
PARTSIZE);
            else{
                ASSERT( m_pLeftTree->m_nDataSize == nLeft );
            }
            if (m_pRightTree == NULL)
                m_pRightTree = new CAICHHashTree(nRight, false, (nRight <= PARTSIZE) ?
EMBLOCKSIZE : PARTSIZE);
            else{
                ASSERT( m_pRightTree->m_nDataSize == nRight );
            }
            return m_pLeftTree->LoadLowestLevelHashs(fileInput)
                    && m_pRightTree->LoadLowestLevelHashs(fileInput);
        }
    }

    // 将文件中的散列数据写入散列标识指定的位置的散列值
    bool CAICHHashTree::SetHash(CFileDataIO* fileInput, uint32 wHashIdent, sint8 nLevel, bool
bAllowOverwrite){
        if (nLevel == (-1)){
            // 首先确定需要走多少层
            uint8 i;
            for (i = 0; i != 32 && (wHashIdent & 0x80000000) == 0; i++){
                wHashIdent <<= 1;
            }
            if (i > 31){
                theApp.QueueDebugLogLine(/*DLP_HIGH,*/  false,  _T("CAICHHashTree::SetHash  -
found invalid HashIdent (0)"));
                return false;
            }
            else{
                nLevel = 31 - i;
            }
        }
        if (nLevel == 0){
            // 确定到目标层
            if (m_bHashValid && !bAllowOverwrite){
                // 不允许覆盖该散列值，通过读取散列值移动文件指针
                CAICHHash(file);
                return true;
```

www.ckook.com

```
            }
            m_Hash.Read(fileInput);
            m_bHashValid = true;
            return true;
        }
        else if (m_nDataSize <= m_nBaseSize){ // 健全性
            // 已经是最低层，不能再往下了
            ASSERT( false );
            return false;
        }
        else{
            // 调整标识以确定指向下一个节点的路径
            wHashIdent <<= 1;
            nLevel--;
            uint64 nBlocks = m_nDataSize / m_nBaseSize + ((m_nDataSize % m_nBaseSize != 0 )? 1:0);
            uint64 nLeft = ( ((m_bIsLeftBranch) ? nBlocks+1:nBlocks) / 2)* m_nBaseSize;
            uint64 nRight = m_nDataSize - nLeft;
            if ((wHashIdent & 0x80000000) > 0){
                if (m_pLeftTree == NULL)
                    m_pLeftTree = new CAICHHashTree(nLeft, true, (nLeft <= PARTSIZE) ?
EMBLOCKSIZE : PARTSIZE);
                else{
                    ASSERT( m_pLeftTree->m_nDataSize == nLeft );
                }
                return m_pLeftTree->SetHash(fileInput, wHashIdent, nLevel);
            }
            else{
                if (m_pRightTree == NULL)
                    m_pRightTree = new CAICHHashTree(nRight, false, (nRight <= PARTSIZE) ?
EMBLOCKSIZE : PARTSIZE);
                else{
                    ASSERT( m_pRightTree->m_nDataSize == nRight );
                }
                return m_pRightTree->SetHash(fileInput, wHashIdent, nLevel);
            }
        }
    }
```

CAICHHashSet 类是 CKownFile 类的一个成员变量，CKownFile 类通过它执行文件分块操作。该类的成员变量如下所示。

```
    public:
        CAICHHashTree           m_pHashTree;        // 散列树
        static CList<CAICHRequestedData>            m_liRequestedData;
                                                    // 请求数据链表
```


www.ckook.com

```
    static CMutex                          m_mutKnown2File;
                                           // 互斥信号量
private:
    CKnownFile*          m_pOwner;         // CKnownFile 类的对象指针
    EAICHStatus      m_eStatus;            // 状态值
    CArray<CAICHUntrustedHash> m_aUntrustedHashs;
                                           // 不信任的散列数组
```

CAICHHashSet 类对外提供的公有函数如下所示。

```
////////////////////////////////////////////////////////////////////////////////
///CAICHHashSet
CAICHHashSet::CAICHHashSet(CKnownFile* pOwner) : m_pHashTree(0, true, PARTSIZE)
{
    m_eStatus = AICH_EMPTY;
    m_pOwner = pOwner;
}

CAICHHashSet::~CAICHHashSet(void)
{
    FreeHashSet();
}

    // 创建恢复的数据块
    bool CAICHHashSet::CreatePartRecoveryData(uint64 nPartStartPos, CFileDataIO* fileDataOut, bool
bDbgDontLoad){
        ASSERT( m_pOwner );
        if (m_pOwner->IsPartFile() || m_eStatus != AICH_HASHSETCOMPLETE){
            ASSERT( false );
            return false;
        }
        if (m_pHashTree.m_nDataSize <= EMBLOCKSIZE){
            ASSERT( false );
            return false;
        }
        if (!bDbgDontLoad){
            if (!LoadHashSet()){
                theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Created RecoveryData
error: failed to load hashset (file: %s)"), m_pOwner->GetFileName() );
                SetStatus(AICH_ERROR);
                return false;
            }
        }
        bool bResult;
        uint8 nLevel = 0;
```

www.ckook.com

```
        uint32 nPartSize = (uint32)min(PARTSIZE, (uint64)m_pOwner->GetFileSize()-nPartStartPos);
        m_pHashTree.FindHash(nPartStartPos, nPartSize,&nLevel);
        uint16 nHashsToWrite = (uint16)((nLevel-1) + nPartSize/EMBLOCKSIZE + ((nPartSize %
EMBLOCKSIZE != 0 )? 1:0));
        const bool bUse32BitIdentifier = m_pOwner->IsLargeFile();

        if (bUse32BitIdentifier)
            fileDataOut->WriteUInt16(0); // 没有可写入的 16bit 散列
        fileDataOut->WriteUInt16(nHashsToWrite);
        uint32 nCheckFilePos = (UINT)fileDataOut->GetPosition();
         if (m_pHashTree.CreatePartRecoveryData(nPartStartPos, nPartSize, fileDataOut, 0, bUse32BitIdentifier)){
            if (nHashsToWrite*(HASHSIZE+(bUse32BitIdentifier? 4:2)) != fileDataOut->GetPosition() -
nCheckFilePos){
                ASSERT( false );
                theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Created RecoveryData
has wrong length (file: %s)"), m_pOwner->GetFileName() );
                bResult = false;
                SetStatus(AICH_ERROR);
            }
            else
                bResult = true;
        }
        else{
            theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to create RecoveryData
for %s"), m_pOwner->GetFileName() );
            bResult = false;
            SetStatus(AICH_ERROR);
        }
        if (!bUse32BitIdentifier)
            fileDataOut->WriteUInt16(0); // 没有可写入的 32bit 的散列

        if (!bDbgDontLoad){
            FreeHashSet();
        }
        return bResult;
    }

    // 读入恢复的数据
    bool CAICHHashSet::ReadRecoveryData(uint64 nPartStartPos, CSafeMemFile* fileDataIn){
        if (/*TODO !m_pOwner->IsPartFile() ||*/ !(m_eStatus == AICH_VERIFIED || m_eStatus ==
AICH_TRUSTED) ){
            ASSERT( false );
            return false;
```

www.ccin.com

www.ckook.com

```
}
/* V2 AICH Hash Packet:
    <count1 uint16>                                  16bit-hashs-to-read
    (<identifier uint16><hash HASHSIZE>)[count1]     AICH hashs
    <count2 uint16>                                  32bit-hashs-to-read
    (<identifier uint32><hash HASHSIZE>)[count2]     AICH hashs
*/

        // 在这里只检查恢复的数据是否是正确的散列数量
        // 所有的散列将根据散列标识写入散列树

        uint8 nLevel = 0;
        uint32 nPartSize = (uint32)min(PARTSIZE, (uint64)m_pOwner->GetFileSize()-nPartStartPos);
        m_pHashTree.FindHash(nPartStartPos, nPartSize,&nLevel);
        uint16 nHashsToRead = (uint16)((nLevel-1) + nPartSize/EMBLOCKSIZE + ((nPartSize %
EMBLOCKSIZE != 0 )? 1:0));

        // 读入 16bit 标识的散列
        uint16 nHashsAvailable = fileDataIn->ReadUInt16();
        if (fileDataIn->GetLength()-fileDataIn->GetPosition() < nHashsToRead*(HASHSIZE+2) ||
(nHashsToRead != nHashsAvailable && nHashsAvailable != 0)){
            // 这个检查是多余的，CSafememfile 也会捕获该错误
            theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read RecoveryData
for %s - Received datasize/amounts of hashs was invalid (1)"), m_pOwner->GetFileName() );
            return false;
        }
        DEBUG_ONLY( theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("read RecoveryData for
%s - Received packet with  %u 16bit hash identifiers)"), m_pOwner->GetFileName(), nHashsAvailable ) );
        for (uint32 i = 0; i != nHashsAvailable; i++){
            uint16 wHashIdent = fileDataIn->ReadUInt16();
            if (wHashIdent == 1 /*never allow masterhash to be overwritten*/
                || !m_pHashTree.SetHash(fileDataIn, wHashIdent,(-1), false))
            {
                theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read Recovery
Data for %s - Error when trying to read hash into tree (1)"), m_pOwner->GetFileName() );
                VerifyHashTree(true); // 去掉已写入的非法的散列
                return false;
            }
        }

        // 读入 32bit 标识的散列
        if (nHashsAvailable == 0 && fileDataIn->GetLength() - fileDataIn->GetPosition() >= 2){
            nHashsAvailable = fileDataIn->ReadUInt16();
```

www.ckook.com

```
            if (fileDataIn->GetLength()-fileDataIn->GetPosition() < nHashsToRead*(HASHSIZE+4) ||
(nHashsToRead != nHashsAvailable && nHashsAvailable != 0)){
                // 这个检查是多余的，CSafememfile 也会捕获该错误
                theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read
RecoveryData for %s - Received datasize/amounts of hashs was invalid (2)"), m_pOwner->GetFileName() );
                return false;
            }
            DEBUG_ONLY( theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("read RecoveryData
for %s - Received packet with %u 32bit hash identifiers)"), m_pOwner->GetFileName(), nHashsAvailable ) );
            for (uint32 i = 0; i != nHashsToRead; i++){
                uint32 wHashIdent = fileDataIn->ReadUInt32();
                if (wHashIdent == 1 /*never allow masterhash to be overwritten*/
                    || wHashIdent > 0x400000
                    || !m_pHashTree.SetHash(fileDataIn, wHashIdent,(-1), false))
                {
                    theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read
RecoveryData for %s - Error when trying to read hash into tree (2)"), m_pOwner->GetFileName() );
                    VerifyHashTree(true); // 去掉已写入的非法的散列
                    return false;
                }
            }
        }

        if (nHashsAvailable == 0){
            theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read RecoveryData
for %s - Packet didn't contained any hashs"), m_pOwner->GetFileName() );
            return false;
        }

        if (VerifyHashTree(true)){
            // 最终检查是否所需要的散列都在这里
            for (uint32 nPartPos = 0; nPartPos < nPartSize; nPartPos += EMBLOCKSIZE){
                CAICHHashTree* phtToCheck = m_pHashTree.FindHash(nPartStartPos+nPartPos,
min(EMBLOCKSIZE, nPartSize-nPartPos));
                if (phtToCheck == NULL || !phtToCheck->m_bHashValid){
                    theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read
RecoveryData for %s - Error while verifying presence of all lowest level hashs"), m_pOwner->GetFileName() );
                    return false;
                }
            }
            // 全部完成
            return true;
        }
```


www.ckook.com

```
        else{
            theApp.QueueDebugLogLine(/*DLP_VERYHIGH,*/ false, _T("Failed to read RecoveryData for %s - Verifying received hashtree failed"), m_pOwner->GetFileName() );
            return false;
        }
    }

    // 该函数只允许在成功计算散列设置之后才能被调用
    // 保存后将删除散列设置以释放内存
    bool CAICHHashSet::SaveHashSet(){
        if (m_eStatus != AICH_HASHSETCOMPLETE){
            ASSERT( false );
            return false;
        }
        if ( !m_pHashTree.m_bHashValid || m_pHashTree.m_nDataSize != m_pOwner->GetFileSize()){
            ASSERT( false );
            return false;
        }
        CSingleLock lockKnown2Met(&m_mutKnown2File, false);
        if (!lockKnown2Met.Lock(5000)){
            return false;
        }

        CString fullpath = thePrefs.GetMuleDirectory(EMULE_CONFIGDIR);
        fullpath.Append(KNOWN2_MET_FILENAME);
        CSafeFile file;
        CFileException fexp;
        if (!file.Open(fullpath, CFile::modeCreate | CFile::modeReadWrite | CFile::modeNoTruncate | CFile::osSequentialScan | CFile::typeBinary | CFile::shareDenyNone, &fexp)){
            if (fexp.m_cause != CFileException::fileNotFound){
                CString strError(_T("Failed to load ") KNOWN2_MET_FILENAME _T(" file"));
                TCHAR szError[MAX_CFEXP_ERRORMSG];
                if (fexp.GetErrorMessage(szError, ARRSIZE(szError))){
                    strError += _T(" - ");
                    strError += szError;
                }
                theApp.QueueLogLine(true, _T("%s"), strError);
            }
            return false;
        }
        try {
            //setvbuf(file.m_pStream, NULL, _IOFBF, 16384);
            uint8 header = file.ReadUInt8();
```

www.docin.com

```
if (header != KNOWN2_MET_VERSION){
    AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());
}
// 首先检查需要写入的散列设置是否已经保存好
CAICHHash CurrentHash;
uint32 nExistingSize = (UINT)file.GetLength();
uint32 nHashCount;
while (file.GetPosition() < nExistingSize){
    CurrentHash.Read(&file);
    if (m_pHashTree.m_Hash == CurrentHash){
        // 该散列设置已经可以获取，不用再保存
        return true;
    }
    nHashCount = file.ReadUInt32();
    if (file.GetPosition() + nHashCount*HASHSIZE > nExistingSize){
        AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());
    }
    // 忽略剩余的散列设置
    file.Seek(nHashCount*HASHSIZE, CFile::current);
}
// 写入散列
m_pHashTree.m_Hash.Write(&file);
nHashCount = (uint32)((PARTSIZE/EMBLOCKSIZE + ((PARTSIZE % EMBLOCKSIZE !=
0)? 1 : 0)) * (m_pHashTree.m_nDataSize/PARTSIZE));
if (m_pHashTree.m_nDataSize % PARTSIZE != 0)
    nHashCount += (uint32)((m_pHashTree.m_nDataSize % PARTSIZE)/EMBLOCKSIZE +
(((m_pHashTree.m_nDataSize % PARTSIZE) % EMBLOCKSIZE != 0)? 1 : 0));
file.WriteUInt32(nHashCount);
if (!m_pHashTree.WriteLowestLevelHashs(&file, 0, true, true)){
    file.SetLength(nExistingSize);
    theApp.QueueDebugLogLine(true, _T("Failed to save HashSet: WriteLowestLevelHashs() failed!"));
    return false;
}
if (file.GetLength() != nExistingSize + (nHashCount+1)*HASHSIZE + 4){
    file.SetLength(nExistingSize);
    theApp.QueueDebugLogLine(true, _T("Failed to save HashSet: Calculated and real size of
hashset differ!"));
    return false;
}
theApp.QueueDebugLogLine(false, _T("Successfully saved eMuleAC Hashset, %u Hashs + 1
Masterhash written"), nHashCount);
file.Flush();
file.Close();
```


www.ckook.com

```
    }
    catch(CFileException* error){
        if (error->m_cause == CFileException::endOfFile)
          theApp.QueueLogLine(true, GetResString(IDS_ERR_MET_BAD), KNOWN2_MET_FILENAME);
        else{
            TCHAR buffer[MAX_CFEXP_ERRORMSG];
            error->GetErrorMessage(buffer, ARRSIZE(buffer));
            theApp.QueueLogLine(true,GetResString(IDS_ERR_SERVERMET_UNKNOWN),buffer);
        }
        error->Delete();
        return false;
    }
    FreeHashSet();
    return true;
}

// 导入散列设置
bool CAICHHashSet::LoadHashSet(){
    if (m_eStatus != AICH_HASHSETCOMPLETE){
        ASSERT( false );
        return false;
    }
    if ( !m_pHashTree.m_bHashValid || m_pHashTree.m_nDataSize != m_pOwner->GetFileSize() ||
m_pHashTree.m_nDataSize == 0){
        ASSERT( false );
        return false;
    }
    CString fullpath = thePrefs.GetMuleDirectory(EMULE_CONFIGDIR);
    fullpath.Append(KNOWN2_MET_FILENAME);
    CSafeFile file;
    CFileException fexp;
    if (!file.Open(fullpath,CFile::modeCreate|CFile::modeRead|CFile::modeNoTruncate| CFile::os_Sequential
Scan|CFile::typeBinary|CFile::shareDenyNone, &fexp)){
        if (fexp.m_cause != CFileException::fileNotFound){
            CString strError(_T("Failed to load ") KNOWN2_MET_FILENAME _T(" file"));
            TCHAR szError[MAX_CFEXP_ERRORMSG];
            if (fexp.GetErrorMessage(szError, ARRSIZE(szError))){
                strError += _T(" - ");
                strError += szError;
            }
            theApp.QueueLogLine(true, _T("%s"), strError);
        }
        return false;
```

www.ckook.com

```
}
try {
    //setvbuf(file.m_pStream, NULL, _IOFBF, 16384);
    uint8 header = file.ReadUInt8();
    if (header != KNOWN2_MET_VERSION){
        AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());
    }
    CAICHHash CurrentHash;
    uint32 nExistingSize = (UINT)file.GetLength();
    uint32 nHashCount;
    while (file.GetPosition() < nExistingSize){
        CurrentHash.Read(&file);
        if (m_pHashTree.m_Hash == CurrentHash){
            // 建立散列设置
            uint32 nExpectedCount =     (uint32)((PARTSIZE/EMBLOCKSIZE + ((PARTSIZE
% EMBLOCKSIZE != 0)? 1 : 0)) * (m_pHashTree.m_nDataSize/PARTSIZE));
            if (m_pHashTree.m_nDataSize % PARTSIZE != 0)
                nExpectedCount  +=  (uint32)((m_pHashTree.m_nDataSize  %  PARTSIZE)/
EMBLOCKSIZE + (((m_pHashTree.m_nDataSize % PARTSIZE) % EMBLOCKSIZE != 0)? 1 : 0));
            nHashCount = file.ReadUInt32();
            if (nHashCount != nExpectedCount){
                theApp.QueueDebugLogLine(true, _T("Failed to load HashSet: Available Hashs
and expected hashcount differ!"));
                return false;
            }
            //uint32 dbgPos = file.GetPosition();
            if (!m_pHashTree.LoadLowestLevelHashs(&file)){
             theApp.QueueDebugLogLine(true, _T("Failed to load HashSet: LoadLowestLevelHashs
failed!"));
                return false;
            }
            //uint32 dbgHashRead = (file.GetPosition()-dbgPos)/HASHSIZE;
            if (!ReCalculateHash(false)){
                theApp.QueueDebugLogLine(true, _T("Failed to load HashSet: Calculating
loaded hashs failed!"));
                return false;
            }
            if (CurrentHash != m_pHashTree.m_Hash){
                theApp.QueueDebugLogLine(true, _T("Failed to load HashSet: Calculated
Masterhash differs from given Masterhash - hashset corrupt!"));
                return false;
            }
            return true;
```


www.ckook.com

```
            }
            nHashCount = file.ReadUInt32();
            if (file.GetPosition() + nHashCount*HASHSIZE > nExistingSize){
                AfxThrowFileException(CFileException::endOfFile, 0, file.GetFileName());
            }
            // 忽略剩余的散列设置
            file.Seek(nHashCount*HASHSIZE, CFile::current);
        }
        theApp.QueueDebugLogLine(true, _T("Failed to load HashSet: HashSet not found!"));
    }
    catch(CFileException* error){
        if (error->m_cause == CFileException::endOfFile)
          theApp.QueueLogLine(true, GetResString(IDS_ERR_MET_BAD), KNOWN2_MET_FILENAME);
        else{
            TCHAR buffer[MAX_CFEXP_ERRORMSG];
            error->GetErrorMessage(buffer, ARRSIZE(buffer));
           theApp.QueueLogLine(true,GetResString(IDS_ERR_SERVERMET_UNKNOWN),buffer);
        }
        error->Delete();
    }
    return false;
}

// 删除根散列以外的散列设置（并不为了保存资源而维持散列设置）
void CAICHHashSet::FreeHashSet(){
    delete m_pHashTree.m_pLeftTree;
    m_pHashTree.m_pLeftTree = NULL;
    delete m_pHashTree.m_pRightTree;
    m_pHashTree.m_pRightTree = NULL;
}

// 设置根散列
void CAICHHashSet::SetMasterHash(const CAICHHash& Hash, EAICHStatus eNewStatus){
    m_pHashTree.m_Hash = Hash;
    m_pHashTree.m_bHashValid = true;
    SetStatus(eNewStatus);
}

// 获取新的散列算法对象指针
CAICHHashAlgo* CAICHHashSet::GetNewHashAlgo(){
    return new CSHA();
}
```

www.ckook.com

```
// 重新计算散列
bool CAICHHashSet::ReCalculateHash(bool bDontReplace){
    CAICHHashAlgo* hashalg = GetNewHashAlgo();
    bool bResult = m_pHashTree.ReCalculateHash(hashalg, bDontReplace);
    delete hashalg;
    return bResult;
}

// 检验散列树
bool CAICHHashSet::VerifyHashTree(bool bDeleteBadTrees){
    CAICHHashAlgo* hashalg = GetNewHashAlgo();
    bool bResult = m_pHashTree.VerifyHashTree(hashalg, bDeleteBadTrees);
    delete hashalg;
    return bResult;
}

// 设置文件大小
void CAICHHashSet::SetFileSize(EMFileSize nSize){
    m_pHashTree.m_nDataSize = nSize;
    m_pHashTree.m_nBaseSize = (nSize <= (uint64)PARTSIZE) ? EMBLOCKSIZE : PARTSIZE;
}

// 接收到不信任的散列
void CAICHHashSet::UntrustedHashReceived(const CAICHHash& Hash, uint32 dwFromIP){
    switch(GetStatus()){
        case AICH_EMPTY:
        case AICH_UNTRUSTED:
        case AICH_TRUSTED:
            break;
        default:
            return;
    }
    bool bFound = false;
    bool bAdded = false;
    for (int i = 0; i < m_aUntrustedHashs.GetCount(); i++){
        if (m_aUntrustedHashs[i].m_Hash == Hash){
            bAdded = m_aUntrustedHashs[i].AddSigningIP(dwFromIP);
            bFound = true;
            break;
        }
    }
    if (!bFound){
        bAdded = true;
```

www.cbocn.com

www.ckook.com

```
        CAICHUntrustedHash uhToAdd;
        uhToAdd.m_Hash = Hash;
        uhToAdd.AddSigningIP(dwFromIP);
        m_aUntrustedHashs.Add(uhToAdd);
    }

    uint32 nSigningIPsTotal = 0;                // 独立的客户端发送了散列
    int nMostTrustedPos = (-1);                 // 最多客户端发送的散列
    uint32 nMostTrustedIPs = 0;
    for (uint32 i = 0; i < (uint32)m_aUntrustedHashs.GetCount(); i++){
        nSigningIPsTotal += m_aUntrustedHashs[i].m_adwIpsSigning.GetCount();
        if ((uint32)m_aUntrustedHashs[i].m_adwIpsSigning.GetCount() > nMostTrustedIPs){
            nMostTrustedIPs = m_aUntrustedHashs[i].m_adwIpsSigning.GetCount();
            nMostTrustedPos = i;
        }
    }
    if (nMostTrustedPos == (-1) || nSigningIPsTotal == 0){
        ASSERT( false );
        return;
    }
    // 检验是否相信某些散列
    if ( thePrefs.IsTrustingEveryHash() ||
        (nMostTrustedIPs >= MINUNIQUEIPS_TOTRUST && (100 * nMostTrustedIPs)/nSigningIPsTotal >=
MINPERCENTAGE_TOTRUST)){
        // 如果相信
            //theApp.QueueDebugLogLine(false, _T("AICH Hash received: %s (%sadded), We have
now %u hash from %u unique IPs. We trust the Hash %s from %u clients (%u%%). Added IP:%s, file: %s")
            //, Hash.GetString(), bAdded? _T(""):_T("not "), m_aUntrustedHashs.GetCount(),
nSigningIPsTotal, m_aUntrustedHashs[nMostTrustedPos].m_Hash.GetString()
            //, nMostTrustedIPs, (100 * nMostTrustedIPs)/nSigningIPsTotal, ipstr(dwFromIP &
0x00F0FFFF), m_pOwner->GetFileName());

        SetStatus(AICH_TRUSTED);
        if (!HasValidMasterHash() || GetMasterHash() != m_aUntrustedHashs[nMostTrustedPos].m_Hash){
            SetMasterHash(m_aUntrustedHashs[nMostTrustedPos].m_Hash, AICH_TRUSTED);
            FreeHashSet();
        }
    }
    else{
        // 如果不相信
        //theApp.QueueDebugLogLine(false, _T("AICH Hash received: %s (%sadded), We have now %u
hash from %u unique IPs. Best Hash (%s) is from %u clients (%u%%) - but we dont trust it yet. Added IP:%s, file: %s")
        //        , Hash.GetString(), bAdded? _T(""):_T("not "), m_aUntrustedHashs.GetCount(),
```

```
nSigningIPsTotal, m_aUntrustedHashs[nMostTrustedPos].m_Hash.GetString()
                    //        , nMostTrustedIPs, (100 * nMostTrustedIPs)/nSigningIPsTotal, ipstr(dwFromIP &
0x00F0FFFF), m_pOwner->GetFileName());

                    SetStatus(AICH_UNTRUSTED);
                    if (!HasValidMasterHash() || GetMasterHash() != m_aUntrustedHashs[nMostTrustedPos].m_Hash){
                        SetMasterHash(m_aUntrustedHashs[nMostTrustedPos].m_Hash, AICH_UNTRUSTED);
                        FreeHashSet();
                    }
                }
            }

            // 客户端请求失败
            void CAICHHashSet::ClientAICHRequestFailed(CUpDownClient* pClient){
                pClient->SetReqFileAICHHash(NULL);
                CAICHRequestedData data = GetAICHReqDetails(pClient);
                RemoveClientAICHRequest(pClient);
                if (data.m_pClient != pClient)
                    return;
                if(theApp.downloadqueue->IsPartFile(data.m_pPartFile)){
                    theApp.QueueDebugLogLine(false, _T("AICH Request failed, Trying to ask another client (file
%s, Part: %u,   Client%s)"), data.m_pPartFile->GetFileName(), data.m_nPart, pClient->DbgGetClientInfo());
                    data.m_pPartFile->RequestAICHRecovery(data.m_nPart);
                }
            }

            // 清除客户端请求
            void CAICHHashSet::RemoveClientAICHRequest(const CUpDownClient* pClient){
                for (POSITION pos = m_liRequestedData.GetHeadPosition();pos != 0; m_liRequestedData.GetNext(pos))
                {
                    if (m_liRequestedData.GetAt(pos).m_pClient == pClient){
                        m_liRequestedData.RemoveAt(pos);
                        return;
                    }
                }
                ASSERT( false );
            }

            // 判断客户端请求是否处理完成
            bool CAICHHashSet::IsClientRequestPending(const CPartFile* pForFile, uint16 nPart){
                for (POSITION pos = m_liRequestedData.GetHeadPosition();pos != 0; m_liRequestedData.GetNext(pos))
                {
                    if (m_liRequestedData.GetAt(pos).m_pPartFile == pForFile && m_liRequestedData.GetAt(pos).m_nPart
```

www.docin.com

www.ckook.com

```
== nPart){
                return true;
            }
        }
        return false;
    }

    // 获取客户端请求的详细信息
    CAICHRequestedData CAICHHashSet::GetAICHReqDetails(const  CUpDownClient* pClient){
        for (POSITION pos = m_liRequestedData.GetHeadPosition();pos != 0; m_liRequestedData.GetNext(pos))
        {
            if (m_liRequestedData.GetAt(pos).m_pClient == pClient){
                return m_liRequestedData.GetAt(pos);
            }
        }
        ASSERT( false );
        CAICHRequestedData empty;
        return empty;
    }

    // 判断该文件块数据是否可用
    bool CAICHHashSet::IsPartDataAvailable(uint64 nPartStartPos){
        if (!(m_eStatus == AICH_VERIFIED || m_eStatus == AICH_TRUSTED || m_eStatus ==
AICH_HASHSETCOMPLETE) ){
            ASSERT( false );
            return false;
        }
        uint32 nPartSize = (uint32)(min(PARTSIZE, (uint64)m_pOwner->GetFileSize()-nPartStartPos));
        for (uint64 nPartPos = 0; nPartPos < nPartSize; nPartPos += EMBLOCKSIZE){
            CAICHHashTree*  phtToCheck  =  m_pHashTree.FindHash(nPartStartPos+nPartPos,  min
(EMBLOCKSIZE, nPartSize-nPartPos));
            if (phtToCheck == NULL || !phtToCheck->m_bHashValid){
                return false;
            }
        }
        return true;
    }
```

7.5 eMule 系统部署

eMule 系统的 VeryCD 版在 eMule 的官方下载页面 http://www.emule.org.cn/download/上可以下载到，本小节将具体针对该软件的安装和使用等进行简单的介绍。

www.ckook.com

7.5.1 系统编译和运行

从 eMule 的官方网站上，可以下载到最新的 eMule 源代码以及它用到的一些其他的库，目前其提供的版本是“eMule 20070920 Release”。将该版本解压缩，使用 Visual Studio .NET 2003，打开 src\emule.sln，如图 7-33 所示。单击编译按钮，完成对整个工程的编译，如图 7-34 所示。单击执行按钮，运行 eMule 系统，如图 7-35 所示。

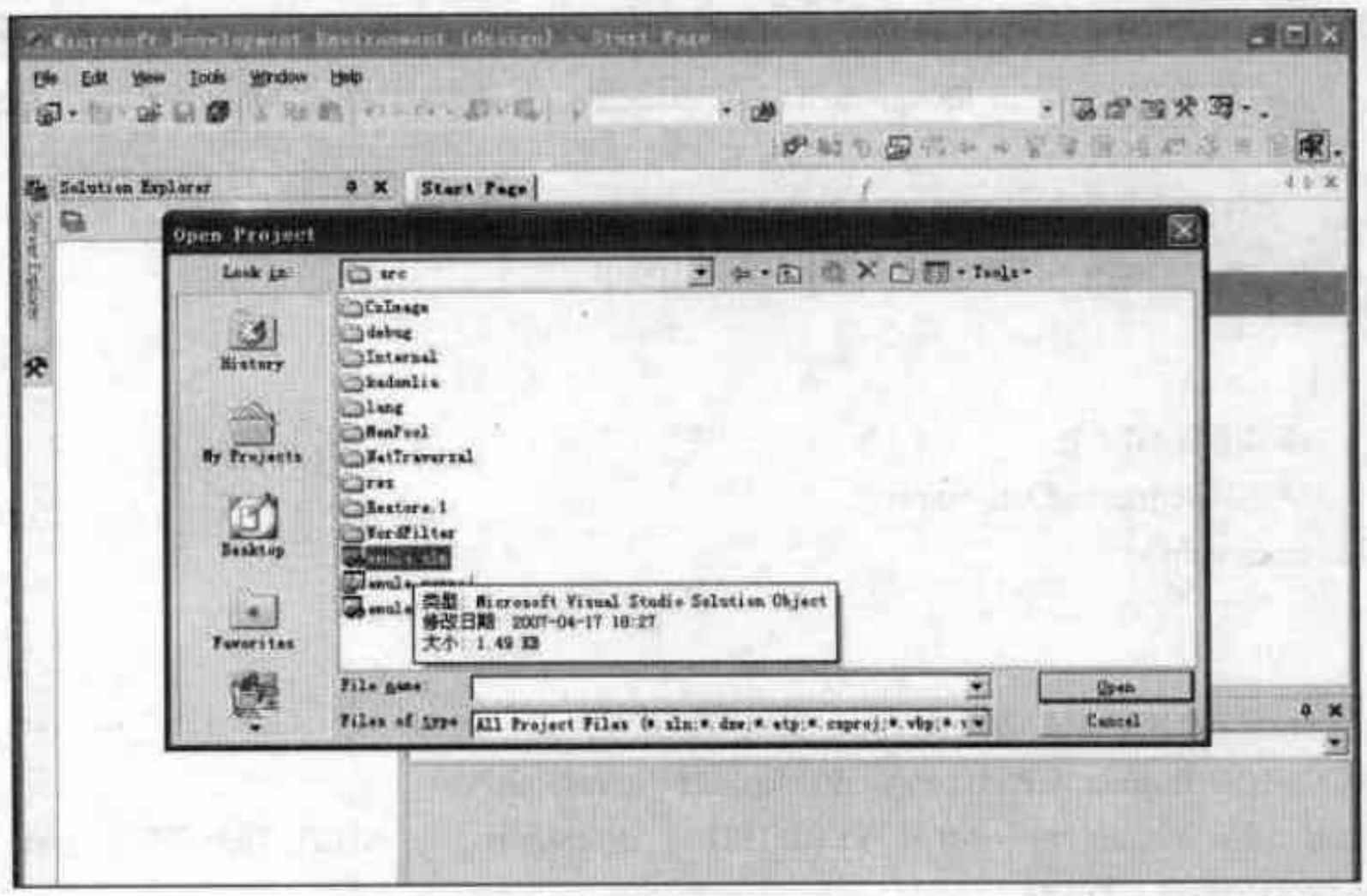


图 7-33 eMule 工程示意图

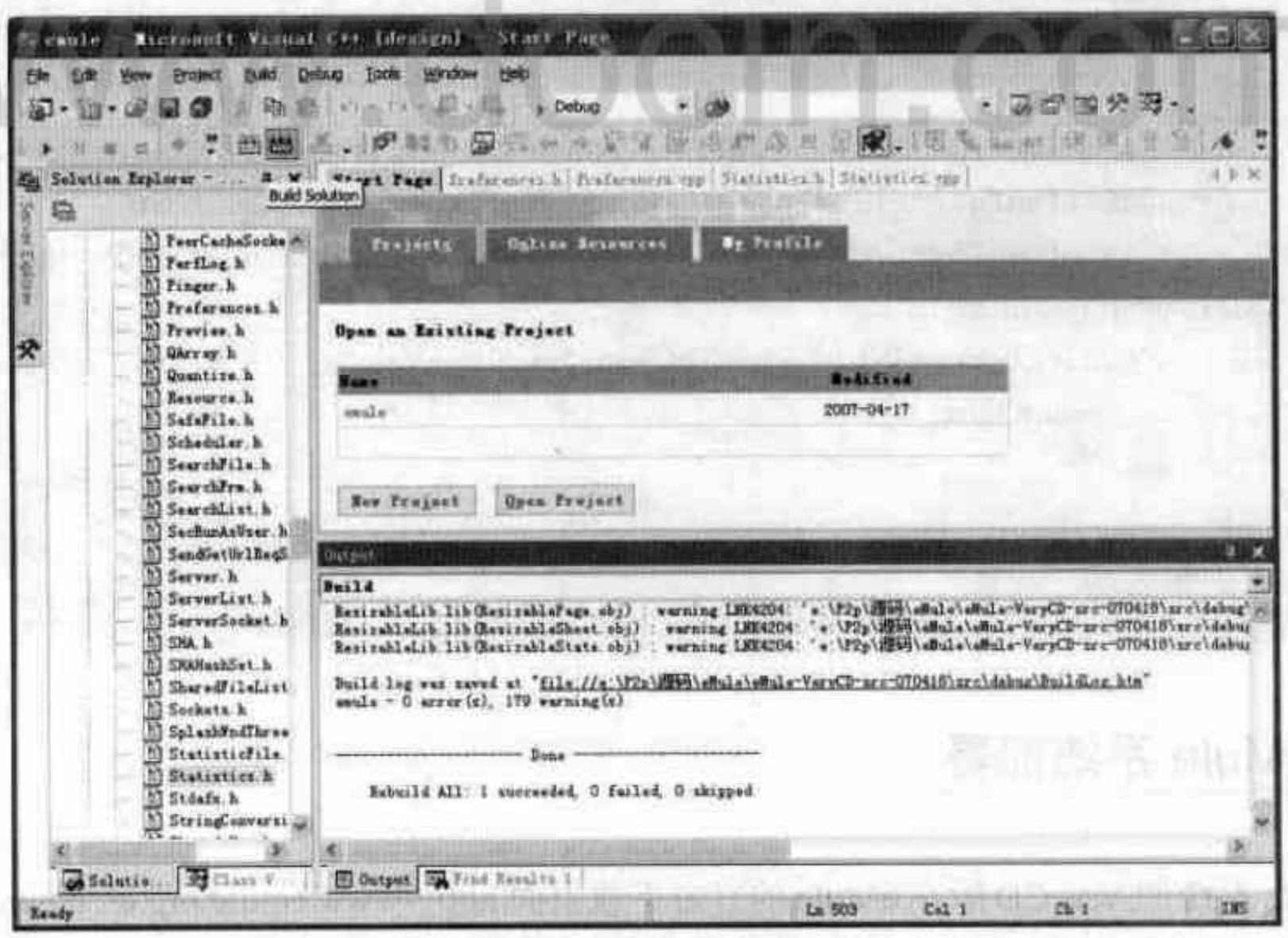


图 7-34 eMule 工程编译

www.ckook.com

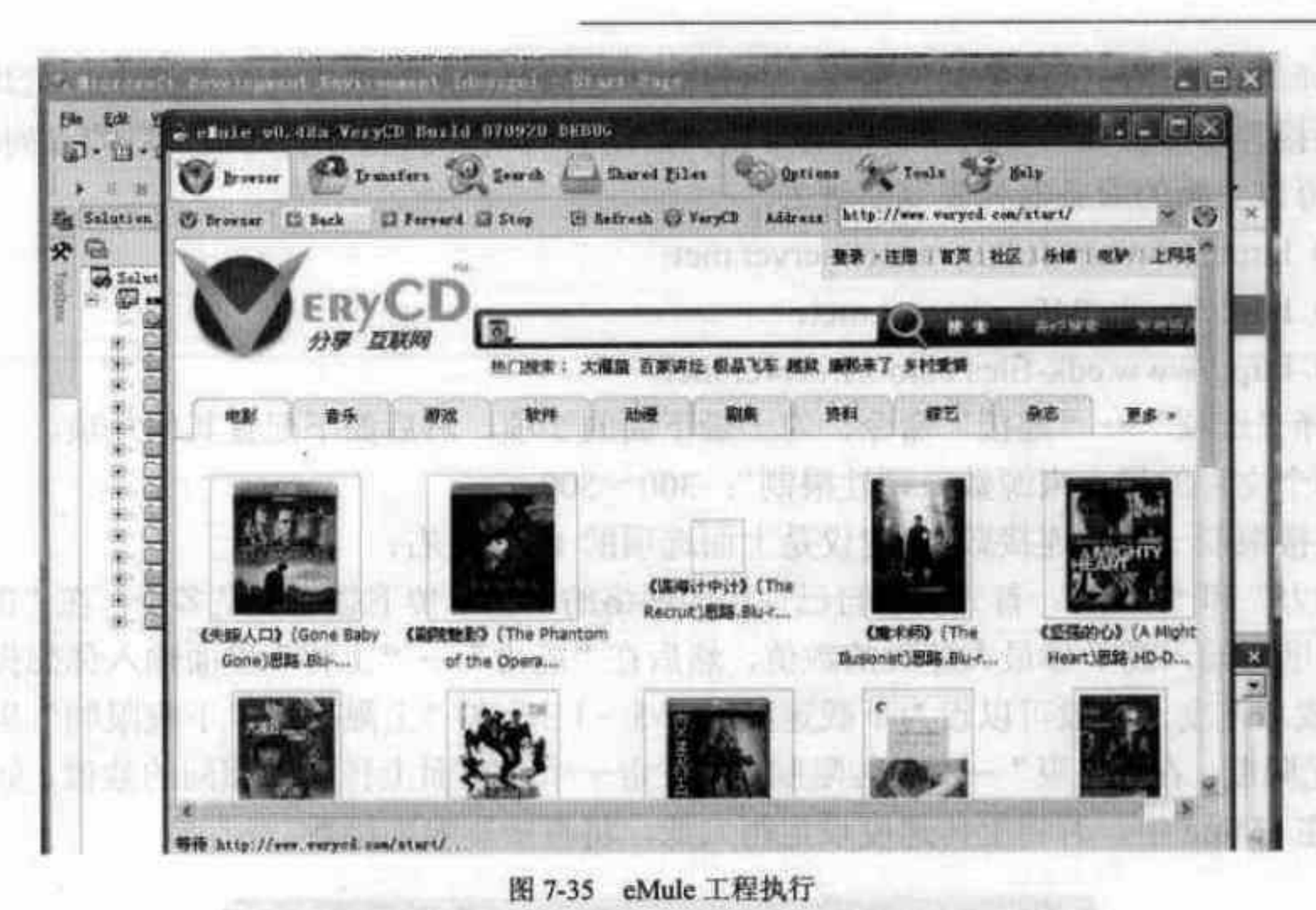


图 7-35 eMule 工程执行

7.5.2 系统安装与配置

如同大部分的 Windows 软件一样，eMule 的安装也非常的人性化，可以通过一直单击“下一步”按钮来完成安装。在选择安装的组件时，可以根据自己的需要，选择是否安装第三方赞助厂商的捆绑软件。

在安装完成后，选择“选项”→“目录”命令，把“下载的文件”和“临时文件”两个目录选择到不是系统盘的分区。此外，还有一个共享目录，用于选择用户希望共享的分区、目录和文件，如图 7-36 所示。

www.docin.com

图 7-36 目录选择示意图

www.ckook.com

选择“选项”→“服务器”命令，将里面的选项除最后一项以外全部选上，尤其应该把“自动连接静态服务器”项选上。然后单击“启动时自动更新服务器列表”后的“列表”，将以下的每日更新的服务器列表复制进去。

（1）http://www.srv1000.com/x1/server.met；

（2）http://emule.945.cn/server.met；

（3）http://www.edk-files.com/x1/server.met。

选择“选项”→“连接”命令，勾上最下面的 3 项，然后如下配置其他几项。

“每个文件的最大来源数—硬性限制”：300～500；

“连接限制—最大连接数”：建议是上面选项的 15 倍左右；

“能力”和“上限”：首先明确自己使用的网络的最大上传下载速度是多少，在“能力”→“下载”里面输入高于你最大速度的数值，然后在“能力”→“上传”里面输入你想提供给别人的下载总速度，一般可以设为下载速度的 1/5～1/3。在“上限”→“下载限制”里面不选中，即无限制，在“上限”→“上传限制”中设定一个和前面上传能力相同的数值，如图 7-37 所示。在 eMule 中，若将上传速度设定得太低，将直接影响连接数。

图 7-37　连接配置示意图

7.5.3　文件搜索和下载

通过 eMule 搜索和下载文件非常简单，它提供了多种方式供用户进行文件的搜索，如图 7-38 所示，具体步骤如下。

（1）单击“搜索”菜单，在“名字”中键入文件的关键字，在“类型”中选择下载的文件的类型，一般可以选择“任意”，在“方法”中选择下载方式，包括“服务器”、“全局（服务器）”、“Kad 网络”、“VeryCD（Web）”等，然后单击“开始”按钮，即可得到符合你下载条件的文件。

（2）选中你希望下载的文件，双击，即开始下载。

在文件搜索和下载过程中需要注意以下几点。

www.ckook.com

（1）信息搜索的时候应该尽可能全面，譬如分别使用简体中文、繁体中文、英文等查找；

（2）应该尽可能选择“可用源数”栏中数目多的文件下载，避免因为提供源的某个客户端关机而导致无法继续下载；

（3）eMule 是多点下载，不存在续传的问题。

图 7-38 文件搜索下载示意图

7.5.4 文件上传

通过 eMule 共享用户的文件也非常简单，单击菜单中的“共享文件”，选择用户希望共享的文件或者文件目录，右击选择“共享”即可，如图 7-39 所示。

www.docin.com

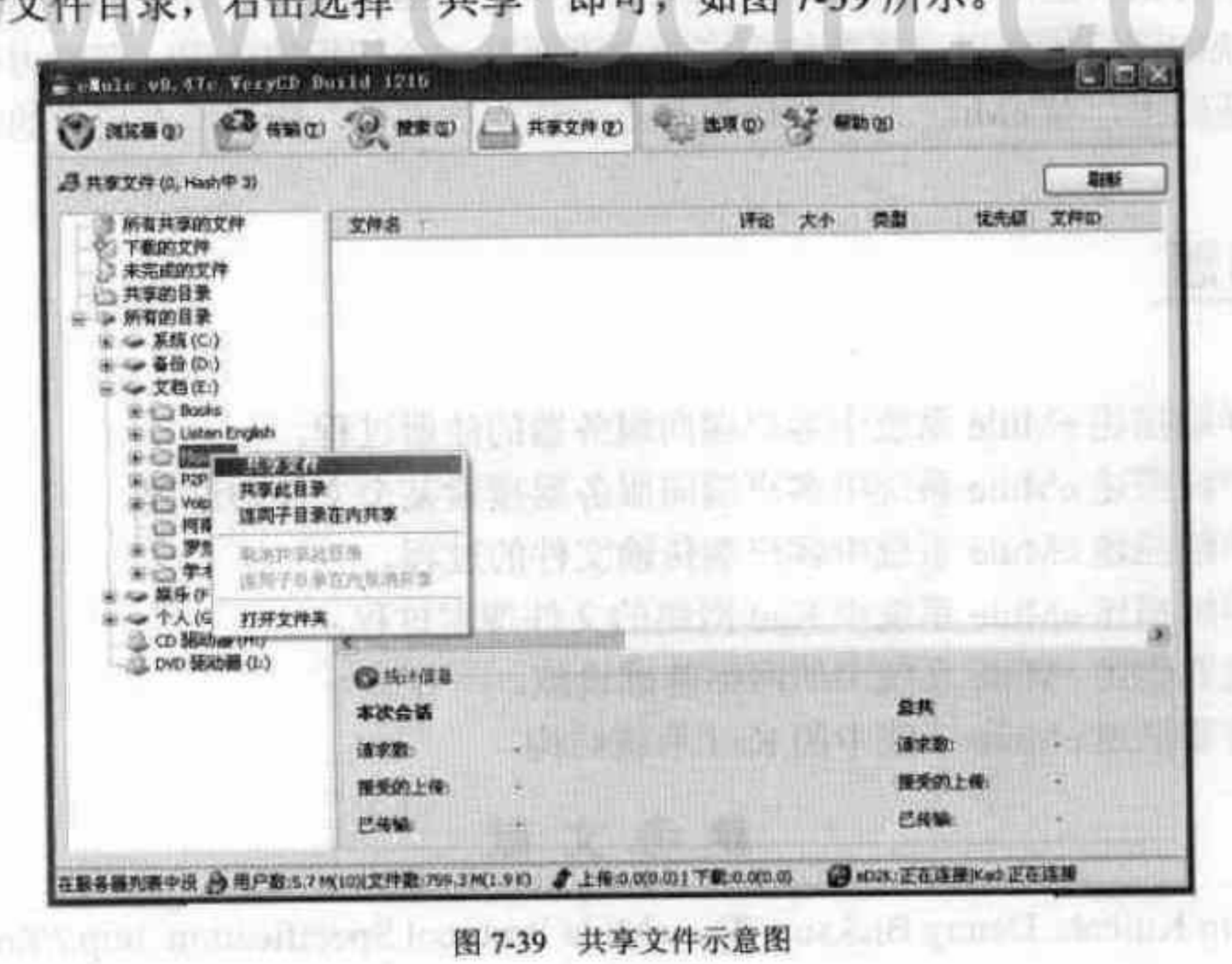


图 7-39 共享文件示意图

www.ckook.com

7.6 eMule 系统分析

eMule 系统作为一个文件共享系统，它包含了一个非结构化的 P2P 网络（eMule 网络）和一个结构化的 P2P 网络（Kad 网络），使得系统本身兼具了两类网络的特性，互补不足，拥有非常好的顽健性。eMule 系统在文件搜索以及文件下载等方面表现出了 P2P 系统的良好特性，其搜索的深度以及下载的快速较之传统的搜索引擎技术有很大的优势。而就 P2P 文件共享系统本身而言，eMule 系统也表现出了其独有的特性，独特的文件共享精神、稳定的系统性能表现、完备的 P2P 网络，可以说，eMule 是目前把文件共享、文件搜索和文件下载结合的最好的一个 P2P 文件共享软件。

eMule 系统作为一个开源的 P2P 系统，有很多很好的设计包含在其中。在其中的很多类的实现上，如果只是单纯地实现文件搜索以及下载的功能，其很多代码是可以进行精简的。但正是由于它们的存在，使得 eMule 的功能更加的完善。在了解和学习 eMule 系统及源代码的过程中，也可以学到很多 P2P 网络的基本设计思想。

7.7 本章总结

本章概括性地介绍了 P2P 文件共享系统产生的背景，简要地描述了 eMule 系统的设计目标，详细地阐述了 eMule 系统的系统原理，包括其两个不同网络协议的 P2P 网络。本章针对 eMule 的系统设计进行了简单的分析，主要包括 eMule 的系统结构、文件基础设施、网络基础设施、eMule 通信协议、任务处理机制、Kad 系统结构等几个方面。此外，本章还针对 eMule 的文件分块处理模块进行了源代码分析和实例，对 eMule 系统的系统部署进行了简单的描述。

本章力图为读者全面地了解 eMule 系统提供一定的基础知识，希望读者通过本章的介绍对 eMule 系统的网络原理以及系统结构这两个方面有一个初步的认识。有能力的读者可以在阅读本章的基础上，对 eMule 的源代码进行更加深入的剖析，应该能有更多的收获。

7.8 练习题

（1）请详细描述 eMule 系统中客户端向服务器的注册过程。
（2）请详细描述 eMule 系统中客户端向服务器搜索某个文件的过程。
（3）请详细描述 eMule 系统中客户端传输文件的过程。
（4）请详细描述 eMule 系统中 Kad 网络的文件搜索过程。
（5）请简要描述 eMule 系统中的网络基础设施。
（6）请简要描述 eMule 系统中的 Kad 系统结构。

参 考 文 献

[1] Yoram Kulbak, Danny Bickson. The eMule Protocol Specification. http://ftp.citkit.ru/pub/

www.ckook.com

sourceforge/e/ em/emule/protocol_guide.pdf, 2005 -01/2005-11

[2] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. of the 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS 2002). 2002. 153-161

[3] eMule 源代码学习心得. http://www.newsmth.net/pc/pcdoc.php?userid=wolfenstein

[4] 入门手册. http://www.emule.org.cn/guide/

[5] 新手教程. http://www.emule.org.cn/guide/

[6] 最新版 eMule VeryCD 源代码. http://www.emule.org.cn/download

www.docin.com

www.ckook.com

CHAPTER 8

第 8 章　P2P 文件共享系统开发实例二——BT 的设计与实现

BitTorrent（简称 BT）是一款使用最为广泛的 P2P 文件共享软件，在全世界的范围内，BitTorrent 拥有不计其数的用户。也有许多网站专门提供 BitTorrent 的 Tracker 服务。本章分析了 BitTorrent 系统的框架结构、Torrent 文件、Tracker 服务器的工作原理。并分析了一个使用 C++开发的 BitTorrent 客户端。

在学习了本章后，希望读者能够掌握以下几点：

（1）了解 BitTorrent 系统的原理；

（2）了解 BitTorrent 系统的架构及部署；

（3）了解 LibTorrent 库的基本结构和关键类。

8.1　BitTorrent 系统概述

www.dearin.com

BitTorrent（中文全称比特流）是一种多点下载的 P2P 文件共享软件。BitTorrent 也是 P2P 应用的典型代表，它拥有极其广泛的用户群，在整个因特网的流量当中占了相当大的比例。不同于 FTP（文件传输协议），只有一个发送源，BitTorrent 有多个发送点，当 BitTorrent 用户在下载时，同时也在上传，使大家都处在同步传送的状态。BitTorrent 的官方版本由 Bram Cohen 使用 Python 开发、维护。

8.2　BitTorrent 系统原理

一个 BitTorrent 用户在下载数据的同时也在为其他用户上传数据，因为大家都相互帮助，所以不会随着用户数的增加而降低下载速度。相反，用户数越多意味着能够为你提供数据的节点越多，所以下载速度越快。图 8-1 所示是使用 FTP、HTTP 等协议的分享流程。图 8-2 所示是使用 BitTorrent 做文件分享的流程。

BitTorrent 可以通过 URL 识别内容，并且设计为可以和 Web 无缝结合。

www.ckook.com

BitTorrent 相比 HTTP 的优势在于，同时下载一个文件的下载者在下载的同时不断互相上传数据。使用 BitTorrent 可以在网络负载增加很少的情况下，使文件同时供许多的下载者共享。

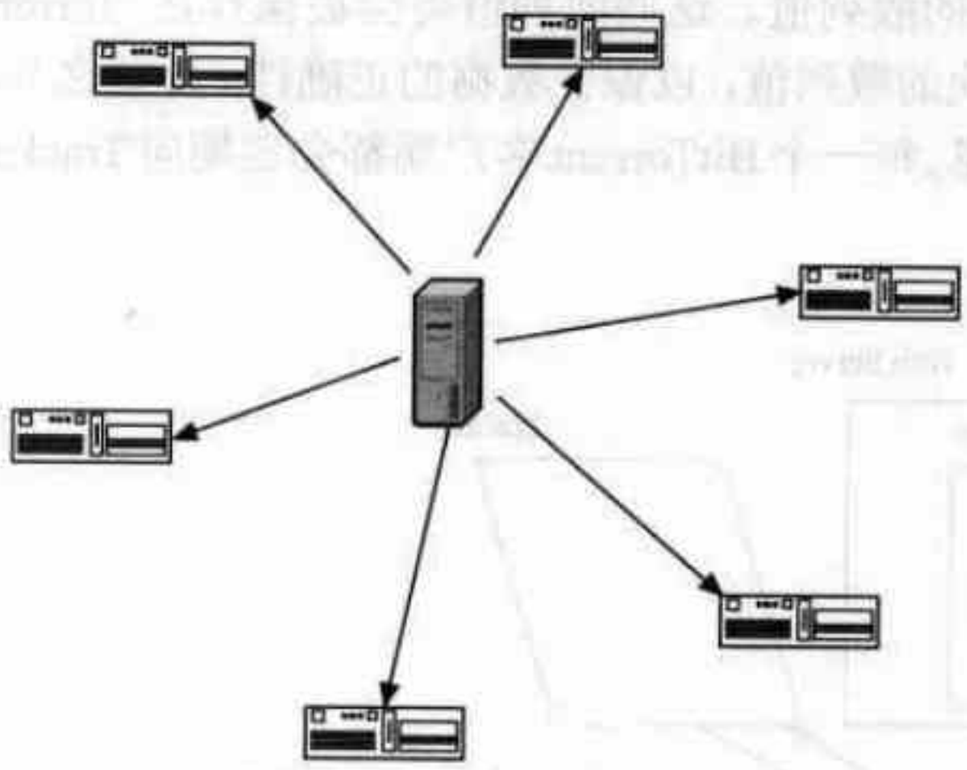


图 8-1 FTP/HTTP 分享流程

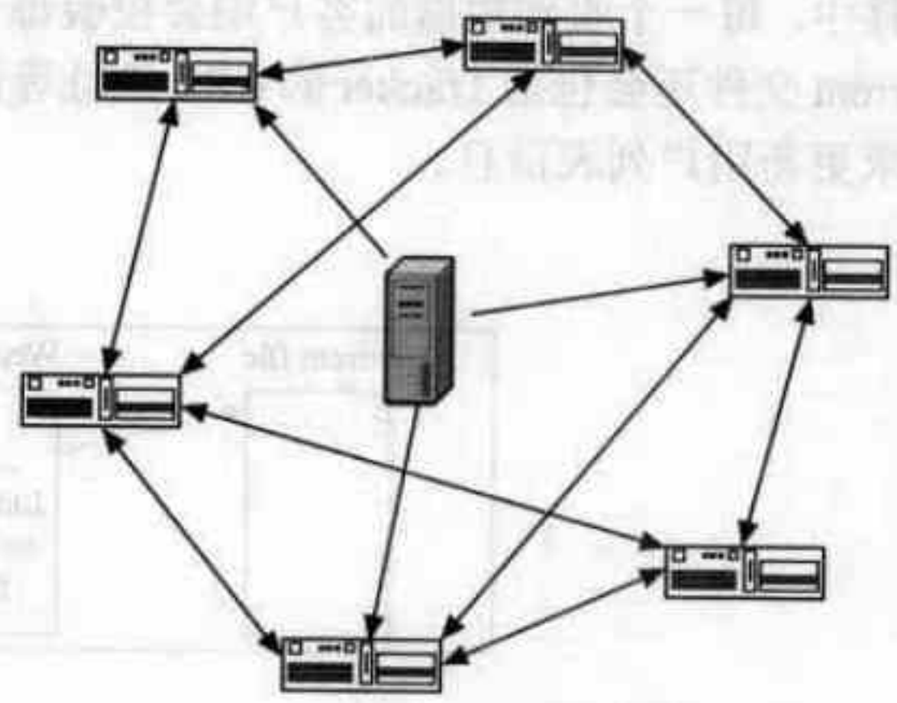


图 8-2 BitTorrent 分享流程

一个 BitTorrent 文件共享系统需要以下实体：

（1）一个普通的 Web 服务器；

（2）一个静态的“元信息”文件；

（3）一个 Tracker 服务器；

（4）一个原始下载者；

（5）终端用户的 Web 浏览器；

（6）终端下载者。

一个可以运行的 BitTorrent 系统至少包括 Tracker 服务器和 BitTorrent 客户端两部分。“元信息”文件发布的时候，需要先安装 Tracker 服务器，然后将要发布的文件打包成一个或多个文件，可以使用 btmakemetafile 程序对该文件进行分析，将该文件分成多个段和创建 checksum 信息以保证下载数据的正确性。

btmakemetafile 创建了一个包含 Tracker 服务器信息、文件分段和 checksum 信息的，以.torrent 为扩展名的小文件，然后发布第一个 Peer。每个 Peer 都会和 Tracker 建立连接，因为第 1 个 Peer 为发布者，所以其必须包含完整的文件。当第 2 个 Peer 开始下载的时候，会和 Tracker 服务器建立联系，取得已有 Peers 的信息，依照 Torrent 的文件划分规则开始向 Peer1 下载数据。比如将发布的文件分为 10 段，这时 Peer2 开始下载第一段，当 Peer3 下载的时候 Tracker 会自动告诉 Peer3 从 Peer2 下载文件的第一段，而从 Peer1 下载文件的其他段。以次类推，每个 Peer 可以同时从其他 Peers 那里下载数据段，而下载的同时，也向其他 Peers 提供自己已下载的数据段，从而提高了整体的下载速度也减轻了第一个信息发布者的负载压力。

8.2.1 BitTorrent 协议原理

BitTorrent 协议是 BitTorrent 系统运行的基础，也是使用各种不同的开发语言开发的 BitTorrent 部件能够相互正常通信的基础。而 Torrent 文件又是 BitTorrent 系统中最重要的信息与协议载体。

www.ckook.com

图 8-3 所示描绘了 BitTorrent 系统中的 3 个重要实体及其通信：一个 Tracker 服务器、一个保存在 Web 服务器上的 Torrent 文件以及对等点用户（Peers）。每一个将要被下载的文件首先被分成许多块，然后每一块都会计算出对应的散列值。这些散列值将会被保存在 Torrent 文件中，每一个收到数据的客户端会校验每一块的散列值，以保证数据的正确性。除此之外，Torrent 文件还会包括 Tracker 的 URL 地址等信息。每一个 BitTorrent 客户端都会定期向 Tracker 请求更新用户列表信息。

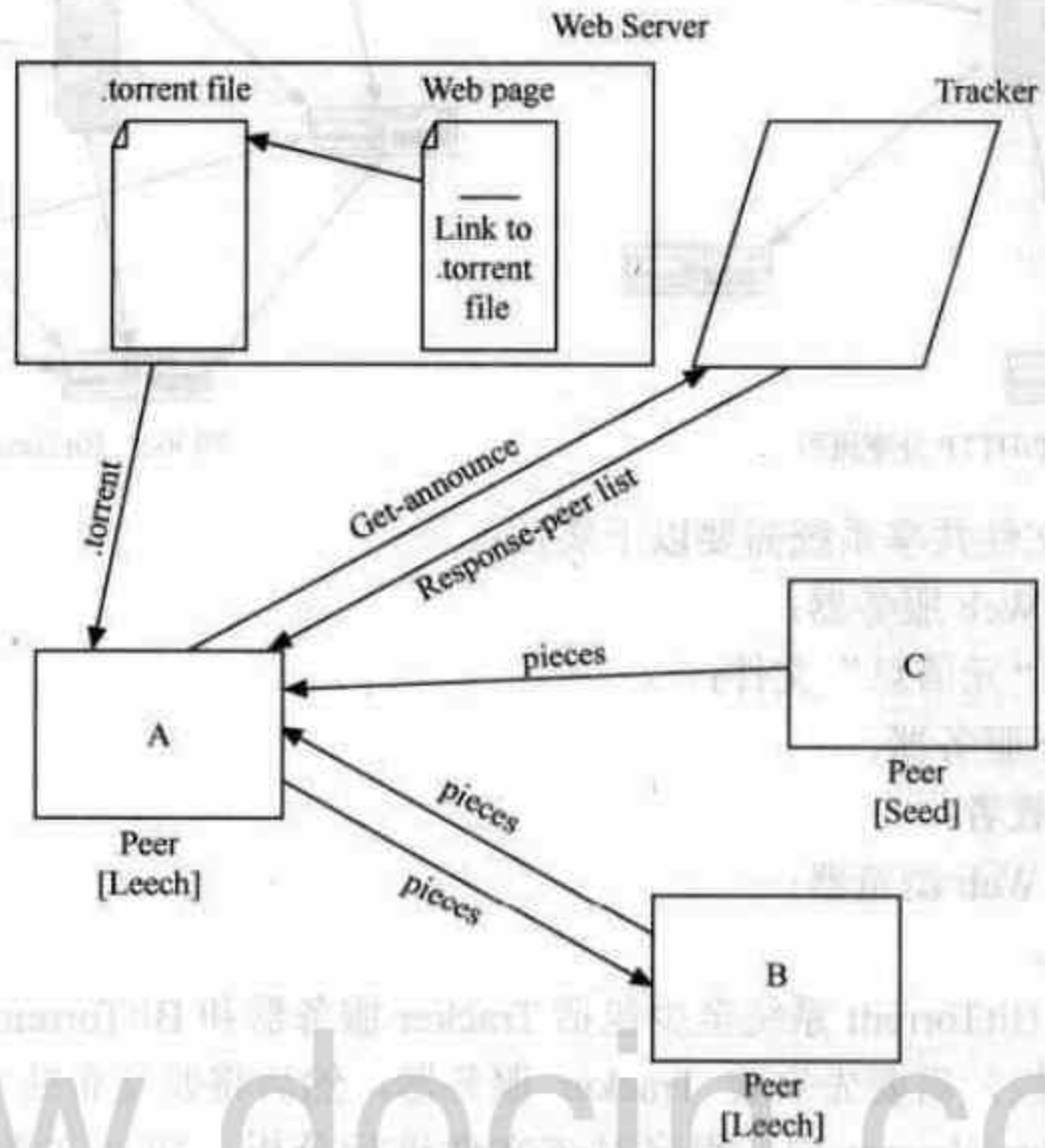


图 8-3 BitTorrent 系统实体关系图

www.docin.com

BitTorrent 的对等点用户可以被分为两类：一种是种子，一种是下载者。种子指的是那些已经完成下载的用户，他们不再从其他用户获取数据，而仅仅是无偿地为其他用户提供数据上传。而下载者是统称那些还没有完成数据下载的 BitTorrent 用户，他们在下载的同时也为其他用户提供上传。在 BitTorrent 系统中是鼓励一个下载完成的用户无偿地为其他用户提供下载的，但是这并没有强制的要求。事实上许多用户都是一下载完毕就关掉客户端，一般这种用户有一个不好听的名字——水蛭（leech）。

1. Torrent 文件

Torrent 文件是 BitTorrent 系统的一个重要组成部分，它为 BitTorrent 的客户端软件提供了和 Tracker 服务器通信的信息以及下载文件所需的其他信息，可以说它是 BitTorrent 协议的载体。使用 BitTorrent 下载数据时都必须先下载对应的 Torrent 文件。

Torrent 文件包含了待下载文件的一些基本信息，例如文件名、文件大小，除此之外，包含了发布服务的位置信息，所以 BitTorrent 客户端软件可以通过这些信息去查找服务器。最后 Torrent 文件还包含片段的 SHA1 校验码，以确保数据的完整性与正确性。

Torrent 文件以及 Tracker 服务器的回应信息都以一种简单而又高效可扩展的格式传送，

www.ckook.com

这种格式叫做“B 编码”。使用“B 编码”编码过的信息就是以包含字符串和整型数据的字典和列表的嵌套。表 8-1 所示介绍了 B 编码中的基本数据类型，以及其表示方法。

表 8-1　　B 编码中的基本数据类型

类　型	表 示 方 式	示　例
字符串（String）	<字符串长度>：字符串正文	4:spam 表示字符串'spam'
整型数据（Integer）	<i>整数值<e>	i3e 表示十进制整数 3，i-3e 表示十进制整数-3。所有以'i0'开头的除了代表 0 的 i0e，其他都无效
列表（List）	<l>Bencode Value<e>	l4:spam4:eggse 就表示['spam', 'eggs']
字典（Dictionary）	<d><bencodedstring><bencodedelement><e>	d3:cow3:moo4:spam4:eggse 相当于{'cow': 'moo', 'spam': 'eggs'}，d4:spaml1:a1:bee 相当于{'spam': ['a', 'b']}

Torrent 文件就是使用 B 编码的有以下关键字的字典，表 8-2 描述了 Torrent 文件的关键字以及其意义。

表 8-2　　Torrent 文件关键字

关 键 字	类　型	键值的意义
声明（announce）	字符串	此关键值代表 Tracker 的 URL 地址
信息（info）	字典	此关键值对应一个字典包含以下描述的关键值
名称（name）	字符串	对应一个字符串，代表默认的下载文件或存成目录的名字
块长（piece length）	整数	块长对应文件分割成的块的字节数。出于传输需要，文件被分割成大小相等的块，除了最后一块通常会小一些。块长一般来说是 2 的权值，大部分设块长为 256kByte
块（pieces）	字符串	对应一个字符串，此字符串长度是 20 的倍数。它可以再分成每 20Byte 一段的多个字符串，分别对应块在索引中的 SHA1 校验码（Hash）
长度（length）	整数	关键值长度和文件不能同时出现也不能都不出现。当长度出现说明这个元信息文件只是单文件下载，否则说明是多文件的目录结构下载。单文件情况下，长度对应文件长度的字节数。多文件情况被看作是把许多单文件按文件列表中的顺序连成一个大文件下载，而关键值文件就对应文件列表，是一个字典的列表，其中每个字典又包含下面的关键值
文件（files）	列表	
长度（length）	整数	文件长度的字节数
路径（path）	列表	一个包含字符串的列表，字符串就是子目录名，最后一项的字符串是文件名。在单文件情况下，关键值“名称”（name）是文件名；多文件情况下，它就成了目录名
建立日期（creation date）	整数	Torrent 文件建立日期
描述（comment）	字符串	目标文件描述
创建者（created by）	字符串	生成 Torrent 文件的软件

我们使用 UltraEdit 打开一个真实的 Torrent 文件，对比上表分析这个 Torrent 文件的构成，如图 8-4 所示。

读者也可以随手打开自己电脑上的 Torrent 文件对比学习。首先我们看到 Torrent 文件是一个字典，所以它的第一个字符是“d”。字典的第一个键值是一个字符串“8:announce”，按照协议我们知道后面对应的是 Tracker 服务器的地址。可以看到这个 Torrent 文件提供了多个 Tracker 服务器的地址，此后是 info 信息。有兴趣的读者可以自己仔细分析，本书到此不再深入讲解。

www.ckook.com

```
00000000h: 64 38 3A 61 6E 6E 6F 75 6E 63 65 33 34 3A 68 74 ; d8:announce34:ht
00000010h: 74 70 3A 2F 2F 74 72 61 63 6B 65 72 2E 79 64 79 ; tp://tracker.ydy
00000020h: 2E 63 6F 6D 3A 38 39 2F 61 6E 6E 6F 75 6E 63 65 ; .com:89/announce
00000030h: 31 33 3A 61 6E 6E 6F 75 6E 63 65 2D 6C 69 73 74 ; 13:announce-list
00000040h: 6C 6C 33 34 3A 68 74 74 70 3A 2F 2F 74 72 61 63 ; ll34:http://trac
00000050h: 6B 65 72 2E 79 64 79 2E 63 6F 6D 3A 38 39 2F 61 ; ker.ydy.com:89/a
00000060h: 6E 6E 6F 75 6E 63 65 65 6C 33 36 3A 68 74 74 70 ; nnouncee136:http
00000070h: 3A 2F 2F 62 74 66 61 6E 73 2E 33 33 32 32 2E 6F ; ://btfans.3322.o
00000080h: 72 67 3A 38 30 30 30 2F 61 6E 6E 6F 75 6E 63 65 ; rg:8000/announce
00000090h: 65 6C 34 35 3A 68 74 74 70 3A 2F 2F 77 77 77 2E ; el45:http://www.
000000a0h: 74 6F 72 72 65 6E 74 2D 64 6F 77 6E 6C 6F 61 64 ; torrent-download
000000b0h: 73 2E 74 6F 3A 32 37 31 30 2F 61 6E 6E 6F 75 6E ; s.to:2710/announ
000000c0h: 63 65 65 6C 34 31 3A 68 74 74 70 3A 2F 2F 64 65 ; cee141:http://de
000000d0h: 6E 69 73 2E 73 74 61 6C 6B 65 72 2E 68 33 71 2E ; nis.stalker.h3q.
000000e0h: 63 6F 6D 3A 36 39 36 39 2F 61 6E 6E 6F 75 6E 63 ; com:6969/announc
000000f0h: 65 65 31 30 3A 63 72 65 61 74 65 64 20 62 79 31 ; ee10:created by1
00000100h: 32 3A 42 69 74 43 6F 6D 65 74 2F 00 2E 5B 31 33 ; 2:BitComet/..[13
00000110h: 3A 63 72 65 61 74 69 6F 6E 20 64 61 74 65 69 31 ; :creation datei1
00000120h: 31 39 31 35 36 34 35 35 39 65 38 3A 65 6E 63 6F ; 191564559e8:enco
00000130h: 64 69 6E 67 33 3A 47 42 4B 34 3A 69 6E 66 6F 64 ; ding3:GBK4:infod
00000140h: 35 3A 66 69 6C 65 73 6C 64 34 3A 65 64 32 6B 31 ; 5:filesld4:ed2k1
00000150h: 36 3A 26 2E C8 C7 77 FC A0 1D 32 43 5D FC B8 34 ; 6:&.惹w鼬.2C]  4
00000160h: BE 5D 38 3A 66 69 6C 65 68 61 73 68 32 30 3A C6 ; 綸8:filehash20:?
00000170h: 63 CB 15 27 75 E7 27 C8 4B 63 9C AC 67 62 F7 3A ; c?'u?蓟c滃gb?
00000180h: 4C A8 D2 36 3A 6C 65 6E 67 74 68 69 32 36 30 32 ; L丅6:lengthi2602
00000190h: 36 34 36 38 38 65 34 3A 70 61 74 68 6C 33 31 3A ; 64688e4:pathl31:
000001a0h: 42 42 43 2E 41 72 74 2E 43 6F 6C 6C 65 63 74 69 ; BBC.Art.Collecti
000001b0h: 6F 6E 2E 43 44 31 2D 52 76 39 2E 72 6D 76 62 65 ; on.CD1-Rv9.rmvbe
000001c0h: 31 30 3A 70 61 74 68 2E 75 74 66 2D 38 6C 33 31 ; 10:path.utf-8l31
000001d0h: 3A 42 42 43 2E 41 72 74 2E 43 6F 6C 6C 65 63 74 ; :BBC.Art.Collect
000001e0h: 69 6F 6E 2E 43 44 31 2D 52 76 39 2E 72 6D 76 62 ; ion.CD1-Rv9.rmvb
000001f0h: 65 65 64 36 3A 6C 65 6E 67 74 68 69 33 30 36 34 ; eed6:lengthi3064
00000200h: 34 38 65 34 3A 70 61 74 68 6C 38 32 3A 5F 5F 5F ; 48e4:pathl82:___
00000210h: 5F 5F 70 61 64 64 69 6E 67 5F 66 69 6C 65 5F 30 ;   padding file 0
```

图 8-4　Torrent 文件

2．DHT 网络

目前发展出的 DHT 网络技术，使得无 Tracker 服务器的下载成为可能。DHT 全称为分布式散列表（Distributed Hash Table），是一种分布式存储方法。在不需要服务器的情况下，每个客户端负责一个小范围的路由，并负责存储一小部分数据，从而实现整个 DHT 网络的寻址和存储。使用支持该技术的 BitTorrent 下载软件，用户无需连上 Tracker 就可以下载，因为软件会在 DHT 网络中寻找下载同一文件的其他用户并与之通信，开始下载任务。有些软件还会自动通过 DHT 搜索种子资源，构成种子市场。另外，这里使用的 DHT 算法叫 Kademlia（在 eMule 中也有使用，常把它叫做 Kad，具体实现协议有所不同）。这种技术的优点十分明显，就是大大减轻了 Tracker 的负担（甚至不需要）。用户之间可以更快速地建立通信（特别是与 Tracker 连接不上的时候）。

8.2.2　BitTorrent Tracker 服务器原理

在 BitTorrent 系统中，Tracker 是指运行于服务器上的一个服务程序，也称 Tracker 服务器。这个程序能够追踪到底有多少人同时在下载或上传同一个文件。客户端连上 Tracker 服务器，就会获得一个正在下载和上传的用户的信息列表（通常包括 IP 地址、端口、客户端 ID 等信息），根据这些信息，BitTorrent 客户端会自动连上别的用户进行下载和上传。每个 Tracker

www.ckook.com

都有一个前端网页，HTTP 服务与 Tracker 可以运行在同一台主机上。用户从网页上可以搜索、下载以及上传 Torrent 文件，如图 8-5 所示。

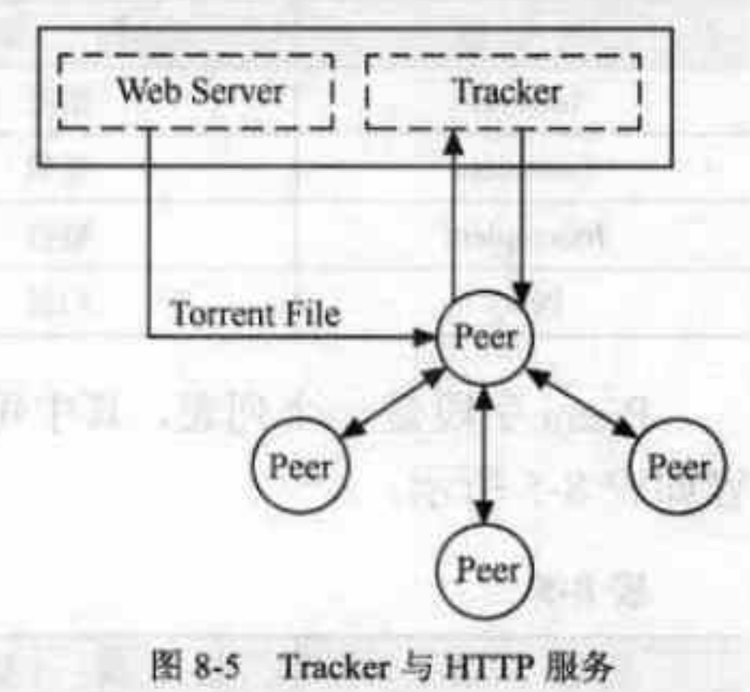


图 8-5 Tracker 与 HTTP 服务

Tracker 接收到节点发送的 announce 消息后，Tracker 根据 Info_hash 字段判断节点要下载的目标文件是哪个；找出 Info_hash 对应的节点列表，返回给节点；然后把节点的 IP 地址、Peer_id 记录到 Info_hash 对应的节点列表中。

节点通过 Torrent 文件得到追踪器地址，连接到追踪器，请求得到其他节点信息，追踪器就会返回正在下载同一目标文件的其他节点的信息，这个过程称为 announce（宣布）。这个节点与追踪器之间的通信是基于 HTTP 协议的，节点发给追踪器的 URL 基础部分是 Torrent 文件中记录的追踪器地址，后面加上一些 CGI 格式的参数，也就是用“?”分割开 URL 和参数，参数之间用“&”分隔，每个参数形如 param=value。URL 中可用的参数如表 8-3 所示。

表 8-3　URL 中可用参数

字　段	可否省略	描　述
Info_hash	否	Torrent 文件 info 字段摘要
Peer_id	否	节点标识
Port	否	节点的通信端口
Uploaded	可	已上传字节数
Downloaded	可	已下载字节数
Left	可	还需下载字节数
Event	可	自定义字段，如果包括这一字段，取值只能为 3 种：started，stopped，completed。started 表示加入网络，stopped 表示退出网络，completed 表示完成下载，作为种子
Ip	可	显式设置通信 IP 地址
Numwant	可	请求 tracker 返回节点数量

在标准中规定，Info_hash 字段必须把不可打印字符按%HH 形式展开，而 Peer_id 字段则必须不展开。

Event 字段都是为追踪器做统计用途的，每收到一个 started，追踪器就认为总节点多一个，每收到一个 stopped，追踪器就认为总节点少一个，每收到一个 completed，追踪器就认为一个节点变成种子，追踪器可以很方便地累计节点数和种子数。

前 7 个字段一般的客户端程序都有实现，最后两个字段没有明确标准，各种客户端程序、追踪器程序的处理也不一样。

在收到节点发来的请求之后，追踪器会返回一个节点信息列表。追踪器通过匹配 Info_hash 字段把正在下载同一目标文件的节点归类，记录这些节点的节点标识（Peer_id）、IP 地址和端口，每次有新节点请求就根据 Info_hash 字段找出正在下载同一目标文件的节点信息，返回给请求节点。返回信息是一个文本，内容是一个 B 编码的字典，如果追踪器处理出错，返回的信息很简单，只有一个字段 failure reason，字段内容是可读的出错描述。如果追踪器处理正常，返回信息如表 8-4 所示。

www.ckook.com

表 8-4　　failure reason 字段

字　段	类　型	描　述
Interval	整数	下一次请求需要等待的时间
Complete	整数	种子数
Incomplete	整数	未完成的节点数
Peers	列表	节点信息

Peers 字段是一个列表，其中每个元素是一个字典，记录一个节点。这个节点信息字典内容如表 8-5 所示。

表 8-5　　Peers 字段

字　段	类　型	描　述
Peer_id	整数	节点标识
Ip	整数	节点通信 IP
Port	整数	节点通信端口

Peer_id、Ip、Port 这 3 个参数是节点与追踪器通信时提供的，追踪器把这些信息返回给其他节点，这样节点之间就可以互相通信。

8.3　BitTorrent 系统设计

8.3.1　Arctic 客户端系统结构概述

Arctic 是一个开源 BitTorrent 客户端项目，图 8-6 是其程序运行界面。读者可以登录其官方网站 http://dev.int64.org/arctic.html 了解其最新的信息。Arctic 没有其他一些 BitTorrent 客户端软件的漂亮外观，它主要关注低内存消耗、低 CPU 使用率。不同于其他一些使用 Java 或者 Python 开发的 BitTorrent 客户端，Arctic 使用 C++开发，这使得它不会像 Java 或者 Python 程序那样使用较高的内存。Arctic 的开发使用了以下开源项目代码 Boost C++、Inno Setup、LibTorrent、zlib。

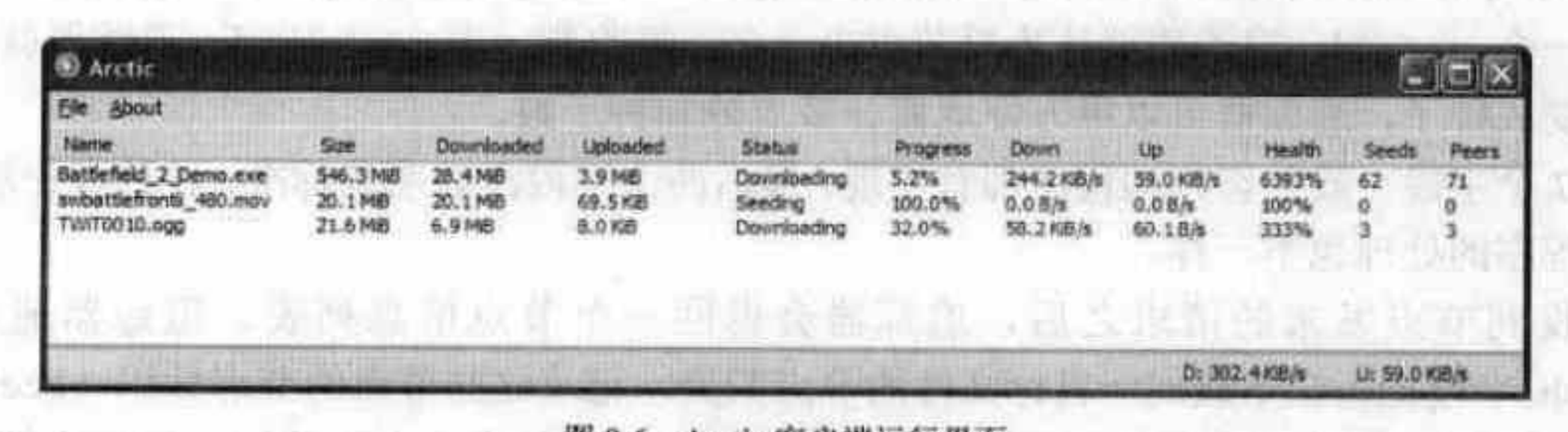


图 8-6　Arctic 客户端运行界面

如图 8-7 所示，Arctic 项目本身的代码并不多，它的 BitTorrent 核心部分使用的是 LibTorrent 库，Arctic 的程序主要是实现了简单的界面。Arctic 并没有使用 MFC，而是直接使用 WinAPI 做界面设计。

www.ckook.com

图 8-7 Arctic 源码结构

8.3.2 LibTorrent 库系统结构概述

LibTorrent 是使用 C++编写的优秀的 BitTorrent 库，它也同样是一个开源项目。LibTorrent 以性能著称，在高带宽的情况下，使用 LibTorrent 开发的客户端比使用 Python 开发的官方客户端的速度要快 3 倍以上。LibTorrent 的最新版本是 0.11.7，它的官方网站是 http://libtorrent.sourceforge.net。

如图 8-8 所示，LibTorrent 项目有 22 个 CPP 文件以及很多头文件，它们的编码风格都很

图 8-8 Libtorrent 库源码结构图

www.docx.com

www.ckook.com

适合阅读。

图 8-9 所示是 BitTorrent 中核心的类图。

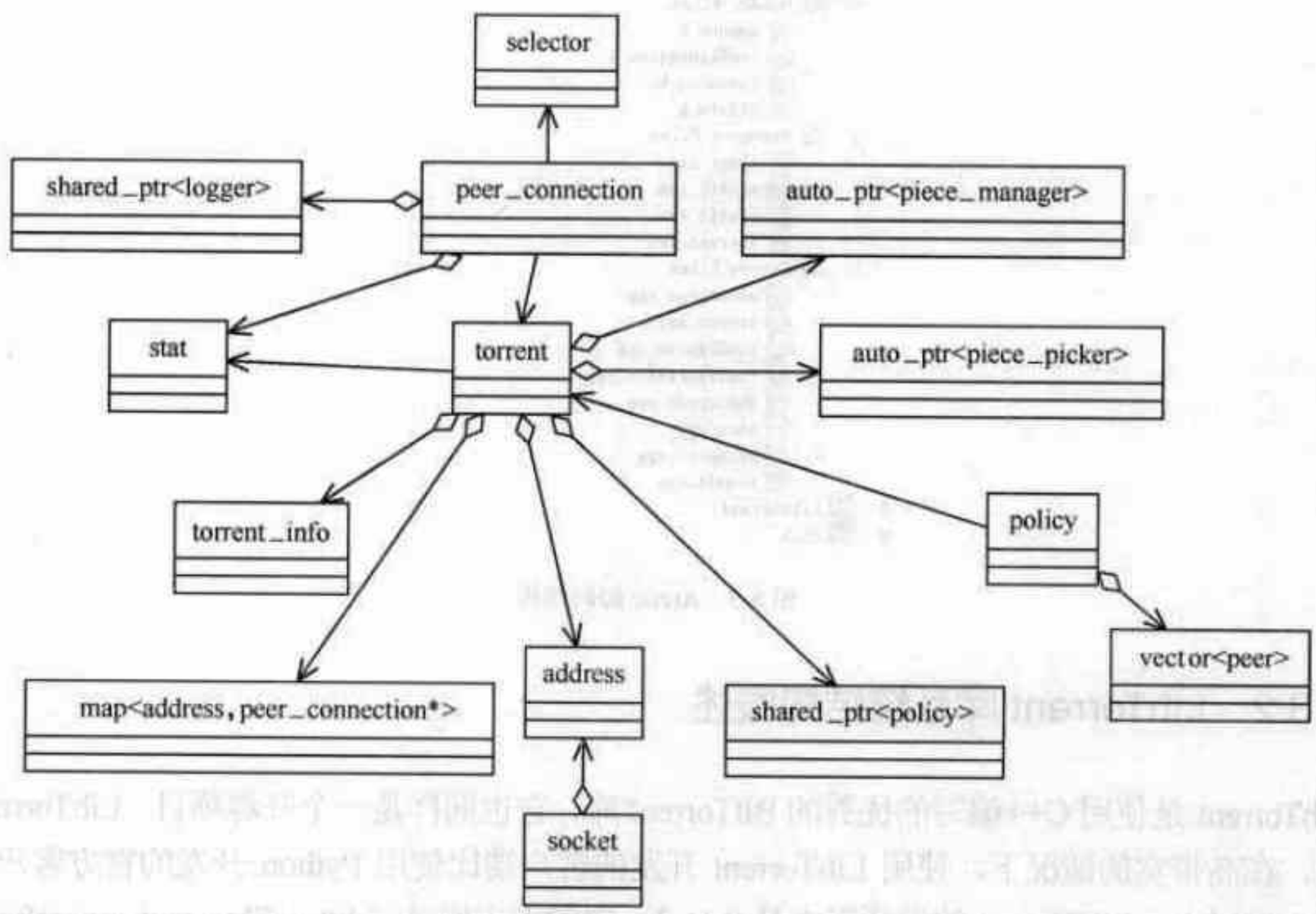


图 8-9 Libtorrent 核心类图

LibTorrent 的核心类包括 torrent 类、peer_connection 类等。

8.4 BitTorrent 程序代码分析

www.docin.com

8.4.1 Arctic 客户端代码分析

本节将对 BitTorrent 的源代码进行分析，其中 Arctic 的源代码可以在 http://dev.int64.org/arctic.html 中找到，LibTorrent 的源代码可以在 http://libtorrent.sourceforge.net 中找到。

我们首先分析 Arctic 的 main.cpp 文件。此文件是 Arctic 程序入口点函数所在的文件，是 Arctic 程序的框架所在。前一节已经讲过，Arctic 没有使用 MFC 类库而是直接使用 WinAPI 开发，所以我们简要回顾一下 Windows 应用程序开发的框架。有兴趣的读者可以参考《Programming Windows》以及《Programming Applications for Microsoft Windows》，它们的中文版分别叫做《Windows 程序设计》和《Windows 核心编程》，当然实际上它并不是讲 Windows 核心编程的书籍。

进行 Windows 程序设计，实际上是在进行一种面向对象的程序设计（OOP）。这一点在 Windows 中使用得最多的对象上表现最为明显。这种对象正是 Windows 之所以命名为“Windows”的原因，它具有人格化的特征，甚至可能会在梦中出现，这就是那个叫做“窗口”的东西。桌面上最明显的窗口就是应用程序窗口。这些窗口含有显示程序名称的标题列、菜

www.ckook.com

单甚至可能还有工具列和滚动条。另一类窗口是对话框，它可以有标题列也可以没有标题列。作为对象，使用者会在屏幕上看到这些窗口，并通过键盘和鼠标直接与它们进行交互操作。更有趣的是，程序编写者的观点与使用者的观点极其类似。窗口以“消息”的形式接收窗口的输入，窗口也用消息与其他窗口通信。对消息的理解将是学习如何写作 Windows 程序所必须越过的障碍之一。

这有一个 Windows 的消息范例。我们知道，大多数的 Windows 程序都有大小合适的应用程序窗口。也就是说，能够通过鼠标拖动窗口的边框来改变窗口的大小。通常，程序将通过改变窗口中的内容来响应这种大小的变化。你可能会猜测(并且你也是正确的)，是 Windows 本身而不是应用程序在处理与使用者重新调整窗口大小相关的全部杂乱程序。由于应用程序能改变其显示的样子，所以它也“知道”窗口大小改变了。

应用程序是如何知道使用者改变了窗口的大小的呢？由于程序写作者习惯了往常的文字模式程序，操作系统没有设置将此类消息通知给使用者的机制。问题的关键在于理解 Windows 所使用的架构。当使用者改变窗口的大小时，Window 给程序发送一个消息指出新窗口的大小。然后程序就可以调整窗口中的内容，以响应大小的变化。操作系统怎么给程序发送消息呢？其实，所谓“Windows 给程序发送消息”，是指 Windows 呼叫程序中的一个函数，该函数的参数描述了这个特定消息。这种位于 Windows 程序中的函数称为“窗口消息处理程序”。无疑，读者对程序呼叫操作系统的做法是很熟悉的。例如，程序在打开磁盘文件时就要使用有关的系统呼叫。读者所不习惯的，可能是操作系统呼叫程序，而这正是 Windows 对象导向架构的基础。

程序建立的每一个窗口都有相关的窗口消息处理程序。这个窗口消息处理程序是一个函数，既可以在程序中，也可以在动态链接库中。Windows 通过呼叫窗口消息处理程序来给窗口发送消息。窗口消息处理程序根据此消息进行处理，然后将控制传回给 Windows。更确切地说，窗口通常是在“窗口类别”的基础上建立的。窗口类别标识了处理窗口消息的窗口消息处理程序。使用窗口类别使多个窗口能够属于同一个窗口类别，并使用同一个窗口消息处理程序。例如，所有 Windows 程序中的所有按钮均依据同一个窗口类别。这个窗口类别与一个处理所有按钮消息的窗口消息处理程序（位于 Windows 的动态链接库中）连接。

在对象导向的程序设计中，对象是程序与数据的组合。窗口是一种对象，其程序是窗口消息处理程序。数据是窗口消息处理程序保存的信息和 Windows 为每个窗口以及系统中那个窗口类别保存的信息。

窗口消息处理程序处理给窗口发送消息。这些消息经常是告知窗口，使用者正使用键盘或者鼠标进行输入。这正是按键窗口知道它被“按下”的奥妙所在。在窗口大小改变，或者窗口表面需要重画时，由其他消息通知窗口。Windows 程序开始执行后，Windows 为该程序建立一个“消息队列”。这个消息队列用来存放该程序可能建立的各种不同窗口的消息。程序中有一小段程序代码，叫做“消息循环”，用来从队列中取出消息，并且将它们发送给相应的窗口消息处理程序。有些消息直接发送给窗口消息处理程序，不用放入消息队列中。

下面是 Main.cpp 的代码。

```
/*
    Copyright （c） 2004-2005 Cory Nelson

    Permission is hereby granted, free of charge, to any person obtaining
```

www.ckook.com

```
a copy of this software and associated documentation files （the
"Software"）, to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

#include "stdafx.h"
#include "resource.h"

using std::wstring;

//为主界面初始化 WNDCLASSEX 结构，WNDCLASSEX 结构包括窗口类的各种信息，包括窗口对
应的处理函数、背景的颜色、窗口类的名称等，并使用 RegisterClassEx 函数注册
static void InitMainClass（）  {
    WNDCLASSEX wc={0};
    wc.cbSize=sizeof（WNDCLASSEX）;
    wc.lpfnWndProc=Main_WndProc; //窗口消息的处理函数
    wc.hInstance=GetModuleHandle（NULL）;
    wc.hCursor=LoadCursor（NULL, IDC_ARROW）;
    wc.hbrBackground=（HBRUSH）（COLOR_WINDOW+1）;
    wc.lpszClassName=L"ArcticMain";
    wc.lpszMenuName=MAKEINTRESOURCE（IDR_MAINMENU）;
    wc.hIcon=（HICON）LoadImage（GetModuleHandle（NULL）, MAKEINTRESOURCE
（IDI_ARCTIC）, IMAGE_ICON, 32, 32, LR_SHARED）;
    wc.hIconSm=（HICON）LoadImage（GetModuleHandle（NULL）, MAKEINTRESOURCE
（IDI_ARCTIC）, IMAGE_ICON, 16, 16, LR_SHARED）;
    RegisterClassEx（&wc）; //注册 WNDCLASSEX 结构
}

//初始化调试窗口的 WNDCLASSEX 结构
static void InitDebugClass（）  {
    WNDCLASSEX wc={0};
```


www.ckook.com

```
        wc.cbSize=sizeof（WNDCLASSEX）；
        wc.lpfnWndProc=Debug_WndProc;
        wc.hInstance=GetModuleHandle（NULL）；
        wc.hCursor=LoadCursor（NULL, IDC_ARROW）；
        wc.hbrBackground=（HBRUSH）（COLOR_WINDOW+1）；
        wc.lpszClassName=L"ArcticDebug";
        wc.hIcon=（HICON）LoadImage（GetModuleHandle（NULL）, MAKEINTRESOURCE
（IDI_ARCTIC）, IMAGE_ICON, 32, 32, LR_SHARED）；
        wc.hIconSm=（HICON）LoadImage（GetModuleHandle（NULL）, MAKEINTRESOURCE
（IDI_ARCTIC）, IMAGE_ICON, 16, 16, LR_SHARED）；
        RegisterClassEx（&wc）；
    }
    //在注册表里注册".torrent"文件
    static bool IsAssociated（）{
        HKEY key;
        RegCreateKeyEx（HKEY_CLASSES_ROOT, L".torrent", 0, NULL, REG_OPTION_NON_
VOLATILE, KEY_READ, NULL, &key, NULL）；

        wchar_t app[128];
        unsigned long applen=sizeof（app）；

        bool associated=（RegQueryValueEx（key, NULL, NULL, NULL, （LPBYTE）app, &applen）
== ERROR_SUCCESS && !wcscmp（app, L"Arctic.torrent"））；

        RegCloseKey（key）；
        return associated;
    }
    //以下是 Arctic 程序的入口函数 wWinMain，不同于普通的 C/C++程序，Windows 程序的入口函数
并不是 main 函数
    int WINAPI wWinMain（HINSTANCE, HINSTANCE, LPWSTR, int nCmdShow）{
        HWND hwnd;
        {
            HANDLE mutex=CreateMutex（NULL, TRUE, L"Global\\ArcticTorrent"）；
            if（GetLastError（）==ERROR_ALREADY_EXISTS）{
                hwnd=FindWindow（L"ArcticMain", L"Arctic"）；
                if（hwnd）{
                    ShowWindow（hwnd, SW_SHOWNORMAL）；
                    SetForegroundWindow（hwnd）；
                }
                return 0;
            }
        }

        CoInitializeEx（NULL, COINIT_APARTMENTTHREADED）；  //初始化 COM 组件
```

www.docin.com

```
        InitCommonControls（）；  //初始化通用控件
        InitMainClass（）；          //调用 InitMainClass 初始化并注册主窗口的 WNDCLASSEX 类
        InitDebugClass（）；

        if（!conf.Load（））  { // Configuration conf;定义在 mainproc.cpp Configuration 类包含了一
些用于显示的信息
            if（!IsAssociated（））  {
                wstring text=loadstring（IDS_ASSOCIATE）；
                wstring caption=loadstring（IDS_ASSOCIATECAPTION）；

                if（MessageBox（NULL, text.c_str（）, caption.c_str（）, MB_ICONQUESTION|MB_YESNO）
==IDYES）
                    Associate（）；
            }
            conf.Save（）；
        }

        {
            int x, y, w, h;

            const RECT &wpos=conf.position;
            if（wpos.top!=0 || wpos.left!=0 || wpos.bottom!=0 || wpos.right!=0）  {
                x=wpos.left;
                y=wpos.top;
                w=wpos.right-wpos.left;
                h=wpos.bottom-wpos.top;
            }
            else {
                RECT rect;
                GetClientRect（GetDesktopWindow（）, &rect）；

                x=（（rect.right-rect.left）/2）-380;
                y=（（rect.bottom-rect.top）/2）-100;
                w=760;
                h=200;
            }
        //创建窗口
            hwnd=CreateWindowEx（WS_EX_ACCEPTFILES, L"ArcticMain", L"Arctic", WS_ OVERLA
PPEDWINDOW, x, y, w, h, NULL, NULL, NULL, NULL）；
            if（!hwnd）  return -1;
        }

        SendMessage（hwnd, DM_REPOSITION, 0, 0）；
```

www.ccin.com

www.ckook.com

```
    ShowWindow（hwnd, nCmdShow）;//开始显示窗口
    UpdateWindow（hwnd）;

//此后主程序进入消息循环，收到 Windows 系统发出消息后，程序调用 Main_WndProc 函数处理消息
    MSG msg;
    while（GetMessage（&msg, NULL, 0, 0）>0） {
        TranslateMessage（&msg）;
        DispatchMessage（&msg）;
    }

    return （int）msg.wParam;
}
```

下面我们分析另一个重要的文件 mainproc.cpp。mainproc.cpp 主要包括了 Main_WndProc 函数，当 Windows 向程序发送消息时，程序会调用 Main_WndProc 函数来处理消息。

```
/*
    Copyright （c） 2004-2005 Cory Nelson

    Permission is hereby granted, free of charge, to any person obtaining
    a copy of this software and associated documentation files （the
    "Software"）, to deal in the Software without restriction, including
    without limitation the rights to use, copy, modify, merge, publish,
    distribute, sublicense, and/or sell copies of the Software, and to
    permit persons to whom the Software is furnished to do so, subject to
    the following conditions:

    The above copyright notice and this permission notice shall be included
    in all copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
    IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
    CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
    SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
#include "stdafx.h"
#include "resource.h"

using std::pair;using std::sort;using std::vector;using std::string;
using std::wstring;using std::auto_ptr;using std::exception;using boost::filesystem::path;
using namespace boost::spirit;
```

www.ckook.com

```
#define IDC_LIST            201
#define IDC_STATUS  202
#define TIMER_UPDATE     1
#define TRAY_ARCTIC            1
#define WM_ARCTIC_TRAY  （WM_APP+1）

static const UINT columns[]={IDS_NAME, IDS_SIZE, IDS_DOWNLOADED, IDS_UPLOADED, IDS_STATUS, IDS_PROGRESS, IDS_DOWNSPEED, IDS_UPSPEED, IDS_HEALTH, IDS_SEEDS, IDS_PEERS};
static const size_t columncount=sizeof（columns）/sizeof（UINT）;

//Torrent 结构
struct Torrent {
    std::string file; //文件名
    libtorrent::torrent_handle handle; //使用 LibTorrent 库
    wstring cols[columncount];

    bool operator<（const Torrent &t）  const {
        return wcsicmp（cols[0].c_str（）, t.cols[0].c_str（））<0;
    }
};

Configuration conf; //定义全局变量 conf

static libtorrent::session *session=NULL;
static vector<Torrent> torrents;
static bool allpaused=false;

HWND main=NULL;
HWND about=NULL;
static HWND debug=NULL;
bool debugopen=false;

static wstring paused;
static wstring queued;
static wstring checking;
static wstring connecting;
static wstring downloading;
static wstring seeding;
static wstring unknown;
static wstring bytes;
static wstring kibibytes;
static wstring mebibytes;
static wstring gibibytes;
```

www.docin.com

www.ckook.com

```
static NOTIFYICONDATA nid={0};
static unsigned int traycreatedmessage=0;

static void Main_OnClose（HWND hwnd）  {
    DestroyWindow（hwnd）;
}

static wchar_t *strsize（double s）  {
    double downloaded;
    const wchar_t *units;
    if（s>=1073741824）  {
        downloaded=s/1073741824.0;
        units=gibibytes.c_str（）;}
    else if（s>=1048576）  {
        downloaded=s/1048576.0;
        units=mebibytes.c_str（）;}
    else if（s>=1024）  {
        downloaded=s/1024.0;
        units=kibibytes.c_str（）;}
    else {
        downloaded=s;
        units=bytes.c_str（）;}

    static wchar_t buf[64];
    StringCchPrintf（buf, 64, L"%.1f %s", downloaded, units）;
    return buf;
}

static void AddTorrent（HWND hwnd, path file）  {
    try {
        const path rfile=getmodulepath（）/"resume"/file;

        libtorrent::entry metadata=bdecode（file）;
        libtorrent::entry resumedata;

        if（boost::filesystem::exists（rfile））  {
            try {
                resumedata=bdecode（rfile.leaf（））;}
            catch（...）  {
                wchar_t text[256], title[128];
                LoadString（GetModuleHandle（NULL）, IDS_RESUMEERR, title, 128）;
                LoadString（GetModuleHandle（NULL）, IDS_RESUMEERRTEXT, text, 256）;
```

www.ckook.com

```
                MessageBox（hwnd, text, title, MB_ICONERROR|MB_OK）;

                boost::filesystem::remove（rfile）;
            }
        }

        if（!boost::filesystem::exists（getmodulepath（）/"torrents"））
            boost::filesystem::create_directory（getmodulepath（）/"torrents"）;

        if（!boost::filesystem::exists（getmodulepath（）/"torrents"/file.leaf（））））
            boost::filesystem::copy_file（file, getmodulepath（）/"torrents"/file.leaf（））;

        if（!boost::filesystem::exists（conf.savepath））
            boost::filesystem::create_directory（conf.savepath）;

        vector<libtorrent::torrent_handle>::size_type i=torrents.size（）;

        Torrent t;
        t.file=file.leaf（）;
        t.handle=session->add_torrent（metadata, conf.savepath, resumedata）;

        t.handle.set_max_uploads（conf.maxup）;
        t.handle.set_max_connections（conf.maxcon）;

        const libtorrent::torrent_info &info=t.handle.get_torrent_info（）;

        t.cols[0]=mbstowcs（info.name（））;
        t.cols[1]=strsize（（double）info.total_size（））;

        torrents.push_back（t）;
        sort（torrents.begin（）, torrents.end（））;

        if（allpaused） t.handle.pause（）;

        HWND list=GetDlgItem（hwnd, IDC_LIST）;
        ListView_SetItemCount（list, torrents.size（））;
    }
    catch（exception &ex） {
        wstring text=mbstowcs（ex.what（））;
        wstring caption=loadstring（IDS_EXCEPTION）;

        MessageBox（hwnd, text.c_str（）, caption.c_str（）, MB_ICONERROR|MB_OK）;
    }
```

www.ccom.com

www.ckook.com

```
}
//处理 windows 消息的函数
static void Main_OnCommand（HWND hwnd, int id, HWND hwndCtl, UINT codeNotify） {
    switch（id） {
        case ID_FILE_OPEN1: {
            wchar_t file[MAX_PATH]=L"";

            OPENFILENAME ofn={0};
            ofn.lStructSize=sizeof（OPENFILENAME）;
            ofn.hwndOwner=hwnd;
            ofn.lpstrFilter=L"Torrent Files （*.torrent）\0*.torrent\0All Files （*.*）\0*.*\0";
            ofn.lpstrFile=file;
            ofn.nMaxFile=MAX_PATH;
            ofn.Flags=OFN_EXPLORER|OFN_HIDEREADONLY|OFN_FILEMUSTEXIST;
            ofn.lpstrDefExt=L"torrent";

            if（GetOpenFileName（&ofn）） AddTorrent（hwnd, wcstombs（file））;
        } break;
        case ID_FILE_CONFIGURATION:
            if（DialogBox（GetModuleHandle（NULL）, MAKEINTRESOURCE（IDD_CONFIG）,
hwnd, Config_DlgProc）==IDOK） {
                try {
                    session->set_upload_rate_limit（（conf.uplimit!=-1）?conf.uplimit*1024:-1）;
                session->set_download_rate_limit（（conf.downlimit!=-1）?conf.downlimit*1024:-1）;

                    session->listen_on（pair<int,int>（conf.firstport, conf.lastport））;

                    for（vector<Torrent>::size_type i=0; i<torrents.size（）; i++）
                        torrents[i].handle.move_storage（conf.savepath）;
                }
                catch（exception &ex） {
                    wstring text=mbstowcs（ex.what（））;
                    wstring caption=loadstring（IDS_EXCEPTION）;

                    MessageBox(hwnd, text.c_str( ), caption.c_str( ), MB_ICONERROR|MB_OK);
                }
            }
            break;
        case ID_FILE_DEBUG:
            if（debugopen） {
                ShowWindow（debug, SW_HIDE）;
```

www.ckook.com

```
                HMENU menu=GetMenu（hwnd）;
                CheckMenuItem（menu,ID_FILE_DEBUG,MF_BYCOMMAND|MF_UNCHECKED）;
                debugopen=false;}
            else {
                ShowWindow（debug, SW_SHOWNORMAL）;
                SetForegroundWindow（debug）;

                HMENU menu=GetMenu（hwnd）;
                CheckMenuItem（menu, ID_FILE_DEBUG, MF_BYCOMMAND|MF_CHECKED）;
                debugopen=true;
            }
            break;
        case ID_FILE_EXIT:
        case ID_TRAY_EXIT:
            DestroyWindow（hwnd）;
            break;
        case ID_ABOUT_ABOUT:
        case ID_TRAY_ABOUT:
            if(!about) about=CreateDialog(GetModuleHandle(NULL), MAKEINTRESOURCE（IDD_ABOUT),
hwnd, About_DlgProc）;
            else SetForegroundWindow（about）;
            break;
        case ID_CONTEXT_OPEN: {
            HWND list=GetDlgItem（hwnd, IDC_LIST）;
            int index=ListView_GetSelectionMark（list）;

            if（index!=-1）  {
                path  p=torrents[index].handle.save_path（）/torrents[index].handle.get_torrent_info
（）.name（）;
                ShellExecuteA（NULL, "open", p.native_file_string（）.c_str（）, NULL, NULL,
SW_SHOW）;
            }
        } break;
        case ID_CONTEXT_ANNOUNCE: {
            HWND list=GetDlgItem（hwnd, IDC_LIST）;
            int index=ListView_GetSelectionMark（list）;

            if（index!=-1）  torrents[index].handle.force_reannounce（）;
        } break;
        case ID_CONTEXT_PAUSE: {
            HWND list=GetDlgItem（hwnd, IDC_LIST）;
            int index=ListView_GetSelectionMark（list）;
```

www.ocm.com

www.ckook.com

```
        if（index!=-1） torrents[index].handle.pause（）；
    } break;
    case ID_CONTEXT_RESUME: {
        HWND list=GetDlgItem（hwnd, IDC_LIST）；
        int index=ListView_GetSelectionMark（list）；

        if（index!=-1） torrents[index].handle.resume（）；
    } break;
    case ID_CONTEXT_REMOVE: {
        HWND list=GetDlgItem（hwnd, IDC_LIST）；
        int index=ListView_GetSelectionMark（list）；

        if（index!=-1） {
            wchar_t text[512];
            wstring caption=loadstring（IDS_REMOVECAPTION）；

            wstring fmt=loadstring（IDS_REMOVE）；
            string name=torrents[index].handle.get_torrent_info（）.name（）；
            StringCchPrintf（text, 512, fmt.c_str（）, name.c_str（））；

            if(MessageBox(hwnd,text,caption.c_str(), MB_ICONQUESTION|MB_YESNO)==IDYES)
{
                ListView_DeleteItem（list, index）；
                session->remove_torrent（torrents[index].handle）；
                {
                    try {
                        boost::filesystem::remove（getmodulepath（）/"torrents"/torrents[index].file）；
                        boost::filesystem::remove（getmodulepath（）/"resume"/torrents[index].file）；
                    }
                    catch（...） {// whatever.}
                }
                torrents.erase（torrents.begin（）+index）；
            }
        }
    } break;
    case ID_TRAY_OPEN:
        ShowWindow（hwnd, SW_SHOWNORMAL）；
        SetForegroundWindow（hwnd）；
        break;
    case ID_TRAY_PAUSE:
        for（vector<Torrent>::size_type i=0; i<torrents.size（）；i++）
            torrents[i].handle.pause（）；
        allpaused=true;
```

www.ckook.com

```
                break;
            case ID_TRAY_RESUME:
                for（vector<Torrent>::size_type i=0; i<torrents.size（）; i++）
                    torrents[i].handle.resume（）;
                allpaused=false;
                break;
        }
    }
    //处理 windows 消息 create 的函数
    static BOOL Main_OnCreate（HWND hwnd, LPCREATESTRUCT lpCreateStruct） {
        // 设置图标
        nid.cbSize=sizeof（NOTIFYICONDATA）;
        nid.hWnd=hwnd;
        nid.uID=TRAY_ARCTIC;
        nid.uCallbackMessage=WM_ARCTIC_TRAY;
        nid.uFlags=NIF_ICON|NIF_MESSAGE|NIF_TIP;
        nid.hIcon=（HICON）LoadImage（GetModuleHandle（NULL）, MAKEINTRESOURCE
（IDI_ARCTIC）, IMAGE_ICON, 16, 16, LR_SHARED）;
        StringCchCopy（nid.szTip, 64, L"Arctic"）;

        Shell_NotifyIcon（NIM_ADD, &nid）;
        traycreatedmessage=RegisterWindowMessage（L"TaskbarCreated"）;

        ////////////////////////////////////////////////////
        //设置状态栏

        HWND status=CreateWindow（STATUSCLASSNAME, NULL, WS_CHILD|WS_VISIBLE,
            0, 0, 0, 0,
            hwnd, （HMENU）IDC_STATUS, NULL, NULL）;

        RECT statusrect;
        GetWindowRect（status, &statusrect）;

        int parts[3];
        parts[0]=std::max（（（statusrect.right-statusrect.left）-192）, （LONG）0）;
        parts[1]=parts[0]+96;
        parts[2]=parts[1]+96;

        SendMessage（status, SB_SETPARTS, 3, （LPARAM）parts）;
        SendMessage（status, SB_SETTEXT, 0|SBitTorrent_NOBORDERS, （LPARAM）L""）;
        SendMessage（status, SB_SETTEXT, 1|SBitTorrent_NOBORDERS, （LPARAM）L""）;
        SendMessage（status, SB_SETTEXT, 2|SBitTorrent_NOBORDERS, （LPARAM）L""）;
```


www.ckook.com

```
////////////////////////////////////////////////////////////
//设置列表显示控件 ListView

RECT rect;
GetClientRect（hwnd, &rect）;

HWND list= Create Window （WC_LISTVIEW,NULL, WS_ CHILD|WS_ VISIBLE|LVS
_NOSORTHEADER|LVS_OWNERDATA|LVS_REPORT|LVS_SHOWSELALWAYS|LVS_SINGLESEL,
    0, 0, rect.right-rect.left, （rect.bottom-rect.top）-（statusrect.bottom-statusrect.top）,
    hwnd, （HMENU）IDC_LIST, NULL, NULL）;
ListView_SetExtendedListViewStyle（list, LVS_EX_FULLROWSELECT|LVS_EX_LABELTIP）;

wstring buf;

LVCOLUMN lvc={0};
lvc.mask=LVCF_FMT|LVCF_WIDTH|LVCF_TEXT|LVCF_SUBITEM;
lvc.fmt=LVCFMT_LEFT;

for（size_t i=0; i<columncount; i++）  {
    lvc.iSubItem=（int）i;
    lvc.cx=conf.columns[i];
    buf=loadstring（columns[i]）;
    lvc.pszText=（LPWSTR）buf.c_str（）;

    ListView_InsertColumn（list, i, &lvc）;
}

////////////////////////////////////////////////////////////
///创建一个 torrent 会话

try {
    session=new libtorrent::session（libtorrent::fingerprint（"AR", 1, 0, 0, 1）, pair<int,int>
（conf.firstport, conf.lastport））;

    session->disable_extensions（）;
    session->enable_extension（libtorrent::peer_connection::extended_metadata_message）;
    session->enable_extension（libtorrent::peer_connection::extended_peer_exchange_message）;
    session->enable_extension（libtorrent::peer_connection::extended_listen_port_message）;

    session->set_upload_rate_limit（（conf.uplimit!=-1）?conf.uplimit*1024:-1）;
    session->set_download_rate_limit（（conf.downlimit!=-1）?conf.downlimit*1024:-1）;
    session->set_max_connections（conf.maxcon）;
```

www.ckook.com

```
#ifdef _DEBUG
        session->set_severity_level（libtorrent::alert::debug）;
#else
        session->set_severity_level（libtorrent::alert::info）;
#endif
    }
    catch（exception &ex） {
        wstring text=mbstowcs（ex.what（））;
        wstring caption=loadstring（IDS_EXCEPTION）;

        MessageBox（hwnd, text.c_str（）, caption.c_str（）, MB_ICONERROR|MB_OK）;
        return -1;
}

////////////////////////////////////////////////////////
///创建调试窗口

main=hwnd;

GetWindowRect（hwnd, &rect）;

        wstring caption=loadstring（IDS_DEBUG）;
        debug=CreateWindowEx（WS_ EX_ APPWINDOW, L"ArcticDebug", caption.c _ str（）, WS _
OVERLAPPEDWINDOW,
            rect.left+32, rect.top+32, 512, 160,
            hwnd, NULL, NULL, NULL）;
        if（!debug） DestroyWindow（hwnd）;

        ////////////////////////////////////////////////////////
        ///加载字符串，并设置计时器

        paused=loadstring（IDS_PAUSED）;
        queued=loadstring（IDS_QUEUED）;
        checking=loadstring（IDS_CHECKING）;
        connecting=loadstring（IDS_CONNECTING）;
        downloading=loadstring（IDS_DOWNLOADING）;
        seeding=loadstring（IDS_SEEDING）;
        unknown=loadstring（IDS_UNKNOWN）;
        bytes=loadstring（IDS_BYTES）;
        kibibytes=loadstring（IDS_KIBIBYTES）;
        mebibytes=loadstring（IDS_MEBIBYTES）;
        gibibytes=loadstring（IDS_GIBIBYTES）;
```


www.ckook.com

```
        SetTimer（hwnd, TIMER_UPDATE, 1000, NULL）;

        ////////////////////////////////////////////////////////
        ///从上一次会话中恢复 torrent

        path p=getmodulepath（）/"torrents";

        WIN32_FIND_DATAA finddata={0};
        HANDLE find=FindFirstFileA（（p/"*.torrent"）.native_file_string（）.c_str（）, &finddata）;
        if（find!=INVALID_HANDLE_VALUE）  {
            do AddTorrent（hwnd, p/finddata.cFileName）;
            while（FindNextFileA（find, &finddata））;
            FindClose（find）;
        }

        return TRUE;
    }
    //处理关闭程序的消息
    static void Main_OnDestroy（HWND hwnd）  {
        Shell_NotifyIcon（NIM_DELETE, &nid）;

        path p=getmodulepath（）/"resume";
        CreateDirectoryA（p.native_directory_string（）.c_str（）, NULL）;

        for（vector<Torrent>::size_type i=0; i<torrents.size（）; i++）  {
            torrents[i].handle.pause（）;
            libtorrent::entry e=torrents[i].handle.write_resume_data（）;

            bencode（p/torrents[i].file, e）;
        }

        delete session;

        HWND list=GetDlgItem（hwnd, IDC_LIST）;

        for（int i=0; i<columncount; i++）
            conf.columns[i]=ListView_GetColumnWidth（list, i）;

        conf.Save（）;

        PostQuitMessage（0）;
    }
```

www.ckook.com

```
static void Main_OnDropFiles（HWND hwnd, HDROP hdrop）  {
    char file[MAX_PATH];

    unsigned int amount=DragQueryFileA（hdrop, 0xFFFFFFFF, 0, 0）;
    while（amount--）  {
        DragQueryFileA（hdrop, amount, file, MAX_PATH）;
        AddTorrent（hwnd, file）;
    }
    DragFinish（hdrop）;
}
//关闭会话
static void Main_OnEndSession（HWND hwnd, BOOL fEnding）  {
    if（fEnding）  Main_OnDestroy（hwnd）;
}

static BOOL Main_OnEraseBkgnd（HWND hwnd, HDC hdc）  {
    return TRUE;
}

static void Main_OnGetMinMaxInfo（HWND hwnd, LPMINMAXINFO lpMinMaxInfo）  {
    lpMinMaxInfo->ptMinTrackSize.x=512;
    lpMinMaxInfo->ptMinTrackSize.y=200;
}

static LRESULT Main_OnNotify（HWND hwnd, int idCtrl, LPNMHDR lpNmh）  {
    if（lpNmh->idFrom==IDC_LIST）  {
        if（lpNmh->code==LVN_GETDISPINFO）  {
            NMLVDISPINFO *di=（NMLVDISPINFO*）lpNmh;

            if（di->item.iItem<（int）torrents.size（）  && di->item.iSubItem<（int）columncount）
                di->item.pszText=（LPTSTR）torrents[di->item.iItem].cols[di->item.iSubItem].c_str（）;
        }
        else if（lpNmh->code==NM_RCLICK && ListView_GetSelectedCount（lpNmh->hwndFrom）>0）  {
            POINT pt;
            GetCursorPos（&pt）;

            HMENU menu=LoadMenu（GetModuleHandle（NULL）, MAKEINTRESOURCE（IDR_
CONTEXTMENU））;
            HMENU submenu=GetSubMenu（menu, 0）;

            int index=ListView_GetSelectionMark（lpNmh->hwndFrom）;

            if（torrents[index].handle.is_paused（））  {
```


www.ckook.com

```
                EnableMenuItem（submenu, ID_CONTEXT_PAUSE, MF_BYCOMMAND|MF_GRAYED）;
                EnableMenuItem（submenu, ID_CONTEXT_RESUME, MF_BYCOMMAND|MF_
ENABLED）;
            }
            else {
                EnableMenuItem（submenu, ID_CONTEXT_PAUSE, MF_BYCOMMAND|MF _
ENABLED）;
                EnableMenuItem（submenu, ID_CONTEXT_RESUME, MF_BYCOMMAND|MF_
GRAYED）;
            }

            SetForegroundWindow（hwnd）;
            TrackPopupMenu（submenu, TPM_LEFTALIGN|TPM_TOPALIGN, pt.x, pt.y, 0, hwnd, NULL）;
        }
    }
    return 0;
}
//处理调整窗口大小的消息
static void Main_OnSize（HWND hwnd, UINT state, int cx, int cy）  {
    if（state==SIZE_MINIMIZED）  ShowWindow（hwnd, SW_HIDE）;
    else {
        HWND status=GetDlgItem（hwnd, IDC_STATUS）;
        SendMessage（status, WM_SIZE, 0, 0）;

        RECT statusrect;
        GetWindowRect（status, &statusrect）;

        int parts[3];
        parts[0]=std::max（（（statusrect.right-statusrect.left）-192）,（LONG）0）;
        parts[1]=parts[0]+96;
        parts[2]=parts[1]+96;

        SendMessage（status, SB_SETPARTS, 3,（LPARAM）parts）;

        MoveWindow(GetDlgItem(hwnd, IDC_LIST), 0, 0, cx, cy-(statusrect.bottom-statusrect.top), TRUE);

        if（state==SIZE_RESTORED）  {
            RECT rc;
            GetWindowRect（hwnd, &rc）;

            if（rc.left>=0 && rc.top>=0 && rc.right>=0 && rc.bottom>=0）
                conf.position=rc;
        }
```

www.ckook.com

```
        }
    }
    //处理计数器消息
    static void Main_OnTimer（HWND hwnd, UINT id） {
        if（id==TIMER_UPDATE） {
            ////////////////////////////////////////////////////////
            /// 更新状态栏

            wchar_t buf[32];
            HWND status=GetDlgItem（hwnd, IDC_STATUS）;
            libtorrent::session_status s=session->status（）;

            StringCchPrintf（buf, 32, L"D: %s/s", strsize（s.download_rate））;
            SendMessage（status, SB_SETTEXT, 1|SBitTorrent_NOBORDERS, （LPARAM）buf）;

            StringCchPrintf（buf, 32, L"U: %s/s", strsize（s.upload_rate））;
            SendMessage（status, SB_SETTEXT, 2|SBitTorrent_NOBORDERS, （LPARAM）buf）;

            HWND list=GetDlgItem（hwnd, IDC_LIST）;
            for（vector<Torrent>::size_type i=0; i<torrents.size（）; i++） {
                try {
                    ////////////////////////////////////////////////////////
                    /// 更新 ListView 中的数据

                    libtorrent::torrent_status status=torrents[i].handle.status（）;

                    torrents[i].cols[2]=strsize（（double）status.total_done）;
                    torrents[i].cols[3]=strsize（（double）status.total_upload）;

                    if（status.paused） torrents[i].cols[4]=paused;
                    else switch（status.state） {
                        case libtorrent::torrent_status::queued_for_checking:
                            torrents[i].cols[4]=queued;
                            break;
                        case libtorrent::torrent_status::checking_files:
                            torrents[i].cols[4]=checking;
                            break;
                        case libtorrent::torrent_status::connecting_to_tracker:
                            torrents[i].cols[4]=connecting;
                            break;
                        case libtorrent::torrent_status::downloading:
                        case libtorrent::torrent_status::downloading_metadata:
                            torrents[i].cols[4]=downloading;
```


www.ckook.com

```
                    break;
                case libtorrent::torrent_status::seeding:
                    torrents[i].cols[4]=seeding;
                    break;
                default:
                    torrents[i].cols[4]=unknown;
                    break;
            }

            StringCchPrintf（buf, 32, L"%.1f%%", （double）status.progress*100.0）;
            torrents[i].cols[5]=buf;

            StringCchPrintf（buf, 32, L"%s/s", strsize（status.download_rate））;
            torrents[i].cols[6]=buf;

            StringCchPrintf（buf, 32, L"%s/s", strsize（status.upload_rate））;
            torrents[i].cols[7]=buf;

            StringCchPrintf（buf, 32, L"%d%%", （int）（status.distributed_copies*100.0f））;
            torrents[i].cols[8]=buf;

            torrents[i].cols[9]=_itow（status.num_seeds, buf, 10）;
            torrents[i].cols[10]=_itow（status.num_peers, buf, 10）;

            InvalidateRect（list, NULL, FALSE）;

            for（auto_ptr<libtorrent::alert> a=session->pop_alert（）; a.get（）; a=session->pop
_alert（））{
                string timestamp=to_simple_string（a->timestamp（）.time_of_day（））;
                string message=a->msg（）;

                wstring timestamp_wcs=mbstowcs（timestamp）;
                wstring message_wcs=mbstowcs（message）;

                AddDebugMessageData data;
                data.time=（LPWSTR）timestamp_wcs.c_str（）;
                data.message=（LPWSTR）message_wcs.c_str（）;

                SendMessage(debug, WM_ARCTIC_ADDDEBUGMESSAGE, 0, （LPARAM）
&data）;
            }
        }
        catch（exception &ex）{
```

www.ckook.com

```
                wstring text=mbstowcs（ex.what（））;
                wstring caption=loadstring（IDS_EXCEPTION）;

                MessageBox（hwnd, text.c_str（）, caption.c_str（）, MB_ICONERROR|MB_OK）;
                continue;
            }
        }
    }
}

static void Main_OnDDEInitiate（HWND hwnd, HWND client, ATOM appatom, ATOM topicatom）{
    ATOM a=GlobalAddAtom（L"ArcticTorrent"）;
    ATOM t=GlobalAddAtom（L"system"）;

    if（appatom==a && topicatom==t）
        SendMessage（client, WM_DDE_ACK,（WPARAM）hwnd, MAKELPARAM（a,t））;

    GlobalDeleteAtom（a）;
    GlobalDeleteAtom（t）;
}

static bool parse_ddeopen（vector<wstring> &res, const wstring &str）{
    return parse（str.begin（）, str.end（）,
        L"open（"> >（（""> >（+~ch_p（""））[push_back_a（res）]> > ""）%','）> > ')'
    ）.full;
}

static void Main_OnDDEExecute（HWND hwnd, HWND client, HGLOBAL hCmdStr）{
    wstring command（（const wchar_t*）GlobalLock（hCmdStr））;
    GlobalUnlock（hCmdStr）;

    vector<wstring> res;
    if（parse_ddeopen（res, command））{
        for（vector<wstring>::size_type i=0; i<res.size（）; i++）
            AddTorrent（hwnd, wcstombs（res[i]））;
    }

    LPARAM ack=PackDDElParam（WM_DDE_ACK, 1,（UINT_PTR）hCmdStr）;
    PostMessage（client, WM_DDE_ACK,（WPARAM）hwnd, ack）;
}

static void Main_OnDDETerminate（HWND hwnd, HWND client）{
    PostMessage（client, WM_DDE_TERMINATE,（WPARAM）hwnd, 0）;
```


www.ckook.com

```
}

static void Main_OnTray（HWND hwnd, UINT idTray, UINT msg）{
    if（idTray==TRAY_ARCTIC）{
        if（msg==WM_LBUTTONUP）{
            ShowWindow（hwnd, SW_SHOWNORMAL）;
            SetForegroundWindow（hwnd）;
        }
        else if（msg==WM_RBUTTONUP）{
            POINT pt;
            GetCursorPos（&pt）;

            HMENU menu=LoadMenu（GetModuleHandle（NULL）, MAKEINTRESOURCE（IDR
_ TRAYMENU））;
            HMENU submenu=GetSubMenu（menu, 0）;

            if（allpaused）{
                EnableMenuItem（submenu, ID_TRAY_PAUSE, MF_ BYCOMMAND|MF_ GRAYED）;
                EnableMenuItem（submenu, ID_ TRAY_RESUME, MF_ BYCOMMAND|MF _ENABLED）;
            }
            else {
                EnableMenuItem（submenu, ID_TRAY_PAUSE, MF_BYCOMMAND|MF_ ENABLED）;
                EnableMenuItem （ submenu, ID_TRAY_RESUME, MF_BYCOMMAND|MF_
GRAYED）;
            }

            SetForegroundWindow（hwnd）;
            TrackPopupMenu（submenu, TPM_LEFTALIGN|TPM_TOPALIGN, pt.x, pt.y, 0, hwnd, NULL）;
        }
    }
}
//窗口回调函数
LRESULT CALLBACK Main_WndProc（HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam）{
    switch（msg）{
        HANDLE_MSG（hwnd, WM_CLOSE, Main_OnClose）;
        HANDLE_MSG（hwnd, WM_COMMAND, Main_OnCommand）;
        HANDLE_MSG（hwnd, WM_CREATE, Main_OnCreate）;
        HANDLE_MSG（hwnd, WM_DESTROY, Main_OnDestroy）;
        HANDLE_MSG（hwnd, WM_DROPFILES, Main_OnDropFiles）;
        HANDLE_MSG（hwnd, WM_ERASEBKGND, Main_OnEraseBkgnd）;
        HANDLE_MSG（hwnd, WM_GETMINMAXINFO, Main_OnGetMinMaxInfo）;
        HANDLE_MSG（hwnd, WM_NOTIFY, Main_OnNotify）;
        HANDLE_MSG（hwnd, WM_SIZE, Main_OnSize）;
```

www.ckook.com

```
        HANDLE_MSG（hwnd, WM_TIMER, Main_OnTimer）;
        case WM_DDE_INITIATE:
            Main_OnDDEInitiate（hwnd, （HWND）wParam, LOWORD（lParam）, HIWORD（lParam））;
            return 0;
        case WM_DDE_EXECUTE:
            Main_OnDDEExecute（hwnd, （HWND）wParam, （HGLOBAL）lParam）;
            return 0;
        case WM_DDE_TERMINATE:
            Main_OnDDETerminate（hwnd, （HWND）wParam）;
            return 0;
        case WM_ARCTIC_TRAY:
            Main_OnTray（hwnd, （UINT）wParam, （UINT）lParam）;
            return 0;
        default:
            if（msg==traycreatedmessage） {
                Shell_NotifyIcon（NIM_ADD, &nid）;
                return 0;
            }
            else return DefWindowProc（hwnd, msg, wParam, lParam）;
    }
}
```

8.4.2 LibTorrent 库代码分析

Torrent 类在 LibTorrent 中是一个用于保存一个下载信息的类。在文件的下载过程中，Torrent 对象会更新自己的各个成员变量。

www.zbcin.com

```
/*

Copyright （c） 2003, Arvid Norberg
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in
  the documentation and/or other materials provided with the distribution.
* Neither the name of the author nor the names of its
  contributors may be used to endorse or promote products derived
```


www.ckook.com

```
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES （INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION） HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT （INCLUDING NEGLIGENCE OR OTHERWISE）
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

*/

#ifndef TORRENT_TORRENT_HPP_INCLUDE
#define TORRENT_TORRENT_HPP_INCLUDE
//Torrent 类使用到 STL 的容器和算法
#include <algorithm>
#include <vector>
#include <set>
#include <list>
#include <iostream>

#ifdef _MSC_VER
#pragma warning（push, 1）
#endif
//还使用到 boost 库
#include <boost/limits.hpp>
#include <boost/filesystem/path.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include <boost/tuple/tuple.hpp>

#ifdef _MSC_VER
#pragma warning（pop）
#endif

#include "libtorrent/torrent_handle.hpp"
#include "libtorrent/entry.hpp"
#include "libtorrent/torrent_info.hpp"
#include "libtorrent/socket.hpp"
#include "libtorrent/policy.hpp"
#include "libtorrent/tracker_manager.hpp"
```

www.docin.com

www.ckook.com

```
#include "libtorrent/stat.hpp"
#include "libtorrent/alert.hpp"
#include "libtorrent/resource_request.hpp"

namespace libtorrent
{
#if defined（TORRENT_VERBOSE_LOGGING） || defined（TORRENT_LOGGING）
    struct logger;
#endif

class piece_manager;

std::string escape_string（const char* str, int len）;
std::string unescape_string（std::string const& s）;


namespace detail
{
    struct session_impl;
    struct piece_checker_data;
}

int div_round_up（int numerator, int denominator）;
std::pair<int, int> req_to_offset（std::pair<int, int> req, int total_size）;
std::pair<int, int> offset_to_req（std::pair<int, int> offset, int total_size）;

// torrent 类
class torrent: public request_callback
{
public:
//两个构造函数
    torrent（detail::session_impl& ses, entry const& metadata, boost::filesystem::path const& save_path, address const& net_interface, bool compact_mode, int block_size）;
    torrent（detail::session_impl& ses, char const* tracker_url, sha1_hash const& info_hash, boost::filesystem::path const& save_path, address const& net_interface, bool compact_mode, int block_size）;
    ~torrent（）;//析构函数

    void init（）;//初始化函数，此函数会初始化存储等

    // this will flag the torrent as aborted. The main loop in session_impl will check for this state on all torrents once every second, and take the necessary actions then.
    void abort（）;
    bool is_aborted（） const { return m_abort; }
```


www.ckook.com

```
void second_tick（stat& accumulator）; //sesssion 会每隔几秒调用此函数，此函数会计算上
//行下行的总数据量，以及连接数
void print（std::ostream& os） const; //为了调试目的
void metadata_progress（int total_size, int received）; //peer_connection 类每收到一块
//metadata 都会调用此函数
void check_files(detail::piece_checker_data& data, boost::mutex& mutex, bool lock_session = true);

stat statistics（） const { return m_stat; }
size_type bytes_left（） const;
boost::tuple<size_type, size_type> bytes_done（） const;

void pause（）;
void resume（）;
bool is_paused（） const { return m_paused; }

void filter_piece（int index, bool filter）;
void filter_pieces（std::vector<bool> const& bitmask）;
bool is_piece_filtered（int index） const;
void filtered_pieces（std::vector<bool>& bitmask） const;

//idea from Arvid and MooPolice todo refactoring and improving the function body marks the
file with the given index as filtered it will not be downloaded
void filter_file（int index, bool filter）;
void filter_files（std::vector<bool> const& files）;

torrent_status status（） const;

void use_interface（const char* net_interface）;
peer_connection& connect_to_peer（const address& a）;
void set_ratio（float ratio）{ assert（ratio >= 0.0f）; m_ratio = ratio; }
float ratio（） const{ return m_ratio; }

// 以下是负责对等点管理的函数
void attach_peer（peer_connection* p）; //peer_connection 对象调用此函数并把它自己与此
//torrent 类相关联起来
void remove_peer（peer_connection* p）; //peer_connection 对象析构时调用此函数

peer_connection* connection_for（const address& a）
{
    peer_iterator i = m_connections.find（a）;
    if （i == m_connections.end（））  return 0;
    return i->second;}
```

www.ckook.com

```
        // 返回有多少用户正在下载此文件
        int num_peers（）  const { return  （int）m_connections.size（）；}
        int num_seeds（）  const;

        typedef std::map<address, peer_connection*>::iterator peer_iterator;
        typedef std::map<address, peer_connection*>::const_iterator const_peer_iterator;

        const_peer_iterator begin（）  const { return m_connections.begin（）；}
        const_peer_iterator end（）  const { return m_connections.end（）；}
        peer_iterator begin（）  { return m_connections.begin（）；}
        peer_iterator end（）  { return m_connections.end（）；}

        //以下是负责 Tracker 管理的函数
//下面是一组 tracker_connection 对象会调用的函数。表示 torrent 收到的消息是请求或者错误回复
        virtual void tracker_response（
            tracker_request const& r
            , std::vector<peer_entry>& e, int interval
            , int complete, int incomplete）；
        virtual void tracker_request_timed_out（
            tracker_request const& r）；
        virtual void tracker_request_error（tracker_request const& r
            , int response_code, const std::string& str）；
        virtual void tracker_warning（std::string const& msg）；

        tracker_request generate_tracker_request（）；//产生请求消息
        std::string tracker_login（）  const;

        boost::posix_time::ptime next_announce（）  const; //返回下一个 tracker announce 的绝对时间
        bool should_request（）；
        void force_tracker_request（）；
        void force_tracker_request（boost::posix_time::ptime）；
        void set_tracker_login（std::string const& name, std::string const& pw）；
        const address& current_tracker（）  const;

        //以下是负责数据块管理的函数

        //返回指定的数据块是否已经下载成功
        bool have_piece（int index）  const{return m_have_pieces[index];}
        const std::vector<bool>& pieces（）  const{ return m_have_pieces; }
        int num_pieces（）  const { return m_num_pieces; }

        // when we get a have- or bitfield- messages, this is called for every piece a peer has gained.
```


www.ckook.com

```
void peer_has（int index）{m_picker->inc_refcount（index）;}

// when peer disconnects, this is called for every piece it had
void peer_lost（int index）{m_picker->dec_refcount（index）;}

int block_size（）  const { return m_block_size; }
void announce_piece（int index）; //告诉 piece picker 哪一块已经下载完成

void disconnect_all（）;
void completed（）; //下载完成时调用，并向 tracker 发送消息告诉它我们已经下载完成
void finished（）; //调用 completed（）之前调用

bool verify_piece（int piece_index）;
void piece_failed（int index）; // peer_connection 会调用此函数，如果收到的数据块没有通
//过散列校验

float priority（）  const{ return m_priority; }
void set_priority（float p）{m_priority = p;}

bool is_seed（）  const{return valid_metadata（）&& m_num_pieces == m_torrent_file.num_pieces（）;}

boost::filesystem::path save_path（）  const;
alert_manager& alerts（）  const;
piece_picker& picker（）{return *m_picker;}
policy& get_policy（）  { return *m_policy; }
piece_manager& filesystem（）;
torrent_info const& torrent_file（）  const { return m_torrent_file; }
std::vector<announce_entry> const& trackers（）  const{ return m_trackers; }

void replace_trackers（std::vector<announce_entry> const& urls）;
torrent_handle get_handle（）  const;

// LOGGING
#if defined（TORRENT_VERBOSE_LOGGING）  || defined（TORRENT_LOGGING）
logger* spawn_logger（const char* title）;

virtual void debug_log（const std::string& line）;
#endif

// DEBUG
#ifndef NDEBUG
void check_invariant（）  const;
#endif
```

www.ckook.com

```
// 下面是资源管理相关的函数及变量

// this will distribute the given upload/download
// quotas and number of connections, among the peers
void distribute_resources（）；

resource_request m_ul_bandwidth_quota;
resource_request m_dl_bandwidth_quota;
resource_request m_uploads_quota;
resource_request m_connections_quota;

void set_upload_limit（int limit）；
void set_download_limit（int limit）；
void set_max_uploads（int limit）；
void set_max_connections（int limit）；
bool move_storage（boost::filesystem::path const& save_path）；

bool valid_metadata（）  const { return m_storage.get（）  != 0; }
std::vector<char> const& metadata（）  const { return m_metadata; }

bool received_metadata（
    char const* buf
    , int size
    , int offset
    , int total_size）；

// returns a range of the metadata that
// we should request.
std::pair<int, int> metadata_request（）；
void cancel_metadata_request（std::pair<int, int> req）；

private:

void try_next_tracker（）；
int prioritize_tracker（int tracker_index）；

torrent_info m_torrent_file;

// is set to true when the torrent has
// been aborted.
bool m_abort;
```

www.docin.com

www.ckook.com

```
// is true if this torrent has been paused
bool m_paused;
// this is true from the time when the torrent was
// paused to the time should_request（）  is called
bool m_just_paused;

tracker_request::event_t m_event;

void parse_response（const entry& e, std::vector<peer_entry>& peer_list）;

// the size of a request block
// each piece is divided into these
// blocks when requested
int m_block_size;

// if this pointer is 0, the torrent is in
// a state where the metadata hasn't been
// received yet.
std::auto_ptr<piece_manager> m_storage;

// the time of next tracker request
boost::posix_time::ptime m_next_request;

// ---------------------------
// DATA FROM TRACKER RESPONSE

// the number number of seconds between requests
// from the tracker
int m_duration;

// the scrape data from the tracker response, this
// is optional and may be -1.
int m_complete;
int m_incomplete;

std::map<address, peer_connection*> m_connections;

// this is the upload and download statistics for the whole torrent.
// it's updated from all its peers once every second.
libtorrent::stat m_stat;

// ---------------------------
```

www.docin.com

```
boost::shared_ptr<policy> m_policy;

// a back reference to the session
// this torrent belongs to.
detail::session_impl& m_ses;

std::auto_ptr<piece_picker> m_picker;

std::vector<announce_entry> m_trackers;
// this is an index into m_torrent_file.trackers（）
int m_last_working_tracker;
int m_currently_trying_tracker;
// the number of connection attempts that has
// failed in a row, this is currently used to
// determine the timeout until next try.
int m_failed_trackers;

// this is a counter that is increased every
// second, and when it reaches 10, the policy::pulse（）
// is called and the time scaler is reset to 0.
int m_time_scaler;

// this is the priority of this torrent. It is used
// to weight the assigned upload bandwidth between peers
// it should be within the range [0, 1]
float m_priority;

// the bitmask that says which pieces we have
std::vector<bool> m_have_pieces;

// the number of pieces we have. The same as
// std::accumulate（m_have_pieces.begin（），
// m_have_pieces.end（），0）
int m_num_pieces;

// is false by default and set to
// true when the first tracker reponse
// is received
bool m_got_tracker_response;

// the upload/download ratio that each peer
// tries to maintain.
// 0 is infinite
```

www.docin.com

www.ckook.com

```
float m_ratio;

// the number of bytes that has been
// downloaded that failed the hash-test
size_type m_total_failed_bytes;

std::string m_username;
std::string m_password;

// the network interface all outgoing connections
// are opened through
address m_net_interface;

// the max number of bytes this torrent
// can upload per second
int m_upload_bandwidth_limit;
int m_download_bandwidth_limit;

// this buffer is filled with the info-section of
// the metadata file while downloading it from
// peers, and while sending it.
std::vector<char> m_metadata;

// this is a bitfield of size 256, each bit represents
// a piece of the metadata. It is set to one if we
// have that piece. This vector may be empty
//  (size 0)  if we haven't received any metadata
// or if we already have all metadata
std::vector<bool> m_have_metadata;
// this vector keeps track of how many times each meatdata
// block has been requested
std::vector<int> m_requested_metadata;

boost::filesystem::path m_save_path;

// determines the storage state for this torrent.
const bool m_compact_mode;

int m_metadata_progress;
int m_metadata_size;

// defaults to 16 kiB, but can be set by the user
// when creating the torrent
const int m_default_block_size;
```

www.docin.com

```
};

inline boost::posix_time::ptime torrent::next_announce（）  const
{
    return m_next_request;
}

inline void torrent::force_tracker_request（）
{
    using boost::posix_time::second_clock;
    m_next_request = second_clock::universal_time（）;
}

inline void torrent::force_tracker_request（boost::posix_time::ptime t）
{
    namespace time = boost::posix_time;
    m_next_request = t;
}

inline void torrent::set_tracker_login（std::string const& name, std::string const& pw）
{
    m_username = name;
    m_password = pw;
}

}

#endif // TORRENT_TORRENT_HPP_INCLUDED
```

www.docin.com

8.5 BitTorrent 系统部署

BitTorrent 完成一次下载，至少需要一个 Tracker 服务器和一个种子，如图 8-10 所示。Tracker 负责帮助 Peers 之间相互建立连接。Peer 是要求下载文件的用户。种子是被下载文件的拥有者。

BitTorrent 通过一个扩展名为 torrent 的文件进行下载部署，Torrent 文件放在一个普通的 Web 服务器上。Torrent 文件包含了要共享的文件的信息，包括文件名、大小、文件的散列信息和一个指向 Tracker 的 URL。下载者通过使用 BitTorrent 客户端打开 Torrent 文件同 Tracker 进行联系。

BitTorrent 完成一次下载部署的过程是：

图 8-10　BitTorrent 下载部署

www.ckook.com

（1）第一个种子向 Tracker 注册，等待为别人提供文件。

（2）第一个 Peer 向 Tracker 注册，并取得种子的信息。

（3）Peer 与种子建立连接，通知 Tracker 自己要下载的文件、自己使用的端口以及其他信息，并从种子处读取文件；由于原始的文件，只有种子拥有，所以种子至少要上传原始文件的一份完整拷贝。

（4）当其他 Peer 加入进来，Tracker 负责帮助 Peer 获取种子和其他 Peers 的信息，Peer 利用这些信息和种子以及前一个 Peer 建立连接，然后从这两者处获取文件，并上传自己拥有的文件片段。

8.5.1 BitTorrent 服务器的部署

架设一个 BitTorrent 服务器需要以下步骤：

（1）运行 Tracker；

（2）运行普通网页服务器程序，如 Apache；

（3）在网页服务器上将.torrent 文件关联到 Mimetype 类型 application/x-bittorrent；

（4）使用要发布的文件和 Tracker 的 URL 创建一个.torrent 文件；

（5）将制作好的 Torrent 文件放置在网页服务器上；

（6）在网页上发布 Torrent 文件的链接；

（7）原始下载者提供完整的供下载文件。

8.5.2 BitTorrent 客户端的部署

通过 BitTorrent 下载需要以下步骤：

（1）安装 BitTorrent 客户端程序（已安装的跳过这一步）；

（2）连接上网络；

（3）点击网页上一个到 Torrent 文件的链接；

（4）选择本地存储路径，选定需要下载的文件（针对有选择下载功能的 BitTorrent 客户端用户）；

（5）等待下载完成；

（6）退出 BitTorrent 客户端程序（在下载完成后，BitTorrent 客户端程序会继续为其他节点提供数据，直到用户手动关闭客户端）。

www.docin.com

8.6 BitTorrent 系统分析

BitTorrent 是 P2P 应用的典型代表，拥有广泛的用户，在整个因特网的流量当中占了相当大的比例。

BitTorrent 最大的优势就是将下载负荷分散到各个用户端，文件进行分片段下载，从而在理论上支持海量用户同时使用该网络。

www.ckook.com

通过分析 BitTorrent 的实现有助于开发特定的 P2P 文件共享系统，同时可以在 BitTorrent 的基础上修改出自己需要的共享功能。现在已经有很多种编程语言实现的类 BitTorrent 软件，而且大多数都对其进行了增强和改进。目前已经有开发人员开发出了基于 BitTorrent 的 P2P 流媒体应用。

8.7 本章总结

本章介绍了一个典型的 P2P 应用系统——BitTorrent。在分析了 BitTorrent 系统的原理之后，又分析了一个简单的 BitTorrent 客户端源码并分析了开源代码库 LibTorrent 的结构和部分内容。最后介绍了一个 BitTorrent 系统的部署。

8.8 练习题

（1）简述 BitTorrent 的工作原理。
（2）分析.torrent 文件的结构。
（3）部署一个 BitTorrent 服务器供大家使用。
（4）设计一个 BitTorrent 客户端程序。

参 考 文 献

[1] BitTorrent Protocol Specification. http://www.bittorrent.org/protocol.html, 2005
[2] Arctic Home Page. http://dev.int64.org/arctic.html, 2007
[3] LibTorrent Home Page. http://libtorrent.sourceforge.net, 2007
[4] Petzold C. Programming Windows.Microsoft Press, 1998

www.docin.com

www.ckook.com

CHAPTER 9

第 9 章　P2P 即时通信系统开发实例——Hermes 的设计与实现

Hermes 是本书作者设计开发的一款简单的 P2P 即时通信软件。Hermes 在希腊神话中是“神的信使”的含义。本章通过介绍 Hermes 即时通信系统的设计和编码，希望读者能掌握即时通信系统设计的普遍方法。

在学习了本章后，希望读者能够掌握以下几点：

（1）即时通信系统的概念；

（2）如何从整体上来考虑一个即时通信系统的设计。

9.1　Hermes 系统概述

9.1.1　P2P 即时通信系统的背景

www.dbcin.com

最早的即时通信系统可以说是从 ICQ 开始。1996 年 7 月，4 个以色列年轻人成立了 Mirabilis 公司，并在同一年推出了全球第一个即时通信软件 ICQ，意思为“我在找你”——“I Seek You”。从那以后，各种各样的即时通信系统都很快冒出来了，目前比较流行的有 MSN、雅虎通、网易泡泡、腾讯 QQ 以及 GTalk 等。

在因特网上，相对于传统的 BBS、E-mail 等通信方式来说，即时通信不仅节省费用，而且效率更高。首先，即时通信具有很高的互动性，用户可以很清楚地知道通信对方的状态，而 E-mail 虽然也能很快到达，但是很可能收信人已经不在线了；其次，即时通信除了传送传统的文本信息外还可以传送多种媒体，如语音、视频和图片等，而 E-mail 只能发送文本，其他要以附件的方式来发送，用户之间的交流具有很大的局限性。

随着网络的发展，即时通信系统已经逐渐不再局限于传统的娱乐应用，而是逐渐地向商业应用转化，有可能成为一种新的商业模式。目前，微软 MSN 在中国已经进入了企业办公领域，阿里巴巴（雅虎中国）推出的“贸易通”，明显针对商用市场，就连一向专注于网络娱乐的腾讯也为此推出了其企业即时通信产品 RTX。

www.ckook.com

但由于即时通信系统最初是为了娱乐而设计的，当上升到商务层面后，就出现了很多问题。安全和互联互通是当今即时通信系统面临的首要问题。

在安全方面，无论是个人用户，还是企业用户，都面临新的威胁。对于个人用户来说，通过即时通信软件传播的病毒就像潜藏的炸弹，一旦爆发，轻则资料丢失，重则电脑瘫痪，更有甚者，会造成即时通信用户之间的误会。而对于企业用户来说，安全问题更多地体现在商业秘密安全上。大多数即时通信系统是公开的，这意味着一个用户只要知道对方的即时通信地址就可以发送消息，非常有利于恶意员工向外发送企业的商业秘密，而且即时通信的内容不经过服务器中转，使得网络监管非常困难。因为安全的问题，很多企业禁止员工在上班时间使用即时通信系统。虽然这样可以暂时防止商业泄密，但同时也限制了员工之间的正常交流。不过对于即时通信系统提供商来说，只要有足够的用户群和商业利益，安全问题可以比较好地解决。

互联互通主要体现在不同的即时通信系统使用了不同的协议和命名方式。如 QQ 使用 QQ 号码，而 MSN 使用用户的邮箱，而且 QQ 和 MSN 使用的通信协议也互不相同。互联互通最大的难点在于即时通信企业之间的利益分配。因为多数即时通信软件的用户都希望通过即时通信软件寻找网友实现网络交流，当然会首先选择使用已经拥有相当多使用者的即时通信软件。由此，“强者愈强，弱者愈弱”。对于腾讯 QQ、MSN 这样已经拥有数百万用户的企业而言，他们更愿意将精力放在对自身即时通信产品的进一步研发上，产品间的互联互通在一定程度上会分流这些企业的潜在用户甚至现有用户。

即时通信系统商业化应用还存在着这样或那样的问题，但在娱乐领域即时通信已经成为了人们日常生活不可缺少的一部分。而即时通信在娱乐领域的流行势必会带来它在商业领域的发展。

9.1.2 Hermes 系统设计目标

www.docin.com

Hermes 是一款专注于 P2P 语音服务的软件系统。Hermes 支持使用多种语音编码格式传输语音数据，包括 PCM、G.723、GSM610 等。Hermes 是基于 Windows 开发的，使用 Windows Audio 系统作为其语音采集和播放的基础。

9.2 Hermes 系统原理

虽然目前存在着多个即时通信系统，但是我们对这些系统做简单的分析后，不难发现它们之间的一些共同特点。

从结构上来说，几乎所有的即时通信系统都可以划分为服务器端和客户端两个部分。

服务器端是即时通信系统的控制中心，主要负责用户的注册、用户好友列表的保存、用户在线消息收集、离线消息处理以及其他辅助作用。从功能上看，服务器端是整个即时通信系统的心脏，它的正常运转决定着用户之间是否能正常交流。所以，服务器端一般都用服务器集群来实现，以最大程度地保证系统的稳定性。例如，QQ 用户登录时，会先用 UDP 的方

www.ckook.com

式来依次连接多个服务器，如果登录成功就进行下一步操作，否则再用 TCP 的方式来连接多个服务器。只有多次登录都失败后 QQ 客户端才会判断为登录失败。

客户端是即时通信系统和用户之间的接口，一般可以分为用户界面、好友信息处理和通信 3 个模块。用户界面模块主要负责客户端和用户的交互，如选择好友聊天、添加好友以及删除好友等；好友信息处理模块主要负责管理好友列表和好友在线信息；通信模块负责在两个或多个用户之间传送消息，如文本消息、文件或者语音等。

典型的即时通信系统结构如图 9-1 所示。在图 9-1 中，虚线代表客户端和服务器之间的通信，实线代表客户端之间的通信。客户端和服务器之间的通信主要包括用户登录、注册、获取好友列表和状态、发送自身状态消息以及离线消息转发等；客户端之间的通信是即时通信系统中真正的即时消息发送，内容可以包括文本消息、图片以及视频等。

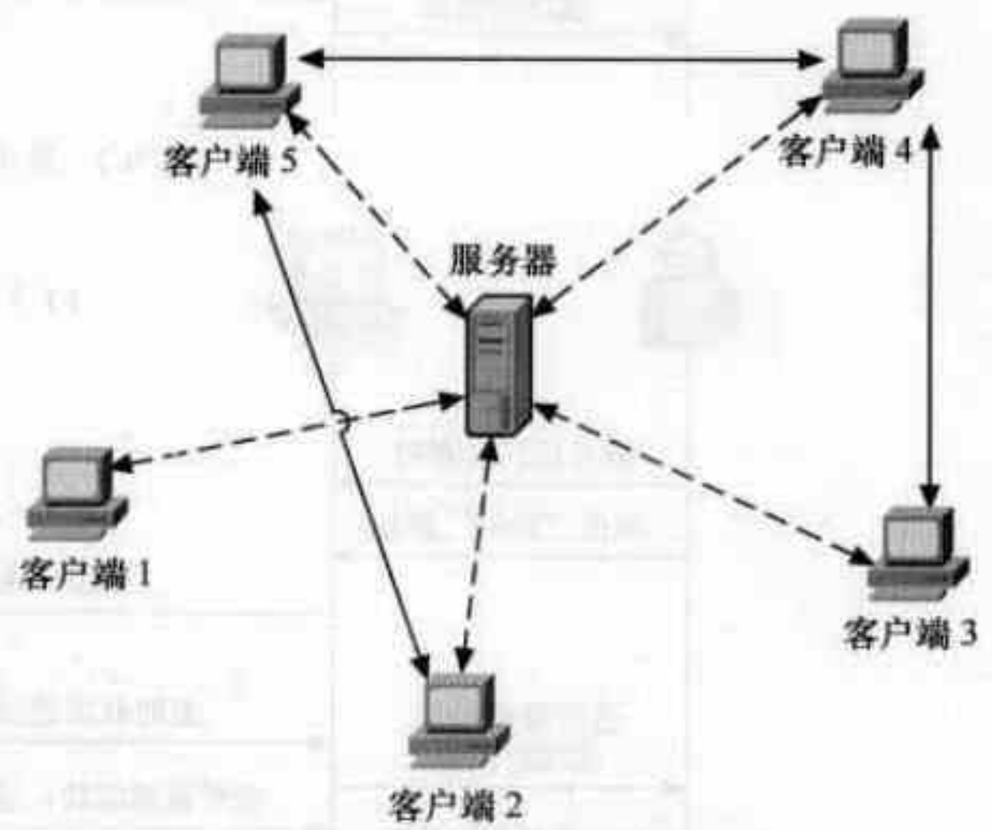


图 9-1 典型的即时通信系统结构图

目前几乎所有的即时通信系统都是基于客户端/服务器模式的，用户在使用之前首先要登录服务器验证用户名和密码，然后才能获取好友列表进行进一步通信。采用这样的架构主要基于两个目的。一是为了商业利益，即时通信提供商需要控制网络以便推行一些增值服务，如腾讯 QQ 的很多其他应用；二是为了穿越防火墙和 NAT，使得处于防火墙和 NAT 后面的用户也能使用即时通信。

Hermes 语音系统也是采用这种模式，用户必须首先登录服务器才能获取在线好友列表，而用户间的语音通信并不通过服务器转发。

www.docin.com

9.2.1 登录认证机制

Hermes 的服务器端维护着注册用户列表，当一个用户想要加入 Hermes 系统时，它需要首先安装 Hermes 客户端程序，然后单击 Hermes 客户端程序上的“新用户”按钮。Hermes 客户端会要求用户输入新用户名与密码，并通过注册消息将新用户名与密码发送给 Hermes 服务器。Hermes 服务器在收到注册消息之后首先检查此用户名是否已被注册，如果此用户名已被注册，服务器就向客户端发送用户名已存在的错误消息。如果用户名不存在，服务器把新的用户名以及通过散列函数计算之后的密码保存入数据库中。图 9-2 所示描绘了新用户注册的流程。

当用户通过注册之后就可以登录到 Hermes 系统当中。首先在 Hermes 客户端键入自己的用户名与密码，然后点击登录。Hermes 客户端会随之发出登录消息给 Hermes 服务器，服务器在收到登录消息以后验证用户名与密码的正确性。如果通过了验证，Hermes 服务器将向用户发送当前他的在线好友列表，列表中包括了好友的 IP 地址。图 9-3 所示描绘了用户登录的流程。

www.ckook.com

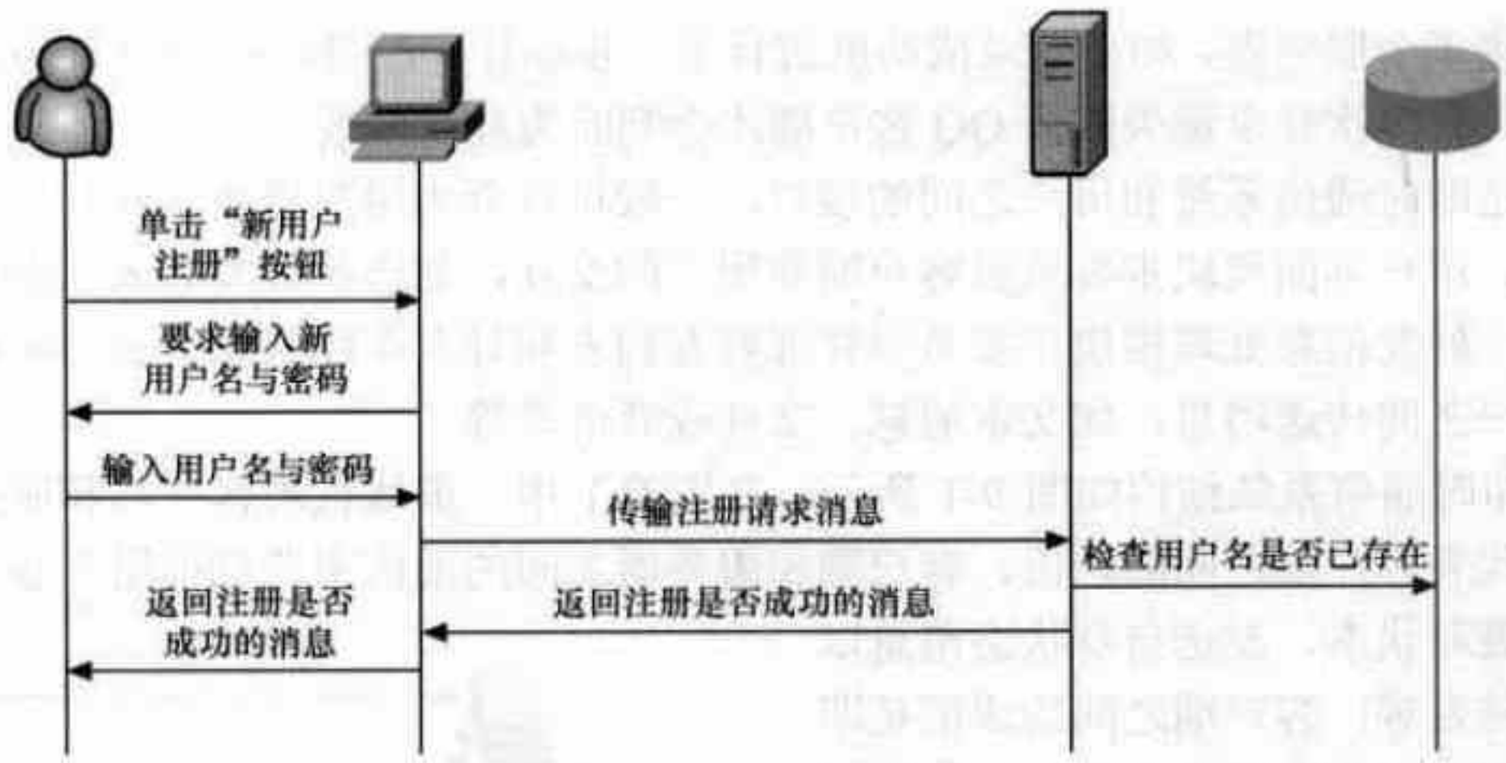


图 9-2　新用户注册流程

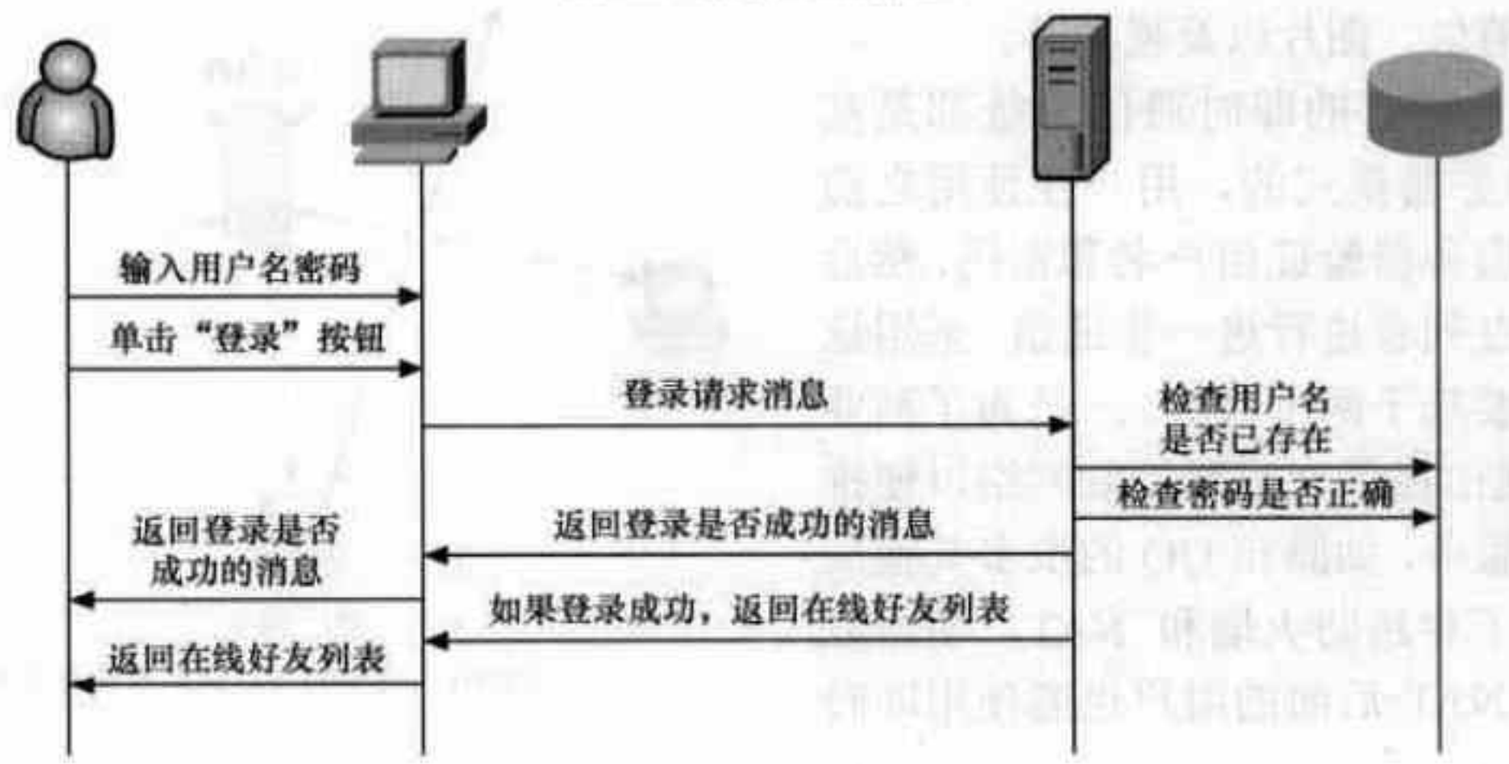


图 9-3　登录流程

当用户成功登录 Hermes 系统之后，就可以呼叫在线好友了。在 Hermes 客户端的在线好友列表中选择一个好友，然后点击“呼叫”。Hermes 客户端会向被叫方发送一个呼叫请求消息，被叫方的 Hermes 客户端在收到回叫请求之后会在界面上显示呼入请求以及振铃响，被叫用户决定是否接听来电。如果被叫用户决定接听来电，被叫 Hermes 客户端会向主叫 Hermes 客户端发送同意呼叫的消息。同时，主被叫的 Hermes 客户端都会打开输入输出语音设备，比如麦克风、扬声器等。此时主被叫的 Hermes 客户端之间就建立起了可靠的通信连接，并且不断地发送和接收语音数据。图 9-4 所示描绘了用户呼叫流程。

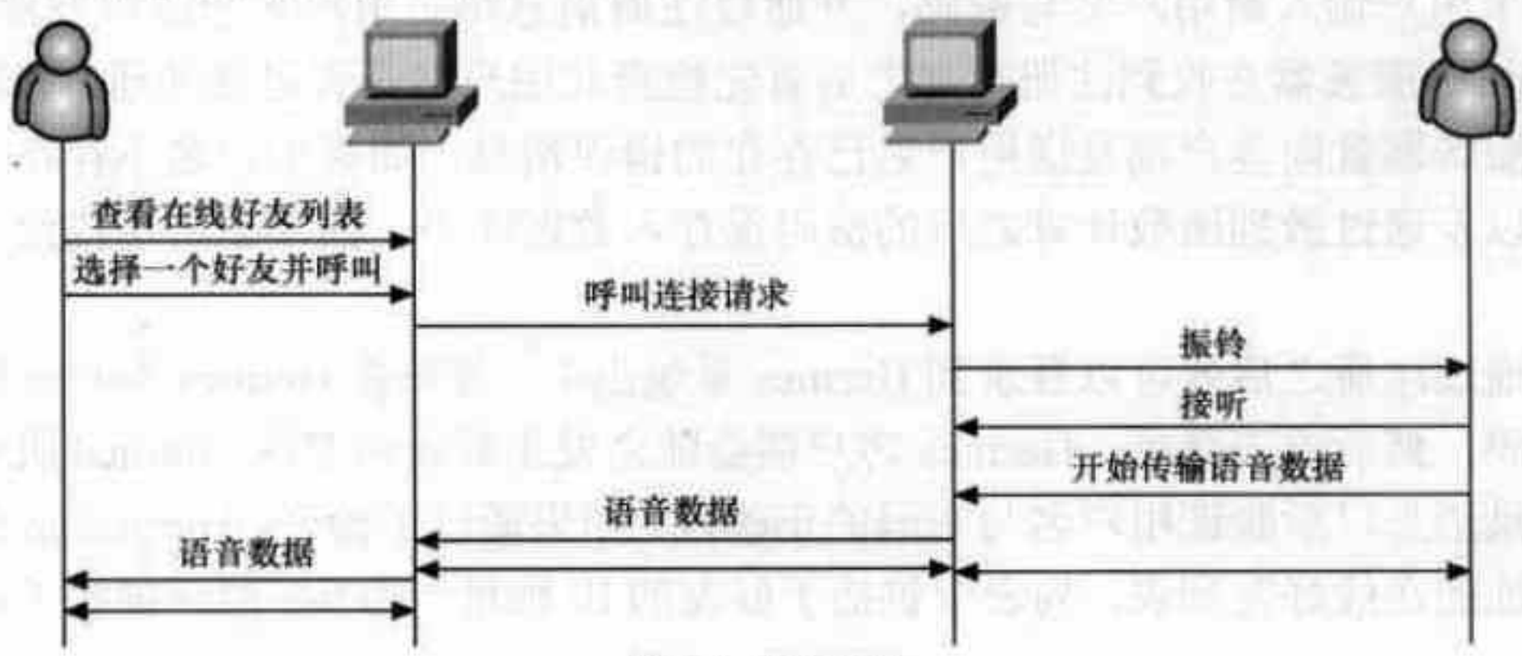


图 9-4　呼叫流程

www.ckook.com

当用户推出 Hermes 客户端时，Hermes 客户端会向服务器发送推出请求消息，Hermes 服务器在收到退出请求消息后会在在线用户列表中将其删去。图 9-5 所示的是用户退出系统的流程。

图 9-5 退出流程

9.2.2 语音提取与播放机制

Hermes 程序使用 Waveform Audio API 实现语音流的提取与播放。本节简要介绍一下 Waveform Audio API。首先介绍负责采集语音的一系列函数。

（1）waveInOpen

waveInOpen 函数打开指定的语音设备用于语音的采集。

```
MMRESULT waveInOpen(
  LPHWAVEIN          phwi,          //语音设备句柄
  UINT_PTR           uDeviceID,     //设置为 WAVE_MAPPER
  LPWAVEFORMATEX pwfx,              //语音数据的格式
  DWORD_PTR          dwCallback,    //回调函数的地址，最重要的参数，当收到语音数据时会将消息发送给此函数
  DWORD_PTR          dwCallbackInstance,
  DWORD              fdwOpen        //根据第 4 个参数的类型设置
);
```

（2）waveInPrepareHeader

waveInPrepareHeader 函数初始化用于采集语音的数据块。

```
MMRESULT waveInPrepareHeader(
  HWAVEIN hwi,          //语音设备句柄
  LPWAVEHDR pwh,        //指向语音数据缓存和信息的结构
  UINT cbwh             //WAVEHDR 的结构大小
);
```

（3）waveInAddBuffer

waveInAddBuffer 函数将把一个初始化之后的语音数据块提供给语音输入设备使用。当此数据块被填满后，waveform audio 系统会调用 waveInOpen 里设置的回调函数。

```
MMRESULT waveInAddBuffer(
  HWAVEIN hwi,          //语音设备句柄
  LPWAVEHDR pwh,        //指向语音数据缓存和信息的结构
  UINT cbwh
);
```

www.ckook.com

（4）waveInStart

waveInStart 函数使语音输入设备开始工作，采集到的语音数据将保存在被 waveIn-PrepareHeader 函数初始化并被 waveInAddBuffer 函数添加到数据块链表的数据块中。

```
MMRESULT waveInStart(
  HWAVEIN hwi   //语音设备句柄
);
```

（5）waveInReset

waveInReset 函数停止语音流的采集。

```
MMRESULT waveInReset(
  HWAVEIN hwi   //语音设备句柄
);
```

负责播放语音的函数与采集语音的函数的功能基本一致。

（6）waveOutOpen

waveOutOpen 函数打开语音输出设备。

```
MMRESULT waveOutOpen(
  LPHWAVEOUT        phwo,          //语音设备句柄
  UINT_PTR          uDeviceID,     //设置为 WAVE_MAPPER
  LPWAVEFORMATEX pwfx,             //语音数据的格式
  DWORD_PTR         dwCallback,    //回调函数的地址，最重要的参数，当语音数据播放完成时
//会将消息发送给此函数
  DWORD_PTR         dwCallbackInstance,
  DWORD             fdwOpen        //根据第 4 个参数的类型设置
);
```

（7）waveOutPrepareHeader

waveOutPrepareHeader 初始化用于播放的语音数据块。

```
MMRESULT waveOutPrepareHeader(
  HWAVEOUT hwo,          //语音设备句柄
  LPWAVEHDR pwh,         //指向语音数据缓存和信息的结构
  UINT cbwh
);
```

（8）waveOutUnprepareHeader

waveOutUnprepareHeader 函数释放语音数据块结构的资源。

```
MMRESULT waveOutUnprepareHeader(
  HWAVEOUT hwo,          //语音设备句柄
  LPWAVEHDR pwh,         //指向语音数据缓存和信息的结构
  UINT cbwh
);
```

（9）waveOutWrite

waveOutWrite 播放指定语音块的语音数据。

```
MMRESULT waveOutWrite(
  HWAVEOUT hwo,          //语音设备句柄
  LPWAVEHDR pwh,         //指向语音数据缓存和信息的结构
  UINT cbwh
);
```


www.ckook.com

Windows Audio 系统使用消息、回调的机制来使应用开发者可以处理语音数据，应用开发者可以根据自己的开发需要决定对哪些消息进行处理。本节余下部分介绍一下 Windows Audio 使用的消息。

（1）WIM_OPEN

当调用 waveInOpen 函数时，Windows Audio 系统会把 WIM_OPEN 消息发送给 waveInOpen 设置的回调函数。

（2）WIM_CLOSE

当调用 waveInClose 函数时，Windows Audio 系统会把 WIM_CLOSE 消息发送给 waveInOpen 设置的回调函数。

（3）WIM_DATA

当一个数据缓存装满之后，Windows Audio 系统会把 WIM_DATA 消息发送给 waveInOpen 设置的回调函数。

9.3 Hermes 系统设计

如图 9-6 所示，用户运行 Hermes 客户端后，屏幕上就会出现 Hermes 登录界面，注册用户只需输入自己的用户名和密码就可以登录，新用户可以单击“新账号”按钮登录到系统中。

成功登录 Hermes 系统后，用户可以在屏幕右侧的好友列表选择自己打算呼叫的好友。如果有好友呼叫自己，Hermes 系统会有振铃提示。图 9-7 所示是 Hermes 的运行界面。

图 9-6 Hermes 登录界面

图 9-7 Hermes 运行界面

如图 9-8 所示，Hermes 客户端主要由 4 部分组成：主控模块、用户界面、通信模块和语音处理模块。其中主控模块负责整个客户端的初始化工作，它还要管理程序的状态机等。用户界面用于接收来自用户的输入命令。通信模块实现对传输层的封装，使得上层能够方便地使用 TCP 或者 UDP 协议传输数据。语音处理模块实现语音的采集与播放功能。

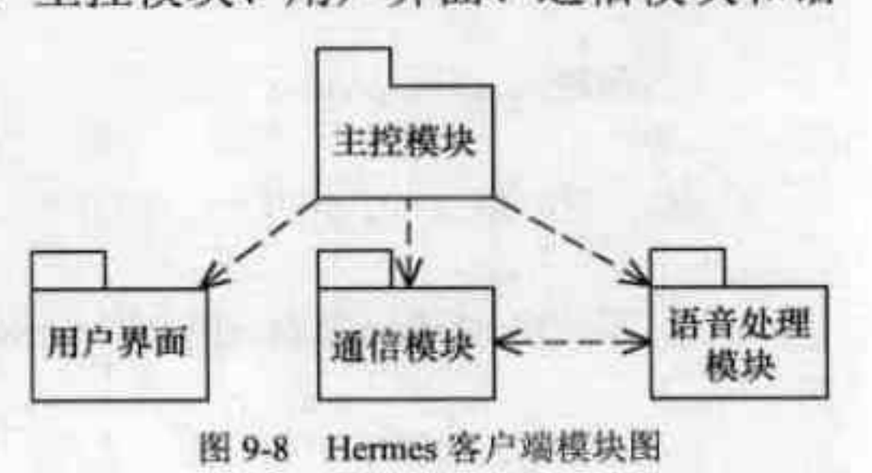


图 9-8 Hermes 客户端模块图

www.ckook.com

9.4 Hermes 程序代码分析

本节将就 Hermes 的源代码进行详细分析，其源代码可以到人民邮电出版社网站 www.ptpress.com.cn 上下载。

9.4.1 语音处理模块代码分析

语音处理模块负责 Hermes 系统的语音提取与播放等和语音相关的内容。设计上主要有两个类，一个是 CwaveBuffer 类，另一个是 CwaveManage 类。CWaveBuffer 类实现了对一块语音数据的封装。用户可以通过修改 CWaveBuffer 对象中的参数来修改语音编码的格式、速率等。每一个 CWaveBuffer 对象的生存期可以很长，一个 CWaveBuffer 对象可以首先用来采集语音，然后 CWaveManage 会对采集来的语音进行发送，然后这个 CWaveBuffer 对象不需要被释放就可以被重新使用采集语音。这样就降低了大量的重新创建 CWaveBuffer 对象所需花费的时间。图 9-9 和图 9-10 分别描述了 Hermes 语音处理模块的原理图和类图。

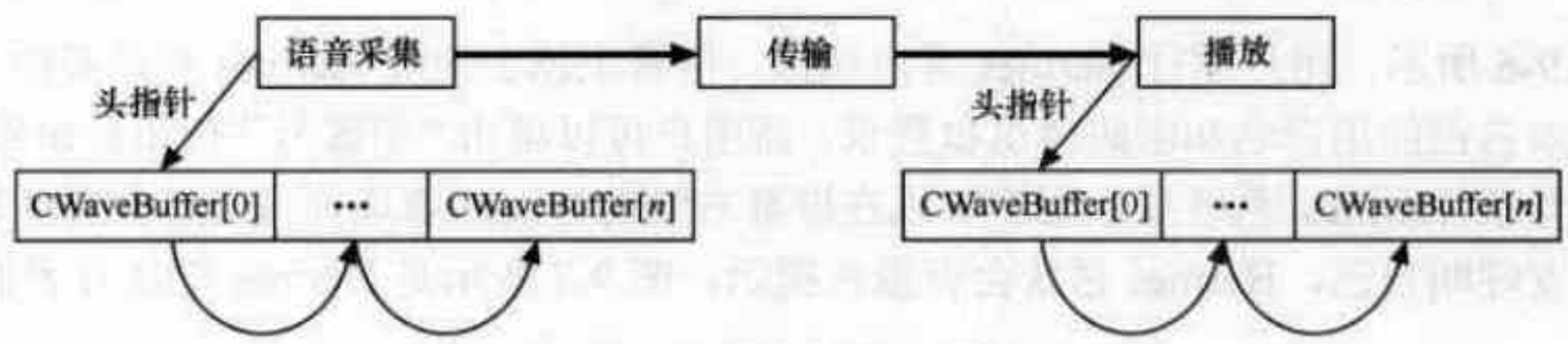


图 9-9 语音处理模块原理图

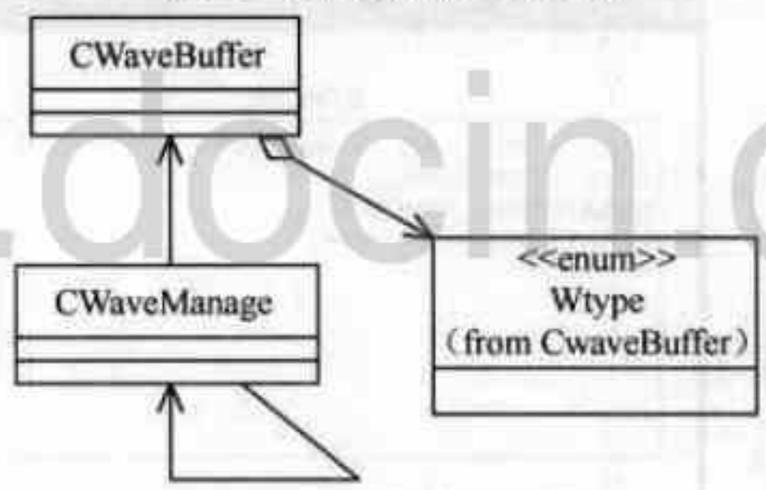


图 9-10 语音处理模块类图

文件 WaveBuffer.h 如下。

```
//-----------------------------------------------------------------------
//
//   文件名: WaveBuffer.h
//
//创建人: ziming.zhao
//
//   日   期: 2007-5-30
//
//描   述: Define CWaveBuffer, capsulate WAVEHDR and buf.
//
//版   本: 1.0
```


www.docin.com
www.ckook.com

```
//
//-----------------------------------------------------------------------------

#if !defined(WAVEBUFFER_H)
#define WAVEBUFFER_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Mmsystem.h"

const int BUFNUM = 6;

#define WM_DATA_RECEIVED (WM_USER+5117)

class CWaveBuffer
{
public:
    enum Wtype {WAVEIN, WAVEOUT};
    enum {BUFSIZE = 2205}; //11025*0.2m

    CWaveBuffer(HWAVEIN hwi); //3 个构造函数用于 3 种不同情况下的构造
    CWaveBuffer(HWAVEOUT hwo);
    CWaveBuffer(HWAVEOUT hwo, LPSTR p_buf, DWORD dwNum);

    virtual ~CWaveBuffer();

    MMRESULT Prepare();

    MMRESULT AddToWaveInBuffer();

    void ResetWaveInBuffer(); //重置 CWaveBuffer
    void ResetWaveOutBuffer(); //重置 CWaveBuffer
    void UseWaveOutBuffer(LPSTR p_buf, DWORD dwNum);

    MMRESULT Play();
private:
    HANDLE m_hwio;
    Wtype m_iotype;

    WAVEHDR m_waveh;
    LPSTR m_buffer;
```

www.docin.com

```
};

//////////////////////////////////////////////////////////////////////
// 11 025kHz
// 8bit/s = 1Byte/s
// So, the buffer you need is T×11 025 Byte
//////////////////////////////////////////////////////////////////////
class CWaveFormat
{
public:
    WAVEFORMATEX m_wfx;

public:
    CWaveFormat();
    virtual ~CWaveFormat();
};

#endif // !defined(WAVEBUFFER_H__)
```

文件 WaveBuffer.cpp 如下。

```
//-----------------------------------------------------------------------
//
//   文件名: WaveBuffer.cpp
//
//      创建人: ziming.zhao
//
//   日    期: 2007-5-30
//
//      描    述: Define CWaveBuffer, capsulate WAVEHDR and buf.
//
//      版    本: 1.0
//
//-----------------------------------------------------------------------

#include "../Common/StdAfx.h"
#include "WaveBuffer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
```

www.ccgn.com

www.ckook.com

```
//////////////////////////////////////////////////////////////////////

CWaveBuffer::CWaveBuffer(HWAVEIN hwi)
{
    m_buffer = new CHAR[BUFSIZE];

    m_waveh.lpData = m_buffer;
    m_waveh.dwBufferLength = BUFSIZE;
    m_waveh.dwBytesRecorded = 0;
    m_waveh.dwUser = 0;
    m_waveh.dwFlags = 0;
    m_waveh.dwLoops = 1;
    m_waveh.lpNext = NULL;
    m_waveh.reserved = 0;

    m_iotype = WAVEIN;
    m_hwio = (HWAVEIN)hwi;
}

CWaveBuffer::CWaveBuffer(HWAVEOUT hwo)
{
    m_buffer = new CHAR[BUFSIZE];

    m_waveh.lpData = m_buffer;
    m_waveh.dwBufferLength = BUFSIZE;
    m_waveh.dwBytesRecorded = 0;
    m_waveh.dwUser = 0;
    m_waveh.dwFlags = 0;
    m_waveh.dwLoops = 0;
    m_waveh.lpNext = NULL;
    m_waveh.reserved = 0;

    m_iotype = WAVEOUT;
    m_hwio = (HWAVEIN)hwo;
}

CWaveBuffer::CWaveBuffer(HWAVEOUT hwo, LPSTR p_buf, DWORD dwNum)
{
    m_buffer = new CHAR[dwNum];
    CopyMemory(m_buffer, p_buf, dwNum);

    m_waveh.lpData = m_buffer;
    m_waveh.dwBufferLength = dwNum;
```

www.ccin.com

www.ckook.com

```
    m_waveh.dwBytesRecorded = 0;
    m_waveh.dwUser = 0;
    m_waveh.dwFlags = 0;
    m_waveh.dwLoops = 0;
    m_waveh.lpNext = NULL;
    m_waveh.reserved = 0;

    m_iotype = WAVEOUT;
    m_hwio = (HWAVEIN)hwo;
}

CWaveBuffer::~CWaveBuffer()
{
    if (m_iotype == WAVEIN)
    {
        waveInUnprepareHeader((HWAVEIN)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }
    else
    {
        waveOutUnprepareHeader((HWAVEOUT)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }

    if (m_buffer == NULL)
        return;

    delete []m_buffer;
    m_buffer = NULL;
}
//调用 waveInPrepareHeader 或者 waveOutPrepareHeader 初始化数据块
MMRESULT CWaveBuffer::Prepare()
{
    if (m_iotype == WAVEIN)
    {
        return waveInPrepareHeader((HWAVEIN)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }
    else
    {
        return waveOutPrepareHeader((HWAVEOUT)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }
}

MMRESULT CWaveBuffer::AddToWaveInBuffer()
{
```

www.docin.com

www.ckook.com

```
        if (m_iotype == WAVEOUT)
            return 0;

        return waveInAddBuffer((HWAVEIN)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }

    void CWaveBuffer::ResetWaveInBuffer()
    {
        if (m_iotype == WAVEOUT)
            return;

        m_waveh.lpData = m_buffer;
        m_waveh.dwBufferLength = BUFSIZE;
        m_waveh.dwBytesRecorded = 0;
        m_waveh.dwUser = 0;
        m_waveh.dwFlags = 0;
        m_waveh.dwLoops = 1;
        m_waveh.lpNext = NULL;
        m_waveh.reserved = 0;

        m_iotype = WAVEIN;
    }

    void CWaveBuffer::ResetWaveOutBuffer()
    {
        if (m_iotype == WAVEIN)
            return;

        m_waveh.lpData = m_buffer;
        m_waveh.dwBufferLength = BUFSIZE;
        m_waveh.dwBytesRecorded = 0;
        m_waveh.dwUser = 0;
        m_waveh.dwFlags = 0;
        m_waveh.dwLoops = 0;
        m_waveh.lpNext = NULL;
        m_waveh.reserved = 0;

        m_iotype = WAVEOUT;
    }

    void CWaveBuffer::UseWaveOutBuffer(LPSTR p_buf, DWORD dwNum)
    {
        if (m_iotype == WAVEIN)
```

www.docin.com

www.ckook.com

```
            return;

        CopyMemory(m_buffer, p_buf, dwNum);
        m_waveh.dwBufferLength = dwNum;
    }

    MMRESULT CWaveBuffer::Play()
    {
        if (m_iotype == WAVEIN)
            return 0;

        return waveOutWrite((HWAVEOUT)m_hwio, &m_waveh, sizeof(WAVEHDR));
    }

    ////////////////////////////////////////////////////////////////////
    //CWaveFormat
    ////////////////////////////////////////////////////////////////////
    CWaveFormat::CWaveFormat()
    {
        m_wfx.wFormatTag = WAVE_FORMAT_PCM;
        m_wfx.nChannels = 1;
        m_wfx.nSamplesPerSec = 11025;
        m_wfx.nAvgBytesPerSec = 11025;
        m_wfx.nBlockAlign = 1;
        m_wfx.wBitsPerSample = 8;
        m_wfx.cbSize = 0;
    }

    CWaveFormat::~CWaveFormat()
    {
    }
```

文件 WaveManage.h 如下。

```
    //-----------------------------------------------------------------------
    //
    //  文件名: WaveManage.h
    //
    //      创建人: Zhao Ziming
    //
    //  日    期: 2007-7-4
    //
    //      描    述: 语音库，实现语音的提取、播放、异或
    //
    //      版    本: 1.0
```


www.ckook.com

```
//
//-----------------------------------------------------------------------------

#if !defined(WAVEMANAGE_H__)
#define WAVEMANAGE_H__

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "WaveBuffer.h"

class CWaveManage
{
public:
    static CWaveManage *GetInstance();
    static void ExitInstance();

    void Start();          //开始语音数据的采集与播放语音
    void HandleWaveoutData(LPSTR p_buf, DWORD dwNum, BOOL xor); //处理要播放的语音数据，由外部调用

    enum{OFFLINE = 0, ONLINE = 1};

private:
    static void CALLBACK waveInProc(HWAVEIN hwi, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2); //WaveInOpen 指定的回调函数
    static void CALLBACK waveOutProc(HWAVEOUT hwo, UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2); //WaveOutOpen 指定的回调函数

    static DWORD WINAPI InitThreadFunc(LPVOID pParam);

    void waveInInit();
    void waveOutInit();

    CWaveBuffer *m_pWaveInBuf[BUFNUM];          //输入语音缓存
    CWaveBuffer *m_pWaveOutBuf[BUFNUM];         //输出语音缓存

    HWAVEIN m_hWavein;                          //语音输入句柄
    HWAVEOUT m_hWaveout;                        //语音输出句柄

    HANDLE m_hWaveinBufReady;                   //4 个用于同步的 event 句柄
    HANDLE m_hWaveoutBufReady;
```

www.ckook.com

```
        HANDLE m_hWaveinReady;
        HANDLE m_hWaveoutReady;

        unsigned m_WaveInIndex;                                    //4 个上下行语音
        unsigned m_WaveOutIndex;

    private:
        static CWaveManage *m_spInstance;
        static UINT m_State;

        CWaveManage(); //私有构造函数
        virtual ~CWaveManage(); //私有析构函数
    };

    #endif // !defined(WAVEMANAGE_H__)
```

文件 WaveManage.cpp 如下。

```
    //-----------------------------------------------------------------------------
    //
    //   文件名: WaveManage.cpp
    //
    //       创建人: Zhao Ziming
    //
    //   日    期: 2007-7-4
    //
    //       描    述: 语音库，实现语音的提取、播放、异或
    //
    //       版    本: 1.0
    //
    //-----------------------------------------------------------------------------
    #include "../Common/StdAfx.h"
    #include "WaveManage.h"

    CWaveManage *CWaveManage::m_spInstance = NULL;
    UINT CWaveManage::m_State = OFFLINE;

    ////////////////////////////////////////////////////////////////////////////
    // PUBLIC METHOD
    ////////////////////////////////////////////////////////////////////////////
    // Using "singleton pattern"
    ////////////////////////////////////////////////////////////////////////////
    //
    //       CWaveManage* CWaveManage::GetInstance()
    //       void CWaveManage::ExitInstance()
```


www.ckook.com

```
//
//  函数功能：在每一次通话结束后必须调用 CWaveManage::ExitInstance()释放语音库，下一次通话时再调用 CWaveManage::GetInstance()
////////////////////////////////////////////////////////////////////
CWaveManage* CWaveManage::GetInstance()
{
    if (m_spInstance == NULL)
    {
        m_spInstance = new CWaveManage();
    }

    m_State = ONLINE;
    return m_spInstance;
}

void CWaveManage::ExitInstance()
{
    int ret;
    ret = waveInStop(CWaveManage::GetInstance()->m_hWavein);

    if (m_spInstance != NULL)
    {
        delete m_spInstance;
        m_spInstance = NULL;
    }
    m_State = OFFLINE;
}

////////////////////////////////////////////////////////////////////
//
//    void CWaveManage::Start()
//
//  函数功能：开始语音数据的采集与播放语音
//
//  函数参数：无
//
//  函数返回：void
////////////////////////////////////////////////////////////////////
void CWaveManage::Start()
{
    CreateThread(NULL, 0, InitThreadFunc, (LPVOID)this, 0, NULL);
}
```

www.docin.com

```
DWORD CWaveManage::InitThreadFunc(LPVOID pParam)
{
    int ret;
    CWaveFormat wf;

    ret = waveInOpen(&(CWaveManage::GetInstance()->m_hWavein), WAVE_MAPPER, &wf.m_wfx,
reinterpret_cast<DWORD>(waveInProc), NULL, CALLBACK_FUNCTION);
    SetEvent(CWaveManage::GetInstance()->m_hWaveinReady);
    CWaveManage::GetInstance()->waveInInit();
    if (ret != 0)
    {
        AfxMessageBox("waveInOpen Error");
    }

    ret = waveOutOpen(&(CWaveManage::GetInstance()->m_hWaveout), WAVE_MAPPER,
&wf.m_wfx, reinterpret_cast<DWORD>(waveOutProc), NULL, CALLBACK_FUNCTION);
    SetEvent(CWaveManage::GetInstance()->m_hWaveoutReady);
    CWaveManage::GetInstance()->waveOutInit();
    if (ret != 0)
    {
        AfxMessageBox("waveOutOpen Error");
    }

    return 0;
}

////////////////////////////////////////////////////////////////////////
//
//    void CWaveManage::HandleWaveoutData(LPSTR p_buf, DWORD dwNum, BOOL xor)
//
//  函数功能：处理用于播放的语音数据
//
//  函数参数
//  LPSTR p_buf [IN] 用于播放的语音数据的地址，执行此函数后此地址上的数据就可释放
//  DWORD dwNum [IN] 用于播放的语音数据的大小
//  BOOL xor     [IN] 预留
//
//  函数返回：void
////////////////////////////////////////////////////////////////////////
void CWaveManage::HandleWaveoutData(LPSTR p_buf, DWORD dwNum, BOOL xor)
{
    int ret;
```


www.ckook.com

```
    WaitForSingleObject(m_hWaveoutBufReady, INFINITE);

    if (m_State == OFFLINE)
        return;

    if (m_spInstance == NULL)
        return;

    m_pWaveOutBuf[m_WaveOutIndex]->ResetWaveOutBuffer();
    m_pWaveOutBuf[m_WaveOutIndex]->UseWaveOutBuffer(p_buf, dwNum);

    ret = m_pWaveOutBuf[m_WaveOutIndex]->Prepare();
    if (ret != 0)
    {
        AfxMessageBox("Prepare Error in HandleWaveoutData");
    }

    m_WaveOutIndex = (m_WaveOutIndex + 1) % BUFNUM;

    ret = waveOutReset(m_hWaveout);
    if (ret != 0)
    {
        AfxMessageBox("waveOutReset Error in OnWM_DATA_RECEIVED.");
        return;
    }

    ret = m_pWaveOutBuf[(m_WaveOutIndex + BUFNUM - 1) % BUFNUM]->Play();
    if (ret != 0)
    {
        AfxMessageBox("Play Error");
        return;
    }

    return;
}

/////////////////////////////////////////////////////////////////////////////
// Private Construction/Destruction
/////////////////////////////////////////////////////////////////////////////
CWaveManage::CWaveManage()
{
    for (int i = 0; i < BUFNUM; i++)
    {
```

```
        m_pWaveInBuf[i] = NULL;
        m_pWaveOutBuf[i] = NULL;
    }

    m_hWavein = 0x0;
    m_hWaveout = 0x0;

    m_hWaveinReady = CreateEvent(NULL, FALSE, FALSE, NULL);
    m_hWaveoutReady = CreateEvent(NULL, FALSE, FALSE, NULL);
    m_hWaveinBufReady = CreateEvent(NULL, FALSE, FALSE, NULL);
    m_hWaveoutBufReady = CreateEvent(NULL, TRUE, FALSE, NULL);
}

CWaveManage::~CWaveManage()
{
    int ret;

    ret = waveInClose(m_hWavein);
    ret = waveOutClose(m_hWaveout);

    for(int i = 0; i < BUFNUM; i++)
    {
        if (m_pWaveInBuf[i] != NULL)
        {
            delete m_pWaveInBuf[i];
            m_pWaveInBuf[i] = NULL;
        }

        if (m_pWaveOutBuf[i] != NULL)
        {
            delete m_pWaveOutBuf[i];
            m_pWaveOutBuf[i] = NULL;
        }
    }

    CloseHandle(m_hWaveinReady);
    CloseHandle(m_hWaveoutReady);
    CloseHandle(m_hWaveinBufReady);
    CloseHandle(m_hWaveoutBufReady);
}

/////////////////////////////////////////////////////////////////////////////
//
```

www.ccn.com

www.ckook.com

```
//  void CALLBACK CWaveManage::waveInProc(HWAVEIN hwi, UINT uMsg, DWORD dwInstance,
DWORD dwParam1, DWORD dwParam2)
//  void CALLBACK CWaveManage::waveOutProc(HWAVEOUT hwi, UINT uMsg, DWORD
dwInstance, DWORD dwParam1, DWORD dwParam2)
//
//  函数功能：WaveInOpen 指定的回调函数，waveOutProc 指定的回调函数
//
//  函数返回：void
////////////////////////////////////////////////////////////////////////
void CALLBACK CWaveManage::waveInProc(HWAVEIN hwi, UINT uMsg, DWORD dwInstance,
DWORD dwParam1, DWORD dwParam2)
{
    int ret;

    switch(uMsg)
    {
    case WIM_DATA:
        {
            if (m_State == OFFLINE)
                return;

            if (m_spInstance == NULL)
                return;

            ret = ::SendVoiceMsg(((WAVEHDR*)dwParam1)->lpData, CWaveBuffer::BUFSIZE);
            if (ret != 0)
            {
                return;
            }

            m_spInstance->m_pWaveInBuf[CWaveManage::GetInstance()->m_WaveInIndex]->
ResetWaveInBuffer();

            ret = m_spInstance->m_pWaveInBuf[m_spInstance->m_WaveInIndex]->Prepare();
            if (ret != 0)
            {
                AfxMessageBox("Prepare Error in OnMM_WIM_DATA");
                return;
            }

              ret = m_spInstance->m_pWaveInBuf[m_spInstance-> m_WaveInIndex]-> AddTo
WaveInBuffer();
            if (ret != 0)
```

www.docin.com

```
            {
                AfxMessageBox("add buffer Error in OnMM_WIM_DATA");
                return;
            }

            m_spInstance->m_WaveInIndex = (m_spInstance->m_WaveInIndex+1) % BUFNUM;
            break;
        }

        default:
            return;
        }
    }

    void CALLBACK CWaveManage::waveOutProc(HWAVEOUT hwi, UINT uMsg, DWORD dwInstance,
DWORD dwParam1, DWORD dwParam2)
    {
        return;
    }

    void CWaveManage::waveInInit()
    {
        int ret;
        m_WaveInIndex = 0;

        for (int i = 0; i < BUFNUM; i++)
        {
            m_pWaveInBuf[i] = new CWaveBuffer(m_hWavein);

            ret = m_pWaveInBuf[i]->Prepare();
            if (ret != MMSYSERR_NOERROR)
                AfxMessageBox("WAVEIN Prepare Error.");

            ret = m_pWaveInBuf[i]->AddToWaveInBuffer();
            if (ret != MMSYSERR_NOERROR)
                AfxMessageBox("WAVEIN Add Error.");
        }

        SetEvent(m_hWaveinBufReady);

        ret = waveInStart (m_hWavein);
        if (ret != MMSYSERR_NOERROR)
            AfxMessageBox("waveInStart Error.");
```


www.ckook.com

```
}

void CWaveManage::waveOutInit()
{
    m_WaveOutIndex = 0;

    for (int i = 0; i < BUFNUM; i++)
    {
        m_pWaveOutBuf[i] = new CWaveBuffer(m_hWaveout);
    }

    SetEvent(m_hWaveoutBufReady);
}
```

9.4.2 通信模块代码分析

网际协议（IP，Internet Protocol）是一种用于因特网的网络协议，已经广为人知。它可广泛用于大多数计算机操作系统上，也可用于大多数局域网（LAN，比如办公室小型网络）和广域网（WAN，比如说因特网）。从它的设计看来，IP 是一个无连接的协议，不能保证数据投递万无一失。两个比它高级的协议（TCP 和 UDP）用于依赖 IP 协议的数据通信。面向连接的通信是通过传输控制协议（TCP，Transmission Control Protocol）来完成的。

TCP 提供两台计算机之间的可靠无错的数据传输。应用程序利用 TCP 进行通信时，源和目标之间会建立一个虚拟连接。这个连接一旦建立，两台计算机之间就可以把数据当作一个双向字节流进行交换。Hermes 系统使用 Windows 下的 Winsock 进行开发。每个 Winsock 应用都必须加载 Winsock DLL 的相应版本。如果调用 Winsock 之前，没有加载 Winsock 库，这个函数就会返回一个 SOCKET_ERROR，错误信息是 WSANOTINITIALISED。加载 Winsock 库是通过调用 WSAStartup 函数实现的。对 TCP 连接中，往往把被动接受连接的一方叫作“服务器”。“服务器”其实是一个进程，它需要等待任意数量的客户端连接，以便为它们的请求提供服务。对服务器监听的连接来说，它必须在一个已知的名字上。在 TCP/IP 中，这个名字就是本地接口的 IP 地址，加上一个端口编号。每种协议都有一套不同的定址方案，所以有一种不同的命名方法。在 Winsock 中，第一步是将指定协议的套接字绑定到它已知的名字上。这个过程是通过 API 调用 bind 函数来完成的。下一步是将套接字置为监听模式，这是用 API 函数 listen 来完成的。最后，若一个客户端试图建立连接，服务器必须通过 accept 或 WSAAccept 调用来接受连接。在图 9-11 中，我们展示了为建立一个通信信道，服务器和客户端必须执行的基本调用。

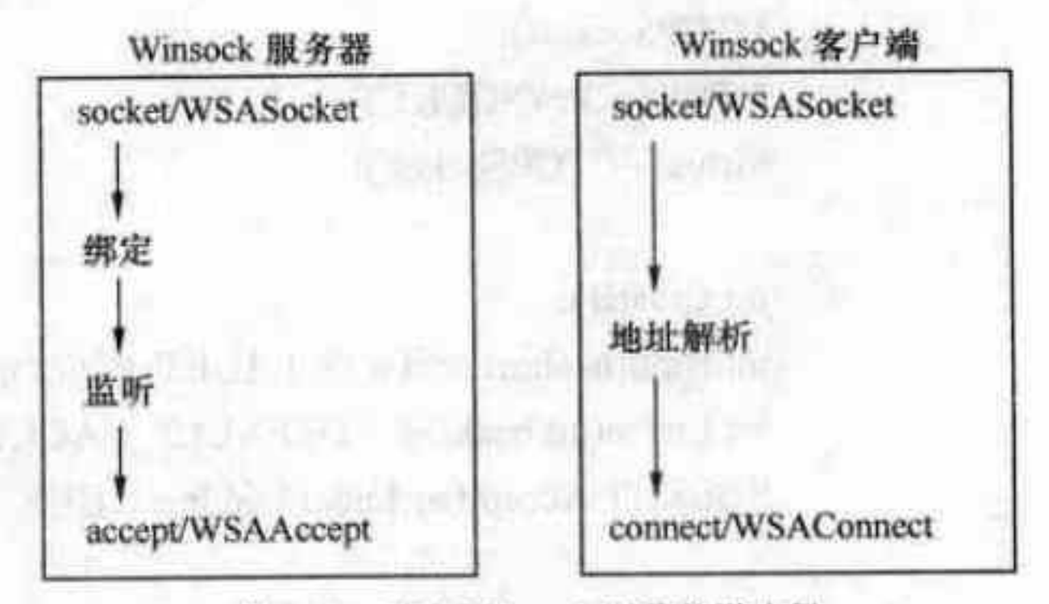


图 9-11 建立 Winsock 通信信道流程

www.ckook.com

Hermes 语音数据的传输使用 TCP 协议，TCPSocket 类封装了 Winsock 系统下 TCP 协议的一些常用函数，可以实现简单的 TCP 通信。

文件 TCPSocket.h 如下。

```
//-----------------------------------------------------------------------------
//
//   文件名: TCPSocket.h
//
//      创建人: ziming.zhao
//
//   日    期: 2007-5-30
//
//      描    述: Define CTCPSocket.
//
//      版    本: 1.0
//
//-----------------------------------------------------------------------------

#if !defined(_TCPSOCKET_H_)
#define _TCPSOCKET_H_

#if _MSC_VER > 1000
#pragma once
#endif

#include "winsock2.h"

class CTCPSocket
{
public:
    static int Initialize();
    static int CleanUp();

    CTCPSocket();
    CTCPSocket(SOCKET s);
    virtual ~CTCPSocket();

    int Create();
    int Bind(u_short port = DEFAULT_PORT);
    int Listen(int backlog = DEFAULT_BACKLOG);
    SOCKET Accept(sockaddr* addr = NULL, int* addrlen = NULL);

    int Connect(DWORD ip, u_short port = DEFAULT_PORT);
    int Send(const char *pBuf, int len);
```

www.docin.com

www.ckook.com

```
        int Recv(char *pbuf, int len);

        int Close();

        SOCKET m_socket;
    private:

        sockaddr_in m_local;
        sockaddr_in m_foreign;

        enum {DEFAULT_PORT = 56177, DEFAULT_BACKLOG = 1};
    };

    #endif // _TCPSOCKET_H_
```

文件 TCPSocket.cpp 如下。

```
    //-----------------------------------------------------------------------
    //
    //   文件名: TCPSocket.cpp
    //
    //      创建人: ziming.zhao
    //
    //   日    期: 2007-5-30
    //
    //      描    述: Define CTCPSocket.
    //
    //      版    本: 1.0
    //
    //-----------------------------------------------------------------------

    #include "stdafx.h"
    #include "TCPSocket.h"

    #ifdef _DEBUG
    #undef THIS_FILE
    static char THIS_FILE[]=__FILE__;
    #define new DEBUG_NEW
    #endif

    ////////////////////////////////////////////////////////////////////////
    //
    //      CTCPSocket::CTCPSocket()
    //      CTCPSocket::~CTCPSocket()
    //
```

www.docin.com

```
//  函数功能：构造析构函数
//
////////////////////////////////////////////////////////////////
CTCPSocket::CTCPSocket()
{
    m_socket = INVALID_SOCKET;
}

CTCPSocket::CTCPSocket(SOCKET& s)
{
    m_socket = s;
}

CTCPSocket::~CTCPSocket()
{
}

////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Initialize()
//
//  函数功能：初始化 Winsock
//
//  函数返回：int
////////////////////////////////////////////////////////////////
int CTCPSocket::Initialize()
{
    WSADATA wsd;

    return WSAStartup(MAKEWORD(2,2), &wsd);
}

////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Create()
//
//  函数功能：创建 Socket
//
//  函数返回：int
////////////////////////////////////////////////////////////////
int CTCPSocket::Create()
{
    m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
```

www.ccin.com

www.ckook.com

```
    return m_socket;
}

////////////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Bind(u_short port /* = DEFAULT_PORT */)
//
//  函数功能：绑定端口
//
//  函数参数：
//                  u_short port  端口号
//
//  函数返回：int
////////////////////////////////////////////////////////////////////////
int CTCPSocket::Bind(u_short port /* = DEFAULT_PORT */)
{
    m_local.sin_addr.s_addr = htonl(INADDR_ANY);
    m_local.sin_family = AF_INET;
    m_local.sin_port = port;

    return bind(m_socket, (struct sockaddr*)&m_local, sizeof(m_local));
}

////////////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Listen(int backlog)
//
//  函数功能：监听端口
//
//  函数参数：
//                  int backlog 队列长度
//
//  函数返回：int
////////////////////////////////////////////////////////////////////////
int CTCPSocket::Listen(int backlog)
{
    return listen(m_socket, backlog);
}

////////////////////////////////////////////////////////////////////////
//
//      SOCKET CTCPSocket::Accept(sockaddr* addr /* = NULL */, int* addrlen /* = NULL */)
```

www.ckook.com

```
//
//  函数功能：接受连入请求
//
//  函数参数：
//      sockaddr* addr      输出地址
//      int* addrlen        输出地址的长度
//
//  函数返回：Socket
////////////////////////////////////////////////////////////////////
SOCKET CTCPSocket::Accept(sockaddr* addr /* = NULL */, int* addrlen /* = NULL */)
{
    return accept(m_socket, addr, addrlen);
}

////////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Connect(DWORD ip, u_short port /* = DEFAULT_PORT */)
//
//  函数功能：连接
//
//  函数参数：
//      DWORD ip        连接的 IP 地址
//      u_short port    要连接的端口号
//
//  函数返回：int
////////////////////////////////////////////////////////////////////
int CTCPSocket::Connect(DWORD ip, u_short port /* = DEFAULT_PORT */)
{
    ip = htonl(ip);

    m_foreign.sin_addr.s_addr = ip;
    m_foreign.sin_family = AF_INET;
    m_foreign.sin_port = port;

    return connect(m_socket, (struct sockaddr*)&m_foreign, sizeof(m_foreign));
}

////////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::Send(const char *pBuf, int len)
//
//  函数功能：发送数据
//
```


www.ckook.com

```
//  函数参数:
//  const char *pBuf   数据的地址
//  int len            数据的长度
//
//  函数返回: int
////////////////////////////////////////////////////////////////
int CTCPSocket::Send(const char *pBuf, int len)
{
    return send(m_socket, pBuf, len, 0);
}

////////////////////////////////////////////////////////////////
//
//     int CTCPSocket::Recv(char *pbuf, int len)
//
//  函数功能: 接收数据
//
//  函数参数:
//  char *pbuf     接收数据的地址
//  int len        打算接收的长度
//
//  函数返回: int 实际接收的长度
////////////////////////////////////////////////////////////////
int CTCPSocket::Recv(char *pbuf, int len)
{
    return recv(m_socket, pbuf, len, 0);
}

////////////////////////////////////////////////////////////////
//
//     int      CTCPSocket::Close()
//
//  函数功能: 关闭连接
//
//  函数返回: int
////////////////////////////////////////////////////////////////
int      CTCPSocket::Close()
{
    int ret;

    ret = shutdown(m_socket,2);
    return closesocket(m_socket);
}
```

www.ckook.com

```
//////////////////////////////////////////////////////////////////
//
//      int CTCPSocket::CleanUp()
//
//   函数功能：释放资源
//
//   函数返回：int
//////////////////////////////////////////////////////////////////
int CTCPSocket::CleanUp()
{
      return WSACleanup();
}
```

9.5 Hermes 系统分析

Hermes 实现了一个简单的 P2P 语音通信系统，设计上比较简单清楚，比较适合读者学习。Hermes 系统还有许多可以改进的地方，比如提供加密的语音通信，在两个节点不能直接通信时，语音的传输不采用直连的方式而采用其他节点中转。

9.6 本章总结

本章分析了 P2P 即时通信系统的背景，并介绍了作者设计的一个简单的 P2P 即时通信系统。在介绍 Windows 多媒体语音库的 API 之后，分析了此即时通信系统的设计和编码，让读者掌握即时通信系统设计的普遍方法。

www.docin.com

9.7 练习题

（1）简述 P2P 即时通信系统的背景。
（2）学习 Windows 系统提供的多媒体语音 API。
（3）设计一个可以连续播放和录音的程序。

参 考 文 献

[1] Petzold C. Programming Windows. Microsoft Press, 1998

[2] Richter J，著．王建华，等，译．Windows 核心编程．北京：机械工业出版社，2005

[3] Jones A, Ohlund J. Network Programming for Microsoft Windows. Microsoft Corporation, 2002

[4] Stevens R，著．范建华，等，译．TCP/IP 协议详解卷一：协议．北京：机械工业出版社，2000

www.ckook.com

CHAPTER 10

第10章 P2P流媒体系统开发实例——PeerCast的设计与实现

在学习了本章后，希望读者能够掌握以下几点：

（1）流媒体系统的概念；

（2）P2P网络中的媒体传输技术。

10.1 P2P流媒体系统概念

10.1.1 流媒体内容发布网络技术

1．IP单播技术

单播技术是一种单点到单点的数据传输模式，这种模式也是最常见的Internet通信方式。HTTP和FTP等应用都采用这种单播模式。这种模式的缺点是如果多个用户同时请求同一份数据，服务器必须通过网络给每个用户发送多份相同的数据，由于多媒体的数据量大，随着客户端数目的增加，很容易造成服务器端的网络拥塞。

2．IP广播技术

广播技术是单点到所有主机的数据传输模式，服务器发送一份数据，发送到连接在网络上的所有主机。采用这种模式，服务器只需要发送一次数据，效率很高，但是由于发送模式的盲目性，数据会扩散到所有的网段，而不关心网段中的主机是否需要接收。如果多媒体通信采用广播发送方式，则大量的数据将造成“广播风暴”，使网络通信处于瘫痪状态。

3．IP多播技术

多播技术可以将一份数据同时发送给一组特定的用户，避免了数据的冗余，又不会盲目的造成网络带宽的浪费。目前很多的流媒体应用都采用这种传输模式。多播的定义是：将IP数据包发送到共享相同IP地址的一个主机组中，该主机组的成员数目可以为零个或多个。多播数据包的传输可靠性和IP单播相同。主机组的成员是动态的，任何主机可以在任何时刻加入或退出某个主机组，主机组对其成员没有位置和数量的限制，且一台主

www.ckook.com

机可以同时是多个主机组的成员。

4. P2P 流媒体分发技术

目前 P2P 流媒体模型主要分为以下两大类。

（1）基于树状拓扑的协议及扩展（Tree-based protocol and extensions）。此模型将节点组织成一棵多播树，树的父节点负责为子节点传送数据。此模型主要考虑多播树的构造和平衡。

（2）基于 Gossip 的协议（Gossip-based protocol）。基于 Gossip 的算法是目前流行的在 P2P 系统中分发消息的算法。一个典型的 Gossip 算法中，节点随机的给系统中的部分节点发送消息，每个接收到消息的节点继续向其他节点发送消息，重复这个过程，直到消息被发送给系统中的所有节点。基于 Gossip 协议的 P2P 流媒体系统并不显式的构造节点之间的拓扑结构，而是通过 Gossip 协议，每个节点维护系统中部分其他节点的视图，每个节点动态的和其他节点交换缓存信息，并根据缓存信息交换节点之间的媒体数据。在这种系统中，通常需要比较大的缓存，系统的启动延迟相对比较大。

10.1.2 P2P 流媒体系统概述

随着信息网络技术的发展，对流媒体的需求也逐渐增大。但是传统的流媒体主要是采用客户端—服务器模式，服务器以单播的方式和每个客户建立连接。由于流媒体服务具有高带宽、持续时间长等特点，随着客户数目的快速增加，服务器的资源很快就会被消耗完，成为系统瓶颈。对于这些问题，现有的解决办法主要是采用因特网内容发布网络（CDN）和 IP 多播技术等。CDN 技术主要是通过代理服务器的复制，将流媒体数据分散到各地，用户进行就近访问，但是其昂贵的成本，始终是阻碍其大规模部署的主要因素。而 IP 多播技术由于自身的种种限制，比如很难实现可靠性多播和拥塞控制及其协议的复杂性等，并没有得到广泛的应用。P2P 网络的出现提出了新的解决办法。在对等网络中，每个对等实体既是服务的提供者，又是服务的享用者，将服务器的负载分散到 Peer 中，从而有效地减轻了服务器的负载和对网络带宽的占用，极大地提高了系统的可扩展性。

10.1.3 P2P 流媒体系统架构

根据源节点提供数据的方式，可把 P2P 流媒体系统分为两种：单源的 P2P 流媒体传输和多源的 P2P 流媒体传输，分别如图 10-1 和图 10-2 所示。

单源的 P2P 流媒体系统建立在应用层多播技术的基础上，是树型多层结构。由一个发送者向多个接收者发送数据，接收者有且只有一个数据源。服务器和所有客户节点组织成多播树，多播树的中间节点接收来自父节点多播的媒体数据，同时将数据以多播的方式传送给其子节点。该结构中，由于每台客户端计算机只单一地从上层节点下载数据，因此上层的节点变化对系统性能的影响较大，而因特网上的用户计算机并不保证服务质量，维持一个稳定的树结构将很困难。另外，每个客户端处理能力有限，大量新增的用户节点将导致树的深度增加，这样就增加了末端计算机的延迟时间，而最下层的节点不再提供上传服务，因此系统内的网络资源并未完全得到利用。

www.ckook.com

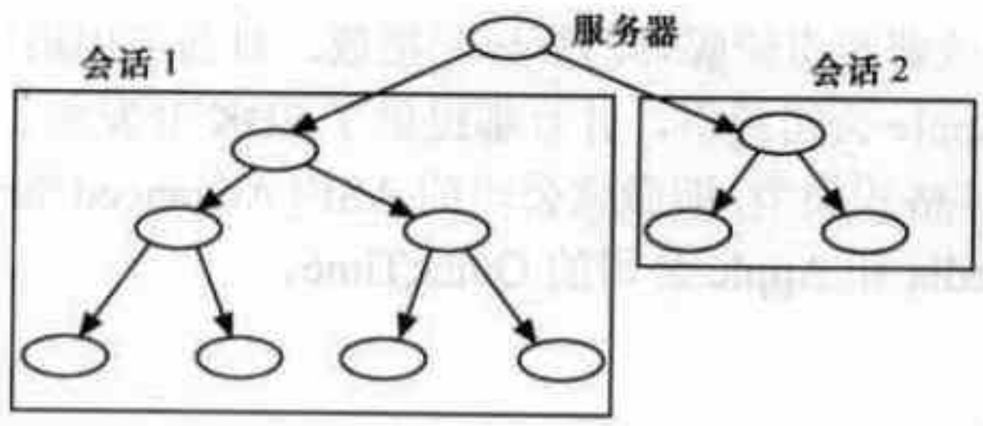


图 10-1　单源 P2P 流媒体传输

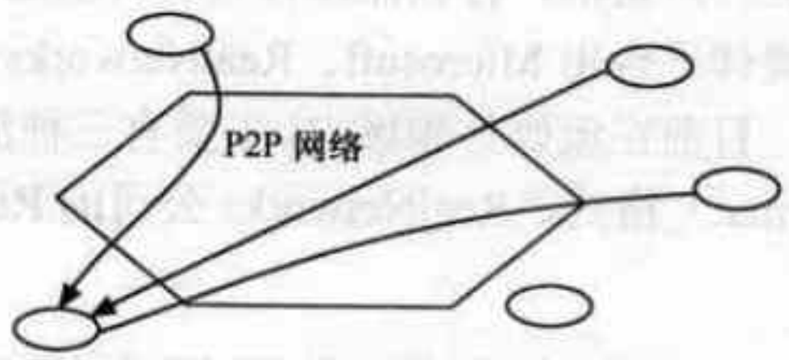


图 10-2　多源 P2P 流媒体传输

多源的 P2P 流媒体传输系统属于网状结构，由多个发送者以单播的方式同时向一个接收者发送。每个客户端节点从多个客户端或服务器上分散下载。缓冲区内的数据按一定大小分块，客户端按一定的数据请求规则向不同的节点发送不同的数据块请求。这样能使速度迅速提高，但技术复杂，涉及如何选择合适的发送 Peer，如何协调多个发送 Peer 之间的传输速率以及如何分配各个发送 Peer 的数据段等问题。目前 P2P 流媒体系统已经由单源向多源转化。

10.1.4　现有系统

目前有较多的 P2P 开源项目，代表了当前的发展方向。有很多成熟的应用就是建立在这些项目上的。可以通过学习这些项目的实现原理，来改造出自己的 P2P 流媒体系统。目前主要的开源项目如表 10-1 所示。

表 10-1　　P2P 流媒体开源项目

项　目	网　站	开发语言	描　述
PeerCast	http://www.peercast.org/	C++	可以用于网络电台的流媒体软件。能够穿越防火墙、NAT
FreeCast	http://www.freecast.org/	Java	
Stream2Stream	http://s2s.sourceforge.net/	Java	
P2Pradio	http://p2p-radio.sourceforge.net	Java	P2P 音视频服务和客户端系统

www.docin.com

10.2　流媒体技术

通常意义上的流媒体是指在 Internet/Intranet 中使用流式传输技术的连续时基媒体，如：音频、视频或多媒体文件。流媒体把连续的影像和声音信息经过特殊的压缩方式分成一个个压缩包，由视/音频服务器向用户计算机连续、实时地传送。用户可以通过播放器一边下载一边观看，不需要等整个压缩文件下载到本地后才可欣赏。为了消除网络连接速率的抖动，播放器先在用户端的计算机上创造一个缓冲区，于播放前预先下载文件的一小段数据作为缓冲，播放程序取用这一小段缓冲区内的数据进行播放。在播放的同时，多媒体文件的剩余部分在后台继续下载填充到缓冲区。这样，当网络实际连接速度暂时小于播放所耗用数据的速度时，就可以避免播放的中断，使得播放品质得以维持。

一般来说流媒体平台由编码器、流媒体服务器和播放器三部分组成。编码器对多媒体文件或即时音/视频信号进行压缩、编码，生成适合以串流方式传输的媒体内容，流媒体服务器

www.ckook.com

响应客户请求，打包流媒体内容并发送；播放器负责流媒体的解码和播放。目前应用最广的流媒体平台由 Microsoft、RealNetworks 和 Apple 公司提供，并且都提供了 SDK 开发包。

目前在流媒体领域中，主要有三种流媒体格式类型，即微软公司的 ASF（Advanced Stream Format）格式、RealNetworks 公司的 RealMedia 和 Apple 公司的 QuickTime。

10.3 设计 P2P 音视频点播系统

10.3.1 多媒体数据压缩

为了有效地利用网络带宽，媒体流通常经过压缩后才进行传输。压缩技术主要可以分为非扩展性编码和扩展性编码，扩展性编码可以适应网络带宽的变化，主要编码技术如下。

（1）基于分块运动补偿及离散余弦变换（DCT）的压缩算法。这是目前工业应用的主流标准，比如 MPEG 系列及 H.26X 系列标准都采用这种算法。

（2）分层可扩展性编码技术（Layered Scalable Coding）。分层可扩展性编码主要有三种，时域可扩展性、空域可扩展性和质量可扩展性。

（3）精细的可扩展性编码（Fine Granular Scalable Coding）。

（4）多重描述编码（MDC，Multiple Description Coding）。它的基本思想就是把一个码流编码成多个码流，并称其为多描述。任何一个描述都可以被用户单独解码，不同的描述也可以叠加解码。

10.3.2 应用层 QoS

www.docin.com

对于实时的视频播放来说，保证 QoS 非常重要。带宽、丢包、网络的异构、误码和延迟等因素都会影响 QoS。应用层 QoS 的主要目的是避免网络拥塞，保证视频质量。应用层 QoS 控制技术主要包括拥塞控制（congestion control）和错误控制（error control）。

（1）拥塞控制。网络拥塞可以引起高的丢包率和时延，从而导致视频质量的急剧降低。拥塞控制机制就是试图减少丢包率及时延。在视频传输中，拥塞控制的主要手段就是对服务器端发送比特率的控制，通过使发送的比特率接近空闲的网络带宽来避免拥塞。码率控制（Rate control）通过估计当前的网络带宽来调整发送的比特率。码率控制可以分为三种：基于发送端（send based）的、基于接收端（receive based）的和混合的码率控制（hybrid rate control）。

（2）错误控制包括前向纠错编码（FEC）、重传机制（Retransmission）、容错编码（Error-Resilient Encoding）和错误隐藏（Error Concealment）技术。

10.3.3 应用层多播技术

IP 多播技术作为 IP 的扩展，可以有效的提供大规模数据的多播支持。然而 IP 多播有其

www.ckook.com

自身的问题，包括网络管理、扩展性、部署和上层协议的实现。应用层多播就是把多播功能从网络协议栈的 IP 层转移到应用层。

10.3.4 流媒体同步技术

在流媒体播放的时候也必须保证视频和声音的同步，否则用户体验会急剧恶化。媒体的同步指的就是维护多媒体数据之间的时序关系，媒体的同步有以下三层。

（1）流内同步：连续媒体或时间相关的数据的最低层是媒体层。媒体层的基本单元是一个逻辑单元，逻辑单元之间需要遵循严格的时序关系。流内同步就是维护逻辑单元之间的连续性和时序性。比如视频中，帧与帧之间的顺序不能被打乱。

（2）流间同步：时间相关数据的第二层是流层。流层的基本单元就是整个流。这一层的同步就是维护各个不同流之间的时间关系，比如维护音频视频之间的同步。

（3）对象间同步：多媒体数据最高层就是对象层。这一层整合了流和时间无关的数据，比如文本和图像，这一层的同步就是对象间的同步。对象间同步的目的就是当时间无关的数据到达预先设定需要显示的点时，在一个可以忍受的时间间隔内，开始和停止显示这些时间无关的数据。

媒体数据从发送端传输到接收端会使数据失去同步，传输时延不可预期。同步机制的目的就是保证接收端以正确的方式显示接收到的媒体数据。媒体同步机制中，最基本的就是指定媒体流内或媒体流之间的时序关系。指定时序关系的方法包括基于时间间隔的（interval-based）、基于轴的（axes-based）、基于流量控制的（control flow-based）和基于事件的（event-based）。在连续媒体中广泛使用的方法是基于轴的或者称为基于时间戳（time-stamping）的。数据源的每个流都有自己的时间戳。在接收端，应用程序通过时间戳来显示媒体流。

www.docin.com

10.3.5 PeerCast 实现分析

1．PeerCast 概述

PeerCast.org 成立于 2002 年 4 月，它是一个非营利性的站点，提供免费的 P2P 电台软件。这个项目的目标是创建一个容易使用、简单和可靠的软件客户端，从而使任何人都可以传播流媒体，而不必使用昂贵的服务器或者耗费带宽。

PeerCast 是一个健壮的网络。因为不存在中心服务器，每个用户既可以是客户，也可是服务器，也可以是流的广播者。它为广播者提供了匿名服务，因为要追踪到源文件流并不是件简单的事。

PeerCast 是一个把音视频服务器和客户端集合在一起的软件。用户可以通过 PeerCast 来收听众多的网络电台，也可以自己广播。PeerCast 最重要的特性是不需要一个有庞大带宽的服务器来为众多听众提供广播服务，所需要的只是 PeerCast 和一个外部的广播工具。

Peercast 和其他 P2P 文件共享软件的工作方式大部分相同，除了一点，即用户下载的不是文件而是流。然后这些流实时地与其他用户进行交换。对于任何连接到网络的机器来说，

www.ckook.com

没有任何数据会被存储到本地设备上。

2．Peercast 多播树

在基于树状拓扑协议及扩展的模型中，首先要解决的问题是多播树的构建。最简单的模型是 PeerCast 模型。在 PeerCast 中，节点被组织成一个树状结构，树的父节点给子节点提供服务。在 Peercast 中，节点的加入和离开策略都很简单，但也容易导致树的不平衡。在多播树中，节点离根节点越远，则数据的传输时延就越大，故树的深度应该尽可能短。但是每个节点的有限输出带宽限制了节点的宽度。理想的多播树应在深度和宽度之间能够有效的平衡。事实上，当所有节点的深度都为 1 的时候，PeerCast 多播树模型就退化成了传统的客户—服务器模型了。

PeerCast 的流媒体应用层多播树示意图如图 10-3 所示。

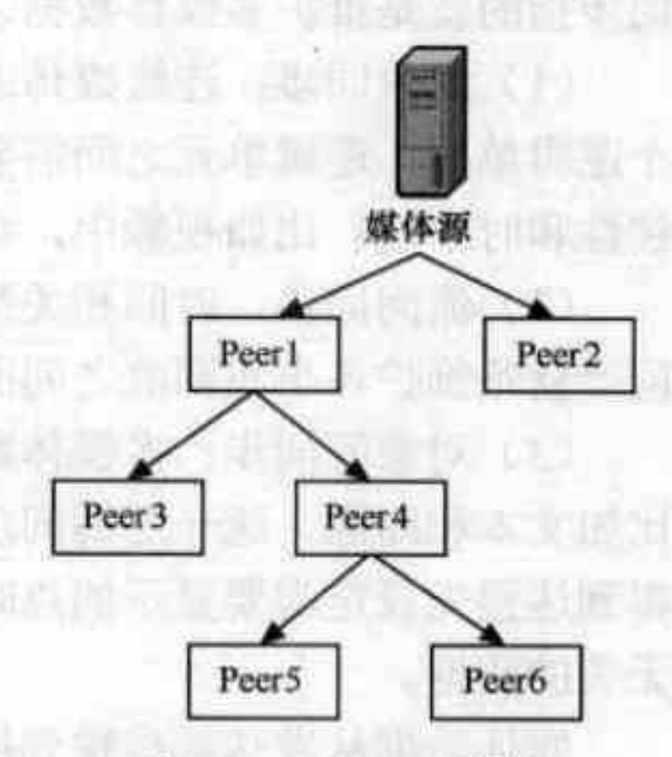


图 10-3　PeerCast 多播树

Peer1 和 Peer2 从媒体源获得数据，Peer3 和 Peer4 从 Peer1 获得数据，Peer5 和 Peer6 从 Peer4 获得数据。

由于各子节点参与分发数据，服务器的负载就被分散到整个网络中。要建立应用层的多播树，必须解决以下的问题。

（1）新节点的加入。当一个新节点加入网络时，如何为该新节点分配一个父节点。

（2）节点的离开。正常离开和异常离开两种情况下，如何重新构建树。

3．新节点的加入

新节点的加入过程如图 10-4 所示。

www.docin.com

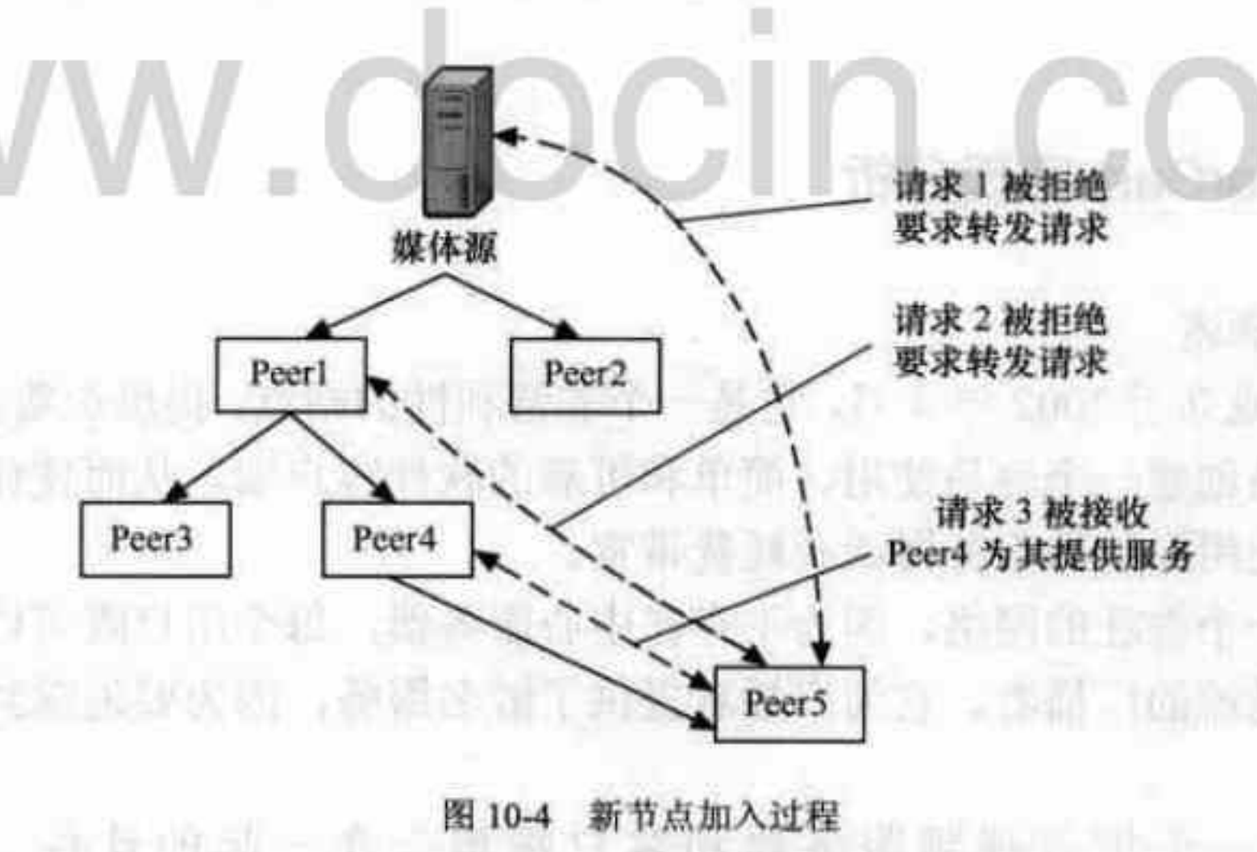


图 10-4　新节点加入过程

新节点 Peer5 试图加入时，首先向服务器请求服务，如果服务器有足够的资源，则服务器向 Peer5 提供服务，否则服务器把 Peer5 的请求转发给其某个直接的子节点，比如 Peer1。Peer1 子节点根据自己的资源情况判断是否给 Peer5 提供服务，以此类推直到 Peer5 找到一个父点，比如 Peer4。在这个模型中，每个节点仅仅维护自己的父节点和直接子节点的信息。

www.ckook.com

转发节点加入请求的策略如表 10-2 所示，假设 PeerN 向 PeerE 请求。

表 10-2 节点加入请求转发策略

转发策略	转发过程
随机选择	PeerE 随机选择其子节点进行转发
Round-Robin	PeerE 将请求依次转发给自己的子节点队列
Smart-Placement	PeerE 将请求转发给离 PeerN 路由最近的子节点
Smart-Bandwidth	若 PeerN 的带宽情况好于 PeerE，则 PeerE 将自己的位置给 PeerN，PeerE 重定向到 PeerN

4．节点正常离开

当节点 PeerL 离开的时候，如果 PeerL 是叶子节点，则 PeerL 只需要向其父节点发一个退出的请求，通知 PeerL 的父节点释放所有为 PeerL 提供服务的资源，多播树的其他节点维持不变。

如果节点 PeerL 不是叶子节点，则节点 PeerL 的离开会导致 PeerL 的所有后续子节点失去服务，所以在节点 PeerL 离开之前，必须为 PeerL 的所有子节点重新分配父节点。

重新分配父节点有两种模式。

（1）PeerL 的所有后续节点的每一个节点都被重新加入。

（2）维持节点 PeerL 后面的子树不动，也就是说节点 PeerL 只把它的每个直接子节点作为一个子树重定向给新的服务节点 PeerS。这种模式由于不需要重新加入所有子节点，树的修复会更快。

节点 PeerL 至少知道服务器 S 和节点 PeerL 的父节点 PeerP 可以给 PeerL 的子节点提供服务。所以四种离开策略如下：

（1）节点 PeerL 向其所有直接子节点发消息，通知它们通过 PeerL 的父节点修复。

（2）节点 PeerL 向其所有后续节点发消息，通知它们通过 PeerL 的父节点修复。

（3）节点 PeerL 向其所有直接子节点发消息，通知它们通过服务器 S 修复。

（4）节点 PeerL 向其所有后续节点发消息，通知它们通过服务器 S 修复。

5．节点异常离开

每个节点需要周期性的向它们的父节点和子节点发送消息，说明自己工作正常。如果某个节点在几个周期内都没有收到子节点正常的消息，则认为子节点已经非正常离开，该节点将回收服务该子节点的资源。同样子节点在几个周期内都没有收到父节点正常的消息。则认为父节点已经非正常离开，该节点以及所有后续节点都需要重新加入网络。

6．PeerCast 代码结构

在 PeerCast 的官方网站 http://www.peercast.org 上可以获得其工程源代码，不过请注意，trunk 目录下最新的源代码有可能是运行不了的，请下载 tags 目录下可用的版本。

安装了 svn 后，在命令行下运行以下命令获取源代码。

```
svn list svn://peercast.org/peercast/tags
```

此命令将列出可用的版本目录，最新版本为 0.1217，运行以下命令。

```
svn co svn://peercast.org/peercast/tags/v0.1217
```

在当前目录下可以看到工程目录 v0.1217。

PeerCast 工程目录下有两个目录。

www.ckook.com

（1）core 目录，包含实现 PeerCast 核心功能的模块。此目录下有 Common、Unix 和 Win323 个目录。Common 目录实现通用操作模块，Unix 目录下是和 UNIX 系统相关的类，Win32 目录下是和 Windows 相关的类。

（2）ui 目录，包含实现软件交互的界面。此目录下的 html 目录是多语言的网页文件，用来进行系统设置。Templates 目录是网页模板，Win32 目录是 Windows 界面实现。

用 VC6 打开工作区“\v0.1217\ui\win32\PeerCast.dsw”，会有两个提示要加载其他的工程文件，单击取消即可。直接编译生成“PeerCast.exe”文件在“\v0.1217\ui\win32\simple\Debug”目录下。将 PeerCast.exe 与目录 html 和目录 Templates 放在同一目录下就可以运行了，运行界面如图 10-5 所示。

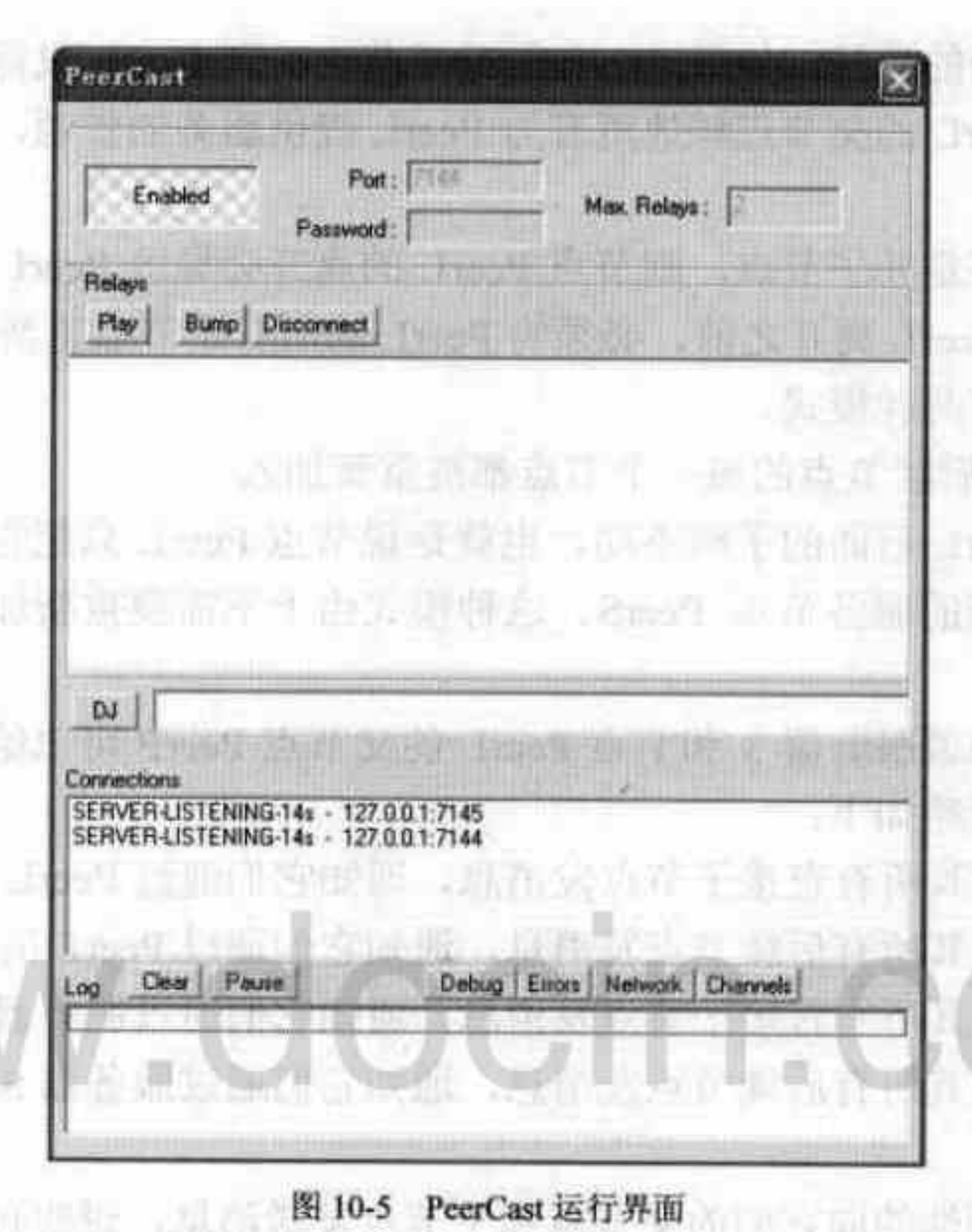


图 10-5　PeerCast 运行界面

整个工作区包含三个工程。

（1）Corelib 工程。PeerCast 的主要功能都由 Corelib 实现，它通过 peercast.h 导出两个类 PeercastInstance 和 PeercastApplication。

（2）Simple 工程。Simple 工程是程序的主体实现，通过继承 peercast.h 导出的两个类实现程序功能。

（3）winamp2 工程。winamp2 是程序响应后调用的播放插件。

7．PCP 协议实现

PCP 协议，也就是 PeerCast Protocol，是 PeerCast 用来在节点间传送信息所定义的协议。协议的实现在 core\common 目录下的 pcp.h 和 pcp.cpp 文件中。

pcp.h 中定义了 PCP 协议字段，比如主机信息字段都是以 PCP_HOST 开始，如下所示：

```
static const ID4 PCP_HOST              = "host";
static const ID4 PCP_HOST_ID           = "id";
```


www.ckook.com

```
static const ID4 PCP_HOST_IP          = "ip";
static const ID4 PCP_HOST_PORT        = "port";
static const ID4 PCP_HOST_NUML        = "numl";
static const ID4 PCP_HOST_NUMR        = "numr";
static const ID4 PCP_HOST_UPTIME      = "uptm";
static const ID4 PCP_HOST_TRACKER = "trkr";
static const ID4 PCP_HOST_CHANID      = "cid";
static const ID4 PCP_HOST_VERSION  = "ver";
static const ID4 PCP_HOST_FLAGS1      = "flg1";
static const ID4 PCP_HOST_OLDPOS      = "oldp";
static const ID4 PCP_HOST_NEWPOS   = "newp";
```

这些信息的发送和接收由 AtomStream 类来完成，AtomStream 类在 atom.h 中实现。

8．播放模块实现

和频道流有关的类在 channel.cpp 中实现，其中实现了 Peer 之间媒体流的控制。当用户点播一个频道后，会调用频道管理类 ChanMgr 中的 playChannel 方法，playChannel 方法会调用 sys→executeFile 方法启动特定的播放程序，具体实现是通过 ShellExecute 函数调用。

PeerCast 本身并不包含媒体播放器。它是通过本地转发媒体流到本地播放器来实现流媒体播放的，转发时使用 http 协议。

10.3.6 改造 BitTorrent 成为流媒体系统

由于 BitTorrent 是针对传输一般文件所设计，为了快速分享与下载，其片段选择方法会令已下载的部分非常分散，以致不适用于传输影音多媒体流。因此，需要将提出针对使用 BitTorrent 下载影音多媒体文件的较适合的片段选择方法。与一般 BitTorrent 采用的分散片段选择方法比较，所提出的片段选择法应该可以让影音多媒体文件未完整下载前就可以实时播放，并且可以让用户流畅且连续地收看完影片，减少跳帧的产生，且设计上不会改动 BitTorrent 的对等协议，可以兼容于现有的 BitTorrent 环境，即来源 Peer 使用的软件也可以是一般的 BitTorrent 软件。发布视频流也就是发布一个种子，按照改造的片段选择算法下载视频流，然后客户端调用播放器播放。

10.4 本章总结

本章介绍了 P2P 网络和语音通信及视频点播相结合的主要方面。这一领域是一个新兴的领域，需要读者不断的去尝试。但是重点还是一些基本的概念，只有清楚这些基本的技术应用，才能有创新的技术结合产生。

10.5 练习题

（1）阐述 P2P 语音通信的技术架构。

www.ckook.com

（2）阐述 P2P 流媒体系统的技术架构。

（3）尝试修改 BitTorrent 的文件片段选择算法，使之适合视频的实时播放。

参 考 文 献

[1] 钟玉琢，向哲，沈洪．流媒体和视频服务器.北京：清华大学出版社，2003

[2] 龚海刚，刘明，毛莺池．P2P 计算研究现状及关键技术.计算机研究与发展，2005

[3] 杨再晗，陈建二，王建新．P2P 流媒体关键技术的研究进展.现代电子技术，2004

[4] http://www.peercast.org

www.docin.com

www.ckook.com

CHAPTER 11

第 11 章　P2P 视频点播系统开发实例——Myseelite 的设计与实现

Myseelite 是 Mysee 独立研发的网络视频流媒体播放系统。它采用国际通行的双重协议，用户可以在 GPL 协议下免费地编译和使用 Myseelite 系统。对于希望对 Myseelite 进行修改和定制的用户，需要按照 GPL 协议将自己的改进版本开源。如果因为商业或者安全原因不方便或者不愿意开源，则需要支付一定的版权费用。本章将针对 Myseelite 的运行原理、系统结构进行介绍，并且对部分模块的代码进行详细的分析。

在学习了本章后，希望读者能够掌握以下几点：

（1）了解该系统的原理；

（2）了解该系统的架构；

（3）掌握该系统的使用方法；

（4）了解该系统的优缺点。

11.1　Myseelite 系统概述

www.docin.com

Myseelite 是 Mysee 公司独立开发的网络视频流媒体播放系统。它的源代码首先在 OpenMysee 社区上开放。该社区是由 Mysee 公司推进成立的，该社区在 Mysee 公司的指导和帮助下独立运行，拥有独立的知识产权，是该公司与开源社区互动和交流的管道。在应用了 P2P 技术和一系列先进的流媒体技术之后，Mysee Lite 能够在现有网络带宽的基础上极大地提高网络视频的质量，包括画面的流畅、清晰度以及用户并发数。该系统能够应用于网络视频直播、大规模电视会议、大规模远程教学、IPTV 等多个领域。本节将针对 Myseelite 系统的背景、设计目标等进行简要的阐述，使读者对该系统有一个初步的认识。

11.1.1　P2P 视频点播系统的背景

视频点播系统是分布式流媒体的一种重要的应用系统。视频点播系统也被称为 VoD 系统，其全称是 Video on Demand，即按需要的视频流播放。视频点播技术是当前流媒体通信应用的一项前沿技术，是面向未来的信息、

www.ckook.com

通信等相关技术相结合的新技术。同时，视频点播又是一项业务，它泛指利用交互式网络将声音图像进行实时传输，以实现影视点播、信息查询、电视购物、远程教育等服务的一项交互式多媒体应用业务。

因特网上的传统流媒体系统一般是基于 C/S 模式的，通常包括一台或多台服务器和若干客户端。在 C/S 模式下，流媒体系统容量（同时服务的客户端数量）主要由服务器端的网络输出带宽决定，有时服务器的处理能力、内存大小、I/O 速率也影响到系统的容量。在 C/S 模式下，由于传输流媒体占用的带宽大，持续时间长，而服务器端可利用的网络带宽有限，所以即使是使用高档服务器，其系统容量也不过几百个客户，不具有经济规模性。另外，由于因特网的不确定性，如果客户端距离服务器较远，则流媒体传输过程中的延迟、抖动、带宽、丢包率等指标也将更加不确定。服务器为每一个客户单独发送一次流媒体内容，故网络资源的消耗也十分巨大。所以在当前资源有限的条件下，如何满足不断增长的用户需求，并在确保服务质量的同时，降低流媒体的服务成本，成为流媒体技术研究中的重要课题。近几年兴起的 P2P 技术，能利用因特网中的各个节点进行对等计算，充分利用因特网上的空闲资源，允许两个客户端直接交换信息，因此受到了广泛关注。将 P2P 技术应用于流媒体业务，为解决服务器端网络带宽和服务器能力限制问题提供了一种新的思路。在此背景下，基于 P2P 技术的视频点播系统产生了。

11.1.2 Myseelite 的设计目标

Myseelite 是一个基于 P2P 技术的视频点播系统，目前有两个版本，Myseelite1 和 Myseelite2。Myseelite1 是用纯 C 语言写成，整体架构和协议有所欠缺。为了促进开源项目 Myseelite 的更好发展，Mysee 公司对 Myseelite 进行了较大的改进，推出 Myseelite2。

Myseelite2 的主要设计目标如下。

（1）支持多种流传输格式。支持使用者最多的 WME、WMS、Helix 服务器，支持 WMV 和 Real 格式的传输。今后可以扩展支持 FLV、H264 等格式。

（2）小幅修改了原有 Myseelite1 的协议。编写了新的 SP，取消的 CP 层。

（3）根据协议重新编写了 Tracker 服务器。

（4）编写新的 CaptureServer，下载 mmst、mmsh、rtsp（real）格式的流，并使用 Myseelite 的协议打包发送到 SP。

（5）修改客户端的播放方式，使用 HTTP Server 向播放器提供数据。同时根据协议修改客户端的下载调度。

（6）实现跨平台。系统采用 ACE 跨平台库开发，目前除了客户端外（客户端的跨平台将在下一个小版本实现），其余各部分均已实现跨平台，可以在 ACE 所支持的平台（如 Windows、大部分的 Linux 发行版、大部分的 UNIX 发行版）上运行。

Myseelite2 与 Myseelite1 的比较如表 11-1 所示。

表 11-1　　Myseelite2 与 Myseelite1 的比较

比 较 项	Myseelite2	Myseelite1
支持平台	采用 ACE 跨平台库，支持大部分 Linux，UNIX 发行版等多种 ACE 所支持的平台	SP、CP、TS 部署在 Linux 平台，Capture、Client 部署在 Windows 平台
下载协议	MMSH、MMST、RTSP	MMS

www.ckook.com

续表

比 较 项	Myseelite2	Myseelite1
客户端播放方式	Httpserver 方式	DirectShow 方式
CP 部署协议	合并到 SP	单独部署

11.2 Myseelite 系统原理

Myseelite 系统作为一个 P2P 视频点播系统，支持多种流媒体协议。为了支持系统的跨平台性，Myseelite2 在网络编程方面使用了 ACE 组件，在客户端界面上使用了 wxWidgets 库。本节将针对该系统的基本概念、流程设计以及 ACE 组件和 wxWidgets 库等进行简要的描述，使读者对 Myseelite 系统的构建基础有一个全面的了解。

11.2.1 基本概念

在 P2P 视频点播系统中，P2P 流媒体和 P2P IPTV 网络是两个非常基本的概念。本小节将对这些概念进行简单的介绍。

1．P2P 流媒体

P2P 流媒体，是指用 P2P 的方式传输流媒体数据。它要解决的问题是特定目标流媒体的传输时，如何用 P2P 的方式优化资源。在流媒体数据的传输过程中，包括很多资源：带宽、运算力、节点的缓冲区、节点稳定在线时长等。P2P 流媒体的最终目的，就是要通过资源的优化调度，使得数据块在适当的时间被适当的节点持有，达到尽量多的节点能够流畅播放流媒体的效果。

P2P 流媒体可以细分为 3 种应用模式：直播、点播和下载。这 3 种模式是可以相互转化的。点播的实质就是可以使指定文件偏移位置的数据在一定范围内顺序到达，通过计算偏移量对数据重新定位。如果只是切换频道，那不是点播，而是轮播。点播和下载传输的都是静态文件中的数据块，直播和轮播传输的都是实时接入系统的数据片段。

P2P 传输可以针对流媒体的特点进行优化，同理，流媒体也可以针对 P2P 的特点进行优化。采用高性能的视频、音频编码算法，能够极大地缓解 P2P 传输的压力。索引区块的优化可以让点播的 Seek 过程更加高效。特殊的数据打包格式可以加入跟随流同步的各类信息，如字幕、广告、图片、程序动作等。把静态文件编码成为在网上传输的数据块集合，可以优化从读取文件到网络发送的过程。由于点播、轮播、下载可以进行预先编码，可以使用计算复杂度高但非实时的编码算法，故可以获得更高的质量。而现场直播必须使用实时编码。

2．P2P IPTV 网络

基于 P2P 的一种可能的 IPTV 网络结构如图 11-1 所示，省间的骨干网按地理位置分成几个大区，每个大区中心架设一个区域中心媒体服务平台（ACMS，Area Central Media Station），用于存储过期的（比如依据直播后 24h 区分是否过期）点播内容以及最新的公众性直播节目。边缘媒体服务平台（EMS）用于存储地方性直播和最新的（如前所述的 24h 内）点播节目。

www.ckook.com

存储时可将地方性节目存储在当地服务器中，公众性的节目按照均摊的思想存储在剩余空间较大的服务器中。

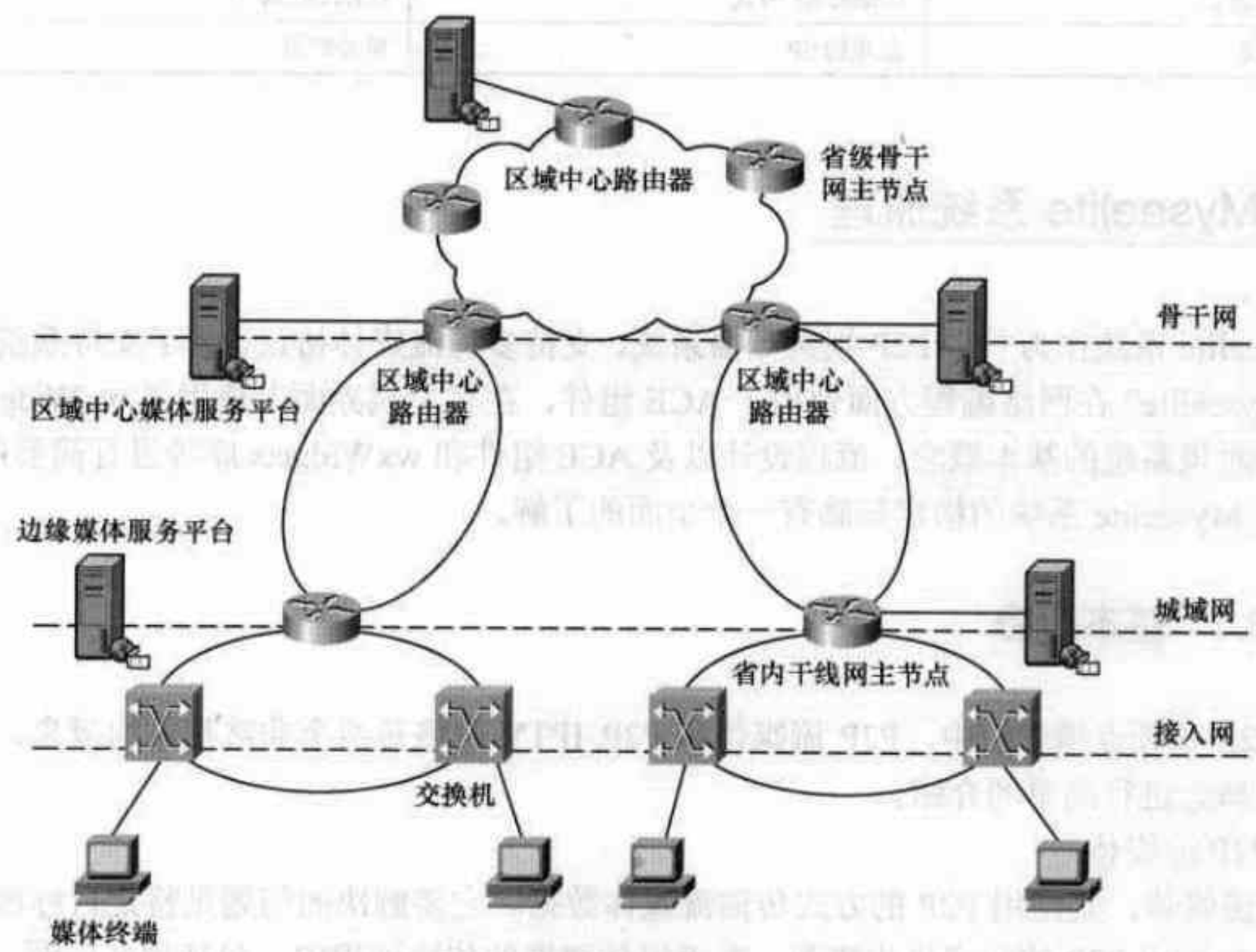


图 11-1　基于 P2P IPTV 的 CDN 拓扑结构

该网络的数据获取原则是：优先从低级数据源获取数据，高级数据源提供稳定性支持。如果划分优先级，从低到高依次为用户驻地网、接入网、骨干网、核心网。

把省间的核心网按地理位置分成几个大区，每个大区中心架设一个区域中心媒体服务平台（CMS），用于存储过期的点播内容以及最新的直播和点播节目。由于大容量存储设备较为昂贵，各 CMS 之间分布式差异化存储陈旧的内容，可以节约大量的存储成本。

P2P TV 的数据内容可以按“节目录入位置+日期+节目编号+节目类别码”这样类似于身份证编码的方式进行命名。以北京地区 2006 年 2 月 24 日的《新闻联播》为例，如果新闻的节目类别是 A，那么可命名为：AA0001 20060224 AAAAB A。此字段中，第 1 字段 AA 表示北京，0001 表示节目进入 IPTV 系统的媒体平台编号；第 2 字段 2006224 是录入日期；第 3 字段 00001 是节目的三十六进制编号，5 位三十六进制是很丰富的码号资源了；第 4 字段 A 表示节目类型是新闻。这里采用二十六进制，码号也很丰富，可以提供内容分级服务。明确的节目编号方便内容审察，统一管理，便于检索，还可以加入节目录入人员编号等信息。这个编号可以用作文件名，也可以作为一个文件别名或者备注信息。

11.2.2　工作机制

Myseelite2 由 Capture、Super Peer、Tracker、Client 共 4 个子系统构成，其组成结构如图 11-2 所示。

www.ckook.com

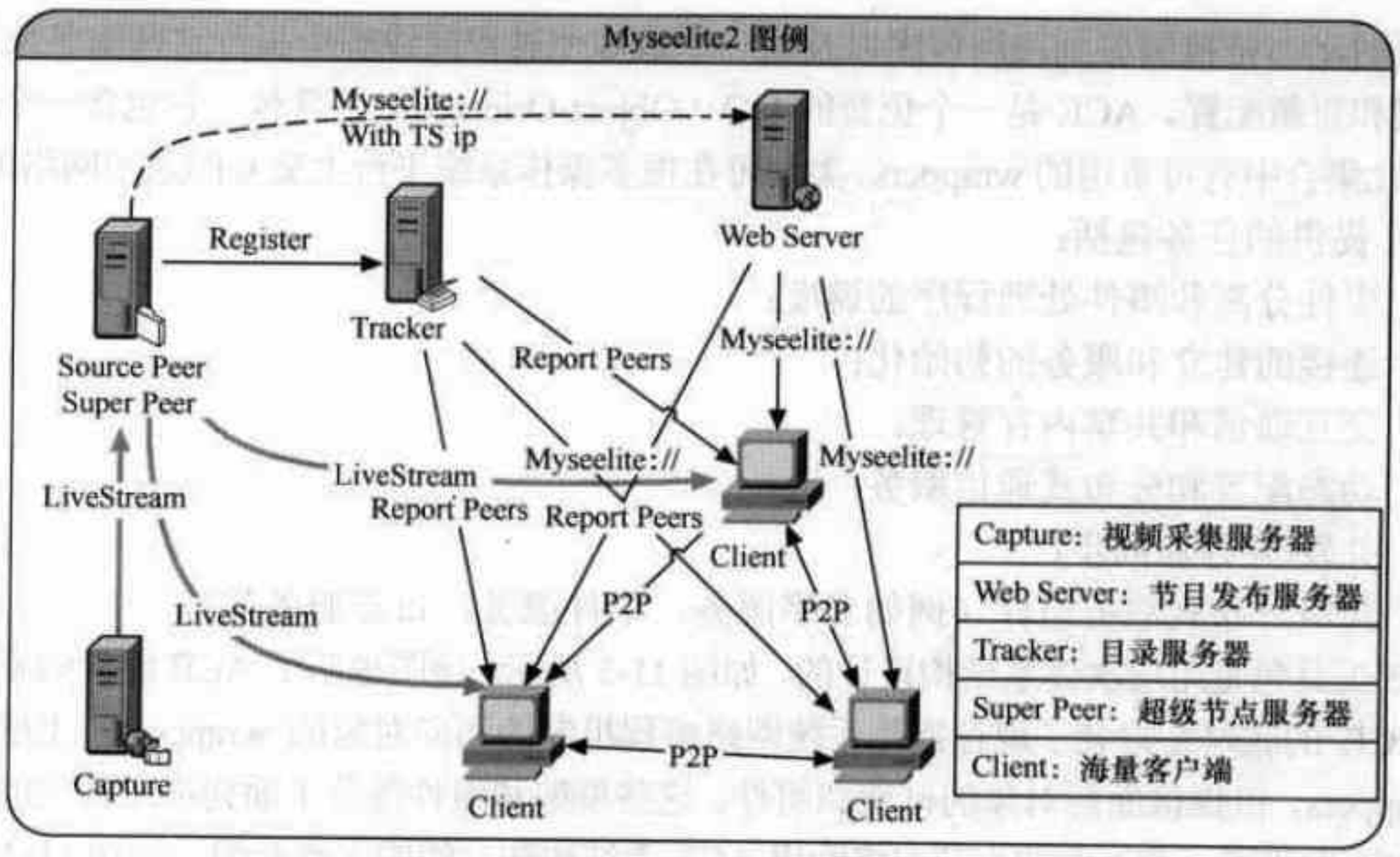


图 11-2 Myseelite2 结构图

Myseelite2 的 4 个子系统的定义如表 11-2 所示。

表 11-2 Myseelite2 子系统定义

子 系 统	功 能 定 义
Capture	其主要功能是获取直播流。在 Myseelite2 中，Capture 可以从 Windows Media Encoder（WME）、Windows Media Server（WMS）、Helix 获取节目流。可以支持多种格式的下载流，如 mmst、mmsh、rtsp（real）
Super Peer	其主要功能是接收 Capture 传送的流，作为整个 P2P 系统的源节点和缓冲节点，为 Client 提供直播流
Tracker	其主要功能是为 Client 提供下载信息。接收 Client 信息，返回 Client 要求的流的 SuperPeer 地址和拥有此流的 Client 节点
Client	其主要功能是从 SuperPeer 和其他 Client 处获取直播流，并且提供给 GUI 界面进行播放

www.ccom.com

Myseelite2 按照下面的流程完成登录。

（1）Capture 获取直播节目流，经过打包后上传到 Super Peer。

（2）Super Peer 将此节目流发布到 Web Server，并且向 Tracker 注册此节目流。

（3）Client 登录 Web Server，获取 myseelite 协议，从 myseelite 协议串中获得 Tracker 的 IP 地址以及节目信息。

（4）Client 登录 Tracker，Tracker 根据节目信息，返回 Client 要求节目的 Super Peer 地址和拥有此节目流的 Client 列表。

（5）Client 依次连接 Super Peer 和拥有此节目流的 Client，获得节目流。

11.2.3 ACE 简介

ACE 是一个面向对象的工具开发包，它实现了通信软件的基本设计模式。ACE 面向在 UNIX 和 Win32 平台上开发高性能通信服务的开发人员。它简化了面向对象的网络应用程序和服务的开发，这些程序和服务用到了进程间通信、事件分离、直接动态链接和并发机制。ACE 通

www.ckook.com

过在运行时动态链接服务到应用程序以及在一个或多个进程或线程中执行这些服务来自动完成系统配置和重新配置。ACE 是一个免费的 OO（Object-Oriented）工具包，它包含一个内容丰富的集合，此集合中有可重用的 wrappers、类和可在很多操作系统平台上交互的通用网络编程框架。

ACE 提供的任务包括：

（1）事件分离和事件处理程序的调度；

（2）连接的建立和服务的初始化；

（3）交互通信和共享内存管理；

（4）动态配置和分布式通信服务；

（5）并发/并行和同步；

（6）高级分布式服务组件（例如名字服务，事件服务，日志服务等）。

ACE 工具包是用层次体系结构设计的，如图 11-3 所示，该图说明了 ACE 组件的垂直和水平关系。ACE 的底层是封装了现存操作系统网络编程机制的面向对象的 wrappers。上层扩展了底层的 wrappers，以提供面向对象的框架和组件。这些框架和组件覆盖了面向应用程序的网络任务和服务的较大范围。在下面的小节中将给出 ACE 类结构和功能的大致介绍，如图 11-4 所示。

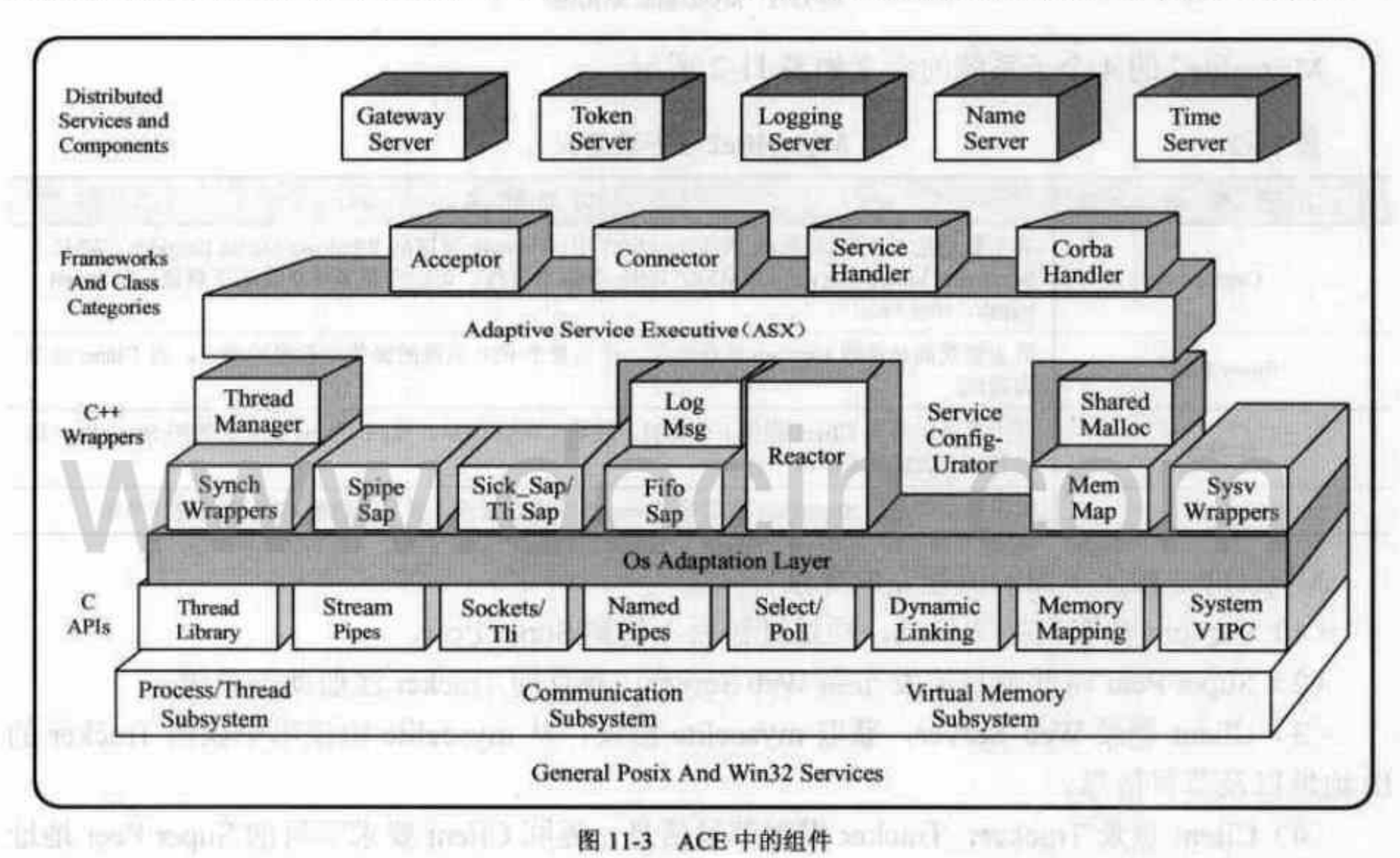


图 11-3　ACE 中的组件

1．ACE 操作系统适配层

ACE 源码包含超过 85 000 行 C++语句。大约 9 000 行代码属于操作系统适配层。该层将 ACE 的上层和与平台相关的非独立性屏蔽起来。它包括以下操作系统机制：

（1）多线程和同步；

（2）进程间通信；

（3）事件分离；

（4）直接动态链接；

（5）内存映射文件和共享内存。

www.ckook.com

2. ACE OO Wrappers

在操作系统适配层之上的是面向对象的 wrappers。这些 wrappers 封装和加强了并发性、进程间通信（IPC）和现代操作系统的虚拟内存管理机制。应用程序可以通过有选择的继承、聚合和/或实例化下列 ACE 包的分类来结合以及组织这些组件。

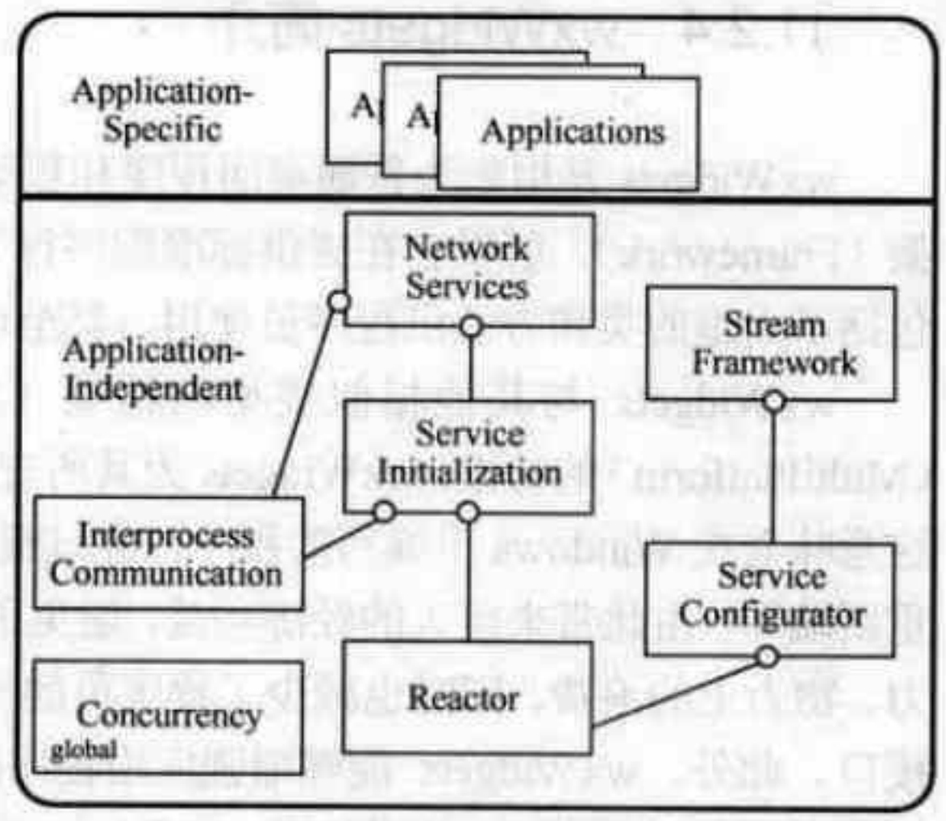


图 11-4 ACE 的类包的结构

（1）IPC SAP——封装了本地和/或远地的服务访问点（IPC SAP 机制，例如 Sockets、TLI、UNIX FIFOs 和 STREAM 管道，以及 Win32 命名管道）。

（2）访问初始化——ACE 提供了 CONNECTOR 和 ACCEPTOR 组件的集合。在初始化完成后，这些组件从通信访问执行的任务中分别分离主动和被动初始化角色。

（3）并发机制——ACE 抽象了底层操作系统的多线程和多进程机制（例如互斥和信号量）以生成高级的面向对象并发抽象（例如主动对象）。

（4）内存管理机制——ACE 内存管理组件为管理共享内存和本地内存的动态定位和内存单元分配提供了一个灵活的可扩展的抽象。

（5）CORBA 集成——ACE 可以和 CORBA 产品集成起来（例如单线程和多线程 Orbix）。

OO wrappers 的使用通过类型安全的面向对象接口封装操作系统的通信、并发和虚拟内存机制提高了应用程序的顽健性。这样就避免了应用程序直接访问底层操作系统库，而底层操作系统库是用弱类型的 C 接口实现的。因此，OO 语言的编译器，例如 C++和 Java 可以在编译时检测到类型系统冲突，而不是在运行时。C++版的 ACE 使用了内联函数扩展以消除性能代价，否则 wrapper 层附加的类型安全和抽象会导致性能下降。

3. ACE 框架

ACE 包含一个高级的网络编程框架，它集成并加强了底层 OS wrappers。这个框架支持由应用程序服务组成的动态配置和并发网络后台程序。ACE 的框架部分包含下列分类。

（1）Reactor——ACE Reactor 提供了可扩展的、面向对象的多路输出，根据不同事件的类型进行处理器调度（如基于 I/O 的事件、基于定时器的事件、信号和同步的事件等）。

（2）Service Configurator——ACE Service Configurator 支持在安装或运行时配置服务的应用程序结构。

（3）Streams——ACE streams 组件简化了包含一个或多个层次相关服务的并发通信软件开发（例如协议栈）。

4. ACE 网络服务组件

除了 wrappers 和框架，ACE 还提供了一个标准的网络服务组件库。这些组件在 ACE 中有以下两个角色。

（1）它们解释了如何利用 ACE IPC wrappers、Reactor、Service Configurator、Service Initialization、Concurrency、Memory Management 和 Streams components；

（2）它们提供了通用分布式系统任务的可重用组件，例如日志、命名、时间同步等。

当和 OO 语言特性以及设计模式相结合时，可重用的 ACE 组件方便了通信服务的开发，

www.docin.com

还方便了那些需要在不修改、不重新编译、不重新链接的情况下可以升级的应用程序的开发。

11.2.4 wxWidgets 简介

wxWidgets 是用来为普通桌面程序和移动设备程序开发用户界面的工具包。由于它是一个框架（Framework），所以它在提供标准程序行为的同时也消除了程序底层的复杂性。wxWidgets 库包括了大量的类和方法供程序员使用，甚至可供程序员定制其中某些类别以满足不同的需求。

wxWidgets 与其他相似类型的框架（如 MFC、OWL）不同的一点是，它具有跨平台（MultiPlatform）的特性。wxWidgets 为其所支持的各种平台提供几乎一致的应用程序接口（API），这意味着在 Windows 上编写的程序，经过很少的修改（甚至原封不动），就可以在 Linux 平台上重新编译。由此带来巨大的经济利益，避免了同一软件需要在不同平台上全部重新编写时产生人力、物力上的浪费。同时也减少了程序员的负担，因为他们不需要再去强记不同平台的应用程序接口。此外，wxWidgets 能够增强应用程序的持久性。随着计算机行业的不断发展，wxWidgets 框架也随之不断更新，使得基于 wxWidgets 框架的程序能够轻松移植到最新最好的系统平台上去。

另一点不同是 wxWidgets 提供的本地外观（Native look and feel）。有些框架为不同平台提供相同的窗体效果，或者使用某些预先设定的主题来实现与平台相仿的外观。相对而言，wxWidgets 使用了本地的窗口风格，所以程序外观不仅看起来与主平台一致，而且事实上它就是本地外观，这对于用户来说是非常重要的。因为程序运行时，它所产生的行为相对平台标准有一丁点的不同，都会令用户有不好的感受。

11.3 Myseelite 系统设计

www.dnzin.com

Myseelite 工程主要包括 4 个子系统：Capture、Super Peer、Tracker 和 Client。本节将就这 4 个子系统的主要结构以及接口进行概括性地分析和描述，为读者学习其源代码奠定基础。

11.3.1 Capture 子系统

CS（Capture Server）是 Myseelite 系统中一个相对独立的子系统。它的主要功能是采集 mms、rtsp 流，并向 SP（Super Peer）提供直播数据。目前的版本中，CS 主要完成的功能就是把采集的 mms、rtsp 流用 SP 所支持的协议传输到 SP。未来扩展的功能可能包括：支持本地采集并编码音视频流，把本地文件虚拟成直播流等音视频流传输到 SP。以下将讨论采集 mms、rtsp 流的处理和转发。CS 的数据流程如图 11-5 所示。

图 11-5 Capture 数据流程图

图 11-5 中的 mms/rtsp 流采集模块和 SP 协议处理模块就是 CS 子系统需要完成的功能模块。其中 mms/rtsp 流采集模块使用了一个名叫 dget 的开源项目的代码。SP 协议处理模块是

www.ckook.com

按照 CS 与 SP 通信协议进行处理的，此协议定义在 protocol.h 文件中的 CSandSP 协议部分。

1．系统结构

从整体上来看，CS 子系统主要由 3 个功能模块组成：流采集模块、与 SP 的通信模块和用户交互接口模块。系统结构图如图 11-6 所示。图中的双箭头表示数据流，也表示状态信息和控制流。

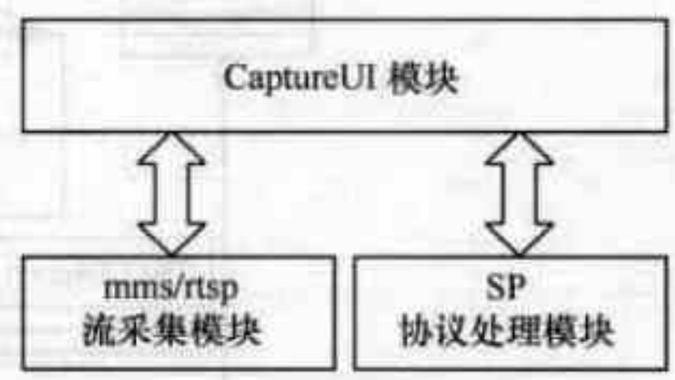


图 11-6 Capture 系统结构图

2．子模块内部设计

在 CS 的 3 个子模块中，UI 部分就不再介绍，它用的是简单的 MFC 界面设计。下面着重介绍 mms、rtsp 流采集模块和 SP 协议处理模块。

mms、rtsp 流采集模块是从开源项目 dget 中经过很小的修改而成的。此模块从功能上看主要由两部分组成，mms 模块和 rtsp 模块，分别实现对 mms 流和 rtsp 流的接收处理功能。

mms 流处理是通过类 TMmsCliEx 实现的，在文件 MmsCliEx.h 中定义，在文件 MmsCliEx.cpp 中实现。rtsp 流处理是通过类 TRtspCliEx 实现的，在文件 RtspCliEx.h 中定义，在文件 RtspCliEx.cpp 中实现。

SP 协议处理模块负责处理 CS 与 SP 之间的协议，该协议定义在 Protocol.h 中。CS 与 SP 之间的连接是基于 TCP 的，包括以下几类消息：

（1）CS2SP_REGISTER，由 CS 注册到 SP；

（2）CS2SP_UPDATE，由 CS 发送更新到 SP；

（3）CS2SP_BLOCK，由 CS 发送媒体数据块到 SP；

（4）CS2SP_MEDIA_TYPE，由 CS 发送媒体类型到 SP；

（5）SP2CS_WELCOME，由 SP 通知 CS 从某个媒体数据块开始传送；

（6）SP2CS_MSG，由 SP 通知 CS 错误消息。

www.docin.com

11.3.2 Super Peer 子系统

SP 是服务器端的一个重要组成部分，它的功能是接收 CP 上传的视频数据，并向 NP 提供视频数据的下载。

1．系统结构

SP 在启动时需要向 Tracker 注册，并且同 Tracker 交互信息，该部分的功能由 TrackerUDPClient 类完成。

SP 接收 CS 的上传请求，该部分的功能由 CSTCPServer 类完成。

SP 与 NP、SP 的交互分为两部分。一部分是 SP 接受 NP、SP 的连接请求，该部分的功能由 P2PTCPServer 类完成，其协议定义在 enum SPandSP 中；另一部分是 SP 作为一个中间缓冲节点，在接收到 NP 的请求后，向目标 SP 发送请求连接，该部分的功能由 P2PTCPClient 类实现。

SP 对视频数据的管理功能由 ChannelMgr 类实现，它提供了大量的方法实现 SP 对视频数据的添加、获取、查询等操作。

2．类关系图

SP 子系统的 UML 静态结构如图 11-7 所示。

www.ckook.com

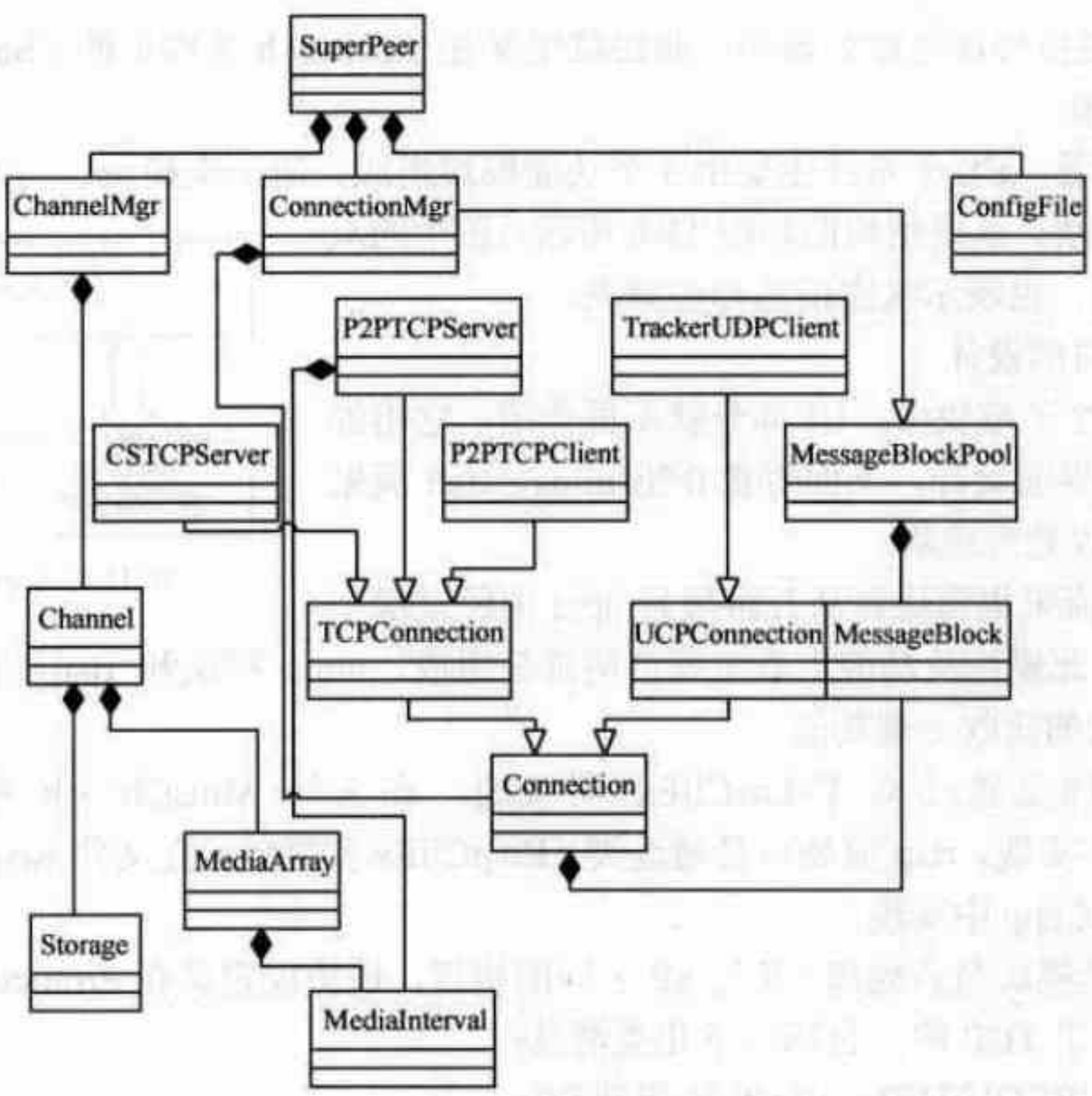


图 11-7　SuperPeer UML 静态结构图

11.3.3　Tracker 子系统

TS（Tracker Server）也是服务器端的一个重要组成部分，它的功能是接受 SP 提供注册的频道信息，接受 NP（Normal Peer）的注册信息，以及提供 NP 相关的信息查询。TS 动态地维护了 SP、NP 的状态和注册信息。

1．系统结构

图 11-8 示意了与 TS 通信的实体模块。TS 要处理 SP 与 TS、NP 与 TS 间的通信。TS 的设计就是围绕处理这两类通信进行的。

2．程序框架

由于 TS 需要与两个不同的实体进行通信，因此它被设计成为两个实体提供不同的通信链路。由于需要大规模的并发链接，这里选择 UDP 协议进行网络通信，以减轻系统的负载。

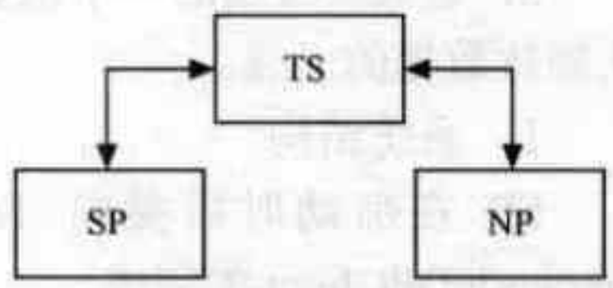


图 11-8　Tracker 通信示意图

TS 作为服务端，提供被动的服务，分别处理由 SP、NP 主动提交的信息，不需要主动向这些实体发送数据。为了方便地进行处理，TS 设计了两个模块分别处理不同实体提交的协议数据，包括接收、发送以及协议处理，每个模块都有自己的处理线程，保证处理的及时性和并发性。此外，TS 还维护了两个全局表，分别是 SP 信息对应的 chnl_map_和 NP 信息对应的 NPNode_map_。TS 两个模块的处理过程分别如下。

（1）SP 处理模块接收到 SP 发来的协议数据，将相应信息创建成 channelNode，并把 channelNode 插入、更新到 chnl_map_中。

（2）NP 处理模块接收到 NP 发来的协议数据，将相应信息插入、更新到 NPNode_map_

www.docin.com
www.ckook.com

中；对于 NP 的请求信息，NP 处理模块根据请求在两个全局表中查询相应的信息。

NP 与 Tracker 的协议通信如图 11-9 所示。

SP 与 Tracker 的协议通信如图 11-10 所示。

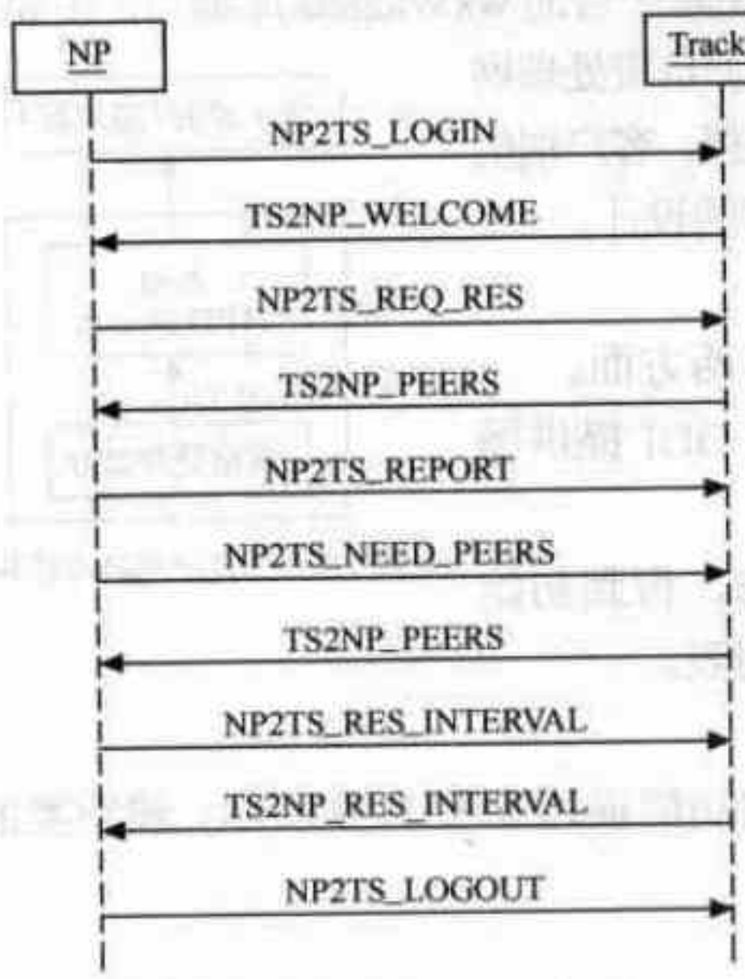


图 11-9 NP 与 Tracker 通信协议

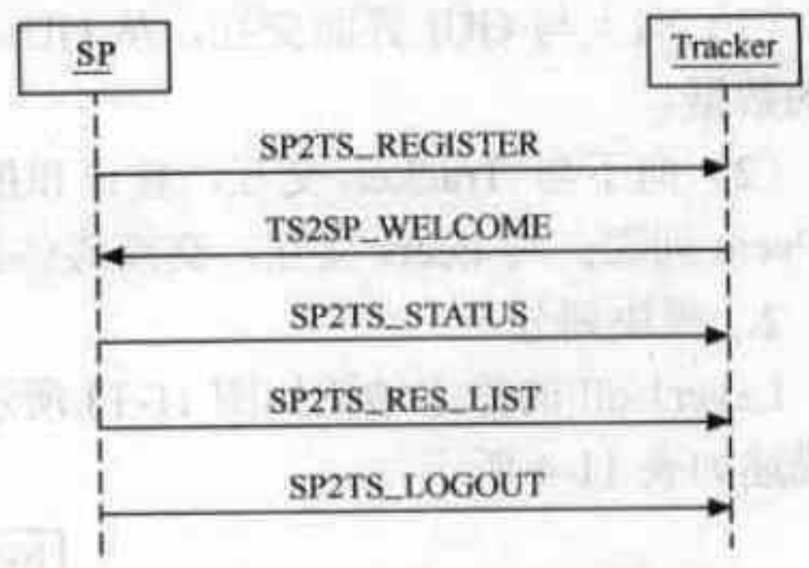


图 11-10 SP 与 Tracker 通信协议

3．类关系图

Tracker 的 UML 静态结构如图 11-11 所示。

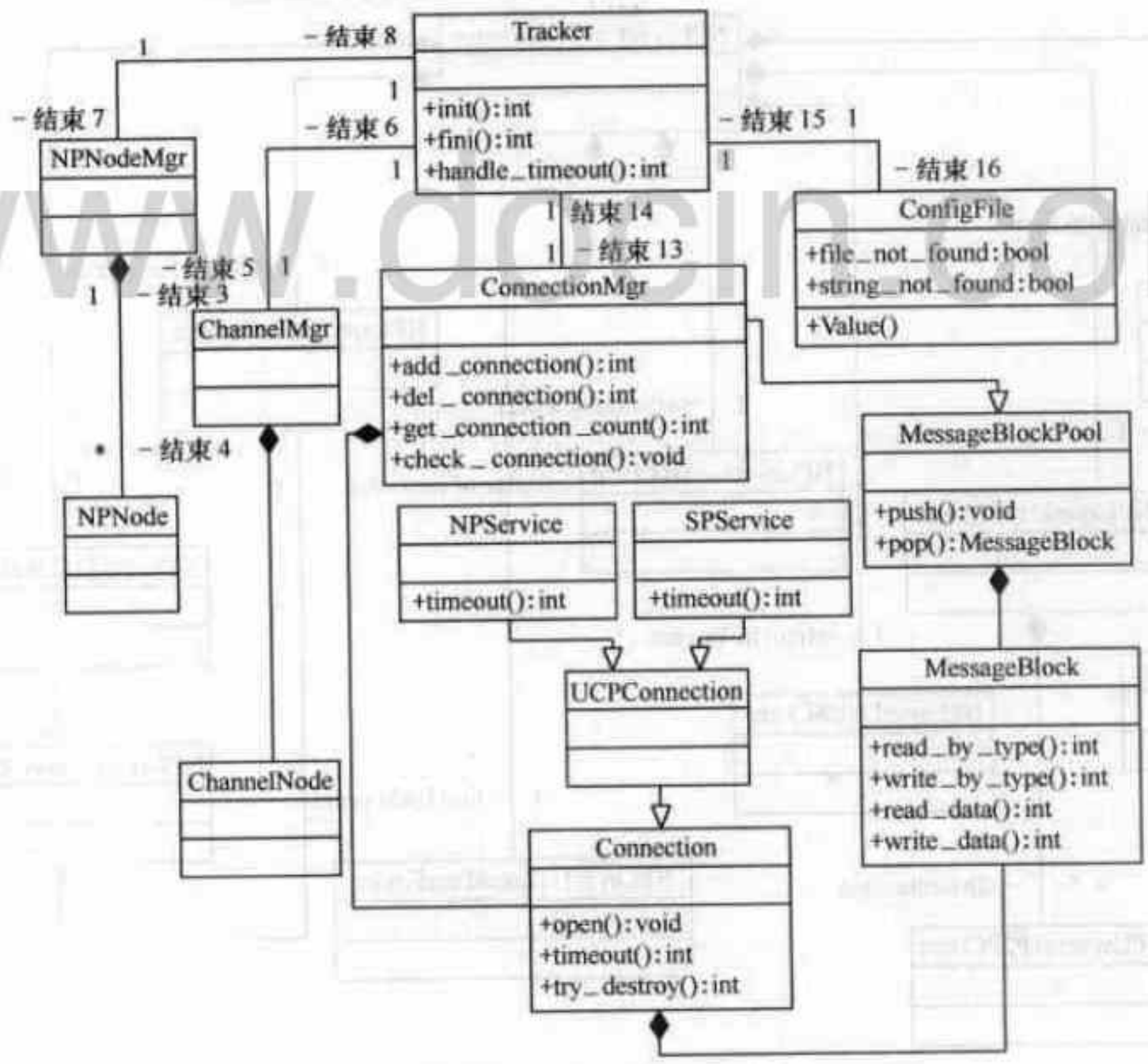


图 11-11 Tracker UML 图

www.ckook.com

11.3.4 Client 子系统

客户端由 GUI 界面和 Layer1.dll 库组成，GUI 界面采用跨平台的 wxWidgets 库编写，主要包括数据播放和播放器控制部分。Layer1.dll 库可分为两部分：①负责处理网络数据的部分；②本地 httpserver，负责向 GUI 发送播放数据。客户端的基本结构如图 11-12 所示。本小节将着重介绍 Layer1.dll 库的设计。

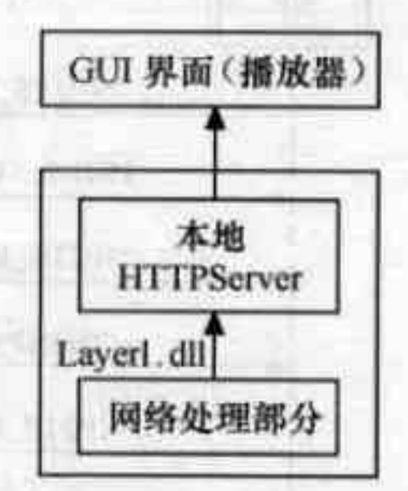


图 11-12 客户端基本结构图

1．功能描述

Layer1.dll 在 Myseelite 系统中承担的作用包括以下两方面。

（1）向上与 GUI 界面交互，从 GUI 获取指令，为 GUI 提供播放的数据。

（2）向下与 Tracker 交互，登记和报告自己的状态，得到初始的 Peers 列表；与 Peers 交互，交换媒体数据和 Peers 列表。

2．模块划分

Layer1.dll 的静态结构如图 11-13 所示，其主要类的功能描述如表 11-3 所示，辅助类的功能描述如表 11-4 所示。

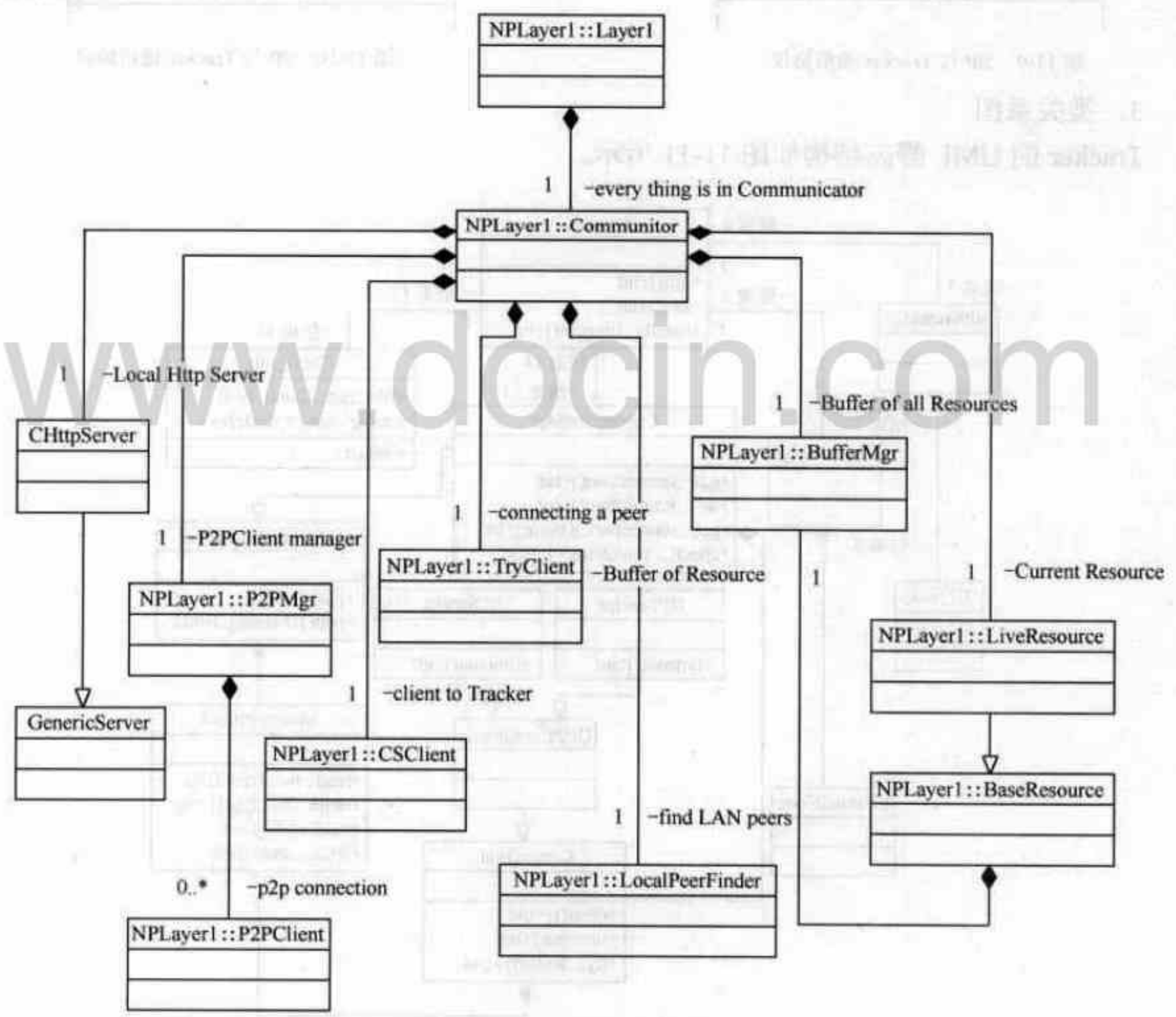


图 11-13 Layer1.dll 静态结构图

www.docin.com

www.ckook.com

表 11-3 Layer1.dll 主要类功能描述

类　名	功能定义
Layer1	对外的接口，可以看成是对 Communicator 的线程安全包装，Layer1 运行在调用者所在的线程，通过 Communicator::resEvent、Communicator::resAckEvent、Communicator::needControl3 个变量实现和主线程的同步
Communicator	控制类，LiveResource 、CSCClient、TryClient、P2PMgr 的容器，提供启动、停止服务的接口，对其他所有对象的访问都通过这个类
CSCClient	负责和 Tracker 通信
P2PClient	负责和 Peer 通信
TryClient	负责发起和 Peer 的连接
LiveResource	继承自 BaseResource，存放流媒体数据的 pool，保存流媒体的 metadata
P2PMgr	主线程所在的类，RunManager 是主线程方法，一个 for(;;) select 死循环
CHTTPServer	继承自 CGenericServer，负责将从网络获取的数据发送给 GUI 界面，进行播放

表 11-4 Layer1.dll 辅助类功能描述

类　名	功能定义
BufferMgr	一个内存池
BlockInterval /IntervalArray	用区间来描述一组 blockID 的序列，可以视作一种简单压缩
NormalAddress /P2PAddress	地址描述
CorePeerInfo /PeerInfoWithAddr	描述 Peer 的信息
TransferCalculator	辅助统计传输状态
TransferInfo	描述传输状态
PushList	描述待上传/下载的 block id 列表
LogMgr	日志系统

www.docin.com

在上述的模块中，Layer1 运行在调用者的线程中。P2PMgr::RunManager 运行在一个独立线程中，由它调度其他的模块。模块间的关系如图 11-14 所示。

3．运行流程说明

Layer1.dll 的序列图如图 11-15 所示，其具体流程如下：

（1）GUI 创建 Layer1 对象；

（2）GUI 调用 Layer1::Request，Layer1 调用 Communicator::Request，P2PMgr::RunManager 独立线程启动，CHttpServer::ClientThread 独立线程启动；

（3）ClientThread 调用 Layer1::GetXXX 获取各种数据；

（4）GUI 调用 Layer1::Stop，P2PMgr::RunManager 独立线程暂停活动。

www.ckook.com

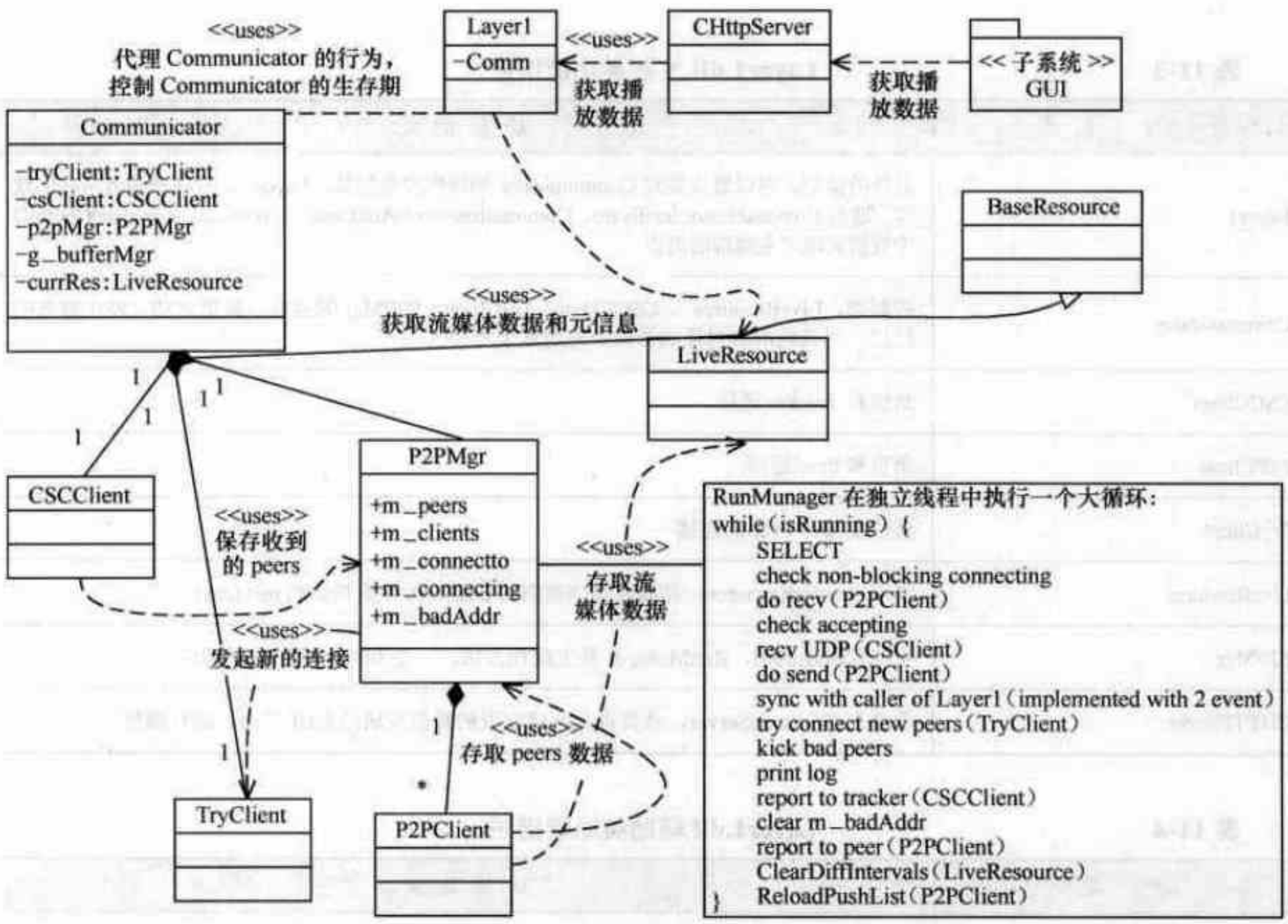


图 11-14　Layer1.dll 模块间的关系

4．部分模块内部说明

（1）P2PMgr

P2PMgr 是 Layer1.dll 的核心。它的代码分为以下两部分。

① P2PMgr::RunManage 大循环，如下所示。

```
while(isRunning) {
        SELECT
        check non-blocking connecting
        do recv ( P2PClient )
        check accepting
        recv UDP ( CSClient )                    // 当 pushlist 完成时重新调度
        do send ( P2PClient )
        sync with caller of Layer1 ( implemented with 2 event )
        try connect new peers ( TryClient )
        kick bad peers
        print log
        report to tracker ( CSCClient )
        clear m_badAddr
        report to peer ( P2PClient )
        ClearDiffIntervals ( LiveResource )
        ReloadPushList ( P2PClient )             // 定时强制重新调度
    }
```


www.ckook.com

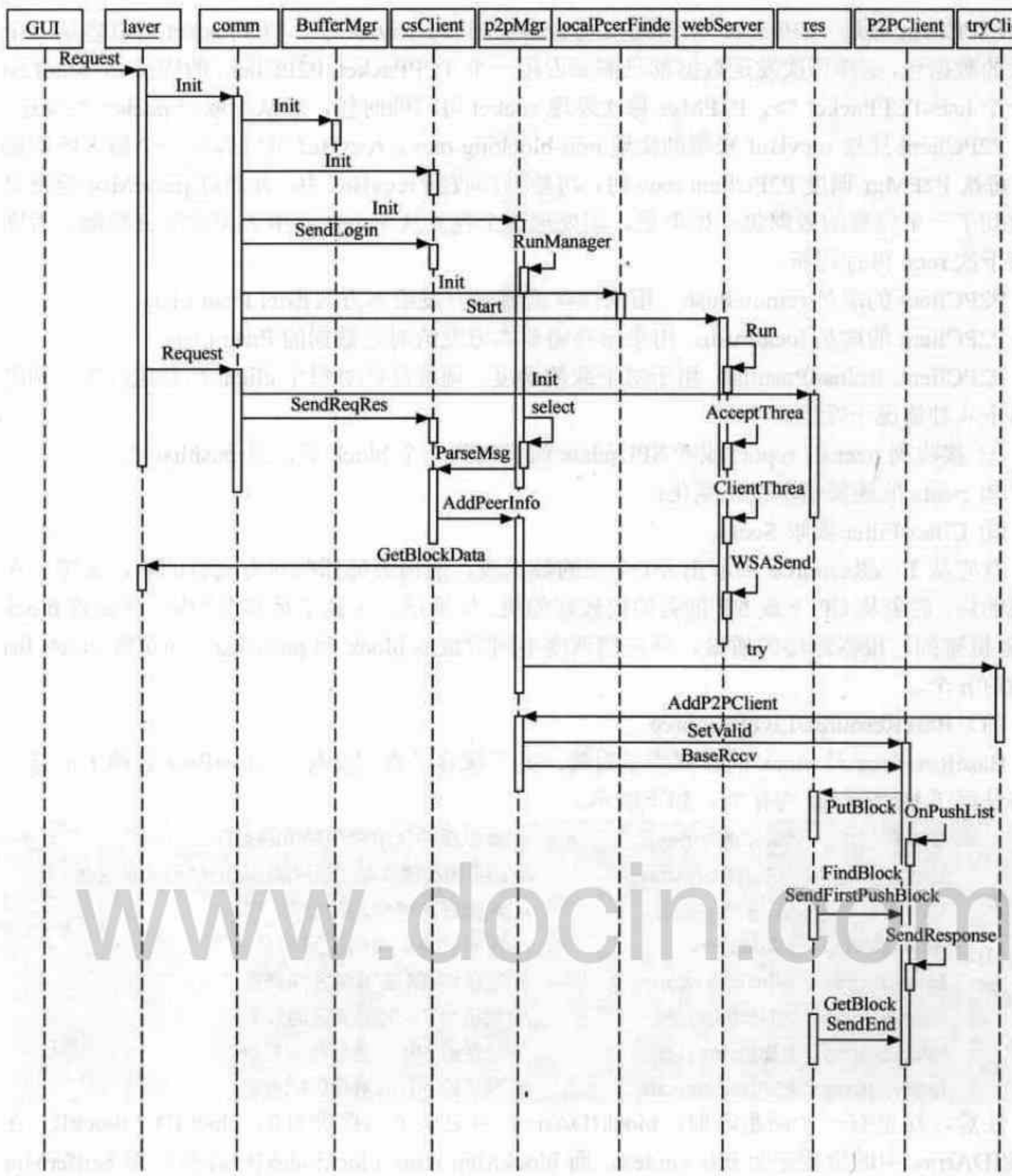


图 11-15 Layer1.dll 序列图

② 一组核心数据结构，以及围绕它们的管理函数，如下所示。

```
list<P2PClient>          m_clients;        // 所有连接的列表
list<PeerInfoWithAddr>   m_peers;          // 已知 Peer 列表
list<P2PAddress>         m_badAddr;        // 无法连接的地址
list<ConnectingPeer>     m_connectto;      // 对方要求本机主动连接
list<ConnectingPeer>     m_connecting;     // 正在连接的 Peer
```

（2）P2PClient

每个 P2PClient 对应一个和 peer 建立的 tcp 连接。

www.docin.com

P2PClient 通过 TCPPacket 来帮助实现 non-blocking send。每个 TCPPacket 可以容纳一个最大的数据包，这样每次发送数据都只需要占用一个 TCPPacket。P2PClient 的成员 m_sendList 是一个 list<TCPPacket *>，P2PMgr 每次发现 socket 可写的时候，就从中取出 packet 来发送。

P2PClient 通过 recvBuf 来帮助实现 non-blocking recv。recvBuf 可以容纳一个最大的数据包。每次 P2PMgr 调度 P2PClient recv 时，将数据接收到 recvBuf 中，并通过 parseMsg 检查是否收到了一个完整的数据包。如果是，则处理这个包并从 recvBuf 中去掉这部分数据，否则留待下次 recv 再行判断。

P2PClient 的成员 remotePush，用来保存需要对方发给本方数据的 Push List。

P2PClient 的成员 localPush，用来保存需要本方发给对方数据的 Push List。

P2PClient::ReloadPushlist，用于对下载的调度。调度是针对每个 client 单独进行的。调度在以下 4 种情况下发生。

① 接收到 peer 的 report 或者 SPUpdate 或者收到一个 block 后，且 pushlist 为空；

② peers 的连接情况发生变化；

③ ChaosFilter 要求 Seek。

首先从 LiveResource 中取出 90 个空的数据块，然后去掉那些对方没有的块，去掉正在下载的块。避免从 CP 下载 NP 拥有的比较好的块。如果已经下载了足够多的块，那么将 block list 随机排列。根据连接的质量，分三档调度不同数量的 block 到 push list。简单取 block list 中的前 n 个。

（3）BaseResource/LiveResource

BaseResource 对 block 的存储作了封装，并且保存了索引文件、SuperPeer 等种子信息。底层使用了 BufferMgr 内存池，如下所示。

```
UINT          *blockIDArray;      // 虚拟缓冲区中对应的 BlockID
UINT          *blockMapArray;     // 虚拟缓冲区与真正缓冲区(BufferMgr)的对应表
UINT          blockMapSize;       // 虚拟缓冲区中块的个数
IntervalArray allIntervals;       // 所有已下载块的区间列表
IntervalArray csIncIntervals;     // 发送给 TS 的增量区间列表
IntervalArray csDecIntervals;     // 发送给 TS 的增量区间列表
IntervalArray P2PIncIntervals;    // 发送给 NP 的增量区间列表
IntervalArray P2PDecIntervals;    // 发送给 NP 的增量区间列表
```

注意，这里有一个两重映射，blockIDArray 中记录了当前拥有的 blockID，blockID 在 blockIDArray 中的位置记为 blockindex，而 blockMapArray[blockindex]中存放的是 BufferMgr 分配的 block 页号。

LiveResource 相对于 BaseResource 增加了很多细节参数。影响调度、比较重要的有：

```
SPUpdate      spUpdate;           // SP 发送的广播
UINT          V_currBlockID;      // 当前视频块的编号
LONGLONG      V_lastSampleTime;   // 最近一个视频 Sample 的 start time
UINT          A_currBlockID;      // 当前音频块的编号
LONGLONG      A_lastSampleTime;   // 最近一个音频 Sample 的 start time
```

LiveResource 封装了 sample 和 block 之间的转换关系（GetSampe LoadSample）。相对于 BaseResource，它增加了一些安全检查（PutBlock）。

www.ckook.com

11.4 Myseelite 程序代码分析

本节将针对 Myseelite 系统中的客户端 P2P 管理模块的代码进行详细的代码分析和程序例举，希望可以帮助读者深入了解 Myseelite 的源代码。Myseelite 客户端 P2P 管理模块是由 P2PMgr 类实现的。在其头文件中，首先定义了 NP 和 SP 的连接数的枚举类型。**Myseelite 的源代码可以在 http://www.openmysee.com/modules/newbb/viewtopic.php?topic_id=118&forum=11 中找到。**

```
enum {
    // 每个 NP 上的连接数
    MAX_CONNECTION_PER_NP = 1,
    // 每个 SP 上的连接数
    MAX_CONNECTION_PER_SP = 1,
    // 所有和 SP 的连接数，这个限制不太严格，只在 CSClient::SendNeedPeers 中作了限制
    MAX_CONNECTION_OF_SP = 2,
};
```

此外，在类的声明中，还定义了该类所用到的定时器的枚举类型，该定时器都是以毫秒为单位。

```
enum {
    CS_REPORT=10000,            // 定时向 TrackServer 发送 NP 的状态
    CS_REQ_PEERS=15000,         // 定时向 TrackServer 请求 Peer 列表
    CS_REQ_INTERVAL=10000,      // 定时向 TrackServer 请求 SP BlockInterval
    P2P_CONNECT=5000,           // 发出连接的超时时间
    P2P_SEND_REPORT=500,        // 定时向连接中的对方发送自身消息
    P2P_MANAGE=5000,            // 定时检查所有连接，删除超时的连接，并打印传输等状况
    P2P_NEAR_PEERS=60000,       // 定时向连接中节点发送其可能需要的节点信息
    P2P_CLEAR_BADADDR=180000,
                                // 定时清空无法连接的地址列表，因为有些地址可能值得再次尝试
    TRY_CONNECT=2000,           // 定时检查有没有可以连接的 Peer
    P2P_CHECK_PUSHLIST=500,     // 定时检查所有连接的 PushList，如果为空，则重新分配
    LPF_BROADCAST = 10000,      // 定时向局域网广播自己的信息
};
```

P2PMgr 类主要包括以下几个成员变量。

```
SOCKET m_Socket;    // 当前监听的 Socket
BOOL isRunning;     // 是否正在运行
HANDLE mgrThread;   // 主线程的句柄

list<P2PClient>          m_clients;      // 所有连接的列表
list<PeerInfoWithAddr>   m_peers;        // 已知 Peer 列表
list<P2PAddress>         m_badAddr;      // 无法连接的地址
list<ConnectingPeer>     m_connectto;    // 对方要求本机主动连接
list<ConnectingPeer>     m_connecting;   // 正在连接的 Peer
```

www.ckook.com

```
FreeList<TCPPacket> m_freeList;                    // TCPPacket 动态分配/释放的列表
LONGLONG m_totalBlockDataUp, m_totalBlockDataDown;
                                                   // 上传和下载的数据大小（只包括 BLOCK 数据部分）
LONGLONG m_totalBytesFromSP;                       // 从 SP 下载的数据
UINT m_connectFailCount;                           // 连接失败的次数
UINT m_lowversionCount;                            // 低版本客户端的连接次数
UINT m_incomingCount;                              // 连入连接成功的次数
UINT m_outgoingCount;                              // 连出连接成功的次数
UINT m_incomingElapsedTime;                        // 所有连入连接存在的总时间
UINT m_outgoingElapsedTime;                        // 所有连出连接存在的总时间

CorePeerInfo lastSent;                             // 最近一次广播的 CorePeerInfo
bool m_bBlockIntervalHasChanged;                   // 最近一次广播之后，块区间发生了变化
Communicator* comm;                                // 控制容器类对象的指针
```

在 P2PMgr 的文本文件中，主要实现了以下几个公有的成员函数，供外部调用。

```
// 构造函数，传递参数是控制容器类对象的指针
P2PMgr::P2PMgr(Communicator* c) : comm(c), m_freeList(8, 1) {
    m_Socket = INVALID_SOCKET;
    isRunning = FALSE;
    mgrThread = NULL;
    m_totalBlockDataUp = m_totalBlockDataDown = 0;
    m_totalBytesFromSP = 0;
    m_connectFailCount = m_lowversionCount = 0;
    m_incomingCount = m_outgoingCount = 0;
    m_incomingElapsedTime = m_outgoingElapsedTime = 0;
    m_bBlockIntervalHasChanged = true;
}

// 析构函数
P2PMgr::~P2PMgr() {
    Uninit();
    comm->logFile.StatusOut("P2PMgr: exited...");
}

// 开始运行
P2P_RETURN_TYPE P2PMgr::Init() {
    if(!isRunning) {
        isRunning = TRUE;

        // 初始化足够的 P2PClient
        m_clients.resize(comm->cfgData.GetSum());
```


www.ckook.com

```
            P2P_RETURN_TYPE ret = Binding();
            if(ret < PRT_OK)
                return ret;

            comm->logFile.StatusOut("P2PMgr Thread %d Begin, this=%p, comm=%p, comm->P2P
Mgr=%p", ::GetCurrentThreadId(), this, comm, &comm->P2PMgr);
            DWORD threadID;
            mgrThread = CreateThread(
                NULL, 0, (LPTHREAD_START_ROUTINE)(P2PMgr::RunManager), this, 0,
&threadID);//////vivi////启动连接管理线程 RunManager
            if(mgrThread == NULL)
                return PRT_SYS;
            // 设置较低的优先级
            SetThreadPriority(mgrThread, THREAD_PRIORITY_BELOW_NORMAL);
        }
        return PRT_OK;
    }

    // 停止运行
    void P2PMgr::Uninit() {
        if(isRunning) {
            isRunning = FALSE;
            SafeCloseSocket(m_Socket);

            if(mgrThread) {
                // 如果一段时间内不能正常停止，就强行终止线程
                comm->logFile.StatusOut("P2PMgr: Wait for Thread exit...");
                DWORD ret = WaitForSingleObject(mgrThread, 5000);
                if(ret == WAIT_TIMEOUT) {
                    comm->logFile.StatusOut("P2PMgr: Wait for Thread exit...Failed! Terminate it!");
                    TerminateThread(mgrThread, 0);
                }
                CloseHandle(mgrThread);
                mgrThread = NULL;
            }

            comm->logFile.StatusOut("P2PMgr: Clear P2PClient list...");
            for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
                (*it).SetInvalid();
            }
            m_clients.clear();
```

www.docin.com

```
        comm->logFile.StatusOut("P2PMgr: Clear other lists...");
        // 清空正在连接的列表
        m_connecting.clear();
        // 清空已知 Peer 的列表
        m_peers.clear();
        // 清空无法连接的地址列表
        m_badAddr.clear();
        // 清空需要连接的列表
        m_connectto.clear();
    }
}

// 安全关闭 Socket，会启动 RunCloseSocket 这个线程
void P2PMgr::SafeCloseSocket(SOCKET sock) {
    SOCKET* nSock = new SOCKET;
    HANDLE closeThead = NULL;
    if(nSock) {
        *nSock = sock;
        DWORD threadID;
        closeThead = CreateThread(
            NULL, 0, (LPTHREAD_START_ROUTINE)(P2PMgr::RunCloseSocket), nSock, 0,
&threadID);
    }
    if(closeThead == NULL || nSock == NULL) {
        // 如果不能启动一个线程，就只好慢慢 Close
        assert(0);
        TE_CloseSocket(sock, FALSE);
    }
    else if(closeThead){
        // 设置较低的优先级
        SetThreadPriority(closeThead, THREAD_PRIORITY_BELOW_NORMAL);
        // 这个线程启动之后我们将不再关心它
        CloseHandle(closeThead);
    }
}

// 停止所有连接
void P2PMgr::StopAll() {
    // 断开所有连接
    for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
        if(!it->IsValid())
```

www.dooin.com

www.ckook.com

```
                continue;
            it->SetInvalid();
        }
        // 清空正在连接的列表
        for(CPeerIt itt = m_connecting.begin(); itt != m_connecting.end(); ++itt)
            SafeCloseSocket(itt->sock);
        m_connecting.clear();
        // 清空已知 Peer 的列表
        m_peers.clear();
        // 清空无法连接的地址列表
        m_badAddr.clear();
        // 清空需要连接的列表
        m_connectto.clear();
    }

    // 清除所有连接发送媒体类型区间的记录
    void P2PMgr::ClearSentMediaArray() {
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid())
                continue;
            it->ClearSentMediaArray();
        }
    }

    // 添加正在连接的连接
    void P2PMgr::AddConnecting(const ConnectingPeer& peer) {
        // 正在连接中
        m_connecting.push_back(peer);
    }

    // 添加连接成功的连接
    void P2PMgr::AddP2PClient(const ConnectingPeer& peer) {
        // 已经连接上，检查此种类型连接是否已满
        bool bConnectionFull = IsConnectionFull(peer.isIncoming, peer.isForFree, false);
        if(bConnectionFull) {
            comm->logFile.StatusOut("connection is full, abandon connection.");
            SafeCloseSocket(peer.sock); // no force disconnect
            return;
        }

        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid()) {
                if(it->SetValid(comm, peer))
```

www.ckook.com

```
                    break;
                else
                    it->SetInvalid();
            }
        }
        if(it == m_clients.end()) {
            assert(0);
            comm->logFile.StatusOut("Shouldn't be here!");
            SafeCloseSocket(peer.sock); // no force disconnect
        }
    }

    // 添加需要连接的地址
    void P2PMgr::AddConnectTo(const P2PAddress& addr, bool connectForFree) {
        if(addr == comm->localAddress || addr.outerIP.sin_addr.s_addr == 0)
            return;
        ConnectingPeer peer;
        memcpy((P2PAddress*)&peer, &addr, sizeof(P2PAddress));

        peer.isForFree = connectForFree;
        peer.isSuperPeer = false;

        PeerAddrIt it = find(m_peers.begin(), m_peers.end(), peer);
        if(it != m_peers.end()) // found in known list
            return;
        list<P2PAddress>::const_iterator it1;
        it1 = find(m_badAddr.begin(), m_badAddr.end(), addr);
        if(it1 != m_badAddr.end()) // found in bad address list
            return;
        list<ConnectingPeer>::const_iterator it2;
        it2 = find(m_connectto.begin(), m_connectto.end(), peer);
        if(it2 != m_connectto.end()) // found in connectto list
            return;

        m_connectto.push_back(peer);
    }

    // 添加 Peer 信息
    void P2PMgr::AddPeerInfo(PeerInfoWithAddr& peer) {
        if(peer == comm->localAddress || peer.outerIP.sin_addr.s_addr == 0)
            return;

        PeerAddrIt it = find(m_peers.begin(), m_peers.end(), peer);
```

www.dbcin.com

www.ckook.com

```
        if(it != m_peers.end()) { // found in known list
            memcpy(&*it, &peer, sizeof(PeerInfoWithAddr)); // replace old
            return;
        }
        list<P2PAddress>::const_iterator it1;
        it1 = find(m_badAddr.begin(), m_badAddr.end(), peer);
        if(it1 != m_badAddr.end()) // found in bad address list
            return;
        list<ConnectingPeer>::const_iterator it2;
        for(it2 = m_connectto.begin(); it2 != m_connectto.end(); ++it2) {
            if((PeerInfoWithAddr)*it2 == peer) // found in connectto list
                return;
        }

        m_peers.push_back(peer);
    }

    // 获取需要连接的 Peer
    BOOL P2PMgr::GetPeer4Connect(ConnectingPeer& peer) {
        // 当前资源尚未下载完毕
        if(comm->currRes) {
            // 1. 选择一个地址主动连接
            for(PeerAddrIt it = m_peers.begin(); it != m_peers.end(); ++it) {
                // 判断连接是否已满
                if(!IsConnectionFull(false, false, true)) {
                    // 复制 Peer 地址
                    memcpy((PeerInfoWithAddr*)&peer, &*it, sizeof(PeerInfoWithAddr));
                    m_peers.erase(it);
                    // 一个地址只能有一个非 Free 的连接和一个 Free 的连接(SP 的不定)
                    if(CheckN4One(peer, false, false, peer.isSuperPeer ? MAX_CONNECTION_PER_SP :
MAX_CONNECTION_PER_NP)) {
                        peer.isForFree = peer.isIncoming = peer.isPassive = false;
                        return TRUE;
                    }
                    else {
                        it = m_peers.begin();
                        continue;
                    }
                }
            }
        }

        // 2. 如果没有主动去连接的 Peer，则连接需要被动去连接的 Peer
```

www.ckook.com

```
        for(CPeerIt it = m_connectto.begin(); it != m_connectto.end(); ++it) {
            // 判断连接是否已满
            if(!IsConnectionFull(true, it->isForFree, true)) {
                // 复制 Peer 地址
                memcpy((ConnectingPeer*)&peer, &*it, sizeof(ConnectingPeer));
                m_connectto.erase(it);
                // 判断此地址是否已经有连接
                if(CheckN4One(peer, peer.isForFree, false, MAX_CONNECTION_PER_NP)) {
                    peer.isIncoming = true;          // 算作连入连接，因为是对方需要己方的数据
                    peer.isSuperPeer = false;        // 非 SP
                    peer.isPassive = true;           // 被动连接
                    peer.layer = 0xff;               // 对方的 layer 未知
                    peer.isMaxIn = peer.isMaxOut = peer.isMaxFreeIn = peer.isMaxFreeOut = false;
                    return TRUE;
                }
                else {
                    it = m_connectto.begin();
                    continue;
                }
            }
        }

        // 没有任何可以连接的地址
        return FALSE;
    }

    // 添加连接失败的地址
    void P2PMgr::AddBadAddr(P2PAddress addr) {
        list<P2PAddress>::const_iterator cit =
            find(m_badAddr.begin(), m_badAddr.end(), addr);
        if(cit == m_badAddr.end()) {
            // 如果 BadAddr 的列表持续增加，在特殊情况下可能会占用很多内存
            // 所以保持在少于 100 个比较好
            if(m_badAddr.size() > 100)
                m_badAddr.pop_front();
            m_badAddr.push_back(addr);
        }
    }

    // 确认一个地址不超过 N 个连接
    BOOL P2PMgr::CheckN4One(P2PAddress addr, bool isForFree, bool isConnected, UINT max) {
        // 连接计数
        UINT count = 0;
```

www.docin.com

www.ckook.com

```
// 如果已经连上，要先加上此连接自身
if(isConnected)
        ++count;
// 查找来自同一地址的连接，如果超过 max 值，就返回 false
for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
        if(!it->IsValid())
                continue;
        if(it->GetAddress() == addr) {
                if(isForFree == it->GetIsForFree()) {
                        count++;
                        if(count == max)
                                return FALSE;
                }
        }
}
// 如果是已经连上的连接，显然优先级比正在连接的高，所以不予计算
if(!isConnected) {
        // 在正在连接的地址列表中查找同一地址，如果超过 max 值，就返回 false
        for(CPeerIt itt = m_connecting.begin(); itt != m_connecting.end(); ++itt) {
                ConnectingPeer* peer = &*itt;
                if(*peer == addr) {
                        if(isForFree == itt->isForFree) {
                                count++;
                                if(count == max)
                                        return FALSE;
                        }
                }
        }
    }
    return TRUE;
}

// 取得某个已连接 Peer 有用的 Peer 列表
void P2PMgr::GetPeersForNeighbour(list<PeerInfoWithAddr> &array, UINT wantNum, P2PAddress&
addr) {
        for(ClientIt it = m_clients.begin(); it != m_clients.end() && array.size() < wantNum; ++it) {
                if(!it->IsValid())
                        continue;
                if(it->GetIsSP()) // 不用告诉别人 SP 的地址
                        continue;
                if(!IsConnectablePeer(addr, it->GetAddress())) // not connectable
                        continue;
                array.push_back(it->GetPeerInfo());
```

www.docin.com

```
        }

        PeerAddrIt it1 = m_peers.begin();
        for(; it1 != m_peers.end() && array.size() < wantNum; ++it1) {
            if(!IsConnectablePeer(addr, *it1))
                continue;
            array.push_back(*it1);
        }

        for(CPeerIt it2 = m_connecting.begin(); it2 != m_connecting.end() && array.size() < wantNum;
++it2) {
            if(!IsConnectablePeer(addr, *it2))
                continue;
            array.push_back(*it2);
        }
    }

    // 取得本机的层数，就是最小的层
    UINT8 P2PMgr::GetSelfLayer() {
        if(GetSPConCount() > 0)
            return 1;       // 连接了 SP，肯定是第一层
        UINT8 layer = 0xff;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid())
                continue;
            // 必须和本机是同一个资源才有价值
            if(!it->GetIsSameRes())
                continue;
            if(it->GetLayer() == 0xff) // 尚未收到 P2P_HELLO 消息
                continue;
            if(it->GetLayer() <= 1)
                assert(0);
            layer = min(layer, it->GetLayer());
        }

        return layer;
    }

    // 取得本 Peer 信息
    void P2PMgr::GetSelfInfo(CorePeerInfo& thisPeer) {
        thisPeer.isSuperPeer = false;
        thisPeer.layer = GetSelfLayer();
        thisPeer.isMaxIn = (GetIncomingCount() >= comm->cfgData.INCOMING_NUM);
```

www.docin.com

www.ckook.com

```
        thisPeer.isMaxOut = (GetOutgoingCount() >= comm->cfgData.OUTGOING_NUM);
        thisPeer.isMaxFreeIn = (GetFreeInCount() >= comm->cfgData.IN_4FREE_NUM);
        thisPeer.isMaxFreeOut = (GetFreeOutCount() >= comm->cfgData.OUT_4FREE_NUM);
}

// 以下几个函数是关于连接的统计
UINT16 P2PMgr::GetLowVersionCount() {
        if(m_lowversionCount >= 0xffff)
                return 0xffff;
        return static_cast<UINT16>(m_lowversionCount);
}

UINT16 P2PMgr::GetConnectFailCount() {
        if(m_connectFailCount >= 0xffff)
                return 0xffff;
        return static_cast<UINT16>(m_connectFailCount);
}

UINT16 P2PMgr::GetTotalIncomingCount() {
        if(m_incomingCount >= 0xffff)
                return 0xffff;
        return static_cast<UINT16>(m_incomingCount);
}

UINT16 P2PMgr::GetTotalOutgoingCount() {
        if(m_outgoingCount >= 0xffff)
                return 0xffff;
        return static_cast<UINT16>(m_outgoingCount);
}

UINT16 P2PMgr::GetAvgIncomingTime() {
        UINT totalTime = m_incomingElapsedTime;
        UINT count = m_incomingCount;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
                if(!it->IsValid())
                        continue;
                if(it->GetIsIncoming()) {
                        count++;
                        totalTime += it->GetElapsedTime();
                }
        }
        if(count == 0)
                return 0;
```

www.docin.com

```
        UINT avgTime = totalTime/count;
        if(avgTime/1000 >= 0xffff)
            return 0xffff;
        return static_cast<UINT16>(avgTime/1000);
    }

    UINT16 P2PMgr::GetAvgOutgoingTime() {
        UINT totalTime = m_outgoingElapsedTime;
        UINT count = m_outgoingCount;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid())
                continue;
            if(!it->GetIsIncoming()) {
                count++;
                totalTime += it->GetElapsedTime();
            }
        }
        if(count == 0)
            return 0;
        UINT avgTime = totalTime/count;
        if(avgTime/1000 >= 0xffff)
            return 0xffff;
        return static_cast<UINT16>(avgTime/1000);
    }

    void P2PMgr::ConnectionEstablished(bool isIncoming) {
        if(isIncoming)
            m_incomingCount++;
        else
            m_outgoingCount++;
    }

    void P2PMgr::ConnectionClosed(bool isIncoming, DWORD elapsedTime) {
        if(isIncoming) {
            m_incomingElapsedTime += elapsedTime;
        }
        else {
            m_outgoingElapsedTime += elapsedTime;
        }
    }

    float P2PMgr::GetMessagePercent() {
        // 计算 P2P 网络中数据长度与控制信息长度的比例，以不超过 5%为佳
```


www.ckook.com

```
        TransferInfo tiTemp;
        GetTransferInfo(tiTemp);

        LONGLONG allMsg = 0;
        LONGLONG allData = 0;
        allMsg += tiTemp.totalDownBytes + tiTemp.totalUpBytes - m_totalBlockDataUp -
m_totalBlockDataDown;
        allData += tiTemp.totalDownBytes + tiTemp.totalUpBytes;
        double a = (double)allMsg/(1024);
        double b = (double)allData/(1024);
        double msgPercent = (allData==0?0:(((double)allMsg/allData)*100));
        comm->logFile.StatusOut("Msg Percent: %f%(%fKB/%fKB), Share Percent: %f%(%.2f/%.2fMB),
Data From SP: %.2fMB", msgPercent, a, b,
            (tiTemp.totalUpBytes*100.f)/tiTemp.totalDownBytes,
            tiTemp.totalUpBytes/1024.f/1024.f, tiTemp.totalDownBytes/1024.f/1024.f,
            m_totalBytesFromSP/1024.f/1024.f);
        return static_cast<float>(msgPercent);
    }

    // 重新分配所有连接的 Push List.
    // 如果 Client 非 NULL，则只分配此连接的块；否则，分配所有连接的块
    // 这个开销(网络流量)其实比较大，还需要商榷，作为暂时的解决办法是没问题的
    void P2PMgr::RedistributeAllBlocks(P2PClient* client) {
        assert(comm->currRes);
        if(!comm->currRes)
            return;

        if(client) {
            // 如果是普通下载方式，只给此连接分配少量的块
            client->ReloadPushList();
        }
        else {
            for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
                if(!it->IsValid() || client == &*it)
                    continue;
                it->ReloadPushList();
            }
        }
    }

    // 在除了 Client 以外的所有连接的 PushList 中查找 blockID
    bool P2PMgr::FindInAllPushList(const UINT blockID, P2PClient* client) {
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
```

www.docin.com

```
            if(!it->IsValid() || client == &*it)
                continue;
            if(it->FindPushBlock(blockID))
                return true;
        }
        return false;
    }

    // 在除了 Client 以外的所有连接(是否包含 SP)的 remoteInterval 中查找 blockID
    UINT8 P2PMgr::FindInAllRemoteInterval(const blockID, P2PClient* client, bool includeSP, float
minSpeed) {
        UINT8 count = 0;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid() || client == &*it)
                continue;
            if(!includeSP && it->GetIsSP())
                continue;
            if(it->GetCurDownSpeed()/1024 < minSpeed)
                continue;
            if(it->FindRemoteBlock(blockID))
                ++count;
        }
        return count;
    }

    // 在除了 Client 以外的所有 LAN 连接的 remoteInterval 中查找 blockID
    UINT8 P2PMgr::FindInAllLanRemoteInterval(const blockID, P2PClient* client, float minSpeed) {
        UINT8 count = 0;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            if(!it->IsValid() || client == &*it)
                continue;
            if(!it->GetIsSameLan())
                continue;
            if(it->GetCurDownSpeed()/1024 < minSpeed)
                continue;
            if(it->FindRemoteBlock(blockID))
                ++count;
        }
        return count;
    }

    // 以下两个函数的功能是打印出 IP 地址
    LPCSTR P2PMgr::FormatIPAddress(P2PAddress& addr) {
```


www.ckook.com

```
        strcpy(TempIP, inet_ntoa(addr.outerIP.sin_addr));
        if(addr.subnetIP.sin_addr.s_addr != UINT_MAX) {
            strcat(TempIP, "->");
            strcat(TempIP, inet_ntoa(addr.subnetIP.sin_addr));
        }
        strcat(TempIP, ":");
        sprintf(TempIP+strlen(TempIP), "%d", ntohs(addr.subnetIP.sin_port));
        return TempIP;
    }

    LPCSTR P2PMgr::FormatIPAddress(NormalAddress& addr) {
        strcpy(TempIP, inet_ntoa(addr.sin_addr));
        strcat(TempIP, ":");
        sprintf(TempIP+strlen(TempIP), "%d", ntohs(addr.sin_port));
        return TempIP;
    }
```

此外，该文本文件还实现了以下几个私有成员函数。

```
    // 主线程的回调函数
    void __stdcall P2PMgr::RunManager(P2PMgr* mgr) {
        timeval timeout;
        timeout.tv_sec = 0;
        timeout.tv_usec = 50000;

        fd_set read_set, write_set;

        sockaddr_in tmpAddr;
        memset(&tmpAddr, 0, sizeof(tmpAddr));
        int addrlen = sizeof(tmpAddr);

        bool nullLastTime = true;
        bool bFirstTime = true;

        ConnectingPeer incomingPeer;

        DWORD lastManageTime=0, lastCSReportTime=0,
            lastP2PReportTime=0, lastP2PNearPeerTime=0,
            lastTryConnectTime=0, lastClearBadAddress=0,
            lastCheckPushListTime=0, lastCSNeedPeerTime=2,
        lastCSReqIntervalTime=3,
            lastBroadcastInfoTime=0, currTime=0;

        // 每隔一段比较长的时间，比如 20 次 report 之后，就发送一次更新全部区间列表的消息
        UINT csReportCount=0, P2PReportCount=0;
```

www.docin.com

```
        // 设置这个时间意味着，第一次 While 循环时，csClient.SendLogin 不会被执行
        // 为什么要这样做？因为 SendLogin 已经在 Communicator 初始化之际被执行了
        // 不必在这么短的时间内连续发送两个 Login，徒然增加 TS 负担而已
        lastCSReportTime = timeGetTime();

        mgr->comm->logFile.StatusOut("P2PMgr Thread %d Started, mgr=%p, mgr->comm=%p,
mgr->comm->P2PMgr=%p", ::GetCurrentThreadId(), mgr, mgr->comm, &mgr->comm->P2PMgr);

        while(mgr->isRunning) {
            CriticalSection::Owner lock(mgr->comm->mainThreadLock);

            FD_ZERO(&read_set);
            FD_SET(mgr->m_Socket, &read_set);
            FD_SET(mgr->comm->csClient.GetSocket(), &read_set);
            if (mgr->comm->localPeerFinder.IsRunning()) {
                FD_SET(mgr->comm->localPeerFinder.GetSocket(), &read_set);
            }
            for(ClientIt it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                if(!it->IsValid())
                    continue;
                FD_SET(it->GetSocket(), &read_set);
            }

            FD_ZERO(&write_set);
            for(CPeerIt itt = mgr->m_connecting.begin(); itt != mgr->m_connecting.end(); ++itt)
                FD_SET(itt->sock, &write_set);

            int s = select(0, &read_set, &write_set, NULL, &timeout);
            if(!mgr->isRunning)
                break;
            if (s > 0) {
                // 检查正在连接的 Socket，有没有连接成功的
                for(itt = mgr->m_connecting.begin(); itt != mgr->m_connecting.end(); ++itt) {
                    if((FD_ISSET(itt->sock, &write_set))) {
                        mgr->AddP2PClient(*itt);
                        mgr->m_connecting.erase(itt);

                        // 回到开始，再找一遍，直到都是尚未连接成功的
                        itt = mgr->m_connecting.begin();
                        continue;
                    }
                }
                // 检查所有连接有没有收到数据
                for(it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                    if(!it->IsValid())
                        continue;
                    if((FD_ISSET(it->GetSocket(), &read_set))) {
```

www.docin.com

www.ckook.com

```
                if(!it->BaseRecv()) {
                        // 关闭该连接，并清空该连接要发送的数据包
                        it->SetInvalid();
                }
                break;
            }
        }
        // 检查有没有新的连入连接
        if(FD_ISSET(mgr->m_Socket, &read_set)) {
            SOCKET sock = accept(
                mgr->m_Socket, (struct sockaddr *)&tmpAddr, &addrlen);

            if(sock == INVALID_SOCKET) {
                mgr->comm->logFile.StatusErr("accepting socket", WSAGetLastError());
                mgr->SafeCloseSocket(sock); // no force disconnect
            }
            else {
                // Set this socket as a Non-blocking socket.
                ULONG flag = 1;
                if (ioctlsocket(sock, FIONBIO, &flag) == -1) {
                    mgr->comm->logFile.StatusErr("Setting accepted socket as
non-blocking", WSAGetLastError());
                }
                else {
                    incomingPeer.sock = sock;
                    incomingPeer.isForFree = false;
                    incomingPeer.isSuperPeer = false;
                    incomingPeer.isIncoming = true;
                    incomingPeer.isPassive = false;
                    incomingPeer.outerIP.sin_addr.s_addr = tmpAddr.sin_addr.s_addr;
                    incomingPeer.isSameLan = false;
                    mgr->AddP2PClient(incomingPeer);
                }
            }
        }
        // 检查 CSClient 有没有收到 UDP 包
        if(FD_ISSET(mgr->comm->csClient.GetSocket(), &read_set)) {
            if(!mgr->comm->csClient.ParseMsg(tmpAddr, addrlen)) {
                ;
            }
        }

        // 检查 LocalPeerFinder 有没有收到广播
        if (mgr->comm->localPeerFinder.IsRunning() &&
            FD_ISSET(mgr->comm->localPeerFinder.GetSocket(), &read_set)) {
            mgr->comm->localPeerFinder.ReadBroadcast();
        }
    }
```

www.ckook.com

```
else if(s == SOCKET_ERROR) {
        mgr->comm->logFile.StatusErr("selecting", WSAGetLastError());
        Sleep(1); // prevent dead loop
}

// 从所有连接的发送队列中提取数据发送
mgr->SendPackets();

// 获取当前时间
currTime = timeGetTime();

// 如果出现多个超时的 socket，也只删除第一个，因为这里的循环很快，不用担心删不掉
for(itt = mgr->m_connecting.begin(); itt != mgr->m_connecting.end(); ++itt) {
        if(currTime - itt->connectTime >= P2P_CONNECT) {
                mgr->m_connectFailCount++;
                mgr->comm->logFile.StatusOut("Connecting socket %s is Timeout!",
mgr->FormatIPAddress(*itt));
                // 超时！关闭 Socket，从连接列表中删除，并且添加到 badAddr 列表
                mgr->SafeCloseSocket(itt->sock); // no force disconnect
                mgr->AddBadAddr(*itt);

                mgr->m_connecting.erase(itt);
                break;
        }
}

// 每隔一段时间尝试连接其他 Peer
if(currTime-lastTryConnectTime >= TRY_CONNECT) {
        lastTryConnectTime = currTime;
        mgr->comm->tryClient.Try();
}

if(currTime-lastManageTime >= P2P_MANAGE) {
        lastManageTime = currTime;
        // 删除超时的连接
        //mgr->comm->logFile.StatusOut("processing kill idle connections....");
        for(it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                if(it->IsValid() && it->IsIdleTooLong()) {
                        mgr->AddBadAddr(it->GetAddress());
                        it->SetInvalid();
                }
        }

        // 生成传输信息并打印
        mgr->GenerateTransferInfo(TRUE);
        TransferInfo tiTemp;
        mgr->GetTransferInfo(tiTemp);
        mgr->comm->logFile.StatusOut("Cur: (%.2f/%.2f)KB/s. Avg: (%.2f/%.2f)KB/s. Total:
```


www.ckook.com

```
%.2f/%.2fMB. Layer: %d.\n\tConnect: low version %d, Fail %d, In %d, Out %d. AvgTime: In %d, Out %d.",
                tiTemp.currDownSpeed/1024, tiTemp.currUpSpeed/1024,
                tiTemp.avgDownSpeed/1024, tiTemp.avgUpSpeed/1024,
                tiTemp.totalDownBytes/1024.f/1024.f, tiTemp.totalUpBytes/1024.f/1024.f,
                mgr->GetSelfLayer(), mgr->GetLowVersionCount(), mgr->GetConnectFailCount(),
                mgr->GetTotalIncomingCount(), mgr->GetTotalOutgoingCount(),
                mgr->GetAvgIncomingTime(), mgr->GetAvgOutgoingTime());
            if(mgr->comm->currRes) {
                if(!bFirstTime)
                    assert(!nullLastTime);
                bFirstTime = false;
                nullLastTime = false;
                // 打印当前资源的状态
                mgr->comm->currRes->PrintStatus();
            }
            else
                nullLastTime = true;
        }

        if(currTime-lastCSReportTime >= CS_REPORT) {
            csReportCount++;

            // CSClient 发送
            lastCSReportTime = currTime;
            mgr->comm->csClient.SendLogin();
            mgr->comm->csClient.SendReport();
            // 清空增量列表
            if(mgr->comm->currRes)
                mgr->comm->currRes->ClearDiffIntervals(true);
        }

        if(currTime-lastCSNeedPeerTime >= CS_REQ_PEERS) {
            lastCSNeedPeerTime = currTime;
            mgr->comm->csClient.SendNeedPeers();
        }

        if(currTime-lastCSReqIntervalTime >= CS_REQ_INTERVAL) {
            lastCSReqIntervalTime = currTime;
            mgr->comm->csClient.SendResInterval();
        }

        if(currTime-lastClearBadAddress >= P2P_CLEAR_BADADDR) {
            lastClearBadAddress = currTime;
            // 清空无法连接的地址列表
            mgr->m_badAddr.clear();
        }

        if(currTime-lastP2PReportTime >= P2P_SEND_REPORT) {
```

www.ckook.com

```
                lastP2PReportTime = currTime;
                P2PReportCount++;

                CorePeerInfo thisPeer;
                mgr->GetSelfInfo(thisPeer);
                // 如果发现本次的 SelfPeerInfo 与上次的相同，或者本 Peer 拥有的块发生变化，则广播
                if(memcmp(&thisPeer, &mgr->lastSent, sizeof(thisPeer)) != 0 || mgr->m_bBlockInter
valHasChanged) {
                    for(it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                        if(it->IsValid())
                            it->SendReport(thisPeer, (P2PReportCount%20==0));
                    }
                    // 记录本次广播的自身信息
                    memcpy(&mgr->lastSent, &thisPeer, sizeof(thisPeer));
                }

                // 清空增量列表
                if(mgr->comm->currRes)
                    mgr->comm->currRes->ClearDiffIntervals(false);
            }
            if(currTime-lastP2PNearPeerTime >= P2P_NEAR_PEERS) {
                lastP2PNearPeerTime = currTime;
                for(it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                    if(it->IsValid())
                        it->SendNearPeers();
                }
            }
            if(mgr->comm->currRes && currTime-lastCheckPushListTime >= P2P_CHECK_PUSHLIST) {
                lastCheckPushListTime = currTime;
                for(it = mgr->m_clients.begin(); it != mgr->m_clients.end(); ++it) {
                    if(it->IsValid()) {
                        it->ReloadPushList();
                    }
                }
            }
            if (mgr->comm->localPeerFinder.IsRunning() && currTime - lastBroadcastInfoTime >=
LPF_BROADCAST) {
                lastBroadcastInfoTime = currTime;
                mgr->comm->localPeerFinder.SendBroadcast();
            }

            if(!mgr->comm->currRes) {
                mgr->StopAll();
            }
        }
    }

    // 安全关闭 Socket 的线程
    void __stdcall P2PMgr::RunCloseSocket(SOCKET* sock) {
```


www.ckook.com

```
        if(sock) {
            // 因为 TE_CloseSocket 很慢，所以需要启动一个线程
            TE_CloseSocket(*sock, FALSE);
            delete sock;
        }
    }

    // 绑定端口并开始监听
    P2P_RETURN_TYPE P2PMgr::Binding() {
        // Microsoft Knowledge Base: WSA_FLAG_OVERLAPPED Is Needed for Non-Blocking Sockets
        // http://support.microsoft.com/default.aspx?scid=kb;EN-US;179942
        m_Socket = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_
OVERLAPPED);
        if(m_Socket == INVALID_SOCKET) {
            comm->logFile.StatusErr("Creating listening socket", WSAGetLastError());
            return PRT_NET;
        }

        // SO_EXCLUSIVEADDRUSE is only supported in 2000/XP/2003 and higher OS.
        if(comm->osvi.dwMajorVersion > 5) {
            //独占端口
            BOOL bExAddrUse = TRUE;
            if(setsockopt(m_Socket, SOL_SOCKET, ((int)(~SO_REUSEADDR)), (const char*)
&bExAddrUse, sizeof(BOOL)) == SOCKET_ERROR) {
                comm->logFile.StatusErr("Setting TCP socket as SO_EXCLUSIVEADDRUSE",
WSAGetLastError());
                return PRT_NET;
            }
        }

        // 不使用 Nagle 算法
        BOOL bNoDelay = TRUE;
        if(setsockopt(m_Socket, SOL_SOCKET, TCP_NODELAY, (const char*)&bNoDelay,
sizeof(bNoDelay)) == SOCKET_ERROR) {
            comm->logFile.StatusErr("Setting TCP socket as TCP_NODELAY", WSAGetLastError());
            return PRT_NET;
        }

        // Set this socket as a Non-blocking socket.
        ULONG flag = 1;
        if (ioctlsocket(m_Socket, FIONBIO, &flag) == -1) {
            comm->logFile.StatusErr("Setting listening socket as non-blocking", WSAGetLastError());
            return PRT_NET;
        }

        // Associate the local address with m_Socket.
        sockaddr_in addr;
        memset(&addr, 0, sizeof(sockaddr_in));
        addr.sin_family = AF_INET;
```

www.docin.com

```
        addr.sin_addr.s_addr = htonl(INADDR_ANY);

        USHORT port = 0;
        for(int i = 0; i < 1000; ++i) {
            switch(i) {
            case 0:         port = 80;      break;
            case 1:         port = 21;      break;
            case 2:         port = 22;      break;
            case 3:         port = 23;      break;
            case 4:         port = 20;      break;
            default:    port = static_cast<USHORT>(rand(&comm->ctx))%10000 + 45000;     break;
            }
            addr.sin_port = htons(port);
            if(bind(m_Socket, (sockaddr*)&addr, sizeof(addr)) != SOCKET_ERROR)
                break;
            comm->logFile.StatusOut("Bind TCP socket at port %d. Failed", port);
        }
        if(i == 1000)
            return PRT_INIT_BIND;

        // Establish a socket to listen for incoming connections.
        if(listen(m_Socket, 3) == SOCKET_ERROR) {
            comm->logFile.StatusErr("Listening socket", WSAGetLastError());
            return PRT_NET;
        }

        comm->localAddress.subnetIP.sin_port = addr.sin_port;
        comm->logFile.StatusOut("Binding TCP socket at port %d.", port);

        return PRT_OK;
    }

    // 发送 TCP 数据包
    void P2PMgr::SendPackets() {
        bool noIncrease = false;
        TCPPacket* packet = NULL;
        for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
            // 如果不用继续发送上轮循环连接的数据，则把 it 降回去
            if(noIncrease) {
                --it;
                noIncrease = false;
            }

            if(!it->IsValid())
                continue;
            it->GetPacket(packet);
            if(!packet)
                continue;
            int ret = send(it->GetSocket(), packet->buf+packet->sent, packet->size-packet->sent, 0);
```

www.docin.com

www.ckook.com

```
                if(SOCKET_ERROR == ret) {
                        DWORD lastError = ::WSAGetLastError();
                        if (WSAEWOULDBLOCK == lastError) {
                                it->PutPacketBack(packet);
                        }
                        else {
                                it->SetInvalid();
                                ReleasePacket(packet);
                                comm->logFile.StatusErr("Sending data on TCP", lastError);
                        }
                }
                else {
                        AddOutgoingBytes(ret);
                        it->AddOutgoingBytes(ret);

                        assert(ret <= packet->size-packet->sent);
                        if(ret == packet->size-packet->sent) {
                                // 已经发送完毕，释放 Buffer
                                if(packet->GetMsgType() == NP2NP_RESPONSE) {
                                        AddBlockDataUp(*packet->GetBlockSize());
                                }
                                ReleasePacket(packet);
                                //此连接的一个块已经发完了，接着发送下一个
                                if(!it->SendFirstPushBlock())
                                        assert(0);
                                noIncrease = true;
                        }
                        else {// 尚未发送完毕，下次继续发送
                                packet->sent += ret;
                                it->PutPacketBack(packet);
                        }
                }
        }
}

// 判断两个 Peer 是否有建立连接的可能性
bool P2PMgr::IsConnectablePeer(const P2PAddress& connectAddr, const P2PAddress& targetAddr) {
        if(connectAddr == targetAddr) // 不用告诉别人他自己的地址
                return false;

        // 如果外网地址不同，那么……
        if(connectAddr.outerIP.sin_addr.s_addr != targetAddr.outerIP.sin_addr.s_addr) {
                if(!connectAddr.IsNAT() && targetAddr.IsNAT()) // 外网无法连接内网
                        return false;
                if(connectAddr.IsNAT() && targetAddr.IsNAT()) // 双内网无法连接
                        return false;
        }
        return true;
}
```

www.ccrn.com

```
// 取得特定类型的连接数目
UINT8 P2PMgr::ExGetCount(bool in, bool out, bool freeIn, bool freeOut, bool cponly, bool includeConnecting) {
	UINT8 count = 0;
	for(ClientIt it = m_clients.begin(); it != m_clients.end(); ++it) {
		if(!it->IsValid())
			continue;
		if(cponly) {
			count += it->GetIsSP()?1:0;
			continue;
		}
		if(it->GetIsIncoming()) {
			if(it->GetIsForFree())
				count += freeIn?1:0;
			else
				count += in?1:0;
		}
		else {
			if(it->GetIsForFree())
				count += freeOut?1:0;
			else
				count += out?1:0;
		}
	}
	if(includeConnecting) {
		for(CPeerIt it1 = m_connecting.begin(); it1 != m_connecting.end(); ++it1) {
			if(cponly) {
				count += it->GetIsSP()?1:0;
				continue;
			}
			if(it1->isIncoming) {
				if(it1->isForFree)
					count += freeIn?1:0;
				else
					count += in?1:0;
			}
			else {
				if(it1->isForFree)
					count += freeOut?1:0;
				else
					count += out?1:0;
			}
		}
	}
	return count;
}

// 检查连接数目是否已满
```

www.ccn.com

www.ckook.com

```
bool P2PMgr::IsConnectionFull(bool isIncoming, bool isForFree, bool includeConnecting) {
    if(isIncoming) {
        if(isForFree)
            return GetFreeInCount(includeConnecting) >= comm->cfgData.IN_4FREE_NUM;
        return GetIncomingCount(includeConnecting) >= comm->cfgData.INCOMING_NUM;
    }
    if(isForFree)
        return GetFreeOutCount(includeConnecting) >= comm->cfgData.OUT_4FREE_NUM;
    return GetOutgoingCount(includeConnecting) >= comm->cfgData.OUTGOING_NUM;
}
```

11.5 Myseelite 系统部署

Myseelite1 和 Myseelite2 均可以在 openmysee 的官方下载页面上下载到。本小节将具体针对 Myseelite2 的安装、使用等进行简单的介绍。

11.5.1 系统编译

Myseelite2 的编译环境是 Windows 2000（及以上版本）+Visual Studio 2003，编译前需要先下载 wxWidgets-2.8.4 和 ACE 5.5，然后实施以下的步骤。

（1）打开 wxWidgets-2.8.4\build\msw\wx.sln，生成解决方案；

（2）进入 ACE_wrappers\ace，新建 config.h，写入#include "ace/config-win32.h"，打开 ace.sln，生成解决方案；

（3）打开 myseelite2\capture\capture.sln，生成解决方案，在 myseelite2\capture\bin 下找到 liveStream.exe；

（4）打开 myseelite2\client\mediaplayer\mediaplayer.sln，添加包含目录："\wxWidgets-2.8.4\include\msvc"、"\wxWidgets-2.8.4\include\"，添加附加库目录"\wxWidgets -2.8.4\lib\vc_lib"，生成解决方案，在 myseelite2\client\bin\debug 下找到 playerd.exe；

（5）打开 myseelite2\server\superpeer\superpeer.sln，添加包含目录："\ACE_wrappers\"，添加附加库目录"\ACE_wrappers\lib"，生成解决方案，在 myseelite2\server\superpeer\Debug 下找到 superpeer.exe；

（6）打开 myseelite2\server\tracker\tracker.sln，添加包含目录："ACE_wrappers\"，添加附加库目录"ACE_wrappers\lib"，生成解决方案，在 myseelite2\server\tracker\Debug 下找到 tracker.exe；

（7）复制 ACE_wrappers\lib\ACEd.dll 到 C:\windows\system32 下。

11.5.2 系统运行

Myseelite2 中包括 4 个子系统：Tracker、Super Peer、Capture、Client，它们的运行方法如表 11-5 所示。

www.docin.com

www.ckook.com

表 11-5　　Myseelite2 子系统运行说明

子系统名	运行描述
Tracker	① 将 tracker.cfg 复制到 Debug 或者 Release 目录下； ② 运行 tracker.exe
Super Peer	① 用文本编辑器打开 superpeer.cfg，确认 P2P 端口和 CS 端口没有被占用，确认 Windows 数据路径和 XML 路径存在，确认 tracker 的地址正确，确认 Super Peer 和 Nomal Peer 的端口号正确； ② 将 superpeer.cfg 复制到 Debug 或者 Release 目录下； ③ 运行 superpeer.exe
Capture	① 运行 LiveStream； ② 在媒体源地址一栏输入已知的 MMS 流或者 RTSP 流的地址，可以利用 Windows Media Encoder 或者 Windows Media Service 启动一个 MMS 流，也可以利用 Helix Server 启动一个 RTSP 流，最好是直播的流，因为目前 Capture 对于时断时续的流还处理不好； ③ 在 SP 地址一栏输入 Super Peer 的 IP，在 SP 端口一栏输入 superpeer.cfg 中的 PortForCS 的值； ④ 在 ChannelName 一栏任意输入一个字符串，为了测试方便可以每次输入固定的字符，比如“test”； ⑤ 用户 ID 和用户密码两栏可以跳过不动； ⑥ 单击“开始上传”按钮； ⑦ 到 superpeer 的 DataPath_Windows 或者 DataPath_NonWin 目录查看，如果频道建立成功，则会出现一个目录，目录名是频道的散列值，如频道名为“test”，则对应的目录名是“d8e8fca2dc0f896fd7cb4cb0031ba249”
Client	① 到 superpeer 的 DataPath_Windows 或者 DataPath_NonWin 目录查看，使用文本编辑器打开此目录下的 channel.xml 或者 channel.txt，将目标频道的 URL 一栏复制出来； ② 运行 Windows Media Player； ③ 从菜单选择 Open URL，粘贴上面复制的 URL，等待 Loading 状态过后，即可开始播放

11.5.3　轮播流程

Myseelite 系统可以为用户提供了轮播节目的解决方案，它的具体节目发布流程如下所示。

（1）寻找片源（.avi、.wmv、.rmvb、DVD 等格式均可）。

（2）格式重新编码。

把片源用视频 VP70、音频 GSM6.10 转换成标准统一的.avi 格式。

如果需要把多个节目合并为一个频道，需要用到（WinAVIVideoConverter）软件对片源进行处理。具体转换流程如图 11-16、图 11-17、图 11-18 和图 11-19 所示。

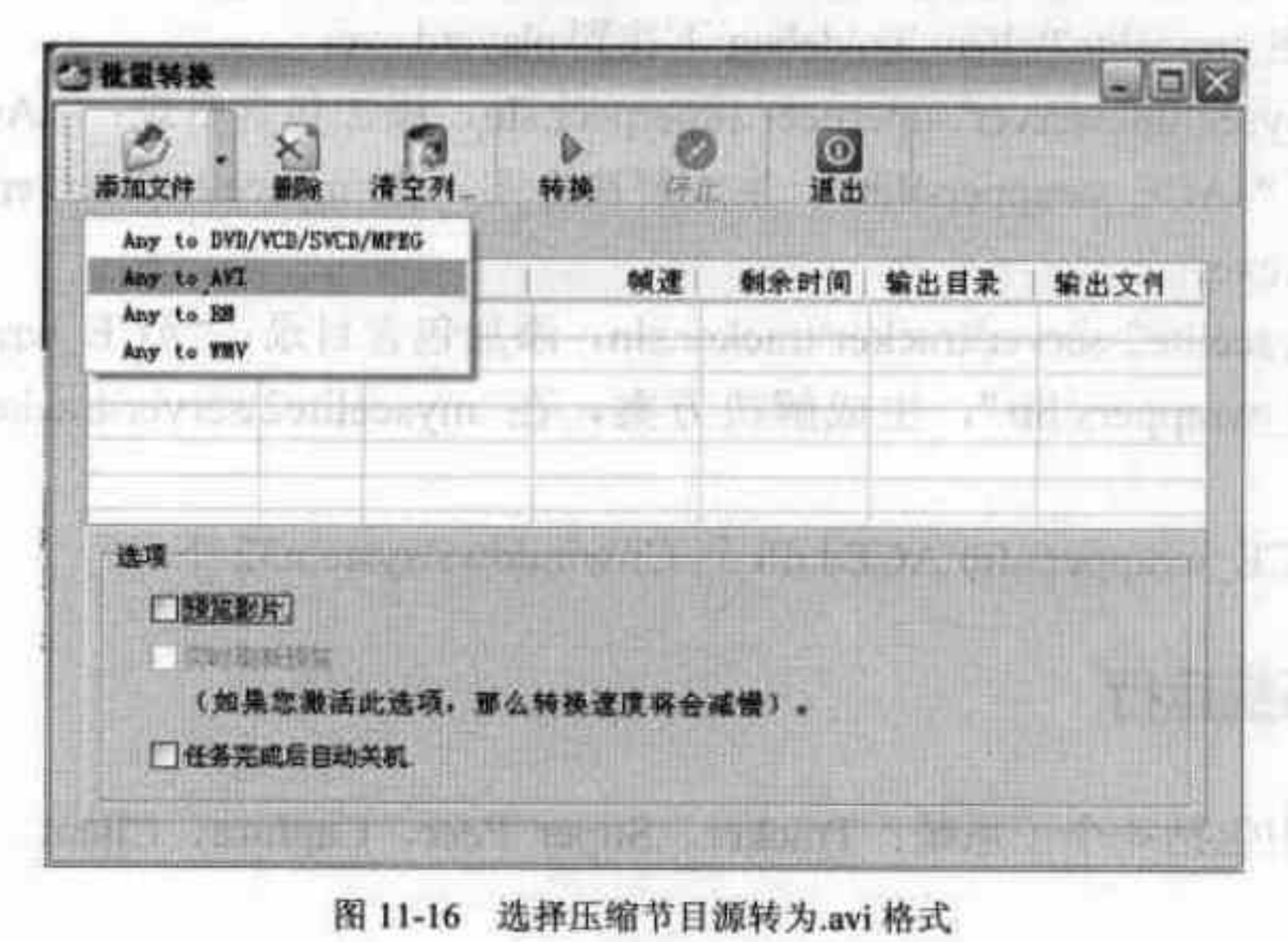


图 11-16　选择压缩节目源转为.avi 格式

www.ckook.com

图 11-17 选择.avi 源文件存放路径

图 11-18 输出文件保存地址

图 11-19 大多数文件压缩的配置数据

www.docin.com

（3）转换为 ZZL 文件。

用华清峰凌的压缩软件（MD5 码批量转换软件：ZZLConvertSetup.exe）把.avi 格式片源进行重新编码，转换为 ZZL 文件，如图 11-20 所示。

编码完成后，会生成一个文件夹。文件夹里包含一个 ZZL 文件和一个 MD5 加密过的文件名。单击打开此文件夹，会发现里面有 3 个文件（0、config、keysample）。其中，ZZL 文件的主要功能是用来预览转换后的效果的，无需上传到服务器，可以用 WMP 观看。查看 MD5 文件夹中的“0”文件大小是否与转换前.avi 格式文件大小一致。如果 ZZL 能够预览并且“0”文件与 AVI 文件大小一致，说明转换成功。

图 11-20　添加.avi 格式文件设置保存路径

（4）上传到 Source Peer（SP）源服务器。

把 MD5 码文件名的文件夹（无需上传 ZZL 文件）上传到 Source Peer 服务器的/data/sp/pro gram 文件夹里面。在 data/sp/playlist/ 里面新建一个文件，文件名就是上传上去的 MD5 文件名。建立方法：vi [MD5 文件名]，在第 1 行写 40，第 2 行写视频文件的拼音名字，第 3 行写视频文件的 MD5 名字。保存后重启 SP 服务器程序。方法：执行 kill –HUP * 命令重启此配置，gtv 地址就生成了。经过了寻找片源、格式重新编码、上传服务器、Web 频道发布几个流程就可以把频道发布到网站上了。该流程图 11-21、图 11-22 和图 11-23 所示。

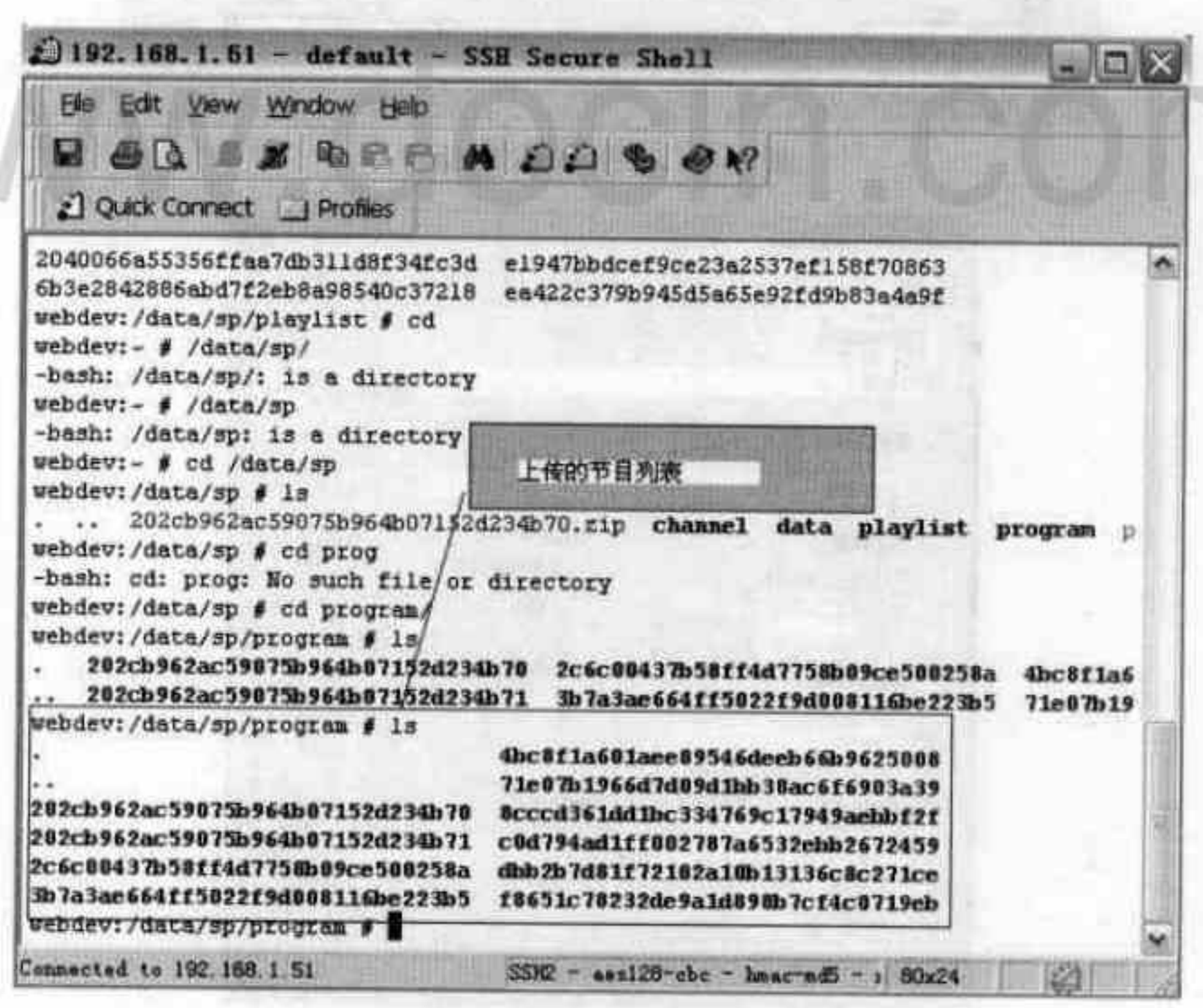


图 11-21　上传到源服务器

www.ckook.com

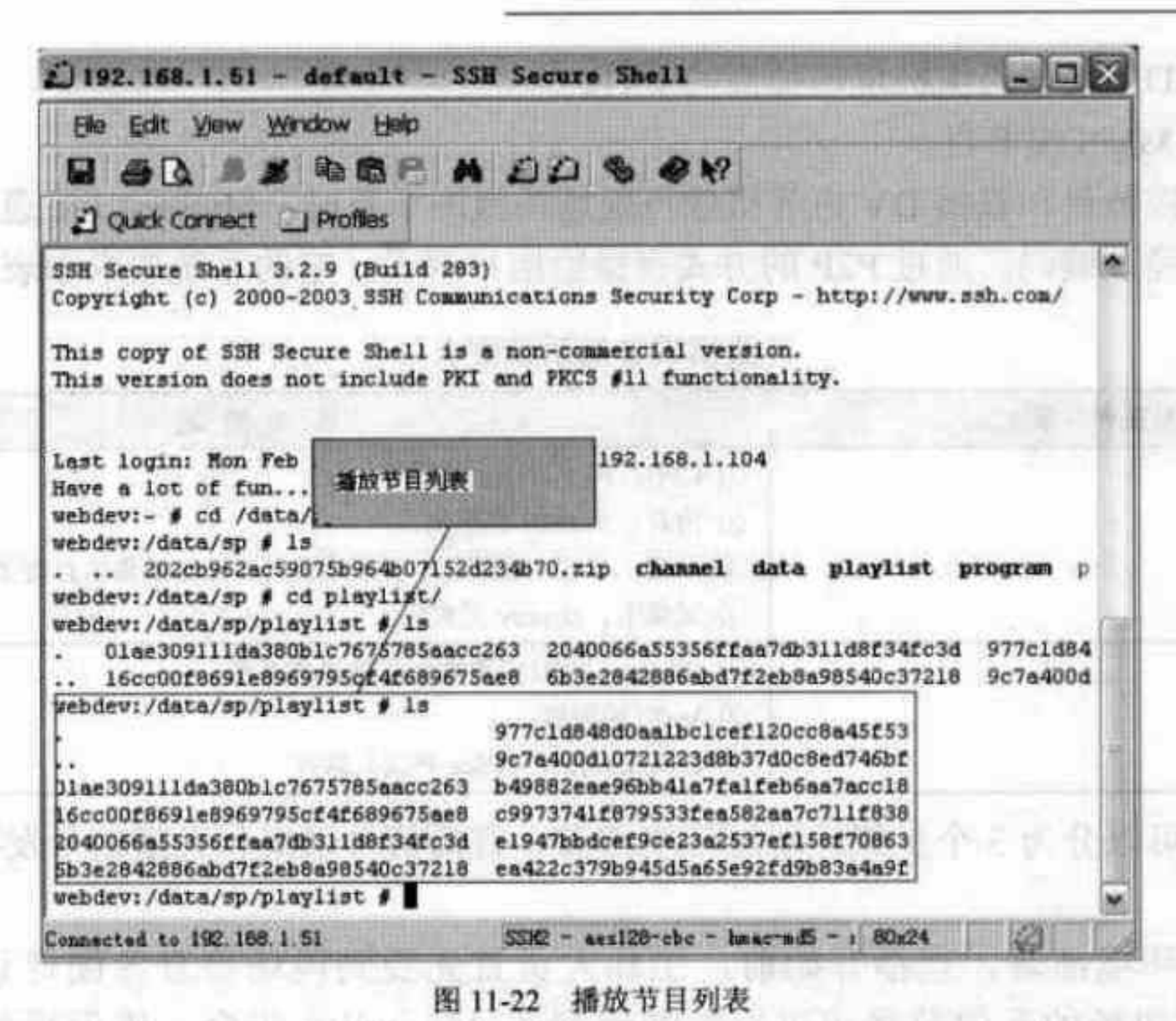


图 11-22 播放节目列表

图 11-23 playlist（md5）文件内容

11.5.4 直播流程

MySeelite 系统可以为用户提供多种直播解决方案，方便用户通过各种方式进行节目直播，包括：

（1）直播电视节目；

www.docin.com

（2）直播 DV 机等摄像机现场拍摄的节目；

（3）直播 MMS 流节目。

在直播电视节目和直播 DV 机等摄像机现场拍摄的节目时，MySee Live 通过采集卡，把不同的信号源经过编码，通过 P2P 的方式直播给用户观看。它的系统要求如表 11-6 所示。

表 11-6　　直播设备的系统要求

系统要求项	具体描述
硬件要求	① CPU：P4 2.4GHz 或更高； ② 内存：512MB 或更高； ③ 硬盘：40GB 或更高，根据是否需要保存采集信息而定； ④ 采集卡：Osprey 采集卡
软件要求	① Window 2000 或更高版本的操作系统； ② Vp70 编码器； ③ GVCapture（MySee 开发）软件

现场直播可以分为 3 个步骤：现场环境部署、直播信号采集和直播信号发布。详细步骤如下所示。

（1）现场环境部署。直播开始前，工作人员首先要到现场检查各硬件设备与软件程序，确保演播现场的音像数据流可以顺畅地到达 Myseelite 平台，确保摄像设备到采集机的连接正常，采集机工作正常，采集机的网络连接（到 Myseelite 平台的网络连接）正常。

（2）直播信号采集。由摄像设备在演播现场进行录制，并实时将录制信号通过采集卡，发送到采集机上。在采集机上运行 MySee 软件 GVCapture，由该软件把采集到的信号编码后发送到 Myseelite 系统网络平台的源服务器。采集过程应注意信号的稳定性与网络的稳定性。

（3）直播信号发布。Myseelite 系统网络平台的源服务器在收到采集现场通过 GVCapture 发过来的信号后，自动形成一个频道种子，并将此种子发布到直播网站上，普通用户通过浏览直播网站，收看直播节目。

www.docin.com

在直播 MMS 流节目时，它的系统要求如表 11-7 所示。

表 11-7　　直播设备的系统要求

系统要求项	具体描述
硬件要求	① CPU：P4 2.4GHz 或更高； ② 内存：512MB 或更高； ③ 硬盘：40GB 或更高
软件要求	① Media Services（Windows 2003 Server）自带； ② MMSServer（MySee 开发）软件

MMS 流节目的直播流程如下。

（1）Media Services 节目广播。直播开始前，准备好直播的内容，注意，不能音频视频混合编排 Media Services 的节目单，不能加图片 Logo。

（2）MMSServer 流转发。把 MMS 流的 URL 填写到 MMSServer 中，发送到 Myseelite 的源服务器。注意，保证 MMS 流到安装了 MMSServer 机器之间的带宽；Media Services 与 MMSserver 可以安装在同一台机器上；保证 MMSServer 到 Open MySee lite 源服务器之间有足够的带宽。

www.ckook.com

11.6 Myseelite 系统分析

Myseelite 中采用了 CDN 的多播方案，部署了两类服务器：Super Peer 和 Tracker。Client 获取数据依赖于 Super Peer 的缓存服务器，并且需要从 Tracker 中获取 Super Peer 和其他 Client 的地址。可以看出，Myseelite 网络是一个有中心节点的 P2P 网络。在这种网络中，为了减低系统的复杂度，增加了服务器的部署和管理成本，简化了客户端的功能，并不是一个纯粹的 P2P 网络。但是，服务器很容易变成整个系统的瓶颈，存在中心节点效应，并且系统的可扩展性受到一定的制约。

可以考虑将 Myseelite 中 Tracker 服务器的功能向客户端转移，使客户端能够自组织成为一个 P2P 网络，使用第 3 章中描述的结构化或非结构化的 P2P 路由算法进行路由。这样，就避免了当某个节点失效时，客户端无法连接到系统的情况。同时也减少了 Tracker 服务器的部署和管理成本。

11.7 本章总结

本章概括性地介绍了 P2P 视频点播系统产生的背景，简要地描述了 Myseelite 系统的设计目标、基本概念、工作机制以及在该系统中用到的两个跨平台库，ACE 和 wxWidgets。本章详细地介绍了 Myseelite 的 4 个子系统——Capture、Super Peer、Tracker 和 Client 的设计，针对这 4 个子系统的功能和结构着重进行了分析。此外，本章还针对 Myseelite 的几个小的模块进行了源代码分析和举例，对 Myseelite 系统的部署进行了简单的描述。

本章力图为读者全面地了解 Myseelite 系统提供一定的基础知识。希望读者能够通过本章的介绍对 Myseelite 系统的工作原理、系统结构和网络架构等方面有一个初步的认识。有能力的读者可以在阅读本章的基础上，对 Myseelite 的源代码进行更加深入的剖析，以求取得更多的收获。

11.8 练习题

（1）请详细描述 Myseelite 系统中客户端的登录流程。
（2）请列举出 Myseelite 系统中的子系统，并且对每个子系统的功能进行阐述。
（3）请简要描述 Myseelite 系统中 Capture 子系统的结构。
（4）请简要描述 Myseelite 系统中 Super Peer 子系统的结构。
（5）请简要描述 Myseelite 系统中 Tracker 子系统的结构。
（6）请简要描述 Myseelite 系统中 Client 子系统的结构。
（7）请简要描述 Myseelite 系统的轮播流程。
（8）请简要描述 Myseelite 系统的直播流程。

www.ckook.com

参 考 文 献

[1] Schmidt D C. The Adaptive Communicaton Environment: An Object-oriented Network Programming Toolkit for Developing Communication Software[C]. Proc. of the 12th Annual Sun Users Group Conf. San Francisco, CA, 1994: 214-225

[2] wxWidgets 2.6.4: A portable C++ and Python GUI toolkit. Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al. March, 2007. http://www.wxwidgets.org/manuals/2.6/wx_contents.html

[3] 使用 wxWidgets 编程——第一步. http://blog.csdn.net/snaill/archive/2007/06/29/ 1671333.aspx

[4] myseelite_2007_06_28.zip. http://www.openmysee.com/modules/article/view.article.php/ 33

[5] myseelite2.alpha.rar. http://www.openmysee.com/modules/newbb/viewtopic.php?topic_id=118&forum=11

[6] P2P TV，超越 IPTV——浅析 P2P TV 市场发展的现状与挑战. http://www.P2Psky.com/tech/ article2480.html

[7] myseelite2 安装说明. http://www.openmysee.com/modules/newbb/viewtopic.php?topic_id=123&forum=11

www.docin.com

www.ckook.com